Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Performance Evaluation of Smart Contracts

Raul Alexandre Vaz Oliveira

Orientador(es) | Salvador Abreu

Évora 2022

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Performance Evaluation of Smart Contracts

Raul Alexandre Vaz Oliveira

Orientador(es) | Salvador Abreu

Évora 2022

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente | Lígia Maria Ferreira (Universidade de Évora)

Vogais | Salvador Abreu (Universidade de Évora) (Orientador)
Simão Melo de Sousa (Universidade da Beira Interior) (Arguente)

Évora 2022

To my family

Acknowledgments

First of all, I would like to express my gratitude to Salvador Abreu, my dissertation's advisor, for his continued support, revision and analysis of the different phases which this paper went through, in addition to, the patience that was needed to deal with every inquiry that I had about anything related to what was being done. Without his guidance and assistance, this dissertation would not be possible.

I would also like to thank my family for their constant support throughout the whole progress of, not only this dissertation, but also, my entire academic journey.

And lastly, I would like to give a special thanks to all of my friends and colleagues, who either purposely or unintentionally, transferred some of their knowledge onto me, who helped improve my skills, become better at researching and understanding different topics, and also overall, to aiding me make the person that I am today.

Contents

List of Figures

List of Tables

Acronyms

ABI	Application Binary Interface
API	Application Programming Interface
DApps	Decentralized Applications
DeFi	Decentralized Finance
DSL	Domain-Specific Language
EVM	Ethereum Virtual Machine
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LPoS	Liquid Proof-of-Stake
NFTs	Non-Fungible Tokens
P2P	Peer-to-Peer
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
ROM	Read-only memory
RPC	Remote Procedure Call

Abstract

Blockchain is a distributed ledger with records of transactions made between accounts stored in it through blocks. This type of technology is becoming a part of our daily lives, due to its secure, immutable, verifiable and transparent nature.

One relevant aspect of blockchain is that of *smart contracts*, these are very important and useful because they can automatically carry out certain actions based on what was written in its code. Transactions which should be carried out under specific conditions can be done more effectively by using a smart contract, which explicitly incorporates those conditions in its code.

Since smart contracts are so critical, it is very important to make the correct decision when choosing a high-level language to use to code these agreements. Different languages that essentially do the same thing, can compile to a completely different set of instructions.

The objective of this dissertation is to compare the performance of different smart contract high-level languages, set to carry out the same goal. We will compare languages from the Tezos and Ethereum blockchains. Different languages will produce different low-level code, some of which will be more efficient, as they will require less resources such as gas.

Keywords: Smart Contracts, Gas, Tezos, Ethereum, Blockchain, Performance, Evaluation, Comparison

Sumário

Avaliação do Desempenho de Smart Contracts

Blockchain é um registo distribuído com documentações das transações feitas entre contas guardadas nele através de blocos. É bastante provável que este tipo de tecnologia faça parte do nosso dia-a-dia, devido à sua segura, imutável e transparente natureza.

Um aspecto importante da blockchain são *contractos inteligentes*, estes são muito importantes e úteis por serem capazes de executar automaticamente certas ações baseadas no que foi escrito no seu código. Algumas transações com condições particulares podem ser feitas mais rapidamente se um contrato inteligente é utilizado e essas condições são escritas na sua lógica.

Devido aos contratos inteligentes serem tão valiosos, é muito importante fazer a decisão correta na escolha de uma linguagem de alto nível para utilizar para programar estes acordos. Diferentes linguagens que essencialmente fazem o mesmo, podem compilar para um conjunto de instruções completamente diferente.

O objectivo desta dissertação é comparar o desempenho de diferentes linguagens de alto nível de contratos inteligentes escritos para fazerem a mesma coisa. Vão haver comparações entre as linguagens das blockchains Tezos e Ethereum. Linguagens diferentes irão produzir diferentes códigos de baixo nível, considerando algumas destas linguagens mais eficientes, devido a requererem menos recursos como gas.

Palavras chave: Contratos Inteligentes, Gas, Tezos, Ethereum, Blockchain, Desempenho, Avaliação, Comparação

1

Introduction

Blockchain is a relatively new technology which has being growing more and more recently, and may very well be a part of our daily lives in the near future. Because it's a distributed ledger, all of its users can access every single transaction made, by anyone, to anyone, but can't modify any of them because they are immutable, making this technology extremely secure. Blockchains are now recognized as the "fifth evolution" of computing, the missing trust layer for the Internet because they are able to provide trust in digital data. When information has been written into a blockchain database, it's nearly impossible to remove or change it. This capability has never existed before [Lau17].

Another important aspect of this technology is smart contracts. These are a set of instructions recorded on the blockchain that execute actions when previously set conditions have been met and confirmed. These programs are used to speed up transactions and are created through a trusted agreement between two parties. These contracts are very useful, and because of this, if someone wishes to create one, it is really important to be careful when choosing a smart contract language.

This dissertation will focus on the performance of equivalent programs in a variety of different smart contract languages on two blockchains, Tezos and Ethereum. Programs that execute and perform the same actions, in different languages, will compile to a distinct set of instructions, which will be evaluated and then compared to each another.

1.1 Motivation

The use of smart contracts has increased to the point of being very useful and important. There have been many applications created with the need to not be controlled by a single entity, in other words, the need to make it decentralized.

So, Decentralized Applications (DApps) were invented, these are applications created on top of a blockchain, in a Peer-to-Peer (P2P) network. They represent one of the latest advancements in decentralization technology [Bas20]. The way these applications work is by having their backend code, which is written into smart contracts, running in a decentralized environment, like a blockchain. They usually consist of one or more smart contracts in conjunction with a user interface, meant to increase transparency around commercial transactions, governmental processes, supply chains, and all those systems that currently require mutual trust between user and provider [Inf19]. Once they are deployed into a blockchain network, they cannot be tampered with, they can only be interacted with using what was already previously written into its logic.

There are many areas where DApps are able to show their true potential, examples of this are: Decentralized Finance (DeFi), creating new ways to borrow, lend, or invest money while disregarding the need of a company or bank to hold the money; Supply chain management, reducing the production of counterfeit goods, decreasing fraud, and ensuring a more efficient distribution system; and healthcare, making the patients records more protected and controllable by the data owners, and improving healthcare models, such as implementing new technologies to better the whole sector.

This is why it's important to be careful when choosing a language to create the contracts which control the operations of these applications. It's a big reason for the motivation of this paper, because with the stats of the performance of different languages in two distinct blockchains which have a large engagement in smart contract development, a developer can have a better idea in choosing a language more appropriate to what they plan to create into their decentralized application.

1.2 Objectives

As already mentioned, the goal with this dissertation is the understanding of different high-level smart contract languages available on Tezos and Ethereum, and then the execution of programs that perform the same actions in these languages and compare their performances with each other. At the end, with the values gathered, tables and graphs will be made to better understand the data assembled, and conclusions will be reached about the languages used.

In order to reach the main goal, it is necessary to research information on each of the blockchains, the nodes that they have available, the respective networks that are run in the nodes, the smart contract languages and their documentations, environments and compilers that each ledger will have accessible for its users, as well as the creation and funding of accounts in order to be able to create and record a smart contract in the blockchain and also send transactions to the smart contract address to perform different actions based on the arguments sent. After all this, create relatively simple programs that do the same thing in distinct languages, migrate them into the blockchain, call these programs with different argument numbers and use the gas values originated from each transaction to represent the performance. Gather all these values and for each different argument, compare the gas values given by each language and reach conclusions based on what has been evaluated.

1.3 Structure

This dissertation is structured as follows: in chapter 2, we present a revision of current relevant state of the art, speaking of the concepts important to this paper, as well as, some related work done by third parties. Chapter 3 describes every single tool, framework and environment utilized in the progress to address and reach the objectives stated. In chapter 4, every parametric problem chosen to be written into smart contract code, will be described, in addition to, analyzing in table form, the performance of every language used in each of the problems. And in chapter 5, the values shown in the previous chapter, will be displayed here in a more suitable form, through the use of graphs, for a more easy understanding of the data calculated. Lastly, chapter 6, concludes this dissertation with an overview of the most important aspects of the entire paper, and also, a bit of information is given into what could potentially be carried out as future work of this dissertation.

2

State of the Art

In this chapter we discuss the state of the art. The concept of blockchain and other specific aspects of it will be explained, as this is the focus of everything that is mentioned and demonstrated in this work, while also identifying the two specific blockchains used in this dissertation in conjunction with a brief look at how each of them operates. We shall also explain what smart contracts are, how are these agreements useful and the type of languages in which these smart contracts can be coded in. The concept of gas within the blockchain will also be introduced and its relevance to this work will be described.

2.1 Blockchain

Before going into specifics about each of the two blockchains chosen and how they work, it is important to first explain what blockchain technology consists of. A blockchain is a new type of shared database which is able to store records and transactions between accounts. A big difference between this database and the already existing ones that store people's assets of value, like banks for example, is the trust involved. People trust that banks won't steal their money as the government regulates them. If a bank fails, people trust that the government will ensure their deposits of money are safe [Gat17]. This trust doesn't only occur in financial situations, it also applies to companies that have your personal information, like your home address. We mindlessly trust that these entities that can see and manage this information won't reveal or share it with others. These databases are centralized networks, unlike the blockchain which is a decentralized one, where everyone can see and validate transactions, it also removes the need of an intermediary entity, creating transparency and trust. The blockchain allows people to transact directly between each other with anything of value, this can be used for property, shares, money, digital files... almost anything [Gat17].

As the name indicates, a blockchain is formed by blocks chained together. These blocks are where the transactions are recorded, and when a block is considered full, all of its contents can be validated by being broadcast by one or several nodes. These nodes share the same ledger across different systems, distributed around the world. When a user performs a transaction, their client turns it into an hash code of a fixed size and injects it into the node which they are making use of, after this it waits for the entire block to be validated for it to at last inform the user that the operation was successfully included. If there is a mistake of some sort with the code or the execution exceeds the gas limit, it doesn't even transform the transaction into an hash code, the client gives the user an error message instead. And when a block is validated, an hash code is formed with all of its transactions with the addition of the previous block's hash code, making the blocks linked. In figure 2.1, a simplified version of how the blocks in a blockchain are formed and connected can be seen.

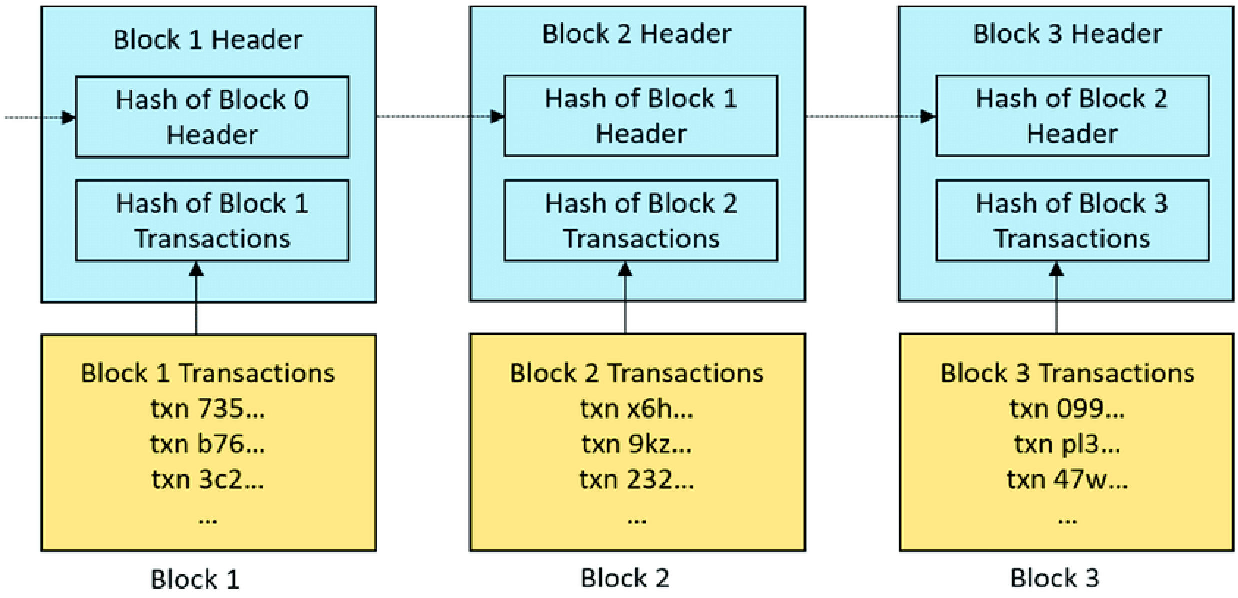


Figure 2.1: Simple example of how blocks are chained (from [AME19])

This chain that exists between the blocks makes each block more secure the more blocks there are after it. Since each block contains the hash from the previous one, if a hacker were to attempt to change any piece of information contained in a block, he would have to change every single block after it, but even in this scenario, the existence of the other nodes is proof of the integrity of the data, making it very easy to detect that this version of the blockchain would not match with every other node in the network.

In order to perform any transaction, each account has to have a certain quantity of the cryptocurrency that the blockchain provides. A cryptocurrency is a digital token that has a market value and are often traded on exchanges like stocks [Lau17]. This cryptocurrency has real value if utilized on the main network of a blockchain, and in order to obtain some, a person would have to either buy it or get it transferred from a different account into theirs. There are also networks considered test networks. These networks are mostly used by developers for any testing of any kind. In this situation, obtaining funds is much simpler because test networks are accompanied with a faucet, where someone can basically obtain as much as they would like. However, some faucets either have a limit, within a certain time, per user, or a recommendation not to abuse it. In these networks, the currency doesn't have any real value outside of them and it only allows a user to make transactions in the respective test network.

Each network has a protocol that defines how it is operated. This protocol states the variables of how it works. It can describe things such as the size of a block, the number of blocks per cycle, improvements in the execution language of smart contracts, optimizations to the gas utilized, the maximum time an operation can wait until it gets approved or rejected, etc.

If a transaction is considered to be a valid transaction, it will be inserted into the block. Then, this block must associate with the chain. All nodes must be aware of any updates occurring in the network, any node that has a new block must inform all other nodes to update their local chains [AA19]. Although, making use of this method would cause uncertainty if different nodes try to broadcast a new block at the same time. So, in order to avoid this issue, a consensus algorithm is utilized in the blockchain.

Different blockchains and/or networks possess distinct ways to validate their transactions, and they all consist in the usage of a certain consensus mechanism, but there are two consensus mechanisms which are the most prevalent of them all. These two are called Proof-of-Work (PoW) and Proof-of-Stake (PoS).

In PoW, the members that wish to validate a block, have to solve a puzzle before doing so. The puzzle they solve is known as the proof of work. It is a mathematical puzzle that is very difficult to solve but easy to verify the answer once it has been solved [Gat17]. The ones who attempt to solve these puzzles are called miners, and if successful will receive a certain amount of the respective cryptocurrency that the blockchain provides for contributing their own power and electricity. An example of this could be, imagine there is a rule that states that the hash code of an entire block must start with four zeros. The puzzle here would be to go through every number to find one that when matched together with the rest of the block would give an hash that would fit the rule. The first miner to find this specific number would be rewarded, the number found would be easily confirmed by the other nodes that it works, and this block would be added to the chain of blocks.

In PoS, the nodes chosen to validate a block, are chosen based on the amount of cryptocurrency that the miner holds. This is an alternative consensus protocol that retains the advantages of PoW while overcoming some of its weaknesses [OTJA21]. The bigger the amount someone has, the higher chance of them being chosen, in other words, the more cryptocurrency the miner owns, the higher the mining power he has. The main difference between this method and the PoW one, is the massive reduction in the cost of energy, therefore being more environmentally friendly. This may make it more attractive for individuals to run nodes in the network, which would increase decentralization and increase security [Lau17].

2.2 Smart Contracts

The arrival of Bitcoin in 2008 marks the birth of cryptocurrencies where the core technology that enables them is the blockchain. And in 2014, Ethereum extended Bitcoin and introduced the smart contract into the blockchain. This event greatly improved the ability to create and develop applications in the blockchain, and it also made Ethereum one of the biggest motivators of blockchain technology, because it made applications with smart contracts slowly become prevalent [HZL+21]. But the actual concept of smart contracts is even older. The idea of them was introduced in the early 1990's by computer scientist, lawyer and cryptographer Nick Szabo, who in one of his papers, referred to them as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises" [Sch18].

Smart contracts are programs stored in the blockchain with the intention of being automatically executed when a certain condition is met. These programs are living on the chain, where the users can call them with whatever arguments the contract allows and execute them. This is useful with agreements between multiple parties, they are written into a smart contract so that everyone knows exactly what action would perform automatically when a certain event is triggered.

The oldest piece of technology which most similarly resembles a smart contract is the vending machine. Vending machines are defined as self-contained automatic machines that dispense goods when a form of payment is made, they are programmed with certain rules that could be arranged in a contract, and perform such rules [Sav16]. This would mean that, when wanting to operate a certain contract, the user would have to send a fixed amount of money to run it, but not every function is obligated to be paid in order to execute. Smart contracts function in the blockchain much like a normal account, they have an address and also a balance, with the difference being the code written into it. Some functions of a contract may require a certain amount of cryptocurrency to allow them to execute, others are free, and can be used as much as anyone would want with no charge.

These contracts have a state, or storage, which can change based on what is possible to achieve in the code. A smart contract can't actually perform many functions against the blockchain since it is immutable. It can, however, add or read the data of its own storage, but updating data is really an add function that changes the current state [ZWM21].

In listing 2.1, a simple contract written in the Solidity language (see 3.4.5), can be seen. In this example, a user can either deposit ether into this contract's balance, or withdraw ether that was previously sent by this user. In the beginning, a variable called “balances” is set, saving a map with the addresses of users as the key and an unsigned integer representing the user's balance as its associated value, this variable corresponds to the storage of this contract. Whenever a user calls the deposit function, his respective balance is increased in the state and decreased in his own account, and when the withdraw function is used, the opposite of this occurs. In fact, the keyword “payable” on the function deposit, makes it mandatory to send some amount of ether to execute this particular task, making that value sent, the amount to deposit into the respective balance in the storage.

```
pragma solidity ^0.5.0

contract Bank {
    mapping (address => uint256) public balances;
    function deposit() external payable {
        require(balances[msg.sender] + msg.value >= balances[msg.sender]);
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint256 amount) external {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        msg.sender.transfer(amount);
    }
}
```

Listing 2.1: Simple smart contract example

2.3 Languages

There are two different types of languages in which smart contracts can be displayed in, high-level and execution. High-level languages are the ones that are more developer friendly and easier to write a contract in. Execution languages are the ones that the blockchain itself accepts so that it can actually run the contracts.

2.3.1 High-level

The high-level languages are constructed with a blockchain target in mind. The language must be able to be compiled down to the respective execution level language of the blockchain. In the case of the Tezos blockchain (see 2.4), the ones tested in this paper are:

- SmartPy
- Liquidity
- PascaLIGO
- CameLIGO
- ReasonLIGO
- Archetype

And in the case of the Ethereum blockchain (see 2.5), the ones tested in this paper are:

- Solidity
- Vyper

2.3.2 Execution

The execution languages are the ones that the blockchain itself uses in order to run the smart contracts. In the Tezos blockchain (see 2.4), the six languages mentioned will be compiled to the low-level language called Michelson, which is a stack-based language. And in the Ethereum blockchain (see 2.5), the two languages mentioned will be compiled to Ethereum Virtual Machine (EVM) bytecode. Developers can, if they prefer, write the contracts directly in Michelson, in the case of the Tezos blockchain, because it is composed of instructions and operations that can be understood, although much harder than the high-level languages. In the case of the Ethereum blockchain, developers are required to use the high-level languages due to the bytecode not being human-readable.

2.4 Tezos

Tezos is an open-source decentralized platform for assets and applications that has the ability to evolve by upgrading itself. This blockchain is an innovative one, because it improves multiple aspects of already existing ones. It focuses on formal methods to improve safety and possesses the capacity to amend its own protocol, the one that validates blocks and implements the consensus algorithm, through a voting mechanism [ABT19]. Its own cryptocurrency is called tez, represented with the code XTZ.

The current Tezos protocol is based on a Liquid Proof-of-Stake (LPoS) consensus algorithm, which considers the amount of tokens someone holds as the main resource to make the pool of block producers (in this case called bakers) [ABT19].

The selection of bakers from the baker pool is randomly made using a list of slots, this list has all the rolls of every baker that has at least the minimum amount of tokens required to participate. An example of this could be, let's say the minimum amount of tokens is 10.000, a baker with 45.000 tokens, has 4 slots on the list and therefore 4 rolls in the selection. The more rolls someone has, the bigger chance they have to be selected. In addition to this, participants that do not have enough tokens or simply do not wish to bake blocks, have the ability to delegate their tokens to another baker, potentially giving this baker more slots, this is where the "Liquid" part of LPoS comes into play because much like in Liquid Democracy, one can delegate its right to vote [ABT19].

Self-amendment means that Tezos can upgrade itself without the need of a hard fork, in other words, without the need to divide the blockchain into two separate branches. Because of this, the coordination and execution costs for future upgrades are reduced and improved protocols can be not only faster but also more easily carried out.

In order to upgrade itself, the voting system functions in four different periods. First, there's the proposal period, where any baker can submit a proposal, bakers can upvote proposals submitted by others and the most upvoted one advances to the next period; Second, the bakers vote to determine if the proposal selected should be tested, if the quorum (percentage needed for approval) is met, it moves on to the testing period; Third, a new test network is created to experiment with this proposal and during this period anyone can evaluate and test it out; and fourth, the bakers vote once again to decide if they wish to promote the protocol from the test network into the main network, and this also requires the quorum to be achieved, in order to be approved [Goo14].

This blockchain is considered reliable when it comes to creating and publishing smart contracts, not only because of the number of high-level languages available, but also due to the fact that its accepted Domain-Specific Language (DSL), Michelson, was explicitly designed to facilitate the readability and verifiability of contracts while being low level enough to fulfill the performance predictability requirement of on-chain execution [ABT19]. When dealing with the cryptocurrency in Michelson code, the data type used is called mutez, which is an alternative denomination of tez, 1 XTZ is equal to 10^6 mutez.

In listing 2.2, a simple contract displayed in Michelson can be seen. In Michelson, since it is a stack-based language, the code part will consistently start off with a pair of the parameter and storage in the stack, and it always ends with a pair of "NIL operation" and the updated value of the storage. This contract takes an integer value as parameter and holds another integer as its storage. It duplicates the initial pair, and chooses the first member of the 1st pair, and the second member of the 2nd pair, basically having the parameter on top of the current storage value. It then adds them together and saves the new value in the storage, overriding the old one.

```
parameter int;
storage int;
code {  DUP ;
        CAR ;
        DIP {CDR};
        ADD;
        NIL operation ;
        PAIR
     }
```

Listing 2.2: Simple michelson contract example

Tezos was initially proposed in a whitepaper published in 2014 [Goo14], its test network was deployed in June 2018, and the main network was launched in September 2018. And as of November 2021, there have been 7 approved protocols tested, each with their initial corresponding to a letter of the alphabet, the current approved one is called the Granada protocol, and there's already an 8th one called Hangzhou being tested in its respective test network. In the context of this paper, all of the smart contracts created on the Tezos blockchain were published and tested on Florencenet (see 3.5.1), the test network running Florence, the 6th protocol approved.

2.5 Ethereum

Ethereum is an open-source decentralized platform with the ability to allow applications and smart contracts to run on it. Ethereum provides a blockchain with a built-in Turing complete programming language that can be used to create systems for a number of different purposes, such as decentralized finance applications. Another thing this platform permits is Non-Fungible Tokens (NFTs), these tokens represent digital art or other items such as images, videos, audio, etc, and can be sold digitally as unique property from user to user. Ethereum's cryptocurrency is called ether, which is represented with the code ETH. In terms of market capitalization, Ether is in second while Bitcoin is first.

Ether has many different denominations, some of the more used ones, in conjunction with their use and value, in both wei and ether, are represented in table 2.1.

Name	**Value (Wei)**	**Usage**	**Value (ETH)**
wei	1	Transaction fees	10^{-18}
gwei	10^{9}	Gas prices	10^{-9}
szabo / microether	10^{12}	Transaction fees and protocol implementation	10^{-6}
finney / milliether	10^{15}	Microtransactions	10^{-3}
ether	10^{18}	Normal transactions	1

Table 2.1: Denominations of Ether

Unlike Tezos, Ethereum currently uses a PoW consensus algorithm. In this case, a certain block has a difficulty limit, which is calculated from the previous block's difficulty and timestamp. There is a "nonce" hash which is the 64-bit number to be found and also a 256-bit "mixHash", which is the result of combining the "nonce" hash with the block's hash, proving the proof of work [SDP18]. In other words, a miner tries to find a specific number to which when combined with a block fits some sort of predefined rule. The first miner to find this number gets rewarded and the block gets validated onto the blockchain. Even though, it currently uses PoW, there are plans to upgrade it to PoS. In 2022, Ethereum plans to fully shift to a PoS model, reducing the environmental impact of Ethereum by 99%, actually, right now it has both a PoW and PoS chain running in parallel, having both chains with validators but only the PoW one processing users' transactions [CNB21].

One advantage that Ethereum possesses in contrast to Tezos, is a bigger number of users. Because of this, the blockchain is more decentralized due to the fact of more nodes existing, and also having a larger developer community helps immensely in the development of smart contracts and, in extension, DApps, causing the creation of more tools and frameworks available for the people to use. Compilers, debug tools, libraries and more efficient environments to deploy and make transactions on smart contracts are all examples of this.

This blockchain contains a Turing complete virtual machine or EVM (Ethereum Virtual Machine) which is a simple 256-bit stack machine with a stack size of 1024, where all of the code is commited to the blockchain and accessed from the EVM as virtual Read-only memory (ROM) [BAI+18]. This is also where the smart contracts are read and understood by the blockchain. The high-level languages are compiled down to a low-level, stack-based bytecode language, usually considered EVM code. This code consists of a series of bytes, where each byte represents an operation, which means that the code execution is basically an infinite loop that consists of repeatedly carrying out the operation at the current program counter, which begins at zero, and then incrementing the program counter by one, until the end of the code is reached, or an error, "STOP" or "RETURN" instruction is detected [But13]. EVM bytecode is often represented with a human-readable form called opcodes, which consists of a group of instructions. Opcodes, much like Michelson code, possess instructions with different amounts of weight, meaning each one carries a distinct gas value associated with it. A paper created in 2020 goes in-depth about opcodes, more specifically, it does an analysis of Ethereum's smart contracts' source code, and how they are reflected on the opcode level [BMM+20].

In the context of this paper, in order to deploy the contracts created, there was the need of getting the bytecode and the contract Application Binary Interface (ABI), with the exception of Solidity programs deployed through the truffle environment (see 3.1.5). This ABI, in the context of Ethereum, is an interface referencing the state of the contract, as well as information about the functions in it. This is useful in the case of DApps for example, in order to call a function from an application, one would call via the contract ABI.

Ethereum's whitepaper was originally published in 2013 by Vitalik Buterin, the founder of Ethereum, before the project's launch in 2015 [But13]. And, over the years, the platform has evolved, but the whitepaper maintains as a useful point of reference and a precise depiction of the blockchain to this day.

There is also a yellowpaper associated with this blockchain [Woo21]. This paper thoroughly describes Ethereum in a conglomerate of different concepts. It discusses its design, implementation, potential and protocol it provides. It was originally created by Gavin Wood, co-founder of Ethereum, in 2014, and it is currently being maintained by Nick Savers and other contributors from around the world.

Ethereum has a few public test networks available, in addition to the main network. These networks are called "Görli", "Kovan", "Rinkeby" and "Ropsten". In the context of this dissertation, all of the smart contracts created on the Ethereum blockchain were published and tested on the test network Rinkeby (3.5.2), which is a Proof-of-Authority (PoA) testnet.

2.6 Gas

In the context of both of the blockchains utilized in this dissertation, Tezos and Ethereum, gas represents the computational cost related to a transaction. In these transactions, it is also included the deployment and interaction with a smart contract. This means that the gas number of a transaction executed can and will be used to represent the performance of a certain contract, which will be a very important aspect of this work, because these values are what is going to represent how much effort was used in each interaction and execution of each smart contract created throughout this paper. Each different instruction in the low level languages costs a different amount of gas, which means, depending on the compilation, the resulting set of instructions could amount to very different quantities of gas. Gas can also be used to determine how much an account has to pay to make a certain transaction, for example, in Ethereum, if a transaction is going to use 100.000 gas and the gas price is 0.000000001 ETH or 1 gwei, the price to pay to make this transaction would have been 0.0001 ETH.

2.7 Related Work

With the rise of the blockchain technology, more people have been getting interested in this topic, and therefore more projects and research have been conducted around it. There has been work done revolving around the ledgers themselves, their efficiency, the way they operate, but also the smart contracts embedded in them. Since smart contracts can be very practical and useful, a great deal of DApps were developed with the usage of them. Real life examples of these include financial services, insurances and mortgage systems. Studying and evaluating things such as the performance of these programs can be really important and informative, especially if their use keeps increasing.

What follows is a few studies made in relation to the performance of different concepts involved in blockchain technology, such as, the type of blockchain platforms used, smart contracts of a healthcare system, and also, the evaluation of smart contracts in a blockchain architecture consisted of both on and off-blockchain components. These are not exactly similar to this dissertation, but it involves topics such as the performance of certain applications in blockchains and also the performance of different types of blockchain platforms, which are somewhat related.

There is a paper, made in 2020, about the performance of permissioned blockchain platforms [MSA20]. Permissioned blockchains can be described as a blockchain with an additional layer of security since these platforms require an access control layer. In this study, different platforms are analyzed, such as, a private deployment of Ethereum, Quorum, Corda and Hyperledger Fabric, in metrics of throughput and network latency. It concludes by saying that Hyperledger Fabric performs better than the other platforms.

In 2019, a study was made about the performance of smart contracts in specific healthcare environments [LNB+19]. This paper proposes a solution for healthcare economics systems using blockchain. It evaluates smart contracts, written in Solidity, of the presented solution in both the “Ropsten” test network and a private instance. It also discusses costs related to using a private instance, in comparison to using the Ethereum main network. It concludes stating that the performance in the “Ropsten” network resulted in very similar values to the private instance, and also, that the financial costs of the main network of Ethereum were higher when the number of transactions is high, however, when using the private instance, the cost of the infrastructure and personnel must be considered.

And in 2020, an article was published about the creation and evaluation of smart contracts with the usage of an hybrid on and off-blockchain architecture [SWS20]. This means that these smart contracts are being tested on a environment with components from centralized and decentralized platforms, which the paper states could be more adequate for certain applications. This hybrid architecture was built with the usage of Ethereum, in connection with a centralized smart contract management system. Three different types of blockchain were compared: on-chain, off-chain and hybrid. In the conclusion, it is mentioned that hybrid solutions may be very effective for certain applications such as asset tracking, as well as, when dealing with time related issues. In addition to this, a hybrid solution also reduces the costs involved when comparing to a blockchain based solution.

3

Tools & Frameworks

In this chapter of the dissertation, we list and explain all the tools and frameworks which were used throughout this work. The locations and environments in which the contracts were deployed will be mentioned, the accounts created so it was possible to interact with the contracts and the respective test networks of each blockchain chosen to interact with will be talked about. The nodes that were accessed which contained the test networks will be mentioned. And also each one of the high-level languages that were made use of to create the contracts in this paper will be described in more detail.

3.1 Environments

There are a few environments which were crucial to the realization of this paper, because these involved in the interaction with the nodes of the blockchains, as well as, the deployment and interaction with the smart contracts created. Some of these served as an interface between user and client of the blockchain, others are frameworks which assist in different aspects.

3.1.1 Ubuntu

Ubuntu is a Linux command-line interface. This terminal can be used to navigate through folders and directories, and also the installation of packages and other programs. It's really useful for file management, development, administration, etc. In order to use this terminal on a Windows operating system, which is the case, something called the "Windows subsystem for Linux" was installed, so Ubuntu can be run. The version used here is "Ubuntu 18.04.5 LTS", and this is the location where the tezos-client was installed.

3.1.2 Tezos-Client

Tezos-client is used as an interface between the user and the blockchain Tezos. This client reached Tezos on version 9.0. To interact with the blockchain, it was necessary to install several binaries, which are executable files, consisting of a client, a node, a baker and an endorser [TDR21]. In these files is also included the respective network meant to work in, which in this case, is the Florence network. In order to connect to a node, a user can either run their own node, or use a community node to access the Tezos blockchain. This is the tool used to basically do everything in Tezos, in this case these things are: create accounts with the usage of a faucet, deploy smart contracts (already compiled to Michelson code), execute transactions with a certain argument of a specific smart contract address, and check the storage of a contract. Not all of the Tezos' smart contracts were deployed by using tezos-client, some languages were able to deploy from their compiler interfaces.

3.1.3 Powershell

Powershell is a command-line terminal interface. In some versions it is called Windows Powershell, but on later versions it's just Powershell or Microsoft Powershell. With this tool, one can access, and explore data in multiple locations, and just like Ubuntu, it can also be used to provide third-party packages to be installed. The version used here is 5.1.19041.1320, and it was used to install npm, which in turn was used to install and interact with the frameworks Truffle and Waffle.

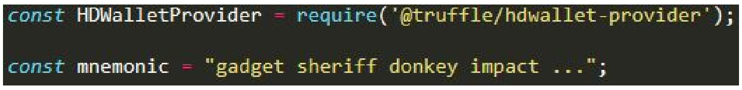
3.1.4 Node.js & npm

Node.js is a JavaScript runtime environment designed to build scalable network applications with the use of making connections to servers [OF21]. The version used is 14.16.1.

npm is a JavaScript package manager which requires Node.js to be installed in order to be able to run. The free npm registry provides a public collection of over one million packages available, making it the largest software registry in the world [nI21]. Among these packages, the ones installed and used were Truffle and Waffle. The npm version used here is version 7.24.2.

3.1.5 Truffle

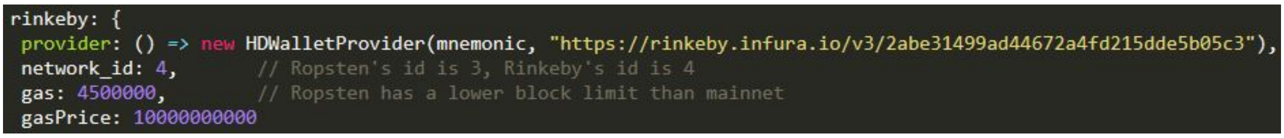
Truffle is a world class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make life as a developer easier [Sui21]. Truffle offers many useful tools, especially with a built-in smart contract compilation and deployment, which is the primary aspect involved in this paper. Through the "truffle-config.js" file, the wallet of the account can be associated, and also the provider of the node to be used. The Truffle version used is 5.4.0. All of the solidity contracts compiled with Truffle use the version 0.5.0 with compiler “solc”. The figures 3.1, 3.2 and 3.3 refer to configurations in the “truffle-config.js” file. In figure 3.1, the mnemonic of an account is defined as a variable.



```
const HDWalletProvider = require('@truffle/hdwallet-provider');

const mnemonic = "gadget sheriff donkey impact ...";
```

Figure 3.1: truffle-config.js account mnemonic

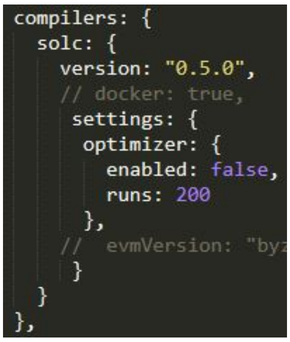
In figure 3.2, the definition of the node to be used is shown, in combination with the account’s mnemonic, and also, the choosing of the correct test network id.



```
rinkeby: {
 provider: () => new HDWalletProvider(mnemonic, "https://rinkeby.infura.io/v3/2abe31499ad44672a4fd215dde5b05c3"),
 network_id: 4,       // Ropsten's id is 3, Rinkeby's id is 4
 gas: 4500000,        // Ropsten has a lower block limit than mainnet
 gasPrice: 10000000000
```

Figure 3.2: truffle-config.js node provider

And in figure 3.3, it can be seen the solidity compiler version, which is 0.5.0, and also, the location of where the optimizer can be turned off and on (by changing “false” to “true”, and vice-versa).

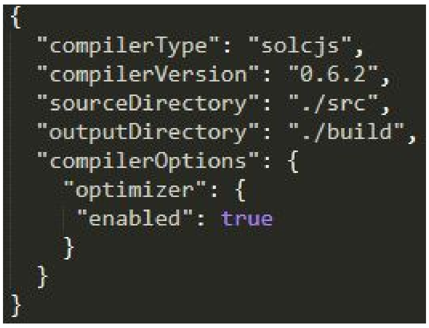


```
compilers: {
  solc: {
    version: "0.5.0",
    // docker: true,
     settings: {
      optimizer: {
        enabled: false,
        runs: 200
      },
    //  evmVersion: "by
     }
  }
},
```

Figure 3.3: truffle-config.js compiler and optimizer

3.1.6 Waffle

Waffle is a library for writing and testing smart contracts, and it is apparently simpler and faster than truffle [Waf20]. This is a library with the intend to be simpler, with less dependencies, easy syntax to understand and faster on execution. As can be seen in figure 3.4, the “waffle.json” makes use of the “solcjs” compiler and uses the solidity version 0.6.2, which is the version the contracts compiled here are at. This file also enables the possibility to turn the optimizer on or off, in addition to, defining the folder where the contracts are, which is “./src”, and defining the folder where the compilation results of said contracts are, which is “./build”.



```
{
  "compilerType": "solcjs",
  "compilerVersion": "0.6.2",
  "sourceDirectory": "./src",
  "outputDirectory": "./build",
  "compilerOptions": {
    "optimizer": {
      "enabled": true
    }
  }
}
```

Figure 3.4: waffle.json compiler and optimizer

3.2 Accounts

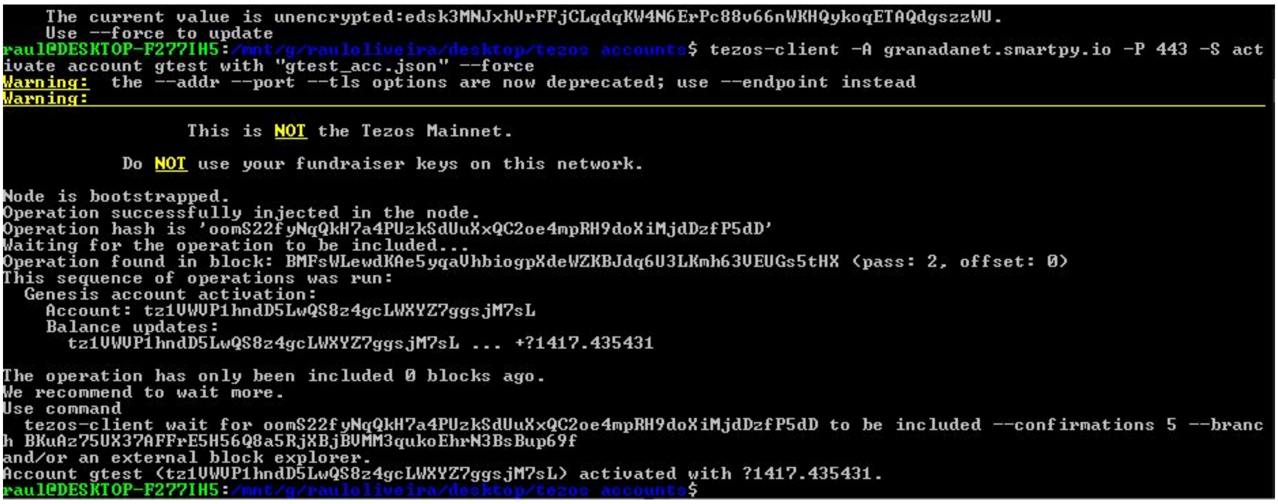
There are two types of accounts that can be created on these blockchains, there are user accounts and smart contract accounts. In Tezos, user account addresses start with "tz1", and smart contract addresses start with "KT1". In Ethereum, they both start with "0x". Here it will be mentioned the user accounts created to interact with the deployment and transactions of the contracts. When an account is created, it has something called "mnemonic", that defines a list of strings which designates the identification of an account.

In the case of Tezos blockchain, an account can be generated with the use of a faucet (test networks only), which can then be activated by using tezos-client. In figure 3.5 a JavaScript Object Notation (JSON) file can be seen with a generated account from a faucet.

```
{
  "mnemonic": [
    "own",
    "garlic",
    "dwarf",
    "art",
    "color",
    "lunch",
    "exclude",
    "diary",
    "cake",
    "hybrid",
    "someone",
    "ostrich",
    "charge",
    "rescue",
    "unfold"
  ],
  "secret": "83729dda1314bffdb412cccfa1bf3ddba3cccb72",
  "amount": "1417435431",
  "pkh": "tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL",
  "password": "W5kePPwl66",
  "email": "cecbrjal.vspbnvib@tezos.example.org"
}
```

Figure 3.5: JSON file of account generated from faucet

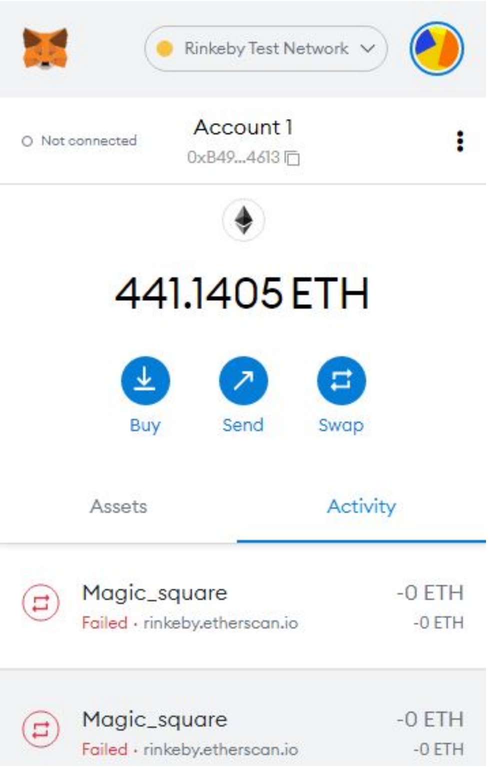
And in figure 3.6, the respective activation can be seen done on the Granada test network with around 1417 tez. This wasn't the network used in this paper since it's no longer maintained but the process is the same. The activation is done using a public node and the account is given the alias "gtest", calling the JSON file shown in figure 3.5.



```
    The current value is unencrypted:edsk3MNJxhVrFFjCLqdqKW4N6ErPc88v66nWKHQykoqETAQdgszzWU.
    Use --force to update
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/tezos accounts$ tezos-client -A granadanet.smartpy.io -P 443 -S act
ivate account gtest with "gtest_acc.json" --force
Warning:  the --addr --port --tls options are now deprecated; use --endpoint instead
Warning:

                 This is NOT the Tezos Mainnet.

           Do NOT use your fundraiser keys on this network.

Node is bootstrapped.
Operation successfully injected in the node.
Operation hash is 'oomS22fyNqQkH7a4PUzkSdUuXxQC2oe4mpRH9doXiMjdDzfP5dD'
Waiting for the operation to be included...
Operation found in block: BMFsWLewdKAe5yqaVhbiogpXdeWZKBJdq6U3LKmh63VEUGs5tHX (pass: 2, offset: 0)
This sequence of operations was run:
  Genesis account activation:
    Account: tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL
    Balance updates:
      tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL ... +?1417.435431

The operation has only been included 0 blocks ago.
We recommend to wait more.
Use command
  tezos-client wait for oomS22fyNqQkH7a4PUzkSdUuXxQC2oe4mpRH9doXiMjdDzfP5dD to be included --confirmations 5 --branc
h BKuAz75UX37AFFrE5H56Q8a5RjXBjBVMM3qukoEhrN3BsBup69f
and/or an external block explorer.
Account gtest (tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL) activated with ?1417.435431.
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/tezos accounts$
```

Figure 3.6: Activation of account generated

In the case of the Ethereum blockchain, an account was created with the use of the tool MetaMask. This was used as a crypto wallet of the account created. MetaMask is used as a browser extension and equips the user with a key vault, secure login, token wallet and token exchange, which can be used to buy, store, send or swap tokens [Met21]. In figure 3.7, the account used in this paper is seen, in combination with the amount of ether that it holds and the last few transactions made.



Rinkeby Test Network

Not connected

Account 1

0xB49...4613

441.1405 ETH

Magic_square -0 ETH

Failed · rinkeby.etherscan.io -0 ETH

Magic_square -0 ETH

Failed · rinkeby.etherscan.io -0 ETH

Figure 3.7: Metamask account

And in order to fund this account with ether, a faucet was used. But, unlike the one in Tezos, which, when a button is pressed, generates an account with a random amount of currency, in this one, the user has to have the account already created. In figure 3.8, Rinkeby's faucet interface can be seen. In this faucet, each person has to make a predefined post on one of two social media networks, twitter or facebook, with the address wishing to be fund. The post only needs to be made once for each account. There are 3 options of currency to get. The left amounts correspond to the ether to gain, and the right ones indicate a cool-down to have to wait to use the faucet again. So if an individual chose the 18.75 ETH option, they would have to wait 3 days (72 hours), to use the faucet once again.

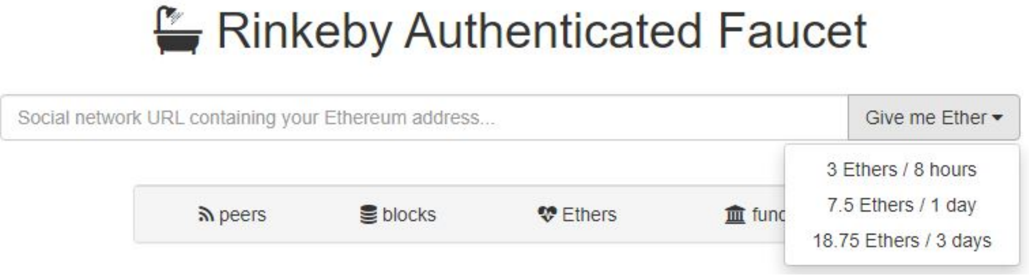


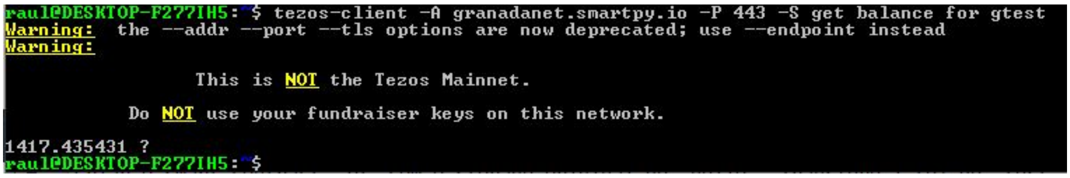
Figure 3.8: Rinkeby faucet

3.3 Nodes

Like mentioned previously, one can either make use of a public community-run node to interact with the networks, or run their own node, which is usually considered more secure. In the case of the Florence Network, public nodes were used, due to the fact that the local node couldn't sync up to the current version of the blockchain. And in the case of the Rinkeby network, a tool named Infura was used to connect to the blockchain.

3.3.1 Public Nodes

Multiple public nodes were utilized on the Tezos blockchain. Every time a certain action was executed using tezos-client, whether it be, creating a user account, deploying a smart contract or interacting with a contract, there was the need to point out the respective node to interact with. The public nodes used consisted of "smartpy.io" nodes and "giganode.io" nodes. Tezos Giganode has, since October of 2021, been shutdown, so the "smartpy.io" ones were more used has of late. For the making of this work, the nodes "testnet-tezos.giganode.io" and "florencenet.smartpy.io" were the ones who were accessed the most. To connect to a node using tezos-client, one would have to do "-A" indicating the address with the node in front, and then "-P" with the number of the port and then "-S" which is used to connect to the node. And after all of this, the command wanted to be executed. In figure 3.9, an example can be seen. In this example, the account balance of the "gtest" account created earlier (see figure 3.6), is retrieved. The "?" represents the tez symbol, which Ubuntu couldn't process.



```
raul@DESKTOP-F277IH5:~$ tezos-client -A granadanet.smartpy.io -P 443 -S get balance for gtest
Warning:  the --addr --port --tls options are now deprecated; use --endpoint instead
Warning:

                 This is NOT the Tezos Mainnet.

           Do NOT use your fundraiser keys on this network.

1417.435431 ?
raul@DESKTOP-F277IH5:~$
```

Figure 3.9: Get account balance

3.3.2 Infura Node

Infura is an Application Programming Interface (API) provider, helping developers get easy, simple access to the Ethereum blockchain by making use of their nodes. Infura built services and APIs around JSON Remote Procedure Call (RPC) over both Hypertext Transfer Protocol Secure (HTTPS) and WebSocket that someone can use with their favorite libraries and frameworks, on four Ethereum networks [Inf21]. In other words, by using Infura, I am not running my own node exactly, but basically using a framework that gives me access to a node of my own.

After signing up and creating a project, there is a project ID, in the project's node general details, which is shown in figure 3.10. In this case, the endpoint which has the ID was used in the "truffle-config.js" file, which can be seen on figure 3.2.

PROJECT ID

2abe31499ad44672a4fd215dde5b
05c3

PROJECT SECRET

d15952624581495c91e602cb4020
4b4f

ENDPOINTS RINKEBY

https://rinkeby.infura.io/v3/2abe31499ad44672a4fd215dde5b05c3

wss://rinkeby.infura.io/ws/v3/2abe31499ad44672a4fd215dde5b05c3

Figure 3.10: Infura node project

3.4 Languages

In this section it will be mentioned a brief description of every single high-level language chosen to evaluate in this dissertation. It will be indicated as well, the versions of the languages in each of the problems used and described in section 4.1. In terms of how they were compiled and deployed is explained and demonstrated in section 3.5.

3.4.1 SmartPy

SmartPy is a Python based language for the Tezos blockchain, which is available through a Python library which comes with a framework built for testing and compilation of the programs. It also possesses an explorer area, where someone could deploy a previously compiled contract. The versions used are 0.6.9 (on FirstNPrimes), 0.6.11 (on FirstNPrimesList, FirstNPrimesBoth and NQueens), 0.7.1 (on FirstNPrimesMap) and 0.7.4 (on MagicSquare).

3.4.2 Liquidity

Liquidity is a functional language for the Tezos blockchain, which borrows syntax from the languages OCaml and ReasonML, with focus on security. It contains a testing and compiler framework and a deployment method for the Dune network, which is a platform of applications over a blockchain. It also possesses a way to decompile Michelson code into Liquidity, which is very interesting. All of the programs written in Liquidity in this paper are done so in version 2.0.

3.4.3 LIGO

The LIGO languages are functional languages composed of three different languages, which can be used on the Tezos blockchain. LIGO was created to bring a familiar interface in three distinct aspects.

PascaLIGO

PascaLIGO is one of the LIGO languages, which has a syntax inspired by Pascal. The versions used are 0.20.0 (on FirstNPrimes), 0.21.0 (on FirstNPrimesList and FirstNPrimesBoth), 0.22.0 (on NQueens), 0.23.0 (on FirstNPrimesMap) and 0.25.0 (on MagicSquare).

CameLIGO

CameLIGO is another of the LIGO languages, which has a functional style inspired by OCaml. The versions used are 0.20.0 (on FirstNPrimes), 0.21.0 (on FirstNPrimesList and FirstNPrimesBoth), 0.22.0 (on NQueens), 0.23.0 (on FirstNPrimesMap) and 0.25.0 (on MagicSquare).

ReasonLIGO

And ReasonLIGO has a ReasonML similar syntax that goes off of the strong points of OCaml, this is probably why they ReasonLIGO and CameLIGO are so alike. The versions used are 0.20.0 (on FirstNPrimes), 0.21.0 (on FirstNPrimesList and FirstNPrimesBoth), 0.22.0 (on NQueens), 0.23.0 (on FirstNPrimesMap) and 0.25.0 (on MagicSquare).

3.4.4 Archetype

Archetype is a DSL for the Tezos blockchain, where it restricts the type of programs that can be expressed. The reason for this is that it makes this language easy to specify and verify formally. Like Liquidity, it also focuses on security. It provides a work space framework which consists of compilation and deployment of contracts. The versions used are 1.2.6 (on FirstNPrimes, FirstNPrimesList and FirstNPrimesBoth), 1.2.7 (on FirstNPrimesMap and NQueens) and 1.2.8 (on MagicSquare).

3.4.5 Solidity

Solidity is a statically-typed, object-oriented language for the Ethereum blockchain. It's most likely the most used out of all the high-level languages mentioned in this paper, due to being the main one of Ethereum (the biggest of the two blockchains), where most of its smart contracts are written in this language. The contracts made in Truffle have the version 0.5.0 in all of the problems, and the ones made in Waffle have the version 0.6.2 in all of the problems as well.

3.4.6 Vyper

Vyper is a strongly-typed language for the Ethereum blockchain, which leverages Python's unique features. Vyper was created intentionally restrictive in order to make the contracts more secure and easier to inspect, which is why it possesses less attributes than Solidity. All of the Vyper programs created, are done so in version 0.2.13.

3.5 Networks

Here are the test networks determined to deploy the contracts into. After setting up the accounts with the respective currencies and a way to access the nodes needed to reach these networks, compiled contracts can now be deployed, or in some cases, compiled and deployed in the same framework.

3.5.1 Florence Network

Florence network is a Tezos test network, which was the network chosen due to, not only being the newest test network at the beginning of its usage in this work (around June 2021), but also in comparison to the previous one, it had gas optimizations. It came with reduced gas consumption in smart contract execution by increasing the efficiency of gas computation inside the Michelson interpreter [TAF21b].

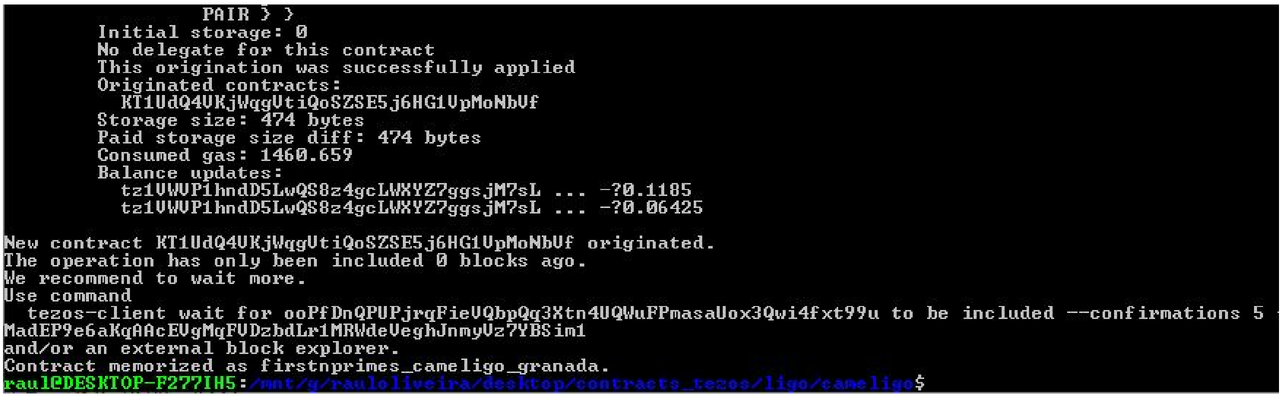
In the case of the Florence network, contracts were deployed and interacted using tezos-client and a public node. This is not the case with all of the Tezos' languages though, and those cases will be shown further ahead. But for now, an example of a contract, already in Michelson code, is going to be shown being deployed into the Granada network, and also being interacted with.

As an example, the "FirstNPrimes" (see 4.1.1) program made in CameLIGO, was deployed on Granadanet. The command used to achieve this can be seen in figure 3.11.

```
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/contracts_tezos/ligo/cameligo$ tezos-client -A granadanet.smartpy.io -
P 443 -S originate contract firstnprimes_cameligo_granada transferring 0 from gtest running FirstNPrimes.tz --init '0'
--burn-cap 0.2
```

Figure 3.11: Deploy contract command

In this command, it's done the access to the node, and then, by being in the appropriate directory, it originates the contract, already compiled to Michelson from CameLIGO, of "FirstNPrimes". It is given a name, and then a number is transferred from the user account into it, in this case, there's no need to send an actual amount bigger than zero because this contract doesn't enforce payable transactions, nor does it need money in its balance. The "–init '0"' means it initializes the contract's storage at 0, and the "–burn-cap 0.2", puts a cap on the amount of tez to burn with this operation, otherwise it would default to 0. After the operation being successful and injecting it in the node, in figure 3.12, it can be seen what is shown next.



```
                PAIR } }
        Initial storage: 0
        No delegate for this contract
        This origination was successfully applied
        Originated contracts:
          KT1UdQ4VKjWqgVtiQoSZSE5j6HG1VpMoNbVf
        Storage size: 474 bytes
        Paid storage size diff: 474 bytes
        Consumed gas: 1460.659
        Balance updates:
          tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL ... -?0.1185
          tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL ... -?0.06425

New contract KT1UdQ4VKjWqgVtiQoSZSE5j6HG1VpMoNbVf originated.
The operation has only been included 0 blocks ago.
We recommend to wait more.
Use command
  tezos-client wait for ooPfDnQPUPjrqFieVQbpQq3Xtn4UQWuFPmasaUox3Qwi4fxt99u to be included --confirmations 5
MadEP9e6aKqAAcEVgMqFVDzbdLr1MRWdeVeghJnmyVz7YBSim1
and/or an external block explorer.
Contract memorized as firstnprimes_cameligo_granada.
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/contracts_tezos/ligo/cameligo$
```

Figure 3.12: Contract deployed with success

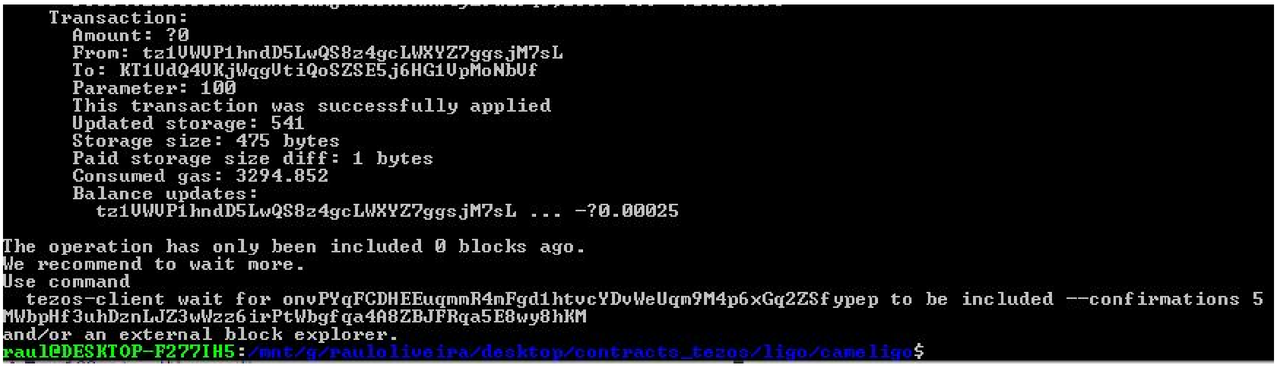
At the end, is shown the initial storage, the address of the new contract created on the network, the storage size and the consumed gas for this operation. There's also balance updates on how much tez was spent from the user account used to make this transaction.

And then, a transaction is made into this newly deployed contract, and figure 3.13 shows the command that was used to call the contract with the argument 100.

```
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/contracts_tezos/ligo/cameligo$ tezos-client -A granadanet.smartpy.io -
P 443 -S transfer 0 from gtest to KT1UdQ4VKjWqgVtiQoSZSE5j6HG1VpMoNbVf --arg '100' --burn-cap 0.2
```

Figure 3.13: Contract transaction command

In this command, the node is accessed, sending 0 tez to the contract because there is no need to send more, the account address is used to identify the contract to use (the alias, "firstnprimes_cameligo_granada" created before, would've also worked here, since this contract was deployed in the same location, the alias is saved locally as that specific address), and then the argument 100 sent to make the transaction. In figure 3.14, we see the aftermath of said transaction.



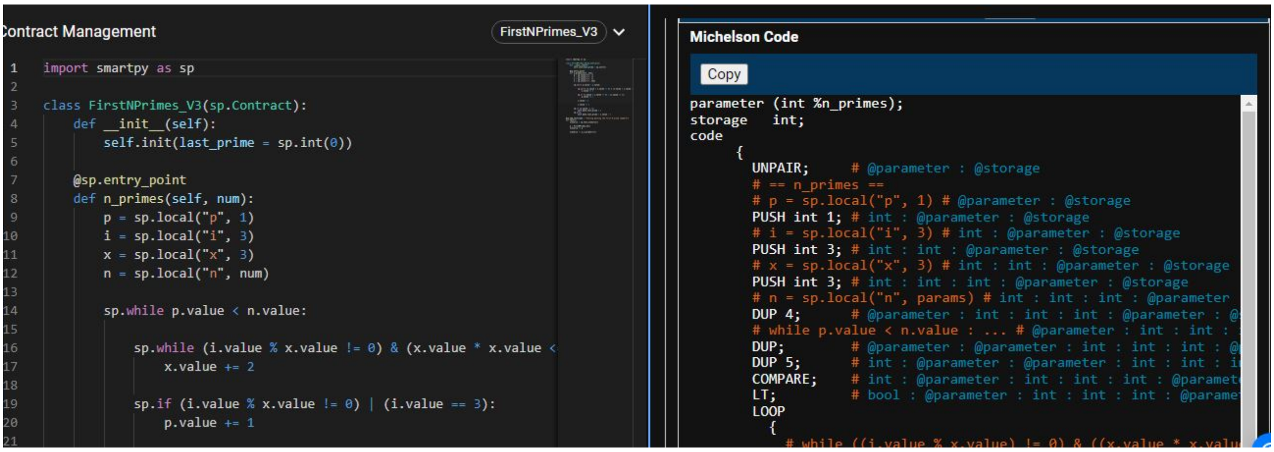
```
    Transaction:
      Amount: ?0
      From: tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL
      To: KT1UdQ4VKjWqgVtiQoSZSE5j6HG1VpMoNbVf
      Parameter: 100
      This transaction was successfully applied
      Updated storage: 541
      Storage size: 475 bytes
      Paid storage size diff: 1 bytes
      Consumed gas: 3294.852
      Balance updates:
        tz1VWVP1hndD5LwQS8z4gcLWXYZ7ggsjM7sL ... -?0.00025

The operation has only been included 0 blocks ago.
We recommend to wait more.
Use command
  tezos-client wait for onvPYqFCDHEEuqmmR4mFgd1htvcYDvWeUqm9M4p6xGq2ZSfypep to be included --confirmations 5
MWbpHf3uhDznLJZ3wWzz6irPtWbgfqa4A8ZBJFRqa5E8wy8hKM
and/or an external block explorer.
raul@DESKTOP-F277IH5:/mnt/g/rauloliveira/desktop/contracts_tezos/ligo/cameligo$
```

Figure 3.14: Contract transaction with success

And here, we can see the parameter sent, the updated storage, which is now 541, since this is the 100th prime number and this is the intended result so it's correct. We see that the storage got increased by 1 byte, the gas consumed and also the balance update of the user account.

But before being able to deploy, one would need the respective compiled Michelson code of each program. All the following examples were shown with the FirstNPrimes program (see 4.1.1).

In SmartPy, the SmartPy Integrated Development Environment (IDE) was used to write, test and compile SmartPy code. In figure 3.15, the contract written in SmartPy can be seen, and also the compiled Michelson code on the right.



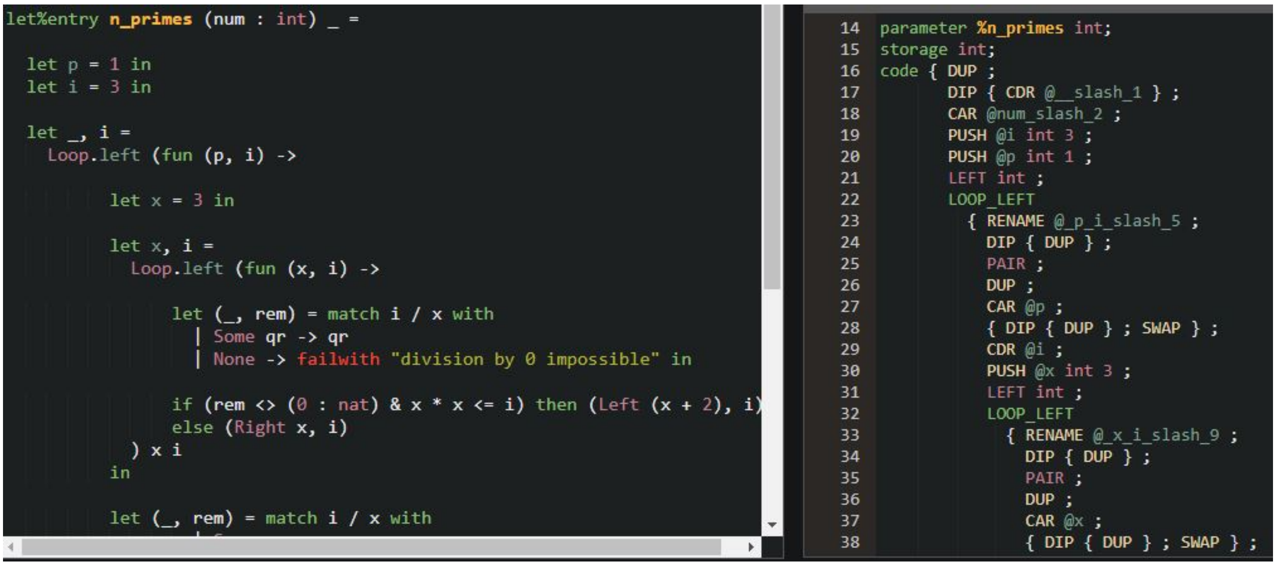
```
import smartpy as sp

class FirstNPrimes_V3(sp.Contract):
    def __init__(self):
        self.init(last_prime = sp.int(0))

    @sp.entry_point
    def n_primes(self, num):
        p = sp.local("p", 1)
        i = sp.local("i", 3)
        x = sp.local("x", 3)
        n = sp.local("n", num)

        sp.while p.value < n.value:

            sp.while (i.value % x.value != 0) & (x.value * x.value <
                x.value += 2

            sp.if (i.value % x.value != 0) | (i.value == 3):
                p.value += 1

```

```
parameter (int %n_primes);
storage   int;
code
     {
       UNPAIR;      # @parameter : @storage
       # == n_primes ==
       # p = sp.local("p", 1) # @parameter : @storage
       PUSH int 1; # int : @parameter : @storage
       # i = sp.local("i", 3) # int : @parameter : @storage
       PUSH int 3; # int : int : @parameter : @storage
       # x = sp.local("x", 3) # int : int : @parameter : @storage
       PUSH int 3; # int : int : int : @parameter : @storage
       # n = sp.local("n", params) # int : int : int : @parameter
       DUP 4;      # @parameter : int : int : int : @parameter : @
       # while p.value < n.value : ... # @parameter : int : int :
       DUP;        # @parameter : @parameter : int : int : int : @
       DUP 5;      # int : @parameter : @parameter : int : int : i
       COMPARE;    # int : @parameter : int : int : int : @paramet
       LT;         # bool : @parameter : int : int : int : @parame
       LOOP
         {
           # while ((i.value % x.value) != 0) & ((x.value * x.valu
```

Figure 3.15: SmartPy IDE

This website also contains an explorer area, where the compiled contract could be deployed. This was tested but the results in gas were the same as doing it in tezos-client.

In Liquidity, the Liquidity IDE was used to write, test and compile Liquidity written programs, shown in figure 3.16. These were deployed on tezos-client.

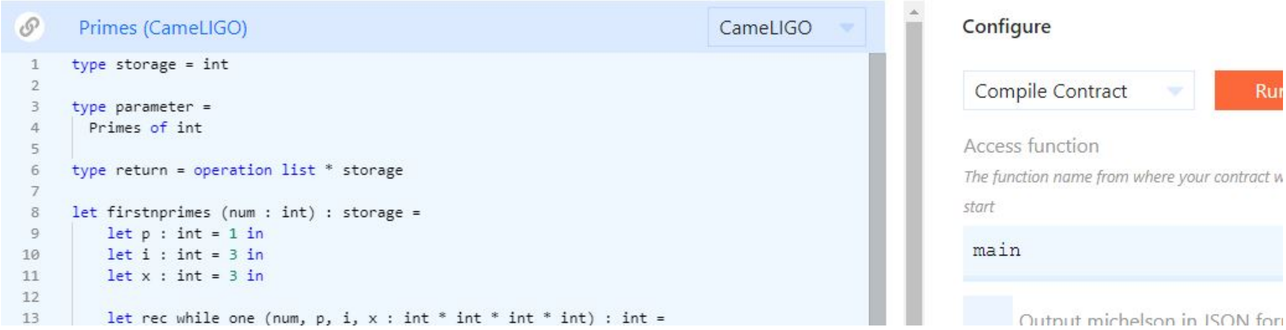


```
let%entry n_primes (num : int) _ =

  let p = 1 in
  let i = 3 in

  let _, i =
    Loop.left (fun (p, i) ->

          let x = 3 in

          let x, i =
            Loop.left (fun (x, i) ->

                let (_, rem) = match i / x with
                  | Some qr -> qr
                  | None -> failwith "division by 0 impossible" in

                if (rem <> (0 : nat) & x * x <= i) then (Left (x + 2), i)
                else (Right x, i)
            ) x i
          in

          let (_, rem) = match i / x with
```

```
parameter %n_primes int;
storage int;
code { DUP ;
       DIP { CDR @__slash_1 } ;
       CAR @num_slash_2 ;
       PUSH @i int 3 ;
       PUSH @p int 1 ;
       LEFT int ;
       LOOP_LEFT
         { RENAME @_p_i_slash_5 ;
           DIP { DUP } ;
           PAIR ;
           DUP ;
           CAR @p ;
           { DIP { DUP } ; SWAP } ;
           CDR @i ;
           PUSH @x int 3 ;
           LEFT int ;
           LOOP_LEFT
             { RENAME @_x_i_slash_9 ;
               DIP { DUP } ;
               PAIR ;
               DUP ;
               CAR @x ;
               { DIP { DUP } ; SWAP } ;
```

Figure 3.16: Liquidity IDE

In the LIGO languages, the LIGO IDE was used to write, test and compile all three of the LIGO languages. These were deployed on tezos-client. In figure 3.17, the LIGO IDE can be seen with a program written in CameLIGO, by choosing the respective language at the top.



```
type storage = int

type parameter =
  Primes of int

type return = operation list * storage

let firstnprimes (num : int) : storage =
    let p : int = 1 in
    let i : int = 3 in
    let x : int = 3 in

    let rec while_one (num, p, i, x : int * int * int * int) : int =
```

Access function

The function name from where your contract w

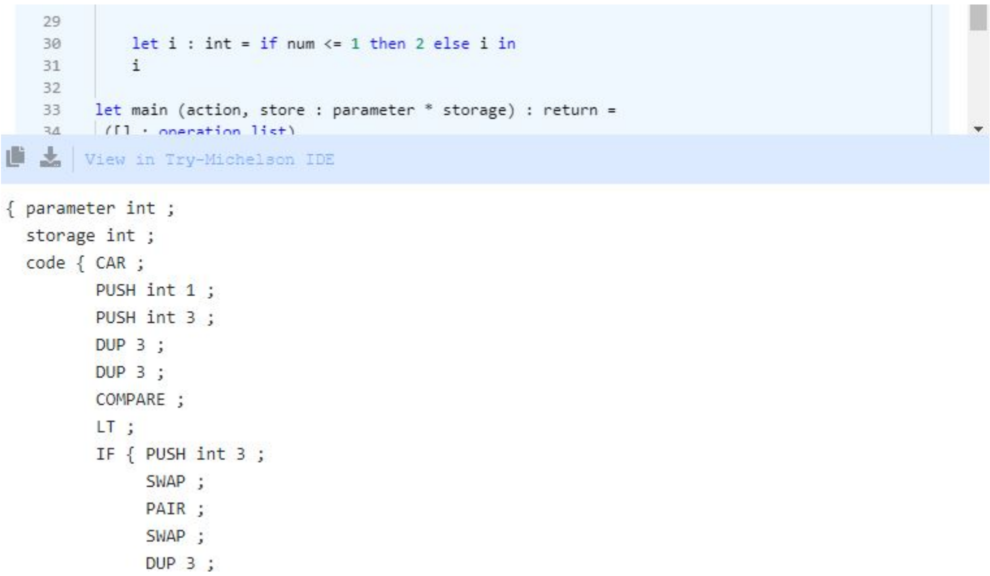
start

main

Output michelson in JSON for

Figure 3.17: LIGO IDE

And by clicking "Run" on the "Compile Contract" function, we can see the Michelson output at the bottom below the program written, as can be seen in figure 3.18.

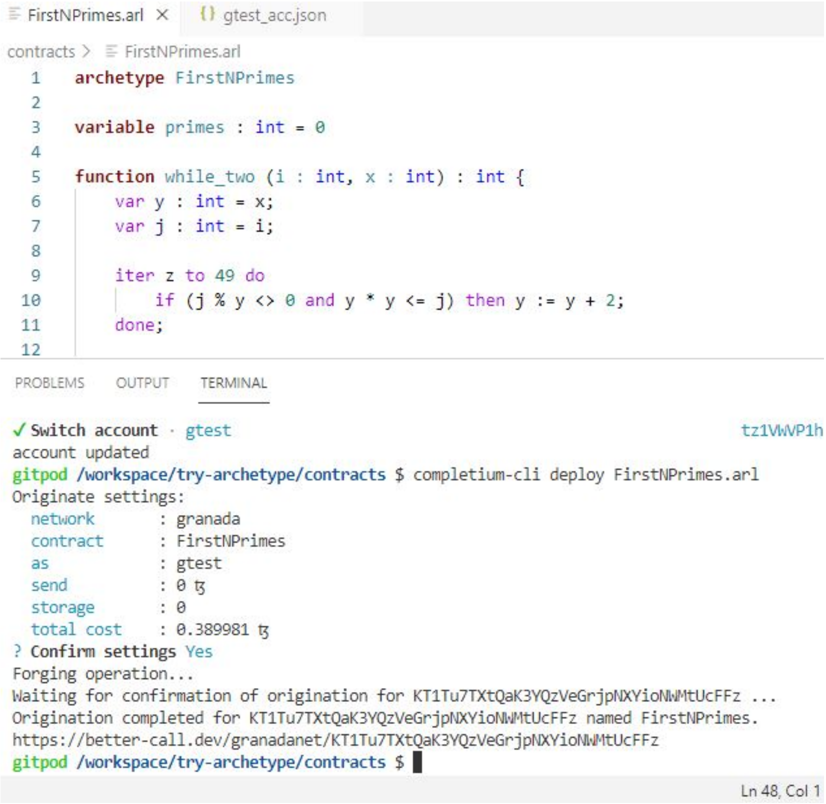


```
    let i : int = if num <= 1 then 2 else i in
    i

let main (action, store : parameter * storage) : return =
 ([] : operation list)
```

```
{ parameter int ;
  storage int ;
  code { CAR ;
         PUSH int 1 ;
         PUSH int 3 ;
         DUP 3 ;
         DUP 3 ;
         COMPARE ;
         LT ;
         IF { PUSH int 3 ;
              SWAP ;
              PAIR ;
              SWAP ;
              DUP 3 ;
```

Figure 3.18: LIGO IDE (Michelson output)

In the case of the Archetype language, the website possesses a button which redirects the user to gitpod.io. After logging in with GitHub, it creates a fresh work space with previously made archetype contracts.



```
archetype FirstNPrimes

variable primes : int = 0

function while_two (i : int, x : int) : int {
    var y : int = x;
    var j : int = i;

    iter z to 49 do
        if (j % y <> 0 and y * y <= j) then y := y + 2;
    done;

```

```
✓ Switch account · gtest                                                    tz1VWVP1h
account updated
gitpod /workspace/try-archetype/contracts $ completium-cli deploy FirstNPrimes.arl
Originate settings:
  network       : granada
  contract      : FirstNPrimes
  as            : gtest
  send          : 0 ꜩ
  storage       : 0
  total cost    : 0.389981 ꜩ
? Confirm settings Yes
Forging operation...
Waiting for confirmation of origination for KT1Tu7TXtQaK3YQzVeGrjpNXYioNWMtUcFFz ...
Origination completed for KT1Tu7TXtQaK3YQzVeGrjpNXYioNWMtUcFFz named FirstNPrimes.
https://better-call.dev/granadanet/KT1Tu7TXtQaK3YQzVeGrjpNXYioNWMtUcFFz
gitpod /workspace/try-archetype/contracts $ █
```

Figure 3.19: Archetype gitpod work space

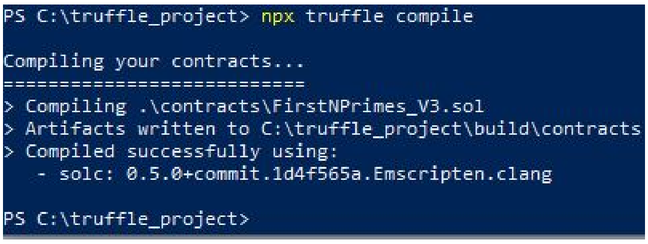
The FirstNPrimes program is created here, the details of account "gtest" are imported and activated with commands "completium-cli import faucet ftest_acc.json as ftest", and "completium-cli switch account" to switch to it. Now there's two ways to go about here. Either the command "completium-cli generate michelson FirstNPrimes.arl" is used to just get the Michelson code, or the command "completium-cli deploy FirstNPrimes.arl" to deploy right away. In figure 3.19, this interface with the code and the terminal can be seen, with the second command executed.

3.5.2 Rinkeby Network

Rinkeby network is a PoA test network for the Ethereum blockchain. PoA is a variation of the PoS. In this consensus algorithm, the nodes who are known for a while and trusted are chosen to validate the blocks. A validator doesn't need to stake any of its assets, like in PoS, but instead its own reputation. This basically means they stake the amount of time they have spent in the network, this becomes better than PoS in the sense that in PoS, the more wealthy get more of the benefits, but here its the oldest members who get the rewards, which in turn, runs into the issue of only a few nodes being in charge, which highly decreases decentralization. [OTJA21].

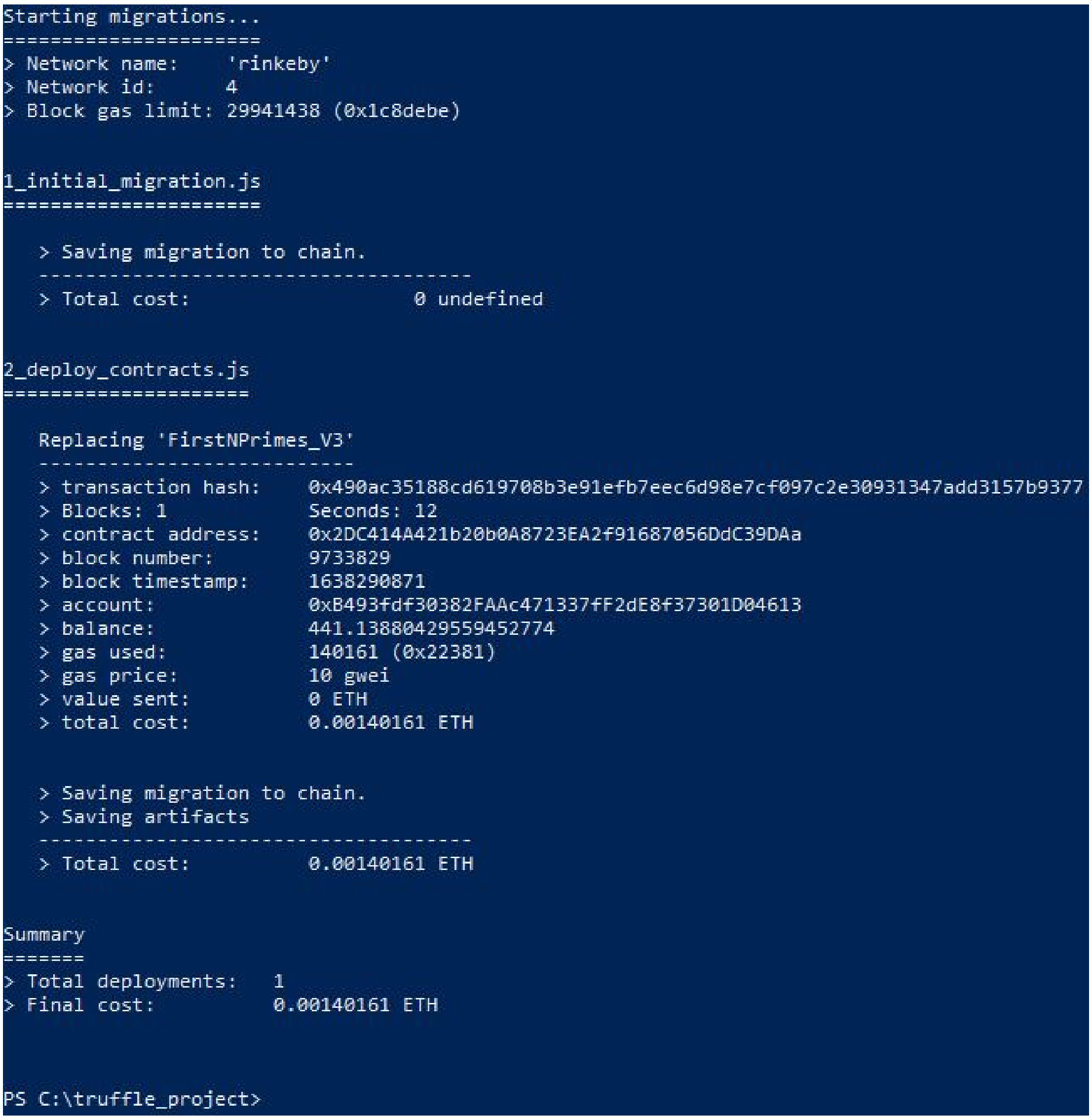
The Ethereum blockchain has different free test networks available to the public. It has four test networks, and each one of these correspond to a unique ID. These networks are "Görli" with an ID of 6284, "Kovan" with an ID of 42, "Ropsten" with an ID of 3 and the relevant one used here "Rinkeby" with an ID of 4. This ID is the network ID indicated in the "truffle-config.js" file, to tell Truffle that the provider link of Infura corresponds to a Rinkeby node (see 3.2).

In the case of the Rinkeby network, contracts got compiled in various different ways. With the use of Truffle, Waffle, a private environment and MyEtherWallet. All of these ways will be shown through examples, and also how the transactions were deployed and executed.



```
PS C:\truffle_project> npx truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\FirstNPrimes_V3.sol
> Artifacts written to C:\truffle_project\build\contracts
> Compiled successfully using:
   - solc: 0.5.0+commit.1d4f565a.Emscripten.clang

PS C:\truffle_project>
```

Figure 3.20: Truffle program compiled



```
Starting migrations...
======================
> Network name:    'rinkeby'
> Network id:      4
> Block gas limit: 29941438 (0x1c8debe)


1_initial_migration.js
======================

   > Saving migration to chain.
   -------------------------------------
   > Total cost:                   0 undefined


2_deploy_contracts.js
=====================

   Replacing 'FirstNPrimes_V3'
   ---------------------------
   > transaction hash:    0x490ac35188cd619708b3e91efb7eec6d98e7cf097c2e30931347add3157b9377
   > Blocks: 1            Seconds: 12
   > contract address:    0x2DC414A421b20b0A8723EA2f91687056DdC39DAa
   > block number:        9733829
   > block timestamp:     1638290871
   > account:             0xB493fdf30382FAAc471337fF2dE8f37301D04613
   > balance:             441.13880429559452774
   > gas used:            140161 (0x22381)
   > gas price:           10 gwei
   > value sent:          0 ETH
   > total cost:          0.00140161 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00140161 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.00140161 ETH



PS C:\truffle_project>
```

Figure 3.21: Truffle program migrated

In Truffle, after setting up the truffle-config.js file, and putting the solidity programs wanted to be compiled and deployed (because truffle does both) on the "contracts" folder, a variable is set in the file

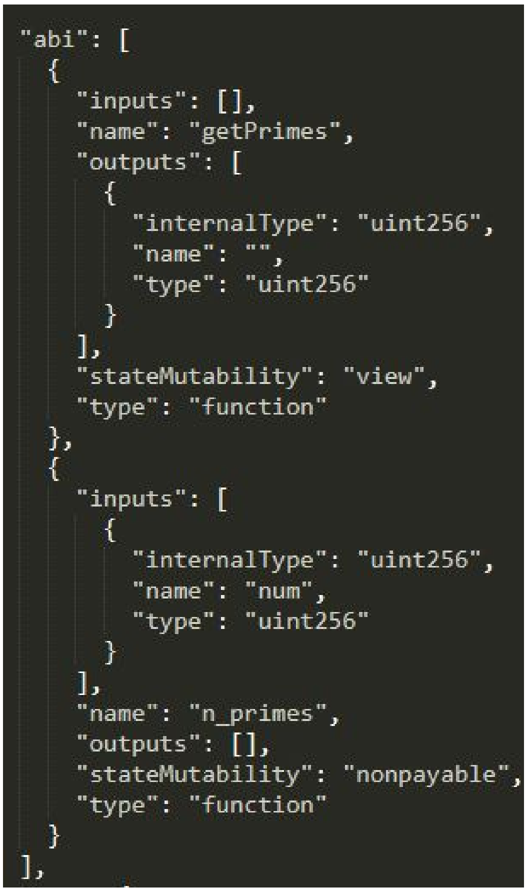
“2_deploy_contracts.js” that calls the “.sol” file which calls a function to deploy said variable. After this, on Powershell, while being on the truffle project created directory, the command “npx truffle compile” is used to compile all of the contracts and the compiled versions go to the folder “build”, also called artifacts, as can be seen on figure 3.20.

In this case, like before, the program FirstNPrimes (see 4.1.1) written in Solidity, was chosen as an example.

And then, after having the specific contracts meant to be deployed on the “2_deploy_contracts.js” configured to be migrated, the command “npx truffle migrate –network rinkeby” can be issued to deploy the chosen contracts into Rinkeby. Sometimes, “–reset” is needed in front of the command due to already having deployed these programs previously. This command then executes a dry-run simulation of the migration, and then if everything goes well, it does the actual migration. In figure 3.21, the migration can be seen.

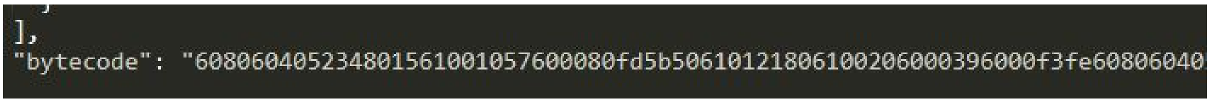
In this case, it says “Replacing “FirstNPrimes_V3””, because this is migrating a program which was deployed previously under the same name. It shows the new contract’s address, the transaction hash, the user account address used to make this transaction, the amount of gas used, the gas price, and also, the total cost of ether taken from the user account.

Before going on to show how to interact with this contract, the remaining examples are going to show how the waffle solidity programs and the vyper programs were compiled and deployed.



```
"abi": [
  {
    "inputs": [],
    "name": "getPrimes",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
      }
    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "num",
        "type": "uint256"
      }
    ],
    "name": "n_primes",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
],
```

Figure 3.22: Waffle compiled contract ABI

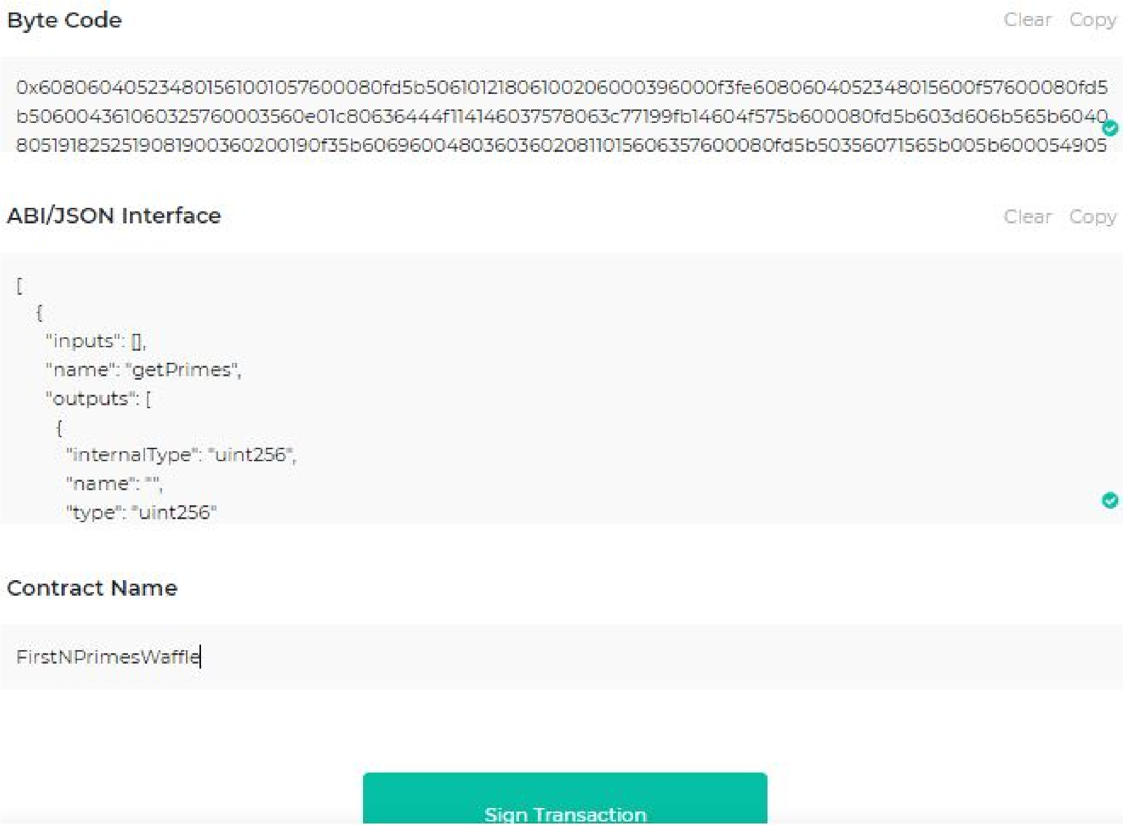


```
],
"bytecode": "608060405234801561001057600080fd5b506101218061002060003960 00f3fe60806040
```

Figure 3.23: Waffle compiled contract bytecode

Using the same problem, and being on the waffle directory, while having this specific program in the "src" folder, the user can run the command "npm build run", and waffle will compile every solidity program in the "src" folder and put the JSON files of the result of the compilation on the "build" folder. This JSON file possesses a lot of diferent information, but most importantly, it has the two necessary things needed to deploy, which are, the contract ABI and the bytecode. The contract ABI can be seen figure 3.22 and the bytecode on figure 3.23. This figure doesn't show the entirety of the bytecode, which in this case, is 642 characters long.

And in order to deploy this contract, because this was only the compilation, the interface MyEtherWallet assisted in the deployment of the smart contract into the Ethereum blockchain. In this case, the Infura node provider is not used. In figure 3.24, after connecting the MetaMask account with the website, the information given to MyEtherWallet is shown. And in figure 3.25, MetaMask shows how much this deployment transaction would cost. After confirming, a notification is received and clicked on.



Byte Code

```
0x608060405234801561001057600080fd5b506101218061002060003960 00f3fe6080604052348015600f57600080fd5
b5060043610603257600003560e01c80636444f114146037578063c77199fb14604f575b600080fd5b603d606b565b6040
805191825251908190036020 0190f35b60696004803603602081101560635760 0080fd5b50356071565b005b600054905
```

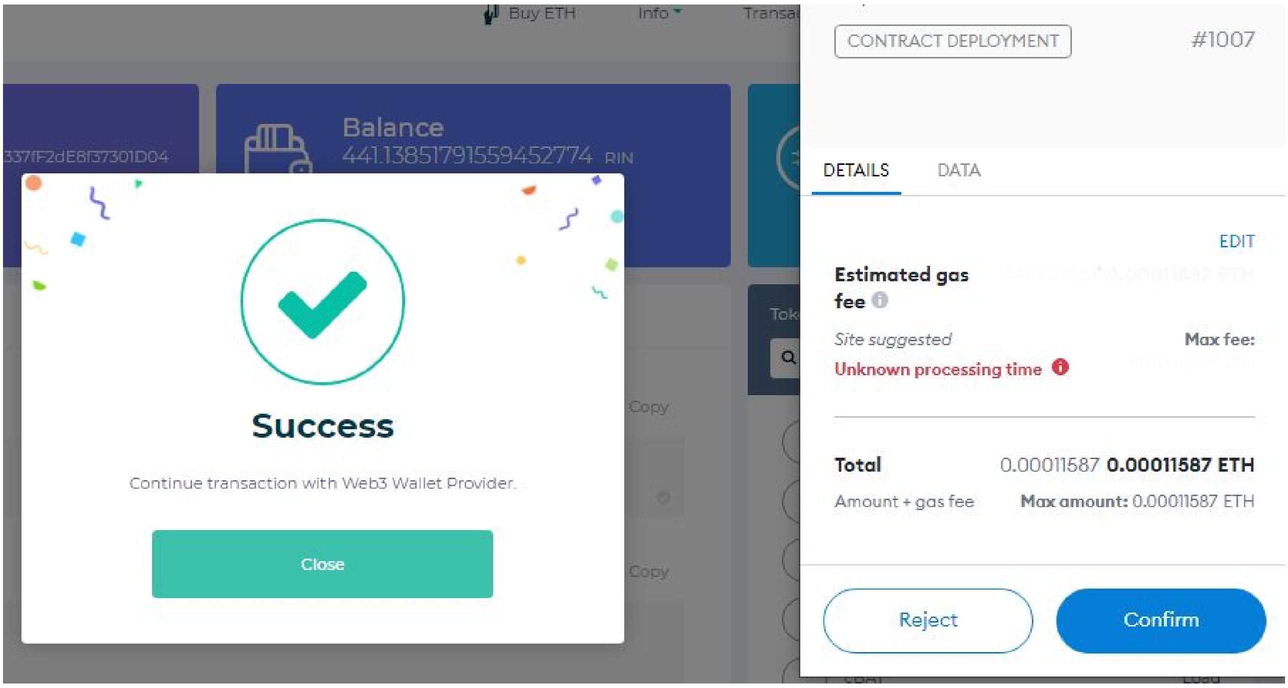
ABI/JSON Interface

```
[
  {
    "inputs": [],
    "name": "getPrimes",
    "outputs": [
      {
        "internalType": "uint256",
        "name": "",
        "type": "uint256"
```

Contract Name

FirstNPrimesWaffle

Figure 3.24: Contract ABI and bytecode on MyEtherWallet



337fF2dE8f37301D04

Balance 441.13851791559452774 RIN

Success

Continue transaction with Web3 Wallet Provider.

Estimated gas fee

Site suggested

Unknown processing time

Max fee:

Total 0.00011587 **0.00011587 ETH**

Amount + gas fee

Max amount: 0.00011587 ETH

Figure 3.25: Contract deployment on MyEtherWallet

This notification opens a page on Etherscan, where it has information about the transaction that just occurred. Which can be seen on figure 3.26.

Transaction Hash: 0xe2f024a05699980f2fa0f1ae26b618ef300cee7874018ff9627789611651bdea

Status: Success

Block: 9733957 1 Block Confirmation

Timestamp: 12 secs ago (Nov-30-2021 05:19:54 PM +UTC)

From: 0xb493fdf30382faac471337ff2de8f37301d04613

To: [Contract 0x644bbd9deeeaeb28b1d9312091535d5bdb332614 Created]

Value: 0 Ether ($0.00)

Transaction Fee: 0.000115869405167186 Ether ($0.00)

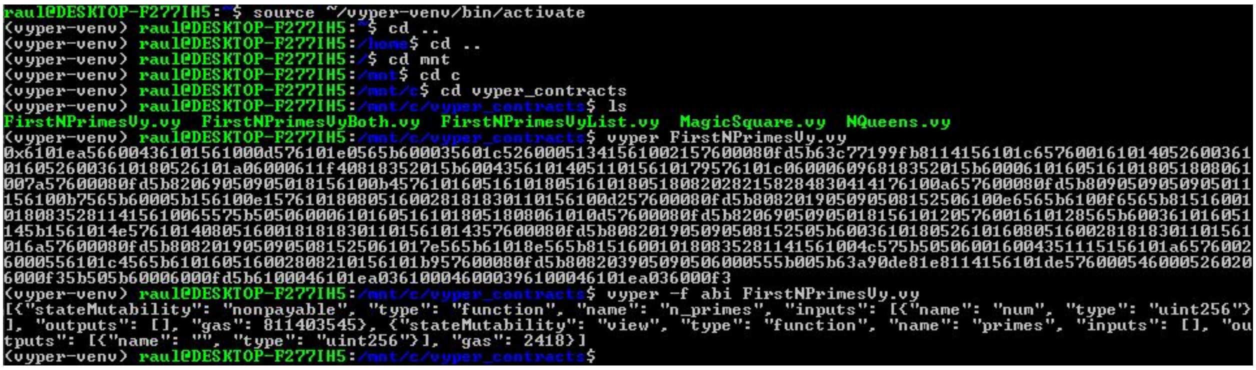
Gas Price: 0.000000001000020758 Ether (1.000020758 Gwei)

Gas Limit & Usage by Txn: 115,867 | 115,867 (100%)

Gas Fees: Base: 0.000018474 Gwei | Max: 1.000020758 Gwei | Max Priority: 1.000020758 Gwei

Figure 3.26: Etherscan information on contract deployment

Here, the transaction hash can be seen, along with, the block where it was inserted, the address of the account created, with the transaction fee showed earlier, the gas price paid to the miner and the gas used in this transaction.



```
raul@DESKTOP-F277IH5:~$ source ~/vyper-venv/bin/activate
(vyper-venv) raul@DESKTOP-F277IH5:~$ cd ..
(vyper-venv) raul@DESKTOP-F277IH5:/home$ cd ..
(vyper-venv) raul@DESKTOP-F277IH5:/$ cd mnt
(vyper-venv) raul@DESKTOP-F277IH5:/mnt$ cd c
(vyper-venv) raul@DESKTOP-F277IH5:/mnt/c$ cd vyper_contracts
(vyper-venv) raul@DESKTOP-F277IH5:/mnt/c/vyper_contracts$ ls
FirstNPrimesVy.vy  FirstNPrimesVyBoth.vy  FirstNPrimesVyList.vy  MagicSquare.vy  NQueens.vy
(vyper-venv) raul@DESKTOP-F277IH5:/mnt/c/vyper_contracts$ vyper FirstNPrimesVy.vy
0x6101ea56600436101561000d576101e0565b600035601c52600051341561002157600080fd5b63c77199fb8114156101c657600161014052600361
0160526003610180526101a06000611f40818352015b600435610140511015610179576101c060006096818352015b60006101605161018051808061
007a57600080fd5b82069050905018156100b45761016051610180516101805180820282158284830414176100a657600080fd5b8090509050905011
156100b7565b60005b156100e157610180805160028181830110156100d257600080fd5b808201905090508152506100e6565b6100f6565b81516001
01808352811415610065575b505060006101605161018051808061010d57600080fd5b8206905090501815610120576001610128565b600361016051
145b1561014e576101408051600181818301101561014357600080fd5b808201905090508152505b6003610180526101608051600281818301101561
016a57600080fd5b8082019050905081525061017e565b61018e565b815160010180835281141561004c575b505060016004351115156101a6576002
6000556101c4565b610160516002808210156101b957600080fd5b808203905090506000555b005b63a90de81e8114156101de576000546000526020
6000f35b505b60006000fd5b6100046101ea036100046000396100046101ea036000f3
(vyper-venv) raul@DESKTOP-F277IH5:/mnt/c/vyper_contracts$ vyper -f abi FirstNPrimesVy.vy
[{"stateMutability": "nonpayable", "type": "function", "name": "n_primes", "inputs": [{"name": "num", "type": "uint256"}
], "outputs": [], "gas": 811403545}, {"stateMutability": "view", "type": "function", "name": "primes", "inputs": [], "ou
tputs": [{"name": "", "type": "uint256"}], "gas": 2418}]
(vyper-venv) raul@DESKTOP-F277IH5:/mnt/c/vyper_contracts$
```

Figure 3.27: Vyper compiler bytecode and ABI

And finally, the last way used to compile programs, in this case, Vyper programs, was with the use of a private virtual environment and the vyper compiler. The compiler uses version 0.2.13. Important to note, that the Vyper programs were also tested on Truffle and Waffle and the results were the same. This private environment is used on Ubuntu. To activate, the command "source ∼/vyper-venv/bin/activate" is used. After putting the terminal on the directory where the vyper programs are, they are ready to be compiled.

By using "vyper FirstNPrimesVy.vy", it outputs the bytecode of the program, and if "vyper -f abi FirstNPrimesVy.vy" is used, the terminal shows the contract ABI. In figure 3.27, the respective outputs of both commands are shown.

These were the methods used to compile and deploy the contracts in Rinkeby. But, in order to actually interact with them and send arguments to make transactions, Etherscan was used. Using the contract address, one can access the previously deployed contracts in the blockchain. Here, though, the contracts had to be verified first, with their program code, before being able to make transactions with them. After this, the account is connected with the website and the transaction is made and verified on MetaMask, as demonstrated in figure 3.28. After waiting a bit and receiving the usual notification, an etherscan page with information on the transaction is shown, much like in figure 3.26.

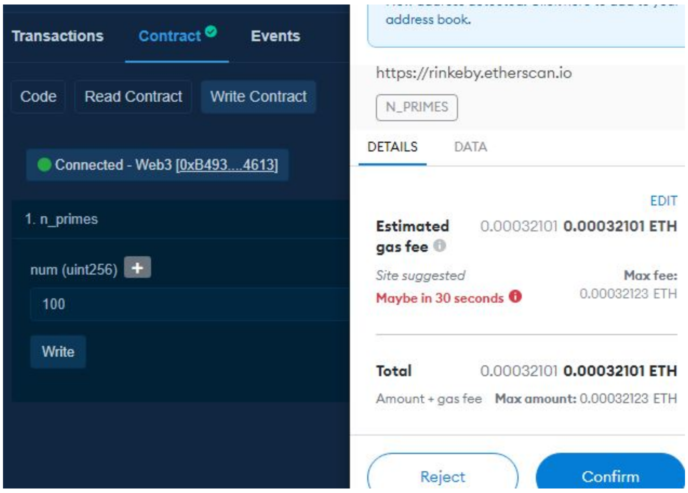
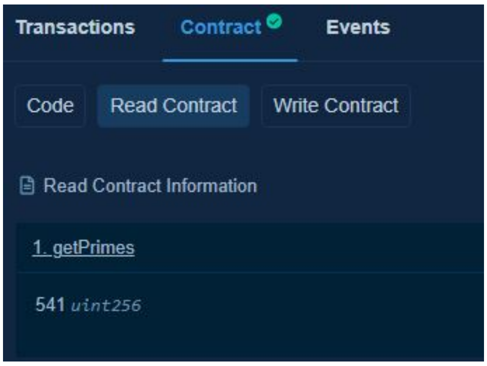


Figure 3.28: Etherscan contract transaction

If we go back, and go to the "Read Contract" tab, we can access the storage. After executing the transaction with argument 100, we can see in figure 3.29 the respective storage, 541, which is the correct output.



1. getPrimes

541 *uint256*

Figure 3.29: Etherscan contract storage

4

Performance Analysis

In this chapter, we discuss the parametric problems chosen to be used as a benchmark in the performance evaluation. These problems were written in the high-level smart contract languages already mentioned in section 2.3.1. The only language which was used more than once is Solidity, it was tested on the truffle environment with the optimization off and on, and it was also tested on the waffle library with optimization off and on. The Vyper language was evaluated in truffle, waffle and in a vyper compiler in a virtual environment, and the outcome was the same for all, therefore only one instance of this language will be displayed. Afterwards, the evaluation of the contracts created and deployed will be mentioned and represented with the various deployment and transaction gas values.

4.1 Parametric Problems

In this section, there will be a listing and explanation of the parametric problems chosen and written into smart contracts. The names of the programs are “FirstNPrimes”, “FirstNPrimesList”, “FirstNPrimesBoth”, “FirstNPrimesMap”, “NQueens” and “MagicSquare”.

The choice of the problems to represent and transform into smart contracts was made with the idea that an argument could be increased to see the rise of the gas values. With the programs “NQueens” and

"MagicSquare", the arguments increase in a linear manner, but in the other ones, which are to reach the N prime number with different approaches, the arguments are increased in a distinct manner, it starts with the argument 100, it increases by 10% of the previous value (the growth of a number is done by doing the respective rise by the percentage of the prior number rounded to the tenths, with one decimal point, but the arguments used are rounded to the integer), until reaching 214, then it increases by 25% until reaching 523, finally it increases by 50% thrice, ending with the final argument of 1766. This totals 16 arguments, not counting the gas value of the deployment of the contracts in the respective blockchains.

4.1.1 First N Primes

This first parametric problem chosen was one where the argument given is N, and the program, starting at 3, goes 2 by 2 checking if the number at the moment is a prime number or not. A prime number is a number in which it is only divisible by itself and 1, and no other value. Another variable starts at 3 every time a new number is inspected and going 2 by 2 until the square root of the current number, checks if the current number is divisible by this second one, if it is, moves onto the next number, while increasing a counter every time a prime number is found and goes through all first N prime numbers. This counter starts at 1, considering the number 2 right away. Every time a number is checked, it increments 2 and checks again. This program keeps running while the counter variable is smaller than the N argument. Initially, it was meant to save all the primes calculated until the Nth one in the storage of the contract, but unfortunately that resulted in a very big storage which sometimes was too large and it caused errors or the transactions were executed with an increased gas number which made the program overflow too soon. Ultimately, the decision made was to only save the last prime found in the state, which is the Nth prime number.

4.1.2 First N Primes (List)

In this problem, basically the same thing as the previous one occurs, but the difference here is that it starts with a list or array (depending of what data types each language has available) with the value 2 in it, and the counter of primes found at 1. It goes 2 by 2 much like the original problem, but in order to check whether or not the number is prime, it doesn't go 2 by 2 until the square root of the current one, instead it goes through all the values already inserted in the list, and checks if the number is divisible by every member; if it is divisible by at least one, another variable exists, starting at 0, to basically state whether the number is prime or not. After going through every element of the list, this variable is either 0 or 1, if it's 0, the current number inspected is not divisible by any of the numbers in the list and therefore is a prime number and gets added to the list, and the prime counter is also incremented; if it's 1, it found at least one element by which it is divisible by and therefore not a prime, in this case, the variable gets reset, which means set back to 0. Then, the variable where the current number was just examined gets incremented by 2 and the loop repeats until it finds the Nth prime.

4.1.3 First N Primes (Both)

This one is a combination of two previous programs. It goes 2 by 2 until the square root of the current number, but there's also a list/array of the primes found. Instead of having a second variable which goes 2 by 2, like the first program, it goes through the members of the list, like the second program, but with the difference that it comes to a stop until it reaches the square root of the current number, or finds a divisible number, and doesn't verify the entirety of the list of primes found at the moment. After this cycle, it does one last examination to determine if the loop stopped because it reached the square root of the current number or found a number by which it is divisible. This is done by a condition checking the divisibility by the value in the latest index verified in the loop. If the current one is divisible by this number, that's the reason the cycle came to an end and it means it's not a prime; if it isn't, that means it reached the square root of the number, getting to the conclusion that this specific number is prime.

The issue here is that in most of the high-level languages, which just happens to be all of the Tezos languages, it is impossible to access the index of a list. To circumvent this, in these languages, a map is used instead, with the id representing the index, and the keys representing the primes saved so far. Because of this change, a fourth problem was created, which is similar to "FirstNPrimesList" but with a map called "FirstNPrimesMap", just to compare the performance of the usage of a list versus a map in the Tezos languages. This also means that the Solidity and Vyper languages are not considered in this program due to a map never being used in these programs.

4.1.4 First N Primes (Map)

In this problem, like mentioned before, the program does exactly the same as "FirstNPrimesList" but does it with a map as opposed to a list. The one with a list makes use of a variable, which is 0 or 1, that like mentioned before, indicates whether the current number is prime or not. Along with this, in this program, there is an additional variable representing the index, used to check and increment each position in the map. Before incrementing by 2 and checking the next number, both this variable and the one that can only have 0 or 1 are reset to 0. Like mentioned before, Solidity and Vyper were not tested here.

4.1.5 N Queens

This program is slightly bigger than the previous ones. This problem is the classic one of placing N queens in a chessboard of size NxN, without any one of them attacking the others. Thinking of the rules of how the pieces move in chess, in the case of the queen, she can move horizontally, vertically and diagonally, basically she can move any number of squares in any direction [Ead16]. With this in mind, this program, while in the progress of placing the queens, shouldn't place a queen in the same line, column or diagonal of another. It takes as an argument a value N, which will determine the number of queens to place, as well as the length of the side of the board. In the state of the contract, the board is saved, which is represented by a map in the Tezos languages and by an array in the Ethereum ones. Chessboard are two dimensional, but a one dimensional data type is enough to save it, the index of the array or id of the map will represent the row and the value of this position is the column. For example, if N = 4, then an acceptable solution could be the following, where the values on the left are the keys or indexes, representing the row, and the ones on the right are the respective values, representing the column:

$$Queens : [0, 1, 2, 3] = [1, 3, 0, 2]$$

Ignoring the N's of 2, 3, and below and equal to 0, since it is impossible to find a solution in these cases, the program goes and places the queens one by one until having all of them on the board. It does this by putting a queen in the first position accessible, and then moves onto the next row, where it takes into account the restrictions of spaces based on the previous queens already placed. Every time a queen is placed on the board a variable increments, saving the number of queens already put down, and if a queen is removed, this variable decrements. When placing a certain queen, it goes through every row (until the current one), sees where the other queens are located, and blocks those specific columns, while also, blocking the two diagonals of each queen in the current row, as long as it doesn't go outside the bounds of the board. This means that if queen 1 (row 0) is in the second position (position 1) and we're placing queen 2 (row 1), it blocks the positions equal to the position of the other queen minus the distance between rows, which is 0 in this case, and equal to the same position plus the distance between rows, which is 2. It would also block position 1, since this column was already used. If this program finds a situation where it cannot place a queen in a single row, due to every position being blocked by the previous queens, it goes into backtracking mode, and goes back. What it does here, is it goes to the latest placed queen and moves it to the next available space, if the queen is already on the last available space in a row, it deletes her, decreases the counting of queens variable, and keeps going back until it finds a new spot for any queen.

In the case of N = 4, this is exactly what happens. It starts by putting the 1st queen in position 0 of row 0, it goes to the next row, blocking positions 0 and 1, and places the 2nd one on position 2. And then, when it tries to place the 3rd one, it notices that both columns of the previous queens and the diagonals of the second one, blocked the entire row, making it impossible to put a queen in this situation. To fix this, it backtracks and changes queen number two to position 3, then it tries again on the 3rd queen and this time there is a spot available on position 1, so it gets placed there. Now there's another issue, it tries to put the fourth and final queen but it is again impossible, the two previous queens block everything for this row and so the only thing left is to backtrack the board once again. It deletes the third queen since there isn't another spot after the current one to place (in this row), and then does the same thing for queen number two since it is in the last position. It tries the whole board again, going to the very first queen and changes it to position 1 from 0. It then puts the second one in position 3, since this is the only one available, goes to the 3rd queen, putting it on the first location free on the row, which is position 0, and finally the last queen has only one spot to be put in, and it is position 2, where it is placed, finishing the board and reaching a solution.

The program uses this algorithm for any N argument, with some requiring more backtracking iterations than others, which will be shown and discussed a bit ahead with the respective gas values.

Running this program with N = 8, the solution that it encounters, can be viewed in figure 4.1 in the form of a chessboard. With this visualization, it can be seen that none of the queens share the same row, column or diagonal.

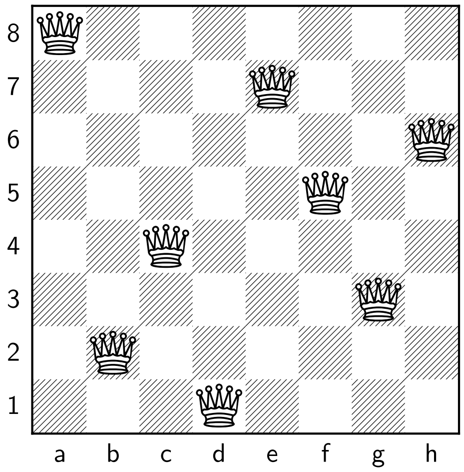


Figure 4.1: 8-Queens solution in a chessboard

4.1.6 Magic Squares

Out of all of the problems chosen to create into smart contracts, this one originated the largest program. It involves the creation of magic squares of order N, or in other words, a NxN magic square. This square, in order to be classified as a magic square, has to fulfill some restrictions. A magic square is a square divided into a square number of cells in which, natural numbers, all different, are arranged in such a way that the same sum is found in each row, each column and each of the two main diagonals [Ses19]. This sum, called the magic constant, let's say M, can be calculated using the following formula:

$$M = N \times \frac{N^2 + 1}{2}$$

This program, much as the others, takes the number N as an argument, this N defines the length of the side of the square to be made, since it designates its order. To construct the square, all of the numbers that are available to use range from 1 to N^2, and their usage in the square cannot be repeated. Just like the n-queens problem, in theory, it presents itself in a two dimensional environment, but a simple one dimensional data type is enough, for instance, using an array or map is sufficient to save the magic square. The square will be saved in the storage of the contract with one of the aforementioned variable types.

There are multiple different ways to tackle this problem, some being faster but restrictive and others acting much slower. Three distinct algorithms were previously tested and experimented with before reaching the one chosen to be evaluated in this paper.

The first one was a simple filling one, which had as many iterations as cells in the square, so time wise it was very quick. It always starts in the cell in the middle of the first row, putting the number 1, and moving in a diagonal manner, with one cell above and one to the right of the previous one, filling it with the numbers available in order 1 by 1. If it went out of bounds of the square it wrapped around to the opposite side, and if, after placing a number and moving to the next cell, it encounters an already filled cell, it ignores the diagonal movement and goes one cell down instead. This way of finding a solution is very fast, would always end up with a magic square but has two issues with it. First it barely increases the gas value when the N argument increases a great deal of times. This makes the program eventually reach an overflow on storage capacity before it even gets to exceed the gas limit. Second, this form of filling the square only works with odd N arguments, if the number is even, this strategy doesn't work.

Another method tested involved using permutations to find a solution. It would generate every permutation with the numbers from 1 to N^2, and in each instance created, it would have a different order of the sequence and would check the sums of the lines, columns and both the main diagonals to see if it was a magic square, if not, it moved on to the next permutation, if it was, it finished and saved the solution found in the state. The mechanism used in the generation of the permutations is called Heap's algorithm, which involved in obtaining all permutations only by interchanging two of the objects between each permutation and its predecessor [Hea63]. This method was more gas exhaustive than the previous one, but to an excessive degree. When using N equal to 3, the smallest magic square possible to achieve, it would be somewhat fast, but with N as 4, it would take an enormous amount of time, and therefore, gas would obviously overflow.

The last experimental method involved the usage of backtracking like it was used in the n-queens problem. This procedure travels from cell to cell filling it with the first number unoccupied, while keeping track of the numbers being used and the ones free to use. When it reaches the end of each row, it checks the sum and sees if it is equal to the magic constant. Does the same when it comes to the end of each column and also in the bottom corners, so it can check each of the main diagonals sums. Basically, it doesn't have to fill the entire grid with numbers to inspect every one of the different sums, saving a great deal of time. In other words, by the time it reaches the second row, it should be certain that the first row adds up to the magic constant. This method proved to be faster than the permutations one, showing capacity of reaching a solution when N is 3 and 4, but when it is bigger than 4, it takes far too long.

Finally, the one actually implemented and evaluated here is one where it uses a method of superposition. This specific sub-method of superposition, because there are several, was discovered by Leonhard Euler. The original paper was called "*De quadratis magicis*", it was written in Latin in 1776 and was published in 1849. The translated English version was published in 2004. In this paper, Euler basically explains that magic squares can be constructed with the usage of Latin letters in one square and Greek letters in another. These two squares' cells can then be paired up resulting in a square where each pair is different from each other, while following some restrictions and general rules depending if the N is even or odd [Eul04].

In the case of this program, instead of using Latin and Greek letters, numbers starting at 0 until reaching N-1 were used in both the initial squares, following the respective restrictions, the pairing of the two squares should be unique for each cell and, after the calculations, it should result in a magic square. So, in the beginning, the code starts by filling the first main diagonal of square A (A and B are the initial squares), and the second main diagonal of square B, in the following manner: if N is odd, it takes the value in the middle of the 0 to N-1 range, and fills the entirety of both diagonals with it; if N is even, it starts at 0, and fills 1 by 1, increasing this value by 1 for each cell, the diagonal of A, starting at the top left corner, and does the same thing with B, but starting with the top right corner instead. After this, it goes and fills the opposite diagonals of both of the squares, this time is the second main diagonal of A and the first of B. The way it achieves this, is starting at the bottom left corner of A and the top left of B, if N is odd, it simply goes from 0 to N-1 and fills them 1 by 1 in order, while ignoring the middle cell, since it is already filled. If N is even, it is slightly different, because for each position, it has to block and not place a number which already exists in that row or column, so there are no repetitions.

One important thing to note, is that throughout the filling of both these squares, if it notices there are two numbers placed in the same cell position in each square, it saves that pair to make sure it doesn't get repeated later. Another aspect of the program, is that it uses coordinates when dealing with squares, but since it is a one dimensional data type, in order to access each specific cell it calculates coordinate X multiplied by N plus the coordinate Y. If, for example, N is 3, and it wants to access the position with the coordinates (1,1), by doing the math, it results in the position 4 ($1 \times 3 + 1$), which is the index or key of whatever data type is being used.

After dealing with both main diagonals in both squares, it goes on to fill the rest of the squares. It accomplishes this with the use of backtracking. This backtracking is different than the one previously mentioned, because in the earlier one it was a backtracking method for the magic square itself, and it would check the sums when the time for it arose. In this case, it is filling the rest of the squares with the numbers ranging from 0 to N-1, but, with the restriction that they cannot repeat in the same row or column. And since the diagonals are already filled and each of their cells are distinct at this point, it is not necessary to check those. So, it does that, it goes and for each empty cell, it keeps track of the numbers already used in that respective row and column, and inserts one that is not in use yet. If it comes to a situation where there's an empty cell but all of the numbers in the range are unavailable, it goes back to the previous cell, removing the pair previously saved, since it is about to alter pairs made before.

Following the filling of the entirety of both A and B squares, it comes down to a simple calculation in each pair of cells to get to the final square. This determination is done by passing from cell to cell, on both squares at the same time, and multiplying the value in square A with N and adding the value in square B plus one. If all is done correctly, the final result should be a magic square without the need to check the sums throughout the whole program. So it can be illustrated in a more visual manner, the next two figures, will show how squares A and B would look like with this procedure and also their resulting square.

Since the process is slightly different depending if the N is odd or even, two examples are shown in both these cases. In figure 4.2, we can see the squares A and B and the resulting magic square when N is 3.

1	0	2
2	1	0
0	2	1

A

0	2	1
2	1	0
1	0	2

B

	4	3	8	15
	9	5	1	15
	2	7	6	15
15	15	15	15	15

Figure 4.2: Order 3 magic square solution

As can be seen, each pair forms a different number, if we take the values in coordinates (2,1), which are the numbers 2 in A and 0 in B, and doing the calculation explained beforehand, it results in 7 ($2 \times 3 + 0 + 1$), which is the number in the same position in the magic square on the right. The outer layer of the resulting square shows that each sum correctly equals to the intended magic constant. And the same thing occurs in figure 4.3, where we can see the squares A and B and the resulting magic square when N is 4.

0	3	1	2
2	1	3	0
3	0	2	1
1	2	0	3

A

1	3	2	0
2	0	1	3
0	2	3	1
3	1	0	2

B

	2	16	7	9	34
	11	5	14	4	34
	13	3	12	6	34
	8	10	1	15	34
34	34	34	34	34	34

Figure 4.3: Order 4 magic square solution

This method of superposition shown here, should in principle work for every integer with N greater or equal to 3 with the exception of 6. It is not certain if it works with 9 or higher because using the programs evaluated here, some reached the solution of $N = 7$, which was the maximum accomplished, and 8 worked in theory (done by hand).

4.2 Programs & Performance

Here it will be shown the gas values calculated with every language in each parametric problem, displayed in table form, and afterwards, references will be made to the source code structures of each program written and created in each of the languages.

All of the transactions done, were executed on the test networks Florencenet (see 3.5.1), for Tezos, and Rinkeby (see 3.5.2), for Ethereum. And it's important to keep in mind that each of these locations possess different gas limits on the transactions. Rinkeby has a gas limit of around 30.000.000 and Florencenet has a gas limit of 1.040.000. Because of this, the languages tested on Rinkeby, which are Solidity and Vyper (5 instances, since there's 4 for Solidity), are usually going to have bigger gas values than the Tezos' ones. This difference will be dealt with in chapter 5, but for now, comparisons with the languages in different blockchains will be made based on the arguments that they successfully executed.

The language Solidity is tested four times, with the usage of truffle with the optimization on and off, and in waffle, with the optimization on and off as well, these four instances are indicated with “Solidity T.ON”, “Solidity T.OFF”, “Solidity W.ON”, “Solidity W.OFF”, respectively. The numbers in the top row of the tables are the arguments used in each transaction. The word “Deploy” represents the gas value of the deployment of each corresponding contract into the blockchain.

In tables 4.1 and 4.2, all of the gas values calculated for the program FirstNPrimes (see 4.1.1) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), and the solidity variations.

Language	Deploy	100	110	121	133	146	161	177	195
SmartPy	4304	8474	8967	9484	10227	10979	11981	13001	14283
Liquidity	7275	19951	21679	23481	26094	28730	32226	35769	40240
PascaLIGO	4543	10848	11619	12420	13585	14756	16299	17856	19825
CameLIGO	4222	9299	9909	10544	11465	12394	13622	14866	16436
ReasonLIGO	4222	9299	9909	10544	11465	12394	13622	14866	16436
Archetype	9150	87639	96304	104995	117947	130643	146245	161319	180962
Solidity T.ON	128509	180091	203259	227496	262463	297824	344857	392653	452807
Solidity T.OFF	140161	193955	219206	245625	283744	322295	373590	425724	491335
Solidity W.ON	115867	171827	193777	216742	249867	283368	327929	373217	430209
Solidity W.OFF	130951	185682	209715	234862	271139	307830	356653	406279	468728
Vyper	158082	416710	476250	538842	628436	719400	840968	965104	1120656

Table 4.1: Gas Values of FirstNPrimes (1 of 2)

Language	214	268	334	419	523	785	1178	1766
SmartPy	15598	20035	26149	34663	46734	82709	148659	272039
Liquidity	44780	60151	81246	110477	151823	274442	498056	914684
PascaLIGO	21804	28508	37651	50232	67949	120095	214427	389023
CameLIGO	18025	23405	30772	40957	55342	97895	175292	319180
ReasonLIGO	18025	23405	30772	40957	55342	97895	175292	319180
Archetype	198427	258456	334351	429867	556302	887510	Overflow	Overflow
Solidity T.ON	514314	722101	1008038	1405584	1968788	3644617	6711528	12441571
Solidity T.OFF	558449	785188	1097298	1531357	2146422	3977160	7328702	13592278
Solidity W.ON	488494	685379	956328	1333066	1866804	3455069	6361954	11793329
Solidity W.OFF	532620	748457	1045579	1458830	2044429	3787603	6979119	12944027
Vyper	1284284	1825758	2574352	3620906	5107498	9554892	17740708	Overflow

Table 4.2: Gas Values of FirstNPrimes (2 of 2)

One important aspect to note first, is that the values demonstrated here, are not the speed of the programs, even though, the words “slower” and “faster” (and similar) will still be used. These values represent the gas (see 2.6) amount used in each transaction.

All of these programs source codes are listed at the very end of this paper in the appendix A. SmartPy (see A.1), Liquidity (see A.2), PascaLIGO (see A.3), CameLIGO (see A.4), ReasonLIGO (see A.5), Archetype (see A.6), Solidity (see A.7) and Vyper (see A.8).

With the values shown here, we can notice a few different things. It can be seen that SmartPy, CameLIGO and ReasonLIGO start off with smallest values in deployment, this is actually the only moment where SmartPy is not faster than every single other program. Throughout the whole testing of the rest of the arguments, SmartPy is the fastest program in the Tezos blockchain, while in Ethereum, Solidity compiled in waffle with the optimization on is the best one. In terms of the worst ones, Liquidity and Archetype are the slowest programs (of the Tezos' ones), with Archetype overflowing (passing the gas transaction limit), on the two last arguments, while Liquidity almost overflowed on the last one. And Vyper is the slowest (of the Ethereum languages), overflowing as well on the last argument, but not worse than Archetype. Solidity when optimized in both truffle and waffle, is better than their non-optimized counterparts. Another interesting thing to point out, is that both CameLIGO and ReasonLIGO have the same exact gas values. This is because, even though, they are slightly different programs, they compiled to the same Michelson code. This occurrence apparently happens on every single program, so to avoid repetition, it will only be described here.

In tables 4.3 and 4.4, all of the gas values calculated for the program FirstNPrimesList (see 4.1.2) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), and the solidity variations.

Language	**Deploy**	**100**	**110**	**121**	**133**	**146**	**161**	**177**	**195**
SmartPy	3948	25479	29697	34334	41971	50167	61241	72925	89827
Liquidity	6396	72909	86332	101090	125397	151488	186744	223946	277769
PascaLIGO	5233	34154	39773	45937	56068	66923	81565	96992	119278
CameLIGO	4213	26225	30513	35223	42975	51289	62515	74353	91470
ReasonLIGO	4213	26225	30513	35223	42975	51289	62515	74353	91470
Archetype	5259	43826	51390	59707	73319	87931	107639	128439	158458
Solidity T.ON	151441	2550110	3072782	3648162	4596839	5616209	6994941	8451065	10559400
Solidity T.OFF	166093	2601099	3133812	3720181	4686909	5725585	7130318	8613801	10761614
Solidity W.ON	141157	2187972	2635422	3127963	3939990	4812483	5992492	7238688	9042973
Solidity W.OFF	154729	2330170	2806549	3330866	4195246	5123894	6379749	7705936	9625923
Vyper	2725126	4724261	5687642	6747813	8495587	10373079	12911926	15592679	19473561

Table 4.3: Gas Values of FirstNPrimesList (1 of 2)

Language	**214**	**268**	**334**	**419**	**523**	**785**	**1178**	**1766**
SmartPy	106088	172148	276541	439877	708797	Overflow	Overflow	Overflow
Liquidity	329551	539958	872513	Overflow	Overflow	Overflow	Overflow	Overflow
PascaLIGO	140693	227536	364482	578394	930120	Overflow	Overflow	Overflow
CameLIGO	107929	174753	280267	445253	716755	Overflow	Overflow	Overflow
ReasonLIGO	107929	174753	280267	445253	716755	Overflow	Overflow	Overflow
Archetype	187364	304482	489317	778292	Overflow	Overflow	Overflow	Overflow
Solidity T.ON	12589233	20845536	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow
Solidity T.OFF	12829337	21239054	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow
Solidity W.ON	10780027	17844994	29024363	Overflow	Overflow	Overflow	Overflow	Overflow
Solidity W.OFF	11474232	18991095	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow
Vyper	23209209	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow

Table 4.4: Gas Values of FirstNPrimesList (2 of 2)

All of these programs source codes are listed in appendix A, as follows. SmartPy (see A.9), Liquidity (see A.10), PascaLIGO (see A.11), CameLIGO (see A.12), ReasonLIGO (see A.13), Archetype (see A.14), Solidity (see A.15) and Vyper (see A.16).

With this one, a lot of more overflows can be seen. This is because the programs reach an overflow on the gas limit in much earlier arguments than the previous problem. It makes sense, because instead of stopping when it encounters a number which the current one being checked is divisible by or when it reaches the square root of the current number, it keeps going through the entire list of primes found so far. In this one, just like the previous one, SmartPy is overall the best language, it doesn't exceed the limit with arguments equal or less than 523. The same thing occurs with the LIGO languages. One interesting thing happens in two contracts that is distinct from the previous values, which is, Archetype is better than Liquidity on this one, and is able to execute one more argument than Liquidity is, before overflowing. When speaking of Solidity and Vyper, it can be seen that they are worse than all of the Tezos' languages, Vyper is still the worst one (and overall as well in this program) and, this time, one of the Solidity instances was able to do an extra argument in comparison with the others. This instance, with waffle and optimization on, is and was proved to be better of the 4 before as well. Vyper has a much higher deploy value, probably because in this language, every variable needs to be initialized, and since it uses an array of 2000 positions (to save the primes found), every single one of those positions had to be initialized.

In tables 4.5 and 4.6, all of the gas values calculated for the program FirstNPrimesBoth (see 4.1.3) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), and the solidity variations.

Language	Deploy	100	110	121	133	146	161	177	195
SmartPy	5845	11350	11982	12620	13523	14410	15582	16782	18267
Liquidity	9528	26079	28155	30242	33250	36196	40049	43952	48840
PascaLIGO	6758	21253	23344	25529	28889	32332	36893	41600	47989
CameLIGO	5300	13300	14232	15169	16522	17847	19572	21314	23501
ReasonLIGO	5300	13300	14232	15169	16522	17847	19572	21314	23501
Archetype	7374	78058	85839	93637	105243	116613	130580	144081	161659
Solidity T.ON	167653	404473	456790	509646	584268	657634	754800	854598	977898
Solidity T.OFF	181885	421242	475786	530883	608721	685239	786541	890542	1019090
Solidity W.ON	155209	352667	397820	443437	507853	571184	655033	741133	847531
Solidity W.OFF	167065	367501	414671	462316	529642	595822	683438	773381	884565
Vyper	2755900	709770	800000	891053	1020029	1146696	1314649	1487041	1700271

Table 4.5: Gas Values of FirstNPrimesBoth (1 of 2)

Language	214	268	334	419	523	785	1178	1766
SmartPy	19694	24527	31079	39807	51694	85726	144328	249052
Liquidity	53456	69214	90385	118360	156372	264084	447674	772629
PascaLIGO	54068	77067	111206	161624	240047	507174	Overflow	Overflow
CameLIGO	25558	32582	41980	54358	71148	118503	198838	340369
ReasonLIGO	25558	32582	41980	54358	71148	118503	198838	340369
Archetype	177260	230865	298579	383636	495994	789525	Overflow	Overflow
Solidity T.ON	1096653	1498867	2045678	2775861	3771360	6630638	11569487	20422693
Solidity T.OFF	1142824	1562006	2131683	2892159	3928860	6905296	12044524	21253584
Solidity W.ON	949975	1296965	1768572	2398204	3256531	5721129	9977109	17604281
Solidity W.OFF	991571	1354125	1846834	2504542	3401167	5975387	10420067	18384499
Vyper	1905388	2601286	3547936	4812095	6536711	11492637	20056716	Overflow

Table 4.6: Gas Values of FirstNPrimesBoth (2 of 2)

All of these programs source codes are listed in appendix A, as follows. SmartPy (see A.17), Liquidity (see A.18), PascaLIGO (see A.19), CameLIGO (see A.20), ReasonLIGO (see A.21), Archetype (see A.22), Solidity (see A.23) and Vyper (see A.24).

In this one, it can be observed way less overflows than the previous one. It makes sense, because even though it uses a list of the primes found so far, it doesn't go all the way through but stops when it finds a number which a current one is divisible by or until it reaches the square root of the current number. This makes this problem a combination of the two previous ones, which in theory, it should make it overall better than the first one, FirstNPrimes, but it doesn't. This is because, although it should be faster, when comparing in terms of the gas usage, this one makes use of data types like arrays and maps and their respective constant access is more gas exhaustive than just using integer variables. This is not the case for every single language though. SmartPy and Liquidity, start off with worse gas values here than in FirstNPrimes but the increase in gas is lowered, which can be noticed at argument 1178 for SmartPy and 785 for Liquidity, from this point on, its more efficient in FirstNPrimesBoth than in FirstNPrimes. In the case of the LIGO languages, they are all worse than in FirstNPrimes, especially PascaLIGO which is extensively worse, growing so much more that it even overflows in the last two arguments. This is the first program so far where Liquidity is better than PascaLIGO, even if it starts better in the first half of arguments. Archetype, from the beginning to the end, is better here than in the first program. When mentioning the solidity instances, in general, they are worse than in the first program, but in this case, both of the Solidity ones tested in waffle are better than the two in truffle. And when compared to the rest of the languages, they are better than PascaLIGO, Archetype and Vyper. Vyper here, like in FirstNPrimesList, has a high deploy value because of the need to initialize the variables, especially the array of primes. And also, Vyper is worse here than in FirstNPrimes.

In tables 4.7 and 4.8, all of the gas values calculated for the program FirstNPrimesMap (see 4.1.4) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), with the exception of Solidity and Vyper.

Language	Deploy	100	110	121	133	146	161	177	195
SmartPy	4400	44093	52095	60900	75413	91002	112079	134332	166544
Liquidity	8703	166147	198494	234095	292796	355859	441145	531202	661591
PascaLIGO	6266	108718	129582	152529	190348	230954	285845	343782	427631
CameLIGO	5412	65547	77685	91029	113019	136621	168518	202177	250881
ReasonLIGO	5412	65547	77685	91029	113019	136621	168518	202177	250881
Archetype	5448	45957	53930	62699	77054	92469	113262	135212	166895

Table 4.7: Gas Values of FirstNPrimesMap (1 of 2)

Language	214	268	334	419	523	785	1178	1766
SmartPy	197548	323612	523018	835243	Overflow	Overflow	Overflow	Overflow
Liquidity	787103	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow
PascaLIGO	508318	836310	Overflow	Overflow	Overflow	Overflow	Overflow	Overflow
CameLIGO	297736	488155	789149	Overflow	Overflow	Overflow	Overflow	Overflow
ReasonLIGO	297736	488155	789149	Overflow	Overflow	Overflow	Overflow	Overflow
Archetype	197408	321067	516275	821529	Overflow	Overflow	Overflow	Overflow

Table 4.8: Gas Values of FirstNPrimesMap (2 of 2)

All of these programs source codes are listed in appendix A, as follows. SmartPy (see A.25), Liquidity (see A.26), PascaLIGO (see A.27), CameLIGO (see A.28), ReasonLIGO (see A.29) and Archetype (see A.30).

This problem was created to compare the usage of a map with a list in the Tezos languages. Overall, throughout the arguments, they are all slower in comparison with FirstNPrimesList. Every language overflows either one or more arguments sooner with the exception of Archetype, which has the smallest increase, that overflows in the same spot here as in FirstNPrimesList, but with a superior gas value. In the case of going through the whole gathering of primes found at any given moment, it seems like the Archetype language is the only one that gets only slightly worse, when using a map as opposed to a list, in comparison to the rest of the languages, which get a lot worse.

All of the transactions performed on the problems NQueens and MagicSquare have their arguments increase in a linear manner, one by one. The tables representing these two problems will have a "Deploy" column to represent the deployment of the contract, much like in the previous ones. In NQueens, it is impossible to have a solution to this problem when N is 2 or 3, but the gas values will be shown anyway, because the code deals with these cases by skipping them. In MagicSquare, when N is 1 or 2, there is no solution possible, but the values will be displayed anyway just to show the progression of the increase in gas, even though, in the programs it results in a square that isn't a magic square in these situations.

In tables 4.9 and 4.10, all of the gas values calculated for the program NQueens (see 4.1.5) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), and the solidity variations.

Language	Deploy	1	2	3	4	5	6	7	8
SmartPy	9045	9824	9812	9812	10080	9955	11706	10229	20055
Liquidity	37635	38522	38479	38479	39982	39109	49169	40487	94321
PascaLIGO	22448	23347	23314	23314	24438	23813	30443	24725	58038
CameLIGO	11500	12373	12358	12358	12730	12546	14967	12913	26345
ReasonLIGO	11500	12373	12358	12358	12730	12546	14967	12913	26345
Archetype	17485	18368	18355	18355	19114	18701	23781	19452	46659
Solidity T.ON	296829	26677	28758	31040	50046	42894	168852	65611	765727
Solidity T.OFF	322690	26825	28880	31162	51079	43464	175775	67215	803457
Solidity W.ON	284829	26652	28750	31032	49009	42431	160627	64012	720263
Solidity W.OFF	305295	26802	28869	31151	49592	42757	163380	64678	732766
Vyper	381598	55364	55011	55011	81526	66151	264466	94741	1206917

Table 4.9: Gas Values of NQueens (1 of 2)

Language	9	10	11	12	13	14	15
SmartPy	13525	21036	15483	45357	25374	321919	248490
Liquidity	58341	98413	68553	225641	120514	Overflow	Overflow
PascaLIGO	35517	59122	41117	131807	70113	941713	715818
CameLIGO	17380	27552	19996	60361	33331	433392	333328
ReasonLIGO	17380	27552	19996	60361	33331	433392	333328
Archetype	28469	48592	33539	112589	59283	834386	638231
Solidity T.ON	303160	841680	447870	2632581	1164175	28242737	20473306
Solidity T.OFF	316720	882771	468370	2763119	1220305	29395025	21351941
Solidity W.ON	286699	791299	422331	2472017	1093376	26831113	19389566
Solidity W.OFF	291089	803788	428483	2509032	1109042	27141288	19620648
Vyper	468497	1321380	692879	4082153	1822999	Overflow	Overflow

Table 4.10: Gas Values of NQueens (2 of 2)

All of these programs source codes are listed in appendix A, as follows. SmartPy (see A.31), Liquidity (see A.32), PascaLIGO (see A.33), CameLIGO (see A.34), ReasonLIGO (see A.35), Archetype (see A.36), Solidity (see A.37) and Vyper (see A.38).

On this case, we can see that, unlike the first N primes problems, the values don't increase every time N increases, it changes in a "up, down, up, down" pattern. This is because some arguments use more iterations of backtracking than others, looking at these values, it seems like it increases on even N's and decreases on odd N's. The values for N equal to 2 and 3 are also the same because of the skipping the code does, mentioned earlier. In the solidity instances, for some reason, the values for 2 and 3 are different, even though, the code executes the same set of instructions. It seems that the simple increase in the number of the argument is enough to alter the gas value for Solidity. Overall, they all execute successfully with arguments until 15, with the exception of Liquidity and Vyper, which overflow on the last two. SmartPy seems to be the best once again, with CameLIGO and ReasonLIGO not too far behind. Archetype here is faster than PascaLIGO, which is interesting. It seems the more complex the problem, the better Archetype is compared to PascaLIGO. Another thing observed is that PascaLIGO in this case had to use "for" cycles as well as "while's", which seems to have punished the gas values. In Solidity's cases, the waffle versions seem to be better than the truffle ones, and when the optimization is on, it's the best of the Ethereum's languages.

In table 4.11, all of the gas values calculated for the program MagicSquare (see 4.1.6) can be seen in all of the high-level languages referenced in the state of the art (see 2.3.1), and the solidity variations.

Language	Deploy	1	2	3	4	5	6	7
SmartPy	39540	40185	40276	40596	41108	43052	Impossible	401192
Liquidity	97548	98537	98768	99796	101380	107790	Impossible	Overflow
PascaLIGO	128863	129909	130243	132417	135334	147674	Impossible	Overflow
CameLIGO	49441	50342	50463	50954	51722	54771	Impossible	621052
ReasonLIGO	49441	50342	50463	50954	51722	54771	Impossible	621052
Archetype	64468	65433	65608	66494	67910	73765	Impossible	Overflow
Solidity T.ON	823567	29817	40345	69085	114593	257360	Impossible	Overflow
Solidity T.OFF	903234	30249	40903	70520	117296	265504	Impossible	Overflow
Solidity W.ON	795425	29653	39933	67584	111207	246727	Impossible	Overflow
Solidity W.OFF	854709	30016	40406	68445	112563	250139	Impossible	Overflow
Vyper	1733642	136617	143627	169649	212220	381032	Impossible	Overflow

Table 4.11: Gas Values of MagicSquare

All of these programs source codes are listed in appendix A, as follows. SmartPy (see A.39), Liquidity (see A.40), PascaLIGO (see A.41), CameLIGO (see A.42), ReasonLIGO (see A.43), Archetype (see A.44), Solidity (see A.45) and Vyper (see A.46).

In MagicSquare, it is not possible to find a solution with the arguments 1, 2, and also 6 (given the method used). 1 and 2 are still displayed just to show the progression of the gas even though the result is not a magic square, but 6 was not even considered. In this last program, probably the most exhaustive of them all, due to the fact of overflowing quite early, it can be seen that most of the languages cannot execute the argument 7 with success. The only ones that could were SmartPy, CameLIGO and ReasonLIGO. Like the previous one, Archetype is better than PascaLIGO, reinforcing the case that the more complex the problem is, and also the more "for" cycles it has, the worse its performance becomes. This also makes Liquidity better than PascaLIGO. Vyper is the worst again in the Ethereum's cases and between the solidity instances, it seemed, with the gas values that they were exhibiting, that they were capable of executing the argument 7, but this wasn't the case. It also appears that, just like previously, the waffle instances are better, and the best case is when the optimization is on in waffle.

After doing all these transactions on all of the programs constructed for these different problems, some conclusions can be reached. SmartPy is the fastest, not only of the Tezos' languages, but also overall. Liquidity, Archetype and Vyper are the worst overall, with PascaLIGO not too far ahead, being only better in smaller programs with not much complication. CameLIGO and ReasonLIGO are always behind SmartPy but not by too much. And the Solidity program compiled in waffle with the optimization on is the best overall of the Ethereum's languages.

5

Comparison

In this section, the gas values calculated and gathered in the previous chapter, inside the programs and performance section (see 4.2), will be demonstrated in a more visually friendly approach. This visualization is achieved using graphs, which will be constructed with the gas values shown beforehand. Making use of this way, the making of comparisons and conclusions between the languages and programs can be more easily done.

The graphs will be created in the following way. The x-axis will represent the arguments of the respective program, while the y-axis will represent the gas values calculated. Each line in the graph will represent the languages used, each one displayed with a different color and also the Tezos' languages will be shown with solid lines, while the Ethereum ones will be manifested in dashed lines, to distinguish each of the two blockchains. The dots in each line mark an argument of the problem defined at that location. And also, the value 0 in the x-axis is not a program argument but it represents the deployment of the contract.

To deal with the difference in the gas transaction limit, the graphs will be built with two y-axis. The Tezos' languages will share the left y-axis, which ranges from 0 to 1.040.000, and the Ethereum's languages will share the right y-axis, which ranges from 0 to 30.000.000.

When a certain program overflows on an argument or multiple arguments, in the graph, it will shown the line until the argument that it is capable of executing, and then cut off, ending the line there.

In figure 5.1, the graph for the first problem, FirstNPrimes (see 4.1.1), is demonstrated with all of the languages tested.

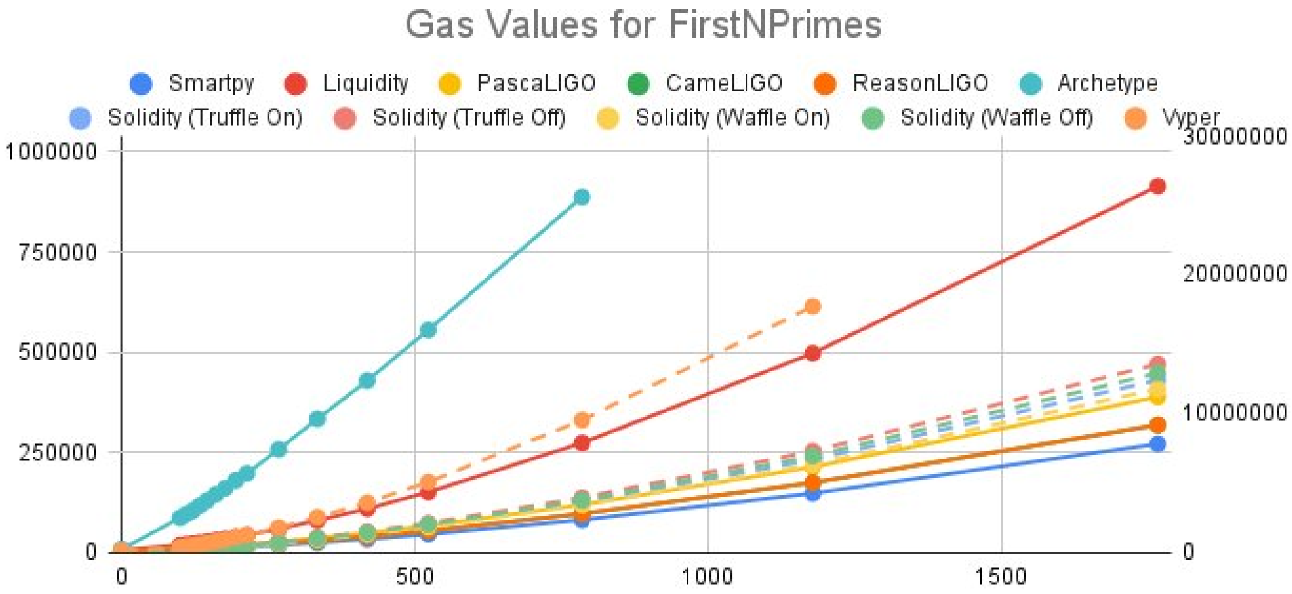


Figure 5.1: Graph for FirstNPrimes gas values

By observing the graph, we can more easily distinguish the differences between the languages in the two distinct blockchains. First, it can be seen that the first two, the worst ones, Archetype and Vyper are cut off before the last argument, this is because of overflowing in one argument, in the case of Vyper, and two, in the case of Archetype. Looking at the growth in the lines, it can be observed that Archetype is the worst, followed by Vyper and then Liquidity. Then afterwards, the four Solidity instances being very close together, with the order (from worst to best) Solidity (Truffle Off), Solidity (Waffle Off), Solidity (Truffle On) and Solidity (Waffle On). After this, it's PascaLIGO, ReasonLIGO (and CameLIGO on top of the same line because of having the same gas values), and ending with the best one, SmartPy.

This basically shows that the language SmartPy is the most efficient in this case, having the least growth of gas throughout the arguments while testing this particular problem. This does not mean that this language is the only one that should be used in cases like this, there are restrictions that apply to each language, and also every developer has its preference to what type of programming language they prefer to use or are more accustomed to. In addition to, a user can't create a smart contract in a high-level language in a specific blockchain with the intention to deploy in a distinct one. For example, a developer can't construct a smart contract in Liquidity, with the intent to use and interact with the Ethereum blockchain. Liquidity does not compile to EVM bytecode so this would be impossible.

In figures 5.2 and 5.3, the graphs for the second problem, FirstNPrimesList (see 4.1.2), are demonstrated with all of the languages tested.

This problem is shown in two different visualizations. On the first one, every argument is marked in the x-axis, but since in this problem, every single smart contract created ends up overflowing after the argument 523, another graph was constructed with the arguments more constrained, showing only until the last argument executed by the different programs, so that the lines are more easily viewed.

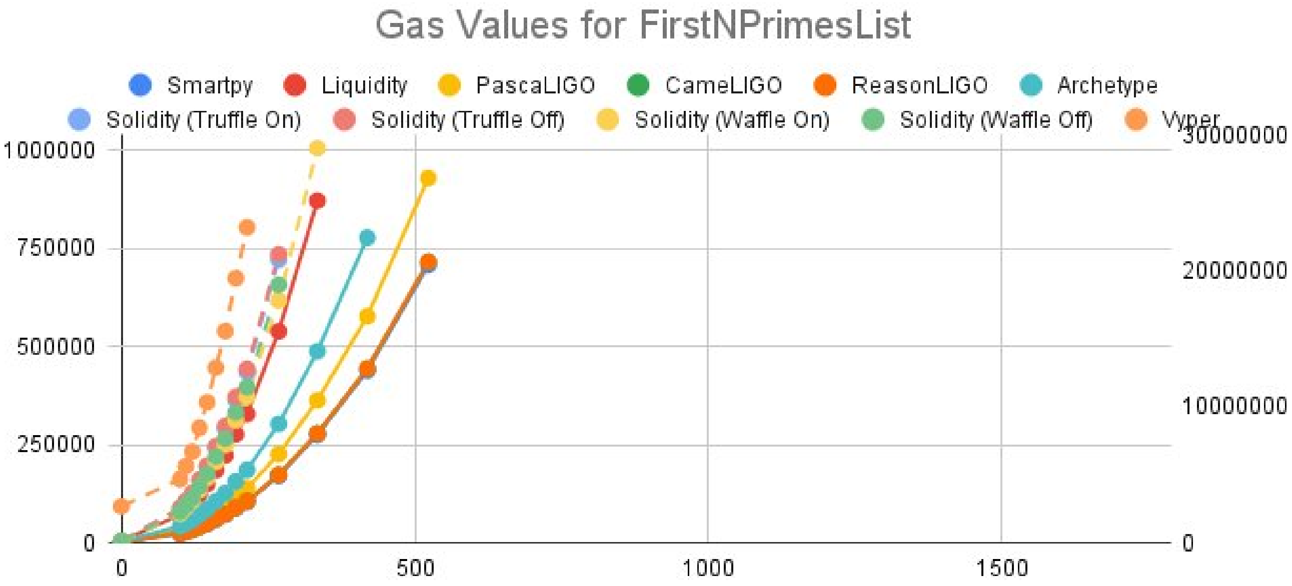


Figure 5.2: Graph for FirstNPrimesList gas values

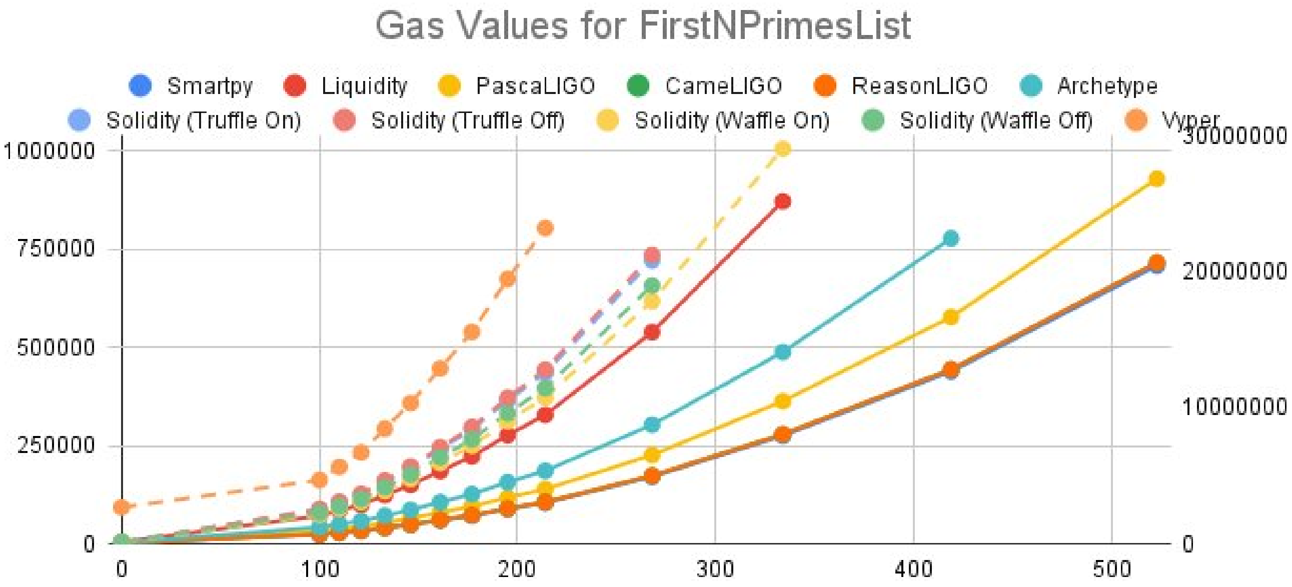


Figure 5.3: Graph for FirstNPrimesList gas values (constrained)

Since the other problems are shown with the arguments in their entirety, it was decided to the same here, and then because of the amount of overflows, a more focused version was created to more closely view the values represented in the graph. Here, Vyper is the worst, even with the deploy (argument 0) value much higher than the rest. This is because of the need to initialize every position in each array. After this, it goes, from worst to best, Solidity (Truffle Off), Solidity (Truffle On), Solidity (Waffle Off) and Solidity (Waffle On). And then, it's all of the Tezos' ones, starting with Liquidity, and then Archetype and

PascaLIGO. At the end, it's hard to notice, but the orange line is ReasonLIGO (with CameLIGO behind it having the same values) and the blue one barely visible, just under the orange line, is SmartPy, being the best one once again.

In this problem, it is observed that all of the Ethereum languages are worse than every language in Tezos. This could be because, even though, the problem goes through the entirety of the list or array of primes currently found, in the cases of Tezos, this is the only program where a list was used (with a map used in the others due to the inability to access a specific index in lists in these languages), and there are actually predefined ways of doing this in each language.

In figure 5.4, the graph for the third problem, FirstNPrimesBoth (see 4.1.3), is demonstrated with all of the languages tested.

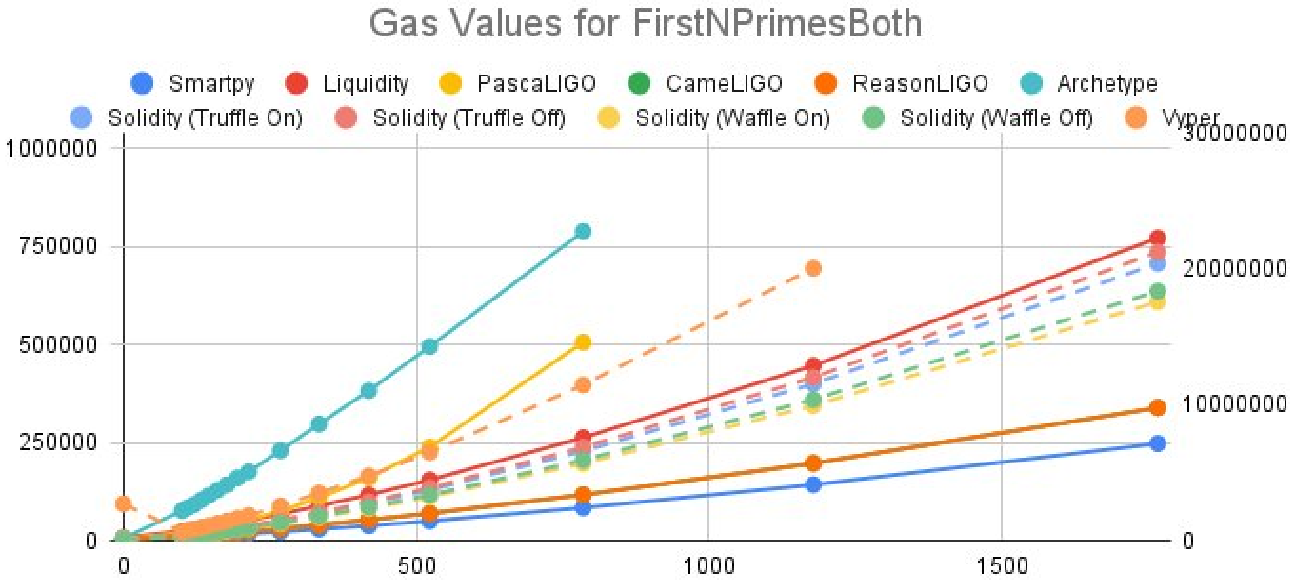


Figure 5.4: Graph for FirstNPrimesBoth gas values

In this graph, we can see more similarities with the first problem's graph, as opposed to the second problem. Starts off with Archetype as the worst one, but then something odd occurs. PascaLIGO is the second worse language, being much more gas exhaustive here than in the first graph (see 5.1). Vyper is not too far ahead either with the same deploy issue as before, but if we look closely, we can see that at the beginning PascaLIGO was actually slightly better than Vyper, but the growth in gas is much worse than Vyper, that it overcame it in gas amount. Also, it is not certain, but if the programs didn't overflow in the next couple of arguments, it's possible that the growth in the PascaLIGO line would become larger than in Archetype, potentially overcoming it and turning itself into the worst one overall. It is not certain as to why PascaLIGO got much worse in this case, it could be due to the fact of not dealing with maps very well.

The remainder of the graph seems to follow a similar trend as the first one with some differences. After Vyper, we have Liquidity, which is very close to the Solidity instances, where in the first problem it found itself farther apart. Then the Solidity instances are presented in a different order here, Solidity (Truffle Off), Solidity (Truffle On), Solidity (Waffle Off) and Solidity (Waffle On) (shown from worst to best), where Solidity with waffle optimization turned off is faster than Solidity with truffle optimization turned on. Then at the end, what was shown before its shown again here, ReasonLIGO (and CameLIGO) take second place and SmartPy appears as the best one.

In figures 5.5 and 5.6, the graphs for the fourth problem, FirstNPrimesMap (see 4.1.4), are demonstrated with all of the languages tested.

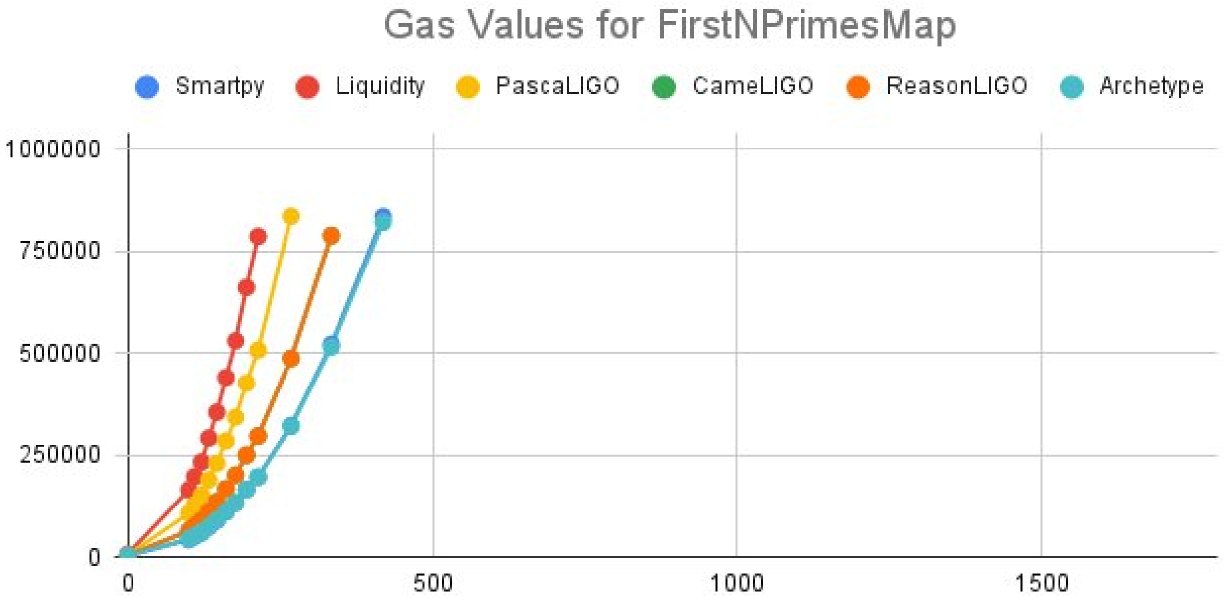


Figure 5.5: Graph for FirstNPrimesMap gas values

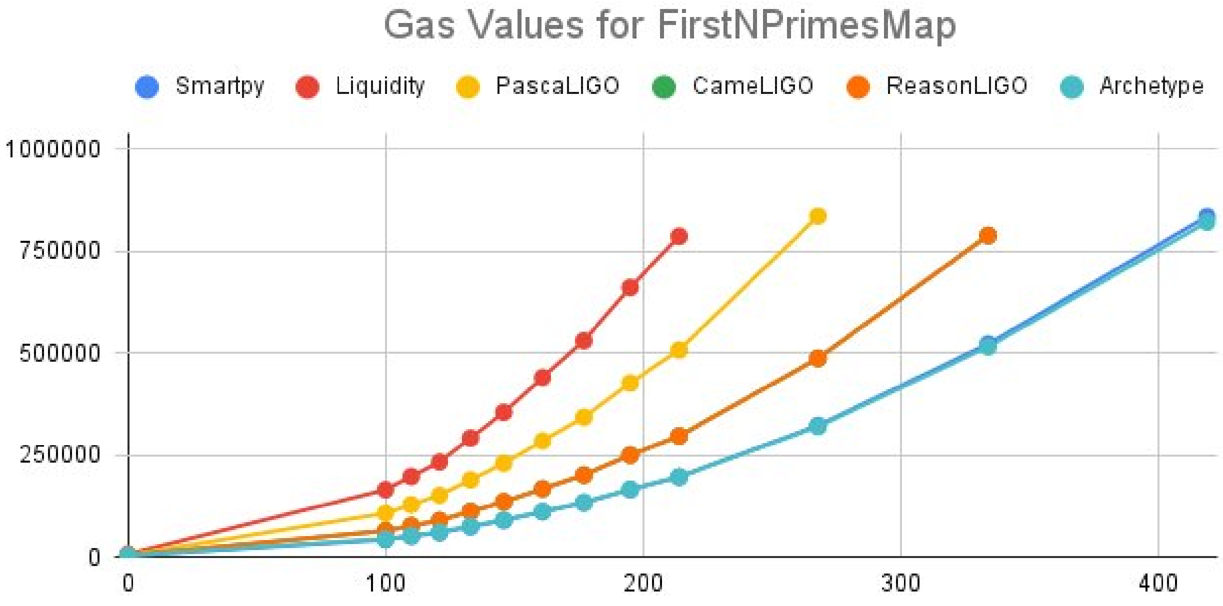


Figure 5.6: Graph for FirstNPrimesMap gas values (constrained)

In this one, the same procedure applied to FirstNPrimesList was done here as well, two graphs were made because the programs overflow too soon, in this case, they all exceed the gas limit past the argument 419. This one problem was basically created just to compare with FirstNPrimesList, on the Tezos' languages. The result is pretty straightforward, they all got worse, with the difference that Archetype in this case, is so much better than the others, to the point of even surpassing SmartPy and taking the lead this time.

The rest are in the same order, with ReasonLIGO and CameLIGO a bit farther away from SmartPy on this one.

In figures 5.7 and 5.8, the graphs for the fifth problem, NQueens (see 4.1.5), are demonstrated with all of the languages tested.

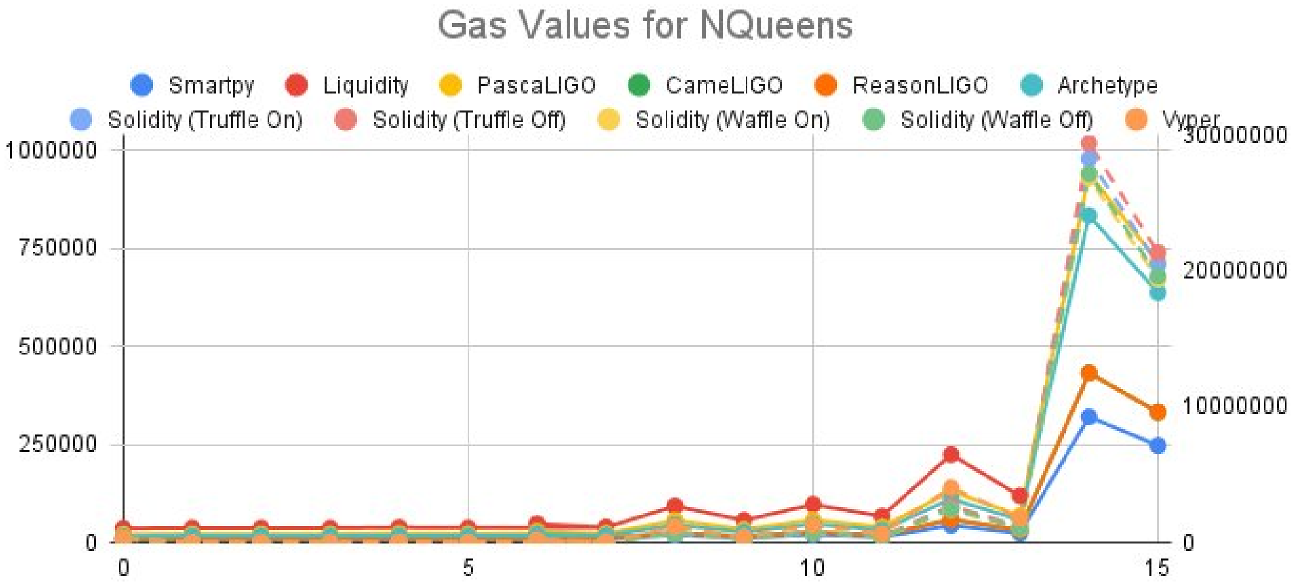


Figure 5.7: Graph for NQueens gas values

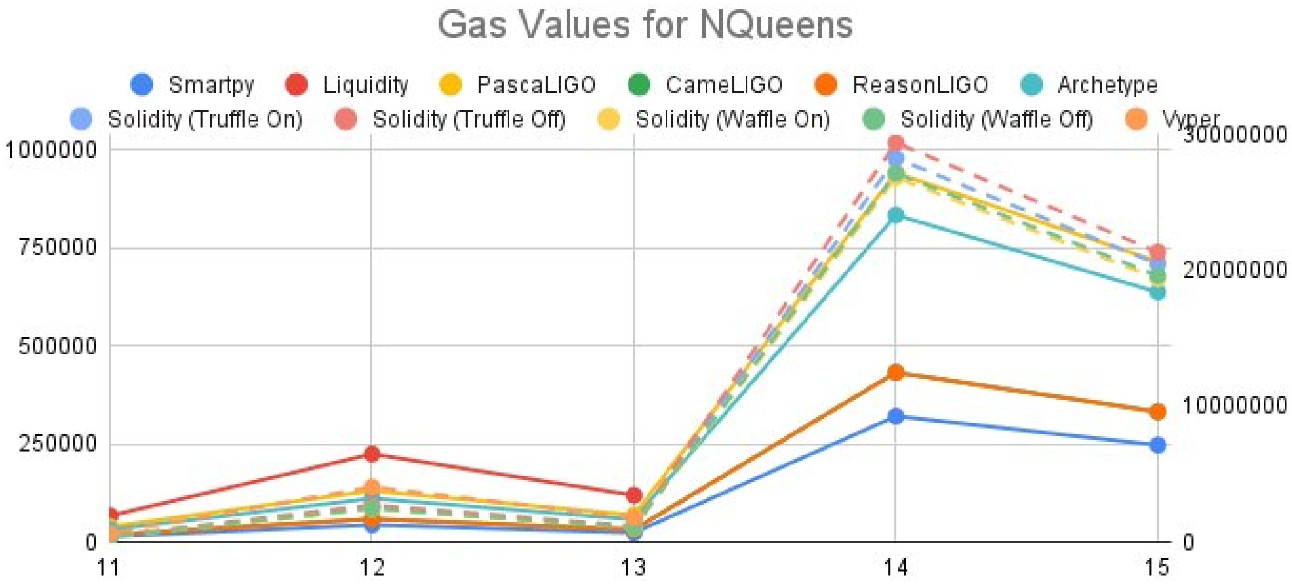


Figure 5.8: Graph for NQueens gas values (constrained)

In this problem, it was decided to display the gas values in two graphs. The first one has all the arguments used, but the second focuses more on the end side, having a better view on the fluctuation that occurs in the last few arguments. This one is a little more strange than the previous problems, due to the values gathered changing up and down constantly, which makes some sense given the backtracking method used in this problem and the fact that some arguments (odd ones), use less iterations of it than other ones (even ones). This one is slightly tougher to read since they are more jumbled together, but Liquidity is the worst one, followed by Vyper, which both overflow after the argument 13. The Solidity instances almost overflow but don't, with PascaLIGO mixed in them, but if noticed carefully, the line growth from argument 14 to 15 indicates it being worse than the Solidity instances, even though, it has a better value on the argument 14 than most of the instances. Then there's Archetype, then ReasonLIGO (with CameLIGO behind it of course), and ending with the usual winner, SmartPy.

In figure 5.9, the graph for the sixth and final problem, MagicSquare (see 4.1.6), is demonstrated with all of the languages tested.

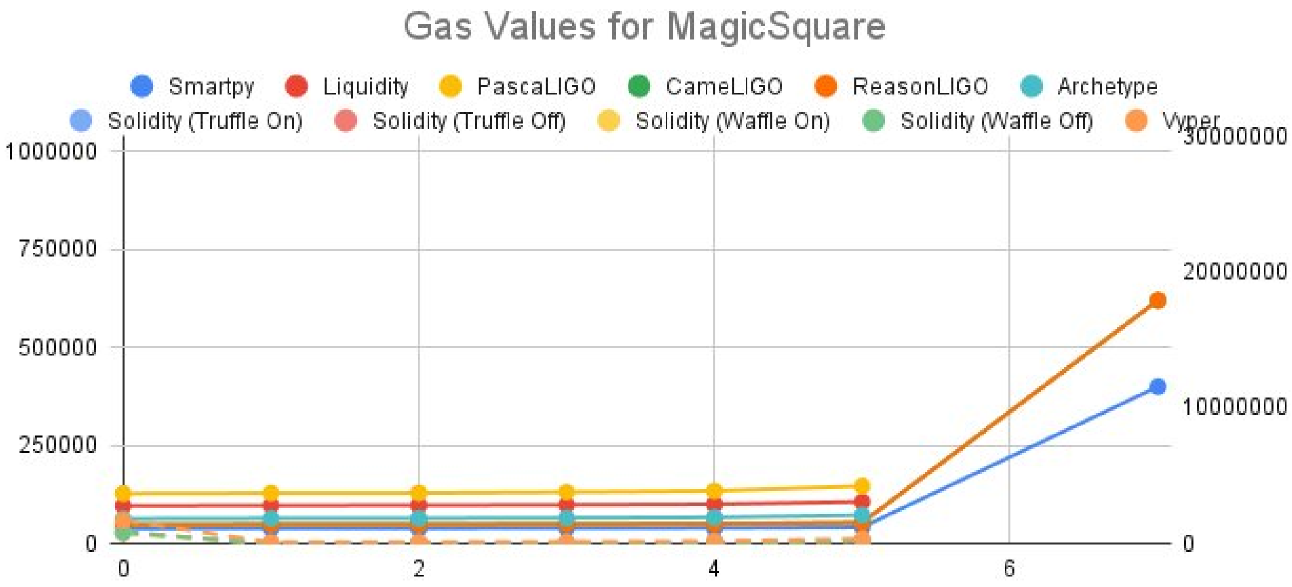


Figure 5.9: Graph for MagicSquare gas values

In this one, there is a very big rise from argument 5 to 7. Most of the programs have similar values in the first arguments, but then on the last argument, they all overflow with the exception of 3. The worst one here is PascaLIGO, followed by Liquidity and then Archetype. Then something strange transpires. The next languages are ReasonLIGO (with CameLIGO behind) and SmartPy, which are able to go and execute argument 7 successfully, but then it's Vyper and after, the Solidity instances, which do not and overflow. While in the process of creating the contracts and interacting with them after deploying, everything indicated that these were going to be able to execute the last argument, or at least, the Solidity ones were, since they were giving low gas results. But they weren't capable, and the blockchain gave back a transaction error saying that there was a gas overflow that passed the limit. Let's hope this was the actual program performing correctly and just reaching the limit normally and not stuck on something that was wrongfully coded, like an infinite loop. Just to be thorough, the Solidity code was tested using a debugging tool before deploying, but since the program is too big, it was impossible to do so with the last argument before it crashed.

In conclusion, in terms of the most efficient ones, SmartPy is the best, not only on Tezos but overall, and Solidity on waffle with optimization on is the best of the Ethereum ones. Vyper is the worst in Ethereum and overall. Liquidity, Archetype and PascaLIGO are the worst in Tezos, changing from one to another constantly. Liquidity has the most complicated code and is probably the worst of the three.

6

Conclusion

This chapter will essentially conclude this dissertation with an overview of everything which was discussed and analyzed, and also conclusions about the performance evaluations made will be mentioned. Afterwards, a segment describing what could make this paper better will be represented, with more information in conjunction with some potential modifications and additions to the testing to have more results of performances to be used for comparison.

6.1 Overview

This dissertation strove to research distinct high-level languages in two different blockchains, Tezos and Ethereum, after acquiring the knowledge of what these blockchains consist of and how they work. After the understanding of each different language, and how to interact and deploy smart contracts into test networks of both of the ledgers, the objective was to test these equivalent contracts and arrange every gas value obtained, representing the performance, so that in the end, they could be listed and then compared with every other language also utilized and reach conclusions with the evaluations done.

The groundwork part of this dissertation proved fundamental to the better understanding of every single concept interacted with and relevant to this respective work. Learning what is and how a blockchain works, and also what do smart contracts consist of and how useful these agreements are, helped with the choosing of the two blockchains, Tezos and Ethereum to be tested here.

Reading and understanding the documentation of every language used, and also the tools available to create the contracts and also deploy these understandings, in the respective languages, assisted in the creation of each smart contract in each different language.

Parametric problems chosen to be turned into smart contracts to be evaluated were described in detail with emphasis on the most important aspects of each of the problems. Arguments designed to test each program in transactions were listed and their growth explained.

Transactions were carried out on every contract integrated into the blockchains and the respective gas values registered. Afterwards, these numbers were displayed, discussed and also transformed into graphs for a better understanding of the values compared to each other.

The conclusions made with these comparisons is that the language SmartPy was the superior one, not only on the Tezos blockchain, but also overall. With Solidity compiled using waffle with the optimization on was the best for Ethereum. ReasonLIGO and CameLIGO (since they both always compiled to the same Michelson code), were tied for second best after SmartPy. Vyper was the worst language on Ethereum and overall. Archetype and PascaLIGO were really awful in some cases but not in others. And Liquidity was the worst one in the Tezos languages, although being slightly hard to tell. This information gives a better understanding of how to choose a high-level language. So, for example, in the case of a developer preferring or being more familiar to programming in the OCaml language and wanting to interact with the Tezos blockchain, they would have the choice between Liquidity and CameLIGO (since these two are the closest to OCaml), and these results show that CameLIGO would've been more gas efficient.

6.2 Future Work

Despite having gathered these values corresponding to the performance of each of the languages experimented, there were some issues encountered and also there's always potential expansions to increase what was achieved in this dissertation, such as the gathering of data, the ledgers used, or even the tools and environments employed here.

One obvious case is having more languages, which increases information gathered about the performances by having more data available. There is a language called "Yul", which works as a intermediate language for Ethereum and is used as a good target for high-level optimization for the EVM, which was a potential extra language to have here, but the way the contracts are interacted with in Ethereum is by deploying them using the bytecode and contract ABI and then confirm the code of the contract using a tool called Etherscan. This language compiles its contracts with bytecode and contract ABI but etherscan does not have "Yul" as one of its language options, only Solidity and Vyper. The same thing occurred with "Yul+", which was a low level extension to "Yul". There was another one called "Fe", which was an easy to learn language, inspired by Python and Rust, but it's in its early stages, which means it wouldn't be possible to confirm its code on Etherscan. There were other cases explored, these ones on Tezos, such as "fi", which apparently did not have loops in the language so it was impossible to use. "Morley" and "Indigo eDSL" were also attempted but errors occurred in their installations, therefore there was no way of compiling contracts in these languages. All of these languages listed here could potentially be future work if they get more development and more tools to engage with them in a more later time, and also, the languages tested here, could also be updated to more recent versions.

Another aspect to modify in the future is the use of newer protocols of the blockchains. In Ethereum, there hasn't been a new one since the development of this paper but in the case of the Tezos blockchain, the test network used had the protocol Florence, which, has of the end of November 2021, Florence is no longer being maintained, due to the fact of there being two new test networks available, "Granadanet" and "Hangzhounet", with the protocols "Granada" and "Hangzhou" running respectively, and the "Granada" protocol came with several improvements to the performance which resulted in a dramatic reduction of the gas costs in Michelson, observing a decrease of a factor of three to six in the gas consumed in the execution of already deployed contracts [TAF21a]. Out of curiosity, the program MagicSquare written in Archetype was tested in "Granadanet", and was able to successfully execute the argument 7, as opposed to the one in Florencenet, which couldn't.

Another potential extension to this paper, is the use of more smart contract friendly blockchains. Tezos and Ethereum were chosen due to the fact of having various tools, languages and environments to develop smart contracts in, but more blockchains could also apply to this fact and could join the original two in the evaluation of the performance of their smart contracts.

Lastly, the metric used to measure the performance. In this dissertation it was gas, because this number is shown in every single transaction and deployment of each contract. This could be increased to have more metrics to evaluate, such as the size of the storage of the contract, which the Tezos blockchain already shows but the Ethereum one doesn't. And also, although not sure how, the metric of time could be studied as well. It can't be through the time a transaction takes to complete, because by the end of it, when the client shows you that it was validated, this corresponds to the whole block as well and not just the specific transaction. Potentially, it could be done by creating a fork of the network being tested into a private one, so that every transaction is local only, but it's not sure if it would work to study the time measure, and if it does it would require a lot of effort.

In summary, the addition of new high-level languages and usage of the ones experimented on a more newer version, the update of the protocols used in the networks, in conjunction with, increasing the number of blockchains to be used as a testing ground, and also, the expansion of different metrics to use in the evaluation of the performance of the contracts, are all potential extensions to this dissertation.

Bibliography

[AA19] Shikah J. Alsunaidi and Fahd A. Alhaidari. A survey of consensus algorithms for blockchain technology. In *2019 International Conference on Computer and Information Sciences (ICCIS)*, pages 1–6, 2019.

[ABT19] Victor Allombert, Mathias Bourgoin, and Julien Tesson. Introduction to the tezos blockchain. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 1–10, 2019.

[AME19] Cc Agbo, Qusay Mahmoud, and J. Eklund. Blockchain technology in healthcare: A systematic review. *Healthcare*, 7:56, 04 2019.

[BAI$^+$18] Joseph J. Bambara, Paul R. Allen, Kedar Iyer, Rene Madsen, Solomon Lederer, and Michael Wuehler. *Blockchain: A Practical Guide to Developing Business, Law, and Technology Solutions*. McGraw-Hill Education, 2018.

[Bas20] Imran Bashir. *Mastering Blockchain : a deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more*. Packt Publishing, 2020.

[BMM$^+$20] Stefano Bistarelli, Gianmarco Mazzante, Matteo Micheletti, Leonardo Mostarda, Davide Sestili, and Francesco Tiezzi. Ethereum smart contracts: Analysis and statistics of their source code and opcodes. *Internet of Things*, 11:100198, 2020.

[But13] Vitalik Buterin. Ethereum whitepaper - a next-generation smart contract and decentralized application platform. `https://ethereum.org/en/whitepaper/`, 2013. Last accessed 25 November 2021.

[CNB21] Taylor Locke CNBC. Ethereum just hit an all-time high of above $4,400 after a recent upgrade. here's what to know. `https://www.cnbc.com/2021/10/28/what-to-know-about-ethereum-altair-upgrade-and-proof-of-stake.html`, October 2021. Last accessed 25 November 2021.

[Ead16] James Eade. *Chess for Dummies*. John Wiley & Sons, Inc., 2016.

[Eul04] Leonhard Euler. On magic squares. `https://arxiv.org/abs/math/0408230`, 2004. Last accessed 18 November 2021.

[Gat17] Mark Gates. *Blockchain: Ultimate guide to understanding blockchain, bitcoin, cryptocurrencies, smart contracts and the future of money.* CreateSpace Independent Publishing Platform, 2017.

[Goo14] L. M Goodman. Tezos - a self-amending crypto-ledger, 2014.

[Hea63] B. R. Heap. Permutations by Interchanges. *The Computer Journal*, 6(3):293–298, 11 1963.

[HZL$^+$21] Bin Hu, Zongyang Zhang, Jianwei Liu, Yizhong Liu, Jiayuan Yin, Rongxing Lu, and Xiaodong Lin. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*, 2(2):100179, 2021.

[Inf19] Roberto Infante. *Building Ethereum Dapps: Decentralized applications on the Ethereum blockchain*. Manning Publications, 2019.

[Inf21] Infura. Infura - documentation. `https://infura.io/docs`, 2021. Last accessed 29 November 2021.

[Lau17] Tiana Laurence. *Blockchain for Dummies*. John Wiley & Sons, Inc., 2017.

[LNB$^+$19] Roben Castagna Lunardi, Henry Cabral Nunes, Vinicius da Silva Branco, Bruno Hugentobler Lippert, Charles Varlei Neu, and Avelino Francisco Zorzo. Performance and cost evaluation of smart contracts in collaborative health care environments. *CoRR*, 2019.

[Met21] MetaMask. Metamask - about. `https://metamask.io/`, 2021. Last accessed 29 November 2021.

[MSA20] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. Performance evaluation of permissioned blockchain platforms. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, pages 1–8, 2020.

[nI21] npm Inc. npm - about. `https://www.npmjs.com/`, 2021. Last accessed 28 November 2021.

[OF21] Node.js OpenJS Foundation. Node.js - about. `https://nodejs.org/en/about/`, 2021. Last accessed 28 November 2021.

[OTJA21] Damilare Peter Oyinloye, Je Sen Teh, Norziana Jamil, and Moatsum Alawida. Blockchain consensus: An overview of alternative protocols. *Symmetry*, 13(8), 2021.

[Sav16] Alexander Savelyev. *Contract Law 2.0: «Smart» Contracts As the Beginning of the End of Classic Contract Law*. PhD thesis, National Research University Higher School of Economics, 2016.

[Sch18] Ruben Schulpen. Smart contracts in the netherlands. Master's thesis, Tilburg University, 2018.

[SDP18] Bikramaditya Singhal, Gautam Dhameja, and Priyansu Panda. *Beginning Blockchain*, chapter How Ethereum Works, pages 219–266. Apress, 2018.

[Ses19] Jacques Sesiano. *Magic Squares: Their History And Construction From Ancient Times To AD 1600*. Springer International, 2019.

[Sui21] Truffle Suite. Truffle overview. `https://trufflesuite.com/docs/truffle/overview`, 2021. Last accessed 28 November 2021.

[SWS20] Ellis Solaiman, Todd Wike, and Ioannis Sfyrakis. Implementation and evaluation of smart contracts using a hybrid on- and off-blockchain architecture. *Concurrency and Computation: Practice and Experience*, 33(1):e5811, 2020.

[TAF21a] Nomadic Labs Tezos Agora Forum. Announcing granada. `https://forum.tezosagora.org/t/announcing-granada/3183`, 2021. Last accessed 24 November 2021.

[TAF21b] Nomadic Labs Tezos Agora Forum. Florence: Our next protocol upgrade proposal. `https://forum.tezosagora.org/t/florence-our-next-protocol-upgrade-proposal/2816`, 2021. Last accessed 29 November 2021.

[TDR21] Nomadic Labs Tezos Developer Resources. How to get tezos. `https://tezos.gitlab.io/introduction/howtoget.html`, 2021. Last accessed 28 November 2021.

[Waf20] Waffle. Waffle documentation. `https://ethereum-waffle.readthedocs.io/en/latest/`, 2020. Last accessed 28 November 2021.

[Woo21] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. `https://ethereum.github.io/yellowpaper/paper.pdf`, 2021. Last accessed 27 November 2021.

[ZWM21] Matt Zand, Xun Wu, and Mark Anthony Morris. *Hands-On Smart Contract Development with Hyperledger Fabric V2: Building Enterprise Blockchain Applications*. O'Reilly Media, 2021.

A

Source Codes

In this appendix, the source code of every single smart contract program created will be listed. Programs were written in all 8 languages mentioned previously (see 2.3.1), in all of the parametric problems (see 4.1), with the exception of “FirstNPrimesMap” (see 4.1.4), where only 6, the Tezos’ ones, were tested. The only aspect that differs from the Solidity programs tested in truffle and waffle is the version number. It’s 0.5.0 in truffle and 0.6.2 in waffle. Another thing to note is that every listing of code shown is displayed with the Solidity language style, since this is the only language which had a predefined style. And also, a repository of all of these programs can be found on `https://github.com/RaulOliveira20/dissertation_smart_contracts`.

```
import smartpy as sp

class FirstNPrimes_V3(sp.Contract):
    def __init__(self):
        self.init(last_prime = sp.int(0))

    @sp.entry_point
    def n_primes(self , num):
        p = sp.local("p", 1)
```

```
        i = sp.local("i", 3)
        x = sp.local("x", 3)
        n = sp.local("n", num)

        sp.while p.value < n.value:

            sp.while (i.value % x.value != 0) & (x.value * x.value <= i.value):
                x.value += 2

            sp.if (i.value % x.value != 0) | (i.value == 3):
                p.value += 1

            x.value = 3

            i.value += 2

        sp.if (n.value <= 1):
            self.data.last_prime = 2
        sp.else:
            self.data.last_prime = i.value - 2
```

Listing A.1: FirstNPrimes in SmartPy

```
type storage = int

let%entry n_primes (num : int) _ =

  let p = 1 in
  let i = 3 in

  let _, i =
    Loop.left (fun (p, i) ->

          let x = 3 in
          let x, i =
            Loop.left (fun (x, i) ->

                let (_, rem) = match i / x with
                  | Some qr -> qr
                  | None -> failwith "division by 0 impossible" in

                if (rem <> (0 : nat) & x * x <= i) then (Left (x + 2), i)
                else (Right x, i)
            ) x i
          in

          let (_, rem) = match i / x with
                  | Some qr -> qr
                  | None -> failwith "division by 0 impossible" in

          let p =
            if (rem <> (0 : nat) || i = 3) then p + 1 else p + 0
          in

          if (p < num) then (Left p, (i + 2))
          else (Right p, i)
    ) p i
  in

```

```
  let i =
    if (num <= 1) then 2 else i
  in

  ( [], i)
```

Listing A.2: FirstNPrimes in Liquidity

```
type storage is int

type parameter is
  Primes of int

type return is list (operation) * storage

function firstnprimes (const num : int) : storage is
  block {
      var p : int := 1;
      var i : int := 3;
      var x : int := 3;

      while p < num block {
        while (i mod x =/= 0n) and (x * x <= i) block {
            x := x + 2;
        };

        if i mod x =/= 0n or i = 3 then p := p + 1 else skip;

        x := 3;

        i := i + 2;
      };

      if num <= 1 then i := 2 else i := i - 2

  } with i

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    Primes (n) -> firstnprimes (n)
  end)
```

Listing A.3: FirstNPrimes in PascaLIGO

```
type storage = int

type parameter =
  Primes of int

type return = operation list * storage

let firstnprimes (num : int) : storage =
    let p : int = 1 in
    let i : int = 3 in
    let x : int = 3 in

```

```
    let rec while_one (num, p, i, x : int * int * int * int) : int =
        let rec while_two (i, x : int * int) : int =
            if i mod x <> 0n && x * x <= i then while_two(i, x + 2) else x
        in
        let x : int = while_two (i, x) in

        let p : int = if i mod x <> 0n || i = 3 then p + 1 else p in

        let x : int = 3 in

        if p < num then while_one(num, p, i + 2, x) else i
    in

    let i : int =
        if p < num then while_one(num, p, i, x) else i
    in

    let i : int = if num <= 1 then 2 else i in
    i

let main (action, store : parameter * storage) : return =
 ([] : operation list),
 (match action with
   Primes (n) -> firstnprimes (n))
```

Listing A.4: FirstNPrimes in CameLIGO

```
type storage = int;

type parameter =
  Primes (int)

type return = (list (operation), storage);

let firstnprimes = ((num) : (int)) : storage =>
    let p = 1;
    let i = 3;
    let x = 3;

    let rec while_one = ((num, p, i, x) : (int, int, int, int)) : int =>
        let rec while_two = ((i, x) : (int, int)) : int =>
            if ((i mod x != 0n) && (x * x <= i)) {while_two(i, x + 2);} else {x;};

        let x = while_two(i, x);

        let p = if ((i mod x != 0n) || (i == 3)) {p + 1;} else {p;};

        let x = 3;

        if (p < num) {while_one(num, p, i + 2, x);} else {i};

    let i = if (p < num) {while_one(num, p, i, x)} else {i};

    if (num <= 1) {2} else {i};

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | Primes (n) => firstnprimes (n)}))
```

```
};
```

Listing A.5: FirstNPrimes in ReasonLIGO

```
archetype FirstNPrimes

variable primes : int = 0

function while_two (i : int, x : int) : int {
    var y : int = x;
    var j : int = i;

    iter z to 49 do
        if (j % y <> 0 and y * y <= j) then y := y + 2;
    done;

    return y
}

function while_one (num : int, p : int, i : int, x : int) : int {
    var y : int = x;
    var j : int = i;
    var pp : int = p;
    var nn : int = num * 5;

    iter z to nn do
        if pp < num then
        (
            y := while_two(j, y);

            if j % y <> 0 or j = 3 then pp := pp + 1;

            y := 3;

            j += 2;
        );
    done;

    return j
}

entry n_primes (num : int) {
    var p : int = 1;
    var i : int = 3;
    var x : int = 3;

    if (p < num) then i := while_one(num, p, i, x);

    if (num <= 1) then primes := 2 else primes := i - 2
}
```

Listing A.6: FirstNPrimes in Archetype

```
pragma solidity ^0.5.0;

contract FirstNPrimes_V3 {

```

```
    uint primes;

    function n_primes(uint num) public {

        uint p = 1;
        uint i = 3;
        uint x = 3;

        while (p < num)
        {
            while ((i % x != 0) && (x * x <= i))
                x += 2;

            if ((i % x != 0) || (i == 3))
                p += 1;

            x = 3;

            i += 2;
        }

        if (num <= 1)
            primes = 2;
        else
            primes = i - 2;
    }

    function getPrimes() public view returns (uint) {
        return primes;
    }
}
```

Listing A.7: FirstNPrimes in Solidity

```
primes: public(uint256)

@external
def n_primes (num : uint256):
  p: uint256 = 1
  i: uint256 = 3
  x: uint256 = 3

  for y in range(8000):
    if p < num:
      for z in range(150):
        if i % x != 0 and x * x <= i:
          x += 2
        else:
          break

      if i % x != 0 or i == 3:
        p += 1

      x = 3

      i += 2
    else:
      break


  if num <= 1:
```

```
    self.primes = 2
  else:
    self.primes = i - 2
```

Listing A.8: FirstNPrimes in Vyper

```
import smartpy as sp

class FirstNPrimesList(sp.Contract):
    def __init__(self):
        self.init(last_prime = sp.int(0), primes = sp.list())

    @sp.entry_point
    def n_primes(self, num):
        self.data.primes = [2]
        p = sp.local("p", 1)
        i = sp.local("i", 3)
        c = sp.local("c", 0)
        n = sp.local("n", num)

        sp.while p.value < n.value:

            sp.for x in self.data.primes:
                sp.if (i.value % x == 0):
                    c.value = 1

            sp.if (c.value == 0):
                self.data.primes.push(i.value)
                p.value += 1

            c.value = 0

            i.value += 2

        self.data.primes = [2]
        sp.if (n.value <= 1):
            self.data.last_prime = 2
        sp.else:
            self.data.last_prime = i.value - 2
```

Listing A.9: FirstNPrimesList in SmartPy

```
type storage = int

let%entry n_primes (num : int) _ =

  let p = 1 in
  let i = 3 in
  let l = [2] in

  let t = (l, p) in

  let i, _ =
    Loop.left (fun (i, t) ->

      let l = t.(0) in
      let p = t.(1) in
```

```

      let c = 0 in

      let c = List.fold (fun (x, c) ->

        let (_, rem) = match i / x with
          | Some qr -> qr
          | None -> failwith "division by 0 impossible" in

        if (rem = (0 : nat)) then 1 else c

      ) l c;
      in

      let l =
        if (c = 0) then i :: l else l
      in

      let p =
        if (c = 0) then p + 1 else p + 0
      in

      let t = (l, p) in

      if (p < num) then (Left (i + 2), t)
      else (Right i, t)
    ) i t
  in

  let i =
    if (num <= 1) then 2 else i
  in

  ( [], i)
```

Listing A.10: FirstNPrimesList in Liquidity

```
type storage is int

type parameter is
  Primes of int

type return is list (operation) * storage

function firstnprimes (const num : int) : storage is
  block {
      var p : int := 1;
      var i : int := 3;
      var c : int := 0;
      var s : set (int) := set [2];

      while p < num block {
        for x in set s block {
            if i mod x = 0n then c := 1 else skip;
        };

        if c = 0 then
            block {
                s := Set.add (i, s);
```

```
                p := p + 1
            }
        else skip;

        c := 0;

        i := i + 2;
      };

      if num <= 1 then i := 2 else i := i - 2

  } with i

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    Primes (n) -> firstnprimes (n)
  end)
```

Listing A.11: FirstNPrimesList in PascaLIGO

```
type storage = int

type parameter =
  Primes of int

type return = operation list * storage

let firstnprimes (num : int) : storage =
    let p : int = 1 in
    let i : int = 3 in
    let c : int = 0 in
    let s : int set = Set.empty in
    let s : int set = Set.add 2 s in

    let rec while_one (num, p, i, c, s : int * int * int * int * int set) : int =
        let sum (c, e : int * int) : int = if i mod e = 0n then 1 else c in
        let c : int = Set.fold sum s c in

        let s : int set = if c = 0 then Set.add i s else s in
        let p : int = if c = 0 then p + 1 else p in

        let c : int = 0 in

        if p < num then while_one(num, p, i + 2, c, s) else i
    in

    let i : int =
        if p < num then while_one(num, p, i, c, s) else i
    in

    let i : int = if num <= 1 then 2 else i in
    i

let main (action, store : parameter * storage) : return =
 ([] : operation list),
 (match action with
```

```
   Primes (n) -> firstnprimes (n))
```

Listing A.12: FirstNPrimesList in CameLIGO

```
type storage = int;

type parameter =
  Primes (int)

type return = (list (operation), storage);

let firstnprimes = ((num) : (int)) : storage =>
    let p = 1;
    let i = 3;
    let c = 0;
    let s : set (int) = Set.empty;
    let s : set (int) = Set.add (2, s);

    let rec while_one = ((num, p, i, c, s) : (int, int, int, int, set (int))) : int =>
        let sum = ((c, e) : (int, int)) : int => if (i mod e == 0n) {1} else {c};
        let c : int = Set.fold (sum, s, c);

        let s : set (int) = if (c == 0) {Set.add (i, s)} else {s};
        let p : int = if (c == 0) {p + 1} else {p};

        let c = 0;

        if (p < num) {while_one(num, p, i + 2, c, s);} else {i};

    let i = if (p < num) {while_one(num, p, i, c, s)} else {i};

    if (num <= 1) {2} else {i};

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | Primes (n) => firstnprimes (n)}))
};
```

Listing A.13: FirstNPrimesList in ReasonLIGO

```
archetype FirstNPrimesList

variable primes : int = 0

entry n_primes (num : int) {
    var l = [2];
    var i : nat = 3;
    var c : int = 0;
    var p : int = 1;
    var nn : int = num * 5;

    iter z to nn do
        if p < num then
        (
            for e in l do
                if i % e = 0 then c := 1;
```

```
            done;

            if c = 0 then
            (
                l := prepend(l, i);
                p := p + 1;
            );

            c := 0;

            i += 2;
        );
    done;

    if (num <= 1) then primes := 2 else primes := i - 2
}
```

Listing A.14: FirstNPrimesList in Archetype

```
pragma solidity ^0.5.0;

contract FirstNPrimesList {

    uint last_prime;

    function n_primes(uint num) public {
        uint[] memory primes = new uint[](num);
        primes[0] = 2;

        uint p = 1;
        uint i = 3;
        uint c = 0;
        uint ch = 0;

        while (p < num)
        {
            while (c < p)
            {
                if (i % primes[c] == 0)
                    ch = 1;

                c += 1;
            }

            if (ch == 0)
            {
                primes[p] = i;
                p += 1;
            }

            c = 0;
            ch = 0;

            i += 2;
        }

        if (num <= 1)
            last_prime = 2;
        else
```

```
            last_prime = i - 2;
    }

    function getPrimes() public view returns (uint) {
        return last_prime;
    }
}
```

Listing A.15: FirstNPrimesList in Solidity

```
primes: public(uint256)

@external
def n_primes (num : uint256):
  p: uint256 = 1
  i: uint256 = 3
  c: uint256 = 0
  ch: uint256 = 0
  s: uint256[2000] = [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

  for y in range(8000):
    if p < num:
      for z in range(2000):
        if s[c] != 0:
          if i % s[c] == 0:
            ch = 1

          c += 1
        else:
          break

      if ch == 0:
        s[p] = i
        p += 1

      c = 0
      ch = 0

      i += 2
    else:
      break


  if num <= 1:
    self.primes = 2
  else:
    self.primes = i - 2
```

Listing A.16: FirstNPrimesList in Vyper

```
import smartpy as sp

class FirstNPrimesBoth(sp.Contract):
    def __init__(self):
        self.init(last_prime = sp.int(0), m = {0 : 2})

    @sp.entry_point
    def n_primes(self, num):
        p = sp.local("p", 1)
        i = sp.local("i", 3)
        c = sp.local("c", 0)
        n = sp.local("n", num)

        sp.while p.value < n.value:

            sp.while (i.value % self.data.m[c.value] != 0) & (self.data.m[c.value] * self
                .data.m[c.value] <= i.value):
                c.value += 1

            sp.if (i.value % self.data.m[c.value] != 0):
                self.data.m[p.value] = i.value
                p.value += 1
```

```

            c.value = 0

            i.value += 2

        self.data.m = {0 : 2}

        sp.if (n.value <= 1):
            self.data.last_prime = 2
        sp.else:
            self.data.last_prime = i.value - 2
```

Listing A.17: FirstNPrimesBoth in SmartPy

```
type storage = int

let%entry n_primes (num : int) _ =

  let p = 1 in
  let i = 3 in
  let m = (Map [0,2] : (int, int) map) in

  let t = (m, p) in

  let i, _ =
    Loop.left (fun (i, t) ->

      let m = t.(0) in
      let p = t.(1) in

      let c = 0 in

      let _, c = Loop.left (fun (m, c) ->

        let mz = match Map.find c m with
          | None -> failwith ("id is not in the map", c)
          | Some v -> v
        in

        let (_, rem) = match i / mz with
          | Some qr -> qr
          | None -> failwith "division by 0 impossible"
        in

        if (rem <> (0 : nat) & mz * mz <= i) then (Left m, (c + 1))
        else (Right m, c)
      ) m c;
      in

      let mz = match Map.find c m with
        | None -> failwith ("id is not in the map", c)
        | Some v -> v
      in

      let (_, rem) = match i / mz with
          | Some qr -> qr
          | None -> failwith "division by 0 impossible"
      in

```

```
      let m =
        if (rem <> (0 : nat)) then Map.add p i m else m
      in

      let p =
        if (rem <> (0 : nat)) then p + 1 else p + 0
      in

      let t = (m, p) in

      if (p < num) then (Left (i + 2), t)
      else (Right i, t)
    ) i t
  in

  let i =
    if (num <= 1) then 2 else i
  in

  ( [], i)
```

Listing A.18: FirstNPrimesBoth in Liquidity

```
type storage is int

type parameter is
  Primes of int

type return is list (operation) * storage

type pr is map (int, int);

function access (const k : int; const m : pr) : int is
  case m[k] of
    Some (val) -> val
  | None -> (failwith ("No value associated.") : int)
  end

function firstnprimes (const num : int) : storage is
  block {
      var m : pr := map [0 -> 2];
      var i : int := 3;
      var p : int := 1;
      var c : int := 0;

      var mz : int := access(0, m);

      while p < num block {
        while (i mod mz =/= 0n) and (mz * mz <= i) block {
            c := c + 1;
            mz := access(c, m);
        };

        if i mod mz =/= 0n then {
            m[p] := i;
            p := p + 1;
        }
        else skip;

```

```
        var mz := access(0, m);

        i := i + 2;
      };

      if num <= 1 then i := 2 else i := i - 2

  } with i

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    Primes (n) -> firstnprimes (n)
  end)
```

Listing A.19: FirstNPrimesBoth in PascaLIGO

```
type storage = int

type parameter =
  Primes of int

type return = operation list * storage

type pr = (int, int) map

let access (k, m : int * pr) : int =
  match Map.find_opt k m with
    Some value -> value
  | None -> (failwith "No value associated." : int)

let firstnprimes (num : int) : storage =
    let m : pr = Map.literal [0, 2] in
    let p : int = 1 in
    let i : int = 3 in
    let c : int = 0 in

    let rec while_one (m, num, p, i : pr * int * int * int) : int =
        let rec while_two (m, mz, i, c : pr * int * int * int) : int =
            if i mod mz <> 0n && mz * mz <= i then while_two(m, access(c, m), i, c + 1)
                else mz
        in

        let mz : int = while_two (m, 2, i, 0) in

        let m : pr = if i mod mz <> 0n then Map.add p i m else m in
        let p : int = if i mod mz <> 0n then p + 1 else p in

        if p < num then while_one(m, num, p, i + 2) else i
    in

    let i : int =
        if p < num then while_one(m, num, p, i) else i
    in

    let i : int = if num <= 1 then 2 else i in
    i

let main (action, store : parameter * storage) : return =
```

```
 ([] : operation list),
 (match action with
   Primes (n) -> firstnprimes (n))
```

Listing A.20: FirstNPrimesBoth in CameLIGO

```
type storage = int;

type parameter =
  Primes (int)

type return = (list (operation), storage);

type pr = map (int, int);

let access = ((k, m) : (int, pr)) : int => {
  switch (Map.find_opt (k, m)) {
  | Some value => value
  | None => (failwith ("No value associated.") : int)
  }
}

let firstnprimes = ((num) : (int)) : storage =>
    let m = Map.literal([(0, 2)]);
    let p = 1;
    let i = 3;
    let c = 0;

    let rec while_one = ((m, num, p, i) : (pr, int, int, int)) : int =>
        let rec while_two = ((m, mz, i, c) : (pr, int, int, int)) : int =>
            if ((i mod mz != 0n) && (mz * mz <= i)) {while_two(m, access(c, m), i, c + 1)
                ;} else {mz;};

        let mz = while_two(m, 2, i, 0);

        let m = if (i mod mz != 0n) {Map.add(p, i, m);} else {m;};
        let p = if (i mod mz != 0n) {p + 1;} else {p;};

        if (p < num) {while_one(m, num, p, i + 2);} else {i};

    let i = if (p < num) {while_one(m, num, p, i)} else {i};

    if (num <= 1) {2} else {i};

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | Primes (n) => firstnprimes (n)}))
};
```

Listing A.21: FirstNPrimesBoth in ReasonLIGO

```
archetype FirstNPrimesBoth

variable primes : int = 0

entry n_primes (num : int) {
```

```

    var p : int = 1;
    var i : int = 3;
    var c : nat = 0;
    var nn : int = num * 5;
    var m : map<nat, int> = [ (0, 2i) ];

    iter z to nn do
        if p < num then
        (
            iter w to 32 do
                if i % m[c] <> 0 and m[c] * m[c] <= i then
                    c := c + 1
            done;

            if i % m[c] <> 0 then
            (
                m := put(m, abs(p), i);
                p := p + 1;
            );

            c := 0;

            i += 2;
        );
    done;

    if (num <= 1) then primes := 2 else primes := i - 2
}
```

Listing A.22: FirstNPrimesBoth in Archetype

```
pragma solidity ^0.5.0;

contract FirstNPrimesBoth {

    uint last_prime;

    function n_primes(uint num) public {
        uint[] memory primes = new uint[](num);
        primes[0] = 2;

        uint p = 1;
        uint i = 3;
        uint c = 0;

        while (p < num)
        {
            while (i % primes[c] != 0 && primes[c] * primes[c] <= i)
                c += 1;

            if (i % primes[c] != 0)
            {
                primes[p] = i;
                p += 1;
            }

            c = 0;

```

```
            i += 2;
        }

        if (num <= 1)
            last_prime = 2;
        else
            last_prime = i - 2;
    }

    function getPrimes() public view returns (uint) {
        return last_prime;
    }
}
```

Listing A.23: FirstNPrimesBoth in Solidity

```
primes: public(uint256)

@external
def n_primes (num : uint256):
  p: uint256 = 1
  i: uint256 = 3
  c: uint256 = 0
  s: uint256[2000] = [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

  for y in range(8000):
    if p < num:
      for z in range(150):
        if s[c] != 0 and i % s[c] != 0 and s[c] * s[c] <= i:
          c += 1
        else:
          break

      if i % s[c] != 0:
        s[p] = i
        p += 1

      c = 0

      i += 2
    else:
      break


  if num <= 1:
    self.primes = 2
  else:
    self.primes = i - 2
```

Listing A.24: FirstNPrimesBoth in Vyper

```
import smartpy as sp

class FirstNPrimesMap(sp.Contract):
    def __init__(self):
        self.init(last_prime = sp.int(0), primes = {0 : 2})

    @sp.entry_point
    def n_primes(self, num):
        p = sp.local("p", 1)
        i = sp.local("i", 3)
        c = sp.local("c", 0)
        d = sp.local("d", 0)
        n = sp.local("n", num)

        sp.while p.value < n.value:

            sp.while (c.value < p.value):
                sp.if (i.value % self.data.primes[c.value] == 0):
                    d.value = 1
                c.value += 1

```

```
            sp.if (d.value == 0):
                self.data.primes[p.value] = i.value
                p.value += 1

            c.value = 0
            d.value = 0

            i.value += 2

        self.data.primes = {0 : 2}
        sp.if (n.value <= 1):
            self.data.last_prime = 2
        sp.else:
            self.data.last_prime = i.value - 2
```

Listing A.25: FirstNPrimesMap in SmartPy

```
type storage = int

let%entry n_primes (num : int) _ =

  let p = 1 in
  let i = 3 in
  let m = (Map [0,2] : (int, int) map) in

  let t = (m, p) in

  let i, _ =
    Loop.left (fun (i, t) ->

      let m = t.(0) in
      let p = t.(1) in

      let c = 0 in
      let d = 0 in

      let cd = (c, d) in

      let _, cd = Loop.left (fun (m, cd) ->

        let c = cd.(0) in
        let d = cd.(1) in

        let mz = match Map.find c m with
          | None -> failwith ("id is not in the map", c)
          | Some v -> v
        in

        let (_, rem) = match i / mz with
          | Some qr -> qr
          | None -> failwith "division by 0 impossible"
        in

        let d =
          if (rem = (0 : nat)) then 1 else d
        in

        let c = c + 1 in

```

```
        let cd = (c, d) in

        if (cd.(0) < p) then (Left m, cd)
        else (Right m, cd)
      ) m cd;
      in

      let d = cd.(1) in

      let m =
        if (d = 0) then Map.add p i m else m
      in

      let p =
        if (d = 0) then p + 1 else p
      in

      let t = (m, p) in

      if (p < num) then (Left (i + 2), t)
      else (Right i, t)
    ) i t
  in

  let i =
    if (num <= 1) then 2 else i
  in

  ( [], i)
```

Listing A.26: FirstNPrimesMap in Liquidity

```
type storage is int

type parameter is
  Primes of int

type return is list (operation) * storage

type pr is map (int, int);

function access (const k : int; const m : pr) : int is
  case m[k] of
    Some (val) -> val
  | None -> (failwith ("No value associated.") : int)
  end

function firstnprimes (const num : int) : storage is
  block {
      var m : pr := map [0 -> 2];
      var i : int := 3;
      var p : int := 1;
      var c : int := 0;
      var d : int := 0;

      var mz : int := access(0, m);

      while p < num block {
        while (c < p) block {
```

```
            if i mod mz = 0n then {
                d := 1;
            }
            else skip;

            mz := access(c, m);

            c := c + 1;
        };

        if d = 0 then {
            m[p] := i;
            p := p + 1;
        }
        else skip;

        var mz := access(0, m);

        c := 0;
        d := 0;

        i := i + 2;
      };

      if num <= 1 then i := 2 else i := i - 2

  } with i

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    Primes (n) -> firstnprimes (n)
  end)
```

Listing A.27: FirstNPrimesMap in PascaLIGO

```
type storage = int

type parameter =
  Primes of int

type return = operation list * storage

type pr = (int, int) map

let access (k, m : int * pr) : int =
  match Map.find_opt k m with
    Some value -> value
  | None -> (failwith "No value associated." : int)

let firstnprimes (num : int) : storage =
    let m : pr = Map.literal [0, 2] in
    let p : int = 1 in
    let i : int = 3 in
    let c : int = 0 in

    let rec while_one (m, num, p, i : pr * int * int * int) : int =
        let rec while_two (m, mz, i, c, d : pr * int * int * int * int) : int =
            let d =
```

```
                if i mod mz = 0n then 1 else d
            in

            if c < p then while_two(m, access(c, m), i, c + 1, d) else d
        in

        let d : int = while_two (m, 2, i, 0, 0) in

        let m : pr = if d = 0 then Map.add p i m else m in
        let p : int = if d = 0 then p + 1 else p in

        let c = 0 in
        let d = 0 in

        if p < num then while_one(m, num, p, i + 2) else i
    in

    let i : int =
        if p < num then while_one(m, num, p, i) else i
    in

    let i : int = if num <= 1 then 2 else i in
    i

let main (action, store : parameter * storage) : return =
 ([] : operation list),
 (match action with
   Primes (n) -> firstnprimes (n))
```

Listing A.28: FirstNPrimesMap in CameLIGO

```
type storage = int;

type parameter =
  Primes (int)

type return = (list (operation), storage);

type pr = map (int, int);

let access = ((k, m) : (int, pr)) : int => {
  switch (Map.find_opt (k, m)) {
  | Some value => value
  | None => (failwith ("No value associated.") : int)
  }
}

let firstnprimes = ((num) : (int)) : storage =>
    let m = Map.literal([(0, 2)]);
    let p = 1;
    let i = 3;
    let c = 0;

    let rec while_one = ((m, num, p, i) : (pr, int, int, int)) : int =>
        let rec while_two = ((m, mz, i, c, d) : (pr, int, int, int, int)) : int =>
            let d = if (i mod mz == 0n) {1;} else {d;};

            if (c < p) {while_two(m, access(c, m), i, c + 1, d);} else {d;};

```

```
        let d = while_two(m, 2, i, 0, 0);

        let m = if (d == 0) {Map.add(p, i, m);} else {m;};
        let p = if (d == 0) {p + 1;} else {p;};

        let c = 0;
        let d = 0;

        if (p < num) {while_one(m, num, p, i + 2);} else {i};

    let i = if (p < num) {while_one(m, num, p, i)} else {i};

    if (num <= 1) {2} else {i};

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | Primes (n) => firstnprimes (n)}))
};
```

Listing A.29: FirstNPrimesMap in ReasonLIGO

```
archetype FirstNPrimesMap

variable primes : int = 0

entry n_primes (num : int) {
    var m : map<nat, nat> = [ (0, 2) ];
    var i : nat = 3;
    var c : int = 0;
    var p : int = 1;
    var nn : int = num * 5;

    iter z to nn do
        if p < num then
        (
            for (k, v) in m do
                if i % v = 0 then c := 1;
            done;

            if c = 0 then
            (
                m := put(m, abs(p), i);
                p := p + 1;
            );

            c := 0;

            i += 2;
        );
    done;

    if (num <= 1) then primes := 2 else primes := i - 2
}
```

Listing A.30: FirstNPrimesMap in Archetype

```python
import smartpy as sp

class NQueens(sp.Contract):
    def __init__(self):
        self.init(m = {0 : 0}, used = {0 : 0})
        #"m" is the board //id is row, value is column
        #"used" are the spots where the current can/cannot be put in (1 is blocked, 0 is open)

    @sp.entry_point
    def n_queens(self, num):
        self.data.m = {}
        self.data.used = {}
        n = sp.local("n", num)           #number of queens to put on nxn board
        count = sp.local("count", 0)     #put on board so far
        k = sp.local("k", 0)             #go through the ones already on the board to compare to current one
        d = sp.local("d", 0)             #check diagonals
        v = sp.local("v", 0)             #add or sub to diagonals
        back = sp.local("back", False)   #check if its backtracking

        sp.if (n.value <= 0) | (n.value == 2) | (n.value == 3):
            self.data.m = {}
        sp.else:
            sp.while count.value < n.value:
                sp.if count.value == 0:
                    sp.if back.value == False:
                        self.data.m[count.value] = 0
                    sp.else:
                        self.data.m[count.value] += 1

                        back.value = False

                    count.value += 1

                sp.else:
                    sp.while k.value < count.value:
                        v.value = self.data.m[k.value]
                        self.data.used[v.value] = 1
                        d.value = count.value - k.value

                        sp.if v.value + d.value < n.value:
                            self.data.used[v.value + d.value] = 1

                        sp.if v.value - d.value >= 0:
                            self.data.used[v.value - d.value] = 1

                        k.value += 1

                    sp.if back.value == True:
                        v.value = self.data.m[count.value]
                        k.value = 0
                        sp.while k.value < v.value + 1:
                            self.data.used[k.value] = 1
                            k.value += 1
                        back.value = False

                    k.value = 0

                    sp.while self.data.used.get(k.value, default_value = 0) == 1:
                        k.value += 1
```

```

                    sp.if k.value >= n.value:
                        back.value = True
                        count.value -= 1
                    sp.else:
                        self.data.m[count.value] = k.value
                        count.value += 1

                    k.value = 0
                    self.data.used = {}
```

Listing A.31: NQueens in SmartPy

```
type storage = (int, int) map

let%entry n_queens (n : int) _ =

  let m =
    if (n <= 0 or n = 2 or n = 3) then Map [] else

      let m = (Map [] : (int, int) map) in
      let used = (Map [] : (int, int) map) in
      let count = 0 in
      let k = 0 in
      let d = 0 in
      let v = 0 in
      let back = false in

      let t = (m, count, back, used, k, d, v) in

      let _, t =
        Loop.left (fun (_, t) ->

          let m = t.(0) in
          let count = t.(1) in
          let back = t.(2) in
          let used = t.(3) in
          let k = t.(4) in
          let d = t.(5) in
          let v = t.(6) in

          let t = (m, count, back, used, k, d, v) in

          let t =
            if count = 0 then
              let m = t.(0) in
              let count = t.(1) in
              let back = t.(2) in
              let used = t.(3) in
              let k = t.(4) in
              let d = t.(5) in
              let v = t.(6) in

              let t = (m, count, back, used, k, d, v) in

              let t =
                if back = false then
                  let m = t.(0) in
                  let count = t.(1) in
```

```
                  let back = t.(2) in
                  let used = t.(3) in
                  let k = t.(4) in
                  let d = t.(5) in
                  let v = t.(6) in
                  let m = Map.add count 0 m in
                  let t = (m, count, back, used, k, d, v) in
                  t
                else
                  let m = t.(0) in
                  let count = t.(1) in
                  let used = t.(3) in
                  let k = t.(4) in
                  let d = t.(5) in
                  let v = t.(6) in

                  let x = match Map.find count m with
                    | None -> failwith ("id is not in the map", count)
                    | Some x -> x
                  in

                  let x = x + 1 in

                  let m = Map.add count x m in

                  let back = false in

                  let t = (m, count, back, used, k, d, v) in
                  t
              in
              let m = t.(0) in
              let count = t.(1) in
              let back = t.(2) in
              let used = t.(3) in
              let k = t.(4) in
              let d = t.(5) in
              let v = t.(6) in

              let count = count + 1 in

              let t = (m, count, back, used, k, d, v) in

              t

            else
              let _, t =
                Loop.left (fun (_, t) ->
                  let m = t.(0) in
                  let count = t.(1) in
                  let back = t.(2) in
                  let used = t.(3) in
                  let k = t.(4) in

                  let v =
                    match Map.find k m with
                    | None -> failwith ("id is not in the map", k)
                    | Some v -> v
                  in

                  let used = Map.add v 1 used in
                  let d = count - k in
```

```

                let used =
                  if v + d < n then
                    let vd = v + d in
                    Map.add vd 1 used
                  else
                    used
                in

                let used =
                  if v - d >= 0 then
                    let dv = v - d in
                    Map.add dv 1 used
                  else
                    used
                in

                let k = k + 1 in
                let t = (m, count, back, used, k, d, v) in

              if (k < count) then (Left m, t)
              else (Right m, t)
            ) m t
            in

            let back = t.(2) in

            let t =
              if back = true then
                let m = t.(0) in
                let count = t.(1) in
                let back = t.(2) in
                let used = t.(3) in
                let d = t.(5) in

                let v =
                  match Map.find count m with
                  | None -> failwith ("id is not in the map", count)
                  | Some v -> v
                in

                let k = 0 in
                let t = (m, count, back, used, k, d, v) in

                let _, t =
                  Loop.left (fun (_, t) ->
                    let m = t.(0) in
                    let count = t.(1) in
                    let back = t.(2) in
                    let used = t.(3) in
                    let k = t.(4) in
                    let d = t.(5) in
                    let v = t.(6) in

                    let used = Map.add k 1 used in

                    let k = k + 1 in
                    let t = (m, count, back, used, k, d, v) in

                    if (k < v + 1) then (Left used, t)
                    else (Right used, t)
```

```
                    ) used t
                  in

                  let back = false in

                  let t = (t.(0), t.(1), back, t.(3), t.(4), t.(5), t.(6)) in

                  t

                else
                  t
              in

              let t = (t.(0), t.(1), t.(2), t.(3), 0, t.(5), t.(6)) in
              let used = t.(3) in

              let _, t =
                Loop.left (fun (_, t) ->

                  let m = t.(0) in
                  let count = t.(1) in
                  let back = t.(2) in
                  let used = t.(3) in
                  let k = t.(4) in
                  let d = t.(5) in
                  let v = t.(6) in

                  let value =
                    match Map.find k used with
                    | None -> 0
                    | Some value -> value
                  in

                  let k = k + 1 in
                  let t = (m, count, back, used, k, d, v) in

                if (value = 1) then (Left used, t)
                else (Right used, t)
                ) used t
              in

              let k = t.(4) in
              let k = k - 1 in
              let t = (t.(0), t.(1), t.(2), t.(3), k, t.(5), t.(6)) in

              let t =
                if k >= n then
                  let m = t.(0) in
                  let count = t.(1) in
                  let used = t.(3) in
                  let k = t.(4) in
                  let d = t.(5) in
                  let v = t.(6) in

                  let back = true in
                  let count = count - 1 in
                  let t = (m, count, back, used, k, d, v) in

                  t
                else
                  let m = t.(0) in
```

```
                  let count = t.(1) in
                  let back = t.(2) in
                  let used = t.(3) in
                  let k = t.(4) in
                  let d = t.(5) in
                  let v = t.(6) in

                  let m = Map.add count k m in
                  let count = count + 1 in
                  let t = (m, count, back, used, k, d, v) in

                  t
              in

              let m = t.(0) in
              let count = t.(1) in
              let back = t.(2) in
              let d = t.(5) in
              let v = t.(6) in

              let k = 0 in
              let used = Map [] in
              let t = (m, count, back, used, k, d, v) in

              t
          in

        if (t.(1) < n) then (Left m, t)
        else (Right m, t)
      ) m t
      in

    let m = t.(0) in

    m
  in

  ( [], m)
```

Listing A.32: NQueens in Liquidity

```
type storage is map (int, int)

type parameter is
  NQueens of int

type return is list (operation) * storage

type q is map (int, int)

function access (const k : int; const m : q) : int is
  case m[k] of
    Some (val) -> val
  | None -> 0
  end

function n_queens (const n : int) : storage is
  block {
      var m : q := map [];
```

```
      var used : q := map [];
      var count : int := 0;
      var k : int := 0;
      var d : int := 0;
      var v : int := 0;
      var back : bool := False;

      if n <= 0 or n = 2 or n = 3 then
        count := n
      else {
        skip;
      };

      while count < n block {
        if count = 0 then {
            if back = False then
                m[count] := 0;
            else {
                v := access(count, m);
                m[count] := v + 1;

                back := False
            };

            count := count + 1
        }
        else {
            for i := k to count-1 block {
                v := access(i, m);
                used[v] := 1;
                d := count - i;

                if v + d < n then
                    used[v + d] := 1
                else skip;

                if v - d >= 0 then
                    used[v - d] := 1
                else skip;
            };

            if back = True then {
                v := access(count, m);
                for i := 0 to v block {
                    used[i] := 1;
                };

                back := False

            } else skip;

            k := 0;

            for i := 0 to n-1 block {
                if access(k, used) = 1 then
                    k := k + 1;
                else
                    i := n-1;
            };

            if k >= n then {
```

```
                back := True;
                count := count - 1
            }
            else {
                m[count] := k;
                count := count + 1
            };

            k := 0;
            used := map [0->0]
        }
      }

  } with m

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    NQueens (n) -> n_queens (n)
  end)
```

Listing A.33: NQueens in PascaLIGO

```
type storage = (int, int) map

type parameter =
  NQueens of int

type return = operation list * storage

type q = (int, int) map
type t = q * int * bool

let access (k, m : int * q) : int =
  match Map.find_opt k m with
    Some value -> value
  | None -> 0

let n_queens (n : int) : storage =
    let m : q = Map.empty in
    let used : q = Map.empty in
    let count : int = 0 in
    let k : int = 0 in
    let back : bool = false in

    let count =
        if (n <= 0 || n = 2 || n = 3) then n else count
    in

    let rec main_while(used, k, t, n : q * int * t * int) : q =
        let (_, count, _) : t = t in

        let t =
            if count = 0 then (
                let (m, count, back) : t = t in

                let t =
                    if back = false then (
                        let m = Map.add count 0 m in
```

```
                        (m, count, back)
                    ) else (
                        let mz = access(count, m) in
                        let mz = mz + 1 in
                        let m = Map.add count mz m in

                        let back = false in
                        (m, count, back)
                    )
                in

                let (m, count, back) = t in

                (m, count + 1, back)
            ) else (
                let (m, count, back) : t = t in

                let rec while_diagonals(m, n, k, count, used : q * int * int * int * q) : q =
                    let v = access(k, m) in
                    let used = Map.add v 1 used in
                    let d = count - k in
                    let vd = v + d in
                    let dv = v - d in

                    let used =
                        if v + d < n then Map.add vd 1 used else used
                    in

                    let used =
                        if v - d >= 0 then Map.add dv 1 used else used
                    in

                    let k = k + 1 in

                    if k < count then while_diagonals(m, n, k, count, used) else used
                in

                let used =
                    if k < count then while_diagonals(m, n, k, count, used) else used
                in

                let used =
                    if back = true then (
                        let v = access(count, m) in
                        let k = 0 in

                        let rec while_fill(used, k, v : q * int * int) : q =
                            let used = Map.add k 1 used in

                            let k = k + 1 in

                            if k < v + 1 then while_fill(used, k, v) else used
                        in

                        if k < v + 1 then while_fill(used, k, v) else used
                    ) else used
                in

                let back =
                    if back = true then false else back
```

```
                in

                let k = 0 in

                let rec while_find_empty(used, k : q * int) : int =
                    let k = k + 1 in

                    if access(k, used) = 1 then while_find_empty(used, k) else k
                in

                let k =
                    if access(k, used) = 1 then while_find_empty(used, k) else k
                in

                let back = if k >= n then true else back in
                let m = if k >= n then m else Map.add count k m in
                let count = if k >= n then count - 1 else count + 1 in

                (m, count, back)
            )
        in

        let (m, count, back) = t in

        if count < n then main_while(used, k, t, n) else m
    in

    let t : t = (m, count, back) in

    let m =
        if count < n then main_while(used, k, t, n) else m
    in

    m

let main (action, store : parameter * storage) : return =
 ([] : operation list),
 (match action with
   NQueens (n) -> n_queens (n))
```

Listing A.34: NQueens in CameLIGO

```
type storage = map (int, int);

type parameter =
 NQueens (int)

type return = (list (operation), storage);

type q = map (int, int);
type t = (q, int, bool);

let access = ((k, m) : (int, q)) : int => {
  switch (Map.find_opt (k, m)) {
  | Some value => value
  | None => 0
  }
}

```

```
let n_queens = ((n) : (int)) : storage =>
    let m : q = Map.empty;
    let used : q = Map.empty;
    let count = 0;
    let k = 0;
    let back = false;

    let count = if (n <= 0 || n == 2 || n == 3) {n;} else {count;};

    let rec main_while = ((used, k, t, n) : (q, int, t, int)) : q =>
        let (_, count, _) : t = t;

        let t =
            if (count == 0) {
                let (m, count, back) : t = t;

                let t =
                    if (back == false) {
                        let m = Map.add(count, 0, m);
                        (m, count, back);
                    } else {
                        let mz = access(count, m);
                        let mz = mz + 1;
                        let m = Map.add(count, mz, m);

                        let back = false;
                        (m, count, back);
                    };

                let (m, count, back) = t;

                (m, count + 1, back);
            } else {
                let (m, count, back) : t = t;

                let rec while_diagonals = ((m, n, k, count, used) : (q, int, int, int, q)
                    ) : q =>
                    let v = access(k, m);
                    let used = Map.add(v, 1, used);
                    let d = count - k;
                    let vd = v + d;
                    let dv = v - d;

                    let used = if (v + d < n) {Map.add(vd, 1, used);} else {used;};

                    let used = if (v - d >= 0) {Map.add(dv, 1, used);} else {used;};

                    let k = k + 1;

                    if (k < count) {while_diagonals(m, n, k, count, used);} else {used;};

                let used = if (k < count) {while_diagonals(m, n, k, count, used);} else {
                    used;};

                let used =
                    if (back == true) {
                        let v = access(count, m);
                        let k = 0;

                        let rec while_fill = ((used, k, v) : (q, int, int)) : q =>
                            let used = Map.add(k, 1, used);
```

```

                            let k = k + 1;

                            if (k < v + 1) {while_fill(used, k, v);} else {used;};

                        if (k < v + 1) {while_fill(used, k, v);} else {used;};
                    } else {used;};

                let back = if (back == true) {false;} else {back;};

                let k = 0;

                let rec while_find_empty = ((used, k) : (q, int)) : int =>
                    let k = k + 1;

                    if (access(k, used) == 1) {while_find_empty(used, k);} else {k;};

                let k = if (access(k, used) == 1) {while_find_empty(used, k);} else {k;};

                let back = if (k >= n) {true;} else {back;};
                let m = if (k >= n) {m;} else {Map.add(count, k, m);};
                let count = if (k >= n) {count - 1;} else {count + 1;};

                (m, count, back);
            };

        let (m, count, back) = t;

        if (count < n) {main_while(used, k, t, n);} else {m;};

    let t : t = (m, count, back);

    let m = if (count < n) {main_while(used, k, t, n);} else {m;};

    m;

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | NQueens (n) => n_queens (n)}))
};
```

Listing A.35: NQueens in ReasonLIGO

```
archetype NQueens

variable queens : map<nat, nat> = []

entry n_queens (num : int) {

    var m : map<nat, nat> = [];
    var used : map<nat, nat> = [];
    var n : nat = abs(num);
    var count : int = 0;
    var k : nat = 0;
    var back : bool = false;
    var it : nat = 0;

    if (n = 1 or n = 5) then it := n + 1 else (
```

```
        if (n = 4 or n = 7) then it := 13 else (
            if (n = 6 or n = 9 or n = 11) then it := n * 9 + 3 else (
                if (n = 8) then it := 220 else (
                    if (n = 10) then it := n * 20 else (
                        if (n = 13) then it := n * 16 + 2 else (
                            if (n = 12) then it := n * 43 else (
                                if (n = 14) then it := 3790 else (
                                    if (n = 15) then it := 2705
                                )
                            )
                        )
                    )
                )
            )
        )
    );

    if (n <= 0 or n = 2 or n = 3) then m := [] else (
        iter i to it do
            if count < n then (
                if count = 0 then (
                    if back = false then m := put(m, abs(count), 0) else (
                        var t = m[abs(count)];
                        t := t + 1;
                        m := put(m, abs(count), t);
                        back := false
                    );

                    count := count + 1
                ) else (
                    iter a to count + 1 do
                        if k < count then (
                            var v = m[k];
                            used := put(used, v, 1);
                            var d = count - k;
                            var vd = v + d;
                            var dv = v - d;

                            if vd < n then used := put(used, abs(vd), 1);

                            if dv >= 0 then used := put(used, abs(dv), 1);

                            k := k + 1
                        );
                    done;

                    if back = true then (
                        var v = m[abs(count)];
                        k := 0;

                        iter e to v + 2 do
                            if k < v + 1 then (
                                used := put(used, k, 1);
                                k := k + 1
                            );
                        done;

                        back := false
                    );

                    k := 0;
```

```

                    iter o to n + 1 do
                        var c = contains(used, k);

                        if c = true then k := k + 1
                    done;

                    if k >= n then (
                        back := true;
                        count := count - 1
                    ) else (
                        m := put(m, abs(count), k);
                        count := count + 1
                    );

                    k := 0;
                    used := []
                );
            );
        done;
    );

    queens := m
}
```

Listing A.36: NQueens in Archetype

```
pragma solidity ^0.5.0;

contract NQueens {

    uint[] q;

    function n_queens(uint num) public {
        uint[] memory m = new uint[](num);
        uint[] memory used = new uint[](num);
        uint count = 0;
        uint k = 0;
        uint d = 0;
        uint v = 0;
        bool back = false;

        if (num == 1 || num >= 4)
        {
            while (count < num)
            {
                if (count == 0)
                {
                    if (back == false)
                        m[count] = 0;
                    else
                    {
                        m[count] += 1;

                        back = false;
                    }

                    count += 1;
                }
```

```
                else
                {
                    while (k < count)
                    {
                        v = m[k];
                        used[v] = 1;
                        d = count - k;
                        int dv = int (v - d);

                        if (v + d < num)
                            used[v + d] = 1;

                        if (dv >= 0)
                            used[uint (dv)] = 1;

                        k += 1;
                    }

                    if (back == true)
                    {
                        v = m[count];
                        k = 0;

                        while (k < (v + 1))
                        {
                            used[k] = 1;
                            k += 1;
                        }

                        back = false;
                    }

                    k = 0;

                    while (k < num && used[k] == 1)
                        k += 1;

                    if (k >= num)
                    {
                        back = true;
                        count -= 1;
                    }
                    else
                    {
                        m[count] = k;
                        count += 1;
                    }

                    k = 0;
                    used = new uint[](num);
                }
            }
        }

        q = m;
    }

    function getPositions() public view returns (uint[] memory) {
        return q;
    }
```

```
}
```

Listing A.37: NQueens in Solidity

```
queens: public(uint256[15])

@external
def n_queens (num : uint256):
    m: uint256[15] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    used: uint256[15] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    count: uint256 = 0
    k: uint256 = 0
    d: uint256 = 0
    v: uint256 = 0
    back: bool = False

    if num == 1 or num >= 4:
        for x in range(4000):
            if count < num:
                if count == 0:
                    if back == False:
                        m[count] = 0
                    else:
                        m[count] += 1
                        back = False

                    count += 1
                else:
                    for y in range(15):
                        if k < count:
                            v = m[k]
                            used[v] = 1
                            d = count - k

                            if v + d < num:
                                used[v + d] = 1

                            if v >= d:
                                used[v - d] = 1

                            k += 1
                        else:
                            break

                    if back == True:
                        v = m[count]
                        k = 0

                        for z in range(15):
                            if k < v + 1:
                                used[k] = 1
                                k += 1
                            else:
                                break

                        back = False

                    k = 0

```

```
                    for a in range(15):
                        if k < num and used[k] == 1:
                            k += 1
                        else:
                            break

                    if k >= num:
                        back = True
                        count -= 1
                    else:
                        m[count] = k
                        count += 1

                    k = 0
                    used = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
            else:
                break

    self.queens = m
```

Listing A.38: NQueens in Vyper

```
import smartpy as sp

#program gets a magic square of NxN with superposition (Euler's method)
#N = 6 doesn't work with this method
class MagicSquareSP(sp.Contract):
    def __init__(self):
        self.init(a = {0 : 0}, b = {0 : 0}, used = {0 : 0}, use = {0 : 0}, m = {0 : 0})
        #"a" is the first square
        #"b" is the second square
        #"used" is the numbers left to use (or in use)
        #"use" is same as "used" (to check both squares at once)
        #"used" is for "a", "use" is for "b"
        #"m" is 1st used to check the pairs of numbers being used
        #and at the end is the final square which should be magic

    @sp.entry_point
    def magic_square(self, num):
        self.data.a = {}
        self.data.b = {}
        self.data.used = {}
        self.data.use = {}
        self.data.m = {}
        n = sp.local("n", num)
        i = sp.local("i", 0)
        j = sp.local("j", 0)
        s = sp.local("s", 0)
        k = sp.local("k", 0)
        t = sp.local("t", 0)
        p = sp.local("p", 0)
        back = sp.local("back", False)
        b_o = sp.local("b_o", 0)
        b = sp.local("b", False)
        #b_o is used to check if either "used" or "use" overflowed
        #0 = no overflow or "use" overflowed, 1 = "used" overflowed
        #2 = both "used" and "use" overflowed

        t.value = abs(n.value - 1) / abs(2)
```

```
        k.value = abs(n.value - 1)

        #fill main diagonal of "a" and 2nd diagonal of "b"
        sp.while (i.value < n.value):
            sp.if (n.value % 2 == 0):
                t.value = i.value

            s.value = i.value * n.value + j.value
            p.value = i.value * n.value + k.value
            self.data.a[s.value] = t.value
            self.data.b[p.value] = t.value
            i.value = i.value + 1
            j.value = j.value + 1
            k.value = abs(k.value - 1)

        i.value = abs(n.value - 1)
        j.value = 0
        p.value = 0
        k.value = 0
        t.value = abs(n.value - 1) / abs(2)

        sp.if (n.value % 2 == 1):
            self.data.used[t.value] = 1
            self.data.m[t.value * n.value + t.value] = 1

        #fill 2nd diagonal of "a" and main diagonal of "b"
        sp.while (j.value < n.value):
            sp.if (i.value == j.value) & (n.value != 1):
                i.value = abs(i.value - 1)
                j.value = j.value + 1
                p.value = p.value + 1

            s.value = i.value * n.value + j.value

            sp.if (n.value % 2 == 1):
                sp.while (self.data.used.get(k.value, default_value = 0) == 1):
                    k.value = k.value + 1
            sp.else:
                sp.while (self.data.used.get(k.value, default_value = 0) == 1) | (k.value == i.value) | (k.value == j.value):
                    k.value = k.value + 1

            self.data.a[s.value] = k.value
            self.data.used[k.value] = 1

            s.value = k.value * n.value + self.data.b[s.value]
            self.data.m[s.value] = 1

            s.value = p.value * n.value + j.value
            self.data.b[s.value] = k.value

            s.value = self.data.a[s.value] * n.value + k.value
            self.data.m[s.value] = 1

            i.value = abs(i.value - 1)
            j.value = j.value + 1
            p.value = p.value + 1
            k.value = 0

        self.data.used = {}
        i.value = 0
```

```
        j.value = 0
        k.value = 0

        #filling the rest of "a" and "b" with backtracking
        sp.while (i.value < abs(n.value - 1)) | (j.value < abs(n.value - 1)):
            t.value = i.value + j.value
            #checks if current position is one of the diagonals
            sp.if (i.value == j.value) | (t.value == abs(n.value - 1)):
                sp.if (back.value == False):
                    sp.if (j.value < abs(n.value - 1)):
                        j.value = j.value + 1
                    sp.else:
                        i.value = i.value + 1
                        j.value = 0
                sp.else:
                    sp.if (j.value > 0):
                        j.value = abs(j.value - 1)
                    sp.else:
                        i.value = abs(i.value - 1)
                        j.value = abs(n.value - 1)

            sp.else:
                k.value = 0

                #go through current line/column (of "a" and "b")
                #to mark the values that cannot be chosen in
                #"a" on "used" and in "b" on "use"
                sp.while (k.value < n.value):
                    sp.if (k.value != j.value):
                        s.value = i.value * n.value + k.value

                        p.value = self.data.a.get(s.value, default_value = n.value)
                        sp.if (p.value != n.value):
                            self.data.used[p.value] = 1

                        p.value = self.data.b.get(s.value, default_value = n.value)
                        sp.if (p.value != n.value):
                            self.data.use[p.value] = 1

                    sp.if (k.value != i.value):
                        s.value = k.value * n.value + j.value

                        p.value = self.data.a.get(s.value, default_value = n.value)
                        sp.if (p.value != n.value):
                            self.data.used[p.value] = 1

                        p.value = self.data.b.get(s.value, default_value = n.value)
                        sp.if (p.value != n.value):
                            self.data.use[p.value] = 1

                    k.value = k.value + 1

                #restrict the position a(1.0) for a faster filling
                sp.if (i.value == 1) & (j.value == 0):
                    k.value = 0

                    sp.if (n.value % 2 == 0):
                        t.value = n.value / 2
                    sp.else:
                        t.value = abs(n.value - 1)

```

```
                    sp.while (k.value < t.value):
                        self.data.used[k.value] = 1
                        k.value = k.value + 1

                #restrict the position b(1.2) for a faster filling
                sp.if (i.value == 1) & (j.value == 2):
                    k.value = 0

                    t.value = abs(n.value / 2 - 1)

                    sp.while (k.value < t.value):
                        self.data.use[k.value] = 1
                        k.value = k.value + 1

                #restrict the position b(1.3) for a faster filling
                sp.if (i.value == 1) & (j.value == 3):
                    k.value = 0
                    t.value = 3

                    sp.while (k.value < t.value):
                        self.data.use[k.value] = 1
                        k.value = k.value + 1

                sp.if (back.value == True):
                    s.value = i.value * n.value + j.value
                    k.value = self.data.a[s.value]
                    t.value = self.data.b[s.value]
                    self.data.a[s.value] = n.value
                    self.data.b[s.value] = n.value

                    s.value = k.value * n.value + t.value
                    self.data.m[s.value] = 0

                    sp.if (b_o.value == 2):
                        k.value = k.value + 1
                        t.value = t.value + 1
                    sp.else:
                        sp.if (b_o.value == 1):
                            k.value = k.value + 1
                            t.value = 0
                        sp.else:
                            t.value = t.value + 1

                sp.else:
                    k.value = 0
                    t.value = 0
                    b_o.value = 0

                #find open value to put in "a"
                sp.while (self.data.used.get(k.value, default_value = 0) == 1):
                    k.value = k.value + 1

                #find open value to put in "b"
                sp.while (self.data.use.get(t.value, default_value = 0) == 1):
                    t.value = t.value + 1

                ########################################
                #Possible scenarios from this point on:
                #1. back = false, no overflow, see if pair exists, go thru all pair opts. If all exist, back with b_o = 1
                #2. back = false, k overflow, backtrack with b_o = 1
```

```
                #3. back = false, t overflow, backtrack with b_o = 0
                #4. back = false, both k and t overflow, backtrack with b_o = 2
                #5. back = true, no overflow, see if pair exists, go thru all pair opts. If all exist, back with b_o = 1
                #6. back = true, k overflow, backtrack with b_o = 0
                #7. back = true, t overflow, t = 0, k = k + 1, go thru all pair opts. If all exist, back with b_o = 1
                #8. back = true, both k and t overflow, back with b_o = 0
                ########################################

                #this covers scenarios 1, 5 and 7
                sp.if ((k.value < n.value) & (t.value < n.value)) | ((back == True) & (k.value < n.value)):
                    sp.if (t.value >= n.value):
                        t.value = 0
                        k.value = k.value + 1

                    #go through all possible pairs, if all exist backtrack with b_o = 1
                    sp.while (k.value < n.value) & (b.value == False):
                        sp.while (self.data.used.get(k.value, default_value = 0) == 1):
                            k.value = k.value + 1

                        sp.while (self.data.use.get(t.value, default_value = 0) == 1):
                            t.value = t.value + 1

                        sp.if (t.value >= n.value) | (k.value >= n.value):
                            k.value = k.value + 1
                            t.value = 0
                        sp.else:
                            s.value = k.value * n.value + t.value
                            sp.if (self.data.m.get(s.value, default_value = 0) != 1):
                                self.data.m[s.value] = 1
                                s.value = i.value * n.value + j.value
                                self.data.a[s.value] = k.value
                                self.data.b[s.value] = t.value

                                b.value = True
                            sp.else:
                                t.value = t.value + 1

                                sp.if (t.value >= n.value):
                                    k.value = k.value + 1
                                    t.value = 0

                    #if b is true, it means it found an open pair, if not, all were taken
                    sp.if (b.value == True):
                        sp.if (j.value < abs(n.value - 1)):
                            j.value = j.value + 1
                        sp.else:
                            i.value = i.value + 1
                            j.value = 0

                        b.value = False
                        back.value = False
                    sp.else:
                        sp.if (j.value > 0):
                            j.value = abs(j.value - 1)
                        sp.else:
                            i.value = abs(i.value - 1)
                            j.value = abs(n.value - 1)

```

```
                        back.value = True
                        b_o.value = 1

                sp.else:
                    #this covers scenarios 2, 3 and 4
                    sp.if (back.value == False):
                        sp.if (k.value >= n.value) & (t.value >= n.value):
                            b_o.value = 2
                        sp.else:
                            sp.if (k.value >= n.value):
                                b_o.value = 1
                            sp.else:
                                b_o.value = 0
                    #this covers scenarios 6 and 8
                    sp.else:
                        b_o.value = 0

                    sp.if (j.value > 0):
                        j.value = abs(j.value - 1)
                    sp.else:
                        i.value = abs(i.value - 1)
                        j.value = abs(n.value - 1)

                    back.value = True


                k.value = 0
                self.data.used = {}
                self.data.use = {}


        self.data.m = {}
        i.value = 0
        j.value = 0
        #multiplying "a" by N, then adding "b" and one
        sp.while (i.value < n.value):
            s.value = i.value * n.value + j.value
            t.value = self.data.a[s.value]
            k.value = self.data.b[s.value]
            p.value = t.value * n.value + k.value + 1
            self.data.m[s.value] = p.value

            sp.if (j.value < abs(n.value - 1)):
                j.value = j.value + 1
            sp.else:
                i.value = i.value + 1
                j.value = 0


        self.data.a = {}
        self.data.b = {}
        self.data.used = {}
        self.data.use = {}
```

Listing A.39: MagicSquare in SmartPy

```
type storage = (int, int) map

let%entry magic_square (num : int) _ =
```

```

let a = (Map [] : (int, int) map) in
let b = (Map [] : (int, int) map) in
let used = (Map [] : (int, int) map) in
let m = (Map [] : (int, int) map) in
let i = 0 in
let j = 0 in
let back = false in
let b_o = 0 in

let (t, _) = match (num - 1) / 2 with
  | Some qr -> qr
  | None -> failwith "division by 0 impossible" in

let k = num - 1 in

let ab = (a, b) in
let ijk = (i, j, k) in

let ab, _ =
  Loop.left (fun (ab, ijk) ->
    let (_, r) = match num / 2 with
      | Some qr -> qr
      | None -> failwith "division by 0 impossible" in

    let t =
      if (r = (0 : nat)) then ijk.(0) else t
    in

    let s = ijk.(0) * num + ijk.(1) in
    let p = ijk.(0) * num + ijk.(2) in

    let a = ab.(0) in
    let b = ab.(1) in
    let a = Map.add s t a in
    let b = Map.add p t b in
    let ab = (a, b) in

    let i = ijk.(0) in
    let j = ijk.(1) in
    let k = ijk.(2) in
    let i = i + 1 in
    let j = j + 1 in
    let k = k - 1 in
    let ijk = (i, j, k) in

    if (ijk.(0) < num) then (Left ab, ijk)
    else (Right ab, ijk)
  ) ab ijk;
in

let i = num - 1 in
let j = 0 in
let p = 0 in

let (t, _) = match (num - 1) / 2 with
  | Some qr -> qr
  | None -> failwith "division by 0 impossible" in

let (_, r) = match num / 2 with
  | Some qr -> qr
```

```
  | None -> failwith "division by 0 impossible" in

let used =
  if r = (1 : nat) then Map.add t 1 used else used
in

let m =
  if r = (1 : nat) then Map.add (t * num + t) 1 m else m
in

let ijp = (i, j, p) in
let a = ab.(0) in
let b = ab.(1) in
let abum = (a, b, used, m) in

let abum, _ =
  Loop.left (fun (abum, ijp) ->
    let ijp =
      if (ijp.(0) = ijp.(1) & num <> 1) then
        let i = ijp.(0) - 1 in
        let j = ijp.(1) + 1 in
        let p = ijp.(2) + 1 in
        (i, j, p)
      else
        ijp
    in

    let s = ijp.(0) * num + ijp.(1) in

    let k = 0 in
    let used = abum.(2) in

    let k =
      if (r = (1 : nat)) then
        let k, _ = Loop.left (fun (k, used) ->
          let mz = match Map.find k used with
            | None -> 0
            | Some v -> v
          in

          if (mz = 1) then (Left (k + 1), used)
          else (Right k, used)
        ) k used;
        in
        k
      else
        let k, _ = Loop.left (fun (k, used) ->
          let mz = match Map.find k used with
            | None -> 0
            | Some v -> v
          in

          if (mz = 1 || k = ijp.(0) || k = ijp.(1)) then (Left (k + 1), used)
          else (Right k, used)
        ) k used;
        in
        k
    in

    let a = abum.(0) in
    let used = abum.(2) in
```

```

      let a = Map.add s k a in
      let used = Map.add k 1 used in

      let b = abum.(1) in
      let m = abum.(3) in

      let x = match Map.find s b with
        | None -> 0
        | Some v -> v
      in

      let s = k * num + x in
      let m = Map.add s 1 m in

      let s = ijp.(2) * num + ijp.(1) in
      let b = Map.add s k b in

      let x = match Map.find s a with
        | None -> 0
        | Some v -> v
      in

      let s = x * num + k in
      let m = Map.add s 1 m in

      let (i, j, p) = ijp in
      let i = i - 1 in
      let j = j + 1 in
      let p = p + 1 in

      let ijp = (i, j, p) in
      let abum = (a, b, used, m) in

      if (ijp.(1) < num) then (Left abum, ijp)
      else (Right abum, ijp)
    ) abum ijp;
  in

  let i = 0 in
  let j = 0 in
  let abmij = (abum.(0), abum.(1), abum.(3), i, j) in
  let backbo = (back, b_o) in

  let abmij, _ =
    Loop.left (fun (abmij, backbo) ->
      let t = abmij.(3) + abmij.(4) in

      let abmijbackbo =
        if (abmij.(3) = abmij.(4) || t = (num - 1)) then
          let abmij =
            if (backbo.(0) = false) then
              let abmij =
                if (abmij.(4) < (num - 1)) then
                  let j = abmij.(4) + 1 in
                  (abmij.(0), abmij.(1), abmij.(2), abmij.(3), j)
                else
                  let i = abmij.(3) + 1 in
                  let j = 0 in
                  (abmij.(0), abmij.(1), abmij.(2), i, j)
              in
```

```
              abmij
            else
              let abmij =
                if (abmij.(4) > 0) then
                  let j = abmij.(4) - 1 in
                  (abmij.(0), abmij.(1), abmij.(2), abmij.(3), j)
                else
                  let i = abmij.(3) - 1 in
                  let j = num - 1 in
                  (abmij.(0), abmij.(1), abmij.(2), i, j)
              in
              abmij
          in
          (abmij.(0), abmij.(1), abmij.(2), abmij.(3), abmij.(4), backbo.(0), backbo.(1))
        else
          let used = Map [] in
          let use = Map [] in
          let k = 0 in
          let a = abmij.(0) in
          let b = abmij.(1) in
          let m = abmij.(2) in
          let i = abmij.(3) in
          let j = abmij.(4) in

          let uu = (used, use) in

          let uu, _ =
            Loop.left (fun (uu, k) ->

            let used = uu.(0) in
            let use = uu.(1) in

            let uu =
              if (k <> j) then
                let s = i * num + k in

                let p = match Map.find s a with
                  | None -> num
                  | Some v -> v
                in

                let used =
                  if (p <> num) then
                    Map.add p 1 used
                  else
                    used
                in

                let p = match Map.find s b with
                  | None -> num
                  | Some v -> v
                in

                let use =
                  if (p <> num) then
                    Map.add p 1 use
                  else
                    use
                in

                (used, use)
```

```
              else
                (used, use)
            in

            let used = uu.(0) in
            let use = uu.(1) in

            let uu =
              if (k <> i) then
                let s = k * num + j in

                let p = match Map.find s a with
                  | None -> num
                  | Some v -> v
                in

                let used =
                  if (p <> num) then
                    Map.add p 1 used
                  else
                    used
                in

                let p = match Map.find s b with
                  | None -> num
                  | Some v -> v
                in

                let use =
                  if (p <> num) then
                    Map.add p 1 use
                  else
                    use
                in

                (used, use)
              else
                (used, use)
            in

            let k = k + 1 in

            if (k < num) then (Left uu, k)
            else (Right uu, k)
            ) uu k;
          in

          let used = uu.(0) in
          let use = uu.(1) in

          let used =
            if (i = 1) & (j = 0) then
              let k = 0 in

              let (q, r) = match num / 2 with
              | Some qr -> qr
              | None -> failwith "division by 0 impossible" in

              let t =
                if (r = (0 : nat)) then q else (num - 1)
              in
```

```

              let used, _ =
                Loop.left (fun (used, k) ->

                let used = Map.add k 1 used in
                let k = k + 1 in

                if (k < t) then (Left used, k)
                else (Right used, k)
                ) used k;
              in
              used
            else
              used
          in

          let use =
            if (i = 1) & (j = 2) then
              let k = 0 in

              let (q, _) = match num / 2 with
              | Some qr -> qr
              | None -> failwith "division by 0 impossible" in
              let t = q - 1 in

              let use = if (t > 0) then
                let use, _ =
                  Loop.left (fun (use, k) ->

                  let use = Map.add k 1 use in
                  let k = k + 1 in

                  if (k < t) then (Left use, k)
                  else (Right use, k)
                  ) use k;
                in
                use
              else
                use
              in

              use
            else
              use
          in

          let use =
            if (i = 1) & (j = 3) then
              let k = 0 in
              let t = 3 in

              let use, _ =
                Loop.left (fun (use, k) ->

                let use = Map.add k 1 use in
                let k = k + 1 in

                if (k < t) then (Left use, k)
                else (Right use, k)
                ) use k;
              in
```

```
              use
            else
              use
          in

          let abmkt = (a, b, m, 0, 0) in

          let abmkt =
            if (backbo.(0) = true) then
              let s = i * num + j in
              let k = match Map.find s abmkt.(0) with
                | None -> num
                | Some v -> v
              in
              let t = match Map.find s abmkt.(1) with
                | None -> num
                | Some v -> v
              in
              let a = Map.add s num abmkt.(0) in
              let b = Map.add s num abmkt.(1) in

              let s = k * num + t in
              let m = Map.add s 0 abmkt.(2) in

              let tk = (t, k) in
              let tk =
                if (backbo.(1) = 2) then
                  (tk.(0) + 1, tk.(1) + 1)
                else
                  let tk =
                    if (backbo.(1) = 1) then
                      (0, tk.(1) + 1)
                    else
                      (tk.(0) + 1, tk.(1))
                  in
                  tk
              in
              (a, b, m, tk.(1), tk.(0))
            else
              (abmkt.(0), abmkt.(1), abmkt.(2), 0, 0)
          in

          let b_o = backbo.(1) in
          let b_o =
            if (backbo.(0) = true) then
              b_o
            else
              0
          in
          let backbo = (backbo.(0), b_o) in

          let k = abmkt.(3) in
          let t = abmkt.(4) in

          let k, _ =
            Loop.left (fun (k, used) ->

              let f = match Map.find k used with
                | None -> 0
                | Some v -> v
              in
```

```

              if (f = 1) then (Left (k + 1), used)
              else (Right k, used)
            ) k used;
          in

          let t, _ =
            Loop.left (fun (t, use) ->

              let f = match Map.find t use with
                | None -> 0
                | Some v -> v
              in

              if (f = 1) then (Left (t + 1), use)
              else (Right t, use)
            ) t use;
          in

          let abmijbackbo =
            if ((k < num & t < num) || (backbo.(0) = true & k < num)) then
              let tk =
                if (t >= num) then
                  (0, k + 1)
                else
                  (t, k)
              in

              let t = tk.(0) in
              let k = tk.(1) in

              let abmktij = (abmkt.(0), abmkt.(1), abmkt.(2), k, t, i, j) in
              let bb = false in

              let abmktij, bb =
                Loop.left (fun (abmktij, _) ->

                  let k = abmktij.(3) in
                  let k, _ =
                    Loop.left (fun (k, used) ->

                      let f = match Map.find k used with
                        | None -> 0
                        | Some v -> v
                      in

                    if (f = 1) then (Left (k + 1), used)
                    else (Right k, used)
                    ) k used;
                  in

                  let t = abmktij.(4) in
                  let t, _ =
                    Loop.left (fun (t, use) ->

                      let f = match Map.find t use with
                        | None -> 0
                        | Some v -> v
                      in

                    if (f = 1) then (Left (t + 1), use)
```

```
                    else (Right t, use)
                    ) t use;
                  in

                  let abmktbb =
                    if (t >= num || k >= num) then
                      (abmktij.(0), abmktij.(1), abmktij.(2), k + 1, 0, false)
                    else
                      let s = k * num + t in
                      let f = match Map.find s abmktij.(2) with
                        | None -> 0
                        | Some v -> v
                      in

                      let abmktbb =
                        if (f <> 1) then
                          let m = Map.add s 1 abmktij.(2) in
                          let s = abmktij.(5) * num + abmktij.(6) in
                          let a = Map.add s k abmktij.(0) in
                          let b = Map.add s t abmktij.(1) in
                          (a, b, m, k, t, true)
                        else
                          let t = t + 1 in
                          let tk = (t, k) in
                          let tk =
                            if (t >= num) then
                              (0, k + 1)
                            else
                              (tk)
                          in
                          (abmktij.(0), abmktij.(1), abmktij.(2), tk.(1), tk.(0), false)
                      in

                      abmktbb
                  in

                  let bb = abmktbb.(5) in
                  let abmktij =
                    (abmktbb.(0), abmktbb.(1), abmktbb.(2), abmktbb.(3), abmktbb.(4), abmktij.(5), abmktij.(6))
                  in

                  if (abmktij.(3) < num & bb = false) then (Left abmktij, bb)
                  else (Right abmktij, bb)
                ) abmktij bb;
              in

              let ijbackbo =
                if (bb = true) then
                  let ij =
                    if (abmktij.(6) < (num - 1)) then
                      (abmktij.(5), abmktij.(6) + 1)
                    else
                      (abmktij.(5) + 1, 0)
                  in
                  (ij.(0), ij.(1), false, backbo.(1))
                else
                  let ij =
                    if (abmktij.(6) > 0) then
                      (abmktij.(5), abmktij.(6) - 1)
                    else
```

```
                      (abmktij.(5) - 1, num - 1)
                  in
                  (ij.(0), ij.(1), true, 1)
              in
              (abmktij.(0), abmktij.(1), abmktij.(2), ijbackbo.(0), ijbackbo.(1), ijbackbo.(2), ijbackbo.(3))

            else
              let b_o =
                if (backbo.(0) = false) then
                  let b_o =
                    if (k >= num & t >= num) then
                      2
                    else
                      let b_o =
                        if (k >= num) then
                          1
                        else
                          0
                      in
                      b_o
                  in
                  b_o
                else
                  0
              in

              let ij =
                if (j > 0) then
                  (i, j - 1)
                else
                  (i - 1, num - 1)
              in

              (abmkt.(0), abmkt.(1), abmkt.(2), ij.(0), ij.(1), true, b_o)
          in

          abmijbackbo
      in

      let abmij = (abmijbackbo.(0), abmijbackbo.(1), abmijbackbo.(2), abmijbackbo.(3), abmijbackbo.(4)) in

      let backbo = (abmijbackbo.(5), abmijbackbo.(6)) in

      if (abmij.(3) < (num - 1) || abmij.(4) < (num - 1)) then (Left abmij, backbo)
      else (Right abmij, backbo)
    ) abmij backbo;
  in

  let m = Map [] in
  let abij = (abmij.(0), abmij.(1), 0, 0) in

  let m, _ =
    Loop.left (fun (m, abij) ->
      let s = abij.(2) * num + abij.(3) in
      let t = match Map.find s abij.(0) with
        | None -> 0
        | Some v -> v
      in
      let k = match Map.find s abij.(1) with
```

```
          | None -> 0
          | Some v -> v
        in
        let p = t * num + k + 1 in

        let m = Map.add s p m in

        let ij =
          if (abij.(3) < (num - 1)) then
            (abij.(2), abij.(3) + 1)
          else
            (abij.(2) + 1, 0)
        in

        let abij = (abij.(0), abij.(1), ij.(0), ij.(1)) in

        if (abij.(2) < num) then (Left m, abij)
        else (Right m, abij)
      ) m abij;
    in

    ( [], m)
```

Listing A.40: MagicSquare in Liquidity

```
type storage is map (int, int)

type parameter is
  MagicSquare of int

type return is list (operation) * storage

type q is map (int, int)

function access (const k : int; const m : q) : int is
  case m[k] of
    Some (val) -> val
  | None -> 0
  end

function accessN (const k : int; const m : q; const n : int) : int is
  case m[k] of
    Some (val) -> val
  | None -> n
  end

function magicsquare (const n : int) : storage is
  block {
      var a : q := map [];
      var b : q := map [];
      var used : q := map [];
      var use : q := map [];
      var m : q := map [];
      var i : int := 0;
      var j : int := 0;
      var s : int := 0;
      var k : int := 0;
      var t : int := 0;
      var p : int := 0;
```

```
    var back : bool := False;
    var b_o : int := 0;
    var bb : bool := False;

    t := (n - 1) / 2;
    k := (n - 1);

    while i < n block {
      if n mod 2 = 0n then {
        t := i;
      } else skip;

      s := i * n + j;
      p := i * n + k;
      a[s] := t;
      b[p] := t;
      i := i + 1;
      j := j + 1;
      k := k - 1;
    };

    i := n - 1;
    j := 0;
    p := 0;
    k := 0;
    t := (n - 1) / 2;

    if n mod 2 = 1n then {
      used[t] := 1;
      m[t * n + t] := 1;
    } else skip;

    while j < n block {
        if (i = j) and (n =/= 1) then {
            i := i - 1;
            j := j + 1;
            p := p + 1;
        } else skip;

        s := i * n + j;

        if n mod 2 = 1n then {
            for z := 0 to n block {
                if access(k, used) = 1 then
                  k := k + 1;
                else
                  z := n;
            }
        } else {
            for z := 0 to n block {
                if (access(k, used) = 1) or (k = i) or (k = j) then
                  k := k + 1;
                else
                  z := n;
            }
        };

        a[s] := k;
        used[k] := 1;

        s := k * n + access(s, b);
```

```
        m[s] := 1;

        s := p * n + j;
        b[s] := k;

        s := access(s, a) * n + k;
        m[s] := 1;

        i := i - 1;
        j := j + 1;
        p := p + 1;
        k := 0;
    };

    used := map [0->0];
    i := 0;
    j := 0;
    k := 0;

    while (i < (n - 1)) or (j < (n - 1)) block {
        t := i + j;

        if (i = j) or (t = (n - 1)) then {
            if (back = False) then {
                if (j < (n - 1)) then
                  j := j + 1;
                else {
                  i := i + 1;
                  j := 0;
                };
            } else {
                if (j > 0) then
                  j := j - 1;
                else {
                  i := i - 1;
                  j := n - 1;
                };
            };
        } else {
            k := 0;

            for z := 0 to n block {
                if k < n then {
                    if k =/= j then {
                        s := i * n + k;

                        p := accessN(s, a, n);
                        if p =/= n then
                          used[p] := 1;
                        else skip;

                        p := accessN(s, b, n);
                        if p =/= n then
                          use[p] := 1;
                        else skip;
                    } else skip;

                    if k =/= i then {
                        s := k * n + j;

                        p := accessN(s, a, n);
```

```
                        if p =/= n then
                          used[p] := 1;
                        else skip;

                        p := accessN(s, b, n);
                        if p =/= n then
                          use[p] := 1;
                        else skip;
                    } else skip;

                    k := k + 1;
                }
                else z := n;
            };

            if (i = 1) and (j = 0) then {
                k := 0;

                if (n mod 2 = 0n) then
                  t := n / 2;
                else
                  t := n - 1;

                for z := 0 to t block {
                    if (k < t) then {
                        used[k] := 1;
                        k := k + 1;
                    } else z := t;
                }
            } else skip;

            if (i = 1) and (j = 2) then {
                k := 0;
                t := n / 2 - 1;

                for z := 0 to t block {
                    if (k < t) then {
                        use[k] := 1;
                        k := k + 1;
                    } else z := t;
                }
            } else skip;

            if (i = 1) and (j = 3) then {
                k := 0;
                t := 3;

                for z := 0 to t block {
                    if (k < t) then {
                        use[k] := 1;
                        k := k + 1;
                    } else z := t;
                }
            } else skip;

            if (back = True) then {
                s := i * n + j;
                k := access(s, a);
                t := access(s, b);
                a[s] := n;
                b[s] := n;
```

```

        s := k * n + t;
        m[s] := 0;

        if (b_o = 2) then {
          k := k + 1;
          t := t + 1;
        } else {
          if (b_o = 1) then {
            k := k + 1;
            t := 0;
          } else t := t + 1;
        };
      } else {
        k := 0;
        t := 0;
        b_o := 0;
      };

      for z := 0 to n block {
        if access(k, used) = 1 then
         k := k + 1;
        else
         z := n;
      };

      for z := 0 to n block {
        if access(t, use) = 1 then
         t := t + 1;
        else
         z := n;
      };

      if ((k < n) and (t < n)) or ((back = True) and (k < n)) then {
        if (t >= n) then {
          t := 0;
          k := k + 1;
        } else skip;

        for zz := 0 to (n*n) block {
          if (k < n) and (bb = False) then {
           for z := 0 to n block {
             if access(k, used) = 1 then
               k := k + 1;
             else
               z := n;
           };

           for z := 0 to n block {
             if access(t, use) = 1 then
               t := t + 1;
             else
               z := n;
           };

           if (t >= n) or (k >= n) then {
             k := k + 1;
             t := 0;
           } else {
             s := k * n + t;
             if (access(s, m) =/= 1) then {
```

```
                              m[s] := 1;
                              s := i * n + j;
                              a[s] := k;
                              b[s] := t;

                              bb := True;
                          } else {
                              t := t + 1;

                              if (t >= n) then {
                                  k := k + 1;
                                  t := 0;
                              } else skip;
                          };
                      };

                    } else zz := n*n;
                };

                if (bb = True) then {
                    if (j < (n - 1)) then
                      j := j + 1;
                    else {
                      i := i + 1;
                      j := 0;
                    };

                    bb := False;
                    back := False;
                } else {
                    if (j > 0) then
                      j := j - 1;
                    else {
                      i := i - 1;
                      j := n - 1;
                    };

                    back := True;
                    b_o := 1;
                }
            } else {
                if (back = False) then {
                    if (k >= n) and (t >= n) then
                      b_o := 2
                    else {
                      if (k >= n) then
                          b_o := 1
                      else
                          b_o := 0
                    };
                } else b_o := 0;

                if (j > 0) then
                  j := j - 1;
                else {
                  i := i - 1;
                  j := n - 1;
                };

                back := True;
            };
```

```

              k := 0;
              used := map[0->0];
              use := map[0->0];
          };

      };

      m := map[0->0];
      i := 0;
      j := 0;

      while i < n block {
          s := i * n + j;
          t := access(s, a);
          k := access(s, b);
          p := t * n + k + 1;
          m[s] := p;

          if j < (n - 1) then
            j := j + 1;
          else {
            i := i + 1;
            j := 0;
          };
      }

  } with m

function main (const action : parameter; const store : storage) : return is
 ((nil : list (operation)),
  case action of
    MagicSquare (n) -> magicsquare (n)
  end)
```

Listing A.41: MagicSquare in PascaLIGO

```
type storage = (int, int) map

type parameter =
  MagicSquare of int

type return = operation list * storage

type q = (int, int) map

let access (k, m : int * q) : int =
  match Map.find_opt k m with
    Some value -> value
  | None -> 0

let accessN (k, m, n : int * q * int) : int =
  match Map.find_opt k m with
    Some value -> value
  | None -> n

let magicsquare (n : int) : storage =
    let a : q = Map.empty in
    let b : q = Map.empty in
```

```
    let used : q = Map.empty in
    let m : q = Map.empty in
    let i : int = 0 in
    let j : int = 0 in
    let s : int = 0 in
    let k : int = 0 in
    let t : int = 0 in
    let p : int = 0 in
    let back : bool = false in
    let b_o : int = 0 in
    let bb : bool = false in

    let t = (n - 1) / 2 in
    let k = (n - 1) in

    let rec diagonal_one(a, b, n, i, j, k : q * q * int * int * int * int) : (q * q) =

        let t =
            if n mod 2 = 0n then i else t
        in

        let s = i * n + j in
        let p = i * n + k in
        let a = Map.add s t a in
        let b = Map.add p t b in
        let i = i + 1 in
        let j = j + 1 in
        let k = k - 1 in

        if i < n then diagonal_one(a, b, n, i, j, k) else a, b
    in

    let a, b =
        if i < n then diagonal_one(a, b, n, i, j, k) else a, b
    in

    let i = n - 1 in
    let t = (n - 1) / 2 in

    let m, used =
        if n mod 2 = 1n then (
            Map.add (t * n + t) 1 m, Map.add t 1 used
        ) else m, used
    in

    let rec diagonal_two(a, b, used, m, n, i, j, p : q * q * q * q * int * int * int *
        int) : (q * q * q) =
        let i, j, p =
            if (i = j) && (n <> 1) then i - 1, j + 1, p + 1 else i, j, p
        in

        let s = i * n + j in
        let k = 0 in

        let k =
            if n mod 2 = 1n then (
                let rec while_dt_odd(used, k : q * int) : int =
                    let k = k + 1 in
                    let v = access(k, used) in

                    if v = 1 then while_dt_odd(used, k) else k
```

```
                in

                let v = access(k, used) in
                let k =
                    if v = 1 then while_dt_odd(used, k) else k
                in
                k
            ) else (
                let rec while_dt_even(used, k, i, j : q * int * int * int) : int =
                    let k = k + 1 in
                    let v = access(k, used) in

                    if (v = 1) || (k = i) || (k = j) then while_dt_even(used, k, i, j) else k
                in

                let v = access(k, used) in
                let k =
                    if (v = 1) || (k = i) || (k = j) then while_dt_even(used, k, i, j) else k
                in
                k
            )
        in

        let a = Map.add s k a in
        let used = Map.add k 1 used in

        let s = k * n + access(s, b) in
        let m = Map.add s 1 m in

        let s = p * n + j in
        let b = Map.add s k b in

        let s = access(s, a) * n + k in
        let m = Map.add s 1 m in

        let i = i - 1 in
        let j = j + 1 in
        let p = p + 1 in

        if j < n then diagonal_two(a, b, used, m, n, i, j, p) else a, b, m
    in

    let a, b, m =
        if j < n then diagonal_two(a, b, used, m, n, i, j, p) else a, b, m
    in

    let used : q = Map.empty in
    let i = 0 in
    let j = 0 in
    let k = 0 in

    let rec backtrack(a, b, m, n, i, j, b_o, back : q * q * q * int * int * int * int * bool) : (q * q) =
        let t = i + j in

        let a, b, m, i, j, b_o, back =
            if (i = j) || (t = (n - 1)) then (
                let i, j =
                    if back = false then (
```

```
                        if (j < (n - 1)) then i, j + 1 else i + 1, 0
                    ) else (
                        if (j > 0) then i, j - 1 else i - 1, n - 1
                    )
                in
                a, b, m, i, j, b_o, back
            ) else (
                let k = 0 in
                let used : q = Map.empty in
                let use : q = Map.empty in

                let rec while_mark_used(a, b, used, use, k, i, j, n : q*q*q*q*int*int*int*int) : (q * q) =
                    let used, use =
                        if (k <> j) then (
                            let s = i * n + k in

                            let p = accessN(s, a, n) in
                            let used =
                                if (p <> n) then Map.add p 1 used else used
                            in

                            let p = accessN(s, b, n) in
                            let use =
                                if (p <> n) then Map.add p 1 use else use
                            in
                            used, use
                        ) else used, use
                    in
                    let used, use =
                        if (k <> i) then (
                            let s = k * n + j in

                            let p = accessN(s, a, n) in
                            let used =
                                if (p <> n) then Map.add p 1 used else used
                            in

                            let p = accessN(s, b, n) in
                            let use =
                                if (p <> n) then Map.add p 1 use else use
                            in
                            used, use
                        ) else used, use
                    in

                    let k = k + 1 in

                    if (k < n) then while_mark_used(a, b, used, use, k, i, j, n) else used, use
                in

                let used, use =
                    if (k < n) then while_mark_used(a, b, used, use, k, i, j, n) else used, use
                in

                let used =
                    if (i = 1) && (j = 0) then (
                        let k = 0 in
                        let t = if (n mod 2 = 0n) then n / 2 else n - 1 in
```

```

                        let rec while_a10(used, k, t : q * int * int) : q =
                            let used = Map.add k 1 used in
                            let k = k + 1 in

                            if k < t then while_a10(used, k, t) else used
                        in
                        if k < t then while_a10(used, k, t) else used
                    ) else used
                in

                let use =
                    if (i = 1) && (j = 2) then (
                        let k = 0 in
                        let t = n / 2 - 1 in

                        let rec while_b12(use, k, t : q * int * int) : q =
                            let use = Map.add k 1 use in
                            let k = k + 1 in

                            if k < t then while_b12(use, k, t) else use
                        in
                        if k < t then while_b12(use, k, t) else use
                    ) else use
                in

                let use =
                    if (i = 1) && (j = 3) then (
                        let k = 0 in
                        let t = 3 in

                        let rec while_b13(use, k, t : q * int * int) : q =
                            let use = Map.add k 1 use in
                            let k = k + 1 in

                            if k < t then while_b13(use, k, t) else use
                        in
                        if k < t then while_b13(use, k, t) else use
                    ) else use
                in

                let a, b, m, k, t, b_o =
                    if back = true then (
                        let s = i * n + j in
                        let k = access(s, a) in
                        let t = access(s, b) in
                        let a = Map.add s n a in
                        let b = Map.add s n b in
                        let s = k * n + t in
                        let m = Map.add s 0 m in

                        let k, t =
                            if b_o = 2 then k + 1, t + 1 else (
                                if b_o = 1 then k + 1, 0 else k, t + 1
                            )
                        in

                        a, b, m, k, t, b_o
                    ) else (
                        a, b, m, 0, 0, 0
                    )
```

```
                in

                let rec while_find_used(used, k : q * int) : int =
                    let k = k + 1 in
                    let v = access(k, used) in
                    if (v = 1) then while_find_used(used, k) else k
                in

                let v = access(k, used) in
                let k =
                    if (v = 1) then while_find_used(used, k) else k
                in

                let rec while_find_use(use, t : q * int) : int =
                    let t = t + 1 in
                    let v = access(t, use) in
                    if (v = 1) then while_find_use(use, t) else t
                in

                let v = access(t, use) in
                let t =
                    if (v = 1) then while_find_use(use, t) else t
                in

                let a, b, m, i, j, b_o, back =
                    if ((k < n) && (t < n)) || ((back = true) && (k < n)) then (
                        let k, t = if t >= n then k + 1, 0 else k, t in

                        let rec while_pairs(a,b,m,used,use,n,i,j,k,t,bb : q*q*q*q*q*int*int*int*int*int*bool) : (q*q*q*bool) =
                            let v = access(k, used) in
                            let k =
                                if (v = 1) then while_find_used(used, k) else k
                            in

                            let v = access(t, use) in
                            let t =
                                if (v = 1) then while_find_use(use, t) else t
                            in

                            let a, b, m, k, t, bb =
                                if (t >= n) || (k >= n) then a, b, m, k + 1, 0, bb else (
                                    let s = k * n + t in
                                    let v = access(s, m) in
                                    let a, b, m, k, t, bb =
                                        if (v <> 1) then (
                                            let m = Map.add s 1 m in
                                            let s = i * n + j in
                                            let a = Map.add s k a in
                                            let b = Map.add s t b in

                                            a, b, m, k, t, true
                                        ) else (
                                            let t = t + 1 in
                                            if t >= n then a, b, m, k + 1, 0, bb
                                            else a, b, m, k, t, bb
                                        )
                                    in
                                    a, b, m, k, t, bb
                                )
                            in
```

```

                            if (k < n) && (bb = false) then
                            while_pairs(a,b,m,used,use,n,i,j,k,t,bb)
                            else a, b, m, bb
                        in
                        let a, b, m, bb =
                            if (k < n) && (bb = false) then
                            while_pairs(a,b,m,used,use,n,i,j,k,t,false)
                            else a, b, m, bb
                        in

                        let i, j, b_o, back =
                            if bb = true then (
                                if (j < (n - 1)) then i, j + 1, b_o, false
                                else i + 1, 0, b_o, false
                            ) else (
                                if j > 0 then i, j - 1, 1, true
                                else i - 1, n - 1, 1, true
                            )
                        in

                        a, b, m, i, j, b_o, back
                    ) else (
                        let b_o =
                            if back = false then (
                                if (k >= n) && (t >= n) then 2 else (
                                    if k >= n then 1 else 0
                                )
                            ) else 0
                        in

                        let i, j =
                            if j > 0 then i, j - 1 else i - 1, n - 1
                        in

                        let back = true in

                        a, b, m, i, j, b_o, back
                    )
                in

                a, b, m, i, j, b_o, back
            )
        in

        if (i < (n - 1)) || (j < (n - 1)) then backtrack(a,b,m,n,i,j,b_o,back) else a, b
    in

    let a, b =
        if (i < (n - 1)) || (j < (n - 1)) then backtrack(a,b,m,n,i,j,b_o,back) else a, b
    in

    let m : q = Map.empty in
    let i = 0 in
    let j = 0 in

    let rec final_while(a,b,m,n,i,j : q*q*q*int*int*int) : q =
        let s = i * n + j in
        let t = access(s, a) in
        let k = access(s, b) in
        let p = t * n + k + 1 in
```

```
        let m = Map.add s p m in

        let i, j =
            if j < (n - 1) then i, j + 1 else i + 1, 0
        in
        if i < n then final_while(a,b,m,n,i,j) else m
    in
    let m =
        if i < n then final_while(a,b,m,n,i,j) else m
    in

    m

let main (action, store : parameter * storage) : return =
 ([] : operation list),
 (match action with
   MagicSquare (n) -> magicsquare (n))
```

Listing A.42: MagicSquare in CameLIGO

```
type storage = map (int, int);

type parameter =
 MagicSquare (int)

type return = (list (operation), storage);

type q = map (int, int);

let access = ((k, m) : (int, q)) : int => {
  switch (Map.find_opt (k, m)) {
  | Some value => value
  | None => 0
  }
}

let accessN = ((k, m, n) : (int, q, int)) : int => {
  switch (Map.find_opt (k, m)) {
  | Some value => value
  | None => n
  }
}

let magicsquare = ((n) : (int)) : storage =>
    let a : q = Map.empty;
    let b : q = Map.empty;
    let used : q = Map.empty;
    let m : q = Map.empty;
    let i : int = 0;
    let j : int = 0;
    let s : int = 0;
    let k : int = 0;
    let t : int = 0;
    let p : int = 0;
    let back : bool = false;
    let b_o : int = 0;
    let bb : bool = false;

    let t = ((n - 1) / 2);
```

```
    let k = (n - 1);

    let rec diagonal_one = ((a, b, n, i, j, k) : (q, q, int, int, int, int)) : (q, q) =>

        let t = if (n mod 2 == 0n) {i;} else {t};

        let s = i * n + j;
        let p = i * n + k;
        let a = Map.add(s, t, a);
        let b = Map.add(p, t, b);
        let i = i + 1;
        let j = j + 1;
        let k = k - 1;

        if (i < n) {diagonal_one(a, b, n, i, j, k);} else {(a, b)};


    let a, b = if (i < n) {diagonal_one(a, b, n, i, j, k);} else {(a, b)};


    let i = n - 1;
    let t = (n - 1) / 2;

    let m, used =
        if (n mod 2 == 1n) {
            (Map.add((t * n + t), 1, m), Map.add(t, 1, used));
        } else {(m, used)};


    let rec diagonal_two = ((a, b, used, m, n, i, j, p) : (q, q, q, q, int, int, int, int
        )) : (q, q, q) =>
        let i, j, p = if ((i == j) && (n != 1)) {(i - 1, j + 1, p + 1)} else {(i, j, p)};

        let s = i * n + j;
        let k = 0;

        let k =
            if (n mod 2 == 1n) {
                let rec while_dt_odd = ((used, k) : (q, int)) : int =>
                    let k = k + 1;
                    let v = access(k, used);

                    if (v == 1) {while_dt_odd(used, k);} else {k};

                let v = access(k, used);
                let k = if (v == 1) {while_dt_odd(used, k);} else {k};

                k
            } else {
                let rec while_dt_even = ((used, k, i, j) : (q, int, int, int)) : int =>
                    let k = k + 1;
                    let v = access(k, used);

                    if ((v == 1) || (k == i) || (k == j)) {while_dt_even(used, k, i, j);}
                         else {k};

                let v = access(k, used);
                let k = if ((v == 1) || (k == i) || (k == j)) {while_dt_even(used, k, i,
                    j);} else {k};

                k
```

```
            };


        let a = Map.add(s, k, a);
        let used = Map.add(k, 1, used);

        let s = k * n + access(s, b);
        let m = Map.add(s, 1, m);

        let s = p * n + j;
        let b = Map.add(s, k, b);

        let s = access(s, a) * n + k;
        let m = Map.add(s, 1, m);

        let i = i - 1;
        let j = j + 1;
        let p = p + 1;


        if (j < n) {diagonal_two(a, b, used, m, n, i, j, p);} else {(a, b, m)};


    let a, b, m = if (j < n) {diagonal_two(a, b, used, m, n, i, j, p);} else {(a, b, m)};

    let used : q = Map.empty;
    let i = 0;
    let j = 0;
    let k = 0;

    let rec backtrack = ((a, b, m, n, i, j, b_o, back) : (q, q, q, int, int, int, int, bool)) : (q, q) =>
        let t = i + j;

        let a, b, m, i, j, b_o, back =
            if ((i == j) || (t == (n - 1))) {
                let i, j =
                    if (back == false) {
                        if (j < (n - 1)) {(i, j + 1);} else {(i + 1, 0)};
                    } else {
                        if (j > 0) {(i, j - 1)} else {(i - 1, n - 1)};
                    };

                (a, b, m, i, j, b_o, back)
            } else {
                let k = 0;
                let used : q = Map.empty;
                let use : q = Map.empty;

                let rec while_mark_used = ((a, b, used, use, k, i, j, n) : (q,q,q,q,int,int,int,int)) : (q, q) =>
                    let used, use =
                        if (k != j) {
                            let s = i * n + k;

                            let p = accessN(s, a, n);
                            let used = if (p != n) {Map.add(p, 1, used);} else {used};

                            let p = accessN(s, b, n);
                            let use = if (p != n) {Map.add(p, 1, use);} else {use};

```

```
                            (used, use)
                        } else {(used, use)};

                    let used, use =
                        if (k != i) {
                            let s = k * n + j;

                            let p = accessN(s, a, n);
                            let used = if (p != n) {Map.add(p, 1, used);} else {used};

                            let p = accessN(s, b, n);
                            let use = if (p != n) {Map.add(p, 1, use);} else {use};

                            (used, use)
                        } else {(used, use)};


                    let k = k + 1;

                    if (k < n) {while_mark_used(a, b, used, use, k, i, j, n)} else {(used, use)};


                let used, use = if (k < n) {while_mark_used(a, b, used, use, k, i, j, n)} else {(used, use)};


                let used =
                    if ((i == 1) && (j == 0)) {
                        let k = 0;
                        let t = if (n mod 2 == 0n) {n / 2} else {n - 1};

                        let rec while_a10 = ((used, k, t) : (q, int, int)) : q =>
                            let used = Map.add(k, 1, used);
                            let k = k + 1;

                            if (k < t) {while_a10(used, k, t);} else {used};

                        if (k < t) {while_a10(used, k, t);} else {used};
                    } else {(used)};

                let use =
                    if ((i == 1) && (j == 2)) {
                        let k = 0;
                        let t = n / 2 - 1;

                        let rec while_b12 = ((use, k, t) : (q, int, int)) : q =>
                            let use = Map.add(k, 1, use);
                            let k = k + 1;

                            if (k < t) {while_b12(use, k, t);} else {use};

                        if (k < t) {while_b12(use, k, t);} else {use};
                    } else {use};

                let use =
                    if ((i == 1) && (j == 3)) {
                        let k = 0;
                        let t = 3;

                        let rec while_b13 = ((use, k, t) : (q, int, int)) : q =>
```

```
                            let use = Map.add(k, 1, use);
                            let k = k + 1;

                            if (k < t) {while_b13(use, k, t);} else {use};

                        if (k < t) {while_b13(use, k, t);} else {use};
                    } else {use};

                let a, b, m, k, t, b_o =
                    if (back == true) {
                        let s = i * n + j;
                        let k = access(s, a);
                        let t = access(s, b);
                        let a = Map.add(s, n, a);
                        let b = Map.add(s, n, b);
                        let s = k * n + t;
                        let m = Map.add(s, 0, m);

                        let k, t =
                            if (b_o == 2) {(k + 1, t + 1);} else {
                                if (b_o == 1) {(k + 1, 0);} else {(k, t + 1)};
                            };

                        (a, b, m, k, t, b_o)
                    } else {
                        (a, b, m, 0, 0, 0)
                    };

                let rec while_find_used = ((used, k) : (q, int)) : int =>
                    let k = k + 1;
                    let v = access(k, used);
                    if (v == 1) {while_find_used(used, k);} else {k};

                let v = access(k, used);
                let k = if (v == 1) {while_find_used(used, k);} else {k};

                let rec while_find_use = ((use, t) : (q, int)) : int =>
                    let t = t + 1;
                    let v = access(t, use);
                    if (v == 1) {while_find_use(use, t);} else {t};

                let v = access(t, use);
                let t = if (v == 1) {while_find_use(use, t);} else {t};


                let a, b, m, i, j, b_o, back =
                    if (((k < n) && (t < n)) || ((back == true) && (k < n))) {
                        let k, t = if (t >= n) {(k + 1, 0)} else {(k, t)};

                        let rec while_pairs = ((a,b,m,used,use,n,i,j,k,t,bb) : (q,q,q,q,q
                            ,int,int,int,int,int,bool)) : (q,q,q,bool) =>
                            let v = access(k, used);
                            let k = if (v == 1) {while_find_used(used, k)} else {k};

                            let v = access(t, use);
                            let t = if (v == 1) {while_find_use(use, t)} else {t};


                            let a, b, m, k, t, bb =
                                if ((t >= n) || (k >= n)) {(a, b, m, k + 1, 0, bb)} else
                                    {
```

```
                                    let s = k * n + t;
                                    let v = access(s, m);
                                    let a, b, m, k, t, bb =
                                        if (v != 1) {
                                            let m = Map.add(s, 1, m);
                                            let s = i * n + j;
                                            let a = Map.add(s, k, a);
                                            let b = Map.add(s, t, b);

                                            (a, b, m, k, t, true)
                                        } else {
                                            let t = t + 1;
                                            if (t >= n) {(a, b, m, k + 1, 0, bb)}
                                            else {(a, b, m, k, t, bb)};
                                        };

                                    (a, b, m, k, t, bb)
                                };


                            if ((k < n) && (bb == false)) {
                            while_pairs(a,b,m,used,use,n,i,j,k,t,bb)
                            } else {(a, b, m, bb)};

                        let a, b, m, bb =
                            if ((k < n) && (bb == false)) {
                            while_pairs(a,b,m,used,use,n,i,j,k,t,false)
                            } else {(a, b, m, bb)};


                        let i, j, b_o, back =
                            if (bb == true) {
                                if (j < (n - 1)) {(i, j + 1, b_o, false)}
                                else {(i + 1, 0, b_o, false)};
                            } else {
                                if (j > 0) {(i, j - 1, 1, true)}
                                else {(i - 1, n - 1, 1, true)};
                            };


                        (a, b, m, i, j, b_o, back)
                    } else {
                        let b_o =
                            if (back == false) {
                                if ((k >= n) && (t >= n)) {2} else {
                                    if (k >= n) {1} else {0};
                                };
                            } else {0};


                        let i, j = if (j > 0) {(i, j - 1)} else {(i - 1, n - 1)};

                        let back = true;

                        (a, b, m, i, j, b_o, back)
                    };


                (a, b, m, i, j, b_o, back)
            };

```

```
        if ((i < (n - 1)) || (j < (n - 1))) {backtrack(a,b,m,n,i,j,b_o,back)} else {(a, b)};


    let a, b = if ((i < (n - 1)) || (j < (n - 1))) {backtrack(a,b,m,n,i,j,b_o,back)} else {(a, b)};


    let m : q = Map.empty;
    let i = 0;
    let j = 0;

    let rec final_while = ((a,b,m,n,i,j) : (q,q,q,int,int,int)) : q =>
        let s = i * n + j;
        let t = access(s, a);
        let k = access(s, b);
        let p = t * n + k + 1;
        let m = Map.add(s, p, m);

        let i, j = if (j < (n - 1)) {(i, j + 1)} else {(i + 1, 0)};

        if (i < n) {final_while(a,b,m,n,i,j)} else {m};

    let m = if (i < n) {final_while(a,b,m,n,i,j)} else {m};

    m;

let main = ((action, store) : (parameter, storage)) : return => {
 (([] : list (operation)),
 (switch (action) {
  | MagicSquare (n) => magicsquare (n)}))
};
```

Listing A.43: MagicSquare in ReasonLIGO

```
archetype MagicSquare

variable mag : map<int, int> = []

entry magicsquare (num : int) {
    var a : map<int, int> = [];
    var b : map<int, int> = [];
    var used : map<int, int> = [];
    var us : map<int, int> = [];
    var m : map<int, int> = [];
    var n : int = num;
    var i : int = 0;
    var j : int = 0;
    var s : int = 0;
    var p : int = 0;
    var back : bool = false;
    var b_o : int = 0;
    var bb : bool = false;

    var t = floor((n - 1) / 2);
    var k = n - 1;

    iter ii to n do
        if i < n then (
```

```
            if n % 2 = 0 then (
                t := i;
            );

            s := i * n + j;
            p := i * n + k;
            a := put(a, s, t);
            b := put(b, p, t);
            i := i + 1;
            j := j + 1;
            k := k - 1;
        );
    done;

    i := n - 1;
    j := 0;
    p := 0;
    k := 0;
    t := floor((n - 1) / 2);

    if n % 2 = 1 then (
        used := put(used, t, 1i);
        m := put(m, (t * n + t), 1i);
    );

    iter jj to n do
        if j < n then (
            if (i = j) and (n <> 1) then (
                i := i - 1;
                j := j + 1;
                p := p + 1;
            );

            s := i * n + j;

            if n % 2 = 1 then (
                iter uu to n do
                    if contains(used, k) = true then k := k + 1
                done;
            ) else (
                iter uu to n do
                    if (contains(used, k) = true or k = i or k = j) then k := k + 1
                done;
            );

            a := put(a, s, k);
            used := put(used, k, 1i);

            s := k * n + b[s];
            m := put(m, s, 1i);

            s := p * n + j;
            b := put(b, s, k);

            s := a[s] * n + k;
            m := put(m, s, 1i);

            i := i - 1;
            j := j + 1;
            p := p + 1;
            k := 0;
```

```
        );
    done;

    used := [];
    i := 0;
    j := 0;
    k := 0;

    var it : int = 0;
    if (n = 5) then it := 44 else (
        if (n = 7) then it := 3862 else (
            it := n * n;
        );
    );

    var pp : option<int> = none;

    iter baba to it do
        if i < (n - 1) or j < (n - 1) then (
            t := i + j;

            if i = j or t = (n - 1) then (
                if back = false then (
                    if j < (n - 1) then j := j + 1 else (
                        i := i + 1;
                        j := 0;
                    );
                ) else (
                    if j > 0 then j := j - 1 else (
                        i := i - 1;
                        j := n - 1;
                    );
                );
            ) else (
                k := 0;

                iter trtr to n do
                    if k < n then (
                        if k <> j then (
                            s := i * n + k;
                            pp := getopt(a, s);
                            p :=
                            match pp with
                            | some(v) -> v
                            | none -> n
                            end;
                            if p <> n then used := put(used, p, 1i);

                            pp := getopt(b, s);
                            p :=
                            match pp with
                            | some(v) -> v
                            | none -> n
                            end;
                            if p <> n then us := put(us, p, 1i);
                        );

                        if k <> i then (
                            s := k * n + j;

                            pp := getopt(a, s);
```

```
                            p :=
                            match pp with
                            | some(v) -> v
                            | none -> n
                            end;
                            if p <> n then used := put(used, p, 1i);

                            pp := getopt(b, s);
                            p :=
                            match pp with
                            | some(v) -> v
                            | none -> n
                            end;
                            if p <> n then us := put(us, p, 1i);
                        );

                        k := k + 1;
                    );
                done;

                if i = 1 and j = 0 then (
                    k := 0;

                    if n % 2 = 0 then t := floor(n / 2) else t := n - 1;

                    iter ij to t do
                        if (k < t) then (
                            used := put(used, k, 1i);
                            k := k + 1;
                        );
                    done;
                );

                if i = 1 and j = 2 then (
                    k := 0;

                    t := floor(n / 2) - 1;

                    iter ij to t do
                        if (k < t) then (
                            us := put(us, k, 1i);
                            k := k + 1;
                        );
                    done;
                );

                if i = 1 and j = 3 then (
                    k := 0;
                    t := 3;

                    iter ij to t do
                        if (k < t) then (
                            us := put(us, k, 1i);
                            k := k + 1;
                        );
                    done;
                );

                if back = true then (
                    s := i * n + j;
                    k := a[s];
```

```
                    t := b[s];
                    a := put(a, s, n);
                    b := put(b, s, n);

                    s := k * n + t;
                    m := put(m, s, 0i);

                    if (b_o = 2) then (
                        k := k + 1;
                        t := t + 1;
                    ) else (
                        if (b_o = 1) then (
                            k := k + 1;
                            t := 0;
                        ) else t := t + 1;
                    );
                ) else (
                    k := 0;
                    t := 0;
                    b_o := 0;
                );

                iter kk to n do
                    if contains(used, k) = true then k := k + 1;
                done;

                iter tt to n do
                    if contains(us, t) = true then t := t + 1;
                done;

                if ((k < n and t < n) or (back = true and k < n)) then (
                    if (t >= n) then (
                        t := 0;
                        k := k + 1;
                    );

                    var ti : int = 0;
                    if n = 5 then ti := 3 else (
                        if n = 7 then ti := 5 else (
                            ti := 1
                        )
                    );

                    iter zx to ti do
                        if k < n and bb = false then (
                            iter kk to n do
                                if contains(used, k) = true then k := k + 1;
                            done;

                            iter tt to n do
                                if contains(us, t) = true then t := t + 1;
                            done;

                            if t >= n or k >= n then (
                                k := k + 1;
                                t := 0;
                            ) else (
                                s := k * n + t;
                                pp := getopt(m, s);
                                p :=
                                match pp with
```

```
                                | some(v) -> v
                                | none -> 0
                                end;
                                if p <> 1 then (
                                    m := put(m, s, 1i);
                                    s := i * n + j;
                                    a := put(a, s, k);
                                    b := put(b, s, t);

                                    bb := true;
                                ) else (
                                    t := t + 1;

                                    if t >= n then (
                                        k := k + 1;
                                        t := 0;
                                    );
                                );
                            );
                        );
                    done;

                    if bb = true then (
                        if j < (n - 1) then j := j + 1 else (
                            i := i + 1;
                            j := 0;
                        );

                        bb := false;
                        back := false;
                    ) else (
                        if j > 0 then j := j - 1 else (
                            i := i - 1;
                            j := n - 1;
                        );

                        back := true;
                        b_o := 1;
                    );
                ) else (
                    if back = false then (
                        if k >= n and t >= n then b_o := 2 else (
                            if k >= n then b_o := 1 else b_o := 0;
                        );
                    ) else b_o := 0;

                    if j > 0 then j := j - 1 else (
                        i := i - 1;
                        j := n - 1;
                    );

                    back := true;
                );

                k := 0;
                used := [];
                us := [];
            );
        );
    done;

```

```
    m := [];
    i := 0;
    j := 0;

    iter mm to n*n do
        if i < n then (
            s := i * n + j;
            t := a[s];
            k := b[s];
            p := t * n + k + 1;
            m := put(m, s, p);

            if j < (n - 1) then j := j + 1 else (
                i := i + 1;
                j := 0;
            );
        );
    done;

    mag := m
}
```

Listing A.44: MagicSquare in Archetype

```
pragma solidity ^0.5.0;

contract MagicSquare {

    uint[] mag;

    function magic_square(uint num) public {
        uint[] memory a = new uint[](num*num);
        uint[] memory b = new uint[](num*num);
        uint[] memory m = new uint[](num*num);

        if (num == 1 || num == 2)
        {
            a = new uint[]((num+1)*(num+1));
            b = new uint[]((num+1)*(num+1));
            m = new uint[]((num+1)*(num+1));
        }

        uint[] memory used = new uint[](num + 3);
        uint[] memory use = new uint[](num + 3);
        uint i = 0;
        uint j = 0;
        uint s = 0;
        uint k = 0;
        uint t = 0;
        uint p = 0;
        bool back = false;
        uint b_o = 0;

        t = (num - 1) / 2;
        k = (num - 1);

        while (i < num)
        {
            if (num % 2 == 0)
```

```
                t = i;

            s = i * num + j;
            p = i * num + k;
            a[s] = t;
            b[p] = t;
            i = i + 1;
            j = j + 1;
            k = k - 1;
        }

        i = num - 1;
        j = 0;
        k = 0;
        t = (num - 1) / 2;

        if (num % 2 == 1)
        {
            used[t] = 1;
            p = t * num + t;
            m[p] = 1;
        }

        p = 0;

        while (j < num)
        {
            if (i == j && num != 1)
            {
                i = i - 1;
                j = j + 1;
                p = p + 1;
            }

            s = i * num + j;

            if (num % 2 == 1)
            {
                while (used[k] == 1)
                    k = k + 1;
            }
            else
            {
                while (used[k] == 1 || k == i || k == j)
                    k = k + 1;
            }

            a[s] = k;
            used[k] = 1;

            s = k * num + b[s];
            m[s] = 1;

            s = p * num + j;
            b[s] = k;

            s = a[s] * num + k;
            m[s] = 1;

            i = i - 1;
            j = j + 1;
```

```
            p = p + 1;
            k = 0;
        }

        used = new uint[](num + 3);
        i = 0;
        j = 0;
        k = 0;


        while (i < (num - 1) || j < (num - 1))
        {
            t = i + j;

            if (i == j || t == (num - 1))
            {
                if (back == false)
                {
                    if (j < (num - 1))
                        j = j + 1;
                    else
                    {
                        i = i + 1;
                        j = 0;
                    }
                }
                else
                {
                    if (j > 0)
                        j = j - 1;
                    else
                    {
                        i = i - 1;
                        j = num - 1;
                    }
                }
            }
            else
            {
                k = 0;

                while (k < num)
                {
                    if (k != j)
                    {
                        s = i * num + k;

                        if (k < j || (i == k || (i + k) == (num - 1)))
                        {
                            p = a[s];
                            used[p] = 1;
                        }

                        if (k < j || (i == k || (i + k) == (num - 1)))
                        {
                            p = b[s];
                            use[p] = 1;
                        }
                    }

                    if (k != i)
```

```
                    {
                        s = k * num + j;

                        if (k < i || (k == j || (k + j) == (num - 1)))
                        {
                            p = a[s];
                            used[p] = 1;
                        }

                        if (k < i || (k == j || (k + j) == (num - 1)))
                        {
                            p = b[s];
                            use[p] = 1;
                        }
                    }

                    k = k + 1;
                }

                if (i == 1 && j == 0)
                {
                    k = 0;

                    if (num % 2 == 0)
                        t = num / 2;
                    else
                        t = num - 1;

                    while (k < t)
                    {
                        used[k] = 1;
                        k = k + 1;
                    }
                }

                if (i == 1 && j == 2)
                {
                    k = 0;

                    t = (num / 2) - 1;

                    while (k < t)
                    {
                        use[k] = 1;
                        k = k + 1;
                    }
                }

                if (i == 1 && j == 3)
                {
                    k = 0;
                    t = 3;

                    while (k < t)
                    {
                        use[k] = 1;
                        k = k + 1;
                    }
                }

                if (back == true)
```

```
                {
                    s = i * num + j;
                    k = a[s];
                    t = b[s];
                    a[s] = num;
                    b[s] = num;

                    s = k * num + t;
                    m[s] = 0;

                    if (b_o == 2)
                    {
                        k = k + 1;
                        t = t + 1;
                    }
                    else
                    {
                        if (b_o == 1)
                        {
                            k = k + 1;
                            t = 0;
                        }
                        else
                            t = t + 1;
                    }
                }
                else
                {
                    k = 0;
                    t = 0;
                    b_o = 0;
                }

                while (used[k] == 1)
                    k = k + 1;

                while (use[t] == 1)
                    t = t + 1;


                if ((k < num && t < num) || (back == true && k < num))
                {
                    if (t >= num)
                    {
                        t = 0;
                        k = k + 1;
                    }

                    back = false;

                    while (k < num && back == false)
                    {
                        while (used[k] == 1)
                            k = k + 1;

                        while (use[t] == 1)
                            t = t + 1;

                        if (t >= num || k >= num)
                        {
                            k = k + 1;
```

```
                            t = 0;
                        }
                        else
                        {
                            s = k * num + t;

                            if (m[s] != 1)
                            {
                                m[s] = 1;
                                s = i * num + j;
                                a[s] = k;
                                b[s] = t;

                                back = true;
                            }
                            else
                            {
                                t = t + 1;

                                if (t >= num)
                                {
                                    k = k + 1;
                                    t = 0;
                                }
                            }
                        }
                    }

                    if (back == true)
                    {
                        if (j < (num - 1))
                            j = j + 1;
                        else
                        {
                            i = i + 1;
                            j = 0;
                        }

                        back = false;
                    }
                    else
                    {
                        if (j > 0)
                            j = j - 1;
                        else
                        {
                            i = i - 1;
                            j = num - 1;
                        }

                        back = true;
                        b_o = 1;
                    }
                }
                else
                {
                    if (back == false)
                    {
                        if (k >= num && t >= num)
                            b_o = 2;
                        else
```

```
                        {
                            if (k >= num)
                                b_o = 1;
                            else
                                b_o = 0;
                        }
                    }
                    else
                        b_o = 0;

                    if (j > 0)
                        j = j - 1;
                    else
                    {
                        i = i - 1;
                        j = num - 1;
                    }

                    back = true;
                }

                k = 0;
                used = new uint[](num + 3);
                use = new uint[](num + 3);
            }
        }

        m = new uint[](num*num);
        i = 0;
        j = 0;

        while (i < num)
        {
            s = i * num + j;
            t = a[s];
            k = b[s];
            p = t * num + k + 1;
            m[s] = p;

            if (j < (num - 1))
                j = j + 1;
            else
            {
                i = i + 1;
                j = 0;
            }
        }

        mag = m;
    }

    function getPositions() public view returns (uint[] memory) {
        return mag;
    }
}
```

Listing A.45: MagicSquare in Solidity

```
mag: public(uint256[49])
```

```

@external
def magic_square (num : uint256):
    a: uint256[49] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    b: uint256[49] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    used: uint256[9] = [0,0,0,0,0,0,0,0,0]
    use: uint256[9] = [0,0,0,0,0,0,0,0,0]
    m: uint256[49] = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    i: uint256 = 0
    j: uint256 = 0
    s: uint256 = 0
    k: uint256 = 0
    t: uint256 = 0
    p: uint256 = 0
    back: bool = False
    b_o: uint256 = 0

    t = (num - 1) / 2
    k = num - 1

    for x in range(8):
        if i < num:
          if num % 2 == 0:
            t = i

          s = i * num + j
          p = i * num + k
          a[s] = t
          b[p] = t
          i = i + 1
          j = j + 1
          if k > 0:
            k = k - 1
        else:
            break

    i = num - 1
    j = 0
    p = 0
    k = 0
    t = (num - 1) / 2

    if num % 2 == 1:
      used[t] = 1
      m[t * num + t] = 1

    for z in range(8):
      if j < num:
        if i == j and num != 1:
          if i > 0:
            i = i - 1
          j = j + 1
          p = p + 1

        s = i * num + j

        if num % 2 == 1:
          for y in range(8):
```

```
          if used[k] == 1:
            k = k + 1
          else:
            break
      else:
        for y in range(8):
          if used[k] == 1 or k == i or k == j:
            k = k + 1
          else:
            break

      a[s] = k
      used[k] = 1

      s = k * num + b[s]
      m[s] = 1

      s = p * num + j
      b[s] = k

      s = a[s] * num + k
      m[s] = 1

      if i > 0:
        i = i - 1
      j = j + 1
      p = p + 1
      k = 0
    else:
      break

  used = [0,0,0,0,0,0,0,0,0]
  i = 0
  j = 0
  k = 0

  for w in range(4000):
    if i < num - 1 or j < num - 1:
      t = i + j

      if i == j or t == num - 1:
        if back == False:
          if j < num - 1:
            j = j + 1
          else:
            i = i + 1
            j = 0
        else:
          if j > 0:
            j = j - 1
          else:
            i = i - 1
            j = num - 1
      else:
        k = 0

        for v in range(8):
          if k < num:
            if k != j:
              s = i * num + k

```

```
                if k < j or (i == k or i + k == num - 1):
                  p = a[s]
                  used[p] = 1
                  p = b[s]
                  use[p] = 1

              if k != i:
                s = k * num + j

                if k < i or (k == j or k + j == num - 1):
                  p = a[s]
                  used[p] = 1
                  p = b[s]
                  use[p] = 1

              k = k + 1
            else:
              break

          if i == 1 and j == 0:
            k = 0

            if num % 2 == 0:
              t = num / 2
            else:
              t = num - 1

            for u in range(7):
              if k < t:
                used[k] = 1
                k = k + 1
              else:
                break

          if i == 1 and j == 2:
            k = 0

            t = num / 2 - 1

            for uu in range(4):
              if k < t:
                use[k] = 1
                k = k + 1
              else:
                break

          if i == 1 and j == 3:
            k = 0
            t = 3

            for uuu in range(4):
              if k < t:
                use[k] = 1
                k = k + 1
              else:
                break

          if back == True:
            s = i * num + j
            k = a[s]
            t = b[s]
```

```
            a[s] = num
            b[s] = num

            s = k * num + t
            m[s] = 0

            if b_o == 2:
              k = k + 1
              t = t + 1
            else:
              if b_o == 1:
                k = k + 1
                t = 0
              else:
                t = t + 1

          else:
            k = 0
            t = 0
            b_o = 0

          for kk in range(8):
            if used[k] == 1:
              k = k + 1
            else:
              break

          for tt in range(8):
            if use[t] == 1:
              t = t + 1
            else:
              break

          if (k < num and t < num) or (back == True and k < num):
            if t >= num:
              t = 0
              k = k + 1

            back = False

            for qq in range(49):
              if k < num and back == False:
                for kk in range(8):
                  if used[k] == 1:
                    k = k + 1
                  else:
                    break

                for tt in range(8):
                  if use[t] == 1:
                    t = t + 1
                  else:
                    break

                if t >= num or k >= num:
                  k = k + 1
                  t = 0
                else:
                  s = k * num + t

                  if m[s] != 1:
```

```
                    m[s] = 1
                    s = i * num + j
                    a[s] = k
                    b[s] = t

                    back = True
                  else:
                    t = t + 1

                    if t >= num:
                      k = k + 1
                      t = 0

              else:
                break

            if back == True:
              if j < num - 1:
                j = j + 1
              else:
                i = i + 1
                j = 0

              back = False
            else:
              if j > 0:
                j = j - 1
              else:
                i = i - 1
                j = num - 1

              back = True
              b_o = 1

          else:
            if back == False:
              if k >= num and t >= num:
                b_o = 2
              else:
                if k >= num:
                  b_o = 1
                else:
                  b_o = 0
            else:
              b_o = 0

            if j > 0:
              j = j - 1
            else:
              i = i - 1
              j = num - 1

            back = True

          k = 0
          used = [0,0,0,0,0,0,0,0,0]
          use = [0,0,0,0,0,0,0,0,0]

      else:
        break

```

```
    m = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0]
    i = 0
    j = 0

    for fin in range(49):
      if i < num:
        s = i * num + j
        t = a[s]
        k = b[s]
        p = t * num + k + 1
        m[s] = p

        if j < num - 1:
          j = j + 1
        else:
          i = i + 1
          j = 0
      else:
        break

    self.mag = m
```

Listing A.46: MagicSquare in Vyper

Contactos:
Universidade de Évora
Escola de Ciências e Tecnologia
Colégio Luis António Verney, Rua Romão Ramalho, nº59
7000 - 671 Évora | Portugal
Tel: (+351) 266 745 371
email: geral@ect.uevora.pt