MONOPOLY WITH BITCOIN

MARTIN TIGGERDINE

This dissertation was submitted in part fulfilment of requirements for the degree of MSc Advanced Computer Science

DEPT. OF COMPUTER AND INFORMATION SCIENCES
UNIVERSITY OF STRATHCLYDE

AUGUST 2019

Declaration

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.
(please tick) Yes [ ] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 11133.

I confirm that I wish this to be assessed as a Type 1 2 3 4 5
Dissertation (please circle)

Signature:

Date:

Abstract

“Bitcoin truly is a new technology, and we can only get so far by explaining it through simple analogies to past technologies” (Narayanan et al., 2016). Cryptocurrency has risen meteorically since its invention 10 years ago and could be the future of currency but is only now being adopted into curriculums. This dissertation explores whether an educational game could be used to communicate how cryptocurrency works. A prototype was designed, implemented, tested and evaluated with the help of a small group of volunteers who played the game and took a survey about it. Their responses were supportive of the idea that playing “Monopoly with Bitcoin” could be more engaging and interactive than traditional methods like reading from a textbook. Their feedback was also informative in terms of how the game could be further improved.

Acknowledgements

For her guidance and counsel, I would like to thank my supervisor, Dr Rosanne English.

I would also like to acknowledge my colleagues from Computer & Electronic Systems, with whom I am lucky to have shared my university years.

I am thankful to my girlfriend, Zuzanna, and my parents, Martina and Stephen, for putting up with me. I do not know what I would do without them.

I would like to dedicate this dissertation to Geoff Robinson.

Contents

Figures

Tables

Code Excerpts

1. Introduction

Currency has existed since before the beginning of written history. The first coins were developed in Iron Age Anatolia and paper currency first developed in Tang Dynasty China. These mediums of exchange have existed for thousands of years, supplemented by inventions like the credit card.

American cryptographer David Chaum feared “automation of the way we pay for goods and services . . . may have a substantial impact on personal privacy as well as on the nature and extend of criminal use of payments” and proposed “a fundamentally new kind of cryptography” (1983). The use of credit cards was, he said, “an act of faith. . . . Each party is vulnerable to fraud by others, and the cardholder in particular has no protection against surveillance” (1988). Paper cash too is “traceable in principle”. He laid the foundations for a new kind of currency.

A cryptocurrency is a secure and decentralised digital asset. Cryptography is used “to secure the transactional flow, as well as to control the creation of additional units of the currency” (Chohan, 2018). Freedom from national monetary policies, insulation from bank failures and collapses, borderless payments and immunity to inflation or deflation (Reese, 2018) are some of the advantages of “a decentralised currency of the people” (Greenberg, 2011).

Cryptocurrency has grown astronomically over the past decade and optimists believe it “will fundamentally alter payments, economics and even politics around the world” (Narayanan et al., 2016). As of March 2019, there are 2100 cryptocurrencies and the market cap is $133 billion.

The aim of this project is to create a system which allows users to explore the concepts of blockchain and cryptocurrencies by participating in games of Monopoly where the Monopoly money is replaced by an artificial cryptocurrency. In addition to being able to participate in the game, users will be able to find out more about how blockchain works (e.g. through a visualisation) and view blocks. This dissertation will ask how such a game could be designed and how successful that approach might be.

For this dissertation, a prototype of the application will be developed using “the Waterfall process model (in which a software product is viewed as progressing linearly from conception, through requirements, design, code, and test)” (Laplante, P. A., & Neill, C. J, 2004) to divide the problem into easier subproblems. An evaluation of the prototype will be carried out in which qualitative and quantitative evidence will be gathered. I expect the evaluation to reflect the evidence that visualisation, engagement and gamification can aid understanding of abstract concepts like cryptocurrency.

2. Background

In this chapter, I will research Bitcoin and other cryptocurrencies, and their dependencies, from peer-to-peer networks to public and private keys. I will also acknowledge some educational theory and think about how I can use visualisation and engagement. Finally, I will review a couple of tools that are analogous to the one I want to develop, to emulate the best things about them.

2.1. Bitcoin

When we think of currency, we think of coins, banknotes and credit cards. Our idea of money is essentially the same as it was thousands of years ago when paper receipts representing grain were used as money in Sumer and Ancient Egypt, and Cowry shells were used by Arab traders. But for as long as money has existed, its value has only been as sound as the forces defending it. This caused the Near Eastern trading system collapse and, by extension, the Late Bronze Age collapse. One only has to remember the unsolved 2004 Northern Bank robbery, in which £26.5 million was stolen, or the 1969 Linwood bank robbery, in which two policemen lost their lives, to know this flaw is not a thing of the past. Money is vulnerable not only to theft, like in these examples, but also to fraud (deception for financial gain) and market manipulation (interference with the free and fair operation of the market). According to the Scottish government (2018) statistics on recorded crime in Scotland 2017-18, just 39.1% of fraud and 19.2% of theft was cleared up, meaning there was a sufficiency of evidence to consider criminal proceedings. These clear up rates compare poorly to the average of 49.5%, implying financial crime is something we cannot afford to underestimate as much as it is difficult to solve. Holistically, this is evidence that traditional money is rarely safe. For every improvement with respect to security, from alarm systems and time-locked heavy vault doors to GPS tracking devices and biometric technology, there seems to exist an equal and opposite countermeasure – kidnapping the bank manager, using electromagnetic pulses to disable alarms, cameras. What is needed, therefore, is not to continue to try in vain to protect traditional currency, but to replace it with something inherently safer.

Nakamoto[1] proposes an alternative:

[1] Satoshi Nakamoto is not a person, but a pseudonym used by the person or people who invented Bitcoin.

> A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone (Nakamoto, 2008).

This abstract preceded the invention of cryptocurrency – digital currency. Not only can it not be stolen because it is not a physical entity, there is no need for a central bank like the Bank of England or the European Central Bank to oversee its use. This has lots of financial implications. Central banks are inevitably tied to states in a way that cryptocurrencies are not. Furthermore, the trade-off between privacy and security that people have little choice but to accept when they deal with banks does not apply to cryptocurrencies because identifying information like email or name does not have to be provided. This gives users a degree of freedom they might not have experienced when sending and receiving money by other means. For example, they are free from some of the red tape inflicted by banks, financial institutions and governments. Another side effect is that cryptocurrencies are finite, the same property that makes precious metals like gold good investments.

Considering its potential benefits, it should not come as a surprise that digital currency is becoming more prevalent. According to EconoTimes, “over 24 countries are currently investing in distributed leger technologies with $1.4bn in investments over the past three years” (2016), making it important for people to learn about. If digital currency is to take over, everyone will need to learn about it sooner or later. There are several computer science and mathematical concepts that must be explored to have some knowledge of cryptocurrency and understand how it achieves these benefits.

In the context of an educational aid to teach people about cryptocurrency, it would be useful for the user to be able to learn about some of these concepts.

One of the ways in which Bitcoin and other cryptocurrencies achieve their independence is thanks to the network architecture they use. The most common models are client-server and peer-to-peer (P2P). Client-server networks (Figure 1) are arranged around a server that acts as an intermediary whenever one client wants to interact with any of the others. In this way, everything that happens on the network is managed by the server. Conversely, P2P networks (Figure 2) are an example of distributed computing. There are no clients or servers, only nodes that connect directly. In the context of cryptocurrency, the server in the client-server network is like a bank and the clients are like customers. The removal of the server from the model is the analogous to the decentralisation cryptocurrency achieves.

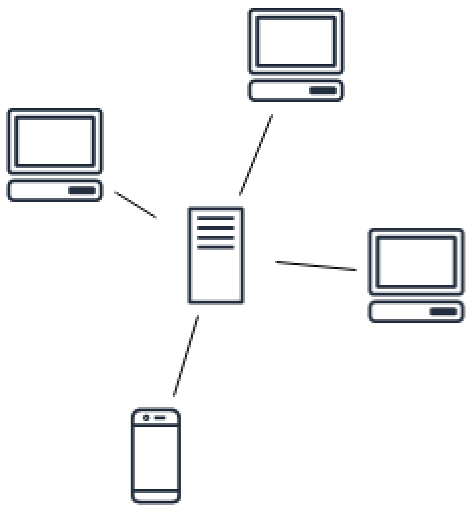


Figure 1: Client-server

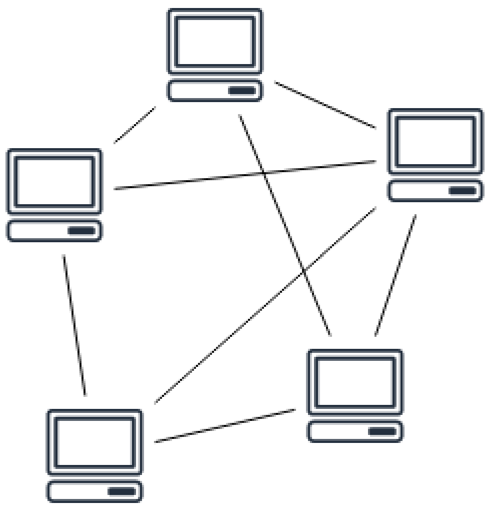


Figure 2: Peer-to-peer

If resources are shared and each node has access to everything, how can something like a digital currency be safe? What is there to stop one node from, say, fraudulently spending more money than it ought to be able to and lying to the others about it? The answer is the blockchain, a public ledger that records bitcoin transactions. The blockchain is, unsurprisingly, a chain of blocks, each of which is like a box that holds a set of transactions. Each block has an index, a timestamp, a list of transactions, a proof and a hash of the previous block. Each transaction has a sender, a recipient, an amount and the index of the block it is recorded in (see below for some examples of real Bitcoin blocks and transactions).

The chain can be followed all the way back to the genesis block (a special block that has no transactions) and is growing all the time. When a block is created – in the case of Bitcoin, this happens about once every ten minutes – it is broadcast to the network to be verified. If one node tries to manipulate the blockchain in some way, its version will not match the one agreed by the majority to be the real one. It is this strength in numbers that makes the blockchain immutable because if an attacker corrupts a block somewhere in the chain, that block's hash will be incorrect. Since each block's hash is linked to the hash of the previous block, the evidence the chain has been tampered with will propagate to the newest block and be obvious.

On the other hand, if someone controls the majority of the computational power on the network, they can use what is known as a 51% attack or majority hash rate attack to make changes to transaction history and stop new transactions from happening. The only inherent protection against attacks of this nature is that the higher the market cap of a cryptocurrency, the higher the cost of trying to attack it. The cost of a one hour 51% attack on Bitcoin is estimated at over $700,000, but on other cryptocurrencies at as low as $100. Cryptocurrencies that are just starting up are therefore more vulnerable to this kind of attack than Bitcoin (57% of the cryptocurrency market) and other cryptocurrencies like Ethereum (11%) and XRP (7%). Mining pools, cryptocurrency's equivalent of lottery syndicates, can reach the 51% threshold easier by sharing their processing power. In 2018, just five 51% attacks cost cryptocurrencies $20 million (Godshall, 2018).

The amount of each cryptocurrency in existence is fixed and depends on something called its Proof of Work algorithm. The goal is to solve a problem in which the previous proof is a variable and the solution is the new proof. This is what links each block to the one before it and makes the blockchain a chain. The idea is that proofs are difficult to find but easy to verify, so when one node finds a proof, the others can verify it. If someone tries to modify a block, for example to increase the amount of money someone sent them, they will invalidate not only that block, but also all the subsequent blocks. Whenever a miner finds a proof, they are rewarded with an amount of the cryptocurrency they are mining. Without this, there would be no reason for anyone to mine and the cryptocurrency would be in jeopardy.

Bitcoin's blockchain is public and can be viewed by anyone at any time. For example, if we look at block #579496, we see it was forged at 10:11:58 on 2019-06-06 and contains 2449 transactions in which about 14,650 BTC or £90 m was exchanged. Its proof is 1788743707 and the hash of the previous block is 0000000000000000000af273011c7ebf416b58761aa060950a0bbb78da13b3a. We also see the sender, recipient(s) and amount(s) of each transaction, like this one, in which 12FqhoHvp58Yg3VY55mvy1FKBXYk8TKn3C sent 3JprjjmfaXVjUgnHC8ujRqBfxwLCVEyw7C about 0.042 BTC.

The "crypto" part of cryptocurrency is short for cryptography, the art of writing or solving codes. In cryptography, a hash function is one that maps arbitrary size data onto fixed size bit strings. Figure 3 is an example of a hash function. The hash of a random word, "Fox", is "DFCD3454". It is almost impossible to recreate the input data given the hash. The second and third inputs ("The red fox runs/walks across the ice") only differ by one word but their hashes ("52ED879E" and "46042841") are completely different. In other words, changing part of the input changes the hash entirely, underlining why it is so difficult to "break the code".

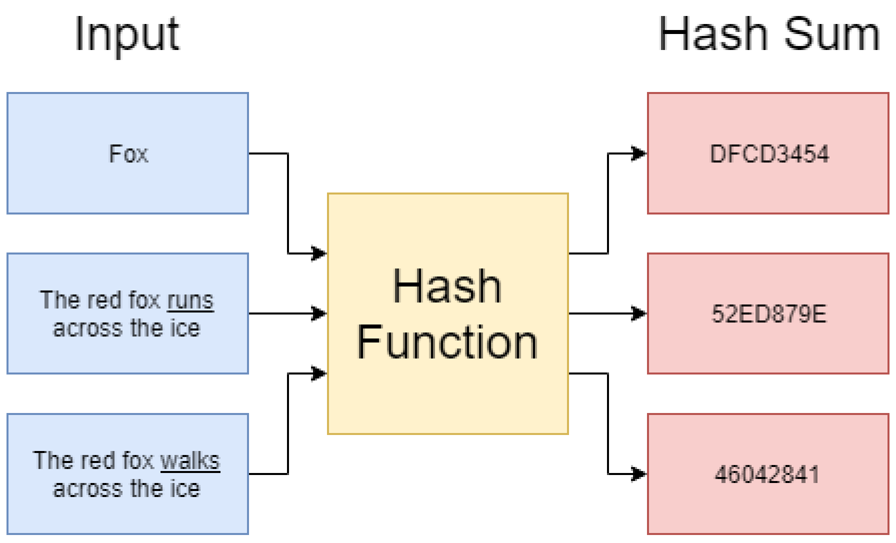

Input
Hash Sum
Fox
The red fox runs across the ice
The red fox walks across the ice
Hash Function
DFCD3454
52ED879E
46042841

Figure 3: Hash function

Cryptocurrencies uses cryptographic hash functions in a couple of different ways. The first is to achieve the difficulty of finding proofs. In Bitcoin, multiplying a proof by then one before it and hashing the answer must yield a hash that starts with “000000000000000000…” (18 leading zeros). Finding a proof that satisfies this condition by trial and error is extremely unlikely, which is why it

The second is as a substitute for usernames and passwords. This is called public-key cryptography. Figure 4 is a demonstration of this Each user has a pair of keys, one public and one private. Bob wants to secretly say “Hello Alice!” to Alice. He encrypts the string “Hello Alice!” using her public key, yielding “6EB…”, and sends it. If a third party tries to spy on them, all they will see are some random letters and numbers. The only person who can decrypt it is Alice because only she possesses her private key. In the context of cryptocurrency, public keys are used to identify who money is being sent to/received from, and private keys are used to authenticate transactions. In this way, public keys function like usernames, and private keys like passwords.

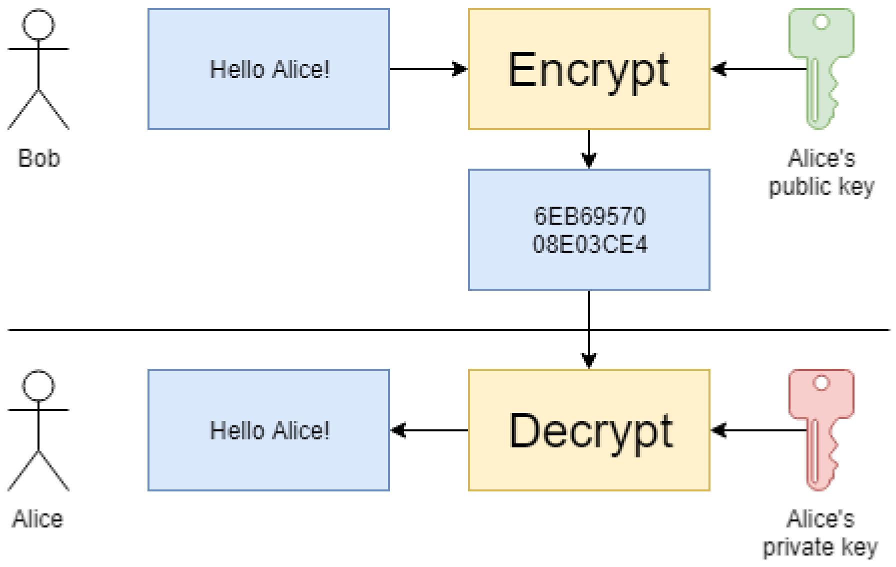


Figure 4: Public-key cryptography

The cryptographic hash function used by Bitcoin is AES-256. Its name refers to its key length of 256 bits, making the chance of guessing a key at random worse than 1/1e+77, comparable to the chance of winning the National Lottery jackpot 10 times in a row (1/4.69e+78) and the estimated number of atoms in the universe (1e+80).

2.2. Education

To imagine a tool that allows users to explore the concepts of blockchain and cryptocurrencies, some research should be done with respect to educational theory. Two concepts that could be relevant to the research question are visualisation and engagement.

2.2.1. Visualisation

Visualisation can be defined as "the ability to represent, transform, generate, communicate, document, and reflect on visual information" (Hershkowitz, 1989). It can be used in science, to explore content, find structure, check assumptions and communicate results (Jacoby, 1997), in education, to teach about something abstract, and to create "infographics" like we see on the internet and in the news. To expand on the educational context, Fouh, Akbar and Shaffer comment "one cannot see or touch an algorithm or data structure" and "much of the content involves the detailed workings of dynamic processes" (2012), underlining why visualisation and other techniques are helpful. For example, atomic structure is something that is easy to understand thanks to visualisation. Being able to see the protons and neurons grouped in the nucleus, and the electrons orbiting them, is worth a thousand words. Likewise, visualisations of sorting algorithms like bubble, selection, insertion, merge and quick sort can be more intuitive than pseudocode.

Hundhausen, Douglas, and Stasko collated 24 experiments related to algorithm visualisation and meta-analysed them (2002). The format of each experiments was for two sets of participants to learn how an algorithm works, one using algorithm visualisation (AV) technology and the other using none. 11 of the 24 experiments yielded a significant result in support of visualisation, like the participants who used visualisation scoring significantly higher than the others. 10 were neutral, implying visualisation made little difference one way or the other. Just one yielded a negative result, "in which students who used AV technology actually performed significantly worse than students who used text-based tools". (Two experiments were discarded because their results could not be attributed exclusively to visualisation.) With 11 positive results to just one negative one, these results support the hypothesis that visualisation is valuable in education and could help someone develop their knowledge and understanding of cryptocurrency. Fouh et al. believed "the most significant findings of the . . . meta-study were that (a) how students use AV, rather than what they see, has the greatest impact on educational effectiveness; and (b) AV technology is effective when the technology is used to actively engage students in the process".

2.2.2. Engagement

Work has been done to break engagement down into different forms. Naps et al. argue “that such technology, no matter how well it is designed, is of little educational value unless it engages learners in an active learning activity” (2002). They define six different forms of learner engagement with visualisation:

1. No viewing (no visualisation technology is used at all)
2. Viewing (the core form of engagement)
3. Responding (answering questions concerning the visualisation presented by the system)
4. Changing (allowing the learner to change the input of the algorithm under study in order to explore the algorithm’s behaviour in different cases)
5. Constructing (learners construct their own visualisations of the algorithm under study)
6. Presenting (presenting a visualisation to an audience for feedback and discussion)

Engagement tends to take the form of viewing (without which engagement is basically impossible) as well as at least one of the other forms. Each of responding, changing, constructing and presenting are valuable and no one of them is automatically better than any other. This view of engagement is instructive both in terms of developing a new tool and using an existing one in the context of education. This project will try to use viewing, responding and changing to engage the user. The manner of their viewing, responding and changing is described in the requirements and design sections.

2.2.3. Examples

Hundhausen et al. refer to Java and the “internet era” as factors in the growth of visualisation. One example is McNear and Pettey’s tool for teaching encryption algorithms, which “shows error free step by step descriptions in an interactive environment” and “allows the user to change the interface, add subsequent algorithms, and even incorporate the package into other software for actual encryption purposes” (2005). One of the advantages of the tool is that the user can interact with it in a number of different ways and has control over their experience. This means they can learn in their own way. For example, one user with no prior knowledge might want to read the descriptions, but another user who is comfortable with the basics might want to write a program that uses the algorithm and learn by doing. It can be concluded that being able to choose how, and how much, to interact with the visualisation aids the user experience. To visualise Bitcoin and other cryptocurrencies, this could mean giving the user the freedom to browse the blockchain however they want (in the same way as the real Bitcoin blockchain) or watch a step by step animation of a transaction.

Another example is SHAvisual, a visualisation tool for the SHA-512 secure hash algorithm (Ma et al., 2014). It has two modes, the demo mode, which demonstrates how the algorithm works, and the practice mode, which provides teachers a way to test students. It was designed to match the layout found in textbooks, so its main advantage is that the user can learn and be tested in the same environment. This is another idea that could be integrated into other visualisation tools. For example, a testing mechanism could be built into a cryptocurrency visualisation tool to find out how much the user knows and use this to tailor their experience.

Writing this chapter has given me a fresh understanding of cryptocurrency, its advantages and disadvantages, and what makes it possible. This, in conjunction with what I know now about visualisation and engagement, will carry over to the requirements phase and give me ideas about what I could teach the player about cryptocurrency, and how I could do it.

3. Requirements

It is common practice for developers and their clients to work together to figure out what a piece of software is going to do before it is designed. In this chapter, I will break the problem of making a Monopoly clone down into a number of requirements. On top of these, I will think about how the player could interact with the cryptocurrency part of the game.

I formalised the requirements by dividing the challenges of implementing a Monopoly clone and adding an artificial cryptocurrency to it into use cases. A use case encapsulates the interactions between an actor[2] and a system to achieve a goal. The advantage of use cases is that they enable the developer and the client to share an understanding of what the system is meant to do. I followed the use case format shown in “UML distilled” (Fowler, 2004). Each use case has a main success scenario and zero or more extensions. The main success scenario is the scenario in which everything works as expected and the goal is achieved. An extension is a variation where a difference compared to the main success scenario results in different interactions and, in some case, the goal not being achieved. Some use cases have preconditions and postconditions. These are conditions with respect to the state of the system that are be true before and after the use case. To make the use cases easier to read, I wrote a one or two sentence preview of each.

I drafted use cases 1-14 from Hasbro’s classic Monopoly rules. These are the rules that have come with Monopoly for almost 30 years. Something I found difficult was drawing a line between use cases and extensions. It would be fair for the reader to question, for example, why collecting $200 when you pass go would not be as valid an extension of moving your token as it is a use case. The answer is that the rules of the game are full of exceptions, like chance and community chest cards. Converting all 36 cards to use cases and extensions would be confusing so there is one umbrella use case, “5: Use Chance, Community Chest Cars”, for all of them. Returning to the example of collecting $200 when you pass go, there are a number of cards that advance the player to a space and allow them to collect $200 on the way, ergo throwing the dice and moving your token is not the only way to achieve this goal. To be as faithful as possible to the rules, I was careful to have at least one use case for each for each heading in the rules.

[2] user

Use cases 15-18 refer to the educational part of the system. They are less explicit than the others because all they do is indicate the user must be able to view the blockchain, blocks and transactions somehow, and watch an animation. The manner of visualisation and engagement used to achieve these goals is something for the next section, design. Finally, use cases 19-21 deal with goals that are not part of the game of Monopoly but are auxiliary goals like saving and loading the game.

These are the first Monopoly use case, “Move Token”, and the first cryptocurrency use case, “View Blockchain”. The rest be found in “Appendix 1: Use Cases”. “Move Token” is a good example of how extensions work. If the player throws doubles, they get an extra turn, but if they throw doubles three times in a row, they go to jail.

1: Move Token

A player starts their turn by throwing their dice and moving their token that number of spaces.

Main Success Scenario:

1. System shows “throw the dice” option
2. Player chooses “throw the dice” option
3. System randomly generates number between 2 and 12
4. System shows “move your token” option
5. Player chooses “move your token” option
6. System moves player’s token that number of spaces in the direction of the arrow

Extensions

1a: Player throws doubles

If a player throws doubles, they get to go again.

.1: System shows “you threw doubles” prompt

.2: Player throws again

.3: Player moves their token as before

If a player throws doubles three times in a row, they go to jail.

1b: Player throws doubles three times in succession

.1: System moves player’s token to space marked “In Jail”

15: View the Blockchain

Main Success Scenario:

1. Player asks to view blockchain
2. System shows blockchain (blocks)
3. Player can view any of the blocks in the blockchain

Acknowledging some requirements are more important than others and prioritising them is a valuable part of the requirements phase. One way to do this is using the MoSCoW method. Requirements are divided into four categories in decreasing order of importance: must have, should have, could have and won’t have. I applied the MoSCoW method to my requirements (Table 1). Basic Monopoly ideas like buying property and paying rent could never be anything other than must haves. On the other hand, selling and mortgaging property are should haves because although they represent more ways for transactions to happen, the game is playable without them. The only won’t have requirement was playing against the computer. Implementing an artificial intelligence agent is an example of something that, whilst fun, would be beyond the scope of this dissertation.

Having requirements makes things easier because I can solve lots of small problems instead of one big one. They will also be invaluable in the next phase, design. The MoSCoW method will help me to prioritise which requirements to implement first in case I run out of time and cannot do everything.

Table 1: Use cases

	Title	**Description**	**Priority**
1	Move Token	*A player starts their turn by throwing their dice and moving their token that number of spaces.*	M
2	Pass Go, Collect $200	*If a player lands on or passes over Go, the first space on the board, they get $200.*	M
3	Buy Property	*If a player lands on an unowned property, they can buy it.*	M
4	Pay Rent	*If a player lands on someone else's property, they have to pay rent.*	M
5	Use Chance, Community Chest Cards	*If a player lands on a chance or community chest space, they draw a card and follow the instructions.*	M
6	Pay Income Tax	*If a player lands on the income tax space, they have to pay $200 or 10% of their total worth.*	M
7	Get Out of Jail	*There are a few ways for a player to get out of jail. They can try to throw doubles (they get out if they fail three times) or just pay $50.*	M
8	Use Free Parking	*The free parking space is just a blank space.*	M
9	Buy Houses	*Houses increase rent. You can only buy a house on, say, a yellow property if you own all three yellow properties.*	M
10	Buy Hotels	*Hotels increase rent even more. You can only buy a hotel if you own four houses.*	M
11	Sell Property	*Players can sell property to each other for whatever price both parties want.*	S
12	Mortgage Property	*Properties can be mortgaged, meaning rent cannot be collected on them.*	S
13	Lift Mortgage	*Likewise, properties can be unmortgaged.*	S
14	Go Bankrupt	*The win condition is to be the last player standing after everyone else goes bankrupt.*	M
15	View the Blockchain		M
16	View a Block		M
17	View a Transaction		M
18	Watch a Step by Step Animation of a Transaction		C
19	Save the Game		C
20	Load the Game		C
21	Play Against the Computer		W

4. Design

This chapter deals with the design of the system in terms of classes, fields, methods and relationships, and in terms of what it is going to look like and how the user will interact with it.

4.1. UML

A UML (Unified Modeling Language) class diagram is a way of visualising a system's classes, their fields, their methods, and the relationships between them, and is the industry standard when it comes to object-oriented system design.

I could have used a UML tool like Visual Paradigm or StarUML but believed it would be better to use draw.io, a diagram editor I was already comfortable with. The only feature I would miss would be code generation and I did not think there would be enough classes to justify downloading, installing and learning a new tool for this single purpose. Creating the diagrams from scratch would also be an opportunity for me to refresh my knowledge of UML's syntax, in which there are lots of different relations (Figure 5), italics indicate abstract classes and methods, and underlines mean static.

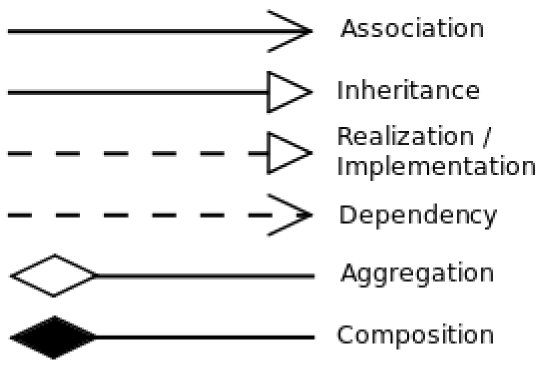


Figure 5: UML relations

4.1.1. Monopoly

The first thing I wanted to design was a collection of classes that would make up a game of Monopoly. I thought about what comes in the box (a board, etc.) and translated it to classes and fields (Figure 6).

The Board is comprised of Spaces. Each Space can have any number of Players (the players on that space) and each Player has a Space (the space that player is on). This allows us to answer the questions “which players are on this space?” and “which space is this player on?” without searching.

In Monopoly, some Spaces are Properties that can be bought or sold, and each Property is a Street, Railroad or Utility. The two biggest differences are that rent is calculated differently on each different type of Property, and Streets can have houses and hotels built on them.

A choice I had to make was whether Property should be an abstract class or an interface. I remembered the Building Software Systems lecture in which I learned “extends is evil”. In his article of the same title, Allen Holub qualifies: “maybe not at the Charles Manson level, but bad enough that it should be shunned whenever possible” (2003). He goes on to warn against inflexibility, coupling, and the fragile base-class problem, all of which are weaknesses of abstract classes.

In defence of abstract classes, Java’s documentation says you should consider using them if “you want to share code among several closely related classes” or “you expect that classes that extend your abstract class have many common methods or fields”, both of which applied. This justified making Property an abstract class. The only real consequence would be some casting from Space to Property, Street, Railroad and Utility.

I also defined two enum types, Building and Colour. These are special data types that represent a fixed set of constants. If building was just an int, it would be possible to set it to -1 or 6 and cause an exception to be thrown. Some other advantages of enum types are that you can use ordinal() to get a constant’s position in its declaration (for example, if building was set to TWO_HOUSES, building.ordinal() would return 2), and overwrite toString() to return a custom string for each constant (for example, to return “Light Blue” instead of “LIGHT_BLUE”).

I did not know whether Token should be a class or an enum type so I omitted it from the UML.

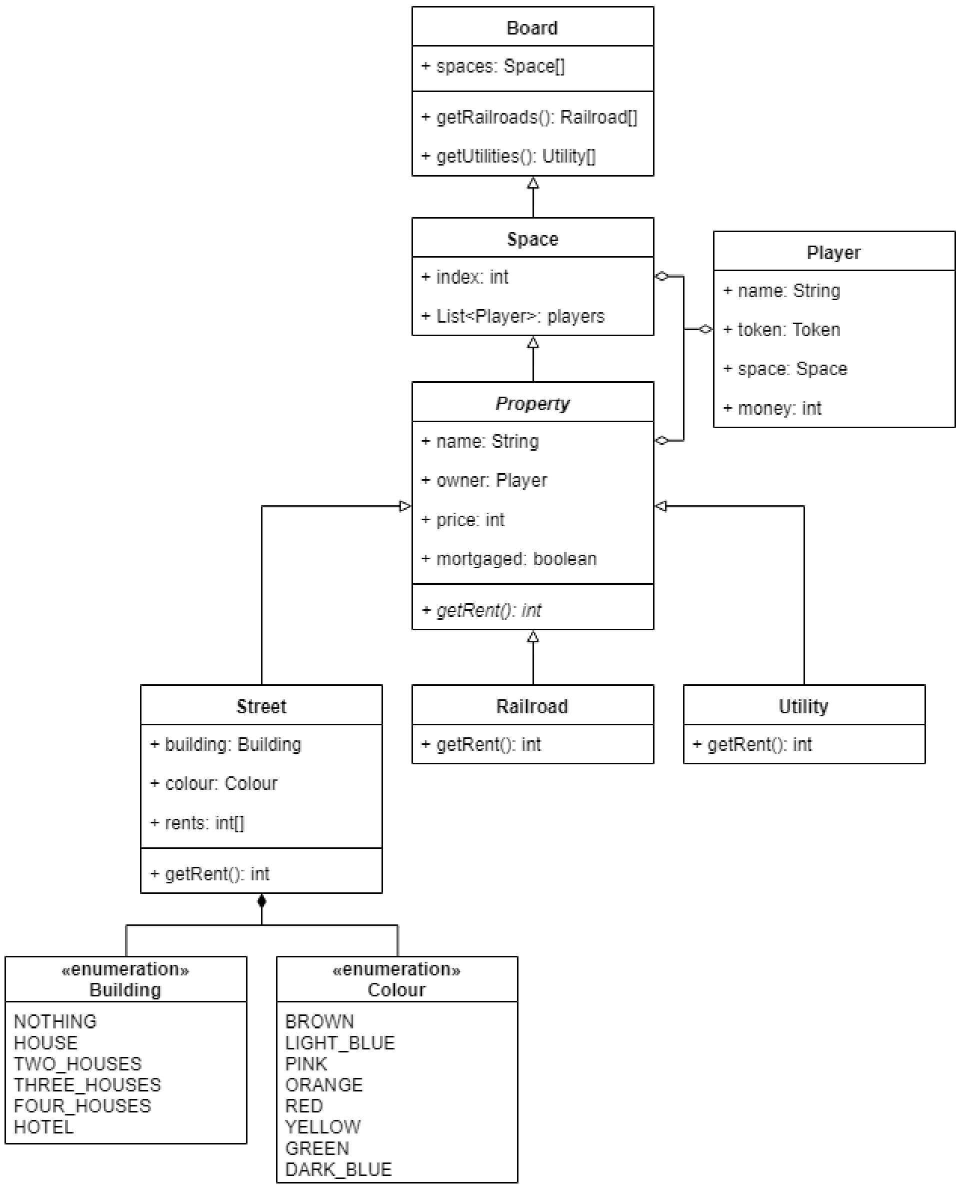


Figure 6: UML class diagram of Monopoly classes

4.1.2. MVC

MVC (Model-View-Controller) is an architectural pattern that can be used to develop applications with user interfaces. “This three-way division of an application entails separating (1) the parts that represent the model of the underlying application domain from (2) the way the model is presented to the user and from (3) the way the user interacts with it” (Krasner and Pope, 1988). The model and view can be worked on by different developers at the same time, modification is easier, and the same model can have multiple views (for example, a GUI (Graphical User Interface) view and a CLI (Command Line Interface) view).

In my design (Figure 7), Controller is a class and Model and View are interfaces. This obfuscates things like Boards and Players from the Controller. In theory, someone could design and implement a view for my game without knowing anything about Controller or Model. I cannot think of any disadvantages of MVC in this context.

MyModel is made up of the classes in Figure 6 (as it is a model of a game of Monopoly). For example, it has one field for the board and another for the players. These are the "things" the controller can manipulate.

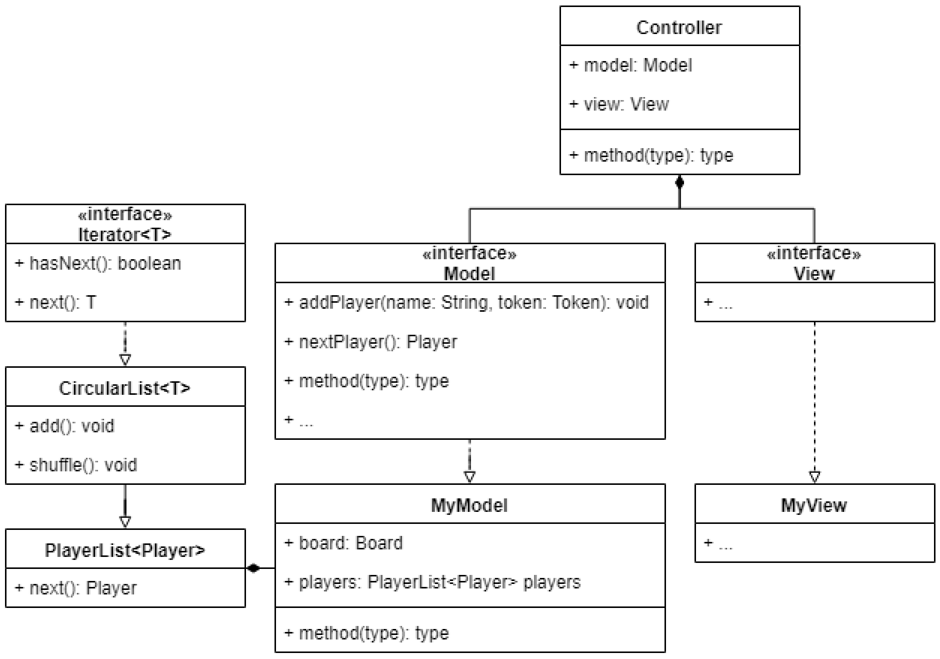


Figure 7: UML class diagram of MVC classes

4.1.3. Cryptography

To be able to replace the Monopoly money with an artificial cryptocurrency, I had to design some classes that would simulate a blockchain (as described in the "Background" chapter). These would be "plugged into" the game such that whenever a player's money changed, a new transaction was recorded.

I used the articles “How to Create Your Own Cryptocurrency” by Emily Long (2018) and “Learn Blockchains by Building One” by Daniel van Flymen (2017), and my prior knowledge, to explore how to achieve this. From my requirements, I knew the blockchain was paramount because viewing the block, a blockchain and a transaction were must have requirements. A Blockchain is a List of Blocks and a Block is a List of Transactions. Block and Transactions are just data classes. In other words, the only methods they have getters and setters (I include getIndex() as an example). hash() creates a hash of a Block, proofOfWork() produces, as its name implies, a proof of work, and validProof() validates the proof. See Figure 8.

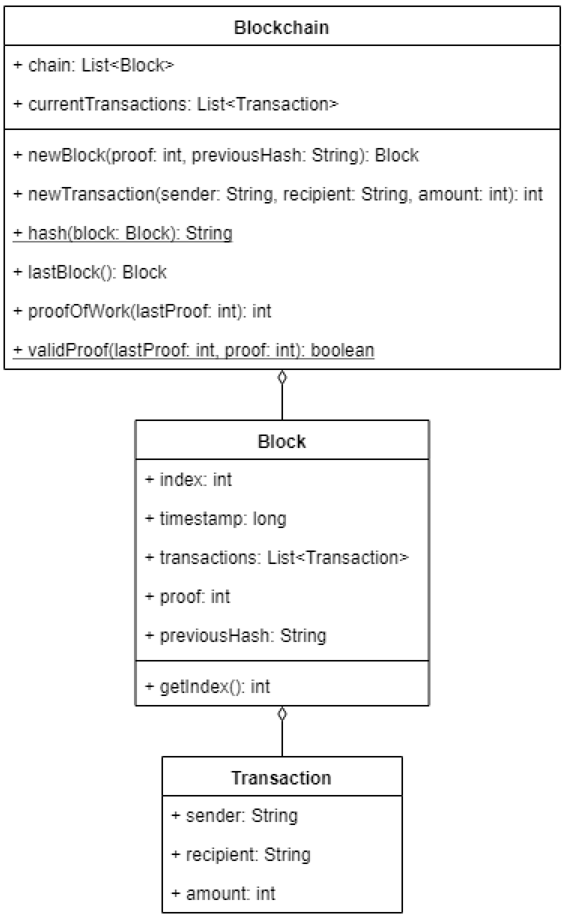

Blockchain
+ chain: List<Block>
+ currentTransactions: List<Transaction>
+ newBlock(proof: int, previousHash: String): Block
+ newTransaction(sender: String, recipient: String, amount: int): int
+ hash(block: Block): String
+ lastBlock(): Block
+ proofOfWork(lastProof: int): int
+ validProof(lastProof: int, proof: int): boolean
Block
+ index: int
+ timestamp: long
+ transactions: List<Transaction>
+ proof: int
+ previousHash: String
+ getIndex(): int
Transaction
+ sender: String
+ recipient: String
+ amount: int

Figure 8: UML class diagram of cryptography classes

4.2. UI

Another thing I had to design was the UI (User Interface). I used the requirements to make a list of things the user would have to be able to do and the best way they could do it. For example, the third use case is “Buy Property”. In Monopoly, the only time a player can buy a property is when they land on it and it is unowned. I knew a button would be enough to satisfy this requirement. On the other hand, a player can mortgage any unimproved property they own. A button would not be enough – there would also have to be a way for the user to indicate which property they wanted to mortgage.

I spent comparatively less time designing the UI for three reasons. First, I had almost no experience of creating projects with UIs (graphical or otherwise) other than Gizmoball, a pinball simulator. I was part of a group of five and did almost no work on the view. This meant I did not know how advanced I would be able to make the view. Second, I was working alone and was not delegating the view to someone else. Third, as this was to be a prototype, not a product, function was more important than form. I did make a couple of sketches to communicate what was in my head to my supervisor. This also allowed me to understand which components (like buttons and labels) to learn about.

Figure 9 is a sketch of the Monopoly view. From my requirements, I knew it had to support showing the board, etc., and capturing user input. What was also obvious was that the board was going to need lots of space, and even then, it would be impossible to show everything about every square at once. I imagined each space as a button that, when clicked, would show that space’s name, type, etc. in the middle of the board. The rest of the window would be a log of what was happening (“Bob rolls a 9”, “Bob moves to Whitechapel Road”, etc.) that the player could scroll through, and buttons that would be lit up/greyed out when the player could/could not do things (“Buy Property”, “Buy House…”). Not included in the sketch are houses and hotels, which would be added and removed to spaces, and tokens, which would also move between spaces. Dialogs would be used whenever the player had to choose between two or more things (like which property to mortgage).

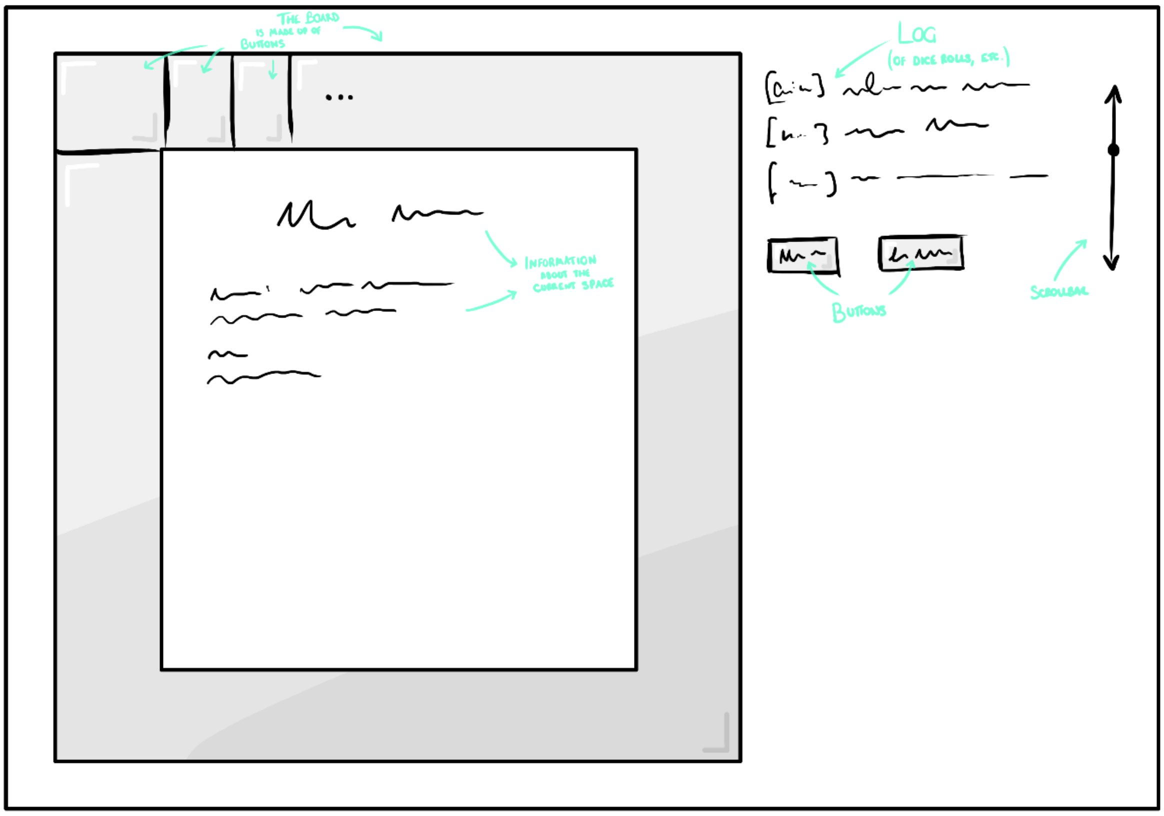

The board is made up of buttons
...
Log
(of dice rolls, etc.)
Information about the current space
Buttons
Scrollbar

Figure 9: Sketch of Monopoly view

I also designed a way for the user to view the blockchain, blocks and transactions (Figure 10). The blockchain would have a scroll pane of buttons, one for each block. Each block would show its timestamp, proof and previous hash, and have a scroll pane of buttons, one for each transaction. Each transaction would show its sender, recipient and amount. Blocks and transactions would also have an "Up" button that would take the user from a transaction to its block or from a block to the blockchain. In this way, the user would be able to navigate the blockchain and read the data that made up each block and transaction.

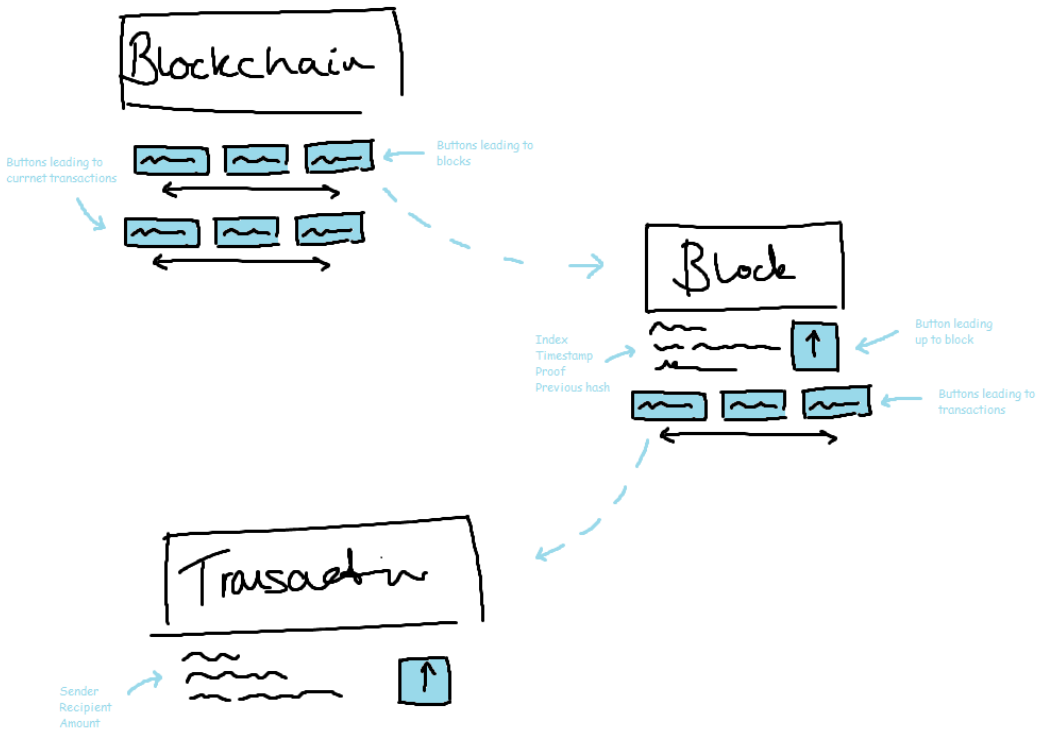

Blockchain
Buttons leading to currnet transactions
Buttons leading to blocks
Block
Index
Timestamp
Proof
Previous hash
Button leading up to block
Buttons leading to transactions
Trasaction
Sender
Recipient
Amount

Figure 10: Sketch of blockchain, blocks and transactions views

I planned to link this to the Monopoly part by having a "View Blockchain…" button that would open the blockchain view in a new window.

There were at least two advantages of the work I have done in this chapter. I know the UML I made will give me a head start in the next phase, which is implementation. I was also able to show my user interface sketches to my supervisor and some of my friends, who approved. This gives me the confidence I need to move on to implementing it.

5. Implementation

In this chapter, I will share my strategy, talk about some of the implementation choices I made, like which language and GUI library I used, and describe what I think are the most important parts of my model, view and controller. Much of this can be traced back to the design and requirements.

5.1. Strategy

I identified two different strategies I could use to break down the problem.

The first was to take the MVC paradigm to its extreme and write the controller from start to finish before so much as creating the model and view classes. The model and view interfaces would be written incidentally. In other words, if I wrote model.getPlayerName() in the controller, I would at the same time be writing a line of the model interface, String getPlayerName;. It would be up to the model class to implement this method and return the name of the player.

The second was to implement the model (the easiest of the three) first and worry about the controller and view later. I would break down the rules of the game and implement them one at a time. For example, I would start with an empty board, then add a token to it, then make the token move. On reflection, a third strategy would have been to start with the view, but I did not think of this at the time. I documented my thoughts about which strategy I should use in Table 2.

Table 2: Advantages and disadvantages of implementing controller or model first

	Advantages	**Disadvantages**
Controller -> model and view	+ Model, view interfaces written incidentally + Decoupling	- Impossible to test controller alone
Model -> controller and view	+ Easier + Can test model before using it	- Model interface shaped by class, not the other way around

On balance, I chose to implement the model first, the controller and view second, and replace the Monopoly money by an artificial cryptocurrency third and last.

5.2. Language

The two languages I am skilled enough in to use for this dissertation are Java and Python. I chose Java for no reason other than personal preference. It is the one I like the most and the one I have written the most lines of code in (by a wide margin).

5.3. GUI Library

I also had to choose a GUI library. Of the three I knew about – AWT, Swing and JavaFX – I discarded AWT because Swing more or less encapsulates it. Both Swing and JavaFX have all the features I was looking for. Again, my mind was made up by the fact I had some experience with Swing and none with JavaFX.

5.4. Model

5.4.1. Interface

The methods in MyModel fall into one of five categories: space methods, player methods, transactions, houses/hotels, and mortgages. About half are getters and setters. There are also methods that iterate over the properties on the board and return (the names of)

- properties the player can buy a house on
- properties the player has one or more houses on
- properties the player can buy a hotel on
- properties the player has a hotel on
- properties the player can mortgage
- properties the player can lift the mortgage on

as well as methods that count the number of houses and hotels the player owns and a method that moves the player, updating the Player object and the old and new Space objects.

5.4.2. Class

MyModel has two fields, CircularList<Player> players and Player player. CircularList is a generic class that implements the Iterator interface. (For CircularList, see “5.9. Utilities”.) When one player’s turn is over, player = players.next(); is called and the next player’s turn begins. MyModel also uses Board, as do Player, Street, Railroad and Utility.

A difference from the design is that the board is a Singleton, ensuring it has only one instance that can be accessed easily. The biggest advantage of this is that any other model class can access the board easily. This also protects against more than one board existing.

5.4.3. model.Board

The only field in Board is List<Space> spaces. There are methods that

- add a space to the board
- add a street to the board
- add a railroad to the board
- add a utility to the board

and prevent there from being two squares with the same name, more than one square with type GO, more than one square with type JAIL, a property with a non-positive price or building price… There are getters that return lists of properties, streets, railroads and utilities by iterating over the list of spaces and using space.isStreet(), etc. to know which spaces to cast and add to the list to be returned. Searching for

- space with index i
- space with name n
- space with type t
- property with name n
- street with name n
- streets with colour c

is also supported.

The way spaces, streets, railroads and utilities are added to the board is not dissimilar to the Factory pattern (with the board being comparable to a space factory that can create spaces of different types).

5.4.4. Space, etc.

Like in my design, Space is a class, Property is an abstract class that extends Space, and Street, Railroad and Utility are classes that extend Property. Every space has an index, a name, a type and a list of players that are on it. Properties add an owner and a price, and can be mortgaged. Streets add a colour, an array of rents and a building price, and can have buildings built on them.

isProperty(), isStreet(), isRailroad() and isUtility() all return false by default. Property overrides isProperty() to return true, Space overrides isStreet(), and so on. The abstract method getRent() in Property is implemented differently in Street, Railroad and Utility. A street's rent depends on how many buildings it has and whether its owner owns every street of its colour.

Code Excerpt 1: Street.getRent()

```java
public int getRent() {
    /*
     * "It is an advantage to hold all the Title Deed cards in a colorgroup
     * because the owner may then charge double rent for unimproved properties in
     * that color-group."
     */
    if (isUnimproved()) {
        List<Street> sameColour = Board.getStreetsWithColour(colour);
        sameColour.remove(this);
        boolean allOwned = true;
        for (Street street : sameColour) {
            if (!street.isOwnedBy(getOwner())) {
                allOwned = false;
                break;
            }
        }
        if (allOwned) {
            return 2 * rents[0];
        }
    }

    return rents[building.ordinal()];
}
```

A railroad's rent is £25/£50/£100/£200 if its owner owns one/two/three/four railroads. These values satisfy the equation $rent(railroad) = 25(2^{r-1})$ where r is the number of railroads the player owns.

Code Excerpt 2: Railroad.getRent()

```java
public int getRent() {
    List<Railroad> railroads = Board.getRailroads();
    int railroadsOwned = 0;
    for (Railroad railroad : railroads) {
        if (railroad.isOwnedBy(getOwner())) {
            railroadsOwned++;
        }
    }

    return (int) (25 * Math.pow(2, railroadsOwned - 1));
}
```

A utility's rent is four/10 times the number on the dice, or ten times if its owner owns one/two utilities. These numbers satisfy the equations $rent(utility) = d(6u - 2)$ and $rent(utility) = du(u + 3)$ where d is the number on the dice and u is the number of utilities the player owns. I chose the latter (because if a railroad's rent increases exponentially with the number of railroads its owner owns, it follows that a utility's rent should increase exponentially too).

Code Excerpt 3: Utility.getRent()

```java
public int getRent() {
    List<Utility> utilities = Board.getUtilities();
    int utilitiesOwned = 0;
    for (Utility utility : utilities) {
        if (utility.isOwnedBy(getOwner())) {
            utilitiesOwned++;
        }
    }

    return utilitiesOwned * (utilitiesOwned + 3) * Dice.add();
}
```

5.5. View

5.5.1. Interface

There are four types of methods in the view interface. There are some “text” methods that print what is going on, and others that print information about players and spaces. The “board” methods move tokens and add and remove houses. Most of the “buttons” methods enable and disable buttons like the buy property button and the end turn button. There is also one that changes the text on the end turn button from “End Turn” to “Extra Turn”. Finally, the “dialogs” methods are used to ask the player

- how they want to try to get out of jail
- which street they want to buy or sell a house or hotel on
- which property they want to mortgage

and return their choice to the controller.

5.5.2. Class

MyView has a frame, a content pane and four components. They are

- view.Board (not to be confused with model.Board)
- a JTextArea (in the middle of the board)
- Buttons
- another JTextArea (in the top right corner)

5.5.3. view.Board

Board is the biggest class in the view package excluding MyView. The board is a JLabel that displays an image I made in Paint. On top of each space is an invisible JButton. Houses and Hotels are JLabels that are added to the board. Tokens are JButtons that are added to the space they are on. (Adding them to the board made them impossible to click. I tried making Board a JLayeredPane and adding tokens on a higher layer than spaces, but clicks on tokens were still being received as clicks on spaces. I found the solution of adding tokens to spaces by trial and error.)

Board, Hotel, House, MyView, Space and Token all have WIDTH and HEIGHT constants that are used to lay them out. MyView also has SMALL_MARGIN and BIG_MARGIN constants. In the beginning, there were two arrays of Points, one in House and one in Token, that were used to set the location of tokens, houses and hotels. I replaced these by methods that take advantage of the WIDTH and HEIGHT constants. House.getLocation(…) takes the index of the space and the number of houses on the space, and calculates the house's position relative to the top left corner of the board (Code Excerpt 4). Token.getLocation(…) takes the index of the space and the number of players on the space, and calculates the token's position relative to the top left corner of the space (Code Excerpt 5). (Hotels use House.getLocation() with house = 1 if the hotel is horizontal and house = 2 if the hotel is vertical.) Thanks to this refactoring, the locations of Houses, Hotels and Tokens are always correct, no matter the dimensions of the components.

Code Excerpt 4: House.getLocation(…)

```
static Point getLocation(int space, int house) {
    if (house < 0 || house > 3) {
        return null;
    }
    int x;
    int y;
    if (space >= 1 && space <= 9) {
        x = Board.WIDTH - Space.HEIGHT - space * Space.WIDTH + house *
House.WIDTH;
        y = Board.HEIGHT - Space.HEIGHT;
    } else if (space >= 11 && space <=19) {
        x = Space.HEIGHT - House.WIDTH;
        y = Board.HEIGHT - Space.HEIGHT - (space - 10) * Space.WIDTH + house *
House.HEIGHT;
    } else if (space >= 21 && space <=29) {
        x = Space.HEIGHT + (space - 21) * Space.WIDTH + (3 - house) *
House.WIDTH;
        y = Space.HEIGHT - House.HEIGHT;
    } else if (space >= 31 && space <=39) {
        x = Board.WIDTH - Space.HEIGHT;
        y = Space.HEIGHT + (space - 31) * Space.WIDTH + (3 - house) *
House.WIDTH;
    } else {
        return null;
    }
    return new Point(x, y);
}
```

Code Excerpt 5: Token.getLocation(…)

```java
private Point getLocation(int space, int player) {
    int x;
    int y;
    int offset = isStreet(space) ? House.WIDTH : 0;
    if (space >= 0 && space <= 9) {
        x = player * Token.WIDTH;
        y = offset;
    } else if (space == 10) {
        x = 0;
        y = player * Token.HEIGHT;
    } else if (space >= 11 && space <= 19) {
        x = Space.HEIGHT - Token.WIDTH - offset;
        y = player * Token.HEIGHT;
    } else if (space == 20) {
        x = Space.HEIGHT - (player + 1) * Token.WIDTH;
        y = Space.HEIGHT - Token.HEIGHT;
    } else if (space >= 21 && space <= 29) {
        x = (2 - player) * Token.WIDTH;
        y = Space.HEIGHT - Token.HEIGHT - offset;
    } else if (space == 30) {
        x = 0;
        y = Space.HEIGHT - (player + 1) * Token.HEIGHT;
    } else if (space >= 31 && space <= 39) {
        x = offset;
        y = (2 - player) * Token.HEIGHT;
    } else {
        return null;
    }
    return new Point(x, y);
}
```

5.5.4. Dialogs

The “Get Out of Jail” dialog is shown using JOptionPane.showOptionDialog(…), which shows a button for each option. There are three ways for a player to get out of jail. The first is to try to throw doubles. The second is to pay a fine of £50 (if they can afford it). The third is to use the “Get Out of Jail Free” card (if they have it). showJailDialog() asks the controller which of these options to give the player.

The “Buy a House/Hotel”, “Sell a House/Hotel” and “Mortgage a Property” dialogs are shown using JOptionPane.showInputDialog(…), which shows a drop-down list of options. They share the same private method that takes a list of options, shows an input dialog, and returns the option the player selects. The title and message of the dialog can also be customised.

5.6. Controller

Controller is, unsurprisingly, the biggest class in the project. Table 3 breaks down its constants and fields.

Table 3: Controller constants and fields

Modifiers	Type	Name	Value	Comment
public static final	boolean	CRYPTO	true/false	Enables/disables crypto mode.
public static final	char	$[3]	'£'/' $\mathcal{M}$'/etc.	Currency symbol.
public static final	int	PLAYERS	2	Number of players.
private	Model	m		
private	View	v		
private	Random	r		Random number generator.
private	Crypto	c		
private	Questions	q		
private	int	moveDoubles	0	"If you throw doubles three times in succession, move your token immediately to the space marked "In Jail"."
private	int[]	jailDoubles	{0, 0}	"If you do not throw doubles by your third turn, you must pay the $50 fine."

When a player lands on a space, a switch statement is used. If the space's type is STREET, RAILROAD or UTILITY, property() is called. If it is GO, JAIL or FREE_PARKING, nothing happens. If it is CHANCE or COMMUNITY_CHEST, chance() or communityChest() is called. If it is INCOME_TAX or SUPER_TAX, the player pays £200 or £100 (ironically, income tax is worse than super tax). If it is GO_TO_JAIL, goToJail() is called.

[3] In Java, "$" is a legal identifier.

When a player lands on a property, one of three things happens. If the property is unowned and the player can afford it, the buy property button is enabled. If someone else owns the property and it is not mortgaged, the player pays rent. If the player owns the property, nothing happens.

Chance and community chest cards are switch statements from which one switch block is randomly executed. For example, the player might advance to “Go” or go to jail. Adding a card to either deck is as easy as adding a switch block to the switch statement and changing the upper bound of the randomly generated number that controls which switch block is executed.

When a player lands on “Go to Jail”, they go to jail and their turn ends.

5.7. Cryptography

I started by implementing Blockchain, Block and Transaction. I used the MessageDigest class in the java.security package, and a helper method that converts bytes to hex (see “Utilities”), to create SHA-256 hashes of blocks. Proofs are validated by concatenating the previous proof, the current proof and the hash of the last block in the chain, hashing, and comparing the first four characters to “0000”. (If the comparison returns true, the proof is valid.) In the “Learn Blockchains by Building One” article I used as a reference, the author multiplies the previous and current proofs but does nothing with the previous hash. This is a bug because the proofs alternate between the same two proofs (one of which is the proof of the genesis block).

As an aside, I wrote a class called Keys that generates, stores and validates public and private key pairs. I used KeyPairGenerator to do this. The algorithm used to generate the keys is DSA.

With these building blocks implemented and tested (see “Testing”), all I had to do was write a class the controller could use to add transactions from the game to the blockchain, get the blocks and transactions and enable the user to browse them somehow. Crypto is that class. It is also responsible for mining. When the number of current transactions reaches TRANSACTIONS_PER_BLOCK, the proofOfWork(…) and newBlock(…) methods are called and a new block is added to the chain.

5.8. Questions

I came up with the idea of replacing the chance and community chest cards by question cards. The Questions class has two accessible methods, add(…) and ask(…). add(…) takes a question string and an array of answer strings (the first of which must be the correct answer). ask(…) asks the user a random question and returns a boolean (true if the answer is correct, else false). If crypto mode is enabled, the controller calls ask() instead of chance() or communityChest(), followed by correct() if the answer is correct, else incorrect().

The questions (and answers) I added were

- “The blockchain is a _ ledger.” (“shared public”, “private”)
- “All _ transactions are included in the block chain” (“confirmed”, “unconfirmed”)
- “A _ key is used to sign transactions.” (“private”, “public”)
- “Transactions are confirmed through a process called _.” (“mining”, “drilling”, “prospecting”)
- “Bitcoin is vulnerable to _ attacks.” (“51%”, “SWK”, “99%”, “JNO”)
- “The block chain uses the _ model.” (“peer-to-peer”, “client server”)
- “Users have _ keys.” (“public and private”, “only public”, “only private”)
- “_ is the dominant cryptocurrency.” (“Bitcoin”, “Etherum”, “XRP”)
- “Bitcoin was invented by _.” (“Satoshi Nakamoto”, “Akazawa Yoshitaka”, “Seta Sotaro”)

If I had more time, I would have loaded the questions from a file (maybe a .txt or .csv file) to make them easy to change. I could have used a library like Jackson (which can parse formats like JSON, CSV and XML) or written my own parser to achieve this.

5.9. Utilities

Something I like to do is extract any code I think I can reuse and put it in its own “util” package. In this project, there are three classes in that package.

CircularList is basically an iterator that always has a next element. This is achieved by overriding hasNext() to always return true, and overriding next() such that if the index would be equal to the size of the list, it is reset to 0. The controller uses this class for the list of players (see “Class” under “Model”).

The Convert class has two methods, one for converting an array of bytes to a hex string, and the other for converging an int to an array of bytes. bytesToHex(…) ANDs each byte with 0xff (and pads it with a leading 0). It is used to return a string representation of the SHA-256 hash generated by Blockchain.hash(…). intToBytes(…) takes advantage of the BigInteger class’ toByteArray() method, which makes this conversion an easy one-liner. It was once used to convert the product of the previous and current proofs to bytes so it could be hashed.

5.10. Testing

First, I used main methods as sandboxes where I can try code as soon as I write it. I have left a few of these in Questions, Blockchain and Keys. I create an instance of the class, call methods, and print values to the console.

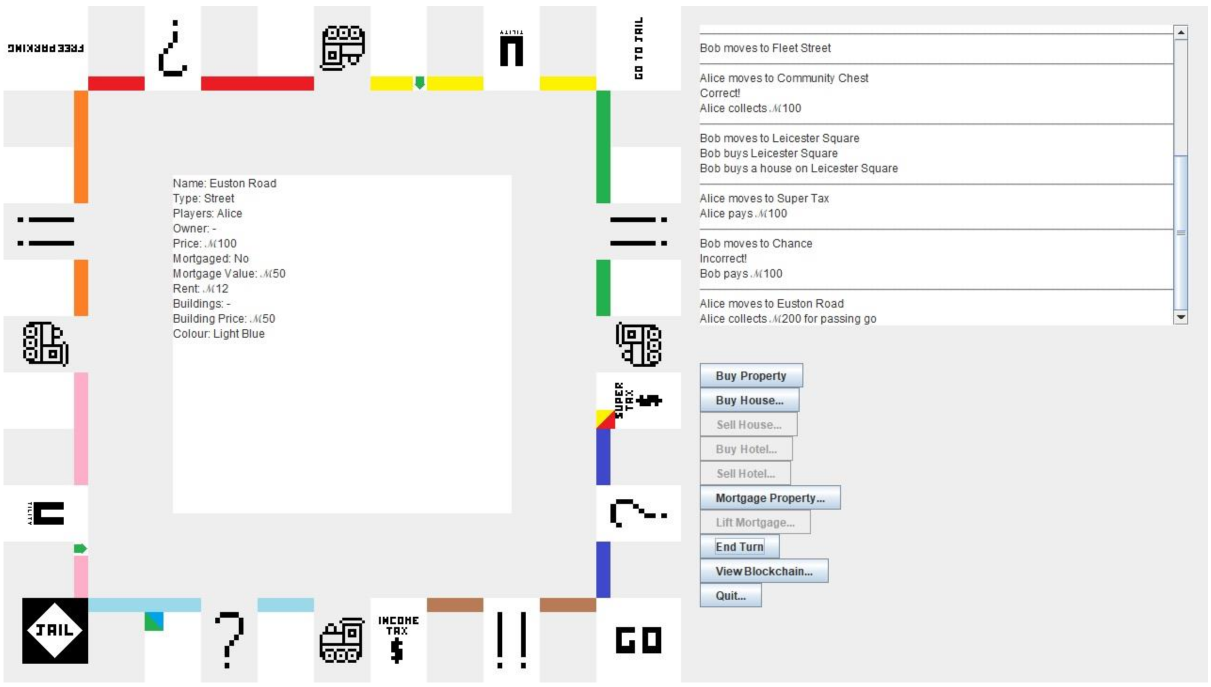
Second, I unit tested many of the classes. I wrote 26 JUnit tests, some of which have as many as 13 assertions. The classes tested are Keys, model.Board, Railroad, Street, Utility, CircularList, Convert and Util. I was able to rerun some or all of the unit tests whenever I made changes to the code.

Unit testing Board was an unexpected challenge because of it being a singleton. In other tests, an instance of the class under test can be created in setUp() before each test is run. To get around the problem of Board's constructor and instance being private, I used reflection (Code Excerpt 6) to access and invoke reset(), a private method that creates a new instance.

Code Excerpt 6: BoardTest.setUp()

```java
@Before
public void setUp() throws Exception {
    Method method = Board.class.getDeclaredMethod("reset");
    method.setAccessible(true);
    method.invoke(null);
}
```

Third, I played the game. A lot. If I wanted to test houses and hotels, I changed each player's money to £999… to be able to afford as many of them as I wanted. If I wanted to test going to/getting out of jail, I rigged the chance and community chest cards to always send the player to jail. Figure 11 is a screenshot of me playing the game.



Name: Euston Road
Type: Street
Players: Alice
Owner: -
Price: ℳ100
Mortgaged: No
Mortgage Value: ℳ50
Rent: ℳ12
Buildings: -
Building Price: ℳ50
Colour: Light Blue

```
Bob moves to Fleet Street

Alice moves to Community Chest
Correct!
Alice collects ℳ100

Bob moves to Leicester Square
Bob buys Leicester Square
Bob buys a house on Leicester Square

Alice moves to Super Tax
Alice pays ℳ100

Bob moves to Chance
Incorrect!
Bob pays ℳ100

Alice moves to Euston Road
Alice collects ℳ200 for passing go
```

Figure 11: Playing the game

In this chapter, I have implemented and tested a Monopoly clone (model, view and controller) and a blockchain (front end and back end) and plugged the latter into the former. At 3144 source code lines across 43 classes, enums and interfaces, the most challenging thing about this project was not writing one method or fixing one bug, but getting everything to work harmoniously, which I did.

6. Evaluation

With respect to evaluation, I will think about what it is I want to learn and compare a number of approaches I could take.

I thought of two good ways to evaluate what I had done and go some way to answering my research questions. The first was to carry out an experiment like the ones in the Hundhausen et al. paper. I would randomly divide participants into two groups. One would play my game and the other would use whichever learning method I wanted to compare it against, like reading a textbook or browsing Bitcoin.com. Both groups would then be given the same test. If the experiment was controlled, the scores of each group could be used to say something about the relative effectiveness of the learning method they used. However, with the deadline looming, I knew the chance of having enough participants for the experiment to be statistically significant, and for the lessons learned from it to carry any weight, were extremely small. It would also be necessary to test participants before, as well as after, to measure the change in their knowledge and understanding.

Another way was to survey or interview participants. This would yield both qualitative and quantitative data and could guide the research by raising questions other than those this report attempts to answer. I felt like this was more achievable and got to work designing a Qualtrics survey with questions for participants to answer after playing my game for about 30 minutes. I wanted to ask each participant whether they think an educational game could be valuable as a way to learn about cryptocurrency, how it compares to other learning methods they have used, and about the usability of the game and what could be done to maximise it.

With these goals in mind, I asked

- How would you describe your knowledge of cryptocurrency? (Q3)
- Which method(s) have you used to learn about cryptocurrency? Select all that apply. (Q4)
    - (If they select "other":) Which other method(s) have you used to learn about cryptocurrency?
- How easy/difficult was the program to use? (Q6)

    - (If they don't answer "extremely easy" or "somewhat easy":) What would make the program easier to use?
- Do you think this method of learning about cryptocurrency could be valuable? (Q8)
- For each of the methods you selected earlier, how much better/worse was this method? (Q9)
- Do you have any other feedback?

The question numbers in brackets are used to caption the figures and tables in this chapter. The I used features like skip conditions, display logic, carry forward statements and validation to tailor the survey flow to the participant's answers. For example, a participant is only asked "what would make the program easier" to use if they say it was "neither easy nor difficult", "somewhat difficult" or "extremely difficult" to use.

I familiarised myself with the "Ethics in CIS" page on the departmental website and submitted an ethics application (ID 996), which was approved August 5th. A copy of this can be found in Appendix 2.

I distributed a link to the survey in the "CES – Strathclyde" Facebook group and asked my friends if they would like to help me by participating. I received eight responses.

6.1. Knowledge of Cryptocurrency

Figure 12 and Table 4 were generated from the responses to Q3, which asked participants how they would describe their knowledge of cryptocurrency. The purpose of this question was to put the rest of the answers in context (a high school student who knows nothing about Bitcoin might have different insights to, for example, someone on Crypto Weekly's "Top 100 Most Influential People In Crypto" list.)

Everyone acknowledged they had at least something left to learn about cryptocurrency, as no one answered "excellent". The mean was 3.38, somewhere between "good" and "average". The standard deviation of 1.22, the highest of any question in the survey, suggested I captured a group of people whose knowledge of cryptocurrency was mixed.

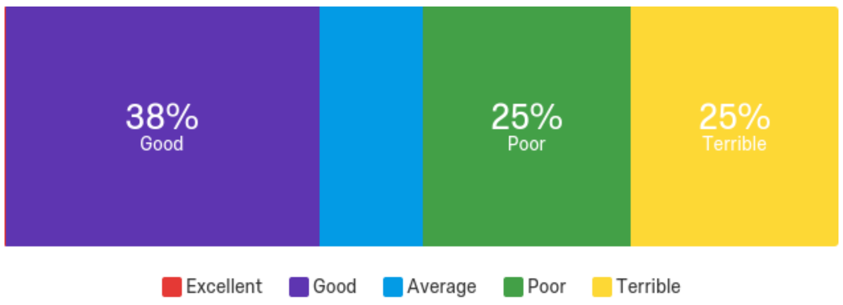


Figure 12: Q3 breakdown bar

Table 4: Q3 statistics table

#	Field	Mean	Std Deviation	Variance	Count
1	How would you describe your knowledge of cryptocurrency?	3.38	1.22	1.48	8

6.2. Learning Methods

Q4 asked participants which methods they have used to learn about cryptocurrency. The most surprising thing about the answers (Figure 13) was that none of the participants had used a textbook to learn about cryptocurrency.

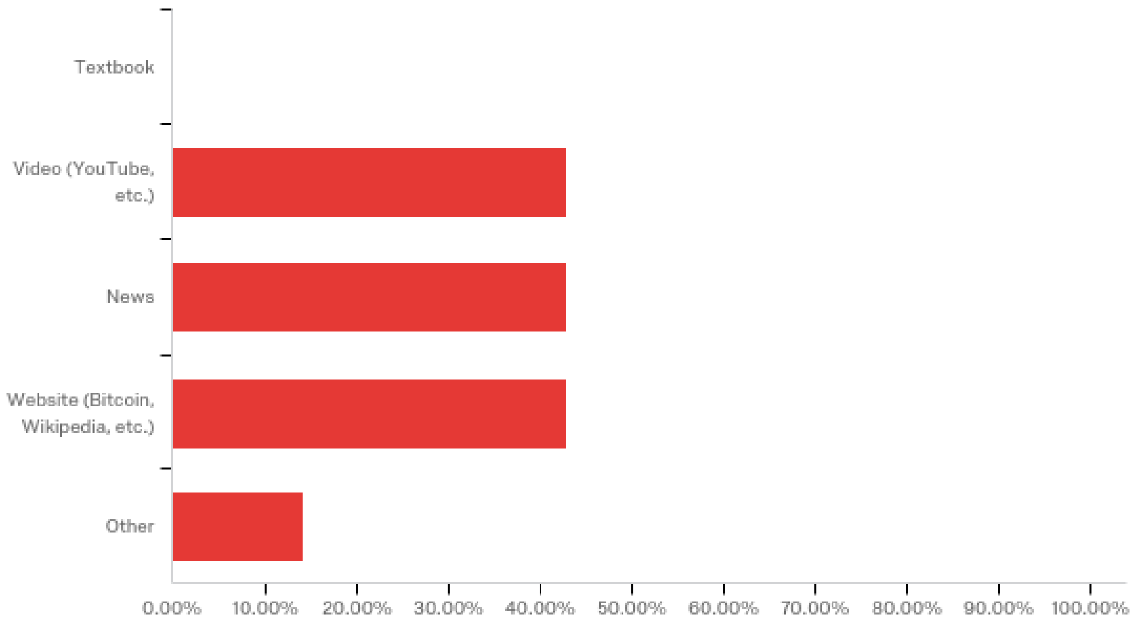


Figure 13: Q4 bar chart

With such a small number of participants, this could be nothing more than a coincidence, but it could also be a sign that students find textbooks less engaging and would prefer to learn about cryptocurrency in other ways. The participant who answered "other" had heard cryptocurrency talked about on a podcast.

6.3. Usability

The answers to Q6, which asked participants how easy/difficult the program was to use, must be taken with the caveat that the participants were all computing students who are used to figuring out how to use programs and comfortable doing so.

None of the participants found the program difficult to use, as Figure 14 and Table 5 show. One participant suggested the program could be made easier to use by showing each player's money (something you can only see by clicking on their token). This is a change that would be easy to make.

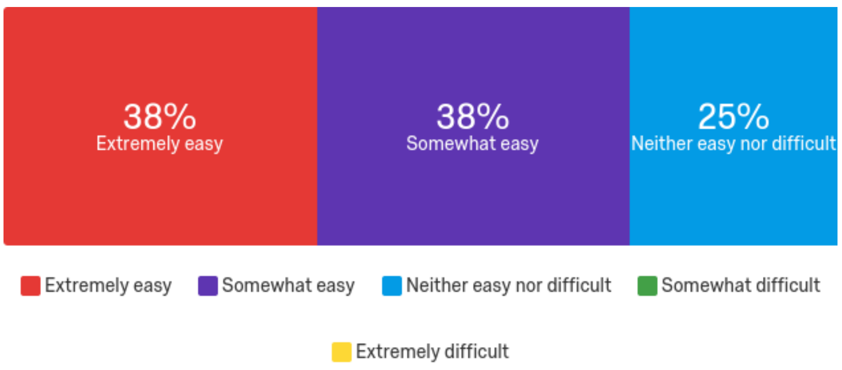
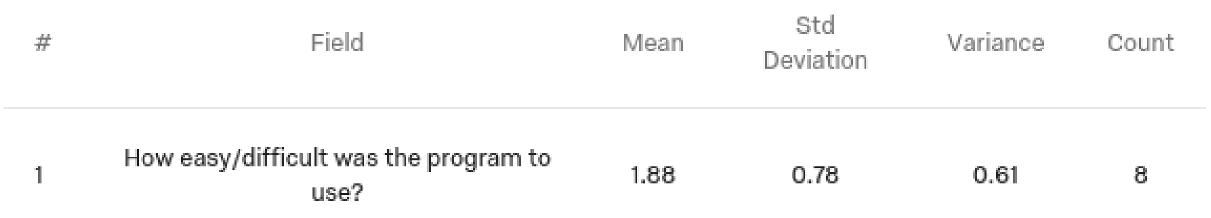


Figure 14: Q6 breakdown bar

Table 5: Q6 statistics table



#	Field	Mean	Std Deviation	Variance	Count
1	How easy/difficult was the program to use?	1.88	0.78	0.61	8

6.4. Value

Q8 asked participants if they thought this method of learning about cryptocurrency could be valuable. The answers (Figure 15 and Table 6) were mixed, but over 50% of the participants answered in the positive, and none answered "definitely not".

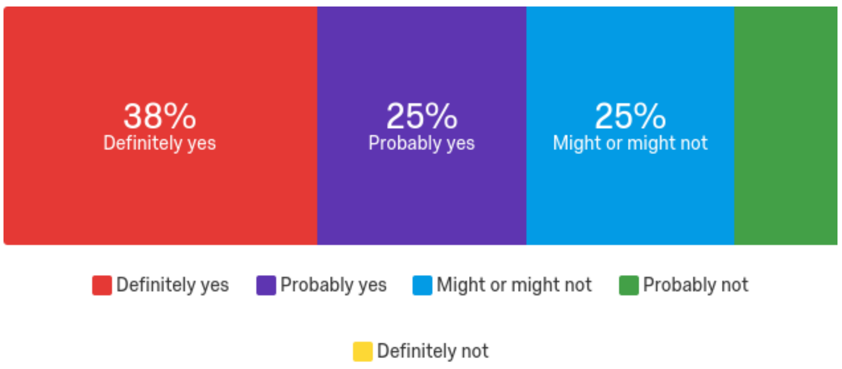
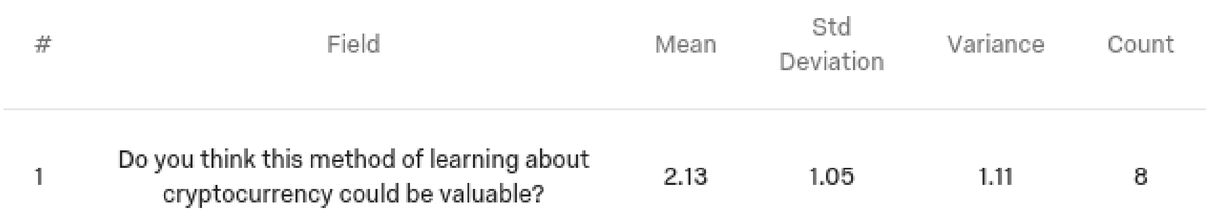


Figure 15: Q8 breakdown bar

Table 6: Q8 statistics table



#	Field	Mean	Std Deviation	Variance	Count
1	Do you think this method of learning about cryptocurrency could be valuable?	2.13	1.05	1.11	8

6.5. Compare

In Q9, participants compared this method to others they had used. Figures 16, 17 and 18 show how they felt about learning by playing “Monopoly with Bitcoin” compared to by reading a website, reading/watching the news and watching a video respectively.

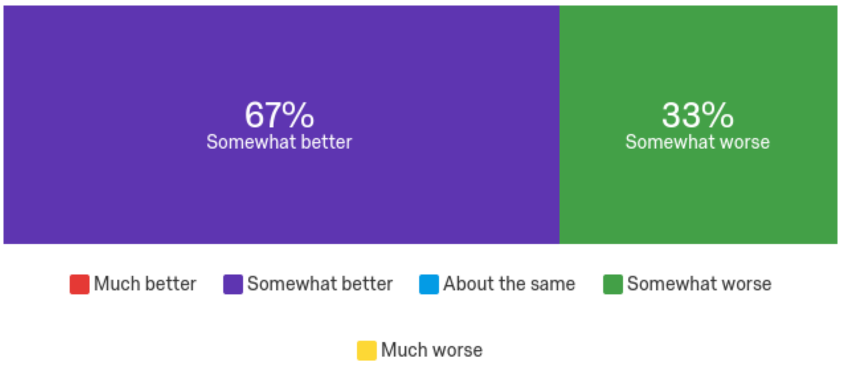


Figure 16: Q9 breakdown bar (website)

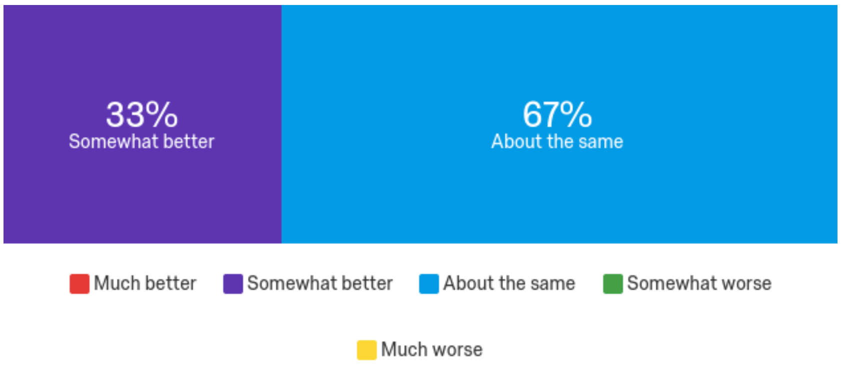


Figure 17: Q9 breakdown bar (news)

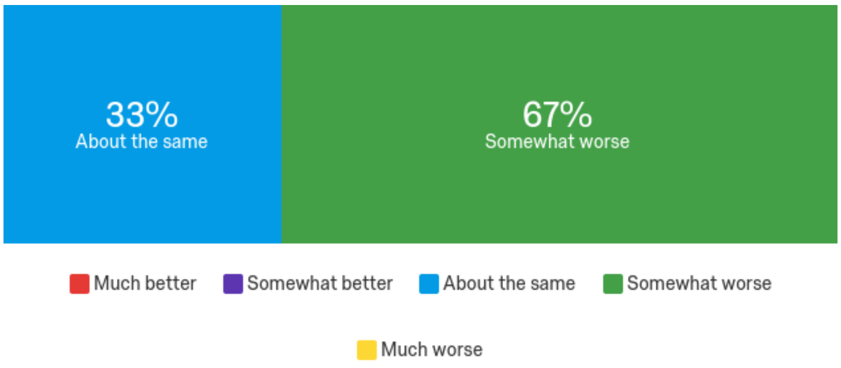
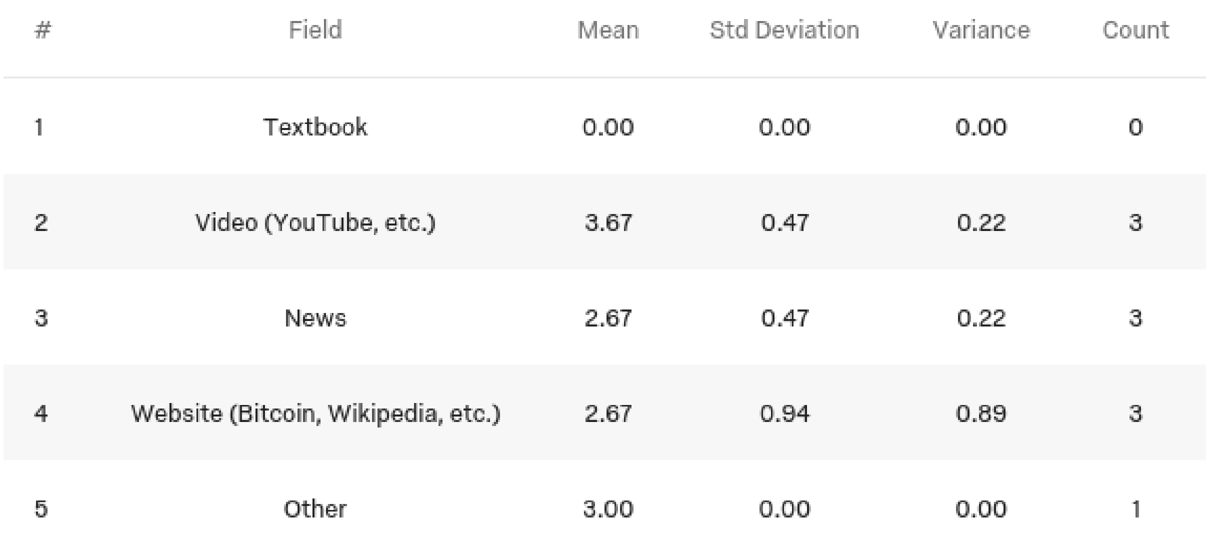


Figure 18: Q9 breakdown bar (video)

The most favourable comparison was to reading a website. On the other hand, all three of the participants who had watched a video about cryptocurrency agreed that method was better. This is backed up by the statistics in Table 7.

Table 7: Q9 statistics table



#	Field	Mean	Std Deviation	Variance	Count
1	Textbook	0.00	0.00	0.00	0
2	Video (YouTube, etc.)	3.67	0.47	0.22	3
3	News	2.67	0.47	0.22	3
4	Website (Bitcoin, Wikipedia, etc.)	2.67	0.94	0.89	3
5	Other	3.00	0.00	0.00	1

The lessons learned from this evaluation prove this method of learning about cryptocurrency to be at least worthy of exploration, and at best a viable alternative to, for example, reading a textbook. They also hint at ways in which it could be made better, which will come under "7.1. Future Work" in the next chapter.

7. Conclusions and Future Work

In this dissertation, I have researched, designed, implemented, tested and evaluated a “Monopoly with Bitcoin” game to explore whether it could be an engaging way to learn how Bitcoin and other cryptocurrencies work. A majority of participants thought it could be valuable, and a plurality preferred it to other sources of the same information (websites, news). If more participants could be found, an experiment could be carried out to support the positive results of the survey.

There is room for improvement. Adding tooltips or videos to describe the different parts of the blockchain, blocks and transactions could make it easier to learn about. There could even be a way for lecturers, etc. to add their own videos that would be triggered at predetermined points, like the completion of a transaction or the viewing of a block. This would enable them to tailor the user’s experience to their level. Embedding a video in the existing Swing code could be achieved using an API like JMF (Java Media Framework) or FMJ (Freedom for Media in Java), which support lots of different formats.

As a game, “Monopoly with Bitcoin” could be developed by

- allowing more than two players to play at the same time
- adding an AI to let one person play against the computer
- writing netcode (from scratch) to allow two or more people to play online
- porting it to JavaScript to make it easier to distribute
- allowing saving and loading by serializing the state of the game and writing it to/reading it from a file

to list but a few ideas.

References

Chaum, D. (1983). Blind signatures for untraceable payments. In *Advances in cryptology* (pp. 199-203). Springer, Boston, MA.

Chaum, D., Fiat, A., & Naor, M. (1988, August). Untraceable electronic cash. In *Conference on the Theory and Application of Cryptography* (pp. 319-327). Springer, New York, NY.

Chohan, U. W. (2017). Cryptocurrencies: A brief thematic review.

Reese, F. (2018). *7 Benefits of Decentralized Currency*. Bitcoin Market Journal. Retrieved from https://www.bitcoinmarketjournal.com/decentralized-currency/

Greenberg, A. (2011). *Crypto Currency*. Forbes. Retrieved from https://www.forbes.com/forbes/2011/0509/technology-psilocybin-bitcoins-gavin-andresen-crypto-currency.html

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press.

Laplante, P. A., & Neill, C. J. (2004). The demise of the waterfall model is imminent and other urban myths. *ACM Queue, 1*(10), 10-15.

Scottish Government. (2018, September 25). Recorded crime in Scotland 2017-2018. Retrieved from https://www.gov.scot/publications/recorded-crime-scotland-2017-18/pages/6/

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

"BoE explores implications of blockchain". (2016, September 9). Retrieved from http://www.econotimes.com/BoE-explores-implications-of-blockchain-and-central-bank-issued-digital-currency-277718

Godshall, J. (2018). Five Successful 51 Percent Attacks Have Earned Cryptocurrency Hackers $20 Million in 2018 - UNHASHED. Retrieved from https://unhashed.com/cryptocurrency-news/five-successful-51-attacks-earned-hackers-20-million-2018/

Hershkowitz, R. (1989). Visualization in Geometry--Two Sides of the Coin. *Focus on learning problems in mathematics, 11*, 61-76.

Jacoby, W. G. (1997). *Statistical graphics for univariate and bivariate data* (No. 117). Sage.

Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools, 29*(1-2), 95-117.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing, 13*(3), 259-290.

McNear, C., & Pettey, C. C. (2005, March). A free, readily upgradeable, interactive tool for teaching encryption algorithms. In *Proceedings of the 43rd annual Southeast regional conference-Volume 1* (pp. 280-285). ACM.

Ma, J., Tao, J., Keranen, M., Mayo, J., Shene, C. K., & Wang, C. (2014, June). SHAvisual: a secure hash algorithm visualization tool. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 338-338). ACM.

Fowler, M., & Kobryn, C. (2004). *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional.

Holub, A. (2019). Why extends is evil. Retrieved from https://www.javaworld.com/article/2073649/why-extends-is-evil.html

Krasner, G. E., & Pope, S. T. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming, 1*(3), 26-49.

Long, E. (2018). How to Create Your Own Cryptocurrency. Retrieved from https://lifehacker.com/how-to-create-your-own-cryptocurrency-1825337462

van Flymen, D. (2017). Learn Blockchains by Building One. Retrieved 16 August 2019, from https://hackernoon.com/learn-blockchains-by-building-one-117428612f46

Appendix 1: Use Cases

Monopoly

1: Move Token

A player starts their turn by throwing their dice and moving their token that number of spaces.

Main Success Scenario:

1. System shows “throw the dice” option
2. Player chooses “throw the dice” option
3. System randomly generates number between 2 and 12
4. System shows “move your token” option
5. Player chooses “move your token” option
6. System moves player’s token that number of spaces in the direction of the arrow

Extensions

1a: Player throws doubles

If a player throws doubles, they get to go again.

.1: System shows “you threw doubles” prompt

.2: Player throws again

.3: Player moves their token as before

If a player throws doubles three times in a row, they go to jail.

1b: Player throws doubles three times in succession

.1: System moves player’s token to space marked “In Jail”

2: Pass Go, Collect $200

If a player lands on or passes over Go, the first space on the board, they get $200.

Main Success Scenario:

1. Player’s token lands on or passes over GO
2. Banker pays him/her a $200 salary

3: Buy Property

If a player lands on an unowned property, they can buy it.

Precondition: Player has enough money to buy property

Main Success Scenario:

1. Player lands on unowned property

2. System shows “buy property” and “skip” options

3. Player chooses “buy property” button

2. Player may buy that property from the bank at its printed price

3. Player receives the title deed card showing ownership

Extensions:

3a: Player does not wish to buy property

.1: Player chooses “skip” option

.2: Banker sells property at auction to the highest bidder (each player is prompted to bid)

.3: Buyer pays the bank the amount of the bid in cash

.4: Buyer receives the title deed card for that property

4 Pay Rent

If a player lands on someone else’s property, they have to pay rent.

Main Success Scenario:

1. Player lands on property owned by another player

2. System shows “pay rent” option

3. Player chooses “pay rent” option

2. Owner collects rent from player in accordance with the list printed on its title deed card

Extensions:

4a: Property is mortgaged

.1: No rent can be collected

4b: Owner holds all the title deed cards in a color-group

.1: Owner may then charge double rent for unimproved properties in that color-group

5: Use Chance, Community Chest Cards

If a player lands on a chance or community chest space, they draw a card and follow the instructions.

Main Success Scenario:

1. Player lands on either of these spaces
2. Player takes top card from deck indicated
3. System shows prompt
4. Player follows the instructions

Extensions:

5a: Player draws "Get Out of Jail Free" card

.1: Player owns card

.2: Player can sell card at any time (system shows button during their turn)

6: Pay Income Tax

If a player lands on the income tax space, they have to pay $200 or 10% of their total worth.

Main Success Scenario:

1. Player lands on the space marked "Income Tax"
2. System shows two options, "pay $200" and "pay 10% of your total worth"
3. Player chooses one of the options
4. Player pays that much money

7: Get Out of Jail

There are a few ways for a player to get out of jail. They can try to throw doubles (they get out if they fail three times) or just pay $50.

Main Success Scenario:

1. System shows two options, “try to throw doubles” and “pay $50”

2. Player chooses one of the options

3. System resolves throwing of doubles/paying of fine

Extensions:

7a: Player owns “Get Out of Jail Free” card

.1: System shows “use Get Out of Jail Free card” option

.2: Player chooses option

.3: System returns card to bottom of deck

.4: System moves player’s token to “Just Visiting”

7b: Player has failed to throw doubles three times in a row

.1 System move’s player’s token to “Just Visiting”

8: Use Free Parking

The free parking space is just a blank space.

Main Success Scenario:

1. Player lands on “Free Parking” space

2. Nothing happens

9: Buy Houses

Houses increase rent. You can only buy a house on, say, a yellow property if you own all three yellow properties.

Precondition: Player has enough money to buy house, player owns all properties of that color-group

Postcondition: House is added to player’s street

Main Success Scenario:

1. Player clicks street they own

2. System shows “buy house” option

3. Player chooses “buy house” option

4. Player pays cost of house

Extensions:

9a: Player doesn’t have enough money

    .1: “Buy house” option is greyed out

10: Buy Hotels

Hotels increase rent even more. You can only buy a hotel if you own four houses.

Precondition: Player has enough money to buy hotel and owns four houses on street

Postcondition: Houses are removed, hotel is added to player’s street

Main Success Scenario:

1. Player clicks street they own
2. System shows “buy hotel” option
3. Player chooses “buy hotel” option
4. Player pays cost of hotel

Extensions:

10a: Player doesn’t have enough money

    .1: “Buy hotel” option is greyed out

10b: Player doesn’t own four houses on street

    .1: “Buy hotel” option is greyed out

11: Sell Property

Players can sell property to each other for whatever price both parties want.

Precondition: Player owns property, no buildings are standing on any properties of that color-group

Main Success Scenario:

1. Player navigates to property they own
2. System shows “sell property” option
3. Player chooses “sell property” option
4. Player chooses player to sell property to

5. Buyer enters price

6. Buyer pays that much money

7. Property is transferred

Extensions:

11a: Player sells property to bank

.1: Bank pays half price paid for property

11b: Property was mortgaged

.1: New owner may lift mortgage by paying off mortgage plus 10% interest

.2: If mortgage is not lifted, new owner must pay bank 10% interest when they buy property

12: Mortgage Property

Properties can be mortgaged, meaning rent cannot be collected on them.

Precondition: Player owns property

Main Success Scenario:

1. Player clicks on property they own

2. System shows “mortgage property” option

3. Player clicks “mortgage property” option

4. Player receives mortgage value from bank

5. System marks property as mortgaged

13: Lift Mortgage

Likewise, properties can be unmortgaged.

Precondition: Player owns property, property is mortgaged

Main Success Scenario:

1. Owner pays bank amount of mortgage plus 10% interest

2. Property is no longer mortgaged

14: Go Bankrupt

The win condition is to be the last player standing after everyone else goes bankrupt.

Precondition: Player A owes more than they can pay to player B or bank

Main Success Scenario:

1. Everything player A owns is transferred to player B or bank

2. Player A stops playing

Extensions:

14a: Everyone else goes bankrupt

.1: The last player standing wins

Cryptocurrency

15: View the Blockchain

Main Success Scenario:

1. Player asks to view blockchain

2. System shows blockchain (blocks)

3. Player can view any of the blocks in the blockchain

16: View a Block

Main Success Scenario:

1. Player asks to view block

2. System shows block (index, timestamp, transactions, proof, hash of previous block)

3. Player can view the blockchain or any of the transactions in the block

17: View a Transaction

Main Success Scenario:

1. Player asks to view transaction

2. System shows transaction (sender, recipient, amount, index of block)

3. Player can view the block the transaction is recorded in

18: Watch a Step by Step Animation of a Transaction

Main Success Scenario:

1. Transaction takes place in game (e.g. player buys property)

2. System shows step by step animation of transaction

Other

19: Save the Game

Main Success Scenario:

1. Player chooses “save game” option

2. Player chooses name and location

3. System saves game to file

Extensions:

19a: Player tries to quit without saving

    .1: System gives player chance to save

19b: Player chooses existing file

    .1: System asks player to confirm

    .2: Player confirms

    .3: System overwrites existing file

20: Load the Game

Main Success Scenario:

1. Player chooses “load game” option

2. Player chooses file

3. System tries to load game from file

Extensions:

20a: File is corrupted

.1: System does not load game

.2: System shows error

21: Play Against the Computer

Main Success Scenario:

1. Human player adds one or more computer players

2. Computer players play

Appendix 2: Ethics Application

Application ID: 996

Title of research:
Monopoly with Bitcoin

Summary of research (short overview of the background and aims of this study):
Bitcoin and other cryptocurrencies were invented in 2009 but were obscure until a couple of years ago. In my Advanced Computer Science dissertation, "Monopoly with Bitcoin", I have developed a Java version of the board game "Monopoly" in which Monopoly money is replaced by artificial cryptocurrency, with the intent that it could be used to explore the concepts of blockchain and cryptocurrencies. This survey is part of an evaluation of whether this way of learning could be valuable in educating people about cryptocurrency.

How will participants be recruited?
I will write a post in the private "CES – Strathclyde" Facebook group asking if anyone would like to participate in my study.

What will the participants be told about the proposed research study? Either upload or include a copy of the briefing notes issued to participants. In particular this should include details of yourself, the context of the study and an overview of the data that you plan to collect, your supervisor, and contact details for the Departmental Ethics Committee.
The participants will know me personally and be able to contact me via email or Facebook. The details listed (supervisor, ethics committee contact details) are included in the opening block of my survey.

How will consent be demonstrated? Either upload or include here a copy of the consent form/instructions issued to participants. It is particularly important that you make the rights of the participants to freely withdraw from the study at any point (if they begin to feel stressed for example), nor feel under any pressure or obligation to complete the study, answer any particular question, or undertake any particular task. Their rights regarding associated data collected should also be made explicit.
The survey opens with the following question:

"Please confirm you understand:

• your participation is voluntary
• you can withdraw your data at any time
• you are under no obligation to answer any question
• your data will be handled in accordance with the Data Protection Act 1998 and the General Data Protection Regulation 2016"

Participants can only advance to the rest of the survey if they select "yes".

What will participants be expected to do? Either upload or include a copy of the instructions issued to participants along with a copy of or link to the survey, interview script or task description you intend to carry out. Please also confirm (where appropriate) that your supervisor has seen and approved both your planned study and this associated ethics application.

Participants will be expected to download the game I have developed and play if for about 30 minutes before answering the survey. A preview of the survey can be found at https://strathsci.eu.qualtrics.com/jfe/preview/SV_cZl5sNCr8u0VdaZ?Q_SurveyVersionID=current&Q_CHL=preview.

What data will be collected and how will it be captured and stored? In particular indicate how adherence to the Data Protection Act and the General Data Protection Regulation (GDPR) will be guaranteed and how participant confidentiality will be handled.

Participants will be asked:

• How would you describe your knowledge of cryptocurrency?
• Which method(s) have you used to learn about cryptocurrency? Select all that apply.
o (If they select "other"): Which other method(s) have you used to learn about cryptocurrency?
• How easy/difficult was the program to use?
o (If they don't answer "extremely easy" or "somewhat easy"): What would make the program easier to use?
• Do you think this method of learning about cryptocurrency could be valuable?
• For each of the methods you selected earlier, how much better/worse was this method?
• Do you have any other feedback?
.
They will also be asked to enter their email address to identify their data in the event they want it withdrawn.

The data will be captured and stored using Qualtrics, an ISO 27001 certified service, and will be password protected. In the event I downloaded the data, for example to include it as an appendix in my report:
• I will delete the email addresses so any instance of the data other than the original is anonymised.
• I will store it on the hard drive of my PC, which is password protected and never leaves the safety of my room.

How will the data be processed? (e.g. analysed, reported, visualised, integrated with other data, etc.) Please pay particular attention to descibing how personal or sensitive data will be handled and how GDPR regulations will be met.

The data will be processed using Qualtric's data & analysis features. Visualisations of some answers may be added, like a table or chart. Individual responses may be reported. In all cases, the data will be anonymised.

How and when will data be disposed of? Either upload a copy of your data management plan or describe how data will be disposed.

Data will be disposed as soon as I have my degree, at which time I will delete it from Qualtrics and delete any copies of it from my hard drive using something like File Shredder to make sure it cannot be recovered.