Conducted Research into an AI Opposition within a Game

Introduction

The following report documents the research and development of an AI Opposition that we wanted to implement within the game. This form of AI would implement a simple, but effective strategy within the game of Property Tycoon to provide a test for the human opposition. Since a game of Property Tycoon could possibly include more than one AI opposition, potentially all six of the players, this strategy should also need a bit of chance in order to be effective. This type of chance could include landing on the right properties, being able to get groups of properties quickly and cheaply, along with not landing on to many negative tiles. Negative tiles for instance could be the tax tiles, opponents properties, or landing on a tile that could lead to a negative impact on the player.

Implementations of AI in other Games

The fact that a game of Property Tycoon cannot be completely strategized, and does require a huge luck factor, the AI could potentially be either the best-off player in the game, or the worst-off player. This means unlike board games such as chess where the AI opponent can be levelled from an easy to a hard-level opponent, the type of opponent we wish to implement will try a standard potentially known way of winning which is less predictable, but not impossible to beat.

The main reason we don't wish to implement an incredibly amazing or stupidly awful opponent for the player is to ensure it doesn't dampen the entertainment factor, which is usually the main reason people are playing the games. By making a more universally skilled opposition, the game invites players of all ages and skill level to take on the opposition.

In chess, one example of an AI opponent is Stockfish. Stockfish is a six-time world champion program that has the ability to look through 60 million potential moves per second. This is possible due to the use of up to 512 CPU threads in multiprocessor systems. It implements an advanced alpha-beta search and uses bitboards. Alpha-beta search, also known as Alpha-beta pruning is an adversarial search algorithm. (Rin, 2018)

This technique works by stopping a move's evaluation when at least one possibility has been found that proves the move to be worse than a previously examined move, thus decreasing the number of nodes evaluated by the minimax algorithm, by trimming away branches that cannot influence the final decision (Aradhya A. L., 2017). A minimax algorithm for reference is a recursive algorithm for choosing the next move in game, usually implemented in two-player game. (Aradhya A. L., 2017)

Even with all the power, resources and techniques used, last year Deep Mind released an updated version of their machine-learning chess opposition named Alpha Zero. This AI opponent is trained solely using 'self-play' only with the pre-existing knowledge of the rules to chess. After 4 hours of training, was playing at a higher rank than Stockfish. In addition to another 5 hours of training, Alpha Zero defeated Stockfish in a time-controlled game 28 times out of 100, drawing all other 72 games.

(Somers, 2018) (Klein, 2017)

Techniques like these are not possible for this project, since they require large teams to develop & maintain, along with much more experience within the field to implement effectively, not forgetting

the large amount of resources needed to use this technology. However, after this research I know have a much better understanding of what types of AI might be feasible within this project.

After further research, I was able to find the source code to the 1997 Monopoly game for the PS1 (Retail Game Source Code, 2018) , coded in C++. After evaluating the AI class, techniques such as seeing whether the opposition would get achieve a ' Monopoly' from buying a certain property the AI has landed on, and checking whether the property is useful to the AI to help achieve a Monopoly for them are more feasible and realistic techniques that could be used within our implementation. In addition to this, I have also acquired a book called 'The Indisputable Existence of Santa Claus' which has a dedicated chapter (10) to how to mathematically win monopoly. (gmdirect, 2018)

More in-depth research on mathematical approach

From the previous research I have determined that the best form of AI that we would like to implement within the game would be one to combine the mathematical side of winning, shown in 'The Indisputable Existence of Santa Claus', along with basic decision making such as, deciding whether to buy a property if it is a need asset for the other players, cleverly displayed in the old Monopoly source code. All of the following figures will be taken directly from 'The Indisputable Existence of Santa Claus'.

Through reading the book, the book clearly conveys that in a standard game of Monopoly, or Property Tycoon, some squares are much more likely to be landed on than others. This is because the dice rolls of a player are not the only factor that can move the players token around the board.

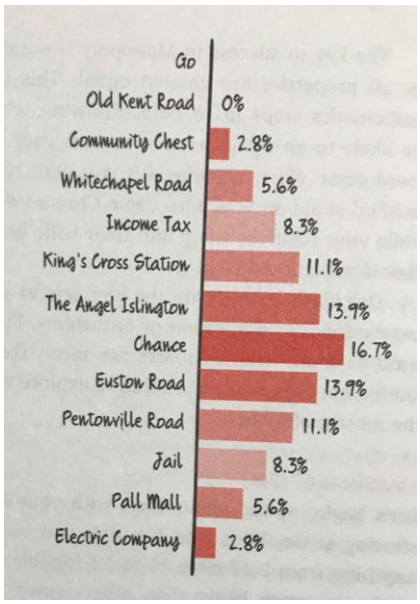

Figure 1 – Probability of landing on a tile after the first roll.

Figure 1 effectively displays the probability of your position after your first turn. But that's not technically correct.

Due to the Pot Luck and Opportunity Knocks cards, known as Chance and Community Chests in a game of Monopoly, can also move the player around the board, if the player appears to land on them. For example in a game of Property Tycoon, 3 of the 16 Pot Luck cards move the player around the board (one being to jail), and 5 of the 16 Opportunity Knocks move the player around the board (again one being jail). This means if the Opportunity Knocks card deck is shuffled, the player has a 31% chance of being moved to another space if they are the first to land on the tile. In addition, to these there is another curveball within the mix. If a

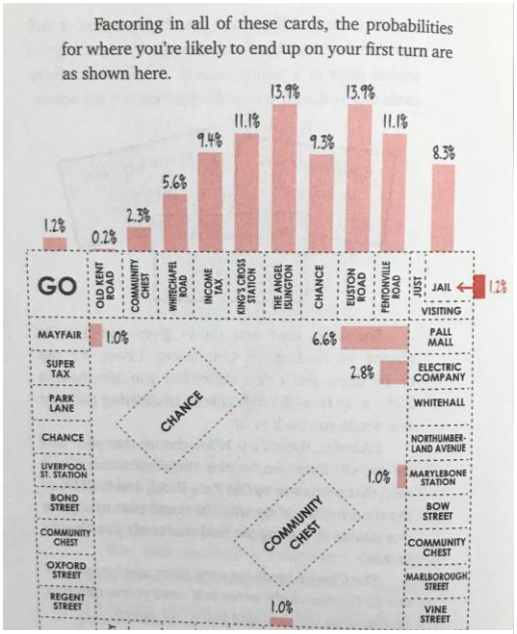

Figure 2 – Probability of landing on a tile, including wildcards such as Pot Luck & Opportunity Knocks.

player rolls 3 doubles in a row, they are sent straight to jail. This is another factor to think of when describing the movement of a player within the game. Without such 'curveballs', standard dice rolls would result in all tiles on the board having an equal probability to be landed on after x amount of rolls. These curveball clearly do have some effect on the probability of landing, shown in Figure 2. For example a player has a 1.2% chance on ending up in jail (not the visiting square which is only taken in account in figure 1), meaning that in 1 every approximate 82 games of Property Tycoon, the first roll could result in the player being in jail. Through using Markov chains, the probability of landing on each tile in a game of Property Tycoon (Monopoly in these cases), is clearly portrayed in Figure 3. Markov Chains (also known as transition probability) are a theoretical way of describing the probability of a single tile to all other tiles. By doing this at each tile, the probability of landing on each tile during a game of Property Tycoon. This next reference is a segment of a paper produced by the University Of Auckland, New Zealand and does a very good job at conveying what Markov Chains are and how they work. (The University of Auclkand, New Zealand, 2015).

To try and put it simply, imagine you are the Goblet token, and every roll and move you distribute a fraction of the goblet according to the probability of it landing on that tile. This is repeated effectively until a 'fragment' of the goblet on every single tile of the board, where the goblet is distributed proportional to the odds of being on any one square.

These show that Jail is the most often landed tile, and Hawking Way (Park Lane) is the least likely (excluding Go to Jail as no turn would end on that tile, as player is also moved immediately to the Jail tile).

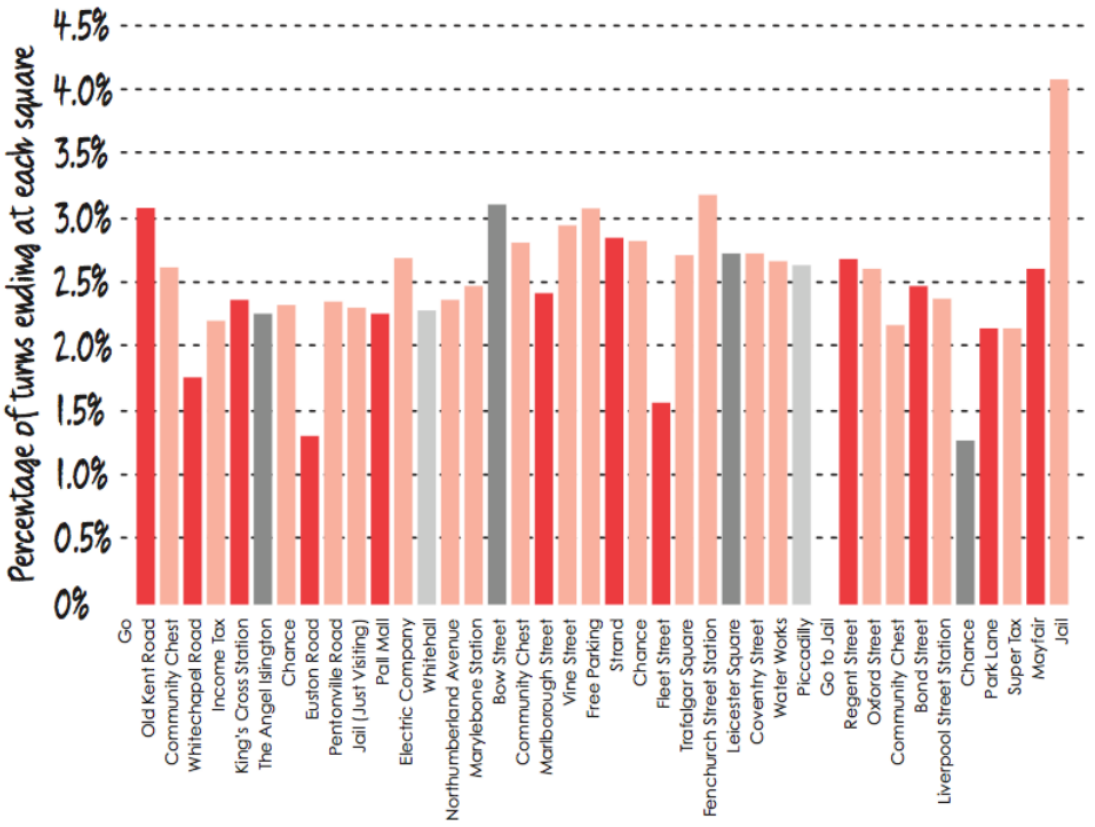

Figure 3 – Probability of landing on a tile in a game of Monopoly (Property Tycoon)

From Figure 3, you must believe that investing in Trafalgar Square (Hans Xin Gardens in Property Tycoon), since it's the property with the highest probability on being landed on, possibly because

there's an Opportunity Knocks card that sends the player forward to that square. But as Hannah Fry (Author of said book I've been referencing) said "Monopoly is not all about the probabilities, it's about winning money and crushing your relatives." This quote further gives advance to the fact that even though a certain tile on the board may be the most visited tile, a winning strategy must be adopted in order to win the game. She further describes that the winning strategy needs to take into account what you'd earn if your opposition lands on your owned property. Luckily, Hannah Fry has also calculated what property would generate the most profit taken into account its odds of being landed on by one single opponent, shown in Figure 4.

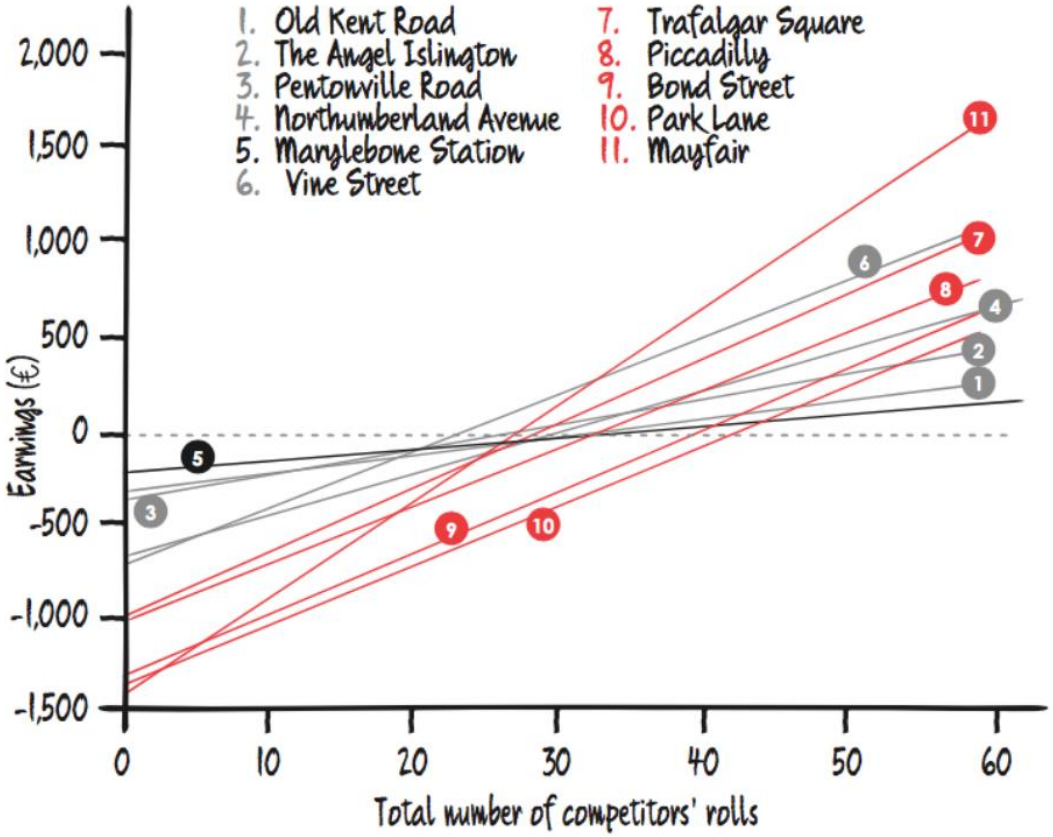

Figure 4 – Using probability of landing on a tile, calculating profit accumulated after x number of competitors rolls onto said tile. (All properties all have a hotel on them)

Figure 4 effectively demonstrated a players expected return on a selection of properties from each set, with a hotel on them (To maximise profits). The graph doesn't display the income (revenue) but the profit that is made after a number of rolls. This because each property will start off with a negative value, as the player must first of all buy the property, and then buy an additional four houses and a hotel. For properties such as Turing Heights (Mayfair in Monopoly), that is a purchase price of £400 plus £1000 in houses and hotels (£200 per house, £200 for hotel), meaning it starts with an earnings of -£1400. The fact it takes approximately 25 competitors' rolls for it to begin to make profit, and by 60 rolls, is evidently the most profitable property shown on the graph. So does this mean you should 100% definitely always buy Turning Heights/ Mayfair? No. Why not?

Well, if you know the rules of Property Tycoon (or equivalent) you'd know in order to purchase houses & hotels you need to own all colours of the set. This means you'd also need to buy Hawking Way/Park Lane and build at least 4 houses on it. As you can see in Figure 4, Park Lane is nearer to the worst of the bunch of properties taking approximately 40+ competitor rolls to start making a profit. So is the Deep Blue group still worth purchasing.

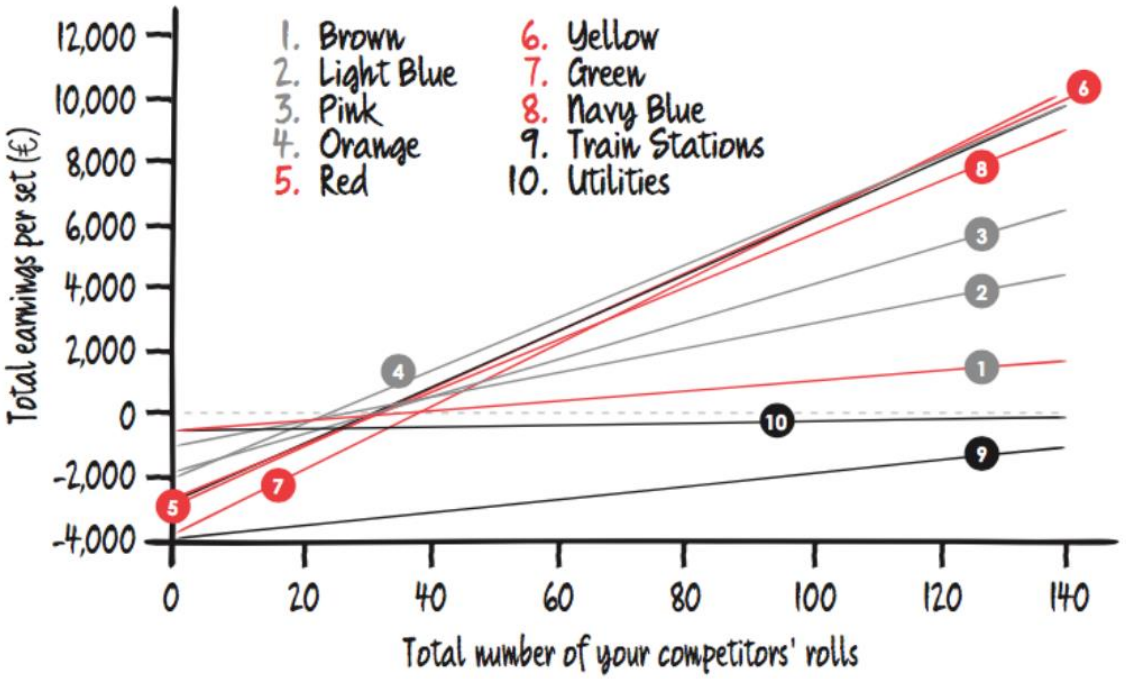

Figure 5 – Comparing the profits made on a complete set after a total of x number of competitors' roles

So, as shown in Figure 5, the Park Lane/Hawking Way effect of being a truly awful property to invest in with its low chance of being landed on, and its lacklustre ability to only begin to make profit after 40 opponent rolls, severely brings down the Deep Blue/Navy Blue colour group to 5th, after Mayfair being the clear and obvious favourite in Figure 4. So what does this mean for the best property group? Well the graph suggests that Green, Yellow/Red and Orange are the best to invest in after a total of greater than 140 opposition roles. Graph also conveys that Utilities and Stations are a no-go as neither are able to produce a profit even after 140 competitor rolls. To put this in perspective, approximately in an abridged game of Property Tycoon, or similar, where the time limit is around 3 hours, with a single turn taking a minute per player, a game of two players would last around 75 turns, three players 50 turns, and four players only 37 turns.

This further raises another question. How do the number of players affect how the game plays out? In addition how does the time period in the game affect how the game spans out? Does it matter how many houses are bought?

All valid questions which definitely do affect how a game of Property Tycoon runs. The number of players in the game changes the number of opposition roles, as there will be more. Also more players as stated before can lead to games being shorter with less turns for each player in a timed version. This means properties that make money back quickly, over big money colour groups may need to be prioritized. The book suggests that if the player is playing against one opponent, Light Blue & Orange are effective colour groups, especially early game, as shown in Figure 5, both begin to make profit after about 10 competitor rolls. For two players, they suggest again suggest Orange, but also Red. Finally with three or more opponents, Green is seen as the go to property group.

The time frame within the game, i.e. early game vs late game is an important factor, as the game progresses. This is because later in the game, players tend to have more money, since they have built up a Monopoly. The price of a property in the early game makes up much more of a percentage of the players balance than it does in the later game. This suggest cheaper groups such as Brown are great early game, but dreadful late game, while Green is not great early game, but the superior colour group in the late game.

Finally, the book suggests that 3 houses is the optimum amount of houses to get as soon as possible in the game. This is further supported by Figure 6 below. It is the best amount of houses to have early game as the time taken to recoup investment, i.e begin to make profit, is by far the quickest.

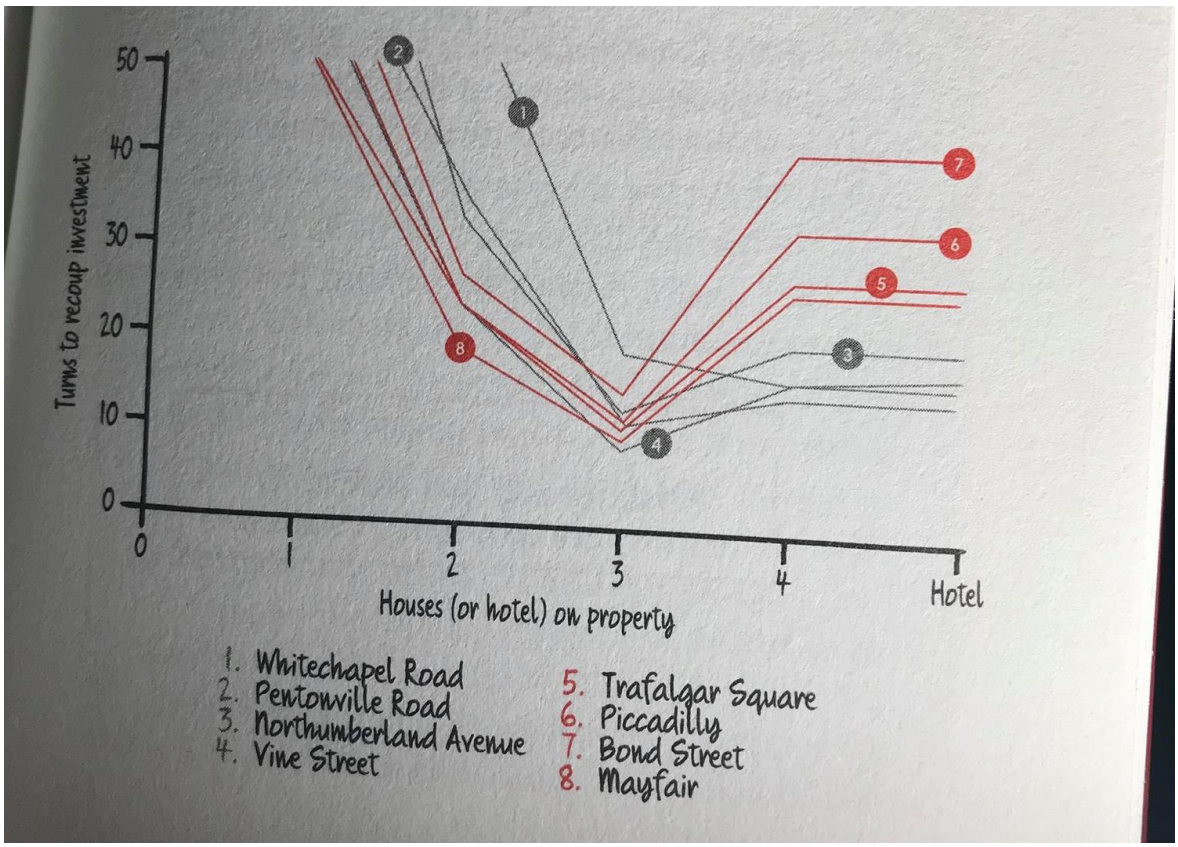

Figure 6 – Amount of time (in turns) it takes to recoup investment into houses/hotel per colour group.

An interesting fact to this strategy is that the bank only has limited number of houses. By claiming lots of houses early within the game, other players will not be able to effectively build their monopoly, further increasing the chance of victory (as long as there is no silly house rules that stop this tactic). So by buying 3 houses as soon as possible , you are not only increasing your potential profits, but you are also potentially limiting your opponents profits, and the aim of the game is to bankrupt your fellow players.

Testing these calculations

Before carrying out any coding of our own AI, and using the data presented to me, I felt like it was right to ensure that these findings presented to me have not been pulled out of thin air, which I highly doubt they are, but you can never be too safe. A way of testing the Markov Chain result presented amazingly in Figure 3, would be to do the Markov Chain myself. The problem with this is I am not a doctor in fluid dynamics, nor hold a mathematics degree from UCL.

I found that another way of testing the probability of landing on each and every space on a Property Tycoon board is to do it not mathematically but physically. Not in the sense of actually sitting and rolling two dice, followed by moving a counter around a board game recording the number of times I land on each tile, as that would take an eternity. By creating a simple program that can simulate a game of Property Tycoon and allow a single player to roll two dice and record the number of times it lands on each and every tile. To ensure the results are reliable, a number of games of property tycoon can be played, as the more data you collect, the more likely it will be accurate. To ensure the probabilities match, all the 'wildcards' must also be encoded into the game, such as the pot luck, opportunity knocks, Go to Jail square and the infamous rolling of doubles three times in a row lands the player in the jail house.

Luckily during my A-Levels, I had constructed a similar prototype in Python, to test if a player would land on every tile equally if x amount of games where played, i.e. each square would have an approximate probability of 1/40 (2.5 %) after simulating 100 games of 100 rolls. Using my knowledge of Property Tycoon, and the Internet, (standupmaths, 2016)

.I was able to produce a program that could do this effectively. This program can be found in the Research & Development folder of the project, and can easily be ran on a python online compiler, such as https://www.programiz.com/python-programming/online-compiler/ .One thing to note is that the program only calculates the chance of landing on a property, similar to Figure 3, and will not use any form of 'money' to end the game early. As Property Tycoon works slightly different to a standard game of Monopoly, for example different cards in Community Chest & Chance compared to Pot Luck & Opportunity Knocks, I will first design a program called Monopoly Sim, which will be compared Dr Fry's tests, and then adapted to suit Property Tycoon.

After running the Monopoly simulator the results displayed are shown in Figure 7. Each element in the array is a tile in the game board that goes around in the traditional way. The parameters used for this simulation where 100,000 games of 100 turns, meaning there was 10 million individual turns. The program can take a bit of time to execute so you can adjust the parameters in the method definition to speed it up.

```
Shell
[279908, 204964, 188173, 224835, 242290, 280571, 239233, 106764, 239635, 234672, 634655, 277608, 264570,
    241794, 251185, 284196, 282096, 264499, 294746, 310424, 288277, 283589, 121558, 272248, 319299, 287700,
    270697, 267226, 280409, 256596, 0, 264373, 257383, 110176, 244097, 251044, 218945, 204972, 203623,
    250970]
>>>
```

Figure 7 – Shell output from the python code for monopoly sim

What do these values mean? Well, if there were 100,000 games of monopoly played each constructed on 100 turns, the Go tile, which is the first position on the game board, and the first item within the array was landed on 279,908 times. As there 10 million individual turns, the probability of landing on the Go tile is 279908/10,000,000 = 0.028 (3.dp) or 2.8% chance. By calculating the chance for each tile on the board and plotting the data in the same way Figure 3, we get Figure 8. One thing to immediately note, Jail and just visiting have been listed as the same tile. All the data can be found along with a larger graph in the excel document named Excel Monopoly Graph in the report folder.

By comparing the two side by side, I can safely say with my full confidence, that all results previously shown are accurate to my knowledge. Comparisons of both Trafalgar Squares being the most

common landed on *property* and Park Lane being the least landed on *property.* In addition, by adding Dr Fry's Jail/Visiting percentages, they do equal approximately 6.2% which my graph does convey.

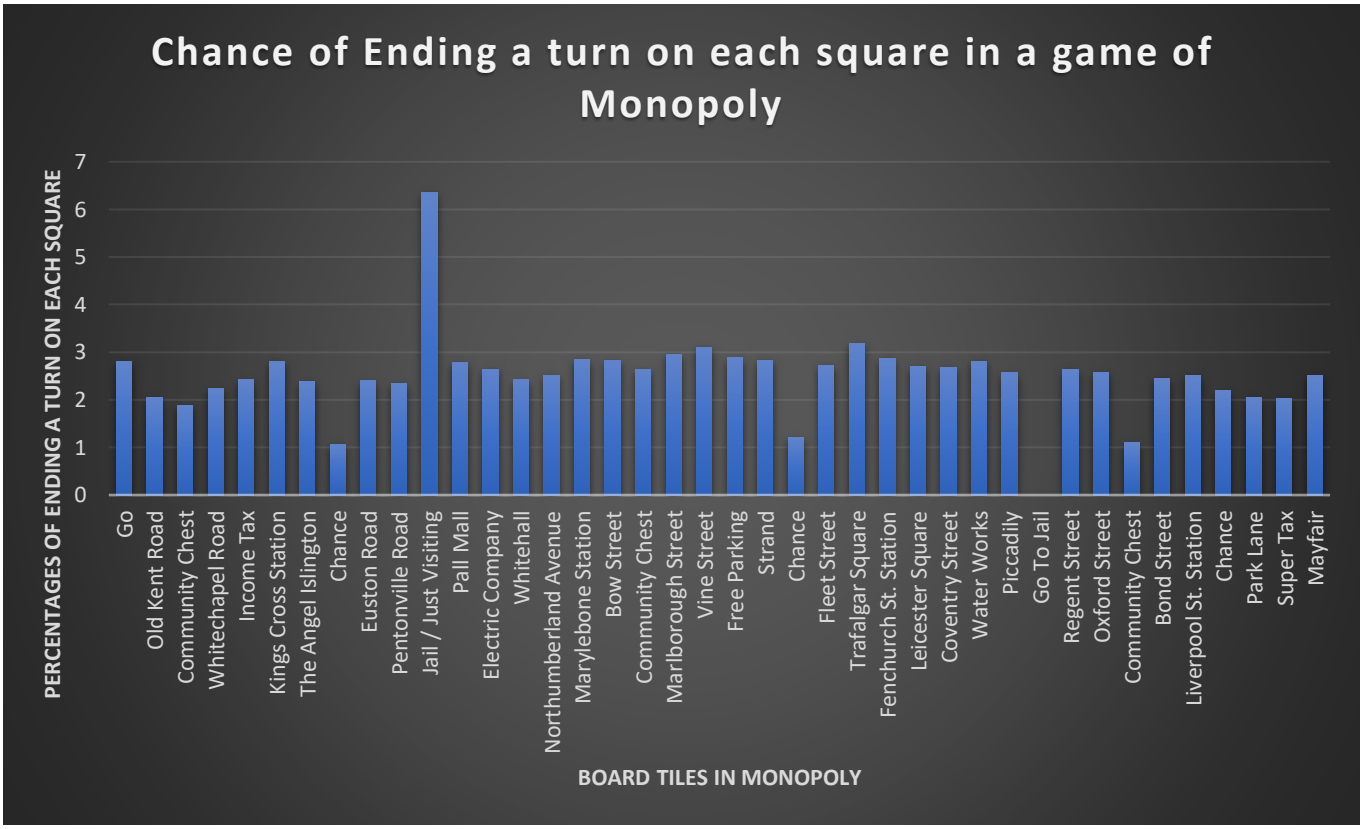

Figure 8 – Graph created from the Python code

Now to code the same simulator for Property Tycoon. The only real difference is the Pot Luck / Opportunity Knocks which do include move to certain locations, but do not include move to nearest station / utilities. The updated code for Property Tycoon is also available within the folder. The code will act the same doing the same amount of simulations, just with the version specific card decks. The code produces the shell output shown in Figure 9. On first glance, data looks similar, but will have to wait to see data displayed in graph like above. All the data can be found along with a larger graph in the excel document named Excel Property Tycoon Graph in the report folder.

Shell

```
[282399, 238593, 175114, 225548, 244538, 234183, 241511, 138735, 241560, 235471, 635204, 275514, 230665,
    240788, 250321, 316633, 281402, 243438, 294597, 307644, 285457, 281890, 156338, 270745, 318787, 271832,
    271158, 268701, 263926, 260144, 0, 264675, 256884, 139922, 242449, 232772, 218884, 204713, 204572,
    252293]>>>
```

Figure 9 – Shell output from python code for Property Tycoon Sim

Below in Figure 10, we can see that the graphs produced do show very similar qualities. Differences seen immediately is that the firs property after Go in the Property Tycoon, Crapper street was landed on much more. This could be because there now is a Pot Luck Card that sends the player to end their turn there. In addition all stations other than Hove Station, which is visited more, are much less likely to be landed on in property tycoon. This is because the Opportunity Knocks card sends the player to that exact station rather than the closest station. The removal of the move to the nearest

utility card has also caused a decrease in the Electric Company equivalent in Property Tycoon (Tesla Power Co), but the Water Works (Edison Water) weirdly was visited more in Property Tycoon. The data for this can be seen below the table within the Excel Property Tycoon Graph document. As the majority of the other property tiles are unaffected, the previous results displayed in Figures 4 & 5 should remain the same, other than for Utilities and Stations which should be worst off. But as said before, avoid buying these as they are useless.

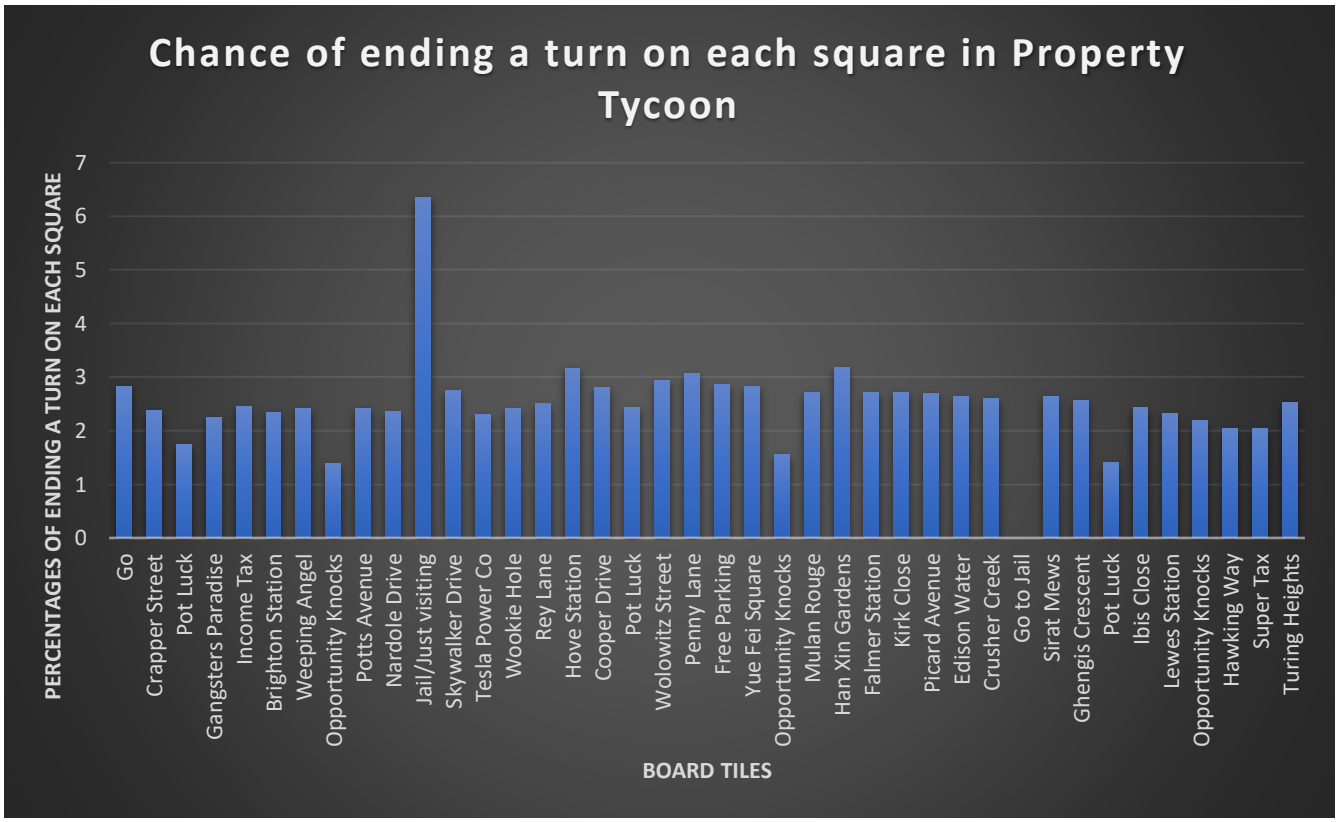

Figure 10 – Graph created from python Property Tycoon code.

Prototyping my Own AI

Using my own results along with Figure 5's data on the best colour groups I have determined the ranking order that the AI will use to influence the AIs strategy towards to higher tiered sets. The lists from **least** to most valuable goes:

- Station
- Utilities
- Brown – possible early game set
- Light Blue – another early game
- Purple
- Deep Blue – Not worth it
- Orange – Very good early game
- Red
- Yellow – Brilliant mid/late game
- Green

This list will be used within the prototype code to influence the AI towards certain properties. Other factors such as current colour group should also affect the choice of the AI. Further additions such as early game sets should be a higher priority at the start of the game compared to 50 turns in. The code will be developed in Java, as it is the most suitable OO language that I have experience in, and can easily be translated into C# by someone who understands the language a lot better than me.

The AI simulation should simulate the first few moves, as the player is unable to buy properties (which is all we care about). To further speed up my implementation, I will only be testing on a board with purchasable properties, other than the initial tile which I've made off the board. This prototype just needs to make a decision.

The first iteration of the prototype will only include 2 players, both being 'AI' which have to make decisions on their own. They will buy properties at complete random. Hopefully the following iterations add more functionality, such as being able to rank their assets from least to most valuable, so they only sell the lower valued ones if money is needed. I created a class diagram, Figure 11, for the Java Prototype to help me plan out what methods are needed, and what kind of variables about the game and players need to be stored, which are used to help the player make more controlled and thought out decisions later on in the development. An example of this could be the player can count how many rolls have been completed, so they can use this to determine a late game situation compared to an early game situation where they would end up investing in completely different properties. A PNG and PDF version of the class diagram are further available in the research folder for a better resolution look.

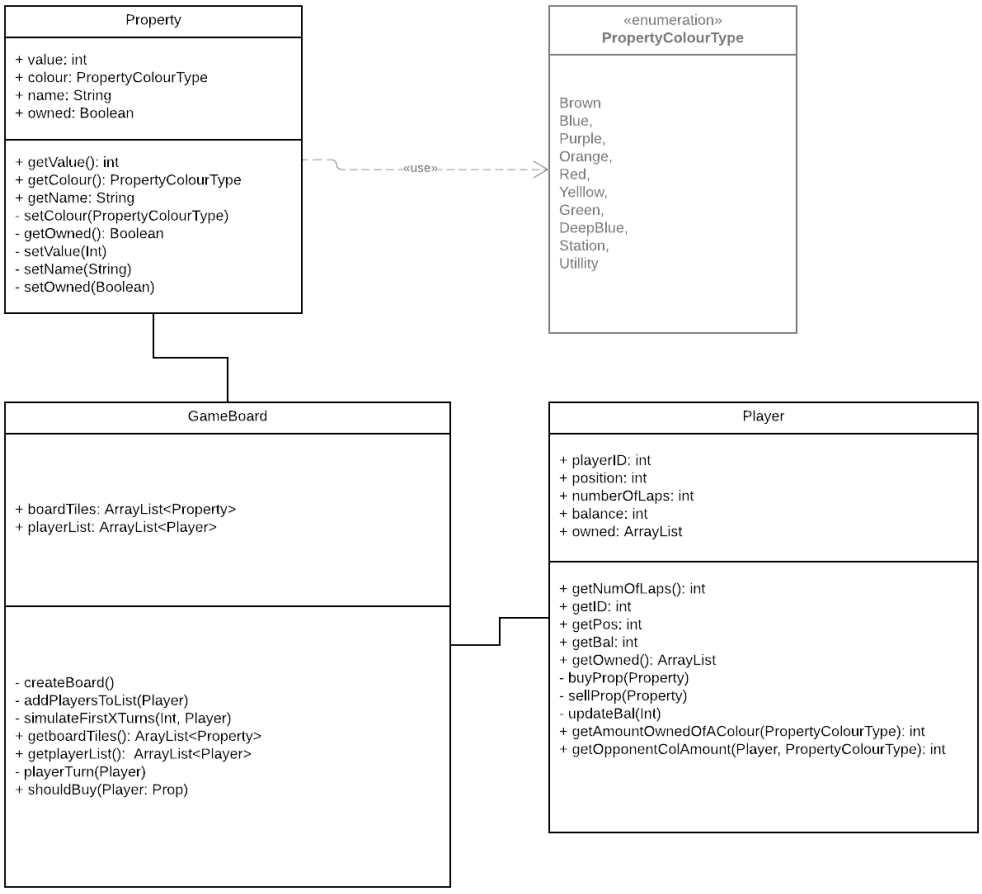

Figure 11 – Class diagram for Prototype

At the time of writing the report, the Random AI prototype is able to use a selection of variables to sway its odds on buying a property. However none of the variables affecting the buy chance have been controlled as all generated randomly. This means they will all add up and as long as they are greater than or equal to, another random number between 0 and 1, the property will be bought. This code is available as a NetBeans project within the research folder. By running the main method in the game board class (Board) the first few moves are simulated (as a player cannot buy until lapped past go at least once) to speed up the testing. In the actual code, the more factors / variables that go in favour of buying the property, the more likely the AI will choose to buy it. For example the final green on the board and the player currently owns the other two green properties, the buy chance would most likely be close to 1.00 if not 1.00. A 1.00 chance of buying (called buyChance in the code) indicates that the player will 100% purchase the property as long as it has enough liquidity available. In further versions of the development, the AI could have a method that even offers properties & equity to others players / the bank in order to gain enough liquidity to buy the desired property, as long as it lands on it. I have further added another Netbeans project with a 'smarter' AI which generates a value of a buyChance depending on how valuable the property is, using the ranking system at the start of the section. This is slightly more what the actual AI would do. For example if the property is a station the code "buyChance += (Math.random() * (0.1 - 0.01) + 0.01);", generates a buyChance value of at most 0.1 plus any further additional variables. While if the property is of colour Green, the code "buyChance += (Math.random() * (1 - 0.7) + 0.7);", generates a buy chance of at least 0.7 plus any other variables. This would convey green tiles are very valuable and stations are not, which is true in the adopted strategy, and the data supporting it.

Conclusion to research

I believe that the research conducted in this report convey a good understanding of the type of strategy I was attempting to implement within the AI. Not finishing the prototype is a big shame but due to given circumstances, and having no access to a computer for extended period of time, I still believe the research and project are still well produced .If given more time, in a good working environment, I believe I could make good progress in determining more exact values each factor of the game would do to increase/decrease the likelihood of the AI Player to wanting to buy the given property.

Bibliography

Aradhya, A. L. (2017). *Minimax Algorithm in Game Theory | Set 1 (Introduction)*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

Aradhya, A. L. (2017). *Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

gmdirect. (2018, June 26). *Winning Monopoly With Math!* Retrieved from Puzzle Nation: https://blog.puzzlenation.com/2018/06/26/winning-monopoly-with-math/

Klein, M. (2017, December 6). *Google's AlphaZero Destroys Stockfish In 100-Game Match*. Retrieved from Chess.com: https://www.chess.com/news/view/google-s-alphazero-destroys-stockfish-in-100-game-match

Retail Game Source Code. (2018, October 23). *Monopoly Source Code.* Retrieved from GitHub: https://github.com/RetailGameSourceCode/Monopoly/tree/master/Source

Rin. (2018, March 8). *How Stockfish Works: An Evaluation of the Databases Behind the Top Open-Source Chess Engine*. Retrieved from Good Fibrations: http://rin.io/chess-engine/

Somers, J. (2018, December 28). *How the Artificial-Intelligence Program AlphaZero Mastered Its Games*. Retrieved from The New Yorker: https://www.newyorker.com/science/elements/how-the-artificial-intelligence-program-alphazero-mastered-its-games

standupmaths. (2016, December 8). *The Mathematics of Winning Monopoly.* Retrieved from Youtube: https://www.youtube.com/watch?v=ubQXz5RBBtU&list=LLKYe9oj9Lhed-UQeGgqOE3A&index=85&t=11s

The University of Auclkand, New Zealand. (2015). *Chapter 8: Markov Chains.* Retrieved from https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf