ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems

Project Report

Acquire:
Simulation of an investment game using Python

Philipp Binkert, Dennis Kunz

Zurich
Dec 2018

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of COSS. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Dennis Kunz

Philipp Binkert

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten Semester-, Bachelor- und Master-Arbeit oder anderen Abschlussarbeit (auch der jeweils elektronischen Version).

Die Dozentinnen und Dozenten können auch für andere bei ihnen verfasste schriftliche Arbeiten eine Eigenständigkeitserklärung verlangen.

Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuer und Betreuerinnen der Arbeit.

Titel der Arbeit (in Druckschrift):

Acquire: Simulation of an investment game using Python

Verfasst von (in Druckschrift):
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Binkert	Philipp
Kunz	Dennis

Ich bestätige mit meiner Unterschrift:

- Ich habe keine im Merkblatt „Zitier-Knigge" beschriebene Form des Plagiats begangen.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu dokumentiert.
- Ich habe keine Daten manipuliert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Winterthur, den 9.12.18	P. Binkert
	D. Kunz

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

Contents

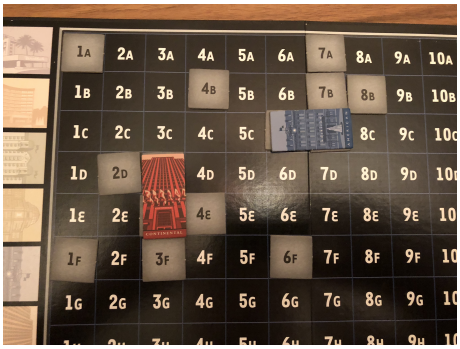
1 Abstract

The subject of this paper is the implementation and analysis of the investment game Acquire. We examine how different strategies affect the outcome of a game and determine the most successful strategy by creating an adapting player that finds the best way to win.
We discovered that a risky player is more successful than a conservative one and that investing in small hotels is more profitable than investing in large hotels. A player who favors creating hotels and investing in these fares better than all of the above players but loses against the adapting player. The strategy of the latter, therefore the most successful, is a combination of trying to take over companies, not saving too much money and investing in companies he created himself.

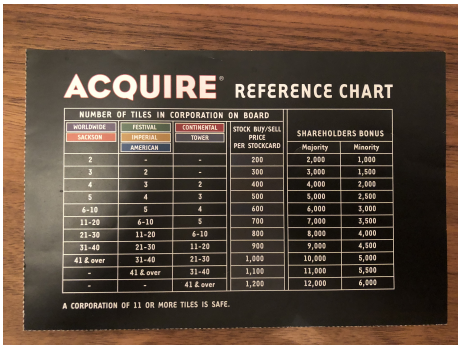
2 Introduction

The stock market is a highly complex and dynamic system that can hardly be predicted or described by a mathematical model. We decided to take a step back and figure out if there is a more simple way to investigate individual aspects of the stock market. We chose to implement the board game Acquire since it is an investment game that includes several aspects that are very important when trying to examine the stock market. The game is played by four players creating, enlarging and merging hotels as well as trading stock. This led to the idea of comparing different strategies and finding the most successful one.

One of our main objectives was to determine whether or not a conservative strategy is more successful than an offensive strategy. Furthermore we wanted to compare a player that invests in small hotels with a player that prefers buying stock of large hotels. Our prediction is that the offensive player should score more victories than the conservative one, but on the other hand also have more last places, and that players investing is smaller hotel chains are more successful that players investing in larger ones. The last goal was to find out the most successful strategy by creating a player that adapts it's strategy himself in order to win as many games as possible.



(a) Acquire game board



(b) Chart with stock prices

3 Model

3.1 Overview

As described in the introduction, we take a somewhat unusual approach to model the stock market in that we do not base it on a proper mathematical model, but rather on a board game which is designed to simulate different hotel chains struggling for survival in a fictive market, with the players acting as stock brokers buying and selling stock and trying to make their fortune. The meaningfulness of our studies is therefore closely tied to the extent of which this game actually represents the actual stock market business; this will be topic in chapter five. In this section we will first give a short overview of the rules of the game and then discuss how we can use it as a foundation upon which we can pursue our questions and test our hypotheses.

3.2 Rules of the game[1]

Acquire is a four-player game. Each player starts off with a capital of 6000$. The fictive city is represented by a 9x12 grid which consists of 108 at first empty spaces, with rows being labeled by letters A-I and columns by numbers 1-12 (Figure a). There is one tile matching each space; the tiles are randomly chosen by the players throughout the game and placed on their according space. Once two or more adjacent spaces are occupied, a hotel chain is created and the player that founded the hotel chain is rewarded with a stock from that hotel chain, as he is the founder and thus at the moment the owner. There are maximum 7 hotel chains that can exist on the market at any given time, and they are divided into three different price-classes; low-cost, medium and high-end.
Each hotel chain has a maximum of 25 stocks which are for public sale to the players. The price of a stock given out by the company is based on two factors;

the size of the company, i.e. the numbers of spaces which belong to it's chain, as well as it's price class. The prices are given by the table depicted in Figure (b). On each turn, a player can place one of his tiles on the board (he always has 6 spaces to chose from), and then buy up to three stocks of any existing company if he wishes to do so. Players are free to trade stock between each other as they wish.
The core of the game happens when a hotel is built such that it "merges" two existing hotel chains, or concretely when a tile is placed so that to existing blocks connect. When this happens, the larger hotel chain swallows the smaller one. The majority and minority stakeholders of the smaller hotel chain, the players with the most and second-most stock, are paid out cash bonuses and every player at this point must decide what to do with his stock in the smaller company. They can either trade their stock into stock of the acquiring company at a ratio of 2:1, sell their stock at the present market price, or hold onto it in the hopes of later re-founding the company.
The game ends when either one company is considered to dominate the market (this happens it reaches a size of 41 tiles), or when there are three or more companies which have firmly established themselves (this happens when it reaches a size of 11 tiles; from this point onward, it can no longer be swallowed by a bigger company). Cash payouts are then made to the majority and minority stakeholders of each surviving company and the winner is the player who earned the most money.

3.3 Outline

Our goal for the project is to in a first step create a simulation of the game in which four players compete against each other. We will be using Python as our programming language. We want to be able to know exactly what each player is doing at every time in order to in the beginning make sure all the rules are being correctly followed, and later on to be able to try to understand the decisions the players are making. In the second phase we will program players of different playing styles to model the different approaches one can have to the stock market. The success of each "strategy" will then be determined by letting the simulation run a large number of times (on the order of 1000) and evaluating which are more successful on the long run.

4 Implementation

In this section we describe the implementation of the game as well as the strategies. Since our code is not based on an other project we want to focus on the implementation of players and especially on how decisions are made. A description of the outputs and functionality of the different programs can be seen in the appendix.
First we will explain the general setup, the implementation of the board and hotels. Then we will describe the implementation the players, how decisions are made and how to change the strategy of a player. In the end we explain how we created an adapting player to find the best winning-strategy.

4.1 General setup

In a first step, we implemented the board. This was straight forward since the board naturally can be represented as a matrix, where each entry corresponds to a tile. We chose the values of an entry to be 0 if no tile is placed on it, 1 if a tile is placed on it that does not belong to any hotel. The values 2,3,4,5,6,7,8 correspond to the seven different hotels. This makes it easy to get the information we need to evaluate a tile placement.

The hotels are implemented via the class *Hotel*. The reason for using a class is that all hotels have the same attributes as well as functions to modify them. A class makes it easy to initialize the hotels as well as interacting with them, e.g. buying stock or enlarging it. As mentioned above, each hotel has a corresponding number to identify it on the board which is one of the attributes a hotel. Others are the size, number of stock available, price group and name.

Two very important functions for the game to work are *merge*() and *placetile*(). The *merge*() function processes the merging of two hotels. First it determines the smaller hotel and the majority and minority stakeholder in it. The bonuses are then payed and all players are asked to decide on what they want to do with their stock of the smaller hotel. In a final step the size of the big hotel is modified and the value of tiles that belonged to the small hotel are changed to the value of the big hotel.

The function *placetile*() is called when a tile is placed on the board. First all the information on the tile are collected including the number of hotels surrounding the tile. If this number is 0, the tile is placed and nothing else happens. If the number is 1, the hotel next to the tile is enlarged. For a value of 2, 3 or 4 a merge of two, three or four hotels occurs.

4.2 Players

In a first step the goal was to implement a player that makes decisions based on our experience in the game. We call this the *normal* player. Since we also wanted to implement a few different players with different strategies, it was very important that the strategy of the player can be adjusted easily. We achieved this by creating the class *Player*. The attributes are how much stock the player owns of each hotel, as well as the money, name and tiles of the player. We also included several arrays and matrices as attributes that describe the strategy of the player, but more about this later.

4.2.1 Normal player

In this paragraph we describe the main functions of the class *Player_normal*(). The function *decide_merge*() makes the decision on which of the two given hotels is acquired. It is only called if two hotels have the same size and the decision is based on the amount of stock and money the player has and whether the player is majority or minority stakeholder. The function is the same for all the players we implemented since this decision has little to do with strategy but with profit. The function *decide_placetile*() is a key function of our program. It decides which tile a player places on the board. For each legal tile the function finds out what tiles are next to it on the board. To decide which tile should be placed, we introduced a system that gives points to each tile. At the end the tile with most points is placed on the board. This allows us to control the decision by modifying the number of points given. We store these points in an array as attribute of the player:

```
self.A7=np.array([20,2000,10,3000,8,12,15,13,4,2])
```

The first entry are the points given if the tile causes a merge where the player is majority stakeholder in the small hotel, so he would get the majority bonus. The entries 2-4 are used if the player is minority stakeholder in the acquired hotel. If the player has less than 2000$ or 3000$ the tile is awarded 10 or 8 points. The fifth entry stands for the case that the player is majority stakeholder in the bigger hotel and has enough stock of the smaller hotel to stay majority stakeholder. The sixth entry stands for the case that the player would become majority stakeholder in the bigger hotel. The last three entries award points if no merge occurs but a hotel would be created or a hotel where he's either majority or minority stakeholder would simply be enlarged.
The normal player will always place a tile that would reward him a majority bonus, which is reasonable since the bonus usually a lot of money. If he's not majority he generally would not place the tile and rather create a new hotel in which he would automatically become majority stakeholder.

buy_stock() is another very important function that each turn decides whether or not to buy stock and if yes, from which hotel and how much. The function works similarly to *decide_placetile*(). It basically awards points to each stock and at the end decides which to buy as well as how much. For every hotel, the function evaluates how much stock the player and all opponents possess and how much stock can still be bought. If the opponents don't own any stock the following array is used to award points:

```
self.A10 = np.array([10,9,8,5])
```

In this case 10 points are awarded if the player owns 1 stock, or 9,8,5 if he owns 2,3 or more than three, respectively.
In case the player is majority stakeholder of the hotel, a function is called to compute the points using the following matrix:

```
self.A1= np.array([[0,  0,  0,  0,  0,  0,  0,  4],
                   [0,  0,  0,  0,  0,  0,  50,  6],
                   [0,  0,  0,  0,  0,  50,  50,8],
                   [0,  0,  0,  0,50,50,100,10],
                   [0,  0,0,50,50,100,100,12],
                   [0,0,50,50,100,100,22,14],
                   [0,50,50,100,100,24,24,16],
                   [0,50,100,100,24,24,15,18]])
```

The row of this matrix corresponds to the lead in stock beginning with the row at the bottom. So if the player has no lead, the last row is observed and if the lead is bigger than 6, the row at the top is observed. The columns correspond to the number of stock available. The first column is considered if 0 stock are available, so all entries are 0 as well. The last column corresponds to more than 6 available stock. A lot of the entries are 0 because if for example the lead of the player is 4 and fewer than 4 stock are available, he will always stay majority stakeholder in this hotel so there's no need to buy further stock. The entries with value 50 stand for buying one stock for sure, which is reasonable when the player can secure the majority bonus by buying one stock, e.g. if the lead is 2 and 2 stock are available. 100 points are given if the purchase of two stock secures the majority bonus. Otherwise 24 is the highest number and from there the points awarded decline proportional to the lead and number of stock available.
If the player is not majority stakeholder but the difference to that player is not big, points are awarded according to the following matrix:

```
self.A2=np.array([[0,20,100,100,100,20,12,16],
                  [0,  0,  20,  24,  24,  12,8,13],
                  [0,  0,  0,  20,  12,  10,  8,10]])
```

The idea of this matrix is the same as for matrix $A1$. The top, middle and lower

row correspond to being 1,2 or three stocks behind the majority stakeholder. The columns from left to right correspond to the number of stocks still available, from 0 to more than 6. As in matrix $A1$ some entries are zero because it makes no sense for the player to buy this stock and the value 100 makes the player buy two stock. In case the player has very few stock compared to the majority stakeholder, the points are awarded using this array:

```
self.A3=np.array([0.5,0.5,2,1])
```

If the player is minority stakeholder, the points are calculated with matrix $A1$ and then multiplied by $A3[0]$. If the player is three or less stocks behind the minority stakeholder, the points are determined using matrix $A2$ and multiplying these by $A3[1]$. Otherwise 2 or 1 points are given for a deficit of 5, or more than 5.

4.2.2 Strategies

Using the matrices and arrays described above, we were able to manipulate the decisions of the player easily and effectively. Changing the values of the matrices leads to a different distribution of points and therefore to a different decision of the player.
As mentioned above, we wanted the normal player as a reference, making decisions that seem best to us. The idea was that the player should have a balanced strategy. He should keep track of his money, always having more money than a certain value, but he should not save too much money by not buying stock that may be valuable later on. Furthermore the normal player, in case he is majority stakeholder, should buy stock if his lead decreases. He should try to become majority stakeholder if his deficit is not too large but otherwise not buy too much stock.
Next we wanted to compare an offensive, aggressive player with a defensive, conservative player. By offensive and aggressive we mean that the player spends more money and is not interested in buying stock of a hotel in which he already is majority stakeholder but rather tries to become majority stakeholder in an other hotel. We implemented this strategy by scaling down the values of matrix $A1$ compared to matrix $A2$. This makes the player prefer stock that correspond to the matrix $A2$ which in turn makes him buy stock of a hotel in which he's not majority stakeholder. We can control how much money is spent by adjusting an affine function of money, with slope $A6[0]$ and displacement $A6[1]$, where $A6$ is an array we added as attribute of the player. Stock is only bought if the points awarded to this stock are higher than the value of that function.
For the conservative player we adjusted the values the other way around, so we scaled down the values of matrix $A2$ compared to matrix $A1$ and changed the function described above.

Next we implemented a player that prefers small hotels and one that focuses on large hotels but otherwise make the same decisions as the normal player. Getting the players to prefer small or large hotels required us to add an other array as attribute and award more points to stock of small or large hotels considering that array.

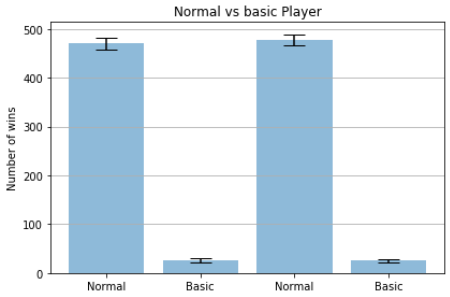
In a last step we implemented a player, named the "entrepreneur", that tries to create new hotels whenever he can and prefers to invest in these. We accomplished this by doubling the values of array $A10$ and increasing the value of $A7[7]$, which is used to award points to a tile that would create a hotel.

4.2.3 Adapting player

The idea of an adapting player is that some values of the matrices and arrays described above are variable and that the player adapts these values himself to win as many games as possible.
We treated the parameters as independent and therefore adapt each parameter individually to achieve the best possible result. The adaption of a variable is performed by the function $converge()$. In a first step it evaluates whether the value of a parameter should be increased or decreased. Then it changes the variables according to that evaluation using an initial step and plays 1000 games with these parameters against another player. In a next step it compares the number of wins before and after the adjustment. If the number of wins increased, the parameter is further scaled up/down and if not the scaling is inverted and the step is halved. After playing an other 1000 games, the number of wins are compared and the parameters are changed once more. This process is repeated until either the difference in wins is below a certain value or the parameters were changed 15 times. In the first case the variable converges but in case the parameter was adjusted 15 times, no progress was made so we interrupt the adaption of that parameter.

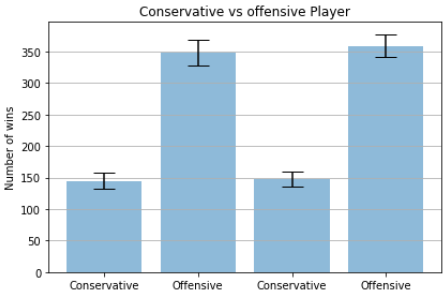
5 Results

To acquire our results, we let our players describing different investing styles play 1000 games against each other. Unless denoted otherwise, there are always two players of the same type playing against two players of a different type. The number of wins by each player is then saved, and the process is repeated 15 times in order to obtain mean and standard deviation (std. dev.) values. We will present the results of our simulations here and then discuss them in the next chapter.

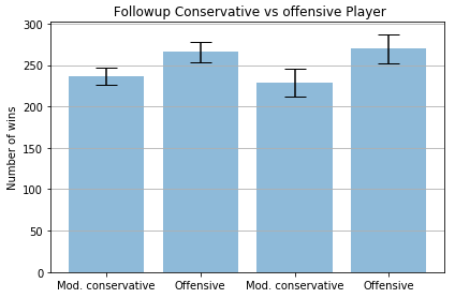


(c) Normal vs basic player

The very first comparison we make is between the normal player adapting a balanced strategy and a basic player, who follows no particular strategy and more or less buys stock and places his tiles randomly. Naturally, we expect the normal player to win this comparison by a very large margin, as this is mainly designed to be a test to make sure our normal player plays smartly and our implementation works properly. The outcome of the test is a mean value of 474 wins for each normal player (std. dev. 11) and 26 for the basic player (std. dev. 3)



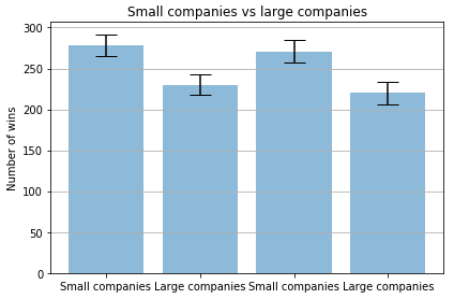
(d) Conservative vs offensive player



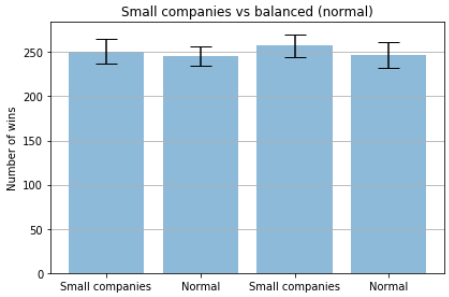
(e) Modified conservative vs offensive player

After this it is time for the first real evaluation; we let the conservative player play against the offensive one. The conservative players scores a mean value of 146 wins ($\pm$12), the offensive players 354 wins ($\pm$19). The outcome of this matchup leads us to conduct an unplanned follow-up experiment, as we will explain in the next the section. It consists of a modified conservative player against the same offensive player as before, with the modified conservative player now scoring 232 wins on average ($\pm$13) and the offensive player 268($\pm$14).

The next difference to model is buying preferably small, emerging hotel chains or large, established ones. The players preferring small hotels fared better with 275

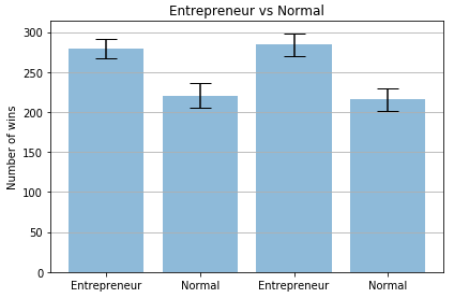


(f) Players focusing on small companies vs players focusing on large companies

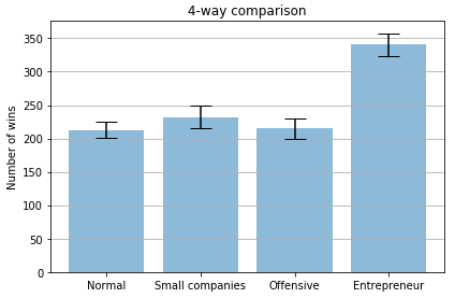


(g) Players focusing on small companies vs players will a balanced strategy

wins(±14) against 225 wins(±13). Here too we decided to perform a follow-up experiment, comparing the more successful small hotel player against the normal player. This resulted in 254 wins(±14) for the small hotel player and 246(±13) for the normal player.



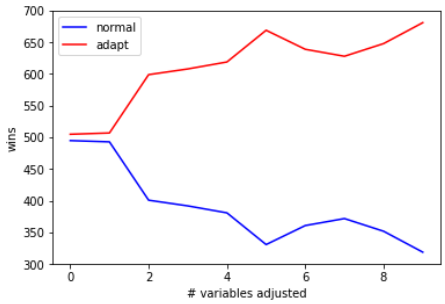
(h) Entrepreneur versus normal player



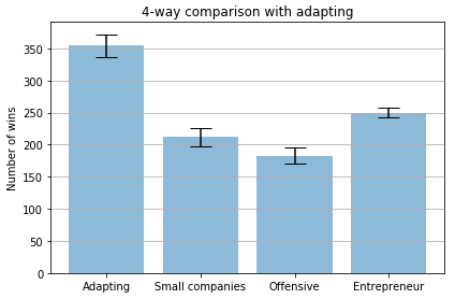
(i) Comparing four different strategies

The final strategy we examine is a player who heavily favors hotels he founded himself ("entrepreneur"). We first let this player go against the normal player, whom he defeated with a mean of 282 victories(±13) to 218(±14). Finally we also compared the four most successful players, the normal, small hotel, offensive and entrepreneur player. The entrepreneur came out on top with 341 wins (±17), in front of the small hotel buyer with 232(±17), the offensive player with 215(±15) and finally the normal player with 213(±12).

After testing the different investing styles, we now test our self-optimizing player. He starts off with the same parameters as the normal player and independently adjusts them; he plays a round of 1000 games against the unchanged normal player after every adjustment. Figure(j) shows how this matchup evolves as a function of the number of variables adjusted. The optimized variables are then



(j) The player with adapting parameters versus the normal player with constant parameters



(k) 4-way comparison including the optimized player

saved and a player using these is pitted against the three most successful players we had implemented by ourselves, the small companies, offensive and entrepreneur players. As shown in figure (k), the result is 354($\pm$18) wins for the adapted player, 212 for the small companies player ($\pm$14), 183($\pm$13) for the offensive and 250($\pm$7) for the entrepreneur.

6 Discussion

6.1 Model accuracy

Before discussing the outcomes obtained from the simulations, we will first try to asses to what extent we can use our model to make statements about the real stock market. The difficulty here is that we have no data to which to compare our results. Nevertheless, we believe there are some key points that do at least suggest parallels from the model to the actual stock market:

- The general basic principles of the game do reflect, abstractly but none the less, real world economic behaviour; Companies are founded, will grow, and sometimes be swallowed by bigger companies. Their stock is publicly traded and it's value will generally increase as the company gets larger.
- If a small company is bought by another one, it's stock holders are generally able to also invest into the acquiring company at beneficial conditions and the major shareholders will usually profit additionally; this concept is well implemented by the majority and minority stakeholder cash bonuses in Acquire.
- Due to the fact that players are able to freely trade stock among themselves and the limited amount of stock available, the principle of supply and demand

is nicely incorporated; as soon as stock of a certain company become sparse, players will be willing to pay more for a single stock, even if another company of equal size and value is around with still plenty of stock.

- Many strategies that can be employed by an Acquire player, and were implemented by us, make logical sense and apply to actual stock brokers as well. A good example of this is the offensive versus conservative comparison. The conservative stock broker will make sure he has a balanced portfolio with lots of different companies and always some cash on reserve, while his more aggressive counterpart will be willing to put most of his money on one horse and not holding much back.
- To be successful, it is necessary to have a certain intuition for how the market will evolve in the future. In Acquire as in the real business it makes sense to get invested in a company of which you believe it will prosper early on when it is still small and it's stock correspondingly cheap. The real-world version of this is perhaps to foresee major trends or to recognize when a company has found a certain niche; the Acquire equivalent of such a company would be one in an area of the board with only few tiles placed and no other hotel chains nearby.

There are, however, also some major differences between the game and the real market that we must keep in mind:

- In Acquire, stock can only be sold to "the bank", e.g. to someone other than another player, during merges of companies or at the end of the game; this is of course nonsense; in the real world you can generally sell your stock at any time (although you might have to take dropping prices into account if you own a sufficiently large portion of all stock).
- In the same category, you can only buy up to three stock at a time in Acquire. This is necessary so that one player cannot just buy all the available stock of a company that just experienced a fortunate turn of events without the other players being able to react. For this reason, we consider this not to be such a limiting factor, as it just helps to smoothen the turn-based nature of the game which isn't realistic.

Because of these limitations we confine ourselves to examining only certain aspects and individual factors, which we will do in the following section.

6.2 Result evaluation

6.2.1 Conservative vs offensive strategy

The first comparison we made was between a conservative and offensive player, as described in section 4.2.2. Our hypothesis that the more risky strategy would pay off in the long run was definitely confirmed, as the big difference in number of wins, shown in figure (d), proves. Somewhat surprised by the magnitude of the victory, we got curious as to which property of the offensive player made him so superior to the conservative one. The answer was found when we adjusted the conservative's array $A6$, which governs the amount of money the player strongly dislikes going below, to the values of the offensive one. This one change reduced the conservative players deficit of around 208 wins per 1000 games per player to only around 36, which can be seen in figure (e). We therefore conclude that willingness to risk more of your capital corresponds strongly to a better outcome.
We had also expected that while the offensive player gets more wins because of his more risky strategy, this would also result in more *last* places as well. This did not turn out to be true; on contrary, the offensive player even scored significantly fewer fourth places than the conservative one. We interpret this result as follows: the mechanisms of the game represent a very "bullish" situation at the stock market, in a sense that the hotel chains will generally grow or be swallowed by a bigger company as time progresses. For the investors, the players, both of this is actually good news, as the value of your stock will either grow or you have the possibility of getting paid out during a merge. In such an environment it makes sense that it is better to invest more of your money to maximize your profits, as the risk of going bankrupt is small.
We think this is an example of a where our model does a good job of depicting reality. In economically healthy times, you are statistically better off if you are willing to invest a greater proportion of your capital.

6.2.2 Buying large hotels vs buying small hotels

The next topic we wanted to analyze was whether it makes a difference if a player is inclined to buy small hotels as opposed to one going for larger, established ones. As predicted, the players targeting the smaller ones fared better, scoring around 50 wins more per 1000 games per player as shown in figure (f). What we found interesting was that the player buying small hotels won almost exactly the same amount of games as the one with the balanced strategy, as can be seen in figure (g).
The reason the small hotel buyer fares better is because the game does not appropriately reflect the chance of a small hotel chain going downright bankrupt;

while it can definitely stay small and weak for the whole game, it cannot actually go bankrupt and cause all it's stock to become completely worthless. This risk obviously would need to be accounted for in real life stock trading.

6.2.3 The entrepreneur

The last style of investing we wanted to take a look at was that of an player investing chiefly in the hotel chains he created himself and not getting too involved in other companies (labeled the "entrepreneur"). The advantage of this philosophy is that as the founder of a hotel chain you immediately start out as the majority stakeholder, and it is then easier to try to defend your position than to outbuy an opponent in other companies, even though they may be more successful. As the results shown in figure (h) and (i), this strategy was the most successful, as it clearly defeated the normal player in head-to-head and also stuck out when we compared the most successful ones: normal, small-hotel buying, offensive and entrepreneur.

6.3 The adapted-strategy player

In this last segment we will discuss the results of the strategy-optimizing player, who continuously improves his parameters as described in chapter 4.3. We had a few missteps with this program at first, which we later found out were due to bad initial values, but in the end we managed to implement this player successfully, as figure(j) proves; In the beginning, the adapting player is identical to the normal player and therefore there is no significant difference in the number of wins between them. Then, as more and more variables are improved, the adapting player gets stronger and stronger, until at the end he eventually crushes the normal player by a huge margin of about 150 wins per 1000 games per player. Plugging the hereby acquired variables for the strategy matrices into the other program and comparing it with the entrepreneur, our strongest player thus far, yields that he too is without the slightest chance against this optimized player, exactly the result we were hoping for.
There are two small drawbacks to the adapted player program worth mentioning at this point. The first is that sensible results require for a large amount of games to be played (around 50'000), this results in quite some computing time, about 25 minutes with our laptops. Secondly, we would have liked the variables obtained by the algorithm to match if the program is executed multiple times, as the they should converge towards an optimum. While the curve will always look similar to the one shown in figure (j) and the optimized player will always beat his starting self, there are some small differences in the final output variables. Removal of this unwanted effect would probably require even more games to be played in

order to decrease the probability of a 1000 game matchup by chance ending close enough so that the converging condition is fulfilled. This would result in even more computing time however, so we deemed these results accurate enough and passed on this possibility.

7 Conclusion

As this was our first bigger project that involved programming, we ran into lots of complications along the way, but during the process it was nice that we were able to apply the knowledge we acquired during the assessment year and the course. Looking back on the project, we were successful in creating a simulation of the board game Acquire that functions properly and allows for diverse flexible strategies to be employed and compared. The results we got were statistically significant, sensible and for the most part confirmed our expectations. The idea of using a board game as a model for a social system did prove to have relatively narrow boundaries, especially for a system as complex as the stock market. It was an interesting approach, but for future research projects on the topic we definitely recommend reverting to classical mathematical models, as they are simply more precise, easier to implement and follow-up questions or small changes to the model can be added more easily.

8 References

References

[1] Complete set of rules: https://www.wizards.com/avalonhill/rules/acquire.pdf

9 Appendix

First we describe the output and functionality of the program Acquire_One_Game.py. This program plays one game and prints all the tiles that were placed on the board as well as the stocks that are bought and the the board after each turn. This allowed us to examine the decisions of the players and assure that the game works properly, i.e. all the rules of the game are implemented correctly.

The program Statistics_program.py is used to simulate a certain number of games and produce a ranking of the players. We opted to include all different players so that a user is able to easily chose which players should be compared. This can be done in lines 6236-6239 by changing the name of the players. The different names are $Player\_normal$, $Player\_entrepreneur$, $Player\_small\_hotels$, $Player\_large\_hotels$, $Player\_conservative$ and $Player\_offensive$.

The program containing the adapting player is called Adaption_program.py. Since the adaption is a long process, the running time of the program is up to 25 minutes. The outputs are a graph as the one in figure (j), all the values of the adjusted variables, whether the variable converged or not and how many iterations were needed. It also prints the ranking when compared with the normal player as well as the time needed and the total number of games played.