Agent Based Simulation, Negotiation, and Strategy Optimization of Monopoly

Nicholas Loffredo

6/13/08

Abstract

This Agent Based Simulation project attempts to study whether Reinforcement Learning can be an effective method for agents to improve their play towards an optimal strategy, including negotiation, for Monopoly. Monopoly provides a useful test-bed for learning algorithms in a relatively simple environment, yet is still complex enough that many of the results and methods used can be applied to more relevant real-life situations. Various policy variables were implemented, along with associated aggressiveness levels of play, for each policy. The agents attempt to learn via reinforcement learning which policy settings improve the likelihood of winning. To test if an agent is learning, it is pitted against a non-learning agent for thousands of games, and the results are statistically analyzed. The outcome of the test gives very strong evidence that reinforcement learning is working. Using this Agent Based Simulation of Monopoly as a testbed provides a fertile environment for further research.

1 Introduction

Computers currently are unable to perform common human tasks such as understanding a language well enough to speak it and effectively communicate. A good example of this is negotiation. Humans are able to negotiate with one another for various goods. Computers, on the other hand, still have a long way to go in this regard. There is significant research and development ongoing towards computer-based systems being able to negotiate effectively:

they could be used in many situations that currently require people, such as in diplomacy, selling/buying goods, trading goods, or just negotiating with other people in general [11]. More importantly, it would allow people to instruct a robot/computer to negotiate using certain items and to meet certain goals, instead of having themselves or hiring other people to do it. These computers would be resistant to common human flaws, such as anger, impatience and/or forgetfulness.

Developing a computer-based system that can learn effectively in a limited environment is a first step towards being able to learn in a more complex one. The game of Monopoly is simple enough that learning should be able to be implemented within a year, yet complex enough that the method used to achieve the results may be able to be applied towards real life situations. By making a working simulation of Monopoly, a learning capability can be implemented for computer agents that will play the game.

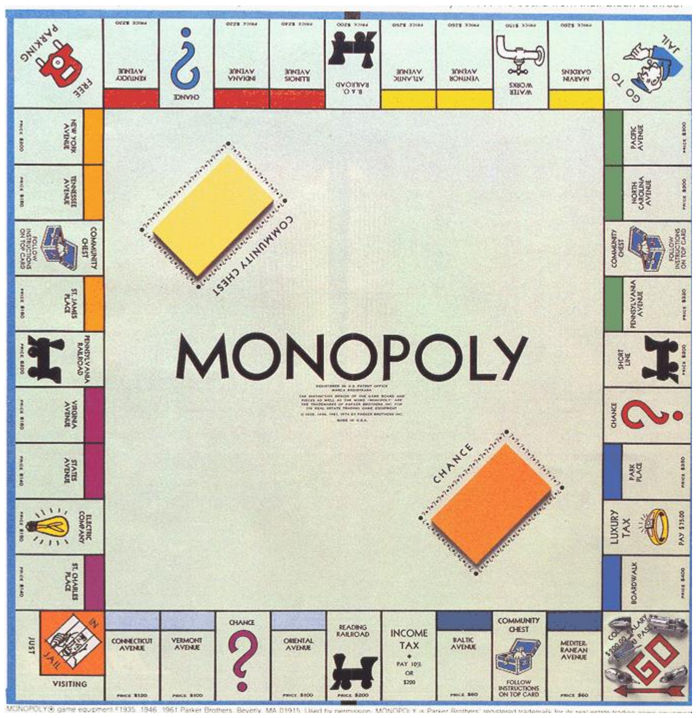


Figure 1: Monopoly Board

Fundamentally, the system must simulate all the rules of Monopoly. Agents must be able to move around the board based on the dice roll, and if they so choose, be able to buy titles they land on, and buy houses on monopolies they own. Additionally, they should be able to sell houses and mortgage properties according to the rules of the game. When an agent lands on a Chance or Community Chest square, they should receive the top card from a deck which was randomly sorted before the game, and follow through on the instructions. Furthermore, to explore the research areas contained herein, agents should also be able to learn to some extent (e.g., negotiate with each other).

2 Background

2.1 Similar Projects

Surprisingly, no research has been found regarding developing learning agents in Monopoly that attempt to learn an optimal strategy, although several handheld and pc based games were found. However, the AI's are likely all expert-based systems.

- An existing Monopoly-related simulation determines the probability of landing on each square in Monopoly. As it turns out, the square with the highest probability is Illinois Avenue! This information can be used as a reference to see if my Monopoly simulation results correlate to theirs, and can be a factor in determining whether or not the simulation works correctly. [8]

- Everything I Need to Know About Business I Learned from Monopoly, which discusses various strategies for Monopoly, can be used as a reference to see if agents develop the strategies the book discusses.

- According to "On Verifying Game Designs and Playing Strategies using Reinforcement Learning", neural nets could also be used for my reinforcement learning strategy. However, as there is not enough time for a neural net to be implemented, a novel application of reinforcement learning will be used. Also, my method of using the aggressiveness levels is general enough to have the potential to be used in other applications.

- "Reinforcement Learning in Extensive Form Games with Incomplete Information: the Bargaining Case Study" examines a Change or Learn

Fast algorithm to attempt to reduce the number of interactions between players to find an optimal strategy. "A Multi-Agent Reinforcement Learning Method for a Partially-Observable Competitive Game" shows that reinforcement learning agents have reduced errors over a long period of time compared to rule-based agents. If I had more time, I would attempt to use methods outlined in these papers to allow my agents to guess at the trading levels of their opponents, and thus be able to trade more efficiently.

- "On using Multi-agent Systems in Playing Board Games" discusses methods of making general programs that are able to play a large number of complex board games. This would be useful for my project, so that if I had enough time I could design my reinforcement learning method to be able to be used in a variety of situations.
- "Evolutionary Function Approximation for Reinforcement Learning" addresses how often the state of a program, game, or agent is evaluated using a 'value' function determined by a human. The paper proposes that the weights the human assigns are often incorrect, and proposes making a program that evolves its own evaluation function. This would help the Monopoly program, as a better evaluating function would allow agents to learn more.

2.2 Relevant Theory

According to Champandard, reinforcement learning "allows the machine or software agent to learn its behaviour based on feedback from the environment. This behaviour can be learnt once and for all, or keep on adapting as time goes by. If the problem is modelled with care, some Reinforcement Learning algorithms can converge to the global optimum; this is the ideal behaviour that maximises the reward." [2] A good example of reinforcement learning in real life would be learning to ride a bike or how to walk. People are not given a complex set of rules to follow—they receive feedback (i.e., the bike tipping over or falling), and modify their actions to prevent the mistakes again. As described by Singh and depicted in his graph below, reinforcement learning is learning from interactions in which agents base their actions on perceptions from an environment and an associated reward system.

An expert system, on the other hand, is when an agent follows a complex set of rules written by humans. An example of this would be filling

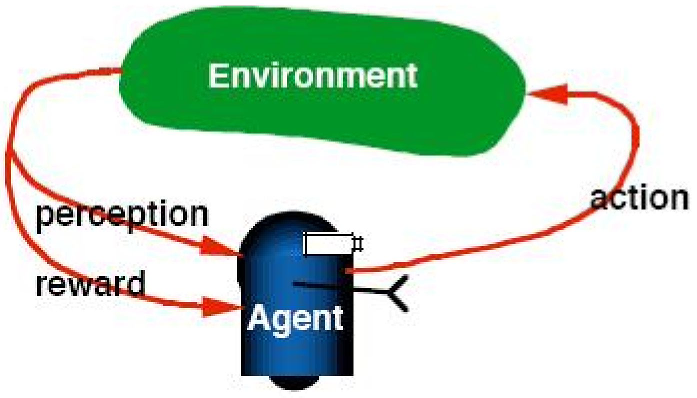


Figure 2: Reinforcement Learning Diagram

out forms or applications, as the applicant is not learning, merely following predetermined rules about how to fill out an application.

I decided to use reinforcement learning rather than expert systems for a number of reasons. The first reason is that I am more interested in the field of machine learning. Also, humans have not perfected Monopoly yet, and thus writing a program that only follows predetermined rules would likely limit the program's ability to play well. Using reinforcement learning, however, allows me watch programs learn, and try to push the envelope on how competent they can become. Finally, many of the items that people consider when playing Monopoly are hard for a computer to measure, and general actions that are simple for a human to take are difficult for a computer to learn how to do—that is a challenge I am willing to undertake.

3 Development

3.1 Objectives

The goals of this project are to be able to create a Monopoly game which follows the official rules of the original Monopoly game, to test whether or not reinforcement learning techniques can be effectively employed, and, time permitting, to find an optimal strategy for Monopoly.

3.2 Limitations

I wasn't able to find any existing source code for a Monopoly game simulation in Java, or designs for how to implement reinforcement learning in Monopoly. This restricted how far I would be able to go with the project, as I had to spend significant time designing, developing, and testing the Java code for the basic Monopoly game engine.

3.3 Development History

In the first quarter of the project, I designed the implementation of Monopoly, developed some of the core components of the monopoly simulation engine: agents 'roll' dice, buy properties, pay rent, buy houses, etc.. I also created an interface to display the monopoly board.

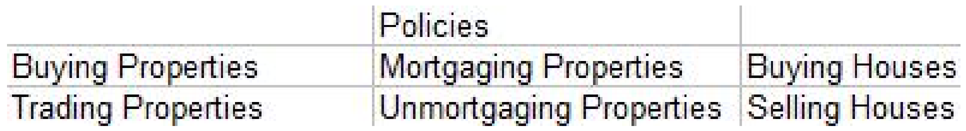
In the second quarter of the project, I implemented buying and selling houses, and mortgaging/unmortgaging properties. I formulated six policy variables and designed the associated aggressiveness level values for each policy, which heretofore will be referred to as the Aggressiveness Policy Vector (APV). The aggressiveness levels are used to determine how often an agent will act on the various policies, i.e., buying properties, buying/selling houses, or mortgaging/unmortgaging properties. I then expanded the scope of the policies to the different color groups. When that was finished, the learning algorithm for the agents was implemented so that agents gained the potential to learn over time to improve its play towards optimal values. A display was then created to show the aggressiveness levels and wins of each agent.

In the third quarter I added trading to my game. This ended up being more complicated than planned. The agents trade properties for cash based on how relevant the property is to their current situation.

In the fourth quarter, I added auctioning to my game, and was able to complete a large amount of testing on recently added items, as well as adding other small features, such as randomly deciding which agent takes his turn first each game. I was then able to explore, implement and test the reinforcement learning application, APV.

3.4 Design Criteria

To help agents learn, I set up policies as depicted below.



	Policies	
Buying Properties	Mortgaging Properties	Buying Houses
Trading Properties	Unmortgaging Properties	Selling Houses

Table 1: Aggressiveness Policies

For each policy, the agent would act according to the value of its aggressiveness level (0-10). A value of 0 means there is no chance of the agent taking the action when the opportunity presents itself, a value of 5 means it will have a 50 percent chance of taking the action, and a value of 10 means it takes the action with 100 percent certainty. To make the agents have more 'discretion' about the policies effects on different properties, I extended the policies to include subgroups consisting of the eight property groups. This extension will allow agents to have different strategies for different property groups. For example, the agent might be more likely to buy a red color group property, one of the properties with the highest chances of being landed on, and less likely to buy a purple color group, which has a lower chance of being landed on. In particular, these negotiations will be based on aggressiveness levels. For example, how far an agent is willing to drop/raise his initial price in order to complete a negotiation, which is a part of the trading policy.

Once the APV object had been created, modifying it to allow agents to 'learn' was fairly straight-forward. First, one of the APV values is randomly picked, and is randomly changed by either positive or negative 1. After 15 games have been played with the modified value, the program first checks that the selected value has been used within those 15 games (Because of the stochastic nature of the APV implementation, as described in section 3.5). If it has, then it checks whether or not the percent wins of the learning agent is greater than the percent wins 15 games ago. If it is, then the agent keeps the value change and stores whatever the sign change was that occurred previously. If it is not, it reverts the item to the previous value, and stores the negative sign change that occurred previously. Then, it repeats the process from the beginning, with the exception of the fact that the value change of any item that had previously been changed is determined by the stored value, and not by chance.

3.5 Testing and Analysis

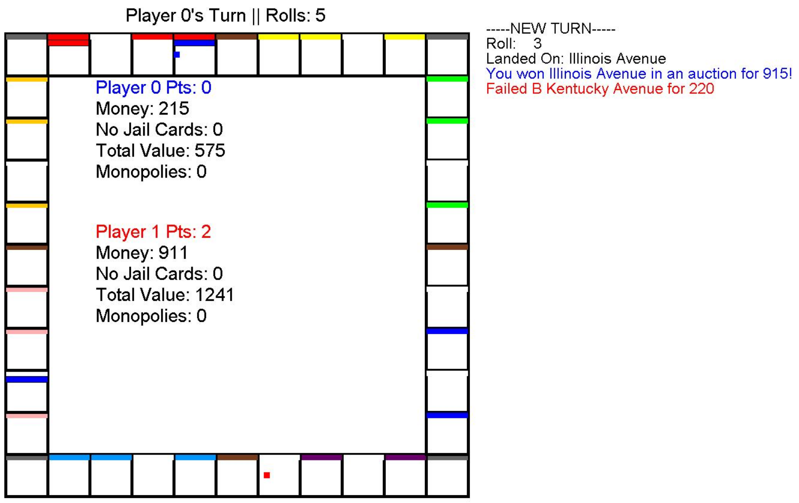
Code for a GUI was developed that allows the state of the Monopoly game to be observed. This allows for testing specific features of the game by running the game turn-by-turn and observing whether or not each new feature being added was implemented correctly. It also facilitated checking to see if any rules were violated.

The overall simulation was tested by adding features that allow the program to play multiple games in one run, display the number of wins and the aggressiveness levels of each agent. By initially setting all the aggressiveness levels to 5, an agent would initially have a 50 percent chance to take actions. Analysis continued with two agents. One agent, the aspiring learning agent, however, was given the potential to learn, whereas the other would not change. By then tallying up and comparing the number of wins of the learning agent versus the non-learning agent, the confidence I would have as to whether the former is actually learning could be ascertained. Doing this would show whether or not the agent was actually evolving sufficiently to beat the non-learning agent, i.e., if his win ratio over a large number of games was greater than 50 percent.

3.6 Procedure

I designed and started coding the game. Initially, to check if it was working, I had to print out what the player rolled and what property it landed on each turn. I found an image of a Monopoly board from the Internet to make it easier to test. I added the agent capability to buy houses on properties. After I was certain the players were moving correctly, I designed and coded a Monopoly interface for testing more sophisticated features. I also stored and displayed a list of messages that describe what happened each turn. This on-screen display made it much easier to debug, as depicted below.

Once the interface was implemented, I was able to begin adding new features. I designed and implemented the chance and community chest cards feature. This included all the varied effects they can have, such as jumping to another board location. After card coding and testing, I began ensuring that players bought houses and hotels according to the rules. After that, I began checking or adding in various smaller features, such as receiving 200 dollars for passing go, agents going bankrupt, etc. Once all these features were added, testing with multiple players was performed. I then began theorizing


Player 0's Turn || Rolls: 5

Player 0 Pts: 0
Money: 215
No Jail Cards: 0
Total Value: 575
Monopolies: 0

Player 1 Pts: 2
Money: 911
No Jail Cards: 0
Total Value: 1241
Monopolies: 0

```
-----NEW TURN-----
Roll:    3
Landed On: Illinois Avenue
You won Illinois Avenue in an auction for 915!
Failed B Kentucky Avenue for 220
```

Figure 3: Monopoly Interface

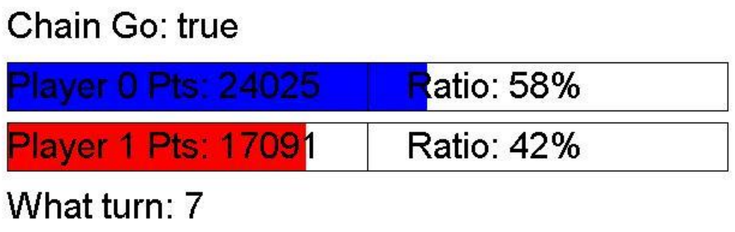
how I might be able to implement reinforcement learning for Monopoly. I then decided on and began coding for the APV.

In the third quarter, trading was tackled. Trading took significantly longer to implement than expected, due to its complexity and the agents' decision-making involved. Additionally, testing the trading feature was difficult, since most of the trading took place behind the scenes. Eventually I was able to implement trading according to the rules. However, the agents don't trade smarty, as they trade for properties every turn as long as they have enough money. As time was running out on the project, the decision was made to move on to auctions.

Auctioning was implemented after trading. Auctioning occurs when a player does not buy a property, and the property is put up to auction to all players (including the player who initially rejected it). The player whose turn it is begins the auction bidding 5 dollars. The game then cycles through each player and 'asks' them whether or not they are willing to bid 5 more dollars than the previous player. This cycle continues until no one is willing to bid more money. At that point, the highest bidder receives the property for the amount it bid.

4 Results

Once all the features were implemented, I began running my program thousands of times to see if the learning agent won significantly more times than the non-learning agent. As depicted in the graphic below, the learning agent had a win ratio of 58 percent after about 40,000 runs.


Chain Go: true

Player 0 Pts: 24025 Ratio: 58%

Player 1 Pts: 17091 Ratio: 42%

What turn: 7

Figure 4: Win Ratio after roughly 40,000 games

I performed a goodness of fit test to determine whether the win ratio was significant.

4.1 Chi-Squared Goodness of Fit Procedure

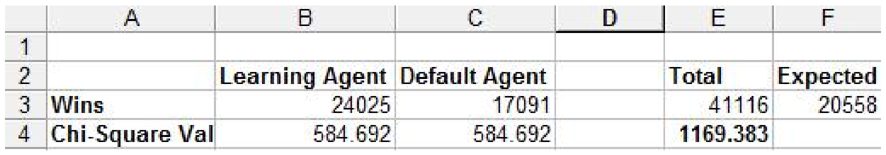
1. H_0: $P_0 = P_1$ (where these represent the true proportions).

   H_1: Proportions are not equal.

2. $\alpha = 0.01$. This means that if the χ^2 value is greater than the critical value, there is less than a 1% chance that the computed value could have occurred by chance and therefore less than a 1% chance of making an error in rejecting the null hypothesis that there is no difference in the win ratios.

3. Criteria: There is one degree of freedom. From Table 4 of Byrkit, the critical value of χ^2 for 1 degree of freedom and $\alpha = 0.01$ is 6.635. Thus we will reject H_0 if χ^2 is greater than 6.635.

4. Results: The computations are summarized in the table below with the observed number of wins for the learning and default agents, the expected number of wins for each, and finally, the computed χ^2 statistic of 1169.383.



	A	B	C	D	E	F
1						
2		Learning Agent	Default Agent		Total	Expected
3	Wins	24025	17091		41116	20558
4	Chi-Square Val	584.692	584.692		1169.383	

Table 2: Statistical Tests

5. Conclusion: Since the χ^2 value of 1169.383 is greater than the critical value of 6.635, we can reject H_0 and conclude that learning agent's wins are significantly different than the non-learning agent's wins.

From this goodness of fit test we can infer that the learning agent is learning.

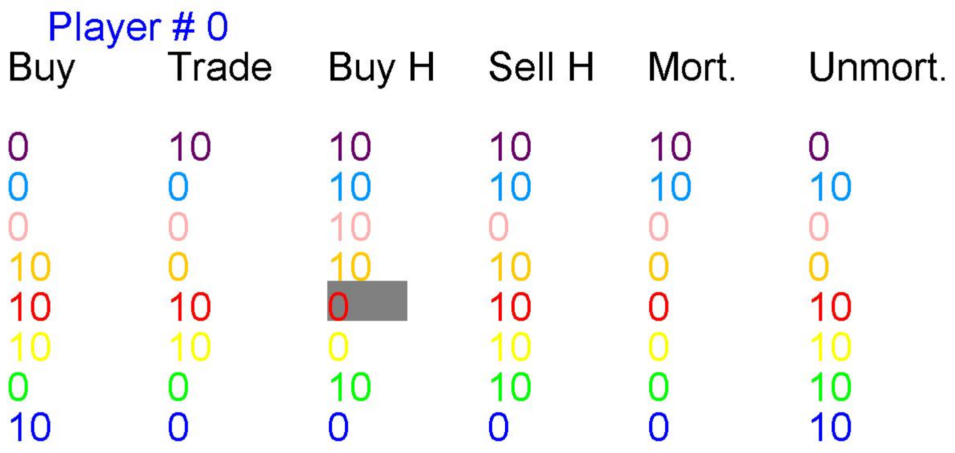
4.2 Aggressiveness Policy Vector

Looking into the results deeper, the final policy values for the learning agent can be investigated. In the system output table of the aggressiveness levels below, the learning agent is player 0. The rows/colors represent the respective color groups, progressing from the GO square clockwise. For example, the first row represents the purple titles (Mediterranean and Baltic) and the 5th row represents the red titles (Kentucky, Indiana, and Illinois). The columns represent the six respective policies, such as buying properties, buying houses, and mortgaging properties. The cells within the table represent the respective aggressiveness level for each color group and policy.

A surprising outcome, however, was that the aggressiveness chart turned out to be composed of entirely 0's or 10's. Although it is unlikely that an optimal strategy would be to never buy a property group under any conditions, and likely is due to an oversimplified learning algorithm, some interesting strategies emerged. For example, the learning agent chose to aggressively purchase the orange, red, and yellow properties, which are the three property groups most likely to be landed on. These are encouraging findings.

4.3 Chances of Landing on Properties

Data was gathered on the number of times agents an agent landed on each property in Monopoly as one additional data point to verify that the game is working correctly. When the results are compared against Jon's results,


Player # 0

Buy	Trade	Buy H	Sell H	Mort.	Unmort.
0	10	10	10	10	0
0	0	10	10	10	10
0	0	10	0	0	0
10	0	10	10	0	0
10	10	0	10	0	10
10	10	0	10	0	10
0	0	10	10	0	10
10	0	0	0	0	10

Figure 5: Aggressiveness Policy Vector

although they are not an exact match, they were closely correlated. I believe this can be attributed to the fact that Jon ran his program more turns than mine by three orders of magnitude.

5 Conclusions and Recommendations for Further Research

Computers have a difficult time performing common human tasks, such as learning a language well enough to be able to "talk" intelligently with someone. Monopoly, one of the most well known and understood board games in the United States, if not the world, provides a good environment to see whether or not a computer can "learn" through a number of strategies.

The goals of this project are to be able to accurately create a Monopoly game which follows the official rules of the original Monopoly game, to test whether or not reinforcement learning techniques can be effectively used to improve performance, such as in negotiation, and, time permitting, to find an optimal strategy for Monopoly.

While my program was not able to find an optimal strategy for Monopoly or have in-depth negotiation, I believe the project was a success because the learning agent does indeed learn. Considering that I had to spend most of my time was creating a Monopoly simulation from scratch–including the complex trading and auctioning capabilities–the fact that an agent was able to learn (using what appears to be an original idea) is notable in itself. Moreover, the fact that the APV application of reinforcement learning does not take long to implement–and works–in a way fulfills a 'proof-of-concept', showing that the idea is a viable option for learning.

Due to the general nature of the APV, it shows great promise for being able to be leveraged in other learning agent applications. This is significant, because the actual time required to implement the APV was fairly low. Most of my time was spent implementing the Monopoly game and related features. If the game engine had already been available, then most of the project time could have been spent to develop a more in-depth APV or other reinforcement learning approach with an even high win ratio.

If I had more time to work on the project, I would have added or modified a large number of features. Based on the current results, I do not think that my APV approach contained enough flexibility to be able to correctly inform an agent how to play. I would implement a model that would allow for important factors such as how far the agent is in the game; how many monopolies he owns; how many monopolies his opponents own; relating the propensity to buy a property as a function of how much money the agent has available with respect to the board position, etc. I would also have attempted to find a better function for evaluating the merits of an agent's position and properties, such as estimating an intrinsic value for each property (as a function of how many properties are already owned). Another area of future research would be to build a complementary expert-based Monopoly agent system and pit it against the reinforced learning agent. Also, I would further develop the trading method so that agents can do more than attempt to buy properties from other players each turn. I would make it so that they also sell their own properties, and trade their properties for other agents' properties.

References

[1] Axelrod, Alan. Everything I Know About Business I Learned from MONOPOLY. Pennsylvania: Running Press Book Publishers, 2002.

[2] Byrkit, Donald R. Elements of Statistics: An Introduction to Probability and Statistical Inference. Van Nostrand Reinhold Co, 1972.

[3] Champandard, Alex. "Reinforcement Learning Introduction." 8 Jun 2008 <http://reinforcementlearning.ai-depot.com/Intro.html>.

[4] Ghory, Imrin. "Reinforcement learning in board games." Publications. 4 May 2004. University of Bristol, Department of Computer Science. <http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf>

[5] Johansson, Stefan J. "On using Multi-agent Systems in Playing Board Games." 2006. <portal.acm.org>

[6] Kalles, Dimitrios. "On Verifying Game Designs and Playing Strategies using Reinforcement Learning." 2001. <portal.acm.org>

[7] Lazaric, Alessandro. "Reinforcement Learning in Extensive Form Games with Incomplete Information: the Bargaining Case Study." 2007. <portal.acm.org>

[8] Lin, Jon. Jons Monopoly Simulation. 1 Nov. 2007 <http://www.tesuji.org/monopoly.html>

[9] Matsuno, Yoichiro. "A Multi-Agent Reinforcement Learning Method for a Partially-Observable Competitive Game." May 28, 2001. <portal.acm.org>

[10] Singh, Satinder. "Reinforcement Learning: A Tutorial." 2006. <http://hunch.net/ jl/projects/RL/RLTheoryTutorial.pdf>

[11] Viamonte, M.J. et al. "Decision Making in an Agent-based Marketplace." Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on. 2006. 30.

[12] Whiteson, Shimon, and Peter Stone. "Evolutionary Function Approximation for Reinforcement Learning." J. Mach. Learn. Res. 7 (2006): 877-917.