Technical University of Crete
School of Electronic and Computer Engineering

Monte Carlo Tree Search in the "Settlers of Catan" Strategy Game

Diploma Thesis:
Emmanouil Karamalegkos

Committee:
Advisor: Georgios Chalkiadakis, Associate Professor
Committee Member: Antonios Deligiannakis, Associate Professor
Committee Member: Michail G. Lagoudakis, Associate Professor

Abstract

Classic approaches to game AI require either a high quality of domain knowledge, or a long time to generate effective AI behavior. *Monte Carlo Tree Search (MCTS)* is a search method that combines the precision of tree search with the generality of random sampling. The family of MCTS algorithms has achieved promising results with perfect-information games such as *Go*. In our work, we apply Monte-Carlo Tree Search to the non-deterministic game "Settlers of Catan", a multi-player board-turned-web-based game that necessitates *strategic* planning and negotiation skills. We implemented an agent which takes into consideration all the aspects of the game for the first time, using no domain knowledge.

In our work, we are experimenting using a reinforcement learning method *Value of Perfect Information (VPI)* and two *bandit methods*, namely, the *Upper Coefficient Bound* and *Bayesian Upper Coefficient Bound* methods. Such methods attempt to strike a balance between *exploitation* and *exploration* when creating of the search tree.

For first time, we implemented an agent that takes into consideration the complete rules-set of the game and makes it possible to negotiate trading between all players. Furthermore, we included in our agent an alternative initial placement found in the literature, which is based on the analysis of human behavior in Settlers of Catan games.

In our experiments we compare the performance of our methods against each other and against appropriate benchmarks (e.g., *JSettlers* agents), and examine the effect that the number of simulations and the simulation depth has on the algorithms' performance. Our results suggests that *VPI* scores significantly better than bandit based methods, even if these employ a much higher number of simulations. In addition to this, the simulation depth of the algorithm needs to be calculated so the method will neither get lost in deep simulations of improbable scenarios neither end up shortly without given a proper estimation of the upcoming moves. Finally, our results indicate that our agent performance is improved when the alternative, human behavior-based, initial placement method.

Δενδρική Αναζήτηση Monte Carlo στο Παιχνίδι Στρατηγικής "Άποικοι του Κατάν"

Περίληψη

Παραδοσιακές μέθοδοι Τεχνητής Νοημοσύνης χρειάζονται είτε υψηλή γνώση πάνω στον τομέα που εφαρμόζονται, είτε χρειάζονται αρκετό χρόνο ώστε να προσαρμοστούν. Η Δενδρική Αναζήτηση MonteCarlo (Monte Carlo Tree Search – MCTS) είναι μια μέθοδος αναζήτησης που συνδυάζει την ακρίβεια της δενδρικής αναζήτησης με την γενίκευση που προσφέρουν οι τυχαίες δειγματοληψίες. Η οικογένεια των αλγορίθμων MonteCarlo έχει επιτύχει αισιόδοξα αποτελέσματα σε κλασικά παιχνίδια, με πλήρη πληροφορία, όπως το επιτραπέζιο παιχνίδι *Go*. Στην εργασία μας, εφαρμόζουμε Δενδρική Αναζήτηση MonteCarlo στο μη-ντετερμινιστικό, επιτραπέζιο παιχνίδι στρατηγικής "Άποικοι του Κατάν". Αναπτύξαμε έναν πράκτορα που, για πρώτη φορά, λαμβάνει υπόψη όλες τις πτυχές του παιχνιδιού με καθόλου, εκ των προτέρων, γνώση του παιχνιδιού.
Στην εργασίας μας τρέχουμε πειράματα χρησιμοποιώντας τη μέθοδο ενισχυτικής μάθησης *Αξία της Τέλειας Πληροφόρησης (Value of Perfect Information)* και δύο μεθόδους «κουλοχέρη», την μέθοδο *Άνω Όριο Συντελεστή (Upper Coefficient Bound)* και μια επέκταση αυτής της μεθόδου - την *μέθοδο Μπαεσιανού Άνω Ορίου Συντελεστή(Bayesian Upper Coefficient Bound).* Στόχος αυτών των μεθόδων είναι να εξισορροπήσουν την σχέση εκμετάλλευσης-εξερεύνησης (exploration-exploitation problem) κατά την δημιουργία του δένδρου αναζήτησης.
Για πρώτη φορά στην σχετική με το Settlers of Catan βιβλιογραφία, αναπτύσσουμε έναν πράκτορα που χρησιμοποιεί MCTS ο οποίος λαμβάνει υπόψη *όλους* τους κανόνες του παιχνιδιού ενώ ταυτόχρονα *μπορεί να διαπραγματευτεί ανταλλαγές* με τους υπόλοιπους παίχτες. Επιπρόσθετα, συμπεριλάβαμε στον πράκτορα μια εναλλακτική στρατηγική αρχικής τοποθέτησης που υπάρχει στην βιβλιογραφία, και η οποία χρησιμοποιεί γνώση προερχόμενη από την ανάλυση συμπεριφοράς ανθρώπων κατά την τοποθέτηση κομματιών.
Στα πειράματά μας εξετάζουμε την απόδοση των μεθόδων μας όταν έρχονται αντιμέτωπες μεταξύ τους, ή με κατάλληλα επιλεγμένους αντιπάλους (benchmarks), όπως οι γνωστοί *JSettlers agents.* Επίσης, εξετάζουμε το πώς το βάθος και το πλήθος των προσομοιώσεων επηρεάζει τη απόδοση των αλγορίθμων μας. Τα αποτελέσματά μας υποδεικνύουν ότι η μέθοδος VPI με λιγότερες προσομοιώσεις είναι πιο ανταγωνιστική σε σχέση με τις μεθόδους ληστές, παρά το γεγονός ότι οι τελευταίες χρησιμοποιούν πολλαπλάσιες προσομοιώσεις. Επίσης, είναι σημαντικό να υπολογιστεί το βάθος των προσομοιώσεων στον αλγόριθμο, ώστε οι μέθοδοι να μην χάνονται σε ατέρμονες προσομοιώσεις απίθανων σεναρίων, αλλά ταυτόχρονα να μην τελειώνουν την εξερεύνηση, χωρίς να έχουν κάνει μια επαρκή εκτίμηση των επόμενων κινήσεων. Τέλος, τα πειράματά μας δείχνουν πως η εναλλακτική στρατηγική τοποθέτησης που εξετάσαμε βοηθάει στην αύξηση της απόδοσης του πράκτορά μας.

Contents

List of Figures

1 Introduction

Developing sufficient Artificial Intelligence (AI) for games was always challenging from a computer science point of view. Most AI agents, for unsolved games, are based either on stored plays either on hand-coded heuristic evaluation factions. A different approach is offered by Monte Carlo Tree Search (MCTS) methods which attracts the significant attention the last decade. MCTS is a method for making optimal decisions by combining the precision of tree search and the generality of random playouts. Research interest in MCTS risen rapidly due to its success with computer Go and potential application to a number of other difficult problems. Its applications extends beyond games and MCTS can theoretically be applied to any domain that can forecast outcomes by simulations.

1.1 Motivation: Real-Time, Non-Deterministic, Partially Observable Multiagent Environments

The family of MCTS algorithm had great success in deterministic games such as Go, therefore we needed to reach for the limits of this method. The next step from deterministic and perfect information games is the addition of Bayesian elements through the game, converting the game to Non-Deterministic and hiding parts among players. A famous strategy board game fulfilling all the criteria is the game *Settlers of Catan*. Several implementations have been made [4][8][21] but none of them tested their agents according to the complete rule set of the game.

Settlers of Catan is a strategy board game played by [2,4] players. Players in this game are trying to colonize an deserted island by gathering resources in order to build cities and roads. There are distinct turns for each player, which is very important for our implementation, and on each round the providing resources are determined on dice result.

1.2 Related Work

Szita *et al.* [4] made an agent selecting actions based on MCTS methods but on the action selection they used weights based on a priori domain knowledge. Mixing the Monte - Carlo philosophy with hand-coded evaluations. They also, remove from the simulation phase all actions concerning Development Cards. Also, they handicapped their agent by removing his ability to negotiate trading with other players, but all the other players can perform trades among them, leaving out one of the most critical aspects of the game.

The agent of Panousis [21] is rather similar to Szita *et al.* [4]. During the making of the tree, the agent makes a stack and place all the available moves, but it favors some moves. An action that will provide more value at the time it will be played (not by calculating a long-term plan) is favored over the others and placed on top of the stack. On each round, the agent is making all the actions that he is able to do by picking the first action from the stack. As in the work of Szita *et al.* [4], the development cards are ignored during the simulation, and throughout the game all trades are disabled. Also, the agent in each round spends all his resources, trying to build as many constructions as possible from his stack. If he cannot build a type of piece in the stack then just continues to the next one until it reach the bottom of the stack or he is out of recourses.

1.3 Contributions and Thesis Outline

In our work, we have implemented a MCTS agent using three different methods in various simulations depths. Our agent is using the whole rule set for the game and does not use any domain knowledge. Trading negotiations among all players are available and we give an estimated reward for the development cards during the simulation phase. Also, we included an empty action to our agents planning, so he can choose to do nothing if he has a better plan calculated.

In the following list we present our contributions, which will be discussed into the thesis:

1. We are the first to implement an agent with pure MCTS methods leaving out any domain knowledge in the sense that action selection is based only on the simulated evaluation of the algorithm.
2. Also, we are the first[1] to test our agent in an environment with complete rules-set. Including Developing cards estimation during the simulation step of the algorithm.
3. We examine the performance of our agent using different simulation depths, in order to optimize the *exploitation-exploration* dilemma at the search of the tree.
4. We devised and used a plethora of metrics to assert methods' performance.
5. We also tested whether results could be improved by using a recent method by Dobre *et al.* [22], which is a sophisticated calculation for the initial placement.
6. We test our agent using three different methods.
    - A standard among the MCTS family bandit method (UCT) and an improvement of this using Bayesian interference (BUCT).
    - And the dominant, based on our results, (VPI) reinforcement learning method.

In the following chapters we will describe the game "Settlers of Catan" (Chapter 2) and point out some of the main strategies in the game. Following that, we will present the JSettlers framework (Chapter 3) that we used for our experimentation and describe the enemies of our agent. At Chapter 4 we will provide all the necessary background for our thesis regarding the Monte Carlo Tree Search Algorithm, as well as all its different methods that we will use. Chapter 5 describes our agent implementation and the adaption of MCTS algorithm to the specific domain. The results from the experiments we made are presented at the chapter 6 while chapter 7 draws conclusions and outlines possible future work.

2 Settlers of Catan

This board game was created by Klaus Teber and it was first published in 1995. It is a landmark among board games because it combines simplicity, strategy and trading skills.
A few references about this game in the press:

“Settlers has become so popular in Silicon Valley that it's now being used as an icebreaker at some business meetings.”
-Wall Street Journal

“Over the past few years, Settlers of Catan has transformed from an activity enjoyed by a small niche of gamers into a mainstream hit.”
-The Atlantic

“Settlers manages to be effortlessly fun, intuitively enjoyable, and still intellectually rewarding, a combination that's changing the American idea of what a board game can be.”
-Wired Magazine

“...the new generation of post collegiate gamers is gravitating toward more complex games of exploration and trade...like Settlers of Catan...”
-The New York Times

“Settlers has spread from Stuttgart to Seoul to Silicon Valley...it has become a necessary social skill among entrepreneurs and venture capitalists.”
-The Washington Post

“Settlers has its own elegant economy, in which the supply and demand for five different commodities are determined by tactics, luck and the stage of the game.”
-Financial Times

The game is played by 2 to 4 players, and their goal is to colonize the uninhabited island of Catan. During the game players will construct cities, settlements and roads. By expanding their territory they win Victory Points (VP) and when a player reach 10 VP wins the game. In order to build something, players needs resources. Resources can be gained either by natural locations around the edifices of each player either by trading with other players. [1][2]

2.1 Board

2.1.1 Components

The board of the game is consists of 19 hexagon tiles of terrain surrounded by 18 hexagon tiles of sea and 18 tiles with numbers.

Terrain tiles:

1. Desert (1), produces nothing
2. Forest (4), produces wood
3. Hills (3), produces clay
4. Meadows (4), produces wool
5. Mountains (3), produces ore
6. Plains (4), produce wheat

Sea tiles:

1. Simple sea (9)
2. Trade ports (9)

Number tiles: The tiles with number represent all the possible dice outcomes aggregation. So the numbers belong in [2,12] with all numbers appear twice except the numbers 2 and 12 which are represented only once.

2.1.2 Setting up the board

The process of setting up the board starts by selecting all the 19 tiles of terrain. After shuffling, one column contained 5 tiles is set. Afterwards another column of 4 tiles is placed on each side of the previous column. Lastly, two columns of 3 tiles are placed on the right and the left side of the previous (figure 1).

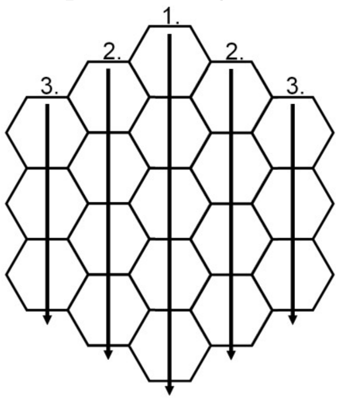


Figure 1: Placing hexes

After all the terrain tiles are set, one port tile is placed alternately by one tile of sea (figure 2).

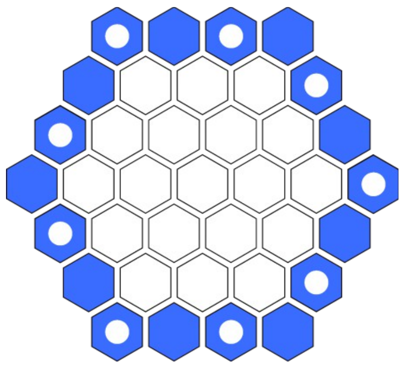


Figure 2: Placing ports

To complete the set up of the board, 19 number tiles are placed on the top of each terrain tile. To do so, the placement starts from a terrain tile at the outline of the hexagon which is located to one of the six corners of the hexagon. The placement continues to the next tile reverse clockwise, and when the outline is filled it continues to the second layer making a vortex to the center, skipping the terrain tile which is the desert (figure 3).

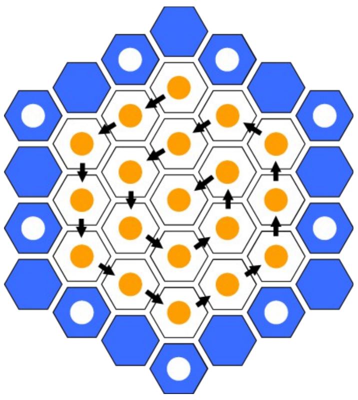


Figure 3: Placing numbers on hexes

The numbers are not placed in random order, but they follow this pattern:
5→2→6→3→8→10→9→12→11→4→8→10→9→4→5→6→3→11
So at the end, it will look like figure 4.

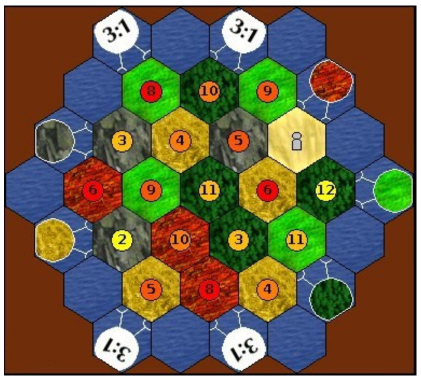


Figure 4: Final board

2.2 General Rules

Each turn of a player has the following actions:

1. Dice roll, all players takes resources based on their establishments.
2. The current player can make trade either with other players either with the stash.
3. The current player can build road/settlement/city or buy/play development cards.

2.2.1 Income - resources

The player starts his turn with a dice roll. The summarized result of the dice dictates which hexagons produce resources. Each player who has built a settlement or a city on an intersection takes resources.

2.2.2 Trade

The player can trade his resources.

I. **Internal Trade**

The player can offer to trade resources with all the other players and take counteroffers from other players. All the trades that will occur must include the current player. Players cannot trade when it's not their turn.

II. **Sea Trade**

a) The player can trade 4:1, meaning that the player can return 4 resources of the same kind and take 1 resource of any kind he wants.

b) If the player has built a settlement or a city at a port location, then better trades 3:1 or 2:1 are available.

Note that a player can always trade 4:1 even if he has not a built in port location.

2.2.3 Construction

In order to build a player must give a specific amount of resources. Players cannot build more than the pieces they have in their inventory.

a) **Road** : 1x[clay] + 1x[wood]
   - The new road must be adjacent with a piece of the player
   - In each corridor can be built only one road
   - The player with the biggest uninterrupted road (minimum 5) is awarded with the "Longest Road" which awards 2 victory points to the player. If a player exceed the road then he takes the "Longest Road" from the other player as well as the 2 victory points.

b) **Settlement**: 1x[clay]+1x[wheat]+1x[wood]+1x[wool]
   - A settlement is built on an intersection only if the three neighbor intersections are not occupied.

- The new settlement must be adjacent with at least one road of the player.
- For each settlement the player take 1 resource from all the hexagons during the income phase.
- Settlement awards 1 victory point to the player.

c) **City**: 3x[iron]+2x[wheat]
- A city can be built only as an upgrade of a settlement.
- When a player upgrades one of his settlements, he replaces the settlement with the city and the settlement becomes again available.
- For each city the player takes 2 resources from all the hexagons during the income phase.
- City awards 2 victory point to the player.

d) **Development** Card: 1x[iron]+1x[wheat]+1x[wool]
- When a player buys a development card, he gain one random from the stash.
- There are five different types:
  - ➢ Knight (14)
  - ➢ Victory Point (5)
  - ➢ Monopoly (2)
  - ➢ Road Building (2)
  - ➢ Year of Plenty (2)

2.2.4 Special Occasions

a) **Robber**
- When a player rolls 7, no player is gains any resources.
- All the players, who have more than 7 resources, choose half of them and they discard them. If they have odd number of cards, they round down the number.
- The player has to move the robber to another hexagon that he likes (cannot remain unchanged)
- When the player place the robber to a hexagon, he can steal 1 resource from a player who has built a city or settlement on this hexagon.
- If the number which is rolled is occupied by the robber, this hexagon does not produce resources for any of the players.

b) **Play Development Card**

A player, during his turn, can play a development card anytime he wants. But he must not have bought this card at the same round.

I. **Knight**
- When a player plays a knight card, we must do all the actions like he has rolled 7 on dice.
- All the Knight cards remain open at the player.
- When someone has 3 knight cards he is awarded with the "Large Army" and 2 Victory Points.
- If another player gain more knights than the owner of the "Large Army" he immediately takes the "Large Army" along with the 2 Victory Points.

II. **Monopoly**
When a player plays a monopoly card, he chooses one type of resource and all players must give him all the resources of the chosen type they have.

III. **Victory Points**
The cards with Victory Points remain hidden until the player has 10 Victory Points and he wins the game.

IV. **Road Building**
When a player plays this card, he can build two road pieces with no cost.

V. **Year of Plenty**
Playing this card, the player can take two resources from the bank.

2.2.5 Starting the Game

Each player starts the game with 5 settlement, 4 city and 15 road pieces in his inventory. The first player is selected randomly and the other players, in clockwise order. The first player places a road piece and a settlement piece on the board and all the other players follow according to their turn. When the last player places his pieces, then they start a new round of placing a road and a settlement but this time in a reverse order. In meaning that the last player will place first and the first player will place the last pieces. So at the beginning of the game each player starts with 2 Victory Points.

2.2.6 Game Overview

On a player's turn, the following actions can be made:

1. The player must roll the dice. The result of the roll, declares which hexagons produce resources (the result applies to all players).
2. The player may trade resource card with other player and/or use marine trade.
3. The player may build roads, settlements or cities and/or buy development cards. Player may also play one development card at any time during his turn.

Figure 5 represents an abstract flow chart of the game.

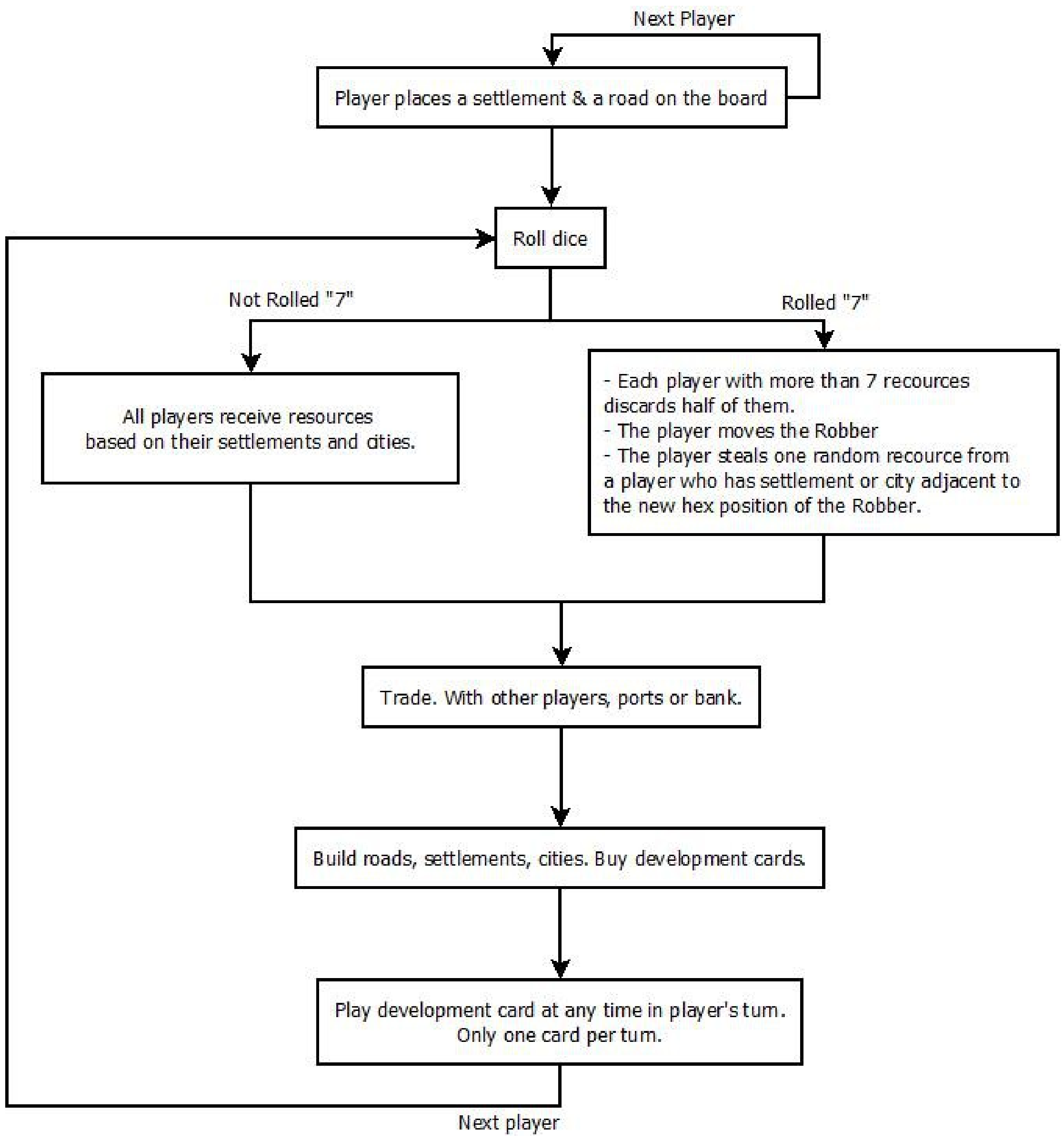

Next Player
Player places a settlement & a road on the board
Roll dice
Not Rolled "7"
Rolled "7"
All players receive resources based on their settlements and cities.
- Each player with more than 7 recources discards half of them.
- The player moves the Robber
- The player steals one random recource from a player who has settlement or city adjacent to the new hex position of the Robber.
Trade. With other players, ports or bank.
Build roads, settlements, cities. Buy development cards.
Play development card at any time in player's turn. Only one card per turn.
Next player

Figure 5: Turn flow-chart

2.2.7 End of the Game

The game ends, when someone in his turn has 10 or more Victory Points. A player in order to win, it must be his turn and he must has or acquire at least 10 Victory Points.

2.2.8 Tactics

Since the game of Settlers of Catan has a variable map, the tactical considerations of each game are different. There are nevertheless, some common points:

- At the beginning of the game, the most important resources are brick and lumber in order to build roads.
- Harbors have value. Especially when they are combined with great amounts of a certain resource.

- The placing of your first two settlements should leave enough space to expand.
- The key to victory is trading.

3 The JSettlers Framework

There are many software platforms for Settlers of Catan players, which are mostly based on heuristics evaluations and can easily be defeated by a human. One of the strongest is Robert S. Thomas' JSettlers, an open-source Java version of the game, which has heuristic-based AI players and is a basis of many Settlers of Catan servers online. For instance, Pfeiffer *et al.* [3] and Szita *et al.* [4] both use that environment to test their agents.
In our implementation, we use elements of and test our agent against the JSettlers environment, so it is important to present it to some extremely.

3.1 Interface

When a player enters a game, s/he interacts with the Game Interface which is composed by 4 player regions - the board region and the chat region. At the beginning there is no player sitting and all hexes of the board are water.

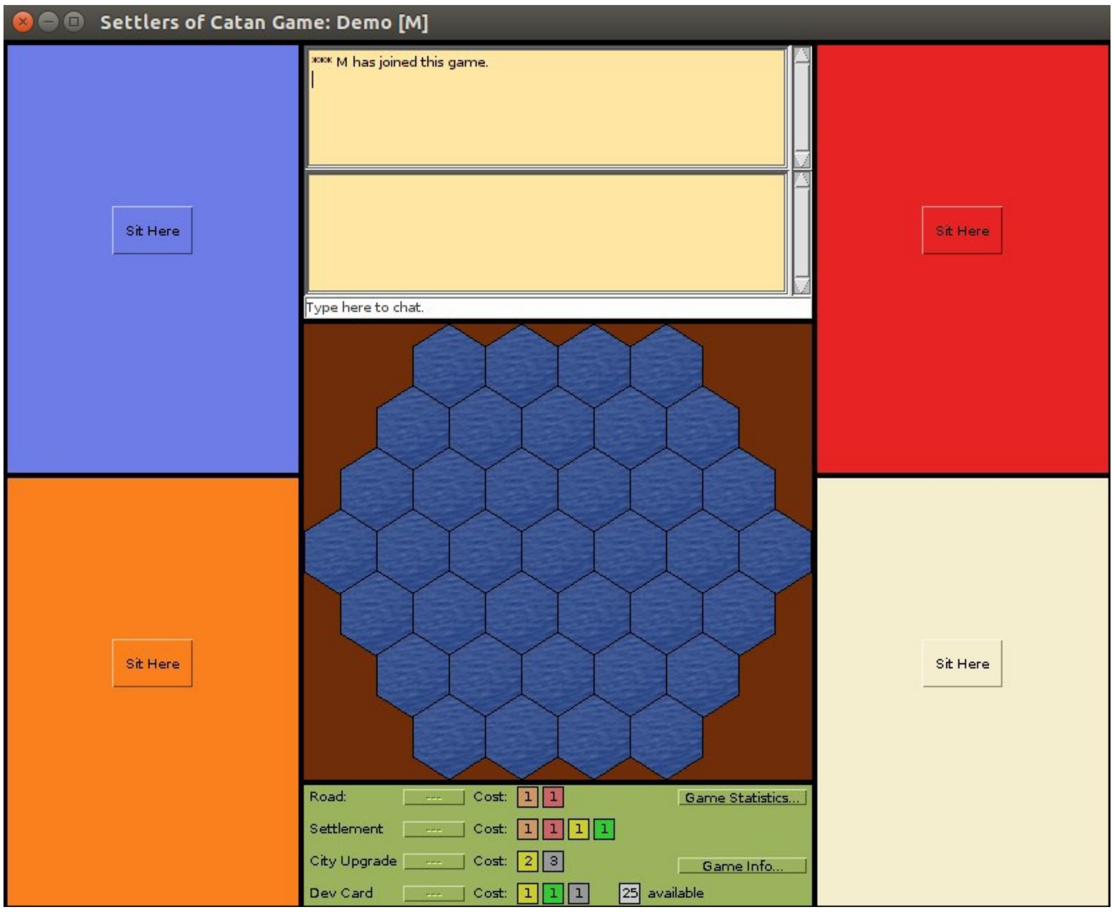
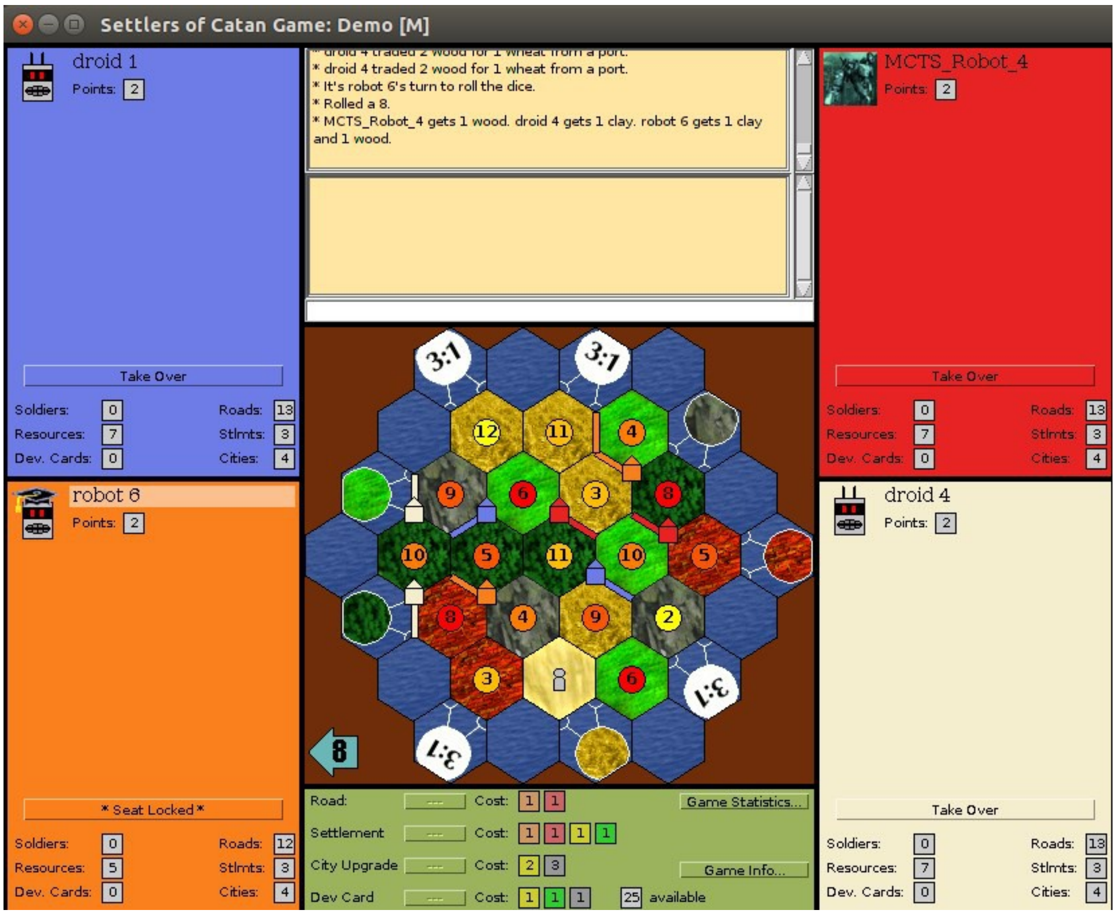


Figure 6: Empty interface

After the player is seated, he chooses against how many player he would like to play against (1 to 3). After the game starts, the board is created as described in the rules and a player, that is selected at random, starts first. After a few turns the view provided by the interface will be similar to that of Figure 7.


Settlers of Catan Game: Demo [M]

* droid 4 traded 2 wood for 1 wheat from a port.
* droid 4 traded 2 wood for 1 wheat from a port.
* It's robot 6's turn to roll the dice.
* Rolled a 8.
* MCTS_Robot_4 gets 1 wood. droid 4 gets 1 clay. robot 6 gets 1 clay and 1 wood.

droid 1 — Points: 2 — Soldiers: 0, Resources: 7, Dev. Cards: 0, Roads: 13, Stlmts: 3, Cities: 4

MCTS_Robot_4 — Points: 2 — Soldiers: 0, Resources: 7, Dev. Cards: 0, Roads: 13, Stlmts: 3, Cities: 4

robot 6 — Points: 2 — Soldiers: 0, Resources: 5, Dev. Cards: 0, Roads: 12, Stlmts: 3, Cities: 4

droid 4 — Points: 2 — Soldiers: 0, Resources: 7, Dev. Cards: 0, Roads: 13, Stlmts: 3, Cities: 4

Road: Cost: 1 1
Settlement: Cost: 1 1 1 1
City Upgrade: Cost: 2 3
Dev Card: Cost: 1 1 1 — 25 available

Figure 7: Complete Interface

There are two different player regions.

- Each player region displays all public information about each player. How many Knights (marked as "soldiers" on the interface) the player has played, how many roads, settlements, cities has available in inventory, how many unused Development Cards and how many resources (only the number not the type). Also, the owner of Large Army and Large Road (if there is any) are displayed in the area of the respective player.
- Each player can view additional own information regarding her actual Victory Points (calculate victory points from development cards) and what type of resources and development cards are available to her. Also, this area features the trading aspect of the player. There are three buttons for *clearing the sets*, *trading with ports or bank*, and for *offering resources* to other players.

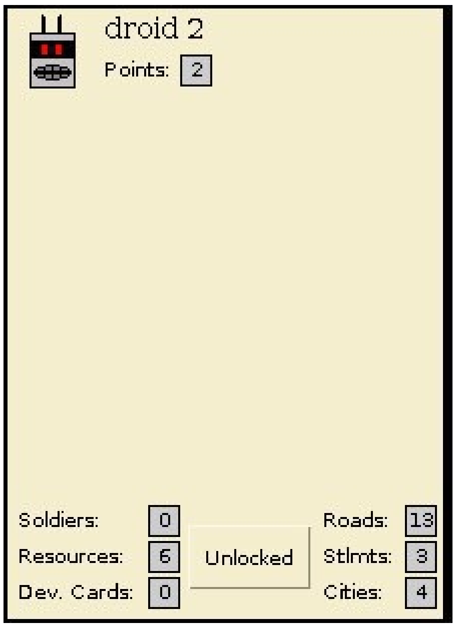

droid 2
Points: 2
Soldiers: 0
Resources: 6
Dev. Cards: 0
Unlocked
Roads: 13
Stlmts: 3
Cities: 4

Figure 8: Enemy player

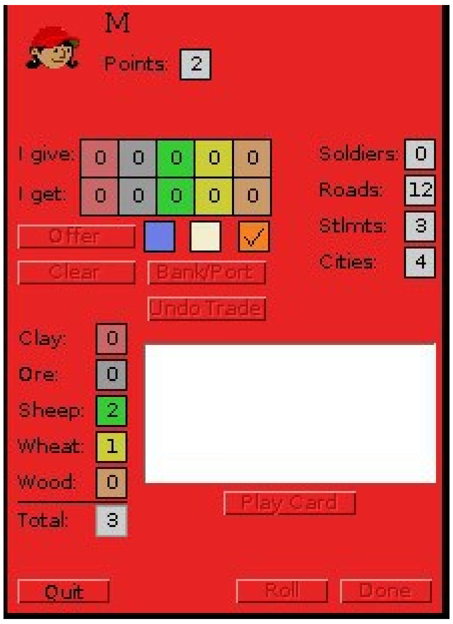

M
Points: 2
I give: 0 0 0 0 0
I get: 0 0 0 0 0
Soldiers: 0
Roads: 12
Stlmts: 3
Cities: 4
Clay: 0
Ore: 0
Sheep: 2
Wheat: 1
Wood: 0
Total: 3

Figure 9: Our player

At the bottom of the interface there is an area containing all the information about buying actions as well as how many are the remaining development cards. It shows what resources are needed in order to build a piece or buy a development card. It is also the interface to perform these actions. The boxes with the dashed indicate that the player does not have the necessary resources to build a piece. When the resources are gained, the dashes are replaced with a Buy indication. Also, Game Statistics and Game info buttons are used to show the statistics of the game and the options of the game respectively.

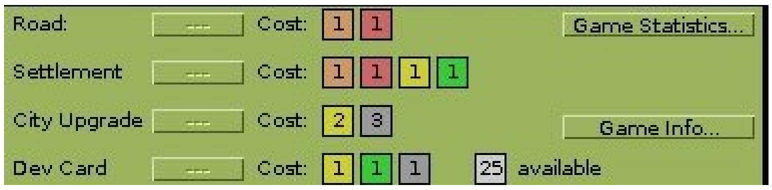
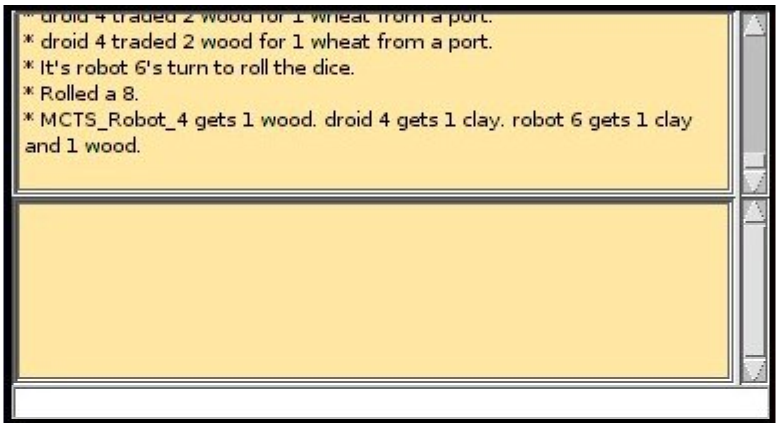

Road: --- Cost: 1 1
Settlement --- Cost: 1 1 1 1
City Upgrade --- Cost: 2 3
Dev Card --- Cost: 1 1 1 25 available

Figure 10: Game index

On the top of the game board, there is a region containing the chat and information area. This region is divided into two subareas. The first area above includes all the information about the game progress like dice roles, resources acquiring and trading offers. On the bottom area there is the chat between players .



```
* droid 4 traded 2 wood for 1 wheat from a port.
* droid 4 traded 2 wood for 1 wheat from a port.
* It's robot 6's turn to roll the dice.
* Rolled a 8.
* MCTS_Robot_4 gets 1 wood. droid 4 gets 1 clay. robot 6 gets 1 clay
and 1 wood.
```

Figure 11: Chat

In the next sections we briefly present the agent implementation of Robert Shaun Thomas.

3.2 Agent Implementation

Robert Shaun Thomas in [5], describes his agent implementation analysis as falling into 3 chapters, as follows.

3.2.1 Determining Options & Resource Estimation of Time

Thomas describes some of the basic strategies that a player can use to decide where make his initial placement of his pieces. These strategies aim to maximize the player's "building speed".
In order to make a choice of what to build, we need to know all the available options we have. The legal places to build are determined by the rules described in (2.2.3 - Construction). In the JSettlers framework there is an object keeping track of all the legal moves for each player and there is another object which includes all the potential places.
Now that the agent knows all the moves which are available to him, he needs to decide where to build. The initial placement of the settlements is important because not only determines what resources will be produced but it also determines where he can build in the future, affecting the strategy of the player. A player should place the settlement according to which subset of the 5 elements he prefer and maybe if he can combine resources with ports (he can choose to place a settlement at a port giving him the disadvantage on gaining incomes from one to two hexes instead of three. But if he has big income for a certain element the 2:1 trade could advantage). The choice that will be made is linked to three potential "end game" strategies:

- *road-building*, which necessities wood and clay production, which are used for roads and settlements builds,
- *city-building*, which require ore and wheat to build cities or buy development cards and

- *monopolizing*, in which a player tries to monopolize a resource and have access to a matching 2:1 port.

The choice between those strategies is based on the "*building speed"*. There are two approaches to calculate building speed, one taking into account trading with ports and the bank and one not. Trading with other players is not considered by any of the strategies.

1. The first is the simplest way, we ignore trading and consider how often we will receive resources based only on the types and numbers of the adjacent hexes of our pieces, but also by estimating how often these numbers are rolled. There are some "intended" inaccuracies in this algorithm because it employs estimates to make different kinds of decisions. So if we have an estimation speed that says that it takes longer to build a city that a road, this should be in reality, too.
   We can enhance this approach by taking into consideration the possibility of trading with ports or the bank. But this addition of the trading phase, can lead to an estimation that is inconsistent in some situations, because the algorithm does not look ahead concerning the resource production.
2. The second approach was based on the assumption that a more accurate estimation can lead to an improvement in gameplay. It is basically a modification to the algorithm described above, in which there was a restriction that a player can only receive at most one type of particular resource per roll. To do so we need to construct a frequency table to keep track of how many resources of a particular type player receives with a given frequency. This approach although, it was proved to be very slow to be used in practice and the author presents it as a motivation for future work.

Sometimes we can sacrifice precision for a rough but rather fast estimation.

3.2.2 Making a Plan and Deciding What to Build

In order to make a selection of the initial settlements and roads, Thomas [5] is calculating all the legal pairs of settlement spots and estimates how long it would take to build each possible piece. Then, the agents simply picks the spot with the faster building speed. This planning can be combined with other strategies of road-building or city-building. To do so, the agent takes weighted sums over potential strategies it could follow. For the placement of the initial roads, the algorithm is guessing spots that other players will occupy and in this new set of pieces is trying to find paths that can build good settlement later on. After the initial placement of settlements and roads, Thomas uses a rough plan as guide for creating a utility-based measure for making decisions in imperfect information environment. The results show that the computer players can win approximately 55% of the time against a single human opponent.

3.2.3 Negotiation and Trading

If one wants to make a sufficient negotiation, he should know first what he can do if the negotiation fails. This is called the Best Alternative to a Negotiated Agreement (BATNA) [5][6]. If we miss resources that we need based on our planning, we need to determine our BATNA by looking what resources we need and what resources we are willing to give up for trading. After the determination of our BATNA we should start making offers. We start with offers better than our BATNA by giving one resource that is easy to get for one resource that is hard to get. As *hard to get resource* is a resource that we have no direct access, like if we want iron and we do not have any settlement on hex with iron. On the other hand, *recourse easy to get* is a resource that either we have settlement on hex that produces that type or we have a port where we can trade for this recourse.

The estimation of BATNA comes from the alteration it affects on building speed. If the speed gets higher (decreased time we need to make it), then the offer is acceptable. To determine if another player will accept our offer, we must keep track of his activities (what resources refuses, what resources he ask for, what resources he gives). If our initial offer is refused, then we increase the amount of resources we give, either adding more resources either giving some more valuable (for us) resources. Counter offers to the players are made the same way.

Results of his attempt [5] showed that the performance of the agents dropped and that the computer players were agreeing to offers that are no in their favor. So, BATNA is not yielding very good results. In our future work, we intend to use Game Theory, Machine learning and argumentations methods in order to improve the negotiation aspect of our agent which now uses the approach at [5].

4 Monte Carlo Tree Search

Real-world games typically involve a delayed reward structure in which only those rewards achieved in the terminal states of the game accurately describe how well each player is performing. On the other hand there is the approach of Monte Carlo Tree Search (MCTS). The MCTS focuses on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space [7]. The application of Monte Carlo tree search in games is based on many simulations (playouts). In each simulation, the game is played out until the end of the game or until a predefined computational budget is reached, by selecting moves at random. The final game result of each simulation is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future simulations. MCTS algorithm has high success in games like computer Go, Scrabble and Bridge.

4.1 Overview

The basic idea of MCTS is very simple. A tree is build in an incremental and asymmetric manner [7]. For each iteration a *tree policy* [7] is used to find the most urgent node. When the most immediate node is found, a simulation is run from this node and the tree is updated based on the results. Moves during the simulation are made through some *default policy* which in the simplest way is to make random moves. A great benefit of MCTS is that the values of intermediate states do not have to be evaluated, as for minimax search, which reduces the amount of domain knowledge required [7]. Only the value of the terminal state at the end of each simulation is required. The key factor in the algorithm of MCTS is the balance between exploration (look to unexplored areas) and exploitation (look in promising areas) which is trying to achieve with the tree policy.

4.2 Algorithm

The basic algorithm involves iteratively building a search tree until some predefined computational budget is reached, at which point the search is stopped and the best-performing root action is returned [7]. Each node in the search tree represents a state of the domain, and directed links to child nodes represent actions leading to subsequent states.
Four steps are applied per search iteration [8]:

1. **Selection**: Starting at the root node, the *tree policy* is recursively applied to descend through the tree until we reach the most urgent *expandable* node. A node is expandable if it represent a non-terminal state and has unexpanded children. The tree policy selects or creates a leaf node from the nodes already contained within the search tree.

2. **Expansion**: A child node is added to expand the tree, according to the available actions.
3. **Simulation**: A simulation is run from the new node according to the *default policy* to produce an outcome. The default policy plays out the domain from a given state to produce a value estimation.
4. **Backpropagation**: The simulated results are updating the selected nodes with their statistics. This step does not use any policy itself, but updates node statistics that inform future tree policy decisions.

Figure 12 shows an iteration of the basic MCTS will start at the root node t_0 and recursively select child nodes, according to some utility function, until a node t_n is reached that is either terminal state or not fully expanded. An unvisited action α from this state s is selected and a new leaf node t_l is added to the tree, which describes the state s' reached from applying action α to state s. This completes the tree policy component for this interaction.
A simulation is then run from the newly expanded leaf node t_l to produce a reward value Δ which is then backpropagated up to nodes selected for this iteration, in order to update node statistics. Each node's visit count is incremented and its average reward updated given Δ.
As soon as the search is interrupted or the computation budget is reached the search terminates and an action α of the root node t_0 is selected by some mechanism.

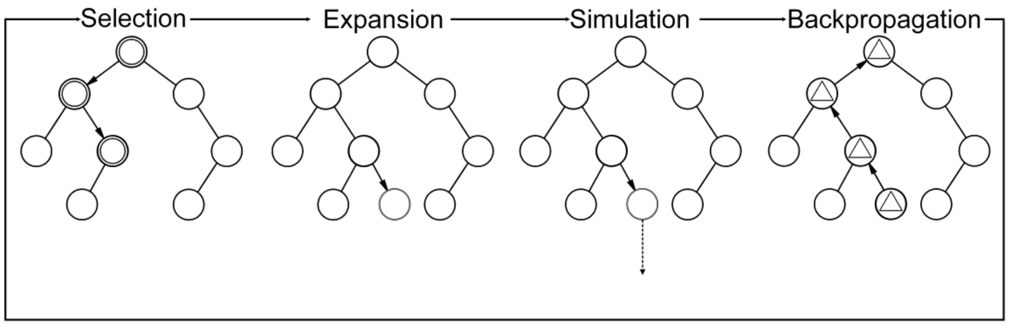


Figure 12: Monte Carlo Tree Search steps

The steps are summarized in pseudocode below [7]:

General MCTS approach

```
function MCTSSEARCH($s_0$)
    create root node $υ_0$ with state $s_0$
    while within computational budget do
        $υ_l$ ← TREEPOLICY($υ_0$)
        Δ ← DEFAULTPOLICY(s($υ_l$))
        BACKUP($υ_l$; Δ)
    return a(BESTCHILD($υ_0$; 0))
```

Figure 13: General MCTS approach

In Figure (13), u_0 is the root node corresponding to state s_0, u_l is the last node reached during the tree policy stage and corresponds to state s_l and Δ is the reward for the terminal state reached by running the default policy from state s_l. The result of the overall search α(BESTCHILD(u_0)) is the action α that leads to the best child of the root node u_0, where the exact definition of "best" is defined by the implementation.

4.3 Characteristics

In this section, we will describe some of the characteristics that make MCTS a popular choice for a variety of domains [7].

4.3.1 Aheuristic

One of the most significant benefits of MCTS is that it does not need domain knowledge, making it readily applicable to any domain that can be modeled using a tree. Although a full-length minimax is optimal, the quality of depth-limited minimax depends significantly on the heuristics used to evaluate leaf nodes. In games like Chess, where reliable heuristics have emerged after decades or research, minimax performs rather well. In cases like Go, however, which has large branching factors and useful heuristics are much more difficult to formulate, the performance of minimax degrades significantly.

4.3.2 Anytime

MCTS algorithm backpropagates the outcome of each game immediately which ensures all values are always updated following every iteration of the algorithm. This allows the algorithm to return an action which is high-quality, given current knowledge at any time.

4.3.3 Asymmetric

The tree selection allows the algorithm to favor more promising nodes, leading to an asymmetric tree over time. The tree shape that emerges can even be used to gain a better understanding about the game itself. For instance, Williams *et al.* [13] demonstrates that shape analysis applied to trees generated during UCT search can be used to distinguish between playable and unplayable games.

4.4 Bandit-Based Methods

Bandit problems are a class of sequential decision problems, in which one needs to choose amongst *K* actions in order to maximize the cumulative reward by consistently taking optimal actions [7][9]. Since the reward distributions are unknown and potential rewards must be calculated based on past observations, the action choice is rather difficult. The trade-off between the need to obtain new knowledge and the need to use that knowledge to improve performance is one of the most basic trade-offs is this kind of methods, and optimal performance usually requires some balance between exploratory and exploitative behaviors. Known as *exploitation-exploration* dilemma.

Now, for Bandit problems, it is useful to know the *Upper Confidence Bound (UCB)* that any given arm will be optimal. The simplest UCB policy is *UCB1* [9] which dictates to play arm j that maximizes:

$$UCB1 = \acute{X}_j + \sqrt{\frac{2\ln n}{n_j}} \qquad (0)$$

where $\acute{X}_j$ is the average reward from arm *j*, n_j is the number of times arm *j* was played and *n* is the overall number of plays so far. The first part of the equation encourages the *exploitation* and the second part the *exploration*.

4.4.1 Upper Confidence Bounds for Trees (UCT)

The *Upper Confidence Bound for Trees* (UCT) is the most popular algorithm in the MCTS family [9]. The success of MCTS is primarily due to the proposed use of UCB1 (4.3.1) as tree policy (4.2), by Koscis *et al.* [10] and Szepesvari *et al.* [11]. In treating the choice of child node as a multi-armed bandit problem, the value of a child node is the expected reward approximated on the simulations.
UCB1 has some promising properties: it is very simple and efficient. It is thus a promising candidate to address the exploration-exploitation dilemma in MCTS: every time a node is to be selected within the existing tree, the choice may be modeled as an independent multi-armed bandit problem. A child node *j* is selected to maximize:

$$UCT = \acute{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \qquad (0)$$

where n is the number of times the current node has been visited, n_j the number of times child j has been visited and $C_p > 0$ is a constant. If more than one child node has same maximal value, we choose randomly [11]. The values of $\acute{X}_j$ are within [0,1]. Previously unvisited children are assigned ∞, to ensure that all children of a node are considered at least once before any child is expanded further. The constant in the exploration term C_p can be adjusted to lower or increase the amount of exploration performed. The value $C_p = 1/\sqrt{2}$ was shown [11] to reward set in the range [0,1]. Figure 14 shows the UCT algorithm in pseudocode [12].

The UCT algorithm

```
function UctSearch($s_0$)
    create root node $v_0$ with state $s_0$
    while within computational budget do
        $\upsilon_l \leftarrow$ TreePolicy($\upsilon_0$)
        $\Delta \leftarrow$ DefaultPolicy(s($\upsilon_l$))
        Backup($\upsilon_l$; $\Delta$)
    return a(BestChild($\upsilon_0$; 0))

function TreePolicy($\upsilon$)
    while $\upsilon$ is nonterminal do
        if $\upsilon$ not fully expanded then
            return Expand($\upsilon$)
        else
            $\upsilon \leftarrow$ BestChild($\upsilon$;$C_p$)
    return $\upsilon$

function Expand($\upsilon$)
    choose $\alpha \in$ untried actions from A(s($\upsilon$))
    add a new child $\upsilon'$ to $\upsilon$
        with s($\upsilon'$) = f(s($\upsilon$), $\alpha$)
        and a($\upsilon'$) = $\alpha$
    return $\upsilon'$

function BestChild($\upsilon$,c)
    return $\underset{\upsilon' \epsilon\, childrens\, of\, \upsilon}{argmax} \frac{Q(\upsilon')}{N(\upsilon')} + c\sqrt{\frac{2\, lnN(\upsilon)}{N(\upsilon')}}$

function DefaultPolicy(s)
    while $s$ is non-terminal do
        choose $\alpha \in$ A($s$) uniformly at random
        $s \leftarrow$ f($s$,$\alpha$)
    return reward for state $s$
```

```
function BACKUP($\upsilon$,$\Delta$)
   while $\upsilon$ is not null do
        $N(\upsilon) \leftarrow N(\upsilon) + 1$
        $Q(\upsilon) \leftarrow Q(\upsilon) + \Delta(\upsilon, p)$
        $\upsilon \leftarrow$ parent of $\upsilon$
```

Figure 14: UCT Algorithm

Each node υ has four pieces of data associated with it: the associated state $s(\upsilon)$, the incoming action $\alpha(\upsilon)$, the total simulation reward $Q(\upsilon)$ and the visit count $N(\upsilon)$. Instead of storing $s(\upsilon)$ for each node, it is often more efficient in terms of memory usage to recalculate it as the Tree Policy descends the tree. The term $\Delta(\upsilon,p)$ denotes the component of the reward vector Δ associated with the current player p at node υ. The return value of the overall search in the pseudocode of Figure 14 is α(BESTCHILD(υ_0)) which will give the action α that leads to the child with the highest reward, since the exploration parameter c is set to 0 for the final call on the root node υ_0. The algorithm could have instead returned the action that leads to the most visited child; these two options will usually describe the same action [12].

4.4.2 Bayesian Upper Confidence Bounds for Trees (BUCT)

In contrast to a distribution-free algorithms like UCT, Tesauro *et al.* [14] proposes a fundamentally different approach based on *Bayesian inference*. Stochastic trial results at leaf nodes are combined with prior reward information to yield posterior distributions; these then propagate upward according to the appropriate inference model to determine interior node distributions.

In BUCT, each node *i* in the tree maintains a probability distribution P*i*(*x*) over its true expected reward value. Inference of the interior node probability distributions begins at the leaves and propagates up to the root node. Before any trials have been performed, the leaf nodes are initialized to conjugate prior distributions that are appropriate to the leaf node reward distributions. When trials at leaf nodes are performed, the results are combined with priors to compute posterior distributions, in the standard way.

Each interior node computes an *extremum distribution* over its child node distributions and is given by:

$$P_{max}(X)=\sum_{i} P_i(X)\prod_{j\neq i} C_j(X) \qquad (0)$$

where C_j is the Cumulative distribution function (CDF) of P_j.

Having computed distributions for the interior nodes, we now consider distributional analogs of existing distributional-free sampling formulae. Tesauro *et al.* [14] proposes two modified versions of UCB1 to descend the tree and choose where to sample next.

The first version (BUCT1) simply replaces the average reward $\acute{r}_i$ of child node *i* by μ_i, the mean of P_i:

$$BUCT\,1\colon maximaze\ B_i=\mu_i+\sqrt{\frac{2\,lnN}{n_i}} \qquad (0)$$

μ_i : Mean of P_i

N: Total trials of all arms

n_i : Number of trials for each arm

P_i: Probability distribution over its true expected reward value

There is an error on equations (4) and (5) on the following references [7][21]

The second sampling formula, additionally replaces the $\frac{1}{\sqrt{n_i}}$ factor in the exploration term by σ_i, the square root of the variance of P_i:

$$BUCT\,2: maximaze\; B_i = \mu_i + \sqrt{2 lnN}\,\sigma_i$$

(0)

μ_i : Mean of P_i

N: Total trials of all arms σ_i

σ_i : Square root of variance of P_i

P_i: Probability distribution over its true expected reward value

Equation 5 is motivated by the central limit theorem result σ_i^2 $1/n_i$ the simple bandit case and by the compelling intuitive notion from Interval Estimation that sampling according to expected value plus expected uncertainty provides effective tradeoff of exploration - exploitation.

The results in [14] indicate that the equation 5 outperforms equation 4 so in our implementations we chose to use equation 5.

There is an error on equations (4) and (5) on the following references [7][21]

4.5 Value of Information Method

As mentioned earlier, a central problem in learning in complex environments is balancing exploration of untested action against exploitation of actions that are known to be good. The benefit of exploration can be estimated using the classical notion of *Value of Information*. Aiming to find a policy that maximizes the expected reward of an agent. Dearden's *et al.* [15] approach builds on the notion of Q-Value Distributions and in [16] they present a Bayesian approach to model-based reinforcement learning. They present two new approaches to exploration:

- **Q-value sampling**: Wyatt *et al.* [17] proposed Q-value sampling as method for solving bandit problems. The idea is to represent explicitly the agent's knowledge of the available rewards as probability distributions; then, an actions is selected stochastically according to the current probability that is optimal. This probability depends monotonically not only on the current expected reward (exploitation) but also on the current level of uncertainty about the actual reward (exploration). Dearden *et al.* [15] extend this approach to multi-state reinforcement learning problems, creating a Bayesian method for representing, updating and propagating probability distributions over rewards.
- **Myopic-VPI**: Myopic value of perfect information provides an approximation to the utility of an information tradeoff. Like Q-value sampling, myopic-VPI uses the current probability distributions over rewards to control exploratory behavior.

Their results indicate that the state space is explored more effectively (than conventional model-free learning algorithms) and that performance advantage appears in bigger problems.
In our implementation, we chose Myopic-VPI action selection, because it was uniformly the best approach on many domains. [15][18][19][20]

4.5.1 Myopic-VPI selection

This method considers quantitatively the question of policy improvement though exploration [15]. Its use in tree search can also been seen as a form of exploration against the expected cost of doing potentially suboptimal action.

We start by considering what can be gained by learning the true value $\mu_{s,\alpha}^{¿}$ of $\mu_{s,\alpha}$. If this knowledge does not change the agent's policy, then rewards would not change. Thus, the only interesting scenarios are those where the new knowledge does change the agent's policy. This can happen in two cases:

a) When the new knowledge shows that an action previously considered sub-optimal is revealed as the best choice.

b) When the new knowledge indicates that an action that was previously considered best is actually inferior to other actions.

For case (a), suppose that *α₁* is the best action; that is $E[\mu_{s,\alpha_1}] \geq E[\mu_{s,\alpha'}]$ for all actions α'. Moreover suppose that the new knowledge indicates that α is a better action; that is, $\mu^{¿}_{s,\alpha} \geq E[\mu_{s,\alpha_1}]$. Thus, we expect to gain $\mu^{¿}_{s,\alpha} - E[\mu_{s,\alpha_1}]$ by virtue of performing α instead of α*.
For case (b), suppose that α_1 is the action with the highest expected value and α_2 is the second-best action. If the new knowledge indicates that $\mu_{s,\alpha_1} < E[\mu_{s,\alpha_2}]$ then the agent performs α_2 instead of α_1 and we expect to gain $E\left[\mu_{s,\alpha_2}\right] - \mu^{¿}_{s,a_1}$.
To summarize , we define the gain from learning the value of $\mu^{¿}_{s,\alpha}$ of $\mu_{s,\alpha}$ as:

$$Gain_{s,\alpha}\left(\mu^{¿}_{s,a}\right) = \begin{cases} E\left[\mu_{s,\alpha_2}\right] - \mu^{¿}_{s,a_1}, if\ a = a_1 \wedge \mu^{¿}_{s,\alpha} < E[\mu_{s,\alpha_1}] \\ E\left[\mu_{s,\alpha_2}\right] - \mu^{¿}_{s,a_1}, if\ a \neq a_1 \wedge \mu^{¿}_{s,\alpha} < E[\mu_{s,\alpha_1}] \\ 0, otherwise \end{cases} \qquad (0)$$

where, again, α_1 and α_2 are the actions with the best and second best expected values respectively. Since the agent does not know in advance what value will be revealed for $\mu^{¿}_{s,\alpha}$, we need to compute the *expected gain* given our prior beliefs. Hence the expected value of perfect information about $\mu_{s,\alpha}$ is :

$$VPI(s,\alpha) = \int_{-\infty}^{\infty} Gain_{s,\alpha}(x) Pr\left(\mu_{s,\alpha} = x\right) dx \qquad (0)$$

The value of perfect information gives an upper bound on the myopic value of information for exploring action α. The expected *cost* incurred for this exploration is given by the difference between the value of α and the value of the current best action. This suggests we choose the action that maximizes:

$$E\left[Q\left(s,a'\right)\right]-E[Q(s,a)]$$
$$\max_{a'} ¿$$
$$VPI\left(s,a\right)-¿ \qquad (0)$$

We see that the value of exploration estimate is used as a way of boosting the desirability of different actions. When the agent is confident of the estimated Q-values, the VPI of each action is close to 0, and the agent will always choose the action with the highest expected value.

5 Agent Implementation

In this chapter we will describe out proposed agent implementation, using Monte Carlo Tree Search. Unlike what is done in [4] and [21] our MCTS implementation is pure, in the sense that it does not contain any heuristics. Note that this is the case even for the node selection [4], as we discuss later on. Also, we implemented negotiations between the player and bank/port for the first time (in an approach using MCTS).

5.1 Overview

We implemented the Settlers of Catan game in the JSettlers software module, as mentioned on Chapter 3. Because of the Internet-based architecture of JSettlers (it was built to allow human participation over the internet), gameplay is fairly slow: a single four-player game takes about 15 minutes on an average PC and our algorithm requires hundreds of simulations of simulated games before making a single step.

5.2 Algorithm

The MCTS algorithm is used for action selection, when the agent has to decide his next move. This move is determined by calculating the available actions which includes:

- buying a City
- buying a Settlement
- buying a Road
- buying a Development Card
- empty action

and where to place them (rules of the game at Chapter 2). Trading with other players are not part of the action selection, therefore are not part of the MCTS Algorithm since the negotiation phase is over when our agent starts to build the search tree. However, our agent takes into consideration trading with ports/bank when calculating all available moves (Chapter 5.8.1).

The algorithm starts from an initial Root Node and calculates all available actions of our agent. Then through the *selection step* the most urgent action (node) is selected. Then the selected node is expanded through the *expansion step*. Subsequently, the expanded node's reward is calculated through *simulation step*. When simulation is complete the reward and all the statistics are back-propagated to the tree through the *backpropagate step*. This algorithm continues until a computational budget is reached. After the computational budget is reached the best-performing action is returned.

5.2.1 Computational Budget

We had to make a decision about setting a computational budget for the main loop of the algorithm concerning the **time cut-off**. The time that we can provide to our algorithm to run its methods. To make that decision we had to make a tradeoff between delaying the game and gaining sufficient time for simulations to run. We did not want to postpone the game risking of "breaking" the pace of the game and frustrate other players. Nevertheless we need to have enough time to run simulations so the estimated value be correct. After careful consideration and experiments on actual games with human adversaries, we decided to set the time cut-off at **15 seconds**. We are currently experimenting with cut-off 30 seconds but despite the increase of simulations we notice only a slight increase in the agent's scores.

5.2.2 Selection Step

The selection step is used for selecting the most urgent expandable node, in order to descend and build the tree. A node is expandable if it represent a non-terminal state and has unvisited children. Which node is the most urgent is determined from the selection formula of the method. In our implementation we use three different methods UCT, BUCT and VPI according to Chapters 4.4.2, 4.4.3 and 4.5.1 respectively. For clarity, we restate the methods.

5.2.2.1 UCT

The selection formula of UCT method is:

$$UCT=\acute{X}_j+2C_p\sqrt{\frac{2\ln n}{n_j}} \qquad (0)$$

$\acute{X}_j$: Average reward of node *j*

C_p: $\frac{1}{\sqrt{2}}$

n: The times this node was visited

n_j: The times child *j* was visited

If more than one child node has same maximal value, we choose randomly [11]. The values of $\acute{X}_j$ are within **[**0,1**]**. Previously unvisited children are assigned ∞, to ensure that all children of a node are considered at least once before any child is expanded further. The constant in the exploration term C_p can be adjusted to lower or increase the amount of exploration performed. The value $C_p = 1/\sqrt{2}$ was shown [11] to reward set in the range [0,1]. Each node maintains its number of visits and the rewards received from simulation.

5.2.2.2 BUCT

The selection formula for Bayesian UCT is:

$$BUCT: maximaze\, B_i = \mu_i + \sqrt{2\, lnN}\, \sigma_i \qquad (0)$$

μ_i : Mean of extremum (minimax) distribution P_i

N: Total trials of all arms

σ_i : Square root of variance of P_i

P_i: Probability distribution over its true expected reward value

At each node we maintain a probability distribution P_i over its true expected reward. Before any simulation, leaf nodes are initialized with conjugate prior distributions fitting the reward distributions. If our payoffs were 0/1 we choose to use *Dirichlet priors* which are described in Chapter 4.4.4. After the simulations are performed, the results are combined with the priors in order to produce a posterior distribution.
For the selection step, using this method, we produce the posterior distribution and calculate equation (15) by taking for each node the result from the updated Dirichlet distribution. The child node of the current node that maximizes the value B_i is chosen.

5.2.2.3 VPI

The strategy of VPI is to choose the action that maximizes:

$$E[q_{s,a}] + VPI(s,\alpha) \qquad (0)$$

where

$$VPI(s,\alpha) = \int_{-\infty}^{\infty} Gain_{s,\alpha}(x)\, Pr(\mu_{s,\alpha} = x)\, dx \qquad (0)$$

where the Gain computes what can be gained from learning the true value $\mu_{s,\alpha}^{¿}$ of $\mu_{s,\alpha}$:

$$Gain_{s,\alpha}\left(\mu_{s,a}^{¿}\right)=\begin{cases} E\left[\mu_{s,\alpha_2}\right]-\mu_{s,a_1}^{¿}, if\ a=a_1 \wedge \mu_{s,\alpha}^{¿}<E[\mu_{s,\alpha_1}] \\ E\left[\mu_{s,\alpha_2}\right]-\mu_{s,a_1}^{¿}, if\ a\neq a_1 \wedge \mu_{s,\alpha}^{¿}<E[\mu_{s,\alpha_1}] \\ 0, otherwise \end{cases} \quad (0)$$

The computation of the integral on equation (17) depends on how we represent our distributions over $\mu_{s,\alpha}$. In order to be able to take decisions, each node corresponding to a state *s*', maintains a distribution over the quality value $\mu_{s,\alpha}$ of having executed the action *α* that led from state *s* (parent) to *s*'. This value refers to the "quality" of following a path in the game tree downwards from *s*. When a decision needs to be taken at some node, the value of its children is calculated.

The distributions are updated by the use of appropriate conjugate priors. The distribution over $\mu_{s,\alpha}$ is a Dirichlet where each parameter α_i represents the frequency of seeing reward *i*, and these frequencies are updated through the observation of simulation results. For these calculations we first find the two best actions α_1 and α_2 from the set of available actions and we estimate the Q-value distributions using sampling on existing Dirichlet distributions. So, these Dirichlet are used in order to implement the sampling process used in Model-Based Bayesian Exploration [15].

We sample a vector *θ* according to the distribution *P(θ|ξ)* using this procedure:

We sample values $y_1,\ldots,y_K$ such that each $y_i\ \Gamma(a_i,1)$, where a_i is the concentration parameter in the index *i* of the α vector and Γ(κ, θ) is the Gamma distribution. Then after normalization we get the probability distribution [15]. Having these values, we calculate equations (17) and (18) and then we select the child-node that maximizes equation (16).

5.2.3 Expansion Step

The expansion step calculates the set of all available actions of our agent, in the given state, and creates children nodes according to the combinations of the selected actions. Each child node has only one action. These actions include:

1. Buying a City
2. Buying a Settlement
3. Buying a Road
4. Buying a Development Card
5. Empty Action

Note that action 1-2-3, may not represented as only one node. Each node represents an action and the positional placement, and all nodes have at least one child-node, an empty action node. Unlike [4] and [21], we introduced in the action space the empty action which makes the action tree more realistic, and also we allow buying development cards (Chapter 5.2.4.3). All available actions are based on the current resources of our agent. We include in the set of available actions, all the actions that can be made after trading some of the current resources of our agent with bank or ports. We cannot include trading with other players in this step because the negotiation step between players is before our MCTS algorithm.

5.2.4 Simulation Step

The simulation step is a highly important step of the algorithm because it estimates the reward of our agent. In this step, we would like to maintain the purity of MCTS and we have not implement any heuristics that affects the decisions on action selections like [4] [21] [22]. In the simulation step we start from a given state and we simulate the game by making fictional dice rolls, resource pay offs, player's move and reconstruct basic behaviors of the game in general, until we reach an end.

5.2.4.1 End of Simulation

First, we define two cut-off conditions for the simulation loop:

- **End-of-Game cut-off**: if one of the players acquires 10 or more Victory Points, we have a winner and the simulation returns the result Δ in order to proceed to the next step.
- **Round cut-off**: an average game takes approximate 20 round to end. We tested the algorithm with various depths (5,10,15,30). And in order to give some extra simulations at the end of the game we adjust the simulation depth based on this formula:

$$Simulation\,rounds = \begin{cases} max(3, c-r), if\ r \leq 20 \\ max\left(3, \frac{c}{2}\right), if\ r > 20 \end{cases} \qquad (0)$$

c: initial round cut-off {5,10,15,30}
r: current round of the game

With this formula we make sure that the search depth is always as deep as we planned and when we pass the 20th round on game (real round not simulated) we half our initial depth because in almost all cases, if the algorithm takes more rounds then it is erroneously guided to non-promising paths (Chapter 7.1.2 and 7.1.3). But in any case, we keep a minimum of 3 rounds depth simulation. This heuristic function does not interfere with the actual MCTS decisions, yet it boosts up its performance in simulations.

5.2.4.2 Simulation

For the simulation step, we keep a copy of players resources, pieces on board, special items and victory points. At each round a random generator is used to generate a number in the [0,1] and associate that value to a point on the CDF of a two-dice roll. This allows us to associate the number of each interval to a specific roll. For instance, a value $x \leq \frac{1}{36}$ corresponds to a dice roll of "2" and a value $\frac{3}{36} < x \leq \frac{6}{36}$ corresponds to a dice roll of "4". Figure (15) shows the probability for each dice outcome.

Dice Result	**Combinations**	**Probability**
2	1	$\frac{1}{36}$
3	2	$\frac{2}{36}$
4	3	$\frac{3}{36}$
5	4	$\frac{4}{36}$
6	5	$\frac{5}{36}$
7	6	$\frac{6}{36}$
8	5	$\frac{5}{36}$
9	4	$\frac{4}{36}$

10	3	$\frac{3}{36}$
11	2	$\frac{2}{36}$
12	1	$\frac{1}{36}$
Total	**36**	**1**

Figure 15: Dice Outcome Probabilities

According to the rules, after each dice roll, the players get resources according to their pieces on the board. For each player we look for pieces adjacent to hexes with a number matching the simulated dice result and we add the resources gained by the dice roll.

Then the player can consider taking actions. The available action set for each player is calculated, like expansion step, considering all legal moves and moves after trading with port or bank. The action selection is made based on the *Default Policy* (Chapter 5.2.4.3). Since each node of expansion step has only one action, each player can make several actions during his turn. The Victory Points of all players are constantly updated by all the actions they make according to the rules expect of development cards. When an ending condition is reached the simulation step returns a vector with the Victory Points of all players normalized to fit [0,1].

5.2.4.3 Default Policy

Default Policy dictates the actions that will be selected during the simulation. It has the same role in the simulation step as the Tree Policy in selection step. As mentioned before, we would like to keep the implementation free of heuristic evaluations in any policy, so the Default Policy is completely random.

After all the actions are listed the player does one action at time until empty action is selected (after each action, actions are recalculated), as the following pseudo algorithm describes.

Algorithm 3 Simulation

```
function SIMULATION
  While (ending conditions not reached)
      Roll dice and award resources
      do
          $\alpha \leftarrow$ CALCULATEMOVES($player_i$)
          move$\leftarrow$ Select random action $a_i$ from action set a
          makemove (move)
```

```
        While (move is not empty)
    Return reward vector
```

Figure 16: Simulation algorithm

5.2.4.3 Development Cards

Development cards are essential part of Settlers of Catan but they are left outside of MCTS algorithm [4][21][22] due to the fact that we can now draw a card and then return it to the deck. We propose a method to estimate their value on the simulation step. We used this method in our implementation.

There are 25 Development Cards at the beginning of the game and all players know how much remain at any time. From those 25 cards, 14 are Knight Cards, 5 Victory Points cards, 2 Monopoly Cards, 2 Road building Cards and 2 Year of plenty (explanation of chapter 2).

First we can calculate rather easy the remaining Soldier cards on the deck, by subtracting all the Knight cards that have been played and find how many knight cards a player needs to gain Largest Army award. Then we calculate the probability to gain a knight card and calculate our reward values based on following algorithm:

Algorithm 4 Development Card Value Estimation

```
  function DevelopmentCardEstimation
      P ← (remaining soldiers estimation)/(remaining cards)
      if (player has the Largest Army)
          if (player has same number of knights as some other player)
              reward ← P*0 + (P-1)*1
          else
              reward ← -1*P+ (P-1)*1
      else
          if (knights needed to get Largest Army > remaining Knights)
              reward ← -1*P+ (P-1)*1
          else if (needed Knights = 1)
              reward ← P*1 + (P-1)*1
          else if (needed Knights = 2)
              reward ← P*0.5 + (P-1)*1
          else
              reward ← (P-1)*1
      return reward
```

Figure 17 Development Card Value Estimation

It is easy to notice that the algorithm gives reward based on our positioning in the race to form the Largest Army. If we need more than 1 or 2 Knight Cards or we have the Largest Army and there are several Knight Cards in the deck the algorithm gives smaller reward. Concerning all the other cards, besides Knight, we give a fix value based on their probability against Knight cards. For playing Development Cards that reward resources, our agent cannot estimate their value through the tree, so we use the same methods as the JSettlers agent.

5.2.4 Backpropagation Step

Each selection method incorporates a different backpropagation method due to the variables used in the different calculation formulas.

5.2.4.1 UCT

The backpropagation method for UCT is quite straightforward. After the simulation, all we have to do is backpropagate the result of each player to the appropriate visited nodes. The result is defined by victory points of each player at the terminal node. For each of the visited nodes of the simulation, starting from the terminal node, we update all the parent nodes with the results. Either by increasing its existing reward X_j (equation 14) if it is node of our agent or either by decreasing its X_j if the action was made by another player.

5.2.4.2 BUCT

For the update of the values needed for calculation of the Bayesian UCT value (equation 15), we need to update the hyper parameter α of the Dirichlet distribution for each node. So according to the result Δ of the simulation, we update the appropriate index value of the vector α:

$$a_i = a_i + 1 \tag{0}$$

Essentially the vector α acts as counter, of how many times we have seen each reward.

5.2.4.3 VPI

The backpropagation procedure at VPI method is the same as BUCT since they both use Dirichlet distribution. So the hyper parameter vector α of Dirichlet priors need to be updated. The hyper-parameters a_i represent the frequency of seeing reward *i* and are used for sampling possible expected reward distribution. The update method is described in equation (15).

5.3 Opening Construction Strategy

Opening construction strategy refers to the placing of first two pairs of settlements and roads by each player at the beginning of the game. We used two different initial placements here.

5.3.1 JSettlers Initial Placement

The JSettlers opening building strategy estimates the building speed that can be acquired on each possible position and trying to guess the position at which the other players will place their settlements, so those spots will be excluded from planning.

5.3.2 European Research Council (ERC) Initial Placement - Strategic Conversation (STAC)

Dobre *et al.* [22] presented a method that combines knowledge extracted from a corpus of human play, with simulations of MCTS implementation for Settlers of Catan. They have promising results when their agent is compared against an agent that relies on exploration seeded with hand-coded heuristics or with human players. Although they also ran all their experiments on JSettlers, they removed completely the trade aspect of the game. Overall, their corpus of human playing the game ensures that the collected data contains a large variety of scenarios with a total of approximately 14000 state-action pairs.

5.4 Robber Strategy

In our implementation we use the built-in Robber Strategy provided by JSettlers. When a player roll "7" or play the Knight card, he must move the special piece Robber and place it on a hex. The hex with the Robber placed on, does not produce resources. Also, when a player moves the Robber he can steal one random resource card from a player adjacent to the hex he places the Robber on. To select where to place the Robber, we estimate the winning time for each player and find the player with the least time. The winning time for each player is calculated by estimating the building speed of each player, same way as (Chapter 3.2.2), combining with the current Victory Points of each player. Then we search for the best way to obstruct the player, by looking the building estimates for each piece of the game for each of his hexes. The hex with the larger total building speed for all the pieces is chosen as the best hex, because in that way we essentially "delay" that player's actions.

5.5 Monopoly Strategy

When a player plays this type of Development Card, he selects a type of resource and all the others players must give him all the resources of the chosen type they have. For our agent's strategy we use the JSettlers strategy which is the following:
The agent determines the trade ration which he can trade, as well as the available pieces and options for future actions and decides which resource is best to monopolize and if it is the best time to play the development card. The parameters that effect the decision when to play a development card, is the current resources our agent has, the resources enemy players have and the importance of buying a piece.

5.6 Road Building Strategy

This Development Card allows the player to build immediately 2 road pieces with no cost. Our Agent plays the card right after it is legal. The estimation of which road pieces he will build, uses the JSettlers strategy. The estimation is based on the Longest Road potential and the ability that their combination provides in order to build more settlements.

5.7 Discard Strategy

When a player rolls seven, then all players (including him) with more than 7 resources, must discard the half of them . If the number is odd then is rounded down (if a player has 9 resources, he must discard 4). All agents keep a building plan, of what they want to buy in the future. When a player needs to discard resources, he discards those which are not affecting as much as possible his building plan, or at least his first moves, if all resources are needed then he discards randomly.

5.8 Negotiation

Trading with ports and the bank, as well as negotiating with other players for resources is a crucial aspect of the game. Most of the times, a player does not have access to all the necessary resources in order to build pieces or buy development cards. Considering the importance of this feature, we gave our agent the ability to trade according to his resource set and even consider trading with other players, if he can be benefited from it. In out implementation we use the existing structures of JSettlers framework (Chapter 3.2.1) in order to consider and initiate trades.

5.8.1 Trading with Bank & Ports

As mentioned in the Game Rules (Chapter 2.2.2), a player can trade with the bank with ratio 4:1or trade with 3:1 or 2:1 with a port (based on type of the port) that has player's settlement or city built on. Trading with banks or ports does not require any particular logic from our agent. This type of trading expands the existing action set based on the current resources, by including all the action that can be made if the agent trades resources with either the bank or a port. For example, if the agent has the following resources {1 wood, 4 ore} he cannot buy any piece. But if he trades with the bank 4 ores : 1 clay, he is able to buy a road piece.

5.8.2 Trading with Players

Before initiate any negotiations our agent calculates his trading ration based on the ports he has. After the trading ratio for each resource is defined, our agent considers trading with other players. The trade offer is not addressed to all players, but instead the agent checks for the current state of the other players. If a player is close to victory, and we see that the trade we are proposing can benefit him, we do not include him in the offer. To determine if the trade is of benefit for another player, we use the existing structures (Chapter 3.2.1) and the specific state of the game to determine their options for future plans. Similar logic is used when we want to determine if we will accept an offer from another player.

6 Results

For each selection method between UCT, BUCT and VPI, we tested our agent's performance against 3 Random players and against 3 JSettlers players. With 4 different simulation depths and for each case we test our agent either with or without STAC initial placement. Each setting was tested on 100 seeded experiments.
Random players are implemented in a way that all available actions are listed and they select a random legal subset of those actions.

For making a point of reference, we experiment with one JSettlers agent against three random players and the agent scores 90% win.

AVG Victory Points	9,31
AVG Place	1,28
AVG Point System	2,71
Win Ratio	90%
AVG Points From Winner	0,34
Victory Point Standard Deviation	3,809
AVG DEV from mean in each game	1,9125
Above mean of Victory Points	92%
Longest Road	55,00%
Largest Army	52,53%

Figure 18: JSettlers vs. 3 Random

6.1 Metrics

We tested our agent's performance based on the following metrics. Average Victory Points, Average Place, Average Point System, win ratio, Average points from winner, Victory points standard deviation, Average Deviation from mean in each game, Above mean of Victory points, Average enemy players Victory points, Longest road acquisition and largest army acquisition.

6.1.1 Average Victory Points

For this metric we calculate the average VP our agent gathered. This metric however is not sufficient because if a game takes more rounds than usual, all players will score higher VP. Regardless, it is helpful for cross-validating the methods since we are using seeds.

$$\frac{\sum_i R}{I} \qquad (0)$$

R : the Victory Points our agent took on game *i*
I : the total number of games

6.1.2 Average Place

For each setting we calculate the average place that our agent won. With this metric we evaluate if our agent is scoring well against all other players by acquiring good positions over all games.

$$\frac{\sum_i P}{I} \qquad (0)$$

P : the place that our agent took on game *i*
I : the total number of games

6.1.3 Average Point System

We implemented a metric to get a more accurate average place estimate for our agent. At (6.1.2) we calculated the average place for our agent but this is method does not take into consideration ties between players which is very common in Settler of Catan. For example, if our agent and another player have the least VP at the end, it will award both the third place. Our formula awards our agent with 1 point for each player with lower VP than our agent.

$$\frac{\sum_i count(P)}{I} \qquad (0)$$

P : the number of players our agent over passed on game *i*
I : the total number of games

6.1.4 Win Ratio

This method is considerate to be the most accurate since it is indicates how many times our agent won the game.

$$\frac{W}{I} \qquad (0)$$

W : the number of times our agent won the game
I : the total number of games

6.1.5 Average Points from Winner

This method measures the Victory Points our agent left behind from the winner of the game. With this method, we can measure by how much our agent was outplayed by the winner. This method is used only in games which our agent lost.

$$\frac{\sum_i P(VP)_i - A(VP)_i}{I} \qquad (0)$$

P(VP) : the Victory Points of winner on game *i*
A(VP) : the Victory Points of our agent on game *i*
I : the total number of games

6.1.6 Victory Points Standard Deviation

This method calculates the Standard Deviation of our agent's Victory Points.

$$\sigma = \sqrt{\frac{\sum_i (VP_i - \acute{VP})^2}{I}} \qquad (0)$$

VP : the Victory Points of our agent on game *i*
$\acute{VP}$: the mean of Victory Points over all games
I : the total number of games

6.1.7 Average Deviation from Mean in each game

This method calculates the Standard Deviation of our agent's Victory Points from the mean in each game.

$$\sigma = \sqrt{\frac{\sum_i (VP_i - \acute{VP}_i)^2}{I}} \qquad (0)$$

VP : the Victory Points of our agent on game *i*
$\acute{VP}$: the mean of Victory Points on game *i*
I : the total number of games

6.1.8 Above Mean of Victory Points ratio

The metric demonstrates how many times our agent is above the mean of Victory Points of all players in each game.

$$\frac{X_i}{N} \qquad (0)$$

X$_i$: the number of times our agent was above the mean of Victory Points at the game
N : the total number of games

6.1.9 Average Opponent Victory Points

For this metric we calculate the average VP of all players but our agent.

$$\frac{\sum_i \acute{VP}_i}{I} \qquad (0)$$

$\acute{VP}$: the mean of Victory Points on game *i* of all players expect ours
I : the total number of games

6.1.10 Longest Road Acquisition Ratio

This metric demonstrates the ratio of Longest Road acquisition from our player

$$\frac{W}{I} \qquad (0)$$

W : the number of times our agent end the game with the Longest Road
I : the total number of games

6.1.11 Largest Army Acquisition Ratio

This metric demonstrates the ratio of Largest Army acquisition from our player

$$\frac{W}{I} \qquad (0)$$

W : the number of times our agent end the game with the Largest Army
I : the total number of games

6.2 Results: Comparisons against JSettlers

Simulation Depth	30		15		10		5	
Method	UCT	UCT +S	UCT	UCT+ S	UCT	UCT+ S	UCT	UCT+ S
AVG Victory Points	3,77	4,54	4,38	**4,58**	4,05	4,30	4,47	4,38
AVG Place	3,68	3,38	3,51	3,52	3,59	3,49	**3,32**	3,45
AVG Point System	0,23	0,33	0,25	0,38	0,31	0,39	**0,61**	0,44
WIN Ratio	3,00%	4,00%	3,00%	4,00%	7,00%	**9,00%**	6,00%	7,00%
AVG Points From Winner	6,36	5,54	5,69	**5,50**	6,11	5,86	5,67	5,79
Victory Point Standard Deviation	**1,25**	1,32	1,55	1,46	1,43	1,80	2,07	1,80
AVG DEV from mean in each game	3,06	**2,40**	2,60	2,59	3,10	3,01	2,86	2,88
Above mean of Victory Points	4,00%	6,00%	5,00%	8,00%	7,00%	**13,00%**	12,00%	10,00%
Average Opponent VP	7,64	7,52	7,64	7,76	7,60	**7,35**	7,59	7,44
Longest Road	17,00%	17,00%	19,00%	**40,40%**	18,00%	23,00%	17,00%	25,00%
Largest Army	13,13%	20,20%	22,22%	20,41%	12,12%	22,22%	**27,27%**	23,23%

Figure 19: UCT vs. 3 JSettlers

Simulation Depth	30		15		10		5	
Method	BUCT	BUCT +S	BUCT	BUCT +S	BUCT	BUCT +S	BUCT	BUCT +S
AVG Victory Points	3,51	3,82	**5,59**	5,23	4,42	4,50	4,23	4,26
AVG Place	3,74	3,70	**3,20**	3,29	3,39	3,60	3,58	3,43
AVG Point System	0,14	0,21	0,53	0,43	0,41	0,40	0,30	**0,45**

Win Ratio	3,00%	3,00%	5,00%	3,00%	11,00%	**12,00%**	6,00%	7,00%
AVG Points from Winner	6,69	6,22	**4,49**	4,87	5,58	5,65	5,89	5,82
VP Standard Deviation	**1,12**	1,32	1,88	1,62	1,89	1,93	1,51	1,71
AVG DEV from mean each game	3,34	3,04	**2,02**	2,12	2,91	3,10	2,97	2,79
Above mean of Victory Points	4,00%	3,00%	**14,00%**	10,00%	11,00%	12,00%	6,00%	7,00%
Average Opponent VP	7,64	7,60	7,72	7,68	**7,29**	7,57	7,63	7,43
Longest Road	16,00%	**28,00%**	23,00%	19,00%	18,00%	27,00%	21,78%	9,00%
Largest Army	20,20%	7,07%	**46,46%**	48,48%	33,33%	27,27%	22,00%	40,40%

Figure 20: BUCT vs. 3 JSettlers

Simulation Depth	**30**		**15**		**10**		**5**	
Method	**VPI**	**VPI+S**	**VPI**	**VPI+S**	**VPI**	**VPI+S**	**VPI**	**VPI+S**
AVG Victory Points	4,62	4,47	4,76	4,54	4,80	**4,96**	4,46	4,68
AVG Place	3,35	3,39	3,42	3,36	3,24	**3,20**	3,40	3,30
AVG Point System	0,49	0,45	0,49	0,53	0,65	**0,66**	0,48	0,54
Win Ratio	11,00%	11,00%	12,00%	13,00%	**17,00%**	16,00%	14,00%	15,00%
AVG Points From Winner	5,45	5,63	5,32	5,60	5,33	**5,13**	5,58	5,37
Victory Point Standard Deviation	1,84	**1,82**	2,03	2,06	2,31	1,96	2,19	2,22
AVG DEV from mean in each game	**2,91**	3,01	3,01	3,06	3,11	2,94	3,24	3,03
Above mean of Victory Points	13,00%	12,00%	17,00%	14,00%	17,00%	**18,00%**	16,00%	17,00%
Average Opponent VP	7,45	7,49	7,67	7,37	**7,24**	7,36	7,53	7,45
Longest Road	**34,00%**	**34,00%**	25,25%	19,00%	21,00%	33,00%	25,00%	28,00%
Largest Army	29,29%	31,31%	28,57%	33,33%	**36,36%**	31,31%	34,34%	31,31%

Figure 21: VPI vs. 3 JSettlers

Figures (19) (20) (21) demonstrate the results concerning the standard Monte Carlo Tree Search algorithm with the use of UCT (19) BUCT (20) and VPI (21) in the selection step, against 3 JSettlers. We can see that performance of our agent is rather low. UCT won 9% of the games, BUCT 14% and VPI 17%, at simulation depth of 10

rounds. Throughout the experiments we observe that the critical simulation depth lies on 10 rounds because, it has enough round to build the tree ahead but the same time the algorithm is not exhausted into building long paths and give time for more simulations to be made. The average of the Victory Points that our agent scored which is half of the goal (10 VP to win when our agent scores 5VP) and the average place is between the third and the last place at the end of the game, indicates that our pure Monte Carlo approach needs to be more competitive against the JSettlers agent. The pointing system as well as the Victory Points difference from the winner in each game, gives a more clear view of what we mentioned before. Lastly, we notice that our agent deviation is rather good since it only diverges by a couple of Victory Points which is understandable given the nature of "Settlers of Catan" .
Regarding the simulations in each method, UCT simulates [3000,10000] games since it is the less complex of all. BUCT has a slightly lower performance simulating [2000,9500] games, the upper bound drops only by 500 since the pick of simulations lies in rounds close to the end of game. VPI is the method that consumes the most computational budget of all three methods, as a result, we notice a dramatically big drop of simulations to [500,3000], yet produces the best score.

6.3 Results: Behavior Against Random Opponents

Simulation Depth	**30**		**15**		**10**		**5**		**-**
Method	**UCT**	**UCT+S**	**UCT**	**UCT +S**	**UCT**	**UCT +S**	**UCT**	**UCT +S**	**JSettlers**
AVG Victory Points	8,87	8,86	8,65	8,68	8,93	**8,97**	8,87	8,57	*9,31*
AVG Place	1,51	**1,49**	1,65	1,65	1,55	1,52	1,59	1,73	*1,28*
AVG Point System	2,32	**2,43**	2,25	2,27	2,36	2,41	2,31	2,19	*2,71*
Win Ratio	63,00 %	67,00 %	68,00%	68,00%	72,00%	**74,00%**	71,00%	73,00%	*90%*
AVG Points from Winner	1,32	1,14	1,40	1,37	1,12	**1,07**	1,21	1,53	*0,34*
Victory Point Standard Deviation	**1,46**	1,54	1,98	1,95	1,69	1,65	1,75	2,23	*3,809*
AVG DEV from mean in each game	**2,40**	2,58	2,91	3,00	3,02	3,11	2,92	3,02	*1,9125*
Above mean of Victory Points	79,00 %	**85,00 %**	74,00%	74,00%	78,00%	80,00%	76,00%	73,00%	*92%*
Average Opponent VP	6,16	6,00	6,16	6,05	6,01	**5,89**	6,13	6,44	-

Longest Road	29,00%	17,00%	55,00%	55,00%	**57,00%**	**57,00%**	56,00%	51,00%	*55,00%*
Largest Army	40,40%	**49,49%**	24,24%	29,29%	32,32%	32,32%	34,34%	32,32%	*52,53%*

Figure 22: UCT vs. 3 Random

Simulation Depth	**30**		**15**		**10**		**5**		**-**
Method	**BUCT**	**BUCT+S**	**BUCT**	**BUCT+S**	**BUCT**	**BUCT+S**	**BUCT**	**BUCT+S**	**JSettlers**
AVG Victory Points	8,93	8,81	8,97	9,11	9,13	9,12	9,13	**9,24**	*9,31*
AVG Place	1,43	1,44	1,39	1,39	**1,33**	1,38	1,34	1,34	*1,28*
AVG Point System	2,50	2,48	2,54	2,57	**2,62**	2,58	2,60	2,59	*2,71*
WIN Ratio	70,00%	71,00%	73,00%	75,00%	**77,00%**	**77,00%**	75,00%	76,00%	*90%*
AVG Points From Winner	1,07	1,19	1,03	0,89	0,87	0,88	0,87	**0,76**	*0,34*
VP Standard Deviation	1,51	1,69	1,51	1,34	1,34	1,36	1,31	**1,16**	*3,809*
AVG DEV from mean each game	3,07	2,93	2,97	2,93	2,93	2,95	**2,86**	3,02	*1,9125*
Above mean of Victory Points	85,00%	84,00%	85,00%	**89,00%**	88,00%	87,00%	**89,00%**	88,00%	*92%*
Average Opponent VP	5,31	5,50	5,63	5,61	5,61	5,70	5,60	5,51	-
Longest Road	47,00%	44,00%	42,00%	52,00%	49,00%	42,00%	44,00%	**54,00%**	*55,00%*
Largest Army	**54,55%**	50,51%	51,52%	41,41%	44,44%	45,45%	41,41%	42,42%	*52,53%*

Figure 23: BUCT vs. 3 Random

Simulation Depth	**30**		**15**		**10**		**5**		**-**
Method	**VPI**	**VPI+S**	**VPI**	**VPI+S**	**VPI**	**VPI+S**	**VPI**	**VPI+S**	**JSettlers**
AVG Victory Points	9,35	9,46	9,50	9,40	9,54	9,46	9,34	**9,63**	*9,31*
AVG Place	1,23	1,21	1,23	1,23	1,16	1,23	1,23	**1,17**	*1,28*
AVG Point System	2,72	2,74	2,74	2,73	2,81	2,75	2,76	**2,77**	*2,71*
Win Ratio	86,00%	85,00%	85,00%	84,00%	87,00%	**88,00%**	87,00%	87,00%	*90%*
AVG Points From Winner	0,63	0,54	0,50	0,60	0,46	0,54	0,66	**0,37**	*0,34*
Victory Point Standard Deviation	1,12	0,92	0,85	1,01	0,80	0,95	1,15	**0,64**	*3,809*
AVG DEV from mean in each game	3,33	3,24	3,15	3,19	**3,12**	3,25	3,38	3,19	*1,9125*

Above mean of Victory Points	88,00%	92,00%	90,00%	91,00%	93,00%	90,00%	91,00%	**96,00%**	*92%*
Average Opponent VP	5,35	5,38	5,46	5,43	5,59	5,44	**5,26**	5,51	-
Longest Road	57,00%	53,00%	48,00%	50,00%	48,00%	**63,00%**	52,00%	51,00%	*55,00%*
Largest Army	**52,53%**	48,48%	42,42%	44,44%	50,51%	48,48%	48,48%	51,52%	*52,53%*

Figure 24: VPI vs. 3 Random

Figures (22) (23) (24) represent the results of Monte Carlo Tree Search algorithm with the use of UCT (22) BUCT (23) and VPI (24) in the selection step, against 3 players that choose randomly a subset of their legal actions per round. Also, the last column shows the results of JSettlers agent against 3 Random opponents. An optimal outcome for us would be if an agent could win more than 95% of the games against random players.

As the results indicate UCT method is scoring low since the best outcome (74%) was produced with simulating depth of 10 rounds and the use of STAC initial placement. The diversion towards random of this method is highly notable in this results set, since the agent is unable to score more than 9 Victory Points on average, meaning that the agent was outplayed by 2 points.

The Bayesian UCT method offers an improvement at the previous method since it breaks the 9 Victory Points average and scoring better outcomes, by winning 77% of the games, but this also method keeps the aberration of UCT.

The most promising results come from VPI witch wins up to 88% of the games using the initial placement of STAC. The method is more stable since, at the games that did not win was only 1 Victory Point behind the winner. Although VPI scores close to the JSettlers agent, the score is inferior than our original goal.

6.4 Pitting Our Methods Against Each Other

We tested all strategies of MCTS in a single game with no JSettlers opponent. The game was played only by three of our agents. We tested the results with different simulation depths.

Method	**UCT**	**BCT**	**VPI**	**UCT+STAC**	**BUCT+STAC**	**VPI+STAC**
AVG Victory Points	6,69	6,62	**8,99**	7,03	6,82	**8,99**
AVG Place	2,25	2,25	**1,39**	2,2	2,27	1,4
AVG Point System	2,36	2,36	1,39	2,29	2,39	1,45
Win Ratio	16,00%	9,00%	**75,00%**	17,00%	11,00%	72,00%

AVG Points From Winner	3,31	3,38	**1,01**	2,97	3,18	**1,01**
Victory Point Standard Deviation	1,86	1,90	1,52	1,79	1,71	**1,46**
AVG DEV from mean in each game	1,73	1,50	2,32	1,57	**1,43**	2,17
Above mean of Victory Points	25,00 %	25,00%	**81,00 %**	30,00%	22,00%	76,00%
Longest Road	19,00 %	35,00%	46,00%	34,00%	18,00%	**48,00%**
Largest Army	22,00 %	30,00%	48,00%	25,00%	25,00%	**50,00%**

Figure 25: UCT vs. BUCT vs. VPI (Depth = 30)

Method	UCT	BCT	VPI	UCT+STAC	BUCT+STAC	VPI+STAC
AVG Victory Points	6,8	7,3	**9,01**	7,27	6,49	8,97
AVG Place	2,31	2,12	**1,38**	2,12	2,35	**1,38**
AVG Point System	2,48	2,29	1,42	2,26	2,48	**1,41**
Win Ratio	11,00%	15,00%	**74,00%**	15,00%	13,00%	72,00%
AVG Points From Winner	3,2	2,7	**0,99**	2,73	3,51	1,03
Victory Point Standard Deviation	1,82	1,67	**1,47**	1,50	2,01	1,49
AVG DEV from mean in each game	1,54	**1,37**	2,01	1,44	1,85	2,09
Above mean of Victory Points	17,00%	28,00%	76,00%	30,00%	29,00%	**80,00%**
Longest Road	25,00%	27,00%	**48,00%**	36,00%	25,00%	39,00%
Largest Army	23,00%	27,00%	**50,00%**	33,00%	24,00%	43,00%

Figure 26: UCT vs. BUCT vs. VPI (Depth = 15)

Method	UCT	BCT	VPI	UCT+STAC	BUCT+STAC	VPI+STAC
AVG Victory Points	6,49	6,9	9,12	6,56	7,3	**9,17**
AVG Place	2,33	2,18	1,36	2,4	2,12	**1,33**
AVG Point System	2,44	2,31	1,38	2,53	2,24	**1,38**
Win Ratio	9,00%	18,00%	73,00%	9,00%	17,00%	**74,00%**
AVG Points From Winner	3,51	3,1	0,88	3,44	2,7	**0,83**
Victory Point Standard Deviation	1,81	1,89	1,28	1,63	1,63	**1,23**
AVG DEV from mean in each game	1,58	1,68	2,15	1,63	**1,40**	1,99
Above mean of Victory Points	22,00%	30,00%	77,00%	18,00%	30,00%	**80,00%**
Longest Road	26,00%	28,00%	**46,00%**	24,00%	37,00%	39,00%
Largest Army	23,00%	28,00%	49,00%	20,00%	25,00%	**55,00%**

Figure 27: UCT vs. BUCT vs. VPI (Depth = 10)

Method	UCT	BCT	VPI	UCT+STAC	BUCT+STAC	VPI+STAC
AVG Victory Points	6,61	7,06	**8,94**	7,16	7,05	8,84
AVG Place	2,29	2,2	**1,35**	2,23	2,16	1,43
AVG Point System	2,43	2,31	**1,42**	2,38	2,31	1,49
Win Ratio	12,00%	17,00%	**71,00%**	14,00%	15,00%	**71,00%**
AVG Points From Winner	3,39	2,94	**1,06**	2,84	2,95	1,16
Victory Point Standard Deviation	1,82	1,65	**1,51**	1,43	1,70	1,66
AVG DEV from mean in each game	1,58	1,56	**2,04**	1,48	1,47	2,00
Above mean of Victory Points	20,00%	25,00%	**76,00%**	21,00%	25,00%	75,00%

Longest Road	28,00%	30,00%	42,00%	33,00%	21,00%	**46,00%**
Largest Army	29,00%	22,00%	**49,00%**	30,00%	28,00%	42,00%

Figure 28: UCT vs. BUCT vs. VPI (Depth = 5)

The method comparison confirms the previous results, VPI scored the best results among all the MCTS methods, since the absent of JSettlers agent, benefits VPI method more than all the others. The VPI scores up to 75% win ratio, significantly higher than the baseline which would win 33% of the games since it is a three players set up. Also, the VPI method was always close to the winner only 1 Victory Point on average behind. UCT and BUCT scores are rather similar with BUCT exceeding a little. The advantages and disadvantages of each depth are shared between the methods, but we can notice that VPI method shows a more stable behavior because it does not rely to simulations count as much as the remaining methods.

7 Conclusions

At the last chapter of this thesis, we present our conclusion concerning the implementation of Monte Carlo Tree Search algorithm is such a complex domain and we provide some ideas concerning our future work.

7.1 Summary

Our intention was to build a strong agent using Monte Carlo Tree Search algorithm without any heuristics in the non-deterministic, partially observable, multi-player game "Settlers of Catan" obeying the complete rules-set. For this task we tested three different methods for the Tree Policy and four Simulation Depths.
Our algorithm was outplayed from the JSettlers agents since it could not reach 25% win ratio when competing against 3 JSettlers agents. We believe this is due mainly to lack of domain knowledge while JSettlers uses a series of domain specific heuristics.

Our experiments among the three MCTS strategies indicated that that the reward based on distribution provided by VPI is the most promising, since it out-played easily all the others bandit-based methods and was the most competitive, against JSettlers agents reaching 17% win ratio when a simulation depth of 10 rounds was used. There is place for improvement in all methods and we believe that a hybrid algorithm would be able to score better than JSettlers agents.

MCTS and Domain Knowledge

We wanted to test the performance of our agent using nothing but MCTS. So in each step of the game, our agent was only aware of all his legal action which initially they were all equal. From there, our agent needed to figure out what was the best action (or set of actions) that will benefit him the most. Because of that, we noticed the following:

- **MCTS does not have strategy**. MCTS algorithm runs on each turn of our agent with no prior knowledge, making the agent have no consistency in strategy. For example, we observe the agent building roads to head for a new position to place settlement and when he does, instead of placing new settlement buys a Development Card or builds roads towards another spot.
- **Set of actions is vast**. Each round our agent has a big amount of actions available to him, but more than half of them have no potential. Most of these action can be "left out" easily with few heuristics as we suggest to section (7.2).
- **MCTS does not make moves to protect his future progress**. In the game Settlers of Catan a player sometimes should make moves to either block or

disturb plans of enemy players. Since blocking enemies will now increase the calculated reward, our agent considerate moves like this as "bad moves".

- **Trade manipulation**. MCTS taking into consideration only trading with bank or ports. Because of the nature of the algorithm, MCTS plans moves only based on current available set and not some positional set after trading with other players. Although, if we make some trades before or after the algorithm we observe an increase of "right moves".

Random Selection

Our agent for Default Policy (5.2.4.3) uses random selection. Making the agent calculate scenarios that will not be played. Losing significant amount of computational budget that could have been used to simulation. An improvement on this Policy will benefit crucial our agent.

Random Playouts

The MCTS algorithm lies along with numerous simulations. The Bayesian aspect of the game interferes severely with the effectiveness of the algorithm since most of the scenarios will not happen. Machine learning algorithms or simple heuristics methods could restrict the algorithm when it goes to improbable scenarios.

Depth versus Simulations

As we can see at the results, there is a trade off we need to make between simulation depth and simulation number. As the results indicate the best set of actions is Simulation Depth of 10 rounds. Because, simulation depth bigger than 10 rounds will make simulations number to drop, with no value on estimations. Simulating the game many rounds ahead is altering the reward estimation of our agent since they are big variations on game states due to the Bayesian elements of the game. On the other hand, fewer rounds than 10 will not allow the agent to actually plan ahead confined only by a few moves. Of course, the more simulation the agent makes the more effective the algorithm will be. However, the improvement of the agent's performance by increasing its simulations it is not commensurate. Since in some experiment we are currently running, doubling the simulation offers only a minor growth to our scores.

ERC Initial Placement (STAC)

Dobre *et al.* [22] initial placement makes our agent perform better increasing in most cases the outcome of our agent. As they have already proven the initial placement gives advantage to our agent and our experiments confirm that. Since in almost all set ups, STAC initial placement gave a small boost up to our scores.

7.2 Future Work

In this final section, we propose some ideas for future work.

Improve MCTS with Domain Knowledge

As proven from the results, MCTS algorithm with no domain knowledge is outplayed by a hand-heuristic agent. So we need to include to MCTS agent domain knowledge to either increase his simulation performance or enchant the selection process.
For example, it is clear that the VPI is an optimistic assessment of the value of performing α; by performing α once, we do not get perfect information about it, but only one more training instance. Thus, we might consider weighing the VPI estimate by some constant.

Making a plan

As mentioned before, MCTS does not have any plan consistency through rounds. By making a plan and each round updating it based on current state of the game, might give our agent a better perspective of the game. This plan could be just to rank the top pieces.

Negotiation

We discussed the power that trading has in the game of Settlers of Catan and we pointed out its importance. The first step is to enhance our Negotiation scheme, using tools from Machine Learning, Game Theory and Argumentation Techniques .

JSettlers Framework

The potential of MCTS is limited because of the JSettlers framework. Since it is a internet based application the computational budget provided to our agent is significant lower than a standalone Java application. Removing communication infrastructure will allows us to test with increased number of simulations. Or maybe complete changing domain might boost the performance of our agent.

Understating Enemy Strategies

Playing a game such Settlers of Catan, a player does not need only to aim to maximize his results, but also counter enemies strategies and prevent them for expanding their plans. Using the work of [23][24][25] we can estimate the strategy of our enemies and give more importance to countering moves.

8 References

[1] An overview of Catan Games: www.catan.com/board-games.

[2] Game rules & almanac 3/4 players. www.catan.com/files/downloads/ catan_5th_ed_rules_eng_150303.pdf

[3] Michael Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. *2004.*

[4] István Szita, Guillaume Chaslot, Pieter Spronck. Monte-Carlo Tree Search in Settlers of Catan. *In Proceeding of the 12th International Conference on Advances in Computer Games, AVG'09. Springer-Verlag, 2010.*

[5] Robert Shaun Thomas. Real-time Decision Making for Adversarial Environments Using a Plan-based Heuristic. *PhD thesis, Northwestern University, 2003.*

[6] R. Fisher and W. Ury. Getting to Yes. *Penguin Books.*

[7] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diago Perez, Spyridon Samothrakis and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games.*

[8] Guillaume Chaslot, Sander Bakkes, Istvan Szita and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In *AIID*E. The AAAI Press.

[9] Peter Auer, Nicolo Cesa-Bianchi, Paul Ficher. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning, 47(2-3):235-256 Kluwer Academic Publishers.*

[10] Levente Kocsis and Csaba Szepesvari. Bandit based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning, ECML'06, pages 282-293. Springer-Verlag, 2006.*

[11] Levente Kocsis, Csaba Szepesvari, Jan Willemson. Improved Monte-Carlo Search. *Technical Report 1, University of Tartu, 2006.*

[12] Sylvain Gelly, David Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *INRIA, 2006.*

[13] Gareth M. J. Williams. Determining Game Quality Through UCT Tree Shape Analysis. *MSc thesis. Imperial College London.2010.*

[14] Gerald Tesauro, VT Rajan, and Richard Segal. Bayesian Inference in Monte-Carlo Tree Search. *2012.*

[15] Richard Dearden, Nir Friedman, Stuart Russell. Bayesian Q-learning. In *Proceeding of the 15th National/ 10th Conference on Artificial Intelligence/ Innovative Applications on Artificial Intelligence, AAAI'98/IAAI'98, pages 761-768. American Association for Artificial Intelligence, 1998.*

[16] Richard Dearden, Nir Friedman, David . Model based Bayesian Exploration. In *Proceeding of the Fifteenth Conference on Uncertainty in Artificial Intelligence, UAI'99, pages 150-159. Morgan Kaufmann Publishers Inc., 1999.*

[17] Jeremy Wyatt. Exploration and Inference in Learning from Reinforcement. *PhD thesis, University of Endiburgh, 1997.*

[18] Georgios Chalkiadakis, Craig Boutilier. Sequentially Optimal Repeated Coalition Formation under Uncertainty. *Autonomous Agents and Multi-Agent Systems, 24(3):441-484, 2012.*

[19] Konstantinos Babas, Georgios Chalkiadakis, Evangelos Tripolitakis. You Are What You Consume: A Bayesian Method for Personalized Recommendations. In *Proceedings of the 7th ACM Conference on Recommender Systems, RecSys'13, pages 221-228. ACM, 2013.*

[20] W. T. Luke Teacy, Georgios Chalkiadakis, Alex Rogers and Nicholas R. Jennings. Sequential Decision Making with Untrustworthy Service Providers. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, pages 755-762, May 2008.*

[21] Konstantinos Panagiotis Panousis. Real-time Planning and Learning in the "Settlers of Catan" Strategy Game. *Diploma thesis. Technical University of Crete. 2014.*

[22] Mihai Sorin Dobre, Alex Lascarides. Online learning and mining human play in complex games. In *Proceedings of the IEEE Conference on Computational Intelligence in Games, 2015.*

[23] Ioannis Efstathiou, Oliver Joseph Lemon. Learning Better Trading Dialogue Policies by Inferring Opponent Preferences. In *Proceedings of AAMAS, 2016.*

[24] Ioannis Efstathiou, Oliver Joseph Lemon. Learning to manage risk in non-cooperative dialogues. In *22nd European Conference on Artificial Intelligence, 2016.*

[25] Ioannis Efstathiou, Oliver Joseph Lemon. Learning non-cooperative dialogue policies to beat opponent models: " The good, the bad and the ugly ". In *Proceedings of the 19th Workshop on the Semantics and Pragmatics of Dialogue, 2015.*

Appendix A

Background

Here we provide an overview of the basic concepts related to the methods we go throughout this thesis.

Multinomial Estimation Problem and Dirichlet Priors

Let X be a random variable that can take K possible values. Given training set D, which contains outcomes of N independent draws $x^1,\dots,x^N$ of X from an unknown multinomial Distribution P^*. Finding a good approximation of P^* constitutes the multinomial estimation problem. This problem can also be stated as predicting the outcome x^{N+1} given $x^1,\dots,x^N$.

The Bayesian estimate, given a prior distribution over the possible multinomial distributions is:

$$P\left(x^{N+1}\middle|x^1,\dots,x^N,\xi\right)=\int P\left(x^{N+1}\middle|\theta,\xi\right)P\left(\theta\middle|x^1,\dots,x^N,\xi\right)d\theta \quad (0)$$

where

$$P\left(\theta\middle|x^1,\dots,x^N,\xi\right)\propto P(\theta \vee \xi)\prod_i \theta_i^{N_i} \quad (0)$$

and $\theta = (\theta_1, ..., \theta_K)$ are the possible values over the probabilities $P^*(1), ..., P^*(K)$ and ξ is a variable containing assumptions over the domain. We chose the Dirichlet distribution as a prior distribution for each node. Dirichlet distribution is a parametric family that is conjugate prior to the multinomial/categorical distribution [14].

A Dirichlet prior consists of two parameters:

- K ≥ 2 , the number of rival events and
- $a_1,a_2,\dots,a_K$ the concentration parameters, where a_i >0

The Dirichlet distribution is a generalization of Beta Distribution and is a distribution over multinomial. It has probability density function:

$$a_i$$

$$\sum_i ¿$$

$$¿$$

$$\Gamma ¿$$

$$p\left(P=\{p_i\}\middle|a_i\right)=\frac{\prod_i(\Gamma\left(a_i\right))}{¿} \qquad (0)$$

The initial prediction for each value of the random variable X, given a Dirichlet prior, is [14]:

$$P(X=i|\xi)=\int\theta_i P(\vartheta|\xi)\,d\vartheta=\frac{a_i}{\sum_j a_j} \qquad (0)$$

If the prior is a Dirichlet prior with concentration parameters $a_1, a_2, \ldots, a_K$ and N_i is the number of occurrences of the symbol *i* in the training data, then the posterior is also a Dirichlet with concentration parameters $a_1+N_1, \ldots, a_K+N_K$ and thus the prediction for X^{N+1} is :

$$P\left(X^{N+1}=j\middle|x^1,\ldots,x^N\right)=\frac{a_j+N_j}{\sum_j(a_j+N_J)} \qquad (0)$$