TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

IMPLEMENTATION OF A SOFTWARE SERVER AND OF AN AUTONOMOUS AGENT FOR THE COOPERATIVE STRATEGIC GAME "PANDEMIC"

DIPLOMA THESIS

Konstantinos Voloudakis

COMMITTEE:

GEORGIOS CHALKIADAKIS, Associate Professor (ECE)

MICHAIL G. LAGOUDAKIS, Associate Professor (ECE)

VASILIS SAMOLADAS, Associate Professor (ECE)

Chania, April 2021

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΥΛΟΠΟΙΗΣΗ ΕΞΥΠΗΡΕΤΗΤΗ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΑΥΤΟΝΟΜΟΥ ΠΡΑΚΤΟΡΑ ΓΙΑ ΤΟ ΣΥΝΕΡΓΑΤΙΚΟ ΠΑΙΓΝΙΟ “PANDEMIC”

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Βολουδάκης

ΕΠΙΤΡΟΠΗ:

ΓΕΩΡΓΙΟΣ ΧΑΛΚΙΑΔΑΚΗΣ, Αναπληρωτής Καθηγητής (ΗΜΜΥ)

ΜΙΧΑΗΛ Γ. ΛΑΓΟΥΔΑΚΗΣ, Αναπληρωτής Καθηγητής (ΗΜΜΥ)

ΒΑΣΙΛΗΣ ΣΑΜΟΛΑΔΑΣ, Αναπληρωτής Καθηγητής (ΗΜΜΥ)

Χανιά, Απρίλιος 2021

ABSTRACT

Artificial Intelligence (AI) has in our times made considerable progress, and manifests itself in various devices and infrastructure used in our everyday lives. It has thus become even more important than ever to measure AI's progress via pitting intelligent agents against humans in board games, following a decades-long tradition. Still, testing AI in board games that require the players' cooperation without any element of competition among them during a game instance, is extremely rare. At the same time, this is important since the ability to cooperate effectively is a key component of intelligence. Against this background, in this thesis we develop the necessary server infrastructure that enables the participation of autonomous agents in the multiagent cooperative board game ``Pandemic''; along with seven autonomous agents that can compete in the game, each of which incorporating different strategies that we also devised. Our server has already been used in the context of an AI-related course at the Technical University of Crete, and can provide the basis for a richer infrastructure that can be potentially employed for holding an international challenge for Pandemic-playing autonomous agents. We complement our work with a preliminary experimental evaluation of our agents' strategies, drawing lessons on their effectiveness in Pandemic domains of varying difficulty.

ΠΕΡΙΛΗΨΗ

Η Τεχνητή Νοημοσύνη (ΤΝ) έχει σημειώσει σημαντική πρόοδο στην εποχή μας, και λογισμικό με περισσότερες ή λιγότερες σχετικές δυνατότητες απαντάται πλέον σε διάφορες συσκευές και υποδομές που χρησιμοποιούμε καθημερινά. Κατά συνέπεια, η αξιολόγηση της προόδου της ΤΝ μέσω ("επιτραπέζιων" και όχι μόνο) παιγνίων στα οποία ανταγωνίζεται τους ανθρώπους, ακολουθώντας μια παράδοση δεκαετιών, έχει γίνει ίσως πιο σημαντική από ποτέ. Παρ' όλα αυτά, η χρήση ΤΝ σε επιτραπέζια παίγνια που απαιτούν τη συνεργασία των παικτών, χωρίς να υπάρχει οποιοδήποτε στοιχείο του ανταγωνισμού μεταξύ τους κατά τη διάρκεια ενός παιχνιδιού, είναι εξαιρετικά σπάνια. Τούτων δοθέντων, στην παρούσα διπλωματική αναπτύξαμε την απαραίτητη υποδομή ενός διακομιστή λογισμικού (Software Server) που επιτρέπει τη συμμετοχή αυτόνομων πρακτόρων λογισμικού, στο πολυπρακτορικό συνεργατικό επιτραπέζιο παιχνίδι ``Pandemic'' · ακόμα, αναπτύξαμε επτά διαφορετικούς αυτόνομους πράκτορες που μπορούν να παίξουν το παιχνίδι αυτό, και τους εξοπλίσαμε με διαφορετικές στρατηγικές (μία ανά πράκτορα) τις οποίες επίσης σχεδιάσαμε. Ο διακομιστής μας έχει ήδη χρησιμοποιηθεί στο πλαίσιο ενός σχετικού με ΤΝ μαθήματος στο Πολυτεχνείο Κρήτης, και μπορεί να παρέχει τη βάση για μια πληρέστερη υποδομή η οποία ευελπιστούμε να χρησιμοποιηθεί για την διεξαγωγή ενός διεθνούς διαγωνισμού αυτόνομων πρακτόρων για το παιχνίδι Pandemic. Τέλος, στα πλαίσια της εργασίας μας διενεργήσαμε μια προκαταρκτική πειραματική αξιολόγηση των στρατηγικών των πρακτόρων μας, η οποία μας οδήγησε στην άντληση χρήσιμων συμπερασμάτων σχετικά με την αποτελεσματικότητά τους σε διαφορετικής δυσκολίας περιβάλλοντα του Pandemic.

Acknowledgements

To all the people that I met in Chania and to the unforgettable moments we created together.

Contents

1 Introduction **1**

1.1 Thesis Contribution 2

1.2 Thesis Overview 2

2 Background **3**

2.1 Pandemic: The Board Game 3

2.1.1 Overview 3

2.1.2 Initial Setup 5

2.1.3 Game play 6

2.1.4 End Of Game 9

2.1.5 Roles 10

2.1.6 Commonly Overlooked Rules 11

2.2 Board Games & AI 11

2.2.1 Connecting Board Games to Real Life 12

2.2.2 Evolution of AI in board games 12

2.2.3 Board games characteristics 14

2.2.4 Agents in AI 15

2.3 Algorithms used in AI 16

2.3.1 Uninformed Search Algorithms 17

2.3.2 Informed Search Algorithms for Games 19

2.3.3 Other Search Methods and Algorithms 24

2.3.4 Advanced AI Techniques 24

2.4 Related Work 26

3 Our Approach **29**

3.1 Overview 29

3.1.1 Pandemic characteristics 29
3.2 Breaking down the components 31
3.2.1 Game Requirements 31
3.2.2 Our vision of Pandemic 32
3.2.3 Agent's Methods 32
3.2.4 Experiment Requirements 33
3.3 Server - Client Architecture 33
3.3.1 Server Side 33
3.3.2 Client Side 34
3.4 Type of Agents 40
3.4.1 Dummy Agent 40
3.4.2 Agent Simple 42
3.4.3 Agent BaseTactic 43
3.4.4 Agent Dynamic 44
3.4.5 Agent Look-ahead 45
3.4.6 Agent HeuristicsCombined 47
3.4.7 Agent Montecarlo Tree Search 48
3.5 Features - Comparison 49
3.5.1 Player Profiling 49
3.5.2 Suggestion Making 50
3.5.3 Comparison 50
3.6 Data For Evaluation 51
3.7 Agent's pseudocodes collection 52

4 Experiments and Results **59**
4.1 Calculating bounds 60
4.2 Experiments with Homogeneous Settings 61
4.2.1 Comparing Strategies while Increasing Game Difficulty 61
4.2.2 Focusing on the performance of each individual strategy 63
4.3 Mixed Teams - Heterogeneous Results 71
4.4 To Cooperate or Not To Cooperate? 73

5 Conclusion **75**
5.1 Conclusions and Future Work 75
5.2 Future Work 76

References **81**

List of Figures

2.1 The board with some of the rest of the games content 5
2.2 Outbreaks explained 10
2.3 Historical AI machines 12
13figure.caption.17
14figure.caption.18
2.6 Differences between tree traversing methods 19
20figure.caption.25
2.8 Montecarlo Tree Search steps [1] 21

3.1 Threads Explained 34
3.2 Coding with variables explained in natural language 35
3.3 Coding with actions explained in natural language 36
3.4 Console output for successful (legal) actions 36
3.5 System Infrastructure Diagram of the whole framework 37
3.6 Class Diagram of our whole system Infrastructure, where we can see each class's main compartments as well us the connection that it has with other classes. 38
3.7 Server's Flowchart. These are the actions made from the server in order to create the game's flow (eg. start/win/lose/next player/ etc.) 39
3.8 Agent_Dummy Flowchart 41
3.9 Agent_Simple Flowchart 42
3.10 Agent_BaseTactic Flowchart 43
3.11 Agent_Dynamic Flowchart 44
3.12 Agent_Lookahead Flowchart 46
3.13 Agent_HeuristicsCombined Flowchart 48

3.14 Agent_Montecarlo Flowchart 49
3.15 Features Compared 51
3.16 End of game message & stats 51

4.1 Homogeneous results using 4 Epidemic cards (50 experiments) 65
4.2 Homogeneous results using 5 Epidemic cards (50 experiments) 66
4.3 Homogeneous results using 6 Epidemic cards (50 experiments) 67
4.4 Homogeneous results using Agent Base-Tactic (50 experiments) 68
4.5 Homogeneous results using Agent Look-ahead (50 experiments) 69
4.6 Homogeneous results using Agent Combined (50 experiments) 70
4.7 Heterogeneous results using 2 Players 71
4.8 Heterogeneous results using 3 Players 71
4.9 Heterogeneous results using 4 Players 72
4.10 Results with & without cooperation 73

List of Algorithms

1 Dummy agent 52
2 Agent Simple 53
3 Agent BaseTactic 54
4 Agent Dynamic 55
5 Agent Look-Ahead 55
6 Agent HueuristicsCombined 56
7 Agent Monte-Carlo Tree Search 56

Chapter 1

Introduction

Now-days, AI is everywhere. From simple machines of everyday use to complex algorithms that can solve some of the most difficult problems. Board games is an other field that the AI is growing and becomes better everyday. Focusing on the field of board games we will emphasize to co-operational strategic games and the agents that are playing in these domains. There is a long tradition of building software agents competitions in order to evaluate new strategic decision making algorithm in adversarial settings The cooperative board game Pandemic [2] is the main topic for this thesis, and our goal is the implementation of a software server for this game, as well as of an autonomous agent to be playing this game. We study various approaches on AI agents and we evaluate how well our agent performs in different scenarios and domains.

So, a question arises. Why do we care about board games? The answer on that is that board games may be unrealistic, but they provide us with the advantage of knowing exactly what it means to win, what moves are available to us and, if not exactly, then we do have a rough idea, what consequences these moves are going to have. That means, that building a replicable system based on a board game, where rules could be the same for a hundred years, will not be that hard as building one based on the real world, where things are getting complicated. A favourable thing is that even people change, times change and maintaining stability in the real world is tough, board games seem to be a measure of intelligence throughout the years.

1.1 Thesis Contribution

This thesis focuses on co-operational strategic games and on the agents that are playing in such domains. Our purpose in this document and work is manifold. First, is to explore the interesting domain of the board game Pandemic and the possibilities on developing new and intelligent agents, that this game has. Second, to develop software for hosting this domain in order to be used for creating and improving autonomous agents and to allow different agents to be fitted against each other in this domain. Third, to develop different versions with various approaches of agents and evaluate them based on their performance. The agents are being tested in different scenarios and are evaluated based on various statistics.

We stress also that a longer term goal for this work is to provide the framework for a new international competition of Pandemic playing agents. Moreover, we report that our server has already been successfully used for the COMP512 “Multiagent Systems” coursework needs, at the School of ECE, Technical University of Crete.

1.2 Thesis Overview

Finally, we give a brief overview of the content of our thesis:

- In Chapter 2 we present all the background information needed for this thesis. We give an overview of the board game itself, and also terms and concepts of tree searching methods. We also state our co-operational game problem and discuss the related work.

- In Chapter 3 we describe our approach on this game, the various agents that were built in order to play in this domain. We will also present the appropriate material for implementing both the server software and the autonomous agent.

- In Chapter 4 we evaluate the performance of our agents and compare them against each other.

- Finally, Chapter 5 acts as an epilogue for this thesis and other future work.

Chapter 2

Background

2.1 Pandemic: The Board Game

Starting off, we will be discussing about the board game that is the topic of this thesis. The fact that Pandemic has a lot of interesting rules that explicitly call for the agents to cooperate, unlike most other board games, and it's high complexity, where the main reasons that incentivised us to choose it for this thesis. It is of high importance to note that the players are not only striving for a mutual goal, but it is also allowed (and encouraged) from the game rules to discuss before making a move. Due to that fact we also need to have a framework that is able to handle the communication part before moving on to any move that are affecting the game board.

2.1.1 Overview

General Info

Based on the description of it, "Pandemic [2] is a board game created by Matt Leacock and distributed by Z-man Games. In this game the main goal is to save humanity from four deadly diseases. As a skilled member of a disease-fighting team, you must keep these diseases at bay while discovering their cures. You and your teammates will travel across the globe, treating infections while finding resources for cures. You must work together, using individual strengths, to succeed. The clock is ticking as outbreaks and epidemics fuel spreading plagues"

Contents

The components of the game, are the following..

- ***4 Role Cards*** : These cards are shuffled and given out to players in order to distribute their roles. The number of players could vary from the minimum of two players, to the maximum of four. There is no need to use all of the cards if the game is played with fewer players.
- ***4 Pawns*** : Each pawn represents the location of a player in the board. As before there is no need to use them all if the game has less than 4 players.
- ***54 Player Cards*** : Player cards contain *48 City Cards* that represent one city on the board and *6 Epidemic Cards*. You can use from 4 to 6 Epidemic cards based on the desired difficulty you want the game to have.
- ***48 Infection Cards*** : Each of these cards represent a city on the board that is going to be infected.
- ***96 Disease cubes*** : There are 24 cubes of 4 different colors. Each cube represent one level of intensity and the color of the cubes represent the disease.
- ***4 Cure Markers*** : These markers are used to notify us about the disease status. Each one has a color and if the marker is there it means that the disease is either cured or eradicated based on which side the marker is flipped.
- ***1 Infection Rate Marker*** : This marker indicates the number of cities that are going to be infected during the infection phase.
- ***1 Outbreak Marker*** : This marker indicates the number of the outbreaks that already have occurred.
- ***6 Research Stations*** : If the city has a research station, some additional actions can be done while the player is in this city.
- ***1 Board*** : Shows all the connections around the cities, with the connected path among them, as well as, the color of each city.

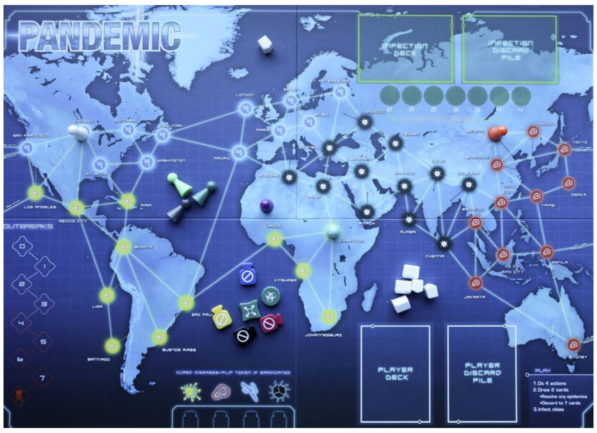


Figure 2.1: The board with some of the rest of the games content

We have to mention that this is not a full version of the game so some of the contents of the actual board game are missing. Some of them are the game's event cards, the Share Knowledge functionality and the throwing a specific card from the player's hand when the hand's card limit is exceeded. Future implementations of this game could include these.

2.1.2 Initial Setup

During the initial setup of the game the following actions need to be done.

1. Atlanta is the main station of the CDC, the Center for Disease Control and Prevention. All the player's paws will be placed in Atlanta and a Research Station will also be placed there.
2. Mark the Outbreak Marker as "0", and set all of the four diseases as Active.
3. For the initial infections the server will..
    - Place the Infection Marker on "2".
    - Shuffle the Infection Deck.
    - Pick the first 3 cards of the deck and place 3 disease cubes of the matching color on each of these cities.
    - Repeat the previous step by placing 2 disease cubes on each city.
    - Repeat the previous step by placing 1 disease cube on each city this time.

- A total number of 18 cubes are placed, each matching the color of the city.
- These 9 cards will be placed face up on the Infection Discarded Pile.
- The rest of the cards form the Infection Deck.

4. The four pawns (or less for fewer players) will be given out randomly to players. Each player must have only one role card. The role cards will be visible from the rest of the players. Based on step 1 all of the player's pawns must me in Atlanta. The unused Role cards and pawns will be removed from the game. Shuffle the player cards without the epidemic cards inside. Give the number of cards needed (accordingly to the number of players) to each player to form the initial hands. For a 2-player game give 4 cards to each player, for a 3-player game give 3 cards and for a 4-player game give 2 cards.

5. The games difficulty is adjusted based on the Epidemic cards used, using either 4,5 or 6 Epidemic cards, for the Introductory, Standard or Heroic game. If there are more Epidemic cards than needed, remove them from the game. Add the number of Epidemic cards you want to the Players Deck and shuffle it.

6. Everything is ready for the game to begin!

2.1.3 Game play

In every turn the player performs 3 Steps:

1. Do 4 actions.

2. Draw 2 Player cards.

3. Infect cities.

The aforementioned steps will explained in more details below.

Actions

Each player can pick a sequence of up to 4 actions. There is also the possibility of “pass” if the player decides not to use one or more actions. The type of actions that are available are the following.

- *Movement Actions*
  - ***Drive/Ferry*** : Move to a neighbour city form the one that you are in.
  - ***Direct Flight*** : Discard a City card to move to the city named on the card.
  - ***Charter Flight*** : Discard the City card that *matches* the city you are in, to move to *any* city.
  - ***Shuttle Flight*** : Move from a city with a Research Station to any other city that ha s a Research Station.
- *Utility Actions*
  - ***Build a Research Station*** : In order for a Research Station to be built the player has to discard the City Card that matches the current city of the player. If the limit of 6 Research Stations has been already met, the player has to remove a Research Station from anywhere in the board, in order to place a new one.
  - ***Treat Disease*** : This action removes 1 disease cube form the current city of the player, and moving it to the cube supply. If the color of this disease has already been cured (See Discover a Cure below), remove all cubes of this specific color from that city. If the removed cube(s) happens to be the last cube of a cured disease, the disease gets eradicated and it should be marked like that, changing the status of this disease to Eradicated.
  - ***Discover a Cure*** : In order for a Cure to be found, the player has to discard 5 City cards of the same color as the disease wished to be cured, at any Research Station to cure this disease. Change the status of this disease as Cured.

Draw Cards

After the competition of the 4 actions, the players draws the top 2 cards from the Player Deck. It's important to note that if the Player Deck has less than 2 cards left when the player needs to draw the top two, the game ends and the player's team has lost! (Reshuffling, to form a new deck is not permitted.)

- ***Epidemic Cards*** : If the draws include any Epidemic cards, immediately do the following steps in order:

    1. ***Increase*** : Move the infection rate marker forward by 1 space on the Infection Rate Track.
    2. ***Infect*** : Draw the *bottom* card from the Infection Deck. Unless its disease color has been eradicated, put 3 disease cubes of that color in the named city. If the city already has cubes of this color, do not add 3 cubes to it. Instead, add just enough cubes, so that it has 3 cubes of this color and then an *outbreak* of this disease occurs in the city (see Outbreaks below). Discard this card to the Infection Discard Pile. If the placement of the number of cubes actually needed on the board is not possible, because there are not enough cubes of the needed color left in the supply, the game ends and the player's team has lost! This can occur during an epidemic, an outbreak, or infections (see Outbreaks and Infections below).
    3. ***Intensify*** : Reshuffle just the cards in the Infection Discard Pile and place them on the top of the Infection Deck. When doing these steps, remember to draw from the bottom of the Infection Deck and then reshuffle only the Infection Discard Pile, placing it on top of the existing Infection Deck.

    It is rare, but possible to draw 2 Epidemic cards at once. In this case, do all three steps above once and then again.

    In this case the second epidemic's infection card will be the only card to "reshuffle", ending on the top of the Infection Deck. An outbreak will then occur in this city during Infections (see Infections, below)

    After resolving any Epidemic cards, remove them from the game. Do not draw replacement cards for them.

Infections

Flip over as many Infection Cards from the top of the Infection Deck as the current Infection Rate. Flip these cards over, one at a time, infecting the city named on each card.

To infect a city, place 1 disease cube matching its color onto the city, unless the disease has been eradicated. If the city already has 3 cubes of this color, do not place a 4^{th} cube. Instead, an outbreak of this disease occurs in the city (see outbreaks below). Discard this card to the Infection Discard Pile.

- ***Outbreaks***

  When a disease outbreak occur, move the outbreak marker forward 1 space on the Outbreaks Track. Then, place 1 cube of the disease color on every city connected to the city. If any of them has already 3 cubes of the disease color, do not place a 4^{th} cube in those cities. Instead, in each of them, a chain reaction outbreak occurs after the current outbreak is done.

  When a chain reaction outbreak occurs, first move the outbreaks marker forward 1 space. Then, place cubes as above, except do not add a cube to the cities that have already an outbreak (or a chain reaction outbreak) as part of resolving the current Infection card.

  As a result of outbreaks, a city may have disease cubes of multiple colors on it, up to 3 cubes of each color.

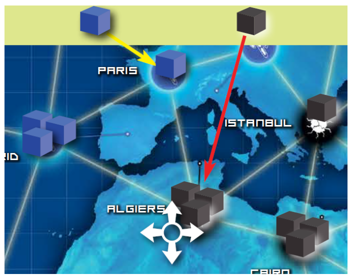
  If the outbreaks marker reaches the last space of the Outbreaks Track, the game ends and your team has lost!

End Turn

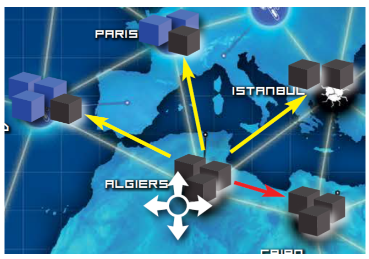
After infecting cities and discarding infection cards, your turn is over. The next player begins a turn.

2.1.4 End Of Game

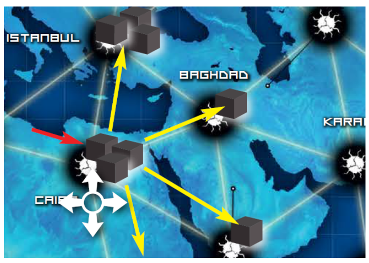
The players win, as soon as the cures to all 4 diseases are discovered and no loss condition is true.



((a)) The beginning of an outbreak



((b)) Outbreak occurs



((c)) A chained outbreak reaction

Figure 2.2: Outbreaks explained

The players do not have to eradicate all 4 diseases to win, just cure them. Once all diseases are cured, the game ends and players win immediately, no matter how many cubes are on the board.

There are 3 ways for the game to end and the players to lose.

1. If the Outbreaks Marker reaches the max outbreak limit (max is 8 outbreaks).
2. If a player does not have enough cubes to add to the city/cities that he/she is asked to, it is not possible to add a different color of cube from the one that was asked to be placed.
3. If the Player Deck is empty and thus, the player cannot draw 2 Player cards after doing his/ her actions.

2.1.5 Roles

Each player is assigned a role in the beginning of the game, with special abilities to improve your team's chances.

Medic

- The Medic removes all cubes, not 1, of the same color when doing the Treat Disease action.

- If a disease has been cured, he automatically removes all cubes of that color from a city, simply by entering it, or being there. This does not take an action.
- The Medic also prevents placing disease cubes (and outbreaks) of cured diseases in his location

Operations Expert

- The Operations Expert may, as an action build a Research Station in his current city without discarding (or using) a City Card.
- Once per turn, he can move form a Research Station to any city, by discarding any City Card.

Quarantine Specialist

- The Quarantine Specialist prevents both outbreaks and the placement of disease cubes in the city she is in and all cities connected to that.

Scientist

- The Scientist needs only 4 (not 5) City cards of the same color to Discover a Cure for that disease.

2.1.6 Commonly Overlooked Rules

1. Players are not drawing a replacement card after drawing an Epidemic card.
2. Players may Discover a Cure in any Research Station. The color of its city does not need to match the disease that the players are curing.

2.2 Board Games & AI

After presenting our game of choice, it is time to talk about how AI can be (and is) related to Board Games. Stupid machines, interesting prototypes and computer world champions, are all part of how the AI was evolved during the years to reach it is current state of intelligence.

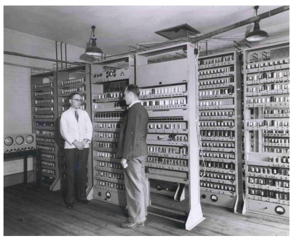
2.2.1 Connecting Board Games to Real Life

We can clearly see that board games have a lot of features that you can spot in real world. Fundamental thoughts of playing a board game, such as planning a strategy, figuring out what moves you are gonna make and how to execute them to achieve your goals, both short term and long term, calculating the risk of each of your moves, are also connected tightly to the thoughts that we have in the real world. So in order to make machines do things that seem intelligent to us, board games could be the perfect place to start off.

2.2.2 Evolution of AI in board games

Games and AI have a long history [3] going all the way back to the 1950s. Back then they were used to kind of drive the initial developments of artificial learning and machine intelligence.

Machines to play board games have been built from computer scientists from a time that goes back to 1952. Back then the Machine OXO, managed to play the first game of tic-tac-toe. Flashing forward in 1995, the computer Chinook became the champion in checkers. An important thing to keep in mind is that even technically Chinook was the current champion, the mach between Chinook and the best player of that time, Marion Tinsley, was not completed because Marion passed away in the middle of the match, so we can not really say that Chinook was the first AI beating human intelligence. Moving on, on chess, in 1997 Deep Blue beat Garry Kasparov, the world chess champion at the time.


((a)) Machine OXO - TicTacToe (1952)

((b)) Chinook - Checkers (1989)

((c)) Deep Blue - Chess (1985)

Figure 2.3: Historical AI machines

Computers were also used in other board games way more complicated than Chess, such as Go, a 2-players, deterministic and perfect information game. The first program for Go, was probably written by Albert Zobrist in 1968, that could beat just a beginner.

After that, we have many more programs written for that game, but the first step-stone was in 1981 from a program called Wally using a 15x15 board that fitted within the KIM-1 microcomputer's 1K RAM. Three years later, in 1984, the first computer tournament of GO was held and was won from Nemesis by Bruce Wilcox, which was also the first program competing in a human Go tournament. In 1998, very strong players were still able to beat programs while giving handicaps of 25 to 30 stones, an enormous handicap that few human players would ever take. By the year 2004 computers could beat an intermediate player with a 9 stones handicap. In 2010 programs showed steady improvement and reduced the handicap to 7 stones for an advanced player. In 2015 the handicap reduced again to 5 stones again for an advanced player. A breakthrough happened one year later, in 2016, when Lee Sedol, one of the top professionals, was beaten with an impressive 0 handicap game form AlphaGo. A few months later, in May of 2017, the second version, AlphaGo Master defeated Kie Jie, the top professional and became the best Go player worldwide. In October of 2017, AplhaGo Zero defeated AlphaGo Master, and became Go champion.

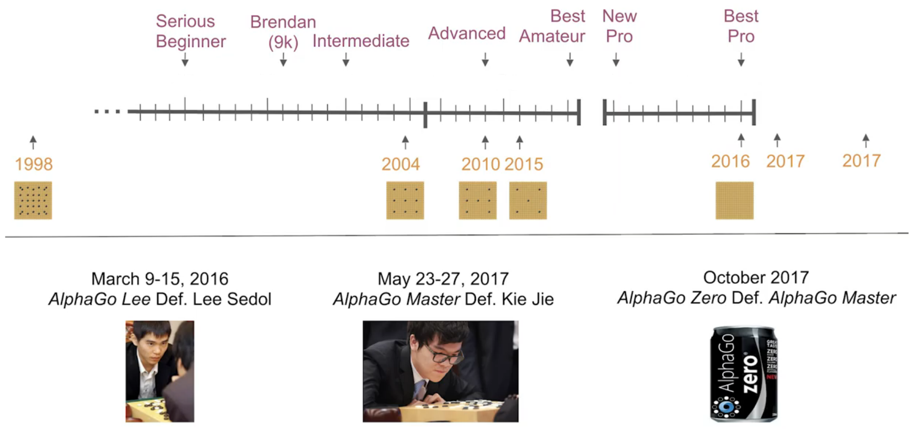


Figure 2.4: The evolution of AI machines in the game of GO. In the top of the diagram we do see the current level of the AI, and in the bottom the number and position of the given handicap that computers had.[2]

[2]Figure retrieved from an online resource that is however no longer available online.

So, from this diagram we can see how much time the computer needed to break decrease the handicap from 9 to 7 stones (6 years), from 7 to 5 (5 years) and lastly from 5 to 0 (less that 1 year). Technology is speeding up as the years go by, so does the AI technology that is used.

2.2.3 Board games characteristics

So, starting off, we are going to mention some characteristics of board games that do have big impact on how we are going to approach each one. Some of them are the following.

- **Players number** : Bard games can be found with a huge diverse on how many people can simultaneously play the game. Games with 2 to 4 players are quite common, but games can vary from player number from 0 (curiously enough they do exist) up to 30 people. Below we can see a diagram of the rating of board games sorted by the players number that each one supports.

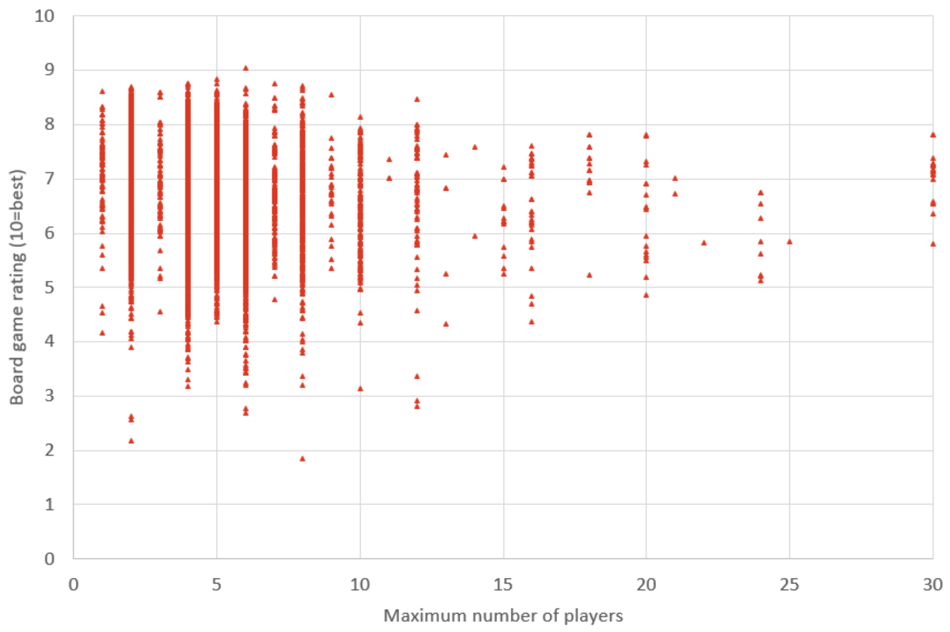


Figure 2.5: Board game ratings by max number of players[2]

[2]Figure retrieved from an online resource that is however no longer available online.

- **Gain or loss of utility** : Based on the utility that is gained from a player compared to the utility that is lost from the opponents, games could be categorized as zero-sum or non zero-sum games. For the first ones each participant's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other participants. We have to mention that in the co-op games, the game itself is considered as an opponent. In the other hand, non zero-sum games may have differences between the utility's gains or losses between players.

- **Perfection using maths** : Games can also be categorized based on how can maths affect the ability of the player. Trivial games are the games that a certain game state can be achieved using maths, or specific equations. On the other side of the coin, we do have non-trivial games, in which the perfect strategy can never be 100% obtainable.

- **Game's popularity** : Based on the popularity of each game, we can describe them as well- known, or not well-know. Do note that this is something that can easily be changed as the game gains or loses popularity.

- **Skills required** : Of course the skills of a player can alter the games results quite much. For that reason we can describe a game as skill demanding when a game's result when the player's skills can alter big time the game's result, or non skill demanding when the skills of a player can not be so important for the game's outcome.

2.2.4 Agents in AI

The term of the word "Agent" may refers to different things in different subjects. In game development community it refers to a player that is a single unit and most of the times in game. In the area of agent research many different definitions are used in order to describe them. But in every different community the agents that are used, have some key features that are generally accepted. Acting autonomously, thinking beforehand, acting independently and being able to act and work in a social environment are some major features that an agent must have. So we can define the agent as such. A software agent is a piece of software that has its own goals available (is proactive) and will try to achieve them without intervention of a user or other program (is autonomous), while sending its environment and reacting to possible changes (is reactive). Additionally, agents in a multi-agent system (MAS) are not centrally controlled, execute asynchronous, and should be able to communicate with each other, the user(s) and the environment. [4].

Starting point of each agent in MASs is that it represents a goal of its own or of the party that it belongs. In order to be autonomous, each agent runs a thread and works towards its goal based on a mechanism that will drive it through the actions that needed to reach (or at least go closer to) the goal state. The agents cannot lack the interaction between them, due to the fact that most of the goals are not independent and may need some kind of cooperation to be achieved.

The agents that are used in AI are using different kind of algorithms in order to achieve their goal. Some of them are using pre-made algorithms that explore the state space blindly, such as Breadth-First Search (BFS) or Depth-First Search (DFS). Others are using techniques to take advantage of the knowledge that they already have in this environment - domain using algorithms that are custom made for the specific problem, such as heuristic search. Outsmarting them both, most of the times, are agents that are exploiting both of these advantages. Working with a heuristic function they invoke a search algorithms to achieve even better results. Algorithms as such is the Minimax or the MCTS.

Our goal is to try out several of these methods and compare them based on their given results. We will also try to combine several methods into one, in order to test out if we can achieve better results using multiple methods at once.

2.3 Algorithms used in AI

In this section we will discuss about various algorithms used in AI. Board games may have different specifications, so choosing the right algorithm(s) that are fitted to work well on your board game is an essential matter. Board games may vary in how many players are in the game, in the number of different possible moves that each player has, or even how much information they do have in the beginning of a round.

2.3.1 Uninformed Search Algorithms

Starting of, we will see some general purpose search algorithms, such as Uninformed search [5], which operate in a brute force way in order to find the solution of a problem. We can also call them blind search algorithms, due to the fact that they do not have any additional information about the search space or state, other than how to do the tree traversing.

Breadth-first Search (BFS)

One of the most common search strategies for tree or graph traversal is Breadth-first search (BFS), which as the name suggests searches breath-wise throughout the tree or graph. Starting from the root node of the tree, it expands all the successor nodes of the current level in order to move on to the next one. Implemented using the data structure of FIFO (first in, first out), BFS is a perfect example of a general graph search algorithm.

- **Advantages**
  - If any solution exist, BFS providing it for sure.
  - It provides the minimal solution found, if solutions are more than one.
- **Disadvantages**
  - It is not memory efficient, because in order to expand to the next level of the tree, you need to store each and every node till there.
  - If the solution of the given problem is far away from the root, BFS is not going to be time efficient.

Depth-first Search (DFS)

Traversing the tree using a recursive algorithm Depth-first search, starts from the root node of the tree and follows each different path to its greatest depth before moving on to the next path. Similar to the BFS that we discussed previously, DFS needs a data structure (stack this time) for its implementation.

- **Advantages**

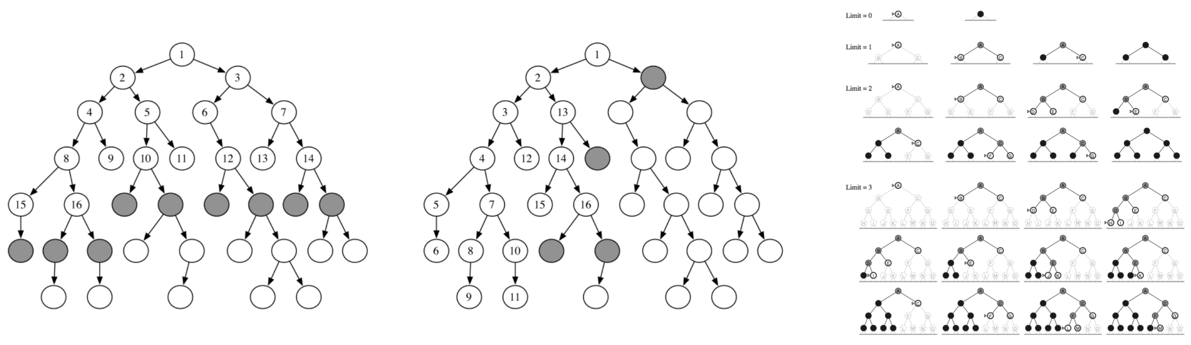
  - DFS is more memory efficient, as just one stack of nodes that shows the path to the current node must be stored.
  - If the right path is chosen for tree traversal then this algorithm is way more time efficient than BFS.

- **Disadvantages**
  - Finding the solution is not guaranteed even if there is one.
  - Re-occurring states may pop up during this search.
  - Infinite loop may occur during this deep down searching method.

Iterative Deepening Search

(IDS) Combining the BFS and DFS algorithms we are moving on to the Iterative Deepening Search (IDs), which is an algorithm that calculates the best depth limit, by increasing gradually the depth limit until the goal is found. So, IDS implements a depth-first search to a certain depth and keeps increasing the depth until the goal node is found. Combining these two methods, we get the advantages of the BFS (time efficiency) and DFS (memory efficiency). Best usage of the IDS is in large space states with an unknown goal node.

- **Advantages**
  - Time efficiency of BFS
  - Memory efficiency of DFS
- **Disadvantages**
  - Keeps repeating all the workload from previous states.



((a)) Tree Traversal using BFS [6]   ((b)) Tree Traversal using DFS [6]   ((c)) Tree Traversal using IDS [7]

Figure 2.6: Differences between tree traversing methods

2.3.2 Informed Search Algorithms for Games

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But, informed search algorithms [8] contains a set of knowledge, that help agents to explore less to the search space and find more efficiently the goal node. This set of knowledge could be how far we are from the goal node, the path cost, or how to reach the goal node, etc.

Pure Heuristic Search

The most simple form of heuristic search algorithms is the Pure Heuristic Search algorithm. Expanding nodes based on their heuristic value, it traverses through the tree in order to find the solution. The implementation of this algorithm, keeps two different lists, an “OPEN” and a “CLOSED” list in order to distinguish the node that have been visited or the ones that are unexplored yet. In every iteration the node with the lowest heuristic value in order to explore its successors and add them to the list that we mentioned before. This algorithm stops when the goal state is found.

- **Advantages - Disadvantages**
  - The quality of this search is purely based on the heuristic function that is based on.

Minimax Search

Again, with the combination of two different methods we have the Minimax Search. This algorithm is based on DFS, but it add a limit to the depth of the search procedure. So, Minimax, searches down to a certain depth and then rates the nodes as they were terminal nodes, using a static evaluation function (heuristic). These values are passed by recursively back up to the tree and each player's node is assigned the best possible move (highest value) and the opponent is assigned again the best possible move (minimum value - worst for the player). So this algorithm is trying to find the best possible moves, based on the best strategy of the opponent. The only problem about this algorithm is to find out the depth that we want to cut-off the search and start rating the nodes based on our heuristics. Things to take in consideration, in order to find out the optimal depth are the number of plays explored, how promising the path is, how much time the computer has left to think, and how "stable" the configuration is.

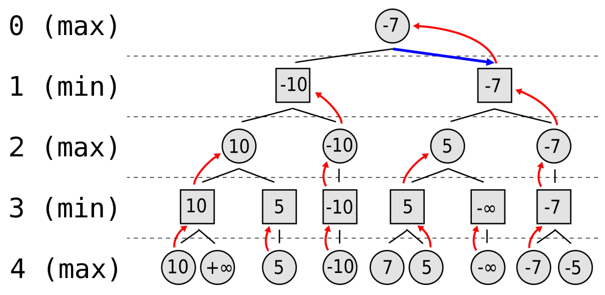


Figure 2.7: Tree traversing, and Back propagation using Minimax[2]

- **Advantages**
  - Gives us the player's optimal move based on the assumption that the opponent is also playing optimally.
- **Disadvantages**
  - A main disadvantage of this algorithm is that in complex games with big branching factors, such as Chess or Go, it gets really slow, because the player has a lot of choices to decide from. Using different pruning methods really

[2]https://en.wikipedia.org/wiki/Minimax

improve the performance of this algorithm. One of the most know pruning methods that is used with this algorithm is alpha-beta pruning.

Montecarlo Tree Search

As an extension of the Minimax Search algorithm mentioned above, the Montecalro Tree Search (MCTS) [9] is using various simulation methods in order to be usable and also efficient in wide search spaces with big search trees. The game tree that both of these algorithms are building is similar, as well as the logic for selecting moves. The big and main difference between them is how the scoring system works. MCTS is not using an explicit score - estimation heuristic, instead, it uses simulated games (play-outs) in the nodes of the game tree, in order to find out which is the best move. The MCTS is a simple game tree that is build accordingly to the results of the random play-outs and can be broken down into four steps that are the following:

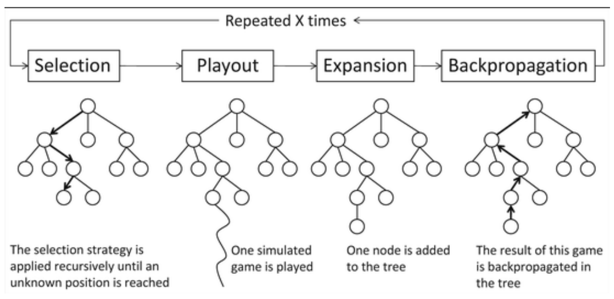

Repeated X times
Selection
Playout
Expansion
Backpropagation
The selection strategy is applied recursively until an unknown position is reached
One simulated game is played
One node is added to the tree
The result of this game is backpropagated in the tree

Figure 2.8: Montecarlo Tree Search steps [1]

1. **Selection** : During the selection phase the good child nodes are selected, starting form the game tree's root node (R), in order for these selections to lead us to a better overall outcome (win).

2. **Expansion** : In the expansion phase, if the current node (C) is not a terminal node, thus the game continues, we create one or more children and we select one of them as the new current (C).

3. **Simulation (Roll-out)** : We do run a simulated random game from the current node (C) till the end of the game where we do have a result for the game.

4. **Back-propagation** : We Back-propagate the result that we found in the previous phase back to the game tree in order to update the current move sequence with the simulation results.

MCTS has two important pieces of information stored to each node. Fist an estimated value based on the random play-outs from that node and how many time this node has been visited. Also, the simplest and most memory efficient form of this algorithm is adding one node per child, but it may be beneficial to increase this number based on the requirements of the application.

- **Advantages**
  - MCTS can make reasonable decisions even without any strategic knowledge about the domain that is working on. The only thing that this algorithm needs to operate functionally, is the domain legal moves and end conditions. That makes this algorithm. That makes MCTS flexible to work with, even in different environments and problems with minimum modifications.
  - MCTS is perfect for domain with big branching factors such as Go that has a 19x19 domain. It performs asymmetrical tree growth that adapts to the topology of a search space, that would normally cause problems to standard search methods such as BFS or DFS. MCTS is highly adaptive and will (eventually) find the nodes that do appear optimal and focus the work load there.
  - MCTS can be stopped any time in order to return the best estimate so far, without even discarding the search tree, that could be used again in the future.
  - The algorithm is simple to implement.
- **Disadvantages**
  - MCTS can sometimes be time inefficient. Even in games of medium complexity, the sheer size of the combinatorial move space, may pose a problem to the time efficiency of the algorithm. Other than that, nodes that have not been visited enough times, can give reliable estimates. Luckily, the performance of the algorithm can be significantly improved using a number of techniques.

The family of the Montecalro Tree Search algorithms can be found combined with a large amount of variations of tree policies. Each of the tree policies is trying to handle with the best possible way the best way to adjust the exploration of the search space and the exploitation of the knowledge that is acquired. In our version of MCTS we use one of the most popular of them, the UCB1 Upper Confidence Bound algorithm, that is also explained more in detail below.

UCB1 Algorithm

The Upper Confidence Bound UCB1 [10] is used in Reinforcement Learning and focuses on exploration and exploitation based on a certain confidence boundary. A thing to keep in mind is that each machine (could be just a node to us) has different distribution which the profit is determined. This deterministic algorithm is assigning a value to each node on each round of exploration, in order to know how profitable is this node to us, based on how many times more is used than the rest of the nodes.

Given below is the algorithm inside UCB that updates the Confidence bounds of each machine after each round step by step.

1. Two values are considered for each round of exploration of a machine. The number of times each machine has been selected till round n. The sum of rewards collected by each machine till round n.

2. At each round, we compute the average reward and the confidence interval of the machine i up to n rounds as follows

   - **Average reward** : $\overline{r_i}(n) = \frac{R_i(n)}{N_i(n)}$
   - **Confidence Interval** : $[\overline{r_i}(n) - \Delta_i(n), \overline{r_i}(n) + \Delta_i(n)]$, with $\Delta_i(n) = \sqrt{\frac{3}{2} \cdot \frac{log(n)}{N_i(n)}}$

3. The machine with the maximum UCB is selected as follows

   - **UCB** : $\overline{r_i}(n) + \Delta_i(n)$

Both the Monte Carlo method and the Upper Confidence Bound for Trees are commonly used in the research of the best possible tactics in board games. For instance, there are algorithms developed for the board games “Settlers of Catan” [11], [12] and

“Diplomacy [13]. In both of these two games the algorithm of Monte Carlo is used, in order to find an optimal solution.

2.3.3 Other Search Methods and Algorithms

The use of a heuristic Informed Search Algorithm [14] may not produce the best outcome for the current problem, but it will likely produce a good outcome in a reasonable time. Using the current state of a search tree it calculates how close is the agent to the goal node, represented by h(n), and needs a pair of nodes to compare in order to give results. Heuristic functions can also be mixed up in the other methods to make them more time or cost efficient. A good example is the usage of heuristic function for smart pruning in big search trees.

A great example of a heuristic function that is improving Search algorithms, lies behind the idea of A* Search Algorithm. A* is based on Dijkstras Algorithm but in order to be even more efficient in combines information that Greedy Best-First-Search uses. Combining these, the A* algorithm is one of the most common path-finding algorithms and it can be used efficiently in a lot of scenarios.

Algorithms can be combined to one each other to possibly create a more power full one. As we seen above the A* is the combination of Dijkstra with Heuristics, but it is also possible to combine the A* again with Heuristics and create an even more improved version of A*, the IDA* algorithm. IDA* is based on A* and it used a smart way to expand the search tree using heuristics, which lead to keep expanding nodes that are closer to our goal nodes.

We won’t analyze more in depth these two functions mentioned above, because our main goal was to describe how a function can be improved through the usage of smart heuristic methods.

2.3.4 Advanced AI Techniques

The field of AI is keep progressing and due to that, some interesting techniques are getting more and more popular these days. Some of them are the following.

Player Profiling

Player profiling is a technique that is a popular in solutions for board games, that are using AI to achieve the optimal results [15]. It is commonly used in collaborative games in order for the players to be more efficient as a team. The collaborative aspect of games has been shown to potentially increase player performance and engagement over time. Having said that, collaborating players need to perform well for the team as a whole to benefit and for that reason, teams often end up performing no better than a strong player would have performed individually.

Personalising each player in game, most of the times, that means the improvement of overall performance and engagement, but in collaborative games, personalisation is seldom implemented, and when it is, it is overwhelmingly passive such that the player is not guided to goal states and the effectiveness of the personalisation is not evaluated and adapted accordingly. For that reason if the player profiling will be used in order to improve the game, it must be done in an efficient way in order to avoid having a negative outcome out of it.

Opponent Modeling

One more interesting concept similar to the previous one is the opponent modeling technique. Most of the times it is based on data that was gathered from the player profiling that was done before proceeding with the opponent modeling. Opponent modelling is a technique to recognize the strategy of an opponent and make predictions about their behaviour. In AI, opponent modelling is used for decision making by exploiting sub-optimal opponents [16].

An interesting approach on this technique is described in a paper that is using the Theory of Mind (ToM) [17]. Based on the science of psychology ToM is analyzing how players are acting during various situations and in different environments. A follow up in this method was a paper [18] that has examined the interesting application of ToM in groups using a stochastic game as a case study and based on their results understood that ToM under uncertainty performs worse, whereas ToM with complete information performs better in terms of the stochastic game metrics.

2.4 Related Work

During the last few years the game of Pandemic grows more and more popular. It is a challenging cooperative game so it is often used in researches with the usage of AI agents trying to achieve the best possible outcome. Some of the approaches of this game are the following.

- **Pandemic as a Challenge for Human-AI Cooperation** [19] : In this paper, Pablo Sauma Chacon and Markus Eger are introducing a challenging domain for the human and AI cooperation. They implement a game server using Python and Unity (with some restrictions from the original game) in order to get us through the mechanisms and the difficulties that do exist in order to make an AI agent co-operating with a human. They also do propose a first approach for this cooperative environment with incomplete information.

- **PAIndemic: A Planning Agent for Pandemic** [20] : In a following article, Pablo Sauma Chacon and Markus Eger are again using the game of Pandemic as their main subject. This time they do implement a planning agent with some look-ahead functions and the search algorithm of MCTS in order to achieve better results. In this implementation again some functionalities of the game are missing in order for the branching factor to be fairly low. Using the agent that they created can be used as training grounds for other new agents, to test their abilities in this complicated domain.

- **Design and implementation of TAG: A tabletop games framework** [21] : In this research, in version 1, 7 games are implemented with a lot of data about each one. If we focus in the game of Pandemic, that is our main subject, we will notice that the domain of Pandemic is not easy, but quite complex. Comparing it with the other 6 games Pandemic is the most demanding in pretty much every aspect than the rest of the games. So, the game that we are going to work with needs a lot of attention in order to implement something that will be both result and time efficient in this tough domain. The research mentioned above provide us with a framework that can be used for further improvement of autonomus agents.

- **Learning Team Theories and Measurement through the Game Pandemic** [22] : One more article in the game of Pandemic that is teaching us how crucial cooperation between players is. In this article we can see which are the main attributes of cooperation abilities each player must have to be a good player in this domain. Even if this paper is not a AI based research we can evaluate how close our AI is to a human mindset and it provides us crucial information about how cooperation improves nearly every skill.
- **Collaborative Agent Game-play in the Pandemic Board Game** [23] : An other paper written by Konstantinos Sfikas and Antonios Liapis, shows how difficult is still for AI driven players to control the environment in a collaborative game and anticipating long term dangers. Also even their implementation of the agent has good results, they do mention that there is still a lot of thing to find in the unexplored domain of the collaborative games and they do highlight the challenges and opportunities that collaborative games like Pandemic pose to AI agents.

Chapter 3

Our Approach

3.1 Overview

In this chapter we will talk in depth about the approach that we used in this project. How the game was implemented, which kind of methods our clients are using, and some more info about each method are going to be this chapter's main subject. We will also analyze the heuristic functions that were used in this approach, and the tactics behind them.

3.1.1 Pandemic characteristics

So, we do need to specify the environment that we want to implement our game. The game of Pandemic has the following specifications.

1. **Multi-player game:** This is a multiplayer game, opposed to zero-player games (eg. Conway's life) and one player games (eg. 15-puzzle, Rubik's cube or Solitaire). For Pandemic we need at least 2 players to be active and we can have a max of 4 players playing the game.

2. **Zero-sum game:** This means that in this game if a player wins, the opponent (computer for our case) looses. In this board game the event of a player winning the game, ends up with all the players as winners and the computer as loser. This also goes vice - versa with the loss of a player an event is triggered and all the

players lose against the computer. Games that are not zero-sum do not have the rule of one player wins, one loses (eg. The prisoner's dilemma).

3. **Non-trivial game:** Setting up a best playing strategy can not be established using enumeration or mathematical analysis. The big difference with trivial games is that you never know if your strategy is the best (eg. of trivial game is tic-tac-toe where a draw can always be achieved).

4. **Well-known game:** Until recently, this game was not so well-know. The last few years more and more papers and techniques are developed in order to analyze this cooperative game. Known in several countries, available online, with several variations and extensions of the original game (eg. Pandemic Legacy), Pandemic builds up a fair amount of audience around the globe.

5. **Requires skill:** Some games are easy to adapt and do not require a lot of skill. Pandemic is not one of them due to the big amount of rules, the different roles and the general difficulty of the game that is more than the average. In this game an experienced player should have a big advantage over a beginner. Also it does have a strong co-relation between luck and skill in order to win this game.

Based on the analysis [24] on this game we end up with the following results. Due to the (1) we know that we do need to investigate, the cooperation between players. Due to the (2) we know that we do not have to investigate further if one of the players win or lose. The (3) is a trivial specification that it let us have something to investigate further that just analyzing the "perfect" algorithm for the solution. The properties (4) and (5) are the ones that we need in order to be able to check and evaluate our results against other players.

So, we are now ready to present our problem statement, consisting of three questions. Fist is what are the methods that we need for our game. Secondly the same question arises about the agent that we want to create. Last, but not least, we want to specify the experiments that we need to do in order to get our results.

3.2 Breaking down the components

Understanding in depth the need that we do have is a crucial stage before starting the implementation. Different needs may need a different way of approaching the problem. So, following on, we will be defining the specs of our game.

3.2.1 Game Requirements

The game will be composed from 2 sections, the server and the client. The requirements for each one of the is described below.

- Server Side
  - We need our server to be working with 2 to 4 clients.
  - The server needs to exchanging info with the clients (eg. game states).
  - The server needs to initialize the game state for each new game.
  - The server needs to maintain the flow of the game (eg. player's turn).
  - The server needs to stop the game when needed and announce the game result.
  - The server must prevent agents for doing illegal actions.
  - We do need to get all the info about the game via the server.
- Client Side
  - Understanding when it is time to play.
  - Sending info to the server about our actions.
  - Making recommendations.
  - Knowing current role in each game.
  - If an actions is legal.
  - Info about the game state.

3.2.2 Our vision of Pandemic

For our version of Pandemic we do have the following requirements.

- The clients will be having perfect knowledge of the board. The only thing that will be hidden will be the 2 decks.
- It will be a turn-based game and for each turn all the players will be asked for giving out a suggestion to the player that is playing, as well as for an action for the user that is currently playing.
- Infinite time will be given to the clients to give and evaluate the suggestions. Similarly the time given for deciding for an action and evaluating it, will be infinite.
- The clients will be able to use one of the roles of Medic, Scientist, Quarantine Specialist and Operations Expert.
- The ability of cards trading will not be implemented.
- Due to the fact that the players will not be able to trade cards, the hand's card limit will not be implemented as well as the cards needed for curing a disease will be lowered by 1 (4 instead of 5).

3.2.3 Agent's Methods

Following up, is the full specification of the desired methods that our clients will have. Note that, not all of the clients will be implementing all these methods.

- Creation of a way to understand distances around the map.
- Creation of basic tactic.
- Take advantage of each role's special abilities.
- Creation of a heuristic function.
- Using a heuristic search method.
- Using the MCTS algorithm to improve performance.

3.2.4 Experiment Requirements

Lastly, we need to specify what kind of experiments we want do have, in order to implement corresponding evaluation methods.

- Creating different scenarios for our agents to be tested.
- Run a number of random tests in each agent.
- Use different teams of agents for different experiments
- Increase difficulty, test results again.
- Store results and compare them.

3.3 Server - Client Architecture

The first thing that was needed was the implementation of the server as described above. The system that was made is using socket-programming and a server/client model to allow users to communicate over the network. Messages from the clients are all sent to the server, which processes the messages and re-broadcasts them to all the clients doing also any changes that are needed for the game state.

3.3.1 Server Side

Creating a simple TCP/IP server for sending string messages to a client was a fairly simple task. The implementation was using a single-threaded server (no more threads needed due to the fact that the client was just one) and a single-threaded client.

In the early stage of development, the clients were just sending different strings to one each other. As the project was going on, the need of sending objects arises, due to the fact that the current board needed to be sent from the server to the clients. That was the reason why we switched from data input-output stream to object input-output stream. The way that it works is that the server sends the board with all the necessary info to the client using an interval.

The first problem we encountered when multi-threading came in. The system needed more than one client to be active, so more that one threads needed to be there in order to communicate with each client using a different thread. So we created a thread pool using the method “ClientHandler” in order to handling each client. Each thread was listening for a message from the client that was connected to and processes it.

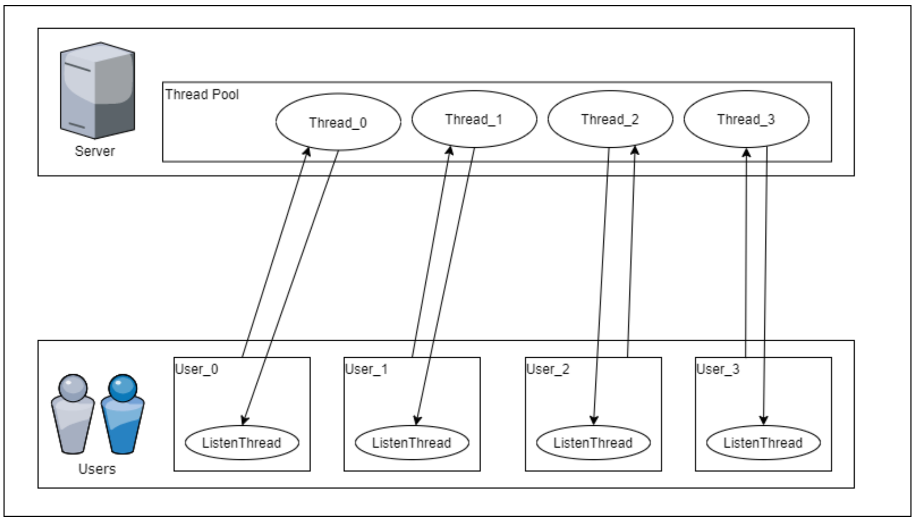


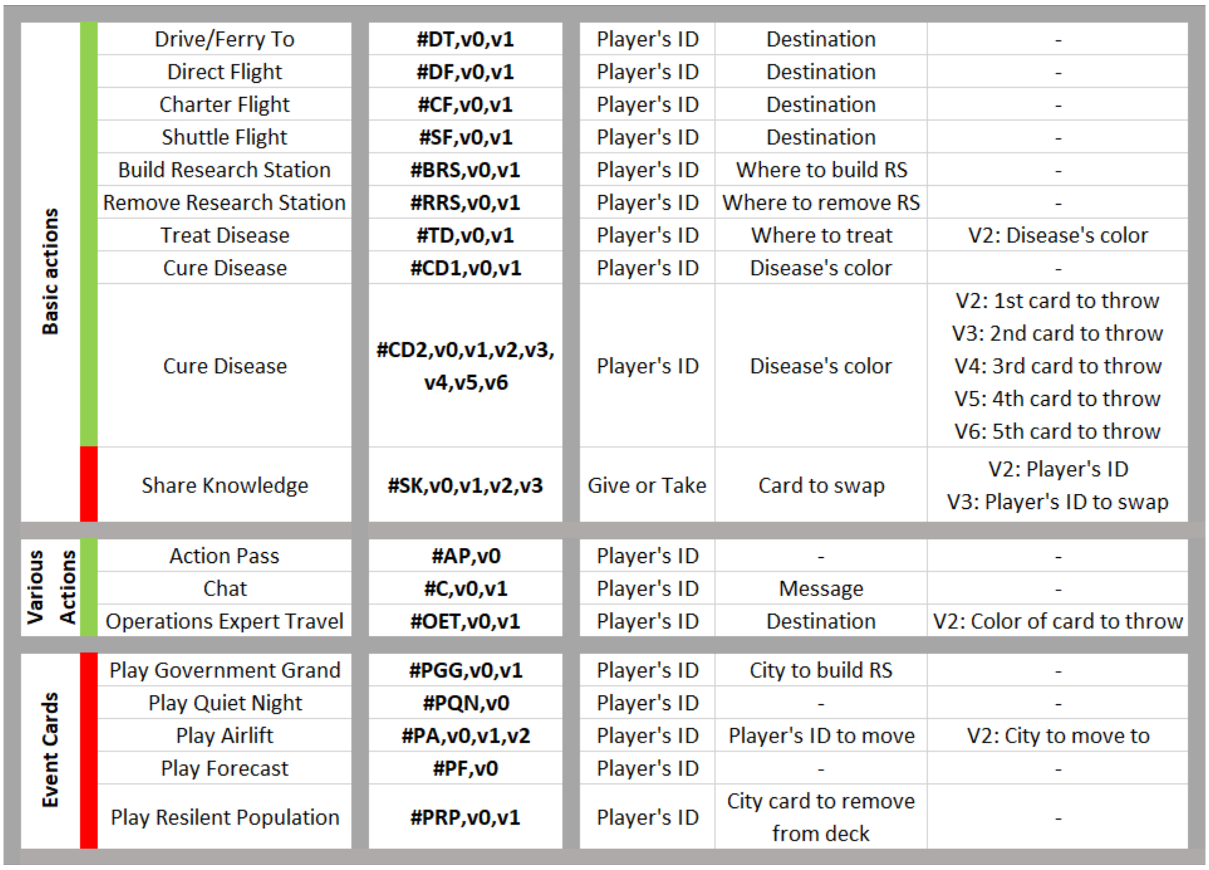
Figure 3.1: Threads Explained

3.3.2 Client Side

After the creation of the server, we moved on to the creation of the client. The client had a listening interval that receives the board with all the messages, suggestions and info on who is paying. Then when the client is playing, it uses a coding system to send a Sting message back to the server with the actions or suggestions. Then the server decodes the message, updates the board and sends the updated board to all the clients. The way that the coding and decoding of actions works is describes below.

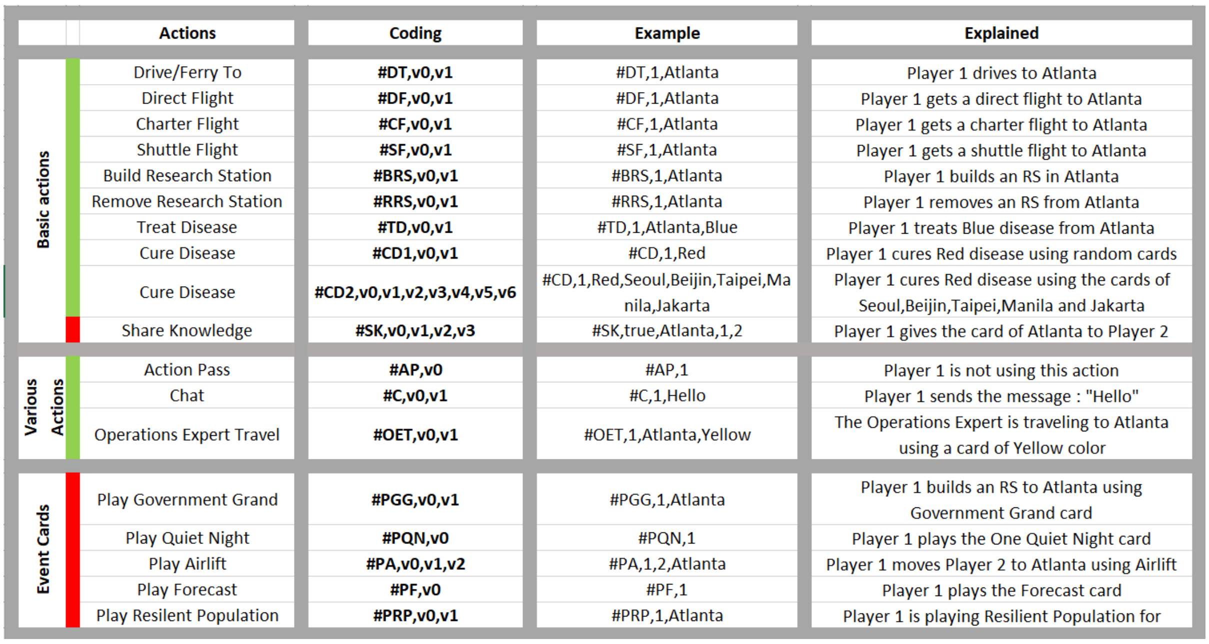
Also, we have to mention that each client is independently listening for messages from the server, so in order to implement that we have one independent Listen Thread to each client. That is how each client is listening for messages back from the server, the same way that the server is listening for messages from the clients.

The actions marked with red are not yet implemented fully and they will be added in the future editions of this game that are being described in the future work section.



Basic actions	🟩	Drive/Ferry To	**#DT,v0,v1**	Player's ID	Destination	-
	🟩	Direct Flight	**#DF,v0,v1**	Player's ID	Destination	-
	🟩	Charter Flight	**#CF,v0,v1**	Player's ID	Destination	-
	🟩	Shuttle Flight	**#SF,v0,v1**	Player's ID	Destination	-
	🟩	Build Research Station	**#BRS,v0,v1**	Player's ID	Where to build RS	-
	🟩	Remove Research Station	**#RRS,v0,v1**	Player's ID	Where to remove RS	-
	🟩	Treat Disease	**#TD,v0,v1**	Player's ID	Where to treat	V2: Disease's color
	🟩	Cure Disease	**#CD1,v0,v1**	Player's ID	Disease's color	-
	🟩	Cure Disease	**#CD2,v0,v1,v2,v3, v4,v5,v6**	Player's ID	Disease's color	V2: 1st card to throw V3: 2nd card to throw V4: 3rd card to throw V5: 4th card to throw V6: 5th card to throw
	🟥	Share Knowledge	**#SK,v0,v1,v2,v3**	Give or Take	Card to swap	V2: Player's ID V3: Player's ID to swap
Various Actions	🟩	Action Pass	**#AP,v0**	Player's ID	-	-
	🟩	Chat	**#C,v0,v1**	Player's ID	Message	-
	🟩	Operations Expert Travel	**#OET,v0,v1**	Player's ID	Destination	V2: Color of card to throw
Event Cards	🟥	Play Government Grand	**#PGG,v0,v1**	Player's ID	City to build RS	-
	🟥	Play Quiet Night	**#PQN,v0**	Player's ID	-	-
	🟥	Play Airlift	**#PA,v0,v1,v2**	Player's ID	Player's ID to move	V2: City to move to
	🟥	Play Forecast	**#PF,v0**	Player's ID	-	-
	🟥	Play Resilent Population	**#PRP,v0,v1**	Player's ID	City card to remove from deck	-

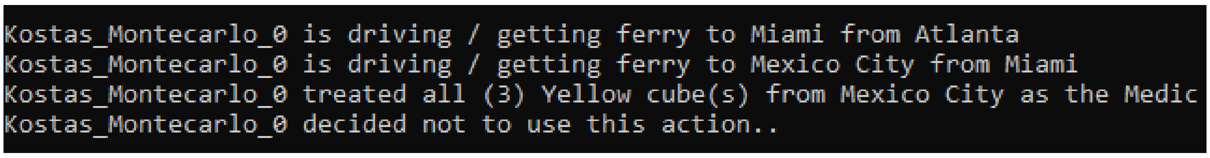
Figure 3.2: Coding with variables explained in natural language



		Actions	Coding	Example	Explained
Basic actions	🟩	Drive/Ferry To	**#DT,v0,v1**	#DT,1,Atlanta	Player 1 drives to Atlanta
	🟩	Direct Flight	**#DF,v0,v1**	#DF,1,Atlanta	Player 1 gets a direct flight to Atlanta
	🟩	Charter Flight	**#CF,v0,v1**	#CF,1,Atlanta	Player 1 gets a charter flight to Atlanta
	🟩	Shuttle Flight	**#SF,v0,v1**	#SF,1,Atlanta	Player 1 gets a shuttle flight to Atlanta
	🟩	Build Research Station	**#BRS,v0,v1**	#BRS,1,Atlanta	Player 1 builds an RS in Atlanta
	🟩	Remove Research Station	**#RRS,v0,v1**	#RRS,1,Atlanta	Player 1 removes an RS from Atlanta
	🟩	Treat Disease	**#TD,v0,v1**	#TD,1,Atlanta,Blue	Player 1 treats Blue disease from Atlanta
	🟩	Cure Disease	**#CD1,v0,v1**	#CD,1,Red	Player 1 cures Red disease using random cards
	🟩	Cure Disease	**#CD2,v0,v1,v2,v3,v4,v5,v6**	#CD,1,Red,Seoul,Beijin,Taipei,Manila,Jakarta	Player 1 cures Red disease using the cards of Seoul,Beijin,Taipei,Manila and Jakarta
	🟥	Share Knowledge	**#SK,v0,v1,v2,v3**	#SK,true,Atlanta,1,2	Player 1 gives the card of Atlanta to Player 2
Various Actions	🟩	Action Pass	**#AP,v0**	#AP,1	Player 1 is not using this action
	🟩	Chat	**#C,v0,v1**	#C,1,Hello	Player 1 sends the message : "Hello"
	🟩	Operations Expert Travel	**#OET,v0,v1**	#OET,1,Atlanta,Yellow	The Operations Expert is traveling to Atlanta using a card of Yellow color
Event Cards	🟥	Play Government Grand	**#PGG,v0,v1**	#PGG,1,Atlanta	Player 1 builds an RS to Atlanta using Government Grand card
	🟥	Play Quiet Night	**#PQN,v0**	#PQN,1	Player 1 plays the One Quiet Night card
	🟥	Play Airlift	**#PA,v0,v1,v2**	#PA,1,2,Atlanta	Player 1 moves Player 2 to Atlanta using Airlift
	🟥	Play Forecast	**#PF,v0**	#PF,1	Player 1 plays the Forecast card
	🟥	Play Resilent Population	**#PRP,v0,v1**	#PRP,1,Atlanta	Player 1 is playing Resilient Population for

Figure 3.3: Coding with actions explained in natural language

The server is also checks if the action(s) of the player are legal. If they are, the game board is updated and a message with all the actions is showing up. If not, a message with the illegal action(s) of the player is showing up in the Server's console.



```
Kostas_Montecarlo_0 is driving / getting ferry to Miami from Atlanta
Kostas_Montecarlo_0 is driving / getting ferry to Mexico City from Miami
Kostas_Montecarlo_0 treated all (3) Yellow cube(s) from Mexico City as the Medic
Kostas_Montecarlo_0 decided not to use this action..
```

Figure 3.4: Console output for successful (legal) actions

So, up to this point our server and our clients are communicating and the objects that are sent from the Server to the clients needed to be described more accurately. For that reason the following diagram with the system infrastructure may be useful for deeper understanding on how the whole architecture works. Using this diagram we can see the different classes of the framework. We can distinguish the ones that are used form the server and the ones that are used be the clients.

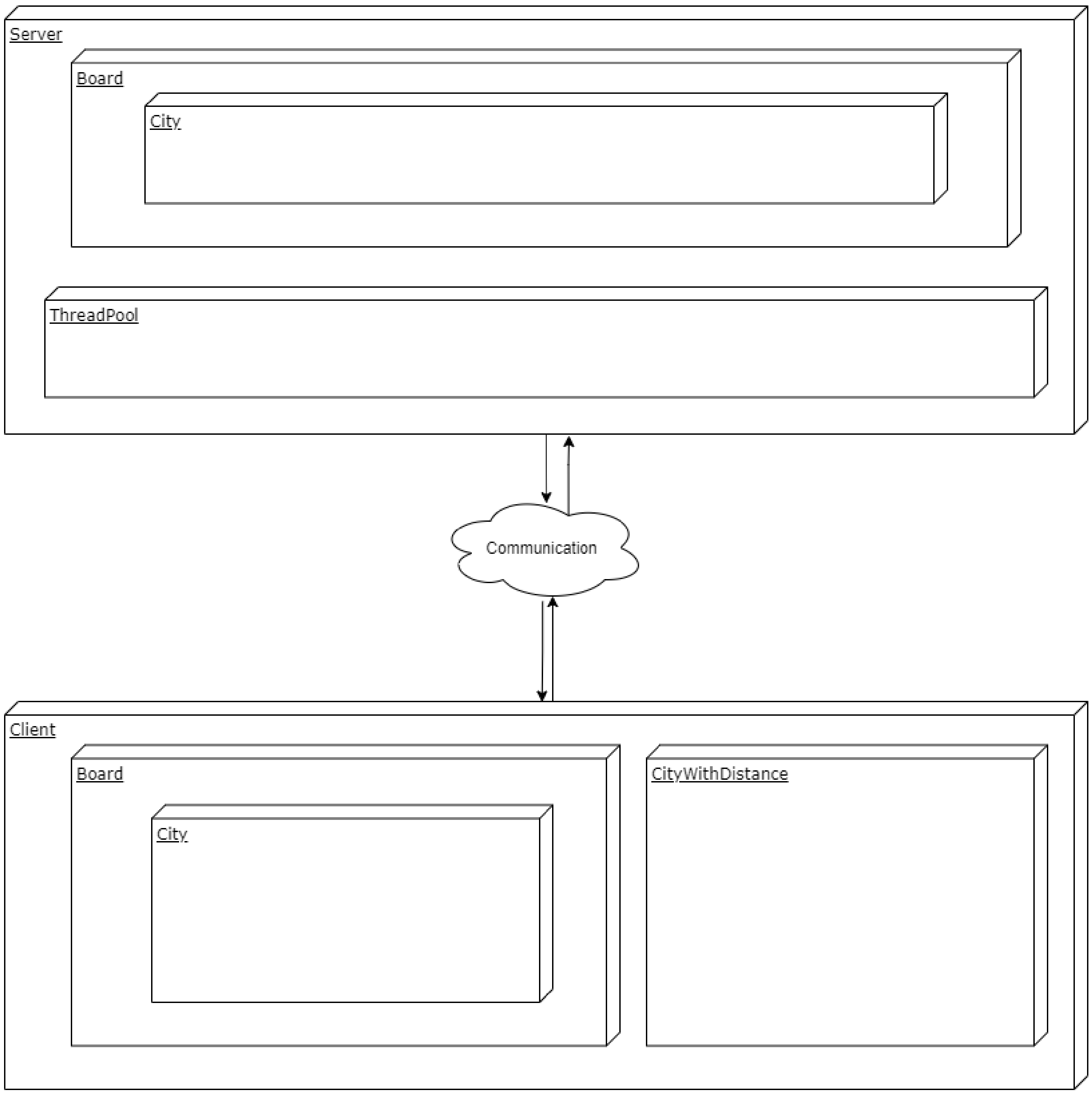


Figure 3.5: System Infrastructure Diagram of the whole framework

A closer look with an abstract class diagram is shown below for better understanding on what every single class is doing, and what are the co-relations between them. We also have a short description of each class in order to make clear what is class is used for.

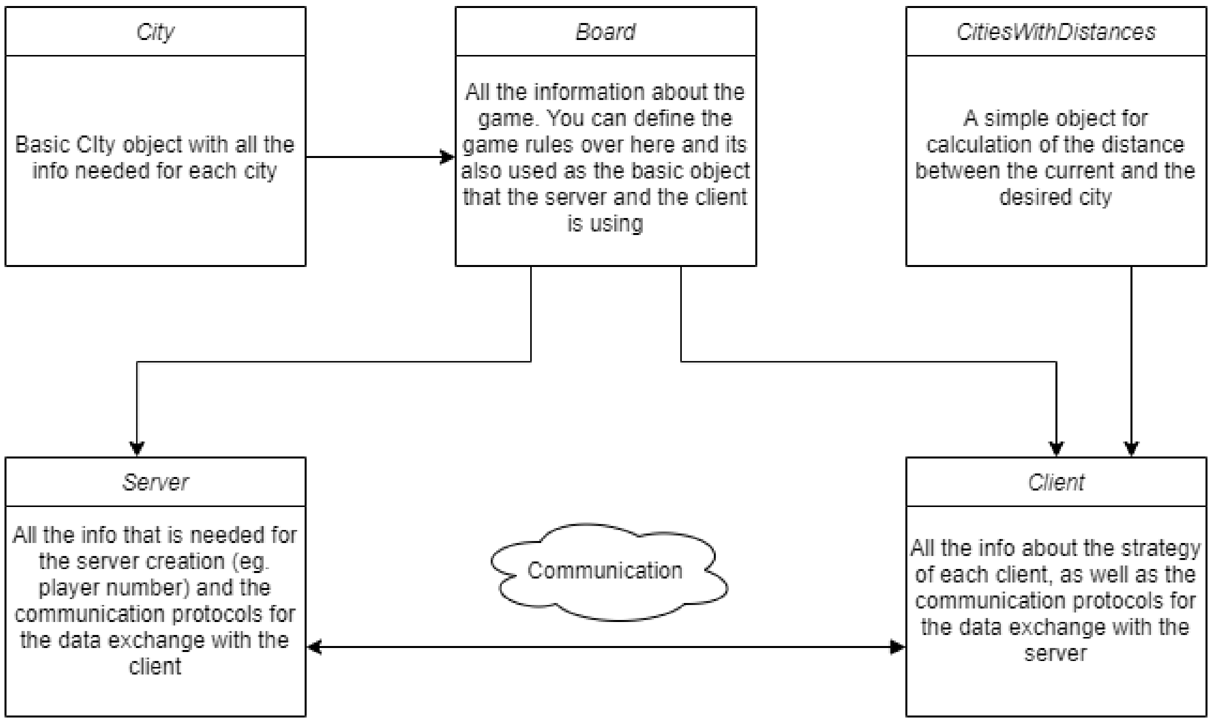

City
Basic CIty object with all the info needed for each city
Board
All the information about the game. You can define the game rules over here and its also used as the basic object that the server and the client is using
CitiesWithDistances
A simple object for calculation of the distance between the current and the desired city
Server
All the info that is needed for the server creation (eg. player number) and the communication protocols for the data exchange with the client
Communication
Client
All the info about the strategy of each client, as well as the communication protocols for the data exchange with the server

Figure 3.6: Class Diagram of our whole system Infrastructure, where we can see each class's main compartments as well us the connection that it has with other classes.

Last, but not least we do have the abstract flowchart of the server. Using this diagram we can understand how the server is running and what are the possible action from the server's side.

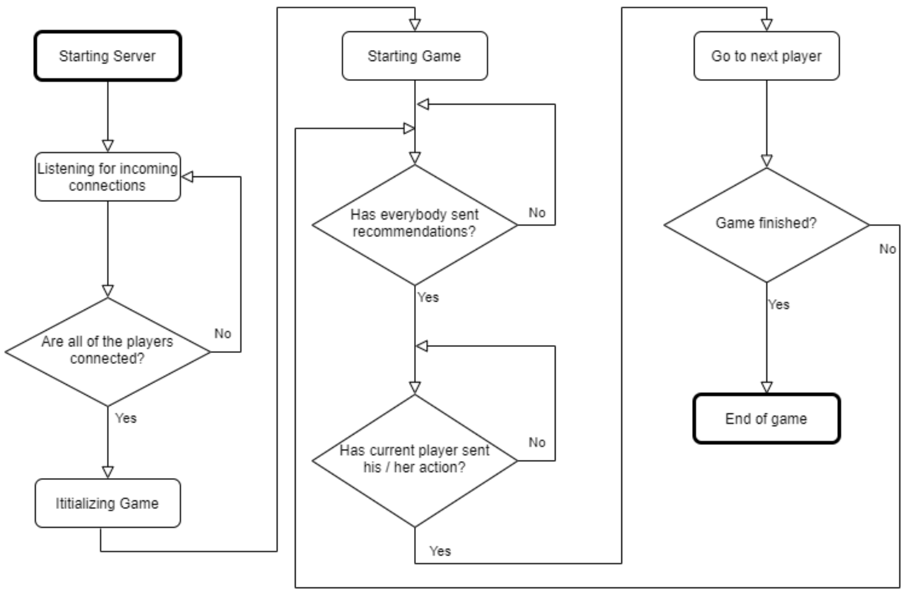

Starting Server
Listening for incoming connections
Are all of the players connected?
No
Yes
Ititializing Game
Starting Game
Has everybody sent recommendations?
No
Yes
Has current player sent his / her action?
No
Yes
Go to next player
Game finished?
No
Yes
End of game

Figure 3.7: Server's Flowchart. These are the actions made from the server in order to create the game's flow (eg. start/win/lose/next player/ etc.)

3.4 Type of Agents

For this project 7 different agents were implemented, with different tactics and algorithms (described in section 2.3) behind each move, in order to test out which is the optimal way to play this cooperative game optimally. Dummy agents, heuristic function based agents and even more intelligent agents are described below. Note that each agent is based on the previous one except the Montecarlo agent, which is not an improvement, but a totally different agent.

3.4.1 Dummy Agent

The first client we implemented was the dummy one. The Dummy client would be the one that is going to test the server for any mistakes. Also the dummy client would be the base, for creating new agents. For the implementation of this client we needed some really simple moves to be added (eg. driving from city to city)

- If the agent has the number of cards needed for curing a disease, it will try to cure without checking if there is an RS in the current position so the curing action may not be successful.
- The agent checks if the current city has cubes placed there. If it does, the agent tries to treat them.
- If none of the above is true, the agent will check the closest cities (distance = 1) for cubes. If they do have a cube, the agent will move there and try to treat them.
- If the (distance = 1) cities are clear from cubes, the agent will check the cities with (distance = 2) for cubes. If they do have a cube, the agent will move there and try to treat them.
- If the agent has checked all of the above and still has some available moves, the agent will randomly drive to a city close to it. When the position of the agent changes, the agent will check again all of the above before driving randomly again.
- The agent is not giving or taking any suggestions.

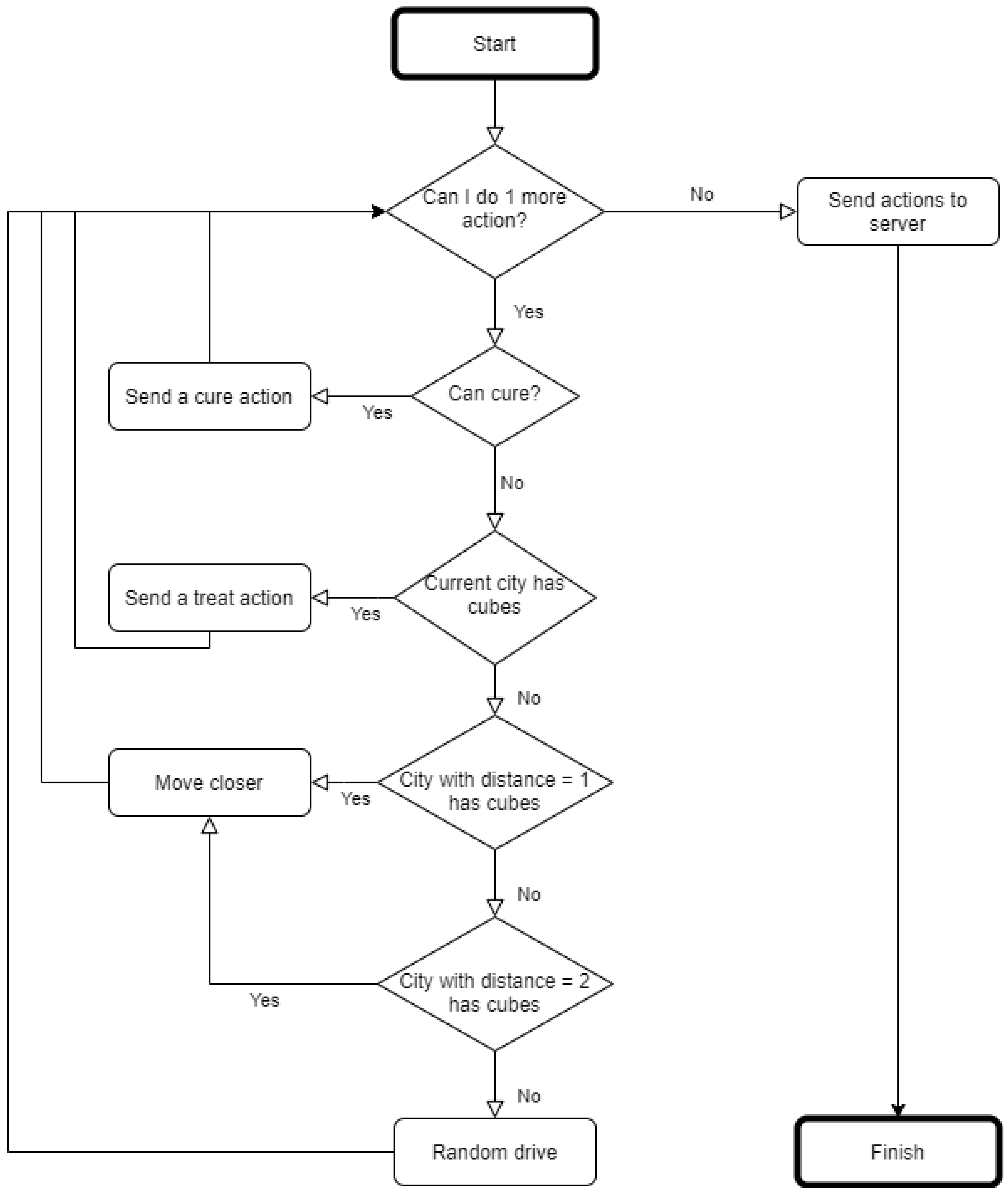


Figure 3.8: Agent_Dummy Flowchart

3.4.2 Agent Simple

After the implementation of the dummy agent, some simple moves were added to the logic of the agent. Moves like going to an RS or the implementation of the special abilities were crucial for wining the game. In this section two different agents were implemented (see algorithm 2).

- This agent is not able to use flight traveling methods (eg. Direct Flight) and can only walk.
- If the agent has the cards needed to cure a disease, the agent walks to the closest Research Station and cures the disease.
- This agent can now build Research Stations.
- The agent can now use the special abilities of the given roles.
- Implements the Dummy logic if nothing all the above do not use all of the 4 actions.
- The agent is not giving or taking any suggestions.

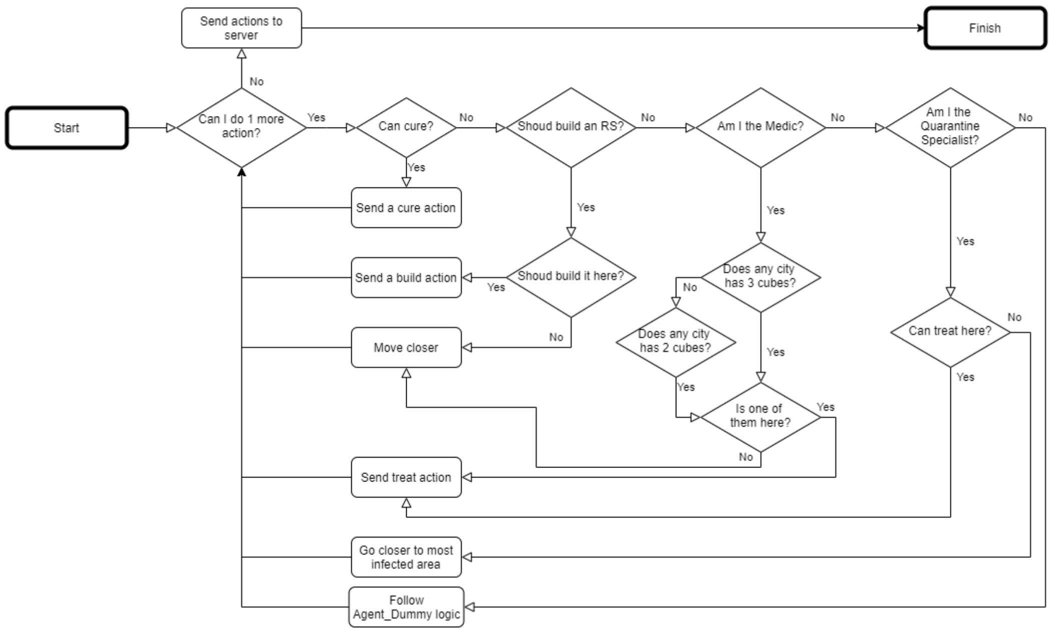


Figure 3.9: Agent_Simple Flowchart

3.4.3 Agent BaseTactic

The next step er took was to add some more functionality to the Agent_Simple to also implement the traveling via flights (Direct Flight, Shuttle Flight and Charter Flight). Adding the flight functionalities and improving some of the already mentioned methods gives us the latest version so far of the tactic that our agent is going to use without using any search algorithms (see algorithm 3).

- The agent has now the flight abilities. He can all the available travel actions of the game.
- The agent can now optimize traveling through cities using a distance map.
- Follows the rest of the Agent_Simple methods.
- The agent is not giving or taking any suggestions.

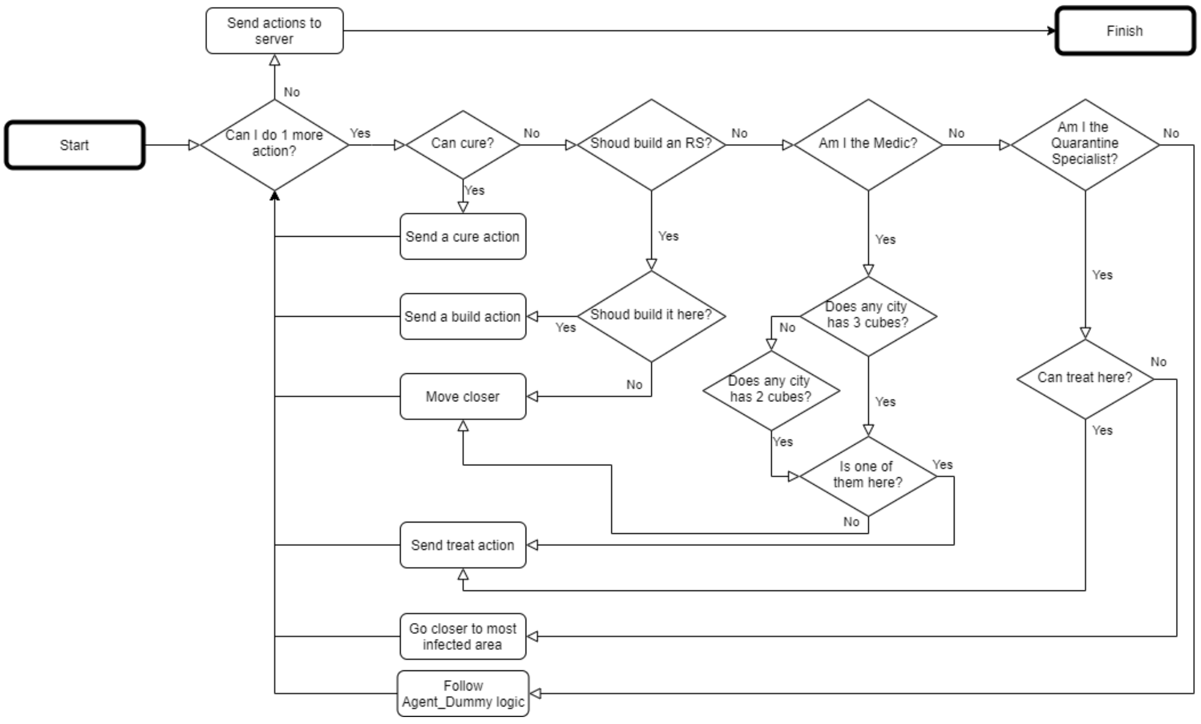


Figure 3.10: Agent_BaseTactic Flowchart

3.4.4 Agent Dynamic

From this point on, various weights were implemented in order to evaluate the different states of the board. Based on these weights our client would decide the actions that are going to be performed in order to achieve the best state possible (see algorithm 4).

- This agent uses dynamic weights for treating cubes.
- This agent uses dynamic weights for preventing outbreaks.
- This agent has optimized traveling using flights between cities.
- This agent has optimized RS placement based on centrality of each city (how many neighbours each city has).
- This agent exploits the advantages of each role.

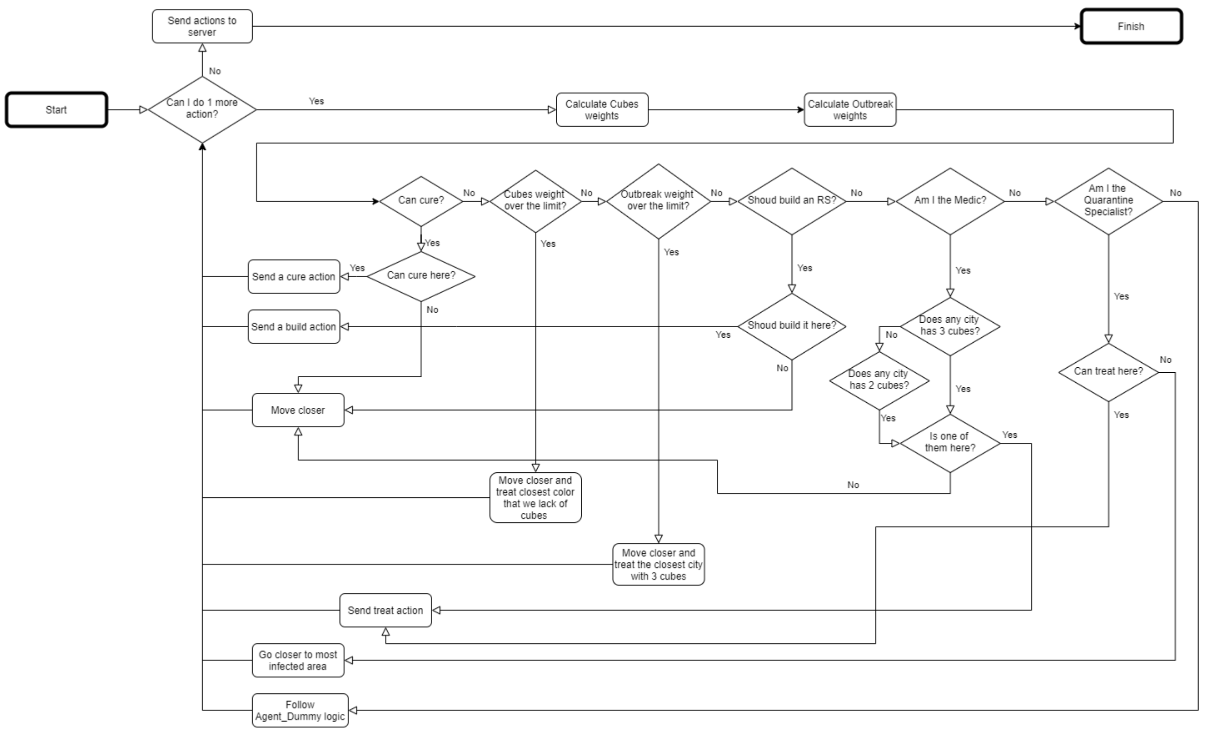

Send actions to server
Finish
Start
Can I do 1 more action?
No
Yes
Calculate Cubes weights
Calculate Outbreak weights
Can cure?
Cubes weight over the limit?
Outbreak weight over the limit?
Shoud build an RS?
Am I the Medic?
Am I the Quarantine Specialist?
Send a cure action
Can cure here?
Send a build action
Shoud build it here?
Does any city has 3 cubes?
Does any city has 2 cubes?
Can treat here?
Move closer
Is one of them here?
Move closer and treat closest color that we lack of cubes
Move closer and treat the closest city with 3 cubes
Send treat action
Go closer to most infected area
Follow Agent_Dummy logic

Figure 3.11: Agent_Dynamic Flowchart

3.4.5 Agent Look-ahead

One issue with the previous agent, that was using the dynamic weights distribution, was the fact that the creation of weights based on future moves was not possible. Thus, we created some more weights to improve the performance of our agent (see algorithm 5). This version, "Agent Look-ahead",

- Uses look-ahead functions for "guessing" the future based on prop abilities of cards that are still in game. Propabilities are based both on the Player Deck as well as on the Infection Deck.
- Uses dynamic weights for treating cubes.
- Uses Dynamic weights for preventing outbreaks.
- Has optimized traveling techniques, same as Agent Dynamic.
- Optimized RS placement based on centrality, same as Agent Dynamic.
- Exploiting the advantages of each role.

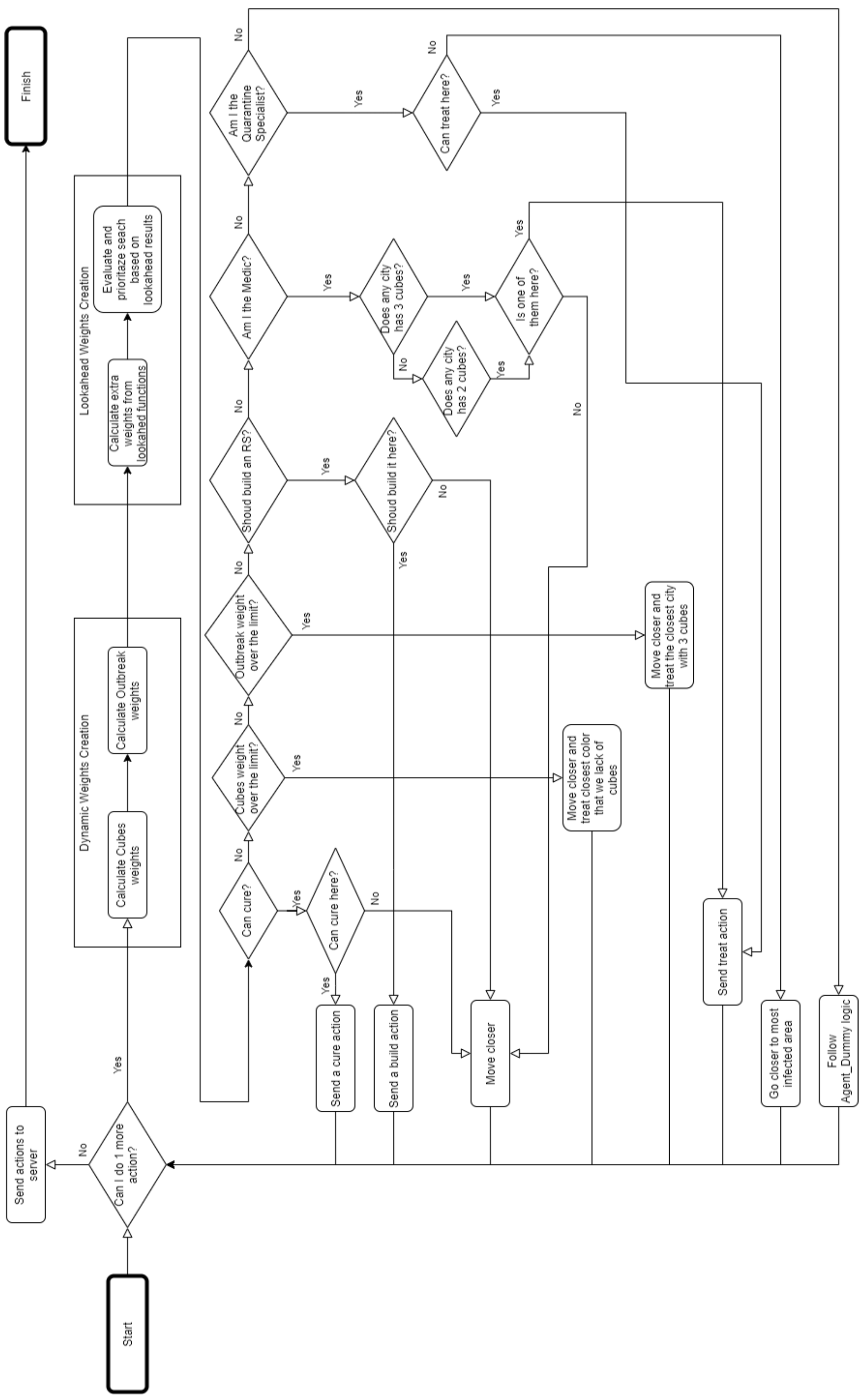

Start
Can I do 1 more action?
No
Send actions to server
Finish
Yes
Dynamic Weights Creation
Calculate Cubes weights
Calculate Outbreak weights
Lookahead Weights Creation
Calculate extra weights from lookahead functions
Evaluate and prioritaze seach based on lookahead results
Can cure?
Yes
Can cure here?
Yes
No
No
Cubes weight over the limit?
Yes
No
Outbreak weight over the limit?
Yes
No
Shoud build an RS?
Yes
Shoud build it here?
Yes
No
No
Am I the Medic?
Yes
Does any city has 3 cubes?
Yes
No
Does any city has 2 cubes?
Yes
Is one of them here?
Yes
No
No
Am I the Quarantine Specialist?
Yes
Can treat here?
Yes
No
No
Move closer and treat closest color that we lack of cubes
Move closer and treat the closest city with 3 cubes
Send a cure action
Send a build action
Move closer
Send treat action
Go closer to most infected area
Follow Agent_Dummy logic

Figure 3.12: Agent_Lookahead Flowchart

3.4.6 Agent HeuristicsCombined

Based on these weights we also built an heuristic function (as described in section 2.3.2). Given the old state of the board and the state for evaluation, the function returns a number that the bigger the number the best the target state is.

The heuristic function that was build was evaluating the board for evaluation based on the following values (see algorithm 6).

- ✓ Cards in hand
- ✓ Possible chained reaction outbreaks
- ✓ Cities with 3 cubes
- ✓ Remaining cubes (color based)
- ✓ Pawn's location on end of turn
- ✓ How many diseases are cured
- ✓ How many diseases are eradicated
- ✓ How many RS are in game

The agent with the look-ahead abilities had far better results, but still sometimes (rarely), agents more simple than the Agent_Lookahead had chosen the best action for a specific round. So, an even more completed evaluation function was built that would be used with an agent that combines all the above techniques and evaluate each one independently.

- Combines tactics and evaluates the outcomes based on the heuristic.
- Finds the best outcome for meta heuristic
- Tactic 1 - Base Tactic
- Tactic 2 - Dynamic weights for outbreaks
- Tactic 3 - Dynamic weights for outbreaks and treating cubes.
- Tactic 4 - Look-ahead functions.
- Evaluates all the suggestions.
- Stores data for player profiling.

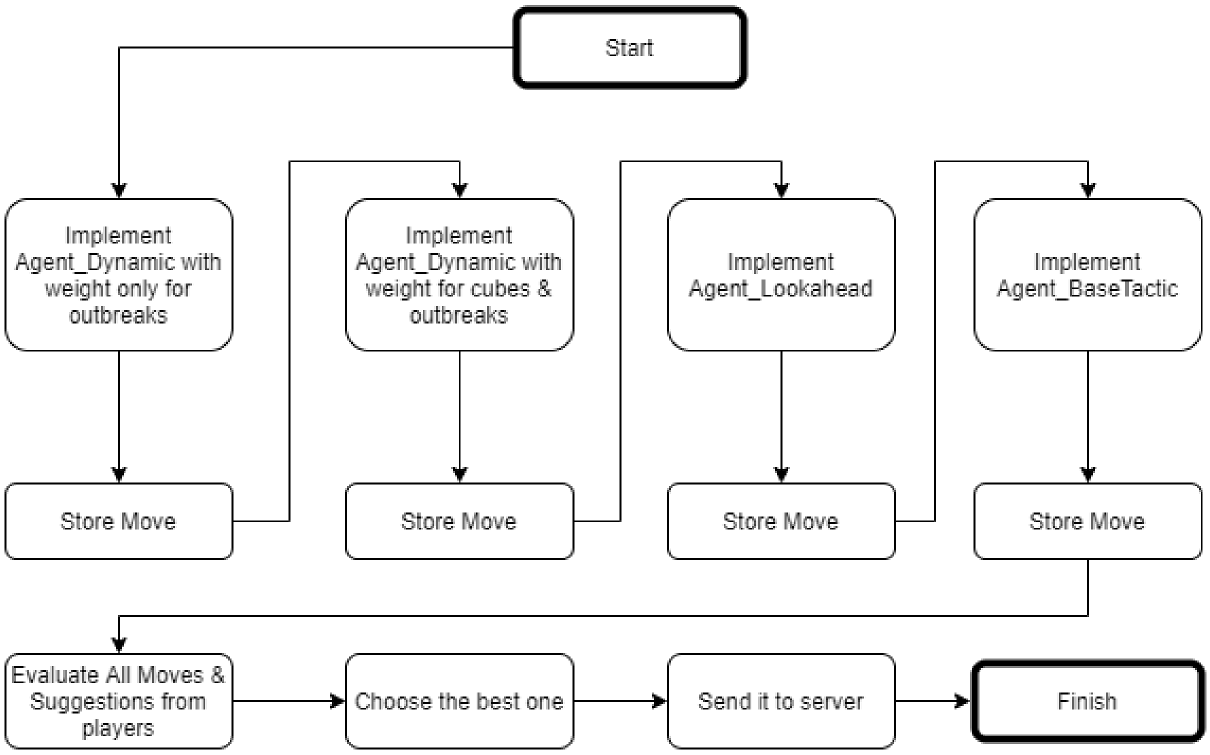

Start
Implement Agent_Dynamic with weight only for outbreaks
Implement Agent_Dynamic with weight for cubes & outbreaks
Implement Agent_Lookahead
Implement Agent_BaseTactic
Store Move
Store Move
Store Move
Store Move
Evaluate All Moves & Suggestions from players
Choose the best one
Send it to server
Finish

Figure 3.13: Agent_HeuristicsCombined Flowchart

3.4.7 Agent Montecarlo Tree Search

Last but not least is the implementation of the agent that will use the MCTS method. The agent will be evaluating states using the evaluation function and will be traversing states using the UCB1 function. See more in section 2.3.2 and in algorithm 7). Thus,

- This agent uses MCTS
- This agent uses the heuristic function for evaluating game states.
- This agent implements UCB1 function for tree traversing
- It explores the whole game tree with depth of 4 actions (max actions per player).
- It does not have a certain time limit, to stop the tree search.

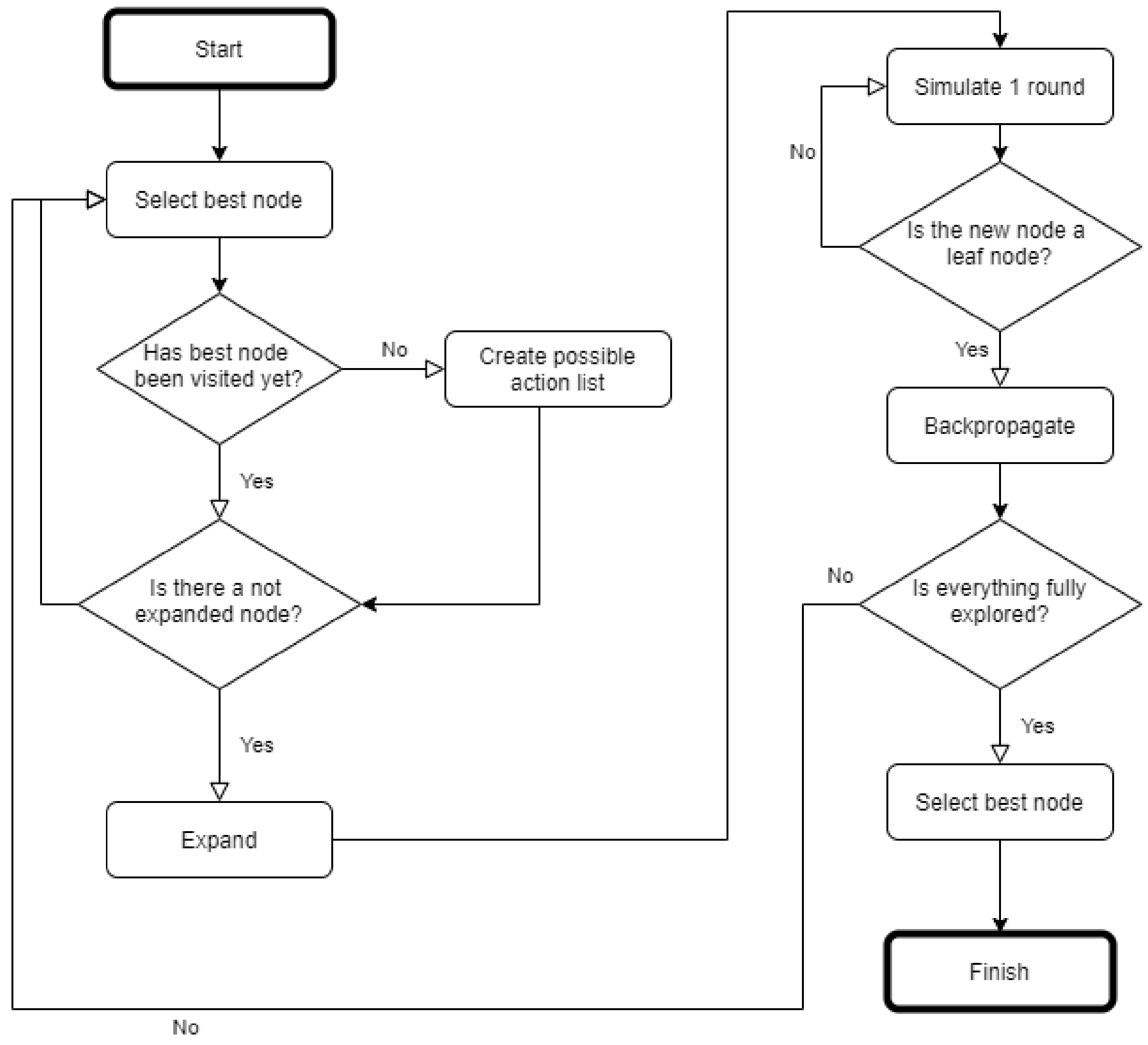

Start
Select best node
Has best node been visited yet?
No
Create possible action list
Yes
Is there a not expanded node?
Yes
Expand
No
Simulate 1 round
No
Is the new node a leaf node?
Yes
Backpropagate
No
Is everything fully explored?
Yes
Select best node
Finish

Figure 3.14: Agent_Montecarlo Flowchart

3.5 Features - Comparison

3.5.1 Player Profiling

The player profiling (section 2.3.4) is a method that can, sometimes, lead us to the most beneficial move, even when this move has not the maximum heuristic value. That is happening, because some complicated thoughts, may contain more than 4 actions, and so, more than one round to be achieved.

For this reason the player profiling is keeping data in order to decide if and how trusted another agent (i.e., one of our teammates) can be. In each round that our players sends an action, we first decide which is the action that we are going to use and if we are going to use any of the suggested moves. If we decide to use a move that was suggested from an other player, we do increase the weight of his next suggestions. That means that the more suggestion of a certain player we decide to follow, the more trustworthy this player becomes. In order to visualize better this behavior, we can take as an example an agent that has already sent us a lot of helpful suggestions. This agent would (eventually - would need more that one game for this to happen) be our “guide” throughout the game, meaning that the weight of his/her suggestion would be so big that we would trust him even in low heuristic value suggestions.

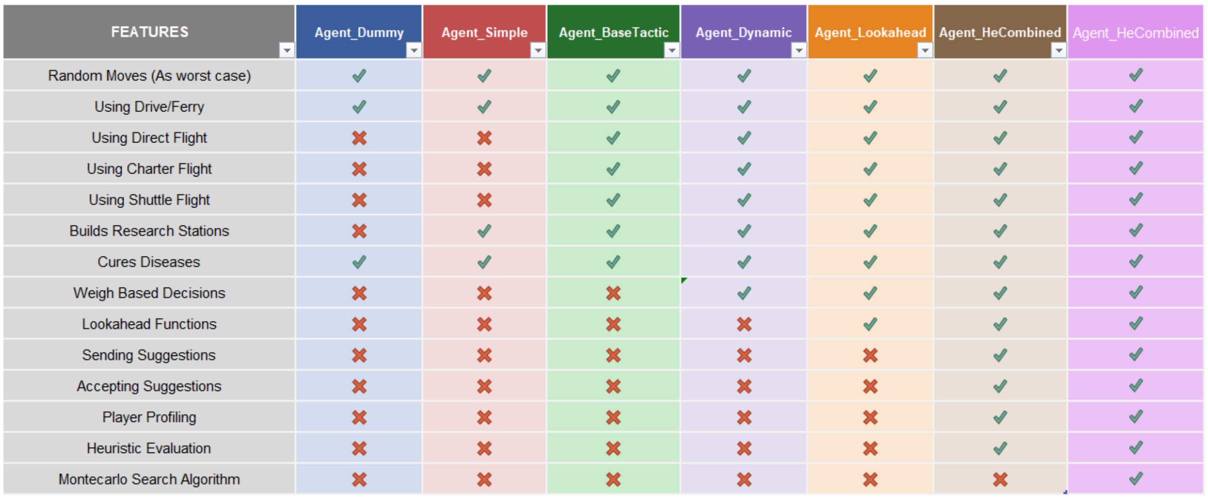
In addition to this behavior, especially for the Agent Heuristics Combined, we do have an internal player profiling. That means that we do keep internally how trusted a tactic can be. This agent is using a tactic that attempts to keep and update beliefs about how well an internal tactic we use does, so if a certain one keeps leading us in better choices, eventually we will keep following this tactic without evaluating the rest of the tactics so much.

3.5.2 Suggestion Making

Our agents have also a suggestion making ability. In order to achieve that, each agent is “seeing” through the eyes of the player that is currently playing, and decides for an action based on his beliefs - tactics. So the suggestion that is being sent, is the action that this player would do if he/she was in the exact same situation. We have to note that agents Simple and BaseTactic do not have the suggestion making ability.

3.5.3 Comparison

So, after describing each agent now it is time to gather up all agents and compare their features. In the following table we can see the basic features mentioned above, and which agents do implement each feature.

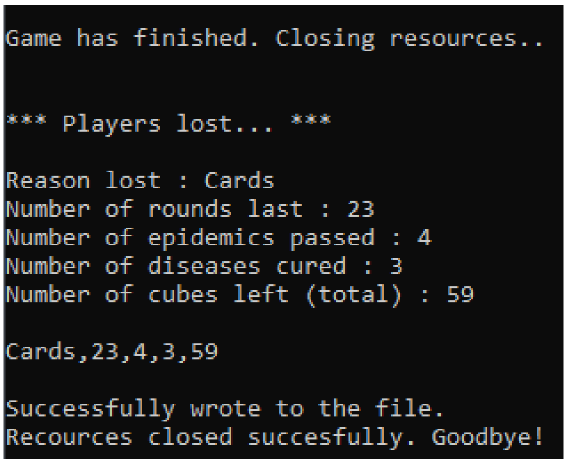


FEATURES	Agent_Dummy	Agent_Simple	Agent_BaseTactic	Agent_Dynamic	Agent_Lookahead	Agent_HeCombined	Agent_HeCombined
Random Moves (As worst case)	✓	✓	✓	✓	✓	✓	✓
Using Drive/Ferry	✓	✓	✓	✓	✓	✓	✓
Using Direct Flight	✗	✗	✓	✓	✓	✓	✓
Using Charter Flight	✗	✗	✓	✓	✓	✓	✓
Using Shuttle Flight	✗	✗	✓	✓	✓	✓	✓
Builds Research Stations	✗	✓	✓	✓	✓	✓	✓
Cures Diseases	✓	✓	✓	✓	✓	✓	✓
Weigh Based Decisions	✗	✗	✗	✓	✓	✓	✓
Lookahead Functions	✗	✗	✗	✗	✓	✓	✓
Sending Suggestions	✗	✗	✗	✗	✗	✓	✓
Accepting Suggestions	✗	✗	✗	✗	✗	✓	✓
Player Profiling	✗	✗	✗	✗	✗	✓	✓
Heuristic Evaluation	✗	✗	✗	✗	✗	✓	✓
Montecarlo Search Algorithm	✗	✗	✗	✗	✗	✗	✓

Figure 3.15: Features Compared

3.6 Data For Evaluation

Some data gathering tools needed also to be implemented in order to be able to test the results in large scale. For that reason the server gathers cumulative stats about each game and writes them into a .csv file at the end of the game.



```
Game has finished. Closing resources..


*** Players lost... ***

Reason lost : Cards
Number of rounds last : 23
Number of epidemics passed : 4
Number of diseases cured : 3
Number of cubes left (total) : 59

Cards,23,4,3,59

Successfully wrote to the file.
Recources closed succesfully. Goodbye!
```

Figure 3.16: End of game message & stats

The following stats are presented in the server's console

- **Game's results** : If players won or lost.

- **Reason of defeat** : Could be “cards” if no more cards to draw are available, “outbreaks” if max outbreaks number is reached, “cubes” if there are not enough cubes to place in a city or nothing if players won the game.
- **Number of epidemics passed** : The number of epidemic cards that the players have drawn during this game.
- **Number of diseases cured** : The number of the diseases that players managed to cured during the game.
- **Number of cubes left** : The number of the total cubes that were left unplaced at the end of the game. A division by 4 could give an average number of how many cubes are left from each color.

Then the server writes in the given .csv file that need manual input in the server.class from the user. **Each row of the file represents one game**.

3.7 Agent’s pseudocodes collection

Algorithm 1 Dummy agent

```
while 1 more move is possible do
    if can cure then
        send cure action
    else if can treat here then
        send treat action
    else if city with distance = 1 has cubes then
        move there
    else if city with distance = 2 has cubes then
        move closer
    else
        move randomly
    end if
end while
send actions to server
```

Algorithm 2 Agent Simple

```
 1: while 1 more move is possible do
 2:     if can cure then
 3:         send cure action
 4:     else if should build an RS then
 5:         if should build it here then
 6:             send build action
 7:         else
 8:             move closer
 9:         end if
10:     else if am I the medic then
11:         if yes then
12:             if any city has 3 cubes then
13:                 if it is here then
14:                     send treat action
15:                 else
16:                     move closer
17:                 end if
18:             else if any city has 2 cubes then
19:                 if it is here then
20:                     send treat action
21:                 else
22:                     move closer
23:                 end if
24:             end if
25:         end if
26:     else if am I the quarantine specialist then
27:         if yes then
28:             if can treat here then
29:                 send treat action
30:             else
31:                 move closer to most infected area
32:             end if
33:         end if
34:     else
35:         follow dummy logic
36:     end if
37: end while
38: send actions to server
```

Algorithm 3 Agent BaseTactic

```
while 1 more move is possible do
    if can cure then
        send cure action
    else if should build an RS then
        if should build it here then
            send build action
        else
            move closer
        end if
    else if am I the medic then
        if yes then
            if any city has 3 cubes then
                if it is here then
                    send treat action
                else
                    move closer
                end if
            else if any city has 2 cubes then
                if it is here then
                    send treat action
                else
                    move closer
                end if
            end if
        end if
    else if am I the quarantine specialist then
        if yes then
            if can treat here then
                send treat action
            else
                move closer to most infected area
            end if
        end if
    else
        follow dummy logic
    end if
end while
send actions to server
```

Algorithm 4 Agent Dynamic

```
while 1 more move is possible do
    calculate cubes weights
    calculate outbreak weights
    if can cure then
        send cure action
    else if cubes weights over limit then
        move to closer colored cube
    else if outbreak weights over limit then
        move to closer 3cube city
    else
        follow base tactic logic
    end if
end while
send actions to server
```

Algorithm 5 Agent Look-Ahead

```
while 1 more move is possible do
    calculate cubes weights
    calculate outbreak weights
    calculate lookahead weights
    evaluate priority of weighs based on predictions
    if can cure then
        send cure action
    else if max weight ¿ threshold then
        do actions to fix it
    else
        follow base tactic logic
    end if
end while
send actions to server
```

Algorithm 6 Agent HueuristicsCombined

```
1: implement Dynamic with only outbreak weight
2: store move
3: implement Dynamic with all weights
4: store move
5: implement Lookahead
6: store move
7: implement BaseTactic
8: store move
9: decide based on heuristic function
10: send actions to server
```

Algorithm 7 Agent Monte-Carlo Tree Search

```
1: create root node $u_0$ with state $s_0$
2: while true do
3:    Make root node the current node
4:    if is current node a leaf node then
5:       if node has not been visited then
6:          $U_1 \Leftarrow treePolicy(u_0)$
7:          $\Delta \Leftarrow defaultPolicy(s(u_i))$
8:          $backup(u_i; \Delta)$
9:       else
10:         Create all possible child nodes (Expand)
11:         Make first child the current node
12:         $U_1 \Leftarrow treePolicy(u_0)$
13:         $\Delta \Leftarrow defaultPolicy(s(u_i))$
14:         $backup(u_i; \Delta)$
15:      end if
16:   else
17:      Make current the child of current with max UCB1
18:      Go to step 4
19:   end if
20:   if game ended then
21:      break
22:   end if
23: end while
24: return $a(bestChild(u_0; 0))$
```

The base structure of the Agent Monte-Carlo Tree Search is further explained below.

- line 2: The break condition may be something else than true. For example we could break the loop based on time or in nodes that were created. For our case wee are breaking the loop when every possible node is explored.
- line 6,12: We are using our tree policy for the simulation that is going through random choices till we reach and OEG (End Of Game) node.
- line 7,13: Based on our default policy (heuristics for our example) we calculate how close (or far) we are from winning the game. So the default policy is the utility function used for our game.
- line 8,14: We are backpropagating our results increasing the total utility value of each parent node and the times that the node has been visited in order to (re)calculate the UCB1 value.
- line 10: For the expansion phase all possible actions are creates as children to the current node. For simplicity reasons we excluded the "Action Pass" from the possible actions.
- line 17: The best node is calculated based on the UCB1 value of each separate node.
- line 24: Node with the best UCB1 value is returned as the best possible node.

Chapter 4

Experiments and Results

In this chapter we use the Pandemic server and related game infrastructure we built, in order to test and compare the performance of the various Pandemic agents we developed.

For the evaluation and comparison among agents, we use the following metrics to evaluate the performance of each one, in various situations and domains.

- **Win Rate**: This is the percentage of the games that the agent won among the games it participated in.
- **Outbreaks Rate**: This is the percentage of the games that the agent lost the game, due to over-passing the Outbreaks limit.
- **Cubes Rate**: This is the percentage of the games that the agent lost the game, due the lack of Cubes.
- **Cards Rate**: This is the percentage of the games that the agent lost the game, due the lack of Cards.

We also use the following stats for each game's result

  - **Average Cured**: An average estimation of how many diseases the agents were able to cure in each game. The more diseases are cured the closer the players are to win the game.
  - **Average Cubes Remaining**: An average estimation of how many cubes were still unused in the end of the game. Be aware that this number represents the

average of the number of the cubes that are unused **in each cube pile** and not the average of all of the unused cubes in game. The closer this value goes to zero, the closer the players are to lose the game. So having a big average value of remaining cubes, means a better cube handling from the clients.

  - **Average Rounds Lasted**: An average estimation of how many rounds the agents lasted before the game ends. One round represents the action of one player and not the actions of all of the players in game (in a 2-players game, tit is the same). This variable is also representing how good the players were in the disease spreading process. The more rounds lasted, means also good distribution of importance given to each color.

Following up we will conduct several experiments and discuss the results taken from each one.

4.1 Calculating bounds

Having a different version of this board game may lead to big differences performance-wise. When the ability of the players to trade cards between them is not active, as is the current implementation in which this ability is turned off, that also means that the possibilities of winning are also quite lower. Gathering a set of cards during the game may not be possible for every game, whichever tactic the agent may use.

For that reason, in addition to the agents presented in the previous chapter, we created a completely dummy agent, that was not moving at all, but we also changed some parameters of the server in order to create the “God Mode”. This mode means that the agents are now unable to lose due to cubes or outbreaks, because the limit for both of them is now infinite. The dummy agent that was created was just standing there in Atlanta, without moving and whenever a set of card was in hand, the agent was immediately treating the disease. Using the following pattern we conducted a set of experiments to calculate the upper bound for winning this game and the results are the following.

- In domains with **2 players**, the upper bound is close to **100%**
- In domains with **3 players**, the upper bound is close to **98%**

- In domains with **4 players**, the upper bound is close to **78%**

The **upper bound**, is reducing when the number of players is increasing. That is happening due to the lack of the ability of trading cards between players. More players means less chances for all the four sets of cards of each color, to end up in a player's in order to cure the disease.

For the **lower bound**, as expected, nothing changes and it is **still 0%**

4.2 Experiments with Homogeneous Settings

In this section we evaluate our various algorithms in homogeneous settings with respect to strategies used. That is, all competing agents in a given game are assumed to be employing the same strategy. As explained below, the experiments conducted evaluate a given strategy in terms of its success in environments with increasing difficulty. We begin in 4.2.1 by taking a look on how different agent strategies compare against each other in domains with increasing difficulty; then in Section 4.2.2 we anchor our analysis on the performance of each individual strategy in turn.

4.2.1 Comparing Strategies while Increasing Game Difficulty

Here we take a look on how different agent strategies compare against each other in domains with increasing difficulty. The agents are going to be evaluated in scenarios with varying number of players, but every time just one kind of agent is tested, that is, as mentioned above, all competing agents in a game are assumed to be of the same kind, thus the domain is a homogeneous one. In order to increase or decrease the difficulty of the game we change the number of epidemic cards that are in game. More epidemics lead us to a harder game and vice versa.

Results using **4 Epidemics** cards (Figure 4.1)

Based on these results the Agent Combined is the one that has the best scores in games of 2 and 3 players. For the 3 players game we notice that the Base Tactic Agent is ahead of the Combined, and the one with the best score is the Agent Look-ahead. This results give rise to the following question. How can an agent that contains both of the 2 agents that are ahead, to have less score than them?

The answer for this question is a characteristic that can differentiate humans from AI agents. Humans can easily look ahead and calculate long terms effects in contrast with the agents. In order to predict the future agents must either have a really good knowledge on the domain that they are playing in, or have resources to search the game tree. This perfect example can show us how choices that may seem good for the time being (good results of a heuristic) may be less useful for us, in the long term.

Another interesting fact about these results is that the Monte Carlo Agent falls behind when compared with all of the other agents. Furthermore, we do not present results for the Monte Carlo agent with fewer players that 4. But why is that?

To answer that question, we need to keep in mind that this is a game that contains a lot of unknown factors (both decks), so this makes the game more complex and increases the branching factor of the game tree. So, in order to create an agent that can give us back some results, we had to reduce the branching factor of the game, so moves that greatly increase the branching factor of the game tree (shuttle flight, Operation Expert Travel) were not implemented for the MCTS Agent and that is why we can not really compare it with the rest of the agents. In addition to that, when playing with fewer that 4 players, the branching factor is also greatly increased, because each player now holds more cards in their hands. For that reason games with 2 or 3 players, were not tested using the MCTS Agent.

Based on the above, from this point forward we will be using the 3 best agents of this domain (Base-Tactic, Look-ahead & Combined) in order to conduct the rest of the experiments for this game. These agents have good overall results on all of these 3 domains.

Results using **5 Epidemics** cards (increased game difficulty - Figure 4.2)

Playing in a harder domain now, with a quick look on the results, we can see that the Agent Combined is now first in every kind of game. Increasing the difficulty of our domain resulted to the agents that were developed in a specific environment to be less efficient. That can be seen from the decreased performance of Agent Lookahed (which was developed initially for a 2-Players game), that lost the first place to the Agent Combined.

Having said that, impressive results are seen for the Agent Base Tactic, that was developed with no use of a heuristic function, in a much more simple way. By having a static technique, this agent manages to achieve a ratio of 18% wins in a 5-Epidemic and 4-Players game.

Results using **6 Epidemics** cards (even great game difficulty - Figure 4.3)

Last but not least we are going to discuss about the results of our most difficult domain in this game, the 6-Epidemic game (aka heroic difficulty - in game). Once again the combined agent exhibits outstanding results for this game. The average value for 2-player games is close to 50% (specifically 46%), making it nearly 1 out of 2 games won.

For the 2 player domain the two other agents have also great result, with the agent Base-Tactic outperforming Look-ahead. For the rest of the domains (3 & 4 players game), again the Combined agent is the one that takes the lead. The rest of the 2 agents produce similar results in this domain, with the Look-ahead agent to have some improved losses results over the Base-Tactic one.

4.2.2 Focusing on the performance of each individual strategy

Here we take a closer look on the performance of each individual strategy while increasing game difficulty in a homogeneous setting. We have to mention that all the following experiments are the same as above, but we examine them from a different perspective, "anchoring" our analysis on the performance of each individual agent in turn.

Results using **Agent Base-Tactic** (Figure 4.4)

Looking at the stats of the agent Base-Tactic, we can notice that the results are really good. Of course, by not being able to change it's approach, based on the game's current parameter is a reason that this agent has sometimes lower results that it's counterparts. That can be noticed, compared with the following agents results (figures 4.5 & 4.6).

This agent excels (as a static tactic agent) in games of 2 players, where his results are even better than the Look-ahead's ones (see figure 4.5). That is because this agent was developed in a 2-player game domain, so it's well suited for such environment.

Results using **Agent Look-ahead** (Figure 4.5)

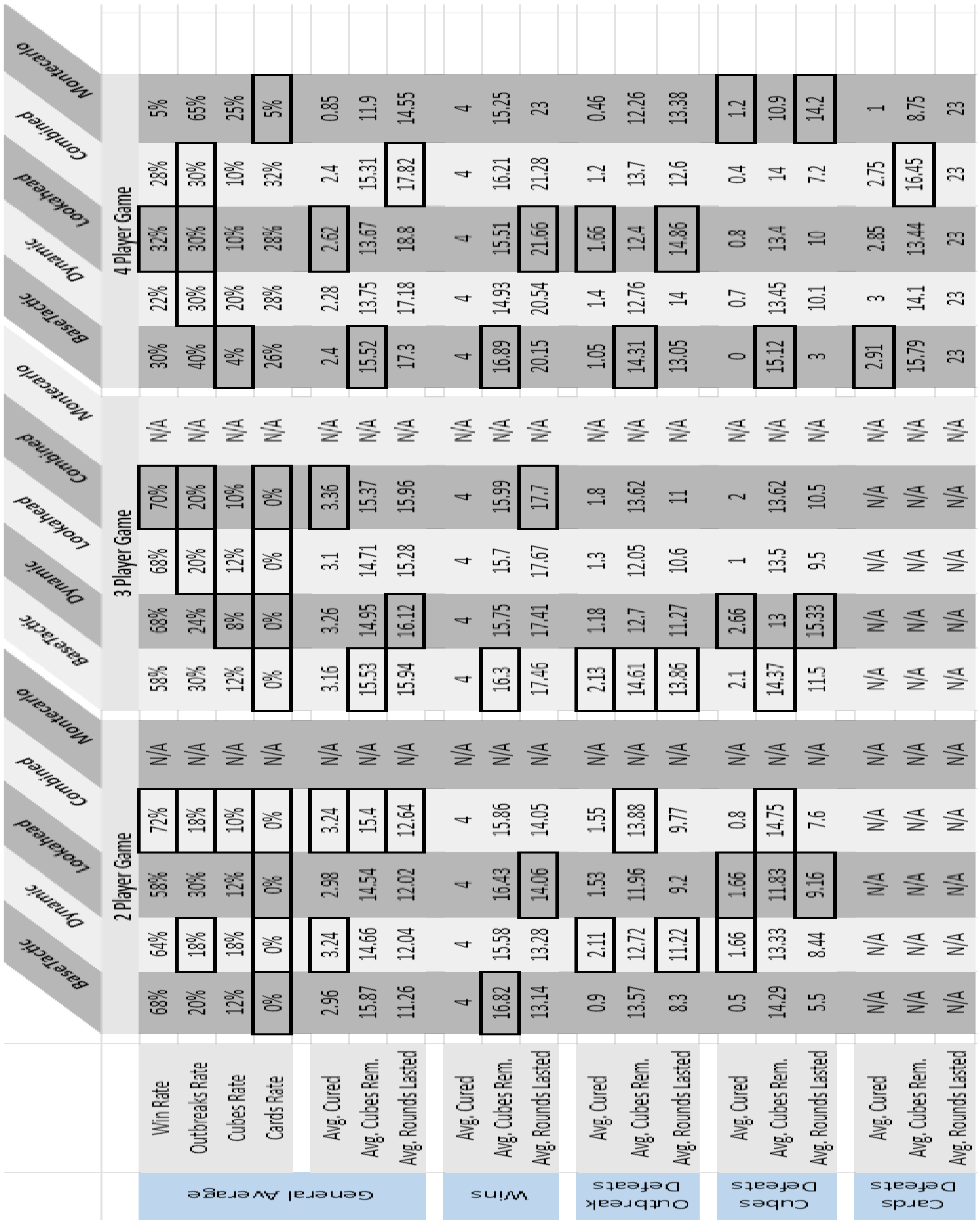
The Look-ahead agent is the second agent that had the ability to change the thresholds of it is weights as the game was moving on. It was also the first agent that was "thinking" about what the future cards will be and using it as one of the parameters that will affect the last decision.

Looking at the results it is improved compared to the Base-Tactic agent in every domain except the 2-players game (for the reason mentioned before). By keeping the Cubes rate low in every domain, we can understand that the handling of the cubes treatment is quite effective and it minimizes the losses due to outbreaks. When losing due to cubes (meaning that all the available cubes are used, and one of the diseases went out of control) the average diseases treated rate is low, but also the average rounds lasted is low, meaning that a lot of epidemic cards were gathered at the top of the deck making the game's even more increased (also know as bad luck).

Results using **Agent Combined** (Figure 4.6)

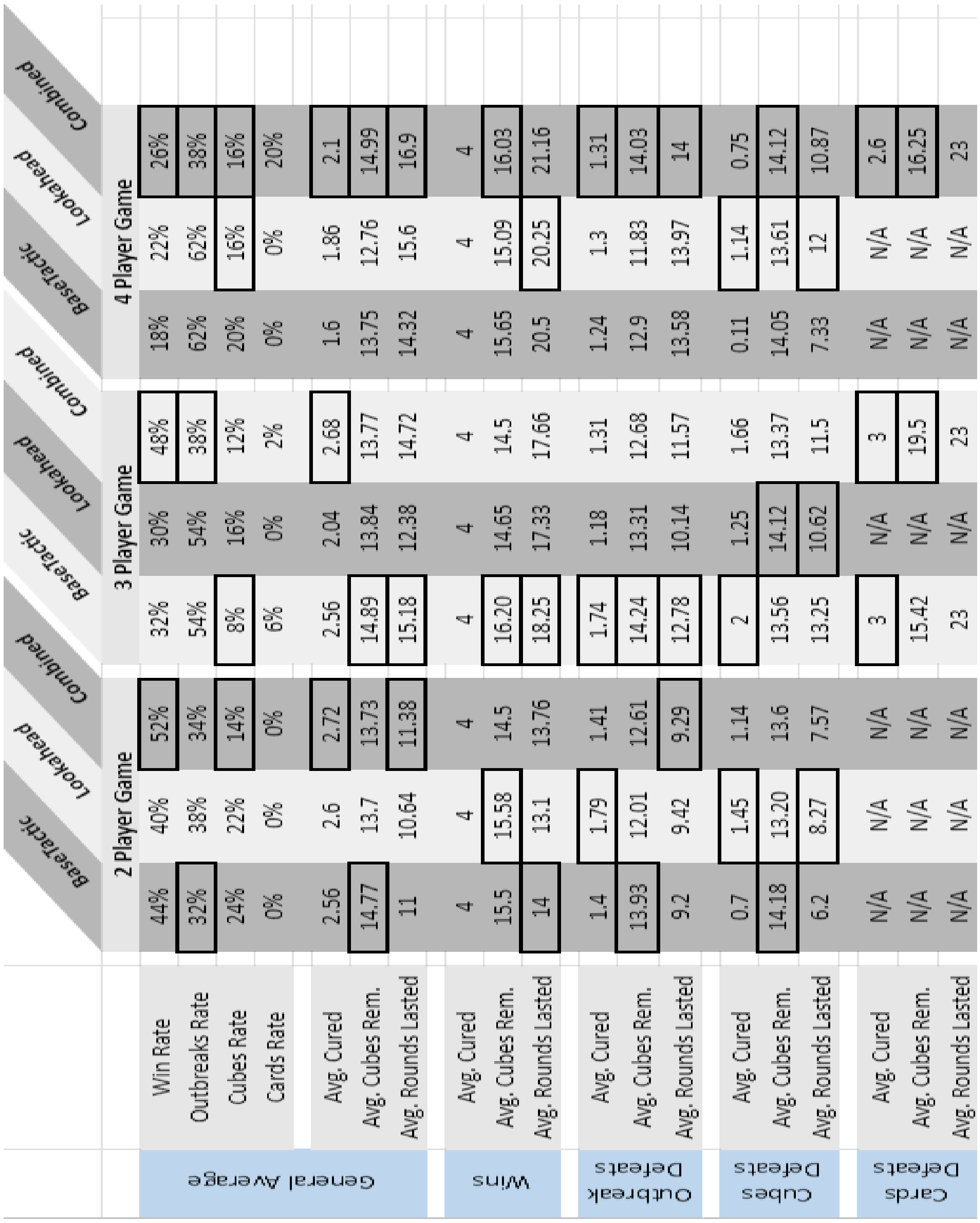
This is the agent that performs best, combining as it does the techniques of the previously implemented agents. Based on the results, the agent Combined is dominant in the most metrics, proving that the heuristic functions may be really useful, but they may also have performance issues - miscalculations in some domains.

Other than that, the rest of the results are very good. With a maximum winning rate of 72% in simple domains, and 12% in the hardest domain of the game, this agent produces really satisfying results.



		2 Player Game					3 Player Game					4 Player Game				
		BaseTactic	Dynamic	Lookahead	Combined	Montecarlo	BaseTactic	Dynamic	Lookahead	Combined	Montecarlo	BaseTactic	Dynamic	Lookahead	Combined	Montecarlo
General Average	Win Rate	68%	64%	58%	72%	N/A	58%	68%	68%	70%	N/A	30%	22%	32%	28%	5%
	Outbreaks Rate	20%	18%	30%	18%	N/A	30%	24%	20%	20%	N/A	40%	30%	30%	30%	65%
	Cubes Rate	12%	18%	12%	10%	N/A	12%	8%	12%	10%	N/A	4%	20%	10%	10%	25%
	Cards Rate	0%	0%	0%	0%	N/A	0%	0%	0%	0%	N/A	26%	28%	28%	32%	5%
	Avg. Cured	2.96	3.24	2.98	3.24	N/A	3.16	3.26	3.1	3.36	N/A	2.4	2.28	2.62	2.4	0.85
	Avg. Cubes Rem.	15.87	14.66	14.54	15.4	N/A	15.53	14.95	14.71	15.37	N/A	15.52	13.75	13.67	15.31	11.9
	Avg. Rounds Lasted	11.26	12.04	12.02	12.64	N/A	15.94	16.12	15.28	15.96	N/A	17.3	17.18	18.8	17.82	14.55
Wins	Avg. Cured	4	4	4	4	N/A	4	4	4	4	N/A	4	4	4	4	4
	Avg. Cubes Rem.	16.82	15.58	16.43	15.86	N/A	16.3	15.75	15.7	15.99	N/A	16.89	14.93	15.51	16.21	15.25
	Avg. Rounds Lasted	13.14	13.28	14.06	14.05	N/A	17.46	17.41	17.67	17.7	N/A	20.15	20.54	21.66	21.28	23
Outbreak Defeats	Avg. Cured	0.9	2.11	1.53	1.55	N/A	2.13	1.18	1.3	1.8	N/A	1.05	1.4	1.66	1.2	0.46
	Avg. Cubes Rem.	13.57	12.72	11.96	13.88	N/A	14.61	12.7	12.05	13.62	N/A	14.31	12.76	12.4	13.7	12.26
	Avg. Rounds Lasted	8.3	11.22	9.2	9.77	N/A	13.86	11.27	10.6	11	N/A	13.05	14	14.86	12.6	13.38
Cubes Defeats	Avg. Cured	0.5	1.66	1.66	0.8	N/A	2.1	2.66	1	2	N/A	0	0.7	0.8	0.4	1.2
	Avg. Cubes Rem.	14.29	13.33	11.83	14.75	N/A	14.37	13	13.5	13.62	N/A	15.12	13.45	13.4	14	10.9
	Avg. Rounds Lasted	5.5	8.44	9.16	7.6	N/A	11.5	15.33	9.5	10.5	N/A	3	10.1	10	7.2	14.2
Cards Defeats	Avg. Cured	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	2.91	3	2.85	2.75	1
	Avg. Cubes Rem.	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	15.79	14.1	13.44	16.45	8.75
	Avg. Rounds Lasted	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	23	23	23	23	23

Figure 4.1: Homogeneous results using 4 Epidemic cards (50 experiments)

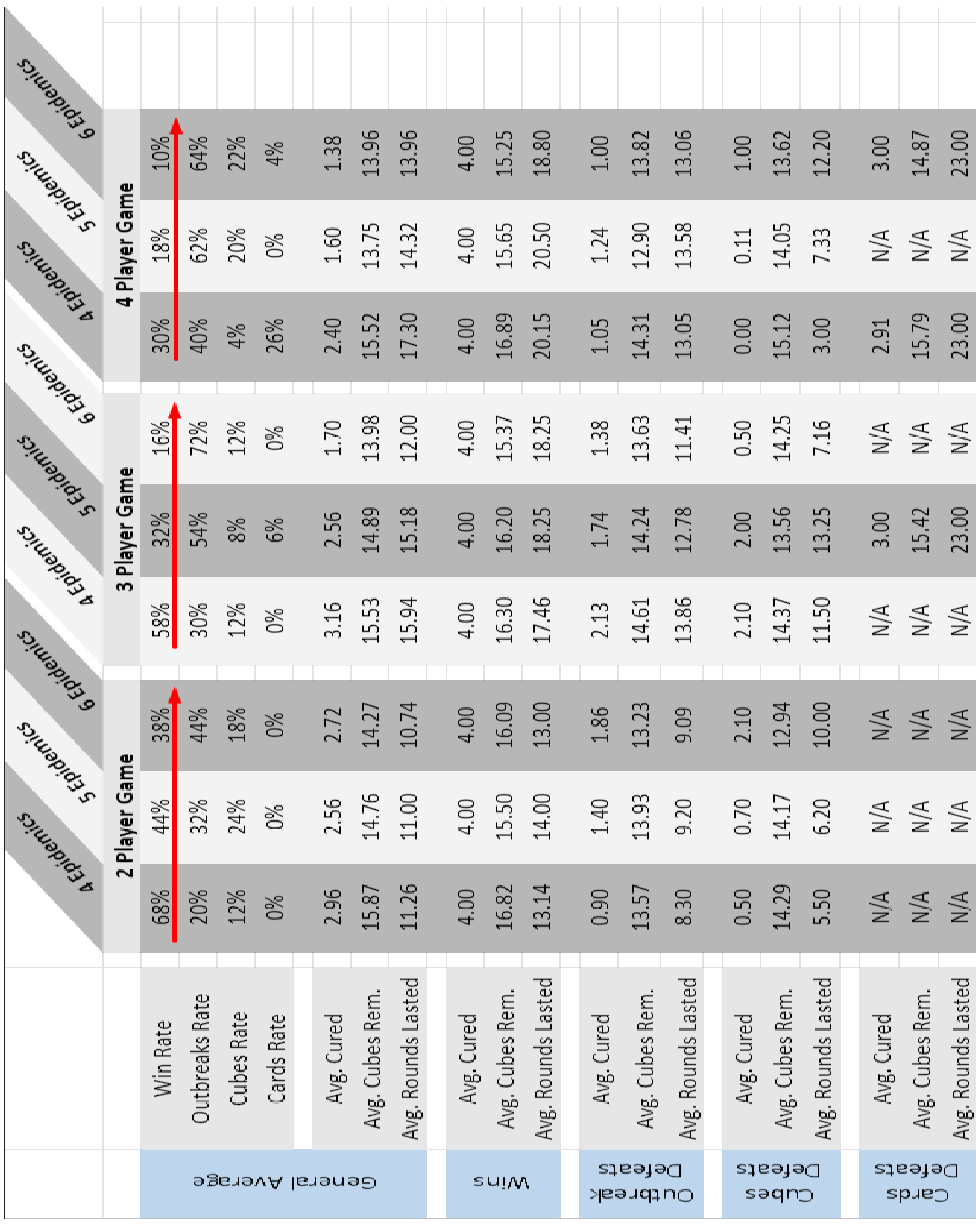


		2 Player Game			3 Player Game			4 Player Game		
		BaseTactic	Lookahead	Combined	BaseTactic	Lookahead	Combined	BaseTactic	Lookahead	Combined
General Average	Win Rate	44%	40%	52%	32%	30%	48%	18%	22%	26%
	Outbreaks Rate	32%	38%	34%	54%	54%	38%	62%	62%	38%
	Cubes Rate	24%	22%	14%	8%	16%	12%	20%	16%	16%
	Cards Rate	0%	0%	0%	6%	0%	2%	0%	0%	20%
	Avg. Cured	2.56	2.6	2.72	2.56	2.04	2.68	1.6	1.86	2.1
	Avg. Cubes Rem.	14.77	13.7	13.73	14.89	13.84	13.77	13.75	12.76	14.99
	Avg. Rounds Lasted	11	10.64	11.38	15.18	12.38	14.72	14.32	15.6	16.9
Wins	Avg. Cured	4	4	4	4	4	4	4	4	4
	Avg. Cubes Rem.	15.5	15.58	14.5	16.20	14.65	14.5	15.65	15.09	16.03
	Avg. Rounds Lasted	14	13.1	13.76	18.25	17.33	17.66	20.5	20.25	21.16
Outbreak Defeats	Avg. Cured	1.4	1.79	1.41	1.74	1.18	1.31	1.24	1.3	1.31
	Avg. Cubes Rem.	13.93	12.01	12.61	14.24	13.31	12.68	12.9	11.83	14.03
	Avg. Rounds Lasted	9.2	9.42	9.29	12.78	10.14	11.57	13.58	13.97	14
Cubes Defeats	Avg. Cured	0.7	1.45	1.14	2	1.25	1.66	0.11	1.14	0.75
	Avg. Cubes Rem.	14.18	13.20	13.6	13.56	14.12	13.37	14.05	13.61	14.12
	Avg. Rounds Lasted	6.2	8.27	7.57	13.25	10.62	11.5	7.33	12	10.87
Cards Defeats	Avg. Cured	N/A	N/A	N/A	3	N/A	3	N/A	N/A	2.6
	Avg. Cubes Rem.	N/A	N/A	N/A	15.42	N/A	19.5	N/A	N/A	16.25
	Avg. Rounds Lasted	N/A	N/A	N/A	23	N/A	23	N/A	N/A	23

Figure 4.2: Homogeneous results using 5 Epidemic cards (50 experiments)

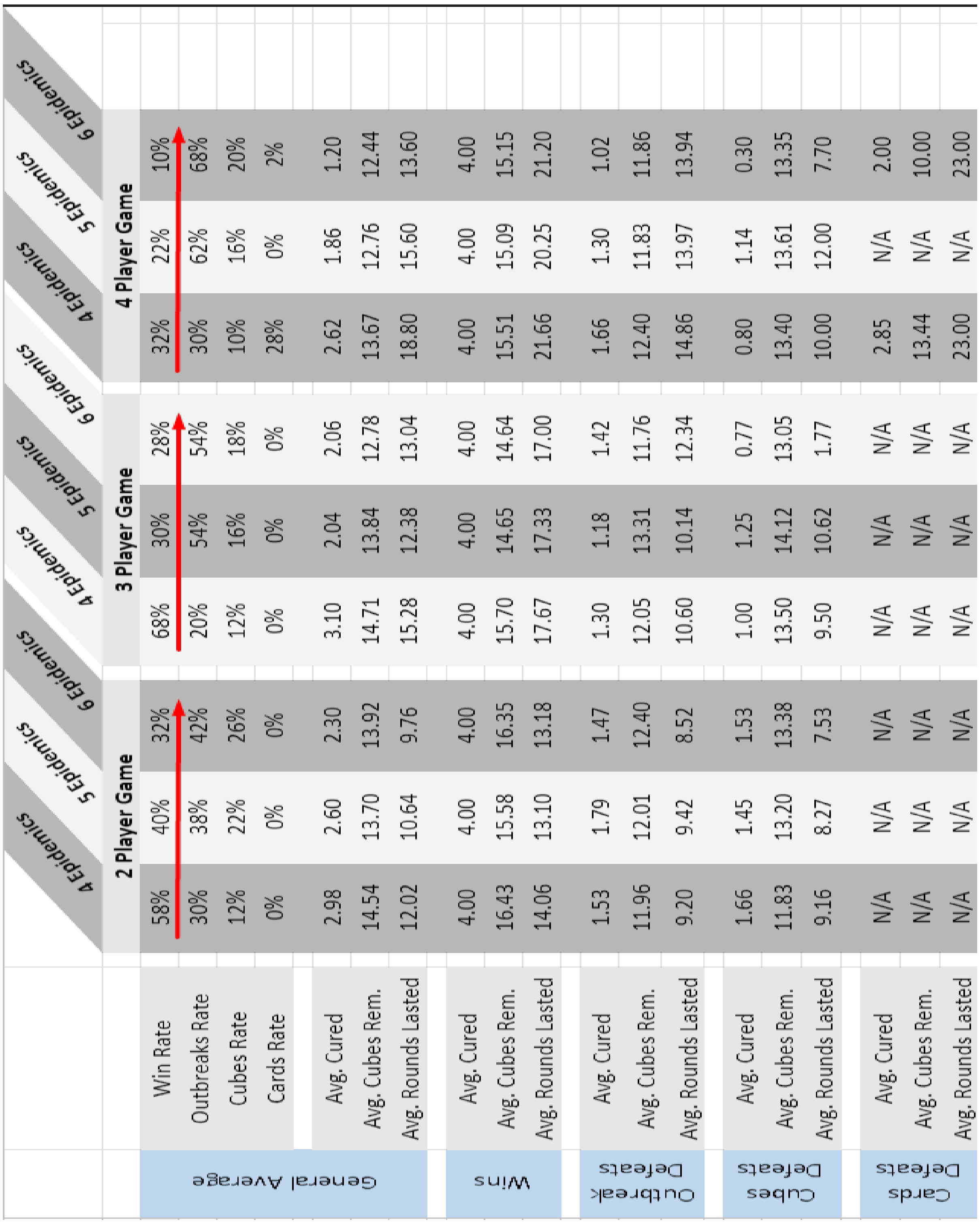
		2 Player Game			3 Player Game			4 Player Game		
		BaseTactic	Lookahead	Combined	BaseTactic	Lookahead	Combined	BaseTactic	Lookahead	Combined
General Average	Win Rate	38%	32%	**46%**	16%	**28%**	**28%**	10%	10%	**12%**
	Outbreaks Rate	44%	42%	**28%**	72%	**54%**	60%	64%	68%	**60%**
	Cubes Rate	**18%**	26%	26%	**12%**	18%	**12%**	22%	20%	**18%**
	Cards Rate	0%	0%	0%	0%	0%	0%	4%	**2%**	10%
	Avg. Cured	**2.72**	2.3	2.6	1.7	2.06	**2.12**	**1.38**	1.2	1.34
	Avg. Cubes Rem.	14.27	13.92	**14.84**	**13.98**	12.78	13.73	**13.96**	12.44	13.16
	Avg. Rounds Lasted	10.74	9.76	**10.82**	12	13.04	**13.12**	**13.96**	13.6	13.84
Wins	Avg. Cured	4	4	4	4	4	4	4	4	4
	Avg. Cubes Rem.	16.09	**16.35**	16.16	**15.37**	14.64	15.21	**15.25**	15.15	14.08
	Avg. Rounds Lasted	13	13.18	**14.04**	**18.25**	17	17.76	18.8	**21.2**	20.16
Outbreak Defeats	Avg. Cured	**1.86**	1.47	1.64	1.38	**1.42**	1.36	1	**1.02**	0.9
	Avg. Cubes Rem.	13.23	12.4	**13.41**	**13.63**	11.76	13.19	**13.82**	11.86	12.88
	Avg. Rounds Lasted	9.09	8.52	**9.5**	11.41	**12.34**	10.9	13.06	**13.94**	12.83
Cubes Defeats	Avg. Cured	**2.1**	1.53	1.15	0.5	0.77	**1.66**	**1**	0.3	0.33
	Avg. Cubes Rem.	12.94	13.38	**14.03**	**14.25**	13.05	12.75	**13.62**	13.35	13.47
	Avg. Rounds Lasted	**10**	7.53	6.46	7.16	1.77	**12.5**	**12.2**	7.7	7.33
Cards Defeats	Avg. Cured	N/A	N/A	N/A	N/A	N/A	N/A	**3**	2	2.6
	Avg. Cubes Rem.	N/A	N/A	N/A	N/A	N/A	N/A	**14.87**	10	13.2
	Avg. Rounds Lasted	N/A	N/A	N/A	N/A	N/A	N/A	23	23	23

Figure 4.3: Homogeneous results using 6 Epidemic cards (50 experiments)



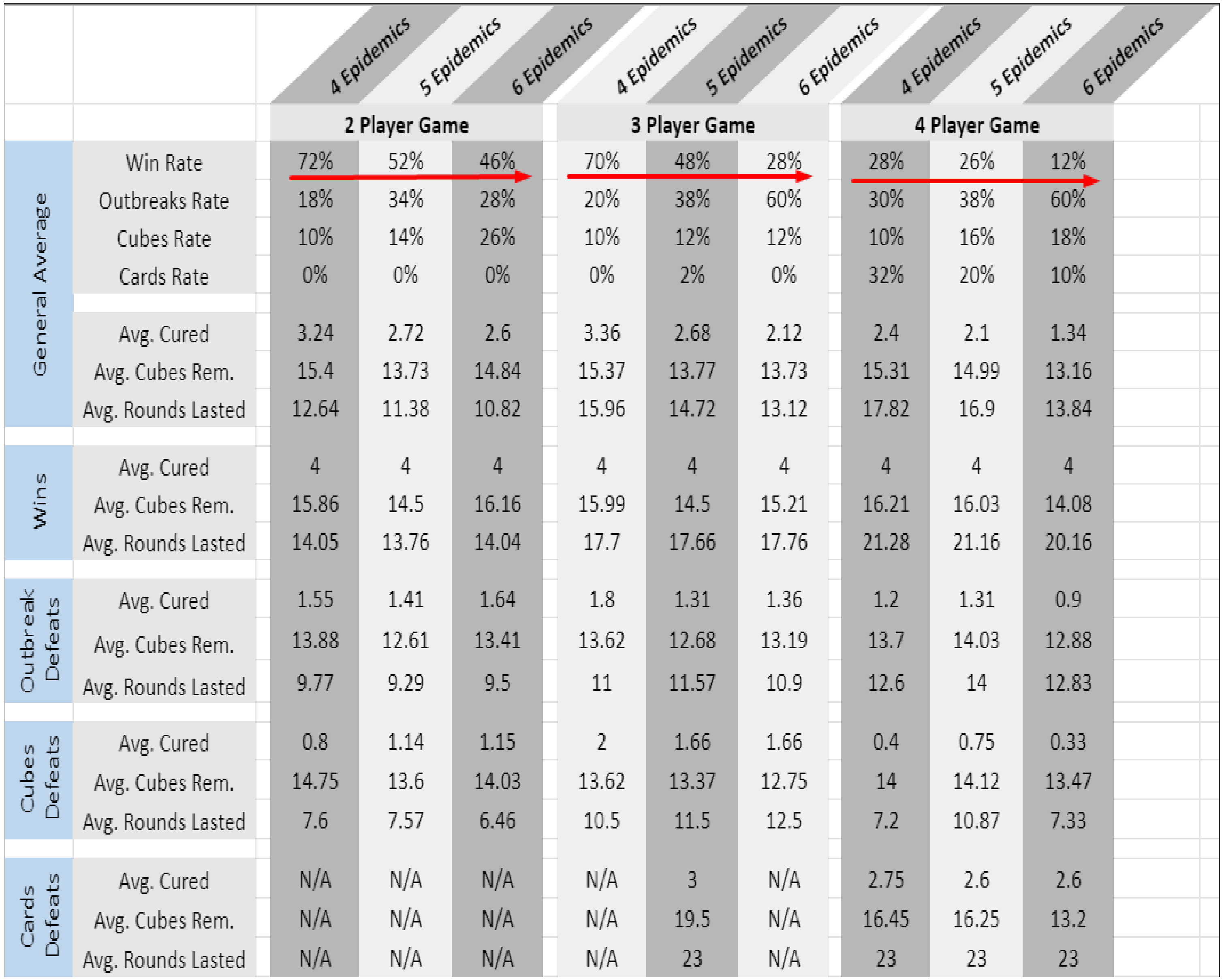
		2 Player Game			3 Player Game			4 Player Game		
		4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics
General Average	Win Rate	68%	44%	38%	58%	32%	16%	30%	18%	10%
	Outbreaks Rate	20%	32%	44%	30%	54%	72%	40%	62%	64%
	Cubes Rate	12%	24%	18%	12%	8%	12%	4%	20%	22%
	Cards Rate	0%	0%	0%	0%	6%	0%	26%	0%	4%
	Avg. Cured	2.96	2.56	2.72	3.16	2.56	1.70	2.40	1.60	1.38
	Avg. Cubes Rem.	15.87	14.76	14.27	15.53	14.89	13.98	15.52	13.75	13.96
	Avg. Rounds Lasted	11.26	11.00	10.74	15.94	15.18	12.00	17.30	14.32	13.96
Wins	Avg. Cured	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
	Avg. Cubes Rem.	16.82	15.50	16.09	16.30	16.20	15.37	16.89	15.65	15.25
	Avg. Rounds Lasted	13.14	14.00	13.00	17.46	18.25	18.25	20.15	20.50	18.80
Outbreak Defeats	Avg. Cured	0.90	1.40	1.86	2.13	1.74	1.38	1.05	1.24	1.00
	Avg. Cubes Rem.	13.57	13.93	13.23	14.61	14.24	13.63	14.31	12.90	13.82
	Avg. Rounds Lasted	8.30	9.20	9.09	13.86	12.78	11.41	13.05	13.58	13.06
Cubes Defeats	Avg. Cured	0.50	0.70	2.10	2.10	2.00	0.50	0.00	0.11	1.00
	Avg. Cubes Rem.	14.29	14.17	12.94	14.37	13.56	14.25	15.12	14.05	13.62
	Avg. Rounds Lasted	5.50	6.20	10.00	11.50	13.25	7.16	3.00	7.33	12.20
Cards Defeats	Avg. Cured	N/A	N/A	N/A	N/A	3.00	N/A	2.91	N/A	3.00
	Avg. Cubes Rem.	N/A	N/A	N/A	N/A	15.42	N/A	15.79	N/A	14.87
	Avg. Rounds Lasted	N/A	N/A	N/A	N/A	23.00	N/A	23.00	N/A	23.00

Figure 4.4: Homogeneous results using Agent Base-Tactic (50 experiments)



		2 Player Game			3 Player Game			4 Player Game		
		4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics
	Win Rate	58%	40%	32%	68%	30%	28%	32%	22%	10%
	Outbreaks Rate	30%	38%	42%	20%	54%	54%	30%	62%	68%
	Cubes Rate	12%	22%	26%	12%	16%	18%	10%	16%	20%
	Cards Rate	0%	0%	0%	0%	0%	0%	28%	0%	2%
General Average	Avg. Cured	2.98	2.60	2.30	3.10	2.04	2.06	2.62	1.86	1.20
	Avg. Cubes Rem.	14.54	13.70	13.92	14.71	13.84	12.78	13.67	12.76	12.44
	Avg. Rounds Lasted	12.02	10.64	9.76	15.28	12.38	13.04	18.80	15.60	13.60
Wins	Avg. Cured	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
	Avg. Cubes Rem.	16.43	15.58	16.35	15.70	14.65	14.64	15.51	15.09	15.15
	Avg. Rounds Lasted	14.06	13.10	13.18	17.67	17.33	17.00	21.66	20.25	21.20
Outbreak Defeats	Avg. Cured	1.53	1.79	1.47	1.30	1.18	1.42	1.66	1.30	1.02
	Avg. Cubes Rem.	11.96	12.01	12.40	12.05	13.31	11.76	12.40	11.83	11.86
	Avg. Rounds Lasted	9.20	9.42	8.52	10.60	10.14	12.34	14.86	13.97	13.94
Cubes Defeats	Avg. Cured	1.66	1.45	1.53	1.00	1.25	0.77	0.80	1.14	0.30
	Avg. Cubes Rem.	11.83	13.20	13.38	13.50	14.12	13.05	13.40	13.61	13.35
	Avg. Rounds Lasted	9.16	8.27	7.53	9.50	10.62	1.77	10.00	12.00	7.70
Cards Defeats	Avg. Cured	N/A	N/A	N/A	N/A	N/A	N/A	2.85	N/A	2.00
	Avg. Cubes Rem.	N/A	N/A	N/A	N/A	N/A	N/A	13.44	N/A	10.00
	Avg. Rounds Lasted	N/A	N/A	N/A	N/A	N/A	N/A	23.00	N/A	23.00

Figure 4.5: Homogeneous results using Agent Look-ahead (50 experiments)



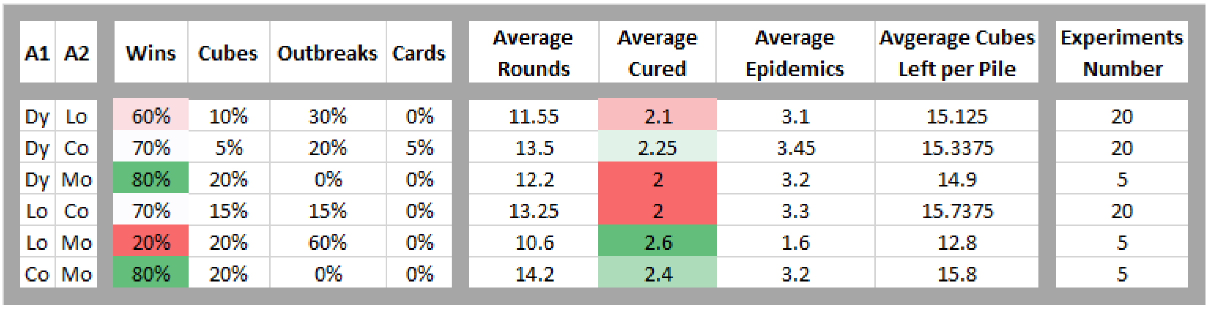
		2 Player Game			3 Player Game			4 Player Game		
		4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics	4 Epidemics	5 Epidemics	6 Epidemics
General Average	Win Rate	72%	52%	46%	70%	48%	28%	28%	26%	12%
	Outbreaks Rate	18%	34%	28%	20%	38%	60%	30%	38%	60%
	Cubes Rate	10%	14%	26%	10%	12%	12%	10%	16%	18%
	Cards Rate	0%	0%	0%	0%	2%	0%	32%	20%	10%
	Avg. Cured	3.24	2.72	2.6	3.36	2.68	2.12	2.4	2.1	1.34
	Avg. Cubes Rem.	15.4	13.73	14.84	15.37	13.77	13.73	15.31	14.99	13.16
	Avg. Rounds Lasted	12.64	11.38	10.82	15.96	14.72	13.12	17.82	16.9	13.84
Wins	Avg. Cured	4	4	4	4	4	4	4	4	4
	Avg. Cubes Rem.	15.86	14.5	16.16	15.99	14.5	15.21	16.21	16.03	14.08
	Avg. Rounds Lasted	14.05	13.76	14.04	17.7	17.66	17.76	21.28	21.16	20.16
Outbreak Defeats	Avg. Cured	1.55	1.41	1.64	1.8	1.31	1.36	1.2	1.31	0.9
	Avg. Cubes Rem.	13.88	12.61	13.41	13.62	12.68	13.19	13.7	14.03	12.88
	Avg. Rounds Lasted	9.77	9.29	9.5	11	11.57	10.9	12.6	14	12.83
Cubes Defeats	Avg. Cured	0.8	1.14	1.15	2	1.66	1.66	0.4	0.75	0.33
	Avg. Cubes Rem.	14.75	13.6	14.03	13.62	13.37	12.75	14	14.12	13.47
	Avg. Rounds Lasted	7.6	7.57	6.46	10.5	11.5	12.5	7.2	10.87	7.33
Cards Defeats	Avg. Cured	N/A	N/A	N/A	N/A	3	N/A	2.75	2.6	2.6
	Avg. Cubes Rem.	N/A	N/A	N/A	N/A	19.5	N/A	16.45	16.25	13.2
	Avg. Rounds Lasted	N/A	N/A	N/A	N/A	23	N/A	23	23	23

Figure 4.6: Homogeneous results using Agent Combined (50 experiments)

4.3 Mixed Teams - Heterogeneous Results

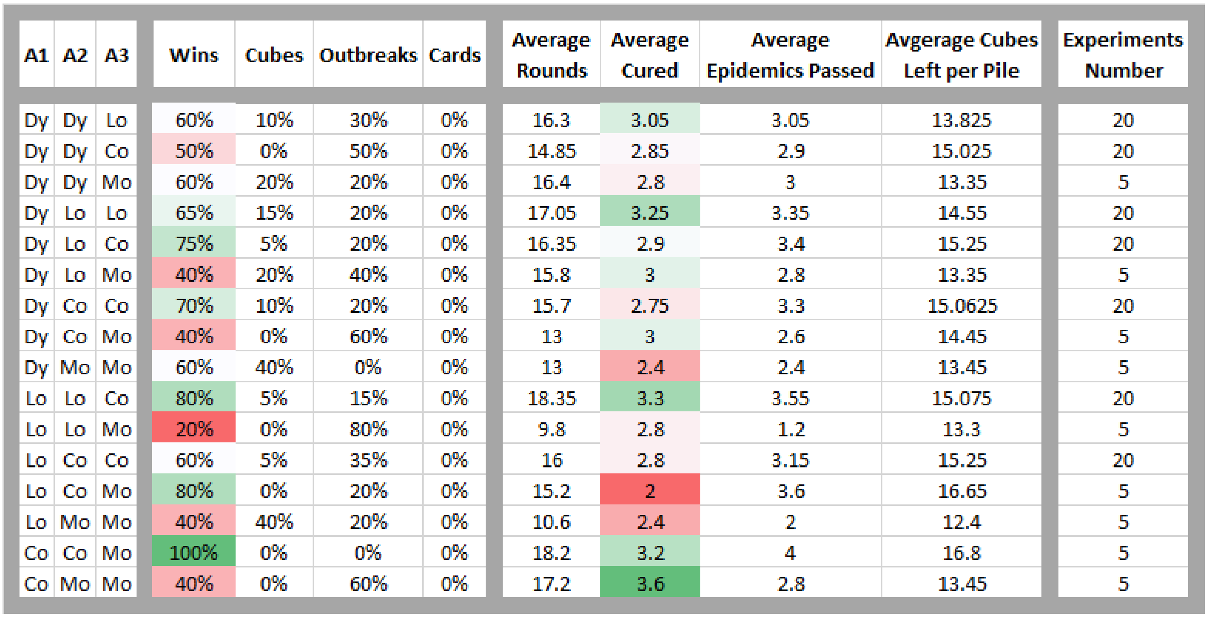
While analyzed the homogeneous results, now we should take a look at the heterogeneous results. Teams consisting always with the same kind of players (aka. having the same logic-tactic) are not that common. For that reason we used every possible combination of agents to form heterogeneous teams and conduct experiments.

Following on we can see the heterogeneous results of the experiments using 2 & 3 players for our game.



A1	A2	Wins	Cubes	Outbreaks	Cards	Average Rounds	Average Cured	Average Epidemics	Avgerage Cubes Left per Pile	Experiments Number
Dy	Lo	60%	10%	30%	0%	11.55	2.1	3.1	15.125	20
Dy	Co	70%	5%	20%	5%	13.5	2.25	3.45	15.3375	20
Dy	Mo	80%	20%	0%	0%	12.2	2	3.2	14.9	5
Lo	Co	70%	15%	15%	0%	13.25	2	3.3	15.7375	20
Lo	Mo	20%	20%	60%	0%	10.6	2.6	1.6	12.8	5
Co	Mo	80%	20%	0%	0%	14.2	2.4	3.2	15.8	5

Figure 4.7: Heterogeneous results using 2 Players

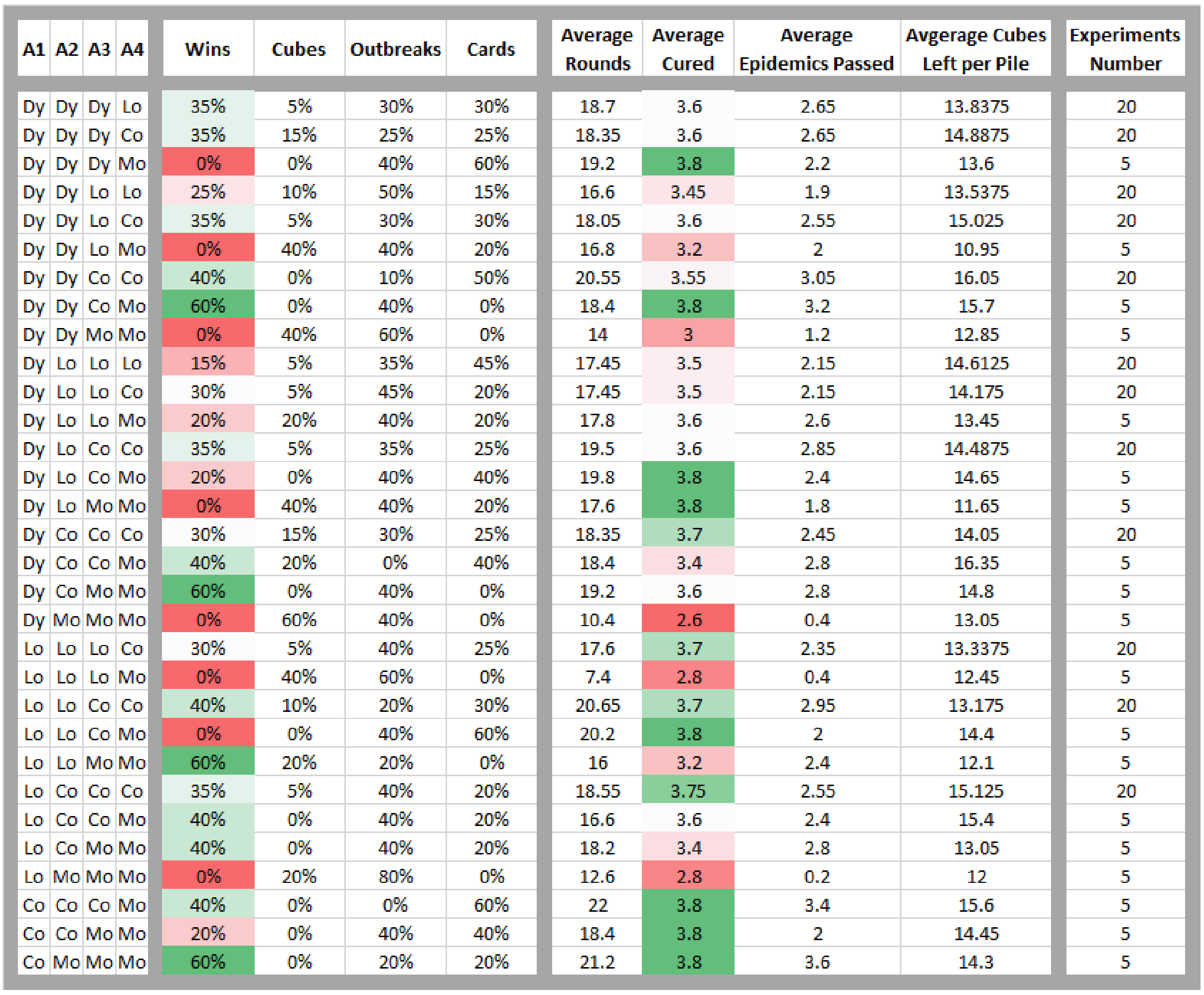


A1	A2	A3	Wins	Cubes	Outbreaks	Cards	Average Rounds	Average Cured	Average Epidemics Passed	Avgerage Cubes Left per Pile	Experiments Number
Dy	Dy	Lo	60%	10%	30%	0%	16.3	3.05	3.05	13.825	20
Dy	Dy	Co	50%	0%	50%	0%	14.85	2.85	2.9	15.025	20
Dy	Dy	Mo	60%	20%	20%	0%	16.4	2.8	3	13.35	5
Dy	Lo	Lo	65%	15%	20%	0%	17.05	3.25	3.35	14.55	20
Dy	Lo	Co	75%	5%	20%	0%	16.35	2.9	3.4	15.25	20
Dy	Lo	Mo	40%	20%	40%	0%	15.8	3	2.8	13.35	5
Dy	Co	Co	70%	10%	20%	0%	15.7	2.75	3.3	15.0625	20
Dy	Co	Mo	40%	0%	60%	0%	13	3	2.6	14.45	5
Dy	Mo	Mo	60%	40%	0%	0%	13	2.4	2.4	13.45	5
Lo	Lo	Co	80%	5%	15%	0%	18.35	3.3	3.55	15.075	20
Lo	Lo	Mo	20%	0%	80%	0%	9.8	2.8	1.2	13.3	5
Lo	Co	Co	60%	5%	35%	0%	16	2.8	3.15	15.25	20
Lo	Co	Mo	80%	0%	20%	0%	15.2	2	3.6	16.65	5
Lo	Mo	Mo	40%	40%	20%	0%	10.6	2.4	2	12.4	5
Co	Co	Mo	100%	0%	0%	0%	18.2	3.2	4	16.8	5
Co	Mo	Mo	40%	0%	60%	0%	17.2	3.6	2.8	13.45	5

Figure 4.8: Heterogeneous results using 3 Players

Based on the figures above, we can clearly see that our best performing agent, the Agent_Compared, is the one that is present in every good wining percentage result. This was to be expected, due to the fact that this agent is the most complicated, that combines different approaches to end up with results.

We also deducted experiments for 4 players game, and the results are shown below.



A1	A2	A3	A4	Wins	Cubes	Outbreaks	Cards	Average Rounds	Average Cured	Average Epidemics Passed	Avgerage Cubes Left per Pile	Experiments Number
Dy	Dy	Dy	Lo	35%	5%	30%	30%	18.7	3.6	2.65	13.8375	20
Dy	Dy	Dy	Co	35%	15%	25%	25%	18.35	3.6	2.65	14.8875	20
Dy	Dy	Dy	Mo	0%	0%	40%	60%	19.2	3.8	2.2	13.6	5
Dy	Dy	Lo	Lo	25%	10%	50%	15%	16.6	3.45	1.9	13.5375	20
Dy	Dy	Lo	Co	35%	5%	30%	30%	18.05	3.6	2.55	15.025	20
Dy	Dy	Lo	Mo	0%	40%	40%	20%	16.8	3.2	2	10.95	5
Dy	Dy	Co	Co	40%	0%	10%	50%	20.55	3.55	3.05	16.05	20
Dy	Dy	Co	Mo	60%	0%	40%	0%	18.4	3.8	3.2	15.7	5
Dy	Dy	Mo	Mo	0%	40%	60%	0%	14	3	1.2	12.85	5
Dy	Lo	Lo	Lo	15%	5%	35%	45%	17.45	3.5	2.15	14.6125	20
Dy	Lo	Lo	Co	30%	5%	45%	20%	17.45	3.5	2.15	14.175	20
Dy	Lo	Lo	Mo	20%	20%	40%	20%	17.8	3.6	2.6	13.45	5
Dy	Lo	Co	Co	35%	5%	35%	25%	19.5	3.6	2.85	14.4875	20
Dy	Lo	Co	Mo	20%	0%	40%	40%	19.8	3.8	2.4	14.65	5
Dy	Lo	Mo	Mo	0%	40%	40%	20%	17.6	3.8	1.8	11.65	5
Dy	Co	Co	Co	30%	15%	30%	25%	18.35	3.7	2.45	14.05	20
Dy	Co	Co	Mo	40%	20%	0%	40%	18.4	3.4	2.8	16.35	5
Dy	Co	Mo	Mo	60%	0%	40%	0%	19.2	3.6	2.8	14.8	5
Dy	Mo	Mo	Mo	0%	60%	40%	0%	10.4	2.6	0.4	13.05	5
Lo	Lo	Lo	Co	30%	5%	40%	25%	17.6	3.7	2.35	13.3375	20
Lo	Lo	Lo	Mo	0%	40%	60%	0%	7.4	2.8	0.4	12.45	5
Lo	Lo	Co	Co	40%	10%	20%	30%	20.65	3.7	2.95	13.175	20
Lo	Lo	Co	Mo	0%	0%	40%	60%	20.2	3.8	2	14.4	5
Lo	Lo	Mo	Mo	60%	20%	20%	0%	16	3.2	2.4	12.1	5
Lo	Co	Co	Co	35%	5%	40%	20%	18.55	3.75	2.55	15.125	20
Lo	Co	Co	Mo	40%	0%	40%	20%	16.6	3.6	2.4	15.4	5
Lo	Co	Mo	Mo	40%	0%	40%	20%	18.2	3.4	2.8	13.05	5
Lo	Mo	Mo	Mo	0%	20%	80%	0%	12.6	2.8	0.2	12	5
Co	Co	Co	Mo	40%	0%	0%	60%	22	3.8	3.4	15.6	5
Co	Co	Mo	Mo	20%	0%	40%	40%	18.4	3.8	2	14.45	5
Co	Mo	Mo	Mo	60%	0%	20%	20%	21.2	3.8	3.6	14.3	5

Figure 4.9: Heterogeneous results using 4 Players

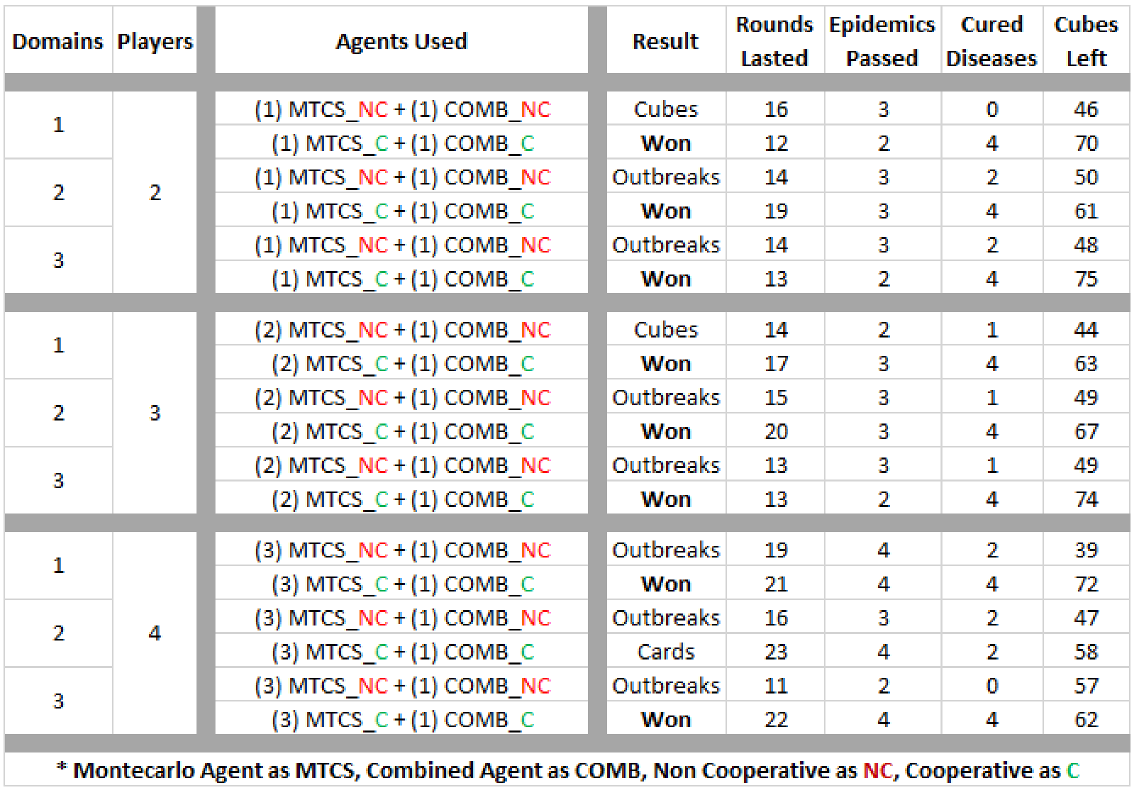
Looking at the MCTS agent, the results are not that great, but the average cure rate is really high. Even the low efficiency that we currently have, this agent has great potential, and we hope for futures improvements in order to create an even more powerful agent for this game.

4.4 To Cooperate or Not To Cooperate?

As we said previously, one of the characteristic of this board game that making is so special as a domain, is the fact that players can cooperate between them in order to be more effective. Even though that trading cards function is not available in this version of the game, still the functionality of suggestions may (or may not) have a great impact on how our agents perform.

By being cooperative, the agents may be able to explore the domain using a different perspective. Different tactics produce different results, so when an agent is capable of accepting suggestions, he/she can "fuse" the current tactic with the tactic that the player who made the suggestion was using.

Following on, we can see some experiments (Figure 4.10) that were made in exact same domains (same amount of players, same decks, same roles), using Montecarlo Agents as our main players and an Agent Combined that was sending recommendations.



Domains	Players	Agents Used	Result	Rounds Lasted	Epidemics Passed	Cured Diseases	Cubes Left
1	2	(1) MTCS_NC + (1) COMB_NC	Cubes	16	3	0	46
		(1) MTCS_C + (1) COMB_C	**Won**	12	2	4	70
2		(1) MTCS_NC + (1) COMB_NC	Outbreaks	14	3	2	50
		(1) MTCS_C + (1) COMB_C	**Won**	19	3	4	61
3		(1) MTCS_NC + (1) COMB_NC	Outbreaks	14	3	2	48
		(1) MTCS_C + (1) COMB_C	**Won**	13	2	4	75
1	3	(2) MTCS_NC + (1) COMB_NC	Cubes	14	2	1	44
		(2) MTCS_C + (1) COMB_C	**Won**	17	3	4	63
2		(2) MTCS_NC + (1) COMB_NC	Outbreaks	15	3	1	49
		(2) MTCS_C + (1) COMB_C	**Won**	20	3	4	67
3		(2) MTCS_NC + (1) COMB_NC	Outbreaks	13	3	1	49
		(2) MTCS_C + (1) COMB_C	**Won**	13	2	4	74
1	4	(3) MTCS_NC + (1) COMB_NC	Outbreaks	19	4	2	39
		(3) MTCS_C + (1) COMB_C	**Won**	21	4	4	72
2		(3) MTCS_NC + (1) COMB_NC	Outbreaks	16	3	2	47
		(3) MTCS_C + (1) COMB_C	Cards	23	4	2	58
3		(3) MTCS_NC + (1) COMB_NC	Outbreaks	11	2	0	57
		(3) MTCS_C + (1) COMB_C	**Won**	22	4	4	62

*** Montecarlo Agent as MTCS, Combined Agent as COMB, Non Cooperative as NC, Cooperative as C**

Figure 4.10: Results with & without cooperation

Looking at the results we can see how important cooperation is in this game. The Montecarlo Agent was unable to produce good results, due to the fact that the branching factor of the game is huge. The combination of an agent that is not able to achieve good results with an agent that is capable of achieving great results in not of any importance until the player recommendation feature is enabled.

From the total of 18 games without cooperating, the agents lost 9 of them (**0% win ratio**), but when the recommendation feature was enabled, the same team manages to win 8 out of 9 games (**88% win ratio**).

Really interesting games are two of the ones that took place when using 2 players. In the 1st and 3rd pair of game instantiations, we can see that the agents needed less rounds to win a game that the same team previously lost. The interesting fact about these games, is that even the domains were identical (even the cards were set us to form the exact same deck) the players lost, when they had the "chance" (aka enough cards) to win the game. That indicates that when the agents are cooperating they might end performing a move that they would not do otherwise, because they do have knowledge about the other player's intentions.

A worth noticing game is also the one that took place in the second pair of games instantiations among the games using 4 players, using 4 players. In this game we can see that without cooperation the agents lost due to outbreaks. The same agents, when cooperating were also unable to win the game, but they lost due to the lack of cards, which means that even though were unable to gather the sets of cards needed, they did manage to keep the diseases under control.

Chapter 5

Conclusion

5.1 Conclusions and Future Work

In this thesis we developed a server, a related communication infrastructure, and several clients (different agents employing various strategies) for the Pandemic cooperative board game.

The game was implemented using Java, and it the features of it are the same with the original "Pandemic" board game, excluding the Events Cards, the Share Knowledge Ability that were also implemented, but never tested. Small adjustments were made to equalize the lack of the Share Knowledge ability, such as the game needed one less card in order to cure a disease.

All of the agents that were implemented, were also built using Java. Seven (7) agents were built in total, each one with different characteristic. Most noticeable agents are the Dummy Agent (serving the purpose of introductory code for future builds), the Agent Combined (that using a heuristic function and player profiling achieves great results), and an early build of an agent that implements MCTS (that is also an introductory model for future improvement).

These agents were tested throughout different environments, and based on the results that are presented in this thesis the following conclusions were made:

- "Pandemic" is a great board game in order to improve the abilities of autonomous agents. A complex game, a big branching factor, and the need for cooperation are the most important characteristics of this domain.
- Using a heuristic function is a great way to achieve good results, but miscalculations are unavoidable so fine tuning the heuristic function is a must.
- The algorithm of MCTS is an amazing way to explore state space and achieve good results without any real knowledge of the domain, but when the branching factor is high the MCTS needs adjustments, like tree pruning, in order to be a viable solution.
- This infrastructure and the agents were used in the course COMP512 of the Spring semester of 2020, and we would like it to be used as a stepping stone for the creation of agents (even outside of the School of Computer & Electrical Engineering of Technical University of Crete) competing in international completions.

5.2 Future Work

The world of AI and autonomous agent, is making progress in an amazing pace, but still domains like co operational games are not fully explored. Due to that fact we do propose the following ideas as future work based on this thesis:

First, a visualization of the game that was implemented and / or adding missing features of the original "Pandemic" game (eg. extra roles, event cards). Second, we believe that an improvement of the existing Montecarlo Agent has to be implemented, using smart pruning methods in order to "restrict" the size of the state space. Finally, we prospect the creation of different heuristic approaches using our dummy agent as a base structure.

The source code of this thesis will be open source for everyone to be able to check, inspect and improve it. The code that will be available will be the source code of the server as well as the test agent's code that will be used as a base for building new agents. Code can be found here[1].

[1] https://github.com/kvoloudakis-git/Pandemic.git

References

[1] Winands, M.H.M. In: Monte-Carlo Tree Search in Board Games. Springer Singapore, Singapore (2017) 47–76 xi, 21

[2] Leacock, M.: Pandemic from z-man games®[online] https://www.zmangames.com/en/products/pandemic/. 1, 3

[3] Kolchinsky, A., Brendan, T.: Fundamentals of machine learning (04 2019) [online https://www.complexityexplorer.org/courses/81-fundamentals-of-machine-learning/segments/7407?summary. 12

[4] Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and agents: Designing intelligent gameplay. International Journal of Computer Games Technology **2009** (Mar 2009) 837095 16

[5] Patel, R., Pathak, M.: Comparative analysis of search algorithms. (06 2018) [online] https://www.researchgate.net/publication/333262471_Comparative_Analysis_of_Search_Algorithms. 17

[6] Poole, D.L., Mackworth, A.K.: Artificial Intelligence: Foundations of Computational Agents. 2nd edn. Cambridge University Press, USA (2017) 19

[7] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 3rd edn. Prentice Hall Press, USA (2009) 19

[8] Korf, R.E. In: Artificial Intelligence Search Algorithms. 2 edn. Chapman and Hall (2010) 22 19

[9] Theodoridis, A., Chalkiadakis, G.: Monte carlo tree search for the game of diplomacy. In: 11th Hellenic Conference on Artificial Intelligence. SETN 2020, New York, NY, USA, Association for Computing Machinery (2020) 1625 21

[10] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2) (May 2002) 235–256 23

[11] Panousis Konstantinos: Real-time planning and learning in the "settlers of catan" strategy game. (2014) 23

[12] Karamalegkos Emmanouil: Monte carlo tree search in the "settlers of catan" strategy game. (Technical University of Crete, 2016) 23

[13] Theodoridis Alexios: Monte carlo tree search for the diplomacy multi-agent strategic game. (Technical University of Crete, 2020) 24

[14] Romanycia, M.H.J., Pelletier, F.J.: What is a heuristic? Computational Intelligence **1**(1) (1985) 47–58 24

[15] Daylamani-Zad, D., Agius, H., Angelides, M.C.: Reflective agents for personalisation in collaborative games. Artificial Intelligence Review **53**(1) (Jan 2020) 429–474 25

[16] Ganzfried, S., Sandholm, T.: Game theory-based opponent modeling in large imperfect-information games. Volume 1. (01 2011) 533–540 25

[17] Margolis, E., Samuels, R., Stich, S.P., eds.: The Oxford Handbook of Philosophy of Cognitive Science. Oxford University Press (May 2012) 25

[18] von der Osten, F.B., Kirley, M., Miller, T.: The minds of many: Opponent modeling in a stochastic game. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. (2017) 3845–3851 25

[19] Chacon, P.S., Eger, M.: Pandemic as a challenge for human-ai cooperation. In: Sixteenth Artificial Intelligence and Interactive Digital Entertainment Conference. (2020) 26

[20] Sauma-Chac³n, P., Eger, M.: Paindemic: A planning agent for pandemic. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment **16**(1) (Oct. 2020) 287–293 26

[21] Gaina, R.D., Balla, M., Dockhorn, A., Montoliu, R., Perez-Liebana, D.: Design and implementation of tag: A tabletop games framework (2020) 26

[22] Keebler, J., DiazGranados, D., Smith, D.: Learning team theories and measurement through the game pandemic. Proceedings of the Human Factors and Ergonomics Society Annual Meeting **58** (10 2014) 442–446 27

[23] Sfikas, K., Liapis, A.: Collaborative agent gameplay in the pandemic board game. In: International Conference on the Foundations of Digital Games. FDG '20, New York, NY, USA, Association for Computing Machinery (2020) 27

[24] Allis, L.V.: Searching for solutions in games and artificial intelligence (1994) 30