



Technical University of Crete
School of Electrical and Computer Engineering

Monte Carlo Tree Search for the “Diplomacy” Multi-agent Strategic Game

Diploma Thesis: Alexios Theodoridis

Committee:

Advisor: Georgios Chalkiadakis, Associate Professor

Committee Member: Antonios Deligiannakis, Associate Professor

Committee Member: Michail G. Lagoudakis, Associate Professor

Abstract

Monte Carlo Tree Search (MCTS) is a collective name for a family of methods seeking to identify optimal decisions in a given domain, while making use of the results of simulated outcomes in the search space. MCTS has received considerable interest in the past decade due to its demonstrated effectiveness in a number of domains, including its much-celebrated success in the game of Go.

In this thesis, we explore the application of MCTS in the “Diplomacy” multi-agent strategic board game. Diplomacy is a game of high complexity; though it is a game with full observability, the players’ actions are simultaneous. This renders any attempted MCTS implementation challenging, because each action (i.e., a player’s “move order” for this domain) has to be combined with the unknown beforehand, simultaneous actions of six other opponents, thus resulting in a non-predictable outcome. To the best of our knowledge, this is the first work to employ MCTS in the game of Diplomacy.

In our work we developed and experimented with several variants of the MCTS algorithm, differentiating in each one of them the amount and quality of domain knowledge provided to the main algorithm. We attempted to keep this domain knowledge to a minimum, while at the same time making the approach worthwhile in terms of time required for effective decision-making. In the core of our MCTS approach lies the well-known “bandit method” Upper Confidence Bounds for Trees (UCT), which attempts to strike a balance between exploration and exploitation during the search tree creation.

We provide a thorough experimental evaluation of our approach, in which we systematically compare the performance of our agents against each other and against other known agents, including the to date most successful Diplomacy agent, “D-Brane”. Our results show that several of our agents are quite competitive in this domain, exhibiting as they do performance which is comparable and, in some instances, superior to that of D-Brane. Interestingly, the MCTS agents consistently beat D-Brane in tournaments in which one MCTS agent faces one D-Brane agent and several other opponents. Our experiments also demonstrate that carefully injecting high quality domain knowledge into the MCTS algorithm, improves its performance significantly. Finally, our thesis resulted in the creation of a basic computational framework that allows further research on using MCTS for the game of Diplomacy.

Δενδρική Αναζήτηση Monte Carlo στο Πολυπρακτορικό Στρατηγικό Παιχνίδι “Diplomacy”

Περίληψη

Ο γενικός όρος Δενδρική Αναζήτηση Monte Carlo (Monte Carlo Tree Search - MCTS) περιγράφει μια κατηγορία αλγορίθμων εύρεσης βέλτιστων αποφάσεων σε ένα δοσμένο περιβάλλον, χρησιμοποιώντας τα αποτελέσματα προσομοιωμένων εκβάσεων στο χώρο αναζήτησης. Τα τελευταία χρόνια η Δενδρική Αναζήτηση Monte Carlo είναι ιδιαίτερα δημοφιλής λόγω της αποδεδειγμένης αποτελεσματικότητάς της σε μια σειρά από περιβάλλοντα, συμπεριλαμβανομένου του εξαιρετικά δύσκολου για τον άνθρωπο παιχνιδιού Go.

Στην παρούσα διπλωματική εργασία, ερευνούμε την εφαρμογή αλγορίθμων MCTS στο πολυ-πρακτορικό στρατηγικό παιχνίδι «Diplomacy». Το Diplomacy είναι ένα παιχνίδι υψηλής πολυπλοκότητας. Αν και είναι ένα πλήρως παρατηρήσιμο παιχνίδι (όχι μερικής παρατηρησιμότητας), απαιτεί ταυτόχρονες ενέργειες εκ μέρους των επτά συμμετεχόντων αντιπάλων. Αυτό αυξάνει τη δυσκολία της εφαρμογής της MCTS προσέγγισης, επειδή κάθε ενέργεια (εντολή κίνησης σε αυτό το περιβάλλον) σε συνδυασμό με τις ενέργειες των υπόλοιπων (έξι στον αριθμό) αντιπάλων, καταλήγει σε μη προβλέψιμο αποτέλεσμα.

Στη δουλειά μας προτείνουμε και πειραματιστήκαμε με πολλαπλές εκδοχές του αλγόριθμου MCTS, διαφοροποιώντας την ποσότητα και την ποιότητα της σχετικής με το περιβάλλον γνώσης που χρησιμοποιούμε σε κάθε μία από αυτές. Προσπαθήσαμε να μην παρέχουμε υπερβολική γνώση του περιβάλλοντος στον πράκτορα ώστε να κρατήσουμε την υλοποίησή του όσο πιο γενική και εγγύτερη στην κλασική MCTS προσέγγιση γίνονταν. Ταυτόχρονα, προσπαθήσαμε να κτίσουμε έναν αποτελεσματικό αλγόριθμο που να βελτιστοποιεί τις αποφάσεις του σε συνάρτηση με το χρόνο που έχει στη διάθεσή του για τη λήψη τους. Στο κέντρο της MCTS προσέγγισής μας βρίσκεται μια γνωστή μέθοδος «κουλοχέρη», η λεγόμενη μέθοδος χρήσης “Ανω Ορίου Εμπιστοσύνης για Δέντρα” (Upper Confidence Bounds for Trees - UCT), η οποία προσπαθεί να ισορροπήσει μεταξύ εξερεύνησης και εκμετάλλευσης πληροφορίας κατά τη δημιουργία του δέντρου αναζήτησης.

Η εργασία μας παρέχει μια εκτενή και προσεκτική πειραματική αξιολόγηση της προσέγγισής μας. Συγκεκριμένα, παρέχουμε μια συστηματική αξιολόγηση της απόδοσης των αλγορίθμων μας, συγκρίνοντάς τους μεταξύ τους καθώς και με άλλους αντίπαλους πράκτορες γνωστούς από τη βιβλιογραφία. Στους τελευταίους περιλαμβάνεται και ο “D-Brane”, ο πλέον επιτυχημένος ως τώρα πράκτορας στο παιχνίδι Diplomacy. Τα αποτελέσματα δείχνουν ότι αρκετοί από τους πράκτορές μας αποδεικνύονται εξαιρετικά ανταγωνιστικοί σε αυτό το απαιτητικό παιχνίδι, με απόδοση που είναι συγκρίσιμη και ορισμένες φορές καλύτερη από αυτήν του D-Brane. Είναι σημαντικό το ότι οι πράκτορές μας κερδίζουν συστηματικά τον D-Brane σε τουρνουά στα οποία συμμετέχει ένας MCTS πράκτορας, ένας D-Brane πράκτορας, και πέντε άλλοι, διαφορετικοί, αντίπαλοι. Τα πειράματά μας επίσης αποδεικνύουν ότι η προσεκτική εισαγωγή καλής ποιότητας γνώσης πεδίου στον MCTS αλγόριθμο, βελτιώνει την απόδοσή του σημαντικά. Σε αυτή τη δουλειά αναπτύσσουμε διάφορες εκδοχές της προσέγγισής μας και μια συστηματική τους αξιολόγηση. Τέλος, η διπλωματική μας εργασία προσέφερε τη δημιουργία ενός βασικού υπολογιστικού πλαισίου που επιτρέπει την περαιτέρω έρευνα σχετικά με την χρήση MCTS τεχνικών στο παιχνίδι Diplomacy.

Contents

Table of Figures	6
1. Introduction	8
2. Background	10
2.1. Monte Carlo Tree Search	10
2.2. The Upper Confidence Bounds for Trees (UCT) algorithm	12
2.3. Diplomacy strategic game.....	14
2.4. The ANAC Diplomacy Challenge	18
2.5. BANDANA framework	19
2.6. Related work	21
2.6.1. Monte Carlo Tree Search in strategic games	21
2.6.2. Diplomacy agents.....	21
3. Our Approach.....	24
3.1. MCTS_Agent Version 1	29
3.2. MCTS_Agent Version 2	31
3.3. MCTS_Agent Version 3	31
3.4. MCTS_Agent Version 4	32
3.5. MCTS_Agent Version 5	32
3.6. MCTS_Agent Version 6	35
3.7. MCTS_Agent Version 7	35
3.8. MCTS_Agent Version 8	35
4. Experimental Evaluation	36
4.1. Experimental Settings and Detailed Results	36
4.1.1. MCTS_Agents against Random Bots	37
4.1.2. MCTS_Agents against Dumb Bots.....	43
4.1.3. MCTS_Agents against D-Brane	48
4.2. Comparative Results	64
4.3. Increasing the year limit and the decision-making time.....	68
4.4. Discussion of Results.....	69
5. Conclusions and Future Work.....	71
6. Appendix A: Monte Carlo Tree Search variants	72
7. Appendix B: Extra Experiments.....	74
References	75

Table of Figures

Figure 1. One iteration of the general MCTS approach. (Browne, et al. 2012)	11
Figure 2. Diplomacy Map. The dots indicate that this Province is a Supply Center and the colors that it is owned by the corresponding Great Power.....	14
Figure 3: Examples of move orders resolution of an Army from Paris and a Fleet from English Channel. (Calhamer 2000).....	16
Figure 4: Examples of two move orders causing a standoff. At the first picture, both Armies try to move to Silesia, and at the second picture an Army from Prussia tries to move to Berlin and a Fleet from Berlin tries to move to Prussia. (Calhamer 2000)	16
Figure 5: The support from the Army in Silesia is cut by an attack from Bohemia. So, the move order of the Army from Prussia to Warsaw and the hold order of the Army at Warsaw cause a standoff. (Calhamer 2000).....	17
Figure 6: Running a 30-game tournament with BANDANA framework	19
Figure 7: Example of the reward values updated after four iterations. X_{ja}' represents the first update of X_j at node a, X_{ja}'' represents the second update of X_j at node a etc.....	25
Figure 8: Flowchart of the basic algorithm	28
Figure 9: Use of an alternative simulation policy in MCTS_Agent Version 2.....	31
Figure 10: Monte Carlo Tree Search with two expansion steps	35
Figure 11: D-Brane vs RandomBots. Average over 30 games.....	37
Figure 12: MCTS_Agent V1 vs RandomBots. Average over 30 games.....	38
Figure 13: MCTS_Agent V2 vs RandomBots. Average over 30 games.....	38
Figure 14: MCTS_Agent V3 vs RandomBots. Average over 30 games.....	39
Figure 15: MCTS_Agent V4 vs RandomBots. Average over 30 games.....	39
Figure 16: MCTS_Agent V5 vs RandomBots. Average over 30 games.....	40
Figure 17: MCTS_Agent V6 vs RandomBots. Average over 30 games.....	40
Figure 18: MCTS_Agent V7 vs RandomBots. Average over 30 games.....	41
Figure 19: MCTS_Agent V8 vs RandomBots. Average over 30 games.....	41
Figure 20: versions of the MCTS_Agent and D-Brane vs RandomBot. Performance according to the $SC_{agent} - SC_{RandomBot}$ metric.....	42
Figure 21: D-Brane vs DumbBots. Average over 30 games.	43
Figure 22: MCTS_Agent V1 vs DumbBots. Average over 30 games.....	43
Figure 23: MCTS_Agent V2 vs DumbBots. Average over 30 games.....	44
Figure 24: MCTS_Agent V3 vs DumbBots. Average over 30 games.....	44
Figure 25: MCTS_Agent V4 vs DumbBots. Average over 30 games.....	45
Figure 26: MCTS_Agent V5 vs DumbBots. Average over 30 games.....	45
Figure 27: MCTS_Agent V6 vs DumbBots. Average over 30 games.....	46
Figure 28: MCTS_Agent V7 vs DumbBots. Average over 30 games.....	46
Figure 29: MCTS_Agent V8 vs DumbBots. Average over 30 games.....	47
Figure 30: versions of the MCTS_Agent and D-Brane vs DumbBot. Performance according to the $SC_{agent} - SC_{DumbBot}$ metric.....	47
Figure 31: MCTS_Agent V1 vs 2 D-Brane and 4 RandomBots. Average over 30 games.....	48

Figure 32: MCTS_Agent V2 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.	49
Figure 33: MCTS_Agent V2 vs 2 D-Brane and 4 RandomBots. Average over 30 games.	49
Figure 34: MCTS_Agent V3 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.	50
Figure 35: MCTS_Agent V3 vs 2 D-Brane and 4 RandomBots. Average over 30 games.	50
Figure 36: MCTS_Agent V4 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.	51
Figure 37: MCTS_Agent V4 vs 2 D-Brane and 4 RandomBots. Average over 30 games.	51
Figure 38: MCTS_Agent V5 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.	52
Figure 39: MCTS_Agent V5 vs 3 D-Brane and 3 RandomBots. Average over 30 games.	52
Figure 40: MCTS_Agent V5 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	53
Figure 41: MCTS_Agent V5 vs 5 D-Brane and 1 DumbBot. Average over 30 games.	53
Figure 42: 3 MCTS_Agent V5 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	54
Figure 43: 5 MCTS_Agent V5 vs 1 D-Brane and 1 DumbBot. Average over 30 games.	54
Figure 44: MCTS_Agent V6 vs 1 D-Brane 2 DumbBots and 3 RandomBots. Average over 30 games.	55
Figure 45: MCTS_Agent V6 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	55
Figure 46: MCTS_Agent V6 vs 3 D-Brane and 3 RandomBots. Average over 30 games.	56
Figure 47: MCTS_Agent V6 vs 5 D-Brane and 1 DumbBot. Average over 30 games.	56
Figure 48: 3 MCTS_Agent V6 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	57
Figure 49: 5 MCTS_Agent V6 vs 1 D-Brane and 1 DumbBot. Average over 30 games.	57
Figure 50: MCTS_Agent V7 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.	58
Figure 51: MCTS_Agent V7 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	58
Figure 52: MCTS_Agent V7 vs 3 D-Brane. Average over 30 games.	59
Figure 53: MCTS_Agent V7 vs 5 D-Brane and 1 DumbBot. Average over 30 games.	59
Figure 54: 3 MCTS_Agent V7 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	60
Figure 55: 5 MCTS_Agent V7 vs 1 D-Brane and 1 DumbBot. Average over 30 games.	60
Figure 56: MCTS_Agent V8 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.	61
Figure 57: MCTS_Agent V8 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	61
Figure 58: MCTS_Agent V8 vs 5 D-Brane and 1 DumbBot. Average over 30 games.	62
Figure 59: 3 MCTS_Agent V8 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.	62
Figure 60: 5 MCTS_Agent V8 vs 1 D-Brane and 1 DumbBot. Average over 30 games.	63
Figure 61: One MCTS_Agent vs one, three or five D-Brane agents. Subtraction of the average SCs owned by the agents at every tournament. Average over 30 games.	64
Figure 62: One MCTS_Agent vs one, three or five D-Brane agents. Subtraction of the average rankings achieved by the agents at every tournament. Average over 30 games.	65
Figure 63: One D-Brane agent vs one, three or five MCTS_Agents. Subtraction of the average SCs owned by the agents at every tournament. Average over 30 games.	66
Figure 64: One D-Brane agent vs one, three or five MCTS_Agents. Subtraction of the average rankings achieved by the agents at every tournament. Average over 30 games.	67
Figure 65: MCTS_AgentV8 vs D-Brane. Greater year limit (10 years). Average of 20 games.	68
Figure 66: MCTS_AgentV8 vs D-Brane. Greater year limit (10 years). Greater desicion-making time limit (12 seconds). Average of 20 games.	68
Figure 67: MCTS_AgentV8 vs 5 D-Branes and 1 RandomBot. Greater year limit (10 years). Average of 20 games.	74
Figure 68: MCTS_AgentV8 vs 5 D-Branes and 1 RandomBot. Greater year limit (10 years). Inreased time for MCTS desicion-making (from 8 to 12 seconds). Average of 20 games.	74

1. Introduction

Monte Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain using the results of simulated outcomes in the search space. It has received considerable attention since its important success at the game of computer Go (Silver, et al. 2017) and at the same time it has been effective in a variety of other domains (Browne, et al. 2012). What makes MCTS really effective is its ability to combine the accuracy of a tree search with the generality of random playouts. It is a promising method for Artificial Intelligence (AI) approaches, especially for domains that can be represented as trees of sequential states.

“MCTS rests on two fundamental concepts: that the true value of an action may be approximated using random simulation; and that these values may be used efficiently to adjust the policy towards a best-first strategy.” (Browne, et al. 2012) The family of MCTS algorithms consists of a significant number of variations of tree policies. Each of them tries to handle in an efficient way the dilemma between the exploitation of the currently gained knowledge about the actions and the exploration of the search space in order to improve that knowledge. In our project we use the most popular of them, the Upper Confidence Bound for Trees (UCT) algorithm.

Responding to the ANAC Diplomacy Challenge (de Jonge, ANAC 2019 Diplomacy Challenge Manual and Rules 2018), we aimed at implementing an MCTS algorithm for a Diplomacy playing agent. Diplomacy is a multi-agent strategic game of high complexity and is fully observable (no partial observability), but with simultaneous actions. It is played by seven (7) players and requires quite developed strategic and negotiation skills. Representing this domain as a tree seemed challenging because of the extremely high number of actions that can be taken from each state of the game. There is a need for a fine balance between the pruning of the tree (which greatly limits the search space) and the benefits of the generality of random sampling that MCTS offers.

We note that Diplomacy is a game of high complexity; though it is a game with full observability, the players’ actions are simultaneous. This renders any attempted MCTS implementation challenging, because each action (i.e., a player’s “move order” for this domain) has to be combined with the unknown beforehand, simultaneous actions of six other opponents, thus resulting in a non-predictable outcome.

Although ANAC Diplomacy Challenge focuses on the negotiation part of Diplomacy (participants should implement a negotiation algorithm that will be used on top of an existing Diplomacy playing agent), we managed to experiment with just the strategic part of the game and challenge the existing strategic Diplomacy agents that are used as base models at ANAC. We tried to enhance MCTS with various heuristic methods and we concluded to different versions of the algorithm that we present at this work.

In order to accomplish that, we used the BANDANA framework (de Jonge, The BANDANA Framework v1.3 2019). BANDANA is a Java framework for the development of automated agents that play the game of Diplomacy. The main advantage of BANDANA that concerns us for our project, is its clear distinction between the strategic module and the negotiating module of the agent. That helped us implement our algorithms for the strategic module and compete with other agents that this framework provides.

In our work we experiment with several variations of the MCTS algorithm, differentiating the amount and quality of domain knowledge in each of them, trying not to provide too much domain knowledge and, at the same time, making the algorithm worthwhile from the aspect of time needed for effective decision-making. Nevertheless, by importing a sorting system for the nodes' actions, we show that high quality domain knowledge is highly beneficial for the MCTS algorithm. At the core of our MCTS approach lies the well-known "bandit method" Upper Confidence Bounds for Trees (UCT) which attempts to balance between exploration and exploitation at the creation of a search tree.

We found the thesis of (Karamalegos 2016) remarkably helpful for our work here. In spite of the fact that it is an implementation of MCTS to a completely different domain (the Settlers of Catan board game), we based our algorithm on a similar structure.

To the best of our knowledge, this is the first work to implement algorithms of the Monte Carlo Tree Search family for the domain of "Diplomacy". Importantly, we provide a framework upon which further research of the problem can be based. We created a competitive algorithm and comparable to the other strategic algorithms of the domain and we managed to show that research on the topic of employing MCTS in the game of Diplomacy, and more broadly in multi-agent strategic board games, is promising.

In Chapter 2 we provide background on MCTS (Section 2.1) and the UCT tree policy (Section 2.2). Moreover, we describe the game "Diplomacy" (Section 2.3), the BANDANA framework that we used for our Java implementation (Section 2.4) and briefly review related work that helped us with our project (Section 2.5). In Chapter 3 we present our approach and in Chapter 4 we detail our experiments and discuss upon the results. Chapter 5 draws conclusions and Chapter 6 outlines possible future work.

2. Background

This section outlines the background theory that led to the development of Diplomacy agents using Monte Carlo Tree Search techniques. The topics that will be discussed are the Monte Carlo Tree Search (MCTS) algorithm, the Upper Confidence Bounds for Trees (UCT) method as the selected tree policy, the basic rules of the Diplomacy multi-agent strategic game and other work that we found related and helpful for our cause.

2.1. Monte Carlo Tree Search

“Monte Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking samples in the decision space and building a search tree according to the results” (Browne, et al. 2012). Especially in the last decade it is a popular choice at “Artificial Intelligence (AI) approaches for domains that can be represented as trees of sequential decisions” (Chaslot, et al. 2008).

After the great success of the implementation of MCTS in the challenging task of computer Go, researchers have been in the process of attaining a better understanding of which cases and why MCTS works better or worse. As such, the range of MCTS applications has increased further, as well as the number and quality of the basic algorithm’s refinements (Browne, et al. 2012).

The trade-off between exploration of the current domain and exploitation of the obtained knowledge regarding this domain is a well-known problem of reinforcement learning, known as Exploration - Exploitation dilemma. At the basic MCTS process, a tree is built in an incremental and nonsymmetric manner. For each iteration of the algorithm, a tree policy is used to find the most urgent node of the current tree. “The tree policy attempts to balance considerations of exploration (look in areas that have not been well sampled yet) and exploitation (look in areas which appear to be promising)” (Browne, et al. 2012). After the selection of the most urgent node (according to the tree policy), a new child node is added corresponding to the action taken from that node. Then a simulation is run from the new node and the search tree is updated according to the result of the game (or reward found at the end of the game). During the simulation, a default policy is used to determine the actions taken. The simplest policy is to make random actions. A benefit of MCTS is that the values of intermediate states, visited during the simulation step, do not have to be evaluated, as is the case with depth-limited minimax search, a fact that “greatly reduces the amount of domain knowledge required” (Browne, et al. 2012). The reward of the terminal state that is produced at the end of the simulation, is the only one required.

The basic algorithm involves iteratively building a search tree until some predefined computational budget – typically a time, memory or iteration constraint – is reached. At this point the search is halted and the most promising (according to the information gathered by simulation) action is returned (Browne, et al. 2012). Each node in the search tree represents a state of the domain. Actions are represented by the directed links to the child nodes which represent the subsequent states.

Four steps are applied per search iteration (Chaslot, et al. 2008):

- 1) **Selection:** Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unexpanded children.
- 2) **Expansion:** One (or more) child nodes are added to expand the tree, according to the available actions.
- 3) **Simulation:** A simulation is run from the new node(s), according to the default policy, to produce an outcome.
- 4) **Backpropagation:** The simulation result is backpropagated through the selected nodes to update their statistics.

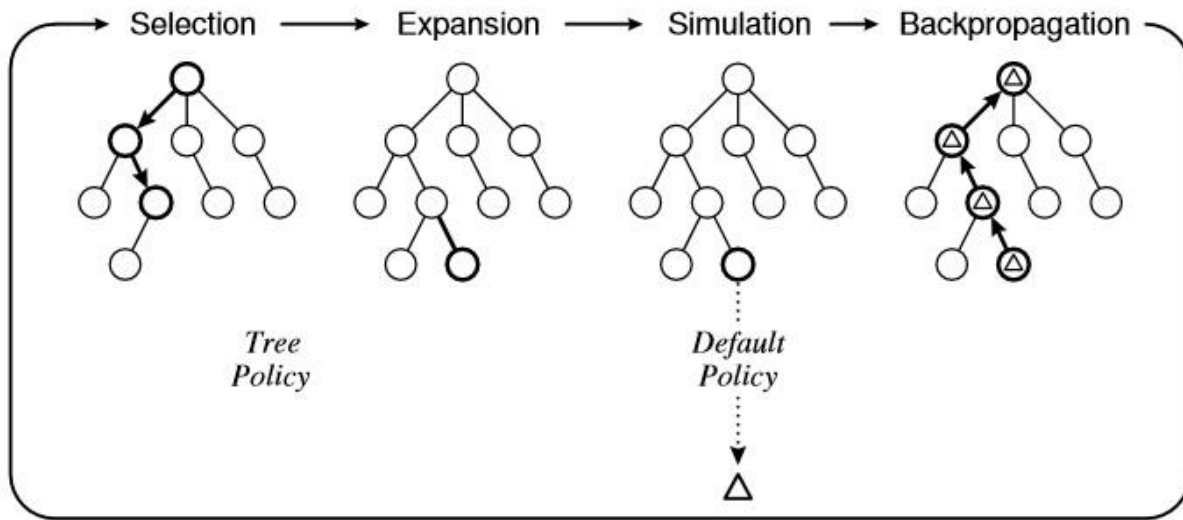


Figure 1. One iteration of the general MCTS approach. (Browne, et al. 2012)

The steps are summarized in pseudocode below (Browne, et al. 2012):

General MCTS approach

```

function  $M_{CTS}SEARCH(s_0)$ 
  create root node  $u_0$  with state  $s_0$ 
  while within computational budget do
     $u_i \leftarrow T_{TREE}POLICY(u_0)$ 
     $\Delta \leftarrow D_{DEFAULT}POLICY(s(u_i))$ 
     $B_{BACKUP}(u_i; \Delta)$ 
  return  $a(B_{EST}CHILD(u_0; 0))$ 

```

As such, “the goal of MCTS is to approximate the (true) game theoretic value of the actions that may be taken from the current state” (Browne, et al. 2012). This is accomplished by gradually building a partial search tree, using the four-step process that was described before and is shown in Figure 1. How the tree is built depends on the way that the nodes in the tree are selected. The tree policy is a key factor for the success of MCTS.

The family of MCTS algorithms consists of a significant amount of variations of tree policies. Each of them tries to handle in an efficient way the dilemma between the exploitation of the currently gained knowledge about the actions and the exploration of the search space in order to improve that knowledge. In our project we use the most popular of them, the Upper Confidence Bound for Trees (UCT) algorithm.

2.2. The Upper Confidence Bounds for Trees (UCT) algorithm

The UCT tree policy works as follows. As stated earlier, the key consideration is how to select a child node to expand. In UCT, a child node j is selected to maximize:

$$UCT = X_j + 2 * C_p * \sqrt{\frac{2 * \ln(n)}{Visits}},$$

where X_j is the normalized average reward ($0 \leq X_j \leq 1$),

$C_p > 0$ and is a constant. The value $C_p = \frac{1}{\sqrt{2}}$ was shown by Kocsis and Szepesvari (Kocsis and Szepesvari 2006), to satisfy the Hoeffding inequality (Hoeffding 1963) with rewards in the range $[0, 1]$.

n is the number of times the parent node has been visited, and

$Visits$ is the number of times current node has been visited.

We now present pseudocode for the complete UCT Monte Carlo Tree Search algorithm (Gelly and Silver 2006):

The UCT algorithm

```
function UCTSEARCH( $s_0$ )
    create root node  $v_0$  with state  $s_0$ 
    while within computational budget do
         $u_i \leftarrow \text{TREEPOLICY}(u_0)$ 
         $\Delta \leftarrow \text{DEFAULTPOLICY}(s(u_i))$ 
        BACKUP( $u_i$ ;  $\Delta$ )
    return a(BESTCHILD( $u_0$ ; 0))
```

```
function TREEPOLICY( $u$ )
    while  $u$  is nonterminal do
        if  $u$  not fully expanded then
            return EXPAND( $u$ )
        else
             $u \leftarrow \text{BESTCHILD}(u; C_p)$ 
    return  $u$ 
```

```
function EXPAND( $u$ )
    choose  $\alpha \in$  untried actions from  $A(s(u))$ 
```

```

add a new child  $u'$  to  $u$ 
    with  $s(u') = f(s(u), \alpha)$ 
    and  $a(u') = \alpha$ 
return  $u'$ 

```

function $B_{\text{ESTCHILD}}(u, c)$

```

return  $\operatorname{argmax}_{u' \in \text{children of } u} \frac{Q(u)}{N(u)} + c \sqrt{\frac{2 \ln N(u)}{N(u)}}$ 

```

function $D_{\text{EFAULTPOLICY}}(s)$

```

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

```

function $B_{\text{ACKUP}}(u, \Delta)$

```

while  $u$  is not null do
     $N(u) \leftarrow N(u) + 1$ 
     $Q(u) \leftarrow Q(u) + \Delta(u, p)$ 
     $u \leftarrow \text{parent of } u$ 

```

The selection of the node to expand next is made according to the UCT calculations above (method BestChild in the pseudocode). The rest of the algorithm proceeds as described in Section 2.1: if the node selected is expandable, an action from that node is chosen randomly and leads to a child node that is added to the tree. The data stored to the newly added node is the action a that led to it and the state s that represents the current state after the action a was taken. Then, the default policy is used until a terminal state has been reached. In the simplest case, this default policy is uniformly random. Following that, the value Δ of the terminal state is backpropagated to all nodes visited during this iteration, from the newly added node to the root (Browne, et al. 2012). X_j value and $Visits$ value of the visited node are updated.

2.3. Diplomacy strategic game

Diplomacy is an American strategic board game created by Allan B. Calhamer in 1954. In this section we present a brief overview of Diplomacy rules and gameplay:

Players

The game of Diplomacy is played by seven (7) players. Each player represents one of the seven “Great Powers of Europe” (Russia, Turkey, Austria, Italia, Great Britain, Germany, France) at the period before World War I.

Mapboard

The map is divided into 75 Provinces. Each Province is divided into one or more Regions. If the Province is *inland*, then it has just one Region on which only Armies can move. If the Province is *water*, then it has just one Region on which only Fleets can move. If the Province is *coastal*, then it has 2 or 3 Regions: one for the Armies and one for each coast on which only Fleets can move.

34 of the Provinces are Supply Centers.



Figure 2. Diplomacy Map. The dots indicate that this Province is a Supply Center and the colors that it is owned by the corresponding Great Power

Units (Armies – Fleets)

Each unit has equal strength with every other unit. During gameplay, the units can support each other so they can increase their strength. There can be only one unit in a Province at a time. If a Power has one of its units to a Province after order resolution, we say that it controls that Province.

At the beginning of the game each Power has 3 units, except Russia that has 4 units. The starting Provinces of each Power's units are called Home Provinces of that Power and can be seen at Figure 2.

Game Phases

Each round has a series of phases. The first round of the game is referred to as Spring 1901, followed by Fall 1901, Spring 1902, Fall 1902 etc. In each round all players simultaneously 'submit orders' for all of their units. Here are the phases in a complete two-turn year:

Spring

1. Diplomatic phase
2. Order writing phase
3. Order resolution Phase
4. Retreat and disbanding phase

Fall

1. Diplomatic phase
2. Order writing phase
3. Order resolution Phase
4. Retreat and disbanding phase
5. Gaining and losing units phase

Diplomatic Phase

During this phase, players are negotiating and making deals and agreements regarding their next moves. Alliances are made and strategies are set. "Conversations, deals, schemes and agreements will greatly affect the course of the game" (Calhamer 2000).

Order writing Phase

During this phase every player has to give a set of orders, one for every unit it has on the map. These orders can be:

- HOLD order: unit stays at the same Region.
- MOVETO order: unit moves to an adjacent Region.
- SUPPORT order: unit supports a HOLD or MOVETO order of another friendly unit.

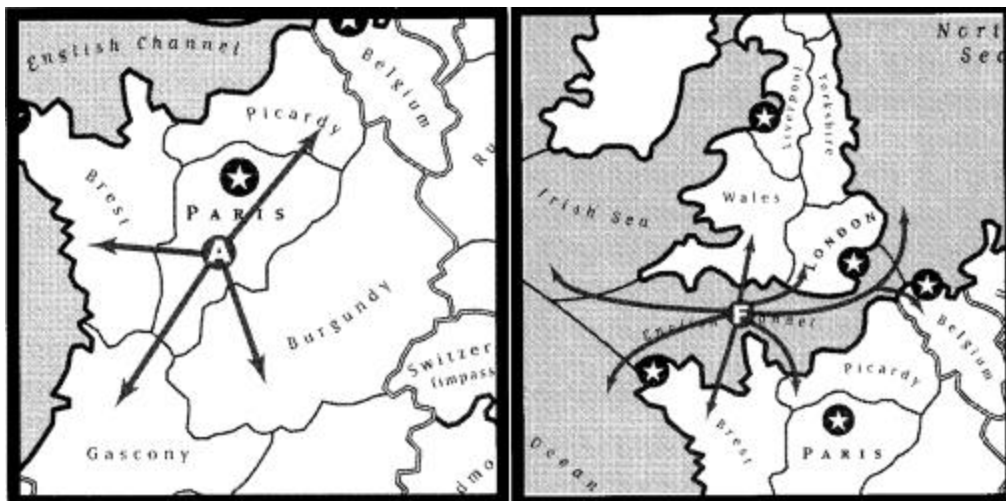


Figure 3: Examples of move orders resolution of an Army from Paris and a Fleet from English Channel. (Calhamer 2000)

Orders resolution Phase

Orders resolution has complicated rules that will not be discussed here entirely (please see <https://www.wizards.com/avalonhill/rules/diplomacy.pdf> for details). In this section we give just a short overview of them.

Each MOVETO order will fail if its strength (1+1 for every SUPPORT order for it) is less than or equal to the strength of an enemy's HOLD or MOVETO order with the same destination. If both strengths are equal, then the enemy's MOVETO order fails as well (standoff).

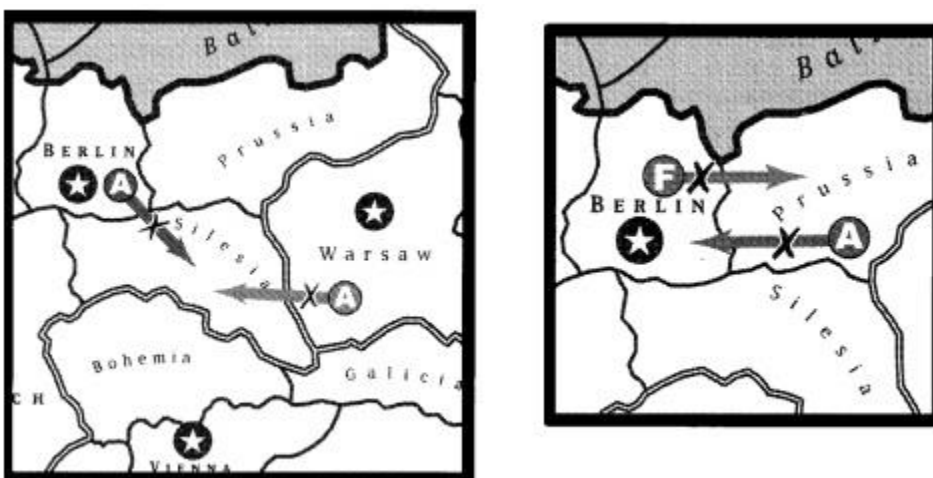


Figure 4: Examples of two move orders causing a standoff. At the first picture, both Armies try to move to Silesia, and at the second picture an Army from Prussia tries to move to Berlin and a Fleet from Berlin tries to move to Prussia. (Calhamer 2000)

Each SUPPORT order will fail (cutoff) if the located Province is attacked by an enemy unit.



Figure 5: The support from the Army in Silesia is cut by an attack from Bohemia. So, the move order of the Army from Prussia to Warsaw and the hold order of the Army at Warsaw cause a standoff. (Calhamer 2000)

Each HOLD order will fail if and only if there is a successful enemy's MOVETO order to the located Province.

In any other case, the order is successfully resolved and the unit reaches its destination or supports a friendly unit.

Every time a MOVETO order towards a Province that is the location of an enemy's SUPPORT/HOLD order or destination of an enemy's MOVETO order, succeeds, we say that we have a Dislodgement of that enemy unit.

Retreat and Disbanding Phase

During this phase, every player has to give a RETREAT order for every Dislodgement its Power has received. If there is no adjacent Region that the dislodged unit can retreat then the dislodged unit is removed from the map.

Gaining and Losing units Phase

At the end of every Fall Phase we update the number of Supply Centers each Power owns. If a Supply Center is controlled by a unit, then it is added to the unit's Power. If a Supply Center is not controlled by any unit then it remains to its previous owner if such an owner exists (some Supply Centers might be still unoccupied).

After we update the Supply Centers lists, we count the number of the units on the map of every Power. if a Power has more Supply Centers than units, then it has to give BUILD orders, indicating the Home Province(s) for the new unit(s) to appear (in the case that the Power controls none of its Home Provinces then nothing happens). If a Power has more units than Supply Centers, then it has to give REMOVE orders, indicating which unit(s) to be removed from the map.

End of the Game

A player is eliminated when he or she loses all his or her units. A player wins the game when he or she owns 18 supply centers (a *Solo Victory*), but a game may also end when all surviving players agree to a draw. Any player may propose a draw when he or she likes to.

At the ANAC Diplomacy Challenge (de Jonge, ANAC 2019 Diplomacy Challenge Manual and Rules 2018), there is a year limit that defines how many rounds will be played. If there is no winner until then, the players are ranked according to the number of Supply Centers that they own. We now give a short overview of the ANAC Diplomacy Challenge.

2.4. The ANAC Diplomacy Challenge

The International Automated Negotiating Agents Competition (ANAC) is an annual event, held in conjunction with the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), or the International Joint Conference on Artificial Intelligence (IJCAI), since 2010 (Negotiation 2019). ANAC is used by the automated negotiation research community to benchmark and evaluate its work and to challenge itself (Jonker, et al. 2018). ANAC includes five different negotiation research challenges (Tenth International Automated Negotiating Agents Competition, ANAC 2019 2019):

- Automated Negotiation League: Agent Negotiation with Partial Preferences
- Human-Agent League
- Supply Chain Management League
- Werewolf Game League
- Diplomacy League

The goal of the ANAC Diplomacy Challenge is the implementation of a negotiation algorithm on top of an existing Diplomacy playing agent (de Jonge, ANAC 2019 Diplomacy Challenge Manual and Rules 2019). That means, that the strategic component of the Diplomacy playing agent is fixed and same for all participants.

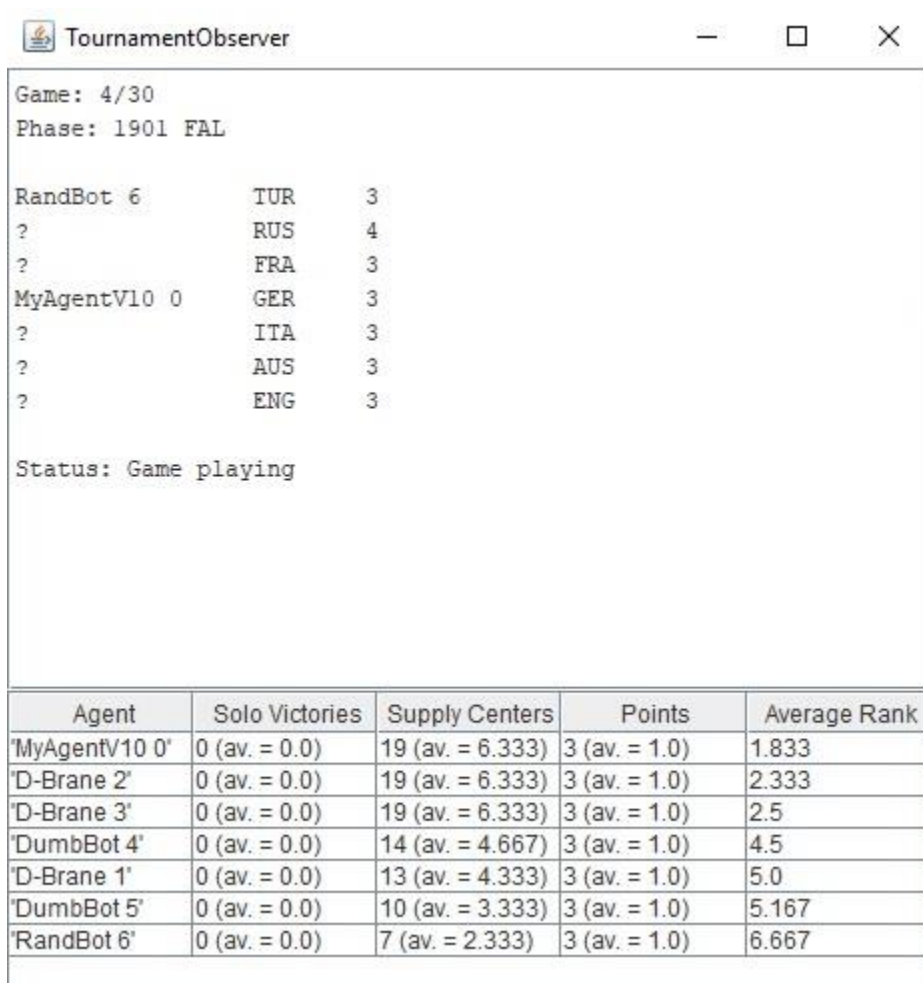
The competition is run in the BANDANA framework (de Jonge, The BANDANA Framework v1.3 2019). The BANDANA framework provides all the necessary Java classes to run a Diplomacy tournament and analyze the results. Furthermore, it provides the Java classes required to implement a negotiating Diplomacy agent, and some example agent implementations in order to help the participants struct their algorithm. Finally, it provides the executables of some negotiating and non-negotiating agents in order to help the participant test his/her implementation.

In the next section we provide a basic presentation of the BANDANA java framework.

2.5. BANDANA framework

For the purposes of this project we use the BANDANA framework (de Jonge, The BANDANA Framework v1.3 2019). BANDANA is a Java framework for the development of automated agents that play the game of Diplomacy. It is an extension of the DipGame framework (Fabregues 2011) However, it provides a new negotiation server and uses a simplified negotiation language. It is easy to use and makes the games and the tournaments observable. Figure 6 is a snapshot of TournamentObserver provided by BANDANA.

An advantage of the BANDANA framework is that it distinguishes between the strategic module and the negotiation module of a Diplomacy agent. That offers us the chance to isolate the module of our interest and evaluate the algorithm of each module while keeping all the other factors the same.



The screenshot shows a window titled "TournamentObserver" with a text area displaying game information and a table below it.

Game: 4/30
Phase: 1901 FAL

RandBot 6	TUR	3
?	RUS	4
?	FRA	3
MyAgentV10 0	GER	3
?	ITA	3
?	AUS	3
?	ENG	3

Status: Game playing

Agent	Solo Victories	Supply Centers	Points	Average Rank
'MyAgentV10 0'	0 (av. = 0.0)	19 (av. = 6.333)	3 (av. = 1.0)	1.833
'D-Brane 2'	0 (av. = 0.0)	19 (av. = 6.333)	3 (av. = 1.0)	2.333
'D-Brane 3'	0 (av. = 0.0)	19 (av. = 6.333)	3 (av. = 1.0)	2.5
'DumbBot 4'	0 (av. = 0.0)	14 (av. = 4.667)	3 (av. = 1.0)	4.5
'D-Brane 1'	0 (av. = 0.0)	13 (av. = 4.333)	3 (av. = 1.0)	5.0
'DumbBot 5'	0 (av. = 0.0)	10 (av. = 3.333)	3 (av. = 1.0)	5.167
'RandBot 6'	0 (av. = 0.0)	7 (av. = 2.333)	3 (av. = 1.0)	6.667

Figure 6: Running a 30-game tournament with BANDANA framework

Agent implementation

The agent implementation following the BANDANA framework is divided into the following sectors:

Spring / Fall phase: At these phases, the first thing an agent has to do is negotiate. That means, be able to handle incoming proposals and reject them, accept them, or make counter-proposals in a well-defined

amount of time. For the ANAC Diplomacy Challenge there is a well-defined restricted set of proposals the players can make:

- a) An Order Commitment,
- b) A Demilitarized Zone (A Demilitarized Zone consists of a phase, a year, a set of Powers and a set of Provinces, with the interpretation that none of these Powers is allowed to enter or stay inside any of these Provinces during that phase and year),
- c) Any combination of the above.

Apart from proposing this type of deals, an agent is also allowed to propose a draw to all the other surviving players, by calling the `proposeDraw()` method of the `ANACNegotiator` class. The game ends in a draw if all agents that have not been eliminated propose a draw in the same round of the game.

Negotiation will end up to a specific set of deals made (or an empty set of deals) that the agent will have to obey while determining the best plan. Given this set of deals and the current game state, the agent has to decide which is the best set of orders to give to its Power's units.

Summer / Autumn phase: Summer and Autumn phases stand for Retreat phase at Diplomacy. That means that the agent needs an algorithm to determine a retreat order for every Dislodgement its Power has received during the previous phase.

Winter phase: Winter phase stands for Gaining or losing units phase at Diplomacy (else Build phase). That means, that the agent has to decide for Build orders (from which Home Supply Centers should the gained units enter the map) if the number of its Power's units is greater than the number of owned Supply Centers, or Remove orders (which units should be removed from the map) if the number of its Power's units is less than the number of owned Supply Centers.

The BANDANA framework provides eleven (11) classes needed for running the tournaments and five (5) classes of agent implementations. The "TournamentRunner" class handles the game server and is the class that we need to modify according to the characteristics (e.g. agents setting, final year of each game, number of games) of the tournament that we want to run.

2.6. Related work

In this section we present some related work and experience that helped us either by offering ideas for the implementation of the Monte Carlo Tree Search algorithm, or by offering agents and algorithms to compete with.

2.6.1. Monte Carlo Tree Search in strategic games

Monte Carlo Tree Search algorithm is widely known for its application at Go. It has been used in conjunction with deep learning to build the successful and much-celebrated AlphaGo algorithm (Silver, et al. 2017). “Developed on October 2015, AlphaGo is the first computer program that defeated a professional human Go player, defeated a Go world champion, and is considered the strongest Go player in history” (Holcomb, et al. 2018).

Also, it has been used in various strategic games. We found helpful its application at the non-deterministic game "Settlers of Catan", a multi-player board-turned web-based game that necessitates strategic planning and negotiation skills (Karamalegos 2016), (Panousis 2014). In particular, we found the structure of the Engineering Diploma Thesis of Manos Karamalegos (Karamalegos 2016) to be extremely helpful, and as such we used some parts of its source code, such as UCT selection method.

2.6.2. Diplomacy agents

This section has descriptions of the agents we used as “enemies” for our experiments. We chose just three agents in order to simplify the experiments: an easy one (RandomBot), a moderate one (DumbBot), and the most successful one found in the literature to date (D-Brane). All of them are provided as executables by BANDANA framework.

2.6.2.1. *D-Brane*

D-Brane’s tactical module is used as the basis for the implementation of a negotiation algorithm for ANAC Diplomacy Challenge (de Jonge, ANAC 2019 Diplomacy Challenge Manual and Rules 2019). Since our goal is to implement an agent that can compete with others, D-Brane is considered to be the ultimate opponent and will be used for the evaluation of the results.

D-Brane uses a series of heuristics based on domain knowledge. We outline below the main algorithm of its strategic component (de Jonge, D-Brane: a Diplomacy Playing Agent for Automated Negotiations Research 2017):

- A deal is a set of actions that the players agree to make. In the BANDANA framework, the agents are forced to make the actions that are agreed in a deal. So, the deal is a restriction at an agent's set of possible actions. Each action that an agent takes, has to include the action indicated by the deal.
- Given a game state e (i.e. a configuration of units on the Diplomacy map), a deal x , and any player α_i the strategic component returns a set of orders for α_i for the game $\text{Dip}_e[x]$ plus the number of Supply Centers that α_i is guaranteed to conquer if it submits those orders.
- A battle plan for a player α_i to conquer a province p is "invincible" if no opponent can choose any battle plan that would prevent α_i from conquering p . One has determined all battle plans for a given province p , for all players, it is easy to determine which of those battle plans are invincible. Similarly, one can determine the invincible pairs of battle plans. An "invincible pair" is a pair of battle plans (β_p, β_q) for two different Supply Centers p and q respectively, such that if α_1 plays both of these battle plans, then at least one of them is guaranteed to succeed.
- If e denotes some state of the game the set of all battle plans for player α_i to conquer or defend a Supply Center p is denoted $B_{i,p}^e$. The set of all Supply Centers is denoted SC .
- Given a game state e , an agreement x , and a player α_i , the strategic component works as follows:
 1. For each $p \in SC$ determine all invincible plans from the set $B_{i,p}^e$.
 2. For all pairs of Supply Centers $(p,q) \in SC \times SC$ (with $p \neq q$) determine all invincible pairs from the set $B_{i,p}^e \times B_{i,q}^e$.
 3. Remove all invincible plans and invincible pairs that are not consistent with x .
 4. Find the largest consistent combination of invincible plans and pairs, using And/Or tree search (Dechter and Mateescu 2007) with Branch & Bound.
 5. For each province for which we have not been able to select an invincible plan or pair, select the strongest non-invincible battle plan that is consistent with x and the plans and pairs chosen in the previous step.
 6. Return the full set of battle plans we have selected.

The BANDANA framework provides the library with the D-Brane methods but not its source code.

2.6.2.2. *DumbBot and RandomBot*

DumbBot is a simple heuristic player developed by David Norman (Norman, <http://www.ellought.demon.co.uk/dipai> 2003). Although its implementation is fairly straightforward, it still plays reasonably well and is widely used as a default 'baseline' bot for researchers to compare their own bots with.

Note that the original DumbBot was implemented in C++ for the DAIDE framework (Norman, DAIDE 2002). The version that we use is an alternative JAVA implementation of that bot, implemented for the DipGame framework (Fabregues 2011). There is no guarantee that it is exactly identical to the original DumbBot.

DumbBot calculates a value for each region. Then, it makes a decision for an order for each unit, based on those values.

For each province, it calculates a value v as following:

- If the agent possesses this supply center: v = the size of the largest adjacent power
- If the agent does not possess this supply center: v = the size of the owning power
- If it is not a supply center: $v = 0$.

Then a table of Proximities is calculated as follows:

- Proximity[0] for each coast, is the above value v , multiplied by a weighting factor.
- Proximity[n] for each coast = (sum(proximity[n-1] for all adjacent coasts * proximity[n-1] for this coast) /5

Also, for each province, it calculates the number of adjacent units that the agent has (strength), and the number of adjacent units the enemies have (competition).

The DumbBot calculates an overall value for each region, based on all the proximity values for that region, and the strength and competition for that region, each of which is multiplied by a weighting.

For MOVE orders, the agent tries to move to the best adjacent region. However, there is a random chance that it moves to the second best, third best, etc. If the best place is where it already is, it holds. If the best place already has an occupying unit, or already has a unit moving there, it either supports that unit, or moves elsewhere, depending on whether the other unit is guaranteed to succeed or not.

For RETREAT orders, DumbBot uses the same rule as for MOVE orders (obviously without the SUPPORT orders).

For BUILD orders and DISBAND orders DumbBot uses the same region evaluation method. It builds in the highest value home Supply Center and disbands the unit in the region with the lowest value.

RandomBot is provided with the BANDANA framework as an example of a Diplomacy agent implementation. As the name indicates, it is an agent that just decides randomly which order to give to every unit. It also negotiates randomly. It makes random proposals and randomly approves or rejects the incoming proposals. It is also able to support friendly units when possible (and that makes its algorithm less random), but it uses no tactics at all.

DumbBot's source code is not accessible by the DipGame framework. In order to understand its basic mode of operation we used its C++ version of the bot (Norman, DAIDE 2002), although there is no guarantee that it is identical to its java version of DipGame framework.

RandomBot's source code is given by the BANDANA framework as the simplest Diplomacy agent implementation, in order to help with the way an agent implementation should be structured.

3. Our Approach

Our purpose in this thesis is to experiment exclusively with the strategic component of a Diplomacy agent. That means that our agent does not negotiate at all. As future work, negotiation algorithms could be used on this tactical base, but that presupposes a quite competitive strategic module. It has been shown that if the strategic module is not competitive enough then the agent will not be either, regardless of the negotiation part. The BANDANA framework is really helpful for that purpose, because it offers a clear distinction between the strategic module and the negotiation module. Our agent makes decisions for the order writing phases of the game (i.e. Spring and Fall Phase). The Retreat phase (i.e. Summer and Autumn Phase) it generates Random orders and the Build Phase (i.e. Winter Phase) it uses D-Brane tactics as the default strategy (the “DBraneTactics” class is provided by BANDANA framework’s library without its source code).

Our agent uses Monte Carlo Tree Search with UCT as its tree policy during the order writing phases of the game. As a heuristic, in each round it tries to maximize the number of Supply Centers conquered during that round. That (the number of the owned Supply Centers) is the only reward function that we take into account.

Each node has a reward value that is equal to the number of Supply Centers that our Power owns at this specific game state. During the Backpropagation step, this reward (normalized in order to be between 0 and 1) is used at every parent node (until root node is reached) to update its X_j value that is used for the UCT formula.

X_j is updated this way:

$$X_j' = \frac{R + \Delta}{Visits}, \quad (1)$$

where R is the previous cumulative number of Supply Centers that has been owned until now,

Δ is the number of Supply Centers that are “found” at the leaf node at the end of the simulation step and

$Visits$ is the number of times that this node has been visited.

In Figure 7, we show the process of tree expansion and simulation, and how the reward values are updated after some iterations of the algorithm.

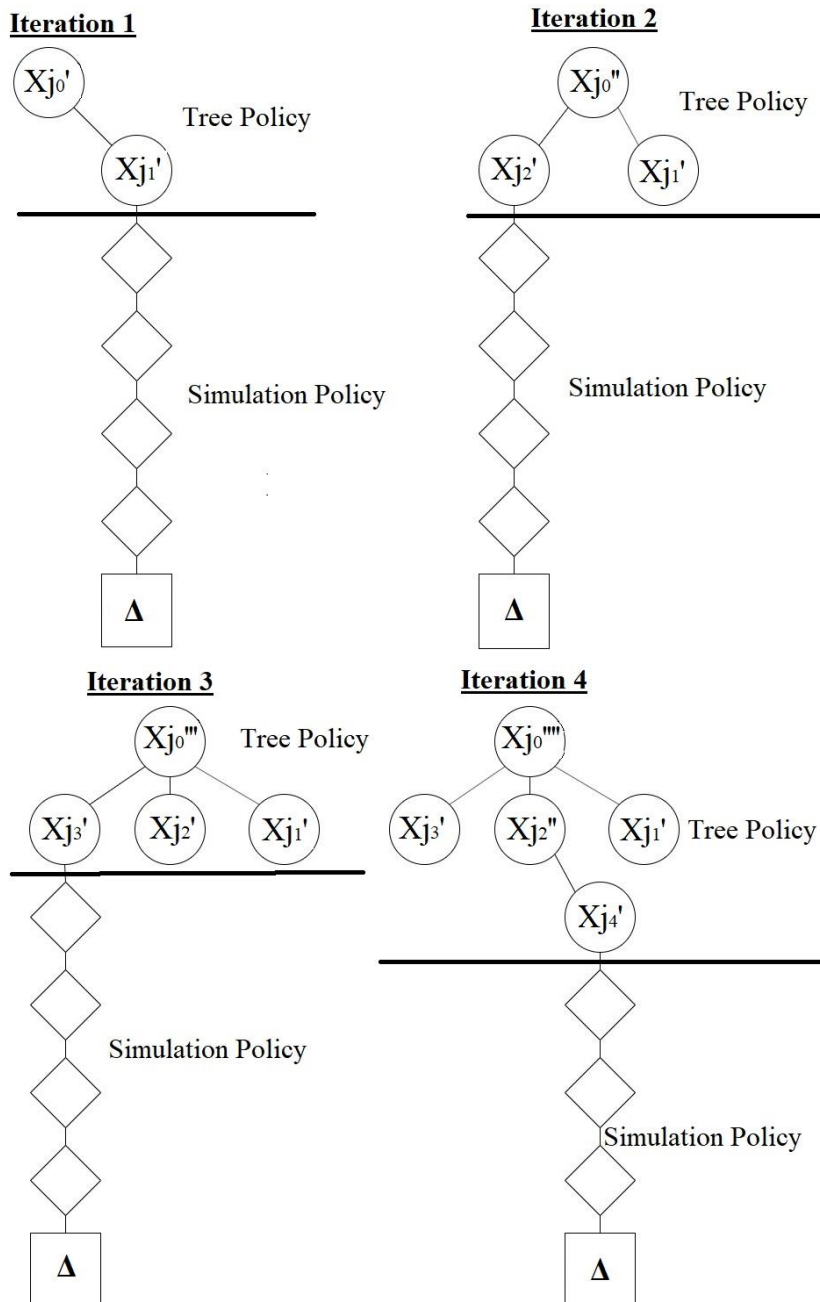


Figure 7: Example of the reward values updated after four iterations. X_{j_0}' represents the first update of X_j at node a , X_{j_0}'' represents the second update of X_j at node a etc.

The first time a node is selected by the Tree Policy, a function named *CreatePossibleActions* is called. This function creates a list with all possible actions that we allow to be taken from this node. It is computationally impossible to create a list with all the combinations of possible orders that can be given to the units on the map. So, we chose a fixed set of orders for all the enemies' units and then add it to each set of orders for our units. This choice is made by a heuristic that supposes that if the enemy unit is next to a Province with a Supply Center that it does not own, then it will move toward that Province.

Otherwise we do not care about the enemy unit movement and suppose a Hold order. Then we have to choose which orders to allow for our units. If all the units' possible orders are allowed then the number of possible actions will be extremely high when we have more than 7 or 8 units. In the next sections, different pruning methods, corresponding to different variants of our agent, will be discussed. The *CreatePossibleActions* function is summarized in pseudocode below:

The *CreatePossibleActions* algorithm

```

function CreatePossibleActions (v)
    get our Power p from node v
    get current state s from node v
    pos_ord  $\leftarrow$  PossibleOrders(p, s)
    ord_comb  $\leftarrow$  GetCombinations(pos_ord)
    exp_ord  $\leftarrow$  CalculateEnemyExpectedOrders(p, s)
    for each list of orders o in ord_comb
        combine o with exp_ord
    return ord_comb

```

A game state *s* is defined as the current layout of all units on the map and each Power's list of owned Supply Centers.

A Tree Node contains the current game state *s*, the action *a* that led to that game state (apart from the root node), a list of actions *ListA* that can be taken in order to expand the current node and create a new node, the reward value Δ (i.e. the number of Supply Centers our Power owns), and the reward X_j that is expected to be found if the agent chooses the action that led to that state.

The Monte Carlo Tree Search algorithm is repeated until a computational limit is reached. In our case this is a time limit that we set at eight (8) seconds (for some extra experiments at which we use a greater time limit, please see Figure 66 and Figure 68). It is obvious that alternative versions of the algorithm can benefit from alternative time limits, but for the purposes of this thesis (so that the results between agents could be comparable) we have chosen this particular time limit.

The main form of our decision-making algorithm of the MCTS_Agent is this:

- **SELECTION**

- Select between nodes sorted by: $X_j + 2 * C_p * \sqrt{\frac{2 * \ln(n)}{Visits}}$ (Browne, et al. 2012),

where X_j is the normalized average reward defined in Equation (1) above ($0 \leq X_j \leq 1$),

$$C_p = \frac{1}{2^{1/2}} \text{ (Kocsis and Szepesvari 2006),}$$

n is the number of times the parent node has been visited,

Visits is the number of times current node has been visited

- The *CreatePossibleActions* function is called once for each selected node. This function creates a list with all the accepted combinations of orders according to the pruning method that we apply. Each of these combinations of orders is an action that can be taken to expand the current node and create a child node with a new game state.
- **EXPANSION**
 - Pick an action from the selected node and remove it from its actions list.
 - Given this action, simulate one round of the game and create a child node with a new game state and a new reward.
- **SIMULATION**
 - Starting from the new node created at expansion step, simulate using random orders (default policy) or non-random orders (heuristic policy) until the game ends. The game ends either when our Power wins (i.e. reaches 18 Supply centers), or when our Power loses (i.e. reaches 0 Supply Centers), or when the game reaches the final year (by default this is set to 5 years of gameplay).
- **BACKPROPAGATION**
 - Starting from the last simulated node and using its reward value, update stats for each parent node until root node is reached.
- **REPEAT** this process for some predefined amount of time. For our case that is set to eight seconds (with the exception of some extra experiments, see Figure 66 and Figure 68).
- **SELECT** and return best child node (and the action that led to that node) according to the updated node rewards.

The flowchart of this algorithm can be seen at Figure 8.

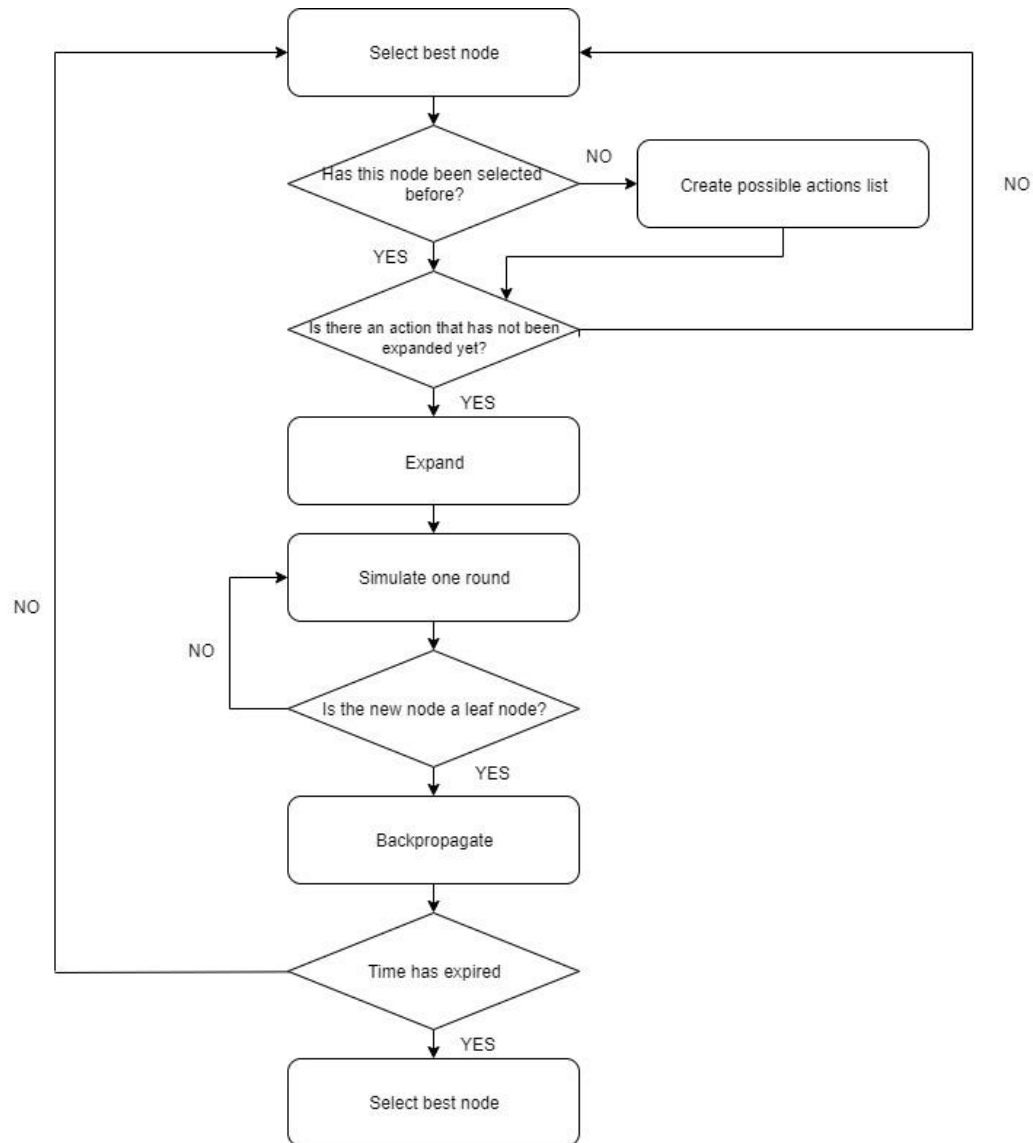


Figure 8: Flowchart of the basic algorithm

Game Simulation: once per expansion step and several times during the simulation step we need to resolve the game and create a new node with a new game state, given an action (set of orders). In order to do that we use two tools that the DipGame framework offers. *GameCloner* is used to create a new Game object in order to create a new tree node and *InternalAdjudicator* that helps at the order resolution.

Game simulation algorithm

- Clone the current game state (using *GameCloner*)
- Generate orders for all units on the map. We tried different versions of this step and experimented with heuristics at finding orders for our Power's units and for enemy Powers' units.

- Resolve orders using *InternalAdjudicator*. Internal Adjudicator takes as input a set of orders and creates a Boolean value for each of them which indicates whether the order was successful or not.
- Using the outcome of the Internal Adjudicator, we update the controlled Regions lists for each Power using the following algorithm:
 - First, we resolve the successful MOVETO orders and store them on a list ('conqueredRegions')
 - If a MOVETO or SUPPORT order fails, then we check on the conqueredRegions list. If the region (target region for MOVETO orders and location region for SUPPORT orders) was conquered then we have a dislodgement. Else we add the location region to the Power's controlled regions.
 - If a HOLD order fails then we have a dislodgement.
 - If a HOLD or SUPPORT order succeeds then we add the region that the order is located to the controlled regions.
- Generate random retreat orders for all Powers and resolve them
- If game phase is Fall, create random build orders and resolve them. To do that, we update the owned Supply Centers lists of every Power and we create random Build or Remove orders (depending on the number of Supply Centers owned and the number of units on the map).
- Update game phase and year
- Create child node with the new game state and the action (set of orders) that led to that state.

Now in the following sections we will present the different variants of the Monte Carlo Tree Search agent that we created in this thesis.

3.1. MCTS_Agent Version 1

This version of the MCTS_Agent uses the following pruning method at the *CreatePossibleActions* step of the Tree Policy:

- Gets all the "accepted" orders for each unit using a heuristic rule
 - If the unit stands on a Supply Center that we do not own, then return HOLD
 - If there is an order towards an adjacent SC then SUPPORT it
 - If an adjacent Province is SC that we do not own or control, then MOVE TO that SC.
 - Else return every possible MOVETO and HOLD order.
- Gets all the combinations of orders found
- Calculates enemies' expected orders using a dummy heuristic
 - If the unit is next to a SC then expect a MOVETO order towards that SC else expect a HOLD order.
- Creates a list with all possible actions from the current node

The *CreatePossibleActions* function used in MCTS_Agent Version 1, is summarized in pseudocode below:

The *CreatePossibleActions* algorithm

```
function CreatePossibleActions (v)
  get our Power p from node v
```

```

get current state  $s$  from node  $v$ 
 $pos\_ord \leftarrow \text{PossibleOrders}(p, s)$ 
 $list\_of\_combined\_orders \leftarrow \text{GetCombinations}(pos\_ord)$ 
 $exp\_ord \leftarrow \text{CalculateEnemyExpectedOrders}(p, s)$ 
for each list of orders  $o$  in  $list\_of\_combined\_orders$ 
    concatenate  $o$  with  $exp\_ord$ 
return  $list\_of\_combined\_orders$ 

```

```

function GetCombinations( $pos\_ord$ )
    for each  $order$  in  $pos\_ord(0)$ 
        add  $order$  to  $list\_of\_combined\_orders$ 
    for each  $list\_of\_orders$  in  $pos\_ord$ 
        clear  $newCombinedLists$ 
        for each  $first\_list$  in  $list\_of\_combined\_orders$ 
            for each  $order$  in  $list\_of\_orders$ 
                clear  $newList$ 
                add  $first\_list$  to  $newList$ 
                add  $order$  to  $newList$ 
                add  $newList$  to  $newCombinedLists$ 
        add  $newCombinedLists$  to  $list\_of\_combined\_orders$ 
    return  $list\_of\_combined\_orders$ 

```

```

function PossibleOrders( $p, s$ )
    get controlled regions  $units$  from  $p$ 
    get owned Supply centers  $sc$  from  $p$ 
    for each  $u$  in  $units$ 
        if  $u$  is Supply Center and  $u \notin sc$  then
            add HOLD( $u$ ) order to PossibleOrdersList( $u$ )
        else if  $\exists \text{ MOVETO}(u, reg)$  at PossibleOrdersList( $u$ ) |  $reg \in adjRegions(u)$  and  $reg$  is Supply Center and  $reg \notin sc$  and  $reg \notin units$  then
            add SUPPORT( $u, reg$ ) order to PossibleOrdersList( $u$ )
        else if  $\exists reg$  at  $adjRegions(u)$  |  $reg$  is Supply Center and  $reg \notin sc$  and  $reg \notin units$  then
            add MOVETO( $u, reg$ ) order to PossibleOrdersList( $u$ )
        else
            add every possible MOVETO and HOLD order to PossibleOrdersList( $u$ )
    return PossibleOrdersList

```

```

function CalculateEnemyExpectedOrders( $p, s$ )
    get controlled regions  $units$  from  $p$ 
    get controlled regions  $all\_units$  from  $s$ 
    for each  $u$  in  $all\_units$ 
        if  $u \notin units$  then
            if  $\exists reg$  at  $adjRegions(u)$  |  $reg$  is Supply Center then

```

```

        add MOVETO(u, reg) order at ExpectedOrdersList(u)
    else
        add HOLD(u) order at ExpectedOrdersList(u)
return ExpectedOrdersList

```

3.2. MCTS_Agent Version 2

This version of the MCTS_Agent has the same Tree Policy as the MCTS_Agent Version 1, but uses the heuristic of *PossibleOrders* as a simulation policy (see Figure 9 below) instead of random simulation (which is the default simulation policy). That means that the orders of our agent during simulation will not be created randomly. If the unit stands on a Province with a Supply Center that we do not own yet, then it is ordered to HOLD. If there is a Province with Supply Center next to a unit, then the unit will be ordered to MOVE to that Province. If there is a MOVETO order already towards that Province then the unit is ordered to SUPPORT that MOVETO order. Else the unit makes a random move.

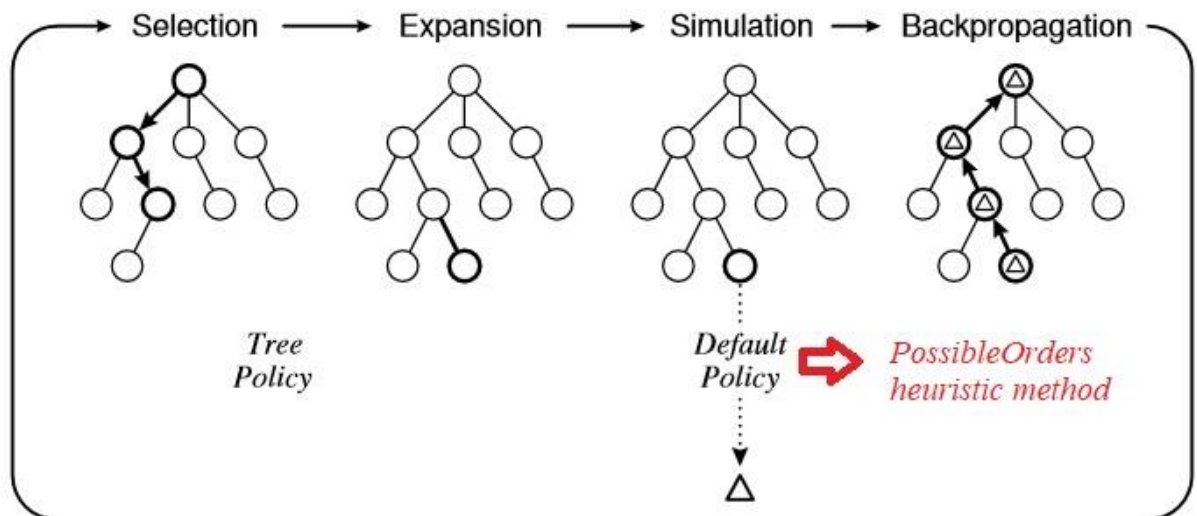


Figure 9: Use of an alternative simulation policy in MCTS_Agent Version 2.

3.3. MCTS_Agent Version 3

This version of the MCTS_Agent uses the *PossibleOrders* heuristic method as a simulation policy (as in MCTS_Agent Version 2) but alters it a bit. The new *PossibleOrders* function, when looking for Supply Centers at the adjacent Provinces, it also looks at the neighbors' neighbors. So, it locates Supply Centers that are two steps away from the current location. The new *PossibleOrders* function that is used as a simulation policy is summarized in pseudocode below:

The *PossibleOrders* algorithm

```

function PossibleOrders(p, s)
  get controlled regions units from p
  get owned Supply centers sc from p
  for each u in units
    if u is Supply Center and u  $\notin$  sc then
      add HOLD(u) order to PossibleOrdersList(u)
    else if  $\exists$  MOVETO(u, reg) at PossibleOrdersList(u) | reg  $\in$  adjRegions(u) and reg is Supply
    Center and reg  $\notin$  sc and reg  $\notin$  units then
      add SUPPORT(u, reg) order to PossibleOrdersList(u)
    else if  $\exists$  reg at adjRegions(u) | reg is Supply Center and reg  $\notin$  sc and reg  $\notin$  units then
      add MOVETO(u, reg) order to PossibleOrdersList(u)
    else if  $\exists$  reg at adjRegions(adjRegions(u)) | reg is Supply Center and reg  $\notin$  sc and reg  $\notin$ 
    units then
      get r | r  $\in$  adjRegions(u) and reg  $\in$  adjRegions(r)
      add MOVETO(u, r) order to PossibleOrdersList(u)
    else
      add every possible MOVETO and HOLD order to PossibleOrdersList(u)
  return PossibleOrdersList

```

3.4. MCTS_Agent Version 4

This version of the MCTS_Agent uses the following heuristic rule for the estimation of the enemies' expected orders during the simulation step: "If the enemy unit is next to a Supply Center, then expect a MOVETO order towards that Supply Center, else expect a random order". The calculation of our Power's orders during simulation is the same as in MCTS_Agent Version 2.

3.5. MCTS_Agent Version 5

This version of the MCTS_Agent has the same simulation policy as in MCTS_Agent Version 4, but has a significant change at the action selection in the Tree Policy. When a node is selected an action is chosen from its actions list in order to expand that node. This action is no longer selected randomly but the actions list is sorted in a certain way. Specifically, that happens at the *CreatePossibleActions* algorithm which is modified as follows, with the addition of a *CalculateWeights* method and a *CalculateActionValue* method:

The *CreatePossibleActions* algorithm

```

function CreatePossibleActions (v)
  get our Power p from node v
  get current state s from node v

```



```

CalculateWeights(p,s)
pos_ord ← PossibleOrders(p, s)
list_of_combined_orders ← GetCombinations(pos_ord)
exp_ord ← CalculateEnemyExpectedOrders(p, s)
for each list of orders o in ord_comb
    value ← CalculateActionValue(o)
    concatenate o with exp_ord
    put o and value at ActionsListMap
return ActionsListMap

```

```

function CalculateActionValue(list_of_orders)
    ActionValue ← 0
    for each o in list_of_orders
        add Weight(o) to ActionValue
    return ActionValue

```

The agent uses a weight system based on the DumbBot algorithm for move selection. With this algorithm, it keeps relative preferences for each Region on the map depending on the distance from a supply center, the amount of enemy forces around, and the phase of the game. The possible actions (actions that pass the pruning procedure) have a value which is the summary of the weights of each order's destination Region. The possible actions list is sorted according to those values. So, when a certain node is selected by the Monte Carlo Tree Search policy, the expanded action is always the "best" according to that weight system.

The calculation of each Region's weight is done as follows:

- For each province, it calculates the following as its value:
 - If it is our supply center, the size of the adjacent enemies' units
 - If it is not our supply center, a constant (i.e. 6) minus the size of the owning power
 - If it is not a supply center, zero.
- Proximity[0] for each Region, is the above value, multiplied by a weighting factor.
- Proximity[n] for each Region = (sum(proximity[n-1] for all adjacent Regions) / 5
+ proximity[n-1] for this coast)
* weighting
- The final weight for each Region is the sum value of its Proximity Table.

The Proximity table mentioned above is initialized using the following weighting factors:

```

// Importance of proximity[n] in Spring phase
m_spring_proximity_weight(0) = 10
m_spring_proximity_weight(1) = 100
m_spring_proximity_weight(2) = 3
m_spring_proximity_weight(3) = 1
m_spring_proximity_weight(4) = 0.6

```

```

m_spring_proximity_weight(5) = 0.5
m_spring_proximity_weight(6) = 0.4
m_spring_proximity_weight(7) = 0.3
m_spring_proximity_weight(8) = 0.2
m_spring_proximity_weight(9) = 0.1

// Importance of proximity[n] in Fall phase
m_fall_proximity_weight(0) = 100
m_fall_proximity_weight(1) = 10
m_fall_proximity_weight(2) = 3
m_fall_proximity_weight(3) = 1
m_fall_proximity_weight(4) = 0.6
m_fall_proximity_weight(5) = 0.5
m_fall_proximity_weight(6) = 0.4
m_fall_proximity_weight(7) = 0.3
m_fall_proximity_weight(8) = 0.2
m_fall_proximity_weight(9) = 0.1

```

Note: a Power can own a Supply Center only after one of its units spends a Fall Phase controlling it. That explains the difference between the first two positions of the Spring and Fall weight tables.

The process above can be summarized as follows. For each Region r , the proximity values are calculated first:

$$Proximity_r(0) = v * proximityWeight(0) \quad (2)$$

$$Proximity_r(n) = \left(\frac{\sum_{x \in adjRegions(r)} Proximity_x(n-1)}{5} + Proximity_r(n-1) \right) * proximityWeight(n) \quad (3)$$

and give rise to a regional weight:

$$Weight(r) = \sum_{n=0}^9 Proximity_r(n) \quad (4)$$

Then the importance of each action a is calculated as:

$$Value(a) = \sum_{r \in a} Weight(r) \quad (5)$$

The list of possible actions at every node is created as shown in *CreatePossibleActions* algorithm. The *CalculateWeights* method calculates all the regional weights as shown at the Equations (2)-(4). Then, after the calculation of each possible action, the *CalculateActionValue* method calculates the value of each action using Equation (5). Then, the list of possible actions along with their values are stored in the current node. At the expansion step, we always choose the action with the greatest value.

Also, at this version we added an extra provision in the pruning step at *PossibleOrders* function. If an order has for destination a region with value less than the mean value of all regions, then the order is pruned. If that leads to extensive pruning (no orders allowed) then we pick just the best order.

This agent is used as a basis for the next MCTS_Agent versions.

3.6. MCTS_Agent Version 6

In this version we make a slight change at the expansion step. After the expansion we expand again the newly created child in order to reach a greater depth, and then simulate from the second child node produced. That leads to a quite different proportion between the time spent for expansion and the time spent for simulation, in favor of the first. Other than that, the agent is exactly the same with MCTS_Agent Version 5.

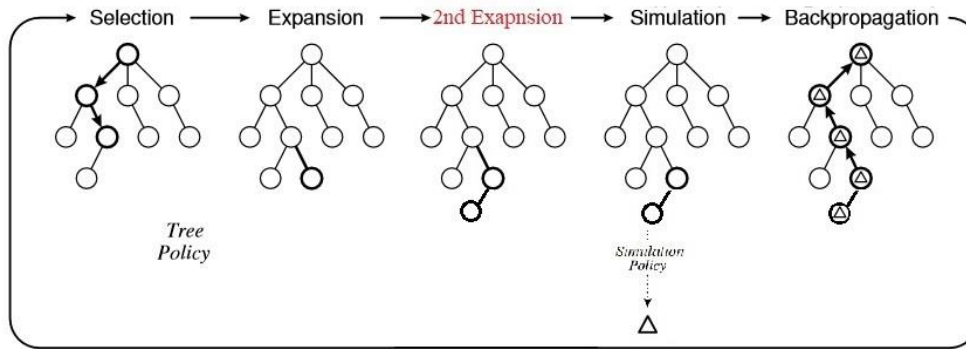


Figure 10: Monte Carlo Tree Search with two expansion steps

3.7. MCTS_Agent Version 7

In this version we use the weighting system defined in Section 3.5 and used for action selection on the Tree Policy, in the Simulation Policy also. Specifically, to create our agent's action during simulation, we choose the orders that maximize the action's value (Equation 5). Other than that, the agent is exactly the same with MCTS_Agent Version 5.

3.8. MCTS_Agent Version 8

In this version, we keep the *CreatePossibleActions* function as it is at MCTS_Agent Version 5 (Section 3.5) and we modify the simulation step. Instead of the MCTS_Agent playing against only one specific heuristic opponent (i.e., one behaving as specified by the *CalculateEnemyExpectedOrders* method in Section 3.1), we use a variety of opponents trying to achieve a better exploration of the search space (Matsubara and Marcolino 2011).

Thus, for this version, each opponent during the simulation is chosen randomly between three options: (A) Random Bot, (B) a heuristic using the weighting system as MCTS_Agent Version 7 does (see Section 3.7), (C) the simple and greedy heuristic of *CalculateExpectedOrders* function described at Section 3.1.

4. Experimental Evaluation

In this Chapter we will present all the experiments that we made in order to evaluate MCTS_Agent and the results of those experiments. We make a systematic comparison and evaluation of MCTS_Agents using RandomBot, DumbBot and D-Brane as opponents (see chapter 2.5.2 “Diplomacy Agents”).

4.1. Experimental Settings and Detailed Results

For the experiments, we test each version of the MCTS_Agent against a variety of combinations of opponents. For the most promising versions of MCTS_Agent (that are version 5, 6, 7 and 8) we test the agent against RandomBots, Dumbbots, against one, three and five D-Branes and the opposite: one, three and five MCTS_Agents against one D-Brane agent.

Experiments are run at the Parlance server using the BANDANA framework. The results are organized in tables with the following indications:

- Solo Victories: how many times it won the game by a solo victory (owned 18 Supply Centers or more).
- Supply Centers: how many Supply Centers it conquered in total (the number of Supply Centers it owned at the end of a game, summed over all games).
- Points: a player receives 0 points if it gets eliminated, 12 points for each solo victory, 6 points for each 2-player draw, 4 points for each 3-player draw, 3 points for a 4-player draw, 2 points for a 5-player or 6-player draw and 1 point for a 7-player draw.
- Average Rank: the average over all ranks it obtained in each game. A player obtains rank 1 in a game if it scored the highest number of Supply Centers, rank 2 if it scored the second highest number of Supply Centers, etcetera. If two players both end with 0 Supply Centers the players are ranked according to who was eliminated last. If two players rank equally, they both receive the average of the two ranks. (de Jonge, The BANDANA Framework v1.3 2019)

In each game the agents were randomly assigned to the 7 Powers. Every round of each game had a deadline of 30 seconds. That is the time that each agent has to decide its orders. Each game was run with a year limit of five (5) years (that means 10 order-writing phases). A draw was declared automatically in any game that advanced to the Winter 1905 phase. That limits the “depth” of the Monte Carlo Tree Search algorithm at simulation step. As we will see at the results, usually that time limit is not enough to achieve a Solo Victory (own 18 Supply Centers). So, given that limit, agents compete about who will end up with more Supply Centers from the others. Increasing the year limit would show in a clearer way the agents’ ranking because it would let the agents deploy their strategies in a greater timescale. Despite that fact, we decided not to increase the year limit because the time needed for the experiments would be

increased rapidly and our agents' ranking compared to the RandomBot, DumbBot and D-Brane agents is clear already.

We ran tournaments of a specific number of games because it was observed that after that number of games, the ranking of the agents did not change significantly. Obviously, the exact number of games to guarantee results' stability depends also on the version of our agent, still we ended up using a number of thirty (30) games per tournament that appears to provide stable results for all different agent versions.

When there is no Solo Victories indication, it means that no agent achieved a Solo Victory.

4.1.1. MCTS_Agents against Random Bots

D-Brane vs Random Bots

The following table (Figure 11) presents how D-Brane behaves against RandomBots. We observe that it diminishes this opponent. However, we observe that 5 years of gameplay are not enough for the D-Brane to achieve any Solo Victories. This table will be used as a baseline result in the sense that we would like to see MCTS_Agent achieve similar performance against RandomBots.

	Supply Centers	Points	Average Rank
'D-Brane'	299 (av. = 9.967)	33 (av. = 1.1)	1.233
'RandomBot 2'	120 (av. = 4.0)	31 (av. = 1.033)	4.033
'RandomBot 3'	111 (av. = 3.7)	33 (av. = 1.1)	4.217
'RandomBot 1'	111 (av. = 3.7)	33 (av. = 1.1)	4.533
'RandomBot 4'	107 (av. = 3.567)	29 (av. = 0.967)	4.433
'RandomBot 5'	104 (av. = 3.467)	33 (av. = 1.1)	4.75
'RandomBot 6'	101 (av. = 3.367)	33 (av. = 1.1)	4.8

Figure 11: D-Brane vs RandomBots. Average over 30 games.

MCTS_AgentV1 vs Random Bots

It is clear by the results presented below that MCTS_AgentV1 outplays RandomBot. What we also observe is that it does not manage to conquer more than a few Supply Centers. That means that it does not see “far” from its starting position and conquers just the Supply Centers that are near to it.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	179 (av. = 5.967)	30 (av. = 1.0)	2.167
'RandomBot 2'	136 (av. = 4.533)	30 (av. = 1.0)	3.817
'RandomBot 3'	129 (av. = 4.3)	30 (av. = 1.0)	4.15
'RandomBot 1'	127 (av. = 4.233)	30 (av. = 1.0)	4.2
'RandomBot 4'	126 (av. = 4.2)	30 (av. = 1.0)	4.0
'RandomBot 5'	116 (av. = 3.867)	30 (av. = 1.0)	4.45
'RandomBot 6'	114 (av. = 3.8)	30 (av. = 1.0)	4.767

Figure 12: MCTS_Agent V1 vs RandomBots. Average over 30 games.

MCTS_AgentV2 vs Random Bots

Version 2 of MCTS_Agent behaves similarly to version 1.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	156 (av. = 5.2)	32 (av. = 1.067)	3.283
'RandomBot 2'	140 (av. = 4.667)	32 (av. = 1.067)	3.733
'RandomBot 3'	139 (av. = 4.633)	32 (av. = 1.067)	3.7
'RandomBot 1'	133 (av. = 4.433)	32 (av. = 1.067)	4.133
'RandomBot 4'	124 (av. = 4.133)	32 (av. = 1.067)	4.383
'RandomBot 5'	121 (av. = 4.033)	32 (av. = 1.067)	4.433
'RandomBot 6'	117 (av. = 3.9)	28 (av. = 0.933)	4.333

Figure 13: MCTS_Agent V2 vs RandomBots. Average over 30 games.

MCTS_AgentV3 vs Random Bots

At this table, we see that Version 3 of MCTS_Agent does not behave differently from the previous versions.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	176 (av. = 5.867)	35 (av. = 1.167)	2.167
'RandomBot 2'	145 (av. = 4.833)	33 (av. = 1.1)	3.65
'RandomBot 3'	128 (av. = 4.267)	35 (av. = 1.167)	4.083
'RandomBot 1'	127 (av. = 4.233)	35 (av. = 1.167)	4.133
'RandomBot 4'	126 (av. = 4.2)	35 (av. = 1.167)	4.183
'RandomBot 5'	116 (av. = 3.867)	33 (av. = 1.1)	4.633
'RandomBot 6'	107 (av. = 3.567)	29 (av. = 0.967)	4.7

Figure 14: MCTS_Agent V3 vs RandomBots. Average over 30 games.

MCTS_AgentV4 vs Random Bots

Here we also observe no significant differences in performance for this agent version, as compared to the previous ones.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	165 (av. = 5.5)	30 (av. = 1.0)	2.933
'RandomBot 2'	130 (av. = 4.333)	30 (av. = 1.0)	3.667
'RandomBot 3'	124 (av. = 4.133)	30 (av. = 1.0)	4.233
'RandomBot 1'	124 (av. = 4.133)	30 (av. = 1.0)	4.3
'RandomBot 4'	122 (av. = 4.067)	30 (av. = 1.0)	4.3
'RandomBot 5'	118 (av. = 3.933)	30 (av. = 1.0)	4.117
'RandomBot 6'	117 (av. = 3.9)	30 (av. = 1.0)	4.45

Figure 15: MCTS_Agent V4 vs RandomBots. Average over 30 games.

MCTS_AgentV5 vs Random Bots

Version 5 of MCTS_Agent has a completely different behavior from the previous versions. Not only it has a high expansion rate, but it also achieved some Solo Victories.

	Supply Centers	Points	Average Rank	Solo Victory
'MCTS_Agent'	386 (av. = 12.467)	74 (av. = 2.467)	1.0	3 (av. = 0.1)
'RandomBot 2'	111 (av. = 3.7)	36 (av. = 1.2)	3.933	0
'RandomBot 3'	100 (av. = 3.333)	36 (av. = 1.2)	4,4	0
'RandomBot 1'	97 (av. = 3.233)	32 (av. = 1.067)	4.367	0
'RandomBot 4'	94 (av. = 3.133)	34 (av. = 1.133)	4.567	0
'RandomBot 5'	89 (av. = 2.967)	34 (av. = 1.133)	4.683	0
'RandomBot 6'	80 (av. = 2.667)	32 (av. = 1.067)	5.1	0

Figure 16: MCTS_Agent V5 vs RandomBots. Average over 30 games.

MCTS_AgentV6 vs Random Bots

Version 6 of MCTS_Agent has similar behavior to version 5. Again, we see fast expansion and some Solo Victories.

	Supply Centers	Points	Average Rank	Solo Victory
'MCTS_Agent'	378 (av. = 12.6)	74 (av. = 2.467)	1.033	3 (av. = 0.1)
'RandomBot 2'	128 (av. = 4.267)	36 (av. = 1.2)	3.7	0
'RandomBot 3'	124 (av. = 4.133)	36 (av. = 1.2)	4,1	0
'RandomBot 1'	102 (av. = 3.4)	32 (av. = 1.067)	4.033	0
'RandomBot 4'	88 (av. = 2.933)	34 (av. = 1.133)	4.967	0
'RandomBot 5'	80 (av. = 2.667)	34 (av. = 1.133)	5.1	0
'RandomBot 6'	78 (av. = 2.6)	32 (av. = 1.067)	5.067	0

Figure 17: MCTS_Agent V6 vs RandomBots. Average over 30 games.

MCTS_AgentV7 vs Random Bots

At Version 7 of MCTS_Agent we see a slightly better performance than the previous versions. It also achieves some Solo Victories.

	Supply Centers	Points	Average Rank	Solo Victory
'MCTS_Agent'	415 (av. = 13.833)	85 (av. = 2.833)	1.0	4 (av. = 0.133)
'RandomBot 2'	103 (av. = 3.433)	33 (av. = 1.1)	4.117	0
'RandomBot 3'	100 (av. = 3.333)	33 (av. = 1.1)	4,2	0
'RandomBot 1'	94 (av. = 3.133)	33 (av. = 1.1)	4.5	0
'RandomBot 4'	87 (av. = 2.9)	31 (av. = 1.033)	4.733	0
'RandomBot 5'	85 (av. = 2.833)	31 (av. = 1.033)	4.617	0
'RandomBot 6'	79 (av. = 2.633)	37 (av. = 1.233)	4.833	0

Figure 18: MCTS_Agent V7 vs RandomBots. Average over 30 games.

MCTS_AgentV8 vs Random Bots

At Version 8 of MCTS_Agent we observe no significant differences to Version 5 and 6.

	Supply Centers	Points	Average Rank	Solo Victory
'MCTS_Agent'	395 (av. = 13.167)	61 (av. = 2.033)	1.0	2 (av. = 0.067)
'RandomBot 2'	102 (av. = 3.4)	37 (av. = 1.233)	4.317	0
'RandomBot 3'	101 (av. = 3.367)	37 (av. = 1.233)	4,2	0
'RandomBot 1'	96 (av. = 3.2)	33 (av. = 1.1)	4.467	0
'RandomBot 4'	94 (av. = 3.133)	31 (av. = 1.033)	4.583	0
'RandomBot 5'	94 (av. = 3.133)	35 (av. = 1.167)	4.517	0
'RandomBot 6'	83 (av. = 2.767)	35 (av. = 1.167)	4.917	0

Figure 19: MCTS_Agent V8 vs RandomBots. Average over 30 games

Comparative Results versus Random Bots

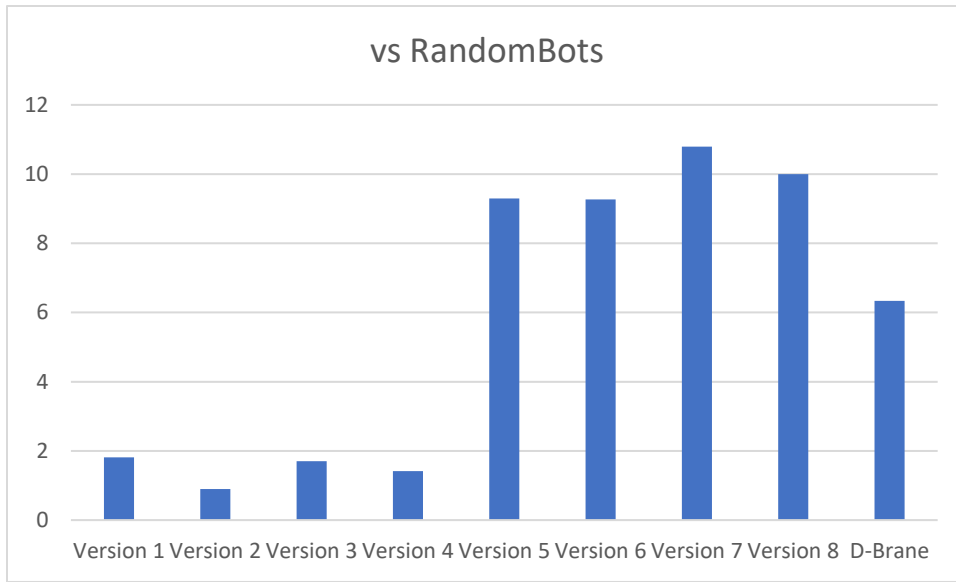


Figure 20: versions of the MCTS_Agent and D-Brane vs RandomBot. Performance according to the $SC_{agent} - SC_{RandomBot}$ metric.

Each of the values shown on the bars in the figure corresponds to the value of the following metric:

$$SC_{agent} - SC_{RandomBot} ,$$

where SC_{agent} is the average number of Supply Centers that the agent (MCTS_Agent or D-Brane) owns at a game and

$SC_{RandomBot}$ is the average number of Supply Centers that RandomBot owns at a game (when there are more than one RandomBots we take the mean value).

Firstly, we see that versions 1. 2. 3 and 4 of the MCTS_Agent outplay Random Bot but are not comparable with D-Brane. They only conquer the Supply Centers that are near to their starting positions and that is enough to outplay RandomBot. That happens because of the greedy heuristic used as a pruning method. MCTS algorithm does not help the agent to behave strategically in the long run.

On the other hand, we observe that versions 5. 6. 7 and 8 of MCTS_Agent expand quicker than D-Brane when playing against Random Bots. It is also important to highlight that they achieved some solo victories despite the strict year limit. At these versions, the algorithm used for sorting (see Version 5 of MCTS_Agent at Section 3.5) each node's actions list made the MCTS method efficient.

4.1.2. MCTS_Agents against Dumb Bots

D-Brane vs DumbBots:

The following table (Figure 21) presents how D-Brane behaves against DumbBots. We observe that it diminishes this opponent as well. This figure will be used for comparison in the sense that we would like to see MCTS_Agent achieve similar enrollment against DumbBots. As we will see in the following tables, with these settings (playing 5 year games against DumbBot) we observe no Solo Victories.

	Supply Centers	Points	Average Rank
'D-Brane'	165 (av. = 5.5)	35 (av. = 1.167)	2.183
'DumbBot 2'	116 (av. = 3.867)	31 (av. = 1.033)	4.0
'DumbBot 3'	113 (av. = 3.767)	35 (av. = 1.167)	3.917
'DumbBot 1'	109 (av. = 3.633)	33 (av. = 1.1)	4.333
'DumbBot 4'	105 (av. = 3.5)	35 (av. = 1.167)	4.4
'DumbBot 5'	103 (av. = 3.433)	33 (av. = 1.1)	4.317
'DumbBot 6'	97 (av. = 3.233)	33 (av. = 1.1)	4.85

Figure 21: D-Brane vs DumbBots. Average over 30 games.

MCTS_AgentV1 vs DumbBots

Version 1 of the MCTS_Agent does not manage to outplay DumbBot. DumbBot is a quite competitive agent and it takes more than a simple greedy heuristic to be outplayed.

	Supply Centers	Points	Average Rank
'DumbBot 2'	127 (av. = 4.233)	37 (av. = 1.233)	3.6
'DumbBot 3'	126 (av. = 4.2)	33 (av. = 1.1)	3.483
'DumbBot 1'	119 (av. = 3.967)	37 (av. = 1.233)	4.783
'DumbBot 4'	112 (av. = 3.733)	37 (av. = 1.233)	4.2
'DumbBot 5'	108 (av. = 3.6)	33 (av. = 1.1)	4.0
'MCTS_Agent'	96 (av. = 3.2)	37 (av. = 1.233)	4.567
'DumbBot 6'	95 (av. = 3.167)	29 (av. = 0.967)	4.367

Figure 22: MCTS_Agent V1 vs DumbBots. Average over 30 games.

MCTS_AgentV2 vs DumbBots

Version 2 of the MCTS_Agent behaves similarly to version 1.

	Supply Centers	Points	Average Rank
'DumbBot 2'	130 (av. = 4.333)	32 (av. = 1.067)	3.567
'DumbBot 3'	120 (av. = 4.0)	34 (av. = 1.133)	3.733
'DumbBot 1'	120 (av. = 4.0)	34 (av. = 1.133)	3.833
'DumbBot 4'	114 (av. = 3.8)	32 (av. = 1.067)	3.75
'DumbBot 5'	107 (av. = 3.567)	32 (av. = 1.067)	4.15
'MCTS_Agent'	101 (av. = 3.367)	34 (av. = 1.133)	4.483
'DumbBot 6'	97 (av. = 3.233)	32 (av. = 1.067)	4.483

Figure 23: MCTS_Agent V2 vs DumbBots. Average over 30 games.

MCTS_AgentV3 vs DumbBots

Version 2 of the MCTS_Agent does not behave differently from the previous versions of MCTS_Agent. It is still outplayed by DumbBot.

	Supply Centers	Points	Average Rank
'DumbBot 2'	135 (av. = 4.5)	33 (av. = 1.1)	3.2
'DumbBot 3'	124 (av. = 4.133)	33 (av. = 1.1)	3.617
'DumbBot 1'	112 (av. = 3.733)	31 (av. = 1.033)	4.0
'DumbBot 4'	111 (av. = 3.7)	29 (av. = 0.967)	4.0
'DumbBot 5'	108 (av. = 3.6)	33 (av. = 1.1)	3.983
'DumbBot 6'	101 (av. = 3.367)	33 (av. = 1.1)	4.517
'MCTS_Agent'	95 (av. = 3.167)	33 (av. = 1.1)	4.683

Figure 24: MCTS_Agent V3 vs DumbBots. Average over 30 games.

MCTS_AgentV4 vs DumbBots

Here we also observe no significant differences in performance for this agent version, as compared to the previous ones.

	Supply Centers	Points	Average Rank
'DumbBot 2'	130 (av. = 4.333)	36 (av. = 1.2)	3.55
'DumbBot 3'	122 (av. = 4.067)	34 (av. = 1.133)	3.617
'DumbBot 1'	122 (av. = 4.067)	34 (av. = 1.133)	3.75
'DumbBot 4'	114 (av. = 3.8)	36 (av. = 1.2)	4.017
'DumbBot 5'	111 (av. = 3.7)	34 (av. = 1.133)	3.967
'DumbBot 6'	106 (av. = 3.533)	30 (av. = 1.0)	4.167
'MCTS_Agent'	93 (av. = 3.1)	36 (av. = 1.2)	4.933

Figure 25: MCTS_Agent V4 vs DumbBots. Average over 30 games.

MCTS_AgentV5 vs DumbBots

Finally, this version of our MCTS_Agent manages to outperform the DumbBot agents. The results listed in this table will be used to compare between D-Brane and MCTS_Agent dominance over DumbBot.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	150 (av. = 5.0)	33 (av. = 1.1)	3.067
'DumbBot 2'	124 (av. = 4.133)	37 (av. = 1.233)	3.85
'DumbBot 3'	124 (av. = 4.133)	35 (av. = 1.167)	3.817
'DumbBot 1'	116 (av. = 3.867)	37 (av. = 1.233)	4.133
'DumbBot 4'	116 (av. = 3.867)	33 (av. = 1.1)	4.25
'DumbBot 5'	105 (av. = 3.5)	33 (av. = 1.1)	4.283
'DumbBot 6'	104 (av. = 3.467)	37 (av. = 1.233)	4.6

Figure 26: MCTS_Agent V5 vs DumbBots. Average over 30 games.

MCTS_AgentV6 vs DumbBots

Version 6 of MCTS_Agent also outperforms DumbBot. However, we see a slight improvement with respect to the performance of Version 5 of MCTS_Agent when facing DumbBots (see Figure 26).

	Supply Centers	Points	Average Rank
'MCTS_Agent'	156 (av. = 5.2)	33 (av. = 1.1)	2,783
'DumbBot 2'	132 (av. = 4.4)	35 (av. = 1.167)	3.8
'DumbBot 3'	125 (av. = 4.167)	33 (av. = 1.1)	4,067
'DumbBot 1'	122 (av. = 4.067)	33 (av. = 1.1)	3,667
'DumbBot 4'	103 (av. = 3.433)	35 (av. = 1.167)	4.7
'DumbBot 5'	102 (av. = 3.4)	35 (av. = 1.167)	4.4
'DumbBot 6'	102 (av. = 3.4)	31 (av. = 1.033)	4.583

Figure 27: MCTS_Agent V6 vs DumbBots. Average over 30 games.

MCTS_AgentV7 vs DumbBots

Versions 7 of MCTS_Agent also outperforms DumbBot. However, we can see both in terms of the Average Rank and of Supply Centers won, that there is a slight deterioration with respect to the performance of Versions 5 and 6 of the MCTS_Agent when facing DumbBots.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	145 (av. = 4.833)	33 (av. = 1.1)	3.167
'DumbBot 2'	127 (av. = 4.233)	33 (av. = 1.1)	3.583
'DumbBot 3'	119 (av. = 3.967)	33 (av. = 1.1)	4.017
'DumbBot 1'	115 (av. = 3.833)	31 (av. = 1.033)	4.283
'DumbBot 4'	114 (av. = 3.8)	31 (av. = 1.033)	4.05
'DumbBot 5'	105 (av. = 3.5)	31 (av. = 1.033)	4.45
'DumbBot 6'	102 (av. = 3.4)	33 (av. = 1.1)	4.45

Figure 28: MCTS_Agent V7 vs DumbBots. Average over 30 games.

MCTS_AgentV8 vs DumbBots

At Version 8 of MCTS_Agent we observe a slight improvement in its performance in terms of Supply Centers occupied, compared to the previous versions.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	157 (av. = 5.233)	35 (av. = 1.167)	2,867
'DumbBot 2'	134 (av. = 4.467)	37 (av. = 1.233)	3.367
'DumbBot 3'	113 (av. = 3.767)	35 (av. = 1.167)	4,2
'DumbBot 1'	112 (av. = 3.733)	35 (av. = 1.167)	4.3
'DumbBot 4'	111 (av. = 3.7)	35 (av. = 1.167)	4.283
'DumbBot 5'	109 (av. = 3.633)	35 (av. = 1.167)	4.25
'DumbBot 6'	94 (av. = 3.133)	33 (av. = 1.1)	4.733

Figure 29: MCTS_Agent V8 vs DumbBots. Average over 30 games.

Comparative Results versus DumbBots

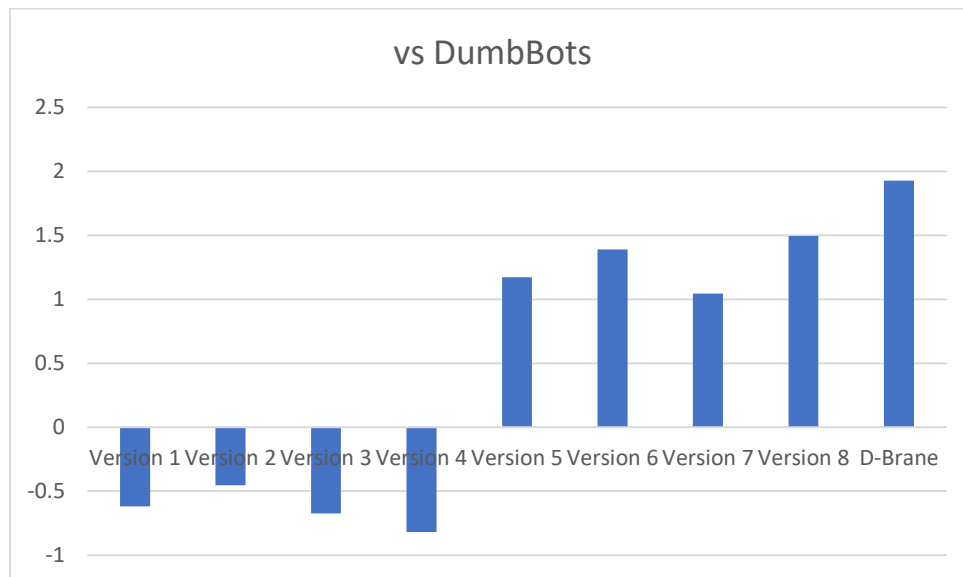


Figure 30: versions of the MCTS_Agent and D-Brane vs DumbBot. Performance according to the $SC_{agent} - SC_{DumbBot}$ metric.

Each of the values shown on the bars in the figure corresponds to the value of the following metric:

$$SC_{\text{agent}} - SC_{\text{DumbBot}},$$

where SC_{agent} is the average number of Supply Centers that the agent (MCTS_Agent or D-Brane) owns at a game and

SC_{DumbBot} is the average number of Supply Centers that DumbBot owns at a game (when there are more than one DumbBots we take the mean value).

First thing we observe is that the first 4 versions of MCTS_Agent are outplayed by DumbBot. Then we see that versions 5, 6, 7 and 8 clearly outperform DumbBot but score less than D-Brane. DumbBot is a competitive agent so we do not see the same expansion rate as we saw when MCTS_Agents played against RandomBots. That is why MCTS_Agent has a better score than D-Brane in Figure 20 and worse score at Figure 30.

4.1.3. MCTS_Agents against D-Brane

In this section, we present results on how our agents fare against facing D-Brane. To this purpose, we ran tournaments where our agents face different number of D-Brane agents, DumbBots, and RandomBots.

Our results show that the first four versions of our agents do not do well, as was expected given our previous observations. For this reason, the number of experiments we present for the first four versions against D-Brane is limited, when compared with the experiments we present for versions 5, 6, 7, and 8.

We see that for agent versions 1,2,3,4, the MCTS algorithm cannot find any competitive move to make. It outperforms random bot due to the pruning method that obliges the agent to attack any Supply Center next to it. If there is not a Supply Center near to it, then it moves quite randomly.

However, the situation changes drastically for agents 5, 6, 7 and 8, as we show in Sections 4.1.3.5, 4.1.3.6, 4.1.3.7 and 4.1.3.8 respectively.

4.1.3.1. Version 1:

	Supply Centers	Points	Average Rank
'D-Brane 1'	268 (av. = 8.933)	46 (av. = 1.533)	1.617
'D-Brane 0'	260 (av. = 8.667)	46 (av. = 1.533)	1.95
'MCTS_Agent'	141 (av. = 4.7)	46 (av. = 1.533)	3.35
'RandomNegotiator 5'	86 (av. = 2.867)	42 (av. = 1.4)	5.017
'RandomNegotiator 4'	83 (av. = 2.767)	36 (av. = 1.2)	5.2
'RandomNegotiator 3'	71 (av. = 2.367)	34 (av. = 1.133)	5.35
'RandomNegotiator 6'	70 (av. = 2.333)	34 (av. = 1.133)	5.517

Figure 31: MCTS_Agent V1 vs 2 D-Brane and 4 RandomBots. Average over 30 games.

4.1.3.2. Version 2:

As it was mentioned before, Version 2 of MCTS_Agent does not have a competitive performance as it does not outperform DumbBot.

	Supply Centers	Points	Average Rank
'D-Brane 4'	180 (av. = 6.0)	37 (av. = 1.233)	2.4
'DumbBot 3'	151 (av. = 5.033)	35 (av. = 1.167)	3.333
'DumbBot 2'	142 (av. = 4.733)	31 (av. = 1.033)	3.567
'DumbBot 1'	128 (av. = 4.267)	35 (av. = 1.167)	3.817
'MCTS_Agent'	114 (av. = 3.8)	37 (av. = 1.233)	4.15
'RandomNegotiator 5'	81 (av. = 2.7)	35 (av. = 1.167)	5.5
'RandomNegotiator 6'	80 (av. = 2.667)	35 (av. = 1.167)	5.233

Figure 32: MCTS_Agent V2 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.

As it was expected, MCTS_Agent cannot outperform D-Brane.

	Supply Centers	Points	Average Rank
'D-Brane 2'	272 (av. = 9.067)	42 (av. = 1.4)	1.6
'D-Brane 1'	257 (av. = 8.567)	42 (av. = 1.4)	1.8
'MCTS_Agent'	141 (av. = 4.7)	42 (av. = 1.4)	3.383
'RandomNegotiator 5'	82 (av. = 2.733)	42 (av. = 1.4)	5.217
'RandomNegotiator 6'	81 (av. = 2.7)	36 (av. = 1.2)	5.15
'RandomNegotiator 3'	77 (av. = 2.567)	28 (av. = 0.933)	5.317
'RandomNegotiator 4'	70 (av. = 2.333)	36 (av. = 1.2)	5.533

Figure 33: MCTS_Agent V2 vs 2 D-Brane and 4 RandomBots. Average over 30 games.

4.1.3.3. Version 3:

Version 3 of MCTS_Agent also is outperformed by DumbBot. We observe no significant difference from the previous versions of the MCTS_Agent.

	Supply Centers	Points	Average Rank
'D-Brane 4'	192 (av. = 6.4)	39 (av. = 1.3)	2.15
'DumbBot 2'	156 (av. = 5.2)	37 (av. = 1.233)	3.183
'DumbBot 3'	150 (av. = 5)	39 (av. = 1.3)	3.7
'DumbBot 1'	145 (av. = 4.833)	37 (av. = 1.233)	3.383
'MCTS_Agent'	112 (av. = 3.733)	39 (av. = 1.3)	4.233
'RandomNegotiator 5'	73 (av. = 2.433)	33 (av. = 1.1)	5.6
'RandomNegotiator 6'	65 (av. = 2.167)	27 (av. = 0.9)	5.75

Figure 34: MCTS_Agent V3 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.

Playing against more RandomBots makes the MCTS_Agent occupy more Supply Centers than, for example, in Figure 34. However, it is still diminished by D-Brane.

	Supply Centers	Points	Average Rank
'D-Brane 1'	284 (av. = 9.467)	46 (av. = 1.533)	1.6
'D-Brane 2'	257 (av. = 8.567)	46 (av. = 1.533)	1.833
'MCTS_Agent'	137 (av. = 4.567)	46 (av. = 1.533)	3.783
'RandomNegotiator 4'	100 (av. = 3.333)	42 (av. = 1.4)	4.733
'RandomNegotiator 3'	85 (av. = 2.833)	35 (av. = 1.167)	4.883
'RandomNegotiator 5'	72 (av. = 2.4)	35 (av. = 1.167)	5.383
'RandomNegotiator 6'	60 (av. = 2.0)	31 (av. = 1.033)	5.783

Figure 35: MCTS_Agent V3 vs 2 D-Brane and 4 RandomBots. Average over 30 games.

4.1.3.4. Version 4:

Version 4 of MCTS_Agent is not able to outperform DumbBot. There is no significant difference from the previous versions.

	Supply Centers	Points	Average Rank
'D-Brane 4'	212 (av. = 7.067)	42 (av. = 1.4)	1.983
'DumbBot 3'	187 (av. = 6.233)	42 (av. = 1.4)	2.417
'DumbBot 2'	135 (av. = 4.5)	42 (av. = 1.4)	3.7
'DumbBot 1'	134 (av. = 4.467)	40 (av. = 1.333)	3.85
'MCTS_Agent'	111 (av. = 3.7)	42 (av. = 1.4)	4.45
'RandomNegotiator 5'	70 (av. = 2.333)	32 (av. = 1.067)	5.517
'RandomNegotiator 6'	58 (av. = 1.933)	30 (av. = 1.0)	6.083

Figure 36: MCTS_Agent V4 vs 1 D-Brane, 3 DumbBots and 2 RandomBots. Average over 30 games.

Again, the MCTS_Agent is clearly outperformed by D-Brane.

	Supply Centers	Points	Average Rank
'D-Brane 2'	274 (av. = 9.133)	46 (av. = 1.533)	1.75
'D-Brane 1'	268 (av. = 8.933)	46 (av. = 1.533)	1.817
'MCTS_Agent'	143 (av. = 4.767)	46 (av. = 1.533)	3.467
'RandomNegotiator 3'	87 (av. = 2.9)	42 (av. = 1.4)	4.85
'RandomNegotiator 5'	75 (av. = 2.5)	38 (av. = 1.267)	5.2
'RandomNegotiator 6'	67 (av. = 2.233)	38 (av. = 1.267)	5.467
'RandomNegotiator 4'	66 (av. = 2.2)	30 (av. = 1.0)	5.45

Figure 37: MCTS_Agent V4 vs 2 D-Brane and 4 RandomBots. Average over 30 games.

4.1.3.5. Version 5:

The implementation of the weight system at the action selection part of the algorithm (see Section 3.5), made the agent iterate through the most promising nodes first, and then look at the others. For the first time, we outperform DumbBot and we have a competitive algorithm against D-Brane.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	220 (av. = 7.333)	37 (av. = 1.233)	2.2
'D-Brane 4'	209 (av. = 6.967)	37 (av. = 1.233)	2.233
'DumbBot 2'	161 (av. = 5.367)	37 (av. = 1.233)	3.367
'DumbBot 1'	139 (av. = 4.633)	37 (av. = 1.233)	3.65
'RandomNegotiator 3'	78 (av. = 2.6)	33 (av. = 1.1)	5.5
'RandomNegotiator 5'	76 (av. = 2.533)	33 (av. = 1.1)	5.433
'RandomNegotiator 6'	76 (av. = 2.533)	27 (av. = 0.9)	5.617

Figure 38: MCTS_Agent V5 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.

This table confirms that Version 5 of MCTS_Agent is a competitive agent. It shows a descent result when playing against 3 D-Brane agents.

	Supply Centers	Points	Average Rank
'D-Brane 2'	211 (av. = 7.033)	44 (av. = 1.467)	2.567
'MCTS_Agent'	203 (av. = 6.767)	44 (av. = 1.467)	2.617
'D-Brane 1'	203 (av. = 6.767)	44 (av. = 1.467)	2.8
'D-Brane 3'	202 (av. = 6.733)	44 (av. = 1.467)	2.683
'RandomNegotiator 5'	70 (av. = 2.333)	27 (av. = 0.9)	5.517
'RandomNegotiator 3'	63 (av. = 2.1)	35 (av. = 1.167)	5.817
'RandomNegotiator 4'	60 (av. = 2.0)	29 (av. = 0.967)	6.0

Figure 39: MCTS_Agent V5 vs 3 D-Brane and 3 RandomBots. Average over 30 games.

This setting includes 2 DumbBots and 1 RandomBot in the place of the 3 RandomBots that we saw in the previous table (see Figure 39). DumbBot is a competitive agent that makes it difficult for the other agents to expand. We observe that both D-Brane's and MCTS_Agent's performance is worse than it was in the previous table.

	Supply Centers	Points	Average Rank
'D-Brane 2'	175 (av. = 5.833)	35 (av. = 1.167)	2.917
'D-Brane 1'	175 (av. = 5.833)	35 (av. = 1.167)	3.283
'D-Brane 3'	171 (av. = 5.7)	35 (av. = 1.167)	3.083
'MCTS_Agent'	159 (av. = 5.3)	35 (av. = 1.167)	3.533
'DumbBot 2'	133 (av. = 4.467)	31 (av. = 1.033)	4.467
'DumbBot 1'	113 (av. = 3.767)	33 (av. = 1.1)	4.667
'RandomNegotiator 6'	67 (av. = 2.233)	31 (av. = 1.033)	6.05

Figure 40: MCTS_Agent V5 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

We see that when we play against many competitive agents (many D-Branes and many DumbBots for this experiment), it worsens our results. This is clearer in Figure 41 below.

	Supply Centers	Points	Average Rank
'D-Brane 2'	169 (av. = 5.633)	35 (av. = 1.167)	3.233
'D-Brane 1'	166 (av. = 5.533)	35 (av. = 1.167)	3.283
'D-Brane 3'	157 (av. = 5.233)	35 (av. = 1.167)	3.417
'D-Brane 4'	152 (av. = 5.067)	35 (av. = 1.167)	3.817
'D-Brane 5'	151 (av. = 5.033)	35 (av. = 1.167)	3.983
'MCTS_Agent'	138 (av. = 4.6)	35 (av. = 1.167)	4.217
DumbBot 6'	81 (av. = 2.7)	25 (av. = 0.833)	6.05

Figure 41: MCTS_Agent V5 vs 5 D-Brane and 1 DumbBot. Average over 30 games.

The MCTS performance also deteriorates when MCTS_Agent plays against itself (Figures 42 and 43). Multiple instances of competitive agents decrease the performance of the MCTS_Agent against D-Brane.

	Supply Centers	Points	Average Rank
'D-Brane 0'	180 (av. = 6.0)	34 (av. = 1.133)	2.617
'MCTS_Agent 1'	171 (av. = 5.7)	34 (av. = 1.133)	3.217
'MCTS_Agent 2'	159 (av. = 5.3)	34 (av. = 1.133)	3.533
'MCTS_Agent 3'	151 (av. = 5.033)	34 (av. = 1.133)	3.617
'DumbBot 4'	136 (av. = 4.533)	30 (av. = 1.0)	4.117
'DumbBot 5'	135 (av. = 4.5)	34 (av. = 1.133)	4.35
'RandomNegotiator 6'	63 (av. = 2.1)	30 (av. = 1.0)	6.55

Figure 42: 3 MCTS_Agent V5 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

However, notice that when we have 5 MCTS_AgentsV5 in the tournament, they clearly outperform their opponents, including D-Brane (Figure 43).

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	164 (av. = 5.467)	35 (av. = 1.167)	3.417
'MCTS_Agent 2'	159 (av. = 5.3)	35 (av. = 1.167)	3.517
'MCTS_Agent 3'	153 (av. = 5.1)	35 (av. = 1.167)	3.55
'MCTS_Agent 4'	153 (av. = 5.1)	35 (av. = 1.167)	3.917
'D-Brane 0'	151 (av. = 5.033)	35 (av. = 1.167)	3.867
'MCTS_Agent 5'	146 (av. = 4.867)	35 (av. = 1.167)	4.067
'DumbBot 6'	90 (av. = 3.0)	27 (av. = 0.9)	5.667

Figure 43: 5 MCTS_Agent V5 vs 1 D-Brane and 1 DumbBot. Average over 30 games.

4.1.3.6. Version 6:

In the following table we observe the odd result that in spite of the fact that MCTS_AgentV6 occupies more Supply Center than D-Brane, D-brane achieved one Solo Victory when MCTS_AgentV6 achieved none.

	Supply Centers	Points	Solo Victories	Average Rank
'D-Brane 4'	228 (av. = 7.6)	53 (av. = 1.767)	1 (av. = 0.0167)	2.2
'MCTS_Agent'	251 (av. = 8.367)	41 (av. = 1.367)	0 (av. = 0.0)	1.85
'DumbBot 2'	151 (av. = 5.033)	41 (av. = 1.367)	0 (av. = 0.0)	3.4
'DumbBot 1'	133 (av. = 4.433)	41 (av. = 1.367)	0 (av. = 0.0)	3.817
'RandomNegotiator 3'	68 (av. = 2.267)	29 (av. = 0.967)	0 (av. = 0.0)	5.433
'RandomNegotiator 5'	64 (av. = 2.133)	33 (av. = 1.1)	0 (av. = 0.0)	5.517
'RandomNegotiator 6'	61 (av. = 2.033)	33 (av. = 1.1)	0 (av. = 0.0)	5.783

Figure 44: MCTS_Agent V6 vs 1 D-Brane 2 DumbBots and 3 RandomBots. Average over 30 games.

As it was expected, increasing the number of D-Brane opponents weakens MCTS_Agent performance.

	Supply Centers	Points	Average Rank
'D-Brane 2'	190 (av. = 6.333)	41 (av. = 1.367)	2.65
'D-Brane 1'	178 (av. = 5.933)	41 (av. = 1.367)	3.067
'MCTS_Agent'	174 (av. = 5.8)	41 (av. = 1.367)	3.167
'D-Brane 3'	169 (av. = 5.633)	41 (av. = 1.367)	3.233
'DumbBot 2'	116 (av. = 3.867)	37 (av. = 1.233)	4.633
'DumbBot 1'	110 (av. = 3.667)	35 (av. = 1.167)	4.75
'RandomNegotiator 6'	52 (av. = 1.733)	27 (av. = 0.9)	6.5

Figure 45: MCTS_Agent V6 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

This table shows an interesting result because playing against 3 RandomBots instead of just 1, makes the MCTS_AgentV6 improve its performance.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	451 (av. = 7.517)	94 (av. = 1.567)	2.417
'D-Brane 2'	410 (av. = 6.833)	94 (av. = 1.567)	2.634
'D-Brane 1'	410 (av. = 6.833)	94 (av. = 1.567)	2.659
'D-Brane 3'	412 (av. = 6.867)	94 (av. = 1.567)	2.567
'RandomNegotiator 4'	96 (av. = 1.6)	59 (av. = 0.98)	6.025
'RandomNegotiator 5'	114 (av. = 1.9)	63 (av. = 1.05)	5.85
'RandomNegotiator 6'	122 (av. = 2.033)	75 (av. = 1.25)	5.9

Figure 46: MCTS_Agent V6 vs 3 D-Brane and 3 RandomBots. Average over 30 games.

As it was expected, playing against 5 D-Brane opponents weakens MCTS_Agent performance.

	Supply Centers	Points	Average Rank
'D-Brane 2'	181 (av. = 6.033)	35 (av. = 1.167)	2.933
'D-Brane 1'	162 (av. = 5.4)	35 (av. = 1.167)	3.417
'D-Brane 3'	157 (av. = 5.233)	35 (av. = 1.167)	3.45
'D-Brane 4'	155 (av. = 5.167)	35 (av. = 1.167)	3.633
'D-Brane 5'	149 (av. = 4.967)	35 (av. = 1.167)	4.0
'MCTS_Agent'	142 (av. = 4.733)	35 (av. = 1.167)	4.05
'DumbBot 6'	71 (av. = 2.367)	25 (av. = 0.833)	6.517

Figure 47: MCTS_Agent V6 vs 5 D-Brane and 1 DumbBot. Average over 30 games.

In this table, the results do not let us decide if D-Brane clearly outperforms MCTS_AgentV6.

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	176 (av. = 5.867)	32 (av. = 1.067)	3.215
'D-Brane 0'	168 (av. = 5.6)	32 (av. = 1.067)	3.067
'MCTS_Agent 2'	161 (av. = 5.367)	32 (av. = 1.067)	3.433
'MCTS_Agent 3'	157 (av. = 5.233)	32 (av. = 1.067)	3.25
'DumbBot 4'	137 (av. = 4.567)	32 (av. = 1.067)	4.1
'DumbBot 5'	128 (av. = 4.267)	30 (av. = 1.0)	4.483
'RandomNegotiator 6'	69 (av. = 2.3)	30 (av. = 1.0)	6.517

Figure 48: 3 MCTS_Agent V6 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

As in the previous table, we see that MCTS_AgentV6 has a comparable performance to that of D-Brane.

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	161 (av. = 5.367)	32 (av. = 1.067)	3.55
'MCTS_Agent 2'	158 (av. = 5.267)	32 (av. = 1.067)	3.467
'MCTS_Agent 3'	149 (av. = 4.967)	32 (av. = 1.067)	3.683
'D-Brane 0'	148 (av. = 4.933)	32 (av. = 1.067)	3.767
'MCTS_Agent 4'	147 (av. = 4.9)	32 (av. = 1.067)	3.85
'MCTS_Agent 5'	146 (av. = 4.867)	32 (av. = 1.067)	4.45
'DumbBot 6'	108 (av. = 3.6)	28 (av. = 0.933)	5.233

Figure 49: 5 MCTS_Agent V6 vs 1 D-Brane and 1 DumbBot. Average over 30 games.

4.1.3.7. Version 7:

In this table we see again a competitive performance to D-Brane. We observe no significant differences to the performance of Versions 5 and 6 of MCTS_Agent.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	229 (av. = 7.633)	39 (av. = 1.3)	2.067
'D-Brane 4'	205 (av. = 6.833)	39 (av. = 1.3)	2.45
'DumbBot 2'	170 (av. = 5.667)	39 (av. = 1.3)	2.917
'DumbBot 1'	148 (av. = 4.933)	39 (av. = 1.3)	3.617
'RandomNegotiator 3'	81 (av. = 2.7)	37 (av. = 1.233)	5.4
'RandomNegotiator 5'	68 (av. = 2.267)	33 (av. = 1.1)	5.65
'RandomNegotiator 6'	61 (av. = 2.033)	29 (av. = 0.967)	5.9

Figure 50: MCTS_Agent V7 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.

This table shows a quite surprising result. This is the only version of MCTS_Agent that outperforms D-Brane in a 1 vs 3 tournament.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	188 (av. = 6.267)	43 (av. = 1.433)	2.85
'D-Brane 2'	181 (av. = 6.033)	43 (av. = 1.433)	3.167
'D-Brane 1'	178 (av. = 5.933)	43 (av. = 1.433)	3.067
'D-Brane 3'	176 (av. = 5.867)	43 (av. = 1.433)	3.0
'DumbBot 2'	116 (av. = 3.867)	39 (av. = 1.3)	4.55
'DumbBot 1'	101 (av. = 3.367)	35 (av. = 1.167)	4.933
'RandomNegotiator 6'	50 (av. = 1.667)	27 (av. = 0.9)	6.433

Figure 51: MCTS_Agent V7 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

As in Figure 46, we see that many weak opponents (RandomBots in our case) improves our performance.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	222 (av. = 7.4)	43 (av. = 1.433)	2.483
'D-Brane 2'	218 (av. = 7.267)	43 (av. = 1.433)	2.533
'D-Brane 1'	201 (av. = 6.7)	43 (av. = 1.433)	3.033
'D-Brane 3'	197 (av. = 6.567)	43 (av. = 1.433)	2.817
'RandomNegotiator 4'	60 (av. = 2.0)	35 (av. = 1.167)	5.617
'RandomNegotiator 5'	58 (av. = 1.933)	23 (av. = 0.767)	5.85
'RandomNegotiator 6'	52 (av. = 1.733)	31 (av. = 1.033)	5.9

Figure 52: MCTS_Agent V7 vs 3 D-Brane. Average over 30 games.

As it was expected, playing against 5 D-Brane opponents weakens MCTS_Agent performance.

	Supply Centers	Points	Average Rank
'D-Brane 2'	163 (av. = 5.433)	32 (av. = 1.067)	3.4
'D-Brane 1'	163 (av. = 5.433)	32 (av. = 1.067)	3.433
'D-Brane 3'	152 (av. = 5.067)	32 (av. = 1.067)	3.633
'D-Brane 4'	151 (av. = 5.033)	32 (av. = 1.067)	4.0
'D-Brane 5'	146 (av. = 4.867)	32 (av. = 1.067)	3.737
'MCTS_Agent'	132 (av. = 4.4)	30 (av. = 1.0)	4.283
DumbBot 6'	108 (av. = 3.6)	30 (av. = 1.0)	5.483

Figure 53: MCTS_Agent V7 vs 5 D-Brane and 1 DumbBot. Average over 30 games.

Again, we see a performance comparable to that of D-Brane, although not able to outperform it.

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	184 (av. = 6.133)	35 (av. = 1.167)	2.667
'D-Brane 0'	177 (av. = 5.9)	35 (av. = 1.167)	2.85
'MCTS_Agent 2'	169 (av. = 5.633)	35 (av. = 1.167)	3.233
'MCTS_Agent 3'	160 (av. = 5.333)	35 (av. = 1.167)	3.5
'DumbBot 4'	117 (av. = 3.9)	35 (av. = 1.167)	4.517
'DumbBot 5'	111 (av. = 3.7)	33 (av. = 1.1)	4.9
'RandomNegotiator 6'	62 (av. = 2.067)	32 (av. = 0.833)	6.333

Figure 54: 3 MCTS_Agent V7 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

Version 7 of MCTS_Agent has the worst performance compared to the other versions of the MCTS_Agent when playing 5 vs 1 tournaments.

	Supply Centers	Points	Average Rank
'D-Brane 0'	163 (av. = 5.433)	34 (av. = 1.133)	3.083
'MCTS_Agent 1'	160 (av. = 5.333)	34 (av. = 1.133)	3.483
'MCTS_Agent 2'	155 (av. = 5.167)	34 (av. = 1.133)	3.817
'MCTS_Agent 3'	151 (av. = 5.033)	34 (av. = 1.133)	3.65
'MCTS_Agent 4'	149 (av. = 4.967)	34 (av. = 1.133)	4.083
'MCTS_Agent 5'	142 (av. = 4.733)	34 (av. = 1.133)	4.1
'DumbBot 6'	93 (av. = 3.1)	26 (av. = 0.867)	5.783

Figure 55: 5 MCTS_Agent V7 vs 1 D-Brane and 1 DumbBot. Average over 30 games.

4.1.3.8. Version 8:

The results from the table below show that Version 8 of the MCTS_Agent exhibits a better performance than the previous versions, when playing against 1 D-Brane, in terms of Supply Centers won.

	Supply Centers	Points	Average Rank
'MCTS_Agent'	235 (av. = 7.833)	40 (av. = 1.333)	2.0
'D-Brane 4'	202 (av. = 6.733)	40 (av. = 1.333)	2.35
'DumbBot 2'	144 (av. = 4.8)	40 (av. = 1.333)	3.7
'DumbBot 1'	139 (av. = 4.633)	38 (av. = 1.267)	3.65
'RandomNegotiator 3'	81 (av. = 2.7)	36 (av. = 1.2)	5.317
'RandomNegotiator 5'	75 (av. = 2.5)	38 (av. = 1.267)	5.467
'RandomNegotiator 6'	67 (av. = 2.233)	28 (av. = 0.933)	5.517

Figure 56: MCTS_Agent V8 vs 1 D-Brane, 2 DumbBots and 3 RandomBots. Average over 30 games.

The results from the table in Figure 57 below show that Version 8 of the MCTS_Agent has an inferior performance than that of the previous versions of the MCTS_Agent when playing against 3 D-Branes.

	Supply Centers	Points	Average Rank
'D-Brane 2'	194 (av. = 6.467)	41 (av. = 1.367)	2.633
'D-Brane 1'	184 (av. = 6.133)	41 (av. = 1.367)	2.8
'D-Brane 3'	176 (av. = 5.867)	41 (av. = 1.367)	2.95
'MCTS_Agent'	157 (av. = 5.233)	39 (av. = 1.3)	3.617
'DumbBot 2'	114 (av. = 3.8)	33 (av. = 1.1)	4.617
'DumbBot 1'	107 (av. = 3.567)	39 (av. = 1.3)	5.167
'RandomNegotiator 6'	63 (av. = 2.1)	29 (av. = 0.967)	6.217

Figure 57: MCTS_Agent V8 vs 3 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

As in the previous versions of the MCTS_Agent, playing against 5 D-Brane opponents weakens MCTS_Agent performance.

	Supply Centers	Points	Average Rank
'D-Brane 2'	174 (av. = 5.8)	37 (av. = 1.233)	3.133
'D-Brane 1'	167 (av. = 5.567)	37 (av. = 1.233)	3.45
'D-Brane 3'	162 (av. = 5.4)	37 (av. = 1.233)	3.417
'D-Brane 4'	152 (av. = 5.067)	37 (av. = 1.233)	3.733
'D-Brane 5'	152 (av. = 5.067)	37 (av. = 1.233)	3.983
'MCTS_Agent'	138 (av. = 4.6)	37 (av. = 1.233)	4.133
DumbBot 6'	67 (av. = 2.233)	23 (av. = 0.767)	6.15

Figure 58: MCTS_Agent V8 vs 5 D-Brane and 1 DumbBot. Average over 30 games.

This table shows that the performance of Version 8 of the MCTS_Agent is comparable to that of D-Brane.

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	185 (av. = 6.167)	36 (av. = 1.2)	2.617
'MCTS_Agent 2'	170 (av. = 5.667)	36 (av. = 1.2)	3.017
'D-Brane 0'	168 (av. = 5.6)	36 (av. = 1.2)	3.133
'MCTS_Agent 3'	163 (av. = 5.433)	36 (av. = 1.2)	3.35
'DumbBot 4'	122 (av. = 4.067)	34 (av. = 1.133)	4.5
'DumbBot 5'	120 (av. = 4.0)	34 (av. = 1.133)	4.75
'RandomNegotiator 6'	54 (av. = 1.8)	26 (av. = 0.867)	6.633

Figure 59: 3 MCTS_Agent V8 vs 1 D-Brane, 2 DumbBots and 1 RandomBot. Average over 30 games.

Again, this table shows that the performance of Version 8 of the MCTS_Agent is comparable to that of D-Brane. However, it is confirmed that playing against many competitive agents (in this case many MCTS_Agents) weakens the MCTS_Agent's performance.

	Supply Centers	Points	Average Rank
'MCTS_Agent 1'	174 (av. = 5.8)	35 (av. = 1.167)	3.083
'MCTS_Agent 2'	159 (av. = 5.3)	35 (av. = 1.167)	3.467
'D-Brane 0'	153 (av. = 5.1)	35 (av. = 1.167)	3.817
'MCTS_Agent 3'	149 (av. = 4.933)	35 (av. = 1.167)	3.65
'MCTS_Agent 4'	140 (av. = 4.667)	35 (av. = 1.167)	4.083
'MCTS_Agent 5'	135 (av. = 4.5)	35 (av. = 1.167)	4.1
'DumbBot 6'	99 (av. = 3.3)	27 (av. = 0.9)	5.824

Figure 60: 5 MCTS_Agent V8 vs 1 D-Brane and 1 DumbBot. Average over 30 games.

With these experiments we complete a systematic set of experiments that will help us draw some conclusions regarding the performance of the versions 5, 6, 7 and 8 of MCTS_Agent. Each MCTS_Agent has played against 1, 3 and 5 D-Brane agents and D-Brane has played against 1, 3 and 5 MCTS_Agents of each version.

4.2. Comparative Results

The figures of this section will help us compare the most promising of our agents (versions 5, 6, 7 and 8) with each other and with D-Brane. Figures 61 and 62 show how our agents perform against one (1), three (3) and five (5) D-Brane agents. Figures 63 and 64 show how D-Brane agent performs against one (1), three (3) and five (5) of our agents.

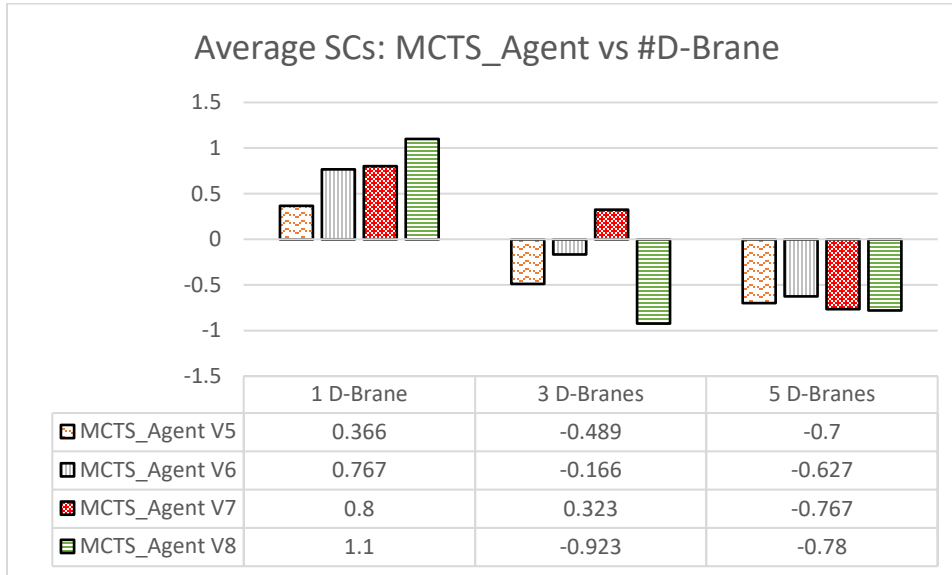


Figure 61: One MCTS_Agent vs one, three or five D-Brane agents. Subtraction of the average SCs owned by the agents at every tournament. Average over 30 games.

Each agent has played three 30-game tournaments against one, three and five D-Branes respectively. We chose this method of evaluation because it is important for a seven-player game to consider the quality of the nearby opponents. There is a greater need for better strategy in order to expand when having tough neighbors than when having RandomBots or DumbBots.

The tournaments (the three columns of the graph) are between the following agents:

- 1st: 1 MCTS_Agent, 1 D-Brane, 2 DumbBots, 3 RandomBots.
- 2nd: 1 MCTS_Agent, 3 D-Brane, 2 DumbBots, 1 RandomBot.
- 3rd: 1 MCTS_Agent, 5 D-Brane, 1 DumbBot, 0 RandomBots.

Each value of the graph represents the subtraction:

$$SC_{MCTS} - SC_{DBrane} ,$$

where SC_{MCTS} is the average number of Supply Centers that MCTS_Agent owns at a game and

SC_{DBrane} is the average number of Supply Centers that D-Brane owns at a game (when there are more than one D-Brane agents we take the mean value).

We observe that all MCTS_Agents beat D-Brane at one vs one tournaments, but they lose at the one vs three and one vs five tournaments (with the exception of MCTS_AgentV7 vs three D-Branes).

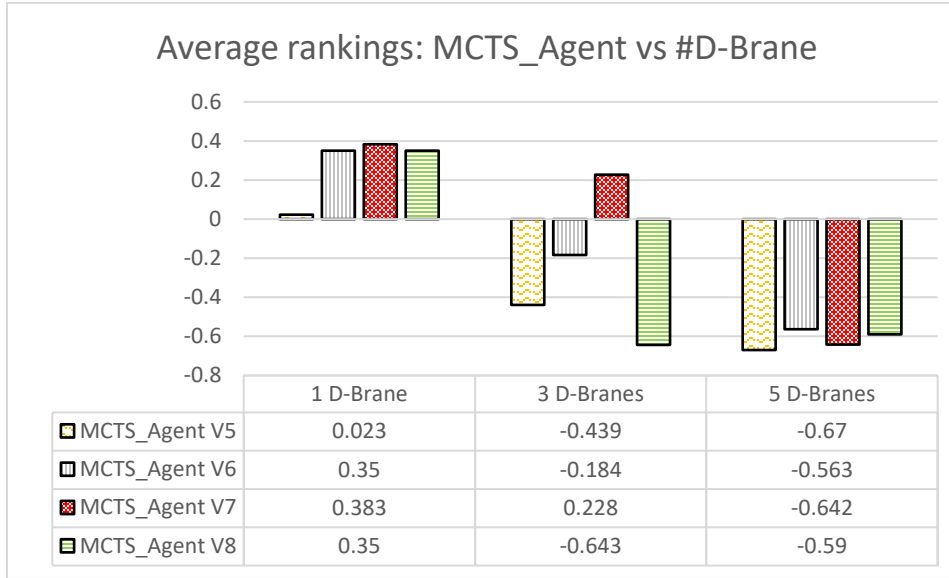


Figure 62: One MCTS_Agent vs one, three or five D-Brane agents. Subtraction of the average rankings achieved by the agents at every tournament. Average over 30 games.

In Figure 62 we present more comparative results among MCTS_Agents Version 5, 6, 7 and 8. Each MCTS_Agent has played three 30-game tournaments against one, three and five D-Branes respectively.

The tournaments (the three columns of the graph) are between the following agents:

- 1st: 1 MCTS_Agent, 1 D-Brane, 2 DumbBots, 3 RandomBots.
- 2nd: 1 MCTS_Agent, 3 D-Brane, 2 DumbBots, 1 RandomBot.
- 3rd: 1 MCTS_Agent, 5 D-Brane, 1 DumbBot, 0 RandomBots.

Each value of the graph represents the subtraction:

$$\text{Rank}_{\text{D-Brane}} - \text{Rank}_{\text{MCTS}}$$

where $\text{Rank}_{\text{D-Brane}}$ is the average ranking that D-Brane achieved at the tournament (when there are more than one D-Brane agents we take the average of their average value), and

$\text{Rank}_{\text{MCTS}}$ is the average ranking that MCTS_Agent achieved at the tournament

Note that $\text{Rank} \in [1, 7]$ and represents the average ranking that an agent achieved in a tournament. That means that the closest the Rank is to 1, the better it means for the agent's performance. So, if $\text{Rank}_{\text{D-Brane}} - \text{Rank}_{\text{MCTS}} > 0$, then MCTS_Agent has better Rank than D-Brane.

We observe that all MCTS_Agents beat D-Brane at one vs one tournaments, but they lose at the one vs three and one vs five tournaments (with the exception of MCTS_AgentV7 vs three D-Branes). MCTS_Agent

becomes less effective when it faces a lot of D-Brane agents. However, the difference never gets bigger than 0.7 positions (see MCTS_AgentV5 in Figure 62).

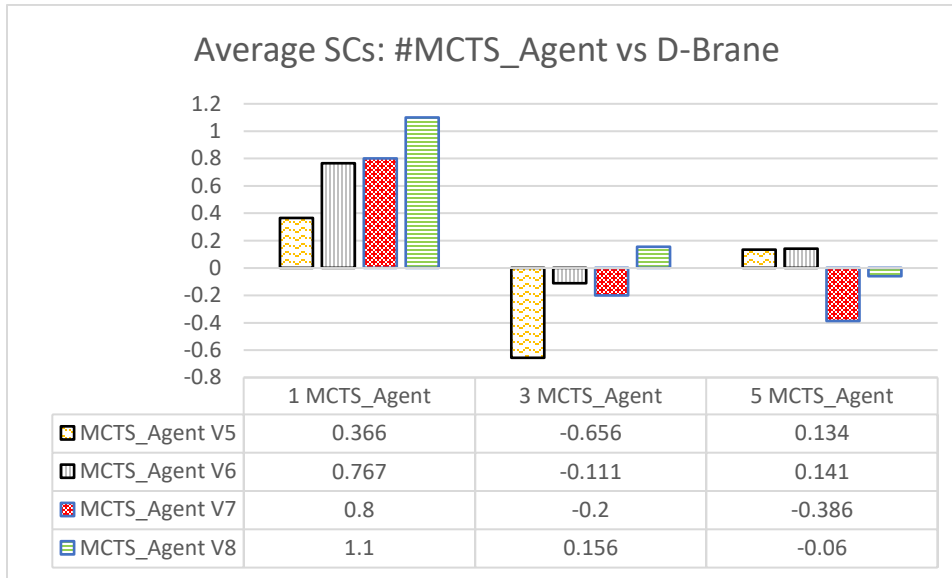


Figure 63: One D-Brane agent vs one, three or five MCTS_Agents. Subtraction of the average SCs owned by the agents at every tournament. Average over 30 games.

In Figure 63 we present more comparative results for MCTS_Agents Version 5, 6, 7 and 8. D-Brane agent has played a 30-game tournament against one, three and five of each version of MCTS_Agent.

The tournaments (the three columns of the graph) are between the following agents:

- 1st: 1 MCTS_Agent, 1 D-Brane, 2 DumbBots, 3 RandomBots.
- 2nd: 3 MCTS_Agent, 1 D-Brane, 2 DumbBots, 1 RandomBot.
- 3rd: 5 MCTS_Agent, 1 D-Brane, 1 DumbBot, 0 RandomBots.

Each value of the graph represents the subtraction:

$$SC_{\text{MCTS}} - SC_{\text{DBrane}},$$

where SC_{MCTS} is the average number of Supply Centers that MCTS_Agent owns at a game (when there are more than one MCTS_Agents we take the mean value) and

SC_{DBrane} is the average number of Supply Centers that D-Brane owns at a game.

First, we observe that all MCTS_Agents beat D-Brane at one vs one tournaments. At 3 MCTS_Agent vs 1 D-Brane tournaments we observe that MCTS_AgentV8 slightly outperforms D-Brane. Also, at 5 MCTS_Agent vs 1 D-Brane tournaments we observe that versions 5 and 6 of the MCTS_Agent slightly outplay D-Brane. Still we do not see significant differences from Figure 61 with respect to the agents' relative performance.

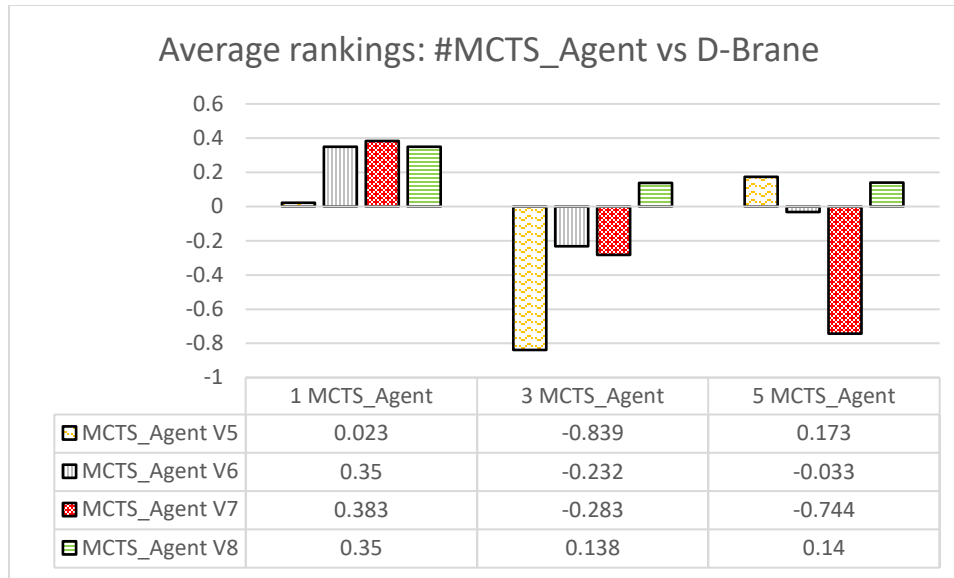


Figure 64: One D-Brane agent vs one, three or five MCTS_Agents. Subtraction of the average rankings achieved by the agents at every tournament. Average over 30 games.

In Figure 64 we present more comparative results for MCTS_Agents Version 5, 6, 7 and 8, when facing one D-Brane agent. That is, one D-Brane agent has played a 30-game tournament against one, three and five instances of each version of MCTS_Agent.

The tournaments (the three columns of the graph) are between the following agents:

- 1st: 1 MCTS_Agent, 1 D-Brane, 2 DumbBots, 3 RandomBots.
- 2nd: 3 MCTS_Agent, 1 D-Brane, 2 DumbBots, 1 RandomBot.
- 3rd: 5 MCTS_Agent, 1 D-Brane, 1 DumbBot, 0 RandomBots.

Each value of the graph represents the subtraction:

$$\text{Rank}_{\text{D-Brane}} - \text{Rank}_{\text{MCTS}}$$

where $\text{Rank}_{\text{D-Brane}}$ is the average ranking that D-Brane achieved at the tournament and

$\text{Rank}_{\text{MCTS}}$ is the average ranking that MCTS_Agent achieved at the tournament (when there are more than one MCTS_Agents we take the mean value).

We observe that all MCTS_Agents beat D-Brane at one vs one tournaments. At 3 MCTS_Agent vs 1 D-Brane tournaments we observe that MCTS_AgentV8 slightly outperforms D-Brane. Also, at 5 MCTS_Agent vs 1 D-Brane tournaments we observe that versions 5 and 8 of the MCTS_Agent slightly outperforms D-Brane.

4.3. Increasing the year limit and the decision-making time

In order to be certain about our conclusions, we ran some experiments with a different year limit of 10 years (20 decision-making rounds) instead of 5. First of all, we choose MCTS_AgentV8 as a quite promising agent. In the next table (Figure 65) we see the results when facing one D-Brane and many RandomBots. Then, in Figure 66 we see the same setting but we give more time to our agent for decision-making.

At Figure 65 we see that our MCTS_Agent clearly outperforms D-Brane in all metrics; interestingly, achieving many Solo Victories. That confirms that, at least when the rest of the opponents are weak (for example, RandomBots), the MCTS_Agent has a better rate of expansion than D-Brane.

	Supply Centers	Points	Solo Victories	Average Rank
'MCTS_Agent'	300 (av. = 15.0)	129 (av. = 6.45)	10 (av. = 0.5)	1.375
'D-Brane 4'	226 (av. = 11.3)	69 (av. = 3.45)	5 (av. = 0.25)	1.925
"RandomNegotiator 2"	40 (av. = 2.0)	7 (av. = 0.35)	0 (av. = 0.0)	4.4
"RandomNegotiator 1"	38 (av. = 1.9)	5 (av. = 0.25)	0 (av. = 0.0)	4.7
"RandomNegotiator 3"	30 (av. = 1.5)	5 (av. = 0.25)	0 (av. = 0.0)	5.025
'RandomNegotiator 5'	24 (av. = 1.2)	7 (av. = 0.35)	0 (av. = 0.0)	5.175
'RandomNegotiator 6'	17 (av. = 0.85)	7 (av. = 0.35)	0 (av. = 0.0)	5.4

Figure 65: MCTS_AgentV8 vs D-Brane. Greater year limit (10 years). Average of 20 games.

At Figure 66 we see that MCTS_Agent outplays D-Brane again. However, increasing the decision-making time of MCTS_Agent from 8 seconds to 12 seconds does not appear to have helped.

	Supply Centers	Points	Solo Victories	Average Rank
'MCTS_Agent'	286 (av. = 14.3)	107 (av. = 5.35)	7 (av. = 0.35)	1.65
'D-Brane 4'	251 (av. = 12.55)	59 (av. = 2.95)	3 (av. = 0.15)	1.55
"RandomNegotiator 2"	34 (av. = 1.7)	15 (av. = 0.75)	0 (av. = 0.0)	4.6
"RandomNegotiator 1"	34 (av. = 1.7)	12 (av. = 0.6)	0 (av. = 0.0)	4.775
"RandomNegotiator 3"	34 (av. = 1.7)	11 (av. = 0.55)	0 (av. = 0.0)	4.6
'RandomNegotiator 5'	22 (av. = 1.1)	12 (av. = 0.6)	0 (av. = 0.0)	5.375
'RandomNegotiator 6'	17 (av. = 0.85)	11 (av. = 0.55)	0 (av. = 0.0)	5.45

Figure 66: MCTS_AgentV8 vs D-Brane. Greater year limit (10 years). Greater decision-making time limit (12 seconds). Average of 20 games.

4.4. Discussion of Results

Our experiments show that MCTS_Agent is a competitive and comparable agent to D-Brane. In certain settings, it can outplay D-Brane (for example at one vs one tournaments) and at the settings that it is outplayed, still has a competitive performance.

The first challenge we had to face while implementing MCTS in this domain, is the urgent need of heuristics. The search space is huge and it is impossible for the agent to simulate enough to obtain reasonable rewards. Even if we had unlimited time to make a decision, it would be computationally impossible for the agent even to count all the possible actions that the algorithm could make from a state that our Power has the control of many units (e.g. more than ten). That is the reason why we got much better results when we used the weight system heuristic (versions 5, 6, 7 and 8) to sort the action lists, that is to indicate to the agent which moves/actions should be expanded and searched first.

For every state, the agent estimates the orders that the enemy units will make. For this purpose, we use a heuristic (see Section 3.1). It is not efficient to suppose that the enemies are using a competitive algorithm (e.g. D-Brane) because running that algorithm for six times (that is the amount of the enemies) would cost an important amount of time. So, we chose a rather greedy heuristic in order to make this estimation. This saves us some time that is used for more iterations of the algorithm but makes the agent have a poor defensive behavior. It never actually recognizes a threat except maybe when the threat is too close.

MCTS_Agent outperformed D-Brane when playing one vs one tournaments (see Figures 61 and 62) and was outperformed in the other settings. It is obvious (and we see it at the experiments as well) that we have a greater rate of expansion (i.e. rate of occupying new Supply Centers) when facing RandomBots than when facing D-Branes. This rate of expansion deteriorates somewhat when facing DumbBots (see Figure 30).

The fact that MCTS outperforms D-Brane is confirmed by the experiment that we set the year limit to ten years (see Figure 65 and Figure 66). In these experiments MCTS_Agent achieved more Solo Victories and clearly outperformed D-Brane.

The other results that MCTS_Agent outplays D-Brane, come under the following agents setting: 1 MCTS_Agent, 1 D-Brane, 2 DumbBots, 3 RandomBots (see Figure 61). That means that in the worst case, MCTS_Agent would have just one stronger opponent that neighbors it on the map. That is exactly what gives the opportunity of a strong start and, in fact, a stronger start than D-Brane's. As is of course to be expected, when facing five very strong opponents (D-Brane agents), our agent is unable to achieve as successful a performance. We report additional such experiments (with year limit increased to ten years) in Appendix B.

It is important to highlight at this point, that the 5 years limit is too strict to allow Solo Victories to be achieved. Although there are some exceptions when MCTS_Agent faces a lot of weak opponents (see Figures 16, 17, 18 and 19), that seems to be the case. At the experiments that we set the year limit to 10 years (see Figures 65 and 66), MCTS_Agent achieves Solo Victories even against a D-Brane agent. However, the average Supply Centers occupied by the agents is the decisive factor for their evaluation.

Another issue that we paid attention to is the proportion between the time spent for expansion and the time spent for simulation during the MCTS iterations. A simulation is run until a leaf node is reached. A leaf node represents a state that our agent either has achieved a solo victory (i.e. occupied 18 Supply Centers), or has lost (i.e. left with 0 Supply Centers), or the game has reached the default final year. The last condition is the most common one because the year limit is set to five years - with the exception of the last experiments (see Section 4.3). Therefore, while the game proceeds, the proportion mentioned before changes in favor of the time spent for expansion. That happens because there are fewer and fewer rounds to simulate. By implementing version 6, we tried to experiment with that trade off but, unfortunately, with no significant effects. Still, we know that using an alternative default final year, or using different heuristics would demand a modification at the expansion versus simulation trade off. We experimented a bit on that issue (see Figure 66 and Appendix B) but with no significant results.

The first four versions of the agent (version 1, 2, 3 and 4) are not competitive. They lose both from D-Brane and from DumbBot. The MCTS algorithm does not seem to find any clever move to make, and the agent manages to occupy just the few Supply Centers that are located at the adjacent Provinces of its starting position. We believe that the results are mainly affected by the pruning method used.

Implementing a weight system method for sorting each node's action list, and thus effectively injecting more domain knowledge to the agent, allows MCTS to produce much better results, because the weighting indicates which actions should be expanded first, which actions seem more promising than the others. In our perception, this is the main reason why MCTS_Agent's versions 5, 6, 7 and 8 outperform DumbBot.

While comparing the MCTS agents, we notice that version 8 has a slight advantage against the others. However, this advantage does not exceed a value that could let us conclude that it is surely better than the others. Including the fact that it has come last when played against three D-Branes (see Figure 61) makes us believe that it is an occasional advantage.

Perhaps the safest conclusion thing that can be drawn on the relative ranking of the methods issue, is that Versions 8 and 7 appear to be the most successful of our agents, with versions 5 and 6 coming close.

5. Conclusions and Future Work

In this chapter we present some conclusions regarding the implementation of Monte Carlo Tree Search algorithm using The Upper Confidence Bounds for Trees (UCT) as its tree policy.

Our goal was to implement a competitive non-negotiating agent that uses MCTS algorithm at its strategic module in the multi-agent strategic game of Diplomacy. For this task we created eight (8) versions of our agent with differences at the MCTS algorithm or the implemented heuristics.

The Monte Carlo Tree Search family of algorithms contains a huge variety of methods and policies. By creating a competitive agent, we showed that MCTS can be used to obtain competitive results in the Diplomacy domain. Moreover, it can be shown from our work that the (limited) injection of domain knowledge, in the form of certain heuristics to guide MCTS search, may cause a vast improvement of the overall performance of MCTS. That, in combination with the variety of approaches of MCTS create a promising ground for further research.

In our experiments we compare the performance of our agents against the to date most successful agent of the ANAC Diplomacy competition, namely the so called “D-Brane” agent. Our results show that MCTS_Agent outperforms D-Brane in certain settings (see specifically in one vs one settings as reported in Chapter 4), and in most other cases has a competitive performance, comparable to that of D-Brane. By implementing the sorting of the actions lists technique at versions 5, 6, 7 and 8 of the MCTS_Agent, we show that importing high quality domain knowledge (especially in such a complicated domain) improves the performance of MCTS and promises even better results when combined with a variety of methods of the MCTS family. Our thesis resulted to the creation of a basic framework for further research on MCTS for a Diplomacy playing agent.

Regarding future work, altering the year limit of the games would cause a different performance of the agent because it would alternate the simulation depth. Making the agent adaptive to different year limits is still a challenging but promising goal. Also, using different MCTS methods such as Bayesian UCT or AMAF (see Appendix A) might cause an improvement in updating of the agent’s preferences as it is implied at other domains (Tesauro, Rajan and Segal 2012), (Helmbold and Parker-Wood 2009). Another issue that needs further optimization is the time limit that we let the MCTS algorithm to iterate.

6. Appendix A: Monte Carlo Tree Search variants

There have been a lot of variations, extensions and enhancements of Monte Carlo Tree Search algorithm used at a wide range of domains. Here are some of them that could be used for Diplomacy strategy game.

Monte Carlo Tree Search with a variety of simulation depths

The MCTS algorithms that we have implemented by now, have five years of game play (that means ten rounds) as a default simulation limit. Creating variations of this limit would cause different behaviors of the algorithm because it would change drastically the proportion between the time used for expansion of the tree and the time used for simulation of the game. Furthermore, lengthening the year limit would let the agents compete in a more long-term scale and achieve solo victories. That could give us a clearer view of their abilities in order to compare and rank them.

Bayesian Upper Confidence Bounds for Trees (BUCT)

BUCT is an approach of the UCT policy based on Bayesian inference. These are some “comprising mechanisms for computing leaf node posterior distributions, propagating inference up the tree to compute interior node distributions, and then computing distribution-based upper confidence estimates as a basis for choosing where to sample next.” (Tesauro, Rajan and Segal 2012)

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

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

where C_j is the Cumulative Distribution Function (CDF) of P_j .

Tesauro et al. proposes two modified versions of UCB1 to descend the tree and choose where to sample next. (Tesauro, Rajan and Segal 2012)

BUCT 1: maximize $B_i = \mu_i + \frac{\sqrt{2 \ln N}}{n_i}$, where

μ_i : Mean of P_i

N : Total trials of all arms

n_i : Number of trials for each arm

P_i : Probability distribution over its true expected reward value

BUCT 2: maximize $B_i = \mu_i + \sqrt{2 \ln N} \sigma_i$, where

μ_i : Mean of P_i

N : Total trials of all arms σ_i

σ_i : Square root of variance of P_i

P_i : Probability distribution over its true expected reward value

All Moves As First (AMAF)

The AMAF algorithm treats all moves played during selection and simulation as if they were played on a previous selection step. This means that the reward estimate for an action a from a state s is updated whenever a is encountered during a playout, even if a was not the actual move chosen from s .

The basic AMAF algorithm combines UCT with the AMAF update after each play-out. This algorithm rapidly grows the counts at the nodes in the UCT tree, and thus increases the algorithm's confidence in the win rates. On the other hand, the counts at nodes are increased not because the move was made in the position represented by the parent, but because it was made in a (perhaps very) different context. (Helmbold and Parker-Wood 2009)

7. Appendix B: Extra Experiments

As is of course to be expected, when facing 5 very strong opponents (D-Brane agents), our agent is unable to achieve as successful a performance.

	Supply Centers	Points	Solo Victories	Average Rank
'D-Brane 4'	124 (av. = 6.2)	44 (av. = 2.2)	1 (av. = 0.05)	3.65
'D-Brane 2'	126 (av. = 6.3)	34 (av. = 1.7)	0 (av. = 0.0)	2.975
'D-Brane 1'	116 (av. = 5.8)	32 (av. = 1.6)	0 (av. = 0.0)	3.675
'D-Brane 3'	113 (av. = 5.65)	34 (av. = 1.7)	0 (av. = 0.0)	3.35
D-Brane 5'	112 (av. = 5.6)	34 (av. = 1.7)	0 (av. = 0.0)	3.4
'MCTS_Agent'	83 (av. = 4.15)	28 (av. = 1.4)	0 (av. = 0.0)	4.025
'RandomNegotiator 6'	6 (av. = 0.3)	4 (av. = 0.2)	0 (av. = 0.0)	6.925

Figure 67: MCTS_AgentV8 vs 5 D-Branes and 1 RandomBot. Greater year limit (10 years). Average of 20 games.

At this table we see that increasing the decision-making time for MCTS algorithm did not improve the performance at all. The results are similar to the previous table.

	Supply Centers	Points	Solo Victories	Average Rank
'D-Brane 4'	143 (av. = 7.15)	34 (av. = 1.7)	0 (av. = 0.00)	2.875
'D-Brane 2'	122 (av. = 6.1)	34 (av. = 1.7)	0 (av. = 0.0)	3.425
'D-Brane 1'	115 (av. = 5.75)	31 (av. = 1.55)	0 (av. = 0.0)	2.975
'D-Brane 3'	110 (av. = 5.5)	34 (av. = 1.7)	0 (av. = 0.0)	3.7
D-Brane 5'	104 (av. = 5.2)	28 (av. = 1.4)	0 (av. = 0.0)	3.75
'MCTS_Agent'	70 (av. = 3.5)	21 (av. = 1.05)	0 (av. = 0.0)	4.525
'RandomNegotiator 6'	14 (av. = 0.7)	9 (av. = 0.45)	0 (av. = 0.0)	6.75

Figure 68: MCTS_AgentV8 vs 5 D-Branes and 1 RandomBot. Greater year limit (10 years). Increased time for MCTS decision-making (from 8 to 12 seconds). Average of 20 games.

References

- Browne, Cameron, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. "A Survey of Monte Carlo Tree Search Methods ."
- Calhamer, Allan B. 2000. *The Rules of Diplomacy*.
- Chaslot, Guillaume, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. "Monte-Carlo Tree Search: A New Framework for Game AI." *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*.
- de Jonge, Dave. 2018. "ANAC 2019 Diplomacy Challenge Manual and Rules."
- . 2019. "ANAC 2019 Diplomacy Challenge Manual and Rules."
- . 2017. "D-Brane: a Diplomacy Playing Agent for Automated Negotiations Research."
- . 2019. "The BANDANA Framework v1.3."
- Dechter, R, and R Mateescu. 2007. "And/or search spaces for graphical models." *Artificial Intelligence*. doi:<http://dx.doi.org/10.1016/j.artint.2006.11.003>.
- Fabregues, Angela. 2011. "DipGame: A challenging negotiation testbed."
- Gelly, Sylvain, and David Silver. 2006. "Monte-Carlo tree search and rapid action value estimation in Computer Go." *INRIA*.
- Helmhold, David. P., and Aleatha Parker-Wood. 2009. "All-Moves-As-First Heuristics in Monte-Carlo Go." *International Conference on Artificial Intelligence, ICAI*.
- Hoeffding, Wassily. 1963. "Probability inequalities for sums of bounded random variables." *Journal of the American Statistical Association* 58:13–30.
- Holcomb, Sean D, Shaun V Ault, William K Porter, Guifen Mao, and Wang Jin. 2018. "Overview on DeepMind and Its AlphaGo Zero AI ."
- Jonker, Catholijn M., Reyhan Aydogan, Tim Baarslag, Katsuhide Fujita, Takayuki Ito, and Koen Hindiks. 2018. "Automated Negotiating Agents Competition (ANAC)."
- Karamalegos, Emmanouil. 2016. *Monte Carlo Tree Search in the "Settlers of Catan" Strategy Game*. Engineering Diploma Thesis, School of ECE, Technical University of Crete.
- Kocsis, Levente, and Csaba Szepesvari. 2006. "Bandit based Monte-Carlo Planning."
- Matsubara, Hitoshi, and Leandro Soriano Marcolino. 2011. "Multi-Agent Monte Carlo Go." 2019. *Negotiation*. <https://ii.tudelft.nl/nego/node/7>.
- Norman, David. 2002. *DAIDE*. <http://www.daide.org.uk/>.
- . 2003. "<http://www.ellought.demon.co.uk/dipai>."

- Panousis, Konstantinos Panagiotis. 2014. *Real-time Planning and Learning in the "Settlers of Catan"*. Engineering Diploma Thesis, School of ECE, Technical University of Crete.
- Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, et al. 2017. "Mastering the game of Go without human knowledge." *Nature*.
2019. *Tenth International Automated Negotiating Agents Competition, ANAC 2019*.
<http://web.tuat.ac.jp/~katfuji/ANAC2019/>.
- Tesauro, Gerald, VT Rajan, and Richard Segal. 2012. "Bayesian Inference in Monte-Carlo Tree Search."