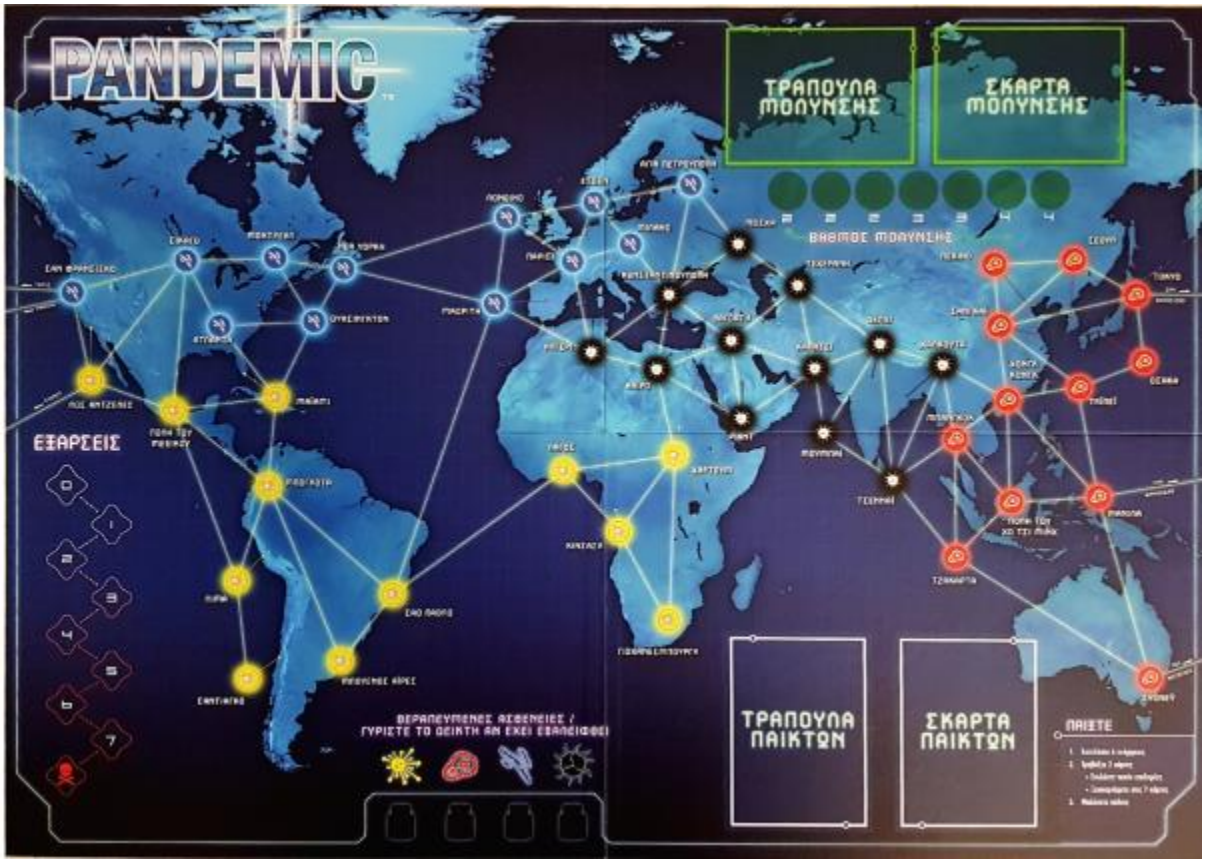
COMP 512: Multiagent Systems Course

Building an agent for the board game Pandemic



Φοιτητής:

Κωνσταντίνος Δανόπουλος 2016030131

Κωδικός ομάδας : LAB51246599

Αναφορά

Θεωρητικό Υπόβαθρο Αλγόριθμου

Στο συγκεκριμένο πρότζεκτ ζητήθηκε να υλοποιήσουμε έναν agent για το Pandemic Board Game. Για την δημιουργία του agent υλοποίησα τον αλγόριθμο Monte Carlo Tree Search (MCTS). Ως θεωρητική πηγή χρησιμοποίησα εξολοκλήρου το paper **Monte Carlo Tree Search for the Game of Diplomacy** του Αλέξιου Θεοδωρίδη και του Γεώργιου Χαλκιαδάκη που δημοσιεύθηκε στο 11ο Ελληνικό Συνέδριο του Artificial Intelligence. Όπως και στο paper εφάρμοσα και εγώ τον πιο διαδεδομένο από την οικογένεια των MCTS αλγόριθμο ο οποίος είναι ο *Upper Confidence Bound for Trees (UCT)*. Έτσι όπως και στον αλγόριθμο (UCT) επιλέγω να κάνω expand το καταλληλότερο child node με βάση την παρακάτω εξίσωση:

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

όπου X_j είναι η κανονικοποιημένη μέση ανταμοιβή $(0 \leq X_j \leq 1)$, ως C_p χρησιμοποιήθηκε η τιμή $\frac{1}{\sqrt{2}}$ η οποία είχε αποδειχθεί από τους Kocsis και Szepesvari ότι ικανοποιεί την Hoeffding ανισότητα με ανταμοιβές στο εύρος [0,1], n είναι ο αριθμός των φορών όπου ο parent node έχει επισκεφθεί και Visits είναι ο αριθμός των φορών που έχει επισκεφθεί ο τωρινός node.

Υπολογίζω το UTC για καθένα από τα child node και επιλέγω να κάνω expand το node που μεγιστοποιεί την παραπάνω εξίσωση. Ο αλγόριθμος αναζήτησης που δημιούργησα βασίστηκε στον παρακάτω ψευδοκώδικα που περιείχε το paper:

Algorithm 2: The UTC algorithm

```
1  Function UctSearch(s_0):
2      create root node v_0 with state s_0;
3      while within computational budget do
4          v_i ← TreePolicy (v_0);
5          Δ ←DefaultPolicy (s(v_0));
6          Backup(v_i; Δ);
7      return a(BestChild(v_0; 0));
8  Function TreePolicy(v):
9      while v is non-terminal do
10         if v not fully expanded then
11             return Expand(v);
12         else
13             v ←BestChild(v; C_p);
14         end
15     return v;
```

```
16 Function EXPAND(v):
17     choose α ∈ untried actions from A(s(v));
18     add a new child v′ to v
19         with s(v′) = f(s(v), α)
20         and a(v′) = α;
21     return v′;
22 Function BESTCHILD(v, c):
23     return arg max v′ children of v  Q(v)/N(v) + c√(2 ln N(v)/N(v));
24 Function DEFAULTPOLICY(s):
25     while s is non-terminal do
26         choose α ∈ A(s) uniformly at random;
27         s ← f(s, α);
28     return reward for state s;
29 Function BACKUP(v, Δ):
30     while v is not null do
31         N(v) ← N(u) + 1;
32         Q(v) ← Q(u) + Δ(v, p);
33         v ← parent of v
```

Υλοποιήθηκαν όλες οι συναρτήσεις του ψευδοκώδικα εκτός από την συνάρτηση treePolicy την οποία την έγραψα εσωτερικά μέσα στην συνάρτηση UTCsearch. Οι συναρτήσεις backup και bestChild υλοποιήθηκαν ακριβώς όπως και στον ψευδοκώδικα. Οι διαφορές είναι ότι η συνάρτηση expand που δημιούργησα δεν προσθέτει ένα child κάθε φορά στον parent node αλλά προσθέτει απευθείας όλα τα παιδιά. Εφόσον άλλαξε η expand θα έπρεπε να αλλάξει και η UTCsearch ώστε να διατηρεί την ίδια λειτουργία με τον ψευδοκώδικα. Η UTCsearch έχει την ίδια δομή με την πάνω με την διαφορά ότι όταν κληθεί η expand τότε βαθμολογεί όλα τα νέα childs από μια φορά και μετά συνεχίζει να κάνει treePolicy, defaultPolicy και backup.

Σκοπός της Άσκησης

Σύμφωνα με την εκφώνηση οφείλαμε να υλοποιήσουμε έναν agent ο οποίος θα πρέπει να κάνει τα εξής:

Α) κάθε φορά που παίζει κάποιος συμπαίχτης του, θα πρέπει να του δίνει μια συμβουλή για ποια κίνηση οφείλει να κάνει

Β) κάθε φορά που θα έρχεται η σειρά του να παίξει, θα πρέπει να λαμβάνει τις συμβουλές των συμπαιχτών του, να τις αξιολογεί και έπειτα να επιλέγει τελικά την κίνηση που αποφάσισε να εκτελέσει

Όπως θα συνέβαινε και στον πραγματικό κόσμο, εάν ένας άνθρωπος βρισκόταν στην θέση των agents, θα σκεφτόταν αρχικά μερικές πιθανές κινήσεις που θα μπορούσε να κάνει. Έπειτα θα επέλεγε την καλύτερη κίνηση από αυτές με βάση την δικιά του προσωπική αντίληψη. Στην συνέχεια θα του πρότειναν οι συμπαίχτες του κάποιες άλλες πιθανές κινήσεις και ενδεχομένως κάποιες τις οποίες μπορεί να μην είχε σκεφτεί καν. Τέλος θα σύγκρινε τις κινήσεις που του πρότειναν με την αρχική απόφαση που είχε πάρει. Ουσιαστικά θα αξιολογούσε τις κινήσεις και πάλι με βάση τον δικό του τρόπο σκέψης και θα επέλεγε αυτήν που θα φαινόταν η καλύτερη. Έτσι αποφάσισα να υλοποιήσω τους agents με βάση την παραπάνω δομή ώστε να προσομοιώνουν αυτό που θα συνέβαινε και στον πραγματικό κόσμο.

Υλοποίηση

Υλοποιήθηκαν οι συναρτήσεις **utcSearchSuggestion** και **utcSearchSolo**.

- Η **utcSearchSuggestion** θα καλείται κάθε φορά που ο agent οφείλει να δώσει μια συμβουλή σε κάποιο συμπαίχτη του για να τον συμβουλέψει την καλύτερη κίνηση κατά την δική του άποψη.
- Όταν έρχεται η σειρά του agent να παίξει, θα καλεί την συνάρτηση **utcSearchSolo.** Η συναρτήση **utcSearchSolo** εκτελείται πριν λάβει υπόψιν του τις κινήσεις που του πρότειναν οι συμπαίχτες του**.** Αυτό επιδιώκεται διότι θα πρέπει να επιλέξει ο ίδιος πρώτα για τον εαυτό του την καλύτερη

κίνηση. Στην συνέχεια αφού η συνάρτηση **utcSearchSolo** του επιστρέψει κάποια πιθανή κίνηση, θα την βαθμολογήσει με βάση την συνάρτηση **defaultPolicy**. Έπειτα θα βαθμολογήσει και τις κινήσεις που του προτάθηκαν με την ίδια συνάρτηση. Τέλος εφόσον έχει ολοκληρωθεί η βαθμολόγηση θα επιλέξει να εκτελέσει την κίνηση με το μεγαλύτερο score ανάμεσα στην δική του και των συμβουλών που δέχτηκε.

Με αυτόν τον τρόπο ο agent επιλέγει με βάση τα δικά του κριτήρια την καλύτερη κίνηση που θα μπορούσε να κάνει αλλά και αξιολογεί και τις κινήσεις που τον συμβουλεύουν οι συμπαίχτες του έτσι ώστε να ελέγξει εάν του πρότειναν κάτι καλύτερο που δεν είχε σκεφτεί.

Η συνάρτηση **defaultPolicy** ουσιαστικά προσομοιώνει την αντίληψη και τον ατομικό τρόπο σκέψης του κάθε agent για το πώς θα αξιολογούσε μια κίνηση στο συγκεκριμένο παιχνίδι. Η οποία χρησιμοποιείται και εντός των συναρτήσεων **utcSearchSuggestion** και **utcSearchSolo**. Εφαρμόζεται η ίδια συνάρτηση και για τις δύο λειτουργίες διότι και ένας άνθρωπος θα αποφάσιζε με τα ίδια κριτήρια και για τις δύο αποφάσεις.

Αξιολόγηση Συμπαιχτών

Επίσης ο agent διαθέτει έναν **πίνακα βαθμολόγησης για τον καθένα από τους συμπαίχτες του**. Ουσιαστικά δημιουργεί μια μεταβλητή για τον κάθε συνεργάτη του. Στις μεταβλητές αυτές αποθηκεύει την βαθμολογία που αντιστοιχεί στον καθένα. Πρώτα όλες οι μεταβλητές αρχικοποιούνται στο μηδέν.

Στην συνέχεια εάν κάποιος συμπαίχτης προσφέρει κάποια πολύ χρήσιμη συμβουλή και τελικά εκτελεστεί από τον agent τότε του προσδίδει βαθμολογία +5. Συνεπώς τον κατατάσσει ως συνεργάσιμο και έξυπνο συμπαίχτη ώστε στους επόμενους γύρους να δώσει περισσότερο βαρύτητα στις συμβουλές του σε σχέση με τους υπόλοιπους.

Στην περίπτωση που ο συνεργατικός παίχτης σταματήσει να του δίνει σωστές συμβουλές για παραπάνω από 5 γύρους τότε ο agent του αφαιρεί την έξτρα βαθμολογία, με στόχο ο λόγος του πλέον μη συνεργάσιμου παίχτη να σταματήσει να έχει παραπάνω βαρύτητα από των υπολοίπων.

Η διαδικασία αξιολόγησης των παιχτών που αναλύθηκε παραπάνω υλοποιήθηκε με σκοπό να συμπεριλαμβάνεται στην βαθμολόγηση των κινήσεων που προσφέρθηκαν ως συμβουλή.

Η παραπάνω διαδικασία υλοποιείται με τον εξής ψευδοκώδικα :

int value1 = evaluate (mySoloMovement)

int value2 = evaluate (suggestionMovementOfPlayer1) + scorePlayer1

int value3 = evaluate (suggestionMovementOfPlayer2) + scorePlayer2

int value4 = evaluate (suggestionMovementOfPlayer3) + scorePlayer3

bestMovement = findBestValue(value1,value2,value3,value4)

if (bestMovement == value2){

    scorePlayer1 = 5;

}else if (bestMovement == value3){

    scorePlayer2 = 5;

} else if (bestMovement == value4){

    scorePlayer3= 5;

}

If player1 didn't helped for 5 rounds then => scorePlayer1 = 0;

If player2 didn't helped for 5 rounds then => scorePlayer2 = 0;

If player3 didn't helped for 5 rounds then => scorePlayer3 = 0;

Πίνακας Καταστάσεων Παιχνιδιού

Για να μπορέσω να φτιάξω ένα δέντρο καταστάσεων του παιχνιδιού χρειάστηκε να δημιουργηθεί μια κλάση Node η οποία περιέχει τις εξής μεταβλητές:

- ✓ `private Board curBoardState; -> την τωρινή κατάσταση του Board`
- ✓ private Node parent; -> link για το parent node
- ✓ private ArrayList<Node> childs; -> links για τα child nodes
- ✓ private boolean expanded; -> Boolean μεταβλητή για τον αν έχουμε κάνει το συγκεκριμένο node fully expand

- ✓ private String prevMovement; -> String που περιέχει την προηγούμενη κίνηση που έγινε ώστε να φτάσουμε στην τωρινή κατάσταση

- ✓ private int whoAmI; -> χρησιμοποιείται για να ξέρουμε τι user id έχει ο παίχτης μας

- ✓ private int whoIsPlayingInitialy; -> χρησιμοποιείται στην περίπτωση που έχουμε suggestion ώστε να ξέρουμε ποιος παίχτης παίζει αρχικά ώστε να του δώσουμε συμβουλή

- ✓ private int timesVisited; -> είναι οι φορές που το έχουμε επισκεφθεί το συγκεκριμένο node

- ✓ private int reward; -> η ανταμοιβή που έχει οριστεί για τον συγκεκριμένο node (έχει να κάνει με το τι ανταμοιβές έχουν τα παιδιά του (UTC))

- ✓ private int depth; -> εδώ ορίζουμε σε τι βάθος έχουμε φτάσει ώστε να ξέρει ο αλγόριθμος σε ποιο βάθος να σταματήσει επειδή δεν γίνεται να τρέχουμε τον MCST μέχρι να φτάσουμε σε terminal ή leaf Node

Τρόπος Σκέψης

Αρχικά υλοποίησα τις συναρτήσεις **utcSearchSuggestion** και **utcSearchSolo** ώστε να λαμβάνουν υπόψη και να προσομοιώνουν όλες τις πιθανές κινήσεις που μπορούσαν να κάνουν όλοι οι συμπαίχτες του κάθε agent. Δηλαδή στην περίπτωση που εκτελούσαμε την **utcSearchSolo** αρχικά στον πρώτο expand υπολογίζονται όλες οι πιθανές κινήσεις του δικού μας agent. Ενώ αν ξανά γίνει δεύτερο expand σε κάποιο από αυτά τα childs που δημιουργήθηκαν από το πρώτο expand, τότε θα υπολογιστούν όλες οι πιθανές κινήσεις του παίχτη που έχει σειρά να παίξει μετά από τον δικό μας agent κ.ο.κ.

Στην συνέχεια αποφάσισα πως σωστότερο θα ήταν να μην βασίζεται ο agent μας στο τι κινήσεις θα μπορέσουν να κάνουν οι υπόλοιποι και να μην σπαταλάει τους υπολογιστικούς του πόρους σε προσομοιώσεις κινήσεων των συμπαικτών του. Αυτό είναι μάταιο διότι δεν θα μπορούσε να προσομοιώσει και να υπολογίσει τι κινήσεις μπορεί να κάνει ο κάθε ένας. Μερικοί συμπαίχτες του μπορεί να μην είναι καθόλου συνεργάσιμοι και απλά να κάνουν κύκλους σε κάθε γύρω και να σπαταλούν άσκοπα τις κινήσεις τους. Θα ήταν προτιμότερο να βασίζεται αποκλειστικά και μόνο στις κινήσεις που μπορεί να κάνει ο ίδιος και έτσι γλυτώνοντας πολλούς υπολογιστικούς πόρους να μπορέσει να προσομοιώσει περισσότερους ατομικούς του γύρους κινήσεων. Έτσι υλοποιήθηκαν οι συναρτήσεις **utcSearchSuggestion2** και **utcSearchSolo2.**

Υλοποίηση Τελικών Συναρτήσεων Αναζήτησης

Η συνάρτηση **utcSearchSolo2** στο πρώτο expand υπολογίζει όλες τις πιθανές κινήσεις του δικού μας agent και φτιάχνει όλα τα πιθανά child node από αυτές τις κινήσεις. Σε κάθε νέο child node που φτιάχνεται προσομοιώνει μια ολόκληρη παρτίδα μέχρι να ξανά φτάσει η σειρά του agent μας, υποθέτοντας πώς οι υπόλοιποι παίχτες απλά κάνουν άσκοπες κινήσεις και δεν προφέρουν κάτι για την συνέχεια ή την νίκη του παιχνιδιού. Επίσης προσομοιώνει και τυχαίες κάρτες που μπορεί να τραβήξει ο κάθε παίχτης(και οι 4 παίχτες). Στο δεύτερο expand έχει σειρά ξανά ο παίχτης μας διότι έχουμε υποθέσει ότι οι υπόλοιποι έχουν ήδη παίξει

και έχουν ήδη τραβήξει κάρτες. Έτσι σε αυτήν την συνάρτηση αναζήτησης το βάθος κάθε node ταυτίζεται με τις παρτίδες που έχει παίξει ο agent μας.

Η συνάρτηση **utcSearchSuggestion2** ακολουθεί την ίδια δομή με την συνάρτηση **utcSearchSolo2** με την διαφορά ότι δεν υπολογίζονται μόνο οι κινήσεις του agent που καλούμαστε να συμβουλέψουμε αλλά και οι δικές μας. Αποθηκεύονται εντός της κλάσης node το user id του agent μας και το user id του συμπαίχτη που καλούμαστε να συμβουλέψουμε. Ουσιαστικά παίζει εναλλάξ ο agent μας με αυτόν που συμβουλεύουμε και υποθέτουμε ότι οι υπόλοιποι κάνουν άσκοπες κινήσεις και παίρνουν κάρτες στο τέλος του γύρου τους.

Στον κώδικα του πρότζεκτ έχω υλοποιήσει και τις δύο εκδόσεις των συναρτήσεων **utcSearchSuggestion** - **utcSearchSolo**, απλά χρησιμοποιούνται εξολοκλήρου μόνο οι δεύτερες εκδόσεις για τους λόγους που ανέφερα παραπάνω.

Προκειμένου να μπορώ να δημιουργώ όλα τα πιθανά child nodes εντός των συναρτήσεων expand έχω υλοποιήσει τις συναρτήσεις **getAllNewStatesWithoutCards** και **getAllNewStatesWithCards**. Η διαφορά των δύο αυτών συναρτήσεων είναι ότι στην **WithoutCards** δεν γνωρίζουμε τα χαρτιά που έχει ο παίχτης ενώ στην **WithCards** γνωρίζουμε.

Για την υλοποίηση των παραπάνω συναρτήσεων δημιούργησα μια κλάση **Action** η οποία χρησιμοποιείται ώστε να αναπαραστεί μια από τις τέσσερεις κινήσεις που μπορεί να κάνει ένας παίχτης σε κάθε γύρο. Η κλάση αυτή έχει τις εξής μεταβλητές:
private String stringAction; -> η σειρά από actions που έχει κάνει ο παίχτης μέχρι στιγμής

private Board newBoardState; -> η τωρινή κατάσταση του board που έχει προκύψει μετά από τις κινήσεις που έχει κάνει μέχρι στιγμής ο παίχτης

private Boolean oeMove; -> Boolean μεταβλητή ώστε να γνωρίζουμε αν ο παίχτης έχει εκτελέσει μια Operation Expert Move ώστε να μην ξανά κάνει

private ArrayList<String> prevCities; -> μια λίστα με προηγούμενες πόλεις από τις οποίες έχει έρθει ο παίχτης ώστε να μην τις ξανά επισκεφθεί

Οι συναρτήσεις **findAllPossibleMovesForThisRoundWithoutCards** και **findAllPossibleMovesForThisRoundWithCards** δημιουργούν μια λίστα με όλα τα πιθανά **Actions** που θα μπορούσαν να γίνουν από ένα συγκεκριμένο από παίχτη σε μια συγκεκριμένη κατάσταση του παιχνιδιού. Ουσιαστικά δεν επιστρέφουν όλους τους πιθανούς συνδυασμούς τεσσάρων κινήσεων που θα μπορούσε να κάνει ο παίχτης αλλά όλες τις πιθανές κινήσεις μιας από τις τέσσερεις κινήσεις. Στην συνέχεια οι συναρτήσεις **getAllNewStatesWithoutCards** και **getAllNewStatesWithCards** καλούν τέσσερεις φορές τις παραπάνω συναρτήσεις έτσι ώστε να δημιουργήσουν όλους τους πιθανούς συνδυασμούς των τεσσάρων κινήσεων που θέλουμε να βρούμε.

Pruning των πιθανών κινήσεων

Αρχικά υλοποίησα τις παραπάνω συναρτήσεις ώστε απλά να δημιουργούν όλους τους συνδυασμούς όλων των τεσσάρων πιθανών κινήσεων. Στην συνέχεια όμως για να μειώσω το τεράστιο αυτό πλήθος από πιθανούς συνδυασμούς, επέλεξα να χρησιμοποιήσω uniform κατανομή πιθανοτήτων μέσα στον κώδικα ώστε όταν ο παίχτης έχει για παράδειγμα ήδη κάνει την κίνηση drive το μία φορά, να έχει λιγότερες πιθανότητες να ξανά κάνει drive το μέσα στον ίδιο γύρο. Οι πιθανότητες που εφαρμόστηκαν για την υλοποίηση του κώδικα είναι οι παρακάτω:

$1^{η}$ φορά εκτέλεσης μιας κίνησης = 100% πιθανότητα να συμβεί
$2^{η}$ φορά εκτέλεσης μιας κίνησης = 50% πιθανότητα να συμβεί
$3^{η}$ φορά εκτέλεσης μιας κίνησης = 30% πιθανότητα να συμβεί
$4^{η}$ φορά εκτέλεσης μιας κίνησης = 10% πιθανότητα να συμβεί

Οι παραπάνω χρήση πιθανοτήτων χρησιμοποιήθηκε μόνο για τις δευτερεύοντες κινήσεις που δεν έχουν μεγάλη βαρύτητα για την νίκη του παιχνιδιού.

Συνάρτηση Αξιολόγησης Στρατηγικής - defaultPolicy

Η συνάρτηση **defaultPolicy** όπως αναφέρθηκε και παραπάνω χρησιμοποιείται ώστε να αξιολογήσει και να βαθμολογήσει έναν συνδυασμό τεσσάρων κινήσεων που έγινε σε κάποια συγκεκριμένη κατάσταση του παιχνιδιού. Πιο συγκεκριμένα βαθμολογεί τις κινήσεις με τον παρακάτω τρόπο :

- Drive / Ferry:
  - Αν role = Medic =>

$$if\ (black = cured)$$
$$reward+= destinationCity.blackCubes$$
$$if\ (yellow = cured)$$
$$reward+= destinationCity.yellowCubes$$
$$if\ (blue = cured)$$
$$reward+= destinationCity.blueCubes$$
$$if\ (red = cured)$$
$$reward+= destinationCity.redCubes$$

  - Αν role != Medic => $reward = 0$

- Direct Flight:
  - Αν role = Medic =>

$$if\ (black = cured)$$
$$reward+= destinationCity.blackCubes$$
$$if\ (yellow = cured)$$
$$reward+= destinationCity.yellowCubes$$
$$if\ (blue = cured)$$
$$reward+= destinationCity.blueCubes$$
$$if\ (red = cured)$$
$$reward+= destinationCity.redCubes$$

$$if\ (colorOfCardToThrow = black)$$
$$reward-= totalBlackCardsOfHand * 4$$

$$if\ (colorOfCardToThrow = yellow)$$
$$reward{-}= totalYellowCardsOfHand * 4$$

$$if\ (colorOfCardToThrow = blue)$$
$$reward{-}= totalBlueCardsOfHand * 4$$

$$if\ (colorOfCardToThrow = red)$$
$$reward{-}= totalRedCardsOfHand * 4$$

- Charter Flight:
  - Av role = Medic $=>$
$$if\ (black = cured)$$
$$reward{+}= destinationCity.blackCubes$$
$$if\ (yellow = cured)$$
$$reward{+}= destinationCity.yellowCubes$$
$$if\ (blue = cured)$$
$$reward{+}= destinationCity.blueCubes$$
$$if\ (red = cured)$$
$$reward{+}= destinationCity.redCubes$$

  - $reward{-}= 5$

- Shuttle Flight:
  - Av role = Medic $=>$
$$if\ (black = cured)$$
$$reward{+}= destinationCity.blackCubes$$
$$if\ (yellow = cured)$$
$$reward{+}= destinationCity.yellowCubes$$
$$if\ (blue = cured)$$
$$reward{+}= destinationCity.blueCubes$$
$$if\ (red = cured)$$
$$reward{+}= destinationCity.redCubes$$

  - Av role != Medic $=>$ $reward = 0$

- Build a Research Station:
  - $reward = 10$

- Treat disease:
  - $if\ (role = Medic)\ \ =>$
    $reward += 10$

  - $if\ (getCubesLeft(colorToTreat) < 3)$
    $reward += 20$

  - $reward += cityToTreat.getCubes(colorToTreat) * 4$

  - $if\ (cityToTreat.getCubes(colorToTreat) > 2)$
    $reward -= ((\ cityToTreat.getCubes(colorToTreat) - 2) * 4) + 1$

  - $else\ if\ (cityToTreat.getCubes(colorToTreat) = 2)$
    $reward += 2 * board.getOutbreaksAlreadyOccured$
    $reward += countOfNeighboursWith2Cubes * 5$

- Discover a Cure:
  $reward = getCubesOfColorToCureInsideBoard + 20$

- Operation Expert Travel:
  - Av role = Medic $=>$
    $if\ (black = cured)$
    $reward += destinationCity.blackCubes$
    $if\ (yellow = cured)$
    $reward += destinationCity.yellowCubes$
    $if\ (blue = cured)$
    $reward += destinationCity.blueCubes$
    $if\ (red = cured)$
    $reward += destinationCity.redCubes$

    $if\ (colorOfCardToThrow = black)$

$$reward -= totalBlackCardsOfHand * 3$$

$$if\ (colorOfCardToThrow = yellow)$$
$$reward -= totalYellowCardsOfHand * 3$$

$$if\ (colorOfCardToThrow = blue)$$
$$reward -= totalBlueCardsOfHand * 3$$

$$if\ (colorOfCardToThrow = red)$$
$$reward -= totalRedCardsOfHand * 3$$

Για την διαμόρφωση της παραπάνω αρχιτεκτονικής της συνάρτησης **defaultPolicy** χρειάστηκαν να πραγματοποιηθούν πολλές πειραματικές εκτελέσεις του παιχνιδιού ώστε να βρεθούν οι κατάλληλες ανταμοιβές οι οποίες οδηγούν τους agents πιο κοντά στην νίκη.

Παρόλο που δοκιμάστηκαν πολλές αλλαγές στην αρχιτεκτονική φαίνεται ότι υπάρχει δυσκολία στην κατάκτηση της νίκης. Αρχικά στις πρώτες δοκιμές δόθηκε πολύ μεγάλη έμφαση στις κινήσεις cure για να πραγματοποιηθεί μια γρήγορη νίκη. Όμως αυτό οδήγησε σε ήττες στους αρχικούς κιόλας γύρους, διότι οι agents δεν έδιναν την απαραίτητη σημασία στα treat disease ώστε να διασφαλίσουν περισσότερο χρόνο παιχνιδιού μέχρι να έχουν τις απαραίτητες κάρτες με στόχο να κάνουν cure.

Στην συνέχεια χρειάστηκε να γίνουν πολλές δοκιμές μέχρι να βρεθεί η κατάλληλη ισορροπία ανταμοιβών μεταξύ των treat και cure disease έτσι ώστε οι agents να γνωρίζουν πότε πρέπει να θυσιάσουν τις κάρτες και τις κινήσεις τους ώστε να μην χάσουν και πότε θα πρέπει εξοικονομήσουν τις κάρτες τους ώστε στο μέλλον να τους δοθεί η ευκαιρία να κάνουν cure disease.

Αποτελέσματα

Στην παρούσα φάση, στις περισσότερες περιπτώσεις οι agents χάνουν περίπου στον 23ο γύρο επειδή τους έχουν τελειώσει οι κάρτες και έχουν θεραπεύσει τους 3 από τους 4 ιούς. Αυτό μας δείχνει ότι φθάνουν πάρα πολύ κοντά στην νίκη αλλά χάνουν επειδή δεν κάνουν καλή εξοικονόμηση καρτών ώστε να μπορέσουν να θεραπεύσουν και τους 4 ιούς.

Σε αυτό το στάδιο δοκίμασα να αυξήσω το κόστος κατά 1 ή 2 μονάδες των κινήσεων που χρησιμοποιούν κάρτες αλλά αυτό οδήγησε στο τεράστιο αρνητικό αποτέλεσμα να χάνουν μέσα στους πρώτους 10 γύρους. Γίνεται αντιληπτό ότι η παραμικρή αλλαγή στρατηγικής ακόμα και για 1 μονάδα μπορεί να οδηγήσει σε τελείως διαφορετικά αποτελέσματα.

Προκειμένου να αυξηθούν τα ποσοστά νίκης μας θα πρέπει να συνεχιστεί η αλλαγή των νούμερων των ανταμοιβών μέχρι να βρεθεί η «χρυσή τομή» των συνδυασμών που θα οδηγήσουν στην βέλτιστη στρατηγική.

Τα στατιστικά αποτελέσματα του κώδικα στο τελικό τουρνουά είναι τα εξής :

2 (same) + 2(dummy)

Win (%)	Cubes (Avg)	Card (Avg)	Cures (%)	Outbreak (Avg)
0	53.5	22	0	8

3 (same) + 1(dummy)

Win (%)	Cubes (Avg)	Card (Avg)	Cures (%)	Outbreak (Avg)
0	54	14	0	7

3 (team) + 1(dum)

Win (%)	Cubes (Avg)	Card (Avg)	Cures (%)	Outbreak (Avg)
0	48	14	25	8

Παρατηρήθηκαν σημαντικές αποκλείσεις αποτελεσμάτων όταν μειώθηκε το budget χρόνου που έχουν ως όρισμα οι συναρτήσεις αναζήτησης. Αυτό οφείλεται στο ότι το δέντρο πιθανών κινήσεων παρόλο που έχει δεχθεί ένα pruning με πιθανότητες κανονικής κατανομής εξακολουθεί να είναι πολύ περίπλοκο και με πάρα πολλές διακλαδώσεις. Αυτό συμβαίνει διότι ο αλγόριθμός μου εντοπίζει όλες τις πιθανές καταστάσεις που μπορούν να προκύψουν με τον συνδυασμό όλων των πιθανών κινήσεων. Είναι αντιληπτό ότι αυτό προσδίδει τεράστια πολυπλοκότητα η οποία και αυξάνεται εκθετικά καθώς μεγαλώνει το βάθος του δέντρου. Αυτή η μεγάλη πολυπλοκότητα απαιτεί πολύ χρόνο για τον υπολογιστή ώστε να καταφέρει μια αξιολόγηση μεγάλου αριθμού διακλαδώσεων και να καταλήξει σε μια καλή κίνηση.

Κλείνοντας, προκειμένου να βελτιωθεί η απόδοση του αλγόριθμου η καλύτερη στρατηγική θα ήταν να μειωθεί περαιτέρω η πολυπλοκότητα του δέντρου πιθανών κινήσεων-καταστάσεων και να βελτιωθεί ο τρόπος αξιολόγησης των κινήσεων εντός της defaultPolicy με στόχο την πραγματοποίηση περισσότερων cures χωρίς όμως οι agents να χάνουν το παιχνίδι από τους υπόλοιπους παράγοντες.