Τεχνητή Νοημοσύνη

2η Προγραμματιστική Εργασία

Ομάδα: LAB31146705

Προύντζος Αθανάσιος, ΑΜ: 2016030078

Πάντος Κωνσταντίνος, ΑΜ: 2016030015

Εισαγωγή

Σκοπός αυτής της εργασίας είναι να σχεδιάσουμε και να υλοποιήσουμε ένα πρόγραμμα που θα παίζει το παιχνίδι TUC-CHESS που αποτελεί μία παραλλαγή του γνωστού παιχνιδιού σκάκι. Για τον σκοπό αυτό υλοποιήθηκαν δύο αλγόριθμοι ο minimax και ο Monte Carlo tree search. Για την υλοποίηση των αλγορίθμων δημιουργήθηκαν επίσης και evaluation functions που χρησιμοποιήθηκαν ως αξιολογήσεις για την κάθε κίνηση μέσα στο παιχνίδι. Μέσω διάφορων δοκιμών και επεκτάσεων στους αλγορίθμους εξάγαμε πολλά συμπεράσματα σε σχέση με την συμπεριφορά του παίκτη και την εξέλιξη του παιχνιδιού για κάθε αλγόριθμο ξεχωριστά.

1.Υλοποίηση

Evaluation Function

Αρχικά ως συνάρτηση αξιολόγησης χρησιμοποιήσαμε τη συνάρτηση της εκφώνησης. Έπειτα προσθέσαμε πληροφορία σχετικά με το αν οι βασιλιάδες βρισκόντουσαν υπό επίθεση αλλά και σχετικά με τον αριθμό των πιονιών που «προστάτευαν ή απειλούσαν» τον βασιλιά και παρατηρήσαμε μια αμυντική στάση των πιονιών κοντά του. Ακόμη με την προσθήκη πληροφορίας σχετικά με τη μικρότερη απόσταση των πιονιών από τον αντίπαλο βασιλιά αλλά και το τέλος της σκακιέρας τα πιόνια έγιναν πιο επιθετικά. Τέλος προσθέσαμε πληροφορία σχετικά με τα πιο κοντινά πιόνια σε «δώρα», κάτι που όμως δε θέλαμε να επηρεάζει υπερβολικά τη ροή του παιχνιδιού, έτσι ώστε τα πιόνια να μη ξεφεύγουν από τον μεγαλύτερό τους στόχο, την αιχμαλώτιση του βασιλιά. Για το λόγο αυτό, έπειτα από αρκετό πειραματισμό θέτοντας βάρη στη κάθε πληροφορία καταλήξαμε στην εξής συνάρτηση αξιολόγησης,

$$eval = 5 * \big(Math.abs(info[0] + info[2]) - Math.abs(info[1] + info[3])\big) - 5 * (underAttack[0] - underAttack[1]) - 5 * (min\_distance[0] - min\_distance[1]) - 5 * (dinstanceFromKings[0] - dinstanceFromKings[1]) + 1 * (prizeDistance[0] - prizeDistance[1]) - 5 * (kingPoints[0] - kingPoints[1])$$

όπου η info περιέχει πληροφορία για τον αριθμό των πιονιών που βρίσκονται στη σκακιέρα και το άθροισμα των πόντων που αυτά, η underAttack σχετικά με το αν οι βασιλιάδες πλησιάζονται από αντίπαλο πιόνι, η distanceFromKings και min_distance για την μικρότερη απόσταση (Manhattan distance) των πιονιών από τους αντίπαλους βασιλιάδες και το τέρμα τής σκακιέρας αντίστοιχα και τέλος η kingPoints για τον αριθμό των πιονιών που προστατεύουν ή απειλούν τον κάθε βασιλιά.

Minimax

Αρχικά δημιουργήθηκαν οι συναρτήσεις makeMove που αποτελεί απλοποιημένη μορφή της ήδη υπάρχουσας χωρίς όμως αναφορά στα «δώρα» και η deepCopy ώστε να δημιουργείται νέο αντίγραφο του array του board γιατί οι αλλαγές μετά από κάποια κίνηση παρέμεναν στο αρχικό board. Πριν την εκτέλεση του αλγορίθμου της minimax δημιουργούμε ένα αντίγραφο του world ώστε να μην επηρεάζονται τα στοιχεία/μεταβλητές του. Κατά την εκτέλεση του αλγορίθμου δημιουργούνται τόσα αντίγραφα του board όσο και ο αριθμός των κινήσεων που ελέγχονται. Ξεκινώντας την εκτέλεση έχοντας θέσει ένα επιτρεπόμενο βάθος, εκτελείται η αναδρομή και ελέγχονται κάθε φορά οι επόμενες κινήσεις του αντιπάλου αλλάζοντας αντίστοιχα τον «παίκτη», μειώνοντας σε κάθε αναδρομή την τιμή του βάθους κατά μία μονάδα. Όταν ο αλγόριθμος φτάσει σε βάθος με τιμή 0 επιστρέφει την τιμή αξιολόγησης του board, με χρήση του evaluation function που προαναφέρθηκε, για τη δεδομένη κατάσταση του. Χρησιμοποιώντας την τιμή αυτή ο αλγόριθμος επιλέγει την κατάλληλη κίνηση του αρχικού μας «παίκτη».

[Σημείωση: Όταν ο παίκτης ήταν ο λευκός μετά από αρκετές δοκιμές παρατηρήσαμε ότι μετα από έναν αριθμό κινήσεων που επέτρεπαν τον δεξιό πύργο να μετακινηθεί, αυτός εκτελούσε συνεχόμενα την ίδια οριζόντια κίνηση, κάτι που οδηγούσε στην ήττα του λευκού παίκτη. Αυτό διορθώθηκε με την αντιστροφή του array των available moves όταν ο παίκτης είναι ο λευκός.]

Μετα από εκτελέσεις του αλγόριθμου για διαφορετικά βάθη είδαμε μεγαλύτερη απόδοση για βάθος με τιμή 7. Αρκετές φορές όμως η επιλογή της κατάλληλης κίνησης καθυστερούσε περισσότερο από το ζητούμενο χρονικό πλαίσιο των 6 δευτερολέπτων. Αυτό αντιμετωπίστηκε με την επέκτασή της minimax με χρήση a-b pruning. Έτσι πετύχαμε μείωση τον καταστάσεων που η minimax έκανε evaluate άρα και ελάττωση του χρόνου εκτέλεσης, καταλήγοντας όμως στην ίδια επιλογή

κίνησης με την κίνηση που επέλεγε ο αλγόριθμος πριν την επέκταση. Τέλος για να σιγουρέψουμε ότι δεν παραβιάζεται το επιτρεπόμενο χρονικό πλαίσιο, προστέθηκε έλεγχος που σταματάει την επέκταση του δέντρου.

[Σημείωση: Λόγω χρονικής πίεσης και φόρτου και από άλλες εργασίες δεν καταφέραμε να πετύχουμε περαιτέρω βελτιώσεις, παρά μόνο του a-b pruning.]

Monte Carlo Tree Search

Για την υλοποίηση του δημιουργήσαμε μια δεντρική δομή η οποία περιέχει σαν κόμβους την κατάσταση που βρίσκεται το παιχνίδι μετά από κάποια συγκεκριμένη κίνηση . Ως κατάσταση ορίσαμε την εικόνα του board ανά πάσα στιγμή μετά την εκτέλεση κάποιας κίνησης. Για το σκοπό αυτό δημιουργήσαμε μια συνάρτηση η οποία επιστρέφει όλες τις πιθανές καταστάσεις στις οποίες μπορούμε να μεταβούμε από την τωρινή μας κατάσταση. Ουσιαστικά προσομοιώναμε κάθε διαθέσιμη κίνηση που είχαμε και επιστρέφαμε την κατάσταση του board μετά την εκτέλεση της κινήσεων αυτών.

Η συνάρτηση αυτή μας φάνηκε χρήσιμη στην επέκταση των κόμβων και το δέντρου μας για να τρέξει ο αλγόριθμος. Η ποιότητα του κάθε κόμβου καθοριζόταν από μία μετρική που ονομάζεται uct και ορίζεται παρακάτω:

$$uct(n) = \frac{n.value}{n.numOfVisits} + c\sqrt{\frac{\log\left(parent.numOfVisits\right)}{n.numOfVisits}}$$

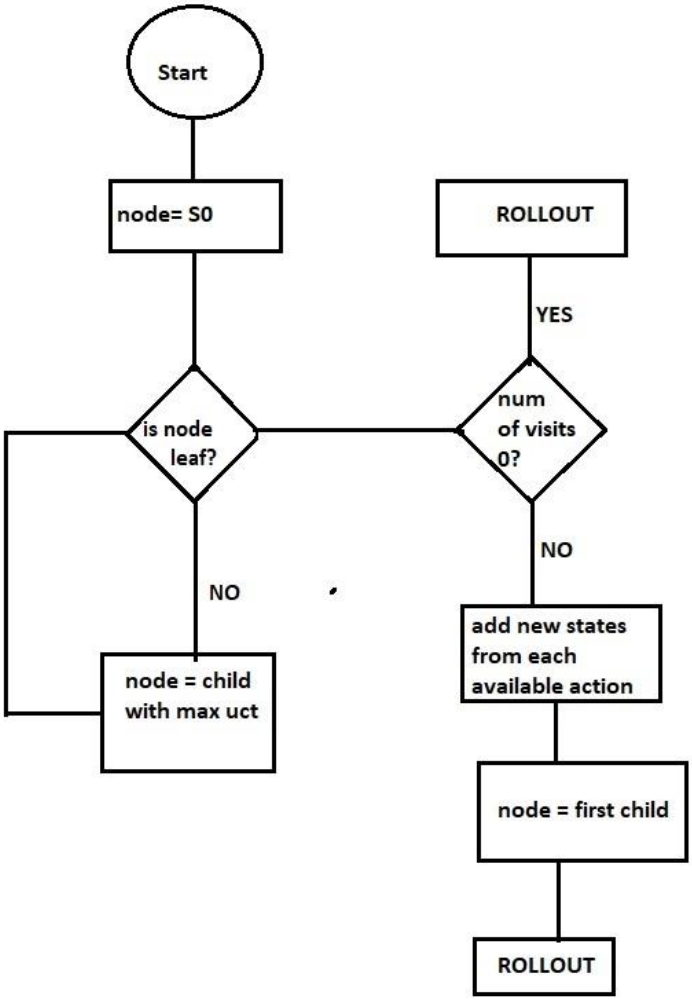
,όπου n ένα τυχαίος κόμβος και c μία σταθερά(c=2 στην περίπτωση μας).

Ο πρώτος όρος μας δίνει το exploitation , δηλαδή την αναμενόμενη αξία που θα κερδίσουμε σε περίπτωση επιλογής αυτού του κόμβου, ενώ ο δεύτερος αφορά το exploration μας δίνει τη συχνότητα επίσκεψης του κόμβου σε σχέση με τον γονέα του. Η σταθερά c ουσιαστικά μας δείχνει το βαθμό σημασίας του exploration σε σχέση με το exploitation.

Η επιλογή του κάθε κόμβου και ουσιαστικά της κίνησης για κάθε παίκτη γίνεται με κριτήριο την μεγιστοποίηση της μετρικής που αναφέραμε παραπάνω.

Να σημειωθεί ότι για την ενημέρωση της εκτίμησης της αξίας του κάθε κόμβου κάνουμε διάφορες προσομοιώσεις (rollout) μελλοντικών κινήσεων του παιχνιδιού που μας επιστρέφουν μια τιμή που χρησιμοποιείται για να ενημερώσει την αξία του κάθε κόμβου πάνω στο δέντρο. Τυπικά το rollout γίνεται μέχρι την λήξη του παιχνιδιού, ωστόσο εδώ για λόγους εξοικονόμησης χρονικής πολυπλοκότητας χρησιμοποιούμε ένα όριο βάθους κινήσεων που αν ξεπεραστεί τερματίζει η προσομοίωση. Ο περιορισμός της αποστολής της κίνησης στα 6 δευτερόλεπτα δεν υλοποιήθηκε καθώς για τον αριθμό των επαναλήψεων και το βάθος στις προσομοιώσεις παρατηρήθηκε ότι κάθε κίνηση δεν ξεπέρναγε το όριο αυτό.

Η διαδικασία που περιγράψαμε φαίνεται και στο παρακάτω σχήμα.



2. Αξιολόγηση-Συμπεράσματα

Δοκιμάσαμε και τρέξαμε το παιχνίδι αρκετές φορές θέλοντας να δούμε την εξέλιξη του παιχνιδιού όταν ένας παίκτης επιλέγει κινήσεις σύμφωνα με τον minimax και ο άλλος σύμφωνα με τον Monte Carlo. Για τον Monte Carlo παρατηρήθηκε ότι για αριθμό επαναλήψεων ίσο με 1000 και για βάθος 50 στις προσομοιώσεις γίνονται οι καλύτερες επιλογές κινήσεων. Γενικά και στις δύο περιπτώσεις παρατηρείται λογική

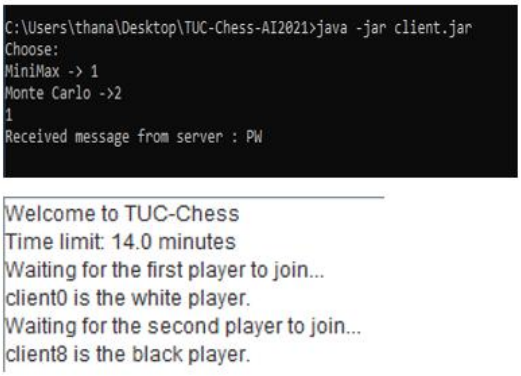
στις κινήσεις των παικτών καθώς φαίνεται η προσπάθεια ανάπτυξης και άμυνας καθώς επίσης και η τάση να πάρουν το bonus όταν αυτό είναι κοντά τους. Ωστόσο υπάρχουν και περιπτώσεις που δεν επιλέγονται και οι προφανώς καλές κινήσεις κατά την διάρκεια του παιχνιδιού. Αυτό οφείλεται στην απλοϊκότητα της evaluation function και την έλλειψη λεπτομέρειας στην περίπτωση του minimax και στην οριοθέτηση του βάθους κινήσεων για τα rollouts στην περίπτωση του Monte Carlo.

Συγκριτικά μέσα από τις δοκιμές παρατηρείται ότι ο minimax κερδίζει επί το πλείστων τον Monte Carlo στις μεταξύ τους αναμετρήσεις. Αυτό οφείλεται στο ότι ο minimax διαλέγει κινήσεις με βάση τον αντίπαλο κάτι που ταιριάζει στα πρότυπα του σκακιού σε αντίθεση με τον monte Carlo που «χτίζει» ουσιαστικά εμπειρία και λειτουργεί καλύτερα σε παιχνίδια με υψηλό παράγοντα διακλάδωσης.

Παρακάτω φαίνεται η κατάληξη ενός παιχνιδιού ανάμεσε σε έναν minimax και έναν Monte Carlo παίκτη.

Δοκιμή Minimax εναντίων Monte Carlo

Όπως φαίνεται στις παρακάτω εικόνες, τρέχουμε ένα match, με τον Minimax ως τον λευκό παίκτη (client0) και τον Monte Carlo ως τον μαύρο παίκτη (client8). Ο Minimax τρέχει για βάθος 7, ενώ ο Monte Carlo για βάθος 50 και 1000 iterations. Παρατηρούμε ότι ο Minimax νικάει. Αντιστρέφοντας ρόλους πάλι οδηγηθήκαμε στο ίδιο αποτέλεσμα με τον Minimax ως νικητή.



3.Βιβλιογραφία- Σύνδεσμοι

- https://www.youtube.com/watch?v=UXW2yZndl7U&t=666s
- https://www.youtube.com/watch?v=l-hh51ncgDI&t=38s
- Διαλέξεις Μαθήματος – Φροντιστήρια