Τεχνητή Νοημοσύνη – ΠΛΗ417
Εργασία Προγραμματισμού TUC-CHESS

Μερσινιάς Μιχαήλ
Τρουλλινός Δημήτριος
Κωδικός Ομάδας: LAB41732909

1) Δημιουργία δέντρου:

Για την υλοποίηση του agent, αξιοποιήσαμε τον δοσμένο κώδικα του Client, και τον αναπτύξαμε σε γλώσσα Java. Χρησιμοποιήσαμε την κλάση Node που είχαμε υλοποιήσει για το 1ο project, η οποία χρειάστηκε για την δημιουργία του δέντρου. Επιπλέον, δημιουργήσαμε μια κλάση Vertex με τα απαραίτητα δεδομένα για κάθε δυνατή κατάσταση παιχνιδιού, που αντικείμενά της θα περιέχονται σε κάθε κόμβο. Για την δημιουργία του δέντρου καταστάσεων, χρησιμοποιούμε τις συναρτήσεις που περιέχει ο Client, με κάποιες μικρές διαφοροποιήσεις για να ταιριάζει ο κώδικας στην υλοποίησή μας(μέθοδοι whiteMoves,blackMoves,makeMove). Το δέντρο δημιουργείται όταν λάβουμε μήνυμα από τον server ότι αρχίζει το παιχνίδι. Θεωρούμε πως πάντα ο παίχτης που έχει τα λευκά είναι ο MAX και ο παίχτης που έχει τα μαύρα ο MIN. Μας εξυπηρετεί αυτή η σύμβαση καθώς το δέντρο ξεκινάει με αρχική κατάσταση πριν γίνει η πρώτη κίνηση.

Έχουμε ορίσει μέγιστο βάθος δέντρου maxDepth=4. Αυτό σημαίνει ότι βλέπουμε 4 κινήσεις μετά την τρέχουσα. Η δέσμευση και η αποδέσμευση μνήμης είναι κάτι που κοστίζει σε χρόνο, και γι αυτό το λόγο, σε κάθε νέα αναζήτηση στο δέντρο, δημιουργούμε μόνο το τελευταίο επίπεδο αναζήτησης(βάθος=4), αφού έχουμε διατηρήσει τους κόμβους που είχαμε δημιουργήσει σε προηγούμενη αναζήτηση. Το πρόβλημα που δημιουργείται με αυτή την λογική είναι ότι τα δώρα εμφανίζονται με τυχαίο τρόπο, και η αξία που θα δώσουν ακολουθεί μία πιθανοτική κατανομή. Για να αντιμετωπιστεί αυτό το πρόβλημα, σε κάθε νέα αναζήτηση, σε κάθε κόμβο που επισκεπτόμαστε ξανά, ανανεώνουμε τη σκακιέρα και το σκορ. Όμως, με την εμφάνιση ενός δώρου, υπάρχει η περίπτωση να περιοριστούν οι κινήσεις του πύργου. Σε τέτοια περίπτωση, καθώς επισκεπτόμαστε τους κόμβους, διαγράφουμε αυτούς που αντιστοιχούν σε κίνηση που δεν μπορεί να πραγματοποιηθεί πλέον. Όταν πραγματοποιείται μία κίνηση στο παιχνίδι, καλείται η συνάρτηση makeMove, και μεταφερόμαστε στον κόμβο που αντιστοιχεί στην κίνηση που έγινε, για να μπορούμε να κάνουμε αναζήτηση της επόμενης κίνησης.

2) Αλγόριθμος αναζήτησης Minimax:

Ο Minimax είναι ένας αναδρομικός αλγόριθμος αξιολόγησης κινήσεων ο οποίος προσπαθεί να διαλέξει μια κίνηση, για τον κάθε παίκτη Max και Min εναλλάξ κάθε φορά, έτσι ώστε να μεγιστοποιήσει το τελικό κέρδος μας.

Όσον αφορά την τεχνική υλοποίηση του αλγορίθμου, αυτή έγινε με την εξής λογική:

- Αρχικά, κάνουμε έναν έλεγχο για το εάν έχει τελειώσει το παιχνίδι ή εάν έχουμε εξαντλήσει όλες τις δυνατές κινήσεις (δηλαδή να βρεθεί στην ρίζα, depth = 0). Σε αυτές τις περιπτώσεις η συνάρτηση Minimax επιστρέφει το τελικό score μέσω του Evaluation function που επιστρέφει μια τιμή ανάλογα με τον αλγόριθμο που περιγράφεται στην ενότητα 4 της αναφοράς "Συνάρτηση αξιολόγησης".

- Έπειτα, έχουμε ένα while loop στο οποίο προσπελαύνουμε με την χρήση Iterator το ArrayList που αναπαριστά όλα τα Available και Legal moves και δημιουργούμε τα παιδιά του κόμβου μας(Αν έχουν δημιουργηθεί από προηγούμενη κίνηση, απλά τα ανανεώνουμε). Ανάλογα με το ποιος παίχτης είμαστε (άσπρος ή μαύρος), αποφασίζουμε εάν είμαστε ο Max ή ο Min.

- Εάν είμαστε ο Max, δοκιμάζουμε το move το οποίο προσπελαύνουμε και καλούμε την συνάρτηση Minimax αναδρομικά με όρισματα depth-1 και όρισμα που αντιστοιχεί στον αντίπαλο. Το αποτέλεσμά του το αποθηκεύουμε στην μεταβλητή myMove.

Έπειτα ελέγχουμε αν αυτό το myMove έχει καλύτερο value από το εκάστοτε bestMove που έχουμε (ή εάν δεν έχουμε κάποιο bestMove ακόμα), και σε αυτήν την περίπτωση αναθέτουμε το καλύτερο αυτό move ως το νέο μας bestMove.

- Εάν είμαστε ο Min, πράττουμε το ίδιο με πριν με τη μόνη διαφορά ότι ελέγχουμε εάν το myMove έχει μικρότερο (αντί για μεγαλύτερο) value από το εκάστοτε bestMove.

- Τελικά η συνάρτηση Minimax επιστρέφει το bestMove αφού βγει από το while loop των Available και Legal Moves.

3) Alpha-Beta Pruning:

Ο Alpha-Beta Pruning είναι ένας αλγόριθμος ο οποίος μπορεί να εφαρμοστεί στον αλγόριθμο Minimax που περιγράφεται παραπάνω, έτσι ώστε να μειώσει τον αριθμό των κόμβων οι οποίοι αξιολογούνται από αυτόν.

Όσον αφορά την τεχνική υλοποίηση του Alpha-Beta Pruning, αυτή έγινε ως εξής εντός του Minimax αλγορίθμου:

- Αρχικά προσθέτουμε 2 ορίσματα στην συνάρτηση Minimax, το όρισμα Alpha και το όρισμα Beta, τα οποία αρχικά καλούνται με τιμές μέιον άπειρο και συν άπειρο αντίστοιχα(-Double.MAX_VALUE και Double.MAX_VALUE).

Έπειτα στο while loop των Available και Legal Moves, για κάθε move:

- Στην περίπτωση που είμαστε ο Max, με μια IF ελέγχουμε εάν το σκορ είναι μεγαλύτερο από το όρισμα Alpha, τότε αναθέτουμε στην μεταβλητή-όρισμα Alpha την τιμή του σκορ. Έπειτα, με μια ακόμα IF ελέγχουμε εάν το Alpha είναι μεγαλύτερο του Beta και σε αυτήν την περίπτωση κάνουμε break και return την κίνηση που αντιστοιχεί στο Alpha(και έμμεσα και το alpha, καθώς έχουμε αντικείμενο που αποθηκεύει το action και το value) (δηλαδή pruning).

- Στην περίπτωση που είμαστε ο Min, η λογική είναι η ίδια με διαφορετικές όμως συνθήκες. Η πρώτη IF ελέγχει εάν το σκορ είναι μικρότερο από το όρισμα Beta και σε αυτήν την περίπτωση αναθέτει στην μεταβλητή-όρισμα Beta την τιμή του σκορ. Στην δεύτερη IF ελέγχει πάλι έαν το Alpha είναι μεγαλύτερο του Beta και εφόσον ισχύει, κάνουμε break και return την κίνηση που αντιστοιχεί στο Beta (δηλαδή pruning).

- Σημειώνουμε ότι οι τιμές των Alpha και Beta, οι οποίες ξεκινάνε από μείον άπειρο και άπειρο αντίστοιχα, αλλάζουν διότι η συνάρτηση Minimax καλείται αναδρομικά όπως περιγράφουμε στην ενότητα 2 της αναφοράς “Αλγόριθμος αναζήτησης Minimax”. Επομένως, τα ορίσματα της συνάρτησης Minimax, το Alpha και το Beta, θα ανανεώνουν τις αντίστοιχες τιμές τους κάθε φορά.

4) Συνάρτηση αξιολόγησης:

Στην ενότητα 2 της αναφοράς “Αλγόριθμος αναζήτησης Minimax” αναφέρεται ο τρόπος με τον οποίον χρησιμοποιείται η συνάρτηση αξιολόγησης εντός του Minimax. Πρακτικά, η συνάρτηση αξιολόγησης υπολογίζει και επιστρέφει μια τιμή βάση κάποιων δεδομένων της εκάστοτε κατάστασης. Όσο μεγαλύτερη είναι η τιμή αυτή, τόσο πιο καλή και ευνοϊκή θα θεωρηθεί η κίνηση που δοκιμάστηκε από τον αλγόριθμο Minimax. Οι συναρτήσεις αξιολόγησης που δοκιμάσαμε περιγράφονται παρακάτω:

- V = (whitepawns-blackpawns)+(scorewhite-scoreblack)

Η συνάρτηση αυτή μας δόθηκε από την εκφώνηση. Η τιμή που επιστρέφει βασίζεται στην διαφορά των κομματιών μας έναντι αυτών του αντιπάλου προστιθέμενη με την διαφορά του score μας έναντι του αντιπάλου.

Εάν και φαίνεται αρκετά απλοική, είχε απρόσμενα καλό αποτέλεσμα στις δοκιμές μας. Παρόλα αυτά, ψάξαμε να βρούμε τρόπους να την κάνουμε καλύτερη. Την χρησιμοποιήσαμε, λοιπόν, ως μέτρο σύγκρισης έναντι άλλων συναρτήσεων αξιολόγησης που σκεφτήκαμε.

- V = 3*(WRooks-BRooks)+(WPawns-BPawns)+(scorewhite-scoreblack)

Η συνάρτηση αυτή είναι μια παραλλαγή της παραπάνω. Η διαφορά είναι ότι αντιμετωπίζει κάθε κομμάτι σύμφωνα με την αξία του μέσω του σπασίματος του συνόλου των κομματιών σε επιμέρους κατηγορίες . Επίσης κάνουμε χρήση ενός συντελεστή στην καθεμία κατηγορία ανάλογα με την αξία της.

Η αλλαγή αυτή αποδείχθηκε επιτυχημένη, αφού δοκιμάζοντας αυτήν την συνάρτηση αξιολόγησης έναντι αυτής της εκφώνησης, παρατηρήσαμε ότι στα περισσότερα (όχι όμως σε όλα) τα παιχνίδια είναι νικηφόρα.

If(!GamehasEnded)

- V = 3*(WRooks-BRooks)+(WPawns-BPawns)+(scorewhite-scoreblack)

Else

-V=10*(WKing-BKing)+scorewhite-scoreblack

Η συνάρτηση αυτή είναι μια παραλλαγή της προηγούμενης. Η μόνη αλλαγή είναι ότι προσθέσαμε μια IF η οποία ελέγχει εάν το παιχνίδι τελειώνει, δηλαδή εάν πρόκειται να αιχμαλωτιστεί κάποιος βασιλιάς. Σε αυτήν την περίπτωση, επιστρέφουμε το τελικό score μαζί με το 10*(WKing-BKing) ως αποτέλεσμα της συνάρτησης αξιολόγησης.

Με αυτήν την τροποποίηση, ο πράκτοράς μας λαμβάνει περισσότερο υπόψην τόσο τον δικό μας όσο και τον αντίπαλο βασιλιά. Ορισμένες φορές μπορεί να γίνει ιδιαίτερα επιθετικός, προσπαθώντας να αρπάξει την νίκη επιτιθέμενος στον αντίπαλο βασιλιά εφόσον δει ότι το τελικό score θα είναι θετικό. Άλλες φορές, θα παίξει αμυντικά όταν μειονεκτούμε και θα προστατέψει τον βασιλιά μας όταν δει ότι το τελικό score θα είναι αρνητικό. Τέλος, υπάρχουν και φορές (αν και όχι συχνά) που ο βασιλιάς μας θα θυσιαστεί επίτηδες εάν το τελικό score είναι θετικό, αφού μας ενδιαφέρει το τελικό score μόνο και η απώλεια του βασιλιά (αν και ιδιαίτερα μειονεκτική από άποψη πόντων) δεν συνεπάγεται με ήττα.

If(!GamehasEnded)

- V = 3*(WRooks-BRooks)+(WPawns-BPawns)+(scorewhite-scoreblack)+0.1*center_control

Else

- V = 10*(WKing-BKing)+scorewhite-scoreblack

Στην συνάρτηση αυτή προσθέσμαμε μία ακόμα σημαντική μέθοδο, την Center Control πολλαπλασιασμένη με έναν συντελεστή. Η μέθοδος αυτή δίνει αξία στο πόσο μπροστά είναι τα απλά πιόνια μας αφού ελέγχει από την 2η σειρά του παίχτη μας μέχρι την μέση της σκακιέρας (διότι δεν θέλουμε να κάνουμε την μέθοδο άκρως επιθετική και να ρισκάρουμε να χάσουμε τα πιόνια μας). Ανάλογα με το πόσο μπροστά είναι το κάθε απλό πιόνι μας, τόσο μεγαλύτερη τιμή θα επιστρέψει και η μέθοδος Center Control, επομένως και γενικότερα η συνάρτηση αξιολόγησής μας.

Η αλλαγή αυτή βοήθησε ιδιαίτερα στο να διασφαλίσουμε αρκετό χώρο σε σχέση με τον αντίπαλο, έτσι ώστε να έχουμε περισσότερο mobility άρα και περισσότερες επιλογές κινήσεων. Επιπλέον, περισσότερος χώρος σημαίνει μεγαλύτερη πιθανότητα απόκτησης δώρων από τον αντίπαλο που είναι ένας σημαντικός παράγοντας για την νίκη μας.

Οι 2 πρώτες κινήσεις μας είναι προκαθορισμένες. Κινούμε τα δύο πιόνια που βρίσκοντα μπροστά από τους πύργους.

Η παραπάνω τροποποίηση σε συνδυασμό με την συνάρτηση αξιολόγισης αποτελεί την τελική τακτική μας. Διαμορφώθηκε με ένα σκεπτικό που περιγράφει πολύ καλά ο παγκόσμιος πρωταθλητής του σκάκι Bobby Fischer με το εξής quote: “Στο σκάκι έχει τόσο μεγάλη σημασία το opening theory που μπορεί να κάνει κάποιον με καλή γνώση αυτού να νικήσει έναν πρωταθλητή που το αμελεί”.

Βάση αυτής της λογικής, κάναμε ένα πολύ απλό αλλά δυνατό opening. Οι δύο πρώτες κινήσεις μας είναι να μετακινήσουμε το 2ο και το 4ο πιόνι μας κατά μία θέση μπροστά αντίστοιχα. Με αυτόν τον τρόπο, επιτυγχάνουμε κατοχύρωση του κέντρου γρηγορότερα ή σπάμε το κέντρο εάν αντιμετωπίσουμε επιθετικό αντίπαλο. Το σκεπτικό μας ήταν ότι με δύο κινήσεις (2ο πιόνι μπροστά, 4ο πιόνι μπροστά) έχουμε την δυνατότητα να δεσμεύσουμε 3 τετράγωνα (αυτά μπροστά και διαγώνια των πιονιών μας, δηλαδή το 1ο, το 3ο και το 5ο της επόμενης σειράς).

Το opening αυτό βοήθησε σε σημαντικό βαθμό τον πράκτορά μας βάση των δοκιμών που κάναμε, αφού βοηθάει στον γενικότερο έλεγχο του κέντρου της σκακιέρας, γεγονός το οποίο μας προσφέρει μεγαλύτερο mobility και μεγαλύτερη πιθανότητα απόκτησης δώρων, παράγοντες καθοριστικούς για την νικηφόρα έκβαση του παιχνιδιού.

Επιπλέον, δοκιμάσαμε και άλλες τακτικές έτσι ώστε να βελτιώσουμε την συνάρτηση αξιολόγησής μας. Αυτές ήταν το King Safety, Pawn Structure, το Game Type και το Open File και περιγράφονται παρακάτω:

- Το King Safety προέτρεπε τα πιόνια μπροστά από τον βασιλιά και τον ίδιο τον βασιλιά να παίξουν αμυντικά στο early και στο middle game. Στο late game, προέτρεπε τα πιόνια και τον βασιλιά να παίξουν επιθετικά και μάλιστα έδινε κάποιο penalty όταν ο βασιλιάς ήταν σε κάποια γωνιακή θέση με το σκεπτικό ότι όσο περισσότερες επιτρεπτές κινήσεις έχει ο βασιλιάς, τόσο μικρότερη η πιθανότητα να αιχμαλωτιστεί. Παρόλα αυτά, η πολυπλοκότητα της τακτικής αυτής σε θέμα μνήμης (memory) καθώς και η δυσκολία οριοθέτησης critical point για το πότε ξεκινάει το late game μας έκανε να την απορρίψουμε.

- Το Pawn Structure ήταν ένα lookup table που έδινε διαφορετική αξία στα απλά πιόνια σε σχέση με την θέση τους στην σκακιέρα. Αντικαταστάθηκε από την καλύτερη βάση δοκιμών τακτική Center Control που περιγράφηκε παραπάνω.

- Το Game_Type έπαιρνε 3 τιμές ανάλογα με το εάν είμαστε στο early game, middle game ή late game. Τα όρια καθορίζονταν σε σχέση με το πόσα απλά πιόνια έχουν μείνει στο παιχνίδι. Το Game_Type χρησιμοποιήθηκε ως συντελεστής σε άλλες τακτικές που άλλαζαν δυναμικά κατά την διάρκεια του παιχνιδιού. Οι δοκιμές, όμως, έδειξαν ότι ένας τέτοιος συντελεστής υπονομεύει το early game μας στην διάρκεια του οποίου κρίνονται πολλά. Επομένως, η τροποποίηση αυτή απορρίφθηκε.

- Το Open_File ήταν ένας μετρητής που έπαιρνε την τιμή των στηλών οι οποίες δεν είχαν κανένα απλό πιόνι. Έπειτα, χρησιμοποιήθηκε ως συντελεστής για να δώσει δύναμη στους πύργους αφού θεωρητικά όσο περισσότερα Open Files έχουμε, τόσο μεγαλύτερη δύναμη έχει ο πύργος. Παρόλα αυτά, η ύπαρξη και η τυχαιότητα των δώρων καθώς και το γεγονός ότι ο πύργος δεν μπορεί να υπερπηδήσει τα δώρα, αποτέλεσαν παράγοντα αναστολής της συγκεκριμένης τροποποίησης.

5) Advanced search:

Η αναζήτηση MiniMax περιορίζεται σε βάθος 4 κινήσεων. Αυτό δημιουργεί το horizon effect, δηλαδή δεν μπορούμε να δούμε τι θα γίνει παρακάτω και κρίνουμε μόνο βάσει της αξιολόγησης που θα δώσουν τα φύλλα του δέντρου. Ο agent μας, σε κάθε αναζήτηση, πραγματοποιεί Quiescence Search.

Η αναζήτηση Quiescence αυξάνει το βάθος του δέντρου σε κινήσεις που δεν είναι “ήρεμες”. Ουσιαστικά, επεκτείνουμε το βάθος του δέντρου, μέχρι να φτάσουμε σε κατάσταση παιχνιδιού που να μην μπορεί κάποιος παίχτης να εχμαλωτίσει πεσσό του αντιπάλου. Με αυτό τον τρόπο μπορούμε να δούμε εάν όντως μας ωφελεί ή όχι η αιχμαλώτιση ενός πεσσού. Για να είμαστε μέσα σε κάποια χρονικά όρια, το Quiescence κάνει αναζήτηση σε βάθος 5 κινήσεων από κάποιο φύλλο του δέντρου. Δηλαδή, αθροιστικά, μπορούμε να εξετάσουμε μέχρι και σε βάθος 9 κινήσεων. Σε συγκεκριμένες περιπτώσεις, όταν υπάρχουν αρκετές συγκρούσεις στο παιχνίδι, ο παίκτης μας μπορεί να ξεπεράσει το όριο των 4 δευτερολέπτων(μερικά δευτερόλεπτα παραπάνω, και κυρίως στο early και middle game), αλλά βασιζόμενοι στο γεγονός ότι το όριο αυτό δεν είναι αυστηρό, κρατήσαμε το συγκεκριμένο βάθος. Σε περίπτωση που υπάρχει πρόβλημα χρόνου(πχ λόγω διαφορά υπολογιστικής ισχύος), η λύση είναι να μειωθεί το βάθος αναζήτησης του Quiescence από 5 σε 4(Η κλήση του γίνεται στις πρώτες γραμμές κώδικα της μεθόδου MiniMax).

Τέλος, σημειώνουμε ότι εξετάσαμε και άλλους advanced search αλγορίθμους αλλά αποφασίσαμε να χρησιμοποιήσουμε μόνο αυτόν ως χρυσή τομή βέλτιστης επιλογής (καλά και σίγουρα αποτελέσματα) και εξοικονόμησης μνήμης.