ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

[PLH-311]-Τεχνητή Νοημοσύνη

Αναφορά 2ης
Προγραμματιστικής άσκησης

Ομάδα χρηστών 28:

Ιωάννης Περίδης

A.M. 2018030069

Γεώργιος Σκουλάς

A.M. 2018030148

Διδάσκων:

Γεώργιος Χαλκιαδάκης

30 Μαΐου 2022

1) Εισαγωγή & Σκοπός:

Σκοπός της δεύτερης εργαστηριακής άσκησης, ήταν η υλοποίηση ενός ανταγωνιστικού και στρατηγικού πράκτορα, που θα προσπαθεί να κερδίσει τον αντίπαλο του στο παιχνίδι TUC-chess. Το TUC-chess, αποτελεί μια παραλλαγή του σκάκι , φυσικά με κάποιες αλλαγές στους κανονισμούς οι οποίες απλοποιούν σε έναν βαθμό το παιχνίδι, για την διευκόλυνσή μας. Το ρομπότ που σχεδιάστηκε, λειτουργεί με δύο διαφορετικούς τρόπους. Αρχικά, χρησιμοποιεί τον αλγόριθμο Minimax και ύστερα τον Monte Carlo Tree Search. Οι αλγόριθμοι αυτοί , είναι και οι δύο βασισμένοι στην αναζήτηση υπό αντιπαλότητα, της καλύτερης πιθανής κίνησης, μέσα σε ένα συγκεκριμένο περιβάλλον (σκακιέρα) και έχουν στόχο να νικήσουν τον αντίπαλό παίκτη.

2) Ανάλυση Αλγόριθμων:

A) Minimax:

Η ιδέα της λειτουργίας του αλγορίθμου Minimax, έχει ως εξής. Βρισκόμενοι σε μια συγκεκριμένη κατάσταση , θέλουμε να αναλύσουμε όλα τα πιθανά σενάρια , στα οποία μπορούμε να οδηγηθούμε από αυτήν την κατάσταση και να επιλέξουμε το καλύτερο δυνατό για τον παίκτη μας. Για να γίνει αυτό, χρησιμοποιείται φυσικά ,μια συνάρτηση αξιολόγησης , η οποία προσδιορίζει την ποιότητα της τωρινής μας κατάστασης. Η αναζήτηση των σεναρίων, δηλαδή των μελλοντικών καταστάσεων, είναι εφικτό να γίνει σε οποιαδήποτε στιγμή μετά την σειρά του παίκτη μας. Παρόλα αυτά, είναι αναμενόμενο πως όσο πιο μετά γίνει, τόσο περισσότερες καταστάσεις θα πρέπει να ελεγχθούν.
Συνοψίζοντας, ο αλγόριθμος , επιλέγει την κίνηση με το μεγαλύτερο κέρδος για τον παίκτη μας (και το ελάχιστο για τον αντίπαλο), αφού έχει αξιολογήσει όλες τις καταστάσεις , μέσα σε ένα συγκεκριμένο βάθος.

Παρακάτω, φαίνεται η υλοποίηση του αλγορίθμου στον κώδικα και οι συναρτήσεις που χρησιμοποιήθηκαν. Ξεκινώντας, στο στιγμιότυπο του world δημιουργήθηκε μία μεταβλητή minimax, η οποία θα περιέχει όλες τις συναρτήσεις που δημιουργήθηκαν. Μέσα από το world θα καλεστεί η συνάρτηση selectMiniMax, η οποία αρχίζει την διαδικασία του αλγορίθμου. Αυτή θα κάνει τις κατάλληλες αρχικοποιήσεις και ανάλογα με μια boolean μεταβλητή επιλέγουμε αν θα κάνουμε a-b Pruning (δημιουργηθηκε αργότερα) ή Minimax. Ο αλγόριθμος του minimax καλεί αναδρομικά τον εαυτό του και έχει σαν συνθήκη τερματισμού είτε αν το βάθος είναι ίσο με 0 είτε αν ο κόμβος, στον οποίο βρισκόμαστε σηματοδοτεί το τέλος του παιχνιδιού(game Over). Στον αλγόριθμο του minimax δεν λήφθηκε υπόψη ο περιορισμός του χρόνου όπως θα γίνει στην συνέχεια στον αλγόριθμο του Monte Carlo. Όταν μία από τις 2 παραπάνω συνθήκες γίνει αληθής τότε καλείται μια συνάρτηση αξιολόγησης, η οποία θα υπολογίσει μία τιμή "κόστους" για τους κόμβους αυτούς. Εφόσον καμία από τις 2 παραπάνω συνθήκες δεν είναι αληθής τότε εκτελούνται προσωρινά όλες οι διαθέσιμες κινήσεις, οι οποίες θα φαίνονται σε ένα προσωρινό ταμπλό, το οποίο αποθηκεύεται στην μεταβλητή του Minimax tmpBoard. Όταν έχουν πραγματοποιηθεί όλες οι δυνατές κινήσεις τότε υπολογίζεται η βέλτιστη κίνηση. Πιο συγκεκριμένα αν πρόκειται για maximizing player θα γυρίσει την κίνηση που θα αυξήσει στο έπακρο το αποτέλεσμα της ευρετικής συνάρτησης, ενώ αν πρόκειται για minimizing player θα προσπαθήσει να μειώσει το αποτέλεσμα όσο το δυνατόν περισσότερο. Τέλος, αξίζει να σημειωθεί ότι στην υλοποίηση μας χρησιμοποιήθηκε μια κλάση Node, η οποία εμπεριέχει 2 μεταβλητές την κίνηση που έγινε και το "κόστος" της κίνησης αυτής

B) a-b Pruning :

Όπως γίνεται αντιληπτό, όσο στον αλγόριθμο Minimax,αυξάνεται το βάθος μετά από κάποιο σημείο το πλήθος των καταστάσεων που πρέπει να ελεγχθούν για να παρθεί απόφαση, αυξάνεται εξαιρετικά ραγδαία . Επομένως, δημιουργείται η ανάγκη εύρεσης ενός τρόπου να μειωθούν αυτές οι προς έλεγχο καταστάσεις, χωρίς όμως, να μειωθεί η αποδοτικότητα του αλγορίθμου. Την δουλεία αυτή ακριβώς ,, αναλαμβάνει ο αλγόριθμο a-b pruning, ο οποίος αποτελεί μια μέθοδο αποκοπής των περιττών καταστάσεων, κατά την αναζήτηση minimax. Συγκεκριμένα, γνωρίζοντας πως η αναζήτηση του Minimax, υλοποιείται σαν Depth First Search, ο a-b pruning, αποκόπτει όσο το δυνατόν μεγαλύτερα υποδέντρα αποφάσεων, τα οποία με σιγουριά , δεν θα οδηγούσαν τον παίκτη σε κάποια καλύτερη κατάσταση από αυτήν που βρίσκεται τώρα.

Η υλοποίηση του έχει ως εξής. Το a, συμβολίζει την καλύτερη τιμή αξιολόγησης του παίκτης μας , δηλαδή την μέγιστη τιμή που βρίσκεται κατά μήκος της διαδρομής μας. Αντίστοιχα, το b συμβολίζει, την καλύτερη τιμή αξιολόγησης του αντιπάλου μας, δηλαδή, την ελάχιστη αυτή τη φορά τιμή. Επομένως, κατά την αναζήτηση minimax, όταν έχει σειρά ο max και βρεθεί κατά την εκτέλεση κάποιο evaluation>a, τότε γίνεται ανανέωση του a και ύστερα γίνεται η συνθήκη ελέγχου b $\leq a$. Σε περίπτωση που η συνθήκη είναι αληθής, έχει βρεθεί στην αναζήτηση μια καλύτερη τιμή , σε κάποιο προγενέστερο κόμβο, σε προηγούμενο επίπεδο. Αυτό το συμπεραίνουμε διότι ο b , προέρχεται από κόμβο πατέρα και αποτελεί την καλύτερη τιμή έως τώρα για τον πατέρα. Επομένως, αν η καλύτερη τιμή του κόμβου(max), δηλαδή το a, είναι μεγαλύτερο του b, αυτό έχει ως αποτέλεσμα ο max, να έχει την καλύτερη δυνατή τιμή του, αφού $evaluation \geq a > b$. Τελικά λοιπόν, δεν έχει νόημα να εξερευνηθεί περεταίρω ο κόμβος max, αφού ο πατέρας min, στα σίγουρα δεν θα τον επέλεγε , αφού θα διάλεγε τον κόμβο που του έδωσε την τιμή β από την αρχή.

Μια βελτιστοποίηση που θα μπορούσε να γίνει στον a-b pruning, βασισμένη στην στοίχιση των κόμβων , είναι να τοποθετούνταν οι βέλτιστοι κόμβοι πρώτοι, πριν αρχίσει η εξερεύνηση. Αυτό θα είχε ως αποτέλεσμα οι κόμβοι με την καλύτερη πιθανότητα, να είναι οι βέλτιστοι και άρα θα εξερευνιούνται πρώτοι. Έτσι, οι τιμές για τους συντελεστές a-b θα βρίσκονταν γρήγορα και θα αποκόπτονταν μεγαλύτερα υποδέντρα, άρα θα αυξανόταν η ταχύτητα του αλγορίθμου.

C) Monte Carlo Tree Search:

Συνεχίζοντας, με την δεύτερη λύση στο πρόβλημα μας, θα εξεταστεί ο αλγόριθμος Monte Carlo TS. Ο αλγόριθμος αυτός, σε αντίθεση με τον Minimax, χρησιμοποιεί μια πιο καθολική αντιμετώπιση του προβλήματος , βασιζόμενη στον υπολογισμό κάθε στιγμή , της καλύτερης δυνατής κίνησης για τον παίκτη μας. Είναι μια πιθανοτική μέθοδος, που χρησιμοποιεί στατιστικά για την λήψη αποφάσεων και όχι περίτεχνες ευρεστικές συναρτήσεις αξιολόγησης και υπολογισμούς.Ο Monte Carlo, είναι χωρισμένος σε τέσσερα βασικά βήματα και λειτουργεί σε γενικές γραμμές, δοκιμάζοντας πάρα πολλές (χιλιάδες) φορές τυχαίες προσομοιώσεις, καταγράφοντας παράλληλα κάποια αποτελέσματα για τις πιθανότητές νίκης των μονοπατιών.

ΒΗΜΑ 1): Select
Το βήμα αυτό , υλοποιήθηκε κάνοντας την στρατηγική UCB1. Αναλύοντας,
χρησιμοποιήθηκε αλλαγμένη η $\mathrm{UCT} = \mathrm{Vi} + 2\,\mathrm{C}\sqrt{\frac{\ln \mathrm{N}}{n_i}}, με\ C = \frac{1}{\sqrt{2}}\ , Vi \in [-1,1]$

Όπου Vi= μέσο κέρδος της κατάστασης i
N= πλήθος των συνολικών επισκέψεων
ni= πλήθος επισκέψεων της κατάστασης i.

Επομένως, επιλέχθηκε κάθε φορά ο κόμβος με την καλύτερη τιμή UCB1. Γίνεται αντιληπτό πως στην πρώτη επανάληψη , θα έχω $\mathrm{UCB} = \infty,\ αφού\ ni = 0.$

ΒΗΜΑ 2): Expand
Το στάδιο αυτό, εισερχόμαστε μονάχα στις περιπτώσεις που η select (βήμα 2), επιστρέφει έναν κόμβο με ni=0, δηλαδή μόνο σε περίπτωση που έχει τρέξει τουλάχιστον 1 επανάληψη από όλους τους κόμβους .Επομένως, αν υπάρχουνε διαθέσιμες κινήσεις στην τωρινή μας κατάσταση και δεν βρισκόμαστε σε τερματικό κόμβο, τότε πρέπει να εφαρμόσουμε κάθε μια κίνηση από αυτές .Αποτέλεσμα αυτού, είναι να δημιουργήσουμε όλες τις νόμιμες κινήσεις από τον τωρινό κόμβο, δηλαδή όλα τα παιδιά boards. Εφόσον υλοποιηθεί το expand, τότε σαν νέος current κόμβος προς αναζήτηση , ορίζεται ένα τυχαίο από τα παιδιά που μόλις επεκτάθηκαν. Σε περίπτωση που δεν υλοποιήθηκε το expand, μεταφερόμαστε στην επόμενη φάση και έχουμε σαν κόμβο, αυτόν που δίνει σαν αποτέλεσμα η select.

ΒΗΜΑ 3): Rollout
Στο βήμα αυτό, πραγματοποιείται ένα τυχαίο simulation από την κατάσταση του current κόμβου, μέχρι κάποια τερματική κατάσταση. Όταν λοιπόν φτάνουμε στην τερματική την αξιολογούμε ανάλογα το αποτέλεσμά της. Επιστρέφουμε -1 αν δίνει ήττα, 0 αν δίνει ισοπαλία και +1 αν δίνει νίκη.

ΒΗΜΑ 4): Backpropagate
Αυτό, είναι το τελικό στάδιο. Ξεκινάμε από τον κόμβο που έγινε η αρχή της προσομοίωσης και γίνονται ανανεώσεις στις επισκέψεις και στο συνολικό έπαθλο στην διαδρομή από τον κόμβο αυτό έως την ρίζα του δέντρου. Κατ ‘ αυτόν τον τρόπο, ενημερώνονται όλοι οι κόμβοι για το αποτέλεσμα του rollout (βήμα 3).

3) Ευρετικές & Cutoff Search:

Όπως αναφέρθηκε και προηγουμένως, για τον παραπάνω αλγόριθμο και για την μέθοδο αποκοπής καταστάσεων, χρησιμοποιήθηκε κάποια συνάρτηση αξιολόγησης , η οποία ήταν υπεύθυνη να κρίνει ποιες θα είναι οι περισσότερο και ποιες οι λιγότερο επιθυμητές καταστάσεις για να βρίσκεται ο παίκτης μας.

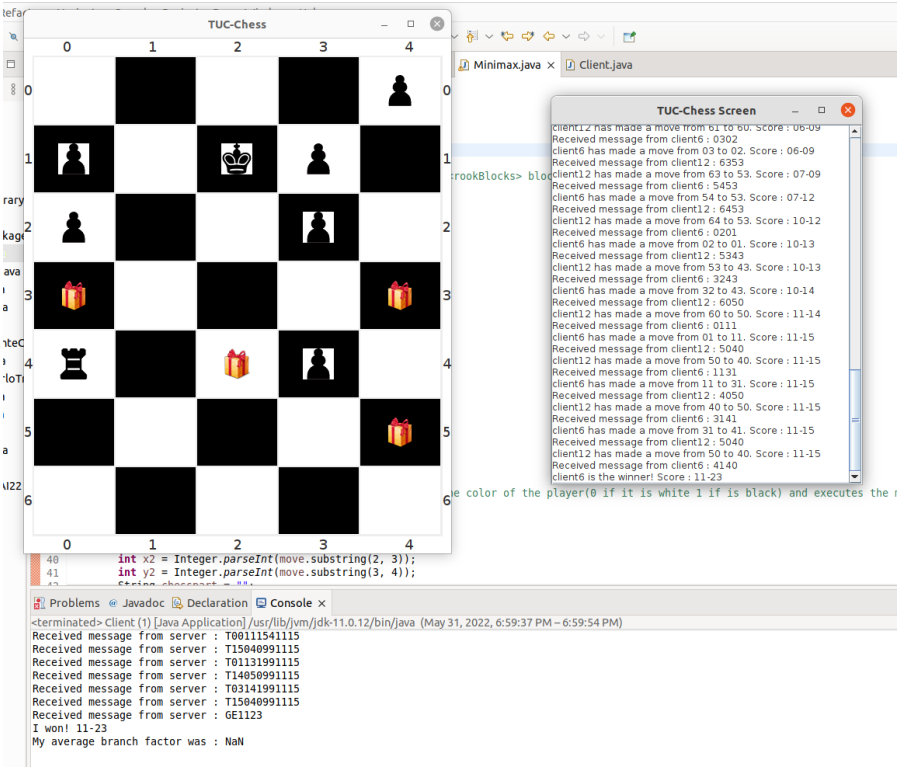
Ο τύπος της συνάρτησης αξιολόγησης που χρησιμοποιήθηκε, είναι εκείνος της εκφώνησης:
$V(state) = |my(points + pieces)| - |opponent's(points + pieces)|$

Αναλύοντας, η κατάσταση κρίνεται από την διαφορά των απολύτων των πόντων και των πιονιών του παίκτη μας, αφαιρώντας τους πόντους και τα πιόνια του αντιπάλου.
Η δοθείσα συνάρτηση, είναι αρκετά καλή όσον αφορά την μεγιστοποίηση του σκορ και το προβάδισμα στα πιόνια, το οποίο είναι χρήσιμο όσο προχωράει το παιχνίδι , διότι με παραπάνω στρατιώτες, έχεις και παραπάνω επιλογές. Αυτό συμβαίνει διότι κάθε κίνησή της, είναι βασισμένη στην ικανοποίηση αυτών των δύο παραγόντων. Παρόλα αυτά, δεν σημαίνει πως δεν υπάρχουν καλύτερες , πιο δημιουργικές και πιο σύνθετες συναρτήσεις , οι οποίες να συνδυάζουν και δυναμικά κριτήρια απόφασης τα οποία θα αλλάζουν κατά την διάρκεια του παιχνιδιού και αναλόγως σε ποια φάση βρίσκεται ο παίκτης.

Η συνάρτηση αυτή, χρησιμοποιήθηκε όσο στο Minimax για την επιλογή των καταστάσεων, όσο και στον a-b pruning, για τον καθορισμό των κόμβων που θα εξεταστούν παραπάνω και των κόμβων που θα αποκοπούν και θα αγνοηθούν.

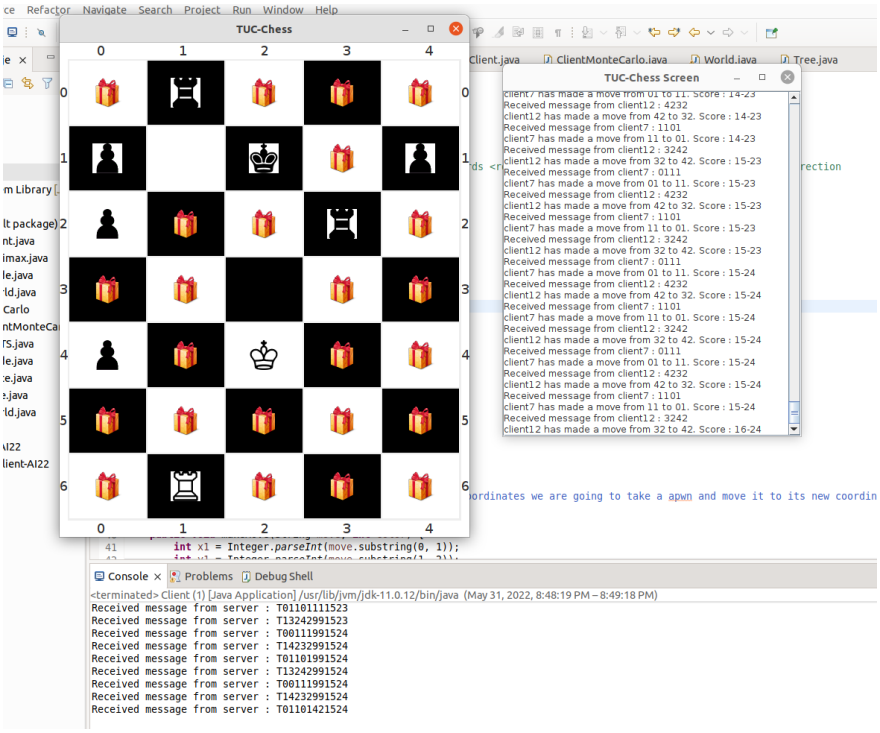
4) Αποτελέσματα Πειραμάτων:

Παρακάτω παρατίθονται εικόνες από κάποιες προσομοιώσεις που έγιναν με σκοπό την σύγκριση των αλγορίθμων και την διασαφήνιση στην ερώτηση ποιος αλγόριθμος είναι αποδοτικότερος. Όπως είναι φυσικό όλες οι υλοποιήσεις κερδίζουν τον παίχτη που κάνει τυχαίες επιλογές. Επίσης, έγινε η σύγκριση ανάμεσα σε 2 παίχτες που χρησιμοποιούν τον αλγόριθμο του Minimax, με την διαφορά ότι ο ένας από τους δύο θα παίρνει σαν είσοδο μεγαλύτερο βάθος. Σε αυτή την περίπτωση ο παίχτης με το μεγαλύτερο βάθος επειδή ψάχνει βαθύτερα στο δέντρο αποφάσεων παίρνει καλύτερες αποφάσεις όπως φαίνεται και στο παρακάτω παράδειγμα



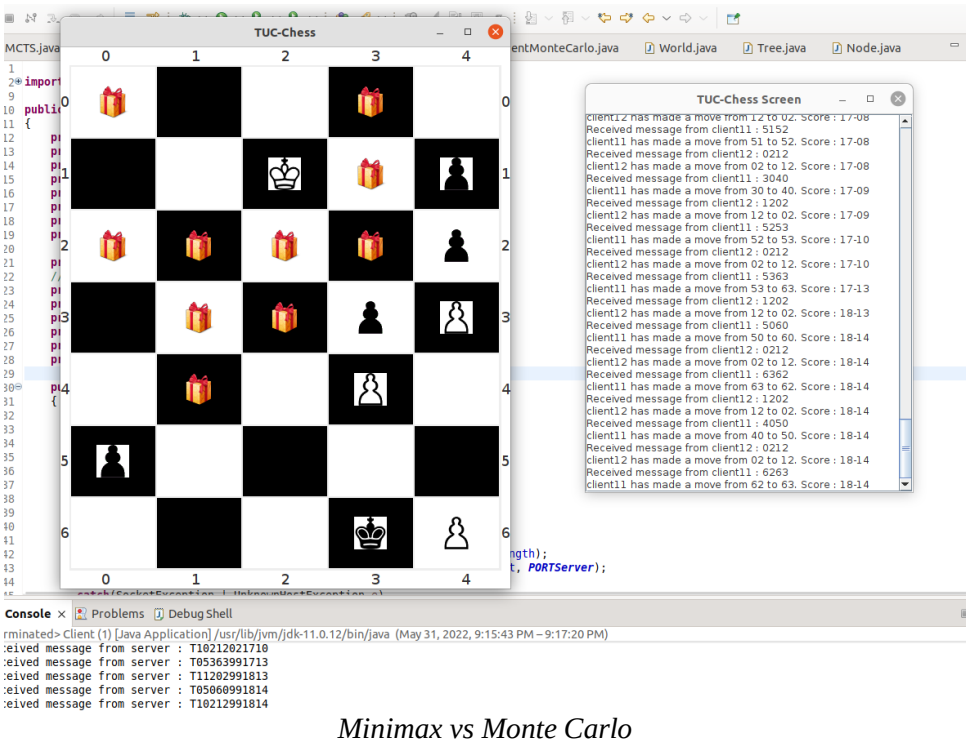
Με μαύρα είναι ο νικητής μας, ο οποίος χρησιμοποιούσε μεγαλύτερο βάθος Όταν συγκρίθηκαν οι Minimax και a-b pruning οι απόψεις διίστανται. Στις προσομοιώσεις που έγιναν σε κάποιες

περιπτώσεις κέρδιζε ο Minimax και σε άλλες κέρδιζε ο a-b Pruning. Μάλιστα κάποιες φορές(πολύ σπάνια) μπορεί να αργούσε πάρα πολύ να βρεθεί νικητής. Στις περισσότερες περιπτώσεις νικούσε ο a-b pruning



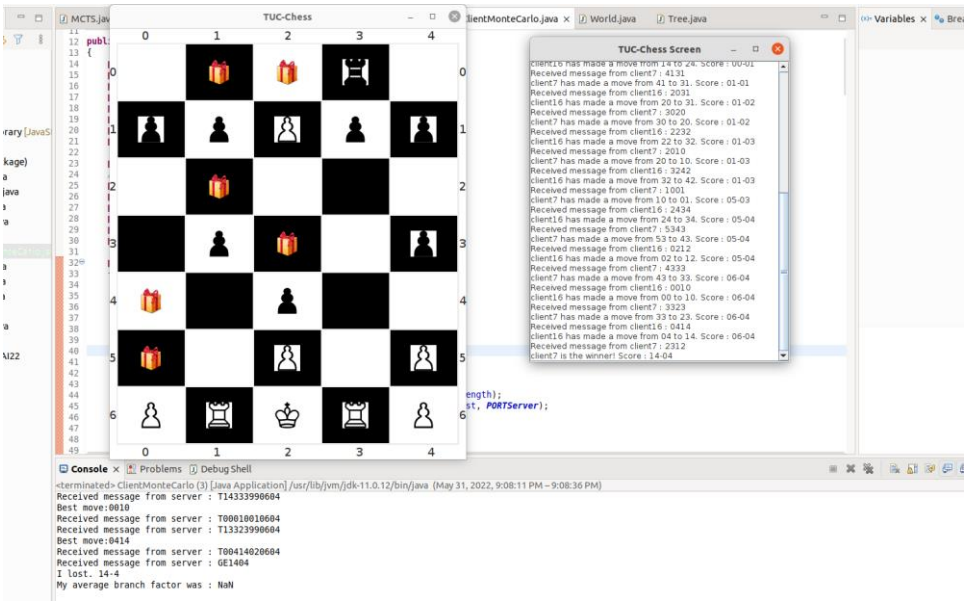
Περίπτωση που αργεί να βρεθεί το αποτέλεσμα

Τέλος, στις προσομοιώσεις που έγιναν ανάμεσα σε παίχτη που χρησιμοποιούσε Monte Carlo με αντίπαλο παίχτη που χρησιμοποιεί minimax ή a-b pruning συνήθως νικήτης αναδεικνυόταν ο δεύτερος(Minimax, a-b Pruning):



Minimax vs Monte Carlo

Νικητής στην παραπάνω προσομοίωση ήταν ο άσπρος, ο οποίος αναπαριστά τον minimax. Τέλος, θα παρουσιαστεί η προσομοίωση που έγινε ανάμεσα στον monte carlo και τον a-b Pruning.



A-B Pruning vs Monte Carlo

5) Σύγκριση και Συμπεράσματα:

Όπως αναμένεται από θεωρία επειδή η υλοποίηση μας χρησιμοποίει συγκεκριμένο αριθμό βάθους στην αναζήτηση και όχι περιορισμό χρόνου η υλοποίηση του Minimax με αυτή του a-b Pruning είναι εξίσου καλές, καθώς καταλήγουν στο ίδιο αποτέλεσμα, δηλαδή θα εκτελείται η ίδια κίνηση αν βρεθούν στην ίδια κατάσταση. Πρέπει να σχολιαστεί όμως ότι ο αλγόριθμος του a-b Pruning βρίσκει την νέα κίνηση γρηγορότερα από τον Minimax, καθώς θα αποκοπούν κάποια κομμάτια του δέντρου όπως έχει προαναφερθεί. Αν ληφθεί αυτό υπόψη προτιμάται η τροποποίηση του a-b Pruning από τον Minimax. Επίσης, αν λαμβανόταν υπόψη και ο χρόνος σαν περιορισμός τότε η υλοποίηση του a-b Pruning θα έφτανε σε μεγαλύτερο βάθος στο δέντρο αποτελεσμάτων, οπότε θα έβρισκε και καλύτερα αποτελέσματα.
Θεωρητικά ο αλγόριθμος του monte carlo είναι καλύτερος σε σχέση με τους άλλους αλγορίθμους, εφόσον η ευρετική συνάρτηση που έχουμε δεν είναι πολύ εύστοχη. Επίσης, ο αλγόριθμος αυτός μπορεί να τρέχει έως ότου να τελειώσει ο επιτρεπόμενος χρόνος, σε αντίθεση με τον minimax, ο οποίος τρέχει για συγκεκριμένο βάθος. Πλεονέκτημα του Minimax είναι ότι τρέχει πολύ μικρό αριθμό επαναλήψεων σε σχέση με τον branching factor. Στο παρακάτω πινακάκι παρατίθονται τα branching factor όλων των αλγορίθμων που υλοποιήθηκαν:

	Random	Minimax	a-b Pruning	MCTS
Branch Factor	7,6	6,4	6,16	8,45

6) Μελλοντικές Βελτιώσεις:

Υπήρχαν διάφορες ιδέες για βελτιώσεις, οι οποίες λόγω περιορισμένου χρόνου δεν κατάφεραν να υλοποιηθούν. Για αρχή θα μπορούσε να βρεθεί μια καλύτερη ευρετική συνάρτηση, η οποία πιστεύω θα βελτίωνε την απόδοση των αλγορίθμων. Επίσης, σαν βελτίωση θα μπορούσαμε

όταν κατά την αναζήτηση βρισκόμασταν σε κάποιο τερματικό κόμβο(κατάσταση στην οποία τελειώνει το παιχνίδι), ο οποίος οδηγεί σε νίκη τον δικό μας παίχτη(maximizing player) θα μπορούσαμε να αυξήσουμε το βάρος του κόμβου αυτού. Σε αντίθεση αν ο κόμβος αυτός οδηγεί σε ήττα του παίχτη μας θα μπορούσαμε μειώσουμε κατά πολύ την τιμή της ευρετικής, έτσι ώστε να μην καταλήξουμε ποτέ σε αυτή την κατάσταση. Τέλος, θα μπορούσε να ταξινομήσουμε από πριν τις κινήσεις(pre sort) παίρνοντας τις καλύτερες κινήσεις. Ειδικότερα στον a-b pruning η βελτίωση αυτή θα βοηθούσε πάρα πολύ, καθώς θα υπάρχει μεγαλύτερη πιθανότητα να κοπούν κάποιοι κόμβοι κατά την διάρκεια του παιχνιδιού, αφού ο αλγόριθμος θα αρχίζει με μεγαλύτερο α και μικρό β στην αρχή του κάθε επιπέδου

7) Βιβλιογραφία:

https://ai.stackexchange.com/questions/7159/how-do-i-choose-the-best-algorithm-for-a-board-game-like-checkers
https://www.baeldung.com/java-monte-carlo-tree-search
https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/
https://levelup.gitconnected.com/improving-minimax-performance-fc82bc337dfd
Περσινές Διαλέξεις