Τεχνητή Νοημοσύνη

2η Προγραμματιστική Εργασία

Φοιτητής : Δανόπουλος Κωνσταντίνος

Αριθμός Μητρώου : 2016030131

Παίζοντας το Παιχνίδι TUC-ANTS

Ο στόχος της δεύτερης προγραμματιστικής εργασίας ήταν ο πειραματισμός με το παιχνίδι TUC-ANTS και η υλοποίηση ενός πράκτορα, ο οποίος να έχει την ικανότητα να παίζει αυτό το παιχνίδι με την καλύτερη δυνατή στρατηγική. Σύμφωνα με την εκφώνηση, ζητήθηκε ο πράκτορας να αποφασίζει τις επόμενες κινήσεις του με τις μεθόδους αναζήτησης minimax, καθώς και την α-β pruning. Αυτές οι δύο μέθοδοι αναζήτησης έχουν ακριβώς την ίδια λογική και υλοποίηση με την διαφορά ότι η μέθοδος α-β pruning διαθέτει δύο μεταβλητές επιπλέον όπου αποθηκεύουν τις προηγούμενες μέγιστες και ελάχιστες τιμές ώστε όπου δεν είναι απαραίτητο να μην γίνει σπάταλη διάσχιση κόμβου και αξιολόγηση του. Με αυτόν τον τρόπο, ουσιαστικά η μέθοδος α-β pruning εξοικονομεί χρόνο και επεξεργαστική ισχύ από τον πράκτορα και προτιμάται έναντι της minimax, εφόσον δίνει τα ίδια αποτελέσματα αλλά σε καλύτερο χρόνο.

Το παιχνίδι TUC-ANTS είναι υλοποιημένο με λογική επικοινωνίας client – server, του οποίου ο κώδικας δόθηκε. Η υλοποίηση του πράκτορα ουσιαστικά βρίσκεται εξολοκλήρου μέσα στο αρχείο client.c με εξαίρεση την δημιουργία ενός struct moves και βρίσκεται στο αρχείο move.h. Στο αρχείο client.c υλοποιήθηκαν οι εξής βασικές συναρτήσεις :

- int minimax(state * possition,int depth,int maximazingPlayer,int score)
- int alphaBeta(state * possition,int depth,int alpha,int beta,int maximazingPlayer,int score)
- int stateGenerattion()
- int canIDieInTheNextRound(int new_i,int new_j,char currentColor,char currentEnemyColor,char curBoard[BOARD_ROWS][BOARD_COLUMNS])
- void createStatesFromList(state * curState,list * firstNodeOfList,int doIHaveCaptureMoves,char color)
- function * findAllPossibleMovesForThisRoundAndAddToTheList(char color, state * currentState)

- int canCapture(char row, char col, char player, char board[BOARD_ROWS][BOARD_COLUMNS])

- moves findMyMaximumMoves(int i ,int j ,moves myMove2,int numberOfStep,char color)

- void addASonToThisState(state * myState,state * sonState)

Ο πράκτορας προκειμένου να επιλέξει την επόμενη του κίνηση με βάση την συνάρτηση minimax ή α-β pruning δημιουργήθηκε η συνάρτηση int stateGenerattion(). Η συνάρτηση αυτή εκτελείτε κάθε φορά που ο server ζητάει από τον πράκτορα να αποφασίσει και να του στείλει μια κίνηση. Ο στόχος είναι να δημιουργηθούν όλες οι πιθανές καταστάσεις με τις κινήσεις που μπορούν να συμβούν στο μέλλον. Η συνάρτηση αυτή κατασκευάζει όλες τις πιθανές καταστάσεις με βάθος πέντε μελλοντικών κινήσεων εναλλάξ του πράκτορα και του αντιπάλου. Αρχικά, αποθηκεύει την τωρινή κατάσταση του παιχνιδιού από τη μεταβλητή gamePosition που υπάρχει μέσα στον client και κατασκευάζει όλες τις πιθανές καταστάσεις που μπορούν να προκύψουν με κάποια κίνηση του πράκτορα και τις ορίζει ως παιδιά της τωρινής κατάστασης (head). Στην συνέχεια, πηγαίνει σε κάθε μια από αυτές τις καταστάσεις και δημιουργεί όλες τις πιθανές κινήσεις του αντιπάλου και τις ορίζει ως παιδιά της συγκεκριμένης κατάστασης. Η επόμενη κίνηση είναι του πράκτορα πάλι. Όλη αυτή η διαδικασία πραγματοποιείτε με βάθος πέντε κινήσεων με πρώτη κίνηση αυτή του πράκτορα. Στην περίπτωση που δεν υπάρχουν πέντε πιθανές κινήσεις, είτε εξαιτίας ότι δεν διαθέτει κίνηση ο πράκτορας είτε ο αντίπαλος του, τότε σταματάει να εκτελείτε η συνάρτηση για το συγκεκριμένο βάθος και μετά. Έτσι επιστρέφει το νούμερο του βάθους που κατάφερε να εκτελέσει με επιτυχία. Με σκοπό την απλότητα του κώδικα και την ευανάγνωσή του έχουν δημιουργηθεί επιμέρους συναρτήσεις που χρησιμοποιούνται μέσα στην συνάρτηση stateGeneration(). Οι συναρτήσεις αυτές είναι οι findAllPossibleMovesForThisRoundAndAddToTheList() και η createStatesFromList().

Η συνάρτηση findAllPossibleMovesForThisRoundAndAddToTheList() ουσιαστικά δημιουργήθηκε για να γνωρίζουμε αν από όλες τις πιθανές κινήσεις υπάρχουν κάποιες ή κάποια κίνηση αιχμαλώτισης ώστε να της δοθεί προτεραιότητα. Βρίσκει όλες τις πιθανές κινήσεις που μπορούν να γίνουν και τις αποθηκεύει σε μια λίστα

και αν συναντήσει κάποια κίνηση αιχμαλώτισης η μεταβλητή doIHaveCaptureMoves γίνετε ίση με ένα.

Η συνάρτηση createStatesFromList() δέχεται ως είσοδο τον πρώτο κόμβο της λίστας που έχει δημιουργηθεί από την συνάρτηση findAllPossibleMovesForThisRoundAndAddToTheList(). Αν υπάρχουν κινήσεις αιχμαλώτισης τότε επισκέπτεται όλες αυτές τις κινήσεις και δημιουργεί ένα state για την κάθε μία. Αν δεν υπάρχουν κινήσεις αιχμαλώτισης τότε επισκέπτεται όλους τους κόμβους της λίστας και δημιουργεί ένα state για το κάθε ένα. Επίσης όταν προσπερνάει οριστικά ένα κόμβο που δεν θα τον ξανά χρησιμοποιήσει τότε το κάνει free(). Το struct state ουσιαστικά περιέχει μέσα τον board της συγκεκριμένης κατάστασης , το score που μας προσφέρει αυτή η κατάσταση , το tile το οποίο είναι η κίνηση που θα σταλεί στον server σε περίπτωση επίσκεψης αυτής της κατάστασης και τέλος την λίστα με τα παιδιά του συγκεκριμένου κόμβου και την λίστα που βρίσκεται η συγκεκριμένη κατάσταση σε περίπτωση που έχει πατέρα κάποιο άλλο state. Στην περίπτωση που έχουμε κάποια κίνηση αιχμαλώτισης, τότε η συνάρτηση createStatesFromList() γνωρίζει ότι μπορεί να αιχμαλωτιστεί τουλάχιστον ένα από τα αντίπαλα μυρμήγκια όμως δεν ξέρει την καλύτερη κίνηση που μπορεί να αιχμαλωτίσει όσο το δυνατόν περισσότερα αντίπαλα μυρμήγκια. Έτσι εκτελεί την συνάρτηση findMyMaximumMoves().

Η συνάρτηση findMyMaximumMoves() είναι μια αναδρομική συνάρτηση η οποία αναζητά αν υπάρχει κίνηση αιχμαλώτισης δεξιά ή αριστερά σε μια συγκεκριμένη θέση και αν ναι τότε ξανά καλεί τον εαυτό της για τις καινούργιες θέσεις. Σταματάει να διασχίζει επιπλέον θέσεις είτε όταν έχει φτάσει τον μέγιστο αριθμό επιτρεπτών αιχμαλωτίσεων είτε όταν δεν υπάρχει δυνατότητα επιπλέον αιχμαλώτισης. Τέλος επιλέγει τις διαδρομές που της επιτρέπουν τις περισσότερες αιχμαλωτίσεις.

Η συνάρτηση addASonToThisState() χρησιμοποιείται από την συνάρτηση createStatesFromList() όταν έχει δημιουργήσει ένα καινούριο state και θέλει να το ορίσει ως παιδί του state από το οποίο προέκυψε.

Η συνάρτηση canCapture() χρησιμοποιείται μέσα στην συνάρτηση findAllPossibleMovesForThisRoundAndAddToTheList() ώστε να γνωρίζει σε ένα αρχικό επίπεδο αν υπάρχει κάποια κίνηση αιχμαλώτισης σε μια συγκεκριμένη θέση και να του επιστρέψει το direction αν υπάρχει.

Η συνάρτηση canIDieInTheNextRound() δημιουργήθηκε μετέπειτα ως ένας επιπλέον παράγοντας βαθμολόγησης των κινήσεων με στόχο ο πράκτορας να εκπαιδευτεί καλύτερα και να αποκτήσει καλύτερες στρατηγικές παιξίματος. Ουσιαστικά, η συνάρτηση αυτή εκτελείτε από κάθε καινούργιο state που δημιουργείται και ελέγχει αν στον επόμενο γύρω αυτό το πιόνι μπορεί να αιχμαλωτιστεί από τον αντίπαλο στον επόμενο γύρω. Αν αυτό το πιόνι ανήκει στα πιόνια του πράκτορα, τότε του αφαιρεί μονάδες ενώ αν ανήκει στου αντιπάλου τότε του προσθέτει μονάδες.

<u>Η βαθμολογία της κάθε κίνησης προκύπτει συνολικά από τα εξής :</u>

- Αν θα επισκεφτεί το φαγητό + ή – 4 (ανάλογα αν είναι πιόνι του πράκτορα ή του αντιπάλου).
- Αν μπορεί να αιχμαλωτιστεί από τον αντίπαλο στον επόμενο γύρω - ή + 5.
- Αν αιχμαλωτίσει αντίπαλο μυρμήγκι + ή – 5 για κάθε μυρμήγκι.
- Αν φτάσει στο τέλος της αντίπαλης πλευράς + ή – 5.
- Αν κάνει ένα απλό βήμα χωρίς φαγητό ή αιχμαλώτιση + ή – 1.

Από την εκφώνηση ζητήθηκε να βελτιωθούν περαιτέρω οι συναρτήσεις αναζήτησης minimax και α-β pruning με κάποιο singular extension. Έτσι δημιουργήθηκαν οι συναρτήσεις alphaBetaForwardPruning() και minimax ForwardPruning(). Σε αυτές τις συναρτήσεις όταν βρισκόμαστε στην ελαχιστοποίηση ή στην μεγιστοποίηση υπολογίζεται επιπλέον και ο μέσος όρος της βαθμολογίας των προηγούμενων παιδιών, ο οποίος ανανεώνεται και ξανά υπολογίζεται με μεγαλύτερη ακρίβεια καθώς επισκεπτόμαστε περισσότερα παιδιά. Οπότε στις πρώτες επισκέψεις παιδιών ο μέσος όρος της βαθμολογίας τους θα έχει μεγάλη απόκλιση από την πραγματική του τιμή. Έτσι οι συναρτήσεις αυτές διαθέτουν μια επιπλέον μεταβλητή ως όρισμα η οποία ορίζει πόσες επισκέψεις πρέπει να περιμένουμε μέχρι η τιμή του μέσου όρου να έχει μια ικανοποιητική ακρίβεια. Όταν ο μέσος όρος αποκτήσει αυτήν την ακρίβεια, τότε στην περίπτωση που βρισκόμαστε στην μεγιστοποίηση αν η βαθμολογία του τωρινού παιδιού είναι μεγαλύτερη κατά έναν αριθμό από τον μέσο όρο, θεωρούμε ότι αυτή η βαθμολογία θα είναι η μέγιστη και την επιστρέφουμε. Αν βρισκόμαστε στην ελαχιστοποίηση τότε αν είναι μικρότερη κατά ένα νούμερο από το μέσο όρο θεωρούμε ότι αυτή η βαθμολογία θα είναι η ελάχιστη και την επιστρέφουμε. Αυτό το νούμερο που μας βοηθά να εκτιμήσουμε αν η τωρινή

βαθμολογία είναι η μέγιστη ή η ελάχιστη δίνεται ως όρισμα σε αυτές τις συναρτήσεις.

Τέλος, στην main του client αλλάχθηκε μόνο το κομμάτι του case όπου ο server ζητάει από τον client να αποφασίσει μια κίνηση και να του την στείλει μέσω του socket. Κάθε φορά που ζητείται από τον client να επιλέξει κάποια κίνηση και δεν στέλνει null move, εκτελείτε η συνάρτηση stateGenerattion() ώστε να δημιουργηθούν όλες οι πιθανές καταστάσεις κινήσεων. Έπειτα, αν έχει παραπάνω από μια πιθανή κίνηση που μπορεί να εκτελέσει αξιολογούνται όλες στο μέγιστο βάθος που είναι εφικτό μέσω των συναρτήσεων minimax ή alphaBeta και επιλέγεται αυτή με την καλύτερη βαθμολογία. Στον πράκτορα δεν χρησιμοποιήθηκε ως βασική συνάρτηση η μέθοδος alphaBetaForwardPruning() διότι στο συγκεκριμένο πρόβλημα το δέντρο αναζήτησης καταστάσεων έχει πολύ μικρό βάθος και λίγα κλαδιά οπότε ιδανικότερη είναι η συνάρτηση alphaBeta() καθώς μπορεί να ψάξει ολόκληρο το δέντρο σε πολύ λίγο χρόνο χωρίς τον κίνδυνο να παραλείψει ένα ιδανικότερο μονοπάτι. Επίσης η συνάρτηση alphaBetaForwardPruning() χρειάζεται ιδιαίτερη προσοχή στις τιμές dif και waiting που θα δεχθεί ως όρισμα διότι αυτές καθορίζουν την ορθή λειτουργία της. Αν δοθούν λανθασμένα μικρές τιμές τότε η συνάρτηση θα παραλείπει κρίσιμα μονοπάτια που θα οδηγούσαν σε βέλτιστες στρατιγικές. Η συνάρτηση alphaBetaForwardPruning() είναι ιδανικότερη για δέντρα αναζήτησης με πολύ μεγάλο βάθος και πολύ πυκνά κλαδιά όπου η απλή α-β pruning θα είναι πολύ χρονοβόρα και θα απαιτεί μεγάλη επεξεργαστική ισχύ.