Εισαγωγή στον Προγραμματισμό(2015-2016)

4η εργασία(Quoridor Engine)

Σκοπός του αρχείου αυτού είναι να αναλυθούν τα βασικά σημεία-συναρτήσεις του προγράμματος (τα πηγαία αρχεία δεν περιέχουν επομένως comments). Επίσης, η ομάδα δηλώνει ότι επιθυμεί να συμμετάσχει στο κύπελλο Quoridor που θα διεξαχθεί.

Το πρόγραμμα αποτελείται από τα πηγαία αρχεία :

- main.c
- cmds.c
- playingCmds.c
- createMatrices.c
- bfs.c
- undo.c
- utilities.c
- minimax.c

καθώς και τα αρχεία επικεφαλίδας :

- mainUtilities.h
- matricesUtilities.h
- bfsUtilities.h
- utilities.h

Αναλυτικά :

main.c

Αρχικά, ορίζονται και αρχικοποιούνται οι βασικές μεταβλητές που προσδιορίζουν το παιχνίδι όπως : αριθμός τοίχων(wallNum),μέγεθος του ταμπλό(sizeBoard) κα. Αφού δημιουργηθεί το ταμπλό από κατάλληλες συναρτήσεις, υπάρχει μια συνεχής επανάληψη, στην οποία το πρόγραμμα αφού δεχθεί και επεξεργαστεί την είσοδο καλεί τις αρμόδιες λειτουργίες για να ανταποκριθεί σε αυτήν.

cmds.c

- *toLower* : Χρησιμοποιεί την tolower της ctype.h στους διαδοχικούς χαρακτήρες του string για να τους μετατρέψει σε μικρούς.
- *getCommands* : Επεξεργάζεται την είσοδο(από την fgets) σύμφωνα με το πρωτόκολλο χωρίζοντάς την σε κατάλληλα κομμάτια, τα οποία αποθηκεύονται.

playingCmds.c

- *playmove* : Η συνάρτηση που υλοποιεί την ομώνυμη εντολή. Διαχειρίζεται την είσοδο κατάλληλα ώστε να την μετατρέψει από vertex σε συντεταγμένες του δισδιάστατου πίνακα char , ελέγχει την εγκυρότητα της κίνησης μέσω συνάρτησης και αν αυτή είναι νόμιμη την πραγματοποιεί. Τέλος εισάγει την κίνηση σε λίστα που περιέχει το σύνολο των κινήσεων που έχουν γίνει.
- *playwall:* Αντίστοιχα με την playmove επεξεργάζεται κατάλληλα την είσοδο. Στη συνέχεια αν ο παίκτης διαθέτει τοίχους ελέγχει την εγκυρότητα τους σύμφωνα με τους κανόνες του παιχνιδιού χρησιμοποιώντας και τον αλγόριθμο BFS(Breadth-First Search) για να βεβαιωθεί ότι κάθε παίκτης έχει διαθέσιμο μονοπάτι προς τη νίκη.Αν ο τοίχος τελικά είναι έγκυρος τοποθετείται στο ταμπλό και η κίνηση αποθηκεύεται στη λίστα κινήσεων.(Τα παραπάνω γίνονται με τη βοήθεια κατάλληλων συναρτήσεων).
- *genmove:* Η genmove διαλέγει μία όσο το δυνατόν καλύτερη κίνηση για τον παίκτη για τον οποίο καλέστηκε με τη βοήθεια του αλγορίθμου minimax με βελτιστοποίηση alpha-beta ,την πραγματοποιεί και την εισάγει στη λίστα κινήσεων.
- *showboard:* Εκτυπώνει αναπαράσταση του ταμπλό στην οθόνη με κατάλληλες επαναλήψεις.

createMatrices.c

- *createVisited/freeVisited :* Δυναμική δέσμευση(μέσω malloc) και αποδέσμευση αντίστοιχα μνήμης για τον πίνακα ακεραίων visited που χρησιμοποιείται από τον αλγόριθμο BFS.
- *resetVisited:* Επαναρχικοποίηση του πίνακα visited σε default τιμές .
- *mallocboard/freeboard:* Δυναμική δέσμευση(μέσω malloc) και αποδέσμευση αντίστοιχα μνήμης για το ταμπλό του παιχνιδιού.
- *makeboard:* Αρχικοποίηση του πίνακα-ταμπλό, καθώς και των θέσεων των παικτών στο ταμπλό.

bfs.c

- Αρχικοποίηση ενός στατικού πίνακα moves ο οποίος χρησιμοποιείται για να παραχθούν όλες οι πιθανές κινήσεις ενός παίκτη από μια δεδομένη θέση.Επίσης ορίζεται και η δομή του κόμβου λίστας.
- *insert,extract,emptyQueue :* Συναρτήσεις που υλοποιούν τις ομώνυμες πράξεις-λειτουργίες για μια ουρά η οποία χρησιμοποιείται στον αλγόριθμο BFS.
- *findPath:* Υλοποίηση του αλγορίθμου BFS. Πρόκειται για έναν αναδρομικό αλγόριθμο ο οποίος δεδομένης μιας θέσης παράγει και τοποθετεί όλες τις νόμιμες κινήσεις του παίκτη σε μια ουρά. Κάθε φορά που καλεί τον εαυτό του παίρνει ως δεδομένη-αρχική θέση την κίνηση που βρίσκεται στην

κεφαλή της ουράς και παράγει ξανά από αυτήν τις νέες νόμιμες κινήσεις, τις τοποθετεί στην ουρά και αφαιρεί την κίνηση στην κεφαλή της ουράς, μετακινώνοντας την κεφαλή στην επόμενη κίνηση της ουράς κοκ.Επίσης, χρησιμοποιεί τον πίνακα visited για να σηματοδοτεί τις θέσεις που έχει ήδη επισκεφτεί, ώστε να τις αποφύγει. Ο αλγόριθμος σταματά είτε όταν η κεφαλή της ουράς περιέχει νικητήρια κίνηση για τον παίκτη είτε εάν η ουρά αδειάσει, πράγμα το οποίο σημαίνει ότι κανένα από τα διαθέσιμα μονοπάτια δεν μπορεί να οδηγήσει στην νίκη του παίκτη πράγμα που απαγορεύεται από τους κανόνες . (Έτσι ελέγχονται τα πιθανά "μπλοκαρίσματα" κάποιου παίκτη από τοίχους οι οποίοι είναι παράνομοι).

undo.c

- *undo:* Η λειτουργία της συνάρτησης undo που πραγματοποιεί την ομώνυμη εντολή στηρίζεται στην λίστα στην οποία αποθηκεύεται κάθε νέα κίνηση από την αρχή του παιχνιδιού. Οι κόμβοι της λίστας ορίζονται ως δομή με όνομα move η οποία περιέχει τα κατάλληλα μέλη για τον πλήρη καθορισμό μιας κίνησης (χρώμα παίκτη,τοίχος ή κίνηση, αριθμός κίνησης,..κλπ). Η undo χρησιμοποιεί την λίστα ως στοίβα (LIFO list). Κάθε στιγμή διατηρεί έναν δείκτη στο τελευταίο στοιχείο που εισάχθηκε στη στοίβα(δηλαδή το πρώτο που θα εξαχθεί (list head)) καθώς και έναν δείκτη στο 'παλαιότερο',δηλαδή, το στοιχείο της στοίβας το οποίο θα εξαχθεί τελευταίο (list tail). Η συνάρτηση δουλεύει με τον εξής τρόπο : Αρχικά στην μεταβλητή count αποθηκεύεται ο αριθμός των κινήσεων που έχουν πραγματοποιηθεί από την αρχή του παχνιδιού. Η undo δέχεται ως όρισμα τον αριθμό των κινήσεων που πρέπει να αναιρέσει και υπολογίζοντας την διαφορά count και αριθμού κινήσεων βρίσκει το νέο αριθμό κινήσεων(count) μετά τις αναιρέσεις. Αν ο αριθμός των αναιρέσεων -έστω x- είναι επιτρεπτός, η συνάρτηση αφαιρεί από την στοίβα τις τελευταίες x κινήσεις και για κάθε κίνηση που αφαιρεί μειώνει κατά 1 το count (έτσι προκύπτει ουσιαστικά το νέο count). Στη συνέχεια για να φέρει το ταμπλό στην επιθυμητή κατάσταση κάνει reset το παιχνίδι και παίζει διαδοχικά τις κινήσεις της λίστας που έχουν απομείνει με απλοποιημένες συναρτήσεις 'παιξίματος' οι οποίες δεν περιέχουν ελέγχους αφού στην στοίβα εισάγονται μόνο έγκυρες κινήσεις.

utilities.c

- Αρχικά ορίζονται ως extern μεταβλητές που χρειάζεται να 'βλέπουν' οι συναρτήσεις του αρχείου.
- *check_Legal_Move/check_Special_Move :* Η check_Special_Move είναι προέκταση της check_Legal_Move καθώς καλείται από αυτήν για να ελέγξει τις περιπτώσεις των 'ειδικών' κινήσεων. Και οι δύο συναρτήσεις δέχονται ως όρισμα την κίνηση-στόχο και ελέγχουν αν είναι επιτρεπτή. Αν ναι, επιστρέφουν 1, αλλιώς 0.
- *actualMove* : Αλλάζει την θέση του Black ή White στο ταμπλό ανάλογα με την κίνηση που πραγματοποιήθηκε και ενημερώνει τον πίνακα position ο οποίος διατηρεί αποθηκευμένες τις τρέχουσες θέσεις των παικτών.

- *check_Legal_Wall :* Όμοια με τις checkLegalMove και checkSpecialMove ελέγχει αν ο τοίχος-στόχος είναι επιτρεπτός και επιστρέφει ανάλογη αληθοτιμή. Ο έλεγχος βέβαια δεν περιλαμβάνει το 'μπλοκάρισμα' ενός παίκτη το οποίο ελέγχεται από άλλη συνάρτηση.
- *tryWall :* Η tryWall πραγματοποιεί τον έλεγχο του μπλοκαρίσματος ενός παίκτη. Αρχικά τοποθετεί στο board τον τοίχο και καλεί την συνάρτηση findPath(BFS). Αν ο τοίχος είναι παράνομος τον αφαιρεί από το board.
- *simpleWall* : Δέχεται ως είσοδο τα δεδομένα που καθορίζουν πλήρως ένα τοίχο και απλώς τον τοποθετεί στο ταμπλό.
- *insertMove/extractMove/emptyMoves* : Αυτές οι συναρτήσεις πραγματοποιούν τις λειτουργίες push,pop και empty μιας στοίβας (LIFO list).
- *listcmd* : Απλώς τυπώνει τις εντολές που γνωρίζει το πρόγραμμα.
- *knowncmd* : Ελέγχει αν η εντολή που έχει δοθεί είναι γνωστή από το πρόγραμμα.
- *getMoves* : Επειδή κάποια στοιχεία του πίνακα moves εξαρτώνται από το μέγεθος του ταμπλό, αυτά πρέπει να αλλάζουν για κάθε διαφορετικό μέγεθος ταμπλό, λειτουργία την οποία υλοποιεί η getMoves.

minimax.c

- Αρχικά ορίζονται οι σταθερές SPP, SPO, RD, MDP οι οποίες είναι τα βάρη για τη συνάρτηση αξιολόγησης των κινήσεων (evaluateMove).Επιπλέον ορίζεται μία δομή χρήσιμη για την κωδικοποίηση οποιασδήποτε κίνησης (η οποία περιέχει και πεδίο value για την αξιολόγηση της κίνησης) και μια δομή χρήσιμη για κωδικοποίηση τοίχων.
- *evaluateMove :* Η συνάρτηση αυτή υπολογίζει βάσει κάποιων κριτηρίων το value μιας κίνησης και το επιστρέφει. Τα κριτήρια είναι :

  1) SPP(Shortest Path for Player) = Υπολογίζεται μέσω του αλγορίθμου BFS για τον παίκτη.

  2) SPO(Shortest Path for Opponent) = Υπολογίζεται όμοια για τον αντίπαλο.

  3) RD(Relative Distance) = Σχετίζεται με τη διαφορά μεταξύ των shortest path των δύο παικτών

  4) MDP(Manhattan Distance for Player) = Είναι η απόσταση του παίκτη από νικητήριο τετράγωνο χωρίς να λαμβάνονται υπόψιν τα εμπόδια.

  Προφανώς, όσο μικραίνει το SPP τόσο καλύτερα για τον παίκτη, οπότε στο value δεν προστίθεται το SPP αλλά το ((maxSPP - SPP)/maxSPP) το οποίο μεγαλώνει όσο μικραίνει το SPP, επομένως επιδρά μεγαλώνοντας το value μιας κίνησης.Ακριβώς με τον ίδιο τρόπο λειτουργεί και το SPO το οποίο όμως αφαιρείται από το value αφού έχει αρνητική επίδραση.Το RD(Relative Distance) προστίθεται και επιδρά θετικά στο value όταν ο παίκτης απέχει λιγότερα βήματα από τη νίκη σε σχέση με τον αντίπαλο του.

Το MDP επιδρά θετικά όσο ο παίκτης πλησιάζει σε νικητήριο row για αυτό προστίθεται στο value (όπως και πριν με τα SPO και SPP) το ((maxMDP-MDP)/maxMDP).

Σημείωση : Υπάρχουν και πολλά άλλα κριτήρια όπως και διαφορετικοί συνδυασμοί για τα βάρη τους. Στο πρόγραμμα τα βάρη επιλέχθηκαν αυθαίρετα μέσω δοκιμών.

- *getLegitMoves* : Δοκιμάζει όλες τις πιθανές κινήσεις ενός παίκτη και αποθηκεύει σε έναν πίνακα τις επιτρεπτές. Έτσι μπορούμε να έχουμε τα possible moves ενός παίκτη από οποιαδήποτε θέση.
- *alphaBetaUndo* : Αναιρεί μία κίνηση ή έναν τοίχο.(Όχι μέσω λίστας κινήσεων όπως στην undo.Απλά επεμβαίνει κατευθείαν στο ταμπλό.)
- *alphaBeta* : Υλοποίηση του αλγορίθμου minimax με βελτιστοποίηση alpha beta. Ο minimax είναι ένας αναδρομικός αλγόριθμος ο οποίος δημιουργεί ένα game tree με πιθανές κινήσεις και μόλις φτάσει στο τέλος(base case scenario) είτε επειδή κάποιος παίκτης νίκησε είτε επειδή έφτασε στο μέγιστο βάθος που ορίζεται από τον προγραμματιστή, αξιολογεί τα game state και επιστρέφει.Το βάθος σημαίνει πόσες διαδοχικές κινήσεις θα συμβούν πριν φτάσουμε στο base case όπου βαθμολογείται η κατάσταση του παιχνιδιού από την συνάρτηση evaluateMove (όσο μεγαλύτερο το βάθος δηλαδή τόσο πιο 'μπροστά' κοιτά ο αλγόριθμος στο παιχνίδι).
  Ο αλγόριθμος βασίζεται στην λογική ότι υπάρχουν δύο παίκτες : ο maximizer(max από εδώ και πέρα) και ο minimizer(min από εδώ και πέρα).Ο max είναι ο παίκτης για τον οποίο θέλουμε να παραχθεί κίνηση.
  Ο τρόπος με τον οποίο δημιουργείται το δέντρο είναι ο εξής:
  Υπάρχει ένας πίνακας με όλες τις πιθανές κινήσεις του παίκτη ο οποίος διατρέχεται με έναν επαναληπτικό βρόγχο.Για κάθε κίνηση καλείται αναδρομικά ο αλγόριθμος ώστε να κάνει κίνηση ο άλλος παίκτης(με την ίδια λογική).
  Έτσι δημιουργείται ένα δέντρο στο οποίο ξεκινώντας από την ρίζα και πηγαίνοντας μέχρι ένα φύλλο καταλήγουμε σε μια κατάσταση του παιχνιδιού (μετά από x κινήσεις, όπου x το βάθος που έχει οριστεί) η οποία και βαθμολογείται για τον maximizer.

  Ξεκινώντας τώρα από τα φύλλα του δέντρου επιλέγουν εναλλάξ κίνηση ο max και ο min ώσπου φτάνουμε στην ρίζα που αντιστοιχεί στον max. Αφού τα evaluate γίνονται για τον max, όταν αυτός 'επιλέγει' κίνηση διαλέγει πάντα αυτή με το μεγαλύτερο value ενώ ο αντίπαλος (min) αυτήν με το μικρότερο.Έτσι στην ρίζα καταλήγει μια κίνηση η οποία μπορεί να χαρακτηριστεί ως η καλύτερη δυνατή με την προϋπόθεση ότι ο αντίπαλος κάνει πάντα την καλύτερη κίνηση.

  Η βελτιστοποίηση alphaBeta βοηθάει να μειωθεί το μέγεθος του δέντρου και συνεπώς να γίνει πιο γρήγορα η επιλογή της κίνησης. Αυτό

επιτυγχάνεται επειδή φυλάσσεται η τιμή του parent node και συγκρίνεται με τα παιδιά του. Αν ένα παιδί γυρίσει μια τιμή που δεν πρόκειται να την διαλέξει ο παίχτης που παίζει στο επίπεδο του parent node "κόβεται" το υπόδεντρο αυτού του παιδιού. Στον φάκελο παράδοσης υπάρχει φωτογραφία με παράδειγμα.