Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογισμών
Τομέας Ηλεκτρονικής και Υπολογιστών
ΤΗΜΜΥ Α.Π.Θ

5ο Εξάμηνο
2016-2017

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

DS – Candy Crush

Εργασία Α’ – HeuristicPlayer

των φοιτητών:

Παπαδόπουλου Κωνσταντίνου (8677)
Τοπαλίδη Ευθύμιου (8417)

Κλάση HeuristicPlayer στο πακέτο Player

int deletedCandies(int[] move, Board board)

Μία από τις μεθόδους/κριτήρια που ακολουθήσαμε για τη δημιουργία της συνάρτησης αξιολόγησης είναι η συνάρτηση int deletedCandies(int[] move, Board board). Η συνάρτηση αυτή παίρνει ως ορίσματα έναν πίνακα με move και ένα αντικείμενο της κλάσεις Board και επιστρέφει έναν ακέραιο int με τον αριθμό των ζαχαρωτών, ανεξαρτήτου χρώματος, που απομακρύνονται από το ταμπλό.

Για την υλοποίηση της ακολουθήσαμε την εξής λογική:
Αναλόγως το index "σκανάρουμε" κάθε κατεύθυνση στις συντεταγμένες που ήταν το πλακίδιο και στις συντεταγμένες που θα βρεθεί, ψάχνοντας για διαδοχικά πλακίδια του ίδιου χρώματος με το πλακίδιο στην καινούργια θέση και με το πλακίδιο στην παλιά θέση αντίστοιχα.

Εδώ θα γίνει παρουσίαση των συναρτήσεων :

- int heightOfMove(int[] move)
- boolean vertOrHor(int[] move)
- double moveEvaluation(int[] move , Board board)
- int findBestMoveIndex(ArrayList<int[]> availableMoves , Board board)

που χρησιμοποιούνται προκειμένου να φθάσουμε στον τελικό μας σκοπό που δεν είναι άλλος από την αξιολόγηση των διαθέσιμων κινήσεων και επιλογή της καλύτερης βάση της βαθμολογίας που μας προσφέρει .

Ας ξεκινήσουμε με την int heightOfMove(int[] move) :

Η συνάρτηση αυτή υπολογίζει το ύψος του πλακιδίου πιο κοντά στον άξονα χ που απομακρύνεται .Η ιδέα της υλοποίησής της πηγάζει από τη σκέψη ότι όσο πιο κάτω βρίσκεται το πλακίδιο που απομακρύνεται τόσα περισσότερα πλακίδια θα μετακινηθούν στη συνέχεια(υπάρχοντα και νέα). Άρα μεγαλύτερη η πιθανότητα να επακολουθήσουν και άλλα σκασίματα μετά το πρώτο σκάσιμο και άρα περισσότεροι κερδισμένοι πόνοι.
Η συνάρτηση παίρνει ως όρισμα τον δείκτη κίνησης move και επιστρέφει το ύψος στο οποίο βρίσκεται το νέο πλακίδιο που συμμετέχει σε ένα πιθανό σκάσιμο.

Η δεύτερη κατά σειρά συνάρτηση είναι η boolean vertOrHor(int[] move) :

Η συνάρτηση αυτή βρίσκεται στο ίδιο μήκος κύματος λογικής με την παραπάνω . Δηλαδή σκεπτόμενοι ότι η απομάκρυνση πλακιδίων στον οριζόντιο άξονα μπορεί να προκαλέσει με μεγαλύτερη πιθανότητα αλυσιδωτές απομακρύνσεις πλακιδίων απ'ότι η κάθετη , επιστρέφουμε μία λογική τιμή με την True να αντιστοιχεί σε οριζόντιο σκάσιμο , άρα και σε μεγαλύτερη συνεισφορά στο συνολικό σκορ , και την τιμή False για την άλλη περίπτωση , η οποία θα εκτιμηθεί λιγότερο στην συνάρτηση moveEvaluation().

Η τρίτη συνάρτηση είναι η double moveEvaluation(int[] move , Board board) :

Στη συνάρτηση αυτή αξιοποιούμε την λειτουργια των τριών παραπάνω συναρτήσεων , για τις οποίες έγινε αναφορά , πολλαπλασιάζοντας τις επιστρεφομενες τιμές τους με συντελεστές που εξαρτώνται από την σημαντικότητα αυτών. Συγκεκριμένα θεωρήσαμε ως πιο σημαντική τη συνάρτηση deletedCandies() αφού η κύρια συνεισφορά στο τελικό σκορ έρχεται από τον αριθμό των πλακιδίωνν που θα σκάσουν. Δεύτερη σε σημαντικότητα θεωρήσαμε την heightOfMove() διότι όσο πιο βαθιά βρισκόμαστε στο Board τόσο περισσότερα πλακίδια θα εισαχθούν σε αυτό , άρα έχουμε την ευκαιρία για περισσότερες chainmoves στο τέλος του γύρου. Τρίτη σε εκτίμηση έχουμε την vertOrHor() που βασίζεται σε παρόμοια λογική με την heightOfMove() δηλαδή και εδώ επιλέξαμε ως πιο κερδοφόρο σκάσιμο το οριζόντιο από το κάθετο διότι ευελπιστούμε σε περισσότερα chainmoves στη συνέχεια. Στο τέλος επιστρέφουμε τη συνολική αξία της κίνησης που επιλέχθηκε συγκεντρώνοντας τις εκτιμήσεις από τις τρεις παραπάνω συναρτήσεις που αναφέραμε .

Η τέταρτη συνάρτηση είναι η findBestMoveIndex(ArrayList<int[]> availableMoves , Board board) :

Στα πλαίσια αυτής της συνάρτησης υλοποιούμε την αξιολόγηγση της καλύτερης κίνησης από τη λίστα διαθέσιμων κινήσεων που υπάρχουν και βρίσκονται στη λίστα availableMoves. Η αξιολόγηση πραγματοποιείται με τη βοήθεια της moveEvaluation(int[] move , Board board) .Η συνάρτηση αυτή ,σε μια επαναληπτική διαδικασία , παίρνει ως όρισμα κάθε φορά τον δεικτη κίνησης της διαθέσιμης κίνησης από την ArrayList και αποθηκεύει την τιμή αξιολόγησης της καθε διαθέσιμης κίνησης σε έναν πίνακα evals του οποίου το μέγεθος εξαρτάται από το μέγεθος της λίστας. Στη συνέχεια εφαρμόζοντας τον αλγόριθμο εύρεσης μεγίστου εντοπίζουμε ποιο στοιχείο του evals πίνακα θα επιφέρει μεγαλύτερο κέρδος για τον παίκτη μας και εν τέλει αυτό επιλέγεται ως επιστρεφόμενη τιμή της συνάρτησης findBestMoveIndex(ArrayList<int[]> availableMoves , Board board) .

Η παραπάνω διαδικασία φαίνεται και παρακάτω :

```java
int findBestMoveIndex (ArrayList<int[]> availableMoves, Board board)
{

    // int[] move = availableMoves.get(findBestMoveIndex(availableMoves, board));
    ArrayList<int[]> avmoves = new ArrayList<int[]> ();
    avmoves = availableMoves;
    int sizeofavmoves = avmoves.size();

    double[] evals = new double[availableMoves.size()];


    for(int i = 0; i < sizeofavmoves; i++){

       double evaluation = moveEvaluation(avmoves.get(i), board);
       evals[i] = evaluation;

    }


    // euresi megisths bathmologias gia thn epilogi ths kaluteris kinhshs

    double max = evals[0];
    int bestmove = 0;

    for (int i = 0; i < sizeofavmoves; i++){


        if (evals[i] > max){
            max = evals[i];
            bestmove = i;
        }
    }

    return bestmove;
}
```