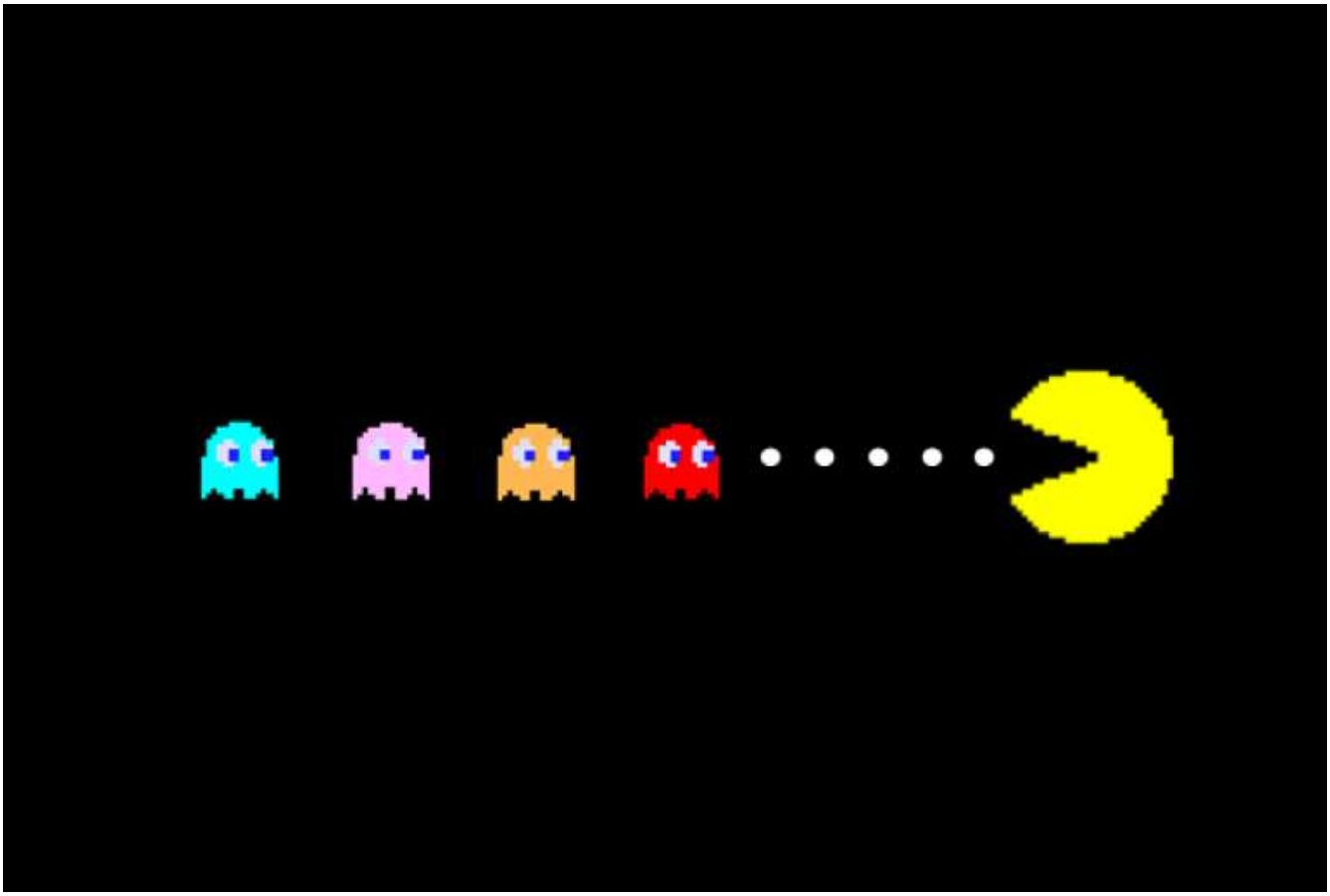
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ: DS – Pac-Man

Γιώργος Εφραίμ Παππάς
ΑΕΜ:9124
Email: gpappasv@ece.auth.gr



Ναπολέων Παπουτσάκης
ΑΕΜ:9170
Email: napoleop@ece.auth.gr

12/12/2017

ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ

Στην φετινή εργασία του μαθήματος καλούμαστε να δημιουργήσουμε μια απλοποιημένη μορφή του *Pac-man*. Ο *Pac-man* και τα φαντάσματα κινούνται μέσα στον χώρο που αποτελείται από κελία, τα όποια ορίζονται το καθένα με τις δίκες του συντεταγμένες. Στο χώρο των κελιών έχουν τοποθετηθεί σημαίες τις οποίες ο *Pac-man* θα πρέπει να συλλέξει. Σκοπός είναι ο *Pac-man* να καταφέρει να μαζέψει όλες τις σημαίες ξεφεύγοντας παράλληλα από τα φαντάσματα. Το παιχνίδι τελειώνει είτε όταν ο *Pac-man* καταφέρει να συγκεντρώσει όλες τις σημαίες , είτε όταν τον πιάσουν τα φαντάσματα , είτε όταν τελειώσει ο προκαθορισμένος αριθμός κινήσεων των φαντασμάτων.

Στο τρίτο μέρος της εργασίας καλούμαστε να υλοποιήσουμε την ξανά κίνηση του Pacman με διαφορετικό τρόπο από αυτόν της δεύτερης εργασίας. Μέσω μιας δενδρικής δομής και της τεχνικής minMax/ AB-Pruning ουσιαστικά επιλεγούμε την επομένη κίνηση λαμβάνοντας υπόψη ότι ο αντίπαλος- δηλαδή τα φαντάσματα –θα επιλέξουν την καλύτερη για αυτά κίνηση.

ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΟΥ

Αρχικά, στο αρχείο Node91709124.java, δηλώνεται η κλάση *Node91709124* με τις συναρτήσεις και τις μεταβλητές που απαιτούνται για την υλοποίηση των διαθέσιμων κινήσεων του *Pac-man.* Περά των μεταβλητών και των συναρτήσεων που προτείνονται υπάρχουν επίσης και κάποιοι getters/ setter για τις μεταβλητές,ώστε να μπορουμε να τις χρησιμοποιουμε σε ολο το προγραμμα. Ακολουθούν οι συναρτήσεις αρχικών συνθηκών της κλάσης ,όπου εχουμε την πρώτη συνάρτηση με μόνο όρισμα ένα δισδιάστατο πινάκα αντικειμένων τύπου *Room* ,που αρχικοποιεί τις μεταβλητές *nodeX,nodeY,nodeMove,depth,parent* και *node Evaluation* με 0(null για το parent) ενώ οι μεταβλητές *currentGhostPos* , *flagPos*, *currentFlagStatus* δέχονται τις επιστροφές των συναρτήσεων *findGhosts*(), *findFlags*(),*checkFlags* αντίστοιχα. Στη συνάρτηση αρχικών συνθηκών με ορίσματα έχουμε ως ορίσματα μια μεταβλητή ακέραιου (*direction*),ένα μονοδιάστατο πινάκα ακέραιων(*currPosition*),μια μεταβλητη ακεραιου (depth),ένα αντικειμενο τυπου Node(για το parent) και ένα δισδιάστατο πινάκα αντικειμένων τύπου *Room* . Για την αρχικοποίηση των μεταβλητών *nodeX, nodeY* καλειται η συναρτηση PacmanUtilities.evaluateNextPosition .Η μεταβλητή *nodeMove* δέχεται τη τιμή του ορίσματος *direction* και η *nodeEvaluation* την επιστροφή της συνάρτησης *evaluate* ,ενώ οι άλλες μεταβλητές αρχικοποιούνται με ακριβώς τον ίδιο τρόπο που αρχικοποιηθήκαν και στην πρώτη συνάρτηση αρχικών συνθηκών .

Έπειτα, ακολουθεί η υλοποίηση της συνάρτησης *findGhosts* η όποια βρίσκει και επιστρέφει τη θέση των φαντασμάτων κάθε στιγμή στο χώρο. Η συνάρτηση αυτή παίρνει ως όρισμα ένα δισδιάστατο πινάκα αντικειμένων τύπου *Room* και επιστρέφει ένα δισδιάστατο πινάκα ακέραιων. Πρώτα υπάρχει η δήλωση και η αρχικοποίηση με 0 μια μεταβλητής ακέραιου *rowghostPos* όπου θα περιέχεται ο

αριθμός της σειράς του πινάκα που η συνάρτηση επιστρέφει καθώς και η δήλωση του δισδιάστατου πινάκα *ghostPos* που η συνάρτηση θα επιστρέφει όπου για κάθε φάντασμα(αριθμός γραμμών) θα αποθηκεύονται οι συντεταγμένες της θέσης του (αριθμός στηλών). Ακολουθούν δυο βρόγχοι for οι όποιοι σκανάρουν όλο τον χώρο Maze , μέσα στους οποίους περιέχεται μια if(Εικόνα 1) όπου ελέγχεται με κλήση της συνάρτησης *isGhost*() του *Maze*[i][j] αν υπάρχει φάντασμα στη θέση I,j του χώρου Maze. Σε περίπτωση που η συνθήκη επαληθευτεί ο πινάκας *ghostPos* στη θέση [*rowghostPos*][0] και [*rowghostPos*][1] δέχεται τις τιμές *i,j* αντίστοιχα ,δηλαδή τις συντεταγμένες της θέσης του φαντάσματος, και η μεταβλητή *rowghostPos* αυξάνεται κατά 1 ώστε η επομένη καταχώριση συντεταγμένων να γίνει στην επομένη γραμμή του πινάκα. Τέλος ,επιστρέφεται ο πινάκας *ghostPos.*

```
if (Maze[i][j].isGhost())// ελεγ
{
    ghostPos[rowghostPos][0]=i;/
    ghostPos[rowghostPos][1]=j;/
 rowghostPos++;// αυξηση της μετ

}
```

Εικόνα 1: Η if για τον έλεγχο ύπαρξης φαντασμάτων σε κάθε θέσης του Maze.

Στη συνεχεία ,υπάρχει η υλοποίηση της συνάρτησης *findFlags*() που είναι υπεύθυνη για την εύρεση των θέσεων της πλατφόρμας όπου υπάρχουν σημαίες. Ως όρισμα μπαίνει και πάλι ένας δισδιάστατος πινάκας αντικειμένων τύπου *Room* και επιστρέφεται ένας δισδιάστατος πινάκα ακέραιων. Αρχικά, εμφανίζεται η δήλωση και η αρχικοποίηση με 0 μια μεταβλητής ακέραιου *rowflagPos* όπου θα περιέχεται ο αριθμός της σειράς του πινάκα που η συνάρτηση επιστρέφει, καθώς και η δήλωση αυτου του δισδιάστατου πινάκα *flagPos*,όπου για κάθε σημαία (δηλ.πρωτη σημαια ,πρωτη γραμμη του πινακα) θα αποθηκεύονται οι συντεταγμένες της θέσης της στην πλατφόρμα. Παρακάτω , υπάρχουν δυο βρόγχοι for οι όποιοι σκανάρουν όλο τον χώρο Maze , μέσα στους οποίους έχουμε μια if όπου ελέγχεται με κλήση της συνάρτησης *isFlag*() του *Maze*[i][j] αν υπάρχει σημαία στη θέση I,j του χώρου Maze. Εάν, η συνθήκη ισχύει ο πινάκας *flagPos* στη θέση [*rowflagPos*][0] και [*rowflagPos*][1] γεμίζει με τις τιμές *i,j* αντίστοιχα ,δηλαδή τις συντεταγμένες της θέσης της σημαίας, και η μεταβλητή *rowflagPos* αυξάνεται κατά 1 ώστε η επομένη καταχώριση συντεταγμένων να γίνει στην επομένη γραμμή του πινάκα. Τέλος ,επιστρέφεται ο πινάκας *flagPos.*

Επιπλέον, παρακάτω υπάρχει η υλοποίηση της συνάρτησης *checkFlags* που είναι υπεύθυνη για την εύρεση της κατάστασης κάθε σημαίας πάνω στην πλατφόρμα(εάν δηλαδή έχει αποκτηθεί από τον *Pac-man*).Και σε αυτή την συνάρτηση ως όρισμα μπαίνει και πάλι ένας δισδιάστατος πινάκας αντικειμένων τύπου *Room* και επιστρέφεται ένας πινάκας *boolean*. Στην αρχή, υπάρχει η δήλωση και η αρχικοποίηση με 0 μια μεταβλητής ακέραιου *rowStatFlags* όπου θα περιέχεται ο αριθμός της σειράς του πινάκα που η συνάρτηση επιστρέφει, ο όποιος δηλώνεται αμέσως παρακάτω με μέγεθος έσο με τον αριθμό των σημαιών και όνομα *statFlags*. Έπειτα, , υπάρχουν δυο βρόγχοι *for* οι όποιοι σκανάρουν όλο τον χώρο *Maze* , μέσα στους οποίους έχουμε μια *if* όπου ελέγχεται με κλήση της συνάρτησης *isFlag*() του *Maze*[i][j] αν υπάρχει σημαία στη θέση I,j του χώρου *Maze*. Σε περίπτωση που η συνθήκη ισχύει ο πινάκας *statFlags* στη θέση [*rowStatFlags]* αποθηκεύεται η επιστροφή της συνάρτησης *isCapturedFlag*() του *Maze*[i][j] ,η όποια επιστρέφει *true*

αν η σημαία στη θέση I,j έχει καταλειφθεί από τον *Pac-man,* και αυξάνεται η μεταβλητή *rowStatFlags* κατά 1 ώστε η επομένη κατάχωση της κατάστασης της επόμενης σημαίας να γίνει στην επομένη θέση του πινάκα. Τέλος επιστρέφεται ο πινάκας *statFlags.*
Τελευταία συνάρτηση στο αρχείο αυτό είναι η *evaluate*() που αξιολογεί την επομένη κίνηση του *Pac-man* συμφώνα με κάποια κριτήρια. Ως όρισμα αυτή η συνάρτηση δέχεται μια μεταβλητή *move* ακέραιου , ένα δισδιάστατο πινάκα ακέραιων *currentPos* καθώς και τον πινάκα *Maze*. Ξεκινάμε ορίζοντας την *double* μεταβλητή *evaluation* με τιμή ίση με το μηδέν ώστε να μεταβληθεί κατάλληλα με τα επόμενα 4 κριτήρια τα όποια λειτουργούν ως εξής:
-Πρώτο κριτήριο (απομάκρυνση από τοίχους): Σε περίπτωση που η επικείμενη κίνηση φέρνει τον *Pac-man* σε κάποια από τις άκρες του χώρου *Maze* (κάτι που ελέγχεται μέσω μιας *if*) αφαιρούμε από την τιμή της evaluation 5 μονάδες καθώς θεωρείται αρνητική επειδή περιορίζει τις κινήσεις του *Pac-man* σε επομενη φαση
-Δεύτερο κριτήριο (απόσταση του *pac-man* από τις σημαίες): Ο μονός τρόπος να νικήσει ο *pac-man*, είναι να καταλάβει όλες τις σημαίες, όποτε πρέπει να ευνοηθούν οι κινήσεις που τείνουν να βοηθήσουν τον *pac-man* να πάρει σημαίες, έτσι το 2ο κριτήριο λειτούργει ως εξής:
Αρχικά δημιουργούμε μια μεταβλητή τύπου *int*, την *availableflags* που θα μετρήσει τις διαθέσιμες σημαίες (όσες δεν έχουν ακόμα καταλειφθεί) στον χάρτη. Ύστερα δημιουργούμε έναν πινάκα *boolean* (τον *curFlSt*) που θα δεχθεί τις τιμές της συνάρτησης *checkFlags*(*maze*) (η λειτουργιά της οποίας εξηγήθηκε παραπάνω) ,ώστε μετά, με μια for (που θα τρέξει στον *curFlSt*), θα δούμε πόσες σημαίες είναι ενεργές και τότε θα πάρει την τελική της τιμή η *available flags*.
Ύστερα ορίζουμε δυο πινάκες μονοδιάστατους, τύπου *double*, τους *flagDistanceFromCurpos* (που θα αποθηκεύσει την απόσταση της τωρινής θέσης του pac-man από κάθε σημαία) και *flagDistanceFromNextPos* (που θα αποθηκεύσει την νέα απόσταση του pac-man από την κάθε σημαία, αφού δηλαδή κινηθεί προς την κατεύθυνση που βρίσκεται υπό αξιολόγηση) και μεγέθους όσου με τις ενεργές σημαίες που μετρήσαμε προηγουμένως. Αμέσως μετά, δημιουργούμε έναν δισδιάστατο *int* πινάκα, τον *flagP* με μέγεθος [4][2] που θα δεχθεί στην επομένη γραμμή την επιστροφή της συνάρτησης *findFlags*(maze). Ύστερα ακολουθεί μια for μέσα στην όποια, αν ισχύει η συνθήκη της *if* (αν δηλαδή στην κάθε θέση του *curFlSt*, που τρέχουμε με την βοήθεια της ίδιας της *for*, έχουμε ενεργη σημαία), βάζουμε στην θέση k, που ορίσαμε μέσα στην *for* με την τιμή μηδέν, των πινάκων *flagDistanceFromCurPos* και f*lagDistanceFromNextPos* τις αντίστοιχες αποστάσεις που αναφέραμε πριν και αυξάνουμε την τιμή του k ταυτόχρονα ώστε να προσπελάσουμε σε επομένη ενδεχόμενη τήρηση της συνθήκης if την επομένη θέση των πινάκων(Εικόνα 2).

```
double[] flagDistanceFromCurPos=new double[availableflags];//πινακας μεγεθους ισου με τις ενεργες σημαι
double[] flagDistanceFromNextPos=new double[availableflags];//πινακας μεγεθους ισου με τις ενεργες σημ
int[][] flagP=new int[4][2];
flagP=findFlags(Maze);
for(int i=0,k=0;i<4;i++){
    if(!curFlSt[i]){//ελεγχουμε αν η σημαια στην θεση i του currentFlagStatus ειναι ενεργη ωστε να αποθ
        flagDistanceFromCurPos[k]=(Math.sqrt((flagP[i][0]-currentPos[0])*(flagP[i][0]-currentPos[0])+
                (flagP[i][1]-currentPos[1])*(flagP[i][1]-currentPos[1])));
        flagDistanceFromNextPos[k]=(Math.sqrt((flagP[i][0]-nodeX)*(flagP[i][0]-nodeX)+
                (flagP[i][1]-nodeY)*(flagP[i][1]-nodeY)));
        k++;}//μετρητης k που αυξανει καθε φορα που βρισκουμε ενεργη σημαια (θα φτασει μεχρι την τιμη
}
```

Εικόνα 2 : Φαίνεται το πως γεμίζουν οι πινάκες αποστάσεων με τιμές

Έπειτα, βρίσκουμε την θέση του *flagDistanceFromCurPos* με την ελάχιστη τιμή ως εξής: Ορίζουμε μια μεταβλητή τύπου *int*, την *minDistance*, με τιμή ίση με το μηδέν και, σε περίπτωση που οι ενεργές σημαίες είναι περισσότερες από 0, μέσω μιας for προσπελάζουμε τον πινακα *flagDistanceFromCurPos* και αποθηκεύουμε τελικά στην μεταβλητή *minDistance* την θεση αυτου με την ελάχιστη τιμη. Αυτό μας επιτρέπει να ελέγξουμε με τις επόμενες 2 if αν η τιμη *flagDistanceFromNextPos*[*minDistance*] είναι μικρότερη η όχι από την τιμή *flagDistanceFromCurPos*[*minDistance*], όπου αν είναι, σημαίνει ότι ο *pac-man* πλησιάζει την κοντινότερη σε αυτόν σημαία αν γίνει η κίνηση που βρίσκεται υπό αξιολόγηση. Αν πλησιάζει την σημαία δίνουμε για την κίνηση 38 βαθμούς στο *evaluation* της, αλλιώς της αφαιρούμε 17 βαθμούς (η διάφορα τιμής οφείλεται στο οτι θεωρούμε πολύ σημαντικό το να πλησιάσει μια σημαία αλλά όχι τόσο σημαντικό το να απομακρυνθεί από αυτήν καθώς ενδέχεται να κοστίσει την ζωή του *pac-man* η μεγάλη υπονόμευση μιας κίνησης που τον απομακρύνει από αυτήν)
-Τρίτο κριτήριο (με βάση την απόσταση από τα φαντάσματα): Με τον ίδιο τρόπο με το κριτήριο 2 δημιουργούμε τους πινάκες *ghostDistanceFromCurPos* και *ghostDistanceFromNextPos*, που τους γεμίζουμε με τιμές ακριβώς με την ιδία λογική που εξηγήθηκε, και βρίσκουμε το αν απομακρύνθηκε από το κάθε φάντασμα με τον ίδιο τρόπο όπως με τις σημαίες. Στην περίπτωση όμως του κριτηρίου 3, όταν απομακρύνεται από ένα φάντασμα ο *pac-man*, κάνοντας την υπό αξιολόγηση κίνηση, προσθέτουμε στην τιμη της *evaluation* 40 βαθμούς, περισσότερους δηλαδή από το να πλησιάσει μια σημαία, ενώ αφαιρούμε 20 βαθμούς αν τείνει να πλησιάσει ένα φάντασμα, επίσης περισσότερους από τους βαθμούς που αφαιρούμε σε περίπτωση απομάκρυνσης από σημαία, καθώς θεωρούμε πιο σημαντική την επιβίωση του pac-man από το να πάρει μια σημαία, έτσι δηλαδή όταν έχουμε 2 κινήσεις όπου στην μια πλησιάζει ο *pac-man* την σημαία αλλά και το κοντινότερο σε αυτόν φάντασμα ενώ στην άλλη απομακρύνεται από το κοντινότερο φάντασμα και από την κοντινότερη σε αυτόν σημαία δίνουμε μεγαλύτερη βαρύτητα στην κίνηση όπου απομακρύνεται από το φάντασμα και την σημαία.
-Τέταρτο κριτήριο: Παρατηρήσαμε ότι υπαρχών περιπτώσεις, όπου το κοντινότερο στον pac-man φάντασμα, είχε μεγαλύτερη απόσταση από την κοντινότερη σε αυτόν σημαία από ότι ο *pac-man* από την σημαία και ταυτόχρονα η κίνηση προς την κοντινότερη σημαία σήμαινε και κίνηση προς το κοντινότερο φάντασμα άρα και απόρριψη αυτής λογω μεγαλύτερης βαρύτητας στο κριτήριο 3 από οτι στο 2, κάτι που είδαμε ότι δεν ήταν αποδοτικό καθώς *ο pac-man* θα μπορούσε να πάει να πάρει την σημαία, να πλησιάσει το φάντασμα, αλλά το φάντασμα δεν θα μπορούσε να τον πιάσει, Κατά το κριτήριο 4 λοιπόν, όταν *ο pac-man* μπορεί να πιάσει μια σημαία πριν τον προλάβει το κοντινότερο φάντασμα (αφού απόσταση φαντάσματος από σημαία > απόσταση pac-man από σημαία) προσθέτουμε στην κίνηση που τείνει να πραγματοποιήσει κάτι τέτοιο +40 ώστε να την προτιμήσει σχεδόν σίγουρα (αφού άλλωστε αποτελεί ιδανική κίνηση καθώς και παίρνει σημαία και παραμένει ασφαλής.)(Εικόνα 3).

```
// ---------------------- μεχρι κριτηριων 4
if(availableflags>0)
if(ghostDistanceFromNextPos[minDistance1]>flagDistanceFromNextPos[minDistance] &&
        (flagDistanceFromCurPos[minDistance]>flagDistanceFromNextPos[minDistance]))evaluation=evaluation+40;
```

Εικόνα 3: Κριτήριο 4

Στο αρχείο Creature.java, υλοποιήσαμε την συνάρτηση *calculateNextPacmanPosition*() μέσα στην όποια υπολογίζεται η επομένη κίνηση του Pac-man. Η συνάρτηση αυτή δέχεται ως ορίσματα έναν δισδιάστατο πινάκα αντικειμένων τύπου *Room* και ένα μονοδιάστατο πινάκα ακέραιων *currPosition*. Αρχικά, δηλώνεται ένα αντικείμενο τύπου Node όπου θα είναι η ρίζα του δέντρου με τιμή -1 για το όρισμα του direction καθώς δεν θέλουμε να κινηθεί και 0 για το όρισμα του depth. Το αντικείμενο –ρίζα- μπαίνει ως όρισμα στη συνάρτηση createSubTreePacman για τη δημιουργία του υποδέντρου του pacman , η υλοποίηση της όποια θα περιγράφει αναλυτικότερα παρακάτω. Στο όρισμα που δέχεται το βάθος των κόμβων που φτιάχνει αυτή η συνάρτηση μπαίνει το βάθος της ρίζας αυξημένο κατά 1. Έπειτα, δημιουργείται και αρχικοποιείται μια μεταβλητή double mostSutMove η όποια δέχεται την επιστροφή της κλίσης της συνάρτησης ABprun με ορίσματα root, βάθος δέντρου 2 α=-οο και b=+οο αρχίζοντας από το max part (maxP=true) δηλαδή την αξιολόγηση της κίνησης που επιλέχτηκε. Η συνάρτηση ABprun υλοποιεί τη μέθοδο abprunning και θα εξηγηθεί επίσης παρακάτω. Στη συνεχεία δημιουργείται μια μεταβλητή τύπου ακεραίου moveToReturn,η όποια θα δεχθεί παρακάτω την κατεύθυνση της επόμενης κίνησης του pacman, και την αρχικοποιούμε με 0. Ακολουθει , διπλός βρόγχος for που σαρώνει όλα τα φύλλα του δέντρου και μέσω μιας συνθήκης if ταυτοποιεί τον κόμβο του δέντρου με το evaluation που έχει αποθηκευτεί στην mostSutMove.Οταν το βρει και ισχύει η συνθήκη τότε η μεταβλητή moveToReturn κρατάει την κίνηση του αντικείμενου του οποίου το evaluation βρήκαμε παραπάνω. Ο πρώτος βρόγχος for τρέχει τόσες φορές όσες ο αριθμός των παιδιών της ρίζας ενώ ο δεύτερος όσες ο αριθμός των παιδιών των παιδιών της ρίζας. Τελικά η συνάρτηση επιστρέφει τη μεταβλητή moveToReturn με την κίνηση που θέλουμε.

```
//διπλος βρογχος for για την ταυτοποιηση του node που περιεχει το evaluation που επιστραφηκε
for(int i=0;i<root.getChildren().size();i++) {
    for(int y=0;y<root.getChildren().get(i).getChildren().size();y++) {

        if (mostSutMove==root.getChildren().get(i).getChildren().get(y).getnodeEvaluation())
            {moveToReturn=root.getChildren().get(i).getChildren().get(y).getNodeMove();
        }
    }
```

Εικόνα 4 : ο διπλός βρόγχος for που βρίσκει την τελική κίνηση που θα επιστραφεί

Η συνάρτηση createSubTreePacman είναι υπεύθυνη για την δημιουργία του δεύτερου επιπέδου του δέντρου. Αρχικά ορίζουμε ένα arraylist με όνομα avpos που θα δεχτεί αντικείμενα τύπου Node91709124. Μετά δημιουργούμε έναν πίνακα 4 θέσεων τύπου Node91709124 και βάζουμε σε αυτόν 4 αντικείμενα τύπου Node91709124 το κάθε ένα από τα οποία έχει διαφορετική κίνηση από τα άλλα (το πρώτο την 0, το δεύτερο την 1 κτλ). Ύστερα με μία for προσπελάζουμε τον πίνακα αυτόν και βλέπουμε αν το εκάστοτε αντικείμενο αντιπροσωπεύει μια εφικτή κίνηση σύμφωνα με την τότε θέση του pacman (δεν επιτρέπεται να κινηθεί πάνω σε τοίχο ή πάνω σε φάντασμα). Όσα αντικείμενα έχουν εφικτή κίνηση μπαίνουν στο παραπάνω arraylist. Μετά από την εύρεση των εφικτών κινήσεων ακολουθεί η δημιουργία ενός πίνακα 3 διαστάσεων με όνομα copies και τύπου Room. Σε κάθε θέση copies[i] αποθηκεύεται και ένα αντίγραφο του maze (που είναι 2διαστατο). Ακολουθεί η δημιουργία ενός πινάκα 2 θέσεων και μιας διάστασης, του nextPositions που θα δεχτεί τις συντεταγμένες του pacman για την αμέσως επομένη κίνηση του, και μετά μία for που προσπελάζει το avpos arraylist και:

Βάζει στην θέση copies[i] ένα αντίγραφο του maze με την συνάρτηση PacmanUtilities.copy, μετά αποθηκεύονται στον πινάκα οι συντεταγμένες του pacman κατά την κίνηση που ορίζει το αντικείμενο της i θέσης του avpos, δημιουργούμε ένα αντικείμενο Node91709124 με ονομα test δίνοντας του την κατεύθυνση που ορίζει το αντικείμενο της i θέσης του avpos, προσομοιώνουμε την κίνηση του pacman στο αντίγραφο του maze που βρίσκεται στην i θέση του copies με χρήση της συνάρτησης movePacman (στην όποια αξιοποιούμε τον πινάκα nextPositions ως όρισμα), φτιάχνουμε ένα νέο αντικείμενο με όνομα child που η τιμή της μεταβλητής του nodeMove μας λέει ποια ήταν η κατεύθυνση που προσομοιώθηκε, μεταβάλουμε την τιμή της evaluation του child εξισώνοντας την με αυτή του test, μεταβάλουμε και τις τιμές nodeX και nodeY εξισώνοντας τα με τα αντίστοιχα του test, αποθηκεύουμε το child στον arraylist children του αντικείμενου parent (που αναφέρεται στο αντικείμενο που βάλαμε ως όρισμα κατά την κλίση της createSubTreeGhosts) και τέλος καλούμε την συνάρτηση createSubTreeGhosts που θα αναλυθεί παρακάτω.

```
//βρογχος for που τρεχει οσο ο αριθμος των διαθεσιμων κινησεων του pacman
for(int i=0;i<avpos.size();i++) {

    copies[i] = PacmanUtilities.copy(Maze); // αποθηκευση του αντιγραφου του maze στην i θεση του πινακα των αντιγραφων
    nextPositions = PacmanUtilities.evaluateNextPosition(Maze,currPacmanPosition,avpos.get(i).getNodeMove(),PacmanUtilities.borders);
    Node91709124 test = new Node91709124(avpos.get(i).getNodeMove(),currPacmanPosition,0,null,Maze);//δημιουργια αντικειμενου τυπου N
    PacmanUtilities.movePacman(copies[i],currPacmanPosition ,nextPositions); //εξομοιωση της κινησης του pacman με χρηση της συναρτησ
    Node91709124 child = new  Node91709124 (avpos.get(i).getNodeMove(),currPacmanPosition,parent.getDepth() +1 ,parent, copies[i]);//
    child.setnodeEvaluation(test.getnodeEvaluation());//εξισωνουμε την τιμη του evaluate του child με αυτη του test
    child.setnodeXY(test.getNodeX(),test.getNodeY());//εξισωνουμε τις τιμες nodeX nodeY του child με αυτες του test
    parent.setChildren(child); // τοποθετηση του child στο arraylist του parent με κληση της αντιστοιχης setter
    createSubTreeGhosts (child.getDepth()+1,child,copies[i],child.getCurrentGhostsPos());  //κληση της συναρτησης  createSubTreeGhosts

}
```

Εικόνα 5 : ο βρόγχος for της συναρτησης createSubTreePacman.

Ο λόγος που δημιουργήσαμε ενδιάμεσα στην for το αντικείμενο test ήταν επειδή εξ ορισμού όταν αρχικοποιούμε ένα αντικείμενο Node91709124 βάζοντας του μια ορισμένη κατεύθυνση (συγκεκριμένο direction στην αρχικών συνθηκών του Node91709124) , αυτόματα παίρνουν τιμές τα nodeX και nodeY καθώς και το evaluate. Θέλαμε ουσιαστικά το child να περιέχει στην μεταβλητή nodeMove την κατεύθυνση που προσομοιώθηκε (να μην δηλώνει το nodeMove δηλαδή στην παρούσα φάση την κίνηση που πρόκειται να κάνει αλλά την κίνηση που έγινε ήδη λογά προσομοίωσης) και να μεταβάλουμε την τιμή του evaluation συμφώνα με την κίνηση που προσομοιώθηκε (γι αυτό την εξισώνουμε με αυτή του test) και αντίστοιχα να μεταβάλουμε και τα nodeX,nodeY (επειδή αν εξομοιώσω την κατεύθυνση πχ 0, αν φτιάξω αντικείμενο με direction 0 θα αξιολογηθεί η επομένη από την εξομοιωμένη)

Ακολουθεί η συνάρτηση createSubTreeGhosts που είναι υπεύθυνη για τη δημιουργία του τρίτου επίπεδου του δέντρου. Αρχικά δημιουργείται ένα arraylist όπου θα αποθηκευτούν αμέσως μετα όλες οι δυνατές κινήσεις των φαντασμάτων με κλήση της συνάρτησης PacmanUtilities.allGhostMoves. Ύστερα δημιουργείται ενας πινάκας όπου αποθηκεύεται η τωρινη θεση των φαντασμάτων ενώ δημιουργείται και ένας πινάκας copies τυπου Room με μέγεθος όσος ο αριθμός των διαθέσιμων κινήσεων , όπου θα αποθηκευτούν τα αντίγραφα του Maze που θα δημιουργηθούν παρακάτω. Έπειτα υπάρχει ένας βρόγχος for που τρέχει τόσες φορές όσες ο αριθμός των διαθέσιμων κινήσεων μέσα στον όποιο γίνονται τα εξης:πρωτα, αποθήκευση στη θέση i του πινάκα copies του αντιγράφου του maze με κλίση της συνάρτησης PacmanUtilities.copy.Έπειτα, προσομοιώνεται η κίνηση των φαντασμάτων με κλίση της συνάρτησης PacmanUtilities.moveGhosts ενώ δημιουργείται και ένας πινάκας ακέραιων με μέγεθος δυο στον όποιο

αποθηκεύουμε τις συντεταγμένες του pacman(ώστε να αξιοποιηθούν στους παρακάτω ελέγχους για το αν τα φαντάσματα πλησίασαν τον pacman) στο αντίγραφο του maze που βάλαμε ως όρισμα κατά την κλίση της CreateSubTreeGhosts. Μετά δημιουργείται ένα καινούριο node που θα αποτελέσει ένα από τα φύλλα στο δέντρο και θα κρατήσει την κατεύθυνση της κίνησης του πατερά του.Αμεσως μετά το evaluation του κόμβου αυτού θα μηδενιστεί προκειμένου να καθοριστεί με βάση τον αριθμό των φαντασμάτων αλλά και το evaluation που κρατήσαμε στον κόμβο-πατερά του. Για να επιτευχτεί αυτό αρχικά τοποθετούμε τον καινούριο κόμβο στο arraylist με τα παιδιά του parent και αλλάζουμε την τιμή του evaluation του καινούριου node σε 0.Ύστερα ,δημιουργούμε μια μεταβλητή ακέραιου ghostCounterCloser όπου θα αποθηκεύουμε τον αριθμό των φαντασμάτων που πλησιάζουν το pacman και θα ελέγχουμε ξεχωριστά αν κάθε φάντασμα πλησιάζει τον pacman (έλεγχος της μεταξύ τους απόστασης) και αν αυτό ισχύει θα αυξάνουμε τη μεταβλητη ghostCounterCloser κατά 1. Τελικα, οριζουμε επακριβως την τιμη της evaluation του καινουριου node.

Η βασική ιδεα των τελευταίων γραμμών είναι η εξης:στα nodes με βαθος 1 έχουμε τα nodes με τις αξιολογήσεις των κινήσεων του pacman καθως και τις αντίστοιχες κινήσεις εκ των οποίων θα επιδεχθεί (λογω max)
 το node (και άρα η κατεύθυνση) με το μεγαλύτερο evaluation. Στα nodes με βάθος 2 έχουμε τις κινήσεις των φαντασμάτων που θα ακολουθήσουν την κίνηση του pacman. Σε αυτό το σημείο θέλουμε να επιλέξουμε την κίνηση με το μικρότερο evaluation (λογά min) καθώς θεωρούμε ότι τα φαντάσματα θα παίξουν την καλύτερη για αυτά κίνηση, αυτή δηλαδή που θα τα φέρει πιο κοντά στον pacman. Ετσι,στα evaluation του προηγούμενου επίπεδου αφαιρούμε το κατάλληλο ποσο αναλογικά με τον αριθμό των φαντασμάτων που πλησιάζουν τον pacman.

```
newNode.setnodeEvaluation(ghostCounterCloser*(-12)+parent.getnodeEvaluation());
```

Εικόνα 6 : Ορισμός της τιμής του evaluation του προηγουμενου επιπεδου.

Κατά την συνάρτηση ABprun προσπελάζουμε το δέντρο που δημιούργησαν οι συναρτησεις createSubTreeGhosts και createSubTreePacman συμφώνα με τους κανόνες του ab pruning. Για την υλοποίηση της συναρτησης ακλουθήσαμε τον ψευδοκωδικα που βρήκαμε στο link :
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Pseudocode [edit]

The pseudo-code for minimax with alpha-beta pruning is as follows:[12]

```
01 function alphabeta(node, depth, α, β, maximizingPlayer)
02      if depth = 0 or node is a terminal node
03          return the heuristic value of node
04      if maximizingPlayer
05          v := -∞
06          for each child of node
07              v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
08              α := max(α, v)
09              if β ≤ α
10                  break (* β cut-off *)
11          return v
12      else
13          v := +∞
14          for each child of node
15              v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
16              β := min(β, v)
17              if β ≤ α
18                  break (* α cut-off *)
19          return v
```

```
(* Initial call *)
alphabeta(origin, depth, -∞, +∞, TRUE)
```

Τα δυο break που τονίζονται στην παραπάνω φωτογραφία είναι το πιο σημαντικό κομμάτι της συνάρτησης καθώς υλοποιούν την βασική ιδέα του ab pruning που είναι να σταματάμε το search του δέντρου όταν δεν πρόκειται να αλλάξει κάτι στον αλγόριθμο minimax