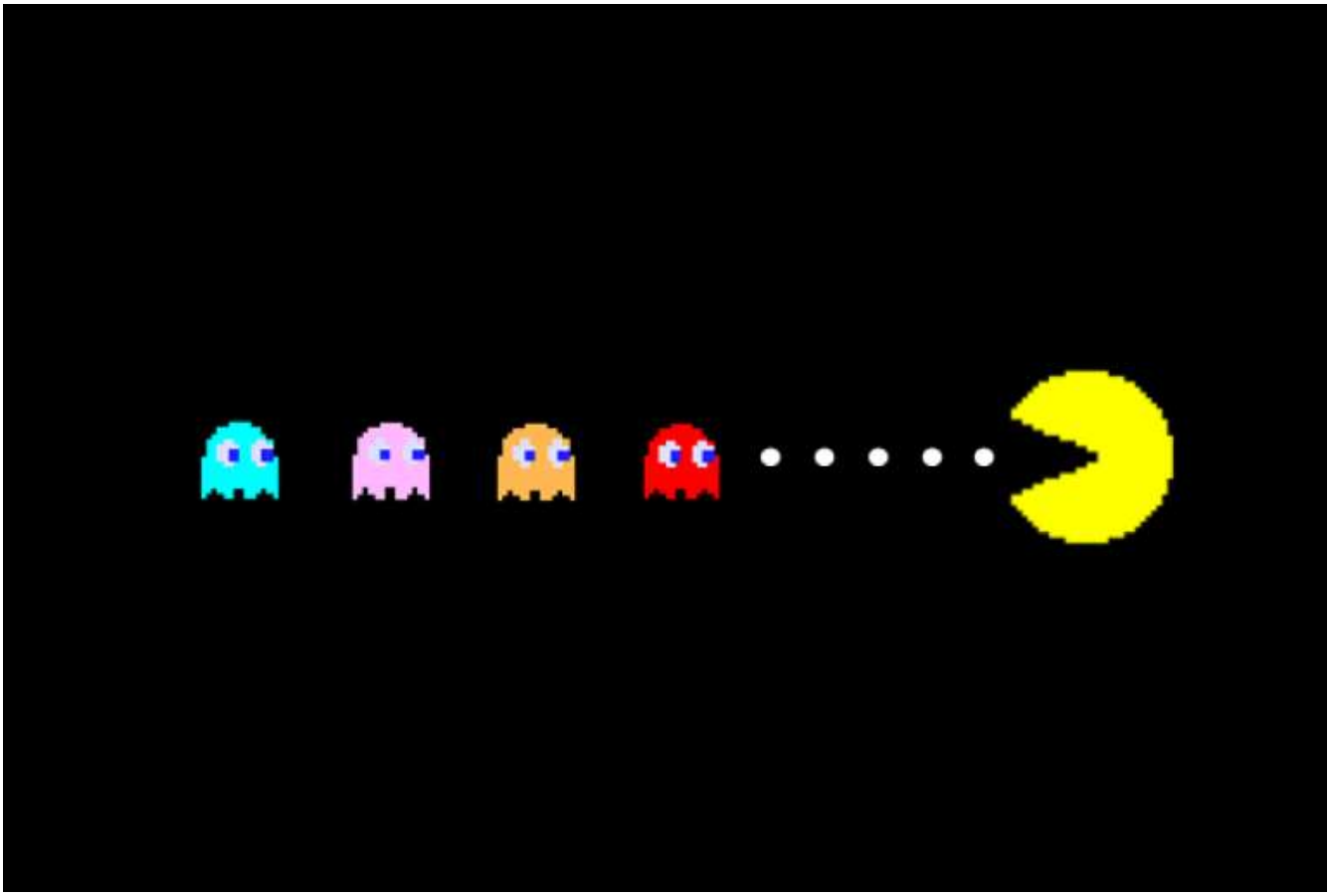
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

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 δεύτερο μέρος της εργασίας καλούμαστε να υλοποιήσουμε τις κινήσεις του *Pac-man* που θα συγκεντρώνονται σε μια δομή δεδομένων καθώς επίσης και μια συνάρτηση η όποια θα αξιολογεί την κάθε κίνηση του *Pac-man* με σκοπό την εύρεση της καλύτερης επόμενης κίνησης με βάση κάποια κριτήρια.

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 για τις μεταβλητές *nodeX,nodeY,nodeMove* και *nodeEvaluation* και ένας setter για τη μεταβλητή *nodeMove.* 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* και *node Evaluation* με 0 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*) και ένα δισδιάστατο πινάκα αντικειμένων τύπου *Room* . Για την αρχικοποίηση των μεταβλητών *nodeX, nodeY* χρησιμοποιείται μια δομή *switch* όπου ελέγχοντας τη μεταβλητή *direction* αποδίδεται κάθε φορά διαφορετική τιμή στις μεταβλητές αυτές ανάλογα το αν η μεταβλητή *direction* περιέχει την τιμή 0,1,2 ή 3.Αυτο επιτυγχάνεται με πρόσθεση ή αφαίρεση κάθε φορά της μονάδας από την τιμή που περιέχει ο πινάκας *currPosition*. 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 κατάχωσης 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. Ως επέκταση αυτού, με μια δεύτερη if ελέγχουμε αν ο Pac-man τείνει να πάει σε κάποια από τις γωνίες του *Maze*, αφαιρώντας του επιπλέον 10 καθώς έρχεται σε ακόμα πιο δύσκολη θέση (με τον τρόπο που υλοποιήθηκαν οι 2 αυτές *if*, είναι σίγουρο πως αν μπει στην δεύτερη, έχει σίγουρα μπει και στην πρώτη).
Μάλιστα, για να τον βοηθήσουμε να απομακρυνθεί από τους τοίχους προσθέτουμε βαθμούς σε περίπτωση που κινηθεί προς το κέντρο ως εξής:
Στις 4 πρώτες *if* ελέγχουμε αν προσεγγίζει το κέντρο (με τις πρώτες 2 ως προς x και με τις άλλες 2 ως προς y, θεωρώντας ως κέντρο το σημείο [αριθμός_γραμμών/2][αριθμός_στηλών/2] του *Maze*), προσθέτοντας 10 μονάδες στο *evaluation*.
Στις 4 επόμενες με παρόμοια λογική, εξετάζουμε αν απομακρύνεται από το κέντρο, αφαιρώντας από το *evaluation* 4 μονάδες (καθώς θέλαμε να δώσουμε λιγότερη βαρύτητα, σε σύγκριση με την κίνηση προς το κέντρο, στο γεγονός οτι κινείται σε πιο περιοριστικές περιοχές, αφού κάτι τέτοιο κάποιες φορές είναι αναγκαίο)
-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 δειχθεί 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 δειχθεί 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).

```
// ------------------- αρχή κριτηρίου 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 πινάκας αντικειμένων τύπου *Room* και ένα μονοδιάστατο πινάκα ακέραιων *currPosition*. Κατά πρώτον, δηλώνεται και αρχικοποιείται μια μεταβλητή *moveToReturn* τυπου ακέραιου με τιμή από 0 μέχρι 3 , δημιουργείται ένα *Arraylist* στο οποίο θα αποθηκεύονται οι διαθέσιμες κινήσεις του Pac-man πριν από κάθε γύρο και ένας πινάκας αντικειμένων *obj* τύπου *Node91709124* με μέγεθος 4 ,τα στοιχειά του οποίου αρχικοποιούνται με τις 4 διαφορετικές κατευθύνσεις που ο Pac-man μπορεί να ακολουθήσει στην επομένη κίνηση του. Μετά με ένα βρόγχο for για i από 0 μέχρι 3 (για κάθε πιθανή κατεύθυνση) ελέγχουμε, σε πρώτη φάση ,αν η επιλογή της κατεύθυνσης i θα στείλει τον Pac-man πάνω σε τοίχο. Αυτό γίνεται μέσω μιας if με συνθήκη την τιμή του πινάκα walls στη θέση i(1 αν δεν υπάρχει τοίχος) για το κελί του Maze, με συντεταγμένες την θέση του Pac-man πριν την κίνηση.Οι συντεταγμένες αυτές είναι τα στοιχεία 0 και 1 του πινάκα *currentPos*. Σε δεύτερο στάδιο(στο όποιο μπαίνουμε αν ο Pac-man δεν πάει πάνω σε τοίχο) ελέγχουμε αν προς την κατεύθυνση i υπάρχει κάποιο φάντασμα πράγμα που γίνεται με χρήση μιας άλλης if (μέσα στην προηγούμενη) όπου ως συνθήκη έχουμε την επιστροφή της συνάρτησης *isGhost*()για το κελί του Maze με συντεταγμένες την επομένη θέση του Pac-man αν ακολουθήσει την κατεύθυνση i. Οι συντεταγμένες αυτές βρίσκονται στις αντίστοιχες μεταβλητές *nodeX* και *nodeY* του αντικείμενου i του πινάκα *obj*, Αν η κίνηση επαληθεύει και τη δεύτερη συνθήκη τότε είναι έγκυρη και μπαίνει στο *ArrayList* με κλήση της συνάρτησης add.

Στην συνεχεία , ορίζεται ένας πινάκας double *evals* με μέγεθος όσο το μέγεθος του *ArrayList* ενώ ακολουθεί μια if στην όποια ελέγχεται αν το μέγεθος του *ArrayList* είναι μεγαλύτερο από 0(αν δηλαδή είναι άδειο ή όχι) .Αν η συνθήκη αυτή ισχύει ο πινάκας *evals* γεμίζει ,μέσω μιας for , με τις αξιολογήσεις κάθε κίνησης με κλήση της *getnodeEvaluation*() του αντιστοίχου αντικείμενου του *ArrayList*. Έπειτα, ακολουθεί η διαδικασία υπολογισμού του μεγίστου του πινάκα *evals*, όπου για το σκοπό αυτό δημιουργείται μια μεταβλητή max τύπου double, που αρχικοποιείται με το πρώτο στοιχείο του πινάκα evals ,και άλλη μια μεταβλητή *indexx* ακέραιου που αρχικοποιείται με 0 (θα είναι αυτή που θα κρατήσει τη θέση όπου βρίσκεται το μέγιστο). Με μια *if* μέσα σε ένα βρόγχο *for* ελέγχεται κάθε φορά αν το στοιχείο στη θέση i του πινάκα είναι μεγαλύτερο από το *max* και αν αυτό ισχύει το max παίρνει την τιμή του στοιχείου αυτού και η μεταβλητή *indexx* την τιμή i. Τέλος, η μεταβλητή

moveToReturn παίρνει την τιμή της αξιολόγησης του στοιχείου του *ArrayList* που βρίσκεται στη θέση *indexx*,(δηλαδή την θέση όπου βρίσκεται η μεγίστη τιμή) με κλήση της *getNodeMove*() της κλάσης *Node91709124*, το *ArrayList* αδειάζει με κλήση της συνάρτησης *clear*() και επιστρέφεται η τιμή της *moveToReturn*.