**ΙΩΑΝΝΗΣ ΚΑΠΕΤΑΝΓΕΩΡΓΗΣ** **Α.Μ. 1115201800061**

# PROJECT 2
# Τεχνητή Νοημοσύνη
# Pacman Project 2 (Πρόβλημα 5)

## Question 1 - Reflex Agent

Η συνάρτηση που πρέπει να υλοποιήσουμε πρέπει να εκτιμά κατά πόσο μια κίνηση είναι καλή η όχι. Μια κίνηση είναι καλύτερη από μια άλλη αν έχει μεγαλύτερο score. Για την εκτίμηση αυτή λαμβάνω υπόψη τις θέσεις των φαγητών και τις θέσεις των φαντασμάτων.
Πιο συγκεκριμένα:

- Αρχικά, δημιουργώ την καινούρια κατάσταση του παιχνιδιού έπειτα από την κίνηση αυτή μέσω της συνάρτησης generatePacmanSuccessors(action).
- Αν η κίνηση που έχει δοθεί είναι η Stop (δηλαδή να παραμείνει την ίδια θέση με πριν) επιστρέφω $-\infty$ καθώς δεν θεωρείται καθόλου καλή κίνηση (αφού δεν κερδίζομε τίποτα) και θέλουμε να γίνει μόνο σε περίπτωση που ο pacman δεν έχει άλλη επιλογή.
- Στην συνέχεια ελέγχω αν ο pacman έφαγε κάποιο φαγητό με αυτήν την κίνηση. Εάν έφαγε, προσθέτω στο score 2000 καθώς θεωρείται καλή κίνηση. Ο αριθμός 2000 είναι τυχαίος. Μπορεί να χρησιμοποιηθεί οποιοσδήποτε μεγάλος αριθμός αυτής της τάξης. Ο σκοπός του είναι απλά να αυξήσει αρκετά το score έτσι ώστε να δείξουμε ότι η κίνηση αυτή είναι καλή.
- Για κάθε ένα φαγητό που υπάρχει στο map υπολογίζω την απόσταση μεταξύ της τρέχουσας θέσης του pacman και του φαγητού αυτού χρησιμοποιώντας manhattan distance. Για κάθε μία απόσταση προσθέτω στο συνολικό score τον αριθμό 1000/απόσταση καθώς θέλουμε όσο πιο κοντά είναι ο pacman στο φαγητό (δηλαδή όσο μικρότερη είναι η απόσταση), τόσο μεγαλύτερος αριθμός να προστίθεται στο score. Πάλι, ο αριθμός 1000 είναι τυχαίος, καθώς μπορεί να αντικατασταθεί από οποιονδήποτε αριθμό συγκριτικά μεγαλύτερο από τις αποστάσεις που υπολογίζουμε.
- Τέλος λαμβάνουμε υπόψιν τα φαντάσματα. Πιο συγκεκριμένα ελέγχουμε αν στην νέα θέση που μετακινήθηκε ο pacman υπάρχει κάποιο φάντασμα. Επίσης ελέγχουμε και το scaredTimer του κάθε φαντάσματος. Έτσι, αν ο pacman και ένα φάντασμα βρίσκονται στην ίδια θέση υπάρχουν οι εξής δυο περιπτώσεις:
  - Αν το scaredTimer είναι μεγαλύτερο του μηδενός σημαίνει ότι ο pacman μπορεί να φάει το φάντασμα. Στην περίπτωση αυτή προσθέτουμε 2000 στο συνολικό score καθώς θεωρείται μια καλή κίνηση.
  - Αν το scaredTimer είναι ίσο με μηδέν σημαίνει ότι το φάντασμα τρώει το pacman, δηλαδή ο pacman χάνει. Στην περίπτωση αυτή επιστρέφουμε $-\infty$ καθώς δεν θέλουμε ο pacman να κάνει αυτήν την κίνηση γιατί αλλιώς θα χάσει.

Έπειτα από όλες τις παραπάνω προσθέσεις στο score προκύπτει το τελικό score (δηλαδή η τελική εκτίμηση της κίνησης) το οποίο και επιστρέφουμε.

## Question 2 - Minimax

Ο αλγόριθμος είναι βασισμένος στον αλγόριθμο που περιγράφεται στις διαφάνειες του μαθήματος. Έχω υλοποιήσει τις εξής 3 συναρτήσεις: minimax, maxValue και minValue. Η minimax ελέγχει κάθε φορά της συνθήκες τερματισμού του αλγορίθμου και αν αυτός δεν πρέπει να τερματιστεί καλεί την κατάλληλη συνάρτηση (minValue ή maxValue) έτσι ώστε να συνεχιστεί ο αλγόριθμός (δίνοντας τους και τα κατάλληλα ορίσματα). Η minValue εκτελεί τους υπολογισμούς για τον max και η minValue τους υπολογισμούς για τον min. Οι maxValue και minValue καλούν την minimax έτσι ώστε να συνεχιστεί η αναδρομή.
Στο πρόβλημα αυτό έχουμε πολλούς min “παίκτες”, δηλαδή πολλά φαντάσματα. Έτσι η διαδικασία που ακολουθείται είναι η εξής. Για κάθε ένα επίπεδο (depth) αρχικά εκτελείται η maxValue για τον pacman. Έπειτα, για κάθε πιθανή κίνηση του max καλείται ή minValue μια φορά για κάθε ένα φάντασμα (agent).

Πιο συγκεκριμένα, η διαδικασία που ακολουθείται είναι η εξής:
1. Η maxValue καλεί την minimax
2. Η minimax καλεί την minValue για τον πρώτο agent/φάντασμα
3. Μέσω της minValue καλείται η minimax
4. Η minimax καλεί ξανά την minValue για τον επόμενο agent
5. Τα βήματα 3 και 4 εκτελούνται μέχρι η minValue να κληθεί για τον τελευταίο agent
6. Έχει κληθεί η minValue για τον τελευταίο agent. Μέσω της minValue καλείται (όπως πάντα) η minimax
7. Στην minimax ελέγχεται αν κλήθηκε από τον τελευταίο agent, το οποίο ισχύει στο βήμα αυτό. Στην περίπτωση αυτή, καλείται η maxValue για τον pacman αυξάνοντας το επίπεδο (depth) κατά ένα, δηλαδή προχωράμε στο επόμενο επίπεδο στο οποίο ακολουθείται η ίδια διαδικασία από την αρχή.

* Προφανώς στην κάθε συνάρτηση γίνονται και άλλες λειτουργίες οι οποίες εξηγούνται παρακάτω. Στα παραπάνω βήματα εξηγείται μόνο το πώς λειτουργεί η αναδρομή και πως γίνονται οι κλήσεις μεταξύ των συναρτήσεων

## minimax

Η συνάρτηση αυτή δέχεται 3 ορίσματα. Την κατάσταση του παιχνιδιού (gameState), το id του agent για τον οποίο πρέπει να συνεχιστεί η διαδικασία (δηλαδή για τον οποίο πρέπει να καλέσει την κατάλληλη συνάρτηση) (agentId) και το τρέχον βάθος (depth).
Αρχικά ελέγχει το agentId. Επειδή οι συναρτήσεις που καλούν την minimax (minValue και maxValue) δίνουν για όρισμα agentId το επόμενο id από το δικό τους (δηλαδή αυξημένο κατά 1), ο τελευταίος agent δίνει σαν όρισμα ένα id μεγαλύτερο από τον αριθμό των agents. Επίσης τα id είναι συνεχόμενα από 0 μέχρι n όπου n ο αριθμός των φαντασμάτων και 0 είναι το id του pacman. Στην περίπτωση αυτή θέτουμε ως agentId το 0 και αυξάνουμε το βάθος κατά 1 καθώς είναι η σειρά του pacman (βήμα 7 που εξηγήθηκε παραπάνω).
Έπειτα από τον έλεγχο αυτό ελέγχουμε τις συνθήκες τερματισμού, δηλαδή αν νίκησε ο pacman αν έχασε ή αν φτάσαμε το επιθυμητό βάθος. Αν ισχύει μία από αυτές επιστρέφουμε το κόστος της τρέχουσας κατάστασης που έχει δημιουργηθεί μέσω της συνάρτησης evaluationFunction και ουσιαστικά τερματίζεται η αναδρομή.
Αν δεν ισχύουν οι συνθήκες τερματισμού, αν το agentId είναι 0 (δηλαδή σειρά του pacman) καλούμε την maxValue αλλιώς καλούμε την minValue για agentId φάντασμα.

## maxValue

Η maxValue δέχεται τα ίδια ορίσματα με την minimax. Βέβαια καλείται πάντα με agentId ίσο με 0 καθώς η maxValue εκτελείται πάντα για τον pacman.
Αρχικά παίρνουμε τις αποδεκτές κινήσεις που μπορεί μπορεί να κάνει ο pacman μέσω της συνάρτησης getLegalActions. Για κάθε μία από αυτές τις κινήσεις:
- Δημιουργούμε την νέα κατάσταση του παιχνιδιού έπειτα από την κίνηση αυτή μέσω της συνάρτησης generateSuccessor
- Καλούμε την minimax δίνοντας ως ορίσματα την νέα κατάσταση του παιχνιδιού και το id του πρώτου φαντάσματος (δηλαδή 1)

Κάθε φορά (δηλαδή για κάθε πιθανή κίνηση του pacman) η minimax θα μας επιστρέψει ένα tuple που περιέχει μία κίνηση (δηλαδή την κίνηση που έκανε το φάντασμα) και το αντίστοιχο value. Αγνοούμε την κίνηση που επιστρέφει η minimax καθώς μας ενδιαφέρει μόνο το value.
Επιλέγουμε την κίνηση του pacman για την οποία η minimax επέστρεψε το μεγαλύτερο value. Επιστρέφουμε ένα tuple με την κίνηση αυτή και το αντίστοιχο value που επέστρεψε η minimax.

## minValue

Ακολουθεί σχεδόν την ίδια διαδικασία με την maxValue. Βρίσκουμε τις αποδεκτές κινήσεις, για κάθε μία κίνηση παράγουμε την νέα κατάσταση του παιχνιδιού και καλούμε την minimax για τον επόμενο agent/φάντασμα (η για τον pacman στην περίπτωση του τελευταίου agent όπως εξηγήθηκε παραπάνω).
Η minimax όπως και πριν μας επιστρέφει μια κίνηση και ένα value. Επιλέγουμε την κίνηση (όχι από τις κινήσεις που επιστρέφει η minimax, αλλά από τις κινήσεις για τις οποίες καλέσαμε την minimax) για την οποία η minimax μας επέστρεψε το μικρότερο value (η διαφορά με την maxValue η οποία επιλέγει το μεγαλύτερο value).

## getAction

Η getAction (η οποία μας ζητείται να υλοποιήσουμε) χρειάζεται απλά να καλέσει τον αλγόριθμο minimax. Η κλήση γίνεται δίνοντας σαν ορίσματα το gameState, βάθος 0 και agentId 0 καθώς ο αλγόριθμος πρέπει να βρει μια κίνηση για τον pacman και επομένως να ξεκινήσει με αυτόν (δηλαδή με κλήση της maxValue).
Η getAction απλά επιστρέφει την κίνηση την οποία της επιστρέφει ο minimax.

**Question 3 - Alpha-Beta Pruning**

Ο αλγόριθμος είναι σχεδόν ολόιδιος με αυτόν του ερωτήματος 2 και ακολουθεί την ίδια λογική με αυτήν που εξηγήθηκε παραπάνω. Η μόνη διαφορά είναι ότι στις 3 συναρτήσεις minimax, minValue και maxValue έχουν προστεθεί τα ορίσματα α και β τα οποία απαιτούνται για το κλάδεμα. Επίσης μετά από κάθε κλήση της minimax εσωτερικά των συναρτήσεων minValue και maxValue έχει προστεθεί η συνθήκη ελέγχου για το κλάδεμα που αναφέρεται και στις διαφάνειες του μαθήματος. Δηλαδή στην maxValue ελέγχουμε αν το value που επέστρεψε η minimax είναι μεγαλύτερο από το β ενώ στην minValue ελέγχουμε αν το value που επέστρεψε η minimax είναι μικρότερο από το α. Αν δεν ισχύει η συνθήκη κλαδέματος τότε ενημερώνουμε τις τιμές των α και β. Εκτός απο αυτές τις διαφορές ο υπόλοιπος κώδικας είναι ο ίδιος.
Η αρχική κλήση της minimax γίνεται με ορίσματα: gameState, agentId=0 (δηλαδή τον pacman), βάθος 0, $\alpha = -\infty$ και $\beta = +\infty$ .

**Question 4 - Expectimax**

Ο αλγόριθμος αυτός είναι επίσης παρόμοιος με αυτόν του ερωτήματος 2. Οι συναρτήσεις minimax και maxValue είναι ακριβώς οι ίδιες και η μόνη διαφορά βρίσκεται στην συνάρτηση minValue. Η διαφορά είναι ότι στον αλγόριθμο αυτό υποθέτουμε ότι τα φαντάσματα δεν θα κάνουν απαραίτητα την καλύτερη κίνηση αλλά μπορεί να κάνει οποιαδήποτε κίνηση. Δηλαδή ότι κάθε πιθανή κίνηση ενός φαντάσματος έχει την ίδια πιθανότητα να συμβεί. Δηλαδή υποθέτοντας ότι το πλήθος των πιθανών κινήσεων ενός φαντάσματος είναι πλήθος_κινήσεων, η πιθανότητα της κάθε κίνησης να πραγματοποιηθεί από το φάντασμα είναι 1/πλήθος_κινήσεων. Έτσι η διαφορά του αλγορίθμου είναι ότι αντί να επιλέγει την κίνηση για την οποία ο minimax επέστρεψε το μικρότερο value, αθροίζει τα values που οι επιστρέφει η minimax για κάθε μια κίνηση πολλαπλασιασμένα με την πιθανότητα του καθενός να συμβεί (δηλαδή 1/πλήθος_κινήσεων).
Τέλος έχουμε υπολογίσει ένα εκτιμώμενο value για την κίνηση των φαντασμάτων (βάση της υπόθεσης ότι κάθε κίνηση είναι ισοπίθανο να συμβεί) το οποίο και επιστρέφουμε. Δεν επιστρέφουμε κάποια κίνηση καθώς με βάση της υπόθεσης αυτής, είναι προφανές ότι δεν μπορούμε να προβλέψουμε ποια από τις κινήσεις θα κάνει το κάθε φάντασμα.

**Question 5 - Evaluation Function**

Η συνάρτηση αυτή πρέπει να εκτιμά το πόσο καλό ή κακό είναι να state που δίνεται σαν όρισμα. Η εκτίμηση αυτή γίνεται επιστρέφοντας ένα value για το state αυτό. Όσο μεγαλύτερο είναι το value τόσο καλύτερη είναι και η κατάσταση αυτή, ενώ όσο μικρότερο είναι τόσο χειρότερη είναι η κατάσταση.
Η συνάρτηση αρχικά ελέγχει αν η δοθείσα κατάσταση είναι κατάσταση νίκης ή κατάσταση ήττας. Αν είναι κατάσταση νίκης επιστρέφει το που επιτεύχθηκε πολλαπλασιασμένο με 10000. Αυτό επιλέχθηκε καθώς θέλουμε σε περίπτωση νίκης να επιστρέφεται ένας πολύ μεγάλος αριθμός (καθώς η νίκη είναι η κατάσταση στην οποία θέλουμε να φτάσουμε). Όμως επειδή σε μια κατάσταση νίκης μπορούμε να φτάσουμε από διαφορετικά “μονοπάτια” με διαφορετικά score, θέλουμε να επιλέξουμε το “μονοπάτι” το οποίο μας οδηγεί στην νίκη με το μεγαλύτερο score. Έτσι πολλαπλασιάζουμε με το τρέχον score έτσι ώστε όσο μεγαλύτερο είναι το score που επιτεύχθηκε, τόσο μεγαλύτερος να είναι και ο αριθμός που επιστρέφεται. Η επιλογή του 10000 είναι τυχαία καθώς απλά χρειαζόμαστε έναν αρκετά μεγάλο αριθμό. Σε περίπτωση ήττας επιστρέφουμε $-\infty$ καθώς θέλουμε σε κάθε περίπτωση να αποφευχθεί η κατάσταση αυτή.
Αν δεν ισχύει καμία από τις 2 παραπάνω περιπτώσεις, προχωράμε στην εκτίμηση της κατάστασης. Αρχικά παίρνουμε της συντεταγμένες των φαγητών που έχουμε απομείνει και για κάθε ένα φαγητό υπολογίζουμε την απόσταση του από την τρέχουσα θέση του pacman μέσω manhattan distance. Αθροίζουμε αυτές τις αποστάσεις και και τις αποθηκεύουμε σε μία μεταβλητή foodDist.
Κάνουμε ακριβώς το ίδιο πράγμα και για τις κάψουλες (δηλαδή τα μεγάλα dots τα όποια άμα τα φάει ο pacman μπορεί να φάει τα φαντάσματα). Αποθηκεύουμε το άθροισμα αυτό σε μία μεταβλητή capsuleDist.

Είναι προφανές ότι όσο μεγαλύτερα είναι τα αθροίσματα αυτά, τόσο χειρότερη είναι η κατάσταση καθώς ο pacman βρίσκεται μακριά από τα φαγητά και τις κάψουλες. Θα φανεί στην συνέχεια πως θα χρησιμοποιήσουμε τα αθροίσματα αυτά.
Στην συνέχεια λαμβάνουμε υπόψιν τις θέσεις των φαντασμάτων. Υπολογίζουμε την απόσταση μεταξύ κάθε φαντάσματος και της τρέχουσας θέσης του φαντάσματος. Επίσης έχουμε μια μεταβλητή ghostDanger για να αποθηκεύσουμε το score σχετικά με τα φαντάσματα. Όσο μεγαλύτερη είναι η τιμή του ghostDanger τόσο καλύτερη είναι η κατάσταση (σε αντίθεση με τα foodDist και capsuleDist που εξηγήθηκαν παραπάνω).
Για κάθε φάντασμα ελέγχουμε το scaredTimer δηλαδή το αν ο pacman μπορεί να το φάει η όχι.
Αν το scaredTimer είναι θετικό (δηλαδή ο pacman μπορεί να φάει το φάντασμα) προσθέτουμε στο ghostDanger την τιμή 10/ghostDist όπου ghostDist η απόσταση του pacman από το φάντασμα. Αυτό σημαίνει ότι όσο πιο κοντά είναι pacman στο φάντασμα, τόσο μεγαλύτερη είναι και η τιμή αυτή. Επέλεξα την τιμή 10 καθώς δεν πρέπει να επηρεάζει κατά πολύ το score αφού δεν είναι σίγουρο για το αν ο pacman θα φάει το φάντασμα στην συνέχεια του παιχνιδιού η όχι. Έτσι αυξάνουμε ελάχιστα το score δείχνοντας ότι υπάρχει πιθανότητα μελλοντικά να φάει ένα φάντασμα και να κερδίσει επιπλέον πόντους.
Αντίθετα, αν το scaredTimer είναι ίσο με μηδέν σημαίνει ότι το φάντασμα μπορεί φάει τον pacman (δηλαδή να χάσει). Έτσι αφαιρούμε από το ghostDanger 100/ghostDist όπου ghostDist η απόσταση του pacman από το φάντασμα. Ο αριθμός αυτός γίνεται όλο και μεγαλύτερος όσο μικρότερη είναι η απόσταση μεταξύ pacman και φαντάσματος. Αφαιρούμε αυτήν την τιμή από το ghostDanger καθώς θεωρείται επικίνδυνο να βρίσκεται ο pacman κοντά σε ένα φάντασμα και μπορεί μελλοντικά να αποβεί μοιραίο. Επίσης αυξήσαμε τον συντελεστή σε 100 καθώς είναι σημαντικότερος παράγοντας γιατί μπορεί να χάσει λόγω αυτού (και για αυτό μειώνουμε και το συνολικό score).
Ο τελευταίος παράγοντας που επηρεάζει το συνολικό score είναι το τρέχον score που έχει πετύχει έως τώρα ο pacman (από τις προηγούμενες του κινήσεις). Παίρνουμε αυτό το score μέσω της συνάρτησης getScore.
Τέλος για να προκύψει το συνολικό score, προσθέτουμε τις εξής ποσότητες:

- Το τρέχον score του παιχνιδιού που μας επέστρεψε η getScore.
- Το άθροισμα foodDist πολλαπλασιασμένο με το -1. Ο πολλαπλασιασμός με το -1 γίνεται καθώς όσο μικρότερο είναι το άθροισμα τόσο καλύτερη είναι η κατάσταση. Έτσι ουσιαστικά το score μειώνεται λιγότερο για τις καλύτερες καταστάσεις ενώ μειώνεται περισσότερο για τις χειρότερες.
- Το άθροισμα capsuleDist πολλαπλασιασμένο με το -2. Ίδια λογική με το foodDist με την διαφορά ότι πολλαπλασιάζουμε με το -2 αντί για το -1 καθώς θέλουμε να δώσουμε λίγο μεγαλύτερη βαρύτητα στις κάψουλες από ότι στα φαγητά.
- Τέλος προσθέτουμε το άθροισμα ghostDanger. Δεν του αλλάζουμε πρόσημο καθώς όπως είπαμε όσο μεγαλύτερο είναι το ghostDanger τόσο καλύτερη είναι η κατάσταση. Επίσης έχουμε λάβει υπόψιν την βαρύτητα κατά τον υπολογισμό του αθροίσματος (δηλαδή με τους αριθμούς 10 και 100 που εξηγήθηκαν παραπάνω), οπότε δεν χρειάζεται να πολλαπλασιάσουμε το ghostDanger με κάποιον πολλαπλασιαστή βαρύτητας όπως κάναμε με την capsuleDist.

Έτσι έχω καταφέρει να υλοποιήσω μία συνάρτηση η οποία εκτιμά καλά τις καταστάσεις και επιτρέπει στον pacman να νικά πάντα στα test cases που γίνονται από τον autograder πετυχαίνοντας πολύ καλά scores σε πολύ καλούς χρόνους (Average Score: 1270.4). Σίγουρα θα μπορούσε να βελτιωθεί ακόμα περισσότερο, αλλά δεν θα είχε ουσιαστική διαφορά για την συγκριμένη άσκηση.

## **Παρατήρηση**

Δεν έχω αλλάξει κάτι στο αρχείο pacman.py οπότε δεν περιέχεται στο zipped αρχείο που παραδίδω.