email at: sdi2000055@di.uoa.gr

Περιεχομενα

1. Ασκηση 1

Για την ερώτηση 1, η τακτική που ακολούθησα είναι να προσπαθήσω να εκμεταλλευτώ όλα τα δεδομένα του προβλήματος για να βγάζω ένα καλό συμπέρασμα για το πακμαν. Ο πακμαν προσπαθεί να φάει κουκίδες φαγητού και να μην πεθάνει, δηλαδή να αποφεύγει τα φαντάσματα. Επομένως η λογική μου βασίστηκε πάνω σε αυτούς τους κανόνες ως ground rules επιλογής μιας καλής επιλογής.
Για την αρχική evaluation function δεν σκέφτηκα κάτι το τρομερό, αφού όπως αναφέρεται και στην εκφώνηση, το ζουμί δεν είναι εδώ. Επομένως απλά υπολόγισα τις ελάχιστες αποστάσεις από τα φαντάσματα και από το φαγητό και έπαιξα με κάποιους δείκτες και συντελεστές για να βγάζω καλά αποτελέσματα.
Όσο πιο κοντά είμαστε σε ένα φάντασμα τόσο το χειρότερο, άρα η πρόσθεση είναι μικρή, ενώ όσο πιο κοντά είμαστε σε φαγητό τόσο καλύτερα άρα διαιρώντας με έναν μικρό αριθμό, το κλάσμα μεγαλώνει και έτσι έχουμε καλύτερο evaluation. Τα παραπάνω τα προσθέτω στο υπάρχων σκορ που δίνεται από την .getScore().
Αν και είναι μια σχετικά πολύ απλή συνάρτηση, έχει καλά αποτελέσματα πετυχαίνωντας 4/4 στον Autograder.

2. Ασκηση 2

Η κρίσιμη ιδέα εδώ είναι ότι το ply αλλάζει πρακτικά μόνο όταν έχουν πάιξει και ο πάκμαν και τα φαντάσματα από 1 φορά. Άρα για ένα $depth = 2$, θα έχουν παίξει, πάκμαν και φαντάσματα, και τα δύο από δύο φορές.
Η υλοποίηση κατά τα άλλα δεν έχει μεγάλες διαφορές από τον κανονικό Minimax αλγόριθμο. Για να επιτύχουμε την εναλλαγή μεταξύ των παικτών καλώ συνέχεια την γενική συνάρτηση minimax η οποία ανάλογα με το depth και τον αριθμό παίκτη (0 αν μιλάμε για το πάκμαν, αλλιώς μιλάμε για κάποιο φάντασμα), σιγουρεύεται ότι καλεί τις σωστές υποσυναρτήσεις. Για το πάκμαν καλούμε την συνάρτηση maximize, και για τα άλλα φαντάσματα αντίστοιχα την συνάρτηση minimize. Για την εναλλαγή στο ply: Αν είμαι σε κάποιο ενδιάμεσο παίκτη, τότε απλά συνεχίζω στον επόμενο παραμένοντας στο ίδιο επίπεδο (nextPlayer++, depth = depth) αλλιώς αν βρίσκομαι στον τελευταίο παίκτη τότε ξέρω ότι στο επόμενο βήμα πρέπει να αλλάξω επίπεδο, άρα ξεκινάω πάλι από τον αρχικό παίκτη, και μειώνω το επίπεδο κατά 1 (χτίζω από

κάτω προς τα πάνω, nextPlayer = 0 (πάλι η σειρά του πάκμαν, άρα μαξ), depth = depth -1)

3. Ασκηση 3

Ομοίως με την άσκηση 2, το μόνο που αλλάζει είναι η προσθήκη των α, β για να μπορούμε να κλαδεύουμε τις επιλογές μας και να κάνουμε τον αλγόριθμο πιο γρήγορο, χωρίς να εξερευνεί κόμβους που δεν χρειάζεται. Ο αλγόριθμος και η υλοποίησή του είναι ίδια με αυτόν που διδαχτήκαμε στο μάθημα και τις διάλεξεις.

4. Ασκηση 4

Ομοίως με τους δύο παραπάνω αλγορίθμους. Αυτό που αλλάζει είναι ότι πλέον έχουμε κόμβους πιθανοτήτων για τις αποφάσεις που μπορεί να πάρει ένα ρομπότ. Η τιμή ενός κόμβου τύχης δίδεται από τον υπολογισμό του: $\frac{totalEvaluations}{numberofchoices}$. Αυτό επειδή όπως αναφέρεται και στην εκφώνηση: **which chooses amongst their getLegalActions uniformly at random.** Ακόμα επειδή μπορούμε να έχουμε περισσότερες από μια επιλογές που μεγιστοποιούν μια επιλογή μας, στον αλγόριθμο maximize βρίσκω το ποιές είναι αυτές και επιλέγω τυχαία εξ αυτών (σειρές: 350-357).

5. Ασκηση 5

Για την άσκηση 5, δεν άλλαξα πολλά πράγματα σε σχέση με την 1, αλλά άλλα λίγο την λογική μου (για να παίξω και με εναλλακτικούς τρόπους για να πετύχουμε ένα καλό evaluation)
Από όλα τα φαγητά, επιλέγω αυτό που είναι πιο κοντά μου. Επειδή σκοπός είναι να τρώει τα φαγητά, όσο πιο κοντά είμαι σε φαγητό τόσο το καλύτερο, και όσο πιο μακριά τόσο το χειρότερο. Επομένως έχω μια ανάλογη σχέση απόστασης φαγητού και πακμαν.
Από όλα τα φαντάσματα, επιλέγω αυτό που είναι πιο κοντά μου. Επειδή σκοπός μας είναι να μην φαγωθούμε από τα φαντάσματα, όσο πιο κοντά είμαι σε ένα φάντασμα τόσο το χειρότερο, και όσο πιο μακριά από ένα φάντασμα τόσο το καλύτερο. Άρα έχω μια αντιστρόφως ανάλογη σχέση απόστασης φαντάσματος και πάκμαν.
Ακόμα, επιδοτώ αν έφαγε ένα φάντασμα, με μεγαλύτερο δείκτη όμως, αφού το να φας ένα φάντασμα είναι αρκετά πιο σημαντικό να επιτευχθεί.
Τέλος, όλα αυτά τα αφαιρώ από το score της δεδομένης στιγμής που δίδεται από το $successorGameState.getScore()$.
Επομένως η γενική φόρμουλα υπολογισμού δίδεται από τον παρακάτω τύπο:

$$
\begin{aligned}
evaluation = successorGameState.getScore() \\
-\alpha \cdot \frac{1}{nearestGhostDistance + c} \\
-\beta \cdot \frac{nearestFoodDistance}{\delta} \\
+ateFood + ateGhost
\end{aligned}
$$

Για παράδειγμα, αν είμαι πολύ κοντά σε ένα φάντασμα το κλάσμα μεγαλώνει πολύ και επηρεάζει την αφαίρεση κατά πολύ, μικραίνωντας το evaluation άρα κάνοντάς το έτσι και πολύ χειρότερο (από π.χ. μια κατάσταση στην οποία είμαστε πολύ μακριά του φαντάσματος, άρα το κλάσμα μικραίνει και έχει μικρότερη επίδραση). Ομοίως αν είμαστε πολύ κοντά σε φαγητό, αφαιρούμε ένα μικρό (σχετικά πάντα) αριθμό που μας δίνει ένα καλό evaluation (μικρότερο), ενώ αν είμαστε πιο μακριά του φαγητού τότε αφαιρούμε ένα μεγαλύτερο αριθμό και επομένως χειροτερεύουμε το evaluation.

Για το ποιά ποσότητα επηρεάζει περισσότερο το evaluation, παίζουμε με τις σταθερές του τύπου. Μετά από αρκετό trial and error, κατέληξα σε μια λύση που μου δίνει πολύ καλά αποτέσματα, για σταθερές: $\alpha = 3, c = 1, \beta = 1, \delta = 4$, συγκεκριμένα στον Autograder δέχεται 6/6.