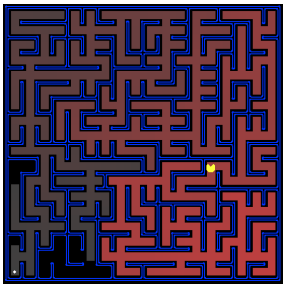
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837



Τεχνητή Νοημοσύνη Ι

Βασιλική Χριστοφιλοπούλου
1115202000216

Νοέμβριος 2024

Χειμερινό Εξάμηνο 2024-2025

Project 2

Table of Contents

1 Ερώτημα 1 : Reflex Agent

Για την υλοποίηση του ερωτήματος ακολουθήσα την εξης λογική:

1. Πρώτα ελέγχουμε αν κάποιο απο τα φαντάσματα είναι σε κατάσταση που μπορεί το packman να τα φάει, προκειμένου να κινηθούμε προς τα εκεί εφόσον ο χρόνος που απομένει σε αυτή την κατάσταση μας το επιτρέπει. Αυτή είναι η πρώτη μας κίνηση, καθώς αν το packman φάει ένα φάντασμα δίνονται περισσότεροι πόντοι.

2. Έπειτα, ελέγχουμε αν η επόμενη μας κίνηση είναι σε σημείο που υπάρχει φάντασμα ή το φάντασμα βρίσκεται σε απόσταση 1, προκειμένου να την αποφύγουμε καθώς είναι σίγουρο ότι θα χάσουμε.

3. Εφόσον δεν κινδυνεύουμε, ελέγχουμε αν στην επόμενη θέση υπάρχει φαγητό, καθως ο βασικός στόχος για να κερδίσουμε είναι να φαέι το packman όλο το φαγητό. Αν υπάρχει, τότε επιστρέφουμε μια μεγάλη τιμή ωστε να πάμε σε αυτό το σημείο.

4. Σε κάθε άλλη περίπτωση, υπολογίζεται η Manhattan απόσταση του τρέχοντος σημείου με το κοντινότερο σημείο που υπάρχει φαγητό, και επιστρέφεται η απόσταση αυτή.

Ο κώδικας δοκιμάστηκε όπως αναφέρεται και απο την εκφώνηση για τις ακόλουθες εντολές:

```
python pacman.py —frameTime 0 –p ReflexAgent –k 1
python autograder.py –q q1
```

2 Ερώτημα 2 : Minimax

Η γενική ιδέα του Minimax είναι ότι ο Pacman (που είναι ο MAX player) προσπαθεί να μεγιστοποιήσει το σκορ του, ενώ τα φαντάσματα (MIN players) προσπαθούν να το ελαχιστοποιήσουν

Για την υλοποίηση του αλγορίθμου χρησιμοποιήθηκε όμοια λογική με αυτή που παρουσιάζεται στις διαφάνεις Game Trees: Minimax σελίδα 19 του Berkeley CS188. Πιο αναλυτικά, δημιουργήθηκαν οι μέθοδοι:

- **value**
  Αυτή η μέθοδος αξιολογεί την κατάσταση του παιχνιδιού σε συγκεκριμένο βάθος και καθορίζει αν η τρέχουσα κατάσταση είναι τερματική ή αν πρέπει να υπολογιστεί με βάση τον Pacman (μέγιστο) ή τα φαντάσματα (ελάχιστο). Αν η κατάσταση είναι τερματική, επιστρέφει την τιμή της κατάστασης μέσω της evaluationFunction. Αν είναι η σειρά του Pacman, καλεί τη μέθοδο maxValue. Αν είναι σειρά των φαντασμάτων, καλεί τη minValue.

- **isTerminalState**
  Αυτή η μέθοδος ελέγχει αν η κατάσταση είναι τερματική, και επιστρέφει την τιμή true εφοσον είναι.

- **maxValue**
  Υπολογίζει τη μέγιστη δυνατή τιμή για τον Pacman για κάθε κατάσταση, ακολουθώντας τα ίδια βήματα οπως στις διαφάνειες. Για κάθε νόμιμη κίνηση του Pacman, δημιουργεί το επόμενο game state και καλεί τη value για να βρει την τιμή της κατάστασης από τον επόμενο πράκτορα (φαντάσματα). Στο τέλος, επιστρέφει τη μέγιστη τιμή από όλες τις κινήσεις, που είναι η βέλτιστη για τον Pacman.

- **minValue**
  Υπολογίζει τη χαμηλότερη δυνατή τιμή για τα φαντάσματα, τα οποία επιδιώκουν να μειώσουν το σκορ του Pacman, με βάση τα βήματα των διαφανειών. Για κάθε νόμιμη κίνηση των φαντασμάτων, δημιουργεί το επόμενο game state και: εάν το φάντασμα είναι το τελευταίο στη σειρά, αυξάνει το βάθος αλλίως ενημερώνει το agentIndex για να καθορίσει αν η επόμενη κίνηση είναι του Pacman ή άλλου φαντάσματος. Τέλος, επιστρέφει τη χαμηλότερη τιμή που αντιπροσωπεύει την πιο βλαβερή κίνηση για τον Pacman.

- **getAction**
  Η μέθοδος getAction ξεκινά τη διαδικασία Minimax και επιλέγει την καλύτερη δυνατή κίνηση για τον Pacman. Εξετάζει κάθε νόμιμη κίνηση (action) που μπορεί να κάνει ο Pacman. Δημιουργεί το επόμενο game state με βάση την κίνηση αυτή και υπολογίζει την τιμή της με τη μέθοδο value.Τελικά, επιστρέφει την καλύτερη κίνηση.

Ο κώδικας δοκιμάστηκε όπως αναφέρεται και απο την εκφώνηση για τις ακόλουθες εντολές:

```
python autograder.py –q q2
```

3 Ερώτημα 3 : Alpha-Beta Pruning

Για την υλοποίηση του αλγορίθμου Alpha-Beta Pruning χρησιμοποιήθηκε όμοια λογική με αυτή που χρησιμοποιήθηκε για την υλοποίηση του αλγορίθμου Minimax, και η ιδέα βασίστηκε στις διαφάνειες Game Trees: Minimax σελίδα 30 του Berkeley CS188. Πιο αναλυτικά, δημιουργήθηκαν οι μέθοδοι:

- **value**
  Είναι ίδια με αυτή του πορηγούμενου ερωτήματος.

- **isTerminalState**
  Αυτή η μέθοδος ελέγχει αν η κατάσταση είναι τερματική, και επιστρέφει την τιμή true εφοσον είναι.

- **maxValue**
  Η μέθοδος αυτή είναι όμοια με αυτή του ποηγούμενο ερωτήματος, με μόνη διαφορά ότι ελέγχεται αν $v \geq beta$, γίνεται pruning, καθώς γνωρίζουμε ότι ο αντίπαλος δεν θα επιλέξει μια κατάσταση χειρότερη από αυτή.

- **minValue**
  Αντίστοιχα με την maxValue, η αλλαγή στην minValue, είναι ότι αν $v \leq alpha$, γίνεται pruning, καθώς γνωρίζουμε ότι ο Pacman δεν θα επιλέξει μια κατάσταση χειρότερη από αυτή.

- **getAction**
  Όπως και στον προγούμενο ερώτημα, η μέθοδος αυτή επιστρέφει την καλύτερη κίνηση για τον Pacman στο αρχικό επίπεδο του δέντρου παιχνιδιού. Ο αλγόριθμος ξεκινά με $alpha = -\infty$ και $beta = +\infty$. Για κάθε νόμιμη κίνηση του Pacman, καλείται η value για την εκτίμηση της αξίας της επόμενης κατάστασης παιχνιδιού. Εάν η αξία αυτής της κίνησης είναι μεγαλύτερη από την τρέχουσα καλύτερη, η κίνηση αυτή ορίζεται ως η καλύτερη. Η alpha ενημερώνεται με την υψηλότερη τιμή που έχει βρεθεί έως τώρα, επιτρέποντας το άλφα-βήτα κλάδεμα.

Ο κώδικας δοκιμάστηκε όπως αναφέρεται και απο την εκφώνηση για τις ακόλουθες εντολές:

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
python autograder.py -q q3
```

4 Ερώτημα 4 : Expectimax

Για το συγκεκριμένο ερώτημα, βασίστηκα στην δομή του αλγορίθμου Minimax, και ακολούθησα την λογική που παρουσιάζεται στις διαφάνειες Game Trees: Expectimax, Utilities, Multiplayer σελίδα 12 του Berkeley CS188. Πιο αναλυτικά, δημιουργήθηκαν οι μέθοδοι:

- **value**
  Είναι ίδια με αυτή του ερωτηματος 2, με μονη διαφορά ότι για τα φαντάσματα αντί να κληθεί η συνάρτηση min καλείται η expValue.

  ```
  return self.expValue(gameState, depth, agentIndex)
  ```

- **isTerminalState**
  Είναι ίδια με αυτή του ερωτήματος 2.

- **maxValue**
  Είναι ίδια με αυτή του ερωτήματος 2.

- **expValue**
  Η συνάρτηση expValue λειτουργεί ως αναμενόμενη τιμή (expectimax) για τους πράκτορες-φαντάσματα, υπολογίζοντας την πιθανή "ποιότητα" κάθε κατάστασης του παιχνιδιού, βασισμένη στις κινήσεις τους με ισοπίθανες επιλογές, οπότε η πιθανότητα κάθε ενέργειας είναι p = 1 / len(actions). Για κάθε ενέργεια του φαντάσματος, δημιουργεί μια νέα κατάσταση παιχνιδιού successor με βάση αυτήν την ενέργεια. Επιστρέφεται η συνολική αναμενόμενη τιμή v, η οποία υπολογίζεται ως το γινόμενο της πιθανότητας p επί την τιμή της επόμενης κατάστασης.

- **getAction**
  Είναι ίδια με αυτή του ερωτήματος 2.

Ο κώδικας δοκιμάστηκε όπως αναφέρεται και απο την εκφώνηση για τις ακόλουθες εντολές:

```
python autograder.py -q q4
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

5 Ερώτημα 5 : Evaluation Function

Για την υλοποίηση του ερωτήματος αυτού ακολουθήθηκε όμοια δομή με αυτή του πρώτου ερωτήματος. Δηλαδή, υπολογίζουμε όλα τα στοιχεία που επηρεάζουν την απόφαση μας για την επόμενη επιλογή και τους αναθέτουμε αντίστοιχα βάρη ώστε να μπούν σε μιά σειρά προτεριαότητας. Πιο αναλυτικά,

1. Αρχικά ελέγχουμε αν βρισκόμαστε είτε σε κατάσταση νίκης είτε σε κατάσταση ήττας ώστε να επιστρέψουμε την αντίστοιχη τιμή.

2. Έπειτα, υπολογίζεται η απόσταση του packman από όλες τις θέσεις που περιέχουν φαγητό χρησιμοποιώντας την Manhattan distance. Τελικά, επιλέγεται η πλησιέστερη απόσταση (closestFood), η οποία θεωρείται ως σημαντικός παράγοντας για την αξιολόγηση, καθώς το Pacman θέλει να φάει το κοντινότερο φαγητό.

3. Στην συνέχεια, ασχολούμαστε με τα φαντάσματα όπου τα scared ghosts μπορούν να φαγωθούν ενω τα active ghosts όχι και απειλούν το Packman. Ανάλογα με την κατάσταση του κάθε φαντάσματος (αν είναι φοβισμένο ή όχι) και την απόστασή του από τον Pacman, οι αποστάσεις αυτές τοποθετούνται σε δύο ξεχωριστές λίστες. Ακολούθως, γίνεται η εύρεση της μικρότερης απόστασης για κάθε μία απο τις δύο λίστες.

4. Τέλος, ορίζουμε τα βάρη και υπολογίζουμε το τελικό score. Για τον υπολογισμο:

   - capsuleWeight: Δίνει μεγάλη αρνητική τιμή ανά κάψουλα, ενθαρρύνοντας τη συλλογή τους καθώς αν "φαγωθεί" ένα φάντασμα κερδίζονται περίσσοτεροι πόντοι.
   - foodWeight: Δίνει σχετικά υψηλή αρνητική τιμή ανά φαγητό, για να ενθαρρύνει τη συλλογή τους μέχρι να μηδενιστούν.
   - activeGhostWeight: Προσθέτει υψηλή αρνητική τιμή αν κάποιο ενεργό φάντασμα βρίσκεται κοντά, για να αποτρέψει την προσέγγιση του Pacman σε φαντάσματα.
   - closestFoodWeight: Δίνει προτεραιότητα στη συλλογή φαγητού αλλά με μικρή τιμή καθώς δεν είναι το πρωταρχικό μέλημα.
   - scaredGhostWeight: Προσθέτει ελαφρώς αρνητική τιμή για να παρακινήσει τον Pacman να κυνηγήσει τα φοβισμένα φαντάσματα.

   Η συνολική βαθμολογία επιστρέφεται από τη συνάρτηση, και όσο υψηλότερη είναι, τόσο καλύτερη θεωρείται η κατάσταση για τον Pacman.

Ο κώδικας δοκιμάστηκε όπως αναφέρεται και απο την εκφώνηση για τις ακόλουθες εντολές:

```
python autograder.py -q q5
```