ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Τεχνητή Νοημοσύνη

Κωνσταντίνος Κωτσαράς | 2022050 | it2022050@hua.gr

Εισαγωγή

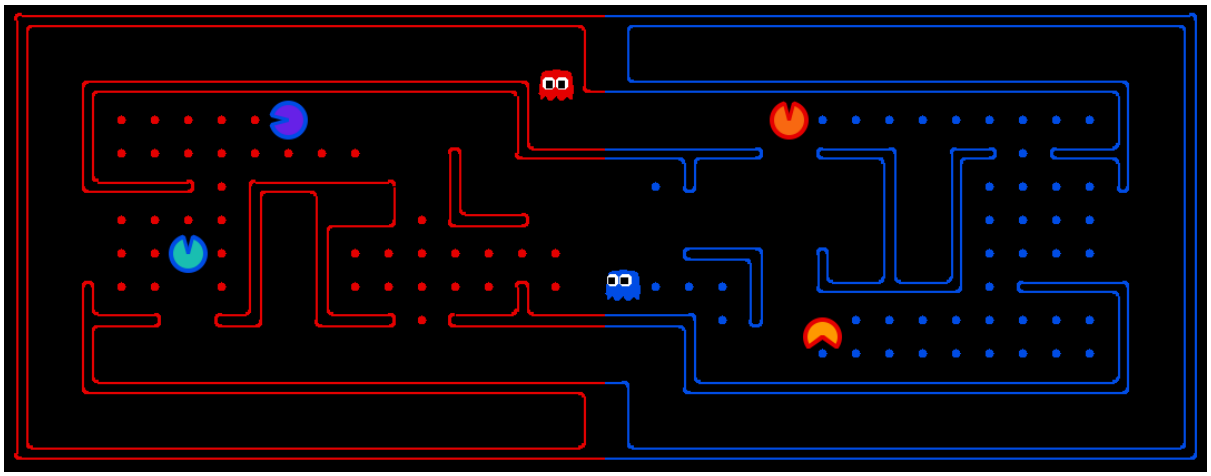
Η παρούσα αναφορά έχει ως στόχο την παρουσίαση των λύσεων που υλοποιήθηκαν στο πλαίσιο του **Project 1** του μαθήματος **Τεχνητή Νοημοσύνη**.

Σκοπός του έργου είναι η κατανόηση και εφαρμογή βασικών αλγορίθμων αναζήτησης σε προβλήματα δέντρων και γράφων, με αφορμή το κλασικό παιχνίδι Pacman. Συγκεκριμένα, ζητείται η υλοποίηση αλγορίθμων όπως ο **DFS** (Αναζήτηση Πρώτα σε Βάθος), ο **BFS** (Αναζήτηση Πρώτα σε Πλάτος), ο **UCS** (Ομοιόμορφο Κόστος) και ο **A*** (Αναζήτηση με Ευρετική).

Στη συνέχεια, δίνεται έμφαση στη βελτίωση των επιδόσεων μέσω ευρετικών συναρτήσεων και στην υλοποίηση πρακτόρων λήψης αποφάσεων, όπως ο **MinimaxAgent**, ο **AlphaBetaAgent** και ο **ExpectimaxAgent**, οι οποίοι καλούνται να παίξουν έξυπνα σε περιβάλλοντα με αντιπάλους.

Η αναφορά αναλύει βήμα-βήμα:

- Την προσέγγιση που ακολουθήθηκε για την επίλυση κάθε ερωτήματος
- Τις τεχνικές που χρησιμοποιήθηκαν
- Τα προβλήματα που προέκυψαν και πώς αντιμετωπίστηκαν
- Τα τελικά αποτελέσματα και παρατηρήσεις για τη συμπεριφορά των πρακτόρων



Ερωτήσεις

Ερώτηση 1

Στη λύση της αναζήτησης σε βάθος (DFS), κάθε κόμβος του δέντρου αναζήτησης αναπαρίσταται ως ένα ζευγάρι (state, path). Το state είναι αυτό που παίρνουμε αρχικά από το **problem.getInitialState()** και στη συνέχεια από κάθε **getNextStates()**, και περιγράφει την τρέχουσα κατάσταση στο πρόβλημα, δηλαδή πού βρίσκεται ο Pacman (ποια είναι η θέση στον χώρο αναζήτησης). Το path είναι μια λίστα από ενέργειες (όπως 'North', 'East', κλπ.) που δείχνει τον δρόμο που ακολουθήσαμε από την αρχική κατάσταση μέχρι το συγκεκριμένο state/ κατάσταση. Με αυτόν τον τρόπο, μπορώ να γνωρίζω τόσο πού βρίσκομαι όσο και πώς έφτασα εκεί, κάτι που μου επιτρέπει να επιστρέφω τη σωστή ακολουθία ενεργειών όταν φτάσω σε μια κατάσταση στόχο. Επιπλέον, το ότι αποθηκεύω το (state, path) μαζί σε ένα tuple βοηθάει πολύ στο πώς χρησιμοποιώ τη στοίβα (**Stack**) στον αλγόριθμο. Κάθε φορά που κάνω **push** ή **pop**, παίρνω ταυτόχρονα και την τρέχουσα κατάσταση και τη διαδρομή που ακολούθησα για να φτάσω εκεί. Έτσι, δεν χρειάζεται να ξαναυπολογίζω τη διαδρομή κάθε φορά.

Output:

```
C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Ερώτηση 2

Η διαφορά ανάμεσα στον **BFS** και τον **DFS** οφείλεται στον τρόπο που εξερευνούν και προσπελάσουν τους κόμβους. Ο **DFS** (Depth First Search) ακολουθεί πρώτα τα πιο βαθιά μονοπάτια, όπως λέει και το όνομα του αλγορίθμου, προσπαθώντας να φτάσει όσο πιο μακριά γίνεται πριν επιστρέψει πίσω για να δοκιμάσει άλλη διαδρομή. Αυτό μπορεί να τον οδηγήσει να ψάχνει πολλές άσχετες ή μη χρήσιμες καταστάσεις σε μεγάλο βάθος και να μην βρίσκει πάντα τη βέλτιστη λύση. Αντίθετα, ο **BFS** (Breadth First Search) επεκτείνει πρώτα τις πιο κοντινές στον αρχικό κόμβο καταστάσεις, εξερευνώντας το δέντρο αναζήτησης επίπεδο-επίπεδο. Με αυτόν τον τρόπο βρίσκει πάντα τη λύση με τα λιγότερα δυνατά βήματα, αλλά χρειάζεται περισσότερη μνήμη, καθώς αποθηκεύει περισσότερους κόμβους. Συμπερασματικά, ο **DFS** μπορεί να φτάσει γρήγορα σε κάποια λύση, αλλά όχι πάντα στη βέλτιστη, ενώ ο **BFS** βρίσκει τη συντομότερη διαδρομή, αν και εξερευνά περισσότερους κόμβους και χρησιμοποιεί περισσότερη μνήμη.

Output:

```
C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Ερώτηση 3

Η συνάρτηση κόστους του StayEastSearchAgent είναι η **costFn = lambda pos: 0.5 ** pos[0]**, δηλαδή το κόστος μειώνεται όσο ο Pacman κινείται προς τα ανατολικά (δεξιά), αφού για μεγαλύτερες τιμές του x το κόστος γίνεται μικρότερο. Έτσι, ο Pacman ευνοείται να κινηθεί δεξιά για να ελαχιστοποιήσει το συνολικό κόστος. Αντίθετα, η συνάρτηση κόστους του StayWestSearchAgent είναι η **costFn = lambda pos: 2 ** pos[0]**, όπου το κόστος αυξάνεται όσο ο Pacman κινείται ανατολικά (μεγαλύτερες τιμές του x), άρα για να έχει χαμηλότερο κόστος, ο Pacman προτιμά να κινείται προς τα δυτικά (αριστερά). Συνοπτικά, ο StayEastSearchAgent κατευθύνει τον Pacman προς τα ανατολικά, ενώ ο StayWestSearchAgent τον κατευθύνει προς τα δυτικά, ανάλογα με τον τρόπο που έχει οριστεί το κόστος στις αντίστοιχες συναρτήσεις.

Output:

```
C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Ερώτηση 4

Για τον λαβύρινθο **bigMaze**, ο αλγόριθμος UCS (Uniform Cost Search) επεκτείνει 620 κόμβους, ενώ ο αλγόριθμος A* με ευρετική Manhattan επεκτείνει 549 κόμβους. Αντίστοιχα, για τον **openMaze**, ο UCS επεκτείνει 682 κόμβους, ενώ ο A* επεκτείνει μόνο 535 κόμβους. Αυτό συμβαίνει γιατί ο UCS (g(n)) δεν χρησιμοποιεί κάποια ευρετική πληροφορία που θα τον βοηθήσει να φτάσει πιο κοντά στον στόχο και επεκτείνει όλα τα μονοπάτια με βάση μόνο το κόστος, ενώ ο A* (h(n)) καθοδηγείται από την Manhattan ευρετική, η οποία εκτιμά πόσο κοντά βρίσκεται κάθε κατάσταση στον στόχο. Έτσι, ο A* επικεντρώνει την αναζήτηση σε πιο υποσχόμενες/ κερδοφόρες περιοχές του λαβύρινθου και καταφέρνει να βρει τη λύση με λιγότερη διερεύνηση του χώρου. Τέλος, με τον αλγόριθμο A* επειδή επεκτείνουμε σε λιγότερους κόμβους, έχουμε λιγότερους υπολογισμούς, λιγότερη κατανάλωση μνήμης, λιγότερο χρόνο στην CPU και λιγότερη κατανάλωση ενέργειας.

BigMaze Output:

```
C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=ucs,heuristic=manhattanHeuristic
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

OpenMaze Output:

```
C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=ucs,heuristic=manhattanHeuristic
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Ερώτηση 5

Η ευρετική συνάρτηση foodHeuristic που υλοποίησα βασίζεται σε δύο κύρια σημεία: υπολογίζει τη μικρότερη απόσταση από τη θέση του Pacman σε κάποιο κομμάτι φαγητού μέσω BFS, και επιπλέον υπολογίζει τη μέγιστη απόσταση μεταξύ δύο κομματιών φαγητού(έτσι ερμηνεύω ότι ο pacman για να φτάσει μέχρι εκεί κατά μεγάλη πιθανότητα θα έχει συναντήσει και άλλη τροφή στον δρόμο του). Το άθροισμα αυτών των δύο αποστάσεων χρησιμοποιείται ως εκτίμηση για το κόστος επίτευξης του στόχου. Η ευρετική θεωρείται αποδεκτή και συνεπής, διότι:

- Ποτέ δεν υπερεκτιμά το πραγματικό κόστος μετακίνησης (είναι δηλαδή υποεκτίμηση ή ακριβής εκτίμηση).
- Η τιμή της μειώνεται ή παραμένει σταθερή όσο πλησιάζουμε στο στόχο, τηρώντας τον κανόνα της συνέπειας.

Στις δοκιμές μου στον λαβύρινθο trickySearch, ο αλγόριθμος A* με την foodHeuristic επεκτείνει 719 κόμβους, ενώ ο UCS χωρίς ευρετική επεκτείνει 16688 κόμβους. Αυτό δείχνει ότι η χρήση της ευρετικής μειώνει κατά πολύ τον αριθμό των κόμβων που χρειάζεται να εξερευνηθούν, καθιστώντας την αναζήτηση πολύ πιο αποδοτική τόσο σε χρόνο όσο και σε υπολογιστικούς πόρους.

Output:

```
C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 1.1 seconds
Search nodes expanded: 719
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\kotsa\Desktop\Hua\6ο_Εξάμηνο\AI\project_1>python pacman.py -l trickySearch -p SearchAgent -a fn=ucs,prob=FoodSearchProblem
[SearchAgent] using function ucs
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 60 in 1.0 seconds
Search nodes expanded: 16688
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Ερώτηση 6

Παρόλο που ο πράκτορας Pacman χρησιμοποιεί τον αλγόριθμο **Minimax** για τη λήψη αποφάσεων, μπορεί να χάνει το παιχνίδι σε κάποιες περιπτώσεις, επειδή η απόδοσή του

εξαρτάται από τη στρατηγική που υλοποιεί και τους περιορισμούς του ίδιου του μοντέλου. Ο Minimax υποθέτει ότι οι αντίπαλοι (φαντάσματα) παίζουν και ενεργούν πάντα βέλτιστα για να μειώσουν τη βαθμολογία του Pacman. Αυτό σημαίνει πως σε κάθε κατάσταση, τα φαντάσματα επιλέγουν την ενέργεια που προκαλεί τη μεγαλύτερη ζημιά ή θα μειώσει το score στον Pacman. Ωστόσο, στην πραγματικότητα τα φαντάσματα μπορεί να κινούνται διαφορετικά ή ακόμα και μη βέλτιστα, κάτι που κάνει τις προβλέψεις του Minimax λιγότερο αξιόπιστες σε ορισμένες περιπτώσεις. Επιπλέον, ο Minimax εκτελείται μέχρι ένα συγκεκριμένο βάθος (όπως ορίζεται από τη μεταβλητή **self.depth**), με αποτέλεσμα να μη λαμβάνει υπόψη του μακροπρόθεσμες συνέπειες ή στρατηγικές που εξελίσσονται πιο αργά. Αν ο κίνδυνος από τα φαντάσματα εμφανίζεται σε μεγαλύτερο βάθος απ' όσο εξετάζει ο αλγόριθμος, τότε ο pacman ενδέχεται να μην τον προβλέψει εγκαίρως και να κινηθεί προς μια επικίνδυνη περιοχή χωρίς να το αντιλαμβάνεται. Ακόμα, σε ορισμένες περιπτώσεις παρατηρούμε ότι ο Pacman "περιμένει" τα φαντάσματα να πλησιάσουν. Αυτό μπορεί να συμβαίνει είτε γιατί δεν υπάρχει διαθέσιμη ασφαλής διαδρομή προς το φαγητό, είτε γιατί το μοντέλο του Minimax βρίσκει προσωρινά βέλτιστο να μην προχωρήσει και να μείνει στην ίδια θέση, βασισμένο σε βραχυπρόθεσμους υπολογισμούς. Ο Pacman προτιμά να αποφύγει άμεση επαφή με τα φαντάσματα, αλλά αν το βάθος είναι μικρό, δεν μπορεί να αντιληφθεί ότι μένοντας ακίνητος δίνει στα φαντάσματα χρόνο να τον πλησιάσουν και τελικά να τον περικυκλώσουν, με αποτέλεσμα να χάσει.

Συνοψίζοντας, ο Minimax πράκτορας μπορεί να χάνει επειδή:

- Υποθέτει ιδανική (βέλτιστη) συμπεριφορά των φαντασμάτων, ενώ στην πράξη αυτή μπορεί να διαφέρει(επειδή μπορεί να μην παίζουν βέλτιστα).
- Περιορίζεται από το μέγιστο βάθος αναζήτησης, αγνοώντας μακροπρόθεσμες συνέπειες.

Ερώτηση 7

Η συμπεριφορά του MinimaxAgent, όπου ο Pacman επιλέγει να χάσει το συντομότερο δυνατό όταν δεν υπάρχει τρόπος να κερδίσει, σχετίζεται με τον τρόπο λειτουργίας του αλγόριθμου **Minimax**. Συγκεκριμένα, η συνάρτηση αξιολόγησης (evaluationFunction) χρησιμοποιείται για να βαθμολογεί τις "τερματικές" καταστάσεις, είτε αυτές είναι νίκες είτε ήττες. Στην περίπτωση μιας ήττας, η συνάρτηση επιστρέφει μία πολύ αρνητική τιμή (π.χ. -αριθμός ή πολύ μικρό σκορ). Έτσι, αν ο Pacman καταλάβει ότι δεν μπορεί να αποφύγει την ήττα, τότε θα επιλέξει τη στρατηγική που οδηγεί στη λιγότερο κακή έκβαση – δηλαδή να χάσει όσο το δυνατόν πιο σύντομα, αποφεύγοντας ενδιάμεσες καταστάσεις που θεωρεί ότι απλώς παρατείνουν την ήττα χωρίς κάποιο όφελος. Ωστόσο, αυτή η στρατηγική μπορεί να αποδειχθεί προβληματική/λάθος σε περιπτώσεις όπου ο Pacman δεν διαθέτει αρκετή πληροφόρηση για να κάνει ακριβή πρόβλεψη. Για παράδειγμα, όταν το βάθος αναζήτησης είναι μικρό, ο πράκτορας δεν μπορεί να δει αρκετά μακριά στο μέλλον ώστε να εντοπίσει μία πιθανή νίκη/σωτηρία. Αν, για παράδειγμα, μια σωτήρια πορεία απαιτεί περισσότερα βήματα απ' όσα επιτρέπει το βάθος, τότε ο Pacman θα την αγνοήσει, επειδή δεν θα μπορεί να την δει. Επιπλέον, ένα ακόμα πρόβλημα μπορεί να προκύψει αν η συνάρτηση αξιολόγησης δεν είναι αρκετά έξυπνη ώστε να διακρίνει τη διαφορά ανάμεσα σε καταστάσεις που οδηγούν σε άμεση ήττα και άλλες που προσφέρουν μια ευκαιρία επιβίωσης. Αν η αξιολόγηση δεν

λαμβάνει υπόψη έστω και μια μικρή πιθανότητα σωτηρίας, τότε ο Pacman ενδέχεται να αγνοήσει διαδρομές που θα μπορούσαν να τον σώσουν. Συνοψίζοντας, ο MinimaxAgent παίρνει αποφάσεις που είναι βέλτιστες με βάση τα δεδομένα και τους περιορισμούς του (evaluationFunction και του βάθος του). Ωστόσο, η περιορισμένη πληροφορία ή οι μη-σωστές προβλέψεις μπορούν να τον οδηγήσουν σε συμπεριφορές που φαίνονται υπερβολικά απαισιόδοξες. Παρόλο που ο αλγόριθμος εφαρμόζεται σωστά, δίνει την εντύπωση ότι ο Pacman τα παρατάει, χωρίς να προσπαθήσει να βρει μια διέξοδο.

Ερώτηση 8

Η συνάρτηση αξιολόγησής μου στο πλαίσιο του παιχνιδιού Pacman έχει ως στόχο να υπολογίσει τη "χρησιμότητα" μιας συγκεκριμένης κατάστασης, δηλαδή να βαθμολογήσει αν η κατάσταση είναι καλή ή κακή για τον Pacman, ώστε να πάρει την καλύτερη απόφαση.

Η λειτουργία της **betterEvaluationFunction()** βασίζεται σε μια σειρά παραμέτρων που επηρεάζουν τη στρατηγική του πράκτορα:

- **Κομμάτια Φαγητού:** Εάν ο Pacman έχει καταναλώσει περισσότερα κομμάτια φαγητού, η συνάρτηση αξιολόγησης επιστρέφει μια θετική τιμή, η οποία υποδεικνύει πρόοδο και πλεονέκτημα για τον πράκτορα. Όσο περισσότερα κομμάτια φαγητού έχει καταναλώσει, τόσο υψηλότερη είναι η βαθμολογία του.
- **Απόσταση από τα Φαντάσματα:** Η συνάρτηση ελέγχει την απόσταση του Pacman από κάθε φάντασμα. Αν το φάντασμα δεν είναι φοβισμένο και βρίσκεται πολύ κοντά (λιγότερο από 3 μονάδες), η αξιολόγηση γίνεται αρνητική, καθώς θεωρείται επικίνδυνο. Ωστόσο, η ποινή εφαρμόζεται μόνο αν δεν υπάρχει τοίχος ανάμεσα στον Pacman και το φάντασμα, δηλαδή αν υπάρχει πραγματικός κίνδυνος σύγκρουσης.
- **Απόσταση από τα Κομμάτια Φαγητού:** Η κοντινή απόσταση του Pacman σε κομμάτια φαγητού επηρεάζει τη βαθμολογία. Εάν είναι κοντά σε φαγητό, η συνάρτηση αξιολόγησης επιστρέφει υψηλή βαθμολογία, υποδεικνύοντας ότι ο Pacman είναι κοντά σε έναν στόχο.
- **Κατάσταση Κινδύνου:** Αν ο Pacman κινδυνεύει να χάσει από τα φαντάσματα, η συνάρτηση επιστρέφει μια πολύ αρνητική τιμή. Αν ο Pacman βρίσκεται σε επικίνδυνη θέση, προσπαθεί να απομακρυνθεί και η συνάρτηση θα το δείξει με μια αρνητική βαθμολογία.
- **Κοντά σε Φοβισμένα Φαντάσματα:** Όταν τα φαντάσματα είναι φοβισμένα, ο Pacman μπορεί να τα φάει, κερδίζοντας πόντους. Η συνάρτηση αξιολόγησης επιβραβεύει τον Pacman όταν πλησιάζει φοβισμένα φαντάσματα, αφού μπορεί να τα “φάει” και να αυξήσει το σκορ του.
- **Απόσταση από Κάψουλες:** Η συνάρτηση αξιολόγησης επιβραβεύει τον Pacman όταν πλησιάζει σε κάψουλες. Η στρατηγική του να συλλέγει τις κάψουλες είναι επίσης ενισχυμένη, με θετική βαθμολογία για κοντινές κάψουλες.

Ανακεφαλαιώνοντας, η συνάρτηση αξιολόγησης υπολογίζει την αξία κάθε κατάστασης, λαμβάνοντας υπόψη τη θέση του Pacman, τα φαντάσματα και το φαγητό, έτσι ώστε να

βοηθήσει τον πράκτορα να επιλέξει τις καλύτερες κινήσεις που οδηγούν σε θετικότερες εκβάσεις, όπως η κατανάλωση φαγητού και η αποφυγή του θανάτου.

Output:

```
C:\Users\kotsa\Desktop\Hua\6o_Εξάμηνο\AI\project_1>py -3.6 autograder.py -q q9 --no-graphics
Starting on 5-16 at 19:13:11

Question q9
===========
Pacman emerges victorious! Score: 1076
Pacman emerges victorious! Score: 1086
Pacman emerges victorious! Score: 1096
Pacman emerges victorious! Score: 822
Pacman emerges victorious! Score: 1090
Pacman emerges victorious! Score: 1151
Pacman emerges victorious! Score: 1347
Pacman emerges victorious! Score: 1120
Pacman emerges victorious! Score: 907
Pacman emerges victorious! Score: 1102
Average Score: 1079.7
Scores:        1076.0, 1086.0, 1096.0, 822.0, 1090.0, 1151.0, 1347.0, 1120.0, 907.0, 1102.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q9\grade-agent.test (6 of 6 points)
***     1079.7 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q9: 6/6 ###


Finished at 19:13:23
```

Κατά την εκτέλεση του πράκτορα με Alpha-Beta Pruning, παρατηρήθηκε ιδιαίτερα καλή απόδοση, καθώς στις περισσότερες (ή και όλες) τις εκτελέσεις ο Pacman κέρδισε. Η στρατηγική του πράκτορα είναι προσεκτική και στοχεύει στη νίκη και στο υψηλό σκορ, αποφεύγοντας περιττούς κινδύνους. Όταν καταναλώνει κάψουλες, επιδιώκει να κυνηγήσει τα φοβισμένα φαντάσματα ώστε να αυξήσει σημαντικά το σκορ του. Δεν παίζει επιθετικά χωρίς λόγο, αλλά φαίνεται να περιμένει τις κινήσεις των φαντασμάτων, αποφεύγοντας την παγίδευση. Επιπλέον, όταν βρίσκεται κοντά σε φαντάσματα και δεν υπάρχουν τοίχοι να τον προστατεύσουν, απομακρύνεται σωστά, δείχνοντας καλή αντίληψη κινδύνου. Οι κινήσεις του είναι έξυπνες και καλά υπολογισμένες, ακόμη και σε καταστάσεις πίεσης, κάτι που αποδεικνύει ότι ο συνδυασμός της συνάρτησης αξιολόγησης με το Alpha-Beta Pruning αποδίδει αποτελεσματικά.

Η κύρια διαφορά είναι ότι ο MinimaxAgent του ερωτήματος 6 χρησιμοποιεί τη default συνάρτηση αξιολόγησης, που βασίζεται αποκλειστικά στο score του παιχνιδιού. Αντίθετα, ο AlphaBetaAgent χρησιμοποιεί τη δική μου συνάρτηση αξιολόγησης better, η οποία λαμβάνει υπόψη πρόσθετα στοιχεία όπως η απόσταση από το πλησιέστερο φαγητό, η θέση των φαντασμάτων, η θέση των capsules και άλλοι παράγοντες. Επιπλέον, ο AlphaBetaAgent εφαρμόζει beta pruning για να μειώσει το χώρο αναζήτησης, στην ουσία δεν εξερευνεί όλα

τα branches όπως ο minimax,με αποτέλεσμα να είναι πιο αποδοτικός υπολογιστικά. Συνολικά, ο AlphaBeta πράκτορας παίρνει πιο "έξυπνες" αποφάσεις και εκτελεί ταχύτερα.

Autograder