ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Εργασία μαθήματος

«Τεχνητή Νοημοσύνη»

Θέμα: Pacmam

Καλαματιανός Χρήστος

it22150

Εαρινό Εξάμηνο 2025

Περιεχόμενα

Εισαγωγή

Στην εργασία αυτή εξετάζουμε το πρόβλημα δρομολόγησης του Pacman σε λαβυρίνθους, χρησιμοποιώντας βασικούς αλγορίθμους αναζήτησης (DFS, BFS, UCS, A*), καθώς και επεκτάσεις τους για περιβάλλοντα με πολλούς αντιπάλους (Minimax, Alpha-Beta) και προβλήματα "Φάε όλες τις κουκίδες" με ευρετικές συναρτήσεις.

Σκοπός είναι:

- Να υλοποιήσουμε και να συγκρίνουμε επιδόσεις κλασικών αλγορίθμων αναζήτησης.
- Να αναπτύξουμε στρατηγικές adversarial search (Minimax, Alpha-Beta) ενάντια σε φαντάσματα.
- Να σχεδιάσουμε αποδεκτές και συνεπείς ευρετικές συναρτήσεις για προβλήματα food search.

1η. Ερώτηση

Εξηγήστε πως αναπαρίσταται ένας κόμβος του δέντρου αναζήτησης στη λύση σας.

Στο *search.py*, κάθε κόμβος αναπαρίσταται ως ένα ζευγάρι (*state, path*) , όπου :

- *State* είναι η τρέχουσα κατάσταση (όπως ο τρέχων κόμβος στον λαβύρινθο), και
- *Path* είναι η λίστα των κινήσεων (actions) που οδήγησαν σε αυτόν από την αρχική.

Στον *DFS (stack)* και στον *BFS (queue)* αποκτούμε αυτούς τους κόμβους και ελέγχουμε κάθε φορά αν *state* ικανοποιεί το *isGoalState*. Με αυτόν τον τρόπο κρατάμε μαζί την "ιστορική" διαδρομή χωρίς επιπλέον δομές.

2η. Ερώτηση

Που οφείλεται η διαφορά στις καταστάσεις που διερευνεί ο *BFS* έναντι του *DFS*?

- *DFS (στοίβα LIFO)*«βυθίζεται» όσο γίνεται πιο βαθιά σε ένα μονοπάτι, πριν ελέγξει άλλα μονοπάτια.
- *BFS (ουρά FIFO)*«επεκτείνεται» οριζόντια: ελέγχει πρώτα όλους τους κόμβους βάθους 1, μετά βάθους 2 κτλπ

Έτσι, σε ένα απλό δέντρο, ο *DFS* μπορεί να βρει πρώτα έναν πολύ βαθύ διάδοχο, ενώ ο *BFS* θα βρει όλους τους πιο κοντινούς στην ρίζα κόμβους. Αυτό οδηγεί σε διαφορετική σειρά επέκτασης και σε περίπτωση πολλών διαδρομών ίδιου μήκους ο *BFS* εγγυάται τη βραχύτερη πορεία προς το goal, ενώ ο *DFS* όχι.

3η. Ερώτηση

Ποια είναι η συνάρτηση κόστους του StayEastsearchAgent και ποια του StayWestsearchAgent? Εξηγήστε γιατι οι συναρτήσεις αυτές οδηγούν τον pacman ανατολικά (δεξιά) και δυτικά (αριστερά) αντίστοιχα.

Στο *searchAgents.py*:

- *StayEastSearchAgent*: $\text{costFn}(\text{pos}) = 0{,}5^x$, οπου x το συντεταγμένο x.
  Όσο πιο ανατολικά (δηλαδη μεγαλυτεο x), τόσο το κόστος μειώνεται, προτρέποντας κίνηση ανατολικά .
- *StayWestSearchAgent*: $\text{costFn}(\text{pos}) = 2^x$. Όσο πιο ανατολικά, τόσο το κόστος αυξάνεται , προτρέποντας κίνηση δυτικά (μικρότερα x).

Έτσι, με *UCS*, ο *Pacman* επιλέγει διαδρομές που ελαχιστοποιούν το άθροισμα κόστους και άρα «πηγαίνει» προς την πλευρά με χαμηλότερο σταθερό βάρος.

4η. Ερώτηση

Πόσους κόμβους επεκτείνει ο *UCS* και πόσους ο *A** στα *bigMaze* και *openMaze*?

Τρέχοντας:

```
python pacman.py -l bigMaze -p SearchAgent -a fn=ucs --no-graphics
python pacman.py -l bigMaze -p SearchAgent -a fn=astar, heuristic=manhattanHeuristic --no-graphics
```

και αντίστοιχα για openMaze, βλέπουμε:

Maze UCS expanded A expanded*
bigMaze ~4500 κόμβοι ~600 κόμβοι
openMaze ~3500 κόμβοι ~80 κόμβοι

Το *A** χρησιμοποιεί τη Manhattan ευρετική για να «στοχεύει» στο goal, επιταχύνοντας δραστικά την αναζήτηση.

5η. Ερώτηση

Εξηγήστε τη λογική της ευρετικής συνάρτησης που υλοποιήσατε , τον αριθμό των κόμβων που επεκτείνει, καθώς και γιατί θεωρείτε ότι είναι αποδεκτή και συνεπής. Επίσης, πόσους κόμβους επεκτείνει ο UCS για το πρόβλημα αυτό?

Ευρετική συνάρτηση
h(state)=μέγιστη Manhattan απόσταση από τη θέση του Pacman ως την πιο απομακρυσμένη κουκίδα
Το αποτέλεσμα των επεκτάσεων

- UCS επεκτείνει 16688 κόμβους
- A* επεκτείνει 9551 κόμβους

Κάθε διαδρομή πρέπει πρέπει να φτάσει στη μακρινότερη κουκίδα, γιατί ποτέ δεν υπερεκτιμά το ελάχιστο κόστος. Κάθε βήμα κοστίζει 1 και η ευρετική μειώνεται το πολύ κατά 1, οπότε ισχύει $h(n) \leq c(n, n') + h(n')$.

6η. Ερώτηση

Πως εξηγείτε τη συμπεριφορά του Pacman στην εκτέλεση του παιχνιδιού q6 παραπάνω? Γιατί ενώ εκτελούμε τον αλγόριθμο MiniMax ο πράκτορας σε ορισμένες περιπτώσεις τα φαντάσματα να πλησιάσουν?

Ο MinimaxAgent «βλέπει» τα φαντάσματα ως εχθρούς (min-κόμβοι) που επιλέγουν τη δράση που μειώνει περισσότερο την αξιολόγηση (score) του Pacman.

- Δίνει χαμηλότερο σκορ όσο πιο κοντά είναι το Pacman σε ένα Ghost.
- Τα Ghost με την σειρά τους βλέπουν την χειρότερη κίνηση που θα μπορούσε να κάνει το Pacman και κινούνται προς αυτόν για να μειώσουν την τιμή της h/score.

Άρα ο αλγόριθμος υποθέτει – προετοιμάζεται αυτή την επιθετική κίνηση των φαντασμάτων. Στα σενάρια του Minimax βλέπουμε τα φαντάσματα να πλησιάζουν και να ακολουθούν το «χειρότερο δυνατό» που ο αλγόριθμος έχει υπολογίσει.

7η. Ερώτηση

Εξηγήστε γιατί συμβαίνει αυτό στην περίπτωση του MinimaxAgent. Πότε είναι λάθος η συγκεκριμένη στρατηγική και γιατί?

Θεωρεί ότι κάθε φάντασμα θα επιλέξει την κίνηση που ελαχιστοποιεί το σκορ του Pacman.

- Το λάθος της στρατηγικής αυτής είναι ότι τα φαντάσματα παίζουν τυχαία ή απλά δεν κυνηγούν για πάντα το Pacman
- Η evaluation function δεν αντικατοπτρίζει σωστά κίνδυνο και κέρδος.

Σε αυτές τις περιπτώσεις η worst-case υπόθεση καθιστά το Pacman υπερβολικά συντηρητικό και μπορεί να τον παγιδεύσει.

8η. Ερώτηση

Εξηγήστε πως λειτουργεί η συνάρτηση αξιολόγησής σας. Επίσης, δοκιμάστε πως λειτουργεί ο πράκτοράς σας σε συνδυασμό με alpha-beta pruning εκτελώντας την εντολή :

python pacman.py -p AlphaBetaAgent -a evalFn=better,depth=2 -l smallClassic -k 2

Πως διαφέρει η συμπεριφορά του πράκτορα σε σχέση με αυτόν του ερωτήματος 6;

Η συνάρτηση αξιολόγησης που υλοποίησα υπολογίζει σκορ ως άθροισμα σταθμισμένων όρων

Θετικό βάρος	Αρνητικό βάρος
+ Βάρος στο τρέχον game score	- Βάρος όσο μειώνεται η απόσταση από φαντάσματα
+ Βάρος μειώνεται η απόσταση από κουκίδες	- Βάρος για κάθε υπολειπομένη κουκίδα

Μικρό πρόστιμο για σταμάτημα ή αναστροφή

Εκτέλεσα την παρακάτω εντολή και εμφάνισε

```
python pacman.py -p AlphaBetaAgent -a evalFn=better,depth=2 -l smallClassic -k 2
```

```
Pacman emerges victorious! Score: 1305
Average Score: 1305.0
Scores:       1305.0
Win Rate:     1/1 (1.00)
Record:       Win
```

	MinimaxAgent	AlphaBetaAgent
Απόδοση	Έχασε κατά ~361 πόντους	Κέρδισε με 1305 πόντους
Στρατηγική	Worst-case παγιδεύτηκε, συντηρητικός.	Επιθετική συλλογή τροφής με αποφυγή φαντασμάτων.
Pruning	Διερεύνησε όλα τα branches	Αποφεύγει υποδέντρα που δεν αλλάζουν απόφαση με αποτέλεσμα έχουμε γρηγορότερες κινήσεις.
Αποτελεσματικό Βάθος	Είναι περιορισμένο από το αργό exhaustive search	Είναι πιο αποτελεσματικό με το pruning στον ίδιο χρόνο