Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο
ΥΠ23 Τεχνητή Νοημοσύνη

Εργασία 1: Αναζήτηση

Έκδοση 2023-1.0

Διδάσκων: Χρήστος Δίου

1 Εισαγωγή

Σ' αυτή την εργασία θα χρησιμοποιήσουμε τον κόσμο του Pacman για να εξοικειωθούμε με τους αλγόριθμους αναζήτησης. Στόχος του Pacman είναι να σχεδιάσει τη διαδρομή προς συγκεκριμένες τοποθεσίες, καθώς και να συλλέξει τροφή (κουκίδες) με αποτελεσματικό τρόπο.

Σας δίνεται ένα αρχείο `1_search.tar.gz`. Από τα αρχεία που προκύπτουν μετά την αποσυμπίεση, θα χρειαστεί να επεξεργαστείτε τα `search.py` και `searchAgents.py`. Επιπλέον, θα χρειαστεί να μελετήσετε και τα αρχεία `pacman.py`, `game.py` και `util.py`, χωρίς ωστόσο να τα αλλάξετε.

Όπως και στο tutorial, μπορείτε να ελέγχετε τις λύσεις σας μέσω του autograder. Παρακαλώ ανατρέξτε στο tutorial για λεπτομέρειες.

1.1 Παραδοτέα και αξιολόγηση

Η υποβολή της εργασίας σας περιλαμβάνει τα αρχεία `search.py`, `searchAgents.py` καθώς και συνοπτικές απαντήσεις στα ερωτήματα της εργασίας τις οποίες θα υποβάλλετε σε ένα συνοδευτικό αρχείο PDF. Όταν τα αρχεία σας (τα `search.py` και `searchAgents.py`) αντικαταστήσουν τα αντίστοιχα αρχεία του `1_search.tar.gz` θα πρέπει να επιτρέπουν την εκτέλεση του autograder. Η ορθή λειτουργία των αρχείων σας θα γίνει αυτοματοποιημένα, επομένως φροντίστε ο κώδικάς σας να εκτελείται σωστά. Επίσης φροντίστε τα αρχεία να έχουν το σωστό όνομα (να ονομάζονται `search.py` και `searchAgents.py` και όχι κάτι άλλο).

Ο κώδικάς σας επίσης θα ελεγχθεί για την ορθότητά του (μπορεί να υπάρχουν σφάλματα ακόμα και αν τρέχουν σωστά τα tests του autograder), καθώς και για πιθανή αντιγραφή.

Σε περίπτωση που διαπιστωθεί ότι η λύση που παραδώσατε δεν είναι δική σας, τότε η εργασία θα μετρήσει *αρνητικά* στη βαθμολογία σας.

2 Ο κόσμος του Pacman

Αποσυμπιέστε το αρχείο `1_search.tar.gz` και μπείτε στον φάκελο `1_search`. Μπορείτε να παίξετε ένα παιχνίδι με την εντολή

```
python pacman.py
```

Οι πράκτορες περιγράφονται στο αρχείο `searchAgents.py`. Ο απλούστερος είναι ο `GoWestAgent` που πάντα πηγαίνει προς τα δυτικά και είναι ένας πράκτορας που λειτουργεί αντανακλαστικά. Χρησιμοποιήστε τον με:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

Ο `GoWestAgent` δε λειτουργεί καλά σε ένα λίγο πιο σύνθετο περιβάλλον:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

Δείτε όλες τα ορίσματα της γραμμής εντολών του pacman με

```
python pacman.py -h
```

3 Ασκήσεις

Άσκηση 1 (3 μονάδες): Αναζήτηση πρώτα σε βάθος (DFS)

Το αρχείο `SearchAgents.py` περιλαμβάνει έναν υλοποιημένο πράκτορα, τον `SearchAgent` ο οποίος σχεδιάζει τη διαδρομή του μέσα στο κόσμο του Pacman και την ακολουθεί. Λείπουν ωστόσο οι αλγόριθμοι σχεδιασμού της διαδρομής, τους οποίους πρέπει να υλοποιήσετε εσείς.

Αρχικά ελέγξτε ότι ο πράκτορας λειτουργεί σωστά εκτελώντας την εντολή

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

Η εντολή αυτή χρησιμοποιεί το `tinyMazeSearch` σαν αλγόριθμο αναζήτησης, ο οποίος υλοποιείται στο `search.py`. Ο αλγόριθμος αυτός έχει πρακτικά κωδικοποιημένη απευθείας τη λύση για τον λαβύρινθο `tinyMaze`.

Υλοποιήστε τον αλγόριθμο αναζήτησης πρώτα σε βάθος (DFS) στη συνάρτηση `depthFirstSearch` στο αρχείο `search.py`. Η υλοποίησή σας θα πρέπει να αφορά αναζήτηση σε γράφους, όπου ο πράκτορας επισκέπτεται κάθε κατάσταση το πολύ μία φορά.

Λάβετε υπόψη σας τα ακόλουθα:

- Οι συναρτήσεις σας θα πρέπει να επιστρέφουν μία λίστα από ενέργειες που οδηγούν τον πράκτορα από την αρχική κατάσταση στο στόχο.
- Είναι απαραίτητο να χρησιμοποιήσετε τις κλάσεις `Stack`, `Queue` και `PriorityQueue` που βρίσκονται στο `utils.py` ώστε η λύση σας να είναι συμβατή με τον autograder.

Η λύση σας θα πρέπει να βρίσκει γρήγορα μία λύση στις ακόλουθες περιπτώσεις

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
```

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs
```

Η κάθε λύση δείχνει επίσης τις καταστάσεις που διερευνήθηκαν, όπου με έντονο χρώμα απεικονίζονται οι θέσεις που εξετάστηκαν πρώτες.

Οι λύσεις στις επόμενες ασκήσεις θα ακολουθούν την ίδια λογική, μόνο που αλλάζει ο τρόπος που διαχειρίζεστε το μέτωπο αναζήτησης.

Ερώτηση 1: Εξηγήστε πως αναπαρίσταται ένας κόμβος του *δέντρου* αναζήτησης στη λύση σας.

Άσκηση 2 (3 μονάδες): Αναζήτηση πρώτα σε πλάτος (BFS)

Όπως και προηγουμένως, μόνο που τώρα υλοποιείτε την αναζήτηση πρώτα σε πλάτος στη συνάρτηση `breadthFirstSearch`. Ελέγξτε την υλοποίησή σας με τις εντολές

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=bfs
```

Άσκηση 3 (3 μονάδες): Αναζήτηση ομοιόμορφου κόστους (UCS)

Στην περίπτωση αυτή εξετάζουμε προβλήματα όπου το κόστος μεταβάλλεται ανάλογα με την περιοχή του λαβύρινθου που κινείται ο Pacman. Δείτε για παράδειγμα τους λαβυρίνθους `mediumDottedMaze` και `mediumScaryMaze`. Μπορεί για σημεία με πολύ φαγητό το κόστος να είναι μικρότερο, ενώ σε σημεία όπου υπάρχουν φαντάσματα το κόστος να είναι μεγαλύτερο.

Υλοποιήστε τον αλγόριθμο αναζήτησης ομοιόμορφου κόστους στη συνάρτηση `uniformCostSearch` στο αρχείο `search.py`. Δείτε τη συμπεριφορά των παρακάτω πρακτόρων του Pacman (οι συναρτήσεις που υπολογίζουν το κόστος σας δίνονται έτοιμες)

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

Δείτε το αρχείο `SearchAgents.py` για λεπτομέρειες σχετικά με τις συναρτήσεις κόστους των παραπάνω πρακτόρων.

Άσκηση 4 (3 μονάδες): A*

Υλοποιήστε τον αλγόριθμο A* για αναζήτηση σε γράφους στη συνάρτηση `aStarSearch` του `search.py`, η οποία δέχεται την ευρετική συνάρτηση ως όρισμα. Οι ευρετική συνάρτηση δέχεται δύο ορίσματα, την τρέχουσα κατάσταση και το πρόβλημα (δείτε και τον κώδικα σχετικά).

Ελέγξτε την υλοποίησή σας χρησιμοποιώντας την απόσταση Manhattan που ήδη έχει υλοποιηθεί ως `manhattanHeuristic` στο αρχείο `searchAgents.py`.

```
python pacman.py -l bigMaze -z .5 -p SearchAgent \
  -a fn=astar,heuristic=manhattanHeuristic
```

Παρατηρήστε ότι ο A* να βρίσκει τη λύση λίγο γρηγορότερα από τον UCS. Κάντε δοκιμές και στο λαβύρινθο `openMaze`.

Ερώτηση 2: Πόσους κόμβους απαιτεί ο UCS και ο A* στον `bigMaze`; Πόσοι είναι οι αντίστοιχοι κόμβοι για τον `openMaze`;

Άσκηση 5 (3 μονάδες): Συντομότερο μονοπάτι για όλες τις γωνίες

Η αξία του A* γίνεται περισσότερο εμφανής σε ένα πιο σύνθετο πρόβλημα αναζήτησης.

Έστω ότι έχουμε τέσσερις κουκίδες, μία σε κάθε γωνία του λαβυρίνθου. Στόχος του αλγόριθμου αναζήτησης είναι να βρει το συντομότερο μονοπάτι που περνά από όλες τις γωνίες.

Η λύση αυτής της Άσκησης βασίζεται στη λύση της Άσκησης 2. Υλοποιήστε την `CornersProblem` στο αρχείο `searchAgents.py`. Θα χρειαστεί να επιλέξετε μία αναπαράσταση καταστάσεων που εντοπίζει αν ο Pacman έχει περάσει από όλες τις γωνίες. Θα πρέπει πλέον να λειτουργούν τα ακόλουθα:

```
python pacman.py -l tinyCorners -p SearchAgent \
-a fn=bfs,prob=CornersProblem

python pacman.py -l mediumCorners -p SearchAgent \
-a fn=bfs,prob=CornersProblem
```

Η αναπαράσταση που θα χρησιμοποιήσετε για τις καταστάσεις θα πρέπει να κωδικοποιεί μόνο την απαραίτητη πληροφορία για την επίλυση του προβλήματος και τίποτε άλλο.

Άσκηση 6 (3 μονάδες): Συντομότερο μονοπάτι για όλες τις γωνίες με ευρετική συνάρτηση

Υλοποιήστε μία συνεπή ευρετική συνάρτηση για το `CornersProblem` στην `cornersHeuristic`.

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
```

Το `AStarCornersAgent` είναι συντομογραφία για το

```
-p SearchAgent \
-a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
```

Η ευρετική συνάρτησή σας δεν πρέπει να είναι τετριμμένη, όπου τετριμμένες ευρετικές συναρτήσεις είναι αυτή που επιστρέφει παντού μηδέν και αυτή που επιστρέφει το πραγματικό κόστος σε κάθε θέση. Επίσης πρέπει να επιστρέφει μηδέν στο στόχο και θετικές τιμές οπουδήποτε αλλού. Τέλος, σημειώνεται ότι στην αναζήτηση A* σε γράφους, η συνάρτησή σας δεν αρκεί να είναι αποδεκτή, αλλά πρέπει να είναι και συνεπής ώστε να εξασφαλίζεται η εύρεση της βέλτιστης λύσης.

Σημείωση: *Για την άσκηση αυτή, ΜΗ χρησιμοποιήσετε τη συνάρτηση* `mazeDistance`.

Η βαθμολόγηση εξαρτάται από τους κόμβους του γράφου που επεκτείνονται, σύμφωνα με τον πίνακα

Αριθμός κόμβων	Βαθμός
> 2000	0
≤ 2000	1
≤ 1600	2
≤ 1200	3

Ερώτηση 3: Εξηγήστε τη λογική της ευρετικής συνάρτησης που υλοποιήσατε, τον αριθμό των κόμβων που επεκτείνει, καθώς και γιατί θεωρείτε ότι είναι αποδεκτή και συνεπής.

Άσκηση 7 (4 μονάδες): Φάε όλες τις κουκίδες

Στην άσκηση αυτή λύνουμε το πρόβλημα "Φάε όλες τις κουκίδες" που είδαμε και στο μάθημα. Στόχος είναι ο Pacman να φάει όλο το φαγητό του με τον μικρότερο δυνατό αριθμό βημάτων.

Θα χρειαστούμε ένα νέο πρόβλημα, το `FoodSearchProblem` στο αρχείο `SearchAgents.py` (έχει υλοποιηθεί). Για τη συγκεκριμένη άσκηση οι λύσεις δε λαμβάνουν υπόψη τους τα φαντάσματα και τα power pellets (τις μεγάλες κουκίδες). Με τον A* που ήδη υλοποιήσατε με μηδενική ευρετική συνάρτηση (ισοδύναμα, με τον UCS), θα πρέπει να μπορείτε άμεσα να βρείτε τη βέλτιστη λύση:

```
python pacman.py -l testSearch -p AStarFoodSearchAgent
```

όπου το `AStarFoodSearchAgen` είναι συντομογραφία του

```
-p SearchAgent \
-a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic
```

Δυστυχώς ο UCS καθυστερεί πολύ, ακόμα και για απλούς λαβυρίνθους όπως ο `tinySearch`. Για να επιταχύνετε τη διαδικασία συμπληρώστε το `foodHeuristic` στο αρχείο `searchAgents.py` με μία συνεπή, μη μηδενική και μη αρνητική ευρετική συνάρτηση για το `FoodSearchProblem`. Δοκιμάστε τον πράκτορά σας στον λαβύρινθο `trickySearch`:

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

Σημείωση: *Για την άσκηση αυτή, ΜΗ χρησιμοποιήσετε τη συνάρτηση* `mazeDistance`.

Ανάλογα με την επίδοση του αλγορίθμου σας θα βαθμολογηθείτε ως εξής:

Αριθμός κόμβων	Βαθμός
> 15000	1
≤ 15000	2
≤ 1200	3
≤ 9000	4
≤ 7000	5 (bonus)

Σημειωτέον, αν η ευρετική συνάρτησή σας είναι μη συνεπής δε θα πάρετε καμία μονάδα από την άσκηση. Αν ο αλγόριθμός σας βρίσκει πολύ γρήγορα τη λύση ίσως να εξετάσετε αν η συνάρτησή σας είναι συνεπής ή όχι.

Ερώτηση 4: Εξηγήστε τη λογική της ευρετικής συνάρτησης που υλοποιήσατε, τον αριθμό των κόμβων που επεκτείνει, καθώς και γιατί θεωρείτε ότι είναι αποδεκτή και συνεπής.

Άσκηση 8 (3 μονάδες): Υποβέλτιστη αναζήτηση

Συχνά στην πράξη ακόμα και ο A* με μία καλή ευρετική συνάρτηση η εύρεση του βέλτιστου μονοπατιού είναι δύσκολη. Στις περιπτώσεις αυτές μπορεί να πρέπει να συμβιβαστούμε με κάποιον πράκτορα που βρίσκει μία υποβέλτιστη λύση γρήγορα. Στην ενότητα αυτή θα χρησιμοποιήσετε έναν πράκτορα που χρησιμοποιεί άπληστη αναζήτηση, και τρώει την κοντινότερη κουκίδα.

Αυτός έχει ήδη υλοποιηθεί ως `ClosestDotSearchAgent` στο αρχείο `searchAgents.py`, αλλά του λείπει η συνάρτηση `findPathtoClosestDot`. Υλοποιήστε τη συνάρτηση και εξετάστε τη συμπεριφορά του πράκτορα στο πρόβλημα

```
python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
```

Θα πρέπει να λύνει το πρόβλημα πολύ γρήγορα, αλλά με υποβέλτιστο τρόπο.

Βοήθεια: Ο γρηγορότερος τρόπος να συμπληρώστε την `findPathToClosestDot` είναι να συμπληρώσετε την `AnyFoodSearchProblem` η οποία δεν έχει υλοποιημένη τη συνθήκη επίτευξης στόχου. Τότε λύστε αυτό το πρόβλημα με μία κατάλληλη συνάρτηση αναζήτησης. Η λύση θα είναι πολύ σύντομη.

Ερώτηση 5: Ο `ClosestDotSearchAgent` δε θα βρίσκει πάντα την καλύτερη λύση. Σκεφτείτε γιατί και βρείτε ένα παράδειγμα όπου συμβαίνει αυτό.