Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο
ΥΠ23 Τεχνητή Νοημοσύνη

Εργασία 1: Αλγόριθμοι αναζήτησης

Έκδοση 2025-1.0

Διδάσκων: Χρήστος Δίου

1 Εισαγωγή

Σ' αυτή την εργασία θα χρησιμοποιήσουμε τον κόσμο του Pacman για να εξοικειωθούμε με τους αλγόριθμους αναζήτησης (με ή χωρίς αντιπάλους). Στόχος του Pacman είναι να σχεδιάσει τη διαδρομή προς συγκεκριμένες τοποθεσίες, να συλλέξει τροφή (κουκίδες) με αποτελεσματικό τρόπο και, αν υπάρχουν φαντάσματα, ο Pacman πρέπει να φάει όλες τις κουκίδες αποφεύγοντας τα φαντάσματα.

Σας δίνεται ένα αρχείο `project_1.tar.gz`. Από τα αρχεία που προκύπτουν μετά την αποσυμπίεση, θα χρειαστεί να επεξεργαστείτε τα `search.py`, `searchAgents.py` και `multiAgents.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 στο κατάλληλο περιβάλλον python (3.6). Παρακαλώ ανατρέξτε στο tutorial για λεπτομέρειες.

1.1 Παραδοτέα και αξιολόγηση

Η υποβολή της εργασίας σας θα πρέπει να αποτελείται από ένα αρχείο `zip` ή `tar.gz` που θα περιλαμβάνει τα αρχεία `search.py`, `searchAgents.py`, `multiAgents.py` καθώς και συνοπτικές απαντήσεις στα ερωτήματα της εργασίας τις οποίες θα υποβάλλετε σε ένα συνοδευτικό αρχείο PDF. Δώστε σημασία στα παρακάτω:

- Η υποβολή σας *δεν* πρέπει να περιλαμβάνει αρχεία πέραν των παραπάνω
- Μην αλλάξετε τα ονόματα των αρχείων
- Η υποβολή των αρχείων θα πρέπει να γίνει μέσω ενός και μόνο αρχείου zip ή tar.gz το οποίο θα έχει όνομα της μορφής `it[id].tar.gz`, όπου `[id]` είναι ο ΑΜ σας. Για παράδειγμα, `it20001.tar.gz` για ΑΜ 20001
- Όταν αποσυμπιεστεί το αρχείο πρέπει να δημιουργεί φάκελο με `it[id]` και περιεχόμενα μόνο τα παραπάνω αρχεία
- Όταν τα αρχεία `search.py`, `searchAgents.py` και `multiAgents.py` αντικαταστήσουν τα αντίστοιχα αρχεία του `project_1.tar.gz` θα πρέπει να επιτρέπουν την εκτέλεση του autograder
- Φροντίστε ο κώδικάς σας να εκτελείται σωστά σε περιβάλλον python όπως αυτό που προσδιορίζεται στο tutorial. Επίσης φροντίστε τα αρχεία να έχουν το σωστό όνομα.

Ο κώδικάς σας θα ελεγχθεί 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 `project_1.tar.gz` και μπείτε στον φάκελο `project_1`. Μπορείτε να παίξετε ένα παιχνίδι με την εντολή

```
python pacman.py
```

Οι πράκτορες περιγράφονται στα αρχεία `searchAgents.py` και `multiAgents.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 Ερωτήματα

Ερώτημα 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 υλοποίησή 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 στα επόμενα ερωτήματα θα ακολουθούν την ίδια λογική, μόνο που αλλάζει ο τρόπος που διαχειρίζεστε το σύνορο αναζήτησης.

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

Ερώτημα 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
```

Ερώτηση 2 (για αναφορά): Που οφείλεται η διαφορά στις καταστάσεις που διερευνεί ο BFS έναντι του DFS;

Ερώτημα 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 λαβυρίθνους `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.

Ερώτηση 3 (για αναφορά): Ποια είναι η συνάρτηση κόστους του `StayEastsearchAgent` και ποια του `StayWestsearchAgent`; Εξηγηστε γιατί οι συναρτήσεις αυτές οδηγούν τον pacman ανατολικά (δεξιά) και δυτικά (αριστερά) αντίστοιχα.

Ερώτημα 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 4 (για αναφορά): Πόσους κόμβους απαιτεί ο UCS και ο A* στον `bigMaze`; Πόσοι είναι οι αντίστοιχοι κόμβοι για τον `openMaze`;

Ερώτημα 5 (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 `AStarFoodSearchAgent` 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:

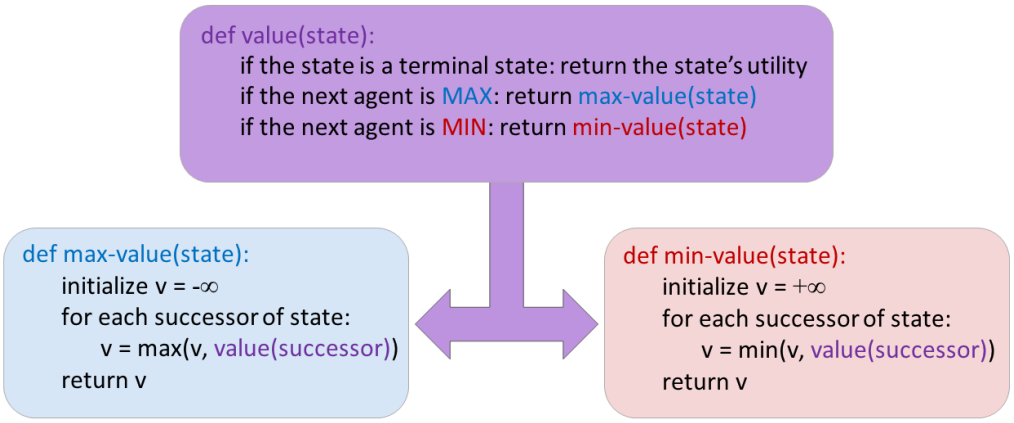
Αριθμός κόμβων	Βαθμός
> 15000	1
≤ 15000	2
≤ 1200	3
≤ 9000	4
≤ 7000	5 (bonus)

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 5 (για αναφορά): 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;

Ερώτημα 6 (3 μονάδες): Minimax

Υλοποιήστε την κλάση `MinimaxAgent` που δίνεται στο αρχείο `multiAgents.py`. Θυμηθείτε τον αλγόριθμο Minimax που είδαμε στο μάθημα:



Πρακτικά θα χρειαστεί να υλοποιήσετε τον παραπάνω ψευδοκώδικα (δείτε και τις διαφάνειες του μαθήματος) με δύο σημαντικές διαφορές:

1. Θα πρέπει ο αλγόριθμος να λειτουργεί σωστά όταν υπάρχουν και πολλοί πράκτορες min (περισσότερα από ένα φαντάσματα). Σκεφτείτε πως θα γίνει αυτό και προσαρμόστε τον κώδικα ανάλογα
2. Ο αλγόριθμος θα πρέπει να δέχεται και περιορισμό στο βάθος αναζήτησης. Συγκεκριμένα, η κλάση `MinimaxAgent` κληρονομεί την `MultiAgentSearchAgent` η οποία έχει τις ιδιότητες `depth` και `evaluationFunction`. Όταν δέχεστε περιορισμό στο βάθος, θα πρέπει ο αλγόριθμος να επιστρέφει την αξία που δίνεται από την `evaluationFunction` όταν φτάσετε στο μέγιστο βάθος (δηλ. θα θεωρεί τις καταστάσεις ως τερματικές)

Σημειώστε ότι θεωρούμε ότι προχωράμε κατά βάθος 1 όταν έχουν εκτελέσει μία ενέργεια ο Pacman και όλα τα φαντάσματα. Ελέγξτε και διορθώστε τον κώδικά σας καλώντας

```
python autograder.py -q q6
```

Αν θέλετε να τρέξει πιο γρήγορα, χωρίς γραφικά, καλέστε

```
python autograder.py -q q6 --no-graphics
```

Ερώτηση 6 (για αναφορά): Πως εξηγείτε τη συμπεριφορά του Pacman στην εκτέλεση του παιχνιδιού `q6` παραπάνω; Γιατί ενώ εκτελούμε τον αλγόριθμο MiniMax ο πράκτορας μας μπορεί να χάνει; Γιατί ο πράκτορας σε ορισμένες περιπτώσεις περιμένει τα φαντάσματα να πλησιάσουν;

Σημειώσεις:

- Όπως και στον ψευδοκώδικα του Minimax, θα χρειαστεί να υλοποιήσετε επιπλέον βοηθητικές συναρτήσεις στην κλάση `MinimaxAgent`.
- Σε ορισμένα τεστ του autograder ο Pacman θα χάνει. Αυτό είναι φυσιολογικό.
- Η συνάρτηση αξιολόγησης σας δίνεται έτοιμη και είναι η `evaluationFunction`. Μην την αλλάξετε. Σημειώστε ότι η συνάρτηση αυτή αξιολογεί καταστάσεις και όχι ενέργειες.
- Ο Pacman έχει πάντα `agentIndex` ίσο με `0`, ενώ τα φαντάσματα κινούνται με αύξοντα αριθμό του `agentIndex`.

Όταν ο Pacman βλέπει ότι θα χάσει, προσπαθεί να χάσει το συντομότερο δυνατό.

```
python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

Ερώτηση 7 (για αναφορά): Εξηγήστε γιατί συμβαίνει αυτό στην περίπτωση του `MinimaxAgent`. Πότε είναι λάθος η συγκεκριμένη στρατηγική και γιατί;

Έχοντας ολοκληρώσει επιτυχώς αυτή την άσκηση, οι λύσεις στα επόμενα ερωτήματα είναι πιο απλές.

Ερώτημα 7 (3 μονάδες): Alpha-Beta pruning

Εδώ καλείστε να υλοποιήσετε τον `AlphaBetaAgent`, ο οποίος χρησιμοποιεί alpha-beta pruning ώστε να επιταχύνει τους υπολογισμούς του αλγόριθμου Minimax. Δείτε τον ψευδοκώδικα.

α: MAX καλύτερη επιλογή προς τη ρίζα
β: MIN καλύτερη επιλογή προς τη ρίζα

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v > β return v
        α = max(α, v)
    return v
```

```
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

Σημαντικές παρατηρήσεις:

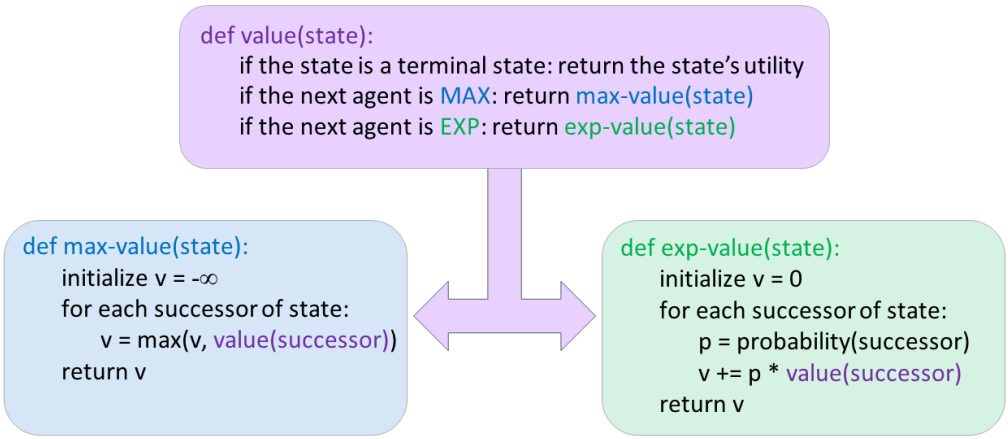
- Ο αλγόριθμος δεν πρέπει να κλαδεύει στην ισότητα, δηλ. χρησιμοποιούμε `v > β` και όχι `v >= β` (αντίστοιχα `v < α`). Ο λόγος που το κάνουμε αυτό είναι για να μπορούμε να επιλέξουμε την κατάλληλη ενέργεια όταν έχουμε πολλές ενέργειες που οδηγούν στην ίδια αξία (δείτε τις διαφάνειες). Άλλες μέθοδοι που λύνουν αυτό το πρόβλημα (όπως το να εκτελούμε τον αλγόριθμο χωριστά για κάθε ενέργεια του Pacman) δεν είναι συμβατές με τον autograder.
- Δεν πρέπει να αλλάξετε τη σειρά με την οποία εξετάζονται οι ενέργειες. Πρέπει να παραμείνουν όπως τις επιστρέφει η `GameState.getAvailableActions`, ώστε οι λύσεις σας να είναι συμβατές με τον autograder.

Όπως και πριν, μπορείτε να αξιολογήσετε και να διορθώσετε την υλοποίησή σας καλώντας

```
python autograder.py -q q7
```

Ερώτημα 8 (3 μονάδες): Expectimax

Στην περίπτωση του Expectimax δεν αποφασίζουμε με βάση τη χειρότερη περίπτωση (όπως κάνει ο Minimax), αλλά με βάση την αναμενόμενη αξία της κάθε ενέργειας. Υλοποιήστε τον `ExpectimaxAgent` ως παραλλαγή του Minimax με βάση τον παρακάτω ψευδοκώδικα. Θεωρήστε ότι οι ενέργειες του κάθε φαντάσματος ακολουθούν ομοιόμορφη κατανομή. Με άλλα λόγια η κάθε ενέργεια που επιστρέφεται από την `GameState.getPossibleActions` για ένα φάντασμα έχει την ίδια πιθανότητα να επιλεγεί.



Μπορείτε να ελέγξετε και να διορθώσετε τον κώδικά σας μέσω του autograder

```
python autograder.py -q q8
```

Ο Expectimax μοντελοποιεί καλύτερα την πραγματικότητα στο συγκεκριμένο πρόβλημα όταν τα φαντάσματα δεν είναι βέλτιστοι πράκτορες, αλλά κινούνται με τυχαίο τρόπο. Μπορείτε να δείτε πως τα πάει ο πράκτοράς σας στο Pacman καλώντας

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

Όταν ο Pacman βρεθεί παγιδευμένος (οπότε και θα έχανε με βέλτιστους αντιπάλους) πλέον προσπαθεί να βρει περισσότερο φαγητό ή να ξεφύγει, εκμεταλλευόμενος την τυχαιότητα των αντιπάλων. Αυτό δε συμβαίνει στην περίπτωση του Minimax που υποθέτει πάντα τη χειρότερη περίπτωση. Δείτε τα παρακάτω παραδείγματα.

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

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

Ο `AlphaBetaAgent` χάνει πάντα, ενώ ο `ExpectimaxAgent` κερδίζει περίπου τις μισές φορές. Σιγουρευτείτε ότι καταλαβαίνετε γιατί συμβαίνει αυτό.

Ερώτημα 9 (6 μονάδες): Συνάρτηση αξιολόγησης

Υλοποιήστε μία βελτιωμένη συνάρτηση αξιολόγησης `betterEvaluationFunction`. Η συνάρτηση θα πρέπει να αξιολογεί καταστάσεις (και όχι ενέργειες). Σε αναζήτηση με βάθος 2, η συνάρτησή σας θα πρέπει να κερδίσει τον λαβύρινθο `smallClassic` με ένα φάντασμα περισσότερες από τις μισές φορές.

Η βαθμολογία εξαρτάται από την επίδοση του πράκτορα σε 10 παιχνίδια:

- Αν ο πράκτορας κερδίσει έστω και μία φορά, παίρνετε μία μονάδα
- Αν κερδίσει 5 φορές +1 μονάδα, και +2 μονάδες αν κερδίσει και τις 10 φορές
- +1 για μέσο σκορ τουλάχιστον 500, +2 για μέσο σκορ τουλάχιστον 1000
- +1 αν τα παιχνίδια παίρνουν λιγότερο από 30 δευτερόλεπτα (στο δικό μου desktop υπολογιστή) με την επιλογή `--no-graphics`.

Μπορείτε να ελέγξετε τα παραπάνω με:

```
python autograder.py -q q9
```

ή, χωρίς γραφικά με

```
python autograder.py -q q9 --no-graphics
```

Ερώτηση 8 (για αναφορά): 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;

Καλή επιτυχία!