Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο
ΥΠ23 Τεχνητή Νοημοσύνη

Εργασία 2: Αναζήτηση με αντιπάλους

Έκδοση 2021-1.0

Διδάσκων: Χρήστος Δίου

1 Εισαγωγή

Σ' αυτή την εργασία θα γράψετε έναν πράκτορα ο οποίος θα μπορεί να παίζει το κλασικό παιχνίδι Pacman, συμπεριλαμβανομένων φαντασμάτων, κουκίδων κλπ. Στην πράξη θα εφαρμόσουμε τους αλγόριθμους Minimax, Alpha-Beta pruning και Expectimax που συζητήσαμε στο μάθημα.

Σας δίνεται ένα αρχείο `multiagent.tar.gz`. Από τα αρχεία που προκύπτουν μετά την αποσυμπίεση, θα χρειαστεί να επεξεργαστείτε μόνο το αρχείο `multiAgents.py`. Όπως και στην προηγούμενη εργασία, πιθανότατα θα χρειαστεί να εξετάσετε τα αρχεία `pacman.py`, `game.py` και `util.py`, αλλά όχι να τα αλλάξετε. Μπορείτε να ελέγχετε τις λύσεις σας μέσω του autograder.

1.1 Παραδοτέα και αξιολόγηση

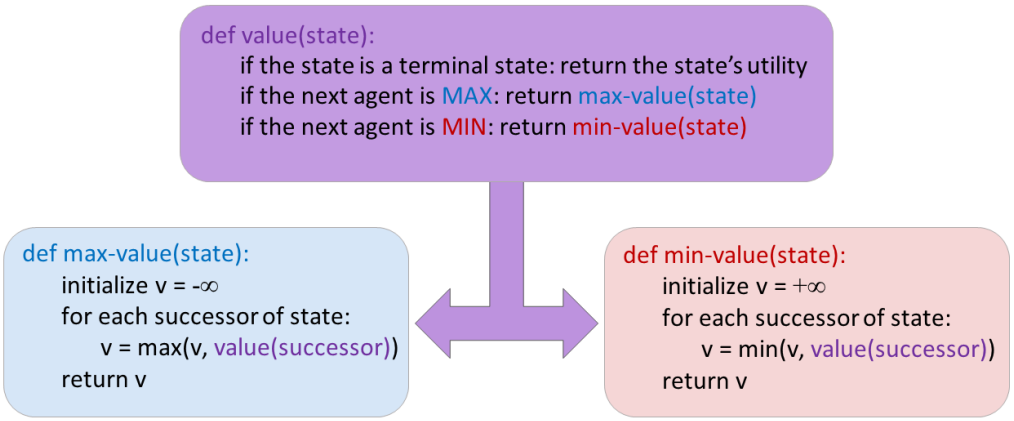
Η υποβολή της εργασίας σας περιλαμβάνει μόνο το αρχείο `multiagent.py`. Όταν το αρχείο που υποβάλετε αντικαταστήσει το `multiAgents.py` που περιέχει αρχικά το `multiagent.tar.gz`, θα πρέπει να επιτρέπει την εκτέλεση του autograder. Η ορθή λειτουργία των αρχείων σας θα γίνει αυτοματοποιημένα, επομένως φροντίστε ο κώδικάς σας να εκτελείται σωστά.

Ο κώδικάς σας επίσης θα ελεγχθεί για την ορθότητά του 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 Ασκήσεις

Άσκηση 1 (5 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 `self.depth` και `self.evaluationFunction`. Όταν δέχεστε περιορισμό στο βάθος, θα πρέπει ο αλγόριθμος να επιστρέφει την αξία που δίνεται από την `self.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 q1
```

Αν θέλετε να τρέξει πιο γρήγορα, χωρίς γραφικά, καλέστε

```
python autograder.py -q q1 --no-graphics
```

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 `self.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`.
- Όλες οι καταστάσεις πρέπει να είναι τύπου `GameState` (για παράδειγμα όπως επιστρέφονται από την `GameState.generateSuccessor`). Σε αντίθεση με τους αλγόριθμους αναζήτησης, εδώ δε χρησιμοποιούμε απλοποιημένους χώρους καταστάσεων.
- Μην καλείτε την `GameState.generateSuccessor` παραπάνω φορές απ' ότι χρειάζεται, γιατί αλλιώς η λύση σας δε θα είναι συμβατή με τον autograder.

Όταν ο Pacman βλέπει ότι θα χάσει, προσπαθεί να χάσει το συντομότερο δυνατό, ώστε να μεγιστοποιήσει το σκορ. Σε ορισμένες περιπτώσεις αυτή είναι η λάθος στρατηγική, όπως για παράδειγμα όταν τα φαντάσματα συμπεριφέρονται με τυχαίο τρόπο.

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

Σιγουρευτείτε ότι καταλαβαίνετε γιατί συμβαίνει αυτό.

Έχοντας ολοκληρώσει επιτυχώς αυτή την άσκηση, οι λύσεις στις επόμενες ασκήσεις είναι πιο απλές.

Άσκηση 2 (5 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.getLegalActions`, ώστε η 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 q2
```

Άσκηση 3 (5 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.getLegalActions` για ένα φάντασμα έχει την ίδια πιθανότητα να επιλεγεί.

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

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

```
python autograder.py -q q3
```

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.

Άσκηση 4 (6 μονάδες): Συνάρτηση αξιολόγηση

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`.
- Μην αντιγράψετε αρχεία από την Εργασία 1 (Αναζήτηση) γιατί η λύση σας δε θα περνάει από τον autograder.

Μπορείτε να ελέγξετε τα παραπάνω με:

```
python autograder.py -q q4
```

ή, χωρίς γραφικά με

```
python autograder.py -q q5 --no-graphics
```

Καλή επιτυχία!