Ειδικά θέματα: Αποκεντρωμένα Συστήματα και Αναδρομική Λήψη Αποφάσεων
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Reinforcement Learning with Pacman

Βαμβουρέλλης Ευστράτιος
1115201600014

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

2019-2020

Contents

1. MDPs

Η εργασία αυτή σχετίζεται με τα MDPs και τους τρόπους επίλυσης τους. Αρχικά θα ορίσουμε τι είναι ένα MDP ή Markov Decision Process. Markov Decision Process είναι μια μερικώς τυχαία διαδικασία αποφάσεων διακριτού χρόνου. Είναι ένα μαθηματικό περιβάλλον για να μοντελοποιούμε διαδικασίες λήψεις αποφάσεων όπου τα αποτελέσματα εξαρτώνται μερικώς από εμάς και μερικώς από την τύχη. Το βασικό τους χαρακτηριστικό είναι οτι το αποτέλεσμα κάποιας πράξης εξαρτάται μόνο από την τωρινή κατάσταση. Μαθηματικά ορίζονται έτσι:

- Ένα σύνολο από καταστάσεις $s \in S$.
- Ένα σύνολο από ενέργειες $a \in A$.
- Μια συνάρτηση μετάβασης $T(s, a, s')$. Δηλαδή η πιθανότητα η ενέργεια a να οδηγήσει από την κατάσταση s στην s', $(P(s'|s, a))$
- Μια συνάρτηση κόστους-κέρδους $R(s, a, s')$ η οποία μπορεί να περιλαμβάνει και ένα πέναλτι γ. Μερικές φορές την συμβολίζουμε ως $R(s)$ ή $R(s')$.
- Μια αρχική κατάσταση.
- Τουλάχιστον μια τελική κατάσταση.

Ως πολιτική ορίζουμε ένα σύνολο από ζευγάρια καταστάσεων-ενεργειών $\pi : S \to A$. Ως βέλτιστη πολιτική π^* ορίζουμε μια πολιτική την οποία όταν ακολουθήσουμε θα μας δώσει το μέγιστο δυνατό ολικό κέρδος (άθροισμα των επιμέρους κερδών).
Επιπλέον ορίζουμε κάποιες ακόμα συναρτήσεις:

- $V^*(s)$ είναι το μέγιστο αναμενόμενο ολικό κέρδος, αν αρχίσουμε από την κατάσταση s και στη συνέχεια ακολουθήσουμε μια βέλτιστη πολιτική. $V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$
- $Q^*(s, a)$ είναι το μέγιστο αναμενόμενο ολικό κέρδος, αν αρχίσουμε από την κατάσταση s, επιλέξουμε την ενέργεια a και στη συνέχεια ακολουθήσουμε μια βέλτιστη πολιτική. Αυτή η συνάρτηση είναι γνωστή και ως Q-value. $Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$

Παρατηρούμε οτι $V^*(s) = \max_a Q^*(s, a)$.

2. Εισαγωγή στο περιβάλλων της άσκησης

2.1 Autograder

Ο autograder είναι ένας εύκολος τρόπος να βαθμολογηθεί αυτή ολόκληρη η άσκηση. Για να τον εκτελέσουμε χρησιμοποιειούμε την εντολή python autograder.py. Μετά από λίγα δευτερόλεπτα θα εμφανίσει αποτελέσματα παρόμοια με τα παρακάτω:

```
/home/stratos/Desktop/reinforcement/venv/bin/python
/home/stratos/Desktop/reinforcement/autograder.py
Starting on 3-18 at 22:38:40

Question q1
===========

*** PASS: test_cases/q1/1-tinygrid.test
*** PASS: test_cases/q1/2-tinygrid-noisy.test
*** PASS: test_cases/q1/3-bridge.test
*** PASS: test_cases/q1/4-discountgrid.test

### Question q1: 6/6 ###


Question q2
===========

*** PASS: test_cases/q2/1-bridge-grid.test

### Question q2: 1/1 ###


Question q3
===========

*** PASS: test_cases/q3/1-question-3.1.test
*** PASS: test_cases/q3/2-question-3.2.test
*** PASS: test_cases/q3/3-question-3.3.test
*** PASS: test_cases/q3/4-question-3.4.test
*** PASS: test_cases/q3/5-question-3.5.test

### Question q3: 5/5 ###


Question q4
===========

*** PASS: test_cases/q4/1-tinygrid.test
*** PASS: test_cases/q4/2-tinygrid-noisy.test
*** PASS: test_cases/q4/3-bridge.test
*** PASS: test_cases/q4/4-discountgrid.test

### Question q4: 5/5 ###


Question q5
===========

*** PASS: test_cases/q5/1-tinygrid.test
*** PASS: test_cases/q5/2-tinygrid-noisy.test
*** PASS: test_cases/q5/3-bridge.test
```

```
*** PASS: test_cases/q5/4-discountgrid.test

### Question q5: 3/3 ###


Question q6
===========

*** PASS: test_cases/q6/grade-agent.test

### Question q6: 1/1 ###


Question q7
===========

Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -511.48
        Average Rewards for last 100 episodes: -511.48
        Episode took 0.61 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -511.97
        Average Rewards for last 100 episodes: -512.46
        Episode took 0.78 seconds
Reinforcement Learning Status:
        Completed 300 out of 2000 training episodes
        Average Rewards over all training: -498.81
        Average Rewards for last 100 episodes: -472.50
        Episode took 0.86 seconds
Reinforcement Learning Status:
        Completed 400 out of 2000 training episodes
        Average Rewards over all training: -456.87
        Average Rewards for last 100 episodes: -331.03
        Episode took 1.02 seconds
Reinforcement Learning Status:
        Completed 500 out of 2000 training episodes
        Average Rewards over all training: -425.56
        Average Rewards for last 100 episodes: -300.31
        Episode took 0.99 seconds
Reinforcement Learning Status:
        Completed 600 out of 2000 training episodes
        Average Rewards over all training: -411.23
        Average Rewards for last 100 episodes: -339.62
        Episode took 1.01 seconds
Reinforcement Learning Status:
        Completed 700 out of 2000 training episodes
        Average Rewards over all training: -387.85
        Average Rewards for last 100 episodes: -247.55
        Episode took 0.96 seconds
```

```
Reinforcement Learning Status:
        Completed 800 out of 2000 training episodes
        Average Rewards over all training: -366.87
        Average Rewards for last 100 episodes: -220.03
        Episode took 1.08 seconds
Reinforcement Learning Status:
        Completed 900 out of 2000 training episodes
        Average Rewards over all training: -342.65
        Average Rewards for last 100 episodes: -148.85
        Episode took 1.08 seconds
Reinforcement Learning Status:
        Completed 1000 out of 2000 training episodes
        Average Rewards over all training: -311.05
        Average Rewards for last 100 episodes: -26.66
        Episode took 1.04 seconds
Reinforcement Learning Status:
        Completed 1100 out of 2000 training episodes
        Average Rewards over all training: -273.34
        Average Rewards for last 100 episodes: 103.80
        Episode took 1.16 seconds
Reinforcement Learning Status:
        Completed 1200 out of 2000 training episodes
        Average Rewards over all training: -236.68
        Average Rewards for last 100 episodes: 166.55
        Episode took 1.01 seconds
Reinforcement Learning Status:
        Completed 1300 out of 2000 training episodes
        Average Rewards over all training: -203.36
        Average Rewards for last 100 episodes: 196.52
        Episode took 1.03 seconds
Reinforcement Learning Status:
        Completed 1400 out of 2000 training episodes
        Average Rewards over all training: -170.41
        Average Rewards for last 100 episodes: 257.86
        Episode took 0.99 seconds
Reinforcement Learning Status:
        Completed 1500 out of 2000 training episodes
        Average Rewards over all training: -142.58
        Average Rewards for last 100 episodes: 247.09
        Episode took 1.07 seconds
Reinforcement Learning Status:
        Completed 1600 out of 2000 training episodes
        Average Rewards over all training: -120.08
        Average Rewards for last 100 episodes: 217.33
        Episode took 1.03 seconds
Reinforcement Learning Status:
        Completed 1700 out of 2000 training episodes
        Average Rewards over all training: -97.83
        Average Rewards for last 100 episodes: 258.15
        Episode took 0.99 seconds
Reinforcement Learning Status:
        Completed 1800 out of 2000 training episodes
```

```
        Average Rewards over all training: -83.74
        Average Rewards for last 100 episodes: 155.91
        Episode took 1.05 seconds
Reinforcement Learning Status:
        Completed 1900 out of 2000 training episodes
        Average Rewards over all training: -69.54
        Average Rewards for last 100 episodes: 186.07
        Episode took 1.09 seconds
Reinforcement Learning Status:
        Completed 2000 out of 2000 training episodes
        Average Rewards over all training: -54.19
        Average Rewards for last 100 episodes: 237.40
        Episode took 1.02 seconds
Training Done (turning off epsilon and alpha)
----------------------------------------------
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
```

```
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
```

```
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Reinforcement Learning Status:
        Completed 100 test episodes
        Average Rewards over testing: 499.88
        Average Rewards for last 100 episodes: 499.88
        Episode took 1.09 seconds
Average Score: 499.88
Scores:       503.0, 499.0, 495.0, 495.0, 503.0, 503.0, 495.0, 503.0, 503.0, 499.0, 499.0, 495.0,
499.0, 499.0, 503.0, 499.0, 499.0, 503.0, 499.0, 499.0, 503.0, 495.0, 503.0, 499.0, 503.0, 503.0,
499.0, 495.0, 503.0, 503.0, 495.0, 503.0, 503.0, 495.0, 499.0, 499.0, 503.0, 503.0, 503.0, 495.0,
503.0, 499.0, 503.0, 499.0, 499.0, 503.0, 503.0, 503.0, 503.0, 499.0, 503.0, 495.0, 499.0, 503.0,
499.0, 503.0, 503.0, 503.0, 499.0, 503.0, 499.0, 503.0, 503.0, 499.0, 503.0, 495.0, 503.0, 503.0,
499.0, 503.0, 495.0, 495.0, 499.0, 495.0, 499.0, 495.0, 495.0, 503.0, 495.0, 499.0, 503.0, 499.0,
495.0, 503.0, 495.0, 495.0, 499.0, 503.0, 503.0, 503.0, 495.0, 503.0, 503.0, 495.0, 495.0, 503.0,
495.0, 503.0, 495.0, 503.0
Win Rate:     100/100 (1.00)
Record:       Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q7/grade-agent.test (1 of 1 points)
***     Grading agent using command:  python pacman.py -p PacmanQAgent -x 2000 -n 2100 -l
smallGrid -q -f --fixRandomSeed
***     100 wins (1 of 1 points)
***         Grading scheme:
***          < 70:  0 points
***         >= 70:  1 points

### Question q7: 1/1 ###


Question q8
==========

*** PASS: test_cases/q8/1-tinygrid.test
*** PASS: test_cases/q8/2-tinygrid-noisy.test
*** PASS: test_cases/q8/3-bridge.test
*** PASS: test_cases/q8/4-discountgrid.test
*** PASS: test_cases/q8/5-coord-extractor.test
```

```
### Question q8: 3/3 ###


Finished at 22:39:03

Provisional grades
==================
Question q1: 6/6
Question q2: 1/1
Question q3: 5/5
Question q4: 5/5
Question q5: 3/3
Question q6: 1/1
Question q7: 1/1
Question q8: 3/3
------------------
Total: 25/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.


Process finished with exit code 0
```

Μπορεί να εκτελεστεί και με το όρισμα -q ακολουθούμενο από τον αριθμό του ερωτήματος. πχ python autograder.py -q q2. Αυτό θα εκτελέσει και θα βαθμολογήσει μόνο το δεύτερο ερώτημα.

2.2 Gridworld

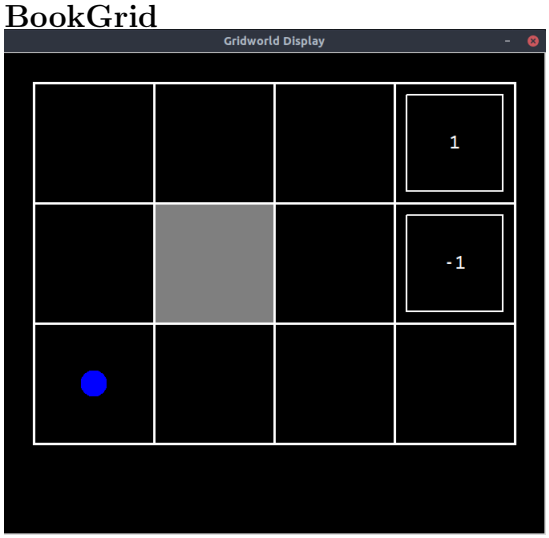
Το Gridworld είναι ένας κόσμος που αναπαριστά ένα απλό MDP. Για να τον εκτελέσουμε με γραφικό περιβάλλων στην χειροκίνητη λειτουργία του χρησιμοποιούμε το python gridworld.py -m. Χρησιμοποιούμε τα βελάκια για να μετακινηθούμε. Υπάρχουν πολλές παράμετροι, για να τις δούμε όλες εκτελούμε την εντολή python gridworld.py -h
Μπορούμε να εκτελέσουμε τον gridworld με τον πράκτορά μας να είναι σε λειτουργία τυχαίας απόφασης με την παρακάτω εντολή python gridworld.py -g MazeGrid.

Οι κανόνες του κόσμου αυτού είναι απλοί. Από την στιγμή που ο πράκτορας μας επιλέξει να κινηθεί προς μια κατεύθυνση (να κάνει μια ενέργεια), αυτή η ενέργεια έχει πιθανότητα 80% να συμβεί. Αν δεν συμβεί, υπάρχει μια πιθανότητα 10% να κινηθεί προς μια κατεύθυνση κάθετη προς την αρχική του επιλογή και 10% να κινηθεί προς την άλλη κάθετη κατεύθυνση. πχ αν ο πράκτορας μας επιλέξει να πάει πάνω τότε 80% θα πάει πάνω, 10% δεξιά, 10% αριστερά.

Ο πράκτορας συμβολίζεται με την μπλε τελίτσα. Η τελικές καταστάσεις με τετραγωνάκια (μέσα στα τετραγωνάκια 4,1 4,2). Το γκρι (2,2) είναι τοίχος, δηλαδή μία κατάσταση που δεν μπορούμε να φτάσουμε (μπορεί και να μην υπάρχει στο σύνολο των καταστάσεων αν μοντελοποιήσουμε κατάλληλα το σύνολο ενεργειών).

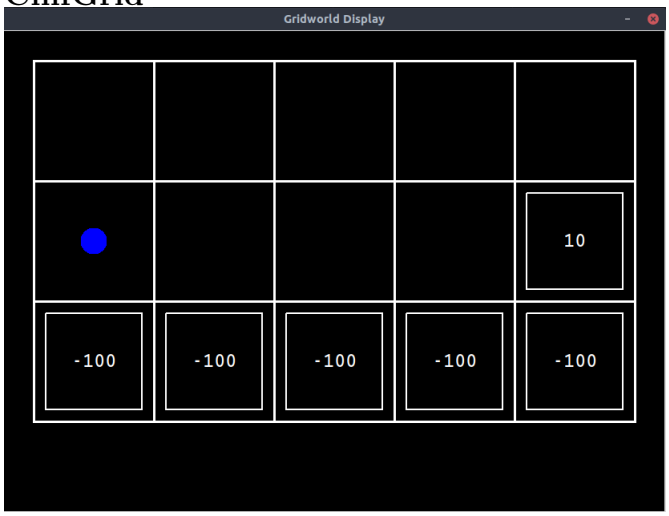
Ο Gridworld έχει πολλές παραδοχές τις οποίες μπορούμε να επιλέξουμε με την επιλογή -g.

BookGrid

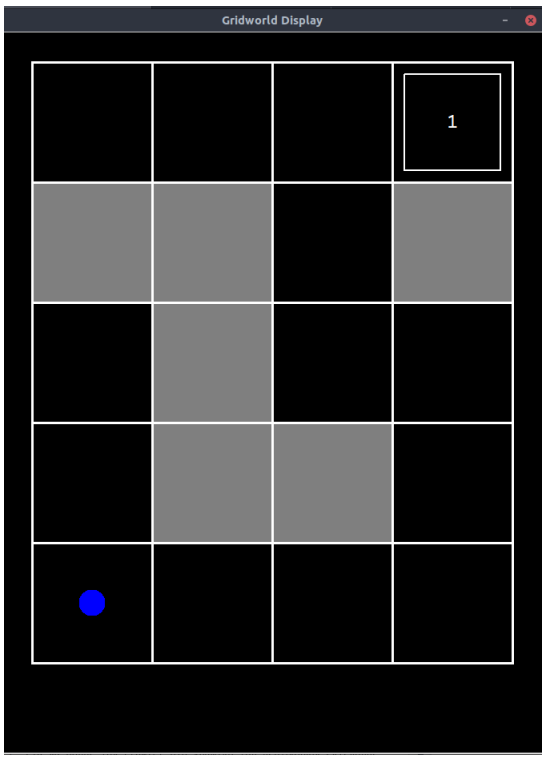


BridgeGrid

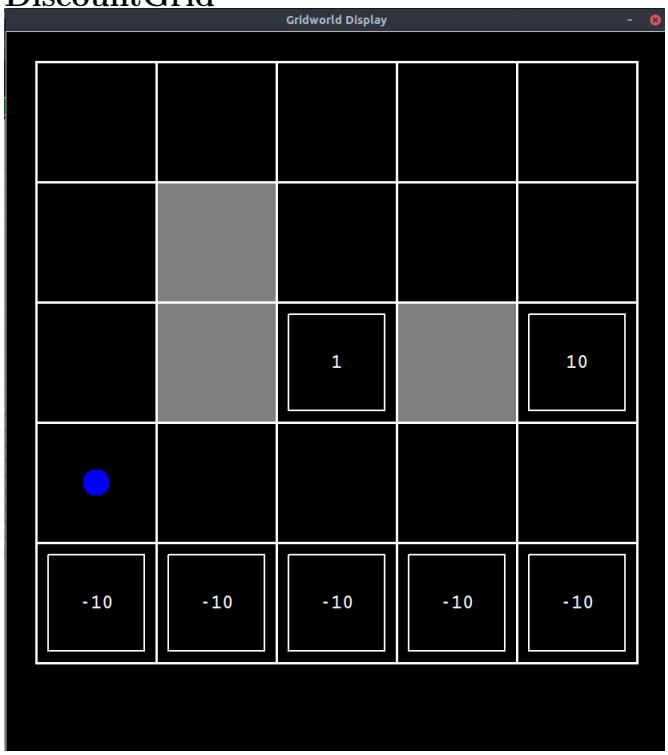
CliffGrid



MazeGrid

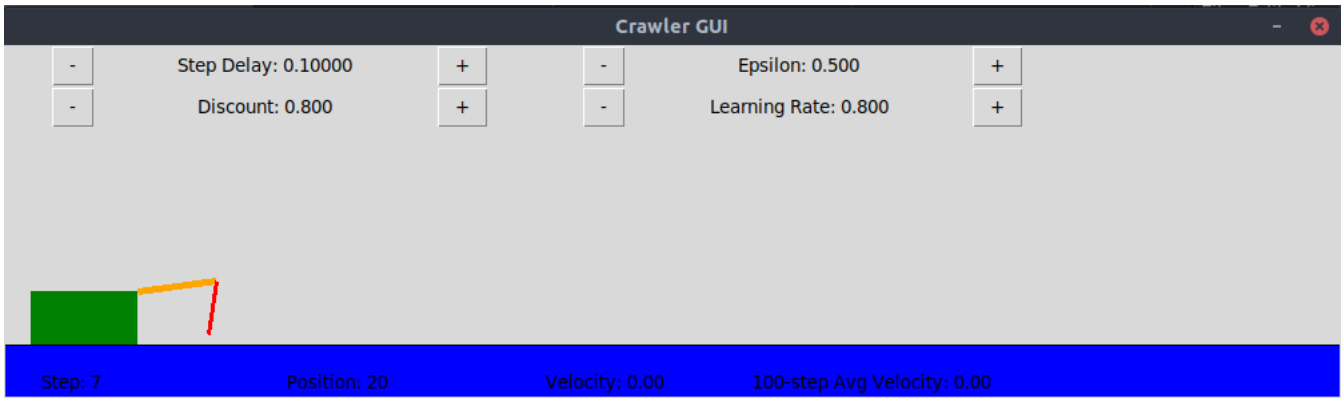


DiscountGrid



2.3 Crawler

Στον κόσμο του crawler υπάρχει ένας πράκτορας ο οποίος προσπαθεί να κινηθεί προς τα δεξιά χρησιμοποιώντας 2 μέλη που συνδέονται στο σώμα του και κινούνται ανεξάρτητα. Ο σκοπός του είναι να κινείται όσο πιο γρήγορα μπορεί προς τα δεξιά.
Κάτω στο παράθυρο εμφανίζεται ο αριθμός επαναλήψεων, η θέση, η ταχύτητα και η μέση ταχύτητα ανά 100 επαναλήψεις. Πάνω εμφανίζονται κουμπιά που ελέγχουν την ταχύτητα των frame, την τιμή του epsilon ϵ, την τιμή του discount γ και την τιμή του learning rate α.
Για να δούμε τον crawler στο γραφικό του περιβάλλον εκτελούμε python crawler.py.

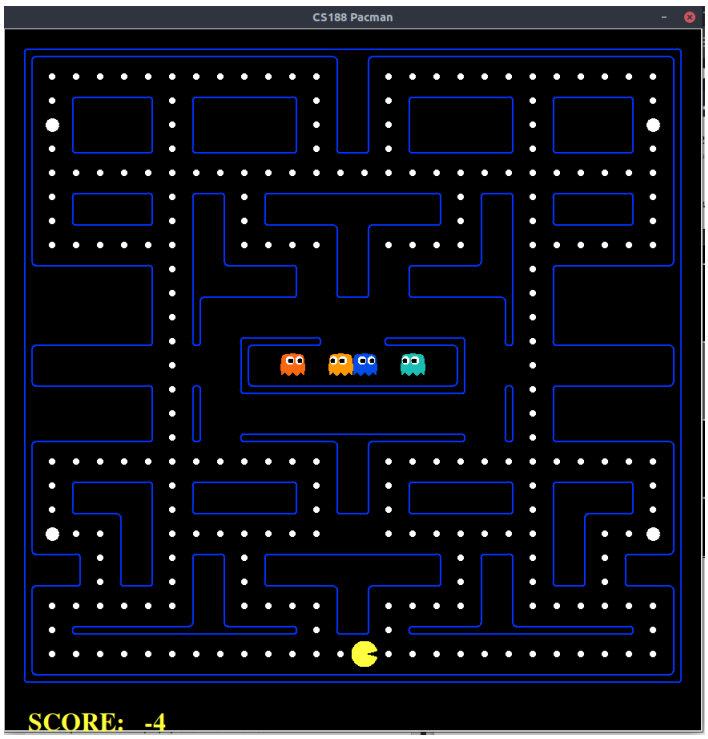


2.4 Pacman grids

Εδώ βλέπουμε τον κλασικό κόσμο του pacman. Για να τον εκτελέσουμε με γραφικό περιβάλλων στην χειροκίνητη λειτουργία του χρησιμοποιούμε το python pacman.py. Χρησιμοποιούμε τα βελάκια για να μετακινηθούμε. Υπάρχουν πολλές παράμετροι, για να τις δούμε όλες εκτελούμε την εντολή python pacman.py -h

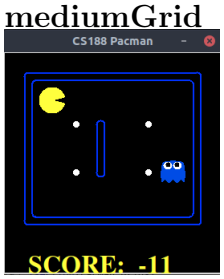
Ο κόσμος αυτός έχει πολλές εκδοχές-χάρτες παρακάτω είναι μια λίστα με τους πιο βασικούς από αυτούς: originalClassic, mediumClassic, mediumGrid, smallClassic, smallGrid, trickyClassic. Μπορούμε να βάλουμε όποιον πράκτορα θέλουμε σε όποιο χάρτη θέλουμε με την επιλογή -l. Στα παρακάτω ερωτήματα δεν χρησιμοποιούνται όλοι οι χάρτες αλλά είναι πολύ ενδιαφέρον να δούμε τον πράκτορά μας να παίζει και σε άλλους χάρτες, κυρίως τον originalClassic.

originalClassic

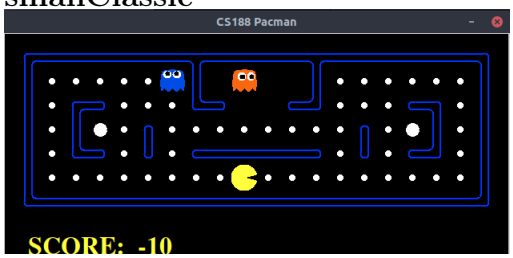


mediumClassic

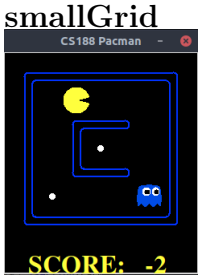
mediumGrid



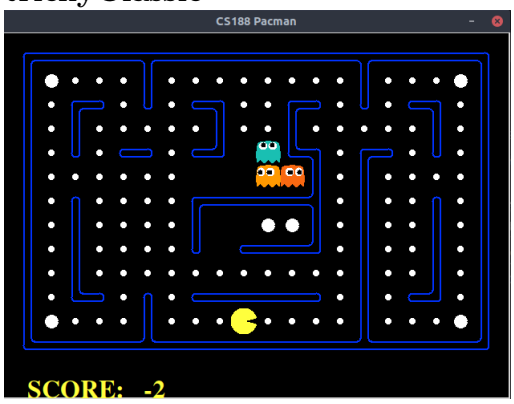
smallClassic



smallGrid



trickyClassic



3. Question 1: Value Iteration

Το πρώτο ερώτημα μας ζητάει να εφαρμόσουμε την πιο απλή τακτική για να λύσουμε το grid-world, ονομαζόμενη Value Iteration.
Αρχικά έχουμε από την εξίσωση του Bellman τον ορισμό του μέγιστου αναμενόμενου ολικού κέρδους: $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
Ο αλγόριθμός Value Iteration ξεκινάει απο μια τιμή $V_0(s)$ (στην συγκεκριμένη περίπτωση $V_0(s) = 0$) και υπολογίζει το $V_{k+1}(s)$ με τον παρακάτω τρόπο:
$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$
Εμείς, όμως, ψάχνουμε το $V^*(s)$. Παρακάτω θα αποδείξω οτι το $V_{k+1}(s)$, μετά από αρκετές επαναλήψεις συγκλίνει στο V^*.
Ορίζουμε τον Bellman optimality operator $B : \Re^N \to \Re^N$ για κάθε συνάρτηση $f \in \Re^N$:
$(Bf)(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma f(s')]$
Παρατηρούμε οτι $(BV^*)(s) = v^*(s)$ και $V_{k+1}(s) = (BV_k)(s)$.
Θέλω να αποδείξω οτι:
$\lim_{k\to\infty} V_k = V^* \quad \Leftrightarrow$
$\| V_{k+1} - V^* \|_\infty = 0 \quad \Leftrightarrow$
$\| BV_k - BV^* \|_\infty = 0 \quad \Leftrightarrow$
$\| BV_k - BV^* \|_\infty \leq \gamma \| V_k - V^* \|_\infty \quad \leq ... \leq \gamma^{k+1} \| V_k - V^* \|_\infty \to 0$

Άρα μετά από αρκετές επαναλήψεις το V_{k+1} θα συγκλίνει στο V^*. Ο αλγόριθμος τερματίζει

όταν $\| V_k - V_{k+1} \| \leq \epsilon$ όπου ϵ ένας "μικρός" αριθμός που ορίζουμε εμείς (στο συγκεκριμένο παράδειγμα είναι αρκετά μικρός ώστε να έχουμε ακρίβεια 2 δεκαδικών δηλαδή $\epsilon \approx 0.01$).

Στο συγκεκριμένο ερώτημα υλοποιώ τις συναρτήσεις __init__, computeQValueFromValues, computeActionFromValues για τον αλγόριθμο Value Iteration, στο αρχείο valueIterationAgents.py:

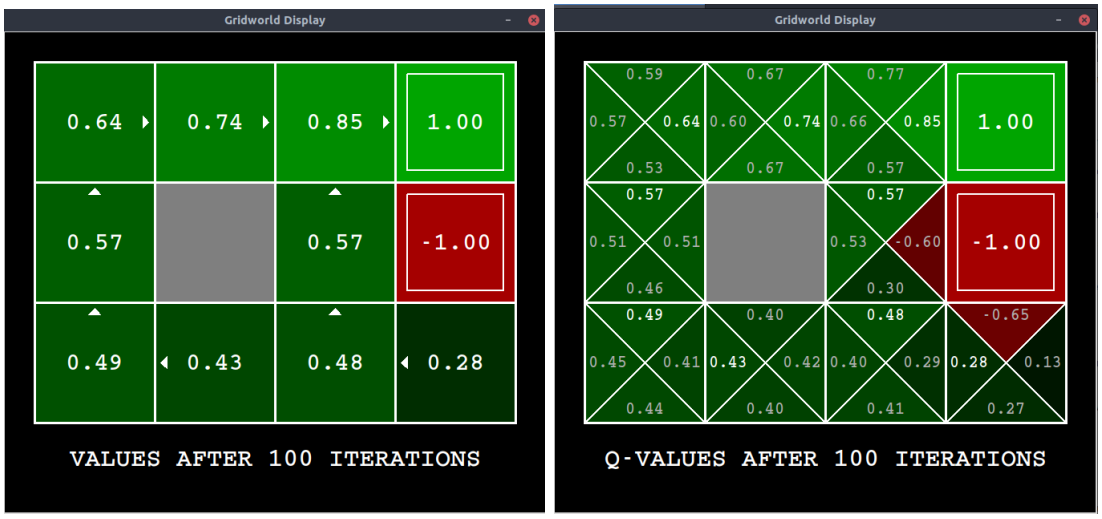
```
class ValueIterationAgent(ValueEstimationAgent):
    """
        * Please read learningAgents.py before reading this.*

        A ValueIterationAgent takes a Markov decision process
        (see mdp.py) on initialization and runs value iteration
        for a given number of iterations using the supplied
        discount factor.
    """
    def __init__(self, mdp, discount = 0.9, iterations = 100):
        """
          Your value iteration agent should take an mdp on
          construction, run the indicated number of iterations
          and then act according to the resulting policy.

          Some useful mdp methods you will use:
              mdp.getStates()
              mdp.getPossibleActions(state)
              mdp.getTransitionStatesAndProbs(state, action)
              mdp.getReward(state, action, nextState)
              mdp.isTerminal(state)
        """
        self.mdp = mdp
        self.discount = discount
        self.iterations = iterations
        self.values = util.Counter()

        "*** YOUR CODE HERE ***"
        for i in range(iterations):#iterations = k
            newValues = self.values.copy()  # copy the values
            for state in self.mdp.getStates():  # go through every state
                if not self.mdp.isTerminal(state):
                    actions = self.getAction(state) # get best action
                    bestValue = self.getQValue(state,actions) # get value of best action
                    newValues[state] = bestValue # store best value,state pair
            self.values = newValues
```

```
def getValue(self, state):
    """
      Return the value of the state (computed in __init__).
    """
    return self.values[state]


def computeQValueFromValues(self, state, action):
    """
      Compute the Q-value of action in state from the
      value function stored in self.values.
    """
    "*** YOUR CODE HERE ***"
    qValue = 0

    for nextState, probability in self.mdp.getTransitionStatesAndProbs(state, action)
        # compute qvalue from equation
        reward = self.mdp.getReward(state, action, nextState)
        qValue += probability * (reward + self.discount * self.values[nextState])

    return qValue

def computeActionFromValues(self, state):
    """
      The policy is the best action in the given state
      according to the values currently stored in self.values.

      You may break ties any way you see fit.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return None.
    """
    "*** YOUR CODE HERE ***"
    policies = util.Counter() # basicaly a dictionary
    for action in self.mdp.getPossibleActions(state): # for every possible action
        policies[action] = self.getQValue(state, action) # store the q-value

    # return the best action (the one with the greater q-value)
    return policies.argMax()

def getPolicy(self, state):
    return self.computeActionFromValues(state)
```

```
    def getAction(self, state):
        "Returns the policy at the state (no exploration)."
        return self.computeActionFromValues(state)

    def getQValue(self, state, action):
        return self.computeQValueFromValues(state, action)
```

Για να εκτελέσουμε αυτό το ερώτημα και το γραφικό του περιβάλλον γράφουμε την εντολή python gridworld.py -a value -i 100 -k 10. Στην αρχή θα εμφανιστούν οι τιμές της $V(s)$. Αν πατήσουμε ένα κουμπί θα εμφανιστούν οι τιμές της $Q(s)$. Αν ξαναπατήσουμε ένα κουμπί θα δούμε τον πράκτορα να αλληλεπιδρά με το περιβάλλον, 10 φορές, ακολουθώντας την βέλτιστη πολιτική π^* την οποία έμαθε, μετά από 100 επαναλήψεις του Value Iteration. Παρατηρούμε ένα βελάκι ανα κατάσταση (τετραγωνάκι του gridworld) αυτό το βελάκι δείχνει την βέλτιστη ενέργεια από αυτή την κατάσταση. Δηλαδή, όπως προείπαμε, όλα τα ζευγάρια βελάκια-τετραγωνάκια είναι η βέλτιστη πολιτική π^*.

Μετά από την εκτέλεση εμφανίζεται το παρακάτω:



4. Question 2: Bridge Crossing Analysis

Σε αυτό το ερώτημα ο πράκτορας ξεκινάει πιο κοντά στην τερματική κατάσταση με το "μικρό" κέρδος 1. Θέλουμε να αλλάξουμε μια απο τις παραμέτρους discount ή noise ώστε να "αναγκάσουμε" τον πράκτορα να διασχίσει την "γέφυρα" και να καταλήξει στην τερματική κατάσταση με το "μεγάλο" κέρδος 10.

Μετά από πειραματισμό παρατήρησα οτι το πρόβλημα είναι ο θόρυβος. Άρχιζα να τον μειώνω στα μισά μέχρι ο πράκτορας να διασχίσει την γέφυρα. Τελικά κατάλαβα οτι το πιο απλό θα ήταν να βάλω τον θόρυβο ίσο με το 0.

Ο κώδικας που έγραψα είναι η συνάρτηση question2 στο analysys.py.

```
def question2():
    answerDiscount = 0.9
    answerNoise = 0
    return answerDiscount, answerNoise
```

Μετά από την εκτέλεση εμφανίζεται το παρακάτω:

5. Question 3: Policies

Αυτό το ερώτημα χωρίζεται σε 5 υποερωτήματα. Σε όλα ο πράκτοράς μας ζει στο περιβάλλον DiscountGrid και το καθένα μας ζητάει να αλλάξουμε τις παραμέτρους discount, noise, living reward με τέτοιο τρόπο ώστε να πετύχουμε συγκεκριμένα αποτελέσματα. Όλα τα υποερωτήματα τα έλυσα διαισθητικά και με λίγο πειραματισμό.

Για να εκτελέσουμε αυτό το ερώτημα και το γραφικό του περιβάλλον γράφουμε την εντολή python gridworld.py -a value -g DiscountGrid –discount x –noise y –livingReward z. Εμφανίζεται ο κόσμος DiscountGrid και βλέπουμε την "βέλτιστη" πολιτική και τις τιμές της για τις συγκεκριμένες παραμέτρους, όπου x,y,z οι επιθυμητές τιμές για κάθε παράμετρο.

5.1 Prefer the close exit (+1), risking the cliff (-10)

Σε αυτό το υποερωτήμα για ευκολία έβαλα το living reward =0 και σκέφτηκα διαισθητικά να αλλάξω μόνο το discount που έχει παρόμοιο ρόλο. Το noise=0 το έβαλα ώστε ο πράκτορας να μην "φοβάται" να περάσει δίπλα απο τον γκρεμό. Τέλος μετά απο πειραματισμό βρήκα ότι το discount πρέπει να είναι κάτω από 0.3, ώστε ο πράκτορας να προτιμάει την κοντινή τερματική κατάσταση με το "μικρο" βραβείο.

Ο κώδικας που έγραψα είναι η συνάρτηση question3a στο analysys.py.

```
def question3a():
    answerDiscount = 0.2
    answerNoise = 0
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
```

5.2 Prefer the close exit (+1), but avoiding the cliff (-10)

Αντίστοιχα με το προηγούμενο υποερώτημα όπου "αναγκάσαμε" τον πράκτορα να καταλήξει στην κοντινή τερματική κατάσταση, σε αυτό το ερώτημα το μόνο που πρέπει να αλλάξουμε είναι το noise ώστε να τον "αποθαρρύνουμε" να περάσει δίπλα από τον "γκρεμό". Μετά απο πειραματισμό βρήκα οτι πρέπει αν είναι περίπου κάτω απο 0.24 και πάνω απο 0.01.

Ο κώδικας που έγραψα είναι η συνάρτηση question3b στο analysys.py.

```
def question3b():
    answerDiscount = 0.2
    answerNoise = 0.2
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
```

5.3 Prefer the distant exit (+10), risking the cliff (-10)

Σε αυτό το υποερωτήμα για ευκολία έβαλα το living reward =0 και σκέφτηκα διαισθητικά να αλλάξω μόνο το discount που έχει παρόμοιο ρόλο. Το noise=0 το έβαλα ώστε ο πράκτορας να

μην "φοβάται" να περάσει δίπλα από τον γκρεμό. Τέλος μετά από πειραματισμό βρήκα ότι το discount πρέπει να είναι μεγαλύτερο από 0.3 (αναμενόμενο από το προηγούμενο ερώτημα και προφανώς μικρότερο απο 1), ώστε ο πράκτορας να προτιμάει την μακρινή τερματική κατάσταση με το "μεγάλο" βραβείο.

Ο κώδικας που έγραψα είναι η συνάρτηση question3c στο analysys.py.

```
def question3c():
    answerDiscount = 0.9
    answerNoise = 0
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
```

5.4 Prefer the distant exit (+10), avoiding the cliff (-10)

Αντίστοιχα με το προηγούμενο υποερώτημα όπου "αναγκάσαμε" τον πράκτορα να καταλήξει στην μακρινή τερματική κατάσταση, σε αυτό το ερώτημα το μόνο που πρέπει να αλλάξουμε είναι το noise ώστε να τον "αποθαρρύνουμε" να περάσει δίπλα από τον "γκρεμό". Μετά από πειραματισμό βρήκα οτι πρέπει αν είναι τουλάχιστον περίπου 0.1 (ώστε να μην αξίζει να "ρισκάρει τον γκρεμό") και λιγότερο από 0.5 (ώστε να έχει τον "κύριο έλεγχο" για τις πράξεις του).

Ο κώδικας που έγραψα είναι η συνάρτηση question3d στο analysys.py.

```
def question3d():
    answerDiscount = 0.9
    answerNoise = 0.2
    answerLivingReward = 0
    return answerDiscount, answerNoise, answerLivingReward
```

5.5 Avoid both exits and the cliff (so an episode should never terminate)

Το μόνο που πρέπει αν κάνουμε σε αυτό το υποερώτημα είναι να δώσουμε στον πράκτορα θετικό living reward ώστε να προτιμάει να κινείται στον κόσμο παρά να τερματίσει το παιχνίδι (είτε φτάνοντας σε μια τερματική κατάσταση με θετικό βραβείο είτε σε μια με αρνητικό δηλαδή τον "γκρεμό"). Οπότε βάζω τις default values discount=0.9 noise=0.2 όπως πριν και αλλάζω το living reward σε τουλάχιστον 1 (ώστε να μην αξίζει να πάει ούτε στην κοντινότερη τερματική κατάσταση +1).

Ο κώδικας που έγραψα είναι η συνάρτηση question3e στο analysys.py.

```
def question3e():
    answerDiscount = 0.9
    answerNoise = 0.2
    answerLivingReward = 1
    return answerDiscount, answerNoise, answerLivingReward
```

5.6 Μπόνους παρατήρηση

Αν θέλαμε να αναγκάσουμε τον πράκτορα να πέσει από τον γκρεμό το μόνο που πρέπει να κάνουμε είναι αν του δώσουμε ένα πολύ μικρό (αρνητικό) living reward πχ -100. Τότε ο πράκτορας θέλει να τερματίσει το παιχνίδι όσο πιο γρήγορα γίνεται, δηλαδή προτιμάει να πέσει από τον "γκρεμό" γιατί είναι ο πιο γρήγορος τρόπος να τερματίσει το παιχνίδι.

6. Question 4: Q-Learning

Στο ερώτημα 1 κάναμε offline planning, δηλαδή ο πράκτορας μας βρήκε μια πολιτική χωρίς να αλληλεπιδράσει καθόλου με το περιβάλλον του. Όταν ο πράκτορας ξεκίνησε να αλληλεπιδρά με το περιβάλλον, ήξερε ήδη ποια πολιτική θα ακολουθήσει και απλά την ακολουθούσε. Ο πράκτορας, λοιπόν, ήταν reflex agent και οχι reinforcement learning agent. Αυτό δεν είναι πολύ πρακτικό για αληθινές εφαρμογές όμως γιατί χρειάζεται να ξέρουμε όλες τις παραμέτρους του MDP (model-based). Στον πραγματικό κόσμο όμως δεν γνωρίζουμε πάντα όλες τις παραμέτρους ενός προβλήματος, συνήθως δεν ξέρουμε ακριβώς τις συναρτήσεις $T(s, a, s'))$ και $R(s, a, s')$. Σε αυτό το ερώτημα θα αρχίσουμε να φτιάχνουμε έναν πράκτορα που θα μαθαίνει καθώς αλληλεπιδρά με το περιβάλλον του. Θα ξεκινήσει χωρίς να ξέρει τις συνάρτησής μετάβασης και κέρδους και θα προσπαθήσει να τις προσεγγίσει καθώς αλληλεπιδρά με το περιβάλλον του και παίρνει περισσότερες πληροφορίες για τον κόσμο που ζει (model-free).

Η μέθοδος λέγεται q-learning γιατί προσπαθεί να προσεγγίσει την συνάρτηση $Q^*(s, a)$. Αυτό το καταφέρνει ξεκινώντας με μια τιμή της συνάρτησης (στην προκειμένη περίπτωση 0) και βελτιώνοντας την κάθε φορά που αλληλεπιδρά με το περιβάλλον του. Σε αυτό το ερώτημα υλοποιώ την συνάρτηση update η οποία κωδικοποιεί την παρακάτω συνάρτηση:
$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha * sample \quad , sample = R(s, a, s') + \gamma \max_a Q(s', a')$

Παρακάτω είναι η συνάρτηση update, computeActionFromQValues, __init__, getQValue, computeValueFromQValues του αντικειμένου QLearningAgent, στο αρχείο qlearningAgents.py:

```
class QLearningAgent(ReinforcementAgent):
    """
```

```
  Q-Learning Agent

  Functions you should fill in:
    - computeValueFromQValues
    - computeActionFromQValues
    - getQValue
    - getAction
    - update

  Instance variables you have access to
    - self.epsilon (exploration prob)
    - self.alpha (learning rate)
    - self.discount (discount rate)

  Functions you should use
    - self.getLegalActions(state)
      which returns legal actions for a state
"""
def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"
    self.q_values = util.Counter()

def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    return self.q_values[(state,action)]


def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    valid_actions = self.getLegalActions(state) # get all valid actions
```

```
        if len(valid_actions) == 0: # if there is no possible action return 0
            return 0
        # compute max value from q-values
        ans = float("-Inf")
        for action in valid_actions:
            ans = max(ans, self.getQValue(state, action))

        return ans # return the best value

    def computeActionFromQValues(self, state):
        """
          Compute the best action to take in a state.  Note that if there
          are no legal actions, which is the case at the terminal state,
          you should return None.
        """
        "*** YOUR CODE HERE ***"
        actions = self.getLegalActions(state) # get all valid actions
        if not actions: # if there is no possible action return none
            return None
        # get q-values for every legal action
        q_values = util.Counter()
        for action in actions:
            q_values[action] = self.getQValue(state, action)

        return q_values.argMax() # return the best action based on q-values

    def update(self, state, action, nextState, reward):
        """
          The parent class calls this to observe a
          state = action => nextState and reward transition.
          You should do your Q-Value update here

          NOTE: You should never call this function,
          it will be called on your behalf
        """
        "*** YOUR CODE HERE ***"
        # calculating Q(s,a) given sample. from the known formula
        sample=reward+self.discount*self.getValue(nextState)
        self.q_values[(state,action)] = (1-self.alpha)*self.getQValue(state,action)+ self

    def getPolicy(self, state):
        return self.computeActionFromQValues(state)

    def getValue(self, state):
```

```
        return self.computeValueFromQValues(state)
```

Για να εκτελέσουμε αυτό το ερώτημα και το γραφικό του περιβάλλον γράφουμε την εντολή python gridworld.py -a q -k 5 -m. Εμφανίζεται ο κόσμος gridworld και εμείς χρησιμοποιώντας τα βελάκια κινούμε τον πράκτορα για να εξερευνήσουμε τον κόσμο. Καθώς το κάνουμε αυτό, ο πράκτορας μαθαίνει. Μετά από 5 φορές που θα τερματίσει το παιχνίδι ο πράκτορας μας θα έχει μάθει λίγα πράγματα για τον κόσμο. Στην γραμμή εντολών θα εμφανίζονται οι πληροφορίες που πήρε ο πράκτορας μας μετά από κάθε κίνηση. Στο τέλος θα εμφανιστεί "AVERAGE RETURNS FROM START STATE: " που θα είναι το μέσο συνολικό σκορ που έκανε.

7. Question 5: Epsilon Greedy

Στο προηγούμενο ερώτημα 4 έπρεπε χειροκίνητα να χειριστούμε τον πράκτορά μας για να μάθει στο περιβάλλον του gridworld. Σε αυτό το ερώτημα προσπαθούμε να κάνουμε αυτή τη διαδικασία αυτοματοποιημένη. Εδώ εμφανίζεται το γνωστό δίλημμα exploration vs exploitation. Αυτό σημαίνει οτι πρέπει ο πράκτοράς μας να πάρει να επιλέξει μεταξύ του να διαλέξει μια τυχαία ενέργεια, ώστε να μάθει περισσότερα πράγματα για τον κόσμο του, ή να ακολουθήσει την πολιτική που πιστεύει προς το παρόν οτι είναι βέλτιστη. Αυτό λύνεται επιλέγοντας κάθε φορά έναν τυχαίο αριθμό $\in [0, 1]$ και όταν αυτός ο αριθμός είναι μικρότερος απο μια παράμετρο ϵ που έχουμε ορίσει εμείς τότε ο πράκτοράς μας επιλέγει μια τυχαία ενέργεια.

Σε αυτό το ερώτημα έγραψα την συνάρτηση getAction του αντικειμένου QLearningAgent, στο αρχείο qlearningAgents.py, η οποία ακολουθεί την παραπάνω λογική:

```
def getAction(self, state):
    """
      Compute the action to take in the current state.  With
      probability self.epsilon, we should take a random action and
      take the best policy action otherwise.  Note that if there are
      no legal actions, which is the case at the terminal state, you
      should choose None as the action.

      HINT: You might want to use util.flipCoin(prob)
      HINT: To pick randomly from a list, use random.choice(list)
    """
    # Pick Action
    legalActions = self.getLegalActions(state)
    "*** YOUR CODE HERE ***"
    if len(legalActions) == 0: # if there are no possible actions return none
        return None
```

```
    # choose to follow the current policy or take a random action with probability epsilo
    if util.flipCoin(self.epsilon):
        action = random.choice(legalActions)
    else:
        action = self.getPolicy(state)

    return action # return the chosen action
```

Για να εκτελέσουμε αυτό το ερώτημα και το γραφικό του περιβάλλον γράφουμε την εντολή python gridworld.py -a q -k 100. Προσοχή παίρνει πολύ χρόνο! Παρακολουθούμε τον πράκτορα καθώς μαθαίνει στον κόσμο του gridworld, βλέπουμε ζωντανά τις τιμές της συνάρτησης Q να αλλάζουν.

Τώρα μπορούμε να εκτελέσουμε και τον crawler με την εντολή python crawler.py. Παρακολουθούμε τον crawler που προσπαθεί να μάθει να κινείται χρησιμοποιώντας τις ίδιες συναρτήσεις που γράψαμε για τον πράκτορα στο gridworld. Μπορούμε να παρατηρήσουμε οτι αν μετά από κάποια δευτερόλεπτα πχ 20 αν μειώσουμε το ϵ ο πράκτοράς μας αρχίζει και κινείται. Είναι προφανές οτι το ϵ πρεπει να μειωθεί με το πέρασμα του χρόνου ή και να γίνει 0 μετά από κάποιο "μεγάλο" αριθμό επαναλήψεων. Με αυτό τον τρόπο ο πράκτοράς μας σταματάει να πειραματίζεται και αρχίζει να ακολουθεί την "βέλτιστη" πολιτική που έχει μάθει.

8. Question 6: Bridge Crossing Revisited

Αυτό το ερώτημα μας ζητάει να "αναγκάσουμε" τον πράκτορα του BridgeGrid (με default παραμέτρους discount=0.9, noise=0.2 livingReward=0 episodes=50) να διασχίσει την γέφυρα (δηλαδή να μάθει την βέλτιστη πολιτική) χρησιμοποιώντας την Epsilon Greedy αλλάζοντας μόνο τις παραμέτρους της epsilon (-e), learning rate (-l). Για να δούμε αυτό να εκτελείτε γράφουμε python gridworld.py -a q -k 50 -n 0 -g BridgeGrid -e 1 -l 1 (μπορούμε να χρησιμοποιήσουμε και το -q για να κλείσουμε το γραφικό περιβάλλον). Μετά από λίγη σκέψη, αν έχουμε καταλάβει διαισθητικά πως λειτουργεί η Epsilon Greedy, μπορούμε να καταλάβουμε οτι αυτό είναι αδύνατο να γίνει σε 50 επεισόδια. Θα εξηγήσω παρακάτω.

Μπορούμε να πειράξουμε 2 παραμέτρους epsilon και learning rate. Υπάρχουν 3 περιπτώσεις:

- Το epsilon=1 τότε ο πράκτορας θα κάνει τυχαίες πράξεις ασχέτως της πολιτικής που γνωρίζει. Σε αυτήν την περίπτωση η πιθανότητα να διασχίσει την γέφυρα είναι πάρα πολύ μικρή, γιατί σχεδόν οποιοδήποτε λάθος βήμα τερματίζει το παιχνίδι.
- Το epsilon<1 και το learning rate=0. Τότε είναι η ίδια περίπτωση με την προιγούμενη.
- Το epsilon<1 και το learning rate>0. Τότε ο πράκτορας έχει τεράστια πιθανότητα να καταλήξει στην +1 τερματική κατάσταση πριν καταλήξει στην +10 (αν καταλήξει ποτέ εκεί, το οποίο είναι πολύ απίθανο). Αν συμβεί αυτό τότε ο πράκτορας θα ξεκινήσει να

προτιμάει τις ενέργειες που τον πηγαίνουν σε αυτή την κατάσταση +1. Βλέπουμε λοιπόν οτι είναι πολύ απίθανο, με αυτές τις παραμέτρους, ο πράκτορας φτιάξει μια πολιτική που να πηγαίνει στην τελική κατάσταση +10 μέσα σε μόλις 50 επαναλήψεις.

Μπορούμε να φανταστούμε οτι σε πολύ περισσότερες επαναλήψεις, με σωστές παραμέτρους ο πράκτορας θα μάθει να περνάει την γέφυρα. πχ python gridworld.py -a q -k 5000 -n 0 -g BridgeGrid -e 1 -l 1 -q.

9. Question 7: Q-Learning and Pacman

Αυτό το ερώτημα δεν απαιτούσε την συγγραφή κώδικα η την λύση κάποιου προβλήματος. Σε αυτό το ερώτημα φεύγουμε από το gridworld και μπαίνουμε στον κόσμο του pacman! Αρχίζουμε να πειραματιζόμαστε για να καταλάβουμε πως το q-learning μέσω της epsilon greedy δεν είναι πολύ ρεαλιστική λύση για πραγματικά προβλήματα.

Αρχικά εκτελώντας την εντολή python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid περιμένουμε τον πράκτορά μας να μάθει αλληλεπιδρώντας με το περιβάλλον του 2000 φορές. Αυτό γίνεται χωρίς γραφικό περιβάλλον γιατί είναι χρονοβόρο. Κάθε 100 παιχνίδια εμφανίζεται στην γραμμή εντολών μια αναφορά για την πρόοδο του pacman. Μετά από 2000 παιχνίδια, που ο pacman εξερευνεί τον κόσμο του με την epsilon greedy, αλλάζουμε αυτόματα το epsilon=0 και alpha=0 για να σταματήσουμε την φάση του πειραματισμού-μάθησης και να δούμε "τι έχει μάθει" (να τον δούμε να ακολουθεί την "βέλτιστη" πολιτική). Σε αυτή την φάση ο pacman παίζει 10 παιχνίδια με γραφικό περιβάλλον. Μπορούμε να αλλάξουμε τις παραμέτρους του κόσμου με το όρισμα -a πχ -a epsilon=0.1,alpha=0.3,gamma=0.7. Παρατηρούμε οτι ο pacman μετά από 2000 "δοκιμαστικά" παιχνίδια έχει μάθει αρκετά καλά ώστε να νικάει τουλάχιστον το 90% των "πραγματικών" παιχνιδιών.

Τώρα, ας βάλουμε τον pacman στον κόσμο του mediumGrid. Ακολουθώντας τα ίδια βήματα με πριν βάζουμε τον pacman να "προπονηθεί" σε αυτόν τον κόσμο και παρακολουθούμε την επίδοση του python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l mediumGrid. Παρατηρούμε οτι η διαδικασία "προπόνησης" παίρνει πολύ περισσότερο χρόνο από πριν και οτι ο pacman χάνει σχεδόν όλα τα παιχνίδια στην φάση "τεσταρίσματος". Αυτό είναι αρκετά προφανές. Αυτός ο κόσμος φαίνεται να έχει λίγα περισσότερα τετραγονάκια, αλλά αν σκεφτούμε οτι έχει τέσσερα φαγητάκια (άσπρες τελίτσες) και ένα φαντασματάκι, βλέπουμε ξεκάθαρα οτι οι καταστάσεις είναι πολύ περισσότερες από τον προηγούμενο κόσμο! Είναι λογικό λοιπόν οτι θα χρειαστούν πολύ περισσότερα παιχνίδια για τον pacman ώστε να μάθει να παίζει αποδοτικά (να αναπτύξει μια πολιτική κοντά στην βέλτιστη). Μπορούμε να φανταστούμε πόσο πιο δύσκολο θα είναι, ο pacman, να μάθει να παίζει στον κλασικό πραγματικό κόσμο του. Καταλαβαίνουμε λοιπόν οτι η μέθοδος που χρησιμοποιούμε μέχρι τώρα δεν είναι αποδοτική για ρεαλιστικά προβλήματα.

10. Question 8: Approximate Q-Learning

Σε αυτό το ερώτημα θα λύσουμε το πρόβλημα που υπάρχει στον τρόπο που μοντελοποιούμε τον κόσμο μας. Στο προηγούμενο ερώτημα καταλάβαμε οτι ο τρόπος μοντελοποίησης του κόσμου μας δεν είναι αποδοτικός για μεγαλύτερα πιο ρεαλιστικά προβλήματα.

Έχουμε έναν καινούριο τρόπο μοντελοποίησης του κόσμου μας τώρα ο οποίος θα μας βοηθήσει να χρησιμοποιήσουμε μια καινούρια μέθοδο λεγόμενη Approximate Q-Learning. Ορίζουμε μία συνάρτηση χαρακτηριστικών $f(s,a)$ που επιστρέφει ένα διάνυσμα $f_1(s,a)...f_n(s,a)$. Αυτή η συνάρτηση κωδικοποιεί τα χαρακτηριστικά μιας κατάστασης. Έτσι καταλήγουμε να έχουμε πολύ λιγότερες καταστάσεις και είναι πιο εύκολο για τον πράκτορά μας να τις επισκεφτεί όλες.

Διαισθητικά, σε σχέση με την προηγούμενή μοντελοποιησή, ο πράκτορας μας δεν χρειάζεται να επισκεφτεί κάθε ξεχωριστή κατάσταση για να τις μάθει όλες. Αυτό συμβαίνει γιατί όταν επισκεφτεί μια καινούρια-άγνωστη κατάσταση μπορεί να την συγκρίνει με μια άλλη γνωστή κατάσταση που έχει επισκεφτεί, χρησιμοποιώντας τα χαρακτηριστικά τους. Έτσι ο πράκτορας αρκεί να επισκεφτεί μια κατάσταση "από κάθε είδος" για να μάθει πως λειτουργεί ο κόσμος. Πρακτικά δεν κωδικοποιούμε όλες τις καταστάσεις, όπως πριν, απλά κωδικοποιούμε τα χαρακτηριστικά τους.

Η συνάρτηση $Q(s,a)$ μετατρέπεται ως γραμμικός συνδυασμός των χαρακτηριστικών του ζεύγους (s,a) επί ενός βάρους w_i. Δηλαδή:

$Q(s,a) = \sum_{i=1}^{n} f_i(s,a)w_i$

Τα βάρη w_i ενημερώνονται όπως και η συνάρτηση Q, δηλαδή:

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a), \quad difference = R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

Σε αυτό το ερώτημα έγραψα τις συναρτήσεις getQValue, update, final του αντικειμένου ApproximateQAgent, στο αρχείο qlearningAgents.py, η οποία ακολουθεί την παραπάνω λογική:

```
class ApproximateQAgent(PacmanQAgent):
    """
       ApproximateQLearningAgent

       You should only have to overwrite getQValue
       and update.  All other QLearningAgent functions
       should work as is.
    """
    def __init__(self, extractor='IdentityExtractor', **args):
```

```
        self.featExtractor = util.lookup(extractor, globals())()
        PacmanQAgent.__init__(self, **args)
        self.weights = util.Counter()

    def getWeights(self):
        return self.weights

    def getQValue(self, state, action):
        """
          Should return Q(state,action) = w * featureVector
          where * is the dotProduct operator
        """
        "*** YOUR CODE HERE ***"
        ans = 0
        # calculating sum of w_i * featureVector_i
        for feat, val in self.featExtractor.getFeatures(state, action).iteritems():
            ans += self.weights[feat] * val
        return ans

    def update(self, state, action, nextState, reward):
        """
           Should update your weights based on transition
        """
        "*** YOUR CODE HERE ***"
        # calculating w-i given sample. from the known formula
        diff = reward + self.discount * self.getValue(nextState) - self.getQValue(state,
        for feat, val in self.featExtractor.getFeatures(state, action).iteritems():
            self.weights[feat] += self.alpha * diff * val

    def final(self, state):
        "Called at the end of each game."
        # call the super-class final method
        PacmanQAgent.final(self, state)

        # did we finish training?
        if self.episodesSoFar == self.numTraining:
            # you might want to print your weights here for debugging
            "*** YOUR CODE HERE ***"
            pass
```

Αυτό το ερώτημα μπορούμε να το ελέγξουμε στο γραφικό του περιβάλλον σε διαφορετικούς κόσμους γράφοντας μια από τις παρακάτω εντολές:

- python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid. Έτσι βάζουμε τον πράκτορα στο ίδιο περιβάλλον που τεστάραμε στο προηγούμενο ερώτημα 7.

- python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid. Έτσι βάζουμε τον πράκτορα σε ένα μεγαλύτερο περιβάλλον. Παρατηρούμε οτι χρειάζεται μόνο 50 "δοκιμαστικά" παιχνίδια για να μάθει να παίζει καλά, σε αντίθεση με πριν που ήθελε 2000!

- python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic. Έτσι βάζουμε τον πράκτορα σε ένα ακόμα μεγαλύτερο περιβάλλον με δύο φαντασματάκια. Παρατηρούμε οτι χρειάζεται μόνο 50 "δοκιμαστικά" παιχνίδια για να μάθει καλά! (μπορεί να πάρει λίγο χρόνο να προπονηθεί)

Μόλις εκτελέσουμε μια από τις παραπάνω εντολές περιμένουμε τον πράκτορά μας να μάθει αλληλεπιδρώντας με το περιβάλλον του. Αυτό γίνεται χωρίς γραφικό περιβάλλον γιατί είναι χρονοβόρο. Κάθε λίγα παιχνίδια εμφανίζεται στην γραμμή εντολών μια αναφορά για την πρόοδο του pacman. Μετά από τα παιχνίδια που ο pacman εξερευνεί τον κόσμο του με την epsilon greedy, αλλάζουμε αυτόματα το epsilon=0 και alpha=0 για να σταματήσουμε την φάση του πειραματισμού-μάθησης και να δούμε "τι έχει μάθει" (να τον δούμε να ακολουθεί την "βέλτιστη" πολιτική). Σε αυτή την φάση ο pacman παίζει 10 παιχνίδια με γραφικό περιβάλλον.

11. Βιβλιογραφία

UC Berkeley CS188 Intro to AI – Course Materials `http://ai.berkeley.edu/lecture_slides.html`:
Lecture 8: MDPs I
Lecture 9: MDPs II
Lecture 10: Reinforcement Learning I
Lecture 11: Reinforcement Learning II

Δημήτρησ Μπερτσεκάς Dynamic Programming and Optimal Control, Vol. I

Δημήτρησ Μπερτσεκάς Dynamic Programming and Optimal Control, Vol. II

Εκφώνηση: `http://ai.berkeley.edu/reinforcement.html`