CSC 580: Artificial Intelligence II
Winter 2026

Homework #2: Snake Game by Q-Learning

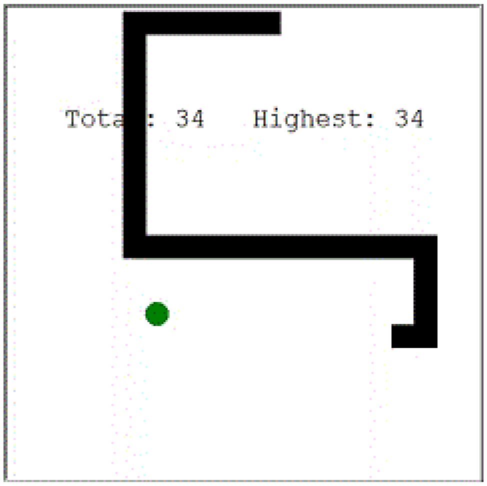
This assignment is to code and run experiments with the Snake Game using a custom environment. It consists of two parts, first to complete an agent who performs Q-Learning, and next to run experiments with the epsilon and gamma parameters.

Snake Game

A snake game is an arcade maze game which has been developed by Gremlin Industries and published by Sega in October 1976. It is considered to be a skillful game and has been popularized among people for generations. The snake in the Snake game is controlled using the four direction buttons relative to the direction it is headed in. The player's objective in the game is to achieve maximum points as possible by collecting food or fruits. The player loses once the snake hits the wall or hits itself. ["Create a Snake-Game using Turtle in Python"]

We implement the game as a Reinforcement Learning, in particular, an RL agent who learns to play using Q-Learning. We also use the Turtle graphics library.

For the environment, we create our own custom environment (inheriting from the OpenAI/Gymnastics Env class). In our implementation, a snake is represented by a contiguous black squares and the food is a apple (a green circle). The snake extends its length as it eats apples. The RL agent controls the snake, in particular in deciding the next action to take (move up/right/down/left).



[gif animation]

For more information on the environment, look at the **Gymnasium Documentation page**.

Preliminary

You start by reading and understanding the start-up code. You can download individual files from below, but course GitHub site (**https://github.com/ntomuro/CSC580_Winter2026**) has them too.

1. Custom environment -- **"SnakeEnv.py"**; completely done
2. Agent -- **"Agent.py"**; partially done
3. Q-Learning/Game -- **"QLearning.py"**: partially done
4. additional library dependency -- "requirements.txt"

Code Setup:

- To set up the code to enable the graphic rendering, we now use **.py script files and run code from the command line**.
- You should create a new folder/subdirectory and place all files above in the folder (or clone from the github).
- Since the code requires additional libraries (Gymnasium and matplotlib in particular), we should create a **virtual environment** and install the libraries inside the environment so as not to cause conflict with other system/library setup. If you are new to virtual environment, these references would be helpful:
    - https://python.land/virtual-environments/virtualenv
    - https://www.geeksforgeeks.org/create-virtual-environment-using-venv-python/
    - https://realpython.com/python-virtual-environments-a-primer/

How to Run -- Basics

1. First create a virtual environment and install dependencies. First navigate to the folder. Then do these:

```
python -m venv venv
source venv/bin/activate  # or run venv/Scripts/activate for windows PC
pip install -r requirements.txt
```

2. Then run the 'QLearning.py' file

```
python QLearning.py
```

The last (Q-Learning) code will run an application and run with no errors.

Note: the SnakeEnv defines these (as you can see in the source ipynb code):

- Action:
    - up -- 0
    - right -- 1
    - down -- 2
    - left -- 3
- Reward:
    - Snake eats an apple -- **+10**
    - Snake comes closer to the apple -- +1
    - Snake goes away from the apple -- -1
    - Snake dies (hits the wall or itself) -- **-100**
- State
    1. Apple is above the snake -- 0 or 1
    2. Apple is on the right of the snake -- 0 or 1
    3. Apple is below the snake -- 0 or 1
    4. Apple is on the left of the snake -- 0 or 1
    5. Obstacle directly above the snake -- 0 or 1
    6. Obstacle directly on the right -- 0 or 1
    7. Obstacle directly below -- 0 or 1
    8. Obstacle directly on the left -- 0 or 1
    9. Snake direction == up -- 0 or 1
    10. Snake direction == right -- 0 or 1
    11. Snake direction == down -- 0 or 1
    12. Snake direction == left -- 0 or 1

Comments:

- There is exactly one apple in the maze at any given time.
- A reward is given for each/every action.
- State returned by the environment is a **list of twelve binary digits, e.g. [0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0]**. It is up to you how you use the format (or possibly convert to a different format) internally in Agent. For your convenience, I've added a few **static methods** that convert a list of binary digits to a string or an int.
- Note that, although the entire state space is large (2^12 = 4096), only a subset of the states are possible/reachable because of the definition of the state -- 640 of them (shown at the bottom of the SnakeEnv.py).

Part 1. Q-Learning Agent

(1) The first task for this part is to complete the code for agent (class Agent in **"Agent.py"**).

1. The most important thing for you is to decide how, in **what data structure, you represent/implement the Q-table**. It is indicated in the code in the __init()__ function:

   ```
   ## TO-DO: Choose your data structure to hold the Q table and initialize it
   self.Q = None
   ```

   You could use any data structure, for example, a 2D array, a 3D array, a dictionary, a pandas dataframe, or anything else as you think is convenient.

   Note that Python dictionary does not hash on lists (or aggregates/objects). If you want to use dictionary as the data structure, you must convert the list of binary ints to a scalar, such as a string or even to an integer, and use that as the key in a dictionary.

2. Functions which you must fill/modify are indicated with "**# (A)**" at the top. Each one has a docstring, and **you should be able to follow and implement it.** Some of them have a comment "# for now" -- the lines there are temporary just to run the initial code. You must elaborate the function.

3. As for the function "**update_Qtable()**", it MUST implement the **TD Q-learning, shown on Slide 26 in Lecture Note #4**. Parameters alpha and gamma are stored in the agent (self.gamma and self.alpha), which you do not need to change at all. Also be sure to leave the last two lines as is. The function adjust_epsilon() is already written, which lets epsilon decay after every timestep.

   ```
           # update the epsilon too
           self.adjust_epsilon()
   ```

4. As for the function "**write_qtable()**", you write the resulting Q-table (i.e., self.Q) to an external, comma-delimited (csv) file. The format should be "state, action, qa_value" for each QA pair, sorted by the states (then by actions), for example,

   ```
   257, 0, -0.23450000000000004
   257, 1, 0.0
   257, 2, 6.6974670724502126
   257, 3, 15.267493558134351
   258, 0, -0.23450000000000004
   258, 1, -0.7
   258, 2, 3.2897875334196067
   258, 3, 0.0
   ```

```
260, 0, 0.0
260, 1, 0.0
260, 2, 0.6864199763653377
260, 3, 0.0
264, 0, 0.0
264, 1, 0.0
264, 2, -0.7
264, 3, 12.122551711757653
```

Note that, in the above, states in the first column are integers converted from the original list of binary ints. You can represent states in any way -- whatever you used would be the first column in the output file, and that's fine.

5. The "**read_qtable**()" function should read in the learned policy, saved in the csv file (by the write_qtable() function).

(2) The second task in this part is to train the agent and save the derived Q-table to a file.

To train the agent, you can comment/uncomment the line that calls run_ql() with training=True in "QLearning.py".

```
## Call run_ql() for either/both training and evaluation
num_runs = 1      #10
num_steps = 1000 #300

params = dict()
params['gamma'] = 0.95
params['alpha'] = 0.7
params['epsilon'] = 0.6  # exploration probability at start
params['epsilon_min'] = .01  # minimum epsilon
params['epsilon_decay'] = .995  # exponential decay rate for epsilon

qtable_file = "qtable_2025.csv" #"qtable_true.csv" #None

# Call run_ql() for either training or evaluation
#results_list = run_ql(num_runs, num_steps, params, qtable_file, display = True, train
results_list = run_ql(num_runs, num_steps, params, qtable_file, display = False, train
```

Note that 'qtable_file' specifies the name of the learned Q-table to be written/created if the call was made for training.

----

(3) The third task in this part is to evaluate the learned Q-table.

You can do so by uncommenting the evaluation line and commenting the training line

```
# Call run_ql() for either training or evaluation
results_list = run_ql(num_runs, num_steps, params, qtable_file, display = True, train =
#results_list = run_ql(num_runs, num_steps, params, qtable_file, display = False, train
```

This run will read the saved qtable_file and run the number of 'num_steps' (and show the stats).

NOTE:

- Repeat the tasks (2) and (3) -- (2) followed by (3) -- **at least 3 times** with different parameter values (especially epsilon and num_steps) to see if you observed **noticeable differences in the snake behavior** by different derived Q-table. Write your reactions and analysis in the write-up.

- In your analysis/reactions, be sure to refer to the evaluation results/outputs as well (in the comparison). For example, here are some example outputs:

```
## Random action choice

* Run 0: Return= -12.302, #Apples=0, #Stops=9, #GoodSteps=140, #UniqueStatesVisite

** Mean: Return= -12.302, #Apples=0.0, #Stops=9.0, #GoodSteps=140.0, #UniqueStates

----------------------------------------------------------------------
## Evaluating one derived Q-table

* Run 0: Return=  26.722, #Apples=20, #Stops=4, #GoodSteps=265, #UniqueStatesVisit

** Mean: Return=  26.722, #Apples=20.0, #Stops=4.0, #GoodSteps=265.0, #UniqueState

----------------------------------------------------------------------
## Evaluating another derived Q-table

* Run 0: Return= -32.699, #Apples=17, #Stops=3, #GoodSteps=265, #UniqueStatesVisit

** Mean: Return= -32.699, #Apples=17.0, #Stops=3.0, #GoodSteps=265.0, #UniqueState
```

Part 2. Epsilon Experiment

Your task for this part is to do some experiments. The main focus is the **exploration vs. exploitation** tradeoff. To that end, you run experiments with various epsilon values and compare results on several aspects.

Use the learned Q-table from Part 1 for this part.

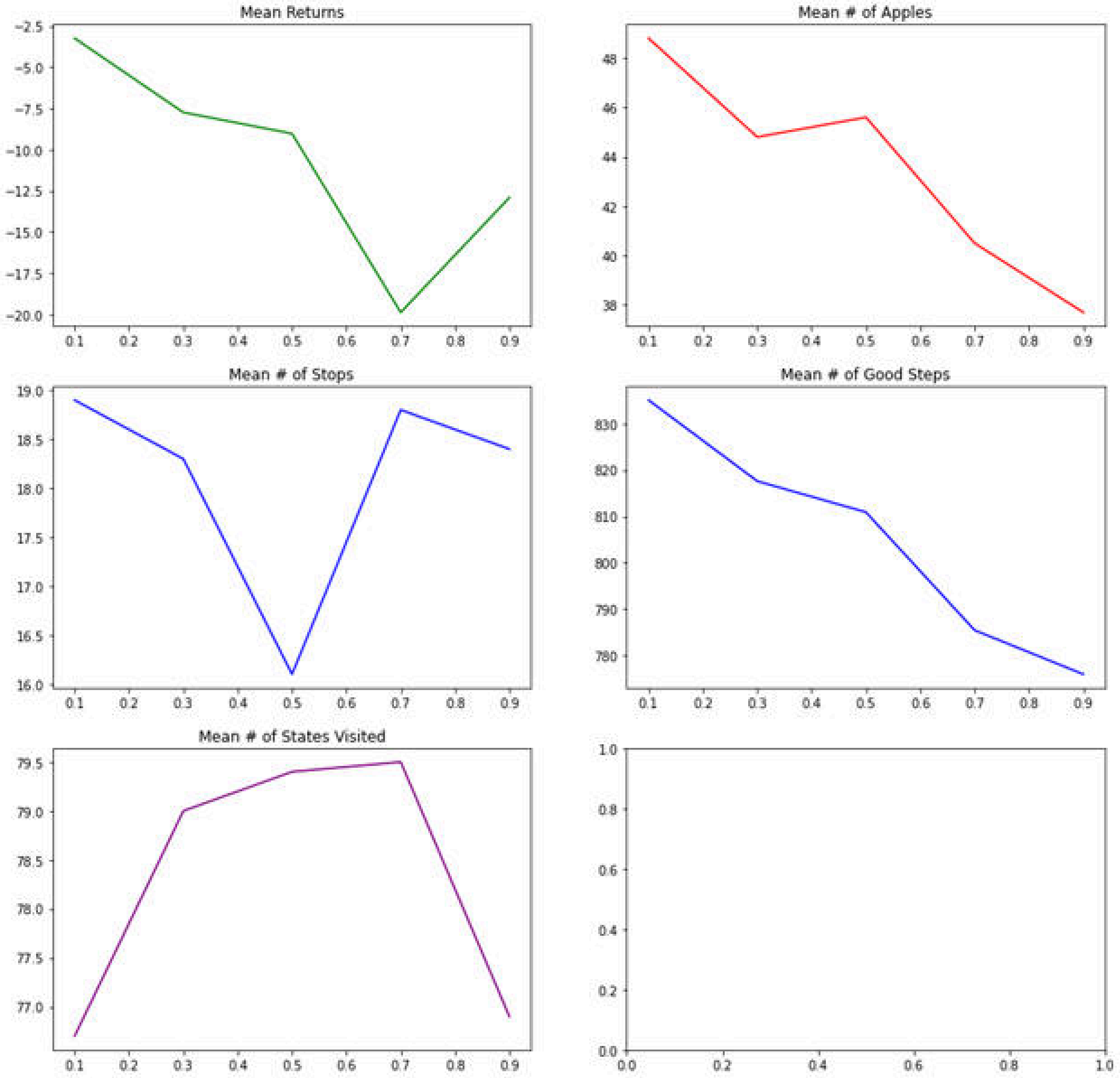
- Do experiments with epsilons 0.1, 0.3, 0.5, 0.7 and 0.9. And for each value, do **1,000 timesteps, for 10 runs**. Those are defined by the variables num_runs and num_steps.
  Note: Be sure to set 'display' to False.

```
num_runs = 1
num_steps = 1000
```

- For each epsilon value, you will get the mean statistics at the end, for instance

```
** Mean: Return= -20.146, #Apples=2.0, #Stops=38.0, #GoodSteps=477.0, #UniqueState
```

Create a line graph for each aspect that shows the performance by various epsilons. For example,



- Also, this won't count for extra credit, if you like to do more experiments, try other parameters such as gamma, alpha and epsilon_decay. It will be fun!

Deliverables

Submit the following:

1. Your source code files, "**Agent.py**" (completely filled). Also if you changed anything at all in "**Q-Learning.py**", submit the file as well.
   Each file must have your name, student ID, course number and assignment number (HW#2) at the top of the file. Failure to comply with this will result in significantly reduced grades.
2. **The Q-table output file (in csv).**
3. A write-up document.
   - **Minimum 2.5 pages** (in pdf or docx).
   - **Your name, student ID, course number and assignment number (HW#2)** at the top of the document. Failure to comply with this will result in significantly reduced grades.
   - **Collaborators** if you worked with other students or used GenAI tools. Claim any reference sites you consulted as well.

- How you implemented the Q-table. Reasoning as to why you chose the data structure. Could it be optimized? If so, what are possible ideas?
- *Detailed* explanations of how the values in the table are updated (with examples).
- Results and your comments of Part 1 experiments. What were the training statistics? What did you observe in the behavior of the snake for different derived Q-tables? Write informative reflections and analyses.
- Results and your comments of Part 2 experiments. Comments must include whether or not the results came out as you had expected, and your speculations as to why the results came out the way they did.

- Your general reflections:
  - What you learned from this exercise.
  - How difficult you felt this exercise was.
  - Any particular difficulties you encountered.
  - How you would do/approach differently next time (if there was one).
  - and anything else.

Submissions

In the submission box 'HW#2'. Do NOT zip the files -- Upload all files separately.