COMP 4010A – Environment Demo
RL for Tower Defense with Evolutionary Towers
Fall 2025

Due: October 27, 2025 23:59.

Group 12
Andrew Wallace - 101210291 - andrewwallace3@cmail.carleton.ca
Mohammad Rehman - 101220514 - mohammadrehman@cmail.carleton.ca
Manal Hassan - 101263813 - manalhassa@cmail.carleton.ca
Derrick Zhang - 101232374 - derrickzhang@cmail.carleton.ca

Overview [@Derrick]

The initial environment is a simplified version of the final project. This document will outline the preliminary environment specifications and reinforcement learning algorithm.

Out of Scope [@Derrick]

The following features are out of scope for the initial environment:

1. Budget constraints
2. Multiple tower types
3. Multiple enemy types
4. Incremental increase in wave difficulty
5. Tower level-ups

MDP [@Derrick]

Below is a description of the `TowerDefenseWorld` environment as a Markov Decision Process.

State space

State is currently represented by the layout of the grid at a given time step. Specifically, a single state is stored as a 3d box with shape $(n, n, 3)$ where n represents the $n \times n$ grid. We store an array of size three in the third dimension to hold additional information. The first value of the array is 0 or 1 to indicate if the cell is a part of a path or not. The second is the tower hp or 0 if no tower exists in that cell. The third is the enemy hp or 0 if no tower

exists in that cell.
The size of our state space is given by

$$((T+1)\cdot(E+1)\cdot 2)^{n\times n} \tag{1}$$

where
T = the max hp of a tower
E = the max hp of a enemy
$n \times n$ is the number of cells in the grid

Currently our environment has
$T = 28$
$E = 13$
$n = 5$
This results in a total state space size of:
$((28+1)\cdot(13+1)\cdot 2)^{25} = 232^{25}$

Action Space

Agent Actions

There are 26 total actions the agent can take at the beginning of a wave. Do nothing, or place a tower in one of the 25 positions.
`self.action_space = spaces.Discrete(size * size + 1)`

Enemy Actions

Note, that the enemy actions are not a part of the action action space, however they are mentioned here for clarity.
The agents have 4 possible actions: do nothing, move down, move left, move right.
`self.enemy_action_space = spaces.Discrete(4)`

Reward Structure

1. Enemy Defeated = +10
2. Tower Defeated = -5
3. Tower Damaged = -1
4. Enemy Reaches Base = -50
5. Tower Level's up = +5 (not utilized yet)
6. Wave Cleared = +20

7. All Waves Cleared = +200
8. Base Destroyed = -10

@ Derrick, please add anything else that you would like, thank you!

Code Walkthrough [@Andrew]

__init__()

1. Environment initialization
2. Setting up

reset()

1. New episode

step()

1. Transition to next state and emit a reward
2. Return (observation, reward, terminated, truncated, info)

render()

1. Human readable output

Q-Learning Demo [@Manal]

For the initial environment, we implemented the Q-Learning Algorithm to demonstrate how the agent interacts with it. We run Q-Learning using the following parameters to approximate the Q-values:
Episodes = 10000
Step-size $\alpha = 0.5$
Discount factor $\gamma = 0.9$
Epsilon $\epsilon = 0.9$ that decays to 0.05 after episode 1500
Once we approximate the Q-values, we derive the approximate optimal policy and evaluate it's performance.

To show the agent-environment interaction, we used pygame to draw the 5×5 grid.
In the `_render_frame(self)` function, we have defined a `color_map` where empty cells are white, cells with towers have a green circle, cells with an enemy are red, and the path is a

light grey color.
Having the tower as a green circle allows for both towers and enemies to be displayed even if they are on the same cell.
To demonstrate, we will run `qlearn.py`
In the console, we periodically (every 100th episode) see the results of the Q-learning algorithm, showing the total reward, enemies destroyed, and towers destroyed.
Once we learn the Q-values, we evaluate the approximate optimal policy for 10 episodes, shown in a pygame window.
One interesting observation we made was that the agent will sometimes place towers at the beginning of a path to block an enemy from advancing, often allowing another tower to attack while the enemy is blocked.

Key takeaways

Q-learning appears to be an effective algorithm for learning and brings some interesting results. However, the state space is quite large and is going to grow as the project increases in complexity, which will make traditional tabular Q-Learning infeasible for finding the optimal policy. Our goal will be to transition from Q-Learning to Deep Q-Learning. We will also be exploring other algorithms such as Advantage Actor-Critic (A2C).

@ Manal, please add anything else that you would like, thank you!