ASSIGNMENT 3

In this project assignment, we coded an AI agent playing the game of Ms. Pacman within the Atari emulator (Stella) using the Arcade Learning Environment ALE. The agent coded for this project uses RL-Glue interface to communicate with the ALE. The C++ code could be found in the folder pacman_rlglue.

I. Introduction – Ms. Pacman

Ms. Pac-Man is an arcade video game from the Golden Age. The game has 4 ghosts and Ms. Pac-man in a maze setting (Figure 1). The player earns points by eating pellets and avoiding ghosts (contact with one causes Ms. Pac-Man to lose a life). Eating an energizer (or "power pellet") causes the ghosts to turn blue, allowing them to be eaten for extra points. Bonus fruits can be eaten for increasing point values, twice per round. As the rounds increase, the speed increases, and energizers generally lessen the duration of the ghosts' vulnerability, eventually stopping altogether. [Wikipedia]

The reward for eating a Normal Pellet is 10 and for Power Pellet the reward is 50. In the powered mode, eating ghosts consecutively yields reward of 200, 400, 800 and 1600 respectively. The detailed gameplay can be found at the Ms. Pac-man AtariAge Manual.

Figure 1: Ms. Pacman game screenshot

II. Setup instructions

1. RL-Glue installation:

As previously stated, the coded agent uses RL-Glue interface to communicate with the ALE. Thus RL Glue core and RL-Glue C/C++ codec must be installed on the system and can be found here as stated in the Manual for ALE.

RL-Glue is a combination of four processes: a core, an experiment, an agent, and an environment. The core is provided by the RL-Glue library itself, and ALE is the environment here. We coded the experiment (30 episodes) and the agent (provided in the folder *pacman_rlglue)*. These processes needs to be compiled as detailed below.

2. Compiling the Environment (ALE):

ALE needs to be compiled with -DUSE_RLGLUE = ON to use rlglue

```
sudo apt-get install cmake libsdl1.2-dev
cd <project_root_folder>
cmake -DUSE_SDL=ON -DUSE_RLGLUE=ON -DBUILD_EXAMPLES=ON
make -j 4
```

3. Compiling the Agent and the Experiment:

```
cd <project_root_folder>
cd pacman_rlglue
make
```

4. Launching the processes:

   Now we need to start the 4 processes using the following commands, each line in a new terminal

```
rl_glue
./pacman_rlglue/pacman_agent
./pacman_rlglue/pacman_experiment
./ale -display_screen true -game_controller rlglue
roms/ms_pacman.bin
```

III. Algorithm

Considering Pacman's location and those of ghosts and pellets for each state means a very large state space which is a burden. Thus we need to describe states through a combination a features: we generalize the description of state by the following:

- Sum of distances between Pacman and the ghosts:
- Sum of distances between Pacman and the remaining pellets
- Distance the closest corner (Will explain why later)
- Sum of distances between Pacman and the teleportation tunnels (the why will be explained later also)
- Game mode: normal or power mode

To compute these, the agent explores the environment using methods defined in the pacman_objects class that allows to detect its own location, maze walls, as well as ghosts and pellets locations. Using them, it computes the possible moves and the features and use them to decides which direction to go. The algorithm smart_agent employed to play the game is a reward based agent that tries to eat pellets and escape from the ghosts at every step in normal mode and tries to eat the ghosts in powered mode. At each position, Ms. Pac-man calculates all the possible moves it can take among UP, DOWN, LEFT and RIGHT. It calculates the reward of each possible next location and chooses to

Algorithm Smart Agent

```
1: function smartAgent(location, gameMode)
2:      diractionArray ← possibleMotion(location) .
possibleMotion function returns all the locations Ms. Pacman can reach by taking an action
3:      Initialize bestReward and bestDirection appropriately
4:      for each direction in directionArray do
5:           nextLocation  getNextLocation(location, direction) .
getNextLocation function returns the next location of Ms. Pacman given the current location and the direction of motion
6:           if gameMode is Normal then
7:                directionReward  Reward_normal_mode(nextLocation)
8:           else if gameMode is Powered then
9:                directionReward  Reward_powered_mode (nextLocation)
10:          if directionReward > bestReward then
11:               bestReward  directionReward
12:               bestReward  direction
13:     return bestDirection
```

go to the one with highest reward. In normal game mode, the reward $R_{normal\_mode}(loc)$ at a location loc is given by the following equation:

$$R_{normal\_mode}(loc) = \sum_{i=1}^{4} \frac{x}{distance(ghost_i, loc)} + \sum_{i=1}^{P} \frac{y}{distance(pellet_i, loc)} + \sum_{i=1}^{4} \frac{z * 1_{corner}(corner_i, loc)}{distance(corner_i, loc)} + \sum_{i=1}^{4} \frac{t}{distance(teleport_i, loc)}$$

where loc is the location of Ms. Pac-man, P denotes the total number of available pellets (left in the game, not eaten yet). $ghost_i$ is the location of i^{th} ghost in the game screen, $pellet_i$ is the location of i^{th} pellet. Now we would like to explain why a reward function including just the ghosts and pellets is not enough: trying with this, Ms. Pac-man got stuck in the corner of the maze most of the time. This could be explained by the fact that the corner points

become some kind of local minima, w.r.t. the reward function above, over a significant period of time and reaching these points make Ms. Pac-man stuck at the position, this is why the corner reward has been added, to push the agent from getting stuck there. To consider only the closest corner, we divided the maze into 4 regions with one corner in each of them. 1_{corner}(loc1, loc2) is the indicator function that equals to 1 if both loc1 and loc2 belong to the same region. Another problem is possible: Ms. Pac-man could oscillate in a teleportation tunnel as its end points are located on the edges of the maze. Teleportation rewards have been added to avoid this. When Pacman eats a power pellet, we move from normal to powered mode of the game. In this case, the agent heads toward the ghost. We do not need teleportation and corner reward here since Pacman tries to eat the ghost which are in continuous motion. The reward function reduces to:

$$R_{powered\_mode}(loc) = \sum_{i=1}^{G} \frac{w}{distance(edible\_ghost_i, loc)}$$

Where G is the number of edible ghosts present in the game.

We note that parameters x, y, z, t and w are chosen empirically in our case (because of lack of time) but they can be tuned as an optimization/training problem to get better performances.

IV. Results/ Performances

The complete results of the simulation could be found in the results_30_episodes.txt file where the numbers of performed steps, the total reward as well as the mean reward and standard deviation over 30 episodes are displayed (the number of simulated episodes could be modified in pacman_experiment.cpp changing the total_episodes parameter). Here is a screenshot of the result:

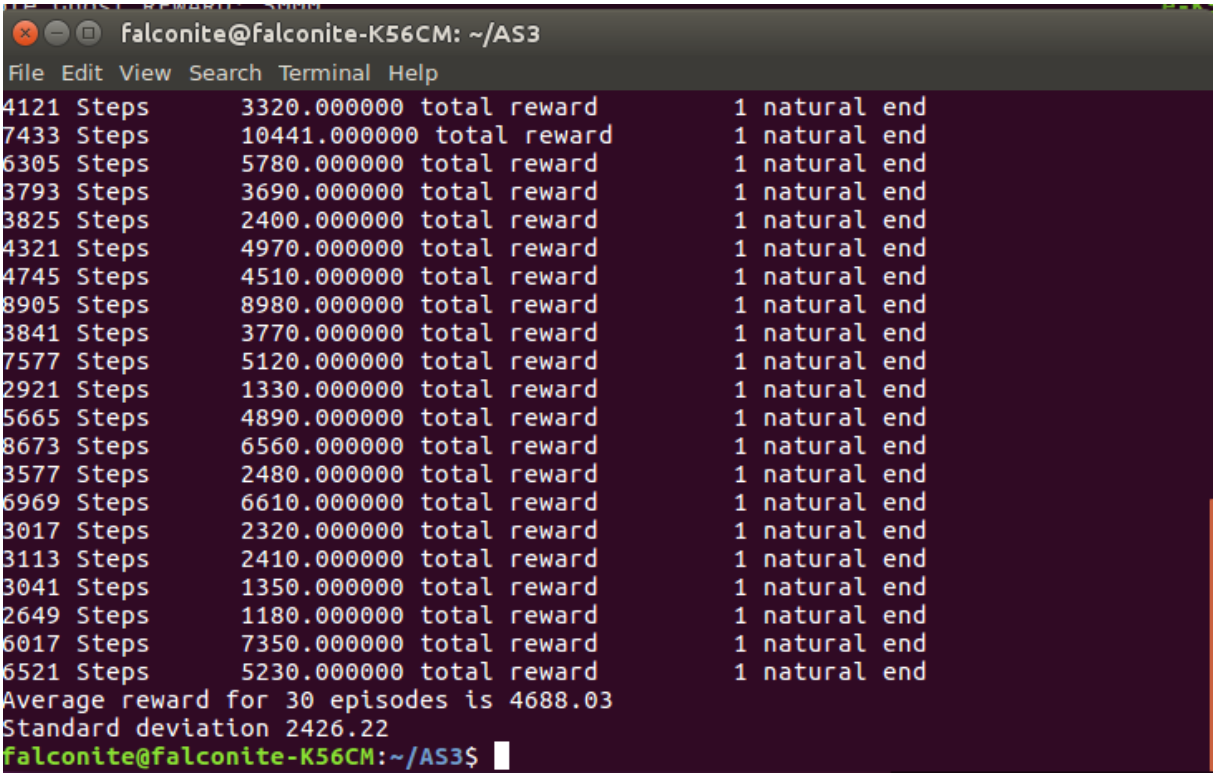

Figure 2: Screenshot of 30 episodes simulation result

To see how good these performances are, we compare them to results of [1] that deploys Deep Learning (Deep Q-Networks DQN) for several Atari games, among which Ms. Pacman. Comparison is summarized in the following table:

Algorithm	Mean	Standard deviation
DQN	2311	525
Ours, smart agent	4688.03	2426.22

We can see that this reward based algorithm outperforms the DQN one on average, but has higher standard deviation.

As stated previously, performance of our algorithm could be tuned through optimizing over the parameters x, y, z, t and w.

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529{533, 2015.