CSC3022H: Machine Learning

Reinforcement Learning Assignment:
Q-Learning

Department of Computer Science
University of Cape Town, South Africa

DUE: 11th October, 2019, 10.00 AM

Problem Description

Implement (in C++) a *Q-learning* [Watkins and Dayan, 1992] algorithm to solve a simulated mine-sweeping task.

Q-learning is to control autonomous mine sweeper agents in a two-dimensional world of *mines* and *super-mines* (figure 1). Exact simulation parameters (mines, sweepers, iteration length, frame rate, etc.) are set in the `CParams.ini` file.

If a sweeper collides with a super-mine both the sweeper and super-mine are destroyed. Destroyed super-mines are re-spawned in the next iteration. If a sweeper collides with a mine, that mine is removed from the world and the sweeper's statistics are updated.

The framework to set up the world, rendering and update loop is supplied. It should not be necessary to modify the framework's core functionality (draw and update loops), aside from the Q-learning controller and supporting methods:

- `CQLearningController.cpp`

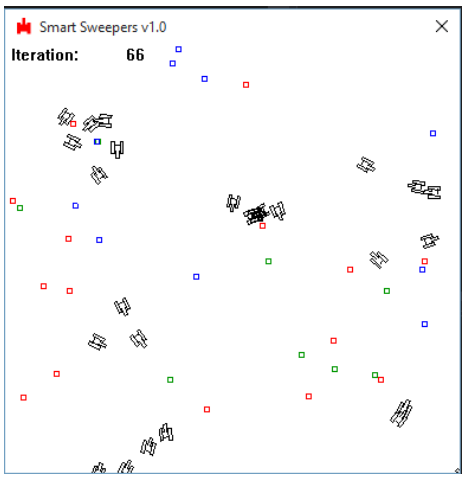

Figure 1: Simulation world consisting of *mine-sweepers*, *mines* and *super-mines*. Mines and sweepers are spawned at random positions at the start of a simulation. The world wraps around on both the x and y axis (i.e. if a sweeper moves off the world on one side they reappear on the opposite side). Sweepers should learn to remove mines (green squares) and avoid super-mines (red squares).

Each of the controllers implements a carefully defined interface.

`CQLearningController.cpp` is a discrete (grid) environment and inherits from `CDiscCollisionObject.cpp`.

The controller `CQLearningController.cpp` overrides the following methods:

- `virtual void Update(void);` This method must call the Update method of the parent controller.
- `virtual void InitializeLearningAlgorithm(void);`

You will need to fill in the details of the methods for the controller and supporting classes, paying special attention to the comments in each file.

Mine-sweeper statistics are drawn by the *PlotStats*() function when the F-key is pressed. Graphs of the *maximum* and *average number of mines swept* are drawn when your mine-sweepers learn to sweep mines. The framework uses the Windows `WIN32` windowing API and is bundled as a Visual Studio 2019 project. Both the Senior and the Games lab machines have Visual Studio installed.

Implementing Q-learning

For a detailed description of the Q-learning algorithm refer to the seminal Q-learning paper [Watkins and Dayan, 1992] and Chapter 13 of *Machine Learning* [Mitchell, 1997] (both are available on Vula)[1].

Note that the Q-learning algorithm keeps a table containing all possible *states* and *actions* and progressively updates the table according to the *reward function*. Since all possible state-action pairs have to be tracked the world must consist of a finite number of grid positions. You can assume that each mine-sweeper can only move *up*, *down*, *left* and *right* at every step of the update cycle.

When implementing Q-learning you must decide on a suitable reward function and algorithm parameters including the *learning rate* and *discount factor*. For example, sweepers could be rewarded for completing the task (clearing the world of mines) and penalized for finding nothing. However, *Q-learning must always run for* 2000 *iterations (simulation ticks) and each simulation must have mine-sweepers and mines initialised to random positions in the world.*

To verify that your agents have learned a suitable mine-sweeping behaviour, your Q-learned agent controllers must be evaluated on the following tests:

Test 1: Initialise the world with 1 *mine-sweeper* and 30 *mines*.

Test 2: Initialise the world with 1 *mine-sweeper*, 25 *mines* and 5 *super-mines*.

Test 3: Initialise the world with 1 *mine-sweeper*, 5 *mines* and 25 *super-mines*.

The success of each of these tests is mine-sweeper task performance, i.e. *average and maximum number of mines swept*, where this average is calculated over 50 simulation runs (note that for each test simulation, mine-sweepers, mines and super-mines must be initialised to random locations in the world).

SUBMIT: ZIP file containing the modified framework (excluding binaries) with your Q-learning implementation and a report (PDF doc) that includes:

- Results table: For tests 1, 2, and 3: *maximum number of mines swept* (from 50 runs) and *average number of mines swept* (over 50 runs).
- A brief justification (***200 words maximum***) of the Q-learning parameter values you selected and how these parameter values influenced learning and mine-sweeping task performance in each test simulation.

[1] You can also use the library *ezproxy* tool http://ezproxy.uct.ac.za/ to download Q-learning articles from sites such as IEEE Explore when you're off campus.

Notes on Submission

- You should run the *clean project* option before the ZIP and submit.
- You must submit a working Visual Studio 2019 Solution or a *makefile* to compile your program. If it does not compile a 50% penalty will apply.
- If you add additional files and or have any compilation instructions you must provide a README file explaining what each file submitted does and how it fits into the program as a whole. The README file should not explain any theory that you have used.
- Ensure that the ZIP archive of your git repo works and is not corrupt. Corrupt or non-working archives will not be marked. Please do not use other formats like WinRAR / 7zip / etc.

References

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning.* McGraw Hill, New York, USA.

[Watkins and Dayan, 1992] Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(1):279–292.