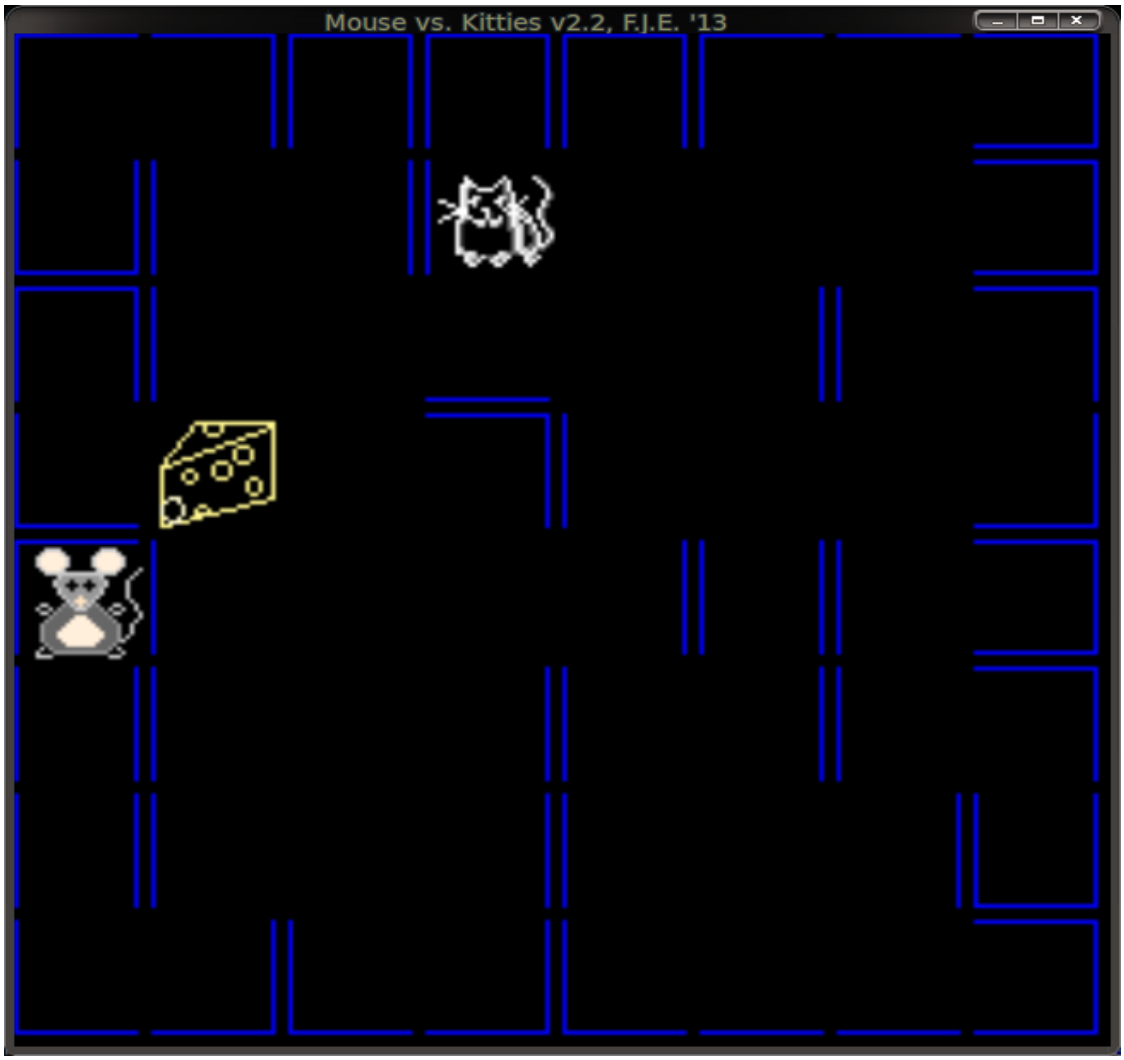
Assignment 3
Reinforcement Learning

(electronic submission on Quercus)

You can complete the assignment individually, or in teams of two



CSC D84 – Assignment 3
Reinforcement Learning

You have now implemented multiple ***search algorithms*** that can be used to guide agents to goals. In the previous assignment, ***minimax trees*** allowed you to observe agents trying to outsmart each other given a common utility function and knowledge of the game state. here, we will go one step further and study agents that ***learn*** from experience. The goal is to ***train*** an agent via ***rewards and punishments*** suitably mapped to specific actions and states.
Your agent will explore, learn, and create a ***policy*** mapping states to ***high reward actions***.

Learning Objectives:

You will strengthen your understanding of reinforcement learning, and in particular, ***Q-learning***

You will implement a simple reward function that can be used by an agent to determine which ***state → action*** pairs result in positive rewards

You will observe how the training process improves agent performance over time

You will understand the relationship between training time and agent performance, and think about the effect of the reward function on the amount of time needed to achieve good performance.

You will think about the state-space for a problem, and observe how quickly it can grow. This will show you potential practical limitations of reinforcement-based methods

You will study the issue of ***generalization*** in reinforcement training, and in this way gain an understanding of the types of problems on which reinforcement learning is likely to succeed

Skills Developed:

Implementing ***Q-learning***, one of the main techniques in reinforcement learning

Thinking about rewards, and how to map states to rewards

Estimating the amount of training required to bring an agent to a good level of performance

Iteratively refining a reward function to improve agent learning speed and agent performance

References:

Your in-class lecture notes on search.

Your course instructor and your TA!

A Mouse that Learns

In this assignment you will implement an agent that learns. Your mouse will start with no knowledge of the works other than this: ***getting eaten is bad, eating cheese is good***. This will be encoded in the form of a reward function. The mouse will then learn through exploration. By exploring different actions that are possible given the game's configuration, and by observing the rewards (or punishment) that result from these actions, the mouse will create a good ***policy*** that will allow it to decide given the state of the game what is the best action to take in order to maximize its reward.

This process is called ***reinforcement learning***. You will implement a specific form of is that Is called ***Q-learning***.

You need to implement the following components for this assignment:

- A reward function. This function determines what the agent learns. The reward Function should favour actions that are likely to lead to the mouse eating the cheese, and discourage actions that may lead the mouse to being eaten.
- The update rule for ***Q-learning*** which updates the estimated reward for each ***state → action*** pair.
- The action selection function that is used by the agent (once training is complete) to decide which action to perform given its current state.

All the code to implement the initialization and the training is given to you.

Step 1

Download and uncompress the starter code into an empty directory.

This code is designed for Linux. I recommend you do all your work on the CS lab at IC 406. You can work remotely by logging into ***mathlab.utsc.utoronto.ca*** but note that the program uses OpenGL graphical display and will run very slowly over ssh.

The starter code contains the following files:

Qlearn_core_GL.o
Qlearn.[c,h]
compile.sh
REPORT.TXT

All your work is done within the ***QLearn.c*** and ***QLearn.h*** files. Detailed comments in these files will tell you what you need to implement.

Read all the comments in Qlearn.[c,h]. The comments describe what needs to be implemented, and the conditions and constraints placed on your solution.

Feel free to ask for clarification at any time. But I expect you to have ***read this handout and all the comments carefully***.

A Mouse that Learns

Step 2

Test-run the starter code:

1) Compile it with the script included

2) Run the code

```
>./Qlearn 1522 1 1 1 .25 0 20 1000000
```

Parameters are ***random seed, number of cats, number of cheeses, maze size, cat Smartness (in [0,1]), mode, number of rounds, and number of trials per round.***

Maze sizes:

1 → 8x8
2 → 16x16
3 → 32x32

Modes:

0 – Standard Q-Learning training	1 – Standard Q-Learning game
2 – Standard Q-Learning evaluation	3 – Feature-based Q-Learning training
4 – Feature-based Q-Learning game	5 – Feature-based Q-Learning evaluation

Develop and test your solution on the 8x8 grid, For the standard Q-learning, and on the ***16x16*** grid for feature-based Q-Learning.

For standard Q-learning, you always get 1 cat and 1 cheese (otherwise the state space is too large!). The number of rounds and number of trials determine the ***duration of the training phase only*** and have no effect on the game or evaluation modes.

Initially, of course, nothing much happens. You really have to get something going in the code before you can watch some mouse moves.

Step 3

Read the comments in ***QLearn.c***. This file contains headers and comments for all of the functions you have to implement. The basic implementation is fairly short, but you will need to spend some time fine-tuning your reward function and training mice.

A Mouse that Learns

Step 3 (cont).

You will have to figure out:

- How many trials to use in training
- How many rounds of training to perform
- How to determine from the stats being printed by the training process (or by looking at the mouse's behaviour after training) whether your training is working. If it is not, you have to figure out what to do about it.

Things to look for:

Once you have a working utility function and Q-learning update, the mouse should slowly get better over the training period. By the end of the training, your mouse should be winning noticeably more often than losing.

Note: The training phase saves the current ***Qtable*** on disk at the end of the training process, that is for standard Q-Learning. For feature based Q-Learning, the final set of weights is also saved to file.

Step 4

Once you have a fully working solution. ***Train the mouse on the 16x16 grid***. This will take considerably longer than the 8x8 grid, but should be interesting to see..

If you have done everything correctly, ***the mouse should be loosing a significantly smaller portion of the time*** after training is done even against a reasonably smart cat (smartness at .85 or more).

Answer all the questions and run all experiments requested in REPORT.TXT

Step 5 – Feature-based Q Learning

With standard Q Learning you can not train a mouse for larger grids than 16x16 in a practical Amount of time. It is also difficult to train for more than 1 cat and 1 cheese. And, your mouse will train for a specific environment (maze) and will have trouble working different mazes.

For this part, implement a feature-based Q Learning approach as discussed in lecture. This will allow your mouse to learn to handle mazes of size 16x16 and 32x32 as well as multiple cats and multiple cheese chunks.

A Mouse that Learns

Step 5 – Feature-based Q Learning (cont.)

Things you will need to do:

- Figure out a set of features to use. Your features should be such that:

    * They are general (i.e. do not depend on the size of the grid or the shape of the maze)
    * Are informative (they provide the mouse with useful information for deciding what to do)
    * Are properly ranged to be in [-1,1] or [0,1] depending on the feature
    * The number of features is fixed regardless of the number of cats/cheese, and the size of the grid.

What we are looking for here, is a set of features such that if we train the mouse in one particular maze, the mouse will still be able to play competently on a different maze, different grid size, and with different numbers of cats and cheese.

This is not so trivial. So think carefully about your features. ***The code allows you to use up to 25 features, and you should not forget to set the number of features in the QLearn.h file.***

You are allowed to do computation to estimate feature values. For example, you can find paths between agents, but notice that you ***can not*** return paths as features, or use the path to move the mouse. At the end of the day, whatever information is provided by your paths it must be reduced to a single real value used as a feature during the Q Learning process.

Testing, testing, testing!

Train your mouse on the 16x16 grid, with 2 cats and 3 or 4 cheeses. With feature-based learning you should not require huge numbers of trials to learn parameters, so test your features, see how the mouse behaves, and then think about how to improve your features.

Very importantly, you should consider ***what each feature is telling the mouse***. Since you have a weight associated with each feature, that tells you what the mouse thinks that feature is worth and whether it will try to move to locations where the feature value increases or decreases.

Once you have a working algorithm and a good set of features, complete the relevant parts of the REPORT.TXT

A Mouse that Learns

Step 6

Submit your work

Create a single compressed ***.zip*** file – and that means it should actually be ***in .zip format***. If you submit a compressed tar file, a .ar, .arj, .arc, .7z, or anything else renamed to have the .zip extension you will incur the wrath of your TA and lose marks.

The .zip file should contain everything in your working directory ***except the QTable data***.

** ***That is essential, otherwise you'll fill-up Quercus with QTables!*** **
`We will deduct 5 marks for submitting QTable.dat`

Your .zip file should be named:

`QLearnSolution_studentNo1_studentNo2.zip`

(e.g. QlearnSolution_012345678_9876543210.zip)

Submit your file on Quercus as usual

Before you submit, make sure your compressed file in fact contains all your files and code, and that it decompresses properly on mathlab. Non-working .zip files will get zero marks.

If you have an emergency and can't submit on Quercus, email me your code, but note I will impose a 5 to 10 marks penalty for waiting until the very last minute to submit your code.