# ME 493: HW 2

Shane Lawson

February 28, 2017

# 1 Introduction

In this homework, the gridworld that will be used in Project $\beta$ was implemented. A few objectives were to make sure the Agent cannot move itself off of the grid, and that the agent could navigate to the goal successfully, both by human controlled methods, and by a hard coded rule of thumb.

# 2 Specifics

## 2.1 Program

In addition to the main program file, I made four classes for this homework. An `Agent` class, a `Grid` class, a `Goal` class, and a `Position` class.

### 2.1.1 Position

The `Position` class is simple, containing two data members, an `x` coordinate and a `y` coordinate. It also contains a constructor, and declares the `Agent` class as a friend so that the agent can interact directly with the `x` and `y` data members without making them `public`.

### 2.1.2 Goal

The `Goal` class contains only a `position`, a constructor, and a function that returns the goal's position. This class is very basic since the Goal doesn't actually do anything except stay in the same spot and tell the agent where it is, so that the agent can navigate to it.

### 2.1.3 Grid

The `Grid` class is more complex than the previous two, but only slightly so. It contains the number of rows and columns, two functions that return these values, a constructor, and a couple other functions. For this homework this class was not fully implemented, as all that was required was the knowledge of the size of the grid. For Project $\beta$, this class will have an actual grid, or table, that will store the Q-values over time. One of the other functions

in this class prompts the user for the desired size of the grid and is used when a `Grid` is constructed. The last function in this class takes in two positions, in the form of four integers, x and y of the agent, and x and y of the goal. These values combined with the size of the grid results in the function outputting a representation of the grid and locations of the agent and goal, respectively, so that the user can see immediate changes when moving the agent around the gridworld.

### 2.1.4 Agent

The `Agent` class is where the majority of the functionality of the program is located. It is where all of the decision and corresponding action occurs. In this implementation, an `Agent` has a `Position`, `Grid`, and `Goal`. It is imperative that the agent have a position when moving around the grid so that it can not move outside of the grid. What makes less sense in this case is giving the agent a `Grid`. In this case, the grid does not provide much, but in Project $\beta$ this will prove very useful, since essentially the agent is given a map of the world, so it can navigate around to the best squares on the grid, all the while updating the Q-values for each square. The `Goal` provided to the agent here is just so the agent can ask where it is to find the ideal path, something which it will have to learn in Project $\beta$ without being directly given this information. This class has a constructor, two overloaded `move()` functions, and `testA()`.

- `move(char)` : This function moves the agent one block in a specified direction (W, A, S, D - up, left, down, right), so long as that movement does not take the agent outside of the grid.
- `move()` : This function executes the agent's rule of thumb method for locating the goal, which identifies what move will take the agent closer to the goal, then executes that move by calling the `move(char)` function.
- `testA()` : This function fulfills the *TestA* functionality required by the rubric. It prompts the user for a location to place the agent off of the grid, it then uses the modulo operator with the number of rows and columns in the grid to place the agent back onto the grid. Using the modulo operator rather than putting the agent on the edge of the grid, drops the agent somewhere within the grid. This means that if some rule has been overlooked where the agent can move past the wall and outside of the grid, it's not put in a position to continually make this move again and again, making the learning fail overall, but given enough opportunity, by getting placed in the middle of the grid, the agent should be able to find the goal.

### 2.1.5 Main

The main program consists of a function which prompts the user for which method of control to use: manual, auto, or just running `testA`. Based on the input, the program will select `testA` (described in Section 2.1.4), `testB` or `testC` (functions in main).

- `testB()` : This functions fulfills the required functionality specified in the rubric, namely, the ability to manually guide the agent to the goal. Based on a W, A, S, or

D input provided by the user, the `Agent.move(char)` function is called with the desired input, and the movement is executed by the agent and the display is refreshed to show the changes, and repeating until the goal is found successfully. Unfortunately, I could not find a simple, cross platform way to allow the user to direct the agent just by pressing a single direction key, so the user must enter the command and then press enter. A cool side effect is because of the use fo `cin` to a `char`, the user can input a string of commands and run them all simultaneously. For example moving the agent down 6 rows by inputting "ssssss", or appearing to move diagonally by inputting "wd", then pressing enter.

- `testC()` : This function fulfills the required functionality specified in the rubric, namely, the ability for the agent to autonomously navigate to the goal used a coded rule of thumb. This function calls the `Agent.move()` function repeatedly, moving one step closer to the goal each time, until the agent reaches the goal. The `Agent.move()` function is described in Section 2.1.4. I used the `<thread>` and `<chrono>` libraries to sleep the thread for a half a second in between each automated step so that the user can watch the agent move along to the goal, rather than happening in a single flash that happens too quick to identify what has occurred.

## 2.2 Conclusion

In conclusion, I completed all the tasks that were necessary (specified in the rubric). These include, for TestA, ensuring the agent cannot leave the grid, and if it somehow does, that it will be bumped back onto the grid, for TestB, that the user can manually control the agent and direct it to the goal, and for TestC, that the agent can navigate to the goal using a programmed rule of thumb. In addition, I made the program easy to use with numerous prompts for user input which allow the user to control exactly what they want to without having to modify code, as well as providing graphical output of the grid, and positions of the agent and goal at each step.