# CS 440: Maze Search

## Environment

The environment that we set up is quite simple. We created **Maze** and **Location** objects for the search methods to take in. The bulk of the work is done in the **Maze** class. A **Maze** is formed by reading in each character row by row in a text file. Each **Location** is assigned a value based on the character read. An empty space is a space, a “%” is a wall, a “P” is the start position, and a “.” is the goal position. Each **Location** in the **Maze** is then added to a two-dimensional array of Locations based on where it is located in the text file. As the maze is being built, the start and goal positions are additionally stored in separate **Location** variables.

A **Location** object stores each position using two integers, one that corresponds to the x-coordinate, and another that corresponds to the y-coordinate. Additionally, it contains a classifier that details the object type, and a heuristic for the A\* search. The **Location** class also has a *getAdjacent* method that returns an ArrayList of **Location**s that neighbor the **Location** called by the method. Before a **Location** is added to the ArrayList, the *getAdjacent* also checks to see if it is valid (within the bounds of the maze).

Both the **Maze** and **Location** classes also have a *toString* method which is used to print the solution. **Location**’s *toString* returns each position’s character as a string, and **Maze**’s *toString* returns a string of all the *toString*’s of the **Location**’s in the array.

## Basic Pathfinding (1.1)

### Depth-First Search

### Breadth-First Search

### Greedy Best-First Search

### A\* (Astar) Search

## Penalizing Turns (1.2)

## Pacman with a GHOST (1.3)