# CS 4150/5150: Game Artificial Intelligence

# Assignment #2

# Movement, Part 1

# Due: 6:00pm, 2/3/20

## Assignment:

Build a path-planner that can operate on a grid.

Your path planner should be implemented in C++, in Visual Studio 2019 (you can download the community version for free from here: <https://visualstudio.microsoft.com/downloads/>).

Your program should take a filename and the start and destination coordinates as inputs, and should print to the screen the solution with the path marked on it, as well as the elapsed time it took to plan the path. It should also have the option of displaying all expanded nodes (i.e. all nodes in the open list) and all touched nodes (all nodes in the open or closed list), but should not do so by default. You will likely find those modes to be very useful for debugging.

## For Mac Users:

1. CLion: <https://www.jetbrains.com/clion/>

Steps to start:

1. If you already have a sample code with **CMakeLists.txt** file, then click **File|Open** and point **CLion** to the top-**level CMakeLists.txt** file, then choose **Open as Project**.
2. You can also use **MakeFiles** and open them as projects.
3. Creating a New Project:
   1. Select **File | New Project** from the main menu or click **New Project** on the welcome screen.
   2. Set the type of your project: C or C++, an executable or a library.
   3. Provide the **root folder** location and select the **language standard**.
   4. CLion creates a new **CMake** project and fills in the top-level **CMakeLists.txt**.
4. Open the existing code base:
   1. Select **File | Open** from the main menu and select the project folder.
   2. Create a **CMakeLists.txt** with basic instructions to run the project. Check this: <https://cmake.org/cmake/help/latest/guide/tutorial/A%20Basic%20Starting%20Point.html>
   3. The IDE will compile the project, and you can run it as shown in the images below.

To run the project, click on the button show as below:

A screenshot of a computer

Description automatically generated with medium confidence

## Input:

The grid will be specified as text file that looks something like this:

11X11  
11X11  
1XX11  
11111

each character represents a single tile as follows.

* ‘X’ – impassible tile
* [1..9] – passible tile. The cost of traversing the tile is given by the integer value (1 is easiest, 9 is hardest).

In addition, your program should accept start and destination coordinates as an x and y value. (0,0) is the bottom left grid tile, so in this case if you wanted to go from the top left tile to the top right tile the coordinates would be (0,3) and (4,3).

Finally, you should provide options to set the output mode. Output mode can be:

* Standard: just display the path
* Expanded: display the path and all expanded nodes
* All: display the path, all expanded nodes, and all other touched nodes

## Output:

Your path planner should output the solution to the screen, along with the elapsed time. The solution should look like the input, but with ‘s’ for the start node, ‘d’ for the destination node, ‘+’ for other nodes along the path, ‘e’ for expanded nodes (if the mode is “Expanded” or “All”), and ‘t’ for all other touched nodes (if the mode is “All”). So for example, if the mode was “All” and the map and coordinates are the ones given as examples above, your output would appear something like this:

seXtd  
+eXt+  
+XX+t  
t++tt  
Elapsed Time: 0.00187455 seconds

## Testing Your Code:

Project1.txt contains a sample map, which I created using Notepad. I strongly recommend creating maps of your own that test as many different edge cases as you can think of, including large and difficult maps.

## Deliverables:

Your deliverables should include a .zip file with:

1. A VS 2019 solution and project
2. All source code
3. Instructions for running your code
4. An executable version that I can run on a Windows 10 machine.
5. A self-evaluation describing which features you believe that you did and didn’t get working

## Grading:

Points will be assigned as follows:

* 60 points: Find a valid path from start to destination
* 10 points: Find the shortest path
* 10 points: Support “Expanded” and “All” output modes
* 5 points: Handles the case when there is no solution
* 5 points: Handle large, complex maps (with reasonable efficiency)
* 5 points: Support diagonal movement. Be careful – sometimes it is valid to move between two passable tiles on the diagonal, and sometimes it is not.
* 5 points: Handle tiles with a cost of 0
* 5 points: Allow the user to choose between straight-line distance and Manhattan distance for their heuristic
* 5 points: Path smoothing. In this case tiles should be treated as 1m x 1m squares, and coordinates should be given using floats rather than ints (so the bottom left corner of the bottom left tile would be (0.0, 0.0) and the top right corner of that tile would be (1.0, 1.0)). The output should give a minimal set of waypoints to define the path, and each waypoint should specify the exact corner of the tile that the path should traverse.

Note that it is possible to receive up to 10 points of extra credit for this assignment.

## Resources:

* Path planning introduction: <https://www.redblobgames.com/pathfinding/a-star/introduction.html>
* Further path planning articles: <http://theory.stanford.edu/~amitp/GameProgramming/>