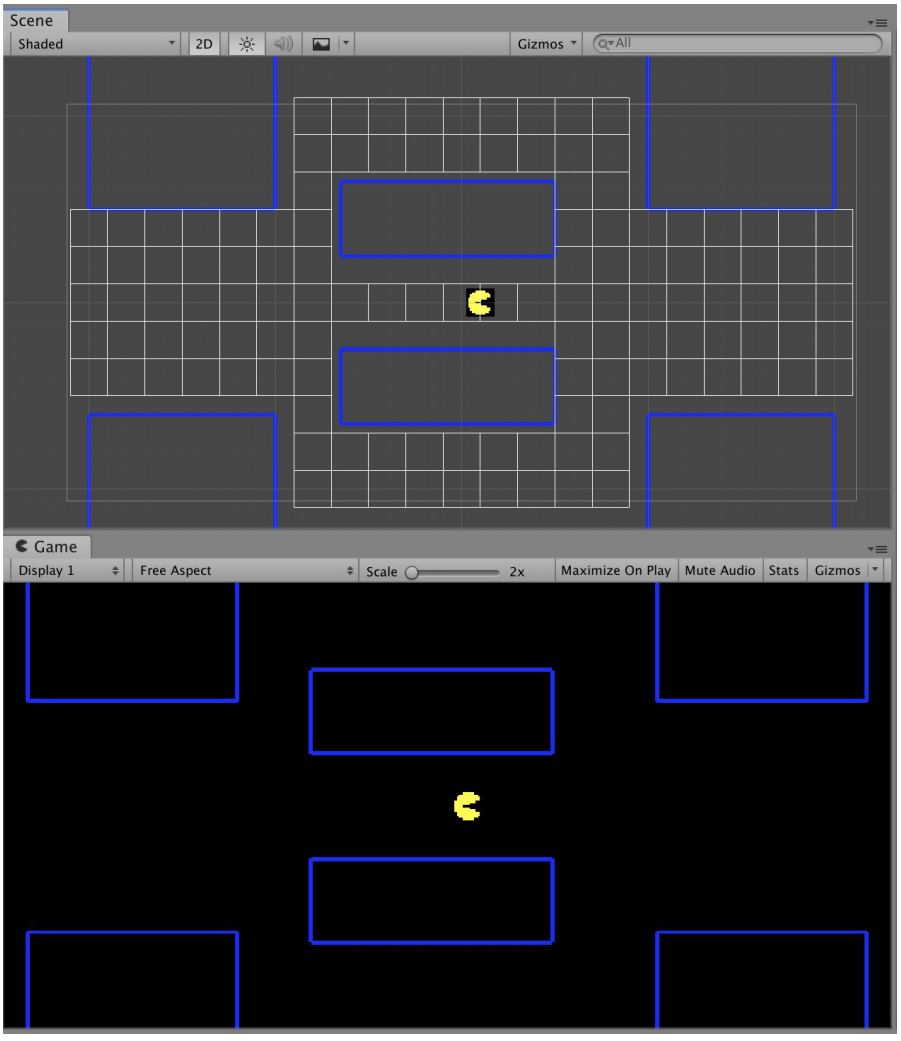
Assignment 1: Grid navigation

The purpose of this exercise is to acquaint you with the Unity game engine we will be using in the class.

One of the main uses of artificial intelligence in games is to perform path planning, the search for a sequence of movements through the virtual environment that gets an agent from one location to another without running into any obstacles. For now we will assume static obstacles. In order for an agent to engage in path planning, there must be some representation of the space for the agent to traverse. The simplest representation is a grid. Think of an imaginary lattice of cells superimposed over an environment such that an agent can be in one cell at a time. Moving in a grid is relatively straightforward: from any cell, an agent can traverse to any of its four (or eight) neighboring cells.



In this assignment, you will write the code to superimpose a grid over any given terrain so that an agent can navigate by moving left, right, up, or down from cell to cell. The code to

generate the grid should work on any terrain such that an agent can never collide with an obstacle.

But first, you need to become familiar with the Unity game engine in which you will be working.

What you need to know

Unity is script-based, meaning that there are small sets of programs and logic organized into individual scripts. For this assignment you will be modifying **GridHandler.cs**, with an optional extra credit involving modifying **NavMeshHandler.cs**. In this assignment you will primarily interact with **ObstacleHandler.cs**, which holds all obstacles in the environment and **Polygon.cs**, which represents a simple polygon. The obstacles are polygons, but you might find **Polygon.cs** broadly helpful in this assignment, as all it does is represent any polygon.

Below are the important bits of information about scripts that you will be working with or need to know about for this assignment.

ObstacleHandler.cs

You can interact with the ObstacleHandler via calling ObstacleHandler.Instance.X where X is a name of a function or public variable.

Member public variables:

- Obstacles: a public array of Polygons that represents all the obstacles

Member functions:

- AnyIntersect(Vector2 pt1, Vector2 pt2): checks to see whether any of the obstacles intersect the line made by these two points.
- AnyIntersect(Vector3 pt1, Vector3 pt2): same as above, but only checks the x and y values of these vectors
- PointInObstacles(Vector2 pnt): returns true if pnt is in any obstacle
- GetMapCorners(): returns the corners of the map as an array of Vector2s
- GetObstaclePoints(): returns an array of Vector2s representing all of the obstacle points

Polygon.cs

All Polygon objects will have the following public variables and functions.

Member public variables:

- Points: a public array of Vector2s that represents all the points of this polygon

Member functions:

- Polygon, Polygon(Vector2[] _points), and Polygon(Vector2[] _points, Material _lineMaterial): Different constructors for Polygon, giving different information.
- GetLines(): returns a 2D array of Vector2 points, where each index stores an array representing a pair of Vector2 points that compose a line.
- HasPoint(Vector2 pnt): returns true if the given pnt is one of the polygon's points.
- HasLine(Vector2[] line): returns true if the given line is one of the Polygon's lines.
- AnyIntersect(Vector2 pt1, Vector2 pt2): returns true if the polygon intersects with the line defined by these two points.
- AnyIntersectWithoutSharedPoint(Vector2 pt1, Vector2 pt2): same as above, but makes sure that the two given points are not the point of intersection.
- ContainsPoint(Vector2 pnt): returns true if this point (pnt) is in the polygon.

There are other public functions in Polygon, but they will only come up if you attempt the extra credit.

Instructions

You must superimpose a grid over an arbitrary, given game world terrain consisting of obstacles. The grid is a dictionary stored in GridHandler.cs that contains all of the locations that the agent is allowed to be.

When you click on the screen, you indicate where you want the Agent to traverse. Without a grid, the agent will take a direct line to the point. With a grid (or some other graph) the agent will follow the given spatial representation.

Note: Do NOT make use of any outside libraries or scripts beyond what has been provided. Only make changes to GridHandler.cs and optionally NavMeshHandler.cs

Step 1: Download and install Unity https://unity3d.com/get-unity/download

Step 2: Download the Assignment1.zip file from eclass, unzip it, and open it via Unity. You can open it by either selecting the unzipped folder from Unity or double clicking Scene1-5.

Step 3: Open Scene1-5 located inside Assets/Hw1. Hit the play button. Click inside the game window to see the pacman agent traverse the game world. Note: Pacman will get stuck if you get too close to an obstacle.

Step 4: Modify GridHandler.cs to complete the CreateNodes() function (see the commented lines of where to edit).

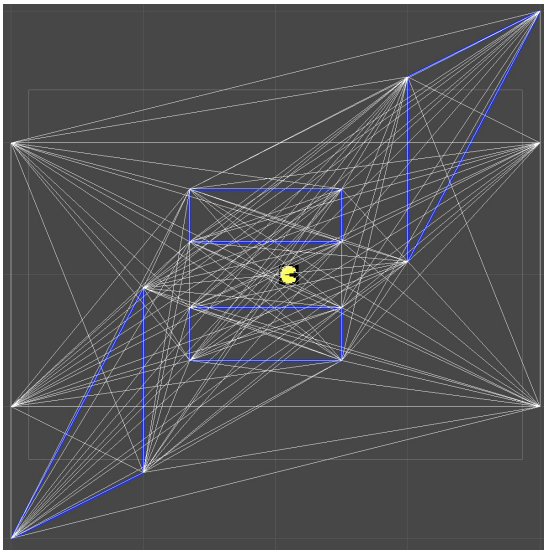
Inside CreateNodes() you can reference the ObstacleHandler object via ObstacleHandler.Instance.X where X is a function or public variable.

Step 5: Test your implementation across the five given scenes (Scene1-5)

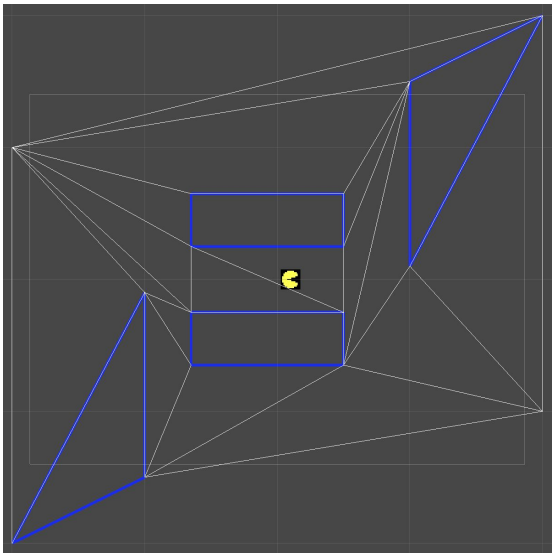
Additional testing can be done by changing the maps by modifying the ObstacleTestPoints scripts.

Step 6 (Optional): Implement a nav mesh generation in NavMeshHandler.cs (see the commented lines of where to edit). You can test your nav mesh in the given “Extra Credit Scene”.

Essentially you should go from this (the default behavior):



To something like this:



Grading

This homework assignment is worth 10 points. Your solution will be graded by an autograder. The autograder will look for grid cells that intersect with obstacles or go beyond the boundaries of the map. For every map that your solution is tested against, 1 point will be deducted per map if at least one cell intersects with an obstacle or goes outside the boundaries of the map. The autograder will test your solution on 10 maps (one point per map), the five provided maps and five withheld test maps.

You may optionally get up to 1 point of extra credit by completing NavMeshHandler.cs. You will gain 0.5 points if you can successfully create a valid Nav Mesh (no meshes intersect with one another or any obstacles). You will gain another 0.5 points if your mesh contains at least one mesh with four or more points. Your NavMeshHandler.cs script will be tested on one of the withheld test maps. Even failing all of this, if your NavMeshHandler.cs performs better than the given default behavior you will receive 0.1 points for attempting the extra credit.

Hints

Debugging within the game engine can be hard. Print statements (Debug.Log(“”) will be one possible way of figuring out what is going on.

In general, you should be able to get most of what you need from ObstacleHandler, and only need to use Polygon’s functions sparingly for the main assignment.

It is good to test your techniques on new maps. If you want to make new maps, modify one of the provided ObstacleTestPoints (e.g. ObstacleTest1Points.cs). This will alter the obstacles in the same numbered scene.

Unity makes use of Vector2 and Vector3 objects, representing points in two and three dimensions respectively (e.g. (0,0) and (0,0,0)). While we work in 2D (and polygons use only 2D points), the game engine represents objects in 3D. This can lead to some confusion, but essentially a Vector3 will be the same as a Vector2 for our purposes since the third “z” dimension will always have a value of 0. Vector2 and Vector3 objects are complex, meaning that unless two Vector2 or Vector3 values are the same (even if the position values are the same) they will not be considered equivalent.

We will make use of three kinds of more advanced data structures in this assignment, Arrays, Lists, and Dictionaries. If you’re unfamiliar with lists and dictionaries or how they work in C#, you might find this tutorial helpful:
https://learn.unity.com/tutorial/lists-and-dictionaries

In case the above links are broken, you can search for the highlighted terms in the Unity Scripting API: https://docs.unity3d.com/ScriptReference/

Submission

To submit your solution, upload your modified GridHandler.cs and optionally NavMeshHandler.cs (if you attempted the extra credit). All work should be done in these files, **do not modify any other file**.

Cheating will be checked for, please do not copy code from anyone.

You should not modify any other files in the game engine.

DO NOT upload the entire game engine.