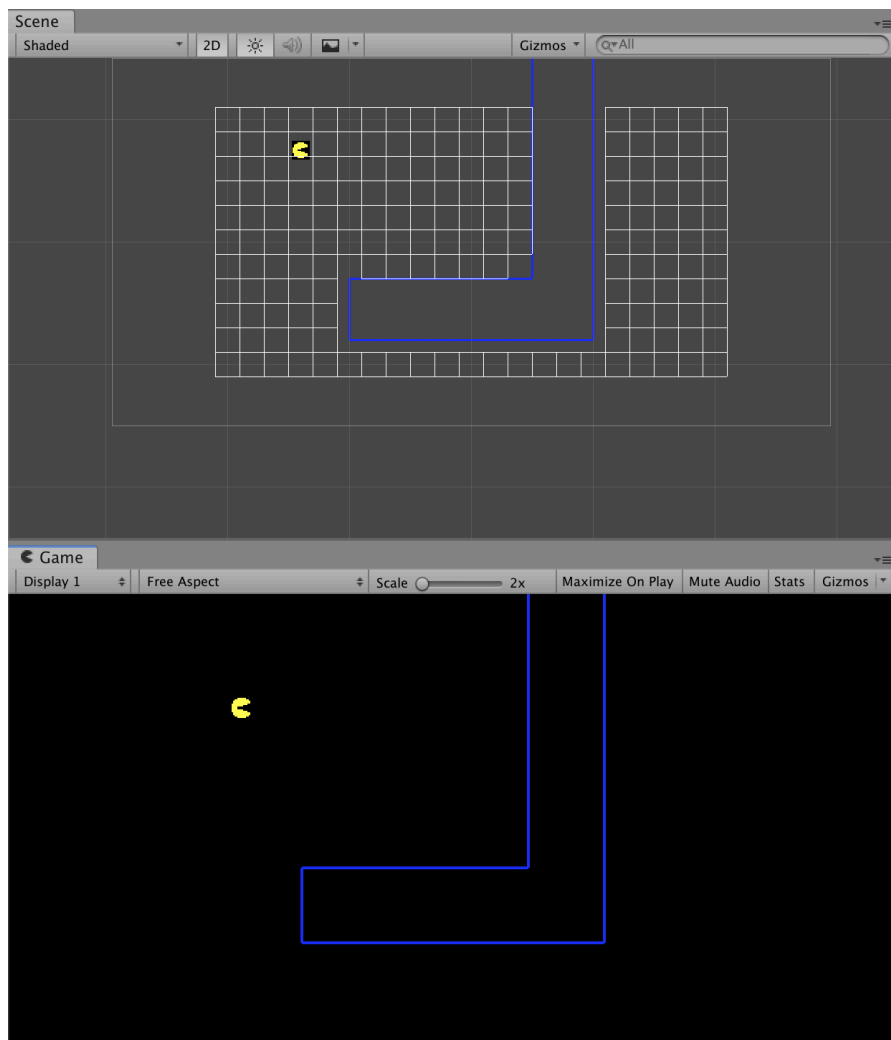


Assignment 2: A*

The purpose of this assignment is to have you implement the A* pathfinding algorithm, the most common pathfinding algorithm in games.

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 will get an agent from one location to another without running into any obstacles. Previously the Pacman agent didn't plan well, and simply greedily attempted to get to a goal (and tried to straight shot if it couldn't find a path). In order to make the agent move more intelligently, you'll need to implement a more advanced pathfinding algorithm.

However, the implementation you've been given has a few quirks. You'll have to demonstrate you understand A* well to complete it successfully.



In this assignment, you will implement A* pathfinding on either a provided grid or on your own generated grid.

What you need to know

The primary script you will need to interact with for this assignment is **AStarPathFinder.cs**. This script makes heavy use of **AStarNode.cs** and **PriorityQueue.cs**. In addition you might find it helpful to look over **GraphNode.cs**.

This assignment also includes optional extra credit. If you pursue this extra credit it might be worth looking back over the information from the last assignment.

GraphNode

Member public variables:

- Location: a Vector3 representing this node's location
- Neighbors: An array of GraphNodes referencing the neighbours of this node (in this case, the grid cells to the north, east, south, and west.).

Member functions:

- GraphNode(Vector3 _location): this function instantiates a new GraphNode
- AddNeighbor(): Adds a new neighbour to the internal list of neighbours kept by the GraphNode (you will not need this).
- GetHashCode(): returns a unique hash for this GraphNode

AStarNode

Member public variables:

- Parent: Returns the AStarNode parent of this node.
- GraphNode: Internal representation of a GraphNode for this AStarNode, stores location and neighbour data.
- Location: Vector3, allows for reference to the location of the GraphNode more easily.
- GScore: The cost of the path up to this AStarNode.
- HScore: The heuristic cost to the goal from this AStarNode.
- FScore: The combined G and H score.

Member functions:

- AStarNode(AStarNode _parent, GraphNode _graphNode, float _hScore): constructor for a new AStarNode.
- GetFScore(): Alternative way to access the FScore.
- CompareTo(AStarNode other): compares two AStarNodes in terms of their FScores.
- Equals(object obj): checks to see if two AStarNodes are equal.
- GetHashCode(): returns a unique hash for this AStarNode.

PriorityQueue.cs

Member public variables:

- Data: a read-only (you cannot make changes to it) List<T> that reflects PriorityQueue's internal list.

Member functions:

- Enqueue(T item): Adds an item to the priority queue.
 - Dequeue(): Equivalent to “pop”, removes and returns the minimum cost item from the queue.
 - Peek(): Returns the minimum cost item (front of the queue), but does not remove it.
 - Count(): Returns the size of the queue as an integer.
 - Remove(T item): Removes the specified item from the queue.
 - ToString(): Returns a string representation of the queue (unlikely to be useful)
 - IsConsistent(): Checks whether the queue is consistent (unlikely to be useful)
-

Instructions

When you click on the screen, you indicate the goal for the path planning algorithm. By correctly implementing A*, your agent will take the most optimal path, if one exists.

Step 1: Download the Assignment2.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 Scene 1-5.

Step 2: Open Scene1-5 located inside Assets/Hw2. Hit the play button. Click inside the game window to see the pacman agent traverse the game world. By default it will only path correctly over very small distances. You'll notice the number of nodes it has to check is also very high.

Step 3 (conditional): If you wish you can replace the GridHandler.cs file in Assets/HW2 with your GridHandler.cs file from assignment 1. Alternatively, if you are using a version of Unity from before 2022, you'll need to download the “GridHandler pre 2022.cs” file from Canvas, rename is GridHandler.cs and use it to replace the GridHandler.cs file in Assets/HW2.

Step 4: Modify AStarPathFinder.cs to complete the CalculatePath() function (see the commented lines to determine where to edit). You can add extra functions to this script but please don't change other functions (except Heuristic if you do the extra credit, and even then **you can't change its arguments**).

Step 4b (optional, extra credit): Modify AStarPathFinder.cs to complete the Heuristic function (see the commented lines of where to edit, **you cannot change its arguments**).

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, though you'll have to calculate the optimal number of nodes yourself.

Grading

This homework assignment is worth 10 points. Your solution will be graded by an autograder. The autograder will attempt to have Pacman navigate from various points across each map. For every map that your solution is tested on, some proportion of 1 point (depending on how over you are) will be deducted if your implementation expands more nodes than is necessary (if you did not implement A* correctly). The autograder will test your solution on 10 maps, the five provided maps and five withheld test maps.

If your code does not compile you will get a 0. If you import (using "using") any additional libraries you will get a 0. Broadly, if your code does not compile you'll get a 0.

Note: You must make use of the provided PriorityQueue if you wish to receive full marks on this assignment (it is intentionally bad, that's part of the challenge of the assignment). Submissions that don't make use of the PriorityQueue will receive a 0.

For the given five scenes/maps, going from the top left to the bottom right corner should lead to the following (or fewer) nodes being created with the default Heuristic:

Scene 1: 140

Scene 2: 175

Scene 3: 175

Scene 4: 115 (the more central/vertical corners, inside the corner obstacles)

Scene 5: 220

Note: These numbers assume the default GridHandler.cs file.

Note 2: Getting below these values **DOES NOT MEAN** you have a perfect A* implementation. If you want to confirm that you can calculate how many nodes would be opened for other pairs of points.

In addition, you may receive up to 1 point of extra credit for implementing a better heuristic than the one currently implemented in Heuristic (the Manhattan Distance). The best student implementation (in terms of fewest nodes explored with the instructor A* implementation) and any implementations with 10% of the best will receive 1 extra credit point. All other implementations will receive a fractional percentage based upon their relative performance, but at minimum students will receive 0.1 extra credit points if they do better than the provided heuristic. **There are multiple heuristics that open the minimum nodes for all test cases.**

Note: You cannot change the input arguments or the returned value type of Heuristic. Further, do not change CalculatePath such that it relies on your specific Heuristic function, it should be able to be run (without crashing) with the default Heuristic function. Otherwise your code will count as not compiling **AND YOU WILL RECEIVE A 0.**

Hints

Remove takes as input an AStarNode, which must **exactly** match the node you wish to remove. However, you don't want to instantiate additional AStarNodes for this assignment. Thus it might be helpful to keep track of your AStarNodes in an additional data structure.

AStarNode uses a unit distance of 1.0 but the grids in this assignment have **a size of 0.2f**. You'll need to account for this if you want full marks.

You will need to make your own data structures for this assignment so overviewing/reviewing Lists and Dictionaries might be a good idea, if you didn't for Assignment 1. Particularly the "Find" function will likely be useful.

Debugging within the game engine can be hard. Print statements (Debug.Log("")) will be one possible way of debugging and understanding your implementation.

For the extra credit Heuristic you might find it helpful to reference ObstacleHandler with ObstacleHandler.Instance.X where X is a function or public variable. Further, we typically assume that a grid cell costs 1 unit (in terms of g and h score) to move across. However, in this project that's not true, as the gridSize has a different value.

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. Note that you will need to use your own GridHandler.cs file to use this option. Further note that you will need to calculate the minimum number of opened nodes for your new maps yourself.

Submission

To submit your solution, upload your modified `AStarPathFinder.cs`. All work should be done within this file. You should not modify any other files in the game engine. DO NOT upload the entire game engine.