Dept. of Computer Science & Software Eng., Concordia University
COMP 476 --- Winter 2021

Advanced Game Development

Assignment 2 --- Due March 13, 2021

PURPOSE: This assignment will give you the opportunity to learn more about programming AI behavior. This assignment contains both **theoretical** questions, which you do not need to implement, and a **practical** question, which requires you to write a C#/Unity program.

SUBMISSION: The assignments must be done individually. On-line submission is required (see details at end) – a hard copy will not be read or evaluated.

PREPARATION: Parts of this assignment require knowledge of basic concepts from parts of the material covered in the course slides (the course schedule lists which exact sections of which books the material is based on).

Question #1: (10%) [Theoretical Question]

We can represent a weighted directed graph like as a set of nodes and edges (with assigned weights indicated):

nodes = {S, A, B, C, D, E, F, G}
edges = {SA: 2, SB: 9, AB: 4, AD: 7, AC: 6, BC: 4, BF: 12, DE: 5,
CD: 7, CE: 5, CF: 5, EF: 2, EG: 2, FG: 5}.

In the graph, assume that *S* is the start node and *G* is the goal node.

a) (5%) If we use Dijkstra's algorithm to find the minimum cost path from *S* to *G*, then the following table shows the contents of the open and closed lists for the first 2 steps of the algorithm. Fill in the remaining lines. Each entry in the lists is of the following format (Node, Cost-So-Far, Connection). You must only stop when the *guaranteed* shortest path has been found.

Current Node	Open list	Closed list
–	(S,0,–)	
S	(A,2,SA), (B,9,SB)	(S,0,–)

b) (5%) For a node *A*, we can specify that the value of the heuristic function at the node is 5 by writing *A* : 5. Update the graph from part a) with following heuristic values:

nodes = {S : 10, A : 12, B : 6, C : 10, D : 5, E : 7, F: 6, G : 0}

If the A* algorithm is applied to the same graph as in question 2, the following table shows the contents of the open and closed lists for the first 2 steps. Fill in the remaining lines. Each entry in the open list is now of the following format: (Node, Cost-So-Far, Connection, Estimated-Total-Cost). Entries for the closed list are same as before. Stop when the *guaranteed* shortest path has been found.

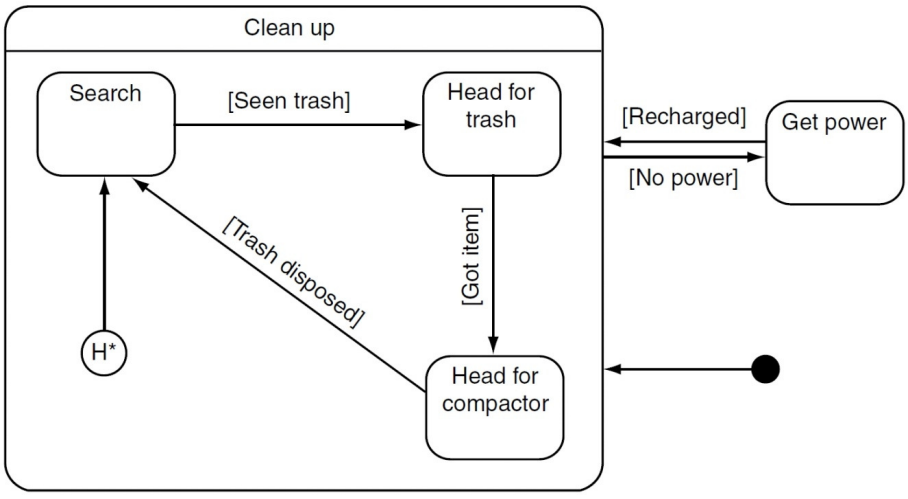
Current Node	Open list	Closed list
–	(S,0,–,11)	
S	(A,2,SA,14), (B,9,SB,15)	(S,0,–)

Question #2: (10%) [Theoretical Question]

Consider the hierarchical state machine for a trash collecting robot shown in Figure 5.16 of Artificial Intelligence for Games 2nd Edition, by M&F, reproduced below.

a) Change the behaviour represented such that rather than just disposing of any trash in the compactor, the robot will first check if the trash is recyclable or not. If it is, it puts the trash in the recycling bin; otherwise, the robot will dispose of the trash in the compactor.

b) In addition to the changes in a), also extend the hierarchical state machine to add the behaviour that if fighting breaks out at any time that the robot has to take cover.



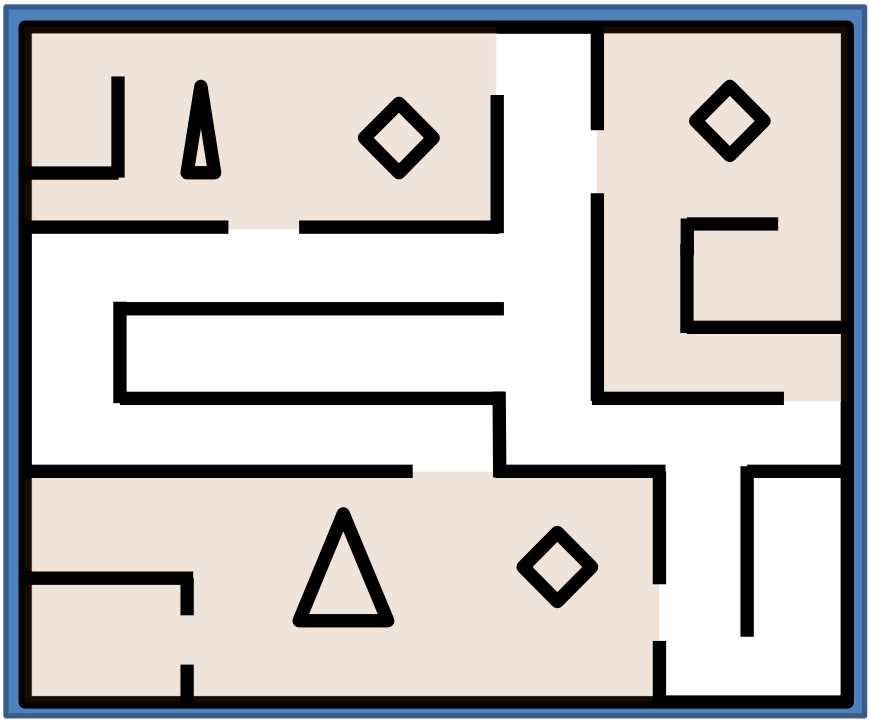
Question #3: (80%) [Practical Question]

Problem Statement:

For the question you are to write a Unity program in C# to have an NPC perform path following. Your program must comprise of the following:

R1) Level and Pathfinding Graphs

- Using Unity as your game engine, define an environment in the form of a 2½D level which is closed (the movement is only within the geometry defined below). Make sure the (stationary) view point can see the entire level (i.e., don't make the walls of the level too high). Fill the level with the following geometry (*must* have all these, at least):
  - Three completely disjoint large rooms (disjoint ➔ they are not touching, not even sharing common walls or doorways). Place these three rooms at opposite corners of the level and have their interiors together occupy about one-third to one-half of the area of the level.
  - Off of these three large rooms, each connected by an open door, place small rooms (like walk-in closets or slightly larger) that have no other exits;
  - For each of the large rooms, place two open doorways to two different long corridors that connect to the other rooms. Each of the three long corridors must have at least one right-angle turn, may split and/or merge, and may share common junctions with other corridors.
  - Off of one or two of these long corridors place a "dead-end" corridor for each corridor.
  - Within each large room place one or two relatively large static convex polygonal (not round) obstacles, not so close to a wall that the NPC can't move around it.
- Below is an example level showing the minimal requirements.



- You are to create **two** different pathfinding graphs, one using a regular grid, and the other using points of visibility. Neither should use Unity's Navmesh in any way.
  - For the regular grid (Tile) graph, follow the description giving in the class slides. Place a node representing each tile in the centre of the tile, which is then connected by directed edges to its (up to eight) neighboring tiles. Choose the size of the tile to yield at least 200 nodes over the level.
  - For the points of visibility (PoV) graph, place nodes at critical shortest path positions (keeping in mind the size of your NPC): center of doorways, at the corners of door frames, around objects in rooms, at the corners of rooms, at the corners of corridors, *etc.* Also place visibility points in the centers of voids, middle of corridors, *etc.*, so that you end up with at least 30 nodes. Connect each visibility point with directed edges to all the other visibility points it can "see" (such that a straight edge between the nodes does not intersect a wall/obstacle; you can do this by visual inspection).
- For both types of the pathfinding graphs, make sure there are enough nodes such that there are a variety of paths through the rooms, and around the level. Render these nodes (simply as points or small discs) on the floor of the level. The weights of the edges of both types of pathfinding graphs will be the Euclidean (shortest) distance between the endpoints.

R2) A* Algorithm:

- Implement the A* algorithm on your pathfinding graph (the lab material will cover most of this). First will be Dijkstra with a null heuristic (Just Dijkstra). The second will use the Euclidean Distance heuristic. The third will use the Cluster heuristic.
- For Cluster heuristic consider the nodes in each room to contain a cluster, and each corridor to contain a cluster. Using Dijkstra's algorithm, between each pair of clusters compute the shortest distance between any two nodes (one from each cluster). Use these results between pairs of clusters to create a lookup table.
- Demonstrate the "fill" for A* with the Euclidean Distance heuristic and that for A* with the Cluster heuristic as follows. For all four combinations of both algorithms and both Tile and PoV pathfinding graphs (in turn), pick roughly the same start node and goal node that are "far" from each other (in terms of path length) in your level. Compute the shortest path and then, on the level, indicate graphically which nodes are included in fill for that algorithm (i.e., all nodes that were ever placed in the open list). In addition, render the path computed by highlighting, graphically, the edges between consecutive nodes on the path.
- **Submit screenshots of all four "fills" in your write-up as part of your question solution.**

R3) NPC Behaviour:

- If you haven't already, add the following *steering* behaviors to your collection from Assignment 1:
  1. *Path Following* – Implement this to delegate to Arrive (since we want the character to stop at the Goal)
  2. *Looking Where You're Going* (delegating to Align)

- Your character should perform the following movement behaviour. Loop over the following, alternating between using the Tile graph and the PoV graph:
  1. You can place your character initially randomly at any position in any of the small rooms adjacent to one of the three large rooms. Determine what node of the pathfinding graph is closest to this starting position. This will be the pathfinding Start node.
  2. Similarly, at random, select a target position in one of the small rooms adjacent to one of the *other* large rooms. Again, determine which node of the pathfinding graph is closest to this target position. This will be the pathfinding Goal node.
  3. Compute a shortest path using A* with Cluster heuristic from Start to the Goal Node.
  4. Move the character along the following path using the Path Following and Looking Where You're Going steering behaviours: initial position to Goal node, shortest path to the Goal Node, then to the target position.
- Aim for "realistic" movement and the character must not pass-through walls or obstacles while moving. For instance, you may want to smooth out the path computed using the Tile graph or adjust the position of the points of visibility for PoV graph. **No moving NPC can have a Rigidbody component attached.** Do not worry about collision detection, but if absolutely needed, it can be implemented by checking isTrigger on the collider.
- As in Assignment 1, you do not need a fully animated character as long as you can perform affine transformations (translate and rotate) on character and can tell which way it is facing. That will be minimally sufficient for this assignment as well.

EVALUATION CRITERIA For Question 3 of this assignment

1. Only working programs will get credit. The marker must be able to run your program if it works to evaluate it, so you must give in your write-up any instructions necessary to get your program running. If your program does not run, we will not debug it.
2. Breakdown:
   - R1: 40% (equally divided among the requirements listed in item R1 above)
   - R2: 15% (equally divided among the requirements listed in item R2 above)
   - R3: 25% (equally divided among the requirements listed in item R3 above)
   - Appropriateness of level design and general aesthetics : 10%
   - Source program structure and readability: 5%
   - Write-up: 5%

What to hand in for Assignment

Questions 1-2: (20%) (Theoretical Questions)
- Submit answers to questions 1 and 2 **electronically** (as a PDF file, for example, **A2TheoryYourID.pdf)**: Submit this file on Moodle under "Theory Assignment 2".

Question 3 (80%) (Practical Question)
Submission Deliverables (Only Electronic submissions accepted):

1. Submit a well-commented C#/Unity source program including data files, if any, along auxiliary files needed to quickly get your program running, and any other instructions for compiling/ building/running your program. The source codes (and brief write-up, explained below) must be submitted **electronically** in a **zip format** with all the required files (example: **A2YourID.zip**): on Moodle under the "Programming Assignment 2" link. ***Please do not e-mail the submission.***
2. Demonstrate your working program from an **exe file dated on or before March 13 at 23h55** to the lab instructor in the lab during the period of March 12-17.
3. A brief write-up about your program, explaining any special features and/or implementation details that you would like us to consider during the evaluation.