Dept. of Computer Science & Software Eng., Concordia University
COMP 476/6331 --- Winter 2022

Advanced Game Development

Assignment 2 --- Due Sunday, March 6, 2022

This assignment will give you the opportunity to learn more about programming AI behavior. This assignment contains both **theoretical** questions, which you don't need to implement (doesn't mean you have to do them by hand), and a **practical** question, which requires you to write a 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 (exact sections of which books the material is based on are on the course schedule).

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, G}
edges = { SA: 3, SB: 10, AB: 5, AD: 6, AC: 9, BC: 3, BG: 15, DE: 6,
CD: 9, CE: 7, CG: 6, EG: 3}.

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

a) (8%) 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). Stop when the *guaranteed* shortest path has been found.

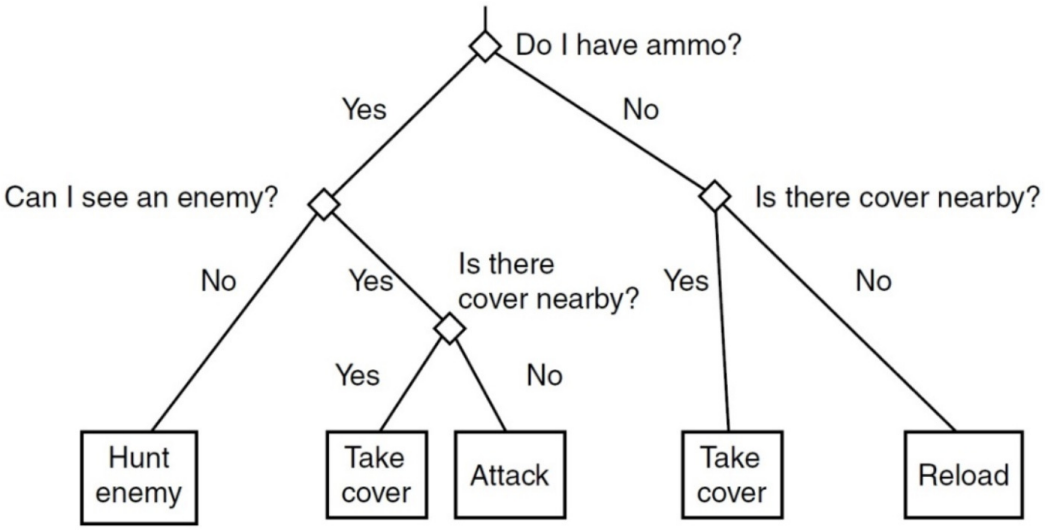
Current Node	Open list	Closed list
–	(S,0,–)	
S	(A,3,SA), (B,10,SB)	(S,0,–)

b) (1%) When can we stop if we are not interested in guaranteed shortest path?

c) (1%) How many paths to the goal node are evaluated if guaranteed shortest path is to be found?

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

Consider the decision tree from Figure 6.6. of Artificial Intelligence for Games 2nd Edition, by Millington and Funge, reproduced below.



a) (6%) Design a hierarchical finite state machine that would produce behavior similar to that of this decision tree.

b) (4%) Add an alarm behavior/mechanism (see Slide 30 of decision making lecture notes). To your hierarchical finite state machine from a), add alarm behavior such that the NPC, regardless of what is going on, will go to sleep (somewhere relatively safe) if too tired. Additionally, add the option of taking a quick nap if no enemies have been visible for a while and the NPC has ammo.

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) Environment and NPCs

- 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) viewpoint can see the entire level (*i.e.*, don't make the walls of the level too high). Fill the level with following geometry (must have all these, at least):
  - Three completely disjoint large rooms (disjoint meaning 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 these three large rooms, each connected by an open door, place smaller 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 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

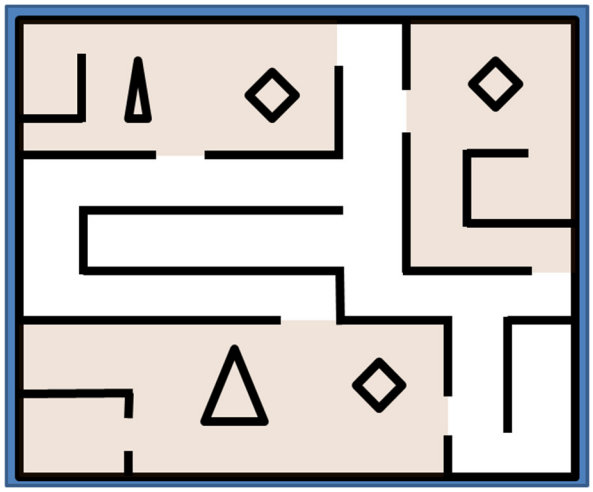


Figure 1. A sample level showing the minimal requirements

- You are to create a pathfinding graph using a regular grid. You are not allowed to use the Navmesh from Unity in any way.
  - For creating the regular grid (Tile) graph, follow the description giving in the class slides. Place the node representing each tile at the center of the tile, which is then connected by directed edges to its (up to eight) neighboring tiles. Choose size of a tile to yield at least 300 nodes over the level.

  For the pathfinding graph, make sure there are enough nodes such that there are a variety of paths through the room, and around the level. Render these nodes (simply as points or small discs) on the floor of the level. Weights of the edges will be sum of differences between the endpoints along X and Y axis (Manhattan Distance).

R2) A* Algorithm:

- Implement the A* algorithm on your pathfinding graph (the lab material will cover most of this). You will have to create two versions, one with the Manhattan Distance heuristic and the other one with the Cluster heuristic.
- For the Cluster heuristic, consider nodes in each room to contain a cluster, as well as each corridor to contain a cluster. Create a lookup table using shortest distance between two nodes (one from each cluster). This table is created offline and there is no need to show any computations for this.
- Demonstrate the "fill" for A* with the Manhattan Distance heuristic and that for A* with the Cluster heuristic as follows. Pick a random start node and goal node that are "far" from each other (in terms of path length) in your level. Compute the shortest path. Then on the level indicate graphically which nodes are included in the 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 behaviours to your collection from Assignment 1:
  1. Path Following – Implement this to delegate to Arrive (since the character must stop at the Goal)
  2. Looking Where You're Going (delegating to Align)
- Your character should perform the following movement behaviour:

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 which node of the pathfinding graph is closest to this initial starting position. This will be the pathfinding Start node.
2. Similarly, randomly select a target position in one of the small rooms adjacent to one of the other large rooms. Again, determine which node of pathfinding graph is closest to this target position. This will be the pathfinding Goal node.
3. Your program must also allow the user to choose a start and Goal node by clicking on the level.
4. Compute a shortest path using A* from the Start node to the Goal Node.
5. Move the character along the following path using the Path Following and Looking Where You're Going steering behaviours: initial position to the Goal node.

- Aim for movement is "realistic" and the character must not pass-through walls or obstacles while moving. E.g., you may want to smooth out the path computed using the Tile graph. No moving NPC should have a Rigidbody component attached. Do not worry about collision detection, but if collision detection is absolutely needed, it can be implemented by checking isTrigger on the collider.
- Perform basic path smoothing (refer to slide 67 from Path finding continued slides) and highlight both original path as well as the smoothed path in the level.
- You must use a fully animated humanoid character in this assignment. Instructions on importing fully animated humanoid models from the mixamo platform are made available through the labs.

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 any instructions necessary to get your program running. If your program does not run, we will not debug it.
2. Breakdown:
   - R1: 15% (equally divided among the requirements listed in item R1 above)
   - R2: 30% (equally divided among the requirements listed in item R2 above)
   - R3: 25% (equally divided among the requirements listed in item R3 above)
   - Richness of behaviors and general aesthetics: 10%
   - Source program structure and readability: 5%
   - Write-up: 5%
   - QnA with the Grader: 10%

What to hand in for Assignment

Questions 1 and 2: (20%) (Theoretical Questions)
- Submit your answers to question 1 and 2 **on Moodle** (as a PDF file **TheoryYourIDnumber.pdf)**

Question 3 (80%) (Practical Question)
Submission Deliverables (Only Electronic submissions accepted):
1. Submit a well-commented C# program for Unity 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 code (and brief write-up, explained below) must be submitted on Moodle in a **zip format** with all the required files (ex.: **YourIDnumber.zip**). ***Please do not e-mail the submission.***
2. Demonstrate your working program from an **exe file dated on or before March 6 at 23h55** to the lab instructor in the lab during the period of March 7-14.
3. Included in your zip file will be a brief write-up about your program depicting several fills of you're A* algorithm that you would like us to consider during the evaluation.