CS 4731/7632 Game AI
Exam 1

Instructions

- Download the attached MS Word Doc. Complete the questions and upload a PDF to Gradescope.
- For short-answer questions, two or three sentences will often be enough.
- There are two versions of question 10 depending on if you are in CS 4731 or CS 7632. Please answer only the one that is indicated for your section.
- Questions can be asked on piazza.
- Work is to be done individually without consultation with others in the class.
- You are allowed to consult the online notes.

Question 1. Which of the following are true about finite state machines and behavior trees? (highlight or bold yes or no for each)

a. Both require the same number of computations per tick to pick the next action to execute — yes **no**

b. There is nothing a behavior tree can do that a finite state machine cannot do — yes **no**

c. Both rely on the ability of the designer to create a good structure — **yes** no

d. Both can respond to novel situations not anticipated by the designer — yes **no**

Question 2. What are the advantages of a behavior tree over a finite state machine? (give two)

1. It is hard to represent sequences in a finite state machine while the behavior tree can easily handle the sequence structure because you can use a Sequence, Selector to represent sequences.
2. It is hard to do decision-making in a finite state machine because transitions are based on external stimulations. However, we can use Selector, Sequence and Daemon in a behavior tree to implement decision-making.

Question 3. What are reasons a game developer to choose a rule system over a behavior tree? (give two)

1. Using a rule system can get rid of coding specific responses to specific situations.
2. A rule system can respond to novel situations while a behavior tree cannot.

Question 4. What are reasons one should use HSP behavior planning instead of behavior trees? (give two)

1. HSP can be used to achieve a goal by determining which actions must be executed and in what order. However, a behavior tree cannot tell how to reach a goal by taking what actions.
2. Also, HSP allows actions to be executed simultaneously, meaning that we don't need to worry about whether this action should happen before the other one. But a behavior tree executes actions in sequence instead of simultaneously.

Question 5. Under what situations would one choose to use A* or all-pairs-shortest-path (e.g. Floyd-Warshall) for pathfinding?

For static path networks, we can use Floyd-Warshall to pre-compute all paths and store them for further need. But if the path networks are dynamic, it would be time-consuming to update the paths whenever the path networks change. However, A* can handle dynamic path networks quickly.

Question 6. How can the use of formations reduce computation time in pathfinding?

Formations can be used to navigate to offset when there are no obstacles. A* algorithm is called when formations lose sight of the offset, which helps reduce computation time in pathfinding.

Question 7: What are pros and cons of grids versus path networks in pathfinding? (give one pro and one con for each)

Grid:

Pros: Discrete space is simple, and grids can eliminate collisions.

Cons: Discretization wastes space, and grids make agent confine to cells, resulting in artificial movements.

Path Network:

Pros: preserve sense of continuous environment

Cons: hard to design efficient path nodes for path networks.

Question 8: List two differences between pathfinding and behavior planning using A*.

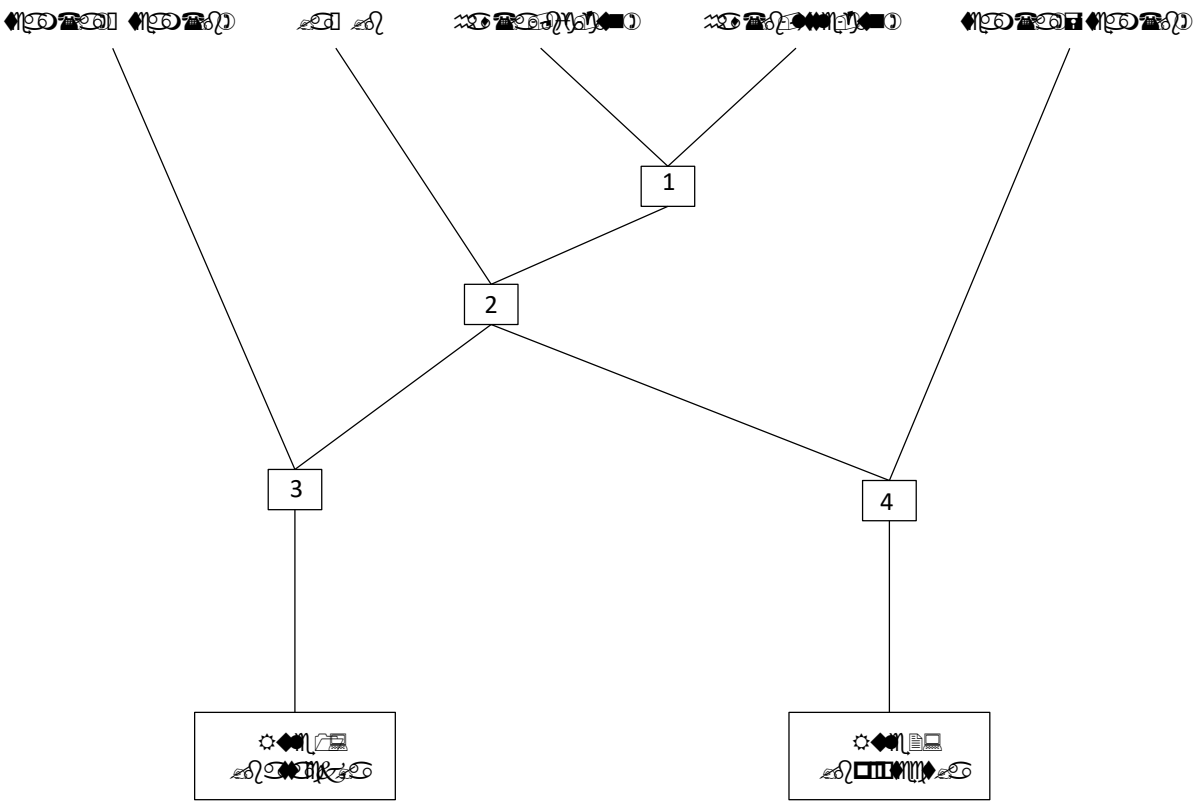
1. A* in pathfinding uses the position (x, y) as a node in the network to represent the path nodes and target location while A* in behavior planning uses STRIPS to represent each state and the goal state.
2. In pathfinding, the Euclidean distance is used for heuristic. However, actions/behaviors are much more complex than in pathfinding and there is no obvious heuristic such as Euclidean distance.

Question 9: Answer the following question about the Rete algorithm.

Suppose the following facts are true in the world:

has(coulson, little-gun)	coulson is on team 1
has(daisy, little-gun)	daisy is on team 1
has(daisy, big-gun)	mac is on team 2
has(mac, little-gun)	may is on team 2
has(may, little-gun)	

Consider the Rete graph below:



Questions are on the next page.

9.a. There are four join nodes in the graph above, numbered 1-4. For each join node, list the different possible combinations of values for ?a and ?b to take on. Each box in a column is for a different possible assignment of ?a and ?b. You may not need to use all the boxes below.

Join 1	Join 2	Join 3	Join 4
?a= daisy ?b= coulson	?a= daisy ?b= coulson	?a= daisy ?b= mac	?a= daisy ?b= coulson
?a= daisy ?b= daisy	?a= daisy ?b= mac	?a= daisy ?b= may	?a= ?b=
?a= daisy ?b= mac	?a= daisy ?b= may	?a= ?b=	?a= ?b=
?a= daisy ?b= may	?a= ?b=	?a= ?b=	?a= ?b=
?a= ?b=	?a= ?b=	?a= ?b=	?a= ?b=
?a= ?b=	?a= ?b=	?a= ?b=	?a= ?b=
?a= ?b=	?a= ?b=	?a= ?b=	?a= ?b=
?a= ?b=	?a= ?b=	?a= ?b=	?a= ?b=

9.b. Give which rules will become activated and the values of ?a and ?b (note: a rule could be activated more than once with different parameters)

Rule 1 activated: may attack daisy
Rule 1 activated: mac attack daisy
Rule 2 activated: coulson protect daisy

Question 10 (CS 7632). Given that a finite state machine (FSM) is Turing complete, why, from a development perspective, might game developers prefer any of the following to an ordinary finite state machine: (a) hierarchical FSMs, (b) behavior trees, (c) rule systems, or (d) A* behavior planning. Give specific examples for each of these techniques in terms of maintenance, reuse, coding effort, cognitive effort, time effort, or other factor that is not related to computation time.

(a) Hierarchical FSM
It is easy to implement the FSM from abstraction because how each state can be transformed is clear. And it is good in encapsulation. But the cognitive effort is bad because it is hard to think about states in sequence and the actions are hidden

(b) Behavior Trees
Behavior trees are good in cognitive effort, making people easily understand the actions sequences. But the coding effort is heavy because the developer has to cover all possible actions for desired control routines.

(c) Rule Systems
Rule Systems are good in reusing and maintenance because it can handle novel situations while previous method cannot. And the coding effort would be acceptable because the developer does not need to consider all different situations.

(d) A* Behavior Planning
A* Behavior planning can be very helpful in pursuing a specific goal with a sequence of actions while previous methods cannot. And reusing is acceptable because as long as we can present the states in STRIPS, we can apply A* Behavior Planning.
But the cognitive effort is bad because the heuristic is complex, not as the Euclidean distance in pathfinding and the coding effort is bad because it has to implement two "relaxed" planning problems.