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: Appearance of goal driven behavior
2: Daemons can help recover from errors

Question 3. What are reasons a game developer to choose a rule system over a behavior tree? (give two)
1: No need to have specific responses for specific situations
2: Can respond to novel situations

Question 4. What are reasons one should use HSP behavior planning instead of behavior trees? (give two)
1: Guarantee agent can get to a goal

2: Thinks ahead. Behavior trees are a specific action for an unique situation. HSP Behavior planning is a path search algorithm so as it searches through combinations of actions, it thinks ahead.ly

Question 5. Under what situations would one choose to use A* or all-pairs-shortest-path (e.g. Floyd-Warshall) for pathfinding?
One would use A* for pathfinding in a dynamic environment since A* is a single source, single target search for an optimal path.
One would use Floyd-Warshalls when paths can be pre-computed before the game is running, in most cases for a static environment. For example, one can pre-compute all pairs shortest paths before a game ships and whenever an agent needs to get to a destination, it can use the lookup table that was pre-computed to get to the desination.

Question 6. How can the use of formations reduce computation time in pathfinding?
When a group of agents are in a set formation, we can reduce the computation time in pathfinding by designating a leader in the formation. Only the leader computes the A* path and all others in the formation use straight shot pathing to follow the leader, with offsets to avoid single file formation. Obstacle collision due to being offset from the leader's position can be mitigated by implementing proxy agents for each follower.

Question 7: What are pros and cons of grids versus path networks in pathfinding? (give one pro and one con for each)
Grid path:
- Pro: inherently eliminates collisions since only one entity can be in a cell. This lets us do less computation since we don't need to handle collisions
- Con: The search space is very large. To find a path, the search algorithm must visit a lot of cells. Its search space is $O(8^N)$ for an 8 -connectivity grid world where N is the number of cells between start and goal.

Path Networks:
- Pro: Path networks discretize a large continuous space into a small representation while preserving the sense of environment. The computation of finding a path is a lot cheaper than grids and agent movement can be made to look fluid and continuous with path smoothing and shortcutting.
- Con: since there are many choices and combinations of path node placement, finding the optimal path node placement is a very difficult problem. This problem needs to take into account every reachable location in the map while optimizing path nodes so that there are not too many or too little. If there are too many, the computation time increases but if there are too little, an agent will take unnatural paths to get on the path network.

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

One difference is the heuristic that is used. Pathfinding A* uses a straight line to destination and distance traveled from source heuristic. The heuristic used for behavior planning is a relaxation of the restrictions on the rules and actions, allowing actions to ignore negative effects and occur simultaneously. While the pathfinding heuristic operates on the original path network, the behavior planning heuristic creates a relaxed representation of an actions path network.

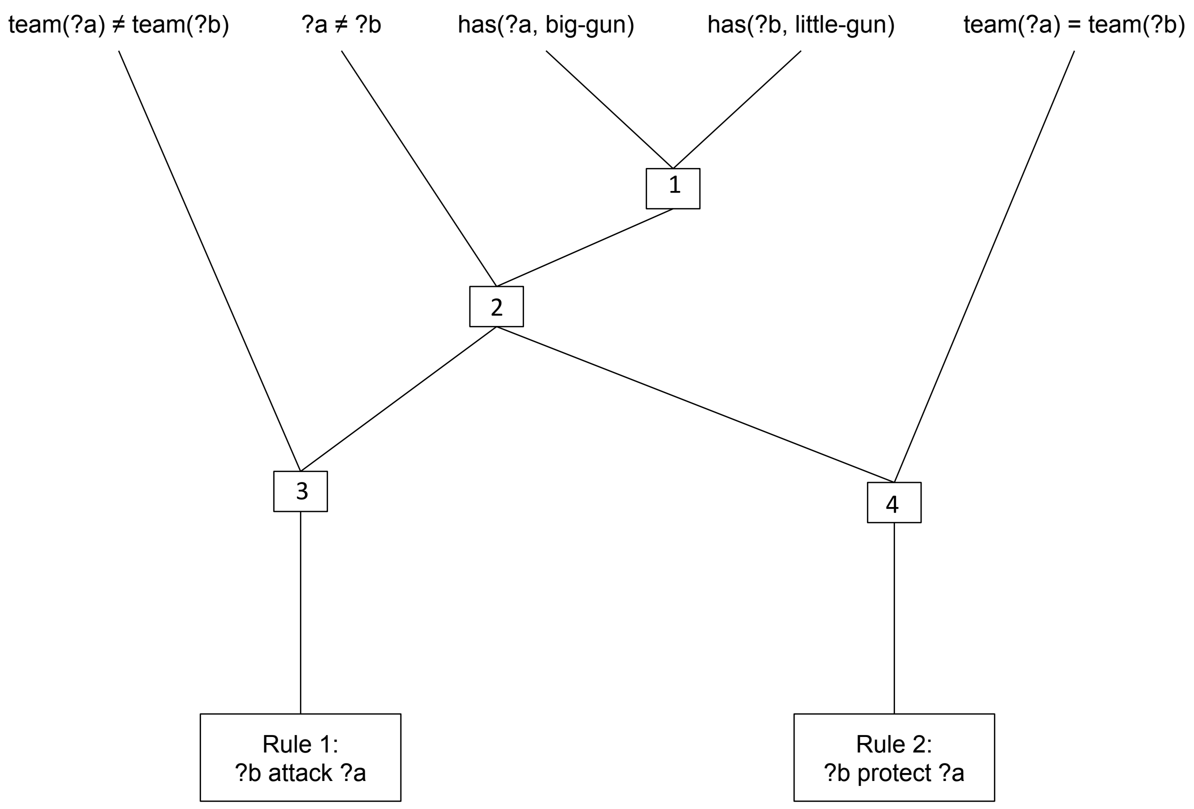
Another difference is what the path network of pathfinding and behavior planning represents. In path planning, nodes are stop locations and edges represent the cost/distance to reach the next node. In behavior planning, each node represents the state of the world and edges represent actions taken. To travel an edge in behavior planning, there are pre-condition and post-conditions associated with the action that the edge represents. To travel an edge in pathfinding, the only cost is the distance traveled.

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:



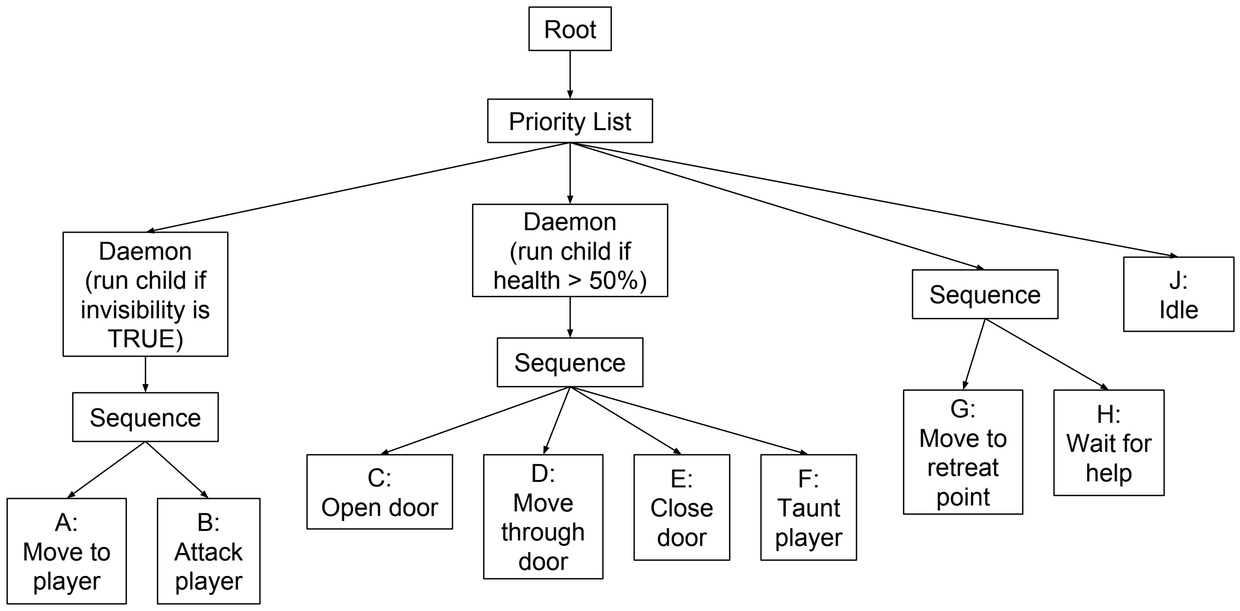
Questions are on the next page.

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= ?b=	?a=daisy ?b=coulson
?a=daisy ?b=daisy	?a= ?b=	?a= ?b=	?a= ?b=
?a=daisy ?b=mac	?a=daisy ?b=mac	?a=daisy ?b=mac	?a= ?b=
?a=daisy ?b=may	?a=daisy ?b=may	?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=

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 will be activated twice with (?a=daisy, ?b=mac) and (?a=daisy, ?b=may).
Rule 2 will be activated once with (?a=daisy, ?b=coulson).

Question 10 (CS 4731): Suppose you are building a behavior tree for an NPC in a combat game. Agents can be invisible or visible and change their tactics accordingly. Assume for the purposes of the scenario below that the agent never acquires invisibility and that the agent has full health. The agent is near a door to a room. Finally, suppose that earlier in the game that the door has been damaged so that it cannot close completely and that any action to close the door will fail. All move actions require 5 ticks to complete. All other actions execute in one tick, except "wait for help", which takes 10 ticks to complete.



List the order in which actions will start execution:

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) The hierarchical aspect of hierarchical FSMs allows for encapsulation of certain sets of states. This makes the designing process a lot easier to think about than designing a low level FSM. Imagine a FSM for the behavior of a cleaning robot that cleans when the floor is dirty and charges itself when the floor is not dirty. Each process requires many sub-actions and conditions to perform. To be able to classify each as a cleaning state and charging state allows designers to focus more on the overall behavior design. Encapsulation generally translates to reusable code which decreases coding effort and time. With object-oriented programming, the same behavior can be changed across all agents implementing it by editing just the encapsulated state.

b) Behavioral trees have all the advantages of hierarchical FSMs in terms of encapsulation, reusability, maintenance, and ease of understanding through sequence and selector nodes. With selector nodes, priority is given to actions by their order in a selector node. With daemons, behavior trees allow designers to short-circuit a sequence of actions if something more important appears. Priority actions and daemons allows for easier and faster implementation of an agent that appears to be goal driven. For example, for a hero that is hunting an enemy, if their health or stamina drops too low, they should go back to heal. Behavioral trees allow this logic to be easily translated into code with a daemon that checks the health and stamina of the hero while it's in the hunting sequence. This lightens the cognitive effort significantly since a behavioral tree is simply a big list of yes or no branches. Overall development time is reduced since we only need to design one decision tree for each agent.

c) Rule systems allows designers to save time, coding effort, and cognitive effort by not designing specific responses to unique situations. Instead, it lets designers define a set of rules and corresponding bag of actions to give an agent a general behavior. Designers can start with a general goal of an agent and create rules and actions that guide the agent towards accomplishing a goal. However, it is important to note that goal achievement is not guaranteed. This system can be used for dialogue design where human dialogues can be modeled as a set of states and the dialogue manager chooses replies for its conversation given a set of rules. The cognitive effort may be increased to develop good rules for an agent, but rule systems eliminate the need to code specific responses to each situation in a game.

d) A* behavior planning allows designers to model the world as a set of conditions that allow for certain actions to happen. A crucial aspect of A* behavior planning is that it guarantees goal achievement. With this, designers do not have to worry about their agents not performing what they intend. If a goal is reachable, the A* behavior planning will be able to find it. In addition to guaranteeing goal achievement, the A* aspect allows for dynamic re-planning of actions. Once the game world and actions is defined, the time to create agents with different goals is significantly decreased. Since all the designers need to do is instantiate an agent and give it a goal state to attempt to reach. There is no designing a specific sequence of action to reach the goal as the agent with search for and execute actions by itself.