**Name:**

**Student ID:**

**CM 146**

**Midterm Exam**

2/7/20

This is a take-home exam, **due before class on Monday, 2/10**. It is open notes & materials.

You can submit it (a) electronically via the submission link, or (b) as hardcopy in class or at Prof. Shapiro’s office (E2-263). If you choose (b), please print out the questions and insert your answers after each question and **REMEMBER** to **WRITE YOUR NAME** and your **STUDENT ID NUMBER** on your test. If you add extra sheets, remember to identify the problem number and part you are answering.

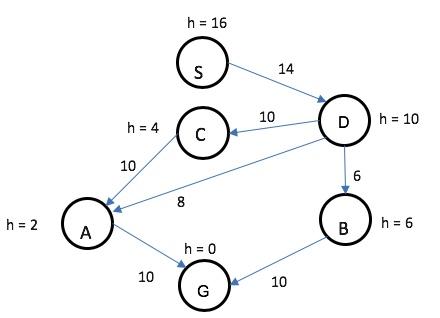
As usual, this exam must represent your personal work, per University policy. Please do not discuss the exam with other students until after Monday’s class.

**You may post clarification questions on Canvas,** **but not answer them for other students**. We will monitor the Canvas discussion and try to answer questions as soon as possible.

**CM 146, Midterm, Fall 2018**

**20 points**

1. Consider the following directed graph where S is the start and G is the goal, transition costs are on arcs, and h is the heuristic estimate of distance to the goal.



A. **8 points**. Apply Greedy Best First Search to find a path from S to G. In the table below, fill in the contents of the open list (the priority queue), where each line in the table shows the contents for a step in the search. Write your open list entries in this form:

(<node> priority: <priority>, <parent>→<node>).

For example, if the priority of node B is two, and if B was found by traversing a link from A, then the open list entry would be (B priority: 2, A → B). Identify the final path and cost in the last line of the table.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

B. **8 Points.** Apply A\* to find a path from the S to G. In the table below, fill in the contents of the open list, where each line in the table shows the contents for a step in the search. Write your open list entries in the form (<node> priority: <priority>, <parent>→<node>), as in part A, above. Identify the final path and cost in the last line of the table.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

C. **4 Points:** Explain why the paths found in parts A and B are the same, or different.

**CM 146, Midterm, Fall 2018**

**20 points**

2. Watch the great chomper scene from Galaxy Quest (click on the image) or chase this link:

<https://www.youtube.com/watch?v=gqRdT8m1Suo>

[](https://www.youtube.com/watch?v=gqRdT8m1Suo)

If anyone were so cruel as to build the chompers shown in this movie, they would also need a method of cleaning up after any mess the chompers would sadly produce. Here, we delegate that task to a cleaning robot (not seen in Galaxy Quest) that is controlled by the same mechanism that operates the chompers.

Your task is to compose a single Hierarchical Finite State Machine that controls a system composed of two chompers, like those in the movie, together with the cleaning robot. Submit a drawing of your HFSM that clearly labels the states, transitions, transition events, and actions that take place in each state (if they are not synonymous with the state’s name). Use hierarchy to prevent duplication of arcs with the same transition condition; parallel execution is also permitted.

Your HFSM can employ the following events:

* Heartbeat Detected (H)
* No Heartbeat Detected (NH)
* Mess (M)
* No Mess (NM)
* nSec – n seconds have elapsed since entering a state

Your HFSM should have the following behaviors:

* While the chompers are active, the robot is idle.
* While the robot is active the chompers are idle. Assume idle chompers are open.
* Chompers begin in an armed state, waiting for anything with a heartbeat to arrive.
* They return to the armed state if the lifeform (or lifeforms) with the heartbeat escapes.
* Once active, chompers keep chomping until nothing with a heartbeat remains.
* The robot is only active when there is a mess.
* The robot returns to idle when the mess is gone. It never stays active for more than 4 minutes.
* The robot cleans by sweeping the floor, hosing down the equipment, blasting everything with a flamethrower, and then polishing out any remaining spots, each for 30 seconds. It repeats this process indefinitely, until it is told to stop.
* Each chomper follows a regular, timed pattern of open and close. The patterns are different.
* Chomper A’s first chomp is immediate. Chomper B’s first chomp is 1 second later.

Sketch your HFSM here or insert an extra sheet.

**CM 146, Midterm, Fall 2018**

**20 points**

3. The behavior trees presented in class have encoded deterministic (if time variant) mappings from situation to action. However, characters often seem smarter if they are less predictable. One way to create that effect is to add nondeterminism to behavior trees.

For parts (a) and (b) below, use the expression child.excute() to invoke the Behavior Tree interpreter on a child node. Assume that the return values of True and False map on Succeed and Fail, respectively. Assume the decorator Until Fail returns true when its child node fails, and Until Succeed returns true when its child succeeds.

(a) **2 points.** Write pseudo-code that implements a version of a Selector, denoted



that is identical to a normal Selector, except it considers its children in a random order, vs a fixed order.

(b) **2 points.** Write pseudo-code for a version of Sequence, denoted



that is identical to a normal Sequence, except it considers its children in a random order, vs a fixed order.

(c) **6 points.** Use these new node types together any of the following (Sequence; Selector; actions; checks; and the decorators Inverter, Until Fail, Until Succeed, Persist) to draw a behavior tree with this specification:

* Your character needs to move through a room. That room might contain a single enemy.
* The character will randomly consider fighting the enemy and crossing the room without a fight.
* Creeping through the room is only relevant if the character hears the enemy. Moving through the room is always relevant. Creeping is preferred.
* Fighting is only relevant if the enemy is visible.
* To fight, the character will hit the enemy, pause, and check to see if the enemy is conscious in a random order. This continues until the enemy is unconscious.
* When the enemy is unconscious, the character ties it up.
* On completion of this behavior, the character has either crossed the room, or is in the room facing a restrained and unconscious enemy.

Insert an extra sheet with your answer.

(d) **10 points.** We built a HFSM for a trash collecting robot in class. This question asks you to define a portion of its activity as a behavior tree with parallel branches.

We define a parallel node, denoted



as a non-primitive node with an arbitrary number of children. All of those children will be tried once, and all must succeed for the parallel node to succeed. All of the children are launched in parallel. The moment any child returns Fail, the parallel node returns Fail.

Draw a behavior tree with the following specification:

* The robot is either tidying up trash, or recharging.
* Tidying is only relevant so long as trash is visible.
* Recharging is only relevant so long as trash is not visible.
* The tidying action never terminates of its own (it neither returns Success or Failure).
* The recharging action never terminates of its own (it neither returns Success or Failure).
* The robot continues to execute its tidying and recharging behavior forever.

As before, you may use Sequence; Selector; actions; checks; and the decorators Inverter, Until Fail, Until Succeed in your tree.

Draw your tree here or insert and extra sheet with your answer.

**CM 146, Midterm, Fall 2018**

**20 points**

**Note: Your responses for this problem should be attempted in Python for clarity. However, you will not be penalized for syntactic errors.**

4. Consider the implementation of the Monte Carlo Tree Search below in order to solve this question for some imagined *single-player* game.

class Game:

def \_\_init\_\_(self, \*params): pass

def legal\_actions(self): pass

def apply\_move(self, move): pass

def is\_terminal(self): pass

def score(self): pass

class Node:

def \_\_init\_\_(self, last\_action, action\_list, parent\_node=None):

self.child\_nodes = {}

self.score = 0

self.visits = 0

self.parent = parent\_node

self.parent\_action = last\_action

self.untried\_actions = action\_list

def ucb(child\_node):

parent\_node = child\_node.parent

return (child\_node.score/child\_node.visits) + \

sqrt(log(parent\_node.visits)/child\_node.visits)

def traverse\_nodes(node, game, bot\_identity):

while not game.is\_terminal() and not node.untried\_actions:

child\_node = max(node.child\_nodes.values(), key = ucb)

game.apply\_move(child\_node.parent\_action)

node = child\_node

return node

def backpropagate(score, node):

while node:

node.score += score

node.visits += 1

node = node.parent

(a) **8 points.** Partial expansion is a modification to MCTS used primarily for games with large branching factors, i.e. having numerous possible legal actions at every state of the game. The high-level concept is to continue the traversal process beyond a node that still has untried actions if the node has already explored a minimum number of actions (resulting in the same number of corresponding child nodes). One common variant adjusts this minimum as a function of the node’s number of visits.

Reimplement the function traverse\_nodes with partial expansion, where the minimum threshold to continue traversal compares the number of untried actions against the square root of the node’s visit count.

(b) **8 points.** “Mixmax rewards” is a modification to MCTS that seeks to retain information regarding strong outcomes seen in nodes farther down the tree. It requires two changes:

* Each node must store the maximum of the average scores of any of its children.
* The value for a node---typically just the win rate or average score---during the tree traversal is now calculated as 80% of its average score + 20% of the maximum average score of its children.

Change the appropriate functions to implement mixmax rewards. Assume tree nodes have an additional field called “max\_score”.

(c) **4 points.** Describe the behavior of MCTS with mixmax rewards as you vary the relative importance of a node’s average score and the maximum average score of its children.

Please insert extra sheet(s) with your answers to 4a - 4c.

**CM 146, Midterm, Fall 2018**

**20 points**

5. Short Questions.

Insert extra sheet(s) with your answers.

(a) **2 points.** Given a 2D square grid that allows only 4 directions of movement (left, right, up, down), a starting cell (x1,y1), a goal cell (x2, y2), and a current cell (x,y). Which of these heuristics for path search makes A\* explore the least amount of tiles:

1. (x2 - x)^2 + (y2 - y)^2
2. abs(x2 - x) + abs(y2 - y)
3. abs(x2 - x)
4. abs(y2 - y)
5. Euclidean distance to the goal

(b) **2 points.** Suppose you had a perfect heuristic for an A\* search that returned true cost from a node to the destination. What nodes would A\* explore?

(c) **2 points.** By using GOAP, the designers of F.E.A.R were able to reduce the Finite State Machine component of their AI to two states (move to, animate). What work did employing GOAP save relative to encoding the NPC behaviors in F.E.A.R as FSMs (or HFSMs)?

(d) 2 points. What capabilities do Behavior Trees bring to the problem of specifying behavior that HFSMs lack (or make difficult to express)? Check all that apply:

1. more reactive response
2. explicit transition logic
3. parallelism
4. specialized control structures
5. separates goals and actions
6. alternative methods of accomplishing a task
7. hierarchical decomposition of task and actions

(e) **2 points**. It is often difficult to find heuristics for guiding Goal Oriented Action Planning. The lecture on Advanced Planning suggests using the minimum number of steps required to find the goal if you ignore all items in the delete list of actions. Is this heuristic admissible for GOAP? In 1-2 sentences explain why or why not.

(f) **2 points**. Consider the following rules, which were present in PromWeek:

The enemy of my enemy is my friend

(Enemies ?x ?y)( Enemies ?y ?z) !(Friends ?x ?z) => (Friends ?x ?z)

The Enemy of my friend is my enemy

(Friends ?x ?y) (Enemies ?y ?z) !(Enemies ?x ?z) => (Enemies ?x ?z)

Assume that working memory contains the following instances:

(Enemies John George)

(Enemies George Herbert)

(Enemies Herbert Rufus)

What are the contents of working memory after running the rules in the order shown?

(g) **3 points**. You need to decide what classes to take next quarter. You must take 3 classes out of the following list.

|  |  |  |
| --- | --- | --- |
| Introduction to belly-button gazing (IBBG) | MWF | 12:50 - 13:50 |
| Advanced pumpkin carving (APC) | WTh | 13:30 - 15:00 |
| Introduction to Symmetry and its Applications to Quantum Mechanics (IQ) | TTh | 15:00 - 16:30 |
| Binge-watching The Good Place (TGP) | WF | 14:00 - 23:59 |
| Star Trek: Theory and Applications (STTA) | MWF | 14:00 - 15:00 |

Formulate your scheduling problem as a constraint satisfaction task (show variables and domains, name at least two constraints). You can express constraints in English.

(h) **2 points**. An HTN approach to Minecraft planning would decompose the task of making a bench into two component tasks:

def make\_bench\_from\_planks (state):

return [(have\_4\_planks, state), (craft\_bench\_from\_planks, state)]

What is the advantage of Hierarchical Task Planning relative to a forward (or backward) search approach that only has access to primitive actions? (1-3 sentences)

(i) **3 points**: The following version of traverse\_nodes refuses to return a terminal game state; it returns the most desirable non-terminal node in the MCTS tree (or None, if no such node exists). Assume that is\_terminal returns true for a terminal game state, and returning None from traverse\_nodes causes the calling function (Think) to stop MCTS search and play its best move.

What is the benefit of this code relative to returning a terminal node from traverse\_nodes and passing it directly to backpropagate as a game with a known score? (1-3 sentences)

def traverse\_nodes(node):

if node.is\_terminal() return None

if node.untried\_actions return node

for n in sort(node.child\_nodes.values(), key = ucb):

res = traverse\_nodes(n)

if res return res

return None

**Extra Credit**

**7 points**

Consider the following behavior tree:

A

B

C

D

**X** is either a Sequence or Selector node (you do not know which). **A**, **B**, **C**, and **D**, are arbitrary subtrees.

Let **O** be an optional behavior, whose introduction should have no impact on the semantics of the above tree – its Success or Failure, the execution order of its components, or the number of times they are executed.

(a) **2 points.** An unnamed Professor claimed that you cannot introduce a subtree between **B** and **C** that encodes an optional behavior, **O**, while leaving the remainder of the tree intact (and without altering the implementation of the node type **X**). Is this statement true, or false? Explain why in one or two sentences.

(b) **5 points.** Suppose you could transform the tree shown above into another tree by duplicating structure, rewiring structure, and/or introducing additional Sequence and Selector nodes. Draw a transformed tree that encodes an optional behavior, **O**, that will be considered after **B** and before **C**, while preserving the semantics of the original tree.