Full Name:_____

CS 4300, Fall 2019

Final

Instructions:

- Put your name on the paper. Mark your chosen tasks on the paper. Leave any necessary notes on the task page. Turn the paper in at the end of the exam.
- Complete two tasks from the following list. You will not receive credit for multiple tasks in the same project group.
- The work completed here, and the assignment work it is based on, must be your own.
- Most tasks will require a report submitted to Canvas and a push to github.

Rubik's Unique Color Heuristic (50 Points)

As part of the Rubik's cube project, we discussed several ideas for heuristics. You were required to implement a couple of different heuristics. This is an additional idea for a heuristic.

The more organization in the cube colors, the closer it is to being solved. For this heuristic, we'll measure disorganization as the number of colors visible on a face of the cube.

For each of the 6 faces, count the number of unique colors on the face. Sum these values, and divide by 4.

Task

Add the heuristic as an option to your search problem class. Be sure to add it such that its enumeration number is 10. You can do this with an assignment in the enumeration like `COLOR_COUNT_HEURISTIC=10`.

Make sure the rubiks cube solver tool can use `config heuristic 10` to use this heuristic.

Use the tool with this heuristic and A* search to solve the cubes given. Find `ColorCountCubes.txt` in the exam files. Write a brief report with the cost of solution found for each cube, and the number of nodes generated in the search. Submit the report to the Canvas task.

Be sure that your code with this heuristic is pushed to github.

Rubik's Dot Goal (50 Points)

One of the interesting patterns for a Rubik's cube is the four dot pattern. It looks like this in the T-view.

```
    www
    www
    www
bbb ooo ggg rrr
bgb oro gbg ror
bbb ooo ggg rrr
    yyy
    yyy
    yyy
```

Task

Create a rubiks command script that will solve any cube to this state. This is like the final script you built for the homework, only the final goal is different.

Use your script to solve the cubes given to that state. Find `DotCubes.txt` in the exam files. Write a brief report with the commands needed to solve each one. These should be written in a format like this:

```
init cube the cube to be solved
rotate L R' ... the commands to solve the cube.
```

Include the location of this script in the report. Submit the report to the Canvas task.

Be sure that your script is pushed to github.

Jetan Quiescence Search (50 Points)

Quiescence search is an extension of search used when the cut-off depth has been reached, but the board is in a turbulent state. We'll define turbulent to be if either princess is under attack. Because it may take many levels for the princess attack to be resolved, we don't want it to go on for unlimited levels. So, we'll implement the quiescence search with a depth limit of its own.

Task

In the alpha-beta search agent, add a pair of Min/Max methods to be used for quiescence search. The cut-off for these will be if the game is over, or the board has become quiet (princess no longer under attack), or if the quiescence depth has been reached.

The normal Min/Max methods will need to be modified to call the correct quiescence method if the board is not quiet when max depth is reached.

Run your agent against itself. One instance with quiescence search and one without. Use the same max depth for both, and give a quiescence depth of 2 for the quiescence agent. Play at least 4 games (2 in each direction).

Write a brief report on the results. Include instructions for using quiescence search with your agent in the report. Submit the report to Canvas.

Be sure to push your code to github.

Jetan Evaluation Function (50 Points)

As part of the Jetan project, we discussed several ideas for elements of evaluation functions. You were required to implement a couple of evaluation functions as part of the assignments. Here's an idea for an additional heuristic.

Being in the middle of the board could be advantageous, as more moves are available to the pieces in the middle. We'll encourage pieces to move to the middle by assigning value to the location of the piece. The value will be 12, minus the Manhattan distance from the center of the board (4.5,4.5) to the piece's location. For example, a piece located at position (2,9) will have a value of 12 - (abs(2-4.5)+abs(9-4.5)) = 12 - (2.5+4.5) = 5.

So, the evaluation function will calculate this value for your pieces, and subtract from it this value for your opponent's pieces.

Task

Add this evaluation function to your agent.

Play your agent against itself where both agents use this evaluation function. Play your agent against itself where one agent uses this evaluation function, and the other uses your normal evaluation function. That's a total of 2 games.

Write a brief report with the results of the games, and your observation on the motion of the pieces under this evaluation function. The report must also include instructions on running your agent with this evaluation function. Submit the report to the Canvas task.

Be sure that your code with this evaluation function is pushed to github.

Wumpus Probability (50 Points)

We discussed adding risk to the Wumpus agent by allowing it to take a calculated risk when a cell's probability of containing a pit was low enough. If you implemented probability risk in your solution, then this problem is for you. If you didn't already implement it, you probably don't want to take the time now.

Task

Run your agent with risk taking disabled, on a set of 100 (or more) worlds, with the world sizes varying from 3 to 5 in each dimension. Use a pit probability of 0.2. Collect data on the number of deaths, golds, and score. Repeat with the same world parameters, including random number seed, with risk taking turned on. Collect the same data. Repeat both runs again, but with pit probability of 0.1, collecting data.

Now you have 4 sets of data to compare for risk taking vs. non-risk taking. Create a short report with this data, and your interpretation of the results. Submit the report to the Canvas task.

Be sure that your code with probability analysis is pushed to github.

Wumpus Nirvana (50 Points)

The tribe of *Supmuw* considers being eaten by a wumpus to be Nirvana, the highest obtainable state of existence. (Being eaten by the wumpus gives about the same experience as listening to grunge music.) If the wumpus is in the same location as a pit, then the adventurer would fall into the pit before being eaten, not Nirvana. The adventurer must be sure that the wumpus isn't with a pit before walking into the wumpus' location. Nirvana is only reached if the agent yells "Cowabunga Dude!" in the last action before moving forward into the cell that contains the wumpus and no pit.

Task

Modify your wumpus agent to seek Nirvana.

- You will want to able to ask the knowledge base if Nirvana is achievable in a given cell (yes wumpus, no pit).
- Add a method to the agent the checks if the cell that is forward of the agent is Nirvana achievable. If it is, then remove any contents from the plan queue, and push a `ai::Wumpus::Action::YELL` action and a `ai::Wumpus::Action::FORWARD` action, then return true. If the forward cell is not Nirvana achievable, then do nothing and return false.
- Add to the if/else if chain in `ChooseAction()` after `else if(!plan.empty())` and before `else if(ForwardIsSafe())` an `else if` that calls your new method. If it returns true, pop an action from the queue.

Run your agent with this enabled on at least 100 worlds with the nirvana option turned on in the server: `--nirvana 1`. The server will reward the agent with 10000 points if Nirvana is obtained by yelling followed by being eaten. Find `RUN_Nirvana` in the exam files to make this easier.

Save the server error and log files, and submit them to the Canvas task. Use git to push your code.