1)

A) RoboCup 3D Simulation Robot

Performance Measure: Goals scored and goals scored against us.

Environment: Football field with other robots.

Actuators: legs (servo motors).

Sensors: camera, proximity sensor

Agent: A goal based agent should be enough for this type of agent. The robots should be trying to Increase the score without increasing the opponent teams score and do nothing else as that’s their ultimate goal.

B) Search and Rescue Robot

Performance Measure: Time took to search. Safety of the rescued person.

Environment: Terrain with Obstacles and a single or multiple rescue goal(s).

Actuators: wheels. Grappling mechanism.

Sensors: camera, proximity sensor, microphone, thermal camera.

Agent: The agent for this task should be Utility-Based because it involves multiple goals which may not all be optimized at the same time. The ultimate goal is to rescue the target safely but this goal can not be defined as a simple parameter.

C) An agent which decides priorities of emails and sorts these emails by their priorities

Performance Measure: Satisfaction of the mailbox owner. Amount of truly buried mails and truly prioritized mails.

Environment: Mailbox with mails of variable priority.

Actuators: Mechanism to move mails up or down the mailbox.

Sensors: Text Input Stream, date reading mechanism.

Agent: A Simple-Reflex agent can be used for this problem because it does not require a state to process its information and the problems environment is fully observable, a basic set of rules should be enough to classify e-mails.

D) Autonomous car driver agent

Performance Measure: Safety of passangers other cars and pedestrians. Time taken to reach point A from point B.

Environment: Traffic with other cars pedestrians and traffic lights.

Actuators: Pedal, strering wheel, transmission.

Sensors: Cameras, Speedometer, proximity sensors.

Agent: A Utility-Based reflex agent can be the better option for this task. The autonomous car driver agent should be acting on what it perceives from the world and should be trying to accomplish a diverse set of goals at the same time. The agent does not only have to bring passengers from A to B safely but it also should do this as fast as possible.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Task Environment | Observable | Deterministic | Episodic | Static | Discrete | Agents |
| RoboCup 3D Simulation Robot | Fully  (Entire field is visible for the robot) | Strategic  (The changes in the environment depends on what the other teams robots do) | Sequential  (The robots need memory in order to counter the tactics of the opposing robots) | Semi  (there is both time and space factors determining the success of this AI) | Continuous  (The flow for this problem doesn’t consist of set states) | Multi  (The team of bots face an opponent team of similar bots) |
| Search and Rescue Robot | Partially  (Although the environment can be observed not everything is certain as things such as the state of the recue target may be unknown) | Stochastic  (There are a lot of random variables such as the state of rescue target wind etc that can affect the AI) | Sequential  (Robots have to remember past actions in order to try tackling obstacles in multiple ways) | Dynamic  (The environment continuously changes for this AI, if the robot is in a forest trees may fall) | Continious  (there aren’t discrete states for this problem) | Single  (the robot does not compete against anything intelligent.) |
| An agent which decides priorities of emails and sorts these emails by their priorities | Fully  (Entire mailbox is visible to the AI by default) | Deterministic  (The robot wil act on set rules which will never produce random incomes, two distinct mails will always be sorted in same order) | Episodic  (This robot requires no memory as its actions do not depend on its previous actions assuming there is no human feedback) | Static  (Through the sorting of the mailbox always the same mails will exist inside the mailbox) | Discrete  (There are set states for this ai with different sortings of the mails) | Single  (The mail sorter does not compete against anything intelligent) |
| Autonomous car driver agent | Partially  (Environment can be observed but the AI can never know if there is a car behind a building or not, the AI cant see things outside of its line of sight) | Stochastic  (The actions of the driver depends on the outside environment which is fairly random assuming not all cars are driven by AI’s) | Sequential  (The AI can use previous position and velocity of an other car to predict its location if it gets out of sight in order to avoid crashes.) | Dynamic  (The traffic is a dynamic environment which always changes) | Continious  (driving a car is a continuous thing which can not be divided into discrete states) | Multi  (Other drivers exist which the AI should be aware of) |

2)

For the problem the states will be consisting two separate dictionary objects each keeping track of the group they have inside them and their position as start or end With this notation the state of the starting position is:

{'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

{'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

Start

Possible groups to put in the raft are (actions):

Father + son

Father

Mother + daughter

Mother + Father

Mother

Policeman + thief

Policeman

Policeman + Father

Policeman + Son

Policeman + Daughter

Policeman + Mother

For BFS the program generates 54 nodes and takes 0.114 secs on average. 11 possible actions for each state exists yet because of the illegal states the search alghoritm seems to branch way less on average compared to our total action count of 11. The solution is found on depth 17. From information we can say that max branching factor b=11 and solution depth d = 17. During BFS with some additions to the code I have found out that the tree have branched 39 times which makes the average branching factor 1.38.

For the A\* search I have used the following heuristic:

((people\_left in the starting shore -1)\*2-1)

This heuristic is admissible because each time we take a boat to the end island with 2 people one of them has to return to take the boat to the starting shore which equals to 2 moves. This holds except for the case where there are 2 people on the starting shore, in that case it only takes a single step to cross them across. For example if we say that we have 3 people at the first step two of them pass across after that one of them returns and 2 people remaining in the starting shore come back which ends up being equal to 3 hence the -1. With this heuristic we just assume that there are no illegal shore conditions and anyone can drive the boat. The A\* alghoritm have generated 50 nodes and took an average of 0.108 seconds.

The A\* search have found the same path as the bfs and both have 18 states at total which equals to 17 state swaps. Since the A\* alghoritm have generated less nodes and spent less time it is safe to say that this option is preferable when compared with bfs.

To run the program command line arguments of “bfs” or “a\_star” should be given to the program.

The solution found by the program from end to beginning is:

starting island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

ending island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

ending island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

start

starting island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 1}

ending island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 0}

end

starting island: {'son': 1, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

ending island: {'son': 1, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

start

starting island: {'son': 1, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

ending island: {'son': 1, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 0, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 1, 'policeman': 1, 'thief': 1}

start

starting island: {'son': 2, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

start

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 0, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 1, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 0, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 1, 'policeman': 0, 'thief': 0}

start

starting island: {'son': 2, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

end

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

start

starting island: {'son': 2, 'daughter': 0, 'father': 1, 'mother': 0, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 2, 'father': 0, 'mother': 1, 'policeman': 0, 'thief': 0}

end

starting island: {'son': 2, 'daughter': 1, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 1, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

start

starting island: {'son': 2, 'daughter': 1, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 1, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 0}

ending island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 1}

start

starting island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 0, 'thief': 0}

ending island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 1, 'thief': 1}

end

starting island: {'son': 2, 'daughter': 2, 'father': 1, 'mother': 1, 'policeman': 1, 'thief': 1}

ending island: {'son': 0, 'daughter': 0, 'father': 0, 'mother': 0, 'policeman': 0, 'thief': 0}

start