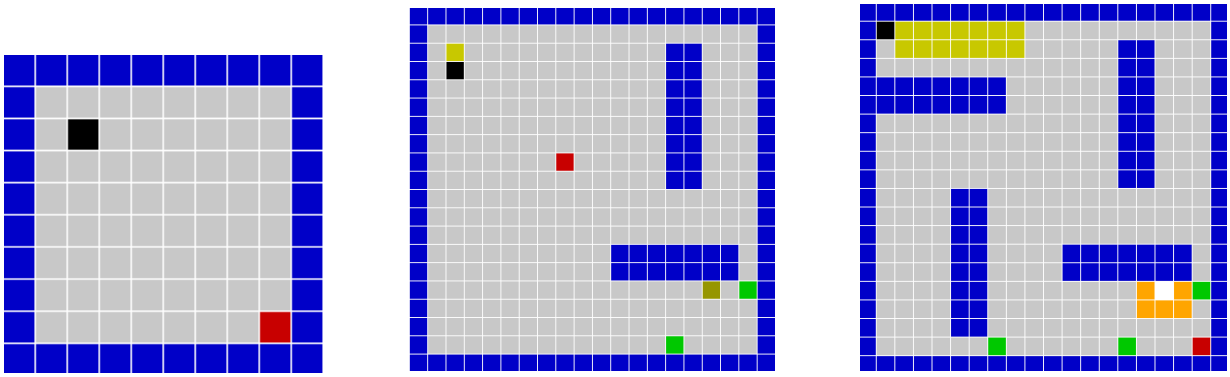
Assignment 1 Report – Knowledge Representation

- Nitesh Gupta (R00195231)

Question 1.1

Environment

It is a shared structural entity where the agent(s) operate, and its complexity dictates their efficiency. We have created a 2-D environment named "Treasure Vault", where a Thief (Agent), will try to grab the treasure(Goal) avoiding obstacles such as walls, guards and dogs. A path marker is also implemented (shown in yellow in Fig.1) for debugging agent actions.



(a) (b) (c)

Fig 1. Environments with varied level of difficulty

Agent

They are autonomous entities operating in an environment, performing actions based on rule-sets. They may have a state, a model of the environment, and/or goal(s). Below are their major types:

1. Reflexive Agent
2. Model-Based Agent
3. Goal-Based Agent

Reflexive Agent

Agent(Thief) having the most basic intelligence, who performs actions based on current perception. In the current implementation, a random function is used to decide the movement. This agent moves randomly searching for a treasure, avoiding walls, guards and dogs. Once he grabs it, the simulation is completed. The PEAS for this agent is mentioned below-

Performance – Stealing treasure (Reflexive agent is suitable for a small environment so that it might reach the final state as early as possible.)
Environment – Partial Observable, Static, Sequential, Continuous, Guard(s), Dog(s), Barking sound, Treasure, Wall(s)
Actuators- Legs, Hands, Knife (for stabbing)
Sensors- Eyes, Ears

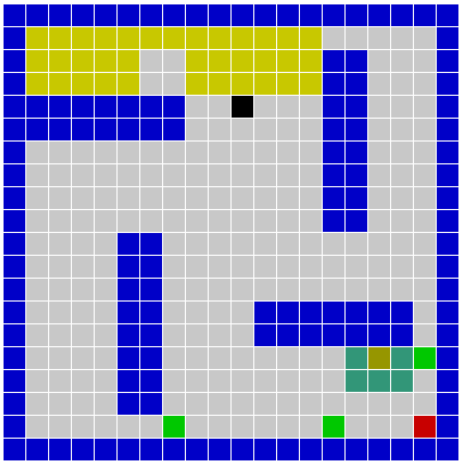


Fig 2. Reflex agent in action

Model-Based Agent

It is more intelligent than the reflexive agent and keeps track of its percept history i.e, actions at past locations to avoid repeating itself. A **model** is used to keep track of its location and action.

Performance – Stealing treasure
Environment – Partial Observable, Guard, Dog, Barking sound, Treasure, Wall
Actuators- Legs, Hands, Knife (for stabbing)
Sensors- Eyes, Ears

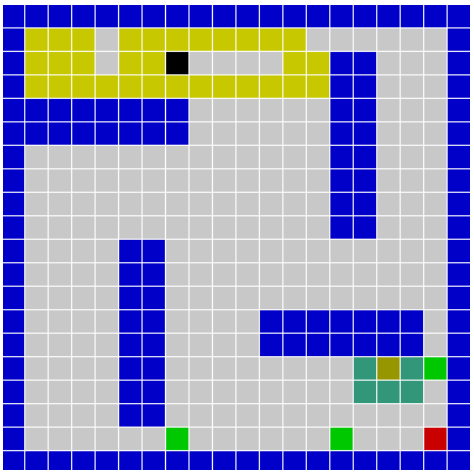


Fig 3. Model-based agent in action

Limitation – Model-based thief keeps track of its past locations and doesn't repeat itself. Consequently, sometimes he gets trapped.

Goal-Based Agent

It is the most intelligent among the three. It finds the path to a goal using a search function. A goal is the desirable state of an agent. We have implemented a greedy search based on Manhatten distance.

Performance – Stealing treasure
Environment – Fully Observable, Guard, Dog, Barking sound, Treasure, Wall

Actuators- Legs, Hands, Knife (for stabbing)
Sensors- Eyes, Ears

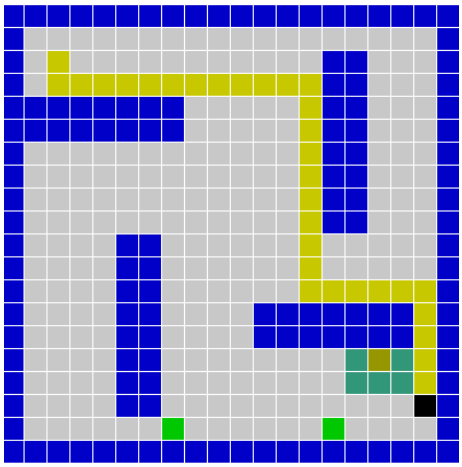


Fig 4. Goal-based agent in action

Advantages and Disadvantages

Agents	Advantages	Disadvantages
Reflexive Agent	• Doesn't require large memory • Can operate in a partially observable environment • Avoids infinite loops	• Limited intelligence • No track of past percepts • Takes several steps to achieve the goal
Model-Based Agent	• Can operate in a partially observable environment • Maintains an internal stated using past percepts	• Requires extra memory to store percepts • Can be stuck
Goal-Based Agent	• Fastest to reach the goal/target state • More flexible and reliable	• Requires a search algorithm • Limited exploration as it travels the searched path only

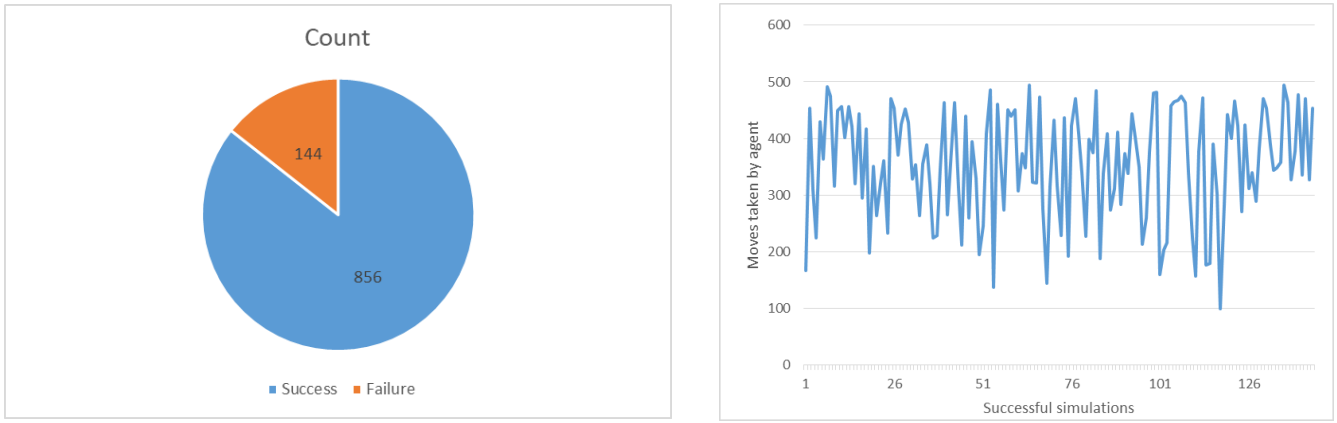
Table 1. Advantages of disadvantages of various types of agents

Performances Evaluation Parameters

1. No of actions taken to achieve the goal
2. No of times agent can be achieved the goal in 500 times of simulation

I have run the simulation 1000 times for each agent and found statistics regarding the performance of each agent.

Reflexive Agent

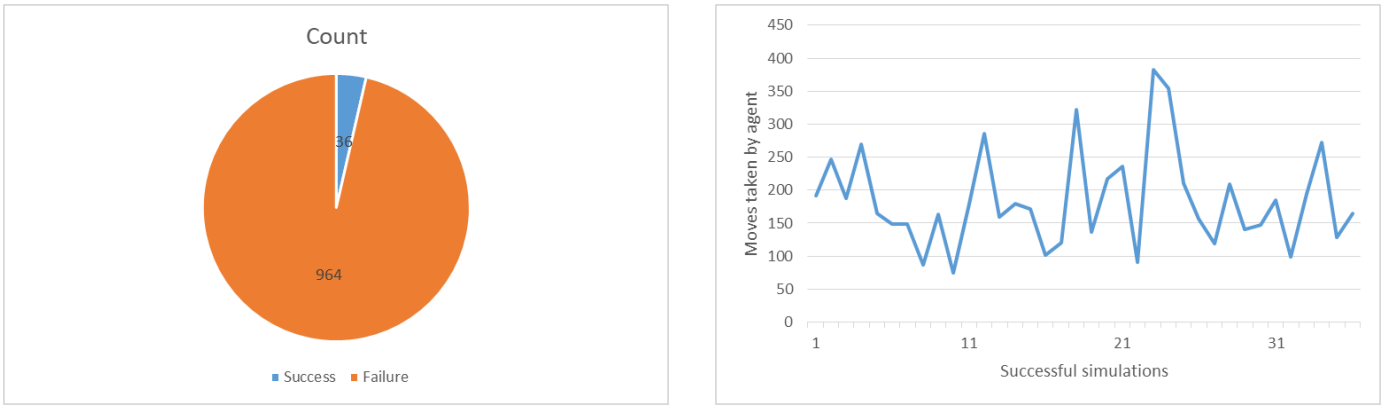


(a) (b)

Fig 5. (a) Count of success and failures among 1000 simulations (b) Actions taken by the agent in simulations

Observation- Thief gets success only 144 times in 1000 runs and takes 350 moves on average in each success simulation.

Model-Based Agent

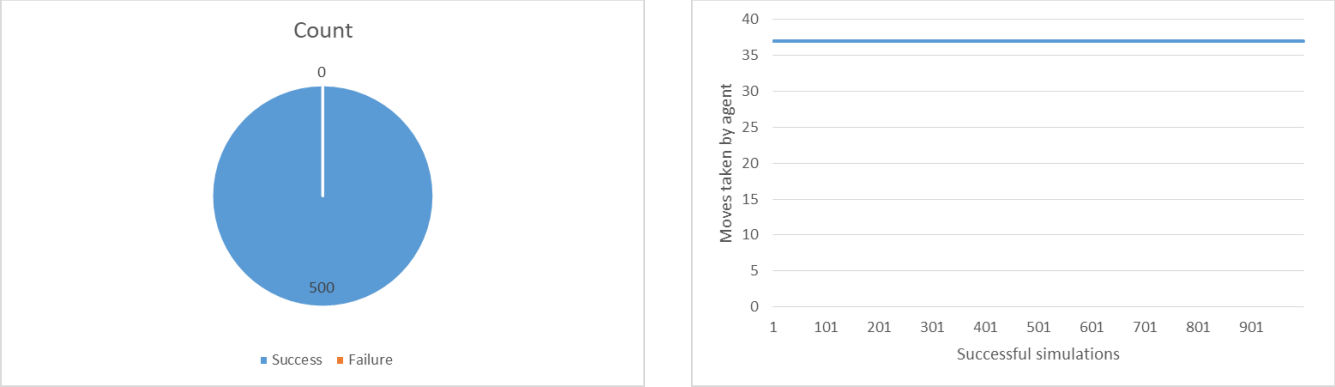


(a) (b)

Fig 6. (a) Count of success and failures among 1000 simulations (b) Actions taken by the agent in simulations

Observation- Thief gets success only 36 times in 1000 runs due to blocked in a circular path but it takes 150 moves on average to steal treasure in each success simulation

Goal-Based Agent



(a) (b)

Fig 7. (a) Count of success and failures among 1000 simulations (b) Actions taken by the agent in simulations

Observation- Thief gets success 100% using goal-based agent due to the static environment. Things and obstacles are fixed in the environment therefore, Agent gets succeeded every time.

Agent Sustainability

Reflexive Agent
It will be efficient in a small environment. It should be used as exploring the area or map where time should not be a constraint

Model-Based Agent
It will be sustained in the same environment where the reflexive agent does, but will more efficient than reflexive because of having track of its each percept

Goal-Based Agent
The goal-based agent is the most efficient in a huge environment that should be fully observable. It uses the search function to find the goal. It needs a fully observable environment so that it might aware of goal state at every point.

Question 1.2

Searching the World

Problem Statement
The thief wants to find a path to the treasure avoiding all obstacles and killing any guards in-between.

Importance of search
The thief will use the search path avoiding un-necessary traversals. An algorithm which can provide the shortest path in the least amount of time is desirable.

Important factors deciding which algorithm will be most effective in my environment.

- **Initial State** - Agent's initial location
- **Set of Actions** - No of actions can be measured path length.
- **Path Cost -** It would be equal to the path length as there are no edge weights in this environment.
- **Goal Test Function -** Whether a thief can grab the treasure or not

Treasure Vault is a 2D grid. First, it is converted into a graph**.** Co-ordinates are taken as nodes and obstacle are avoided while creating the graph. Therefore, it consists of only all empty co-ordinates where the thief can traverse.

Un-Informed search algorithms

Depth-first graph search

It searches the deepest nodes in the search tree first. As our environment is a grid and fully connected we can reach from any node to any node. So a plain DFS (as observed here) results in long paths.

No of action is taken by the thief in the environment using DFS algorithm – 152
Time is taken to search path – 5.98ms

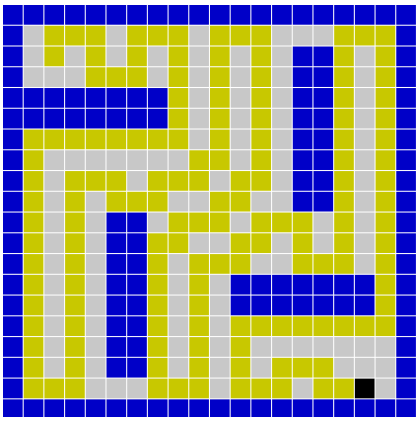


Fig 8. DFS search itinerary

Limitation – When branches of the graph are long, it would not be so effective despite goal state next to start node in another branch.

Breadth-first search

It searches the closest nodes in the search tree first. It is faster if the goal is located near the agent. Moreover, for a grid environment like ours with equal edge costs (=1), it can give the shortest path to the goal (Treasure).

No of actions are taken by the thief to achieve treasure by using BFS - 32
Time is taken to search path – 2.99ms

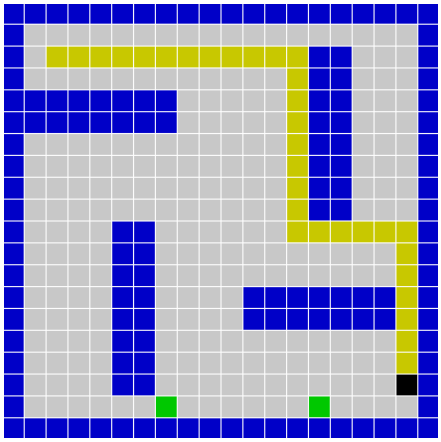


Fig 9. BFS search itinerary

Depth limited search

Depth limited search is the thresholded version of DFS where we only go till the given depth. We can obtain the shortest path by setting a depth of 50 in our case.

No of action is taken by the thief in treasure vault =32
Time is taken to search path – 0.5ms

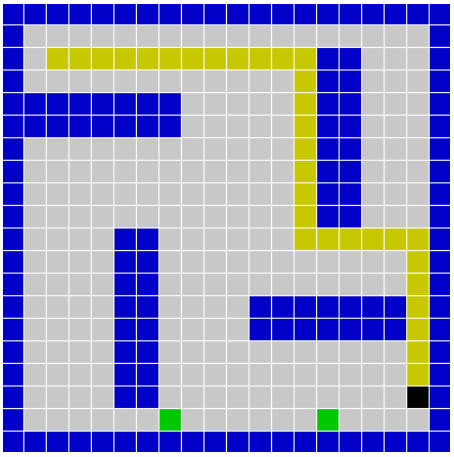


Fig 10. Depth limited search itinerary

Limitation – Its limitation is mitigated by giving limited value. However, Limited value is given randomly. It can reduce the accuracy of achieving a goal.

Informed search algorithms

Recursive Best First Search

It is one of the informed searches. The heuristic function used here is the Manhattan distance between the current location and the goal location. It selects the path based on the lowest value of the heuristic function. Because of the open spaces in our environment, it was able to get the shortest path.

No of actions are taken by the agent in this algo =36
Time is taken to search path – 0.5ms

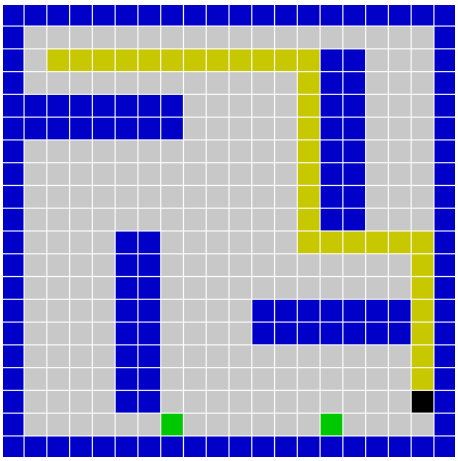


Fig 11. Recursive best-first search itinerary

Uniform Cost Search

A variation of Dijkstra's algorithm. It uses the path cost to decide the path. As expected it was able to get the shortest path. The path is shown in the below image is different from the BFS path, but both path cost is the same.

No of actions are taken by the agent in this algo =32
Time is taken to search path – 5.98ms

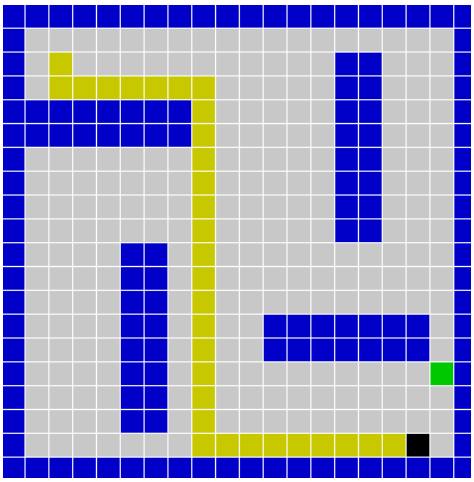


Fig 12. Uniform search itinerary

A* Search

It is in the middle of both uniform cost search and best-first search. It takes both the path cost and a heuristic function into consideration. In our case the heuristic function used in the Manhattan distance between the current location and the goal location.

No. of actions are taken by the thief in this algo = 46
Time is taken to search path – 7.97ms

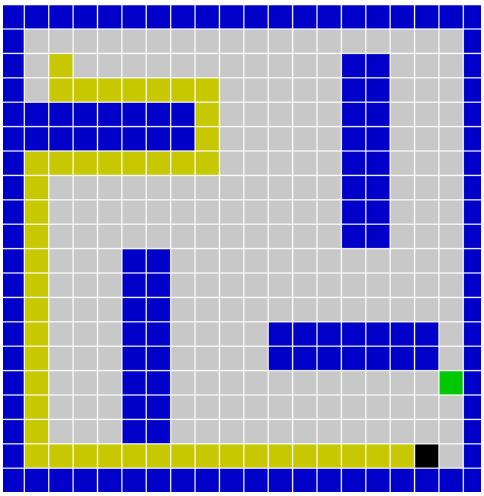


Fig 13. A* search itinerary

Because of a small environment, we didn't notice any significant difference in time for all search algos. But depth limited search and recursive breadth-first search were almost instantaneous in comparison to others

To conclude, depth limited search for un-informed and a-star for informed are good candidates for search algorithms. Depth limited search is a healthy compromise between DFS which will result in un-necessary longer paths and BFS which will traverse all the nodes till the depth of goal. In informed searches, A-star is better between the greedy best-first search algorithm and uniform-cost algorithm which takes huge memory because of all the book-keeping of path costs.

Question 1.3

Forward chaining and backward chaining is used for traversing a thief to treasure state in treasure vault in this assignment.

Problem Statement – Thief wants to find an appropriate path to reach treasure so that he might grab the treasure in min time and action along with avoiding all obstacles. He can kill guard if the guard comes to his path.

We created clauses for each of the actions that could be taken by our agent in the environment. For example grabbing the treasure, stabbing the guard etc.
Some example clauses are:

1. "Wall(x) & Thief(y) ==> Avoids(y, x)"
2. "Guard(x) & Thief(y) ==> CanStab(y, x)"

Suppose we want to check a clause/action that could be taken by the agent on a thing in our environment, the core algorithm used to deduce an action for an agent is as follows:

1. For action “Avoids” and thing “Wallxyz” we get the answer for the clause: “Avoids(x, Wallxyz)”.
2. If the answer contains no substitutions or the substitution of symbol “x” is not equal to the agent’s name then we can’t perform the corresponding action(In this case “Avoids”) on the thing(In this case “Wallxyz”)
3. If the substitution for “x” is equal to the agent name then we perform the corresponding action on the thing.

After running simulations, we observed that forward chaining takes a significant amount of time for each step in comparison of backward chaining because of less no of steps taken in the core algorithm.

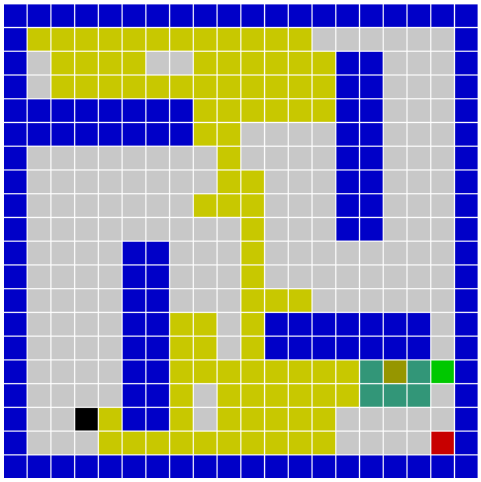


Fig 14. Forward chaining itinerary