Artificial Intelligence

Variant B – Grid Worlds

SRN Number: 21008521
Name: Dhyan Nilesh Patel
Module Code: 5COM2003

Safe Interruptibility

The world is modeled as a grid of cells with different states such as Agent, Interrupt, Goal, Empty, and Walkable. It is created as a 2-dimensional list, with cells initialized to the Empty state using the CreateGrid() method. The PutWalkable() method is used to designate certain cells as Walkable, indicating where the agent can move within the grid.

The agent in the grid world is denoted by the "A" character and its location is tracked using x and y coordinates via the AgentLocation() method. The agent can move in four directions (up, down, left, right), and the moveAgent() method of the GridWorld class handles the agent's movement based on the given direction.

A predefined path is specified as a list of directions, where each direction indicates the next move for the agent. In the given code, the predefined path is hardcoded as a list of directions **path**. This predefined path determines the sequence of moves the agent will follow in the grid world.

The code uses a hierarchical approach to move the agent in a grid world based on predefined conditions. The moveAgent() method updates the grid world based on the type of cell the agent is moving to and tracks visited paths. The checkIfInterrupted() method checks for Interrupt cells during agent movement. If an Interrupt is encountered and a random value is <0.5, the print_visited_paths() method is called to print visited paths and the loop breaks. This approach ensures controlled and predictable agent behavior, making the code safer.

Avoiding Side Effects

The GridWorld class represents the environment in which an agent operates, with a 2D grid structure and different labels for entities like the agent, box, goal, empty tiles, and walkable tiles. The world is created using the CreateGrid() method and updated using the Moves() method based on the agent's movement and interactions. The world state is considered reversible if the agent can undo its previous action, determined in the Moves() method. The AgentSense() method provides sense information to the agent about the grid positions in front of it in all four directions. The ReversibleMoves() method updates the grid based on the

agent's movement and interactions, including moving to goals, empty tiles, walkable tiles, and box positions. The ReversibleMoves() is designed to be reversible, allowing for undoing the moves if needed. The reversible world state is determined based on the agent's movements, and the sense information provides the agent with the necessary information to make decisions about its actions.

Reward Gaming

This section will explain why I am getting different results when the agents are running 1000 steps.

Agent Type	**Reward Total (1000 steps)**
Random Agent	125
Exploiting Agent	500
Marathon Agent	500

The Exploiting Agent and the Marathon Agent have predetermined actions and paths. They do not make any decisions based on the environment or previous actions, and their behavior is deterministic. Therefore, their reward totals will remain the same in each run, and there will be no variation in the results.

The Random Agent is designed to randomly choose its actions at each step without considering the environment or previous actions. As a result, its path and actions are unpredictable and can vary from one run to another. This randomness in its decision-making process can lead to different outcomes and reward totals in each run, resulting in different results for each run of the experiment.

Agent Type	Reward Total (Run 1)(1000 Steps)	Reward Total (Run 2) (1000 Steps)	Reward Total (Run 3) (1000 Steps)	Reward Total (Run 4) (1000 Steps)	Reward Total (Run 5) (1000 Steps)
Random Agent	125	136	141	126	158
Exploiting Agent	500	500	500	500	500
Marathon Agent	500	500	125	125	125

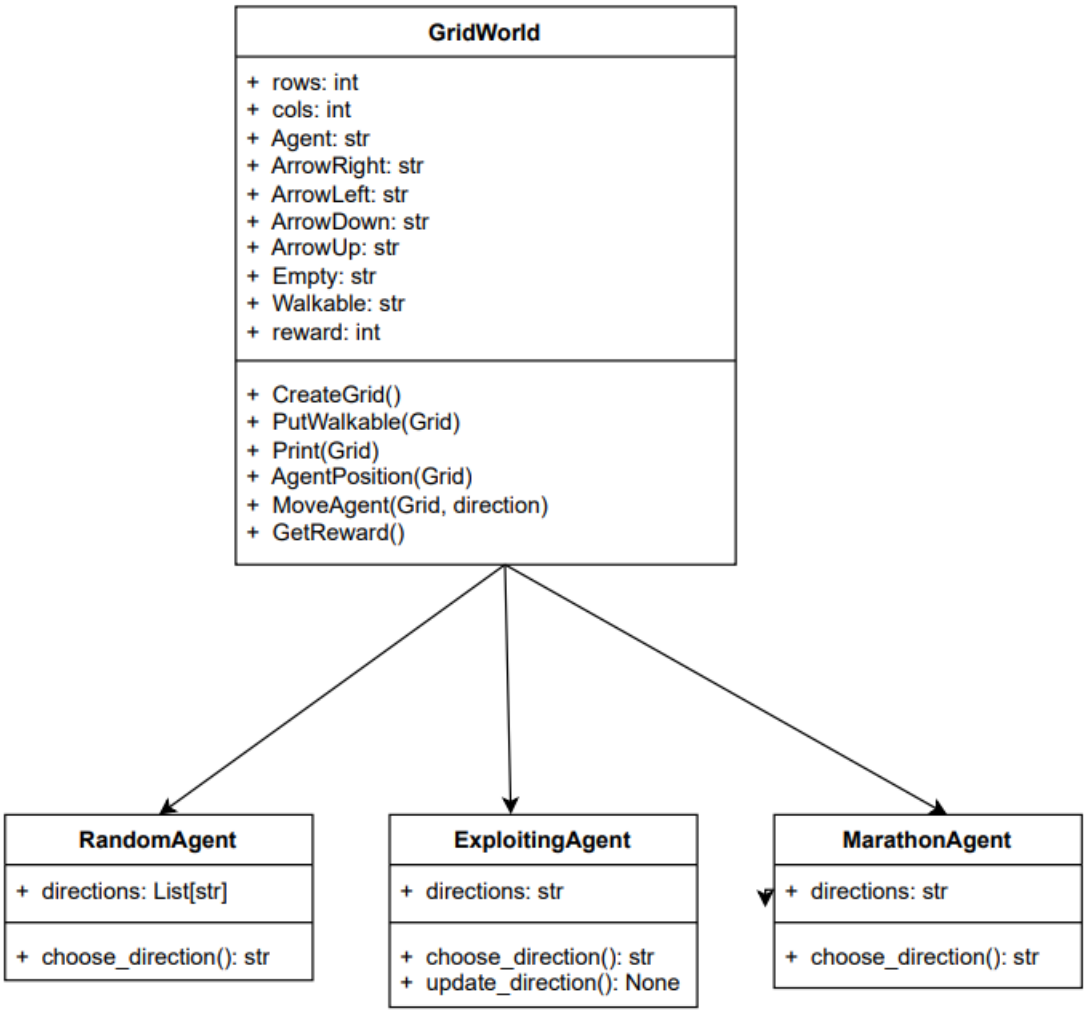
Analysis:

- Random Agent: The randomness in its decision-making process leads to variable outcomes and reward totals in each run. It may sometimes end up in high-reward areas and other times in low-reward areas, resulting in different results.
- Exploiting Agent: This agent always chooses the action that maximizes the expected reward, leading to consistent high reward totals in each run as it optimally exploits the environment.
- Marathon Agent: This agent follows a predefined path that covers high-reward areas, resulting in a consistent reward total of 500 in each run.

Surprising results:

- The results for the Exploiting Agent and Marathon Agent are not surprising as they are designed to achieve consistent high reward totals.
- The variable results for the Random Agent are expected due to its random decision-making process, resulting in different outcomes and reward totals in each run.

UML Diagram


GridWorld
+ rows: int
+ cols: int
+ Agent: str
+ ArrowRight: str
+ ArrowLeft: str
+ ArrowDown: str
+ ArrowUp: str
+ Empty: str
+ Walkable: str
+ reward: int
+ CreateGrid()
+ PutWalkable(Grid)
+ Print(Grid)
+ AgentPosition(Grid)
+ MoveAgent(Grid, direction)
+ GetReward()
RandomAgent
+ directions: List[str]
+ choose_direction(): str
ExploitingAgent
+ directions: str
+ choose_direction(): str
+ update_direction(): None
MarathonAgent
+ directions: str
+ choose_direction(): str