CSC 584: Building Game AI
Homework 4 Report

Part 1: Decision Trees

In designing a decision-making system for controlling movement and pathfinding behaviors in a game or simulation environment, several critical variables need to be considered. These variables form the basis for building a decision tree model that guides the behavior of characters or entities within the environment. Let's delve into the key variables and decision-making logic required for such a system.

Key Variables:

1. **Character's Current Position:** This variable represents the x, y coordinates of the character within the game world. It is essential for determining distances to other points of interest, such as the wander point, room walls, and room center.
2. **Start Time or Clock of the Character:** This variable tracks the elapsed time since the character's initiation or the start of a specific event. It is crucial for time-based decision-making, such as triggering behaviors after a certain duration.
3. **Current Room of the Character:** Knowing the room or area where the character is located provides context for decision-making. It enables actions specific to the current environment, such as navigating within rooms or transitioning between them.
4. **Position of Walls in the Room:** The coordinates of walls or obstacles within a room are vital for collision detection and ensuring that the character navigates without intersecting with obstacles.
5. **Center of the Room:** Calculating the center of a room allows for behaviors such as moving towards the center for specific objectives or when avoiding walls or edges.

Decision-Making Logic:

Now, let's outline the decision-making logic based on these variables:

1. **Initial Wander Point Setup:** Set the initial wander point to the character's current position.
2. **Distance Check for Wander Point:** If the wander point is within a distance of 3 units from the character's position, Proceed to the next step; Else, Move the character towards the last known wander point.
3. **Time-Based Decision:** Check if it has been 5 seconds since the start of the character's clock, If yes; Proceed to wall proximity check, If no; Move the character to any other room apart from the current room.

4. **Wall Proximity Check:** Determine if the character is near the walls of the room (within a distance of 20 units to the corners of the room), If yes; Move the character to the center of the room, If no; Generate a random point within the same room and set it as the wander point.

This decision tree encapsulates the sequential decision-making process for controlling character movement and pathfinding behaviors within the game environment. Each branch of the tree represents a condition based on the key variables, leading to specific actions or transitions. By carefully designing and parameterizing these variables, developers can create dynamic and responsive AI systems that enhance gameplay experiences.

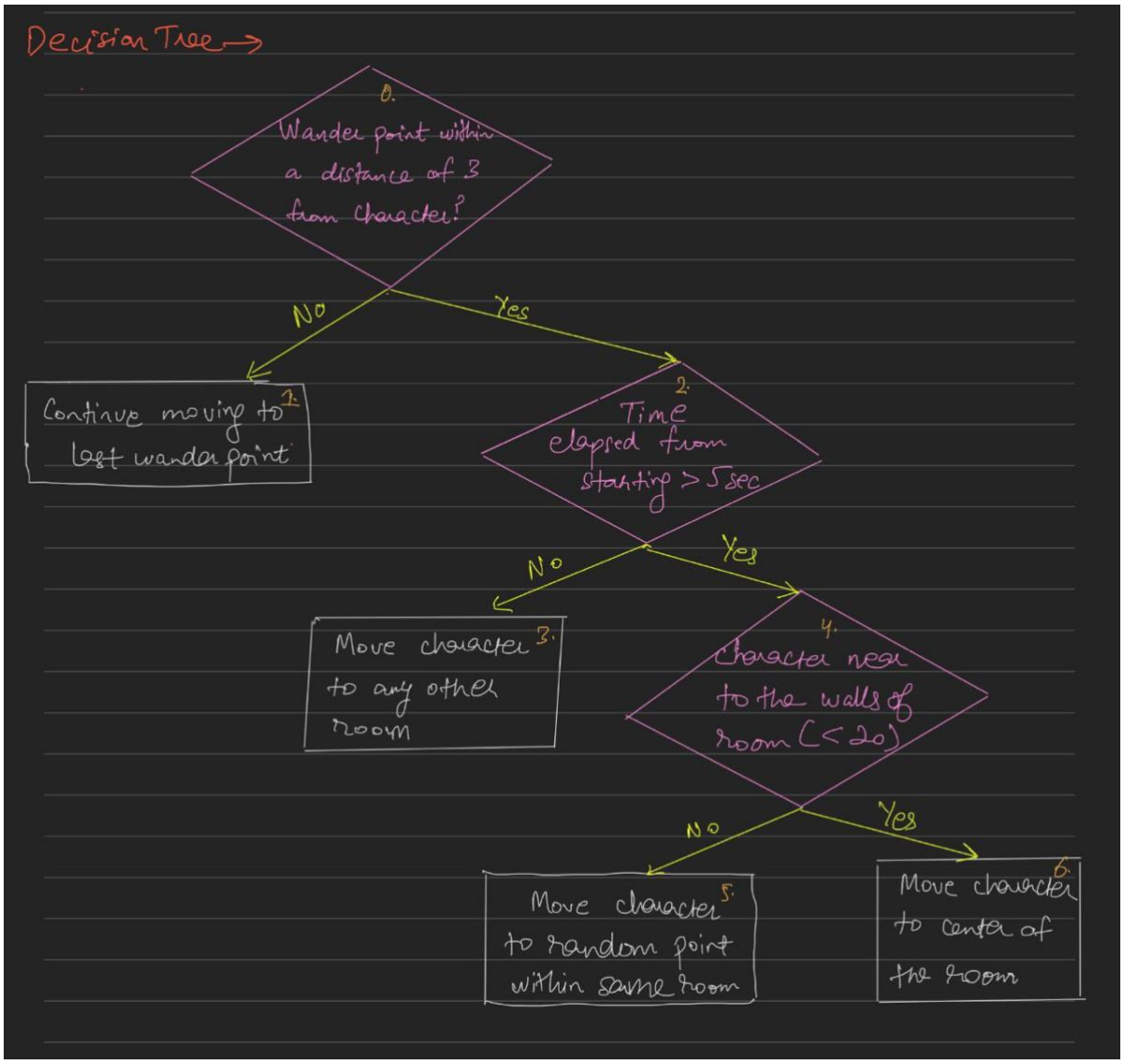


Figure 1. Decision Tree for Character (Part 1)

Part 2: Behavior Trees

Implementing Behavior Tree for Monster Interaction:

In developing the behavior tree for the monster in my environment, I structured it using several composite nodes to encode different behaviors and decision-making logic. The goal was to create a dynamic and engaging interaction between the monster and the character, including chasing, dancing, and wandering behaviors. Let's delve into the details of the behavior tree and its implementation.

Behavior Tree Overview:

At the root of the behavior tree is a selector node, which prioritizes its child nodes based on their conditions and outcomes. The selector node has three main sequences as its children, each representing a distinct behavior pattern for the monster.

1. **Chase Sequence (*chaseSeq*):**

- This sequence is responsible for the monster chasing the character.
- It comprises two nodes:
  - *IsSpriteinNeighbouringOrSameRoom*: A condition node that checks if the monster is in the same room as the character or in a neighboring room.
  - *ChaseCheck* **(Decorated *FollowNode*):** This node initiates the chase behavior if the condition from the previous node is met.

2. **Dance Sequence (*danceSeq*):**
- Solely dedicated to the dance behavior of the monster.
- It contains the *DanceActionNode*: A custom action node that checks the time elapsed since the start of dancing. If within the dance duration limit, it continues dancing; otherwise, it returns *FAILURE*.

3. **Wander Sequence *(wanderSeq*):**
- Handles the wandering behavior of the monster within the environment.
- Comprises two nodes:
  - *WallHitCheck* **(Decorated *WanderNode*):** Checks if the monster is near the walls of the room. If so, it prevents the monster from hitting walls during wandering.
  - *MoveToCenterNode*: Moves the monster towards the center of the room.

Authoring Challenges and Insights:

Authoring the behavior tree posed significant challenges initially, especially in determining the appropriate composite nodes and their interconnections. At first, I attempted to create a highly complex tree structure, which led to code complexity and confusion.

However, through iterative refinement and practical consideration, I realized the importance of simplicity and clarity in behavior tree design. It's crucial to strike a balance between the number of nodes and the logic encapsulated within them. Overcomplicating the tree structure can lead to unnecessary complexity and potential erratic behavior.

One key learning was to think practically about the required behaviors and conditions, avoiding excessive node proliferation. Instead, I focused on leveraging existing nodes effectively and incorporating logical checks within them where possible.

Observations on Monster Behavior:

The implemented behavior tree resulted in the monster behaving as expected, exhibiting chasing behavior when near the character, dancing for a limited duration, and wandering within the environment. However, tweaking the sequence of nodes attached to the selector node can sometimes lead to slight variations in behavior, such as prolonged stays at room centers.

In conclusion, the behavior tree approach effectively captures the diverse behaviors and decision-making processes of the monster, providing an interactive and engaging experience within the game environment. Continual refinement and adjustment of the behavior tree can further enhance the monster's behavior and overall gameplay dynamics.

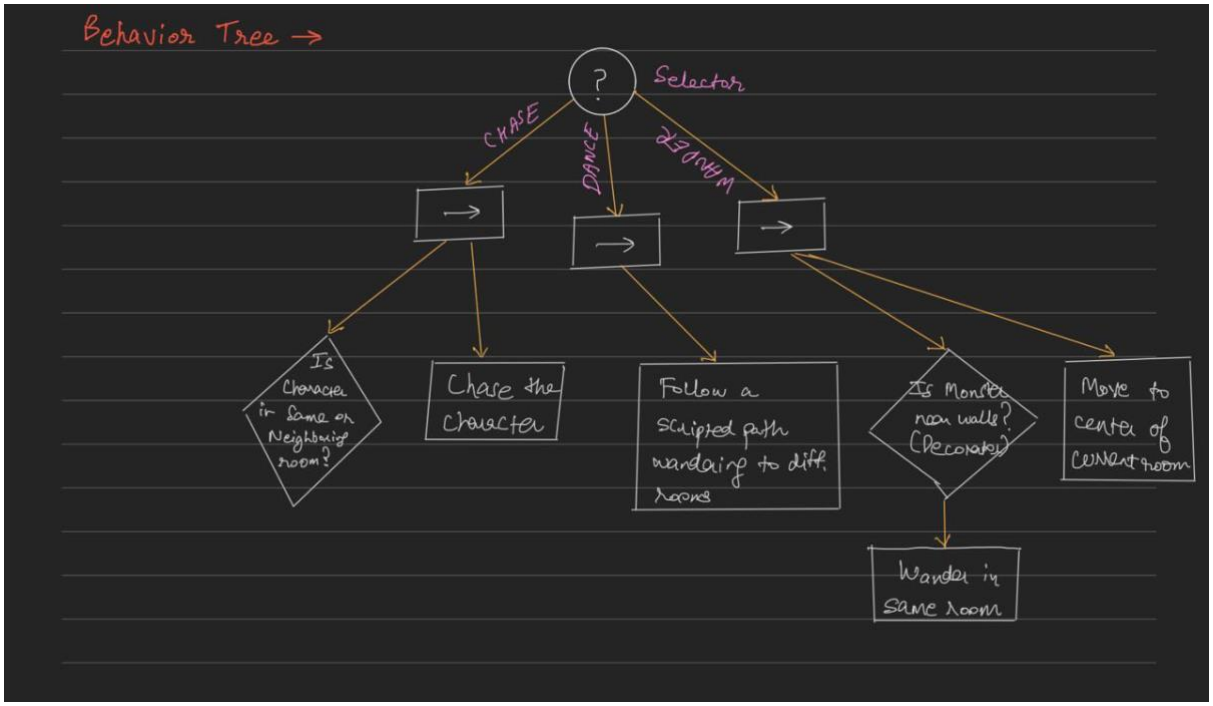


Figure 2. Behavior Tree for Monster (Part 2)

Part 3: Decision Tree Learning

In the final phase of the assignment, the objective was to replicate the behavior of the behavior tree-controlled monster using decision tree learning. This process involved parameterizing the state space with attributes such as room numbers, proximity to obstacles, and the current actions being taken. Here's a detailed overview of the implementation and the comparison between the behavior-tree-controlled monster and the learned-decision-tree-controlled monster.

Parameterization of State Space:

The attributes chosen for parameterizing the state space included:

1. Room number of the monster.
2. Room number of the character.
3. Binary indicator of whether the character and monster are in the same room.
4. Distance of the monster from the character.
5. Distance of the monster from the center of the current room.
6. Whether the character was killed from the current action.
7. Action choice (target variable).

Data Collection and Decision Tree Learning:

To collect data, I performed stratified sampling based on millions of data points at each time stamp, resulting in a dataset of 10,000 data points (final_dataset.csv). The ID3 decision tree learning algorithm was then implemented to construct a decision tree based on information gain, with a maximum of 6 leaf nodes to prevent overfitting.

Attributes Considered in the Decision Tree:

The final decision tree focused on two crucial attributes for determining the monster's action:

1. Distance of the monster from the center of the room.
2. Binary indicator of whether the character and monster are in the same room.

Comparison and Performance Evaluation:

The comparison between the behavior-tree-controlled monster and the learned-decision-tree-controlled monster revealed several insights:

- **Effectiveness in Eating the Character:** The decision tree-learned monster was more effective in chasing and eating the character compared to the behavior tree-controlled monster. This effectiveness stemmed from the decision tree's ability to learn optimal actions based on the current state attributes.
- **Frequency of Chasing Action:** The decision tree-learned monster predominantly chose the action of chasing or pathfinding the character, aligning well with the desired behavior. This frequency indicates that the decision tree successfully captured the essence of the behavior tree's logic.
- **Qualitative Differences:** While both monsters exhibited similar qualitative behavior, the decision tree-learned monster demonstrated a more focused and efficient approach in chasing and capturing the character. It showcased a clear decision-making process based on the learned rules, leading to improved performance.

Quantifying Performance:

To quantify performance, metrics such as the success rate of eating the character, average time taken to catch the character, and frequency of optimal actions (e.g., chasing) can be measured and compared between the behavior tree-controlled and learned-decision-tree-

controlled monsters. These metrics provide a quantitative assessment of the effectiveness and efficiency of each approach.

Conclusion:

In conclusion, the decision tree learning approach successfully replicated and even enhanced the behavior of the behavior tree-controlled monster. By learning from data and optimizing action choices based on current state attributes, the decision tree-learned monster demonstrated improved performance and effectiveness in accomplishing its goal of chasing and capturing the character. This experiment highlights the power of machine learning algorithms in encoding complex behaviors and decision-making processes in interactive environments.

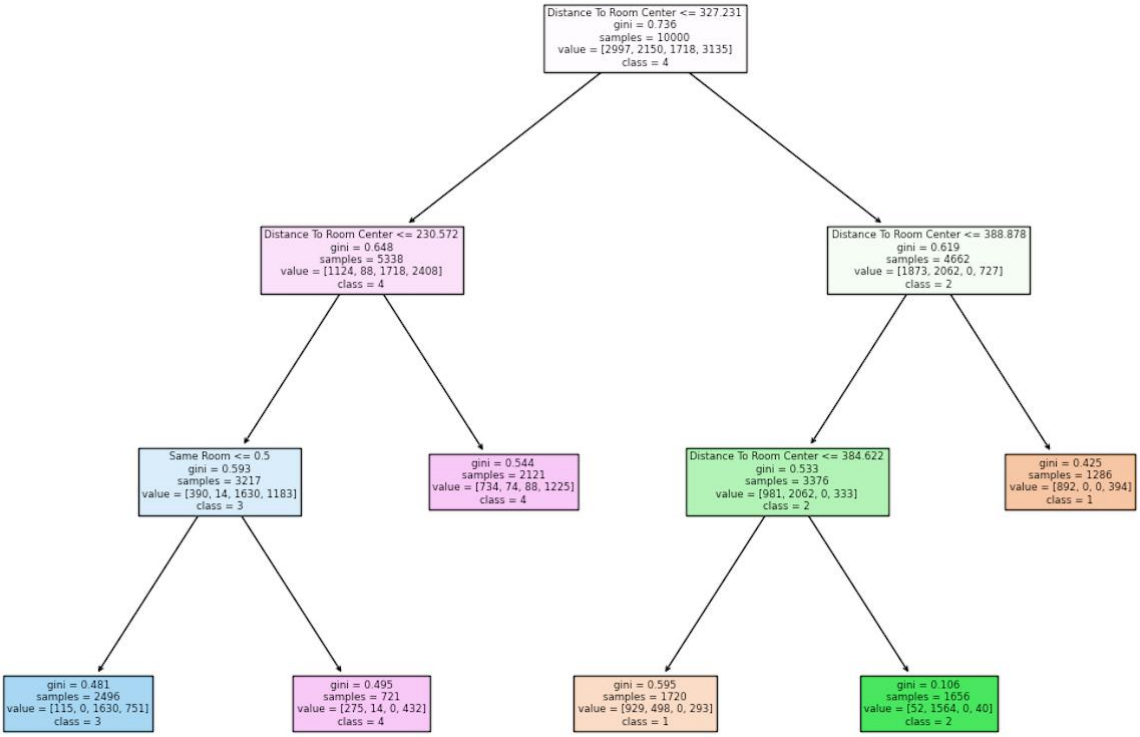
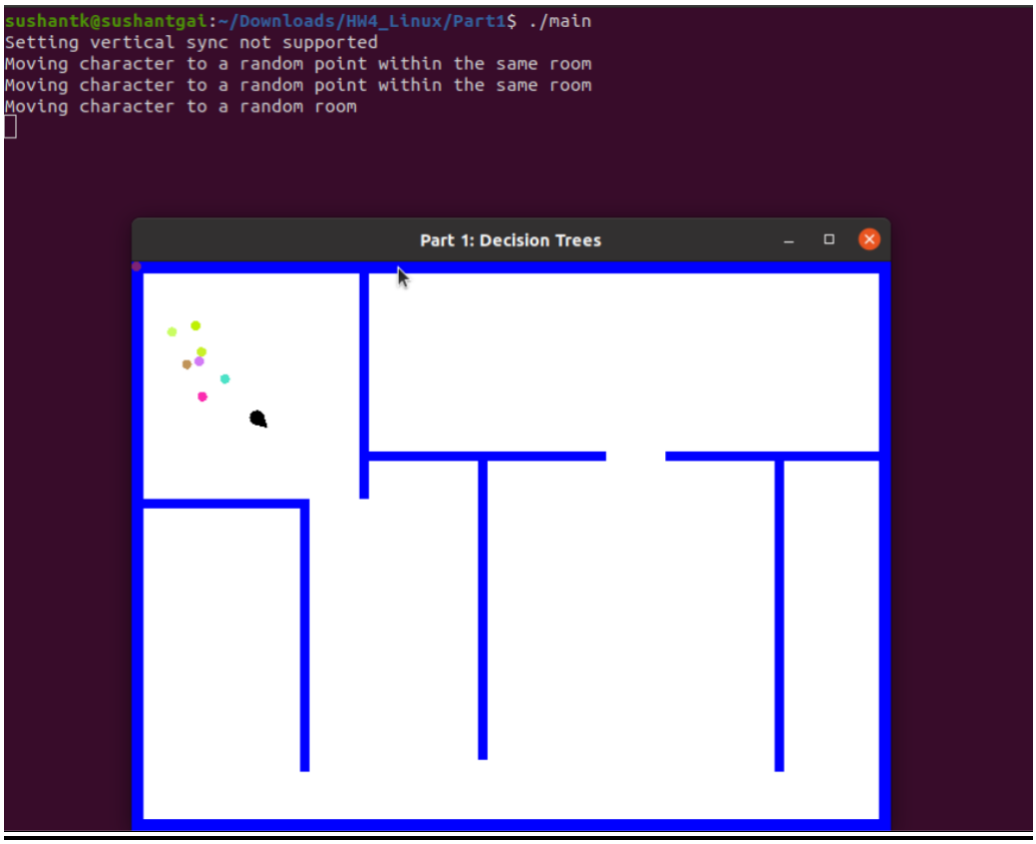
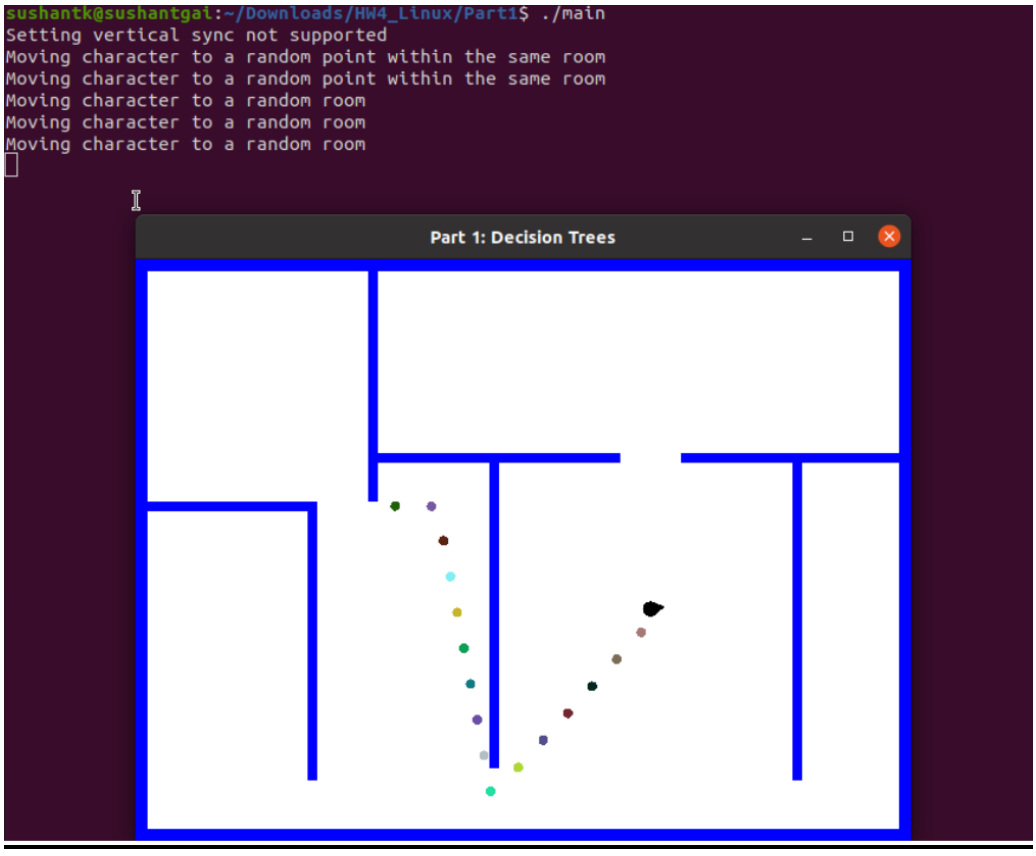


Figure 3. Learned Decision Tree for Monster (Part 3)

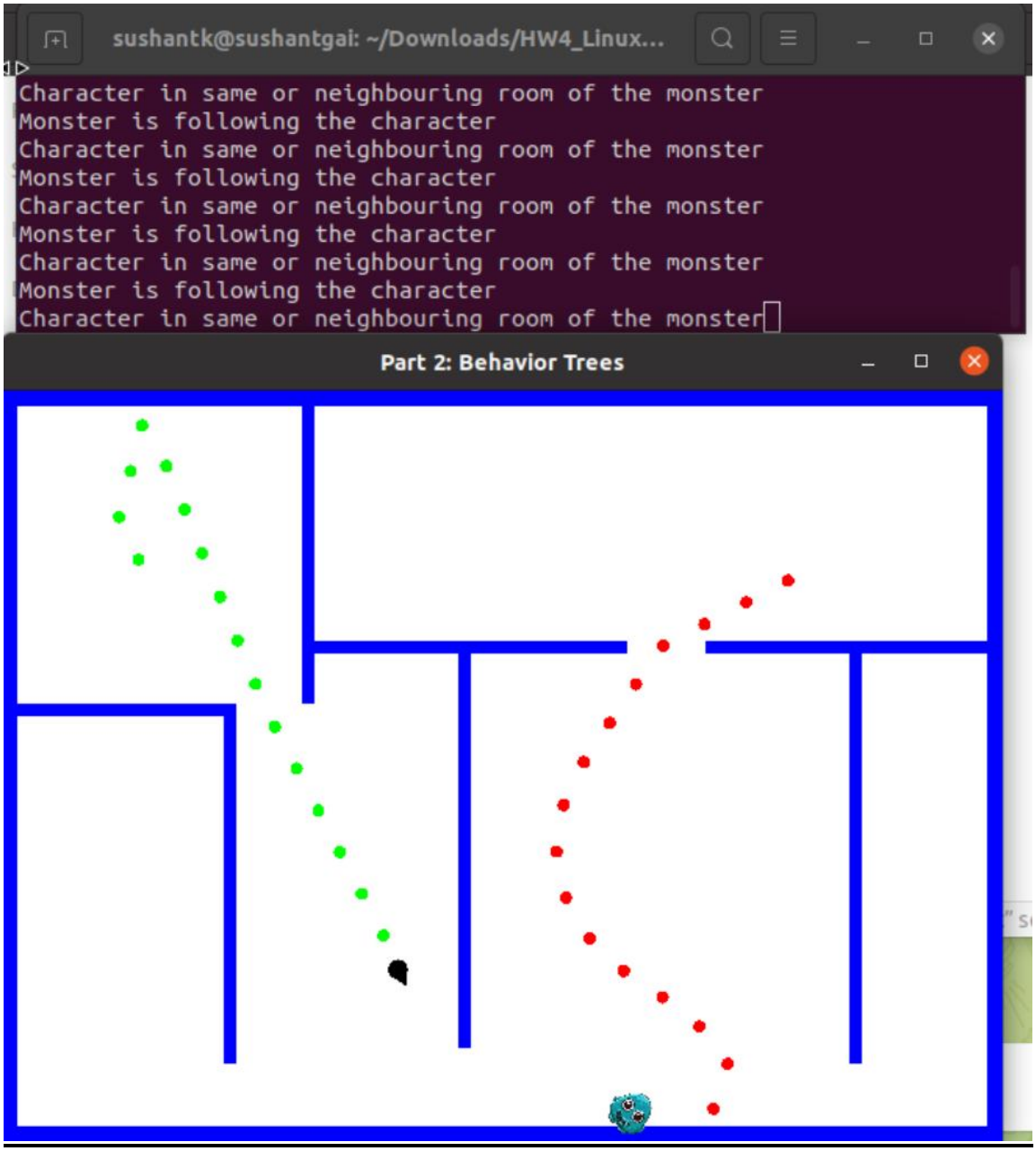
Appendix

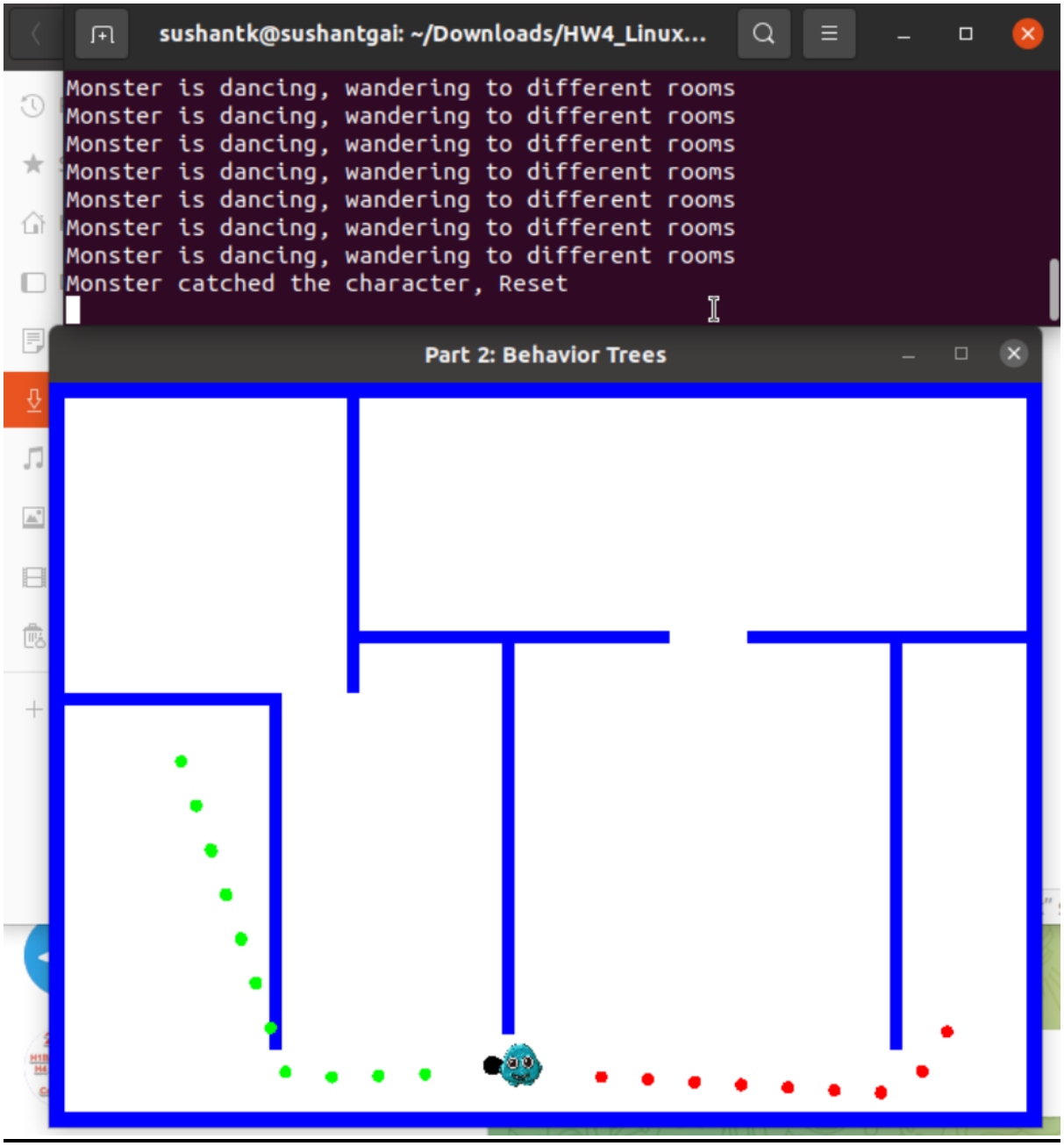
Screenshots for Part 1

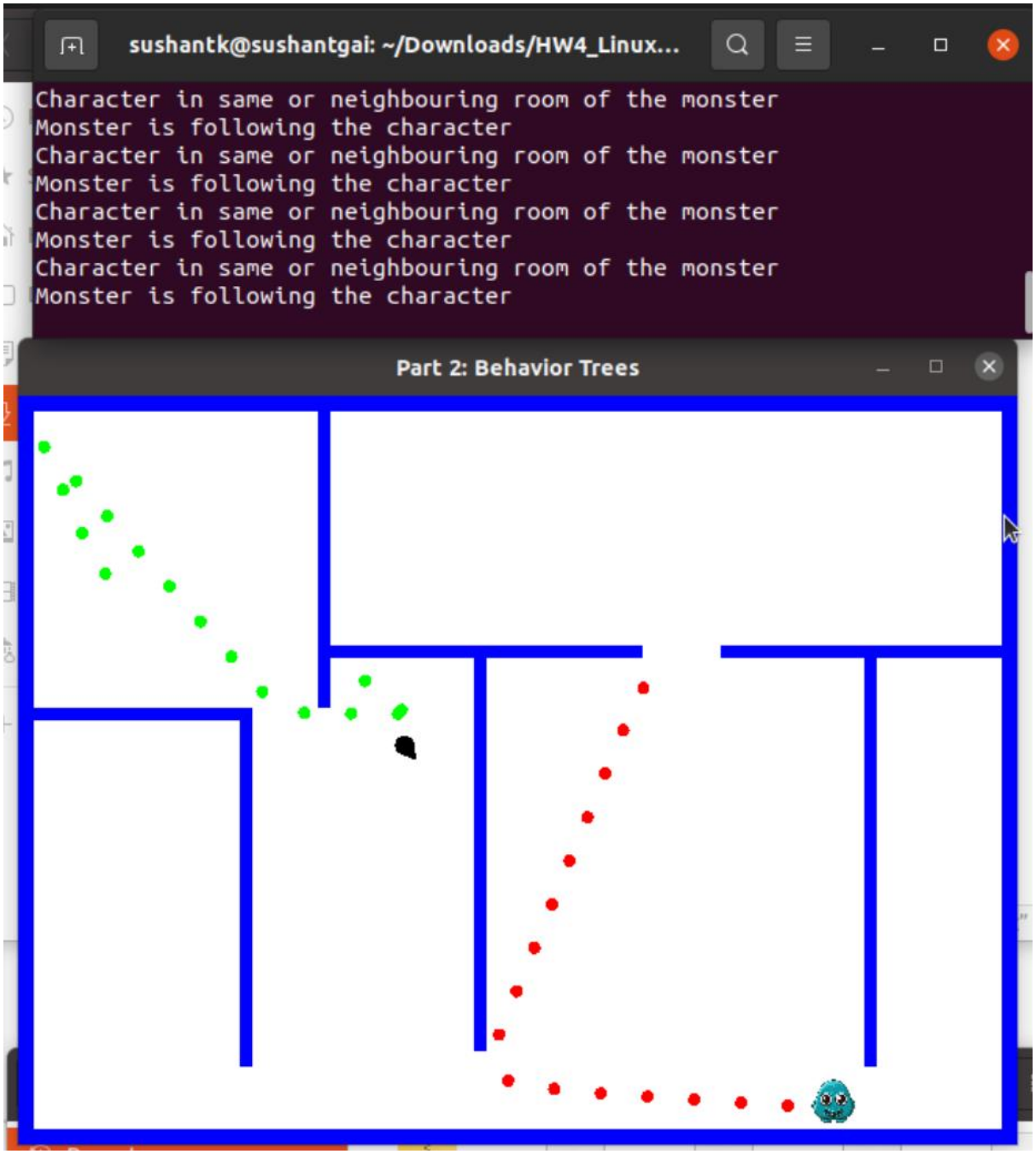




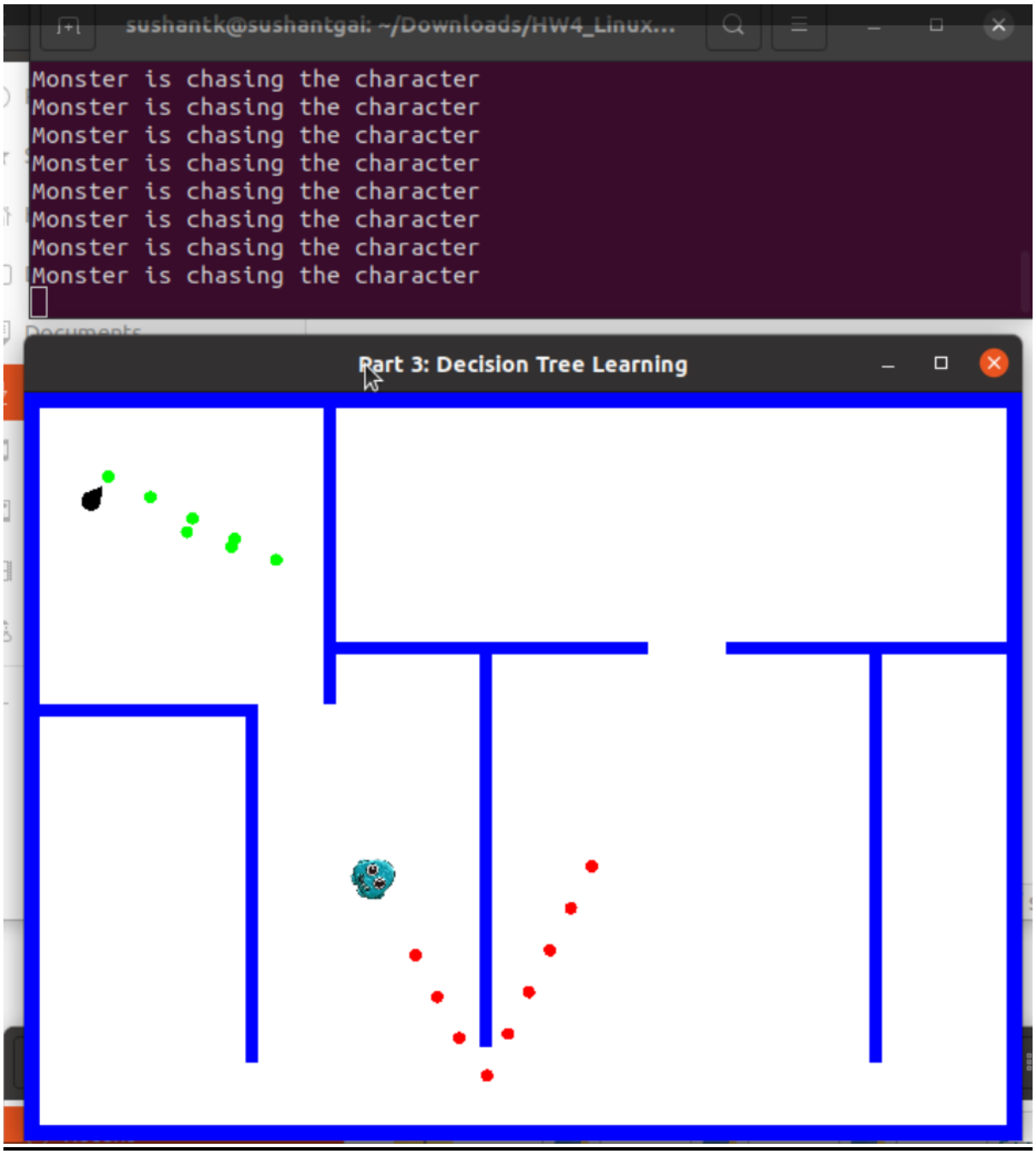
Screenshots for Part 2

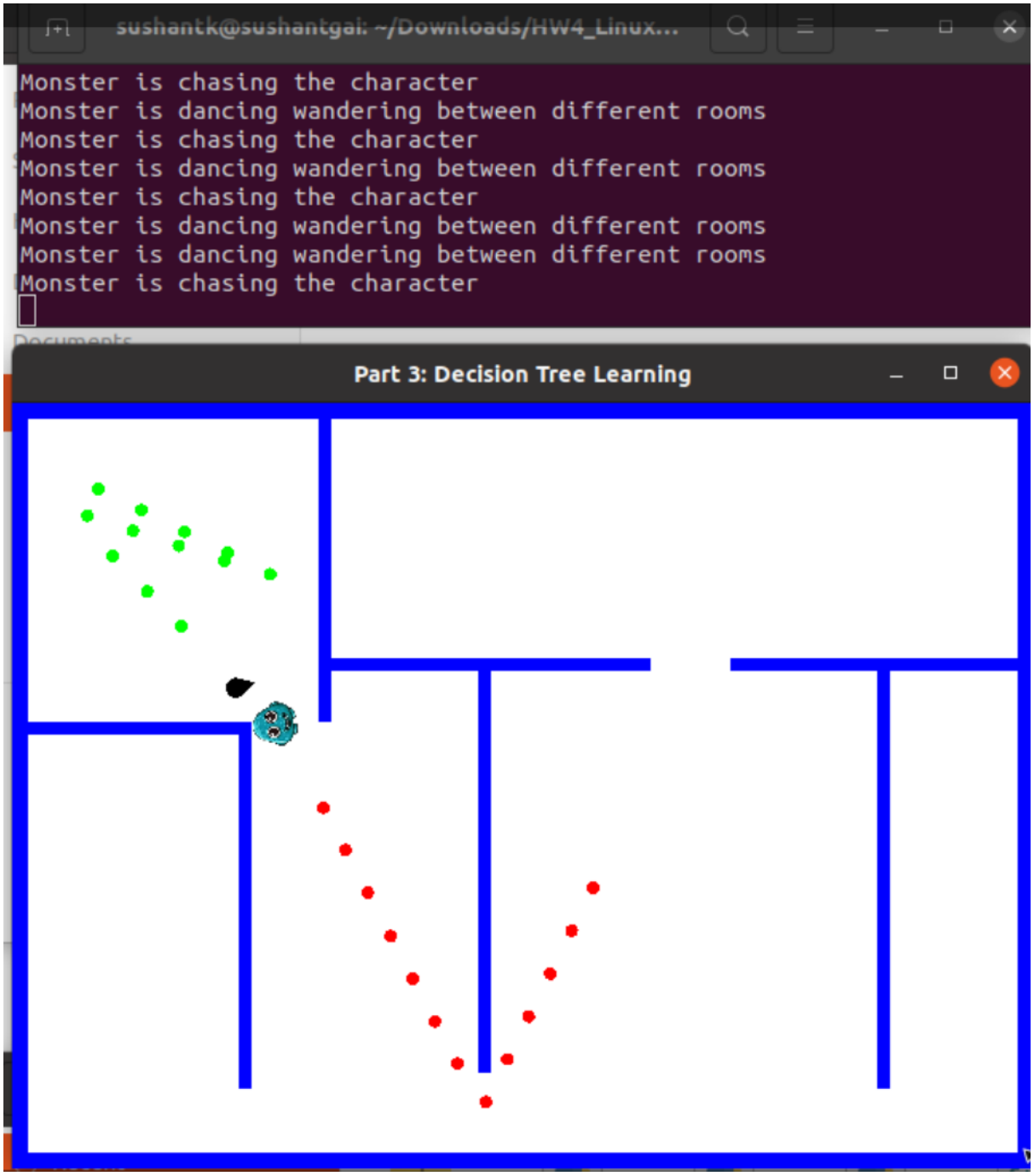


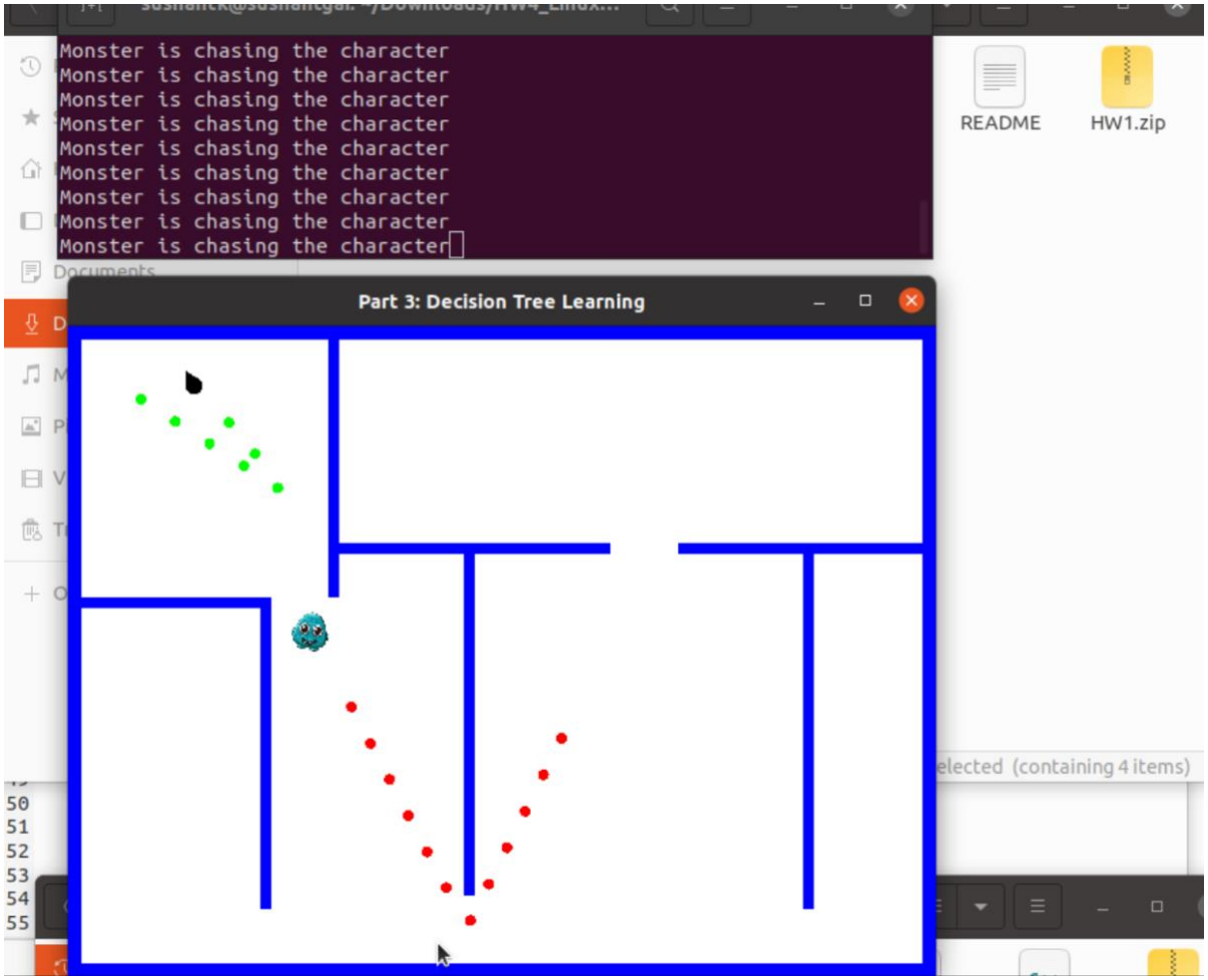

sushantk@sushantgai: ~/Downloads/HW4_Linux...
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster is dancing, wandering to different rooms
Monster catched the character, Reset
Part 2: Behavior Trees


sushantk@sushantgai: ~/Downloads/HW4_Linux...
Character in same or neighbouring room of the monster
Monster is following the character
Character in same or neighbouring room of the monster
Monster is following the character
Character in same or neighbouring room of the monster
Monster is following the character
Character in same or neighbouring room of the monster
Monster is following the character
Part 2: Behavior Trees

Screenshots for Part 3




sushantk@sushantgai: ~/Downloads/HW4_Linux...
Monster is chasing the character
Monster is dancing wandering between different rooms
Monster is chasing the character
Monster is dancing wandering between different rooms
Monster is chasing the character
Monster is dancing wandering between different rooms
Monster is dancing wandering between different rooms
Monster is chasing the character
Part 3: Decision Tree Learning


Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
Monster is chasing the character
README
HW1.zip
Part 3: Decision Tree Learning
elected (containing 4 items)
50
51
52
53
54
55