CSC 584/484 - Building Game AI
Miles Hollifield
April 20, 2025

CSC 584/484 - Building Game AI: Homework 4 Writeup

Introduction

The purpose of this assignment was to explore higher-level decision-making techniques essential for game AI, building upon the movement and pathfinding foundations established in previous assignments. I implemented three key AI decision-making technologies: decision trees, behavior trees, and decision tree learning. By combining these techniques with the previously developed steering behaviors and pathfinding algorithms, I created a complete AI system capable of making strategic decisions and executing complex behaviors.

The primary goal was to analyze how different decision-making approaches affect agent behavior and to investigate how machine learning can be used to replicate behaviors originally defined by hand-crafted decision logic. This project involved creating an indoor environment with multiple rooms and obstacles, implementing a character controlled by a decision tree, and creating "monster" agents that chase the character using different AI control methods.

By examining the behavior and effectiveness of these decision-making techniques, I gained valuable insight into their practical applications and limitations for game AI development.

Implementation and Analysis

Part 1: Decision Trees

For the decision tree component, I implemented a flexible decision tree system that allows for the creation of complex decision-making logic. The implementation includes several node types:

1. **Decision Branch Nodes** - Binary decisions based on boolean conditions
2. **Action Nodes** - Leaf nodes that specify actions to take
3. **Random Decision Nodes** - Nodes that make weighted random selections between options
4. **Priority Nodes** - Nodes that select the first child whose condition evaluates to true

The core of the decision tree implementation is the **DecisionTree** class, which manages the decision-making process, and the **EnvironmentState** class, which tracks various environmental conditions for making informed decisions.

The environmental state parameters I chose to track include:

1. **Distance to obstacles** - Detecting when the agent is near walls or other obstacles
2. **Current speed** - Determining if the agent is moving fast enough to require special handling
3. **Current room location** - Tracking which room the agent is currently in
4. **Distance to targets** - Measuring proximity to points of interest
5. **Time in current state** - Tracking how long the agent has been in its current state
6. **Path status** - Monitoring if a path is completed, blocked, or in progress
7. **Line of sight** - Checking if the agent can see important targets
8. **Movement direction** - Determining if the agent is moving toward specific locations

These parameters provide a rich representation of the agent's situation, allowing for context-aware decision making. The agent can respond appropriately to obstacles, change targets when stuck, and adjust its behavior based on proximity to objectives.

I implemented a decision tree for the autonomous player character that controls target selection and movement behaviors. This tree makes decisions about:

- When to change target locations (for example, when stuck or idle for too long)
- Which room or location to visit next based on current position
- Whether to flee from nearby obstacles
- When to dance
- When to wander to random locations

The decision tree for the character follows this general structure, as depicted in **[Screenshot #1]**. The target selection process is further refined based on which room the character is currently in, creating a more strategic movement pattern where the character naturally moves between different areas of the environment rather than moving randomly.

This structure allows the character to react to the environment while maintaining goal-directed behavior. The tree makes decisions about changing targets based on environmental conditions, creating more dynamic and responsive movement.

Part 2: Behavior Trees

For the behavior tree component, I implemented a comprehensive behavior tree system with the following node types:

1. **Sequence Nodes** - Execute children in order until one fails (AND logic)
2. **Selector Nodes** - Try children in order until one succeeds (OR logic)
3. **Decorator Nodes** - Modify the behavior of their child nodes:
    - **Inverter** - Inverts the result of its child
    - **Repeater** - Repeats its child a specified number of times
4. **Random Selector** - Randomly selects a child to execute
5. **Parallel Nodes** - Execute all children simultaneously

The behavior tree implementation uses a clear status system with SUCCESS, FAILURE, and RUNNING states to track the execution state of each node. This allows for complex behaviors that can run over multiple frames and maintain state between executions.

For the monster agent, I created a behavior tree with the following main components:

1. **Chase Sequence**
    - Check if the player is visible
    - Pathfind to the player
    - Follow the calculated path
2. **Dance Sequence**
    - Check if it's time to dance (random chance with cooldown)
    - Perform a dance sequence with multiple phases (turns in four cardinal directions)
3. **Flee Sequence**
    - Check if near obstacles while moving fast
    - Move away from detected obstacles
4. **Wander Behavior**
    - Used as a fallback when no other behavior is active

The root of the behavior tree is a selector that prioritizes these behaviors in the following order **[Screenshot #2]**:

1. Flee from obstacles (highest priority)
2. Chase the player if visible
3. Occasionally dance
4. Wander as a fallback

This behavior tree creates an interesting and dynamic monster that intelligently chases the player while avoiding obstacles. The dance behavior adds unpredictability and character to the monster's behavior, making it feel more alive and less mechanical **[Screenshot #3]**.

The implementation of the **BehaviorTree** class and its nodes provides a robust foundation for creating complex AI behaviors. The tree-based structure makes it easy to visualize and understand the logic, and the priority-based execution ensures appropriate behaviors are selected based on the current situation.

Part 3: Decision Tree Learning

For the decision tree learning component, I implemented the ID3 algorithm to learn a decision tree from observed behavior data. The learning system includes:

1. **DataPoint** - Structure for storing attribute-label pairs from observation data
2. **DTNode** - Abstract base class for nodes in the learned decision tree
3. **DTLeafNode** - Leaf nodes containing action labels

4. **DTInternalNode** - Decision nodes that split on attribute values
5. **DecisionTreeLearner** - Main class implementing the ID3 learning algorithm

The learning process follows these steps:

1. Record data from the behavior tree monster over time, capturing its state and actions
2. Discretize continuous attributes into categorical values (for example, distance as "near", "medium", "far")
3. Calculate entropy and information gain to determine optimal splitting attributes
4. Recursively build the decision tree based on information gain
5. Apply the learned tree to control a second monster

The state attributes I used for learning include **[Screenshot #4]**:

1. **Distance to player** - Discretized into "very_near", "near", "medium", and "far"
2. **Relative orientation** - Discretized into "direct_front", "front", "side", and "behind"
3. **Speed** - Discretized into "stopped", "very_slow", "slow", "medium_speed", and "fast"
4. **Can see player** - Binary value (0 or 1)
5. **Is near obstacle** - Categorized as "very_near_obstacle", "near_obstacle", or "no_obstacle"
6. **Path count** - Number of waypoints in current path, categorized as "none", "very_few", "few", "medium", or "many"
7. **Time in current action** - Categorized as "very_short", "short", "medium", "long", or "very_long"

These attributes capture the essential information the monster uses to make decisions, allowing the learning algorithm to approximate the behavior tree's logic.

The ID3 algorithm selects splitting attributes based on information gain, which measures how well an attribute separates the examples based on their action labels. Attributes with higher information gain better predict the appropriate action to take in a given state.

I implemented several optimizations to prevent overfitting and improve the quality of the learned tree:

1. Minimum information gain threshold (0.01) to avoid splits with limited value
2. Minimum example threshold (3) to prevent overfitting to rare conditions
3. Majority voting for attribute values with insufficient examples
4. Discretization of continuous attributes to manage state space complexity

During testing, I found that the quantity of training data significantly affected the performance of the learned decision tree. With insufficient data, the learned monster barely moved at all. With more comprehensive data collection (10,000 frames), the monster began to move and chase the player, showing that it had learned some basic behaviors.

The learned decision tree structure from one of my test runs revealed interesting patterns, as depicted in **[Screenshot #5]**:

Analyzing this tree structure reveals:

1. The algorithm correctly learned that speed was the most important factor in deciding what action to take.
2. It learned the flee behavior for high speeds, which corresponds to the behavior tree's flee behavior when moving fast near obstacles.
3. It properly associated path following with having a non-empty path (when PathCount isn't "none").
4. It learned the association between being stopped and dancing, with time spent in this state being a key factor.

However, a significant limitation became apparent: the learned decision tree monster couldn't properly reproduce the pathfinding behavior of the original behavior tree monster. While it learned when to follow an existing path, it didn't effectively learn when to generate a new path to the player. This highlights a fundamental limitation of decision trees, which is that they struggle to represent sequential behaviors where one action must follow another in a specific order **[Screenshot #6]**.

It's important to note that the quality of the recorded data significantly impacts the learned decision tree's effectiveness. **Good data** with diverse situations and behaviors is *essential* for the learned decision tree monster to function correctly.

Part 4: Comparative Analysis

To evaluate the effectiveness of the different decision-making approaches, I implemented a performance tracking system that measures:

1. **Catch count** - Number of times each monster catches the player
2. **Average time to catch** - Average time taken to catch the player
3. **Qualitative behavior differences** - Visual assessment of movement patterns and decision making

Through testing, I found the following results:

Control Method	Catches	Avg. Time to Catch	Notes

Behavior Tree	14	12.827s	Dynamic movement, effective pathfinding, occasionally dances
Learned Decision Tree	3	62.218s	Frequently gets stuck, fails to pathfind effectively

The behavior tree monster significantly outperformed the learned decision tree version, catching the player much more frequently and with substantially faster average catch times. The learned decision tree monster was only able to catch the player in situations where the environment was relatively open and direct pursuit was sufficient. As soon as complex navigation was required, the learned decision tree monster would become stuck or make ineffective movement choices.

The main differences between the two control methods were:

1. **Pathfinding failures** - The most significant limitation of the learned decision tree was its inability to correctly learn pathfinding and path following behaviors. Despite having access to the same pathfinding functionality, the learned decision tree monster frequently failed to generate and follow paths to the player, instead attempting more direct approaches that often failed when obstacles were present. This fundamental limitation prevented it from effectively pursuing the player in complex environments.
2. **Dance behavior** - The learned decision tree rarely performed the dance behavior compared to the behavior tree, likely because this was an infrequent action in the training data.
3. **Obstacle avoidance** - While both monsters attempted to avoid obstacles, the behavior tree monster had significantly smoother avoidance behaviors. The learned decision tree monster would often get stuck against walls or corners, repeatedly trying the same failed movements.
4. **Recovery from stuck states** - The behavior tree was much more effective at recovering when stuck, using its sequential logic to try alternative approaches. The learned decision tree, lacking this sequential structure, would often get trapped in repetitive behaviors, unable to break out of problematic situations.

These differences highlight the strengths and limitations of decision tree learning from behavior observations. The learned tree successfully captured the most common and straightforward behaviors but struggled with more complex or infrequent behaviors that require maintaining state across multiple decisions.

CSC 584/484 - Building Game AI
Miles Hollifield
April 20, 2025

Additional Observations and Analysis

Decision Tree for Character Control

Implementing an autonomous character controlled by a decision tree provided interesting insights:

1. **Room-based navigation** - Having the decision tree consider the character's current room produced more coherent movement patterns than purely random movement. The character would intelligently move between rooms, creating a more believable exploration behavior.
2. **Responsive obstacle avoidance** - The flee behavior triggered by the decision tree when obstacles were detected helped the character navigate through doorways and around obstacles effectively.
3. **Occasional unpredictability** - The random elements in the decision tree (like the occasional dancing or choosing different destinations) prevented the character from becoming too predictable, which made the simulation more engaging.
4. **Idle detection** - Having the tree check if the character had been idle for too long helped prevent situations where the character would get stuck or remain stationary for extended periods.

One challenge in developing the decision tree for the character was balancing reactivity with goal-directed behavior. If the tree was too reactive to environmental conditions, the character would constantly change its mind and appear erratic. If it was too focused on goals, it might persist with ineffective behaviors. Finding the right balance required careful tuning of the decision conditions and hierarchy.

Parameter Tuning and Performance Optimization

Several parameters significantly affected the effectiveness of the decision-making systems:

Decision Tree Parameters

- Target selection preferences significantly affected the agent's movement patterns and efficiency
- Condition threshold values (for example, what constitutes "near" vs. "far") greatly influenced decision boundaries
- The tree structure itself impacted the complexity of decisions

Behavior Tree Parameters

- Dance behavior frequency (5% chance with 10-second cooldown) affected the balance between goal-directed and decorative behaviors

- Flee behavior sensitivity determined how aggressively the monster avoided obstacles
- Chase sequence design balanced between direct pursuit and intelligent pathfinding

Learning Parameters

- Attribute discretization boundaries affected how well the learned tree could distinguish important state differences
- The amount of training data (10,000 frames in my implementation) influenced the tree's ability to learn rare behaviors
- Information gain threshold controlled the complexity of the learned tree

One interesting finding was that the learned decision tree performed better when trained on a diverse set of situations. When the monster encountered varied scenarios during training (obstacles, clear paths, player visibility changes), the learned tree was more robust. Training data collected from a limited range of situations led to poorer generalization.

Decision Tree vs. Behavior Tree: Design Considerations

Through this implementation, I observed some key differences between decision trees and behavior trees as design tools:

Decision Trees

- Strengths:
    - Simple to understand and implement for straightforward decisions
    - Efficient for state-based decisions that don't require complex sequences
    - Clear visualization of decision logic
    - Well-suited for autonomous character movement decisions
- Limitations:
    - Difficult to represent sequential behaviors
    - Can become unwieldy for complex decision spaces
    - No built-in concept of "running" behaviors

Behavior Trees

- Strengths:
    - Natural representation of sequential behaviors
    - Flexible composition of behaviors through various node types
    - Built-in support for behaviors that execute over time
    - Easy to prioritize and fallback between behaviors
- Limitations:
    - More complex to implement than decision trees
    - Can require more state management
    - May be overkill for simple decision logic

In my implementation, the behavior tree was more effective for controlling the monster because its behavior involved sequences (pathfinding then following) and prioritization (flee before chase before dance). The decision tree worked well for the autonomous character, which primarily needed to make state-based decisions about target selection and movement strategies without needing to maintain complex internal state.

Learning from Observation: Challenges and Solutions

Implementing decision tree learning revealed several significant challenges:

1. **State space representation** - Determining which attributes to track and how to discretize continuous values was difficult. The choice of attributes significantly affected learning quality.
2. **Training data collection** - Gathering enough diverse examples to learn robust behavior required extensive testing. Even with 10,000 frames of data, some behaviors remained underrepresented.
3. **Action selection granularity** - Balancing between fine-grained actions and manageable learning complexity proved challenging.
4. **Temporal relationships** - Capturing behaviors that depend on sequences of actions was perhaps the most significant limitation. Decision trees cannot naturally represent sequential logic, making it difficult to learn behaviors like "path-find first, then follow the path."
5. **Complex behavior learning failures** - The most critical limitation was the failure to learn effective pathfinding and path following. Despite numerous attempts with different training datasets and attribute configurations, the learned decision tree could not reliably reproduce these complex behaviors that were essential for effective player pursuit.

To address these challenges, I implemented several solutions, though with mixed success:

1. Used domain knowledge to select relevant attributes (distance, orientation, etc.)
2. Collected 10,000 frames of behavior data to ensure coverage of various situations
3. Discretized continuous attributes into meaningful categories
4. Included the current action and time-in-action as attributes to partially capture temporal context
5. Experimented with adding the last action as context, though this increased the state space complexity

While these solutions helped create a learned decision tree that could approximate some aspects of the behavior tree's logic, the fundamental limitations in representing sequential behaviors proved insurmountable. The learned decision tree simply could not master the pathfinding-then-following sequence that was essential for effective movement in the environment.

CSC 584/484 - Building Game AI
Miles Hollifield
April 20, 2025

Conclusion

This assignment provided valuable insights into decision-making techniques for game AI, including their limitations. The key findings include:

1. **Decision trees provide efficient autonomous character control**. The autonomous character controlled by a decision tree demonstrated coherent movement patterns and appropriate reactions to the environment. The decision tree structure was effective for making room-based navigation decisions and handling obstacle avoidance.
2. **Behavior trees excel at representing complex, sequential behaviors** with fallbacks and priorities, making them ideal for character control that involves multiple behaviors. Their hierarchical structure provides clear organization of behavior logic.
3. **Decision tree learning showed mixed results in approximating complex behavior**. While it could learn simple reactive behaviors, it fundamentally failed to capture critical sequential behaviors like pathfinding followed by path following. This limitation severely restricted its effectiveness in complex environments.
4. **Data quantity and quality significantly affect learning outcomes**. With insufficient training data, the learned decision tree produced almost no movement. Even with substantial data (10,000 frames), it failed to learn complex sequential behaviors effectively.
5. **The environment parameterization is crucial for effective decision making**. The choice of state representation significantly affects both hand-crafted and learned decision systems. Even with careful parameterization, some behaviors proved impossible to learn with decision trees.
6. **Different AI techniques are suited to different types of behaviors**. Behavior trees were superior for controlling complex character behavior with sequential elements, while decision trees were adequate for simpler state-based decisions about movement targets and reactive behaviors.

The integration of decision trees and behavior trees with the movement and pathfinding systems from previous assignments created a complete AI system capable of intelligent navigation, obstacle avoidance, target selection, and complex behavior patterns. However, the limitations encountered when attempting to learn these behaviors highlight important challenges in AI behavior learning.

For future work, I would like to explore more advanced techniques that might overcome these limitations:

- More sophisticated learning approaches like neural networks that can better capture sequential dependencies
- Behavior tree learning methods that preserve the hierarchical and sequential nature of behaviors
- Hybrid systems that use learning for reactive behaviors but hand-crafted logic for sequential behaviors

- Reinforcement learning approaches that might discover effective behaviors through trial and error
- State machines combined with decision tree learning for better handling of different behavioral modes
- Alternative behavior representations that are more amenable to learning from observation

This project has deepened my understanding of AI decision-making systems and their integration into game environments, providing a solid foundation for developing more sophisticated game AI in the future.

Appendix

Acknowledgements

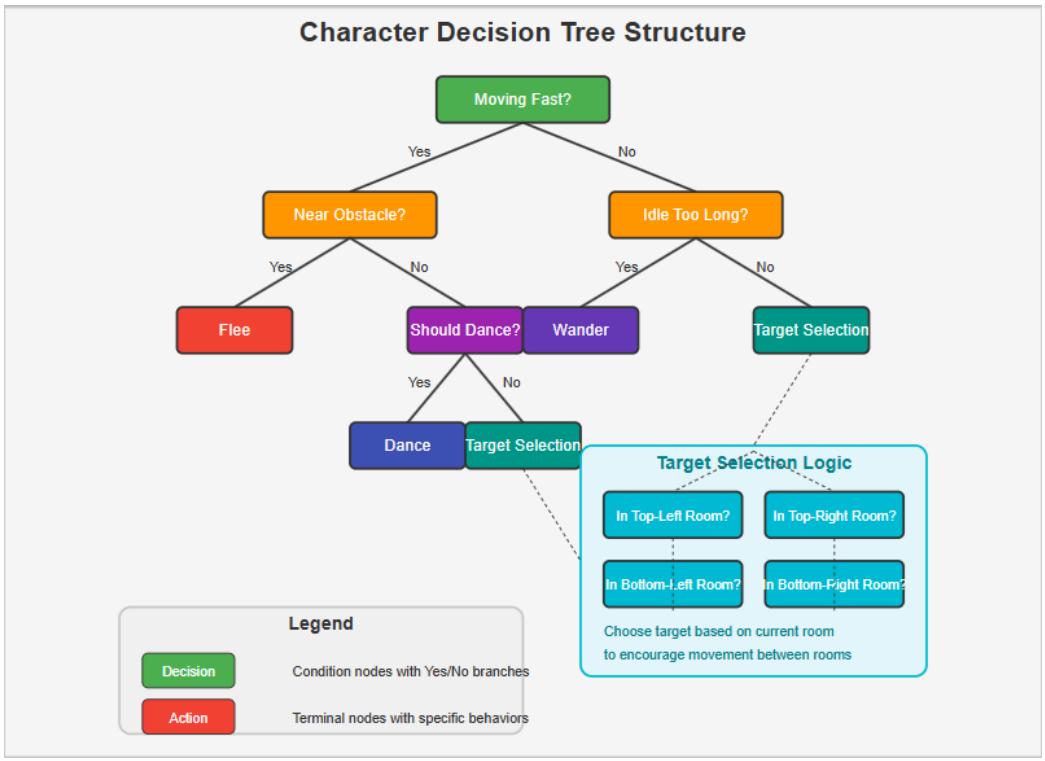
I referred to the SFML official tutorials for implementing graphical components. The book "Artificial Intelligence for Games" by Ian Millington provided insights into decision trees, behavior trees, and their integration with previously implemented movement systems.

I used OpenAI's ChatGPT as a resource to assist in implementing specific components such as the behavior tree, decision tree learning algorithm, and the monster behaviors, as noted in the source code documentation. ChatGPT was also used to assist in writing the template of this writeup, along with fixing grammatical errors and improving the flow of the writeup.

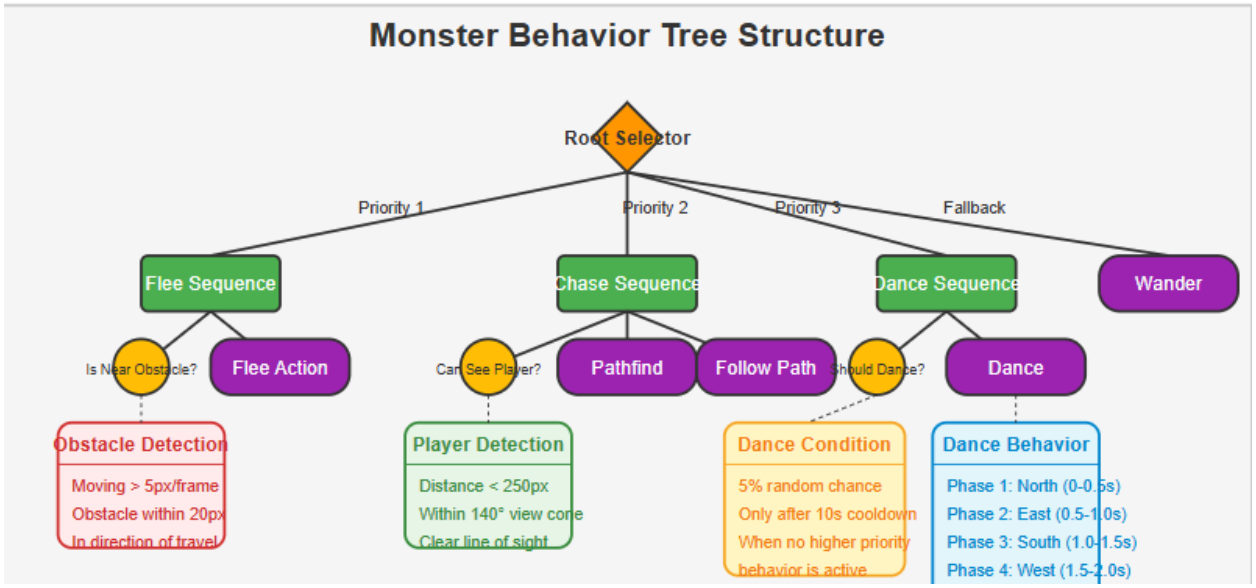
I adapted the Breadcrumb class provided by TA Derek Martin for visualizing agent movement, and used the sample Makefile provided by Dr. David L. Roberts as a starting point for my build configuration.

Screenshots

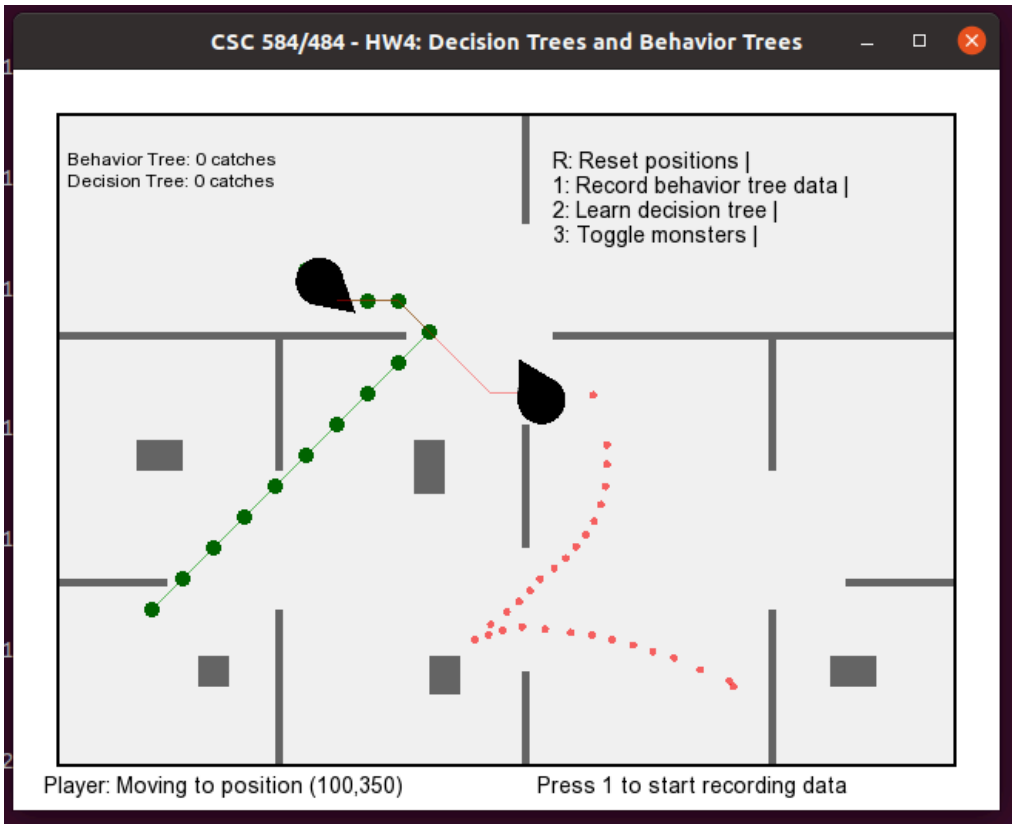
[Screenshot #1]



[Screenshot #2]



[Screenshot #3]



[Screenshot #4]

HW4 > behavior_data.csv

```
DistanceToPlayer,RelativeOrientation,Speed,CanSeePlayer,IsNearObstacle,PathCount,TimeInCurrentAction,Action
478.205,-122.757,50,0,1,0,1.23747,Wander
478.301,-122.771,50,0,1,0,1.23987,Wander
478.43,-122.772,50,0,1,0,1.24312,Wander
478.484,-122.782,50,0,1,0,1.24448,Wander
478.55,-122.782,50,0,1,0,1.24612,Wander
478.611,-122.77,50,0,1,0,1.24765,Wander
478.723,-122.718,50,0,1,0,1.25046,Wander
478.794,-122.677,50,0,1,0,1.25226,Wander
478.942,-122.585,50,0,1,0,1.25598,Wander
479.082,-122.533,50,0,1,0,1.2595,Wander
479.213,-122.452,50,0,1,0,1.2628,Wander
```

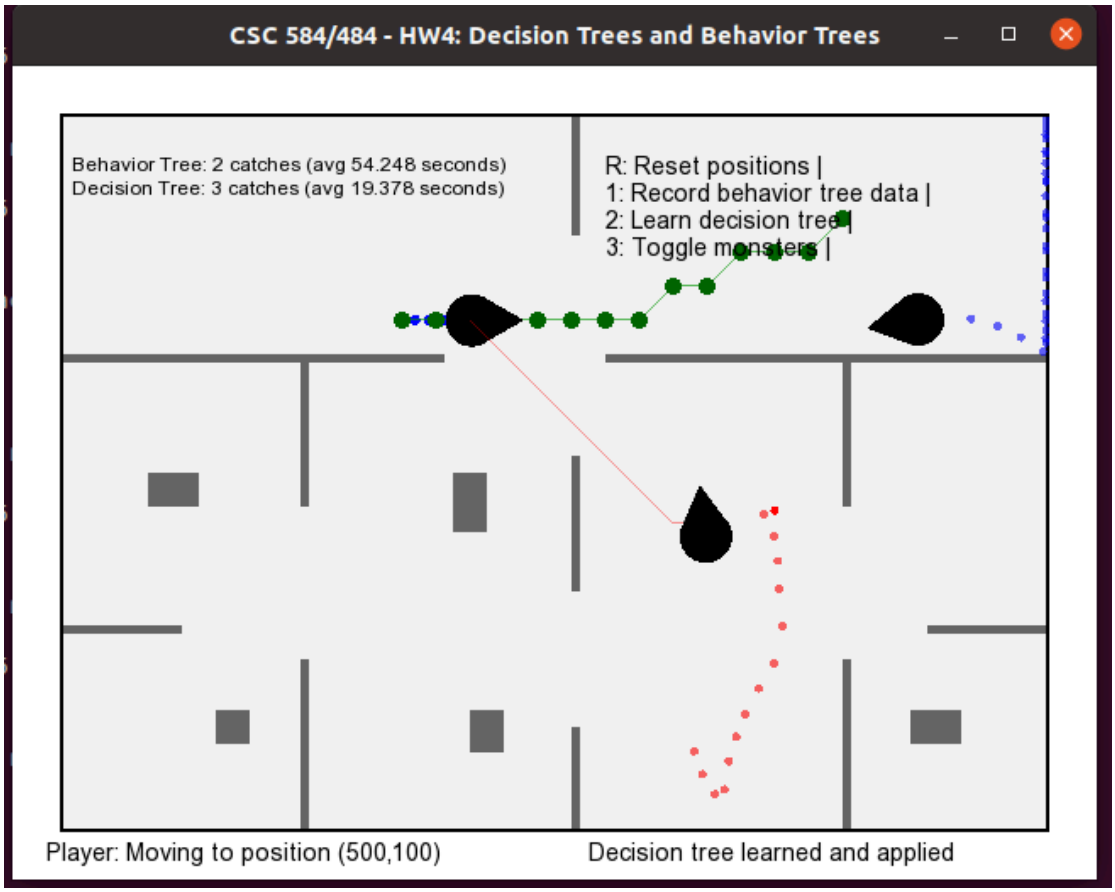
[Screenshot #5]

HW4 > learned_decision_tree.txt

```
DistanceToPlayer,RelativeOrientation,Speed,CanSeePlayer,IsNearObstacle,PathCount,TimeInState
SPLIT ON: Speed
  Speed = very_slow:
    SPLIT ON: PathCount
      PathCount = very_few:
        LEAF: FollowPath
      PathCount = none:
        SPLIT ON: RelativeOrientation
          RelativeOrientation = side:
            LEAF: Wander
          RelativeOrientation = front:
            LEAF: Wander
          RelativeOrientation = direct_front:
            LEAF: Wander
          RelativeOrientation = behind:
            LEAF: Wander
      PathCount = medium:
        SPLIT ON: RelativeOrientation
          RelativeOrientation = side:
            LEAF: FollowPath
          RelativeOrientation = front:
            LEAF: FollowPath
          RelativeOrientation = direct_front:
            LEAF: FollowPath
          RelativeOrientation = behind:
            LEAF: FollowPath
      PathCount = few:
        LEAF: FollowPath
  Speed = stopped:
    SPLIT ON: TimeInState
      TimeInState = short:
        LEAF: Dance
      TimeInState = very_short:
        SPLIT ON: RelativeOrientation
          RelativeOrientation = side:
            LEAF: Dance
          RelativeOrientation = front:
            LEAF: Dance
          RelativeOrientation = direct_front:
            LEAF: Wander
      TimeInState = medium:
        LEAF: Dance
  Speed = slow:
    SPLIT ON: PathCount
      PathCount = none:
        LEAF: Wander
      PathCount = medium:
        LEAF: FollowPath
  Speed = medium_speed:
    LEAF: FollowPath
  Speed = fast:
    LEAF: Flee

```

[Screenshot #6]


CSC 584/484 - HW4: Decision Trees and Behavior Trees
Behavior Tree: 2 catches (avg 54.248 seconds)
Decision Tree: 3 catches (avg 19.378 seconds)
R: Reset positions |
1: Record behavior tree data |
2: Learn decision tree |
3: Toggle monsters |
Player: Moving to position (500,100)
Decision tree learned and applied