For implementation of the decision trees a created a game in which a boid navigates a simple maze to get to the goal at the other side. There is an enemy that spawns in the middle of the maze that will chase the boid and try to hit it. If the boid reaches the goal it wins, but if the enemy monster hits the boid then it wins.

Part1:
The boid is equipped with several abilities that allow it to navigate its environment intelligently. It uses an Arrive steering behavior to smoothly move toward target positions, following waypoints while avoiding obstacles and boundaries. When an enemy is detected within a specified radius, the boid switches to evasive maneuvers, which include randomized behaviors such as flanking, dodging, or bursting in a random direction to escape. The boid also incorporates boundary avoidance, ensuring it does not collide with the edges of the environment by applying corrective forces. Its velocity is dynamically adjusted to stay within its maximum speed, and its orientation is updated based on its movement direction. The boid uses a behavior tree to decide between normal pathfinding and evasive actions, dynamically adapting to the presence of enemies. Additionally, it handles collisions with walls by reversing its velocity and slightly reducing its speed.

Behavior Tree of Boid:
The boid's behavior tree is designed to handle decision-making for pathfinding and evasion. It uses a hierarchical structure of nodes to evaluate conditions and execute actions dynamically.
Structure:

- Root Node: A DecisionBranch that checks if an enemy is within the detection radius (200 units).
- True Branch: Executes evasive maneuvers using an ActionNode that randomly selects one of three behaviors:
  - Random Flank: Moves perpendicular to the enemy's direction.
  - Random Dodge: Executes a random angular dodge.
  - Random Burst: Temporarily accelerates in a random direction.
- False Branch: Executes normal pathfinding using an ActionNode that calculates steering forces to follow waypoints

Key Features:

- Dynamically switches between evasion and pathfinding based on proximity to the enemy.
- Includes obstacle avoidance and boundary checks to ensure safe navigation.

Part 2:

The enemy is designed to aggressively pursue the boid while maintaining its own safety. It uses a behavior tree to make decisions based on its environment. When the boid is detected within a certain radius, the enemy evaluates whether it is near a wall. If a wall is detected, the enemy executes a wall avoidance behavior to steer away from obstacles. Otherwise, it switches to a chasing behavior, calculating the direction to the boid and applying acceleration to close the distance. If the boid is not detected, the enemy defaults to a wandering behavior, moving randomly within the environment. The enemy also limits its speed to ensure realistic movement

and dynamically adjusts its velocity and orientation based on its current behavior. For the LearningEnemy variant, the enemy records gameplay data, such as the boid's position, velocity, and map layout, to train a decision tree. This allows it to predict actions based on environmental states, enabling it to adapt and improve its behavior over time. The learning system balances between predefined fallback actions and learned behaviors, making the enemy more effective in chasing or evading as it gains experience.

Behavior Tree of Enemy

The enemy's behavior tree is more complex, incorporating composite nodes to handle multiple conditions and actions.

Structure:

- **Root Node**: A DecisionBranch that checks if the boid is within the detection radius (500 units).
  - **True Branch**: A DecisionBranch that checks if the enemy is near a wall.
    - **True Branch**: Executes avoidWallAction to steer away from walls.
    - **False Branch**: Executes chaseBoidAction to pursue the boid.
  - **False Branch**: Executes wanderAction to move randomly when the boid is not nearby.

Composite Nodes:

- **Selector Node**: Chooses between chasing the boid or wandering based on proximity.
- **Sequence Node**: Combines wall avoidance and chasing into a sequential behavior.

Key Features:

- Prioritizes survival by avoiding walls.
- Aggressively pursues the boid when in range.
- Falls back to wandering when no target is detected.
-

Part 3:
The learning system collects behavioral data through the LearnedBehavior class, which maintains a vector of BehaviorSample structures. Each sample contains comprehensive state information including the enemy's kinematic data (position, velocity), player's kinematic data, map state, and the action taken (stored as a force vector). During gameplay, the recordBehavior method captures these samples whenever the enemy makes decisions. The system processes this data through a binary decision tree structure, where each non-leaf node represents a decision point based on features extracted from the state (distance to player, angle, relative speed, and wall proximity). The tree is constructed using a recursive algorithm in buildTree that splits the training data based on feature thresholds, choosing splits that minimize action variance within

child nodes. Leaf nodes store average actions from their training samples. When making real-time decisions, the makeDecision method traverses this trained tree based on the current game state, following decision paths until reaching a leaf node whose stored action is then executed. The tree effectively learns to map similar game situations to successful actions observed during training, with a maximum depth of 10 to prevent overfitting. The training process uses approximately 1000 samples (controlled by MAX_SAMPLES) to build a balanced representation of enemy behavior patterns.

Learning Tree

The **learning tree** is a decision tree trained using recorded behaviors to predict actions based on environmental states. It allows the enemy to adapt its behavior dynamically.

Training:

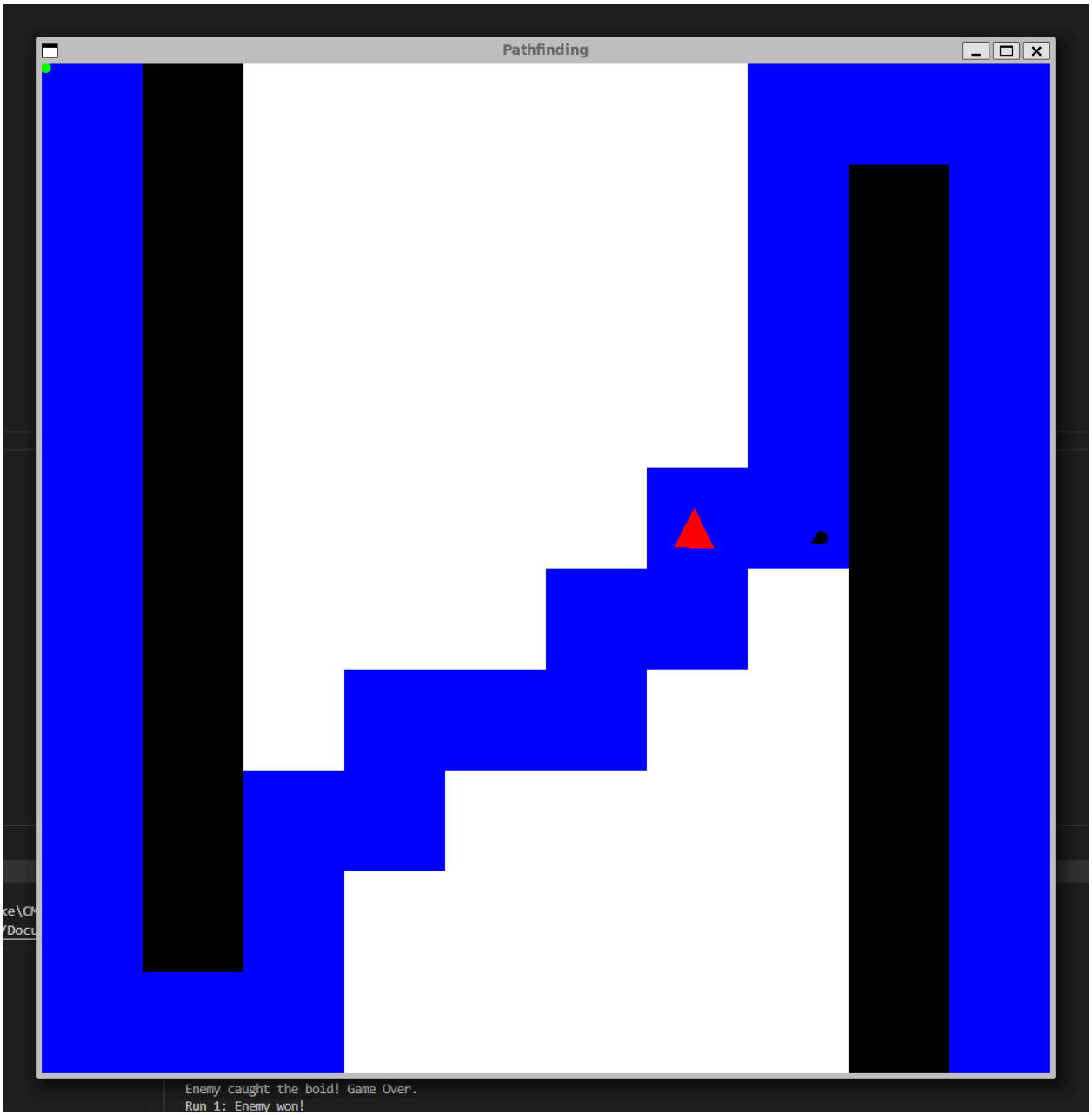
1. **Data Collection**:
    - Records samples of the enemy's state, the boid's state, and the map layout during gameplay.
    - Each sample includes features such as:
        - Distance to the boid.
        - Angle to the boid.
        - Relative speed.
        - Proximity to walls.
2. **Tree Construction**:
    - Uses the recorded samples to build a decision tree.
    - Splits data based on feature thresholds to create internal nodes.
    - Leaf nodes contain actions (e.g., "Move toward the boid" or "Dodge").

Structure:

1. **Internal Nodes**:
    - Evaluate conditions (e.g., "Is distance to the boid < threshold?").
    - Direct the traversal to the appropriate child node based on the result.
2. **Leaf Nodes**:
    - Contain actions that the enemy should execute based on the current state.

Key Features:

- Predicts actions based on the current state by traversing the tree.
- Continuously improves through training, allowing the enemy to adapt to new scenarios.
- Balances between learned behaviors and predefined fallback actions.



Screenshot of game

Appendix:
Used Github copilot