CSCI 218 Foundation of Artificial Intelligence

Assessment 2 - Project

Topic 1 - Adding AI to a game of Frogger

Team Members

Teng Ian Khoo - 8121667
Nathan Gonsalves-7849187
Yash Puri - 7386345
Aatiqul Haque - 7885337

Executive summary

The project and the topic that we have decided focuses on the classic game called frogger, in which we have implemented AI learning tools which adds intelligence and adaptiveness to the game. Frogger is a game where the user guides the frog across a busy road (with multiple lanes) or a lake/river with many obstacles. The code aims to modernize the game by including the AI features which can respond to player actions and game state which makes the game more interactive and engaging.

The updated game uses python as the programming language with Pygame for game development. Some of the functionalities and algorithms that are implemented in the game are decision-making, route optimization and adaptive behavior. For example, obstacles like cars and logs are programmed to adapt their speed and patterns based on the user's performance. The game also includes AI-based path finding and state management, which enables smarter

entities and challenges. These improvements improves the player experience and also serve as a learning tool to apply AI concepts in constraint, rule-based environment

This study demonstrates how AI may be applied practically in games, converting traditional games into complex, modern systems, in addition to the evident gameplay enhancements. It also sets the stage for future advancements, such as the creation of self-governing agents that can play the game by themselves using reinforcement learning. The project offers a compelling player experience by showcasing AI's decision-making and adaption skills while balancing technical complexity and fun.

Motivation

This enhanced game was developed in response to the growing interest in artificial intelligence applications and its ability to revive the appeal of classic games. Because they provide a controlled environment with clear rules and objectives, traditional games like Frogger are ideal for integrating and testing AI technologies. This project seeks to blend innovation and nostalgia by demonstrating AI's ability to enhance interactivity and user engagement.

1. Modernizing Gameplay:

Even though Frogger and other old games are legendary, their gameplay is built on set patterns and difficulty levels. This project creates a dynamic, flexible gaming environment that reacts to player actions in real time by utilizing artificial intelligence (AI). The study demonstrates how AI may transform old games into modern interactive systems while also boosting player engagement by modernizing gameplay.

2. Analyzing AI in Games:

A range of AI concepts, such as pathfinding algorithms, decision-making under constraints, and customizable difficulty, are perfect for testing in games. This project provides a framework for putting various AI techniques into practice while highlighting their benefits and drawbacks. It bridges the gap between the theoretical knowledge of artificial intelligence and its practical implementation.

3. Technical and Educational Development:

This project offers game developers a hands-on opportunity to learn about and use AI techniques. Understanding concepts like obstacle prediction, route optimization, and

state machines is enhanced by Frogger's controlled environment, which enables targeted experimentation with AI algorithms.

4. Better User Experience:

By replacing static patterns with intelligent action, the AI-enhanced Frogger makes the game more engaging and challenging for players. Adaptive AI can improve the immersion of the game and fortify the player-game relationship by responding to the player's actions and desired degree of difficulty.

5. AI's Growing Significance in Gaming:

As AI becomes a more important component of the gaming industry, this research shows how it may enhance user experiences and create more intelligent, interactive games. The study not only illustrates AI's immediate impacts but also lays the groundwork for further investigation into more intricate AI concepts like reinforcement learning.

This project's overall objective is to bring together creativity and enjoyment by using AI to enhance a timeless classic and providing the developers with a practical learning experience. It shows how traditional games can evolve and adjust to new technology while still being popular in a gaming industry that is always changing.

Technical Requirements

The development and execution of this AI-enhanced Frogger game required a specific setup to ensure compatibility and optimal performance. Below are the technical specifications for both the FSM-based and BT-based implementations.

Programming Language and Tools

Language: Python 3.8 or higher was used for this project to take advantage of compatibility with required libraries and ensure stable performance.
Development Tools:

- IDE: Visual Studio Code was used for its efficient code editing and debugging.
- Game Engine: Pygame library was used for game rendering and audio handling.

Libraries and Frameworks

The following Python libraries were utilized:

- Pygame: For game logic, rendering, and event handling.
- py_trees: Used exclusively in the BT implementation for creating and managing behavior tree structures.
- sys: For system-level operations, including exiting the game.
- random: For adding randomness to game mechanics.

Machine Specifications

- Processor: Intel i5 8th Generation to handle game logic and rendering smoothly.
- RAM: Minimum 4GB to accommodate Python runtime and Pygame's graphical processes.
- Graphics: Integrated graphics are sufficient as the game is lightweight and relies on 2D rendering.

Environment Setup:

Steps for setting up the development environment:

- Install Python 3.8 or higher.
- Clone the project repository or download the zip.
- Install required dependencies.
- Run the game file with `python frogger_fsm.py` or `python frogger_BT.py`.

Additional Assets

Game Assets: Images, sprites, and sound effects are located in the images/ and sounds/ directories. These were derived from the original Frogger game project and slightly modified for this implementation.

Main Constructs

For our project we decided to research and implement two classical methods to implementing game AI, used extensively in the past for many older games. We chose them because there are a lot more resources we can refer to, and comparatively they are easier to understand and implement.

It is important to state that we took the base Frogger game from this:
https://www.pygame.org/project/3756

We did not change the game logic of cars,platforms,etc. and assets, we just changed some aspects of the code to suit our use case.

Overview of Frogger

Frogger is a classic arcade game where the player controls a frog, navigating it through various obstacles to reach designated safe zones. The game environment typically comprises:

1. **Street Section:** The frog crosses busy roads while avoiding moving vehicles.
2. **River Section:** The frog leaps onto floating platforms (logs or turtles) to avoid falling into the water.
3. **Safe Zones:** The frog's goal is to reach these safe zones at the top of the screen.

Finite State Machines (FSM)

FSMs are a fundamental approach in game AI used to model an agent's behavior through a set of defined states and transitions. Each state represents a specific behavior or condition of the agent, and transitions between states occur when predefined events or conditions are met. FSMs are widely used in game development because they offer a structured and deterministic method for handling game logic.

Why FSMs?

1. **Predictable Behavior:** FSMs ensure that an agent's actions are clearly linked to its state. This predictability is essential for creating game AI that responds in a way players can understand and anticipate.
2. **Simple Logic Handling:** For scenarios with a limited number of behaviors or conditions, FSMs provide a straightforward and efficient way to organize game logic. This makes FSMs ideal for applications like character movement, enemy AI, or handling basic environmental interactions.
3. **Efficiency:** FSMs are computationally lightweight. Since the system only evaluates transitions from the current state, they are faster than more complex models like Behavior Trees for smaller systems or limited state spaces.

Behavior Trees (BT)

Behavior Trees are a hierarchical decision-making model widely used in game AI. Unlike FSMs, BTs represent actions and decisions as nodes in a tree structure, allowing for more modular and reusable logic. The tree is composed of composite nodes, decorators, and leaf nodes, which collectively define an agent's behavior.

Why BTs?

1. **Modularity:** BTs allow developers to break down complex behaviors into smaller, reusable nodes. This modularity is particularly beneficial in large-scale games with diverse agent behaviors.
2. **Scalability:** BTs can be easily extended by adding new nodes or rearranging the tree. This flexibility makes BTs well-suited for games where AI behaviors need to evolve over time or across different levels.
3. **Dynamic Decision Making:** BTs can dynamically evaluate conditions and select actions based on the agent's current state and environment. This makes them more adaptable than FSMs for scenarios with many possible behaviors or changing conditions.

Implementation

The AI implementation in this project focuses on enabling the frog to autonomously navigate the game's environment. The code is structured into modular components, reflecting the main aspects of the game.

Some important changes that we made to the game was:

1. Make a random delay from 3 -5 seconds before the frog is allowed to move; this is so that the game state is more populated by cars, logs etc. before the Frog is allowed to move. Otherwise we found that the Frog can skip the street section entirely by just going up when cars have not yet populated the streets
2. Have a "cooldown" or delay between moves; this is to better simulate human behaviour, otherwise the frog moves every game tick and is too strong of a player.

Finite State Machine (FSM) Implementation

The FSM approach defines discrete states for the frog's behavior:

- **IDLE:** Waiting for conditions to move.
- **MOVING:** Actively navigating the environment.

Each state is associated with specific logic and transitions, FSM is suitable for games like Frogger due to its simplicity and deterministic behavior. It ensures the frog reacts predictably to environmental triggers.

We chose a very simple and limited number of states for this AI. The reason for this is we found that adding more states simply added more complexity but did not add considerable advantages in terms of performance and accuracy.

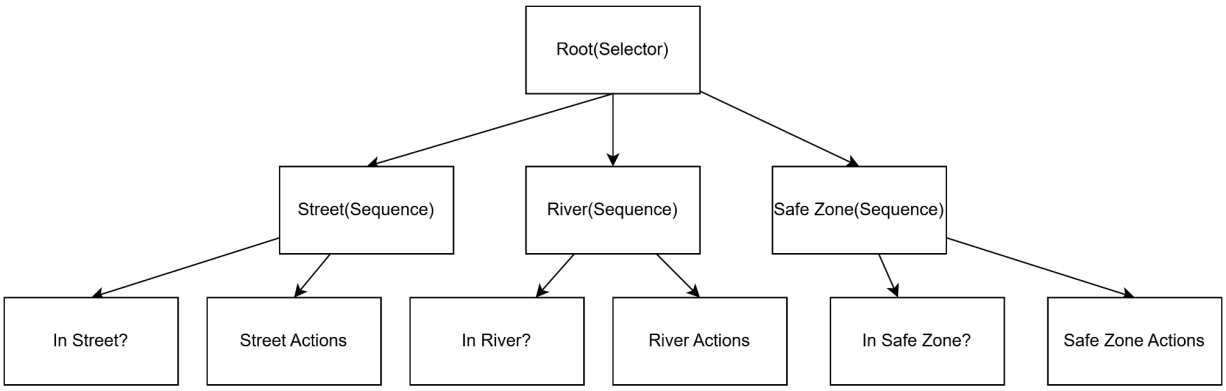
Behavior Tree (BT) Implementation

Behavior Trees (BTs) provide a more flexible and modular alternative. The frog's actions are encapsulated within nodes, organized hierarchically:

- **Composite Nodes:** Control the flow of execution. Example: A selector node prioritizes entering a safe zone over navigating the street.
- **Leaf Nodes:** Perform specific tasks like moving the frog or checking conditions.
- **Decorator Nodes:** Modify the behavior of child nodes, e.g., limiting the frequency of node execution.

The AI ticks the behavior tree every frame, dynamically adapting to the game environment.

We separated the checking of which part of the game we are in(explanation why in "Game Logic by Section" below) with how we handle each section and the actions to take into separate behaviours for more modularity and separation of concerns.



Our Behaviour Tree Implementation

In a BT the sequence of nodes is important, from highest to lowest priority it goes from left to right. We designed the BT to look at the street section first as it is the most dynamic section of our current Frogger game and the added overhead of iterating through the other sections first might have caused a few Frog deaths.

Game Logic by Section

In Frogger, the frog's behavior varies significantly depending on its position within the game. Each section—**street**, **river**, and **safe zone**—has unique challenges requiring tailored logic to successfully solve them. Each of our implementations both FSM and BTs use the same logic for each section so functionally they are the same but how they are implemented and the positives and negatives of each implementation will be discussed later.

Street Section Logic

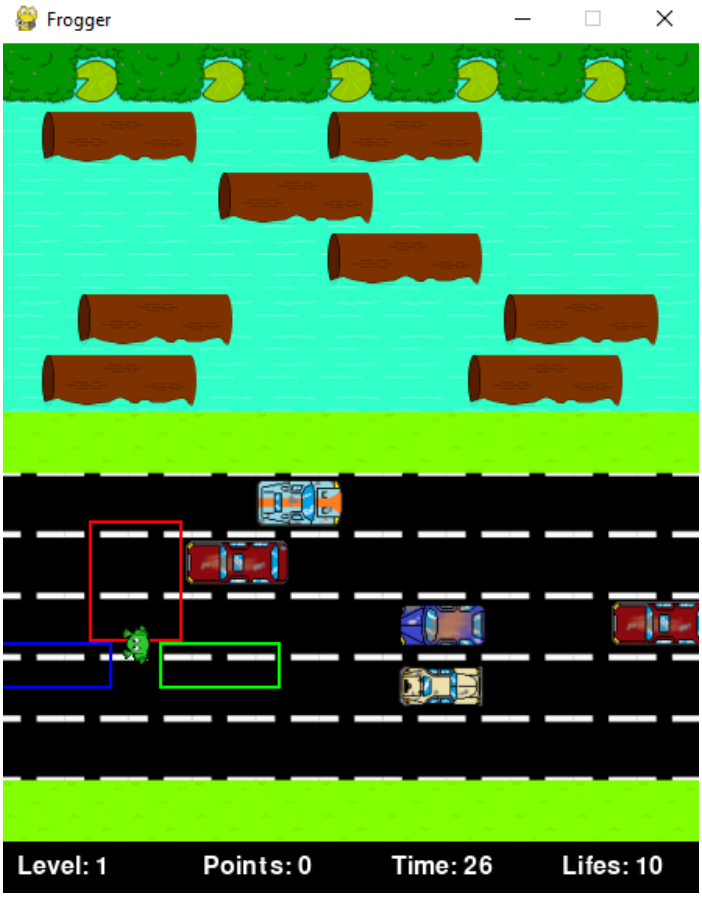
The street is characterized by dynamic obstacles (moving cars) that can instantly kill the frog. The primary objective here is to:

1. Avoid collisions with vehicles.
2. Progress toward the river as quickly and safely as possible.

Danger Zones:

We define danger zones relative to the frog:

- **Up Danger Zone:** Prevents moving forward if cars are detected two spaces ahead.
- **Left Danger Zone:** Avoids moving left if a car moving right might intercept the frog.
- **Right Danger Zone:** Avoids moving right if a car moving left might intercept the frog.



Screenshot from the Game, with us drawing the danger zone detection areas.

Why This Approach?

- **Predictive Safety:** By analyzing danger zones, the wel predicts potential collisions before moving, reducing reactionary behavior.
- **Priority-Based Movement:** The frog always prioritizes moving **up** (toward the river) unless the path is unsafe. If the upward path is blocked, the AI shifts focus to lateral movement to avoid hazards while maintaining progress.
- **Avoids Over-Complexity:** While advanced techniques like trajectory prediction could be implemented, danger zones balance computational efficiency and effectiveness.

Alternative Approaches:

- *Pathfinding Algorithms (e.g., A):** These would consider the entire street layout but add unnecessary computational overhead given the frog's limited movement options (up, left, right).

River Section Logic

The river section poses unique challenges, as the frog cannot remain stationary without a platform. We must navigate logs and other moving platforms to cross the river safely. The main goal in this section is to:

1. Move forward when a platform is directly ahead.
2. Avoid falling into the water by ensuring alignment with platforms.

Platform Detection and Upward Movement

The AI checks if there is a platform directly ahead of the frog. If a platform overlaps the area directly in front of the frog, it moves up, this ensures that the frog only progresses toward the safe zone when it has a stable path forward. The movement is conditional on detecting a platform, prioritizing safety.

Edge Handling

If the frog is near the left or right edges of the playable area, it must avoid falling off. The AI takes the following steps:

- **Left Edge:** If the frog is near the left boundary (less than 30 pixels from the edge), it moves right to stay within the playable area.
- **Right Edge:** Similarly, if the frog is near the right boundary (more than 390 pixels from the edge), it moves left.

Edge handling ensures the frog does not fall off the screen due to misalignment or lack of platforms. We do this in hopes that while moving, a log will appear in front of the frog and it will go upwards.

Why This Approach?

1. **Simplified Logic:** By focusing solely on upward movement and edge avoidance, the AI avoids unnecessary lateral movement, which could waste time and lead to poor positioning.
2. **Prioritization of Goals:** The frog prioritizes progressing toward the safe zone (upward movement) whenever it is safe, ensuring steady progress.
3. **Reliability:** This approach minimizes errors and avoids complex horizontal calculations that could increase computational overhead without significant benefits.

Alternative Approaches and Their Trade-Offs

- **Pathfinding to Nearest Platform:** Calculating the shortest path to the nearest platform would require scanning the river area continuously. While this could optimize movement, it introduces significant computational overhead.

By combining platform detection with straightforward edge handling, this logic simplifies the river section while maintaining robust safety and progress mechanisms. In other implementations where the Frog attempts to align itself to the nearest upcoming log, it would fall and die in the River much, much more frequently.

Safe Zone Logic

Safe zones are the ultimate goal of the frog's journey. The logic here ensures the frog successfully reaches an available safe zone without redundantly entering already-filled zones.

Direct Entry Requirement:

The frog only moves into a safe zone if directly below it. This prevents unnecessary lateral movement.

Why This Approach?

- **Avoids Redundant Behavior:** Without direct entry checks, the frog might repeatedly try to enter filled safe zones, wasting time and risking death.

- **Efficiency:** By removing occupied safe zones from consideration, the AI focuses only on available zones, streamlining decision-making.

By tackling each section individually while maintaining overarching goals (e.g., upward movement priority), this approach balances complexity, performance, and effectiveness. Other more complex behaviours may result in better performance

Limitations/Difficulties and Constraints

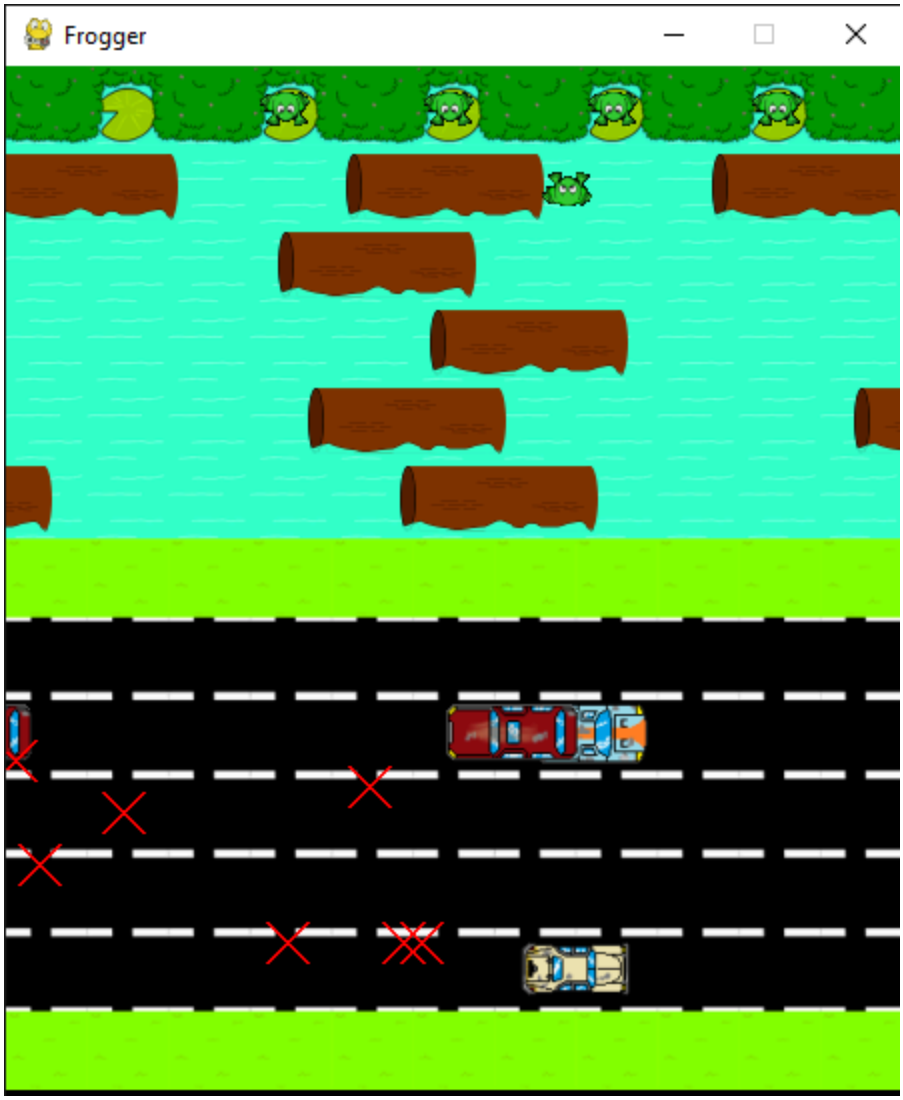
Simplistic Design

The AI implemented in our Frogger game is relatively simple, with a focus on rule-based logic rather than advanced pathfinding algorithms. For example, the frog does not actively "chase" an optimal safe zone but simply moves upward, reducing the strategic depth of its behavior. In addition to that, unlike the original Frogger game, our version features a rather simplified river section. It lacks disappearing logs and a more dynamic river environment.

Inefficient Safe Zone Handling

The logic we implemented behind safe zones is rudimentary. If a safe zone is already occupied or unreachable, the frog does not adapt by seeking an alternative route. This leads to scenarios where the frog waits on the last set of logs, unable to score and waiting to lose and restart.

Here is an example. In the demo screenshot below, it is evident that the zones 2 through 5 are filled. Our frog will just wait at the last set of logs to try and score by going up. This means if a frog is at the last set of logs from zones 2 onwards, it will never score and just wait for death.



The Frog does not attempt to find a way to zone 1, even though all the rest of the zones are filled.It will wait at the last log, and since all of it is full will eventually die.

Behavior Tree Latency

The behavior tree implementation sometimes results in slower decision-making compared to the FSM. The frog takes longer to evaluate its options and execute movements, which can make gameplay feel sluggish.

Limited Obstacle Adaptiveness

While obstacles like cars and logs adapt their speeds and patterns to some extent, this adaptability is constrained to predefined rules. Introducing more randomness or strategic movement in obstacles could enhance the unpredictability of the game.

Performance comparison between AI Technologies Implemented

The FSM-driven version operates faster and more predictably, while the BT-driven version demonstrates slower reactions due to the hierarchical evaluation process. This inconsistency in response times affects the overall fluidity of the game.

Game Design Limitations

Adapting a classic game like Frogger to incorporate AI features while preserving the original gameplay essence was also a challenge. Extensive testing and fine-tuning had to take place to preserve the base working of the game while we ensured the operation of the AI-driven obstacles remained the same.

Scalability

The project's AI design was tailored specifically for Frogger. Extending the current AI implementation to more complex games directly from the logic and working we implemented, would not work.

Conclusion

This project demonstrates the potential of incorporating AI concepts into a classic game like Frogger. By leveraging 2 concepts; Behavior trees and FSM, the game was successfully transformed to work with AI capabilities. Despite its simplicity, our implementation provides a solid foundation for understanding the potential of AI in enhancing challenge dynamics in simple games.

The project also highlights the trade-offs between different AI approaches. FSMs, while straightforward and efficient, lack the flexibility and scalability offered by behaviour trees. However, the behaviour tree's slower decision-making process underscores the importance of balancing complexity with performance in AI systems.

While our solution achieved its objectives, it also revealed several areas for improvement. The simplified handling of safe zones, the lack of advanced pathfinding, and the static nature of certain game elements indicate opportunities to enhance some parts further. Incorporating dynamic features, such as disappearing logs or smarter obstacle patterns, could elevate the gameplay experience.

Potential future enhancements include adding pathfinding algorithms like A*, reinforcement learning for self-learning agents, or multi-agent coordination that could enable the AI to be more intelligent and adaptive. These additions could push the boundaries of the game.

This project has been an invaluable learning experience, demonstrating how the feasibility of traditional gameplay can merge with AI technologies.

Code

Our code should be in a zip file along with the report.
Our main contribution is the modification of the base Frogger code into two separate files running our AI algorithms:
frogger_FSM.py and frogger_BT.py

Acknowledgement

We want to acknowledge and credit the base code and assets for the Frogger game that we modified to come from:
https://www.pygame.org/project/3756