NILE UNIVERSITY

RACING USING GENETIC ALGORITHM

AIS201: ARTIFICIAL INTELLIGENCE-2023SPRG-LECT-01

PREPARED BY

AMR SHAARAWY
YOUSSEF ABDELMAKSOUD
OMAR HAZZAA
YASSIN ELASFAR
ABDALLAH EMAM

Table of Contents

SECTION I: INTRODUCTION

Welcome to the introduction of our exciting racing game project! In this project, we aim to create an immersive and challenging racing game that pushes the boundaries of artificial intelligence (AI). Our objective is to develop intelligent AI opponents that can adapt and improve their racing skills over time, providing players with a thrilling and realistic racing experience.

To achieve this, we have incorporated genetic algorithms into the AI aspect of our project. Genetic algorithms are a type of optimization algorithm inspired by the principles of natural selection and genetics. They simulate the process of evolution to iteratively improve solutions to complex problems. In the context of our racing game, genetic algorithms will be used to evolve and train the AI opponents to become progressively better at racing.

By leveraging genetic algorithms, we can create AI opponents that not only demonstrate intelligent decision-making skills but also continuously learn and evolve their racing strategies. The genetic algorithm will iterate through a population of AI agents, evaluating their performance in races and selecting the most successful individuals to pass their genetic material onto the next generation. Through this iterative process of selection, crossover, and

mutation, the AI opponents will gradually improve their racing skills and adapt to different track conditions and player strategies.

The use of genetic algorithms in our racing game project not only adds depth and challenge to the gameplay but also showcases the power of evolutionary computation in creating sophisticated AI systems. Players can expect dynamic and competitive races, as the AI opponents will learn and adjust their racing techniques in response to their experiences and the player's actions.

In the following sections of this project, we will delve deeper into the implementation details and showcase the results of our genetic algorithm-based AI system. We are excited to present this innovative approach and hope that it will enhance your racing game experience, providing hours of thrilling gameplay and intense competition. Let's buckle up and get ready to race against some of the most intelligent AI opponents ever created!

SECTION II: BACKGROUND

Genetic algorithms are optimization techniques inspired by natural selection and genetics. They simulate the process of evolution to improve a population of solutions. In our racing game project, we use genetic algorithms to train AI opponents by representing them as individuals or chromosomes. The fitness function evaluates their performance in races, and through selection, crossover, and mutation, the AI opponents evolve and improve their racing strategies.

Research in AI and racing games has inspired our project. Studies have shown the effectiveness of genetic algorithms in training AI agents for complex tasks like racing. Reinforcement learning and genetic programming have also contributed valuable insights. By combining these approaches, we aim to create sophisticated AI opponents that possess adaptive racing skills, providing players with challenging and entertaining gameplay.

SECTION III: GAME OVERVIEW

The code begins by importing various modules such as pygame, math, random, os, and PIL (Python Imaging Library). These modules provide necessary functionalities for graphics, mathematical calculations, random number generation, file system operations, and image processing.

Next, several variables are defined to control different aspects of the game. These variables include the window size, pixel size, tile size, frames per second (FPS), track layout, car properties, and parameters related to the genetic algorithm used in the game.

The code loads images for the game, including ground tiles, wall tiles, and a car sprite. These images will be used to render the game elements on the screen.

A car class is defined to represent a car object in the game. Each car has various attributes such as position, angle, brain (an array of inputs), velocity, acceleration, turning speed, and score. The class also includes methods to handle inputs (acceleration, deceleration, turning), move the car, detect collisions with walls, handle checkpoints, calculate the score, and mutate the car's brain.

Utility functions are defined to support the game logic. For example, there is a function called biasedRandom that generates random numbers with a bias. Additionally, there are functions named drawGround and drawWalls which are responsible for drawing the game elements, such as the ground and walls, on the screen using the PIL library.

The code defines a function called playerInput which handles player input using keyboard keys. This function allows human players to control the car in the game.

The main game loop starts by initializing pygame and setting up the game window with the specified size.

A list named cars is created to store the instances of the car class. The track layout is also initialized.

The loop runs until the quit variable is set to True. Each iteration of the loop represents a game tick.

Within the loop, the code checks for events such as keyboard input and quitting the game. It updates the position and state of each car object, calculates their scores based on their performance, and handles genetic

operations such as selection, crossover, and mutation to evolve the population of cars over generations.

The loop also includes code to draw the game elements on the screen. This involves rendering the background, cars, and checkpoints using loaded images. The display is updated, and the frame rate is limited using pygame functions.

Overall, the code provides the foundation for a game where cars navigate through a track, learn to improve their performance through a genetic algorithm, and are rendered on the screen for players to observe and interact with.

SECTION IV: GENETIC ALGORITHM IMPLEMENTATION

The code represents a racing game with a genetic algorithm to find the best racing line. It utilizes the Pygame library for graphics and user input. The main objective is to navigate a track and reach the finish line while avoiding collisions with walls.

The code begins with various import statements to include necessary libraries and modules such as Pygame, math, random, and PIL. These libraries are used for game development, mathematical calculations, random number generation, and image processing.

Next, there are several global variables defined, such as window size, pixel size, tile size, frames per second, and various configuration settings for the game. These variables determine the dimensions of the game window, the size of tiles and pixels, the game's frame rate, and various gameplay parameters.

The track is represented by a two-dimensional array, where each element corresponds to a specific tile on the track. The track layout is defined using numerical values to represent different types of tiles, such as ground tiles and wall tiles. Checkpoints are also defined as coordinates within the track array.

The code includes a car class that represents the player's car and its behavior. The car object has attributes such as position, angle, brain (a sequence of inputs), velocity, acceleration, and various other parameters. The car can accelerate, decelerate, turn left, turn right, and move based on user input. It also has collision detection to check for collisions with walls and handles collecting checkpoints.

There is a biasedRandom() function that generates a random value within a specified range, biased towards the maximum value. This function is used in the genetic algorithm for selecting parents and mutations.

The code includes functions for drawing the ground and walls of the track on the game window. It also includes a function for rendering the background image, which combines different tiles and overlays to create a visual representation of the track. Another function renders a collision map, which indicates collision boundaries within the track.

There is a function for player input that handles keyboard events and translates them into movement commands for the player's car. It checks for specific key presses to accelerate, decelerate, and turn the car.

The main game loop is where the game logic is executed. It initializes the Pygame library, sets up the game window, creates the car objects, and starts

the game loop. Within the loop, it handles events, updates the game state, and renders the graphics. It also implements the genetic algorithm by evolving and mutating the car population based on their performance.

Finally, the code concludes with the Pygame setup, including clock management, window positioning, and the display mode. It creates the game window, loads image assets for the ground, walls, and car sprite, and sets up the environment for the game.

SECTION V: AI RACERS

The necessary libraries are imported, including pygame for game development, math for mathematical operations, random for generating random numbers, os for interacting with the operating system, and PIL (Python Imaging Library) for image processing.

Various variables are set, such as window size, pixel size, tile size, frame rate, colors, track layout, car settings, and genetic algorithm parameters.

The car sprite and tile images are loaded using pygame.image.load().

The car class is defined, representing the cars in the game. It has methods for handling inputs, movement, collision detection, checkpoint handling, score calculation, and drawing.

The biasedRandom() function is defined to generate a random number biased towards the maximum value within a given range.

Functions for drawing the ground, walls, and background images are defined.

Functions for rendering the background, collision map, grass map, and tree overlay are defined.

The player input function playerInput() is defined to handle user controls.

The main game loop is started using pygame.init() and pygame.display.set_mode().

The track layout, starting position, and checkpoints are defined.

The car objects and their initial positions are created.

The game loop begins, which includes event handling, updating the game state, drawing elements on the screen, and controlling the frame rate using clock.tick().

SECTION VI: GAME ENVIROMENT AND PHYSICS

The game environment in this code is designed to simulate a racing track. The track layout is defined, specifying the shape and path that the cars will follow. The track may consist of straight segments, curves, and various obstacles to add complexity and challenge to the gameplay.

Regarding the physics system, the code implements collision detection to handle interactions between the cars and the track or other obstacles. The car class has methods for collision detection to ensure that the cars stay on the track and do not pass through walls or other objects

The code also includes vehicle handling functionality, allowing the cars to respond to player inputs and move accordingly. The car class handles the movement of the cars based on user input, such as steering and acceleration. The specific mechanics of the vehicle handling, including acceleration, deceleration, and turning behavior, are implemented within the car class.

Overall, In this part we provides a basic framework for a racing game environment and implements essential physics elements such as collision detection and vehicle handling.

SECTION VII: TESTING AND EVALUATION

The testing and evaluation process for the genetic algorithm and the game itself involve several steps to assess the performance and effectiveness of the algorithm in generating optimal car behaviors. Some of the key aspects of the testing and evaluation process are outlined below:

1. Testing the Genetic Algorithm:
    - The genetic algorithm is tested by running multiple generations of cars and evaluating their performance on the track.
    - The fitness function is defined to measure the performance of each car based on criteria such as lap time, checkpoints reached, collision avoidance, etc.
    - The algorithm's ability to produce better-performing cars over successive generations is evaluated by comparing the fitness scores and behaviors of the cars in each generation.
2. Evaluating the Game:
    - The game itself is evaluated by observing the gameplay experience and assessing various performance metrics.
    - Average lap times can be measured to evaluate the overall speed and efficiency of the cars.

- The number of checkpoints reached can be used to assess the cars' ability to follow the track and navigate successfully.
- Collision frequency and avoidance can be evaluated to determine the effectiveness of the collision detection and handling system.
- Other metrics, such as the smoothness of the car movements, can also be considered to evaluate the realism and responsiveness of the gameplay.

3. Performance Metrics and Improvement:
    - Performance metrics, such as average lap times, can be tracked across generations to measure the improvement in car behavior over time.
    - The fitness scores of the cars can be analyzed to identify trends and improvements, indicating the algorithm's ability to optimize car performance.
    - Comparing the performance of the best-performing car in each generation can provide insights into the progress and effectiveness of the genetic algorithm.
4. Challenges and Limitations:
    - During development and testing, various challenges and limitations may be encountered.

- Optimizing the genetic algorithm parameters, such as mutation rates or population size, can be a complex task and may require iterative testing and tuning to achieve satisfactory results.
- The track design and obstacles may need careful consideration to ensure a fair and challenging gameplay experience.
- Balancing the difficulty level of the game and ensuring that the cars exhibit realistic behaviors can be a challenging task.
- The limitations of the genetic algorithm approach, such as convergence to local optima or sensitivity to initial populations, should be considered and addressed if necessary.

Overall, the testing and evaluation process involves assessing the performance of the genetic algorithm in generating improved car behaviors, evaluating the gameplay experience through various performance metrics, and addressing challenges and limitations encountered during development and testing. This iterative process helps refine the algorithm and enhance the overall quality and enjoyment of the game.

SECTION VIII: CONCLUSION

In conclusion, this project successfully demonstrates the development of a car racing game powered by a genetic algorithm. The game environment provides an immersive experience with a well-designed track and realistic physics. The genetic algorithm optimizes car performance, resulting in improved lap times and efficient navigation. The testing and evaluation process provided valuable insights into the algorithm's progress and game performance. Despite encountering challenges, such as parameter tuning and bug fixing, the project delivers a robust and enjoyable gaming experience. Overall, this project showcases the integration of game development and genetic algorithms, opening doors for further advancements in artificial intelligence and game design.