Rochester Institute of Technology

B. Thomas Golisano College of Computing and Information Sciences

Master of Science in Game Design and Development

Capstone Final Design & Development Approval Form

Student Name: **Alabhya Maheshwari**

Research Title: **Race Car AI**

Keywords: **Racing, Simulation, Strategy, Falcon, Engine, AI**

Christopher A. Egert, Ph.D.

Lead Capstone Advisor

Erika Mesh

Capstone Process Advisor

Jessica Bayliss

Faculty [Advisor]

Chris Cascioli

Faculty [Advisor]

Jesse O'Brien

Faculty [Advisor]

David Schwartz, Ph.D.

Director, School of Interactive Games and Media

Race Car AI

Introduction:

This article covers a brief review of various AI techniques that get employed in generic racing games. It also explores the design choices that would be suitable for simulating races in The Falcon Project.

Background:

The Falcon Project is a racing-themed simulation game developed over the custom-built game engine, the FalconEngine. The game puts the player in the shoes of a manager of a motorsports outfit. They are responsible for developing their cars and facilities, managing staff, finances, sponsors, and on race day, managing their drivers throughout the race. We'll dive deeper into the game mechanics revolving around the race day for the scope of this research. The player sits on the pitlane and receives all the race data in real-time. The data includes their drivers' position on track, weather conditions, track temperatures, their drivers' fitness levels, and the health of the car's various components like engine, tires, gearbox, brakes, chassis, and aero. The player must read into this data in real-time and react to evolving conditions to instruct their drivers with the most optimal race strategy to snatch a victory.

Being a motorsport simulation, the game's crux revolves around AI-agents racing around the circuit. The players provide high-level instructions to the player-controlled AI agents that influence their driving style and other racetrack decisions. There are two aspects of this simulation that required the employment of a game AI. The first one is the cars themselves on the racetrack. For ease of understanding, we call these agents 'drivers'. The second aspect of the simulation is the AI system that gives instructions to the 'drivers' not controlled by the player. We call this the 'manager'. The manager's role is to behave just like the player would to what is happening on the racetrack. For instance, a driver on Team A has locked up during the race and has worn off his tires, slowing their pace and putting them at risk of tire failure. In this situation, the 'manager' must adapt and pit the driver for fresh tires while making changes to their overall race strategy to ensure an optimal result for them.

Figure 1 The Falcon Project

AI techniques used in Racing games:

We can broadly classify racing games into two categories based on the realism involved. Arcade games lean more towards creating a fun racing experience for the players without sticking to realistic physics behavior. Games like Crazy Taxi, Mario Kart, and TrackMania are some excellent examples. The other side of the spectrum is full of games that strive to achieve realism, making the player's experience as close to driving a real vehicle on track as possible. iRacing, Assetto Corsa, rFactor are a few that fit this category. Both categories approach the AI in a similar fashion with the primary goal to make the experience for the user as satisfying as possible with AI as their competition.

Pole Position, released by Namco in 1982, is the first single-player racing game that involved an AI. The early games approached race AIs with basic steering and pathfinding between two points while avoiding obstacles. Racing lines was another concept utilized a lot in the earlier games wherein a car would follow a spline between nodes placed on the circuit by the developers. The vehicles would follow the line and stay on the same path with minimal steering to avoid obstacles. While this is a cheap tool to create and far less CPU intensive than modern AI techniques, cars' behavior following each other in a line is not realistic.

Figure 2 Pole Position (Namco)

Modern racing games follow a multi-tier AI architecture where the car's movement is separated from the strategic and tactical level of AI that handles the decision making. We can flesh out the decision-making layer with finite state machines where goal states are known or using behavior trees to achieve the same results with better scaling in terms of readability and complexity. For example, choosing the driving line on the track between various pre-determined racing lines or changing driving style based on their position on track relative to other cars. Alternatively, we can utilize Markov or Fuzzy state models to create desired steering inputs for the movement layer to resemble real drivers' driving style.

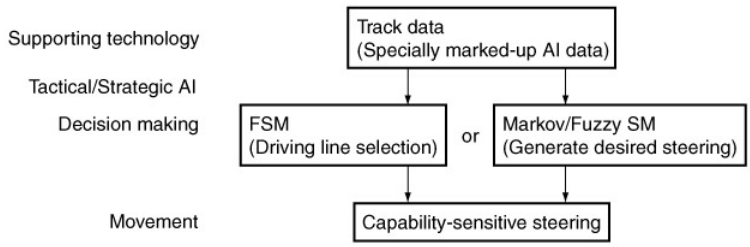

Figure 3 AI Architecture for Racing Games

Another concept popular in racing games is rubber banding. With the end goal of making the game adapt to players of all skill levels, the AI agents have their skill/speed parameters dynamically updated throughout the race to keep up with the player when they're ahead or slow down when they're behind. This is essentially more of a technique to balance the game and make the gameplay experience more enjoyable for the players.

With more advancements in the field of AI, driven by reinforcement learning, racing games have also adapted to this approach. Ubisoft recently employed the soft actor-critic architecture in their game, The Crew, where they successfully trained their model with a hybrid set of actions: continuous (acceleration and steering) and binary discrete (hand brake). Codemasters achieved a similar feat by training their AI with datasets developed from real-world race drivers. Their latest game GRID features driver personas built around real-life drivers and adapts their behavior based on in-game actions.

Figure 4 The Crew 2 (Ubisoft)

Figure 5 GRID (Codemasters)

Falcon Engine Physics System:

The FalconEngine is the game engine developed by the team for this project. The core system that we deal with here is the physics engine. We integrated NVIDIA's PhysX to provide us with a solution that can replicate physics for arcade games and games leaning towards realistic simulation. Another advantage of choosing PhysX was its availability on Unity that would allow us to prototype on Unity in parallel with the development of the FalconEngine itself. The PhysX system also includes a Vehicle SDK that lets you set up any four-wheel vehicle with its parameters. The focus of vehicle physics is the simulation of tires and the forces being applied to the car. The game AI needs to be developed while keeping the limits of these elements like the speed of cornering, the grip of tires, braking, etc., in mind.

AI Design for The Falcon Project:

Based on the research into AI techniques that can be employed for simulating cars on the racetrack in The Falcon Project, we decided to adopt a simple architecture that can be developed on the FalconEngine. The AI system is separated into two tiers. The base tier is the '*driver*' that handles the movement of the car on the track. The upper tier is the 'manager' that takes the strategic and tactical decisions during the race.

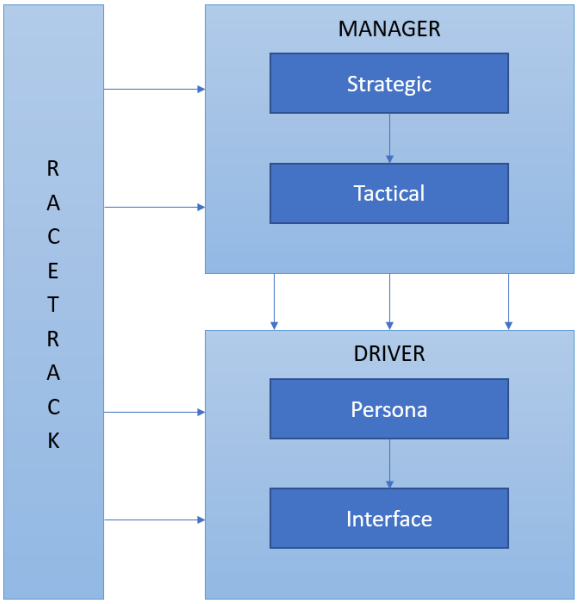

Figure 6 AI Architecture for The Falcon Project

Driver:

Each driver is again broken into two layers.

- *Persona:*

The top layer forms the persona of the driver. This is made up of individual driver attributes that determine their skills. It also incorporates certain driver perks that filter out the manager's decision before passing it on to the interface. For example, if a driver has the perk for not taking tactical instructions given by their manager, they would ignore any inputs and pass default values to the interface.

- Interface:

The bottom layer is the interface that controls the movement of the car. It is designed to behave identically to how a human would drive a vehicle with an input system. It accepts inputs for steering values, acceleration, and braking. It also extracts metrics from the physics model to adjust its inputs to keep the car in control on the track. To make the steering more realistic, we can include some level of randomization. The vehicle does not stick to the racing lines throughout the race. These racing lines act more as guides for them. The interface looks ahead and steers to maintain its position over these racing lines. This look-ahead distance changes based on obstacles and turns ahead. Input from the racetrack assists in determining the look-ahead distance as well. The look-ahead distance would be automatically shortened while approaching a hairpin turn till the car exits the corner. The interface also takes in physics data to ensure that it is not oversteering or understeering constantly. The steer values are adjusted with a correction value to prevent this.

Manager:

The manager layer is the strategic layer that sits on top of the driver. It encompasses a set of predetermined states that a real racing manager would instruct the driver to adopt during a race. We again break this tier into layers.

- *Strategic:*

The strategic layer handles decisions like pitting, fixing or replacing components, pushing, conserving or allowing traffic through, etc. While the player controls these instructions for his team, we can program an AI similarly to match the states a player can adopt for their cars. A few of these strategies would be hardcoded initially to model their behavior on ideal race strategies.

- *Tactical:*

The tactical layer is responsible for controlling what the drivers are doing on the track based on their strategies. It is reviewed on every update of the strategic layer. Based on this, we fleshed out a few behavior patterns on which we can model FSMs or BTs to achieve an autonomous system.

i. Default: In the default state, the racing car follows the optimal racing line through the track without pushing the car over the limit. They maintain a neutral approach in conserving their components and will hold position behind another car on the same line provided their relative speeds are the same.
ii. Overtake: The car would transition into an overtake state when it either receives an instruction from the strategic layer or has an opportunity to make an overtake due to slower moving traffic ahead. In this state, the AI would reevaluate the position of the car ahead on the racing line and look to move onto an alternative racing line to make a pass. The zones where such passes occur can be along long straights or during corners. With another car in front initiating defensive driving with active blocking, the AI must analyze the window in which an overtake is suitable without running out of the track or colliding.
iii. Defend: During defensive driving, real life drivers abandon the optimum racing line when another car is following closely and position themselves on the track in such a manner that they have better traction out of corners and put their opponents out of position. Once again, the AI must analyze the window of opportunity here to make a successful defensive maneuver without compromising their pace or colliding.
iv. Conserve: The conserve state decreases the driver aggression and increases the life of all components on the car. This of course comes at the price of reduced race pace.
v. Push: Opposite of conserve, the driver pushes the car to the limits ignoring the wear on the car components.
vi. Recover: In case the car runs off the track or spins on it for any reason, the car must rejoin the track in a safe manner. This includes analyzing incoming traffic onto the racing lines.
vii. Branch: This state would instruct the car to switch from the racing line to the pit line at the pit entrance and then perform a stop in front of their garage before mechanics work on it and release it. At the end of the release, the car would transition again into the default state and return to the track.

Racetrack:

The racetrack is divided into sectors that allow for more accurate tracking of the position of a vehicle in real-time. It also lets us calculate speeds and sector timings set by individual vehicles. Each sector holds the information of its distance to the start/finish line. They also store information on racing curbs, corner types (hair pin), and braking zones allowing a car to adjust its parameters while approaching a specific sector. For example, a car approaching a sector with a flag for the braking zone will start braking and slowing down to make the corner's apex that follows. At the end of the corner, the next sector would indicate a throttle zone where it will accelerate at its maximum to get out of the corner.

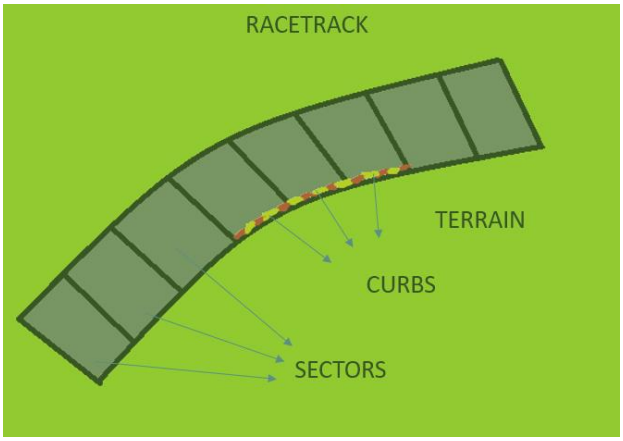

Figure 7 Track Layer

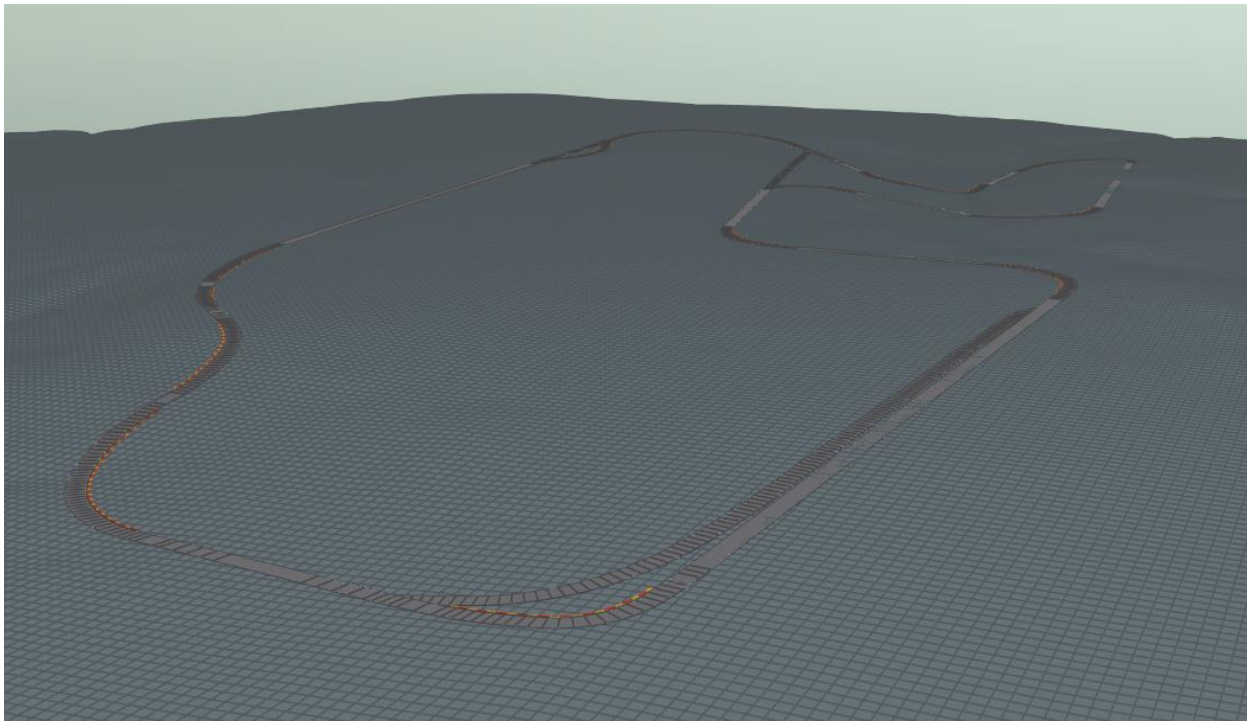

Figure 8 Racetrack and Terrain broken into sectors and grids for containing track information

Discussion:

We implemented the lookahead waypoint system on Unity that yielded satisfactory results for having an AI car follow a predetermined route across the racetrack without crashing into other vehicles or going off the racetrack. With a finite state machine, the car's behavior was modified on track yielding in different lap times being achieved based on the strategy employed. Implementation of the same waypoint system on the FalconEngine is achievable with the framework laid out to support it.

Figure 9 Waypoint System with a Tactical layer implemented on Unity

Conclusion and Future Work:

The game intends to simulate racing behavior as real-life race drivers would race, and while some of it can be achieved by hard coding specific patterns, others can be introduced by adding dynamic state transitions as and when optimal conditions for the transition exist.

We tested the strategic AI with an FSM, but we can further carve it out with an alternative that better fits this test case. Goal-Oriented Action Planning is one of the techniques we looked at during the development, and while we were able to replicate the behavior that can be reproduced with an FSM on a separate project, it's still a work in progress for this model.

References:

[1] Artificial Intelligence for Games, Second Edition, By: Ian Millington, John Funge

[2] Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques, By: Marvin T. Chan, Christine W. Chan, and Craig Gelowitz

https://www.hindawi.com/journals/ijcgt/2015/839721/

[3] Behaviour Trees for decision-making in autonomous driving, By: Magnus Olsson

https://www.diva-portal.org/smash/get/diva2:907048/FULLTEXT01.pdf

[4] Ubisoft uses AI to teach a car to drive itself in a racing game, By: Kyle Wiggers

https://venturebeat.com/2019/12/27/ubisoft-uses-ai-to-teach-a-car-to-drive-itself-in-a-racing-game/

[5] NPC Braking Decisions for Unity Racing Game Using Naïve Bayes, By: Muhammad Aminul Akbar

https://www.researchgate.net/publication/337980561_NPC_Braking_Decision_for_Unity_Racing_Game_Starter_Kit_Using_Naive_Bayes