UP814853 – CT6GAMAI Report

CT6GAMAI - Report

By UP814853

Contents

Contents

[Introduction 1](#_Toc27050232)

[Analysis 2](#_Toc27050233)

[Pathfinding 2](#_Toc27050234)

[Steering Behaviours 2](#_Toc27050235)

[State Machine 3](#_Toc27050236)

[Reflection 3](#_Toc27050237)

[Main Results 3](#_Toc27050238)

[Issues 3](#_Toc27050239)

[Self-evaluation 4](#_Toc27050240)

[References 4](#_Toc27050241)

# Introduction

Racing simulations in video games are AD-HOC, for a purpose, system which simulates real-life driver's behaviour based on factors created by the programming team such as personality, vehicle type, race track and more. These features are the primary focus of the AI development team when creating a single-player racing game.

The problem this project focused on was creating a racing game where Artificial Intelligence would race each other around a circuit and consider strategy based on their personality. The main problems would be steering behaviours, state machines and pathfinding.

Creating realistic Artificial Intelligence for racing games is essential. The project needs the Artificial Intelligence to be able to race each other but fun to race. The Artificial Intelligence that is raced against can either make or break a game.

Forza Horizon 4 (Turn 10 Studios, Playground Games, 2018) and F1 2019 (Codemasters, Swordfish Studios, 2019) are two of the most popular and state of the art racing games both having advanced racing Artificial Intelligence which can be considered to be state of the art. Forza Horizon 4 (Turn 10 Studios, Playground Games, 2018) even has a system in place where the driver profiles are built using a Learning Artificial Intelligence system (Forza Support, 2019).

Both of these games are considered to be the state of the art as they successfully implement fun and challenging Artificial Intelligence systems within their games. These AI can react to the best of players and can be modified by the players within menus.

For the artefact focused on getting the Artificial Intelligence to go around a given track. The AI should also be able to pit when they need fuel based on desires. With the three central systems being Steering Behaviours, Pathfinding and State Machines.

# Analysis

## Pathfinding

Pathfinding can be done using multiple algorithms each used for different results and purposes. The most notable are Breadth-First Search, Depth First Search, Dijkstra's algorithm and A\*

For the pathfinding, A\* Search algorithm was chosen. The technique was chosen as it is the quickest way to search a graph speeding up Dijkstra's algorithm (Buckland, 2004). It is also the standard within the video game industry being the most optimal technique currently available to developers.

The project needed pathfinding so the artificial intelligence would race around the track in the correct direction as well as take the shortest route to go around the track. The primary feature of the A\* was checking if there is already an existing route, this would save performance and time and ensure they wouldn’t search the same two nodes twice. The A\* was done using an adapted priority queue (McCaffrey, 2012) using key value pairs.

The secondary feature was a dynamically generated grid based on the area and gap between the nodes and this would generate a grid map of tiles to be searched in later function. This grid would allow the AI to race around the track in the correct direction and ensure they would not hit walls.

A pro with A\* pathfinding is the speed, compared to Dijkstra’s A\* is more efficient in its search (Millington & John, 2006) and has no need to search every node in the graph. The second pro with A\* is the cost-based search it has, the programmer can change and adapt these costs, the project used the Manhattan distance as well as the distance between each node to calculate the costs. The Manhattan distance was chosen as it is more efficient and effective then the Euclidean distance having no square functions needed and also, they avoid obstacles (Buckland, 2004) . More expansive systems like Unity (Unity Technologies, 2018) allows the developer to set nodes as jumpable, allows them to adapt the costs of nodes as well as bake the nav-mesh saving on performance.

A con with the project’s implementation of A\* is the use of contains, the use of contains can be expensive even with the use of hash sets as it has a complexity of O(n) (Huan, 2010) , another is the use of a key value pair list, this could have been done with a dictionary instead as this is a collection of keys and values (Microsoft, n.d.).

## Steering Behaviours

Steering behaviours are techniques to control the AIs movement in certain conditions. Techniques can include obstacle avoidance, wall avoidance, evade, seek, join, flock, wander and more. The techniques were adapted from Programming Game AI by Example (Buckland, 2004) and were changed to work within Units.

The project needed comprehensive steering behaviours in order to be an effective racing game, the steering behaviours implemented in the project where seek and obstacle avoidance. As a racing game the project required these two features to be implemented so the AI would go around the track and be able to overtake each other.

The primary feature was obstacle avoidance being the more complex out of the two, it would check if anything was within a box and create a force to go around it.

The pro of using steering behaviour’s is to be able to control how the AI moves to a location, how it avoids objects and other behaviour’s based on set factors (Buckland, 2004). Steering behaviours are adaptive and can easily be changed by development teams with little to no issues.

The con of using steering behaviour’s is without calculations such as the Weighted Truncated sum (Buckland, 2004) there is no way the developers to guarantee that a certain force will be to powerful, in the project certain AI would go into the walls as the overtaking force was to powerful and the wall avoidance would be to slow to act.

## State Machine

The AI logic could have been done using three systems, State machines, Fuzzy logic or goal-driven/behaviour trees.

The project used State machines as they would be the most effective towards the game. The AI would assess their desires using utility theory, and then the strongest desire was chosen and executed this way, foundations were made to be able to have dynamic profiles (Buckland, 2004). As the AI was driving around a track, the project only needed three desires, Driving, Overtake and Pit. Their driver profile with the default controlled each AI only wanting to overtake if they meet certain conditions and pitting if they estimate they do not have enough fuel to finish the race.

State machines were chosen as they were simple to implement and design (Yannakakis & Togelius, 2018) and as the project was small, there was no need for more complicated systems such as Fuzzy Logic or goal-driven systems (Yannakakis & Togelius, 2018).

The state machine was designed within three states it could be in, each state was unique, but all of them drove the car around the track ensure the AI still raced. The three states would be controlled by their desire which was updated in their driver profile.

State machines allowed the implementation to be available and straightforward to be abstracted to be used by different driver profiles with the scope of having profiles like aggressive, defensive, strategic and default

The primary feature was the utility theory and being able to control states using desires which leads to more dynamic behaviour (Buckland, 2004) when using the systems. The state machine would always choose what it wanted to do based on the highest desire and lead to an AI which can make simple calls based on the strategies given to it.

The pro of state machines was for a small system it is easy to design and maintain the code and still be readable, the desires are easier to implement and there is no need for any advanced system. However, when the project gets larger and the more states there is state machines become less effective and goal driven behaviours should be used as they are better designed for the bigger systems involved (Yannakakis & Togelius, 2018).

# Reflection

## Main Results

The goal of the project was to create a project which would have a basic AD-HOC racing system for a PC game. The AI behaves in a set pattern according to the driver profiles given to them and the vehicles they are racing. The game has a set amount of random generation within it for the vehicle parameters. The project will simulate the race; however, breaking not being implemented in the vehicle with the highest speed would be the one to win the race most of the time. The AI drive to the route and follow it well, and the state machine controls their desires and pits them when they are needed. The faster AI will always try and overtake the slower AI to win the race.

## Issues

Pathfinding in racing games is an incorrect implementation. The project should have instead just used steering behaviours to simulate the race, pathfinding has caused the biggest issue with the project with the AI trying to overtake each other if they miss a seek checkpoint and it is not removed they may turn around or suddenly turn left essentially given up that position. //Use a combination of the two (path following )The solution to fix them turning backwards on themselves is to use a dot product to check if the agent is ahead of the node if it is then removing that target location. The other solution would be to implement overtaking on straights only and have the AI loosely follow the path when it is overtaking and path find when its finished. This has created unforeseen moments in races where due to this incorrect implementation, the AI have weird behaviour. Often in the project, the AI would turn around or not race at all because the pathfinding was not behaving.

Pathfinding would cause the most lag and whenever the game freezes its due to A\* and there being too many nodes however removing pathfinding would break most systems in the game. The average graph in the game had 700 nodes would take a second to search however implementing unities nav mesh (Unity Technologies, 2018) system would of reduced the number of nodes searched to an estimated 50, this would greatly improve the performance, however unities nav mesh needs to be baked and does not like being generated on the go, meaning the implementation would be more complex,

//Re write this for behaviour trees

State machines in this project could have been done with more comprehensive behaviour trees (Yannakakis & Togelius, 2018). Using behaviour trees allows programmers to create pre-designed modular systems that can be added in or taken out by non-programmers with ease, they are used for simple systems such as making an AI picking up a sword.

The main limitation of this project was time. Due to the project having two months and conceptually created with a month till hand in the project was created to be a foundation of a racing game however are missing implementations such as defending, random items, shortcuts, different driver profiles, randomly generated tracks and procedural generation of the game world.

In extreme cases, the AI likes to dart diagonally into a walk and slam into its multiple times. Developers theorise this is caused by the steering behaviours not avoiding walls as the pathfinding done it for them and when they slammed into a wall there is no programming implemented for them to rescue themselves.

## Self-evaluation

Within 2 months and no previous knowledge of the systems I usefully created a racing game with 8 AI agents racing each other, considering strategy and having systems implemented for the future of the project. While there is bugs with the pathfinding, route following and certain steering behaviour’s considering the project was created a month ago the progress made so for is sufficient enough to be considered successful. However, I spent too much time on feature creeping in the project and trying to experiment to find out how certain things worked. This led to the current issues with the pathfinding and steering behaviour’s as I got them working and just ran experiments on them instead of working on features of the game. I also only got certain features working at a basic level instead of the advanced level I was aiming for.

My time management was a mix of either spending to much time on this project or focusing on getting a tile-based system to work when there were other better methods out there. Most of my time was spend creating a feature then experimenting with it instead of developing that feature into the game it was left out as I had no time left to fix any of these issues.

If I was to do the project again I would spend more time with the research aspect of it ensuring I was well informed, I would also write code which follow programming patterns more and plan the project out so I know what I need to do within the project.

In conclusion, with two months, the project successfully created a racing simulator where 8 AIS race to a lap counter, pit and use created techniques to race each other. The AI use obstacle avoidance to overtake each other, state machines to decide what they are doing and pathfinding to find out where they are going. The project could have been done better with more time, and on reflection, specific systems were not the best implementation however for what the project needed and with the time given the project is a racing game with small bugs which can be ignored.

# References

Buckland, M. (2004). *Programming Game AI by example.* Sudbury: Jones and Bartlett Publishers, Inc.

Codemasters, Swordfish Studios. (2019, June 19). F1 2019. *F1 2019*. Southam, Warwickshire, United Kingdom: Codemasters.

Forza Support. (2019). *Drivatars*. Retrieved from Forza Support: https://support.forzamotorsport.net/hc/en-us/articles/360005302934-Drivatars

Huan, J. L. (2010, March 28). *Runtime Complexity of .NET Generic Collection*. Retrieved from C# Snippets: http://c-sharp-snippets.blogspot.com/2010/03/runtime-complexity-of-net-generic.html

McCaffrey, J. (2012, Febuary 11). *Priority Queues with C#*. Retrieved from Visual Studio Magazine: https://visualstudiomagazine.com/articles/2012/11/01/priority-queues-with-c.aspx

Microsoft. (n.d.). *Dictonary* . Retrieved from Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.8

Millington, I., & John, F. (2006). *Artificial Intelligence for Games.* CRC Press.

Nillson, N. J., Hart, P. E., & Raphael, B. (n.d.). A\* search algorithm. *A\* search algorithm*.

Rusildo. (2015, September 3). Wander steering behaviour in 3D. Retrieved from https://gamedev.stackexchange.com/questions/106737/wander-steering-behaviour-in-3d

Turn 10 Studios, Playground Games. (2018, September 2018). Forza Horizon 4. *Forza Horizon 4*. Redmond, Washington, United States Of America: Xbox Game Studios.

Unity Technologies. (2018, October 10). *Navigation and Pathfinding*. Retrieved from Unity Documentation: https://docs.unity3d.com/Manual/Navigation.html

Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games.* Cham: Springer International Publishing AG.

# Appendices

Appendix A – GitHub Project

<https://github.com/KieranGrist/CT6GAMAI>

Appendix B – Mat Buckland’s code

<https://github.com/HEP85/game-ai>