CIT2213 - Game Engine Architecture

By Nat Waterworth

My Planned Guide

Green - Must be Implemented

Orange - Plan to be implemented

Yellow - Aspire to be implemented

Red - Won't be implemented

GameTypes	Base Game Features	Common Game Features	Unique Game Features
Stunts	Physics	Upgradable vehicle performance	Rewindable gameplay to previous game state
Battle Arena	AI	Recordable gameplay stats	Vehicle Weapon Attachments
Highscore based vehicle runner	KB and Controller Support	Interchangeable Vehicle perspective	In game Pick ups (boosts, spikes…)
Race game		Unlockable Components (new vehicles - preset stats)	In game Events (Avalanche, collapsing bridge...
		Race replay	

My Initial plan...

- I focused myself to create a game that implemented the following basic features :
  - Relatively real physics
  - AI vehicle opponents to compete against which provided moderate level of difficulty to beat
  - Controller and Keyboard support
  - Standard race games checkpoint, lapping, positioning and respawning

- Also delegated my time looking at and implementing the following features:
  - Interchangeable perspective (mostly done already)
  - In game pick ups (Speed boosts/Power bar boosts)
  - Vehicle weapon

- Towards the end of my project I found myself struggling to find time for:
  - Recordable gameplay stats
  - Interchangeable vehicle weapons
  - Changeable Race Settings

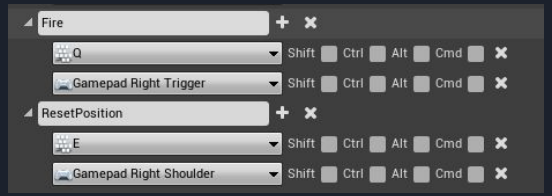
Physics

- The UE4 vehicle template provided most of the core games physics for the vehicles used.
- No additional changes were made.
- The main games vehicle was exchanged for a low poly muscle car and was setup to have the same bone structure.
- A few physics materials were created as terrain types for the vehicle:
    - Standard Terrain - 1.0 Friction - Standard starter vehicle control.
    - Icy Terrain - 0.25 Friction - Lack of user control.
    - Super Unrealistic Terrain - 4.5 Friction - caused chaotic gameplay working nicely with the theme.
- The games physics are stretched between realistic and farfetched, when implementing projectiles unrealistic impulses are added to give the game unpredictability making the player more attentive.

Input - Added inputs

Firing - Button: Q (GamePad: Right Trigger) - Spawns a projectile that upon successful collision with a wheeled vehicle causes a speed reduction and a random impulse along the vehicles X axis. This results in the Vehicle's momentum in the game being reduced and also shifting the vehicle in unpredictable directions causing player to be quick on their feet.

Respawning - Button: E (GamePad: Right Shoulder) - Respawns player at last checkpoint reached in case of vehicle becoming stuck, overturned or in an out of bounds region.




```
PlayerInputComponent->BindAxis("MoveForward", this, &AVehicleTemplatePawn::MoveForward);
PlayerInputComponent->BindAxis("MoveRight", this, &AVehicleTemplatePawn::MoveRight);
PlayerInputComponent->BindAxis("LookUp");
PlayerInputComponent->BindAxis("LookRight");
PlayerInputComponent->BindAction("Fire", IE_Pressed, this, &AVehicleTemplatePawn::Fire);
PlayerInputComponent->BindAction("ResetPosition", IE_Pressed, this, &AVehicleTemplatePawn::ResetPosition);
PlayerInputComponent->BindAction("Handbrake", IE_Pressed, this, &AVehicleTemplatePawn::OnHandbrakePressed);
PlayerInputComponent->BindAction("Handbrake", IE_Released, this, &AVehicleTemplatePawn::OnHandbrakeReleased);
PlayerInputComponent->BindAction("SwitchCamera", IE_Pressed, this, &AVehicleTemplatePawn::OnToggleCamera);
```

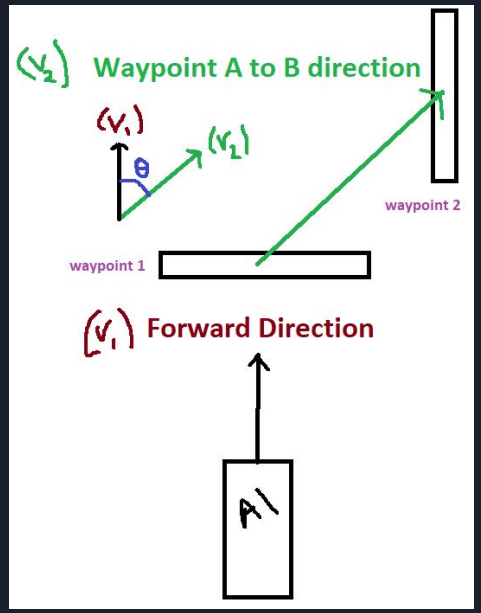
AI Vehicles - Competitors

- The AI are implemented using blackboard and behaviour tree components. An AI Wheeled Vehicle Class.
- The C++ UBTServices created control the vehicle's speed:
  - BTSteeringService - Calculates Throttle and Steering Input values based on AI scenario.
  - BTTaskSteerVehicle - Gets updated steering input from blackboard.
  - BTTaskThrottle - Gets updated throttle input from blackboard.

AI Vehicles - Shooting and Respawning

- AI Raycasts in its forward direction by a few car lengths distance, If raycast hits a vehicle it shoots a projectile at the vehicle. On overlap the projectile causes the vehicle to be flung to either the left or right and reduce the vehicles forward momentum to a set minimum. This is achieved through adding impulse.
- If an AI vehicle is moving less than 1 kph a timer is set to respawn the vehicle at last checkpoint, if vehicle increases its speed within timer parameters (this case 1.5 seconds) the vehicle doesn't respawn. Used in cases where the vehicle has flipped or gotten stuck.

Implementing the Steering Service

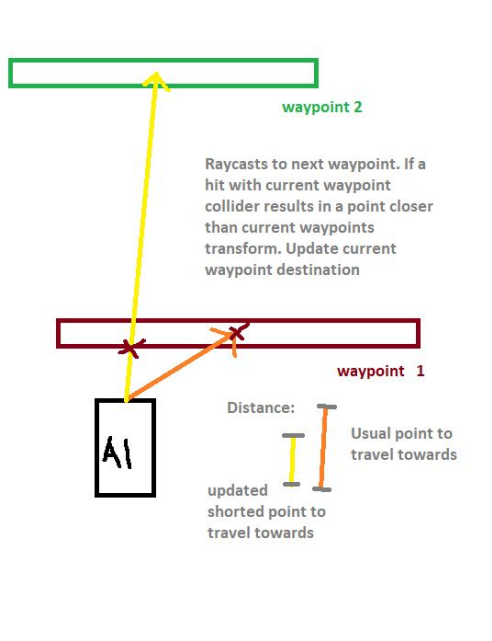


The AI vehicles applies the dot product in conjunction with its forward vector and its destination direction. By rearranging the dot product equation to return the Cos(Angle) (A value between 0 and 1), this is then used to maintain throttle and steering.

Then I determined the dot product between the AI Vehicles forward direction and the destination point after the current destination point it was travelling towards. Also using the Cos(Angle) of this calculation allowed me to determine the throttle value when approaching the current destination point. The waypoint locations obtained from storing all race waypoints in an array on Begin function to access dynamically.

Additionally, with adding in the functionality of braking, I used the standard braking distance calculation to determine when a vehicle should start braking. However this required tweaking as braking would cause the vehicle to spin out at higher speeds. The braking distance would be dependant on the angle between the current waypoint destination and the next so sharper bends would require more speed correction.

Reflecting on the implementation



A problem I ran into was that the AI weren't smart in certain scenarios. For instance if the AI was positioned as shown, It would intend on travelling to a waypoints centre location until overlapping the waypoints collision box at which point it would travel to the next waypoint.

In order to eliminate this problem I introduced raycasting (trace) to the next waypoint from the AIs current position. If the raycast returned a hit that had a shorter distance than the distance to the current waypoints centre, it had a more logical (smarter) path. So the AI wouldn't go out of its way unnaturally.

Implementing this feature also helps the vehicles movement also as it approaches a waypoint as it can provide preemptive steering. However, this is a stronger feature for straights and slight bends. Anything greater receives this preemptive steering too late causing oversteer.

So these cases would have to be dealt with separately. Especially as these are also the cases where a driver would steer wide in order to attack the corner the best way. You can see this demonstrated in the race.

Maintaining Race positions

Both AI Wheeled Vehicle and Vehicle Template pawn store their current waypoint and current lap (The lap being incremented by the transition of last waypoint to first).

In this implementation of the game I only intended upon the 1 player and multiple AI. Therefore the calculations for the player position is in the Vehicle Template pawn class and it is the only implementation of finding a vehicles position as it is primarily used to display on AI.

The position is determined by getting all other vehicles (AI) and comparing current lap, then current waypoint and finally distance to next waypoint. If in any of these cases the player is represented as being further behind the players position is incremented. After comparing against all current vehicles in the race it will return the players position.

Game States, Mode and Instance

I created a game state which contained the different states as ENUMS. I then created a game state manager class to deal what occurs when in the different states.

Using game mode class I maintained checking the winners which mainly involved storing vehicles which had finished the race in an array to retrieve later to use their index as an indication to their finishing position. Additionally, I maintained the constant slight increase of each players boost. In this class it would check players state and accordingly set the game state.

Game instance allows me to access information between levels. For instance when I transition to the Post Race Scene by completing last lap, displaying final positions of players, game instance stores the position information and retrieves it for the UI to display.

Toggling Weapons in game

I have implemented code within the Game Instance, the players and AI to get and set the ability to shoot. Currently the ability to toggle shooting in the game is unfinished and can only be achieved by selecting all the racing vehicles and setting “Shooting Enabled” on and off in editor. This allows the user to see how the AI race with and without the shooting enabled, as when its enabled it can be difficult to judge AIs movements due to chaos unfolding.