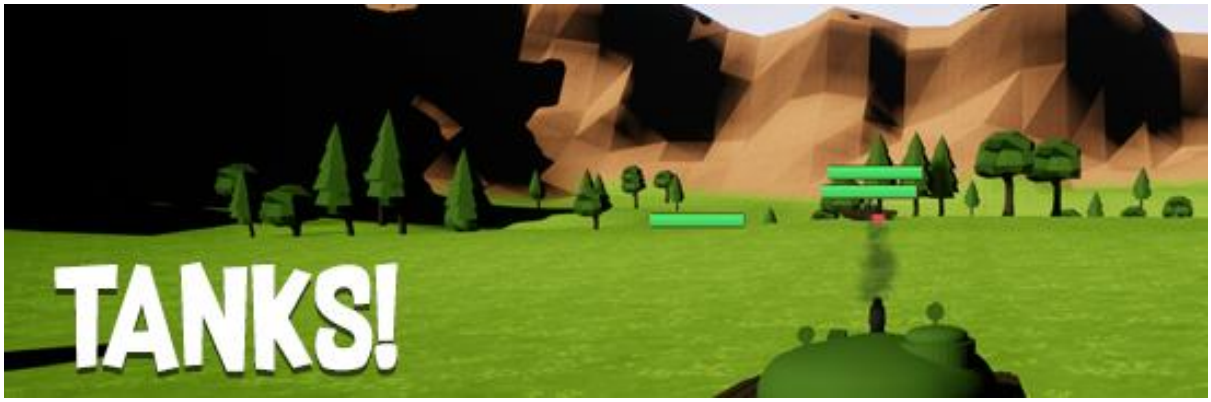
Gameplay Mechanics Report

Ross Adshead – 1502657

https://youtu.be/7kIWiHaOueA



Contents

Summary

In this project, I have developed a number of different small game mechanics which when used together will form a playable tank vs tank game. This includes, fully movable tanks with independent track movement, turret and barrel positioning, AI pathfinding, and tank damage. I have also developed basic user interface elements which are controlled and modified by C++ scripts. For example, there is an ammo counter for the tanks, health bars for the player and AI, and an aiming crosshair which changes colour to show what state the tank is in (reloading, aiming, etc..).

The main inspiration for this project is the video game World of Tanks developed by Wargaming.net. The main aiming mechanic in my project is directly inspired by the way the tanks aim in World of Tanks.

1. Requirements

1.1. Specifications

This project was developed in and designed to be used only with Unreal Engine v4.17.2. The C++ classes were written using Visual Studio 2017 using ReSharper Ultimate.

Recommended Hardware Requirements:

- Windows 7 64bit or later
- Quad-core Intel or AMD processor, 2.5 GHz or faster
- NVIDIA GeForce 470 GTX or AMD Radeon 6870 HD series card or higher
- 8 GB RAM

The mechanics were design to work with an Xbox 360 controller and Keyboard and Mouse. However, they should work with any gamepad.

1.2. Instructions

In this submission, there will be a provided build which you can play as a standalone version. The control scheme is as follows:

Gamepad:

- Left Joystick Up/Down: Moves tank backwards or forwards
- Left Joystick Left/Right: Rotates tank left or right
- Right Joystick: Rotates camera
- A Button or Right Trigger: Fires projectile
- Y Button: Switch between first-person and third-person camera view

Keyboard and Mouse:

- WASD: Moves tank
- Mouse Move: Rotates camera
- Left Click: Fires projectile

1.3. Features

For the project, I created a set of required features that I would like to have developed for the game by submission day. Below is a list of mechanics and a priority which used to decide the order at which I developed them.

1.3.1. Tank Moving

The tank will be able to move forward and backwards using independent tracks. Having independent tracks will also allow for tank turning by applying a force to one and track and applying the opposite force to the other.

Priority: Medium

Priority for movement is only medium because my main focus for this module was the aiming. As said before in the summary, I really wanted to get that World of Tanks aiming style in my mechanic.

1.3.2. Tank Aiming

The tank will be able to rotate its turret and elevate its barrel to point towards wherever the player is looking.

Priority: High

1.3.3. Tank Damage

The tank will be able to shoot projectiles towards where ever it is aiming and if it hits another tank that tank will take damage. If a tank has lost all its health, it will be dispossessed and will no longer be active. This feature will also include UI components for the health bar displaying the current and max health for each tank.

Priority: Low

1.3.4. AI Tanks

AI tanks will be implemented and will move towards the player using the Unreal Engine's nav mesh. They will also always try to aim towards wherever the player tank is situated.

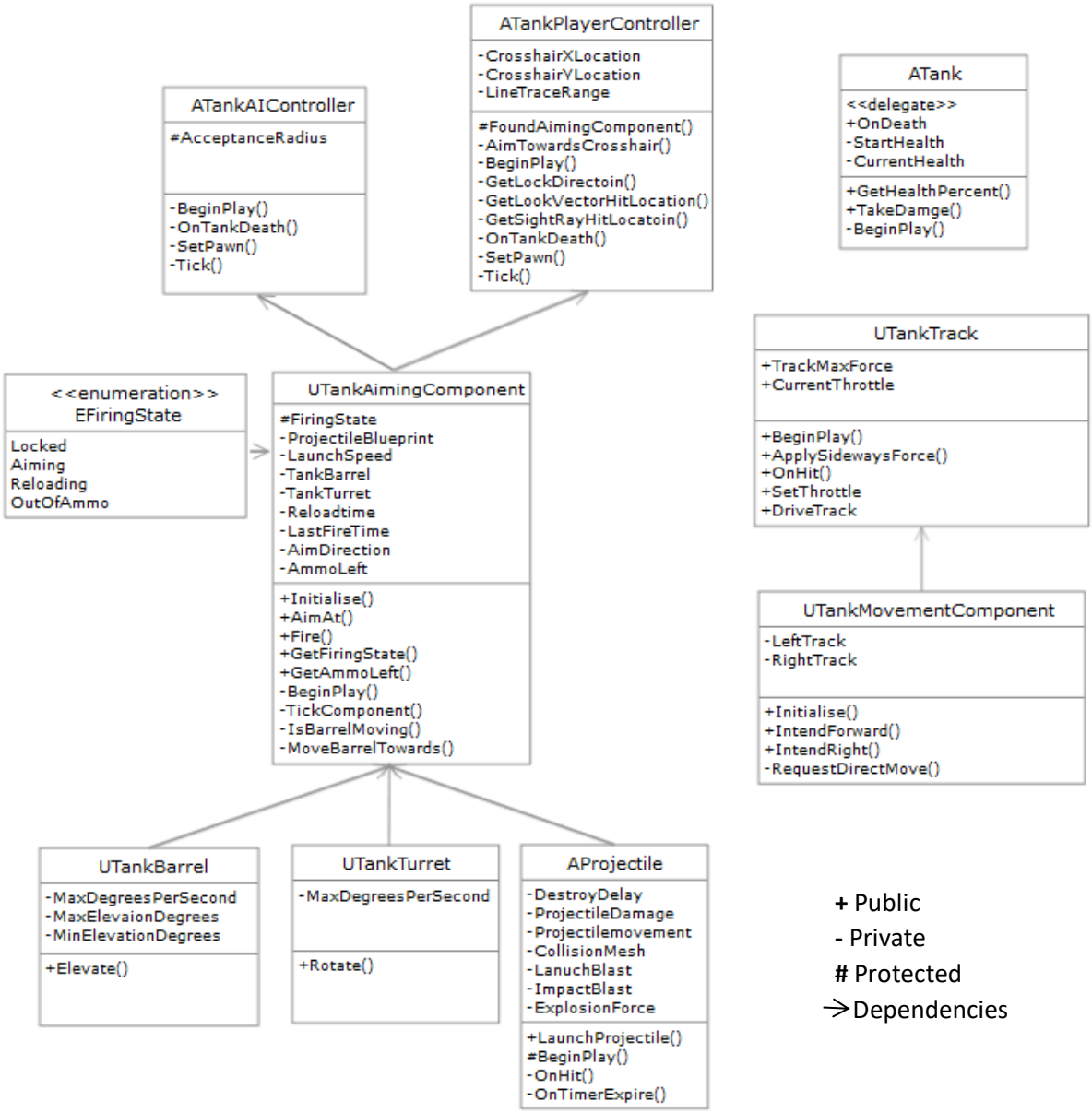
Priority: Medium

1.4. Performance

Before the build was finalised, performance testing was done to make sure that the game would stay a relatively high FPS during runtime. The main concern, was the use of projectiles and their particle effects. Each projectile has two particles effects, a smoke trail effect and an impact blast effect. To test if these would cause significant performance issues, I increased the ammo count for each tank and lowered the reload time allowing me to fire many projectiles in a short space of time. Even with me clicking as fast as I could and several AI tanks firing just a fast, the FPS never lowered below 100.

I also used Unreal's built-in 'TimerManager' to add a timer to the projectile C++ class. This means that after a certain amount of time, set by the designer in blueprints, the projectiles will destroy themselves. For a short runtime, this performance tweak will not affect the performance impact very much. But for longer playtimes, having projectiles stack up without being deleted would eventually start to cause lag.

2. UML Diagram



3. Technical Discussion

3.1. Tank Moving

My project is heavily based around tanks and I wanted to incorporate their track based movement. So instead of applying a general force to the whole tank, I instead apply a force to each track of the tank which moves the tank. This also allows for turning because I can apply different forces to each track therefore if I apply opposite forces to the tracks the tank will rotate.

To apply these forces, I expose two move functions in the 'TankMovementComponent' class to the tank blueprint. I send in a raw axis input from the left joystick to these functions and then I pass those values on to the tank tracks themselves which deal with the actual force. In the 'TankTrack' class the 'SetThrottle' function will clamp the values given by the movement component so that if you try to move forward with both the gamepad and keyboard you won't go twice as fast. With a

throttle value set, the tracks can now apply that to the tank using 'AddForceAtLocation' in the 'DriveTrack' function.

One of the issues found with this method was that if the tank moved sideways the tank would slide as well as rotate. To stop this, I created an 'ApplySidewaysForce' function which gets the dot product of the right vector and velocity resulting in the amount of sideways movement. We get this result because when the velocity is the same direction as the right vector the slide speed will be one. One being 100% of the max move speed in a sideways direction which we don't want for a tank. The function then divides that by two to account for two tracks and applies the opposite as a force which will counteract the sliding force.

3.2. Tank Aiming

One of the main mechanics in my project is the aiming. To achieve this, I have a 'TankAimingComponent' class which as two main functions, 'AimAt' and 'MoveBarrelTowards'. The 'AimAt' function will take in a hit point location which the location the player is trying to look at. The function will use the 'SuggestProjectileVelocity' Unreal function to the needed velocity and direction for a projectile to hit that point. If that velocity is valid, the unit vector of the velocity will be used as the tanks aiming direction which will be passed along to the 'MoveBarrelTowards' function. In the 'MoveBarrelTowards' function, the difference between the desired aim direction and the current barrel forward is calculated and that difference is then used to rotate the barrel and turret.

To get the actual hit point location mentioned before, the 'TankPlayerController' class is used to get the crosshair position in world space and then line trace through that crosshair to get a hit location. The class starts off by finding the crosshair position in the 'GetSightRayHitLocatin' function, and then with this position will get a look direction in the 'GetLookDirection' Function. This function uses the 'DeporjectScreenPositionToWorld' Unreal function which will convert the 2D crosshair position into a 3D position and a direction which is then passed on to the 'GetLookVectorHitLocation'. The 'GetLookVectorHitLocation' function will line-trace (ray cast) from the current camera position in the direction from the previous function and will then return a hit point location within the physical world. The type 'ECC_Camera' was used for the line trace to stop the line trace from hitting the health bars.

3.3. Tank Damage

Applying damage to objects in Unreal is relatively simple, in the case of my project when a projectile hits something it uses the 'ApplyRadialDamage' Unreal function to apply a general damage to any object inside a certain radius. Every actor in Unreal has a virtual 'TakeDamage' function, so when my tanks are in the radius the projectiles explosion the tank C++ class has an override function that will take away a specified amount of health from the tank. Once the tank reaches zero health, the tank will either be dispossessed if it as an AI tank or the game will end and the player will be taken back to the start menu.

To handle tanks dying in script, I used a multi-cast dynamic delegate to broadcast an event to the player and AI controller. The delegate is defined as a public member in the 'Tank' class and when a tank runs out of health points the delegate will broadcast an 'OnDeath' event. To listen for this event, the controllers have a delegate function called 'OnTankDeath' which is subscribed to the 'OnDeath' event when the tank is possessed at the start of the game in the 'SetPawn' function. The 'OnTankDeath' function will now be executed whenever 'OnDeath' is broadcasted. For the AI tanks, their tank will be dispossessed, for the player tank the game will reset back to the start menu.

3.4. AI Pathfinding

For the AI tanks to be able to move towards the player, they need to know where they are, where they need to go, and how to get there. For this, I used the built-in 'NavMeshBoundsVolume' object which calculates a nav mesh for AI components in a given radius. To get the AI tanks to move towards the player, the 'TankAIController' class has a tick function which, during every frame, will find the player tank and then pass it in to the 'MoveToActor' Unreal function. This is a built-in function and does a lot of the arduous work for me but what it will try to do is move the AI tank to the current location of the player and will stop once it hits an acceptance radius which is the minimum distance it has to keep from the player.

To deal with the actual movement request sent by the 'MoveToActor' function, another override function is used within the movement component class. When an AI requests to move, the desired velocity is sent to 'RequestDirectMove' which is overridden and the direction of that velocity will be used the same way as the player's joystick input is. However, in the case of the AI, a dot product and a cross product will be used to keep the AI tank parallel and then face the player.

3.5. UI

A lot of the UI in my project was done using blueprints, but the main elements that are controlled via C++ scripts are the crosshair colour and health bars. The crosshair will change colour depending on the state of the player's tank. For example, if the crosshair is red then it means that player is reloading, if the crosshair is green it means that the barrel is not moving and the tank is ready to fire. To do this in code, I used an enumerator type which I exposed to the blueprint by creating a getter function to get the current state of the tank. The state changes are managed in the 'TickComponent' function using simple logic like "IF AmmoLeft <= 0 (State = OutOfAmmo)".

When it came to health bars, I created a simple progress bar in the widget designer and applied it to every tank blueprint. I then created a simple public 'GetHealthPercent' function which returns a percentage health of the tank which is retrieved in the health bar blueprint which is then used to set the widget UI percent value using a created binding.

3.6. Particles

For the particle effects, I used the starter content particles as a base and then edited them slightly to fit how wanted to function. For example, I took the blast starter particle and added a longer lasting smoke trail which gives the effect of the particle flying through the air and leaving a smoke trail behind it. I then used the same starter particle, and instead of adding a smoke trail I added a quicker and bigger fireball to it. This gives the impression that projectile hit something and then exploded.

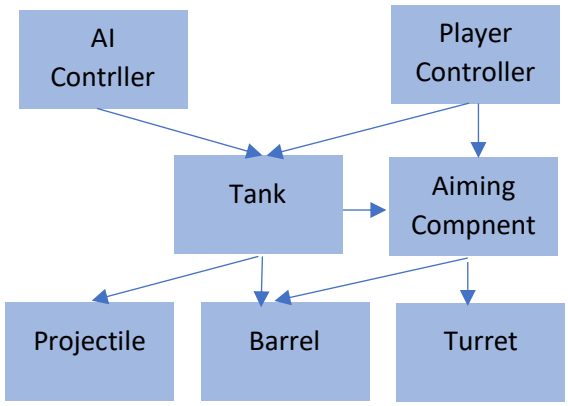
To script the particle to activate once it collides with something, I initialised all the particles in C++ code and attached them to the root component which means that will follow the projectile object through the world. I then override the 'OnHit' Unreal function to activate the impact particle once the project has hit something. I also use this 'OnHit' unction to delete the collider of the projectile so that they don't get stuck on each other once they've hit something.

3. Development Process

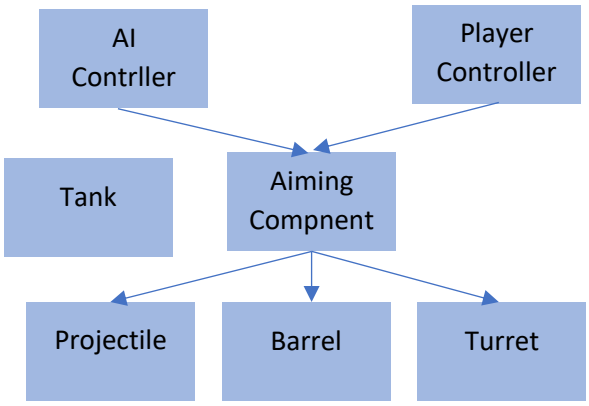
For most of the project development process, I started in C++ and created the classes first. I did this because I found it easier from a programming standpoint to follow my design and think through each required mechanic and program the logic as I went along. This did mean however, that at a point in the development timeline my code architecture became extremely messy and I had classes inheriting classes they didn't need and I had unnecessary dependencies between classes. This meant

that a major code refactor had to be done to make the C++ classes a lot easier to read, understand, and continue to develop. Below, I will show a diagram of how the refactoring changed my code structure for the better.

Old Architecture (Aiming)



New Architecture (Aiming)



4. Conclusion

Even though the project turned out better than expected, it unfortunately failed to include all the planned features. For example, when a tank died a fire particle effect was meant to spawn on top of the tank and stay there until the game ended. The reason this was not fully implemented was because every time I scripted the particle effect to attach itself to the tank object it always attached itself as the root component. This mean that in other classes, any reference to the root component was now a particle component instead of a static mesh component. This lead to a lot of crashes and errors and I unfortunately had to scrap that idea.

In terms of a critical evaluation, the project does have some downfalls. The biggest downfall would be that the blueprints rely too much on variables being set correctly. If a designer were to remove a tank track and then add it back in the track would lose some of its necessary settings. Specifically, it would lose its physic material and make the tank almost impossible to move in-game. This downfall slowed down development time significantly because I could not figure out why the tank was not moving for a long time.

Another feature that is missing and would add a lot of depth to all mechanics is audio. Having explosion sounds for the projectile and engine sounds for the tanks would add a lot to the overall mechanics and it make them a lot more fun to play. Furthermore, the visual appearance of the game is also lacking. I made some custom flat shaders to make the landscape more low-poly and I also imported some low-poly assets to add to the scene but without a low-poly tank model it looks out of place.

The project is also not bug free. There is still a major bug that can cause the tank to get caught on the flat part of the landscape causing the that tank to flip over. This bug is rare and the tank does not flip over most times but it still a bug I could not find a fix for. Sculpting noise into the landscape made this bug a lot less apparent. It appears that a perfectly flat landscape has issues with a perfectly flat collider.

On the other hand, there are some redeeming qualities with my project. The main one being that the C++ classes are very modular in how they operate. For example, the aiming script is very generic in what it needs to work. This means that if I were to model a mortar turret with a body, turret, and

barrel I can add an aiming component and the mortar will aim at my tank with not additional code needed.

5. References

Finch, A. (2013). *The unreal game engine : A comprehensive guide to creating playable levels*. Worcester]: 3DTotal Publishing.

Schildt, H., 2004. C++ : a beginner's guide 2nd ed., London: McGraw-Hill/Osborne.

Epic Games, Inc. (n.d.). *Unreal Engine 4 Documentation*. Available from https://docs.unrealengine.com/en-us/ [Accessed 10/04/2018].

Pluralsight. (n.d.). [Video Tutorial] *Introduction to C++ in Unreal Engine*. Available from https://app.pluralsight.com/library/courses/unreal-engine-introduction-cplusplus/ [Accessed 13/04/2018].

Pluralsight. (n.d.). [Video Tutorial] *Blueprint and C++ Integration in Unreal Engine 4*. Available from https://app.pluralsight.com/library/courses/unreal-engine-4-blueprint-cplusplus-integration [Accessed 19/04/2018].

Epic Games, Inc. (n.d.). *Unreal Engine 4 Documentation*. Available from https://wiki.unrealengine.com/Main_Page [Accessed 15/04/2018].

World of Tanks. [Computer Game]. 2010. PlayStation 4, Xbox One, MS Windows, Xbox 360. Wargaming.