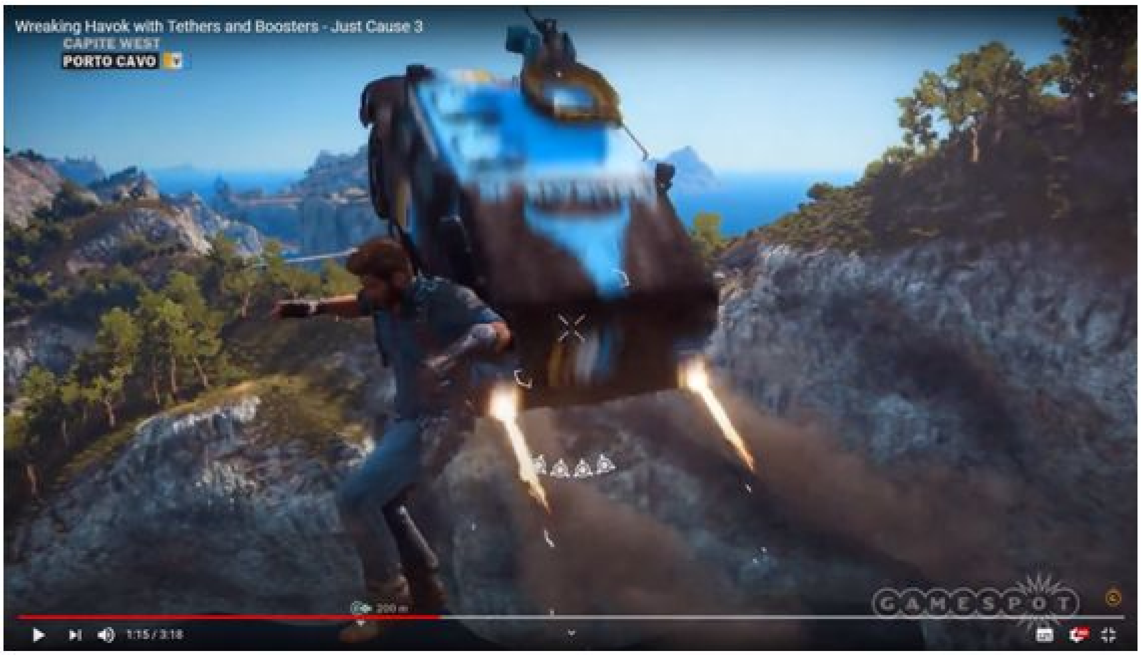
Edwyn Mckie

1901418

CMP302- Gameplay Mechanics Development

Brief Interpretation:

The student has been tasked with creating a unique game mechanic for use in a project using the Unreal Engine version 4.24.3. The mechanic the student decided to recreate in unreal is based upon the booster mechanic that is implemented in the just cause series of games developed by Square Enix. They have decided to do this in order to demonstrate the skills and techniques they have learned through this module.



video available here:
https://www.youtube.com/watch?v=1JiO25qN2NY&ab_channel=GameSpot

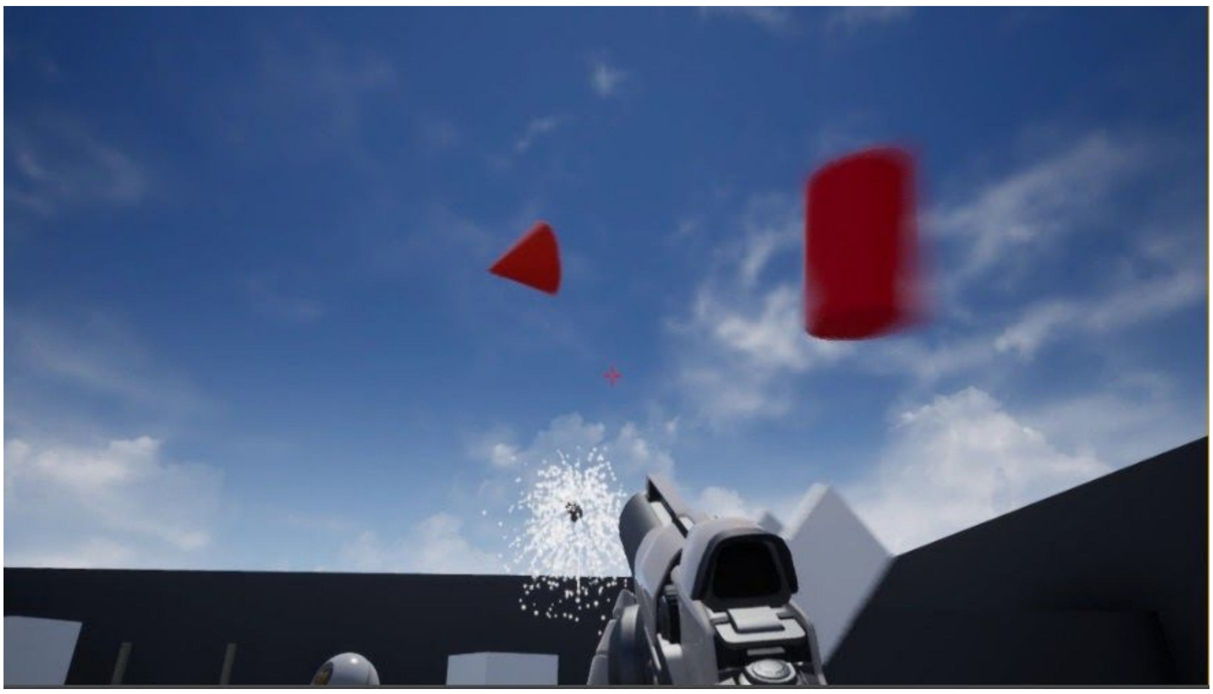
Summary of Mechanic/System:

The mechanic that the student is trying to recreate by using the unreal engine is similar in nature to the boosters pictured above. The player character will have a gun that when they shoot specific "dynamic" objects they will be able to attach rocket boosters to them, when the player presses another button the rockets will activate propelling the object forward depending on what side of the object they are placed on. The object will be affected by physics while being propelled in order to add some extra chaos and use the unreal engine's prebuilt physics engine. The thrust is applied to the object until the player decides to disable them.

Basic summary of how it works

The player will be able to use their gun to shoot projectiles that can stick to other objects in the game world. When the player decides to do so the projectiles can then be used to propel whatever they are stuck to forwards. This only works on objects that can be interacted with by the unreal engines prebuilt physics engine. This was implemented in such a way as the unreal engine does not allow users to change the mesh properties of an object at run time. The objects the projectile will be able to apply forces to must be "moveable" in the editor, However they can and do stick to all types of meshes.

Link to Demo Video



Demo available: https://youtu.be/pXezI9WHhGk

In this example the student is using the "SM_Rock" as the static mesh, the "Radial_Burst" niagra system particle effect and the "explosion01" as the sound effect. All three of these components are available in the starter content pack and are used solely to demonstrate how robust the system is and that you may swap them out depending on your game type. Background music for video available here: https://www.youtube.com/watch?v=9NPaLQY8aKk&ab_channel=VideoGameSoundtracks

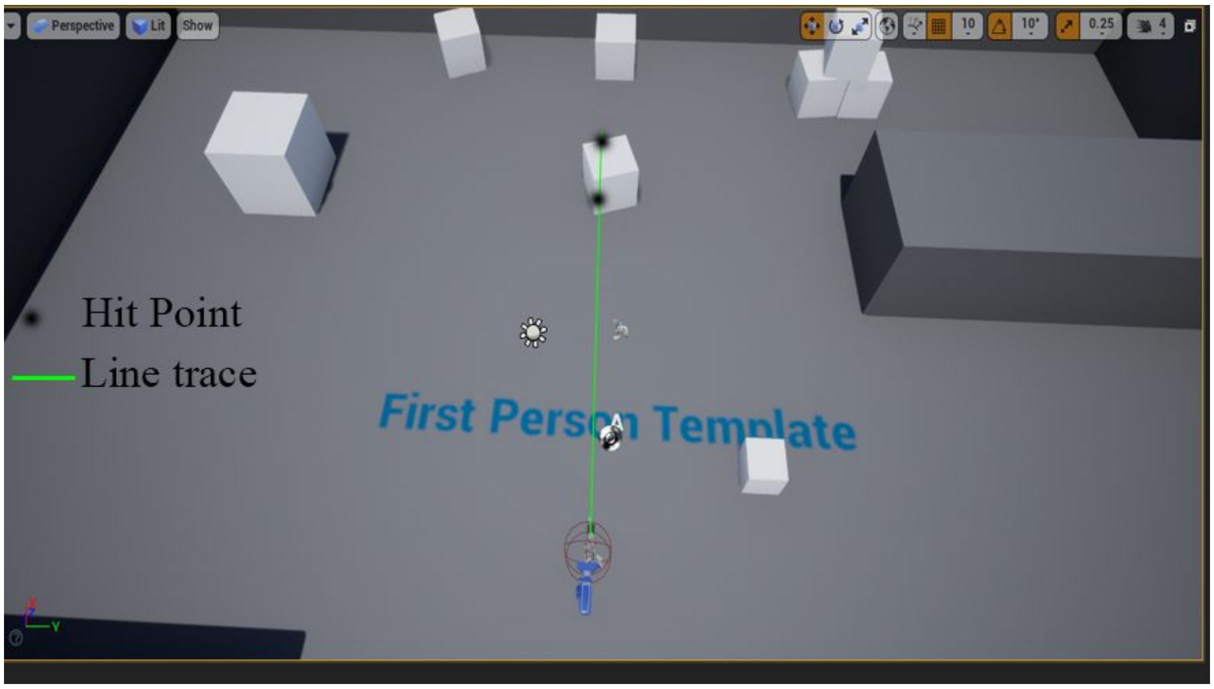
Requirements and Specifications

The mechanic the student has designed and developed is used in sandbox games for exploring the world and having fun. One of the most popular uses of this mechanic is in the game "just cause" as discussed above. In this implementation it is being used in a first person shooter style game to allow the user to have a close view of it in action. As well as to let them have more fun in the sandbox world with it.

The controls used in this example are rather simple but this is to ensure the code is efficient and the mechanic works smoothly even at low frames per second.

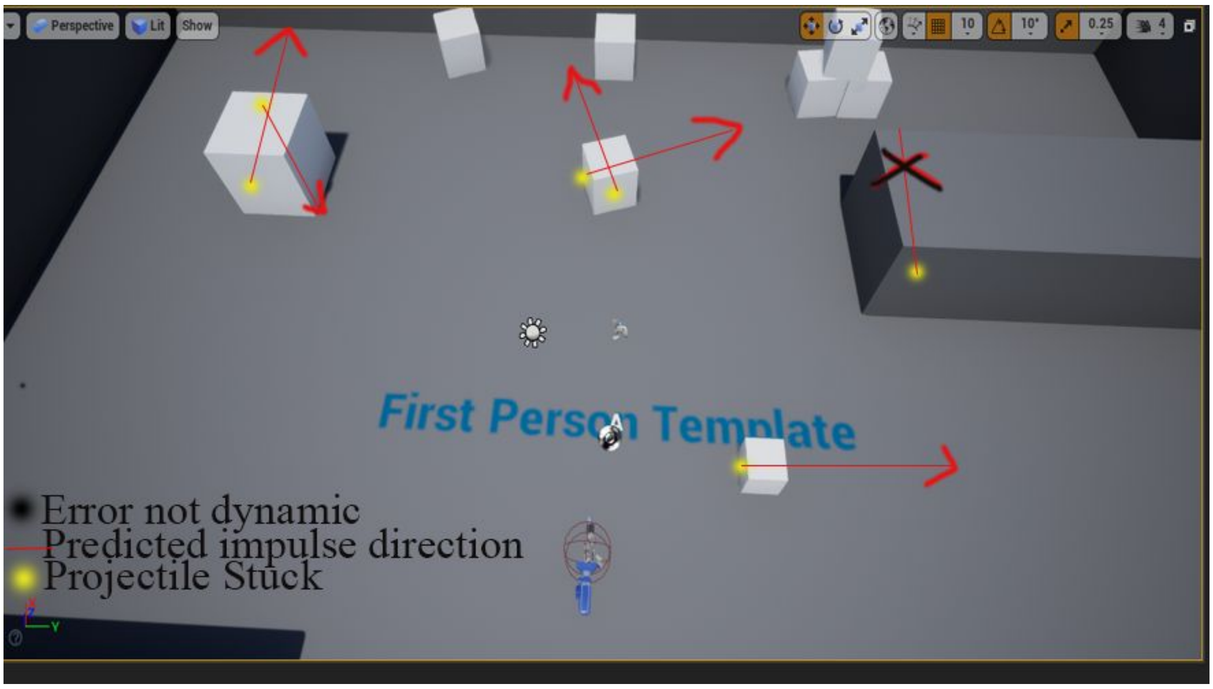
The mechanic obeys rather simple yet powerful rules. It can only apply physics to an object if the one it is attached to is dynamic and can be moved by the player or other forces.As it is impossible to change an object's interaction type from static to movable at runtime. Therefore it cannot move any static objects that are part of the level design, walls and such. The boosters can only be activated if they are close to the player so the player can see what is happening and decide when to disable them. The boosters life cycles are fifteen seconds unless they are activated after which it is then reset to half a second.

When the player left clicks it spawns a jet booster object, this part is handled by the default player gun. The mechanic changes the object that it spawned to one of the jet boosters. Once it creates a jet booster, the booster then tries to stick to any object it collides with be that a dynamic or static object. If the object it collided and stuck to is dynamic it will save the current velocity multiplied by a set value to use to propel the object forward later. Otherwise it will stick there and not do anything else.



In order to determine what it should stick to we use a line trace from the player's gun. This allows us to determine what to attach to based on what is closest to the player at that moment in time, as demonstrated in this diagram. In order to stick the projectile onto the correct side of the object we will use the first result of the raytrace as that is the side closest to the player.

When the boosters are attached to an object and it is dynamic, if the player right clicks the boosters will activate and launch the object they are stuck to onwards.This will use the value



that was saved when they stuck onto the object initially. This object will still collide with others and have an appropriate collision response as expected from the physics engine.

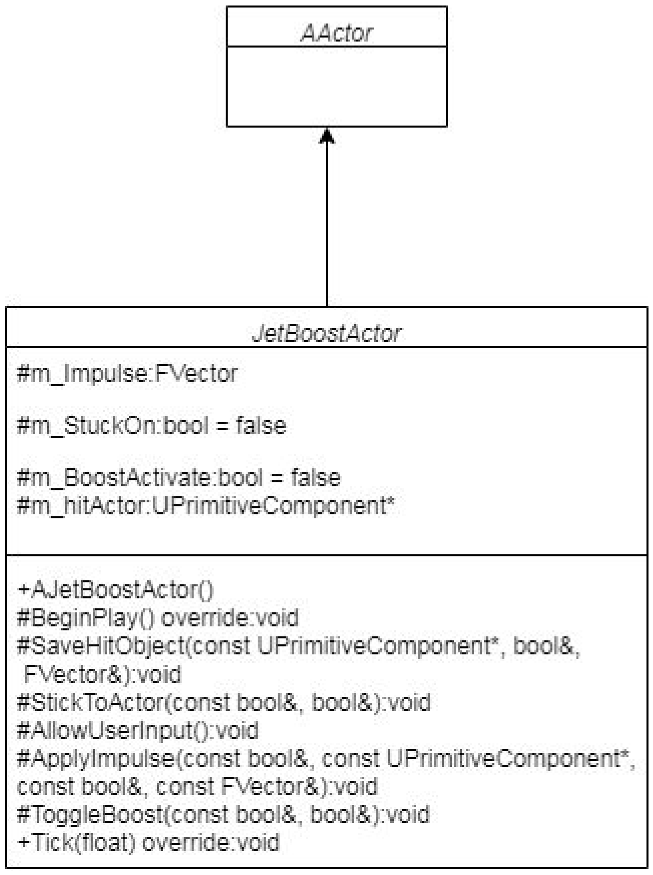
The direction in which the objects will be propelled is calculated when the projectile collides with an object in the world. As depicted in this diagram above in which the predicted directions are drawn. These values are calculated when the object sticks to another as it uses the current velocity to determine which direction it was moving before it was attached to the object.

The jet boosters are a very powerful mechanic and as such are limited so that the player can only activate them one at a time, when testing it was found that activating all of them at once was good but very unbalanced. When thinking about using them in a full game it would make creating level designs that utilize them very difficult. Another way to limit their power would be to limit how many of them can exist at a given time. This method has not been implemented as the current example is used to show their power and uses in a sandbox game.

The jet boosters cannot apply a force to a static object and therefore will not allow the user to try to. They also will not exist forever which is why we have limited their life cycles as mentioned above. These factors are controlled via code however the initial life cycle can be changed via the unreal editor. This is to allow the user to shoot as many as they desire before testing them. In a full game this would be increased or decreased depending on the genre of the game and the difficulty of it.

The mechanic can be used in order to solve puzzles and move environmental obstructions out of the way. It is also flexible enough to be used for some fun implementations such as making your own moveable platforms or sending enemies flying away.

UML diagrams (class and data level design)


AActor
JetBoostActor
#m_Impulse:FVector
#m_StuckOn:bool = false
#m_BoostActivate:bool = false
#m_hitActor:UPrimitiveComponent*
+AJetBoostActor()
#BeginPlay() override:void
#SaveHitObject(const UPrimitiveComponent*, bool&, FVector&):void
#StickToActor(const bool&, bool&):void
#AllowUserInput():void
#ApplyImpulse(const bool&, const UPrimitiveComponent*, const bool&, const FVector&):void
#ToggleBoost(const bool&, bool&):void
+Tick(float) override:void

Technical Discussion about mechanic

The jet booster projectile is instantiated by the player's gun, this is handled by the default first person shooter class. In order to have good encapsulation this functionality is handled by the default class and is not changed in the new class the student created.

In order for the jet boosters to know when to activate they need to enable input from the players controller. The unreal engine has a predefined function that Allows the student to do that, so the new class calls that function and gives it the current player's controller. By doing so this class can now listen for any key inputs from the players controller.

Once the projectile has been fired and has collided with another object in the world the new class then uses the collision response to find out what was hit. Unreal has a prebuilt function that does this and returns what was collided with. From this data the function can then check if the object is an actor and as such if it is static or dynamic. If the object has physics enabled the function then sets a bool to true. It will also save the impulse value to use later on. This impulse value is the current velocity of the projectile as it collided, before sticking to it, with the object multiplied by a substantial amount.

Once the jet booster projectile has collided with an object and the object is simulating physics it then needs to stick to it. This can be achieved in the following way. First the function will check that the projectile isn't already stuck to something. If it already is then the function does not need to run again. However if it isn't The function will then need to use a Line trace by channel. This line trace by channel is used to calculate if there is anything nearby that the projectile should stick to. It does this by creating a line from the camera's position, which we use as the start point and an end point that is set one hundred units away. The end point is calculated by using the forward vector of the camera and extending it by our length. The end point has to be quite a large distance away in case it is trying to stick to objects that are far away. If the function returns a value then it will set the boolean value that controls if it is already stuck to something to true. This boolean value is the one that is checked at the start of the function call. After setting that value to true the function will then attempt to attach the projectile to whatever it originally collided with at the position it collided. The line trace by channel function has a very useful return value which can be used for this. It returns the object that was collided with and the impact point of where it collided. The function will then use these values along with a premade function called Attach to actor. This function allows the user to stick the mesh of the projectile to what was collided with, in order to use this function you will also need to use some predefined Enumerators. Doing so will allow the object to attach in the correct place and retain its own size and shape. Lastly the function will then move the projectiles location the position that was hit by the object at so it looks like it sticks exactly where it collided.

As an earlier function has enabled user inputs this function can then use a simple check to toggle a boolean between states to keep track of if the boosters should or shouldn't be activated. If the boosters were already active and we are deactivating them we set the life cycle of the projectile to half a second in order to dispose of it safely. Otherwise we set the ActivateBoost bool to true in order to trigger the boost function to start.

Once the projectile has stuck to an object that is dynamic and has been attached onto it in the correct place. If the player then right clicks the unreal engine will then run the Apply Impulse function. In this function firstly it will check that the actor that was collided with initially is not a null pointer and not pending kill, as the engine will not apply forces to an object that is soon to be deleted. If it is not going to be deleted soon the function will then check if the player has toggled the boost activate boolean value from the previous function to true. It will also check that it is stuck onto an object. If both of these are true it will then use the apply impulse at location function in order to apply the impulse value that was saved earlier. It also needs to specify where to apply this impulse, this will be done at the current actors position. This is when the impulse force is actually applied and launches the object forward by the value of our impulse vector. However should either of the booleans be false it will then apply a thrust of zero so the projectile will begin to slow down and gradually return to its normal state.

Description of Development Process

To create the mechanic the student made a rough version in blueprints first in order to make sure it worked as intended. As in the past when attempting to go straight into c++ coding with the unreal engine the student has had mixed results. So, to ensure a working prototype the student made it in blueprints first. To do this they broke their mechanic down into three key elements; Stick to object, check the objects data and apply thrust. The student then found some tutorials that covered these topics, the line trace by channel reference guide was particularly helpful as it explained the logic behind how a line trace works as well as what the return data is and how it may be manipulated for our purposes.

Once they had a working prototype that the student was happy with they then went about breaking the blueprint down into functions that could then be called within the blueprint, these are demonstrated in the above sections.

After breaking the blueprint down into these functions, The student could then begin the task of converting the functions into C++. While making sure that each section worked on it's own before moving onto the next one. As such this is why there are some getters and setters in the blueprint screen. It is designed this way so that functions all have access to important data when they need it instead of having to wait to receive a copy from another function. As the code is private only the code or blueprint may change these values, this is to allow designers to tweak the values without having to look through the code.

As C++ is significantly faster the student then converted all the advanced logic and complex function calls into it and only kept the variables that designers may want to play with as directly editable through the blueprints.

Conclusion

In the development of my mechanic I ran into a few issues when trying to convert it to code initially as the class would not appear in the editor regardless of what I did. It was only after discussing with the lecturer did I finally realise my issue was that I was trying to inherit from the wrong type of class which is what was causing my issue.

Getting a working copy in blueprints was a great idea and worked really well. The Unreal engine has a great amount of tutorials on and help online so using blueprints was very fast and convenient. As such creating my mechanic through blueprints was nice and straightforward.

Another issue I ran into when developing my mechanic was that when I finally did manage to get the projectiles to stick to objects they would scale in size to match what they had stuck onto. This issue was quite a confusing one as I was unsure what was causing it until I did some research online and found out about how to set it to keep the current form even when stuck to another object.

Converting to code went well as I could find useful guides online however finding ways to make sure it all worked together was a challenge. For example when using line trace by channel in blue prints you need to use another function to get specific parts of the data from the hit result however in C++ this isn't necessary, I did not know that and wasted precious time trying to work out what my issue was.

Another issue I had was making the projectile move with the object when it is propelled as at the moment it still does not move with the object when applying a force to it which looks rather odd. However I cannot work out what is causing this issue, my inexperience with unreal is the main cause of this issue.

Through doing the module we covered a basic implementation of line trace by channel so when I went to use it for my mechanic I already had a rough idea of the power it holds. I think this helped me to create a nice mechanic as I already knew how it could work and had already started making plans on how to utilise it for my mechanic. By having a simple plan of how it should work before creating it I believe it allowed me to create a fun and nice mechanic without many issues aside from those mentioned above.

Overall I am very happy with my mechanic and I believe making the blueprint first then converting to code is more efficient as I am learning the API as I go and do not know the best way to go about doing it in code straight away. As such having a working blueprint was a great plan to start with. The problem I ran into a lot when creating the mechanic was that if you looked up a blueprint function you only found guides on how to use it in blueprints not convert it to code. The other issue was if you did manage to find some information on it they only explained what the components of the function were. However in the future I would just make the blueprint with the functions initially instead of making it all at once and then breaking it down.

Reference list

* Docs.unrealengine.com. 2020. *Using A Single Line Trace (Raycast) By Channel*. [online] Available at: <https://docs.unrealengine.com/en-US/Engine/Physics/Tracing/HowTo/SingleLineTraceByChannel/index.html> [Accessed 20 November 2020].

*Unreal Engine Forums. 2020. *First Person Projectile Lifespan - Unreal Engine Forums*. [online] Available at: <https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/47163-first-person-projectile-lifespan> [Accessed 22 November 2020].

* Youtube.com. 2020. [online] Available at: <https://www.youtube.com/watch?v=5DK8jF1ygeM&ab_channel=SadowickDevelopment> [Accessed 25 November 2020].

* Unreal Engine Forums. 2020. *Check If Variable Is Set - Unreal Engine Forums*. [online] Available at: <https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/1820-check-if-variable-is-set> [Accessed 1 December 2020].

* Answers.unrealengine.com. 2020. *It Is Pending Kill - UE4 Answerhub*. [online] Available at: <https://answers.unrealengine.com/questions/374846/it-is-pending-kill.html> [Accessed 1 December 2020].

* Answers.unrealengine.com. 2020. *Why Is My Input Key Press Not Working In Blueprint? - UE4 Answerhub*. [online] Available at: <https://answers.unrealengine.com/questions/19063/view.html> [Accessed 5 December 2020].

* Docs.unrealengine.com. 2020. *Uworld::Linetracesinglebychannel*. [online] Available at: <https://docs.unrealengine.com/en-US/API/Runtime/Engine/Engine/UWorld/LineTraceSingleByChannel/index.html> [Accessed 5 December 2020].

* Docs.unrealengine.com. 2020. *Uscenecomponent::Issimulatingphysics*. [online] Available at: <https://docs.unrealengine.com/en-US/API/Runtime/Engine/Components/USceneComponent/IsSimulatingPhysics/index.html> [Accessed 10 December 2020].

* Unreal Engine Forums. 2020. *Enable Inputs (Playercontroller) In C++ - Unreal Engine Forums*. [online] Available at: <https://forums.unrealengine.com/development-discussion/c-gameplay-programming/28847-enable-inputs-playercontroller-in-c> [Accessed 10 December 2020].

* Answers.unrealengine.com. 2020. *[C++] Add Particle System Component - UE4 Answerhub*. [online] Available at: <https://answers.unrealengine.com/questions/719468/c-add-particle-system-component.html> [Accessed 20 December 2020].

*Answers.unrealengine.com. 2020. *How To Getactorlocation() In C++ For "Target" Object - UE4 Answerhub*. [online] Available at: <https://answers.unrealengine.com/questions/669930/how-to-getactorlocation-in-c-for-target-object.html> [Accessed 20 December 2020].