CMP302 GAMES MECHANICS REPORT

Cameron Thustain 1900594

Terminology

Mana – a resource belonging to the player which is required to use the dragon shout.
It recharges over time but if the player runs out of mana then the spell cannot be used.

Spell Cost – Spell cost is the required amount of mana required to use the dragon shout, within many role-playing games different spells have different costs depending on how powerful they are. The player may have mana available to use but it could be under the certain amount required to use the spell.

Dragon Shout/Fus Ro Dah/Unrelenting Force - The name of the spell.

Summary

The mechanic recreated is a spell that allows the player to push/force enemies and objects in a direction inflicting damage upon enemies. It is mainly based around the Dragon Shout, specifically "Fus Ro Dah" from Skyrim (Bethesda Game Studios). (Unrelenting Force (Skyrim), n.d.)

Components of the mechanic

- Area of effect
- Mana
- Mana recharge and recharge rate
- Spell cost
- Ui showing the players mana.

Demo Video Link:

https://youtu.be/TvyyJOQmbVQ

Introduction

The purpose of this project was to rebuild Skyrim's Dragon Shout mechanic, to see how this could be implemented in a newer engine as the game is now over a decade old. The aim was also to see this mechanic structured into code and possibly improved on.
The project is a basic spell rather than an entire system as Skyrim's spell system differs slightly from other role-playing games. The components included within the project include a mana system, a small casting system, and different areas within a level to test the spell.
The product is very easily expandable if need be or chosen to.

Description

The main aim of this project was to recreate Skyrim's Dragon Shout as best and as accurately as possible with slight changes that betters it. The aim was to have enemies within the scene to test this spell on and a mana system so the player can't abuse the spell.
Functions :

1. Show the user how mana they have, and it recharges over time
2. Allow player to test spell on moving enemies
3. Allow any further work to be implemented into the system.
4. Allow any user to modify values of spell costs and mana recharge rates.

Control Scheme

Within the application the user must use these controls to use the mechanic :

- Left Mouse Click (once the player is not moving and mana is >= 0.25)
- Arrow Keys to move the player around
- Mouse movement to aim at enemies

Characteristics

The mechanic was made using Unreal Engine 4 and C++. If this project was to be expanded upon its mainly aimed at designers or programmers who are familiar with Unreal Engine 4's tools, blueprints, and UI.

The current Dragon Shout has been implemented and can be extended on but not required. However, a programmer or designer may create a new Dragon Shout spell or expand on the current one to meet their games requirements or further expand on the gameplay and complexity.

Constraints

Throughout this project there was no artist, this means most of the scenes and actors within the scene are from free marketplace packs. (City of Brass: Enemies in Characters - UE Marketplace, 2021).

Features

Spell - Dragon Shout/Fus Ro Dah/ Unrelenting Force
Description of spell - A spell that is cast by the player pushes objects or entities in a direction. Weaker enemies like skeletons are thrown at a greater distance and their bodies are turned into a ragdoll.

Gameplay – the functionality of the Dragon Shout can be used through mouse input
Development – The blueprint of the shout opens the Unreal Engine 4's blueprint editor while the code for the mechanic, written in C++, opens visual studio 2019.

Requirements for a mechanic:

1. Player moving check

To use the spell requires the player to be standing still, if the player is moving in any given direction or even jumping the spell cannot be used.

2. The required amount of mana

The player must have the required amount of "mana" the spell has a cost of 0.25 mana. The player starts with 1.0 after the spell is used 0.25 is taken away from 1.0 if the player has under 0.25 mana then the spell cannot be used. (Regenerating Health & Mana - #8 Creating A Role Playing Game With Unreal Engine 4, 2017)

3. Valid target

The player must be aiming at a valid target, if the enemy isn't in the line trace the spell won't be cast. The target must also have the tag "Pushable" otherwise the spell can't be used.

4. Valid Input

Finally, if all the requirements are met the last requirement is if the user is pressing a valid input. If the user is pressing anything other than the left mouse click the spell won't be used.

Targets Available

Description - Within the level, multiple pawns chase the player when the player is sensed nearby and if the player is in the mesh boundaries. These pawns will react to the spell as they all have a collision response to the spell.

Gameplay – User/Player has no control over these enemies other than somewhat leading them when moving, the enemies chase the player if the player is in an area of view of the enemy. Anything else however the player has no control over. The enemies within the level will react accordingly to the spell when the player uses it.

Development – The blueprint for the enemies opens the Unreal Engine 4's blueprint editor while the code for them, written in C++, opens visual studio 2019.

Requirements for enemies:

1. Damage

The enemies within the level can receive damage from the spell this is shown in the form of their bodies shattering into parts when hit by the spell since they are shown off using a skeleton model from the City of Brass enemy pack.

2. Following the player

The enemies follow the player around the level as it would be a coming occurrence for players to use this spell within Skyrim against oncoming enemies. The enemies simply follow the player if in a certain distance and view.

3. Applicable to be hit

The enemies within the level are given tags if they don't have that tag then the spell has no effect on them and cannot be used.

<u>Testing Area/Level</u>

Description – an area where the user can test the mechanic and explore freely.

Gameplay – the user can explore this area; however, walls are stopping the player from going any further or leaving the testing area.

Development – the map itself opens within Unreal Engines editor.

Requirements for testing area :

1. Enemies to test on

Within the testing area, it was required to have enemies the players could test the spell on and check if they could be tested on or not.

2. Boundaries

It was also required for the testing area to have boundaries, walls, in this case, to stop the player from leaving since there is nothing past the set area.

<u>User Interface</u>

Description – The UI within the level must display the player's current mana and the mana when it' re-charging. (Health & Mana Setup - #3 Creating A Role Playing Game With Unreal Engine 4, 2017)

Gameplay - The progress bar at the bottom of the viewport should always display and show when mana is being lost or gained again.

Development – for the development of the mana bar, the Unreal Engines widget editor opens.

Requirements for UI:

1. Show the correct amount of mana

The UI is required to always display the correct amount of mana. The bar should change when the player uses the spell, and their mana is depleted. The bar should also show the mana bar filling up again when mana is re-charging.

Player
Description – The player is required to move around the level by using keyboard and mouse input. The player is also required to cast the shout mechanic.
Gameplay – Within the game, the player responds to user input (arrow keys and spacebar) to move around the testing area.
Development – the blueprint for the player's movement will open in the blueprint editor.

Requirements for players:

1. Responding to input

It was a must-have requirement for the player to respond to the input whether that is moving around the level or using the mechanic.

Performance Requirements

If the game mechanic or level was to be expanded upon frame rate is something that needs to be considered. As more enemies are added to the testing area or more complexity is added to the current spell or any new spells this could overall affect the frame rate. The frame rate must be always smooth for the mechanic and level to run properly.

Testing for this project was conducted regularly after the implementation of any new feature whether that was a new UI, a new enemy, or something to do with the mechanic itself.

Method

Mana System
One of the requirements for this project was to have a mana system that would stop the player from abusing the spell. Before the player uses the spell the mana is required to be checked first. The mana's current value is checked against the spell cost. The spell cost is specific to the mechanic but if more dragon shouts were to be implemented they would also get their own cost. (Regenerating Health & Mana - #8 Creating A Role Playing Game With Unreal Engine 4, 2017).

The spell cost and mana are checked using If statements before the dragon shout is executed. However, if spell cost was to change or the total amount of mana were changed this could cause issues as the program and blueprint would need to be recompiled every time a change is made which is inefficient for the programmer. As more dragon shouts got added this would further increase inefficiency. The mana system itself is done within the first person character class however, if more spells were added or the original was to be extended upon it would be given its own.

To fix this for better efficiency the likes of struct could be implemented or setter functions where the data of the dragon shout is stored for example current mana, spell cost, mana recharge rate, etc, if this project was to be developed or extended in the future.

Dragon Shout

Another requirement for the project was the mechanic itself. The mechanic is a dragon shout spell. This is cast by the player. The player aims at an enemy and ragdolls the enemy in any direction. Every time the player clicks the left mouse button the line trace is executed. There are checks done to see if anything hit by the line trace has the tag "Pushable" if not nothing happens, if so the enemy's collision is called, and the spell cost is taken from the mana. The mechanic itself is done within a function in the first person character class. This was decided because in Skyrim spells and mana "belong" to the player. However, if more spells were implemented in the future or the original was given a big upgrade their own classes is something that would be highly considered for increased efficiency and robustness.

Enemy Collision

Within Skyrim if an object or enemy was hit by the dragon shout their body would ragdoll very far. This was a requirement within the application. The enemy's collision profile is set to ragdoll and when it simulates physics its set as true. The enemy is also given a tag called "Pushable" which is a unique identifier for the game objects defining what the spell can be used on and what cant. If an enemy is hit by the line trace and found to have that tag an impulse is added to the enemy. Impulse causes a change in the momentum of the enemies' bodies in the given direction that impulse is applied (Isaac Physics, n.d.). The enemy collision is done within one class but as more enemies and different types were added this can become quite complex. One method of making this better would be having an overall enemy or object class with the types of enemies/objects having their own subclasses inheriting from that parent class.

Mana bar UI

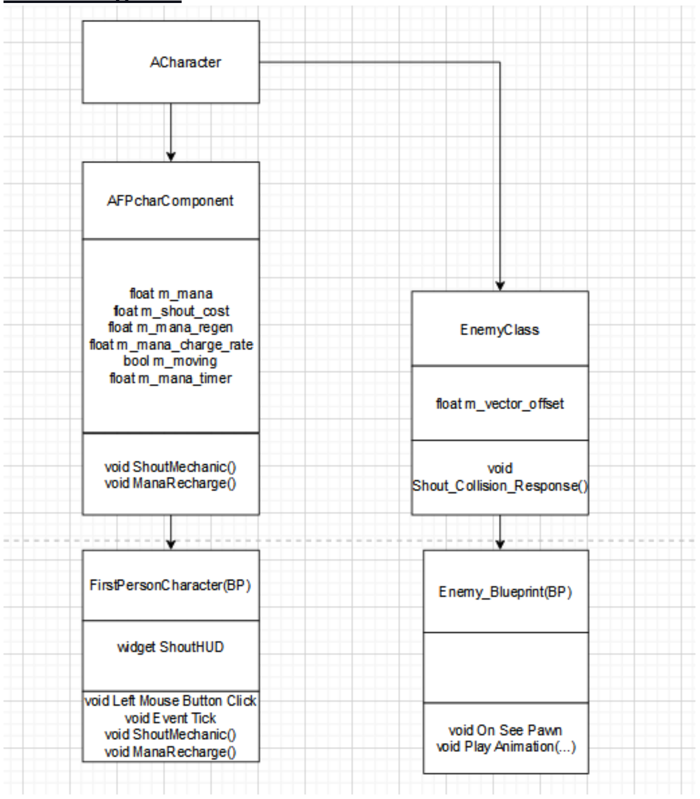
The mana bar UI is created by the first person character blueprint at the beginning of the application's runtime and added to the viewport. This bar is used to display the player's current mana.

The UI has a progress bar for the mana and text, so the user knows what the bar is for.

The UI is created using a widget called "ShoutHUD". The widget always accesses the players mana so it can constantly be updated on screen every tick. The widget itself is dynamically created by the first person character blueprint at the start of the applications runtime and added to the viewport of the camera.

Development

UML Diagram



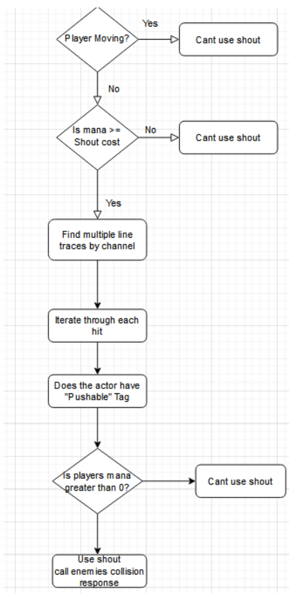
" (...)" is used for increased readability this is for functions that have many parameters.

Technical Discussion

Shout Spell Mechanic:

The shout mechanic itself was programmed firstly by performing a collision trace along multiple lines and returning all hits encountered up to the very first blocking hit (Using a Multi Line Trace (Raycast) by Channel, n.d.). The function "LineTraceMultiByChannel" takes an array of hit results called "OutHit", the next parameter within the function is a starting vector. This was calculated by creating an FVector called start_ which simply gets the actor's location using the default function "GetActorLocation". The next parameter that had to be input is the end vector this is calculated by adding a forward vector to the start vector. The forward vector itself was given its variable called "vector_" this was calculated by using the kismet math library's function "GetForwardVector" this function takes a rotator so the default function "GetControlRotation" is used in this case. Once that is done the forward vector is multiplied by a float which is hard valued as 4000. The final parameter required is an Enum of Collision Channel which is set to "ECC_Visibility". The next step is to iterate through the array of hit results this is simply done through a for loop using “auto it”. While iterating through the hit results a few checks are done. The first check is if the actor has not got the tag “Pushable” (which is set in the enemy class constructor) (How to set Actor's Tag in C++ on Runtime ? - UE4 AnswerHub, 2019) if not it continues. From there it creates an auto pointer called “enemyptr” which is set to a cast of the “AEnemyClass” and uses the iterators “GetActor” function (Casting - C++ syntax and UE syntax, 2015). Once done another check is executed simply for error checking if the cast isn't null it will continue. The final check for the mechanics function is checking if players mana. The conditions are as follows if the player's mana is greater than zero the enemy collision response function is called (Shout_Collision_Response). Once that function has been called the variable “m_shout_cost” is taken away from the mana (1.0 – 0.25). The mechanics function is called within the first person character blueprint after the event “Left Mouse Button (pressed)”. The class created is set to the parent class of the blueprint so these variables and functions can be accessed.

Mechanic Flowchart


Player Moving?
Yes
Cant use shout
No
Is mana >= Shout cost
No
Cant use shout
Yes
Find multiple line traces by channel
Iterate through each hit
Does the actor have "Pushable" Tag
Is players mana greater than 0?
Cant use shout
Use shout call enemies collision response

Enemy Collision Response

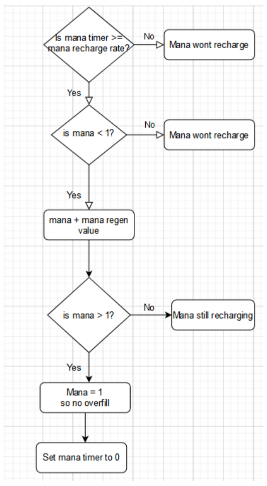
The enemy’s collision response is calculated by firstly setting the mesh collision profile to “Ragdoll”. Once done the meshes physics simulation is set to true. The next step that was taken was creating a cast to the player class “AFPcharComponent” where the gameplay static class function “GetPlayerCharacter” is used this function takes a UObject and a player index which is set to “GetWorld” and zero (Casting - C++ syntax and UE syntax, 2015)

. Next, an error check is done if the “playerptr” is a null pointer return is called. The next step is getting the control rotation from “playerptr” as another FVector needs to be calculated called “impulse_” which is a force that starts a body into motion. The impulse is calculated by using the kismet math library’s function “GetForwardVector” the player's rotation is used as the rotator this time around and the forward vector is multiplied by

15000. Next, the “AddImpulse” function is used using the impulse vector that was just calculated and set the velocity change to true. Finally, the collision of the capsule component is enabled and set to no collision (How do I access the CapsuleComponent via C++ - UE4 AnswerHub, 2020). The enemy’s collision was originally called within the enemy blueprint however, this is longer needed. The function itself is now called at the end of the shout mechanic function in the first person character class so if that happens, the enemy’s collision will happen as well.

<u>Mana System</u>
This is done within the first-person character class component. A function called “ManaRecharge” is created and called every tick. The first step is checking if the mana timer variable is greater than or equal to the mana charge rate (0 >= 0.5). The next check is if the player has run out of mana so if mana is less than 1. If so the mana regen variable is added to mana (0.02 + mana). This is the rate at which the mana regens each tick. Finally, a check is done to see if mana is back to full and so it can’t go over the total amount. If mana is greater than 1, mana is set to 1. Once done the mana timer is set back to 0. The mana system was originally done within blueprints but is now done in the first person character class which is the parent class of that blueprint. The function itself is called within blueprints but after the event “Tick” as the mana must be checked each frame to see if it needs to be recharged or not.

<u>Mana System Flowchart</u>



<u>Development Process:</u>
Through the development of this application, it was always decided to create the mechanic using blueprints and then translate most of those blueprints into C++. The C++ functions themselves are still called through blueprints all apart from the enemies’ collision response which is now called in the Shout Mechanic function. It was decided that the main parts of the mechanic would be done in C++ and additional features such as the mana bar or enemy movement would be done in full blueprints simply due to them not solely impacting that mechanic.

Conclusion:

Overall, the application has been a great learning experience especially being the first time use of a major game engine. Some things caused difficulties and others were easier than expected. Firstly, the ability to create a mana system and have the display on the HUD was a lot simpler than first anticipated and overall, it made the mechanic a whole lot better with that in place. Some difficulties arose when it came to implementing the mechanic. The spell in Skyrim originally can hit multiple targets at one time in any area, within this application it's being programmed so one enemy can be aimed at and hit or multiple in one line due to the use of the "LineTraceMultiByChannel" function. However, this could be an improvement. Throughout many playthroughs, I and other players included always at one point or another didn't hit the enemy or target that was initially targeted or attempted to target. The project is one way or other fixes that and the specific enemy can be targeted rather than missing the one that was trying to be hit and hitting another instead. In conclusion, the project was a great learning experience, many of the things learned can be brought into future projects or different engines and coding languages for example translating blueprints into C++. The project was a breath of fresh air and a lot of fun with new challenging tasks to overcome this has overall shown my new skills within Unreal Engine and the content of the module and met with the set learning outcomes. Working on this project more I would have liked to have more polish, more effects, calling new functions in C++, and would overall add more dragon shouts/spells to play around with.

References:

Unreal Engine Forums. 2015. *Casting - C++ syntax and UE syntax*. [online] Available at: <https://forums.unrealengine.com/t/casting-c-syntax-and-ue-syntax/29652/10> [Accessed 17 January 2022].

Unreal Engine. 2021. *City of Brass: Enemies in Characters - UE Marketplace*. [online] Available at: <https://www.unrealengine.com/marketplace/en-US/product/b7cbc53813a24db1a5bd42f75151698c> [Accessed 17 January 2022].

Youtube. 2020. *GB Force Push Tutorial Unreal Engine 4*. [online] Available at: <https://youtu.be/KhaebDKapCs> [Accessed 17 January 2022].

Youtube. 2017. *Health & Mana Setup - #3 Creating A Role Playing Game With Unreal Engine 4*. [online] Available at: <https://youtu.be/pu56eOqtlQ0> [Accessed 17 January 2022].

Answers.unrealengine.com. 2020. *How do I access the CapsuleComponent via C++ - UE4 AnswerHub*. [online] Available at: <https://answers.unrealengine.com/questions/958417/how-do-i-access-the-capsulecomponent-via-c.html> [Accessed 17 January 2022].

Answers.unrealengine.com. 2019. *How to set Actor's Tag in C++ on Runtime ? - UE4 AnswerHub*. [online] Available at: <https://answers.unrealengine.com/questions/937126/how-to-set-actors-tag-in-c-on-runtime.html> [Accessed 17 January 2022].

Isaacphysics.org. n.d. *Isaac Physics*. [online] Available at: <https://isaacphysics.org/concepts/cp_impulse?stage=all> [Accessed 18 January 2022].

Youtube. 2017. *Regenerating Health & Mana - #8 Creating A Role Playing Game With Unreal Engine 4*. [online] Available at: <https://youtu.be/-1IYgvk_TbQ> [Accessed 17 January 2022].

Docs.unrealengine.com. n.d. *UGameplayStatics*. [online] Available at: <https://docs.unrealengine.com/4.27/en-US/API/Runtime/Engine/Kismet/UGameplayStatics/> [Accessed 17 January 2022].

Elder Scrolls. n.d. *Unrelenting Force (Skyrim)*. [online] Available at: <https://elderscrolls.fandom.com/wiki/Unrelenting_Force_(Skyrim)> [Accessed 17 January 2022].

Docs.unrealengine.com. n.d. *Using a Multi Line Trace (Raycast) by Channel*. [online] Available at: <https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Tracing/HowTo/MultiLineTraceByChannel/> [Accessed 17 January 2022].