FIT2097 – Assignment 3 Program Design Document

Prepared by Joshua Leung

StudentID: 28261526

AuthCate: jleu0001@student.monash.edu

Note: This document is solely for the scope of changes done in A3. For changes in A1 and A2, the document has been moved to their respective folders in the repository.

UE4 Gameplay Framework Classes used:

Slightly Modified (?)

- FIT2097Week3.Build.cs & FIT2097Week3.uproject
    - Not really sure if this applies but changes were made to include Niagara systems and NiagaraExtras
- Animation Blueprint (Confirmed with Jason that this was allowable)
- SoundCue

Used and modified extensively to achieve design goals and targets:

- Character (SpecialEnemyCharacter)
- AIController (SpecialEnemyAI(BP))
- (Attempted, did not work) BTTaskNode (FindPlayerLocation)
- Actor (All new interactable objects such as grenadepickup and final door)

Blueprint Grey Area:

- BehaviourTree
- Materials (M_ScreenDamage, WoodMAT) & Physical Material (ConcreteSURF, WoodSURF)
    - BUT all dynamic instances of materials are manipulated in code.
- Animation Blueprint (Confirmed with Jason that this was allowable)
    - Used animation notifiers to call on C++ defined event (BlueprintNativeEvent and BlueprintCallable)
- SoundCue
- SequencePlayer

Classes changelog and design processes explanation:

(Note, there are some CPP classes created but not discussed here, those classes were created but never used, remained in the project so as to not break anything. Thus, they were not included here)

MODIFIED

FIT2097Week3Character

- Base Parent Class: ACharacter

Logic, information and code encapsulation:

(Custom and modified variables and functions included only)

To implement the grenade functionality, A GrenadeCount variable was declared to store the amount of grenades the player can fire. GrenadePickups were made with the approximately same logic as keys, fuse and healthpickups for the player to cast to their class on collision, bind their event broadcasters to event listener function GiveGrenade, execute the implementation for the grenade pickup to broadcast the GiveGrenade event and call the GiveGrenade function. This function increases the player's grenade count by one. The OnFire code originally used for firing projectiles in the FPSTemplate was repurposed to receive input from a new binding "Grenade". On Q key press, a check is made to see if GrenadeCount is greater than 0. If true, then a projectile will spawn and be launched and grenade count is decremented. This is done by directly linking the projectile, firing sounds and animation in the editor.

To allow the functionality of a dynamic damage screen, APostProcessVolume* was created and set to EditAnywhere and BlueprintReadWrite. This is to allow us to create a damagescreen post processing volume in the world (set it high off map somewhere so we will never actually be in the volume) and be able to set it bound and unbound dynamically to simulate a dynamic damage screen. In the constructor, we initialize this pointer to be bound so that our screen does not have that effect. Once we TakeDamage() from enemies, DamagePostProcess is set to unbound so we can see this effect. We piggyback off of the code that we have made in our Immune reset function and set it back to be bound again so the effect goes away.

Extra Functionality:

To allow implementation of dynamically changing and properly synced to animation footsteps, a complicated process was made. We start by defining and opening to blueprintreadwrite and edit anywhere two separate soundcues. We then define a UFUNCTION() called FootstepEvent to be BlueprintNativeEvent and BlueprintCallable. The implementation is basically a linetrace downwards that detects the surface type from the physical material of the floor. Based on the surfacetype, the appropriate sound cue is played at actor location.

How this function is then called is that in the animation blueprint for the player character – the FPS animation blueprint – Animation notifies were created in the frames where footsteps are seen to be made. Through this animation notify, we then use the animation blueprint to cast our player and call the FootstepEvent we have defined in our C++ Class.

To Facilitate the interaction of instant death upon colliding with the new SpecialEnemyCharacter, the UFUNCTION() KillPlayer was created. This function would be binded to a listener from a broadcast event from SpecialEnemyCharacter to call the PlayerDeath event used to bring up the game over screen. The player makes an oncollision event happen with a SpecialEnemyCharacter object and triggers the execute interact and execute display information. This oncollision event alerts the SpecialEnemyCharacter class that it must broadcast the kill player event in their execute interact which in turn gets listened by the player and then triggers the KillPlayer function, subsequently causing a game over.

AutoDoor, BinaryDoor, FuseDoor, FuseSwitch, FuseBox

- Base Parent Class: AActor

Changes were made as to how the event listeners were binded. Previous iterations had them in tick. This caused many issues with engine crashes. This has been changed to be done in begin play so it is only done once. Opening/Interactions were also changed in that in most scenarios, a bool isActived is checked to see if the door needs to be opened.

Changes to FuseBox were similar with the addition of taking a charge of fuse away from the player. Allowing for further use of the fuse switch as seen in the final door.

BinarySwitch

- Base Parent Class: AActor

Not much changes were made here other than that new events were made to increase the total of BinaryCode events to 4. This is to help scale up the number of switches that could be used at once allowing for the implementation seen with the final door.

EnemyCharacter

- Base Parent Class: ACharacter

The EnemyCharacter class is now designed to have checks to see if they are stunned. The way this works is that when a FIT2097Projectile hits the EnemyCharacter, the enemy's EnemyStun function is called. This function sets the bool isStunned to true and sets charactermovement to deactivate. A delayed timer is then created to call the reset stun function after 5 seconds which would then reactivate the charactermovement component.

Extra Functionality:

The footstep functionality seen in the player character was able to be reworked easily to work with enemies as well. The implementation remains the same, where the difference lies is in the animation blueprints. As this project is based off of an FPS template, the footsteps for the player were done in the FPS animations and animBP. For enemies, the ThirdPerson animation blueprint would be used.

The implementation choice of using play sound at location has turned out to be a good choice as this allowed for interesting applications in hearing where enemies are (positional audio).

FIT2097Week3Projectile

- Base Parent Class: AActor

The base FPS template projectile was reworked to include Niagara function calls as well as adding a new collision sphere to simulate the aoe radius of a grenade. Unfortunately, I could not get the on overlap begin function to work and as a result, the logic is very much constrained to on hit. On hit, checks to see if the other actor was an enemy or a special enemy was collided with. If so, we can then call on the enemy stun function and stun them, proceed to play the Niagara particle system we have directly linked through the editor and destroy actor. When it doesn't detect any other specific actor, the Niagara particle is then also deployed at hit location and the projectile is destroyed.

The reasoning behind reusing the base projectile class was that the base functionality of the class is already working well enough and I did not see reason to create another class, especially when the base character I am working from has already integrated well with it.

EnemyAIController

- Base Parent Class: AIController

Changes made to this class have been deprecated. Attempts were made to find a way to get the player character's location and return this as a blackboard key. This function was intended as a special function to be called by a new behaviour tree for the special character. Unfortunately, this caused the game to crash on play. Thus, the changes have been deprecated.

New Classes

GrenadePickup

- Base Parent Class: AActor

The GrenadePickup is set up in a similar way to KeyPickup in that it has dynamic material instance that is set to pulse in purple, an event broadcaster and its interaction is to display information and broadcast the give grenade event to the player.

FindPlayerLocation

- Base Parent Class: BTTaskNode

Class is designed to be a custom behavior tree task and checks if we have a behaviour tree component and a controller. If we do, we then call on the find player location function to then feed our behaviour tree of where the player is at. This is a special function designed to be used in the special enemy's behaviour tree to simulate RE2's Mr. X. This function is now deprecated.

FinalDoor

- Base Parent Class: AActor

This class is designed to take in and merge the aspects of the fuse door and the binary door by having the fuse box and binary switches interact with the class. Similar setup was done in terms of materials with the other doors, as well as implementing the new way event listener binding was done in tick. The functionality binds 4 BinaryCode event listeners, each with their own binary code function and a setpowered event listener from the fuse box. Implementation of the check combination function remained similar with the only change being a check to see if the door is powered from interactions with the fuse box.

This class was designed in part due to the need to merge two or more gameplay puzzles together, and the way I have made my binary switch and fuse box allowed me to quickly make this gameplay feature quickly without having to modify the code in the classes extensively. However, this is one such way I have decided to merge gameplay puzzles together.

Special Enemy Character (Extra functionality)

The special enemy character is designed to take cues from Mr. X from Resident Evil 2 and Slenderman. The design goals in making this class were to have it know where the player is at all times and make an effort to always try and “catch” the player and collide with him. To take cues from slenderman, the design goals were to add a visible screen effect to notify when the special enemy character was around. This was because of another behaviour which is that if the player collides with the special enemy character, the player immediately dies. Thus, balancing was added in slowing their movement speed to half and the postprocess should always notify that the special enemy character was near.

The way I have tried to implement this was through a postprocessing volume by way of declaring a box component and attaching a post process component as a subobject was made to have a post process volume continuously follow the special enemy character. In terms of killing the player, a new event broadcaster for killplayer event was made in this class and their interact implementation broadcasts this event to listeners. This would then be listened in the player class and broadcast the player death event which brings up the end screen.

In terms of trying to get the Mr. X, always seeking the player functionality in, the code I tried to implement crashes the game on play. Thus, this feature was decided to be deprecated.

Note:

In terms of the special functionality, I was hoping that the footstep and dynamically switching the footstep sounds would be enough in complexity to allow for a 2/3’s of the mark allocated to additional functionality. The special enemy character was planned to be more complex as I have mentioned above, but due to problems, I have decided to deprecate that feature.