FIT2097 – Assignment 2 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 A2. For changes in A1, the document has been moved to the Assignment 1 folder and can still be found there.

UE4 Gameplay Framework Classes used:

Slightly Modified (?)

- FIT2097Week3.Build.cs
    - Not really sure if this applies but changes were made to include navigationsystem and gameplaytasks
- NavigationSystem
- GamePlayTasks
    - Blackboard
    - BehaviorTree

Used and modified extensively to achieve design goals and targets:

- Character (EnemyCharacter, MyEnemyCharacter(BP) and PatrollerCharacter(BP))
- AIController (EnemyAIController, MyEnemyAIController(BP) and PatrollerAIController(BP))
- BTTaskNode (GenerateRandomLocationTask, GeneratePatrolPath)
- Actor (All interactable objects such as doors, switches and pickup objects)

Blueprint Grey Area:

- Blackboard (EnemyBlackboard,
- BehaviourTree
- Materials (DoorMAT, SwitchMAT, PickupMAT)
    - BUT all dynamic instances of materials are manipulated in code.

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)

For this class, to facilitate the interaction between the player and the fuse system, a bool hasFuse and the UFUNCTION GivePlayerFuse() was created. This function would be binded to a broadcast event from FusePickup to simulate the effect of picking up and storing a fuse. The function then modifies the hasFuse bool to true to then be checked by FuseSwitch. When the player makes an oncollision event happen with a FusePickup object it sends information to the fusepickup to execute interact implementation and display information. When the fusepickup executes its interact implementation the broadcasted event in turn gets listened by the player and then triggers the GivePlayerFuse function changing the hasFuse bool to true and simulating that the player now has a fuse.

To Facilitate the interaction of taking damage from making contact with enemies, the UFUNCTION() TakeDamage is created. This function would be binded to a listener from a broadcast event from EnemyCharacter to simulate taking damage from them. The player makes an oncollision event happen with an EnemyCharacter object and triggers the execute interact and execute display information. This oncollision event alerts the enemycahracter class that it must broadcast the damage event in their execute interact which in turn gets listened by the player and then triggers the TakeDamage function which would lead to TakeDamage calling on the decreasehealth function that was implemented in a1 (reduced to 5 health lost per function call due to more hazards and ease of testing).

During the implementation of this function, it was noted that due to the nature of collisions, the TakeDamage function was called too many times a frame and leads to the player getting killed in seconds after colliding with an enemy. To better facilitate this then, the addition of the bool isImmune, FTimerHandle ImmuneTimer and the function ImmuneReset() were all made to create a delay in how frequently decrease health can be called.

HealthPickup

- Base Parent Class: AActor

The HealthPickup class now stores and sets up dynamic material instances. This is done to modify the base material that was created for all pickups. For healthpickup, the color is changed to red and is set to 50.f intensity and is set to pulse more frequently to indicate that it is different to other pickups.

VerticalDoor

- Base Parent Class: AActor

The VerticalDoor now stores and sets up dynamic material instances. The GiveKey function that was binded to the event listener from the givekey event from keypickup now sets the material to glow to 50.f, significantly changing its appearance. This is done to show that the door is now interactable. When the player interacts with the door, when the door is opened, the color changes to green to signify that the door is now opened and when it is closed, it returns to a red color.

KeyPickup

- Base Parent Class: AActor

The key now stores and sets up dynamic material instances. This is done to modify the base material that was created for all pickups. For keypickup, the color is changed to green but stays at the pulse frequency of 0.5f. This is to differentiate the keypickup from healthpickups.

AutoDoor

- Base Parent Class: AActor

The AutoDoor now stores and sets up dynamic material instances. The AutoDoor changes its material instance to green and emits at a greater emission value at 50.f when the door is open (interaction implementation is called).

SwitchDoor

- Base Parent Class: AActor

The SwitchDoor now stores and sets up dynamic material instances. The switch door changes color to green from red to signify that the switch has been activated (interaction implementation is called)..

New Classes

FusePickup

- Base Parent Class: AActor

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

FuseBox

- Base Parent Class: AActor

The FuseBox is set up to contain a player reference that is blueprint exposed so we can directly link FuseBox to the PlayerCharacter, placefuse event broadcaster and a dynamic material. On begin play, we set the material instance to red to signify that the fuse box is inactive and needs a fuse. To simulate collecting a fuse and placing a fuse to the fuse box, when a player makes a linetrace to call the interact implementation, a check is made to see if the player hasFuse Boolean is true. If it is false, it will only trigger the basic display information implementation. If it is true, however, the place fuse event is broadcasted and the material is set to blue to signify that the fusebox is now “powered up” so to speak.

FuseSwitch

- Base Parent Class: AActor

The FuseSwitch set up to contain a FuseBox reference that is blueprint exposed so we can directly link FuseSwitch to the FuseBox, FuseOpen event broadcaster, PlaceFuse function that is binded to an event listener, a fuseActive Boolean and a dynamic material. On begin play the material is set to red to signify that because the fuse box does not have a fuse, the switch is also inactive. The FuseSwitch becomes “active” when the PlaceFuse event is broadcasted and its function is called. This function sets fuseActive to true and changes to color of the material to blue to signify that it is now interactable. A linetrace from the player would then call the interaction implementation would check if fuseActive is true. If it is false, we tell the player to make sure the fuse box is active. If it is true, it broadcasts the FuseOpen event and changes the color of the switch to green to signify that it is activated.

FuseDoor

- Base Parent Class: AActor

The FuseDoor is set up to contain a FuseSwitch reference that is blueprint exposed so we can directly link FuseDoor to which FuseSwitch it will be dependent to, FuseOpen event listener binded function and a dynamic material. When the listener gets information that FuseOpen was called, it calls the binded FuseOpen function. The Function opens the door and changes its material instance to green and emits at a greater emission value at 50.f when the door is open.

BinarySwitch

- Base Parent Class: AActor

To facilitate the basic requirements of the spec “array of 3 on/off switches” 3 event broadcasters and event dispatchers that pass on the Boolean variable code as well as a currentCode Boolean, binaryID and a dynamic material were implemented to the class. BinaryID and currentCode are all blueprint exposed so that we can set which BinarySwitch would be which. When a linetrace occurse with the binary switch, the switch will change the value of the currentCode to either true or false. It will then broadcasetwhich BinaryCode event broadcast to use depending on its binaryID (ex: binaryID 1 = BinaryCode1 event). In the tick event, it constantly checks its currentCode value and changes color to green if currentCode is true or red if its false to signify “on” and “off” states

BinaryDoor

- Base Parent Class: AActor

To interact with the 3 switches that will be in the level, three BinarySwitch references were made to directly link BinarySwitch1 in the world to BinarySwitch1 in the code, more variables were also created and exposed to blueprints in targetCode1, targetCode2 and targetCode3 and tempCode1 tempCode2 and tempCode 3. targetCode is exposed so that we can set the code combination we want in blueprints and have it whatever we want tempCode is also exposed for debug purposes. A dynamic material is also set up based on the default door material. Every tick, it is listening for a BinaryCode event and when it receives information of a BinaryCode event, the function takes in the Boolean variable code that was passed through and stores it in a tempCode and triggers the CheckCombination function. The CheckCombination function then goes through an if check that checks if each tempCode matches the targetCode we have set in blueprint. If it is true it will open the door and set the color to green and emit a glow. When linetraced on, it will display basic information.

EnemyCharacter

- Base Parent Class: Character

This class does not stray far from the implementation found in the tutorials code was implemented to allow the character to rotate when moving and set the rotation to not snap too hard. Modifications were made to implement the interactable interface to be able to send the TakeDamage event broadcast to the player when they collided with an enemy and displays information of said enemycharacter. Changes to MaxWalkSpeed here were also done in the constructor. Due to debugging purposes, at the

time of writing, MaxWalkSpeed is set to 200.f when showcased in the video, the value will be changed to 540.f (10% slower than the player max walk speed).

MyEnemyCharacter(BP)

BP class based on the C++ character. Followed the tutorial on how to set this up, included the mannequin mesh and animation class and set the AI Controller to MyAIController. This class is set up as the seeker that randomly seeks the player in the level.

PatrollerCharacter(BP)

BP class also based on the C++ character. Followed the same implementation as the seeker class, except it is supposed to have the PatrollerController.

EnemyAIController

- Base Parent Class: AIController

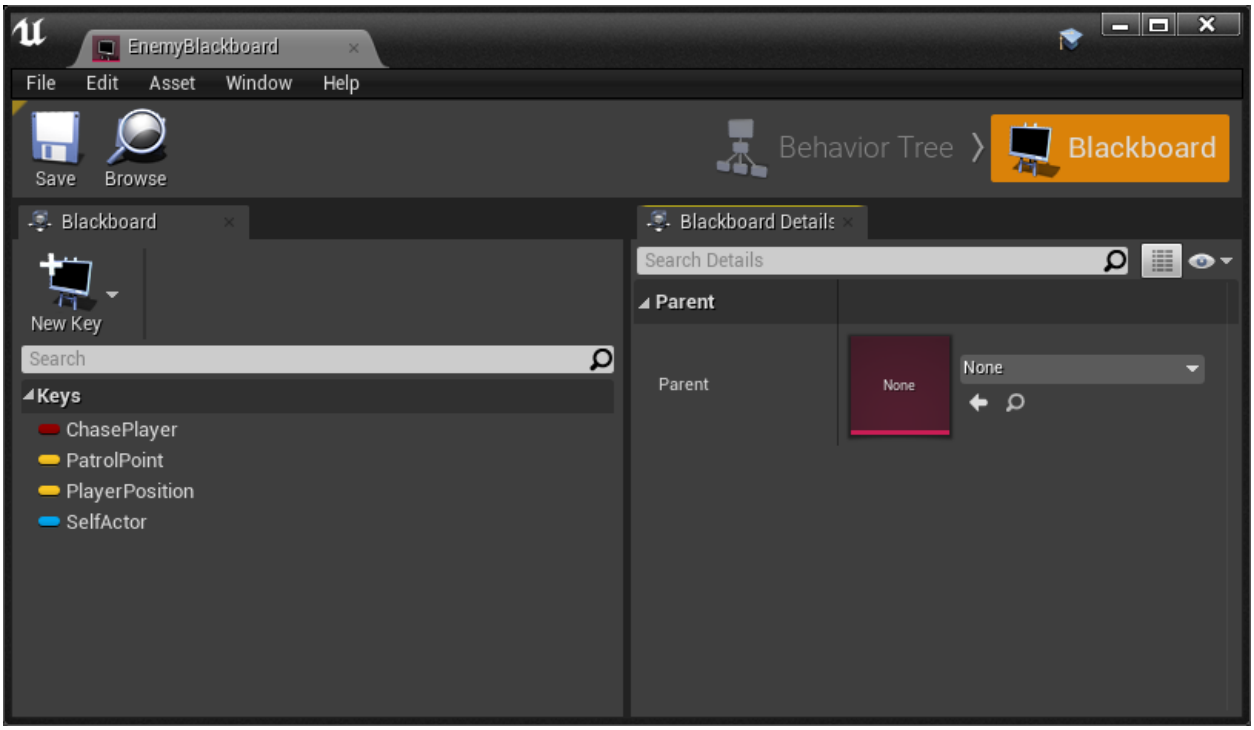
This class also does not stray too heavily away from the implementation shown in the tutorials. We have set up override functions in beginplay, onpossess, tick, getcontrolrotation and onmovecompleted. We have two custom functions such as the shown GenerateNewRandomLocation that finds random points for the AI to go to randomly and a hardcoded GeneratePatrolPath that updates the blackboard patrolpoint value with a set of predefined coordinates from the PatrollerWaypoints TArray and patrol count. A UFUNCTION OnSensesUpdated that uses existing sight system and variables that allow us to define our ai's sight in sightradiues, sightage, losesightradius and fieldofview that we will configure using SightConfiguration. We will also be using the NavigationSystem in conjunction to simulate movement and trying to find our player character in the world. We have also made Blackboard, BehaviourTree references that we can directly link towards in our blueprints.

Information in general is sent via the behaviour tree which tells the AI what to do when conditions are met and built in our functions are senses that can change the conditions found In the blackboard and behaviour tree. For example, when the player is found, it triggers the OnSensesUpdated function. The function then casts if the stimulus was sensed and if it was, updates the playerposition variable in the blackboard and sets chaseplayer to true which sends information to the behaviour tree that it is now possible to chase the player.

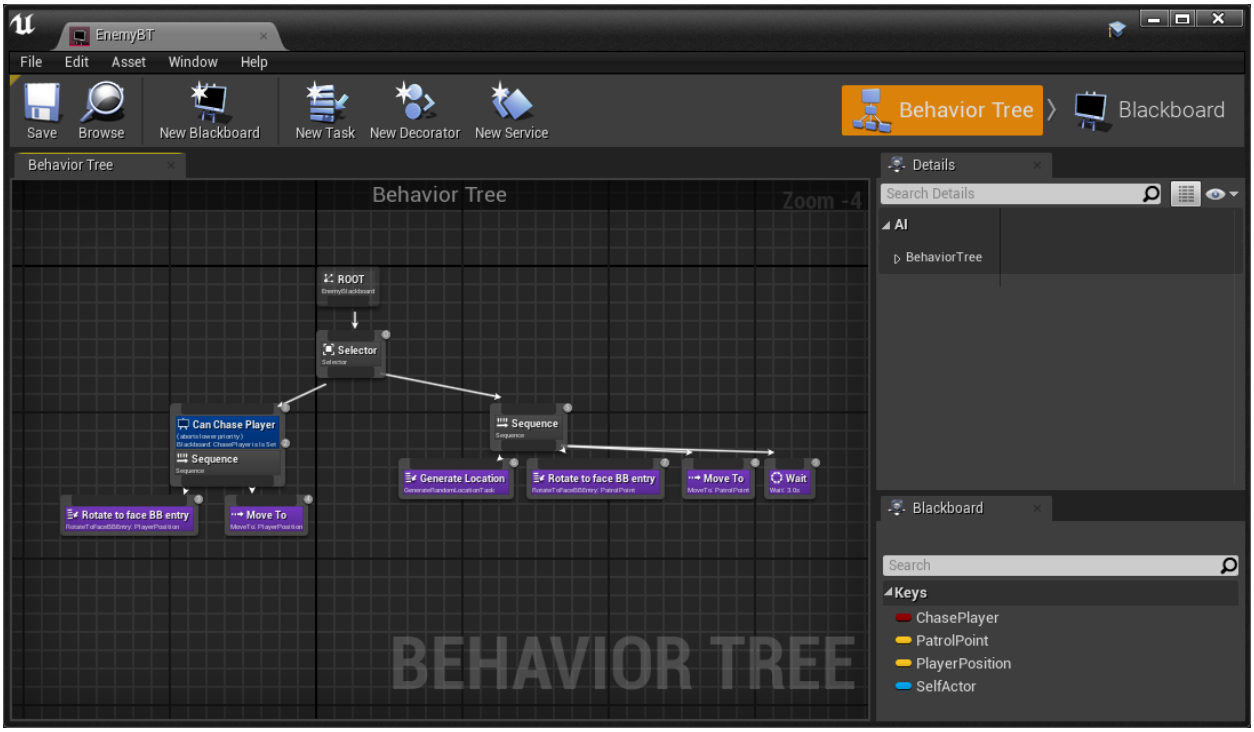
Due to the dynamicity of the senses system, once the enemy loses sight of the player for long enough, the TargetPlayer would eventually become null and lead to the else statement where it is no longer able to chase the player, thus it updates the blackboard to clear the value of chaseplayer

MyEnemyAIController(BP)

BP class directly linked to the EnemyBlackBoard…



and EnemyBT

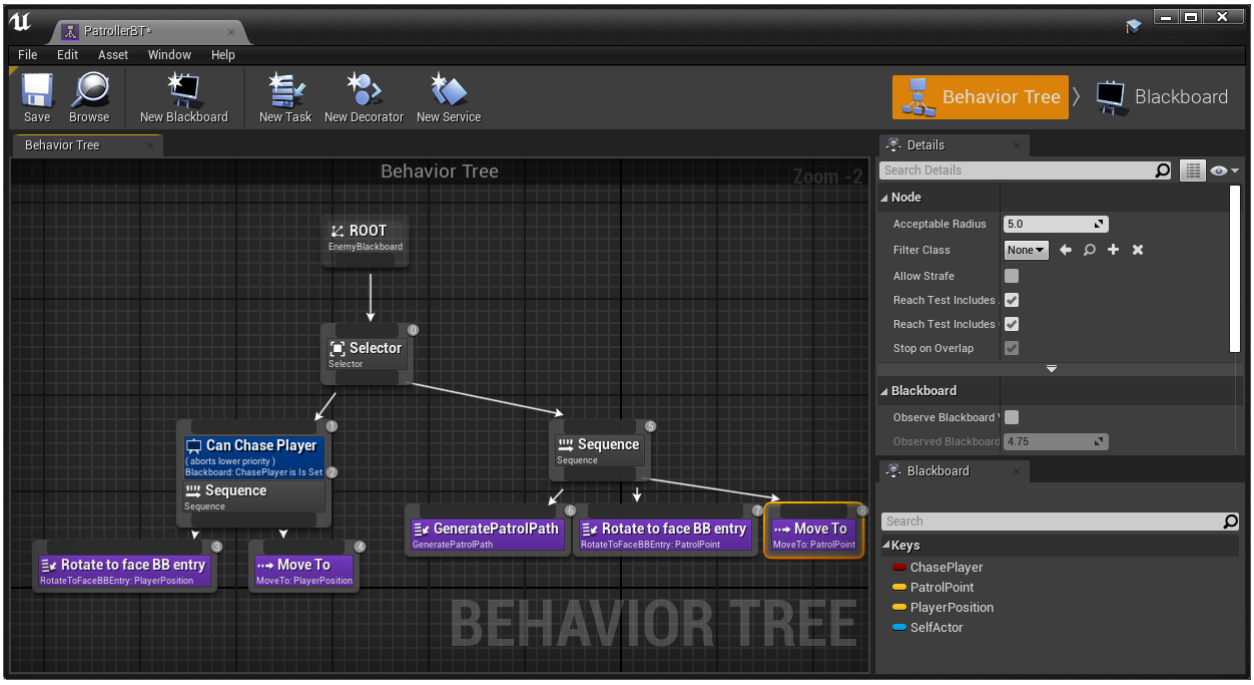


It would check if the enemy can chase player based on if chase player is set to true or false, in which if it can it would skip the lower priority tasks set in the second sequence an move to the playerposition. Else, it would generate a random location, rotate, and move towards that location. After which it will wait a

while. Until a stimuli is inserted where information about the player is added, it will keep looping the lower priority sequence.

PatrollerAIController(BP))

Uses the same blackboard as the regular AI controller but in the behaviour tree, is set to call the generate patrol path function instead



GenerateRandomLocationTask

- 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 generatenewrandomlocation function that will update the patrolpoint to a random location

GeneratePatrolPath

- 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 generatenepatrolpath function that will update the patrolpoint to a patrolwaypoints array index.