Assignment 2 project documentation

NOTE: This is my Assignment 1 project documentation with changes that will be highlighted purple.

Basic premise

You are a little child who is cursed in your house! Quickly solve the puzzle, get the hidden relic in the storage room, and go back to your room before you lose all your mental energy and go insane!

Classes

Unreal Engine 4 Gameplay Framework classes:

AAIController

The class is responsible for the AI's sight and traverse behaviours.

AGameMode

I use this class to implement the game states logic and to bridge the communication between events happening in the game to the menu overlay, HUD, end game screen, etc.

AACharacter

This class supports player's input, as well as pre-defined functions for moving around the 3D space, so this will be my choice of parent class for my player class. Within my main player class, some more UE4 classes that we will encounter includes:

- UCameraComponent for creating the player's perspective in the game. This class supports various controls over exposure, tint, etc. which I will be using for some indicative effects in-game.
- TArray for storing inventory items. It is chosen because of its ease of use, providing all the operations that I need to manage the inventory (find, delete, add, etc.).

This is also the parent class of my Enemy class as its movements handling is smoother than doing by hand.

AActor

Because of its versatility as a game object on-screen and off-screen, this class is the parent class for pretty much all the items that are in the game. Within the Actor class, there are additional UE4 classes that we will encounter in the project:

- UStaticMeshComponent for representing the item in the game. It loads a mesh that is already in the project (in my case all the meshes are taken from Starter Content) as well as give various controls for translate, rotate, and scale to shape the mesh to our need. Some Actor even uses two of these components to combine into a cohesive object.Interface

The provided UE4 Interface class helps me implement different kinds of items with different types of interactions. For example, an Interactable interface allows me to implement interact functions that the player can trigger, or Inventoriable interface for items in inventory.

UBTTaskNode

This class is used to create custom tasks in the behaviour trees that are used for enemies' behaviours.

UUserWidget

I use this class to implement all my UI elements. This class provides a way to add other widgets to the base panel, which then can be constructed to different types of overlay. The classes that are use with this class includes:

- UCanvasPanel, which acts as the base panel for the UI elements. This class was chosen because it is easy to replicate in a Blueprint and allows for flexible widget placements using UCanvasPanelSlot.
- UButton, which is the button class.
- UTextBlock, which contains text.

My implemented classes

AAssignmentGameMode

This class is responsible for the game states logic like switching between states, pausing appropriately, displaying, and updating UI elements based on the status of the game. The only reason why this was chosen instead of the GameModeBase counterpart is because of the RestartGame function. The game mode subscribes to the player's on lose event to trigger lose state and oversees updating the UI elements, including the health bar.

A Blueprint version of this class was created to change the default pawn class to the Blueprint version of the Player class.

AMainPlayer, inheriting ACharacter

This is the player class, where the logic for movement, interactions, and logistics (health, inventory…) happens.

- Movements are handled by the default ACharacter MoveForward and MoveRight implementations, without jumping for the purpose of the game.
- Item interactions are done using a trace line that simulates the character's reach to the item. The reach length can be modified when equipped with an item that can extend it. Consumable item will be consumed when the player presses the interact button instead of the normal tracing.
- Health will start unharmed, until 3 seconds later, when the cursed effect is applied, gradually decreasing the player's health. The player can eat Candy or turn on Matches to stop the effect.
- The player class also contains inventory logic, with adding, removing, checking a certain item is in there, and get currently equipped item available publicly. The inventory is essentially a TArray of pointers to AActors that has a check before adding an item to see whether it is from Inventoriable.
- Some camera effects are added to indicate what state the player is in: Normal, Heal Draining (Cursed) or Match Light On.
- The class will scan around it every tick for nearby enemies. If there is, the health decay rate is greatly increased.
- Lose condition will now broadcast an event to notify the game mode.

A Blueprint version of this class was created to further customise the collision handling.

Interactable interface

Allows the item to have an interact and showcase function. Interact will be triggered when the player presses E to interact with it. Showcase will be triggered when the player is looking at the item

within reaching length. Interactable also provides a way to pick up item through the Interact function.

Inventoriable interface

Allows for a check for items that can be put in the inventory. Provides OnEquip and OnUnequip function to perform actions when the item is equipped or no longer equipped, respectively.

Consumable interface

Allows for item to be consumed when equipped with the Consume function. Also provides a way to check whether an item is consumable (like the Match).

ACandy class, inheriting AActor and Interactable

Upon interaction through Interact function, the player will be healed and the debuff will be purged for a short duration. Contains a gold sphere mesh for distinction

ADoorKey class, inheriting AActor, Interactable and Inventoriable

Upon interaction through Interact function, the player will pick this item up and put it in the inventory because of Inventoriable. If the player equips this item and open the designated locked door, the door will be unlocked, and the key will be gone. Contains a rust pyramid mesh for distinction.

ADoorTrigger class, inheriting AActor and Interactable

Upon interaction through Interact function, the designated door will be unlocked and opened. Contain a base mesh that can be then customised in the editor. Now uses Event Dispatcher to unlock the door.

AEndPointTrigger class, inheriting ABoxTrigger

When the player overlaps with this trigger and is holding the win condition item, the player wins. Has an AActor to be set in the editor as the win condition item.

AInteractableDoor, inheriting AActor and Interactable

This object has 2 mesh components, which are a door frame and a door. When interacted with, the door will be rotated 90 degrees to simulate the “open door” action. Has a Boolean IsOpened to keep track of the open status.

ALockableDoor class, inheriting AInteractableDoor

The same as AInteractableDoor, except it is locked initially (kept tract of by IsLocked Boolean). Can be unlocked using a designated item, which is an AActor that can be set in the editor, or a ADoorTrigger that the door subscribes to listen for unlock events. When locked, it cannot be opened by interacting.

AMatch class, inheriting AActor, Comsumable, Interactable and Inventoriable

AMatch can be used at any time from the inventory to halt the health draining effect without healing the player. In the game world, the player can interact with it to put it in the inventory. Has a copper cube mesh for distinction. It also provides some protection to the player in the form of a stealth mode that prevents the enemy from chasing the player.

AStatue class, inheriting AActor, Interactable and Inventoriable

AStatue acts as the win condition for the prototype. When interacted, item will be picked up and be put in the inventory. Has the mesh with the globe and the Unreal Engine logo for distinction.

AStick class, inheriting AActor, Interactable and Inventoriable

The stick can be used to extend the player's interaction reach when equipped (specifically in the level when the key is up high in the Mom Office Room). In the game world, the player can interact with it to pick it up and put it in the inventory. Has a wooden pipe mesh that is stretched out to look like a stick for distinction.

UMainMenu, inheriting UUserWidget

Contains 2 UButtons for start and quit that is available publicly to have functions from game mode hooked to it, as well as UTextBlock to display how to play guide.

UInGameHUD, inheriting UUserWidget

Contains a UProgressBar for health display, an array of UTextBlock for displaying the inventory (yellow text for the equipped item), a UTextBlock for describing the item the player can interact with (LookAtItemTooltipTextBlock) and a UTextBlock for describing the effect of the interaction (InteractedItemTooltipTextBlock).

UPauseScreen, inheriting UUserWidget

Contains a UTextBlock that says Paused.

UWinScreen and ULoseScreen, inheriting UUserWidget

Both contains 2 UButtons for restart and quit that is available publicly to have functions from game mode hooked to it, as well as UTextBlock to display whether the player win or lose.

AFusedDoorTrigger, inheriting ADoorTrigger and AFuse, inheriting AActor, Interactable and Inventoriable

The fused door trigger is a door trigger that can only be activated when the player is holding a designated fuse item when interacting with it. A fuse is just a simple item that can be picked up via the Interactable and Inventorable interfaces.

ASwitchPanelTrigger, inheriting ADoorTrigger and ASwitch, inheriting AActor and Interactable

The switch is an actor that contains a Boolean value for the switch panel to take advantage of. When the switches match a designated pattern on the panel, it will broadcast an unlock event for the door that is listening to it to respond.

AEnemy, inheriting ACharacter.

This is only a basic class to setup the enemies so I will talk more about the two types of enemies:

- Ghost can fly through walls and will patrol around the house. No
- Walker will run around the fuse to guard it.

They are all slower than the player and will greatly enhance the player's health decay rate when they stand close enough to the enemies.

UCheckMatchLitTask, UGenerateRandomLocationTask, USwitchPatrolPointTask, inheriting UBTTaskNode

These classes are custom nodes in the Behaviour Tree that aids in controlling the Ais' behaviours. Their names are self-explanatory:

- UCheckMatchLitTask checks if the player is lighting up the match. This will block the player chasing sequence.
- UGenerateRandomLocationTask is taken from the labs. It generates a random location on the Nav Mesh and points the AI to head there.

- USwitchPatrolPointTask switches between the two patrol points of the fuse guard.

Information passing

In general, the UI elements will have its widgets public so that the Game Mode can alter it to its need.

The Game Mode also have public functions for the Player and the EndPointTriger to inform of the game events like item interactions, win condition being reached, etc.

In general, the Interactable items knows the reference to the player so that they can take effect on the player. That also applies to Inventoriable items and Consumable items. This is achievable through specific public functions on the player class that modify values on it like Heal, LightMatch, etc..

Items that upon interaction triggers other items behaviour have an AActor variable that can be set later in the editor.

The Game Mode is pretty much the bridge between game events (mainly fired from Player and EndPointTrigger) and the UI, updating the on-screen widgets as needed. To achieve this, the Player have a point to the Game Mode that is passed to it by the environment (specifically World object through GetWorld). This is the same way EndPointTrigger can call Game Mode.

From Assignment 1 feedbacks, the game now uses Event Dispatcher to trigger door unlocking events and lose condition. There are also new ways for classes to interact with one another in the form of Blackboard and Behaviour Trees, where the controller will pass on details to the Blackboard and it will then pass it on to the Behaviour Trees through Blueprints.