BomberMan Game

A. Background

In this project, we aim to make a Bomber man game in 3D using Unreal Engine 4. It's object to a game mode that player will control a Bomber man character that can throw bombs and using punch as an emergency measures in case of run out of bombs.

The goal is to beat the zombies; also, there are some extra content in the game (such as Parkour in the level 1 second area and pushing lighting boxes in level 2).

B. Design Principle

In the structure, Player's hero inherits from the class *Pawn* or *Character* (if need to act like human), and our bomber man is just a human-like hero, so we choose to inherit from *Unreal Engine* default class: *Character*.

As a character that can throw bombs and take damage, we need to create another thing: bomb, which should be the item in the game, so we choose to inherit from *Unreal Engine* default class: *Actor*.

So as the bomb actor, it needs a bomb model, it's basic. Then besides the simple model, we need to add the *particle system* on it to express the fire and the burning of the lead wire. The actor needs to be used to throw, so we'll add *projectile movement* on it. *Projectile movement* is a component in *Unreal Engine*, which makes the actor obey the gravity and collusion. The bomb actor finally defined including:

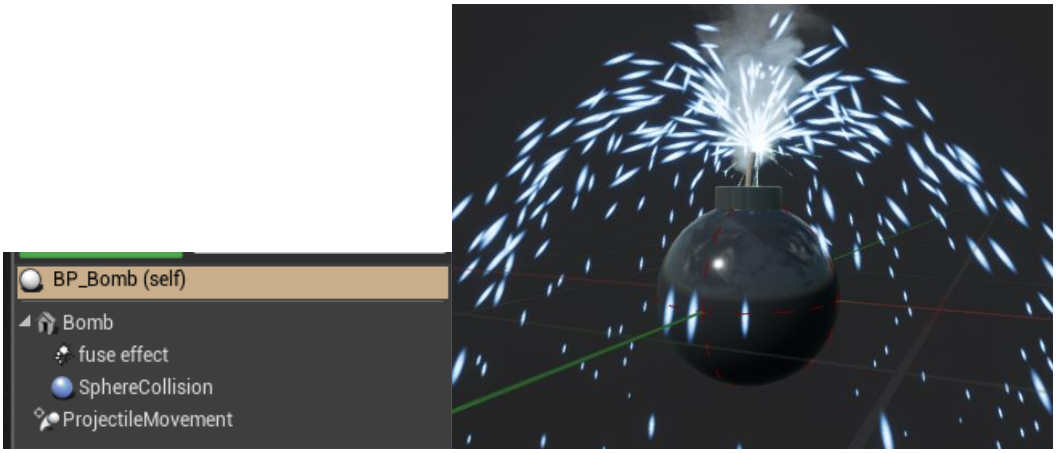

Fig 2.1 The combination of the bomb actor.

Then we define the bomb actor behavior: When the bomb is created, it needs a random velocity (otherwise the situation will look too inflexible for the bomb drop without any roll). So in the construction function, we need to give the bomb a random velocity.

Another thing we need to take is to make the burning point from the start point to the end point. It requires the slow moment of the particle effect. (You can see the key point below)

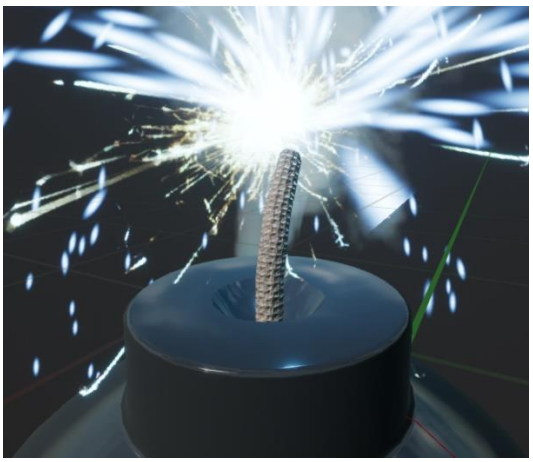

Fig 2.2 Exploding lead part of the bomb.

Then the bomb will explode and cause damage to the characters around it. Here we make a killing radius, the exactly solution is to scan the space in a defined radius, adding all the damageable characters into a list, and then makes them get hunted.

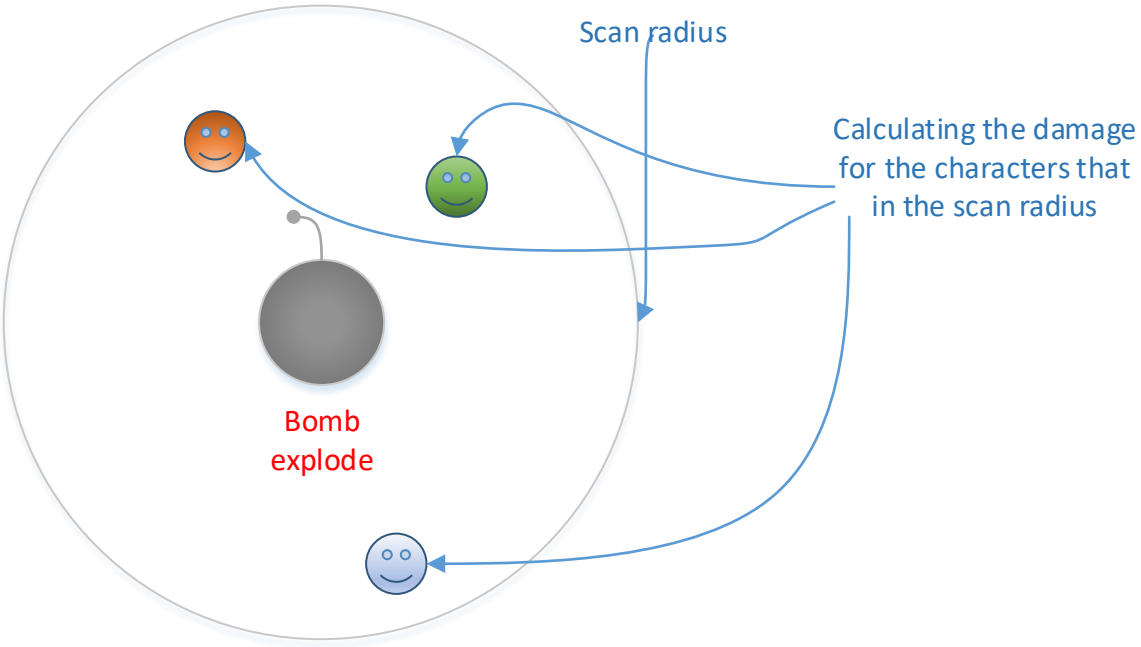

Fig 2.3 Bomb killing radius

In the graph describes the damage process, and how the process was realized by logic. The exploded bomb will be destroyed and release its memory in the computer.

C. Model Discussion

We discuss the player character. WARNING: the player character has VERY MUCH logic to be used (including control, action, player UI, use/pick items and so on), so I may not describe them in the same time. Now I will explain it in several parts:

① Character movement

As a character controlled by the player, the bomber man need to have the ability to respond the input from the player, which will be taken over by a default *controller* class (already defined in the *Unreal*).

② Game attributes

Then it needs some attributes such as health point (HP), bomb numbers, energy, attack damage and so on. So these attributes should be included in the Bomber man class.

③ Spectator

The character player play should be observed at certain position, it should also has the collusion, so we set the camera actor (presents the rendering position) behind the character, the use a capsule body (to simulate the collusion of the character) at the position of the mesh.

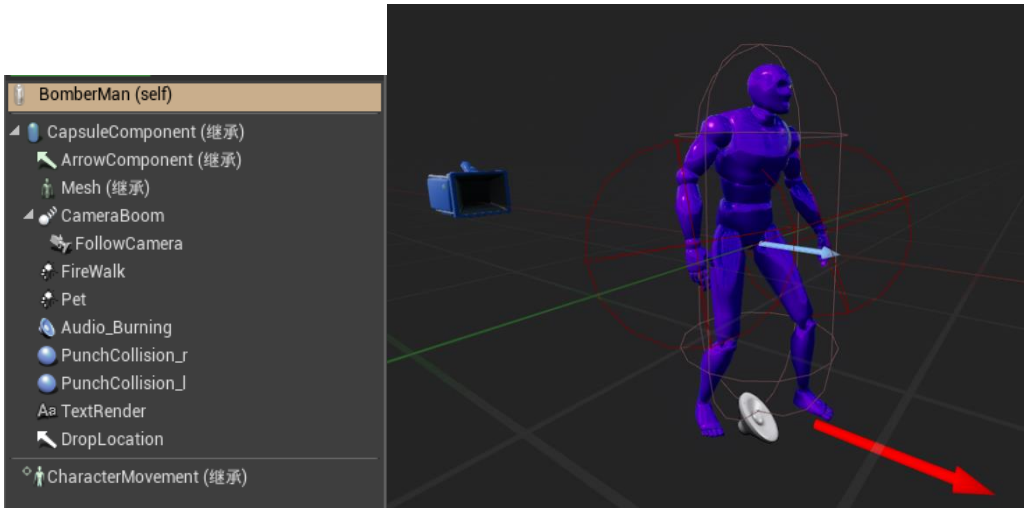

Fig 3.1 Player character structure design

④ State machine of the mesh

Different actions should be played when the character is in different states. The main state include: *stand, run, jump, at air (dropping), punch*. Each state represents a kind of mesh state, to arrange the different animations to be played through different states, we can define each state to other state with different conditions.

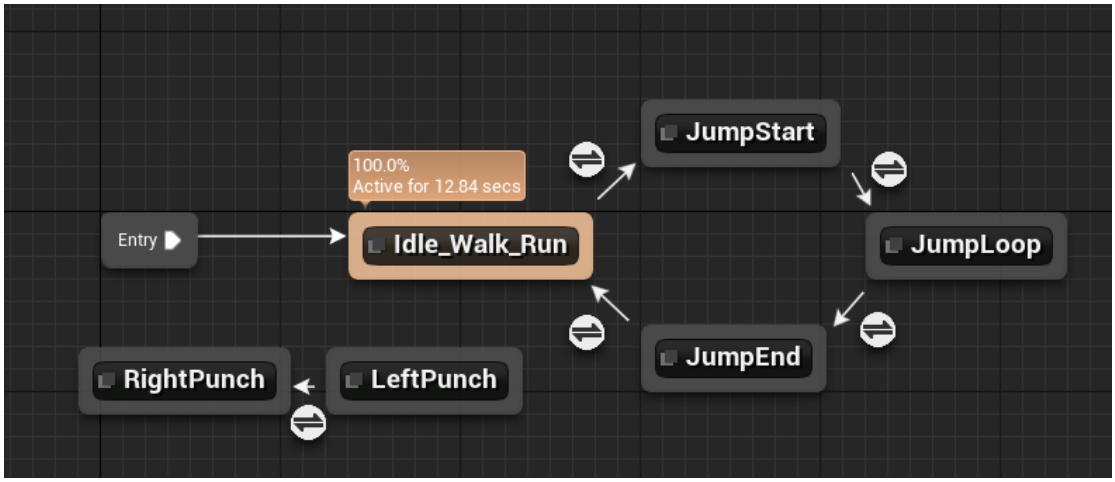

Fig 3.2 State machine of the character

Attention, here *walk_run* state is a mixed state, which means the action is at both "walk" state and "run" state with different weight, what decides weight is the speed of the character movement.

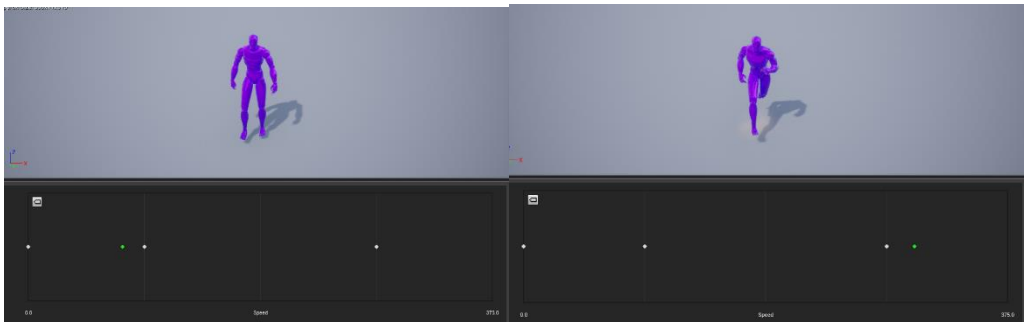

Fig 3.3 Mixture space of the character movement

⑤ Player UI

In game, I designed item pick up system and pause system. It means the UI widget should be shown each time when player wants to use item or pause the game. So we will create *Widget* class defined in *Unreal Engine* to be included in bomber man character class. It means the character includes the UI, usually set hidden in game, and when player calls them, they'll be shown.

Two widgets are:

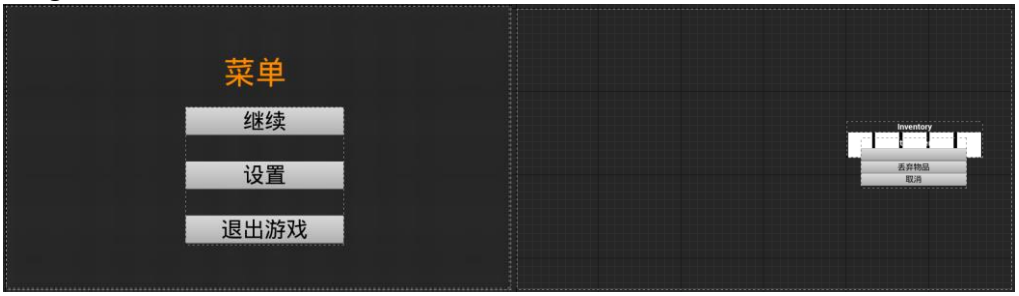

Fig 3.4 Pause UI and inventory UI design

When the pause menu is called, the whole screen will be covered by blur component, which makes pictures fuzzy. The item menu only includes item bar and action button.

As for items to be picked up, we'll discuss later. The final result is:

Fig 3.5 UI in real game

So, let's conclude the structure we now design:

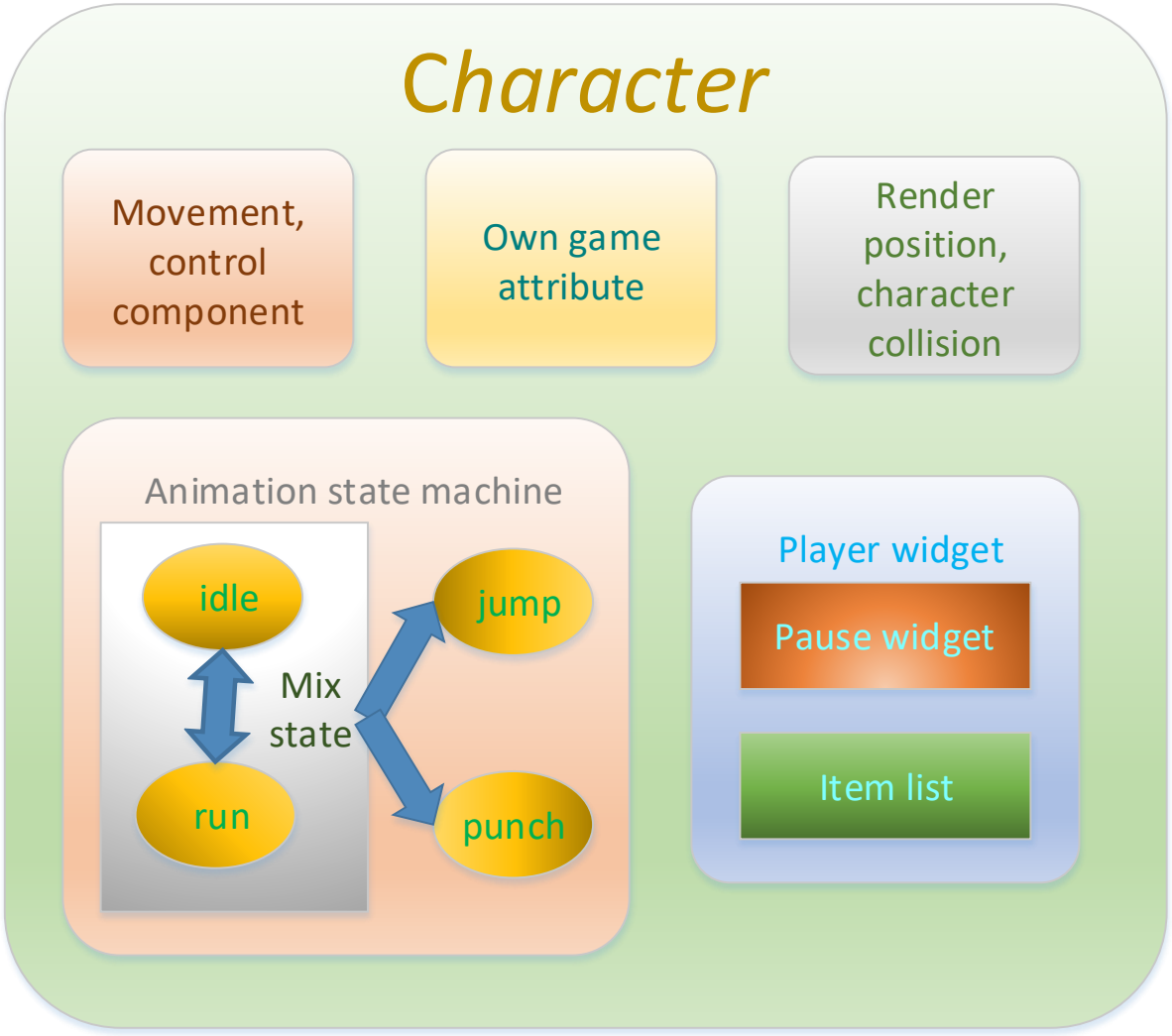

Fig 3.6 The main structure of the character class

Now we discuss about the bomb spawn process: When the character choose to throw bomb, a bomb object should be created at the front position of the character. Then as the bomb object design, it'll have an initial random angular velocity, and several seconds to be detonated.

As we mentioned before, another way to cause damage is to punch the zombie. And we

also need the punch to be more … realistic. So there will be a sphere collusion bonded at the character's hand. When the character is not punching, the collusion will not be enabled, when the character punches, the collusion will be enabled so the punch will really push all the objects in the 3D world. What makes the damage is that when the collusion is enabled, if the overlap event happened on the sphere collusion, the damage will be caused to the other actor that overlaps the collusion. Then the damage system is completed.

As for zombies, also, it's a subclass of the character class in Unreal Engine, what it has is similar to the bomber man, except the UI and the ability of throw bomb. Here to make the zombies to have the ability to cause damage, I'll also bind the collusion on it. And since the attack animation can be played when the zombies get close to the bomber man, I made another state machine: When the current speed is lower than 90, zombies will play the attack animation. Since the damage sphere has been bonded to its hand, attack animation will of course cause damage to the bomber man. This is the main design of the zombie.

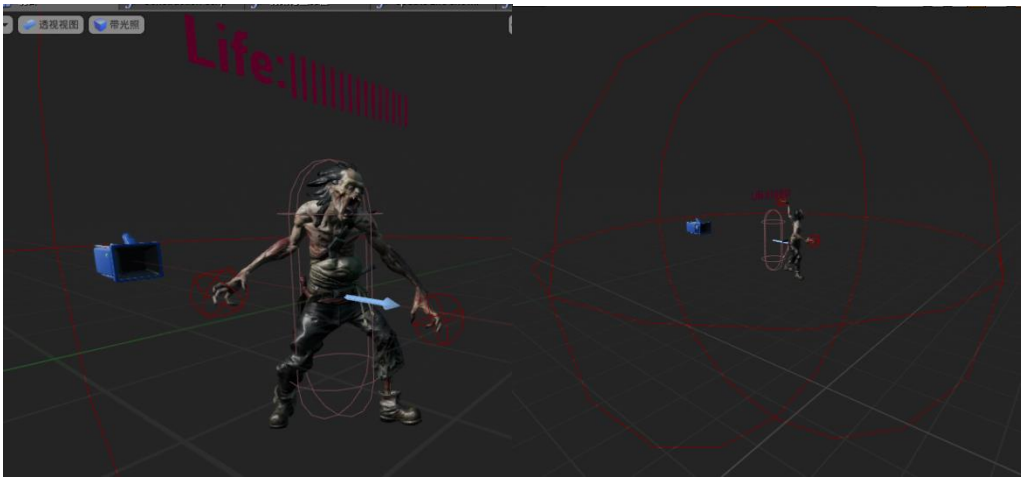

Fig 3.6 Zombie model and structure design

The zombie should be able to chase human player when the player get close, so we add a sphere to "feel" the bomber man coming, by the time an object enters the sphere, check if the object is the player, and if it is, the zombie will chase and attack the player.

To make the fighting ground more abundant, we'll also use items that has physical simulation and destructible in the game. The destructible will be set to bear impact damage, so punch will destroy them.

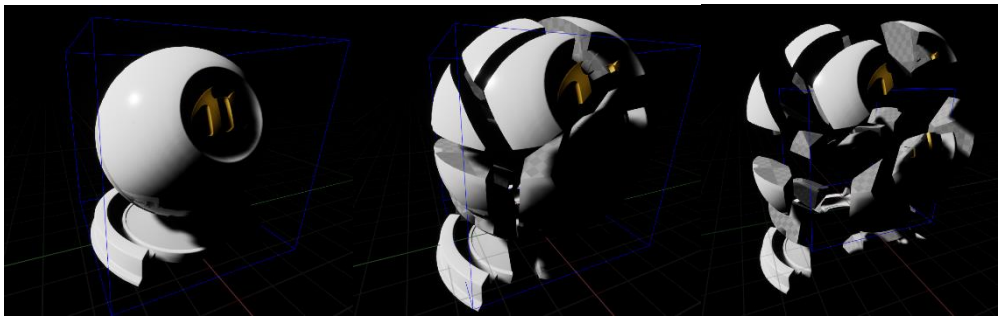

Fig 3.7 Destructible design

The last system is the item system. Its UI has been mentioned through the introduction of the bomber man character. Now we discuss how to design and realize the whole system. First, as you can see in the UI design, I only arrange 5 spaces to be used to place the items, so the widget will have a list of the Inventory Item which is a class I design with four elements: item name, item picture, item introduction and item effect.

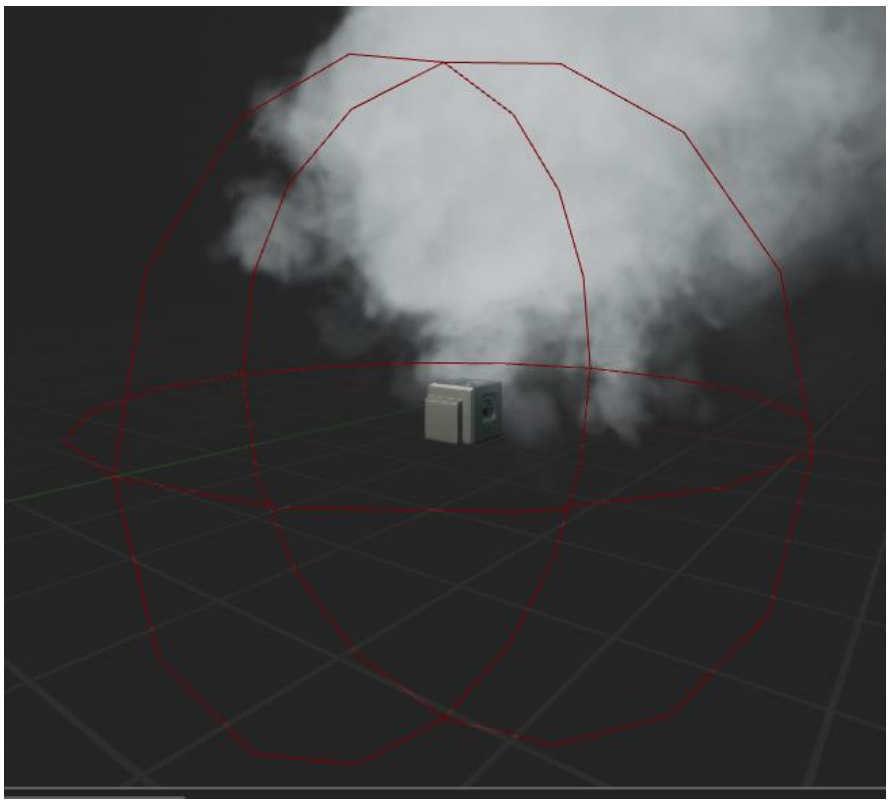

Fig 3.8 Pickup items structure design

All the items appear in the game world is the instances of the Inventory Item class, but they have different effects and different pictures, also, different introductions. The item will have a sphere collusion, which means the scope that the character can pick it. When character chooses to pick, while its item list is not full, the item in the world will be destroyed and the instance will be added to the list.

Fig 3.9 Pickup items to be used in game

Clicking the different items in UI widget will cause different text and effects in game. In addition, we design that the items should be spawned when the zombie is killed by the player.

But it's in probabilistic. In game, we set the probability as 18%, and each kind of items has 6% to drop through the death of the zombie.

Now let's conclude the whole game playing process logic:

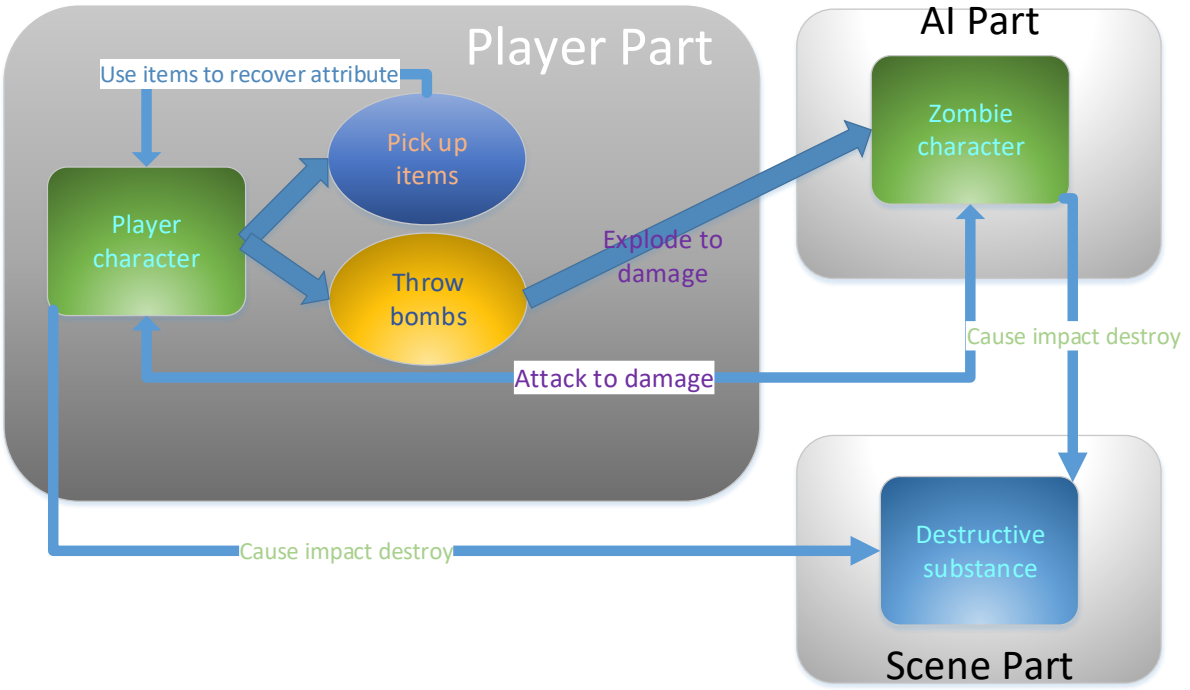

Fig 3.10 Whole game playing process logic

D. Demonstration

Here in the final result, I design two levels, one for day and one for night environment. In level 1, we aim to make player know the basis controlling, including how to use item, throw bombs to the zombies and some jumping skills.

Fig 4.1 Title UI and showing quality setting UI

In the title, there we design the quality setting for different kinds of computers.

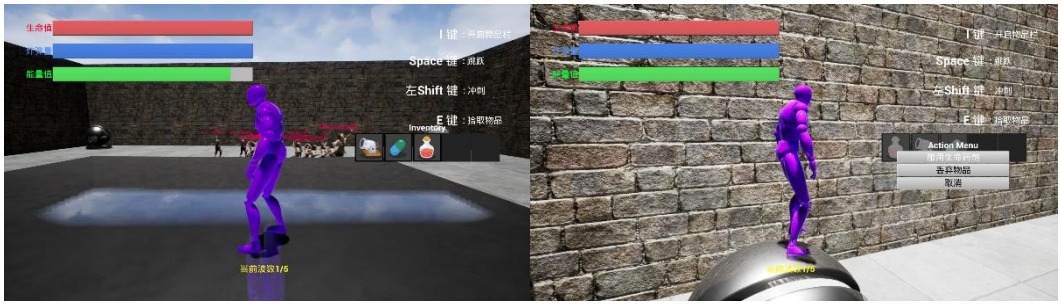

Fig 4.2 Inventory list to show and use items

Fig 4.3 Using items with different action text

An inventory list contains different items with different effects. Each item has its own name and picture and action text. Here is the inventory system for using or dropping items. When the character picks up an item, it will be added to the inventory list to show.

Fig 4.4 Pause menu and hint text

Pause list is also an important part in user's widget. Here we design three button with a blurring background. And by the time user opens the pause menu, the whole game speed will slow down to 2% of the original speed. Kind of like decision time in GTA5.

When user uses an energy drug to get more energy than the maximum, the outer energy won't recovery after used.

E. Conclusion

In this project, we design a 3D game using *Unreal Engine 4*, which consists of UI widget, objects, particle system effects and level logic flow. We finish this project all by

ourselves except using the images and voices.

At the game playing system level, we also make the hurting system: by bomb exploding and hit punch, the exploding system is made by scanning and injury settlement list, and the hit damage is caused by physical component collision.

Fig 5.1 Character can attack by bomb or by punch

At the game UI widget level, there are mainly two kinds of UI: real-time update widget like life bar, bomb bar and energy bar, option choosing widget like button widget. The first kind of UI is bonded with character's attributes, so their change can be applied to the UI, the option choosing widget is hidden under the character class normally, and shows up when needed. In fact, this kind of UI is often combined with effect UI, like blur screen and background picture widget.

Fig 5.2 Many kinds of UI can be shown in the screenshot

At the game playing logic level, we design 2 maps with each map 3 scenes combined. Each scene has different playing strategy. In some scenes, you need to defeat the zombies using bombs and punching, while in other scenes you should have a good controlling on jumping and moving. Map 1 and map 2 are in day and night, it means the basic environment can differ map to map, at night, game will be more difficult, and more, player needs to use light sources in the game, only then will the player has the chance to beat the zombies.

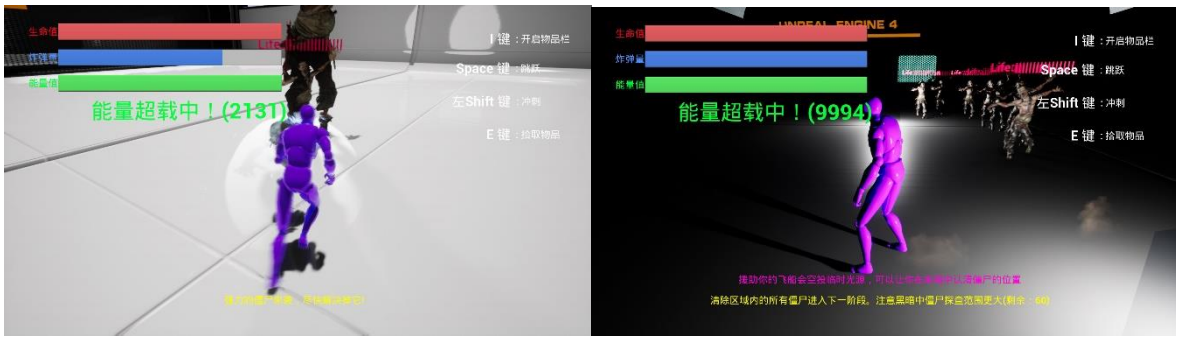

Fig 5.3 Different playing strategies in different maps and scenes

About the project data structure:

Since all the systems above have more or less relation about data structure, such as the arrangement of the items list, or the bomb damage about the bomb.

Here we use another level of design using data structure.

As you can imagine, each zombie is the character class, which has many data to store and it's hard to be created. If when the zombie dies, delete it, and when new zombie is born, create another instance of the zombie class, it'll take much time for the program creating and garbage collection. So here, we just make there are certain number of zombies to be created, and by the time the player kill a zombie, the instance will be hidden with its HP reset, and by the time another zombie should be created, this instance will replace the new instance to be created.

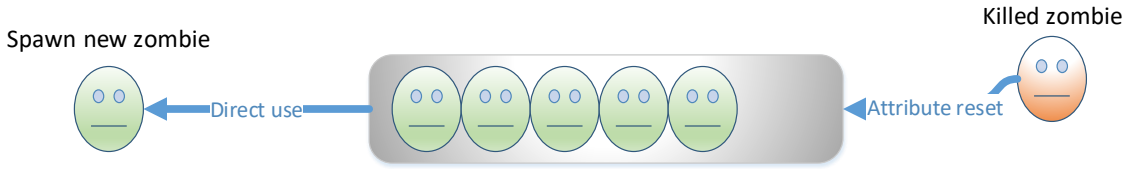

Fig 5.4 Recycle systems in the game process related to data structure

The items can be applied in the same recycle system.

Another data structure is about item searching. Some time to calculate the whole different items in the game world, we need to know all the items currently in the world, and classify them quickly, but each item is a huge data, here we define the hash search:

$$item_{id} = hash(item) = 1 \times HP + 1000 \times bomb + 1000000 \times energy$$

So each item can be arranged to different space by hashing. When we count, we only count if a series of numbers is allocated, that makes the counting faster by data structure.

F. Appendix

Extra data about the data structure analysis I made for *Unreal Engine 4* will be list in the attachment. (Analysis from the source code of *Unreal Engine 4*)

Fig 6.1 *Controller* class in *Unreal Engine 4*

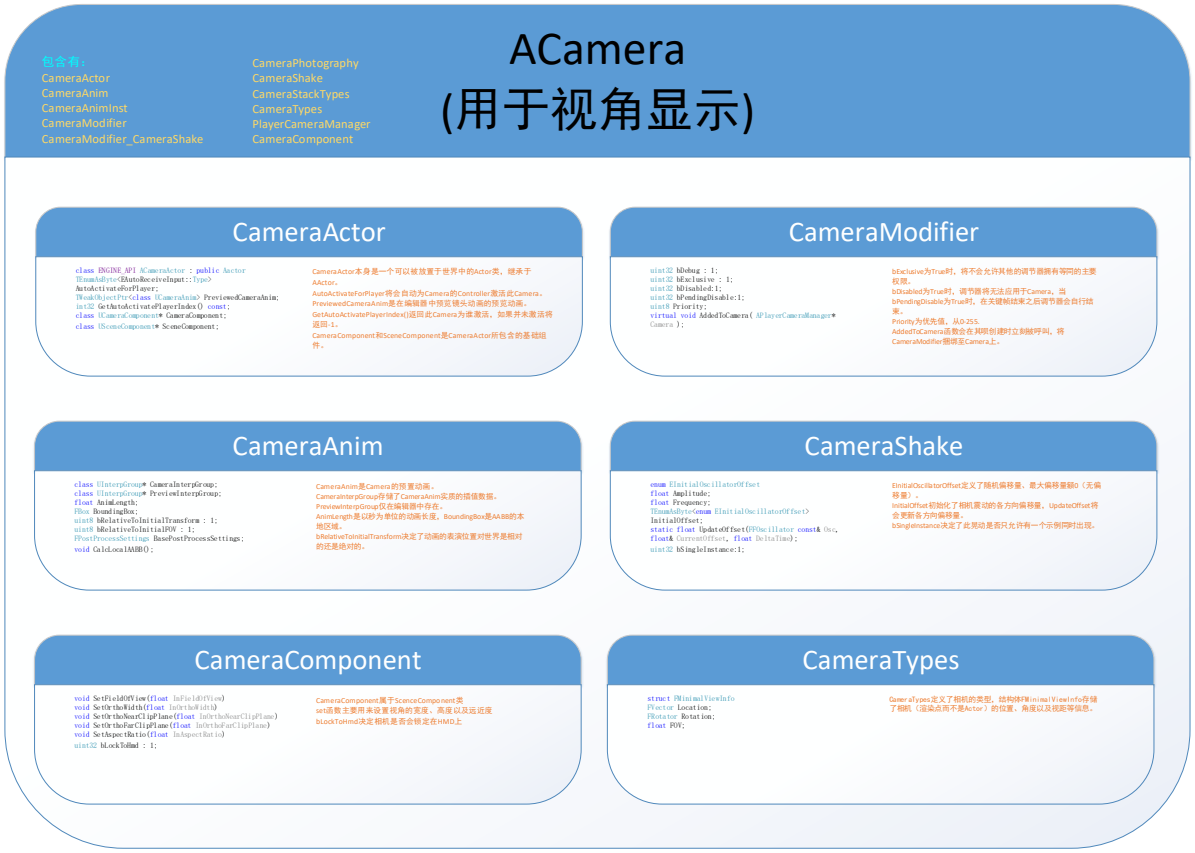

Fig 6.2 *Pawn* class in *Unreal Engine 4*