Introduce to Our Code

This document aims to let all team members quickly familiar with the code in the prototype and start developing upon it. It will focus on how each main feature be achieved through the code.

Table of Contents

[How to go from the title screen to the game? 1](#_Toc146892092)

[What are the game states and how do they affect the game? 1](#_Toc146892093)

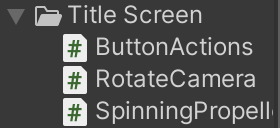
[How can the player walk/run/jump? 2](#_Toc146892094)

[How can the player interact with objects? 4](#_Toc146892095)

[How do the Pause Menu, Task Menu, and Guide Menu show up in the UI when the player presses the according key? 6](#_Toc146892096)

# How to go from the title screen to the game?

The title screen is achieved by a separate scene called “Title”. It has a button “Player” on its “Canvas”. The button is linked to the function “StartGame()” in the script “ButtonAction.cs” in the “Title Screen” folder. When the button is clicked, the function “StartGame()” executes, and it will load the scene “Barrier 1 & 2” to get the player to the main game.



# What are the game states and how do they affect the game?

There are currently four game states stored in “GameState.cs”,

* Running # Player can move. Time passes.
* Paused # Pass cannot move. Time stops.
* Over # Pass cannot move. Time stops.
* LeavingMainScene # Player cannot move. Time passes.

They play important roles in managing the progress of the game and what the player can do.

The game starts with a “Pause” state with the guide menu popping up (Fig1). The player cannot use “wsad” to move the character and the time pauses. After clicking the “X” icon. The game go into “Running” state. In this state, the player can do any movement. When the player fails or finishes the game. The game state becomes “Over”. Like a “Pause” state, the game stops and the player cannot move. The only thing they can do is restart the game or go back to the title screen. When the player enters a special scene (scene here means differently to Unity Scene, the baggage organiser scene and the main scene, the airport, can be in the same Unity Scene), like the “Baggage Organisation Scene”, they should not control the character anymore. Therefore, in “LeavingMainScene” state, the movement of the player is locked, they can only do what the special scene allows them to do, i.e., in Fig2, they can only move items on the three bags with their mouse. However, the time will still pass since the game is still in progress.

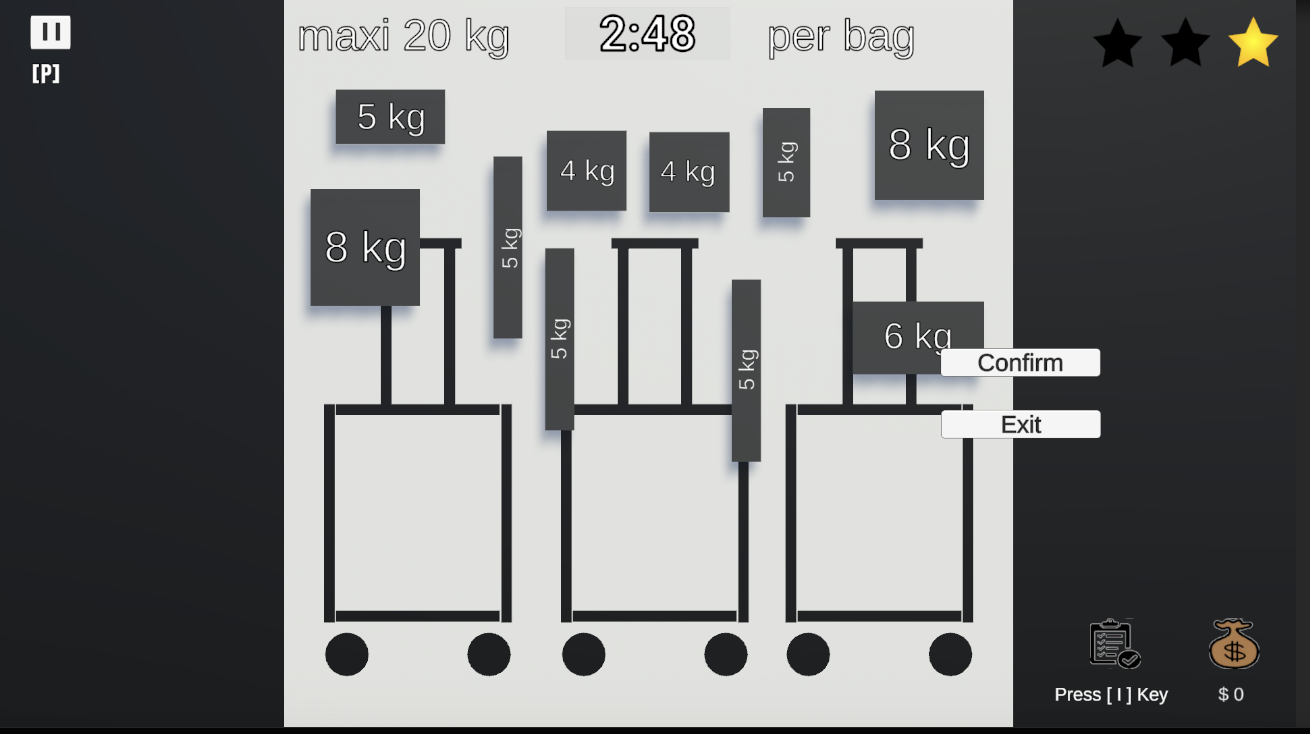


Fig1 Fig2

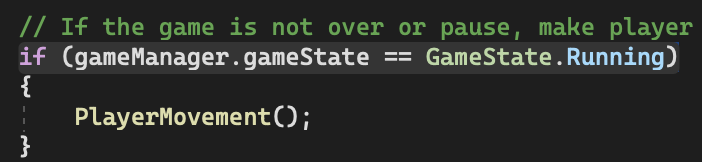
# How can the player walk/run/jump?

The script “PlayerController.cs” manages all the movements of the player.

The “Start” function is “PlayerController.cs” initialises all animation, rigid body, sound effect, stamina, and mouse sensitivity of the player.

The “Update” function the its helper function “PlayerMovement” work together to react to any player input keys and turn them into player movements.

(1)



The “Update” function calls “PlayerMovement” only when the statement is running. In other words, react to the player’s key inputs only when the player is allowed to do any movement.

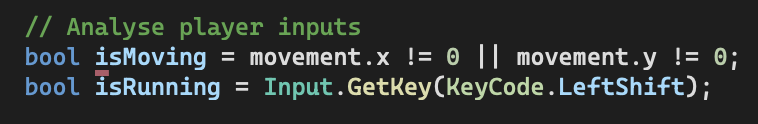
(2)

A black background with white text

Description automatically generated

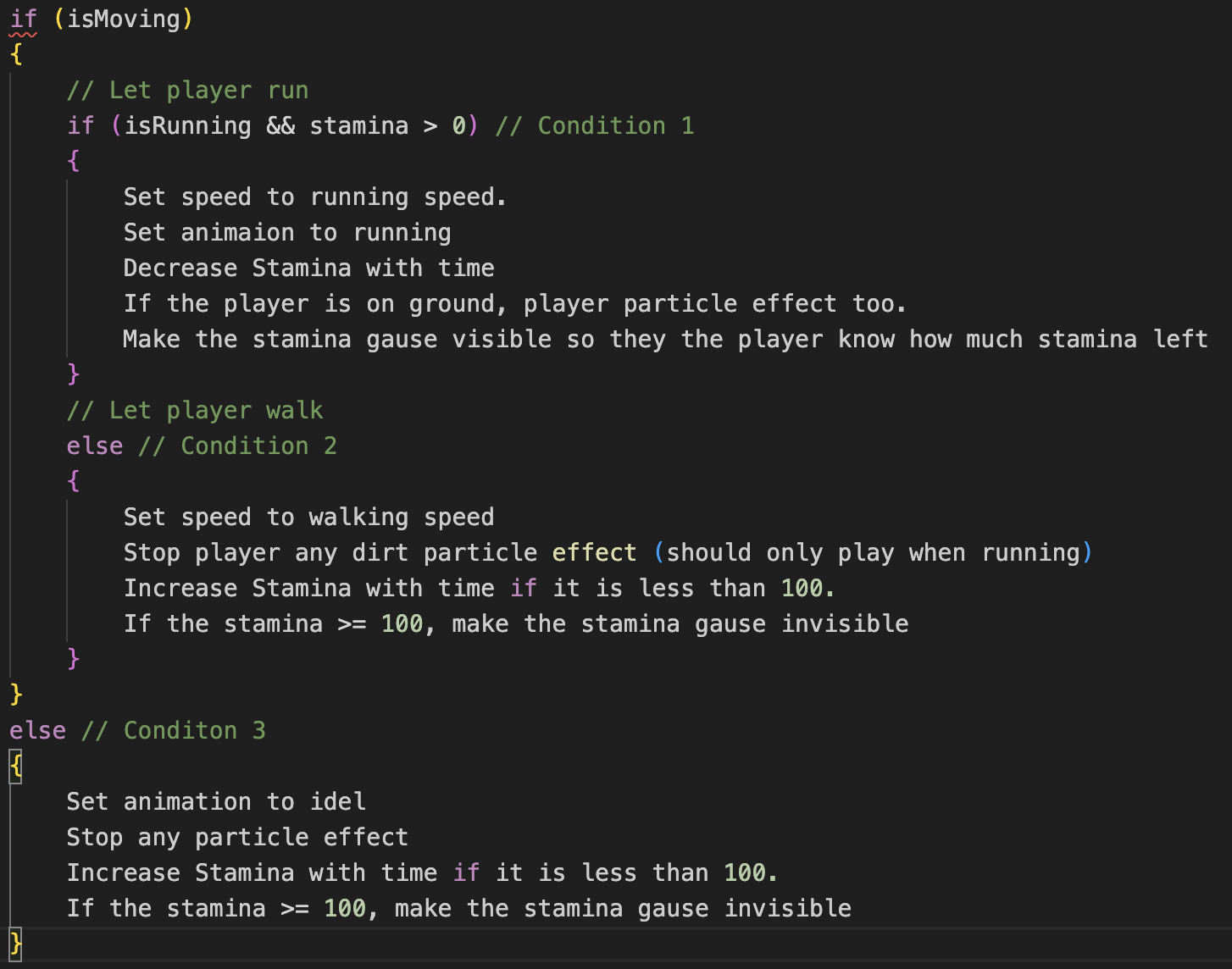
Capture all wasd key inputs and turn them the “movement” variable into speed so that the program how to move the character.

(3)



Adjust speed according to what key inputs the player used. If the player does not press any of the wsad, do not need to move the character. If the player press wasd combined with a SHIFT, the program will know to move the player faster in the following code.

(4) The program then enters an if statement. The following is the pseudo-code.



For example, if the player presses w and “shift” together. “isMoving” and “isRunning” are both set to be true in (3). Then the code enters Condition 1. It set the speed of the player stored in the “newSpeed” variable. This variable will control how much will apply to the player to move. Then set the animation to running. The stamina is decreased with time to prevent from running forever. If the player is on the ground, also enable the dirt effect. Make the stamina gauge visible in the UI so that the player can know how much stamina is left. Similarly, to conditions 2 and 3.

(5)

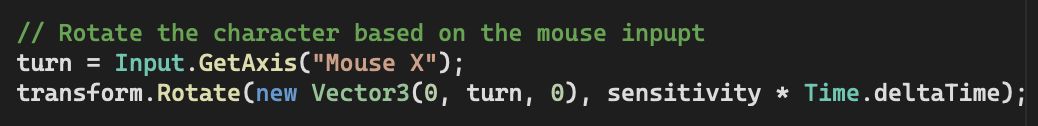
The first line sends the stamina value stored in the “stamina” variable in “PlayerController.cs” to the UI. So that the stamina gauge in the UI can sync with the “stamina” variable

A screen shot of text

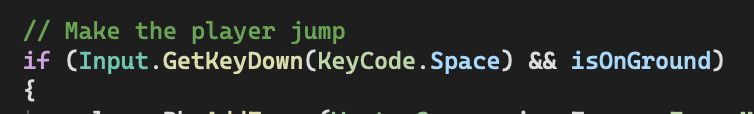
Description automatically generated

The second and the third lines apply force to the player according to the speed set before. If the player is running, the speed is large. If the player is walking, the speed is smaller.

(6) This code allows the player to turn around with the movement of the mouse



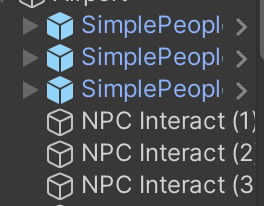
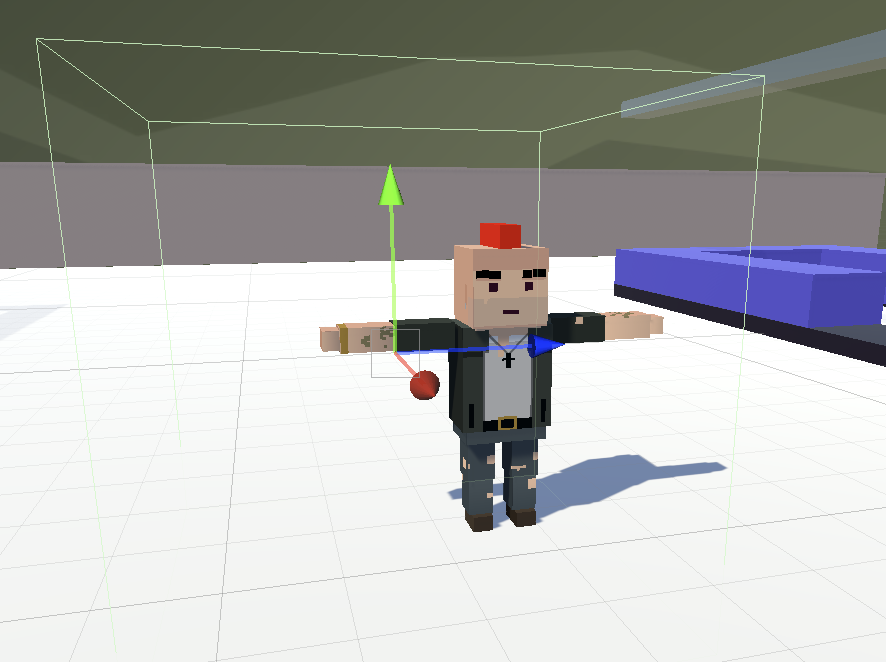
(7) Lastly, if a “Space” is pressed in this frame, make the player jump



# How can the player interact with objects?

The scripts control how the player interactions are stored in every interactable objects instead of in “PlayerController.cs”.

Using this person as an example.



NPC box collider Invisible object box collider

A screenshot of a computer

Description automatically generated

In the game, you may only see one object, but there are actually two objects. One is the NPC itself, the other is an invisible object. There are two box colliders. One is in the NPC object to prevent the player from passing through the body of the NPC (防止穿过NPC的身体). The other is to detect if the player is close to the NPC.

When the player moves closer to this guy, the player will collide with the outer box collider of the “NPC Interact” object. The script “NPCInteract.cs” is actually attached to the “NPC Interact” object instead of the “SimplePeople”.

Inside “NPCInteract.cs”, you will see a “Update”, “OnTriggerEnter” and “OnTriggerExit”.

What “OnTriggerEnter” does is that when the player moves closer to the NPC, it will silently collide with the object “NPC Interact” which is a lot bigger than the size of the NPC. Therefore, the system can know the player is closer it this NPC. In “OnTriggerEnter”, it makes the little “[F] Interact” icon visible in the game UI (Fig1) so that the player can know an interactable object is close to them.

A screenshot of a video game

Description automatically generated

Fig1

Similarly, “OnTriggerExit” make the “[F] Interact” icon invisible when the player no longer collides with this NPC, i.e., the player moves away from the NPC.

“Update” handles the actual interaction. In this case, if the player close to the NPC and press F, a dialogue box will pop up showing the conversation and disappear after 4 seconds.

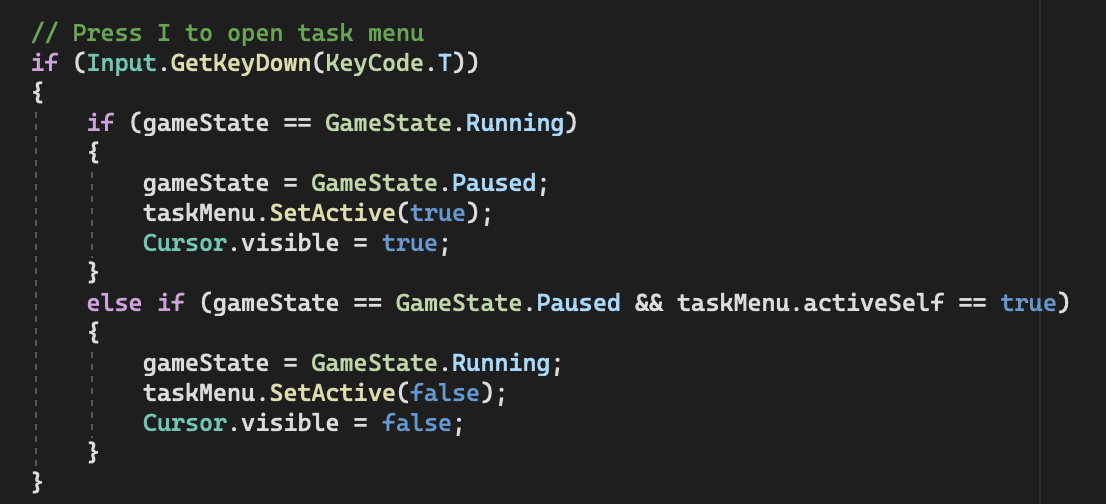
A screen shot of a computer code

Description automatically generated



# How do the Pause Menu, Task Menu, and Guide Menu show up in the UI when the player presses the according key?

Whether this menu is shown up is managed in “GameManager.cs”. In the “Update” function there are some if statement



Using the above if statement for example. If the player presses T and the game state is “Running”, it should first pause the game. Then make the cursor visible (when the game is running the cursor is invisible).

If the player presses T and the game state is “Pause” and the “Task Menu” is active, it means the player has opened the “Task Menu”. Therefore, pressing T will close it and turn the game back to “Running” State.

All the variables “taskMenu”, “guideMenu”, and “pauseMenu” in “GameManager.cs” are linked to the actual object in the Canvas.