Xiang Li
b9047186
Dr. Rich Davison
CSC2034
20 May 2021

The Mellow Yellow Fellow

The Mellow Yellow Fellow game is the main project that I have been worked on through the entire Computer Games Engineering topic. As it was an unfinished replica of an existing game, Pac-Man, I have implemented multiple new features by using Unity engine in order to make this game running properly. For example, ranging from moving player character Fellow in the maze to earn points by eating pellets, from enabling in-game ghosts to chase and eat player to changing the gameplay temporarily by using additional power-up items. Hence, a complete and executable game was redeveloped successfully. And meanwhile, the most important thing that I have learned was how to use one of the popular game engine Unity to create games and how to design and develop each feature of games exactly.

URL

https://nucode.ncl.ac.uk/scomp/student-portfolios/b9047186-portfolio/csc2034/the-mellow-yellow-fellow/-/commit/b95a457a800e7d6362de9ccf22a40a8637eacbc9

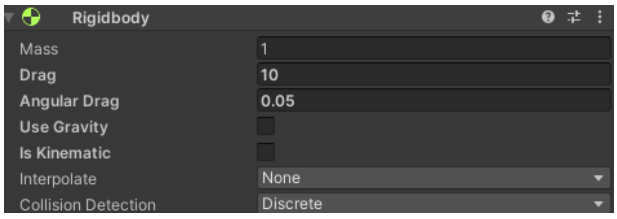
https://nucode.ncl.ac.uk/scomp/student-portfolios/b9047186-portfolio/csc2034/the-mellow-yellow-fellow

Player character object

In the game, the main purpose of the project is that letting gamer to be able to control an on-screen character which has multiple lives and try to pick up all of the pellets dotted around the maze to earn points. Additionally, character could be powerful because of some special in-game items. This is the reason why I have created a new C# script component called Fellow, which can manage its movement and behaviour. Base on the existing physical prefab, the first priority work was making Fellow to move properly in the maze.

MOVEMENT FEATURE

There were two ways that I have been thought about moving player character. One is adding GameObject's Transform component, which can teleport the character immediately from one position to another position, once player press the key (Davison, R. 2021). The other one is adding RigidBody component that can move the character physically under the influence of Newtonian physics (Unity Documentation, 2020). However, when I have tried Transform method, the drawback was really obvious, as Fellow was moving straight through maze wall. I have realised that



<Figure 1: Rigidbody component >

Transform component is the main key to define a position in Unity. Even though I have saved

Fellow's starting position and modified the next position as soon as player press the key, it does not represent that the object is interacting with anything with the game world, including wall (Birch, C. 2021). In order to avoid this problem, I need to make the RigidBody method to hook Fellow object up to the physics system of Unity. Therefore, I have modified velocity instead of modifying position directly. Specifically, FixedUpdate() have instructed the physics engine of player's preferences of the character's moving distance and let the character perform the rest of its actions based on the speed. As the result, the character successfully bumps into the walls rather than going straight through them, and stops properly when the key does not be pressed.

EATING PELLETS AND GAINING POINTS

Since Fellow is able to move properly, the second important feature that I have done was let Fellow to eat pellets and get points. Once all the pellets in the maze be eaten, game is over and player wins with a score. This is the main purpose of playing this game during game development. Hence, the function of making pellets to disappear immediately was quite easy to implement. As pellets were already set up as a prefab and trigger, I have added new C# script component inside the pellet prefab to check whether a RigidBody has a Collider component or not. When two object collides, OnTriggerEnter() method lets Unity to decide the trigger object to disappear (Unity Documentation,

```
private void OnTriggerEnter(Collider other)
{
    gameObject.SetActive(false);
}
```

<Code 1: OnTriggerEnter method for pellets>

2020). With gameObject.SetActive(false) function, trigger objects' - pellets - boolean state turned into false and disappear. Thus, this makes the player recognise that Fellow eats the pellets.

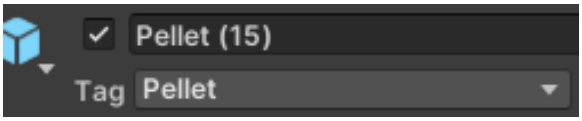
Since pellets were eaten by Fellow properly, gaining points from the eaten pellets was really clear for me. By adding 'score' and 'pointsPerPellet' variable on the Unity editor through Fellow C# script, I have set each pellet as 100 points and tracked how many scores player gets when Fellow ate pellets. Therefore, when I have begun to design Game UI, I can easily import 'score' variable

<Figure 2: Points per pellet variable>

directly from the script and turn those variables into text which can be displayed on the UI. Nevertheless, another problem has been occurred. All the special in-game items such as Powerup were also be counted as score.

SPECIAL ITEM (POWERUP)

Although the pellets seemed to run pretty well at the moment, not all the in-game items were considered as pellets. Powerup are one of the special items and also represent a new game feature which can make Fellow powerful temporarily and to eat the ghosts later on. This idea may change the gameplay in a different way in order to make the game more fun. How to distinguish the object whether it is pellets or Powerups is one of new knowledge I have also learned from Unity. To fix this problem, I have added an identifier to the pellet and Powerup GameObject by using Tag (Unity



<Figure 3: Pellet tag>

Documentation.2020). Each tag corresponded to each GameObject. Then, calling the Unity method CompareTag() will return a boolean value to decide whether the object has the same tag as the parameter. With the same tag of 'Pellet', special game items have never be counted as score.

```
if (other.gameObject.CompareTag("Pellet"))
{
```

<Code 2: CompareTag() method for finding pellet tag>

Following the same OnTriggerEnter() method like pellets, Powerup item will also be effected and make Fellow to turn into powerful mode. How long the Powerups can remain were all depending on new variable that I modified on Unity editor, 'powerupDuration' and 'powerupTime'. Since I set the

<Figure 4: Powerup Duration variable>

duration time as 10 seconds and minus the deltaTime by using Unity engine Time class, program can know how long the time has passed in seconds and what should do once the time finished. Having the power time track function helped me manage the timing of ghost mode more easily later on.

LIFE SYSTEM, PLAYER DEATH AND RESET

In the game, player character object always has multiple lives to control player death. Game will over until all the lives of Fellow were gone. Hence, I have developed a life counter feature to track the lives of Fellow(gamespluskjames, 2015). By adding new variables called 'lifeCounter' and

<Figure 5: Life variable>

'startingLives' in the script, the total lives of Fellow were set as 3 on the Unity editor. Just like variable 'score', 'lifeCounter' helps me display the life numbers on the UI easily.

However, every time when Fellow was caught by ghosts, it should be return to its starting position instead of disappearing forever. Moreover, if Fellow ate the Powerup, it cannot be eaten by Ghost. Those different scenarios indicated the importance of life feature in the game development. To solve this problem, I have saved Fellow's start position into a variable called "dest" and designed a simple

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Ghost"))
    {
        if (PowerupActive())
        {
            gameObject.SetActive(true);

        }
        else
        {
            if(lifeCounter != 0)
            {
                lifeCounter--;
                Debug.Log("You have " + lifeCounter +" lives");
                transform.position = dest;
            }
        }
    }
}
```

<Code 3: OnCollisionEnter method for controlling player death>

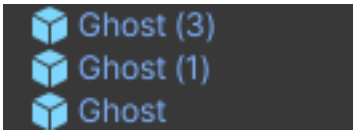
if-else statement in the OnCollisionEnter method. The function helps me to estimate whether Fellow was been power-up or not. If not, Fellow will be eaten and back to the original position 'dest'. As soon as player ran out of lives, Fellow disappeared and game was over.

EVALUATION

Overall, a simple and comprehensive player character object was completed successfully. Looking back the program that I developed for the Fellow, the importance of gaming development is understanding the purpose of creating each features. The project matched perfectly with every single requirement in the task. I felt like this virtual object become more vivid and brought lots of fun for the gamers. However, there were still existing some problems and drawbacks about my character currently. For example, when the Fellow turns around in the corner of the maze, it can move and rotate freely without getting interrupted by wall. Actually, it should be move one by one base on where pellets locates. Moreover, in the future developing, maybe I will add some animation for my Fellow. Just like the Pac-man can open mouth in the real Pac-man game. I think it will make my game more interesting on visualising.

Ghosts object

A completed game not only need to have main player character but also need to have some enemies if necessary. During game development, it is important to add enemy system in order to make game more challenging and diversified. This is why I created three new C# script components called Ghost, Ghost(1) and Ghost(3) to control my ghosts base on an existing ghost prefab. However,

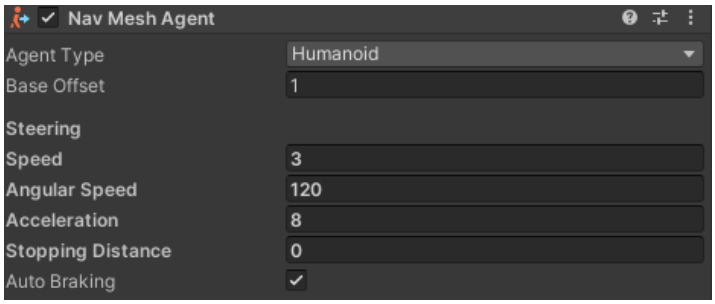


<Figure 6: three different ghosts>

game will make player feel bored if they are able to predict enemies' movement. To avoid this problem, I have searched about Pac-Man ghost behaviour. The main differences between each ghosts are moving route and speed.

MOVEMENT AND BEHAVIOUR

In order to implement unpredictable movement for ghosts, I have added a NavMeshAgent component inside each ghost prefab. In my game scene, the maze was filled by navigation mesh, which indicate the area that AI objects can walk on. Therefore, Unity's NavMeshAgent component will automatically move through the world to the given location (Unity Documentation. 2020). So to get my ghost objects moving, I have developed a PickRandomPosition method to pick a position



<Figure 7: NavMeshAgent component>

within 8 units of the ghost's current position. Then SamplePosition method will return a bool to

decide whether the ghost find a position or not. Once ghost found the given position, it moved to that location and reposition for next random destination (Davison, R. 2021). Hence, all my ghost

```
Vector3 PickRandomPosition()
{
    Vector3 destination = transform.position;
    Vector2 randomDirection = UnityEngine.Random.insideUnitCircle * 8.0f;
    destination.x += randomDirection.x;
    destination.z += randomDirection.y;

    NavMeshHit navHit;
    NavMesh.SamplePosition(destination, out navHit, 8.0f, NavMesh.AllAreas);
```

<Code 4: PickRandomPosition() method for finding position>

started to move unpredictably. However, let the ghost ran randomly around the maze cannot interrupt player object to gain points. So I created another simple CanSeePlayer() method to control ghosts hunting player (Davison, R. 2021). With Raycast method, it helped to detect whether there is player character near its position. If there is, the ghosts will keep following the object which has Fellow tag until kill the player.

```
bool CanSeePlayer()
{
    Vector3 rayPos = transform.position;
    Vector3 rayDir = (player.transform.position - rayPos).normalized;

    RaycastHit info;
    if (Physics.Raycast(rayPos, rayDir, out info))
    {
        if (info.transform.CompareTag("Fellow"))
        {
            return true;
        }
    }
    return false;
```

<Code 5: CanSeePlayer() function for hunting player>

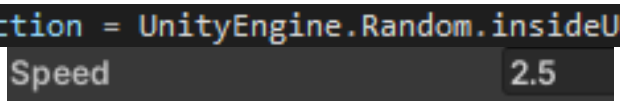
Since the enemy system was behaving more and more smart, designing different behaviour was the second important feature for my ghosts (Birch, C. 2010). Base on Blinky's behaviour as 'chaser' in original Pac-man game, I set Ghost to chase Fellow all the time around the whole map because it had the most wide range of detected unit, 8 units. And Ghost(1) was followed by Pinky in Pac-man

```
Vector2 randomDirection = UnityEngine.Random.insideUnitCircle * 8.0f;
```

<Code 6: units changes for Ghost>

as 'ambusher', which means it can attack Fellow suddenly around the corner. This was due to its detected unit as 4 and the speed of 2.5f. Finally, the Ghost(3) was same as Clyde in Pac-man, it was

```
Vector2 randomDirection = UnityEngine.Random.insideUnitCircle * 4.0f;
```



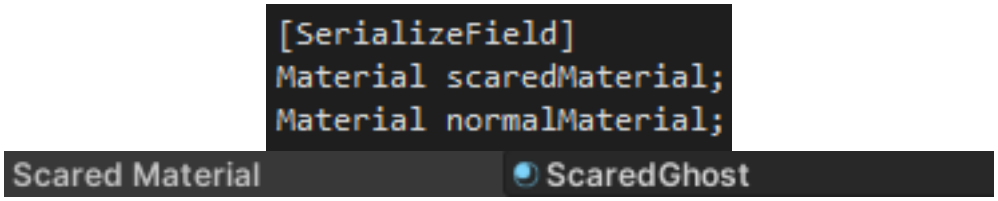
<Code 7: unit and speed changes for Ghost(1)>

a dumb ghost because I didn't control it to chase player character, it just moved randomly in the maze. Therefore, with three different personality ghost, the whole enemy system became more challenging for player to predict and made the game more fun.

GHOST MODE

As the game has special items, when player ate the Powerup, it will bring some positive effect to the gameplay. The ghosts will run away and Fellow can eat them in order to reset those ghosts.

Therefore, the only way to change the all the ghosts' behaviour is to change their action mode (Davison, R. 2021). In Unity, a GameObject's appearance can be changed by its material (Unity Documentation. 2020). So I designed two material for ghosts base on existing prefab, one is 'normalmMaterial' which means ghosts can chase player as usual, and another one is 'scaredMaterial' which indicates player can eat ghosts as they will turn into blue colour because I



<Figure 8: Material variable and component>

provided an alternative material in this mode. Hence, in the scare mode, I used same NavMeshAgent component to make ghost determine a position away from the player character and made them stop at each corner of the maze. Then with a new bool variable called 'hiding' in the if-

```
bool hiding = false;
```

<Code 8: hiding variable for controlling ghost>

else statement, I let the ghosts to check whether player became Powerup or not. If so, 'hiding' turn into true and ghosts run away. Otherwise, ghosts keep chasing player.

EVALUATION

As a whole, the entire enemy system Ghost was completed successfully. It totally help the game to become more challenging. Ghosts not only can hunt player in different way but also contain multiple modes for change the gameplay. Every features that I developed matched what the task required except the movement of the ghosts still little bit buggy. For example, sometimes ghost may collide each other in the maze and block the path. Or some ghosts started to move in one way once they used the teleport tunnel. What I think this is the layer problem in Unity. Later on in the future game developing, I may choose another method for designing ghosts' movement. Instead of using NavMeshAgent component, Node is also useful way to do the implementation (The Weekly Coder. 2017). Node can be consider as a reference point, which means I can set different path for each ghost from node to node. In that case, ghosts will move like a fake AI.

User Interface(UI)

A proper game must have user interface for players to start and stop the game, and also tell players whether they win or lose the game. Moreover, UI can show some extra information like score, character life and level etc (Unity Technologies. 2020). As I described before, the variables that I created before such as score or lifeCounter can all be displayed on the UI. Therefore, I added five new UI panels with different script components in the game scene: MainMenuUI, HighScoreUI, GameUI, WinUI and GameOverUI.

In MainMenuUI, I set the 'Space' key to start the game, which UI turned into real-time GameUI immediately and every score, eaten pellets, life and level will be clearly shew on the board. As soon as player gained the point or lost life, the data will changed automatically at the same time. I also set the 'Enter' key to display HighScoreUI, which players can check their previous highest scores. When player lost every lives in the game, GameOverUI that I modified will come out and

<Figure 9: User interface control panel>

ask player for restarting again by pressing 'R'. Or when player ate all the pellets, WinUI will displayed "Level complete" and let player to start next new Level. All of those UI panels are controlled by new variable in the script called 'gameMode' (Davison, R. 2021), each gameMode helps displayed the UI through SetActive() method.

```
enum GameMode
{
    InGame,
    MainMenu,
    HighScores,
}
```

<Code 9: Gamemode for controlling UI>

EVALUATION

Overall, all the basic user interface for my game are designed successfully. Player can easily access to the game and check their game information. However, there is one feature in the UI which didn't match task requirement. Although the high score text can be displayed properly, but I didn't update it for saving every new high score. I may should add user database into the game which can help me store game score for everyone. Furthermore, I didn't change the style of the existing UI. Later on in the future development, I should try add strong vision style of UI to attract players' attention before starting the game.

Addition feature

Since all the basic game feature and setting were completed, I tried add some more new features in the game to make it more comprehensive.

TELEPORT TUNNEL

As the original Pac-man game has teleport function, I also tried to created the teleport tunnel at each side of the hole. By adding two new trigger cube prefabs in each side, I added new C# script component for them called Teleporter. Each cube also named differently because it is Teleporter to transfer the object, just like elevator (Develop APVN. 2018). Therefore, whatever the object is,

<Figure 10: Teleport cubes>

there must one cube call ‘Teleporter’ another one called ‘Destination’. Then the new variable ‘Target’ help to teleport the in-game objects.

<Figure 11: Target variable>

COUNTDOWN ANIMATION

Before the game start, player should wait until pressing the start button. In order to re-spawn the player character and ghosts, I developed a countdown animation before start game to make it more

<Figure 12: Countdown time variable>

professional. With new variable called ‘countdownTime’ (Turbo Makes Games. 2019), I can modify how long I need for countdown in the IEnumerator.

```
IEnumerator CountdownToStart()
{
    while (countdownTime > 0)
    {
        countdownDisplay.text = countdownTime.ToString();

        yield return new WaitForSeconds(1f);

        countdownTime--;
    }
```

<Code 10: IEnumerator function for countdown>

SOUND

The sounds make the game more vivid and real. Hence, I added the Audio Source component in the main game panel as background music. And also I added AudioClip component for players’ winning and death.

<Figure 13: Sound clips>

Conclusion

Personally, I think the whole Mellow Yellow Fellow game ran successfully. Because of corporation with each part of the features, the player character Fellow can be controlled smoothly and earn points around the maze. Meanwhile, three ghosts try to hunt down player in different ways in order to make the game more challenging. With some of the special item like Powerup, gamers are able to change the gameplay temporarily and make it more fun. Last but not the least, the designing of user interface completed this game in a professional way.

Although I have implemented almost every fundamental parts of the game, some extended parts I still didn’t meet well with the task description. For example, lots of games include multiple level, especially the original Pac-man game. As soon as one level complete, the game difficulty will

increase, which will be affected by ghosts' speed or the number of the ghosts. The level system is one of the future feature that I should develop later on. It will make game more diversified without getting bored quickly. Moreover, the visual style of my game is also quite simple. Later on, I should try to add more animated cutscenes for level system and change the map style in another shape. However, looking through my project, it really helps me learn lots of new things about using Unity engine. Meanwhile, I felt more confident about game development. Even though it was my first time to create game, it brought me lots of fun and I want to learn more about Computer Games Engineering. In the future, I will try to add more feature in the Mellow Yellow Fellow and change it to online game, such as multiple player mode, online pvp mode. More importantly, with the experience of Unity engine, I may implement the same game design on different game engine like Unreal.

Reference

1. Birch, C.(2010) *Understanding Pac-Man Ghost Behaviour.* GameInternals. Available at: https://gameinternals.com/understanding-pac-man-ghost-behavior [Last accessed on 3rd May 2021]
2. Camonu, I.(2020) *Collider and Trigger In Unity3D,* coinBlack. Available at: https://www.codinblack.com/colliders-and-triggers-in-unity3d/#:~:text=The%20purpose%20of%20triggers%20is%20to%20trigger%20events,instead.%20There%20are%20several%20collider%20types%20in%20Unity3D. [Last accessed on 27th April 2021]
3. Davison, R.(2021) *Making Games In Unity.* Newcastle University. Available at: https://ncl.instructure.com/courses/24653/files/3856448?wrap=1 [Last accessed on 20th May 2021]
4. Develop APVN. (2018) *How to create simple teleport UNITY C#.* 30 January, Available at: https://www.youtube.com/watch?v=D1gOtOtwPFA [Last accessed on 3rd May 2021]
5. gamespluskjames. (2015) *Lives System & Getting Extra Lives - Unity 2D Platformer Tutorial - Part 18.* 4 April, Available at: https://www.youtube.com/watch?v=zr7ys5lFakA&t=817s [Last accessed on 6th May 2021]
6. Turbo Makes Games. (2019) *How To Make a Countdown Timer in Unity.* 4 October, Available at: https://www.youtube.com/watch?v=ulxXGht5D2U&t=598s [Last accessed on 6th May 2021]
7. The Weekly Coder. (2017) *How to make a game like Pac-Man in Unity 5 - Part 3 - Nodes 4k.* *2*4 January, Available at: https://www.youtube.com/watch?v=xv1Z9MpnckM&list=PLiRrp7UEG13a4DmYuNWHSoqLqNukEm9ua&index=3 [Last accessed on 10th May 2021]
8. Unity Documentation.(2020) *Rigidbody,* Scripting API. Available at: https://docs.unity3d.com/ScriptReference/Rigidbody.html [Last accessed on 1st May 2021]
9. Unity Documentation.(2020) *Collider.OnTriggerEnter(),* Scripting API. Available at: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html [Last accessed on 26th April 2021]
10. Unity Documentation.(2020) *Tags,* Manual. Available at: https://docs.unity3d.com/Manual/Tags.html [Last accessed on 29h April 2021]
11. Unity Documentation.(2020) *NavMeshAgent,* Scripting API. Available at: https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html [Last accessed on 5th May 2021]
12. Unity Documentation.(2020) *Physics.Raycast,* Scripting API. Available at: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Last accessed on 5th May 2021]
13. Unity Technologies. (2020) *Working with UI in Unity*, Learn Unity. Available at: https://learn.unity.com/tutorial/working-with-ui-in-unity# [Last accessed on 7th May 2021]