MPIE Summative Assessment Report

Background

The experience created is a physics-based platformer that uses a ball in which the player controls to manoeuvre through the environment. As the player progresses through the scene, they will collect coins and power-ups, enabling the player to progress further. The goal of the game is to reach the end in the fewest number of deaths possible, while also collecting coins along the way.

Figure 1. View of in game environment

This environment and user experience was chosen because a game in the platformer genre allows you to fully showcase several technical implementations to create a cohesive and enjoyable game, from physics and collision detection, cameras and transforms, sound and heads-up displays (HUD). Specific elements, such as parts of level design and aesthetics, were inspired by other experiences, like that of Super Mario. These elements were intertwined with original features, like the ball rolling mechanic, to create a unique physics-based platformer. The purpose of the game is to offer a fun and challenging experience, where the player can test their gaming skills to reach the end of the scene. The game is targeted primarily for young people but can be played by anyone regardless of their skill level. However, players with a lesser skill level may find some elements more difficult than others. The game can be experienced anywhere, if the player has access to a desktop computer.

Description of Experience

Figure 2. View of the starting platform

When the game begins, the user will be shown the ball, along with a text prompt showing how to move the ball. In the bottom left, the number of deaths and coins is shown. The player must progress forward, navigating through the various obstacles. As the player runs through coins, the count will increase. It is worth noting that the user can press ‘Escape’ at any time to bring up the pause menu. The menu will pause the game and allow the user to respawn the ball or exit the application.

Figure 3. View of the first checkpoint

When the user reaches the first checkpoint, they can interact with it by moving next to it. The text, “Checkpoint 1 set,” will be displayed, meaning that when the player dies, the ball will respawn in this location. The player has the option to go left or right, but as the player cannot currently jump, the path on the left is inaccessible. The user must go right onto the moving platform and collect the power-up.

Figure 4. View of the jump power up

Like coins, the power-up can be collected by running through it. The power-up will be indicated by the image at the bottom right of the screen, as well as a tutorial on how to use the power-up. In this case, the user can press ‘Space’ to jump. With the power-up collected, the user can progress through the inaccessible path.

Figure 5. View of the second checkpoint

Once the user progresses through the next set of obstacles, they will reach a checkpoint, where again the user is met with two paths, once of which being inaccessible. The user can jump through the moving platforms and collect the speed power-up. This allows the player to move quicker when ‘Left Shift’ is pressed. With this speed increase, the user can achieve the previously impossible jump.

Figure 6. View of the third checkpoint

When the player makes the jump, they will be greeted with another checkpoint, with a series of vertical moving platforms. With the power-ups they have unlocked, they need to navigate through this section to reach the next checkpoint.

Figure 7. View of the path after the fourth checkpoint

When the user makes it past the moving platforms, the fourth checkpoint is reached. This next obstacle requires patience and precision, forcing the player to navigate through a narrow path.

Figure 8. View of a jump after the final checkpoint

Once the final checkpoint is reached, the user can see the finishing flag. However, to reach the platform, the user requires the third and final power-up, the jump boost. This increases the jump boost, allowing the user to make the final jump.

Figure 9. View showing the final jump

Once the user acquires the final power up, and in combination with the previous speed power up, they will be able to make the final jump. When the user reaches the platform and

collides with the finishing flag, they will be displayed with a final menu, showing them the number of deaths and coins that they finished the game with.

A moderately skilled player should be able to finish the game in around five minutes, although it may take longer for lesser skilled players, due to some difficulty revolving certain tricky jumps. All the controls are conventional to other games and should not be difficult to the typical player to pick up.

Description of Technical Implementation

There are numerous technical implementations that are utilised throughout the project. The core features include:

- Ball movement
- Camera movement
- Triggers
- Platform movement
- Heads-Up Display (HUD)
- Sound

Ball Movement

The most important feature of the game was the movement of the ball. Unlike other Unity experiences, a first-person or third-person controller was not featured in the game, and instead the player controls a ball which rolls through the environment. As a result of this, it was necessary to make a script for the ball's movement.

```
void FixedUpdate()
{
    //Ball movement
    float zMovement = Input.GetAxis("Vertical") * speed * Time.deltaTime;
    float xMovement = Input.GetAxis("Horizontal") * speed * Time.deltaTime;

    rb.AddForce(Camera.main.transform.TransformDirection(new Vector3(xMovement, 0.0f, zMovement)));
```

Figure 10. Ball movement code

At the very core, the ball's movement comes down to three lines of code. Two variables named, 'zMovement' and 'xMovement' are used store the distance the ball should travel. By retrieving the vertical and horizontal axis and multiplying it by the speed and the interval in seconds from the last frame to the current one, the distance is found. This can be then used to add force to the ball's 'Rigidbody', a component which applies physics to an object. This allows the ball to roll based on the user's inputs. These lines of code are found in the 'FixedUpdate' function rather than 'Update' because it is more appropriate when computing physics calculations by providing more predictability when updating the code on a fixed interval, rather than every frame.

```
//Jump
if (Input.GetAxis("Jump") > 0 && isGrounded() && playerManager.hasJumpPowerUp == true) {
    rb.AddForce(new Vector3(0.0f, jumpPower, 0.0f));
}
```

Figure 11. Ball jump code

In addition to the ball's movement, the user can also jump when they acquire the jump power-up in-game. Instead of adding force to the x or z axis, force is added to the y axis to allow the ball to jump when the space key is pressed. However, by simply just checking if the user is pressing the space key, it would allow the player to not jump, but fly. It is important to check if the ball is touching the ground before it jumps, stopping the ball from flying upwards.

```
/**
 * This function is based upon an example from the Unity forum
 Reference
 * Author: aldonaletto
 * Location: https://discussions.unity.com/t/how-do-i-check-if-my-rigidbody-player-is-grounded/33250
 * Accessed 21/01/2025
*/
1 reference
bool isGrounded() {
    return Physics.Raycast(transform.position, -Vector3.up, distToGround + 0.1f);
}
```

Figure 12. Code that checks if the ball is grounded

The 'isGrounded' function returns a Boolean, true or false, checking whether the ball is touching the ground. A ray cast is used to record if a collision is detected with the ground. It directs a ray downwards from the ball's position, and checks whether the ball is within range of the ground. If it is, then the function will return true. This is then used in the previous code, allowing the user to properly jump.

```
//Speed Boost
if (Input.GetKey(KeyCode.LeftShift) && playerManager.hasSpeedPowerUp == true) {
    speed = initialSpeed * 2;
    Camera.main.fieldOfView = 95;
}
else {
    speed = initialSpeed;
    Camera.main.fieldOfView = 80;
}
```

Figure 13. Speed boost code

Lastly, the ball's movement script consists of the ability to boost the ball's movement speed when the power-up is acquired. When the user presses the left shift key, the speed will be multiplied by two. The camera's field of view is also widened to emphasise that the speed boost is active. When the user is no longer pressing the shift key, the ball's movement speed and the camera's point of view is reverted.

Camera Movement

In addition to the ball's movement, it was imperative that a camera movement system was added alongside to provide the player with greater control and usability. A static camera following the ball would not have been appropriate for a game which requires the player to travel in multiple directions. Implementing a moving camera was not an easy process. If the camera was simply attached to the ball game object, the camera would rotate as the ball rotates, resulting in a nauseating experience for the user. Instead, the player object consists of a 'BallCenter' game object which the camera tracks.

```
0 references
void Update() {
    //Transform follows parent object
    t = gameObject.transform;
    t.position = parent.transform.position;
}
```

Figure 14. Follow parent object code

With the script in Figure 14, the ‘BallCenter’ game object will follow the position of the ball and provide a stable point for the camera to track.

```
/**
 * This script is based upon an exaxmple from 'envato tuts+'
 Reference
 * Author: Ian Zamojc
 * Location: https://code.tutsplus.com/unity3d-third-person-cameras--mobile-11230t
 * Accessed: 21/01/2025
*/
5 references
public GameObject target;
1 reference
public float rotateSpeed = 5;
2 references
Vector3 offset;
1 reference
public GameObject menu;
1 reference
public GameObject finishMenu;

0 references
void Start() {
    offset = target.transform.position - transform.position;

    //I added the following lines to lock mouse movement
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    //Modification ends
}

0 references
void LateUpdate() {
    //I added the following if statement to only update camera location when menus are not active
    if (!menu.activeInHierarchy && !finishMenu.activeInHierarchy) {
        //My modifcation ends

        //Updates camera location based on mouse axis
        float horizontal = Input.GetAxis("Mouse X") * rotateSpeed;
        target.transform.Rotate(0, horizontal, 0);
        float desiredAngle = target.transform.eulerAngles.y;
        Quaternion rotation = Quaternion.Euler(0, desiredAngle, 0);
        transform.position = target.transform.position - (rotation * offset);

        transform.LookAt(target.transform);
    }
}
```

Figure 15. Camera movement script

With now a stable position to track, the camera’s movement can be implemented. In the ‘Start’ function, the cursor’s visibility and state is changed so that the cursor can no longer be seen or moved. In ‘LateUpdate,’ the camera’s position based on the mouse’s x axis is then

calculated. When the camera's position is updated, the 'LookAt' function is used to point the camera towards the ball.

Triggers

Triggers are colliders which respond based on whether an object has collided with it. In the game, there are multiple triggers which the ball can interact with, including:

- Respawn trigger
- Checkpoints and finish
- Coins
- Power-ups

In each trigger's script, the 'OnTriggerEnter' function will be used. This function executes when a collision is registered with the trigger.

Respawn Trigger

The respawn trigger is positioned underneath the environment and will respawn the player when the ball collides with it. When the trigger activates, the 'RespawnPlayer' function will be activated in the 'PlayerManager' script.

```
//Respawns player and cancels ball's velocity
1 reference
public void RespawnPlayer() {
    ball.GetComponent<Rigidbody>().velocity = Vector3.zero;
    ball.transform.position = respawnPos;
    deaths++;
}
```

Figure 16. Respawn Player function

The function does three different tasks. Firstly, it cancels the ball's velocity before it respawns, to ensure that the ball is not out of control. The ball's position is then updated by the 'respawnPos' variable, which stores a specified position like the position of the starting platform or one of the five checkpoints. Lastly, the number of deaths is incremented by one.

Checkpoints and Finish

Figure 17. View of the checkpoint trigger

Throughout the environment, there are five separate checkpoint triggers and a finish trigger. The checkpoint trigger is positioned in the centre of a platform. It is big enough to ensure that the user does not miss the trigger when navigating through the environment.

```
0 references
void OnTriggerEnter() {
    //Checks checkpoint and updates respawn position in Player Manager, and displays confirmation text
    if (checkpoint.name == "Checkpoint 1") {
        playerManager.UpdateCheckpoint(1, checkpoint.transform.position - new Vector3(5.0f, 0.0f, 0.0f));
        canvasManager.DisplayCheckpointText("Checkpoint 1 set");
    }
    else if (checkpoint.name == "Checkpoint 2") {
        playerManager.UpdateCheckpoint(2, checkpoint.transform.position - new Vector3(5.0f, 0.0f, 0.0f));
        canvasManager.DisplayCheckpointText("Checkpoint 2 set");
    }
    else if (checkpoint.name == "Checkpoint 3") {
        playerManager.UpdateCheckpoint(3, checkpoint.transform.position - new Vector3(5.0f, 0.0f, 0.0f));
        canvasManager.DisplayCheckpointText("Checkpoint 3 set");
    }
    else if (checkpoint.name == "Checkpoint 4") {
        playerManager.UpdateCheckpoint(4, checkpoint.transform.position - new Vector3(5.0f, 0.0f, 0.0f));
        canvasManager.DisplayCheckpointText("Checkpoint 4 set");
    }
    else if (checkpoint.name == "Checkpoint 5") {
        playerManager.UpdateCheckpoint(5, checkpoint.transform.position - new Vector3(5.0f, 0.0f, 0.0f));
        canvasManager.DisplayCheckpointText("Checkpoint 5 set");
    }
}
```

Figure 18. Checkpoint trigger script

When the trigger is activated, there is an if statement checking which checkpoint is triggered based on the name of the parent object. The ‘UpdateCheckpoint’ function is then called from the ‘PlayerManager’ script to update the ‘respawnPos’ variable. It then offsets the X axis by 5 so that the ball does not respawn inside of the flag. Lastly, ‘CanvasManager’ script is accessed to update the checkpoint text object, showing the user that they have set their checkpoint. When the ball leaves the trigger, the text is cleared. The finish trigger simply displays the finish menu through the ‘CanvasManager’ script.

Coins

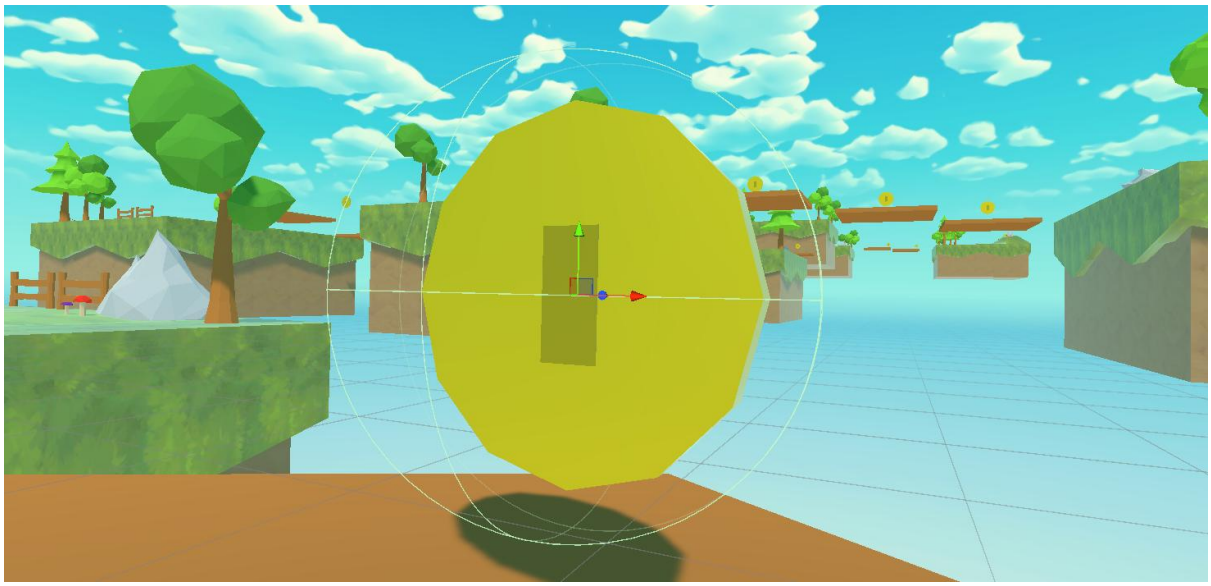

Figure 19. View of the coin trigger

Coins are objects placed throughout the environment with the purpose of being collected by the player. They offer the player with an additional goal to collect all the coins in the environment.

```
0 references
void OnTriggerEnter() {
    //Plays coin sound, deactivates coin and adds to coin count
    coinSound.Play();
    coin.SetActive(false);
    playerManager.coins++;
}
```

The coin's trigger script is simple. When the trigger is entered, a coin sound is played. The coin object is then deactivated so that it cannot be seen or recollected. Lastly, the player manager script is accessed, and the coin count is incremented.

Power-ups

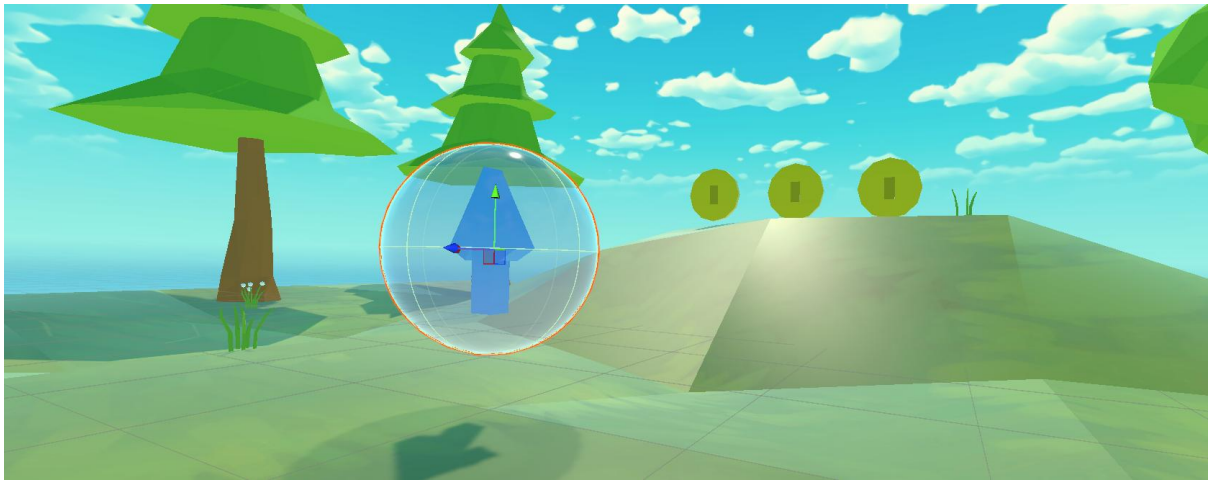

Figure 20. View of the jump power-up trigger

Lasty, there are three power-ups positioned in the environment. These provide the player with power-ups to their movement which allows them to progress through the experience, whether it be the ability to jump, boost their speed or jump higher.

```
0 references
void OnTriggerEnter() {
    //Adds power up
    if (powerUp.name == "Jump Power Up") {
        playerManager.hasJumpPowerUp = true;
    }
    if (powerUp.name == "Speed Power Up") {
        playerManager.hasSpeedPowerUp = true;
    }
    if (powerUp.name == "Jump Boost Power Up") {
        playerManager.hasJumpBoostPowerUp = true;
    }

    a.Play();

    powerUp.SetActive(false);
}
```

Figure 21. Power-up trigger script

Like the checkpoint script, the power-up script checks which power up it is by checking the parent game object's name. Based on the name, it accesses the associated boolean variable in the 'PlayerManager' script and changes the value to true. These variables are checked by if statements in the ball movement script to check if the player has acquired the power-ups. The power-up script then plays a sound when it is collected and then gets deactivated.

Platform Movement

Throughout the environment, there are a series of moving platforms which the user must navigate through. The platforms either move horizontally or vertically, and they provide the game with more difficult jumps to scale the difficulty as the game progresses.

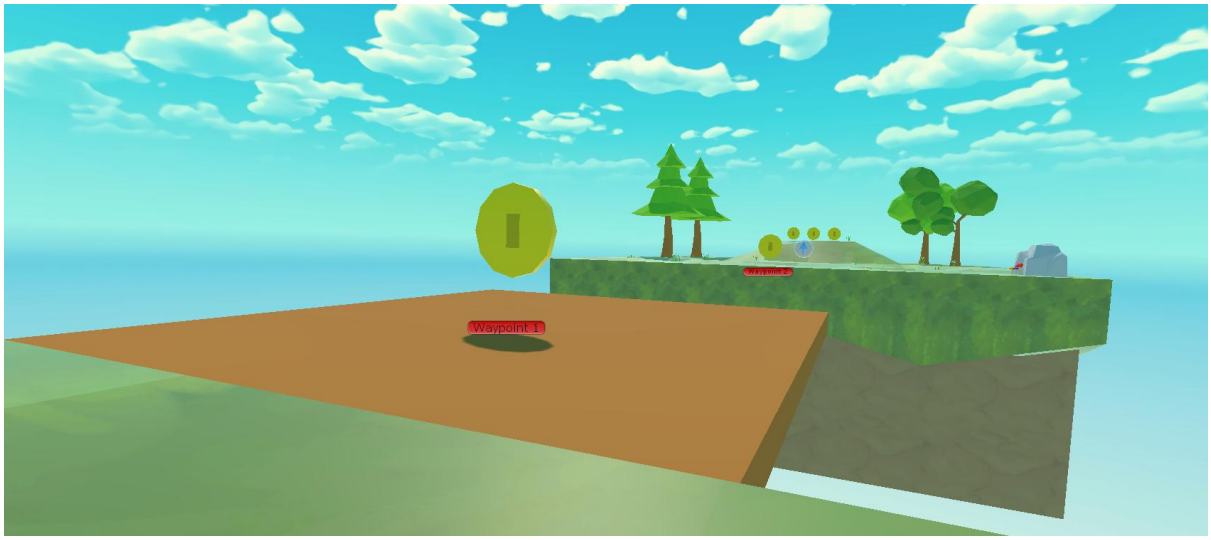

Figure 22. View of a moving platform

As seen in Figure 22, the platform consists of two waypoints. The first waypoint is where the platform is initially positioned, whereas the second waypoint is where the platform is moving towards. When the platform reaches the second waypoint, it changes direction.

```
0 references
void FixedUpdate()
{
    //Moves platform towards first waypoint
    if (isForward) {
        t.position = Vector3.MoveTowards(t.position, waypoint2.position, speed * Time.deltaTime);
    }
    //Moves platform towards second waypoint
    else {
        t.position = Vector3.MoveTowards(t.position, waypoint1.position, speed * Time.deltaTime);
    }

    //Changes platform direction
    if (t.position == waypoint1.position) {
        isForward = true;
    }
    else if (t.position == waypoint2.position) {
        isForward = false;
    }
}
```

Figure 23. Moving platform script

The ‘MoveTowards’ function is used to achieve this effect, using the platform’s current position and moving it towards the waypoint’s position at a set distance per fixed interval of time. When the platform reaches either waypoint, a boolean variable is switched.

Heads-Up Display (HUD)

The HUD conveys vital information to the player. Information displayed on the canvas is handled through the ‘CanvasManager’ script on the Canvas. When playing the game, it shows the number of deaths the player has, as well as the number of coins. It also shows which power-ups are activated.

```
0 references
void Update()
{
    //Updates death and coin count in canvas
    deathsText.text = playerManager.deaths.ToString();
    coinsText.text = playerManager.coins.ToString();

    DisplayJumpPowerUpImage();
    DisplaySpeedPowerUpImage();
    DisplayJumpBoostPowerUpImage();

    //Activates and deactivates menu
    if (Input.GetKeyDown(KeyCode.Escape) && !menu.activeInHierarchy) {
        DisplayMenu();
    }
    else if (Input.GetKeyDown(KeyCode.Escape) && menu.activeInHierarchy) {
        HideMenu();
    }
}
```

Figure 24. Update function in the canvas manager

The death and coin variables are accessed from the ‘PlayerManager’ script. The text objects are then updated. There are three separate functions that enable the power-up images when they are acquired. When the user presses the escape key, the pause menu is displayed. The pause menu displays ‘Start,’ ‘Respawn,’ and ‘Exit’ buttons. Respawn calls the ‘RespawnPlayer’ function to reset the player’s position. The exit button allows the user to close the application. When the user reaches the end of the game, a finish menu is displayed. The finish menu shows the player a summary of the number of deaths and coins they acquired.

Sound

Sounds are implemented through the environment. While the experience is active, there is music playing in the background that fits the aesthetic of the environment. Without music the experience would feel empty. When coins are collected, a coin sound is made to provide an audio response to the player. This is also done with power-ups, to indicate that a power-up is now active.

Reflection

Upon reflection, I believe that I succeeded in creating a fun and enjoyable physics-based platformer. There were many challenges that were faced when creating the experience. There was some difficulty constructing the environment and finding the line between a fun but challenging experience, or a frustrating one. As a result of this, changes were made throughout development. Originally, there was planned to be a life system, but it proved

frustrating when the player has a limited number of lives for difficult jumps. I sound was implemented more into the experience. When trying to apply sound for the ball rolling, I found it difficult to find and apply an authentic sound that worked well. Instead, the sound felt too abrasive and dissonant, to which I felt it detracted from the experience. However, I believe that the core mechanics worked well to create a comprehensive platformer, with the player controls being effectively delivered to create a fluent and accessible experience. Implementations, like checkpoints, moving platforms and power-ups all worked in harmony to produce an environment which provides a fun challenge for the player. The environment could be developed further by improving small details like sound to create a more responsive and engaging experience for players. More levels could also be added to the experience, adding additional game mechanics like double jumping to enhance the player's control, or adding more obstacles, like AI enemies.