Task 1: 3D scene

For the 3D Scene, I've created a floor for the player to move around on, as well as imported free models from the Unity Asset store to make the environment. They are placed on the scene using **transforms (position, rotation, scale).** 'Spot Light' and 'Point Light' are used on the street lamps*(See Ref.1)*. **Materials**, **textures** and **lights** were added for aesthetic purposes*(See Ref.2)*. 'Fireflies' are added with a **Particle System** that emits a cube container, **fade out** after 5 seconds. **Noise** is enabled for the fireflies to move around randomly, for a more realistic feel.*(See Ref.3).* ***'*Day/Night cycle'** is implemented via **keyframe animation** of the 'Directional Light' rotation and color *(See Ref.4).*

Task 2: Moving object with user input

1) Player

Player is an empty object, made up of hierarchal game objects. It has a **rigid body** attached to it to allow collision detection, as well as 3 scripts. '***PlayerMovement.cs'*** is in charge of the front and back **movement (z-axis)**, as well as player **jump (y-axis)**. The z-value is retrieved by getting the **'Vertical' input**, and is than used to **translate the player** to the new position. Player movement can be controlled via the **'w' and up key for forward movement**, and **'s' and down key for backward movement**. This translation is multiplied by **movementSpeed** and **Time.deltaTime(for frame rate independence)**. Player jump is detected via the **spacebar**, where the **y-value is translated** to the predefined jumpHeight. *(See Ref.5)*

'***PlayerDirection.cs'*** controls the rotation(direction) of the player movement with the mouse position. 'mouseRay' contains information of a ray (world space) going from camera through a screen point. It gets the **x, y values** of where the mouse is pointing at (**z-value is ignored**). **Physics.Raycast** is than used to extract the Vector3 information. The **x and z values are used** as parameters for **transform.LookAt**, so that the camera will always look at the player(lookTarget). The y-value used is the **camera's current y position**, as I don't want the camera to look from a different y position (up or down). This makes the player move according to the mouse direction *(See Ref.6)*

'***PlayerCollision.cs'*** checks for **player-fruit collision** via the 'fruit' tag, **plays an audio effect**, **destroy the fruit gameObject**, and calls **AddScore function** *(See Ref.7)*. A '**Trail Renderer'** component is attached onto the player, which **leaves a transparent trail** behind the player when moved. *(See Ref.8) Note: 'goggles' box collider is disabled, as it will result in 2 collision (body + googles) with fruit.*

2) Camera

A **3rd person camera** is implemented via the '***cameraMovement***' script attached to the Main Camera. This script also allows the **camera to pan horizontally** with mouse drag. **LateUpdate** is used to ensure that it's run after the player moves. The camera follows the player by transforming it's position to the player's position, and **offsetting the z-value**.

ScreenToViewportPoint transforms the mouse position from **screen space into viewport space**. Viewport space is normalized and relative to the camera. The bottom-left of the camera is (0,0); the top-right is (1,1). The z position is in world units from the camera. The difference between original and new mousePosition is the **direction**.

Dragging from one end of the screen to the other should rotate the camera to the other side of the object (180 degree). Thus, **1 viewport unit = 180degrees rotation**. So to calculate the rotation angle, we **multiply the -direction(viewport point) with 180**. Negative direction is used so that when we drag from left to right, the camera pans left. **Transform.RotateAround** is used to rotate the camera **around the player** by the rotation angle calculated. *(See Ref.9)*

Extension Task:

1) Start and End Scene

The StartGame function in ***'startGame.cs'*** is called when the StartButton is clicked on the StartScene(first scene). This **gets the next scene (GameScene) and loads it**. *(See Ref.10)*
'endGame.cs' has 2 functions, Restart and Quit, which are called when the respective buttons are clicked. **'RestartButton' redirects to the StartScene** (first scene), while **'Quit' closes the application**. *(See Ref.11)*

2) SpawnFruit

The Start function in ***'spawnFruit.cs'*** gets a **random index** within the range of 0 and the fruitAssets list length. It than initializes a **random Vector3**. These 2 variables are used as parameters to spawn (**Instantiate) the random fruits at random position**. The **spawn area is limited** to about the front half of the map. This is so that fruits spawned **don't overlap** with the house and fences at the back. **Quaternion.identity** is used so that the object doesn't rotate, and **aligns with the world**. *(See Ref.12)*

GenerateSpriteRenderer is called after each fruit is spawned. This function **creates an empty object** called 'FruitIcon', set the layer, and adds a **SpriteRenderer** component to the fruit object position, with a higher y-value(so that it shows on top of trees). This is because the fruits are **too small to be seen on the minimap**, thus a separate sprite renderer is put in place. 'FruitIcons' are represented with a red 'UISprite' on the minimap. The fruit game object is than set as the **parent** of the sprite game object. This is so that the **sprite gets removed** from the game as well, when collision between player and fruit happens. **Quaternion.Euler** is used to rotate the sprite renderer flat. *(See Ref.13)*

3) Score System

'Score.cs' initializes the score to 0 at Start, and **updates the scoreText** on the canvas with the score *(See Ref.14).* **AddScore** function is called in ***'PlayerCollision.cs'*** when collision with fruit happens. **GameManager** is an empty object with the ***'completeLevel.cs'*** script attached to it. *(See Ref.15)* It gets the total number of fruit spawned from ***'spawnFruit.cs'***, checks that if the score equals to the fruitCount. If so, the game is completed and the **sceneManager loads the next scene** (endScene). *(See Ref.16)*

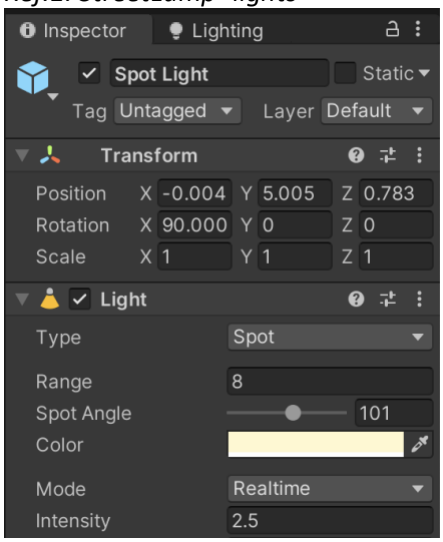
4) MiniMap

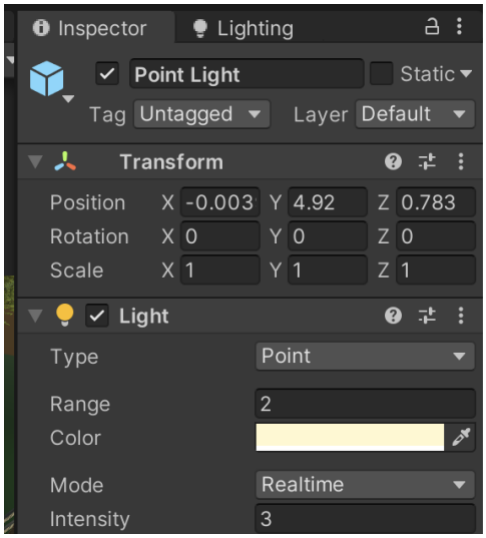
The minimap is used to mainly **display player and fruit locations**, to locate them easier. A MiniMapCamera , RawImage, and MinimapRenderTexture is created. Everything that the **minimapCamera** sees (from top-down view) is put into a render texture (**MinimapRenderTexture**) *(See Ref.17)*, which is then used as the **RawImage's** texture *(See Ref.18)*. This **shows the minimap on the canvas (part of the screen)**, and not on the whole screen covering the game. **Sprite Renderer** for player and fruits are created for a **clearer view of the game object's position on the screen**. They are assigned with layers, **PlayerIcon** and **FruitIcon** *(See Ref.19, 20).* In the MiniMapCamera Culling Mask, we can than select the PlayerIcon and FruitsIcon layers for the icons to show *(See Ref.21).* Then, I deselected them on the Main Camera's Culling Mask, so that than won't be shown in the game *(See Ref.23)*.

The ***'miniMap.cs'*** is attached on MiniMapCamera, which is a **second camera used for the minimap**. **LateUpdate** is used set the camera's x and z values to be the player's position (**follows the player**). The y-value remains the same as the minimap is supposed to be a **2D top-down view**. Lateupdate is used as it is called after Update and FixedUpdate, so that the minimap is only updated after our player has moved. *(See Ref.22, 24)*

Image references:

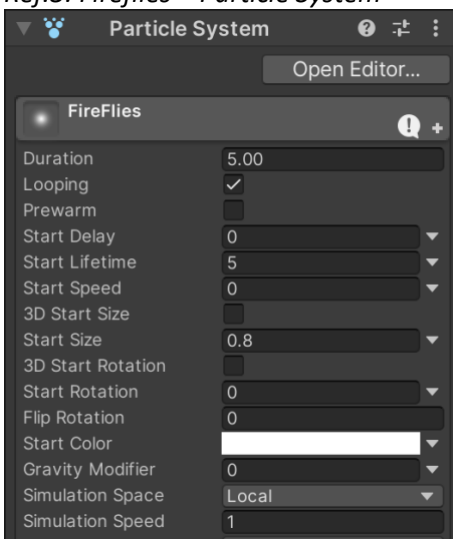
Ref.1: StreetLamp- lights



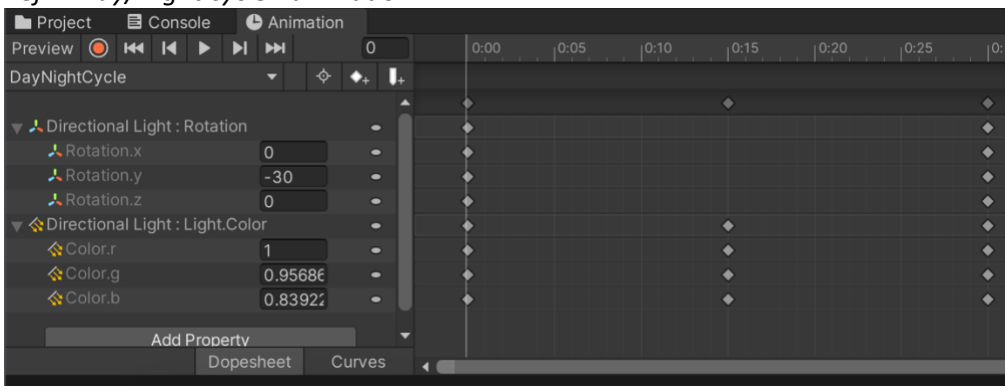


Ref.2: 3D Scene

Ref.3: Fireflies – Particle System



Ref.4: Day/Night cycle - animation



Ref.5: PlayerMovement.cs

```
void Update()
{
    float movement = Input.GetAxisRaw("Vertical");

    // forward and backward motion. +z is forward, -z is backward
    // Time.deltaTime makes frame rate independent
    transform.Translate(new Vector3(0, 0, movement) * movementSpeed * Time.deltaTime)

    // jump
    if (Input.GetButtonDown("Jump"))
    {
        transform.Translate(new Vector3(0, jumpHeight, 0));
    }
```

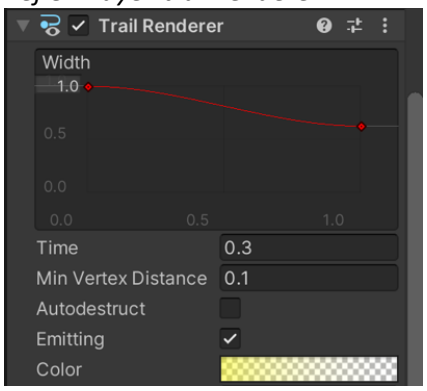
Ref.6: PlayerDirection.cs

```
void Update()
{
    //Create a ray pointing from camera to mouse position on screen
    mouseRay = cam.ScreenPointToRay(Input.mousePosition);

    // To store information of where the ray hit
    RaycastHit hitInfo;

    //Extract vector information from the hitInfo
    if (Physics.Raycast(mouseRay, out hitInfo, 100f))
    {
        hitpoint = hitInfo.point;
    }

    // using only the x and z value of the hitpoint,
    // since I don't want the camera to be looking up or down on the player
    Vector3 lookTarget = new Vector3(hitpoint.x, transform.position.y, hitpoint.z);
    //Rotating the object to that point
    transform.LookAt(lookTarget);
}
```

Ref.7:PlayerCollision.cs

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "fruit")
    {
        // Play sound when collect fruit
        audioSource.Play();

        // Destroy the fruit, along with its child object,
        // the fruitIcon(sprite renderer)
        Destroy(other.gameObject);
        score.AddScore();
    }
}
```

Ref.8: Player-trail renderer

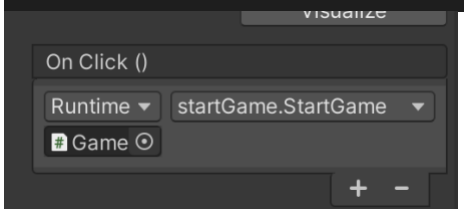


Ref.9: cameraMovement.cs

```
private void LateUpdate()
{
    // camera follows player
    transform.position = playerTransform.position;

    // mouse click
    if (Input.GetMouseButtonDown(0))
    {
        // save mouse position
        originPosition = cam.ScreenToViewportPoint(Input.mousePosition);
    }
    // mouse drag to pan camera
    // eg. if mouse move from left to right, the mouse position difference
    // multiplied by 180degree will be the angle to rotate the camera
    else if (Input.GetMouseButton(0))
    {
        // get the new mouse position
        Vector3 newPosition = cam.ScreenToViewportPoint(Input.mousePosition);

        // calculate the difference between the 2 mouse positions
        Vector3 direction = originPosition - newPosition;

        // calculate the angle to rotate around y axis
        float rotation = -direction.x * 180; // camera moves horizontally

        // rotates camera around player
        transform.RotateAround(playerTransform.position ,new Vector3(0, 1, 0), rotation);

        // make the newPosition to be the originPosition
        originPosition = newPosition;
    }

    //translate to offset the camera
    transform.Translate(new Vector3(0, 0, -Zoffset));
}
```

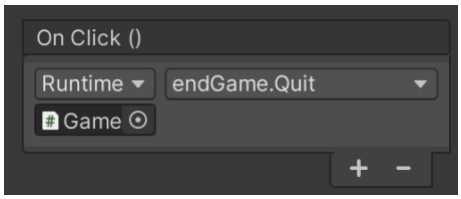
Ref.10: StartGame.cs

```
public void StartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```



Ref.11: Restart.cs

```
public void Restart()
{
    //Loads first scene
    SceneManager.LoadScene(0);
}

public void Quit()
{
    Debug.Log("quit");
    Application.Quit();
}
```



Ref.12: spawnFood.cs - Start

```
void Start()
{
    for (var i = 0; i < fruitCount; i++)
    {
        int randomIndex = Random.Range(0, fruitAssets.Length);
        randomPosition = new Vector3(Random.Range(-5.8f, 22),
                                     Random.Range(1, 4),
                                     Random.Range(-5.8f, 22));

        GameObject fruitObject = Instantiate(fruitAssets[randomIndex],
                                             randomPosition,
                                             Quaternion.identity);

        //Generate sprite for the minimap (to display where the fruits are at)
        GenerateSpriteRenderer();

        // Make the FruitIcon the child of the Fruit object
        FruitIcon.transform.SetParent(fruitObject.transform);
    }
}
```

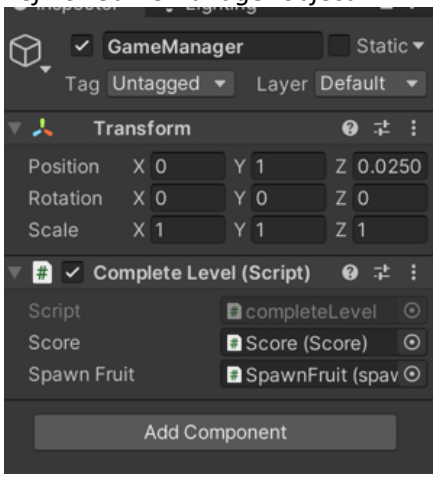
Ref.13: spawnFood.cs - GenerateSpriteRenderer

```
void GenerateSpriteRenderer()
{
    // Create new game object
    FruitIcon = new GameObject("FruitIcon");

    // Set it to layer 9, to show it on the minimap
    FruitIcon.layer = 10;

    // Create a sprite renderer, to use it as the fruit's location on the minimap
    SpriteRenderer renderer = FruitIcon.AddComponent<SpriteRenderer>();
    renderer.sprite = sprite;

    // Set the sprite's color
    Color color;
    if (ColorUtility.TryParseHtmlString("#FF4C4C", out color))
    {
        renderer.color = color;
    }

    // Set the sprite's position to be the same as the fruit object's position
    FruitIcon.transform.position = new Vector3(randomPosition.x, 15f, randomPosition.z);
    FruitIcon.transform.rotation = Quaternion.Euler(90f, 0f, 0f);
    FruitIcon.transform.localScale = new Vector3(10, 10, 1);
}
```

Ref.14: Score.cs

```
// Start is called before the first frame update
void Start()
{
    score = 0;
}

// Update is called once per frame
void Update()
{
    int totalFood = spawnFruit.fruitCount;
    scoreText.text = "Fruits: " + score.ToString("0") + "/" + totalFood;
}

public void AddScore()
{
    score++;
}
```

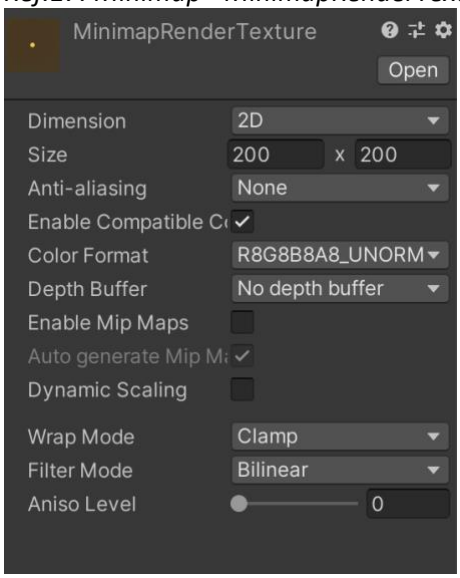
Ref.15: GameManager object



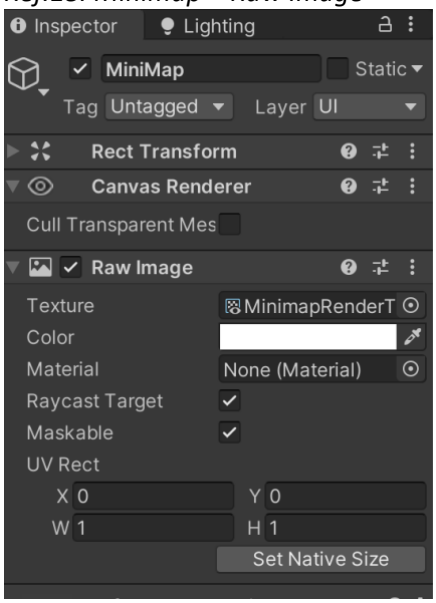
Ref.16: completeLevel.cs

```
// Update is called once per frame
void Update()
{
    int totalFruit = spawnFruit.fruitCount;
    if (totalFruit == score.score)
    {
        Debug.Log("You have cleared the game!");
        // Loads the next scene (end scene)
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

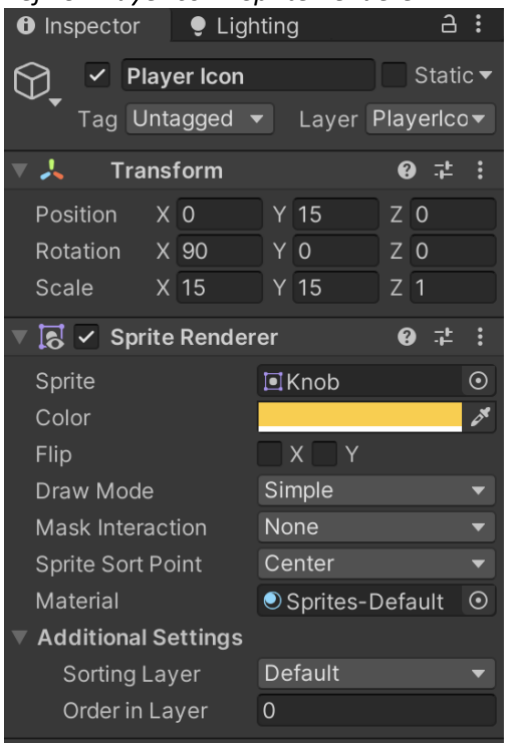
Ref.17: Minimap - MinimapRenderTexture



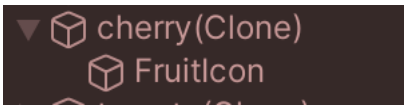
Ref.18: Minimap – Raw Image



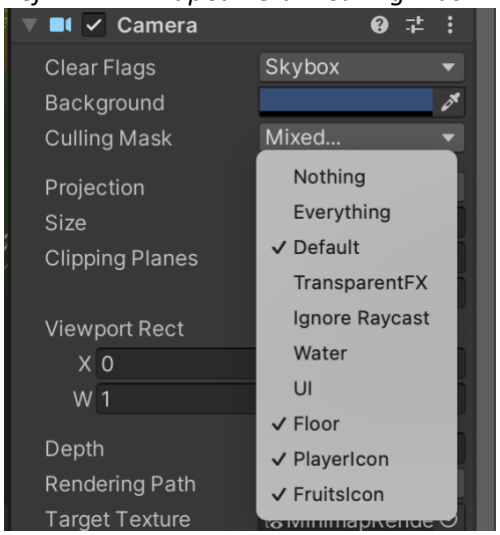
Ref.19: PlayerIcon - Sprite Renderer



Ref.20: FoodIcon - SpriteRenderer



Ref.21: MiniMapCamera – Culling Mask



Ref.22: MiniMap

Ref.23: Main Camera - Culling Mask

Ref.24: minimap.cs

```
void LateUpdate()
{
    // set potition of minimap to be player's position
    Vector3 newPosition = player.position;

    // set the y value to the current y position
    newPosition.y = transform.position.y;

    // so that map moves along
    transform.position = newPosition;
}
```

Other references:

Fireflies Particle System - https://www.youtube.com/watch?v=9O5XU_jshBU
Day/Night cycle animation - https://www.youtube.com/watch?v=uQCYT_WMzN8
PlayerDirection.cs – *video lecture*
cameraMovement.cs *(horizontal pan)* - *https://emmaprats.com/p/how-to-rotate-the-camera-around-an-object-in-unity3d/*
MiniMap - https://www.youtube.com/watch?v=28JTTXqMvOU