Task 1: 3D scene with physics objects

The player has to pass through all physics obstacles, reach the finishing line, to clear the game. **Transforms (position, rotation, scale)** are used to layout the game objects *(See Ref.1).* The **player is an imported game asset**, which contains the player model, materials, and animations.

1) Cannon (physics objects)

The **Cannon** object utilizes physics.*(See Ref.2)* Cannon is an empty game object containing **'cannon' model** (imported asset), and a **'cannonSpawnPoint'***(See Ref.3).* ***'spawnCannon.cs'*** is attached to the cannon model. In the Start function, **StartCouroutine()** is called, with **Cannon()** as its parameter.

Cannon() is an IEnumerator, which waits for a few seconds (delay), before calling **SpawnCannon()**. As the name suggests, it checks if the player is in front of, and within a certain range from the cannon. It than spawns cannon ball prefabs with the **Instantiate()** method, parsing **cannonPrefab**, **cannonSpawnPoint position and rotation** as parameters. **Coroutine** and **IEnumerator** is used to allow **execution of game logic over a number of frames**. It allows pausing of a function to wait for a condition before continuing. This is ideal for spawning cannons as it requires a delay. *(See Ref.4)*

The **cannonPrefab** has a ***'cannonMovement.cs'*** script attached to it. It adds a force to 'shoot' the cannon balls. I've used **'Impulse' ForceMode**, to mimic the cannon's **instantaneous** force. The cannon balls **destroys itself after 1 second**. This is to not fill the platform with cannon balls. When collision between player and cannon ball happens, the **player will be pushed back** by the its force. The **cannon ball will then be destroyed**. *(See Ref.5)*

Task 2: Movement Control

1) Player (physics-based movement)

The player has a **Rigidbody** component and a **Capsule Collider** to detect collision with obstacles *(See Ref.6).* **AudioSource** component is added for jumping sound effect *(See Ref.7).* ***'playerMovement.cs'*** and ***'playerAnimation.cs'*** scripts are attached to this player game object as well *(See Ref.8)*.

The **Update()** function of ***'playerMovement.cs'*** checks if the **player has fell** below the platform for a certain distance, and **reloads the scene**. This acts as a respawn feature when the player dies. I chose to reload the scene instead of resetting the player's position and velocity as, some game object (tile drop) needs to be reset as well. So, reloading the scene is the best method in this case *(See Ref.9)*.

In **FixedUpdate()**, the **'Horizontal' and 'Vertical' input** is used as the **direction** to move to. **Normalizing** it makes the direction have a magnitude of 1. Then, **AddForce** is used on the player's rigidbody, for **physics force movement** on the player, instead of just 'teleporting' (transform position) the player to the desired position. **Time.deltaTime** is used for **frame rate independence**. **'VelocityChange'** ForceMode **adds an instant velocity change** to the rigidbody, ignoring its mass. To rotate the player to the correct angle, first check that if the direction is not Vector.zero, means there's rotation, calculate the angle with **Quaternion.LookRotation**, parsing in the **direction** and **Vector.up**. Using transform.rotation, **rotate the player** towards the calculated angle *(See Ref.10)*.

Next, check if the player is on the ground and spacebar is tapped, **play the jump sound effect**, **add a upward force** to make the player jump, and **set the isGrounded to false**. This is to disable double jumping. To ensure that the player's velocity isn't too high, a maxSpeed is set, and if it passes the speed limit, use **ClampMagnitude to change the velocity to the maxSpeed** *(See Ref.10).*

While the player is on the rotating plates, they aren't able to jump due to its y-value of the player sometimes being lesser than 0 when the plate is tilted. For player to be able to jump while on the rotating plates, **OnCollisionEnter** and **OnCollisionExit** are used. If the gameObject the **player is collided with is the rotatingPlates**, **alter the isGrounded boolean state** accordingly *(See Ref.11).* The player's first position is saved in the start of ***'playerAnimation.cs'***. The player's **last position and speed is updated every frame** in

the Update function. If the last position equals to the first position, this means that the **player is on the platform**, so **set isGrounded to true** *(See Ref.12).*

FixedUpdate() changes the player's animation. If the player's x and y last position is different from the first position, the **player is moving**, set the animator's '**isMoving'** boolean to true. The speed of the player is updated, to transition between '**walking'** and '**running'** animation. To check if the **player is jumping**, compare the first and last position's y-value. The '**isJumping'** Boolean is changed accordingly. Lastly, update to **reset the first position** to be the last position *(See Ref.13).* The animation transitions are conditioned according to the boolean and float variables *(See Ref.14 -17).*

Extension Task:

1) Start and End Scene

'startScene' has a UI button that calls ***'gameManager.cs'*** **StartGame()** when clicked, to load the **'GameScene'** *(See Ref.18)*. The 'GameScene' has a cube used to trigger the game to end *(See Ref.19)*. **'OnTriggerEnter()'** is called when the player collides with this **'endTrigger'**, to load the 'endScene'. The **'endScene'** has 2 buttons, restart and quit, and calls the respective functions when clicked.

2) Camera follow player

'cameraMovement.cs' is attached on the main camera, which offsets the camera, and transform its position to follow the player constantly *(See Ref.20).*

3) Audio effect

An **AudioSource** component is attached onto an empty gameObject called **'audioManager'**, with its audioClip set as the background music for the game. **'Play On Awake'** and **'Loop'** is checked so that the music plays when the scene loads, and loops repeatedly *(See Ref.21).*

4) Rotator

The rotator obstacles *(See Ref.22)* has a script, ***'rotator.cs'*** attached. The Update function within rotates the cylinder at its x-axis, at a constant speed. **Space.Self** signifies that the rotation is relative to the **local coordinate system** *(See Ref.23).*

5) BigBlocks

The 2 blocks moves constantly on the x-axis *(See Ref.24).* In ***'movingBlock.cs'***, the start position is initialized, and the block is transformed by the offset on the x-axis. In **Update()**, **'goRight'** is used to determine which **direction the block should first move**. If the x-value is less than the desired x-value, continue moving in that direction. Else, move towards the other direction *(See Ref.25).*

6) TileDrop

The tiles are attached with ***'fallTile.cs'***, which uses **OnCollisionEnter** to check the player's collision with the tile. When collision happens, the **tile will be destroyed in 0.5seconds** *(See Ref.26).*

7) Pendulum

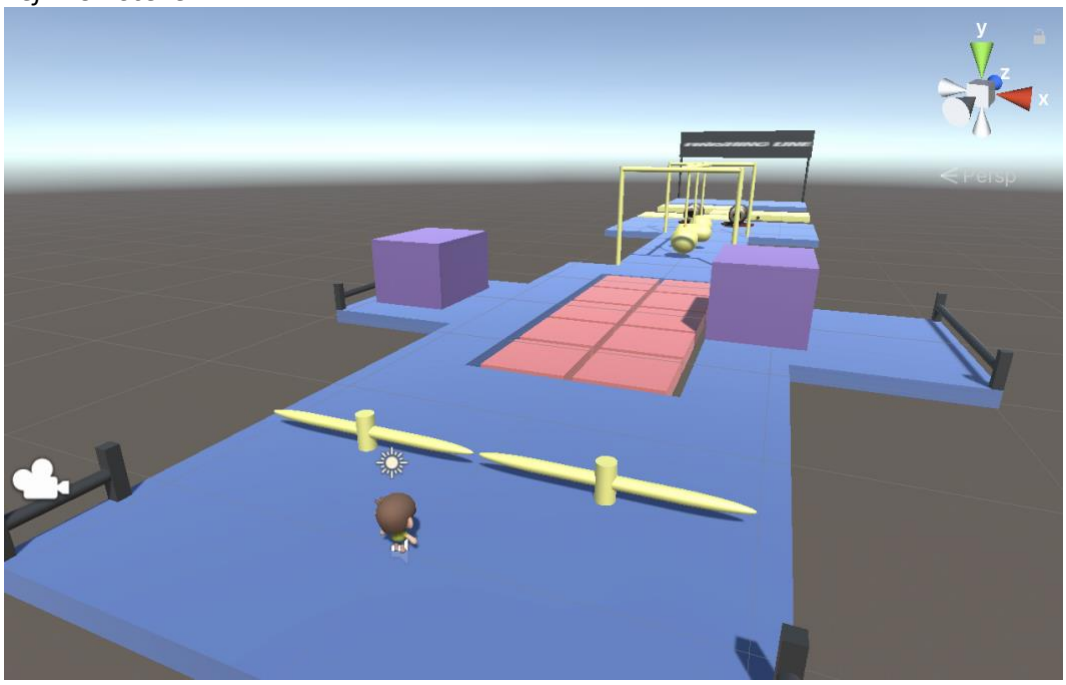
As per the moving blocks, **pendulum's direction** is also decided by the **goRight** boolean, which is than checked/ unchecked based on the direction I want it to start moving towards. *(See Ref.27).*
In ***'pendulum.cs'***, **Mathf.Sin** is used to calculate the rotation angle. **Transform.localRotation** is used along with **Quaternion.Euler** to rotate the pendulum along the z-axis *(See Ref.28).*

8) Rotating Plates (Hinge Joint)

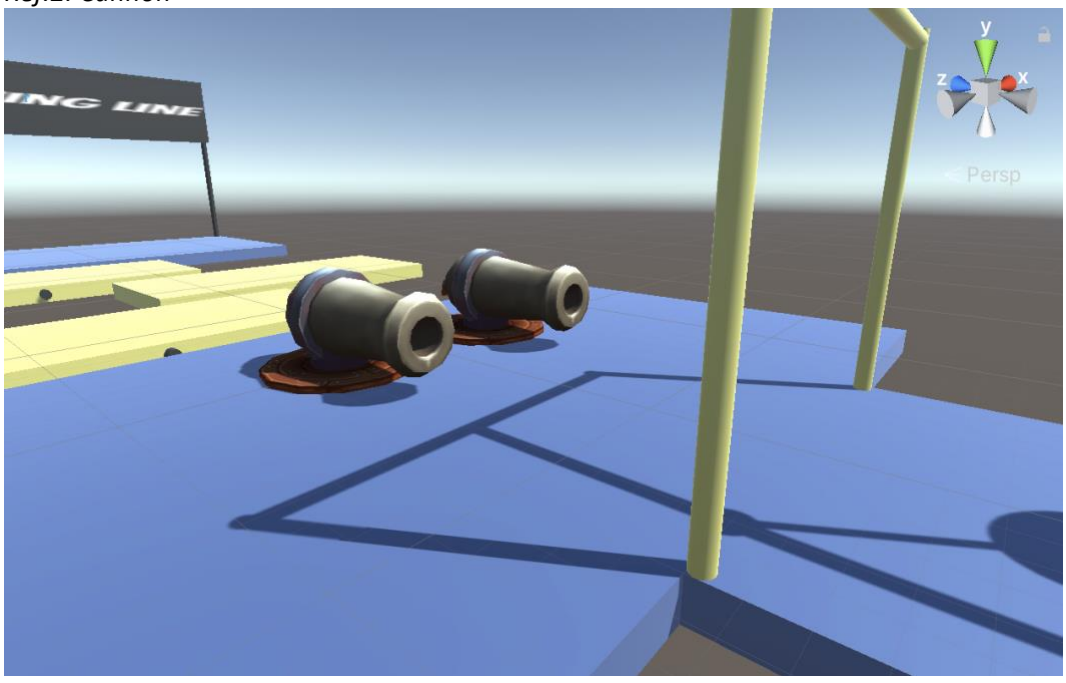
When player stands in one side of the plate, it tilts slowly *(See Ref.29).* This is done by having a rigidbody component, with a mass of 10 so that it doesn't tilt too much. A **hinge joint** is put in place, with its **connected anchor's position** set to the **middle of the plate**. This simulates the tilting plate mechanism *(See Ref.30).*

Image references:

Ref.1: 3D Scene



Ref.2: Cannon



Ref.3: Cannon - game object hierarchy

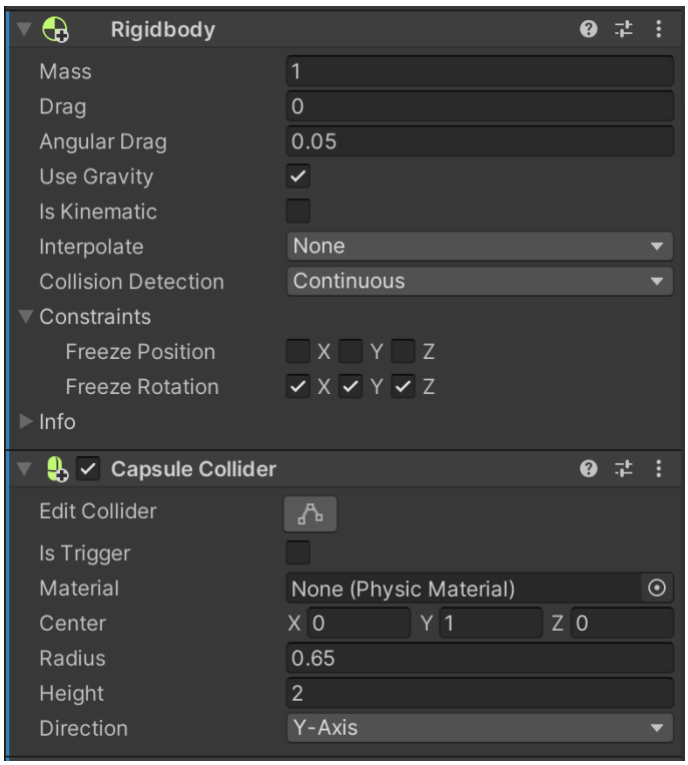
Ref.4: 'spawnCannon.cs'

```
// Start is called before the first frame update
void Start()
{
    //coroutine allows spreading tasks across several frames
    // start the coroutine
    StartCoroutine(Cannon());
}

// Update is called once per frame
private void SpawnCannon()
{
    // if the player is within a certain distance from the cannon
    // and if the player is still infront of the cannon
    // (haven't gone pass the cannon z-position )
    if(playerTransform.position.z + distance >= cannonSpawnPoint.position.z &&
       playerTransform.position.z <= cannonSpawnPoint.position.z)
    {
        // create cannons
        Instantiate(cannonPrefab, cannonSpawnPoint.position, cannonSpawnPoint.rotation);
    }
}

IEnumerator Cannon()
{
    while(true)
    {
        // calls SpawnCannon after a delay
        yield return new WaitForSeconds(spawnTime);
        SpawnCannon();
    }
}
```

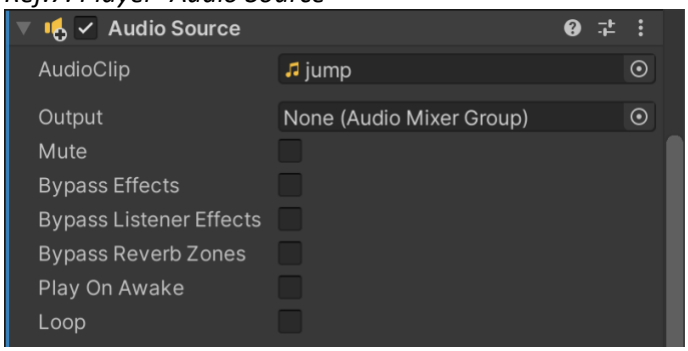
Ref.5: 'cannonMovement.cs'

```
void Start()
{
    // add a force to the cannon to shoot it
    cannon_rb = GetComponent<Rigidbody>();
    cannon_rb.AddForce(transform.forward * speed, ForceMode.Impulse);

    // automatically destroy the cannon after 1 second
    Destroy(gameObject, 1f);
}

private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "player")
    {
        // destroy cannon when it hits player
        Destroy(cannon_rb.gameObject);
    }
}
```

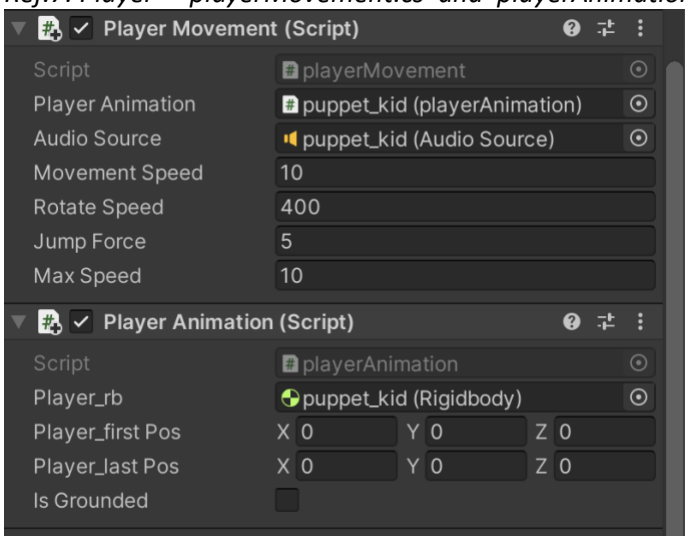
Ref.6: Player – Rigidbody and Capsule Collider



Ref.7: Player- Audio Source



Ref.7: Player – 'playerMovement.cs' and 'playerAnimation.cs'



Ref.9: Player - 'playerMovement.cs' – Start() and Update()

```
// Start is called before the first frame update
void Start()
{
    player_rb = GetComponent<Rigidbody>();
    audioSource = GetComponent<AudioSource>();
}

// Update is called once per frame
void Update()
{
    // if player falls, reload the scene
    if (transform.position.y < -10)
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }
}
```

Ref.10: Player- 'playerMovement.cs' – FixedUpdate()

```
void FixedUpdate()
{
    float xAxis = Input.GetAxisRaw("Horizontal");
    float zAxis = Input.GetAxisRaw("Vertical");

    //calculate the direction to move
    Vector3 direction = new Vector3(xAxis, 0, zAxis);
    // normalize direction to make sure it has a magnitude of 1
    direction.Normalize();

    // move player based on keys (up, down, left, right)
    //Time.deltaTime is used for frame rate independence
    player_rb.AddForce(direction * movementSpeed * Time.deltaTime, ForceMode.VelocityChange);

    if (direction != Vector3.zero)
    {
        // create rotation angle to look at direction
        // set forward direction to movement direction, set up-direction to y-axis
        Quaternion toRotation = Quaternion.LookRotation(direction, Vector3.up);

        // rotate from current rotation to direction
        transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotateSpeed * Time.deltaTime);
    }

    // only allow jumping when player isGrounded
    if (playerAnimation.isGrounded && Input.GetButtonDown("Jump"))
    {
        audioSource.Play();
        player_rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
        playerAnimation.isGrounded = false;
    }

    // limit player's max speed
    if (player_rb.velocity.magnitude > maxSpeed)
    {
        player_rb.velocity = Vector3.ClampMagnitude(player_rb.velocity, maxSpeed);
    }
```

Ref.11: Player- 'playerMovement.cs' – OnCollisionEnter() and OnCollisionExit()

```
//check if player is on the ground
//(for player to be able to jump on rotating plates)
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "rotatingPlates")
    {
        playerAnimation.isGrounded = true;
    }
}

//check if player is NOT on the ground
//(for player to NOT be able to jump when NOT on rotating plates)
void OnCollisionExit(Collision collision)
{
    if (collision.gameObject.tag == "rotatingPlates")
    {
        playerAnimation.isGrounded = false;
    }
}
```

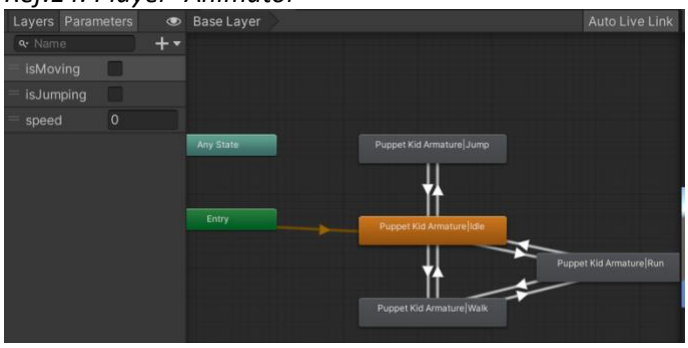
Ref.12: Player- 'playerAnimation.cs' – Start() and Update()

```
// Start is called before the first frame update
void Start()
{
    animator = GetComponent<Animator>();

    // save first position
    player_firstPos = this.transform.position;
}

// Update is called once per frame
void Update()
{
    // save last position
    player_lastPos = this.transform.position;

    // update the player's speed
    speed = player_rb.velocity.magnitude;

    // if y-value didnt change, means player isGrounded
    // this is to not allow double jumps
    if (player_lastPos.y == player_firstPos.y)
    {
        isGrounded = true;
    }
}
```

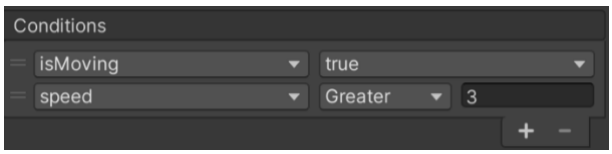
Ref.13: Player- 'playerAnimation.cs' – FixedUpdate()

```
void FixedUpdate()
{
    // if x and z value is different from previous value, means player is moving
    if (player_lastPos.x != player_firstPos.x || player_lastPos.z != player_firstPos.z)
    {
        // set isMoving to true
        animator.SetBool("isMoving", true);
        animator.SetFloat("speed", speed);
    }
    else
    {
        animator.SetBool("isMoving", false);
        animator.SetFloat("speed", 0);
    }

    // if y-value is different from previous value, means player is jumping
    if (player_lastPos.y != player_firstPos.y)
    {
        animator.SetBool("isJumping", true);
        animator.SetFloat("speed", 0);
    }
    else
    {
        animator.SetBool("isJumping", false);
        animator.SetFloat("speed", speed);
    }

    // reset the first position
    player_firstPos = player_lastPos;
}
```

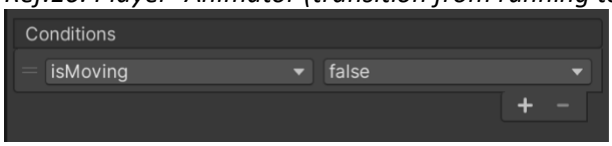
Ref.14: Player- Animator



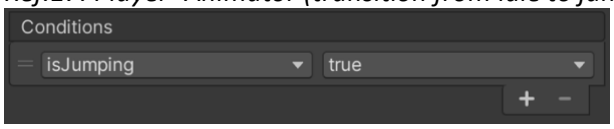
Ref.15: Player- Animator (transition from walking to running)



Ref.16: Player- Animator (transition from running to idle)



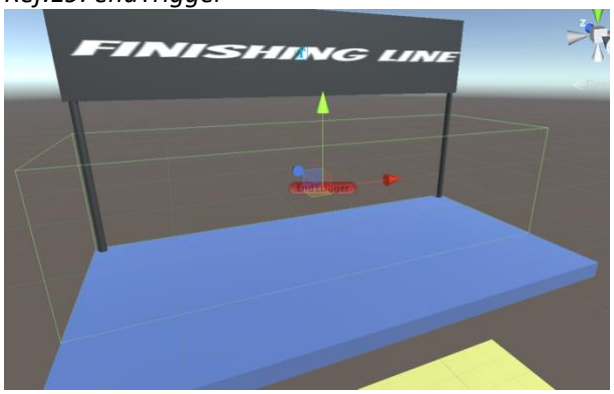
Ref.17: Player- Animator (transition from idle to jumping)



Ref.18: 'gameManager.cs'

```
void OnTriggerEnter(Collider other)
{
    if(other.gameObject.tag == "player")
    {
        // Get the next scene
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

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

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

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

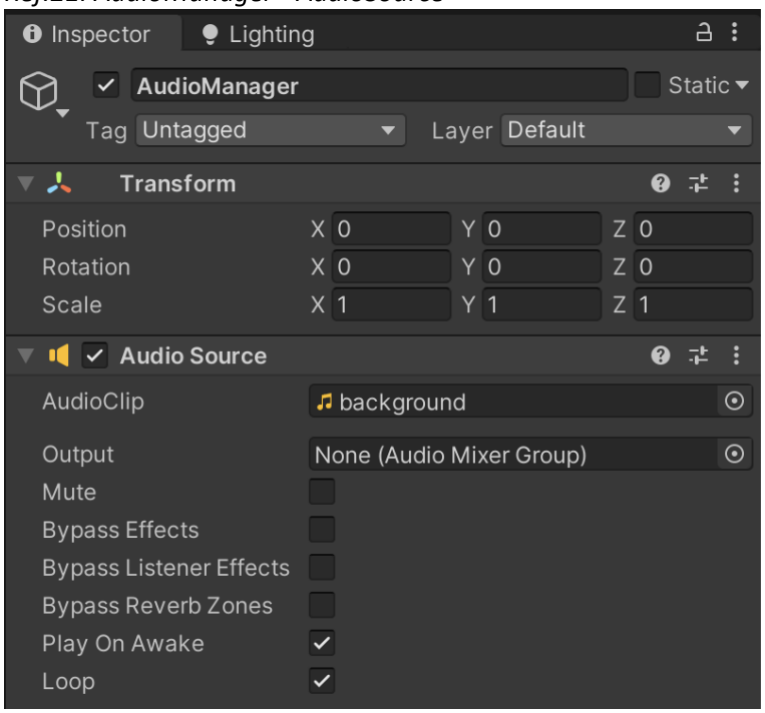
Ref.19: endTrigger



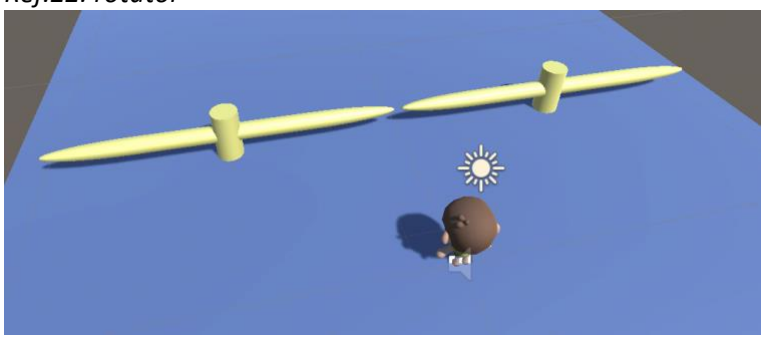
Ref.20: 'cameraMovement.cs'

```
// Start is called before the first frame update
void Start()
{
    //offset how far the camera should be behind the player
    offset = transform.position - player.transform.position;
}

// Update is called once per frame
void Update()
{
    transform.position = player.transform.position + offset;
}
```

Ref.21: AudioManager - AudioSource



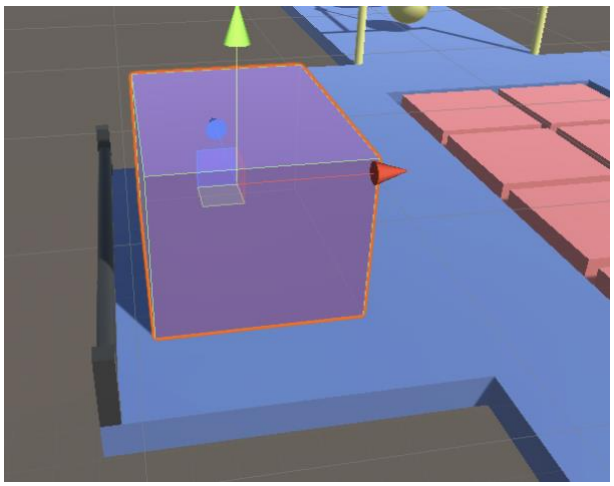
Ref.22: rotator



Ref.23: 'rotator.cs'

```
void Update()
{
    // rotate anticlockwise (-speed)
    transform.Rotate(-speed * Time.deltaTime / 0.01f, 0f, 0f, Space.Self);
}
```

Ref.24: movingBlock



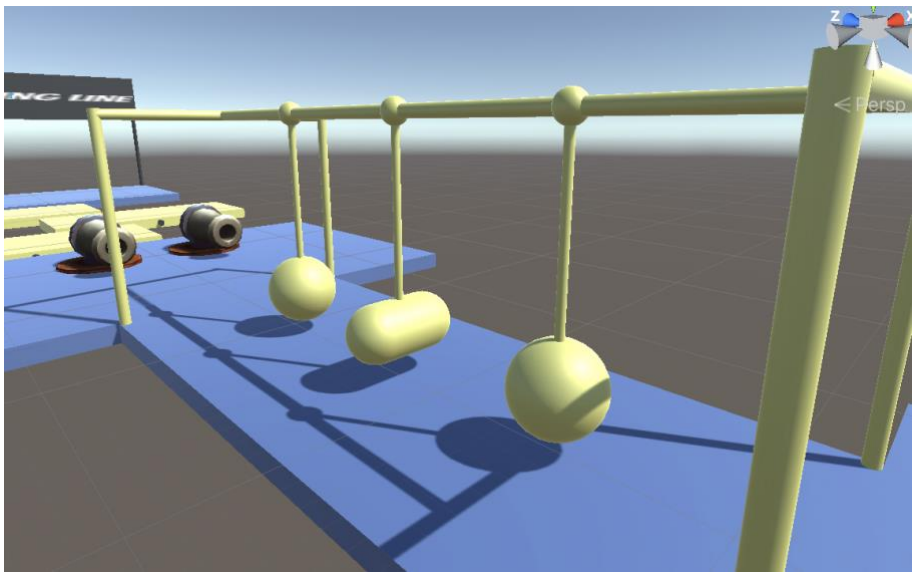
Ref.25: 'movingBlock.cs'

```
void Start()
{
    startPos = transform.position;
    transform.position += Vector3.right * offset;
}

// Update is called once per frame
void Update()
{
    if (goRight)
    {
        // move block right if x-value is less than startPos.x + distance to move
        if (transform.position.x < startPos.x + distance)
        {
            transform.position += Vector3.right * Time.deltaTime * speed;
        }
        else
        {
            // set to false to move left next
            goRight = false;
        }
    }
    else
    {
        // move block left if x-value is more than startPos.x
        if (transform.position.x > startPos.x)
        {
            transform.position -= Vector3.right * Time.deltaTime * speed;
        }
        else
        {
            // set to true to move right next
            goRight = true;
        }
    }
```

Ref.26: 'fallTile.cs'

```
void OnCollisionEnter(Collision collision)
{
    // if player collides with tile, destory that tile after 0.5seconds
    if (collision.gameObject.tag == "player")
    {
        Destroy(gameObject, 0.5f);
    }
}
```

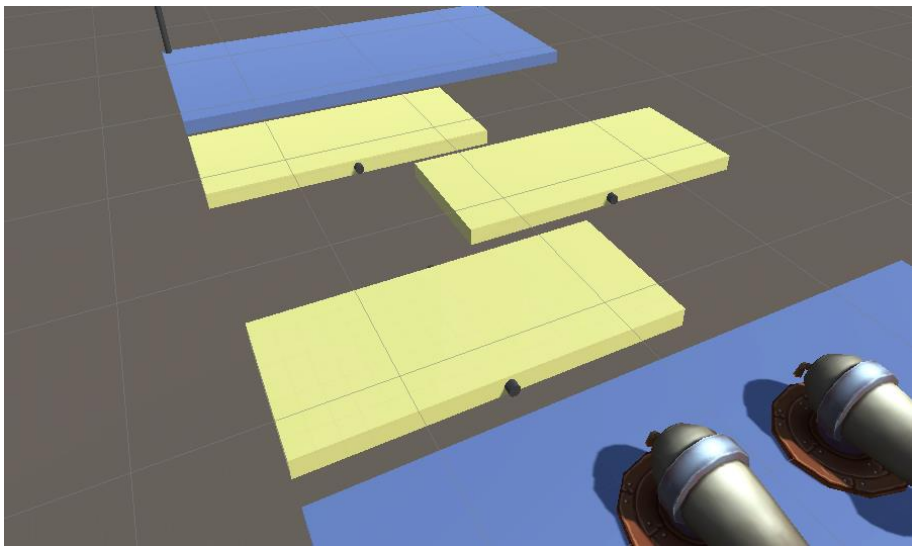
Ref.27: Pendulums



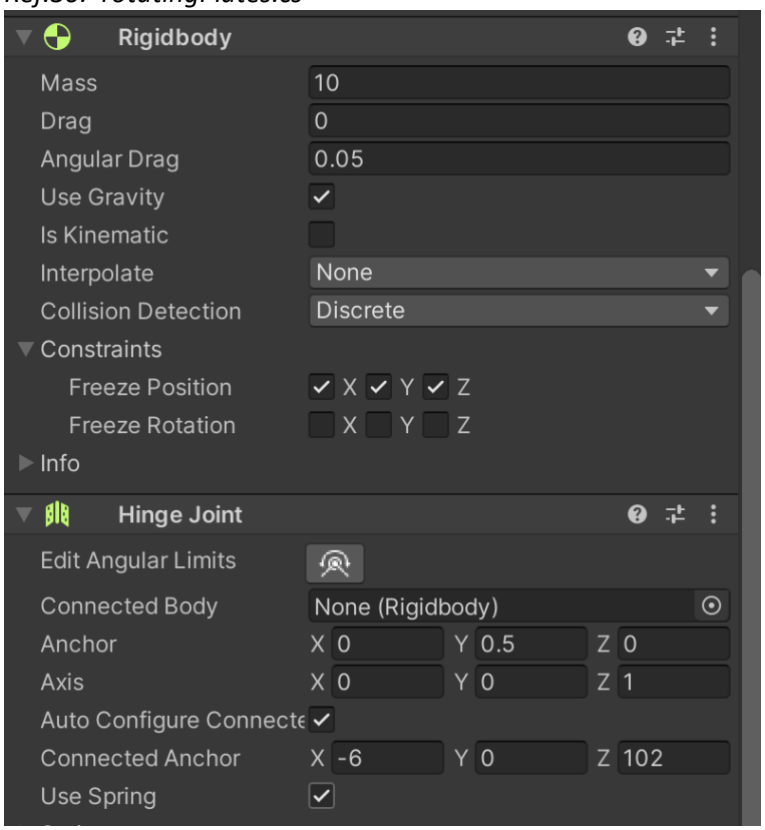
Ref.28: 'pendulumn.cs'

```
float maxAngle = 30.0f;
float speed = 2.5f;
[SerializeField] bool goRight;

// Update is called once per frame
void Update()
{
    // pendulum rotate right first
    if (goRight)
    {
        // calculate the angle to rotate
        float angle = maxAngle * Mathf.Sin(Time.time * speed);
        // use localRotation for the rotation to be relative to the parent
        transform.localRotation = Quaternion.Euler(0, 0, angle);
    }
    // pendulum rotate left first
    else
    {
        // calculate the angle to rotate
        float angle = maxAngle * Mathf.Sin(Time.time * speed);
        // use localRotation for the rotation to be relative to the parent
        transform.localRotation = Quaternion.Euler(0, 0, -angle);
    }

}
```

Ref.29: Rotating Plates



Ref.30: 'rotatingPlates.cs'



Other references:

Animation transition reference - https://www.youtube.com/watch?v=2_Hn5ZsUIXM&t=363s
Player rotation - https://www.youtube.com/watch?v=BJzYGsMcy8Q
Pendulum rotation - https://forum.unity.com/threads/simple-pendulum-movement.595120/