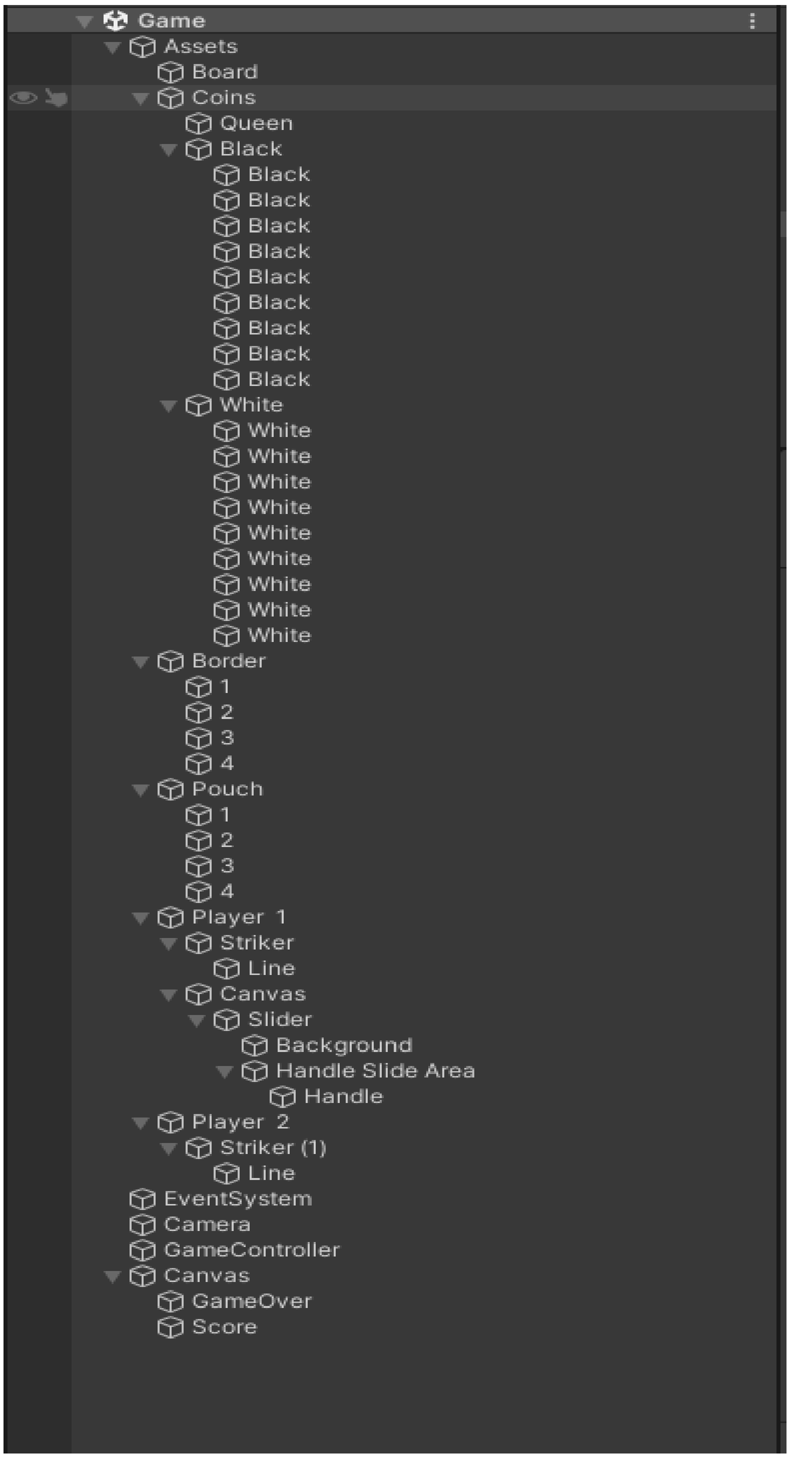
Read Me

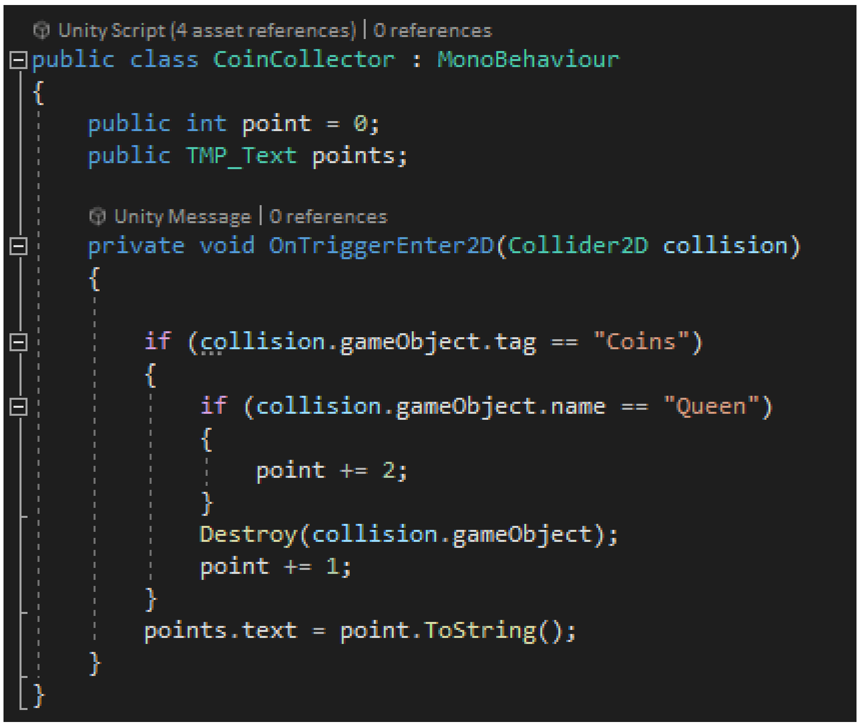
- **Hierarchy**



```
Game
  Assets
    Board
    Coins
      Queen
      Black
        Black
        Black
        Black
        Black
        Black
        Black
        Black
        Black
        Black
      White
        White
        White
        White
        White
        White
        White
        White
        White
        White
    Border
      1
      2
      3
      4
    Pouch
      1
      2
      3
      4
    Player 1
      Striker
        Line
      Canvas
        Slider
          Background
          Handle Slide Area
            Handle
    Player 2
      Striker (1)
        Line
  EventSystem
  Camera
  GameController
  Canvas
    GameOver
    Score
```

  -
  - The hierarchy contains Board, Coins, Striker, Border Colliders, Pouches, Player 1 & 2 GameObjects and a Canvas to store the score and a final Game Over message.

- **Scripts**
  - **Coin Collectors :**

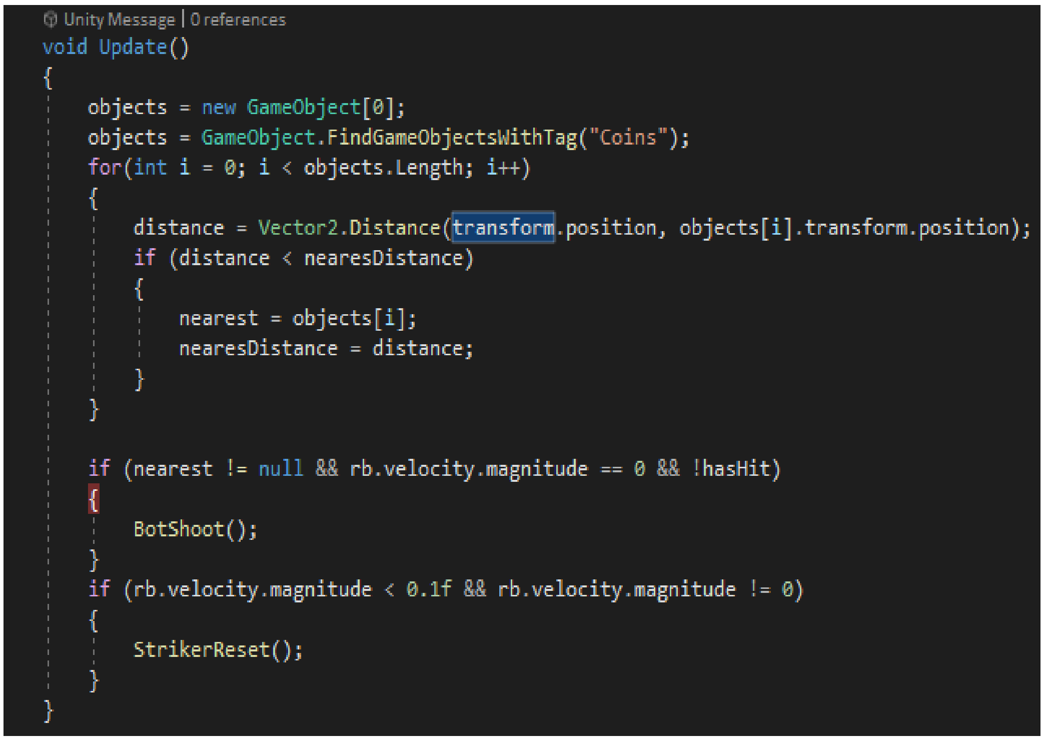


```
Unity Script (4 asset references) | 0 references
public class CoinCollector : MonoBehaviour
{
    public int point = 0;
    public TMP_Text points;

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {

        if (collision.gameObject.tag == "Coins")
        {
            if (collision.gameObject.name == "Queen")
            {
                point += 2;
            }
            Destroy(collision.gameObject);
            point += 1;
        }
        points.text = point.ToString();
    }
}
```

    -
    - The integer "**point**" is to calculate the points and the TMP text "**points**" are used to display on the screen.
    - Points are calculated using the pouches which have colliders attached to them, and trigger the method when a coin comes in contact with it.
    - The Queen has 2 points and every other coin is 1 point each.
  - **AIBot :**

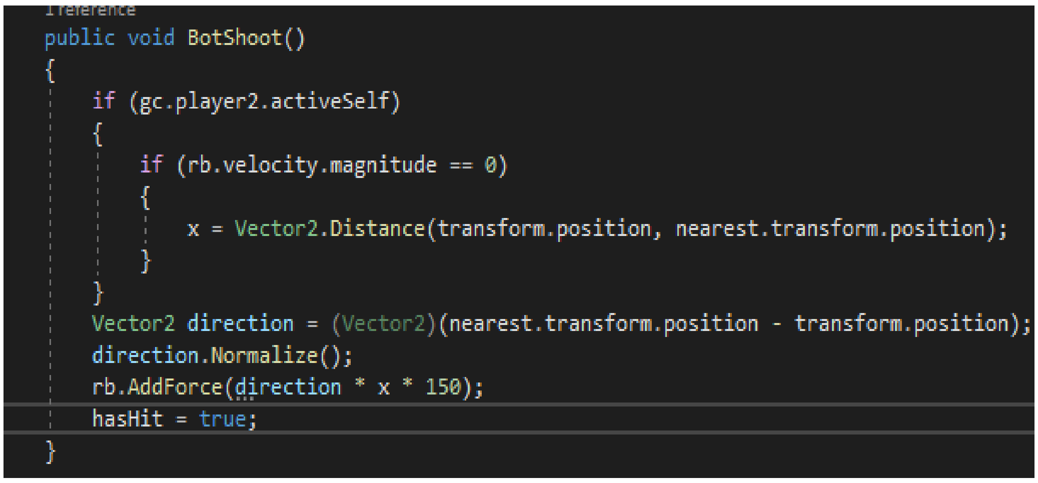


```
Unity Message | 0 references
void Update()
{
    objects = new GameObject[0];
    objects = GameObject.FindGameObjectsWithTag("Coins");
    for(int i = 0; i < objects.Length; i++)
    {
        distance = Vector2.Distance(transform.position, objects[i].transform.position);
        if (distance < nearesDistance)
        {
            nearest = objects[i];
            nearesDistance = distance;
        }
    }

    if (nearest != null && rb.velocity.magnitude == 0 && !hasHit)
    {
        BotShoot();
    }
    if (rb.velocity.magnitude < 0.1f && rb.velocity.magnitude != 0)
    {
        StrikerReset();
    }
}
```

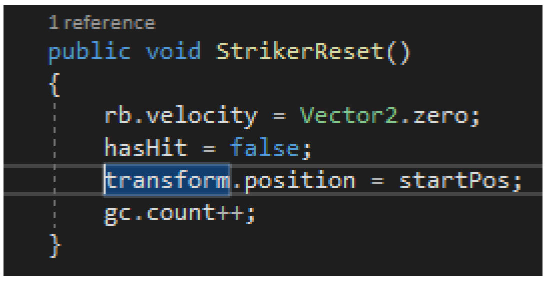
    -

- Update method uses an array of gameobjects to calculate the nearest object to the striker by using a method called **Vector2.Distance** and store it in the GameObject **nearest**.
- If the nearest object is identified, the velocity of the striker is 0 and it has not been striked yet then the bot shoots the striker towards the **nearest** object.
- Once the velocity reaches less than 0.1 and has not reached 0 then the striker position is reset.



```
1 reference
public void BotShoot()
{
    if (gc.player2.activeSelf)
    {
        if (rb.velocity.magnitude == 0)
        {
            x = Vector2.Distance(transform.position, nearest.transform.position);
        }
    }
    Vector2 direction = (Vector2)(nearest.transform.position - transform.position);
    direction.Normalize();
    rb.AddForce(direction * x * 150);
    hasHit = true;
}
```

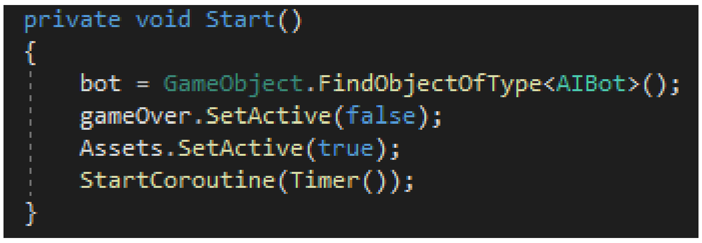
-
- The **Shoot** method checks if the Player 2(bot) is active and if the velocity is 0 then it calculates the distance between striker and the nearest object and calculates the direction to be hit in and a force is added to the rigidbody striker.



```
1 reference
public void StrikerReset()
{
    rb.velocity = Vector2.zero;
    hasHit = false;
    transform.position = startPos;
    gc.count++;
}
```

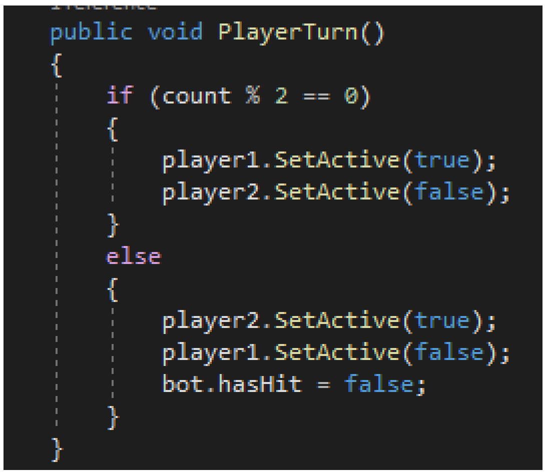
-
- The **Striker Reset** method sets the velocity and the position to 0 and the initial centre position respectively. Count is used to check the player turn.

- **Game Controller :**



```
private void Start()
{
    bot = GameObject.FindObjectOfType<AIBot>();
    gameOver.SetActive(false);
    Assets.SetActive(true);
    StartCoroutine(Timer());
}
```

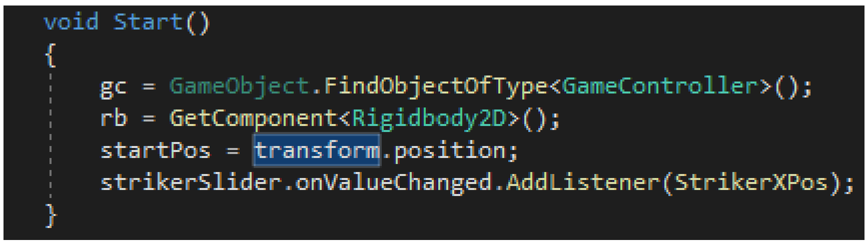
-
- The **Game controller** is used to set the **Assets** to be active and the **Game Over message** to be inactive.
- A coroutine **Timer** is started to clock the 2 mins, after which the **Game Over** message is displayed.

- 

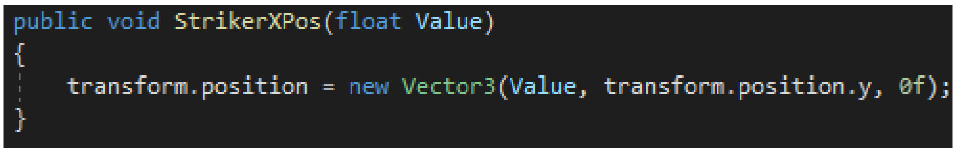
```
public void PlayerTurn()
{
    if (count % 2 == 0)
    {
        player1.SetActive(true);
        player2.SetActive(false);
    }
    else
    {
        player2.SetActive(true);
        player1.SetActive(false);
        bot.hasHit = false;
    }
}
```

- The **Player Turn** method is called in the update method to check whose turn it is to play and their respective Gameobjects like striker and slider are set active.

- **Striker Controller :**

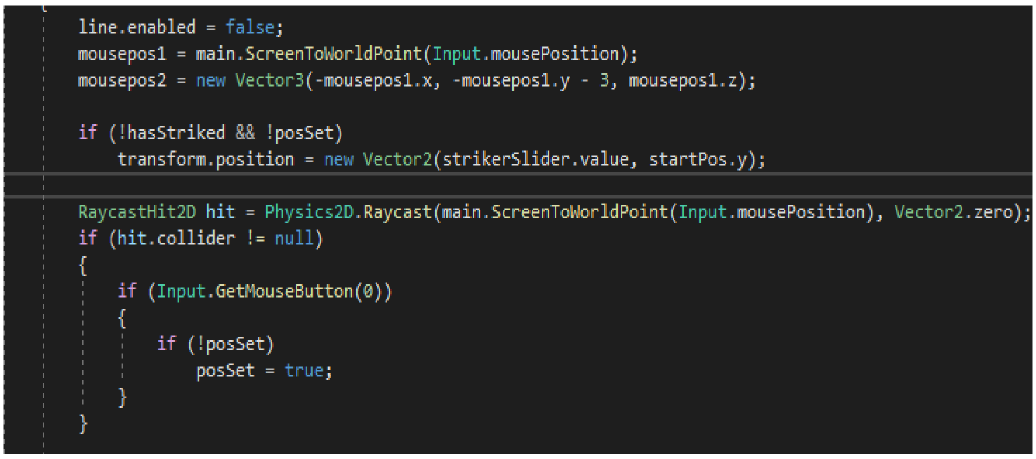
- 

```
void Start()
{
    gc = GameObject.FindObjectOfType<GameController>();
    rb = GetComponent<Rigidbody2D>();
    startPos = transform.position;
    strikerSlider.onValueChanged.AddListener(StrikerXPos);
}
```

- 

```
public void StrikerXPos(float Value)
{
    transform.position = new Vector3(Value, transform.position.y, 0f);
}
```

- A start position variable is used to record the initial position of the user striker and a listener is added to control the position of the striker on the board before hitting.

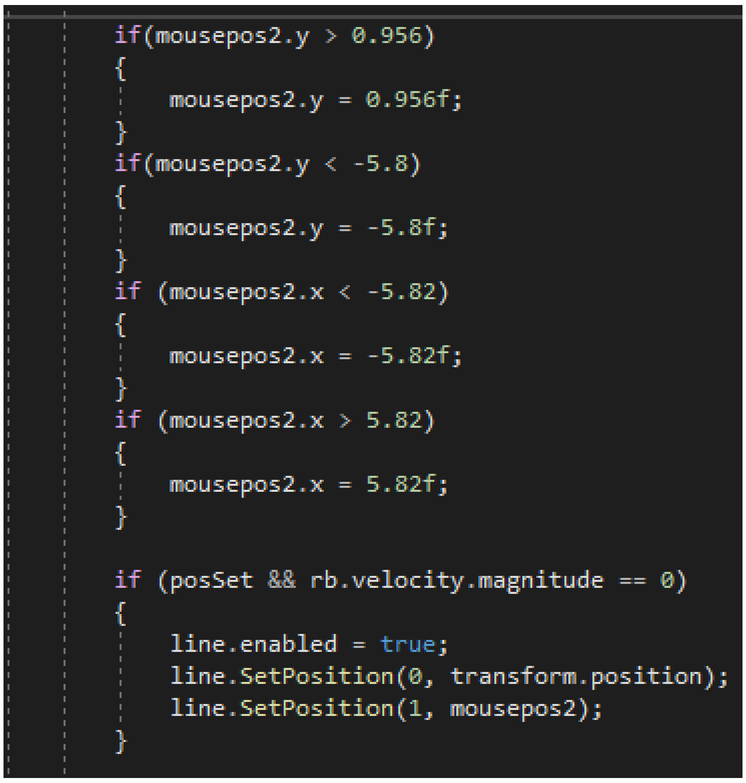
- 

```
line.enabled = false;
mousepos1 = main.ScreenToWorldPoint(Input.mousePosition);
mousepos2 = new Vector3(-mousepos1.x, -mousepos1.y - 3, mousepos1.z);

if (!hasStriked && !posSet)
    transform.position = new Vector2(strikerSlider.value, startPos.y);

RaycastHit2D hit = Physics2D.Raycast(main.ScreenToWorldPoint(Input.mousePosition), Vector2.zero);
if (hit.collider != null)
{
    if (Input.GetMouseButton(0))
    {
        if (!posSet)
            posSet = true;
    }
}
```

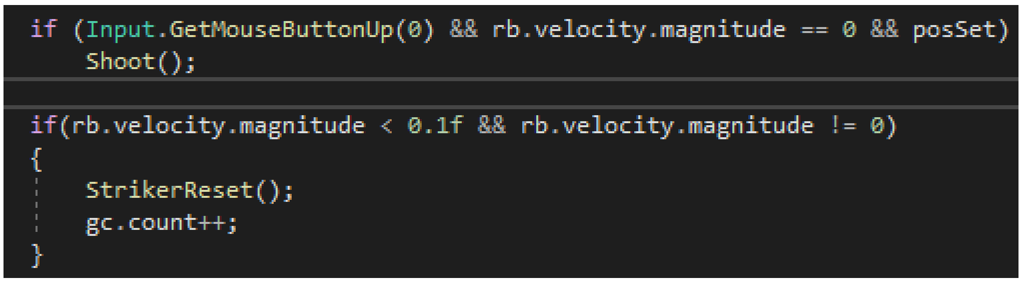
- Using the **mouse position**, a line is drawn in the opposite direction for the pull effect and shoot.
- A raycast is used to check if the striker position is set.



```
if(mousepos2.y > 0.956)
{
    mousepos2.y = 0.956f;
}
if(mousepos2.y < -5.8)
{
    mousepos2.y = -5.8f;
}
if (mousepos2.x < -5.82)
{
    mousepos2.x = -5.82f;
}
if (mousepos2.x > 5.82)
{
    mousepos2.x = 5.82f;
}

if (posSet && rb.velocity.magnitude == 0)
{
    line.enabled = true;
    line.SetPosition(0, transform.position);
    line.SetPosition(1, mousepos2);
}
```

-
- The following if conditions are used to draw the length of the line which is used to shoot the striker. A line is drawn from the striker position to the opposite side of the mouse pull.



```
if (Input.GetMouseButtonUp(0) && rb.velocity.magnitude == 0 && posSet)
    Shoot();

if(rb.velocity.magnitude < 0.1f && rb.velocity.magnitude != 0)
{
    StrikerReset();
    gc.count++;
}
```

-
- Checking for mouse release and previous striking velocity to be 0 and the position of striker before shooting.
- The shoot and striker reset method is also called as explained above to shoot the striker and reset the striker position.