Computer Graphics
(CSO-351) Project
Odd Semester
2020-21

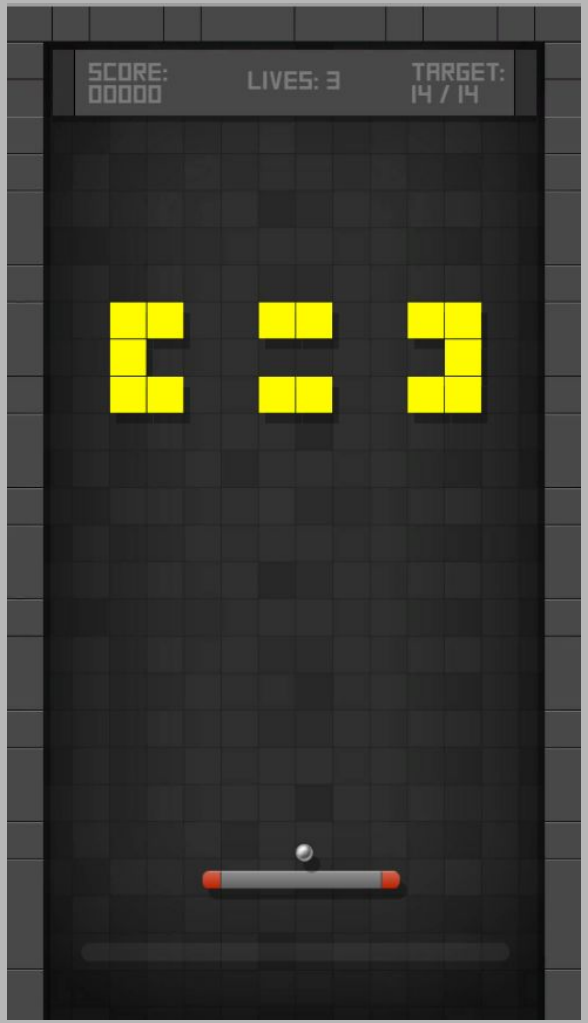
Brick Breaker Game

Under the guidance of
Dr. Pratik Chattopadhyay

Department of
Computer Science
and Engineering
IIT (BHU) Varanasi

Group No. 21
Sachin Srivastava (18075070)
Dvij Joshi (18075026)

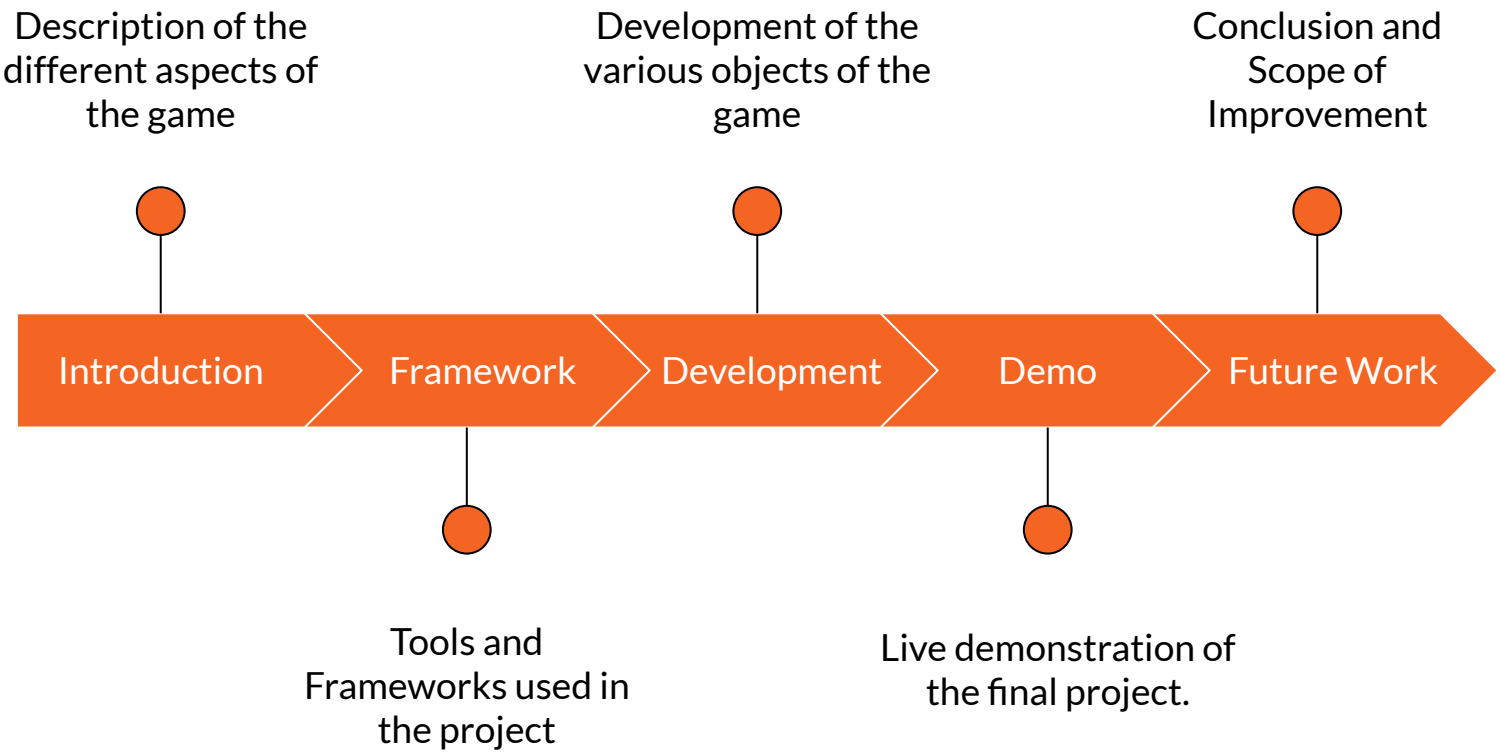
11.12.2020



Overview

The goal of the game is to break all the bricks by controlling the slider that allows the ball to bounce off from its surface and prevents the ball from falling down to the bottom. Difficulty increases as the game progresses.

Structure of Presentation


Description of the different aspects of the game
Development of the various objects of the game
Conclusion and Scope of Improvement
Introduction
Framework
Development
Demo
Future Work
Tools and Frameworks used in the project
Live demonstration of the final project.

Introduction

Introduction

The project **Brick Breaker Game** aims at:

- Providing an interface that supports single player game
- Building a classic game that tests player’s aiming and maneuvering skills
- Easy-to-understand and engaging gameplay
- Attractive Graphics
- Diverse features to keep the game interesting and enjoyable

Gameplay Description

- The task is to break as many bricks as possible without allowing the ball to fall below the level of the paddle.
- The paddle is the horizontal stick controlled by the mouse making it possible to maneuver it in horizontal direction without changing vertical position.
- The ball is an instrument or object that will break the bricks upon contact and will be directed using the paddle when it bounces off of it.

Gameplay Description

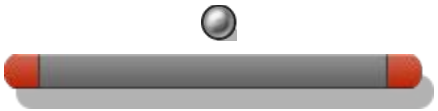
- Bricks are objects with varying power (determined by number of contacts or **HitPoints** required to destroy it) that have to be destroyed using balls.
- Several power-up collectibles have been provided in order to aid the player in destroying bricks apart from the conventional hitpoint method. These include:
    - Extend Paddle, Shooting Paddle, MultiBall, Lightning Ball
- There is also a power-down collectible added, that is to be avoided by the paddle, to make the game challenging.
    - Shrink Paddle

Paddle

- The **paddle** is a **rectangular** stick with *rounded* edges that can be moved to and fro in **horizontal** direction.
- The *motion* is **controlled** using the **cursor** of the mouse.
- Only horizontal motion is allowed and vertical position remains fixed
- It is used to direct the balls in desired **directions** depending on whether the ball strikes the **left** half or the **right** half.
- The **width** of the paddle can be **increased** or **decreased** by absorbing the *power-up* **ExtendPaddle** or *power-down* **ShrinkPaddle** respectively.

Ball

- The **Ball** is an object that is essentially used to **destroy** bricks by hitting them and directed/aimed using the **paddle** by **bouncing** off of it.
- Apart from **paddle**, it also **collides** with walls on either sides of the screen as well as top wall and **bounces** off naturally.
- The player **loses** a life when the ball **crosses** the paddle level and reaches the **bottom** of the screen.
- A **normal** ball can be **upgraded** to a **lightning** ball using a powerup that is described later.



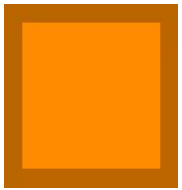
Bricks

- **Bricks** are entities/objects that need to **destroyed** using balls or other means in order to **score** points and eventually win the game.
- There are **three** types of bricks based on **HitPoints** (number of hits required to destroy):

Yellow

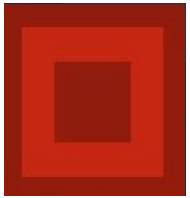
1 hit required

Orange



2 hits required

Red



3 hits required

Powerups - Extend Paddle

- The **Extend Paddle** power-up would be released upon **destruction** of bricks with a **certain probability** and should be absorbed by the paddle when it reaches it.
- It extends the **paddle width** by a certain length either for a duration of **15** seconds or until a next power-up is **absorbed** whichever occurs first.
- The **extended** length allows the player to hit the balls more **conveniently** than with normal/reduced length.

Powerups - Shooting Paddle

- The **Shooting Paddle** power-up would be released upon **destruction** of bricks with a **certain probability** and should be absorbed by the paddle when it reaches it.
- It enables the paddle to shoot **bullets** with its **ends.**
- Any brick **irrespective** of its hitpoints can be destroyed with a **bullet** in a **single** hit.
- The duration of this power-up is **10** seconds.

Powerups - MultiBall

- The **MultiBall** power-up would be released upon **destruction** of bricks with a **certain probability** and should be absorbed by the paddle when it reaches it.
- It **multiplies** the number of balls present at instance of **absorption** of the power-up by a factor of **3** by **spawning 2** extra balls for each present ball.
- All the newly **generated** balls have properties **identical** to their **parent** ball and can be used to destroy bricks at a **quicker** pace.

Powerups - Lightning Ball

- The **Lightning Ball** power-up would be released upon **destruction** of bricks with a **certain probability** and should be absorbed by the paddle when it reaches it.
- It upgrades **every** ball present at the instance of **absorption** to a **Lightning** Ball.
- A **Lightning** Ball has the capability to **penetrate** through any brick destroying it upon contact.

Powerdown - ShrinkPaddle

- Just like powerups, the power-down **Shrink Paddle** is released upon **destruction** of bricks with a certain **probability** and should be **absorbed** by the **paddle** upon reaching it.
- It **shrinks** the **width** of the paddle for a duration of **15** seconds to a reduced length making it **difficult** for the player to **direct** the ball(s) due to less **allowed** area of **contact** for the ball(s).

Lives

- A **life** is an **attempt** that a player has to play the game without **failing** .
- Number of **lives** indicate the number of **attempts** that a player has at a certain **instance** of time during the game.
- **Initially**, a player is given **3** lives.
- Whenever the ball **drops** below the **level** of the paddle and reaches the bottom of the screen, it results in **losing** a life and the tally **decreases** by **1**.
- The game is over when the tally of lives reaches 0.

—

Levels

- **Level** is a particular set of **arrangement** of bricks that need to be **destroyed** or cleared in order to **proceed** ahead in the game.
- There are total **4** levels in the game arranged in **increasing** level of difficulty.
- Player **proceeds** to the **next** level after clearing the **current** level successfully i.e. **destroying/breaking** all the bricks in the current level.
- **Difficulty** has been **increased** by making **structural** changes in the **arrangement** of the bricks making it **harder** for the player to clear that level.
- A player **wins** the game after **finishing** all the levels with the given **set** of 3 **lives** and the game is **concluded.**

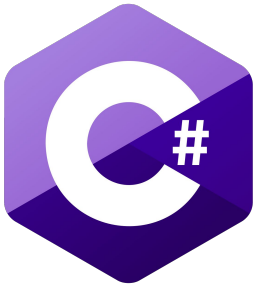
Scoring

- **Score** indicates the number of **points** scored or **earned** by the player by **breaking** the bricks at a certain **instance** during the game.
- **Tally** of score can be **increased** by breaking or **destroying** bricks.
- For **destroying** a brick, 10 points are added to the **tally** of the score.
- The **ultimate** goal of the game is to **maximize** the **tally** of the score.

Framework

Tools & Frameworks

C#



Scripting API for Game Objects

Unity

Game Engine used in the Development

VS Code

Code Editor used for Scripting

—

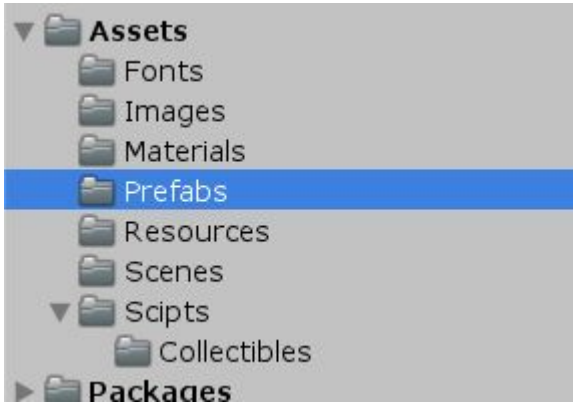
Unity

- Cross-platform game engine to develop 2D and 3D games.
- Written in C++ and uses C# as scripting API.
- Allows visualisation of the changes in real time.
- Games are divided into scenes that can be filled with Game Objects.
- Each Game Objects can be filled with different components to add functionalities.

Development

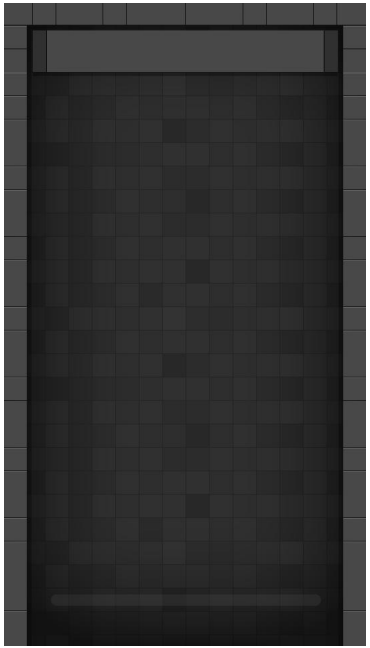
Project Structure

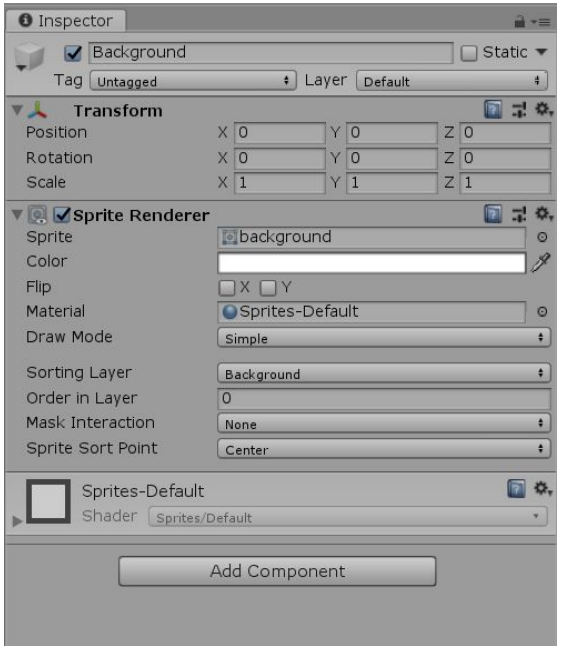
- The structure of the project is shown below.



Background

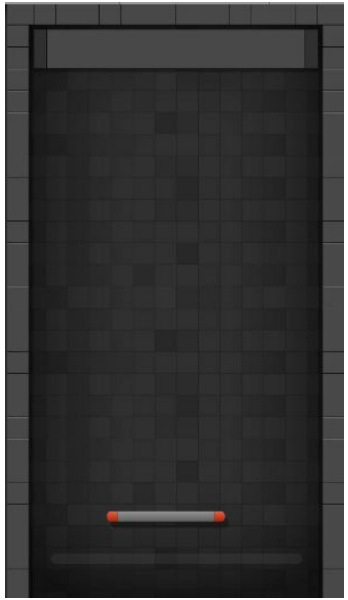
- Contains Sprite Renderer component with the image of the background.

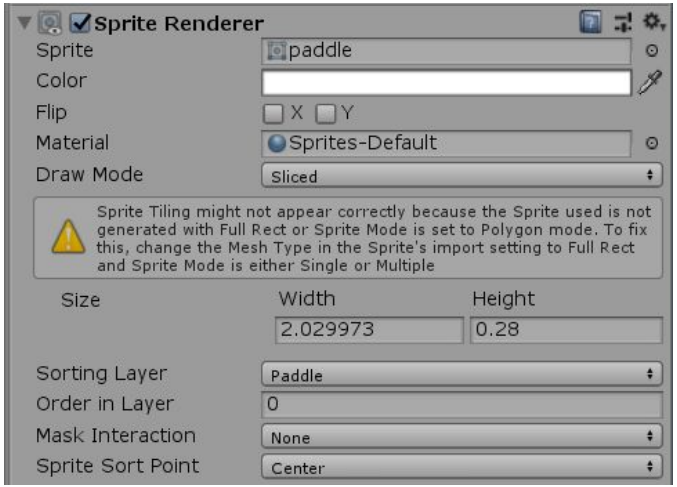


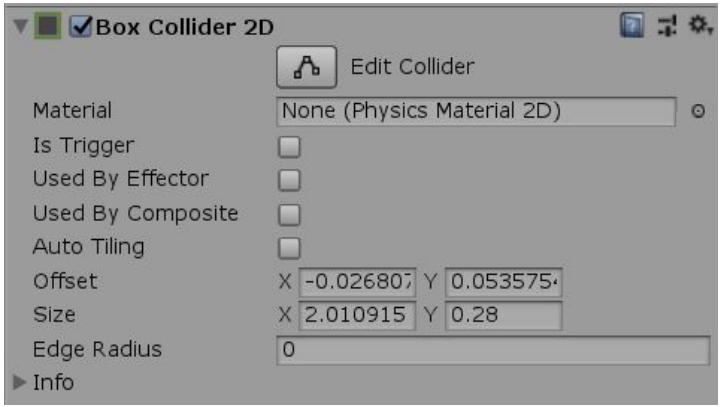


Paddle: Components

- Contains SpriteRenderer and BoxCollider2D components to add texture and implement collision physics.

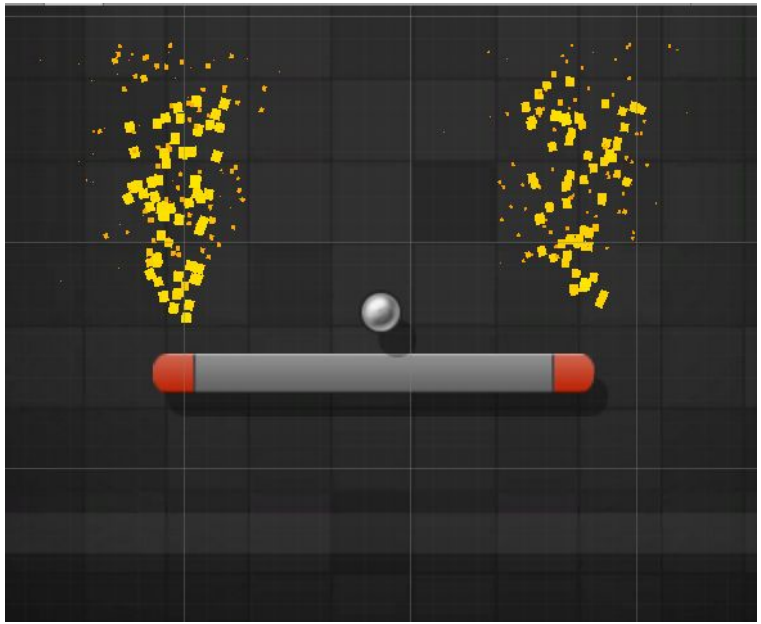


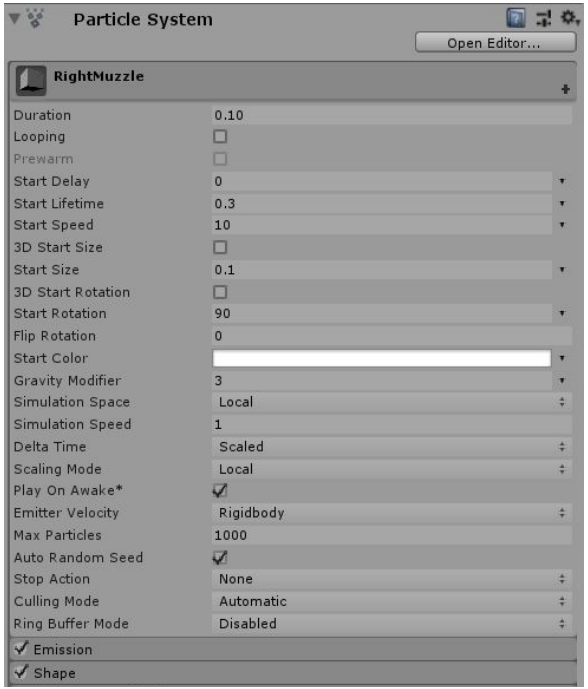




Paddle: Components

- Contains ParticleEffect component at each end for the shooting effect.





Paddle: Script - Movement

- Paddle Movement is determined by the position of Mouse.
- Paddle can not cross the Boundary Walls.
- The offsets are adjusted according to the size of the paddle.

```
private void PaddleMovement () {
    float paddleShift = (defaultPaddleWidthInPixels - (
    (defaultPaddleWidthInPixels / 2) * this.sr.size.x))
    / 2;
    float leftClamp = defaultLeftClamp - paddleShift;
    float rightClamp = defaultRightClamp + paddleShift;
    float mousePositionPixels = Mathf.Clamp (Input.
    mousePosition.x, leftClamp, rightClamp);
    float mousePositionWorldX = mainCamera.
    ScreenToWorldPoint (new Vector3
    (mousePositionPixels, 0, 0)).x;
    this.transform.position = new Vector3
    (mousePositionWorldX, paddleInitialY, 0);
}
```

Paddle: Script - Collision

- The force vector is adjusted according to the point of contact between the ball and the paddle.
- Farther the point of contact from center, greater is the force component.
- The offsets are adjusted according to the size of the paddle.

```
private void OnCollisionEnter2D (Collision2D coll) {
    if (coll.gameObject.tag == "Ball") {
        Rigidbody2D ballRb = coll.gameObject.
        GetComponent<Rigidbody2D> ();
        Vector3 hitPoint = coll.contacts[0].point;
        Vector3 paddleCenter = new Vector3 (this.
        gameObject.transform.position.x, this.
        gameObject.transform.position.y);

        ballRb.velocity = Vector2.zero;

        float difference = paddleCenter.x - hitPoint.x;

        if (hitPoint.x < paddleCenter.x) {
            ballRb.AddForce (new Vector2 (-(Mathf.Abs
            (difference * 200)), BallsManager.Instance.
            initialBallSpeed));
        } else {
            ballRb.AddForce (new Vector2 ((Mathf.Abs
            (difference * 200)), BallsManager.Instance.
            initialBallSpeed));
        }
    }
}
```

Paddle: Script - Length

- The length of the paddle is adjusted according to the width of the Sprite Renderer of the Collectible.
- Delay of a few seconds is added to allow the transformation effect.

```
public IEnumerator AnimatePaddleWidth (float width) {
    this.PaddleIsTransforming = true;
    this.StartCoroutine (ResetPaddleWidthAfterTime (this.extendShrinkDuration));

    if (width > this.sr.size.x) {
        float currentWidth = this.sr.size.x;
        while (currentWidth < width) {
            currentWidth += Time.deltaTime * 2;
            this.sr.size = new Vector2 (currentWidth, paddleHeight);
            boxCol.size = new Vector2 (currentWidth, paddleHeight);
            yield return null;
        }
    } else {
        float currentWidth = this.sr.size.x;
        while (currentWidth > width) {
            currentWidth -= Time.deltaTime * 2;
            this.sr.size = new Vector2 (currentWidth, paddleHeight);
            boxCol.size = new Vector2 (currentWidth, paddleHeight);
            yield return null;
        }
    }
    this.PaddleIsTransforming = false;
}
```

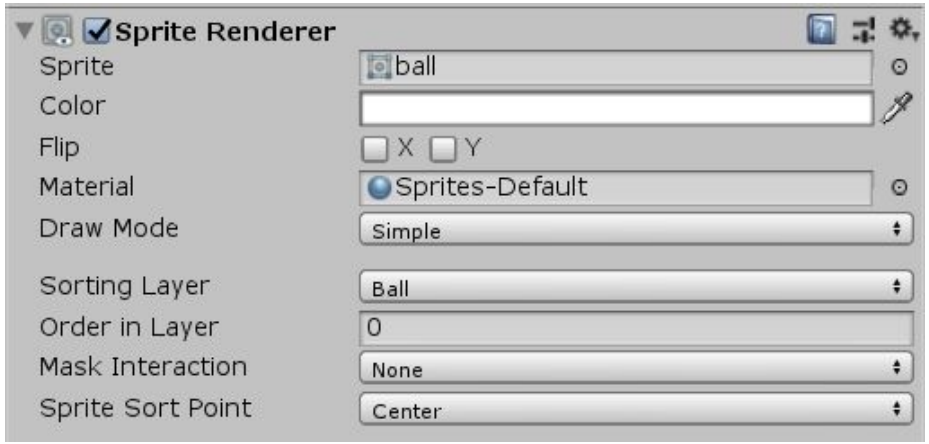
Paddle: Script - Shooting

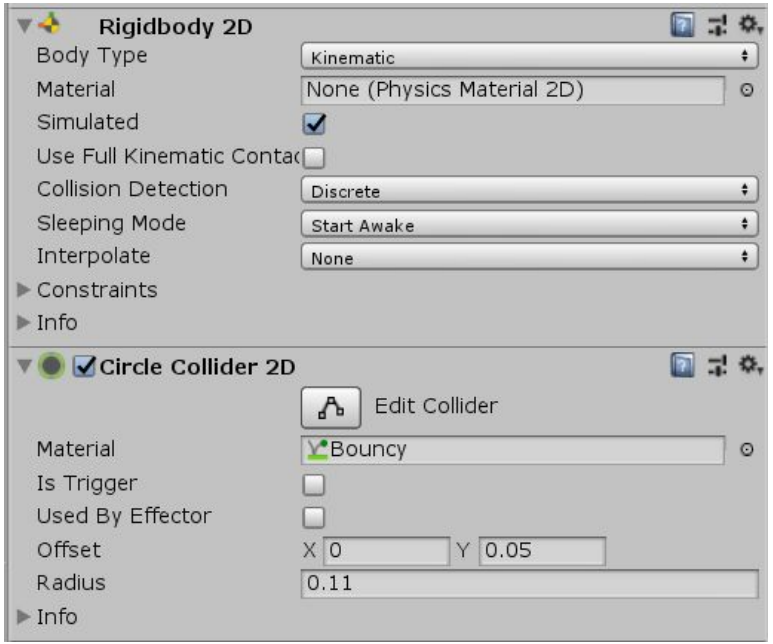
- Both the muzzles start firing bullets on collecting the ShootingPaddle powerup.
- Delay of a few milliseconds is added between successive bullets.
- After the effect of powerup vanishes both muzzles are disabled and the firing stops.

```
public IEnumerator StartShootingRoutine () {
    float fireCooldown = .5f;
    float fireCooldownLeft = 0;

    float shootingDuration = 10;
    float shootingDurationLeft = shootingDuration;

    while (shootingDurationLeft >= 0) {
        fireCooldownLeft -= Time.deltaTime;
        shootingDurationLeft -= Time.deltaTime;

        if (fireCooldownLeft <= 0) {
            this.Shoot ();
            fireCooldownLeft = fireCooldown;
        }

        yield return null;
    }

    this.PaddleIsShooting = false;
    leftMuzzle.SetActive (false);
    rightMuzzle.SetActive (false);
}
```

Ball: Components

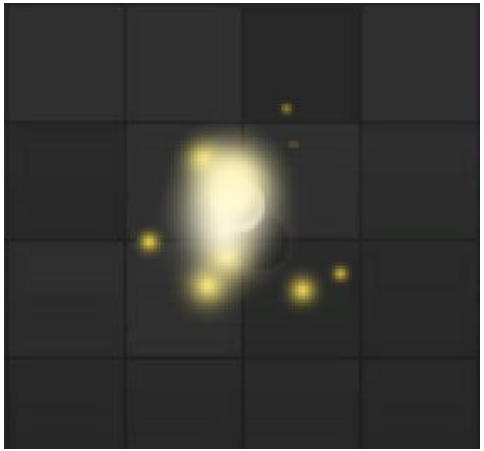
- Contains SpriteRenderer, RigidBody2D and CircleCollider2D components to add texture and implement collision physics.

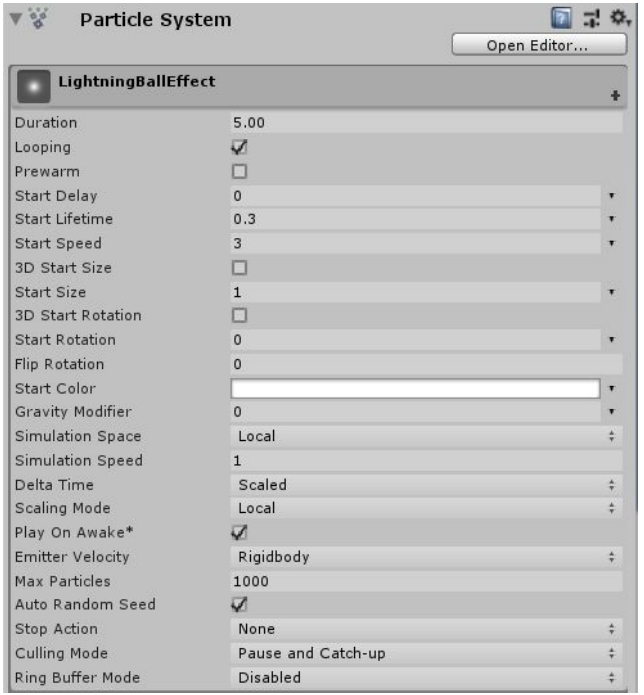




Ball: Components

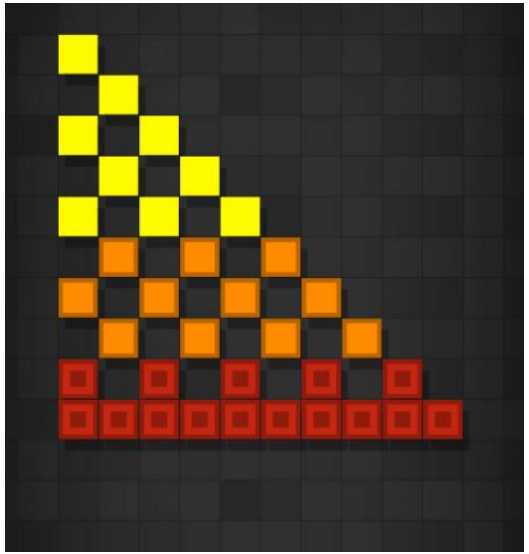
- Contains ParticleEffect component to add Lightning Effect to the ball which is activate after collecting the power-up.

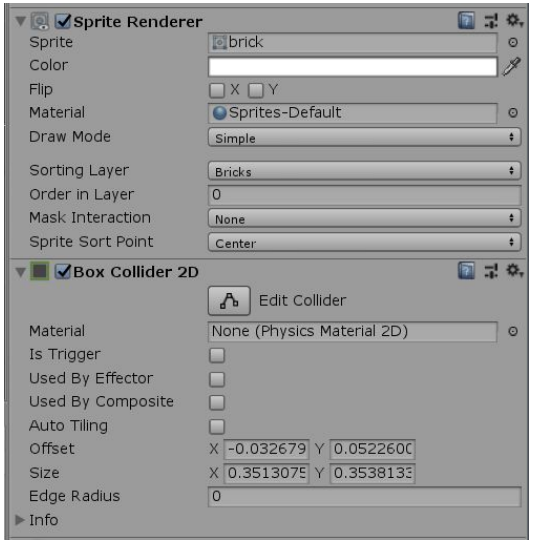




Bricks: Component

- Contains SpriteRenderer and BoxCollider2D components to add texture and implement collision physics.
- Three types of bricks are present based on the strength.





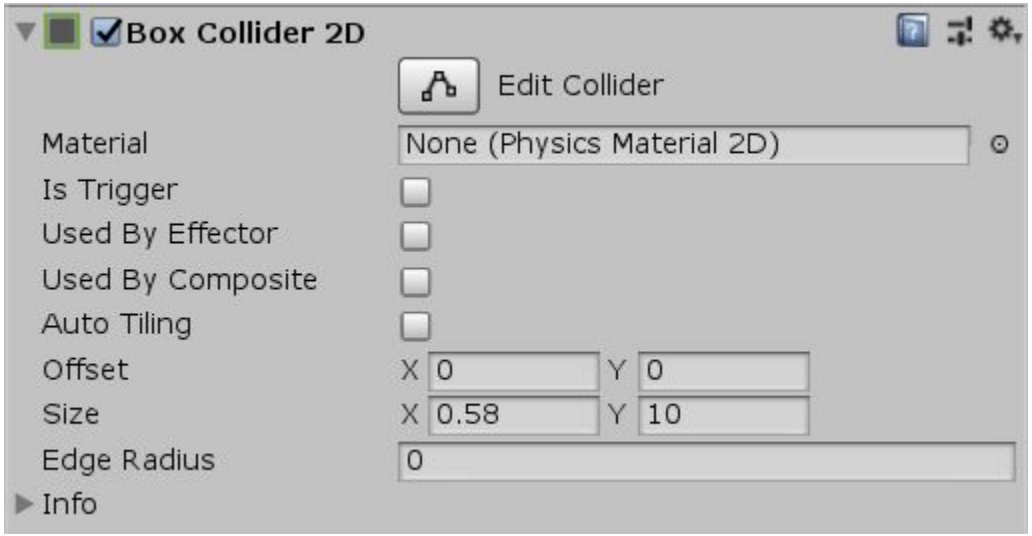
Brick: Script

- On collision with the balls or the bullets the TakeDamage function is called.
- The function decreases the HitPoints of the brick by 1.
- If the brick collides with the LightningBall, the brick is instantly destroyed.

```
private void TakeDamage (bool instantKill) {
    this.Hitpoints--;

    if (this.Hitpoints <= 0 || instantKill) {
        BricksManager.Instance.RemainingBricks.Remove (this);
        OnBrickDestruction?.Invoke (this);
        OnBrickDestroy ();
        SpawnDestroyEffect ();
        Destroy (this.gameObject);
    } else {
        this.sr.sprite = BricksManager.Instance.Sprites[this.
        Hitpoints - 1];
    }
}
```

Walls: Components

- Each Wall contains BoxCollider2D component to implements collision physics of the walls.



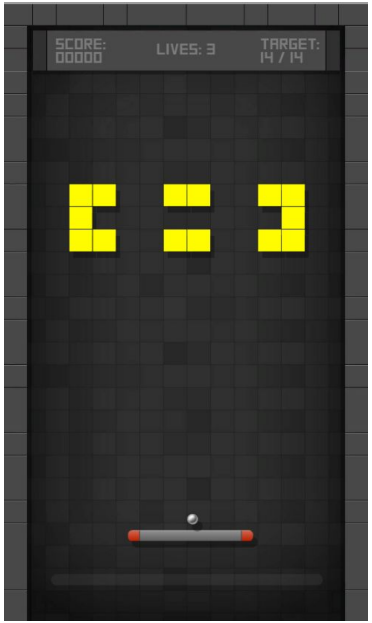
Walls: Script

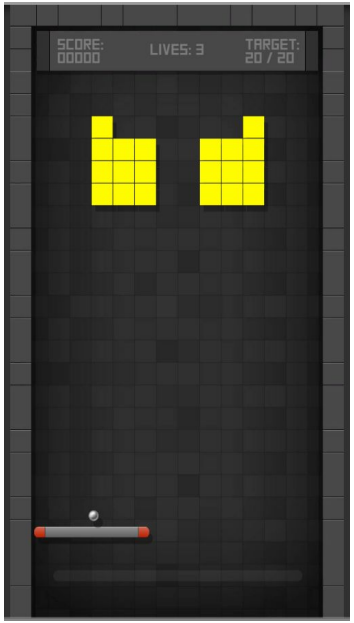
- On collision with bottom wall or the DeathWall the ball is removed from the game.

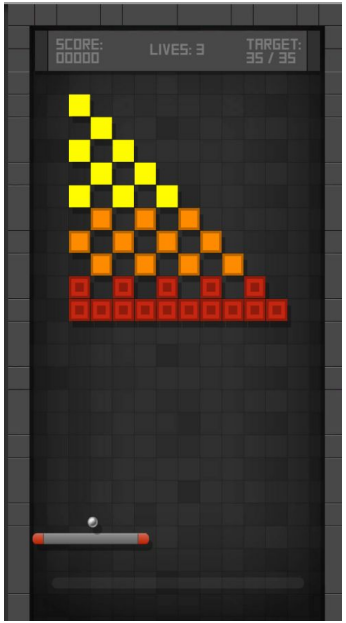
```
public class DeathWall : MonoBehaviour {
    private void OnTriggerEnter2D (Collider2D
    collision) {
        if (collision.tag == "Ball") {
            Ball ball = collision.GetComponent<Ball>
            ();
            BallsManager.Instance.Balls.Remove (ball);
            ball.Die ();
        }
    }
}
```

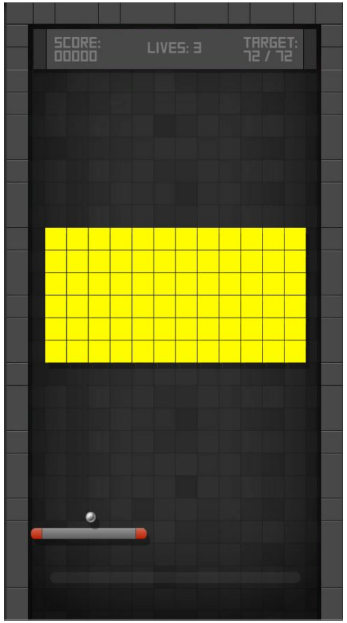
Levels

- Currently, 4 levels are designed in the game.









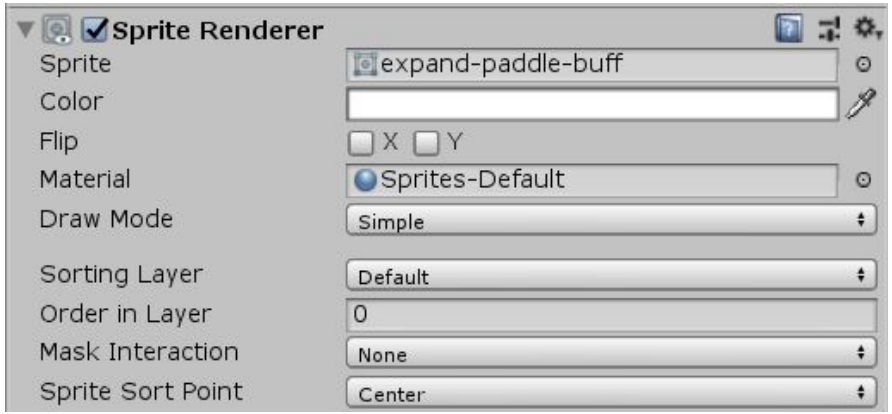
Levels: Script

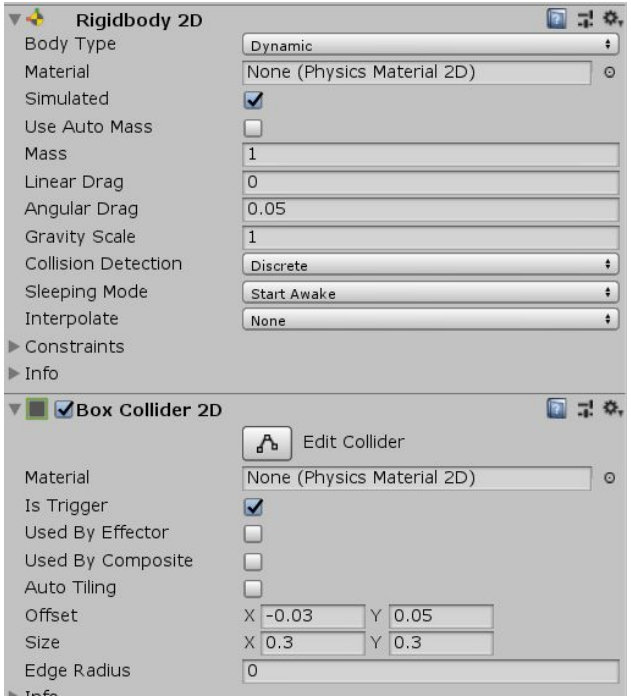
- The bricks are arranged into a pattern by reading data from levels file.

```
private List<int[, ]> LoadLevelsData () {
    TextAsset text = Resources.Load ("levels") as TextAsset;
    string[] rows = text.text.Split (new string[] { Environment.NewLine }, StringSplitOptions.RemoveEmptyEntries);
    List<int[, ]> levelsData = new List<int[, ]> ();
    int[, ] currentLevel = new int[maxRows, maxCols];
    int currentRow = 0;
    for (int row = 0; row < rows.Length; row++) {
        string line = rows[row];
        if (line.IndexOf ("--") == -1) {
            string[] bricks = line.Split (new char[] { ',' }, StringSplitOptions.RemoveEmptyEntries);
            for (int col = 0; col < bricks.Length; col++) {
                currentLevel[currentRow, col] = int.Parse (bricks[col]);
            }
            currentRow++;
        } else {
            currentRow = 0;
            levelsData.Add (currentLevel);
            currentLevel = new int[maxRows, maxCols];
        }
    }
    return levelsData;
}
```

Collectibles

- Contains SpriteRenderer, RigidBody2D and CircleCollider2D components to add texture and implement collision physics.





Live Demo

Conclusion & Future Work

Conclusion & Future Work

- The project demonstrates the interaction between various Game Objects in the game.
- The project covers different aspects of the computer graphics field.
- Sound effects can be added to make the game more appealing.
- More levels can be added in the game.
- Availability of Power-ups can also be varied as a factor to regulate difficulty.
- Other collectibles like IncreaseLife and DecreaseLife can be added to make the game more challenging.

References

References

- https://docs.unity3d.com/Manual/Unity2D.html
- https://docs.unity3d.com/Manual/ScriptingSection.html
- https://en.wikipedia.org/wiki/Breakout_(video_game)

Thank You.