



## ***Last Collector Standing***

**Student Name**

**Student Code**

**Abdeljawad Mohammad Abdeljawad**

**202125001**

**Moataz Ibrahim Jaber**

**202124938**

**Nancy Khaled Sayed**

**202124689**



Under supervision

**Prof.Dr. Shereen Taie**

**Eng. Ahmed Khaled**

**Eng. Eman Abdelaziz**

## Table of Contents

<b>1. Unity Platform Introduction .....</b>	<b>1</b>
<b>1.1 Overview of Unity platform.....</b>	<b>1</b>
<b>1.2 Key features and capabilities .....</b>	<b>1</b>
<b>1.3 Benefits of using Unity for game development.....</b>	<b>2</b>
<b>2. Project Introduction.....</b>	<b>3</b>
<b>2.1 Overview.....</b>	<b>3</b>
<b>2.2 Objectives and Goals.....</b>	<b>3</b>
<b>3. Game Story.....</b>	<b>4</b>
<b>3.1 Story Introduction.....</b>	<b>4</b>
<b>3.2 Gameplay Description.....</b>	<b>5</b>
<b>4. Assets Store.....</b>	<b>11</b>
<b>4.1 Free 8-Bit Pixel Pack .....</b>	<b>11</b>
<b>4.2 Free 2D Mega Pack .....</b>	<b>12</b>
<b>4.3 Free Running and Jumping Mascot Sheets .....</b>	<b>13</b>
<b>5. Game Scripts.....</b>	<b>14</b>
<b>6. Game Publish.....</b>	<b>23</b>

## List of Figures

• <b>Fig. 3.1</b>	Game Character, “Mascot” .....	<b>4</b>
• <b>Fig. 3.2</b>	The Game Start Point .....	<b>5</b>
• <b>Fig. 3.3</b>	“Mascot” Trying To Catch the Coins .....	<b>6</b>
• <b>Fig. 3.4</b>	"Mascot" skillfully evading an adversary. ....	<b>7</b>
• <b>Fig. 3.5</b>	"Mascot" reached the final point in the level.....	<b>8</b>
• <b>Fig. 3.6</b>	“Mascot" arrived the Next level .....	<b>9</b>
• <b>Fig. 3.7</b>	“Mascot” collided with an adversary .....	<b>10</b>
• <b>Fig. 3.8</b>	“Mascot” did a deadly jump .....	<b>10</b>
• <b>Fig. 4.1</b>	Free 8-Bit Pixel Pack .....	<b>11</b>
• <b>Fig. 4.2</b>	Free 2D Mega Pack.....	<b>12</b>
• <b>Fig. 4.3</b>	Free Running and Jumping Mascot Sheets .....	<b>13</b>
• <b>Fig. 6.1</b>	Select Build Setting from file .....	<b>23</b>
• <b>Fig. 6.2</b>	Select the targeted platforms.....	<b>23</b>
• <b>Fig. 6.3</b>	Choose the exportation file location .....	<b>24</b>
• <b>Fig. 6.4</b>	Exporting.....	<b>24</b>
• <b>Fig. 6.5</b>	The Game exported as an execution file.....	<b>24</b>
• <b>Fig. 6.6</b>	The intro theme of the Unity Game as an exported Application.....	<b>25</b>

# 1. Unity Platform Introduction

## 1.1 Overview of Unity platform

Unity is a powerful cross-platform game development engine that provides developers with a wide range of tools and features to create immersive and interactive experiences. Founded in 2004, Unity has become one of the most popular game engines in the industry, used by indie developers, AAA studios, and everything in between.

## 1.2 Key features and capabilities

- **Cross-platform Compatibility:** Unity allows developers to build games for multiple platforms including PC, consoles, mobile devices, with support for platforms like iOS, Android, Windows, macOS, Linux, PlayStation, Xbox, and more.
- **High-Fidelity Graphics:** Unity offers powerful rendering capabilities, including High-definition graphics and advanced rendering.
- **Flexible Asset Management:** Built-in physics engines for realistic interactions.
- **Physics and Simulation:** Built-in physics engines for realistic interactions.
- **Scripting with C# :** Unity uses C# as its primary scripting language, Powerful scripting for game logic and AI.
- **Editor Tools:** Tools for design, prototyping, and effects.
- **Asset Store Integration:** Unity's Asset Store provides a vast library of ready-made assets, tools, plugins, and extensions that developers can use to enhance their projects.
- **Collaboration and Version Control:** Unity Collaborate and Unity Teams allow developers to collaborate seamlessly on projects teamwork and project management.

### 1.3 Benefits of using Unity for game development

- **Ease of Use:** Unity's user-friendly interface and workflow empower developers of all levels.
- **Cost-Effective:** With free and scalable pricing options, Unity provides cost-effective solutions for developers of any budget.
- **Community Support:** Unity boasts an active community that shares knowledge and resources, offering support and collaboration opportunities.
- **Scalability:** Unity's modular architecture allows projects to scale seamlessly from prototypes to full-fledged games.
- **Market Reach:** Unity's cross-platform capabilities enable developers to target a wide audience across different devices, maximizing reach and revenue potential.

## 2. Project Introduction

### 2.1 Overview

The project is a 2D runner game designed to provide players with an exhilarating experience of running, evading enemies, collecting coins, and reaching the final point. Set in a dynamic and challenging environment, players will control a character navigating through obstacles while avoiding contact with enemy entities that threaten their progress. The game features vibrant visuals, responsive controls, and engaging gameplay mechanics that will keep players immersed and entertained.

### 2.2 Objectives and Goals

The primary objective of the game is to survive as long as possible while collecting coins and reaching the final point. To achieve this, players must:

- **Survive and Evade Enemies:** Avoid contact with enemy entities that appear throughout the game environment. Touching an enemy will result in losing the game.
- **Collect Coins:** Gather coins scattered across the level to accumulate points and enhance the player's score.
- **Navigate Obstacles:** Maneuver through various obstacles and challenges, including pits, barriers, and traps, to progress through the game.
- **Reach the Final Point:** Successfully navigate through the level to reach the final destination and complete the game.
- **Achieve High Scores:** Compete with friends and other players to achieve high scores by surviving longer, collecting more coins, and mastering the game's mechanics.

By accomplishing these objectives, players will experience a thrilling and rewarding gameplay experience that challenges their reflexes, strategy, and determination. The game aims to provide entertainment and excitement for players of all ages "+6" years old, offering replay value and opportunities for skill improvement with each play through.

## 3. Game Story

### 3.1 Story Introduction

You are an adventurer seeking fortune in the depths of a perilous dungeon. Armed with only your speed and agility, you must navigate through obstacles, evade enemies, and collect coins to reach the exit and advance to the next level.

- Your journey begins amidst towering flames and fiery pits. Dodge flames, leap over obstacles, and avoid touching the fire to survive.
- Utilize your special abilities, including super jump and extra fast running, to overcome obstacles and reach hidden areas.
- Gather coins scattered throughout the dungeon to increase your score.
- Test your reflexes and strategic thinking as you face off against cunning enemies and navigate perilous traps.



Fig.3.1 Game Character, “Mascot”.

## 3.2 Gameplay Description

- **Start of Journey:**

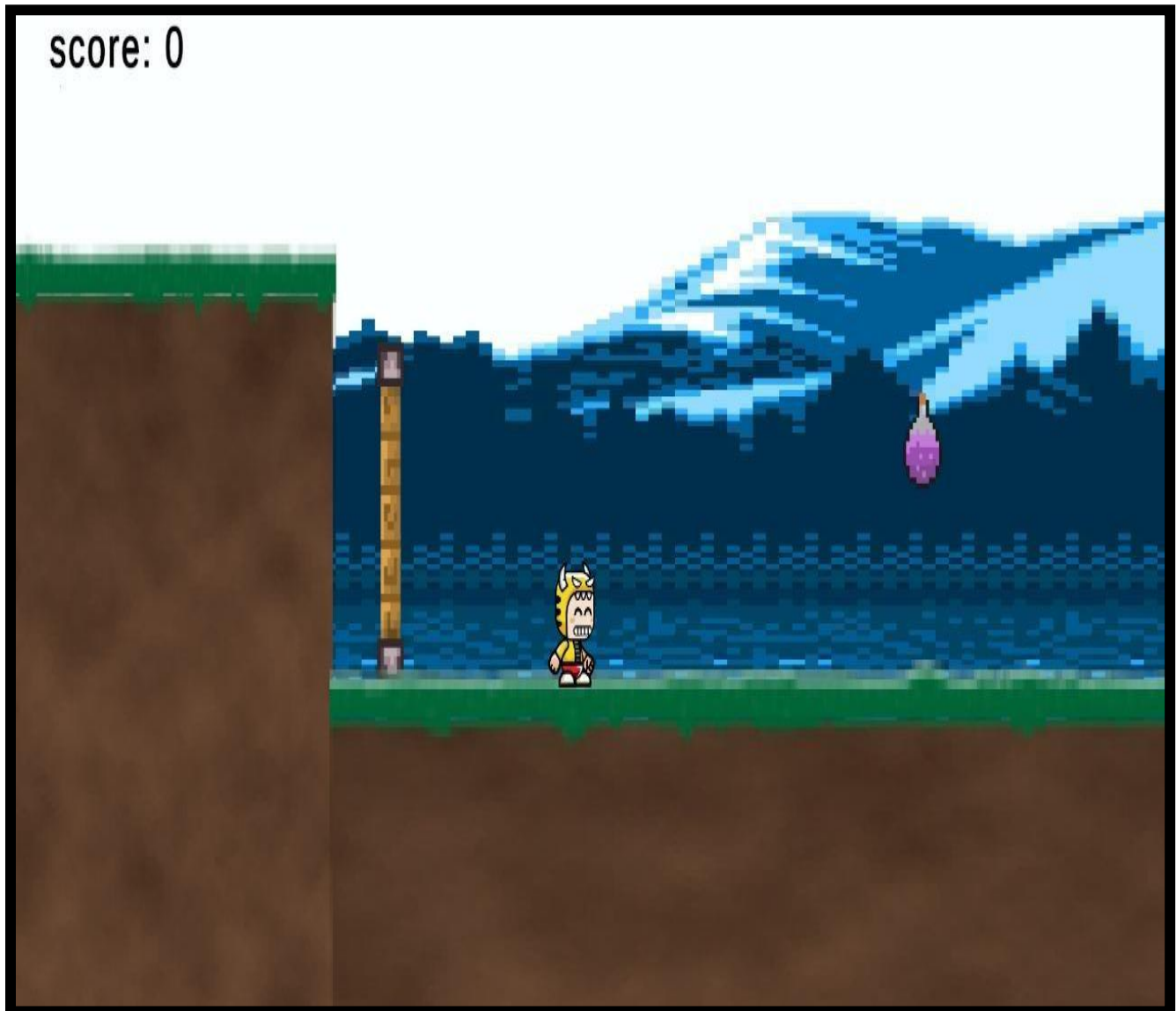


Fig. 3.2 The Game Start Point

“Mascot” begins his journey across the sky islands, eager to collect coins and avoid enemies.



- **Collecting Coins:**



Fig. 3.3 Mr. Mascot Trying To Catch the Coins.

“Mascot” Collects coins to progress through the levels and get the highest record.

- **Encounter with Adversaries:**



Fig. 3.4 "Mascot" skillfully evading an adversary.

Navigate carefully to avoid encounters with adversaries! A single touch spells instant defeat.

- **Reaching the Final Point:**



Fig. 3.5 "Mascot" reached the final point in the level.

Successfully navigate to the final point to win and step to the next level.

## Reaching the Next Level:

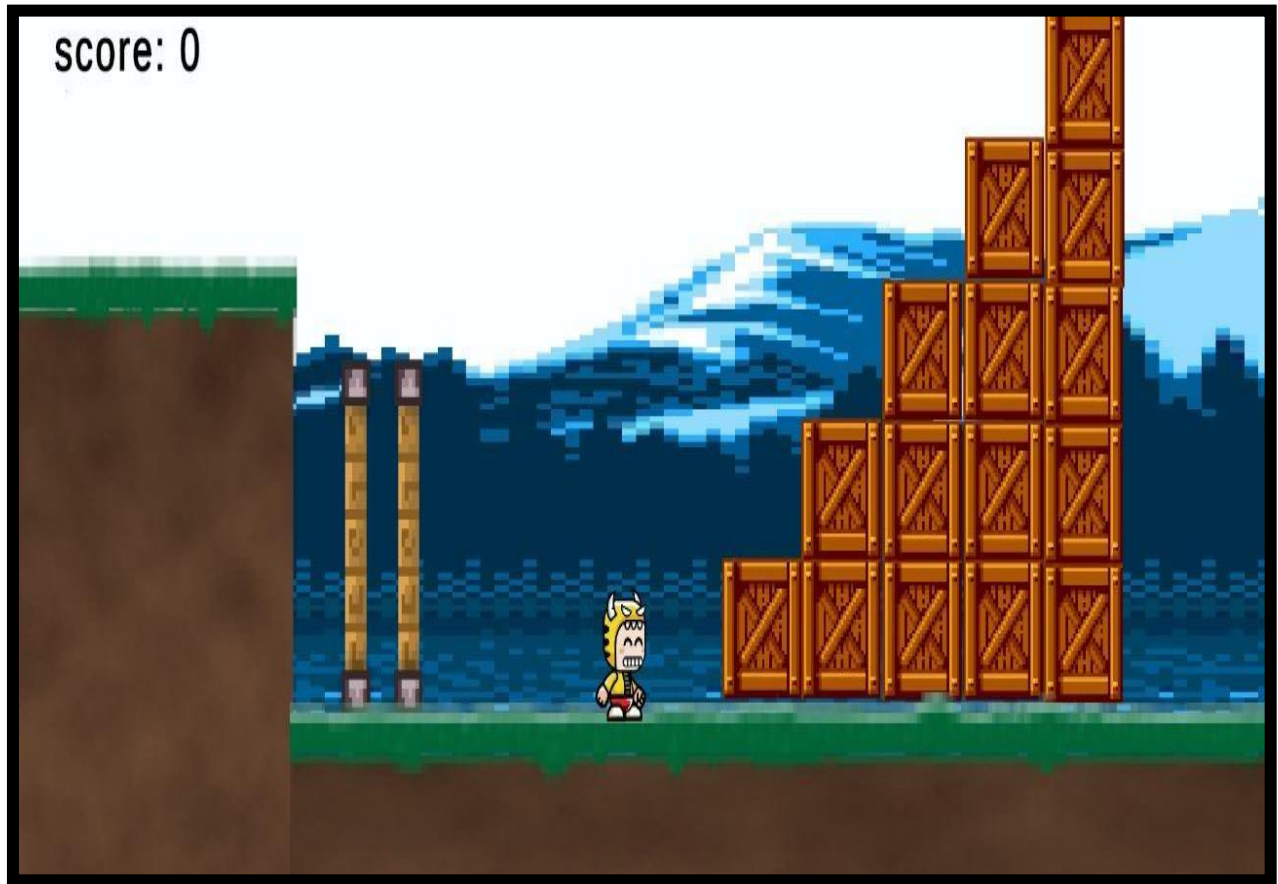


Fig. 3.6 “Mascot” arrived the Next level.

“Mascot” successfully advances to the next level, ready for new adventures.

- **Loss Scenarios:**

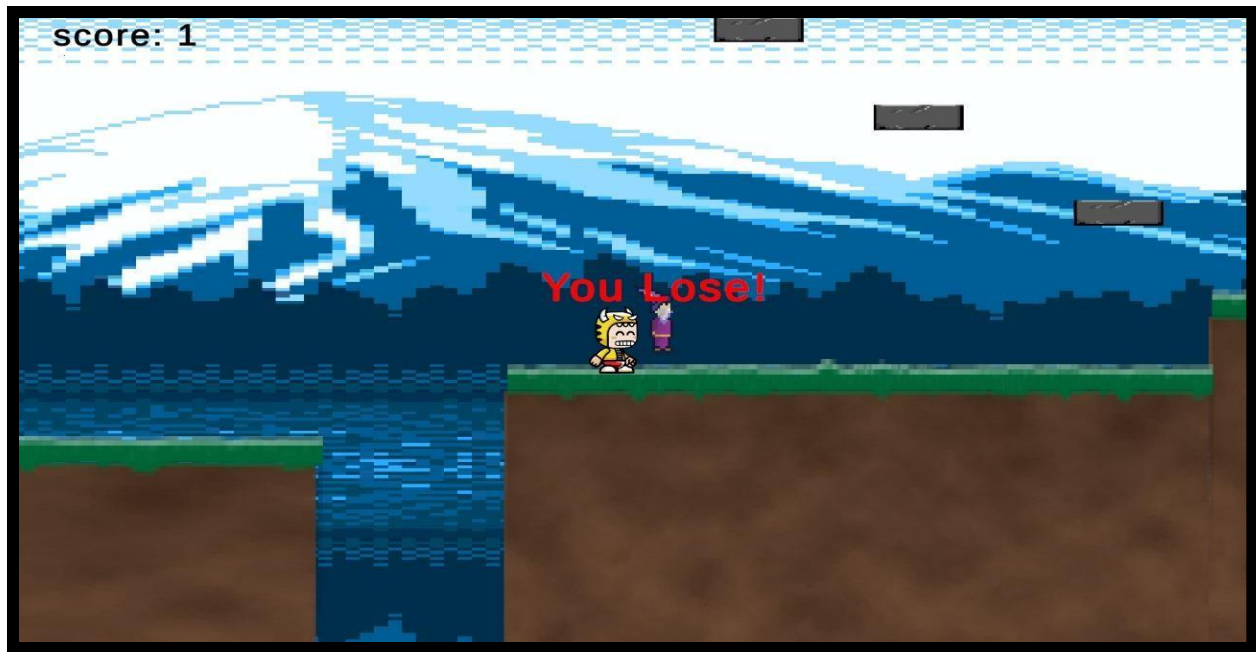


Fig. 3.7 “Mascot” collided with an adversary.

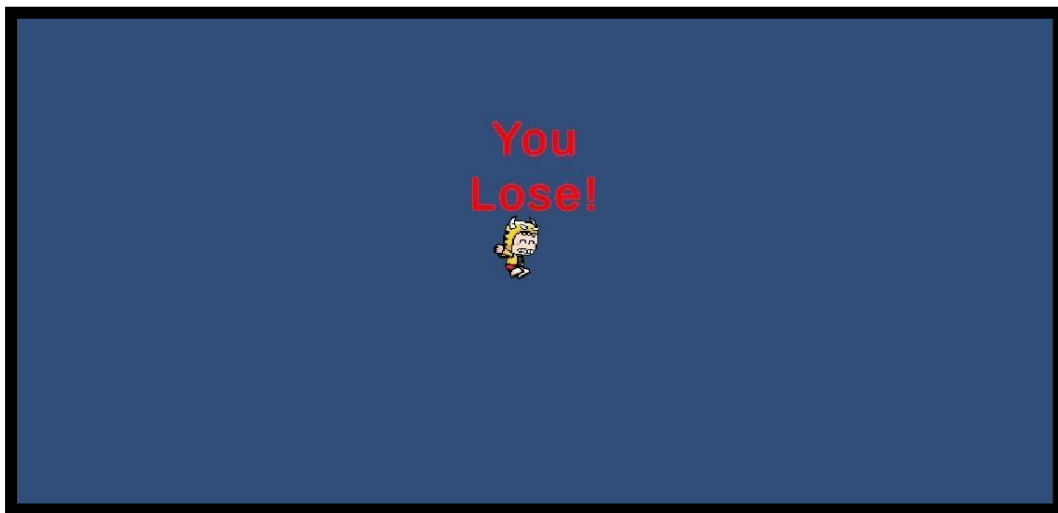


Fig. 3.8 “Mascot” did a deadly jump.

If you collide with an adversary or miss a crucial jump, it's game over. Keep trying to perfect your skills and conquer the challenges!



## 4. Assets Store

### 4.1 Free 8-Bit Pixel Pack

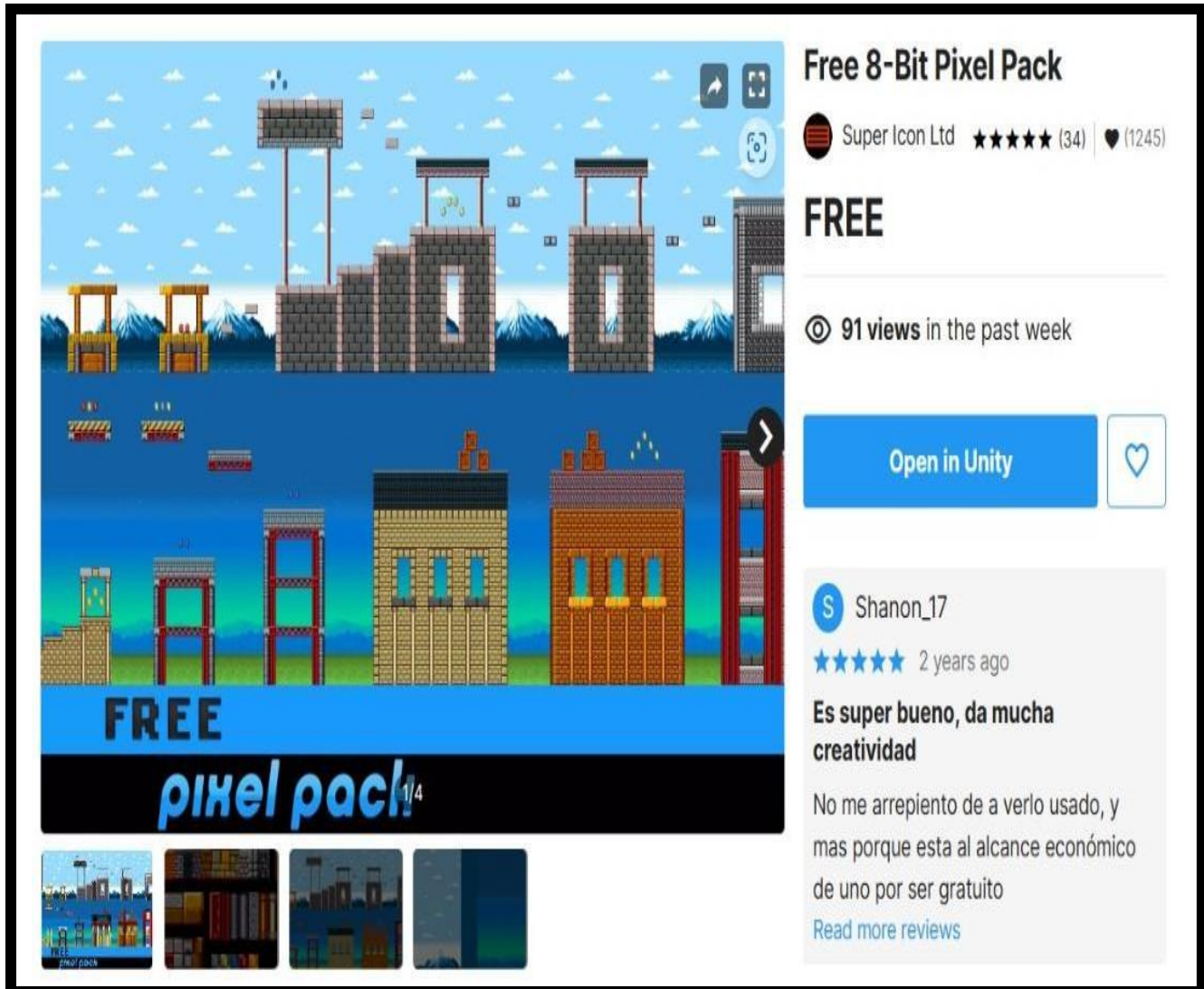


Fig. 4.1 Free 8-Bit Pixel Pack

The image shows a "Free 8-Bit Pixel Pack" from an asset store for Unity game developers, featuring a variety of pixel art assets for creating retro-style games. These assets include building designs, environmental elements, and backgrounds with skies and mountains. Offered for free, it's ideal for indie developers and hobbyists looking to enhance their games with a nostalgic 8-bit aesthetic.

## 4.2 Free 2D Mega Pack

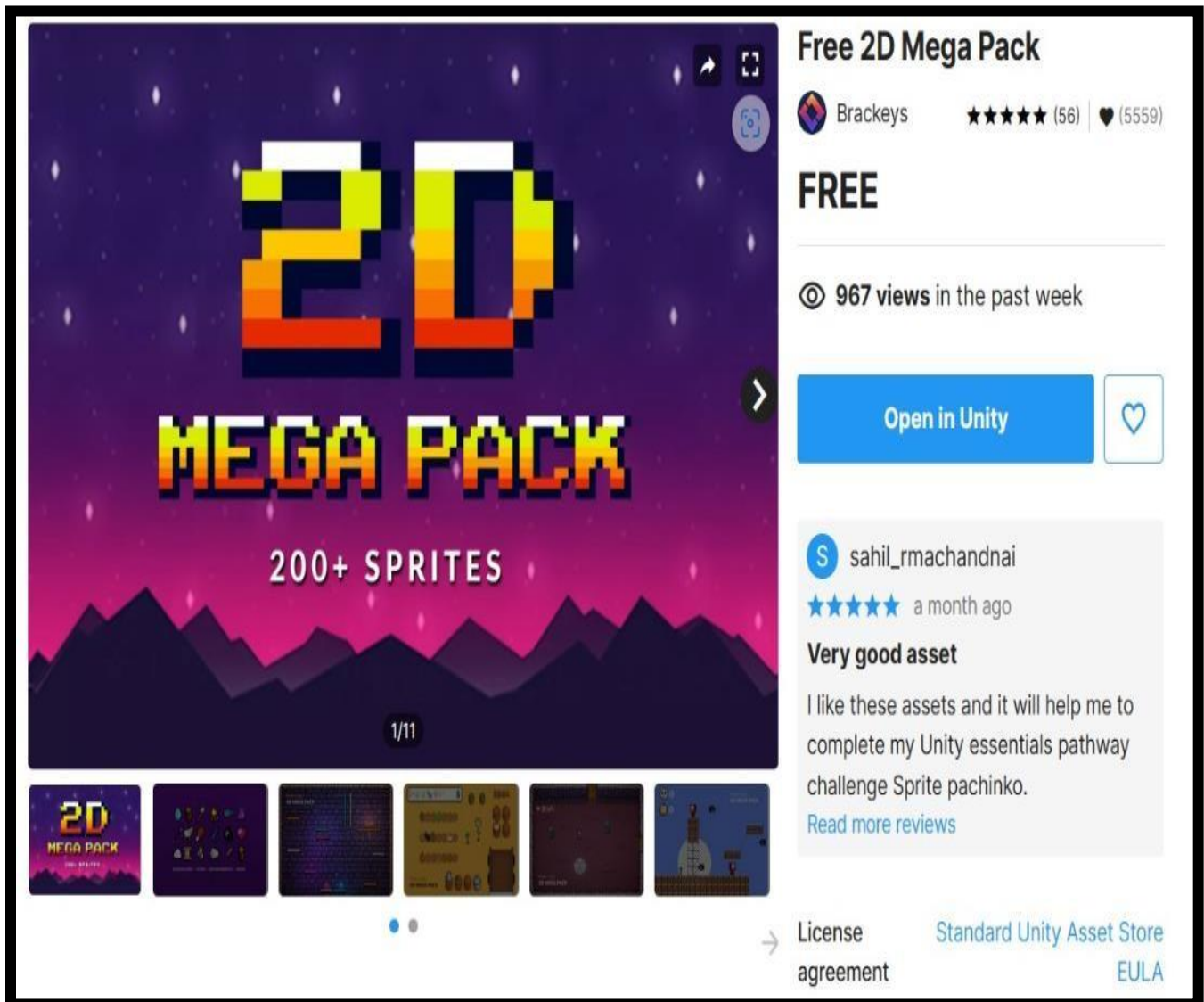


Fig. 4.2 Free 2D Mega Pack

The image features the "Free 2D Mega Pack" available on the Unity asset store, offering over 200 sprites for game development. It's designed for those looking to add a variety of 2D elements to their games, ranging from backgrounds to interface components. Highly rated by users, it's free and considered an excellent resource to aid in completing Unity learning pathways.

### 4.3 Free Running and Jumping Mascot Sheets



Fig. 4.3 Free Running and Jumping Mascot Sheets

The image shows a webpage from OpenGameArt.org displaying the "Bevoulin Free Running and Jumping Mascot Sprite Sheets." This asset pack includes a variety of transparent PNGs and GIF animations of a cartoonish mascot character, ideal for use in side-scrolling and platformer games. It's a valuable resource for creating dynamic and engaging game characters.



## 5. Game Scripts

In this section, we explore the Unity scripts for implementing various gameplay mechanics in the video game. These scripts handle everything from player movement and interaction with game objects to audio management and game state control. Each piece of code is designed to enrich the player's experience by providing responsive and engaging interactions within the game environment.

### ○ Player - Game Manage Script:

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5 using TMPro;
6
7 public class Character : MonoBehaviour
8 {
9     private Rigidbody2D rb;
10    private float dirX;
11    private float moveSpeed = 10f;
12    private float jumpForce = 1200f;
13    public TextMeshProUGUI loseText;
14    public TextMeshProUGUI scoreText;
15    public TextMeshProUGUI winText;
16    private int score = 0;
17    private bool isJumpBoosted = false;
18
19    private void Start()
20    {
21        rb = GetComponent<Rigidbody2D>();
22        loseText.gameObject.SetActive(false);
23        winText.gameObject.SetActive(false);
24        UpdateScoreText();
25    }
26
27    private void Update()
28    {
29        {
30            dirX = Input.GetAxisRaw("Horizontal") * moveSpeed;
31            rb.velocity = new Vector2(dirX, rb.velocity.y);
32
33            if (Input.GetButtonDown("Jump") && rb.velocity.y == 0)
34            {
35                float jumpVelocity = isJumpBoosted ? jumpForce * 1.3f : jumpForce;
36                rb.AddForce(Vector2.up * jumpVelocity);
37                isJumpBoosted = false;
38            }
39
40            if (transform.position.y < -40)
41            {
42                ShowLoseText();
43                StartCoroutine(RestartLevel());
44            }
45        }
46
47        private void ShowLoseText()
48        {
49            {
50                loseText.gameObject.SetActive(true);
51                loseText.text = "You Lose!";
52            }
53
54            private void OnCollisionEnter2D(Collision2D collision)
55            {
56                {
57                    if (collision.gameObject.CompareTag("co"))
58                    {
59                        ShowLoseText();
60                        StartCoroutine(RestartLevel());
61                    }
62                    else if (collision.gameObject.CompareTag("speed"))
63                    {
64                        StartCoroutine(SpeedBoost(collision.gameObject));
65                    }
66                    else if (collision.gameObject.CompareTag("jump") && !isJumpBoosted)
67                    {
68                        StartCoroutine(JumpBoost(collision.gameObject));
69                    }
70                    else if (collision.gameObject.CompareTag("coin"))
71                    {
72                        IncreaseScore();
73                        StartCoroutine(DisableObjectForSeconds(collision.gameObject, 10000));
74                    }
75                    else if (collision.gameObject.CompareTag("finish"))
76                    {
77                        ShowWinText();
78                        transform.position = new Vector2(310f, 7f);
79                    }
80                }
81            }
82        }
83    }
84 }
```

*The code in the above screenshot involves movement, jumping mechanics, and interaction with game objects for the Player.*

- **Start():** Initializes components like Rigidbody2D and sets up the initial game state. It also handles UI elements like deactivating the game over text and updating the score display.
- **Update():** Monitors user input for movement and jumping. It calculates and applies forces for movement and jumping mechanics. Additionally, it checks for conditions such as falling off the map or colliding with obstacles to trigger losing behavior.
- **ShowLoseText():** Activates the game over text and sets its content to notify the player of a loss.
- **OnCollisionEnter2D():** Detects collisions with other game objects. Based on the tags of the collided objects, it triggers various interactions such as increasing the score when collecting coins or initiating a restart when colliding with obstacles.
- **IncreaseScore():** A function called when the player collects items like coins, responsible for increasing the game score.
- **StartCoroutine():** Manages timing and sequences for operations like restarting the level or applying temporary boosts.
- **SetActive():** Controls the visibility of game objects such as game over text or victory text.
- **Rigidbody2D.velocity and Rigidbody2D.AddForce():** Directly influence the player's movement and jumping mechanics by applying forces to the Rigidbody2D component attached to the player GameObject.

```

private void IncreaseScore()
{
    score++;
    UpdateScoreText();
}

private void UpdateScoreText()
{
    if (scoreText != null)
    {
        scoreText.text = "Score: " + score.ToString();
    }
}

IEnumerator SpeedBoost(GameObject speedObject)
{
    speedObject.SetActive(false);
    moveSpeed *= 2;
    yield return new WaitForSeconds(3);
    moveSpeed /= 2;
    yield return new WaitForSeconds(7);
    speedObject.SetActive(true);
}

IEnumerator JumpBoost(GameObject jumpObject)
{
    jumpObject.SetActive(false);
    isJumpBoosted = true;
    float originalJumpForce = jumpForce;
    jumpForce *= 1.3f;
    yield return new WaitForSeconds(3);
    jumpForce = originalJumpForce;
    isJumpBoosted = false;
    yield return new WaitForSeconds(7);
    jumpObject.SetActive(true);
}

private void ShowWinText()
{
    if (winText != null)
    {
        winText.gameObject.SetActive(true);
        winText.text = "You Win!";
    }
}

IEnumerator DisableObjectForSeconds(GameObject obj, float seconds)
{
    obj.SetActive(false);
    yield return new WaitForSeconds(seconds);
    obj.SetActive(true);
}

IEnumerator RestartLevel()
{
    yield return new WaitForSeconds(2);
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
}

```

*These components are integral to enriching the gaming experience by providing dynamic interactions and effects that respond to the player's actions.*

- **IncreaseScore():** Adds one to the game score and updates the score display via UpdateScoreText().
- **UpdateScoreText():** Modifies the UI text element showing the score, ensuring scoreText exists. The text becomes "Score: " followed by the current score.
- **SpeedBoost(IEnumerator):** Temporarily doubles the moveSpeed variable for 3 seconds before reverting it to its original value. The speed-boosting GameObject deactivates during this period and reactivates after 7 seconds.
- **JumpBoost(IEnumerator):** Temporarily increases the jumpForce by 30% for 3 seconds to enhance jumping height, then resets it. Similar to SpeedBoost, the triggering GameObject deactivates and reactivates after 7 seconds.
- **ShowWinText():** Activates the win text and customizes its content if winText exists.
- **DisableObjectForSeconds(IEnumerator):** Temporarily deactivates a GameObject for a specified duration, then reactivates it.
- **RestartLevel(IEnumerator):** Delays for 2 seconds before reloading the current scene, effectively restarting the level. These methods manage scoring, temporary power-ups, and game state transitions smoothly, leveraging Unity's coroutine system for seamless execution.

## ○ Sounds Scripts:

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class CharacterAudio : MonoBehaviour
5 {
6     public AudioClip moveSound;
7     public AudioClip speedHitSound;
8     public AudioClip jumpHitSound;
9     private bool canPlayJumpSound = false;
10    public AudioClip coinHitSound;
11    public AudioClip finishHitSound;
12    public AudioClip coHitSound;
13    public AudioSource audioSource;
14    public AudioSource speedHitAudioSource;
15    public AudioSource jumpHitAudioSource;
16    public AudioSource coinHitAudioSource;
17    public AudioSource finishHitAudioSource;
18    public AudioSource coHitAudioSource;
19
20    public AudioClip newJumpSound;
21    public AudioSource newJumpAudioSource;
22
23    public AudioClip background;
24    public AudioSource backgroundSource;
25
26
27    private Rigidbody2D rb;
28    private float dirX;
29
30    private void Start()
31    {
32        rb = GetComponent<Rigidbody2D>();
33    }
34
35    private void Update()
36    {
37        if (canPlayJumpSound && Input.GetKeyDown(KeyCode.Space))
38        {
39            PlayJumpHitSound();
40            canPlayJumpSound = false;
41        }
42        else if (Input.GetButtonDown("Jump") && rb.velocity.y == 0)
43        {
44            PlayNewJumpSound();
45        }
46    }
47
48    dirX = Input.GetAxisRaw("Horizontal");
49
50    if (dirX != 0 && rb.velocity.y == 0)
51    {
52        PlayMoveSound();
53    }
54 }
55
56 private void PlayMoveSound()
57 {
58     if (audioSource != null && moveSound != null)
59     {
60         if (!audioSource.isPlaying)
61         {
62             audioSource.clip = moveSound;
63             audioSource.pitch = 1f;
64             audioSource.Play();
65         }
66     }
67 }
68
69 private IEnumerator SpeedBoostSoundCoroutine()
70 {
71     if (speedHitAudioSource != null && speedHitSound != null)
72     {
73         speedHitAudioSource.pitch *= 2;
74
75         speedHitAudioSource.clip = speedHitSound;
76         speedHitAudioSource.Play();
77
78         yield return new WaitForSeconds(3);
79
80         speedHitAudioSource.pitch /= 2;
81     }
82 }
83 }
```

*These screenshots provide a look at parts of a Unity script that handles audioeffects and interactions for a video game.*

**Start():** Initializes components, notably the Rigidbody2D component.

**Update():** Monitors conditions for triggering specific sounds. These include:Checking for the ability to play a jump sound when the space key is pressed.

Initiating a new jump sound if the jump button is pressed and the character is grounded.

**PlayMoveSound():**Triggers a movement sound if the character is horizontally mobile and grounded.

**SpeedBoostSoundCoroutine IEnumerator) :**A coroutine orchestrating the speed boost sound effect by temporarily increasing the pitch, waiting for 3 seconds, and then resetting the pitch. This coroutine is activated from the PlaySpeedHitSound()method.

## ○ Animation script

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class animation : MonoBehaviour
6  {
7      private Rigidbody2D rb;
8      private Animator anim;
9      private float dirX;
10     private bool isFalling = false;
11     private bool facingRight = true;
12
13
14
15     void Start()
16     {
17         rb = GetComponent<Rigidbody2D>();
18         anim = GetComponent<Animator>();
19     }
20
21
22     void Update()
23     {
24         dirX = Input.GetAxisRaw("Horizontal") * 10f;
25
26
27         anim.SetBool("isRunning", Mathf.Abs(dirX) > 0 && rb.velocity.y == 0);
28         anim.SetBool("isJumping", rb.velocity.y > 0);
29         anim.SetBool("isFalling", isFalling);
30
31     }
32
33     private void FixedUpdate()
34     {
35
36         if (rb.velocity.y < 0 )
37         {
38             isFalling = true;
39         }
40         else
41         {
42             isFalling = false;
43         }
44     }
45
46     private void LateUpdate()
47     {
48         if ((dirX > 0 && !facingRight) || (dirX < 0 && facingRight))
49             FlipCharacter();
50     }
51
52     private void FlipCharacter()
53     {
54         facingRight = !facingRight;
55         GetComponent<SpriteRenderer>().flipX = !GetComponent<SpriteRenderer>().flipX;
56     }
57 }
58 a
```

### **Start():**

- Initializes essential components for the animation logic :
- Retrieves the Rigidbody2D component (rb) which is used to control the physics of the character.
- Obtains the Animator component (anim) that manages the animation states based on the gameplay.

### **Update():**

- Monitors player inputs and updates animation states accordingly:
- Captures the horizontal input to determine the character's direction and speed. If there's horizontal movement and the character is not in the air (i.e., rb.velocity.y is zero), it triggers the walking or running animations.
- If the space key is pressed and the character is able to jump (canPlayJumpSound is true), it plays the jump animation and resets the ability to trigger the jump sound until conditions are met again.
- Checks for a jump input through a different control (e.g., a "Jump" button) while the character is grounded to trigger an alternative jump animation.

### **FixedUpdate():**

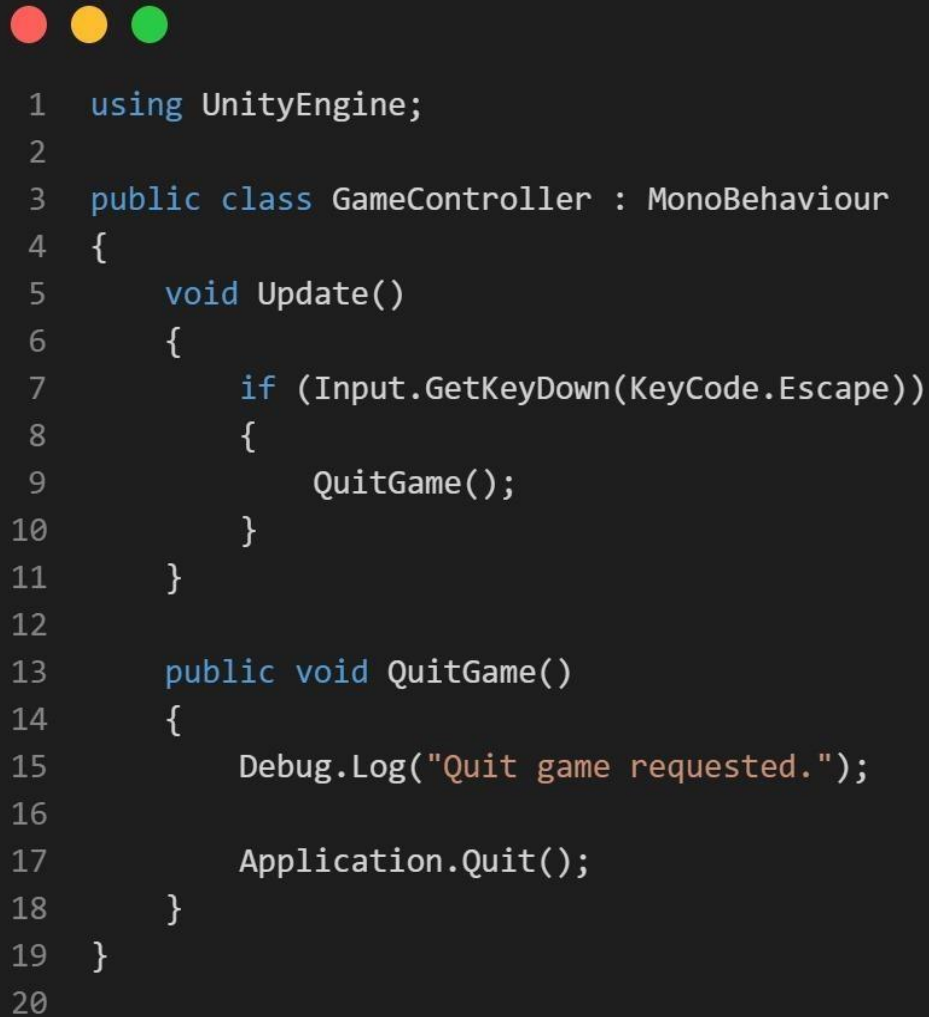
- Used for calculations that are independent of the frame rate, ensuring smooth physics interactions:
- Regularly checks if the character is falling by evaluating the vertical velocity. If it is less than zero, the character is considered to be falling, which can trigger a falling animation.
- **LateUpdate():**
- Adjusts the sprite orientation based on the character's direction of movement:
- If the character moves left while facing right, or moves right while facing left, the FlipCharacter() function is called to invert the sprite's direction, ensuring the character always faces the direction of movement.

### **FlipCharacter():**

- Handles the flipping of the character's sprite:
- Toggles the facingRight boolean to reflect the new facing direction.
- Modifies the flipX property of the SpriteRenderer to visually mirror the sprite, making the character face the correct direction based on the latest input



## ○ Quit Script:



```
1  using UnityEngine;
2
3  public class GameController : MonoBehaviour
4  {
5      void Update()
6      {
7          if (Input.GetKeyDown(KeyCode.Escape))
8          {
9              QuitGame();
10         }
11     }
12
13     public void QuitGame()
14     {
15         Debug.Log("Quit game requested.");
16
17         Application.Quit();
18     }
19 }
20
```

*The script contains a method called `Update()` that listens for the "Escape" key press. If the Escape key is pressed, it triggers the `QuitGame()` method, which logs a message indicating a quit request and then closes the application using `Application.Quit()`. This script is used to add basic game exit functionality.*

## 6. Game Publish

Exporting a game from Unity involves selecting the target platform, configuring player settings, initiating the build process, and testing the build. Once complete, the game can be distributed through app stores, distribution platforms, or your website. Unity streamlines this process, making it easy for developers to export their games and reach a wide audience.

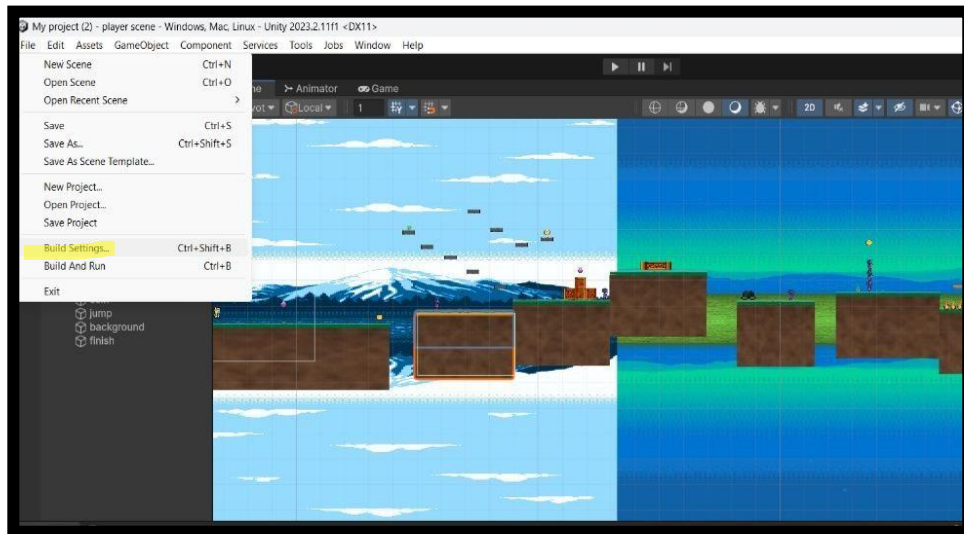


Fig. 6.1 Select Build Setting from file.

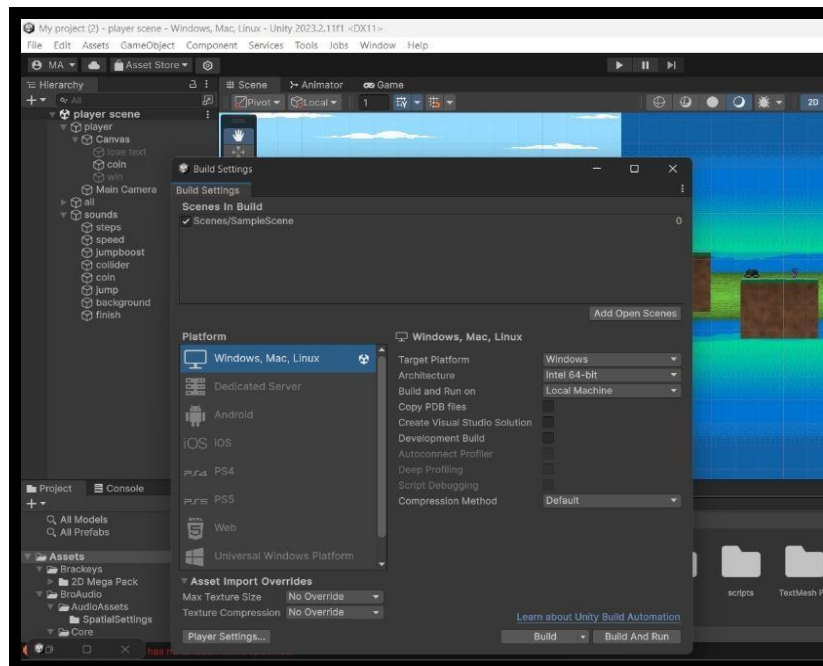


Fig. 6.2 Select the targeted platforms.

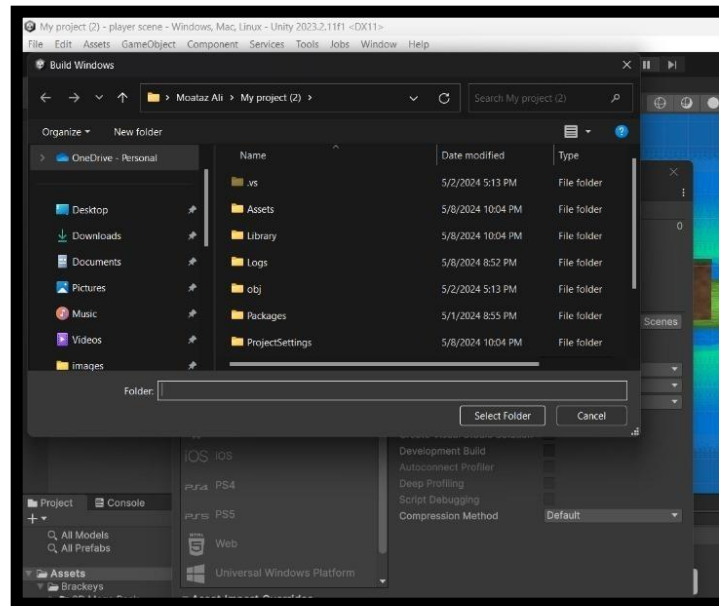


Fig. 6.3 Choose the exportation file location.

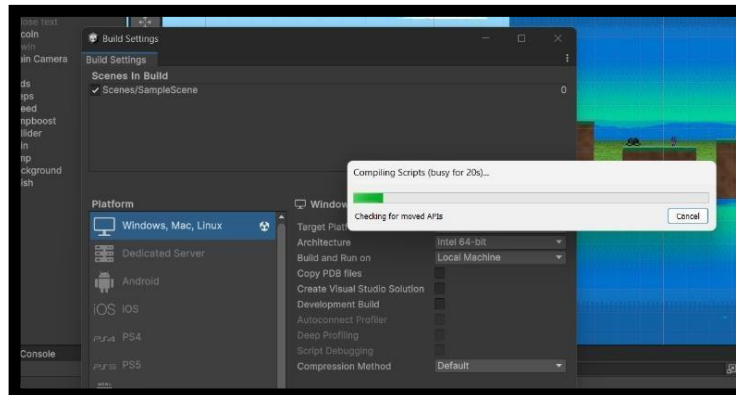


Fig. 6.4 Exporting...

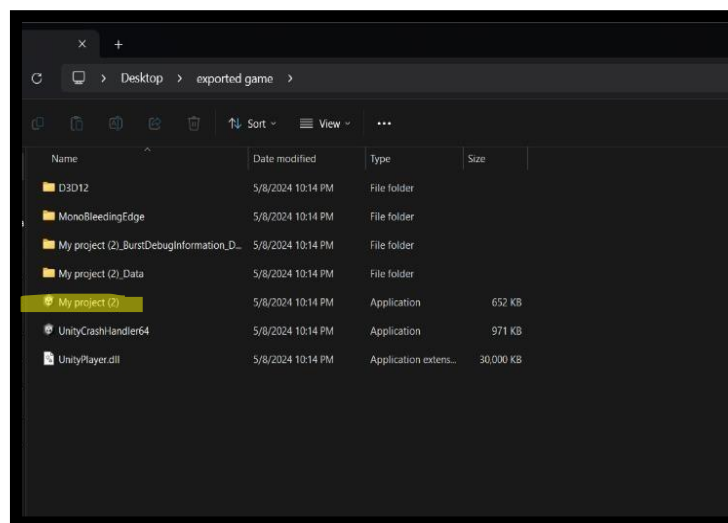


Fig. 6.5 The Game exported as an execution file.



Fig. 6.6 The intro theme of the Unity Game as an exported Application.