“mr. Mini” the Game

Project Report

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

Submitted By:

Suresh

Roll no: 63541

Under the Guidance

of

Er. Akshay Bhardwaj

(Assistant Professor)

University Institute of Information Technology

Himachal Pradesh University Summer Hill, Shimla 171005

CERTIFICATE

This is to certify that this project report on **“mr. Mini”** game is submitted by **Suresh** Roll Number 63541, who carried out the project work under my supervision. I approve this project for submission of the Bachelor of Technology in the Computer Science & Engineering and Information Technology, University Institute of Information Technology affiliated to HPU, Summer Hill, Shimla (H.P). 171005

ACKNOWLEDGEMENT

It gives me immense pleasure to express my deepest sense of gratitude and sincere thanks to my highly respected and esteemed Project Coordinator Er. Akshay Bhardwaj **(Asst. Professor) UIIT, HPU Summer Hill,** for their valuable guidance, encouragement, and help for completing this work. Their useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

I also wish to express my indebtedness to my parents as well as my family member whose blessings and support always helped me to face the challenges ahead. At the end, I would like to express my sincere thanks to all my friends and others who helped me directly or indirectly during this project work.

Index

1. Introduction

“mr. Mini” is a 2D Game Created in “Unity®” the game Engine. It is an endless runner game something like “Chrome run” game in the Google chrome browser. This game is created to explore the various concepts of game development, various concepts like GDD (Game Design Document), life cycle of game development, programming perspectives, graphics development for games, logic development, level design game.

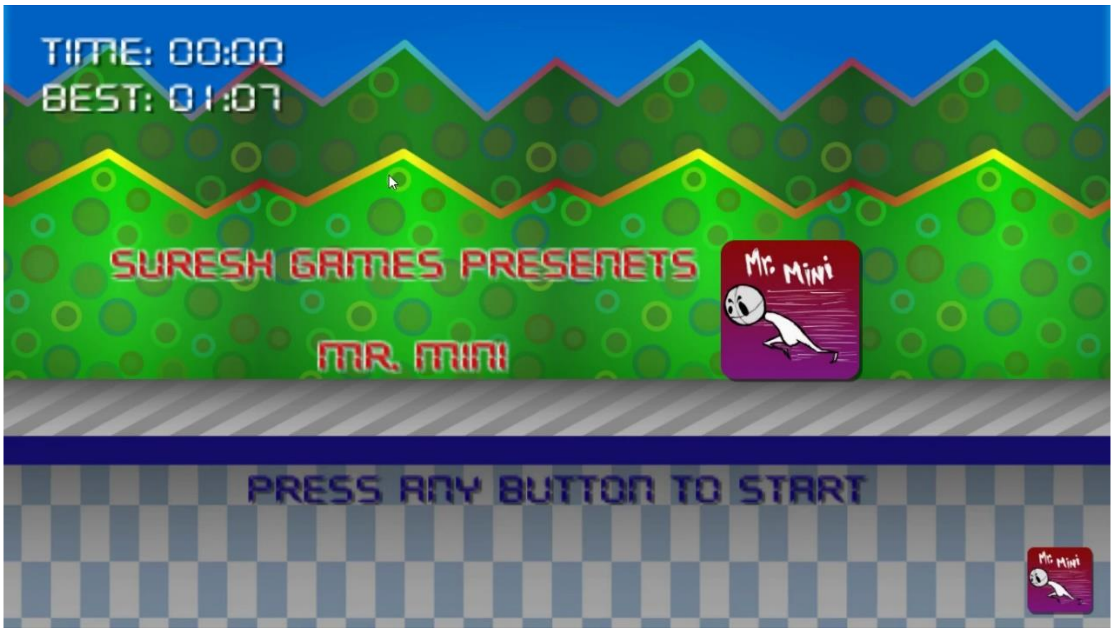


Fig1. Screenshot showing game start screen.

2. Coming up with Ideas

Since the game should be extremely simple to design but it must have some level of challenge as well to explore the various aspects of game design being my first game. So, I skimmed through some diverse games gameplay and then I came to a conclusion that I shall design an endless runner game which is simple to design and as we grasp the basics of game design endless runner games can be designed to very interesting level. Some best examples of endless runner games are: **Subway Surfer, Temple Run, Sonic Dash, Super Mario Run** etc.

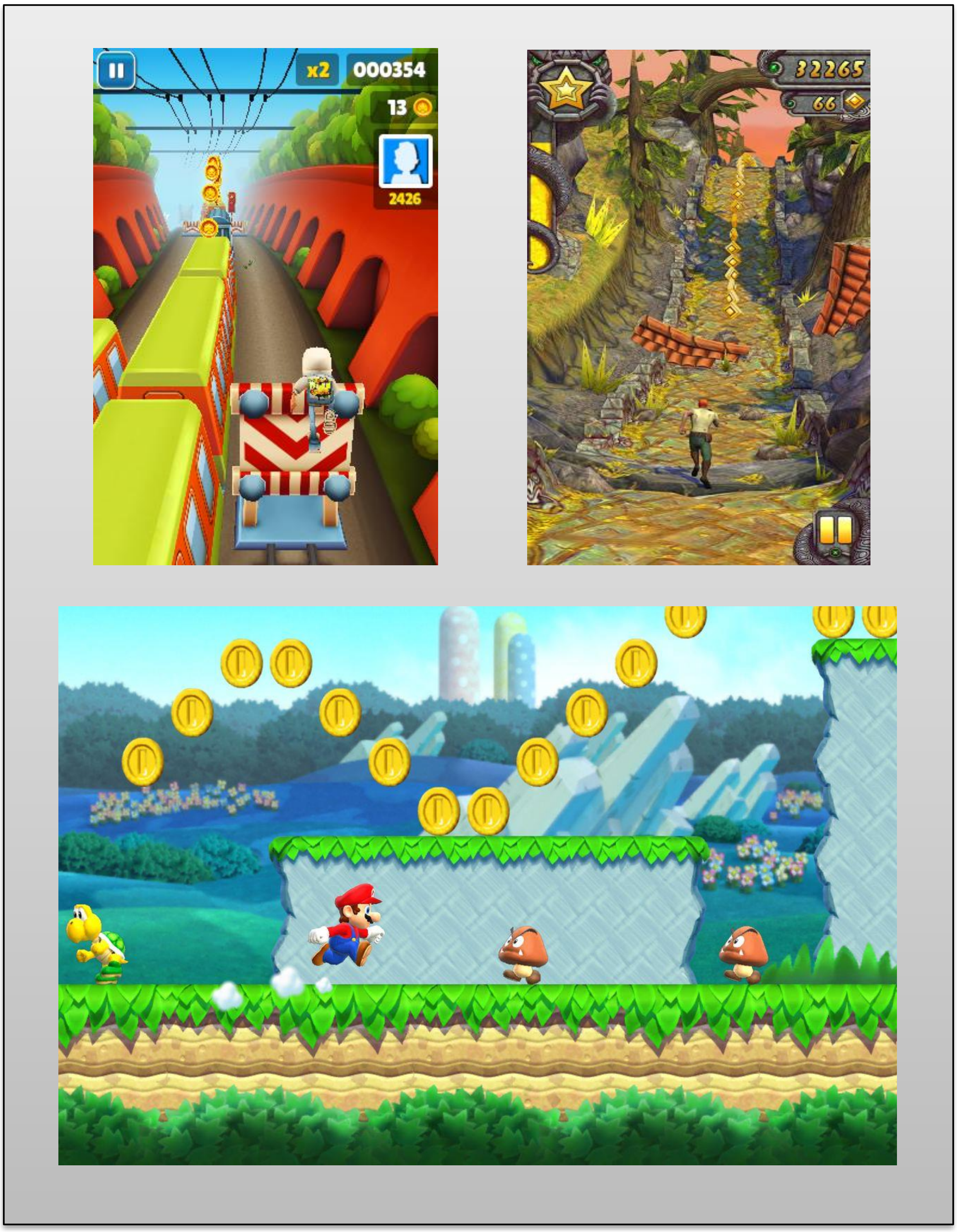


Fig2. Some examples of Endless runner games.

3. “Unity” the Game Engine

According to Wikipedia: -

“Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as an OS X-exclusive game engine. As of 2018, the engine has been extended to support 27 platforms.”

Unity is quite diverse game engine. It supports three scripting languages: **JavaScript**, **C#** and **Bool**. Unity allows the game developers to develop the 2D as well as 3D games for all kinds of devices ranging from Regular devices to **VR-Devices**. Now it supports around 27 platforms like **Android**, **WindowsPC, MacOSX, iOS, Blackberry, PlayStation, Xbox** etc.

Core of the game designing in unity are **GameObjects**, **Prefabs**, UI Elements etc. Well everything inside the scene is a **GameObject** and any number of scripts can be attached to a single GameObject to perform various kinds of functions. A **Camera, UI Element etc** everything is a GameObject.

“C# for Unity” is a scripting language which has largest userbase among all the three scripting languages. Almost all the scripts extend the **MonoBehaviour**. But all the scripts attached to a GameObject should extend the MonoBehaviour class. MonoBehaviour is a class in namespace **UnityEngine**.

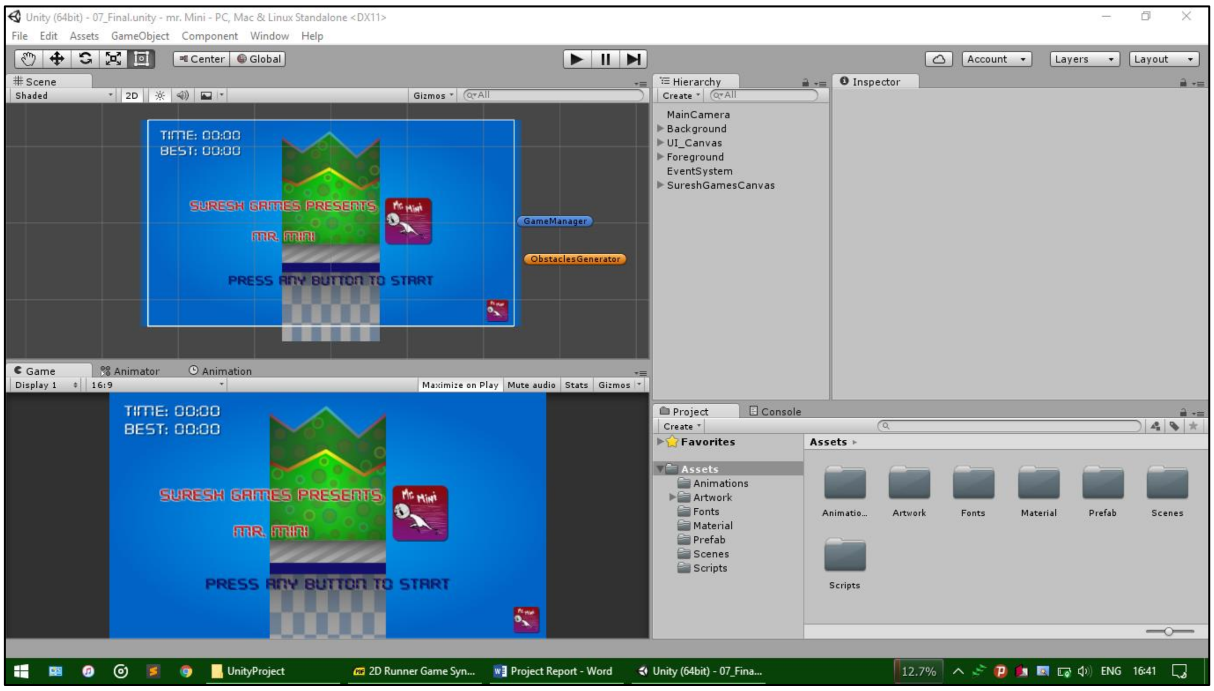


Fig3. Unity Game Engine Interface

4. Planning the Game level and Gameplay

4.1 Welcome Screen

When the game file is opened following screen shall be the welcome screen. Instruction on the screen will tell the player to start the screen. For example let's say "Press Spacebar to start the game.

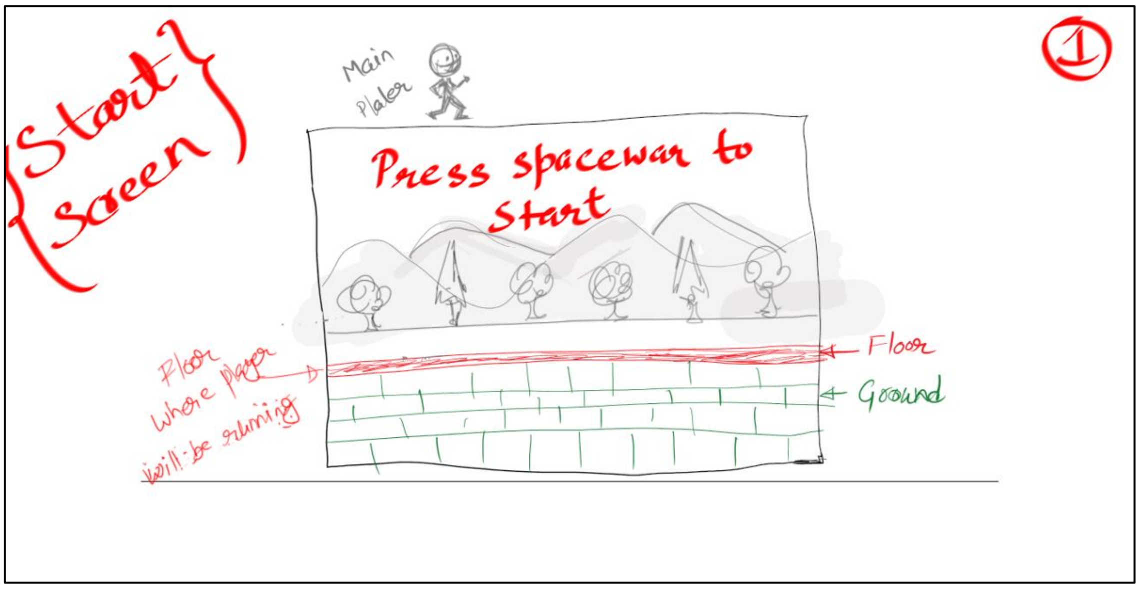


Fig4. Welcome screen of the game.

4.2 Start Screen of the Game

As the game starts after the user cause the starting of game as per the instructions given in the screen. Main Player character comes down inside the screen

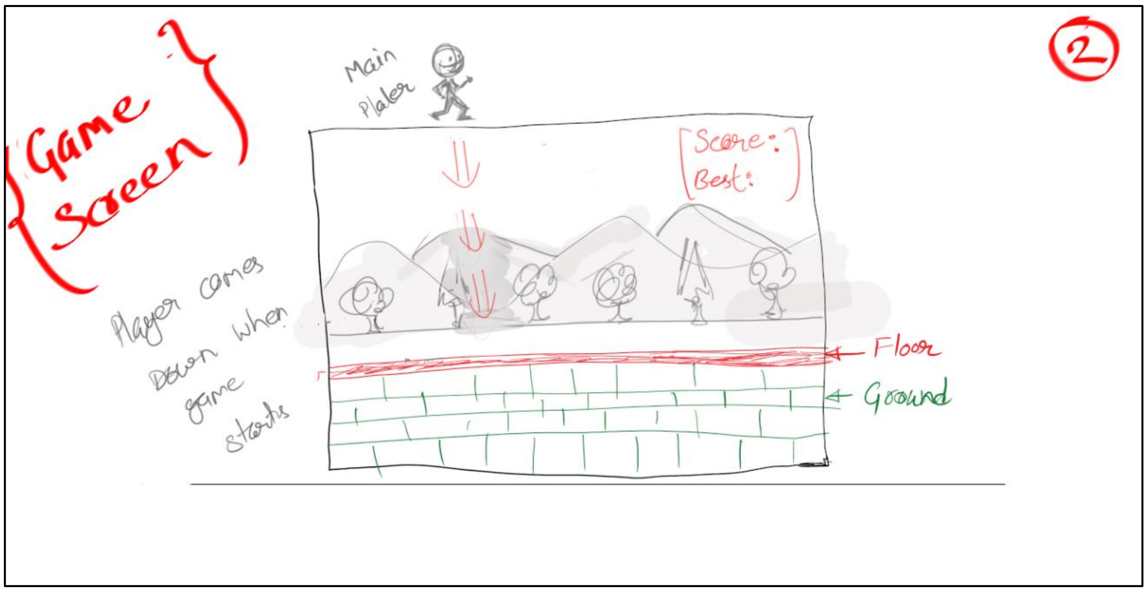


Fig5. Player comes down when starts running.

4.3 Gameplay planning Pt.1

When game is running, obstacles are introduced from the right side of the screen. Here various kinds obstacles can be chair, table, trash-bin, TV etc.

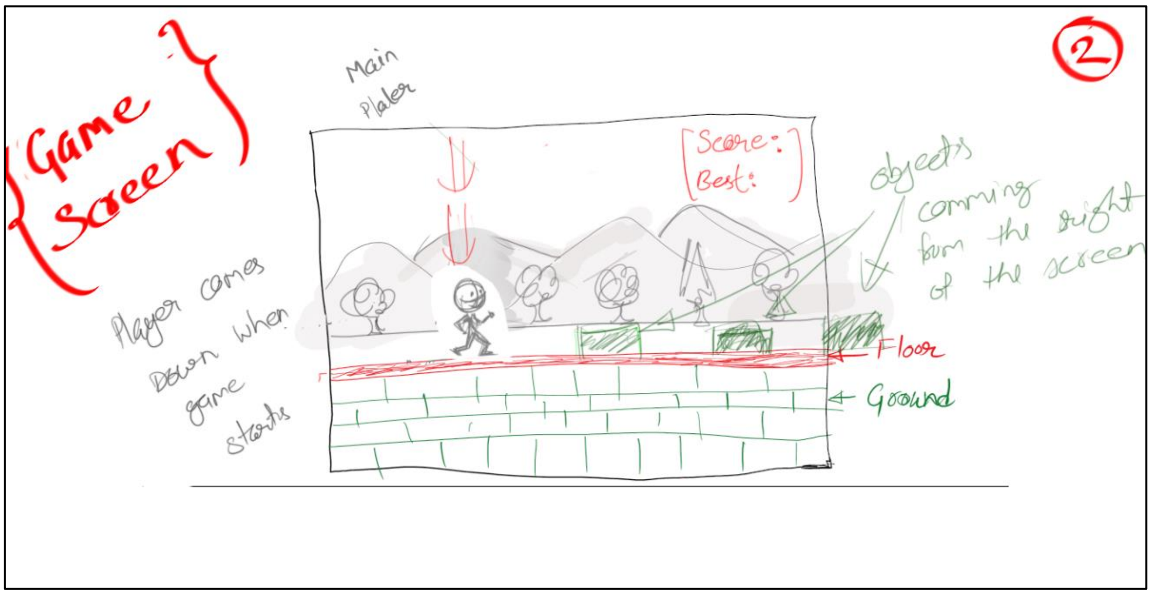

Game Screen
Main Plaler
Score:
Best:
objects comming from the right of the screen
Player comes Down when game starts
Floor
Ground
2

Fig6. Gameplay Planning, various objects are introduced from right side of screen.

4.4 Gameplay planning Pt.2

Ending the game, when player delays the jump, main player character gets pushed out of the screen.

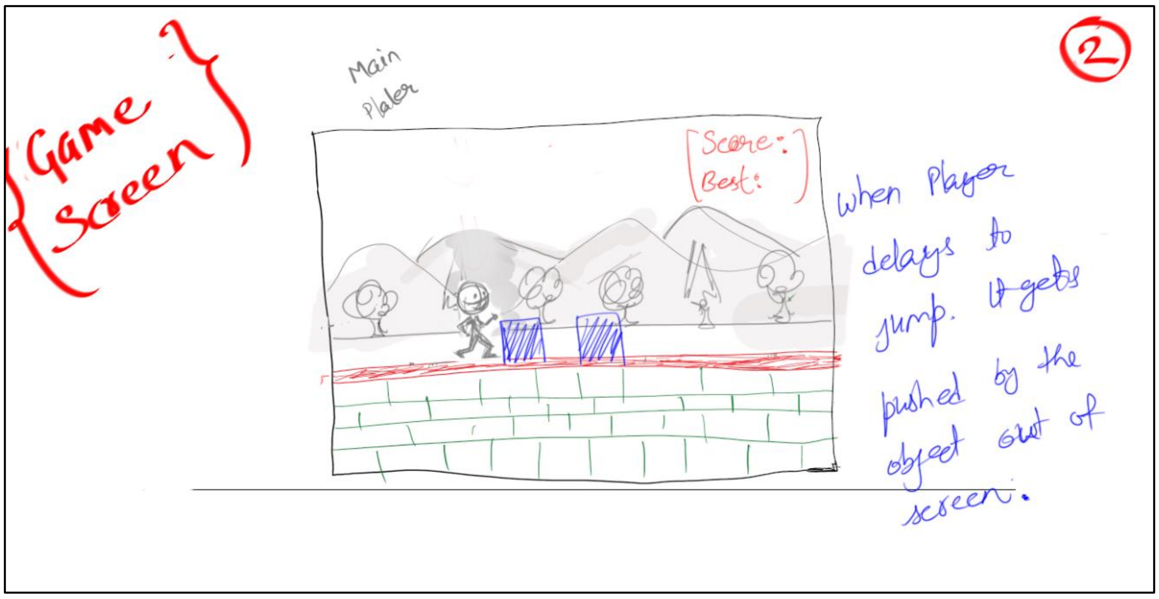

Game Screen
Main Plaler
Score:
Best:
when Player delays to jump. It gets pushed by the object out of screen.
2

Fig7. Player is in trouble when delay the jump.

4.5 Ending and Restarting

When player gets pushed out of the screen, game ends and after it we shall tell the player to restart the game as we did to start the game by simply putting the simple the instruction on the screen. For example, “Press Spacebar to Restart the Game”. And as the game finishes score and best score is updated to reward the player.

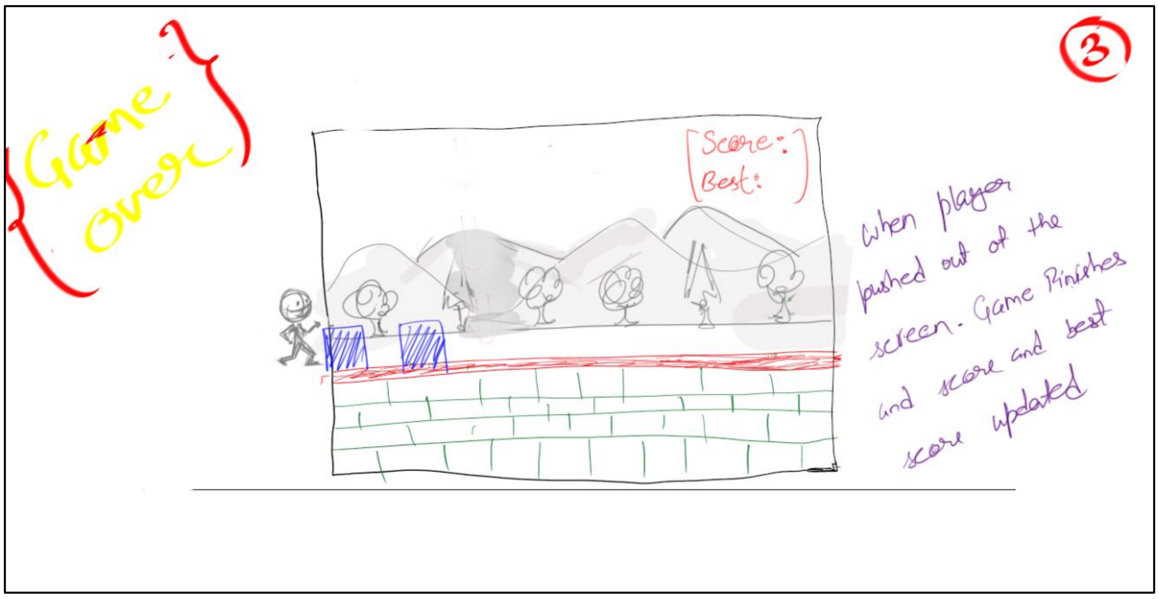

Game Over
3
Score:
Best:
When player pushed out of the screen. Game Finishes and score and best score updated

Fig8. Ending and restarting the game.

5. GDD (Game Design Document)

According to Wikipedia: -

"A game design document is a highly descriptive living design document of the design for a video game. A GDD is created and edited by the development team and it is primarily used in the video game industry to organize efforts within a development team"

Structure of My **GDD** is given below: -

5.1 Name of the Game – "mr. Mini".
5.2 Theme/Genre – Endless runner/2D Game.
5.3 Core Gameplay mechanics brief.
5.4 Project Scope.
  5.4.1 Hardware.
  5.4.2 Software Requirement
5.5 Assets needed.
  5.5.1 Textures.
  5.5.2 Sprites.
  5.5.3 Sprite-sheets/Animation.
5.6 Code/Programs.
5.7 Building the Game.

My entire game development is organized on this GDD (Game Design Document). It works as a blue print or roadmap for game development.

A detailed **GDD** is very useful and is being used by all the big studios which help them to set the budget, finish line for various phases during game Development.

The document is created by the development team as result of collaboration between their designers, artists and programmers as a guiding vision which is used throughout the game development process. When a game is commissioned by a game publisher to the development team, the document must be created by the development team and it is often attached to the agreement between publisher and developer; the developer has to adhere to the GDD during game development process.

<u>Life Cycle: -</u>

Game developers may produce the game design document in the pre-production stage of game development—prior to or after a pitch. Before a pitch, the document may be conceptual and incomplete. Once the project has been approved, the document is expanded by the developer to a level where it can successfully guide the development team. Because of the dynamic environment of game development, the document is often changed, revised and expanded as development progresses and changes in scope and direction are explored. As such, a game design document is often referred to as a living document, that is, a piece of work which is continuously improved upon throughout the implementation of the project, sometimes as often as daily. A document may start off with only the basic concept outlines and become a complete, detailed list of every game aspect by the end of the project.

6. Theme/Genre

"Theme/ Genre – 2D Game, Endless runner."

Genre: -

Endless runner games simply mean games which don't end. You continuously "run" through the game until an obstacle stops you with the **point** being to achieve the highest possible score. Their quick play, **addicting** nature, and simple controls make them an accessible genre that virtually anyone can pick up and play.
Some popular examples of Endless runner games are:

1. Temple run.
2. Sonic Dash.
3. Subway Surfers.
4. Super Mario Run.
5. Jetpack Joyride.
6. Pac-Man Dash!
7. Looney Tunes Dash.

Theme: -

2D games are also called Platform game. The word platform also describes that something held on a platform. Here the player can run, jump, shoot, collect powers on a platform. It is a video game genre; 2D games are become very old. But some developers still like to play 2D games to get the innovative idea, because we learn everything from past. That's why they want to play 2D games to make their game more interesting and get idea to give some new feature in their game. Mostly the characters of 2D games are cartoonish and unrealistic. We can't give a realistic feel in our 2D character. But in 3D it is possible because of depth. By using depth, we can make a character which would look like a real. Mostly platform games are based on some levels, if the player can kill all enemies or cross a certain part which heshe has to cross (like-Mario) then only the player can move to the next level. In next level may be there would be more enemies which the player has to kill. As per my knowledge the era of platform games started in the early 1980's and the 3D games started in mid of 1990. There is some confusion that which is the first 2D games.

Frogs is an arcade game which released in 1978, this is the first game where the character can jump on the screen, making the genre's earliest ancestor. Space Panic, which is also arcade game and released in 1980, is sometimes credited as the first platform game. Donkey Kong is an arcade game created by Nintendo and released in July 1981, was the first game that allowed players to jump over elements which are there and across gaps, making it the first true platformer. There are some versions of Donkey Kong. The next version of Donkey Kong is Donkey Kong Jr. which is also become a famous game. Donkey Kong introduces Mario. The third version of Donkey Kong was not become so famous but it succeeds by Mario Bros, which is a platform game and it has an extra feature of multiplayer, where two players can play simultaneously. By using the same rule in future many gaming companies made multiplayer games. Pitfall is a video game released by Activision in 1982 for Atari 2600.

7. Game Mechanics Brief

According to Wikipedia: -

“**Game mechanics** are methods invoked by agents designed for interaction with the game state, thus providing gameplay. All games use mechanics; however, theories and styles differ as to their ultimate importance to the game. In general, the process and study of game design, or ludology, are efforts to come up with game mechanics that allow for people playing a game to have an engaging, but not necessarily fun, experience.”

The interaction of various game mechanics in a game determines the complexity and level of player interaction in the game, and in conjunction with the game's environment and resources determine game balance. Some forms of game mechanics have been used in games for centuries, while others are relatively new, having been invented within the past decade.

Game Mechanics in “mr. Mini”: -

Player shall run endlessly till it is stuck by an object either because of delaying the jump or by jumping wrongly in front of an object.

Player will be rewarded with time as score. If player beats the old score, he shall be rewarded with a new best score.

Gameplay is very straight and simple player has to make a perfect jump over the objects.

8. Project Scope

8.1 Hardware Requirement

S. No.	NAME	HARDWARE
1	Processor Speed	1.6Ghz Minimum
2	RAM	4GB RAM
3	Hard Disk Capacity	10GB
4	Mouse (Recommended)	3 Button Mouse

8.2 Software Required

1. Adobe Photoshop CC
Purpose – To Design the various Textures needed for the game.

2. Adobe Illustrator CC
Purpose – To Design the various Sprites.

3. Adobe Animate CC
Purpose – To Create the character animation and other animations.

4. Unity 3D
Purpose – To Compile the Whole game.

5. Mono Develop
Purpose – To write the **C#** scripts for various requirements in the game.

9. Assets Needed

9.1 Textures

A texture is Simply an image file. Generally, in **PNG, JPEG, Gif** formats but mostly developers like to use **PNG** file format images. To show the different shapes for transparency is used. Since, **JPEG** do not support alpha i.e. transparency therefore rarely used. And **GIF** is mainly developed for **web** therefore it is also avoided.

Textures bring your Meshes, Particles, and interfaces to life! They are image or movie files that we lay over or wrap around GameObjects. As they are so important, they have a lot of properties.

The shaders you use for your objects put specific requirements on which textures you need, but the basic principle is that you can put any image file inside your project. If it meets the size requirements (specified below), it will get imported and optimized for game use. This extends to multi-layer Photoshop or TIFF files - they are flattened on import, so there is no size penalty for your game. Note that this flattening happens internally to Unity, and is optional, so you can continue to save and import your PSD files with layers intact. The PSD file is not flattened, in other words.

Following texture are created in Adobe Photoshop CC and Adobe Illustrator CC.

1. **Mountains**

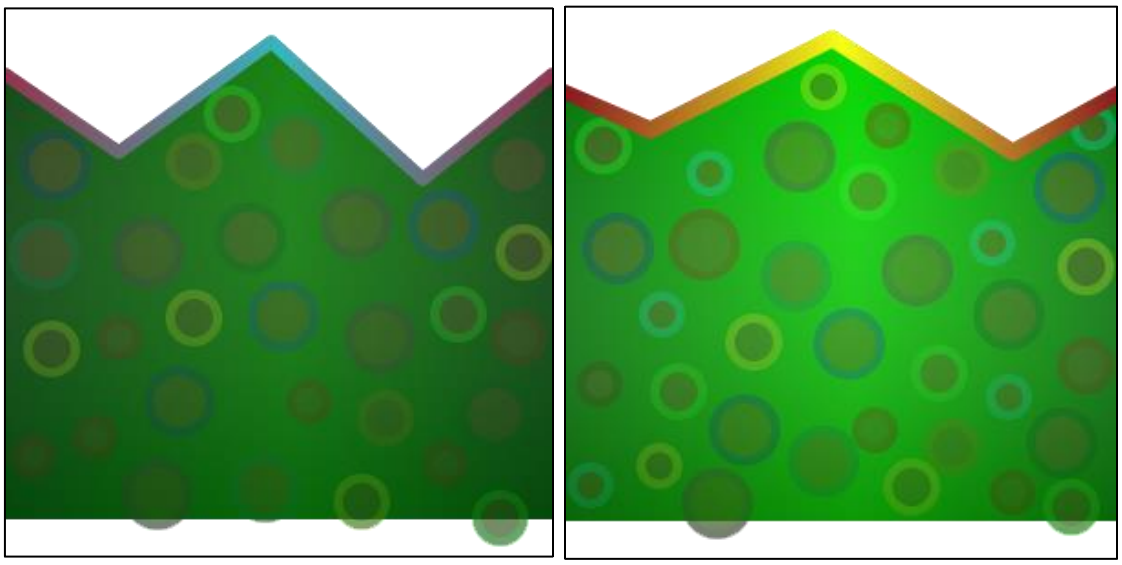


Fig9. Mountain back and fore (left to right).

2. **Ground/Floor**

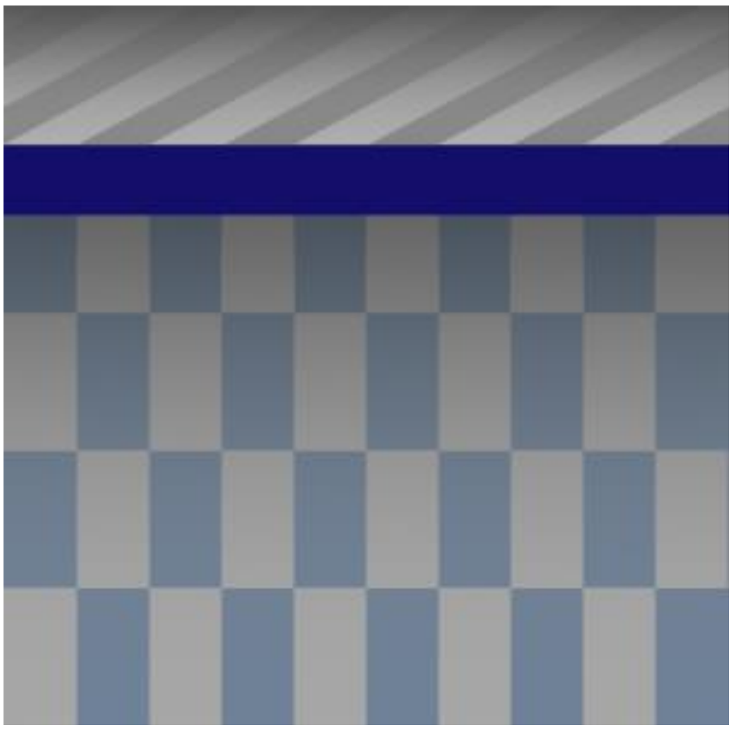


Fig10. Ground/Floor Texture.

3. **Sky**

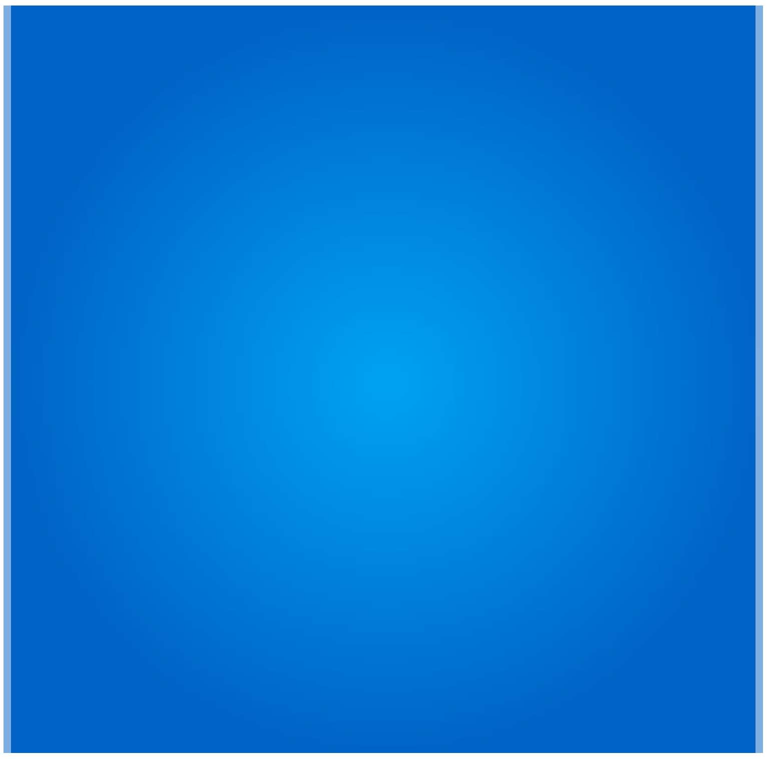


Fig11. Sky Texture.

9.2 Sprites

Sprite is a computer graphics term for a two-dimensional bitmap that is integrated into a larger scene. Originally sprites referred to independent objects that are composited together, by hardware, with other elements such as a background. The composition occurs as each scan line is prepared for the video output device, such as a CRT, without involvement of the main CPU and without the need for a full-screen frame buffer. Sprites can be positioned or altered by setting attributes used during the hardware composition process. Examples of systems with hardware sprites include the Atari 8-bit family, Commodore 64, Amiga, Nintendo Entertainment System, Sega Genesis, and many coin-operated arcade machines of the 1980s. Sprite hardware varies in how many sprites are supported, how many can be displayed on a single scan line (which is often a lower number), the dimensions and colours of each sprite, and special effects such as scaling or reporting pixel-precise overlap.
Use of the term *sprite* has expanded to refer to any two-dimensional bitmap used as part of a graphics display, even if drawn into a frame buffer (by either software or a GPU) instead of being composited on-the-fly at display time.

Following Sprites were created in Adobe Illustrator CC.

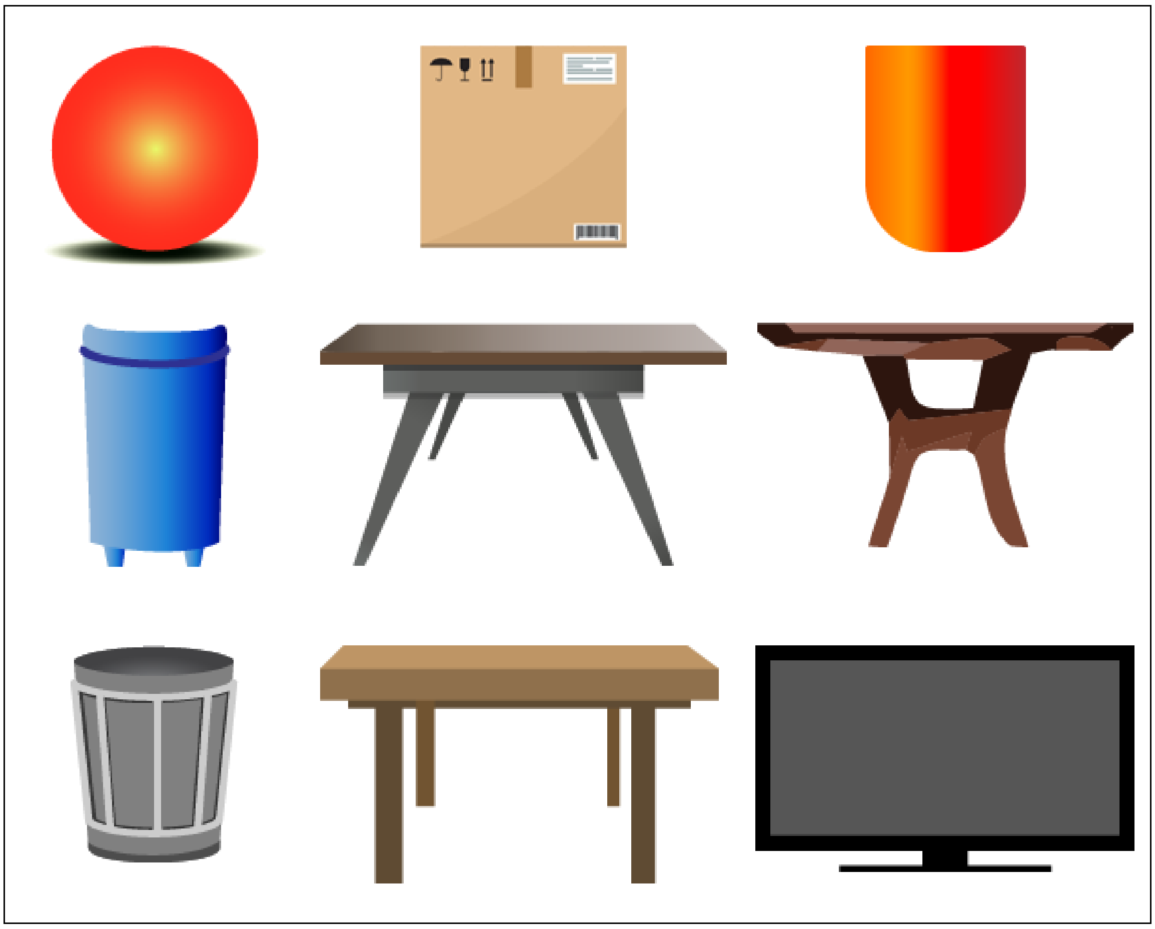


Fig12. Objects (2D Sprites).

9.3 Sprite-sheets/animation

A sprite sheet is a bitmap image file that contains several smaller graphics in a tiled grid arrangement. By compiling several graphics into a single file, you enable Animate and other applications to use the graphics while only needing to load a single file. This loading efficiency can be helpful in situations such as game development where performance is especially important.

We can create a sprite sheet from a selection of any combination of movie clips, button symbols, graphic symbols, or bitmaps. You can select items in the Library panel or on the Stage, but not both. Each bitmap and each frame of the selected symbols appear as a separate graphic in the sprite sheet. If you export from the Stage, any transforms (scaling, skewing, and so on) you have applied to the symbol instance are preserved in the image output.

Following Animations are created using Adobe Photoshop, Adobe Animate CC.

1. Main Character Animation Sheet

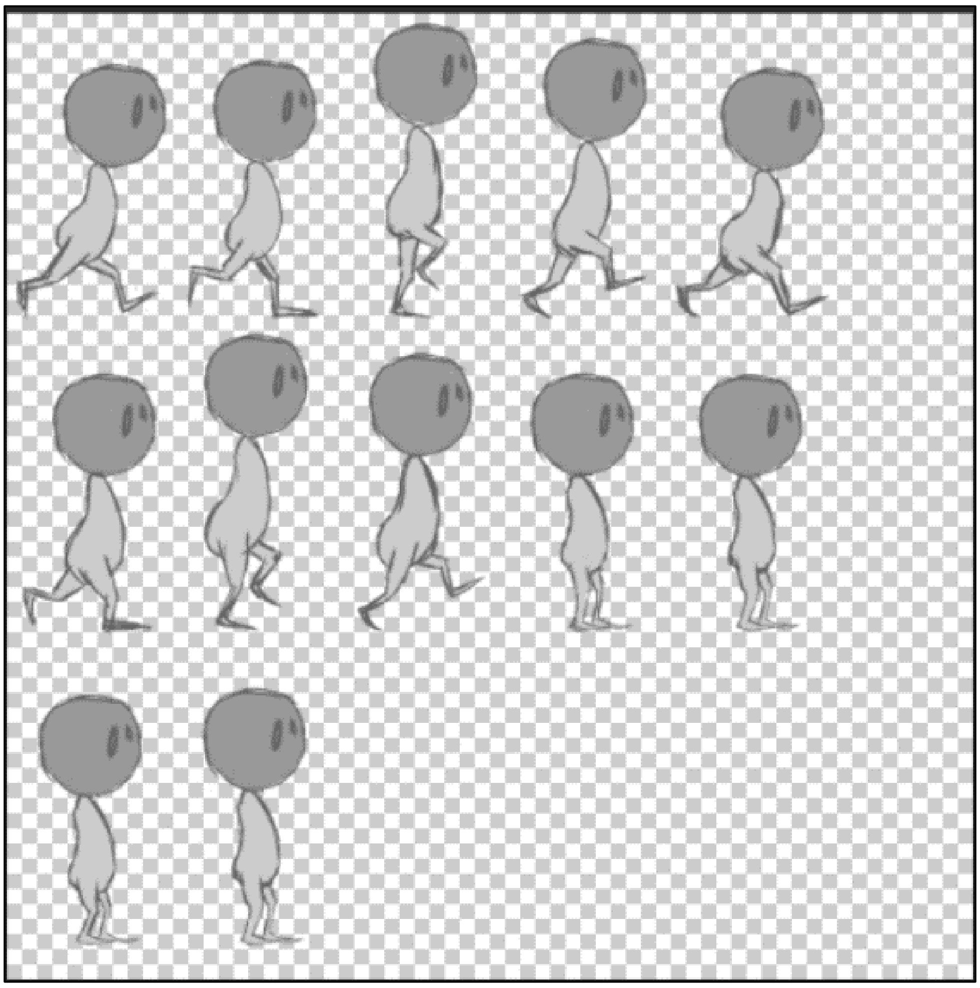


Fig13. Objects (2D Sprites).

2. TV Animation Sheet

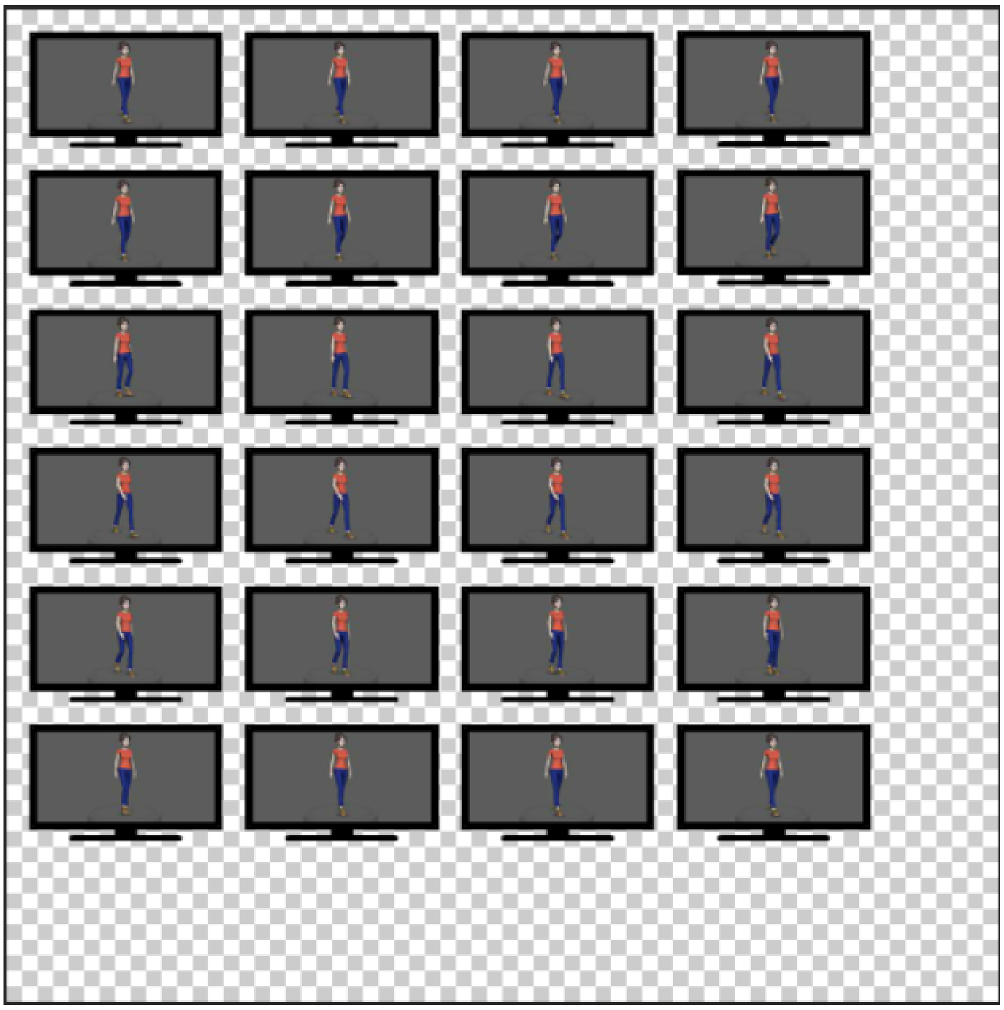


Fig14. Objects (2D Sprites).

Spritesheets have been used in games for a long time. Classic games such as **Legend of Zelda: A Link to the Past** and even modern games like **Cut the Rope** have used them.

A **sprite** is a single graphic image that is incorporated into a larger scene so that it appears to be part of the scene. Sprites are a popular way to create large, complex scenes as you can manipulate each sprite separately from the rest of the scene. This allows for greater control over how the scene is rendered, as well as over how the players can interact with the scene.

It is not **uncommon** for games to have tens to **hundreds of sprites**. Loading each of these as an individual image would **consume** a lot of **memory** and **processing** power. To help manage sprites and **avoid** using so **many images**, many games use **spritesheets**.

9.4 Logo Design

A logo plays a vital role in deployment of a game. So, coming up with a log which resonates the game mechanics is very important. Since “mr. Mini” is extremely straight and simple endless runner game but a logo should be attractive and cool to look and must attract the users for boosted downloads. It is very important that a logo should contain the core of game mechanics in it. Therefore, putting the name and character in the logo can be important thing to do.

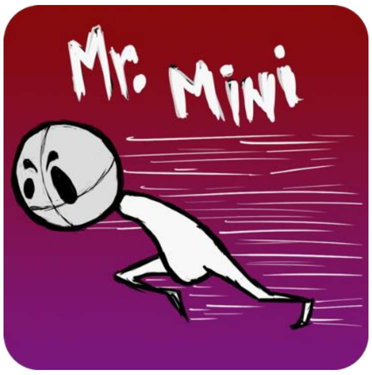


Fig15. Game Logo

10. Programming the scripts

The behaviour of **GameObjects** is controlled by the **Components** that are attached to them. Although Unity's built-in Components can be very versatile, you will soon find you need to go beyond what they can provide to implement your own gameplay features. Unity allows you to create your own Components using **scripts**. These allow you to trigger game events, modify Component properties over time and respond to user input in any way you like.

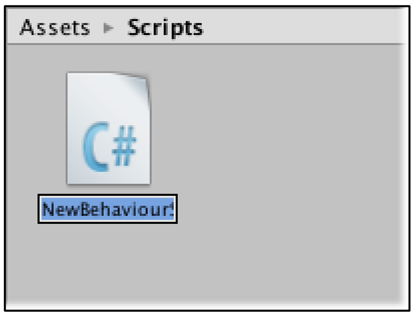
Unity supports the C# programming language natively. C# (pronounced C-sharp) is an industry-standard language similar to Java or C++.

In addition to this, many other .NET languages can be used with Unity if they can compile a compatible DLL.

Creating the Scripts: -

Unlike most other assets, scripts are usually created within Unity directly. You can create a new script from the Create menu at the top left of the Project panel or by selecting **Assets > Create > C# Script** from the main menu.

The new script will be created in whichever folder you have selected in the Project panel. The new script file's name will be selected, prompting you to enter a new name.



Anatomy of a C# Script file: -

When you double-click a script Asset in Unity, it will be opened in a text editor. By default, Unity will use Visual Studio, but you can select any editor you like from the **External Tools** panel in Unity's preferences (go to **Unity** > **Preferences**).

```
using UnityEngine;
using System.Collections;
public class MainPlayer : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }
    // Update is called once per frame
    void Update () {
    }
}
```

Following Scripts were written in C# for all the GameObjects in the Game: -

1. **ScalableCamera.cs**

   This script is used to scale all the graphics to different scales of camera. Since, game is going to run in different displays so scaling the graphics and maintaining the ratio between screen and graphics is important.

```
using UnityEngine;
using System.Collections;

public class ScalableCamera : MonoBehaviour {

     public static float scale = 1f;

     public Vector2 nativeRes = new Vector2(480, 270);

     void Awake(){
          var camera = GetComponent<Camera> ();
          if (camera.orthographic) {
               scale = Screen.height / nativeRes.y;
               camera.orthographicSize = (Screen.height /
               2.0f) / scale;
          }
     }
}
```

2. **TiledTexture.cs**

   This script is used to scale to the textures according to the size of the screen. For example, Mountain, Ground and Sky etc.

```
using UnityEngine;
using System.Collections;

public class TiledTexture : MonoBehaviour {

     public int textureSize = 128;

     // Use this for initialization
     void Start () {

     var newWidth  = Mathf.Ceil (Screen.width /
     (textureSize * ScalableCamera.scale));

     transform.localScale = new Vector3 (newWidth *
     textureSize, textureSize, 1);

     GetComponent<Renderer> ().material.mainTextureScale = new
     Vector3 (newWidth, 1, 1);
     }
}
```

3. TextureAnimation.cs

To Animate the texture. This script is used to animate the texture offset. Since textures are designed in such a way that they are repeatable to either side so just altering the offset of texture can create the sense of animation of moving background.

```
using UnityEngine;
using System.Collections;

public class TextureAnimation: MonoBehaviour {

	public Vector2 speed = Vector2.zero;

	private Vector2 offset = Vector2.zero;
	private Material material;

	// Use this for initialization
	void Start () {
		material = GetComponent<Renderer> ().material;

		offset = material.GetTextureOffset ("_MainTex");
	}

	// Update is called once per frame
	void Update () {
		offset = offset + speed * Time.deltaTime;
		material.SetTextureOffset ("_MainTex", offset);
	}
}
```

4. ObstaclesMovement.cs

Since we have number of objects to move from right of the screen to left and if have to do it manually it will be time consuming and exhausting approach therefore writing a script and letting the scripting object movement will reduce the workload.

```
using UnityEngine;
using System.Collections;

public class ObstaclesMovement : MonoBehaviour {

	public Vector2 velocity = Vector2.zero;

	private Rigidbody2D body2d;

	void Awake(){
		body2d = GetComponent<Rigidbody2D> ();
	}

	void FixedUpdate(){
		body2d.velocity = velocity;
	}
}
```

5. **ObstaclesGenerator.cs**

   We need a spawner to create the objects in the right side of screen which picks random objects everytime among all the available objects in the array.

```
using UnityEngine;
using System.Collections;
using UnityEngine.SocialPlatforms;

public class ObstaclesGenerator : MonoBehaviour {

	public GameObject[] obstaclesPrefabs;
	public float delay = 2.0f;
	public bool active = true;
	public Vector2 delayRange = new Vector2 (1, 3);

	// Use this for initialization
	void Start () {
		ResetDelay ();
		StartCoroutine (ObjectGenerator());
	}

	IEnumerator ObjectGenerator(){
		yield return new WaitForSeconds (delay);

		if (active) {
			var newTransform = transform;

			GameObjectUtil.Instantiate
(obstaclesPrefabs[Random.Range (0,obstaclesPrefabs.Length)],
newTransform.position);

			ResetDelay ();
		}

		StartCoroutine (ObjectGenerator());

	}

	void ResetDelay(){
		delay = Random.Range (delayRange.x, delayRange.y);
	}
}
```

6. **GameObjectUtil.cs**

It is a utility script used as an Interface to maintain the various things in the game.

```
     using UnityEngine;
     using System.Collections;
using System.Collections.Generic;

public class GameObjectUtil {

     private static Dictionary<RecycleGameObjects,
ObjectsPool> pools = new Dictionary<RecycleGameObjects,
ObjectsPool> ();

     //=====================================================
=======================================
     public static GameObject Instantiate(GameObject
prefab, Vector3 pos){
          GameObject instance = null;

          var recycledScript =
prefab.GetComponent<RecycleGameObjects> ();
          if (recycledScript != null) {
               var pool = GetObjectPool (recycledScript);
               instance = pool.NextObject
(pos).gameObject;
          } else {
               instance = GameObject.Instantiate
(prefab);
               instance.transform.position = pos;
          }
          return instance;
     }

     //=====================================================
=======================================
     public static void Destroy(GameObject gameObject){

          var recycleGameObject =
gameObject.GetComponent<RecycleGameObjects> ();

          if (recycleGameObject != null) {
               recycleGameObject.ShutDown ();
          } else {
               GameObject.Destroy (gameObject);
          }
     }

     //=====================================================
=======================================
     private static ObjectsPool
GetObjectPool(RecycleGameObjects reference){
          ObjectsPool pool = null;
          if (pools.ContainsKey (reference)) {
               pool = pools [reference];
          } else {
               var poolContainer = new GameObject
(reference.gameObject.name + "ObjectsPool");
               pool =
poolContainer.AddComponent<ObjectsPool> ();
               pool.prefab = reference;
               pools.Add (reference, pool);
          }

          return pool;
     }
}
```

7. **ObjectsPool.cs**

We are using the objects after sometime therefore destroying and creating them again and again is memory and CPU consuming thing. Therefore we are creating a object pool and instead of destroying them we are reusing them from the pool.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class ObjectsPool : MonoBehaviour {

     public RecycleGameObjects prefab;

     private List<RecycleGameObjects> poolInstances = new
List<RecycleGameObjects> ();

     //=====================================================
     private RecycleGameObjects CreatInstances(Vector3 pos){

          var clone = GameObject.Instantiate (prefab);
          clone.transform.position = pos;
          clone.transform.parent = transform;

          poolInstances.Add (clone);

          return clone;
     }

     //=====================================================
     public RecycleGameObjects NextObject (Vector3 pos){
          RecycleGameObjects instance = null;

          foreach (var gameObj in poolInstances) {
               if (gameObj.gameObject.activeSelf != true) {
                    instance = gameObj;
                    instance.transform.position = pos;
               }
          }

          if (instance == null) {
               instance = CreatInstances (pos);
          }

          instance.Restart ();

          return instance;
     }
}
```

8. **RecycleGameObjects.cs**

   This script reuses the objects from object pool.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public interface IRecyle{

     void Restart ();
     void ShutDown ();

}

public class RecycleGameObjects : MonoBehaviour {

     private List<IRecyle> recycleComponents;

     void Awake(){

          var components = GetComponents<MonoBehaviour> ();
          recycleComponents = new List<IRecyle> ();

          foreach (var component in components) {
               if (component is IRecyle) {
                    recycleComponents.Add (component as
IRecyle);
               }
          }

          //Debug.Log (name + " Found " +
recycleComponents.Count + " Components");
     }

     public void Restart(){
```

```
		gameObject.SetActive (true);

		foreach (var component in recycleComponents) {
			component.Restart ();
		}
	}

	public void ShutDown(){
		gameObject.SetActive (false);

		foreach (var component in recycleComponents) {
			component.ShutDown ();
		}
	}
}
```

9. **InputState.cs**

We are only asking the player to let the main character to jump therefore to make this happen user need to give an input either through keyboard or mouse. And to connect the input to character this script is used.

```
	using UnityEngine;
	using System.Collections;

	public class InputState : MonoBehaviour {

		public bool actionButton;
		public float absVelX = 0f;
		public float absVelY = 0f;
		public bool standing = false;
		public float standingThresold = 1f;

		private Rigidbody2D body2d;

		void Awake(){
			body2d = GetComponent<Rigidbody2D> ();
		}

		// Update is called once per frame
		void Update () {
			actionButton = Input.anyKeyDown;
		}

		void FixedUpdate(){
			absVelX = System.Math.Abs (body2d.velocity.x);
			absVelY = System.Math.Abs (body2d.velocity.y);

			standing = absVelY <= standingThresold;
		}
}
```

10. PlayerAnimationManager.cs

Our main and only character has two animation first is running animation which is default and second is idle animation when our character is standing on any object. So, to connect these two animations this script is used.

```
	using UnityEngine;
	using System.Collections;

	public class PlayerAnimationManager : MonoBehaviour {

		private Animator animator;
		private InputState inputState;

		void Awake(){
			animator = GetComponent<Animator> ();
			inputState = GetComponent<InputState> ();
		}

		// Update is called once per frame
		void Update () {

			var running = true;

			if (inputState.absVelX > 0 &&
			inputState.absVelY <
			inputState.standingThresold) {
				running = false;
			}

			animator.SetBool ("Running", running);
		}
}
```

11. Obstacles.cs

Through this script we are selecting the various sprites as objects form the array of sprites.

```
using UnityEngine;
using System.Collections;

public class Obstacles : MonoBehaviour, IRecyle {

	public Sprite[] sprites;

	public void Restart(){
		var renderer = GetComponent<SpriteRenderer> ();
		renderer.sprite = sprites [Random.Range (0,
sprites.Length)];
```

```
            var collider = GetComponent<BoxCollider2D> ();
            collider.size = renderer.bounds.size / 30;
        }

        public void ShutDown(){

        }
}
```

12. Jump.cs

In this script jump information of the player is given. How much high and forward player goes when he jumps.

```
        using UnityEngine;
        using System.Collections;

        public class Jump : MonoBehaviour {

            public float jumpSpeed = 300f;
            public float forwardSpeed = 20;

            private Rigidbody2D body2d;
            private InputState inputState;

            void Awake(){
                body2d = GetComponent<Rigidbody2D> ();
                inputState = GetComponent<InputState> ();
            }

            // Update is called once per frame
            void Update () {

                if (inputState.standing) {
                    if (inputState.actionButton) {
                        body2d.velocity = new Vector2
        (transform.position.x < 0 ? forwardSpeed : 0,
        jumpSpeed);
                    }
                }

            }
}
```

13. DestroyOffscreen.cs

As the objects goes to the left of the screen

```
using UnityEngine;
using System.Collections;
```

```
public class DestroyOffscreen : MonoBehaviour {

     public float offset = 16f;
     public delegate void OnDestroy();
     public event OnDestroy DestroyCallback;

     private bool offscreen;
     private float offscreenX = 0f;
     private Rigidbody2D body2d;

     void Awake(){
          body2d = GetComponent<Rigidbody2D> ();
     }

     // Use this for initialization
     void Start () {
          offscreenX = (Screen.width / ScalableCamera.scale)
     / 2 + offset;
     }

     // Update is called once per frame
     void Update () {

          var posX = transform.position.x;
          var dirX = body2d.velocity.x;

          if (Mathf.Abs (posX) > offscreenX) {

               if (dirX < 0 && posX < -offscreenX) {
                    offscreen = true;
               } else if (dirX > 0 && posX > offscreenX) {
                    offscreen = true;
               }
          } else {
               offscreen = false;
          }

          if (offscreen) {
               OnOutOfBounds ();
          }
     }

     public void OnOutOfBounds (){
          offscreen = false;
          GameObjectUtil.Destroy (gameObject);

          if(DestroyCallback != null){
               DestroyCallback();
          }
     }
}
```

14. TimeManager.cs

```
	using UnityEngine;
	using System.Collections;

	public class TimeManager : MonoBehaviour {

		public void ManipulateTime(float newTime, float
	duration){
			if (Time.timeScale == 0) {
				Time.timeScale = 0.1f;
			}

			StartCoroutine ( FadeTo (newTime, duration));
		}

		IEnumerator FadeTo(float value, float time){

			for (float t = 0f; t < 1; t +=
	Time.deltaTime/time) {
				Time.timeScale = Mathf.Lerp
	(Time.timeScale, value, t);

				if (Mathf.Abs (value - Time.timeScale) <
	.01f) {
					Time.timeScale = value;
					return false;
				}

				yield return null;
			}
		}
}
```

15. GameManager.cs

```
	using UnityEngine;
	using System.Collections;
	using UnityEngine.UI;
	using System;

	public class GameManager : MonoBehaviour {

		public GameObject playerPrefab;
		public Text continueText;
		public Text gameScoreText;
		public Text sureshGamesPresents;
		public Image logoIntro;
```

```
      private float timeElapsed = 0f;
      private float bestTime = 0f;
      private float blinkTime = 0f;
      private bool blink;
      private bool gameStarted;
      private TimeManager timeManager;
      private GameObject player;
      private GameObject ground;
      private ObstaclesGenerator objGenerator;
      private bool beatBestTime;

      void Awake(){
           ground = GameObject.Find ("Ground");
           objGenerator = GameObject.Find
 ("ObstaclesGenerator").GetComponent<ObstaclesGenerator>
 ();
           timeManager = GetComponent<TimeManager> ();
      }

      // Use this for initialization
      void Start () {
           var groundHeight =
 ground.transform.localScale.y;

           var pos = ground.transform.position;
           pos.x = 0;
           pos.y = -((Screen.height /
 ScalableCamera.scale) / 2) + (groundHeight/2) - 20;
           ground.transform.position = pos;

           objGenerator.active = false;

           Time.timeScale = 0;

           continueText.text = "PRESS ANY BUTTON TO
 START";
           sureshGamesPresents.text = "SURESH GAMES
 PRESENETS \n\nMr. Mini";

           bestTime = PlayerPrefs.GetFloat ("BestTimeS");
      }

      // Update is called once per frame
      void Update () {
           if (!gameStarted && Time.timeScale == 0) {
                if (Input.anyKeyDown) {
                     timeManager.ManipulateTime (1, 1f);
                     ResetGame ();
                }
           }
```

```
            if (!gameStarted) {
                  blinkTime++;
                  if (blinkTime % 40 == 0) {
                        blink = !blink;
                  }

                  continueText.canvasRenderer.SetAlpha
(blink ? 0 : 1);

                  var textColor = beatBestTime ? "#FF0" :
"#FFF";

                  gameScoreText.text = "TIME: " +
FormatTime (timeElapsed) + "\n<color="+textColor+">BEST:
" + FormatTime (bestTime)+"</color>";
            } else {
                  timeElapsed += Time.deltaTime;
                  gameScoreText.text = "TIME: " +
FormatTime (timeElapsed);
            }


      }
      void OnPlayerKilled(){
            objGenerator.active = false;

            var PlayerDestroyScript =
player.GetComponent<DestroyOffscreen> ();
            PlayerDestroyScript.DestroyCallback -=
OnPlayerKilled;

            player.GetComponent<Rigidbody2D> ().velocity =
Vector2.zero;
            timeManager.ManipulateTime (0, 5.5f);

            gameStarted = false;

            continueText.text = "PRESS ANY BUTTON TO
RESTART";

            if (timeElapsed > bestTime) {
                  bestTime = timeElapsed;
                  PlayerPrefs.SetFloat ("BestTimeS",
bestTime);
                  beatBestTime = true;

            }

      }
```

```
          void ResetGame(){
               objGenerator.active = true;

               player = GameObjectUtil.Instantiate
     (playerPrefab, new Vector3 (0, Screen.height /
     ScalableCamera.scale / 2 + 50, 0));

               var PlayerDestroyScript =
     player.GetComponent<DestroyOffscreen> ();
               PlayerDestroyScript.DestroyCallback +=
     OnPlayerKilled;

               gameStarted = true;

               continueText.canvasRenderer.SetAlpha (0);
               sureshGamesPresents.canvasRenderer.SetAlpha
     (0);
               logoIntro.canvasRenderer.SetAlpha (0);

               timeElapsed = 0;
               beatBestTime = false;

          }

          string FormatTime(float value){
               TimeSpan t = TimeSpan.FromSeconds (value);

               return string.Format ("{0:D2}:{1:D2}",
     t.Minutes, t.Seconds);
          }
}
```

11. Building the Game Package

Building the game for deployment is the final step of game development phase. A game can be built for following platforms: -

Windows, Android, iOS, MacOSX, Window Store, Windows Phone, Xbox, Blackberry, Linux, Tizen, HTML5, Samsung TV, Apple TV, PlayStation, Facebook etc.

After building the game particular package should be uploaded to regarding store so that package is accessible to various user in the internet.

Following is the list for various platforms: -

Sr.	Platform	Place
01.	Android	Playstore i.e. http://play.google.com
02.	iOS	AppStore i.e. https://www.apple.com/in/ios/app-store/
03.	Windows Store	Microsoft Store i.e. https://www.microsoft.com/en-in/store/apps/windows

I have uploaded my game to http://itch.io as a zip file. Game is released only for one **WindowsPC** and can be downloaded from following link from my "**itch.io**" profile: -

Download here: -

https://shhotu010.itch.io/mr-mini

Watch gameplay here: -

https://www.youtube.com/watch?v=6ZI25BXc0f4&feature=youtu.be