Game Programming (CSE3029) – Theory

Digital Assignment-2

Name: **Anmol Pant** **Reg. No:** **18BCE0283**

Slot: **B1** **Faculty:** **Prof. Natarajan P.**

Ques: Developing a Part of a game or a Component of a game and upload the following as a word file in VTOP login.

1. Name of the game part.
2. Story of the game.
3. Specify in which part of the game, the designed part belongs to.
4. Description of design and development of the game part.
5. Specify the use of the developed game part on playing the game.
6. Specify in which language, tool, Game Engine and etc. you developed the game part.
7. Copy and Paste the Source code.
8. Describe the Source code.
9. Screen snap shot of the game part which you designed.
10. Compare your design with the existing game part available and identify what innovation you did on the designing and what improvement you did on the design of that game part.

Name of the game

Samurai Dash

SAMURAI DASH

Story of the Game

The story features the tale of Jack, a shinobi from Japan and one of the most renowned swordsmen of his days, who finds himself stuck in a floating labyrinth, with no memory of who he is and his past, his only ally? His sword.

With the only way out being accumulating collectables, slashing through walls and enemies and progressing level after level to escape the labyrinth, there is nowhere else to look but forward as he must overcome all the obstacles the floating labyrinth throws his way so as to redeem himself.

Design Part Insights

The part designed in the scope of this assignment covers the loading screen, the initial start screen, the character, the movements, the obstacles and collectables and the overall HUD and gameplay design.

The initial level that commences once the user presses the start key and the player avatar, Jack, starts running, has also been implemented to the full extent, with other levels that might follow in the future have been left off as an upcoming enhancement.

Hence, the player will get the complete feel of the game in the part that has been implemented and owing to the recent popularity of maze and endless runner games (like subway surfers, temple run, etc.) will find it easy to learn and fun to play.

Description of Design and Development

The design of the game includes the following major components, the canvas, the floating maze, our protagonist, the collectables and doors that unlock once all the collectables are accumulated.

Moreover, the golden rules of design have been kept in mind while designing the user interface and gameplay so as to give just the right amount of information to the player, moreover the movement and jump controls have been kept similar to all other games so as to reduce the load on the short term memory of the user.

This makes sure that the user can quickly and naturally adapt to the in game environment instead of having to pay unnecessary attention to trivial details.

The HUD has been implemented so that it is less cluttered and the color scheme of the entire game has been kept consistent to avoid unforeseen distractions to the players.

Moreover, menus and start and end interfaces, along with in game sound effects and music have also been incorporated to make sure an overall rich and gratifying user experience.

Use of Game Part in Complete Game

The part developed is almost a whole in itself so it isn't a surprise that it plays a vital role in completing and complimenting the entire gameplay.

The first and foremost usage being the allocation and mapping of the player movement controls, which stay the same throughout and are a vital part for giving the user control of the virtual world. The game controls should not only be consistent but also should accurately mirror the user’s mental model and what is visible on the screen at that very moment.

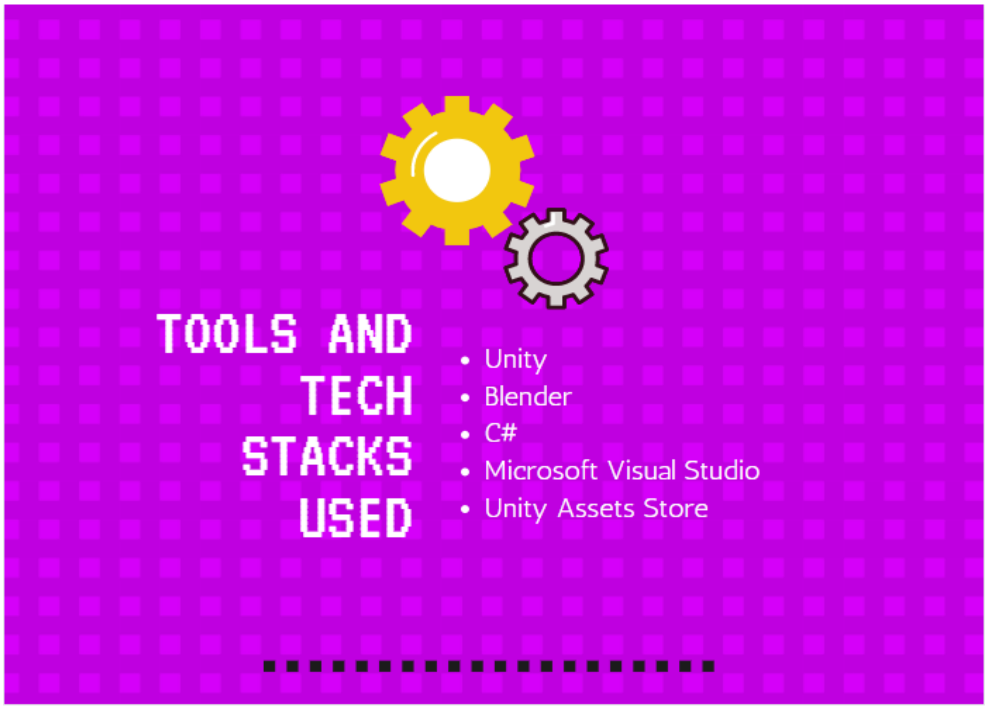
Another game long entity that has been completely implemented is our main character, Jack. The samurai is equipped with a sword and can perform basic in game functions like movement, collection of items, attacking enemies and obstacles etc.

Moreover, the first level of the maze, complete with doors, coins, keys, etc has been implemented. When all the coins are collected and the user reaches a total score of 2000, the game terminates.

Newer levels can then be added by building upon this very implementation by increasing the complexity of the floating maze, adding more gaps in the maze from where our Avatar can fall and adding more collectables and obstacles.

Hence, the part currently developed is the backbone of what this game has the potential to become, with most of the functionalities being implemented end to end, on a smaller scale.

<u>**Tech Stacks Used**</u>



TOOLS AND TECH STACKS USED

- Unity
- Blender
- C#
- Microsoft Visual Studio
- Unity Assets Store

Source Code

Filename – Camera.cs

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

public class Camera : MonoBehaviour
{
    [SerializeField]
    private float xMax;

    [SerializeField]
    private float xMin;

    [SerializeField]
    private float yMax;

    [SerializeField]
    private float yMin;

    private Transform playerTransform;

    //private Transform cameraTransform;

    // Start is called before the first frame update
    void Start()
    {
        playerTransform = GameObject.Find("Player").transform;
        //cameraTransform = GetComponent<Transform>();
    }

    // Update is called once per frame
    void LateUpdate()
    {
        transform.position = new Vector3(
Mathf.Clamp(playerTransform.position.x, xMin, xMax),
Mathf.Clamp(playerTransform.position.y, yMin, yMax), -10);
    }
}
```

Filename – Door.cs

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

public class Door : MonoBehaviour
```

```
{

      [SerializeField]
      private GameObject keyState;

      [SerializeField]
      private GameObject passwordState;

      [SerializeField]
      private GameObject doorOpen;

      [SerializeField]
      private AudioSource doorLockedSound;

      [SerializeField]
      private AudioSource doorOpenSound;

      private void OnTriggerEnter2D(Collider2D collider) {
            if (collider.gameObject.tag == "Player" && (keyState.activeSelf
|| passwordState.activeSelf) ) {
                  doorOpen.SetActive(true);
                  collider.gameObject.SetActive(false);
                  doorOpenSound.Play();
                  SceneManager.LoadScene(1);
            } else if (collider.gameObject.tag == "Player") {
                  doorLockedSound.Play();
            }
      }

    // Start is called before the first frame update
    void Start()
    {

    }

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

    }
```

Filename – Floor.cs

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

public class Floor : MonoBehaviour {

    private BoxCollider2D playerCollider;
//    [SerializeField]
    private BoxCollider2D collisionCollider;

    // Start is called before the first frame update
    void Start() {
        playerCollider = 
GameObject.Find("Player").GetComponent<BoxCollider2D>();
        collisionCollider = GetComponent<BoxCollider2D>();
    }

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

    }

    private void OnTriggerEnter2D(Collider2D collider) {
        if (collider.gameObject.name == "Player") {
            Physics2D.IgnoreCollision(collisionCollider, playerCollider, 
true);
        }
    }

    private void OnTriggerExit2D(Collider2D collider) {
        if (collider.gameObject.name == "Player") {
            Physics2D.IgnoreCollision(collisionCollider, playerCollider, 
false);
        }
    }
}
```

Filename – Player.cs

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

public class Player : MonoBehaviour
{

      private Rigidbody2D playerRigidBody;
      private Animator playerAnimator;

      [SerializeField]
      private float velocity;

      [SerializeField]
      private GameObject keyState;
```

```
     [SerializeField]
     private GameObject passwordState;

    [SerializeField]
    private AudioSource coinSound;

    [SerializeField]
    private AudioSource keySound;

    private bool direction;
    private int score;
    public Text totalScore;
    private bool attackState;
    private bool slideState;
    private bool jumpState;

    private float jumpPower;

    private const float gravity = 9.8f;

    private bool isOnTheFloor;

  void Start()
  {
        jumpPower = 0;

    score = 0;
    direction = true;

    playerRigidBody = GetComponent<Rigidbody2D>();

    playerAnimator = GetComponent<Animator>();
  }

    void Update() {
        checkKeyboardControls();
    }

  void FixedUpdate()
  {
      float horizontal = Input.GetAxis("Horizontal");

        basicMovements(horizontal);

      changeDirection(horizontal);

        playerRigidBody.gravityScale = jumpPower;

        if (jumpPower > gravity) {
              jumpPower -= playerRigidBody.mass;
```

```
                } else {
                      jumpPower = gravity;
                }

                resetValues();
        }

          private void checkKeyboardControls() {
                attackState = Input.GetKeyDown(KeyCode.T);

                slideState = Input.GetKeyDown(KeyCode.Y);

                jumpState = Input.GetKeyDown(KeyCode.U);
          }

        private void basicMovements(float horizontal) {

                if
    (!this.playerAnimator.GetCurrentAnimatorStateInfo(0).IsTag("Attack")) {


                      playerRigidBody.velocity = new Vector2(horizontal *
    velocity, playerRigidBody.velocity.y);

                      playerAnimator.SetFloat("playerVelocity",
    Mathf.Abs(horizontal));

                      if (attackState) {
                            playerAnimator.SetTrigger("attackState");
                            playerRigidBody.velocity = Vector2.zero;
                      }

                      if (slideState) {
                            playerAnimator.SetTrigger("slideState");
                      }

                      if (jumpState && isOnTheFloor) {

                            isOnTheFloor = false;

                            jumpPower = - playerRigidBody.gravityScale *
    velocity;
                            playerAnimator.SetTrigger("jumpState");
                      }

                }
        }

        private void changeDirection(float horizontal) {
```

```
      if (horizontal > 0 && direction == false || horizontal < 0 && direction
== true) {
            direction = !direction;
            Vector3 currentDirecton = transform.localScale;
            currentDirecton.x *= -1;
            transform.localScale = currentDirecton;
      }
    }

    private void OnCollisionEnter2D(Collision2D collision) {
      if (collision.gameObject.tag == "Coin") {
            collision.gameObject.SetActive(false);
            score = score + 100;
            updateScore(score);
            coinSound.Play();
      }

      if (collision.gameObject.tag == "Key") {
            collision.gameObject.SetActive(false);
            keyState.SetActive(true);
            keySound.Play();
      }

            if (collision.gameObject.tag == "Floor") {
                  isOnTheFloor = true;
            }

            if (collision.gameObject.name == "Box" &&
this.playerAnimator.GetCurrentAnimatorStateInfo(0).IsName("PlayerSlide")) {
                  collision.gameObject.SetActive(false);
                  passwordState.SetActive(true);
            }
    }

    private void updateScore(int count) {
      totalScore.text = "Score: " + count.ToString();
    }

      private void resetValues() {

      }
}
```

Description of Source Code

The ***Camera.cs*** file helps initialize the position of the camera on our canvas and sets the coordinates of the location. It then follows the player object, wherever it goes so as to give the user the best possible view of their surroundings. Initially, the camera object only shows the player when the game starts.

The ***Door.cs*** script programs the behavior of the door object, it initializes sounds, position and checks for collision between the player and the door object, the door object, with the attribute, passwordState stores whether or not the player is in possession of the key to that particular door.

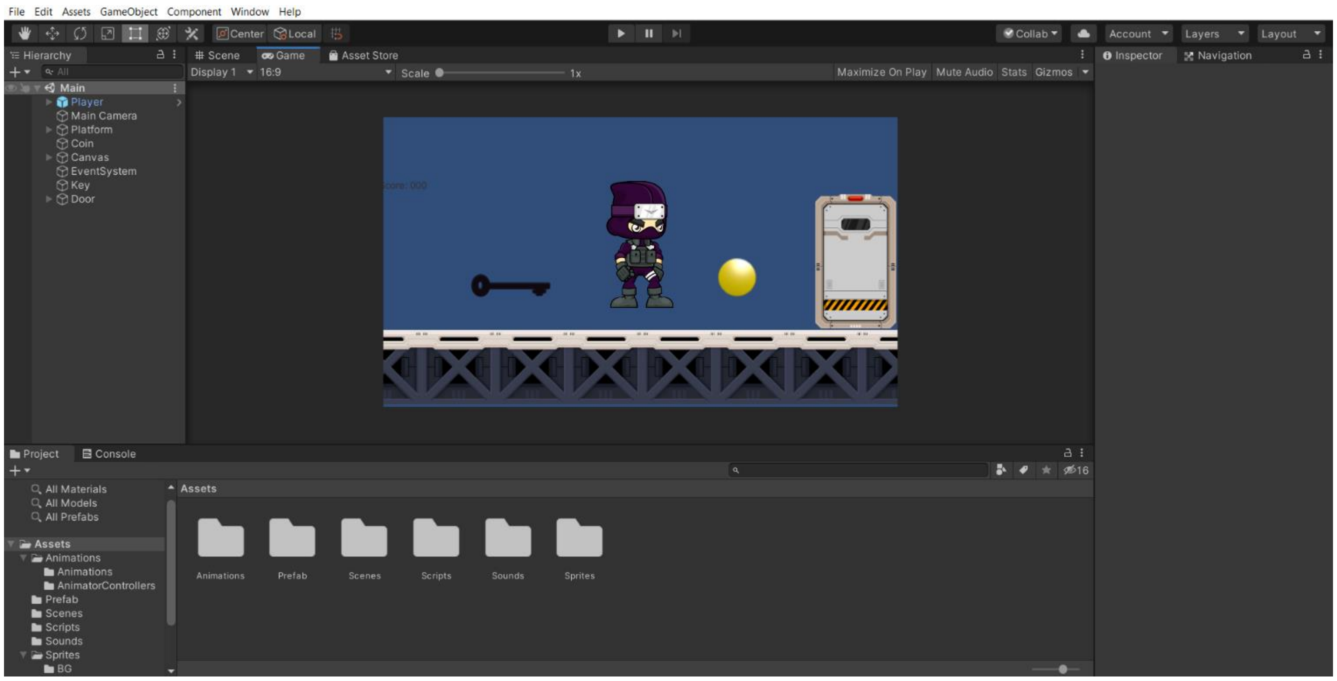
If the collision occurs while the passwordState is active, the door opens.

The ***Floor.cs*** implements the crux of our labyrinth, incorporating all the physics based entities like gravity, and checking whether or not our main player object is colliding with the maze floor or not, to check for walking, running and jump states. The gaps in the maze from where our protagonist can fall off are just empty spaces without any collision detection properties.

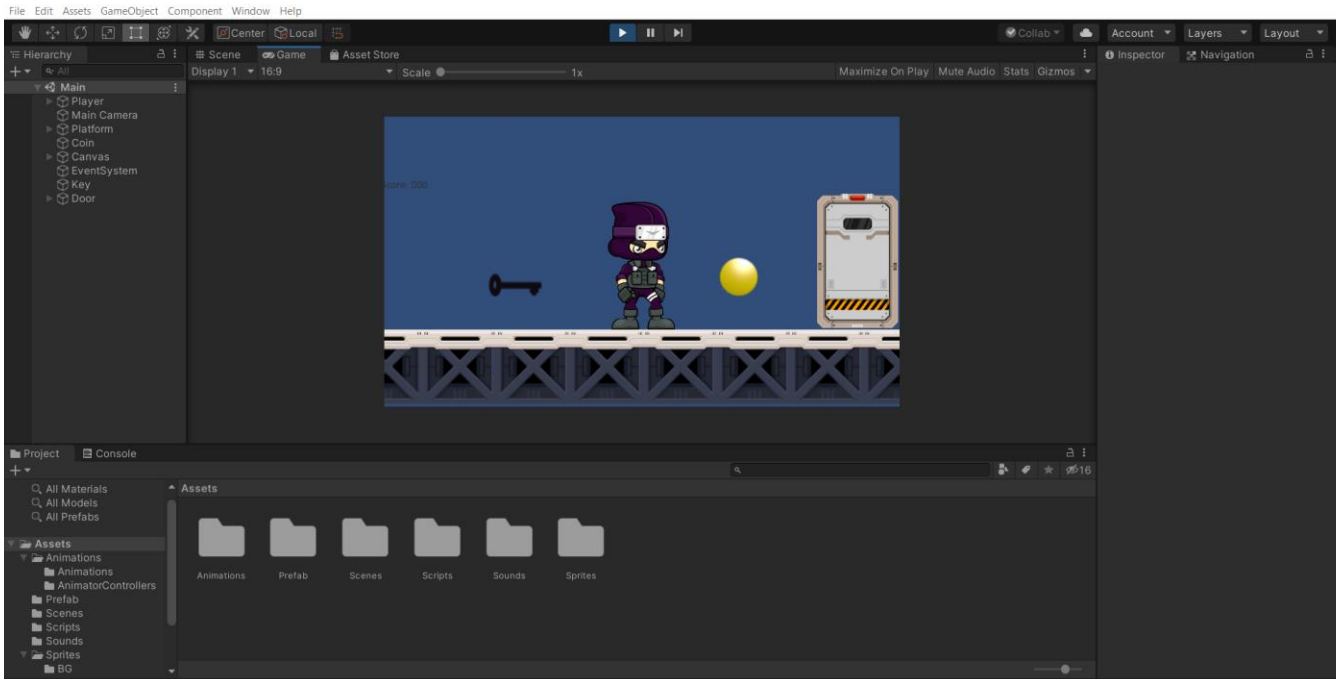
Lastly, the ***Player.cs*** file incorporates ALL the functionalities and states of our player Avatar object, Jack, it animates the player, initializes movement direction and speed, incorporates sounds based on the action the player performs and keeps track of the collectables the player accumulates.

It also tracks various movements and maps them to states and subsequent keystrokes. The basic animations, physics based forces (gravity and collision detection) and other controls (changing directions, performing attacks) are also controlled from this very script. The scoreboard functionality and keeping track of all the collectables amassed by the player is also calculated and stored in an array, which is then converted to string, displayed on the scoreboard and compared a particular threshold value required for level completion.

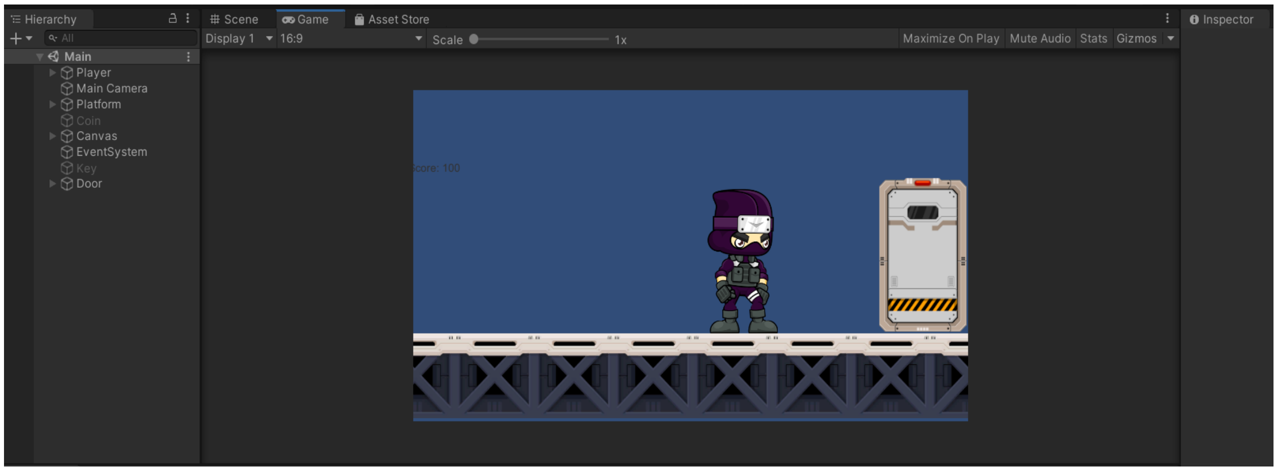
Snapshots of Gameplay

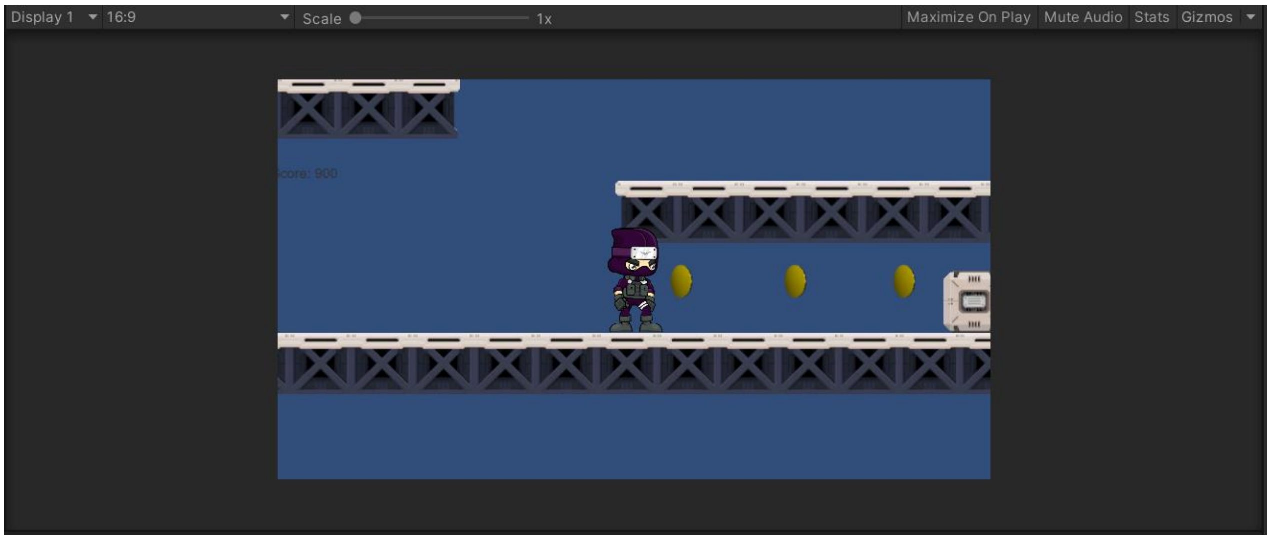


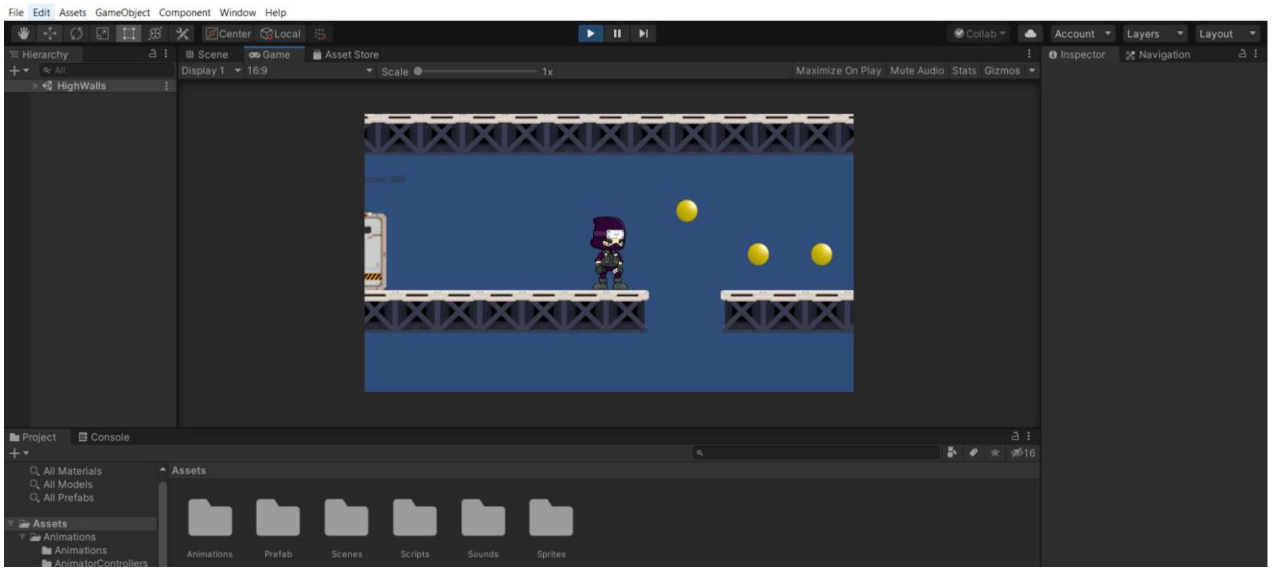
Start screen and the beginning of the game.

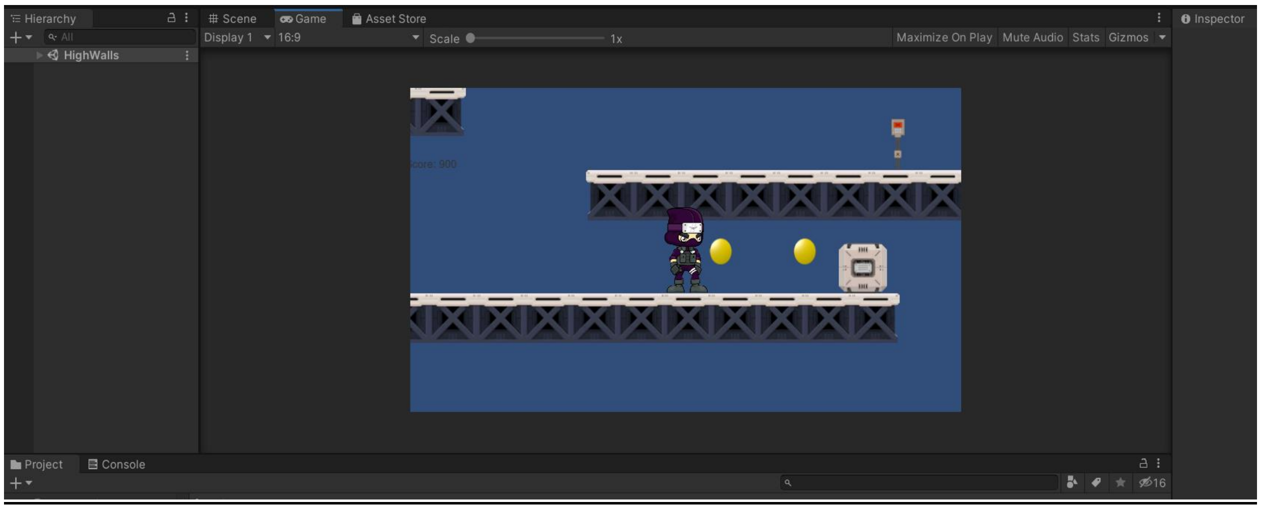


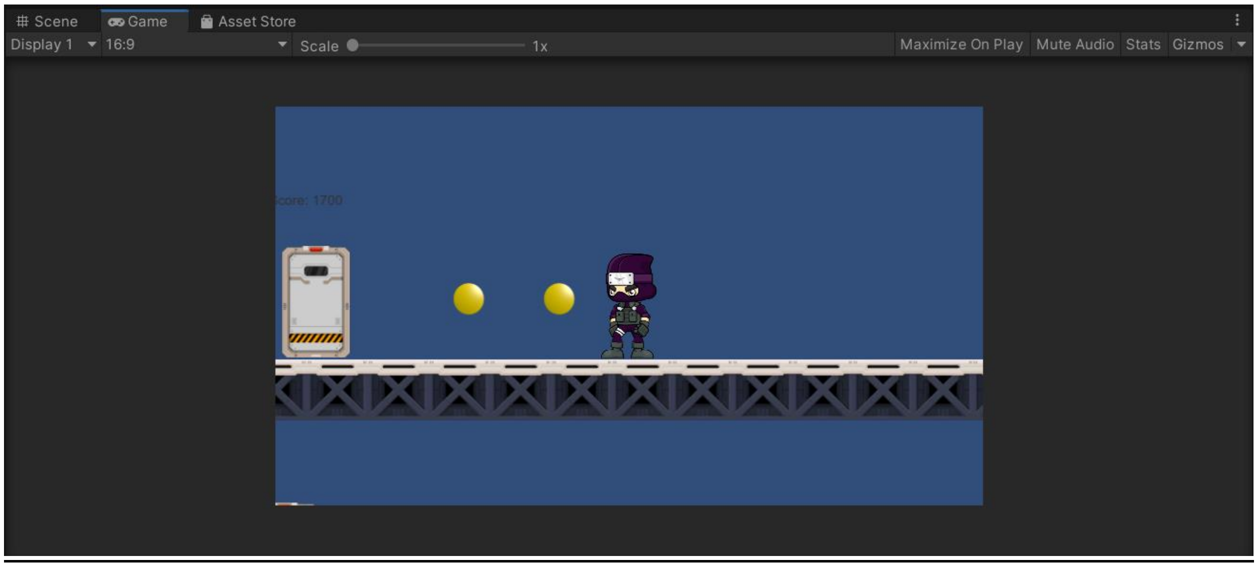
Subsequent Gameplay Screenshots

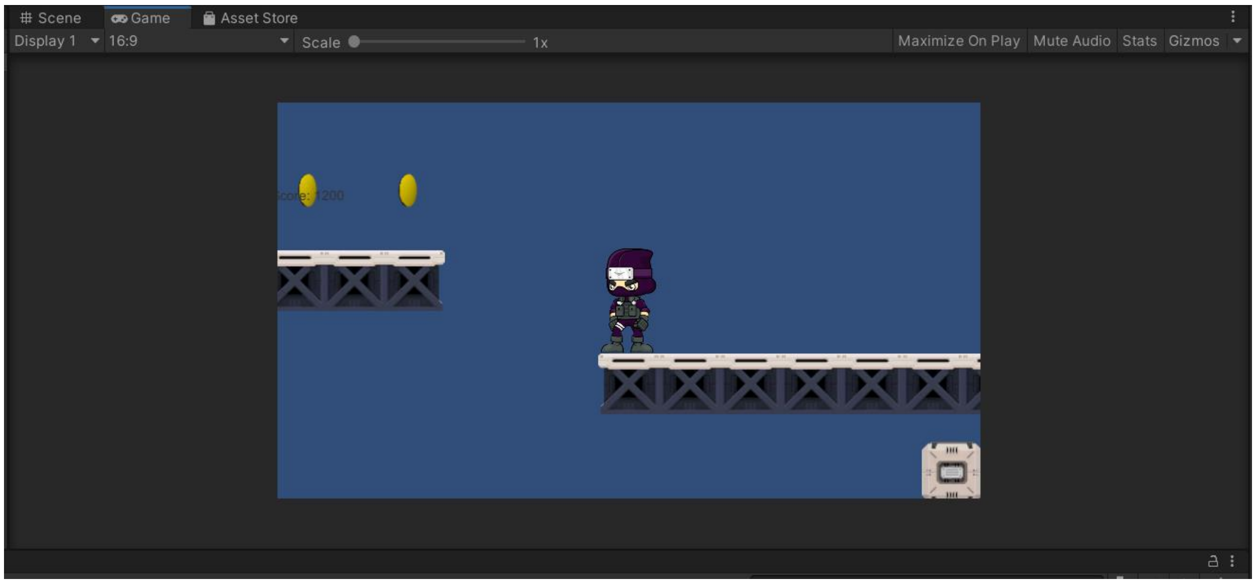

Hierarchy
Scene
Game
Asset Store
Inspector
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
Main
Player
Main Camera
Platform
Coin
Canvas
EventSystem
Key
Door
Score: 100

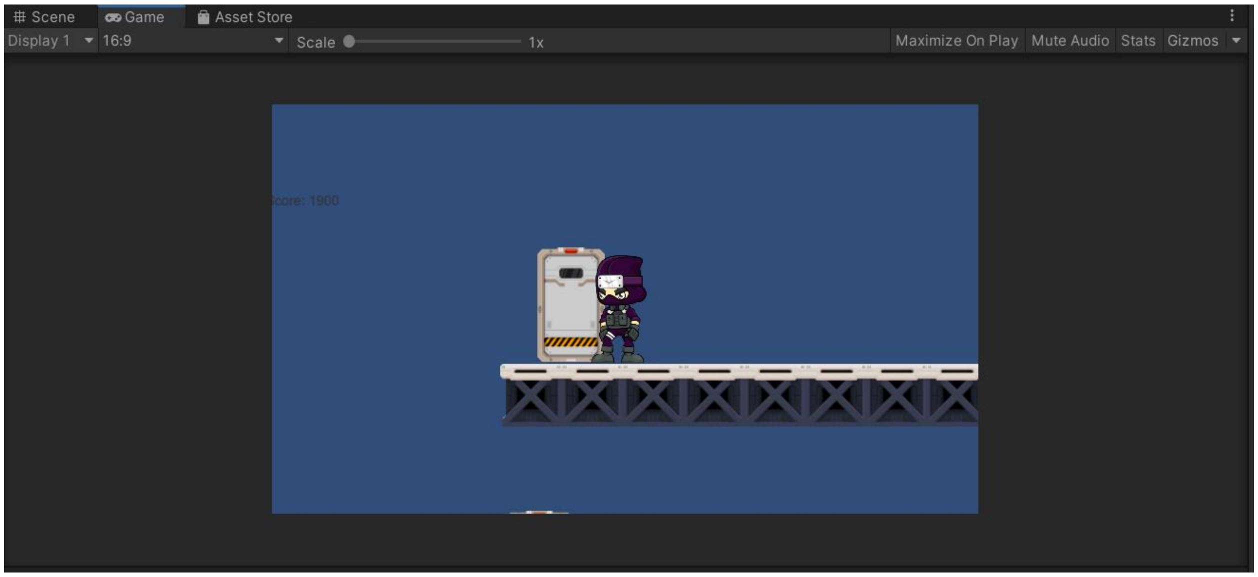

Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
Score: 900


File
Edit
Assets
GameObject
Component
Window
Help
Center
Local
Collab
Account
Layers
Layout
Hierarchy
Scene
Game
Asset Store
Inspector
Navigation
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
HighWalls
Project
Console
All Materials
All Models
All Prefabs
Assets
Animations
Animations
AnimatorControllers
Assets
Animations
Prefab
Scenes
Scripts
Sounds
Sprites
16


Hierarchy
HighWalls
Scene
Game
Asset Store
Inspector
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
core: 900
Project
Console
16


Scene
Game
Asset Store
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
core: 1700


Scene
Game
Asset Store
Display 1
16:9
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmos
core: 1200



Comparative Analysis

The game, Samurai Dash, is a self-made venture and an attempt to capitalize on the popularity of endless runner games, which with their unique idea and concept, have taken the game industry by storm.

Currently, many such games exist in the market, the most popular one being temple run and subway surfers. In most such games, the player can just see the part right ahead of them, instead of the zoomed out version of the map and once a particular collectable is missed unintentionally, the player cannot go back. Our game tackles this predicament by improving upon the design and adding the following features:

I. It implements the best of both, endless runner games, and maze based games, by allowing users to see a limited part of the map (unlike endless runner games that don't do so, and maze based games where one can often see the complete labyrinth). This makes sure that the user can see where the collectables and the keys lie so that they can tread down that path.
II. It allows the player to back trace their steps which is not seen in most endless runner games.

III. It is can be easily up scaled to add more and more levels of increasing difficulty following the same template.
IV. The UI is easier to comprehend and grasp for newbies which makes sure they get a hang of the game seamlessly.

The above are the design innovations incorporated by me in samurai dash that make it different from its existing counterparts.

Source Code

The source code of the entire game, along with the gameplay illustration and other assets used can be found in the GitHub repository mentioned below:

https://github.com/anmolpant/Samurai-Dash

* * *