

**Contents**

* Executive summary
* Objective of game
* Gameplay
* Design of game
* Research
* Appendix

**Executive Summary**

Our game’s storyline is such that a patient from a mental hospital is making a break and trying to escape the very place that was supposed to make him better. A lot of what is shown in the game is how he views the world and how being neurodivergent, he has a different perception of what things are to him compared to how someone without any mental illnesses would view things.

He first interacts with a voice in his head, Zong Han, who claims to be a genie and says that it seems that the player is lost in the desert. This hints to deplorable conditions that the player has been enduring in the mental hospital and wants to escape from. The “genie” seems to be there to “guide” the player through his dangerous adventures outside of the mental hospital.

Paradise is promised after the escape from the mental hospital.

The “genie” tells the player to enjoy the candy along the way, so the player expects it as part of the game play. He is also told to avoid crates in his path, which serves as obstacles in this path to escape the terrible place he was in.

The “genie” finally tells the player to get on the rainbow road urgently to quickly escape the place. The player then gets to choose between taking a straight route, or a rotated path, to escape from the mental hospital.

These represents different states of his mind, where sometimes life is easier and he feels normal, but when he has an episode, the paths start to bend and become more difficult for him to escape imminent death.

**Objective of Game**

Our project involved making a computer game using Unity software. We looked at the game manual provided on NTULearn, as well as perused the Unity User Manual, and countless online tutorials. A lot of self-learning was involved. We were inspired from running games like “Minion Rush” and “Temple Run” and wanted to create an equally exciting game.

These games have a 3rd person camera following the main character as he runs forward in a 3D environment, while he tries to avoid various hazardous objects that kill him upon collision. The player is controlled by moving left and right to avoid these objects.

**Gameplay**

At the start of the game, the player clicks on and triggers a conversation to talk to a non-playable character(NPC) called Zong Han. This will start a dialogue with the NPC, which will go through the storyline of the game, as well as the instructions in the game.

After the dialogue, the player can see a screen with two simple buttons. He can choose one of the two game levels, either the “rotated paths” level or the “straight paths” level.

In the “rotated paths” level, the character follows a narrow platform (“rainbow road”) which goes on forever. While the player is running, the game engine randomly chooses where to place the next platform, either left, right, or straight ahead. The point where this next path will be placed is at the end of the current path. When the character reaches the current path’s end, the player must press the arrow keys in order to go left or right, or risk continuing on the main path. If player does not move in time, then Character will collide with the wall and die (the red walls).

When walking on the path, the character can pick the candy that appears in front of him to get points and increase his score. He must jump to avoid obstacles and, of course, he can move left or right when the platform is about to rotate in order to follow the new path.

This game has two input methods (either arrow keys or swipe in a touch screen). In the “straight path” level, the character follows a wide platform and continuously moves at a straight direction. Player can swipe left or right to move sideways, and he can also pick up candy to increase his score. He also must avoid obstacles (crates) in his path either by moving to a different lane or by jumping over them. If the character hits an obstacle, he dies, and the game is over.

In both cases/levels, the game can theoretically continue indefinitely. The game is over when the character crashes onto a red wall (rotated paths level) or collides with an obstacle (both levels). When this happens, the player can tap the screen to restart the game. The score increases as the character continues running.

This is also one way to tell who the winner of the game is, should two people play. The one with the highest score would win. We had originally wanted to have a multiplayer function where two players, using two different computers, would be able to interact and play the game on a server. However, due to time constraints, we were not able to achieve this in time for the showcase.

As part of the storyline, we had also wanted to show another NPC (mental hospital staff) chasing the player as he made his escape. Unfortunately, due to the lack of time, we were not able to fix the bugs in time for the showcase as well.

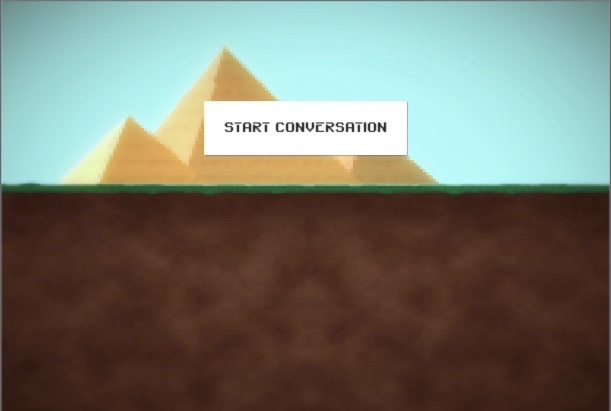
**Screenshots of how the game flows:**

**Dialogue**

**Scene 1**



**Scene 2**



**Scene 3**



**Scene 4**



**Scene 5**



**Scene 6**



**Scene 7**



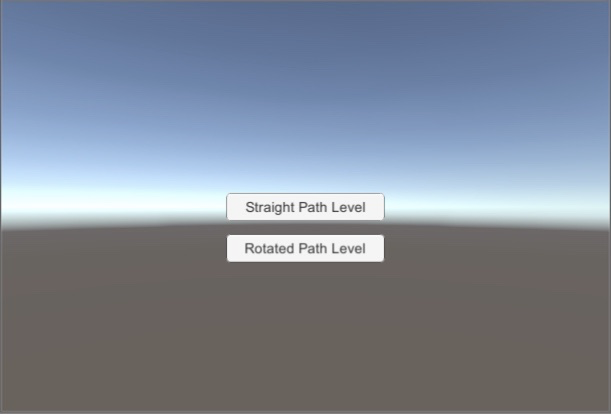
**Scene 8**



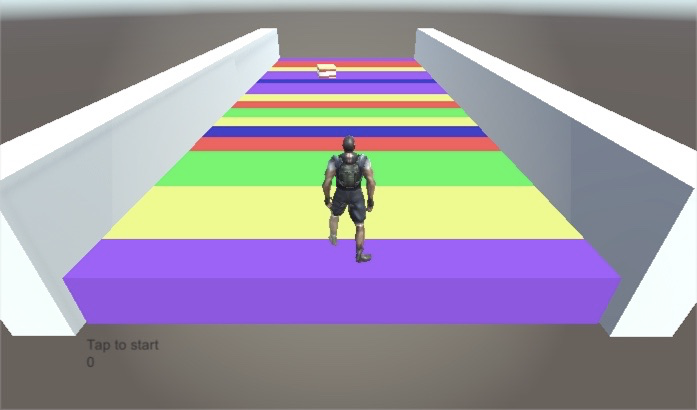
**Scene 9**



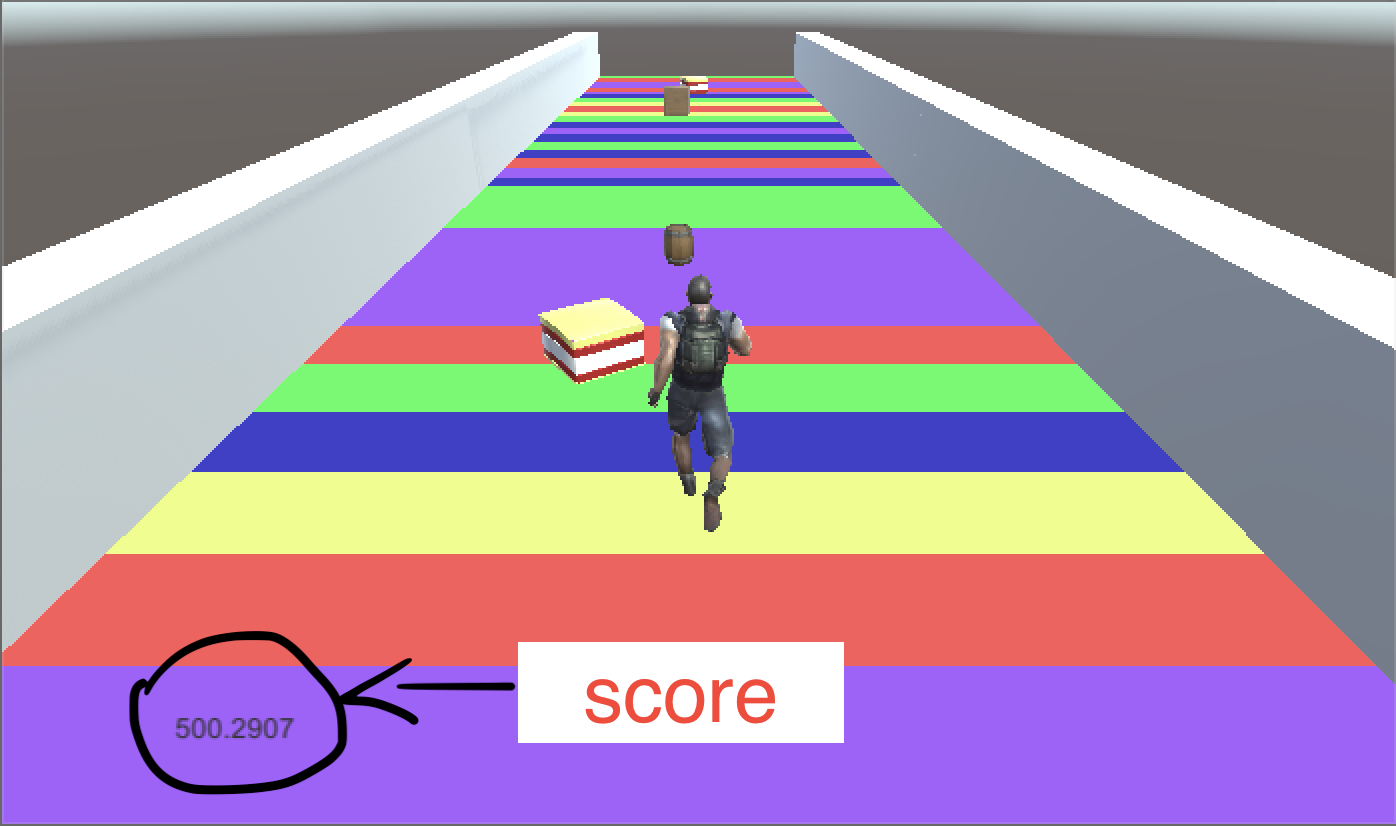
**Scene 10: Menu selection for straight path or rotated path**



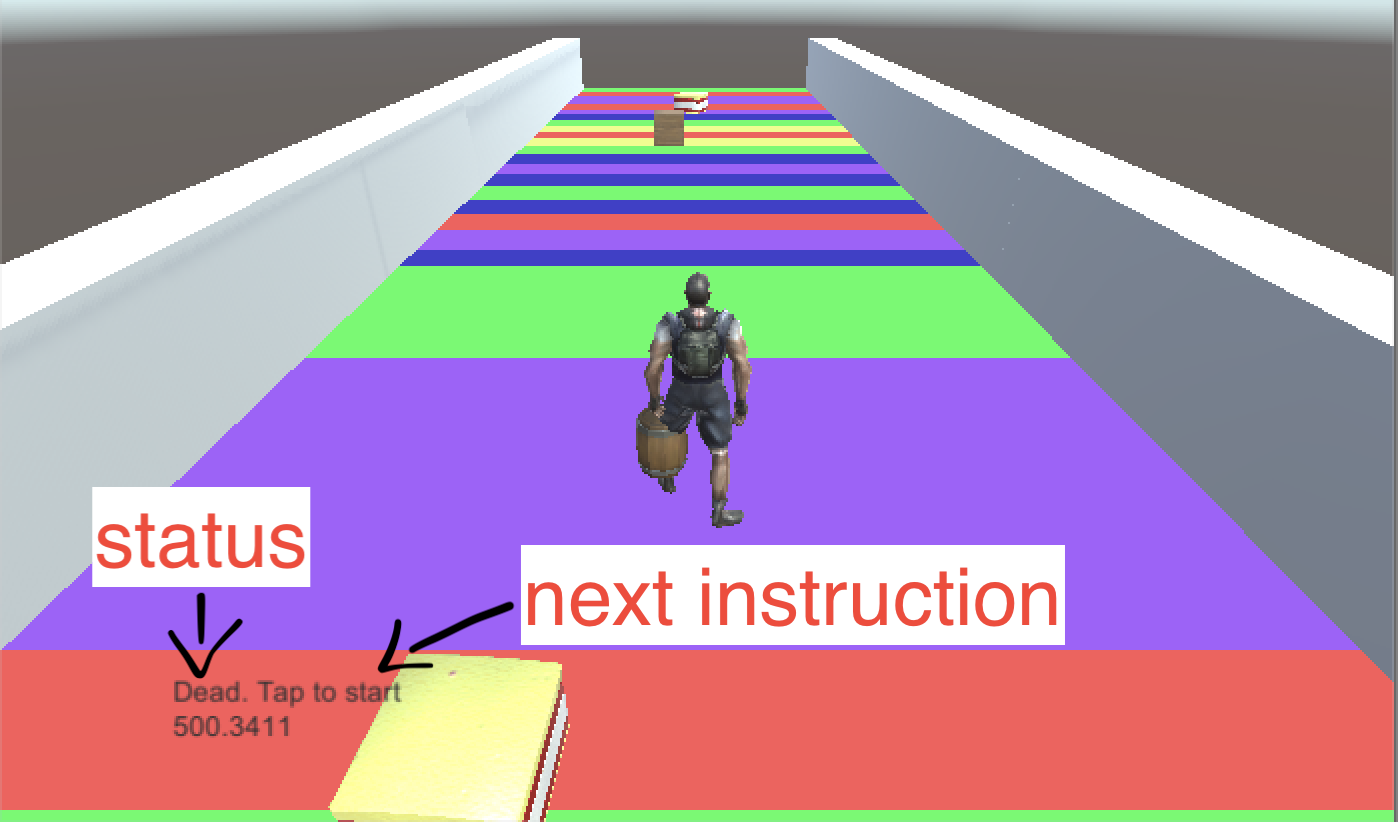
**Scene 11: Straight path**



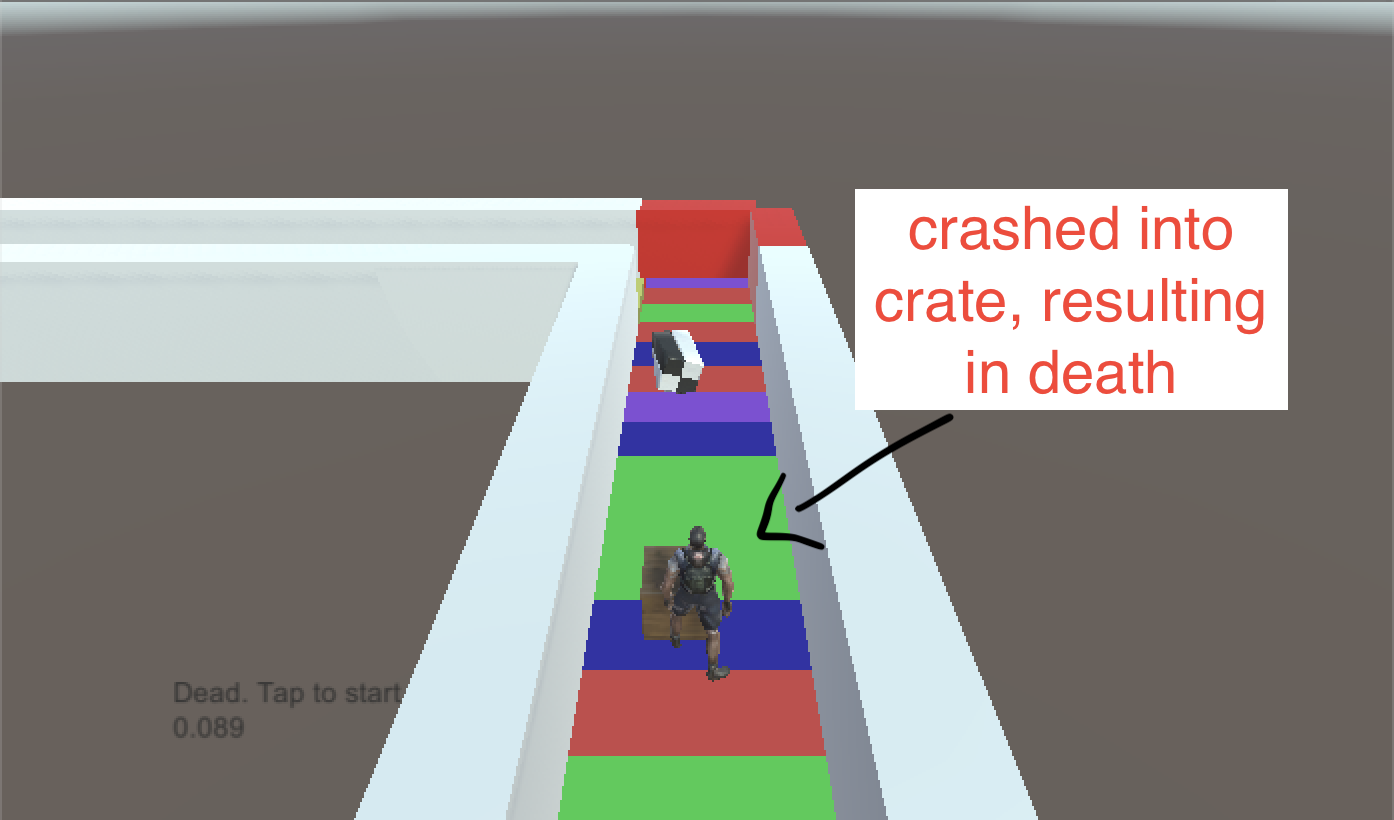
**Scene 12**



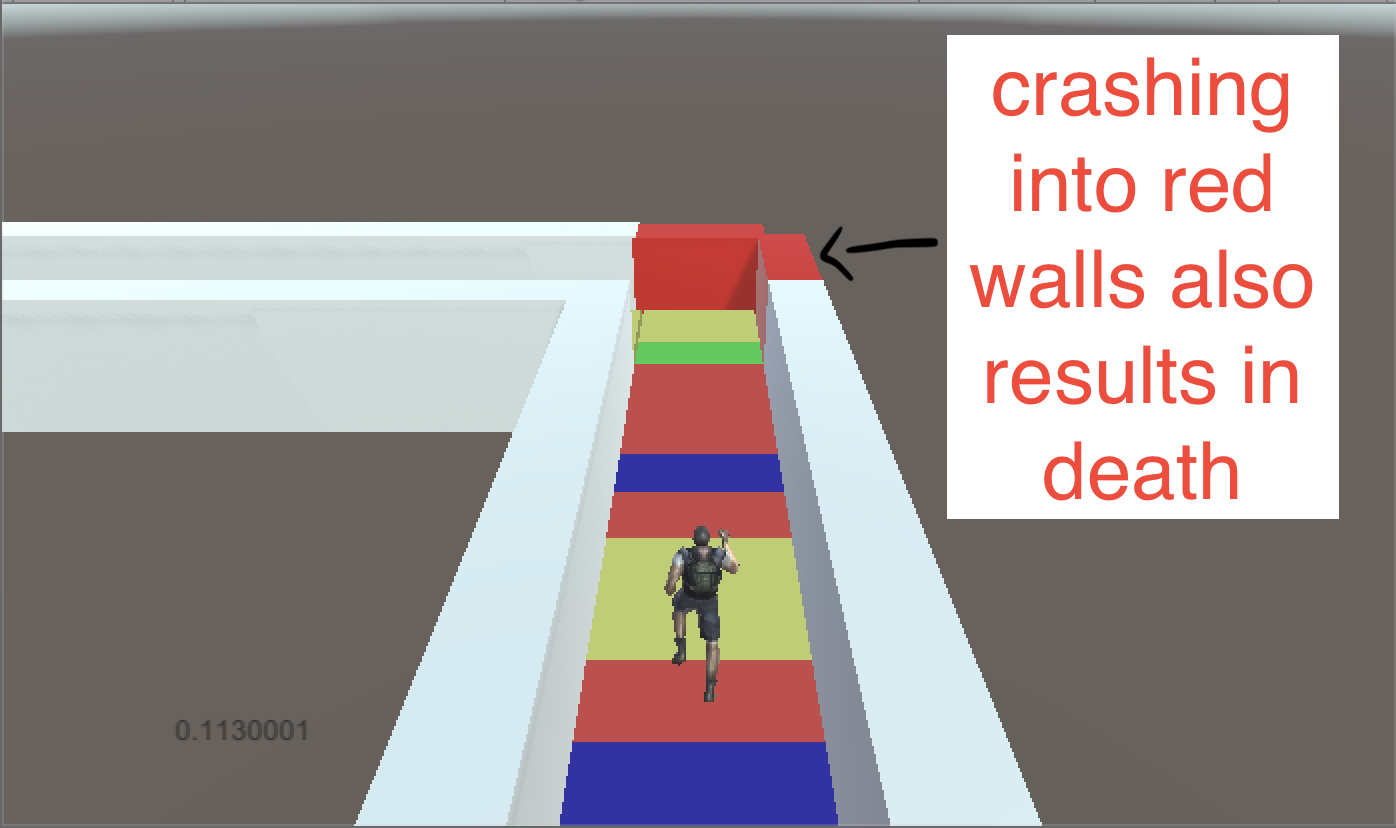
**Scene 13**



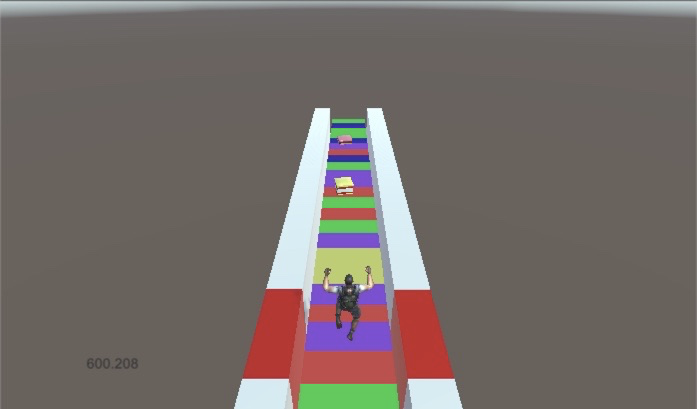
**Scene 14**



**Scene 15**



**Scene 16**



**Game design**

These are the assets we downloaded for the game design.

* All Sorts Candy <https://www.assetstore.unity3d.com/en/#!/content/12512> Player will collect the candy to increase his score.
* Max Adventure Model <https://www.assetstore.unity3d.com/en/#!/content/3012> The main character of our game.
* Campfire Pack <https://www.assetstore.unity3d.com/en/#!/content/11256> We borrowed two models from this pack to create some obstacles for our player.

**Why the Rainbow road?**

The rainbow road inspiration came from Mario Kart. We selected random colors for the rectangle shapes on the path’s floor. This is what this class is used for.

The materials used were blue, green, purple, red, yellow and white.

The code can be found in the random material spawner below in the appendix.

The straight path game object which, by nature, is wider than the one in the “rotated paths” level. The path is a prefab (as designated by its blue color) and all subsequent paths will be clones of this one. Scene also contains some cubical shaped 3D objects (named Cubes), which act as floor of our path. It also contains the NewPathSpawn game object (a simple location in the 3D space), and some other locations called SpawnPoints. The SpawnPoints are positions in which new stuff will be generated. This stuff can be either a candy, or obstacle, or nothing at all.

The character runs from path to path. As we are not generating all paths at the beginning of the game (because we want to save on memory and mainly because we do not know how far in the game the player will proceed, i.e. how many paths to generate), we need a mechanism to generate the N+1 path, where N is the path that the character currently steps on. We’ve used a simple trigger BoxCollider to implement this. When the character collides with it, a new path is generated via the PathSpawnCollider script (described in a while). In the straight paths level, the new path is instantiated in the “NewPathSpawn” position, which conveniently happens to be positioned at the far end of the current path.

**Research**

During this project, we gained a lot of knowledge by watching and re-watching countless Youtube tutorials.

Our most referred to tutorials were by a Youtuber called Brackeys. His videos featured game creation and design. The prominent features in this game is the infinite runner sequence well as the dialogue at the start. These skills were picked up from his tutorial videos.

Link to his channel down below:

<https://www.youtube.com/user/Brackeys>

To learn about how to animate a character, we read the unity tutorial online,(https://unity3d.com/learn/tutorials/s/animation) and watched tutorials like these: <https://www.youtube.com/watch?v=ts24UWC0mY4> to learn about the fundamentals of animation and how to control movement.

For the NPC part, we referred to this Unity tutorial: <https://www.youtube.com/watch?v=zssU0MZcIx8> by Jimmy Vegas, and tried to change it from the first person Point of view (as taught in the tutorial) to a third person Point of view. However, we encountered several issues and were unable to feature an NPC chasing the player in the game.

Regarding the multiplayer component, we referred to a popular youtuber Quill18’s tutorial videos. However, due to time constraints as well, we were unable to get it up and running in time for the showcase.

We relied a lot on Google for debugging help, either on stack overflow or computer science forums. We also read the Unity documentations to try and fix errors we had. There was a lot of trial and error especially regarding creating animations on Unity, as

Our game underwent quite a few drastic changes and evolved along the way before becoming the final game. As such, we have learnt a lot from this project.

**Appendix**

Candy

using UnityEngine;  
 using System.Collections;  
 using Assets.Scripts;  
   
public class Candy : MonoBehaviour  
 {  
 //candy found in https://www.assetstore.unity3d.com/en/#!/content/12512  
   
 // Update is called once per frame  
 void Update()  
 {  
 transform.Rotate(Vector3.up, Time.deltaTime \* rotateSpeed);  
 }  
   
 void OnTriggerEnter(Collider col)  
 {  
 UIManager.Instance.IncreaseScore(ScorePoints);  
 Destroy(this.gameObject);  
 }  
   
 public int ScorePoints = 100;  
 public float rotateSpeed = 50f;  
 }

Character rotate movement

using UnityEngine;  
 using Assets.Scripts;  
 using UnityEngine.SceneManagement;  
   
  
public class CharacterRotateMovement : MonoBehaviour  
 {  
 //character model found in https://www.assetstore.unity3d.com/en/#!/content/3012  
   
 private Vector3 moveDirection = Vector3.zero;  
 public float gravity = 20f;  
 private CharacterController controller;  
 private Animator anim;  
   
 public float JumpSpeed = 8.0f;  
 public float Speed = 6.0f;  
 public Transform CharacterGO;  
   
 bool isInSwipeArea;  
   
  
 IInputDetector inputDetector = null;  
   
 // Use this for initialization  
 void Start()  
 {  
 moveDirection = transform.forward;  
 moveDirection = transform.TransformDirection(moveDirection);  
 moveDirection \*= Speed;  
   
 UIManager.Instance.ResetScore();  
 UIManager.Instance.SetStatus(Constants.StatusTapToStart);  
   
 GameManager.Instance.GameState = GameState.Start;  
   
 anim = CharacterGO.GetComponent<Animator>();  
 inputDetector = GetComponent<IInputDetector>();  
 controller = GetComponent<CharacterController>();  
 }  
   
 // Update is called once per frame  
 void Update()  
 {  
 switch (GameManager.Instance.GameState)  
 {  
 case GameState.Start:  
 if (Input.GetMouseButtonUp(0))  
 {  
 anim.SetBool(Constants.AnimationStarted, true);  
 var instance = GameManager.Instance;  
 instance.GameState = GameState.Playing;  
   
 UIManager.Instance.SetStatus(string.Empty);  
 }  
 break;  
 case GameState.Playing:  
 UIManager.Instance.IncreaseScore(0.001f);  
   
 CheckHeight();  
   
 DetectJumpOrSwipeLeftRight();  
   
 //apply gravity  
 moveDirection.y -= gravity \* Time.deltaTime;  
 //move the player  
 controller.Move(moveDirection \* Time.deltaTime);  
   
 break;  
 case GameState.Dead:  
 anim.SetBool(Constants.AnimationStarted, false);  
 if (Input.GetMouseButtonUp(0))  
 {  
 //restart  
 SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);  
 }  
 break;  
 default:  
 break;  
 }  
   
 }  
   
 private void CheckHeight()  
 {  
 if (transform.position.y < -10)  
 {  
 GameManager.Instance.Die();  
 }  
 }  
   
 private void DetectJumpOrSwipeLeftRight()  
 {  
 var inputDirection = inputDetector.DetectInputDirection();  
 if (controller.isGrounded && inputDirection.HasValue && inputDirection == InputDirection.Top)  
 {  
 moveDirection.y = JumpSpeed;  
 anim.SetBool(Constants.AnimationJump, true);  
 }  
 else  
 {  
 anim.SetBool(Constants.AnimationJump, false);  
 }  
   
  
 if (GameManager.Instance.CanSwipe && inputDirection.HasValue &&  
 controller.isGrounded && inputDirection == InputDirection.Right)  
 {  
 transform.Rotate(0, 90, 0);  
 moveDirection = Quaternion.AngleAxis(90, Vector3.up) \* moveDirection;  
 //allow the user to swipe once per swipe location  
 GameManager.Instance.CanSwipe = false;  
 }  
 else if (GameManager.Instance.CanSwipe && inputDirection.HasValue &&  
 controller.isGrounded && inputDirection == InputDirection.Left)  
 {  
 transform.Rotate(0, -90, 0);  
 moveDirection = Quaternion.AngleAxis(-90, Vector3.up) \* moveDirection;  
 GameManager.Instance.CanSwipe = false;  
 }  
   
 }  
   
}

Character sideways movement

using UnityEngine;  
 using Assets.Scripts;  
 using UnityEngine.SceneManagement;  
 using System.Collections;  
   
public class CharacterSidewaysMovement : MonoBehaviour  
 {  
   
  
 private Vector3 moveDirection = Vector3.zero;  
 public float gravity = 20f;  
 private CharacterController controller;  
 private Animator anim;  
   
 private bool isChangingLane = false;  
 private Vector3 locationAfterChangingLane;  
 //distance character will move sideways  
 private Vector3 sidewaysMovementDistance = Vector3.right \* 2f;  
   
 public float SideWaysSpeed = 5.0f;  
   
 public float JumpSpeed = 8.0f;  
 public float Speed = 6.0f;  
 //Max gameobject  
 public Transform CharacterGO;  
   
 IInputDetector inputDetector = null;  
   
 // Use this for initialization  
 void Start()  
 {  
 moveDirection = transform.forward;  
 moveDirection = transform.TransformDirection(moveDirection);  
 moveDirection \*= Speed;  
   
 UIManager.Instance.ResetScore();  
 UIManager.Instance.SetStatus(Constants.StatusTapToStart);  
   
 GameManager.Instance.GameState = GameState.Start;  
   
 anim = CharacterGO.GetComponent<Animator>();  
 inputDetector = GetComponent<IInputDetector>();  
 controller = GetComponent<CharacterController>();  
 }  
   
 // Update is called once per frame  
 void Update()  
 {  
 switch (GameManager.Instance.GameState)  
 {  
 case GameState.Start:  
 if (Input.GetMouseButtonUp(0))  
 {  
 anim.SetBool(Constants.AnimationStarted, true);  
 var instance = GameManager.Instance;  
 instance.GameState = GameState.Playing;  
   
 UIManager.Instance.SetStatus(string.Empty);  
 }  
 break;  
 case GameState.Playing:  
 UIManager.Instance.IncreaseScore(0.001f);  
   
 CheckHeight();  
   
 DetectJumpOrSwipeLeftRight();  
   
 //apply gravity  
 moveDirection.y -= gravity \* Time.deltaTime;  
   
 if (isChangingLane)  
 {  
 if (Mathf.Abs(transform.position.x - locationAfterChangingLane.x) < 0.1f)  
 {  
 isChangingLane = false;  
 moveDirection.x = 0;  
 }  
 }  
   
 //move the player  
 controller.Move(moveDirection \* Time.deltaTime);  
   
 break;  
 case GameState.Dead:  
 anim.SetBool(Constants.AnimationStarted, false);  
 if (Input.GetMouseButtonUp(0))  
 {  
 //restart  
 SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);  
 }  
 break;  
 default:  
 break;  
 }  
   
 }  
   
 private void CheckHeight()  
 {  
 if (transform.position.y < -10)  
 {  
 GameManager.Instance.Die();  
 }  
 }  
   
 private void DetectJumpOrSwipeLeftRight()  
 {  
 var inputDirection = inputDetector.DetectInputDirection();  
 if (controller.isGrounded && inputDirection.HasValue && inputDirection == InputDirection.Top  
 && !isChangingLane)  
 {  
 moveDirection.y = JumpSpeed;  
 anim.SetBool(Constants.AnimationJump, true);  
 }  
 else  
 {  
 anim.SetBool(Constants.AnimationJump, false);  
 }  
   
  
 if (controller.isGrounded && inputDirection.HasValue && !isChangingLane)  
 {  
 isChangingLane = true;  
   
 if (inputDirection == InputDirection.Left)  
 {  
 locationAfterChangingLane = transform.position - sidewaysMovementDistance;  
 moveDirection.x = -SideWaysSpeed;  
 }  
 else if (inputDirection == InputDirection.Right)  
 {  
 locationAfterChangingLane = transform.position + sidewaysMovementDistance;  
 moveDirection.x = SideWaysSpeed;  
 }  
 }  
   
  
 }  
   
 public void OnControllerColliderHit(ControllerColliderHit hit)  
 {  
 //if we hit the left or right border  
 if(hit.gameObject.tag == Constants.WidePathBorderTag)  
 {  
 isChangingLane = false;  
 moveDirection.x = 0;  
 }  
 }  
   
   
   
}

constants

/// <summary>  
 /// Helpful constants  
 /// </summary>  
 public static class Constants  
 {  
 public static readonly string PlayerTag = "Player";  
 public static readonly string AnimationStarted = "started";  
 public static readonly string AnimationJump = "jump";  
 public static readonly string WidePathBorderTag = "WidePathBorder";  
   
 /// <summary>  
 /// some UI text  
 /// </summary>  
 public static readonly string StatusTapToStart = "Tap to start";  
 public static readonly string StatusDeadTapToStart = "Dead. Tap to start";  
 }

dialogue

using System.Collections;  
 using System.Collections.Generic;  
 using UnityEngine;  
   
[System.Serializable]  
 public class Dialogue {  
 [SerializeField]  
 public string name;  
 [SerializeField]  
 [TextArea(3, 10)]  
 public string[] sentences;  
   
}

Dialogue Manager

using System.Collections;  
 using System.Collections.Generic;  
 using UnityEngine;  
 using UnityEngine.UI;  
 using UnityEngine.SceneManagement;  
   
public class DialogueManager : MonoBehaviour {  
   
 public Text nameText;  
 public Text dialogueText;  
   
 public Animator animator;  
 [SerializeField]  
 private Queue<string> sentences;  
   
 // Use this for initialization  
 void Start () {  
   
 sentences = new Queue<string>();  
 }  
   
 public void StartDialogue (Dialogue dialogue)  
 {  
 animator.SetBool("IsOpen", true);  
   
 nameText.text = dialogue.name;  
   
 sentences.Clear();  
   
 foreach (string sentence in dialogue.sentences)  
 {  
 sentences.Enqueue(sentence);  
 }  
   
 DisplayNextSentence();  
 }  
   
 public void DisplayNextSentence ()  
 {  
 if (sentences.Count == 0)  
 {  
 EndDialogue();  
 SceneManager.LoadScene("IntroLevel");  
 return;  
 }  
   
 string sentence = sentences.Dequeue();  
 StopAllCoroutines();  
 StartCoroutine(TypeSentence(sentence));  
 }  
   
 IEnumerator TypeSentence (string sentence)  
 {  
 dialogueText.text = "";  
 foreach (char letter in sentence.ToCharArray())  
 {  
 dialogueText.text += letter;  
 yield return null;  
 }  
 }  
   
 void EndDialogue()  
 {  
 animator.SetBool("IsOpen", false);  
   
 }  
   
}

Game manager

using System;  
 using System.Collections.Generic;  
 using System.Linq;  
 using System.Text;  
 using UnityEngine;  
   
public class GameManager : MonoBehaviour  
 {  
 void Awake()  
 {  
 if (instance == null)  
 {  
 instance = this;  
 }  
 else  
 {  
 DestroyImmediate(this);  
 }  
 }  
   
   
 private static GameManager instance;  
 public static GameManager Instance  
 {  
 get  
 {  
 if (instance == null)  
 {  
 instance = new GameManager();  
 }  
 return instance;  
 }  
 }  
   
 protected GameManager()  
 {  
 GameState = GameState.Start;  
 CanSwipe = false;  
 }  
   
 public GameState GameState { get; set; }  
   
 public bool CanSwipe { get; set; }  
   
 public void Die()  
 {  
 UIManager.Instance.SetStatus(Constants.StatusDeadTapToStart);  
 this.GameState = GameState.Dead;   
 }  
   
}

Game stats

public enum GameState  
 {  
 Start,  
 Playing,  
 Dead  
 }

Intro level

using UnityEngine;  
 using System.Collections;  
 using UnityEngine.SceneManagement;  
   
public class IntroLevel : MonoBehaviour {  
   
 public void StraightLevelClick()  
 {  
 SceneManager.LoadScene("straightPathsLevel");  
 }  
   
 public void RotatedLevelClick()  
 {  
 SceneManager.LoadScene("rotatedPathsLevel");  
 }  
 }

Obstacle

using UnityEngine;  
 using System.Collections;  
 using Assets.Scripts;  
   
public class Obstacle : MonoBehaviour {  
   
 //box and barrel found here: https://www.assetstore.unity3d.com/en/#!/content/11256  
   
 void OnTriggerEnter(Collider col)  
 {  
 //if the player hits one obstacle, it's game over  
 if(col.gameObject.tag == Constants.PlayerTag)  
 {  
 GameManager.Instance.Die();  
 }  
 }  
 }

Path spawn collider

using UnityEngine;  
 using System.Collections;  
   
public class PathSpawnCollider : MonoBehaviour {  
   
 public float positionY = 0.81f;  
 public Transform[] PathSpawnPoints;  
 public GameObject Path;  
 public GameObject DangerousBorder;  
   
 void OnTriggerEnter(Collider hit)  
 {  
 //player has hit the collider  
 if (hit.gameObject.tag == Constants.PlayerTag)  
 {  
 //find whether the next path will be straight, left or right  
 int randomSpawnPoint = Random.Range(0, PathSpawnPoints.Length);  
 for (int i = 0; i < PathSpawnPoints.Length; i++)  
 {  
 //instantiate the path, on the set rotation  
 if (i == randomSpawnPoint)  
 Instantiate(Path, PathSpawnPoints[i].position, PathSpawnPoints[i].rotation);  
 else  
 {  
 //instantiate the border, but rotate it 90 degrees first  
 Vector3 rotation = PathSpawnPoints[i].rotation.eulerAngles;  
 rotation.y += 90;  
 Vector3 position = PathSpawnPoints[i].position;  
 position.y += positionY;  
 Instantiate(DangerousBorder, position, Quaternion.Euler(rotation));  
 }  
 }  
   
 }  
 }  
   
}

Random material  
   
using UnityEngine;  
 using System.Collections;  
   
public class RandomMaterial : MonoBehaviour {  
   
 // Use this for initialization  
 void Awake () {  
 GetComponent<Renderer>().material = GetRandomMaterial();  
 }  
   
   
  
 /// <summary>  
 /// helper method to get a random color  
 /// </summary>  
 /// <returns></returns>  
 public Material GetRandomMaterial()  
 {  
 int x = Random.Range(0, 5);  
 if (x == 0)  
 return Resources.Load("Materials/redMaterial") as Material;  
 else if (x == 1)  
 return Resources.Load("Materials/greenMaterial") as Material;  
 else if (x == 2)  
 return Resources.Load("Materials/blueMaterial") as Material;  
 else if (x == 3)  
 return Resources.Load("Materials/yellowMaterial") as Material;  
 else if (x == 4)  
 return Resources.Load("Materials/purpleMaterial") as Material;  
 else  
 return Resources.Load("Materials/redMaterial") as Material;  
 }  
   
}

Red border  
  
using UnityEngine;  
 using System.Collections;  
   
public class RedBorder : MonoBehaviour  
 {  
   
 // Use this for initialization  
 void OnTriggerEnter(Collider col)  
 {  
 if (col.gameObject.tag == Constants.PlayerTag)  
 GameManager.Instance.Die();  
 }  
 }  
   
stuff spawner

using UnityEngine;  
 using System.Collections;  
   
public class StuffSpawner : MonoBehaviour  
 {  
 //points where stuff will spawn :)  
 public Transform[] StuffSpawnPoints;  
 //meat gameobjects  
 public GameObject[] Bonus;  
 //obstacle gameobjects  
 public GameObject[] Obstacles;  
   
 public bool RandomX = false;  
 public float minX = -2f, maxX = 2f;  
   
 // Use this for initialization  
 void Start()  
 {  
 //first, let's decide whether we'll place an obstacle  
 //beware, Random.Range is exclusive for integers but inclusive for floats  
 //http://answers.unity3d.com/questions/585266/question-about-randomrange.html  
 bool placeObstacle = Random.Range(0, 2) == 0; //50% chances  
 int obstacleIndex = -1;  
 if (placeObstacle)  
 {  
 //select a random spawn point, apart from the first one  
 //since we do not want an obstacle there  
 obstacleIndex = Random.Range(1, StuffSpawnPoints.Length);  
   
 CreateObject(StuffSpawnPoints[obstacleIndex].position, Obstacles[Random.Range(0, Obstacles.Length)]);  
 }  
   
  
 for (int i = 0; i < StuffSpawnPoints.Length; i++)  
 {  
 //don't instantiate if there's an obstacle  
 if (i == obstacleIndex) continue;  
 if (Random.Range(0, 3) == 0) //33% chances to create candy  
 {  
 CreateObject(StuffSpawnPoints[i].position, Bonus[Random.Range(0, Bonus.Length)]);  
 }  
 }  
   
  
 }  
   
 void CreateObject(Vector3 position, GameObject prefab)  
 {  
 if (RandomX) //true on the wide path, false on the rotated ones  
 position += new Vector3(Random.Range(minX, maxX), 0, 0);  
   
 Instantiate(prefab, position, Quaternion.identity);  
 }  
   
  
}  
   
Swipe collider

using UnityEngine;  
 using System.Collections;  
   
public class SwipeCollider : MonoBehaviour  
 {  
   
 // Use this for initialization  
 void OnTriggerEnter(Collider hit)  
 {  
 if (hit.gameObject.tag == Constants.PlayerTag)  
 GameManager.Instance.CanSwipe = true;  
 }  
   
 void OnTriggerExit(Collider hit)  
 {  
 if (hit.gameObject.tag == Constants.PlayerTag)  
 GameManager.Instance.CanSwipe = false;  
 }  
 }

Time destroyer

using UnityEngine;  
 using System.Collections;  
   
public class TimeDestroyer : MonoBehaviour  
 {  
   
 // Use this for initialization  
 void Start()  
 {  
 Invoke("DestroyObject", LifeTime);  
 }  
   
  
 void DestroyObject()  
 {  
 if (GameManager.Instance.GameState != GameState.Dead)  
 Destroy(gameObject);  
 }  
   
  
 public float LifeTime = 10f;  
 }

UI manager

using UnityEngine;  
 using System.Collections;  
 using UnityEngine.UI;  
   
public class UIManager : MonoBehaviour  
 {  
   
 void Awake()  
 {  
 if (instance == null)  
 {  
 instance = this;  
 }  
 else  
 {  
 DestroyImmediate(this);  
 }  
 }  
   
 //singleton implementation  
 private static UIManager instance;  
 public static UIManager Instance  
 {  
 get  
 {  
 if (instance == null)  
 instance = new UIManager();  
   
 return instance;  
 }  
 }  
   
 protected UIManager()  
 {  
 }  
   
 private float score = 0;  
   
  
 public void ResetScore()  
 {  
 score = 0;  
 UpdateScoreText();  
 }  
   
 public void SetScore(float value)  
 {  
 score = value;  
 UpdateScoreText();  
 }  
   
 public void IncreaseScore(float value)  
 {  
 score += value;  
 UpdateScoreText();  
 }  
   
 private void UpdateScoreText()  
 {  
 ScoreText.text = score.ToString();  
 }  
   
 public void SetStatus(string text)  
 {  
 StatusText.text = text;  
 }  
   
 public Text ScoreText, StatusText;  
   
  
  
}