Creating my own game in Unity

DINOFLIGHT

Hilda Nateghi

TE18

School: International IT College of Sweden

Term: HT-VT (2020-2021)

Supervisor: Babak Garjani Majidzadeh

Abstract

In this project, you will read and follow the process of creating the game in the programming language C#. The game industry keeps growing and is one of the most lucrative industries today. My inspiration to create this project came from my own interest and passion for game development. The aim of this project is to answer the question "*Is it possible to create a game based on the knowledge learned in courses programmering 1 and programmering 2?*". My approach to solve this question was by creating a game from scratch in Unity. The game is about a dinosaur that's swinging in stars while traveling in space. The goal is to gather as many points as you can without touching the lava below you. Even though many issues arose and a lot of troubleshooting was made, I successfully created a fun game that fulfills the criterias!

Sammanfattning

I detta projekt kommer du att läsa och följa processen för att skapa spelet på programmeringsspråket C #. Spelindustrin fortsätter att växa och är en av de mest lukrativa branscherna idag. Min inspiration för att skapa detta projekt kom från mitt eget intresse och passion för spelutveckling. Syftet med detta projekt är att svara på frågan ”Är det möjligt att skapa ett spel baserat på kunskapen som lärts in i kurserna programmering 1 och programmering 2?”. Mitt tillvägagångssätt för att lösa denna fråga var genom att skapa ett spel från grunden i Unity. Spelet handlar om en dinosaurie som svänger i stjärnor när man reser i rymden. Målet är att samla så många poäng som möjligt utan att röra lavan nedanför dig. Även om många problem uppstod och mycket felsökning gjordes, skapade jag framgångsrikt ett roligt spel som uppfyller kriterierna!

Table of contents

1. Introduction

1.1 Background

The gaming industry is one of the most lucrative industries today and has expanded exponentially throughout the years[1]. From Mario Kart, Crash Bandicoot, Modern Warfare 2, Need For Speed, Minecraft and GTA, these are the eras of Gen Z where we all grew up playing video games together. Game development has been a dream job of mine since I was little and have built games from time to time but nothing too advanced, just the basics. The reason why I chose this project was because of my own interest in game development. That's the reason why I chose to go to Teknik line with the intention to become a game developer. I have created games before but not at this level so it was a fun challenge. I have attended and completed the courses programmering 1 and programmering 2. The programming language the code in is C# so naturally, I chose to continue with coding in that language. After researching a bit, according to the article "Is C# a Good Tool for Game Development?"[2], C# is the most popular language to code games in. An example of a game we all played once upon a time, coded in C#, is **Temple Run**! The game I'm creating is a mix between the games JetPack JoyRide and Galaxy Shooter!

1.2 Purpose and frågeställning/question

The purpose of this project is to create a game which is easy and fun. I am aiming to achieve these elements:

- Score system
- Grappling system
- Camera motion and effects

But the question stands, is it possible to create a game with the knowledge learned in the courses programmering 1 and programmering 2? I will answer the question by creating and coding a game from scratch in the language C# in Unity.

[1] Why Video and Mobile Gaming Industry May Be One of the Most Lucrative Sectors on Wallstreet
[2] Is C# a Good Tool for Game Development?

1.3 Theory and key concepts

These are the concepts we have learned in programming 1 and programming 2 that I have been using to complete the code in this project:

1.3.1 Programming 1

- **Variables** - stores values.
- **Arrays/Lists** - stores a group of elements.
- **Conditional** - if statements, bools - executing action based on information given.
- **Loops** - for/while - repeats until condition is met.
- **Methods/Functions** - compacting code and reusing.
- **Objects** - cluster and represent data.

1.3.2 Programming 2

- **Strengthen knowledge from programming 1**
- **Inheritance**: inherits, extends or modifies attributes from another class.
- **Encapsulation**: bundling data.
- **Polymorphism**: single element morphing into different types.

1.3.3 Unity

[3]**Sprites** - objects that contain graphical images, used for characters and objects in game.

[4]**Scenes** - these contain objects in your game. You can create for example menus and levels.

[5]**Script** - a script tells gameobjects in a game how to behave.

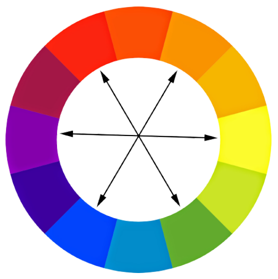
C# - a programming language that uses Object Oriented Programming (OOP).

[6]**OOP** - is about constructing structures that include both data and methods.

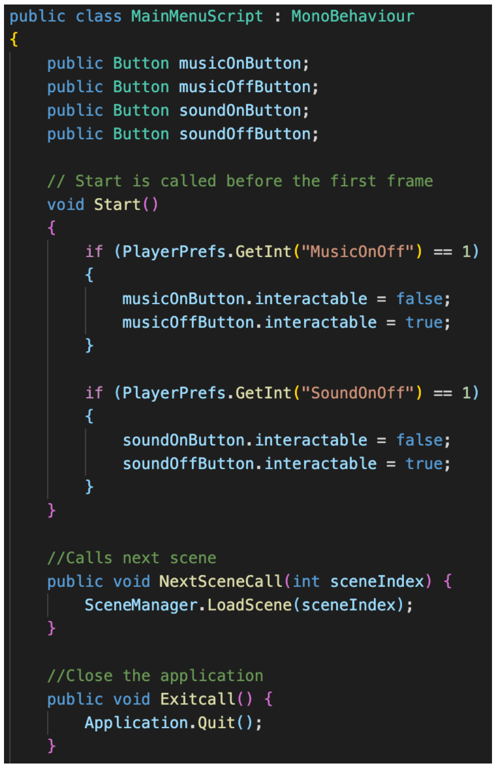
[3] Unity - Creating Sprites.
[4] Scenes
[5] Learning C# and coding in Unity for beginners
[6] C# OOP (Object-Oriented Programming)

2. Method

2.1 Panels and functionality



The first thing was to figure out how I wanted the game to look. With extensive research and planning I came up with this **MainMenuPanel**. Several elements such as texts and buttons were implemented and the buttons have their own functionality. The **play** button changes the scene to the "**GameScene**", the **settings** button takes you to the settings panel and the exit button closes the application. Colors were important and one thing I had the most use out of was the knowledge of complementary colors[7]. The complementary colors made the game pop, an example of this is the dark red background and the green dinosaur.

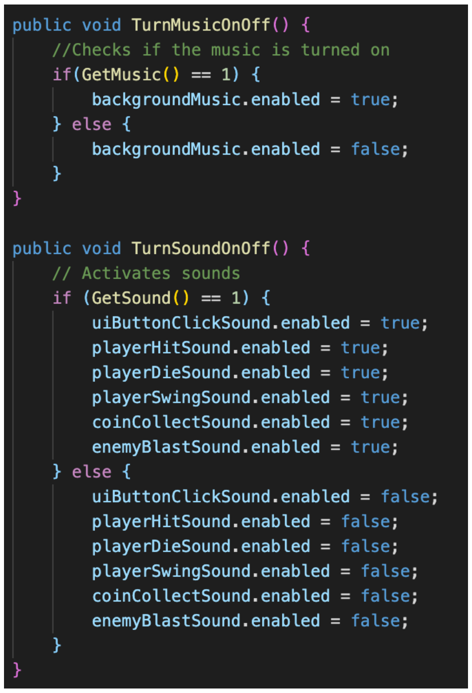


```
public class MainMenuScript : MonoBehaviour
{
    public Button musicOnButton;
    public Button musicOffButton;
    public Button soundOnButton;
    public Button soundOffButton;

    // Start is called before the first frame
    void Start()
    {
        if (PlayerPrefs.GetInt("MusicOnOff") == 1)
        {
            musicOnButton.interactable = false;
            musicOffButton.interactable = true;
        }

        if (PlayerPrefs.GetInt("SoundOnOff") == 1)
        {
            soundOnButton.interactable = false;
            soundOffButton.interactable = true;
        }
    }

    //Calls next scene
    public void NextSceneCall(int sceneIndex) {
        SceneManager.LoadScene(sceneIndex);
    }

    //Close the application
    public void Exitcall() {
        Application.Quit();
    }
```

The next panel was the **Settings Panel** and for this I implemented buttons to make the page interactable. The buttons have functionality; you can turn on/off the music and the sound if that's what the player prefers. These panels are under the **MainMenuScene** and this is what the player sees when starting the game. The **GameScene** is where the actual game is. To be able to control the panels, I've created a script called **MainMenuScript** and in this script are variables and methods to control the scene switches and starting the music and sounds when starting the game. Some of the code handles the settings panel, making it possible to turn the music on/off and when doing so, disabling the other button.

[7] Wikipedia — Complementary colors.

2.2 Soundtracks

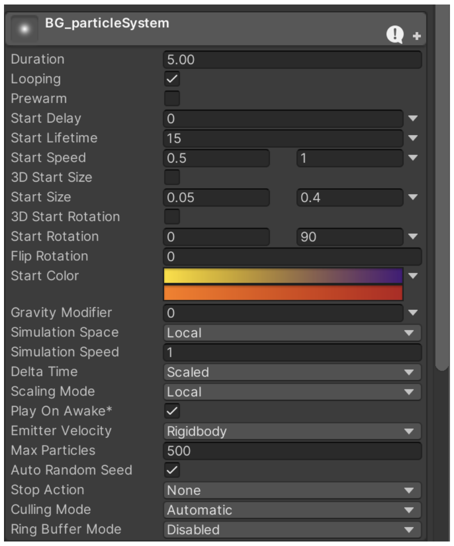


```
public void TurnMusicOnOff() {
    //Checks if the music is turned on
    if(GetMusic() == 1) {
        backgroundMusic.enabled = true;
    } else {
        backgroundMusic.enabled = false;
    }
}

public void TurnSoundOnOff() {
    // Activates sounds
    if (GetSound() == 1) {
        uiButtonClickSound.enabled = true;
        playerHitSound.enabled = true;
        playerDieSound.enabled = true;
        playerSwingSound.enabled = true;
        coinCollectSound.enabled = true;
        enemyBlastSound.enabled = true;
    } else {
        uiButtonClickSound.enabled = false;
        playerHitSound.enabled = false;
        playerDieSound.enabled = false;
        playerSwingSound.enabled = false;
        coinCollectSound.enabled = false;
        enemyBlastSound.enabled = false;
    }
}
```

The sound effects determine the whole ambiance in a game. I made sure to find sounds I liked and that had a sense of catchiness to it. For instance, the background music has elements from our childhood games such as Sonic and has a funky and fun feeling to it. There goes more thought in these things than one might think, the coin sound, the click sound, the player death sound etc. All the buttons have a consistent sound and have a satisfying sound. These buttons have event listeners so when you click, the buttons will act on the functionality given such as turning the music/sound on/off. To be able to control all the sounds, I created a script called **SoundManagerScript**. The script contains variables for the different sounds as well as how they behave and their functionality. Other methods are implemented to, for instance, destroy duplicate sound managers and/or scripts. There are numbers assigned to the on and off function which we will use on the buttons to verify the on/off state.

2.3 Effects and Background

The background image is a galaxy with tiny stars to make the image seem more alive. I created a system and what it does is making small particles/stardust roam in different directions at a slow pace. They vary and size, color, shape and rotation to make it more natural. The particles also have a lifetime which means that they randomly disappear after a couple of seconds. The background itself will be the only background image during the whole game.

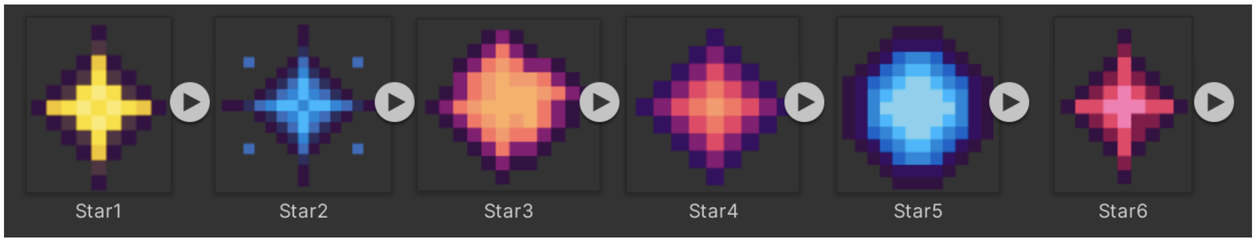


BG_particleSystem

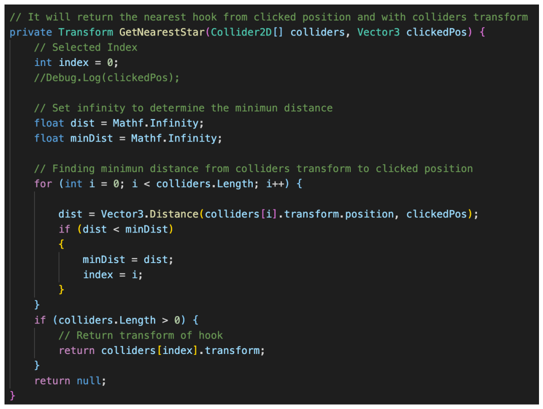
Duration	5.00	
Looping	✓	
Prewarm		
Start Delay	0	
Start Lifetime	15	
Start Speed	0.5	1
3D Start Size		
Start Size	0.05	0.4
3D Start Rotation		
Start Rotation	0	90
Flip Rotation	0	
Start Color		
Gravity Modifier	0	
Simulation Space	Local	
Simulation Speed	1	
Delta Time	Scaled	
Scaling Mode	Local	
Play On Awake*	✓	
Emitter Velocity	Rigidbody	
Max Particles	500	
Auto Random Seed	✓	
Stop Action	None	
Culling Mode	Automatic	
Ring Buffer Mode	Disabled	

2.4 Player, Hooks and Enemy

The character I chose to use as my player was a dinosaur! The planet Mars is the platform. I got inspired by the Perseverance Rover landing on Mars! There are several components to the player and physics play a part in how the player moves and manipulates into different forms. The two important components are **Rigidbody 2D** and **Spring Joint 2D**. The rigidbody is all about motion; basic physics is added such as gravity and collision if the right collider is added. That's where the spring joint comes in; this allows objects to collide with each other. The hooks in the game are represented as six different stars and the enemy is represented as black holes.



A Box Collider is added and a script called **PlayerController** and that's where we create the grappling system and the player interaction. The grappling system consists of nine methods and variables. The grappling system is divided in three stages: **GrapplingStart()**, **Grappling()**, **GrapplingEnd()**. The **GrapplingStart()** method checks if the player is grappling. It also takes care of other things such as finding the nearest star to grab on to and store colliders that overlap with the area. The **Grappling()** method contains mathematical calculations to reduce the distance between the player and the star and attaching a line between the player and the star. To stop the player from continuously spinning around the star, for each frame, we reduce the speed. The **GrapplingEnd()** method essentially resets everything as soon as you stop grappling as the bool is set to false. There's also another method that is quite important! The **GetNearestStar()**



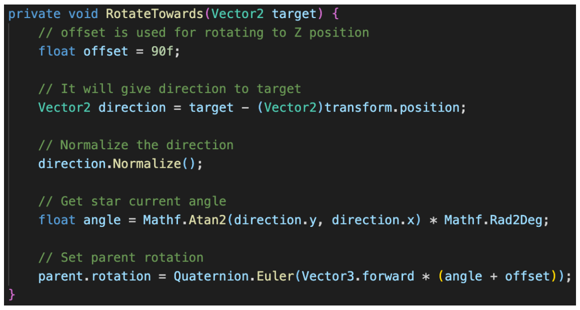
```
// It will return the nearest hook from clicked position and with colliders transform
private Transform GetNearestStar(Collider2D[] colliders, Vector3 clickedPos) {
    // Selected Index
    int index = 0;
    //Debug.Log(clickedPos);

    // Set infinity to determine the minimun distance
    float dist = Mathf.Infinity;
    float minDist = Mathf.Infinity;

    // Finding minimun distance from colliders transform to clicked position
    for (int i = 0; i < colliders.Length; i++) {

        dist = Vector3.Distance(colliders[i].transform.position, clickedPos);
        if (dist < minDist)
        {
            minDist = dist;
            index = i;
        }
    }
    if (colliders.Length > 0) {
        // Return transform of hook
        return colliders[index].transform;
    }
    return null;
}
```

method calculates and returns the nearest star from the nearest clicked position. The loop will calculate the minimum distance between stars that are located very close to each other.

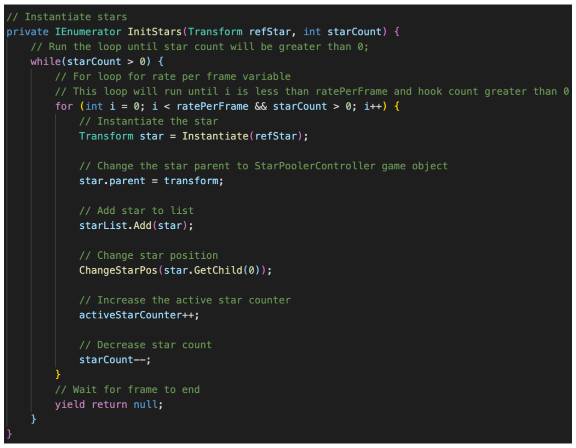
2.5 Camera and GamePlay/GameOver

In the **CameraController** script are all of the different camera effects. The camera follows the player and shakes when colliding with different objects such as stars and blackholes. The loop calculates the new camera position and then proceeds to shake it by rotating the camera side to side as well as back and forth. After the shake effect, the speed back to normal speed with a smooth transition since the shake motion is in slow motion.

While playing the game, you will notice that the stars are rotating when you grab on to them! That's because of a method called **RotateTowards()** that is located in the **StarController** script! This method calculates the direction of the target and then normalize it which means that the length will not be longer than one. Then it calculates the current angle of the hook and then rotates it towards the player.



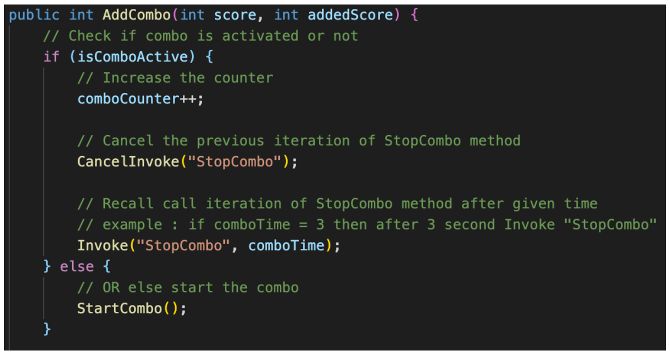
```
private void RotateTowards(Vector2 target) {
    // offset is used for rotating to Z position
    float offset = 90f;

    // It will give direction to target
    Vector2 direction = target - (Vector2)transform.position;

    // Normalize the direction
    direction.Normalize();

    // Get star current angle
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;

    // Set parent rotation
    parent.rotation = Quaternion.Euler(Vector3.forward * (angle + offset));
}
```



```
// Instantiate stars
private IEnumerator InitStars(Transform refStar, int starCount) {
    // Run the loop until star count will be greater than 0;
    while(starCount > 0) {
        // For loop for rate per frame variable
        // This loop will run until i is less than ratePerFrame and hook count greater than 0
        for (int i = 0; i < ratePerFrame && starCount > 0; i++) {
            // Instantiate the star
            Transform star = Instantiate(refStar);

            // Change the star parent to StarPoolerController game object
            star.parent = transform;

            // Add star to list
            starList.Add(star);

            // Change star position
            ChangeStarPos(star.GetChild(0));

            // Increase the active star counter
            activeStarCounter++;

            // Decrease star count
            starCount--;
        }
        // Wait for frame to end
        yield return null;
    }
}
```

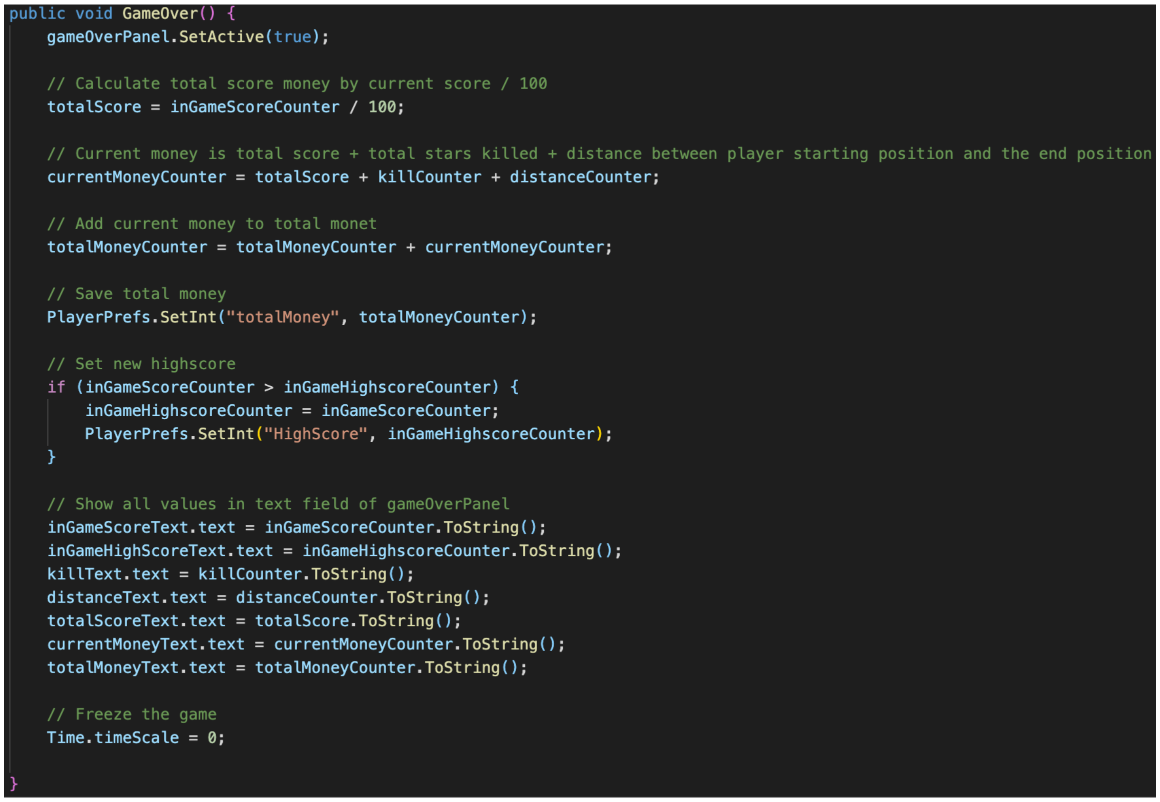
The stars are auto generated which makes the game go on to infinity. This is achieved by allowing x number of stars in a list to be generated and displayed at the same time. The stars have randomized positions and more stars will be generated whilst moving in whichever direction desired.

2.6 Score System



```
public int AddCombo(int score, int addedScore) {
    // Check if combo is activated or not
    if (isComboActive) {
        // Increase the counter
        comboCounter++;

        // Cancel the previous iteration of StopCombo method
        CancelInvoke("StopCombo");

        // Recall call iteration of StopCombo method after given time
        // example : if comboTime = 3 then after 3 second Invoke "StopCombo"
        Invoke("StopCombo", comboTime);
    } else {
        // OR else start the combo
        StartCombo();
    }
```

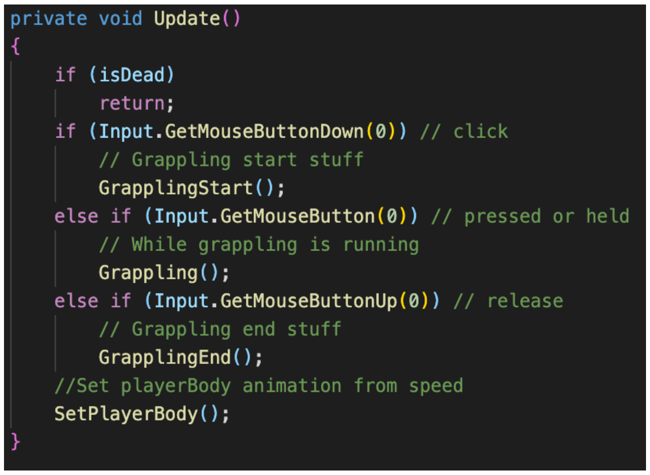
The score system was quite fun to make. When hitting a star, you will score 100 points. These points get stored in variables and can later be used to calculate other things such as the **totalMoneyCounter** or the **totalScore**. When scoring, a small popup text will pop up with the text “**+100**” and this score will be added to the score counter. The text has a lifespan of 2 seconds and will disappear thereafter. The scoreboard will be shown when the game is over. There’s a designated **GameOverPanel** and where you design how it’s going to look and the script **GameSceneScript** is attached to be able to control the different functions. When hitting multiple stars within a small time frame you will get combos. The score doubles with every star hit and there’s a max to how many combos you can achieve within that time frame. The script **ComboController** detects and calculates the combo.



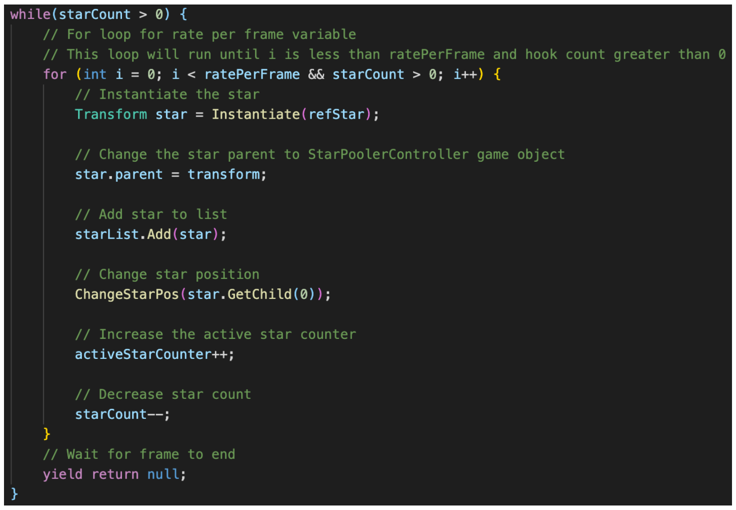
```
public void GameOver() {
    gameOverPanel.SetActive(true);

    // Calculate total score money by current score / 100
    totalScore = inGameScoreCounter / 100;

    // Current money is total score + total stars killed + distance between player starting position and the end position
    currentMoneyCounter = totalScore + killCounter + distanceCounter;

    // Add current money to total monet
    totalMoneyCounter = totalMoneyCounter + currentMoneyCounter;

    // Save total money
    PlayerPrefs.SetInt("totalMoney", totalMoneyCounter);

    // Set new highscore
    if (inGameScoreCounter > inGameHighscoreCounter) {
        inGameHighscoreCounter = inGameScoreCounter;
        PlayerPrefs.SetInt("HighScore", inGameHighscoreCounter);
    }

    // Show all values in text field of gameOverPanel
    inGameScoreText.text = inGameScoreCounter.ToString();
    inGameHighScoreText.text = inGameHighscoreCounter.ToString();
    killText.text = killCounter.ToString();
    distanceText.text = distanceCounter.ToString();
    totalScoreText.text = totalScore.ToString();
    currentMoneyText.text = currentMoneyCounter.ToString();
    totalMoneyText.text = totalMoneyCounter.ToString();

    // Freeze the game
    Time.timeScale = 0;

}
```

2.7 Scripts

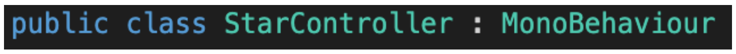
I have mentioned a few scripts above, but there are a total of nine scripts. These scripts control various aspects of the game and the knowledge of the courses programming 1 and programming 2 really came handy here. Some examples of this are OOP, variables, conditions, methods, loop, getters and setters, and arrays. In the pictures all around, you can see a couple of examples from different scripts where these concepts were used!



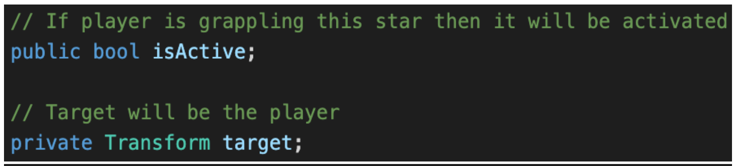
```
private void Update()
{
    if (isDead)
        return;
    if (Input.GetMouseButtonDown(0)) // click
        // Grappling start stuff
        GrapplingStart();
    else if (Input.GetMouseButton(0)) // pressed or held
        // While grappling is running
        Grappling();
    else if (Input.GetMouseButtonUp(0)) // release
        // Grappling end stuff
        GrapplingEnd();
    //Set playerBody animation from speed
    SetPlayerBody();
}
```



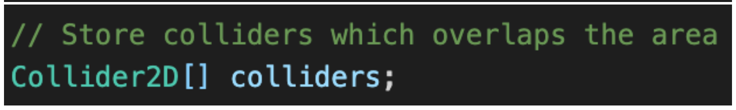
```
while(starCount > 0) {
    // For loop for rate per frame variable
    // This loop will run until i is less than ratePerFrame and hook count greater than 0
    for (int i = 0; i < ratePerFrame && starCount > 0; i++) {
        // Instantiate the star
        Transform star = Instantiate(refStar);

        // Change the star parent to StarPoolerController game object
        star.parent = transform;

        // Add star to list
        starList.Add(star);

        // Change star position
        ChangeStarPos(star.GetChild(0));

        // Increase the active star counter
        activeStarCounter++;

        // Decrease star count
        starCount--;
    }
    // Wait for frame to end
    yield return null;
}
```



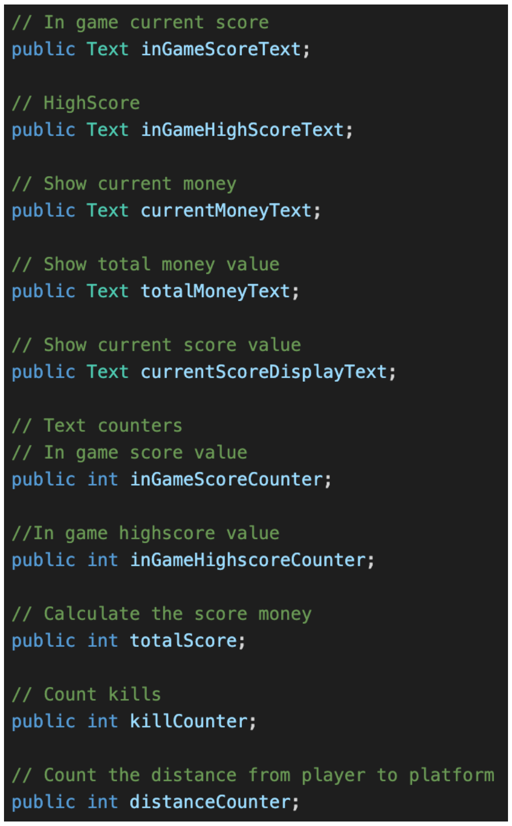
```
public class StarController : MonoBehaviour
```



```
// If player is grappling this star then it will be activated
public bool isActive;

// Target will be the player
private Transform target;
```



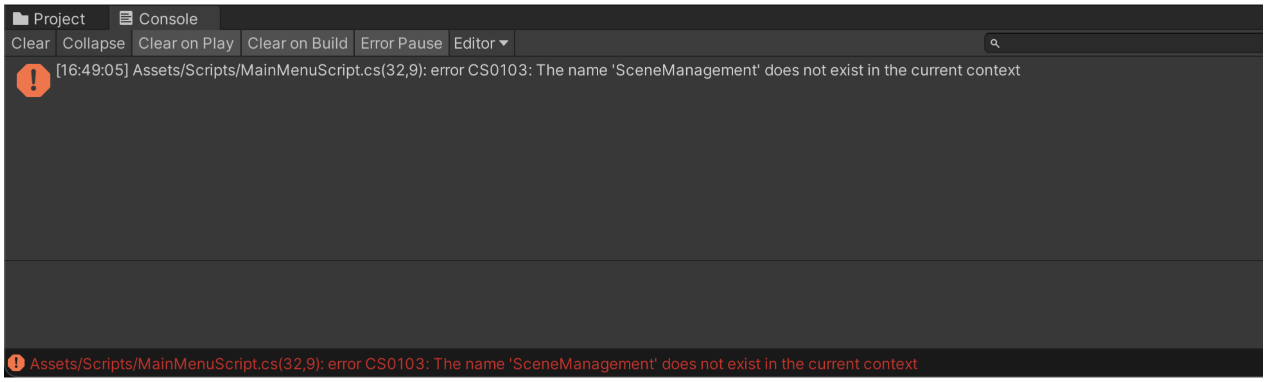
```
// Store colliders which overlaps the area
Collider2D[] colliders;
```



```
// In game current score
public Text inGameScoreText;

// HighScore
public Text inGameHighScoreText;

// Show current money
public Text currentMoneyText;

// Show total money value
public Text totalMoneyText;

// Show current score value
public Text currentScoreDisplayText;

// Text counters
// In game score value
public int inGameScoreCounter;

//In game highscore value
public int inGameHighscoreCounter;

// Calculate the score money
public int totalScore;

// Count kills
public int killCounter;

// Count the distance from player to platform
public int distanceCounter;
```

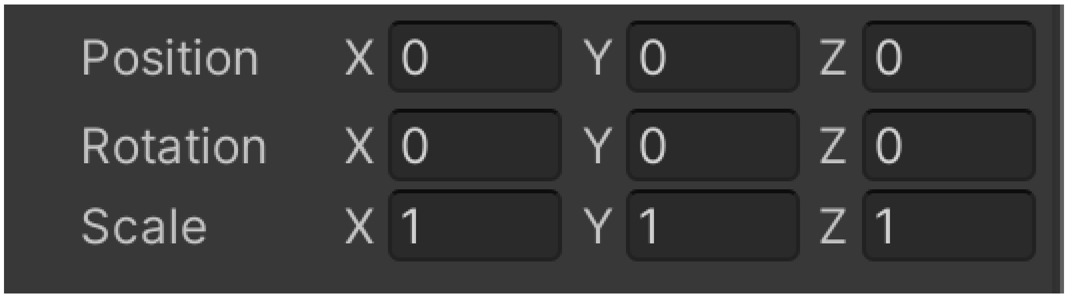
2.8 Problems and Troubleshooting

I ran into many issues and the issues ranged from typos to missing code, or incorrect code. With the help of the internet I could find the solutions to the issues and get on to the next segment. The picture below is an issue I ran into many times:



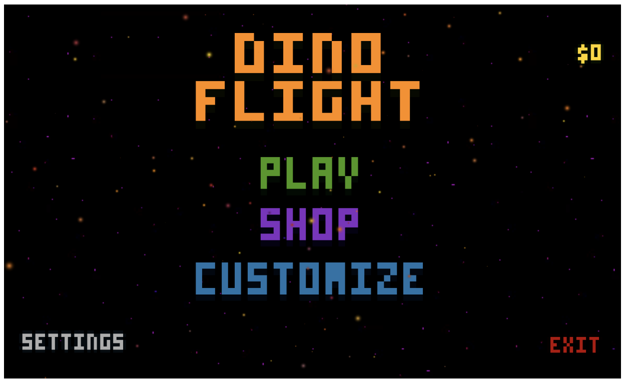
```
[16:49:05] Assets/Scripts/MainMenuScript.cs(32,9): error CS0103: The name 'SceneManagement' does not exist in the current context
Assets/Scripts/MainMenuScript.cs(32,9): error CS0103: The name 'SceneManagement' does not exist in the current context
```

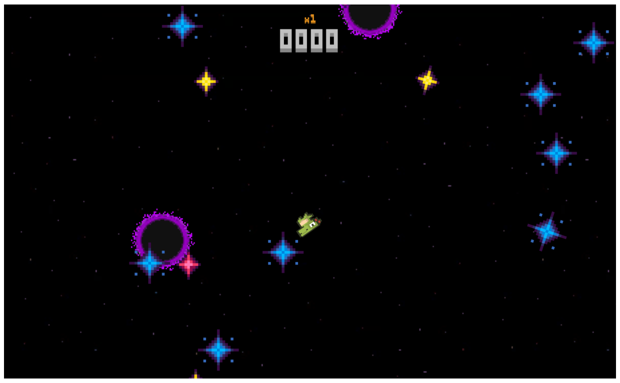
The issue was a typo, instead of "**SceneManagement**", it was "**SceneManager**". And these simple issues are not easy to find since these things fly past your eyes very easily. You always think that the issue is much more complicated but turns out that it was much simpler that you thought. There were many other issues that arose that didn't really have anything to do with the code, but forgetting to, for example, reset the position. This led to my character bouncing and flying uncontrollably and I didn't understand why, but in the end grasped what was wrong.

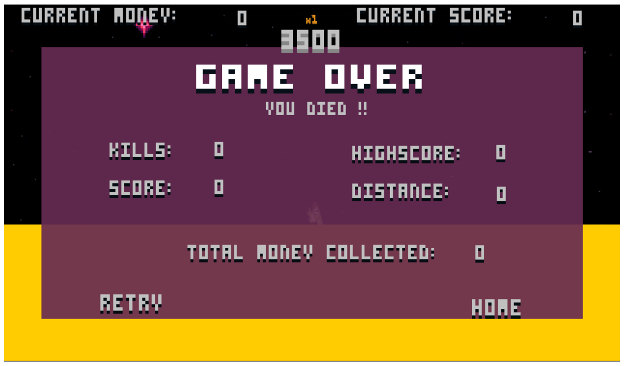


Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

3. Result





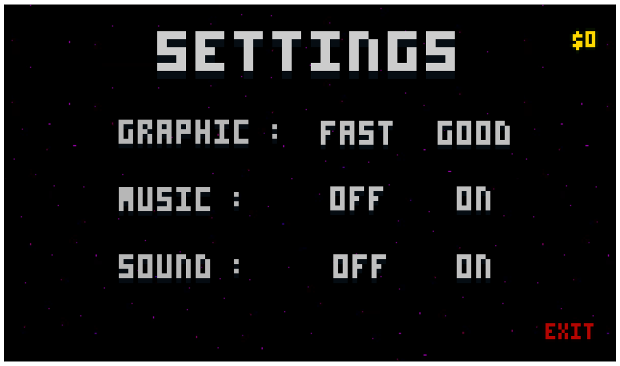


CURRENT MONEY: 0 CURRENT SCORE: 0

KILLS: 0 HIGHSCORE: 0

SCORE: 0 DISTANCE: 0

TOTAL MONEY COLLECTED: 0



GRAPHIC : FAST GOOD

MUSIC : OFF ON

SOUND : OFF ON

In the pictures above, you can see the end result! These are some shots of the main parts of the game. The panels as well as the functionality were designed to make the game more interactable. What makes a button interactable is event listeners. They wait for an event such as clicking on it to proceed with an action assigned. An example of this is the "**PLAY**" button, when you press it, the scene changes and you begin the gameplay. The thought behind the sounds in a game is more complicated than one might think since they determine the ambiance of the game. I made sure to choose the sounds and music to match the aesthetic as well as matching each other in a way that everything seems to fit in naturally.

If you look closely at the MainMenuPanel, you can see small particles/stardust that roam in different directions. This small detail makes a huge difference in the ambiance of the game. The variety of stars and their location also adds a serene feeling where you see the randomness of everything which we can feel when we think about the placements of the stars in our own universe. To solve one of the criterias i've set, which was to create a grappling system which was successfully coded and many concepts from the courses were used. The player can grapple to the stars which makes the player swing and move in any direction desired. The platform the player stands on is planet Mars with the inspiration taken from the Perseverance Rover that landed on Mars!

The camera follows the player and shakes when colliding with objects such as stars or black holes. This effect was very cool to code and gave the game magnitude. The score system was very fun to code! This is where the scores are calculated and where combos are counted. For every collision with a star, you gain 100 points, and when you collide with multiple objects under a certain amount of time, you get combo points. These points are calculated by multiplying with the current score for every collision. When you touch the lava, the player will die and the GameOverPanel will pop up and you can choose to either retry or the home button which takes you to the main menu.

The whole game was very fun to code and create and almost all of the concepts learned in programming 1 and programming 2 were used, which answers the frågeställning/question.

4. Discussion and conclusions

Now that I have completed the project, the frågeställning/question is very easy to answer. It is possible to create a game with the knowledge from the two courses. Since I had previous knowledge of coding with C#, It was easier for me to complete the project. The code took about 6 months to complete for me. But for someone with more experience and time, it could probably take about a month to complete and would do a much better job than I.

There were things I didn't have time to do because of them being very time consuming and things I could've done better. These things include having the GameOverPanel properly working. The code calculates everything but I didn't have time to make it show on the panel when dying in game. One thing I would've loved to have in the game is the option to customize and upgrade your player and grapple hook. This would've been such a cool thing to have in the game, but I solely focused on completing the GameScene and did almost finish it. Another thing I didn't have time to complete fully was the MainMenuPanel since you can't press on, for example, the settings button and get to the

SettingsPanel, but if you go into the SettingsPanel separately, it is functioning! One last feature that I would've loved to implement was the money function, When you hit certain stars, you would gain money that you could spend in the customize scene.

If you gave this project with the same game concept to a game developer, there are many things they would have done differently; code structure, programming languages, features etc. But for a 3rd grade student in highschool with much less experience, I think I pulled it off. The part that was the most fun to do was the design! To design the main menu, all the panels, the choosing of characters, finding the music, coding and then seeing the result!

I had many ideas in mind and the game concept as well as the name changed over time; the name was generated by a website called "*fantasynamegenerators*"[8]. In the beginning, This game was supposed to be about a dinosaur escaping from a burning forest and swinging on vines. The "enemy" would be the fire and you would lose points and there would be lava in the ground. If you touched or fell into the lava, you would die. The final game concept ended with a dinosaur, swinging on stars, traveling in space. Lava would be present in space because why not! The game is pretty unrealistic which makes it more fun! The game idea changed over time, but the concept stayed. I coded the game with great satisfaction with the knowledge I've learned from the courses programming 1 and programming 2. Even though I failed many times, I sat through it and troubleshooted the code and found the issues and solved them despite the extreme frustration.

DinoFlight is a fun game with catchy music with fun quirks and has a fun niche. The game was not completed due to the project being more time consuming than I thought. But the game holds a strong ground and will be further developed and fully completed to the release standard.

[8] Video game names

5. Sources

1. FinancialNewsMedia.com (2021). *Why Video and Mobile Gaming Industry May Be One of the Most Lucrative Sectors on Wallstreet*. [online] Prnewswire.com. Available at: Why Video and Mobile Gaming Industry May Be One of the Most Lucrative Sectors on Wallstreet
2. Jordan, J. (2020). *Is C# a Good Tool for Game Development?* [online] NarraSoft. Available at: Is C# a Good Tool for Game Development?
3. Tutorialspoint.com. (2021). *Unity - Creating Sprites - Tutorialspoint*. [online] Available at: Unity - Creating Sprites
4. Unity Technologies (2017). *Unity - Manual: Scenes*. [online] Unity3d.com. Available at: Scenes
5. Unity. (2017). *Learning C# and coding in Unity for beginners - Unity*. [online] Available at: Learning C# and coding in Unity for beginners
6. W3schools.com. (2021). *C# OOP (Object-Oriented Programming)*. [online] Available at: C# OOP (Object-Oriented Programming)
7. Wikipedia Contributors (2021). *Complementary colors*. [online] Wikipedia. Available at: Wikipedia — Complementary colors
8. Fantasynamegenerators.com. (2012). *Video game names*. [online] Available at: https://www.fantasynamegenerators.com/video-game-names.php