EUROPEAN UNIVERSITY OF LEFKE

FACULTY OF ENGINEERING

Graduation Project 2

GravaFun

Akram Abuhajar

184650

GravaFun, it is an educational video game project made using unity development platform, mainly focuses on embedding learning material to a fun and inter-active video game, explaining the some of the basic laws of physics, which will be introduced to children under the age of 12, to grow and expand their knowledge about how some of the basic daily tasks happen the way they do.

Supervisor

Cem kalyoncu

Publish Date

Table Of Contents

1.Introduction

1.1 Problem definition

Video game industry is one of the biggest - if not the biggest - entertainment industry that is being consumed by people all over the world, and it is expected that it will be consumed and requested even more with the evolution of technology and the amount of possibilities this technology offers.

The fact that such an industry is taking more attention than actual beneficial learning material is increasing with new upcoming generation, so the core problem of it is the lack of adding more and more learning and educational material in the industry, to make the spent time on such entertainment is actually beneficial to the consumer, here is some problems that deserve spotting the light on.

Example-Problems :

- Lack of educational material in the new generation video game industry, due to the common mistakenly thinking that It will not be attractive and catchy to the consumer eyes.
- Lack of understanding the fact that education and entertainment can run in a parallel lines with an equilibrium of the balance between them,
- Wrong approaches in so many attempts that tried to achieve such a purpose, either poorly entertaining but completely educational, or completely entertaining but poorly educational
- New generations are valuing such an industry more than the education industry, which would lead to a late realization of how simple surrounding activities work and what is the reason behind it
- Losing parents trust and concent to allowing products from the video game industry to be consumed by their children, for the fact that parents are noticing and complaining about the negative effects such an industry has grown to.

1.2 Goals

Here is some of the goals I am focusing on and making sure to achieve on the release of the product to the public :

- Achieving the secret formula :
    - The perfect equilibrium between entertainment material and educational material in the same product.

- Provide a relatable educational material :
    - Easiest way to learn and understand is to relate to the topic, which will be saved in the active memory of the brain, such goal can be achieved with relating activities and actions that consumer comes across while playing the video game, will be explained in a very relatable way, as an example, you can take the first law of newton, nothing will change its current state unless an effect is applied on it, which is a concept that is mainly used in the video game industry but never been explained and put the spot light on onto it.

- Parents concent and acceptance :
    - A big inflouenser on the age genre I am aiming for is parents, their concent and acceptance can play a huge role in marketing such a product, making the product parents friendly, will lead to parents supporting and in some times even recommending such products to their children and maybe other parents too.

- Veriety of content:
    - Creating a product for such age genre can be difficult sometimes, and one of the difficulties is avoiding repetition in the content to keep the consumer connected to the product and would want more, having multiple types of playable environments but still find the way of connecting them together is very important, instead of a specific environment but minor changes in content to follow up with the connection.

- The fun and entertaining aspect :
    - Undenyabiliy, this is a core goal that any developer in the industry tries to achieve, making the product re-usable and requested is what such a product has to offer in terms of entertainment.

2. Literature Survey

My project has a variety of environments, 3 types to be specific, and more to come in the future, so survey will be done to each environment in comparison with other products in the market.

Compare1 : the platformer phase

Screenshot of product A.

Image 1 a similar existing platformer project

Screenshot B of my product.

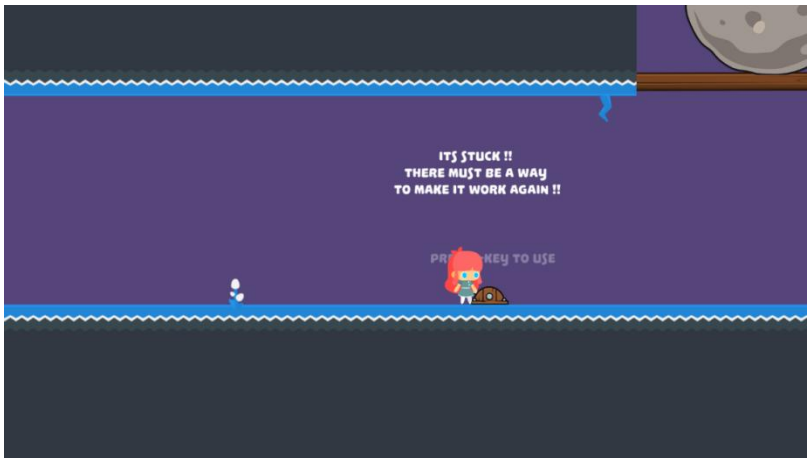


Image 2 project's platfromer level

Taking the amount of stuff that are shown in a single frame from product A, and product B, it is easy to lose attention in the gameplay due to the huge amount of details, the choice of colors, which would make product B more appealing and more acceptable.

Another point to point out is the core concept, both products claim to provide a puzzle type of gameplay, but in product A, it is clearly shown that the puzzle gameplay is minimized and the combat and fast action gameplay was maximized, while in product B, balances between both concept, combat was put for another phase, and puzzle was given its fair share to be enjoyed.

In the puzzle gameplay, product A, provides a decent concept with no explanation or any educational connection to it, and in product B, every action is explained in a way that the specified genre would digest the provided information, while solving, would actually require a good amount of memory and thinking.

Compare2: the free-faller phase.

Screenshot A of an existing product.

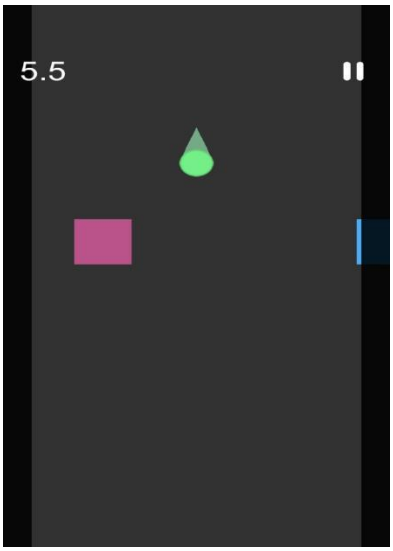


Image 3 a similar free-faller project

Screenshot B of my product.

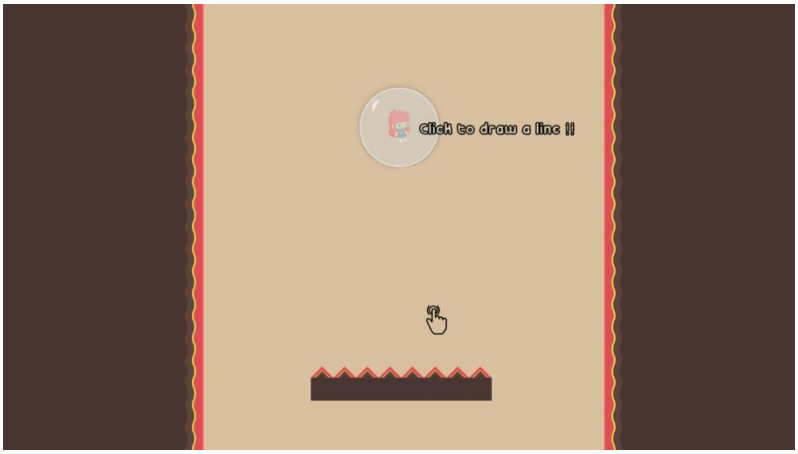


Image 4 project's free-faller level

one of the biggest differences between both of the screenshots, is explaining why such an action is happening, where in the product in screenshot B, it is briefly explained that it is a free fall.

In the product in screenshot A, guide for the gameplay was not provided, even gameplay was easy to win over, so the hardness and the enjoyment of solving the level is much more better in the product B.

Connecting such an environment of gameplay with others was pretty hard to achieve, but the way it was achieved in product B to make the changing of the scenery actually make sense, while in product A, it was a loop, a repetitive type of gameplay.

Compare3 : The space phase.

Screenshot A from an existing product

Image 5 a similer space platfromer level

Screenshot B from my product

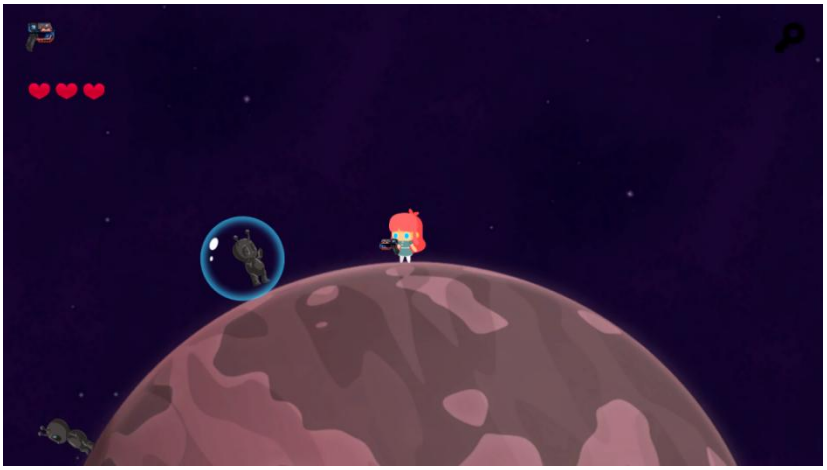


Image 6 the project's space platfromer level

Both screenshots are considered as a platformer, many similar products follow the basic concept of lowering down the gravity affect but still stick with the casual platformer style.

In my product, I have re-designed the way physics work in it, where I have implemented multiple planets with each having its own purposes, an introduction planet, a storyline and explanation to mechanics planet, a combat system with enemies in a planet, and a respawn and finish planet.

Physics in this phase is always driven to the center of the planet the player is on, where every planet has different size and different gravitational force to add extra variety and excitement to the gameplay and to get out of the familiar traditional method.

3. Background Information

3.1 Required & Used software

These are the tools I have used in conducting this project :

- **Unity developing platform and engine :**
  Unity is one of the best developing platforms in the world nowadays, a huge number of some of the biggest hit products in the game development industry was developed using this platform.
  It supports a fully functional engine that assists the developer and simplify the amount of work required to implement a specific concept or bringing an idea on papers to become a real life usable product.
- **Visual Studio Code :**
  Unity's most important skill and biggest component is coding, where everything runs with code, so an editor is required and visual studio code is a great choice, due to the amount of plugins that are offered to support developing in unity's platform, and its intelligent auto-fill system.
- **C# programming language :**
  Developing in unity requires skills in C#, as it is the language of the unity compiler, the way unity works is that unity embedded a lot of functions, data type and extra other stuff to the C# language, only designed to work with unity's environment, they have completely rebuilt it with the existing C# logic, and big modifications from unity's platform, so the compilation is not happening using the C# compiler, it is happening in unity's own compiler.

3.2 Other software

- **AdobePhotoshop :**

  For designing, and modifying a lot of textures and images that were used in the product.

- **Git / Sourcetree :**
  Used for repository submittions.
- **Gimp :**

  For designing, and modifying a lot of textures and images that were used in the product.

- **InkScape :**
  Used for designing the project poster.

4. Methodology

Going with such a project was a great challenge that I was looking forward to stumble upon, and there was multiple steps to reach to the final output, and I would like to spot on the light on some of them :-

- **Planning**.

  Which basically was collecting the material that will be used in the project, draw graphs of the levels, draw scenes and create connections with it, during that period of time It was all about choosing the concept, idea, the textures that will be used.

  Trying out the development environment and run some tests, to see how the functionalities work and how I can use the set of tools that are offered by unity to come up with something that would show my skills.

  On this level, I did not fully draw all the concept on papers and just follow steps, i made the main base structure of the project such as textures, player and its functionalities.

- **Designing and working on the player:**

Working one the player to get the most dynamic and authentic animation was a great challenge I was facing, because I had the textures and I had to do animation mapping, transitioning between animation levels, and most importantly the movement script, I didn't want to follow the basic method of development in unity, I wanted to create my own movement code which I will share and briefly explain.

Starting off with the player code.

```
1- [SerializeField] public float speed = 5f;
2-  [SerializeField] public float jumpSpeed = 5f;
3-  [SerializeField] public float runningSpeed = 15f;
4-  [SerializeField] public float climbSpeed = 5f;
5-  public GameObject pausePanel;
6-  private bool isGrounded = false;
7-  private Vector2 moveSpeed;
8- private float moveDir;
9-  private Rigidbody2D player;
10-  private Animator Girlanimation;
11-  private BoxCollider2D feetCollider;
12-  private bool isRunning;
```

```
13-  private bool isClimbing;
14-  private float timer;
```

The chunk of code on-top is the variables that were used in the player script, some data types might be confusing in the first glance but I will go through them one by one.

Firstly anything that is assigned as a public, can be accessible through the unity platform without the need of going back to the code and change it, which will allow the developer to easily modify the values depending on the trail and error method, and here is a screenshot of how it looks

Image 7 script component for player object

And for the gameobject data types such as the PausePanel, simply you drag the object from the hierarchy of objects to its place in the script component that was attached to, in this case the script was attached to the player, and captured a reference from the pausepanel gameobject.

Vector2 data type, is simply a data type that can contain 2 values in the same index, think about it like a cartesian point, (x , y), this data type can hold the same, but values must be floats.

Rigidbody2D , Boxcollider2D and animator are all object components that can be added to any object in the scene, modified using code or the platform itself, and in this script a reference of the ones attached to the player object were captured for further modification on the upcoming code.

```
1.  private void Start() {
2.  player = GetComponent<Rigidbody2D>();
3.  Girlanimation = GetComponent<Animator>();
4.  feetCollider = GetComponent<BoxCollider2D>();
5.  timer = 0;
6.  }
```

the start function, this function will exist in every script that is created in unity, and the way this function works, is that it only runs once, it only runs on the first frame of the project.

And for this reason usually it is used for capturing references, inheriting information and other multiply stuff.

In this code, it captures the rigidbody2D component of the same object this script is attached to and saves it under the label player for easier access in code, and it does the same to the animator, being saved in Girlanimation label, and boxcollider2D saved in feetcollider label

```
1. private void Update(){
2. if(!pausePanel.activeSelf){
3. Run();
4. Jump();
5. climbingLadder();
6. FlipSprite();
7. }
8. }
```

This **is the update function** that calls everything inside it on every frame.

Inside of it there is an if condition that was added later on in the development to prevent the functions inside the update to work - pausing the player -, where it checks if the pause panel is active or not.

It has the run(), Jump(), climbingladder(), and flipsprite() functions.

Starting **off with the run() function** which is responsible of moving on the X-axis of the player object.

```
1)  void Run() {
2)  moveDir = Input.GetAxisRaw("Horizontal");
3)  if (Input.GetKey(KeyCode.LeftShift)) {
4)  isRunning = true;
5)  Vector2 playerV = new Vector2(moveDir * runningSpeed,player.velocity.y);
6)  player.velocity = playerV;
7)  }
8)  else {
9)  isRunning = false;
10) Vector2 playerV = new Vector2(moveDir * speed, player.velocity.y);
11) player.velocity = playerV;
12) } if (isRunning) {
13) bool playerHasHorizantolSpeed = Mathf.Abs(player.velocity.x) > Mathf.Epsilon;
14) Girlanimation.SetBool("IsRunning", playerHasHorizantolSpeed);
15) Girlanimation.SetBool("IsWalking", false);
16) } else {
17) bool playerHasHorizantolSpeed = Mathf.Abs(player.velocity.x) > Mathf.Epsilon;
```

```
18)   Girlanimation.SetBool("IsWalking", playerHasHorizantolSpeed);
19)   Girlanimation.SetBool("IsRunning", false);
20)  }
21)  }
```

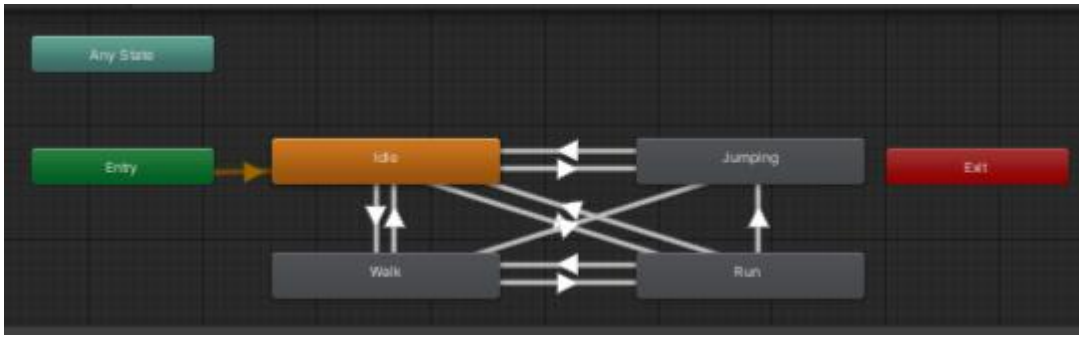
The way this function works, on line 2, there is an input listener, that is listening on the X-axis, where it will return 3 types of values, 1, -1, and 0 depending on the direction it is going, embedded in the system that on clicking the D key or the R-arrow key with trigger this function to understand the movement to the right, which is the positive side on the X-axis, so it will return 1, the opposite works on the A key and the L-arrow key to return -1, and if none was pressed, it will return 0, and using this we can determined the direction using the keyboard input system.

Putting that out of the way, line 3 is a key listener, that will return true if the key so pressed or being pressed, now in the true condition, turns the isRunning bool to true for animation usage soon, and creates a new vector2 data type that will save the direction movement from line 2 multiplied with running speed, because on clicking L-shift, I wanted it to make the player run, and on the second index of the vector2 it will contain whatever the value of the velocity of the player rigidbody2D on the Y-axis, and will pass it to the current velocity of the player rigidbody2D.

And if the if condition was false, it will create a new vector2, that has the move direction multiplied by the walking speed, and on the Y-axis it will take whatever the velocity of the player rigidbody2D on the Y-axis is, and then will pass the vector2 to the velocity of the player's rigidbody2D, with the isRunning bool set to false.

Now for the animation part, basically I put bools for each animation, and I will explain the animation system later on, where if the specific animation's bool is true, it will play it until it becomes false, and using this method, I managed to switch between the Idling, running, walking, and jumping animations with a nested controller, depending on the isRunning bool, it will determine which animation must run, if it is true, it will set the IsWalking animation to false, and the IsRunning animation to a bool that will check if the input listener's value is anything other than 0, and that is happening on line 13, where a new bool is created and it saves the result of a condition if the absolute value of the input listener is bigger than mathf.epsilon, and mathf.epsilon is a positive value that never zero but very close to it, that is more accurate to use is such situations, and the opposite happens for if the isRunning bool is false, turns off the IsRunning animation, and creates a new bool to check if player is moving or not.

Image 8 player animaton controller



Here is a screenshot **of the animation controller**, where you see 4 different situations of animation, idling, walking, running, jumping, and arrows between them are transition conditions, and each transition condition contains a set of bools, if true and if false will play the animation, if player is walking but not running, then play the walking animation, and the opposite for running, if both bools

are false, and the player is not moving, it will play the entry animation which is the idling, the entry animation gets played if no any other animation is played, and for jumping, it will always return to idling, can switch from walking or running or idling to jumping, but always returns to idling.

And the way each animation is created, the animation tool of unity was used and here is a screenshot of it.

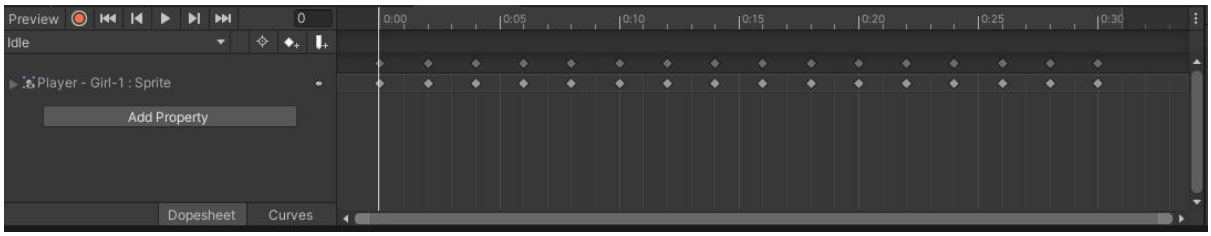


Image 9 example animation for idling

Basically setting each texture of each frame in the timeline of the animator accurately with a balance in the movement speed will lead to a dynamic animation.

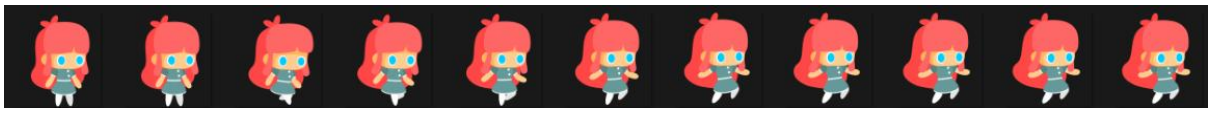


Image 10 example used texture frames for animation

Example set of textures for jumping.

```
1.  void Jump() {
2.  if (Input.GetButtonDown("Jump") && isGrounded) {
3.  player.velocity += new Vector2(0f, jumpSpeed);
4.  Girlanimation.SetBool("IsJumping", true);
5.  }else if (Input.GetButtonUp("Jump")) {
6.  Girlanimation.SetBool("IsJumping", false);
7.  }
8.  }
```

This is the jump function, simple but efficient, and the way it works is by an if conditon that contain a keylistener of type buttonDown, and isGrounded bool, this bool will be spoken about it later on because it is a very important bool.

Now for the buttonDown(“Jump”), in unity, familiar words are used to define keys that are already connected to the word in a way or another, hearing the word jump while in a video game, will always lead you to click on the space button, and that is exactly what unity did, made the word jump refer to the space button, and the getButtonDown is used because there is two actions that actually happen when you click a button, clicking down, then leaving it to go back up, and each action has its own listener, the GetButtonDown is when the button is pushed down, and getButtonUp is when the button is released to go up, and using these and an accurate value of incrementing on the Y-axis, I managed to make the player just jump to the point it reaches to the middle of the jump animation, because the second part of the jump animation is the landing animation, and on the getButtonUp, it just cancels the animation in all.

```
1)  void climbingLadder() {
2)  if (feetCollider.IsTouchingLayers(LayerMask.GetMask("Climbable"))) {
3)  moveDir = Input.GetAxisRaw("Vertical");
4)  Vector2 climbV = new Vector2(player.velocity.x, moveDir * climbSpeed);
5)  player.velocity = climbV;
6)  if(moveDir != 0){
7)  isClimbing = true;
8)  }
9)  Girlanimation.SetBool("IsJumping", false);
10) } else {
11) isClimbing = false;
12) return;
13) }
14) }
```

The climbingLadder() function, and the way it works, is by checking if the feetcollider of the player is touching a layer that has a mask called climbable, which is referring to the ladders objects in the scenes, and by that, it will start input listening to the vertical input, which uses W key and Up-arrow for 1, S-key and Down-arrow for -1, and zero for none, then creates a vector2 saving the player velocity on the X-axis, and the velocity on the Y-axis is the input listener multiplied by the climbSpeed float variable, and on line 7, just disables the isJumping animation if the player was jumping towards the ladder and tries to climb in the same time, or tries to jump while climbing.

```
void FlipSprite() {
 bool playerHasHorizantolSpeed = Mathf.Abs(player.velocity.x) > Mathf.Epsilon;
 if(playerHasHorizantolSpeed)
{ transform.localScale = new Vector2 (Mathf.Sign(player.velocity.x), 1f);
 }
 }
```

The FlipSprite() function, is a simple yet, one of the most important functions that was used in multiple other stuff in the project, basically what this function does, is checking which scale the object must have, since the textures are all in a movement to the right side, I had to find a way to flip the gameobject instead of re-scaling each image and create separate animations and separate key listeners with nested for loops to make look clean and smooth, so using this function I avoided all this work, and the way it works, is by checking if the player is moving (used and explained above), and if the player is moving, it will check for the sign of the velocity on the X-axis, left will return negative and

right will return positive, and by that, since scaling the object can take vector2 data type, I create a vector2, assign 1 to the y-axis of the vector2, and on the x index, I pass the sign of the velocity on the x-axis, which basically if it is negative (no matter the how small the value is but as long as it has a negative value) it will return -1, and if positive (no matter how big the value is as long as it positive) will return 1, then the vector2 gets assigned to the localscale of the player object.

Now to finish the player script, I will speak about the isGrounded bool, which was a great helper in developing the code once I sit it up, because using this bool, I can detect if the player is touching something I can consider as ground or not, that will help a lot with manipulating a lot of tricks in the development.

Basically unity has a set of functions called :-

1- OnCollisionEnter2D(Collision2D col)

2- OnCollisionStay2D(Collision2D col)

3- OnCollisionExit2D(Collision2D col)

These functions detect collisions between objects colliders, to detect, both colliding objects need to have a collider, and in these 3 functions, the first one, gets activated on the first frame of collision, the second on keeps on being activated until both objects stop colliding, and the third function will get activated the frame the collision stops, and the parameter will always refer to the colliding object, to get control over it, so basically what I did, is capturing the tag that the colliding object has, and gets it compared in an if condition full of or gates to check if it in the list or not, and using this concept, I managed to detect wither player is touching objects that are considered ground or not.

Here is an example of the function that is self-explanatory.

```
private void OnCollisionEnter2D(Collision2D col) {
 if ((col.gameObject.CompareTag("Grabable") ||
 col.gameObject.CompareTag("Doors") ||
 col.gameObject.CompareTag("Movables") ||
 col.gameObject.CompareTag("Joints") |
| col.gameObject.CompareTag("Ground")||
 col.gameObject.CompareTag("Buttons")|
| col.gameObject.CompareTag("planet")) &&
 !col.collider.isTrigger) {
 isGrounded = true;
 } }
```

Another script attached to the player is **the grab controller**, where it allows the player to grab a specifically tagged objects and move with them, and this happens by considering the player is the parent object, and on grabbing it makes the object a child of the object, where any affect on the parent will be inherited to the child.

```
1.  public Transform grabDetect;
2.  public Transform BoxHolder;
3.  public float rayDist;
4.  private GameObject heldObject;
5.  public bool isHoldingBox = false;
6.  void Update() {
7.  RaycastHit2D grabCheck = Physics2D.Raycast(grabDetect.position, Vector2.right *
    transform.localScale, rayDist);
8.  if (grabCheck.collider != null && grabCheck.collider.tag == "Grabable") {
9.  if (Input.GetKeyDown(KeyCode.E) && heldObject == null) {
10. heldObject = grabCheck.collider.gameObject;
11. heldObject.transform.parent = BoxHolder;
12. heldObject.transform.position = BoxHolder.position;
13. Rigidbody2D heldRigidbody = heldObject.GetComponent<Rigidbody2D>();
14. heldRigidbody.isKinematic = true;
15. heldRigidbody.velocity = Vector2.zero;
16. heldRigidbody.angularVelocity = 0f;
17. isHoldingBox = true;
18. } else if (Input.GetKeyDown(KeyCode.E) && heldObject != null) {
19. heldObject.transform.parent = null;
20. heldObject.GetComponent<Rigidbody2D>().isKinematic = false;
21. heldObject = null; isHoldingBox = false;
22. }
23. }
24. }
```

In the code snippet above, main concept is that there is 2 point objects attached to the player, where one is the grab detector, and the other is the boxHolder, where if the grabable object is in the ray distance of the grab detector point, it will be activated with a key listener, and you can see on line 7, this is where it checks if the box is in the raycast of the point, it creates a raycasthit2D object which returns a null on empty detection, raycast() function is used that wil receive 3 parameters, position of detection, and the grab detector point was passed to it, a localscale for detecting the direction of detection (left or right to the grab detect point), and a float that will determine the radius of triggering.

On line 8, an if condition to determine if the raycast is capturing something in its radius or not, so it gets check if it was null, and to make sure it is the right object that would be grabable, && gate with a tag check to make sure, if both true, then activate a keylistener for the E-key, and checks if the heldobject gameobject is equal to null, this gameobject is an empty child of the parent, which will be used as a container, now checking if it is null which means it is empty, on line 9, it will take the object

that is detected and assign it to the heldobject gameobject, and then change the parent of the heldobject to the box holder point, and after that some modification act on the grabbed object, which are :-

1- Transforming its position to its new parents position

2- Zeroing its velocity if it has

3- Disabling its rigidbody2D by turning it to kinematic

4- Zeroing its angular velocity

And now on dropping the object from player, gets input from a keylistener, and checks if the heldobject != null, which means there is an object that is being held by the player, it will return all the specifications of the grabable object back to original, and making it a parent instead of a child.

‣ **building the first heavy level.**

The amount of code was big and was used a lot of times due to the fact there is multiple objects in the scene , and each object has its own script for making it functional, so I will go with a diagram to explain the **platformer physical puzzle in the first level**, that will demonstrate how does the level work.

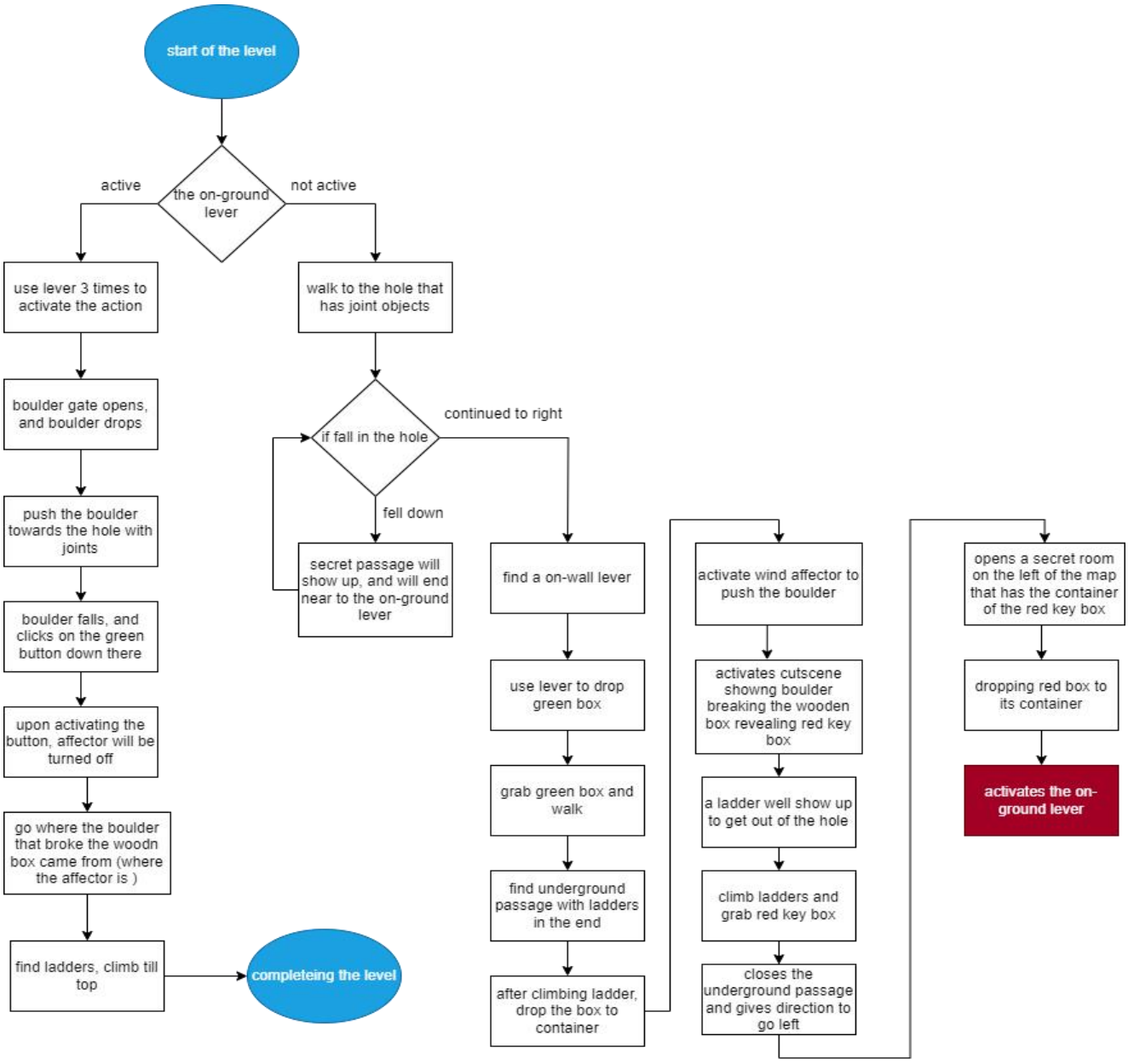


Image 11 flow chart for platfromer puzzle

In this diagram, actions where all connected smoothly with code, can be little tricky on reading, but on gameplay, is some sort of natural to solve, where player can never be in a place that he is not intended to be at, and by that it will give some sort of open world feeling, that wherever you go to discover you will end up with a part of the puzzle.

The way it was done by placing 2 keys and their containers in different places, and player needs to use the environment around him to get the keys, where getting the green key will require the usage of the lever, and all those actions followed the inheritance concept in coding, where bools are the main source of signals between different objects.

I made cutscenes to see what actually happens so the player can follow up what should be done next.

The concept of the level was not planned from the beginning, core idea was there, but method of implementation was following inspiration step by step where in every step, I come up with the next one, which toke some time to come up with, but the output was very worth it.

Here is one of the scripts in this level, this script is for the **wooden box** that hides the redbox key, that gets destroyed by the boulder.

```
1.  public Sprite[] boxStages;
2.  public float delayTime = 4.0f;
3.  public GameObject tilemap;
4.  public GameObject playerCam;
5.  public GameObject boulderCam;
6.  public GameObject playerCol;
7.  public AudioSource sfx;
8.  private int currentStage = 0;
9.  private SpriteRenderer spRender;
10. private bool isCollided = false;
11. private float timer = 0;
12. private void OnCollisionEnter2D(Collision2D other) {
13. if(other.gameObject.tag == "Movables"){
14. isCollided = true;
15. sfx.Play();
16. StartCoroutine(textureLoop());
17. this.gameObject.GetComponent<BoxCollider2D>().enabled = false;
18. this.gameObject.GetComponent<Rigidbody2D>().bodyType = RigidbodyType2D.Static;
19. }
20. }
21. private void OnCollisionStay2D(Collision2D other) {
22. if(other.gameObject.tag == "Movables"){
23. isCollided = true;
24. }
25. }
26. void Start() {
27. spRender = GetComponent<SpriteRenderer>();
28. spRender.sprite = boxStages[currentStage];
29. }
```

```
30.  void Update() {
31.  if(isCollided){
32.  StartCoroutine(Fader());
33.  }
34.  }
35.  IEnumerator Fader() {
36.  Color originalColor = spRender.color;
37.  float t = 0f;
38.  while(t < delayTime){
39.  t += Time.deltaTime;
40.  float normalizedTime = t / delayTime;
41.  spRender.color = Color.Lerp(originalColor, Color.clear, normalizedTime);
42.  yield return null;
43.  }
44.  boulderCam.SetActive(false);
45.  playerCam.SetActive(true);
46.  playerCol.SetActive(true);
47.  Destroy(this.gameObject);
48.  Destroy(tilemap);
49.  }
50.  IEnumerator textureLoop(){
51.  for (int i = 0; i < boxStages.Length; i++) {
52.  spRender.sprite = boxStages[i];
53.  yield return new WaitForSeconds(0.2f);
54.  }
55.  }
```

In this script, what happens is that it checks if the object that is holding this script got hit by a boulder, and this is the main starting condition of all the functions, now on this script, a new type of functions is introduced, an IEnumerator function, this function was used a lot of times in this project, different purposes of course, but same concept, where this function is like a routine, will only start if it is requested to start, this function is capable of delaying the update function, to make it more sensable I will break one of the IEnumerator functions in this script.

Taking the fader IEnumerator, it starts by creating a new color data type, that will capture the color of the sprite in RGBA, then a while loop with a timer and a time limit in the condition part, and inside the loop, the timer receives real time into it, and a new float that will hold the time difference between real time and the delay limit, and will pass it to the color.lerp function , which take 3 parameters,

firstly the original color, then final color, then amount of lerp, and the amount of lerp is the time difference in this situation, to give a smooth fading effect to the object's color, and on line 42, yielding is basically waiting (putting on hold), where using it inside the while loop as a null return, will not wait for any amount of time, so when the while loop finishes, it will move to the camera switchers and objects destroying and then the function will break.

And on the second IEnumerator function, you can see on line 53, yield return is returning a waitforseconds() function, which is basically forcing the time of the game to wait for the required amount.

‣ **Building the Free-Faller level.**

During this level, I was firstly introduced with the UI, buttons, texts and all different sorts of stuff, so It was my first attempt on creating a functional button, after designing the level, i added spikes that will pop the bubble to give a challenge for the player, so after designing I made **this script for all the spikes,**

```
1)  public GameObject bubble;
2)  public GameObject button;
3)  public float timeDelay = 3f;
4)  public AudioSource sfx;
5)  private Vector2 startpos;
6)  private bool isPopped = false;
7)  private void OnTriggerEnter2D(Collider2D other) {
8)  if(other.gameObject.tag == "Bubble"){
9)  isPopped = true;
10) sfx.Play();
11) }
12) }
13) private void OnTriggerExit2D(Collider2D other) {
14) if(other.gameObject.tag == "Bubble"){
15) isPopped = false;
16) }
17) }
18) void Start() {
19) startpos = new Vector2(bubble.transform.position.x, bubble.transform.position.y);
20) } void Update() {
21) if(isPopped){
22) bubble.GetComponent<Rigidbody2D>().bodyType = RigidbodyType2D.Static;
23) bubble.transform.position = startpos; bubble.transform.rotation = Quaternion.identity;
```

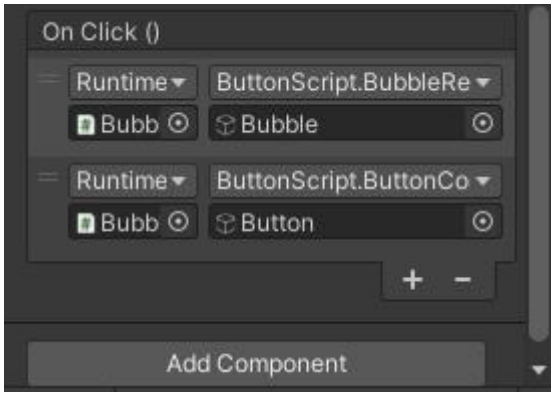
```
24)  button.SetActive(true);
25)  }
26)  }
```

On this script, made it capture the start position of the bubble in the start function to act like a respawn point using a vector2, made collision detectors for the spikes, and upon collison with any of the spikes, it will activate the button, and respawns the bubble with a fixed rotation and a fixed position, and most importantly in a static bodytype, because I wanted to use the button to turn it back to its original dynamic bodytype.

```
1)   public void BubbleRelease(GameObject bubble){
2)   bubble.GetComponent<Rigidbody2D>().bodyType = RigidbodyType2D.Dynamic;
3)   }
4)   public void ButtonController(GameObject button){
5)   button.SetActive(false);
6)   }
```

The way **UI buttons** work in unity, is you call out functions using them, and in these functions you determine the functionality of the button, the button functions gotta be in a script that is attached to an object that must stay active during the whole scene, so usually they attach such a script to an empty useless object that is only there to provide the button its functionalities, by that, attaching the the object that has the functionalities to the button, and choosing them from the list button as shown in the image below.

Image 12 Button function component



‣ **Building the space level.**

This part of the project was the most time consuming and the most part that was put ideas, thoughts, and work on, because I wanted to come up with something that is unique and unfamiliar to the consumer, so I came up with the idea of the planets.

Now unity physics engine is smart when it comes to gravitational force, but the problem with it that since usually the gravitational force in a scene is aiming downwards, on the positive Y-axis, all what it does is assigning a constant velocity to the object, and this velocity is changeable with the mass of the object, incrementing of the mass variable will lead for a stronger gravity effect downwards, and I wanted to come up with a new physics method.

So basically I had to re-implement any movement script, specially the player script, which will be shown and explained soon, but firstly lets talk about the planets, the level consists of 4 different planets, each with a different size, different gravitational force value, and different gravitational force radius, and the way I have made it work is using this script.

```
1.  public float graForce;
2.  public float planetR;
3.  public float objectRotationSpeed = 2f;
4.  private void OnTriggerStay2D(Collider2D other) {
5.  if(other.gameObject.tag != "Bullet" && other.gameObject.tag != "white-Bullet"){
6.  Vector3 dir = (transform.position - other.transform.position) * graForce;
7.  other.GetComponent<Rigidbody2D>().AddForce(dir);
8.  if(other.CompareTag("Player") ||
9.  other.CompareTag("Grabable" )||
10. other.CompareTag("rocket") ||
11. other.CompareTag("Alien") ){
12. other.transform.up = Vector3.MoveTowards(other.transform.up, - dir, graForce * Time.deltaTime
    * objectRotationSpeed);
13. }
14. }
15. }
16. private void OnTriggerEnter2D(Collider2D other) {
17. if(other.CompareTag("rocket")){
18. other.GetComponent<RocketScript>().enabled = false;
19. other.GetComponent<Rigidbody2D>().velocity = new Vector2(0f, 0f);
20. }
21. }
22. void OnDrawGizmosSelected() {
23. Gizmos.color = Color.blue;
24. Gizmos.DrawWireSphere(transform.position, planetR);
```

```
25.  }
26.  }
```

This script is attached to every planet, and as I explained earlier, any public values can be modified in the unity platform instead of going to the code, so what happens in this code is that having 2 main floats, determining the force of the gravitational field, and the radius of the field itself, now every planet has 2 circular colliders, one will act like a trigger, and the other is an actual collider, the trigger collider's role is detecting intersection without an actual collision which can be useful in a lot of cases and this case is one of them, now what happens on triggering in the onTriggerStay2D() function on line 4, is firstly the comparison if conditions to make sure that the gravitational force is activated on the right objects, so I excluded bullets (that later on were modified to bubbles), and checked for each object I wanted the effect to be applied on.

The way it starts is by creating a vector3 data type called Dir short for Direction, which is a variable that can hold (x , y , z) inside of it, and saving saving in the Dir the distance between the center of the planet and the triggering object multiplied with the gravitational force, which is basically the original gravitational force equation = gravitation constant (substituted with gravitation force float) * mass of first * mass of second * distance between them squared, masses were 1's for each objects, and the distances was not squared because I didn't want the gravitation force to be very heavy, so I used it un-squared.

Now I added the direction vector3 as force to whatever triggering this function, and by that what happenes is that .addForce() function can receive a vector3 as a parameter, multiplies the components of the parameter with the velocity of the triggering object, and velocity is a vector3 variable, simple.

After doing that part I started noticing that the rotation of the object is not always accurate and I will explain it with a simplified drawing

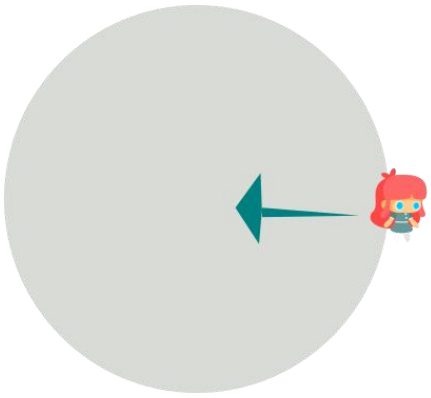


Image 13 demonstration of gravity output

the blue arrow shows the direction of the gravitational force, and as it is shown in the image, the player is sticking onto the surface of the planet, but not in a way I wanted, I wanted the player to look like he's walking on the surface of the planet, where attracted bottom first to the planet just like the image below

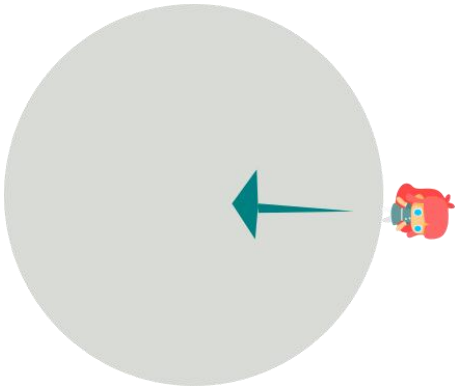


Image 14 demonstration of output

and to make such a thing possible, line 12 in the code makes that happen, where it refers to the up direction of the object (no matter the rotation is, if it was rotated 90 degree's the up will be to the left), by other.transform.up is the refer, and then I change it's value with a vector3 that will use a function called moveTowards(), this function allows smooth movement between two positions, where it has 3 parameters, and the return of this function is more accurate distance depending on the parameters, which are :-

1- First position (in this case it is the up position of the object)

2- Second position (in this case it is the negative value of the calculated direction earlier)

3- The maximum distance to move between the first position and the second position (which in this case it was the gravitation force float * real time (to increment by time) a float to accurately modify)

And using this function, it will fix the rotation for any object.

Then there is the ontriggerEnter2D() function, and the reason I used it is for the red rocket in the beginning of the level, I wanted the rocket to land, and when it lands the player spawns next to it as a theme, so using this function I made the rocket's scripts get deactivated, and zeroing its velocity too so it does not move during the landing, since I already made a function to fix the rotation, the rocket will always land on its bottom on every planet, so I managed to make it look like an autopilot for the landing.

And since unity's gravity engine is not used for this level, every type of movement needed to be reworked, starting with the player object itself, **here is a piece of the player script**

```
1)  public float moveSpeed = 5f;
2)  public float jumpSpeed = 5f;
3)  public GameObject playerCam;
4)  public GameObject gunGuide;
5)  public GameObject pausePanel;
6)  private Animator PlayerAnimation;
7)  private Rigidbody2D player;
8)  private float moveDir;
9)  private bool isClimbing = false;
10) private BoxCollider2D feetCollider;
11) private bool isGrounded = false;
```

```
12)  private float DirH;
13)  private float DirV;
14)  private SpriteRenderer sprite;
15)  private void Start() {
16)  player = GetComponent<Rigidbody2D>();
17)  sprite = GetComponent<SpriteRenderer>();
18)  PlayerAnimation = GetComponent<Animator>();
19)  feetCollider = GetComponent<BoxCollider2D>();
20)  }
21)  private void Update() {
22)  if(!pausePanel.activeSelf){
23)  DirH = Input.GetAxisRaw("Horizontal");
24)  if(isGrounded && Input.GetButtonDown("Jump")){
25)  player.AddForce(transform.up * jumpSpeed, ForceMode2D.Impulse);
26)  PlayerAnimation.SetBool("IsJumping", true);
27)  }
28)  else if (Input.GetButtonUp("Jump")) {
29)  PlayerAnimation.SetBool("IsJumping", false);
30)  } moveAnimation();
31)  playerCam.transform.rotation = this.transform.rotation;
32)  }
33)  } private void FixedUpdate() {
34)  if (isGrounded && !pausePanel.activeSelf) {
35)  player.AddForce(transform.right * DirH * moveSpeed);
36)  bool playerHasHorizantolSpeed = Mathf.Abs(player.velocity.x) > Mathf.Epsilon;
37)  PlayerAnimation.SetBool("IsWalking", playerHasHorizantolSpeed)
38)  ; if (DirH > 0) {
39)  transform.localScale = new Vector2(Mathf.Sign(DirH), 1f);
40)  if(gunGuide != null){
41)  gunGuide.GetComponent<RectTransform>().localScale = new Vector2(1f, 1f);
42)  }
43)  } else if (DirH < 0) {
44)  transform.localScale = new Vector2(Mathf.Sign(DirH), 1f);
45)  if(gunGuide != null){
46)  gunGuide.GetComponent<RectTransform>().localScale = new Vector2(-1f, 1f);
```

```
47)  }
48)  }
49)  }
50)  }
```

The biggest difference between the player script in the previous phases and this phase, is that the velocity functionality can not work properly, because unity's gravity is disabled, which means change on the Y-axis of the object does not automatically change, so I had to remake it all manually with force apply functions, it was very simple on the jumping mechanism, where a key listener with adding simple force on the Y-axis and activating the animation the same way it was done before, but on the movement it was another challenge, where force functions can not be accurate sometimes on the update function, so I had to use a fixedUpdate() function, and the reason behind that, is that the FixedUpdate() function does not update every frame, it updates every specified amount of time which can be changed in the unity's platform settings, using the default did the trick for me which is 0.02 seconds, just used an input listener, with a force applied to the right direction of the player, and a re-scaler that was used in previous code, and the isgrounded follows the same way it was done before.

Since no combat game works without an enemy to defeat, **I have created some bots**, a semi AI objects that move, stop, shange animation, shoot, and die.

Here is an image of what actually happens behind the scene

Image 15 bot colliders

Bots are surrounded bygreen edged boxes, these green edged boxes are transparent colliders that only the bots can collid with them, and the reason behind me using this, is so the bot changes the direction of its movement, using a collision detection, it will be easier to manipulate a switch.

And **here is the code for the bots**, this script was used for all of them, which some changes in the values of course.

The **update()** function

```
1.  void Update() {
2.  SFXtimer += Time.deltaTime;
3.  float distance = Vector2.Distance(transform.position, player.transform.position);
4.  if(distance <= shootRange){
5.  timer += Time.deltaTime;
6.  float distanceX = player.transform.position.x - transform.position.x;
7.  if(timer > bulletCycle && !isShot){
8.  timer = 0;
9.  pullTheTrigger();
10.  lazerSFX.Play();
11.  }
12.  if (distanceX > 0 ) {
13.  transform.localScale = new Vector2(Mathf.Sign(distanceX * scaleFixer), 1f);
14.  } else if (distanceX < 0 ) { transform.localScale = new Vector2(Mathf.Sign(distanceX * scaleFixer),
     1f);
15.  }
16.  isInRange = true;
17.  } else{
18.  isInRange = false;
19.  if(movementSwitcher > 0){
20.  transform.localScale = new Vector2(Mathf.Sign(movementSwitcher), 1f);
21.  } else if(movementSwitcher < 0){
22.  transform.localScale = new Vector2(Mathf.Sign(movementSwitcher), 1f);
23.  }
24.  }
25.  if(isShot){
26.  StartCoroutine(DeadProcess());
27.  }
28.  }
```

On this snippet, a new float is created with every updated and will save the distance with the vector2.distance() function that takes two parameters and calculates the distance between them, and in this case, it takes the player distance, and the object holding this script, and with a specified float to

determine what is the acceptable distance called shootRange, which is a raduis, because the vector2.distance() returns a radius value, and with this method, player detection is achieved.

now when the player is in range, a timer for the sound effects starts adding real time to it, and a new distance float that holds just the distance on the X-axis between the player and the object that is holding this script, and this distance will be used to rescale the sprite of the bot, where if the player gets in range, the bot will immediately change its scale to look at the player direction, after that an if condition is called to check if the bot is allowed to shoot or not, with a delay between each shot, and a condition to check if the bot got shot by the player or not, pullTheTrigger() function is called, which will be brought soon.

On the movement switcher, this one changes its value with the collision function detector, where in the screenshot above, there was multiple collider boxes, and whenever the bot collides with any of these boxes, the movement switcher gets multiplied by -1, to switch direction, and the same value was used in the upcoming function the **fixedUpdate() for the movement.**

```
1)  private void FixedUpdate() {
2)  if(isGrounded && !isInRange){
3)  enemyRB.AddForce(transform.right * movementSwitcher * moveSpeed);
4)  if(SFXtimer >movementSFXDelay){ movementSFX.Play(); SFXtimer = 0;
5)  }
6)  }
7)  if(!isInRange){
8)  ShooterAnimation.SetBool("IsWalking", true);
9)  ShooterAnimation.SetBool("IsShooting", false);
10) }else if(isInRange) {
11) ShooterAnimation.SetBool("IsShooting", true);
12) ShooterAnimation.SetBool("IsWalking", false);
13) }
14) }
```

On this function, the control over the movement is done, basically the holder of the script will always move to the right multiplied with the movementswitcher explained above with a specific speed as long as the player is not in range and the bot is grounded, is grounded is determined the same method all the isGrounded bool is determined in the previous scripts, and IsInRange is calculated in the Update() function above.

Now with the player is in Range, the walking animation will stop and the shooting animation will be on

For the PullTheTrigger() function.

```
private void pullTheTrigger() {
 Instantiate(bullet, bulletPos.position, Quaternion.identity);
```

```
}
```

Basically all what it does, is calling the instantiate function, this function create an object automatically with a specific parameters, and there parameters are :-

1- The object reference

2- The position where the object will be spawned at

3- The rotation of the object.

And in this case, the bullet object is used, and the bullet Position object is a child of the bot where it is put on the bot gun, and the rotation is set to original using the quaternion.identity function.

Now since we spoke about bullets spawning, let me tell you there is 2 different bullet types in this scene, one that is shot by the player, and one that is shot by the bot, both are pre-created objects that were saved as prefabs, (re-usable edited objects) saved in files and not active in the scene, and both have very similar scripts but with small differences in the collisions, **so here is the script of the bullet that is shot by the bot.**

```
1.  private GameObject player;
2.  private Rigidbody2D bulletBody;
3.  public float bulletSpeed;
4.  private void OnCollisionEnter2D(Collision2D other) {
5.  Destroy(gameObject);
6.  }
7.  void Start() {
8.  player = GameObject.FindGameObjectWithTag("Player");
9.  bulletBody = GetComponent<Rigidbody2D>();
10. Vector3 direction = player.transform.position - transform.position;
11. bulletBody.velocity = new Vector2(direction.x, direction.y).normalized * bulletSpeed;
12. float rotation = Mathf.Atan2(-direction.y, -direction.x) * Mathf.Rad2Deg;
13. transform.rotation = Quaternion.Euler(0, 0, rotation + 180);
14. }
```

On this script, first things first, that on any collision with any object the bullet will immediately be destroyed.

And since such an object will be instantiated, all its properties will be taken from the instantiate function, so there is no need for an update() function, just on the start of it spawning it will find the the object that has the player tag and create a vector3 called direction, and saves in it the distance between the player object and the instantiated bullet, then add a velocity vector2 with the direction.x and direction.y multiplied by the desired bulletspeed.

Now talking about a bullet, it is necessary to speak about its gun, I have added a gun for the player, and player will be able to shoot from it, now to differentiate between what the player shoots and what the bots shoot, lets call player's bullets white bullets, and bot's bullets purple bullets, because in the game both have the same textures.

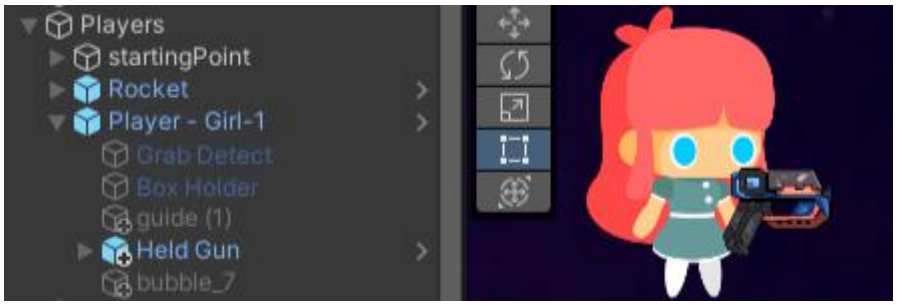


Image 16 player object's children objects

The player gun is a child object of the player object, which will make it much more easier to take control over the gun and the way it functions, now here is the code snippet that makes the gun work.

```
1.  public GameObject bullet;
2.  public GameObject guide;
3.  public Transform bulletPos;
4.  public AudioSource lazerSFX;
5.  private bool isHoldingGun;
6.  public float bulletSpeed = 5f;
7.  public float bulletDelay = 5f;
8.  private bool isAllowed = true;
9.  private bool isTriggered = false;
10. void Start() {
11. guide.SetActive(true);
12. }
13. void Update() {
14. if(Input.GetKeyDown(KeyCode.Q) && isAllowed){
15. Destroy(guide);
16. pullTheTrigger();
17. lazerSFX.Play();
18. isAllowed = false;
19. Invoke("pullingAllowed", bulletDelay);
20. }
21. if(guide != null && gameObject.transform.localScale.x == 1){
22. guide.transform.localScale = new Vector3(1f, 1f, 1f);
```

```
23.  } else if(guide != null && gameObject.transform.localScale.x == -1){
24.  guide.transform.localScale = new Vector3(1f, 1f, 1f);
25.  }
26.  }
27.  private void pullTheTrigger(){
28.  Instantiate(bullet, bulletPos.position, Quaternion.identity);
29.  }
30.  private void pullingAllowed(){
31.  isAllowed = true;
32.  }
```

Alright, now over here is the gun script, now immediately when the gun spawns, due to the start() function, the guide object that says how to use the gun spawns, and since it is a child of the gun, it will be moving with the gun, which is the child of the player.

In the update() function, basically a key listener for the key G, with a bool that determines if it is allowed to shoot, because I put a delay between each bullet shot, we will come to that part soon, now if player shoots, the guide gets destroyed, and the pullthetrigger() function will be activated, and the shooting sound effects will player, and then a special function called invoke(), this function literally invokes within the update() function, it takes 2 parameters, a string of a name of a function that exists in the same script, and a float value that will act like a delay, now what it does is that it calls the function with the same name as it is passed in the string parameter, but only calls it after a certain amount of time, and that amount of time is decided by the float parameter, where every complete number is considered a second, 1 is a second, 0.5 is half a second and what so.

Now before calling the invoke() function, I switched the isAllowed bool to false before it, because the function that is called in the invoke() function resets it back to true, which acts like an efficient delay.

Now for the pullthetrigger() function, what it does is simply instantiate the white bullet, sets its spawning position, and fixes its rotation, the same way it was explained with the bot.

And the one major difference between the white bullet (the player bullet) and the purple bullet (The bot bullet) is that the purple bullet tracks the position of the player, but the white bullet follows a child point object to the gun that acts like a direction and a way to destroy the bullet too so it does not run to infinity if it does not hit anything.

‣ **Building the UI, Menu System and pause system.**

Here is some screenshot’s of the Menu system

Image 17 Main Menu

This is the start boot up menu, where I added some animation for the project name, the background, since the backgrounds are layers not a solid picture, I managed to animate it in a free smooth way where clouds move constantly.

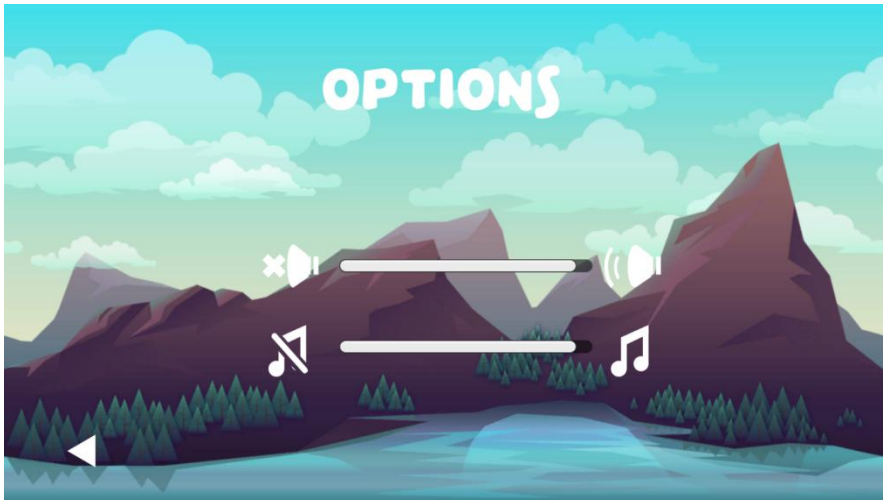


Image 18 Options menu

This is the options menu, yet a lot to come to this specific menu in future updates after the graduation, but for now it is fully functioning, changing the background music and the sound effects of the game, and whatever input you choose using the sliders in the screenshots, gets saved and used in every level, to avoid fixing the level ratio every level you load in.

I also animated a load animation for this screen, for components to enter smoothly instead of just popping out of no where, including the exit animation for going back to the main menu using the arrow on the bottom left corner.

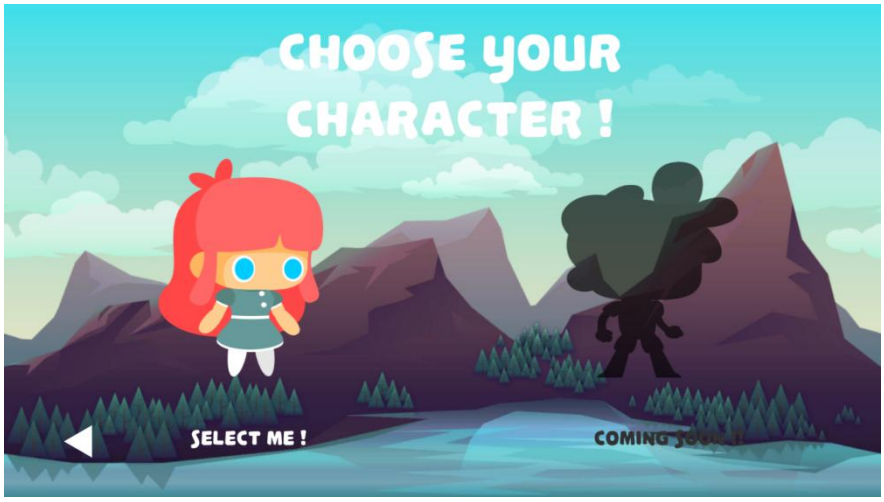


Image 19 Character choosing menu

After clicking the start button in the main menu, **the choose your character screen** pops up, and basically it lets you choose the character you want to play the game with, second character is half way in the development, yet I animated both characters in this screen to be idling until selection, with animation for components of this screen to enter and exit in a smooth way.

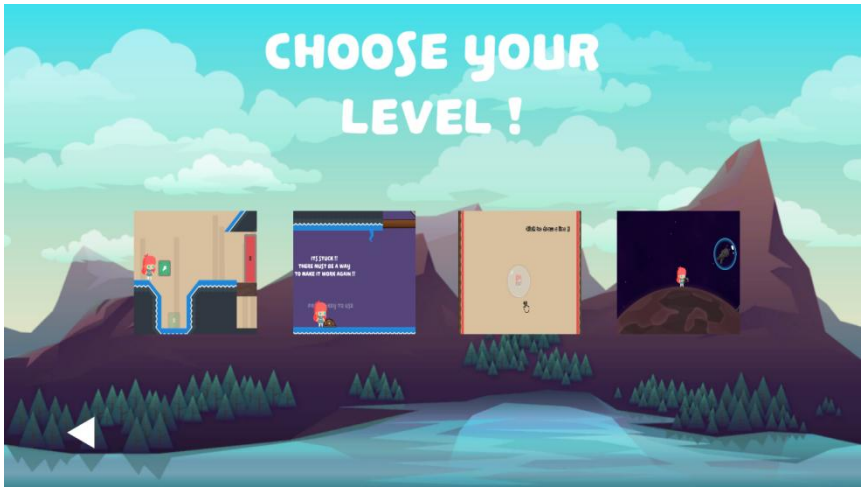


Image 20 Level choosing menu

This is the final UI before loading the game, in this one, it gives you the ability to choose between levels, pictures are clickable, which makes them the buttons to load the levels, not to forget that everything is fully animated, entering this screen, or leaving it using the arrow.

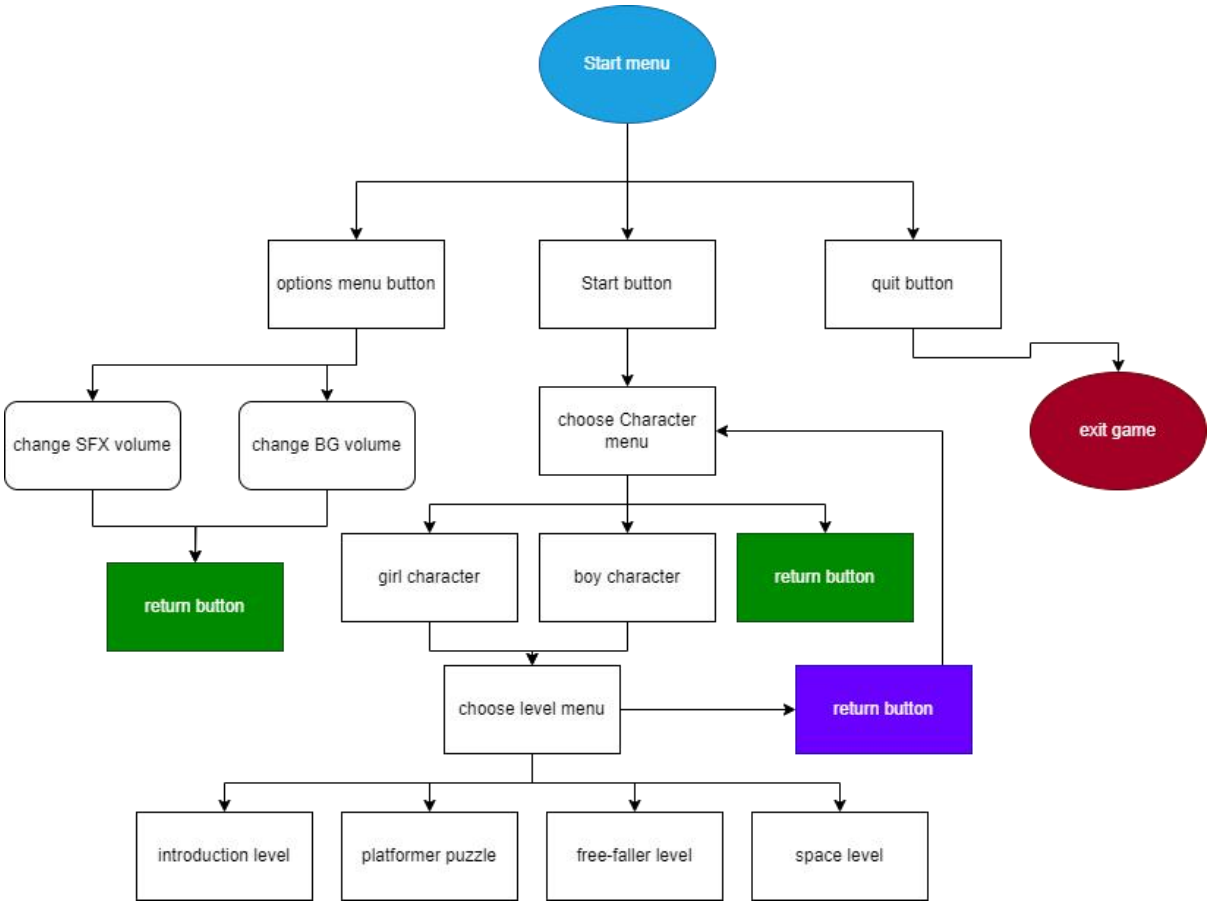


Image 21 GUI flow chart

Here is flow chart of the whole GUI system, where green return buttons take you back to the main menu (start menu).

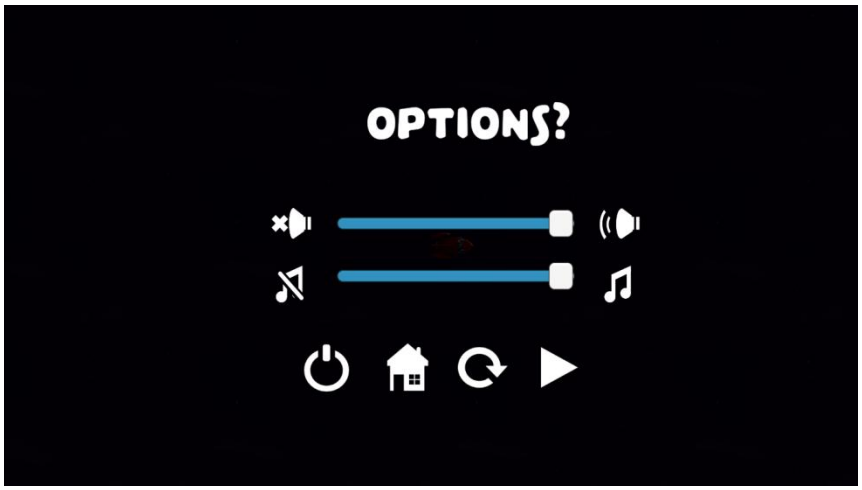


Image 22 Pause menu in-game

And over here is the pause menu where it freezes the gameplay, using it allows the player to quit the game, return back to start menu, repeat the level, exit the pause panel, while the ability to modify on the sound system of the game.

5. Conclusion

5.1 Benefits

a. Benefits to users :

1. a fun and interactive way to learn basic physics for children.

2. taking advantage of user free-time to come up with something beneficial

3. gaining parents acceptance and consent for using such products (due to the bad image such products have in parents eyes).

4- fun, easy to understand and to relate to (in sort of activities in the game), and I way to improve the child memory and creative thinking.

b. Benefits to me :

You should explain the benefits to you, what will you learn, etc.

1. I have gained a new experience in a field I have never stepped my feet in before

2. learned a new programming language, C#.

3. I became better with dealing with framework projects, now I have a very solid background and base structure to solve a good number of problem.

4- I got the chance to practice on a heavy and well known platform such as unity, which will open a lot of doors for me in future.

5- learned how to use photoshop, and other types of applications for dealing with image editing.

5.2 Ethics

You should explain the ethics of your project area, ethics should be supported with your resources

- Child safety and protection :
    - Creating an environment for gameplay that fits the age genre I am aiming at, which must not include anything that would harm the child mentally, or open him up to stuff that can not be digested by his age.
- Positive learning environment :
    - Learning material is introduces as a part of the gameplay, instead of introducing it as a learning material, it should be introduced in an embedded way inside the fun and entertaining sections of the gameplay
- Not too hard, not too easy :
    - The point of the game is to enhance the brain abilities of the consumer, which wont be achieved by a very direct solving technique, and in the same time not an impossible solution that will devastate the consumer, puzzle is hard, but clues are there to help, which will encourage the consumer to read and learn, understand and enjoy the satisfaction of solving the puzzle.
- Privacy, and data protection :
    - Including any malware that would harm or gain benefit for the developer by embedding it in the system without the consumer (or who is responsible of the consumer) is an illegal activity and it is punished by law, which was completely avoided in such a project, for the fact such a project has been made with clear and good intentions in the project purpose.

Why did I choose this project?

Well, multiple reasons starting off with the reason that have attraction the bigger portion of the developers around the world to the world of development in any field, which is video game and the concept of building it, and I am sure that the idea of giving it a try to develop a game have came across everyone's mind, including me, and it was a big motivation for me to take such a path, keeping in mind I did not have any experience or any previous knowledge about this field before, so it was a challenge that was very fun to take.

It grew a big number of skills and opened even bigger number of opportunities for me, since right now I can proudly say, that during my experience in the university, I have had a bite from every development bath plate that exists nowadays.

It was a great way to show my skills and what I am capable to do when it comes to come up with a functional product, I feel proud and confident about my choice.

5.3 Future Works

Such a project can be considered a long term project because the of many reasons :

1- I, personaliy had a good time coming out with the prototype I am submitting as my graduation project, calling it a prototype meaning that there is a great amount of possibilities and creative idea's that can be added to improve and to enhance the gameplay.

2- Such a project is an open project, where an end to it will be very far in the future.

3- The goal of the project is noble, and actually prov ides a product that would leave a good effect for the up-coming generations in the name of entertainment, where it can be something wonderful to provide such an entertainment type of products to existing family members, and the ones that would come in the future.

4- Can be used as a reference to my skills and what I am capable to do and what I am capable to provide of my skills to benefit and improve my field of work.

6. References

You should write your references which you found at your research and literature survey. You can use APA 6 Format

[1] : https://www.retrogamer.net/retro_games00/the-lost-vikings/

[2]: https://www.codester.com/items/32970/free-fall-2d-game-template-for-unity

[3]: https://brendangass.itch.io/project-moon

[4]: https://kenney.nl/assets/abstract-platformer

[5]: https://opengameart.org