Grau
**Multimèdia, Aplicacions i Videojocs**
FACULTAT DE CIÈNCIES I TECNOLOGIA
UVIC | UVIC·UCC

# THORNY SEASONS

NEREA WENTING LLUGANY MONTOYA AND JUDIT MORAL CAMPOS

Videogame Programming

Jose Díaz Iriberri

Universitat de Vic, May and 2024

## 1. Introduction / Goals

In this practice we were asked to create a game in Unity 2D, in which there had to be a long level, a final boss, a ranking with at least the 3 best scores, a start screen, an end screen, enemies, attacks, control of lives, and that it had audio. To meet these requirements we developed a game we call "Thorny Seasons", it is a 2D platform game developed in Unity with the aim of providing an immersive and exciting experience to players. Where you will have to explore the unique challenges that each season of the year presents. From the warm summer to the blossoming dawn of spring, players will face obstacles and enemies as they progress through this diverse world full of surprises. But they will have to be careful with their lives, because if they lose all of them they will have to start the adventure again from scratch unless they use the heart store. At the end of their adventure they will encounter an angry boss who will not make things easy for them to complete the adventure. The objective is to complete the entire game as soon as possible to appear in the final ranking where the 3 people who have completed the game in the least time will be shown.

To meet all the requirements requested in practice and therefore also the objectives, our game consists of 4 different levels that between them make more than one long level, it consists of different enemies, a start screen, a map where you can choose what level to play, a store where you can buy lives, a final screen where the 3 best times in completing the game will be shown, a final boss, audio for the different actions of the character and background music and a UI in which you will see the control of the lives you have and the coins you have obtained throughout the different levels, as well as the time it takes you to beat the game.

## 2. Project Development

To start developing the project, first of all we started looking for some assets that fit with the idea we had, which was to make a 2D platform game that was inspired by the seasons of the year. After searching in the Unity Store we found some

called "Free Platform Game Assets", where there was a main person, an enemy that was a square block with spikes, some spikes and the different scenes inspired by the different seasons. There were assets that were not included, such as the yellow enemy that walks from side to side and if it touches you, it takes a life from you. We had to do this using Illustrator, as well as the character's shooting animation, the gun, the bullet. and the explosion of the bullet upon collision with an object.

Once we had all the assets, we began assembling the different levels and scenarios. We started with the levels we were going to play and then the start screen, the map where you could select which level you wanted to play, the store and the end screen. Next we placed all the necessary colliders and made the UI of the game, placing where the coins, the stopwatch and the 3 hearts were going to be collected.

Then we made all the necessary animations, which were those of the character, enemies and chests. Once this is done, we begin to make all the necessary scripts for the game to work perfectly and apply playerPrefs to save the information.

Finally we tested the entire game to see that everything worked perfectly and fixed everything that wasn't quite right as well as placing all the necessary sound effects as well as the background music.

## 2.1. Game Design

The general design of our game is based on a 3D style but applied to 2D, in this way our world had more depth and was more attractive but without losing the objective of the practice, which was to make a 2D game.

To offer a much more immersive experience that would not be the same all the time, we decided that each level would have a different design but maintaining the same essence, for them each level is inspired by a different season of the year and thus making you go through all the seasons. of the year until reaching the end of the game.

Also to get the attention of the players, our game gradually increases the difficulty, making it more and more visually attractive due to the increase in spirit and

elements within the game. Below we will explain how each level and each scene that can be found in our game has been designed.

- Home screen: On this we find our character, a fairly simple red doll but that fits perfectly with the aesthetics of the game. In this one she appears blinking and still under a tree, while with an animation a transition is made from station to station showing you what you will be able to experience within the game. At first it is summer, then it is autumn, then winter and finally spring. Apart from this we can also find a start button that if you press it takes you to the next screen
- Game map: In this scene you find a level of platforms in which you can move through them and there are different doors that indicate what level each one is. The first one we find is level 1 and we can see that the rest have a fence and that if you try to access it it will not allow you. Once the first level is completed, this door changes to green to indicate that it has been completed and the possibility of accessing the next door opens. At the end of everything we can see that there is a door that is the store, that door is always open and gives you access to the store where you can buy lives. And onces you have completed the level you are able to repeat again the level in case you haver un out of coins and you want to buy more hearts.

- Level 1: This level is inspired in the summer, with all the green vegetation and well-populated trees. This level at the beginning tells you that it is an introductory level and shows you the controls that you can use to play and leaves you a space to try. It is the easiest level of all since it only consists of two enemies and they are placed in a very simple way so that our players can understand what is happening and what they have to do. In this level we find platforms on which you will have to jump, an enemy that walks and you will have to shoot him to kill him and some spikes so that the player sees that if he collides with them he will lose a life. They will also be able to see that there is water and that if they fall In it they will lose a life and will have to return to the beginning of that level.
- Level 2: This level is inspired in the autumn, everything is orange-brown and there are fewer flowers, since this season of the year is when the flowers begin to wither and the trees lose their leaves. At this level the difficulty increases a little more although it is still a flat level. Unlike the

previous level, two new enemies appear in this one, one is a square block with spikes that shakes for a few moments and then falls hard to the ground, if this enemy touches you you will lose a life. The second new enemy is a wheel with spikes that move in a straight line and go from side to side in certain parts of the level, if this wheel reaches the character it will cause them to lose a life. We can also find a modification in the spikes, towards the end of the level we can see that there are no spikes but the players will have to be careful since when they approach they will appear and they will have to be quick to avoid them.

- Level 3: This level is inspired in the winter, as the other two levels the color palette and enviroment follows the aesthetics of the real season in which is mostly cold colors such as blue and white tones. As the temperatures goes down, in this level we add new components such as snowman, ice cubes and igloos and therefore we also implement new mechanics that in the other levels doen't have. The roll of the ice cubes is to fall whenever the player touches it since ice is very frail. As for the snowman's roll is to guard the treasure of one coin, that in order to obtain it the player will touch the only element that really doesn't belong there. Another thing we added is the ice crystal, whose job is to also hide a precious treasure but for obtaining that the player will need to shoot at it. Finally, as the snow pile up we also add slopes.
- Level 4: This last level is inspired in the spring season, in which everything is nicer with more color, vegetation like flowers, trees and bushes and more difficulties. In this level is inttended to mesh up almost every mechanics of all the other levels so the Player can show that he/she has master every mechanic and since there is a better weather new things might happen such as a new friend coming after you. This friend is the last enemy the player must overcome in order to win and this frenemy's mechancs is to keep an eye on you until you are too close for him so he will start to chase you until you are close enough to slash one of this three attacks. The first one consists in a hammer that will take a life from you if the acomplish to smash you. The second one is a simple jump in which the boss will try to smash you with his butt. And lastly, his ace up to sleeve is the combination of the previous attacks edscribed before.

- Doors: In each level the player will enter from one door (the one at the beggining) and you will need to reach the other door (the one at the end) in order to omplete the level and get you to the game map. But the player will need to keep the hearts up to 1 so he/she can get through the next level and gather as much coins as he wants.
- Store: If you enter in the door that says “Store” you will see a new character all alone that is a staller and he is selling hearts, but do not be afraid since he can be your salvation as he will exchange one heart for 3 coins if you need it. But if you have the full life units he will deny to sell you more hearts and if you do not have enough coins he will not sell you anything since anyhting is free.
- Ranking screen: This screen is the last one in wich you will automatically go there if you run out of life units or you have defeated the boss. If you haver un out of hearts you will not enter in the top three ranking, however if you defeat the boss you will have a possibility to enter since only the three top festes plays will be shown in the ranking. It also have two buttons that one allows you to restart the game and the other to quit the application.
- User Interface: The UI of this game has a cartoony style and consist, on the one hand, in almost every screen, the life uinits in the right side of the screen and the number of coins in the left. In the center, the player can see how well he/she is doing since it is shown the time that has passed so far. Moreover, when the time comes that you need to face the boss or that you encounter an enemy the life bar of them will be also shown. On the other hand in the start and game over screen the follow the same aesthetics and they have two little animations, one for each and the buttons that will let the player start for the former screen and to restart or quit for the later screen.

## 2.2. Implementation

In order to achive all we wanted to do in the design what we did first was to chose the assets from the Unity Assets Store. Onces we decided which wanted we start designing the levels and for each sprite of the background we wanted to use we changed the resolution so the imatge would not be blurred once we resize it. For the elements that were not “humanoid” and do not needed animations we slice

the those sprites that had more han one sprite by automatically or by count cell. However, those that needed an animation we needed to slice in a more calm way since we need to wach out that they had the same size because if they did not have it they seem to be cold.

After finishing the slicing and changing resolution we started with designing the levels interface and since they were squares and we needed to build everything from scratch we decided to grop the onces that had the same functionality with an empty object for inctance the floor and the background.

Onces we placed everything we start doing both things putting the colliders and the animations at the same time. We put colliders in each place we thought we might needed and dedided wheather we should put it trigger or not. For exemple, the floor we decided to not make it trigger since the character needs to stand in the floor and not fall down. We also put a collider triggered and rigid body in each two legged enemy so it would follow the physics in the Unity engine, but in the case of the character we put another collider but triggered in the feet so it would allow to jump. And for all the other enemies, doors and other elements that needed interaction we also put a triggered collider.

So, in orther to make this triggered collider to work in the scripts we also put tags to all of them to identify the object that is colliding with them.

As said before, we also did at the same time the animations by dragging and droping the sprites that should have the animation. With this we created the idle, jump, walking and shoot animation of the character; the moviment of the yellow with a spike enemy; among other. But there are other that we could not do that since they were single sprites, so in order to achiveve this animations we di dit manually with the tool that Unity provide us called Animation, and thanks to that we could animate the other enemies, interations with the envitoment by changing some of their propierties like the position, color, scale, opacity, etc.

So the next step we porceed was setting the behaviour of the things in the game by scripting.

The first thing we did when we started programming was the character, since if we couldn't move we wouldn't be able to try anything else. So we programmed how he walked, jumped and shot. Consequently, we also had to program the

bullet because if we didn't do this we wouldn't be able to test whether it shoots correctly or not or whether it hurt the enemies and killed them.

Once we had our character ready we decided to continue through the enemies, for this we programmed that the yellow character would walk from side to side and that he would have a red bar above his head that marked the life he had and that if the character shot him this It was decreasing in a way that indicated how much life he had left. We also programmed the snowman and made it so that if he touched a flower the character would change state and let you go to pick up the chest. The last enemy we had to program was the spike wheel and we had to make it go from side to side and roll on itself.

Once we had the enemies programmed, we decided to also make the chests so that they would let you open them once and a coin would appear. After this we thought it was a good idea to program the SceneControllers to be able to see that if they inflicted damage on the player, a life would be subtracted and that if they opened a chest a coin would be added. So this was the next thing we did, just like if the player lost all the hearts he would have to start over from zero.

What I had to do next was the PlayerPrefs, to be able to see that if in a level you lost a heart or got a coin that it would update in all the other scenes, just like the timer, which would stop when you finished a level and would continue counting when you started other. The PlayerPrefs that we used were one to save the coins, another for the lives, another for the timer, another for the level you were in at that moment and another for the last level you had completed, which had to be the largest and never a previous one in case of repeating a level.

Once we had all this, we made the scene of the levels where the doors were. What we did in this one was that when you appeared for the first time you did so at the door on the first level, and the rest were blocked except for the store. And when you completed a level you appeared at that door and the next one was unlocked and the one you had completed was painted green to indicate that you had already completed that level.

As now we had more or less the entire game programmed, we went on to make the store, in this we made the player have to do what he would do in a small store and in this he would get a text about whether he could buy a life or not. What I told him was that if he had all the hearts he couldn't buy, if he didn't have three coins he couldn't buy but if he didn't have all the hearts and he had three coins he had to buy. If the character decided that he wanted to buy a heart, his lives were subtracted in the UI and a heart was shown in the UI.

Now there was only the boss, the home screen and the game Over. We decided to start with the home screen, where we took the 4 different backgrounds of the 4 seasons and programmed it so that it would make a transition and thus show the 4 different landscapes. We also programmed the start button so that if you pressed it you would It led to the screen of the levels where the doors were.

Afterwards we did the Game over, in this we used playerprefs again to be able to save the 3 best times, what we did was that when they left through the last door, the one at level 4, we took the total time, let's check if it was smaller than the first time what was in a PlayerPrefs, if it wasn't like that I looked at it in the second, if it wasn't like that I looked at it in the third and if not I discarded it. If it was smaller than the first, it updated the PlayerPrefs of that one with that time and finished, if not, it did the same with the second and the third. In the game over we also program the restart button to start over and restart all the player prefs except the 3 best times and it takes you to the start screen, and a quit button that does the same thing as the above but it closes the game.

Finally we made the boss, for this we programmed him to look at the character all the time and follow him, when he is at a certain distance from the player we make him perform one of the 3 attacks he has randomly and if he collides with the player it takes away his life. We also made it so that if the character shoots him, the boss's life will be reduced. To show the boss's life, we program, just as with the enemies, a bar that marks the life he has left and for each shot he receives he takes a piece off. Unlike the yellow enemy, where the bar is carried above his head, it appears on the Ui when the combat is going to appear, for this it makes it so that when the character collides with a collider the combat begins and this bar is activated. We also set this collider to stop being a trigger at the

moment the character exited and thus functioned as a barrier to prevent them from leaving the combat zone. Once you kill him, the exit door of the level is enabled and that is when time is saved.

Below we will explain in more detail what each of the scripts we have created does:

- **CharacterControllerScript**: controls the player character. Manage the character's movement, jumping and shooting, along with interaction with in-game objects such as the ground and ramps. It also manages the collection of coins and lives, the damage received by the character, and activates the fight against a boss if the player enters a specific area. It also gives the player an effect for each time they receive damage and gives them a few seconds without being able to receive damage to be able to leave that area
- **BulletControllerScript**:It is responsible for the behavior of the bullets in the game. Defines variables for the speed and life of the bullets, as well as their direction of movement. In the Start() method, initialize these variables. In Update(), updates the bullet's lifetime and destroys it if it has expired, as well as moving it in the set direction. When the bullet collides with the ground, it is destroyed. The SetDirection() method sets the direction of the bullet depending on the direction the character who fired it is facing. Finally, DestroyBullet() destroys the bullet and creates an impact visual effect on its position. Together, this script manages the behavior of bullets, including their movement, collisions, and visual effects.
- **BulletHitControllerScript**: Controls the behavior of bullet impact. It does not contain variable initialization or updates in Update(). The destroyHit() function is responsible for destroying the object to which it is associated. In short, this script simply destroys the object it is attached to when the destroyHit() function is called.

- **SellerScript**:The script controls the interaction with the seller, displaying a dialog box when the player approaches. When activated, the seller displays a message indicating whether the player can purchase an additional life and how to do so. If the player presses the "E" key and has at least three coins, they can purchase a life, which is reflected in the player's lives and coins updating. If you don't have enough coins, the player is informed. If the player presses the "S" key, he cancels the purchase.The seller activates when the player enters its range and displays a message appropriate to the player's conditions in terms of lives and coins. If the player has three lives or more, they cannot purchase more, and if they have less than three lives but do not have enough coins, they are informed of the purchase requirements.
- **YellowSpikeScript**: The script handles the lateral movement of the yellow enemy at a constant speed. When the yellow enemy collides with a bullet, he loses a life. If the number of lives reaches zero, the yellow spike is destroyed and the death animation appears. If the yellow enemy collides with the player character, the player loses a life. Additionally, the yellow enemy can change its direction of movement when colliding with certain objects in the scene. The script also contains a routine that temporarily changes the pickaxe's color from yellow to red when it takes damage.
- **SpikesHitScript**: The script is triggered when a spiked object collides with the player character. When this happens, the script calls the DecreaseLives() function of the CharacterControllerScript associated with the player character. This feature reduces the player's life counter.
- **SpikeControllerScript**: The script is triggered when the player character collides with the spikes. When this happens, the script enables the visibility of all the sprites of the selected peaks.
- **SpikeDieScript**: The script does not perform any action on the Start() or Update() methods, and its sole purpose is to destroy the yellow enemy when the destroyHit() function is called, which is when the death animation ends.
- **SnowmanControllerScript**: The script manages two SpriteRenderer objects that represent two different states of the snowman: one "good" and one "bad". At the start of the game, the "good" snowman is disabled and the "bad" snowman is enabled. Additionally, a boolean variable called

flowerTriggered is initialized that indicates whether the snowman has been activated by interaction with a flower.When the player character collides with the flower, the "good" snowman is activated and the "bad" snowman is deactivated, and flowerTriggered is set to true.

- **SawScript**: The script defines two private variables for the movement and rotation speed of the saw. Additionally, there is a boolean variable called turnRight that indicates whether the saw should turn right or left.In the Update() method, the saw continuously rotates in the direction set by the turnRight variable and moves in a straight line toward that direction. When the saw collides with an object labeled "TurnRight," it changes its turning direction to the right. If it collides with an object labeled "TurnLeft", it changes its turning direction to the left. If it collides with the player character, it calls the DecreaseLives() function of the CharacterControllerScript associated with the character to decrease the player's lives.
- **MaceHitScript**: The script is triggered when a mace object collides with the player character. When this happens, the script calls the DecreaseLives() function of the CharacterControllerScript associated with the player character.
- **HealthBarScript**: The script contains a private variable of type Image called barImage, which represents the image of the health bar that will be displayed on the screen.The UpdateHealthBar() method takes two parameters: maxHealth and health. This method updates the fillAmount of the health bar (represented by the barImage image) by dividing the current health by the maximum health. This ensures that the health bar updates correctly to reflect the character's current health level.
- **StartSceneScript** : The Start() method checks to see if the player has not set the "CurrentLevel" key in PlayerPrefs. If this key does not exist, three key values are initialized in PlayerPrefs: "CurrentLevel" is set to 0, "CurrentLifeHearts" is set to 3, and "CurrentCoins" is set to 0. This ensures that when you start the game for the first time, configure the necessary initial values.The startButtonClick() method is triggered when the player clicks a start button in the scene. This method loads the scene with index 1 using SceneManager.LoadScene(1), allowing the player to start the game.

- **SceneManagerStore**: The script defines several public variables, including arrays of GameObjects to represent hearts (both full and empty) and a text object to display the number of coins. It also has private variables to store the number of lives and coins of the player. The Update() method is responsible for continuously updating the player's lives and coins. Gets the current lives and coins values from the PlayerPrefs and then calls the DisplayLives() method to update the display of the player's lives. Also, if the player has more than 0 coins, it displays the number of coins in the textCoins text object. The Start() method initializes the player's lives and coins at the start of the scene. Calls DisplayLives() to ensure that the lives display is correct at the start of the game. It also sets some keys in PlayerPrefs, such as "Shoop" and "CurrentLevel", which can be used in other parts of the game.The DisplayLives() method is responsible for correctly displaying full and empty hearts according to the player's number of lives. Activates full and empty hearts based on the player's number of lives, making sure the display is accurate.
- **SceneMnagerLevel1Script**: This is the level 1 script, it defines several public variables, including arrays of GameObjects to represent hearts (both full and empty), a text object to display the number of coins, and another to display the time elapsed in the level. It also has private variables to store the player's number of lives and coins, as well as references to the character's controller and the level's starting door.

  The Update() method is responsible for continuously updating the player's lives and coins, as well as the time spent in the level. Gets the current lives and coins values from the PlayerPrefs and updates the lives display by calling the DisplayLives() method. It also updates the text that shows the number of coins and the time spent in the level.

  The Start() method initializes the player's lives, coins, and time at the start of the level. It also sets some keys in PlayerPrefs, such as "CurrentLevel", "CurrentLifeHearts" and "CurrentCoins".Maps the player's starting position to the position of the level's starting door and adjusts the character's jump speed.The DisplayLives() method is responsible for correctly displaying full and empty hearts according to the player's number of lives.
- **SceneMnagerLevel2Script**:This script does the same as the previous one, but is used to control level 2

- **SceneMnagerLevel3Script**: This script is used to control level 3, it begins with the declaration of variables for the ice crystals, the ice boxes, the initial and secondary ice boxes, the initial and secondary ice crystals, as well as for the hearts (both full as empty), lives, coins, coin text, and timer text. There are also variables to store references to the character controller and the level start gate, as well as the timer.The Start() method initializes the player's lives, coins, and time at the start of the level. Assigns the positions of ice crystals and ice boxes and sets their initial state. Also maps the player's starting position to the position of the level's starting door.The Update() method is responsible for continually updating the player's lives and coins, as well as the time spent in the level. Updates the status of ice boxes and ice crystals based on their interaction with the player. It also updates the text that shows the number of coins and the time spent in the level.
- **SceneMnagerLevel4Script**:This script is responsible for controlling level 4. It begins by declaring variables to represent the elements of the game, such as the player's hearts (both full and empty), lives, coins, the character controller, the start and end doors of the game. level, and the boss's GameObject. Additionally, there are variables to display the coin text and the level timer. In the Start() method, the player's lives, coins, and time are initialized at the beginning of the level. The player is placed in the starting position assigned by the starting gate, and the ending gate is disabled until the boss is defeated.In the Update() method, the player's lives, coins, and time are continuously updated. It is checked if the boss has been defeated, and if so, the final door is enabled. The logic when the player is damaged is also managed, and the texts showing the number of coins and the elapsed time are updated.

  The DisplayLives() method is responsible for visually displaying the hearts corresponding to the player's lives on the interface.
- **GameOverManagerScript**: This script manages the game when the player loses, displaying animations and updating time rankings if the player is at level 4. It begins by declaring several variables, including animators for the character and enemy, timers for the character's shot and jump. of the enemy, prefabs for bullets, references to the character's gun and variables for classification times.In the Start() method, you initialize several

variables and get the previous ranking times from the PlayerPrefs to display on the Game Over screen. If the player is at level 4, the ranking times are updated if the player's total time is better than the previous times.In the Update() method, the character's shot and the enemy's jump are handled. The character fires bullets at random intervals and the enemy performs a jump when hit by a bullet.

The ConvertTime() method converts a time in seconds to a format string of minutes and seconds.The OnTriggerEnter2D() method detects when a bullet hits the enemy and triggers its jump animation.The restartButtonClick() and quitButtonClick() methods are called when the restart and exit buttons are clicked, respectively.

- **GameManagerScript**:Controls the character's position and the appearance of doors and fences in completed levels. It also manages the display of the player's lives and the number of coins. It begins by declaring several variables, including references to doors and fences at different levels, as well as the character and its components.In the Start() method, initialize the variables and get the player's lives and coins. Then, set the character's starting position and define the current level. In the Update() method, check if the character is in a door and update the number of coins.The UpdateCharacterPosition() method moves the character to the gate position corresponding to the current level and adjusts the color of completed gates and fences.The ChangeColorDoor() and CompletedLevelFence() methods handle visual changes to doors and fences in completed levels respectively. The StartingPoint() method sets the initial state of the doors.The DisplayLives() method displays the player's lives in the user interface.The SaveStoreDoorPosition() method saves the position of the store door for later use.
- **DoorEndScript**: Manages the behavior when the character comes into contact with an end-of-level door. Upon contact with the door, it saves the player's current number of lives and coins to the player preferences and loads the next scene. If the current level is the final level, load the game over scene instead of the next scene.The Start() method initializes the reference to the CharacterControllerScript component. The OnTriggerStay2D(Collider2D collision) method is triggered when the player remains inside the gate area. If the collision object is the character

and the current level is not the final level, save the player's current lives and coins to the player preferences and load the next scene. If the current level is the final level, load the end of game scene. When the player is inside the door area and presses the "E" key, the next scene loads. The "E" key activates when the player is inside the door area and disappears when the player leaves the area.In the Start() method, the reference to the CharacterControllerScript component is initialized and the player is not inside the door area at the beginning. Additionally, object e is deactivated.The Update() method runs every frame and checks if the player is inside the door area and presses the "E" key. If so, load the next scene.The OnTriggerStay2D(Collider2D collision) and OnTriggerExit2D(Collider2D collision) methods are triggered when the player enters and exits the door area, respectively. When the player enters the door area, the e item activates, indicating that they can press the "E" key to interact. When the player leaves the door area, item e is deactivated.

- **DoorLevel1Script**: Manages the behavior of a door at level 1 of the game. When the player is inside the door area and presses the "E" key, it loads the next level (level 2), and updates the LastLevel value in PlayerPrefs based on the player's current level.In the Start() method, the reference to the CharacterControllerScript component is initialized and the player is not inside the door area at the beginning. Additionally, object e is deactivated.The Update() method runs every frame and checks if the player is inside the door area and presses the "E" key. If so, load the next level (level 2) and update the LastLevel value in PlayerPrefs.
  The OnTriggerStay2D(Collider2D collision) and OnTriggerExit2D(Collider2D collision) methods are triggered when the player enters and exits the door area, respectively. When the player enters the door area, the e item activates, indicating that they can press the "E" key to interact. When the player leaves the door area, item e is deactivated. The rest of the door scripts do the same as this one but adapted to their level.
- **ChestControllerScript**: controls the behavior of a chest in the game. When the player enters the chest area and presses the "E" key, the chest opens and a coin is displayed. Additionally, the player's coin counter is increased.The Update() method runs every frame and checks if the player

is inside the chest area, has pressed the "E" key, and has not yet obtained the coin from the chest. If these conditions are met, the chest opens, a coin is displayed, and the player's coin counter is incremented.The OnTriggerEnter2D and OnTriggerExit2D methods are triggered when the player enters and exits the chest area. When the player enters the chest area, the e item activates, indicating that they can press the "E" key to interact. When the player leaves the chest area, item e is deactivated.The Open() method triggers the chest opening animation and then calls the ShowCoin() method to display a coin.The ShowCoin() method calculates the position where the coin will be displayed (just above the chest) and then instantiates a coin at that position.

- **CloudDEstroyScript**:Destroys the cloud when it collides with a collider that is off screen
- **CloudGeneratorScript**: It is responsible for generating clouds in the game. In the Update() method, the cloudsTimer timer is decremented each frame. When this timer reaches zero or less, a new cloud is generated.The Y position of the new cloud is chosen randomly between two values. Then, a cloud prefab is randomly selected from the cloudsPrefabs array and instantiated at a predetermined starting position on the left side of the screen.The cloud size is also chosen randomly between two values, and is applied to the scale of the instantiated cloud object. Once a new cloud has been generated, the cloudsTimer timer is reset to schedule the generation of the next cloud. Additionally, on each frame, the MoveClouds() method is called to move all existing clouds to the right. The speed of each cloud is chosen randomly between minCloudSpeed and maxCloudSpeed. This creates a moving effect of the clouds in the scene.
- **CoinScript**: Destroy the coin when the animation has finished
- **IceCrystallControllerScript**: is responsible for managing the behavior of ice crystals in the game. When hit by a bullet, they change color and activate a "hit" state.In the Start() method, the SpriteRenderer component of the glass is initialized and the initial state is set to "not hit".The GetHit() method returns the current state of the crystal, whether it has been hit or not. When an object with the tag "Bullet" collides with the glass, the OnTriggerEnter2D method is activated. In this case, it changes the state of the crystal to "hit", changes its color and destroys the bullet that hit it.

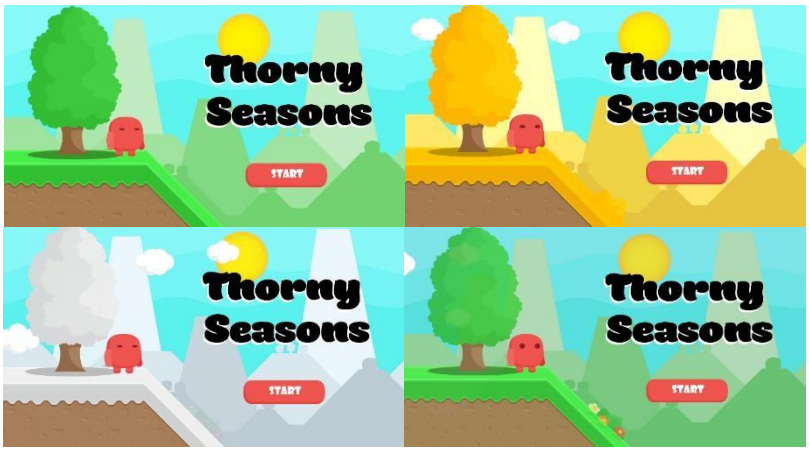
- **IceCubeScript**: Control the behavior of an ice cube in the game. When the character collides with it, the ice cube triggers an animation that simulates its fall.In the Start() method, the ice cube Animator component is initialized and the initial state of the animation is set to not playing.When an object with the tag "Character" collides with the ice cube, the OnTriggerEnter2D method is triggered. In this case, the ice cube animation is activated to simulate its fall and the animation is marked as played.The ResetBool() method is responsible for resetting the animation boolean to its initial state so that it can be played again.
- **LimitsControllerScript**:Control the limits of the game. When the character goes out of bounds and collides with the object this script is associated with, the character's life is reduced and the character is returned to the starting position.In the Start() method, game objects are assigned to the startDoor and characterGO variables using their respective labels.When an object with the tag "Character" collides with the game boundaries, the OnTriggerEnter2D method is triggered. In this case, the DecreaseLives() method of the CharacterControllerScript script associated with the character is called to reduce its lives, and then the character is placed back at the starting gate position.This script ensures that the character cannot leave the boundaries of the game and penalizes his attempt with loss of life.
- **BossControllerScript**: This scripts controls the movement of the boss. In this script there is a simplified way of a state machine. At first the boss is not moving and always looking at the player with the function LookAtPlayer() and when the trigger of the sartCombat is set to true, the boss will start running toward the player. It also starts not being defeated and while it is not defeated the boss and depending on the distance between the player and the boss it will select three types of attacks or it returns to the starting point if it is too far away. The first ones is Jump, that applies a force in the rigid body of the boss and if it lands on top of the player the player will move to the side and will decrease the lives. The second one, the Attack, is where the boss jumps and attacks the player with the mace and it also decrease the lives. And the third one, JumpAttack, is a combination of the two above.

- **WaterControllerScript**: This does the same as the previous script but with water.
- **ParallaxEffectScript**: This scripts controls the parallax effect that makes each layer of the background (such as the sky, the farthest mountains, the nearest mountains and more) to move to different velocities so it make an effect creating an illusion of depth in the 2D scene seen from the player point of view.

## 3. Results

After battling and crying because of the animations, colliders, scripts and triggers the results of our game are the following ones and they are more or less the same as we expected in our objectives, although we were not able to do the transitions between screens.

As explained before to reinforce our idea of the game being inspired in the four seasons, we achieve it by implementing that in an our start screen too with a little animation that changes the season our main character is. In that screen, there's also the button start and the title that we decided to colour it lack since is a neutral color which we thought it could stand out without getting lost in the diferent colors of the seasons as seen in the imatge below.



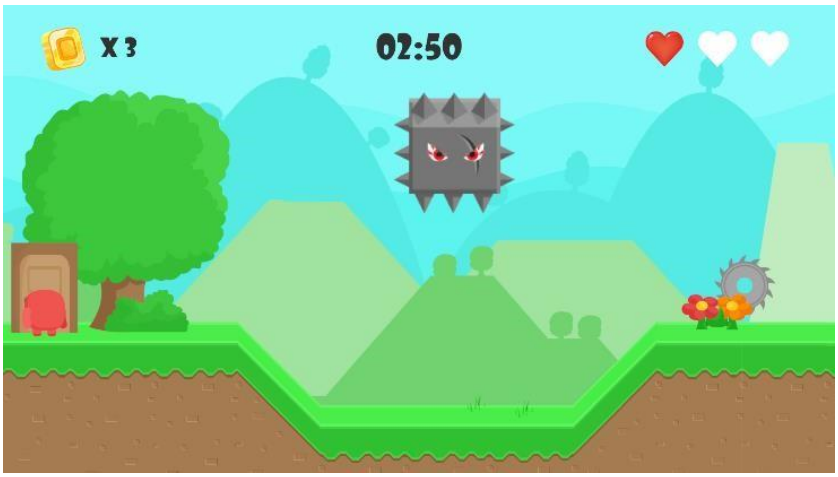

After you click the button start, the game will lead you to the map game level in which you can enter as many times as you want to each level in order to obtain coins if you had unlocked that level. So, at first whatis shown i UI with the number of coins and hearts the player have and all levels are unlocked with the fence excepte the first one as seen in this picture.


But, after you have completed a level the fence of the next level is unlocked and the upper part of the door is tinted as green as a sign that the Player has completed the level.

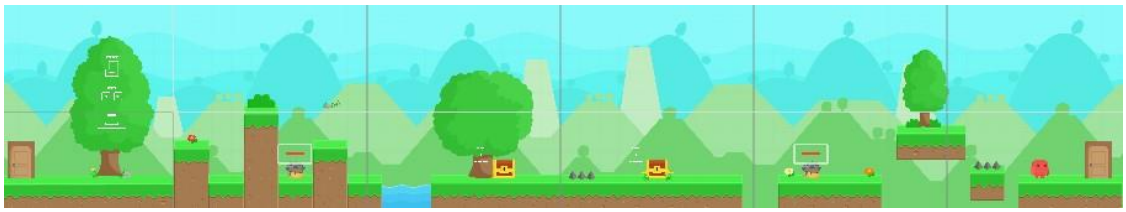

Then, the player in the “Store” screen, in which is always welcome to enter, the player will be able to buy hearts if he/she is low of life units. Ones you enter you will see the seller who might seem a bit scary at first since you can only see the eyes of it and it also will talk to you when you approach him and depending on the life units and coins you might have a diferent conversation with it.


In each level apart from aesthetically being a diferent season, we also implemented a new thing, weather it is a new enemy or a new game mechaninc.

Also, the user interface is the same as the one shown in the map level with a little difference that is that you have the time that has passed since you started.

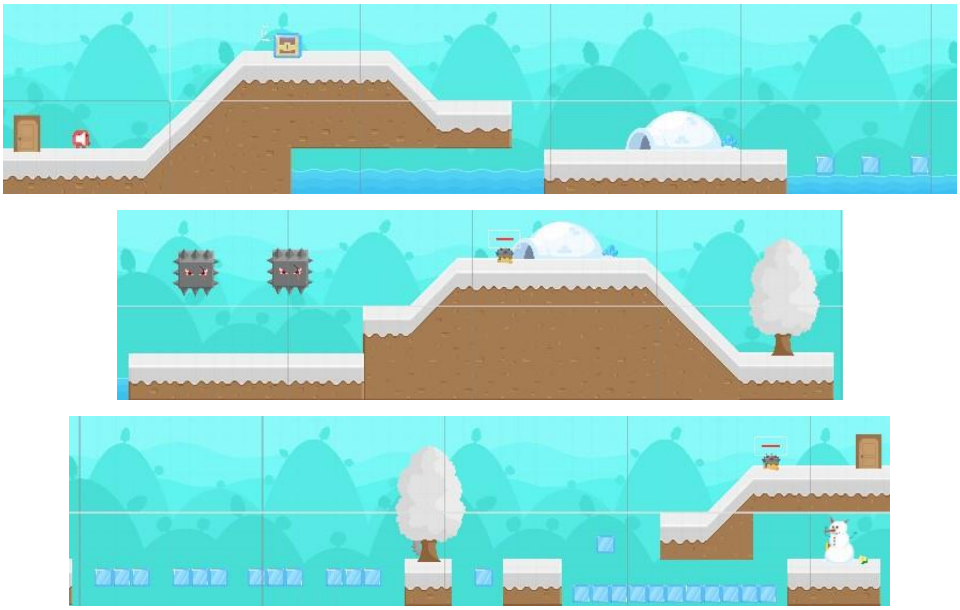



So in the first level, the summer season one, is meant to be a intoductory level where the player will be able to undersand some mechanics of the game, the main enemies and the controls such as walk, jump or shoot. And also he/she will be able to identify the enemies that aret he yellow thingy that has a spikie hat and the spikes.

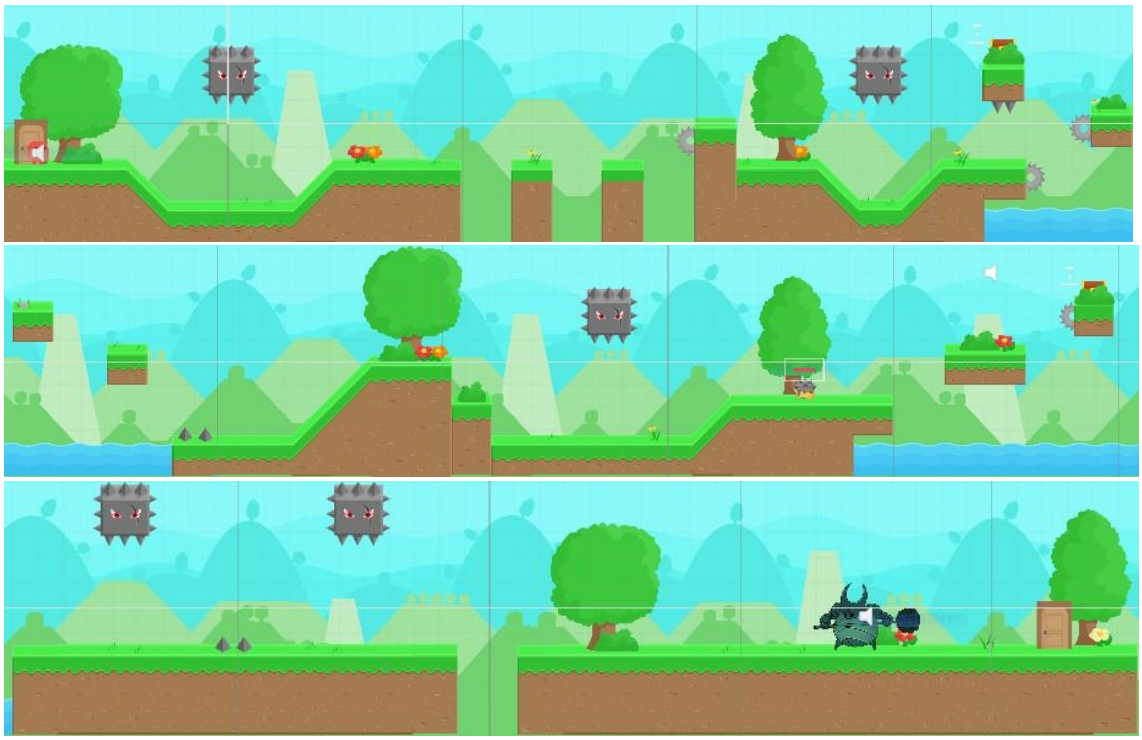


The second level, the autumn one, we level up a little bit and we added a new object that acts as an enemy that is the saw, that as explained befeore it rotates and moves from left to right and from right to left, we also changed the mechanics of the spikes that they might surprise you since they might appear when you least expecti t and we added the maces that they will try to smash you.

The third level, the winter one, when it comes to enemies it doesn’t change much since it only adds a snowman that will not let you Touch his treasuere unless you find the only thing that do not belong in that landscape. In this level basically what we did is to add new mmechanics for instance the slopes, the ice cubes that falls down when you touch it and the ice crystals next to the igloos that you need to shoot them in order for you to discover what they hide.



The forth and last level, the spring one, is a larger level in wich combines all what the player has seen so far and is in this level in wich the player can show how he or she has mastered all the mechanics of the game. Here is also the place where your archienemy that is a cocorach shaped being hides and he will try to defeat you with his three attacks, the first ones is a jump in which he will try to smash you with his butt, the second one that will try to smach you with his hammer and the last one is a combination of the two first attacks described before.



And last but not least we have the game over scene in which it will only appear your score, that is the time that you have compleated the game, if you only have defeated the boss, if not we are sorry but the player will not be classifed for the top three. It is also displayed a little animation where the character tries to defeat the enemy but the enemy dogges his attacks. There are also two buttons that allows to quit the app or to restart the game from the beggining.

## 4. Conclusions

In this project we have leared how to create and develop a 2d platform game from scratch with Unity engine with the goal to put in practice and reflect all what we have leared so far during the semester.

Therefore, thanks to the API, videos and the lessons we had we achieved to obtain a 2d platform game inspired in the four seasons and meet all the requirements and objectives we were given and thought. Those objectives where a long leveled game with enemies and different types of movement of the player, control of lives and we also added the control of coins, a game start screen, a game over screen with the top three ranking in which we decided we would sort it by the time the player has completed the game and the audios that strengthen the player perception of what is happening or what he or she is doing.

Moreover, what we have leared from this project is that sometimes you need to think beyond what we are used to do in order for us to obtain what we desire since we are were not used to program and desig this kind of way, in which you really need to think wisely of how all the sprites, colliders and animations would interact each other. And also that most of the times if we think as the real world works it also helps to know what to do as there are some maths and physics applied on everything so everything might interfere with other things and mess up what you had already done.

However, due to the time limitations we achieved almost everything we wanted. But, we would like to have a better boss with different types of attacks and also have implemented the transitions since we thought they were a cool way to change between screens. But due to we have left that part for the last thing to do we had no much time to develop a cooler boss and integrate a multiplayer feature. So in the next development of a game, we would like to have a better understanding of the difficulties of each part of the game development since this might help us to improve our time management and obtain better results.

In conclusion, we have seen that Unity is a very powerful and useful engine to develop a game that has many both tools and methods (when it comes to

programming) that makes your work easier. We have accomplished all the goals we have set at the beginning, and although we are satisfied with what we have obtained we would like to have had a better understanding of the different difficulties of each stage of the game development so we could have fixed some bugs we have and make a better experience. Finally, the other thing we have leared and that we should apply in everything in life is to think before doing something.