GMD - Lenka Orincakova (293085)

1. Background story

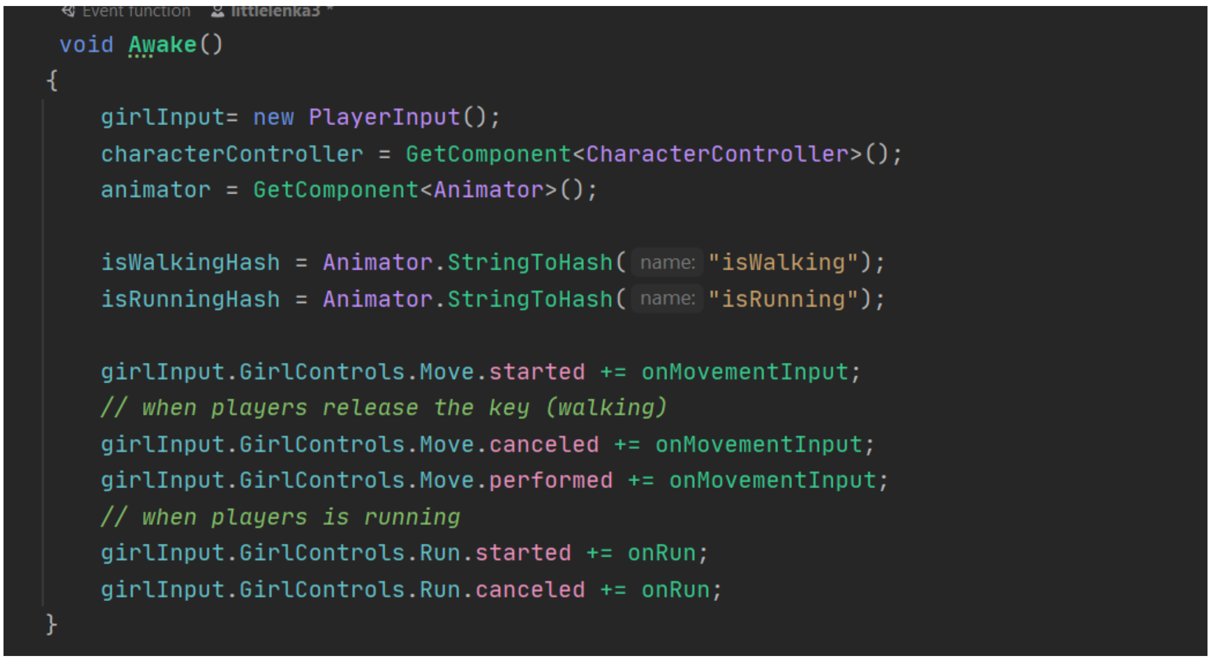
From the beginning of the project, our idea was to create a game that we like and that it has a story itself. We started with creating the first green scene. We imported assets that contained different prefabs from nature and other assets that were suitable to the design. The other scene Sahara was created to expand the project and also as a place to finish the game. The idea for the game was to survive as long as you can by collecting keys, but after implementing some features, we decided to keep the goal to collect 10 flowers that are randomly distributed. There is also a minimap which represents the movement of the character and it displays the location of the flowers so it's easier to find them. With this project, we wanted to show our skills by implementing the topics that we learnt and went through.

2. Requirements:

Input and Vectors

To make a character move, it needs to listen to inputs from a user. The input usually is for movement or various actions, although this game doesn't have actions based on the input. For controlling the character, it's needed to add a component PlayerInput, which will create an ActionMap with a controller. In the action map, all of the possible inputs are set and binded. To move around with character, the WASD keys are being pressed. After having some issues with rotation and navigation of the character, the keys were changed and adjusted. The Move action was chosen to have a control type Vector 2. The input for making the character run, left shift was chosen.

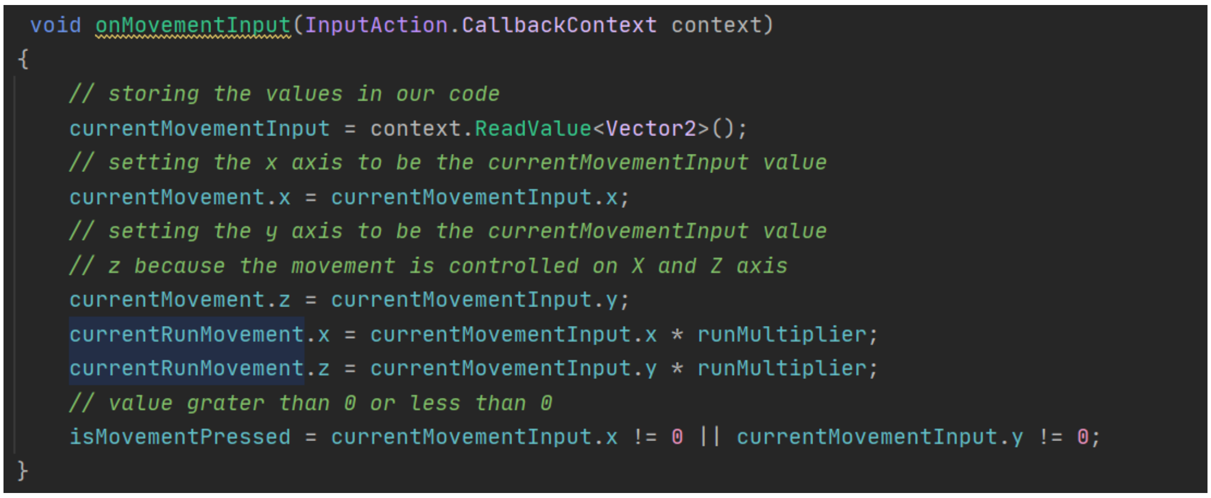
In code, the PlayerInput variable has to be initialized as a reference to access the actions in Unity. To listen to input from a user, we used callbacks which are executed when the event is happening, such as clicking the W for movement and then the function is invoked.


```
  Event function   littlelenka3 *
  void Awake()
  {
      girlInput= new PlayerInput();
      characterController = GetComponent<CharacterController>();
      animator = GetComponent<Animator>();

      isWalkingHash = Animator.StringToHash( name: "isWalking");
      isRunningHash = Animator.StringToHash( name: "isRunning");

      girlInput.GirlControls.Move.started += onMovementInput;
      // when players release the key (walking)
      girlInput.GirlControls.Move.canceled += onMovementInput;
      girlInput.GirlControls.Move.performed += onMovementInput;
      // when players is running
      girlInput.GirlControls.Run.started += onRun;
      girlInput.GirlControls.Run.canceled += onRun;
  }
```

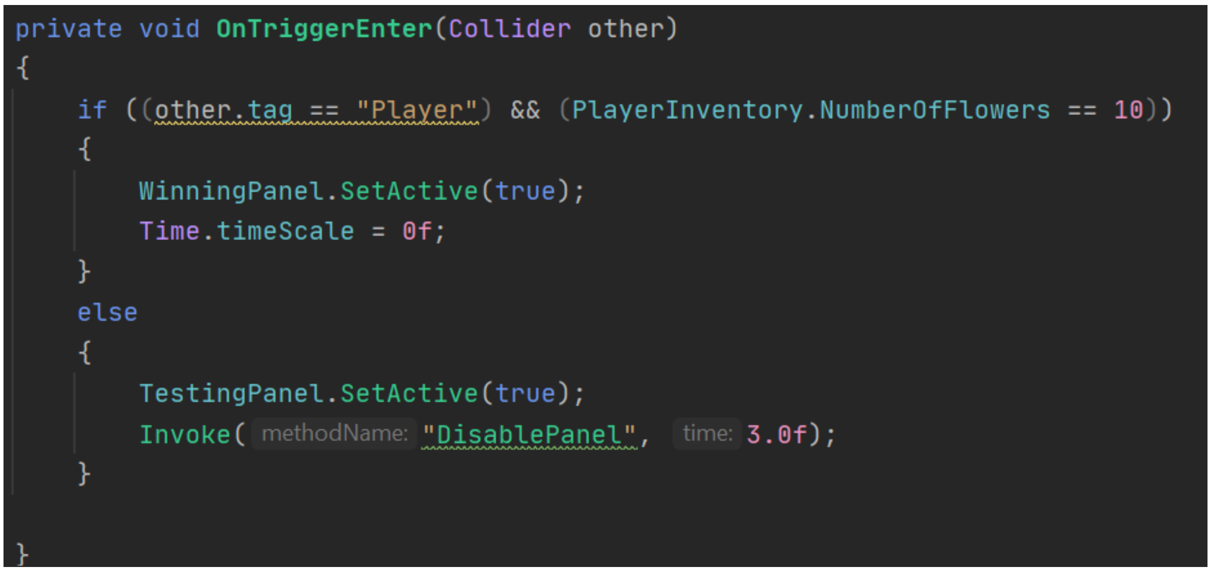
girlInput.GirlControls.Move.started is written to access the action map and move function in it. This will enable listening to the player whenever he makes input.


```
 void onMovementInput(InputAction.CallbackContext context)
{
    // storing the values in our code
    currentMovementInput = context.ReadValue<Vector2>();
    // setting the x axis to be the currentMovementInput value
    currentMovement.x = currentMovementInput.x;
    // setting the y axis to be the currentMovementInput value
    // z because the movement is controlled on X and Z axis
    currentMovement.z = currentMovementInput.y;
    currentRunMovement.x = currentMovementInput.x * runMultiplier;
    currentRunMovement.z = currentMovementInput.y * runMultiplier;
    // value grater than 0 or less than 0
    isMovementPressed = currentMovementInput.x != 0 || currentMovementInput.y != 0;
}
```

+= onMovementInput method gives access to input whenever the .started callback is invoked. It checks for which key was pressed. Next lines are reading the vector's value according to pressed keys (WASD).

Physics

One of my tasks was to create an end position for the character to finish the game, which is done with the particle system and image that is described below. In the circle, I put a cube which could start a trigger whenever the character touches it. In Order to perform the collision, the Girl has a Rigidbody component which reacts to collision with another Collider component added to the second object. By setting the tag “Player” to a Girl character, I made sure that the character will start the trigger and not anything else. When the collision is present, the trigger is called and based on the condition, the player either wins or loses.



```
private void OnTriggerEnter(Collider other)
{
    if ((other.tag == "Player") && (PlayerInventory.NumberOfFlowers == 10))
    {
        WinningPanel.SetActive(true);
        Time.timeScale = 0f;
    }
    else
    {
        TestingPanel.SetActive(true);
        Invoke( methodName: "DisablePanel",  time: 3.0f);
    }

}
```

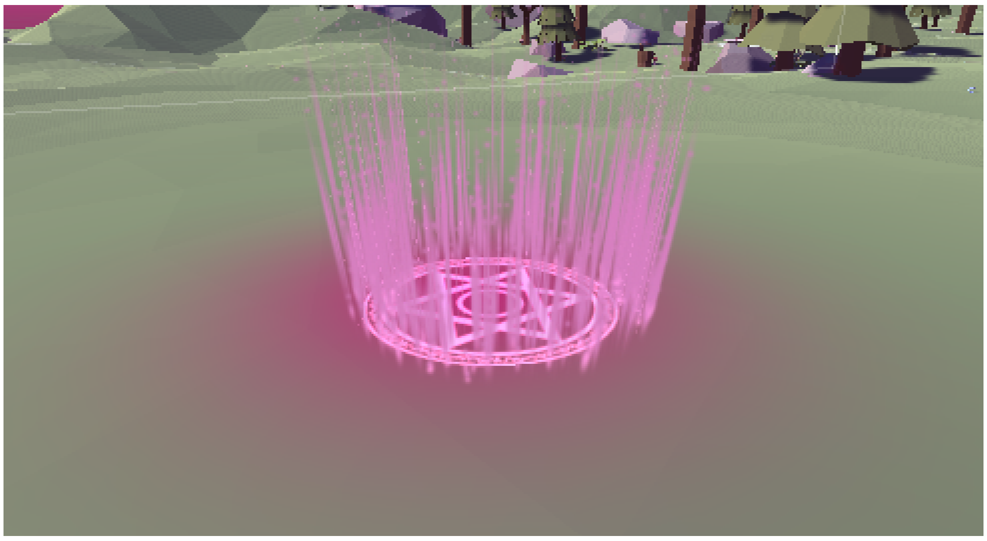
Another collision and trigger that was present is while picking the keys (flowers) which is also triggered when the Girl touches them. This task was made by my groupmate.

Graphics and Audio

To have a better view of the actions of the character, we implemented a third-person camera.

Particle system is also included in the Graphics section and the final circle position is made from the particle system. They are displayed and moved in huge numbers such as fireworks or fire. In this case, it's indicating that this circle is the final circle for winning or losing the game.
The particles were used in flowers to make them more visible.

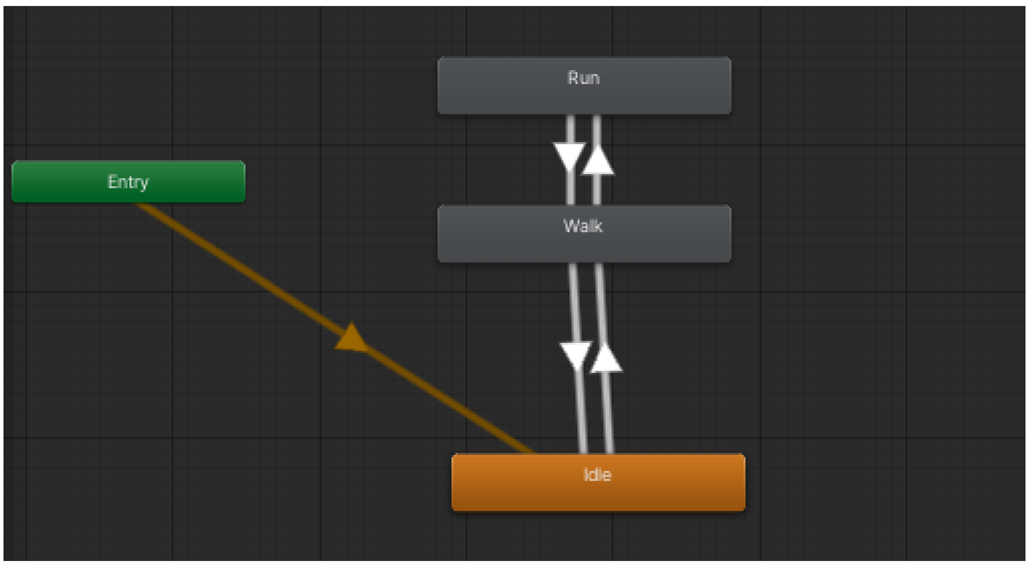
Lot of objects that are implemented in the game are mostly meshes such as Girl character. There is material from asset that was used to create the real look.



I also added an audio clip to a game which is made by adding an Audio Source component. In code, first I checked if there is some background music and if there is no audio, the game will be played with the added audio clip. Audio is by default destroyed and played multiple times and to prevent this, I added a DontDestroyOnLoad method for not stopping the audio when loading a new scene. Since the unity doesn't recognize the difference between sound and audio, the sound for picking the flowers was created in the same way by adding an Audio Source component with the clip inside. I implemented a settings option to mute the Audio and Sound, and it will save the preference until unmuted again.

Animation

Animations in Unity are often used and can be created from scratch. In this project, I imported an asset with Girl Character that also included the already made animations as a prefab. Binding the animations to a game object is done in Animator Component where I made a bind from Idle state to a Walking and Running state as shown in the picture below.



Girl Game Object contains an Animation Controller to which I added the animation files that are needed to communicate between the GameObject and Animations. In implementation, the method *animationHandler()* has booleans that are checking if the state of the character is in Running or Walking state and based on input, if the Run action is pressed (Left Shift), the *isWalking* state is false and otherwise. The T-Pose and Idle breathing look like animation is also added.

User interface

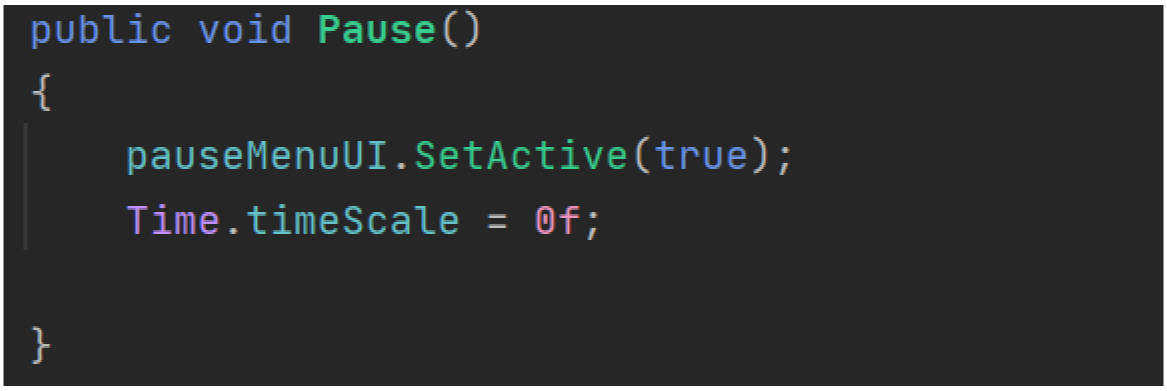
Before creating the UI itself, we decided and agreed on what are the needed buttons for the game, since the user should be able to navigate through different scenes and eventually quit or pause the game. First, I created a Play button to navigate to the next slides and give an overview of the game story. We also assumed that the rules are important to mention as well as the possibility to mute or unmute the game. The whole UI is made as a next scene, apart from the main game scene. To load both scenes in one play, I had to add the UI scene to Build Settings, which is adding an index (unique id) to each scene. This game starts with a UI scene, that's why the index is 0.

User interface in this game has a lot of images, such as background image and also background images for button columns to add a background color on them.

Each button (e.g *Play*) has a Button component which contains the action that should happen onClick. The navigations are done such as the Menu we are on is disabled and another one is enabled to be seen. The design is rather detailed, since I also played with different colors when the text is highlighted or pressed.

Whenever the Canvas component is added, the Event System is automatically added as well. Without the Event System, the buttons are not working since it can not register and detect an input from a user such as clicking.

If the user wants to pause the game, I added an Pause icon which will make the Pause Menu visible and the user has two options. During this pop up, the time in the game is stopped.



```
public void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;

}
```

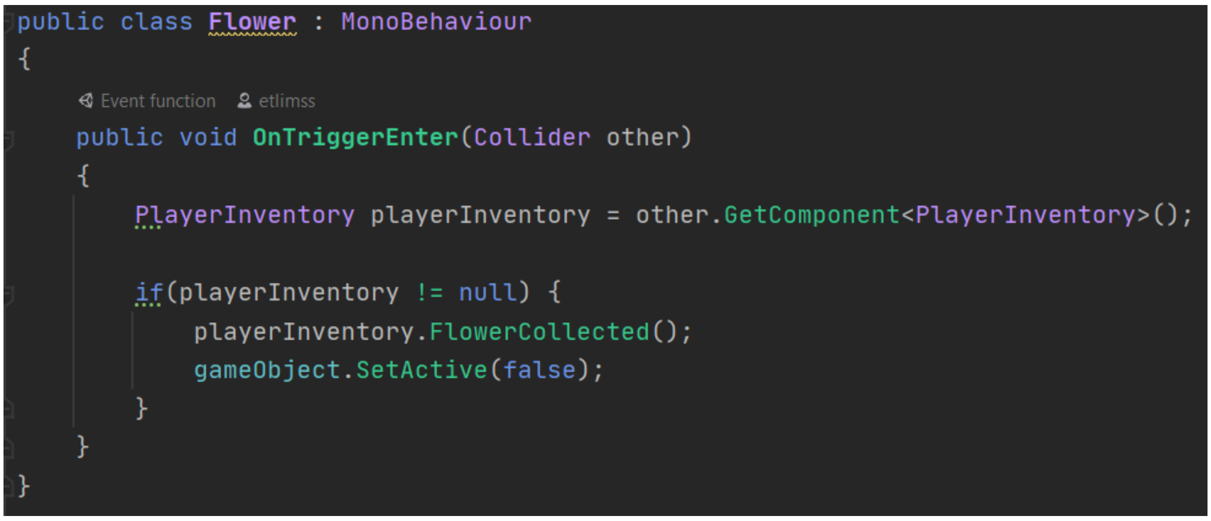
If the user chooses to continue the game, the game is resumed and the time is again set to 1f. By clicking on the Quit button, the function Home is called and the scene with selected Id of the front UI is loaded again. Later during the game, I created a Win UI and Game Over UI which will be visible when the user collects/doesn't collect flowers and enters a destination.

Intermediate Scripting

While setting a button to turn on/off the audio and sound in the game, I used *[SerializeField]* attribute to make sure that the icon variables are private but also accessible in the editor. Based on the input of the user, the method *UpdateButtonIcon* is called which is checking if the audio and sound is muted or to be mute. To save the preference of the player, I used PlayerPrefs to store and access them. Since PlayerPrefs doesn't store boolean types, the conversion had to be made. Instead, it's setting 0 and 1 accordingly. Events - flowers

Game Architecture

It is important to keep the code clean, flexible and easy to work with. SOLID principles are set to help the programmers with structured code. Most of the scripts in this game follow the SOLID principle as shown below.

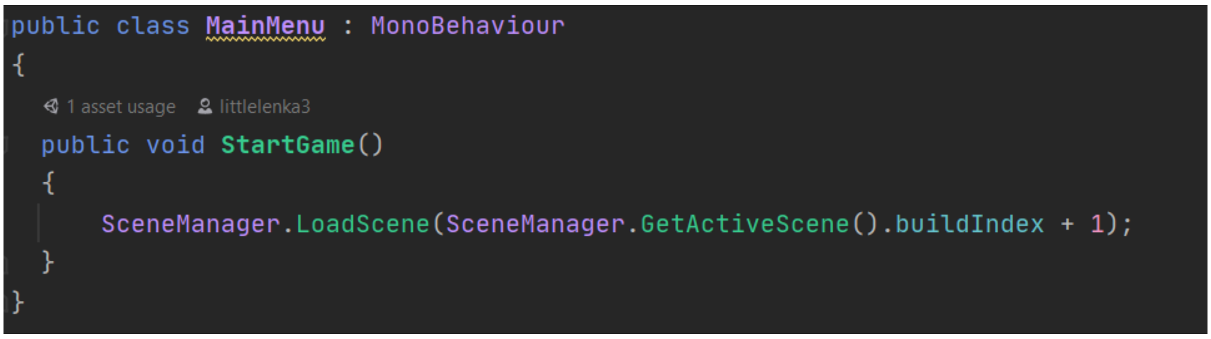

```
public class Flower : MonoBehaviour
{
    Event function   etlimss
    public void OnTriggerEnter(Collider other)
    {
        PlayerInventory playerInventory = other.GetComponent<PlayerInventory>();

        if(playerInventory != null) {
            playerInventory.FlowerCollected();
            gameObject.SetActive(false);
        }
    }
}
```

The Flower is created in a separate class and contains the logic for collision between a Girl and flower. The script PlayerInventory on the other hand keeps track of the number of collected flowers.

To create a flexible and easy to maintain code, the controllers or managers are added as a script which is responsible for a single area, such as Audio. In AudioManager script, only functions that are controlling the audio are implemented there, in this case it's muting the audio and sound, saving them and accessing the preferences of the player. It's also responsible for disabling the icons for audio being muted.

When we want to load multiple scenes in one game, we have to add the scenes to a build settings and load them when needed. In the code, the scene with index 0 is MenuFront which is the front UI. OnButtonClick we are starting a game with loading a scene with index 1 as seen below.

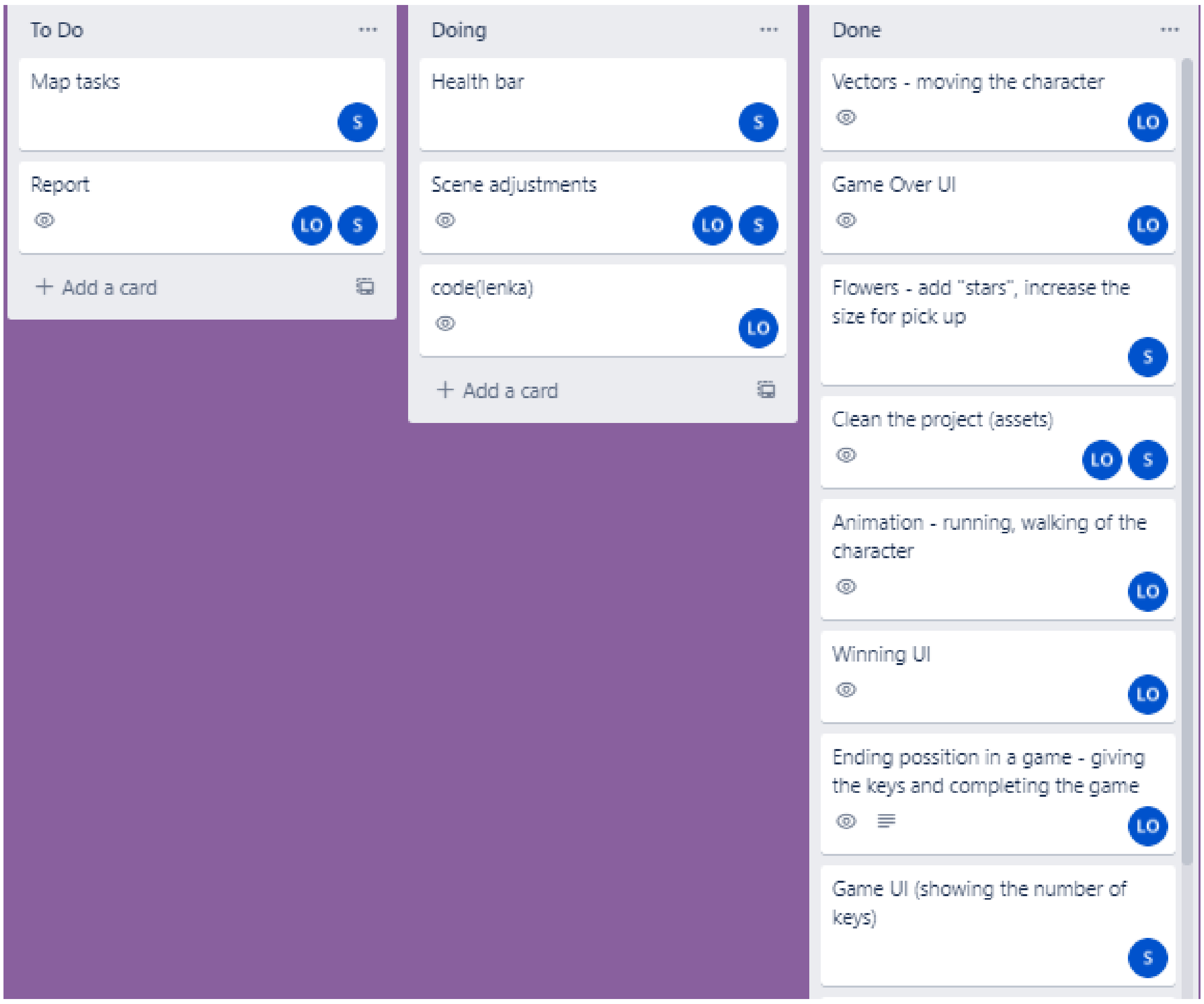

```
public class MainMenu : MonoBehaviour
{
    1 asset usage   littlelenka3
    public void StartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

Another example for having a manager is also Animator which was described earlier. Animator is responsible for connecting the states and making transitions for them as we did.

Conclusion

In the beginning, we struggled with a lot of features and topics since we haven't worked with Unity before, but after some exercises and time, we found a way on how to do different things. Personally, I watched a lot of videos either from Unity or tutorials on different functionalities and after understanding them, I tried to create it on my own. Some functions and code is inspired by tutorials, such as Animation, because some topics were challenging to implement. After having the functions working and going through the code again, I understand how everything works and we can easily fix the errors or small details when they occur.

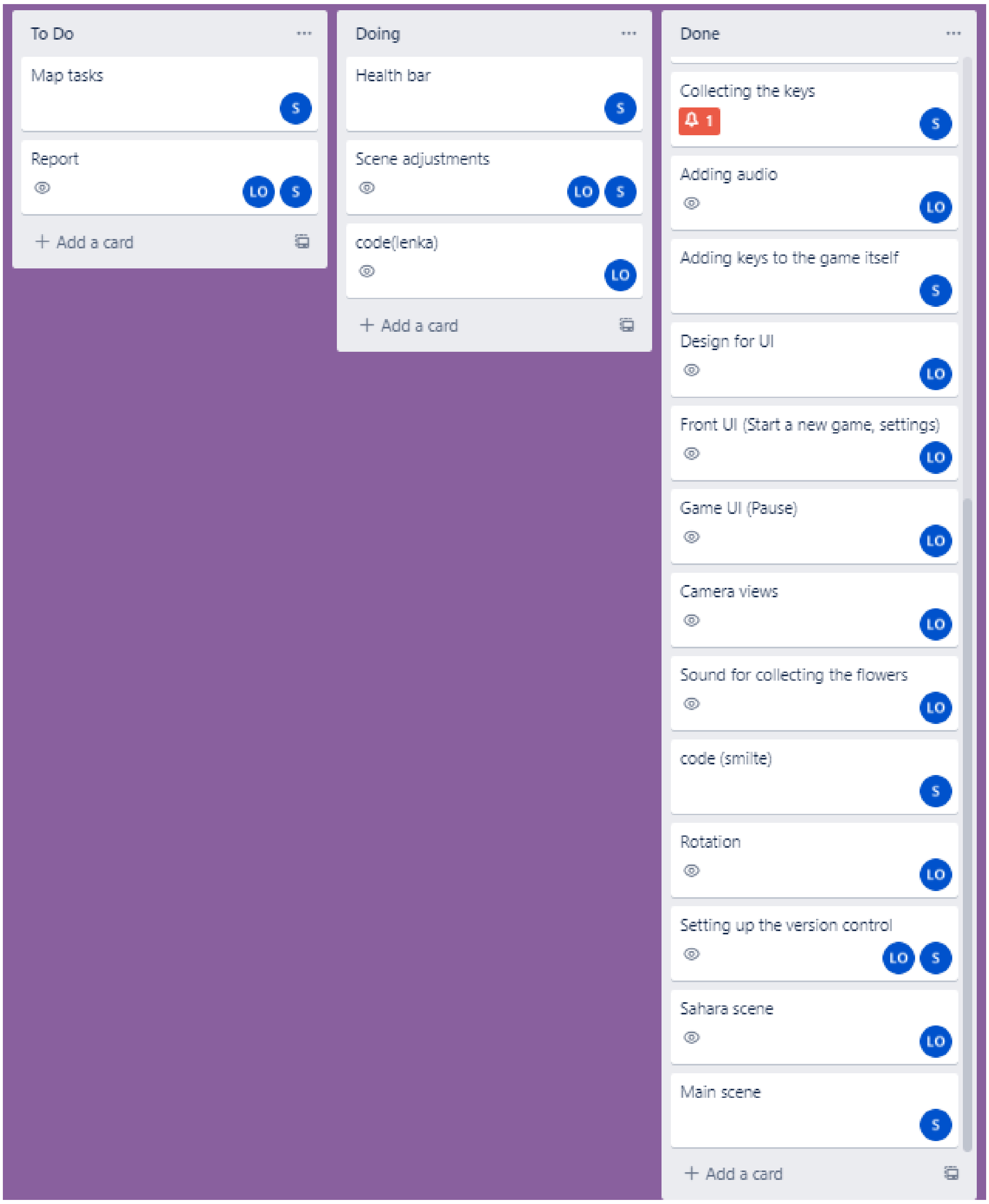
We built this game on a simple basis and the focus was put on scalability of the game. Our goal was to fulfill the requirements and have a working game and then slowly add some more features to it. The game could have been more complicated and complex with time, but we chose to focus on fully understanding what we created. Adding some physics such as shooting would help the quality of the game as well as adding more players (enemies). More animations could have been added, such as Jumping, but after some issues with implementation, I decided to skip jumping animation and work on different tasks. The tasks are documented in the Trello application which can be seen below.



To Do
- Map tasks — S
- Report — LO, S
- + Add a card

Doing
- Health bar — S
- Scene adjustments — LO, S
- code(lenka) — LO
- + Add a card

Done
- Vectors - moving the character — LO
- Game Over UI — LO
- Flowers - add "stars", increase the size for pick up — S
- Clean the project (assets) — LO, S
- Animation - running, walking of the character — LO
- Winning UI — LO
- Ending possition in a game - giving the keys and completing the game — LO
- Game UI (showing the number of keys) — S



To Do ···

Map tasks S

Report LO S

+ Add a card

Doing ···

Health bar S

Scene adjustments LO S

code(lenka) LO

+ Add a card

Done ···

Collecting the keys 1 S

Adding audio LO

Adding keys to the game itself S

Design for UI LO

Front UI (Start a new game, settings) LO

Game UI (Pause) LO

Camera views LO

Sound for collecting the flowers LO

code (smilte) S

Rotation LO

Setting up the version control LO S

Sahara scene LO

Main scene S

+ Add a card

I am also aware that some code written by me is not following the SOLID principle, because I didn't take it into account and changing the code at the end wouldn't be the best step.