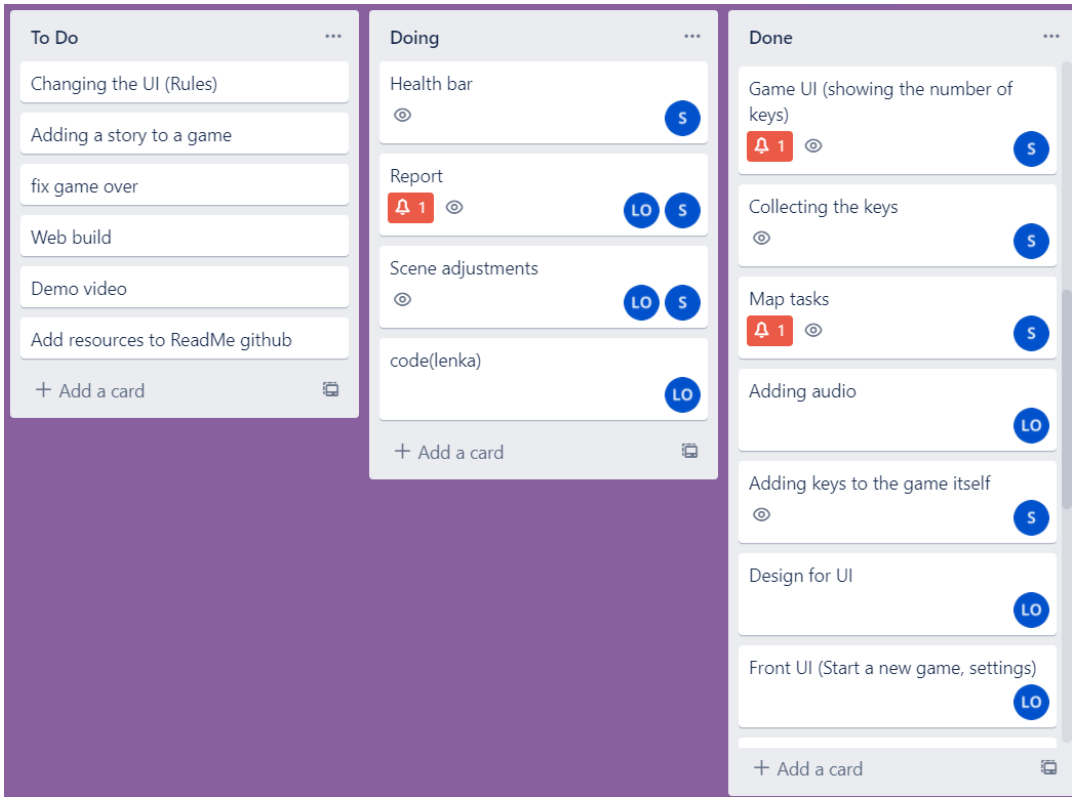
GMD – Smilte Pavilionyte (285056)

Story:

The project started when we were forming the group. We decided to work together because we had similar ideas for the project. The idea was to make a treasure game where the player can explore the scene using the map. We got inspired by the games - *Valheim, Don't starve together*, and other survival games. The concept was to make the player collect "keys" which in our case are flowers to complete/survive the game. We started with building the scenery with imported assets. One main asset was used to build the scene with different trees. Our ground is made of two parts - one part is the green forest and the Sahara desert separated by the bridge. The player ends the game in the Sahara when collecting the last flowers. Important to mention that minimap is considered a Treasure map because it helps players to find and collect all the flowers.

To plan our time and tasks we used trello where we divided everything equally and it we were able to keep track of our progress.



Input & Vectors (e.g. input systems, transforms)

To control the character, we needed to add a component "PlayerInput" which creates an "ActionMap" with a controller. Our character moves with WASD keys that are being pressed. The move action is chosen to have a control type "Vector2". For the character to run, we chose the "shift" key.

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

    isWalkingHash = Animator.StringToHash("isWalking");
    isRunningHash = Animator.StringToHash("isRunning");
    girlInput.GirlControls.Move.started += onMovementInput;
    girlInput.GirlControls.Move.canceled += onMovementInput;
    girlInput.GirlControls.Move.performed += onMovementInput;
    girlInput.GirlControls.Run.started += onRun;
    girlInput.GirlControls.Run.canceled += onRun;
}
```

“girlInput.GirlControls.Move.started” is accessing the action map with the move function in it. That will let listen to the character when it makes an input. “+= onMovementInput” gives access to the input when the “.started” callback is invoked.

```
void onMovementInput(InputAction.CallbackContext context)
{
    currentMovementInput = context.ReadValue<Vector2>();
    currentMovement.x = currentMovementInput.x;
    currentMovement.z = currentMovementInput.y;
    currentRunMovement.x = currentMovementInput.x * runMultiplier;
    currentRunMovement.z = currentMovementInput.y * runMultiplier;
    isMovementPressed = currentMovementInput.x != 0 || currentMovementInput.y != 0;

}
```

Physics (e.g. rigid bodies, colliders, triggers)

The flowers were used from one of the assets we downloaded. All the flowers were put in one parent game object called “Collectables”. Making the player collect the flowers was one of my tasks.

To pick the flower, the Girl character needs to detect collision with the flower, so to do that, we added a box collider to all the flowers. Because of that player will be collecting flowers during a collision.

To let the player detect collisions without the flower being an obstacle, I had to select the “Is Trigger” check box in Unity.

To handle the collection of items the “PlayerInventory” script was made to keep track of how many flowers it was collected and assigned to the player game object.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class PlayerInventory : MonoBehaviour
{
    public int NumberOfFlowers { get; private set; }
    [SerializeField] private AudioSource CollectingSound;

    public UnityEvent<PlayerInventory> OnFlowerCollected;

    public void FlowerCollected()
    {
        CollectingSound.Play();
        NumberOfFlowers++;
        OnFlowerCollected.Invoke(this);
    }
}
```

In the script above I'm creating a new public integer property for the number of flowers collected which is set to have a private setter. By that, any script could read the value, but only the "PlayerInventory" script can set the value.
The public method "FlowerCollected" will increment the number of flowers which was collected during the game.
Also, the private field "AudioSource" contains collectible sound, so whenever a flower is collected "CollectingSound.Play()" will be activated.

Another script called "Flower" has the logic for when our Girl player collides (touches) with the flower.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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

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

The script is made to detect collision between the flower and the Girl character. The public method "OnTriggerEnter" and this is called when the player collides with the flower. First is checking the collision with the character by getting the "PlayerInventory" component from the object that has been collided with. In the if statement used the "PlayerInventory" component to call the "FlowerCollected" method. "GameObject.SetActive" is set to false so collected/collided flowers are not pickable/shown in the game.

If the character collides with a flower it will increment the number of flowers in the inventory and then the flowers displayed in the game will not be active.

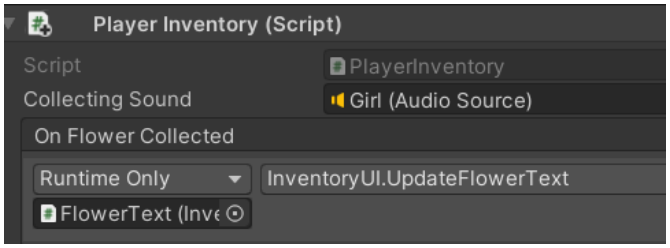
To display the number of collected flowers the script “InventoryUI” was made which is connected to the text called “FlowerText”.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class InventoryUI : MonoBehaviour
{
    private TextMeshProUGUI flowerText;
    // Start is called before the first frame update
    void Start()
    {
        flowerText = GetComponent<TextMeshProUGUI>();
    }

    // Update is called once per frame
    public void UpdateFlowerText(PlayerInventory playerInventory)
    {
        flowerText.text = playerInventory.NumberOfFlowers.ToString();
    }
}
```

The private field is added using TMPro for the flower text (number of the flowers). In the “Start” method, I get the “flowerText” component assigned to the text field. A public method “UpdateFlowerText” for updating the flower number which is initially set to 0. The “Playerinventory” is set as a parameter and in the method, the text is set to the number of flowers in the player's inventory. This method needs to be called whenever a flower is collected by the player and to do so, I choose Unity Events. To trigger a Unity Event it needs something to happen as in our case when the flower is collected.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class PlayerInventory : MonoBehaviour
{
    public int NumberOfFlowers { get; private set; }
    [SerializeField] private AudioSource CollectingSound;

    public UnityEvent<PlayerInventory> OnFlowerCollected;

    public void FlowerCollected()
    {
        CollectingSound.Play();
        NumberOfFlowers++;
        OnFlowerCollected.Invoke(this);
    }
}
```

The event is fired in the “PlayerInventory” script. I specified that the event will take an argument of the type “PlayerInventory” and called “OnFlowerCollected”. In the “FlowerCollected” the event is invoked, it will go through “this” to pass the “PlayerInventory” thought to the subscribers.
After that in Unity, I had to add a subscriber to the “OnFlowerPicked” event and put the flower text (from the canvas).



It is important to say that this code follows SOLID principal by distributing methods in separate classes. It was easier to maintain and find used methods.

Graphics & Audio (e.g. models, shaders, audio clips)

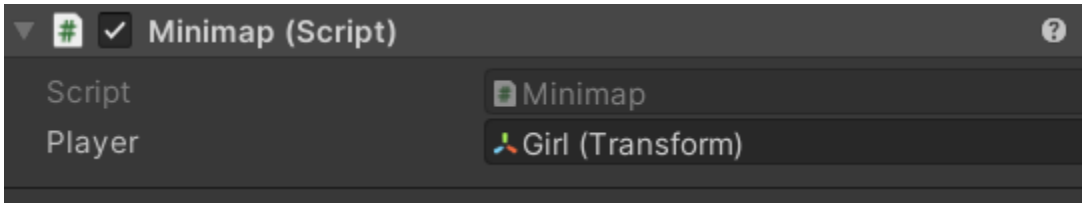
We used a lot of meshes and prefabs - even making our own out of the assets or making sprite images from the pictures.

Audio is used for the game in general as the main sound and also whenever the Girl character picks the flower.

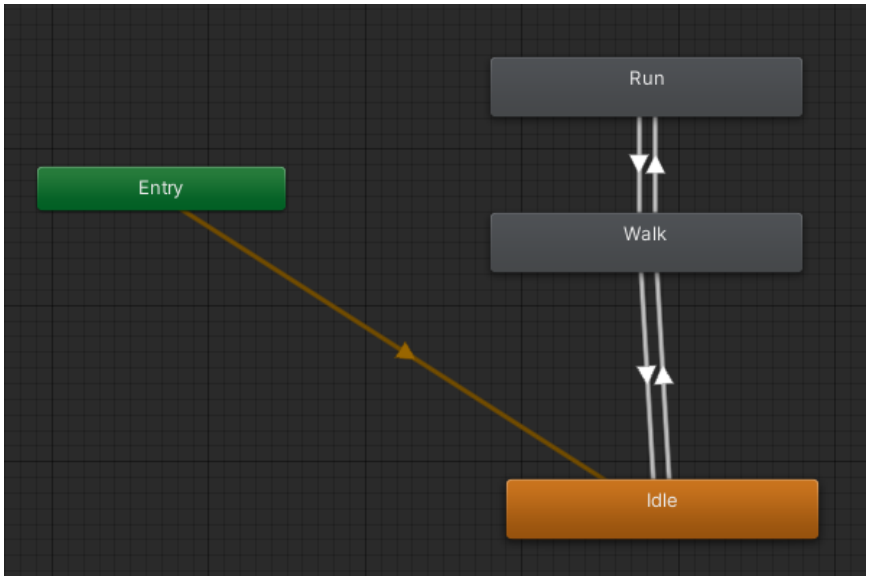
I made a minimap which acts as Treasure Map in the game. It has a re-created camera from the main (character camera) which shows where the player is in 90 degrees form.

```
public class Minimap : MonoBehaviour
{
    public Transform player;

    void LateUpdate () {
        Vector3 newPosition = player.position;
        newPosition.y = transform.position.y;
        transform.position = newPosition;

        transform.rotation = Quaternion.Euler(90f, player.eulerAngles.y, 0f);
    }
}
```

To display the player's “live” location using the camera and the view, I had to make a new camera for the player which has the same fields as the player’s camera. To allow camera rotation from the player’s perspective which is displayed on the minimap the above script was written and attached to the minimap camera.



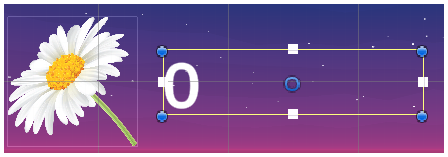
Animation (e.g. animators, animations)



The animation was with our Girl character who had running, walking, and idle position. To make the girl move and have a smooth transaction between the states, we had to connect it. In the implementation, the method “animationHandler()” have booleans that check if the state of the player changed based on running and walking input. If the player enters a running state (with the “shift” key), the state “isWalking” will be false.

User Interface (e.g. menus)

To display the number of picked flowers, I made canvas in unity which is in the “Collectables” game object with all the flowers. Canvas consists of two things - image (used sprite) and text.



To display the number of flowers I used Events (described in the physics chapter).

Each button in our made canvas (for example main game screen has a pause button) which has an event system. In the example below it’s a snippet for the pausing and resuming the game. If the user decides to pause the game, the popup will show up immediately stating that the game is paused and “pauseMenuUI” is set to active. If user decides to continue the game, it will resumed and the time is set to 1f.

```
public class PauseMenu : MonoBehaviour
{
    [SerializeField] GameObject pauseMenuUI;

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;

    }

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

    }
```

After clicking on pause, user also has an option of going to the home page by clicking the quit button. In that case, the method “Home” is called and the scene with selected “sceneID” of the home page is loaded.

```
public void Home(int sceneID)
{

    Time.timeScale = 1f;
    SceneManager.LoadScene(sceneID);
}
```

We also have “Game over” and “Win” user interface which is active when the player collects all the flowers (wins) or looses when enters the end point without 10 flowers and has to re-do the game.

Intermediate Scripting (e.g. coroutines, events, optimization)

Since the event used in collecting the flowers was described in the physics chapter, it is important to mention that we had another event for the UI buttons. The buttons would not work otherwise because they cannot detect an input from a user which clicks on the buttons.

As well, having a [SerializedField] made attributes private but accessible in the class. For example we used it for the collecting flowers sound and pause button, so it can be accessed during the certain actions. So based on the user input (click) the method “UpdateButtonIcon” is called which checks if the game audio is muted or not.

```
private void UpdateButtonIcon()
{
    if (muted == false)
    {
        soundOnIcon.enabled = true;
        soundOffIcon.enabled = false;
    }
    else
    {
        soundOnIcon.enabled = false;
        soundOffIcon.enabled = true;
    }
}
```

To safe the input (muted/not muted), we user “PlayerPrefs” to store and access it.

```
private void Save()
{
    PlayerPrefs.SetInt("muted", muted ? 1 : 0);
}
```

Because “PlayerPrefs” don’t store boolean types, it’s setting 0 and 1 accordingly.

Game Architecture (e.g. game managers, SOLID principles)

The Collecting flowers scripts described in the physics chapter follows SOLID principal by having methods distributed in different classes. We have “PlayerInventory” which is connected to the player, “InventoryUI” is connected with player inventory to display number of flowers and “Flower” class has “PlayerInventory” which “tells” how many flowers are collected.

We used scene manager for two scenes and to load it in one game, we had to add the scenes to build settings and load it when needed. Our main menu scene is set to 0 and game scene is set to 1. In the example below it shows then the player clicks on start game the method “StartGame” will load our game scene.

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

As well separating “Audio” and “AudioManager” where in audio we set background music and in the manager we set the settings such as saving players preferences and such.

Conclusion

For the project's future, there is always room to improve. We would like to add other players like enemies fighting with the player a health bar which indeed we tried to implement but failed because we didn’t have enough experience and at first our main focus was to understand every single step we made. We started with simple functionality and when improving over time. As well, in the beginning we were wasting our time and working days on difficult functionalities which in the end did not work and started to brake what we already had. But overall, the game with the functionality we chose it might look very basic but our goal was to make functioning game and learn with the process. The game could have more things and players but I am confident about the knowledge I gain during the development process, the amount of articles read, videos watched and extra exercises made paid off.