Swing To Win

04.06.2022

Sevde Sarıkaya - 2017400081

Şule Erkul - 2017400051

CMPE485 - Term Project

Overview

Swing to win is a 2D-platformer game where a player tries to beat the levels by avoiding some obstacles and lavas.

Currently, there are only 3 levels.

To start the game: open scene with the name “First0”

Game Mechanics

- Player uses ‘a’ button to move left, ‘d’ button to move right, ‘w’ button to jump.

- There are 2 types of obstacles. One is a fence and the other is the lava. If the player touches or hits one of them, it fails the level. A level panel will be open. The user will be asked to try again.

- If the user doesn’t hit the obstacles and arrive at the green flag. The level finishes successfully. A level panel will be open. The player will be asked to play the next level.

How to avoid obstacles:

- Use ropes! If the user collides with the rope object which is swinging back and forth constantly, s/he will be hanging on it and start swinging with the rope. To release yourself from the rope, hit the ‘space’ button and land to a safe surface. Be careful not to land on lava!

The link of the project : https://github.com/schroscatt/Game-Term-Project

Software Design:

The player has idle, walking, jumping and swinging states. Therefore, we have used the FSM approach to implement our game.

Possible state changes:

- From Idle: to Swing, to Walk and to Jump states according to the current inputs.
- From Jump: to Idle and to Swing states.
- From Swing: to Idle state
- From Walking: to Swing, to Idle and to Jumping states.

We also have some controllers:

- SceneController: To load the scenes according to current level and state. For example, if the user fails, the fail level panel will be opened by the controller.

- AudioController: To play the music accordingly. During gameplay, it plays the defined music. If the user fails, it plays sad music. If the user wins, it plays happy music.

- RopeController: It handles the motion of the ropes by using a coroutine. Ropes constantly swing back and forth with a maximum of 60 degrees.

While coding the FSM, we studied TA Bartu's code in the ICP #3 project. It had clear organizational structure. In our project, after some point, getting input, logic updates and physics updates were all over the place, without logical and consistent structure. Using the ICP#3 example, we understood where to do the updates. We also followed tutorials of youtubers "Coding in Flow" (https://www.youtube.com/watch?v=Uv5tfMSKlnU), "Table Flip Games" (https://www.youtube.com/watch?v=21yDDUKCQOI&list=PL8lV_joQZ5sczN_xHOEXEmfSIt3gYr1Rh) for the state machines.

Because we hadn't done any projects in 2D before, we had to learn how to use the 2D plane. For this reason, we followed the videos of youtuber "iHeartGameDev" (https://www.youtube.com/watch?v=UUJMGQTT5ts).

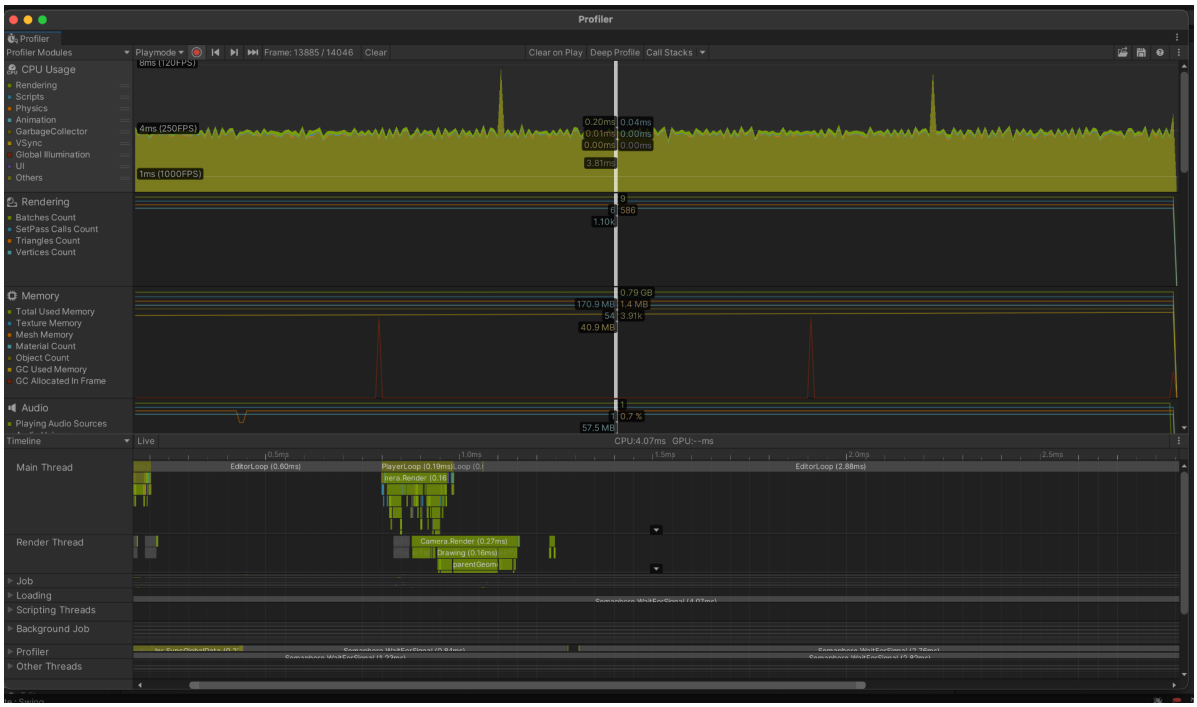
Challenges:

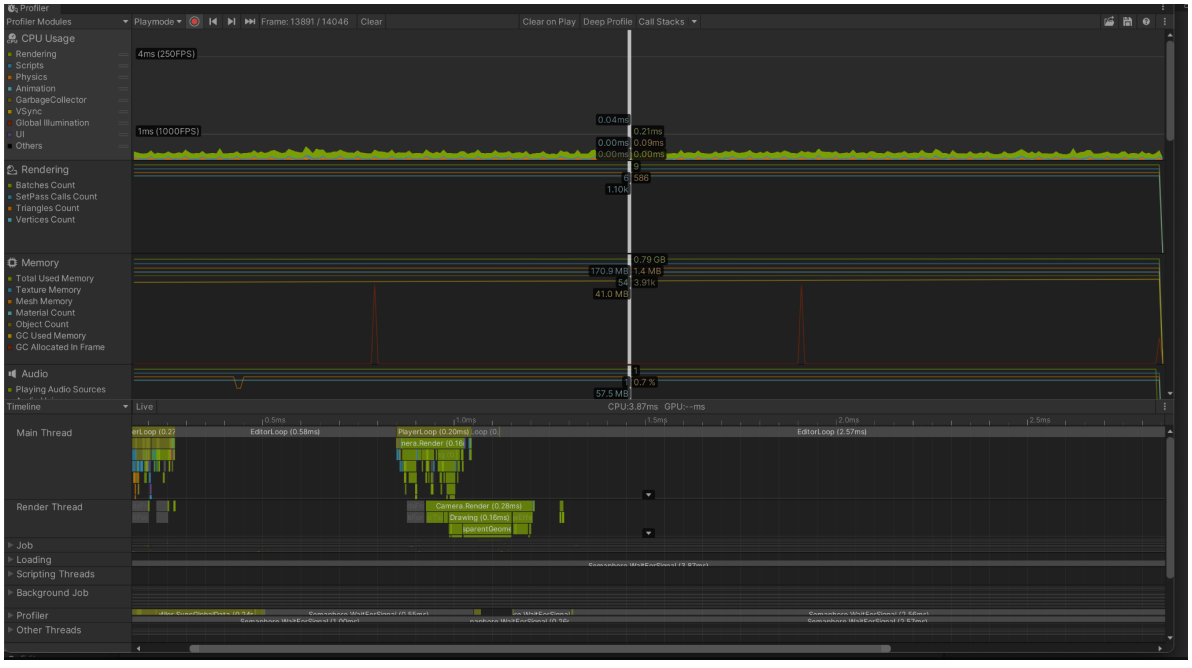
- Creating a rope motion with Bezier Curves was really challenging. However, even though we've achieved a rope with realistic motion, we couldn't make it swing with the player. Therefore, we've changed our design to an auto swinging rope.

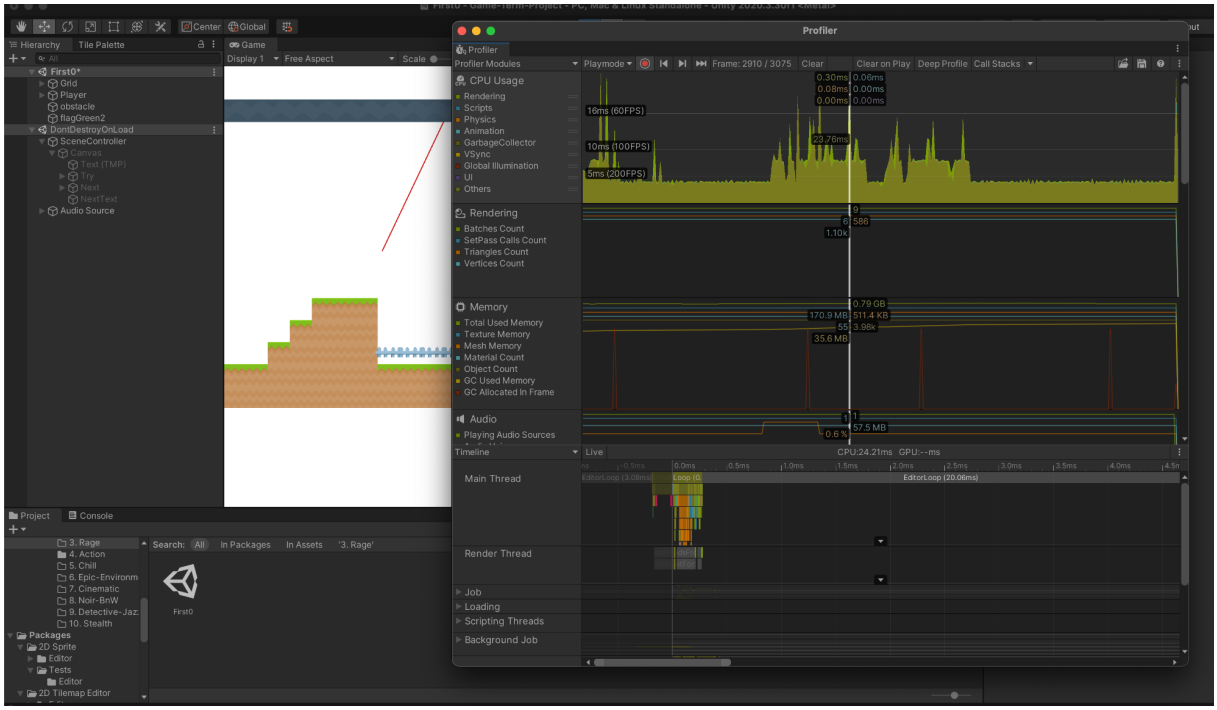
- Creating the structure of a finite state machine was challenging in the beginning. Designing a clear structure for states and their controllers seems logical before starting. After coding the state changes, it was confusing. Coding the physics effects should be done on Player script, states should be able to access those codes and run them, and so on. We wondered if it's an excessive approach or not during this phase. However, once we've achieved the basic states, it became easier and we understood the logic better.

- Designing the levels was also a bit difficult. We tried to adjust the difficulty levels of each level so that users can play the game without much frustration. It took more time than we expected.

Profiler:


Profiler
Profiler Modules
Playmode
Frame: 13885 / 14046
Clear
Clear on Play
Deep Profile
Call Stacks
CPU Usage
Rendering
Scripts
Physics
Animation
GarbageCollector
VSync
Global Illumination
UI
Others
4ms (250FPS)
1ms (1000FPS)
Rendering
Batches Count
SetPass Calls Count
Triangles Count
Vertices Count
Memory
Total Used Memory
Texture Memory
Mesh Memory
Material Count
Object Count
GC Used Memory
GC Allocated In Frame
Audio
Playing Audio Sources
Timeline
Live
CPU:4.07ms GPU:--ms
Main Thread
Render Thread
Job
Loading
Scripting Threads
Background Job
Profiler
Other Threads


Profiler
Profiler Modules
Playmode
Frame: 13891 / 14046
Clear
Clear on Play
Deep Profile
Call Stacks
CPU Usage
Rendering
Scripts
Physics
Animation
GarbageCollector
VSync
Global Illumination
UI
Others
4ms (250FPS)
1ms (1000FPS)
Rendering
Batches Count
SetPass Calls Count
Triangles Count
Vertices Count
Memory
Total Used Memory
Texture Memory
Mesh Memory
Material Count
Object Count
GC Used Memory
GC Allocated In Frame
Audio
Playing Audio Sources
Timeline
Live
CPU:3.87ms GPU:--ms
Main Thread
Render Thread
Job
Loading
Scripting Threads
Background Job
Profiler
Other Threads



As we can see above the profiler results, our game mostly spends resources on rendering because the game doesn't have a complicated physics structure. Therefore, the physics machine doesn't have to solve complicated collisions. It also doesn't have complicated calculations. Also, our software design helps to improve performance thanks to the FSM structure. There is no unnecessary computation to check the current state.

Our review for the game:

We think it is fun to play our game. Swinging on the ropes makes the game more challenging and fun! You have to wait for the perfect time to catch the perfect force while releasing the rope. Our game gets challenging level by level, which helps to keep the players in the game over time.