KARABUK UNIVERSITY
ENGINEERING DEPARTMENT
COMPUTER ENGINEERING

Software Engineering Project Assignment

INSTRUCTOR: Nesrin AYDIN ATASOY

Project Name: Fox's Tale

NAME/SURNAME/NO: Yılmaz Tarık / İpekçi / 2010205060

1.Project Plan

1.1 Project Information

In today's modern world, everyone is more or less busy with games. People play these games sometimes to escape the difficulties of real life, sometimes for fun, sometimes to kill time, and sometimes for competition. What causes this situation is that there are a lot of games of different genres. My game named Fox's Tale, which I developed and which I will describe in this documentation, is a game in the 2D platformer category. Although it includes typical platformer elements, it also contains many mechanics that I added as an extra such as s dashing, firing, double jumping, dropping items from enemies and gaining some buffs by collecting items. While developing this game, I used Unity game engine, C# programming language, and Aseprite for drawings.

1.2 Acceptance and Constraints

When starting the game, it will be assumed that the tutorial section has been passed successfully, the character is full of health, and has not reached any checkpoints. Since the game does not have a save system, if the player closes the game and then opens it again, the game starts over.

1.3 Project Tasks & Time-Work Table

The tool that I used to create this table: Excel

	Tasks	Duration	Starting Date	End Date
2	Project Information	2	22.09.2022	25.09.2022
3	Determining Acceptance and Constraints	2	25.09.2022	27.09.2022
4	Creating the Project Work-Timeline	3	27.09.2022	29.09.2022
5	Creation of Team Organization Chart	3	29.09.2022	3.10.2022
6	Creating Risk Table	3	3.10.2022	5.10.2022
7	Creating The Project Report	7	5.10.2022	13.10.2022
8	Creating User Scenarios	7	13.10.2022	21.10.2022
9	Creating Interaction Diagrams	3	21.10.2022	25.10.2022
10	Preparation of Contracts	9	25.10.2022	5.11.2022
11	Updating The Project Report	16	5.11.2022	25.11.2022
12	Modeling Of UI	7	25.11.2022	5.12.2022
13	Drawing The Class Diagram	5	5.12.2022	10.12.2022
14	Drawing The Sequence Diagram	4	10.12.2022	14.12.2022
15	Preparing The UI Design	9	14.12.2022	26.12.2022
16	Updating The Project Report 2	5	26.12.2022	31.12.2022
17	Performing The Coding Process	22	31.12.2022	29.01.2023
18	Testing	8	29.01.2023	8.02.2023
19	Publishing Process	3	8.02.2023	23.09.2023

1.4 GANTTCHART DIAGRAM

The tool that I used to make this chart:

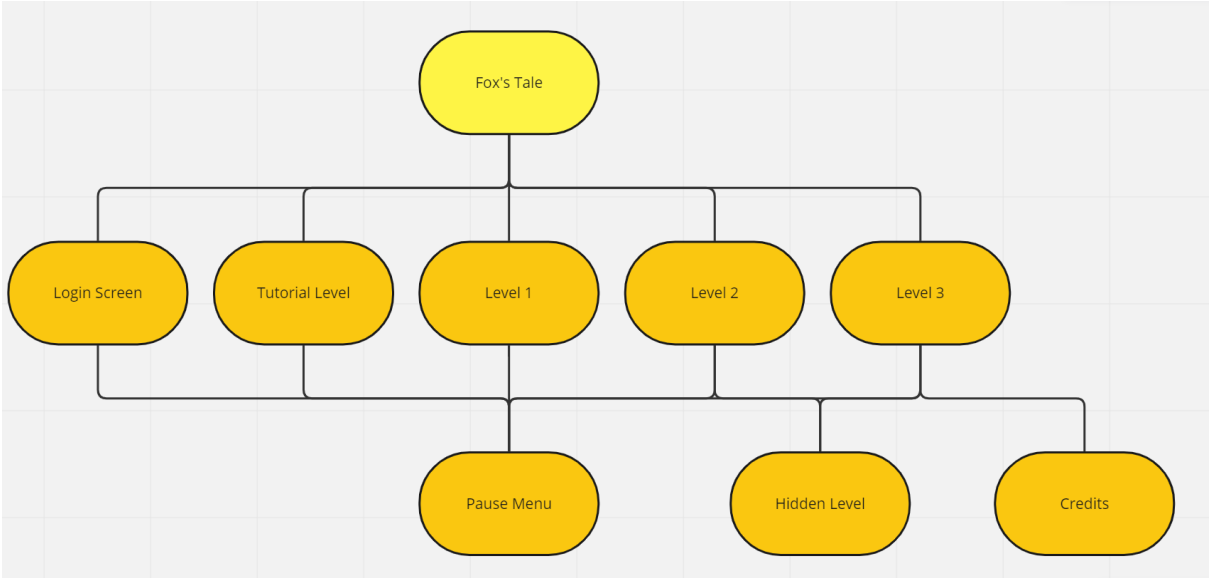
https://www.onlinegantt.com/#/gantt

1.5 Team Organization Chart, Task Distribution

	Name	Task
2	Yılmaz Tarık İpekçi	Project Information
3	Yılmaz Tarık İpekçi	Determining Acceptance and Constraints
4	Yılmaz Tarık İpekçi	Scenario 1-5
5	Yılmaz Tarık İpekçi	Scenario 5-10
6	Yılmaz Tarık İpekçi	Scenario 10-15
7	Yılmaz Tarık İpekçi	Gantt Diagram
8	Yılmaz Tarık İpekçi	Creation of Team Organization Chart
9	Yılmaz Tarık İpekçi	Task Distributions
10	Yılmaz Tarık İpekçi	Activity Diagram Scenario 1-5
11	Yılmaz Tarık İpekçi	Activity Diagram Scenario 5-10
12	Yılmaz Tarık İpekçi	Activity Diagram Scenario 10-15
13	Yılmaz Tarık İpekçi	Use Case
14	Yılmaz Tarık İpekçi	Contracts
15	Yılmaz Tarık İpekçi	Class Diagram
16	Yılmaz Tarık İpekçi	Sequence Diagram
17	Yılmaz Tarık İpekçi	Risk Table
18	Yılmaz Tarık İpekçi	Software Coding
19	Yılmaz Tarık İpekçi	Documentation
20	Yılmaz Tarık İpekçi	Testing

1.6 General Schema

The tool that I used to make this schema: https://miro.com/



1.7 Risk Table

#	Type Of Risk	Risk	Reason	Effect	Possibility	Precautions/Solutions
1	Player	The player does not have much gaming experience	The player may not have played this type of game before	High	Low	When advertising the game, it has to be well stated what kind of game it is and add various warnings.
2	Player	Player's budget	The fact that the player cannot buy the game or access all the contents of the game due to the exchange rate difference or economic reasons creates this risk.	High	Mid	Local pricing or lowering the price of the game.
3	Process	Postponing the end date	Sometimes everything may not go as planned, as the possibilities in the games bring a lot of bugs with them.	Mid	High	Game design documentation should be well and detailed. Also, software engineers need to be experienced.
4	Process	Employee fire/add	The change of people in tasks may disrupt the harmony and balance in the process. Reading and understanding other people's code and continuing it will slow down the process.	Mid	Mid	A Contract can be settled or the comfort of the employees should be provided by company.
5	Process	Lack of testing	Some optimization problems may arise after the games are released, as wrong results can often be obtained from the companies that are contracted to perform the testing process.	Very High	Mid	Inspection of test processes should be carried out frequently and well.
6	Technology	Inappropriate game engine usage	Some game engines are made to create specific games.	Very High	Very Low	You should code or use a game engine yourself depending on the type of game you wanna make.
7	Product	Product Scope	It may not be foreseen to what extent the product will reach a place.	High	Low	The planning phase of the project should be well designed and analyzed.
8	Management	Expectation	What players expect and encounter may be different.	Very High	High	To solve this, we need to analyze the market very well and evaluate the comments received if we have already released other games.
9	Source	Sourcing risks	Insufficient budget and other resources	Very High	Mid	The budget and other resources to be allocated to the game should be well analyzed.
10	Employee	Lack of information and experience	Employee's lack of experience or general lack of knowledge about the type of project being done	Very High	Low	Employees in recruitment need to be well interviewed and acquaintances should not be hired.

2.Stage Designing

2.1 Usage Scenarios and Interaction Diagrams

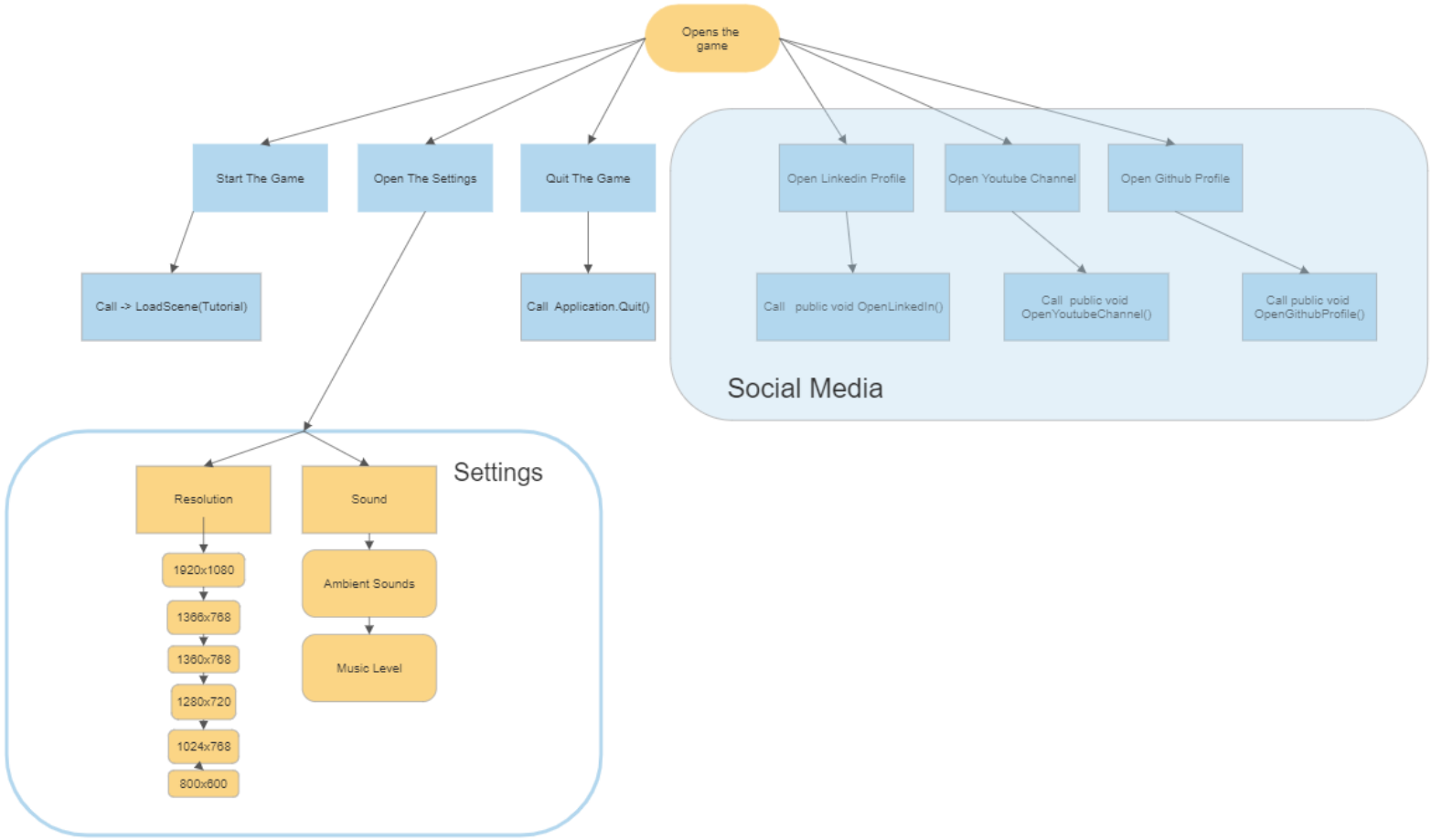
Use Case Scenario 1 : Open the main menu

Primary Actor: Player

Pre-Condition: The player must have downloaded the game.
Post-Condition: The player opened the menu.

Main Scenario:
1-Player opens the game.
2-Player opens the main menu.

Alternative Scenario: If the platform of the device the player is using is not supported by this game the game won’t open.

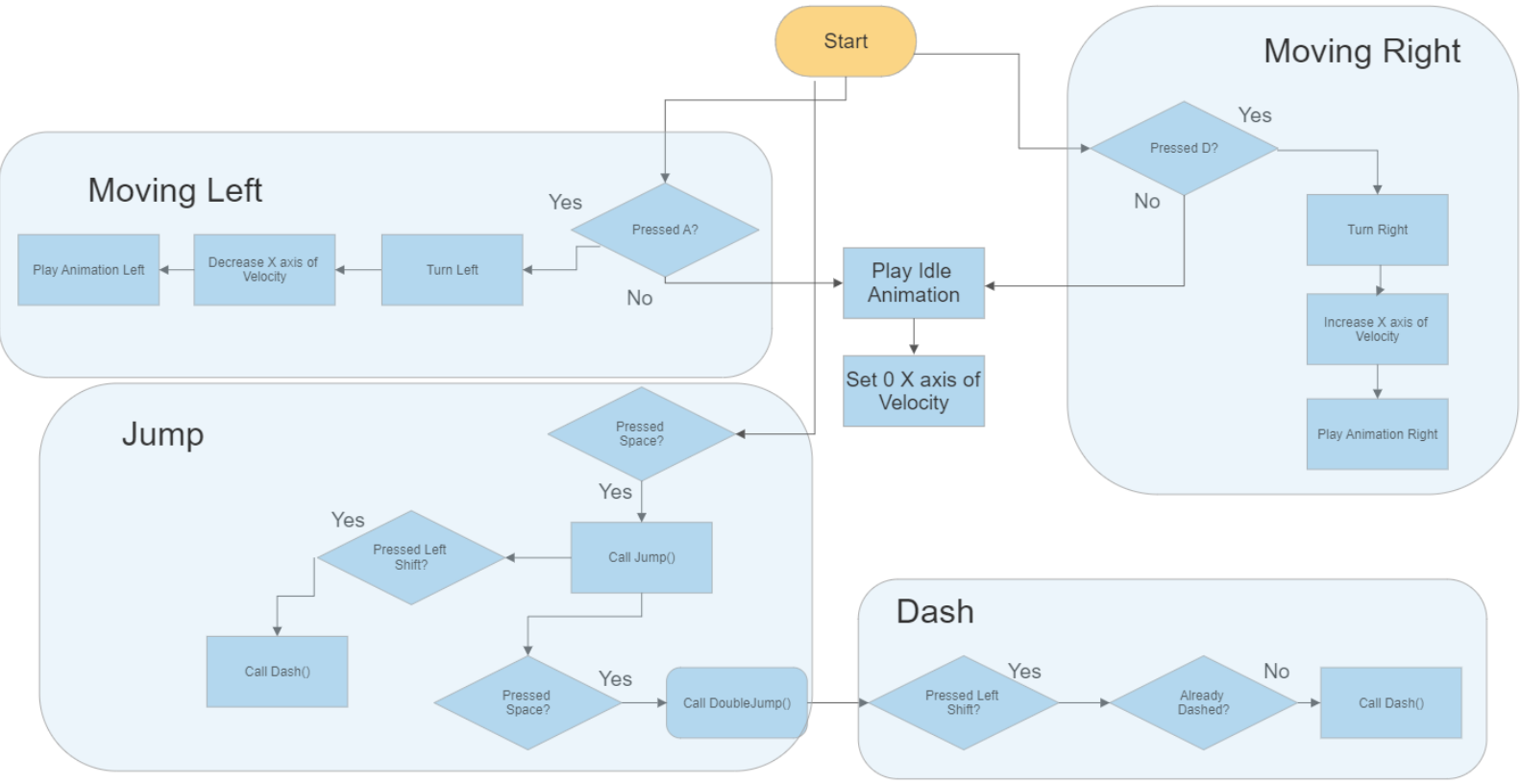


Use Case Scenario 2: Player Movement

Primary Actor: Player

Pre-Condition: The A,D and space keys must have been pressed.
Post-Condition: The player has moved.

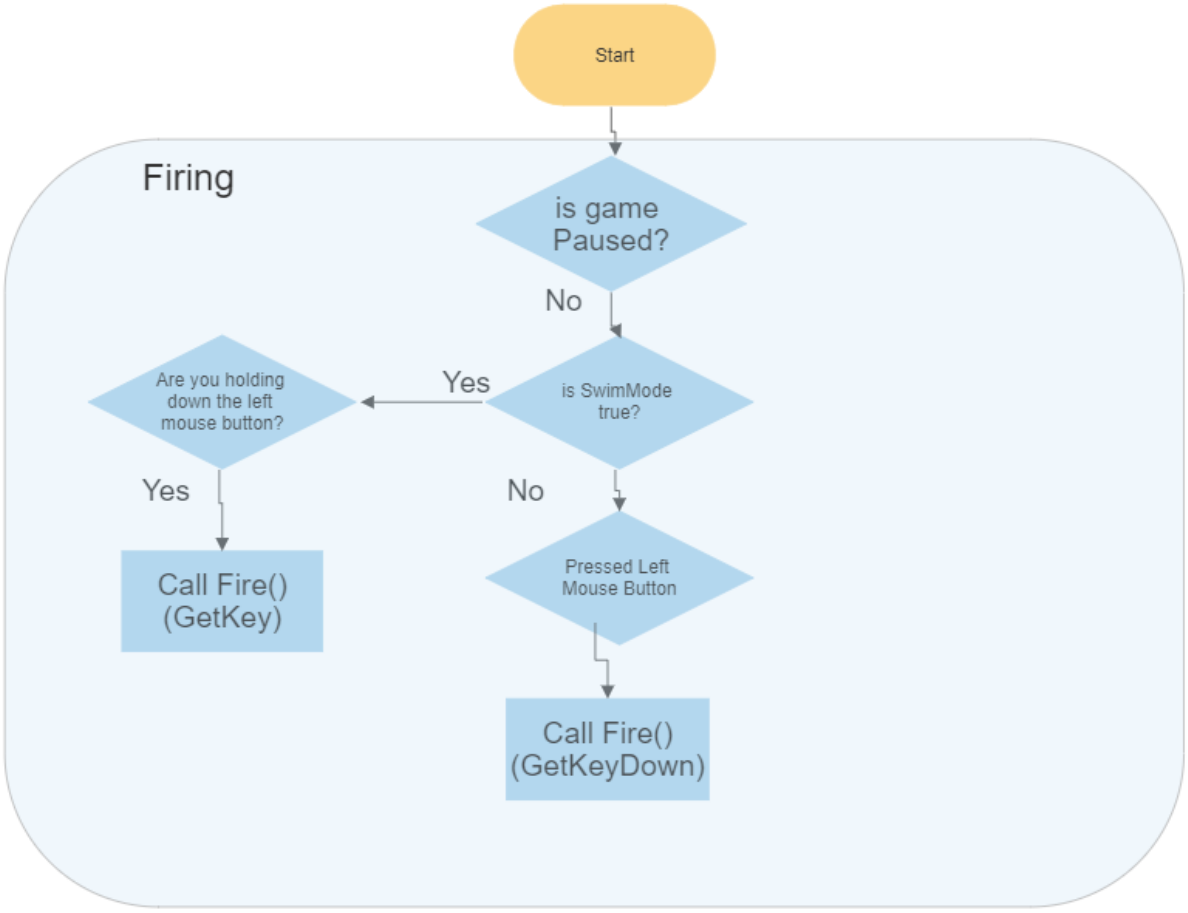
Main Scenario:
1-Player has gone left.
2-Player has gone right.
3-Player has jumped.
4-Player has dashed.



Use Case Scenario 3: Shooting/Firing
Primary Actor: Player

Pre-Condition: The player must press the left mouse button.
Post-Condition: The player has shot to way which he/she looked.

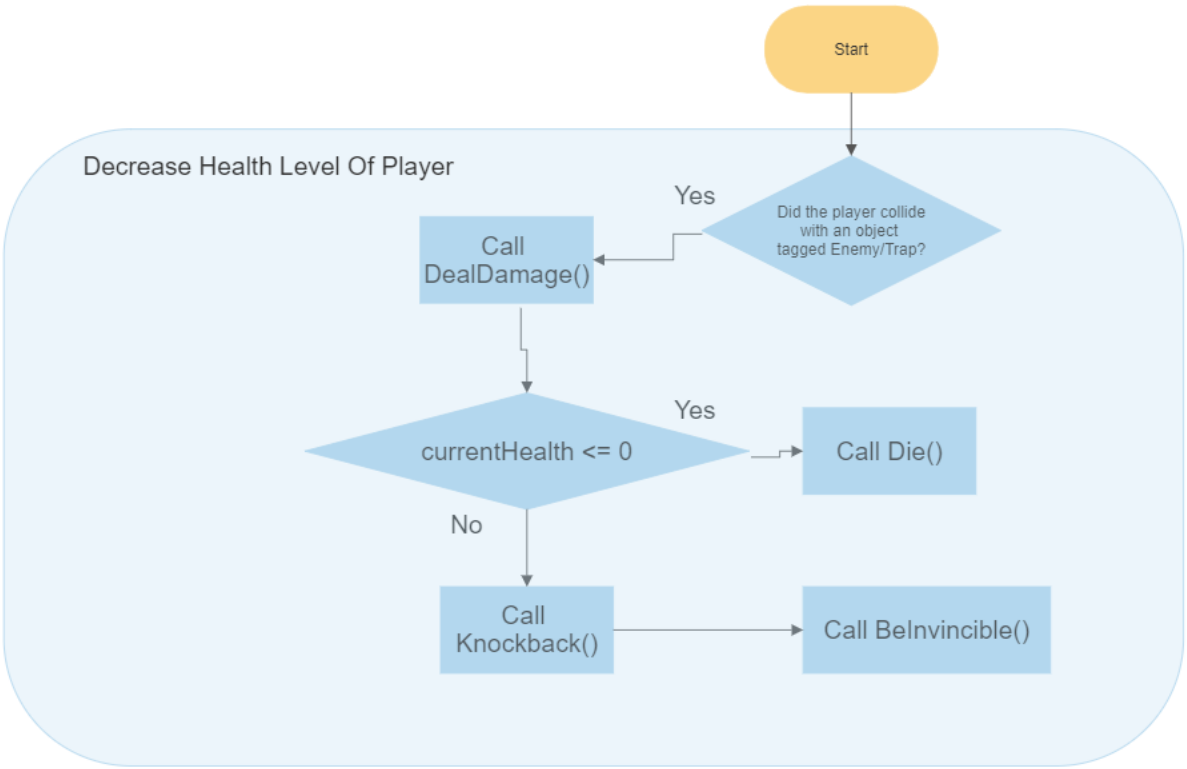
Main Scenario:
1-The player has fired.
2-The player has killed a monster.



Use Case Scenario 4: Decrease Health Level of Player
Primary Actor: Player

Pre-Condition: Health level of the player must be greater than 0.
Post-Condition: Player has got damage or died.

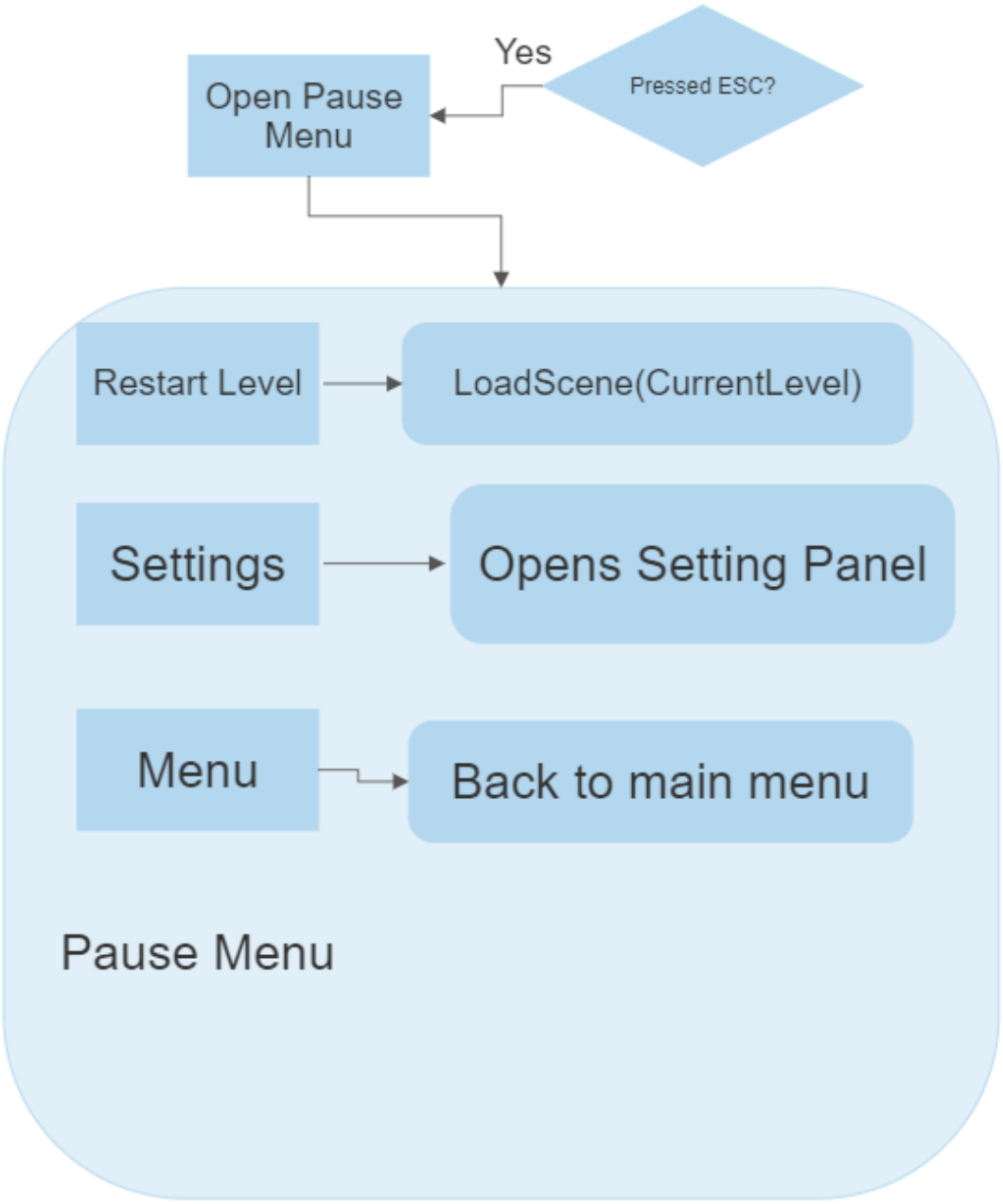
Main Scenario:
1- The player collided with the arrow.
2- The player collided with the spike.
3- The player collided with the enemy/boss.



Use Case Scenario 5: Open the Pause Menu
Primary Actor: Player

Pre-Condition: Player must be alive and escape button must have been pressed.
Post-Condition: Game has stopped.

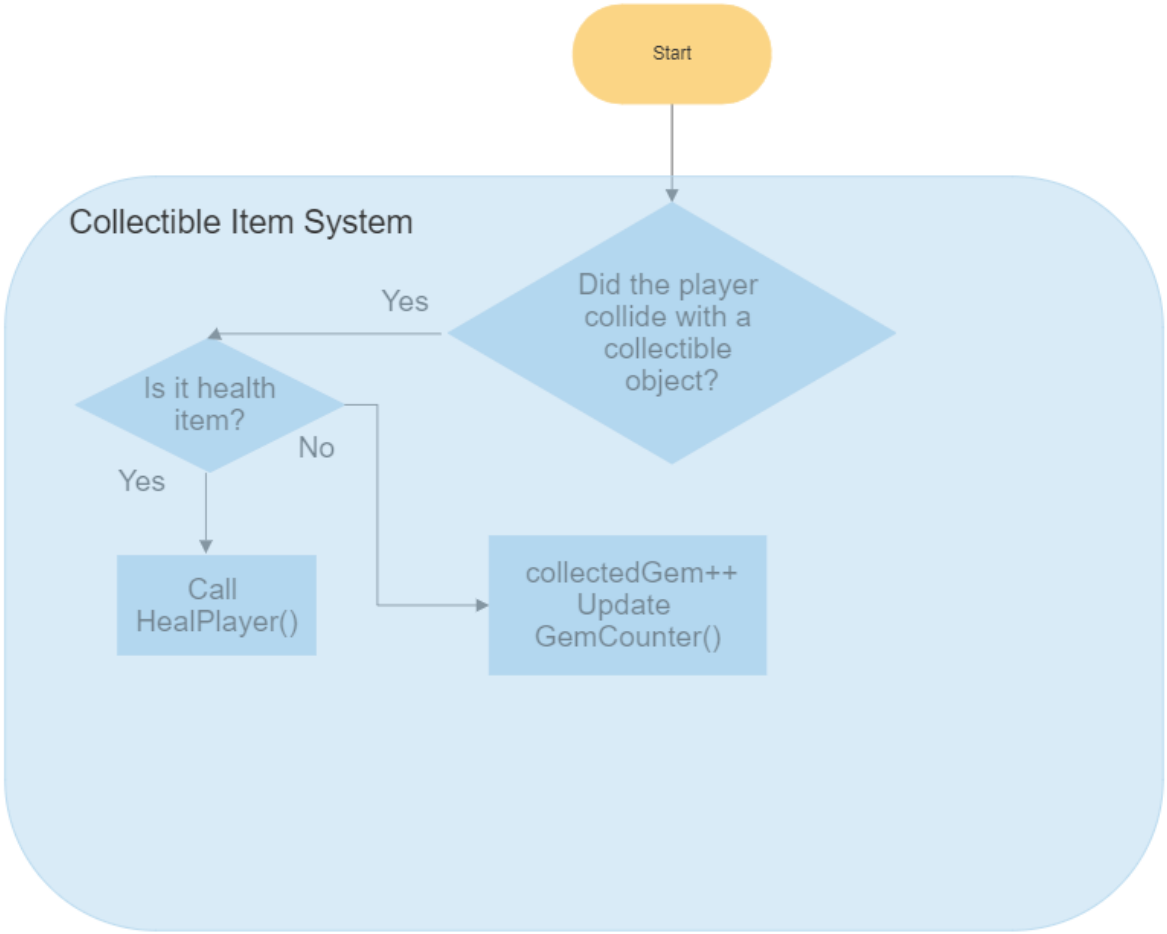
Main Scenario:
1- The player restarted the current level.
2- The player has changed the resolution of screen.
3- The player has changed general sound level.
4-The player has changed ambient/music sound level.
5-The player has closed the game.


Yes
Pressed ESC?
Open Pause Menu
Restart Level
LoadScene(CurrentLevel)
Settings
Opens Setting Panel
Menu
Back to main menu
Pause Menu

Use Case Scenario 6: Collectible Item System
Primary Actor: Player

Pre-Condition: Player object must have a collider component.
Post-Condition: Player got an item.

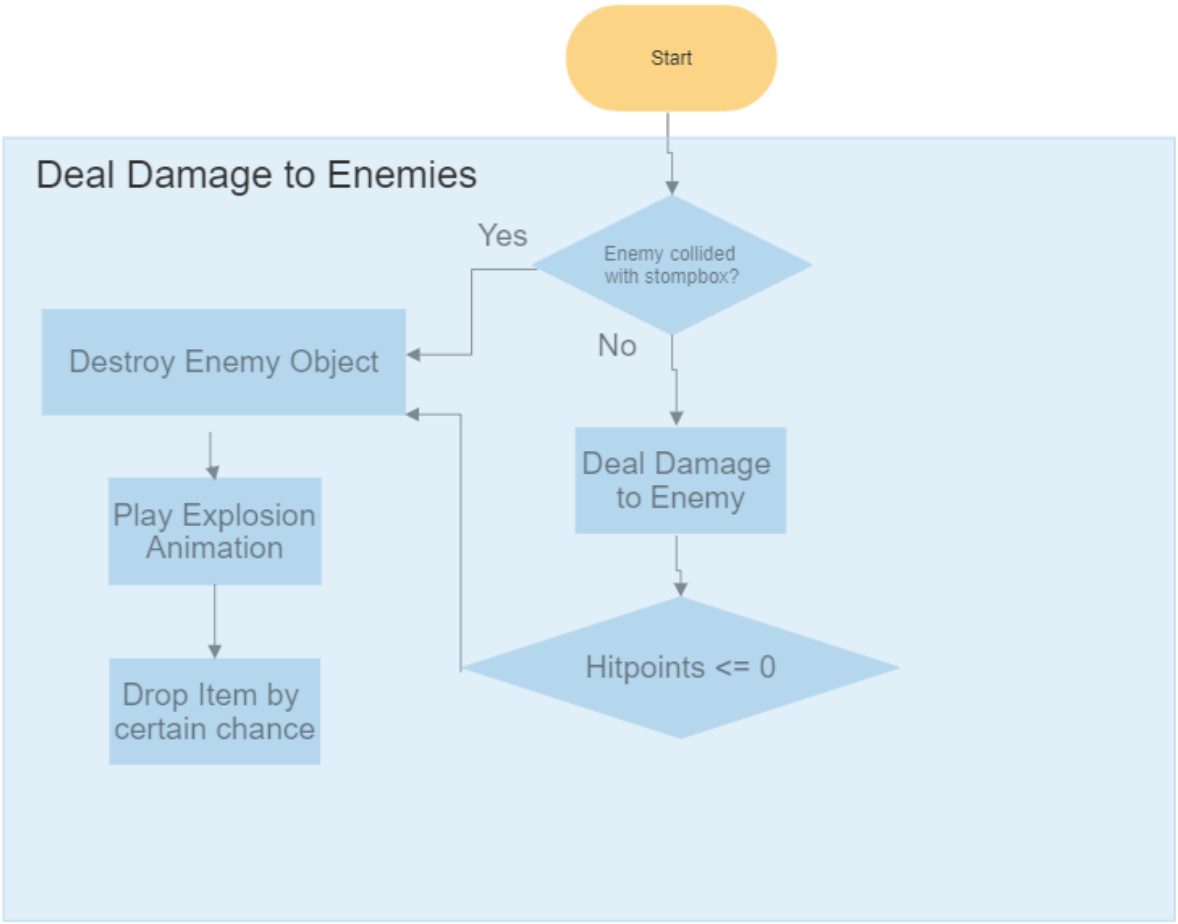
Main Scenario:
1- Player has got a health item and has increased his health level.
2- Player has got a gem and gem counter has increased.



Use Case Scenario 7: Deal Damage to Enemies
Primary Actor: Player

Pre-Condition: Enemy's health level must be greater than 0 and player's knockback cooldown must be >0 0
Post-Condition: Enemy has got damage.

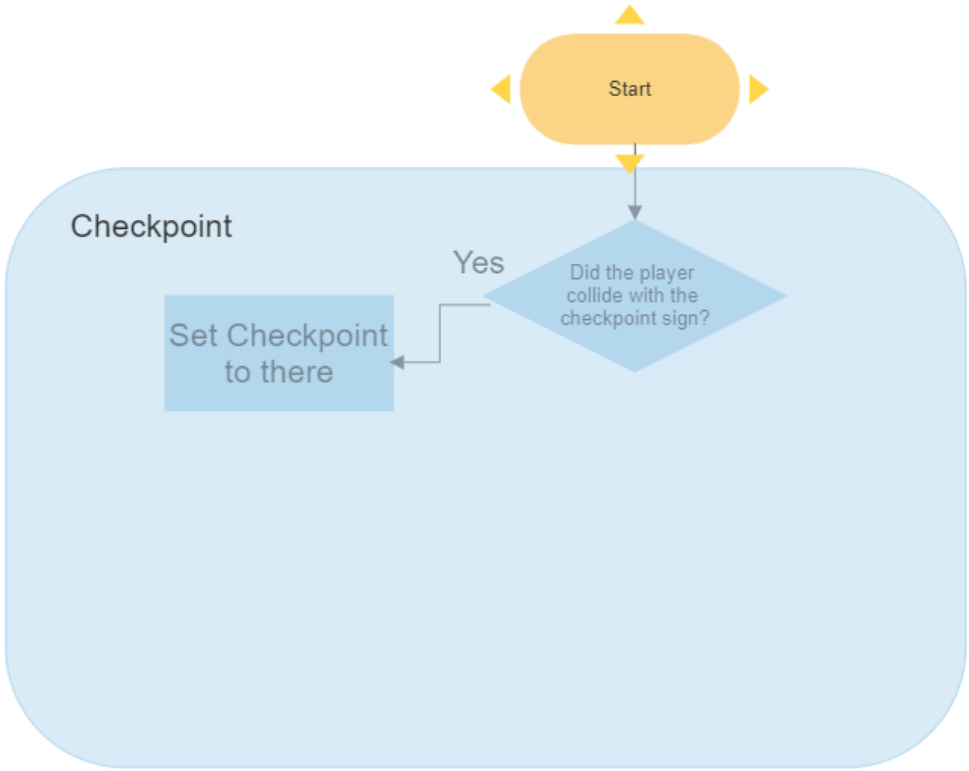
Main Scenario:
1- Enemy has died and dropped health item.
2- Enemy has died and dropped gem.
3- Enemy has died and dropped nothing.
4- Enemy has got damage.



Use Case Scenario 8: Checkpoint System
Primary Actor: Player

Pre-Condition: Player object must have a collider component.
Post-Condition: Spawn point saved.

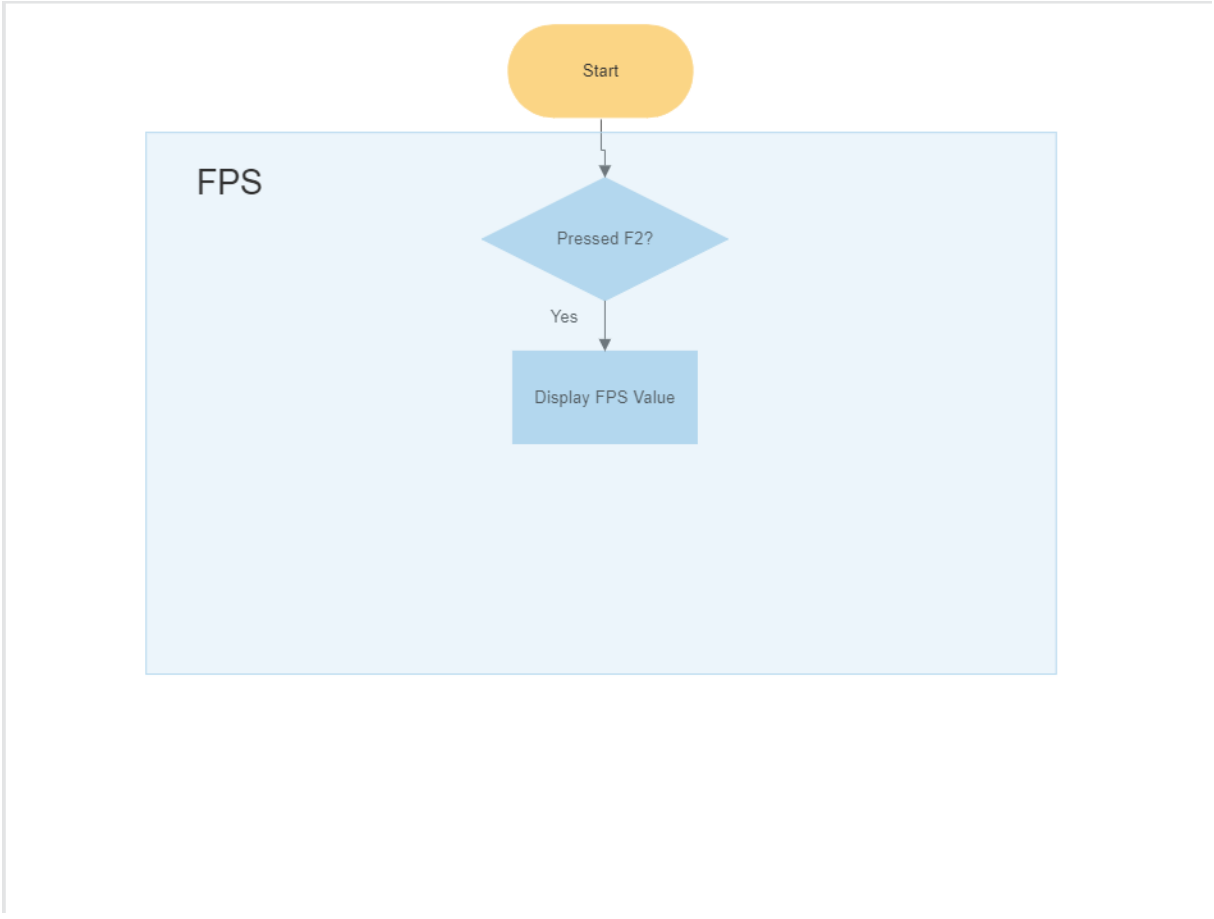
Main Scenario:
1- The player has reached a save point for the first time.
2- The player has reached another save point and has inactivated the previous ones.



Use Case Scenario 9: Display FPS
Primary Actor: Player

Pre-Condition: The game should be opened.
Post-Condition: The fps value is displayed.

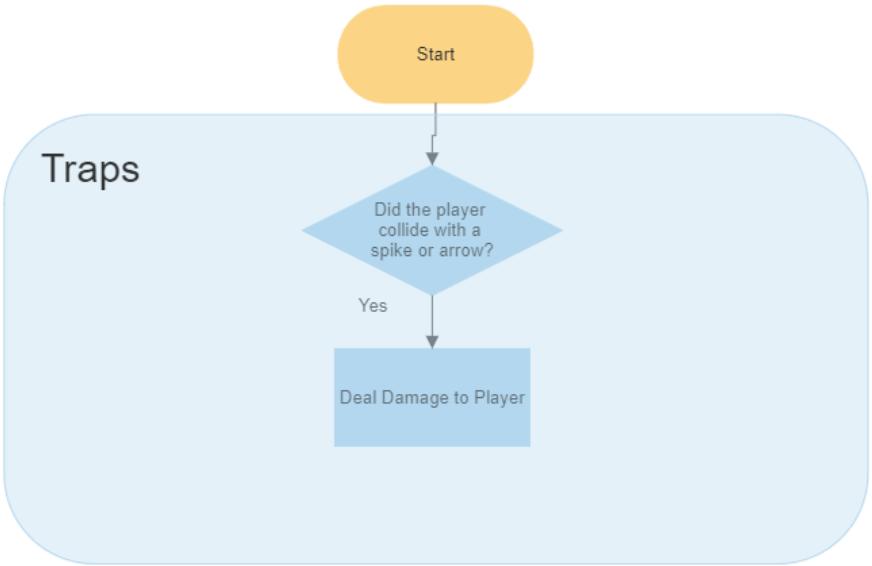
Main Scenario:
1- The fps value appears in the upper right corner of the screen.



Use Case Scenario 10: Traps
Primary Actor: Player

Pre-Condition: Player object must have a collider component
Post-Condition: Player gets damage.

Main Scenario:
1- Player has got damage and died.
2- Player has got damage and knocked back.

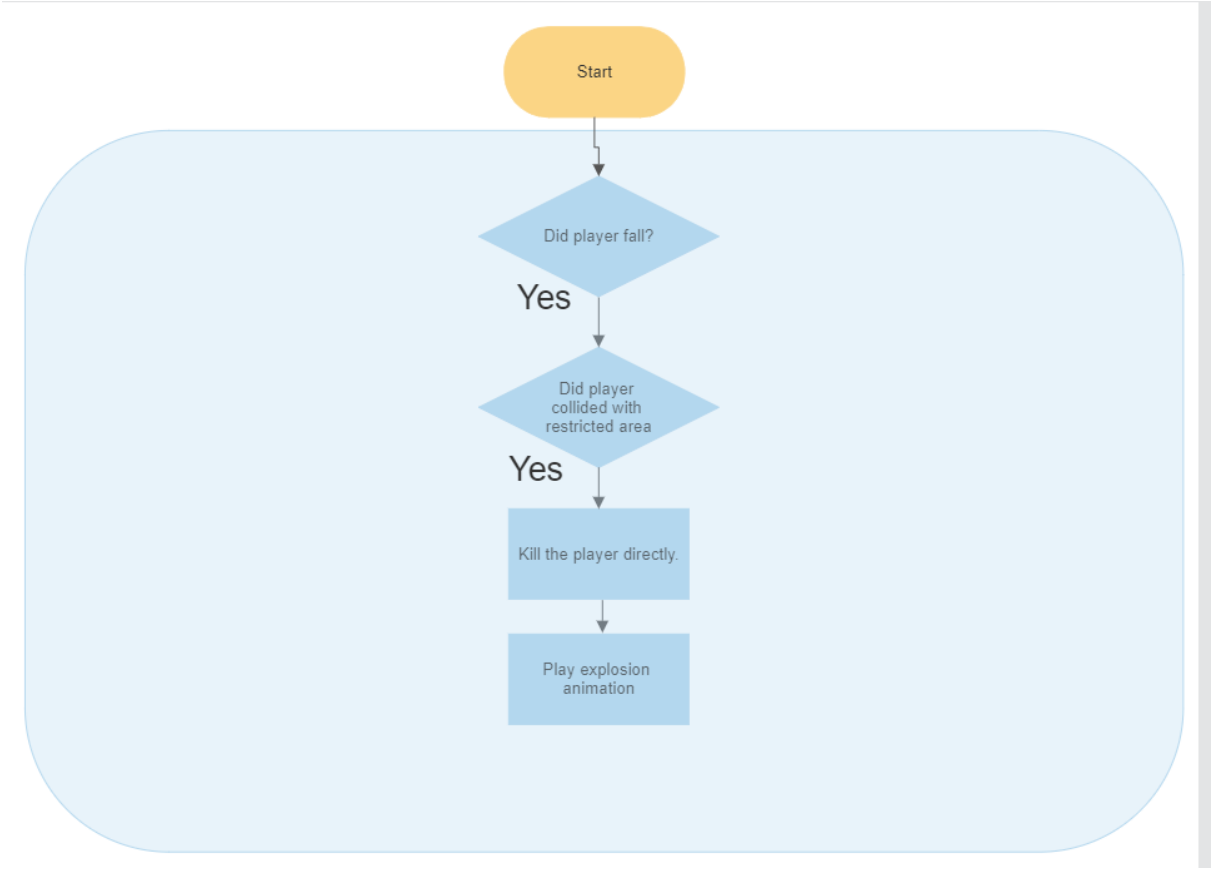


Use Case Scenario 11: Falling

Primary Actor: Player

Pre-Condition: Player object's rigid body component's gravity value must be greater than 0.
Post-Condition: The player has fallen.

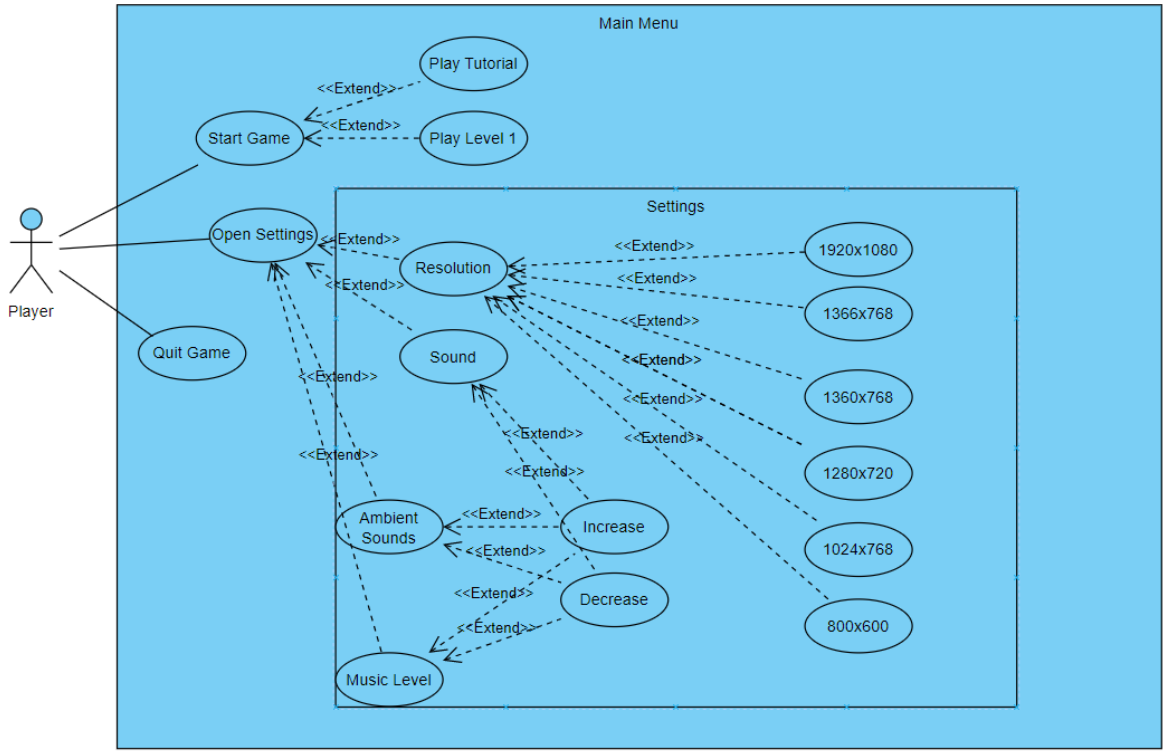
Main Scenario:
1- Player has fallen and nothing happened.
2- Player has fallen and died.



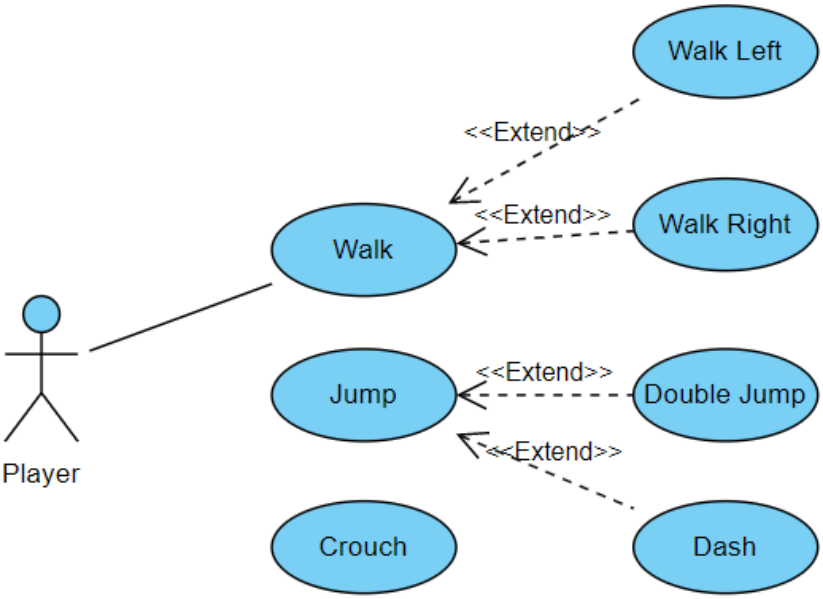
2.2 Use Case Diagrams

All of use case diagrams has made with: https://online.visual-paradigm.com/

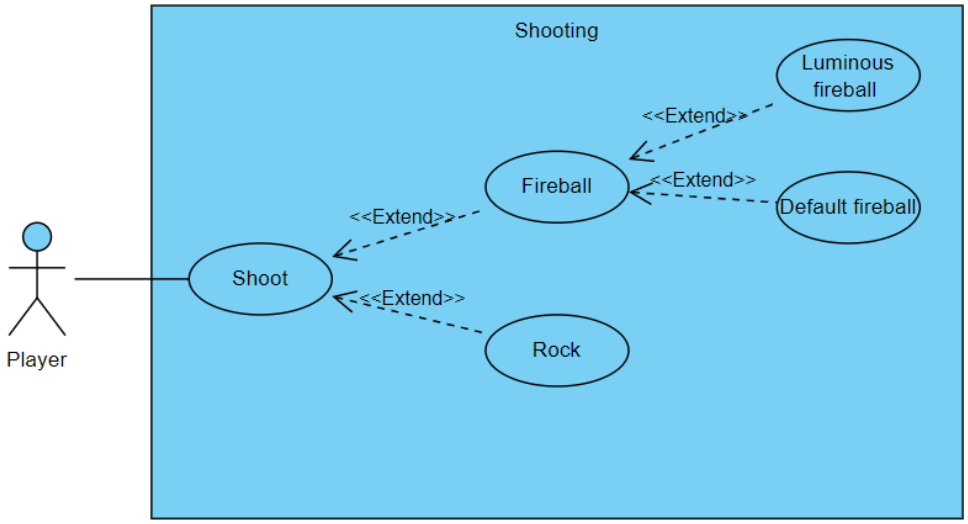
Use Case Diagram 1: Main Menu



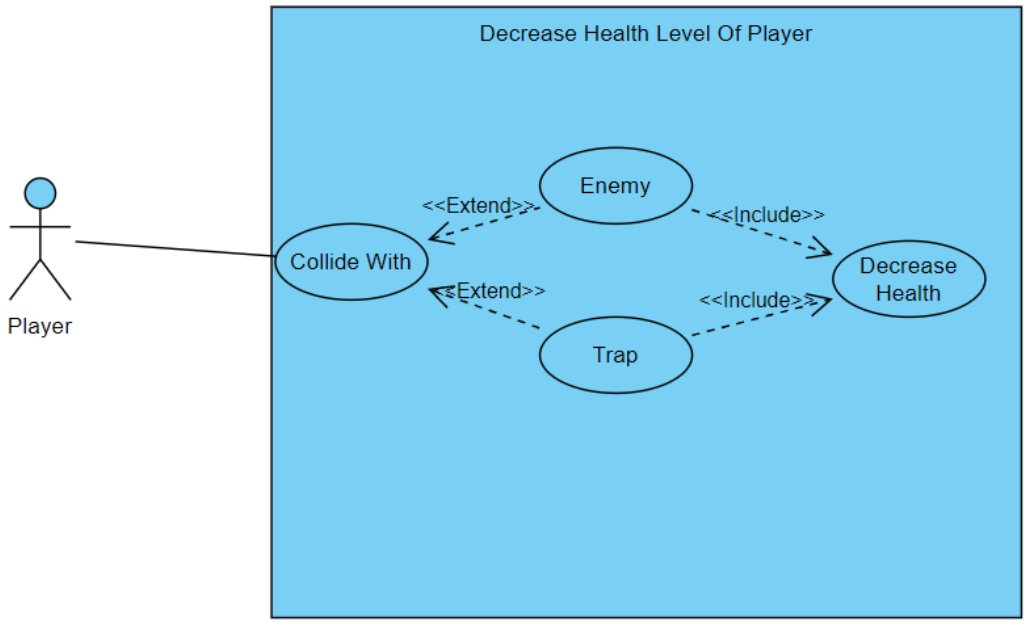
Use Case Diagram 2: Player Movement



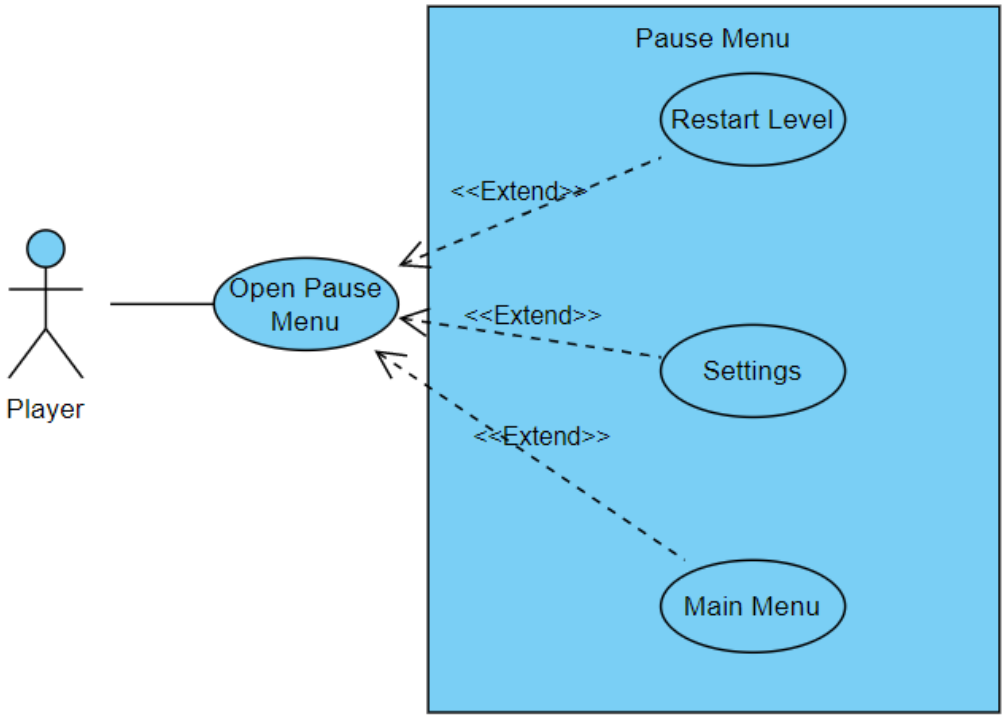
Use Case Diagram 3: Shooting/Firing



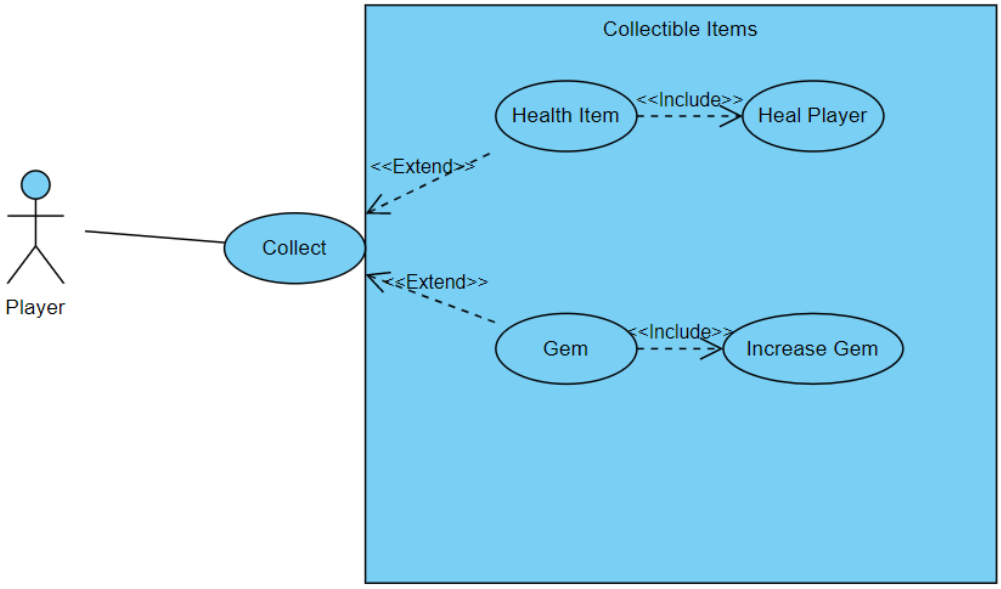
Use Case Diagram 4: Decrease Health



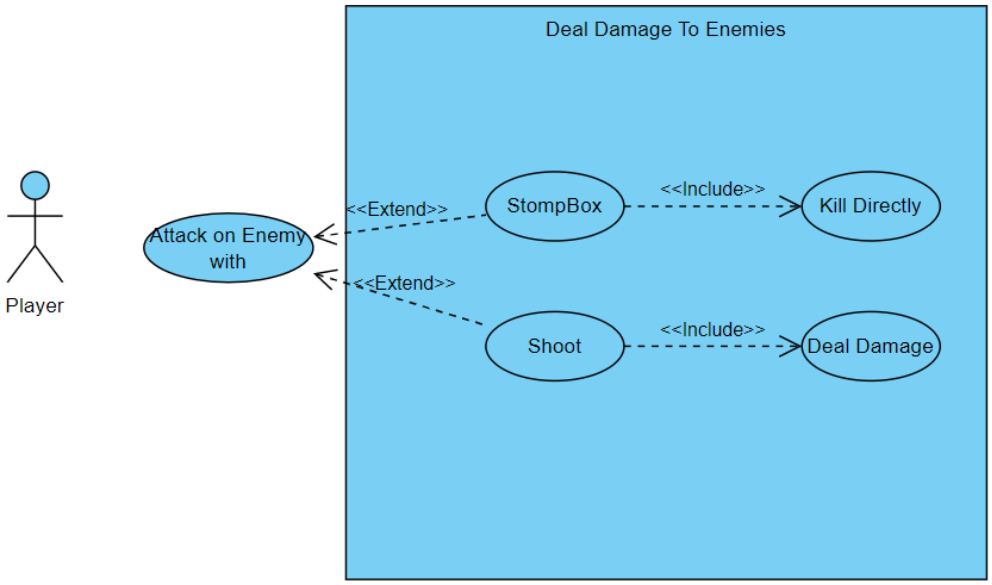
Use Case Diagram 5: Open Pause Menu



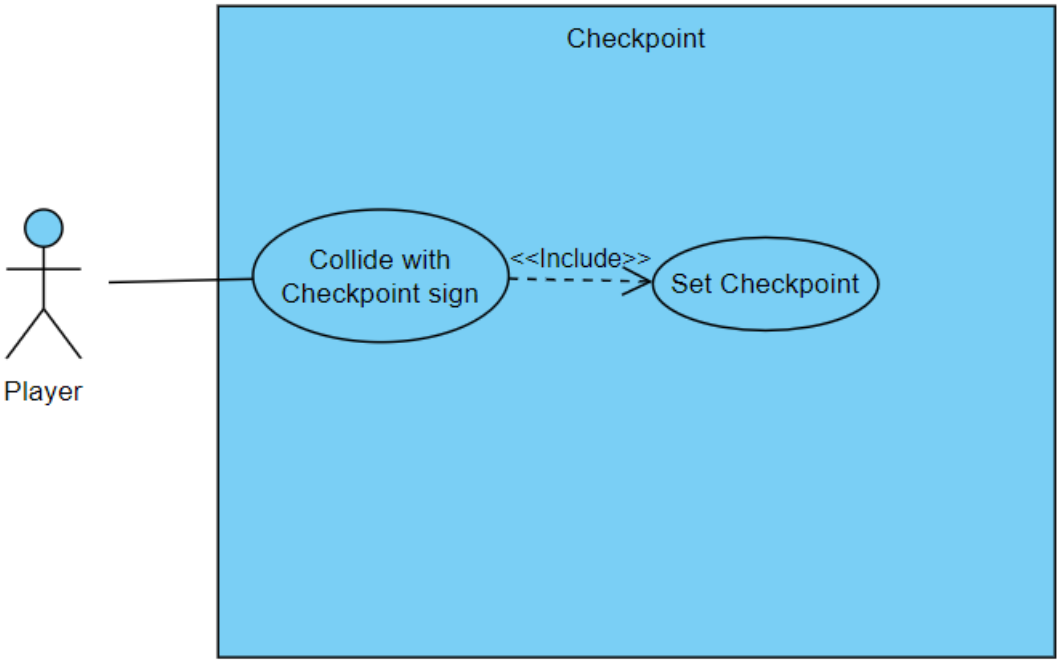
Use Case Diagram 6: Collectible Items



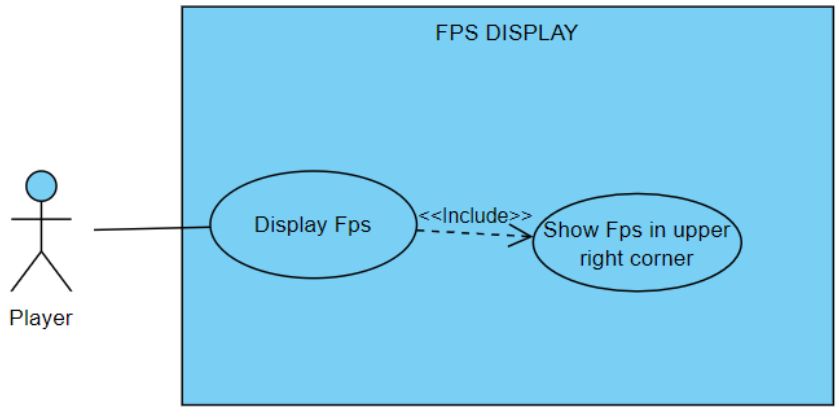
Use Case Diagram 7: Deal Damage To Enemies



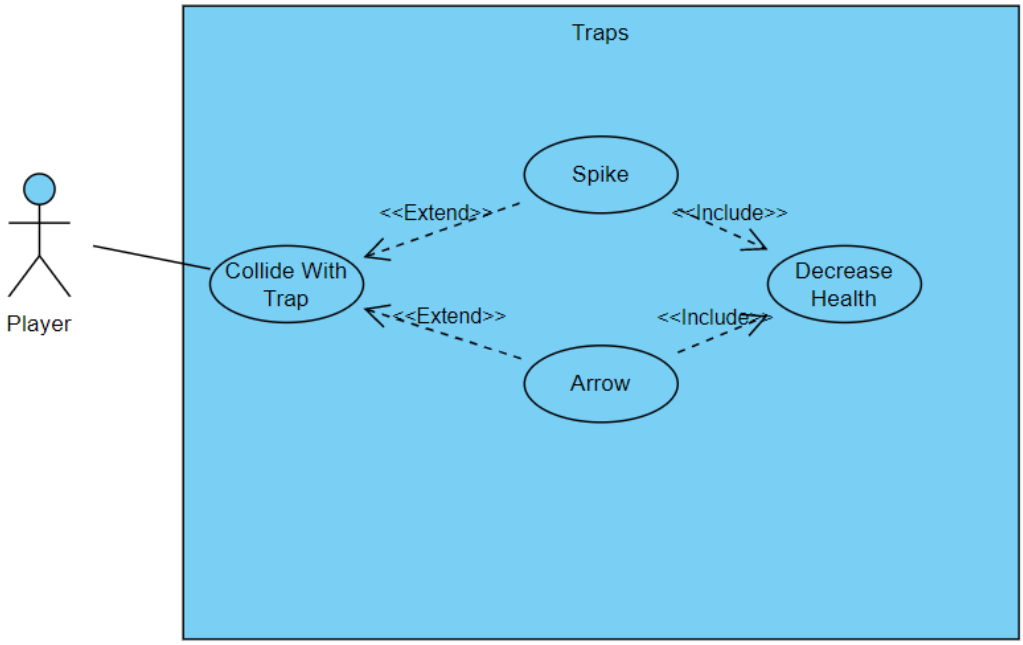
Use Case Diagram 8: Checkpoint System



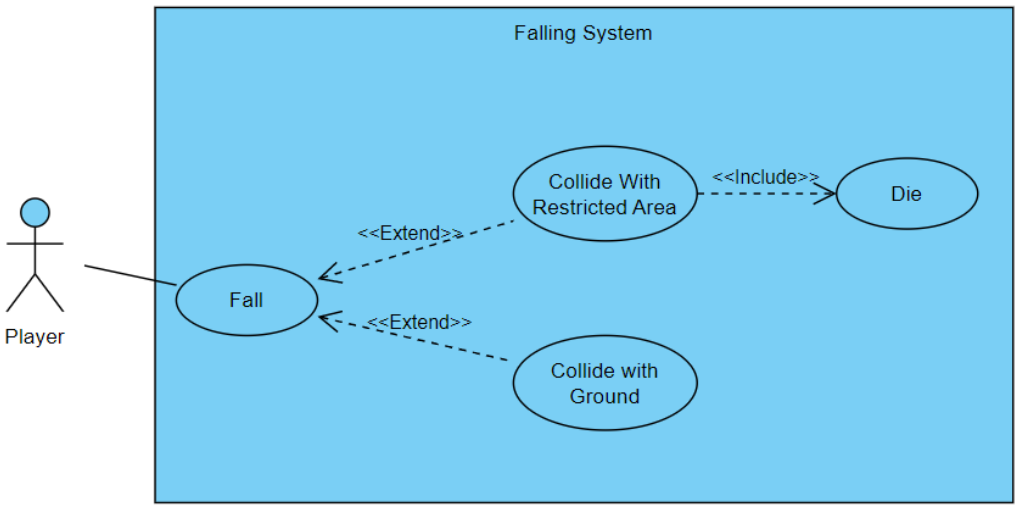
Use Case Diagram 9: Display FPS



Use Case Diagram 10: Traps



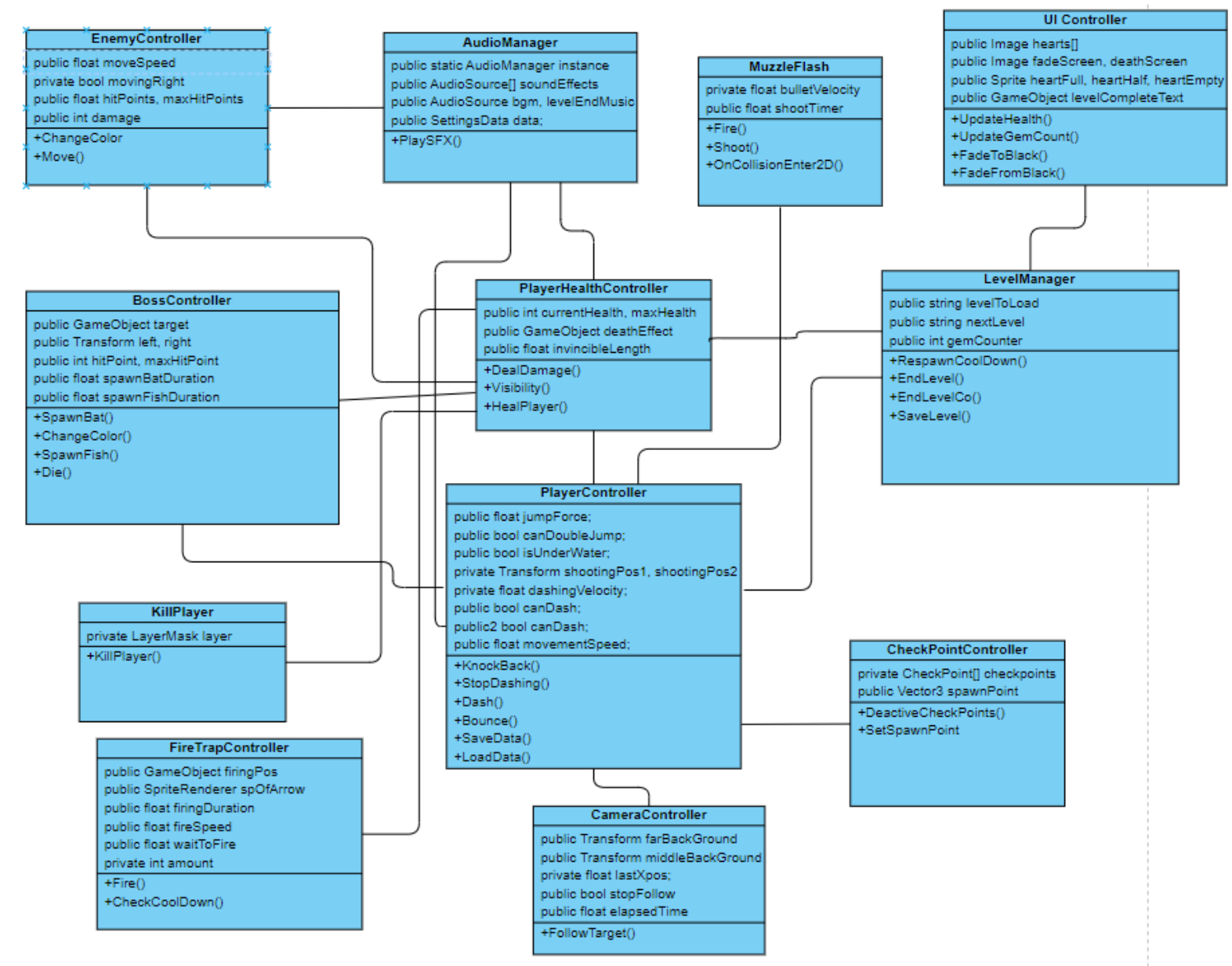
Use Case Diagram 11: Falling



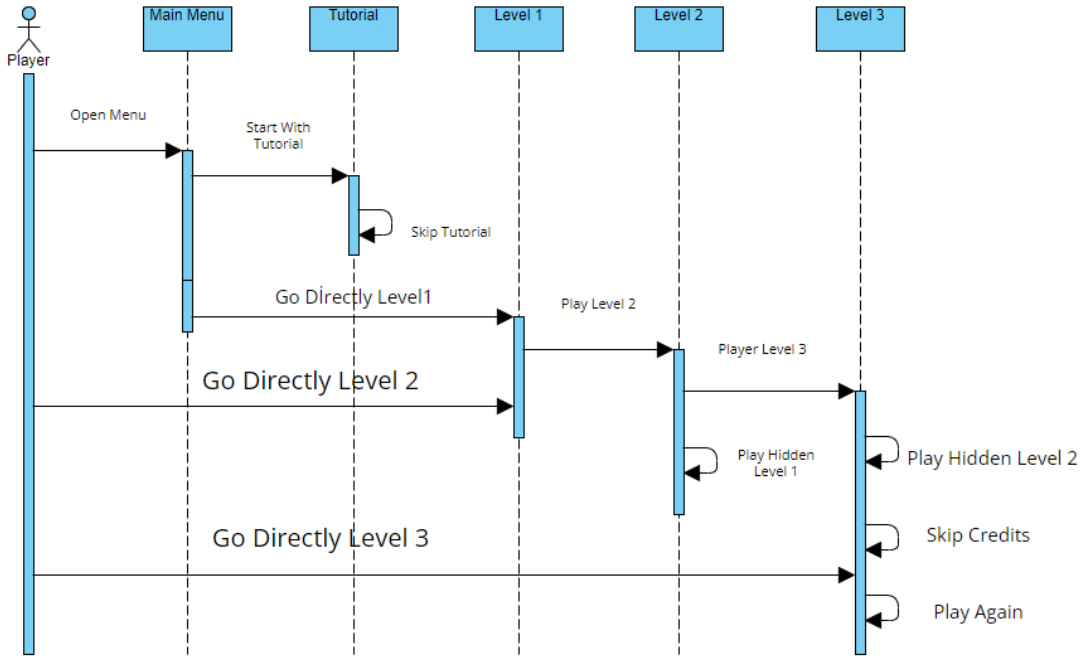
2.3 Contracts

Contract No: 1	Main Menu
Process	LoadScene(MainMenu)
Prerequisites	-
Results	The player opened the game.
Contract No:2	Player Movement
Process	TurnLeft(), TurnRight(), GoLeft(), GoRight()
Prerequisites	The A and D keys must have been pressed.
Results	Player turns right/left or goes right/left
Contract No:3	Firing
Process	Shoot()
Prerequisites	The player must press the left mouse button
Results	The player shoots in the direction facing
Contract No:4	Decrease Health Level Of Player
Process	DealDamage(), KillPlayer()
Prerequisites	Health level of player must be greater than 0
Results	Player gets damage or dies
Contract No:5	Pause Menu
Process	OpenPauseMenu(), PauseGame(), Settings(),MainMenu()
Prerequisites	Player must be alive
Results	Game stops
Contract No:6	Collectible Item System
Process	HealPlayer(), UpdateGemCounter(), CollectGem()
Prerequisites	Player object must have a collider component.
Results	Gem counter or health level increases
Contract No:7	Deal Damage to Enemies
Process	DropItem(), DealDamageToEnemy()
Prerequisites	Enemy's health level must be greater than 0 and player's knockback cooldown must be >= 0
Results	Enemy dies and maybe item drops.
Contract No:8	Checkpoint System
Process	SetCheckpoint(), DisableOtherCheckpoints()
Prerequisites	Player object must have a collider component
Results	The new checkpoint is saved.
Contract No:9	FPS
Process	DisplayFpsValue()
Prerequisites	The game should be opened.
Results	The fps value appears in the upper right corner of the screen.
Contract No:10	Traps
Process	DealDamage()
Prerequisites	Player object must have a collider component
Results	Player gets damage
Contract No:11	Falling
Process	-
Prerequisites	Player object's rigid body component's gravity value must be greater than 0
Results	Player falls and dies

2.4 Class Diagram



2.5 Sequence Diagram



3. Testing Stage

3.1 Quality Criteria

#	State
Functionality	By performing all the events with functions, I can easily intervene in the events and quickly solve the problems.
Reliability	Everything requested from the project works smoothly and as expected.
Usability	Since it is a game project that is very easy to play and can be adapted quickly, I think I have done a successful job in usability.
Efficiency	Some parts were coded as optimized as possible, but since other parts could have been much more optimized, we cannot say that it was a successful project.
Maintainability	All events in the project are provided by functions in a systematic way. This simplifies maintability.
Portability	It can be considered successful in portability as it will be easy to transition to different platforms.

3.2 What is Unit Test and Test Runner in Unity?

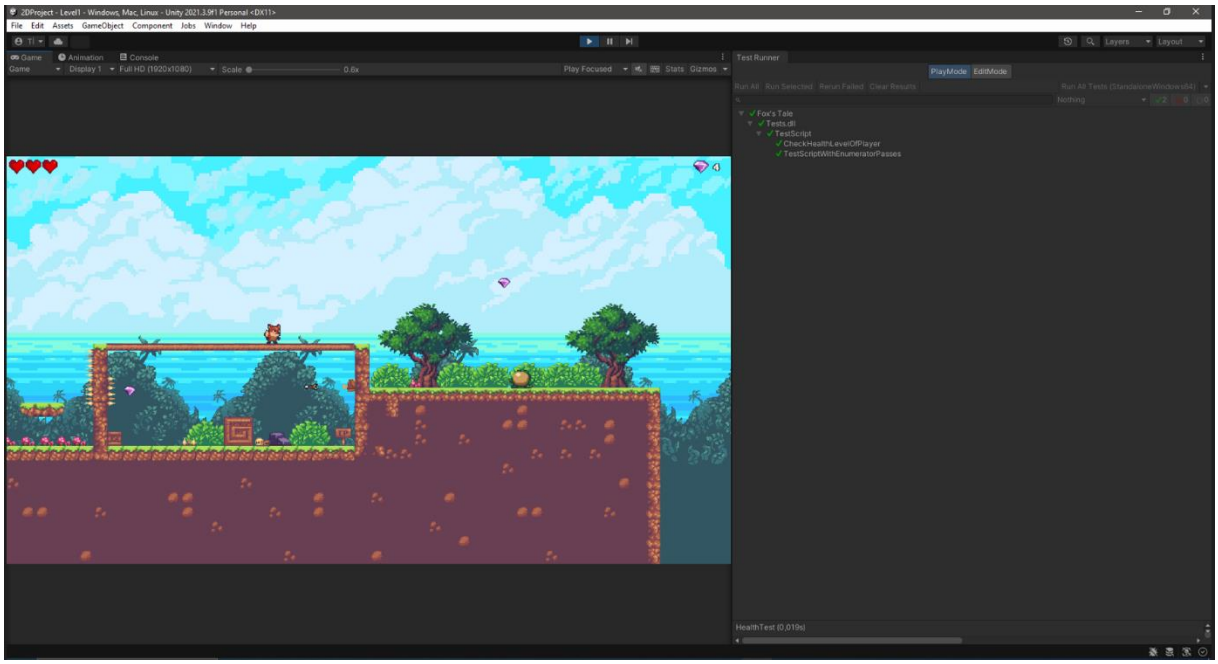
Unit Test: A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property.

Test Runner: The Unity Test Runner uses a Unity integration of the NUnit library, which is an open-source unit testing library for .Net languages. More information about NUnit can be found on the official NUnit website and the NUnit documentation on GitHub.

3.3 Unit Tests Of My Project

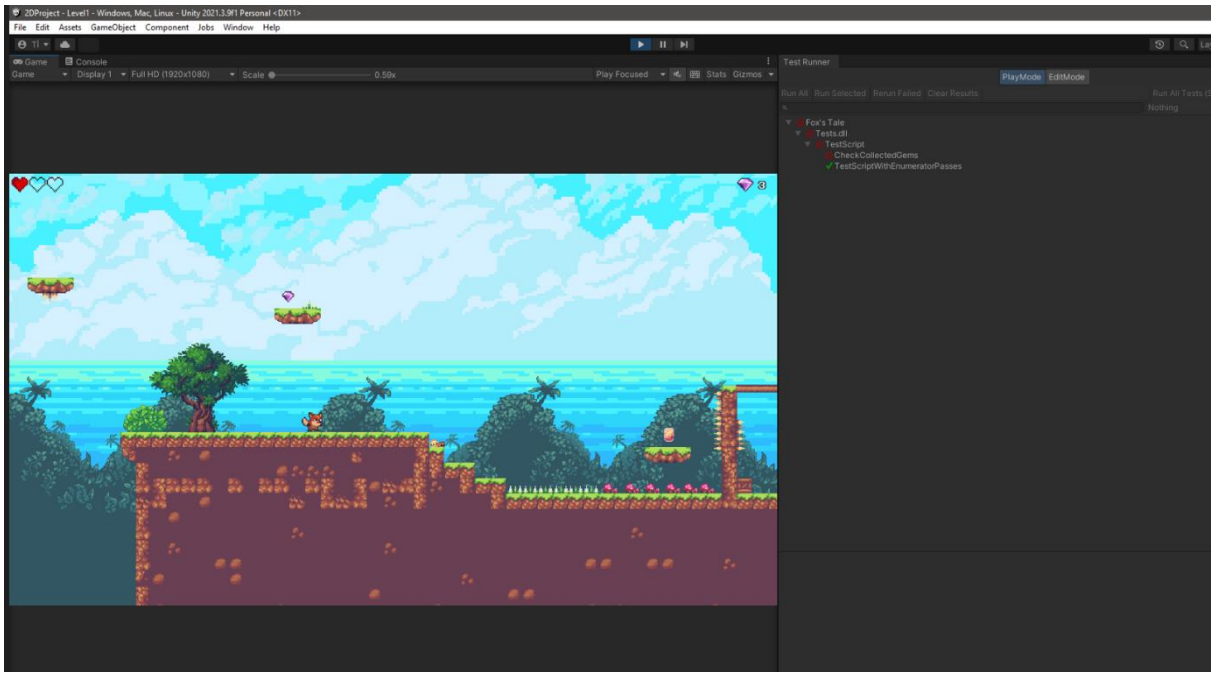
Testing Health Level Of Player:

```
using System.Collections;
using System.Collections.Generic;
using NUnit.Framework;
using Player;
using UnityEngine;
using UnityEngine.TestTools;

public class TestScript
{
    // A Test behaves as an ordinary method
    [Test]
    public void CheckHealthLevelOfPlayer()
    {
        // Use the Assert class to test conditions
        var healthOfPlayer = new PlayerHealthController();
        var healthLevelOfPlayer :int  = healthOfPlayer.currentHealth;
        Assert.AreEqual( expected: healthLevelOfPlayer,  actual: healthOfPlayer.maxHealth);
    }

    // A UnityTest behaves like a coroutine in Play Mode. In Edit Mode you can use
    // `yield return null;` to skip a frame.
    [UnityTest]
    public IEnumerator TestScriptWithEnumeratorPasses()
    {
        // Use the Assert class to test conditions.
        // Use yield to skip a frame.
        yield return null;
    }
}
```



Testing Collected Gems:

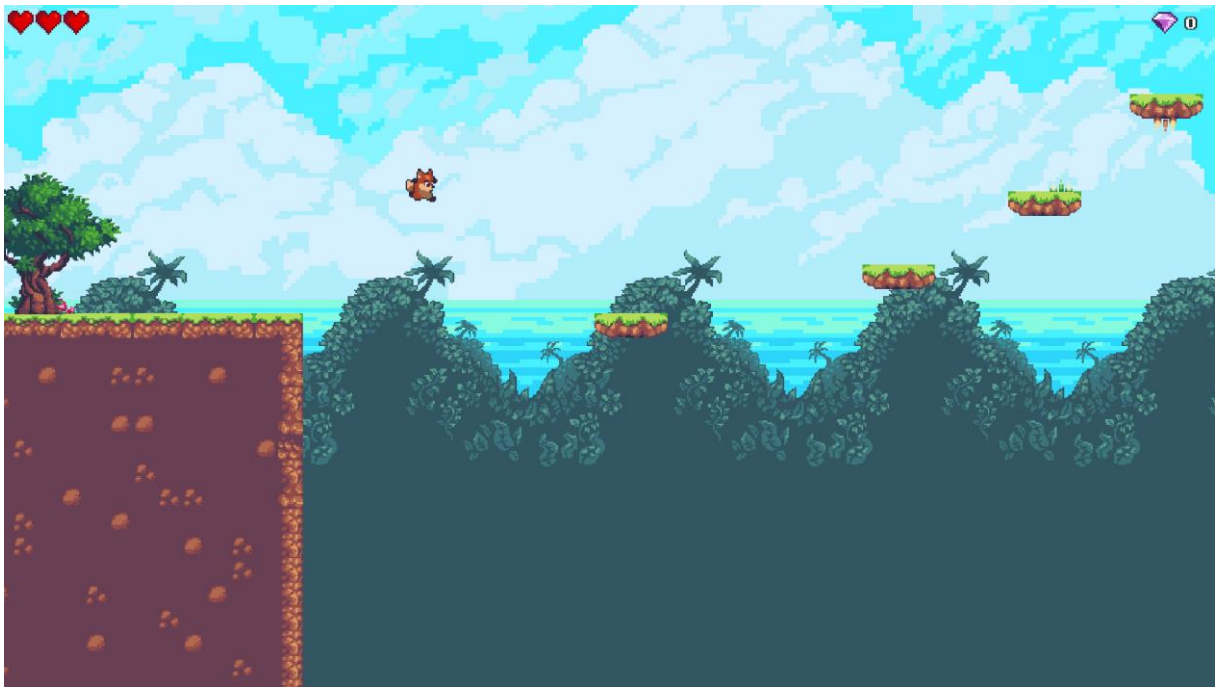
```
C# TestScript.cs  C# LevelManager.cs

using System.Collections;
using Level;
using NUnit.Framework;

using UnityEngine.TestTools;

public class TestScript
{
    // A Test behaves as an ordinary method
    [Test]
    public void CheckCollectedGems()
    {
        // Use the Assert class to test conditions
        var gemCounter:int = LevelManager.GemsCollected;
        Assert.Greater(gemCounter, 5);
    }

    // A UnityTest behaves like a coroutine in Play Mode. In Edit Mode you can use
    // `yield return null;` to skip a frame.
    [UnityTest]
    public IEnumerator TestScriptWithEnumeratorPasses()
    {
        // Use the Assert class to test conditions.
        // Use yield to skip a frame.
        yield return null;
    }
}
```



3.4 Some Pictures From Fox's Tale

Main Menu:

Beginning of the Level 1:



Boss Fight of Level 2:

Beginning of the Level 3: