MAZE RUNNER GAME

Report of Java Based Game

AUTHORS

ALAA SHEHAB
AHMED HUSSINE
BASSANT AHMED
BASSAM AIMAN

Table of Content

1. Project Description:

Maze Runner game is like the famous Pac-Man game, instead, here we will have a maze, and you will be the actor (maze runner), you should run through the maze to find your way out! But be careful not to hit a bomb, you can also destroy any obstacle you face by a weapon, the weapon will be fulfilled with limited ammo. While you are moving inside the maze, you should collect some gifts in your way. The end of the game is by reaching the gate of the maze, or if you died due to hitting a bomb.

2. Project Design:

2.1. Packaging Design

2.1.1. Game Engine package:

The game engine package contains 2 main packages the interface package and the controller package, which contain classes to generate the GUI and control the user movement along with the connection between back and front end.

2.1.2. Dynamic Objects package:

This package extends the Drawable package where each and every object in the game that would appear to the user would extend the drawable. Dynamic drawable contains all moving Objects as in Bullets, Characters and Monsters. It contains several classes and packages for setting the Objects classes.

2.1.3. Static Objects package:

This package extends the Drawable package where each and every object in the game that would appear to the user would extend the drawable. Static drawable contains all non-moving Objects as in Trees, Walls and gifts.

2.1.4. Game generation package:

The game generation package itself contains several packages to handle the game generation.
The Main 4 packages are the grid generation package where a 2D array of Bytes is generated, the level generation where 3 game levels are generated, the player generation and this includes generating player bullets and position in grid and finally the maze component generation which includes the bombs, trees, gifts and monsters as well.

2.2. Design Patterns

2.2.1. Singleton Design Pattern:

Singleton Design Pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

In our Code the Singleton importance is seen when creating an object of the character class, as only one player can play this game at a time and this player can only choose one character to play with during the game period.

2.2.2. Factory Design Pattern:

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

The factory design pattern is of great importance in our design it is being used 3 times in our project, once with the grid generation where the 2d Byte array is generated with numbers and these numbers are mapped to the classes using a factory, also in the game engine and with saving and loading.

2.2.3. Strategy Design Pattern:

In Strategy pattern, a class behavior or its algorithm can be changed at run time. This type of design pattern comes under behavior pattern.
In Strategy pattern, we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. The strategy object changes the executing algorithm of the context object.

The Strategy is used when loading and saving the game and the game is saved as xml file where each class contains different attributes and is saved accordingly.

2.2.4. Flyweight Design Pattern:

Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.
Flyweight pattern tries to reuse already existing similar kind objects by storing them and creates new object when no matching object is found

The Flyweight is used for image loading at runtime as loading a 31x31 grid with 961 image loaded separately at run time would

consume memory and time loading increases.
A class named shared image icon is used to load all images then a factory is called to get the image needed with no need to load all images separately during runtime.

2.2.5. Observer Design Pattern:

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
Observer pattern falls under behavioral pattern category, and it is being used in our design as the main core of all player and monster movements and the change in health, score and player direction for setting images and all related objects.

2.2.6. Builder Design Pattern:

Builder pattern builds a complex object using simple objects and using a step by step approach. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
A Builder class builds the final object step by step. This builder is independent of other objects

It is used in Level generation as there's more than one level and each has its own set of properties, therefore the builder pattern is used to build every level components.

2.2.7. Iterator Design Pattern:

This pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.
Iterator pattern falls under behavioral pattern category.

It is used with bullets class, where the bullets are considered a collection of objects connected with the player and we can iterate on each to use or add bullets.

2.3. Design Decisions

2.3.1. Maze Generation:

The Algorithm chosen to generate the maze is prim's Algorithm as although there was more than 10 algorithms found this was the easiest and generates a maze with only one 1 path between start and end points with several dead-end paths in between.

As for the component generation, components are generated in a 2d Byte array for memory efficiency where every cell component is mapped to a specific number and after grid generation all components are generated randomly in different areas and sections of the maze.

2.3.2. Data Structure Division:

Data Structures are divided into static and dynamic objects and both are under the drawable objects abstract class as they both have common attributes as setting image and position. The Dynamic Objects all have the same concept of moving in the grid whether dynamically or by user movement which allows them to have some common attributes as direction and ability to move to a specific cell.

3. UML Diagram:

All UML Diagrams are included in a separated file as due to the enormous amount of classes its vision won't be clear.

4. Game Features:

4.1. Player can choose between different players.

4.2. Player can choose between 3 levels of difficulties, where in each level the number of dragons and obstacles increases.

4.3. There are 2 types of bullets, ice and fire bullets and each with a certain amount of damage.

4.4. There are 2 types of dragons, fire and ice dragons and each has its specific health and damage to cause.

4.5. There are 3 types of Coins, bronze, silver and gold, each one increase the player score with a certain range.

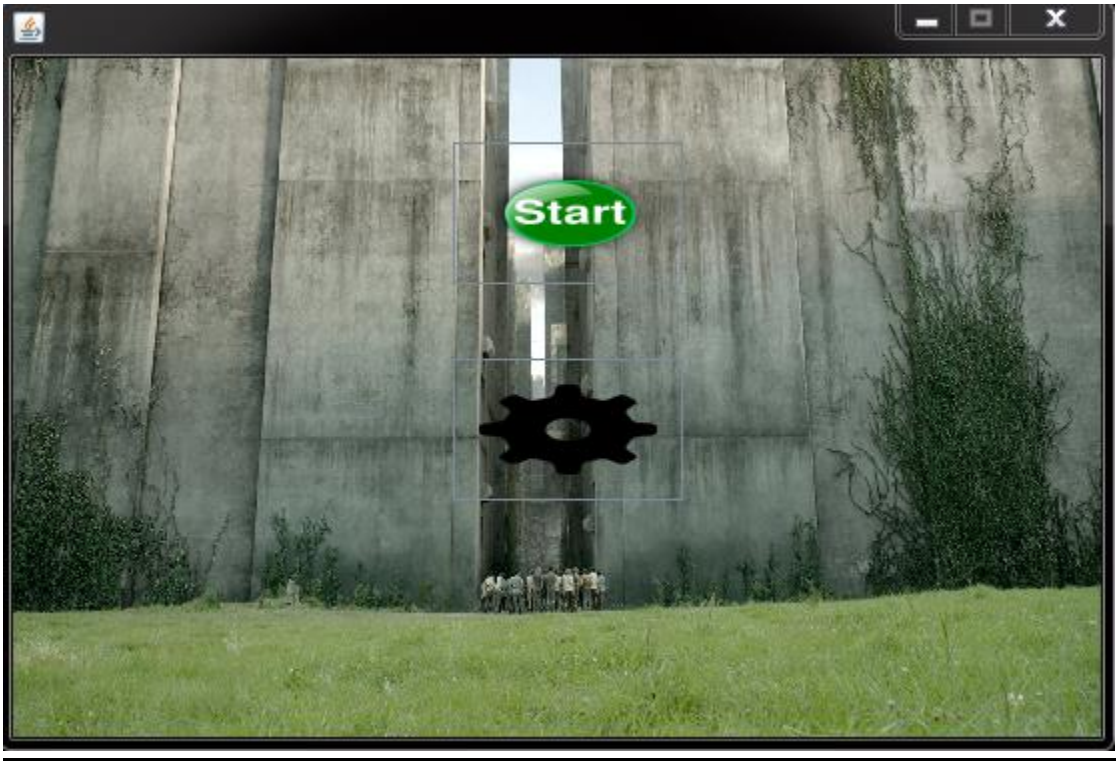
4.6. A background music plays while the game is running

4.7. A simplified mini-maze is presented to the user at the right end of the game interface to give the user the ability to check the whole maze at one glimpse.

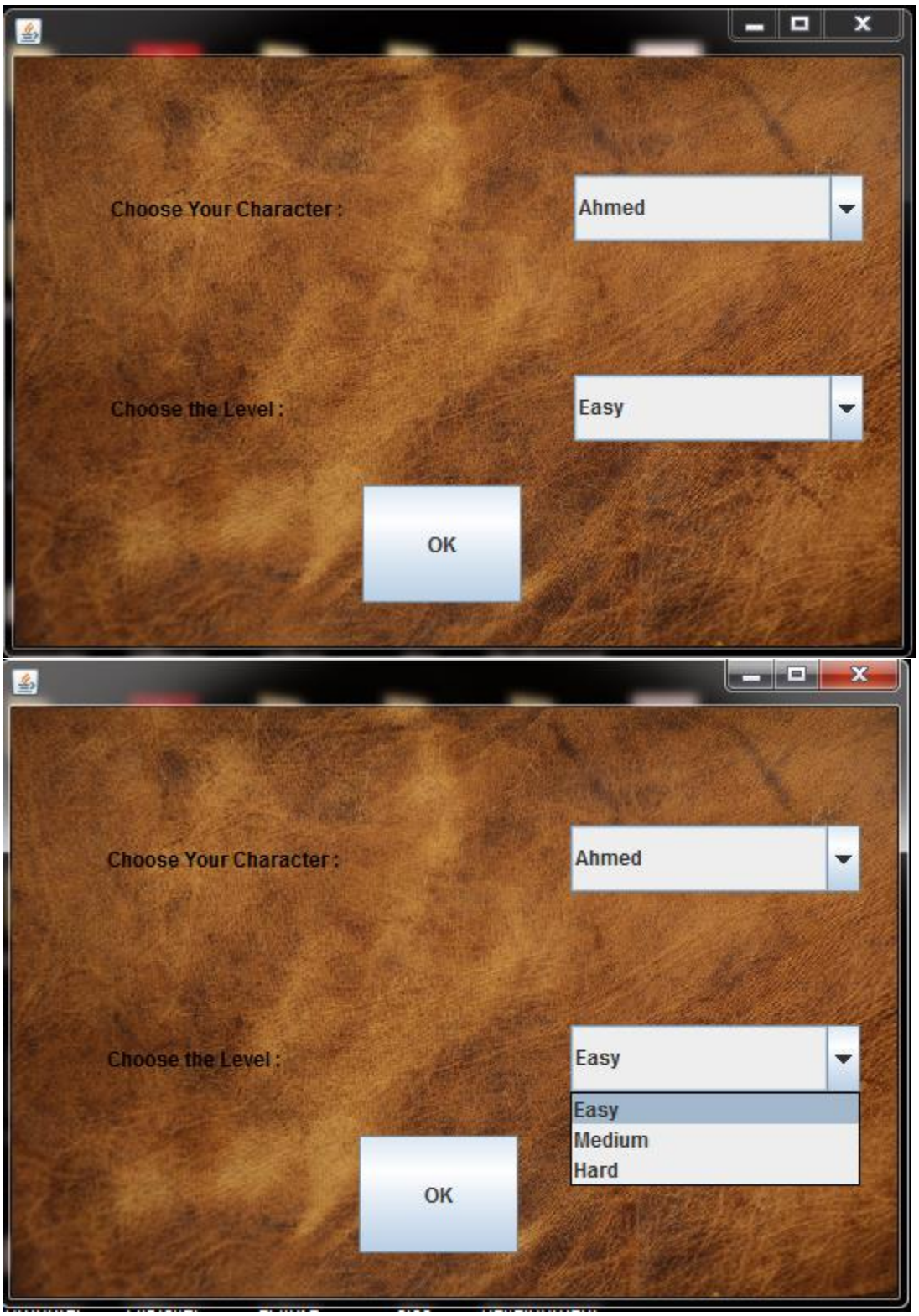
4.8. A right panel is set to allow player to monitor its score and health.

5. User Interface:

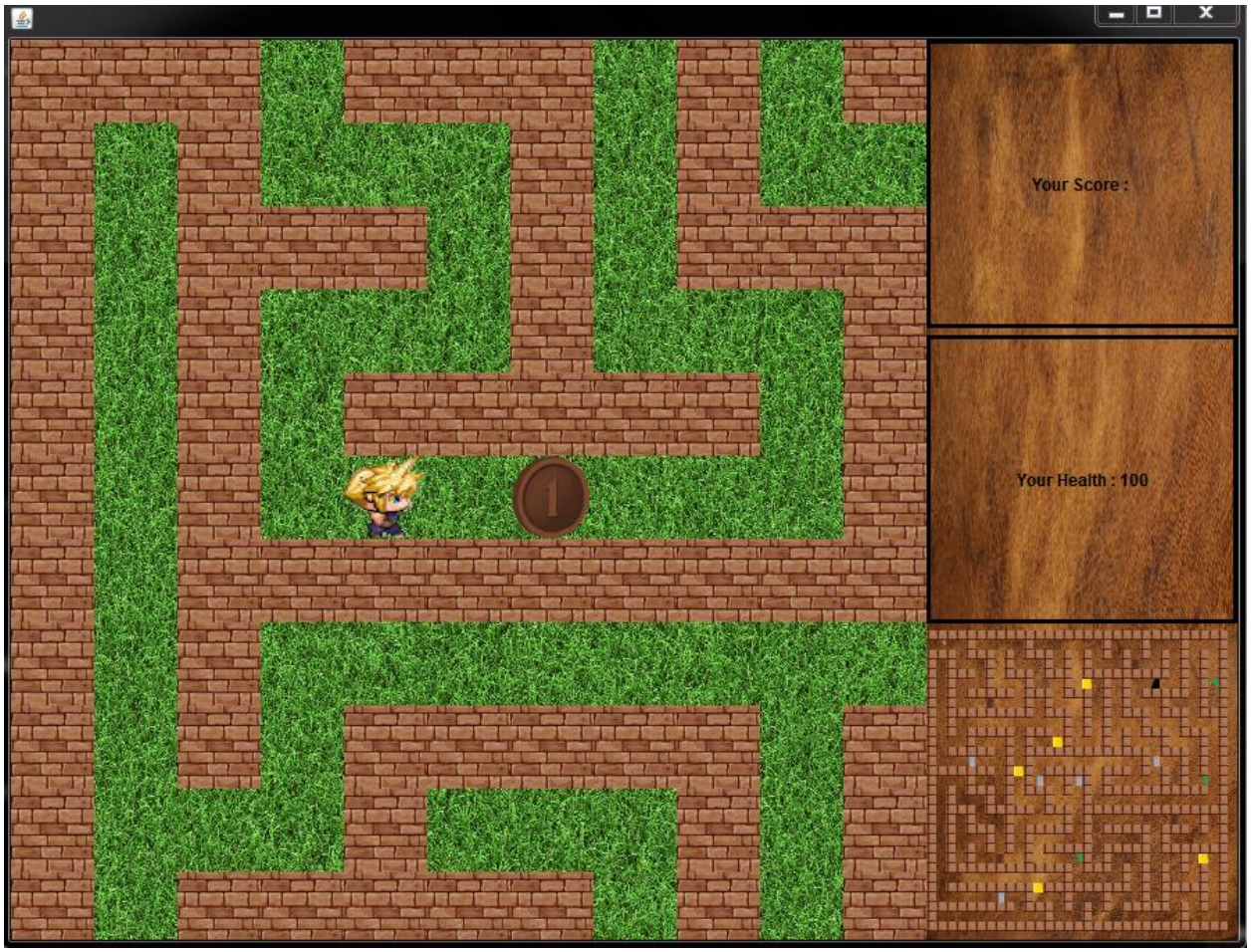
5.1. The game starts as seen below, with 2 buttons one to start the game and the other to change the game settings.



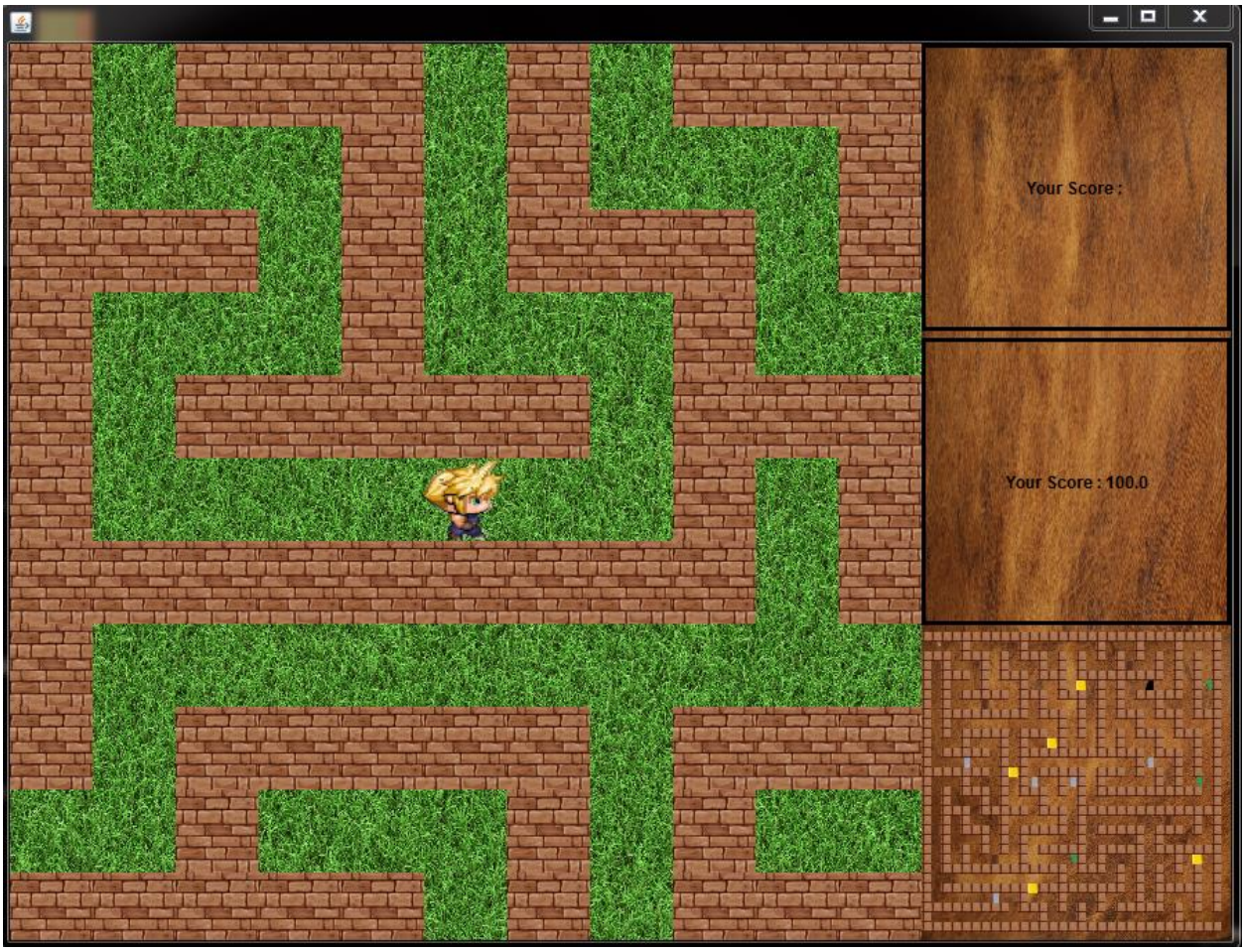
5.2) On pressing The settings button, the following screen appears where the player can change the character and the level of the game.



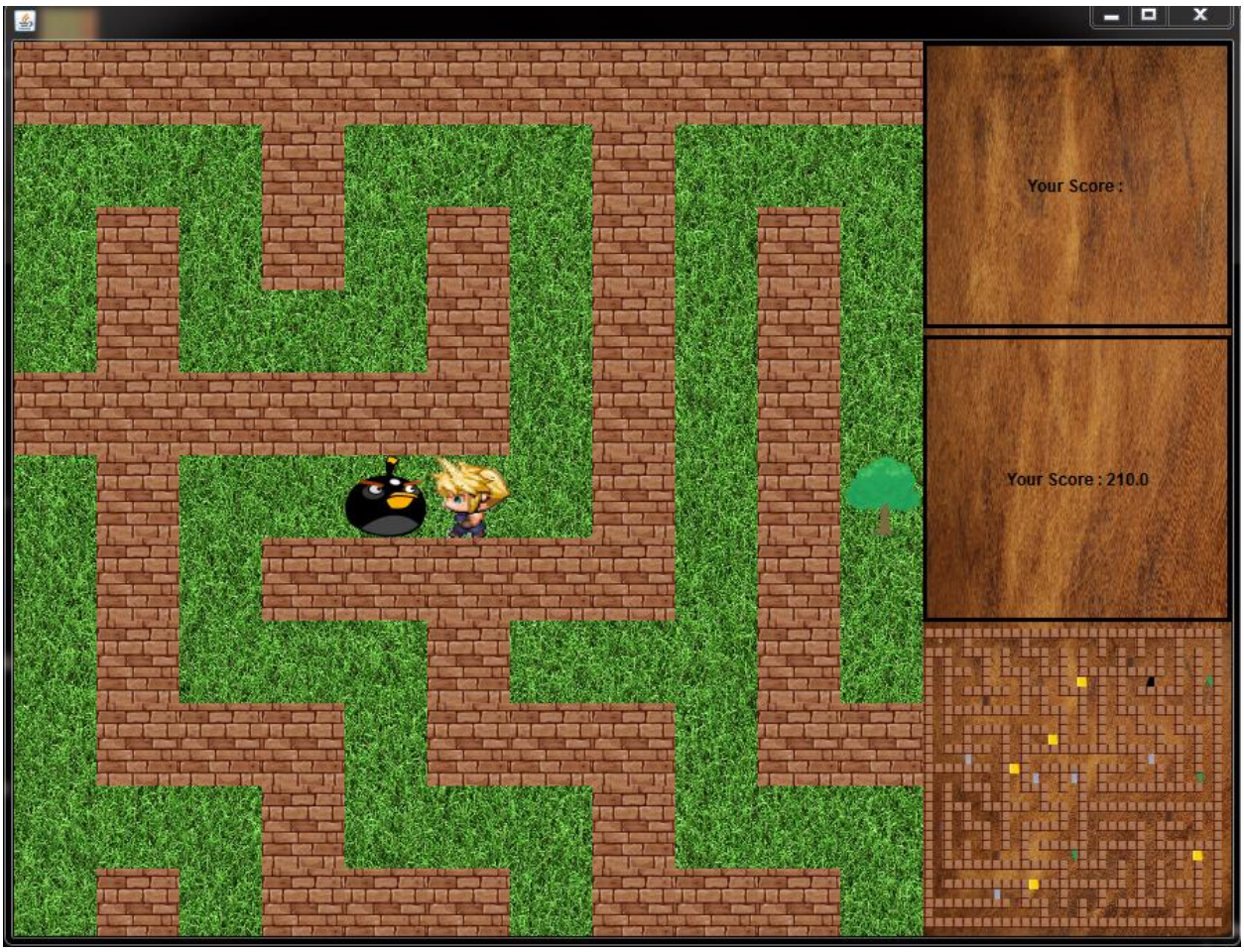
5.3) After user presses the start button, the maze appears as follows, Where the score and health are on the right position of the display and the plan view of the maze as well.



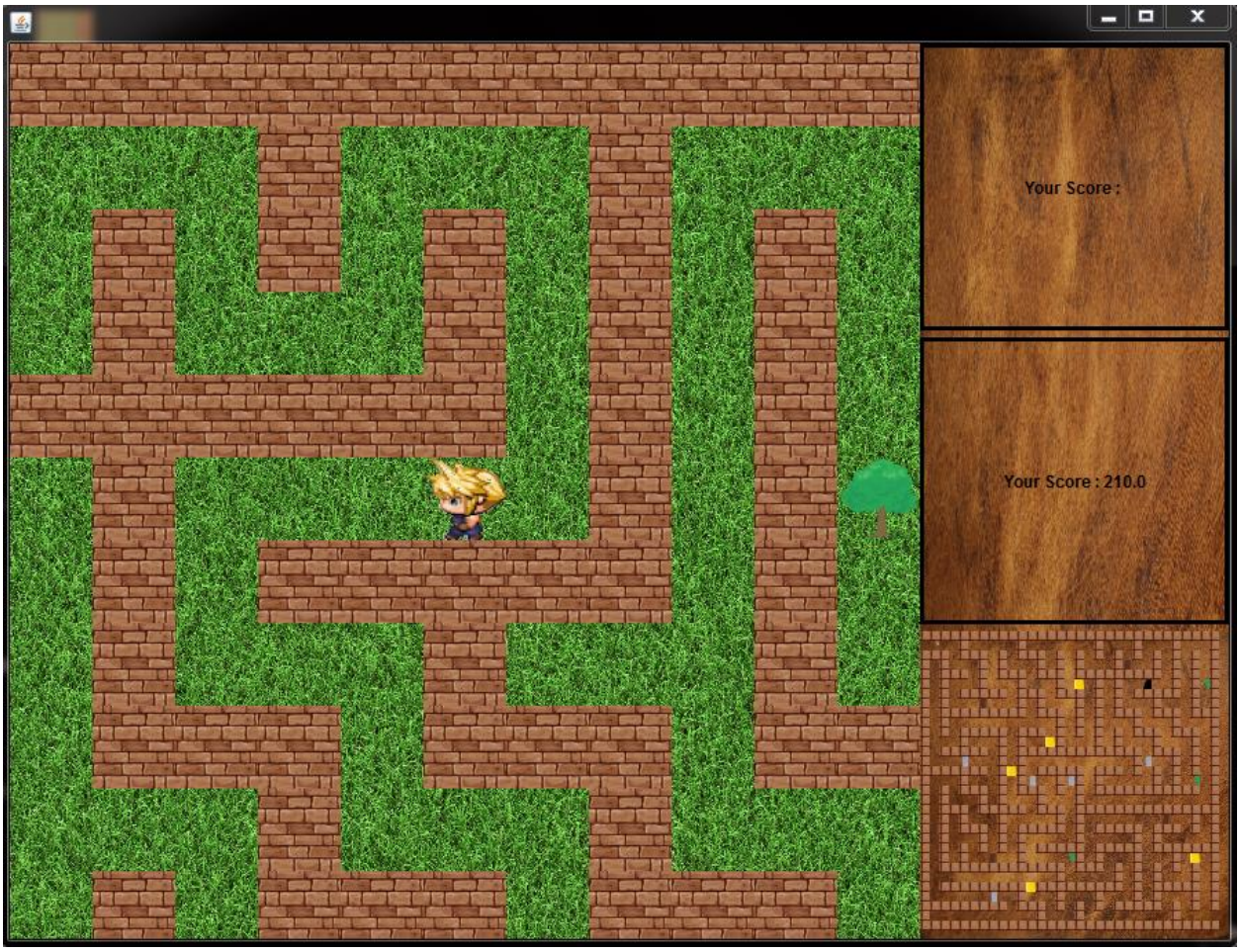
5.4) On Taking a coin the player score increases as shown in the second right box.



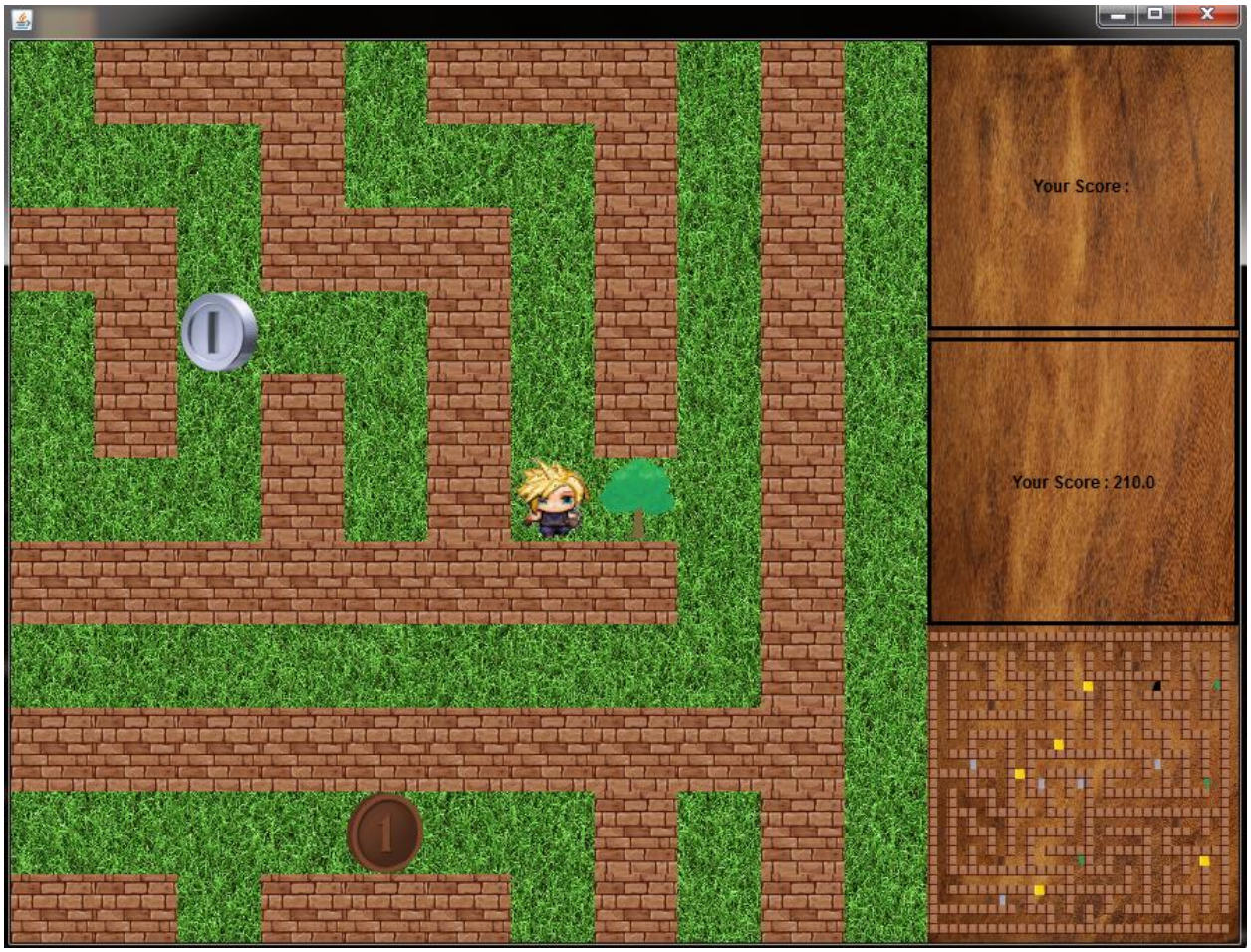
5.5) On Finding a bomb, the user can press space bar to fire bullets to remove the bomb from the way.



5.5) On firing the bullet the bomb is destroyed as shown.



5.6) Player can’t penetrate a tree but can destroy it by firing bullets to move in it’s path.



5.7) On firing the bullet the tree is destroyed and the path is opened for the user to path through.