CS319 Object-Oriented Software Engineering
Section 2
Group: 2.E

Design First Report

Maze Runner

Group members: **Ali Sabbagh, Mehmet Furkan Dogan, Umitcan Hasbioglu, Xheni Caka**

Course Instructor: **Ugur Dogrusoz**

Date: 25 Mar 2017

Contents

Introduction

"Maze Runner" is a game that combines two games (Pac-man and Bomberman) and offers many new features. The game will be implemented in Java and it is a desktop game. The game can be played as a single player or multiplayer. The game purpose is to reach the finish point of a sophisticated maze as soon as possible with the least possible attraction of patrols.

This report is about the high level design of the game system. In the first section, we are discussing the purpose of this system and what features will it bring better than any other similar implementation. The second following section is where we describe the goals of our design and all the tradeoffs and our picks. In that part, we are discussing four goals of our design based on the non-functional requirements. We will also discuss what our choices conflict with and why our decision was. This section is then followed by the architecture of our software. In this section we will be discussing the components and the architecture type we chose as well as detailed description of all the components services. Visual paradigm was used to create all the schematics.

Purpose of the system

The game consists of a Pac-man like moving object which goal is to finish the maze as soon as possible without being killed by the patrols which are continuously moving around and trying to stop the player from reaching the exit of the maze. The purpose of the game is to enjoy the player by competition and help to increase their problem solving skills by solving the maze as fast as possible. This game is different from other similar game because while trying to solve the maze, also the player have to keep an eye on his enemy and patrols. Another difference to other games is that the player can plant a bomb on his enemy side and give a damage to him.

Design Goals

Modifiability

Many of our functions are easily modifiable since the subsystem couplings are at minimum level . In our game, the subclasses do not have huge effects on their sub or parent classes. Three layer architecture helps to add new features or modifications to the game easily since one layer is dependent from the others and it leads to possible changes in future.

Portability

With the variety of the platforms and devices, creating a software which is going to have a stable performance in all these environments is not easy. In order to overcome this situation we are going to use Java language being that it runs on different platforms. So the game will be playable in all the platforms able to run Java Virtual Machine (JVM).

An advantage of writing the game in Java is that making it available for Android devices will be very easy being that the Android applications use Java as well. For our game this is easily realized. By changing the first layer, which is the user interface, it can easily turn to a Android application (Check Subsystem Decomposition for more).

Understandability

We aim our game to be playable by both adults and young people. Therefore we thought that in order to achieve that, we need to make game more understandable such that people need to understand how the game works. In game wise, we are planning to add many tutorials and make game components simple and clear as much as possible. In design/implementation wise, we decided to go with make it as less complicated as we can to make the project's progression as fast as possible.

Minimum Number of Errors

Errors make our programs unbearable. Therefore we have to go with least number of errors in our programs. Since players are unable to save their progress, a high implemented game with minimal to no errors is our aim. In the implementation phase, we will have to catch most of the exceptions and handle them in a proper way. In the aspect of design and game, we thought that, in order to face with and handle possible errors we should not add complicated features such that saving for the possibility of crashing game. Also in order to not allow users to crash the game, we are going to strictly use encapsulation on classes and save system's required files into .jar as we decided to use Java.

Tradeoffs

Understandability vs Complex Game and Features

Since the audience we aim to is of different ages and abilities, we had to sacrifice having a complex game in order to make the game easy to grasp and widen our range of audience. Complex games add constraints on who is going to be able to play the game, understand it and enjoy it. That is why our features are simple and not interconnected. Thus having a tutorial that shows images of game simple laws will help understand the game features at first try and will be sufficient.

Portability vs High Graphics and UI

In order to have bigger audience, we picked Java as the language to implement with, as Java is a portable language that works on any system that has Java Virtual Machine. In addition to that, we will make the game computations simple and will use basic 2D graphics in order to insure that the game works on any computer with minimal requirements of hardware.

Software Architecture

Subsystem Decomposition

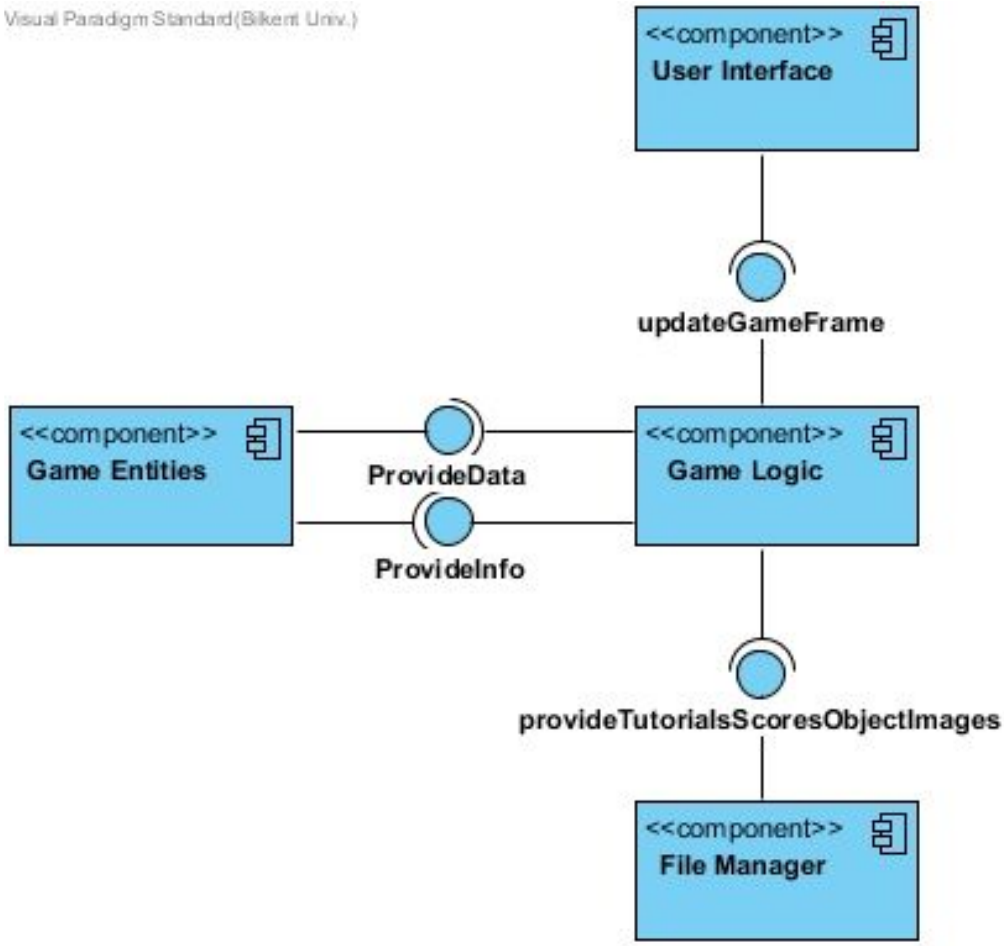


Figure 1- Components Diagram

The diagram represents a three layered architecture style of the MazeRunner.

Reasons for choosing this architecture

During the process of design, we discussed the architectural structure of our project and we decided that three layered architecture is better for us due to following reasons:

- This architectural style allow us to create low coupling since all classes have specific task to do.
- For future modifications, because of low coupling it is much is easier to modify and easier to debug since the classes are more dependent from each other architectural styles. Also, it allows that the game will be implemented easily on a new platform.
- It turns a more complex design into much easier one for our game and it helps us to satisfy our non-functional requirements which we discussed in the analysis report.

First Layer

The first layer consists of the User Interface. User requests are sent to the second layer and gets back what needs to be changed. This layer in the future can be easily changed to work for Android UI.

Second Layer

In the second layer the business logic is handled. Here we manage the data coming from the third layer and we send data to the first layer. Data is send back and forth among the Game Entities and the Game Logic.

Third Layer

In the third layer we have the data about the tutorials, high scores and the object images are held here. This layer interacts with the second layer giving data upon request.

Hardware/Software Mapping

Maze Runner is a Java based game. Thus it is playable on all the machines that support Java. Since we will develop our game on latest update of JRE, latest version of Java is recommended for users. Also, game will get input from user from keyboard with supported characters. Those characters will be ASCII characters since we want the game be playable across the world. Neither a working internet connection nor a high performance CPU/GPU is required for to game be played.

Persistent Data Management

Game will read data of images, high scores, maps from the disk and initialize the data no matter what a user does with the game, thus these are expected data whenever the game starts executing. In the writing to disk wise, If a user gets a high score where it is in the top 5 recorded datas to disk, user will get a chance to save that score, therefore writing to disk is also expected.
About the data organisation, for the maps, we are going to use 2D arrays and save them to the disk in a proper way, such as .txt files; for images, we are going to store them as .png format; for high scores, again we are going to use .txt files in a proper way.

Access Control and Security

The only aspect of the project that we have to ensure of security is the high scores part, other than that since all the users are able to use same functionality of the program, we do not really need a user authentication system.
In order to properly secure high scores part, we will encode the high scores in a such way that, any improper change will be detected by program, also we will be ensure that the names and the scores of top players will not be readable by a normal person, so that anyone who wants to cheat cannot go ahead and change those values.

Boundary Conditions

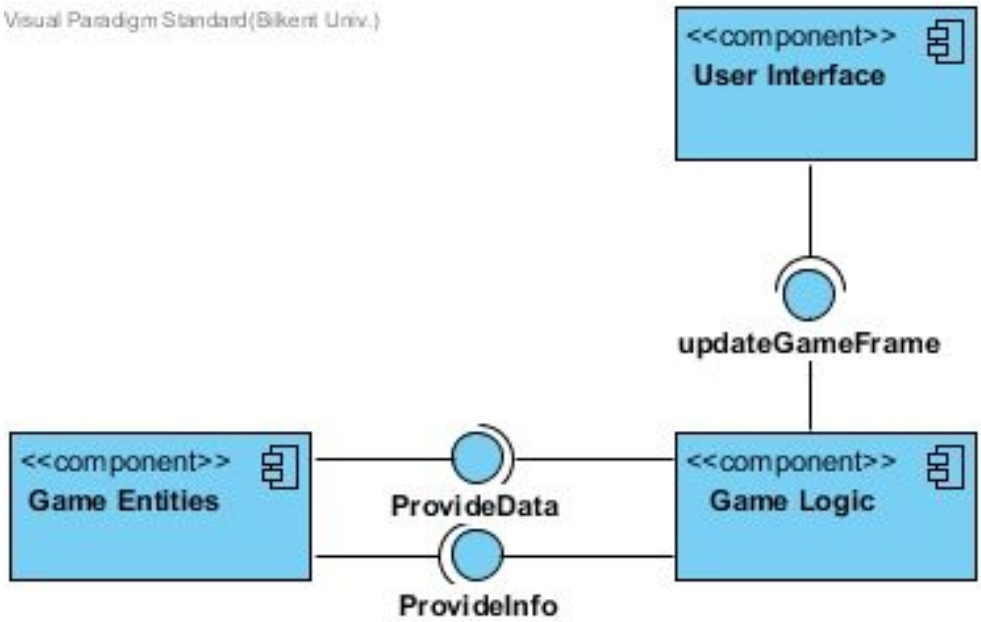
Since Maze Runner is an offline program, we will not have any data loss. But in the case of a security infraction, whereas the change of game data, or an invalid value for externally changed resources but used internally such as, changing the map in such a way that it will not be playable, will obviously cause errors, but the game itself will determine those by having certain checkpoints that if the data

cannot get through those, logic will not let the user to play the game, will pop up an error message and then will stop executing.

Subsystem Services

User Interface Component

This component contains what will directly interact with the player(s). It includes the main menu, leaderboard and tutorials windows, and the game board when playing. It listens to events according to user input through the keyboard (Choosing menu, playing the game, etc..) and passes them to the Game Logic Component for example: player chose to play single player, player pressed move left key) and gets back what needs to be updated or changed on screen.

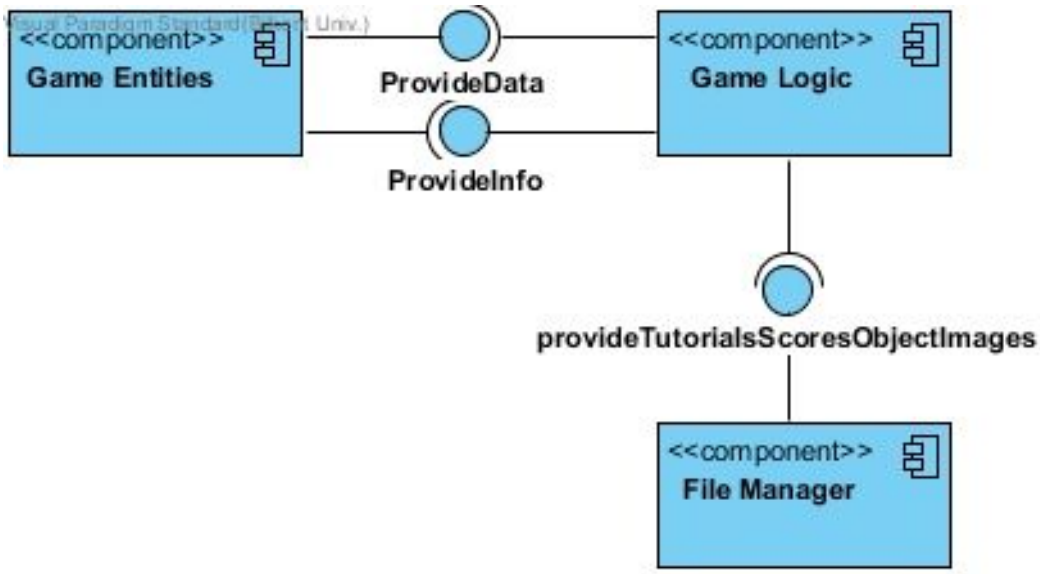


Game Entities Component

This component holds the basic parts that form the game. It consists of classes like: Pacman, Patrol, Bomb, Leaderboard, etc. It will be responsible to save the game data temporarily (while game is running). At the start it gets loaded with data through the Game Logic Component, and sends required data back to Game Logic when asked to. As the game proceeds it gets changes needed from Game Logic and updates properties accordingly.

Game Logic Component

This component is the dynamic component. It does all the logic and computations of the game and connects other components together. At the start of the game it gets data from the File Manager Component and sends it to required classes in Game Entities. While game is running, it receives user input from User Interface component and acts accordingly. For example changing objects' locations according to moving keys pressed, creating bombs when plant bomb key is pressed, etc. It has timers running to check collisions and game ending scenarios.



File Manager Component

This component is responsible for connecting Game Logic with the real data saved on hard disk. It can read and write to data. It sends scores when GL needs to create a Leaderboard, tutorials images when GL needs to create Tutorial, and board mazes and real game objects' images when GL is initializing the game. It receives new scores from GL in order to make changes to the table of top 5 scores.