

# CS319 PROJECT DESIGN REPORT

# PART 1

## **GROUP G**

### **ERKUT ALAKUŞ**

### **DOĞANCAN DEMİRTAŞ**

### **GÜNÇE KALYONCU**

### **CANSU TÜZMEN**

**TABLE OF CONTENT**

1. Introduction3
   1. Purpose of the System3
   2. Design Goals3
2. Software Architecture4
   1. Subsystem Decompostion4
   2. Hardware/Software Mapping……………………………………………………………5
   3. Persistent Data Management…………………………………………………………….6
   4. Access Control and Security…………………………………………………………….6
   5. Boundry Conditions……………………………………………………………………..6
3. **Subsystem Services7**
4. **Introduction**
   1. **Purpose of the System**

Digger Unlimited is a basic arcade game and it main aim is entertaining players with a good design. Thus, the levels are sorted from easy to hard and each of them consists of some bonuses changing the course of the game to increase the attraction. It will also have well-designed interface so it is easy to use and learn its tools.

* 1. **Design Goals**

**Response time**

Since Digger Unlimited is an arcade game, the response time is very important so

the average respond time will be 0.5 second in order not to destract the player’s attention or interest.

**Extensibility**

In order to make game more exciting and intersting, it could be added new features like new maps, bonuses, characters and concepts over time because its object oriented structure enables them easiliy**.**

**Usability**

The system will have user-friendly interface to make the usage more easy . Thus, it provides players well designed menus and a “help” option to learn the goal of the game and its tools.

**Trade-Off:**

**Space vs. speed**

Speed is more important than space, because the players should collect coins without being caughted by the monsters so the system should respond in seconds to make the game more entertaining and interesting. There will be also some bonuses to change course of game for 30 seconds so system should respond immediately after the player collects it.

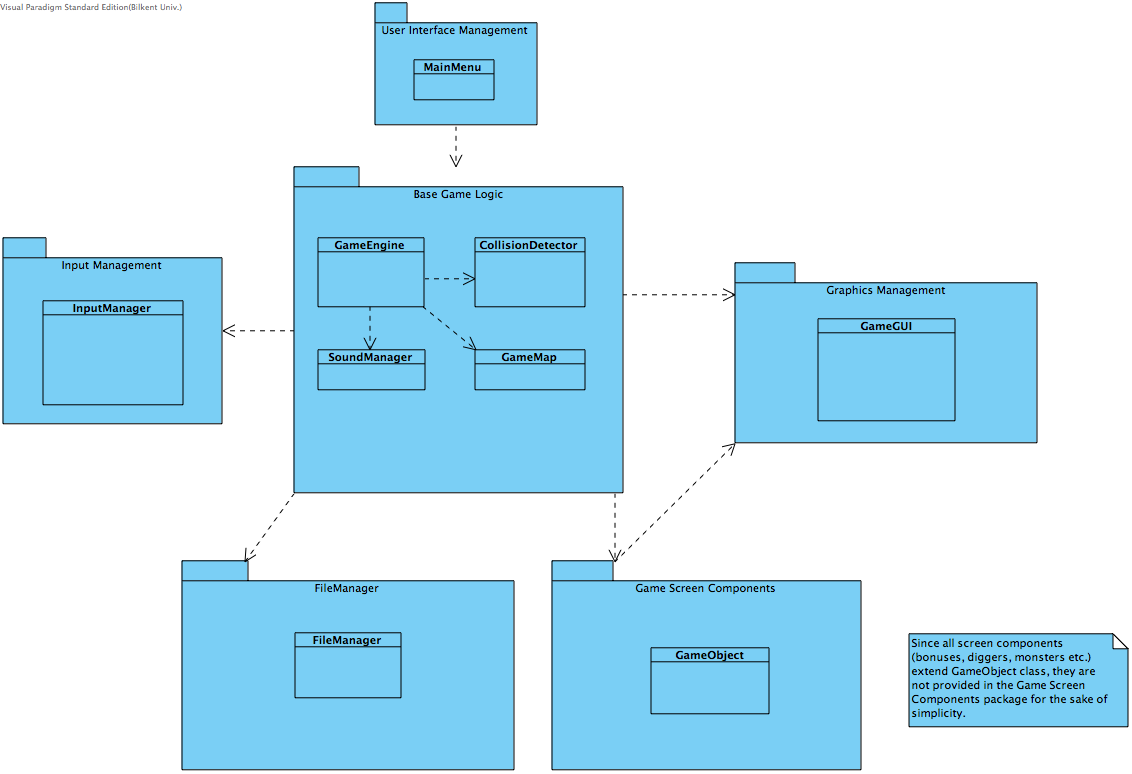
**Usability vs. Functionality**

Usablity is more important than functionality so in order to reach wide range of customers, the system interface should not be prepared too complex.

**2. Software Architecture**

**2.1 Subsystem Decomposition**

We choose 3-tier architectural style although it is widely used in DBMS applications. Since our current design will be constructed on 5 main stages –**Engine**, **Graphical User Interface**, **Menu**, **Input Manager** and **Game Objects**, we decided to represent these stages in 3-tier architecture. In the below diagram, interaction of 3 layers can be seen:

****

*Figure 1- Subsystem Decomposition*

## **2.2. Hardware / Software Mapping**

"Digger Unlimited" will require a JRE or JDK to be executed since it is developed in Java language. As hardware requirements, game requires a keyboard to give input and interact with the system. Minimum system requirements will be minimal; a basic computer with an operating system should be sufficient to run the game. Input are only 4 controls, which is nothing for today's operating computers. We will also have stored information on .txt files, so system should support .txt file format. It does not require any Internet connection.

## **2.3. Persistent Data Management**

“The Digger Unlimited” does not require a complex database, because we hold small amount of information and it can be done within client’s hard disk drive, which will include text and sound files and images. Game Map will be included in the folders as a text file, and before every new stage, game map will be created from these text files. There will be five stages that means five text files. If system would unable to open text file, game will not start and ask user for a restart the game or repair the game files. Sound effects and object images will also be in the client’s hard disk drive already with simple formats, this helps user to add his own sounds to the game and relieve us from having a database.

## **2.4. Access Control and Security**

"Digger Unlimited" does not have any user authentication and there is also no database to store this information. Only "GameEngine" has access to the file system and to prevent changes on game logic, necessary security measurements were satisfied with the critical decision variables to protect the flow of information. We gave input, image and sound controls to another managers; specific classes for jobs to increase the robustness of the game and at the same time reduce the complexity of "GameEngine". "Digger Unlimited" will not need any kind of network connection, which decreases our job to prevent in case of security vulnerabilities we might face.

## **2.5. Boundary Conditions**

**Initialization**

Due to the “Digger Unlimited” is a desktop application; it does not require an install. Users only need to have JRE or JDK than can run .jar file of the game.

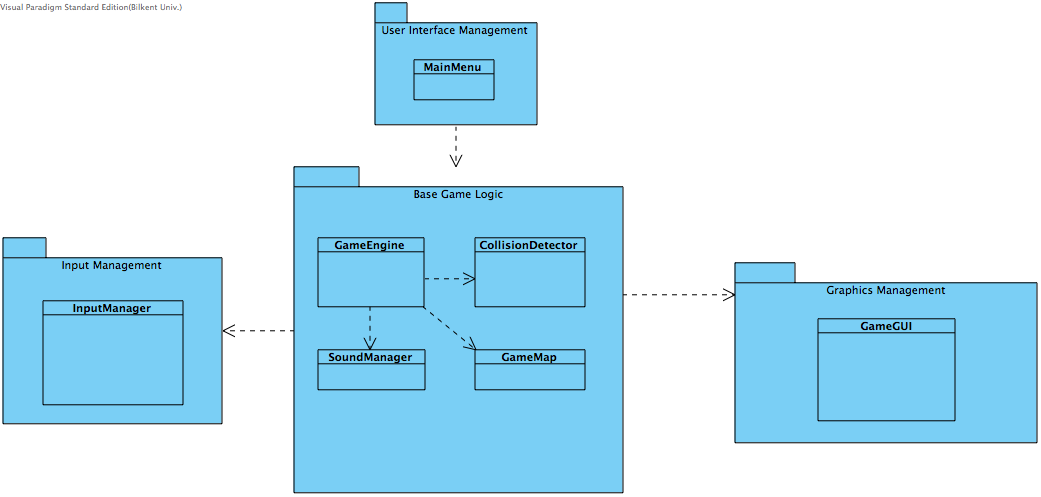
**Termination**

It is possible to terminate the game by clicking “X” button on the upper right side of the game screen. For in-game termination, user could click “Exit” in the main menu, or pressing “Esc” to open pause menu and return “Main Menu” by selecting it from the pause menu than perform exit operation.

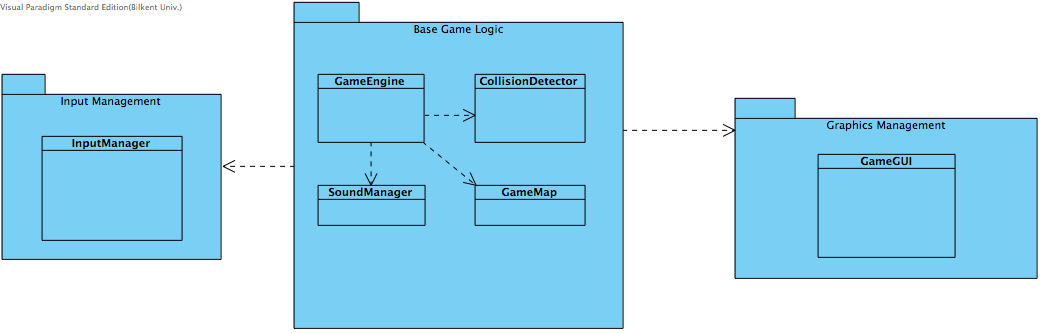
**Error**

If system encounters an error that prevent game resources to load, such as maps, sounds and images, “Digger Unlimited” will close itself and return an error message to user and ask for a reopen or repair. An error based on a performance issue, which is unlikely, will cause loss of current data.

**3. Subsystem Decomposition**

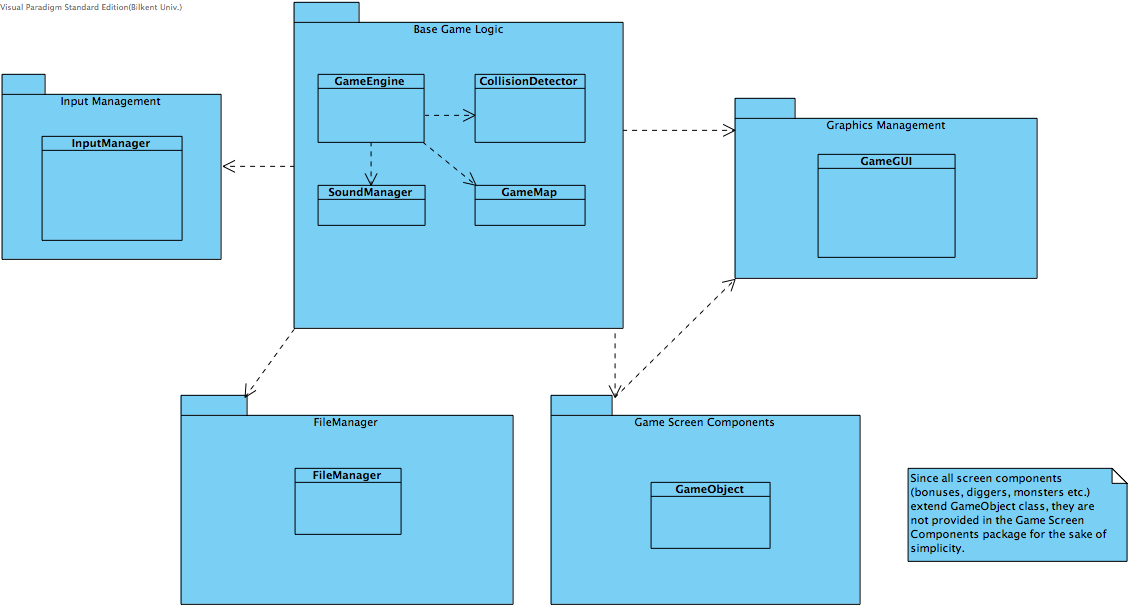
Since user will first interact with MainMenu, it is assigned into the presentation layer. Depending on user’s choice game concept –characters- will be changed. Game theme and sound options will be passed to the GameEngine class, which is the main logic of the game, from MainMenu.

*Figure 2– Interaction of Presentation and Application Layers*

****

*Figure 3 – Application Layer*

As indicated above, GameEngine is the logic of the game. Initialization of game map, input manager and collision detector, GameMap, InputManager and CollisionDetector classes, is done here. Base Game Logic, Input Management and Graphic Management packages are related with each other such that InputManager class provides every user input to the GameEngine class. Depending on these inputs GameEngine updates the instance of GameMap class and triggers GameGUI class in every iteration to draw the map again.

****

*Figure 4 – Interaction of Application and Data Layers*

File Management and Game Screen Components packages are considered as Data Layer. All I/O is handled with these packages. FileManager class in the File Management package is responsible to read game map contents from a text file. GameMap and GameObject instances are created from this text file and provided to GameEngine and Game GUI classes.