Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

Project short-name: Mr.&Mrs. Pac-Man Ext.

Design Report

Ecem İlgün, Aziz Osman Kozhan, Başak Şevval Ekici, Talha Şeker

Supervisor: Bora Güngören

Progress Report

Oct 21, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object Oriented Software Engineering Project course CS319/2.

Contents

1. Introduction1

**1.1 Purpose of the System2**

**1.2 Design Goals2**

1.2.1 End User Criteria2

1.2.2 Maintenance Criteria2

1.2.3 Trade-offs2

2. Software Architecture1

**2.1 Subsystem Decomposition2**

**2.2 Hardware/Software Mapping2**

**2.3 Persistent Data Management3**

**2.4 Access Control and Security4**

**2.5 Boundary Conditions7**

3. Subsystem Services1

**3.1 GUI Layer19**

**3.2 Game Logic Layer20**

**3.3 Data Layer20**

4. Low Level Design1

**4.1 Object Design Trade-offs2**

**4.2 Final Object Design2**

**4.3 Packages3**

4.3.1 User Interface Package19

4.3.2 Game Logic Package20

4.3.3 Data Package20

**4.4 Class Interfaces4**

**1. Introduction**

**1.1 Purpose of the System**

Mr.&Mrs. Pacman Extended is 2D arcade game which is based on classical Pac-Man game. Therefore, the main purpose of the game is to eat all food on the map without colliding with ghosts. In addition to that, our game has new features such as new food, shield option and creating your own map. By these new features, interest of players will be sustained. This system will be constructed by considering object-oriented design techniques

**1.2 Design Goals**

The main purpose of a game is to entertain player(s). To do that, throughout the project, we have to focus on details which do not directly have an effect on the system. Moreover, with the help of object-oriented design, we aim making a project which is easy to extend. This part details the design goals of the system such as end user criteria, maintenance criteria, performance criteria and trade-offs.

1.2.1 End User Criteria

Usability: From user’s perspective, a game should be easy to use and learn. To provide this, we do not change commonly used keyboard buttons to move Pac-Mans. First user plays with arrows and other one plays with W, A, S, D combination. Additionally, the purpose is to avoid confusing GUI design; therefore, GUI will have simple design. On the main menu, there is help section to adapt user to new features. User can realize what to do with suitable icons and prompts. Shield, pause and help panel will appear as pop-up for simplicity. Therefore, user does not have to pass windows each time.

Performance: For a good game experience, we try to overcome input lags and low fps. For example, user can hit a ghost due to keyboard input lag and this is frustrating. To achieve this, we focus on writing as possible as efficient code. Not to experience rendering problems we prefer 2D game.

1.2.2 Maintenance Criteria

Extensibility: Design of the game allows us to add new features and components to continue interest of users. In time, new shields, new food or new ghosts can be added. Now, two players can play the game locally; however, in the future, number of players can be incremented. All of these are beneficial for player experience.

Modifiability: The system will be designed by considering into multi layered architecture. Basically, three-tier architecture will be used. Presentation tier includes user interface parts; logic tier makes logical decisions and calculations and data tier stores and fetches data.

This architecture allows us to modify easily a subsystem without affecting any other subsystem.

Reusability: Subsystems can be used in other games or similar systems because the system is designed as multitier architecture. Therefore, subsystems can carry out their functionality with other subsystems.

Portability: We implement the game in Java. Java provides Java Virtual Machine (JVM) which can be runnable on many operating systems. This feature of Java increases portability of the game.

1.2.3 Trade-Offs

Portability – Performance & Memory: In this project, we will use Java Programming which is runnable on many operating systems. This situation has positive effect on portability of the game. However, to do this, Java has to call interpreter named Java Virtual Machine. This fact causes performance decrement. Comparing to C++, Java uses more resources and makes less function with more code.

Functionality – Usability: At first glance, new features decrease usability because user encounters something different and to get used to new features take time. Therefore, there will not be too much functionality which causes confusion to decrease orientation time. However, without new features the game might be quite boring.

Reusability – Performance: To increase reusability multitier architecture will be followed. However, this design increases function calls; therefore, multitier architecture will have negative effect on system performance.

**2. Software Architecture**

In this part, we will examine composition of our system. As it has been mentioned before, the system will be constructed by smaller subsystems. In this way, we can prevent huge chunk of codes and codes can be ordered systematically. In our project, we will use Model View Controller (MVC) design which is suitable for this kind of game project.

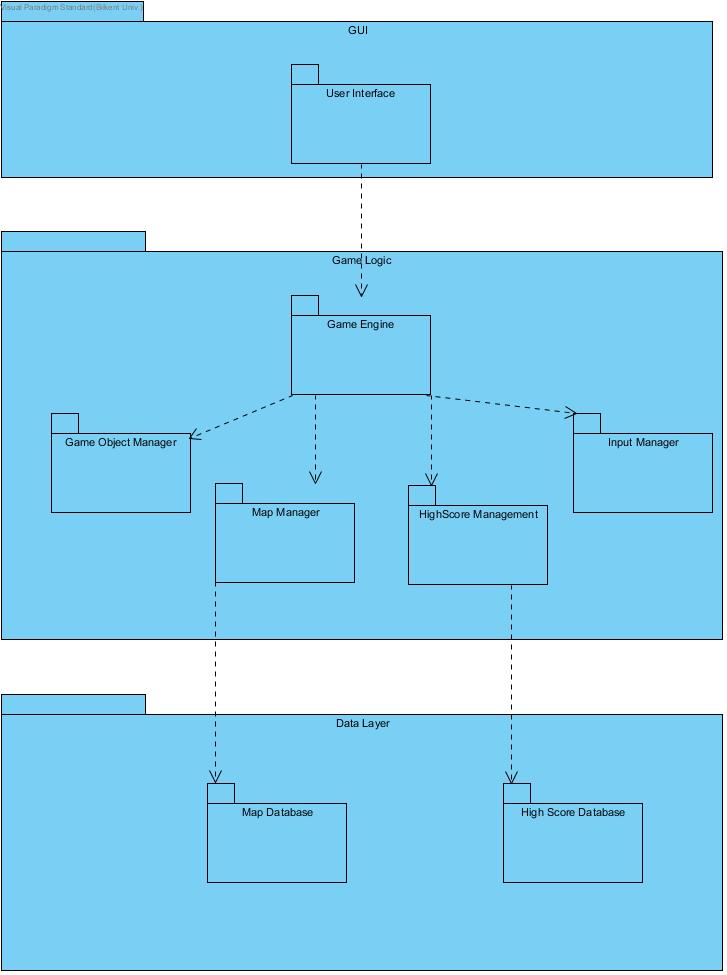
**2.1 Subsystem Decomposition**

In this section, we will demonstrate subsystems roughly. Detailed information about subsystems is in third chapter. While deciding subsystems, we aim loosely coupling each system and followed three-tier architectural style. Throughout the project, some code blocks will need to be changed; therefore, we do not couple subsystems strictly. Otherwise, all subsystems are affected due to a change in one subsystem. Additionally, such a design is beneficial to develop the game in the future. Briefly main concerns are modifiability and extensibility. Our system consists of three subsystems: GUI Layer, Game Logic Layer and Data Layer

- GUI Layer will have classes which is responsible for providing user interface and having interaction with user.

- Game Logic Layer will carry out all decisions and calculations.

- Data Layer will be responsible for protecting data about high scores and maps.



**2.2 Hardware/Software Mapping**

Our game will be implemented by using Java programming language. All user interface and graphics components will be constructed by using Java libraries. Most computers can run this game because it does not contain any 3D rendering and intense calculations. Our game requires a bit of memory because in data tier layer the system will make some operations such as saving high scores, map and game. To give inputs, basic mouse and keyboard will be enough. Due to these simple hardware and software requirements, this game is runnable in many computers and in many operating systems such as Windows, Linux and mac.

**2.3 Persistent Data Management**

In our game high scores, created map and current position of a game level are persistent identity that needs to be stored after each game. We will use local database for all of them because our game will not have any network connection. It means that user can only see high scores which obtained his/her own computer. By using file system, data will be written into txt. files.

**2.4 Access Control and Security**

Our game does not require any authentication or account because all data stored locally and do not require any private information from user. Therefore, the project does not require any access control or security.

**2.5 Boundary Conditions**

Execution

Mr.&Mrs. Pac-Man will be launched by double clicking .jar file of the game. The game does not need any special software other than Java Runtime Environment.

Termination

Scenario 1: Player clicks exit button on the pause menu.

Scenario 2: The game can be terminated by using Alt+f4 combination or clicking X icon on the window (for Windows).

Failure

Scenario 1: If there is an error with database system, a prompt will be appeared and user will not be able to save what s/he wants to save.

Scenario 2: If electricity or computer problem happen, the game will not give any opportunity to save the last situation. The user will lose all progress.

**3. Subsystem Services**

In this section, there are detailed expressions of subsystems.

**3.1 GUI Layer**

GUI Layer contains all user interface components. These components will be contained in user interface package. In this package, there will be main menu screen, help screen, pause menu screen, high scores screen, create map screen, game over screen, game screen, shield panel and name request screen.

If user clicks play game, game screen will be activated and GUI Layer and Logic Layer will communicate with each other to carry out logic. If first or second level is achieved, shield panel will appear as pop-up and again these two layers interact. If user saves a map, Logic Layer interacts with Data Layer to store the map.

**3.2 Game Logic Layer**

**//SOME TEXT HERE**

**3.3 Data Layer**

Data Layer contains high scores database, map database and current game database. Map database saves created maps and names. High scores database saves high scores and names. Current game database saves last situation of a saved game.

//CHAPTER 4 SPACE DESPOT DESIGN REPORT PART2’DEN BAKILABİLİR. BAŞLIKLAR ONA GÖRE AYARLANDI.

**4. Low Level Design**

**4.1 Object Design Trade-Offs**

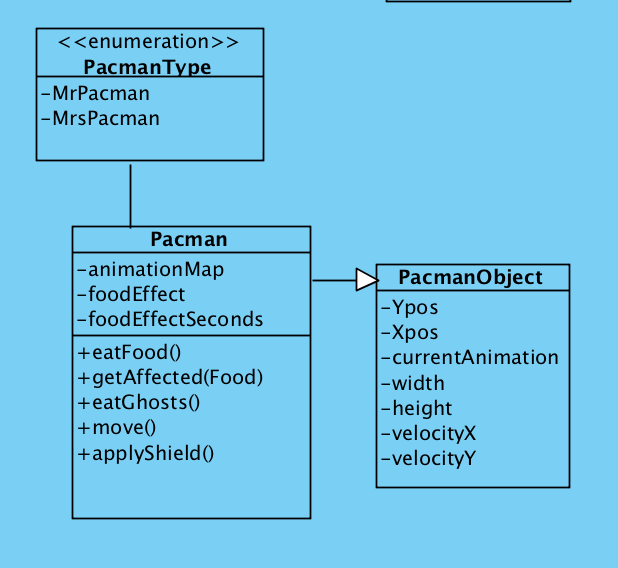
Designing the objects need some principles applied in order to get rid of the possible bugs in the program and also make the program in the most efficient way. In game, we design the objects in a resuable way to deal the complexity. We also keep the data background in an uncompressed form and this leads to a space-time tradeoff.

**4.1.1 Reusability of the Objects**

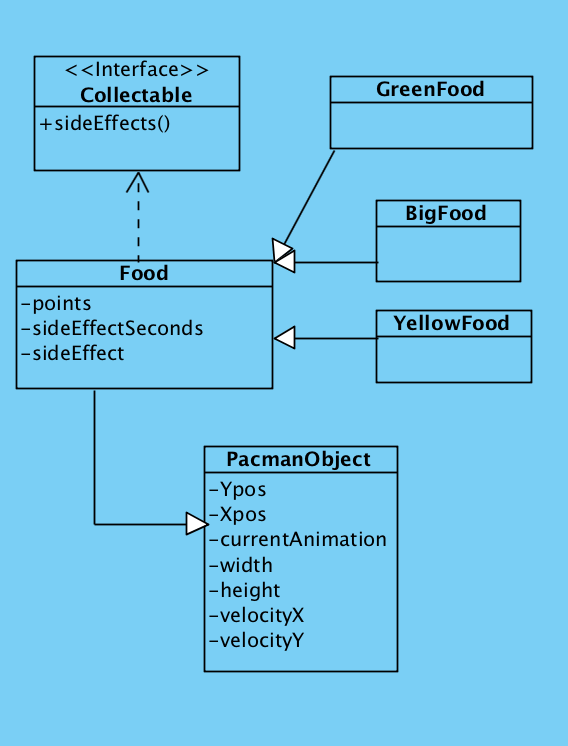
The program is divided into three layers, which are complex subsystems itselves. The subsystems are connected in suitable forms and there is a hierarchicy in between the classes.

**4.1.1.1 Inheritance**

Using inheritance between classes had reduced the complexity of the program and the amount of code would have written. Extending helps to have the main properties and abilities of the super class and use them in another object, which is based on the super class but formed in a more complex and specific object. For example in the game, the game panel designed in a grid form so that every object needs to have coordinates, an image of the character etc. So we created a Pacman Object that will be a base to every character object in the game. For example Pacman is extended to the Pacman Object, but has characteristic methods and properties (Figure 1). Similarly, in Figure 2, it can be seen that Food extends to Pacman Object and some other types of Food, which are Green, Yellow or Big Food, also extend to the Food class.



*Figure 1*

**

*Figure 2*

**4.1.1.2 Interface**

To have a reusable object design, interfaces are an essential component of the system. It helps decoupling separate components of the program and provides a polymorphic behavior. The usage of interfaces in our program is good for breaking up the complex designs. Although Java interfaces are slower and more limited than the other ones, using interface clearifies the dependencies between the classes of the game. In figure 2, it can be seen that the class Food uses an interface Collectable, which is also used by the Food-extended classes such as Yellow, Green or Big Food.

**4.1.1.3 Polymorphism**

Polymorphism is another important object oriented programming concept that we had used in our game. It allows the game objects to take on many forms. We used polymorphism in our project and by overriding the methods of parent class we were able to call the same method for all children of that class.

**4.1.2 Space-Time TradeOffs**

Data storage brings some problems with itself. In our program, the storage method caused the tradeoff. We store the data of the game in an uncompressed way, which takes more space. But by not compressing data, the program does not run the decompressing algorithm and the program runs in less time, which is more practical in our case. To reduce the transmission time and so the cost at the expense of CPU time to perform the compression and decompression, we used the uncompressed form of the data.

**4.2 Final Object Design**

**//TALHAYLA YAPTIĞIMIZ DİAGRAM**

**4.3 Packages**

4.3.1 User Interface Package

4.3.2 Game Logic Package

4.3.3 Data Package

**4.4 Class Interfaces**