Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

Project Short-Name: Space Despot

Design Report

Cihangir Mercan, Çağdaş Han Yılmaz, Fırat Sivrikaya, Gökçe Şakir Özyurt

Supervisor: Prof. Dr. Uğur Doğrusöz

Progress Report
Nov 11, 2016

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.

Contents

Design Report

Project short-name: Space Despot

1 Introduction

1.1 Purpose of the system

Purpose of Space Despot is to develop a game that will entertain the player by using our knowledge that we learned from our department. We are inspired by the classics Space Impact and Asteroids [1] [2]. Developing this system will help us to understand the notion of object oriented software, enhance our current skills on Java and help us to practice the work we will be doing after we graduate.

1.2 Design goals

- Game and its development will include the principles we learned so far from our department.

- Optimizing the system will be one of our most important goals. Because, players doesn't want to play a game which freezes even though it doesn't require server connection during the game play. Game elements should enter or exit the screen without any visible errors or freezes in the game. Every action that takes place in the system mustn't affect game play experience of player. We must also decrease the input lag as much as possible.

- We will try to design the best user interface and graphics possible with the current Java libraries. User interface will be user friendly and good looking. We will try to design the images of objects

through Adobe Photoshop to increase the user's game play experience.

- Space Despot will be expendable, in terms of its content, mechanics, interface and graphics. Using Object Oriented approach it is easier to enhance the system over time by doing reverse-engineering and then adding new features or enhancing the existing ones.

- Since we are developing a game, we should also handle the game design, not in computer science context, but in context of Psychology. While designing the game, we will try to obey the key aspects of the game design as much as possible. We will try to implement the four essential characteristics of the game design and Flow Theory, to provide a better game play experience [3] [4].

Trade Offs

- Performance vs. Memory: In order to increase the game play experience, user must not see any kinds of graphical or mechanical (system's) errors. In order to prevent those errors and increase performance, we won't be concerning on keeping the memory requirements low. We must check for collisions, calculate damages and display the game efficiently and error-free, to do this we will use memory a lot more.

- Functionality vs. Usability: We are thinking to develop a game has a lot of functionalities. For beginners usability will be relatively low because of the game's functionalities. Player will find it difficult at first to understand how to do use these functionalities. However,

after playing for a while player will get used to the game and hence its functionalities.

2 Software architecture

Our software is a single game with numerous object interactions. While we are designing the game, our duty will be to come up with a simple design that has the lowest coupling so that implementation will be easier.

2.1 Subsystem decomposition

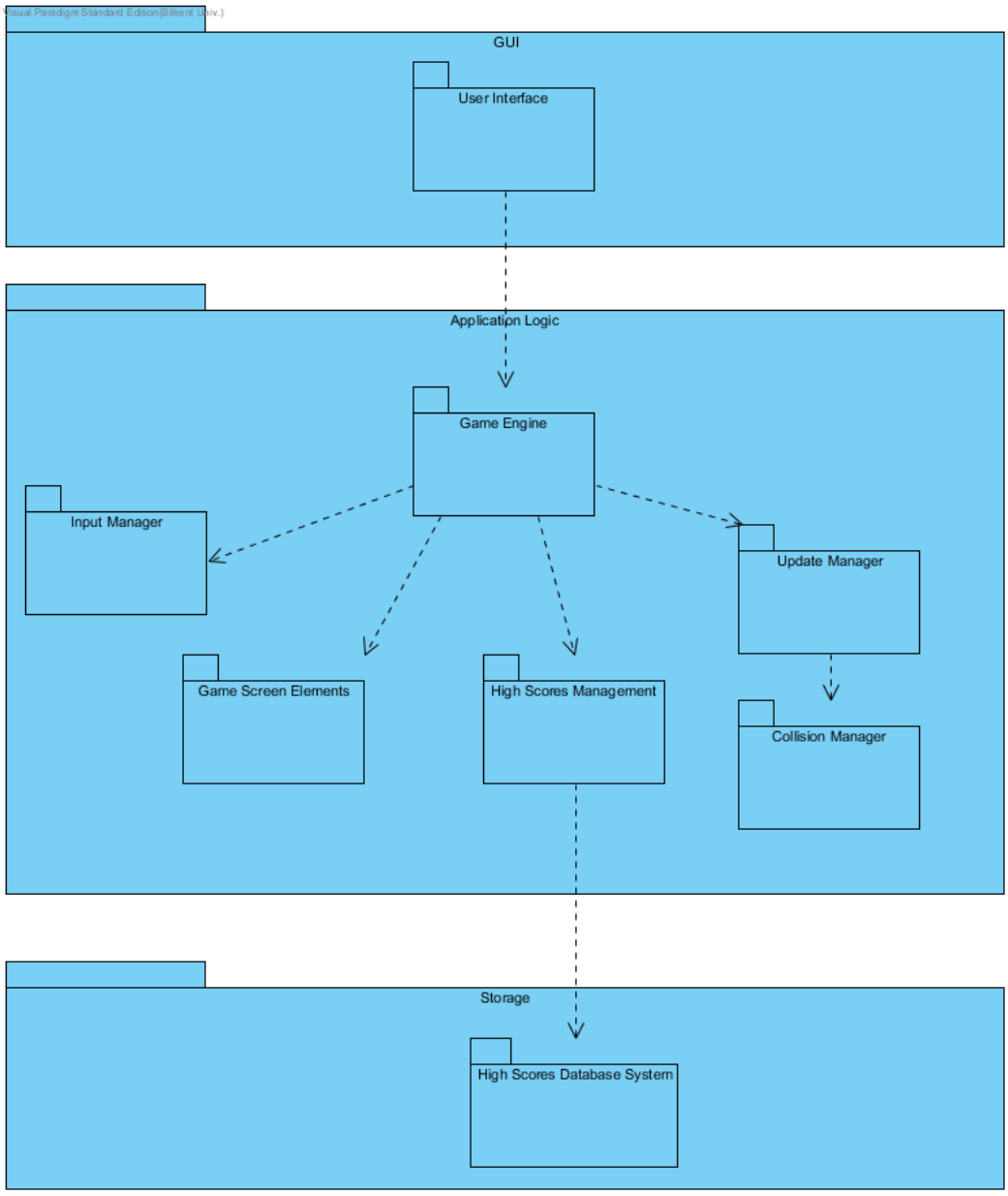

Figure 1 – Subsystem Decomposition

For Space Despot, we choose the three-tier architectural style which will organize our subsystems into three layers: GUI, Application Logic and Storage (Figure 1).

The **GUI layer** will have classes that are responsible for providing an interface between the user and the system.

The **Application Logic** layer will includes all control and entity objects.

The **Storage layer** will take care of the storage of information in the database.

2.2 Hardware/Software mapping

2.2.1 General

Our game will be implemented in Java programming language using the latest JDK (1.8). Most computers will be able to run the program since our program does not contain any intense graphical and physical operations such as 3D rendering and scientific modeling. All the graphical operations and physical calculations will be implemented using the built in Java Libraries. We expect that operations will be nearly instantaneous but may require a reasonable amount of memory and processor to provide a better game experience.

In terms of storage, we will have an external database in our 3rd tier of our architecture to store the high scores of the game. That will require network connectivity to perform the transactions of the data. The transactions will be done when the game is over. Thus,

database system will not have a huge impact on the overall system performance.

The reason why we use database system along with Java language is to address the modularity, performance and portability of our program. We want our program to run fast, platform independent and have independent components which leads to better maintainability.

2.2.2 Input/output

User will need keyboard and mouse combination to secure the interactions between the user and boundary objects. No extra piece of hardware will be needed.

2.2.3 Memory

The database server and PC will have enough memory more than needed for entity objects in our program to buffer burst of requests.

2.2.4 Processor

The visual subsystems do not require heavy computational power because all the components will be designed in 2D. Thus, no rendering will be needed since we do not have any 3D objects. Game Engine will make basic arithmetic calculations such as checking the collisions and updating the health points of space mobs. Other than that, no heavy calculations will be done on CPU. Most PCs with single processors will handle these operations easily. No multiple processors will be needed.

2.3 Persistent data management

In Space Despot, high scores have a persistent identity that needs to be archived every time the game finishes. Database is accessed one time for each game. We decided to use database online because we are going to deal with multiple users; high scores will be used by concurrent readers and writers. Also, database queries data efficiently.

We will use MySQL to do our query operations and phpMyAdmin to manage the database [5] [6].

2.4 Access control and security

Our game will not have user authentication system. However, every time user finishes the game, we will get user's name to save high scores. Every user will have same rights to use the game. This means, there is no need to authenticate a user via ID and password to give to set the limitations of user in the program. Thus, we are not going to implement an authentication system using access matrices.

Since we do not hold any valuable user information there will be no security issues to deal with.

2.5 Boundary conditions

Initialization

CASE: Space Despot will be initialized if user launches the program via .jar file.

Termination

CASE 1: Space Despot can be terminated if user clicks Quit Game.

CASE 2: Space Despot can be terminated if user presses the key combination Alt + F4 while running the game on Windows (or corresponding combinations on other operating systems).

Failure

CASE 1: If there appears a database failure, program will throw an exception and prompt the user with the proper message. The main program will not crash but the user will not be able save high score to the database.

CASE 2: If there appears a system failure or a hardware failure (Screen of Death, electricity issues etc.), there is no way to prevent the crash of the program; in this case, user will lose all progress since our game does not have a progress-saving feature, in other words, user will not be able to continue the game from the point game crashed.

3 Subsystem Services

Here is the detailed explanation of our subsystem layers (Page 3 - Figure 1).

3.1 GUI layer

In the GUI layer, there will be big user interface management system. In this User Interface, these screens will be located: Main Menu Screen, View Settings Screen, View High Scores Screen, View Help Screen, Upgrades Screen, Pause Screen and Next Level Screen.

After user chooses to play game, Play Game Screen will be activated and Application Logic tier will step in. After the game ends for the

user, a pop-up will appear and ask for user's nickname if he achieved a high score and Application Logic tier will interact with Storage tier to save the high score.

3.2 Application logic layer

The main subsystem of this layer is Game Engine subsystem. When user chooses to play game, at first, Game Engine will construct the game.

Game Engine is the main synchronizer class of game objects. In our game there are numerous types of space objects. At first, there will be spaceship for player. Then, there will be space mobs: creatures, asteroids, and bosses. Also, there will be space obstacles: stars and black holes. Space mobs may drop space items: power-ups and coins. Spaceship can shoot spaceship bullets and space mobs can shoot creature bullets and boss bullets, depending on the mob type. Then, in each time interval, Game Engine will trigger Update Manager to do work. For example, Update Manager will check and handle collisions between all objects with the help of Collision Manager. There are a lot of interactions between objects in our game. For example, when spaceship collides with power-up, spaceship will pick this power-up and power-up will be removed from space. Another example is: when space mobs collide with spaceship bullets their health points will decrease. Hence, Collision Manager will have a lot of work to do. With working hand-in-hand with Collision Manager, Update Manager will remove objects from the game if their health bars drop to zero or if they become out of

map. Also, it will be responsible for sending new space objects to the map. For example, periodically, it will send space mobs like creatures and asteroids. It will send boss at the end of level. It will send space objects like stars and black holes. Besides, it will allow space mobs to shoot bullets. After each single update, game map will be repainted by Game Engine. When level time ends, Update Manager will set next level and update the Game Level for Game Engine.

Game Engine will use Input Manager for user interactions. User will use mouse to switch between menus, input manager will handle these. In game play, user will use arrow keys for moving spaceship, X key as default for shooting, Space key as default for using power-up and P key as default for pausing the game. Input Manager will catch these and update Game Engine and User Interface accordingly.

Lastly, game engine will use High Score Manager at the end of each game. High Score Manager will get user's final score. With regard to current top 10 scores, this final score might be placed into the storage with user's nickname if it is at top 10 by High Score Manager.

3.3 Storage layer

There is High Scores Database in this tier which will be used by High Score Manager. High Scores Database System keeps the high score info. It saves user nicknames and their high scores.

References

[1] Space Impact. https://en.wikipedia.org/wiki/Space_Impact [Accessed: Nov 12, 2016].

[2] Asteroids. https://tr.wikipedia.org/wiki/Asteroids [Accessed: Nov 12, 2016].

[3] Cognitive Flow: The Psychology of Dream Game Design. http://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology_of_.php [Accessed Nov 12, 2016].

[4] Flow Theory. https://en.wikipedia.org/wiki/Flow_(psychology) [Accessed Nov 12, 2016].

[5] MySQL. https://en.wikipedia.org/wiki/MySQL [Accessed Nov 12, 2016].

[6] phpMyAdmin. https://en.wikipedia.org/wiki/phpMyAdmin [Accessed Nov 12, 2016].