

CS 319 - Object-Oriented Software Engineering  
Analysis Report  
  
SpaceGuard

Group 9 Section -2

Ege Karaarslan

Mehmet Ali Deligöz

Ferit Özcan

Table of Contents

[1. Introduction 6](#_Toc435738979)

[2. Requirement Analysis 7](#_Toc435738980)

[2.1 Overview 7](#_Toc435738981)

[2.1.1 Game Play 7](#_Toc435738982)

[2.1.2 Play Modes 7](#_Toc435738983)

[2.1.3 Spaceship 8](#_Toc435738984)

[2.1.4 Aliens 8](#_Toc435738985)

[2.2 Functional Requirements 8](#_Toc435738986)

[2.2.1 Play Game 8](#_Toc435738987)

[2.2.2 Change Settings 9](#_Toc435738988)

[2.2.3 Pause Game 9](#_Toc435738989)

[2.2.4 View Help 10](#_Toc435738990)

[2.2.5 View Credits 10](#_Toc435738991)

[2.3 Non-Functional Requirements 10](#_Toc435738992)

[2.3.1 Comfortable Interface 10](#_Toc435738993)

[2.3.1 Graphic Quality 10](#_Toc435738994)

[2.3.2 Low Response Time 11](#_Toc435738995)

[2.4 Constraints 11](#_Toc435738996)

[2.5 Scenarios 11](#_Toc435738997)

[2.6 Use Case Models 12](#_Toc435738998)

[2.6.1 Main Model 12](#_Toc435738999)

[2.6.2 Play Game 13](#_Toc435739000)

[2.6.3 View Credits 18](#_Toc435739001)

[2.6.4 View Help 19](#_Toc435739002)

[2.7 User Interface 20](#_Toc435739003)

[2.7.1 Main Menu 20](#_Toc435739004)

[2.7.2 Play Game 21](#_Toc435739005)

[2.7.3 Pause Game 21](#_Toc435739006)

[2.7.4 Aliens 22](#_Toc435739007)

[2.7.5 PowerUps 22](#_Toc435739008)

[2.7.6 Change Settings 23](#_Toc435739009)

[2.7.7 Help 24](#_Toc435739010)

[2.7.8 Credits: 25](#_Toc435739011)

[3. Analysis 26](#_Toc435739012)

[3.1 Object Model 26](#_Toc435739013)

[3.1.1 Domain Lexicon 26](#_Toc435739014)

[3.1.2 Class Diagram 27](#_Toc435739015)

[3.2 Dynamic Model 29](#_Toc435739016)

[3.2.1 State Diagram 29](#_Toc435739017)

[3.2.1 Activity Diagram 31](#_Toc435739018)

[3.2.2 Sequence Diagram 33](#_Toc435739019)

[4. Design 37](#_Toc435739020)

[4.1 Design Goals 37](#_Toc435739021)

[Reliability: 37](#_Toc435739022)

[User-Friendliness 37](#_Toc435739023)

[Portability: 37](#_Toc435739024)

[Efficiency: 38](#_Toc435739025)

[Efficiency vs. Reusability: 38](#_Toc435739026)

[Functionality vs. Usability: 38](#_Toc435739027)

[4.2 Subsystem Decomposition 39](#_Toc435739028)

[4.2.1 User Interface Subsystem 40](#_Toc435739029)

[4.2.2 Controller Subsystem 41](#_Toc435739030)

[4.2.3 Models 42](#_Toc435739031)

[4.3 Architectural Patterns 43](#_Toc435739032)

[4.3.1 MVC Pattern 43](#_Toc435739033)

[4.3.2 Layered Pattern 44](#_Toc435739034)

[4.4 Hardware/Software Mapping 44](#_Toc435739035)

[4.5 Addressing Key Concerns 45](#_Toc435739036)

[4.5.1 Persistent Data Management 45](#_Toc435739037)

[4.5.2 Access Control and Security 45](#_Toc435739038)

[4.5.3 Global Software Control 45](#_Toc435739039)

[5. References 47](#_Toc435739040)

# 

# 1. Introduction

SpaceGuard is a 2D java implemented shooting game influenced by Space Invaders. Games like Space Invaders aim to shoot and destroy aliens with a laser cannon and earn as many points as possible.

In our game, we plan to have different features. The game will have three modes. Easy mode will aim to destroy aliens and reach to a certain score limit. Medium and hard modes will aim to destroy aliens and reach to a certain score limit within a certain time limit where the aliens are harder to be destroyed.

Medium and hard mode will allow its user to get a high score as medium and hard modes will end only when the player has no life left.

The game will be a desktop application and will be controlled by a keyboard.

# 2. Requirement Analysis

## 2.1 Overview

SpaceGuard is a shooting/arcade video game that mainly inspired by Space Invaders. It differs from the original game by extra features and modified gameplay. Like any other game embracing infinite gameplay, the main purpose of this game is destroying aliens with a laser gun. Each hit from the SpaceGuard weakens or destroys aliens depending on strength of the aliens. The game will continue forever unless the player has no lives or doesn’t exit the game intentionally.

### 2.1.1 Game Play

The player has the ability to move spaceship horizontally by using left/right arrow keys. Since the game is a 2D fixed shooter game, spaceship is only allowed to move between left and right borders at the bottom of screen. In order to survive longer, player must avoid bombs and destructive powerups dropping from aliens.

To make the game more competitive and keep it exciting, we decided to change the main design from goal focused (levels, saving the princess etc.) to infinite gameplay. Therefore, the player will always be motivated to play again.

### 2.1.2 Play Modes

There are 3 different play modes in the game.

* *Easy Mode*: This mode is for beginner level and like a tutorial. The only goal is surviving until achieving the target score by avoiding destructive powerups/bombs. When an alien is hit, only the addition operation occurs. There is no time limit.
* *Medium Mode*: When the player selects medium mode, gameplay gets more complicated. Firstly, the frequency of destructive powerups and bombs increases. There is a time limit. Therefore, the player is supposed to collect time bonuses which drop from aliens.
* *Hard Mode*: When the player selects hard mode, in addition to medium mode, in order to increase risk, different operations such as divison, multiplication, substraction are added.

### 2.1.3 Spaceship

The spaceship is the only thing that user can control manually. User can shift the spaceship left and right to aim and hit the aliens. By using keyboard, user can fire to target. Spaceship also has some powerups. This powerups can be gained from the aliens. Powerups varies in different types. The effects of the powerups will be negative or positive on the spaceship. For example, when spaceship catches speed powerup, it will increase spaceships speed so that it will easier to aim to aliens.

### 2.1.4 Aliens

There will be three type of aliens in the game and the aliens is the funny part of the game. They can be destroyed by one hit, two hit or three hits from the spaceship. User gains points by destroying the aliens. Aliens has shift left and right and they throw down bombs. If aliens hit spaceship, spaceship will lose one of its lives.

## 2.2 Functional Requirements

### 2.2.1 Play Game

SpaceGuard is a 2D shooting game. The aim of the game is to destroy the aliens and score higher by using the laser of the spaceship. In the beginning player has 3 lives and if a bomb thrown by the alien hits the spaceship player loses one of his/her lives. The game is over when the player loses all of 3 lives and wins if the score limit is reached. The game ends in easy mode if the player reaches the target score. However, as one of the objectives of this game is to reach highest score the game continues until the player loses all of his/her lives or reaches the time limit in medium and hard mode.

Player will face bombs thrown by the aliens and power downs which show up from the destroyed aliens. These features makes the game more challenging. Additionally, power ups that are added makes the game more entertaining.

While playing this game, the player is supposed to move the spaceship and shoot at the same time. This game requires the user to keep track of the spaceship and the aliens, shoot according to their movement directions and keep track of the operation to be done in the hard mode where the operation will be either +, -, \* or /. As a result of playing this game, this will enable the player to develop hand-eye coordination along with the ability to manage multiple objectives.

### 2.2.2 Change Settings

The program provides easy mode as a default to its players. Player can change the game mode from easy to medium or hard and background picture. If the player wants to change play modes or background he selects the preferred background and play mode before he begins to play the game. If a player wants to go back to the default settings a player can chose default system settings. Additionally, whenever the game is closed and reopened the default system settings apply.

### 2.2.3 Pause Game

During the gameplay, player can pause the game. When game is paused, from the pause menu a player can click “continue” to continue playing the game. However, when a player closes the application while the game is paused, player loses any game progress.

### 2.2.4 View Help

User will find instructions and explanations about the game in the help menu.

This help menu includes:

- How to Play

- Controls

This help menu will help user to learn the main concept of the game and teach user how to control the spaceship. Also user will be allowed to learn about the power ups and power downs.

### 2.2.5 View Credits

Player will be able to get information about the game developers.

## 2.3 Non-Functional Requirements

### 2.3.1 Comfortable Interface

The interface of the game will be designed in a way that user can easily find anything about the game and also thanks to this interface, user can play this game effortlessly. Interface is one of the most important non-functional property of a game. As a group, we will try to make our interface as user-friendly as possible to make users feel comfortable while playing our game.

### 2.3.2 Graphic Quality

Graphics is one of the most important property of the video games. So, we want to make our graphics attractive by selecting interesting images for spaceship and aliens. Also graphical smoothness is very important for us. Because we want to create a game with smooth graphics to not disturb user's eyes.

### 2.3.3 Low Response Time

To increase the playability of the game we will try to make our game with lowest response time possible. It means that, the game will respond quickly to the user's spaceship movement so that there will be no delay in the game and this will increase the game experience.

## 2.4 Constraints

The game will be implemented in Java. The JPanel and JFrame classes will be used for frames and panels.

## 2.5 Scenarios

**Scenario 1:** Player requests to start game by pressing play game button from Main Menu. After that System initializes the game manager and creates graphics objects. Then the game loop starts and the system checks if there is any hit. Game manager, updates itself and an alien drops down a bomb. Then the system checks if the game is over and the loop continues until the game is over.

**Scenario 2:** Player desires to change game settings. He presses change settings button in the main menu. System displays settings. Player changes settings according to his desire. After that he presses save, System updates new settings by Game Manager.

**Scenario 3:** Assuming that the Player has changed settings, as in the Scenario 1, Player desires to use default settings provided by the system at the startup. The player presses change settings from Main Menu. System displays settings. Player presses save and the system applies the default settings.

**Scenario 4:** The Player requests to start game by pressing play game button from Main Menu. After that, the user continues to play game as in scenario 1 and at some point in Scenario one after the game loop starts the user presses pause game and then the user clicks main menu to quit the game and return to main menu.

## 2.6 Use Case Models

### 2.6.1 Main Model

This section gives information about the main use case model of SpaceGuard game.

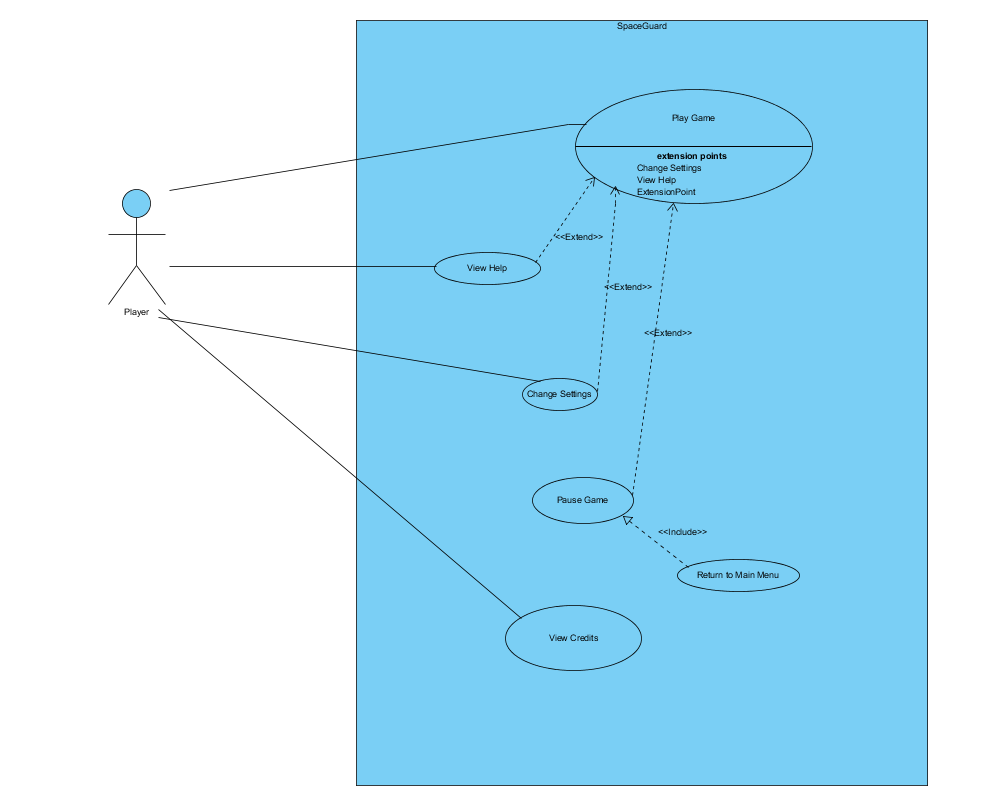


Figure 1 Use case model

Detailed explanations of use cases are in explained below.

### 2.6.2 Play Game

**Use Case Name:** Play Game

**Actor:** Player

**Stakeholders and Interests:**

* Player aims to survive as long as possible and make the highest score.
* System records each player’s score

**Pre-condition:**

* Unless the player changes game settings, easy mode is selected as default.

**Post-condition:** None

**Entry Condition:** Player selects “Play Game” button from main menu

**Exit Condition:** Player returns to main menu by interrupting the game flow.

**Success Scenario Event Flow for Easy Mode:**

1. Game is started by System.
2. Player starts to play without time limit.
3. Player plays until he/she pass the target score.
4. System asks player to change game mode or continue with the same mode.

*If the player chooses to continue with same mod, Step-3 is repeated until player loses all lives.*

1. System updates highscore list
2. System returns to Main Menu

**Alternative Flows for Easy Mode:**

**3.1)** Player tries to survive until achieving the target score:

**3.1.1)** Player fires the laser gun by pressing spacebar.

**3.1.2)** Laser hits alien.

**3.1.3)** System removes aliens that get hit, if they are type of OneHitAlien.

**3.1.4)** System updates the score with respect to value of the shot alien.

*Steps 3.1.1 - 3.1.4 are repeated until achieving target score*

**3.1.5)** Player finishes the easy mode, after achieving target score.

**3.2)** Player gathers the powerUps during game:

**3.2.1)** Laser hits alien.

**3.2.2)** System removes aliens that get hit, if they are type of OneHitAlien.

**3.2.3)** Power up pops up, if the shot alien has one.

**3.2.4)** Player gathers the power up by using spaceship.

**3.2.5)** System applies the power up, if power up hits the spaceship.

**3.3)** Player gets hit by falling bombs:

**3.3.1)** Bombs falls down from the top of screen, randomly.

**3.3.2)** Player gets hit by falling bombs.

**3.3.3)** System removes one life from spaceship, if the player has any.

**3.3.4)** System records the score and returns to main menu, if the player has no lives.

**Event flow for medium/hard mode:**

1. Game is started by system.
2. Player starts to play with a time limit.
3. Player plays the game until time is up or the player has no lives.
4. System updates highscore list.
5. System returns to main menu.

**Alternative flows for medium/hard mode:**

**3.1)** Player gets hit by falling bombs:

**3.1.1)** Bombs falls down from the top of screen, randomly.

**3.1.2)** Player gets hit by falling bombs.

**3.1.3)** System removes one life from spaceship, if the player has any.

**3.1.4)** System records the score and returns to main menu, if the player has no lives.

**3.2)** Player gathers the powerUps during game:

**3.2.1)** Laser hits alien.

**3.2.2)** System removes aliens that get hit, if they are type of OneHitAlien.

**3.2.3)** Power up pops up, if the shot alien has one.

**3.2.4)** Player gathers the power up by using spaceship.

**3.2.5)** System applies the power up, if power up hits the spaceship.

**3.3)** Time is up.

**3.3.1)** Player cannot collect enough time bonus.

**3.3.2)** System records the score and returns to main menu

**3.4)** Player collects points

**3.4.1)** Player fires the laser gun by using keyboard (Default: Space Bar).

**3.4.2)** Laser hits alien.

**3.4.3)** System removes aliens that get hit, if they are type of OneHitAlien.

**3.4.4)** System updates the score with respect to value of the shot alien and the current operation.

*Steps 3.4.1 - 3.4.4 are repeated until achieving target score*

### 2.6.3 Pause Game

**Use Case Name:** Pause Game

**Actor:** Player

**Stakeholders and Interests:**

* Player wants to see the pause menu.
* System displays the pause menu.

**Pre-condition:** Player must be in the game.

**Post-condition:** None

**Entry Condition:** Player presses ESC key during the game.

**Exit Condition:** Player returns to game by pressing ESC key or returns to main menu by selecting “Return to Main Menu” button.

**Event Flow for Pause Menu:**

1. System displays the pause menu.
2. User selects “Return to Main Menu” button.
3. System records the score and terminates the game

**Alternative Flow for Pause Menu:**

1. System displays the pause menu.
2. User selects “Help” button.
3. System dislays Help panel.
4. User returns to pause menu
5. User continues to play the game.

### 2.6.4 View Credits

**Use Case Name:** View Credits  
**Primary Actor:** Player  
**Interests and Stakeholders:**- Player wants to see the information about the developers.  
- System displays the information about the developers  
**Pre-conditions:** Player needs to be in Main Menu.  
**Post-condition:** No Post-condition  
**Entry Condition:** Player clicks “View Credits” in main menu.  
**Exit Condition**: Player clicks “Back” button in View Credits Menu.  
  
**Success Scenario Event Flow:**  
1.System displays information about the developers of the game.  
**Alternative Flows:**  
A. If player wants to return main menu:  
 A.1. Player clicks “Return” button.  
 A.2. System shows Main Menu.

### 2.6.5 View Help

**Use Case Name: View Help**  
**Primary Actor:** Player  
**Interests and Stakeholders:**  
- Player wants to get information about the game.  
- System displays a text which is giving information about the game.  
  
**Pre-conditions:** Player needs to be in Main Menu.  
**Post-condition:** -  
  
**Entry Condition:** Player clicks “View Help” in Main Menu.  
**Exit Condition:** Player clicks “Back” to got back.  
  
**Success Scenario:**  
1.Player clicks “View Help” in Menu.  
2. System shows help document giving information about the game.  
  
**Alternative Flows:**  
B. If Player requests to go back:

**B.1. Player clicks “Back” button in “View Help” screen.  
B.2. Player goes back.**

## 2.7 User Interface

### 2.7.1 Main Menu

When player starts the application, main menu shows up after the splash screen.  
Main menu has five options: Play Game, Settings, Help, Credits, Exit Game. Application is terminated when user selects “Exit Game” option.



Figure 2 Main Menu

### 2.7.2 Play Game

If the player chooses Play Game, game starts with default settings. Here is a screenshot from easy mode gameplay.

Figure 3 Screenshot from easy mode

### 2.7.3 Pause Game

The game can be stopping by pressing escape button during the game. Users can resume to the game or quit existing game by clicking on the buttons.

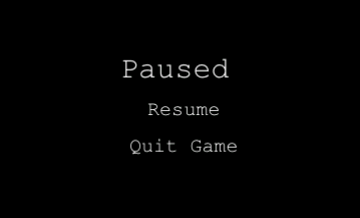
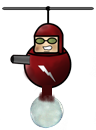
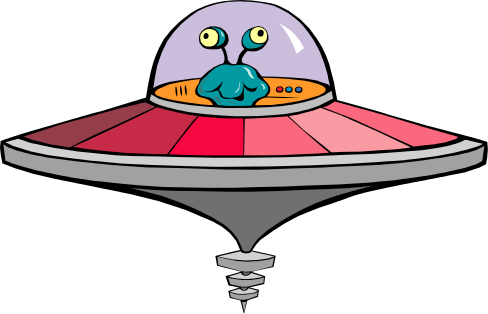


Figure 4 Pause Menu

### 2.7.4 Aliens

In the game, we have three types of aliens. One hit aliens, two hit aliens and three hit aliens. One hit and two hit aliens are moving right and left and bombing constantly. Three aliens are being produced to avoid player to keep pressing left and right buttons which causes to a fix game play. Spawn rate of three hit aliens is %20. They calculate if it is going to hit the player if they release the bomb in that moment or not by physical calculations.



One hit alien Two Hit Alien Three Hit Alien (Sniper)

C:\Users\Ege\Desktop\ferit3119.rtfd\bomb.png This bomb is thrown by the aliens and it causes the player to lose a life if it hits the spaceship controlled by the player.

### 2.7.5 PowerUps

There are %10 possibility that an alien can drop a power up instead of bomb. There are three kind of power up.

Health power up refills the health bar.

C:\Users\Ege\Desktop\ferit3119.rtfd\slowdown.png The snail slows the spaceship for 10 seconds.

C:\Users\Ege\Desktop\ferit3119.rtfd\anim2.pngThe spaceship is the object that is controlled by the user and throws laser.

### 2.7.6 Change Settings

When player presses on the change settings button, a list of options are displayed. These are change difficulty, change background image and apply default settings. If player does not make any change, the default settings are used by the system. Player can select only one difficulty among easy, medium or hard and only one background image for gameplay. Background image is selected using the paddles in the figure. If the player wants to use default settings he should nly click on the “Default settings” checkbox. 

Figure 5 Settings

### 2.7.7 Help

If player enters help menu, there will be a text shown in the screen including playing instructions of the game which means the aim of the game, information about the buttons of the game and information about the power ups. Player can go back to the main menu by clicking on the Back button.

Figure 6 Help

### 2.7.8 Credits:

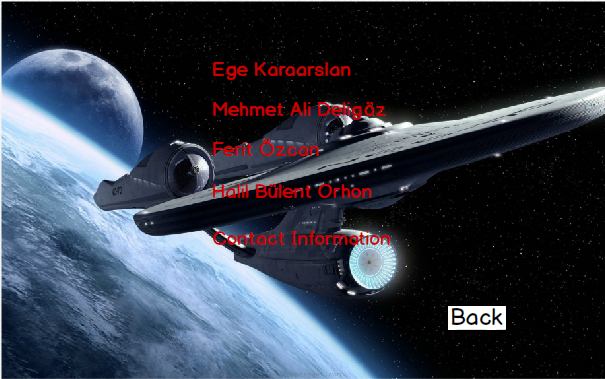
When Player enters the Credits Menu, player will see the contact information of the game developers and their name. Player can go back to the main menu by clicking on the Back button.

Figure 7 Credits

# 3. Analysis

## 3.1 Object Model

### 3.1.1 Domain Lexicon

**Aliens**

One hit alien: Alien that die with one hit.

Two hit alien: Alien that dies with two hits and more challenging bomb throws.

Three hit alien: Also called sniper. It dies with two hits and bomb throws are as challenging as the two hit alien.

**PowerUps**

Snail: It is the slow down power up.

Shield: Shield power up will protect the spaceship for a while.

Time Bonus: In medium and hard modes, player needs to collect these bonuses in order to make the game last longer. (It adds +5 seconds to timer.)

### 3.1.2 Class Diagram

The class diagram of SpaceGuard is illustrated below.

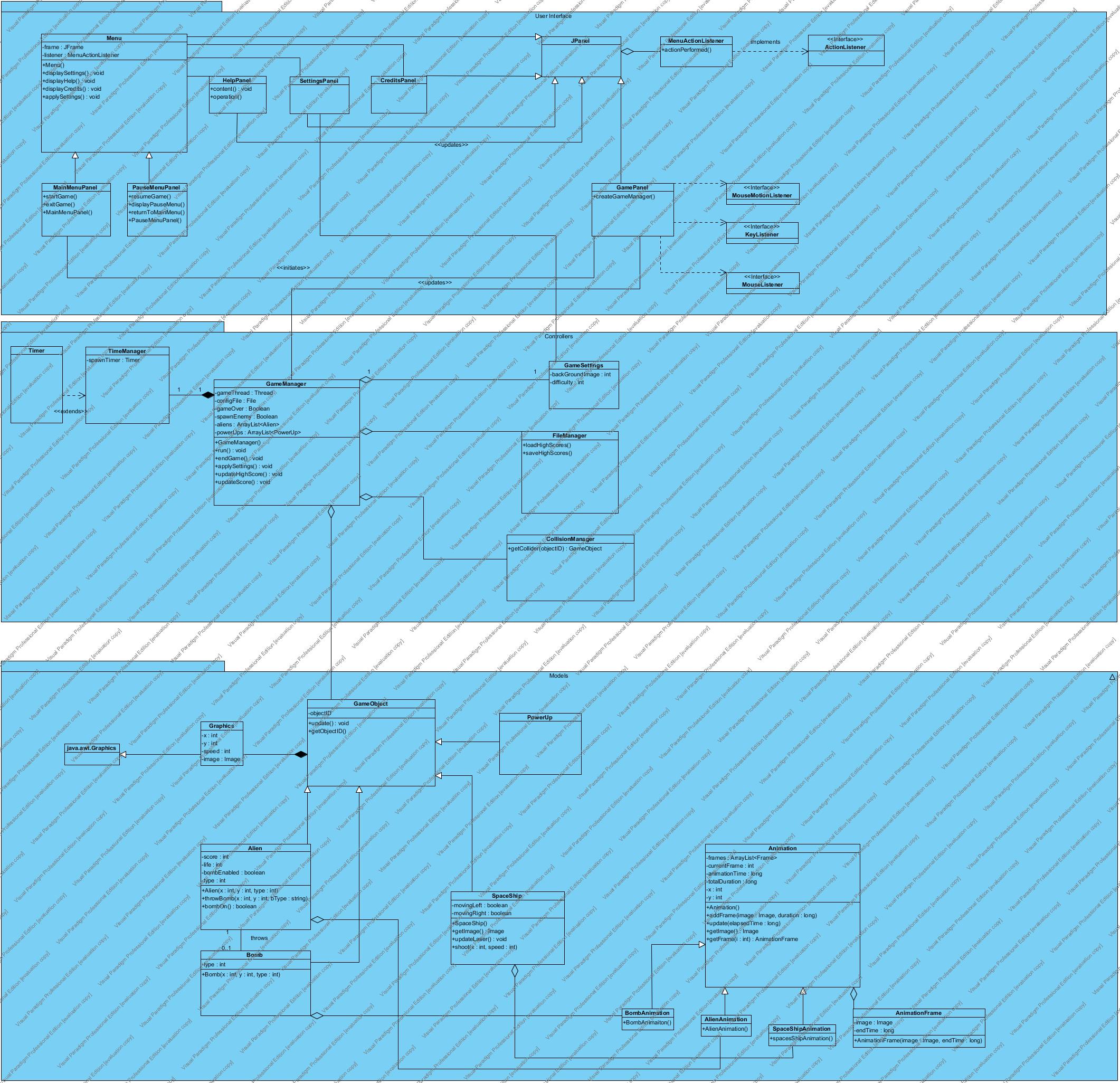


Figure 8 Class diagram

JPanel is one of the classes in java.swing package which we extended in MainMenu, PauseMenu, SettingsPanel, HelpPanel, CreditsPanel, and GamePanel.

Since **MainMenu** and **PauseMenu** have many common properties, we decided to create a parent class **Menu**.

SettingsPanel, HelpPanel, CreditsPanel, and GamePanel can be initiated by **MainMenu.**

In order to maintain User interaction we utilized KeyListener, MouseListener, and MouseMotionListener.

**GameManager** class performs the proper actions on game such as the interaction between objects and the effects on the current map.

Any change on object of **GameSettings** leads modification in background image and difficulty of the game.

**FileManager** is for loading from/saving to the encrypted binary file which holds highscores.

**CollisionManager** is for checking collision between Alien-Laser, SpaceShip-Bomb, and SpaceShip-PowerUp.

**TimeManager** is for keeping track of alien spawn time and the timer which is used in the medium and hard modes of game.

Since all game objects (Alien, Spaceship, and PowerUp) have common properties, we decided to create a **GameObject** classwhichis the parent class of all game models (objectID, update(), etc.).

## 3.2 Dynamic Model

### 3.2.1 State Diagram

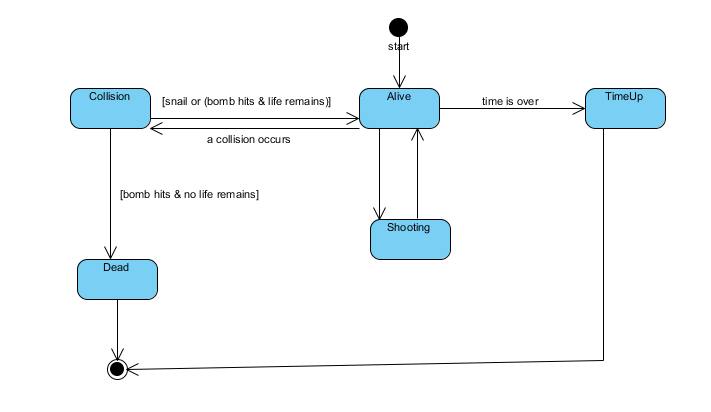
****

Figure 9 State Diagram for SpaceShip

The diagram above represents possible states of spaceship object. When the game is started, spaceship has 3 lives and at the state “Alive”. There is three possible scenarios; the first state is “Shooting. In this state shooting animation is triggered. The second state is “Collision”. Collision may occur between spaceship and a power up or a bomb. After collision, if spaceship still has any lives, spaceship state returns to “Alive” state. Otherwise, spaceship object is destroyed (“Dead” state) and system returns to main menu. Finally, even spaceship has lives, if time is up, again spaceship object is destroyed (“TimeUp”) and system returns to main menu.

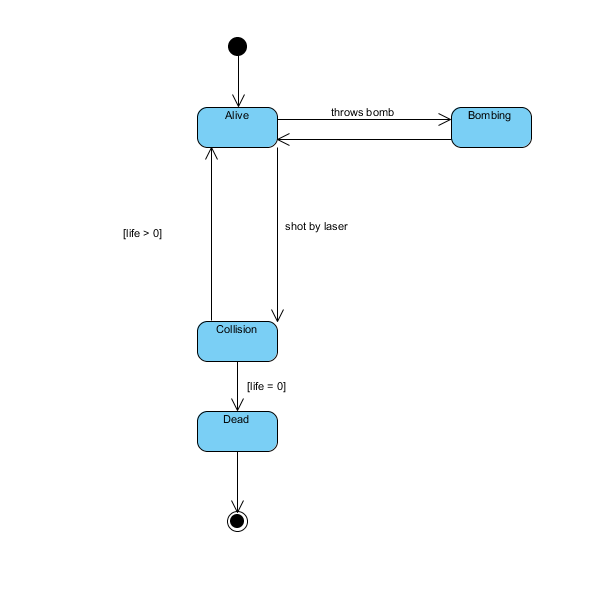


Figure 10 State diagram for Alien object

The diagram above represents possible states of alien object. When the game is started, alien has [1-3] lives depending on the type of alien. Initial state is “Alive”. Alien may drop bombs (“Bombing” state). In this state bombing animation is triggered. Also when alien gets shot by the spaceship, “Collision” state is called. “Collision” state checks whether alien object is destroyed or not. If alien has any lives, then returns to initial state which is “Alive” state. Otherwise alien object is destroyed by the system.

### 3.2.1 Activity Diagram

Figure 11 Activity Diagram for Play Game Use Case

When user selects Play Game, system initializes the game by creating game manager and game objects. When this is done, system waits for the user to press right/left arrow or space from the keyboard to start the game. When spaceship begins to move, the game starts and the system checks if the game is over. If player has no life left the game is over, otherwise the game continues. If player has life, system checks if there is a hit.

If the hit is between a powerUp and spaceship then the corresponding bonus feature is applied to the spaceship. If the hit is between spaceship’s laser and Alien then, the alien is destroyed. After this two cases, system updates game manager with remaining game objects. If the hit is between alien’s bomb and spaceship then player loses one of his lives and the system checks if the game is over. The game is over only when player has no lives left and the system checks whether the game is over after every hit.

### 3.2.2 Sequence Diagram

#### 3.2.2.1 Initializing the Game

Game Loop shown below is the Sequence Diagram of scenario 1.

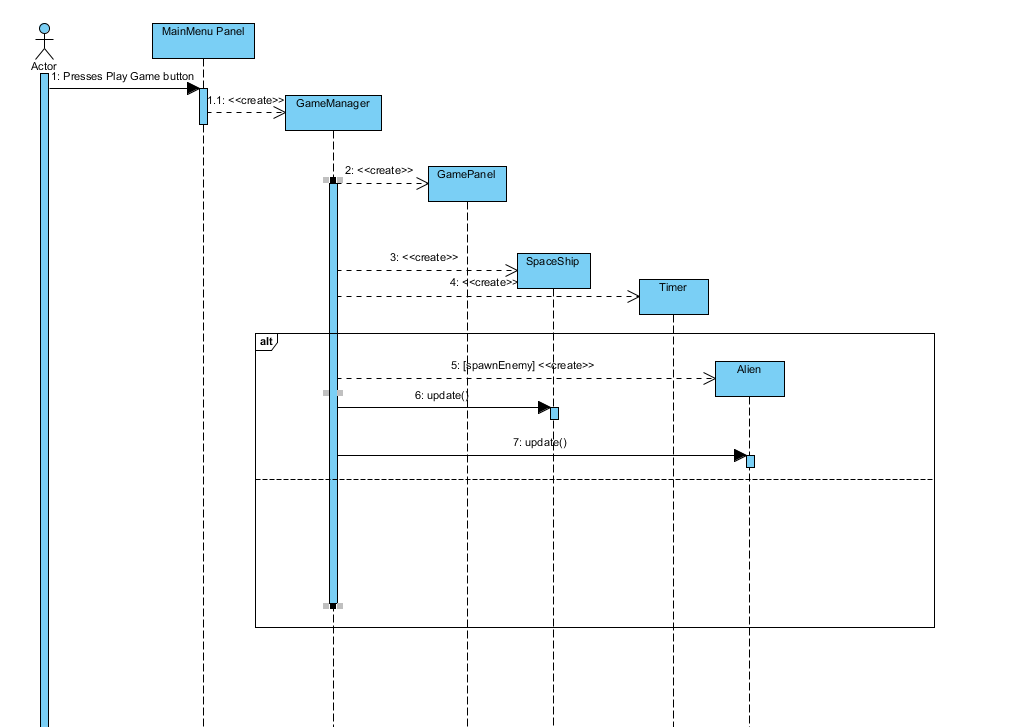


Figure 12 Sequence Diagram for initializing the game (Scenario-1)

In order to initialize the game, player presses “Play Game” button on the MainMenuPanel. MainMenuPanel has a 1-1 association with the main game controller “GameManager”. GameManager creates GamePanel which contains “SpaceShip” and “Alien” objects. To keep track of the alien spawn times and the counter (the time limit), a Timer object is created. In the game loop, system creates Alien objects and Alien/SpaceShip objects are updated continuously.

#### 3.2.2.2 Game Loop

This is a scenario where the User presses spacebar to shoot aliens. GamePanel notifies GameManager which takes care of the game entities. In the loop, GameManager checks collisions between bomb-spaceship, powerup-spaceship, and laser-alien. When a collision occurs, GameManager is able to notify animations of objects or to destroy objects.

#### 3.2.2.3 Moving the SpaceShip



User can move in horizontal axis at the bottom of GamePanel. In this scenario, user presses the right/left arrow key, and SpaceShipAnimation is triggered by the SpaceShip object.

#### 3.2.2.4 Settings

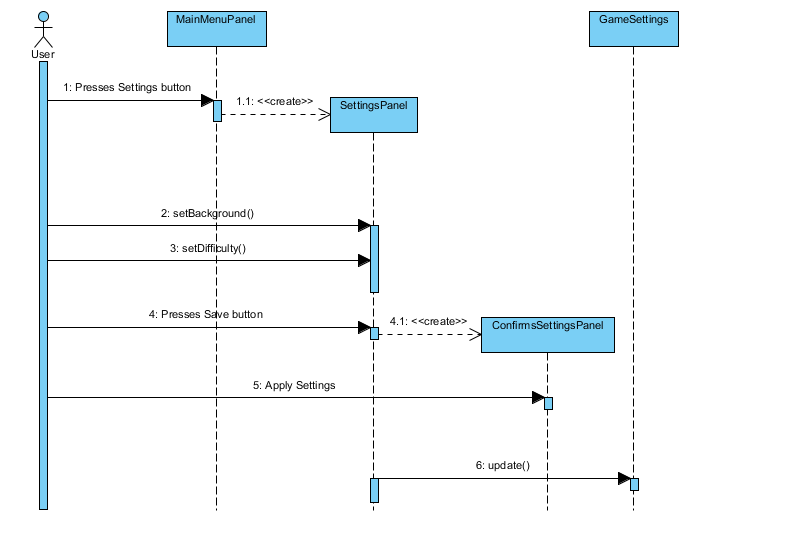


Figure 13 Sequence Diagram for changing settings

This sequence diagram refers to the Scenario 2. In this diagram, player presses change settings button and menu creates a settings panel. Player sets difficulty, background and presses save and the changed settings are sent back to the main menu.

# 4. Design

SpaceGuard is a simple shooting game. Since it is a basic shooting game, we thought that it should have high performance and be user-friendly. The default game setting is easy mode and it aims to make user involved in the gaming experience and play against less challenging aliens. Afterwards, the user can change the game mode to medium or to hard whenever he/she feels that he is ready for the next step. By selecting medium and hard mode the game becomes more challenging and preserves user’s attraction. Therefore, the aim of our game is to make it easy to learn in the beginning and make it challenging with harder modes.

## 4.1 Design Goals

Reliability: The Spaceguard should be reliable. This means that when a player is playing the game, it must not crash because of internal reasons. To achieve this, during the implementation stage, testing procedures will be done simultaneously with the development.

User-Friendliness**:** One of the main goals of SpaceGuard is to be simple, easy to be played and entertaining. In order to catch the user’s attention, the game should not require the player to spend a lot of time trying to figure out how to use our system. To achieve this, the system will provide user friendly interfaces for menus where users can find desired operations easily. Additionally, the system will perform it’s actions according to the mouse input and keyboard input from the user, like clicking buttons by using mouse, moving spaceship by using keyboard.

Portability:The SpaceGuard must be portable. In other words, the game should be implemented with a programming language that can be run on any environment so that the game can be played on different operating systems.

Efficiency: The SpaceGuard is a shooting game, that is, it must be smooth. Therefore, the game must have high performance. This goal is one of the most important design goals since it has a crucial role on user’s excitement.

Efficiency vs. Reusability: The classes are designed specifically for the tasks of our game so that the code is not more complex than it needs to be. Since we do not have any intention to integrate any of our classes to a different game or any other system reusability is not the main concern for designing our system.

Functionality vs. Usability:The system should not be too complex to play. It means that the functionality of the system will be basic. Since the purpose of the system is to entertain the users, we focused on the usability of the system rather than making it more functional than necessary. The game has a simple interface and familiar instructions to play instead of complex menus and various features. Thus, the users can spend time enjoying the game rather than struggling to learn it.

## 4.2 Subsystem Decomposition

Subsystem decomposition is one of the essential parts of system design. In the subsystem decomposition part, the system is separated into relatively independent parts in order to clarify how the system is organized. Decomposition of parts has an important role in order to create high quality software and meeting nonfunctional requirements. The whole system is divided into three subsystem:

1. *Models*
2. *Views*
3. *Controllers*

We are going to explain MVC architecture later in [§4.3.1](#_4.3.1_MVC_Pattern).

Dividing the system into three subsystem regulates the interaction of each classes, reduces the complexity of design, and also helps developer to manage the code.

In the figures below, it can be seen that there is no connection between Figure 14 User Interface Subsystem and Figure 16 Models Subsystem. This design limits the models from accessing user interface directly. A game object can only interact with GameManager.

### 4.2.1 User Interface Subsystem

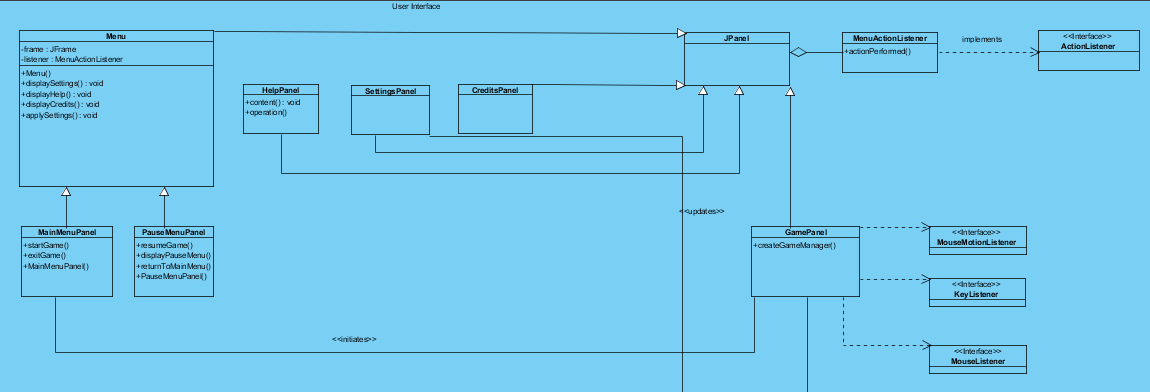


Figure 14 User Interface Subsystem

User interface contains visual parts of the game such as panels, buttons, and labels. On the right side; MouseListener, MouseMotionListener, KeyListener, and ActionListener handles interaction between user and the system. It also manages transitions between panels which are constructed for variety of options from the main menu screen.

Java.awt.\* and java.swing.\* packages are implemented in order to present the user interface.

### 4.2.2 Controller Subsystem

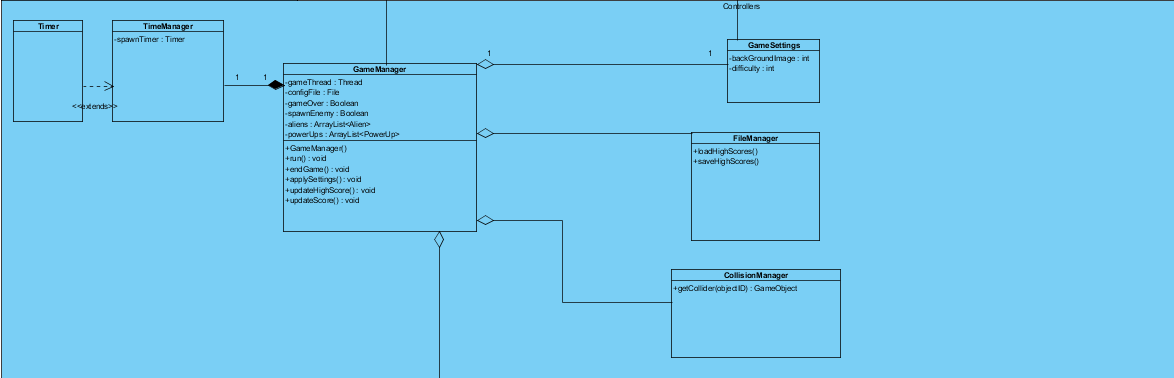


Figure 15 Controller Subsystem

In this subsystem, **GameManager** class performs the proper actions on game such as the interaction between objects and the effects on the current map.

Any change on object of **GameSettings** leads modification in background image and difficulty of the game.

**FileManager** is for loading from/saving to the encrypted binary file which holds highscores.

**CollisionManager** is for checking collision between Alien-Laser, SpaceShip-Bomb, and SpaceShip-PowerUp.

**TimeManager** is for keeping track of alien spawn time and the timer which is used in the medium and hard modes of game.

### 4.2.3 Models

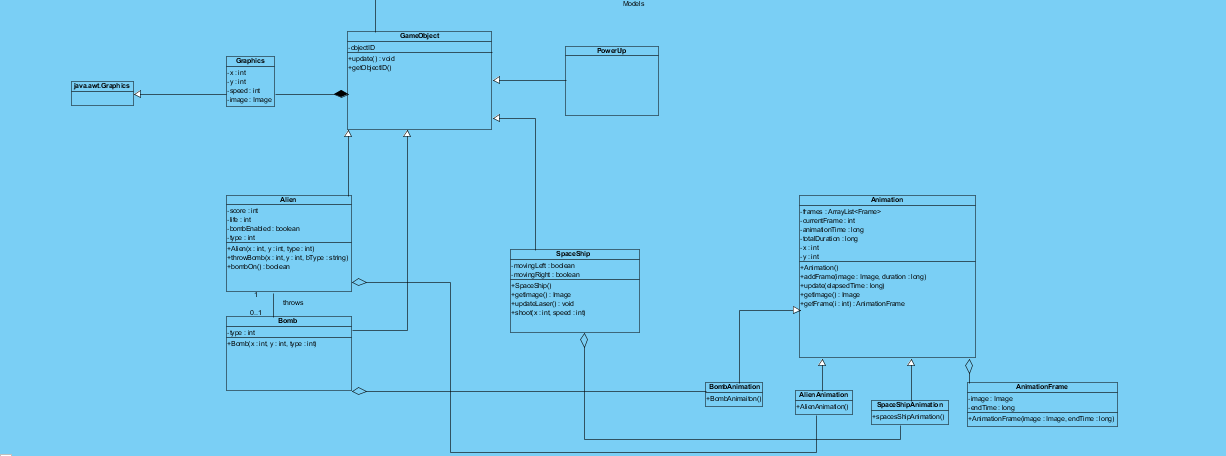


Figure 16 Models Subsystem

Since all game objects (Alien, Spaceship, and PowerUp) have common properties, we decided to create a **GameObject** classwhichis the parent class of all game models (objectID, update(), etc.).

## 4.3 Architectural Patterns

### 4.3.1 MVC Pattern

While implementing a game, the changes caused by the user controls or effects of the game

engine should be viewed and updated immediately. Additionally, in most of the games when a

developer wants to improve the User Interface of a game, the developer prefers to have a

separate graphics and core packages since separating them enables improvability and

Model, View is the interface and controller part is the Handlers/Listeners that deals with control

commands and changes. For example, we have GameManager class for controlling the game

parameters and modifying current game variables according to the controls applied by the user.

We have a GamePanel class for displaying the data and the current state of the game to the user.

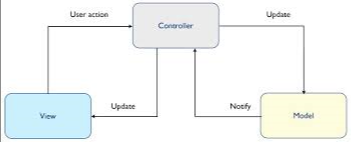


Figure 17 MVC Architecture

### 4.3.2 Layered Pattern

We have Interface Layer, Data access layer and Game logic layer. Interface layer can use Game logic layer and Game logic layer can use Data Access layer. In Data Access layer, the game keeps the game data. In Game logic, we have model and controller classes. In interface layer we have view classes. Interface can’t reach to Game Logic or Data Access layer.

## 4.4 Hardware/Software Mapping

SpaceGuard does not require any internet or lan connection to play. The underlying reason is that our initial design of the game supports single player. As a result of this, SpaceGuard only needs a PC to run. As hardware configuration, SpaceGuard needs a keyboard to give input the gameplay, mouse to give input to the menu of the system. The language we preferred is Java and we will be using the latest JDK (JDK 8). Since system requirements for this game will be minimal, a computer with a java compiler will be enough and the operating system will not affect since java is platform independent. The high score data will be stored in the local storage of the client’s PC.

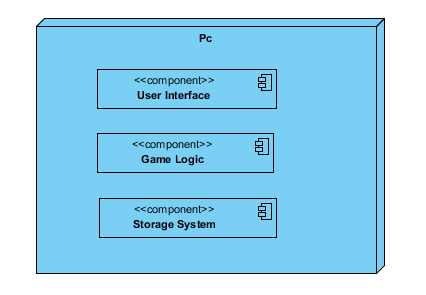


Figure 18: SpaceGuard Deployment Diagram

## 4.5 Addressing Key Concerns

### 4.5.1 Persistent Data Management

The game will be used by only one user and the game data will be stored in the user’s hard disk drive. We will not use any database since the data we use in the game needs to be accessed in real-time. Therefore, we will load all the necessary files on to the memory and access those files when the game logic or the rendering system requires. The background images and spaceship image will be stored unencrypted to encourage modifying background images for their personal preference. The leaderboard will include player’s name typed by the user and the high score. Since the stored data will consist of only username and high score it will not be complex.

### 4.5.2 Access Control and Security

As stated in the Hardware / Software Mapping section, SpaceGuard will not require any kind of network connection. So, the system will have only one actor which is the player. For that reason anyone who initializes SpaceGuard will be able to play the game. So, there will not be any kind of restrictions or control for access. Also it will not include any user profile. Therefore, there will be no kind of security issues in SpaceGuard. The player can have access to background images and alter them from the stored file if the user gives the same name to the images.

### 4.5.3 Global Software Control

In our project, we simply create necessary classes and objects. Then, the system waits for an input from the user and choices from the menus. To handle inputs, we will use event-driven programming paradigm. Whenever, KeyListener of Java, detects an input, it causes the application to start a related event. We have a centralized control model, one single component controls the system operations like starting, stopping and coordination and it is called GameManager.

#### 4.5.4 Boundary Conditions

**Initialization**

SpaceGuard is implemented in Java. Therefore all Java classes and associated metadata and resources (text, images, etc.) are archived into one application software on the Java Platform. User can execute the archived file which has **.jar** extension.

**Termination**

User can terminate the game by clicking on “Exit” in main menu. Alternative way to terminate the game is to click “X” sign at the upper right.

**Error**

If any corruption occurs in game resources, game will not be initialized.

# 5. Object Design

## 5.1 Design Patterns

In this section, we describe the patterns we used and the reason behind those patterns, also at last section we present the patterns applied class diagram of the system. In the context of the project, we applied Strategy, Observer as behavioral, Singleton as creational and Façade as structural patterns.

### 5.1.1 Singleton Pattern

In our system, we use creational pattern singleton in order to ensure that some components have only one instance in the active system. Those components are GameManager.

 “GameManager” class is the main controller class of the project which deals with controlling both GUI components and run-time behaviors so that it is needed to be accessed in most of the other classes in a global way with only one instance and we should ensure that it has only one instance so that we applied Singleton Pattern here.

The code of the pattern:

private static GameManager manager = null;

public static GameManager getInstance()

{

If (manager == null)

manager = new GameManager ();

return manager;

}

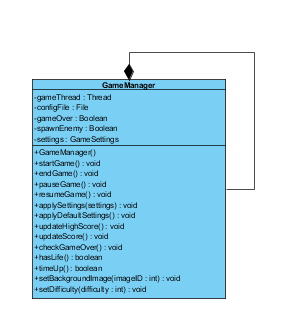


Figure 18 Singleton Pattern of GameManager

### 5.1.2 Façade Pattern

In the system design, Model-view-controller pattern was applied as an architectural pattern. In order to function properly these packages need to communicate. Model part includes only the entities of the game, controller includes methods to handle inputs and view consists of graphical user interface methods. The problem here is that our view and controller needs to have an information exchange and having such communications increase coupling. To handle this problem, we decided to use façade pattern. Façade pattern encapsulates a complex subsystem within a single interface object. Thus, it reduces complexity by grouping common properties of classes in one interface.

In GameMap class of our project, we use this pattern to simplify interaction and access issues of model subsystems and controller subsytems. We group all game entities under GameMap since they have common operations and attributes. Besides, we applied Façade pattern to GameManager class.

### 5.1.3 Abstract Factory Pattern

Abstract factory pattern is another pattern that can be used to provide an alternative to our code for the menu class. A menu class like PanelContainer can be used for transitions between panels. Instead of creating new panels whenever a user presses Start Menu button or any other buttons inside a panel that will create a new panel and destroy it once it is done with it, there can be a changePanel(newPanel: Jpanel) method that will cause transitions to be more smooth. The class interface of Panel container would be:

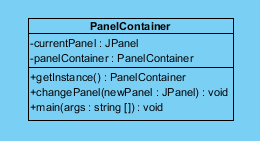


Figure 19PanelContainer

## 5.2 Class Interfaces

### 5.2.1 Model Classes

#### GameMap class

GameMap class is the Façade class of the GameEntities subsystem. It includes methods related to creating and modifying game entities. Communication between the other components of the GameEntities subsystem is performed by this class.

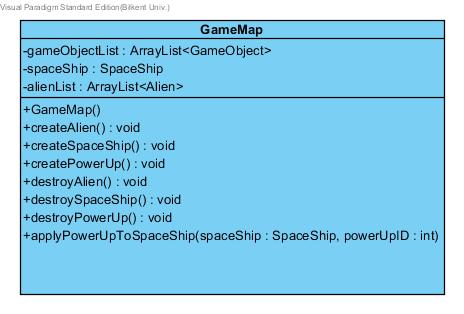


Figure 20 GameMap Class

##### Attributes:

privateArrayList<GameObject> **gameObjects:** Holds all entities of the system.

private SpaceShip **spaceShip:** Since there is onlt one spaceship this attribute represents the spaceship.

private ArrayList<Alien> **alienList:** holds reference to whole 10 aliens.

##### Constructor:

public **GameMap():** constructs a game map with default attribute values.

##### Methods:

public void **applyPowerUpToSpaceShip(SpaceShip spaceShip, int powerUpID):**  applies the power up indicated by powerUpID to the given **SpaceShip** object.

public PowerUp **createPowerUp( int powerUpID ):**  creates a **PowerUp** object with given ID.

public SpaceShip **createSpaceShip():** creates and returns a **SpaceShip** object.

public Alien **createAlien():** creates and returns a **Alien** object.

public void **updateAlien(Alien alien):** updates the location of the given **Alien** object.

public void **destroyAlien(int alienId):** Removes the alien with the given ID from the alienList.

public void **destroySpaceship():** Removes the spaceship.

#### PowerUp Class

This class is composed of the static constant variables and the constructor of PowerUp.

##### Attributes:

SNAIL, ACCELERATE\_BOMB, TIME\_BONUS, SHIELD are private int constants that hold id values for powerUps.

##### Constructors:

**public powerUp ( int ID ):** This constructor of PowerUp class uses an integer to define the type of the powerUp objects. By using this ID, it loads the proper image from disk for a specific powerUp.

#### GameObject Class

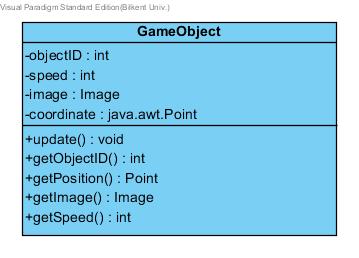


Figure 21 GameObject

GameObject class represents the object in the game namely, spaceship, bombs, aliens. It encapsulates the properties of object id, speed, image and current place. Each game objects has also a point for their coordinates.

##### Attributes:

**private int objectID:** Object attribute of Game Object class is used by some of child classes of Game Object class to specify the type of other objects of child classes.

**private int speed:** All child classes of GameObject have speed attribute.

**private Image image:** All child classes of GameObject have image attribute.

##### Methods:

**public int getObjectID():** returns the ID of an object.

**public Point getPosition():** returns the current position of an object.

**public int getSpeed():** returns the speed of an object.

#### Alien Class

**Extends:** GameObject.java



Figure 22Alien

##### Attributes:

**private int score:** Each alien has a score that randomly changes in interval (0, 9).

**private int life:** Depending on the type, an Alien may have (1, 3) lives.

**private boolean isBombEnabled**: If it is false, alien can’t throw a bomb. bombEnable() function determines its value.

**private int alienType**: Instead of creating sub-classes for different aliens, alienType is used for determining strength and life of an alien.

##### Constructor:

public Alien(Point coordinate, int alienType): Constructor for alien object

##### Methods:

**public void throwBomb(Point coordinate, int bombType)**: If alien is able to throw a bomb (if isBombEnabled == True), alien throws a bomb vertically from given coordinate.

**public void enableBomb()**: The algorithm uses normal distribution with mean 3 and standard deviation with 0.7. By using normal distribution, predictability of bomb dropping time decreases.

**public void setHealth(int value)**: This setter will be used for decreasing the alien’s health being hit.

**public int getHealth()**: Returns the current health of alien.

#### Bomb Class

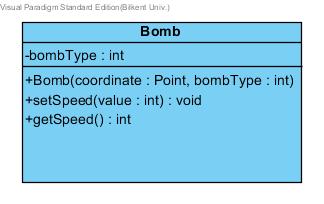


Figure 23Bomb

##### Attributes

**private int bombType**: There are two bomb types which have different damage values.

##### Constructor:

public Bomb(Point coordinate, int bombType)

##### Methods:

public void setSpeed(): Setter for bomb’s speed.

public int getSpeed(): Getter for bomb’s speed.

#### SpaceShip Class

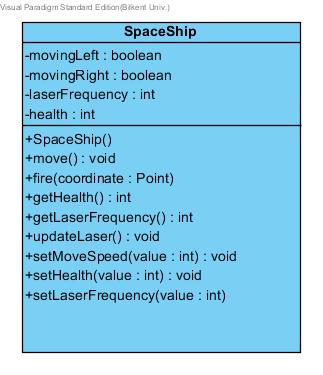


Figure 24SpaceShip

##### Attributes

**private boolean movingLeft**: Becomes true, if player is moving to left at the moment.

**private boolean movingRight**: Becomes true, if player is moving to right at the moment.

Above two Boolean values are also used in alien.move() method to dodge spaceship’s laser if possible. Aliens are slightly intelligent.

**private float laserFrequency**: Fired laser per second may change with power ups.

**private int health** : Spaceship’s health may change with bombs and power ups.

##### Constructor:

public SpaceShip(): Default constructor for spaceship object

##### Methods

**public void move():** Invoked by updateSpaceShipLocation() method, calculates the new position.

**public void fire(coordinate: Point):** Spaceship fires the laser from given coordinate.

**public void setMoveSpeed(int value):** Spaceship’s speed may change with power ups.

**public void setLaserFrequency(int value)**: Fired laser per second may be changed by power ups.

**public void setHealth(int value):** Spaceship’s health may change with power ups or bombs.

#### Animation Class

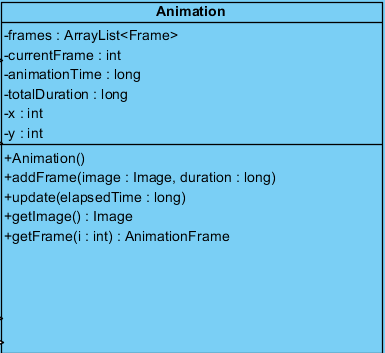


Figure 25Animation

##### Attributes:

private ArrayList<Frame> frames: Holds frames for alien/bomb/spaceship animations.

private int currentFrame: Index of current frame

private long animationSpeed: Speed of animation.

private long totalDuration: Total duration of animation.

##### Constructors:

public Animation(): Default constructor for Animation class.

##### Methods:

public void addFrame(Image image, long duration):

public void update(long elapsedTime):

public Image getImage(): Returns the image of a game object

public AnimationFrame getFrame(int i): Returns the frame with given index

#### AnimationFrame Class

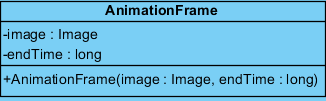


Figure 26AnimationFrame

##### Attributes:

private Image image: Simple image instance.

private long endTime: Determines duration of animation.

##### Constructors:

public AnimationFrame(Image image, long endTime):

##### 

### 5.2.2 Controllers Subsystem Interface

In this subsystem, our controller objects are grouped together to manage the actual game dynamics and game logic. We have 6 components in this subsystem. As illustrated in Figure-1, we have GameManager, CollisionManager, GameMapManager and a TimeManager which is a subclass of GameManager for time related issues. We have also a property class called GameSetting. These classes will be explained in detail.

#### GameManager Class

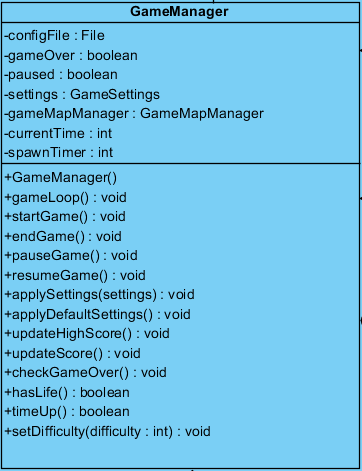


Figure 27GameManager

This class is the Façade class of the Controllers subsytem, it performs the proper operations according to the requests that came from User Interface subsystem, also this class runs the game in a loop. This class also implements runnable interface, since the game loop runs as a thread.

##### Attributes:

**private boolean paused**: this attribute is used for whether the game is paused or not, in game loop

**private GameSetting settings:** this attribute holds the settings of the game , like background type and difficulty.

**private GameMapManager mapManager:** this attribute is a GameMapManager object, by which GameManager class associates with proper methods of GameMapManager class.

private int currentTime: Simply holds the current time.

private int spawnTime: GameManager determines spawn time of aliens.

private Boolean gameOver: Becomes false, if time is up or spaceship has no life.

##### Constructors:

**public GameManager():** it initializes the attributes of the GameManager for the first run of the system.

##### Methods:

**public void gameLoop():** this methods runs a loop in which the system is updated continuously.

**public void applyDefaultSettings():** this method changes the setting attribute of the class according to the default settings that are determined by system.

**public void applySettings(GameSetting settings ):** applies the given settings to system by calling setDifficulty() and setBackground().

**public boolean hasLife():** checks whether the number of lives bigger than zero, to check this communicates with GameMapManager class by mapManager attribute.

**public void startGame():** starts a new game, by using the chosen settings, obviously it is made by mapManager attribute.

**public void pauseGame():** when invoked causes to prevent the game loop to iterate by setting paused attribute to false.

**public void resumeGame():** when invoked causes the game loop to continue to iterate, by setting the paused attribute to true.

**public boolean isHighScore(score: int):** returns true if the given score is eligible for the high score list else it returns false.

**public void endGame():** if hasLife() returns false or timeUp() method returns true, this method invokes the isHighScore method and returns to the main menu.

**public boolean timeUp():** returns true if time is up. GameLoop stops if time is up and endgame() is called.

**public void setDifficulty(GameSettings setting):** sets the difficulty of the game. This is useful for determining which game mode is used(gameMode attribute of GameMapManager).

#### InputManager class

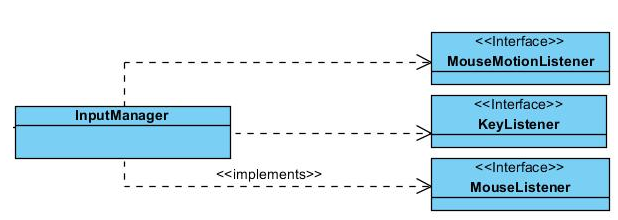


Figure 28InputManager

This class is designed to detect the user actions performed by keyboard( to move paddle on screen etc. ), and also performed by mouse( to pause the game and invoke a pause menu during game ). In this context, this class implements proper interfaces of Java.

#### FileManager Class

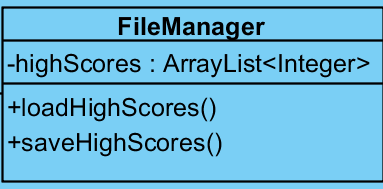


Figure 29FileManager

##### Attributes:

private ArrayList<Integer> highScores: System doesn’t store usernames, but stores highscores.

##### Methods:

public void loadHighScores(): Reads the highscore file, and puts values to highScores.

public void saveHighScores(): After the game is finished, GameManager invokes this method.

#### GameSettings Class

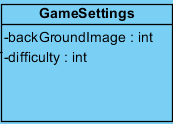


Figure 30GameSettings

This class is a simple property class, which holds the settings of the system. It is used by GameManager class.

##### Attributes:

private int backGroundImage: References specific background images.

private int difficulty: Represent different game modes.

#### GameMapManager Class

##### **C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\GameMapManager.jpg**Attributes:

**private GameMap gameMap:** this class performs the map specific operations by referencing to this object. Like getting the alien objects on the map etc.

**private int score:** score achieved by the player

**private int gameMode:** this attribute holds the current game mode information. A game mode can be 1,2 or 3 where they stand for easy, medium and hard respectively.

**private int spaceShipLives:**  number of remaining lives of the spaceship is hold.

##### Constructors:

**public GameMapManager():** initializes a GameMapManager object with default attribute values.

##### Methods:

**public GameMap getGameMap():** returns the current map which is processing by GameMapManager class.

**public drawGameMap (Graphics g, int score ,int spaceShipLives,int gameMod):** draws the current map according to given attributes.

**public void setBackGroundImage(int ImageID ):** sets the background image according to given ID, by loading a proper image by this id.

**public void applyPowerUp(int powerUpID):**  applies the powerUp to the game by referencing the gameMap attribute, also it references to powerUp class for interpreting which id refers to which powerUp ( which is a class of Game Entities subsystem ).

**public void update(long elapsedTime) :**  updates the game objects of gameMap by given elapsed time.

**public void generatePowerUp() :**  when invoked, generates a random integer and generates a powerUp object with this integer by associating with the powerUp class of GameEntities subsystem, to figure out which powerUp this integer corresponds. This power up is created and added to map by referencing to gameMap object. Then inside gameMap object powerUp will be created and added to powerUpList in gameMap.

**public void handleLaserAlienCollisions() :** by referencing the gameMap object, this method finds and handles the collisions between laser and alien objects of gameMap object.

**public void handlePowerUpSpaceShipCollisions() :** by referencing the gameMap object, this method finds and handles the collisions between powerUp and spaceShip objects of gameMap object.

**public void handleBombSpaceShipCollisions() :** by referencing the gameMap object, this method finds and handles the collisions between spaceShip and bomb objects of gameMap object.

**public void checkAndHandleCollisions():** this method invokes the methods of this class, which are designed to handle the collision issues, and finds and handles the collisions on the currentMap by one invocation.

**public void updateSpaceShipLocation(Point coordinates):**  updates the location of the spaceShip on gameMap, since spaceShip is controlled by keyboard actions. Also this method is invoked by GameManager when there is a keyboard event.

### 5.2.3 User Interface Subsystem Interface

User Interface subsystem provides our software system with graphical system components. Besides, it also manages transition between panels which are constructed for different selections in menu screens. The reference of User Interface Subsystem to other subsystems is provided by menu class which is considered as an interface.

#### Menu Class

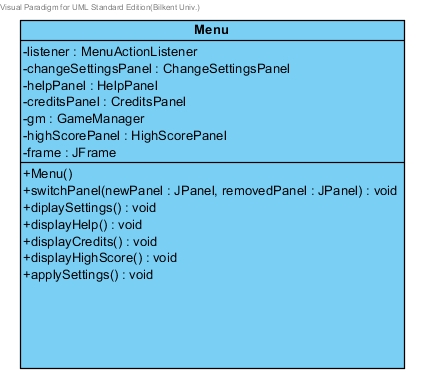


Figure 32Menu

##### Attributes:

**private JFrame frame:** This is the frame in which we display all visual context.

**private MenuActionListener listener:** This attribute is to get the user input from the graphical user interface.

##### Constructors:

**public Menu:** Initializes *changeSettingsPanel, creditsPanel, helppanel, gm, settings and listener* properties.

##### Methods:

**public void displayHelp():** This method includes switchPanel() method and opens a *helpPanel.*

**public void displayCredits():** This method includes switchPanel() method and opens *creditsPanel*

**public void displaySettings():** This method includes switchPanel() method and opens S*ettingsPanel.*

#### MainMenuPanel class

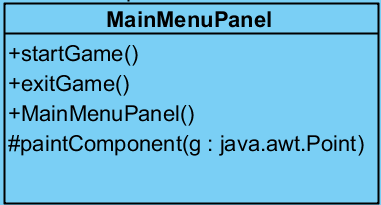


Figure 33MainMenuPanel

##### Constructor:

**public MainMenuPanel():** It initializes instances of MainMenu object.

##### Methods:

**public void displayMainMenu():** This method includes switchPanel() method and adds *mainPanel* to frame by replacing the current panel on frame.

**public void startGame():** It opens a gamePanel and the gamePanel creates the gameManager, objects and the game begins.

**public void exitGame():** This method ends the run of application.

#### PauseMenu class

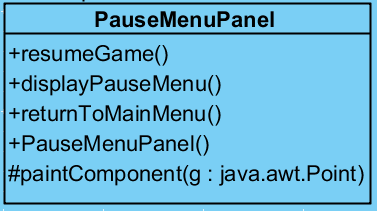


Figure 34PauseMenuPanel

##### Attributes:

**private PauseMenuMainPanel mainPanel:** This PauseMenuMainPanel type attribute of PauseMenu class is used in graphical user interface to show Pause Menu on screen.

##### Constructor:

**public pauseMenu():** It initializes instances of PauseMenu object.

##### Methods:

**public void resumeGame():** This method removes PauseMenu panel and continuous game routine.

**public void returnToMainMenu():**  This method creates a new mainMenu object and displays it on screen.

**public void paintComponent(Graphics: g)** this method paints the pause menu panel.

#### HighScorePanel Class:

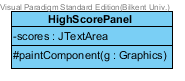
HighScorePanel is the panel that provides the high score list to the user. It has scores as the property and paintComponent(g:Graphics) method as a protected method.

Figure 35HighScorePanel

#### SettingsPanel Class:

SettingsPanel is the panel that provides the user with the ability to change the settings of the game. It has paintComponent(g: Graphics) method as a protected method. It has also rightArrow, leftArrow Jbuttons, difficulty JButtonBar and defaultSetting JCheckBox.

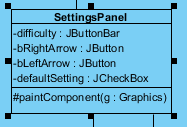


Figure 36SettingsPanel

#### HelpPanel Class

HelpPanel is the panel that provides the required information on how to play the game to the user. It has paintComponent(g:Graphics) method as a protected method as well as the property info as a JTextArea.

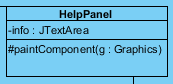


Figure 37HelpPanel

#### CreditsPanel Class

CreditsPanel is the panel that provides the required information on the creators of the game. It has paintComponent(g:Graphics) method as a protected method as well as the property info as a JTextArea.

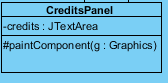
****

Figure 38CreditsPanel

#### GamePanel Class

GamePanel is created by the main menu when the user presses play game. It creates a single object of GameManager only. It has paintComponent(g:Graphics) method as a protected method.

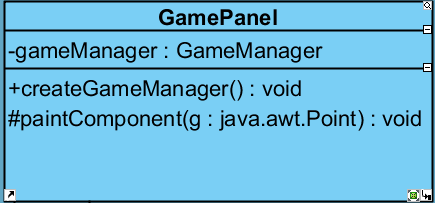


Figure 39GamePanel

##### Attributes:

private GameManager gameManager: A GameManager instance.

##### Methods:

public void createGameManager():When user presses *play* button gameManager object is created.

## 5.3 Specifying Contracts using OCL

1. context SpaceShip inv: speed = 7

//the spaceship’s initial speed is 7

1. context SpaceShip::getHealth():int post:result = health

//getHealth method returns health

1. context Bomb inv: speed = 6

//initial value of speed of bomb is 6

1. context Bomb::setSpeed(speed : int)

post : self.speed = speed

//setSpeed method sets the speed of bomb where speed is an int

1. context SpaceShip::applyPowerUp():pre: laserSpeed = 7, post: laserSpeed = 12

//in applyPowerUp method of spaceShip laser speed which is initially 7 becomes 12 as a result of //this method

1. context SpaceShip::deletePowerUp(): pre: laserSpeed = 12, post: laserSpeed = 7

//deletePowerUp method causes the laserSpeed to get back to the initial laserSpeed value which is //7

1. context GameManager::applyDefaultSettings():Settings post:result = defaultSettings

//default settings are enabled

1. context GameManager::applySettings(newSettings:Settings) post: savedSettings = newSettings

//GameManager applies new settings

1. context Bomb inv: HEIGHT = 20

//height of image of bomb is 20

1. context Bomb inv: WIDTH = 20

//width of image of bomb is 20

1. context Bomb::setSpeed() : int

post : result = speed

1. context GameManager::createAlien(Alien alien) Pre: ( alienList.Size() < 10 && spawnEnemy = true)

//To be able to create aliens number of aliens should be less then 10

1. context Alien::bombEnabled: boolean init: self.bombEnabled=false

//initially alien is not allowed to throw bomb

1. context GameMap inv: alienList.size() >= 0 && alienList.size() <=10

//alienList’s size should be greater than or equal to 0 and it should be less then or equal to 10

1. context GameMapManager inv: score = 0

//Score should be 0 initially

1. context GameManager::gameOver boolean: init: false

//gameOver is false initially

1. context SpaceShip::isHit: boolean: init: false

//isHit is false initially

1. context SpaceShip::setMoveSpeed(shipSpeed:int) post: speed = shipSpeed

//moveSpeed is set to shipSpeed

1. context GameObject inv: objectID > 0 //objectID is greater than 0
2. context Alien::getalienType() int post: result = alienType

//alienType which is an integer is returned

1. context Alien::throwBomb() body:

if(self.bombEnabled)

throwBomb()

//alien throws bomb if bombEnabled is true

1. Context GameMap::destroyAlien(Alien a) Post: (score >=score@pre)
2. context Bomb inv: type > 0 //type of bomb is greater than 0
3. context SpacesShip::movingLeft boolean: init: false
4. context SpacesShip::movingRight boolean: init: false
5. context SpaceShip::setHealth(value) pre:

getHealth() > 0

//health of spaceShip should be greater than 0 to do this

1. context SpaceShip:: setHealth (value) post:

getHealth() = self@pre.getGetHealth() + 1

//setHealth causes the health that is initially given to the setHealth is increased by one

1. context Alien::getHealth(value) pre:

getHealth() > 0

//health of alien should be greater than 0 to call getHealth otherwise it is destroyed

1. context Alien::setHealth(value) pre:

setHealth() = self@pre.getGetHealth() + 1

//alien’s health should be 1 greater than it’s initial health

1. context GameObject inv coordinate.getX() >= 0

//x coordinate of game object should be greater than or equal to 0

# Conclusion

## 6.1 Summary

SpaceGuard is a game that entertain people and get away from daily problems for a while. This game enables users to challenge themselves, try to reach higher score, deal with time limit and help develop ones hand-eye coordination.

In particular this is system is developed in three main stages. First was the analysis process. In analysis, we describe our problem and define functional and non-functional requirements, derive use case diagrams and uses cases to handle our requirements. Additionally, we derived class diagram and domain lexicon to provide a deeper understanding application domain. We derived sequence diagrams and state diagrams to show dynamical behavior.

The second process was design process. In this part, we defined the main systematic behavior of our game. We began by defining our system goals; we divide our system into subsystems and we completed the decomposition of our system. Then, we applied two architectural patterns, MVC and Layer to make our system more modular and derived the map of hardware-software architecture and listed tradeoffs.

The third part in this project was object design. In object design process, we applied four different design patterns to our system to improve our implementation process. Those patterns were singleton, façade and abstract factory. After this, we derive the class interfaces and specified contracts using OCL.

## 6.2 Lessons Learned

The lessons learned during the project can also be separated into three parts which are the knowledge gained during analysis, system design and object design.

In analysis design, we learned how to describe a problem statement and provide solutions using functional requirements. We learned how to write use-cases, draw use case diagrams which enabled us to capture dynamic aspect of a system. We also learned how to detect use-cases and creating use-case diagram that corresponds to the use-cases. Lastly, we draw the dynamic models of our system to explain the behavior of our system and to draw these models we have learned how to use Visual Paradigm.

In system design, we have considered different implementation types and architectural patterns like MVC, Layered pattern, peer to peer and some other patterns. Among all the architectural patterns, we have used MVC and layered patterns and we have decomposed our system into subsystems. During the system design report, we noticed that we need to think critically since this process required a deep thought on design and make the required changes accordingly.

In object design we have learned structural patterns in detail and we had to apply the appropriate patterns to our system to provide a more concrete system. But most importantly, during this design we have noticed how to be a part of an iterative life-cycled project. This was most effectively seen in this part since whenever we added a new property to a class we had to alter the class diagrams and other related diagrams. Additionally, we have noticed that design process requires a lot more effort than we thought it would require.

## 6.3 Problems Faced

First problem we faced was in the analysis phase. We didn’t know how to derive functional requirements and use cases for our system. We have seen functional requirements examples in our CS319 classes but we have had trouble deciding what requirements our system needs to have. Secondly, we didn’t know how to write domain lexicon and what should be put into domain lexicon. Lastly, we didn’t know how to make dynamic modeling. We get over these problems by making researches on these concepts and we tried to find sample projects so that the concept understanding will become knowledge. After deciding the patterns, we had to learn how to implement them in the project and this required more work to be done in a tightly scheduled semestre.

The last obstacle we faced was the deadline of implementation since the deadline of the implementation and the final report were only four days apart.

## 6.4 How can this project be developed?

As a future work, we can add levels to this game as an arcade mode, and the current version of the game would be freestyle mode since the game continues until the time is over or player loses all of his/her lives. Additionally, more fancy powerups can be added and the game can be implemented for android devices since mobile games are getting more and more popular every day.

# 7. References

Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.SS