

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.

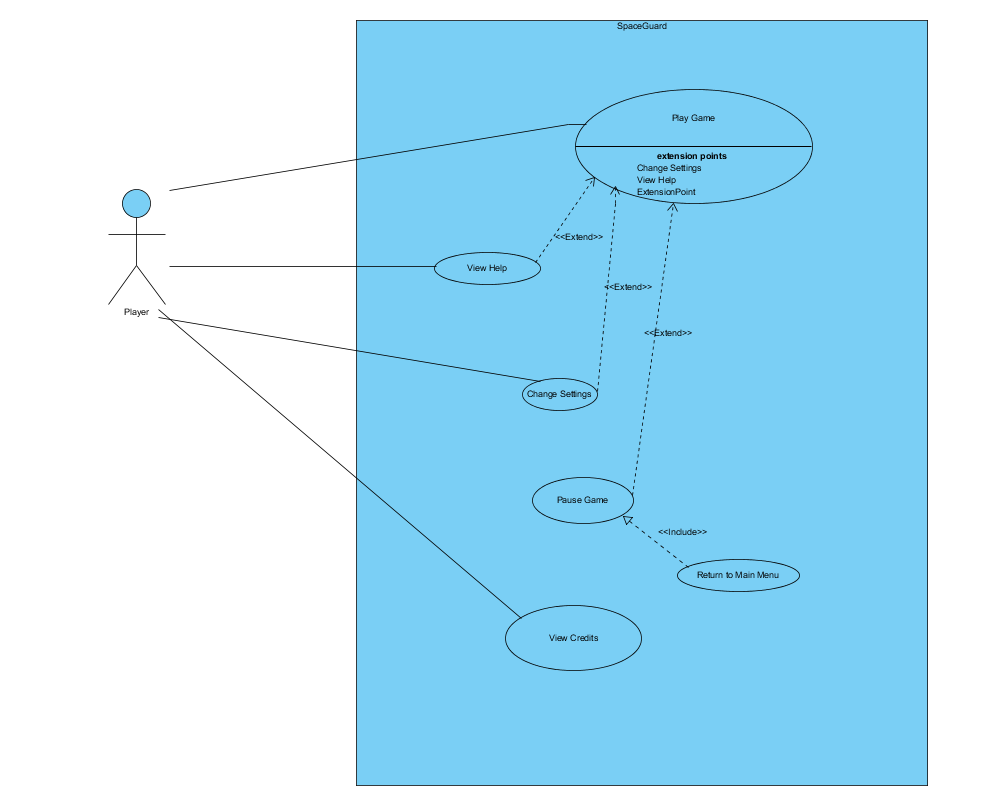


Figure Use case model

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 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.

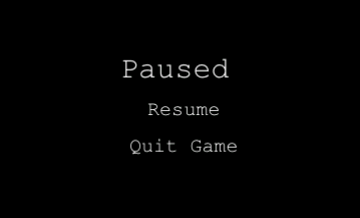
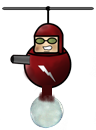
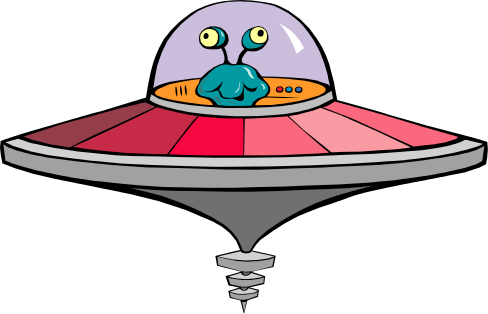


Figure Pause Menu

### 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 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 Help

### 2.7.8 Credits:

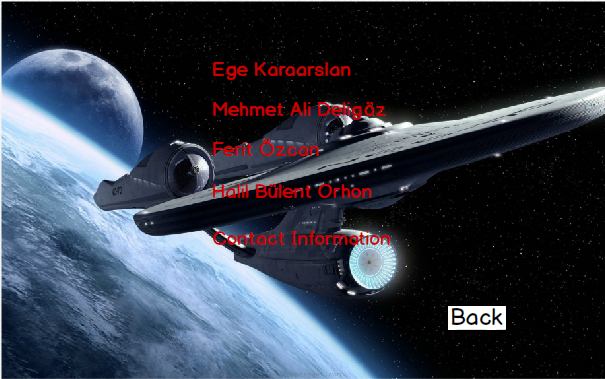
When Player enters the Credits Menu, player will see the contact information of the game developers and their name. Player can go back to the main menu by clicking on the Back button.

Figure 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.

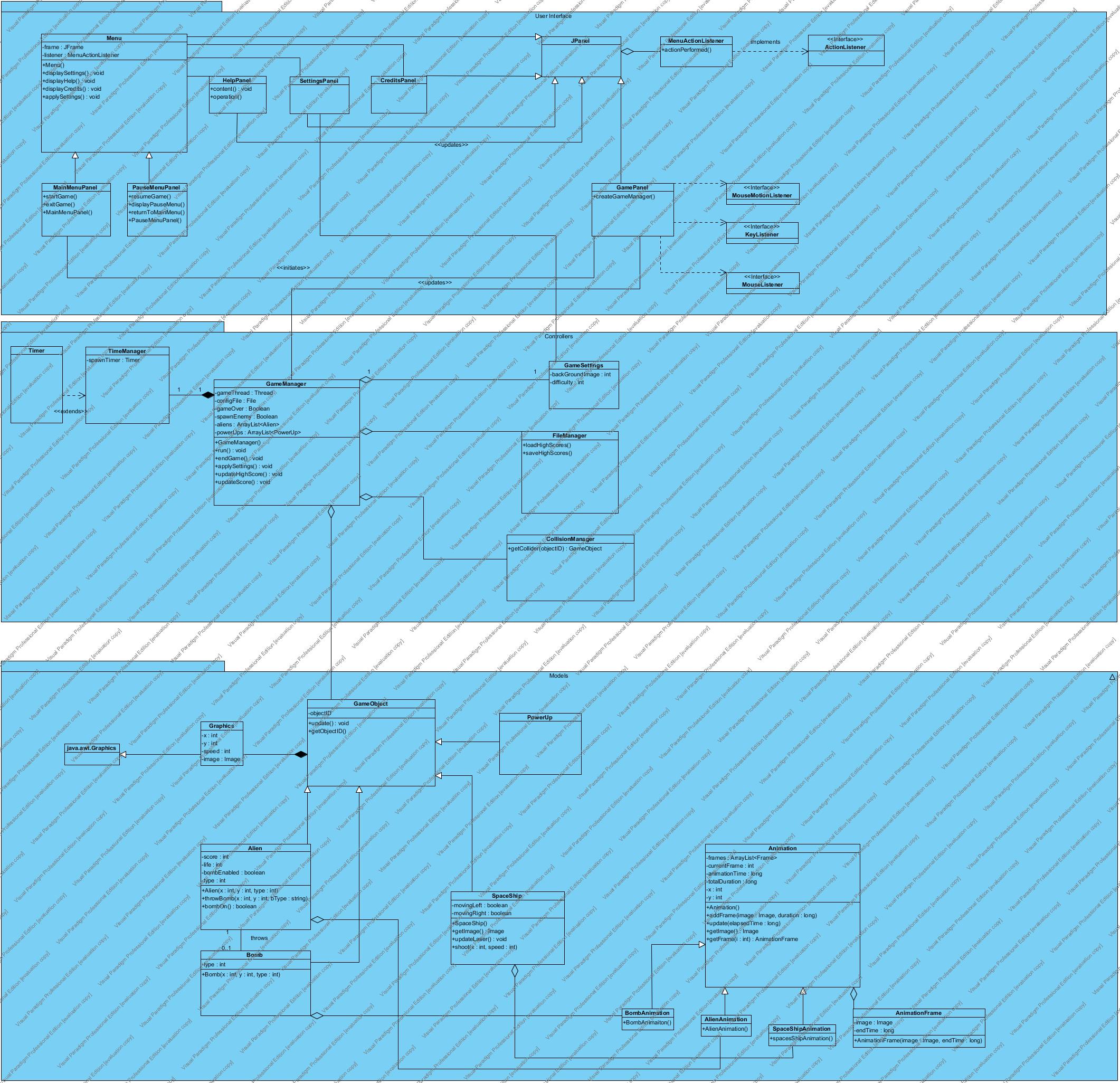


Figure Class diagram

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

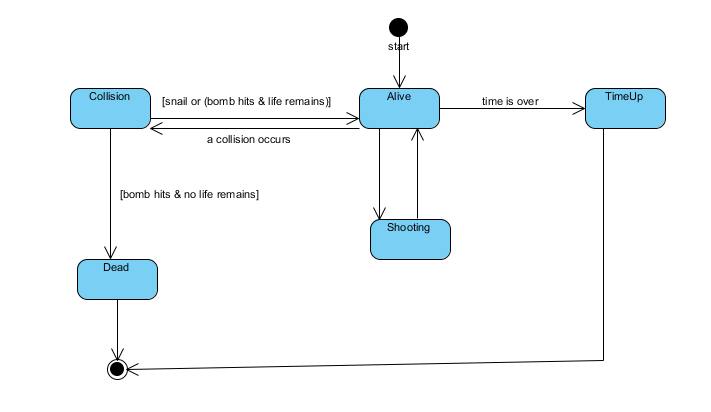
****

Figure 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.

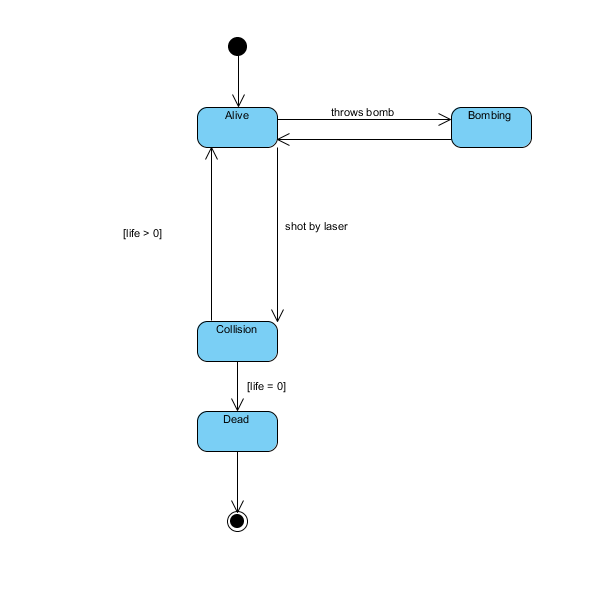


Figure State diagram for Alien object

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.

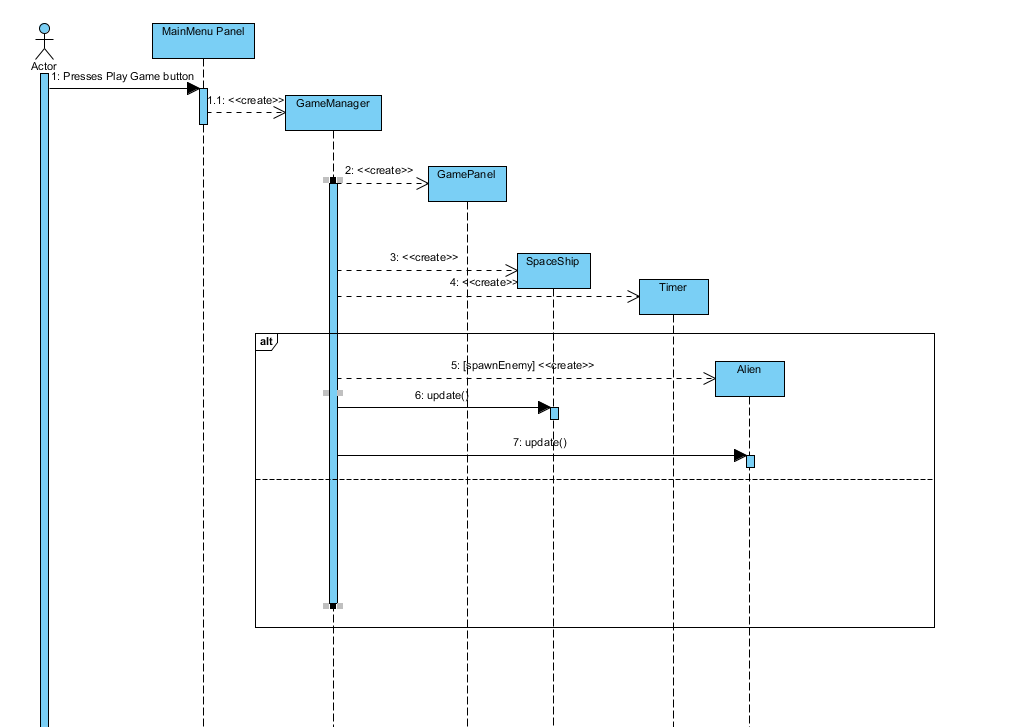


Figure Sequence Diagram for initializing the game (Scenario-1)

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

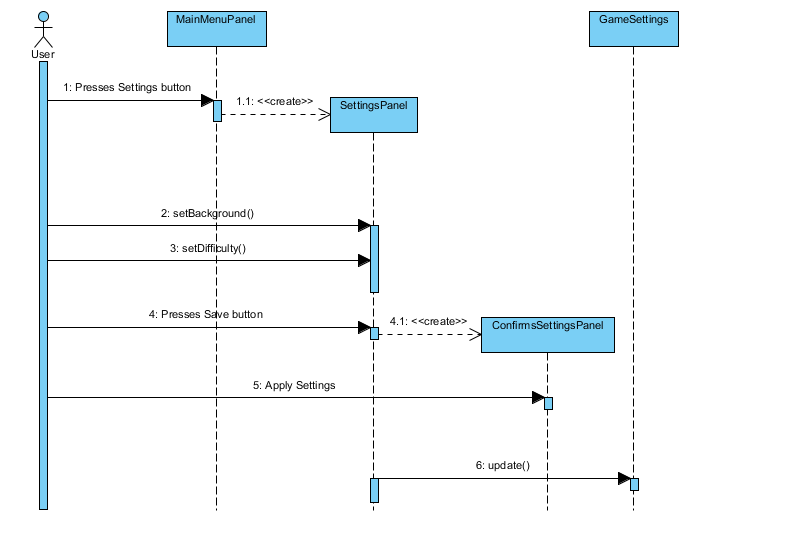


Figure Sequence Diagram for changing settings

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

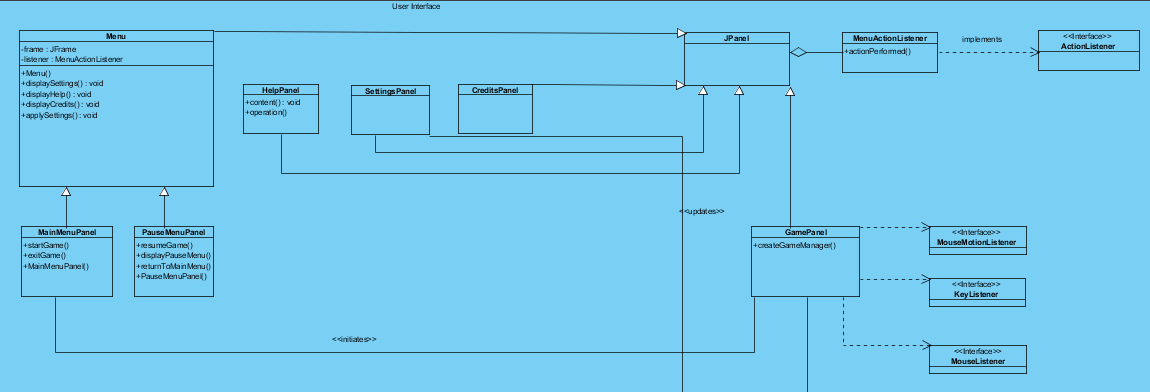


Figure User Interface Subsystem

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

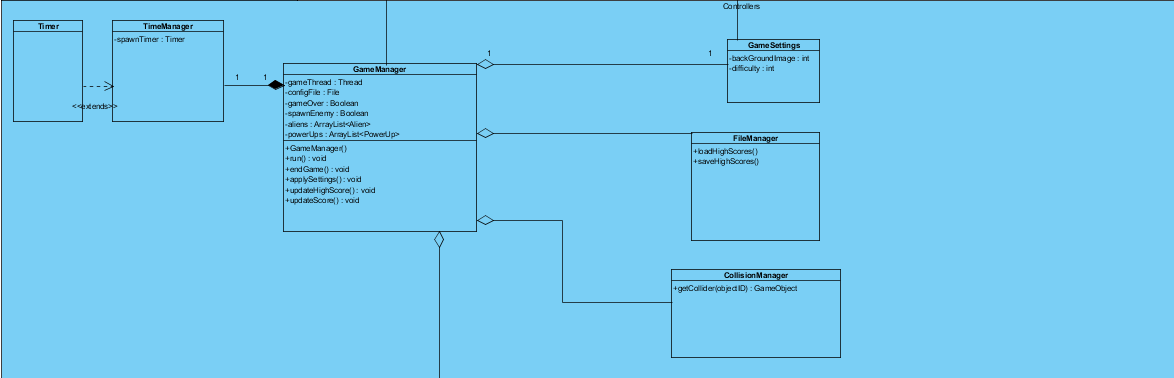


Figure Controller Subsystem

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

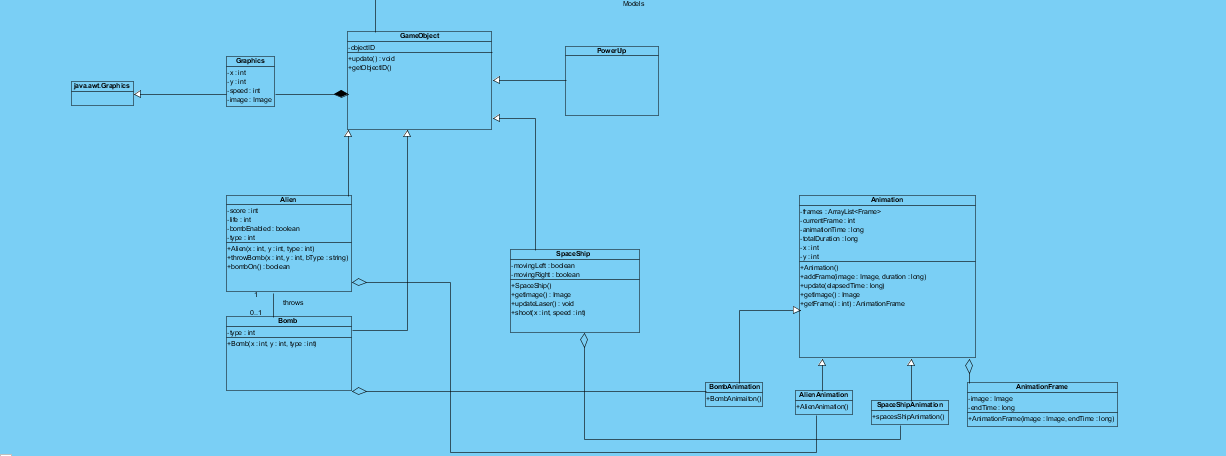


Figure Models Subsystem

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.

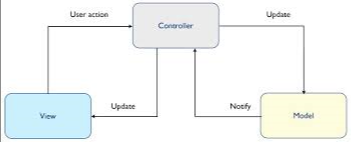


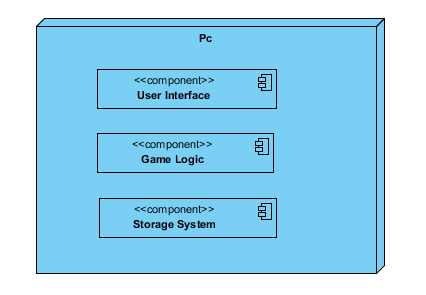
Figure MVC Architecture

### 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.



## 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. 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