Bilkent University

Department of Computer Engineering

CS319

Object Oriented Software Engineering

Final Report

Crazy Space War

Group Members

Ömer Hancı	21001218
Muhammed Emin Öztürk	20901349
Erdinç Albayrak	21202312
Balanur İçen	21101621

Assistant Professor Can Alkan

December 25, 2014

This report is submitted to the Department of Computer Engineering of Bilkent University in partial

Table of Contents

1.Introduction

We have decided to design simplified version of the space war game which we called crazy space war. The aim of the game is destroying aliens by controlling a spaceship and to be survived until end of the game. We can also obtain some bonus features by colliding some special flags, which call power up. The way to win this game is destroying several types of aliens. The game will be a desktop application and will be controlled via keyboard. This report includes an overview of the game and explains functional & non-functional requirement and system model.

2.Case Descriptions

Crazy Space War is a kind of arcade game, which is uncomplicated to play because just controlled via keyboard. Yet it is very addictive. The player controls a spaceship along screen from down to up and right to left. The game includes several levels that we have to complete to win the game. In order to complete a level, user has to destroy all of the aliens in a level. Some levels contain certain number of aliens however other levels have time limit that you have to survive up to level is completed. The goal of the game is to move up the levels without losing all energy.

Lets describe ways in order to destroy enemies' spaceships in game. As first step the User collides spaceship with enemies so as to explode enemy's spaceships. In this case our spaceship also loses its life energy. Other way is to fire them with our weapons. As noted above the game contains several kind of power up, which appears randomly on the screen. Some of power up can increase our weapon destruction power. Power-ups make the game more enjoyable than you expected.

The game has sound and graphic effects to emphasize certain events like dying or passing the level. We also will add several in-game music.

2.1.Start

The user can start new game as a single player. However, he/she can change to multiplayer choice from setting menu. The game provides user to change game setting, view help and obviously the choice of quit game any time she/he wants.

2.2.Gameplay

The player uses keyboard to play the game. The arrow buttons and space button are needed during the game. Arrows buttons are used to direct moving on the screen. Space button is used for firing to kill enemies target. In the case of multiplayer choice, instead of arrow keys the letter button "W", "A", "S", ”D" on the keyboard are used by second player. Second player uses tab button for firing

2.3.Levels

The game has 10 different levels and 3 different difficulty degrees i.e. easy, medium and hard. The difficulty can be set at change setting submenu before starting playing game. The levels get harder from first to last. End of the some 2-3 levels special aliens space ship is appeared. The user must defeat this type of enemies so as to complete level successfully and continue the other levels. At each level there is different background that represent emotion of the game.

2.4.Aliens & Spaceships

Depends on the level, different number of aliens appears and at end of the level, boss alien comes. The boss alien is five times larger than other types of alien and have more health energy so it is hard to defeat boss aliens than the others. To defeat boss aliens, power-ups play crucial roles.

2.5.Power- Ups

The power-ups are the essential part of the game that makes it so fun. The player will never know when power-up come. Also the player cannot know exactly what type of power-up coming. So it makes game more enjoyable. As it mentioned above, the level of degree triggers the difficulty of game. Especially at higher level, power-ups are indispensable requirement to complete level with high score.

In the game, there are seven types of power-ups.

2.5.1.Power-Ups "Armor"

After taking this special type of power-ups our spaceship is not be affected by enemies bomb. During 45 seconds spaceship have armor that protect player from enemies bomb so you can ignore the enemies fires.

2.5.2.Power-Ups -Attractive "Gun Power"

This makes our spaceships gun more destructive. After taking this special power ups during 60 seconds player can easily destroy aliens.

2.5.3.Power-Ups- Freeze Aliens

The type of power ups "Freeze Aliens" freezes all aliens. It makes them disable to fire and move. This feature will be active during 15 seconds. This power up provides important chance to destroy all aliens at short time.

2.5.4.Power Ups-Double Gun

This power-up presents extra gun to users spaceship. Then our spaceship can fire along two different directions so that destroy two aliens at the same time. This special features will be active until the end of level.

2.5.5. Power Ups- Mirror of spaceship

After gaining this power ups user's spaceship shadow has appeared at its symmetric coordinates with respect to y-axis. The shadow spaceship imitates users spaceship and move wherever our spaceship goes.

2.5.6.Power Ups-Block Hole

This power ups creates black hole so as to pull aliens into hole. When aliens approach black hole, immediately disappeared because of black holes gravity. On the screen, block hole is represented by special images. It will be active during 20 seconds after being created.

2.5.7.Power Ups-Speed Up

This type of power up increases spaceship's speed as approximately 2 times. The spaceship move has been more flexible after taking this power up. It can easily escape from enemies bomb, handle time limit which is requirement some level.

3.Requirement Analysis

3.1. Functional Requirements

-The user must be able to start a new game.

-The game must provide multiplayer support for two people.

-The user must be able to pause during gameplay indefinitely.

-The user must be able to change sound level and music volume during a pause.

-The game must score the player for every completed game.

-The user must be able to view a list of high scores.

-The game must provide a tutorial containing information about power-ups, aliens and controls.

3.2.Non-functional requirements

-The game must not require any documentation to play.

-The game must lose the current score in the event of a failure.

-The game must register user actions and produce output within 0.5 seconds.

-The game must allocate less than 16 MB of space during runtime.

-The game must run on every operating system.

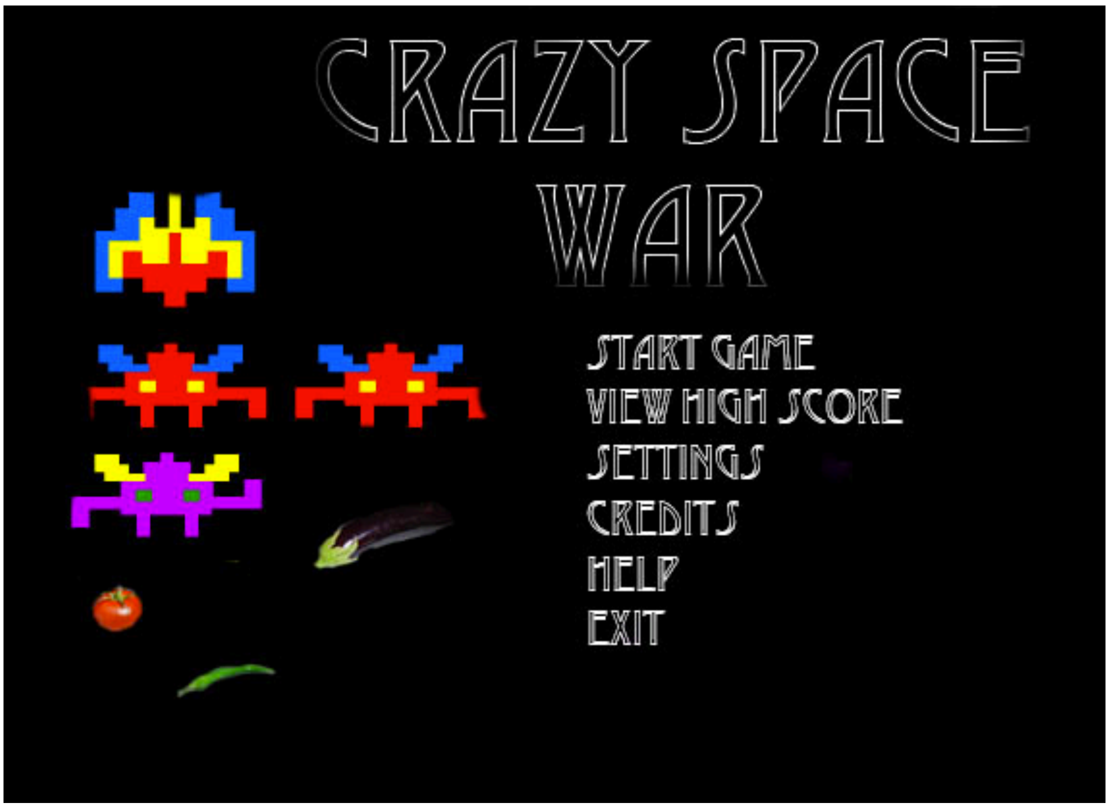
-The game must not require an installation.

3.3.Constraints

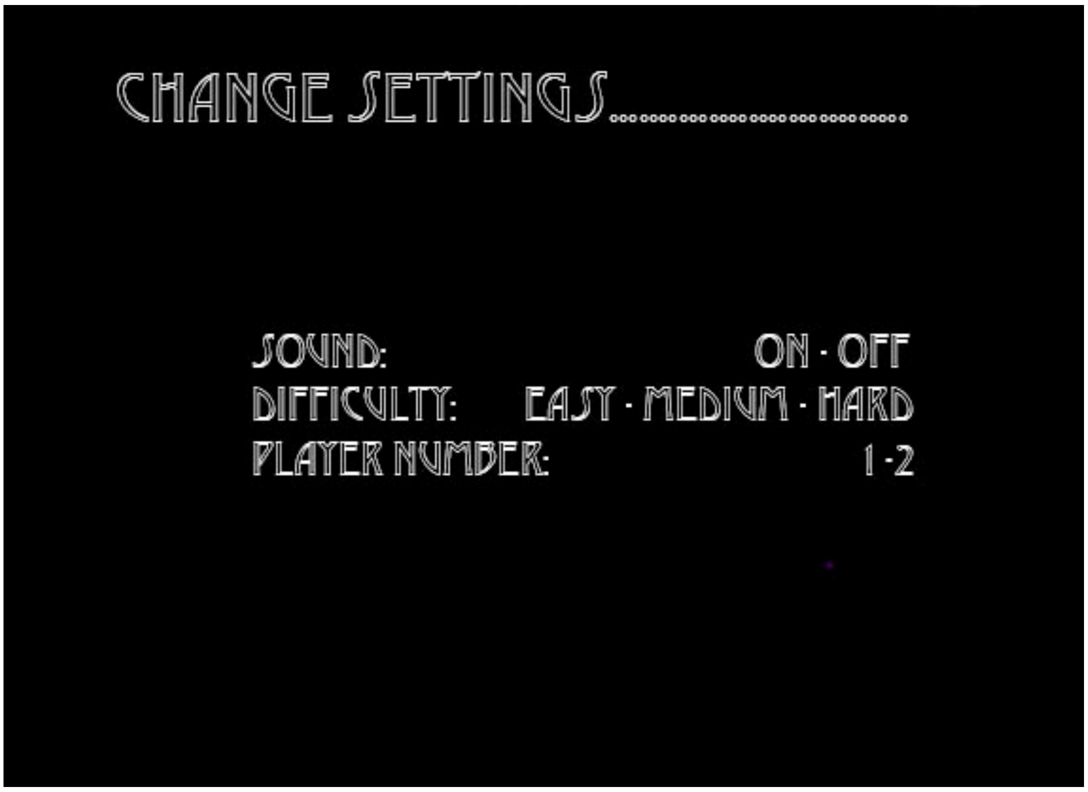
-The implementation language must be Java.

3.4. User Interface

3.4.1. Main Menu

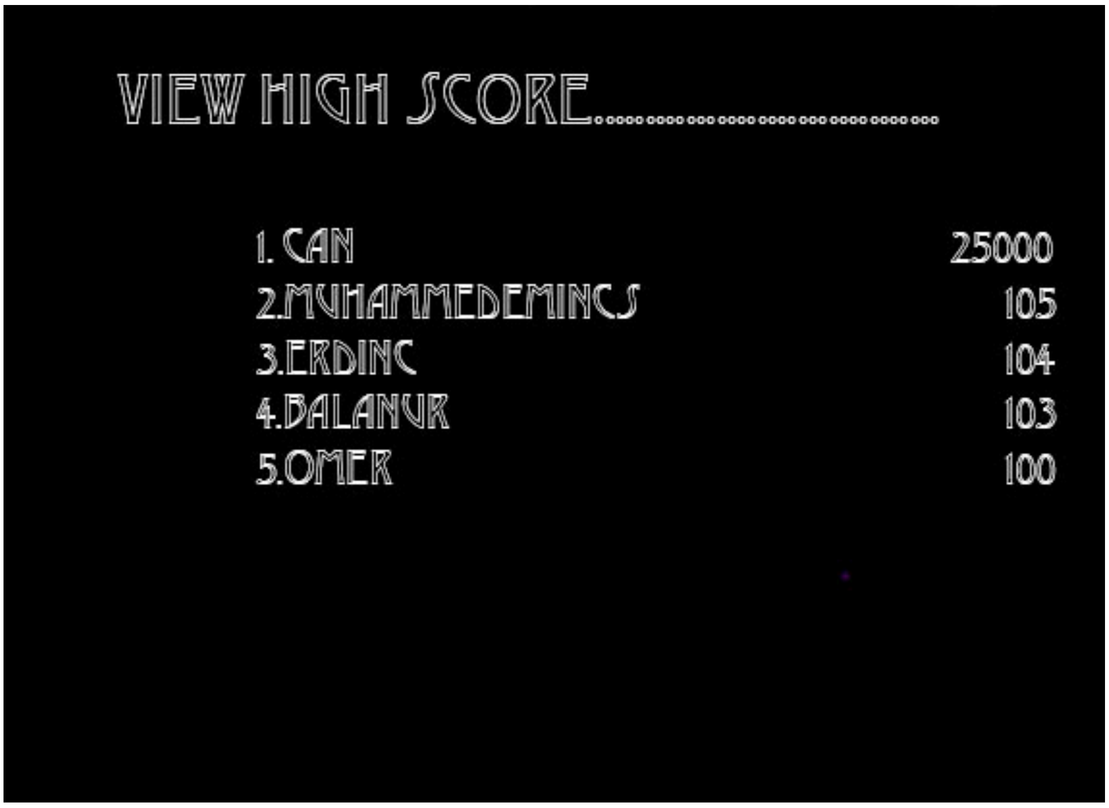


3.4.2. Change Settings


CHANGE SETTINGS.........................

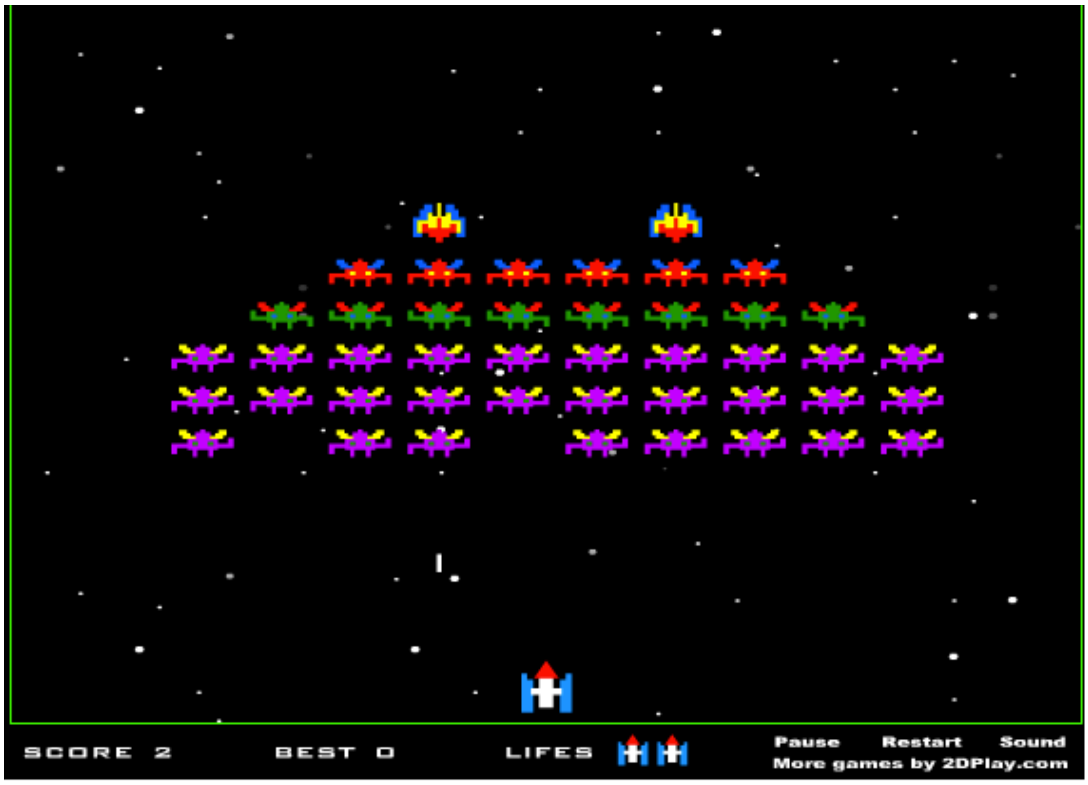
SOUND: ON - OFF
DIFFICULTY: EASY - MEDIUM - HARD
PLAYER NUMBER: 1 - 2

3.4.3. View High Score


VIEW HIGH SCORE..............................

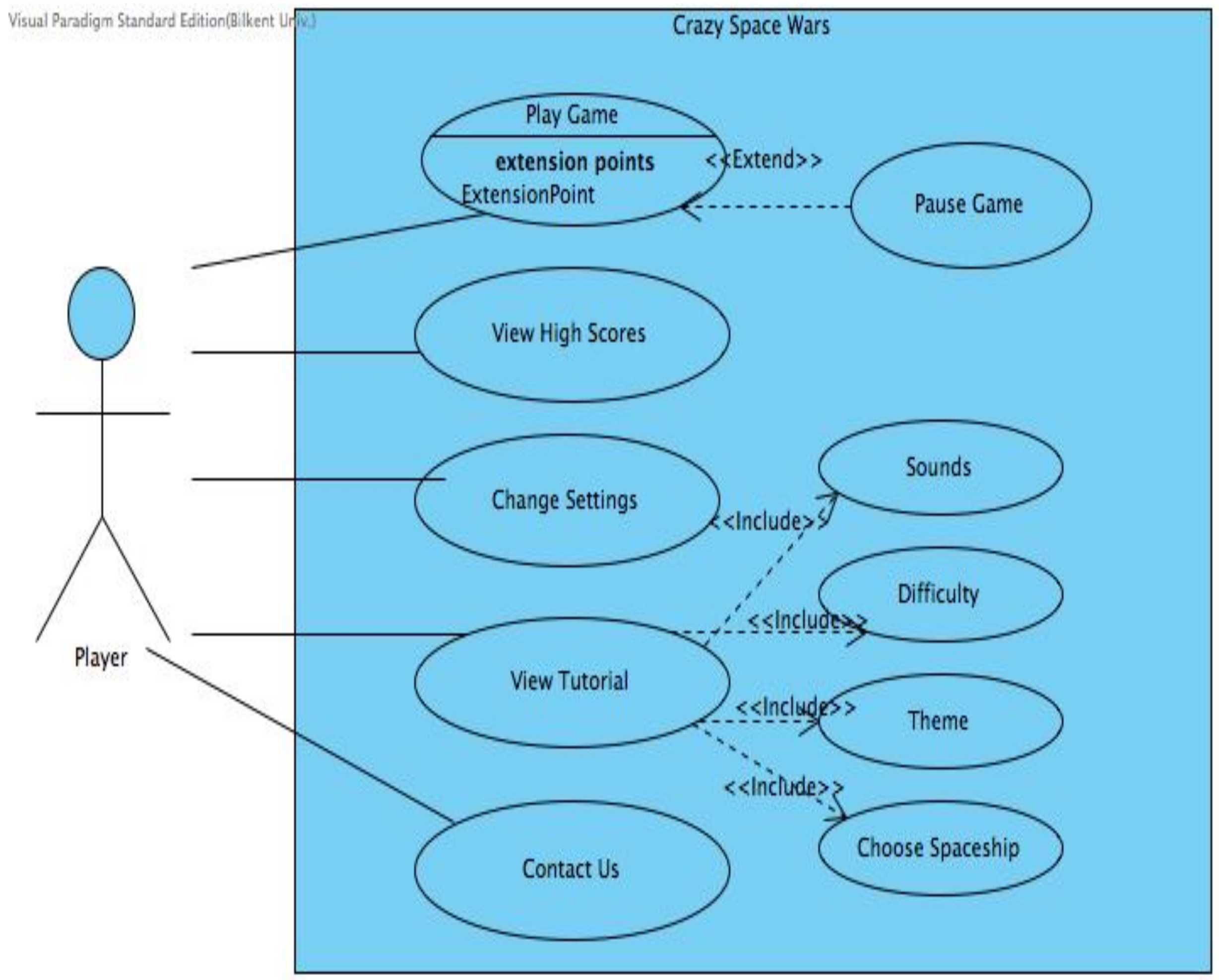
1. CAN	25000
2.MUHAMMEDEMINCS	105
3.ERDINC	104
4.BALANUR	103
5.OMER	100

3.4.4. Play Game



4.System Model

4.1.Use Case Model



4.1.1.Play Game

Use Case Name: Play Game

Primary Actor: Player

Stakeholders and Interests:

-Player aims to complete levels and make highest score by killing enemy spaceships.

-System keeps the score of the Player.

Pre-condition: After player selects the Play Game, s/he chooses single/multi player options.

Post-condition: If player gets a higher score than ones in the high score list, System adds new score in the list.

Entry Condition: Player selects "Play Game" button from Main Menu.

Exit Condition:

- ✓ Player chose to exit the game.
- ✓ Player lost the game (no more lives or time up).
- ✓ Player completed all levels successfully.

Success Scenario Event Flow:

1. Player chose single/multi player option.

2. Player starts the game.

3. Game is started from the first level.

4. Player plays until the level goals are completed. Goals maybe completing some time without getting killed or shoot specific number of enemies according to level.

5. System constructs the next level.

6. Player starts playing next level.

*Player repeats the steps 4 – 6 until all levels are completed or Player loses all lives.

7. System displays the score of the player. If high score is enough to get into high score list, user enters name.

8. System returns to Main Menu.

Alternative Flows:

A. If player desires to pause the game at any time:

A.1. Player selects "Pause" button.

A.2. System shows a list of options during pause time. This pause game screen blocks all view of game to prevent extra thinking time.

B. If player desires to exit the game and return to main menu:

B.1. Player selects "Pause" button.

B.2. System shows a list of options during pause time which includes "Exit and Return Screen".

B.3. Player selects the "Exit and Return Screen".

B.4. System goes to 7th step of success flow.

4.1.2.Pause Game

Use Case Name: Pause Game

Primary Actor: Player

Stakeholders and interests:

-Player wants to pause to game for a while.

Pre-condition: Player must be playing the game.

Post-condition: Player returns to game without losing any information in the game.

Exit Condition: Player selects back from the options given for her/him.

Success Scenario Event Flow:

1. Player selects Pause from Game Screen.

2. Pause window is displayed to player.

3. User selects back to return to the game where s/he left.

4.1.3.View High Scores

Use Case Name: View High Scores

Primary Actor: Player

Stakeholders and Interests:

-Player wants to see highest 5 scores with player names.

-System shows the list containing top five scores.

Pre-conditions: System keeps records of top five scores with player names.

Post-condition: -

Entry Condition: Player selects "View High Scores" from Main Menu.

Exit Condition: Player selects "Back" to return Main Menu.

Success Scenario Event Flow:

1. System displays top 5 scores with player names.

Alternative Flows:

A. If player desires to return main menu at any time:

A.1. Player selects "Return to Main Menu" button.

A.2. System displays Main Menu.

4.1.4.Change Settings

Use Case Name: Change Settings

Primary Actor: Player

Stakeholders and Interests:

-Player can turn sound on/off

-Player can choose the difficulty level.

-Player can choose theme.

-Player can choose spaceship.

Pre-condition: Game settings are set as default. If Player changes game settings, adjusted settings will be saved and used by system.

Post-condition: Settings are updated.

Entry Condition: Player selects “Settings” button from menu.

Exit Condition: Player selects “Back to Menu” to return menu.

Success Scenario Event Flow:

1. Player presses “Settings” button to make changes about game settings.

2. Game settings are displayed to Player in “Settings”.

3. Player adjusts settings according to his desire.

4. System updates game settings successfully.

5. Player returns to the main menu by selecting “OK”.

Alternative Flow of Events:

A. Player does not change any settings (go to step 5).

4.1.5.View Tutorial

Use Case Name: View Tutorial

Primary Actor: Player

Stakeholders and Interests:

- Player opens guide and learn how to play the game from instructions.

- Menu shows a tutorial that explains how to play game, its features, rules and it gives useful tips.

Pre-conditions: Player should be in Main Menu.

Post-condition:

Entry Condition: Player selects “View Tutorial” from Main Menu.

Exit Condition: Player selects “Back” to return previous menu.

Success Scenario Event Flow:

1. Player selects “View Tutorial” from Main Menu.

2. System displays a guide about game.

Alternative Flows:

A. If Player requests to return previous menu at any time:

> **A.1.** Player selects “Back” button from “View Tutorial” screen.
>
> **A.2.** Player returns previous menu.

4.1.6.Contact Us

Use Case Name: Contact Us

Primary Actor: Player

Stakeholders and Interests:

- Player wants to reach the contact information of developers of the project.

- System displays contact information of the developers.

Pre-condition: Player should be in Main Menu.

Entry Condition: Player selects “Contact Us” from main menu.

Exit Condition: Player selects “Back” to return previous menu.

Success Scenario Event Flow:

1. Player selects “Contact Us” from main menu.

2. System displays contact information of the developers.

Alternative Flows:

A. If player desires to return main Screen at any time:

> **A.1.** Player selects to go back to return main menu.
>
> **A.2.** System displays main menu.

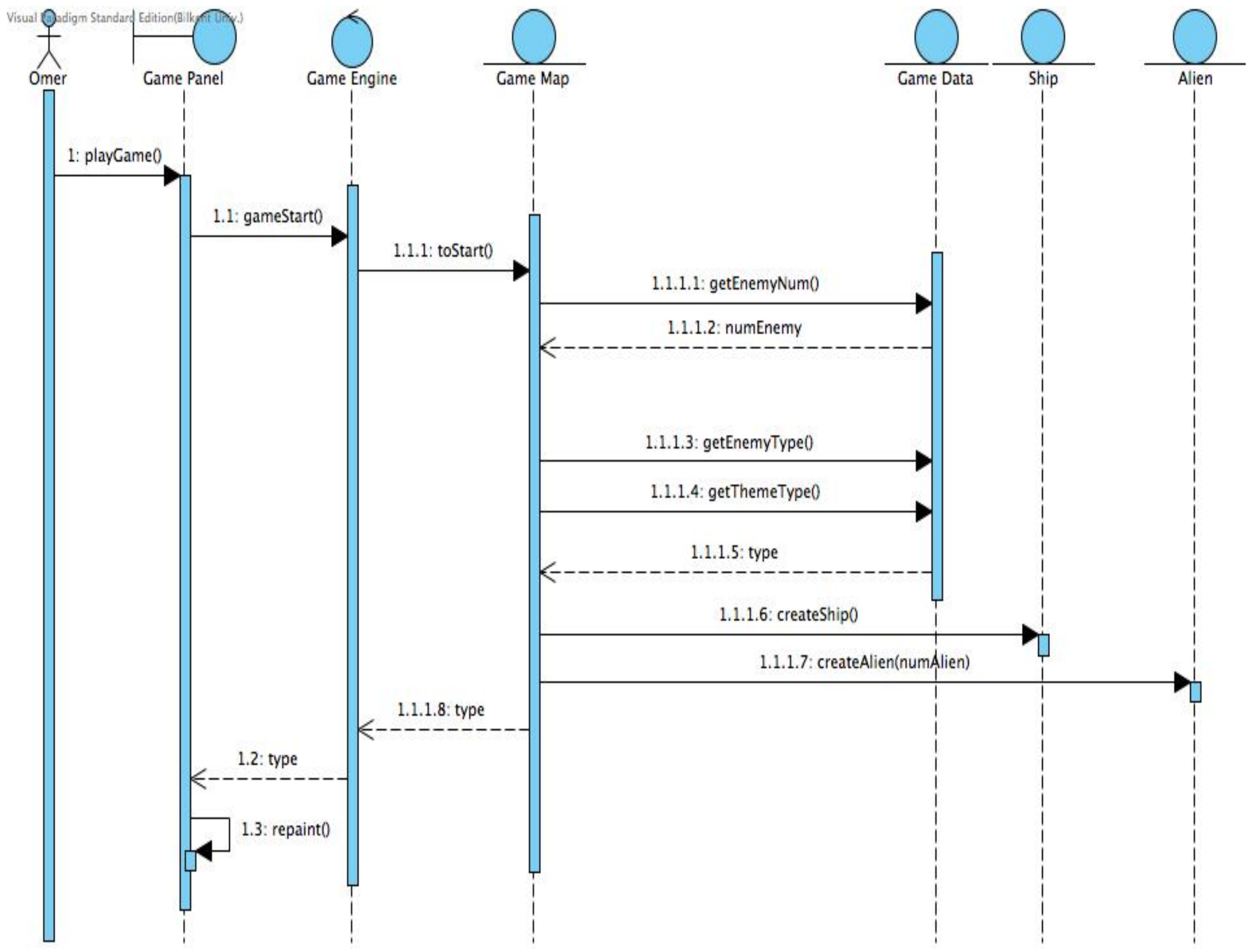
4.2. Dynamic Models

4.2.1 Sequence Diagram

4.2.1.1.Scenario Name : Start New Game

Scenario :

Omer has already downloaded this game and he is excited to try the game. So he selects "Play Game" from main screen. As a default setting choice "single" game starts. Omer wants to play alone so he doesn't change it to multiplayer from setting menu.



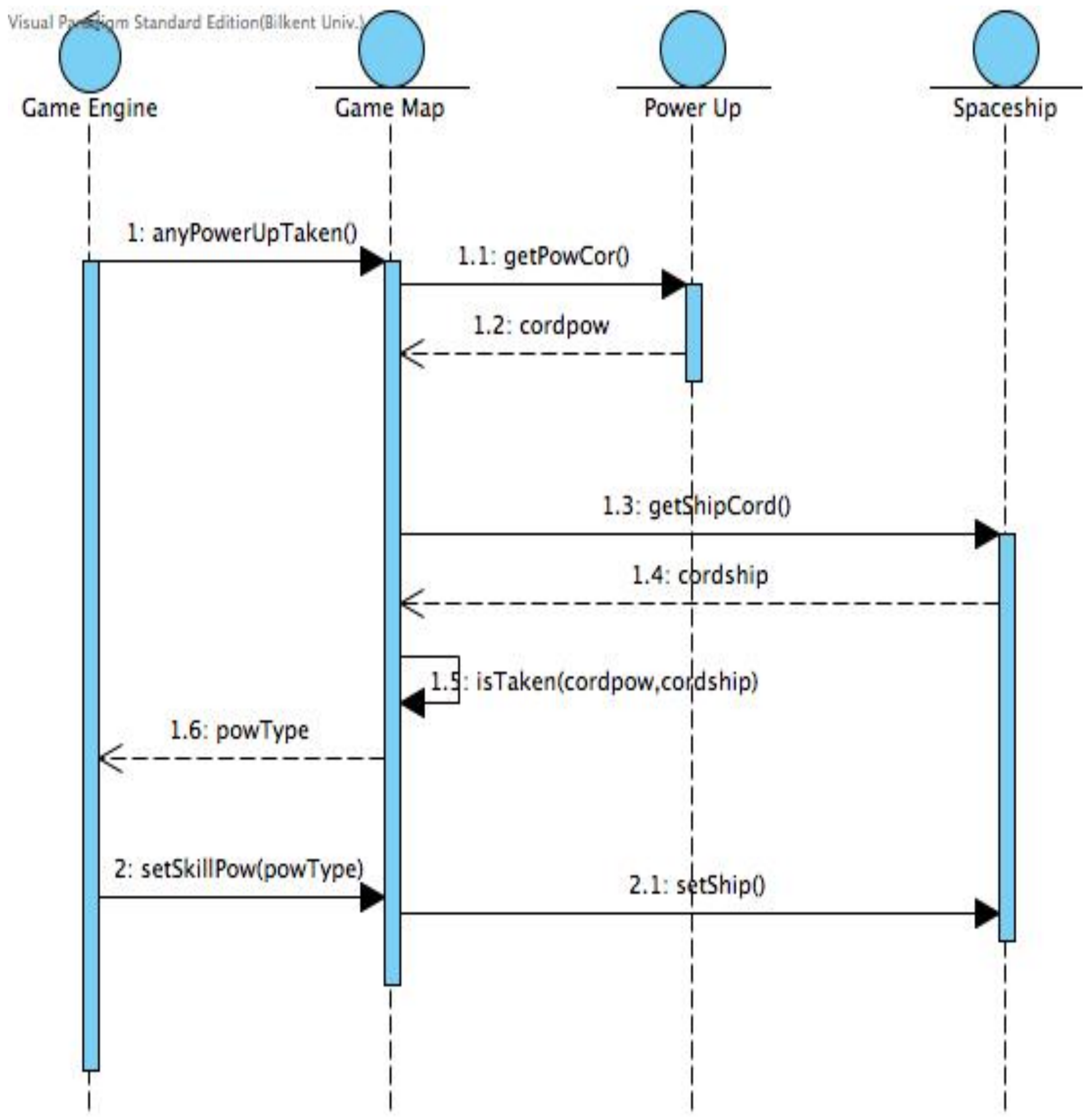
Description :

By selecting play game option Game Panel call to Game Engine that controls all models. Game Engine also calls Game Map to apply game data, which keeps information about level and setting. Game map creates aliens according to data taken from Game Data. After this process Game Panel display game.

4.2.1.2.Scenario Name : Collect Power Up

Scenario :

During the game , Omer have realized that power-up appeared. He intents to gain the power-ups to get high score and complete level. He goes to the direction where power-up is coming. And get it. According to power up type, spaceship has some features.



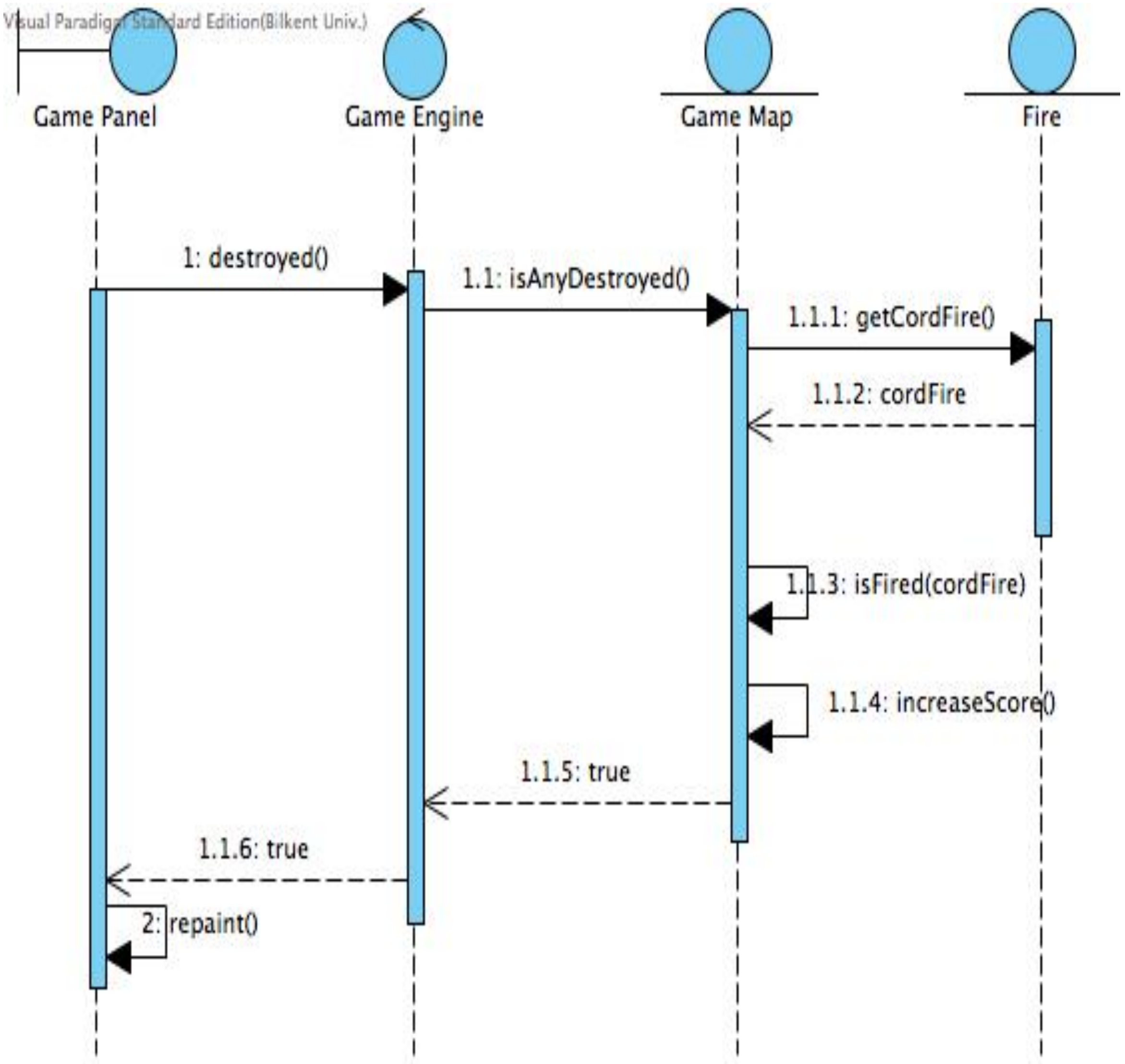
Description :

Game engine call Game Map by anyPowerUpTaken() method. Game Map take power up and spaceship coordinates. isTaken() methods in Game map checks whether power up is taken or not by comparing coordinate. If power up is taken, Game Map return power up type to Game engine. According to power up type, Game engine call Game Map so that Game Map can set spaceship.

4.2.1.3.Scenario Name : Destroy Enemy by firing

Scenario :

Omer is struggling all aliens at the game. There are two ways to kill enemies as explained above. Firstly, the user can choose to collide enemies and destroy them. However this method cause the user's spaceship lose health energy. Second way is focusing target and firing. For this reason, Omer prefers to fire.



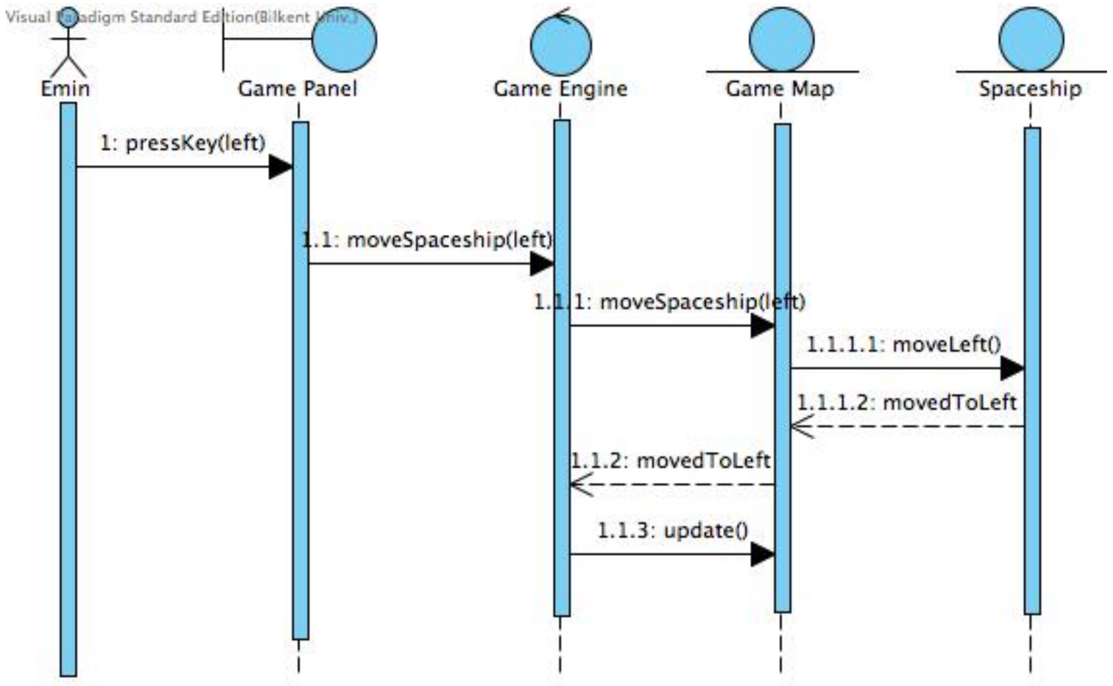
Description :

At any time there are so many bullets on screen, which are fired user's spaceship. In this scenario it is controlled that any enemy ship is destroyed from any bullet fired by users spaceship. Game Panel call Game Engine. Game engine applies Game Map via isAnyDestroyed method. Game Map describe fires coordinates and controls that any enemies ship are hit. Game Map class includes vectors which keeps all enemies object. Isfired method in Game map defines which aliens ship is destroyed. After this method, It triggers to increaseScor method to give point the user.

4.2.1.4. Scenario Name: Move Spaceship

Scenario:

Emin starts to game and tries to move by pressing arrow keys.



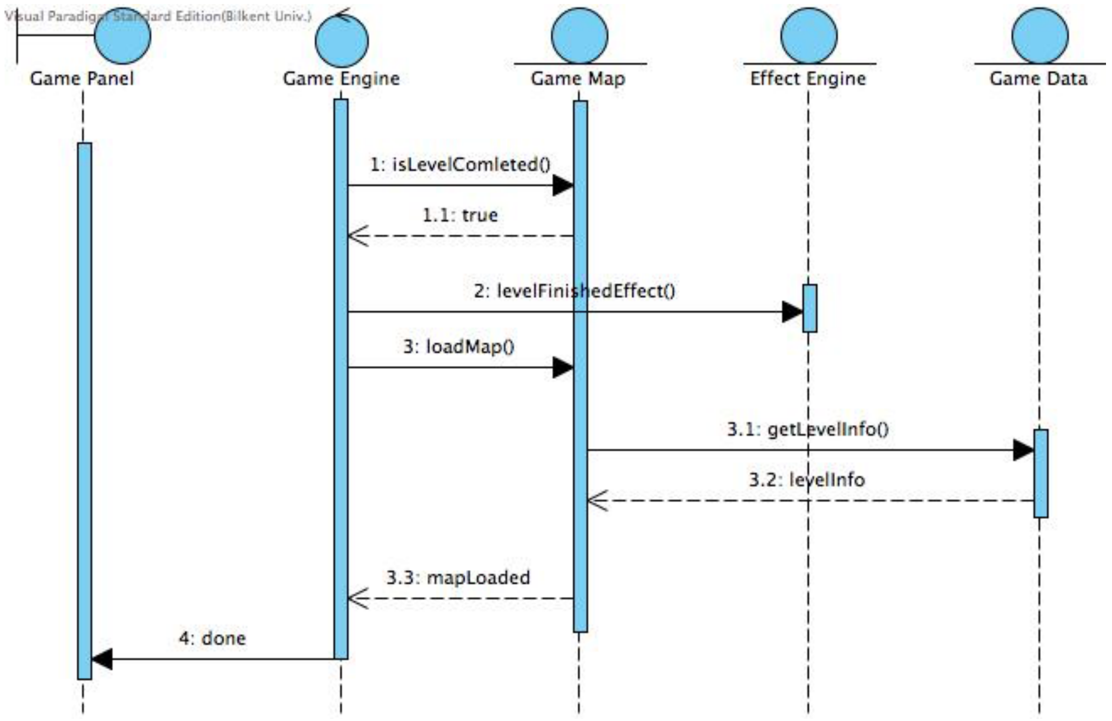
Description:

When the user presses to arrow keys, the signal goes to our boundary object which is Game Panel, Game Panel sends the message to the controller which is Game Engine and Game Engine forwards this message to the Game Map, Game Map calls the moveLeft() method of the Spaceship object to change its coordinates. Then Spaceship changes its coordinates and returns that to Game Map, Game Map forwards that to Game Engine and Game Engine updates the map.

4.2.1.5. Scenario Name: Level-up

Scenario:

Emin is very successful while playing the game and finished one level successfully and goes next level of the game.



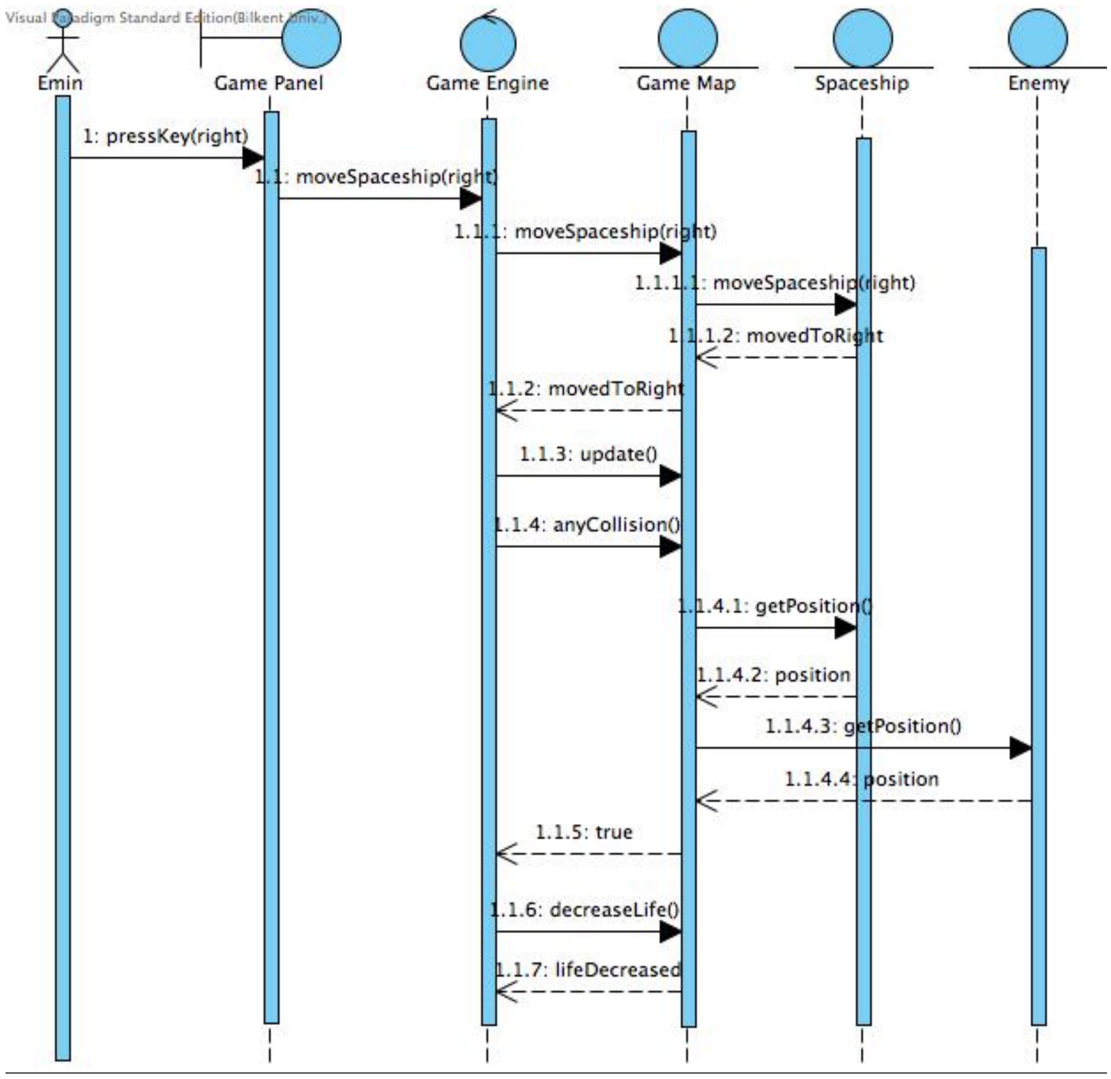
Description:

Game Engine controls the Game Map to learn whether the level is done or not. Because the level is up, Game Map returns true and then Game Engine calls Effect Engine to make the sound effect of ending the level successfully. Then calls Game Map to load new map, Game Map asks to the Game Data about new level information and calls the getLevelInfo() method and gets it then sends the signal of mapLoaded to the Game Engine and Game Engine sends the signal “done” to the Game Panel.

4.2.1.6. Scenario Name: Die From Collision

Scenario:

Emin is brave warrior in the game, but one of the bullets of the enemies collides with the Emin’s spaceship and Emin loses one of his lives.



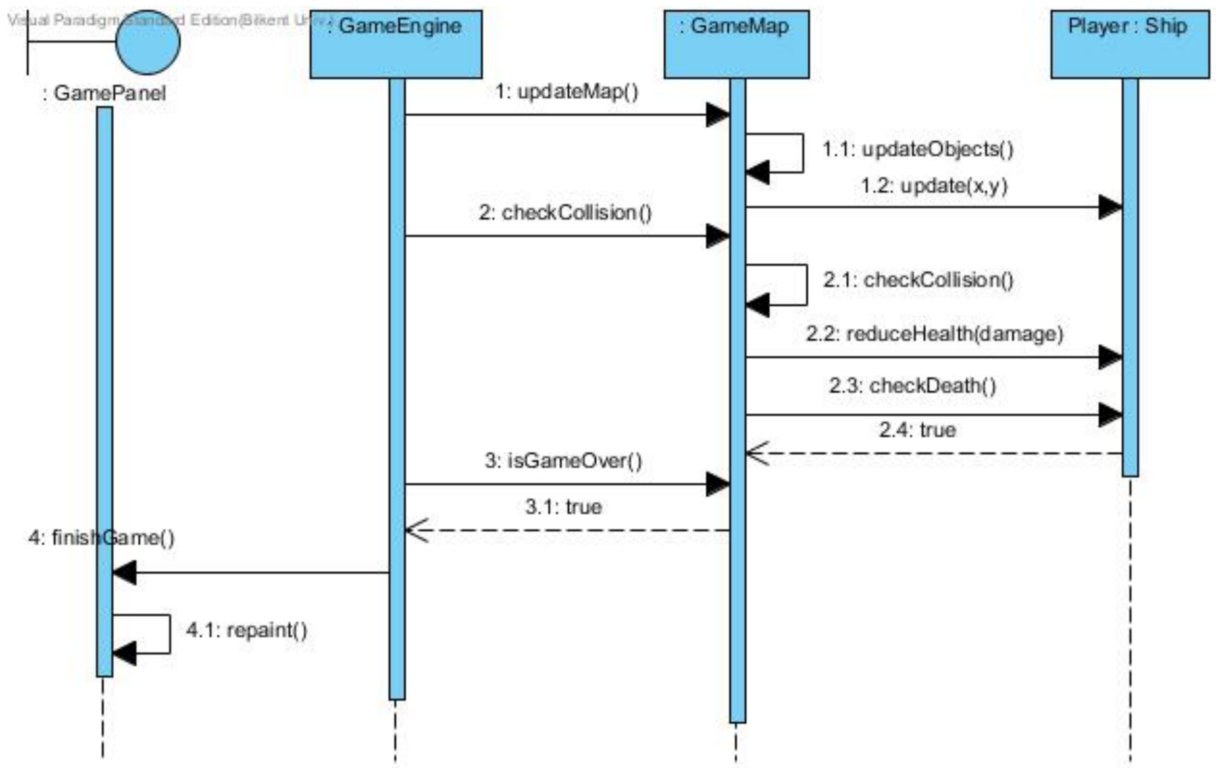
Description:

Emin presses to arrow key and moves after movement, Game Engine checks for any collision by calling the Game Map with anyCollision() method. Game Map gets positions of Spaceship first and then get positions of enemies and if it finds a match sends true to the Game Engine. Game Engine calls decreaseLife() method and Emin's one of lives is decreased.

4.2.1.7.Scenario Name :Die from Projectile

Scenario :

While Erdinç is playing the game, his ship is hit by an enemy projectile. The ship is out of energy and is destroyed. Then the game is over and the session is finished.



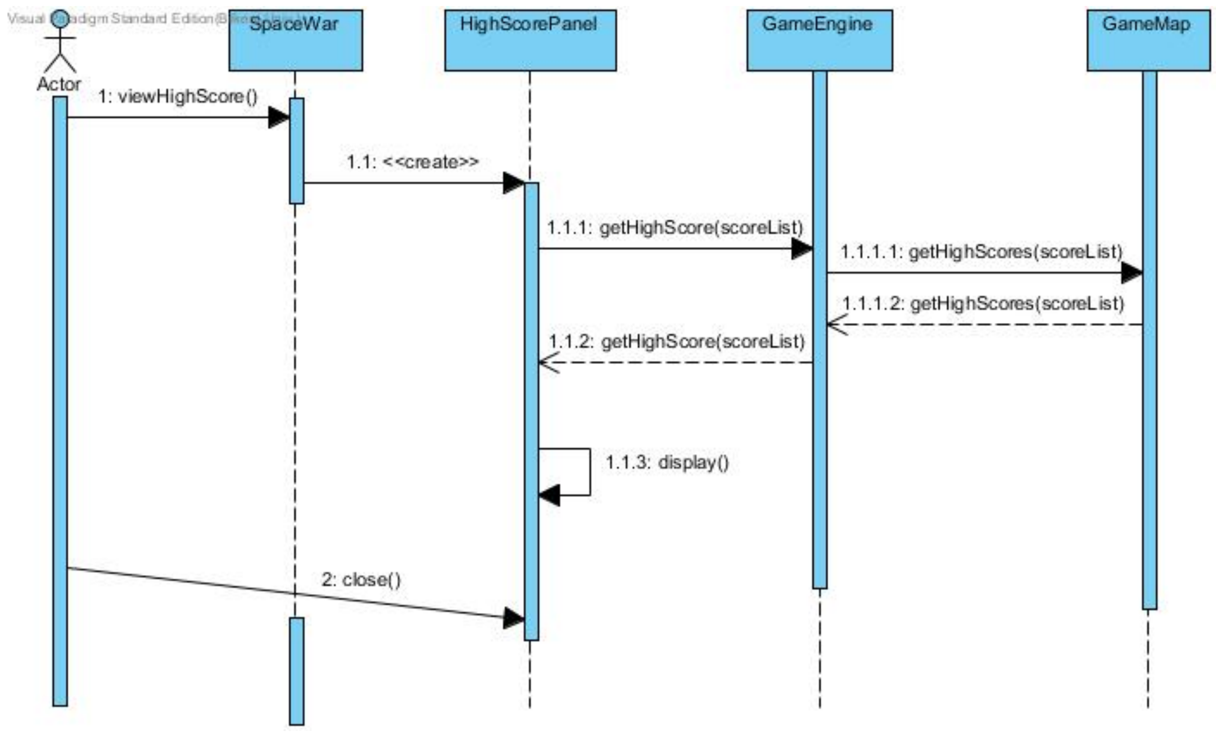
Description:

GameEngine periodically commands GameMap to update the map and check for collision between all instances of GameObject. When a collision is detected, GameMap handles the collision. Here the player is hit by an enemy bomb so GameMap reduces the Ship object's health. After this, GameMap asks whether the Ship object is out of health. If the Ship object returns true, the game is over. GameEngine is notified that game is over. GameEngine notifies boundary objects that game is over.

4.2.1.8.Scenario Name : View High Scores

Scenario :

Erdinç wants to view the high scores list. Erdinç clicks the High Scores button in the main menu. The game displays the high scores and Erdinç stays in this screen for a while. Then he clicks the exit button to go back to the main menu.



Visual Paradigm Standard Edition
Actor
SpaceWar
HighScorePanel
GameEngine
GameMap
1: viewHighScore()
1.1: <<create>>
1.1.1: getHighScore(scoreList)
1.1.1.1: getHighScores(scoreList)
1.1.1.2: getHighScores(scoreList)
1.1.2: getHighScore(scoreList)
1.1.3: display()
2: close()

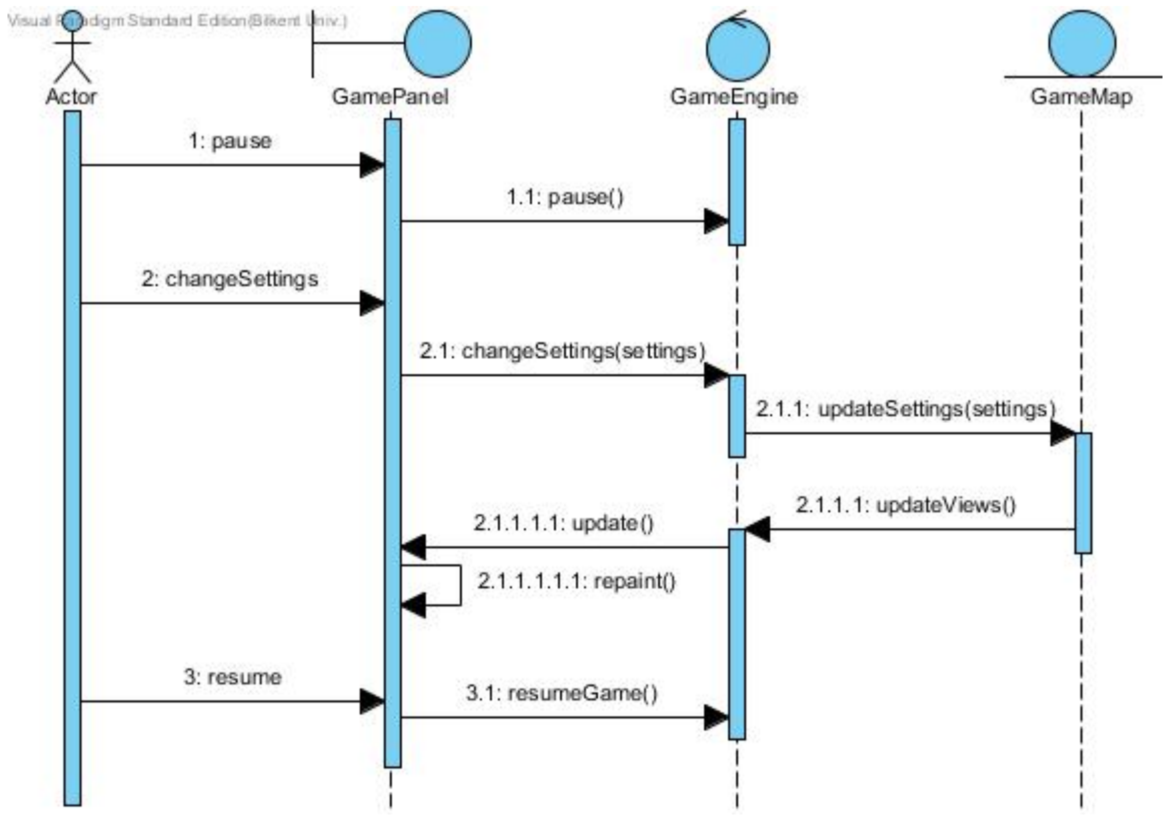
Description:

When the user clicks High Scores in main menu, SpaceWar creates an instance of HighScoresPanel. HighScoresPanel asks GameEngine to get the list of high scores, and GameEngine asks GameMap for the list. When GameMap fetches and delivers the list to GameEngine and GameEngine to HighScoresPanel, HighScoresPanel is complete and displays relevant information. When the user is done, the user closes the window and SpaceWar takes control.

4.2.1.9.Scenario Name : View High Scores

Scenario :

While Erdinç is playing the game, he pauses the game. The game displays the pause menu and Erdinç chooses to change the settings. After Erdinç changes the settings, game takes him back to the pause menu. Next, he clicks Resume Game and continues to play the game.



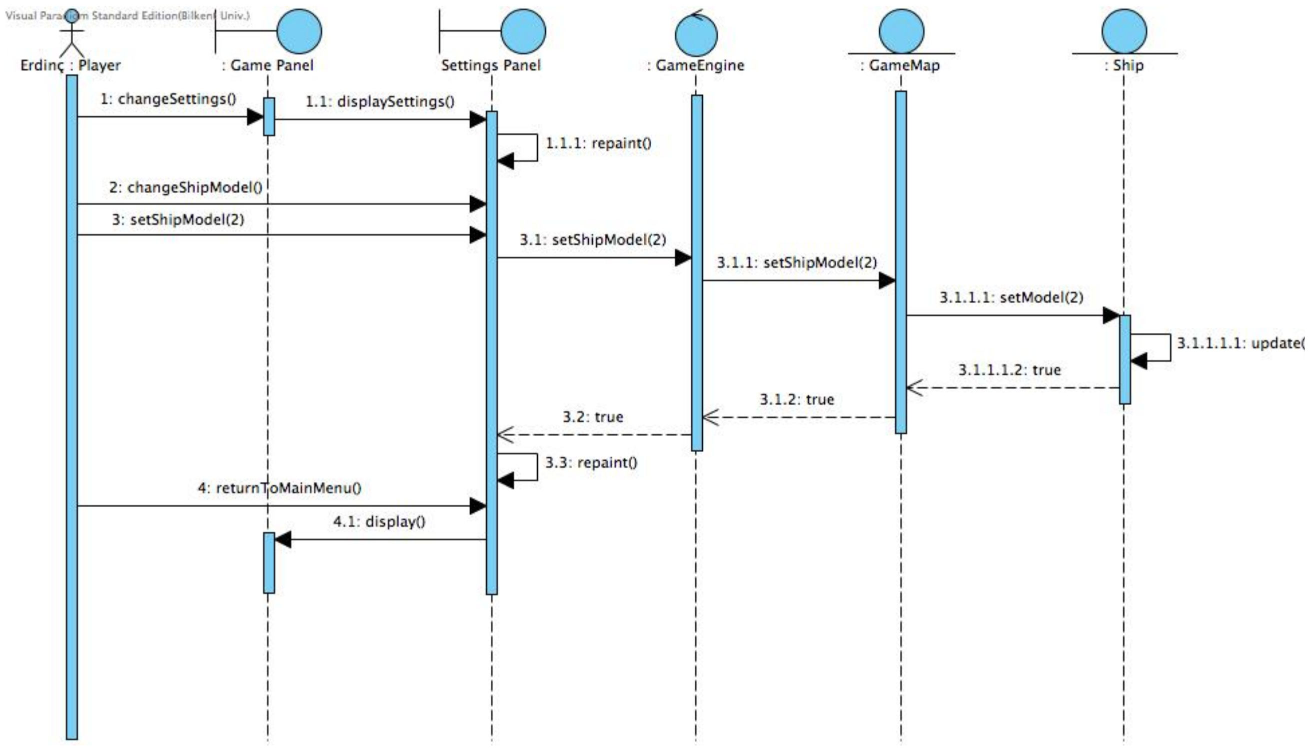
Description:

The user clicks the hotkey for pausing the game. GamePanel tells GameEngine to pause the game so the GameEngine starts to wait until the next order from user. The user changes settings from the pause menu and submits the settings form. GamePanel delivers the form to GameEngine to change the settings. GameEngine commands GameMap to update the settings according to new information. When GameMap is done, it notifies GameEngine. Then GameEngine notifies boundary objects to update. Next, user clicks resume so GamePanel tells GameEngine to resume the game.

4.2.1.10.Scenario Name: Change Settings

Scenario:

In this scenario, player Ayşe has already started the game and looks at the main screen. Ayşe wants to change ship model from game settings; she selects change settings from the main menu. System *display settings menu. Ayşe selects change ship model option. System show possible ship models. She selects one of them. System updates the ship model. She returns to main screen by selecting return to main menu.*



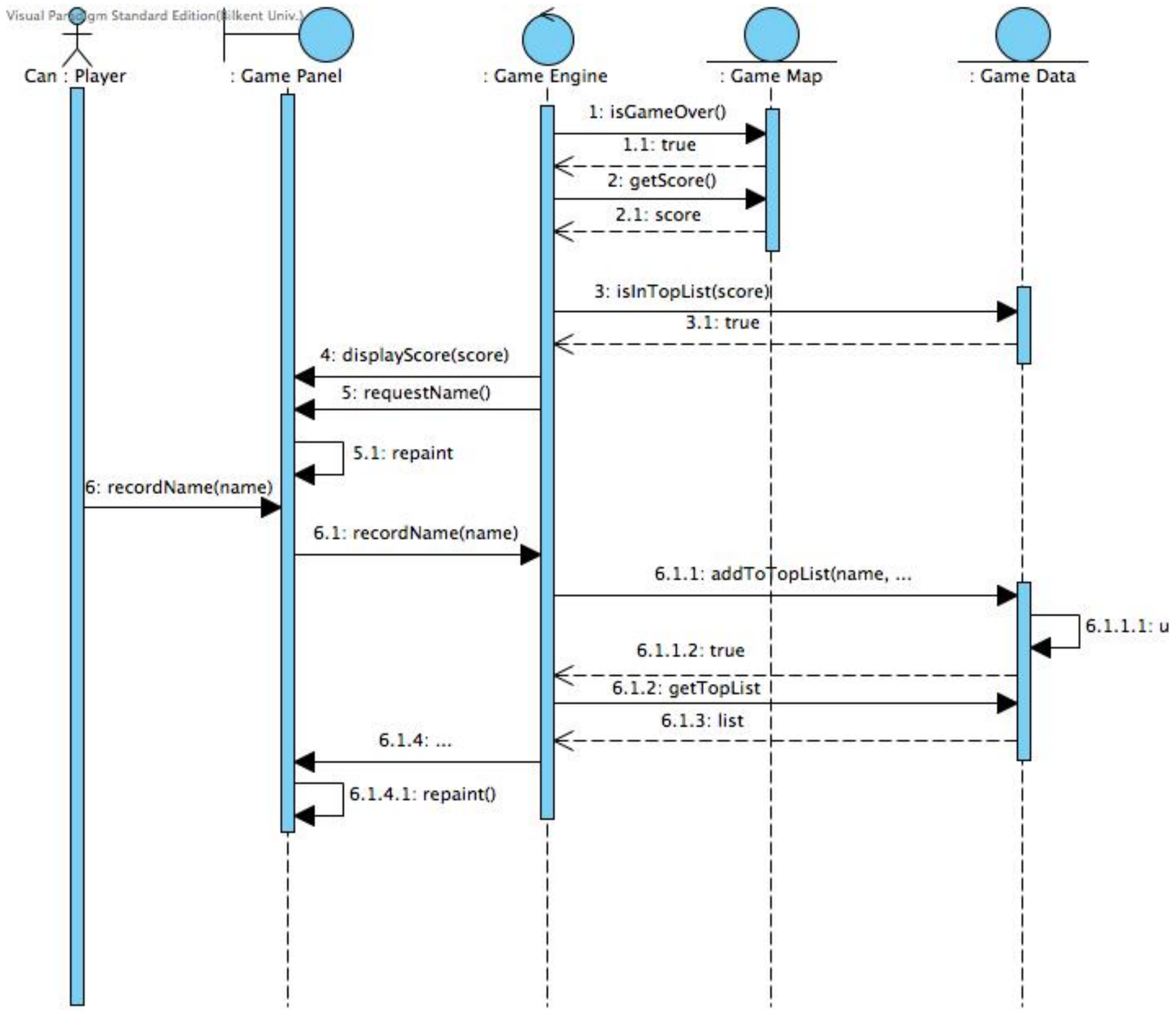
Description:

Ayşe sends a request to Game Panel object by invoking the method changeSettings. Then, Game Panel invokes displaySettings method of Settings Panel. Settings Panel invokes repaint and displays settings menu. Ayşe requests to change ship model and calls changeShipModel method of Settings Panel. Settings Panel repaints and shows available ship models. Ayşe selects 2nd model and calls setShipModel method. Settings Panel calls setShipModel of Game Engine. Then, Game Engine calls setShipModel of Game Map. Finally, Game Map invokes setModel of Ship. Ship calls update method. Invoked setShipModel methods return true until Settings Panel. Ayşe calls returnMainMenu method from Settings Panel. Main menu is displayed.

4.2.1.11.Scenario Name: Make High Score

Scenario:

In this scenario, Can has already finished the game. System displays her score on the screen and check whether it's in the top 5 list. System confirms that she made the top list. System asks for her name for record. She enters her name. System updates the ranking with her name and score. Then, system displays updated list on the screen.



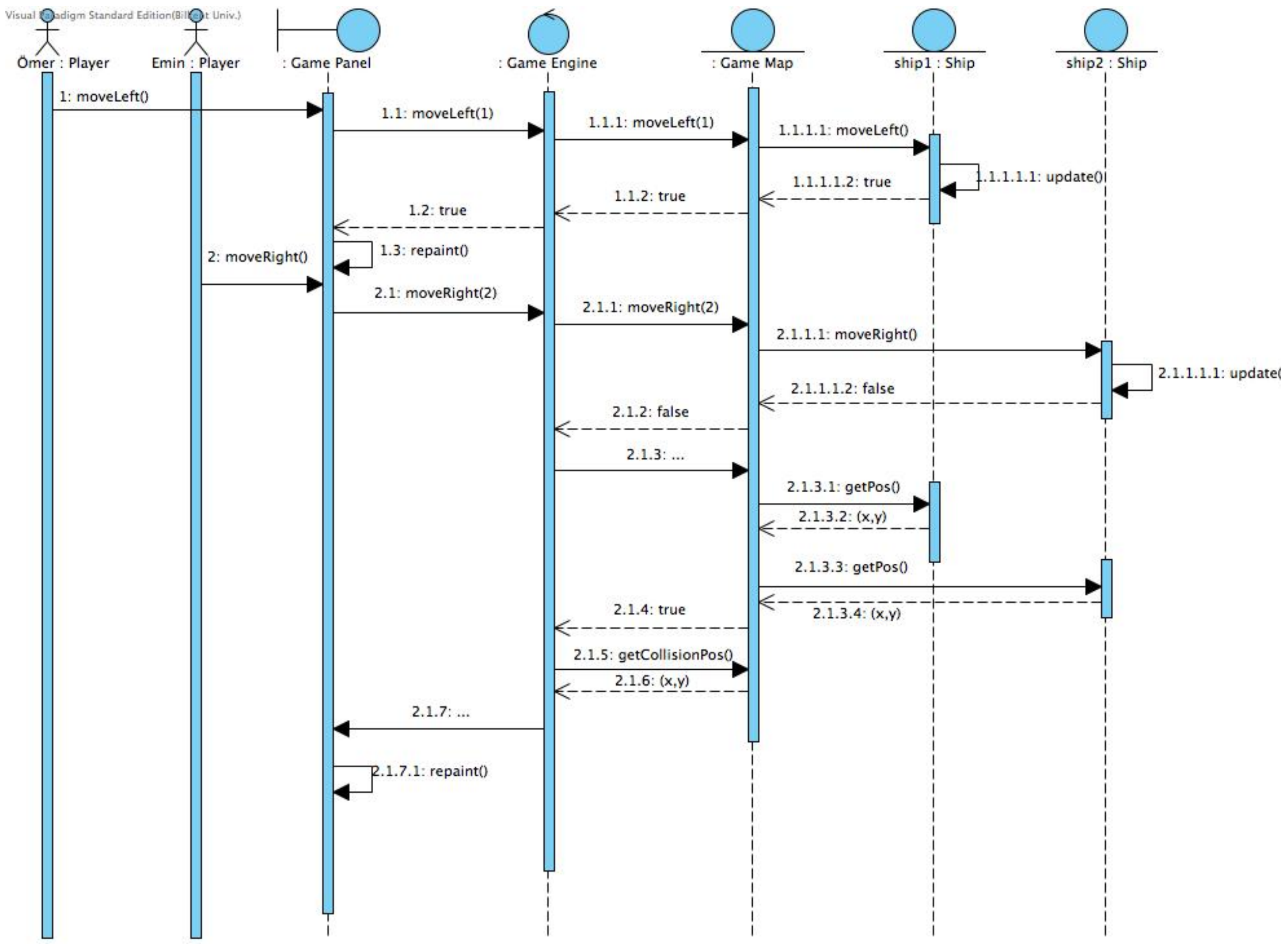
Description:

At some point of the game, Game Engine invokes isGameOver method of Game Map and it returns true. Then, Game Engine invokes getScore method. Game Map returns score. Game Engine invokes isInTopList method of File Manager. File Manager returns true. Then, Game Engine calls displayScore of Game Panel. And Game Engine also invokes request Name of Game Panel. Game Panel self-invoke repaint method. Then, Ayşe enters her name. Then, Game Panel calls recordName method of Game Engine with the parameter name. Game Engine calls addToTopList of File Manager with parameters name and score. Then, File Manager self-invokes updateList method with name and parameter and return true to Game Engine. Game Engine calls getTopList of File Manager. File Manager returns top list to Game Engine. Game Engine calls displayList of Game Panel with the parameter top list. Finally, Game Panel self-invoke repaint method.

4.2.1.12.Scenario Name: Multi Player Mode – Ship Collision

Scenario:

While playing the game in multi-player mode, Ömer moves his ship to left. System updates its position. Then, Emin tries to move his ship to right. Then, system notifies a collision and display a collision effect. None of the players loses their energy or get any points.



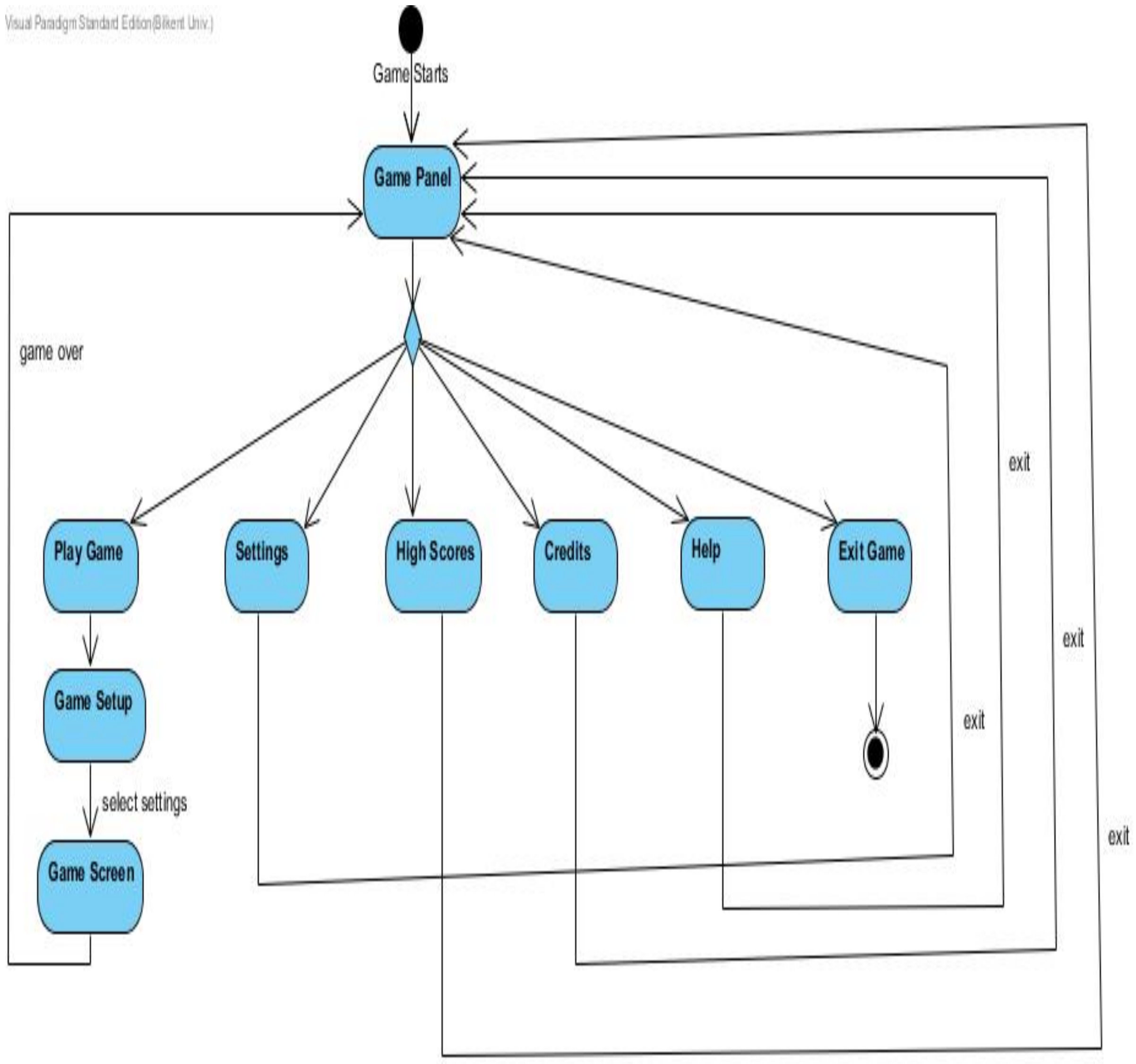
Description:

Player Ömer calls moveLeft of Game Panel. Game Panel calls moveLeft method of Game Engine with parameter 1. Game Engine invokes moveLeft of Game Map with parameter 1. Game Map calls moveLeft of ship1. Ship1 updates and return true. Then, Emin calls moveRight of Game Panel. Game Panel calls moveRight method of Game Engine with parameter 2. Game Engine invokes moveRight of Game Map with parameter 2. Game Map calls moveRight of ship2. Ship2 updates and return false since the position is already occupied. Then Game Engine calls shipCollision method of Game Map. Then, Game Map invokes getPos methods for both of the ships and compare their positions. Game Map returns true to Game Engine. Game Engine calls getCollisionPos method of Game Map. Game Map returns position of collision. Game Engine invokes displayCollisionEffect of

GamePanel with this position. Finally, Game Panel self-invokes repaint.

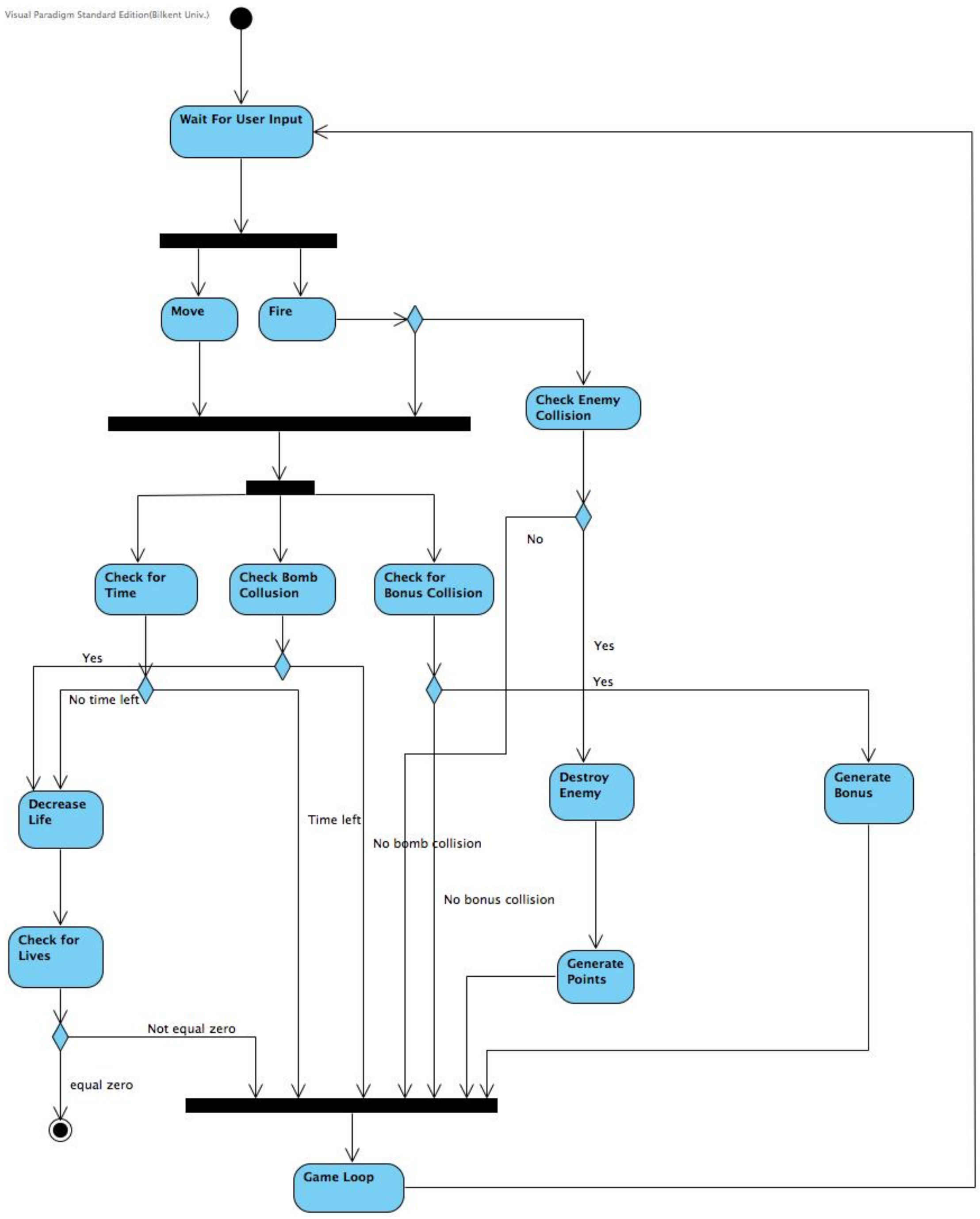
4.2.2. Activity Diagram

4.2.2.1.User Interface Navigational Path



Game Panel is the main menu where the user can view the state of the game. All of the interfaces come back to Game Panel when their function is over. Play game button opens another panel where the user chooses settings such as difficulty and spaceship icon. Once the user submits the settings, game begins.

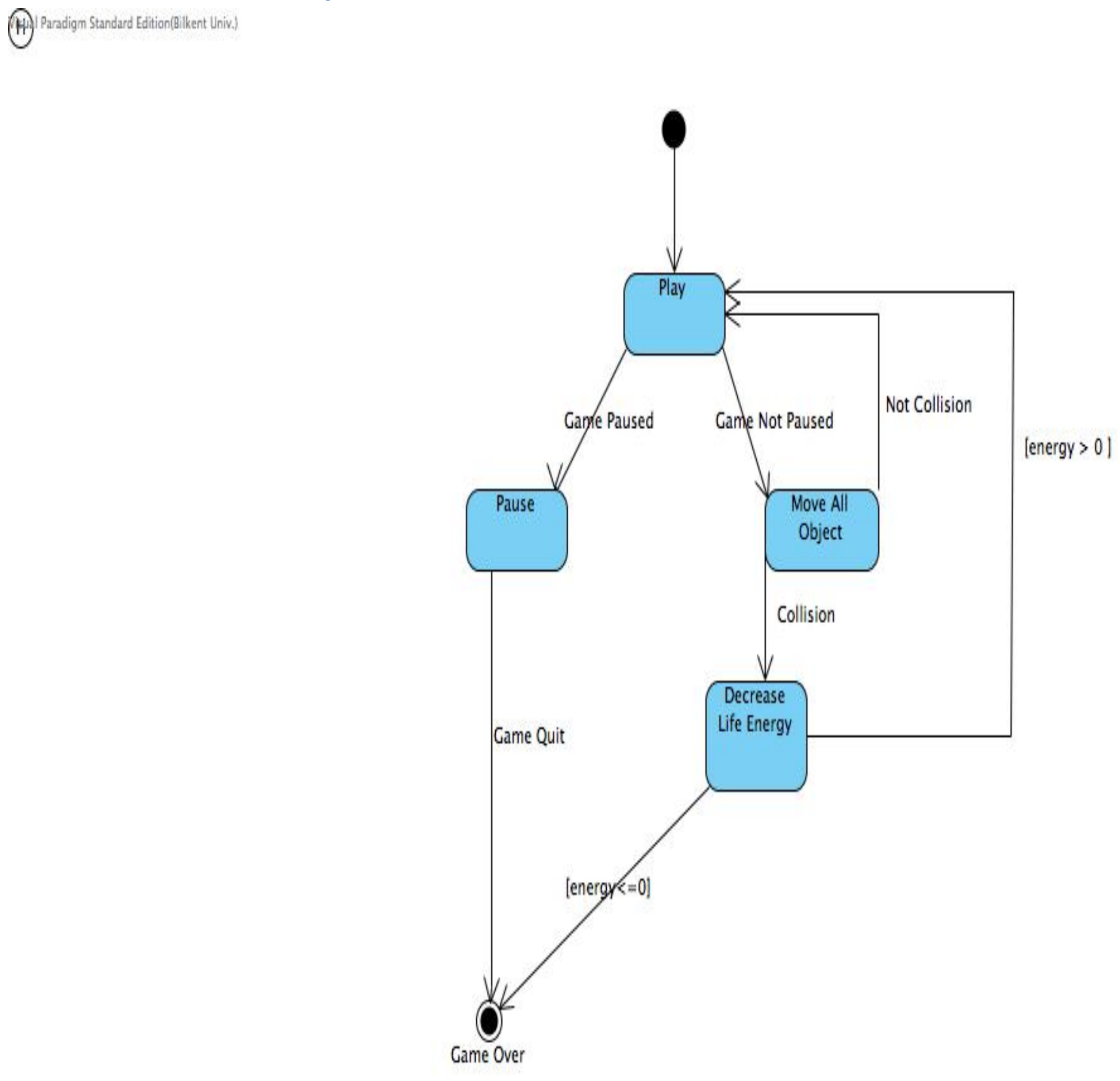
4.2.2.2. Play Game



When the game starts it waits for user input, user can move and fire with pressing keys, after these inputs if user fires, game checks whether the fire is reached to any enemy or not, if it reaches to one of the enemies, it destroys the enemy and generates point for user. Also if user fires or moves, the game checks for time, checks for bomb collision and bonus collision. If the time is up or any bomb is received from the enemy, user loses one of its lives, if its lives are over the game is over. If any bonus is received the bonus will be generated and the game will be continued. After all of these if the player did not die, the Game loop will be active again and wait for user input.

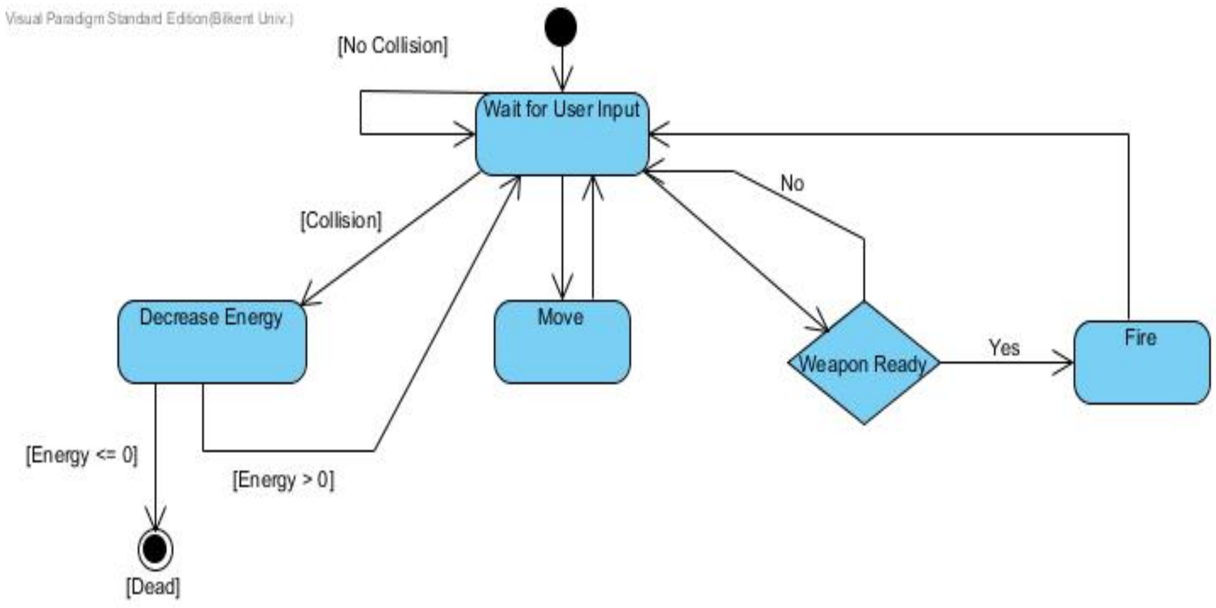
4.2.3. State Diagrams

4.2.3.1 Play Game



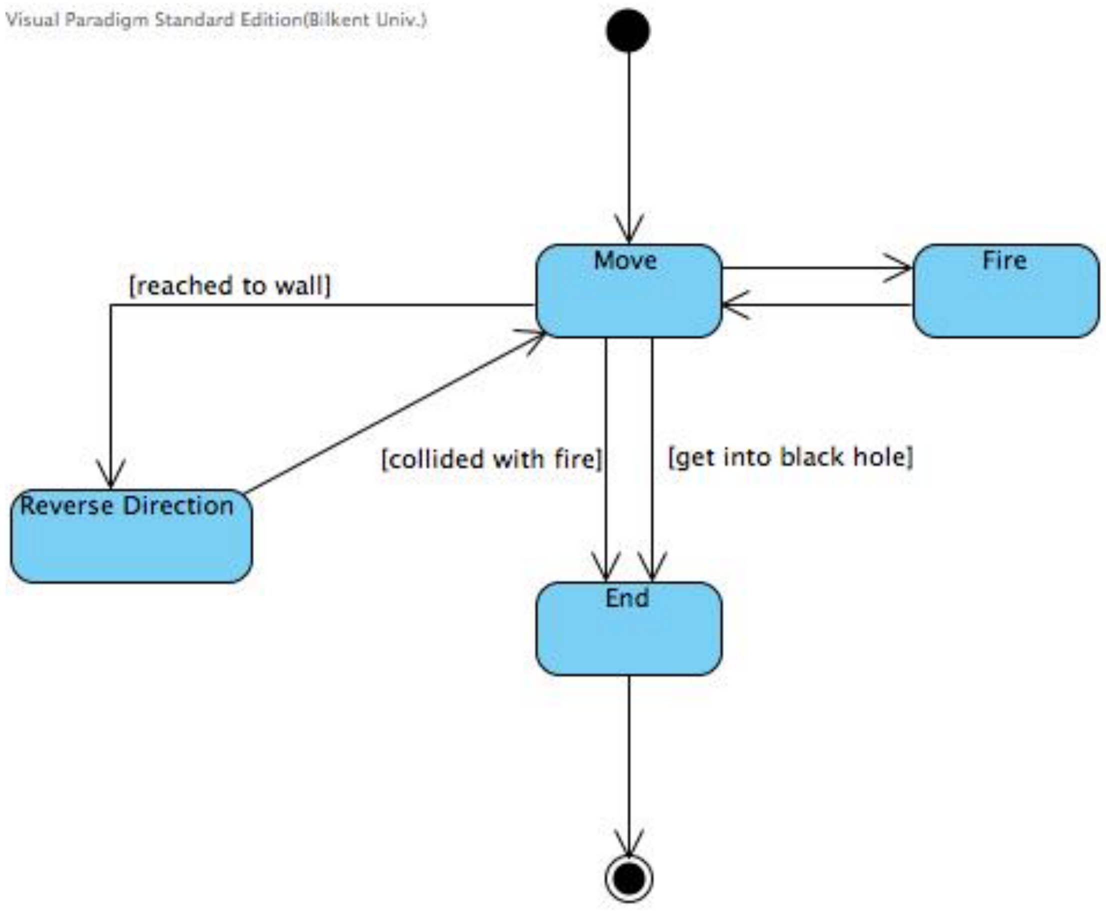
While inside the loop, it is in Play state. Unless the game is paused, all movable objects are moved. Then possible collisions of spaceship with the aliens are checked. If collision is happened , spaceship loses its energy.The game loop continues in this manner as long as the spaceship is still alive (not all of its life energy are lost). Note that a paused game is either resumed or the user quits the current game.

4.2.3.2.Ship



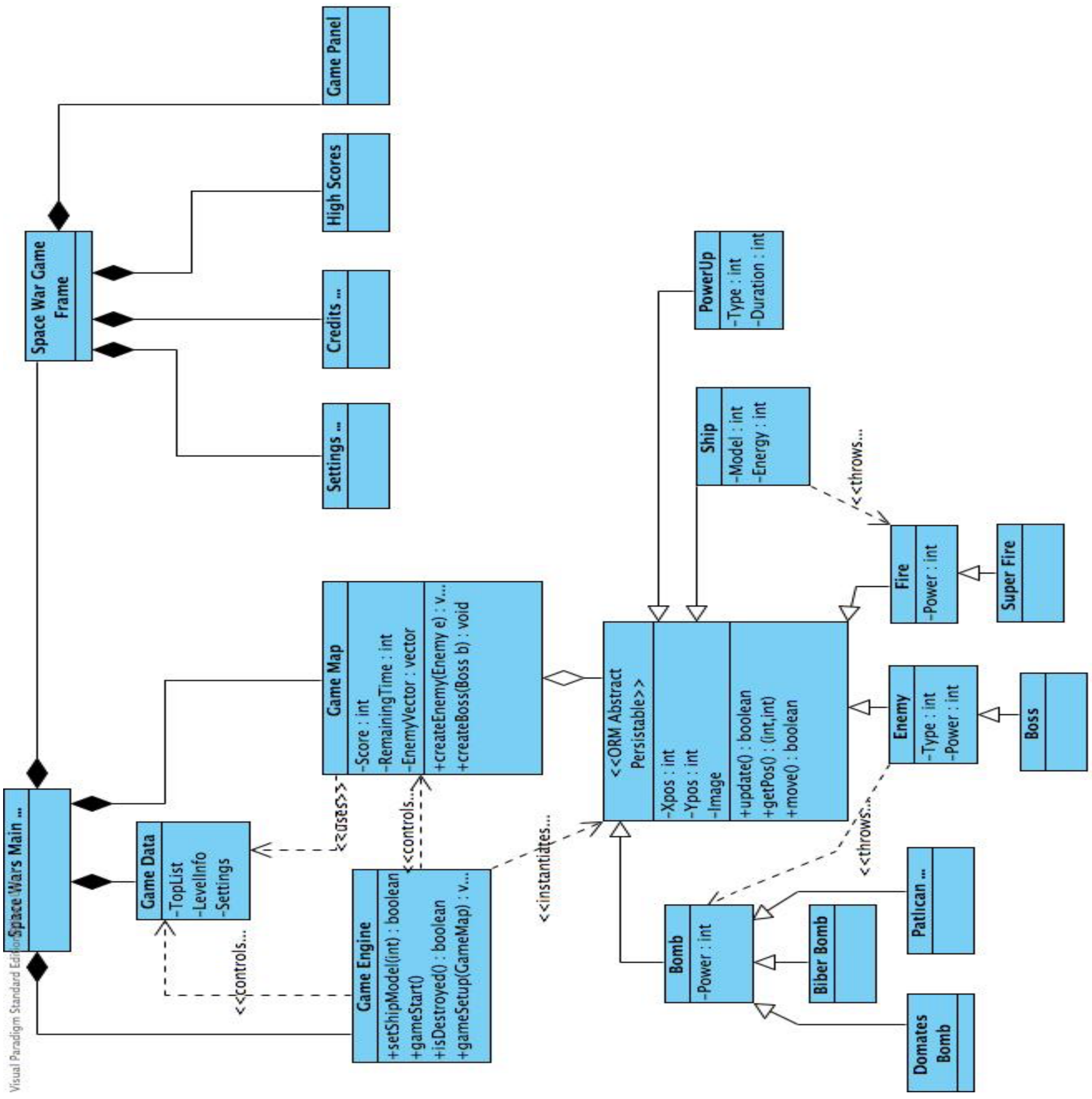
Ship object periodically checks for collisions and decreases energy if it is hit by a projectile. Then it checks its energy and decides whether it is dead. Besides this, ship object can move with user input and fire if its weapon is ready.

4.2.3.3. Enemy



Enemy always moves and periodically fires. If it reaches the wall moves in other direction. If it collides with a fire or get into black hole(which is a power-up of spaceship) it dies and gets to end state.

4.3. Object and Class Model



5.System Design

5.1. Purposes Of The System

Crazy Space War is game that user controls a ship. User's aim is to kill enemies, where the boss and other alliens wander.

Points that user gets is proportional to number of the enemies captured.

Therefore, it's important to decide when or where to start to destroy to be able to get higher points. There are several power-ups in the game that have different features. For examples, by a shield (circle around the ship), ship is protected from allien. However, user should try to kill more enemies as s/he can do because shield will fade after a while. Therefore, user has to manage his/her time and play as fast as s/he can. As a result, it can be stated that our system can help user improve his/her ability to make decision and take action more quickly and also it improves hand-eye coordination of user.

5.2. Design Goals

Extensibility : Game will be suitable for game developments. User can add new feautures to the game. New power-ups, levels can be added to make the game more complex and more enjoyable game experience.

Reliability : Our system will be cautious for unexpected things bugs etc. We will analyze boundary conditions and check them carefully in order to give the user more consistent game experience.

Usability: Game will be user friendly. There will be a screen which gives to user a background knowledge about how to play. Game will be easy to learn. Also, it will have an user friendly interface to help reaching this goal.

Portability: Portability is important for small games. It should work properly on different fields. To reach this goal, our system will be implemented on java which provides platform indepency.

Performance: One of the most important concern is to get high on performance in terms of visual graphics. Since game consists of multiple objects and its movements, it is crucial to have graphics which work on high performance. To reach this goal, we will use efficient memory usage for objects.

Trade offs

Functionality – Usability: we give importance to learnability and usability because game caters towards a wide range of user. That's why we prefer to keep game control as simple as possible, but no simpler. Because controllers matching the real world expectation makes learning easier, we assign the arrow keys to move character, "P" button to pause and "esc" to quit game as in other games. In this way, user will remember how to play game easily and this will conform for maximum usability. Since our priority is usability and ease of learning, functionality of our system won't be complex.

Performance vs Memory: Crazy Space war is a visually rich arcade game including animation, sound and fire effects that can slow down the performance of the game. To be able to make our system fast, we have to sacrifice the memory.

5.3. Definitions, acronyms and abbreviations

Abbreviations:MVC: [2] Model View Controller

JDK: JVM:

5.4. References

[1] Java Development Kit [1] Java Virtual Machine

[1] http://en.wikipedia.org/wiki/Java_(programming_language)[2] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.[3] http://en.wikipedia.org/wiki/JTS_Topology_Suite

5.5. Overview

Objection of our game is to entertain the user. In this manner, to be able to achieve this purpose, we defined our design goals which are extensibility, reliability, usability, portability and high performance. Besides, we also analyzed the trade- offs to be able to make the game simple but, at the same time, functional and effective.

6.Software Architecture

6.1. Overview

In this section we will explain the decomposition of our system for reaching the best solution that includes maximum cohesion and minimum coupling. Since designed a game, the architectural design we use will be

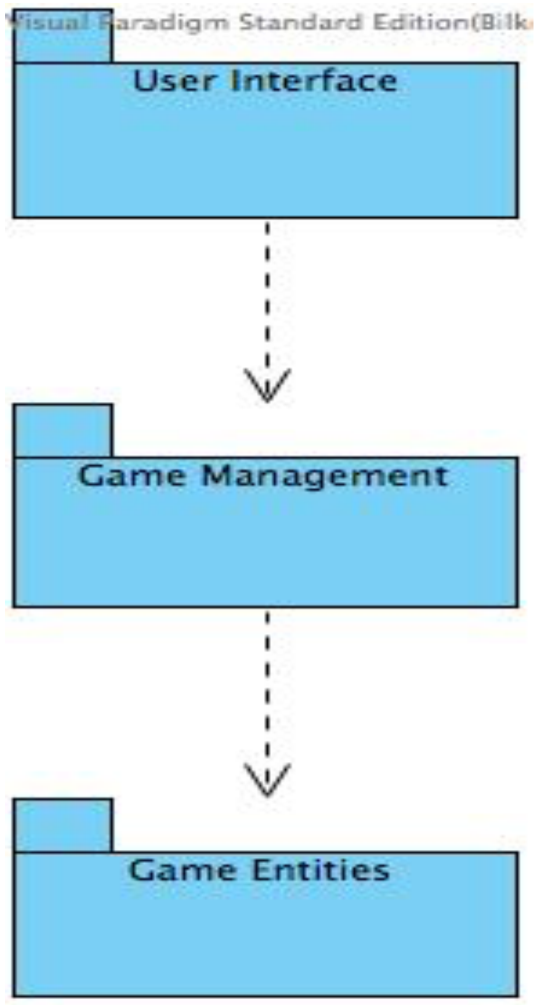
MVC(Model View Controller). For 2D geometry operations like collision we will be using JTS(Java Topology Suite) library.

6.2. Subsystem Decomposition

We are going to divide our system into three subsystems, they are User Interface, Game Management and Game Entities. These subsystems refer to Model, View and Controller in our architecture. User Interface refers to View, Game Management refers to Controller, Game Entities refer to Model. This division provides us better understanding for design of the system.

The classes in the same subsystem will have similar purposes and thus similar structures. For example in the UI subsystem, most of the classes will be inherited from the same ancestor class. It is the same for game entities class. Most of the entity classes will be descendants of GameObject class.

Extendibility and maintainability will be increased thanks to this decomposition because the communication between subsystems is clear and debugging is easier for the programmer. So, the implementation also will be easier for our project group members because if one of the member implements a class for a subsystem, other classes of the same subsystem will be similar to first one and implementation will be easier and faster. Also if a new object will be created for the system, it will be easier to implement too since its mission will be similar to one of the subsystems.



This decomposition will help us to have high coherence and low coupling so we realize our nonfunctional requirements better and reach our design goals.

6.3. Hardware/Software Mapping

The game is going to be implemented in the Java language and additionally Java Swing Library will be used. The game will be a .java file, so to run it the computer should have an operating system, a java compiler and since we hold our database as a txt file, the operating system should support txt files. As hardware configuration, the game will be played using keyboard and mouse. Keyboard will be used during the game for player to fire and move spaceship, mouse will be used to select panel on menu.

6.4 Persistent Data Management

Our game does not need a complex database system. Our game will store persistent data such as level information, settings and high scores. These will be stored using text files in the hard disk, that is, they will be independent from the game. If these text files are corrupted or deleted, the game cannot restore them. In addition to these text files, the game will use images and sounds. These images and sounds will be stored in the hard disk, in gif and wav files formats respectively.

6.5. Access Control and Security

The game will not require authentication and it will not support multiplayer gameplay over a network connection, hence there is no access control and security issues involved. Having the jar file of the game is the only requirement to play it.

6.6 Boundary Conditions

Initialization

The game does not require an installation. Executing the jar file will start the game.

Termination

The game can be closed only using the Exit Game button in the main menu. During the game player would need to pause and quit the game in order to arrive at the main menu. Since the game will work on full screen, the 'X' button will not be available to close the game.

Failure

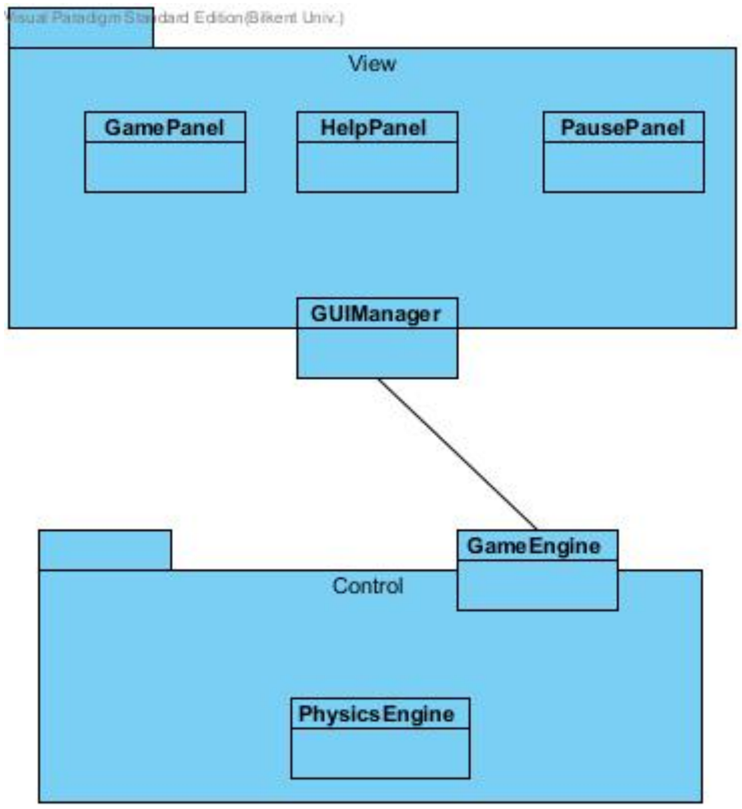
If there are problems about image and sound files in the game folder, the game will run without these sounds and images. If there are problems about the text files for settings and high scores, it will result in data loss.

7. Object-Design

7.1.Design Patterns

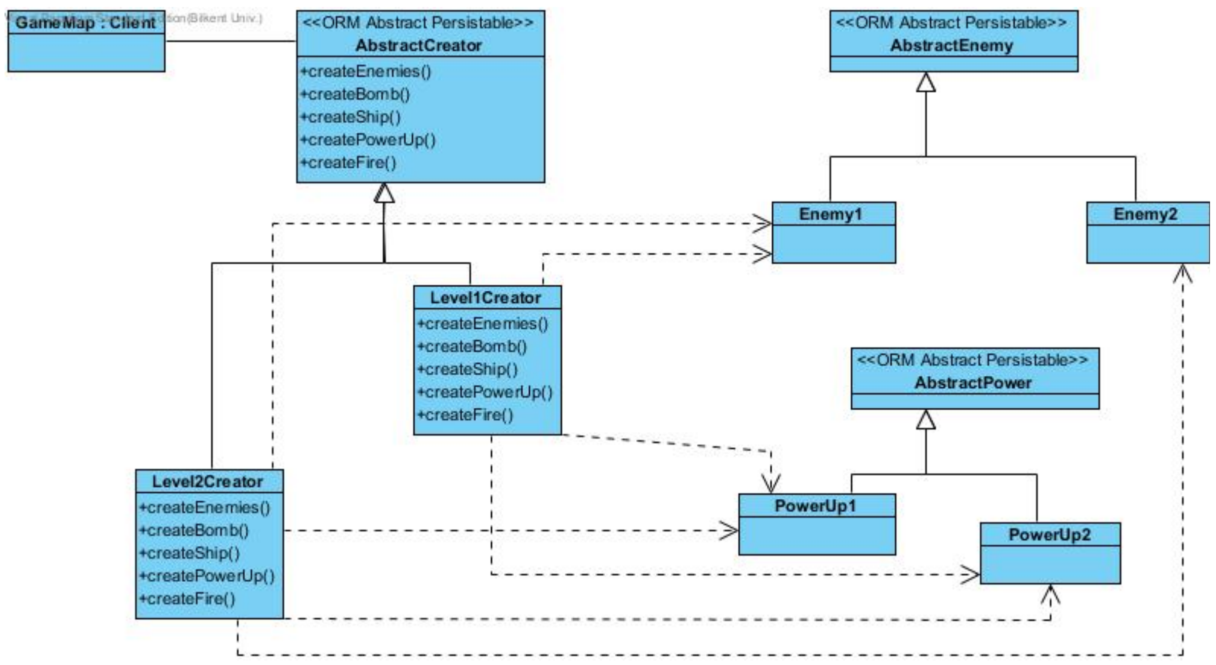
Facade Pattern

Facade Pattern used to introduce a common interface for set of subsystems for interacting with other classes. Our GUIManager class is a facade for Panels in view of our project while interacting with control subsystem. Also GameEngine is a facade for Control subsystem while interacting with view subsystem. Facades makes easier to use the system and hide implementation details of internal classes. For example, GameEngine does not have to know details of each panel but it only interacts with GUIManager.



Abstract Factory Pattern

We used Abstract Factory Pattern for creation of game objects. There is a concrete creator class for each level. Through AbstractCreator (Abstract Factory) class, corrsponding level creator class creates movable objects such as Enemy, PowerUp and Bombs. With Abstract Factory Pattern we are able to change properties and counts of objects for each level independently witout disturbing client's requests.



Observer Pattern

7.2.Architectural Patterns

Model – View- Controller Architectural Style (MVC)

Subsystems are classified into 3 different types. Data package including GameData, GameMap and GameObjects constitutes the Model subsystem. User Interface package including GUIManager and panels, is responsible for displaying application domain objects to the user and it is called View subsystem. Game Management package includes GameEngine, PhysicsEngine etc. constitutes Controller subsystem which is responsible for interactions with the user and model and it notifies views of changes in the model.

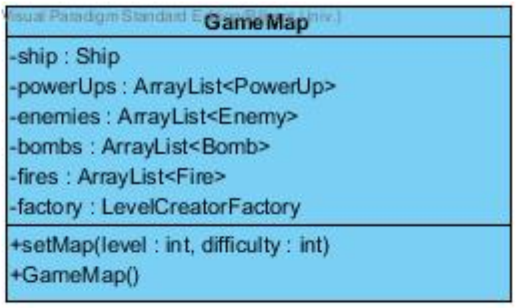
3-Layer Architectural Style

The game consists of 3 hierarchical subsystems. There are a user interface , middleware which is the Game Management package and a database system namely, Data package containing GameMap and

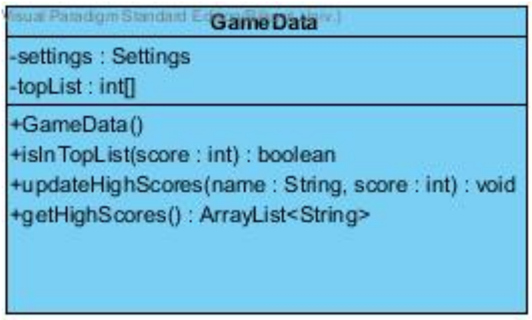
GameData GameObject classes. Game Management subsystem services data requests between the user interface and the Data package.

7.3.Class Interfaces

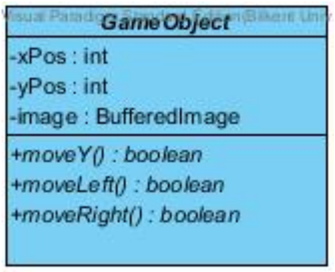
7.3.1. Model


GameMap
-ship : Ship
-powerUps : ArrayList<PowerUp>
-enemies : ArrayList<Enemy>
-bombs : ArrayList<Bomb>
-fires : ArrayList<Fire>
-factory : LevelCreatorFactory
+setMap(level : int, difficulty : int)
+GameMap()

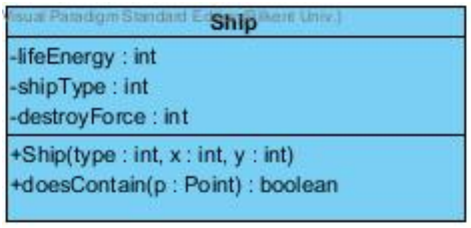
GameMap class stores all ship, enemy, power-up, fire and bomb objects. It calls corresponding LevelCreator class to create game objects. It is updated by PhysicsEngine through the game.


GameData
-settings : Settings
-topList : int[]
+GameData()
+isInTopList(score : int) : boolean
+updateHighScores(name : String, score : int) : void
+getHighScores() : ArrayList<String>

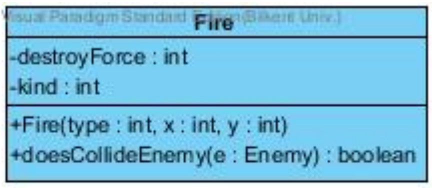
GameData class stores persistent data for the game such as high scores and settings. Class is responsible for providing and updating data as it changes.



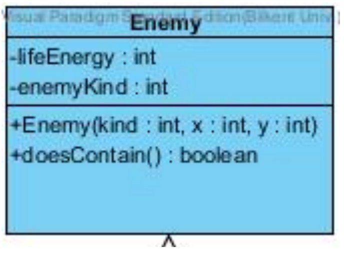
Game Object is an abstract class created by generalization inheritance. It has common attributes Xpos and Ypos of Ship, Fire, Enemy, Power Up and Bomb subclasses.


Visual Paradigm Standard Edition(Bilkent Univ.)
Ship
-lifeEnergy : int
-shipType : int
-destroyForce : int
+Ship(type : int, x : int, y : int)
+doesContain(p : Point) : boolean

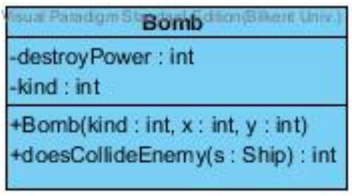
Ship extends Game Object class. It has type and remaining life count as lifeEnergy.


Visual Paradigm Standard Edition(Bilkent Univ.)
Fire
-destroyForce : int
-kind : int
+Fire(type : int, x : int, y : int)
+doesCollideEnemy(e : Enemy) : boolean

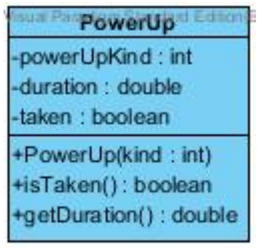
Fire extends Game Object class. It has a kind depending of the level. Also it has a destroy force which determines how many lives it reduces from Enemy's lives in case of collision.


Visual Paradigm Standard Edition(Bilkent Univ.)
Enemy
-lifeEnergy : int
-enemyKind : int
+Enemy(kind : int, x : int, y : int)
+doesContain() : boolean

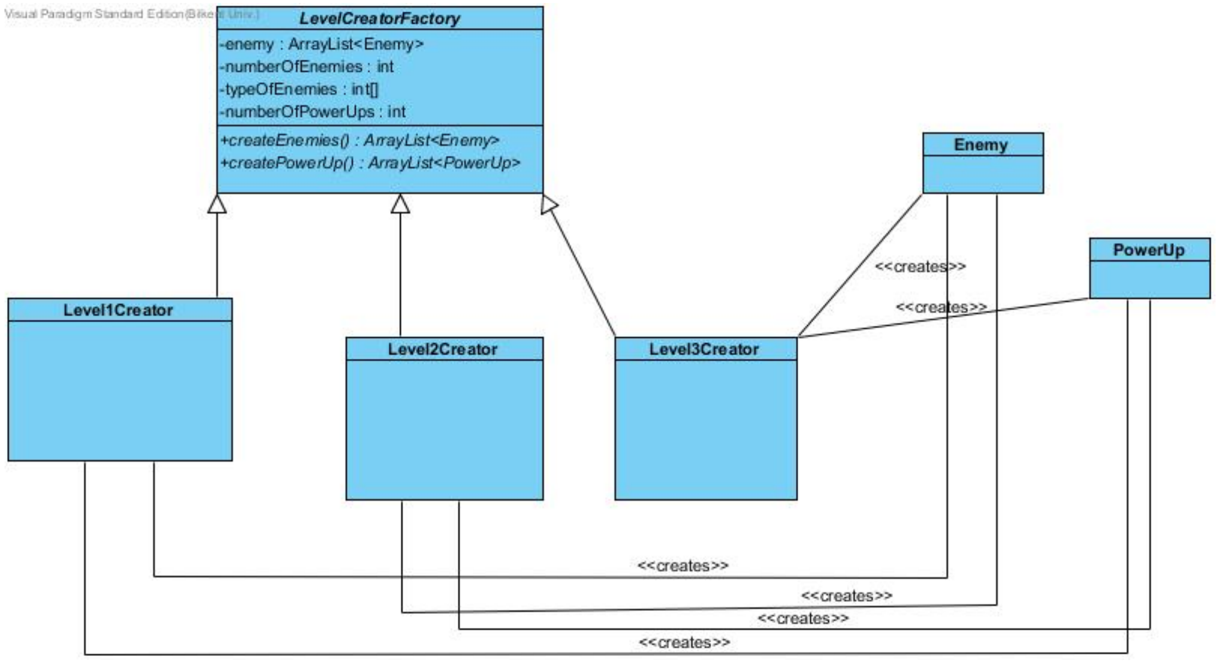
Enemy class is extended from Game object class. It has power attribute in addition to its parent. Shoot method creates bomb object. In case of a collision with a fire object, Enemy's power will be decreased.


Visual Paradigm Standard Edition(Bilkent Univ.)
Bomb
-destroyPower : int
-kind : int
+Bomb(kind : int, x : int, y : int)
+doesCollideEnemy(s : Ship) : int

Bomb class extends Game Object class. Bomb objects created by Enemy objects and they are destroyed when they collide with ship or when they reached the bottom level.

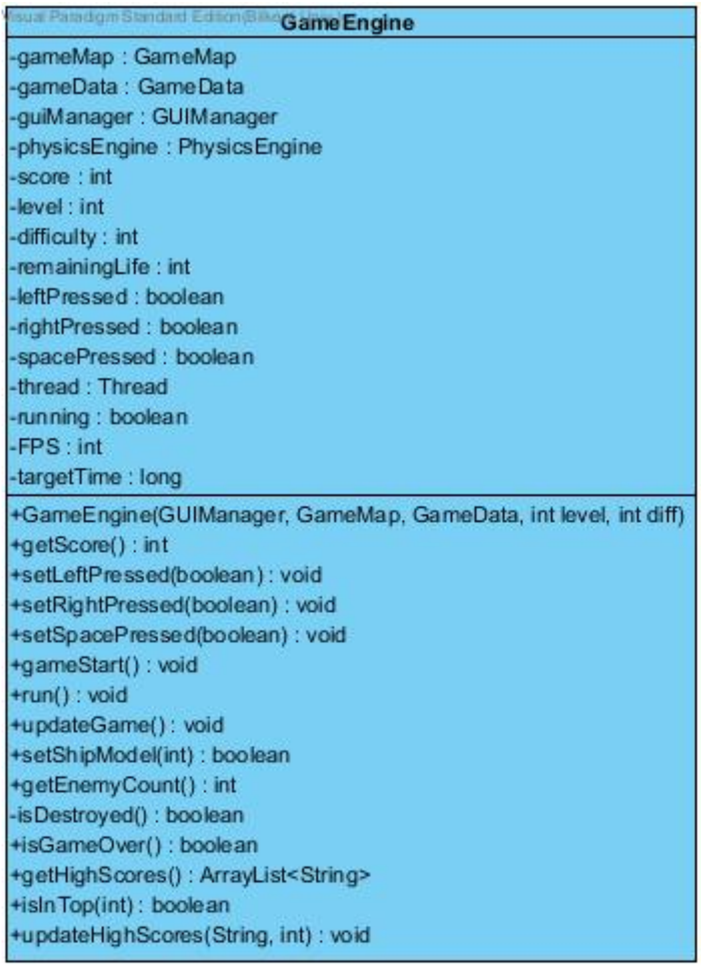


Power up also extends Game Object class. It has three additional attributes; its type duration and a Boolean value as taken. Power Up Object will be destroyed when it collides with the ship. Ships properties will be adjusted accordingly.

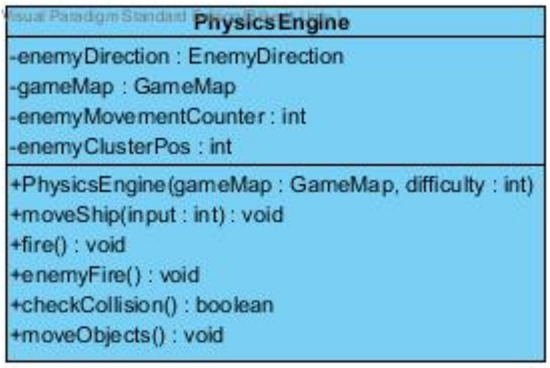


LevelCreatorFactory is an abstract class which is responsible for creation of gameobject depending on the level. Implementation of the class is performed by concrete LevelCreator classes.

7.3.2. Controller



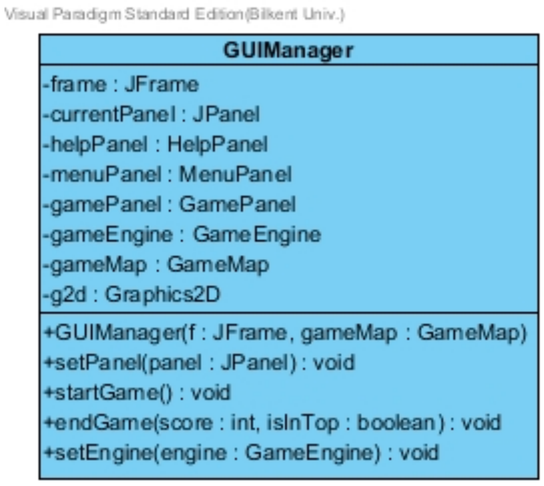
GameEngine Class is a Façade class. View classes use GamEngine to communicate with Control subsystem. We used a fixed time game loop to cover irregularities caused by different collision checks etc. during each update.


PhysicsEngine
-enemyDirection : EnemyDirection
-gameMap : GameMap
-enemyMovementCounter : int
-enemyClusterPos : int
+PhysicsEngine(gameMap : GameMap, difficulty : int)
+moveShip(input : int) : void
+fire() : void
+enemyFire() : void
+checkCollision() : boolean
+moveObjects() : void

PhysicsEngine class makes computations such as moving Enemy objects and making them fire, checking for collisions and manipulates data in the GameMap. Methods of this class is called in GameEngine class during the game loop.

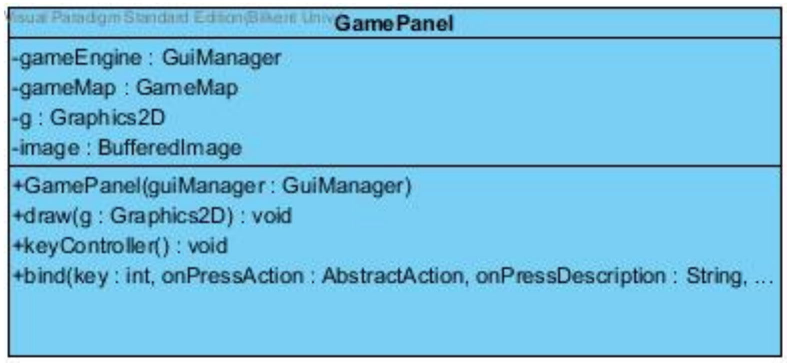
7.3.3. View

GUIManager Class


Visual Paradigm Standard Edition(Bilkent Univ.)
GUIManager
-frame : JFrame
-currentPanel : JPanel
-helpPanel : HelpPanel
-menuPanel : MenuPanel
-gamePanel : GamePanel
-gameEngine : GameEngine
-gameMap : GameMap
-g2d : Graphics2D
+GUIManager(f : JFrame, gameMap : GameMap)
+setPanel(panel : JPanel) : void
+startGame() : void
+endGame(score : int, isInTop : boolean) : void
+setEngine(engine : GameEngine) : void

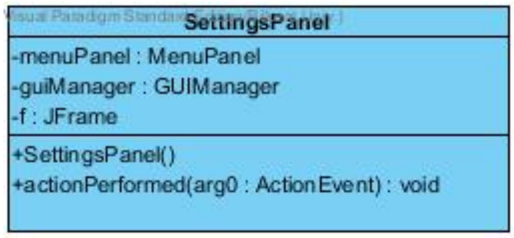
GUIManager is the controller class of the user interface subsystem. It communicates with GameEngine and GameMap classes in order to supply the view classes with sufficient information. Hence, it is a façade class. GUIManager mostly receives commands from GameEngine such as starting and ending the game, and arranges the view classes accordingly.

GamePanel


Visual Paradigm Standard Edition(Bilkent Univ.)
GamePanel
-gameEngine : GuiManager
-gameMap : GameMap
-g : Graphics2D
-image : BufferedImage
+GamePanel(guiManager : GuiManager)
+draw(g : Graphics2D) : void
+keyController() : void
+bind(key : int, onPressAction : AbstractAction, onPressDescription : String, ...

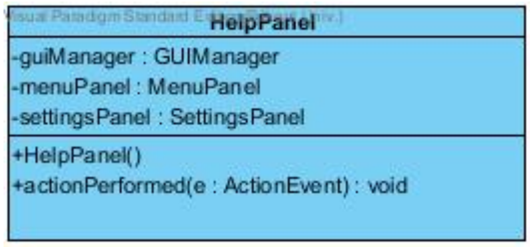
This class is a panel which contains information about the state of the game such as current level number, score, number of lives remaining, and it draws graphical representations of all instances of GameObject class on the screen. The class also implements ActionListener interface to capture key events and notify the GameEngine class when a keys are pressed.

SettingsPanel



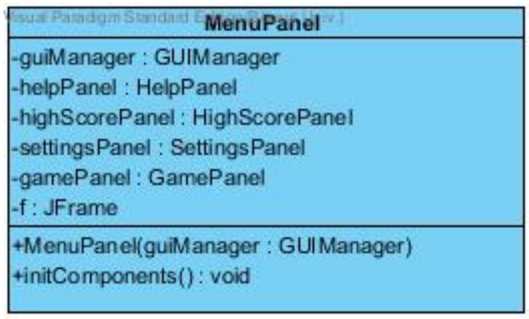
SettingsPanel contains three pieces of information which are difficulty, music and sound. User can select the desired option using radio buttons and check boxes. When new settings are submitted, GUIManager class is notified.

HelpPanel Class


HelpPanel
-guiManager : GUIManager
-menuPanel : MenuPanel
-settingsPanel : SettingsPanel
+HelpPanel()
+actionPerformed(e : ActionEvent) : void

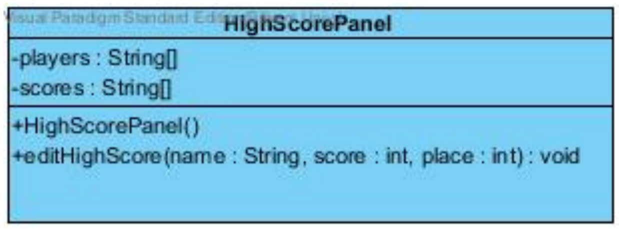
HelpPanel displays information regarding rules, enemy types and power ups. This panel is accessed by the MenuPanel class.

MenuPanel Class



MenuPanel is displayed during opening of the game and during pauses. It includes a reference to all view classes and transfers control between these classes according to the ActionListener. Hence, it serves as a navigation tool inside the game for the user.

HighScorePanel Class


Visual Paradigm Standard Edition
HighScorePanel
-players : String[]
-scores : String[]
+HighScorePanel()
+editHighScore(name : String, score : int, place : int) : void

HighScorePanel is a simple view class which fetches high scores and draws them on the screen using labels.

7.4.Contract with OCL

7.4.1. Enemy Class

Context Enemy :: getLifeEnergy() pre:

Self.getLifeEnergy() > 0

Context Enemy:: getLifeEnergy() post

Self.getLifeEnergy() <= 0

In previous condition indicate that to kill any enemy , it is compulsory that enemies life energy have to be more than zero. On the other hand post condition says that after being killed its life energy must become zero or less than zero.

7.4.2. PowerUp Class

Context PowerUp :: isTaken() pre :

Not self.getTaken()

The isTaken() operation can only be invoked if taken is false which means that ship has no powerUp when it is taken.

Context PowerUp::isTaken() post:

Self.getTaken()

Context PowerUp:: isTaken() post:

self.taken = true

After isTaken() method is called taken becomes true.

7.4.3.PhysicsEngine Class

Context PhysicsEngine :: isEnemyDestroyed() pre :

Self.gameData.getEnemies().size()>0

To destroy any enemies game screen must includes enemies.

Context PhysicsEngine::isEnemyDestroyed() post :

self@pre.gameData.getEnemies().size()-1

After being killed any enemy size must be decreased.

7.4.4. GUIManager Class

Context GUIManager:: editHighScore() pre:

self.isInTop == true

To edit the highscore, isInTop should be true

Context GUIManager::setPanel(panel:JPanel) post:

self.panel = panel

After the setPanel method, parameter must equal to panel.

Context GUIManager::startGame() post:

self.setPanel(gamePanel)

After the startgame button is pressed, current panel will be gamePanel

Context GUIManager::setEngine(gameEngine: GameEngine) post:

self.gameEngine = gameEngine;

After setEngine method, parameter must equal to engine.

7.4.5. Bomb Class

Context Bomb::moveRight() post:

self.xPos = xPos + cell;

After moveRight method is called, bomb's x coordinate will be change to the right.

Context Bomb::moveY() post:

self.yPos = yPos – cell;

After moveY() method is called bomb's y coordinate will change about cell amount.

Context Bomb:setKind(kind : int) post:

self.kind = kind;

After setKind() method is called, parameter must equal to the kind.

7.4.6. GameMap Class

Context GameMap::addPowerUp(pw: PowerUp) post:

self.powerUps.add(pw)

After addPowerUp method is called, selected power up will be added to the powerUps arraylist.

Context GameMap::addFire(fire: Fire) post:

self.fires.add(fire)

After addFire() method is called selected fire will be added to the fires arraylist.

7.4.7. Settings Class

Context Settings::changeSettings(difficulty:int, music:Boolean, sound:boolean) post:

self.difficulty = difficulty

self.music = music

self. Sound = sound

After changeSettings() method is called, settings will be updated according to he parameters.

7.4.8. Ship Class

Context Ship::setShipTypeImg(shipType : int) post:

    self.image = ImageIO.read(new File("ship"+shipType+".png"))

After setShipTypeImg() method is called, ships image will be set according to its type.

7.4.9. GameEngine Class

Context GameEngine::setLeftPressed(b:boolean) pre:

    GamePanel.keyController()

To setLeftPressed be called, firstly keyController should be colled which is a GamePanel method.

Context GameEngine::gameStart() post:

    gameMap.setMap(level,difficulty)

After gameStart method is called, gameMap will be set.

Context GameEngine::run() post:

    self.updateGame()

After the run() method is called, updateGame method is called inside.

8.Conclusion

Crazy Space War is a shooter game in which the player meets waves of enemies and destroys them without getting hit. There are many other games in this genre however, we tried to design an original game by twisting some of the rules and objects in the game.

In the analysis part, we tried to design a game which would be different than the other games in the genre. We came up with new ideas for power ups, enemies, and level types. We identified requirements and use cases for the system so that it would run properly.

In the design part, we focused on separating the problems of the system and solving them individually so that it could be implemented smoothly and provide a pleasant experience for the users.

In conclusion, we tried to apply the principles we learned throughout the course to produce a viable project.

Appendix

```
/**
 *
 * @author omerhanci
 */


package view;
import javax.swing.JFrame;

import java.awt.BorderLayout;

import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

import model.*;
import controller.*;

public class SpaceWar {
    public SpaceWar(){
      JFrame frame = new JFrame("SpaceWar");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setResizable(false);
      GameMap gameMap = new GameMap();
      GUIManager guiManager = new GUIManager(frame,gameMap);
      GameData gameData = new GameData();
      GameEngine gameEngine = new GameEngine(guiManager,
gameMap,gameData, 1,1);
      guiManager.setEngine(gameEngine);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new SpaceWar();//create an instance which
includes GUI etc
            }
        });
    }
}


package view;
import javax.swing.*;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JRadioButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.GroupLayout.Alignment;
```

```
/**
*
* @author omerhanci
*/
public class SettingsPanel extends javax.swing.JPanel{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private MenuPanel menuPanel;
    private GUIManager guiManager;
    private JFrame f;
    public SettingsPanel() {

      JLabel lblSound = new JLabel("Sound :");

      JButton btnBack = new JButton("Back");
      btnBack.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent arg0) {
                new SpaceWar();
           }
      });


      JCheckBox chckbxSound = new JCheckBox("Sound");

      JLabel lblDifficulty = new JLabel("Difficulty :");

      JLabel lblSpaceship = new JLabel("Spaceship :");

      JRadioButton rdbtnEasy = new JRadioButton("Easy");

      JRadioButton rdbtnMedium = new JRadioButton("Medium");

      JRadioButton rdbtnHard = new JRadioButton("Hard");

      ButtonGroup group = new ButtonGroup();
         group.add(rdbtnEasy);
         group.add(rdbtnMedium);
         group.add(rdbtnHard);
         GroupLayout groupLayout = new GroupLayout(this);
         groupLayout.setHorizontalGroup(
            groupLayout.createParallelGroup(Alignment.LEADING)
                 .addGroup(groupLayout.createSequentialGroup()
                      .addGap(55)
                      .addComponent(lblSound,
GroupLayout.PREFERRED_SIZE, 71, GroupLayout.PREFERRED_SIZE)
                      .addGap(183)
                      .addComponent(chckbxSound))
                 .addGroup(groupLayout.createSequentialGroup()
                      .addGap(354)
                      .addComponent(btnBack))
                .addGroup(groupLayout.createSequentialGroup()
                     .addGap(55)

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)
                          .addComponent(lblSpaceship)

    .addGroup(groupLayout.createSequentialGroup()

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

    .addComponent(lblDifficulty, GroupLayout.PREFERRED_SIZE,
71, GroupLayout.PREFERRED_SIZE)

    .addGroup(groupLayout.createSequentialGroup()
                                              .addGap(70)

    .addComponent(rdbtnEasy, GroupLayout.PREFERRED_SIZE, 71,
GroupLayout.PREFERRED_SIZE)))
                               .addGap(9)
                               .addComponent(rdbtnMedium)
                               .addGap(22)
                               .addComponent(rdbtnHard,
GroupLayout.PREFERRED_SIZE, 141,
GroupLayout.PREFERRED_SIZE))))
        );
        groupLayout.setVerticalGroup(
           groupLayout.createParallelGroup(Alignment.LEADING)
                .addGroup(groupLayout.createSequentialGroup()
                     .addGap(42)

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

    .addGroup(groupLayout.createSequentialGroup()
                               .addGap(4)
                               .addComponent(lblSound))
                          .addComponent(chckbxSound))
                     .addGap(23)

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

    .addGroup(groupLayout.createSequentialGroup()
                               .addGap(4)
                               .addComponent(lblDifficulty))
                          .addComponent(rdbtnEasy)
                          .addComponent(rdbtnMedium)
                          .addComponent(rdbtnHard))
                     .addGap(28)
                     .addComponent(lblSpaceship)
                     .addGap(92)
                     .addComponent(btnBack))
        );
        setLayout(groupLayout);
    }
}


/**
 *
 * @author omerhanci
 */
package view;
import controller.*;

import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import javax.swing.JFrame;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;

import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.LayoutStyle.ComponentPlacement;

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class MenuPanel extends javax.swing.JPanel {
    private BufferedImage backgroundImage ;
    GUIManager guiManager;
    private HelpPanel helpPanel;
    private HighScorePanel highScorePanel;
    private SettingsPanel settingsPanel;
    private GamePanel gamePanel;
    private JFrame f;

    public MenuPanel(GUIManager guiManager) {
      try{
            backgroundImage = ImageIO.read(new 
File("/Users/muhammedemincs/git/crazy-space-
war/src/crazy.jpeg"));
      }catch(IOException ex){
            System.out.println("error");
      }

      this.guiManager = guiManager;
      initComponents();

    }
    private void initComponents() {

      JButton btnStartGame = new JButton("Start Game");
      btnStartGame.addActionListener(new ActionListener()
      {
           public void actionPerformed(ActionEvent arg0) {
                guiManager.startGame();
           }
      });

           JButton btnViewHighScores = new JButton("View High
Scores");
           btnViewHighScores.addActionListener(new
ActionListener() {
                public void actionPerformed(ActionEvent arg0)
{
                     highScorePanel = new HighScorePanel();
                     guiManager.setPanel(highScorePanel);
                }
           });

                JButton btnSettings = new JButton("Settings");
                btnSettings.addActionListener(new
ActionListener() {
                     public void actionPerformed(ActionEvent
arg0) {
                          settingsPanel = new
SettingsPanel();
                          guiManager.setPanel(settingsPanel);
                     }
                });

           JButton btnHelp = new JButton("Help");
           btnHelp.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0)
{
                     helpPanel = new HelpPanel();
                     guiManager.setPanel(helpPanel);
                }
           });
           GroupLayout groupLayout = new GroupLayout(this);
           groupLayout.setHorizontalGroup(

    groupLayout.createParallelGroup(Alignment.LEADING)

    .addGroup(groupLayout.createSequentialGroup()

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

    .addGroup(groupLayout.createSequentialGroup()
                                   .addGap(178)
                                   .addComponent(btnHelp))
    .addGroup(groupLayout.createSequentialGroup()
                                  .addGap(169)

    .addComponent(btnSettings))

    .addGroup(groupLayout.createSequentialGroup()
                                  .addGap(141)

    .addComponent(btnViewHighScores))

    .addGroup(groupLayout.createSequentialGroup()
                                  .addGap(161)

    .addComponent(btnStartGame)))
                       .addContainerGap(157,
Short.MAX_VALUE))
           );
           groupLayout.setVerticalGroup(

    groupLayout.createParallelGroup(Alignment.LEADING)

    .addGroup(groupLayout.createSequentialGroup()
                       .addGap(42)
                       .addComponent(btnStartGame)
                       .addGap(18)
                       .addComponent(btnViewHighScores,
GroupLayout.PREFERRED_SIZE, 28, GroupLayout.PREFERRED_SIZE)
                       .addGap(18)
                       .addComponent(btnSettings)
                       .addGap(18)
                       .addComponent(btnHelp)
                       .addGap(89))
           );
           setLayout(groupLayout);
    }

}


/**
 *
 * @author omerhanci
 */
package view;
import javax.swing.JButton;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.awt.Graphics;
import javax.swing.ImageIcon;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.JLabel;
import javax.swing.LayoutStyle.ComponentPlacement;

public class HighScorePanel extends javax.swing.JPanel {

    public String player1,player2,player3,player4,player5;
    public String score1,score2,score3,score4,score5;
    public void editHighScore(String name, int score, int
place){
      if(place == 1){
            player1 = name;
            score1 = String.valueOf(score);
      }
      else if(place == 2){
            player2 = name;
            score2 = String.valueOf(score);
      }
      else if(place == 3){
            player3 = name;
            score3 = String.valueOf(score);
      }
      else if(place == 4){
            player4 = name;
            score4 = String.valueOf(score);
      }
      else{
            player5 = name;
            score5 = String.valueOf(score);
      }

    }

    public HighScorePanel(){
      JButton btnBack = new JButton("Back");
      btnBack.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                  new SpaceWar();
            }
      });


      JLabel lblHighScores = new JLabel("High Scores");

      JLabel label = new JLabel("1");

      JLabel label_1 = new JLabel("2");

      JLabel label_2 = new JLabel("3");

      JLabel label_3 = new JLabel("4");

      JLabel label_4 = new JLabel("5");
```

```
        JLabel lblScore = new JLabel(score1);

        JLabel lblScore_1 = new JLabel(score2);

        JLabel lblScore_2 = new JLabel(score3);

        JLabel lblScore_3 = new JLabel(score4);

        JLabel lblScore_4 = new JLabel(score5);

        JLabel lblPlayer = new JLabel(player1);

        JLabel lblPlayer_1 = new JLabel(player2);

        JLabel lblPlayer_2 = new JLabel(player3);

        JLabel lblPlayer_3 = new JLabel(player4);

        JLabel lblPlayer_4 = new JLabel(player5);

        GroupLayout groupLayout = new GroupLayout(this);
        groupLayout.setHorizontalGroup(
              groupLayout.createParallelGroup(Alignment.LEADING)
                    .addGroup(groupLayout.createSequentialGroup()

      .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

      .addGroup(groupLayout.createSequentialGroup()
                                    .addGap(363)
                                    .addComponent(btnBack))

      .addGroup(groupLayout.createSequentialGroup()
                                    .addGap(170)
                                    .addComponent(lblHighScores)))
                          .addContainerGap(12, Short.MAX_VALUE))
                    .addGroup(groupLayout.createSequentialGroup()
                          .addGap(58)

      .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)
                                .addComponent(label)
                                .addComponent(label_1)
                                .addComponent(label_2)
                                .addComponent(label_3)
                                .addComponent(label_4))
                          .addGap(66)

      .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)
                                .addComponent(lblPlayer)
                                .addComponent(lblPlayer_1)
                                .addComponent(lblPlayer_2)
                                .addComponent(lblPlayer_3)
                                .addComponent(lblPlayer_4))

    .addPreferredGap(ComponentPlacement.RELATED, 169,
Short.MAX_VALUE)

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)
                                .addComponent(lblScore)
                                .addComponent(lblScore_1)
                                .addComponent(lblScore_2)
                                .addComponent(lblScore_3)
                                .addComponent(lblScore_4))
                          .addContainerGap(38, Short.MAX_VALUE))
      );
      groupLayout.setVerticalGroup(
            groupLayout.createParallelGroup(Alignment.LEADING)
                  .addGroup(groupLayout.createSequentialGroup()
                        .addGap(16)
                        .addComponent(lblHighScores)
                        .addGap(32)

    .addGroup(groupLayout.createParallelGroup(Alignment.BASELI
NE)
                                .addComponent(label)
                                .addComponent(lblScore)
                                .addComponent(lblPlayer))
                        .addGap(18)

    .addGroup(groupLayout.createParallelGroup(Alignment.BASELI
NE)
                                .addComponent(label_1)
                                .addComponent(lblScore_1)
                                .addComponent(lblPlayer_1))
                        .addGap(18)

    .addGroup(groupLayout.createParallelGroup(Alignment.BASELI
NE)
                                .addComponent(label_2)
                                .addComponent(lblScore_2)
                                .addComponent(lblPlayer_2))
                        .addGap(18)

    .addGroup(groupLayout.createParallelGroup(Alignment.BASELI
NE)
                                .addComponent(label_3)
                                .addComponent(lblScore_3)
                                .addComponent(lblPlayer_3))
                        .addGap(18)

    .addGroup(groupLayout.createParallelGroup(Alignment.BASELI
NE)
                                .addComponent(label_4)
                                .addComponent(lblScore_4)
                        .addComponent(lblPlayer_4))
                    .addGap(46)
                    .addComponent(btnBack))
        );
        setLayout(groupLayout);
    }

}


/**
 *
 * @author omerhanci
 */
package view;
import javax.swing.JPanel;
import java.awt.FlowLayout;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.SwingUtilities;
import javax.swing.JTextField;
import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.JLabel;

public class HelpPanel extends javax.swing.JPanel{
    private GUIManager guiManager;
    private MenuPanel menuPanel;
    private SettingsPanel settingsPanel;
    public HelpPanel() {

      JButton btnBack = new JButton("back");
      btnBack.setBounds(118, 50, 75, 29);

      JLabel lblCrazySpaceWar = new JLabel("Crazy Space War");
      GroupLayout groupLayout = new GroupLayout(this);
      groupLayout.setHorizontalGroup(
            groupLayout.createParallelGroup(Alignment.LEADING)
                  .addGroup(groupLayout.createSequentialGroup()

    .addGroup(groupLayout.createParallelGroup(Alignment.LEADIN
G)

    .addGroup(groupLayout.createSequentialGroup()
                                .addGap(363)
                                .addComponent(btnBack))

    .addGroup(groupLayout.createSequentialGroup()
                                .addGap(154)

    .addComponent(lblCrazySpaceWar)))
                      .addContainerGap(12, Short.MAX_VALUE))
      );
      groupLayout.setVerticalGroup(
            groupLayout.createParallelGroup(Alignment.LEADING)
                  .addGroup(groupLayout.createSequentialGroup()
                        .addGap(18)
                        .addComponent(lblCrazySpaceWar)
                        .addGap(214)
                        .addComponent(btnBack))
      );
      setLayout(groupLayout);
      btnBack.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                  new SpaceWar();
            }
      });
    }
}


/**
 *
 * @author omerhanci
 */
package view;
import javax.swing.JFrame;

import controller.*;
import model.*;

import java.awt.Graphics;
import java.awt.Graphics2D;

import javax.swing.JPanel;
public class GUIManager {
    private final static int WIDTH = 1000;
    private final static int HEIGHT = 1000;
    private JFrame frame;
    private JPanel currentPanel;
    private HelpPanel helpPanel;
    private MenuPanel menuPanel;
    private GamePanel gamePanel;
    private GameEngine gameEngine;
    private GameMap gameMap;
    private Graphics2D g2d;

    public GUIManager(JFrame f, GameMap gameMap){
      frame = f;
      this.gameMap = gameMap;
      menuPanel = new MenuPanel(this);
      gamePanel = new GamePanel(gameMap);
      currentPanel = menuPanel;
      setPanel(menuPanel);
    }
    public void setPanel(JPanel panel){
      frame.setVisible(false);
      frame.getContentPane().removeAll();
      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
    }

    /*public void drawToScreen(){
      gamePanel.drawToScreen();
    }*/

    public void startGame(){
      setPanel(gamePanel);
      gamePanel.init();
      gameEngine.gameStart();
    }

    public void endGame(int score, boolean isInTop)
    {
      //display score
      if(isInTop)
      {
            //ask for name do sth
            String name;     // get name
            //gameEngine.updateHighScores(name, score);
            //highScorePanel.editHighScore(name,score,place);
      }
    }
    public void setEngine(GameEngine gameEngine) {
      this.gameEngine = gameEngine;
      this.gamePanel.setEngine(gameEngine);
    }
}


/**
 *
 * @author omerhanci

 */
package view;
import model.*;
import controller.*;

import java.awt.Dimension;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import javax.swing.*;

import java.awt.Graphics2D;

import javax.swing.JPanel;


public class GamePanel extends javax.swing.JPanel{
    private GameMap gameMap;
    private GameEngine gameEngine;
    private HighScorePanel highScorePanel;
    private JPanel panel;

    public static final int WHEN =
JComponent.WHEN_IN_FOCUSED_WINDOW;

    //Panel Dimensions
    public static final int WIDTH = 520;
    public static final int HEIGHT = 240;
    public static final int SCALE = 2;


    //Drawing
    private Graphics2D g;
    private BufferedImage image;

    public GamePanel(GameMap gameMap)
    {
      super(true);
      setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT *
SCALE));
      setFocusable(true);
      requestFocus();
      this.gameMap = gameMap;
      keyListeners();
    }

    public void init()
    {
      image = new BufferedImage(WIDTH, HEIGHT,
BufferedImage.TYPE_INT_RGB);
      g = (Graphics2D) image.getGraphics();
    }

    public void paintComponent(Graphics g)
    {
      super.paintComponent(g);
      Graphics2D g2d = (Graphics2D) g;
      draw(g2d);
    }

    public void draw(Graphics2D g2d)
    {
      g2d.setColor(Color.BLACK);
         g2d.fillRect(0, 0, 1040, 600);
         g2d.drawImage(gameMap.getShip().getCurrentImage(),
gameMap.getShip().getXpos(), gameMap.getShip().getYpos(),
null);

         for(int i = 0; i < gameMap.enemies.size(); i++){

     g2d.drawImage(gameMap.enemies.get(i).getCurrentImage(),
gameMap.enemies.get(i).getXpos(),
gameMap.enemies.get(i).getYpos(), null);

         }
         for(int i = 0; i < gameMap.fires.size(); i++){

     g2d.drawImage(gameMap.fires.get(i).getCurrentImage(),
gameMap.fires.get(i).getXpos(),
gameMap.fires.get(i).getYpos(), null);
         }
         repaint();
     }

     public void drawToScreen()
     {
      Graphics g2 = getGraphics();
      g2.drawImage(image, 0, 0, WIDTH * SCALE, WIDTH * SCALE,
null);
      g2.dispose();
      repaint();
     }

     public void keyListeners(){

      this.addKeyListener(new KeyAdapter() {

             @Override
             public void keyReleased(KeyEvent e) {
                 if(e.getKeyCode() == KeyEvent.VK_RIGHT){
                      gameEngine.setRightPressed(false);
                 }
                 else if(e.getKeyCode() == KeyEvent.VK_LEFT){
                      gameEngine.setLeftPressed(false);
                 }
                 else if(e.getKeyCode() == KeyEvent.VK_SPACE){
                      gameEngine.setSpacePressed(false);
                 }
             }

             @Override
             public void keyPressed(KeyEvent e){
                 if(e.getKeyCode() == KeyEvent.VK_RIGHT){
                      gameEngine.setRightPressed(true);
                 }
                 else if(e.getKeyCode() == KeyEvent.VK_LEFT){
                      gameEngine.setLeftPressed(true);
                 }
                 else if(e.getKeyCode() == KeyEvent.VK_SPACE){
                      gameEngine.setSpacePressed(true);
                 }
             }
         });
     }

     public void setEngine(GameEngine gameEngine) {
      this.gameEngine = gameEngine;
     }
}


package controller;
//Author: ErdinÁ
//Computations regarding GameMap
import java.awt.Point;
import java.awt.Rectangle;

import model.*;

public class PhysicsEngine {

     //Attributes
     private enum EnemyDirection {left, right, up, down};
     private EnemyDirection enemyDirection;

     private int LEFT_MOST_PIXEL = 30;
     private int RIGHT_MOST_PIXEL = 30;
     private int BOTTOM_MOST_PIXEL = 30;

     private int enemyClusterX;
     private int ENEMY_MOVEMENT_COUNT;
     private GameMap gameMap;
     private int enemyMovementCounter;

     //Constructor
     public PhysicsEngine( GameMap gameMap, int difficulty) {
      enemyDirection = EnemyDirection.right;
      enemyMovementCounter = 0;
      ENEMY_MOVEMENT_COUNT = 0;
      this.gameMap = gameMap;
      ENEMY_MOVEMENT_COUNT = 15 / difficulty; //fix
     }

     //Methods

     //checks whether ship has collided enemy
     public boolean checkCollision() {
      for( int i = 0; i < gameMap.bombs.size(); i++) {
         //check coordinate
//         if ( gameMap.getShip().getXpos() ==
gameMap.bombs.get(i).getXpos() &&
//              gameMap.getShip().getYpos() ==
gameMap.bombs.get(i).getYpos() )
           if ( gameMap.getShip().doesContain( new Point(
gameMap.bombs.get(i).getXpos(),gameMap.bombs.get(i).getYpos()
) ) )
           return true;
      }
      for( int i = 0; i < gameMap.enemies.size(); i++) {
           //check coordinate
//         if ( gameMap.getShip().getXpos() ==
gameMap.bombs.get(i).getXpos() &&
//              gameMap.getShip().getYpos() ==
gameMap.bombs.get(i).getYpos() )
           if ( gameMap.getShip().doesContain( new Point(
gameMap.enemies.get(i).getXpos(),gameMap.enemies.get(i).getYpo
s() ) ) )
           return true;
      }
      return false;
    }

    //check whether ship collide with powerUps
    public boolean checkPowerUp() {
      for( int i = 0; i < gameMap.powerUps.size(); i++) {
           //check coordinate
//         if ( gameMap.getShip().getXpos() ==
gameMap.powerUps.get(i).getXpos() &&
//                   gameMap.getShip().getYpos() ==
gameMap.powerUps.get(i).getYpos() )
//              gameMap.getShip().getPowerUp(
gameMap.powerUps.get(i));
           if ( gameMap.getShip().doesContain( new
Point(gameMap.powerUps.get(i).getXpos(),
gameMap.powerUps.get(i).getYpos() ) ) )
                if ( gameMap.powerUps.get(i).getPowerUpkind()
== 0) //add health
                     gameMap.getShip().setLifeEnergy(
gameMap.getShip().getLifeEnergy() + 1);
                else if (
gameMap.powerUps.get(i).getPowerUpkind() == 1) ; //speed up
//                   gameMap.getShip().setSpeed(
gameMap.getShip().getSpeed() * 2);
                else if (
gameMap.powerUps.get(i).getPowerUpkind() == 2) ; //destructive
bullets
//                   gameMap.getShip().setDestroyForce(
gameMap.getShip().getDestroyForce() * 2);
           return true;
      }
      return false;
    }

    //check whether enemies collide with fires
    public int isEnemyDestroyed() {
      int enemyDestroyedNumber = 0;
      for( int i = 0; i < gameMap.fires.size(); i++) {
           for( int j = 0; j < gameMap.enemies.size(); j++) {
                //check coordinate, destroy objects if
necessary
                if ( gameMap.fires.get(i).getXpos() ==
gameMap.enemies.get(j).getXpos() &&
                          gameMap.fires.get(i).getYpos() ==
gameMap.enemies.get(j).getYpos() ) {
//                     gameMap.enemies.get(i).isHit(
gameMap.fires[i]);

    gameMap.enemies.get(j).setLifeEnergy(gameMap.enemies.get(j
).getLifeEnergy() - gameMap.fires.get(i).getDestroyForce());
                     gameMap.fires.remove(i);
                     if (
gameMap.enemies.get(j).getLifeEnergy() <= 0) {
//
    gameMap.enemies.get(j).dropPowerUp();
                          gameMap.enemies.remove(j);
                          enemyDestroyedNumber++;
                     }
                }
           }
      }
      return enemyDestroyedNumber;
    }

    //move ship according to input from GameEngine
    public void moveShip( int input) {
      System.out.println("move haci "+input) ;
      if ( input == 0) //left
           gameMap.getShip().setXpos(
gameMap.getShip().getXpos() - 1);
      if ( input == 1) //right
           gameMap.getShip().setXpos(
gameMap.getShip().getXpos() + 1);
//    if ( upPressed)
//         gameMap.getShip().setY( gameMap.getShip().getY( ) +
1);
//    if ( downPressed)
//         gameMap.getShip().setY( gameMap.getShip().getY( ) -
1);
    }

    //wrapper function for move objects, is called by game
engine
    public void moveEnemies() {
      moveObjects();
      //checkPowerUp();
      //enemyFire();
    }
    //move enemies, fires, bombs
    public void moveObjects() {
      //enemies are not ready to move yet
      if ( enemyMovementCounter != ENEMY_MOVEMENT_COUNT) {
            enemyMovementCounter++;
      }
      //move enemies
      else {
            //cannot go left, move down and turn right
            if ( enemyClusterX == LEFT_MOST_PIXEL &&
enemyDirection == EnemyDirection.left) {
                  for ( int i = 0; i < gameMap.enemies.size();
i++)
//                      gameMap.enemies.get(i).setYpos(
gameMap.enemies.get(i).getYpos() -1);
                        gameMap.enemies.get(i).moveY();
                  enemyDirection = EnemyDirection.right;
            }
            //cannot go right, move down and turn left
            else if ( enemyClusterX == RIGHT_MOST_PIXEL &&
enemyDirection == EnemyDirection.right) {
                  for ( int i = 0; i < gameMap.enemies.size();
i++)
//                      gameMap.enemies.get(i).setYpos(
gameMap.enemies.get(i).getYpos() - 1);
                        gameMap.enemies.get(i).moveY();
                  enemyDirection = EnemyDirection.left;
            }
            else if ( enemyDirection == EnemyDirection.right) {
                  for ( int i = 0; i < gameMap.enemies.size();
i++)
//                      gameMap.enemies.get(i).setXpos(
gameMap.enemies.get(i).getXpos() + 1);
                        gameMap.enemies.get(i).moveLeft();
            }
            else if ( enemyDirection == EnemyDirection.left) {
                  for ( int i = 0; i < gameMap.enemies.size();
i++)
//                      gameMap.enemies.get(i).setXpos(
gameMap.enemies.get(i).getXpos() - 1);
                        gameMap.enemies.get(i).moveLeft();
            }
      }
      //move fire and bomb objects
      for ( int i = 0; i < gameMap.bombs.size(); i++)
//          gameMap.bombs.get(i).setYpos(
gameMap.bombs.get(i).getYpos() - 1);
            gameMap.bombs.get(i).moveY();
      for ( int i = 0; i < gameMap.fires.size(); i++)
//          gameMap.fires.get(i).setYpos(
gameMap.fires.get(i).getYpos() + 1);
            gameMap.fires.get(i).moveY();
    }
    //fire on input
    public void fire() {
      //create and set properties
      Fire fireObject = new Fire(0,
gameMap.getShip().getXpos(), gameMap.getShip().getYpos(), new
Rectangle());
      //fireObject.setPower( gameMap.getShip().getPower());
      //add to fire list
      gameMap.fires.add( fireObject);
    }

    //enemies are chosen randomly to bomb
    public void enemyFire() {
      int bombNo = (int) (Math.random() * 5);
      int[] enemiesWhoShoot = new int[bombNo];
      //generate who shoot
      for ( int i = 0; i < bombNo; i++) {
            int newBomb = (int) (Math.random() *
gameMap.enemies.size() );
            //check that same enemies can't shoot twice
            boolean same=false;
            for ( int j =0 ; j < i; j++) {
                  if ( enemiesWhoShoot[j] == newBomb)
                        same = true;
            }
            //if same enemy shoots twice, reiterate
            if ( same)
                  i--;
            else
                  enemiesWhoShoot[i] = bombNo;
      }
      //when all is decided, prompt to shoot
      for(int i = 0; i < enemiesWhoShoot.length; i++) {
//          gameMap.enemies.get(i).shoot();
            Bomb newBomb = new Bomb( gameMap.enemies.get(
enemiesWhoShoot[i]).getEnemyKind(),

    gameMap.enemies.get( enemiesWhoShoot[i]).getXpos(),

    gameMap.enemies.get( enemiesWhoShoot[i]).getYpos(),
                                                    new
Rectangle() );
            gameMap.bombs.add(newBomb);
      }
    }

    //checks whether enemies reached end on Y-axis
    public boolean hasEnemyArrived() {
      for ( int i = 0; i < gameMap.enemies.size(); i++) {
            if ( gameMap.enemies.get(i).getYpos() ==
BOTTOM_MOST_PIXEL )
                  return true;
      }
      return false;
```

```java
    }

    //return number of remaining enemies
    public int getEnemyCount() {
      return gameMap.enemies.size();
    }
}

package controller;
import java.util.ArrayList;
import model.*;

//Author: Erdinc
//abstract factory for level creator
public abstract class LevelCreatorFactory {
    //attributes
    int numberOfEnemies;
    int[] typeOfEnemies;
    int numberOfPowerUps;
    ArrayList<Enemy> enemy;

    //methods
    public abstract ArrayList<Enemy> createEnemies();
    public abstract ArrayList<PowerUp> createPowerUp();
}


package controller;
import java.awt.Rectangle;
import java.util.ArrayList;

import model.*;
public class Level2Creator extends LevelCreatorFactory{

    //constructor
    public Level2Creator() {
      numberOfEnemies = 15;
      typeOfEnemies = new int[3];
      typeOfEnemies[0] = 10;
      typeOfEnemies[1] = 5;
      typeOfEnemies[2] = 0;
      numberOfPowerUps = (int) ((Math.random() * 3) + 1);
    }

    public ArrayList<Enemy> createEnemies() {
      ArrayList<Enemy> enemies = new ArrayList<Enemy>();
      //enemy type 0
      for ( int j = 0; j < typeOfEnemies[0]; j++) {
            enemies.add( new Enemy(0,j,2,new Rectangle()));
      }
      //enemy type 1
      for ( int j = 0; j < typeOfEnemies[1]; j++) {
            enemies.add( new Enemy(1,j,1,new Rectangle() ));
      }
```

```
        //enemy type 2
        for ( int j = 0; j < typeOfEnemies[2]; j++) {
                enemies.add( new Enemy(2,j,0,new Rectangle() ));
        }
        return enemies;
      }

      public ArrayList<PowerUp> createPowerUp() {
        //decide type of power ups
        int type;
        ArrayList<PowerUp> powerup = new ArrayList<PowerUp>();
        int[] enemiesBound = new int[ numberOfPowerUps];
        int enemyBound;
        boolean bound;

        for ( int i = 0; i < numberOfPowerUps; i++) {
                bound = false;
                type = (int)(Math.random() * 3);
                //bind powerUp to an enemy
                enemyBound = (int) (Math.random() *
numberOfEnemies);
                for ( int j = 0; j < numberOfPowerUps; j++)
                        if ( enemiesBound[j] == enemyBound)
                                bound = true;
                if ( !bound) {
                        PowerUp newPowerUp;
                        if ( type == 0) //add health
                                newPowerUp = new PowerUp(0,0,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
                        else if ( type == 1) //speed up
                                newPowerUp = new PowerUp(1,20,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
                        else //destructive power
                                newPowerUp = new PowerUp(2,20,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
//                      enemy.get( enemyBound).addPowerUp(
newPowerUp);
                        enemiesBound[i] = enemyBound;
                        powerup.add( newPowerUp);
                }
                else
                        i--;
        }
        return powerup;
      }
}


package controller;
//Author: Erdinc
import java.awt.Rectangle;
```

```
//Description: creates all types of enemies, bind powerUps
accordingly
import java.util.ArrayList;
import model.*;

public class Level1Creator extends LevelCreatorFactory {

    //constructor
    public Level1Creator() {
      numberOfEnemies = 15;
      typeOfEnemies = new int[3];
      typeOfEnemies[0] = 15;
      typeOfEnemies[1] = 0;
      typeOfEnemies[2] = 0;
      numberOfPowerUps = (int) ((Math.random() * 2) + 1);
    }

    //methods
    public ArrayList<Enemy> createEnemies() {
      ArrayList<Enemy> enemies = new ArrayList<Enemy>();
      //enemy type 0
      for ( int j = 0; j < typeOfEnemies[0]; j++) {
            enemies.add( new Enemy(0,j,2,new Rectangle()));
      }
      //enemy type 1
      for ( int j = 0; j < typeOfEnemies[1]; j++) {
            enemies.add( new Enemy(1,j,1,new Rectangle() ));
      }
      //enemy type 2
      for ( int j = 0; j < typeOfEnemies[2]; j++) {
            enemies.add( new Enemy(2,j,0,new Rectangle() ));
      }
      return enemies;
    }

    public ArrayList<PowerUp> createPowerUp() {
      //decide type of power ups
      int type;
      ArrayList<PowerUp> powerup = new ArrayList<PowerUp>();
      int[] enemiesBound = new int[ numberOfPowerUps];
      int enemyBound;
      boolean bound;

      for ( int i = 0; i < numberOfPowerUps; i++) {
            bound = false;
            type = (int)(Math.random() * 3);
            //bind powerUp to an enemy
            enemyBound = (int) (Math.random() *
numberOfEnemies);
            for ( int j = 0; j < numberOfPowerUps; j++)
                  if ( enemiesBound[j] == enemyBound)
                        bound = true;
            if ( !bound) {
                  PowerUp newPowerUp;
```

```
                if ( type == 0) //add health
                     newPowerUp = new PowerUp(0,0,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
                else if ( type == 1) //speed up
                     newPowerUp = new PowerUp(1,20,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
                else //destructive power
                     newPowerUp = new PowerUp(2,20,
enemy.get(enemyBound).getXpos(),
enemy.get(enemyBound).getYpos());
//              enemy.get( enemyBound).addPowerUp(
newPowerUp);
                powerup.add( newPowerUp);
                enemiesBound[i] = enemyBound;
           }
           else
                i--;
      }
      return powerup;
    }
}


package controller;
import java.util.ArrayList;
import java.util.Vector;

import view.*;
import model.*;

public class GameEngine extends Thread{
    private GameMap gameMap;
    private GameData gameData;
    private GUIManager guiManager;
    private PhysicsEngine physicsEngine;

    private int score;
    private int level;
    private int difficulty;
    private int remainingLife;
    private int powerUp;

    private boolean leftPressed;
    private boolean rightPressed;
    private boolean spacePressed;

    private boolean isInTop;

    //main loop
    private boolean running;
    private int FPS = 60;
    private long targetTime = 1000 / FPS;

    //Constructor
    public GameEngine( GUIManager guiManager, GameMap gameMap,
GameData gameData, int level, int diff)
    {
      this.gameMap = gameMap;
      this.gameData = gameData;
      this.guiManager = guiManager;
      this.level = level;
      this.difficulty = diff;
      score = 0;
      remainingLife = 3;
      leftPressed = false;
      rightPressed = false;
      spacePressed = false;
      setPowerUp(0);
      physicsEngine = new PhysicsEngine( gameMap, difficulty);
    }

    //Getter
    public int getScore()
    {
      return score;
    }

    //Setter
    public void setLeftPressed(boolean b){
      leftPressed = b;
      System.out.println("I set left pressed to " + b);
    }
    public void setRightPressed(boolean b){
      rightPressed = b;
    }
    public void setSpacePressed(boolean b){
      spacePressed = b;
    }

    //Methods
    public void gameStart()
    {
      gameMap.setMap(level,difficulty);

      running = true;
      this.start();

    }
    //game loop
    public void run()
    {
      long start;
      long elapsed;
      long wait;
```

```
        while(running)
        {
            start = System.nanoTime();

            updateGame();
            //guiManager.draw();
            //guiManager.drawToScreen();

            elapsed = System.nanoTime() - start;
            /*wait = targetTime - elapsed / 1000000;

            if( wait < 0) wait = 5;
            try{
                Thread.sleep(wait);
            }catch(Exception e) {
                e.printStackTrace();
            }*/

        }

        guiManager.endGame(score,isInTop); //end game and show
    score
                                        //calls isInTop() & ask for
    name etc.
      }

      public void updateGame()
      {
        physicsEngine.moveEnemies();
        if ( leftPressed) physicsEngine.moveShip(0);
        if ( rightPressed) physicsEngine.moveShip(1);
        if( spacePressed) physicsEngine.fire();

        leftPressed = rightPressed = spacePressed = false;

        boolean shipCollision = physicsEngine.checkCollision();
    //ship vs. enemies & bombs
        if( shipCollision) remainingLife--;

        int enemyDestroyed = physicsEngine.isEnemyDestroyed();
    //enemies vs. & fires
        score += enemyDestroyed;

        if( isGameOver()) running = false;
      }

      public void setShipModel(int model)
      {
        gameMap.setShipModel(model);
      }

      public int getEnemyCount()
      {
        return physicsEngine.getEnemyCount();
      }

      public boolean isDestroyed()
      {
        if( remainingLife == 0)
            return true;
        return false;
      }


      public boolean isGameOver()
      {
        if( isDestroyed() || physicsEngine.hasEnemyArrived())
            return true;

        return false;
      }

      public ArrayList<String> getHighScores()
      {
        return gameData.getHighScores();
      }

      public boolean isInTop(int score)
      {
        return gameData.isInTopList(score);

      }

      public void updateHighScores(String name, int score)
      {
        gameData.updateHighScores(name, score);
      }

      public int getPowerUp() {
        return powerUp;
      }

      public void setPowerUp(int powerUp) {
        this.powerUp = powerUp;
      }

    }

    package model;


    import java.awt.Color;
    import java.awt.Graphics;
    import java.awt.Point ;
    import java.awt.Rectangle;
    //import java.awt.Event;
    import java.awt.image.BufferedImage ;
    import java.io.File;
```

```java
import javax.imageio.ImageIO;
//import javax.swing.JPanel;

public class Ship extends GameObject
{
    //Properties

    // Bu classta ship'in
    //carpısma algoritması için gemi bi circle ve r si var
    final int r=10;

    //private int x , y ;
    //Rectangle bounds;

    private int lifeEnergy;
    private int shipType ;
    private int destroyForce;

    public Ship(int shipType, int x, int y)
    {
      this.setShipTypeImg(shipType);
      this.xPos=x;
      this.yPos=y;
      //this.bounds = bounds;
      this.setLifeEnergy(3) ;
    }

    //Methods
    public boolean moveRight()
    {
      xPos+=cell ;
      //if(!bounds.contains(xPos,yPos))
      //    return false;
      return true;
    }

    public boolean moveLeft()
    {
      xPos-=cell ;
      //if(!bounds.contains(xPos,yPos))
           //return false;
      return true;
    }

    public boolean moveY()
    {
      return true;
    }

    public int getShipType()
    {
      return shipType;
    }
    public void setShipType(int type)
    {
      shipType = type ;
    }

    public void setShipTypeImg(int shipType)
    {
      try{
            String png = "ship"+ shipType +".png";
            System.out.println(png);
            image = ImageIO.read(new
File("/Users/muhammedemincs/git/crazy-space-
war/src/ship1.png"));
      }
      catch(Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
      }

      this.setShipType(shipType) ;
    }

    public int getLifeEnergy() {
      return lifeEnergy;
    }

    public void setLifeEnergy(int lifeEnergy) {
      this.lifeEnergy = lifeEnergy;
    }

    public boolean doesCollideEnemy(Enemy e)
    {
      if(Math.pow(xPos- e.getXpos(),2)+Math.pow(yPos-
e.getYpos() ,2) > 4*Math.pow(r, 2) )
            return false;
      else
            return true;
    }

    public boolean doesContain(Point p)
    {
      return Math.pow(xPos-p.x, 2)+Math.pow(yPos- p.y, 2) <
Math.pow(r, 2) ;
    }

    public int getDestroyForce() {
      return destroyForce;
    }

    public void setDestroyForce(int destroyForce) {
      this.destroyForce = destroyForce;
    }
}
```

```
package model;

public class Settings
{
    private int difficulty ;
    private boolean music , sound ;


    public Settings() {
      this.setDifficulty(1) ;
      this.setMusic(true) ;
      this.setSound(true) ;

    }


    public Settings( int difficulty, boolean music , boolean
sound)
    {
      this.setDifficulty(difficulty) ;
      this.setMusic(music) ;
      this.setSound(sound) ;

    }

    public Settings changeSettings(int difficulty, boolean
music , boolean sound)
    {
      this.difficulty = difficulty ;
      this.setMusic(music) ;
      this.sound = sound ;
      return this;
    }

    public int getDifficulty() {
      return difficulty;
    }


    public void setDifficulty(int difficulty) {
      this.difficulty = difficulty;
    }


    public boolean isSound() {
      return sound;
    }


    public void setSound(boolean sound) {
      this.sound = sound;
    }
    public boolean isMusic() {
      return music;
    }


    public void setMusic(boolean music) {
      this.music = music;
    }


}


// Model part of Project
// Author Muhammed Emin Öztürk


package model;


import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point ;
import java.awt.Rectangle;
//import java.awt.Event;
import java.awt.image.BufferedImage ;
import java.io.File;
import javax.imageio.ImageIO;
//import javax.swing.JPanel;

public class PowerUp extends GameObject
{
    //Properties
    private BufferedImage image ;
    private int powerUpkind;
    private int duration;
    //private int x,y ;
    Rectangle bounds ;
    private boolean taken;

    public PowerUp(int kind, int duration , int x , int y )
    {
      this.setPowerUpType(kind);
      this.duration=duration ;
      taken = false ;

      xPos =x ;
      yPos =y ;

    }

    public void setPowerUpType(int powerUpType) {
      try{
           image = ImageIO.read(new
File("powerUp"+powerUpType+ ".png")) ;
      }
      catch(Exception e){

      }

      this.setPowerUpkind(powerUpType);

    }
    public boolean moveRight()
    {
      return true;
    }

    public boolean moveLeft()
    {
      return true;
    }

    public boolean moveY()
    {
      this.yPos+=cell/3;
      return true;
    }

    public boolean isTaken()
    {
      taken = true;
      return taken;
    }


    public int getPowerUpkind() {
      return powerUpkind;
    }


    public void setPowerUpkind(int powerUpkind) {
      this.powerUpkind = powerUpkind;
    }


    public int getDuration() {
      return duration;
    }


    public void setDuration(int duration) {
      this.duration = duration;
    }
```

```
}
package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point ;
import java.awt.Rectangle;
import java.awt.Event;
import java.awt.image.BufferedImage;




public abstract class GameObject {

    //Properties


    public final int cell = 10;
    protected int xPos;
    protected int yPos;
    protected BufferedImage image ;
    //Methods

    abstract boolean moveLeft();
    abstract boolean moveRight();
    abstract boolean moveY();
    //Get and Set Methods

    public int getXpos()
    {
      return xPos ;
    }

    public int getYpos()
    {
      return yPos ;
    }

    public void setXpos(int x)
    {
      xPos = x;
    }

    public void setYpos(int y)
    {
      yPos = y ;
    }
    public BufferedImage getCurrentImage(){
      return image;
    }
}
```

```
package model;
import controller.*;

import java.util.ArrayList;

import javax.swing.*;

import java.awt.Graphics2D;

import javax.swing.JPanel;




public class GameMap {

    //Properties

    private Ship ship ;
    public ArrayList<Bomb> bombs ;
    public ArrayList<PowerUp> powerUps;
    public ArrayList<Fire> fires;
    public ArrayList<Enemy> enemies;
    private LevelCreatorFactory factory ;


    public GameMap()
    {
      ship = new Ship(1 , 100, 100);
      bombs = new ArrayList<Bomb>() ;
      //powerUps = new ArrayList<PowerUp>() ;
      fires = new ArrayList<Fire>() ;
      //enemies = new ArrayList<Enemy>() ;
    }


    //Methods

    public void setMap(int lev, int dif)
    {
      if(lev==1)
      {
           factory = new Level1Creator();

      }

      else
      {
           factory = new Level2Creator() ;

      }
      enemies = factory.createEnemies();
```

```
		//powerUps = factory.createPowerUp();
	}


	public Ship getShip()
	{
		return ship;
	}

	public void addPowerUp(PowerUp pw)
	{
		powerUps.add(pw);
	}

	public void addFire(Fire fr)
	{
		fires.add(fr);
	}

	public void addEnemey(Enemy en)
	{
		enemies.add(en);
	}

	public void setShipModel(int model)
	{
		this.ship.setShipTypeImg(model);

	}

}
package model;

public class GameLevel
{
	//Properties

	int gameLevel ;
	int [][] levelInfo;







}


package model;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
```

```java
import java.io.IOException;
import java.util.ArrayList;



public class GameData
{
    //Properties
    GameLevel level ;
    Settings setting ;
    private int[] topList;

    //Methods


    public GameData()
    {
      level = new GameLevel() ;
      setting = new Settings();
      topList = new int[3];

    }

    public void  writeToFileExample(String str) {

            try {

                    String content = str ;

                    File file = new File("score.txt");

                    // if file doesnt exists, then create it
                    if (!file.exists()) {
                            file.createNewFile();
                    }

                    FileWriter fw = new
FileWriter(file.getAbsoluteFile());
                    BufferedWriter bw = new BufferedWriter(fw);
                    bw.write(content);
                    bw.close();

                    System.out.println("Done");

            } catch (IOException e) {
                    e.printStackTrace();
            }
    }

    //to be done
    public ArrayList<String> getHighScores()
    {
      ArrayList<String> l = new ArrayList<String>();
      //get list from file and return
```

```
        return l;
    }


    //to be done
    public boolean isInTopList(int score)
    {
        return true;
    }

    //to be done
    private void readScoreToFile()
    {

    }

    //writes new score to list
    public void updateHighScores(String name, int score) {
        String str = name + " " + score + "\n";
        writeToFileExample(str);
    }



}


package model;


import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point ;
import java.awt.Rectangle;
import java.awt.Event;
import java.awt.image.BufferedImage ;
import java.io.File;

import javax.imageio.ImageIO;

public class Fire extends GameObject
{
    //Properties
     private int destroyForce ;
     private int kind ;
     Rectangle bounds;

     //Constructor

     public Fire()
     {
        destroyForce = 100 ;
        setKind(0) ;
     }
     public Fire(int fireType, int x, int y,  Rectangle
bounds)
    {
      destroyForce = 100 ;
     this.setFireType(fireType);
     this.setFireTypeImg(fireType);
     this.xPos=x;
     this.yPos=y;
     this.bounds = bounds;
     System.out.println("fire created");

    }

    //Methods

    public void setFireType(int type)
    {
           kind = type ;
    }

    public void setFireTypeImg(int fireType)
    {
     try{
                image = ImageIO.read(new File("ship1.png"));
     }
     catch(Exception e){
     }
    }

    public int getDestroyForce()
    {
     return destroyForce ;
    }

    public void setDestroyForce(int newForce)
    {
     destroyForce = newForce ;
    }

    public boolean moveY()
    {
     yPos -= cell/5 ;
     return true;
    }

    public boolean moveRight()
    {
     return true;
    }

    public boolean moveLeft()
    {
     return true;
    }


    public int getKind() {
      return kind;
    }

    public void setKind(int kind) {
      this.kind = kind;
    }


}


package model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point ;
import java.awt.Rectangle;
import java.awt.Event;
import java.awt.image.BufferedImage;
import java.io.File;

import javax.imageio.ImageIO;

public class Enemy extends GameObject
{
    //Properties
    private BufferedImage image ;
    private int lifeEnergy ;
    private int enemyKind ;
    Rectangle bounds ;
    private int r ;

    //Constructor

    public Enemy(int enemyType, int x, int y, Rectangle
bounds)
    {
      this.setEnemyType(enemyType);
      this.bounds = bounds ;

      lifeEnergy = 1;
      xPos=x ;
      yPos=y ;

    }

    public void setEnemyType( int enemyType)
    {
      try{
           image = ImageIO.read(new
File("enemy"+enemyKind+".png"));
      }
      catch(Exception e){

      }

    }

    //Methods
    public int getLifeEnergy()
    {
      return lifeEnergy ;
    }

    public int getEnemyKind()
    {
      return enemyKind ;
    }

    public void setEnemyKind(int kind)
    {
      enemyKind = kind;
    }

    public void setLifeEnergy(int lifeEnergy)
    {
      this.lifeEnergy = lifeEnergy ;
    }


    public boolean moveY()
    {
      yPos -= cell ;
      return true;
    }

    public boolean moveRight()
    {
      xPos += cell;
      return true;
    }

    public boolean moveLeft()
    {
      xPos -= cell ;
      return true ;
    }


    public boolean doesCollideEnemy(Ship ship)
    {
      if(Math.pow(xPos- ship.getXpos(),2)+Math.pow(yPos-
ship.getYpos() ,2) > 4*Math.pow(r, 2) )
            return false;
      else
           return true;
    }

    public boolean doesContain(Point p)
    {
      return Math.pow(xPos-p.x, 2)+Math.pow(yPos- p.y, 2) <
Math.pow(r, 2) ;
    }





}


package model;

import java.awt.Rectangle;

public class Boss extends Enemy
{
    public Boss(int enemyType, int x, int y, Rectangle bounds)
    {
      super( enemyType,  x,  y,  bounds);

    }

}
package model;


import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point ;
import java.awt.Rectangle;
import java.awt.Event;
import java.awt.image.BufferedImage;
import java.io.File;

import javax.imageio.ImageIO;


public class Bomb extends GameObject {

    //Properties

    private int destroyPower;
    private int kind ;
    private int r ;

    private  BufferedImage image ;
    Rectangle bounds;


    //Constructor
    public Bomb(int bombType, int x, int y,  Rectangle bounds)
    {
      this.setBombType(bombType); // GitHub push deneme
      this.setKind(bombType);
      this.xPos=x;
      this.yPos=y;
      this.bounds = bounds;
      destroyPower = 1;
    }

    //Methods

    public void setBombType(int shipType)
    {
      try{
              image = ImageIO.read(new 
File("ship"+shipType+".png"));
      }
      catch(Exception e){

      }
      this.setKind(shipType);
    }

    public boolean moveLeft()
    {
      xPos = xPos - cell ;
      return true;
    }

    public boolean moveRight()
    {
      xPos = xPos + cell ;
      return true;
    }

    public boolean moveY()
    {
      yPos = yPos - cell;
      return true;
    }

    public boolean doesCollideEnemy(Ship sp)
    {
      if(Math.pow(xPos- sp.getXpos(),2)+Math.pow(yPos-
sp.getYpos() ,2) > 4*Math.pow(sp.r, 2) )
              return false;
      else
              return true;
    }

    public boolean doesContain(Point p)
    {
      return Math.pow(xPos-p.x, 2)+Math.pow(yPos- p.y, 2) <
Math.pow(r, 2) ;
    }

    public int getDestroyPower() {
      return destroyPower;
    }

    public void setDestroyPower(int destroyPower) {
      this.destroyPower = destroyPower;
    }

    public int getKind() {
      return kind;
    }

    public void setKind(int kind) {
      this.kind = kind;
    }




}
```