Bilkent University

Department of Computer Engineering

CS319
Object Oriented Software Engineering

Design Report

Crazy Space War

Group Members

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

Assistant Professor Can Alkan

December 3, 2014

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

Table of Contents

1.Introduction 3
1.1.Purposes Of The System 3
1.2. Design Goals 3
Trade offs *4*
1.3. Definitions, acronyms and abbreviations 4
1.4. References 4
1.5. Overview 4

2.Software Architecture 5
2.1. Overview 5
2.2. Subsystem Decomposition 5
2.3. Hardware/Software Mapping 6
2.4 Persistent Data Management 7
2.5 Access Control and Security 7
2.6 Boundary Conditions 7
Initialization *7*
Termination *8*
Failure *8*

3.Subsystem Services 9
3.1. Detailed Object Design 9
3.2. Design Patterns 9
Model – View- Controller Architectural Style (MVC) *9*
3-Layer Architectural Style *10*
Façade *10*
Delegation *10*
3.3. User Interface Subsystem 11
3.3.1. GUIManager Class *12*
3.3.2. KeyController Class *12*
3.3.3. GamePanel Class *13*
3.3.4. SettingsPanel Class *13*
3.3.5. Help Panel Class *14*
3.3.6. MenuPanel Class *14*
3.3.7. PausePanel Class *15*
3.3.9. HighScorePanel Class *15*
3.4. Game Management Subsystem 16
3.4.1. Game Engine Class *16*
3.4.2. Physics Engine Class *17*
3.4.3. Game Loop Class *18*
3.5. Game Entities Subsystem 19
3.5.1. Game Level Class *20*
3.5.2. Settings Class *20*
3.5.3. Game Data Class *20*
3.5.4. Game Map Class *21*
3.5.5. Game Object Class *21*
3.5.6. Enemy Class *21*
3.5.7. Power Up Class *22*
3.5.8. Bomb Class *22*
3.5.9. Ship Class *23*
3.5.10. Fire Class *23*

1.Introduction

1.1.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.

1.2. 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.

1.3. Definitions, acronyms and abbreviations

Abbreviations:MVC: [2] Model View Controller

JDK: JVM:

1.4. 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

1.5. 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.

2.Software Architecture

2.1. 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.

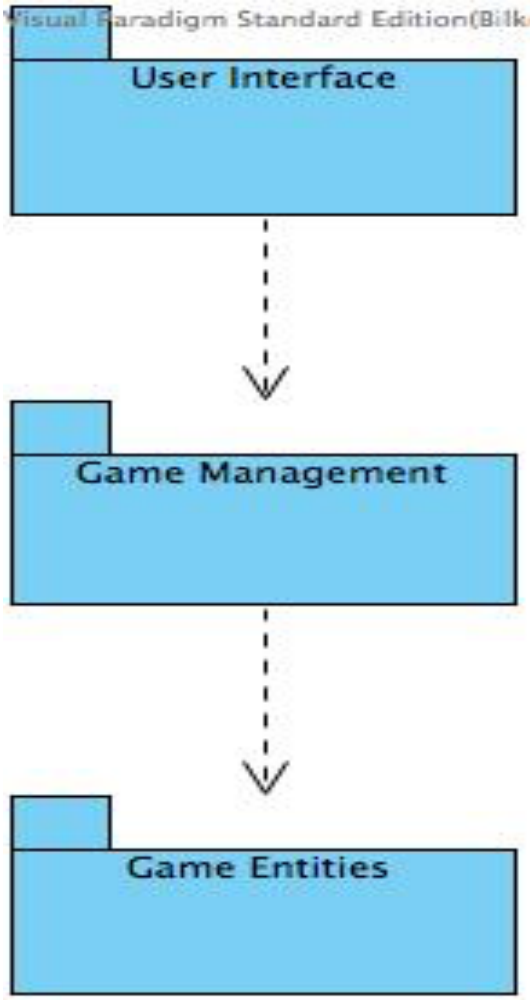
2.2. Subsystem Decomposition

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.

2.3. 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.

2.4 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.

2.5 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.

2.6 Boundary Conditions

Initialization

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

Termination

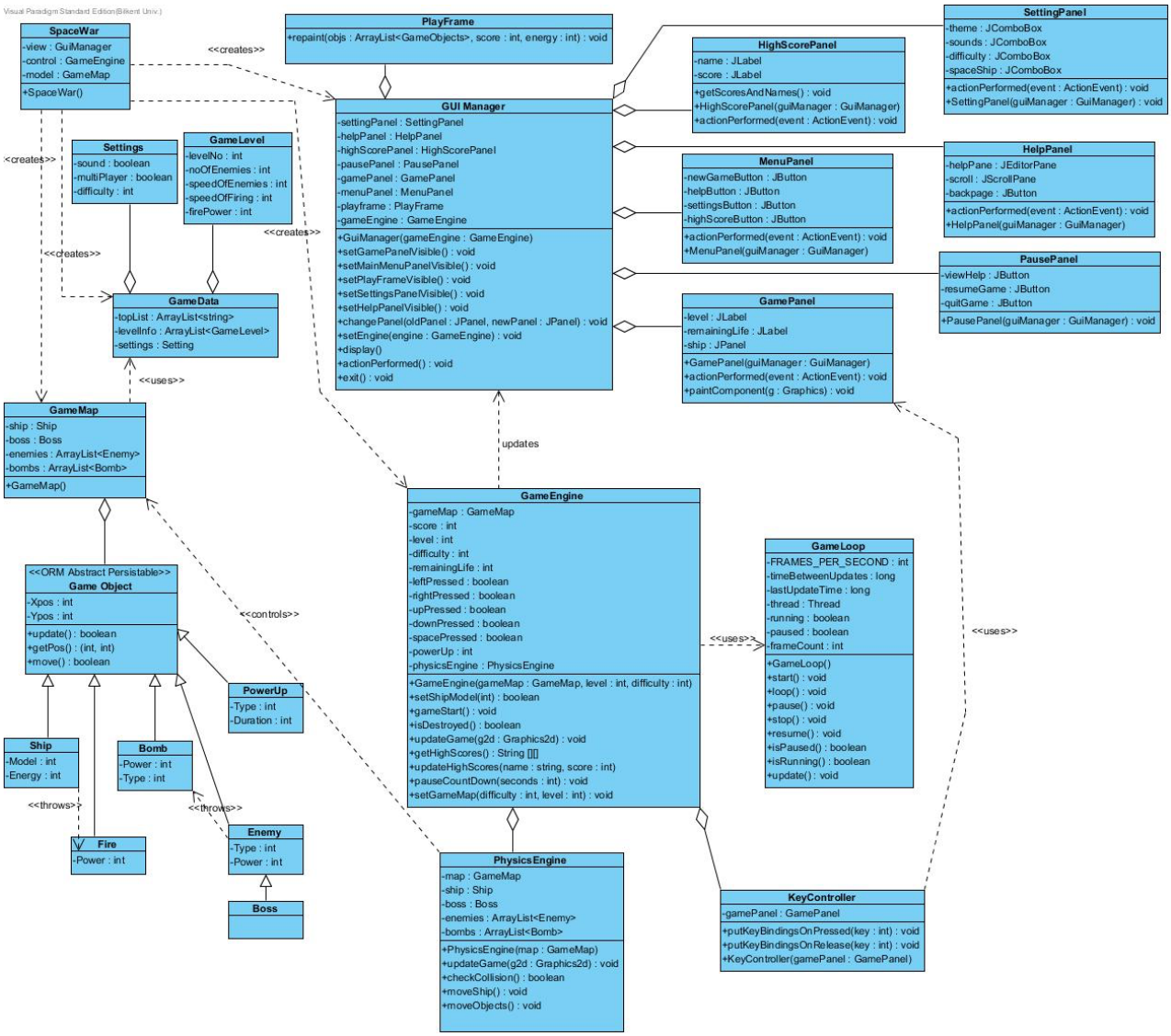
The game can be closed only using the Exit Game button in the main menu. During the game player would need to pause and quit the game in order to arrive at the main menu. Since the game will work on full screen, the 'X' button will not be available to close the game.

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.

3.Subsystem Services

3.1. Detailed Object Design



3.2. Design 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.

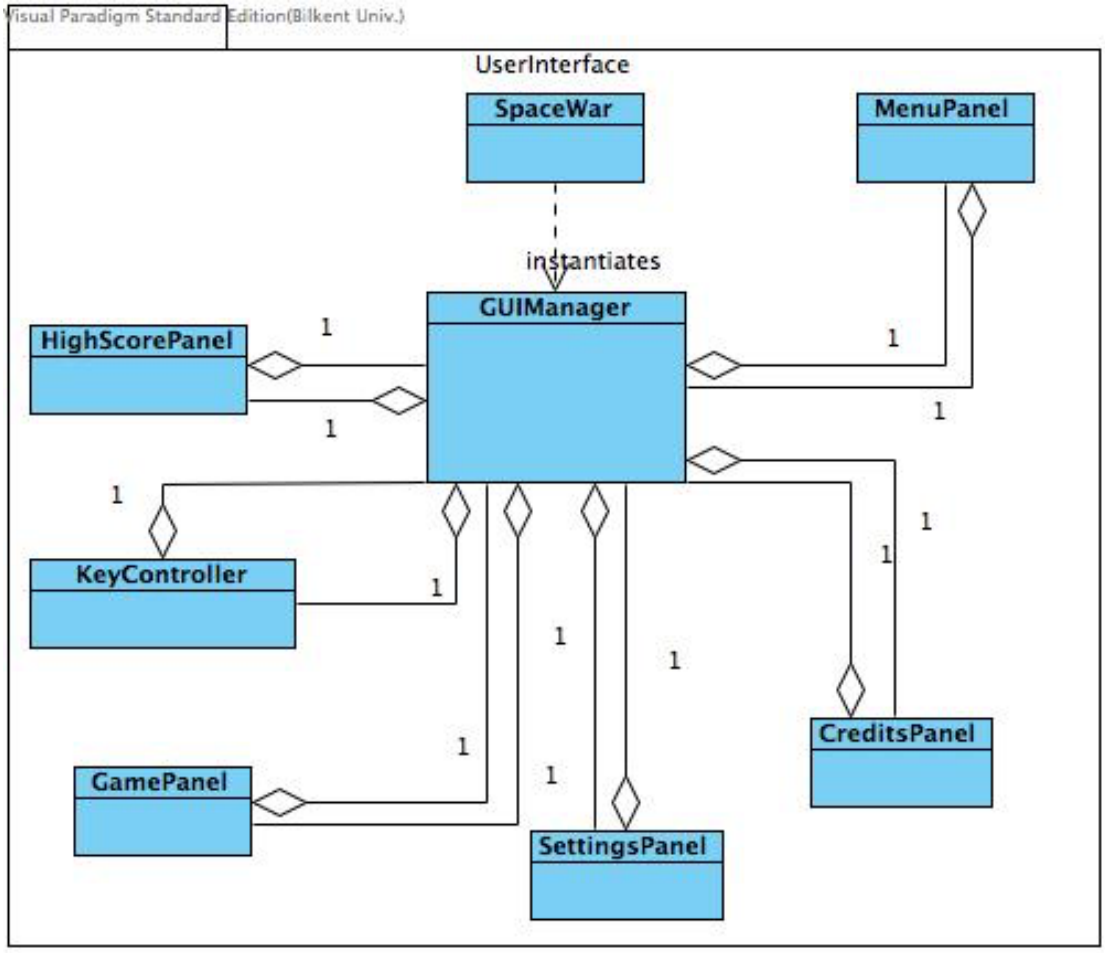
Façade

Game Engine class is the interface class of the Game Management subsystem, the User Interface subsystem only know about Game Engine class and its methods so Game Engine is a Façade class. Also only GUIManager is connected with other subsystems in the user interface subsystem. So GUIManager is also a Façade class.

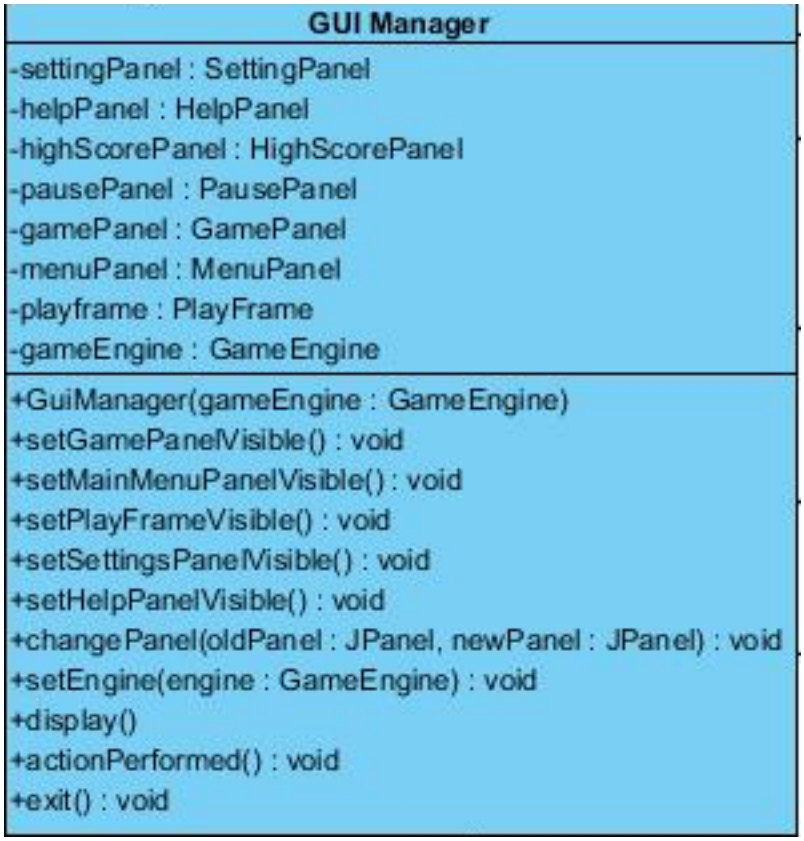
Delegation

To control game object movements Game Engine delegates Physics Engine. UpdateGame() method of GameEngine calls updateGame() method of PhysicsEngine at each cycle of the game loop.

3.3. User Interface Subsystem



3.3.1. GUIManager Class



It is the controller class of the user interface subsystem. Also the communication between other subsystems of user interface subsystem is handled via this class so it Is a Façade class.

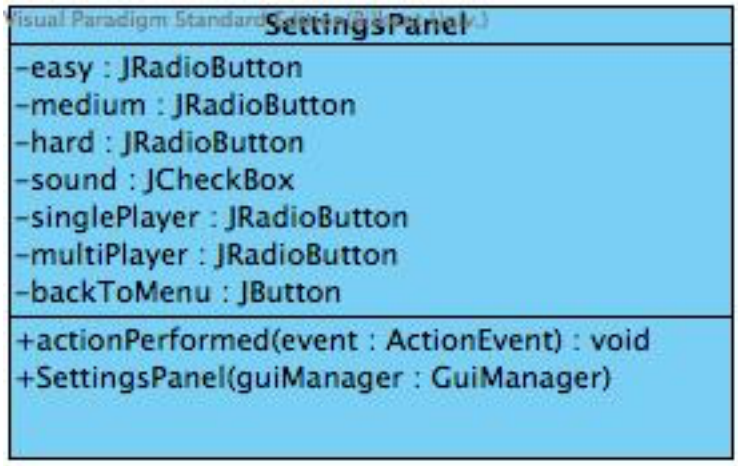
3.3.2. KeyController Class

It is initialized by GameEngine. It responds event which was triggered GamePanel . Player uses keyboard to control ship in the game. Especially when user uses arrow this controller class was triggered .

3.3.3. GamePanel Class

This class is a panel that is used during gameplay. The class instantiates labels and panels to display the state of the game information such as current level, current score, energy of the ship and remaining lives. Class also implements ActionListener which processes user input to play and pause the game. paintComponent method repaints the background and graphical representations of instances of Game Object class.

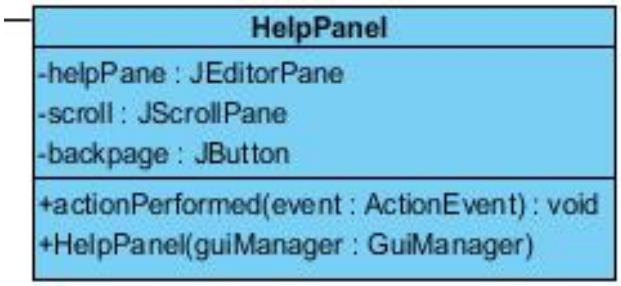
3.3.4. SettingsPanel Class



SettingsPanel is to hold the specifications about the game. There are three setting in that panel. They are difficulty level, sound and multiplayer. There are three difficulty levels, a sound on or off option and multiplayer or single player option. User can

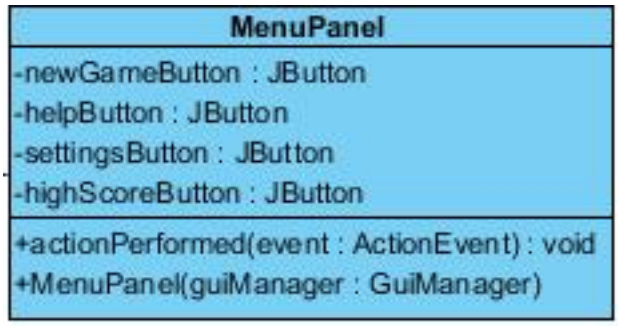
choose one of them thanks to a radio button and play game according to that specific options.

3.3.5. Help Panel Class



HelpPanel class shows help menu that can be accesses MenuPanel or pausepanel . During game user press 'P' to pause game. When player open help menu he/she can get information games rule and specific tips for level.

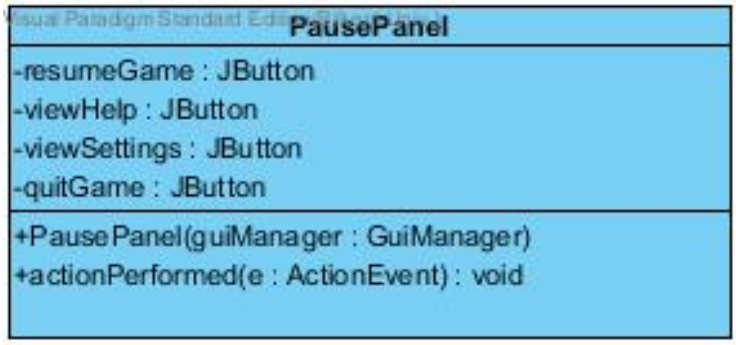
3.3.6. MenuPanel Class



MenuPanel class shows the opening screen of the game. It includes 4 buttons, which are newGameButton, helpButton, changeSettingsButton, highScoreButton. Thus player can access NewGame, help menu , setting menu and high score menu the via Menu Panel . In the actionPerformed method, by using the GuiManager class's methods and attributes, MenuPanel class changes

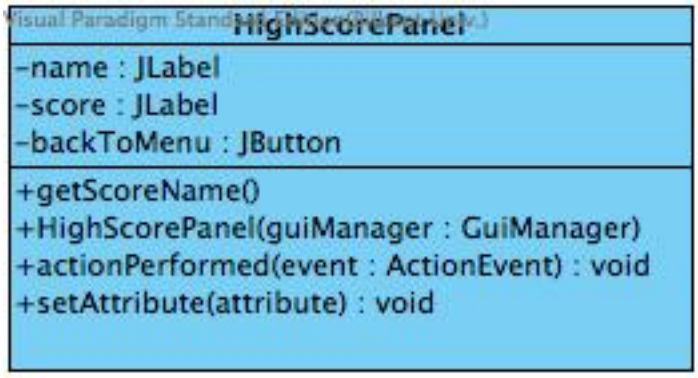
the screen and displays the new panel

3.3.7. PausePanel Class



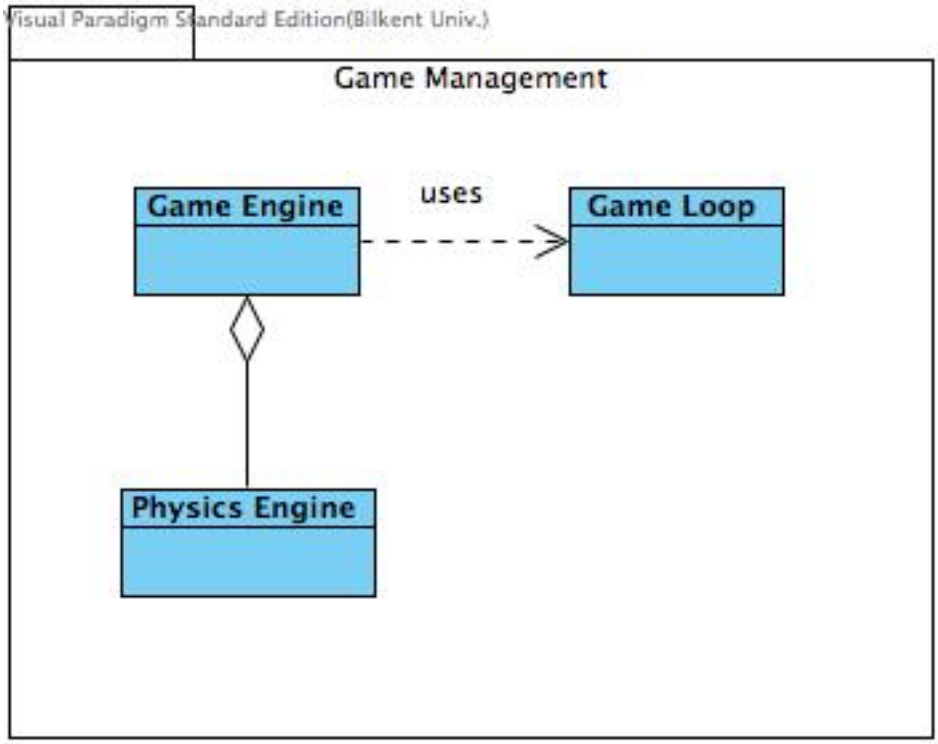
This class is a panel that is shown when the user pauses the game. The class instantiates buttons and implements ActionListener interface. The panel navigates the view using this set of buttons. Every button invokes another panel in the body of actionPerformed method.

3.3.9. HighScorePanel Class



HighScorePanel is to show the high scores to the user. This class holds name and score of the user who get the high score.

3.4. Game Management Subsystem



Game Management is responsible for game logic. The Game Engine Class is a Façade class. User Interface classes use Game Engine to communicate with Game Management subsystem.

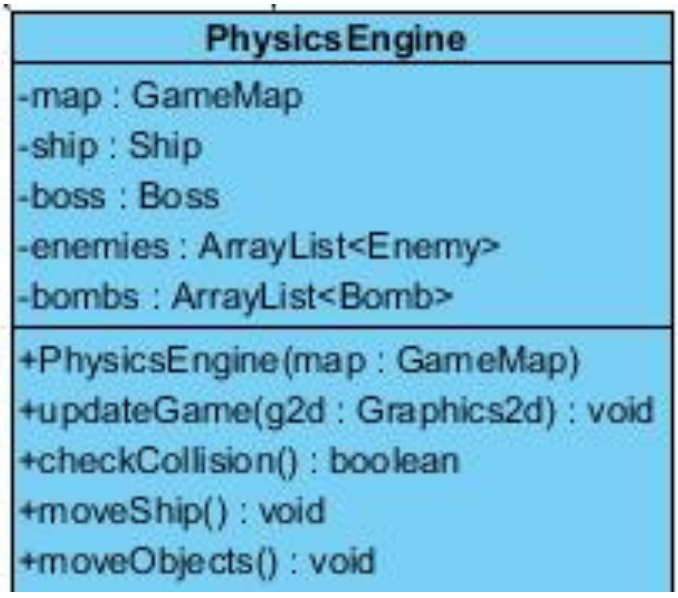
3.4.1. Game Engine Class

We will use fixed time step game loop so that each cycle through the loop takes a constant amount of time. Since we do a lot of geometric calculations and collusion detection, the amount of these can vary greatly depending on the situation. That's why we chose fixed time step to cover these irregularities.

As seen in the class diagram, GameEngine is an abstract class. The reason for this is to be able to mix in our game's operations into the Swing event loop. In the SpaceWar class, a GameEngine object is created and passed to GuiManager's constructor as a parameter. Then only when our GameLoop decides to update the game at each cycle, the repaint() method of GamePanel is called. repaint() calls paintComponent() and paintComponent() calls updateGame() method on GameEngine.

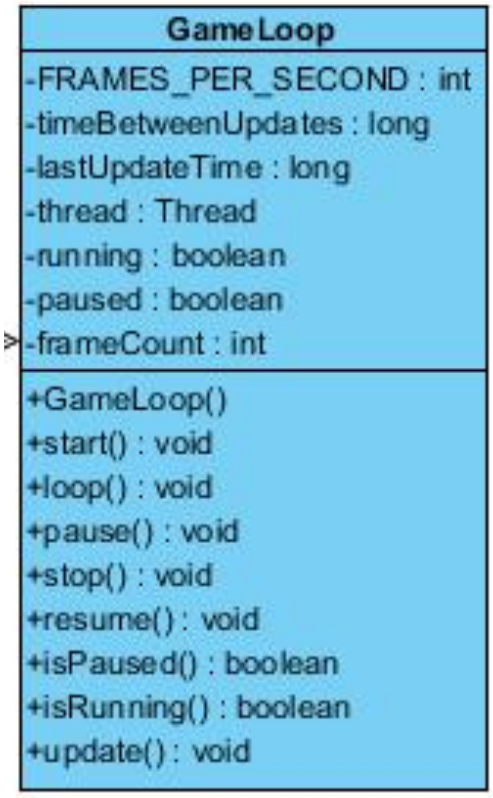
3.4.2. Physics Engine Class

Note that the collision detection, bound checking , and actually drawing the game object is done by the PhysicsEngine class. PhysicsEngine have several methods that will be explained below. When updateGame() of Game Engine is called , GameEngine calls updateGame() method of PhysicsEngine to get the job done.



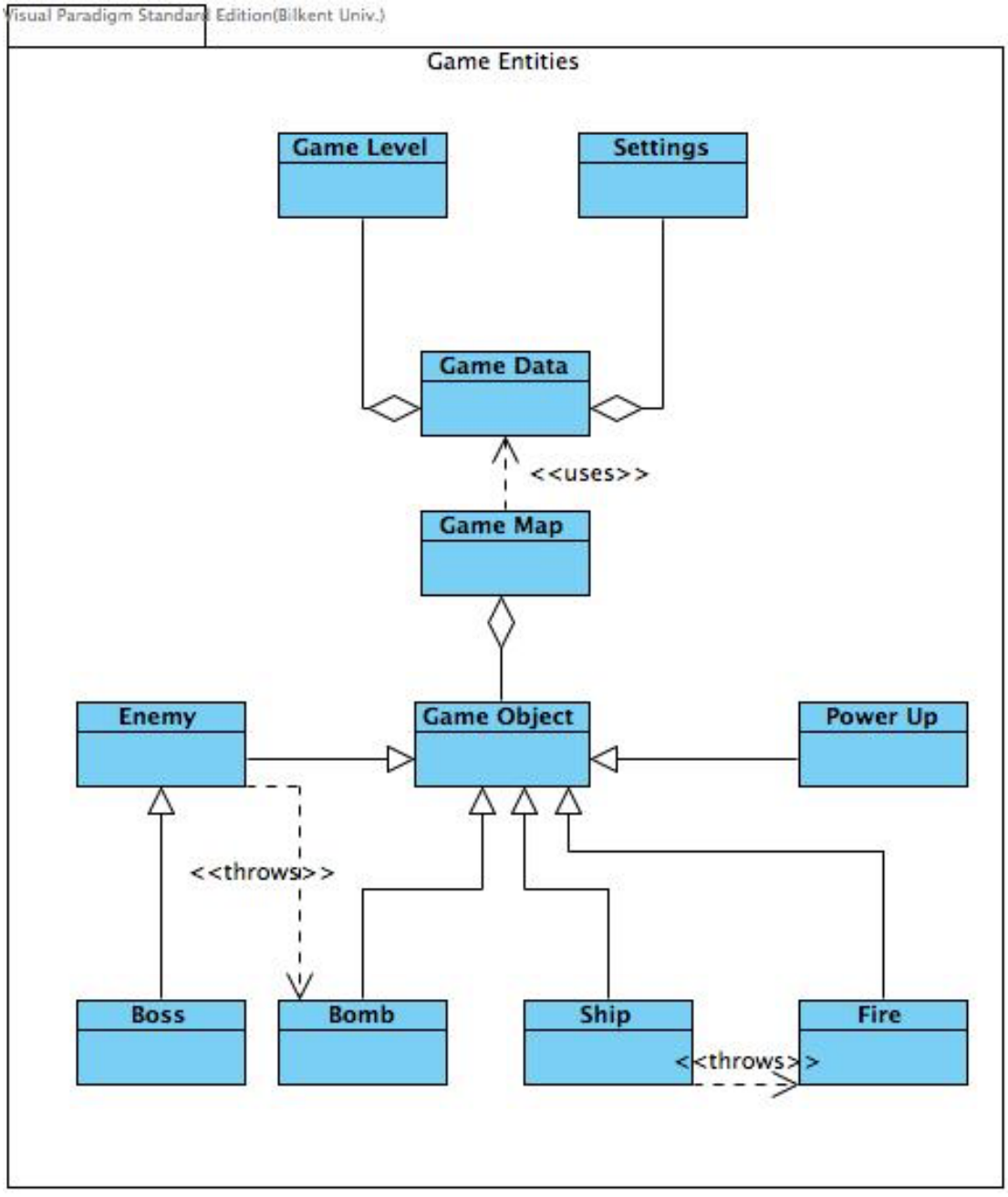
PhysicsEngine handles all game objects and the game map. It changes the map whenever any aliens is destroyed. It makes the aliens move and check collision with checkCollision() and moveObjects() methods. In the other words, it can be said that PhysicsEngine makes all game objects move, then it draws everything by calling the draw methods of these objects and it detects collision. Since these operations are handled by PhysicsEngine , the entity classes are very simple. They keep position, speeds and images.

3.4.3. Game Loop Class



GameEngine class uses GameLoop class to control the game loop. We restructured the methods related with the loop such as start(), pause(), etc. to the GameLoop class. This made the design more understandable.

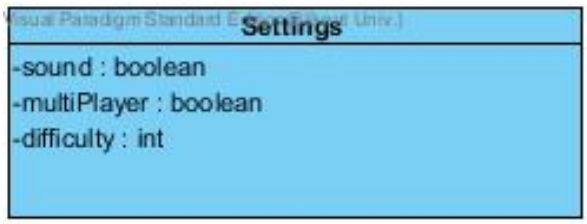
3.5. Game Entities Subsystem



3.5.1. Game Level Class

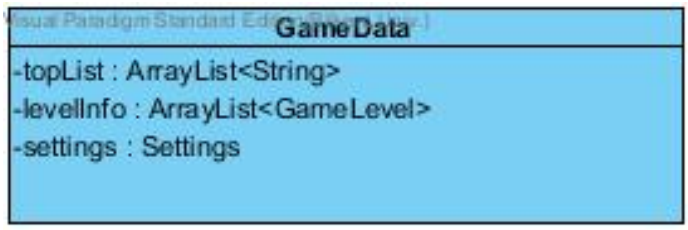
This class stores data about the contents of individual levels such as number and type of enemies, time limit if it exists, speed of enemies.When a new game is started or a level is completed, this data is loaded into GameMap to start the new level. Instances of this class are stored in GameData class.

3.5.2. Settings Class



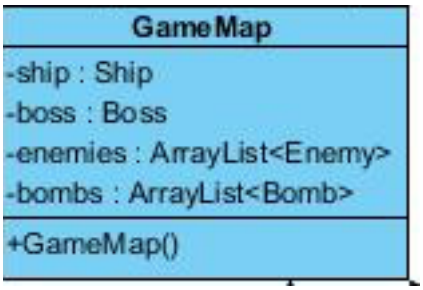
This class stores the settings of the game such as sound status, multi player status and difficulty. An instance of this class is stored in GameData class to be distributed.

3.5.3. Game Data Class



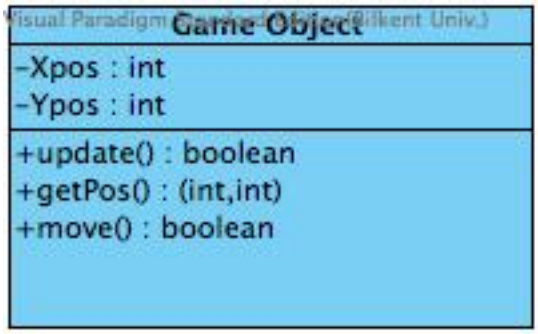
This class stores persistent data for the game such as high scores, information for each level and settings. Class is responsible for distributing data and updating data as it changes.

3.5.4. Game Map Class



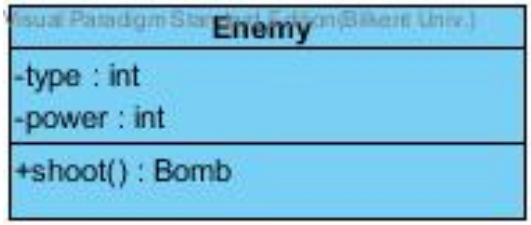
The GameMap class specifies the position of enemies and bombs. It contains ship and boss as attributes. It keeps all enemies object and determine bombs position at anytime.

3.5.5. Game Object Class



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 classes.

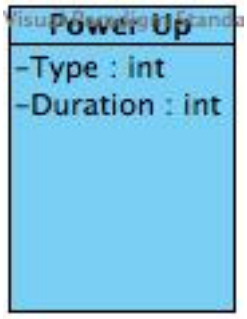
3.5.6. Enemy Class



This class is a child of Game Object class. Hence, Enemy objects have a position on the map and can move. power attribute indicates its health. Most significantly, shoot method creates Bomb

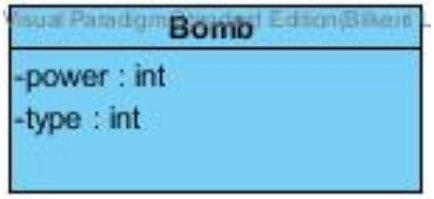
objects depending on the type attribute. If Enemy objects collide with a Ship object, it deals damage to the Ship object and is destroyed in the process.

3.5.7. Power Up Class



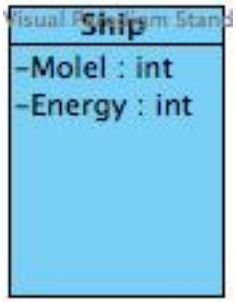
Power Up class has two attributes. They are Type and Duration, Type is to determine which power up the user got and duration

3.5.8. Bomb Class



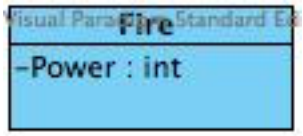
This class is a child of Game Object class. Hence, Enemy objects have a position on the map and can move. Instances of this class are created by Enemy objects. If Bomb objects collide with a Ship object, it deals damage to the Ship object and is destroyed in the process.

3.5.9. Ship Class



Ship is a game object and has model and Energy attributes other than Xpos and Ypos attributes. Model is to determine the type of the ship, and Energy is for calculating how many damage can ship handle in that particular time.

3.5.10. Fire Class



Fire is a game object and has only one different attribute that is power. Power represents the how many damage the enemy gets from a fire.