Bilkent University

Department of Computer Engineering

CS319 Object-Oriented Software Engineering Project

Siege

Design Report

Project Group 26: İrem Yüksel, Gökhan Şimşek, Emir Acımış, Furkan Küçükbay

Course Instructor: Uğur Doğrusöz

November 28, 2016

Table of Contents

1. Introduction

1.1. Purpose of the System

Siege is a game aiming to entertain users while they are trying to protect the sacred object from the enemies. It is influenced by the classical game Tank 1990. The game will have different levels, each have different maps and difficulties.By adding some power-ups and different types of enemies, we tried to increase the player's attention. The game will be successfully completed if the player can prevent the enemy tanks from capturing the sacred object.

1.2. Design Goals

1.2.1. Adoptability

The game Siege will be implemented in Java. The main reason we preferred Java to other programming languages such as C or C++ is that Java provides cross platform portability. So the users will be able to run and play the game in any platform.

1.2.2. Efficiency

The game must run fast enough to respond each movement of every object such as bullets and the player's tank. Since the player's lives depend on it, this is crucially important that the game runs smoothly.

1.2.3. Reliability

Siege will be bug-free and will not crush while the user plays the game. In order to provide this goal, the tests will be done for each part separately and continuously. To prevent the game from crushing with unexpected inputs, we will define different exceptions which will not affect the game.

1.2.4. Usability

The system will be designed so that any user can easily understand and interact even on the first opening of the game. While being as easy as possible, it will also be interesting for the user’s excitement. This will be provided with different graphics.

1.2.5. Extendability

We will structure the system in a way that it will enable us to customize it for any further improvements. It should be designed in a way that these improvements will not harm the structure of the original design and cause any crushes.

2. Software Architecture

2.1. Subsystem Decomposition

For the system design, MVC architectural style is chosen. As this architectural style prompts, the system is decomposed into three subsystems: Model, View and Controller. The model subsystem is developed such that they do not rely on any view or controller subsystem. The reason behind choosing MVC is that the components become less susceptible to any further changes. If, for instance, the interface is decided to be changed, Model subsystem would never be affected by that change. As well as this extendibility issue, the decomposition of the system shares out the complexity among the subsystems.

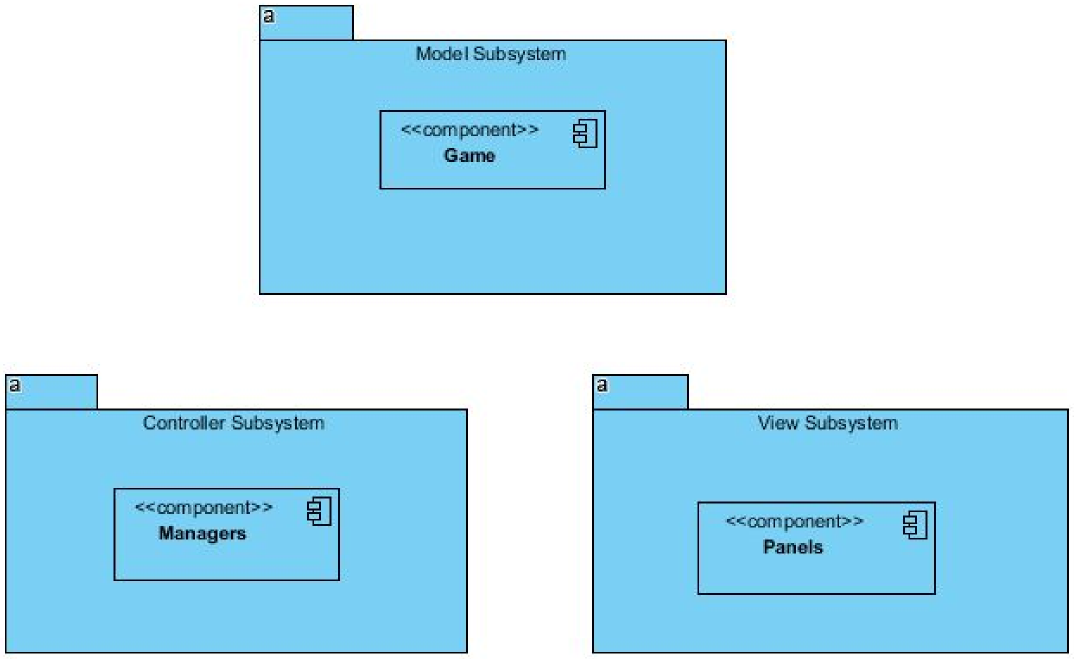


Figure 1: Subsystem Decomposition Diagram: The system is designed to have an MVC Architectural Style. Thus, three subsytems are Model, View and Controller subsytems. These subsystems serve for Data, GUI and Management respectively

2.1.1 Model Subsystem

In the Model subsystem, domain knowledge is maintained. Model objects and their relationships are represented in this subsystem. Basically, Model subsystem is responsible for keeping the data.

2.1.2 View Subsystem

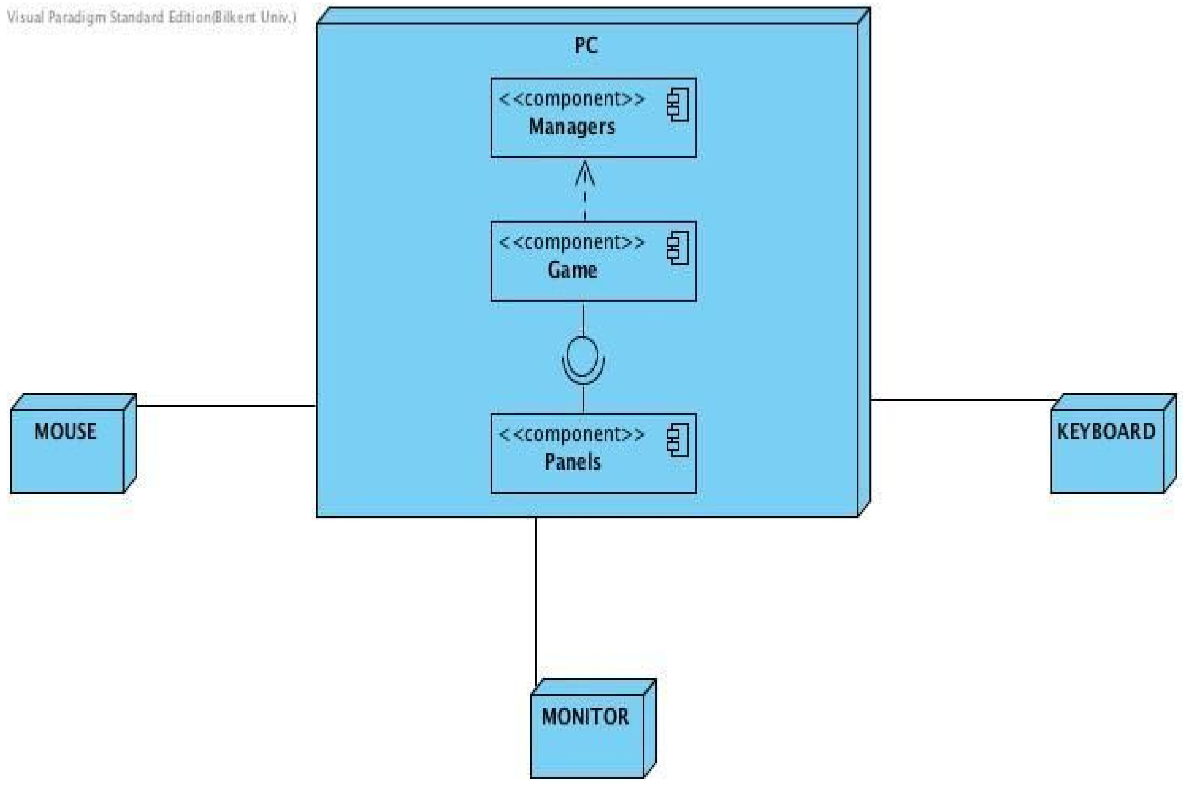
View subsystem involves view components. From the start to the end, View Subsystem will be responsible from displaying each specific screen to the user. Whenever the Model Subsystem changes, renderings of the objects will be handled through this subsystem as well.

2.1.3 Controller Subsystem

Controller subsystem shelters Manager objects which are responsible for controlling the sequence of interactions in the game logic.

2.2. Hardware/Software Mapping

The Siege game will be developed by the Java programming language and we will use the last development kit of Java which is called JDK8. For playing the game the users should have Java compiler inside their computer, if not, the users cannot play the game. When user play the game, user just use monitor and keyboard of the PC. When the user types the name to the High Score list the user use keyboard and mouse for typing their name. For the high score list we will use .txt file, for sound we will use .wav and for the images we will use .png format. Therefore, the users' computers should support and have .txt, .wav and .png. There is no need an internet connection for the game. The following diagram shows the communication between the hardware and software components of the game. When we create the game we will use Model-Controller-View architecture.



2.3. Persistent Data Management

In the game Siege there is no need to use a complex database system because the high score system will just keep 5 best score and this will be done by the text files. In the game there is no load function so that when the user terminates from the game the user will restart the game, the game does not keep old games. The sound and images are not much complex and we will keep them into the hard disk of the computer. The format of the image is .png and the format of the sound is .wav.

2.4. Access Control and Security

The game does not require any internet connection. After user open the game, the game can be played easily. There will be no limitation for accessing the game. The game does not require a user profile, that's why the game Siege is a secure game too.

2.5. Boundary Conditions

Initialization

- When the player opens the .jar file he/she can access the game. The game does not require any install.
- When the user pushes the play button from the main menu, the user can access the game and levels,
- When player pushes the help button then the user see the controls of the game,
- When the Settings is open the user can control the settings functions,
- When the user pushes the High Score button then the user see the 5 best high score of the game.

Termination

- When the user pushes to the Exit button from the main menu, the program will be terminated.
- While user playing the game and wants to quit the game, the user press the pause button and return the main menu and terminate the program by pressing Exit button.
- Player can close the game by pushing the ‘X’ from the Panel.

Failure

- If the user computer does not have Java compiler, then the program will not open and execute.
- If the files like image, sound do not inside the computer then the program will run but the other staffs of program like image, sound will not work.
- Possible hardware problem can affect the program.
- Cutting the power of computer.

3. Subsystem Services

The proposed system will follow the MVC model, and thus will be divided into three parts named model, view and controller. As a brief introduction to their services, it can be said that the view subsystem is responsible for displaying system to the user, the model subsystem maintains the domain knowledge, and the controller subsystem handles user inputs and makes changes to the system accordingly.

3.1. Services of the Model

getObjectStatus: This is a service that the Model subsystem provides to the Controller subsystem. Model provides the current status of the game objects to the Controller, such as the remaining hits for a destroyable object to be destroyed, the remaining time for a power-up before it disappears, remaining lives of the player tank, current locations of the objects on the layout.

updateView: This is a service that the Model subsystem provides to the View subsystem. With this service, the information processed will be provided to the user to be viewed.

3.2. Services of the View

showView: The View subsystem provides the visuals of the game to the user. According to the inputs coming from the Model subsystem, View subsystem changes the respective locations and stati of the objects, and shows this to the user.

3.3. Services of the Controller

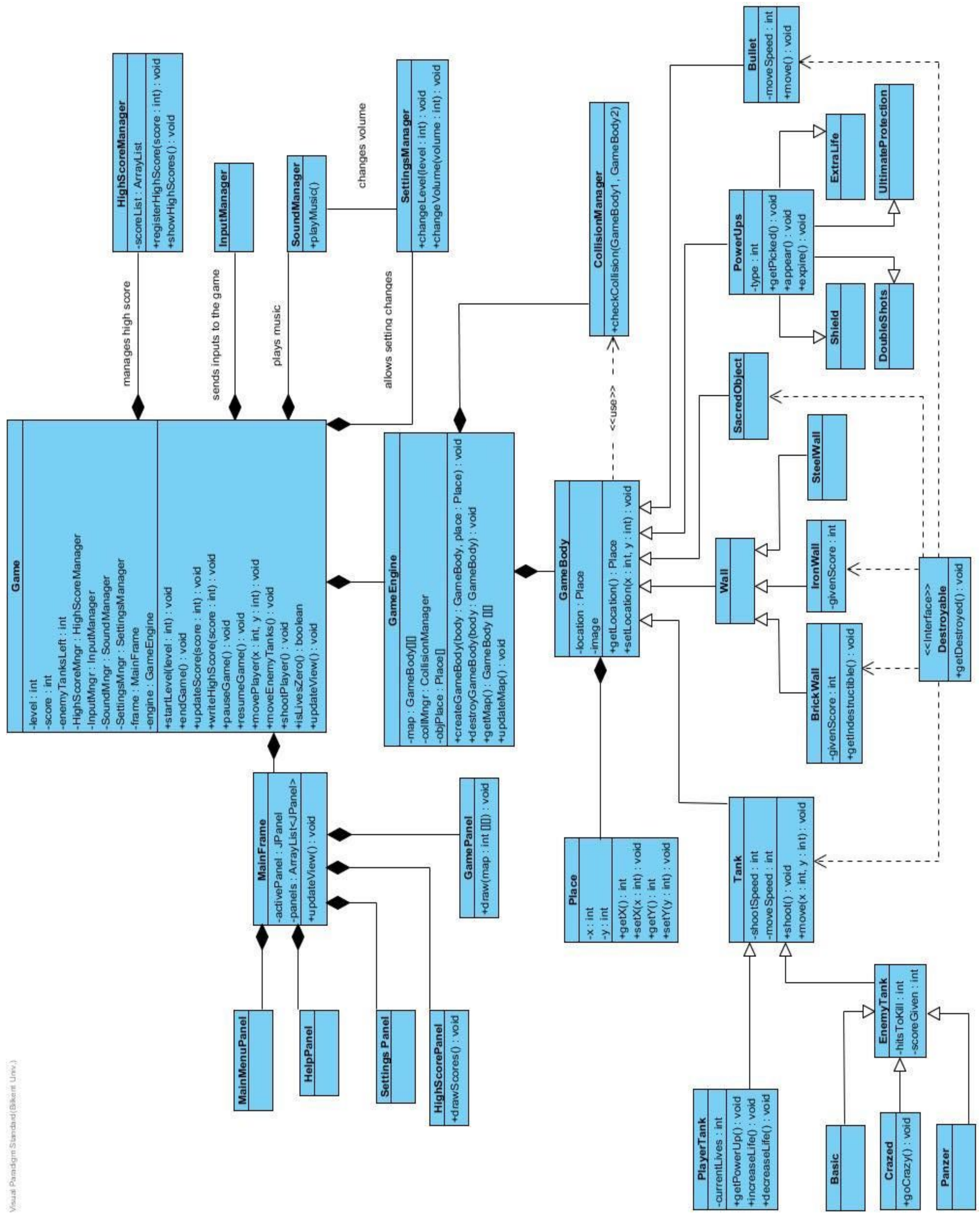
updateObjectStatus: After the necessary operations are performed by the controller, in light of the current state of the game gathered from the Model subsystem and the provided user inputs, the Controller subsystem sends the changes that should be made to the Model subsystem. e.g. after checking for collisions between a bullet and a wall, if the wall had required only 1 hit to be destroyed, Controller informs the Model to update the said Wall and Bullet objects, which the Model will in turn destroy both of those objects.

4. Low-Level Design

4.1. Object Design Trade-Offs

Ease of Use vs. Functionality: In order to reach a wider audience, and provide games that can be played by different age groups with equal ease, we have decided to make the game as easy to play as possible. To achieve this, some functionalities that might make the game much more complicated were left out.

Extendibility vs. Performance: The game is designed in a way to allow extending easily for future additions, as this is just a first version of the game. This reflects on the design patterns and subsystems that are used in the design process. Due to these considerations, creating subsystems and increasing the number of communications between the classes decreases the performance.


Game
-level : int
-score : int
-enemyTanksLeft : int
-HighScoreMngr : HighScoreManager
-InputMngr : InputManager
-SoundMngr : SoundManager
-SettingsMngr : SettingsManager
-frame : MainFrame
-engine : GameEngine
+startLevel(level : int) : void
+endGame() : void
+updateScore(score : int) : void
+writeHighScore(score : int) : void
+pauseGame() : void
+resumeGame() : void
+movePlayer(x : int, y : int) : void
+moveEnemyTanks() : void
+shootPlayer() : void
+isLivesZero() : boolean
+updateView() : void
HighScoreManager
-scoreList : ArrayList
+registerHighScore(score : int) : void
+showHighScores() : void
manages high score
InputManager
sends inputs to the game
SoundManager
+playMusic()
plays music
SettingsManager
+changeLevel(level : int) : void
+changeVolume(volume : int) : void
changes volume
allows setting changes
GameEngine
-map : GameBody[][]
-collMngr : CollisionManager
-objPlace : Place[]
+createGameBody(body : GameBody, place : Place) : void
+destroyGameBody(body : GameBody) : void
+getMap() : GameBody [][]
+updateMap() : void
CollisionManager
+checkCollision(GameBody1, GameBody2)
<<use>>
GameBody
-location : Place
-image
+getLocation() : Place
+setLocation(x : int, y : int) : void
Place
-x : int
-y : int
+getX() : int
+setX(x : int) : void
+getY() : int
+setY(y : int) : void
MainFrame
-activePanel : JPanel
-panels : ArrayList<JPanel>
+updateView() : void
GamePanel
+draw(map : int [][]) : void
MainMenuPanel
HelpPanel
Settings Panel
HighScorePanel
+drawScores() : void
Bullet
-moveSpeed : int
+move() : void
PowerUps
-type : int
+getPicked() : void
+appear() : void
+expire() : void
ExtraLife
UltimateProtection
Shield
DoubleShots
SacredObject
Wall
SteelWall
IronWall
-givenScore : int
BrickWall
-givenScore : int
+getIndestructible() : void
<<Interface>>
Destroyable
+getDestroyed() : void
Tank
-shootSpeed : int
-moveSpeed : int
+shoot() : void
+move(x : int, y : int) : void
PlayerTank
-currentLives : int
+getPowerUp() : void
+increaseLife() : void
+decreaseLife() : void
EnemyTank
-hitsToKill : int
-scoreGiven : int
Basic
Crazed
+goCrazy() : void
Panzer
Visual Paradigm Standard (Bilkent Univ.)

4.2. Final Object Design

4.2.1. Façade Pattern:

The system is designed according to the Model-Controller-View (MVC) architectural style. To implement this style and categorize classes according to their own subsystem, a façade pattern is utilized. To make communication easier between subsystems, each façade pattern in the UML class diagram can be taken as the head classes of such clusters, which can be classified as: Game class represents the Controller, MainFrame represents the View, GameEngine represent the Model. All classes that are connected to these head classes can be considered in their respective façade patterns.

4.2.2. Singleton Pattern:

Since there can only be one active game at any given time, the singleton pattern is applied to the Game class. There can only be one Game class, and any newly created Game class starts a whole new process with default values for Game class and its dependent classes.

4.3. Packages

4.3.1. Controller Package

Controller Package is the brain of the system and involves the control mechanisms for the system. Other packages are also created and managed by Controller Package. In order to use the system, Game class of Controller Package is utilized. The package also shelters HighScoreManager, SoundManager, InputManager and SettingsManager under its roof.

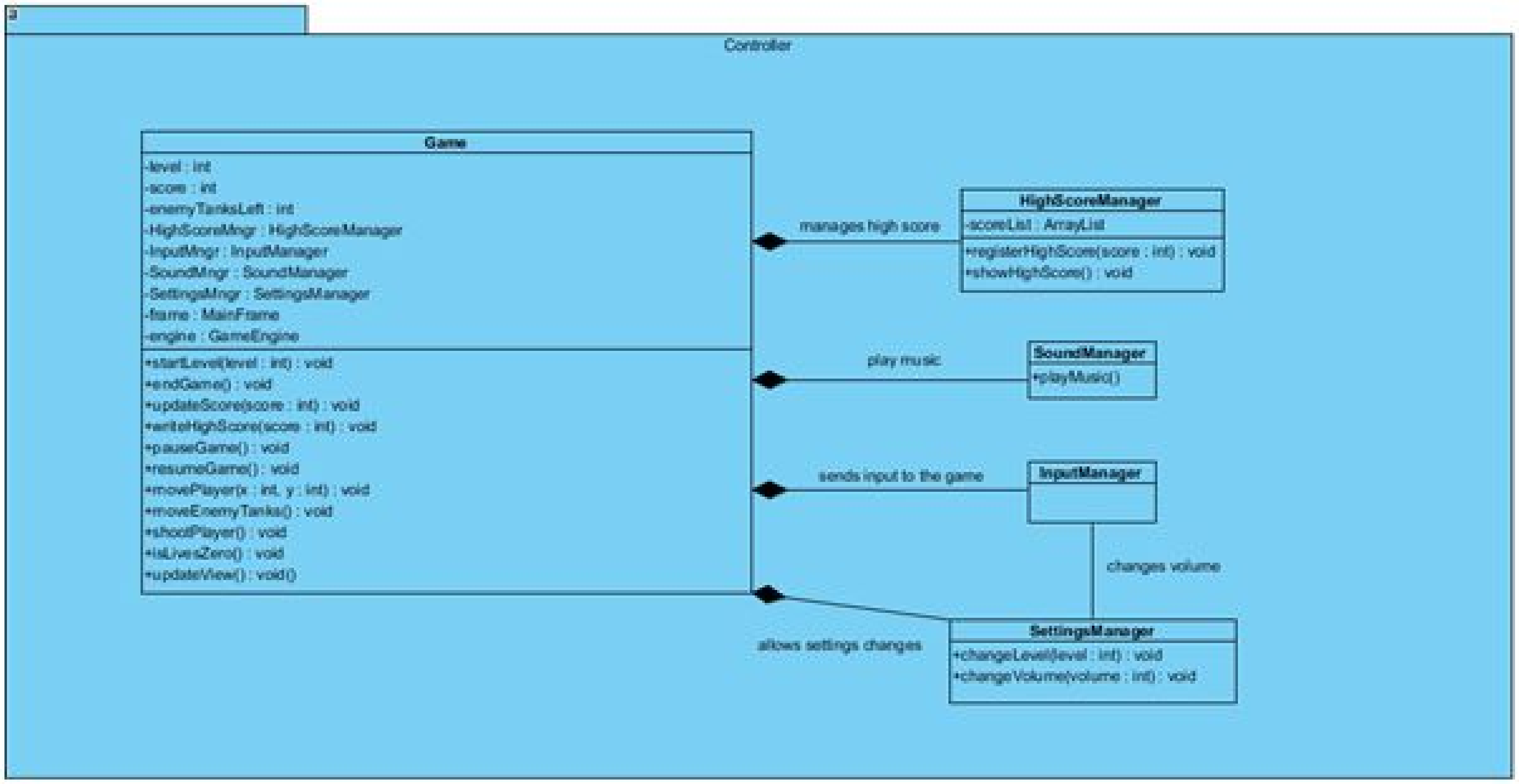


Figure 4: Controller Package

4.3.2. Model Package

Model Package involves the classes that work as the entity objects of the system as well as model mechanisms. Tank, Wall, SacredObject and PowerUps hold for the classes bearing entity objects whereas GameEngine and GameBody hold for the Model mechanism. Controller subsystem reaches Model subsystem via GameEngine class. In order to invoke the same method for the objects destroyed, Tank, Bullet, SacredObject and Wall classes implement the "Destroyable" interface. Objects are updated by GameEngine through using data coming from GameBody class.

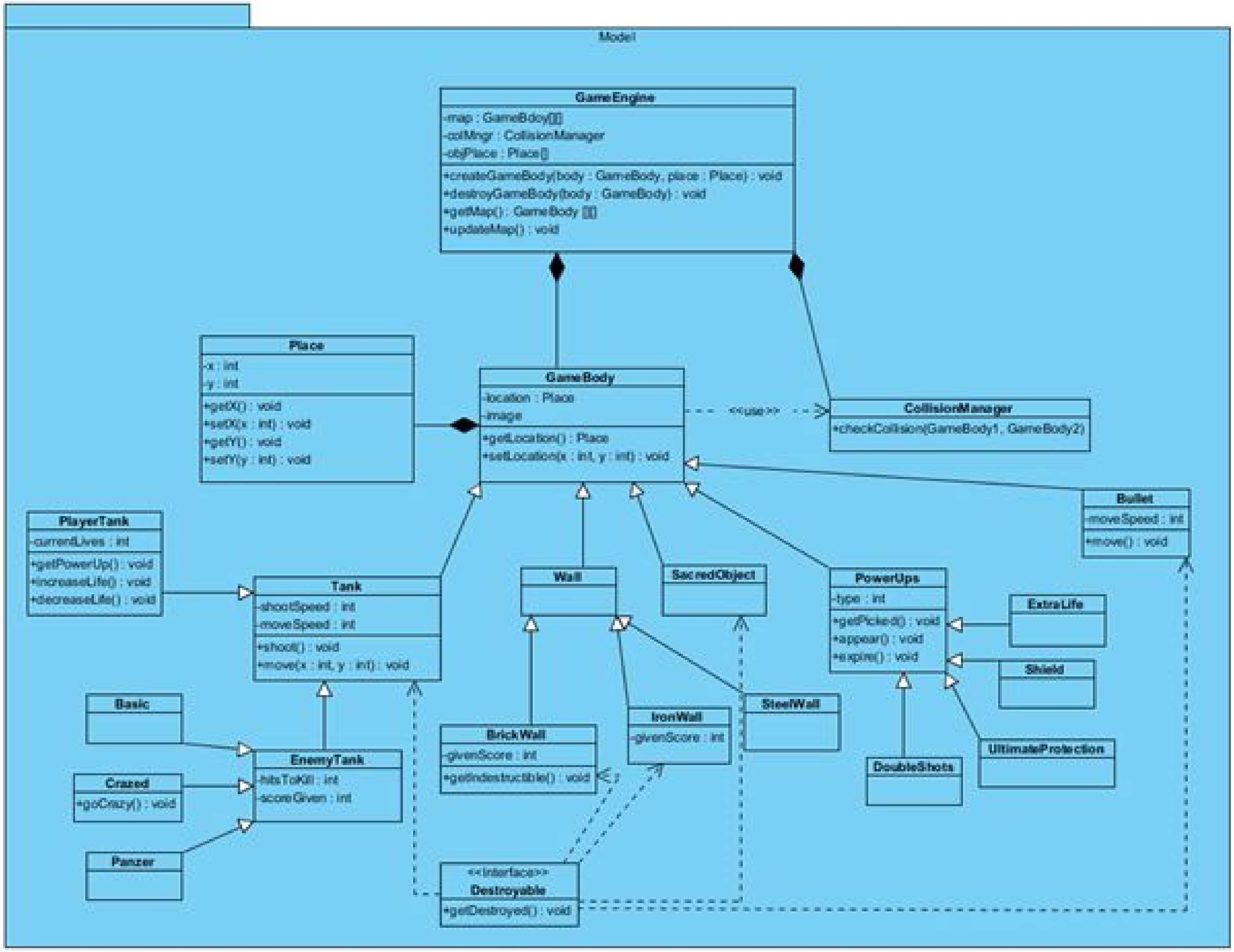


Figure 5: Model Package

4.3.3. View Package

View Package works as the boundary of the system. The package involves JFrame and JPanels as well as action listeners to interact with the user. Each use case is shown to the user with different panels like HelpPanel, SettingsPanel and so on under the control of MainFrame. The Controller subsystem updates the view by reaching persistent Model data.

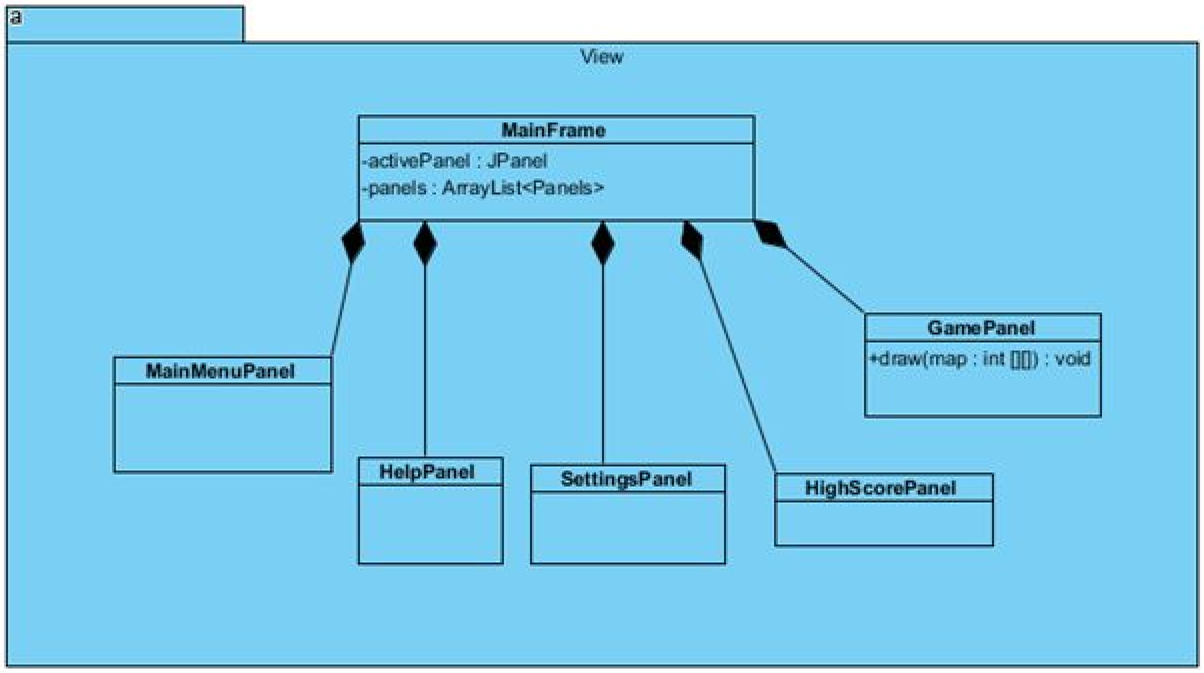


Figure 6: View Package

4.4. Class Interfaces

4.4.1. Controller Classes

Controller packet contains all the information about the manager classes which means controller of the each menu and the objects of the game. This packet contains GameClass which is the main controller of the game that controls all of the classes on the controller packet, HighScoreManger, InputManager, GameEngin, SoundManager, SettingsManager and CollisionManager.

1) **Game Class**

Game class is the main class for the game and for the controller and its control all the game manager classes which are HighScoreManager, InputManager, SoundManager and SettingsManager. These are control all the game functionality, levels and different inputs like images and sound.

a) **Attributes:**

private int level: This attribute hold the current level of the game which is played by the player.

private int score: This attribute hold the current score of the game which is also displayed to the panel.

private int enemyTankLeft: This attribute holds how many enemy tank is left in the game.

private HighScoreManager HighScoreMngr: This attribute is to reach the high score class for update the score.

private InputManager InputMngr: This attribute is to reach input manager class.

private SoundManager SoundMngr: This attribute is to reach the sound class to get the music of the game.

private SettingManager SettingMngr: This attribute is to reach the setting class for reach the settings functions.

private GameEngine engin: This attribute is to reach the game engine class to get the game body.

b) **Methods**

public void startLevel(int level): This is get the start level of the game method.

public void endGame(): This method end the game that played.

public void updateHighScore(int score): This is a method that updates the high score if there is new high score for the game.

public void writeHighScore(int score): This method activates if there is new highscore in the game and change the value.

public void pauseGame(): This method allow the user for pause the game.

public void movePlayer(): This method allows the user for move the player's tank

public void moveEnemyTank(): This method moves the enemy tanks and these tanks are try to kill the player tank.

public void shootPlayer(): The enemy tanks shoot the player's tank with this method functionality.

public Boolean isLiveZero(): This method checks the player lives zero or not. If it is zero then the game is end.

public void updateView(): This method is update the view of the panel.

public void pauseGame(): This function is used for pause the game.

public void resumeGame(): This function is used for resume the game while the game is paused.

2) HighScoreManager Class:

This class allows the game to register new high scores and show the high score in a panel.

a) Attributes:

private ArrayList<int> scoreList: This method collect the high score data inside the arraylist.

b) Methods:

public void registerHighScore(int score): If there is a new high score it save the data inside the text file.

public void showHighScore(): This method allow the game to show the high scores.

3) **InputManager Class:**

   This class is allow the game to show some inputs like error messages.

4) **SoundManager:**

   This class is a sound manager that allow the user to open and close the sound of the game.

a) **Methods:**

   public void playMusic(): This method allow game to open the music while game is played.

5) **SettingsManager:**

   This class allow the user for changing settings functionality which are change level and change settings.

a) **Methods:**

   public void changeLevel(int level): The method is change the level of the game.

   public void changeVolume(int volume): The method allow the user for increase or decrease the volume of the game.

6) **GameEngine Class:**

a) **Attributes:**

   private GameBody[][] map: This is a variable that allow the game to create a map from the game body class.

   private CollisionManager colMngr: This variable is allow to use the collision manager class methods to check the collision between the objects.

Private Place[] objPlace: This variable is allow to place the objects for the map of the game.

b) **Methods:**

public void createGameBody(GameBody body, Place place): This method create a new game body for a new game or a new level.

public void destroyGameBody(GameBody body): This method destroy the game body for create a new game body.

public GameBody[][] getMap(): This method allows the game to get the desired level map from the GameBody class.

public void updateMap(): This method is allow the game to update the old map with a new desire level map.

4.4.2. Model Classes

Model packet involves all of the information about classes that act as entity objects. The package contains GameEngine and GameBody whild hold for the Model mechanism and Tank, Wall, SacredObject and PowerUps which work as entity objects.

1. **GameBody Class:**

   This is the class which will represent the map of the game where user will be moving around his/her tank.

a) **Attributes:**

private Place location: This attribute holds the location of the object as a Place instance.

private image: This is the image of the object which will be used on GUI.

b) **Methods:**

public Place getLocation(): This method returns the location of an object.

public void setLocation(int x, int y): This is the method for updating an object's location.

2. **Place Class:**

   This is an abstract class for collecting the coordinates of the objects together.

a) **Attributes:**

   private int x: This holds the horizontal coordinate value of the object.

   private int y: This holds the vertical coordinate value of the object.

b) **Methods:**

   public void setX(int x): This method will be used to modify the horizontal coordinate of the object.

   public void setY(int y): This method will be used to modify the vertical coordinate of the object.

   public int getX(): This method returns the horizontal coordinate of the object.

   public int getY(): This method returns the vertical coordinate of the object.

3. **Wall Class:**

   This is the class which represents the wall objects -the obstacles- in our map.

4. **BrickWall Class:**

   This class is the representation of the brick wall objects and extends from Wall class.

a) **Attributes:**

   private int givenScore: This attribute holds the score that the player can get by hitting this object.

b) **Methods:**

public void getIndestructable(): This method makes the brick wall object indestructable.

5. **IronWall Class:**

This class is the representation of the iron wall objects and extends from Wall class.

a) **Attributes:**

private int givenScore: This attribute holds the score that the player can gat by hitting this object.

6. **SteelWall Class:**

This class is the representation of the steer wall objects and extends from Wall class.

7. **Tank Class:**

This class represents the tank objects in our game.

a) **Attributes:**

private int shootSpeed: This attribute holds the speed of a shot of an enemy tank.

private int moveSpeed: This attribute holds the speed of an enemy tank.

b) **Methods:**

public void shoot(): This method makes a tank shoot a bullet.

public void move(int x, int y): This method moves an enemy tank x units horizontally and y units vertically.

8. **PlayerTank Class:**

This class represents the tank of the user in our game and extends from Tank class.

a) **Attributes:**

public int currentLives: This attribute will hold the remaining lives of the player's tank as an integer.

b) **Methods:**

public void getPowerUp(): This method enables the player tank to activate a power up.

public void increaseLife(): This method increases the player's lives by 1.

public void decreaseLife(): This method decreases the player's lives by 1.

9. **EnemyTank Class:**

This class represents the tanks which are the enemies of the player in our game and extends from Tank class.

a) **Attributes:**

private int hitsToKill: This attribute holds the number of hits an enemy tank must collide with, in order to be killed.

private int givenScore: This attribute holds the score that the player can get by hitting this object.

10. **Basic Class:**

This class is the representation the most basic enemy tank in the game and extends from EnemyTank class.

11. **Crazed Class:**

This class is the representation of an enemy tank which can move and shoot crazily after being hit 1 time and extends from EnemyTank class.

a) **Methods:**

public void goCrazy(): This method will be used to make this tank object get crazy and double its moving and shooting speed.

12. Panzer Class:

This class is the representation of the most durable tank of the game and extends from EnemyTank class.

13. SacredObject Class:

This class represents the sacred object that the player tries to protect.

14. PowerUps Class:

This class represents the power up objects.

a) Attributes:

private int type: This attribute will hold an integer value which will be interpreted as the power up's type. It will be 1 for shields, 2 for double shots, 3 for extra life and 4 for ultimate protection.

b) Methods:

public void getPicked(): This method enables a power up to be taken.
public void appear(): This method makes a power up appear in the map.
public void expire(): This method makes a power up disappear if it is not taken after a while.

15. Shield Class:

This class is the representation of shield type power up and extends from PowerUp class.

16. DoubleShots Class:

This class is the representation of double shots power up and extends from PowerUp class.

17. ExtraLife Class:

This class is the representation of extra life power up and extends from PowerUp class.

18. **UltimateProtection Class:**

    This class is the representation of ultimate protection power up and extends from PowerUp class.

19. **Bullet Class:**

    This class represents the bullet objects in our game.

a) **Attributes:**

    private int moveSpeed: This attribute holds the value of the speed of a bullet object.

b) **Methods:**

    public void move(): This method makes the bullet move in the map.

20. **CollisionManager:**

    This class allow the game to check that the player bullet is intersect with the enemy tank or a wall to destroy them.

a) **Methods:**

    public boolean checkCollision(GameBody1,GameBody2): The method return a Boolean that check the collision of to game object.

4.3.3. View Classes

View class is an another packet of the game that contain all the user interface information and algorithms of the game. The class allow the game to draw the map, update view and the important game subsections panels like Setting, High score etc.

1) **MainFrame**

    This class is acting like controller of the panels that are create the view of the system which are HelpPanel, GamePanel, HighScorePanel, SettingsPanel and MainMenuPanle.

a) **Attributes:**

private JPanel activityPanel : This variable is used the JPanel object from the javax.swing class.

private ArrayList<JPanel> panels: This variable is for collecting all of the panels for put the screen.

b) **Methods:**

public void updateView(): The method allow to change the existing view with a different for example from main menu to settings is update by this method.

2) **GamePanel**

a) **Methods:**

public void draw(int[][] map): This method is draw the map of the game. The method draw the 5 of the level with the help of this function.

3) **HelpPanel:** In this class there are some algorithms for the Help menu.

4) **SettingsPanel:** In this class there are some algorithms for the Settings menu.

5) **HighScorePanel:** In this class there are some algorithms for the High score menu.