  
  
  
CS 319 - Object-Oriented Software Engineering  
Design Report  
  
Tanks Game  
Yryskul Turduev

Hamza Saeed Khan  
Soleh Ruziboev  
Seyedehshahrzad Seyfafjehi

Contents

[1 Introduction 3](#_Toc488886627)

[1.1 Purpose of the System 3](#_Toc488886628)

[1.2 Design Goals 3](#_Toc488886629)

[1.2.1 Reliability 3](#_Toc488886630)

[1.2.2 Modifiability 3](#_Toc488886631)

[1.2.3 Readability 3](#_Toc488886632)

[1.2.4 Portability 3](#_Toc488886633)

[1.2.5 Response Time 4](#_Toc488886634)

[1.2.6 Ease of Use 4](#_Toc488886635)

[2 Software Architecture 5](#_Toc488886636)

[2.1 Subsystem Decomposition 6](#_Toc488886637)

[2.1.1 User Interface 6](#_Toc488886638)

[2.1.2 Game Play Controller 6](#_Toc488886639)

[2.1.3 Game Entities 7](#_Toc488886640)

[2.2 Persistent Data Management 7](#_Toc488886641)

[2.3 Hardware / Software Mapping 7](#_Toc488886642)

[2.4 Access Control and Security 8](#_Toc488886643)

[2.5 Boundary Conditions 8](#_Toc488886644)

[2.5.1 Initialization 8](#_Toc488886645)

[2.5.2 Termination 8](#_Toc488886646)

[2.5.3 Error 8](#_Toc488886647)

[3 Subsystem Services 9](#_Toc488886648)

[3.1 Design Pattern 9](#_Toc488886649)

[Facade Design Pattern: 9](#_Toc488886650)

[3.2 User Interface Subsystem Interface 9](#_Toc488886651)

[Start Menu Class 10](#_Toc488886652)

[Settings Class 11](#_Toc488886653)

[View 11](#_Toc488886654)

[3.3 Gameplay controller Subsystem 12](#_Toc488886655)

[GamePlayController 12](#_Toc488886656)

[MapController 13](#_Toc488886657)

[CollisionController 14](#_Toc488886658)

[3.4 Game Entities Subsystem Interface 15](#_Toc488886659)

[GamePlay Class 16](#_Toc488886660)

[AbstractTank 17](#_Toc488886661)

[Tank1 18](#_Toc488886662)

[Tank2 19](#_Toc488886663)

[Shell 19](#_Toc488886664)

[Abstract Mine Class 20](#_Toc488886665)

[Tank Mine Class 21](#_Toc488886666)

[Game Play Mine Class 21](#_Toc488886667)

[Abstract Obstacles Class 21](#_Toc488886668)

[Unbreakable Obstacle Class 22](#_Toc488886669)

[Breakable Obstacle Class 22](#_Toc488886670)

[Transparent Obstacle Class 23](#_Toc488886671)

[Health (power-up) 23](#_Toc488886672)

[Lifebar 24](#_Toc488886673)

# 1 Introduction

## 1.1 Purpose of the System

Tank Wars is a multiplayer action game, that can be played on Windows, macOS, and Linux devices. It has been modelled on the Arcade games of the 70s and 80s. It provides a nostalgic experience to adult gamers while giving young gamers a glimpse of the Arcade culture, right on their computers.

The goal of the game is to eliminate your opponent. Each player is provided with a tank that has shells that can be fired at the opponent’s tank. The player also has to deal with secondary challenges such as obstacles and mines.

## 1.2 Design Goals

### 1.2.1 Reliability

Java’s built in protective features ensure that the program will not harm the device. The encapsulation of pointers in Java makes sure that there are no memory leaks. Furthermore, to minimize the program’s memory imprint we have tried to make the memory usage almost constant. The memory usage does not increase with the passage of time. These safeguards stop memory overflow and any resulting crash.

### 1.2.2 Modifiability

The use of MVC and extensive decoupling in the sub systems makes it easy for any developer to modify or extend the program. A change in one subsystem would not demand a change in another. New features can be added and easily integrated with the existing ones. This is possible because each subsystem and each class within the subsystems are coherent, well defined, independent, and with a distinct purpose. So, any new feature will have its own place in the system and will not need large amounts of changes to existing classes.

### 1.2.3 Readability

The code will be well documented and commented. Well will follow the standard coding style so that it easier to read. This will make it easy for us to maintain the code and to for anyone to modify it.

### 1.2.4 Portability

The program can be run without any change to code on different machines with different OS because we will implement it in Java. Java Virtual Machine (JVM) removes the need of modifying the code for different environments.

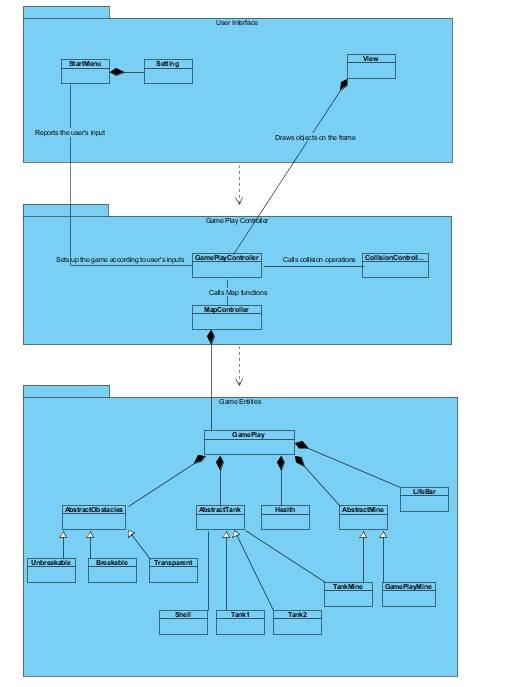
### 1.2.5 Response Time

The game is low on graphics and is in 2D. It does not need a dedicated GPU. Any average processor will be enough to run the game at a reasonable speed. At any moment, the maximum number of inputs is four, two from each player. Therefore, the response time will be almost instantaneous.

### 1.2.6 Ease of Use

The learning curve for the game is not steep. A couple of minutes playing the game is enough to understand everything about the game. The only difficult part is to get comfortable with the input buttons. However, the random nature of obstacles and mines make it challenging every time it is played.

# 2 Software Architecture

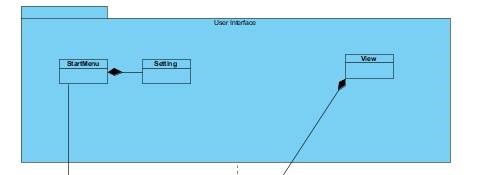
We have decomposed the system into several subsystems. Each subsystem has classes that perform similar tasks. This ensures that each subsystem has coherence and a well-defined function. The architecture that we chose is a three-layered architectural style, that follows the paradigms of MVC (Model-view-controller). The use of this style will ensure that the code is easier to understand, has no redundancy and is easier to maintain.

## 2.1 Subsystem Decomposition

The three layers are the Game Entities, User Interface, and Gameplay Controller that correspond to Model, View, and Controller respectively. The limited coupling between the subsystems ensures that changes in one system do not necessitate a change in other subsystems. For instance, User Interface can be changed without the need of changing anything in Game Entities.

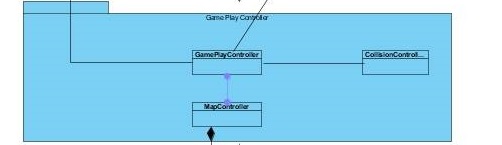
### 2.1.1 User Interface

User Interface is the top most layer of the system. It is the shell that the player interacts with in the main menu, where it relays the user’s inputs to the controller. The diagram shows this with an association between the Menu and the GamePlayController. The User Interface is also responsible for the GUI component of the game. Screen renderings and changes to the View are done by the User Interface according to the directions of Game Play Controller.



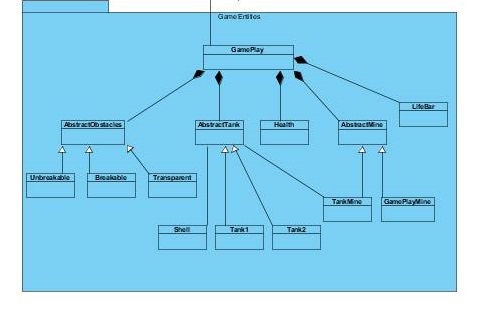
### 2.1.2 Game Play Controller

The Game Play Controller is responsible for the Game Logic and the management of the game data. It is part of the system that decides what happens when an event occurs, such as, when a player gives an input, or a collision occurs, or when the game ends. The controller is further divides into three sub controllers. The GamePlayController, the CollisionController and the MapController. The MapController decides on the relative positions of the game objects. The CollisionController decides what happens if a collision occurs. Finally, the GamePlayController, controls the movement of the objects, score keeping, game instantiation and termination, and the main menu.



### 2.1.3 Game Entities

Game Entities has all the classed that describe the objects of the game. These include the Tank, Health, Obstacle, LifeBar, TankMine, Shell, GamePlayMine and the GamePlay. Game Entities determines the attributes and functions each object has.



## 2.2 Persistent Data Management

The game does not need a complex database because all the data used by the game will be accessed in real time. The game cannot be saved and restarted from the same moment. No high scores are kept and the settings default to their original values once the program is shut down.

## 2.3 Hardware / Software Mapping

Java will be used to implement the program. A device will need to have JRE (Java Runtime Environment) to run the program. A keyboard will be needed for the inputs to the game. The game is multiplayer but will run on a single device. The players will share the keyboard. Windows, macOS, Linux coupled with JRE will be able to run the program, so the software requirements are minimum.

## 2.4 Access Control and Security

The game does not require a connection to the internet. It is solely offline. There is no user authentication at the start up. As there is login or a need of making a profile, there is no threat of breach of data.

## 2.5 Boundary Conditions

### 2.5.1 Initialization

The user will have to click on an executable .jar file to start the program. JRE will be needed on the target device for the game to run.

### 2.5.2 Termination

The program can be terminated through the main menu exit button or the X on the top right corner.

### 2.5.3 Error

The program will show an error message of the image or the audio files are corrupted.

# 3 Subsystem Services

In this section the classes in each subsystem is provided.

## 3.1 Design Pattern

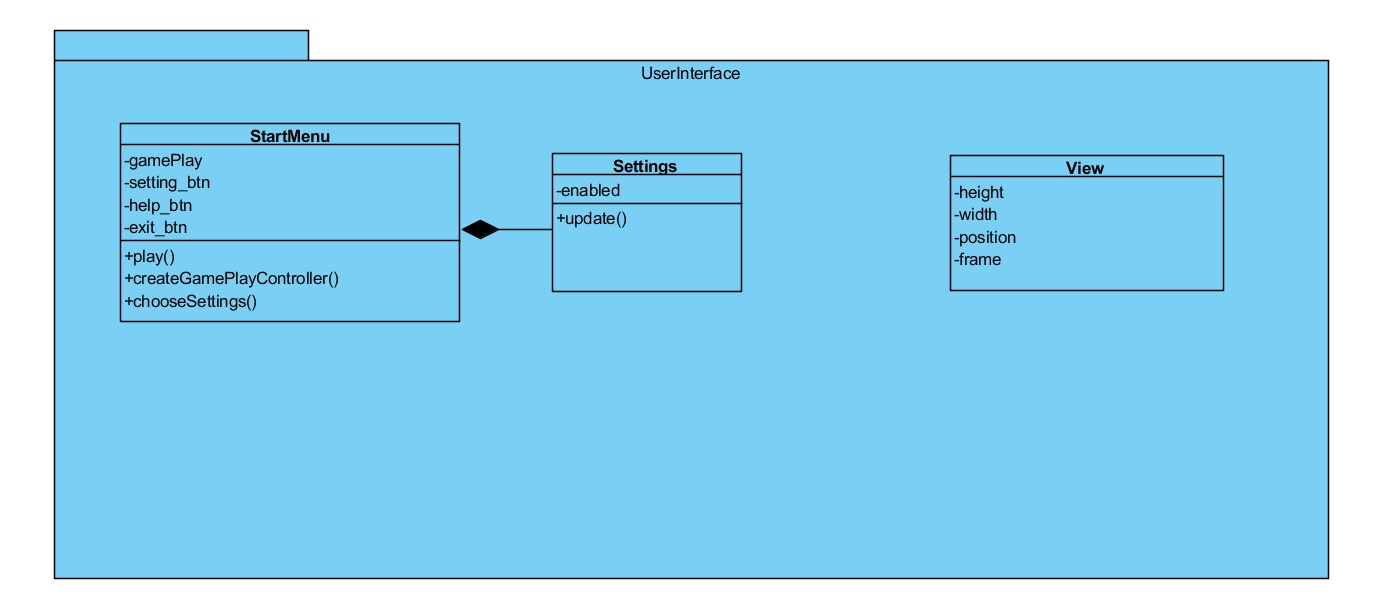
### Facade Design Pattern:

Facade design pattern is a structural pattern that provides a unified interface to a set of interfaces in a package. Facade creates a higher level interface which makes using the subsystem easier.

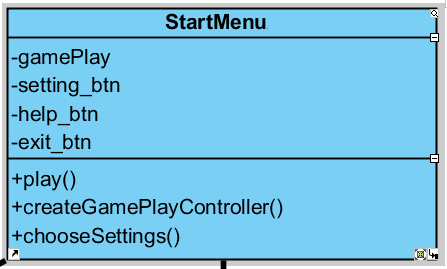
In this project facade design pattern has been used in order to easily link different subsystems of the program together. In the Gameplay controller subsystem, the GamePlayController class is a facade class for this package and the StartMenu class communicates with this class in order to set up the game according to user’s inputs. Also, in Game entities package GamePlay class is the facade class for this package as it provides a general interface for all the entity objects in the game. Thus, MapController which creates the objects communicates with GamePlay class.

## 3.2 User Interface Subsystem Interface

User Interface subsystem includes all classes related to graphical components. Also, the class providing frames for different panels is a part of this package in order to make transitions between these panels easier.



### Start Menu Class



* This class basically displays the Menu for the game. It creates Game Play Controller and Setting objects. And it also has setting, help and exit buttons which are displayed in JPanel.

#### Attributes:

* **private GamePlay gamePlay:** This is a class of GamePlay object. It is the main attribute which draws the all objects in the game display.
* **private Setting setting\_btn:** This is a class of Setting object. It displays the setting option in JPanel.
* **private Jbutton help\_btn:** This is a button displays the help options to the player in the JPanel.
* **private Jbutton exit\_btn:** This is a button exits from the Game.

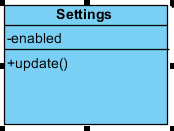
#### Constructors:

* **public StartMenu:** Initializes  *help\_btn, setting\_btn, help\_btn, and exit\_btn*.

#### Methods:

* **private void play():** This method starts the game by creating MapController class.
* **private void createGamePlayContoller():** This method creates the GamePlayController class which has all controller objects for the game.
* **private void chooseSettings():** This method creates the Setting class and displays its option in the JPanel.

### Settings Class



* Settings have “update” method which updates the music’s and sound’s condition by setting it to ON or OFF.

#### Attributes:

* **public boolean enabled:** This boolean check whether the music

#### Constructors:

* **public Settings:** Initializes  *the boolean to the false when it created firstly.*

#### Methods:

* **private void update():** This method updates the boolean enabled.

### View



It is a library for the graphics. It handles the animation of tanks, shells and mines. It has getGraphics method. Additionally, it holds the frame for the gameplay.

#### Constructor:

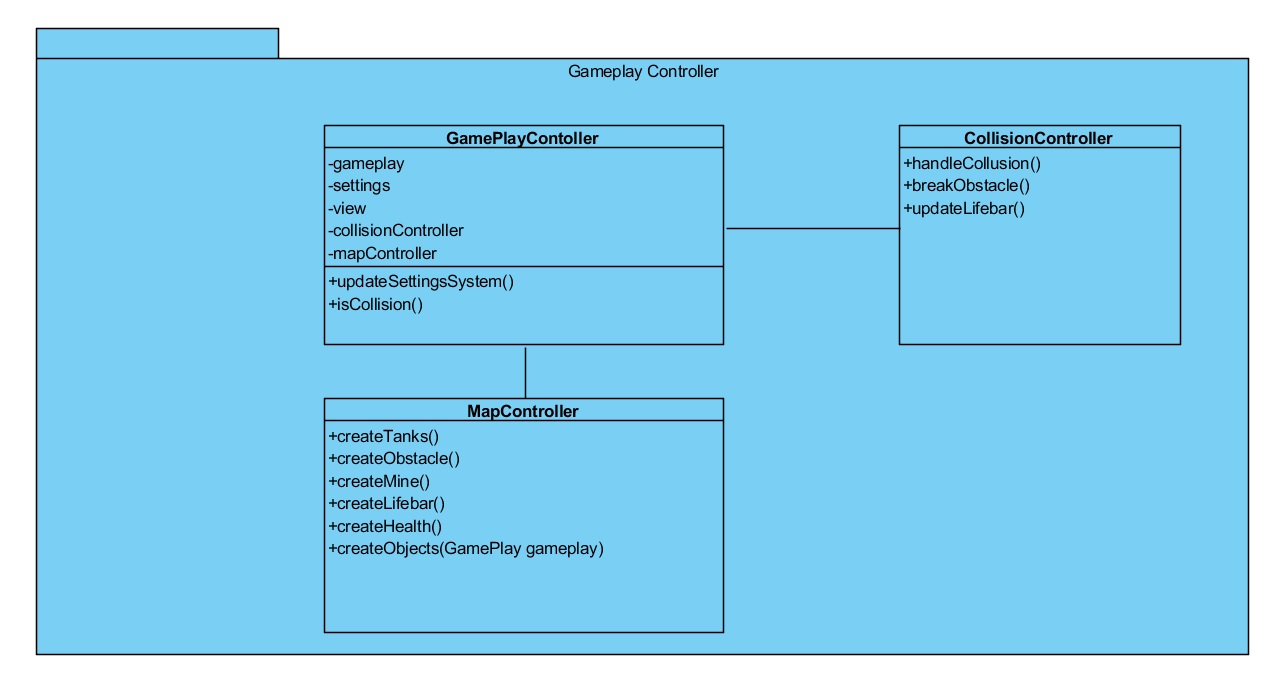
* **public View() –** creates an object and initializes the position with some point

#### Atrributes:

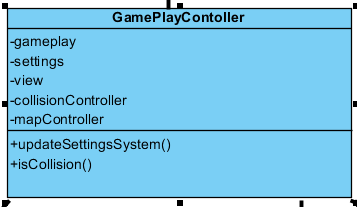
* **private const int height –** holds the height of the frame
* **private const int width –** holds the width of the frame
* **private int position –** holds the current position of the frame
* **private JFrame frame –** holds the frame of the view class

## 3.3 Gameplay controller Subsystem

All control classes are grouped in this subsystem in order to facilitate controlling the dynamic of the game. The details of each of these classes and their functionality and attributes are explained in this section.



### GamePlayController

****

This class is one of the main classes. It creates all control objects and GamePlay playground object. It creates all the objects of the game by calling relevant methods which in return calls others’ controllers methods which gets the job done.

#### Constructor:

* **public void GamePlayController() –** initializes the attributes(calles their constructors) and calls mapController’s createObjects() method

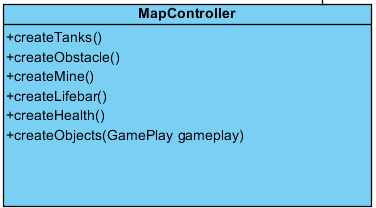
#### Atrributes:

* **private GamePlay gameplay –** the instance of GamePlay class in the GamePlayController class holds all the objects of the game and is responsible for any update occurring in the game
* **private Settings settings –** the main purpose of this instance is to let know the GamePlayController whether to turn the sound on or off
* **private View view –** View instance is needed to identify the frame, panel and other graphical nuances
* **private CollisionController collisionController –** whenever there is a collision between two relevant objects the collision controller is needed to handle the collision
* **private MapController mapController –** the instance of MapController is needed to create all the initial objects of the game

#### Methods:

* **public void updateSettingsSystem() –** the method updates the sound on/off option whenever the user decides to turn on or off the sound
* **public boolean isCollision() –** the method detects whether there is a collision between two relevant objects in the game

## MapController



This class controls the creation of the objects on the game’s map once the user starts the game and GamePlayController calls creatObject method. Later this class adds the created objects to the GamePlay class which draws all these objects on the map of the game.

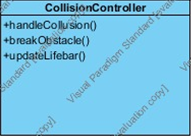
#### Constructor:

* **public MapController() –** empty

#### Methods:

* **public void createObjects (GamePlay gamePlay)** - It takes a gamePlay object as a parameter and creates all the objects in the game which includes tank1, tank2, mine, obstacles, lifeBar1, lifeBar2 and health by calling each object’s create function. Then it adds all the created objects to the passed gamePlay instance by calling addObject method on it.
* **public AbstractTank createTank()** - it creates a tank object and returns it.
* **public AbstractObstacles createObstacles()** - it creates an obstacle object and returns it.
* **public AbstractMine createMine()** - it creates a mine object and returns it.
* **public LifeBar createLifeBar()** - it creates a lifeBar object and returns it.
* **public Health createHealth()** - it creates a health object and returns it.

### CollisionController

****

The CollisionController class is used to control any type or any combination of object which can occur during the game in the map. There might be different type of collisions such as bullet to obstacle, bullet to tank, bullet to mine, mine to obstacle, mine to mine, mine to tank, bullet to power-up or bullet to borders of the map.

#### Constructor:

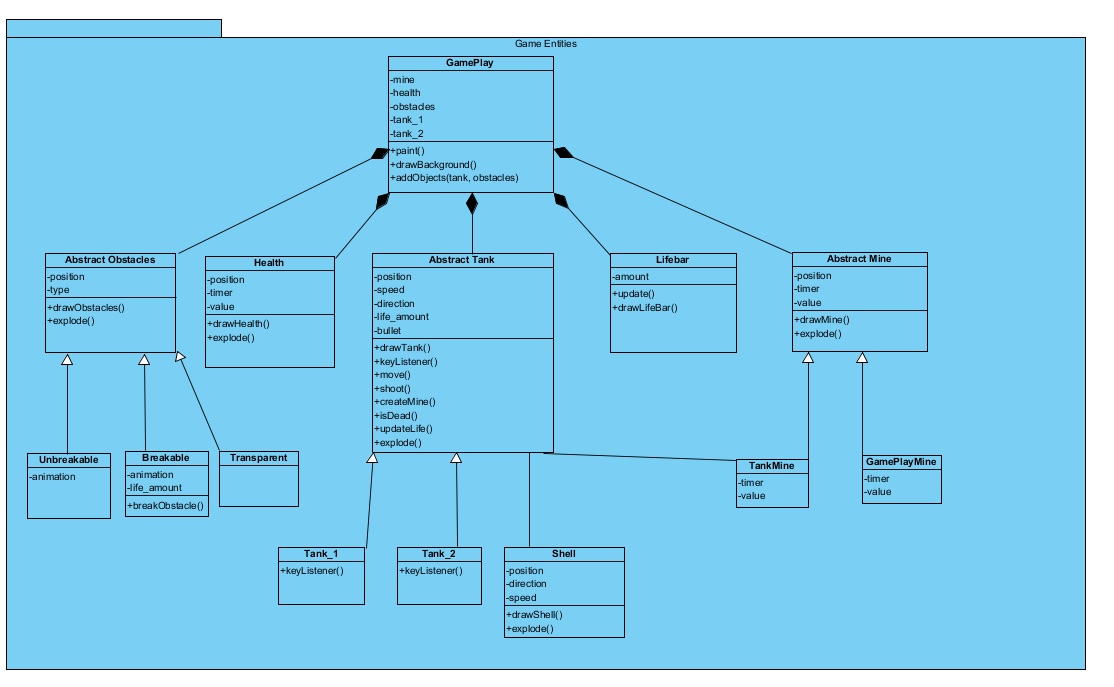
* **public void CollisionController() -** empty

#### Methods:

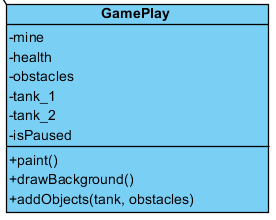
* **public void handleCollision()** – if there are any two objects located in the same position of the map, then this method first of all detects what kind of collusion it is and after calls the needed method
* **public void breakObstacle() –** whenever there is a collision between bullet and obstacle or mine to obstacle the method will be called to break the obstacle
* **public void updateLifebar() –** whenever there is a collision between a shell and tank or a mine and tank the method will be called to update the life of a tank and also update a lifebar

## 3.4 Game Entities Subsystem Interface

All the domain specific objects are held in this package. The facade class of this package which is GamePlay class provides a unified interface for all the entities in the game which makes Gameplay controller package loosely coupled with this class. Therefore, adding new entities to this subsystem will not compromise the performance of other packages and changes could be made easily.



### GamePlay Class

******

This class inherits paint method from JPanel and draws all objects in the display.

* **Public Mine mine:** GamePlay draws the mines of Tank and automatic mines.
* **public Health health:** GamePlay draws the health conditions of the tanks.
* **public Obstacles obstacles:** Gameplay draws the three types of obstacles.
* **public Tank tank\_1:** GamePlay draws the tank of first Player.
* **public Tank tank\_1:** GamePlay draws the tank of second Player.,
* **public boolean isPaused:** During the game, if user pauses the game, Game Play sets the isPaused to true.

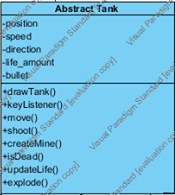
#### Constructors:

* **public GamePlay:** Initializes the attributes and automatically calls its paint method.

#### Methods:

* **public void paint():** This method paint all objects in the GamePlay.
* **public void drawBackground():** This method draws background picture or color for GamePlay.
* **public void addObjects(mine, obstacles, tank, health):** This method adds the objects into the Game Play and initiate paint method to display objects on the screen.

### AbstractTank

****

It is an abstract form of a Tank which has their mutual attributes and operations. It is abstract because the first and the second tanks have different implementation for the same method, keyListener(). Tanks are controlled by players and their amount of life is updated when they get hit by a shell or when they pick up a power-up.

#### Constructer:

* **public void AbstractTank() –** initializes the attributes

#### Attributes:

* **private point position -** an attribute is used to identify a specific position of the particular tank in the map
* **const private int speed –** needed to identify the speed of a tank
* **private int direction –** the attribute holds the current direction of a specific tank, {up = 1, right = 2, down = 3, left = 4}
* **private int lifeAmount –** the attribute holds the value of the remaining amount of health which needs to be reduced by other player in order to win
* **private Shell bullet –** this object is needed when one or both of the players are shooting either at each other or other objects currently present in the map

#### Methods:

* **public void drawTank() –** the method draws the tank (picture) in the map using its current direction and position
* **public abstract void keyListener() –** whenever any key is pressed the method checks whether the pressed keys are the ones needed to move or to apply any other action to the tank an if so it calls other methods such as move(), shoot() or createMine()
* **public boolean move(int direction) –** gets the direction and moves (updates the position) according to pressed keys
* **public void shoot() –** according to the current direction, when the relevant key is pressed the tank releases the shell in the same direction
* **public boolean createMine() –** when the appropriate key is pressed the tank sets a mine in the map in the current position of a tank, however it first checks whether the place in the map is suitable
* **public boolean isDead() –** returns whether the tank is alive in other words whether the amount of life of a tank is more than zero
* **public void updateLife() –** updates the lifeAmount attribute of the tank whenever the tank is shot with a shell, hurt by a mine or when the tank picked the power-up

### Tank1

****

This tank(class) is controlled by the first player by using keys w, a, s, d, space, m.

#### Constructor:

* **public void Tank1() –** uses super constructor

#### Attributes:

All attributes are inherited from a parent class AbstractTank

#### Methods:

All methods from a parent class AbstractTank +

* **public void keyListener() –** once the relevant keys are pressed the tank does one out of three actions, calls the move() method, (moves up for w, left for a, down for s, right for d), calls setMine() method (sets a mine for m) or calls the shoot() method(shots a shell for space)

### Tank2

****

This tank(class) is controlled by the second player by using keys left arrow, up arrow, right arrow, down arrow, enter, 0.

#### Constructor:

* **public void Tank2() –** uses super constructor

#### Attributes:

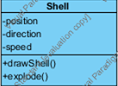
All attributes are inherited from a parent class AbstractTank

#### Methods:

All methods from a parent class AbstractTank +

* **public void keyListener() –** once the relevant keys are pressed the tank does one out of three actions, calls the move() method, (moves up for up arrow, left for left arrow, down for down arrow, right for right arrow), calls setMine() method (sets a mine for 0) or calls the shoot() method(shots a shell for enter)

### Shell

****

An Object created by a Tank once the relevant keys are pressed. When the shell reaches the map limits it disappears, when it encounters the transparent obstacle, it goes through it, for any other object it either destroys it or reduces the amount of life.

#### Constructor:

* **public void Shell() –** initializes position to center of the screen and direction to **0**

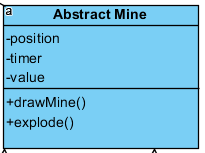
#### Atrributes:

* **private int position –** tracks the current position of a bullet, starting from the current position of a tank that it has been shot from and untill it reaches some other object in the map
* **private int direction –** the direction of the bullet is needed to identify in which way the bullet should go, and it’s obtained from the direction of the tank which shot the shell
* **private const int speed –** this attribute is needed for the speed of tank (say pixels per second) which is a constant value

#### Methods:

* **public void drawShell() –** this method draws a shell (a picture) in the map from the beginning (when the tank released it) till the end (when it encounters some other obstacle)

### Abstract Mine Class

******

This class is a parent class for the TankMine and GamePlayMine. It has main methods which children classes should implement.

#### Attributes:

* **public int position:** It has position to be placed.
* **public int timer:** This is timer for exploding the mine.
* **public int value:** The power of the mine.

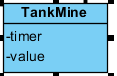
#### Constructors:

* **public abstract Mine:** Initializes the attributes.

#### Methods:

* **public void drawMine():** This method draws the mines in the GamePlay.
* **public void explode():** This method initializes the explosion of mine.

### Tank Mine Class



#### This class is initiated by tanks. Tanks can drop its mines.

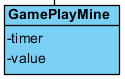
#### Attributes:

* **public int timer:** This is timer for exploding the mine.
* **public int value:** The power of the mine.

#### Constructors:

* **public TankMine:** Initializes  *the attributes.*

### Game Play Mine Class

******

#### This class is initiated by GamePlay class automatically. There will be randomly generated mines which players can take.

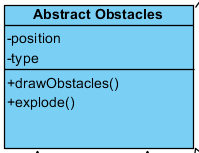
#### Attributes:

* **public int timer:** This is timer for exploding the mine.
* **public int value:** The power of the mine.

#### Constructors:

* **public GamePlayMine:** Initializes  *the attributes.*

### Abstract Obstacles Class

**

This class functions as an obstacle. It has three types as children classes according to their breakability.

#### Attributes:

* **public int position:** It has position to be placed.
* **public int type:** It has type of breakability and not breakability.

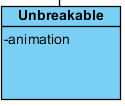
#### Constructors:

* **public abstract Obstacles:** Initializes the attributes.

#### Methods:

* **public void drawObstacles():** This method draws the obstacles in the GamePlay.
* **public void explode():** This method initializes the explosion of obstacles according to their type.

### Unbreakable Obstacle Class

**

This class functions as an obstacle. It is unbreakable.

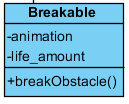
#### Attributes:

* **public int animation:** It makes an animation to show its type.

#### Constructors:

* **public Unbreakable Obstacles:** Initializes the attributes.

### Breakable Obstacle Class

**

This class functions as an obstacle. It is breakable.

#### Attributes:

* **private int animation:** It makes an animation to show its type.
* **private int life\_amount:** It is life amount of breakable obstacle. It is broken after 3 shots.

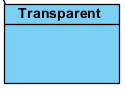
#### Constructors:

* **public Breakable Obstacles:** Initializesthe attributes.

#### Methods:

* **public void breakObstacles():** This method breaks the obstacle by removing from the obstacle arraylist in the GamePlay.

### Transparent Obstacle Class

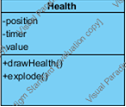
**

This class functions as an obstacle. It is transparent.

#### Constructors:

* **public Transparent Obstacles:** Initializes the attributes.

### Health (power-up)

****

It is an object that GamePlay creates in random periods of time in the game. Tanks controlled by players can pick the power-ups (health objects in the form of a waffle) in order to increase their amount of life.

#### Constructor:

* **public void Health() –** initializes the attributes

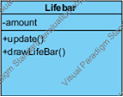
#### Attributes:

* **private int position –** the attribute holds the current position of the power-up in the map
* **private Timer timer –** when the power-up just appears in the map, it’s not going to stay there forever or until some tank picks it up. Therefore, we this attribute in order to track the time it has been in the map and if it exceeds some threshold the it automatically explodes
* **privet int value –** the attribute holds the value of the power-up, when the tank picks up the power-up, its life will be updated (value will be added to lifeAmount)

#### Methods:

* **public void drawHealth() –** this method draws a health or a power-up (a picture) in the map whenever the GamePlayController decides to

### Lifebar

****

This class shows the health of each tank, and it is automatically updated when tank loses his health or picks up a power-up.

#### Constructor:

* **public void Lifebar() –** initializes amount will full life amount

#### Attributes:

* **private int amount –** the attribute holds the value of the remaining amount of life of the particular tank

#### Methods:

* **public void update() –** each time the tank is shot by another tank, hurt by a mine or the tank picked the power-up this method is called, it simply updates the amount of life left of the tank