Bilkent University

Department of Computer Engineering

CS 319 - Object Oriented Software Engineering Project

Bullet Drop

Design Report

Group 2F

Ulaş İş, Alperen Erkek, Ömer Faruk Ergün, Abdullah Seçkin Özdil

Course Instructor : Bora Güngören

Table of Contents

1.Introduction

1.1 Purpose of the system

Bullet Drop is a 2D shooting game. The main difference of Bullet Drop from the other arcade shooting games is that the target is invisible in the beginning. Therefore, the user doesn't know where to shoot on the first attempt, so they need to track the route the bullet follows and observe the outside effects to be successful on the next attempts. This way, we aimed to make the game more challenging and enjoyable for the player. Moreover, Bullet Drop will be graphically appealing to the player. We will have realistic pictures of weapons, bullets and maps that will be created through various programs and integrated into the game so that the game will include realistic models of weapons, bullets and maps(background) with different effects.

1.2 Design Goals

1.2.1 Adaptability

The game will be implemented in Java because one of our most important goals is that the game should work in all platforms and since Java provides cross-platform portability, it will allow the game to work in all JRE installed platforms. For that reason, we preferred Java to develop the game over other alternatives.

1.2.2 Efficiency

It is really important that the game runs fluently and the movement of the objects should be very smooth for the satisfaction and entertainment of the player. Response time is also very crucial. When player adjusts the position of the weapon or he/she fires the bullet, there shouldn't be any delay or lag. This would affect both gameplay and the enjoyment of the player. The requests of the user should be responded instantly. Since the game will have high-quality graphics and animations, it is really important that it doesn't affect the performance and smoothness of the game.

Therefore, we want to minimize the input lag as much as possible and the game should run smoothly.

1.2.3 Reliability

One of the most common reasons for the bugs in games are invalid inputs. Therefore, we need to create the system such that it will ignore any irrelevant or unexpected input so that the game will not give any errors during run-time and it won't crash. To achieve this, the tests will be done throughout the implementation of the game. We will try to consider each situation that can happen in order not to miss anything that can cause crashes in the game. Briefly, the game will be bug-free and reliable for the user.

1.2.4 Usability

It is really crucial for us that our game will be user-friendly. To achieve this, we will have a very simple yet effective user-interface. The player will easily learn how to play the game and not get lost in a complex user-interface. There won't be much options on the setting screen and one of the options will be how to play which will teach the player about the game, which is really easy to play anyway. Therefore, the player won't have difficult time trying to figure out how to play the game and how things work. The only controllers in the game are arrow keys and space key in the keyboard. Again, these features will be explained to the player in the game. As one of our most important goals is to make the game entertaining for the player, the user-interface should be simple enough, learning how to play should be easy, but this doesn't mean that the gameplay will be easy either. Figuring out how the game works will be easy, however, the game itself should be difficult enough to keep the player in the game and prevent them from getting bored. Of course, the game shouldn't be very difficult either, that would also decrease the enjoyment of the player.

1.2.5 Extensibility

We desire our game to be open ended, meaning it should be open to new features and can be developed further since the system is object-oriented. There can

be more weapons, weapons or outside forces in the game, or the current versions of these features can be modified easily. By adding new features, we aim to keep the player interested in the game. Moreover, one important goal of the project is that adding new features or modifying them should be simple and easy, so there won't be any problems and conflicts.

1.2.6 Tradeoffs

1.2.6.1 Efficiency - Reusability

One of our most important design goals is efficiency and in order to achieve this, we need to sacrifice reusability. Since none of our classes in our project will be used for different projects, we will design them so that they will only be functional for our game, which will help us write more simple codes which will improve efficiency of the system.

1.2.6.2 Usability - Functionality

In our system, it is really important to make the game easy to use for the player. The player should be able to play it with a simple and easy user-interface. Therefore, to achieve this goal, we prioritized usability over functionality. In other words, our system won't have complex user-interface or complex features that may be difficult for the user to learn and use. We want our system to be simple and easy to use but also effective. Therefore, we won't sacrifice any basics of the game, but remove some details and insignificant features for our game. For instance, we won't have any difficulty level for the game. We also won't have too many options and settings in the game that can confuse the players.

1.2.6.3 Performance - Memory

As discussed before, performance and efficiency is one of the most important design goal for our project. We want the movements of the objects be really smooth so performance is a really important goal for our system. To get and maintain high performance, the memory space required may increase. This will also increase the speed of our game and prevent and input lags or other problems related to efficiency and performance. Increasing memory will also reduce the workload of "Game Logic"

class. Hence, we will sacrifice memory in order to achieve our performance goal. However, it will increase the speed of the game and the game will run more smoothly, and we will provide the player with a better game experience.

1.3 Definitons, acronyms and abbreviations

- JRE : Java Runtime Environment. It's a software package that contains what is required to run a Java program.[1] Therefore, JRE needs to be installed on your computer in order to run Java applications.[2]

- FPS : Frames Per Second. FPS is the frequency at which frames are displayed in an animated display. [3]

- Game Logic : The core class for our system. It will use different components and will work pretty much like an engine.

1.4 References

[1] https://en.wikipedia.org/wiki/Java_virtual_machine

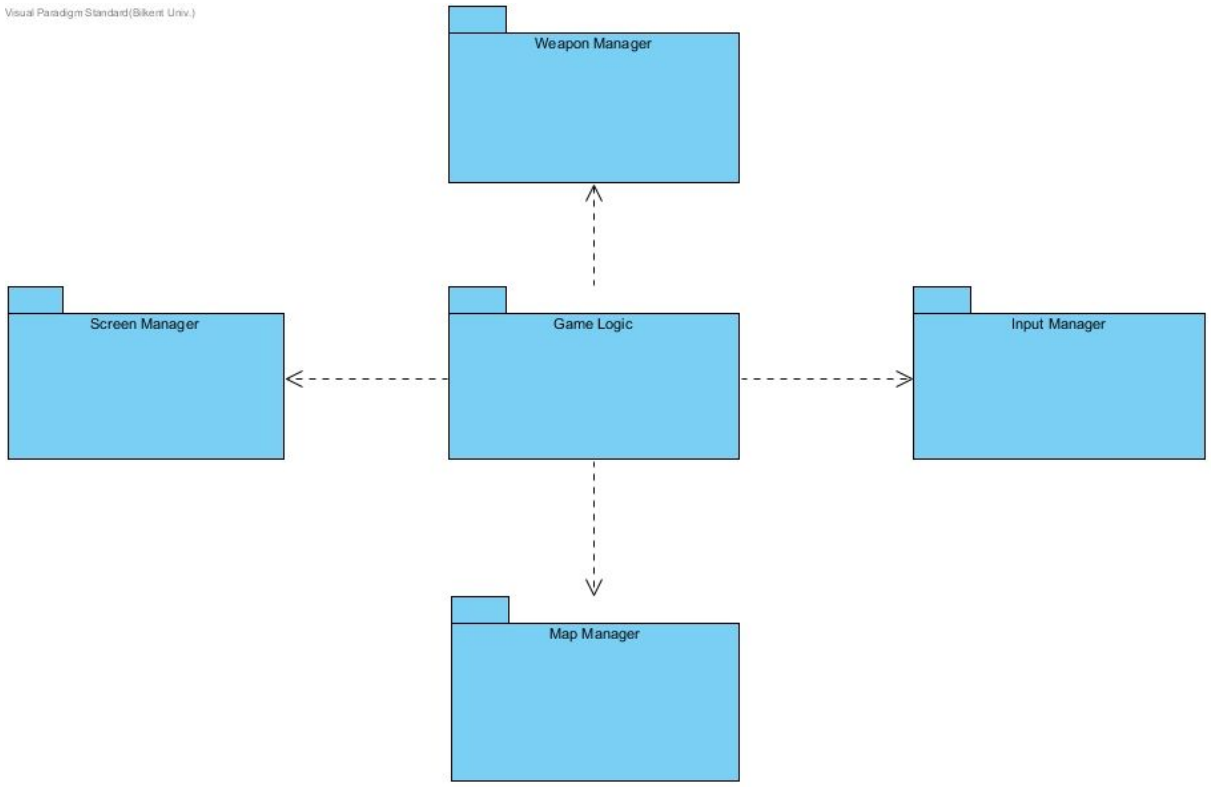
[2] https://techterms.com/definition/jre

[3] https://en.wikipedia.org/wiki/Frame_rate

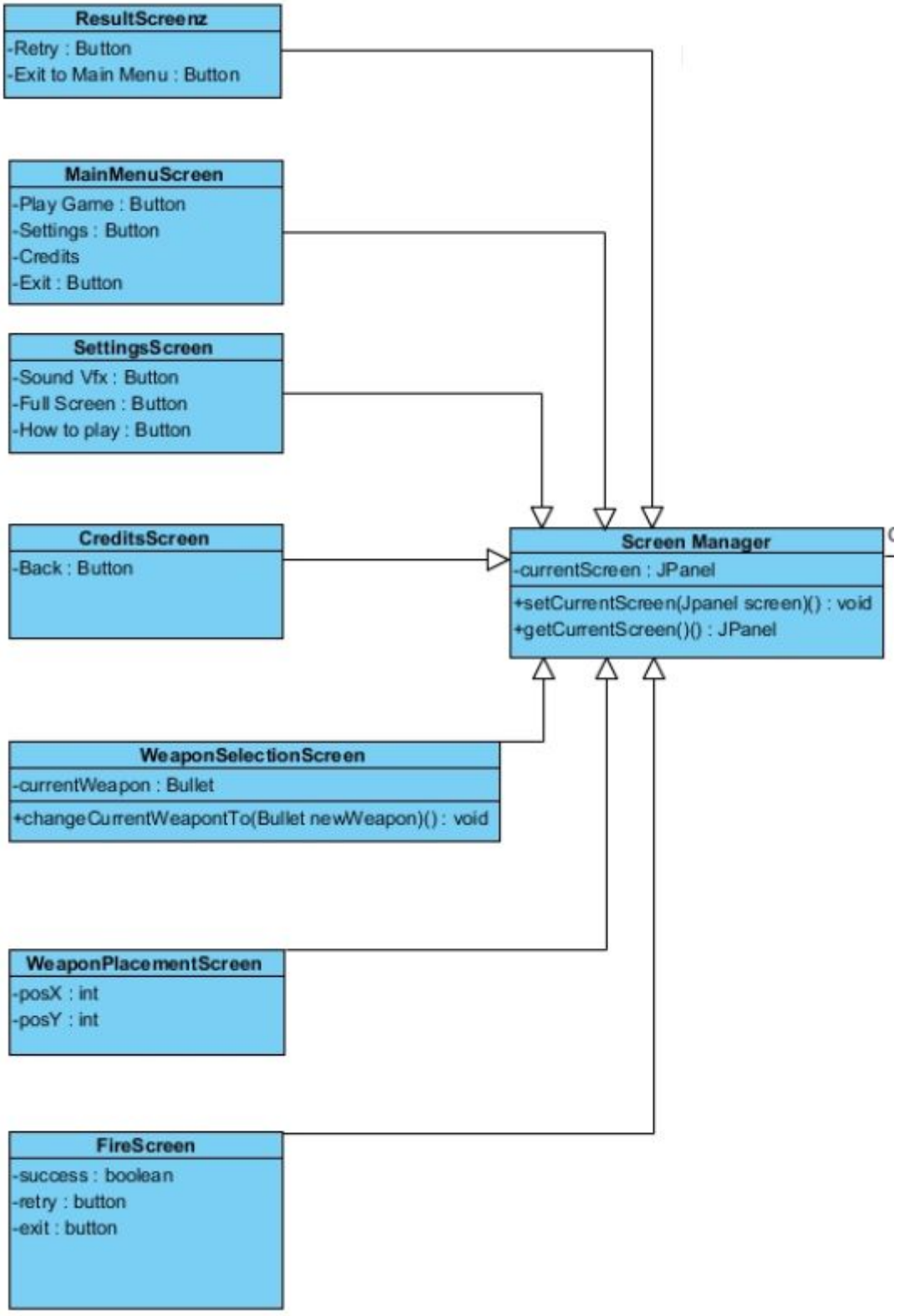
2. Software Architecture

2.1 Subsystem Decomposition

We have decided to create our system based on 4 major components and 1 game logic to control them all.



Above figure shows these components and their controller. “Weapon Manager, “Input Manager”, “Screen Manager” and “Map Manager” will be main components.The “Game Logic” component will use these components and control them.


ResultScreenz
-Retry : Button
-Exit to Main Menu : Button
MainMenuScreen
-Play Game : Button
-Settings : Button
-Credits
-Exit : Button
SettingsScreen
-Sound Vfx : Button
-Full Screen : Button
-How to play : Button
CreditsScreen
-Back : Button
Screen Manager
-currentScreen : JPanel
+setCurrentScreen(Jpanel screen)() : void
+getCurrentScreen()() : JPanel
WeaponSelectionScreen
-currentWeapon : Bullet
+changeCurrentWeapontTo(Bullet newWeapon)() : void
WeaponPlacementScreen
-posX : int
-posY : int
FireScreen
-success : boolean
-retry : button
-exit : button

Screen Manager will be responsible for switching between screens and screens itselves. There will be 7 different screens. These will be: “MainMenuScreen” as the name suggest, will be showing main menu and corresponding buttons. “SettingsScreen” will be showing settings such as “SFX Toggle”, “How to play” and similar settings like that. “CreditsScreen” will be showing contributors. “WeaponSelectionScreen” will be showing different weapons and first map that user will be showing. So that user will be looking at current map select its weapon based on that map. “WeaponPlacementScreen” will be showing current position of the selected weapon proper hints to how to place it in an efficient way. “FireScreen” will be showing current movement of the bullet and how it behaves under forces. Finally “ResultScreen” will be showing whether user hit the target or not. All of these screens will be used by Screen Manager and this component will have proper methods to chose current screen.

Similarly “WeaponManager” component will be responsible for 4 different weapon types and current selected weapon. It will be holding anything related to weapon’s properties.

“Input Manager” as the name suggest, will be responsible for the inputs that user give. When “Game Logic” ask the “Input Manager” whether a key-button has been pressed, it will do the calculations and tell the answer.

“Map Manager” will be responsible for the map properties such as their forces. There will be 4 different maps and each map will have different force value and different force type. “Map Manager” can tell “Game Logic” the current map, current forces or “Game Logic” can tell “Map Manager” to create a random map or update the level.

2.2 Hardware/Software Mapping

Since we’ll be developing this game on Java, game will require JRE to be executed. As hardware requirements, the game will be require a keyboard and a mouse. So that user can interact with the game. Since we’ll be using sound effects and graphical visual, user will also be needing a sound device to hear the sounds and proper GPU to see visuals. Java’s graphics does not require a complicated GPU so

anything would work in that case. Only requirement here is, GPU should have GPU Acceleration feature.

2.3 Persistent Data Management

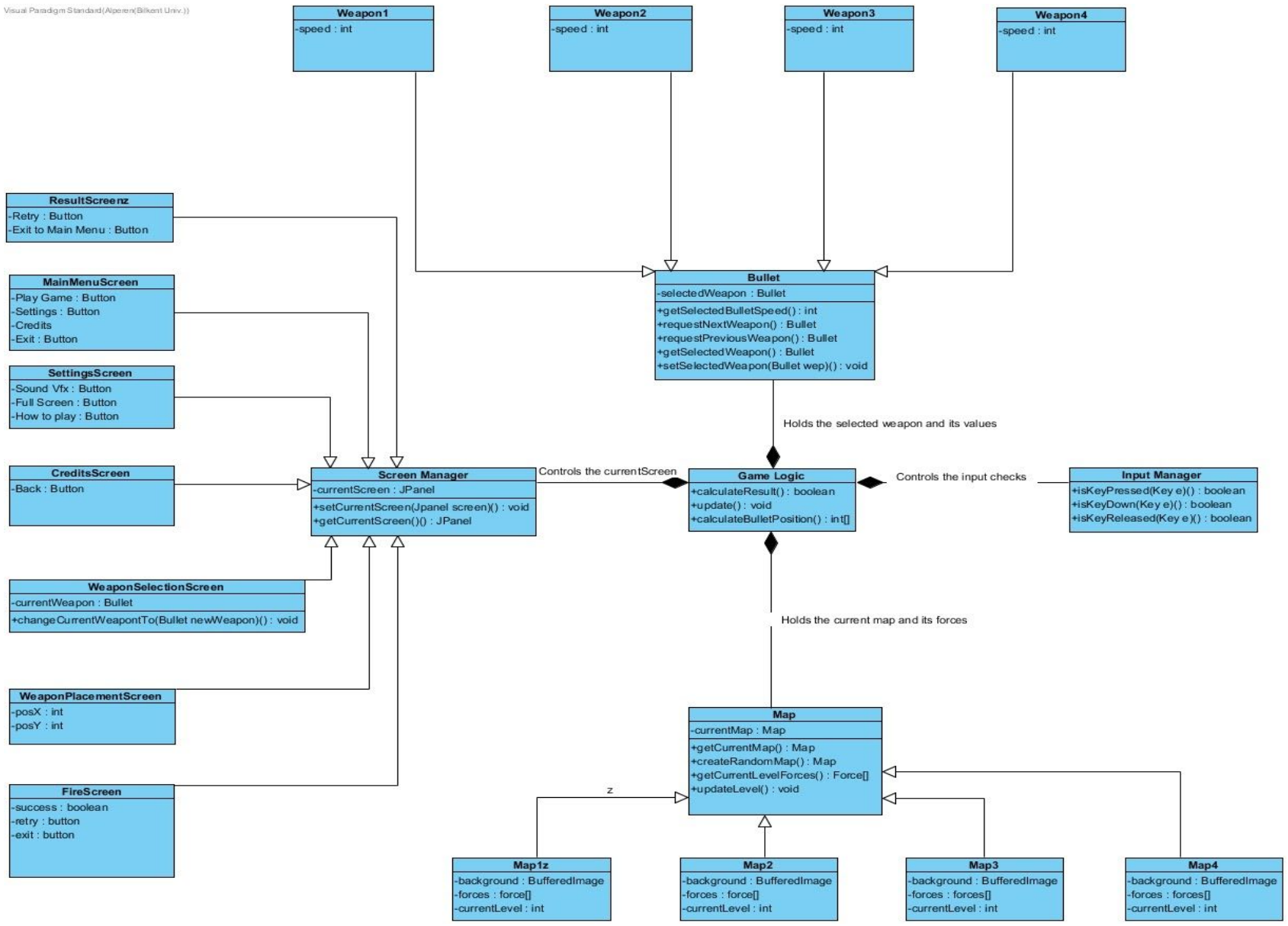
We'll be saving our resources in the same file where game will be located. They won't be encrypted. Any modders who would like to "mod" the game in their desire will be able to change it. That's being said, we also will not be require complicated data management system. We will simply "load" the resources that will be required in that screen. Then once we used that screen and wanted to switch next one, we'll load new ones. Since we'll use minimum amount of pictures and sound effects, loading them in a efficient way won't matter in our case.

2.4 Access Control and Security

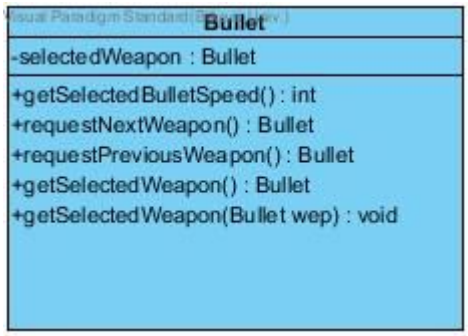
There wont be any security level other than input controls. We want our user to not reach our core components. Therefore game logic should only use the functions of these components. The variables inside the components should be private and should be written only in their class. Our game logic will only read it. The part of the reason why we separate controller, model and view is this. We want components to responsible only themselves.

3. Subsystem Services

The detailed class diagram is given below. It features basic functions and features that each component does and should give basic understanding of how each component works. "Game logic" will have the main class.


Visual Paradigm Standard(Alperen(Bilkent Univ.))
Weapon1
-speed : int
Weapon2
-speed : int
Weapon3
-speed : int
Weapon4
-speed : int
ResultScreenz
-Retry : Button
-Exit to Main Menu : Button
MainMenuScreen
-Play Game : Button
-Settings : Button
-Credits
-Exit : Button
SettingsScreen
-Sound Vfx : Button
-Full Screen : Button
-How to play : Button
CreditsScreen
-Back : Button
Bullet
-selectedWeapon : Bullet
+getSelectedBulletSpeed() : int
+requestNextWeapon() : Bullet
+requestPreviousWeapon() : Bullet
+getSelectedWeapon() : Bullet
+setSelectedWeapon(Bullet wep)() : void
Holds the selected weapon and its values
Screen Manager
-currentScreen : JPanel
+setCurrentScreen(Jpanel screen)() : void
+getCurrentScreen()() : JPanel
Controls the currentScreen
Game Logic
+calculateResult() : boolean
+update() : void
+calculateBulletPosition() : int[]
Controls the input checks
Input Manager
+isKeyPressed(Key e)() : boolean
+isKeyDown(Key e)() : boolean
+isKeyReleased(Key e)() : boolean
WeaponSelectionScreen
-currentWeapon : Bullet
+changeCurrentWeapontTo(Bullet newWeapon)() : void
Holds the current map and its forces
WeaponPlacementScreen
-posX : int
-posY : int
FireScreen
-success : boolean
-retry : button
-exit : button
Map
-currentMap : Map
+getCurrentMap() : Map
+createRandomMap() : Map
+getCurrentLevelForces() : Force[]
+updateLevel() : void
z
Map1z
-background : BufferedImage
-forces : force[]
-currentLevel : int
Map2
-background : BufferedImage
-forces : force[]
-currentLevel : int
Map3
-background : BufferedImage
-forces : forces[]
-currentLevel : int
Map4
-background : BufferedImage
-forces : forces[]
-currentLevel : int

Bullet Class



Bullet class determines and rules the main options about different weapon types and their bullet reactions under the effect of external forces.

selectedWeapon : Bullet

It keeps the selected current weapon that user choose.

getSelectedBulletSpeed() : int

It returns the speed of the bullet according to current weapon as an integer.

requestNextWeapon() : Bullet

According to current selected weapon it demands next weapon in weapon selection class, for instance; if current weapon is weapon1, it demands weapon2.

requestPreviousWeapon() : Bullet

Reversely, according to current selected weapon it demands previous weapon in weapon selection class.

getSelectedWeapon() : Bullet

It returns current selected weapon as a Bullet object.

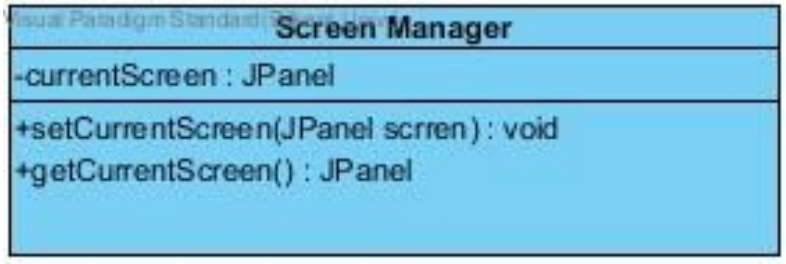
setSelectedWeapon(Bullet wep) : void

Finally it sets the selected weapon according to user's choice.

Weapon1, Weapon2, Weapon3, Weapon4 Classes

These are the classes which extends Bullet class and they have their own speed variable which determines the bullet speed of the current weapon that results the different reactions under the same affected force.

ScreenManager Class



This class rules which screen will appear on game according to users choices.

currentScreen: JPanel

The current JPanel that is running on game.

setCurrentScreen(JPanel screen) : void

Sets the JPanel to change the screen that seen by user.

getCurrentScreen() : JPanel

Returns the current screen panel.

MainMenuScreen Class

It is the class of the first screen that user see when he/she open the game. It will consist of buttons which will have all listeners for their actions.

PlayGame : Button

This button directs user to WeaponSelectioScreen.

Settings : Button

Directs to the SettingsScreen class.

Exit : Button

Exits the game.

SettingsScreen Class

Keeps the basic choices/settings that user can change.

Sound : Button

User can open or close the sound

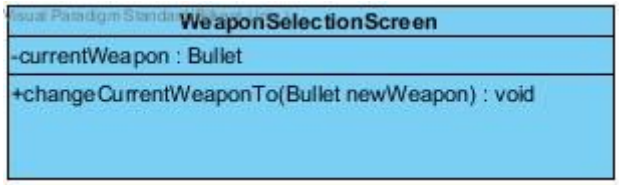
FullScreen : Button

User can change the size of the window(setting it as full screen mode or not).

HowToPlay : Button

Opens a text box that will explain how to play game.

WeaponSelectionScreen Class



It is the screen that user can select/change his/her current weapon.

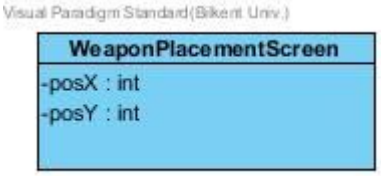
currentWeapon : Bullet

This variable keeps the current weapon that will shown on screen as a bullet object.

changeCurrentWeaponTo(Bullet newWeapon) : void

Sets/changes the current weapon according to the users keyboard choices which is taken from InputManager class.

WeaponPlacementScreen Class



After selection of weapon this screen will shown by ScreenManager class. It is the screen that user can first determine the position of weapon to fire.

posX : int, posY : int

Clearly it is the position of the weapon that is set by user for fire.

FireScreen Class

The screen that fire process occur after the placement of weapon is done.

success : Boolean

Boolean variable that is determined true if user can hit the target, otherwise it is false.

Retry : Button

If the user think that he/she placed weapon in wrong position, this button gives a chance to replace it and come to fire screen again.

Exit : Button

Exit game/program.

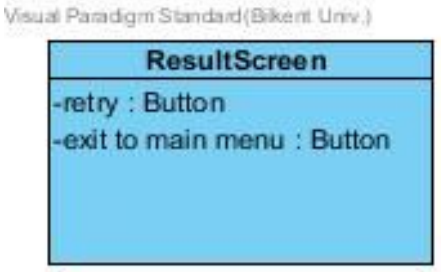
CreditsScreen Class

Basically it is credits screen and will have text and return back button.

Back : Button

Returns user back to main menü.

ResultScreen Class



After fire is done, result screen appears, shows the result and if he/she cannot hit gives retry option.

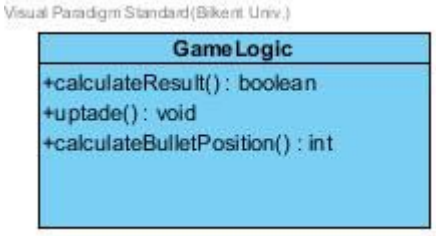
Retry Button

If the target cannot be hit and button gives a chance to replace it and come to fire screen again.

Exit Button

Returns user back to main menu.

GameLogic Class



It is the main class of the game which will have inputmanager, screenmanager, bullet and map manager components and will combine them all to play the game.

CalculateResult() : boolean

According to the position of the bullet, when it come to the end point where target occurs, if they meet at same position interval.

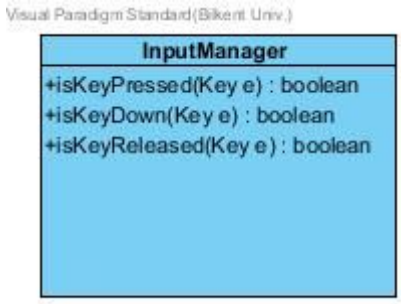
CalculateBulletPosition() : int[]

It will return the position of the bullet which is determined by different forces and weapon types.

update() : void

As it is the main ruler class, it is updated on following time to control connections between classes by taking current variables.

InputManager Class



Input Manager class arrange the inputs which are pressed by users.

isKeyPressed(Key e): boolean

This operation checks arrow or space key's position. It means if user press one of them it returns false or true (depends on the implementation).

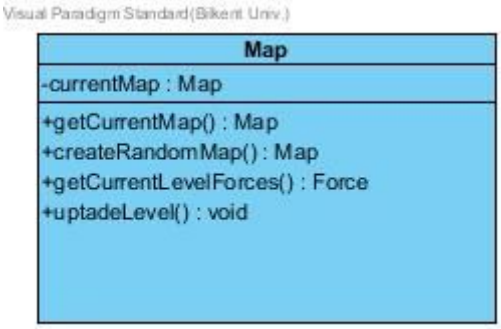
isKeyDown(Key e): boolean

This operation takes the duration of pressing and return a boolean.

isKeyReleased(Key e): boolean

When user stop pressing button, this method invokes the other screen or proccessing managers.

Map Class



In map class, we have four different operation and one variable which keeps the current map. In this class we will organize the level system as well and ofcourse the force operation will appear in this class.

currentMap: Map

This is a map object which keeps the present map. It is really useful to organize level system.

getCurrentMap(): Map

This method is necessary when we need to change current map while playing.

getCreateRandomMap(): Map

In our game we will have a random map in each level system. This method or approach allows user to see target. This method is for it.

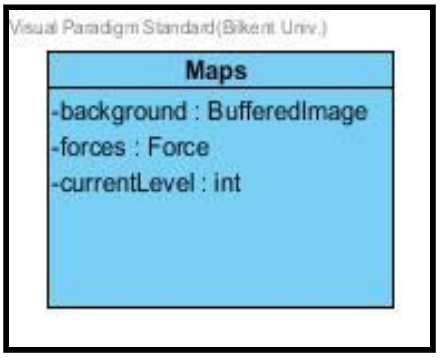
getCurrentLevelForces(): Force

As we discussed before, we have a couple of forces that make a little bit challenge for users. This method keeps the level's force.

uptadeLevel(): void

This method simply changes or uptades current level.

Also we have four maps for user to increase variety of game's scnerio. One of their diagram is here and others have same diagram as well.



background: BufferedImage

Each map have its own theme and background image. This variable keeps the image that we declare to each map.

forces: Force

The group of forces' names and their effects are kept in big map class. In this class, forces variable keeps the current force of current map.

currentLevel: int

Each map is, actually, a level fr our game, and we hold the level by using this variable.

4. Glossary