



CS 319 - Object-Oriented Software Engineering System Design Report

Defenders Of The Kingdom

Group 3-H

Alp Ege Baştürk

Barış Eymür

Emre Gürçay

Öykü Ece Ayaz

Table Of Contents

1. Introduction	3
1.1 Purpose of the System	3
1.2 Design goals	3
1.3 Definitions, Acronyms, Abbreviations	5
1.4 Overview	5
2. Software Architecture	6
2.1 Subsystem Decomposition	6
2.2 Architectural States	7
2.2.1 Layers	7
2.3 Hardware / Software Mapping	9
2.4 Persistent Data Management	9
2.5 Access Control and Security	9
2.6 Boundary Conditions	10
3. Subsystem Services	11
3.1 Detailed Class Diagram	12
3.2 Design Patterns	13
3.3 User Interface Subsystem	13
3.5 Game Screen Objects Subsystem	22
3.6 File Management Subsystem	28
3.7 Input Management Subsystem	29
3.8 Game Map Management Subsystem	30
4.Low-level Design	32
4.1 Object Design Trade-Offs	32
4.2 Final Object Design	33
4.3 Packages	34
4.3.1 java.util	34
4.3.2 java.awt	34
4.3.3 java.awt.event	34
4.3.3 javax.swing	34
4.4 Class Interfaces	34
4.4.1 ActionListener	34
4.4.2 MouseListener	34
5. References	35

1. Introduction

1.1 Purpose of the System

Defenders of the Kingdom is a basic tower defense game. There are a lot of tower defense games available today, which have different gameplays and which require different strategies to be successful. Compared to other available tower defense games in the market, graphics and transitions of Defenders of the Kingdom are genuinely simple. Another distinguishing feature of the game is that the gameplay provides heroes and obstacles for the purpose of defense. These entities are purchased by the player. The aim of such innovations in the game is to build a more appealing and complicated game. Defenders of the Kingdom requires good time management and hand-eye coordination to play.

1.2 Design goals

In this section we will focus on the design goals of the system. We have provided our design goals in the non-functional requirements section of the analysis report.

End user criteria:

Ease of Use: The system we are developing is a game. In this game, our aim is to entertain people. The type of game we are developing is a strategy game. So, it is also good for people who likes to think different strategies while playing games. In order to make the game more playable and entertaining, the game will have a user friendly interface which is compatible with the story of the game. Also, the player will be able to access to desired menus and go through them easily. The gameplay is very easy to learn. The user can play the game just by using the mouse.

Ease of Learning: Before starting to play the game, the player may not have any knowledge about the game. To help the user, we decided to make our game

easy to learn. In the main menu of the game we will provide a help button for the user. When the user has questions about the game or gameplay, (s)he can find answers to his/her questions in the help page.

Maintenance Criteria:

Extendibility: To make our game compatible with the modern software world, we are writing our code in a way that, it will be easy to add new features to the game. For example, different types of towers, heroes and obstacles can be added to the game later on. This will make our game more dynamic.

Performance: It is one of our priorities to provide a high performance game. The implementation of the game is designed to ensure the response of the game is instantaneous and the flow of animations are smooth.

Portability: We will implement the game in Java, because Java is an object oriented language and offers many features. According to Oracle, currently over 3 Billion devices run Java [1].

In terms of probability that is a huge number.

Modifiability: While we are developing our game, we will implement it in a modifiable way, for the future enhancements. To make this possible we will try to minimize the ripple effects that can happen.

Security: In terms of security, there is not a possible threat that can cause some damage to user's computer. As the game requires no personal information or passwords, the user can play game with no security worries.

Short Response Time: As the system we will develop is a game, good performance is a must. The game should be responsive because every second in the game matters. When the user decides to create something it should be quick

because meanwhile the attackers do not stop. Also in terms of gameplay, the animations will be smooth so that the user can enjoy the game.

Trade Offs:

In the game, we want the animations to be smooth and quick but this increases the usage of the memory. At the beginning of each level, a map will be generated by using the matrix in the code. This makes the beginning of the levels a little bit slow and it also increases the memory usage at the beginning. However, later during the game play, in terms of performance and memory usage it will give good performance, since we only create the map at the beginning of each level. We want to have smooth transitions between moves and this causes some extra memory usage but when we look at the games in the market, they all have some trade offs.

To make the game easier to learn and play, we decided not to add functionalities that may cause the user to get lost in the system. Thus, we did not add functionalities such as settings menu. We used minimalistic design at the user interface to make it possible for the user to easily switch between the menus.

1.3 Definitions, Acronyms, Abbreviations

Abbreviations:

JDK: Java Development Kit

1.4 Overview

The main purpose of our system is to make the player enjoy our game and encourage him/her to think clever strategies to be successful in the game. To achieve this, we have determined some design criteria, which include ease of use, ease of learning, extendibility, performance, portability, modifiability, security and short response time. To fulfill our design criteria, we have made some trade-offs. We sacrificed from memory to generate new maps and make smooth transitions. Also, we made some functionality sacrifices, such as limiting the user interface and not including a settings menu.

2. Software Architecture

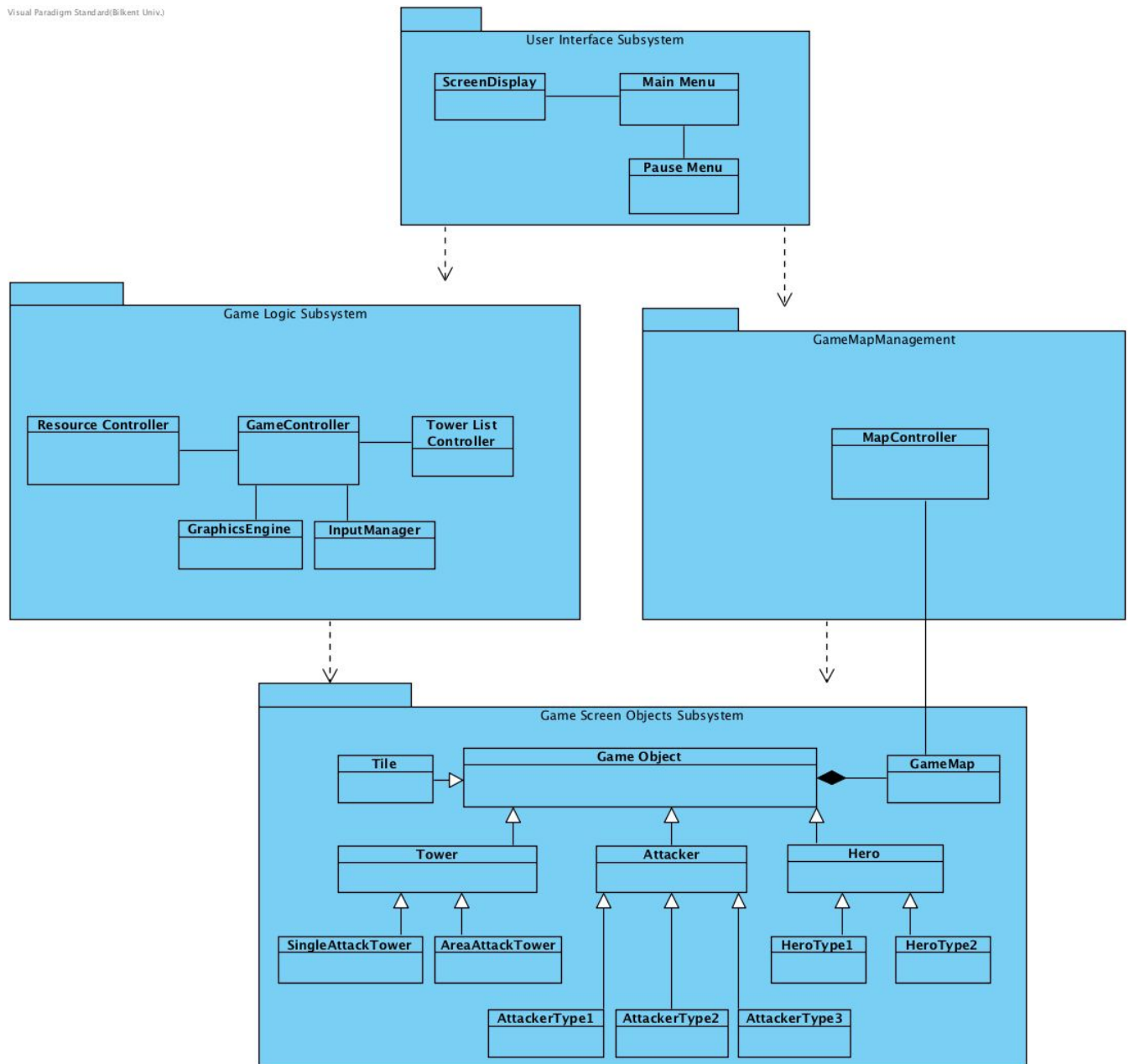
2.1 Subsystem Decomposition

In this section, we have designed the subsystems in our software. During our design, we have considered the functional and non-functional requirements and attempted to fulfill them as we have created the decomposition of the system.

There are four main subsystems which are User Interface, Game Logic, Game Screen Objects and Game Map Management. User Interface subsystem directly interacts with user and contains menu-related classes. Game Logic Subsystem is a controller subsystem and it manages the entity objects in the game. Similarly, Game Map Management subsystem controls the game map specifically. Game Screen Object subsystem is an abstraction of all the classes regarding the game objects.

The subsystems do not heavily rely on each other, they are rather loosely connected. The purpose of this design is to allow us to be flexible in the design and implementation parts if any required changes or flaws arise in the future. For instance, Game Map Management Subsystem only directly interacts with the Game Map class in the Game Screen Objects Subsystem.

The decomposition of the subsystems is illustrated below.

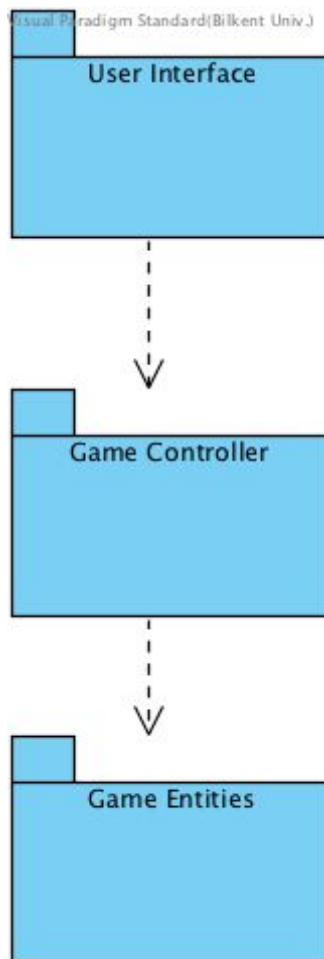


2.2 Architectural States

2.2.1 Layers

The layers of our system has three main components: User Interface, Game Controller, Game Entities. We have packed the Game Logic and Game Map Management subsystems in one layer called Game Controller since they both serve

for the same purpose of game controlling. These three layers have hierarchical relationship. The highest hierarchy in the layer is the User Interface, which the player directly interacts with. User Interface layer is followed by the Game Controller. Game Controller layer is responsible to manage the game with user selection from the Main Menu. Our bottom layer is Game Entities, in which the entity objects of the system interacts with each other. The layer decomposition of the system is illustrated below.



2.3 Hardware / Software Mapping

Java will be used to implement Defenders of the Kingdom. JDK 8 will be used. To reduce the complexity and to provide easier game play, only a mouse will be required to play the game. The mouse will be used to get the user input during game play. We tried to keep our system requirements minimal. As a result, a simple computer with an operating system and a Java compiler to compile and run java programs will be enough to play. While choosing the programming language to implement our project, we considered that Java has platform independency. We thought that, it will make our game more portable.

We will use text files to store the game maps and the data of the player's success in the levels. Our game will work offline.

2.4 Persistent Data Management

The game maps will be stored in .txt files in hard disk. These text files will be generated before the game is released. Each level's map will be unique and they will be persistent. If an issue about the text files occur, then the game will not be able to load game maps. However, this will not affect our game objects such as defense towers, attackers, etc. We will store our game object images as .png files in hard disk.

2.5 Access Control and Security

Defenders of the Kingdom will not require any network connection to play. The users will be able to play this game, after they compile it in their computer or download pre-compiled version. There won't be a user profile structure in the game. As a result, there won't be any kind of security or privacy issues in our game.

2.6 Boundary Conditions

Initialization

The game will be an executable .jar file. This will make the game portable

Termination

The game will be terminated by clicking quit game button in the main menu.

Player may pause the game during the gameplay, return to main menu and quit from this menu.

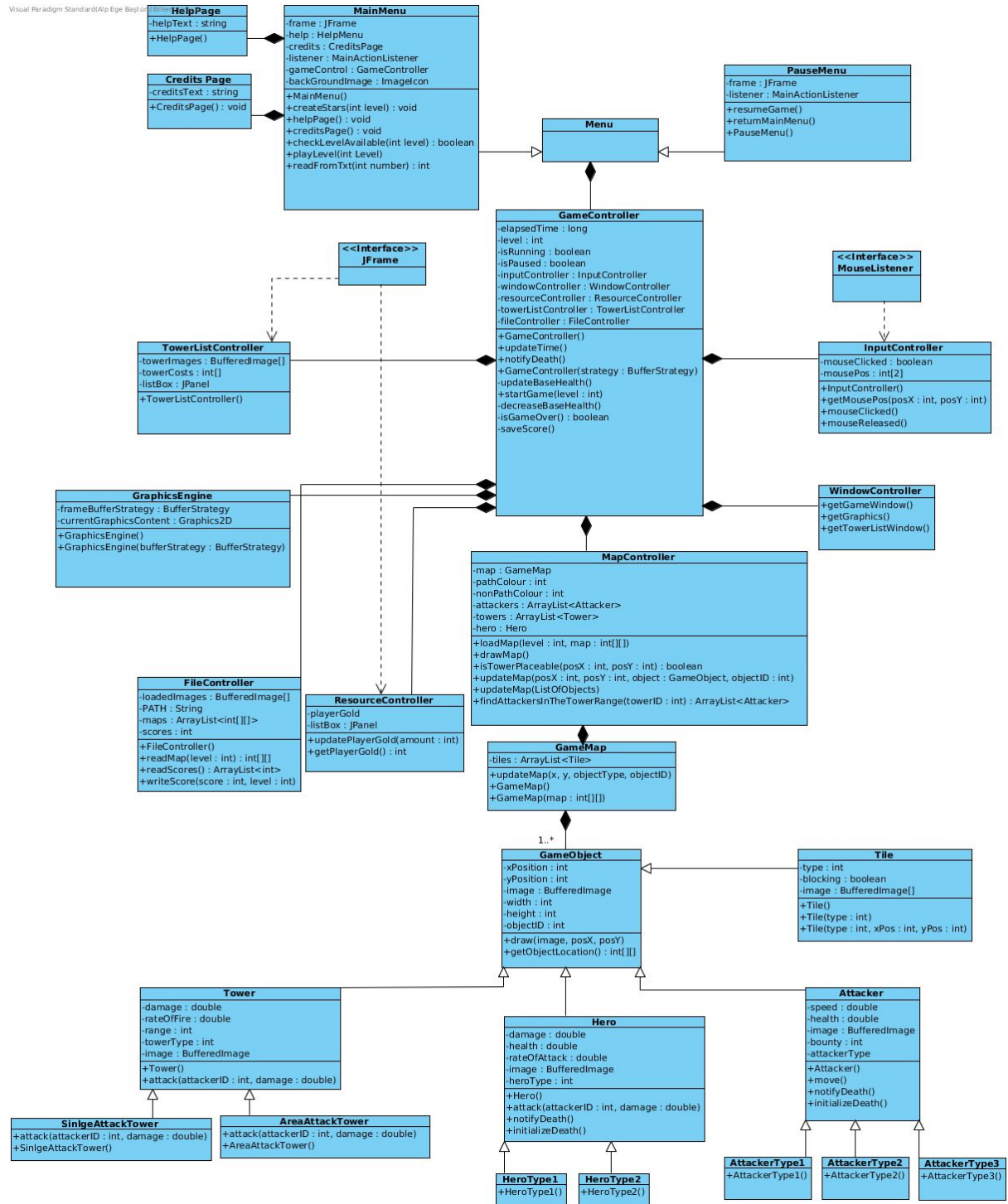
Error

If an error occurs during the loading of resources, game will give an error message accordingly and quit. If an error occurs during the gameplay, game will return to main menu and give an error message. Player information before the start of the failed game will be valid. If program crashes without any control, player will lose all current data and he or she may lose previous data, depending on the error.

3. Subsystem Services

In this section we will provide detailed information about the interfaces of our subsystems. Detailed class diagram of the project is provided below.

3.1 Detailed Class Diagram



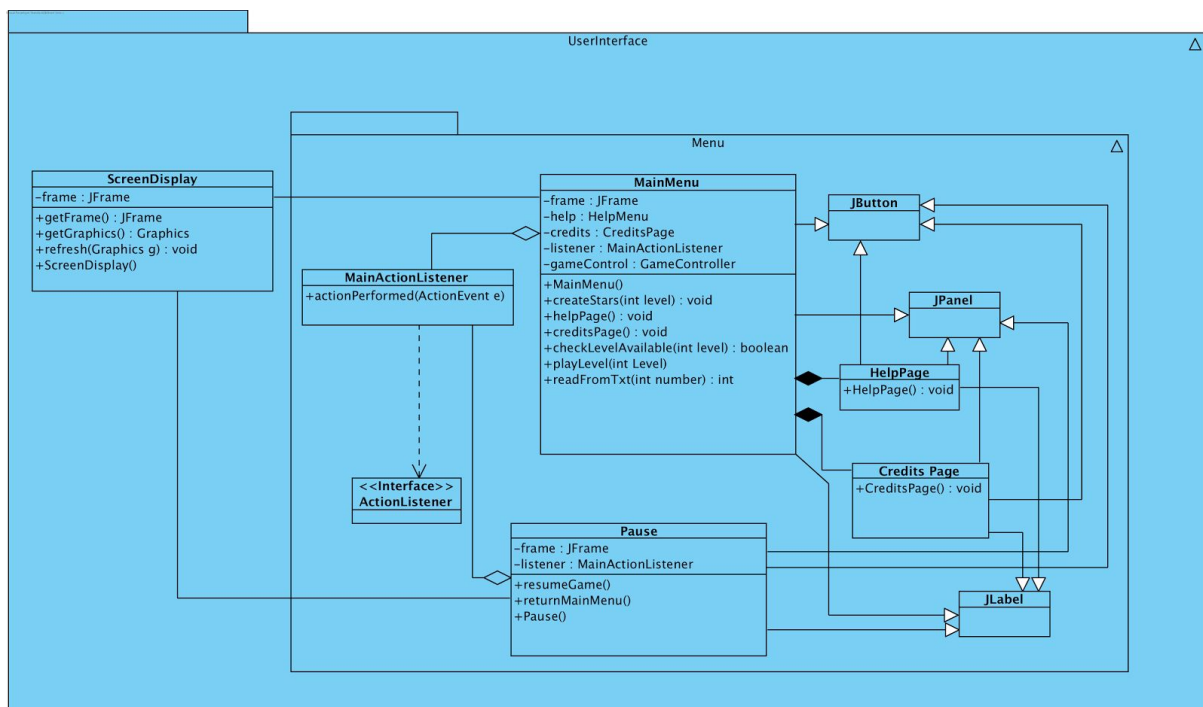
3.2 Design Patterns

We will use Façade design pattern while implementing our code. Façade class for the game logic package is the Game Controller class. It interacts with most of the objects and subsystems. Façade class of the user interface is the MainMenu.

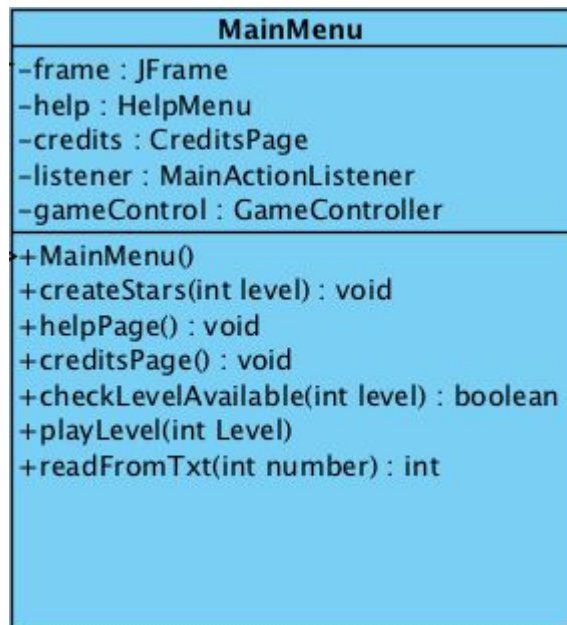
3.3 User Interface Subsystem

According to *bwired* “usability is the foundation of a successful user interface... and A positive user experience is a critical part of the overall customer experience...” Thus in our game we will implement the user interface as usable as possible [2].

This subsystem is responsible for user interactions.



MainMenu Class



MainMenu is the façade class of this subsystem.

Attributes

private JFrame frame: This is the frame where we display the visual concepts of the MainMenu.

This frame also uses components JButton, JLabel, JPanel...

private MainActionListener listener: This is the listener we use for getting the user actions from user interface.

private HelpMenu help: This is the panel of Help class where user can find all the information's about the game. To create the panel we will use JPanel.

This panel also uses components JButton, Jabel..

private CreditsPage credits: This is the panel of Help class where user can find contact informations about developers of the game. To create the panel, we will use JPanel.

This panel also uses components JButton, JLabel..

private GameController gameController: Stores an instance of GameController class which is initialized when users selects to start the game.

Constructors:

MainMenu(): MainMenu initialize the components of the class. Such as **frame**, JButton, JLabel, JPanel, **listener**, **gameControl**, **credits**

Methods:

public playLevel(int Level) : This method calls for the desired the level from the GameController to play. In order to that it also checks for level availability by calling the function checkLevelAvailable().

public checkLevelAvailable(int Level): This method checks whether the desired level is allowed to be play. In the game for example can not play the Level2 before succesfully finishing Level1. The success rate of the levels is kept in a .txt file. So this method reads the level success rate from a .txt file by calling readFromTxt(int number) function and returns a boolean value accordingly.

public createStars(int Level): This method creates stars for showing the success rate. To show the success rate it uses star image and displays them on the screen. To know about the success rate it calls the readFromTxt(int number) function and places them on a JPanel.

public helpPage(): This method creates the helpPage.

public creditsPage(): This method creates the creditsPage.

public readFromTxt(int number): This method reads the success rate of a desired level from the .txt file and returns the value.

ScreenDisplay Class

ScreenDisplay
-frame : JFrame
+getFrame() : JFrame +getGraphics() : Graphics +refresh(Graphics g) : void +ScreenDisplay()

Attributes

private JFrame frame: This is main frame of the page and all the contents are displayed on this frame.

Constructors

publicScreenDisplay() : It initializes the components of the class. Such as JFrame.

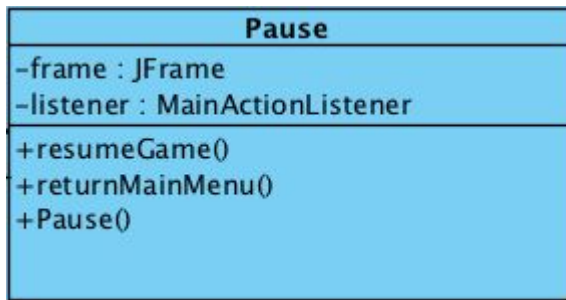
Methods

public getFrame(): This method returns the JFrame object .

public Graphics getGraphics(): In the there are lots of graphics content used for displaying images and drawing. This method returns those graphics contents.

public refresh(Graphics g): In order to have good and smooth transitions for the images we are required to use refresh method for graphics.

PauseMenu Class



Attributes:

private JFrame frame: This is the frame of the pause menu.

private MainActionListener listener: Listener for the objects on the frame.

Constructor:

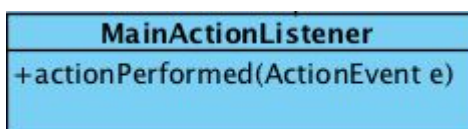
Pause(): It initializes the components of the class.

Methods:

public void resumeGame(): Depending on the button that is clicked, this method is called and it returns the user to the game again. Game continues from the last position of the actors.

public void returnMainMenu(): This method returns the user to the main menu but when the user chooses to go to the main menu then it means (s)he did not successfully finish the level (s)he was playing.

MainActionListener

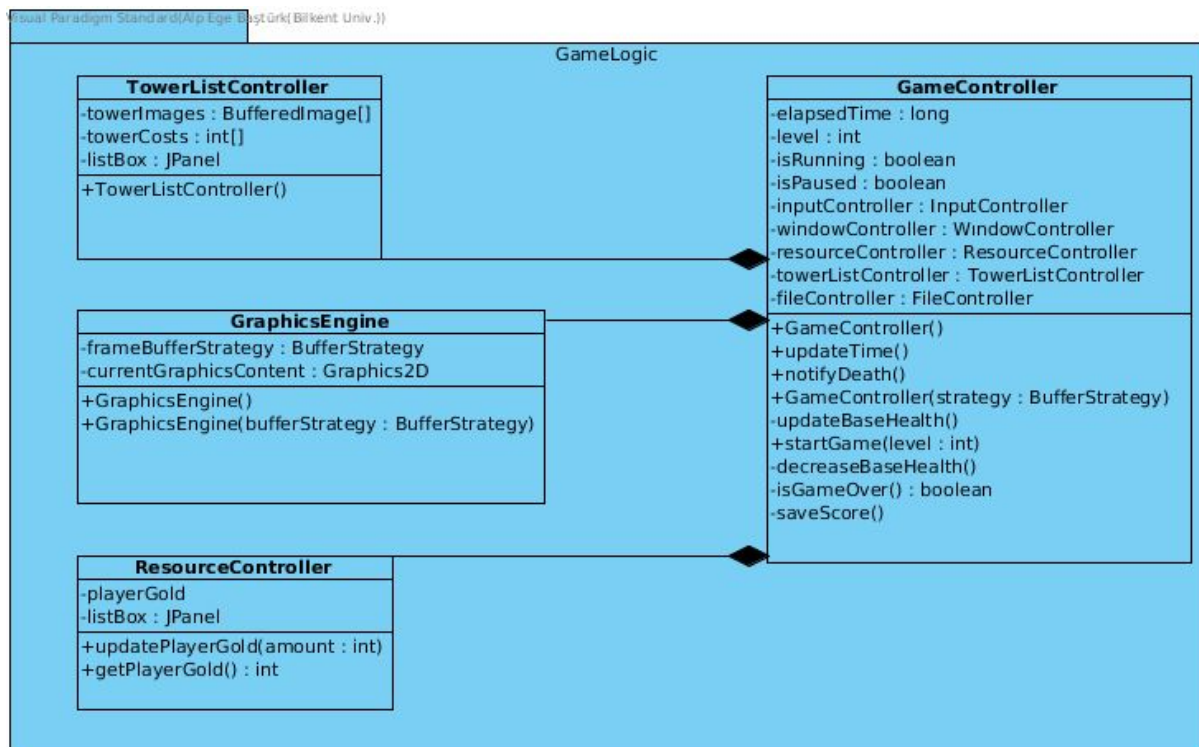


Method:

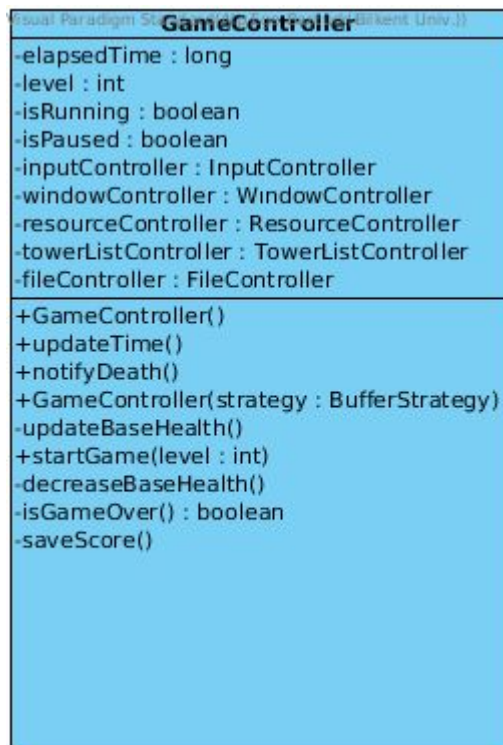
public void actionPerformed(ActionEvent e): This is the overridden version of the actionPerformed(ActionEvent e) function of ActionListener interface.

3.4 Game Logic Subsystem

This system is responsible for the main logic of the game. It interacts with user actions and controls the elements and the gameplay.



Game Controller Class



This is the façade class of the game. It interacts with user and other systems. It starts the game loop according to the inputs. It interacts with the input management in order to load the game map according to the level selected by the user in the main menu. It starts resource and tower controllers. Also this class handles the movement and attack of the objects on the map. Later it provides information to map management to update the map. At the end of the loop it makes Graphics Engine to Draw. If isGameOver() returns true it checks the score display success or fail and saves score for that level by calling input management. After these it returns to the main menu.

Attributes:

private long elapsedTime: This is the time that the current level is played.

private int level: This is the number of the current level which is being played.

private boolean isRunning: This is the Boolean value which is TRUE if the game is currently running and FALSE if the game is paused.

private boolean isPaused: This is the Boolean value which is TRUE if the game is currently paused and FALSE if the game is running.

private InputController inputController: This is a controller to control the user input and take actions according to it.

private WindowController windowController: This is a controller to control the game windows.

private ResourceController resourceController: Stores resource controller instance

private TowerListController towerListController: This is a controller to control the list of towers.

private FileController fileController: Stores file controller instance

Constructors:

GameController(): It initializes the components and attributes of the class GameController.

GameController(BufferStrategy strategy): Constructor with passed strategy parameter

Methods:

public void updateTime(): This method updates the time of the game.

public void notifyDeath(): This method prints a message to the screen to notify the user that an attacker has been killed.

public void updateBaseHealth(): This method is used to increase the base health of the player's city.

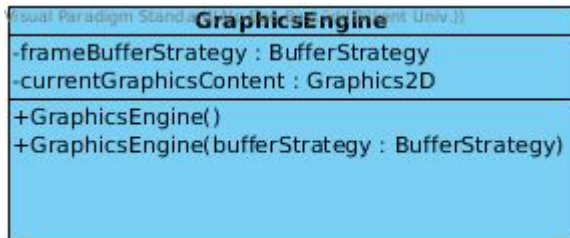
public void startGame(int level): This method starts the level of the game which is taken as a parameter.

public void decreaseBaseHealth(): This method is used to decrease the base health of the player's city.

public boolean isGameOver(): This method returns TRUE if the game is over, returns FALSE if the game continues.

public void saveScore(): This method is used to store the score the player has obtained in a particular level.

GraphicsEngine Class



This class handles the drawing of the objects to the screen. This class will be called by the GameController class to render the screen.

Attributes:

private BufferStrategy frameBufferStrategy: This will be used to define buffer strategy for rendering the images.

private Graphics2D currentGraphicsContent: This is a graphics object to generate the graphics of the GraphicsEngine.

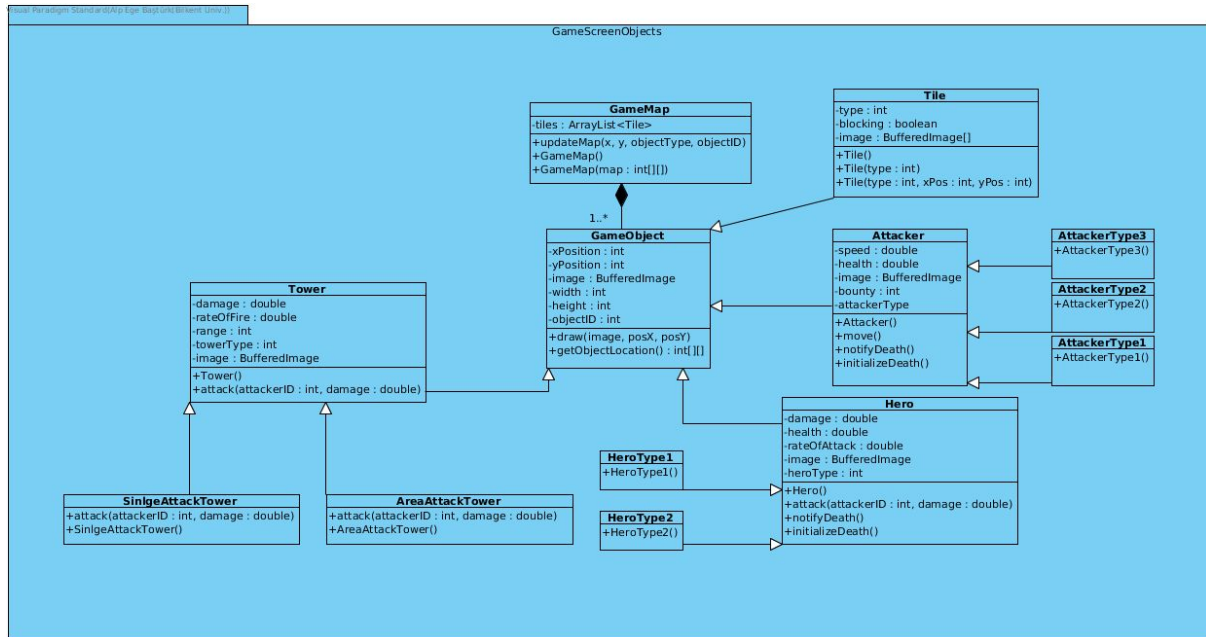
Constructors:

GraphicsEngine(): It initializes the components and attributes of the class GraphicsEngine.

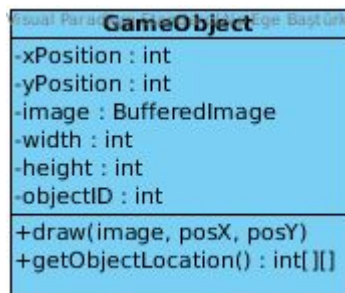
GraphicsEngine(BufferStrategy strategy): Initializes the object with given buffer strategy

3.5 Game Screen Objects Subsystem

This subsystem consists the objects that will be drawn to the game screen. It has the main elements of a tower defence game such as “Tower” and “Attacker”.



GameObject Class



This is the parent class of all the objects that can be shown on the screen. It only provides attributes related to positioning and drawing of the object.

Attributes:

private int xPosition: This represents the x-position of the GameObject in the coordinate system.

private int yPosition: This represents the y-position of the GameObject in the coordinate system.

private BufferedImage image: Game objects will be represented as images on the screen.

private int width: This represents the width of the GameObject in the coordinate system.

private int height: This represents the height of the GameObject in the coordinate system.

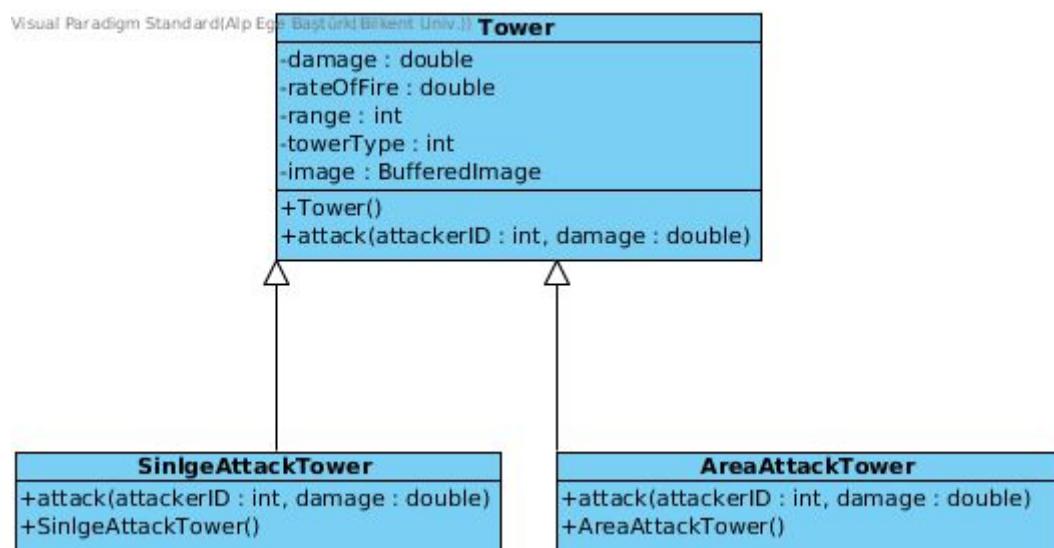
private int objectID: Each GameObject will have a unique ID, which will be stored as objectID.

Methods:

public void draw(BufferedImage image, int posX, int posY): This method draws a gameObject according to the given image and coordinates.

public int[][] getObjectLocation(): This method returns the object location as a 2D array.

Tower Class



Tower is the main class of the tower defence game. It is a game object that will have a position. Towers will have attributes related to their attack such as damage and rate of fire. Child classes of the tower define different types of attack.

These towers will have different attack methods. Graphics for the attack and mechanisms of the attack will be different.

Attributes

Private double damage: This is the variable that holds the damage level of the tower,

Private double rateOfFire: This is the power of the shooting the tower is capable of.

Private double range: This is the affecting range of the tower when it shoots.

Private int towerType: This is the kind of the tower.

Private BufferedImage image: This is the image of the tower.

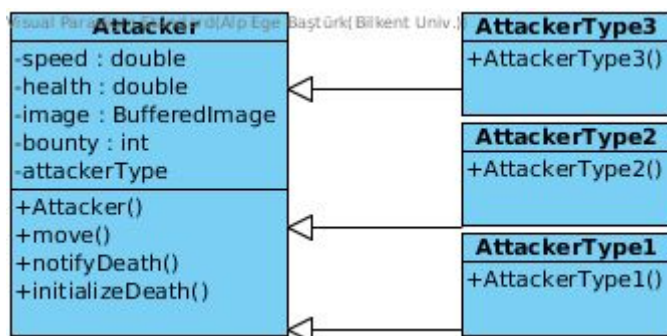
Constructors

Tower(): It initializes the components of the tower.

Methods

Public void attack(int attackerID, double damage): This method attacks the attacker and increases the damage on the attacker.

Attacker Class



Attacker is the parent class of attackers. We plan to have three types of attackers. These attackers will have different attributes.

Attributes

Private double speed: This is the speed of the attacker.

Private double health: This is the health level of the attacker, it decreases as the attacker gets damaged.

Private BufferedImage image: This is the image of the attacker.

Private int bounty: This is the amount of money that the player will earn when s/he kills the attacker.

Private int attackerType: This is the type of the attacker.

Constructor

Attacker(): It initializes the attacker components.

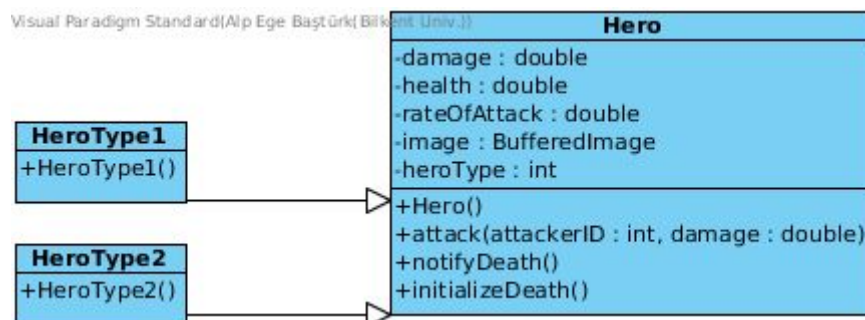
Methods:

Public void move(): It moves the attacker.

Public void notifyDeath(): If the attacker dies, the game controller is notified.

Public void initializeDeath():

Hero Class



Hero class is a class between tower and attacker. We plan to implement it like a defender version of the attacker. However it will have attributes and methods related to attack similar to tower. Also it will block movement along the path until death.

Attributes

Private double damage: This is the damage of the attacker.

Private double health: This is the health level of the attacker, it decreases as the attacker gets damaged.

Private double rateOfAttack: This is the power of the attack damage the hero is capable of causing.

Private BufferedImage image: This is the image of the attacker.

Private int bounty: This is the amount of the money the player needs to purchase hero item.

Private int heroType: This is the type of the attacker.

Constructor

Public Hero(): It initializes the components of hero.

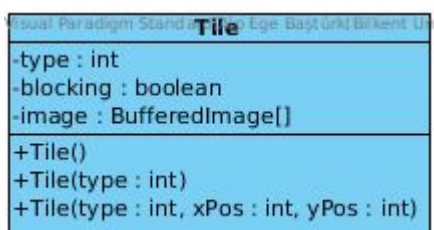
Methods

Public void attack(int attackerID, double damage): It attacks the attackers.

Public void notifyDeath(): If the hero dies, the game controller is notified.

Public void initializeDeath(): This method implements the required actions if health of the hero drops to or below zero.

Class Tile



The image shows a UML class diagram for the 'Tile' class. The class name 'Tile' is in a box at the top. Below it, the attributes are listed: '-type : int', '-blocking : boolean', and '-image : BufferedImage[]'. Below the attributes, the methods are listed: '+Tile()', '+Tile(type : int)', and '+Tile(type : int, xPos : int, yPos : int)'. The diagram is a standard UML representation of a class with its attributes and methods.

Tile
-type : int
-blocking : boolean
-image : BufferedImage[]
+Tile()
+Tile(type : int)
+Tile(type : int, xPos : int, yPos : int)

Tile will be used to draw images on the map, which is a 2-D array. According to the tile type, images will be selected and drawn on the screen. Also we plan to

make the player base a tile. GameController will handle what happens when attackers manage to enter this tile.

Attributes:

private int type: This specifies the type of the tile as an integer.

private boolean blocking: If this is TRUE, the tile will be blocking and game objects will not be able to pass through it. If it is FALSE, then the game objects will be able to pass through it freely.

private BufferedImage image: The tile will be represented as an image on the screen.

Constructors:

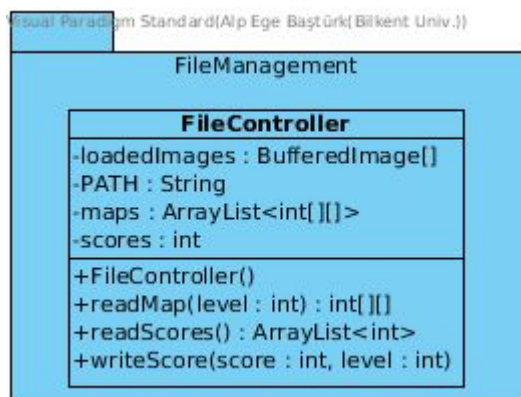
Tile(): It constructs a default Tile object.

Tile(int type): It constructs a tile object according to the given type.

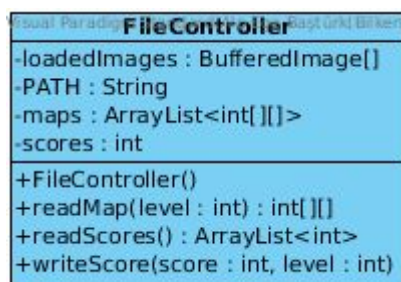
Tile(int type, int xPos, int yPos): It constructs a tile object according to the given type and coordinates..

3.6 File Management Subsystem

This subsystem is used for saving and loading operations. Operations will be handled by the FileController class. This class will be called by the GameController for read and write operations.



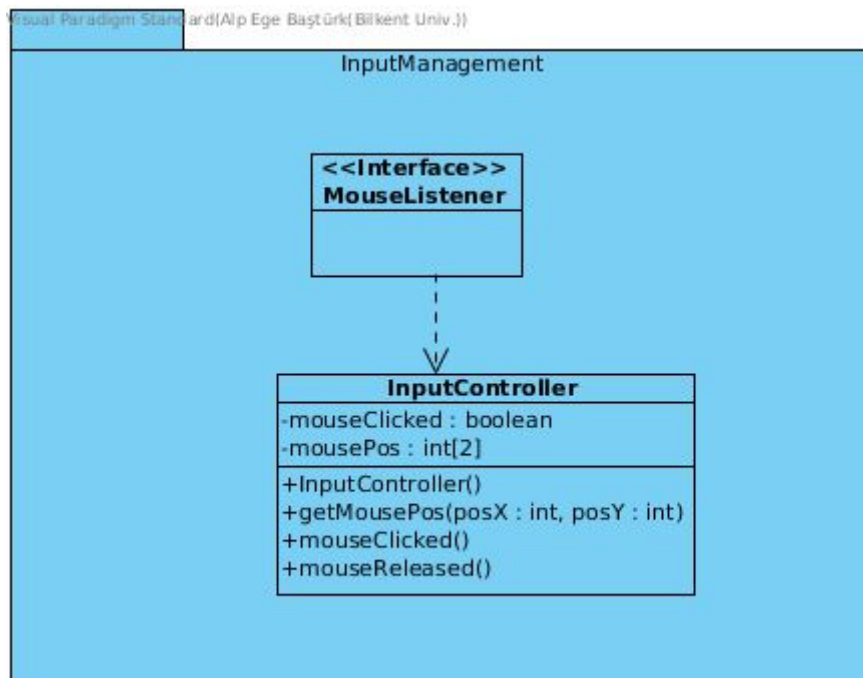
FileController Class



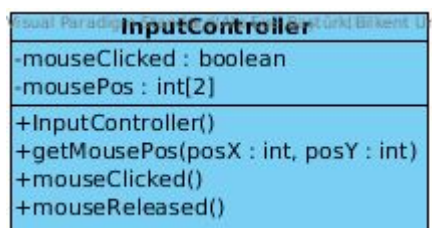
This class handles read and write operations. It will load images from the local storage by using the PATH. Furthermore, it will read all maps and scores. The GameController will request a map of a level by calling the readMap method with the level parameter. Also, it will write new scores to the storage when a level ends.

3.7 Input Management Subsystem

This subsystem handles inputs of the user. It receives inputs from the user and notifies other subsystems.



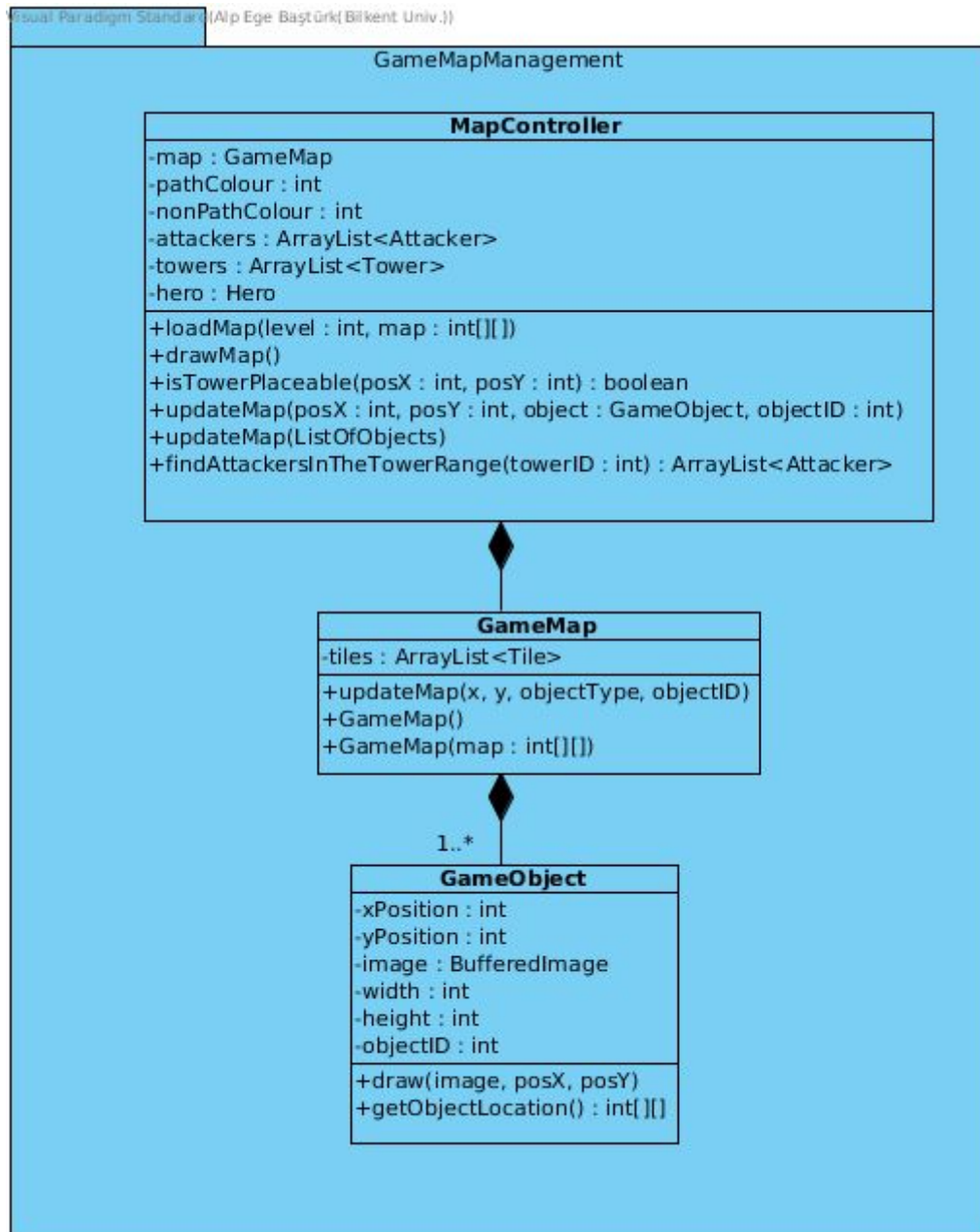
InputController Class



This class handles inputs given by the screen. Since we plan to have mouse only, it will only listen to mouse. It will notify other subsystems according to the mouse position and action.

3.8 Game Map Management Subsystem

This subsystem controls the map operations. It stores objects of the current level.



GameMap Class

GameMap
-tiles : ArrayList<Tile>
+updateMap(x, y, objectType, objectID)
+GameMap()
+GameMap(map : int[][])

This is the class which binds the 2-D array map to the game objects. It according to this map objects will be placed and operations will be done.

// TODO: Interactions are not complete. Should change according to the implementation.

MapController Class

MapController
-map : GameMap
-pathColour : int
-nonPathColour : int
-attackers : ArrayList<Attacker>
-towers : ArrayList<Tower>
-hero : Hero
+loadMap(level : int, map : int[][])
+drawMap()
+isTowerPlaceable(posX : int, posY : int) : boolean
+updateMap(posX : int, posY : int, object : GameObject, objectID : int)
+updateMap(ListOfObjects)
+findAttackersInTheTowerRange(towerID : int) : ArrayList<Attacker>

This class provides methods to operate on the map. It will store the lists of objects and the map. Map is a 2-D array. This controller will provide information to the GameController class by checking the map and objects. Main function of this class is determining the locations of objects and possible interactions.

// TODO: Expected to change according to implementation.

4.Low-level Design

4.1 Object Design Trade-Offs

Development Cost vs User Experience

We have decided to use Java for the implementation of the game. Java is easier to develop and maintain. We have decided to implement component and graphics with Java SWING libraries. This is because the development team is familiar with these libraries and it will be cheaper for us to develop. However JAVAFX libraries are better in terms of graphics, thus they would yield better results for the user.

Development Cost vs Reusability

We tried to make the project expandable. Our plan is to make additions to project easily. Additional levels and objects with different functionalities can be added to the project. We tried to find common parts of the game objects and use abstraction in order to make reusability easier. This increased the time spent on thinking about the project.

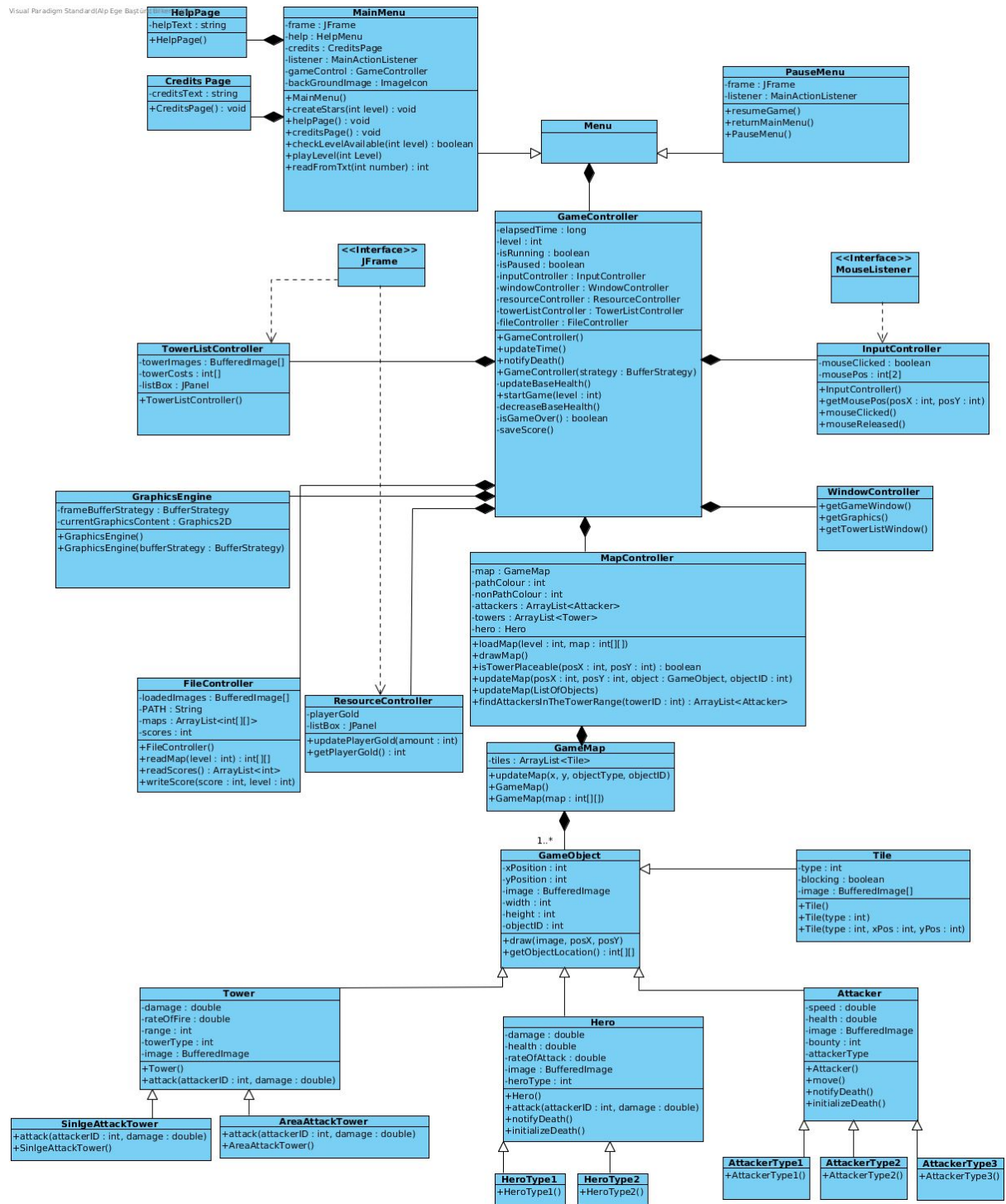
Functionality vs Understandability

The game will be easy to learn and understand as stated before. In order to do this we have made properties of game objects distinct. Users can understand the functioning of the towers and the gameplay without reading any documentation. Also mouse clicks will be the only input source in the game, thus user interactions will be simple.

User Experience vs Portability

We thought about using gpu acceleration with Java Volatile Images, however our initial research showed that this may reduce portability. Since there will be many moving objects in the game, gpu acceleration can increase the user experience and reduce cpu load. However we will try to implement this depending on the cost of development.

Visual Paradigm Standard (Alp Ege Bastı)



4.3 Packages

4.3.1 java.util

Contains ArrayList which will be used as a container for objects such as Attackers and Towers.

4.3.2 java.awt

Contains BorderLayout, Color and FlowLayout, which are used for the user interface of the game.

4.3.3 java.awt.event

Contains ActionEvent and ActionListener to handle events which come from java.awt components.

4.3.3 javax.swing

Contains ImageIcon, JButton, JFrame, JLabel and JPanel, which are used for the user interface of the game.

4.4 Class Interfaces

4.4.1 ActionListener

This interface will be used to check when actions occur. It will be implemented at the menu to check mouse inputs.

4.4.2 MouseListener

This interface is used to detect mouse actions. Since the game will be played with the mouse this will be the main source of inputs. It will be implemented at the menu to check mouse inputs.

5. References

1 Anon, (2017). [online] Available at:

<http://www.oracle.com/us/corporate/advertising/115m-java-3b-devices-2283055.pdf>

[Accessed 21 Oct. 2017].

2 bwired. (2017). *The Importance of Great User Interface (Plus 7 Commandments of a Great UI Design)*. [online] Available at:

<https://www.bwired.com.au/blogs/the-importance-of-user-interface-and-why-it-is-critical-to-your-success-online> [Accessed 21 Oct. 2017].