CS 319 - Object-Oriented Software Engineering
Design Report
Iteration 2

Defenders Of The Kingdom

Group 3-H

Alp Ege Baştürk

Barış Eymür

Emre Gürçay

Öykü Ece Ayaz

Table Of Contents

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 our 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

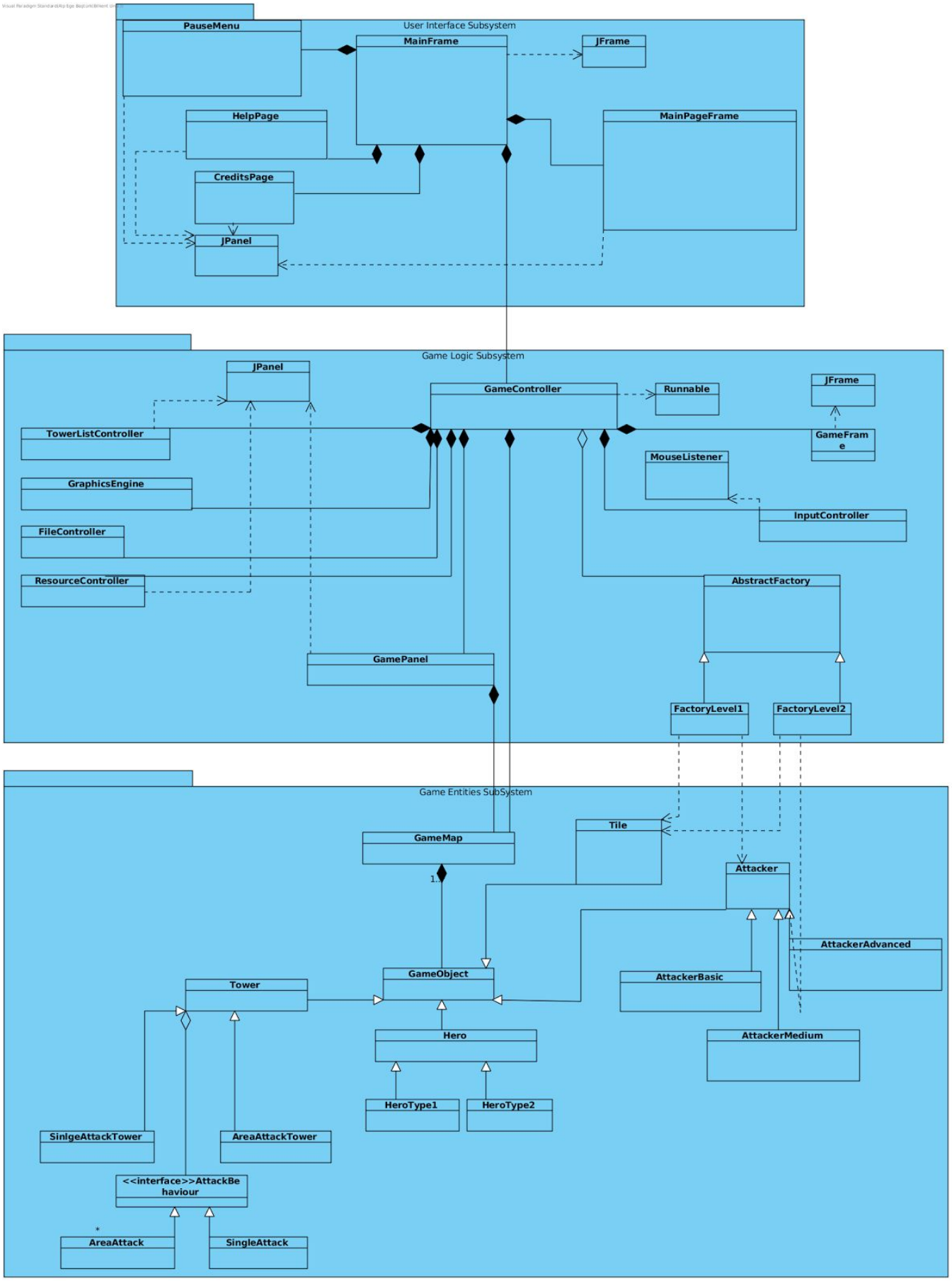
2.1 Subsystem Decomposition

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. Our main goal during the subsystem decomposition design is to achieve high coherence and low coupling.

There are three main subsystems which are User Interface, Game Logic and Game Entities. User Interface subsystem directly interacts with user and contains menu-related classes. Game Logic subsystem is a controller subsystem and it manages the game logic. Game Entities 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 (low coupling). 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, User Interface subsystem only directly interacts with the Game Controller class in the Game Logic Subsystem.

The decomposition of the subsystems and their relations are illustrated below.

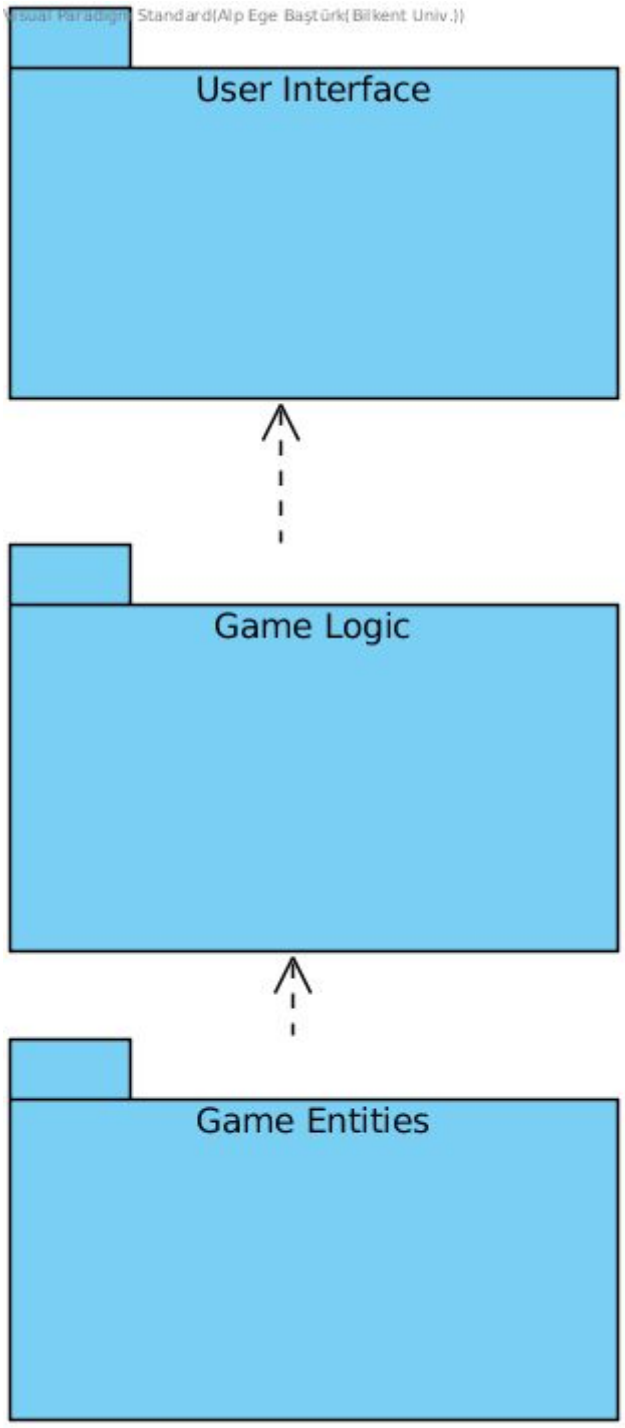

User Interface Subsystem
PauseMenu
MainFrame
JFrame
HelpPage
MainPageFrame
CreditsPage
JPanel
Game Logic Subsystem
JPanel
GameController
Runnable
JFrame
TowerListController
GameFrame
MouseListener
GraphicsEngine
InputController
FileController
ResourceController
AbstractFactory
GamePanel
FactoryLevel1
FactoryLevel2
Game Entities SubSystem
GameMap
Tile
Attacker
1..
AttackerAdvanced
GameObject
Tower
AttackerBasic
Hero
AttackerMedium
HeroType1
HeroType2
SinlgeAttackTower
AreaAttackTower
<<interface>>AttackBehaviour
*
AreaAttack
SingleAttack

2.2 Architectural Styles

2.2.1 Layers

The layers of our system has three main components: User Interface, Game Logic, Game Entities. 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 Logic. Game Logic 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 interact with each other. Our design includes concepts similar to MVC however we have violated some rules. For example there are panels in the Game Logic subsystem. GamePanel was designed to be in that part because it is directly related to the map where game will be drawn. It does not have any logic or control operations. We have decided not to create too many connections to separate the panel from the game elements. Similarly TowerListController and ResourceController are also panels. These controllers have required operations. Also there are images related to their operations. The drawing operations given to these panels could be given to the GamePanel however this would require extra operations, such as location calculations for position in the GamePanel, and extra message passing for updates which would reduce readability and maintainability. Another option was creating additional facade class between separated panels and logic elements however we have decided that this would create too much overhead compared to the size of the project. Since these classes are small and closely related to the game, they are close to the elements in the design.

The layer decomposition of the system is illustrated below.


Visual Paradigm Standard(Alp Ege Baştürk(Bilkent Univ.))
User Interface
Game Logic
Game Entities

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. Text files for the maps will be a matrix which will be read as a 2D array.

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.

In order to reach the files and their name in the code we will create a class called Assets. In this class we will write path names for each image so that whenever we need to reach those files we can easily write string name of the path rather than writing the whole path.

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. When executed images, map and score data will be read. User interface shows the main menu where player can select the level and start the game.

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. GameLogic subsystem can be terminated when player returns to main menu.

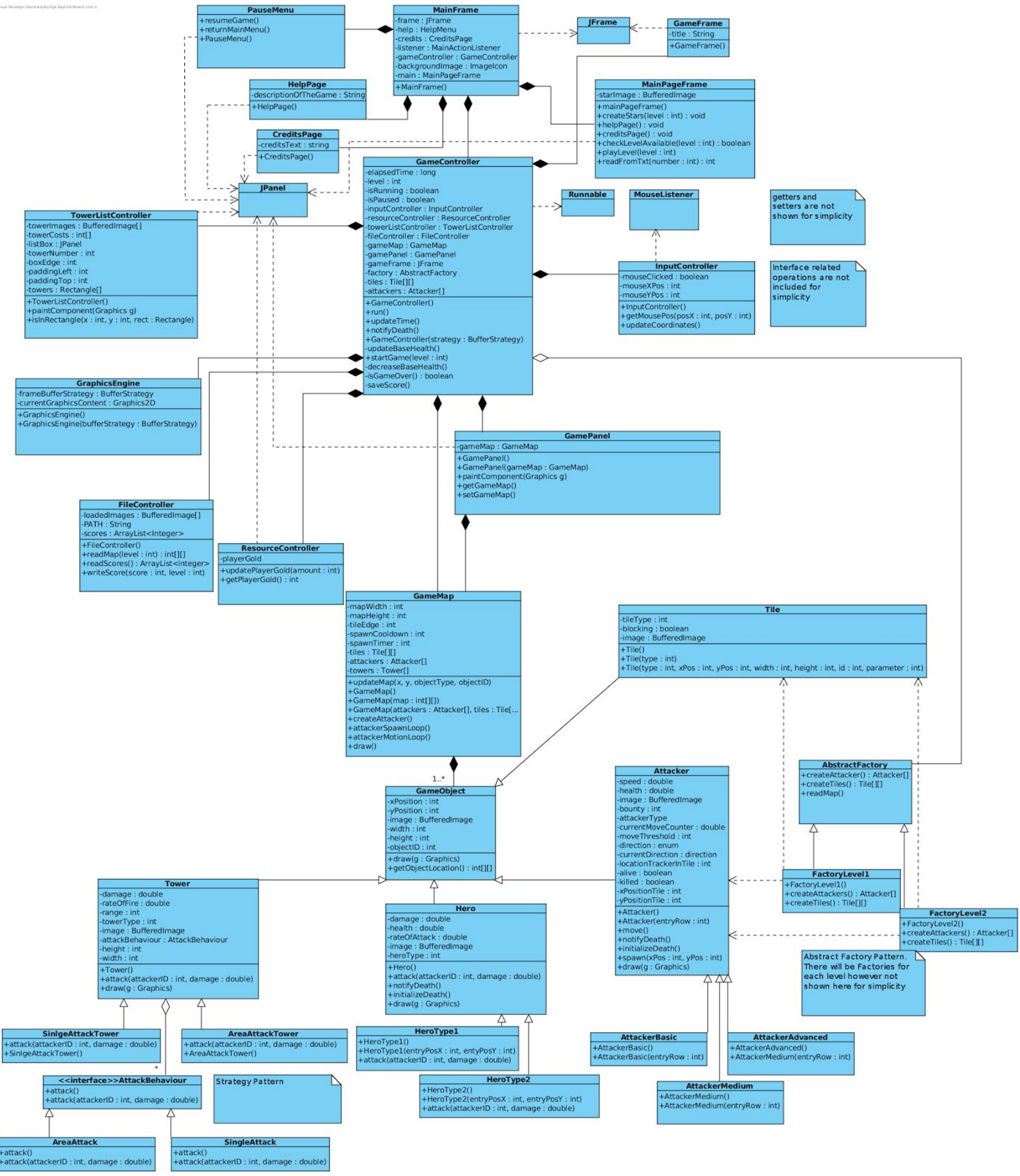
Error

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, strategy and abstract factory design patterns 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.

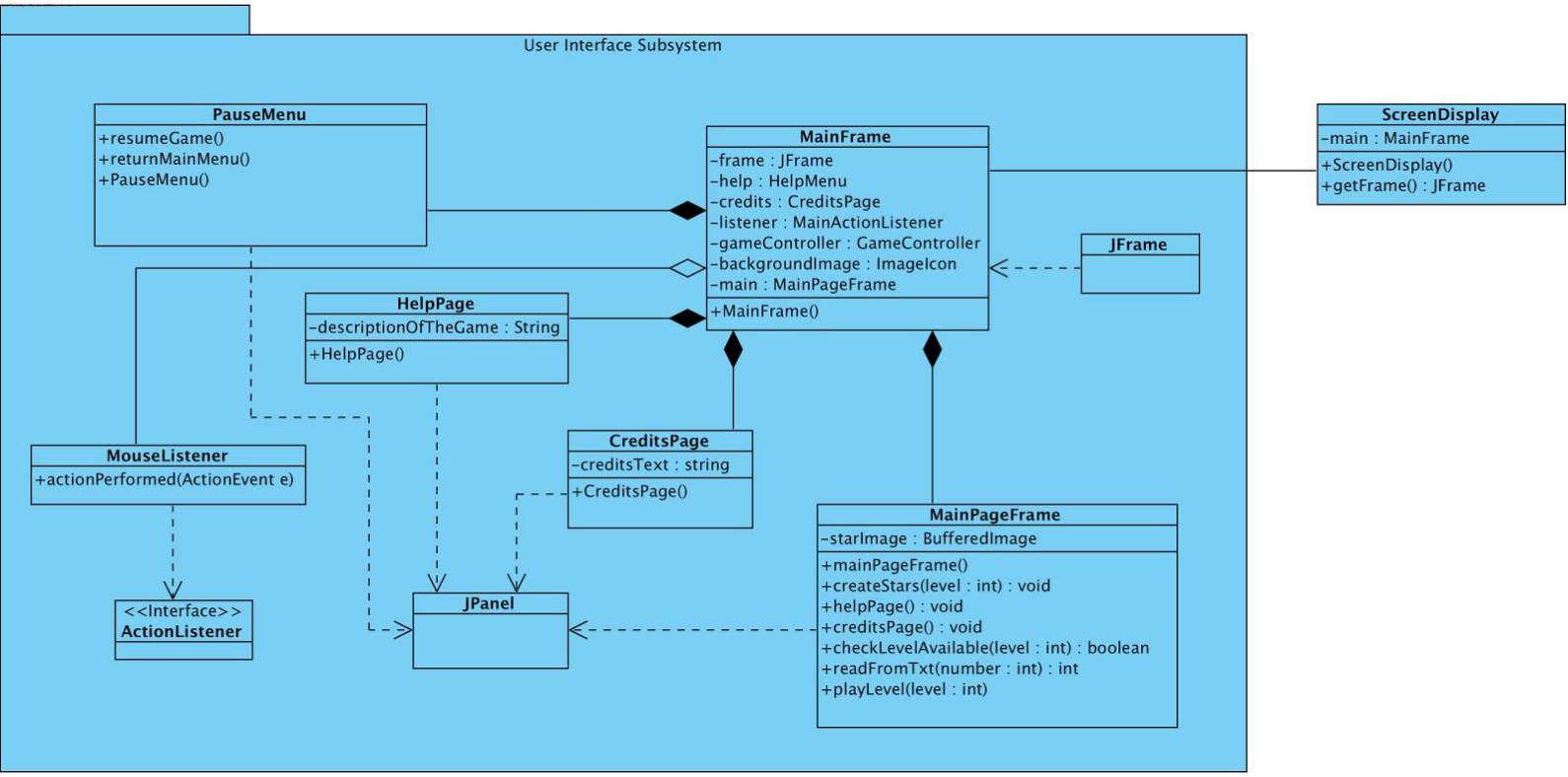
Strategy pattern is used for implementing different attack algorithms for tower attack operations, which are single attack and area attack. We have not changed the previous two tower type classes. They will implement different towers and can choose different algorithms while implementing. This will allow us to implement different algorithms easier. Another reason is that it makes us easier to change implementation similar to bridge pattern. An interface for attack behavior was implemented, and the two different attack classes were implemented using this interface.

Abstract factory pattern was used in order to create game objects for the map. In the first iteration this was done in the constructor of the GameController however it was making code difficult to read and maintain. Also current design makes implementing different levels easier in the future.

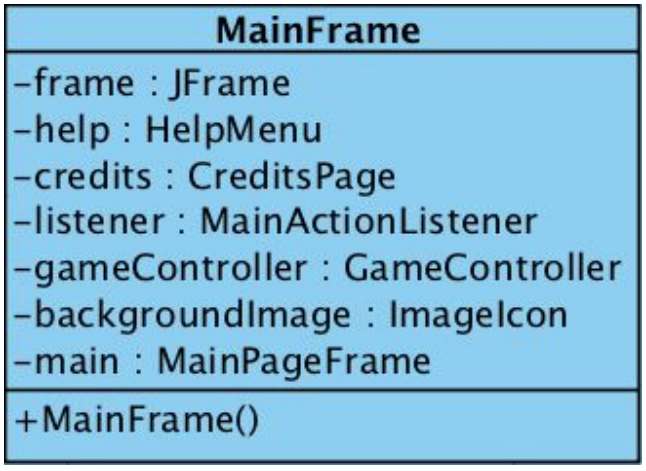
3.3 User Interface Subsystem

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.



3.3.1 MainFrame Class



MainFrame 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, JLabel..

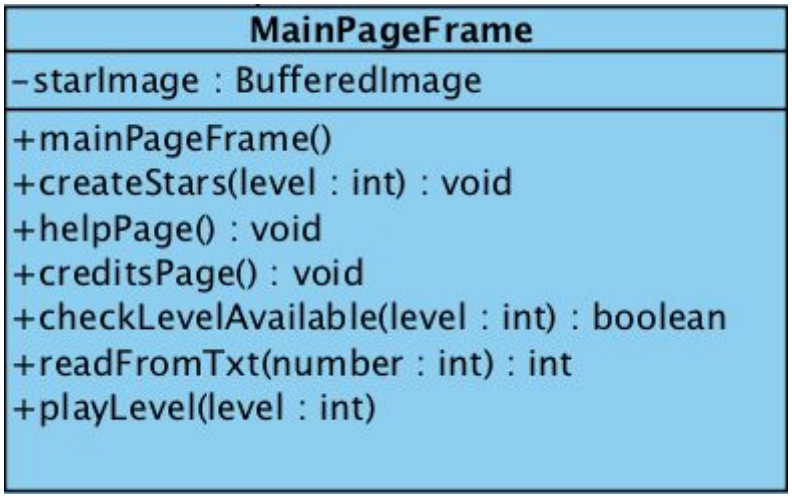
private CreditsPage credits: This is the panel of Help class where user can find contact informations about developers of the game. To create the panel, we will use JPanel.
This panel also uses components JButton, JLabel..

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

Constructors:

MainFrame(): MainFrame initialize the components of the class. Such as frame, JButton, JLabel, JPanel,listener, gameControl, credits and the background image of the user interface. It also provides the interaction between creditsPage, helpPage and mainmenu.

3.3.2 MainPageFrame Class



Attributes:

private BufferedImage starImage: This is the image of the stars. The number of the star images printed is depended on the success rate of the levels.

Constructors:

MainPageFrame: MainPageFrame 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 Level 1. 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.

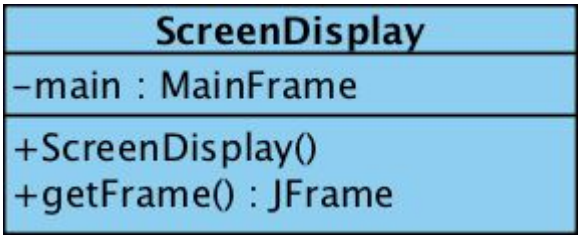
public createStars(int Level): This method creates stars for showing the success rate. To show the success rate it uses star image and displays them on the screen. To know about the success rate it calls the readFromTxt(int number) function and places them on a JPanel.

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.

3.3.3 ScreenDisplay Class



Attributes

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

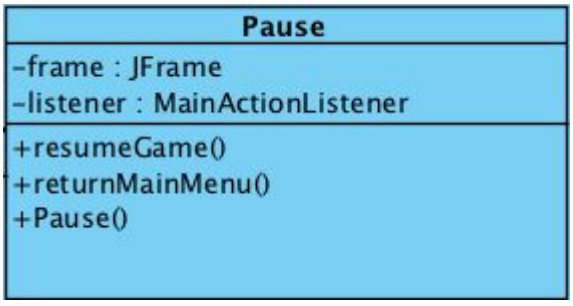
Constructors

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

Methods

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

3.3.4 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.

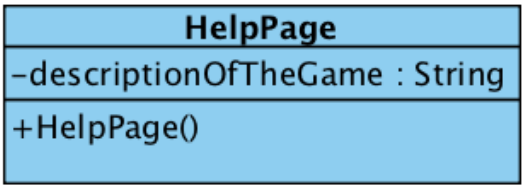
public void returnMainMenu(): This method returns the user to the main menu but when the user chooses to go to the main menu then it means (s)he did not successfully finish the level (s)he was playing.

MainActionListener

Method:

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

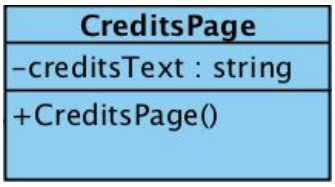
HelpPage



Attributes:

private String descriptionOfTheGame: This text block includes the description of the game and it is diplayed on the JLabel.

CreditsPage

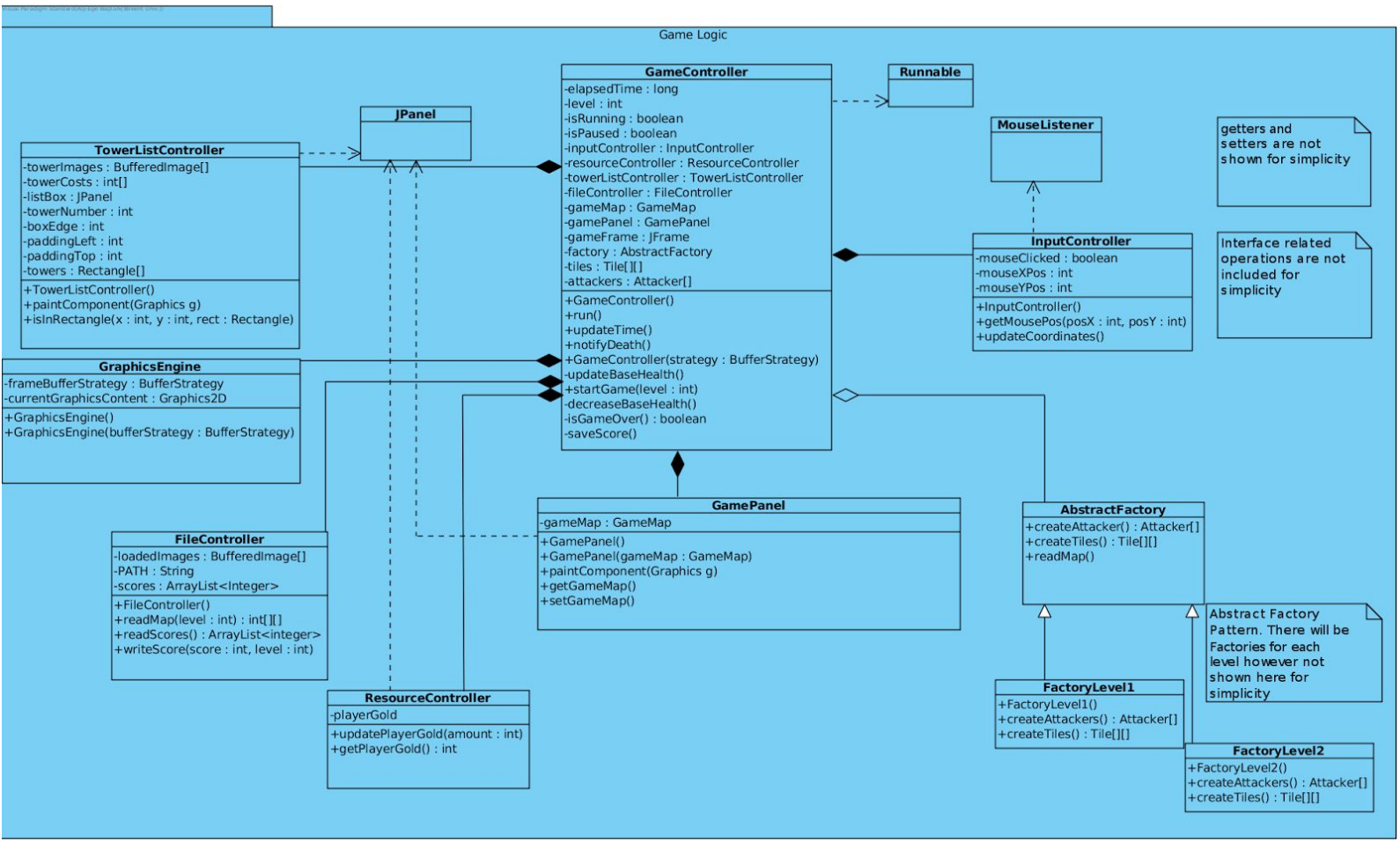


Attributes:

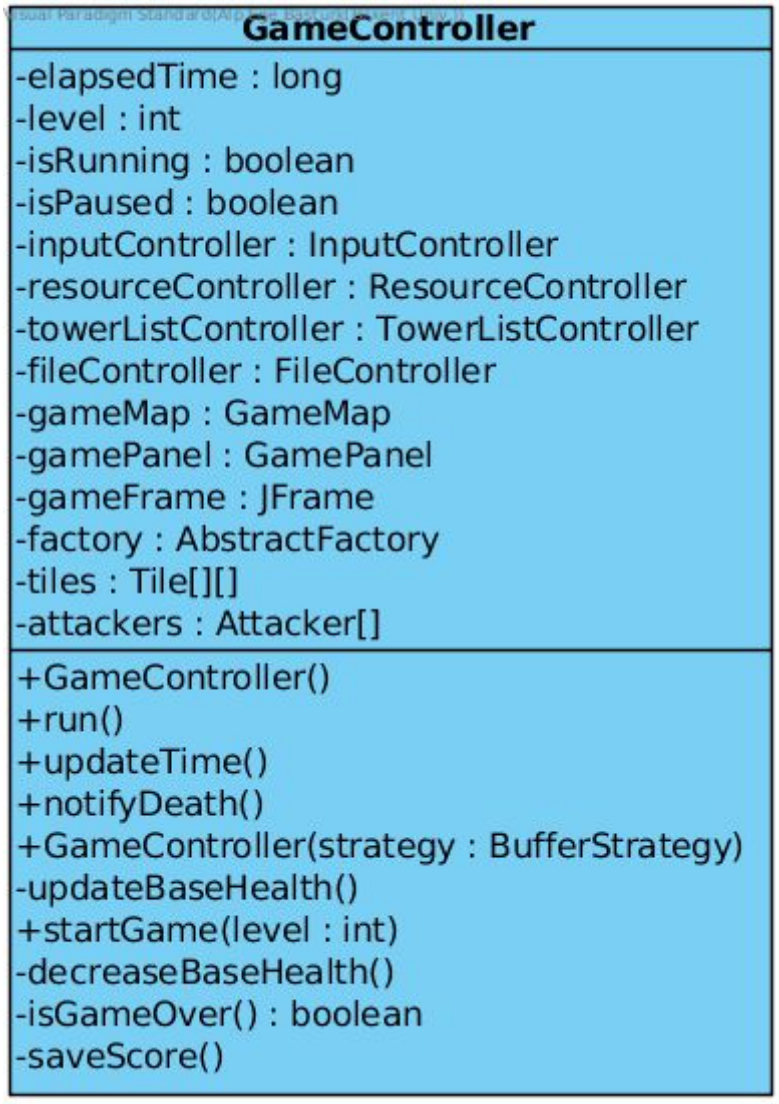
private String creditsText: This text block includes the credits about the game and it is diplayed on the JLabel.

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 Logic
GameController
-elapsedTime : long
-level : int
-isRunning : boolean
-isPaused : boolean
-inputController : InputController
-resourceController : ResourceController
-towerListController : TowerListController
-fileController : FileController
-gameMap : GameMap
-gamePanel : GamePanel
-gameFrame : JFrame
-factory : AbstractFactory
-tiles : Tile[][]
-attackers : Attacker[]
+GameController()
+run()
+updateTime()
+notifyDeath()
+GameController(strategy : BufferStrategy)
-updateBaseHealth()
+startGame(level : int)
-decreaseBaseHealth()
-isGameOver() : boolean
-saveScore()
Runnable
JPanel
MouseListener
getters and setters are not shown for simplicity
TowerListController
-towerImages : BufferedImage[]
-towerCosts : int[]
-listBox : JPanel
-towerNumber : int
-boxEdge : int
-paddingLeft : int
-paddingTop : int
-towers : Rectangle[]
+TowerListController()
+paintComponent(Graphics g)
+isInRectangle(x : int, y : int, rect : Rectangle)
InputController
-mouseClicked : boolean
-mouseXPos : int
-mouseYPos : int
+InputController()
+getMousePos(posX : int, posY : int)
+updateCoordinates()
Interface related operations are not included for simplicity
GraphicsEngine
-frameBufferStrategy : BufferStrategy
-currentGraphicsContent : Graphics2D
+GraphicsEngine()
+GraphicsEngine(bufferStrategy : BufferStrategy)
GamePanel
-gameMap : GameMap
+GamePanel()
+GamePanel(gameMap : GameMap)
+paintComponent(Graphics g)
+getGameMap()
+setGameMap()
AbstractFactory
+createAttacker() : Attacker[]
+createTiles() : Tile[][]
+readMap()
Abstract Factory Pattern. There will be Factories for each level however not shown here for simplicity
FileController
-loadedImages : BufferedImage[]
-PATH : String
-scores : ArrayList<Integer>
+FileController()
+readMap(level : int) : int[][]
+readScores() : ArrayList<integer>
+writeScore(score : int, level : int)
ResourceController
-playerGold
+updatePlayerGold(amount : int)
+getPlayerGold() : int
FactoryLevel1
+FactoryLevel1()
+createAttackers() : Attacker[]
+createTiles() : Tile[][]
FactoryLevel2
+FactoryLevel2()
+createAttackers() : Attacker[]
+createTiles() : Tile[][]

3.4.1 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 list controllers. It creates game panel, frame and input controller and binds input controller. It calls factories to create game objects. Later it provides information and objects to be drawn on the screen, to the GamePanel. It starts the game loop and calls update methods of the game objects and calls repaint method of the panel to show changes on the screen. If isGameOver() returns true it checks the score display success or fail and saves score for that

level by calling input. Also this class handles the movement and attack of the objects on the map 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 ResourceController resourceController: Stores resource controller instance
private TowerListController towerListController: This is a controller to control the list of towers. It also works as the panel where towerlist is shown. Input controller is also binded to this panel and detects inputs.
private FileController fileController: Stores file controller instance to read files.
private GameMap gameMap: This is the game map instance which stores game objects. GameController creates this and passes it as a parameter to the GamePanel which calls draw method by using Graphics object g to properly fit graphics in the panel.
private GamePanel gamePanel: This is the main panel instance where map is drawn.
private Frame gameFrame: This is the frame instance which holds all panels together.
private AbstractFactory Factory: This is a factory instance. It has type of AbstractFactory however actual factory instance is determined at runtime.
private Tile[][] tiles: This attribute is used to store tiles of the map which is a 2D array.
private Attacker attackers: This attribute is used to store the attackers as a 1D array.

Constructors:
GameController(): It initializes the components and attributes of the class GameController.
GameController(BufferStrategy strategy): Constructor with passed strategy parameter

Methods:

public void run(): This method allows running game loop as a separate thread.

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

public void notifyDeath(): This method notifies related classes when an attacker has been killed.

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

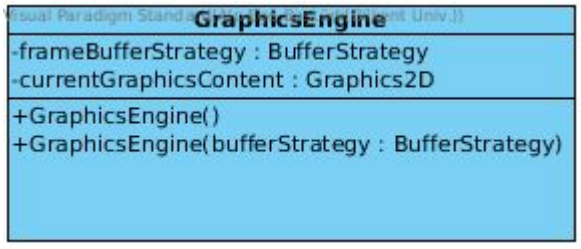
public void startGame(int level): This method starts the level of the game which is taken as a parameter.

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.

3.4.2 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:

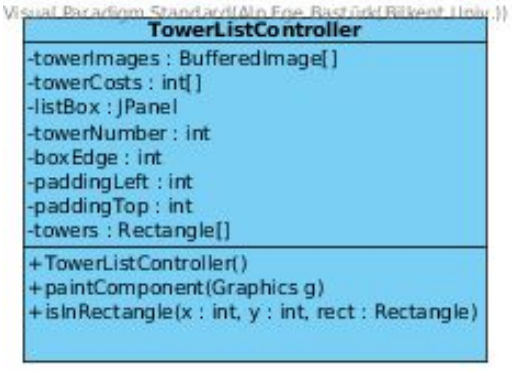
private BufferStrategy frameBufferStrategy: This will be used to define buffer strategy for rendering the images.

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

3.4.3 TowerListController Class



This subsystem is used for the tower list where player will select towers. It displays the view and controls buy/sell operations of the player.

Attributes:
private BufferedImage[ ] towerImages: It is an array which holds tower images.
private int[ ] towerCosts: It is an array which holds the costs of towers.
private int towerNumber: It holds the number of tower types.
private int boxEdge: It is the length of an edge of the box which will hold the tower image in it, in the list of available towers to buy.
private int paddingLeft: It determines the padding from left of a box in the panel.
private int paddingTop: It determines the padding from top of a box in the panel.
private Rectangle[ ] towers: It is an array which holds rectangles which will be drawn for different tower types as its elements.

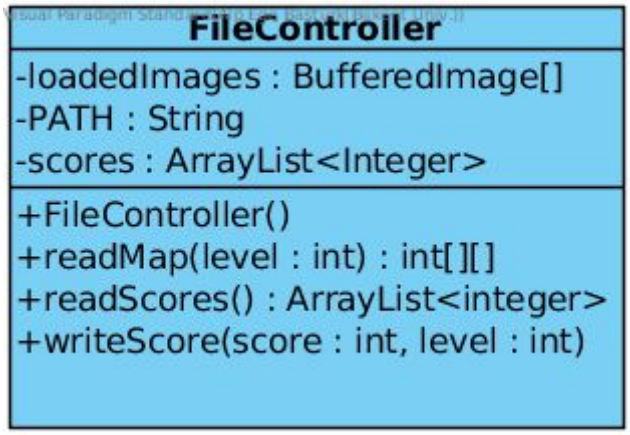
Constructors:

public TowerListController(): It is the default constructor for TowerListController class.

Methods:

public boolean isInRectangle(int x, int y, Rectangle rect): It returns true if the given point is in the specified rectangle, returns false if it is not.

3.4.4 FileController Class



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.

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

Reading the map from the FileController was given to the AbstractFactory because it was not reused in other places.

Attributes:

public BufferedImage[ ] loadedImages: This image array will store the images which are loaded.

public final static String PATH: This attribute is the relative path of the files which will be read.

public int scores: This attribute used to store scores read from the file.

Constructor:

public FileController(): It is the default constructor for FilecOntroller class.

Methods:

public int[ ][ ] readMap(int level): According to the level number, it returns the map of that level as a 2-D integer array.

public ArrayList<integer> readScores(): It returns the player's success rates from different levels as an integer ArrayList. It reads the success rates from a text file.

public void writeScore(int score, int level): It writes the success rate of a current level to a text file which can be read later by the method readScores().

3.4.5 ResourceController Class

This class manages the gold resources of the player.

Attributes:

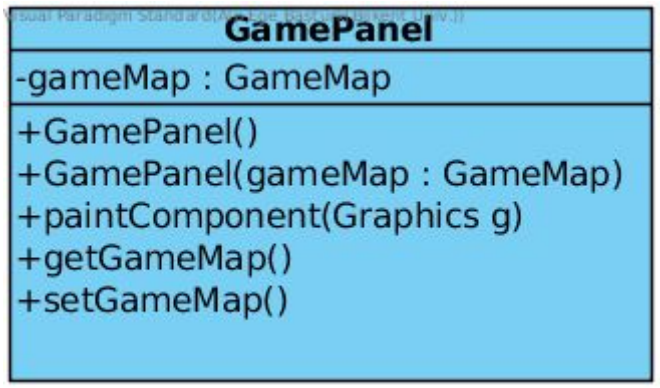
private int playerGold: It holds the amount of the gold resource of the player.

Methods:

public void updatePlayerGold(int amount): It updates the changes in the amount of gold.

public int getPlayerGold: It returns the amount of the player's gold.

3.4.6 GamePanel Class



This class manages the game panel during gameplay. The panel calls draw(Graphics g) method of the current GameMap instance passed to the panel as a parameter by the GameController.

Attributes:

private GameMap gameMap: It holds the Game Map object passed from the GameController.

Constructor:

GamePanel(): It initializes the components of the GamePanel object.

GamePanel(GameMap gameMap): It initializes the components of the GamePanel object and sets game map passed by the GameController.

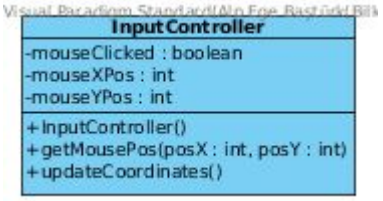
Methods:

public void paintComponent(Graphics g): Paint method to draw the panel and map.

public void getGameMap(): It returns the Game Map object of the game.

public void setGameMap(): It sets the Game Map object of the game.

3.4.7 InputController Class



Input Controller class handles the inputs of the user. It receives inputs from the user and notifies Game Controller class. Since we plan to have only mouse, it will only listen to the mouse. It will notify other subsystems according to the mouse position and action.

Attributes

private boolean mouseClicked: It holds the boolean value of whether the mouse is clicked or not.

private int mouseXPos: It holds the x- axis position of the mouse.

private int mouseYPos: It holds the y- axis position of the mouse.

Constructor

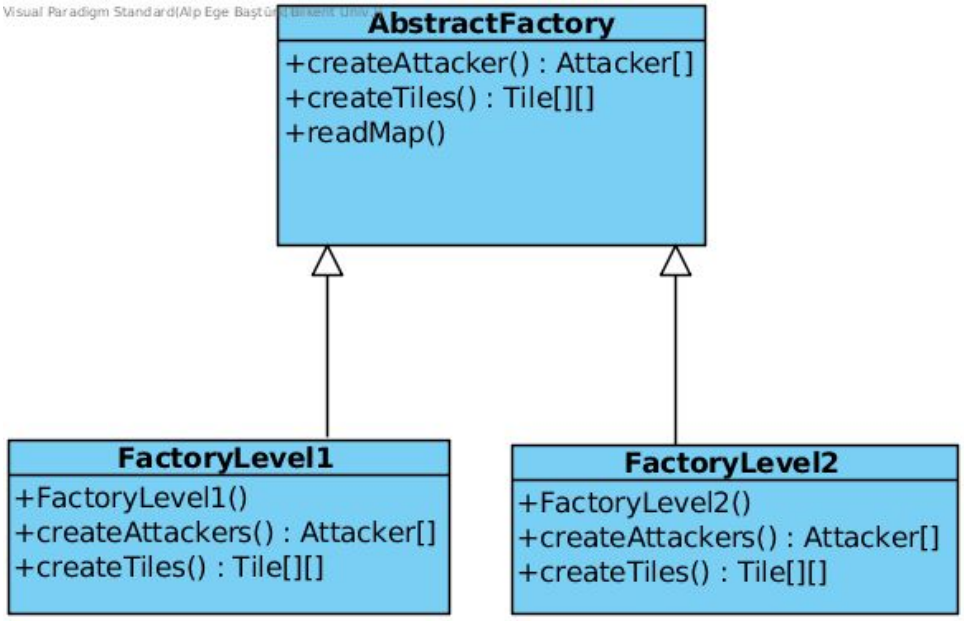
InputController(): It is the default constructor of the input controller.

Methods

public void getMousePos(int PosX, int PosY): It gets the x and y positions of the mouse.

public void updateCoordinates(): It updates the coordinates of the mouse.

3.4.8 AbstractFactory Class & Factories



AbstractFactory

AbstractFactory is the generalization of Factories for levels. A factory reads the map from a 2D matrix file and creates the Tiles for the game map also creates attackers depending on the level. There will be different factories for different levels. This implementation was planned to increase extensibility in the future.

Methods

public Attacker[] CreateAttacker(): Abstract call to create and return attackers array.

public Tile[][] createTiles(): Abstract call to create and return tiles array filled with Tile objects.

FactoryLevel<i>

Constructor:

FactoryLevel<i>(): Default constructor for a factory for level *i.*

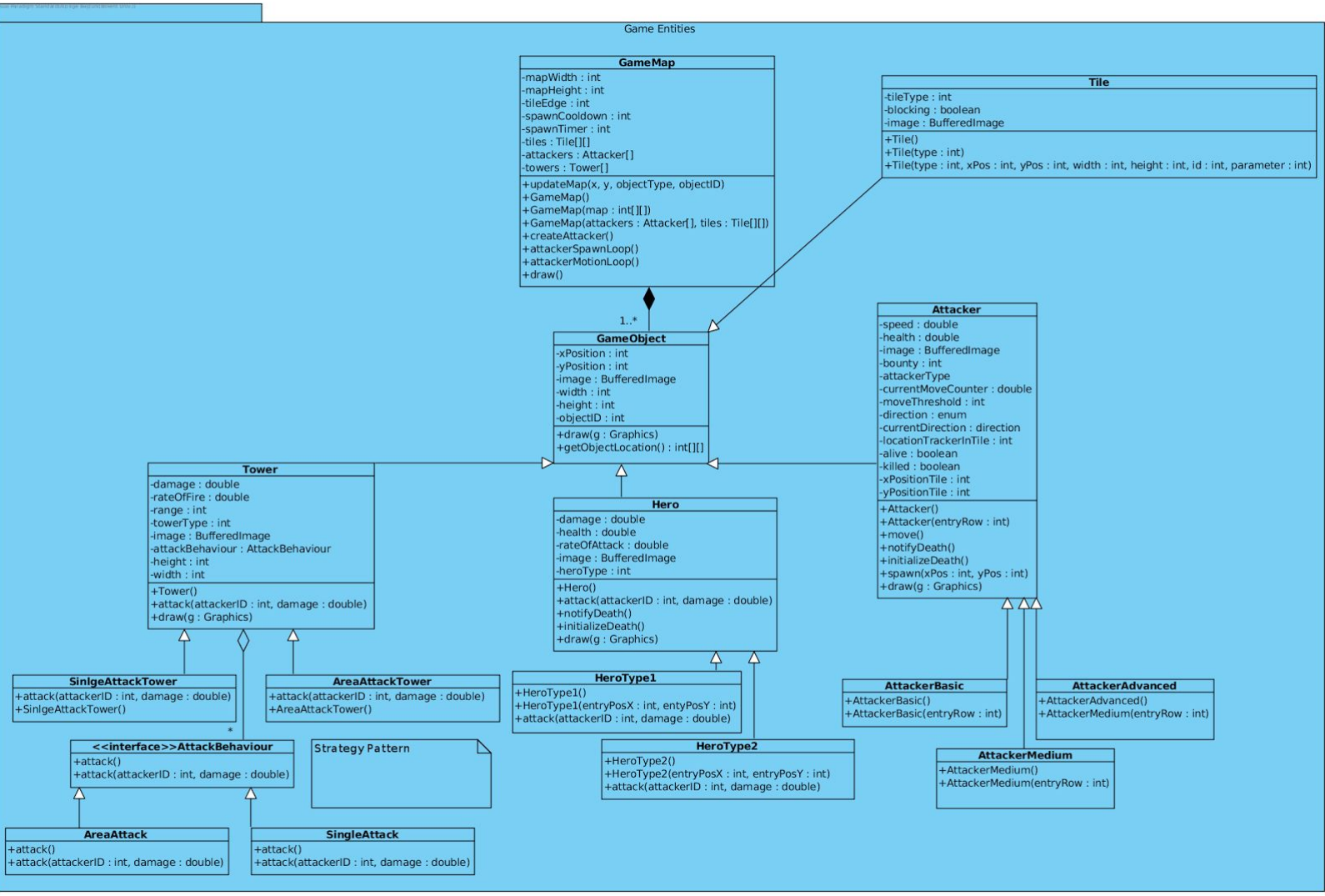
Methods:

public Attacker[] CreateAttacker(): Implementation of the abstract call.

public Tile[ ][ ] createTiles(): Implementation of the abstract call.

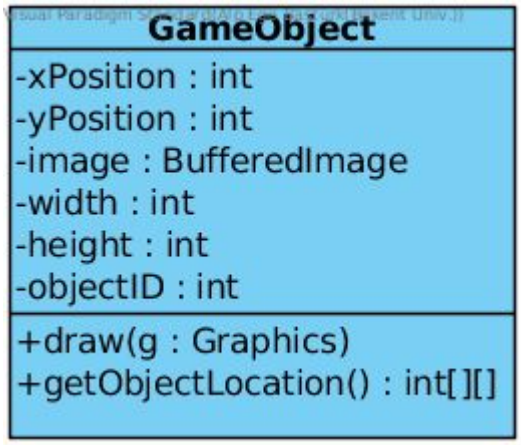
3.5 Game Entities Subsystem

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”.



3.5.1 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.

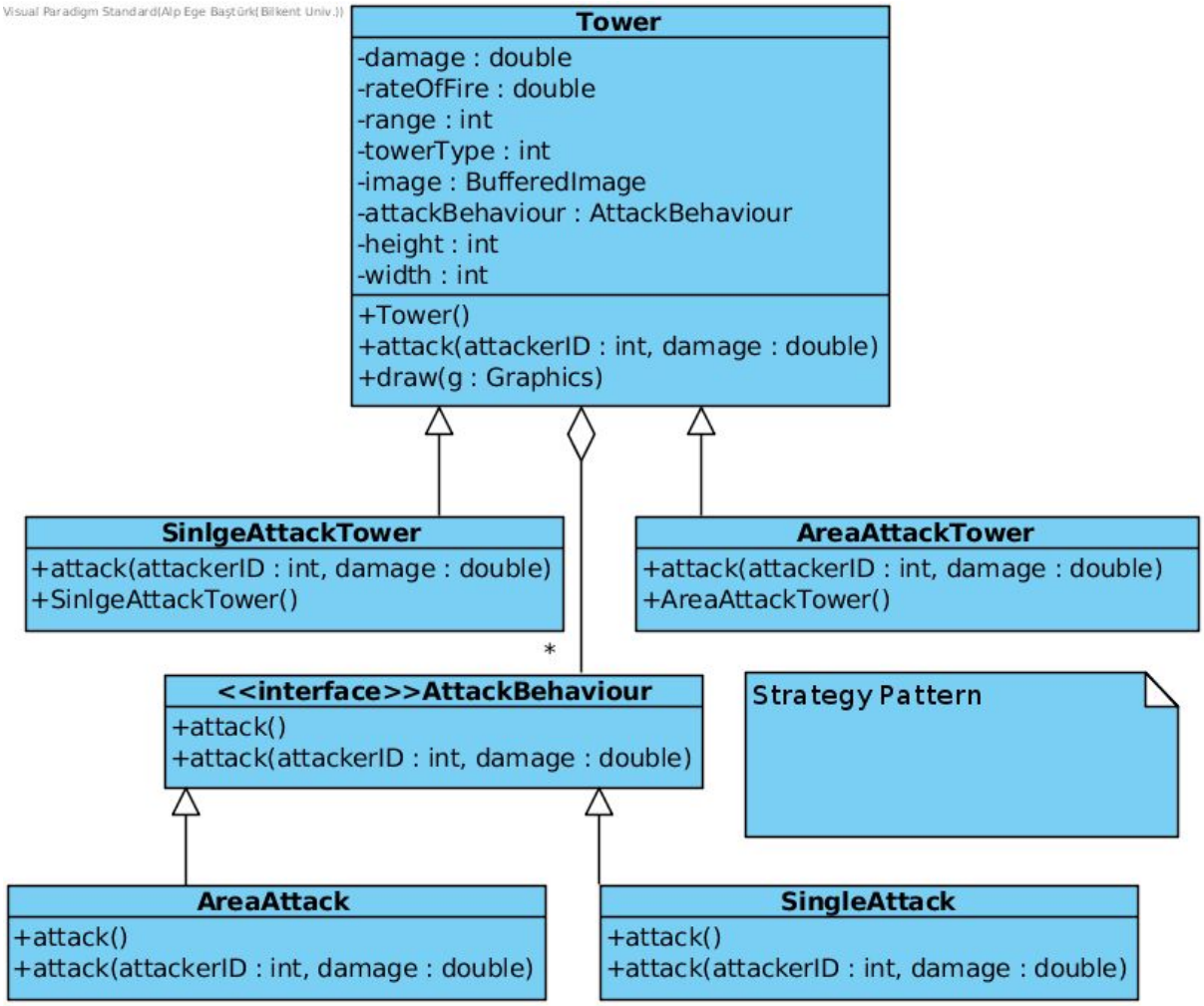
private int objectID: Each GameObject will have a unique ID, which will be stored as objectID.

Methods:

public void draw(Graphics g): This method draws a gameObject according to Graphics object. Graphics object is planned to be passed from the Panel to objects.

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

3.5.2 Tower Classes & Related Classes



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 have different types of attack. These towers will have different attack methods. Graphics for the attack and mechanisms of the attack will be different. Different tower types could be used to implement different attack strategies however this would result in embedded algorithms. Strategy pattern was used to manage changes. Also it makes it easier to develop and change algorithms in the future. Strategy pattern was used to implement different attack algorithms for different tower types.

Tower:

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.

private AttackBehaviour attackBehaviour: Attribute to determine the attack algorithm to be called.

private int height: Height value of the tower image.

private int width: Width value of the tower image.

Constructors

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

Methods

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

Public void draw(Graphics g): Draw method of the towers.

SingleAttackTower:

Constructors

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

Methods

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

AreaAttackTower:

Constructors

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

Methods

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

AttackBehaviour:

Methods

Public void attack(): default attack method.

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

AreaAttack:

Methods

Public void attack(): default attack method.

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

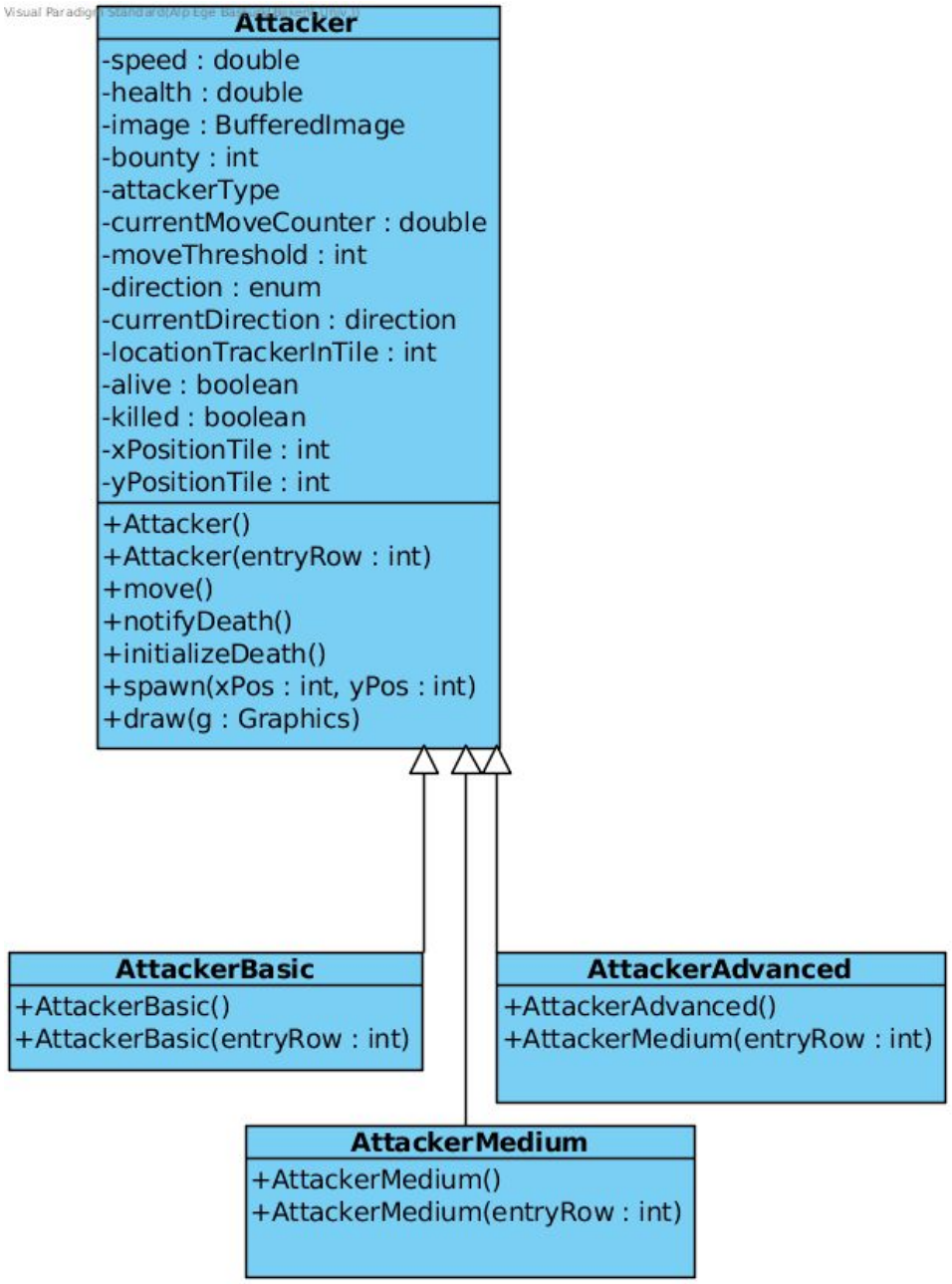
SingleAttack:

Methods

Public void attack(): default attack method.

public void attack(int attackerID, double damage): This method attacks the attacker and decreases the health of the attacker.

3.5.3 Attacker Classes



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.

Private double currentMoveCounter: This attribute counts the location of the attacker oject.

Private int moveThreshold: This attribute is the threshold to initiate move. When currentMoveCounter passes this value attacker object updates its location by one pixel in the current direction.

Private enum Direction: enumeration of four directions, *Left, Right, Up, Down.*

Private Direction currentDirection: current direction of the attacker object.

Private int locationTrackerInTile: Tracks the location of the attacker in a tile. Starts from 0 to tile edge. If it passes the tile edge it means object has entered the next tile in its direction.

Private boolean killed: Attribute to determine if object was killed. Used to differentiate attackers killed and not became alive to determine the spawn.

Private boolean alive: Attribute to determine if object is alive.

Private int xPositionTile: x position of the attacker in the current Tile.

Private int yPositionTile: y position of the attacker in the current Tile.

Constructor

Attacker(): It initializes the attacker components.

Attacker(int entryRow): It initializes the attacker components and sets attacker at the given row in the map.

Methods:

public void move(): It moves the attacker.

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

public void initializeDeath(): Starts operations related to death.

Public void spawn(xPos:int, yPos: int): Spawns object in the given tile if it was not created or created and killed before.

Public void draw(Graphics g): Draws the object according to given Graphics object.

AttackerBasic:

Constructor

AttackerBasic(): It initializes the attacker components.

AttackerBasic(int entryRow): It initializes the attacker components and sets attacker at the given row in the map.

AttackerMedium:

Constructor

AttackerMedium(): It initializes the attacker components.

AttackerMedium(int entryRow): It initializes the attacker components and sets attacker at the given row in the map.

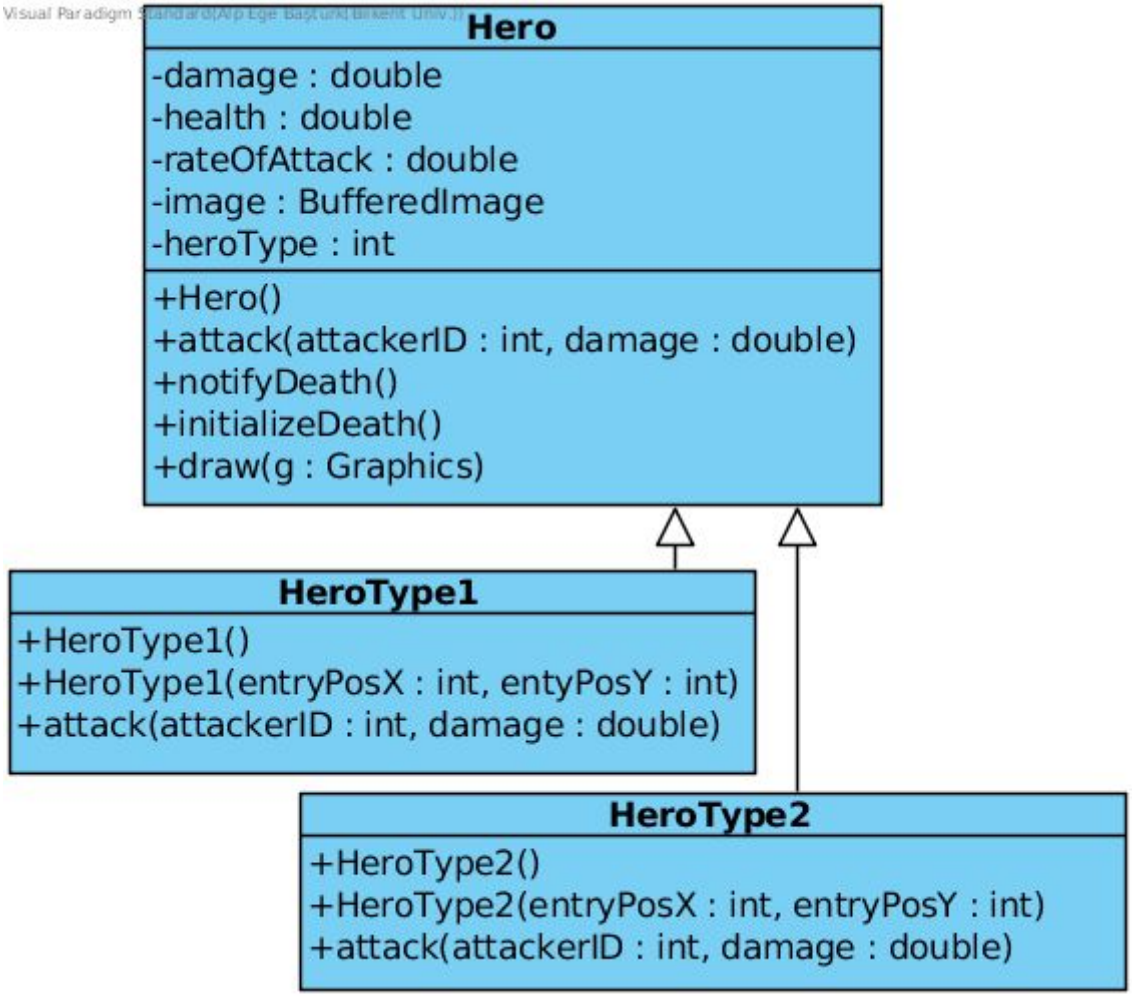
AttackerAdvanced:

Constructor

AttackerAdvanced(): It initializes the attacker components.

AttackerAdvanced(int entryRow): It initializes the attacker components and sets attacker at the given row in the map.

3.5.4 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.

Hero:

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 heroType: This is the type of the hero.

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.

Public void draw(Graphics g): Object draw method according to given Graphics object.

HeroType1:

Constructors:

public HeroType1(): It initializes the components of hero.

public HeroType1(int entryPosX, int entryPosY): It initializes the components of hero and sets its position.

Methods

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

HeroType2:

Constructors:

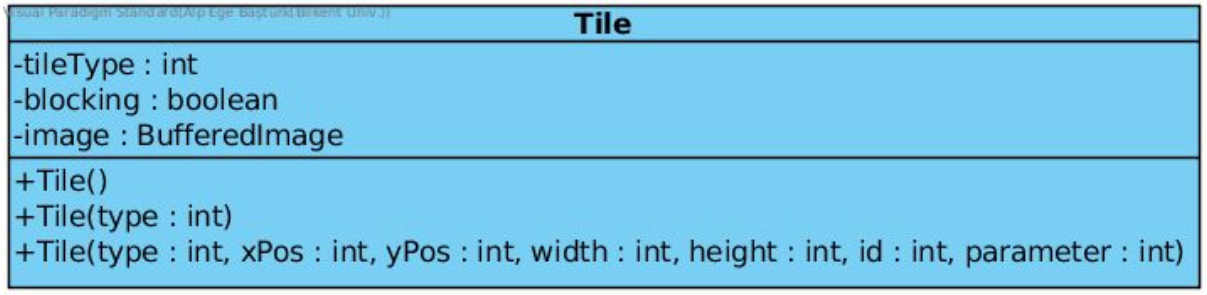
public HeroType2(): It initializes the components of hero.

public HeroType2(int entryPosX, int entryPosY): It initializes the components of hero and sets its position.

Methods

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

3.5.5 Class Tile



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..

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.

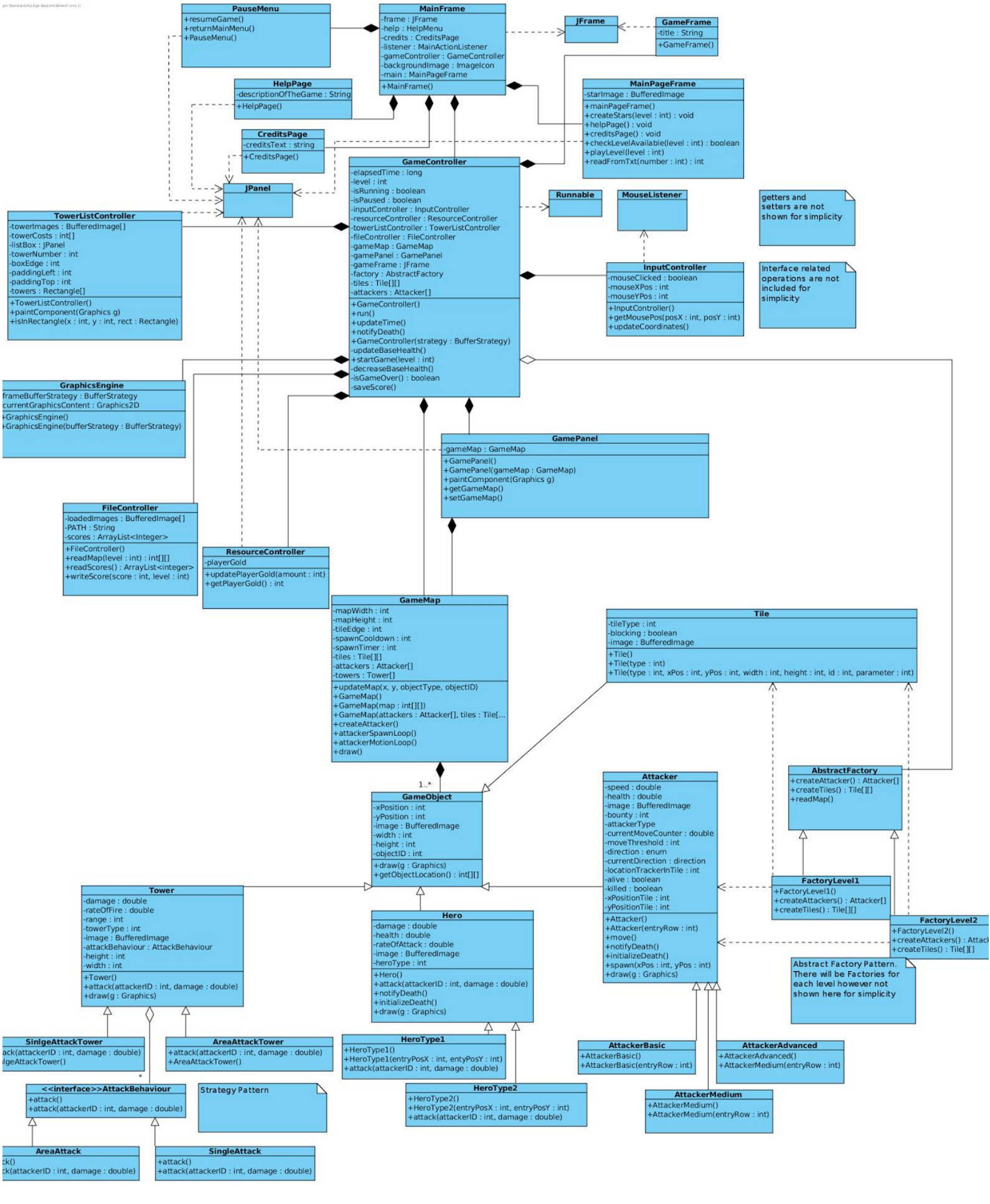
Functionality vs Understandability

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.

4.2 Final Object Design



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.

4.3.4 javax.swing

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

4.3.5 User Interface

Contains user interface classes written for the project.

4.3.6 Game Logic

Contains game logic classes written for the project.

4.3.7 Game Entities

Contains game entity classes written for the project.

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.

4.4.3 Runnable

This interface is implemented by GameController class to make game loop run as a thread.

5. Improvement Summary

During the second iteration process of the project, we made some changes as well as improvements to the project. We had changes in our class diagram. While implementing the project for the second iteration, we did some minor changes to our main menu. For example rather than extending a main menu class, we used pages of each menu object in the MainFrame which resulted in a smoother transition.

We have also added two new design patterns to our object design. We used Strategy design pattern for implementation of different attack types for different types of towers. We have also used Abstract Factory design pattern for creation of game objects. We think that this will help us implement different levels easily in the future. This lead us to change the initial design with the FileController. In the current design, we read map matrix from the Factory and FileController is used for score operations.

Furthermore, MapController class was changed to GamePanel which does not have any control and logic operations. In the current design, GameController creates GameObjects with the help of the Factory and passes them to GameMap in the constructor while creating it. Later GameMap instance is passed to the GamePanel. GameController calls repaint method of the GamePanel in the game loop which calls draw method of the GameMap instance with Graphics object g. GameMap calls draw methods of all game objects. This improvement was a result of decentralization of control. Updates related to operations like motion and collision detection were implemented in the objects themselves. This reduced the control functionality in GameController and removed the ones in the MapController.

In addition, ResourceController and TowerListController implemented the panel which they operate on. In the initial design, display functionality of these classes was planned to be implemented in the main game panel. However, implementing as separate panels made views more isolated and calculations related to positions of the objects on the panel simpler. Also different panels can be added to the game frame without requiring change to other panels.

We have updated the subsystem decomposition of the system. In the updated version, there are three subsystems which are lowly coupled. We have included the Game Map Management subsystem into the Game Logic subsystem since they are closely related.

6. 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].