



**CS 319 - Object-Oriented Software Engineering
Design Report**

Second Iteration

Instructor: Eray Tüzün

Project Name: The Wall

Group 3E

Erdem Ege Maraşlı - 21602156

Ayça Begüm Taşcıoğlu - 21600907

Alperen Koca - 21502810

Hammad Malik – 21600468

Ensar Kaya - 21502089

Contents

| | |
|----------------------------------|----|
| 1. Introduction | 3 |
| 1.1 Purpose of the system | 3 |
| 1.2 Design Goals | 3 |
| End User Criteria | 4 |
| Maintenance Criteria | 4 |
| Performance Criteria | 5 |
| Trade Offs | 5 |
| 2. High Level Architecture | 6 |
| 2.1 Subsystem Decomposition | 6 |
| 2.2 Architectural Styles | 8 |
| 2.3 Design Decisions | 8 |
| Hardware / Software Mapping | 8 |
| Deployment Diagram | 10 |
| Persistent Data Management | 10 |
| Access Control and Security | 10 |
| Boundary Conditions | 11 |
| 3. Low-level Object Model | 13 |
| 3.1 Presentation Layer Subsystem | 13 |
| 3.2 Business Layer Subsystem | 20 |
| GameController Subpackage | 21 |
| GameEntities Subpackage | 27 |
| 3.3 Data Layer Subsystem | 44 |
| 4.Improvement Summary | 48 |

1.Introduction

1.1 Purpose of the system

Our aim is to develop a 2D strategy game named "The Wall", which is inspired of the board game "Walls & Warriors". The main objective of the game is splitting friendly and enemy units by using the designated walls¹. The units will mostly be regular knights. However, there will also be some additional units with additional features. The designated walls are same as the walls in "Walls & Warriors" and players are expected to use mouse to drag these walls to the game board and split the enemy and friendly units. Basically, we add some new features and play modes into the Walls & Warriors game such as changing the icons of game objects, challenge mode which is a play mode with a time limit and 3 different map from different difficulty levels, campaign mode which is a play mode with a storyline aims to rescue the Princess, developer mode which is a play mode with 2 different options such as creating a brand new map or play a map made by different users. The game will be a desktop application and can be played by using mouse merely. This report consists of an overview of "The Wall", description of gameplay and game objects. Then, the report specifies functional and non-functional requirements. The system models namely use case, dynamic, object class and sequential diagrams will also be presented. In addition, there will be some example screen mock-ups.²

1.2 Design Goals

It's important to show up design goals of the system in order to clarify the quantities which we are going to focus on during this project. We explained most of our system's nonfunctional requirements in the analysis stage. Important design goals of our system are described below.

¹"Walls & Warriors." *SmartGames*, Smart NV, www.smartgames.eu/uk/one-player-games/walls-warriors.

²"Walls & Warriors." *SmartGames*, Smart NV, www.smartgames.eu/uk/one-player-games/walls-warriors.

End User Criteria:

Ease of Using & Learning:

The User-Interface will be designed and implemented in such a manner that the game can be understood and played easily as well as stimulate interest in the user. The mechanics will be general, similar to those seen in many online board-games. The main menu will have less than 8 buttons and each of them will be understandable. There is also a "How to play" section where users can get help. The game also offers hints to the users while gameplay to make the game comprehensible. We are planning to conduct a survey to detect the understandability of the game among the possible user profiles. For example, a person who is between 12-60 ages can understand the user interface functional at most in 10 minutes.

Maintenance Criteria:

Extensibility:

When creating any software, extendibility and reusability are always important considerations. As it is often observed that there are continuous updates to all famous applications. Thus, "The Wall" will also be implemented such that it can be extended and updated with time according to the needs and feedbacks received by users.

Portability:

Portability is a crucial thing for a software development because it provides the application can reach wide range of user. So, we determined to implement our system in Java because its Java Virtual Machine provides platform independency.

Modifiability:

In our “The Wall” game, we constructed our system as a three-tier architectural decomposition since it’s more suitable for our project design. Because we are going to have 3 different layers which are presentation, business and data layer. All the communication between these layers must through the business layer. It means presentation and data layer never communicates directly. The game engine is inside the business layer and it both gets and sets our data layer and presentation layer. We choose this decomposition because we are going to have important amount of work with data and this decomposition allow us to it more easier than MVC which does not have a suitable data layer. We are going to use MySql and .txt files for our data storages. The developer mode is going to use MySql and other modes are going to use .txt files. Because we have sharing map, uploading map, rating map properties in developer mode.

Performance Criteria:

Response Time: In our “The Wall” game we don’t need to update the screen regularly because the original Walls&Warriors game is not a game which challenges players with time. Player need to think a lot, and play a move. Being to fast won’t make a player better than other player if his/her answer is not correct. Thus, all we need to do is update screen when an action comes up such as selecting a game object, dragging a game object or dropping a game object. Thus, our game must respond instantly to the user when a click comes up.

Trade Offs:

Functionality vs Time: We want do our best while doing this game. We want to add additional characters, coin feature, hints, different modes, making money by showing ads for free coins, online and offline modes, allow players to upload, download, play and rate other players maps features and etc. But we have limited time to compile all of these features. Therefore, we may not

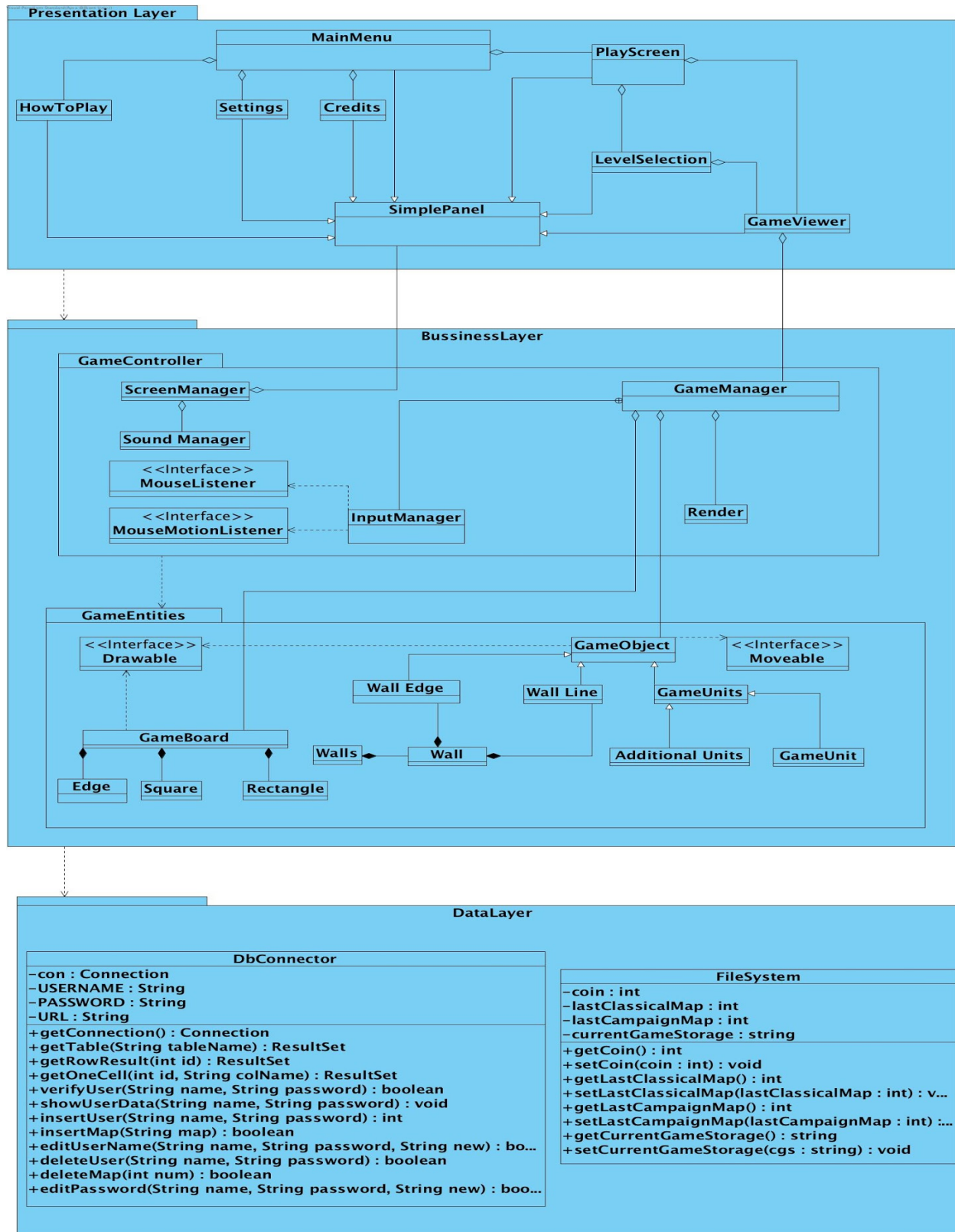
complete all the features that we want to add our game.

Easy of Using & Learning vs. Functionality: We want our game to be easy to learn and use. That's our priority against functionality. We don't want to confuse our user with complex functions. We want to increase usability instead of functionality. We may add lots of functions for trivial things but it will decrease the ease of using and playing of The Wall game. For instance, a person between 12-60 age is going to understand and play our game at most in 10 minutes.

2.High Level Architecture

2.1 Subsystem Decomposition

In this section, our system is divided into almost independent part to show how our game organization works. We constructed our system as a three-tier architectural decomposition because it's more suitable for our project design which we are going to database systems.



2.2 Architectural Styles and Design Decision

Layers

Our The Wall game system decomposition consists of 3 different layers such as Presentation Layer, Business Layer and Data Layer. These three layers have a hierarchy between each other. Our top layer is Presentation Layer as expected since its responsible for the interaction with user. Business Layer layer is the following layer. It is actual game engine for our project which get information from Game Objects and Database and fills the Presentation Layer's background. We have 1 bottom layers since we are going to implement our game which can be played either online or offline. First of them is Game Entities Package which contains all the necessary objects for our game. Finally our last layer is Data Layer which will store the necessary information for our online game on a local server.

Three-tier Architecture Decomposition

Three-tier architecture decomposition is mainly similar to Model View Controller architectural style. The difference is three-tier is also has base layer for database systems. Other than we are going to have our model, view and controller classes separately. By doing these divisions from a big system to smaller subsystems, we are planning to decrease complexity of our project. Basically, we isolated the main game engine

2.3 Design Decisions

Hardware / Software Mapping

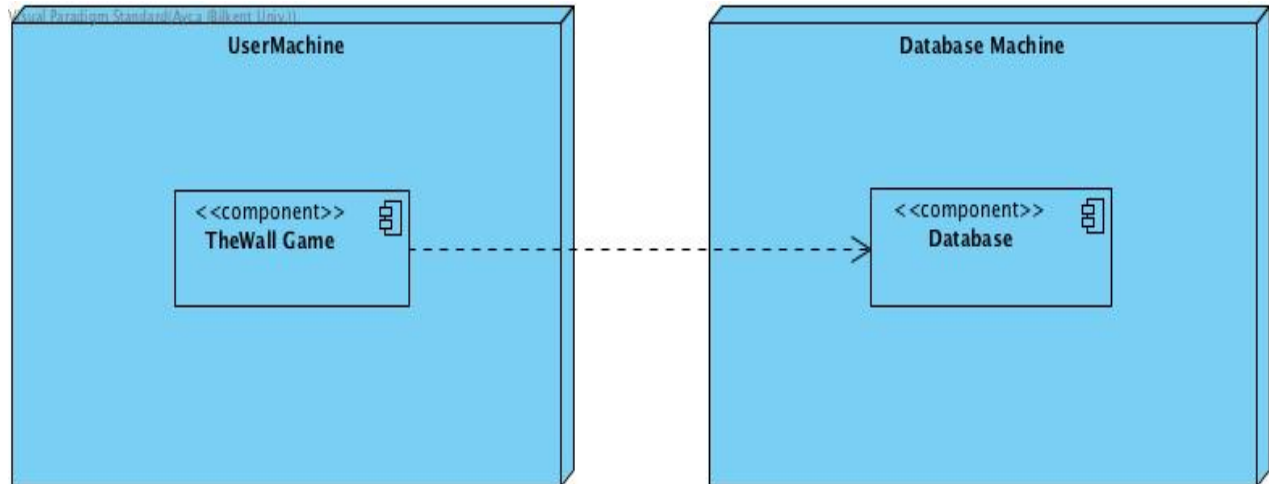
During the implementation process of the game "The Wall", Java programming language will be used, which was developed by the corporation Oracle. We will use the latest Java development kit during the implementation process(Java SE DK 8u191).

In terms of Hardware configuration, "The Wall" mostly requires a basic computer mouse in order to enable users to play and traverse the game. A keyboard is also

required as the users are expected to type their names and password for authentication processes. Therefore, it can be said that user's devices hardware should include both mouse and keyboard.

The game will be implemented and to be able to run on Java. Therefore, a computer with operating system and java system will be enough to run our game. The computer should be able to compile and run the files with the extension of ".java". Therefore existence of the compiler and any operating system is adequate to run the game. The Wall game have both file system and database system for keeping its data. Login information, uploaded maps will be kept in database. We are going to use MySql tables for this purpose. On the other hand, all the informations about Classical, Challenge and Campaign modes, coin information, last game information(if a player exits from the game in level 5, he can start from level 5 not from the beginning), will be stored in file system which are ".txt" files. While the implementation process, we will use 2D graphics libraries of Java. It does not require and additional software rather than Java environment to be rune. In terms of the hardware, most of the computers are able to run 2D graphics.

Deployment Diagram



Our deployment diagram is the for showing a system design after these system design decisions have been made. In our project, we have two nodes: UserMachine such as a personal computer and our Database Machine which is set to a computer of the one of us(developer). Database component provides requested data to the TheWall Game which is a component of UserMachine Node.

Persistent Data Management

The game of "The Wall" enables users to share their own maps with the game community. Therefore, a server database is required to let users upload and download the desired maps. The maps will be hold and saved in text format (.txt), and these data will be persistent in the server. The server will also store the usernames and their passwords in the database. We will have a server database consisting of the maps and user's data's.

Access Control and Security

"The Wall" will enable users to play the base game without any internet connection. However, to share and download the designed maps created by community, internet connection is required.

The users have to login into their accounts in order to access the online features. The users are expected to remember their passwords and usernames for the sake of security. We are planning to help the users having struggles while logging in (forgot password option). The usernames and password will not be shared with anyone else and these data will be stored in the server database. The maps created by users will be stored in the local memory until the users want to share it with the community. Therefore, server will not access these data's without the permission of the users. The other data's to be used in the game (sounds, images, maps) will be stored in the local hard drive in order to make the game playable without the internet connection.

Boundary Conditions

Initialization:

"The Wall" will not require any installation of the game. Therefore, ".exe" or some other extension formats will not be used. We are planning to make the game executable with the format of ".jar".

In order to initialize the online features, authentication from the server is required. The users will enter their login information and the server will try to match these information's with the respective data's in the server database.

Termination:

The game "The Wall" can be terminated by using (clicking) the "quit game" option that is in the main menu or by using the top right button imaging cross. The termination of the game will also terminate the connection between the user and the server.

The users can also terminate their connection with the server by logging out. This enables them to continue the game in local database.

Error:

If an error occurs while initializing the game such that it will not launch, the game will not start.

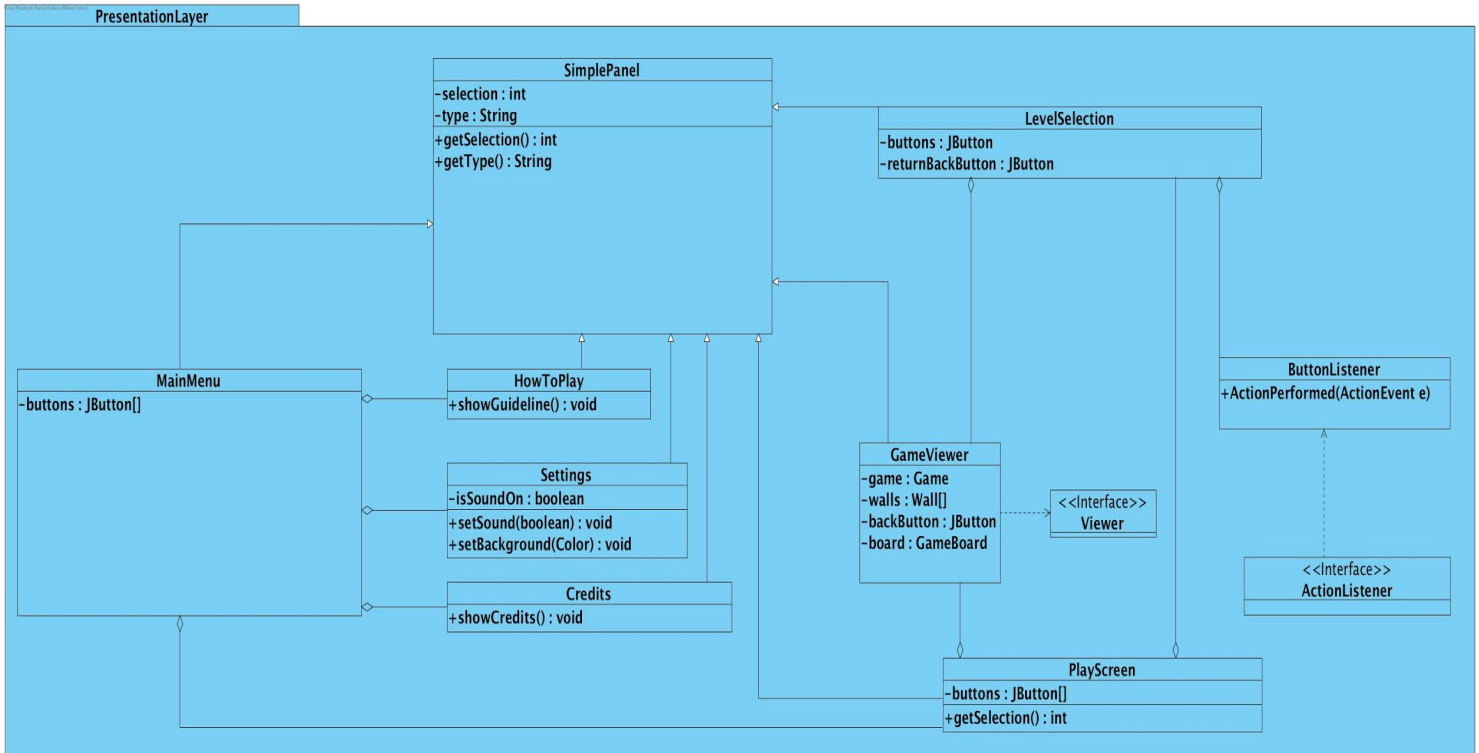
If a user will not be able to login to the server, user will be asked to re-enter the login information or to be redirected to the "forgot password" option. If the problem is because of the internet connection, users will be informed about the error.

If an error occurs while playing the game, most probably the progress of the last game will be lost. However, the progress of the "campaign mode" will be held in the local hard drive and users would continue from the last level they have come.

If an error occurs while sharing and/or downloading data from the server database, the users will be informed.

3.Low-Level Object Model

3.1. Presentation Layer Subsystem



MainMenu Class:



Attributes:

JButton[] buttons: to create buttons such as "Play Game", "Settings", "Credits" and "How to Play".

Constructor:

`public MainMenu():` Initializes the main menu of the game, set the panel according to the desired layouts, adds action listener and the created buttons on the panel.

SimplePanel Class:

| SimplePanel | |
|-----------------------|--|
| -selection : int | |
| -type : String | |
| +getSelection() : int | |
| +getType() : String | |

Attributes:

`int selection:` This attribute is for the detect which panel will be presented in mainframe, each panel has got an unique selection id.

`String type:` This attribute is for specifying which panel's name's, in our implementation, panels are also differentiated with their unique names.

Constructor:

`public SimplePanel():` Initializes the SimplePanel for the first time, which will be arranged to the mainframe to be displayed on the application.

Methods:

`public int getSelection():` This method returns to the SimplePanel's selection ID.

`public String getType():` This method returns to the SimplePanel's type which is also panel's name.

PlayScreen Class

| | |
|------------------------------------------------|-------------------|
| Visual Paradigm Standard (Ayca @Bilkent Univ.) | PlayScreen |
| -buttons : JButton[] | |
| +getSelection() : int | |

Attributes:

JButton[] buttons: to create buttons such as "Classical Mode", "Developer Mode", "Challenge Mode", "Campaign Mode" and "Return to Main Menu".

Constructor:

public PlayScreen(): Initializes the play screen of the game which that user can select the mode that s/he desired. Setting the panel according to the desired layouts, adds action listener and the created buttons on the panel.

GameViewer Class

| | |
|------------------------------------------------|-------------------|
| Visual Paradigm Standard (Ayca @Bilkent Univ.) | GameViewer |
| -game : Game | |
| -walls : Wall[] | |
| -backButton : JButton | |
| -board : GameBoard | |
| +paintComponent(Graphics g) : void | |
| +update() : void | |

Attributes:

private Game game: To initialize a game to present.

`private Wall[] walls`: Wall array to present walls in the game screen.

`private JButton backButton`: A button which will give the user the opportunity to return to the previous screen.

`private GameBoard board`: To present the initial game board to the user.

Constructor:

`public GameViewer(Game game)`: Initializes the game screen which that user can play the game in that screen. Setting the panel according to the desired layouts, adds action listener and the created buttons on the panel. Takes a game as a parameter to set up according to its pattern.

Methods:

`public void paintComponent(Graphics g)`: This method draws the game board and the walls according to the game pattern.

`public void update()`: updates the screen simultaneously so the player can move and build the walls.

LevelSelection Class

| | |
|-----------------------------------------------------------------|-----------------------|
| Visual Paradigm Standard (Ayca Bilkent Univ.) | LevelSelection |
| -buttons : JButton -returnBackButton : JButton | |

Attributes:

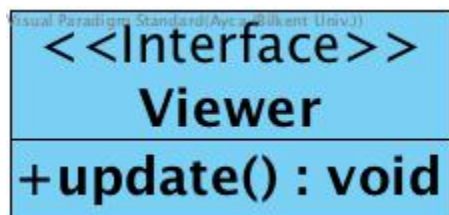
`JButton[] buttons`: to create level buttons.

`JButton returnPrev`: a button which will give user the opportunity to return to previous screen.

Constructor:

`public LevelSelection()`: Initializes the level selection screen of the game which that user can pick the level that s/he wants to play. Setting the panel according to the desired layouts, adds action listener and the created buttons on the panel.

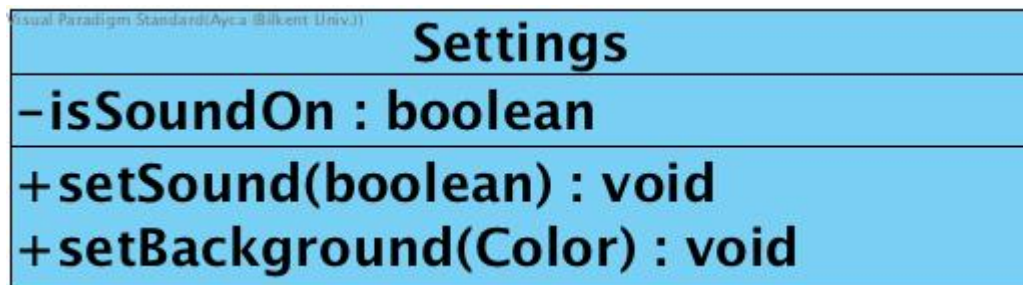
Viewer Class



Methods:

`public void update()`: This method is used in different classes of the game to update the screen.

Settings Class



Attributes:

`private boolean isSoundOn`: This attribute indicates whether the voice is opened or not. True means the voice is opened while false means that the voice is closed.

`private <arraylist> icons`: This attribute is the array of images that users might prefer instead of the default images.

Constructor:

`public Settings()`: This constructor initializes the Settings for the first time.

Methods:

`public void setSound(boolean isSoundOn)`: This method sets sound according to the parameter, if the parameter is true, sound will be turned on, else turned off.

`public void setBackground(Color color)`: This method sets background color according to the given parameter which is in the Color type.

HowToPlay Class

| | |
|----------------------------------------------|--------------------------------|
| Visual Paradigm Standard/Ayca Bilgin (UoV.I) | HowToPlay |
| | +showGuideline() : void |

Constructor:

`public HowToPlay()`: Initializes the HowToPlay panel of the game which will guide user to playing the game. Setting the panel according to the desired layouts, adds action listener.

Methods:

`public void showGuideline():` This method lead to give a brief explanation about the game.

Credits Class

| | |
|----------------------------------------------|----------------|
| Visual Paradigm Standard(Ayca Bilkent Univ.) | Credits |
| +showCredits() : void | |

Constructor:

`public Credits():` Initializes the Credits panel of the game which that the developers' names will be included..

Methods:

`public void showCredits():` This method displays the credits, which include the names of the developers, on the screen.

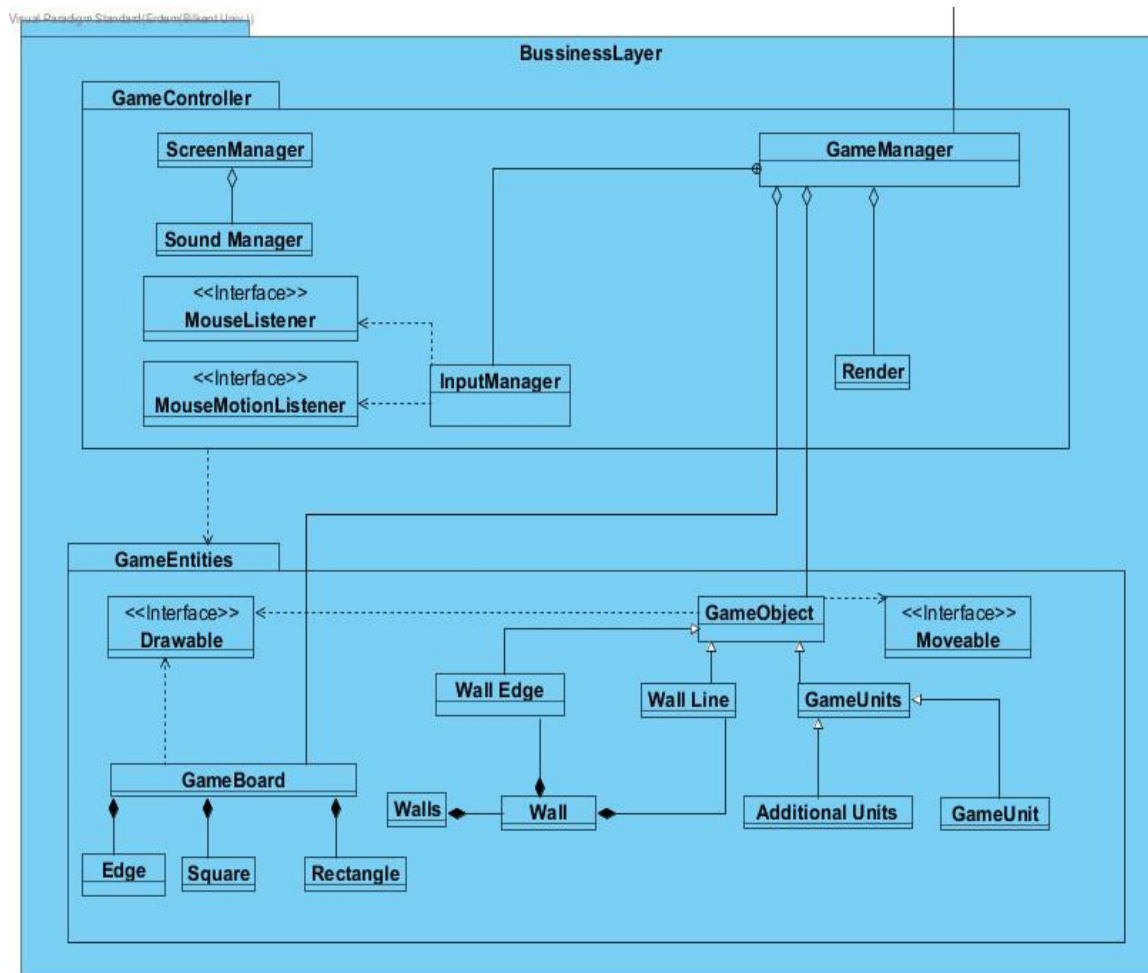
ButtonListener Class

| | |
|----------------------------------------------|-----------------------|
| Visual Paradigm Standard(Ayca Bilkent Univ.) | ButtonListener |
| +ActionPerformed(ActionEvent e) | |

Methods:

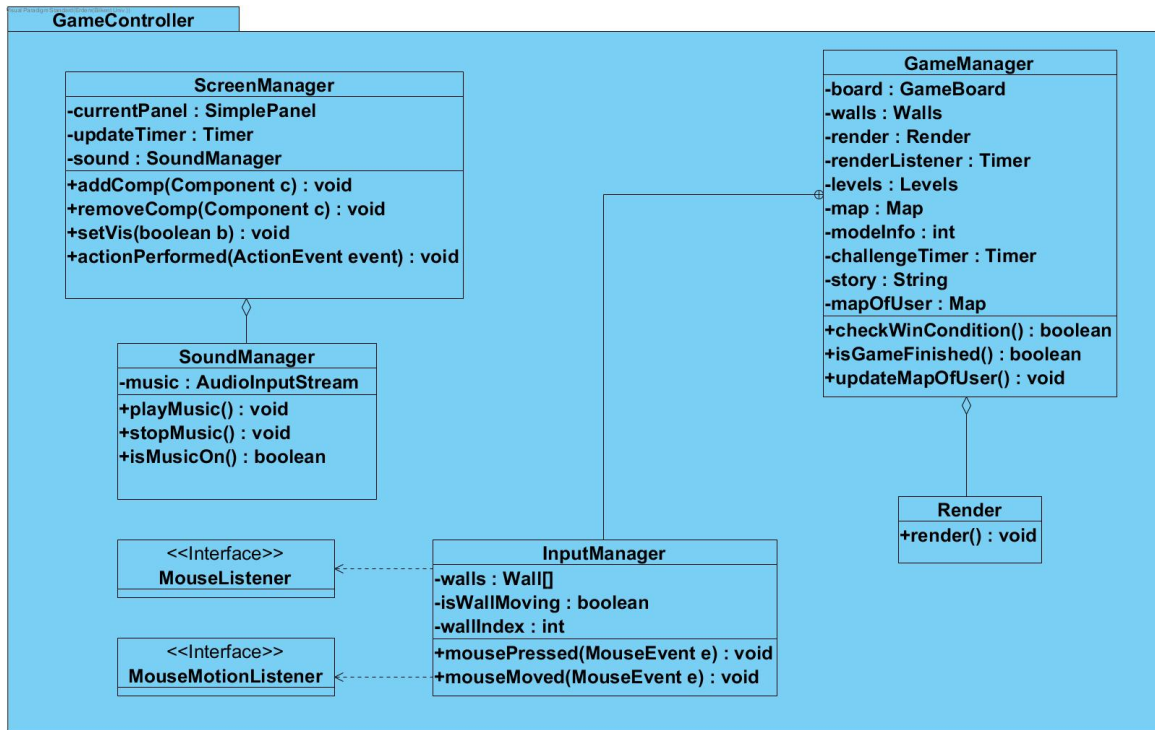
`public void actionPerformed(ActionEvent event):` This method takes the event's source and based on this information decides which button is pressed.

3.2 BusinessLayer Subsystem



Business Layer which contains two subpackages: GameController Package and the GameEntities Package.

GameController SubPackage



GameController subpackage for the controller of the game; logic of the game is managed in here. GameController subpackage includes Screen Manager, SoundManager, InputManager(listeners such as MouseListener and MouseMotionListener) , GameManager (a render object for GameManager).

GameManager Class



GameManager class to control game screen and whole game logic. This class sets game screen based on the selected level, map pattern etc. . Based on this class the whole game screen will be presented at GameViewer.

Attributes:

private GameBoard board: A game board which lead user to play on that.

private Walls walls: The walls is the class which all the game walls are in it. Walls type wall variable lead to present them in game logic.

private Render render: This variable leads to update and repaint the game simultaneously.

private Timer renderListener: This variable sets the timer to call render consistently.

private Levels levels: the levels of the game for initializing the game based

on this.

`private Map map`: A map to implement game board.

`private int modeInfo`: simple ID for the mode to implement the game based on this.

`private Timer challengeTimer`: A timer for the "Challenge Mode" of the game.

`private String story`: A basic stories which will be shown in "Campaign Mode"

`private Map mapOfUser`: `mapOfUser` copies "squareLocations" variable of the map object and make all `edgeLocations`, `horizontalRectangleLocations` and `verticalRectangleLocation` variable's values as 0 which mean no wall placed to the wall. When user places a new wall it will be updated according to its locations.

Constructor:

`public GameManager(Map map, int modeInfo)`: Initializes a game logic with specified Map and mode information. According to these parameters a wall line is constructed.

Methods:

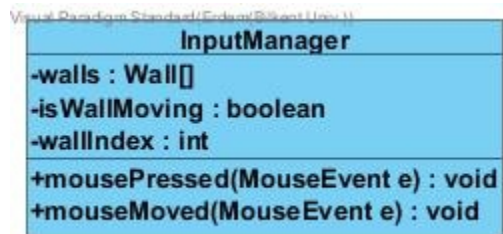
`public boolean checkWinCondition()`: Checks the win condition with the comparison of the `mapOfUser` and `map`. If both objects variables are same win condition is satisfies.

`public boolean isGameFinished()`: Checks if the user still plays the game or left without won the game (game is not finished), otherwise, game is

finished and isGameFinished method returns true.

`public void updateMapOfUser():` Updates the mapOfUser when user makes a move.

InputManager Class



Attributes:

`private Wall[] walls:` An array of Wall type, this array will be used to take input based on walls (user clicks on walls).

`private boolean isWallMoving:` Checks if the user holds and replaces the wall's position, which is basically moving.

`private int wallIndex:` An integer type variable to understand which wall is held, returns wall number.

Constructor:

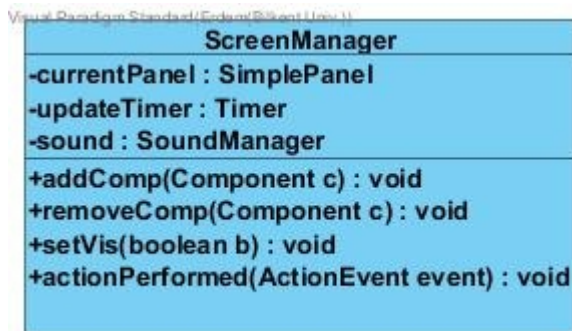
`public InputManager(Wall[] walls):` Initializes an input manager/controller to take input which will be received on walls.

Methods:

`public void mousePressed(MouseEvent e):` A listener to receive inputs from mouse presses.

`public void mouseMoved(MouseEvent e):` A listener to receive inputs from mouse presses.

ScreenManager Class



A controller to control screens, transitions, time conditions and pop-ups.

Attributes:

private SimplePanel currentPanel: A simple panel which is currently displaying.

private Timer updateTimer: A timer to update the screen simultaneously.

private SoundManager sound: A controller which user can controll the sounds of the game.

Constructor:

public ScreenManager(): Initializes the screen controller and its specified timer, screen's dimensions and other attributes..

Methods:

public void addComp(Component c): This method adds a component to the screen; the components such as current panel.

public void removeComp(Component c): This method removes a component from the screen; the components such as current panel.

public void setVis(boolean b): A method to lead the screen controller's current panel to be visible.

`public void actionPerformed(ActionEvent event):` This method checks the current stage of the game and accordingly updates the panels.

SoundManager Class



A class to control the background music.

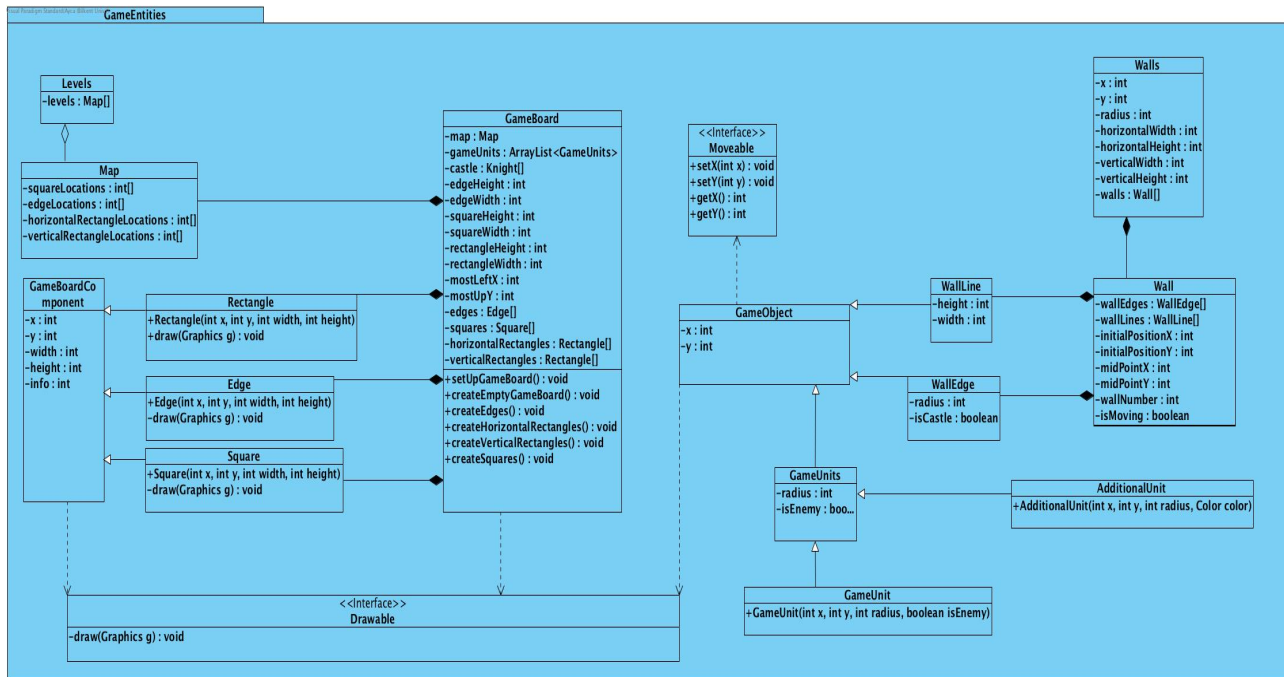
Methods:

`public void playMusic():` Continues to play the music.

`public void stopMusic():` Stop the music.

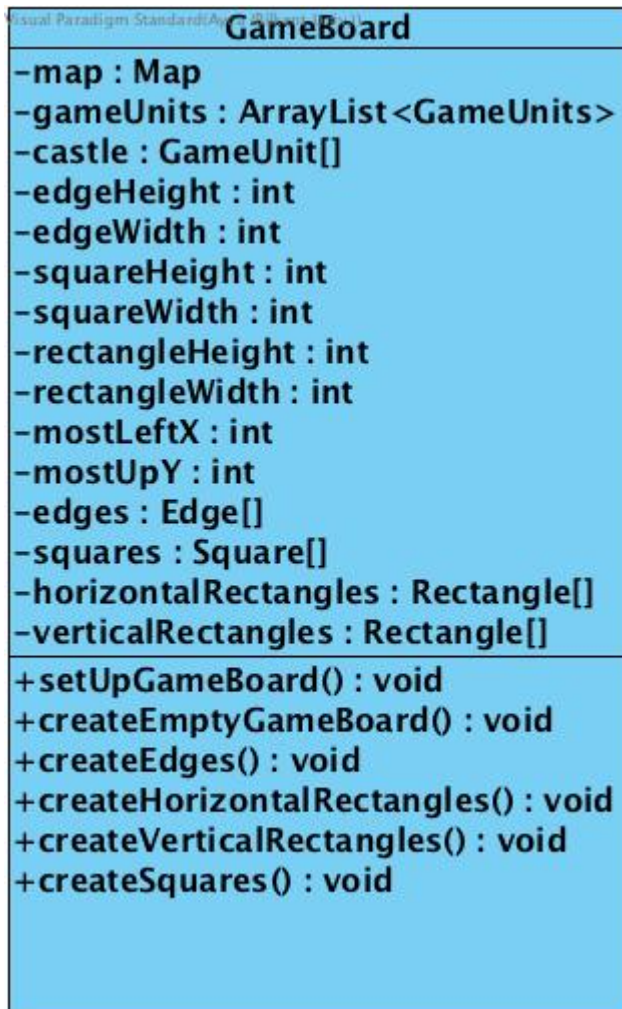
`public boolean isMusicOn():` Checks if music is on.

GameEntities Subpackage



In the Game Entities sub-system, all the game entity classes are grouped together with their relations. We have the following classes: Levels, Map, GameBoardComponent, Rectangle, Edge, Square, GameBoard, GameObject, WallLine, WallEdge, GameUnits, GameUnit, AdditionalUnit, Wall, Walls and the interfaces: Drawable and Moveable. These classes and interfaces are explained in this section.

GameBoard Class



GameBoard class creates objects which are related to the gameboard such as edges, squares, horizontalRectangles and verticalRectangles. This creation basically forms a basic gameboard. After creation of the basic(empty) gameboard, also creates GameUnits according to the map information.

Attributes:

`private Map map`: A map object contains the information of the map.

`private ArrayList<GameUnits> GameUnits`: List contains the GameUnit objects such as knights.

`private GameUnit[] castle`: Array contains the special GameUnit object which is castle

`final private int edgeHeight`: Fixed height of edges.

`final private int edgeWidth`: Fixed width of edges.

`final private int squareHeight`: Fixed height of squares.

`final private int squareWidth`: Fixed width of squares

`final private int rectangleHeight`: Fixed height of rectangles.

`final private int rectangleWidth`: Fixed width of rectangles.

`final private int mostLeftX`: Most left position of the game board.

`final private int mostUpY`: Most upper position of the game board.

`private Edge[] edges`: Edges which are one of the objects creates the game board.

`private Square[] squares`: Squares which are one of the objects creates the game board.

`private Rectangle[] horizontalRectangles`: horizontalRectangles which are one of the objects creates the game board.

`private Rectangle[] verticalRectangles`: verticalRectangles which are one of the objects creates the game board.

Constructor:

`public GameBoard(Map map)`: creates empty game board and setups the gameboard related to the map.

Methods:

`public void setUpGameBoard()`: Creates the GameUnits with respect to the map information and place them to the game board.

`public void createEmptyGameBoard()`: Creates empty game board.

`public void createHorizontalRectangles()`: Creates HorizontalRectangeles for the game board.

`public void createVerticalRectangles()`: Creates VerticalRectangeles for the game board.

`public void createEdges()`: Creates Edges for the game board.

`public void createSquares()`: Creates Squares for the game board.

GameBoardComponent Class

| GameBoardComponent |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -x : int -y : int -width : int -height : int -info : int |
| +getX() : int +setX(x : int) : void +getY() : int +setY(y : int) : void +getWidth() : int +setWidth(width : int) : void +getHeight() : int +setHeight(height : int) : void +getInfo() : int +setInfo(info : int) : void |

Main class for the game board components such as edge, square and rectangle. All of these game board components inherit from this GameBoardComponent class.

Attributes:

private int x: X position of the component.

private int y: Y position of the component.

private int width: Width of the component.

private int height: Height of the component.

private int info: Information number to identify component's type.

Constructor:

`public GameBoardComponent(int x, int y, int width, int height):` Basic constructor which sets the values of the parameters.

Methods:

Basic Getters and setters are listed below.

`public int getX():`

`public int getY():`

`public void setX(int x):`

`public void setY(int y):`

`public int getWidth():`

`public void setWidth(int width):`

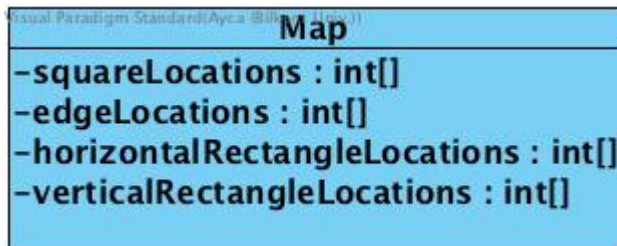
`public int getHeight():`

`public void setHeight(int height):`

`public int getInfo():`

`public void setInfo(int info):`

Map Class



Map class keeps the map informations as arrays. These arrays contains the information of the game units which are placed into squares, edges, horizontalRectangles and verticalRectangles. Arrays contains the locations of the GameUnits. For example;

For squareLocations:

0 = empty

1 = blueKnight is placed on that square

2 = redKnight is placed on that square

3 = castle is placed on that square

For edgeLocations, horizontalRectangleLocations and verticalRectangleLocations:

0 = empty

1 = full

Attributes:

`private int[] squareLocations`: Basic int array contains the game unit info.

`private int[] edgeLocations`: Basic int array contains the game unit info.

`private int[] horizontalRectangleLocations`: Basic int array contains the

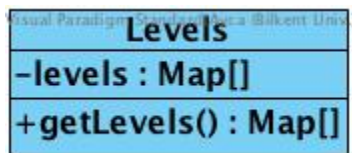
game unit info.

`private int[] verticalRectangleLocations`: Basic int array contains the game unit info.

Constructor:

`public Map(int[] squareLocations, int[] edgeLocations, int[] horizontalRectangleLocations, int[] verticalRectangleLocations)`: Basic constructor which sets the values of the parameters.

Levels Class



The image shows a UML class diagram for the 'Levels' class. It is a rectangular box with a light blue background and a black border. The box is divided into three horizontal sections. The top section is labeled 'Levels' in bold black text. The middle section is labeled '-levels : Map[]' in bold black text. The bottom section is labeled '+getLevels() : Map[]' in bold black text.

| |
|---------------------------------------------------|
| Visual Paradigm (Student License) (Bilkent Univ.) |
| Levels |
| -levels : Map[] |
| +getLevels() : Map[] |

This Class contains the all levels informations for the single player modes.

Attributes:

`private Map[] levels`: Map Array to hold map information for example levels[0] contains the map of level 1.

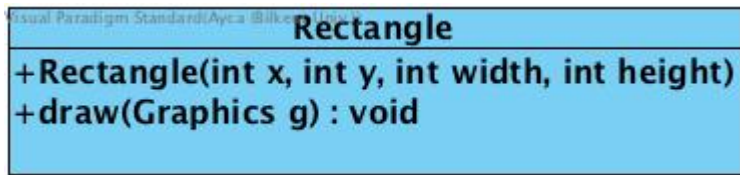
Constructor:

`public Levels()`: Setups all the level informations.

Methods:

`public Map[] getLevels()`: Method for get the desired map.

Rectangle Class



Inherits from `GameBoardComponent`. Rectangles are parts of the game board.

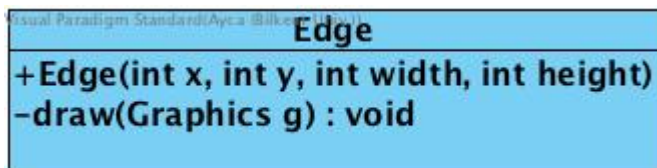
Constructor:

`public Rectangle(int x, int y, int width, int height)`: Directly calls the `GameBoardComponent`'s constructor.

Methods:

`public void draw(Graphics g)`: Draws the rectangle with respect to its positions, width and height.

Edge Class



Inherits from `GameBoardComponent`. Edges are parts of the game board.

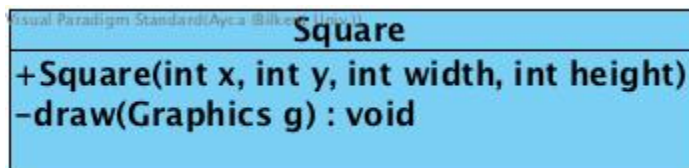
Constructor:

`public Edge(int x, int y, int width, int height)`: Directly calls the `GameBoardComponent`'s constructor.

Methods:

`public void draw(Graphics g):` Draws the edge with respect to the its positions, width and height.

Square Class



Inherits from GameBoardComponent. Squares are parts of the game board.

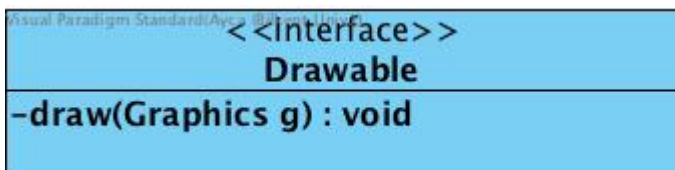
Constructor:

`public Square(int x, int y, int width, int height):` Directly calls the GameBoardComponent's constructor.

Methods:

`public void draw(Graphics g):` Draws the square with respect to the its positions, width and height.

Drawable Interface

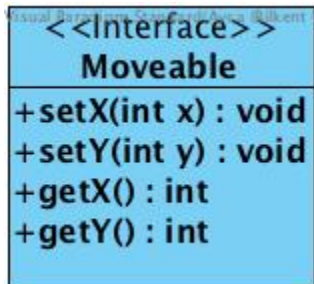


An interface to draw objects on the current panel.

Methods:

`public void draw(Graphics g):` A method to lead other classes to draw and present objects in the game.

Moveable Interface



An interface to move objects on the current panel.

Methods:

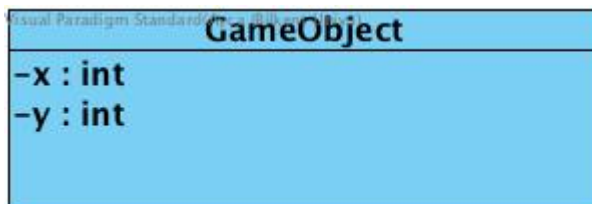
`public void setX(int x):` A method to set x coordinate.

`public void setY(int y):` A method to set y coordinate.

`public int getX():` A method to get x coordinate.

`public int getY():` A method to get y coordinate.

GameObject Class



An abstract class for game objects which has x and y coordinate. This class is used to create GameUnits(knights and the additional roles) and walls.

Attributes:

`private int x:` The x coordinate of the abstract GameObject.

`private int y`: The y coordinate of the abstract GameObject.

Constructor:

`public GameObject(int x, int y)`: Initializes a game object with specified x & y coordinate.. According to these parameters a wall line is constructed.

GameUnits Class



An abstract class which extends GameObject class. This class is used for creating knights and castle.

Attributes:

`private int radius`: The radius of a Game Unit which will be drawn as a circle.

`private boolean isEnemy`: A boolean type of flag which will check if the game unit will be presented as a blue knight or red knight.

Constructor:

`public GameUnits(int x, int y, int radius, boolean isEnemy)`: Initializes the GameUnits with specified x & y coordinate, radius and a flag which checks if the GameUnit will be presented as a red knight or blue knight. If is enemy is true than the game unit will be presented as red knight, else it will be presented as a blue knight. According to these parameters a wall line is constructed.

GameUnit Class

| GameUnit |
|------------------------------------------------------|
| +GameUnit(int x, int y, int radius, boolean isEnemy) |

Inherits from the GameUnits class.

Constructor:

`public GameUnit(int x, int y, int radius, boolean isEnemy):` Initializes a game unit with specified x & y coordinate, radius and a flag which checks if the GameUnit will be presented as a red knight or blue knight. If is enemy is true then the game unit will be presented as red knight, else it will be presented as a blue knight. According to these parameters a wall line is constructed.

AdditionalUnit Class

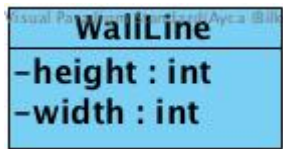
| AdditionalUnit |
|--------------------------------------------------------|
| +AdditionalUnit(int x, int y, int radius, Color color) |

A class which inherits from the GameUnits class. This class is used for additional characters/roles.

Constructor:

`public AdditionalUnit(int x, int y, int radius, Color color):` Initializes the additional unit with specified x & y coordinate, radius and the color. According to these parameters a wall line is constructed.

WallLine Class



Inherits from GameUnits. WallLines are parts of the wall.

Attributes:

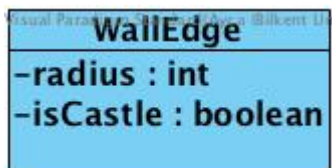
`private int height`: The height of a wall line.

`private int width`: The width of a wall line.

Constructor:

`public WallLine(int x, int y, int width, int height)`: Initializes a line of a wall with specified x & y coordinate, width and height. According to these parameters a wall line is constructed.

WallEdge Class



Inherits from GameUnits. WallEdges are parts of the wall.

Attributes:

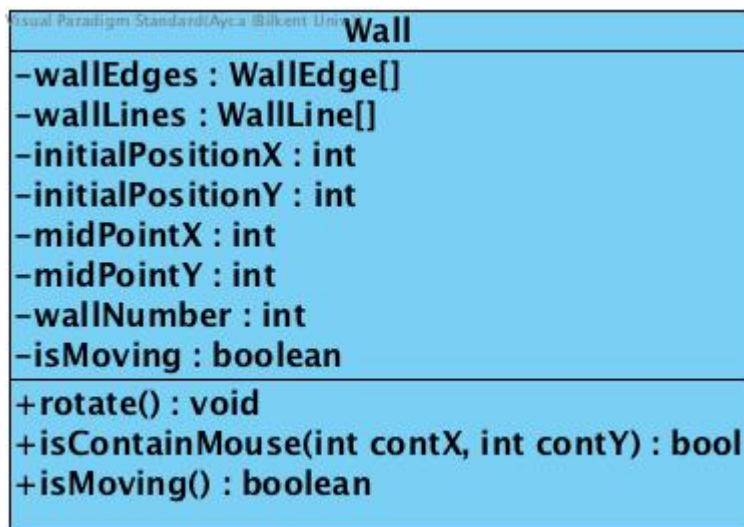
`private int radius`: The radius of a wall's edge.

`private boolean isCastle`: A boolean type flag to check is the wall edge is castle or not. This attribution is important in the real version of the game and it took significant importance for playing accurate.

Constructor:

`public WallEdge(int x, int y, int radius, boolean isCastle):` Initializes an edge of a wall with specified x & y coordinate, radius and the flag which checks if the edge will be presented as a castle or not. According to these parameters a wall edge is constructed.

Wall Class



Wall class to lead Walls class to create specific walls which are used in the game. Implements Moveable and Drawable interfaces.

Attributes:

`private WallEdge[] wallEdges:` A WallEdge type array which contains specified wall edges.

`private WallLine[] wallLines:` A WallLine type array contains specified wall lines.

`private int initialPositionX:` The x coordinate of the initial position of a wall.

`private int initialPositionY`: The y coordinate of the initial position of a wall.

`private int midPointX`: The x coordinate of the wall's specified (by us-developers) middle point. This attribute is for moving the wall (the user can click and hold the wall wherever s/he touches the wall are-is controlled with `isContainMouse()` method- however, mouse cursor will be replaced to the midpoint of the wall)

`private int midPointY`: The y coordinate of the wall's specified (by us-developers) middle point.

`private int wallNumber`: integer type wall number to initialize/specify a wall.

`private boolean isMoving`: a boolean type flag to check if the wall is moving (holding and moving by the user). If the wall is hold and moved by the user then `isMoving = true` else it is false.

Constructor:

`public Wall(WallEdge[] wallEdges, WallLine[] wallLines, int wallNumber)`: Initializes a wall with specified wall edges, wall lines and a wall number. According to these parameters a wall is constructed.

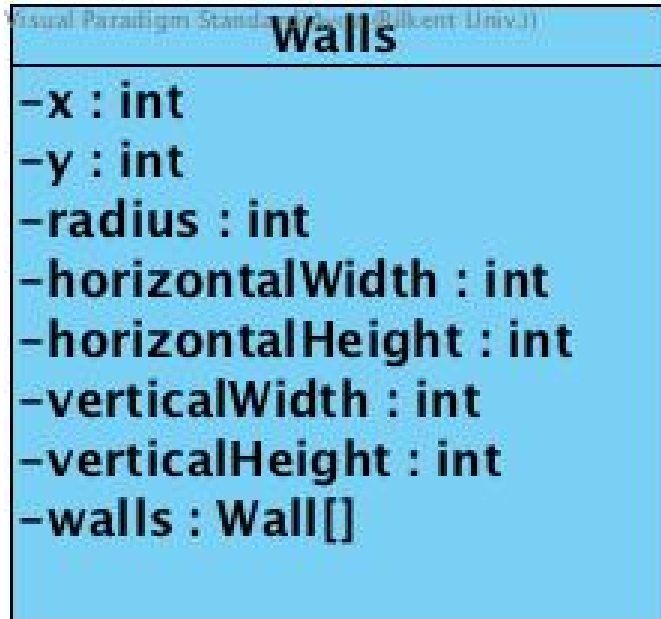
Methods:

`public void rotate()`: A method to rotate a wall in the game panel.

`public boolean isContainMouse(int contX, int contY)`: This method controls if the user click contains the wall's area.

`public boolean isMoving`: This method checks if the wall is moving.

Walls Class



Walls class to create the four walls of the game.

Attributes:

`private int x`: x coordinate of the wall.

`private int y`: y coordinate of the wall.

`private int radius`: radius of the edge for the presentation.

`private int horizontalWidth`: horizontal width of a wall.

`private int horizontalHeight`: horizontal height of a wall.

`private int verticalWidth`: vertical width of a wall.

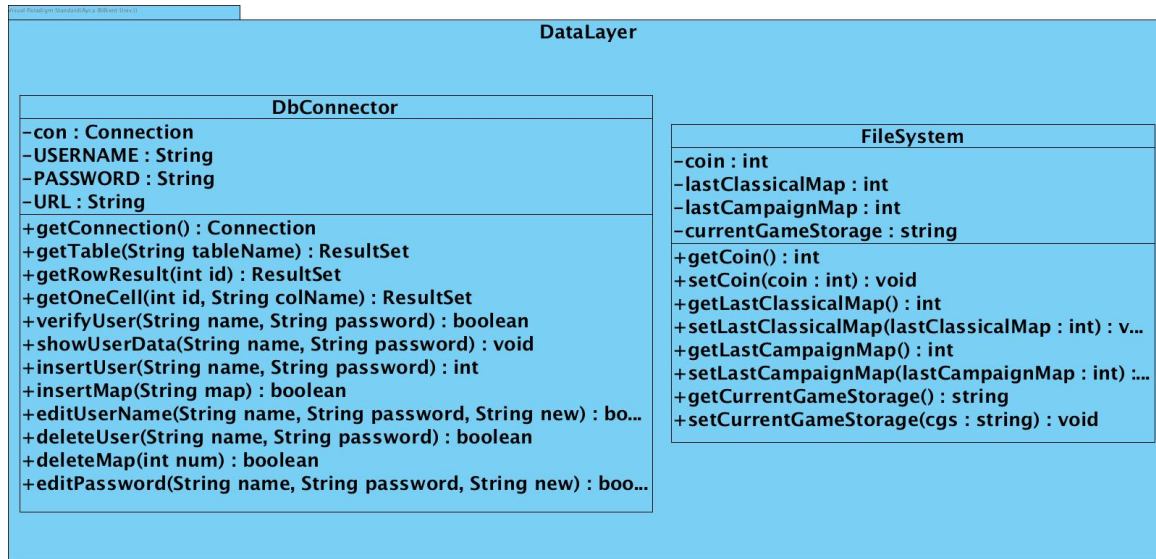
`private int verticalHeight`: vertical height of a wall.

`private Wall[] walls`: an array of walls (e.g. array size = 4 for classical game).

Constructor:

`public Walls()`: Initializes the Walls of the game screen that user can replace them to play the game.

3.3 Data Layer Subsystem



In data layer subsystem, we store the important informations for our game. We used 2 different system. Firstly, FileSystem which uses .txt files to store data. We used FileSystem for our offline game modes which are Classical mode, Campaign mode and Challenge mode. We keep coin, last passed level information for modes and current stage for recovery in case of crush. Secondly, DbConnector which uses MySql to store data. We used DbConnector for our online game mode which is Developer mode. We keep all the uploaded maps, downloaded maps, user informations in this

FileSystem Class:

Visual Paradigm Standard (Ersan@Sikent Univ.))

| FileSystem |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -coin : int -lastClassicalMap : int -lastCampaignMap : int -currentGameStorage : string |
| +getCoin() : int +setCoin(coin : int) : void +getLastClassicalMap() : int +setLastClassicalMap(lastClassicalMap : int) : void +getLastCampaignMap() : int +setLastCampaignMap(lastCampaignMap : int) : void +getCurrentGameStorage() : string +setCurrentGameStorage(cgs : string) : void |

FileSystem class stores coin, last passed map information of classical and campaign mode and current game state for starting game from where player left in case of crushing game or unwanted closes.

Attributes:

private int coin: coin information

private int lastClassicalMap: keeps the information of last passed classical mode map.

private int lastCampaignMap: keeps the information of last passed Campaign mode map.

private string currentGameStorage: it store current stage of a map. It is

updated after each move of player. If any case of crush, it will help us to recover game where a player left.

Methods:

`public int getCoin():` gets the number of coins

`public int getLastClassicalMap():` gets an integer which remarks a level number for classical mode

`public int getLastCampaignMap():` gets an integer which remarks a level number for campaign mode

`public string getcurrentGameStorage():` gets a string which stores an uncompleted map

`public void setCoin(coin:int):` sets the number of coins

`public void setLastClassicalMap(lastClassicalMap:int):` sets an integer which remarks a level number for classical mode

`public void setLastCampaignMap(lastCampaignMap:int):` sets an integer which remarks a level number for campaign mode

`public void setcurrentGameStorage(cgs:string):` sets a string which stores an uncompleted map

DbConnector Class:

| DbConnector |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -con : Connection -USERNAME : String -PASSWORD : String -URL : String |
| +getConnection() : Connection +getTable(String tableName) : ResultSet +getRowResult(int id) : ResultSet +getOneCell(int id, String colName) : ResultSet +verifyUser(String name, String password) : boolean +showUserData(String name, String password) : void +insertUser(String name, String password) : int +insertMap(String map) : boolean +editUserName(String name, String password, String new) : boolean +deleteUser(String name, String password) : boolean +deleteMap(int num) : boolean +editPassword(String name, String password, String new) : boolean |

DbConnector class is related to Developer mode. It stores user informations such as username, password, id numbers which we give users to ease the usage of table, uploaded maps which uploaded from all the users and downloaded maps from a specific user.

Attributes:

Connection con: connection between Java and MySql database.

String USERNAME: Username of database

String PASSWORD: password of database

String URL: the address of the database

Methods:

`Connection getConnection()`: creates a connection between Java and Mysql.

`ResultSet getTable(String tableName)`: take tableName as a parameter and goes into specific table in the database and returns it's Resultset.

`ResultSet getRowResult(int id)`: takes all the information belongs to a user.

`ResultSet getOneCell(int id, String colName)`: takes a specific information belongs to a user.

4. Improvement Summary

We have made some reasonable and technical changes based on the changes in our design decisions in our project since the first iteration. First, starting the implementation made a lot of classes clearer in our minds so we have added and removed so classes or some methods from our design. It was much easier to understand what was really necessary and what was not. So, this was definitely an improvement for our project. Changes are as follows:

- We changed our architectural style (the former wa MVC, now it is three-tier layer). Hence we changed all the packages, rewrite them all.
- We increased the number of classes while coding the project. Hence these most of these diagrams are reverse engineering product. (In every package). However also we deleted some of our classes which are unnecessary to implement our game, such as Mode classes in the User Interface package, BuildManager class in the Game Management, Castle class in the Game Entities Package in our first iteration report.
- We added new methods to our all classes based on the code that we have implemented.

As for the reports, we changed our design report second iteration based on

the feedback and the mail which our instructor sent as an email to whole students. Also, our changes based on the code that we implemented. As we writing and managing the classes in the implementation part(since the demo), we realised some of the crucial mistakes in design and fixed them.