Object Oriented Software Engineering Project

CS319 Project: Quantum Chess

Design Report

Group 2A

Rubin Daija

Serhat Bezmez

Kaan Aktürk

Mastan Abdulkhaligli

Contents

1.Introduction

1.1. Purpose of the System

Quantum Chess is sufficient to satisfy needs of the players. Graphics are shown as 2D. The interface of the game is not complicated. Thus, users will avoid confusions and will not waste their time to understand the game interface. It was desired players to gain new experiences and demonstrate them different and surprising dimensions of chess. So, we decided to design developed version of the classic chess game by shaping the rules of chess. It is also believed that the game will attract chess lovers.

1.2. Design Goals

In design goals part, it is demonstrated that what we expect and aim from our application. The essential factors that we paid attention during the development of the program are explained below.

1.2.1. Adaptability

The game is developed on Java environment. Even if Java requires extra effort to run GUI packages, it can be playable on major operating systems that contain JVM [1] such as Windows, Linux and Mac OS. We wanted to utilize this advantage of Java. So that, we can prevent people from dealing with operating system compatibility issues and let our game be reached by many people.

1.2.2. Efficiency

The state of being playable is one of the most essential matters for users. Therefore, we direct our attention to efficiency of the game. Power-ups and movements of the objects are designed with considering their level of smoothness. Thus, it is offered users to relish the game. To boost the performance of the game, we optimized our code. It is tried to find most suitable algorithms for our scenario. It is avoided finding tricky ways to implement the code in order not to sacrifice the performance of our program. So that, the burden of the code is lightened.

1.2.3. Reliability

For the unproblematic game pleasure, it is attempted to detect all the blanks in the code. To succeed in finding faults, during the process of developing the program, all of the possible reasons of crash of program are evaluated.

Borders of algorithm are investigated and tried to find solutions to get rid of them.

1.2.4. Usability

All the phases of the project, it is aimed to create a friendly interface. Since, we placed the complication into the play game screen not into the user interface. The interface of the application is simple and comprehensible for all users. It contains few buttons. So that, starting and discovering the interface do not require any knowledge and experience. In addition, there is also tutorial that people who are unfamiliar with the game can gain knowledge.

1.2.5. Extensibility

Sometimes, it is important to expand the work done. During the process, new functions, classes, components and API's [2] are required in order to increase the quality of the algorithms. So, extensibility of Java is utilized.

1.2.6. Trade-Offs

- ***Performance and Memory***

As it is mentioned before, our main purpose is to offer users pure entertainment. To boost the performance, we made concessions to memory. When players select one of the power-up options, our system will require more performance. In addition, the power-ups in the game are most crucial matter in the game. So that, we gave priority to the animation of power-ups and let our memory to increase.

- ***Usability and Functionality***

Generally, players give importance to friendly usability of applications. Since people do not cope with the complications. They desire system with simplest interface. We could not ignore their demand. So, we reduced the functionality of our menu interface.

- ***Efficiency and Reusability***

Our main idea is to create unique chess game. We did not decide to expand our game to another game. We confined our attention to the efficiency of the game. So, we did not consider the reusability of our project.

1.3. Definition, Acronyms and Abbreviations

JVM [1] : JAVA Virtual Machine which is an engine that can run JAVA programs. So that, operation systems that contain JVM can start JAVA applications.

API [2] : Application Programming Interface is a set of classes that developed in Java.

1.4. References

[1] "Java Virtual Machine (JVM)," *W3schools*, 20-Sep-2017. [Online]. Available: https://www.w3schools.in/java-tutorial/java-virtual-machine/. [Accessed: 18-Oct-2017].

[2] T. Sintes, "Just what is the Java API anyway?," *JavaWorld*, 13-Jul-2001. [Online]. Available: https://www.javaworld.com/article/2077392/java-se/just-what-is-the-java-api-anyway.html. [Accessed: 18-Oct-2017].

Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010

1.5. Overview

Our main purpose is to broaden people's experiences in chess. In this design report, our prospects about the game are displayed. We desire that almost every person can utilize from our game. So, the project is developed on Java

Environment. As much as possible, the faults of program are determined. To meet the demands of users, a friendly user interface is created. In addition, there are also some conflicts that we have to deal with. To increase the performance, we extend our limitations of memory. To provide friendly user interface, the functionality of menu page is diminished.

2. Software Architecture

2.1. Overview

In this section, we will start detailing the composition of our system to a higher level. It has decided to divide system into smaller subsystems in order to carry out the maintenance of code.

In the QuantumChess software project, the Model/View/Controller will be used as the architectural style since it enables us to divide the system into subsystems in accordance with the boundary, control and entity objects. Also, it provide fast access to the model since the model subsystem do not depend on the data flow of controller or view subsystems.

The layers and dependencies between the subsystems are explained and the hardware and software components are given. Also, the boundaries for the entities and interfaces are discussed along with the object’s lifecycle.

2.2. Subsystem Decomposition

Figure-2.1(below) shows the subsystem decomposition of the software application by using the MVC. On the other hand, Figure-2.2(below) shows the subsystem decomposition in details with the links between subsystems and packages. The system is divided to three subsystems including a controller subsystem, a view subsystem and two model (one Game and one Entity) subsystems. Also, the subsystem decomposition has a closed architecture with three layers.

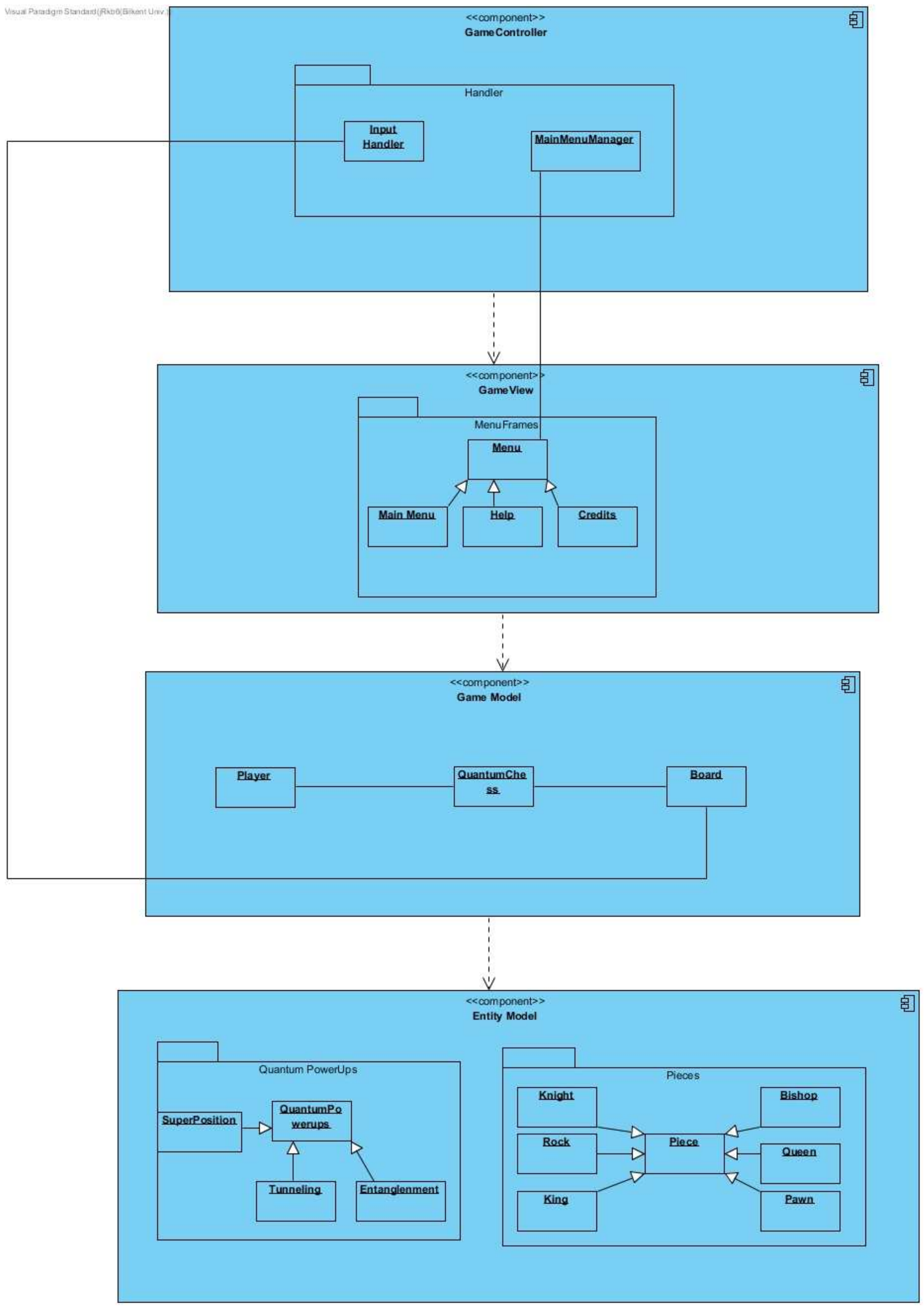


Figure-2.1. Subsystem Decomposition with Dependencies

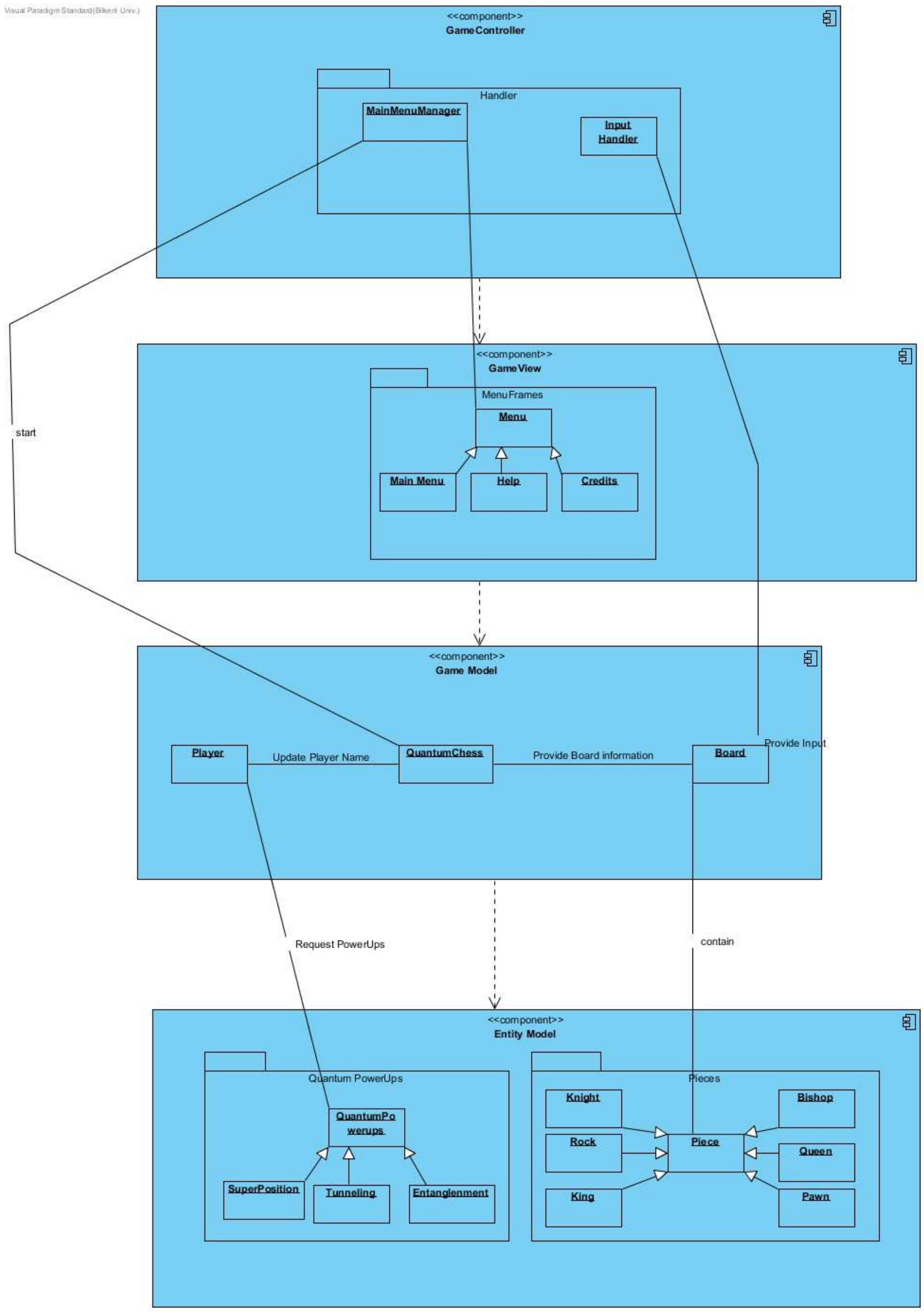


Figure-2.2. The detailed subsystem decomposition

The controller subsystem consists of handler classes and depends on the model subsystems. It controls the flow of the inputs and the movement changes. The controller subsystem request changes in the game logic to update the current game conditions.

The game model subsystem consists of QuantumChess, Player, Board and Input Handler classes. It controls the general game logic and makes the decisions needed to carry forward the game.
Board class takes the inputs and information from input handler in the controller subsystem. QuantumChess class updates the player information and game information. The game model subsystem request changes from the entity subsystem in accordance with the game logic used by game class.

The entity model subsystem consists of Pieces and QuantumPowerups packages and piece types and power up types as classes. Entity subsystem stores the objects and data relevant to the software application. Throughout the game, entity model subsystem makes updates to the entity objects of the game according to the requests of the game model subsystem. Entity model class notifies the game view subsystem when any change occurs in the pieces or powerups.

The game view subsystem consists of MainMenuManager class and menuFrames package which has different menu screens in itself. It displays the views on the screen and updates them when any prompt from the game view subsystem is received. MainMenuManager is responsible from the update of the game view due to the prompts from inputs. The menu package is responsible from the shifts between different menu views.

2.3. Architectural Styles

The model view controller architectural pattern is used for the system decomposition of the project. The choice of the MVC pattern aimed to decrease the dependencies between different subsystems and to increase the association between classes inside a subsystem. Therefore, two different model classes are designed in which game model interacts with the controller and controls the game logic and entity model serve as a repository for objects and notifies the view model when any change occurs in the stored data of the objects. Hence, a four-layered architecture is preferred for the decomposition of the subsystems with a closed architecture that the layers can only access the layer below them. Therefore, each layer in the subsystem decomposition uses the services of the layer below, which provides a more consistent system. The roles and functionalities of the objects and classes are considered to identify the subsystems of the software application.

2.4.Hardware / Software Mapping

The game will be implemented in Java programming language. Therefore, the computer system should support Java and have Java virtual machine on it. Since game will be supported with Java Virtual Machine, it has portability to play in different environments such as Linux, macOS and Windows. Also, the game will use keyboard inputs for the game control and mouse inputs for the shifts between menus. Hence, the necessary hardware requirements only include the keyboard and mouse

The graphical requirements for the game consist of a computer system that can support java swing library objects since the swing library will be used for the development of the user interface.

2.5. Data Management

We have only image files of the components that we have in two model subsystems. It is considered that move logs can be stored in text file in the future with better improvements. The text files will have the textual definitions of the both board and piece information and these textual data will be kept after the execution of the program. Therefore, if necessary, the persistent objects of the project will be stored in the textual format after the termination of the entity objects in the game. In case of system crash or data corruption, the retrieval of the images of components and move logs, might not be possible. Since it is runtime problem, it won't affect the data flow of new game execution.

2.6.Access Control and Security

QuantumChess does not require any access to the controls or security features since game data will be available on only execution time and players will be using the same computer.

2.7.Boundary Conditions

Execution: QuantumChess will not require any software other than Java Runtime Environment installation on the computer.

Start-up and Shutdown: QuantumChess will come with an executable .jar file. The user can initialize the program by clicking the executable file. The program will be terminated when one of the player quits the game by clicking close button. The active entity objects that are used inside the game and are not stored as persistent objects will be terminated after the game is finished.

Exception Handling: The game does not have any database or network connection that can result in exceptions except I/O hardware. However, if there is an exception in the loading of the persistent objects or the initialization of the game and entity objects or the connection exception due to I/O hardware, the system will display an error message that shows the exception and continue the game by ignoring the exception if it is possible.

3. Subsystem Services

3.1 Game Controller

This subsystem has two classes which control the input of the user and also the displaying of the material to the user.

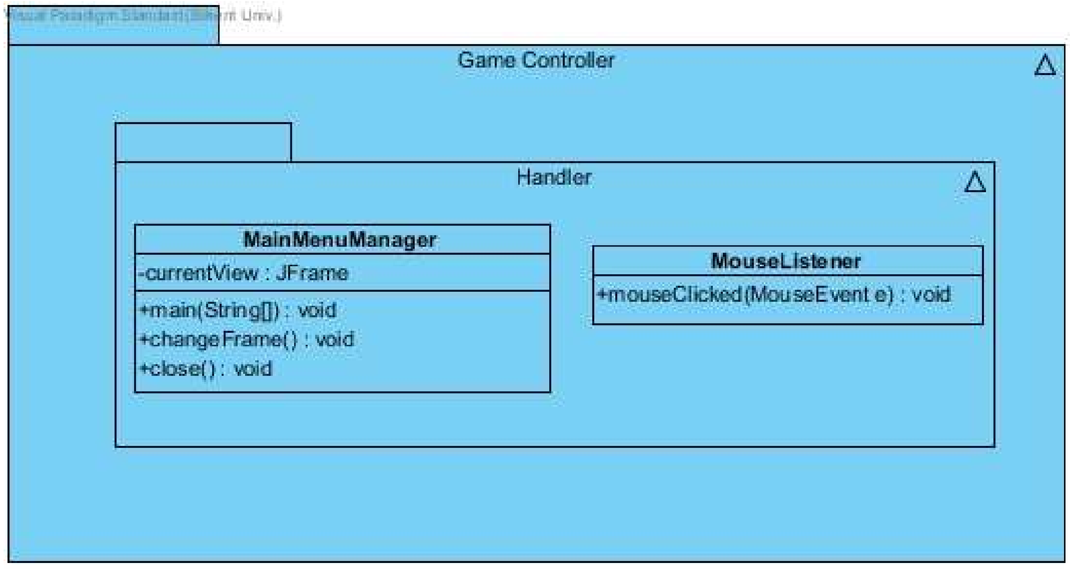


Figure 3.1.1 – Game Controller Subsystem

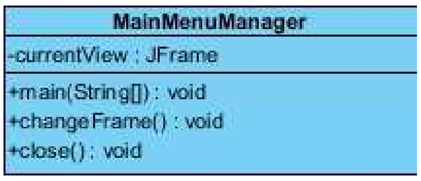


Figure 3.1.2 – MainMenuManager class

<u>MainMenuManager:</u>

Handles the controlling of that appears in the users screen.

Attributes:

currentView – holds the current view that is being displayed

Opperations:

main(String[]) – is the main function called in Java

changeFrame() – will change between frames when called

close() – will close the program programmatically

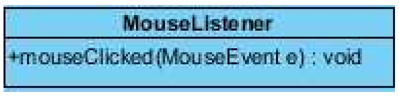


Figure 3.1.3 – MouseListener class

<u>MouseListener:</u>

Handles the input on the board by the user.

Operations:

mouseClicked(MouseEvent e) – will inform the system that the mouse has clicked and give the coordinates of the click.

3.2 Game View

This subsystem controls the navigation panels for the user.

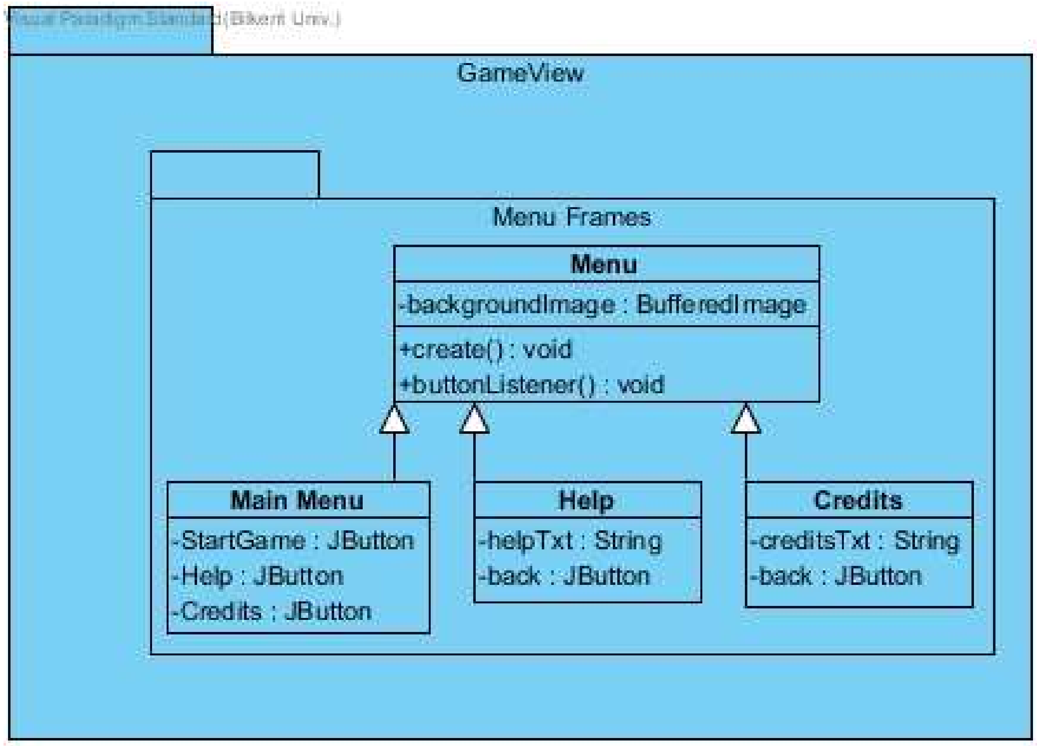


Figure 3.2.1 – GameView subsystem

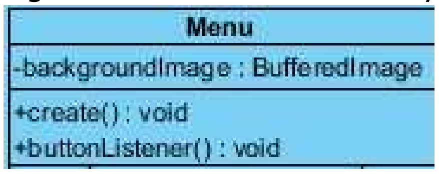


Figure 3.2.2 – Menu class

<u>Menu:</u>

The parent of the other navigational frames.

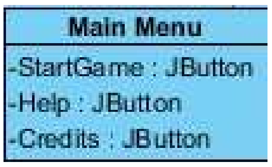
Attributes:

backgroundImage – holds the image that will be in the background

Operations:

create() – creates a specific menu

buttonListener() – listens to a specific button press



Figrue 3.2.3 – Main Menu class

<u>Main Menu:</u>

It is the main menu in which the player can start a game or move to the other directories.

Attributes:

StartGame – a button which if clicked will start the game if pushed, function handled by button listener

Help – a button which if clicked will go to the help frame, function handled by button listener

Credits – a button which if clicked will go to the credits frame, function handled by button listener

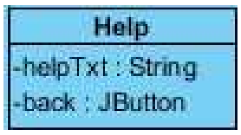


Figure 3.2.4 – Help class

<u>Help:</u>

It is the help frame in which help information is given to the player.

Attributes:

helpTxt: holds the help information

back: is a button which sends you back, functionality handled by button listener

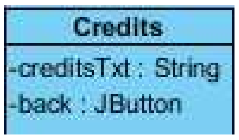


Figure 3.2.5 – Credits class

Credits:

It is the credit frame in which information about the developers and the game is given to the player.

Attributes:

creditsTxt: holds the credit information

back: is a button which sends you back, functionality handled by button listener

3.3 Game Model

This subsystem controls the gameplay of the game. It is responsible for the graphics mainly.

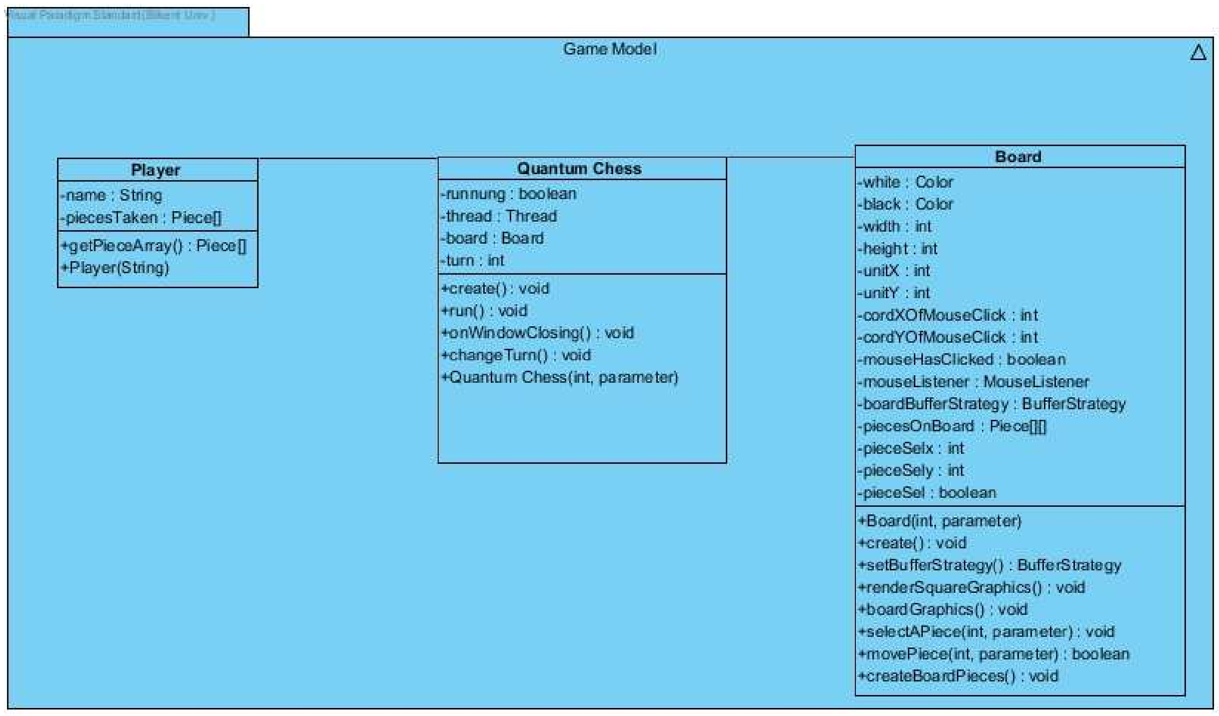


Figure 3.3.1 – Game Model subsystem

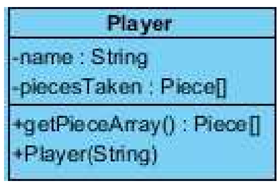


Figure 3.3.2 – Player class

Player:

Will include some information about the player.

Attributes:

name – will hold the name of the player

piecesTaken – will be an array with the pieces of this player which aren't present in the game anymore

Operations:

getPieceArray() – will return all the pieces taken

Player(String) – is the constructor

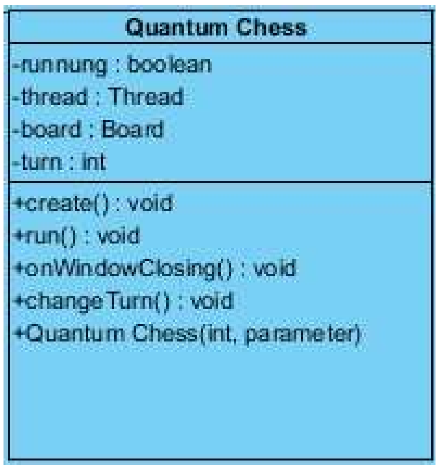


Figure 3.3.3 – Quantum Chess class

<u>Quantum Chess:</u>

It is the connection between the board and the player. It extends Jframe and implements Runnable.

Attributes:

running – will be a signal for the thread as to run or not

thread – hold the thread which will run the games gameplay and graphics

turn – represents the turn of a player

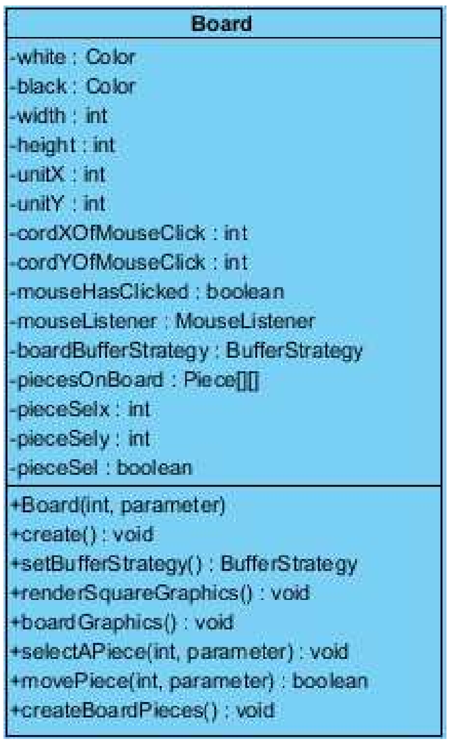
Operations:

create() – will start the game automatically

runt() – will run the graphics in another thread onWindowClosing() – will stop the game and the thread properly when called

changeTurn() – will change the turn of the players

QuantumChess(int,parameter) – is a constructor who will create the elements needed for the game to start


Board
-white : Color
-black : Color
-width : int
-height : int
-unitX : int
-unitY : int
-cordXOfMouseClick : int
-cordYOfMouseClick : int
-mouseHasClicked : boolean
-mouseListener : MouseListener
-boardBufferStrategy : BufferStrategy
-piecesOnBoard : Piece[][]
-pieceSelx : int
-pieceSely : int
-pieceSel : boolean
+Board(int, parameter)
+create() : void
+setBufferStrategy() : BufferStrategy
+renderSquareGraphics() : void
+boardGraphics() : void
+selectAPiece(int, parameter) : void
+movePiece(int, parameter) : boolean
+createBoardPieces() : void
Figure 3.3.4 – Board class

<u>Board:</u>

Draws the board as well as the other pieces. Extends Canvas.

Attributes:

white – predefined color

black – predefined color

width – width of the canvas

height – height of the canvas

unitX – width unit of a single square, width / 8

unity – height unit of a single square, height / 8

cordXOfMouseClick – holds the x coordinate of the last click on the canvas

cordYOfMouseClick – holds the y coordinate of the last click on the canvas

mouseHasClicked – true if there was a unique click on a particular square

mouseListener – is used to listen to the mouse clicks on the board

boardBufferStrategy – a buffer strategy used for the drawing of the board and the pieces actively

piecesOnBoard – a 2D array that holds all the Piece objects present on the game

pieceSelx – the x coordinate of the selected piece

pieceSely – the y coordinate of the selected piece

pieceSel – true if a piece is selected

Operations:

Board(int, int) – constructor

create() – creates the board and puts the pieces to their appropriate places

setBufferStrategy() – creates the buffer strategy and returns it

renderSquareGraphics() – renderes the squares in the board

boardGraphics() – renders the board with pieces

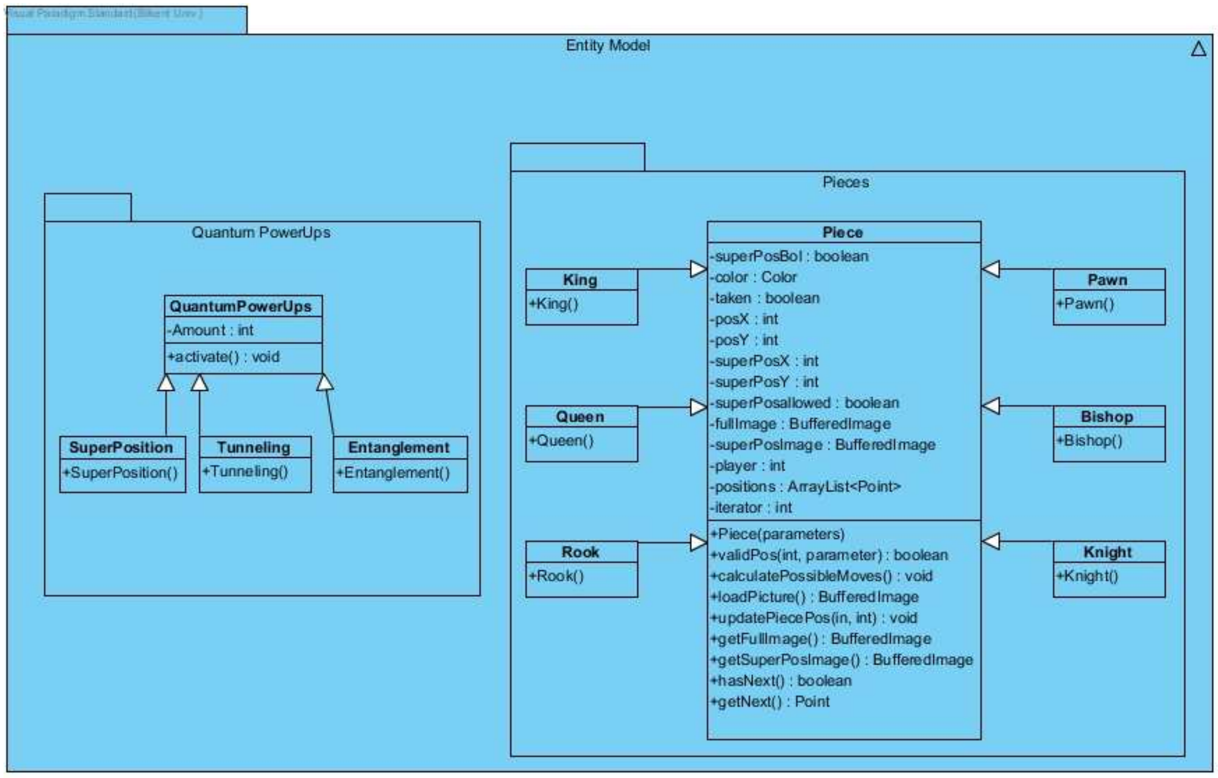
selectAPiece(int , int) – selects a piece on the board based on the click of the mouse

moveAPiece(int, int) – moves the piece and returns true if thee piece is moved

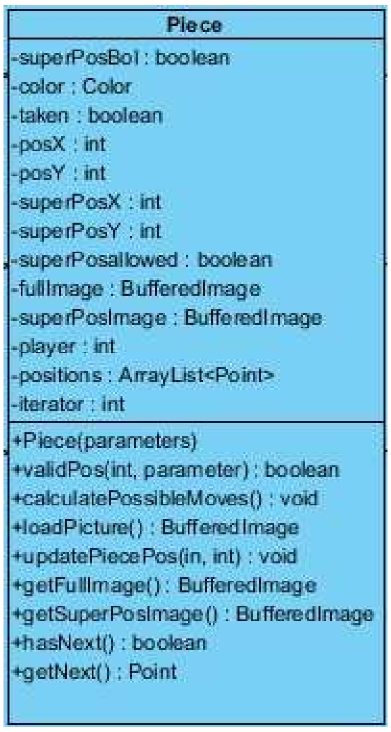
createdBoardPieces() – creates all the appropriate instances of the pieces needed

3.4 Entity Model

This subsystem holds all the necessary pieces of the gameplay. The pieces which are the objects used to play the game and the power ups to enhance the ability of the pieces.Figure



3.4.1 – Entity Model subsystem


Piece
-superPosBol : boolean
-color : Color
-taken : boolean
-posX : int
-posY : int
-superPosX : int
-superPosY : int
-superPosallowed : boolean
-fullImage : BufferedImage
-superPosImage : BufferedImage
-player : int
-positions : ArrayList<Point>
-iterator : int
+Piece(parameters)
+validPos(int, parameter) : boolean
+calculatePossibleMoves() : void
+loadPicture() : BufferedImage
+updatePiecePos(in, int) : void
+getFullImage() : BufferedImage
+getSuperPosImage() : BufferedImage
+hasNext() : boolean
+getNext() : Point
Figure 3.4.2 – Piece class

Piece:

It is the parent of all the other pieces.

Attributes:

superPosBol – true if a piece is in superposition

color – color of the piece

taken – true if a piece is taken

posX – x position on the board of the piece [0-7]

posY – y position on the board of the piece [0-7]

superPosX - x position on the board of the super positioned piece [0-7]

superPosY - y position on the board of the super positioned piece [0-7]

superPosallowed – true if superposition is allowed in this piece fullImage – holds the full image

superPosImage – hold the super positioned image

player – the value of the player to which this piece belongs

positions – an array list with the allowed positions for the specific piece saved as Point

iterator – used to iterate through the array list

Operations:

Piece(parameters) – constructor of the piece

validPos(int, int) – returns true if the position to be moved of the piece is correct

calculatePossibleMoves() – calculates all the possible moves for a piece based on their location

loadPicture() – fetches the picture file and loads it

updatePiecePos(int, int) – updates the position of the particular piece

getFullImage() – returns the BufferedImage object that has the full picture

getSuperPosImage() - returns the BufferedImage object that has the super positioned picture

hasNext() – is used to check if there are any positions left unchecked in the positions array list

getNext() – returns the current value of the array list based on the iterator and updates the iterator

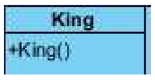


Figure 3.4.3 – King class

Figure 3.4.4 – Rook class

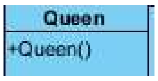


Figure 3.4.5 – Queen class

Knight
+Knight()
Figure 3.4.6 – Knight class

Bishop
+Bishop()
Figure 3.4.7 – Bishop class

Figure 3.4.8 – Pawn class

<u>King, Queen, Rook, Bishop, Pawn and Knight</u>

They extend the Piece class and implement calculatePossibleMoves() separately.

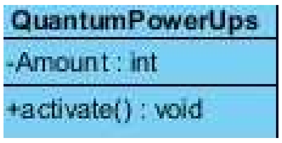


Figure 3.4.9 – QuantumPowerUps class

<u>QuantumPoweUps:</u>

It holds the essential information for each power up.

Attributes:

Amount – the amount of a selected power up left

Operations:

activate() – activates the power up in the game

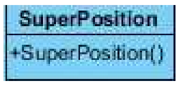


Figure 3.4.10 – SuperPosition class

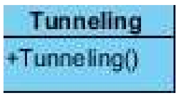


Figure 3.4.11 – Tunneling class

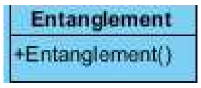


Figure 3.4.12 – Entanglement class

SuperPosition, Tunneling and Entanglement

They all extend QuantumPowerUps class and define their amounts accordingly.

4. Low Level Design

4.1.Object Design Trade-offs

4.1.1. Functionality vs. Usability

High usability is one of the our main purposes. When we design "Quantum Chess" we focus on usability level more seriously. Game's functionality is not complex that is why every player of our game can play easily. Before the game we put tutorial and it increases game's usability. Using mouse is enough to enjoy "Quantum Chess". Game's powerups is not complicated that is why users will not be confused.

4.1.2. Performance vs. Complexity

We offer high performance to player and when player push button game instantly chance page. Power-ups are the only thing that require more memory. We tried to keep memory allocation low because we want to get high performance . "Quantum Chess" is not require more space on computer 60-70 MB free space is enough to enjoy our game.

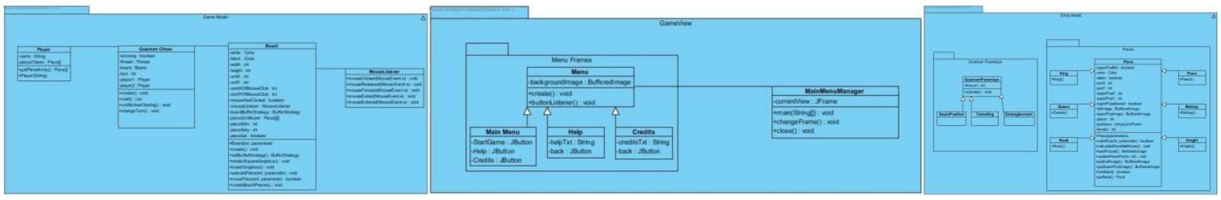
4.1.3. Responsiveness

All the screens in the game are easily changeable. For example from the "Main Menu" when user push "Credits", "Help" and "StartGame" screen changes instantly and user will not get bored. "Quantum Chess" do not distract the player's interest and entertainment.

4.2 Packages

We have 2 packages for two subsystems.Using these packages we divide our code into small subsystems. These subsystems help us to control and design system with more easily. When we have bug we can easily find. "Menu" packages contains menu screen such as "Credits" "Help" and "Startgame". Another packages for power-ups. We have three power-ups. Superpositions, Tunneling and Entanglenment.

4.3 Final Object Design



4.4 Mouse Listener

“Quantum Chess” is played by mouse. Using mouse, players can select “Credits”, “Help” and “Startgame”. In the game, players always use mouse to drag chess pieces. Also, “Credits” and “Help” uses the mouse actions and player can scroll down in the page using mouselistener.