**2. Software Architecture**

**2.1. Overview**

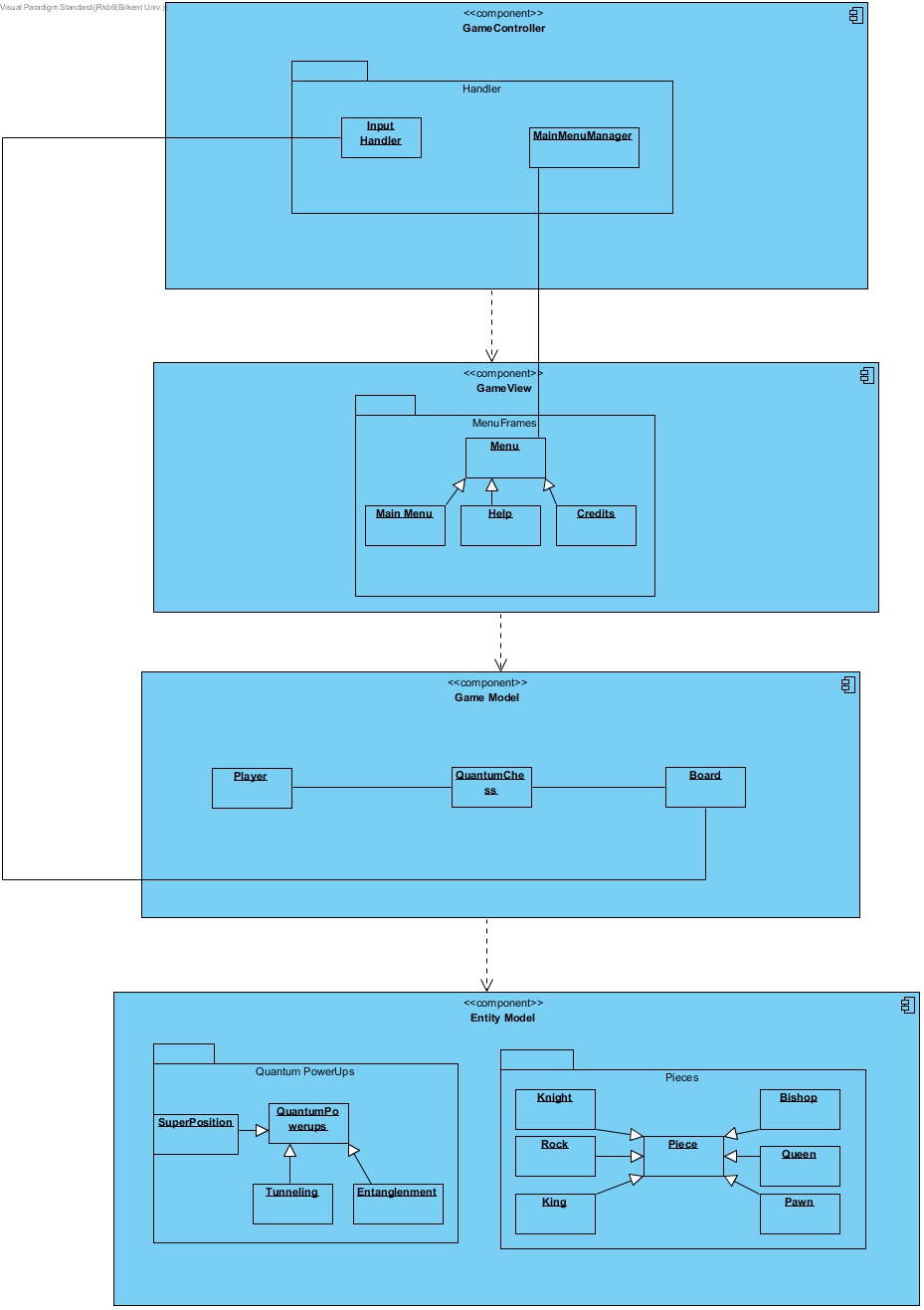
In this section, we will start detailing the composition of our system to a higher level. It has decided to divide system into smaller subsystems in order to carry out the maintenance of code.

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

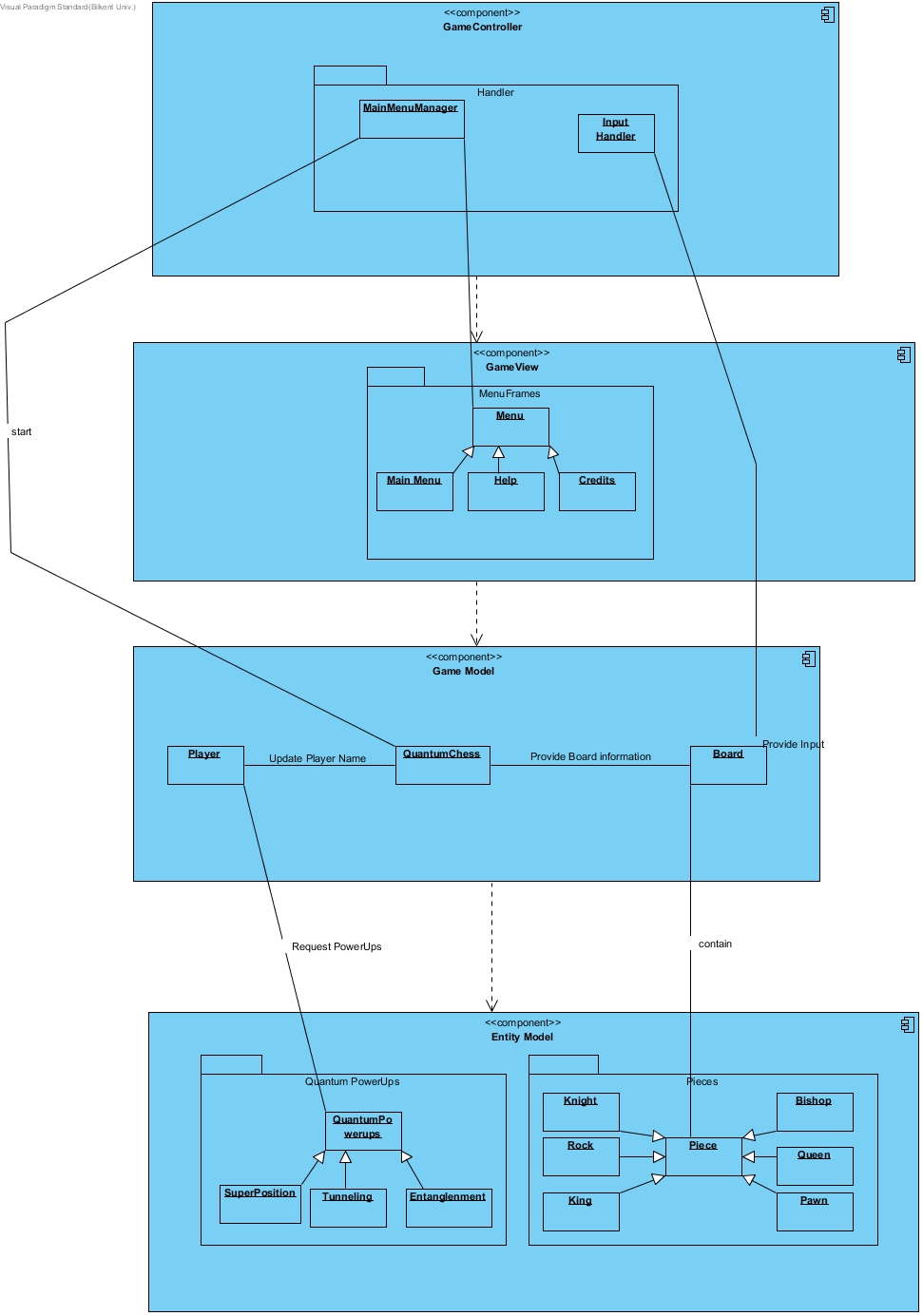


Figure-2.2. The detailed subsystem decomposition

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.