Concordia University

Department of Computer Science and Software Engineering

Advanced Programming Practices

SOEN 6441 – Winter 2020

Software Architecture & Methodologies

Team **Java Bean**

Jatan Gohel	4007 8112
Siddhant Arora	4008 5538
Krunal Jagani	4005 6939
Harsh Vaghani	4008 4099

Guided by

Dr. Rodrigo Morales

Architectural Design Document

Introduction

Our purpose is to build the game system framework for turn-based strategy games implementing a Model View Controller (MVC) architectural design model. We have studied and made sure we follow the extreme programming approach for the smooth development of software by implementing features like:

- **Planning:** Instead of only focusing on short term goal (build 1) we decided the whole schema architecture of the game and determined what needs to be done in first build. We studies various board games to extract out common features among them and decided to develop those common features in our first build.
- **Testing:** Testing is done by writing unit test cases for each modules and determined the veracity of the code we have developed. We have used Junit framework for this purpose.
- **Collective Ownership:** Collective Ownership encourages everyone to contribute new ideas to all segments of the project. All group members have made simultaneous changes and bug fixes of their own code as well as on the code written by other group members.
- **Simple Design:** Complex designs were avoided in order to make the software more user- friendly, and to also avoid faults. Smooth features were implemented with the help of Simple designs which resulted in quick, structured and clear game flow.
- **Continuous Integration:** Git was used as Revision Version Control which made all the group members up to date. Maintenance of concurrent changes, rollback sequences were maintained through local branches and master branch. Errors and conflicts detected at a time of integration were rectified quickly.

Model View Controller

Model View Controller architecture aims for separation of Concerns, meaning the components should more than one thing by dividing it into three parts a Model, a View and a Controller.

View: It is responsible for displaying all or portion of the data to the users. If the model data changes, the view must update its presentation as needed. This can be achieved by using a push model, in which the view registers itself with the model for change notifications, or a pull model, in which the view is responsible for calling the model when it needs to retrieve the most current data.

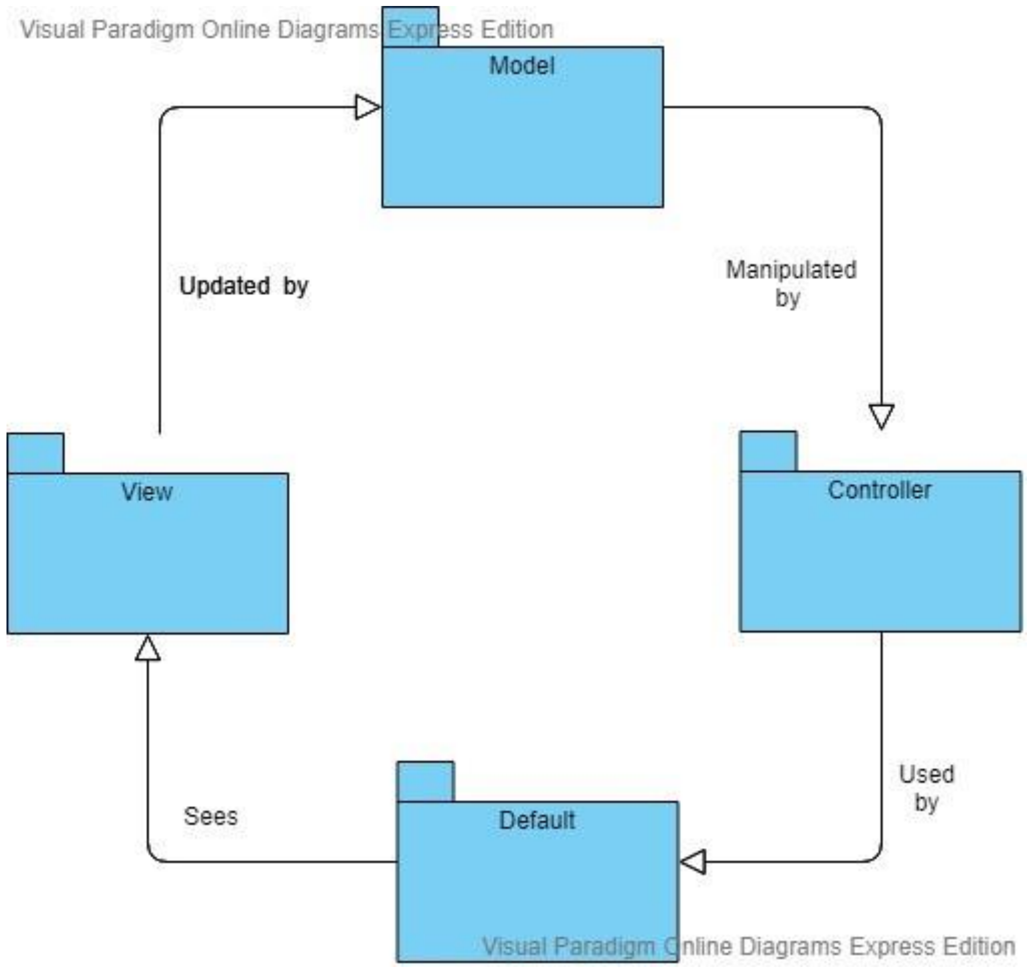
Visual Paradigm Online Diagrams Express Edition
Model
Updated by
Manipulated by
View
Controller
Sees
Default
Used by
Visual Paradigm Online Diagrams Express Edition

Fig. 1. Package Diagram

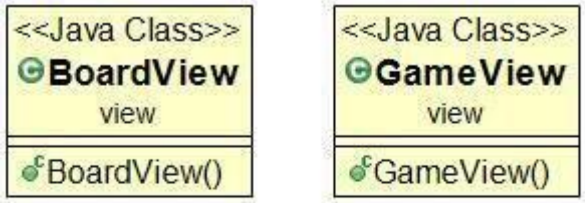

Fig. 2. View Class Diagram

Model: It is the lowest level of the pattern which is responsible for maintaining data. In enterprise software, a model often serves as a software approximation of a real-world process. Here is a class diagram of different models and their interconnections.

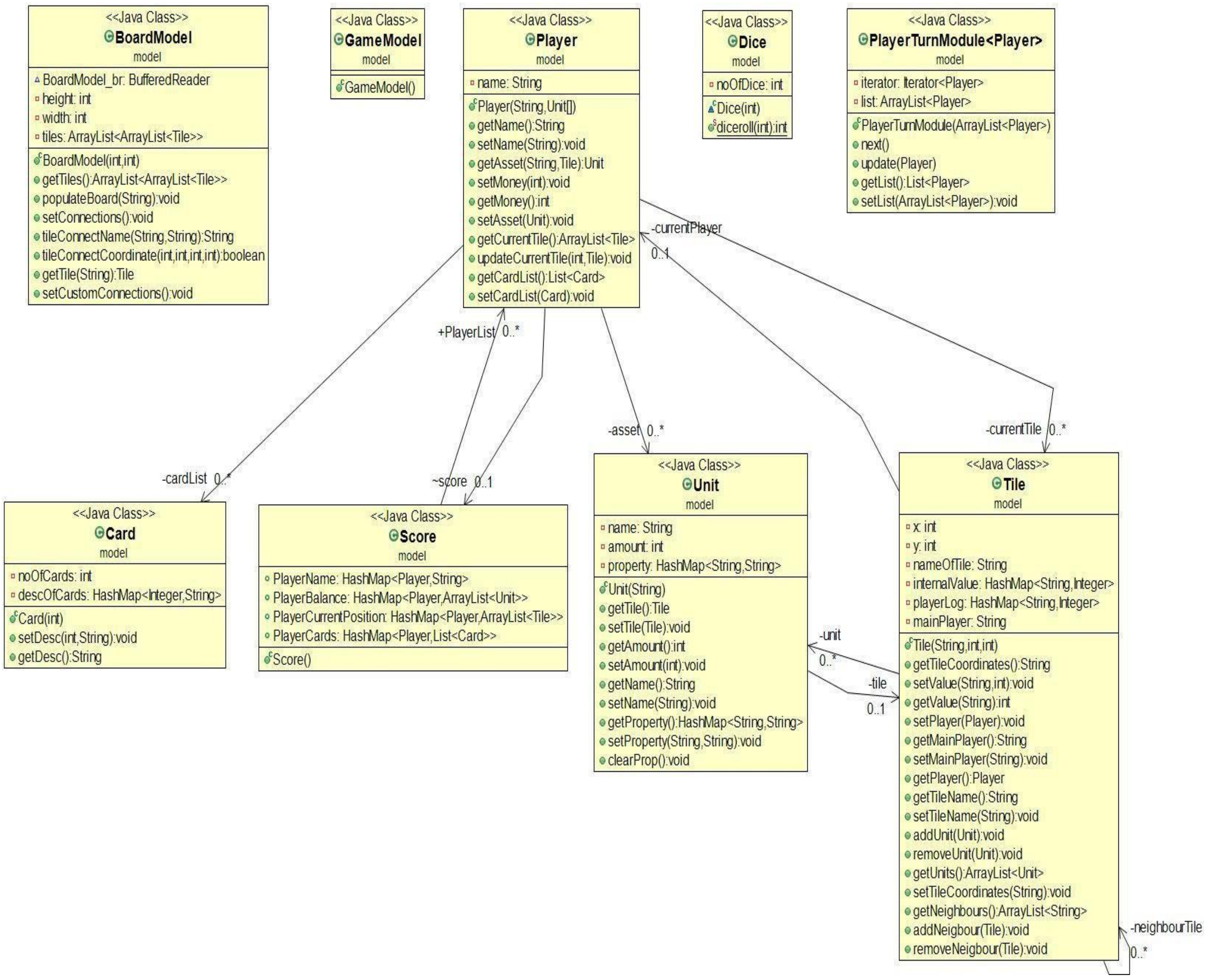

Fig. 3. Model Class Diagram

Controller: It is a software code that controls the interaction between the model and the view. The controller translates the user's interactions with the view into actions (ActionListener) that the model will perform. In a stand-alone application, user interactions could be button clicks or mouse over events. A controller may also change the view as and when the action wants.

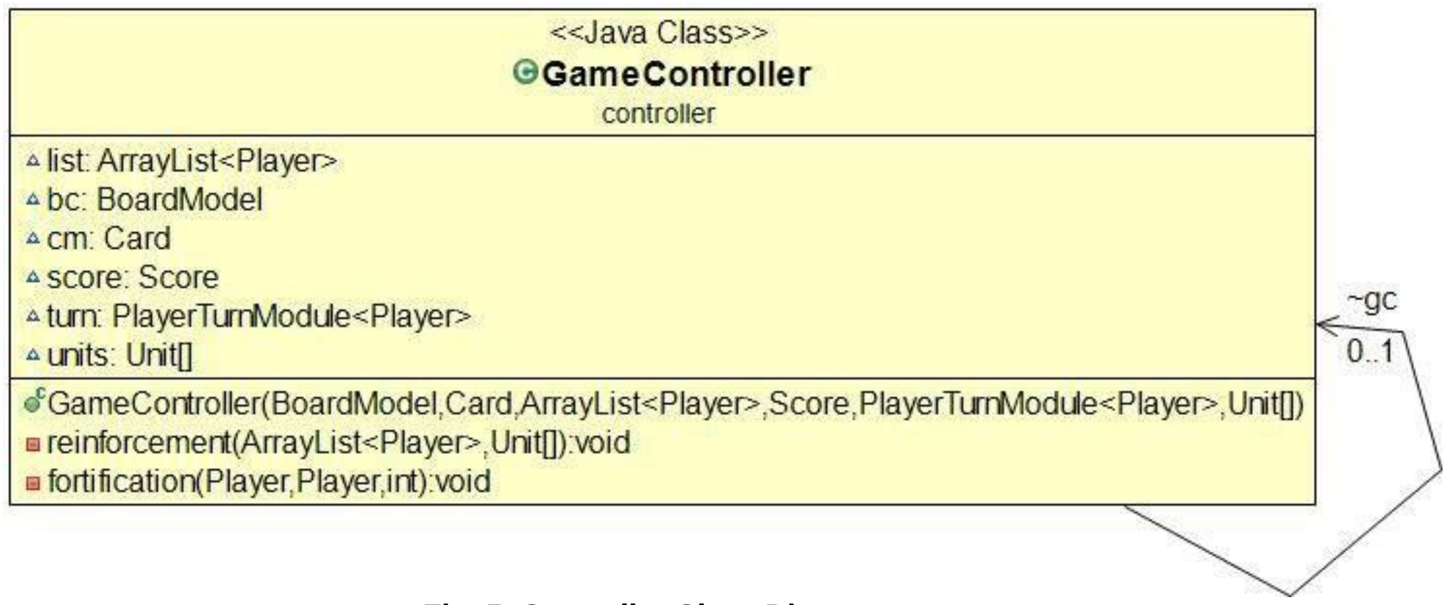
<<Java Class>>
GameController
controller
list: ArrayList<Player>
bc: BoardModel
cm: Card
score: Score
turn: PlayerTurnModule<Player>
units: Unit[]
GameController(BoardModel,Card,ArrayList<Player>,Score,PlayerTurnModule<Player>,Unit[])
reinforcement(ArrayList<Player>,Unit[]):void
fortification(Player,Player,int):void
~gc
0..1

Fig. 5. Controller Class Diagram

Models *(boardmodel, card, score, Player,PlayerTurnModule etc.)* manages the data of the application domain. If the model gets a query for change state from the Views, they respond to the instruction via Controllers.

Views *(MainMenuScreen)* on the other hand renders the model into a form suitable for visualization or interaction, in a form of UI (user interface). If the model data changes, the view must update its presentation as needed. In our case, it is implemented using Java FX.

Controllers *(GameController, MainMenuScreenController etc.)* are designed to handle user input and initiate a response based on the event by making calls on appropriate model objects. Thus, accept various input from the user and instruct the model to perform operations.

- The controller translates the user's interactions with the view it is associated with, into actions that the model will perform that may use some additional/changed data gathered in a user-interactive view.
- Controllers are also responsible for invoking new views upon conditions.

References

- http://www.oracle.com/technetwork/articles/javase/index-142890.html
- https://mdn.mozillademos.org/files/16042/model-view-controller-light-blue