**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:

* **The Planning Game:** The build plan gives sufficient idea to clients of results. This also determines the goals of each small release. This represent the detailed plan for the current build, assigned tasks and making the updated plan as the latest development will be release.
* **System Metaphor:** The XP developers to replace the standard project architecture used in traditional software development methodologies. The names of these parts of code cannot be complex.
* **Testing:** Testing is an integrated activity. The development team needs continual feedback, with the customer expressing their needs in terms of tests, and programmers expressing design and code in terms of tests. The tester will play both the customer and programmer roles.
* **Collective Ownership:** Collective Ownership encourages everyone to contribute new ideas to all segments of the project. Any developer can change any line of code to add functionality, fix bugs, improve designs or refactor. It's almost inconceivable that an entire team can be responsible for the system's design.
* **Pair Programming:** There were modules where more than two programmers worked on the sametasks to complete the functionalities. This not only increased the pace, but also the productivity. It efficiently resulted in knowing our responsibilities too.
* **Simple Design:** Complex designs were avoided in order to make the software more user-friendly, andto 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 programmers onthe same branch. Maintenance of concurrent changes, rollback sequences increased the productivity by automating integrating tasks. Errors detected by integration were rectified quickly.
* **Coding Standards:** All the programmers were on the same page to follow all the coding standardsincluding the naming conventions, File organization, etc. which helped us to peer reviewing and maintaining the source codes. Because of the standardized implementation, the sustainability of the source codes would be helpful in further builds.
* **Refactoring:** Refactoring is the technique of improving code without changing functionality. Refactoring is an ongoing process of simplification that applies to code, design, testing, and XP itself. In XP, developers will be refactoring during the entire process of development.

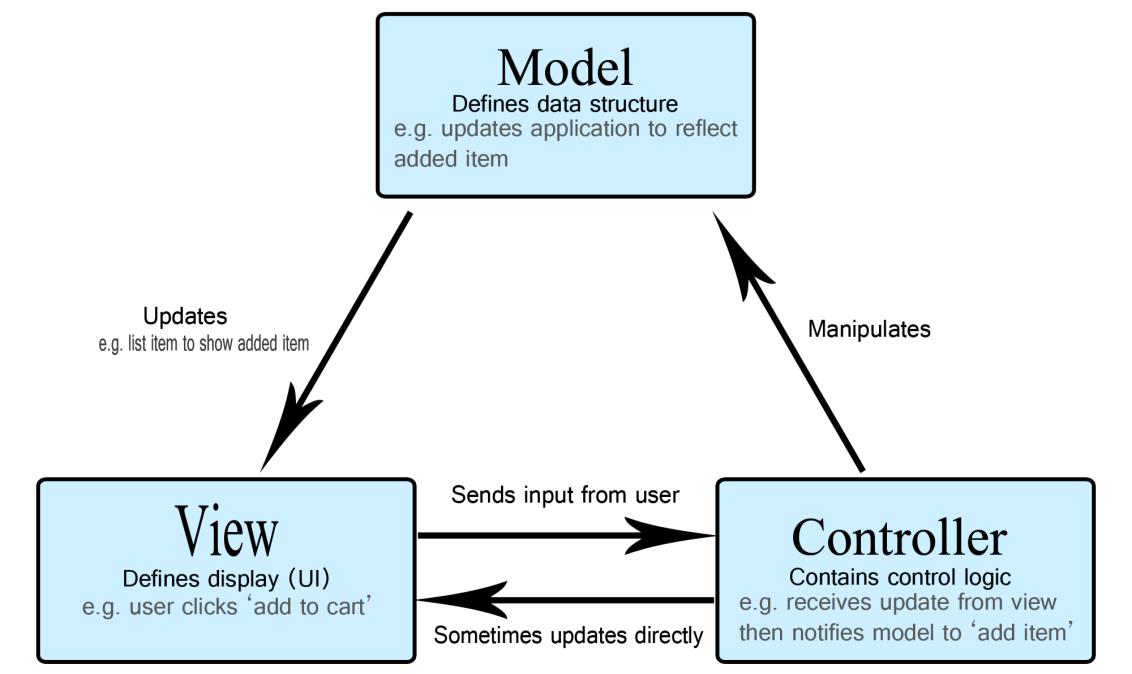
**Model View Controller**

Model View Controller architecture aims for separation of Concerns, meaning the components should not do more than one thing by dividing it into three parts a Model, a View and a Controller.

**Model**: It is the lowest level of the pattern which is responsible for maintaining data. In enterprise software, amodel often serves as a software approximation of a real-world process.

**View**: It is responsible for displaying all or portion of the data to the users. If the model data changes, the viewmust 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.

**Controller**: It is a software code that controls the interaction between the model and the view. The controllertranslates 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.



**Fig. 1. Basic MVC architecture**

As shown in Fig. 2, this is how MVC has been implemented in the project.

* **Models** *(BoardModel, Card, GameModel, PlayerTurnModel, etc.)*manages the data of theapplication domain. If the model gets a query for change state from the Views, they respond to the instruction via Controllers.
* **Views** *(BoardView, GameView)*on the other hand renders the model into a formsuitable 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, Transaction etc.)*are designed to handle user inputand 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.