**SOEN 6441- Fall 2014 “Tower Defense”**

**Deliverable 1**

**Team 8**

**Dr. Paquet Joey**

|  |  |
| --- | --- |
| Name | ID |
| Wang, Guan | 7225229 |
| Sokke Nagaraj, Pavan | 7608144 |
| Karnwal, Arjun | 7592043 |
| Balekuttira, Kariappa Thimmaiah | 7420269 |
| Baiazid, Reem | 5782465 |
| Singh, Kanwaldeep | 6756581 |

**Introduction**

Design document outlines the architectural and component design of the Tower Defense Game; it combines textual descriptions and UML class diagrams. The architectural design used is MVC architecture, internal design of each of its subsystem model, view and controller and the reason for this design.

**Functional Requirements**

The functional requirements for deliverable 1 of the game include:

1. Saving a map to a file.
2. Loading a map from an existing file, then edit the map.
3. User driven interactive creation of a map
4. User-driven allocation of grid elements such as scenery, path, entry point and exit point.
5. Verification of map correctness before saving
6. Game starts by selecting a saved map, then loads the map
7. User-driven placing of towers on the map, following the game’s restrictions
8. Implementation of currency and cost to buy or sell a tower
9. Implementation of towers’ level-dependent characteristics such as level, cost to increase level, refund rate, range, power, rate of fire, special effects, etc.
10. Tower inspection window that shows its current characteristics.
11. Tower inspection window allows to sell the tower.
12. Tower inspection window allows to increase the level of a tower, changing its characteristics.

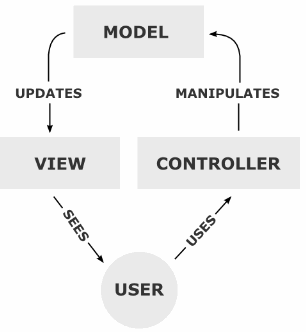
**Architectural Design**

The architecture of our system adopts the widely used Model-View-Controller (MVC) architecture, this model is chosen for our project since its architectural pattern separates the representation of information from the user's interaction with it that is the source code has no reference to the Controller or the view. As a result, a clear separation of concern between the logic, data and presentation is achieved. There are many advantages of highly cohesive software architecture such as maintainable system where modifying one component doesn’t necessarily require changing the other components. In addition, MVC architecture is useful for developing in a team environment because it allows the three components to be developed simultaneously allowing for a very flexible and rapid development.

A **model** notifies its associated views and controllers of state changes. This notification allows the views to produce updated interface, and the controllers to change the available set of commands.

A **view** registers the controller to receive user interface events and requests from the model the information that it needs to generate an output representation.

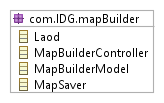
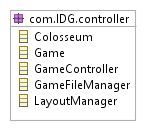
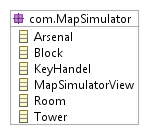
A **controller** handles the user`s input and can send commands to its associated view to change the view's presentation of the model. When the change is purely cosmetic we update the view.



#### Figure 2.1 – Model View Controller Diagram

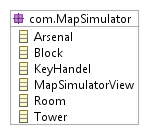
Subsystems Design

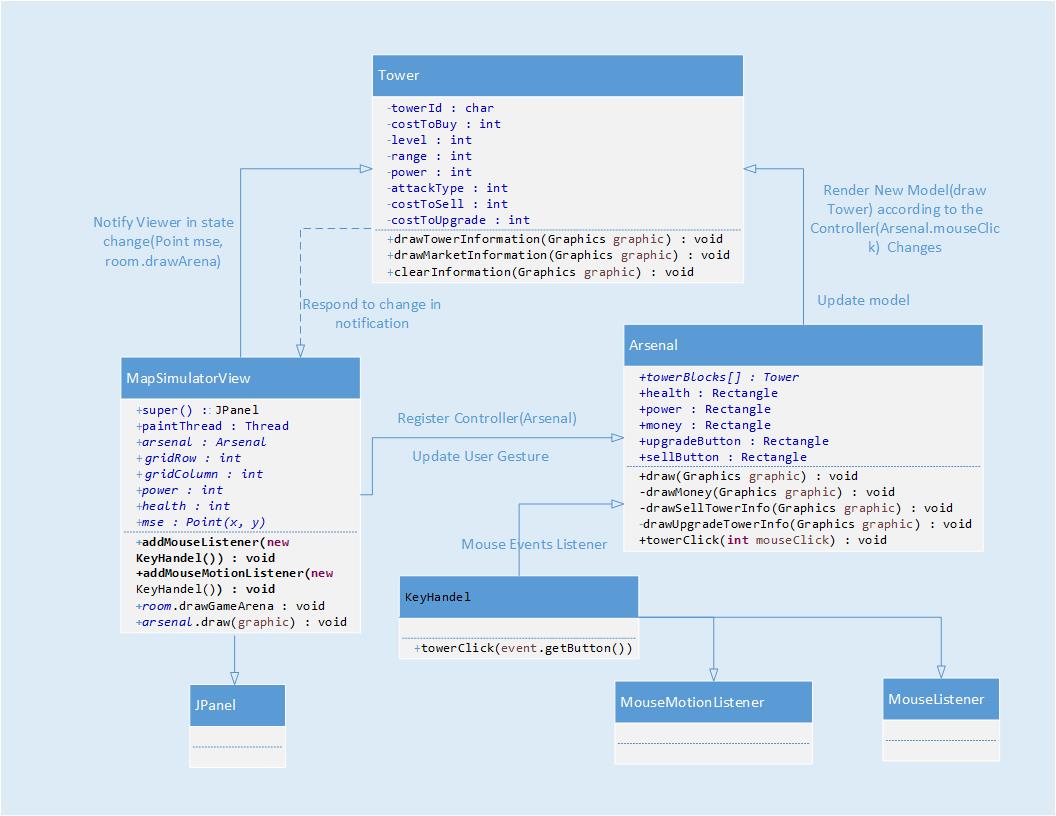
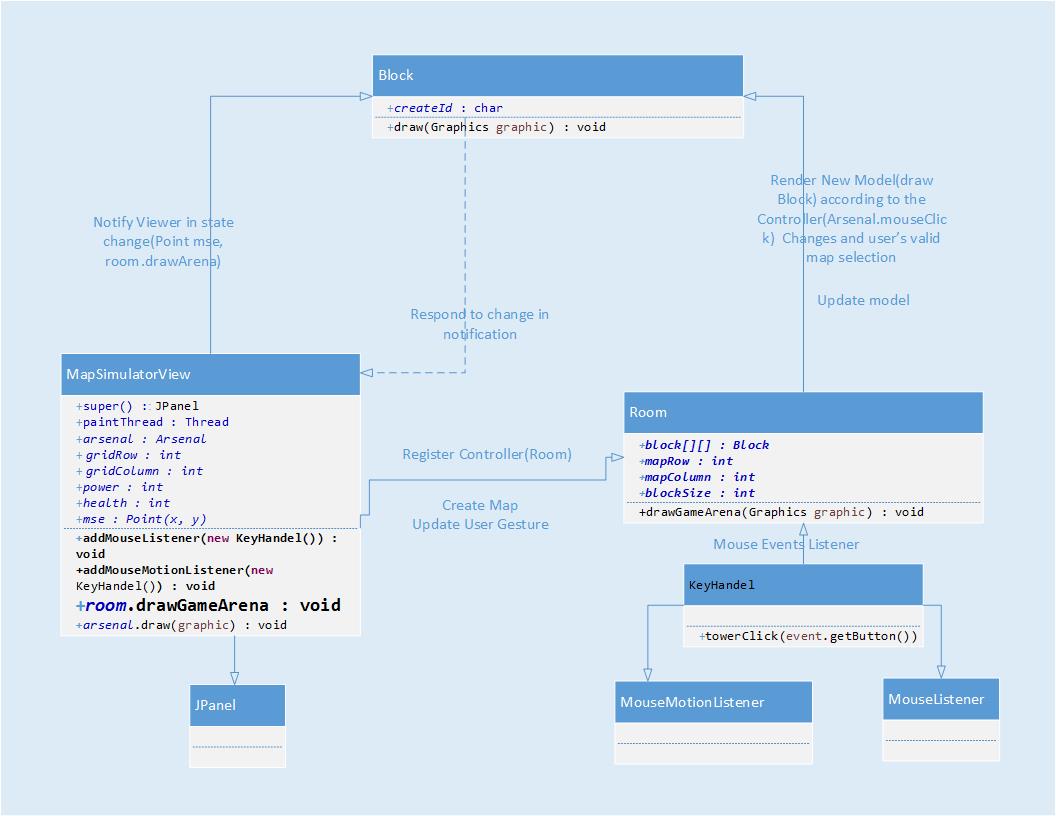
These are the packages in the system and their related classes



UML Class diagram for MVC Architecture

package: com.IDG.mapSimulator





Testing

Unit testing applied to our system is the JUnit Framework for the Java Programming Language. Since its time consuming to test all the methods in the system as the large number of methods in our system is large, only selected sets of methods were tested and the method chosen test the most important aspects of the code.