System design document for Alchemy-defense

authors

date
version

1 Introduction

1.1 System introduction

Alchemy Defense is a tower defense game implemented written in java. The application is structured according to the MVC pattern and makes use of the JavaFX framework to present the graphical user interface. The backend of the application aims to be as modular as possible, enabling seamless exchange of frontend implementation. Further more, the application strives to enable easy extension of future functionality and possible domain objects such as additional enemies and towers, or even new type of objects that can function within the game.

1.2 Definitions, acronyms, and abbreviations

Tower: Placed by the player upon the map to hinder the foes from reaching a set end position, where upon reaching the foe will inflict damage to the players hit points. Board: The main stage of the game. Uses a underlying 2D-grid which can contain and update both enemies and towers. Definitions etc. probably same as in RAD

2 System architecture

The system architecture is structured according to the MVC pattern and can be divided into three separate parts with distinct responsibility.

2.1 Model

The model contains all game logic and tracks each ”living” game object and their state. All computations such as calculating tower range and allocating damage to enemies are done within the model using the standard Java library to minimise external dependencies. Within the application, the model has no external dependencies other than itself.

2.2 View

The controller is the entrance class of the application and should be set up with a launcher. It sets up both the model and the view which it acts as a mediator in between. When the user submits input to the view, the controller parses these and triggers various game events within the model. Vice versa, when the model is updated the controller acknowledges this and translates the current game state to view that renders the appropriate images for the user to communicate the game state.

2.3 Controller

The view only connects to the controller. Its responsibility are restricted to injected images from the controller and send out user input events for the controller to parse. No logic should be contained here.

2.4 Basic flow of the application

TODO

3 System design

3.1 Project UML diagram

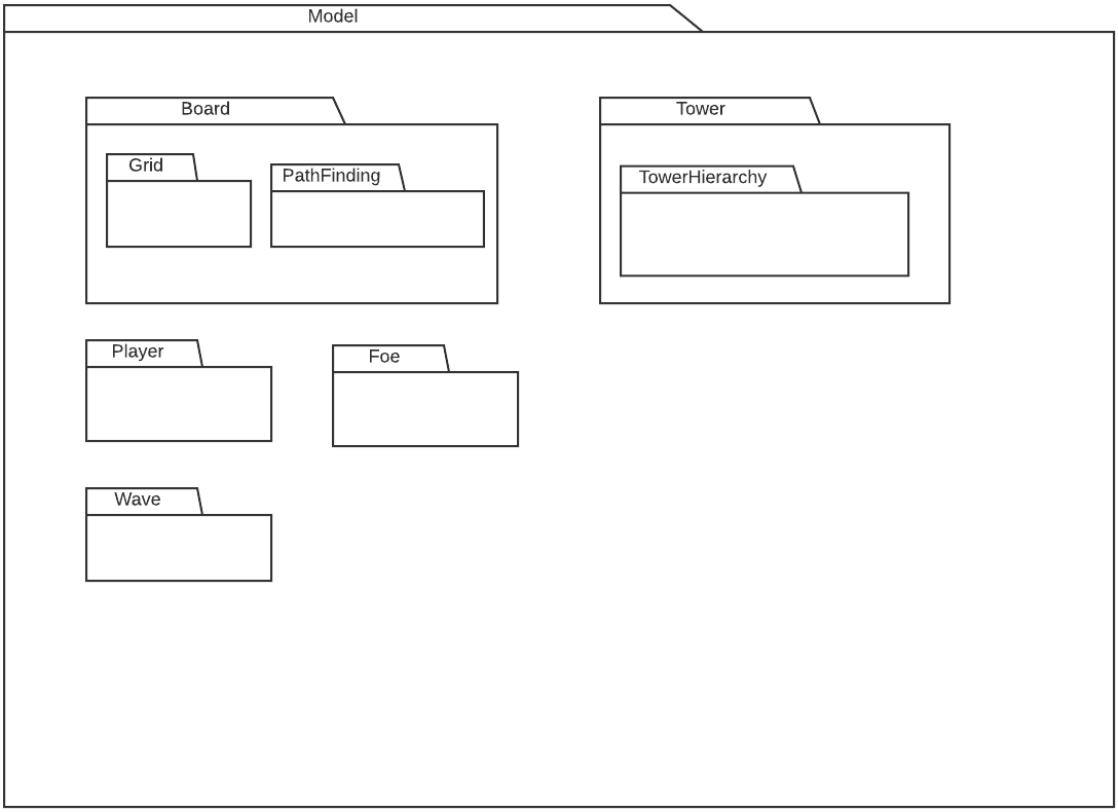


Figure 1: Project UML

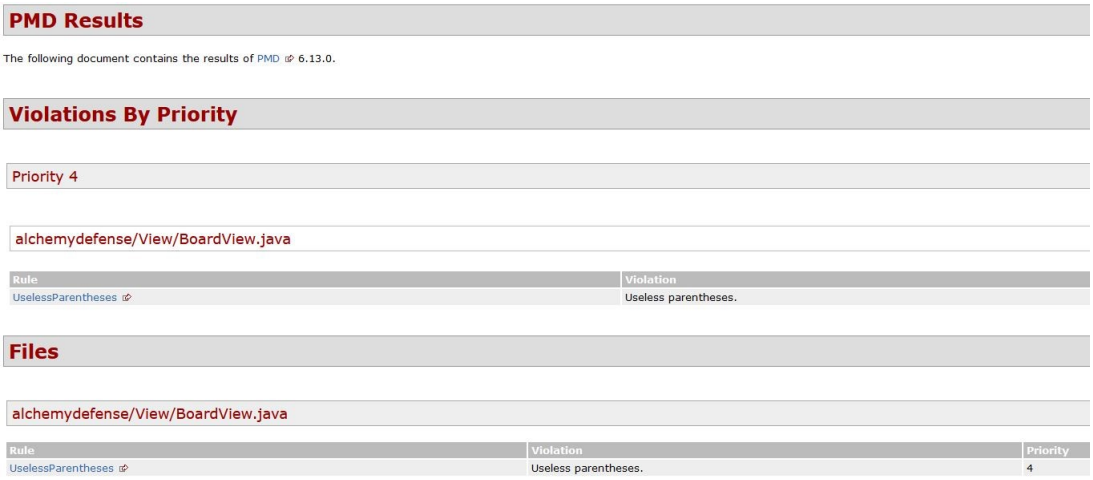
4 Persistent data management

If your application makes use of persistent data (for example stores user profiles etc.), then explain how you store data (and other resources such as icons, images, audio, etc.).

5 Quality

- Describe how you test your application and where to find these tests. If applicable, give a link to your continuous integration.

- List all known issues.
- Run analytical tools on your software and show the results. Use for example:
  - Dependencies: STAN or similar.
  - Quality tool reports, like PMD.



PMD Results

The following document contains the results of PMD 6.13.0.

Violations By Priority

Priority 4

alchemydefense/View/BoardView.java

Rule	Violation
UselessParentheses	Useless parentheses.

Files

alchemydefense/View/BoardView.java

Rule	Violation	Priority
UselessParentheses	Useless parentheses.	4

Figure 2: Caption

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility, used by ..., uses ..., etc.

5.1 Access control and security

If you applications has some kind of access control, for example a login, of has different user roles (admin, standard, etc.), then explain how you application manages this.

6 References

List all references to external tools, platforms, libraries, papers, etc. The purpose is that the reader can find additional information quickly and use this to understand how your application works.