System design document for Meteor Defense  
  
Table of Contents  
  
  
  
 Version: 0.0  
  
 Date 2014-05-12  
  
 Author   
  
This version overrides all previous versions.  
  
1 Introduction  
  
1.1 Design goals

The design should be as loosely coupled as possible to enablereplacement of any module in the architecture.  
  
1.2 Definitions, acronyms and abbreviations

* **GUI**, graphical user interface.
* **Java**, platform independent programming language.
* **JRE**, the Java Run time Environment. Additional software needed to run a Java application.
* **MVC**, a way to partition an application with a GUI into distinct parts avoiding a mixture of GUI-code, application code and data spread all over.
* **Screen**, class from the library libGDX representing a **Controller** (see MVC pattern) in the applications hierarchy.
* **Renderer**, class from the library libGDX representing a **View** (see MVC pattern) in the applications hierarchy.
* **Visitor**, pattern to make the code open for extentions and to avoid the use of “instanceof”.

2 System design  
  
2.1 Overview   
The application will use a modified MVC model, where Controllers are called Screens and Views called Renderer to match the library libGDX “namngivningar” (<-- ändra). A modified version of the Visitor pattern is also used to enable extentions.

2.2 Software decomposition

The models funtionality will be exposed by the interface IGameModel.   
  
2.2.1 General  
  
Package diagram. For each package an UML class diagram in   
appendix

* MeteorDefense is class holding main-method, application entry point.
* factory, classes for composing of larger systems of classes.
* model, the core object model of the game. Model part of MVC.
* armoryitem, subpackage to model cointaining the various ArmoryItem classes for organisation only. Model part of MVC.
* meteor, subpackage to model cointaining the various Meteor classes for organisation only. Model part of MVC.
* renderer, view classes of the MVC.
* screen, control classes of the MVC.
* service, classes for loading and saving data.
* test, classes for testing of the model.
* util, various util classes for sound, fonts, constants etc.

2.2.2 Decomposition into subsystems   
  
2.2.3 Layering  
  
2.2.4 Dependency analysis  
  
2.3 Concurrency issues

NA. Single threaded application. The thread is provided by the library libGDX that the application is dependent on. No concurrency issues will be raised.

2.4 Persistent data management  
The persistent data will be stored in “.txt” files both on desktop and Android with the use of JSON.

2.5 Access control and security

NA

2.6 Boundary conditions   
  
3 References

1. MVC, see <http://sv.wikipedia.org/wiki/Model-View-Controller> (2014-05-13)
2. JSON, see <http://www.json.org/> (2014-05-13)

APPENDIX