System design document for group 16

Erik Sjöström, Filip Labe, Jonatan Källman, Sarosh J. Nasir

May 17, 2017
v1.0

1 Introduction

1.1 Design Goals

Monster Clicker is a mobile game for Android. So the program must therfore run on Android, it must be able to get input from the user, and display the result of these inputs.

1.2 Definitions, acronyms, abbriviations

- *Monster Clicker* - The name of the project.

2 System architecture

Monster Clicker starts when the user opens the application. From there the user has the option of visiting several different activities. Each activity presents and represents different use cases such as, buying upgrades, viewing the map, attacking monsters, etc.
Monster Clicker ends when the user either exits the app or closes it using the android application manager.

2.1 Programming patterns

Monster Clicker, just like any other piece of software relies on different design/architecture patterns, such as:

- Game Loop [1]
- Model View Presenter [2]
- Singelton-pattern
- Factory-pattern
- Observer-pattern

2.2 Dependencies

Monster Clicker is a self contained piece of software, whch means that it does not depend on anything but itself. The application contains all the information required to be able to run.

The application is subdivided into several packages:

- *Player*
- *Monster Pack*
- *Map*
- *Stats*
- *Upgrades*
- *Clock*

Gem Clicker is developed for the Android operating system and is guaranteed to run on android phones running 4.0 or later versions of the operating system.

3 Subsystem decomposition

3.1 Player

This package contains a player model, and an interface for this model. The model contains the state of the player, damage, gold, etc. And methods to access and/or change the state. The interface provides a layer between the model and anyone who wishes to access it, only exposing non-internal methods.

3.1.1 Diagrams

Sequence diagram of getState()

3.1.2 Quality

The tests for *PlayerModel* can be found in */OOPP/app/src/test/.../oopp/PlayerModelTest.java*

3.2 Monster Pack

This package contains a monster model, and a monster factory. The model contains the state of the monster, gold, health, etc. A constructor to set the state and methods to modify the state. The factory handles the creation of monsters in a simplified way.

3.3 Diagrams

Sequence diagram of what happens when a monster is clicked.

3.3.1 Quality

3.4 Map

The package which represents the map activity in the app. From here the player can change area. Takes care of the area, level and map instantiation. The map package consists of the files:

- Area: Container class for Levels.
- Level: A class which contains information about the monster and which methods can be used to act on the monster. Plays a significant role as it contains the monster which you are fighting in the main activity.
- Map: The model.
- MapActivity: The view.
- MapPresenter: The presenter.
- areaType: A simple enum type.
- levelFactory: Produces levels.
- MapMVPInterface: The interfaces which regulate how the classes communicate internaly and how other packages can communicate with the the presenter.

3.4.1 Diagrams

3.5 Stats

The stats package presents the state stored in the PlayerModel in a readable way.
The stats package consists of the files:

- StatsActivity: The view.
- StatsPresenter: The presenter.

3.6 Home

The home package represents the home activity in the app. It consits of the files:

- Home: The model
- HomePresenter: The presenter
- HomeActivity: The view
- HomeMVPInterface: The interface which regulates how the presents communicates with the model and the view, and how everyone else can communicate with the presenter.

3.7 Shop

The shop package represents the shop activity in the app. It consits of the files:

- Shop: The model
- ShopPresenter: The presenter
- ShopActivity: The view
- ShopMVPInterface: The interface which regulates how the presents communicates with the model and the view, and how everyone else can communicate with the presenter.

3.8 Clock

The clock package is the implementation of the game-loop. The Runner class is implemented as a singelton so that everone that wants to register to the loop can get an instance of the runner.

3.8.1 Diagrams

3.8.2 Quality

4 Persistent data management

Gem Clicker saves the state of the app when the app is closed. The data is represented as a simple key-value storage, which is handled by android internaly.

5 Access control and security

There are no different roles for using this application. The only role is that of the user, and the only permission required of the user is to use the storage space of the phone.

6 References

References

[1] http://gameprogrammingpatterns.com/game-loop.html

[2] https://en.wikipedia.org/wiki/Model–view–presenter