**Software Architecture Document**

ANKH-MORPORK

**BUILD 3**

**Submitted by:**

7256264 : Kaur, Navleen

6355897:Hasaneen, Ayman

7248571 :Tanwar, Archana

6997791:Kashyap, Jatin

7734069:Eftekhari, Maryam

##### Introduction

## Purpose and Scope

This Software Architecture Document describes the architecture and design for the Ankh-Morpork game. This document aims to provide the high level information with the development of the game with sufficient information and references to relevant information to allow them to effectively support it.

## Document Evolution

It is not intended that this document be totally complete before development is completed and in fact it is expected to be updated and refined all through the development process of the next builds with design developed, refactored, and finalised.

The above said, changes to the document that occur later are expected to be to the more fine grained details of the java classes.

##### References

1. http://en.wikipedia.org/wiki/Discworld:\_Ankh-Morpork
2. <http://www.treefroggames.com/wp-content/uploads/rules/am-rules-eng.pdf>

##### Architectural Representation

The current build is the implementation of the Ankh-Morpork game with actual playing of 2, 3 or 4 human players. As similar to implementation in Build-1, the game status can be saved in an XML file and also game can be loaded from the given XML file.

In the current implementation the players cards, personality cards, random Event Cards are shuffled. Players have the choice of placing their minions in three regions (Dolly Sisters, The Scours and The Shades). One personality Card and five green-bordered Player cards are assigned to each player. Players take turns and the game plays on until someone wins.

Following is a brief description of each classes along with their intended purpose:

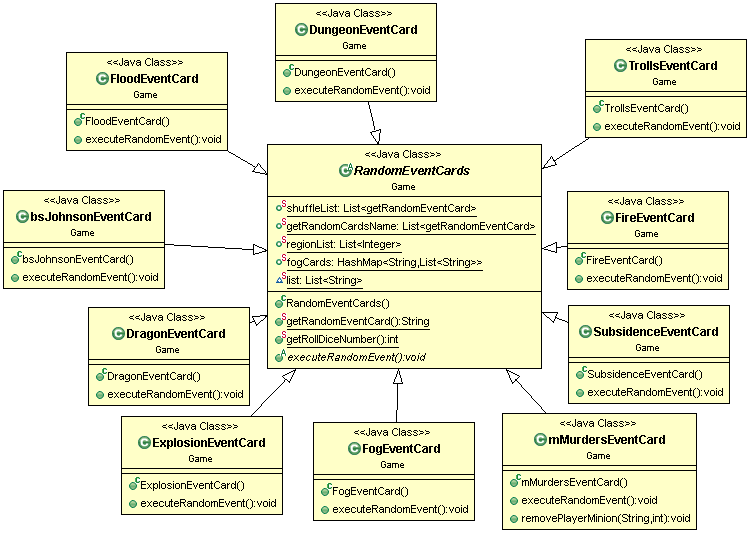
|  |  |
| --- | --- |
| Class Name | Purpose |
| GameEngine.java | Driver class of the game application.  This class is responsible for :   * Execution of the entire game. * Creation of objects for Player and Region Class. * Saving and Loading XML file to/from class objects. |
| NewGame.java | An object of this class is created when Player click on the button "Start New Game".  This class is responsible for :   * Initialization of GUI elements. * Populate objects of the Player Class. * Placing minions in the default area. * Display of City Area Card according corresponding to selected. * Display of Personality Card according to selected. |
| Pair.java | This class is a utility class.  This class is responsible for :   * Returning two values from single function i.e. color and list of player class. |
| PersonalityCards.java | This class is called from the GameEngine class to assign personalities to the Players.  This class is responsible for :   * Return of a random personality card. * Maintain consistency so that no player is assigned with the same personality. |
| Player.java | Player class is used to store the colour, number of minions and presence of building in the region for a particular Player. Number of objects created of this class is equal to number of players in the game.  This class is responsible for :   * Maintaining Player Info like colour, personality, minions, buildings, cash etc. * Handling of the Random Events. * Checking the winning condition related to personality. |
| PlayerCards.java | This class is called from the Players class to assign cards to the players.  This class is responsible for :   * Creation of deck of brown and green cards and shuffling it. * Returning card numbers for each player at random. * Maintain consistency so that no player is assigned with the same cards. * Moving minion of from one region to another. * Checking the Interrupt Card. |
| PlayerStatus.java | This class is used by the Region class to store regionwise data for each player. It keeps track of which player has minion or building in a region. |
| RandomEventCards.java | It is an abstract class used to get the names for the Random Event Cards. Separate classes for each RandomEvent are created as concrete classes.  This class is responsible for :   * Return of a random RandomEventCard. * Maintaining consistency so that no card is drawn twice. |
| Region.java | This class is called from the GameEngine class to assign default values to the all the twelve different regions like Region Name, Region Number, Building Cost etc.  This class is responsible for :   * Maintaining Region Info like number of minions, demons, trolls, existence of building, trouble Marker. * Perform function relevant to the region selected. |
| RegionStatus.java | This class is used from the Player class to store playerwise data for each region. It keeps track of which region has minion or building of which player. |
| SavedGame.java | This class object is created when Player click on the button "Load Game".  This class is responsible for :   * Creation of GUI for showing Players, Regions and other Information by reading XML file. * Loading of GUI. * Populate objects of the other class by the reading XML data file. |
| CityAreaCards.java | This class is called from the GameEngine class to get the names for the City Area Cards.  This class is responsible for :   * Returning a City Area Card for the specific city. * Maintain consistency so that no card is given twice. * Perform action according to the city selected. |
| bsJohnsonEventCard.java | This class is responsible for executing Random Event Bloody Stupid Johnson functionality. |
| DragonEventCard.java | This class is responsible for executing Random Event Dragon functionality. |
| DungeonEventCard.java | This class is responsible for executing Random Event Dungeon functionality. |
| ExplosionEventCard.java | This class is responsible for executing Random Event Explosion functionality. |
| FireEventCard.java | This class is responsible for executing Random Event Fire functionality. |
| FloodEventCard.java | This class is responsible for executing Random Event Floodfunctionality. |
| FogEventCard.java | This class is responsible for executing Random Event Fog functionality. |
| mMurdersEventCard.java | This class is responsible for executing Random Event Mysterious Murders functionality. |
| FogEventCard.java | This class is responsible for executing Random Event Fog functionality. |
| SubsidenceEventCard.java | This class is responsible for executing Random Event Subsidence functionality. |
| TrollsEventCard.java | This class is responsible for executing Random Event Trolls functionality. |
| Assets\_Extractor.java | This class is responsible for creating the spreadsheets for images. It does memory management by including all images into one file. |
| Assets\_Loader.java | This class is responsible for loading the board image using File I/O streams for images. |
| AssetsDepot.java | This class is responsible for cropping images for minions/trolls/buildings/demons etc. |
| AssetManager.java | This class is responsible for updating the board. |

##### Design Pattern Implementation

**Factory pattern** is one of most used design pattern in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

We have created RandomEvent.java abstract class. And we extend this class to create specific Random Events like DungeonRandomEvent.java.



**Observer Pattern** is used when there is one to many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioural pattern category.

Observer pattern uses three actor classes Subject, Observer and Client. Subject an object having methods to attach and de-attach observers to a client object. We've created classes Subject (GameUtility.java), Observer classes (Player.java and Region.java). Client is PlayerCards.java.

##### Deployment View

Following actions need to be implemented in succession:

* Extract the Zip file.
* Import it as an existing Java project into your Java Workspace.
* Sometimes it might throw an error for: jdom-2.0.5.jar . Please include the same in your project and change the classpath if required.
* Run the GameEngine class as a Java application and play the game.

##### 

##### Overview

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]

This UML class diagram shows the classes, interfaces, and their relationships.

