**Software Architecture Document**

ANKH-MORPORK

**BUILD 1**

**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 software architecture of Ankh-Morpork game is concerned with selection of number of Players and how players place their minion in three regions (Dolly Sisters, The Scours and The Shades).

[This section describes what software architecture is for the current system, and how it is represented. Of the **Use-Case**, **Logical**, **Process**, **Deployment**, and **Implementation Views**, it enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

The game also gives an option :

* To save the status of the current game XML file : Player is allowed to save the status of the current game in the player specified XML file.
* To load the game from a given XML file : Player is having the option to load the game from the given XML file . While loading from the XML file, objects of respective classes are initialised and populated with the data from the given XML.

To describe this software architecture the following are the Java classes with their purpose in brief :

|  |  |
| --- | --- |
| **Class Name** | **Purpose** |
| GameEngine.java | It is the main class for the Game Package.  This class is responsible for :   * Controls the execution of the entire game. * Creates object for Player and Region Class. * Saves and Load XML file to/from class objects. |
| NewGame.java | This class object is created when Player click on the button "Start New Game".  This class is responsible for :   * Creating GUI for showing Players, Regions and other Information. * Populate objects of the Player Class. |
| 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 :   * Returning a personality card at random. * Maintain consistency so that no player is assigned with the same personality. |
| Players.java | This class is called from the NewGame to assign default values to the Players.  This class is responsible for :   * Maintaining Player Info like color, personality, minions, buildings, cash etc. |
| PlayerCards.java | This class is called from the Players class to assign cards to the players.  This class is responsible for :   * Returning card numbers for each player at random. * Maintain consistency so that no player is assigned with the same cards. |
| PlayerStatus.java | This class is used from the region class to store regionwise data for each player. |
| RandomEventCards.java | This class is called from the GameEngine class to get the names for the Random Event Cards.  This class is responsible for :   * Returning a RandomEventCard at random. * Maintain 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 12 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. |
| RegionStatus.java | This class is used from the players class to store playerwise data for each region. |
| SavedGame.java | This class object is created when Player click on the button "Load Game".  This class is responsible for :   * Creating GUI for showing Players, Regions and other Information by reading XML file. * Populate objects of the other class by the data read from XML. |
| 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. |

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organise the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

# Deployment View

[This section describes one or more physical network (hardware) configurations on which the software is deployed and run. At a minimum for each configuration it should indicate the physical nodes (computers, CPUs) that execute the software, and their interconnections (bus, LAN, point-to-point, and so on.) Also include a mapping of the processes of the **Process View** onto the physical nodes.]

Main points to note are:

* Extract the Zip file.
* Open the same in your Java Workspace.
* Sometimes it might give 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.

# Implementation View

This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components [This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

This section describes the overall structure of the Java implementation

## 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 for Content Aggregator.

