**Discworld Ankh Morpork**

*Software Architecture Documentation*

# 1 - Introduction

**1.1****Purpose**

This document provides a comprehensive architectural overview of the Discworld Ankh Morpork board game, using a number of different architectural views to depict different aspects of the game. It is intended to capture and convey the significant architectural decisions which have been made on the system.

**1.2****Scope**

This Software Architecture Document provides an architectural overview of the Discworld Ankh Morpork. The Discworld Ankh Morpork is being developed by Group 7 – Advance Programming Practices in Concordia University to play the game on computer by 2 to 4 people.

This Document has been generated directly from the Discworld Ankh Morpork Analysis & Design Model implemented in \_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**1.3****Definitions, Acronyms and Abbreviations**

Need to write.

**1.4****References**

Applicable references are:

Need to write.

# 2 - Architectural Representation

This document presents the architecture as a series of views; use case view, logical view, process view and deployment view. There is no separate implementation view described in this document. These are views on an underlying Unified Modeling Language (UML) model developed using -------------------------------.

# 3.  Goals and Features

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

* Ability to create and populate the data structures required to model the state of a game.
* Program must initialize all data structures appropriately for the start of a new game, following all of the instructions given on page 3 of the [rule book](http://www.fantasyflightgames.com/ffg_content/kingdoms/support/KN20_Kingdoms_Rulebook_sm2.pdf).
* Ability to save the current game state to a file, in a format of your choice (note that the user must be able to specify the name of the file to save to)
* Ability to load the game state from a file (note that the user must be able to specify the name of the file to load from)
* Ability to display the game state in text mode
* Ability to model an entire game, with 2-4 human players (using only the green-bordered Player cards for now; the brown-bordered cards are an optional feature for this build)
* Your code for this build must incorporate at least one design pattern other than Singleton (i.e. if you include Singleton, the requirement is to incorporate two patterns).

# 4.  Use-Case View

A description of the use-case view of the software architecture. The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of iteration. It describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

The Ankh Morpork use cases are:

- Load an existing Game

- Start a New Game

- Assign Assets to each Players (minion, money etc..)

- Select A Player Card

- Action According to Player Card

- Winner Decision

These use cases are initiated by the student or the player actors.

**4.1****Use Cases Diagram**

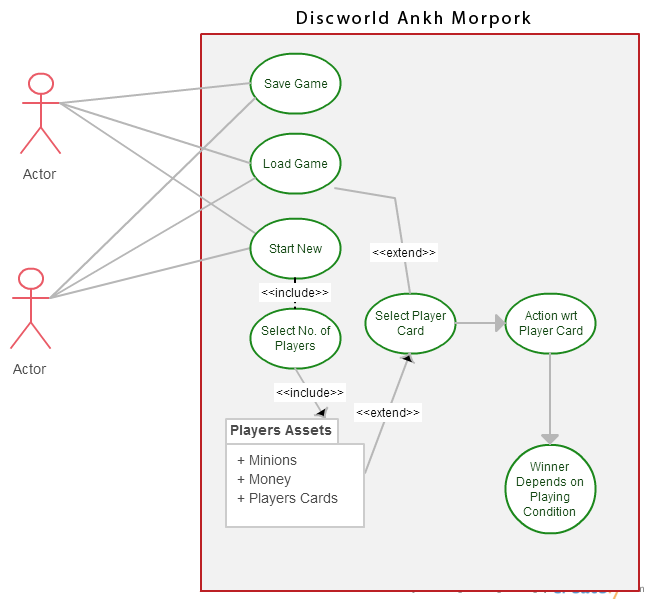


Diagram Name: Ankh Morkpork Use-Cases

*4.1.1 Save Game*

This use case allows a player to save the game if they need to leave the game at any stage. The action collects the information of all the players, their existing assets and objects on the board to save in file. Player has to choose a file and file is saved in json format.

*4.1.2 Load An Existing Game*

This use case allows a player to select an existing game saved previous to resume from the same position where it was left.

*4.1.3 Start A New Game*

It allows you to select if you have to start a new game and further moved to next step.

*4.1.4 Select Players*

This action allows you to select number of you players you want to play. Number of players could be any between 2 to 4.

*4.1.5 Players Assets*

It’s a kind of auto process where each players is assigned a color and according to that color player get 10$, minions, 5 players cards.

*4.1.6 Select Player*

On a player turn player choose a player card from his / her cards and does the action according after completing the action of all cards player can get more cards to maintain 5 cards with him / her.

*4.1.7 Action W.R.T Player Card*

Players do the action while using their playing cards that could be one or more action with respect to the card chooses by the player. There are various action such as Placing the minion, removing the building, Assassination, getting money from other player etc….

*4.1.8 Winner Decision*

With respect to personality card possessed by a plyer winner conditions are applied that make the decision of the winner.

**5.****Logical View**

A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers. Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

The logical view of the game comprised of the 2 main packages: User Interface, and Playing Conditions.

The User Interface Package contains classes for each of the action that the actors use to communicate with the System. Boundary classes exist to support Selecting players, assigning objects, resume the game, taking action with respect to card instructions, maintaining of other players info, adding or removing objects, maintaining personal game info, completing turn, and viewing cards.

The Playing Condition includes classes for the players to play according to the rules and regulation define in the rule book.