Software Design Description

Version 0.0.1

3/22/2015

Tales of White Wolf

Colin Benist

Rodny Joseph

Karl Kraus

# 

# Table of Contents

1. – Introduction
   1. Purpose
   2. Scope
   3. Glossary
   4. References
   5. Overview of Document
2. Deployment Diagram
3. Architectural Design
4. Data Structure Design
   1. Subsystem 1
5. Flow of Control
6. User Interface Design
7. Help System Design

# List of Figures

1. 30 Rooms
2. Monsters
   1. Forest Monster: Wolf, Bear, Giant Spider
   2. Dungeon Monster: Skeleton, Goblin, Undead Witch
   3. City Monster: Wild God, Bandit
   4. Boss: Dragon
3. Puzzles
4. Items: Axe, Sword, Potion, Spell,
5. Zones: Forest, City, Dungeon
6. Data Structure Design - Text File
7. Exception handling
8. Directions/Command: Flee, Examine, Attack, Help, inspect, etc..
9. Save Game
10. Help Menu
11. Exit option

# 1.0. Introduction

## 1.1. Purpose

The game is PG-13. The intended audience of this game is anyone over the age of 13, due to violence. Viewer discretion is advised. The game is adventure and action. Possible comedy may be included. It will be a one player game. In the object design, we describe the conditions under which an operation can be invoked and those under which the operation raises an exception. Boundaries between objects, type signatures, visibility, invariants, preconditions and post-conditions are specified. Missing attributes/operations are identified.

## 1.2. Scope

The game doesn’t have an external systems with which this system must interact besides the computer device being used. This can be, possibly, any computer system such as Windows 7, 8, 8.1, or Linux and Mac. The game may include a GUI interface if time permits. The game will be played using commands, such as “move forward” or “Exit room” command. The user will type these commands in the console and the command will be executed. Invalid commands will cause a throw of exception, prompting the user to enter in another command. Since this is a text base game, the story line will be text base, and this will be generated using a text file. Important information, such as user account information and saved games will be saved in the text file and updates will be made accordingly. If a player wants to exit the game, they will first save the progress and then go to the start menu, where they can then exit the game. If the game is not exited in this manner, then there are no guarantees that the game will save properly. Users will be prompt if they want to exit the game, and if the user agrees, then the game will shut down. When the user runs the game again, they will have the option to login in which will then prompt them whether they want to continue the saved game or start a new game. Then, if the command to continue the saved game is called, the save game will load.

## 1.3. Glossary

**Invoke** - To invoke a method on an object is to ask the object to perform a task.

**Exception** - Exception handling enables a program to deal with exceptional situations and continue its normal execution.

**Visibility** - A mechanism for specifying whether the attribute or operation can be used by other classes or not.

**Invariant** - A predicate that is always true for all instances of a class. Invariants are constraints associated with classes or interfaces. Invariants are used to specify consistency constraints among class attributes.

**Precondition -** a predicate that must be true before an operation is invoked. Preconditions are associated with a specific operation. Preconditions are used to specify constraints that a class user must meet before calling the operation.

**Post-condition** - a predicate that must be true after an operation is invoked. Post-conditions are associated with a specific operation. Post-conditions are used to specify constraints that the class implementor and the class extender must ensure after the invocation of the operation.

**Signatures** – Given an operation, the tuple made up of the types of its parameters and the type of the return value. Operation signatures are specified during object design.

**Use case** - General sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

**Sequence diagram** - Sequence diagrams represent the objects participating in the interaction horizontally and time vertically.

**Software architecture** – description of subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.

**Design goals** – description of the qualities of the system that developers should optimize

**Control flow** - The sequence of execution of operations in the system.

**Boundary condition** - A special condition the system must handle. Boundary conditions include start-up, shutdown, and exceptions.

## 1.4. References

N/A

## 1.5. Overview of Document

**Section 1** shows the introduction of the document, with its purpose and scope outlined. The purpose highlights the intended users as well as general information about the game. The scope allows the reader to have a clearer view concerning the design methodology of the game. This outlines the how the game functions, and how the game is saved. In Section 1 there is also a glossary of terms and any reference materials used.

**Section 2** shows the deployment diagrams. As mentioned, there are none.

**Section 3** shows the Architectural design

**Section 4** shows the Data Structure Design and any Subsystems involved. There are 3 subsystems, as mentioned in that section.

**Section 5** shows the flow of control for the game. There are several flow diagrams that are presented and explained.

**Section 6** shows the User Interface Design. If possible, a GUI will be implemented. Other useful information is explained in details in the section

**Section 7** shows the Help System Design. This design demonstrated how a user will interact with the help menu, which will be accessible throughout the game.

# 2.0. Deployment Diagram

N/A

# 3.0. Architectural Design

< A software system is a set of communicating entities that collaborate to perform a task. The Architectural Design shows these entities, their relationships and the relationship to the actors in the system. This top level is a diagram where each entity has a name, a (proto-) type, an abstract specification and an interface design. The abstract specification is a description of its purpose, its functionality, its attributes (including dependency on other entities) and the constraints under which it must operate. It also describes resources, that is, any elements used by the entity which are external to the design such as physical devices (e.g., printers), software services (e.g., math libraries) and processing resources (e.g., buffers). The interface design is the list of the services that it provides to clients. These services are methods (procedures and functions), each carefully documented using a pre-condition/post-condition formalism.

Each entity in turn may provide its services by having an internal architectural design with its own set of subordinate entities. These entities may be called sub-systems, components, modules or classes. The decomposition of a higher-level entity into subordinate entities must be explicit. The algorithm that shows how each method of the larger entity is performed by these components must be explicit. Any data stored in an entity must be explicitly described (see Data Structure Design below).

Note that while the abstract specification (architecture) and the interface (detailed) design are usually developed separately in an iterative approach, they are combined for documentation. >

# 4.0. Data Structure Design

< Design in detail and specify the data structures to be used in the implementation. If these include databases, define the table structure of all databases including full field descriptions and all relations. Graphical languages are appropriate. Note that a database is an object and may have been fully described in the previous section. Each subsystem in your program should be a subheading. >

## 4.1 Subsystem 1

4.1.1 Object Model

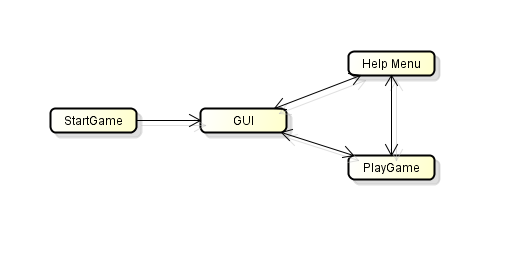
4.1.2 Description of Each object with attributes and methods described

### 

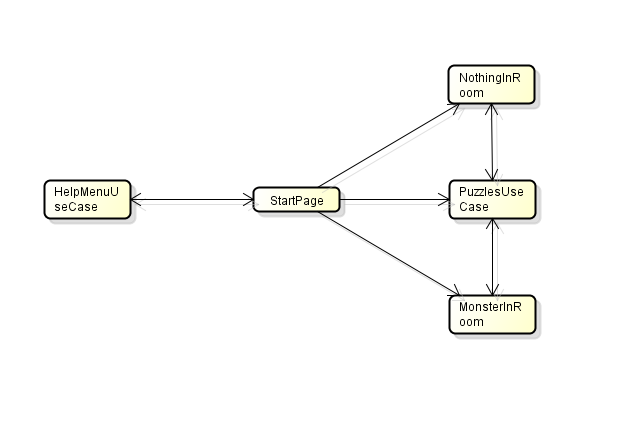
# 5.0 Flow of Control

The flow of controls are shown below and are displayed in the Flow Diagrams detailing layout of the game using sequence diagrams and use cases. The sequence diagrams and use cases are all in previous submitted assignments.

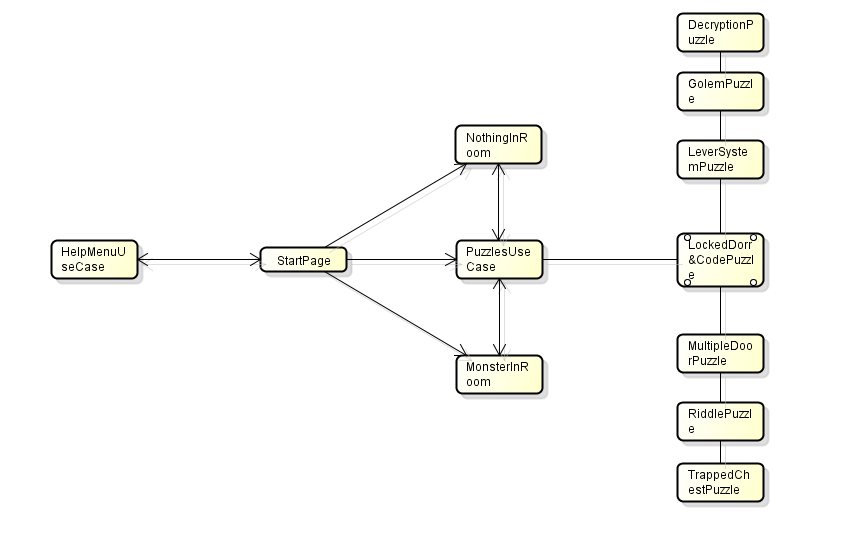
**Flowchart of the sequence diagrams.** The game will start, prompting the user to either login, or create an account, which is displayed in StartGame and GUI. Then after the user logs in or create an account, the game can be played. The Help Menu option is also available.



**Flowchart of the use cases.** From the StartPage, users will be able to go to either a room that is empty, a room with a monster, a room with a puzzle, or the help menu.



**Flowchart of use cases and different puzzles included.** Each puzzle room will have one of the many possible puzzles.



# 6.0 User Interface Design

< Describe the user interface. Prototype screenshots are acceptable if using a GUI. You should discuss the expected effectiveness of your design. >

# 7.0 Help System Design

< Describe the **structure** of the help system and how it is to be accessed. Will it be context sensitive? Will there be a system reference capability? >

# Index

< generate here >