Brian Bowen

Professor Karani

Programming III: Object Oriented Programming

21 February 2020

FINAL PROJECT PLANNING

**Scenario**

Imagine for a moment that you are a fictional writer. You have are planning on writing a story, but it is not best to begin with a blank word document and no plan. Writer’s block is a real thing. Besides having to not know how to write a story, the best-selling fictional novels are exquisitely planned out. Perhaps the best example of “world building” is J.R.R. Tolkien, who spent 10 years writing and editing his famous series *The* *Lord of the Rings*, and many, many years before that developing back stories, fake languages, and characters to be in the world he built. All of this he accomplished before he wrote a word of *The Lord of the Rings.* Now his book is the 3rd best selling novel of all time, only surpassed by *Don Quixote* and *A Tale of Two Cities* (<https://bestlifeonline.com/best-selling-novels/>). Sometimes creating interesting characters with a backstory is one of the most difficult things of writing a story, so to help out with this issue, I will be developing a Fantasy Humanoid Character Generator for Writers. It only does humanoid creatures, so it won’t generate a dragon, an orc, or other monsters, but it will help writers to be able to have ideas to get their creative juices flowing on the creation of their protagonists.

**Class Overview**

My program would be designed with the Object Oriented Programming Style, and consist of five main classes and eight subclasses. The nine main classes will be Person, Role, Adventure, Location, and the five large subclasses of Person (Elf, Hobbit, Dwarf, Human, and Wizard). Person is the biggest class, which holds all of the basic data needed for each of the fantasy races supported by this program, including name, age, hair color, etc. It has five subclasses which extend its description of a person, which consist of Elf, Dwarf, Hobbit, Wizard, and Human. Each of these have slightly different characteristics distinct from each other and those of the Person class. Person and its subclasses are the main classes used by the program, which stores all of the information to later be returned to the user. Each of the other classes are implemented in one way or another by the Person class or its children. Location is another class, which is used by both Person, Adventure, and each of the Abilities’ children. Essentially this is just the place on the map, an Adventure took place, a Person is from, or an Ability was created. Adventure is a class which describes the different adventures that a Person has been on. Again the user selects this before they select their character Race, and more Adventures can be added later in the editing phase. The final class is Role, and this essentially just describes the position of power that a character has – is he a king, a warrior, a lead wizard, a slave, etc. A more detailed class diagram is attached at the bottom of this document for more information.

**Structure and Function**

The general running of the program can be best described in an activity diagram below. In the “Generate” actions, each attribute of the class is randomly assigned, including any instances of other classes as class variables. In the Editing Phase, any of these randomly assigned attributes can be changed by the user (other than race, which is set in stone). A close up of a map

Description automatically generated

While this gives the general activity that takes place in the whole program as an overview, it can also be helpful to analyze the overall sequence that the program takes as it is running. The following sequence diagram shows the interactions between the five main classes A screenshot of a cell phone

Description automatically generatedwhen various functions are called.

Since this program is mostly a character generator, there are not very many actions a writer can take, simply because it is built as an aid to their creative juices. The three things that a writer can do with this program are: generate a character, edit an attribute, and export the character to a text document, so they can see the character later. The following diagram is a simple use case diagram for this program.

A close up of a map

Description automatically generated

**Business Processes**

There are several business processes that are demonstrated within this code, with most of the objects these include Generate, GenerateName, and the Person Object allows an Export. There is also an ability to view the history to be able to edit it. All of these things make up the multiple different types of business processes necessary to make this complex program possible.

A screenshot of text

Description automatically generated

**Complex Class Diagram**