CS 591 ASSIGNMENT – 5

(CAVE ADVENTURE)

**DESIGN DOCUMENT**

|  |  |  |
| --- | --- | --- |
| **NAME** | **BU ID** | **BU EMAIL ID** |
| Yashvardhan Das | U63030800 | yashvdas@bu.edu |

This document provides information about the probable Java classes that are to programmed in the code of this assignment. A brief summary of the overall functionality is described for each of the respective classes. As with most coding practices, minor changes or additions might be incorporated later on when the code is fully implemented.

1. **CaveAdventureMain Class**

* This is the primary class where all the three adventures along with the final mission are created systematically.
* The order of execution or the flow of the above-mentioned events are controlled from this class.
* This class also incorporates the functionality of checking the status of the respective adventure levels and that of the final mission so as to direct the user to move (or instruct the special cases scenario of singing) onto the next level in order.

1. **AdventureLevel Class**

* This class covers most of the happenings of the respective adventures.
* This class’ purpose is to serve as an encapsulation for the whole adventure’s functionality.
* Here, the activities of snoozing and penance are programmed into the respective methods.
* Instance variables for the monster and treasure are created here.
* The main advantage or benefit this class provides is that it hides most of the game’s internal functionality complexities from the user.
* It allows modifications that can be incorporated without letting the end user know about these intricate changes.

1. **Treasure Class**

* This class is used to implement the specifics of the “Treasure” object.
* On a very broad level, the *Treasure* object consists of the condition of its existence and its overall description.
* The benefit of adding this class is to add subsequent features of to the overall order of the game without actually affecting the main functionality

1. **PretendTreasure Class**

* This class is to be implemented where there it serves the functionality of a false treasure that is present in some stages in the adventure.
* Since, the procedure in this class is followed from the functionality of the *AdventureLevel* class, this class can utilize the object-oriented property of inheritance and thus, this class acts as a sub-class of the *AdventureLevel* class.
* The purpose of this class’ creation is that the overall order/flow of the course of the adventure changes to some extent when a false treasure is chosen by the user.
* Along with inheriting the instance variables of the parent class, this class has its own instance variable for the false treasure.

1. **FinalMission Class**

* This class utilizes the principle of encapsulation with respect to the variables and the methods implemented for the final stage/mission of the adventure.
* The last mission is quite complex in structure and hiding away this complexity from the end user by making a separate class makes it easier to extend or alter it as convenient without affecting the flow of the overall game**.**

1. **Villager Class**

* This class provides a descriptive implementation of the “Villager” entity.
* It has its own instance variable. The main benefit of instantiating this variable is to avoid the tedious process of coding of special words in an array of the string data type.

1. **Monster Class**

* This class was made to extract away the subtleties of the "Monster" entity and the tasks that can be performed on it, for example, checking whether it is alive or not, and the action of killing it.
* It's another genuine element we are managing, and has significant related capacities, which is the reason it ought to be a class without anyone else. It abstracts all the part capacities and factors related with monsters.

1. **Animal Class**

* Like the previous two classes, “Animal” is also an entity on its own.
* Here, the required characteristics of the concerned animal and its relevant sounds can be encapsulated.
* This can be used for the snoozing functionality.

1. **Player Class**

* This class includes the minor functionality of providing the getter method for generating the name of the user/player.

1. **Helper Class**

* This class provisions adding any custom utility functions that are needed to be accessible to the entire program.
* **This way, the functions are not required to be duplicated in every class that requires them.**