Node Game

Proposal

Andy Harris, CSCI 121

**Purpose**

This project is intended to demonstrate mastery of concepts learned in CSCI 121, specifically Object-Oriented programming, use of abstract data types, and use of data structures.

**Overview**

NodeGame is a system for building text-based games. It acknowledges that adventure games can be seen as essentially a type of tree data structure. Each decision the player makes, or each combat that the player wins or loses (as well as random chance sometimes) takes the player from the current state to a new state. This system is meant to replicate an adventure game as a sort of binary tree, made of nodes.

**Node Classes**

The basic node will be an interface, describing elements common to all nodes. There will then be three different implementations (for now) of this interface, describing three types of encounters the user can have. Each node will have two child nodes, indicating the two possible results for that encounter. The nodes will also have next and previous properties, so they can be stored in a linked list for easy retrieval and editing. The GameNode interface will indicate a process() method, which will always return an instance of GameNode representing the results of the decision, combat, or random roll. How the process() method is resolved is up to the inherited classes.

The Decision node will have the user make a decision, and will choose a new node based on the decision made.

The RandomNode will have a (hidden) odds value, and will send the user to one choice upon a successful ‘roll’ and the other choice on an unsuccessful roll.

The FightNode will feature the player and a single enemy in combat. Combat will be very simple, but turn-based, and if the player wins the combat, progress goes to the first node. Upon a loss, the combat goes to the other node. This node will require two instances of the character class.

The node details and inheritance relationship can be seen in the attached nodeGame.svg UML diagram.

**Other classes**

The game will also feature two classes not based on the GameNode interface.

The character class will be a very basic roleplaying-style character, with a name, hit percentage, dice number and size, a defensive modifier, an armor value, and a hp value. The character will also have a fight method which will simulate a round of combat with an enemy (also of the character class.) The FightNode will have two instances of the Character class, one indicating the player and one indicating the enemy.

Finally, the Game class will tie all these objects together. It will have a head node, which will contain the first defined game node. This will be the head of the linked list of nodes, and it will also be the first node of gameplay. This head node will serve to store all the nodes as both a linked list and binary tree. The play() method will, of course, play the current game. The initial version of the game will have a makeSample() method, which will build a sample version of the game. If time permits, the Game class will also feature an editor for adding new nodes, editing existing nodes, searching for a node, saving, and loading the game. Object Serialization will be used for node storage and retrieval.

Specifics of these two classes and the aggregation relationships can be viewed in the attached UML diagram.

**Anticipated Use Cases**

The final version will have two use cases:: Game designers will use the editing system to manage creating nodes and connecting them. Game players will be able to load new games and play them.

**Use of Object-Oriented Programming Paradigms**

This project is ideally suited to object-oriented programming. It features 5 classes and an interface. It illustrates all the main features of object-oriented programming:

* **Inheritance** - Three different node classes inherit from the GameNode interface
* **Encapsulation** - All of the data members are hidden and accessible only through methods
* **Polymorphism** - The gameNode has abstract methods for process and edit, which will be implemented in the inherited classes
* **Abstraction** - The GameNode class is an interface, which by definition is abstract
* **Aggregation**  - The Game class holds a linked-list of GameNode-derived instances
* **Composition** - The FightNode class holds two instances of the Character class

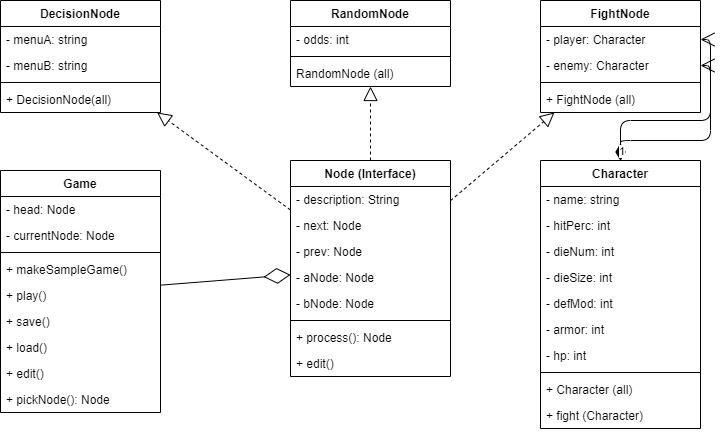
**Use of Data Structures**

The system is unique in that each GameNode class can be seen as either a member of a double-linked list or as a node in a binary tree. For editing purposes, the linked list is more convenient. For gameplay, the binary tree representation is more useful. Regardless of the way the data is viewed, there is only one copy of each node in memory, it can just be accessed in different ways.

**Technology / Implementation**

This system can be written in any object-oriented language. For this project, I will use Java. Some Java-specific features like interfaces and object serialization will be ideal for this project. I do not anticipate any user interface beyond a standard CLI, and the project will run without any external dependencies on any JRE-equipped computer.

**UML Diagram**



**Milestones**

* UML approved
* Node and decision
* Game tree running (sample game)
* Editor
* Save and load
* Random Node
* Edit random node
* Character class
* Combat Node
* Edit combat
* Edit character