Program 4 Design

Alexander DuPree

November 18, 2018

1 Task and Purpose

Program number four tasked us with designing a decision tree to emulate the decisions a player would make in a video game. The exact nature of the video game and the actions a player would make was up to us to implement. For my program I decided to implement a point based game that would reward players based on their actions. The goal of the game is to get 1000 points before reaching an end event. Each path in the tree will present to the player one of four different events to undertake. Successful completion of an event will reward the player with points whereas failure can result in deduction or even failure of the game. Each event can be a decision, chance, riddle or end event. Upon game end, a game summary will be printed out and the high score will be recorded to a file.

2 Design Considerations

Object oriented programming requires a modular design with an emphasis on encapsulation, inheritence, and polymorphism. Accomplishing this design will require multiple classes each with a single responsibility as their focus. This will help encapsulate and protect data related to a specific task while keeping the program organized and scalable. This assignment aim's to apply those design principles and as such, I will be keeping each class as small as possible, and create only as much production code required to accomplish a specific task while avoiding the excessive use of "getters/setters".

3 Classes, Relationships, and Responsibilites

The following is a basic outline of each class to be created for this program and their responsibility and relationship to other classes.

3.1 Event

Responsibilities: The Event abstract base class defines a common interface for each type of event in our system. Each event will have a description, points, and an action method to carry out the event for the player.

Relationships: The Event object acts as the abstract base class for all objects in the events hierarchy. Furthermore, the 2-3-4 tree used in this assignment will use FEvent references and dynamic binding to manage a collection of Events.

Functions: The Event abstract base class will define the compareTo method to use the point value of an event for the natural ordering of any event object. The Event interface will also require all derived objects to define a points and action method for the user.

3.2 Decision

Relationships: The Decision class is derived from the Event base class and expands upon the base class methods and attributes.

Responsibilities: The Decision class maintains the same responsibility outlined in the base class. The Decisions action method will present to a player a scenario and a selection of choices the player can make. The player will be rewarded points based on the decision made.

Functions: The Decision class will define methods for reading in and evaluating user input.

3.3 Chance

Relationships: The Chance class is derived from the Decision class as the structure of the event is held mostly the same for this object as well.

Responsibilities: The Chance class will present to the player a scenario and a selection of choices, just like its super class. However, the chance class will reward points based off a random chance that the players selection was the same as the computers.

Functions: Chance class will utilize the methods defined in its super class as well as define a new method for generating random integers in a specified range.

3.4 End Event

Relationships: End Event is derived from the Event abstract base class and signals the end of the game.

Responsibilities: The End Event present to the player a final scenario and reward the player with a final set of points. After, the players points are tallied and a victory/failure message will be printed to the console, signaling the games end.

3.5 Riddle

Relationships: The Riddle class is derived from the Event abstract base class and presents to the player a unique riddle challenge event.

Responsibilities: When the Riddles action method is called the player will be presented with a randomly selected riddle and be allowed 3 chances to answer correctly. If the riddle is successfully answered the player will be reward points based on the riddles difficulty.

Functions: The Riddle class will define methods for reading/evaulting user input and comparing them against the riddles answer.

3.6 Event Manager

Relationships: The Event Manager is a 2-3-4 Tree that will manage a collection of Event objects. These objects are ordered in the tree by their corresponding point value.

Responsibilities: The Event Managers responsibility, aside from managing the Event object collection, is to find and retrieve specific events for the player to undergo.

Functions: The Event Manager will define methods for search and retrieval of events. The Event Manager will build its tree from an external file upon construction of the object.

3.7 Player History

Relationships: The Player History class is a linear linked list of arrays that will manage a collection of Event references. **Responsibilities:** The Player History responsibility is to record the players choices during an event and keep track of the event itself. The History object will also keep a running tally of the Player's score **Functions:** The Player History class will define methods for adding events and printing out a summary of the Players history during the course of the game.