Seth Denney
COMP 2710 Lab 3
Program Design Outline

I. Use Cases

A. Case – User Begins at A1 and moves East

i. User is begins at Node A1. User is prompted to move North or East. User enters 'E' for East. User is now at Node B1 and is prompted appropriately for new directional options.

B. Case – User Reaches a node with a Chute or Ladder

i. User is in maze and chooses a next direction. The resultant node has a chute/ladder link, and automatically transports the user to the appropriate distant node. The user is then prompted with new directional options.

C. Case – User Reaches Finish Node

i. User moves to Finish node, winning the game. The game prints a congratulations banner and reports information about the gameplay, including nodes visited and number of steps taken.

II. Classes

A. **Maze** – The Maze class holds all game-level functionality and information. It uses the Node class to represent individual rooms in the maze.

i. Methods

a) constructor Maze(string fileIn)
- instantiate a Maze object
- creates Maze from fileIn

b) void createNode(string data)
- takes in a line of data from input file and creates a node
- also creates any appropriate links

c) void playGame()
- runs the sequence for prompting for and receiving user input

d) void move(int direction)
- makes a move from one room to another

e) void victory()
- prints congratulations banner and move history

ii. Data Fields

a) Node *currentRoom
- pointer to currently occupied room

b) Node *mazeArray[]
- stores the maze while it is being built

c) vector<string> history
- stores the ordered list of rooms that have been visited by the user

B. Node – The Node class represents a single room in the maze. It contains the name of the room as well as pointers to available connected rooms.

i. Methods

a) constructor Node(string newName)
- creates a new Node with name = newName

b) constructor Node()
- parameterless constructor; creates a new Node

c) void setNodeName(string newName)
- setter for name

d) string getNodeName()
- getter for name

e) void attachNewNode(Node *newNode, int direction)
- attaches a specified node with the appropriate directional data

f) Node *getAttachedNode(int direction)
- returns next Node

g) void attachLadderChuteNode(Node *newNode, int ladderOrChute)
- creates a chute or ladder to newNode

h) Node *getLadderChuteNode()
- returns the chute or ladder node if it exists

ii. Data Fields

a) string name
- the room name

b) Node *attachedNodes[4]
- list of attached rooms

c) Node *ladderOrChuteNode

- ladder or chute node if it exists

d) int ladderOrChute
   - 1 for ladder; 0 for chute

III. Test Cases

A. Invalid Direction

i. **Scenario –** User enters an option that is either a) incorrectly typed data (integer instead of letter) or b) correctly typed, but not within the domain of acceptable options ("A" where only "N", "E", "S", "W" would be allowed, or "N" where only "E", "W" would be allowed)

ii. **Result –** The System should print an error message and prompt for a correct entry.

B. Input File Is Empty or Not Found

i. **Scenario –** The file containing maze data cannot be read for some reason.

ii. **Result –** The System should let the user know that no maze has been created due to a lack of data, and exit

C. User Visits a Room with a Chute/Ladder

i. **Scenario –** User visits a room that has an attached chute or ladder.

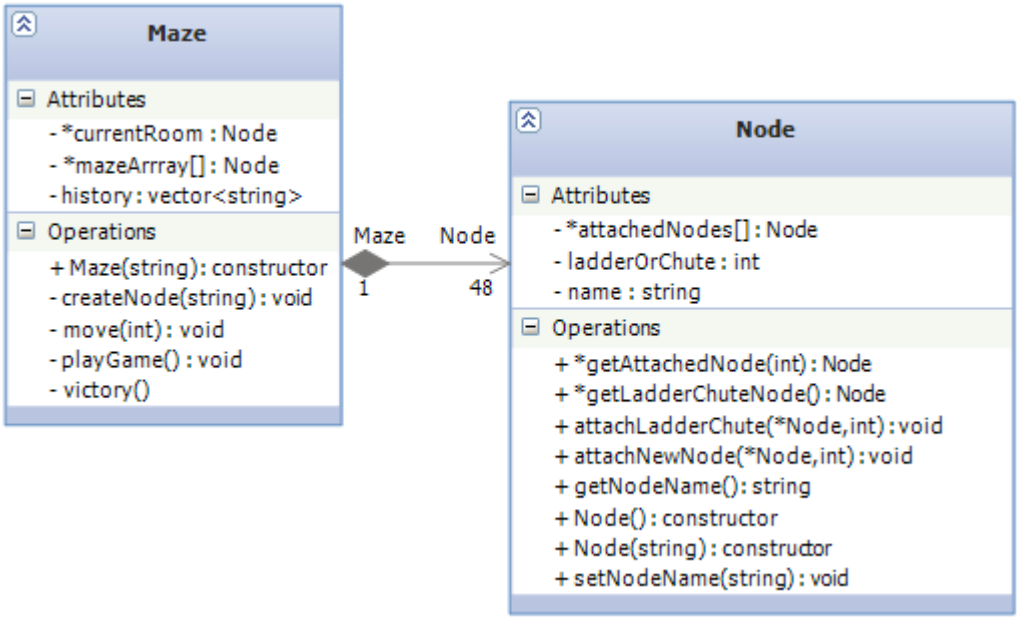
ii. **Result –** The System should automatically transport the user to the other end of the chute or ladder, and then proceed to allow normal gameplay from the new room.

D. User Visits Finish Node

i. **Scenario –** User finds the Finish node, winning the game.

ii. **Result –** The System should not allow the user to continue moving around rooms. Instead, the System should execute the victory instructions appropriately.

IV. Class Diagram



V. Data Flow Diagram