Creating My Own Adventure Game

The objective of this project is to use my knowledge of data structures to create my own text based adventure game. The Create Your Own Adventure (cyoa) utility reads in lines from the given file. Once the entire file is finished, the utility starts a play session to play the adventure described in the file. Play starts in the first room described in the file. Players type one letter to make a choice or enter a command. Play continues until an error is encountered or the player enters the quit command ('q'). Other commands can be found here:

COMMANDS

a - l Standard choices defined by the adventure.

r Restart the adventure in the first room. Loses any existing saved undo state.

q Quit the game.

y Show information about the adventure. Prints one room per line, including tag of the room, then tags for destinations of all possible options from that room.

z Undo the previous choice, go back to previous room (can be done multiple times).

In order to clearly describe data types, I used in this program I must first explain the format the file which contains the contents of the adventure. Any blank lines are ignored. Any non-blank lines must start with a single character command, then a space, then any number of content characters followed by a new line. Content characters can be any non-new line characters. Here are the possible character commands:

The commands allowed are:

```
r     Add a new blank room with a tag given by the contents. The room has no
      options when created.
d     Add a line of description to the most recently added room. If no room has
      been added then this command generates an error.
o     Add a new option to the most recent room. The content is the text of the
      option that will be displayed to the player.
t     Update the destination tag of the most recently added option to the contents.
      The tag must match the tag of a room that appears somewhere in the input file
      (that room may occur later in the file than this command).
```

Here's an example adventure file based on The Walking Dead TV series:

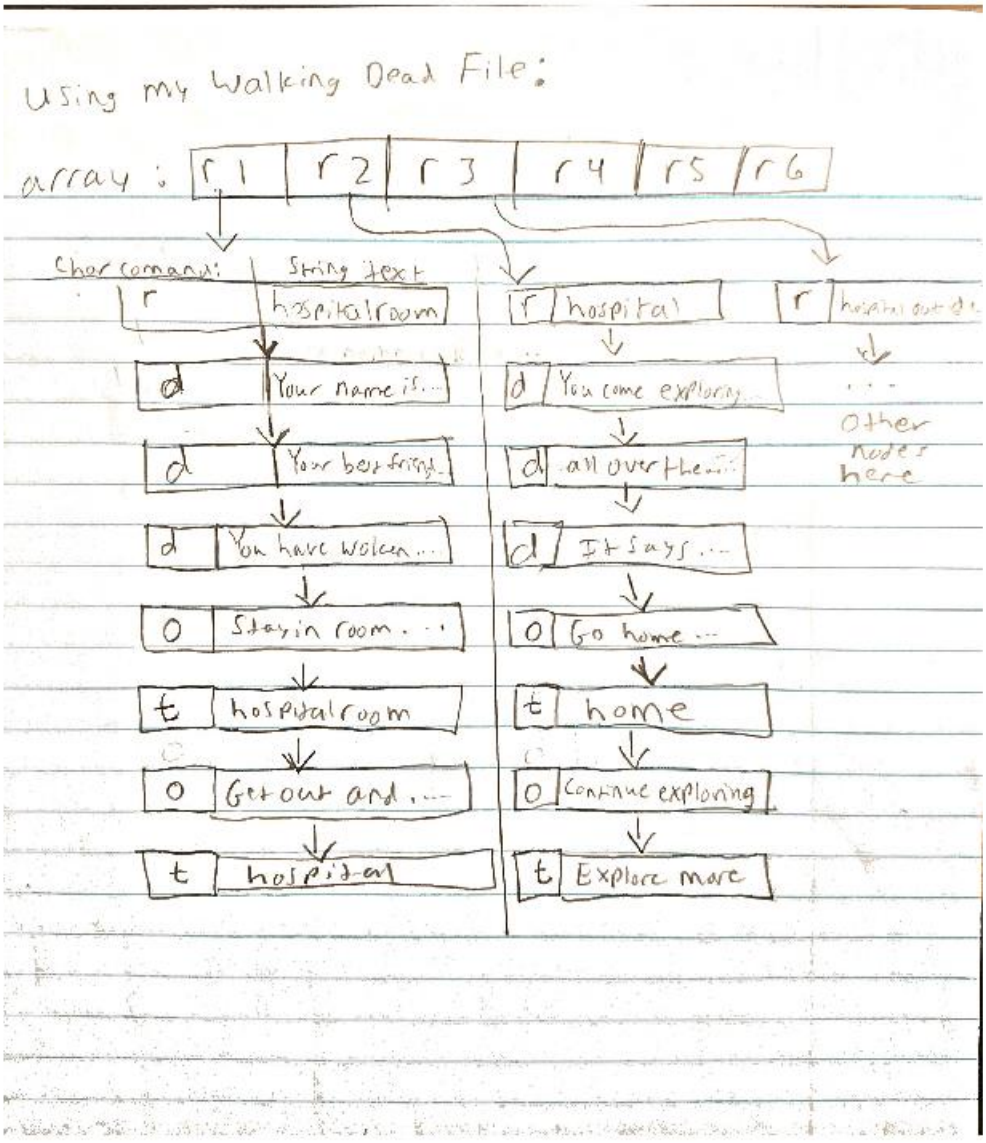
```
r hospitalroom
d Your name is Rick Grimes. You are deputy Sheriff in King County, Georgia.
d Your best friend is Shane. He took you to the hospital after you got shot while on duty.
d You have woken from a coma in a hospital bed. No one is around. You are in hospital clothes.
o Stay in the room until you are rested
t hospitalroom
o Get out and try to tell a doctor you are awake
t hospital

r hospital
d You come exploring the hospital. The hospital is nearly destroyed. Blood and bullets are
d all over the walls. One door has words written in blood.
d It says, "Don't open Dead inside."
o Go home and find your family
t home
o Explore more
t hospitaloutside

r hospitaloutside
d You see hundreds of bodies outside. Half the hospital building is destroyed.
d You see the extent of the damage better from outside. No one is around.
o Go home and find your family
t home
```

The Data Structures

In order to keep each room, separate I wanted to store each room in an array. But the size of the array would have to be different depending on the number of lines in each room. So my best option was to have an array of linked lists, where each index in the array would be "pointing" to the head of each room's linked list (the ' r ' tag). Each element in the linked linked list contains a command char a string text and a node next to point to the next node. Here's a diagram of what I'm trying to get at using my Walking Dead file:



I got the size of my array (number of rooms) by counting the number of blank lines I create 2 arrays of nodes. Their size is the number of rooms. I have a second array to keep track of the rooms in alphabetical order. The second array is called 'orderedrooms' while the first is called 'rooms.' Here's the code:

```
while(s1.hasNextLine()){  //this while loop counts the number of rooms
    temp = s1.nextLine();
    if(temp.length() ==0){ //if there is a blank line....
        numrooms++;        //add to the number of rooms
      }
   }
node[] rooms = new node[numrooms]; //created the rooms array
node[] orderedrooms = new node[numrooms]; //created the ordered rooms array
```

In the program, when the user enters the command 'y' I am supposed to print out all of the rooms followed by their targets. The rooms need to be in alphabetical order. Whenever I ordered my 'rooms' array it messed up the order of the rooms. My history linked list keeps track of the users' history by adding the index of the room array to it every time a choice is made by the user. Once 'rooms' was sorted, every index in the history list would not match up. Since my primary concern is performance, not efficiency, I went ahead and just created a second array which is a complete duplicate of 'rooms.' Once I sort it (using merge sort) the two arrays are different. I could have changed my history array to keep track of specific string 'text' for the room name rather than the index in the array, however I just did not want deal with changing 'history' as it took a long time to get working in the first place.

History

Part of the project involved the need for an undo button by pressing 'z.' This undo button needs to allow the user to undo all of their operations and allow them to go back to the beginning if they wanted to. In order to do this, I created a second type of linked list called 'historynode.' The history node contains an 'int id' and an 'int num.' Id keeps track of the index the node is in the linked list while 'num' keeps track of the room index that is being saved. This linked list adds the current room's index to the linked list. The current room is just the index in 'rooms' where the user is currently looking at on screen. Since I did not want to allow a forward functionality every time the user goes back, the last node in 'history' is removed as the user goes back to the second-to-last node's room index. Here's some code of my addToHistory method:

```
    historyNode temp = history; //create a temp history node
   historyNode newNode = new historyNode(); //create a brand new history node
  newNode.id = historyid; // index of the new node in the history list
//debugging: System.out.println("historyid = "+ historyid);
  newNode.num = num; //
for(int i = historyid-1; i>0; i-- ){ //gets to end of history linked list
  temp= temp.next;
}
  temp.next = newNode;
 // debugging: System.out.print(newNode.id+ "- - - "+ newNode.num);
 historyid++; //one more node has been added to the list
```

Undo

Once my History was inputting everything in correctly I created a separate printHistory function to display the history linked list after every input. Keep in mind, Index 0 is created when the program initially starts. It stores room 0 (the first room). Here is my printHistory output:

```
Your name is Rick Grimes. You are deputy Sheriff in King County, Georgia.
Your best friend is Shane. He took you to the hospital after you got shot while on duty.
You have woken from a coma in a hospital bed. No one is around. You are in hospital clothes.

a  -  Stay in the room until you are rested
b  -  Get out and try to tell a doctor you are awake

b
[Get out and try to tell a doctor you are awake]

You come exploring the hospital. The hospital is nearly destroyed. Blood and bullets are
all over the walls. One door has words written in blood.
It says, "Don't open Dead inside."

a  -  Go home and find your family
b  -  Explore more

There are 2 elements in the history
INDEX: 0 ROOM stored: 0
INDEX: 1 ROOM stored: 1
a
[Go home and find your family]

You arrive at your home. No one is here. The front door is wide open. Your wife, Lori, and son, Carl,
are both absent. The rooms are a messs. Pictures and clothes are missing. You go to sit
on the porch outside. You see a person who doesen't seem fully awake comming towards you.
A child comes to you and hits you on the head with a shovel. His father kills the person comming
towards you. You wake up in their home.

a  -  Ask what's going on

There are 3 elements in the history
INDEX: 0 ROOM stored: 0
INDEX: 1 ROOM stored: 1
INDEX: 2 ROOM stored: 3
```

If you think of this as a normal array, INDEX is just showing the number “i” value when you iterate through an array. The room number is the number of all the rooms the user is/was at. Next I had to test the undo function. What I needed to do was get to the second to last node, make the room number stored in this node the current one, and delete the last node in the list. I could have used a queue for this with a normal array however technically the user can keep on choosing an option that keeps them from ending the game. When they do this, the history array would fill up and error out. This is why I chose to stick with a linked list for history, as I can have as many elements/nodes that I need at any one point in the game. Here’s what my undo code looked like:

```
else if(input.equals("z")){ //if user types in "z"
    //historyid = number of elements in list
    if(historyid>=0 &&history.next!=null){ //if you can go back...
        historyNode temp2 = history; //copy the history list
        while(temp2.next.next!=null){ //increment to the second to last node
            temp2 = temp2.next;
        }
        currentroom = temp2.num; //get assign the room in history to be the current room
        temp2.next = null; //delete the last node (Garbage Collector For The Win!!)
       //Debugging: System.out.println("CURRENT ROOM IS "+ currentroom);
        System.out.println("[undo]");
        printDescription(rooms[currentroom]);//then print this room's description and options
        historyid--; //decrease the number of elements in the history list
     }
     else{
         System.out.println("Invalid input");
     }
```

(Continue to next page)

And here is the history list when I use the z command:

```
You come exploring the hospital. The hospital is nearly destroyed. Blood and bullets are
all over the walls. One door has words written in blood.
It says, "Don't open Dead inside."

a  -  Go home and find your family
b  -  Explore more

There are 2 elements in the history
INDEX: 0 ROOM stored: 0
INDEX: 1 ROOM stored: 1
a
[Go home and find your family]

You arrive at your home. No one is here. The front door is wide open. Your wife, Lori, and son, Carl,
are both absent. The rooms are a messs. Pictures and clothes are missing. You go to sit
on the porch outside. You see a person who doesen't seem fully awake comming towards you.
A child comes to you and hits you on the head with a shovel. His father kills the person comming
towards you. You wake up in their home.

a  -  Ask what's going on

There are 3 elements in the history
INDEX: 0 ROOM stored: 0
INDEX: 1 ROOM stored: 1
INDEX: 2 ROOM stored: 3
z
[undo]
You come exploring the hospital. The hospital is nearly destroyed. Blood and bullets are
all over the walls. One door has words written in blood.
It says, "Don't open Dead inside."

a  -  Go home and find your family
b  -  Explore more
There are 2 elements in the history
INDEX: 0 ROOM stored: 0
INDEX: 1 ROOM stored: 1
```

Conclusion

The history linked list, rooms array and rooms linked list were all the new data structures I created, designed, and developed. Through this project I was able to apply my knowledge of abstract data types to a real application: an actual game. I enjoyed this project because I allowed me to test my skills in developing my own solution to this text based adventure game.