Online Text Adventure

Week 13 Program

Week13Program: Online Text Adventure

- **This is a group project. You must work with your lab group.**
- **All students are expected to participate in the project.**

- **Read "Creating a Text Adventure in Java" (http://www.javacoffeebreak.com/text-adventure/). This will give you an idea of how to develop a basic text adventure with Rooms, a Player, and Items (these will become classes in your program).**

- **REMINDER: This assignment is to be completed without help from students other than your lab partners. Help from your instructor, TA and learning center coaches is acceptable and encouraged. Failure to follow this policy is considered cheating and you will be severely punished when caught.**

Part 1: Design Documentation

1. (20 points) Describe the scenario: the first step in creating an Adventure program is to come up with a scenario, or setting, for the Adventure. The scenario should give a background to the Adventure as well as provide the Adventurer with a main goal to accomplish.

For example:

The Adventurer is on an intergalactic spaceship when it is attacked by hostile enemies. He takes a small shuttlecraft and flees from the ship just before it is destroyed by the enemy. The shuttlecraft, however, runs out of fuel before it can get him to the nearby space station. It crash lands on a nearby planet and is almost totally destroyed by the crash. The Adventurer must now deal with possibly hostile aliens and the many other dangers of the alien environment and try to get off the planet and back to the space station.

Design Documentation

2. (20 points) Describe puzzles/obstacles: The second step in creating an Adventure is the formation of puzzles and obstacles which the Adventurer must solve and overcome in order to achieve his main goal.

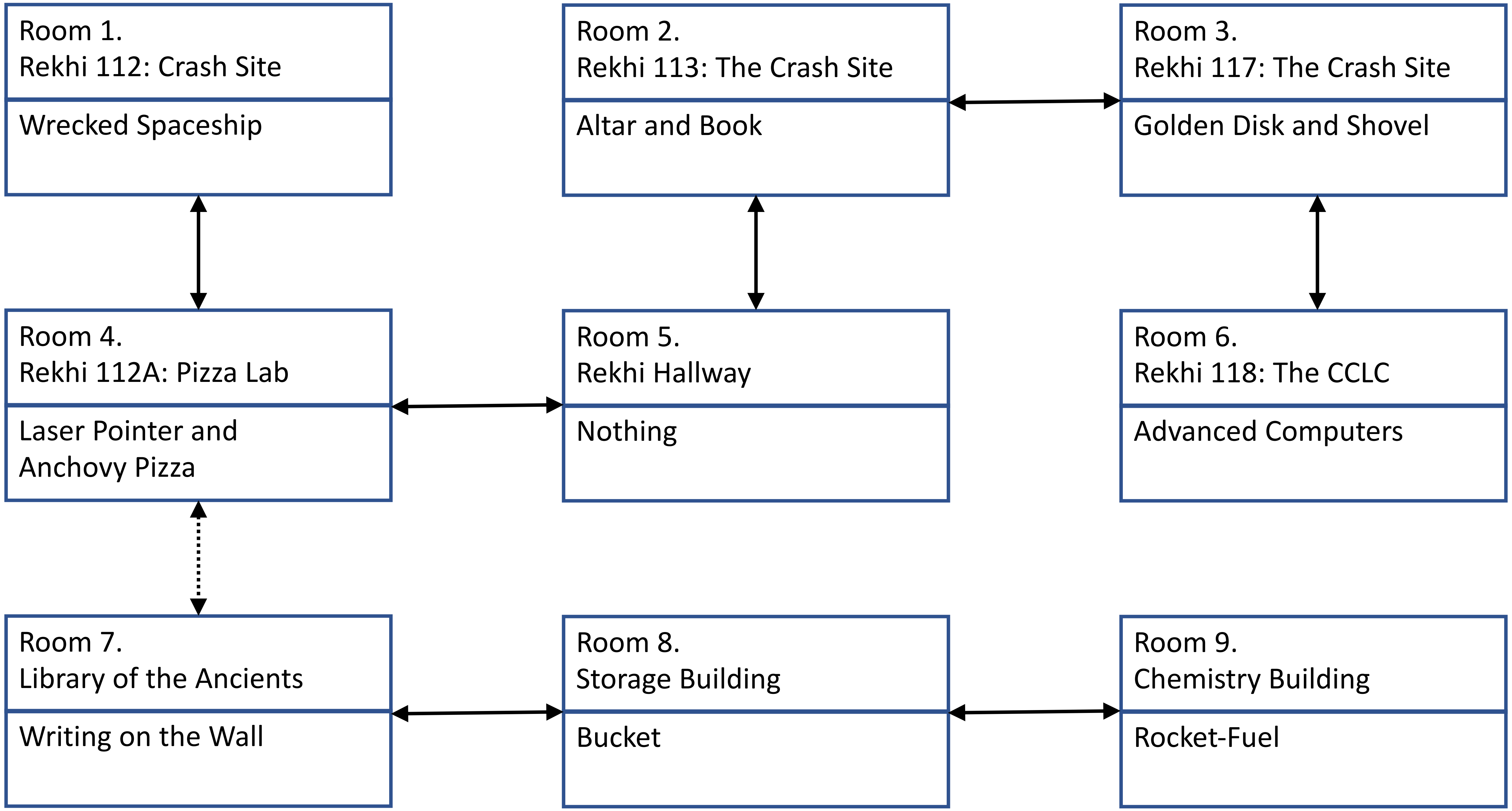
For Example:

- *Aliens surround the Adventurer and gesture menacingly at him. He must get past them somehow without arousing their anger.*
- *The Adventurer must obtain a golden disk in order to enter the walled city of the ancients.*
- *Once inside the city of the ancients, he must decipher the old scrolls of knowledge which tell where things that he needs are hidden.*

Design Documentation

3. (20 points) Mapping it out: an Adventure is divided into many locations called rooms. A room can be anything from a closet to a forest. It may have one or more objects in it and may have exits in any of four directions — north, south, east, or west. What you have to do now is to create and map out rooms in your Adventure. Draw your map as interconnected boxes. Each box will represent one room in the Adventure. Number each box and give it a name. In the bottom of the box, list the items in the room.

   NOTE: You must have at least nine rooms and at least 6 items in your game.


Room 1.
Rekhi 112: Crash Site
Wrecked Spaceship
Room 2.
Rekhi 113: The Crash Site
Altar and Book
Room 3.
Rekhi 117: The Crash Site
Golden Disk and Shovel
Room 4.
Rekhi 112A: Pizza Lab
Laser Pointer and Anchovy Pizza
Room 5.
Rekhi Hallway
Nothing
Room 6.
Rekhi 118: The CCLC
Advanced Computers
Room 7.
Library of the Ancients
Writing on the Wall
Room 8.
Storage Building
Bucket
Room 9.
Chemistry Building
Rocket-Fuel

Design Documentation

4. (20 points) Describe rooms: When the player enters a room, the computer describes the room, lists the objects in the room that can be manipulated, and lists the obvious exits. Repeat this description for each room in your map.

For example:

1. *You are in Rekhi 112: The Crash Site. A warm feeling welcomes you as you enter. You sense the blood and tears of many generations. A tangled mass of computers lay crushed against the podium.*
   You see: Wrecked Spaceship.
   Obvious exits lead: South.

Design Documentation

5. (20 points) Demo Paths: Provide the grader with two demo paths. A demo path is a sequence of commands used to chart a path through the game. The first demo path should lead to the player's demise. The second demo path should lead to victory.

For Example:

Fatal Demo Path: *GO SOUTH, GET PIZZA, EAT PIZZA*

Victory Demo Path: *GO SOUTH, GO EAST, GO NORTH, GO EAST, GET DISK, GO SOUTH, RUN PROGRAM, GO NORTH, GO WEST, PUSH ALTAR, GO SOUTH, GO WEST, GO SOUTH, GO EAST, GET BUCKET, GO WEST, GET ROCKET-FUEL, GO WEST, GO WEST, GO NORTH, GO NORTH, REFUEL SPACESHIP*

Part 2: Provided Code

1. **AdventureServer**: Server code that maintains client internet connections and coordinates all input and output. The AdventureServer implements the **AdventureServerApi**:

```
// Starts the server running
public void startServer ( int port );

// Stop the server
public void stopServer ( );

// Set the transmission listener
public void setOnTransmission ( ConnectionListener listener );

// Send text through the connection.
public void sendMessage( long connectionId, String message ) throws UnknownConnection;

// return true if a connection exists
public boolean isConnected( long connectionId );

// Close the connection
public void disconnect( long connectionId ) throws IOException, UnknownConnection;

// Change the connection ID - to facilitate save/load
public void changeConnectionId( long connectionId, long newConnectionId ) throws UnknownConnection;

// return the port
public int getPort( );

// return the server IP Address
public String getInetAddress( ) throws UnknownHostException;
```

Provided Code

2. ConnectionEvent: An event that is issued when something happens related to a connection. For example, an event is triggered whenever a command is sent from the client to the server.

```
public class ConnectionEvent {
 private ConnectionEventCode code = null;
 private long connectionID = 0L;
 private String data = null;
 // Constructor
 public ConnectionEvent ( ConnectionEventCode code, long connectionID, String data ) {
  this.code = code;
  this.connectionID = connectionID;
  this.data = data;
 }
 // The reason for the event
 public ConnectionEventCode getCode ( ) {
  return code;
 }
 // The ID of the connection associated with the event.
 public long getConnectionID ( ) {
  return connectionID;
 }
 // A text message associated with the event.
 public String getData ( ) {
  return data;
 }
}
```

Provided Code

3. ConnectionEventCode: an enumerated data type that names the reasons for a ConnectionEvent.

```
public enum ConnectionEventCode {
  CONNECTION_ESTABLISHED,
  TRANSMISSION_RECEIVED,
  CONNECTION_TERMINATED
}
```

Provided Code

4. ConnectionListener: a listener interface that you must implement in order to handle incoming connection events from the server.

```
public interface ConnectionListener {
  public void handle ( ConnectionEvent e );
}
```

Provided Code

5. UnknownConnectionException: an exeption generated when the connection associated with a given connectionId cannot be found.

```
public class UnknownConnectionException extends Exception {
  private long connectionId = 0L;

  // Constructor
  public UnknownConnectionException( long connectionId, String message ) {
    super( message );
    this.connectionId = connectionId;
  }

  // Identifies the unknown connection that was requested
  public long getConnectionId ( ) {
    return connectionId;
  }
}
```

Provided Code

6. AdventureClient: a program that can be used to connect to the AdventureServer if you do not have a telnet client.

   Command line arguments are the server_ip_address and port.

Provided Code

7. TechAdventureServerDemo: an example showing how to use the AdventureServer class.

Part 3: Write the Program

1. (25 points) Main Class: Your main class should be called TechAdventure.
2. (25 points) Main Method:
   a. Your main method should take the port number as the first command line argument. If no port is specified use port 2112.
   b. Your main method should also create an instance of the AdventureServer class. You will also want to call setConnectionListener to register the ConnectionListener that will handle connection events. And call startServer with the port to begin accepting client connections.

```
adventureServer = new AdventureServer ( );
adventureServer.setOnTransmission ( e -> System.out.println(e.getData()) );
adventureServer.startServer ( port );
```

Write the Program

3. Your game should minimally build classes for Player and Room.
4. (25 points)The Room class should minimally contain:
   a. A name
   b. A description
   c. Rooms should be linked together using a linked data structure. For example, the Room class might contain fields such as, Room north; Room up; Room east; etc.
   d. A list of items that can be found and manipulated in the room
5. (25 points) The Player class should minimally contains:
   a. A connectionId
   b. A reference to the Room the player is currently occupying.
   c. A list of items that the player is carrying

Write the Program

6. (25 points) When the AdventureServer triggers a CONNECTION_ESTABLISHED event, describe the players current location, its exits, and listing visible items. This can be done by sending text to the AdventureServer sendMessage command.
7. (25 points) When the AdventureServer triggers a CONNECTION_TERMINATED event, this means the player's client is no longer connected. This event should be handled appropriately to clean-up any objects that are no longer needed.

Write the Program

8. (300 points) When the the AdventureServer triggers a TRANSMISSION_RECEIVED event it means the user has entered a command. The text of the command can be retrieved from the event by calling getData().
   a. Minimally implement the following common commands are GET item, DROP item, GO direction, LOOK, INVENTORY, SAVE, RESTORE, QUIT
      - Players GET items to carry them around or DROP items to leave them in a room.
      - Special commands may be used to manipulate items the player is carrying, such as EAT PIZZA. Sometimes these special commands will cause side-effects within the game, such as revealing a hidden passage.
      - Player may follow an exit (usually by going a direction, e.g. "GO NORTH"), which will take them to a new room.
      - LOOK will redisplay the room description. LOOK item will describe an item.
      - INVENTORY lists all the items the player is carrying.
      - SAVE and RESTORE are used to save and load the state of the game for a player.
      - QUIT terminates the player’s connection.

Write the Program

9. (25 points) There should be at least one win condition, such as obtaining some item or arriving at some location.
10. (25 points) There should be at least one lose conditions, such as falling into a bottomless pit or getting eaten by a grue in the dark.

Part 4: Extra Credit

- Add stationary monsters. (100 points) For example: a Grue that hides in a dark room and eats the player who doesn’t have a flashlight.
- Make the game multiplayer. (200 points) Keep track of which players are in a room. Allow players to talk to each other or say things that are heard by everyone in the room. Hint: maintain a HashMap of players by connectionId.
- Add roaming monsters that move from room to room in the game. (200 points)