CSCI 1301: Introduction to Computing and Programming

Project 4: Interactive Fiction

Introduction

Once upon a time, before the dawn of the Internet as we know it, there lived a forgotten type of computer game called interactive fiction. First written in the late '70s, these games were sort of a choose-your-own adventure book, but for geeks. These games contain no graphics, just a 2nd person (everybody remembers English class, right?) description of your character, and a box for entering commands. With the birth of the kindle and other Internet enabled e-readers, these games are experiencing something of a rebirth, as their screens are a perfect match for these games.

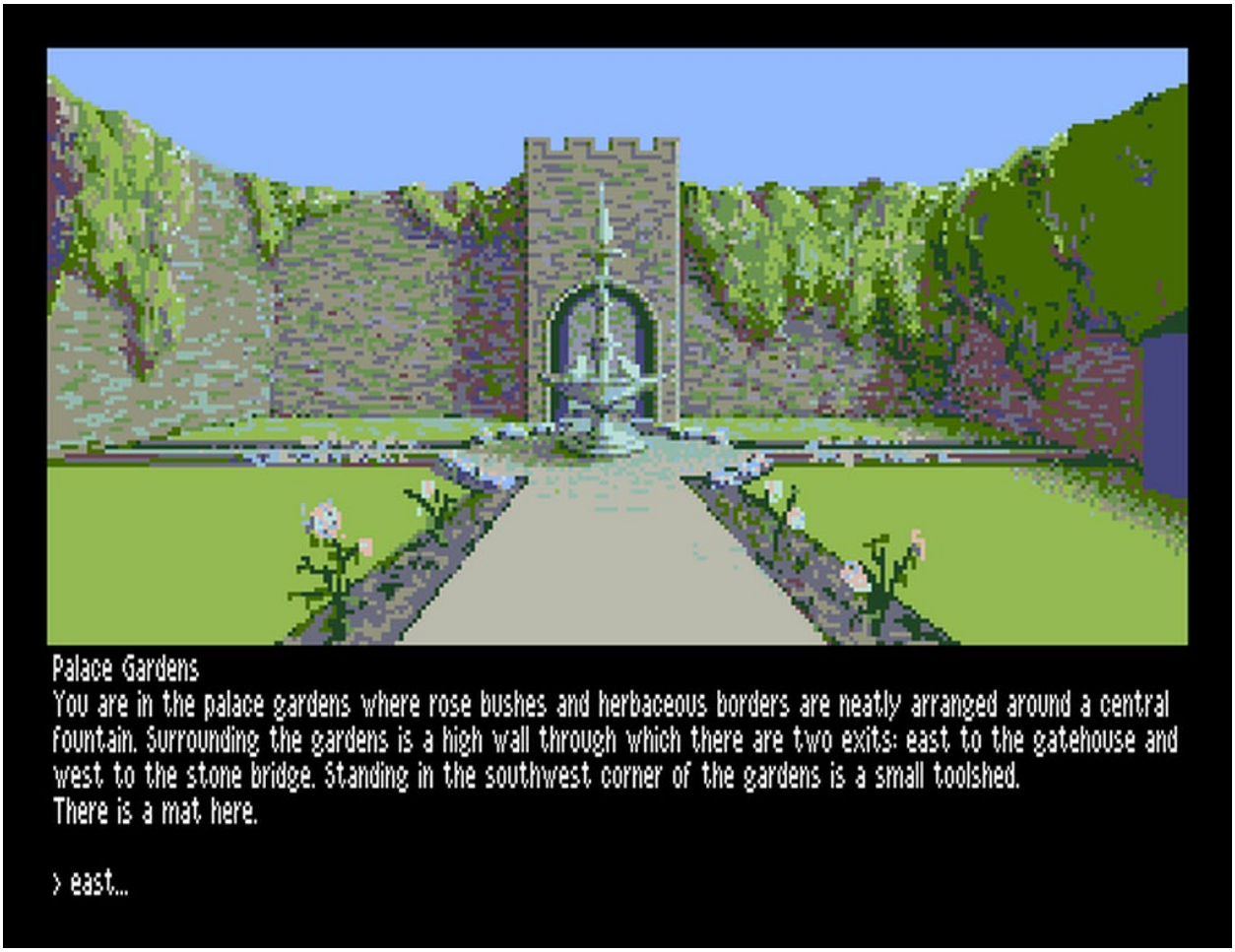

Photo from: http://www.indieretronews.com/2018/01/the-pawn-classic-text-adventure-by.html

To play one for yourself, go to https://eblong.com/zarf/zweb/dreamhold/

Your project is to create an interactive fiction. In this game, you will navigate a map (moving east, west, north, south when possible). In each "room" of the map, the game will print a description of the room to the screen. As you navigate through the map, you will be looking for a light to light up the dark rooms, a key to open a treasure chest, and a chest (that contains the

treasure!). Once you have found the key, you can open the chest. While going through the map, you must watch out for the deadly grue! To help you get started on the game, a couple of classes have been written for you. You need to make the guts of the game, called the engine. You will find Map.java, Room.java, Lamp.java, Key.java, and Chest.java attached to this assignment. Some of these classes are completely finished and others are just skeletons (containing comments and method signatures).

Begin by taking a quick a look at the Map class, which is completely implemented for you (**do not modify the Map class, just know how to use it**). In this game, a map is a square of size NxN where each cell contains a room. Each room contains a description of the room and possibly a few other items. You call the getRoom(X,Y) to retrieve the Room object for a given square. The first number (X) gives the row and the second (Y) gives the column. The starting room is at (0,0).

To the right is a picture of a general map. Note that the map can be of any size. To clarify the X and Y positions, we have marked the map at position (2,1) with an O. In this cell, we would say that the X value is 2 and the Y value is 1.

	0	1	2	…	n
0	X				
1					
2		O			
…					
n					

While debugging your code, you can run the program in "simple map" mode where the map contains only a single room (see below). This is useful in debugging because no one wants to walk through a huge map only to find that the final command doesn't work. When you have tested your code and want to play the entire game, set the simpleMap variable to false for a larger map. The larger map we have provided is of size 4x4.

Illustration of Map when simpleMap = true

	0
0	X

Now take a look at the Room class. This class is implemented for you as well. It contains a large number of methods for you to use during the game. As you can see, there are no setters, only getters. This means that for the most part you cannot modify any Room objects you receive. The exceptions are the clearKey() and clearLamp() methods that modify two of the room's sub objects. Note: these methods allow you to modify a Room object. **You should never modify the Room.java code**. You'll see down below when you need to call these methods. If a room does not have a specific sub-object (key, lamp), its variable will be set to null. You will need to test for this condition to see if a room has a sub-object or not. *The description of the interface for this class is given at the end of this pdf.*

Now, take a look at the Key class, it is written for you. **You do NOT need to update anything in the Key.java file.** The Key class only has one method: public void use(Chest onChest). Calling this method simply calls the unlock method on the given chest. *The description of the*

interface for this class is given at the end of this pdf.

Program Requirements – General Algorithm

You should **NOT modify *Room.java* or *Map.java or Key.java*** to get your game working. Your project will be tested with our own versions of these two files, so any modifications you make could result in your project not compiling or not working correctly.
Also – **you should never create new Key or new Chest objects. This is done already in the preexisting code. So, if you add the words "new Key()" or "new Chest()", you will have logical errors in your code.**

1. You should start your work by implementing the Chest class (a skeleton is provided for you). It needs to store whether it is locked or not, a String describing the contents, and which key it was locked with. The method stubs for this class are provided with comments, be sure to implement them the way they are described. You should not add any more methods to this class.

2. Next, implement the Lamp class (the class definition is provided for you), it only needs to store whether it is lit or not. Follow the rules of encapsulation/information hiding, and prevent any other class from accessing or modifying your instance variables directly.

3. Then, create a player class in a file called **Player.java**. No skeleton is provided for you, you must create it from scratch. You're responsible for adding instance methods and variables. A player object will represent the user playing the game, and thus needs to store which map square they are currently on (integer x and y coordinates), the lamp and key. You should create instance class type variables for the lamp and key, and if they happen to be null, the player does NOT currently have them (same is true for the Room objects as well). Be sure to follow encapsulation rules here too. You do not need to initialize the lamp and key instance variables, as they will default to null. When the player collects the lamp and the key, you will call setLamp and setKey to set these variables.

4. Once these are done, create a class called **Adventure** and write the **main method** of your program in it (remember that the class that contains the main method is the one that you run). The first thing to do in main is to create a Player object and set its starting coordinates to (0,0). Also, you will need to create a new Map object. The idea in the Adventure class is that you loop repeatedly until the player dies (attacked by grue – see below) or finds the treasure (opens chest – see below). Every time the player moves, update the player's position and get the appropriate Room object from the map. Using this object, display the appropriate text to the user based on what command they type in. A list of commands is given below. Ignore case on the user input.

    - GET LAMP – If the lamp is present in the current room (there is a method in the Room class (i.e. getLamp()) that allows you to check this. Note the method's return type and assign the return value accordingly), this transfers the lamp from

the room to the player. Be sure to clear the lamp from the room afterwards. Print "OK" if successful, or "No lamp present" if not. Note: that you can find the lamp in a dark room.

- LIGHT LAMP – If the player has the lamp, this sets it to lit. Print "OK" if successful, or "You don't have the lamp to light" if the player doesn't have the lamp.

- NORTH, SOUTH, EAST, WEST – If the current room (prior to the move) isDark(), AND the player doesn't have the lamp OR they have the lamp but the lamp is not lit, the player is eaten by a grue and the game is over. (see below for an example). Otherwise, move the user North one square (-1 x), South one square (+1 x), East one square (+1 y), or West one square (-1 y). See table below to understand x and y in the context of the map. Once you move into a new room, you should print out its description, so the user doesn't have to type LOOK every time. Be sure to check the current room object to see if the given direction is valid. If not, print ("Can't go that way"). If the room (after the move) isDark() AND the player does not have the lamp OR the lamp is not lit, then instead of printing the description, tell them: "It is pitch black, you can't see anything. You may be eaten by a grue!". (see below for an example)

- LOOK –If the room isDark() and the player does not have the lamp or the lamp is not lit, then instead of printing the description, tell them: "It is pitch black, you can't see anything. You may be eaten by a grue!". (see below for an example). Otherwise, this prints the description of the current room object, as well as any objects that are in the room. You should also print which exits from the room are valid.

    i. If the lamp is present in the room, print "There is an old oil lamp that was made long ago here." after you have printed out the room description.
    ii. If the key is present in the room, print "You see the outline of a key on a dusty shelf that's covered in dust." after you have printed out the room description.
    iii. If the chest is present in the room, print "There is a large, wooden, massive, oaken chest here with the word "CHEST" carved into it" after you have printed out the room description.

- GET KEY – If the key is present in the room, this transfers the key to the user's inventory. Be sure to clear the key from the room afterwards. Print "OK" if successful, "No key present" if not.

- OPEN CHEST – If the chest is present in the room and is unlocked, then this should print out the chest's contents and end the game. If the chest is locked, print "The chest is locked". If the chest is not present in the room, print "No chest present".

- UNLOCK CHEST – If the user has the key, call the use() method with the chest object to unlock it, then print "OK". If the user doesn't have the key, print "Need a key to do any unlocking!". If the chest is not present, print "No chest to unlock".

- (anything else) – Just print "I'm sorry I don't know how to do that.", and re-prompt the user to allow the user to enter another command. The user should be able to continue playing after an invalid command is inputted, and the game should continue normally after an invalid command. The program should not crash or end abruptly if the user inputs an invalid command.

5. When your program starts, print out the following text:
   Welcome to UGA Adventures: Episode I
   The Adventure of the Cave of Redundancy Adventure
   By: (Your name)

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (-3 points) If the source file(s)/class(es) are named incorrectly (case matters!)
- (-3 points) If your source file(s) have a package declaration at the top
- (-3 points) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (-3 points) If you have more than one instance of Scanner in your program. Your program should only have one instance of Scanner.
- (-3 points) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (-30 points) If the source file(s) are not submitted before the specified deadline or if they do not compile. Late submissions will not be accepted for this project since it is the last project of the semester.
- If your (-3 points) comments or (-3 points) variables are "lacking"
  - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (-3 points) Indentation is not consistent throughout your source code
  - Refresh your memory of indentation patterns in chapter 2 in the course textbook
  - Be careful of a combination of tabs and spaces in your files (use one or the other)!

If any of the above do not make sense to you, talk to a TA or ask your lab instructor.

Project Grading

All projects are graded out of a possible 30 points. Programs that do not compile will receive a grade of zero. You must make absolutely certain your program compiles before submitting, and you must thoroughly test your program with many different inputs to verify that it is working correctly. This project will be graded for correctness and adherence to all project instructions and requirements on various test cases, and you are responsible for testing that your program works with all valid inputs. The test cases used for grading may differ than the examples provided in this project. All instructions must be followed in order to receive full credit.

Project Submission

Submit the files **Adventure.java**, **Chest.java**, **Lamp.java** and **Player.java** in eLC. [Note that you are not uploading Map.java, Room.java or Key.java because you should **not** have updated these.]

Example Executions

Your program should work correctly and follow the examples below. Each example is a separate run of a correctly working program.

This example is from simpleMap (simpleMap variable set to true in the Map class)

```
Welcome to UGA Adventures: Episode 1
The adventure of the Cave of Redundancy Adventure
By: Brad and Karen

This is the one room map. Everything you need is here!!
get key
OK
open chest
The chest is locked
unlock chest
OK
open chest
all the gold  (Here, we are printing the contents of the chest)
```

Note: The next 2 examples follow the full map (simpleMap variable set to false)

```
Welcome to UGA Adventures: Episode 1
The adventure of the Cave of Redundancy Adventure
By: Brad and Karen

FOREST TRAIL:
You are standing on a dirt trail that leads to the east. Surrounding you is the famously thick underbrush
of Oconee Forest Park, the fall leaves falling off the trees have just started to poke holes in the canopy.
Off in the distance you hear the cool sounds of Lake Herrick drifting through the trees.
```

look
FOREST TRAIL:
You are standing on a dirt trail that leads to the east. Surrounding you is the famously thick underbrush of Oconee Forest Park, the fall leaves falling off the trees have just started to poke holes in the canopy. Off in the distance you hear the cool sounds of Lake Herrick drifting through the trees.
Exits are: east
east
WOODEN WALKWAY:
The wooden walkway makes a :clump: as you stride along it.
Frogs give a terrified SKRIP! and jump to safety as your adventurous frame comes into view.
A mountain biker, ignoring the many "Fragile Habitat" signs, speeds along a narrow dirt path to your right.
As you reach the end of the walkway, you notice a hole in the ground in a newly fallen tree's footprint, just
large enough for you to fit through.
look
WOODEN WALKWAY:
The wooden walkway makes a :clump: as you stride along it.
Frogs give a terrified SKRIP! and jump to safety as your adventurous frame comes into view.
A mountain biker, ignoring the many "Fragile Habitat" signs, speeds along a narrow dirt path to your right.
As you reach the end of the walkway, you notice a hole in the ground in a newly fallen tree's footprint, just
large enough for you to fit through.
There is an old oil lamp that was made long ago here.
Exits are: east
west
get lamp
OK
east
It is pitch black, you can't see anything. You may be eaten by a grue!
light lamp
OK
look
CAVE ENTRANCE:
The damp earthen crawlspace gives you claustrophobic thoughts as you crawl through its damp earth.
Fortunately, the passageway soon grows large enough for you to stand, though slightly bent over.
The air smells of rotting wood, and a rotting wood smell hangs thick in the air.
Exits are: east
West

[[…The rest of the example removed in the sake of space…]]

In this example, the user is eaten by the grue:

Welcome to UGA Adventures: Episode 1
The adventure of the Cave of Redundancy Adventure
By: Brad and Karen

FOREST TRAIL:
You are standing on a dirt trail that leads to the east. Surrounding you is the famously thick underbrush of Oconee Forest Park, the fall leaves falling off the trees have just started to poke holes in the canopy. Off in the distance you hear the cool sounds of Lake Herrick drifting through the trees.
east
WOODEN WALKWAY:
The wooden walkway makes a :clump: as you stride along it.
Frogs give a terrified SKRIP! and jump to safety as your adventurous frame comes into view.
A mountain biker, ignoring the many "Fragile Habitat" signs, speeds along a narrow dirt path to your right.
As you reach the end of the walkway, you notice a hole in the ground in a newly fallen tree's footprint, just
large enough for you to fit through.
east
It is pitch black, you can't see anything. You may be eaten by a grue!
west
You have stumbled into a passing grue, and have been eaten

Map Class Interface

class Map:
private boolean simpleMap : This is set to true by default and allows you to test your code with a 1-dimension room
public Room getRoom(int x, int y) : Call this method with two int's to get the appropriate room. The valid range for both x and y is 0 to 3.

Room Class Interface

class Room:
public String getDescription() : Returns the text description of this room.
public boolean canGoNorth() : Returns true if the player can go north from this room.
public boolean canGoSouth() : Returns true if the player can go south from this room.
public boolean canGoEast() : Returns true if the player can go east from this room.
public boolean canGoWest() : Returns true if the player can go west from this room.
public Lamp getLamp() : Returns the lamp object in this room. If no lamp is present, returns null.
public void clearLamp() : Sets the lamp variable in this room to null.
public Key getKey() : Returns the key object in this room. If no key is present, returns null.
public void clearKey() : Sets the key variable in this room to null.
public Chest getChest() : Returns the chest object in this room. If no chest is present, returns null.
public boolean isDark() : Returns true if there is no light in this room, veeeeeeery dangerous!

Key Class Interface

class Key:
public void use(Chest onChest): Calls the unlock method on the given chest.
