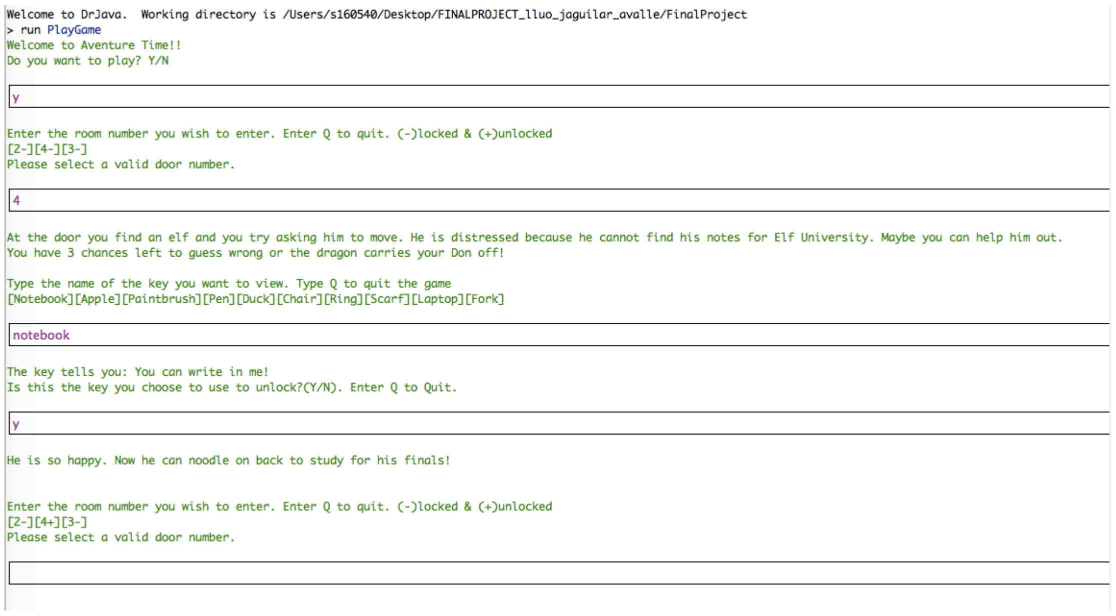
How to run the program:

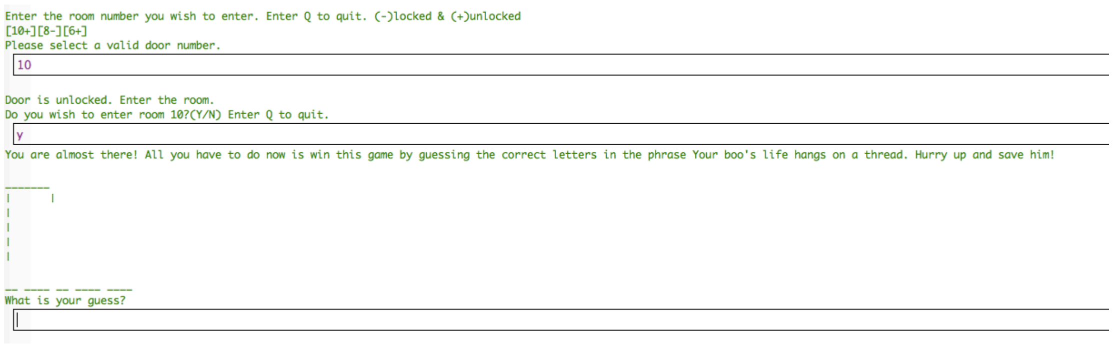
To play using Scanner/user input, run PlayGame.java
To play using GUI, run PlayGameGui.java

Currently we are having issues with the functionality of our GUI, however the game is fully functional through the PlayGame driver. When playing the game using the PlayGame driver, the program prompts the user with text through the interactions panel and allows the user to respond using text input. The numbered square brackets represent the rooms that are connected to the current room the player is int. A plus sign indicates that the door is unlocked and a negative sign indicates that the door is locked. To unlock a locked door, the user is prompted with a set of ten keys, from which they have three chances to choose the correct one to unlock the door (later versions of the game can include more interesting text interactions with the doors and keys).



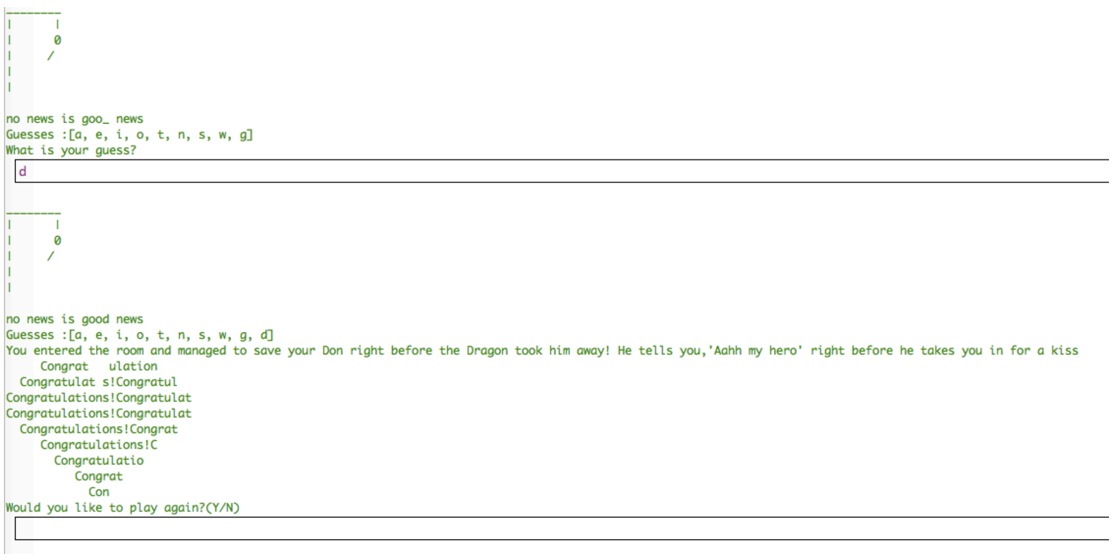
```
Welcome to DrJava.  Working directory is /Users/s160540/Desktop/FINALPROJECT_lluo_jaguilar_avalle/FinalProject
> run PlayGame
Welcome to Aventure Time!!
Do you want to play? Y/N

y

Enter the room number you wish to enter. Enter Q to quit. (-)locked & (+)unlocked
[2-][4-][3-]
Please select a valid door number.

4

At the door you find an elf and you try asking him to move. He is distressed because he cannot find his notes for Elf University. Maybe you can help him out.
You have 3 chances left to guess wrong or the dragon carries your Don off!

Type the name of the key you want to view. Type Q to quit the game
[Notebook][Apple][Paintbrush][Pen][Duck][Chair][Ring][Scarf][Laptop][Fork]

notebook

The key tells you: You can write in me!
Is this the key you choose to use to unlock?(Y/N). Enter Q to Quit.

y

He is so happy. Now he can noodle on back to study for his finals!


Enter the room number you wish to enter. Enter Q to quit. (-)locked & (+)unlocked
[2-][4+][3-]
Please select a valid door number.
```

Once the user successfully makes their way through the map and enters the 10th room, they are prompted to play a game of hangman to save the don in distress who is being held captive by a dragon.



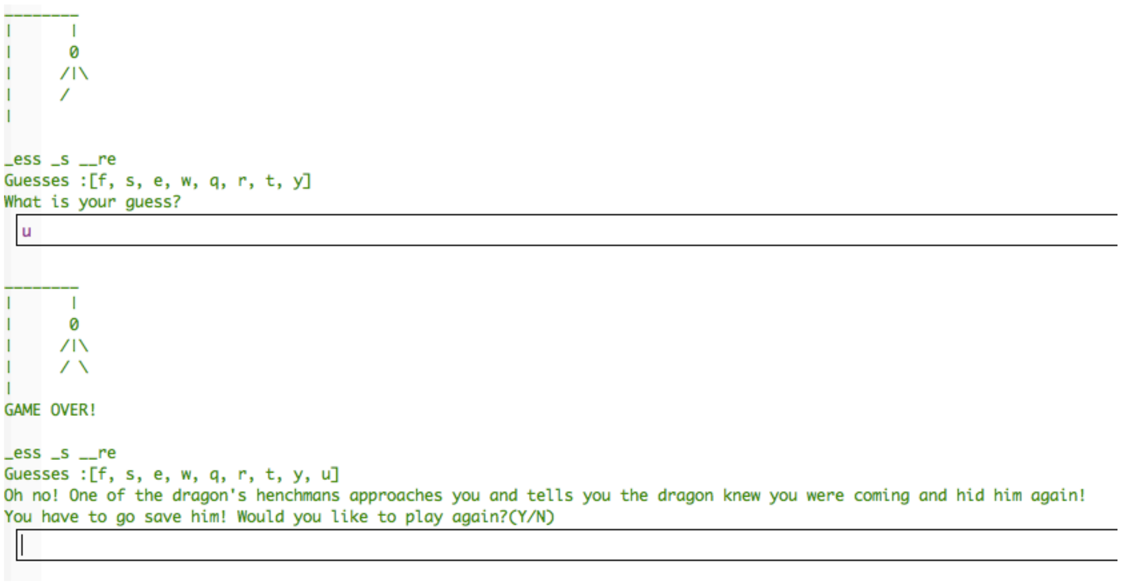
```
Enter the room number you wish to enter. Enter Q to quit. (-)locked & (+)unlocked
[10+][8-][6+]
Please select a valid door number.
10

Door is unlocked. Enter the room.
Do you wish to enter room 10?(Y/N) Enter Q to quit.
y
You are almost there! All you have to do now is win this game by guessing the correct letters in the phrase Your boo's life hangs on a thread. Hurry up and save him!

_______
|     |
|
|
|
|


__ ____ __ ____ ____
What is your guess?
```

If the user correctly guesses the phrase before they run out of chances, a congratulatory message appears on the screen and the user is asked whether or not they wish to replay the game.



```
--------
|      |
|      0
|     /
|
|

no news is goo_ news
Guesses :[a, e, i, o, t, n, s, w, g]
What is your guess?
 d

--------
|      |
|      0
|     /
|
|

no news is good news
Guesses :[a, e, i, o, t, n, s, w, g, d]
You entered the room and managed to save your Don right before the Dragon took him away! He tells you,'Aahh my hero' right before he takes you in for a kiss
     Congrat   ulation
  Congratulat s!Congratul
Congratulations!Congratulat
Congratulations!Congratulat
  Congratulations!Congrat
     Congratulations!C
       Congratulatio
          Congrat
            Con
Would you like to play again?(Y/N)
```

If the user does not guess the phrase correctly before running out of chances (6 wrong guesses are allowed), the don in distress is hidden in a new location by the dragon and the user is asked if they wish to replay the game.



```
---------
|      |
|      0
|     /|\
|     /
|

_ess _s __re
Guesses :[f, s, e, w, q, r, t, y]
What is your guess?
 u

---------
|      |
|      0
|     /|\
|     / \
|
GAME OVER!

_ess _s __re
Guesses :[f, s, e, w, q, r, t, y, u]
Oh no! One of the dragon's henchmans approaches you and tells you the dragon knew you were coming and hid him again!
You have to go save him! Would you like to play again?(Y/N)
```

```java
/**
 * PlayGame.java
 * Written By: Adrianna Valle & Jessenia Aquilar
 * Written On: Dec 19, 2016
 *
 * Driver class. User can use PlayGame class to play the game through
 * the interactions panel and user input. Instantiates a GameMap object.
 */

import java.util.Scanner;
public class PlayGame {

  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    String resp = "y";
    Boolean won = false;

    do {
      //User input to start a new game or start game for first time.
      System.out.println("Welcome to Aventure Time!!");
      System.out.println("Do you want to play? Y/N");
      resp = scan.nextLine().toLowerCase(); //nextLine to account for blank entries
      while(!resp.equals("y") && !resp.equals("n")){
        System.out.println("Please enter a valid response to proceed.");
        System.out.println("Do you want to play? Y/N");
        resp = scan.nextLine();
      }
      if(resp.equals("n"))  //User doesn't want to play anymore
        break;


      //Start game properties
      GameMap game = new GameMap();

    while(!game.endOfMap() && game.chancesLeft()){

      //Showing rooms.
      System.out.println("\nEnter the room number you wish to enter. Enter Q to
quit. (-)locked & (+)unlocked");
      System.out.println(game.printRooms());

        //Gets the response for the room number.
        String choosenRoom = "";
        while(game.getRoom(choosenRoom)==null&& !choosenRoom.equals("q")){
          try{
            System.out.println("Please select a valid door number.");
            choosenRoom = scan.nextLine().toLowerCase();
          }
          catch(IllegalArgumentException ex){
            System.out.println("Input is not a valid number entry.");
          }
          catch(NullPointerException ex){
            System.out.println("Input is not a valid number entry.");
          }
        }
        if(choosenRoom.equals("q")){  //User doesn't want to play anymore
          resp = "n";
          break;
        }

        Room selectedRoom = game.getRoom(choosenRoom);
        //Check if door is locked or not
        Door selectedDoor = game.getDoor(selectedRoom);
        if(selectedDoor.isLocked()){
          System.out.println("\n"+selectedDoor.getLockedMsg());
          String choice = "n";
          //key loop: until door is resolved or you have no more chances
          while(!choice.equals("q") &&!choice.equals("y") && game.chancesLeft()){

            //Deals with user interaction and correct response
            System.out.println("You have " + game.getChances()+" chances left to
guess wrong or the dragon carries your Don off!\n");
            System.out.println("Type the name of the key you want to view. Type Q
to quit the game");
            System.out.println(game.printKeys());

            String selectedKeyStr =scan.nextLine().toLowerCase();
            //Checks if key is in the list
            while(!game.getCurrentRoom().validKey(selectedKeyStr) && !
selectedKeyStr.equals("q")){
              System.out.println("Response is invalid. Type in a vaild key or enter
Q to quit.");
              selectedKeyStr = scan.nextLine().toLowerCase();
            }
            if(selectedKeyStr.equals("q")){  //User doesn't want to play anymore
              resp = "n";
              break;
            }

            Key choosenKey = game.getCurrentRoom().getKey(selectedKeyStr);
            System.out.println("\nThe key tells you: " +choosenKey.getActiveMsg());

            //User interaction to get it to get a vaild response
            System.out.println("Is this the key you choose to use to unlock?(Y/N).
Enter Q to Quit.");
            choice = scan.nextLine().toLowerCase();
            while(!choice.equals("y") && !choice.equals("n") && !choice.equals
("q")){
              System.out.println("Response is invalid. Enter y/n or Q to quit.");
              System.out.println("Is this the key you choose to use to unlock the
door?(Y/N). Enter Q to Quit.");
              choice = scan.nextLine().toLowerCase();
            }
            if(choice.equals("q")){
              resp = "n";
              break;
            }

            if(choice.equals("y")){
              if(selectedDoor.rightKey(choosenKey)){
                System.out.println("\n"+selectedDoor.getUnlockedMsg()+"\n");
                game.unlockDoor(selectedRoom);
              }
              else{
                System.out.println("The key says: " + choosenKey.getInactiveMsg());
                System.out.println("Oh no! That's not the right key!");
                game.wrongAnswer();
                choice = "n";
              }
            }
          }
        }
        else{     //Door is not locked
          System.out.println("\n" + selectedDoor.getUnlockedMsg());

          //Ensures correct user response
          System.out.println("Do you wish to enter room "+ selectedRoom + "?(Y/N)
Enter Q to quit.");
          String  enteringDoorResp = scan.nextLine().toLowerCase();
          while(!enteringDoorResp.equals("y") && !enteringDoorResp.equals("n") && !
enteringDoorResp.equals("q")){
            System.out.println("Response is invalid. Enter a vaild response");
            System.out.println("Is this the key you choose to use to unlock the
door?(Y/N). Enter Q to Quit.");
            enteringDoorResp = scan.nextLine().toLowerCase();
          }

          if(enteringDoorResp.equals("q")){
            resp = "n";
            break;
          }

          if(enteringDoorResp.equals("y")){
            game.setCurrentRoom(selectedRoom);
          }
        }
        if(resp.equals("n"))
          break;

    }
    if(game.endOfMap()){
     Hangman minigame = new Hangman("hangmanText.txt");
      won = minigame.playHangman();
    }
    if(won){ //winng message
      System.out.println("You entered the room and managed to save your Don right
before the Dragon took him away! He tells you,'Aahh my hero' right before he takes
you in for a kiss");
      System.out.println("     Congrat   ulation     ");
      System.out.println("  Congratulat s!Congratul  ");
      System.out.println("Congratulations!Congratulat");
      System.out.println("Congratulations!Congratulat");
      System.out.println("  Congratulations!Congrat  ");
      System.out.println("     Congratulations!C     ");
      System.out.println("       Congratulatio       ");
      System.out.println("          Congrat          ");
      System.out.println("            Con            ");
      System.out.println("Would you like to play again?(Y/N)");
      resp =scan.nextLine().toLowerCase();
      while(!resp.equals("y") && !resp.equals("n")){
        System.out.println("Response is invalid.");
        System.out.println("Would you like to play again?(Y/N)");
        resp = scan.next();
      }
    }
    else{ //game was lost
      System.out.println("Oh no! One of the dragon's henchmans approaches you and
tells you the dragon knew you were coming and hid him again!");
      System.out.println("You have to go save him! Would you like to play again?
(Y/N)");
      resp =scan.nextLine().toLowerCase();
      while(!resp.equals("y") && !resp.equals("n")){
        System.out.println("Response is invalid.");
        System.out.println("Would you like to play again?(Y/N)");
        resp = scan.next();
      }
    }
    }
    while(resp.equals("y"));
    System.out.println("Sorry to see you go! Come again!");
    scan.close();
  }
}
```

```java
/**
 * Written By: Jessenia Aguilar
 * Written On: Dec 21, 2016
 *
 * GUI class - allows user to play game through the GUI.
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PlayGameGui {
  private int rmNum;
  private Boolean won = false;
  private GameMap game;


  private JFrame frame;
  private JPanel instructs;

  private JButton rmBt1;
  private JButton rmBt2;
  private JButton rmBt3;
  private JButton rmBt4;
  private JButton startButt;

  private JLabel instructionTxt;
  private JLabel gameTxt;
  private JLabel doorTxt;
  private JLabel keyTxt;
  //frame and panel will be created but mostly just the frame
  public PlayGameGui(){
    rmNum = 1;//needs to update everytime a new room is entered
    game = new GameMap();

    instructs = new JPanel();
    instructs.setLayout(null);

    frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setPreferredSize(new Dimension(870,700));

    frame.setTitle("Adventure Time");
    frame.setResizable(false);
    frame.pack();

    rmBt1 = new JButton("");
    rmBt2 = new JButton("");
    rmBt3 = new JButton("");
    rmBt4 = new JButton("");
    startButt = new JButton("");

    instructionTxt = new JLabel("");
    gameTxt = new JLabel("");
    doorTxt = new JLabel("");
    keyTxt = new JLabel("");
    //startLabel = new JLabel("");
  }

  public void startPage(){

    JLabel titleTxt = new JLabel("<html><font face ='Courier New' color ='red'
><center><b>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;"+
                                 "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;"+
                                 "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;ADVENTURE TIME</b></center><font></html>");
    instructs.add(titleTxt);
    titleTxt.setOpaque(true);
    Color customColor0 = new Color(0,0,0);
    titleTxt.setBackground(customColor0);
    titleTxt.setBounds(190 , 0 , 680, 30);
    //190 , 0 , 680, 30

    //this will set the instructions on the side column and will not be changed
throughout the game
    instructionTxt.setText("<html><font color= 'white'><b>&nbsp;&nbsp;Adventure
Time &nbsp;&nbsp;Instructions</b><br><br>"
                                        +"&nbsp;&nbsp;This game simulates a role
playing game in which, you the user, chooses"+
                                       "a path to take in order to save your
beloved Don who is kept prisoner by the Dragon. Along your journey through this
maze, you will encounter many challenges."+
                                         "There will be doors presented to you some
of which you cannot enter so you will have to find the Keys that open the doors."+
                                       "Each door will give you a hint as to what
it requires to be opended and the doors already opened will be unlocked. Keep in
mind that all items in the room want"+
                                       "to be used so they may lie and try to trick
you to choose them, however, only one is a key to open the door. "+
                                       "If you happen to choose the wrong key three
times, we're sorry but the dragon will then become aware of your presence"+
                                       " and move your Don to a different location
and you will lose the game. Your goal is to get to the final final room"+
                                        " where you will face the final challenge
and win or lose you Don (and the game).</font><html>");
    instructionTxt.setFont(new Font("Serif", Font.PLAIN, 14));
    instructs.add(instructionTxt);
    instructs.setForeground(new Color(0xffffdd));
    Color customColor = new Color(165,48,73);
    instructionTxt.setOpaque(true);
    instructionTxt.setBackground(customColor);
    instructionTxt.setBounds(0 , 0 , 190, 678);


    //all images need to be resized and put in one folder
//     ImageIcon image1 = new ImageIcon("finalRoom3.png");
    ImageIcon image2 = new ImageIcon("finalRoom.jpg");
    JLabel label2 = new JLabel(image2);
    instructs.add(label2);
    Color customColor2 = new Color(0,0,0);
    label2.setOpaque(true);
    label2.setBackground(customColor2);
    label2.setBounds(190 , 30, 680, 370);


    /**to get images to appear
    ImageIcon appleimg = new ImageIcon("apple2.jpg");
    JButton bt1 = new JButton(appleimg);
    //bt1.setIcon(appleimg);
    bt1.setOpaque(true);

    instructs.add(bt1);
    Color customColor6 = new Color(255,255,255);

    bt1.setBackground(customColor6);
    bt1.setBounds(230,60,40,40);**/

    /*door buttons
    JButton bt2 = new JButton("Room somthing");
    instructs.add(bt2);
    bt2.setBounds(200,410, 120,35);
    startButt.setVisible(false);
    JButton bt3 = new JButton("Room somthing");
    //instructs.add(bt3);
    bt3.setBounds(350,410, 120,35);
    //bt3.setBounds();
    JButton bt4 = new JButton("this will be a string var");
    instructs.add(bt4);
    bt4.setBounds(500,410, 120,35);
    //bt4.setBounds();

    //bt3 = new JButton("something else");
    //instructs.add(bt3);
    //bt3.setBounds(350,410, 120,35);
    **/

    String x = "Welcome to ADVENTURE TIME";
    gameTxt.setText("<html><font color = 'green'>" + x+"");
    //JButton gameTxt = new JButton("test");
    instructs.add(gameTxt);
    Color customColor3 = new Color(255,218,98);
    gameTxt.setOpaque(true);
    gameTxt.setBackground(customColor3);
    gameTxt.setBounds(190 , 400, 680, 235);



    startButt = new JButton("Start Game");
    JLabel startLabel = new JLabel("");
    instructs.add(startButt);
    //Color customColor4 = new Color(165,48,73);
    //startButt.setOpaque(true);
    //startButt.setBackground(customColor4);
    startButt.setBounds(480 , 640, 100, 30);
    //event e = new event();
    //startButt.addActionListener(e);

    JButton quit = new JButton("Exitff");
    event2 ev = new event2();
    quit.addActionListener(ev);
    instructs.add(quit);
    quit.setBounds(600 , 640, 100, 30);

    frame.add(instructs)  ;
    //frame.pack();
    frame.setVisible(true);
  }

  //will be called by action listener event
  public void startGame(){
    startButt.setVisible(false);
    gameTxt.setText("");
    //image2.setIcon(new ImageIcon("room"+ rmNum + ".jpg")); needs to be the name
of the rooms
    //GameMap game = new GameMap();
    //this will call a different method


  }
  //
//  public void roomInput(){
//
//    if(!game.getCurrentRoom().equals(game.getFinalRoom()) && game.chancesLeft()){
//      gameTxt.setText("Click the room number you wish to enter. ");
//      //LinkedList neighbors = game.getConnectingRooms();
//      //generates rooom buttons
//      for (int i = 0 ; i < neighbors.size(); i++){
//        String name = neighbors.get(i).getRoomName();
//        jButton b = new jButton(name);
//        panel.add(b);
//        b.setBounds(200+(150 * i),410, 120,35);
//      }
//    }
//  }
//
//  public void makeButton(JButton bt){
//
//  }
//  //going to call start game method
//  public class event implements ActionListener{
//    public void actionPerformed(ActionEvent e){
//     // startLabel.setText("Now you can see words here and your input" );
//    }
//  }

   public class event2 implements ActionListener{
    public void actionPerformed(ActionEvent ev) {
      System.exit(0);
    }

  }

  public static void main(String[] args){
    PlayGameGui n = new PlayGameGui();
    n.startPage();
  }
}
```

```java
/**
 * GameMap.java
 * Written By: Lauren Luo & Adrianna Valle
 * Written On: Dec 16, 2016
 *
 * Game map creates the conditions for a game using information that a user enters 
for the
 * key text responses, the door text responses, the images, etc. Note that the 
rooms and weights do not correspond
 * nor do they need to be in a certain order with regards to the map.
 *
 * This class allows you to create a game in which the weights of the entered .tgf 
file corresponds
 * to a specified door reference. If the door contains a weight of zero, that mean 
the door is unlocked
 * and no further action is required.
 *
 * TRAVERSING:Every key value present contains a corresponding door value in which 
they interact together depending on
 * the decisions the user makes. These values are saved in their own dictionary. 
Every room contains keys that corresponds
 * to all of the weighted edges[represented as doors]that connect to other rooms. 
In total, every room contains
 * 10 keys where some keys correspond to a door that connects from the room as some
that are used merely to challenge
 * the player.
 *
 * ORIENTATION: The number value assigned to the room name is used to help the user
orient themselves around the game.
 * The structure is maze like and therefore, the user can not enter the same room 
various times in order to find the end.
 *
 * HOW TO WIN: The player does not necessarily need to unlock all the doors but 
must reach the assigned final room and
 * complete the minigame. What's more, the player needs to not run out of chances 
or the lose the game.
 *
 * The user of this class MUST:
 * #1Create a .tgf with vertices that correspond to a saved image file name.
 * #2 Not include multiple edges of the same weight unless it is of weight 0.
 * #3 Must create the Needed Text file for the Key and Dictionary in the exact 
format specified throughout the program.
 *
 * OVERALL GAME EXPANSION:
 * 1--> Control more of the IO input.
 * 2--> Automatically determine best starting and final room ad sets it
 * 3--> Use polymorphism to create diversity in key door interaction.
 * 4--> More game features such as mingames, player character
 */
import java.util.*;
import java.io.*;

public class GameMap {
  //instance variables
  private AdjMatGraph<String> map;
  private Room currentRoom, finalRoom;
  private LinkedList<Room> allRooms;
  private Hashtable<Room, LinkedList<Room>> rooms;
  private static Hashtable<Integer, Key> allKeys; //will contain 25 keys
  private static Hashtable<Integer, Door> allDoors;
  private int chancesLeft;
  private static final int MAX_CHANCES = 3;

  public GameMap() {
    map = new AdjMatGraph<String>();
    try {
      AdjMatGraph.loadTGF("gameMap.tgf", map);   //Check if valid tgf file
    } catch (FileNotFoundException ex) {
      System.out.println("error: file not found");
    }

    chancesLeft = MAX_CHANCES;
    allRooms = new LinkedList<Room>();
    rooms = new Hashtable<Room, LinkedList<Room>>();
    allKeys = new Hashtable<Integer, Key>();
    allDoors = new Hashtable<Integer, Door>();

    buildKeyDict("keyText.txt");
    buildDoorDict("doorText.txt");
    buildRooms();

    setCurrentRoom(findRoom("1room"));
    setFinalRoom(findRoom("10room"));
  }

  /** Builds the key Dictionary out of a text file that contains the key name, a 
message to give
     * when user first interacts with it and a message it gives if it is not one of 
the
     * designated keys for the room.
     *
     * EXPANSION CAPABILITIES: Allows for the key dictionary to expand increasing 
the possible map size.
     * ASSUMPTIONS: Assumes the user enters a vaild format of the necessary text.
```

```java
Note some special characters
    * are not accepted.
    * FUTURE EXPANSION:Could allow for a scanner option for input and resolve special character issue
    *
    * @param: String TextFile @return: none*/
  private void buildKeyDict(String fileIn) {
    try {
      Scanner sc = new Scanner(new File(fileIn));
      int count = 1;

      while (sc.hasNextLine()) {

        String[] temp = sc.nextLine().trim().split(" \\+ ");
        allKeys.put(count, new Key(temp[0], temp[1], temp[2]));
        count++;
      }
      sc.close();
    } catch (IOException ex) {
      System.out.println("error in reading keys from file");
    }
  }

  /** Builds the Door Dictionary out of a text file that contains a messsage when the door
    * is locked and a message it returns when it is sucessfully unlocked by a key.
    *
    * EXPANSION CAPABILTIES: Allows for the number of doors to increase therefore the number
    * of connections between rooms.
    * ASSUMPTIONS: Assumes user formatted the file correctly. Note some special characters are not accepted.
    * FUTURE EXPANSION: Could allow for a scanner option for input and could allow for special characters.
    *
    * @param: String TextFile @return: none*/
  private void buildDoorDict(String fileIn) {
    try {
      Scanner sc = new Scanner(new File(fileIn));
      int count = 0;
      while (sc.hasNextLine()) {
        String[] temp = sc.nextLine().trim().split(" \\+ ");
        allDoors.put(count, new Door(count, temp[0], temp[1]));
        count++;
      }
      sc.close();
    } catch (IOException ex) {
      System.out.println("error in reading doors from file");
    }
  }

  /**Builds the rooms from the tgf vertices where all of the names correspond to a saved img. In building
    * the rooms also finds the keys that must go in the room[determined by the weighted edges which are
    * equivalent to the doors] from allKeys and stores it. Also, creates a dictionary of the rooms, and one
    * for the connecting rooms to refer to.
    * EXAPNSION CAPABILITIES: Allows you to build lots of rooms
    * ASSUMPTIONS: User enters correct img filename.
    * FUTURE EXPANSION: It may be better to organize our rooms in a different structure.*/
  private void buildRooms() {
    //Loads the img names[without '.jpg'] onto an array
    String[] vertices = new String[map.n()];
    for (int h = 0; h < map.n(); h++) {
      vertices[h] = map.getVertex(h);
    }

    //Gets the connecting room names for each room
    for (int i = 0; i < map.n(); i++) {
      LinkedList<String> successors = map.getSuccessors(vertices[i]);

      //Loads the keys that correspond to any of the locked doors that connect from the room.
      LinkedList<Key> activeKeys = new LinkedList<Key>();
      for (int j = 0; j < successors.size(); j++) {
        int keyNum = map.getWeight(vertices[i], successors.get(j));
        if(keyNum!=0)                                //no key is loaded for a weight of zero.
          activeKeys.add(allKeys.get(keyNum));

      }

      String fileName = vertices[i] + ".jpg";
      allRooms.add(new Room(fileName, activeKeys)); //Creates the room with coresponding img name and keys & stores it.
    }

    //Creates a Dictionary of all the rooms with their connecting rooms
    for (int k = 0; k < map.n(); k++) {
      LinkedList<String> successors = map.getSuccessors(vertices[k]);
      LinkedList<Room> successorRooms = new LinkedList<Room>();

      for (int m = 0; m < successors.size(); m++) {
        successorRooms.add(findRoom(successors.get(m)));
      }
      rooms.put(findRoom(vertices[k]), successorRooms);
    }
  }

  /*SETTER: Allows the user to set the starting room. Setting also allows for
   * a room traversal to be simulated as the player enters different rooms.
   *
   * EXPANSION CAPABILITES: Can choose any room to be the first room.
   * ASSUPTIONS: none
   * FUTURE EXPANSION: can select any room at random and from this, use this point
of reference
   * to create pointer to final room
   * @param: Room selectedRoom @return: -- */
  public void setCurrentRoom(Room selectedRoom) {
    currentRoom = selectedRoom;
  }

  /**GETTER: Returns the current room
    * @param: -- @reuturn: Room currentRoom */
  public Room getCurrentRoom() {
    return currentRoom;
  }

  /*SETTER: Allows the user to set the final room.
   *
   * EXPANSION CAPABILITES: Can choose any room to be the final room.
   * ASSUPTIONS: The room isn't close enough to the start where it will end the
game to quickly
   * FUTURE EXPANSION: Using a traversal to find a far enoguh room to be the future
room. User won't need to
   * manually enter it.
   * @param: Room selectedRoom @return: -- */
  public void setFinalRoom(Room selectedRoom) {
    finalRoom = selectedRoom;
  }

  /**GETTER: Returns the final room
    * @param: -- @reuturn: Room finalRoom */
  public Room getFinalRoom() {
    return finalRoom;
  }

  /** Returns the Room with the corresponding room name. If no room is found,
returns null
    * NOTE: When using findRoom, check if null before assigning it anywhere.
    * @param: String roomName @return: Room correspondingRoom */
  private Room findRoom(String roomName) {
    for (int i = 0; i < map.n(); i++) {
      if (allRooms.get(i).getRoomName().equals(roomName)) {
        return allRooms.get(i);
      }
    }
    return null;
  }

  /** Returns the room that is in the list of the connecting rooms to the current
room. If the roomName
    * is not a corresponding room, returns null.
    * Note: When using getRoom, check if null before assigning it anywhere.
    * @param: String roomName @return: Room correspondingRoom*/
  public Room getRoom(String roomName) {
    for (int i = 0; i < getConnectingRooms().size(); i++) {
      if (getConnectingRooms().get(i).getRoomName().equals(roomName+"room")) {
        return getConnectingRooms().get(i);
      }
    }
    return null;
  }

  /** Returns a list of the rooms that connect to the current room.
    * @param: -- @return: LinkedList<Room> connectingRooms */
  public LinkedList<Room> getConnectingRooms() {
    return rooms.get(currentRoom);

  }

  /** Returns true if user player reaches the final room.
    * @param: -- @return: boolean foundFinalRoom */
  public boolean endOfMap(){
    return currentRoom.equals(finalRoom);
  }


  /**GETTER: Returns a randomly selected key from the key Dictionary.Used to fill
the spots of the remaining
    * keys needed[to make 10] in order to challenge the user.
    * @param: -- @return: Key randomKey */
  public static Key getRandomKey() {
    int ran = (int)((Math.random() * 25) + 1);
    return allKeys.get(ran);
  }

  /**Returns all of the keys that exist. Static in order to be accessed by the room class.
    * @param: -- @return: Hashtable<Integer, Key> allKeys */
  public static Hashtable<Integer, Key> getAllKeys() {
    return allKeys;
  }

  /**Returns the Door of that connects the currentRoom with the selectedRoom by getting the weight
    * and pulling the assigned integer door value from the door dictionary.
    * @param: Room selectedRoom @return: Door connectingDoor */
  public Door getDoor(Room selectedRoom) {
    int doorNum = map.getWeight(currentRoom.getRoomName(), selectedRoom.getRoomName());
    return allDoors.get(doorNum);
  }

  /** Replaces the door poniter to that of door zero to simulate the door being unlocked.
    * After this, the door can be entered however many times and would still be unlocked.
    * @param: @return: -- */
  public void unlockDoor(Room lockedRoom) {
    map.addEdge(currentRoom.getRoomName(),lockedRoom.getRoomName(),0);
  }


  /** Returns whether the player still have chances to guess wrong.
    * @return: boolean chancesExist @param: -- */
  public boolean chancesLeft() {
    return chancesLeft > 0;
  }

  /**Returns the number of chances left to guess wrong.
    * @param: -- @return: int numberOfChances */
  public int getChances() {
    return chancesLeft;
  }

  /** Decrements the number of chances when the player guesses incorrectly.
    * @param: --  @return: -- */
  public void wrongAnswer() {
    chancesLeft--;
  }

  /** Prints a string representation of the possible rooms the player can enter. Only used for the Driver
    * class.*/
  public String printRooms() {
    String s = "";
    LinkedList<Room> neighbors = getConnectingRooms();
    for (int i = 0; i < neighbors.size(); i++) {
      String name = neighbors.get(i).getRoomName();
      s += "[" + name.substring(0,name.indexOf("r"));
      if (map.getWeight(currentRoom.getRoomName(),neighbors.get(i).getRoomName()) == 0) {
        s += "+";
      } else {
        s += "-";
      }
      s += "]";
    }
    return s;
  }

  /** Prints a string representation of the possible keys a player can enter. Only used for the Driver Class. */
  public String printKeys() {
    String s = "";
    Key[] currentKeys = currentRoom.getRoomKeys();
    for (int i = 0; i < currentKeys.length; i++) {
      s += "[" + currentKeys[i].getName() + "]";
    }
    return s;
  }

  public static void main(String[] args) {
    //Testing AdjMatGraph with game map graph, with (test) and without weights (test2)
//    AdjMatGraph<String> test = new AdjMatGraph<String>();
//    try {
//      AdjMatGraph.loadTGF("gameMapNoWeights.tgf", test);
//    } catch (FileNotFoundException ex) {
//      System.out.println("error: file not found");
//    }
//    System.out.println(test);

//    AdjMatGraph<String> test2 = new AdjMatGraph<String>();
//    try {
//      AdjMatGraph.loadTGF("gameMap.tgf", test2);
//    } catch (FileNotFoundException ex) {
//      System.out.println("error: file not found");
//    }
//    System.out.println(test2);

//  GameMap map = new GameMap();

//    map.printAllKeys();
//    System.out.println();
//    map.printAllDoors();
//    System.out.println();
//    map.printAllRoomNeighbors();
//    System.out.println();
//    map.printRooms();
//    map.printMapGraph();
//    System.out.println(map.contains(4));
  }
}
```

```
/**
 * Room.java
 * for Adventure Time
 * Created by: Jessenia Aguilar-Hernandez
 * Modified By: Lauren Luo & Adrianna Valle
 *
 * Creates a Room object that contains an array of 10 keys.
 * For each room that is connected to this room, the karray
 * holds a key that can unlock the door to the connecting room.
 */
import java.util.*;

public class Room implements Comparable<Room> {
  //instance variables
  private String img;
  private Key[] karray;//holds the keys in the room
  private final int DEFAULT_CAPACITY = 10;

  /**
   * Constuctor takes in imagefile name, and the active key in the room
   */
  public Room(String imgFile, LinkedList<Key> activeKeys){
    img = imgFile;

    karray = new Key[DEFAULT_CAPACITY]; //each room will always have 10 keys
    for (int i = 0; i < activeKeys.size(); i++) {
      karray[i] = activeKeys.get(i);
    }

    for (int i = activeKeys.size(); i < karray.length; i++) {
      Key temp = GameMap.getRandomKey(); //assigns random keys to the remaining key
slots in karray
      while(contains(temp)) {
        temp = GameMap.getRandomKey();
      }
      karray[i] = temp.copyKey();
    }

    /*Shuffles the key placement so that all the active keys are not
     in the beginning*/
    Random ran = new Random();
    for(int j = 0; j<karray.length; j++) {
      int swapVal = ran.nextInt(karray.length-1)+1;
      Key tempEle = karray[swapVal];
      karray[swapVal] = karray[karray.length-1-j];
      karray[karray.length-1-j] = tempEle;
    }
  }

  /**
   * Returns true if karray contains given key
   * Otherwise returns false
   */
  private boolean contains(Key kIn) {
    for (int i = 0; i < karray.length; i++) {
      if (karray[i] != null && karray[i].getName().equals(kIn.getName())) {
        return true;
      }
    }
    return false;
  }

  public String getRoomName() {
    return img.substring(0,img.indexOf("."));
  }

  /**
   * Getter: @return the keys in the room
   */
  public Key[] getRoomKeys(){
    return karray;
  }

  /**
   * Checks if the input key name is in this room's karray (is it a valid key)
   * @return boolean true if valid, else false
   */
  public boolean validKey(String keyString) {
    for (Key k: karray) {
      if (k.getName().toLowerCase().equalsIgnoreCase(keyString)) {
        return true;
      }
    }
    return false;
  }

  /**
   * Takes in name of the key and returns the Key object with that name.
   * If key doesn't exist in this room's karray, returns null.
   */
  public Key getKey(String nameOfSelectedKey) {
    for (int i = 0; i < karray.length; i++) {
      if (karray[i].getName().equalsIgnoreCase(nameOfSelectedKey)) {
        return karray[i];
```

```java
      }
    }
    return null;
  }

  /**
   * compareTo: @return returns an int based on the string compareTo of room names
   */
  public int compareTo(Room other){

    //calls on integers compareTo
    return getRoomName().compareTo(other.getRoomName());
  }

  /**
   * Returns String representation of the object.
   * For testing purposes
   */
  public String toString() {
    return img.substring(0,img.indexOf("r"));


//    String s = "name: " + getImage(); //+ "\nkey(s): ";
//
//    for (int i = 0; i < karray.length; i++) {
//      s += "[" + karray[i].getName() + "] ";
//    }
//
//    return s;
  }

  //testing
  public static void main(String[] args){
//    Key k1 = new Key("apple","jals;fjdlkf", "adfhkjsdhfj");
//    Key k2 = new Key("toy","jdsl;jaf", "jdsklfjds");
//
//    Room r1 = new Room("somefilename", k1);
//
//    System.out.println("6 - ");
//    System.out.println(r1.getKey(0).getName());
//
//    Room r2 = r1;
//
//    System.out.println("Should be 0 - "+r2.compareTo(r1));
//
//    Room r3 = new Room("somefilename", k2);
//    r3.addKey(k1);
//
//    System.out.println("Should be 0 - "+r2.compareTo(r3));
//
//    Room r4 = new Room("somefilename", k2);
//    r4.addKey(k1);
//    r4.addKey(k1);
//    System.out.println("Should be less than 0 : "+r2.compareTo(r4));
//
//    Room r5 = new Room("anotherfilename",k1);
//    r5.addKey(k1);
//    System.out.println("Should be greater than 0 ? "+r2.compareTo(r5));
  }
}
```

```java
/**
 * Key.java
 * for Adventure Time
 * Created by: Jessenia Aguilar-Hernandez
 * Modified By: Lauren Luo & Adrianna Valle
 *
 * This class creates Key Objects for the Game. It takes in the image name,
 * the message that is given when the key is selected and a message that is given,
 * when the key is not the correct one.
 */

public class Key implements Comparable<Key> {
  //instance variables
  private String keyName; //the name of the object
  private String activeMsg; //response if key is active
  private String inactiveMsg; //response if key is inactive

  /**
   * Constructor takes in key name and text for user interaction purposes
   */
  public Key(String k, String activeMsg, String inactiveMsg){
    keyName = k;
    this.activeMsg = activeMsg;
    this.inactiveMsg = inactiveMsg;
  }

  /**
   * Returns the name of the key
   */
  public String getName(){
    return keyName;
  }

  /** Returns respose of key when first selected. */
  public String getActiveMsg(){
    return activeMsg;
  }

  /** Returns the response of the key if not the correct key.  */
  public String getInactiveMsg(){
    return inactiveMsg;
  }

  /** Returns an individual copy of the key */
  public Key copyKey() {
    return new Key(keyName, activeMsg, inactiveMsg);
  }

  /** Compares this key with other key object by comparing their string name */
  public int compareTo(Key other) {
    return keyName.compareTo(other.getName());
  }

  /** Returns a string representation of the key */
  public String toString() {
    return keyName;
  }

  public static void main(String[] args){
    //Key el = new Key("apple","");
//    System.out.println(el.getKey());
//    el.setInstructions("Find the item!");
//    System.out.println(el.getInstructions());

  }
}
```

```java
/**
 * Door.java
 * Written By: Lauren Luo & Adrianna Valle
 * Date:12-10-16
 *
 * This class simulates a Door object in which it takes in a weight value, which corresponds to
 * a designated key, a message when the door is locked and a message when the door is unlocked.
 * Door locking/unlocking allows for another layer of fun for the player on top of the maze-like
 * structure of our game.
 *
 * Note that a weight of 0 means the door is unlocked and no locked msg is present.
 */

public class Door {
  private String lockedMsg, unlockedMsg;
  private int weight;

  /** Constructor */
  public Door(int weight,  String lockedMsg, String unlockedMsg) {
    this.weight = weight;
    this.lockedMsg = lockedMsg;
    this.unlockedMsg = unlockedMsg;
  }

  /** Checks to see if the key is the correct one to unlock this particular door.
    * @param: Key checkKey @return: boolean rightKey*/
  public boolean rightKey(Key key){
   return GameMap.getAllKeys().get(weight).compareTo(key) == 0;
  }

  /** Returns the weight of the door. */
  public int getWeight() {
    return weight;
  }

  /** Returns the unlocked message of the door. */
  public String getUnlockedMsg() {
    return unlockedMsg;
  }

  /** Returns the locked message of the door. */
  public String getLockedMsg() {
    return lockedMsg;
  }

  /** Returns whether the door is locked or not. */
  public boolean isLocked() {
    return weight > 0;
  }

  /** Returns formatted String representation of Door object. */
  public String toString() {
    return weight + " || " + lockedMsg + " || " + unlockedMsg;
  }
}
```

```java
import java.util.LinkedList;

/**
 * DO NOT CHANGE THIS FILE.
 *
 * A basic Graph interface
 */
public interface Graph<T> {
  /** Returns true if this graph is empty, false otherwise. */
  public boolean isEmpty();

  /** Returns the number of vertices in this graph. */
  public int n();

  /** Returns the number of arcs in this graph. */
  public int m();

  /** Returns true iff a directed edge exists from v1 to v2. */
  public boolean isArc (T vertex1, T vertex2);

  /** Returns true iff an edge exists between two given vertices
    * which means that two corresponding arcs exist in the graph */
  public boolean isEdge (T vertex1, T vertex2);

  /** Returns true IFF the graph is undirected, that is, for every
    * pair of nodes i,j for which there is an arc, the opposite arc
    * is also present in the graph.  */
  public boolean isUndirected();

  /** Adds a vertex to this graph, associating object with vertex.
    * If the vertex already exists, nothing is inserted. */
  public void addVertex (T vertex);

  /** Removes a single vertex with the given value from this graph.
    * If the vertex does not exist, it does not change the graph. */
  public void removeVertex (T vertex);

  /** Inserts an arc from vertex1 to vertex2.
    If the vertices exist. Else it does not change the graph. */
  public void addArc (T vertex1, T vertex2, int weight);

  /** Removes an arc from vertex v1 to vertex v2,
    * if the vertices exist. Else it does not change the graph. */
  public void removeArc (T vertex1, T vertex2);

  /** Inserts an edge between two vertices of this graph,
    * if the vertices exist. Else does not change the graph. */
  public void addEdge (T vertex1, T vertex2, int weight);

  /** Removes an edge between two vertices of this graph,
    if the vertices exist, else does not change the graph. */
  public void removeEdge (T vertex1, T vertex2);

  /** Retrieve from a graph the vertices x following vertex v (v->x)
    and returns them onto a linked list */
  public LinkedList<T> getSuccessors(T vertex);

  /** Retrieve from a graph the vertices x pointing to vertex v (x->v)
    and returns them onto a linked list */
  public LinkedList<T> getPredecessors(T vertex);

  /** Returns a string representation of the adjacency matrix. */
  public String toString();

  /** Saves the current graph into a .tgf file.
    If it cannot save the file, a message is printed. */
  public void saveTGF(String tgf_file_name);

}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Scanner;

/**
 *
 * Created: CS Team
 * Modified: Adrianna Valle
 * Date: 12-04-16
 *
 * AdjMatGraph has been optimized to include and consider weighted edges. Everything else was kept
 * constant. An AdjMatGraph was used instead of an AdjacencyList in order to allow for both
 * expansion capabiltiies as well as easy accessiblity to succeeding vertices/weights.
 */
public class AdjMatGraph<T> implements Graph<T>, Iterable<T> {
  public static final int NOT_FOUND = -1;
  private static final int DEFAULT_CAPACITY = 1; // Small so that we can test expand
  private static final boolean VERBOSE = false;  // print while reading TGF?

  private int n;   // number of vertices in the graph
  private Integer[][] arcs;   // adjacency matrix of arcs
  private T[] vertices;   // values of vertices

  /****************************************************************
    Constructor. Creates an empty graph.
    *****************************************************************/
  @SuppressWarnings("unchecked")
  public AdjMatGraph() {
    n = 0;
    this.arcs = new Integer[DEFAULT_CAPACITY][DEFAULT_CAPACITY];
    this.vertices = (T[])(new Object[DEFAULT_CAPACITY]);
  }

  /********** NEW METHODS ******************************************/
  /**
   * Construct a copy (clone) of a given graph.
   * The new graph will have all the same vertices and arcs as the original.
   * A *shallow* copy is performed: the graph structure is copied, but
   * the new graph will refer to the exact same vertex objects as the original.
   */
  @SuppressWarnings("unchecked")
  public AdjMatGraph(AdjMatGraph<T> g) {
    n = g.n;
    vertices = (T[]) new Object[g.vertices.length];
    arcs = new Integer[g.arcs.length][g.arcs.length];
    for (int i = 0; i < n; i++) {
      vertices[i] = g.vertices[i];
      for (int j = 0; j < n; j++) {
        arcs[i][j] = g.arcs[i][j];
      }
    }
  }

  /****************************************************************
    * Load vertices and edges from a TGF file into a given graph.
    * @param tgfFile - name of the TGF file to read
    * @param g - graph to which vertices and arcs will be added.
    *           g must be empty to start!
    * @throws FileNotFoundException
    ****************************************************************/
  public static void loadTGF(String tgf_file_name, AdjMatGraph<String> g) throws FileNotFoundException {
    if (!g.isEmpty()) throw new RuntimeException("Refusing to load TGF data into non-empty graph.");
    Scanner fileReader = new Scanner(new File(tgf_file_name));
    // Keep a mapping from TGF vertex ID to AdjMatGraph vertex ID.
    // This allows vertex IDs to be written out of order in TGF.
    // It also supports non-integer vertex IDs.
    HashMap<String,Integer> vidMap = new HashMap<String,Integer>();
    try {
      // Read vertices until #
      while (fileReader.hasNext()) {
        // Get TGF vertex ID
        String nextToken = fileReader.next();
        if (nextToken.equals("#")) {
          break;
        }
        vidMap.put(nextToken, g.n());
        String label = fileReader.hasNextLine() ? fileReader.nextLine().trim() : fileReader.next();
        if (VERBOSE) {
          System.out.println("Adding vertex " + g.n() + " (" + nextToken + " = \"" + label + "\")");
        }
        g.addVertex(label);
      }

      // Read edges until EOF
      while (fileReader.hasNext()) {
        // Get src and dest
        String src = fileReader.next();
        String dest = fileReader.next();
        // Discard label if any
        int label= -1;
        if (fileReader.hasNextLine()) {
          try{
            label = Integer.parseInt(fileReader.nextLine().trim());
          }
          catch(NumberFormatException ex){
            label = 1;
          }
        }
        g.addArc(vidMap.get(src), vidMap.get(dest),label);
      }
    } catch (RuntimeException e) {
      System.out.println("Error reading TGF");
      throw e;
    } finally {
      fileReader.close();
    }
  }


  /**
   * An iterator that iterates over the vertices of an AdjMatGraph.
   */
  private class VerticesIterator implements Iterator<T> {
    private int cursor = 0;

    /** Check if the iterator has a next vertex */
    public boolean hasNext() {
      return cursor < n;
    }

    /** Get the next vertex. */
    public T next() {
      if (cursor >= n) {
        throw new NoSuchElementException();
      } else {
        return vertices[cursor++];
      }
    }

    /** Remove is not supported in this iterator. */
    public void remove() {
      throw new UnsupportedOperationException();
    }
  }

  /**
   * Create a new iterator that will iterate over the vertices of the array when
asked.
   * @return the new iterator.
   */
  public Iterator<T> iterator() {
    return new VerticesIterator();
  }

  /**
   * Check if the graph contains the given vertex.
   */
  public boolean containsVertex(T vertex) {
    return getIndex(vertex) != NOT_FOUND;
  }


  /**** FAMILIAR METHODS *******************************************/


  /*****************************************************************
    Returns true if the graph is empty and false otherwise.
    *****************************************************************/
  public boolean isEmpty() {
    return n == 0;
  }

  /*****************************************************************
    Returns the number of vertices in the graph.
    *****************************************************************/
  public int n() {
    return n;
  }

  /*****************************************************************
    Returns the number of arcs in the graph by counting them.
    *****************************************************************/
  public int m() {
    int total = 0;

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (arcs[i][j]!=null) {
          total++;
        }
      }
    }
    return total;
  }

  /**
   * Returns array of all vertices.
   */
  public T[] getVertices() {
    return vertices;
  }

  /*****************************************************************
    Returns true iff a directed edge exists from v1 to v2.
    *****************************************************************/
  public boolean isArc(T srcVertex, T destVertex) {
    int src = getIndex(srcVertex);
    int dest = getIndex(destVertex);
    return src != NOT_FOUND && dest != NOT_FOUND && arcs[src][dest]!=null;
  }

  /*****************************************************************
    Returns true iff an arc exists between two given indices.
    @throws IllegalArgumentException if either index is invalid.
    *****************************************************************/
  protected boolean isArc(int srcIndex, int destIndex) {
    if (!indexIsValid(srcIndex) || !indexIsValid(destIndex)) {
      throw new IllegalArgumentException("One or more invalid indices: " + srcIndex
+ ", " + destIndex);
    }
    return arcs[srcIndex][destIndex]!=null;
  }

  /*****************************************************************
    Returns true iff an edge exists between two given vertices
    which means that two corresponding arcs exist in the graph.
    *****************************************************************/
  public boolean isEdge(T srcVertex, T destVertex) {
    int src = getIndex(srcVertex);
    int dest = getIndex(destVertex);
    return src != NOT_FOUND && dest != NOT_FOUND && isArc(src, dest) && isArc(dest,
src);
  }

  /*****************************************************************
    Returns true IFF the graph is undirected, that is, for every
    pair of nodes i,j for which there is an arc, the opposite arc
    is also present in the graph.
    *****************************************************************/
  public boolean isUndirected() {
    for (int i = 1; i < n(); i++) {
      // optimize to avoid checking pairs twice.
      for (int j = 0; j < i; j++) {
        if (arcs[i][j] != arcs[j][i]) {
          return false;
        }
      }
    }
    return true;
  }

  /*****************************************************************
    Adds a vertex to the graph, expanding the capacity of the graph
    if necessary.  If the vertex already exists, it does not add it again.
    *****************************************************************/
  public void addVertex (T vertex) {
    if (getIndex(vertex) != NOT_FOUND) return;
    if (n == vertices.length) {
      expandCapacity();
    }

    vertices[n] = vertex;
    //  for (int i = 0; i <= n; i++) {
    //  // if (arcs[n][i] || arcs[i][n]) throw new RuntimeException("Corrupted
AdjacencyMatrix");
    //    arcs[n][i] = false;
    //    arcs[i][n] = false;
    //  }
    n++;
  }

  /*****************************************************************
    Helper. Creates new arrays to store the contents of the graph
    with twice the capacity.
    *****************************************************************/
  @SuppressWarnings("unchecked")
  private void expandCapacity() {
    T[] largerVertices = (T[])(new Object[vertices.length*2]);
    Integer[][] largerAdjMatrix =
      new Integer[vertices.length*2][vertices.length*2];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        largerAdjMatrix[i][j] = arcs[i][j];
      }
      largerVertices[i] = vertices[i];
    }

    vertices = largerVertices;
    arcs = largerAdjMatrix;
  }

  /*****************************************************************
    Removes a single vertex with the given value from the graph.
    Uses equals() for testing equality.
    *****************************************************************/
  public void removeVertex (T vertex) {
    int index = getIndex(vertex);
    if (index != NOT_FOUND) {
      removeVertex(index);
    }
  }

  /*****************************************************************
    Helper. Removes a vertex at the given index from the graph.
    Note that this may affect the index values of other vertices.
    @throws IllegalArgumentException if the index is invalid.
    *****************************************************************/
  protected void removeVertex (int index) {
    if (!indexIsValid(index)) {
      throw new IllegalArgumentException("No such vertex index");
    }
    n--;

    // Remove vertex.
    for (int i = index; i < n; i++) {
      vertices[i] = vertices[i+1];
    }

    // Move rows up.
    for (int i = index; i < n; i++) {
      for (int j = 0; j <= n; j++) {
        arcs[i][j] = arcs[i+1][j];
      }
    }

    // Move columns left
    for (int i = index; i < n; i++) {
      for (int j = 0; j < n; j++) {
        arcs[j][i] = arcs[j][i+1];
      }
    }

    // Erase last row and last column
    for (int a = 0; a < n; a++) {
      arcs[n][a] = null;
      arcs[a][n] = null;
    }
  }

  /*****************************************************************
    Inserts an edge between two vertices of the graph.
    If one or both vertices do not exist, ignores the addition.
    An edge cannot be pointed to itself therefore is omitted.
    *****************************************************************/
  public void addEdge(T vertex1, T vertex2, int weight) {
    int index1 = getIndex(vertex1);
    int index2 = getIndex(vertex2);
    if (index1 != NOT_FOUND && index2 != NOT_FOUND && index1!=index2) {
      addArc(index1, index2, weight);
      addArc(index2, index1, weight);
    }
  }

  /*****************************************************************
    Inserts an arc from srcVertex to destVertex.
    If the vertices exist, else does not change the graph. Method
    doesn't allow the vertex to ave an edge to itself.
    *****************************************************************/
  public void addArc(T srcVertex, T destVertex, int weight) {
    int src = getIndex(srcVertex);
    int dest = getIndex(destVertex);
    if (src != NOT_FOUND && dest != NOT_FOUND && src!=dest) {
      addArc(src, dest, weight);
    }
  }

  /*****************************************************************
    Helper. Inserts an edge between two vertices of the graph. Note, an arc
    does not point to itself in the context of our game therefore the ability has
    been omitted.
    @throws IllegalArgumentException if either index is invalid.
    *************************************************************/
  protected void addArc(int srcIndex, int destIndex,int  weight) {
    if (!indexIsValid(srcIndex) || !indexIsValid(destIndex)&& srcIndex== destIndex)
{
      throw new IllegalArgumentException("One or more invalid indices: " + srcIndex
+ ", " + destIndex);
    }
    arcs[srcIndex][destIndex] = weight;
  }

  /*************************************************************
    Removes an edge between two vertices of the graph.
    If one or both vertices do not exist, ignores the removal.
    *************************************************************/
  public void removeEdge(T vertex1, T vertex2) {
    int index1 = getIndex(vertex1);
    int index2 = getIndex(vertex2);
    if (index1 != NOT_FOUND && index2 != NOT_FOUND) {
      removeArc(index1, index2);
      removeArc(index2, index1);
    }
  }

  /*************************************************************
    Removes an arc from vertex src to vertex dest,
    if the vertices exist, else does not change the graph.
    *************************************************************/
  public void removeArc(T srcVertex, T destVertex) {
    int src = getIndex(srcVertex);
    int dest = getIndex(destVertex);
    if (src != NOT_FOUND && dest != NOT_FOUND) {
      removeArc(src, dest);
    }
  }

  /*************************************************************
    Helper. Removes an arc from index v1 to index v2.
    @throws IllegalArgumentException if either index is invalid.
    *************************************************************/
  protected void removeArc(int srcIndex, int destIndex) {
    if (!indexIsValid(srcIndex) || !indexIsValid(destIndex)) {
      throw new IllegalArgumentException("One or more invalid indices: " + srcIndex
+ ", " + destIndex);
    }
    arcs[srcIndex][destIndex] = null;
  }


  /*************************************************************
    Returns the index value of the first occurrence of the vertex.
    Returns NOT_FOUND if the key is not found.
    *************************************************************/
  protected int getIndex(T vertex) {
    for (int i = 0; i < n; i++) {
      if (vertices[i].equals(vertex)) {
        return i;
      }
    }
    return NOT_FOUND;
  }

  /*************************************************************
    * Returns the weight of the edge from the one vertex to another.
    If no edge is present, returns -1.
    * @param: T vertex1, T vertex2 @return int
    *********************************************************/
  public int getWeight(T vertex1, T vertex2){
    int x = getIndex(vertex1);
    int y = getIndex(vertex2);
    if(x<0 || y<0)
      return -1;
    return arcs[x][y];

  }

  /*************************************************************
    Returns the vertex object that is at a certain index
    *************************************************************/
  protected T getVertex(int v) {
    if (!indexIsValid(v)) {
      throw new IllegalArgumentException("No such vertex index: " + v);
    }
    return vertices[v];
  }

  /*************************************************************
    Returns true if the given index is valid.
    *************************************************************/
  protected boolean indexIsValid(int index) {
    return index < n && index >= 0;
  }

  /*************************************************************
    Retrieve from a graph the vertices x pointing to vertex v (x->v)
    and returns them onto a linked list
    ***************************************************************/
  public LinkedList<T> getPredecessors(T vertex) {
    LinkedList<T> neighbors = new LinkedList<T>();

    int v = getIndex(vertex);

    if (v == NOT_FOUND) return neighbors;
    for (int i = 0; i < n; i++) {
      if (arcs[i][v] != null) {
        neighbors.add(getVertex(i)); // if T then add i to linked list
      }
    }
    return neighbors;
  }

  /**************************************************************
    * Retrieve from a graph the vertices x following vertex v (v->x)
    and returns them onto a linked list
    ***************************************************************/
  public LinkedList<T> getSuccessors(T vertex){
    LinkedList<T> neighbors = new LinkedList<T>();

    int v = getIndex(vertex);
    if (v == NOT_FOUND) return neighbors;
    for (int i = 0; i < n; i++) {
      if (arcs[v][i]!= null) {
        neighbors.add(getVertex(i)); // if T then add i to linked list
      }
    }
    return neighbors;
  }

  /**************************************************************
    Returns a string representation of the graph.
    ***************************************************************/
  public String toString() {
    if (n == 0) {
      return "Graph is empty";
    }

    String result = "";

    //result += "\nArcs\n";
    //result += "---------\n";
    result += "\ni ";

    for (int i = 0; i < n; i++) {
      result += "" + getVertex(i);
      if (i < 10) {
        result += " ";
      }
    }
    result += "\n";

    for (int i = 0; i < n; i++) {
      result += "" + getVertex(i) + " ";

      for (int j = 0; j < n; j++) {
        if (arcs[i][j]!=null) {
          result += arcs[i][j] + " ";
        } else {
          result += "- "; //just empty space
        }
      }
      result += "\n";
    }
    return result;
  }

  /**************************************************************
    * Saves the current graph into a .tgf file.
    * If it cannot save the file, a message is printed.
    **************************************************************/
  public void saveTGF(String tgf_file_name) {
    try {
      PrintWriter writer = new PrintWriter(new File(tgf_file_name));

      //prints vertices by iterating through array "vertices"
      for (int i = 0; i < n(); i++) {
        if (vertices[i] == null){
          break;
        } else {
          writer.print((i+1) + " " + vertices[i]);
          writer.println("");
        }
      }
      writer.print("#"); // Prepare to print the edges
      writer.println("");

      //prints arcs by iterating through 2D array
      for (int i = 0; i < n(); i++) {
        for (int j = 0; j < n(); j++) {
          if (arcs[i][j]!=null) {
```

```java
            writer.print((i+1) + " " + (j+1) + " " + arcs[i][j]);
            writer.println("");
          }
        }
      }
      writer.close();
    } catch (IOException ex) {
      System.out.println("***(T)ERROR*** The file could nt be written: " + ex);
    }
  }

  //looping to itself is prohibited.
  /** Testing Driver for AdjMatGraph.  This will not help you test AdjMatGraphPlus. */
  public static void main (String args[]) throws FileNotFoundException {
    System.out.println("NORMAL OPERATIONS");
    System.out.println("_________________");
    AdjMatGraph<String> G = new AdjMatGraph<String>();
    System.out.println("New graph is empty  (true): \t" + G);
    System.out.println("Empty=> undirected  (true): \t" + G.isUndirected());
    System.out.println("Empty graph no vertices(0): \t" + G.n());    System.out.println("Empty graph no arcs    (0): \t" + G.m());
    G.addVertex("A"); G.addVertex("B"); G.addVertex("C");
    G.addVertex("D"); G.addVertex("E"); G.addVertex("F");
    System.out.println("After adding 6 vert.   (6): \t " + G.n());
    System.out.println("After adding no arcs   (0): \t" + G.m());
    System.out.println("Still is undirected (true): \t" + G.isUndirected());

    G.addEdge("A", "B",2); G.addEdge("B", "C",1); G.addEdge("C", "D",3);
    G.addEdge("F", "A",2); G.addEdge("A", "D",5);
    System.out.println("After adding edges AB, BC, CD, AF, AD arcs");
    System.out.println("After adding 5  edges/a.k.a. 5 pairs of arcs = 10 arcs (10): \t" + G.m());
    System.out.println("Still is undirected (true): \t" + G.isUndirected());
    G.addEdge("A", "A",6); // adding a loop
    System.out.println("A->A loop=>directed(false): \t" + G.isUndirected());
    System.out.println(G);
    System.out.println(G.m());
    G.removeArc("C", "A"); // removing an arc that does not exist is okay
    G.removeEdge("A", "A"); // removing a loop
    System.out.println(G.m());
    System.out.println("removing the loop makes it undirected (true): \t" + G.isUndirected());

    G.addArc("A", "C",3); // adding an arc
    System.out.println("adding an arc makes it directed (=>false): \t" + G.isUndirected()); //-->
    System.out.println("Graph now has vertices   (6): \t " + G.n());
    System.out.println("Graph now has arcs      (11): \t" + G.m());
    System.out.println(G);
    System.out.println("Successors to  C (B,D): " + G.getSuccessors("C"));
    System.out.println("Predecess to C (A,B,D): " + G.getPredecessors("C"));



    G.removeArc("A", "C"); // removing an arc
    System.out.println("remov A-C => undirected (true): \t" + G.isUndirected());
    //System.out.println(G);
    System.out.println("FILE SAVED IN withA");
    G.saveTGF("withA.tgf");

    System.out.println("Predeces A (B, D, F) : \t" + G.getPredecessors("A"));
    System.out.println("Success  A (B, D, F) : \t" + G.getSuccessors("A"));

    G.removeVertex("A");
    System.out.println("A removed; graph has now: " + G.n() + " (5) vertices and " + G.m() + " (4) arcs");
    //System.out.println(G);
    System.out.println("Preceeding C: (B, D) " + G.getPredecessors("C"));
    //System.out.println(G);
    System.out.println("FILE SAVED IN withoutA");
    G.saveTGF("withoutA.tgf");

    System.out.println("removing some more vertices");
    G.removeVertex("E");  G.removeVertex("F");
    System.out.println(G);
    G.removeVertex("D");
    System.out.println("removing some more vertices");
    int m = G.m();
    System.out.println(G);
    G.addVertex("Z");
    System.out.println("adding vertex should not 'resurreect' any old edges  (m = " + m + ") [" + G.m() + "]");
    System.out.println(G);
    System.out.println("Returns the weight of the edge[B, C]-->Expected[1]:  "+ G.getWeight("B","C"));
    System.out.println("Returns the weight of the edge[B, D]-->Expected[-1]: "+ G.getWeight("B","D"));
//        AdjMatGraph<String> test1 = new AdjMatGraph<String>();
//        System.out.println(test1);
//        loadTGF("gameMap.tgf",test1);
//        System.out.println(test1);
  }

```

```java
}
```

Apple + Yum! You can eat me! And I can help you + I do not think eating me is going to help you.
Wrench + Wow! You can use me to fix something + Sorry I cannot fix this mistake.
Notebook + You can write in me! + Sorry you cannot write on me.
Boot + Step in me and wear me around + You tripped over me! I am no help
Duck + Rub a dub dub! + Quack do not squeeze me! Quack! I am not the one!
Scarf + Wear me round your neck + It is too warm outside for you to wear me!
Ring + Diamonds are your best friend! I will help you + If you liked me then you should have put me on
Snowman + Hi my name is Olaf and I like warm hugs! + It is too hot outside! I am melting! Bye!
Croissant + I am fancy and French. How can I help you? + Leave me alone to crisp
Backpack + I am friends with Dora the Explorer! Her backpack is my cousin, so whatcha need? + This backpack is not yours. Go away.
Laptop + I want you to look things up on me! + Sorry you do not know the password
Pepper Shaker + Salt and pepper here! We are here to help! + You dropped us and we broke
Ball + I will bounce you to victory! + This ball has been deflated
Paintbrush + I want to paint a Pollack and help you change the world + Sorry this paintbrush has been shipped to a museum
Pen + Click,Click,CLICK! I am ready to help you! + This pen ran out of ink, sorry!
Chair + Yay! Take me to the door + People sit on me all the time so no I cannot help you
Pizza + I am yummy and greasy and deliverable + This pizza has been out for weeks so it is no good
TV + I love watching reruns and helping strangers! + There is nothing on tv right now sorry
Watch + Tick Tock! My name is Clock! How can I help you? + This watch was stepped on. There is nothing but coils left
Fork + Use me to stab things! + People stab me into things all the time. I am done with all that violence.
Phone + LOL! LMFAO! OMG! Like I am so excited to help + This is a Nokia. This will not help at all
Socks + I will warm your toes! And help you with your woes + These socks have holes in them. Sorry, but they are useless to you.
Plant + Okay I will help you + This plant has withered and is not useful
Glasses + I will help you see the light + These glasses have broken to pieces and they are of no use
CD + It has been forever since someone has played me. I will help you!!! + The CD you picked up is a Backstreet Boys CD. That is useless to you. If only it was a NSYNC CD.

```
"" + Door is unlocked. Enter the room.
Looks like there is a troll blocking the door. Try feeding him
something and see what happens. Supposedly Trolls are vegans. + You
gave the troll the apple. It seems as if he is no longer hangry and
has moved out of your way.
You approach the door and it will not open. You notice the bolts are
loose. If only you could find something to tighten them. + You tighten
the door with the wrench. Looks like you can open it now.
At the door you find an elf and you try asking him to move. He is
distressed because he cannot find his notes for Elf University. Maybe
you can help him out. + He is so happy. Now he can noodle on back to
study for his finals!
As you approach the door you notice a puddle of water with an
electrical wire in the middle. Looks dangerous. Maybe you can find
something to stop you from dying? + You made it without electrocuting
yourself! Now, continue on through the door!
There is a child at the door who refuses to move and go take its bath.
How can you make baths fun? + The child ceased its screaming and went
distractedly to its mom. Now keep on moving!
There is a terrible windchill outside and your neck starts to freeze
in place. Hurry and take something that will help! + Congrats, you
managed to defrost your head holder and good thing too otherwise that
would be awkward. Now go ahead and continue through the door and leave
this climate.
At the door you find a sad lad. Turns out he lost the ring he was
going to propose with! You could have sworn you saw it somewhere. Help
him unite with his true love forever. + You give the man his ring. He
happily runs away to propose to his boyfriend.
FIRE!! The door is currently being set on flames! Maybe you can find
something to extinguish it? + You distract Olaf towards the fire and
push him in. He threatens you with his return next year. Guess you
have an enemy now.
You find an angry Frenchman drinking an espresso. He is missing
something. 'And it better be french!' he says. + You give him the
croissant and he thanks you for your understanding of the pastry
craft. He slides away smoothly.
NO WAY! Dora is on the set recording Dora The Explorer! She
says,'Quick! Say backpack' and you notice backpack is nowhere to be
found. Help her find him! + You slide backpack on the set. Whew! That
was a close one. You saved several children from disappointment!
Continue on with your quest, you hero!
A businessman from Wall Street is frantically looking for a device
that will help him connect with the stocks to the microsecond! He is
being stubborn and will not let you pass unless you 'SHOW HIM THE
MONEY!' + You 'SHOWED HIM THE MONEY' now go through the door!
Someone is trying to cook but they will not let you pass until you try
some. Oh no they are missing the 'final touch' condiment! Help them
look to speed up the process. + You had your steak and ate it too! Now
go through the door!
A dog wants to play with you. Get something that he can fetch, catch,
```

and bring back. + You played several rounds with him and now he needs to move on, as do you!
Picasso is in desperate need of a paintbrush! Hurry up before he loses his inspiration! + You just helped create the creative genius complete his artwork. Congrats!
A students worst nightmare! Hurry up and find her something to write with! + Congrats. You got her back in the running. Phew, what is with all these stressed out college students, right?
You find an old lady with a cane in desperate need of a rest. Find her something she an relax on for a bit before she gets back to walking. + You got a chair for her! Perfect, now she can sit down while you continue on!
There is nothing like hungry students studying for finals. Find a way to feed these zombies. They do not like healthy things. Remember that it is the middle of the night! + You have fed them and they are speeding their way through their notes. They do not even notice you anymore.
There is a teen crying about how his mom canceled Netflix and how now he does not know how he will be able to keep up with Criminal Minds. How will he ever keep up? + You have shown his a new way of life. He will forever be grateful!
Time is at the door. They appear to be looking for a way to tell themselves. (Get it? Tell time?) ANYWAYS, help them find a way for them to keep themselves on time. + You have given them a beautiful watch! Continue on through!
Ariel is looking for her brush! Go help her! + You gave Ariel her brush and watched in horror as she then proceeded to eat with it.
There is some ringing going on in the room. Someone is calling you! Find where the call is coming from and pick it up. + It was your mom telling you that you might need to pause the awesome game you are playing because it is almost time for dinner.
Your toes are getting cold, even with your snow boots on. Find something that will give them an extra fuzzy layer of warmth. + Good job! You prevented that frost bite (on most of your toes). Hop on through to the next room!
It makes oxygen and your friend wants you to water it over winter break. It needs sunlight and care. Kind of like a human except you might of happened to misplace it. Find it before she get back. + You found the plant and you made it promise not to tell. Now you can go on with your life!
Your roommate lost her contacts and therefore her ability to see. You need to guide her around all day until she can see again. Help her find a way to see! + You gave her the ability to see again! It’s a miracle. And a heavy prescription! Now you can go on through without worrying she will knock into anything!
A 90s kid is missing the NSYNC music that goes into their CD player. Help them find it! + You found it and also introduced them to Spotify so hopefully it will not happen again. Your mission with them is done now. Continue on!

The apple does not fall far from the tree
Third time is the charm
No news is good news
Less is more
Good things come to those who wait
Curiosity killed the cat
Better late than never
A picture is worth a thousand words
Revenge is a dish best served cold
Talk is cheap