Player:

· Quality points

· Learning

· craft

· integrity

· Name

· Image

· Room

· Cards

package game;

import java.awt.EventQueue;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Image;

import java.awt.List;

import java.awt.Point;

import java.awt.RenderingHints;

import java.applet.Applet;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JPanel;

import java.awt.BorderLayout;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.awt.geom.AffineTransform;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.awt.event.ActionEvent;

import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import javax.swing.JScrollPane;

import javax.swing.JTextField;

import javax.swing.ListSelectionModel;

import javax.swing.SwingUtilities;

import javax.swing.Timer;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.event.MouseInputAdapter;

import javax.swing.JScrollBar;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.\*;

public class WindowBuilder3 extends MouseInputAdapter

{

JFrame frame;

private JTextField statField;

private JTextField PlayerNameTextField;

private int pMoves;

JButton drawtCardButton, moveButton, playCardButton, cardButton;

//ArrayList<String> x = new ArrayList<String>();

static CardList gameDeck;

static RoomList rooms = new RoomList();

static Player currentPlayer ;

static JList<String> roomsList;

static Room nextRoom;

ArrayList <Player> pArray ;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args)

{

SwingUtilities.invokeLater(new Runnable()

{

public void run()

{

try

{

WindowBuilder3 window = new WindowBuilder3();

window.frame.setVisible(true);

}

catch (Exception e)

{

e.printStackTrace();

}

}

});

}// end of MAIN

//new

/\*\*

\* Create the application.

\* @throws IOException

\*/

public WindowBuilder3() throws IOException {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\* @throws IOException

\*/

// Testing - KENNY

private void initialize() throws IOException {//view

gameDeck = new CardList();

pArray = new ArrayList<Player>();

pMoves = 0;

pArray.add(new Player( "Player 1", rooms.getRoomList().get(0), initializePlayerDeck()));

pArray.add(new Player( "Ai 1", rooms.getRoomList().get(0), initializePlayerDeck()));

pArray.add(new Player( "Ai 2", rooms.getRoomList().get(0), initializePlayerDeck()));

currentPlayer = pArray.get(0); //Ruben

//print out the map

BufferedImage img = ImageIO.read(new File("map.png"));

final ImagePanel imgPane = new ImagePanel(img);

JScrollPane scrollPane = new JScrollPane(imgPane);

scrollPane.setBounds(0, 0, 1224, 474);

frame = new JFrame();

frame.setTitle("CECS BS Challenge");

frame.setBackground(Color.WHITE);

frame.getContentPane().setLayout(null);

frame.setSize(1230, 800);

frame.setResizable(false);

////DRAW CARD BUTTOn

drawtCardButton = new JButton("Draw Card");//add the draw card button

moveButton = new JButton("Move");//add the move button

playCardButton = new JButton("Play Card");

moveButton.setEnabled(false);

drawtCardButton.addActionListener(new ActionListener() {//listener

public void actionPerformed(ActionEvent e) {

PlayerNameTextField.setText("Draw card ");

drawtCardButton.setEnabled(false);

moveButton.setEnabled(true);

playCardButton.setEnabled(true);

cardButton.setEnabled(true);

scrollPane.repaint();

}

});

drawtCardButton.setBounds(29, 480, 143, 31);//set bound? what is this?

frame.getContentPane().add(drawtCardButton);

moveButton.setBounds(29, 515, 143, 31);

moveButton.addActionListener(new ActionListener() {//listener

public void actionPerformed(ActionEvent arg0)

{

int nextRoomIndex = roomsList.getSelectedIndex();

nextRoom = currentPlayer.getRoom().getNearByRoomArray().get(nextRoomIndex);

currentPlayer.setRoom(nextRoom);

setRoomsList();

roomsList.setSelectedIndex(0);

PlayerNameTextField.setText("Moved to room " + currentPlayer.getRoom().getRoomName());

scrollPane.repaint();

pMoves++;

if(pMoves == 3)

{

moveButton.setEnabled(false);

}

}

});

frame.getContentPane().add(moveButton);

playCardButton = new JButton("Play Card");

playCardButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

//PlayerNameTextField.setText("Play card " + currentPlayer.getCurrentCard());

playCardButton.setEnabled(false);

moveButton.setEnabled(false);

drawtCardButton.setEnabled(false);

cardButton.setEnabled(false);

// DO AI's TURNS via loop

doAITurn(pArray.get(1));

doAITurn(pArray.get(2));

drawtCardButton.setEnabled(true);

pMoves = 0;

scrollPane.repaint();

}

});

playCardButton.setEnabled(false);

playCardButton.setBounds(29, 550, 143, 31);

frame.getContentPane().add(playCardButton);

statField = new JTextField();

statField.setBounds(490, 490, 711, 171);

statField.setEditable(false);

statField.setBackground(Color.WHITE);

frame.getContentPane().add(statField);

statField.setColumns(10);

PlayerNameTextField = new JTextField();

PlayerNameTextField.setBounds(490, 674, 711, 65);

PlayerNameTextField.setEditable(false);

PlayerNameTextField.setBackground(Color.WHITE);

frame.getContentPane().add(PlayerNameTextField);

PlayerNameTextField.setColumns(10);

//create fly image

ImageIcon flyIcon = new ImageIcon(new ImageIcon("fly.png").getImage()

.getScaledInstance(230, 328, Image.SCALE\_DEFAULT));

JLabel FlyLabel = new JLabel();

FlyLabel.setIcon(flyIcon);

FlyLabel.setBounds(0, 0, 100, 25);

frame.getContentPane().add(FlyLabel);

////new

imgPane.addMouseListener(new MouseAdapter() {

public void mousePressed(MouseEvent me) {

int x = me.getX();

int y = me.getY();

FlyLabel.setLocation(x, y);

imgPane.add(FlyLabel);

System.out.println("You clicked at ("+x+","+y+")");

}

});

//new

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

frame.getContentPane().add(scrollPane);//add schrol panel

cardButton = new JButton("Current Card");

cardButton.setBounds(247, 490, 200, 270);

frame.getContentPane().add(cardButton);

cardButton.setIcon(new ImageIcon("cardm27.png"));

cardButton.setEnabled(false);

cardButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

cardButton.setIcon(new ImageIcon("cardm28.png"));

}

});

//initialize the JList with the the human player's current room

int numRooms = currentPlayer.getRoom().getNearByRoomArray().size();

String roomNames[] = new String[numRooms];

//the constructor takes an array, so populate the array from the arrayList

for(int i = 0; i < numRooms; i++)

{

roomNames[i] = currentPlayer.getRoom().getNearByRoomArray().get(i).getRoomName();

}

roomsList = new JList<String>(roomNames);

setRoomsList();

roomsList.setSelectedIndex(0);

roomsList.setSelectionMode(ListSelectionModel.SINGLE\_INTERVAL\_SELECTION);

roomsList.setLayoutOrientation(JList.VERTICAL);

roomsList.setVisibleRowCount(1);

roomsList.setBounds(29, 592, 143, 155);

roomsList.addListSelectionListener(new ListSelectionListener()

{

public void valueChanged(ListSelectionEvent arg0)

{

int nextRoomIndex = roomsList.getSelectedIndex();

//after moving, it resets to -1

if(nextRoomIndex == -1)

nextRoomIndex = 0;

nextRoom = currentPlayer.getRoom().getNearByRoomArray().get(nextRoomIndex);

}

});

frame.getContentPane().add(roomsList);

}

public static void setRoomsList()

{

int numRooms = currentPlayer.getRoom().getNearByRoomArray().size();

String roomNames[] = new String[numRooms];

//the constructor takes an array, so populate the array from the arrayList

for(int i = 0; i < numRooms; i++)

{

roomNames[i] = currentPlayer.getRoom().getNearByRoomArray().get(i).getRoomName();

}

roomsList.setListData(roomNames);

roomsList.setSelectedIndex(0);

}

public ArrayList<Card> initializePlayerDeck()

{

ArrayList<Card> playerDeck = new ArrayList<Card>();

if(gameDeck.getGameDeck().size() >= 5)

{

for(int i = 0; i < 5; i++)

{

playerDeck.add(gameDeck.getGameDeck().get(0));

gameDeck.getGameDeck().remove(0);

}

}

return playerDeck;

}

public void doAITurn(Player p){

Random rand = new Random();

for(int i = 0; i < 3; i++){

int size = p.getRoom().getNearByRoomArray().size();

int randomIndex = rand.nextInt(size);

p.setRoom(p.getRoom().getNearByRoomArray().get(randomIndex));

}

}

public class ImagePanel extends JPanel {//COPIED CODE. NEED REVISED

private BufferedImage image;

public ImagePanel(BufferedImage img) {

image = img;

}

protected Point getImageLocation() {

Point p = null;

if (image != null) {

int x = (getWidth() - image.getWidth()) / 2;

int y = (getHeight() - image.getHeight()) / 2;

p = new Point(x, y);

}

return p;

}

public Dimension getPreferredSize() {

return image == null ? super.getPreferredSize() : new Dimension(image.getWidth(), image.getHeight());

}

public Point toImageContext(Point p) {

Point imgLocation = getImageLocation();

Point relative = new Point(p);

relative.x -= imgLocation.x;

relative.y -= imgLocation.y;

return relative;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Point p = getImageLocation();

int x = p.x;

int y = p.y;

g.drawImage(image, x, y, this);

g.setColor(Color.RED);

g.setFont(new Font("Monospaced", Font.BOLD, 24));

g.drawString(pArray.get(0).getName(),pArray.get(0).getRoom().xCoord ,pArray.get(0).getRoom().yCoord);

g.drawString(pArray.get(1).getName(),pArray.get(1).getRoom().xCoord ,pArray.get(1).getRoom().yCoord + 20);

g.drawString(pArray.get(2).getName(),pArray.get(2).getRoom().xCoord ,pArray.get(2).getRoom().yCoord - 20);

}

}

}

RoomList

import java.util.ArrayList;

public class RoomList

{

private ArrayList<Room> rooms;

private Room georgeAllenField;

private Room japaneseGarden;

private Room bluePyramid;

private Room studentParking;

private Room recCenter;

private Room forbiddenParking;

private Room computerLab;

private Room ECS302;

private Room eatClub;

private Room cecsConf;

private Room northHall;

private Room southHall;

private Room retireRoom;

private Room elevators;

private Room ECS308;

private Room lactationLounge;

private Room eastW;

private Room westW;

private Room lib;

private Room la5;

private Room bratwurstHall;

public RoomList()

{

rooms = new ArrayList<Room>();

georgeAllenField = new Room("George Allen Field");

japaneseGarden = new Room("Japanese Garden");

bluePyramid = new Room("Blue Pyramid");

studentParking = new Room("Student Parking");

recCenter = new Room("Rec Center");

forbiddenParking = new Room("Forbidden Parking");

computerLab = new Room("Computer Lab");

ECS302 = new Room("ECS 302");

eatClub = new Room("Eat Club");

cecsConf = new Room("Cecs Conference Room");

northHall = new Room("North Hall");

southHall= new Room("South Hall");

retireRoom = new Room("Room of Retirement");

elevators = new Room("Elevators");

ECS308 = new Room("ECS 308");

lactationLounge= new Room("Lactation Lounge");

eastW = new Room("East Walkway");

westW = new Room("West Walkway");

lib = new Room("Library");

la5 = new Room("LA 5");

bratwurstHall = new Room("Bratwurst Hall");

setNearbyRooms();

rooms.add(ECS302);

rooms.add(georgeAllenField);

rooms.add(japaneseGarden);

rooms.add(bluePyramid);

rooms.add(studentParking);

rooms.add(recCenter);

rooms.add(forbiddenParking);

rooms.add(computerLab);

rooms.add(eatClub);

rooms.add(cecsConf);

rooms.add(northHall);

rooms.add(southHall);

rooms.add(retireRoom);

rooms.add(elevators);

rooms.add(ECS308);

rooms.add(lactationLounge);

rooms.add(eastW);

rooms.add(westW);

rooms.add(lib);

rooms.add(la5);

rooms.add(bratwurstHall);

}

public ArrayList<Room> getRoomList()

{

return rooms;

}

//ADD NEARBY ROOMS

public void setNearbyRooms()

{

georgeAllenField.setNearbyRoom(japaneseGarden);

georgeAllenField.setNearbyRoom(bluePyramid);

georgeAllenField.setNearbyRoom(recCenter);

georgeAllenField.setNearbyRoom(westW);

georgeAllenField.xCoord = 32;

georgeAllenField.yCoord = 168;

japaneseGarden.setNearbyRoom(georgeAllenField);

japaneseGarden.setNearbyRoom(bluePyramid);

japaneseGarden.setNearbyRoom(studentParking);

japaneseGarden.xCoord = 368;

japaneseGarden.yCoord = 50;

bluePyramid.setNearbyRoom(georgeAllenField);

bluePyramid.setNearbyRoom(japaneseGarden);

bluePyramid.setNearbyRoom(studentParking);

bluePyramid.setNearbyRoom(recCenter);

bluePyramid.xCoord = 427;

bluePyramid.yCoord = 247;

studentParking.setNearbyRoom(japaneseGarden);

studentParking.setNearbyRoom(bluePyramid);

studentParking.setNearbyRoom(recCenter);

studentParking.setNearbyRoom(forbiddenParking);

studentParking.xCoord = 892;

studentParking.yCoord = 168;

recCenter.setNearbyRoom(georgeAllenField);

recCenter.setNearbyRoom(bluePyramid);

recCenter.setNearbyRoom(studentParking);

recCenter.setNearbyRoom(forbiddenParking);

recCenter.setNearbyRoom(westW);

recCenter.xCoord = 470;

recCenter.yCoord = 478;

forbiddenParking.setNearbyRoom(studentParking);

forbiddenParking.setNearbyRoom(recCenter);

forbiddenParking.setNearbyRoom(eastW);

forbiddenParking.xCoord = 930;

forbiddenParking.yCoord = 459;

computerLab.setNearbyRoom(northHall);

computerLab.xCoord = 234;

computerLab.yCoord = 696;

ECS302.setNearbyRoom(northHall);

ECS302.setNearbyRoom(southHall);

ECS302.xCoord = 548;

ECS302.yCoord = 699;

eatClub.setNearbyRoom(southHall);

eatClub.xCoord = 746;

eatClub.yCoord = 690;

cecsConf.setNearbyRoom(southHall);

cecsConf.xCoord = 928;

cecsConf.yCoord = 710;

northHall.setNearbyRoom(computerLab);

northHall.setNearbyRoom(retireRoom);

northHall.setNearbyRoom(westW);

northHall.setNearbyRoom(ECS302);

northHall.setNearbyRoom(elevators);

northHall.setNearbyRoom(southHall);

northHall.xCoord = 267;

northHall.yCoord = 865;

southHall.setNearbyRoom(eatClub);

southHall.setNearbyRoom(ECS302);

southHall.setNearbyRoom(cecsConf);

southHall.setNearbyRoom(ECS308);

southHall.setNearbyRoom(lactationLounge);

southHall.setNearbyRoom(eastW);

southHall.setNearbyRoom(northHall);

southHall.xCoord = 779;

southHall.yCoord = 853;

retireRoom.setNearbyRoom(northHall);

retireRoom.xCoord = 228;

retireRoom.yCoord = 1033;

elevators.setNearbyRoom(northHall);

elevators.setNearbyRoom(la5);

elevators.xCoord = 461;

elevators.yCoord = 1044;

ECS308.setNearbyRoom(southHall);

ECS308.xCoord = 686;

ECS308.yCoord = 1067;

lactationLounge.setNearbyRoom(southHall);

lactationLounge.xCoord = 915;

lactationLounge.yCoord = 1034;

eastW.setNearbyRoom(forbiddenParking);

eastW.setNearbyRoom(southHall);

eastW.setNearbyRoom(bratwurstHall);

eastW.xCoord = 1091;

eastW.yCoord = 863;

westW.setNearbyRoom(georgeAllenField);

westW.setNearbyRoom(recCenter);

westW.setNearbyRoom(northHall);

westW.setNearbyRoom(lib);

westW.xCoord = 23;

westW.yCoord = 864;

lib.setNearbyRoom(westW);

lib.setNearbyRoom(la5);

lib.xCoord = 103;

lib.yCoord = 1317;

la5.setNearbyRoom(lib);

la5.setNearbyRoom(elevators);

la5.setNearbyRoom(bratwurstHall);

la5.xCoord = 472;

la5.yCoord = 1338;

bratwurstHall.setNearbyRoom(eastW);

bratwurstHall.setNearbyRoom(la5);

bratwurstHall.xCoord = 907;

bratwurstHall.yCoord = 1299;

}

public Room getRoom(String name)

{

Room desiredRoom = null;

for(Room r : rooms)

{

if(r.getRoomName() == name)

desiredRoom = r;

}

return desiredRoom;

}

}

CardList

import java.util.ArrayList;

public class CardList

{

private ArrayList<Card> gameDeck;

public CardList()

{

gameDeck = new ArrayList<Card>();

//To be initialized just like in RoomList

//String cardName, int integReq, int craftReq, int learnReq, String roomReq, int qualityReward, int qualityPen

Card c1 = new Card("Communication", 1, 1, 1, "Communicate room", 2, 1);

}

public ArrayList<Card> getGameDeck()

{

return gameDeck;

}

}

|  |  |  |
| --- | --- | --- |
| USE CASE | Taking a Turn | |
| Goal in Context | The user successfully plays a turn, consisting of: drawing a card, moving 0 to 3 spaces, playing a card, then discarding any excess card(s). | |
| Scope & Level | Personal Laptop, CECS BS Game Program | |
| Preconditions | The user has:  - a copy of CECS BS Challenge  - a functioning computer  - power source for computer | |
| Success End Condition | The user successfully drew a card, moved 0 to 3 spaces, played a card, then discarded any excess card(s). The game now lets the next player take a turn. | |
| Failed End Condition | The user did not successfully finish a turn due to inactivity or technical difficulties. | |
| Primary,  Secondary Actors | User – a player within the game  CECS BS game program | |
| Trigger | The user decides to play CECS BS Challenge game | |
| DESCRIPTION | Step | Action |
| Power on Computer | 1 | User turns on computer. |
| Loading | 2 | The user opens their copy of the game. |
| Draw Card | 3 | The game will start with the human player, the user must draw a card to begin their turn. |
| Move | 4 | The user has the option of not moving rooms, or moving up to 3 rooms, available rooms are displayed on screen. |
| Play Card | 5 | The user must choose a card to play, even if they do not meet any of the requirements for their cards. This is the final step of their turn; the game will now handle the AI turns. |
| EXTENSIONS | Step | Branching Action |
| No Game | 1 | Game not yet downloaded  User must connect to the internet and download a copy of the CECS BS Challenge. |
| Technical Difficulties | 2 | Game crashes during gameplay  User must close and restart the game. |

|  |  |  |
| --- | --- | --- |
| USE CASE | Playing a Card | |
| Goal in Context | The user plays a card that they currently hold in their current deck | |
| Scope & Level | Personal Laptop, CECS BS Game Program | |
| Preconditions | The user::  - has a copy of CECS BS Challenge  - has a functioning computer  - has a power source for computer  - is currently in a game  - is currently taking a turn  - has at least one card in their deck | |
| Success End Condition | The user has successfully chosen and played a card within their current deck | |
| Failed End Condition | The user did not successfully play a card due to inactivity or technical difficulties. | |
| Primary,  Secondary Actors | User – a player within the game  CECS BS game program | |
| Trigger | It is the user’s turn and they must now pick and play a card within their current deck | |
| DESCRIPTION | Step | Action |
| Cycling | 1 | It is now the player’s turn and the player must cycle through their current deck in search of a card they wish to play. |
| Choosing | 2 | For an effective turn, the player should ensure they meet the requirements for the card, but it is not required. Once they have chosen the card they wish to play, they must leave that card at the top of their deck. |
| Playing | 3 | With the chosen card remaining at the top of the deck, the player must click on the Play button. This plays the card and ends their turn. |
| EXTENSIONS | Step | Branching Action |
| No Card | 1 | No cards in the player’s deck:  The user will draw a card from the game deck and play it, regardless of requirements for that card. |
| Technical Difficulties | 2 | Game crashes during gameplay:  User must close and restart the game. |

In Word, make sure the table is 3in by 5in

I’ll do 30 - 40

|  |  |
| --- | --- |
| User Story | As a player, I want to play the “Car Pool” Card so that I can earn 3 Quality Points and 1 Game Card. |
| Acceptance Criteria | - I am inside of Parking Lot  - I have at least 5 Integrity Points  - I lose 2 Quality Points if I play the card without meeting the requirements |

////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////

|  |  |
| --- | --- |
| package game; |  |
|  |  |
|  | import java.util.\*; |
|  | import java.util.Random; |
|  |  |
|  | public class Player { |
|  | int quality, learn, craft, integrity; |
|  | String name; |
|  | Room currRoom; |
|  | ArrayList<Cards> hand;  Card currentCard; //Ruben |
|  |  |
|  | public Player(String n, Room r, ArrayList<Cards> h){ |
|  | n = name; |
|  | currRoom = r; |
|  | hand = h; |
|  | quality = 0; |
|  | startSkill(); |
|  |  |
|  | } |
|  |  |
|  | public void startSkill(){ // randomly generate initial stats |
|  | Random rand = new Random(); |
|  | learn = rand.nextInt(7); |
|  | craft = rand.nextInt(7 - learn); |
|  | integrity = 6 - (learn + craft); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | public void addQuality(int q){ |
|  | quality += q; |
|  | } |
|  |  |
|  | public void addLearn(int l){ |
|  | learn += l; |
|  | } |
|  |  |
|  | public void addCraft(int c){ |
|  | craft += c; |
|  | } |
|  |  |
|  | public void addIntegrity(int i){ |
|  | integrity += i; |
|  | } |
|  |  |
|  | public void addCard(Cards c){ |
|  | hand.add(c); |
|  | } |
|  |  |
|  | public int getQuality(){ |
|  | return quality; |
|  | } |
|  |  |
|  | public int getLearn(){ |
|  | return learn; |
|  | } |
|  |  |
|  | public int getCraft(){ |
|  | return craft; |
|  | } |
|  | public int getIntegrity(){ |
|  | return integrity; |
|  | } |
|  |  |
|  | public String getName(){ |
|  | return name; |
|  | } |
|  | public Room getRoom(){ |
|  | return currRoom; |
|  | } |
|  | public ArrayList<Cards> getHand(){ |
|  | return hand; |
|  | } |
|  |  |
|  |  |
|  | public void setName(String n){ |
|  | name = n; |
|  | } |
|  | public void setRoom(Room r){ |
|  | currRoom = r; |
|  | } |
|  |  |
|  | public void setHand(ArrayList<Cards> h){ |
|  | hand = h; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | public void print(){ |
|  | System.out.println(learn +" "+ craft +" "+ integrity); |
|  | }  } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Room:

· adjacent rooms

|  |  |
| --- | --- |
| import java.util.\*; |  |
|  | public class Room { |
|  | String roomName; |
|  | ArrayList<Room> nearbyRooms; //Ruben  public int xCoord;  public int yCoord;  public Room(String name)//Ruben  {  roomName = name;  nearbyRooms = new ArrayList<Room>();  } |
|  |  |
|  | public Room(String name)  {  roomName = name;  nearbyRooms = new ArrayList<Room>();  } |
|  |  |
|  | public void setRoomName(String name){ |
|  | this.roomName = name; |
|  | } |
|  | public String getRoomName(){ |
|  | return roomName; |
|  | } |
|  |  |
|  | public void setNearbyRoom(String name){ |
|  | nearbyRooms.add(name); |
|  | } |
|  | public ArrayList<Room> getNearByRoomArray() //Ruben  {  return nearbyRooms;  } |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |

Cards:

* Variables
  + name
  + player
  + learningReq
  + craftReq
  + integrityReq
  + skillReward
  + qualityReward
  + qualityPenalty

import javax.swing.ImageIcon;

import javax.swing.JOptionPane;

public abstract class Card

{

public String cardName;

//requirements to play the card, will be compared with the player values

public int integrityReq;

public int craftReq;

public int learningReq;

public String roomNameReq;

public String imageFileName;

//qualityPoints rewarded for playing the card successfully

public int qualityReward;

//quality point penalty for not meeting the requirements and trying to play the card

public int qualityPenalty;

//to be chosen by the player after playing the card successfully

public String skillChipReward;

//when this card is assigned to a player, this Player variable will be set to that player

public Player playerHolder;

public Card()

{

cardName = "";

integrityReq = 0;

craftReq = 0;

learningReq = 0;

roomNameReq = "";

qualityReward = 0;

qualityPenalty = 0;

skillChipReward = "";

playerHolder = null;

}

public Card(String cardName, int integReq, int craftReq, int learnReq, String roomReq, int qualityReward, int qualityPen)

{

this.cardName = cardName;

integrityReq = integReq;

this.craftReq = craftReq;

learningReq = learnReq;

roomNameReq = roomReq;

this.qualityReward = qualityReward;

qualityPenalty = qualityPen;

}

public abstract void playCard();

public void discard()

{

playerHolder = null;

}

public Object getSkillChipReward()

{

Object[] skillChips = { "Integrity", "Craft", "Learning" };

Object selectedValue = JOptionPane.showInputDialog(null,

"Choose your skill chip reward", "Input",

JOptionPane.INFORMATION\_MESSAGE, null,

skillChips, skillChips[0]);

return selectedValue;

}

}

Controller:

* card deck
* Scoreboard
* Map
* Quality point sum (to move on to sophomore)
* Player1
* Player2
* Player3

import java.awt.EventQueue;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Image;

import java.awt.List;

import java.awt.Point;

import java.awt.RenderingHints;

import java.applet.Applet;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JPanel;

import java.awt.BorderLayout;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.awt.geom.AffineTransform;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.awt.event.ActionEvent;

import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import javax.swing.ListSelectionModel;

import javax.swing.SwingUtilities;

import javax.swing.Timer;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.event.MouseInputAdapter;

import javax.swing.JScrollBar;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.\*;

public class WindowBuilder3 extends MouseInputAdapter {

JFrame frame;

static JTextArea statField;

static JTextArea PlayerNameTextField;

static int pMoves;

JButton drawtCardButton, moveButton, playCardButton, cardButton;

// ArrayList<String> x = new ArrayList<String>();

static CardList gameDeck;

static RoomList rooms = new RoomList();

static Player currentPlayer;

static JList<String> roomsList;

static Room nextRoom;

ArrayList<Player> pArray;

/\*\*

\* Launch the application.

\*/

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

try {

WindowBuilder3 window = new WindowBuilder3();

window.frame.setVisible(true);

}

catch (Exception e) {

e.printStackTrace();

}

}

});

}// end of MAIN

// new

/\*\*

\* Create the application.

\*

\* @throws IOException

\*/

public WindowBuilder3() throws IOException {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*

\* @throws IOException

\*/

// Testing - KENNY

private void initialize() throws IOException {// view

pMoves=0;

gameDeck = new CardList();

pArray = new ArrayList<Player>();

pArray.add(new Player("Player 1", rooms.getRoomList().get(0), initializePlayerDeck()));

pArray.add(new Player("Ai 1", rooms.getRoomList().get(0), initializePlayerDeck()));

pArray.add(new Player("Ai 2", rooms.getRoomList().get(0), initializePlayerDeck()));

currentPlayer = pArray.get(0); // Ruben

// print out the map

BufferedImage img = ImageIO.read(new File("map.png"));

final ImagePanel imgPane = new ImagePanel(img);

JScrollPane scrollPane = new JScrollPane(imgPane);

scrollPane.setBounds(0, 0, 1224, 474);

frame = new JFrame();

frame.setTitle("CECS BS Challenge");

frame.setBackground(Color.WHITE);

frame.getContentPane().setLayout(null);

frame.setSize(1230, 800);

frame.setResizable(false);

//// DRAW CARD BUTTOn

drawtCardButton = new JButton("Draw Card");// add the draw card button

moveButton = new JButton("Move");// add the move button

playCardButton = new JButton("Play Card");

moveButton.setEnabled(false);

drawtCardButton.addActionListener(new ActionListener() {// listener

public void actionPerformed(ActionEvent e) {

drawtCardButton.setEnabled(false);

moveButton.setEnabled(true);

playCardButton.setEnabled(true);

cardButton.setEnabled(true);

scrollPane.repaint();

}

});

drawtCardButton.setBounds(29, 480, 143, 31);// set bound? what is this?

frame.getContentPane().add(drawtCardButton);

moveButton.setBounds(29, 515, 143, 31);

moveButton.addActionListener(new ActionListener() {// listener

public void actionPerformed(ActionEvent arg0) {

int nextRoomIndex = roomsList.getSelectedIndex();

currentPlayer= pArray.get(pMoves);

nextRoom = currentPlayer.getRoom().getNearByRoomArray().get(nextRoomIndex);

currentPlayer.setRoom(nextRoom);

setRoomsList();

roomsList.setSelectedIndex(0);

scrollPane.repaint();

printPlayerField();

// pMoves++;

//

// if (pMoves == 3) {

// moveButton.setEnabled(false);

// pMoves=0;

// }

}

});

frame.getContentPane().add(moveButton);

playCardButton = new JButton("Play Card");

playCardButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

// PlayerNameTextField.setText("Play card " +

// currentPlayer.getCurrentCard());

playCardButton.setEnabled(false);

moveButton.setEnabled(false);

drawtCardButton.setEnabled(false);

cardButton.setEnabled(false);

// DO AI's TURNS via loop

doAITurn(pArray.get(1));

doAITurn(pArray.get(2));

drawtCardButton.setEnabled(true);

scrollPane.repaint();

/////////////////////////////////////////////////

currentPlayer= pArray.get(pMoves);

printStatField();

//////////////////////////////////////////////////////////

}

});

playCardButton.setEnabled(false);

playCardButton.setBounds(29, 550, 143, 31);

frame.getContentPane().add(playCardButton);

statField = new JTextArea();

statField.setBounds(490, 490, 711, 171);

statField.setEditable(false);

statField.setBackground(Color.WHITE);

frame.getContentPane().add(statField);

statField.setColumns(10);

statField.setFont( new Font("monospaced", Font.PLAIN, 15));

printStatField();

PlayerNameTextField = new JTextArea();

PlayerNameTextField.setBounds(490, 674, 711, 65);

PlayerNameTextField.setEditable(false);

PlayerNameTextField.setBackground(Color.WHITE);

frame.getContentPane().add(PlayerNameTextField);

PlayerNameTextField.setColumns(10);

PlayerNameTextField.setFont( new Font("monospaced", Font.PLAIN, 15));

// create fly image

ImageIcon flyIcon = new ImageIcon(

new ImageIcon("fly.png").getImage().getScaledInstance(230, 328, Image.SCALE\_DEFAULT));

JLabel FlyLabel = new JLabel();

FlyLabel.setIcon(flyIcon);

FlyLabel.setBounds(0, 0, 100, 25);

frame.getContentPane().add(FlyLabel);

//// new

imgPane.addMouseListener(new MouseAdapter() {

public void mousePressed(MouseEvent me) {

int x = me.getX();

int y = me.getY();

FlyLabel.setLocation(x, y);

imgPane.add(FlyLabel);

System.out.println("You clicked at (" + x + "," + y + ")");

}

});

// new

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

frame.getContentPane().add(scrollPane);// add schrol panel

cardButton = new JButton("Current Card");

cardButton.setBounds(247, 490, 200, 270);

frame.getContentPane().add(cardButton);

cardButton.setIcon(new ImageIcon("cardm27.png"));

cardButton.setEnabled(false);

cardButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent arg0) {

cardButton.setIcon(new ImageIcon("cardm28.png"));

}

});

// initialize the JList with the the human player's current room

int numRooms = currentPlayer.getRoom().getNearByRoomArray().size();

String roomNames[] = new String[numRooms];

// the constructor takes an array, so populate the array from the

// arrayList

for (int i = 0; i < numRooms; i++) {

roomNames[i] = currentPlayer.getRoom().getNearByRoomArray().get(i).getRoomName();

}

roomsList = new JList<String>(roomNames);

setRoomsList();

roomsList.setSelectedIndex(0);

roomsList.setSelectionMode(ListSelectionModel.SINGLE\_INTERVAL\_SELECTION);

roomsList.setLayoutOrientation(JList.VERTICAL);

roomsList.setVisibleRowCount(1);

roomsList.setBounds(29, 592, 143, 155);

roomsList.addListSelectionListener(new ListSelectionListener() {

public void valueChanged(ListSelectionEvent arg0) {

int nextRoomIndex = roomsList.getSelectedIndex();

// after moving, it resets to -1

if (nextRoomIndex == -1)

nextRoomIndex = 0;

nextRoom = currentPlayer.getRoom().getNearByRoomArray().get(nextRoomIndex);

}

});

frame.getContentPane().add(roomsList);

}

public static void setRoomsList() {

int numRooms = currentPlayer.getRoom().getNearByRoomArray().size();

String roomNames[] = new String[numRooms];

// the constructor takes an array, so populate the array from the

// arrayList

for (int i = 0; i < numRooms; i++) {

roomNames[i] = currentPlayer.getRoom().getNearByRoomArray().get(i).getRoomName();

}

roomsList.setListData(roomNames);

roomsList.setSelectedIndex(0);

}

public ArrayList<Card> initializePlayerDeck() {

ArrayList<Card> playerDeck = new ArrayList<Card>();

if (gameDeck.getGameDeck().size() >= 5) {

for (int i = 0; i < 5; i++) {

playerDeck.add(gameDeck.getGameDeck().get(0));

gameDeck.getGameDeck().remove(0);

}

}

return playerDeck;

}

public void doAITurn(Player p) {

Random rand = new Random();

for (int i = 0; i < 3; i++) {

int size = p.getRoom().getNearByRoomArray().size();

int randomIndex = rand.nextInt(size);

p.setRoom(p.getRoom().getNearByRoomArray().get(randomIndex));

}

}

public void printStatField(){

statField.setText("");

statField.append(String.format("%-15s%-15s%-15s%-15s%n"

,"Learning" , "Craft", "Integrity", "Quality Point"));

statField.append(String.format("%-15s%-15s%-15s%-15s%-15s%n"

,pArray.get(0).getName()

, pArray.get(0).getLearn()

, pArray.get(0).getCraft()

,pArray.get(0).getIntegrity()

,pArray.get(0).getQuality()));

statField.append(String.format("%-15s%-15s%-15s%-15s%-15s%n"

,pArray.get(1).getName()

,pArray.get(1).getLearn()

,pArray.get(1).getCraft()

,pArray.get(1).getIntegrity()

,pArray.get(1).getQuality()));

statField.append(String.format("%-15s%-15s%-15s%-15s%-15s%n"

,pArray.get(2).getName()

,pArray.get(2).getLearn()

,pArray.get(2).getCraft()

,pArray.get(2).getIntegrity()

,pArray.get(2).getQuality()));

statField.append("Cards in deck: \n");

statField.append("Discards out of play: \n");

statField.append("You are " +currentPlayer.getName() + " and you are in " +currentPlayer.getRoom().roomName);

}

public void printPlayerField(){

PlayerNameTextField.setText(currentPlayer.getName()+" Moved to room " + currentPlayer.getRoom().getRoomName());

printStatField();

}

public class ImagePanel extends JPanel {// COPIED CODE. NEED REVISED

private BufferedImage image;

public ImagePanel(BufferedImage img) {

image = img;

}

protected Point getImageLocation() {

Point p = null;

if (image != null) {

int x = (getWidth() - image.getWidth()) / 2;

int y = (getHeight() - image.getHeight()) / 2;

p = new Point(x, y);

}

return p;

}

public Dimension getPreferredSize() {

return image == null ? super.getPreferredSize() : new Dimension(image.getWidth(), image.getHeight());

}

public Point toImageContext(Point p) {

Point imgLocation = getImageLocation();

Point relative = new Point(p);

relative.x -= imgLocation.x;

relative.y -= imgLocation.y;

return relative;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Point p = getImageLocation();

int x = p.x;

int y = p.y;

g.drawImage(image, x, y, this);

g.setColor(Color.RED);

g.setFont(new Font("Monospaced", Font.BOLD, 24));

g.drawString(pArray.get(0).getName(), pArray.get(0).getRoom().xCoord, pArray.get(0).getRoom().yCoord);

g.drawString(pArray.get(1).getName(), pArray.get(1).getRoom().xCoord, pArray.get(1).getRoom().yCoord + 20);

g.drawString(pArray.get(2).getName(), pArray.get(2).getRoom().xCoord, pArray.get(2).getRoom().yCoord - 20);

}

}

}

Learning Craft Integrity Quality Points

Matt 2 2 2 0

Amanda 3 0 3 0

Habib 2 1 3 0

Cards in deck: 35 Discards out of play: 0

You are Matt and you are in ECS 30

**Khai Phung: All of my previous codes are on** [**GITHUB**](https://github.com/RubenSV117/CECS_343_Game/commits/master)