/\*

MusicPlayer

Class to play music

@author Ken Tjie

@create 2016 - 07 - 01

@update 2016- 08 - 01

@version 1.1

\*/

import java.applet.Applet;

import java.net.URL;

import java.net.MalformedURLException;

import java.applet.AudioClip;

public class MusicPlayer {

/\*

MusicPlayer

@Post-condition: instantiates an instance of the MusicPlayer class

\*/

public MusicPlayer () {

AudioClip[] sounds = new AudioClip[2];

int randomNum = (int)(Math.random()\*sounds.length);

try {

sounds[0] = Applet.newAudioClip(new URL("file:music/ORASGymLeader.wav"));

sounds[1] = Applet.newAudioClip(new URL("file:music/ORASFrontierBrain.wav"));

}

catch (MalformedURLException e) {}//end try catch

sounds[randomNum].loop();

}//end constructor

}//end class

/\*

Score

Class represents a score

@author Ken Tjie

@created 2016 - 12 - 01

@updated 2016 - 14 - 2015

@version 1.1

\*/

public class Score {

private String name;

private int points, levelsBeat;

/\*

Score

@param name: name of the player for the score

@param points: number of points

@param levelsBeat: number of levels beaten

@Precondition: name must not be null

@Post-condition: Instantiates an instance of the Score class

\*/

public Score (String name, int points, int levelsBeat) {

this.name = name;

this.points = points;

this.levelsBeat = levelsBeat;

}//end constructor

/\*

getName

@Post-condition: gets the name in the score

@return name

\*/

public String getName() {

return name;

}//end getName

/\*

getPoints

@Post-condition: gets the points in the score

@return points

\*/

public int getPoints() {

return points;

}//end getPoints

/\*

getLevels

@Post-condition: gets the number of levels beaten in the score

@return levelsBeat

\*/

public int getLevels() {

return levelsBeat;

}//end getLevels

/\*

toString

@Post-conditon: return a string with the data of the score

@return string of data for the score

\*/

public String toString () {

return "Name: " + name + " Points: " + points + " Levels Beat: " + levelsBeat + "\n\n";

}//end toString

}//end class

/\*

Item

Class represents collectable items

@author Ken Tjie

@create 2015 - 17 - 01

@update 2015 - 17 - 01

@version 1.4

\*/

import javax.swing.ImageIcon;

import java.awt.Graphics;

import java.awt.Graphics2D;

public class Item {

int x, y, maxX, maxY;

String name;

ImageIcon sprite;

/\*

Item

@param name: name of the item

@Pre-condition: name must be Leppa, Sitrus or Coin

@Post-condition: instantiates an instance of the Item class

\*/

public Item (String name) {

this.name = name;

if (name.equalsIgnoreCase("Leppa"))

sprite = new ImageIcon("sprites/leppa.png");

else if (name.equalsIgnoreCase("Sitrus"))

sprite = new ImageIcon("sprites/sitrus.png");

else if (name.equalsIgnoreCase("Coin"))

sprite = new ImageIcon("sprites/coin.png");

x = (int)(Math.random()\*300) + 10;

y = (int)(Math.random()\*591) + 10;

maxX = x + sprite.getIconWidth();

maxY = y + sprite.getIconHeight();

}//end constructor

/\*

getName

@Post-condition: returns name of item

@return name

\*/

public String getName() {

return name;

}//end getName

/\*

moveoffScreen

@Post-condition: moves item out of boundries

\*/

public void moveOffScreen () {

this.x = -50;

this.y = -10 - sprite.getIconHeight();

maxX = x + sprite.getIconWidth();

maxY = y + sprite.getIconHeight();

}//end moveOffScreen

/\*

isColliding

@param collidee: character the it may collide

@Post-condition: checks if collidee is colliding with item

@return true if character collides

@return false if character doesn't collide

\*/

public boolean isColliding (Character collidee) {

if (collidee.getMaxX() < x)

return false;

if (collidee.getX() > maxX)

return false;

if (collidee.getY() > maxY)

return false;

if (collidee.getMaxY() < y)

return false;

return true;

}//end isColliding

/\*

spawn

@param gc: place to draw

@Post-condition: paints the sprite on the screen

\*/

public void spawn (Graphics gc) {

sprite.paintIcon(null, gc, x, y);

}//end spawn

}//end class

/\*

Character

Class represents a character

@author Ken Tjie

@create 2015 - 06 - 01

@update 2015 - 18 - 01

@version 1.6

\*/

import java.awt.Graphics;

import javax.swing.ImageIcon;

public abstract class Character {

protected String name;

protected int x, y, maxX, maxY, health, attack, defense;

protected double currentHealth;

protected ImageIcon sprite;

/\*

Character

@param name: name for the character

@param hp: health for the character

@param attack: base attack stat of the character

@param defense: base defense stat for the character

@Precondition: name must not be null

@Post-condition: instanitates an instance of the Character class

\*/

public Character (String name, int hp, int attack, int defense) {

this.name = name;

health = 204 + 2 \* hp;

this.attack = (int)(1.10 \* (2 \* attack + 5));

this.defense = (int)(1.10 \* (2 \* defense + 5));

currentHealth = (double)health;

}//end constructor

/\*

setName

@param name: name for the character

@Precondition: name must not be null

@Post-condition: sets the name of the character

\*/

public void setName (String name) {

this.name = name;

}//end setName

/\*

getName

@Post-condition: returns name of character

@return name

\*/

public String getName () {

return name;

}//end getName

/\*

setX

@param x: x-coordinate of character

@Post-condition: sets the x value of the character

\*/

public void setX (int x) {

this.x = x;

}//setX

/\*

getX

@Post-condition: returns the x value

@return x

\*/

public int getX () {

return x;

}//end getX

/\*

setY

@param y: y-coordinate of character

@Post-condition: sets the y value of the character

\*/

public void setY (int y) {

this.y = y;

}//end setY

/\*

getY

@Post-condition: returns the y value

@return y

\*/

public int getY () {

return y;

}//end getY

/\*

setMaxX

@param x: x-coordinate of character

@Post-condition: sets the maxX value of the character

\*/

public void setMaxX (int x) {

maxX = x;

}//setMaxX

/\*

getMaxX

@Post-condition: returns the maxX vlaue

@return maxX

\*/

public int getMaxX () {

return maxX;

}//end getMaxX

/\*

setMaxY

@param y: y-coordinate of character

@Post-condition: sets the maxY value of the character

\*/

public void setMaxY (int y) {

maxY = y;

}//end getMaxY

/\*

getMaxY

@Post-condition: returns the maxY value

@return maxY

\*/

public int getMaxY () {

return maxY;

}//end getMaxY

/\*

getMaxHealth

@Post-condition: returns health stat of character

@return health

\*/

public int getMaxHealth () {

return health;

}//end getMaxHealth

/\*

getHealth

@Post-condition: returns character's health

@return currentHealth

\*/

public double getHealth () {

return currentHealth;

}//end getHealth

/\*

setHealth

@param hp: new value of currenthealth

@Post-condition: sets the currentHealth of the character

\*/

public void setHealth(double hp) {

currentHealth = hp;

if (currentHealth > health)

currentHealth = (double)health;

}//end setHealth

/\*

getAttack

@Post-condition: returns attack

@return attack

\*/

public int getAttack() {

return attack;

}//end getAttack

/\*

getDefense

@Post-condition: returns defense

@return defense

\*/

public int getDefense() {

return defense;

}//end getDefense

/\*

toString

@Post-condition: prints out the stats of the character

@return stats of the character

\*/

public String toString () {

return name + "\nHP: " + health + "\nAttack: " + attack + "\nDefense: " + defense + "\n";

}//end toString

/\*

draw

@param gc: place to draw

@Post-condition: draws the sprites on the screen

\*/

public void draw(Graphics gc) {

sprite.paintIcon(null, gc, x, y);

}//end draw

public abstract void updateHP (Graphics gc);

}//end class

/\*

Player

Class prepresents player's character

@author Ken Tjie

@create 2016 - 06 - 01

@update 2016 - 18 - 01

@version 1.8

\*/

import javax.swing.ImageIcon;

import java.awt.Graphics;

import java.awt.Color;

public class Player extends Character {

/\*

Player

@Post-condition: instantiates an instance of the Player class

\*/

public Player () {

super("Pikachu", 35, 55, 50);

sprite = new ImageIcon("sprites/Pikachu.png");

x = 0;

y = 0;

maxX = x + sprite.getIconWidth();

maxY = y + sprite.getIconHeight();

}//end constructor

/\*

Player

@param name: name of player

@Precondition: name must not be null

@Post-condition: instantiates an instance of the Player class

\*/

public Player (String name) {

super(name, 100, 60, 55);

this.name = name;

sprite = new ImageIcon("sprites/Pikachu.png");

x = 0;

y = 0;

maxX = x + sprite.getIconWidth();

maxY = y + sprite.getIconHeight();

}//end constructor

/\*

move

@param velX: velocity of the x-coordinates

@param velY: velocity of the y-coordinates

@Post-condition: moves sprite of the player

\*/

public void move (int velX, int velY) {

if (x < 0) {

x = 0;

maxX = x + sprite.getIconWidth();

velX = 0;

}//end if

if (maxX > 400) {

x = 400 - sprite.getIconWidth();

maxX = 400;

velX = 0;

}//end if

if (y < 0) {

y = 0;

maxY = y + sprite.getIconHeight();

velX = 0;

}//end if

if (maxY > 650) {

y = 650 - sprite.getIconHeight();

maxY = 650;

velX = 0;

}//end if

x += velX;

maxX += velX;

y += velY;

maxY += velY;

}//end move

/\*

updateHP

@param gc: place to draw

@Post-condition: draws the health bar accodring to health of the player

\*/

public void updateHP (Graphics gc) {

gc.setColor(Color.GREEN);

if (currentHealth/health <= 0.5)

gc.setColor(Color.YELLOW);

if (currentHealth/health <= 0.125)

gc.setColor(Color.RED);

gc.drawString(name + " HP: " + Math.round(currentHealth) + " / " + (int) health, 5, 620);

gc.fillRect(5, 630, (int)((currentHealth/health) \* 500), 20);

}//end updateHP

}//end class

/\*

Boss

Class represents character of a boss

@author Ken Tjie

@create 2016 - 08 - 01

@update 2016 - 18 - 01

@version 1.7

\*/

import javax.swing.ImageIcon;

import java.awt.Graphics;

import java.awt.Color;

public class Boss extends Character {

private String type;

/\*

Boss

@param name: name for the boss

@param type: type for the boss

@param hp: health for the boss

@param attack: base attack stat of the boss

@param defense: base defense stat for the boss

@Precondition: name must not be null

@Post-condition: instanitates an instance of the Boss class

\*/

public Boss (String name, String type, int hp, int attack, int defense) {

super(name, hp, attack, defense);

this.type = type;

sprite = new ImageIcon("sprites/" + name + ".png");

x = 1300 - sprite.getIconWidth();

y = 0;

maxX = x + sprite.getIconWidth();

maxY = y + sprite.getIconHeight();

}//end constructor

/\*

getType

@Post-condition: returns the type of the boss

@return type

\*/

public String getType () {

return type;

}//end getType

/\*

move

@param distance: distance to move in pixels

@Post-condition: moves the sprite of the Boss class

\*/

public void move (int distance) {

y += distance;

maxY += distance;

}//end move

/\*

updateHP

@param gc: place to draw

@Post-condition: draws rectangle accodring to health of the boss

\*/

public void updateHP (Graphics gc) {

gc.setColor(Color.GREEN);

if ((currentHealth/health) \* 100 <= 50)

gc.setColor(Color.YELLOW);

if ((currentHealth/health) \* 100 <= 12)

gc.setColor(Color.RED);

gc.drawString(name + " HP: " + Math.round(currentHealth) + " / " + (int) health, 798, 620);

gc.fillRect((int)(1300 - 2 - (currentHealth/health) \* 500), 630, (int)((currentHealth/health) \* 500), 20);

}//end updateHP

}//end class

/\*

Projectile

Class represents a projectile

@author Ken Tjie

@created 2016 - 06 - 01

@update 2016 - 18 - 2015

@version 1.4

\*/

import java.awt.Graphics;

import java.awt.Graphics2D;

public abstract class Projectile {

protected int x, y, maxX, maxY, radius;

/\*

Projectile

@param radius: radius of the projectile

@Post-condition: instantiates an instance of the Projectile class

\*/

public Projectile (int radius) {

this.radius = radius;

}//end radius

/\*

setX

@param x: x-coordinate of character

@Post-condition: sets the x value of the character

\*/

public void setX (int x) {

this.x = x;

maxX = this.x + 2 \* radius;

}//setX

/\*

getX

@Post-condition: returns the x value

@return x

\*/

public int getX () {

return x;

}//end getX

/\*

setY

@param y: y-coordinate of character

@Post-condition: sets the y value of the character

\*/

public void setY (int y) {

this.y = y;

}//end setY

/\*

getY

@Post-condition: returns the y value

@return y

\*/

public int getY () {

return y;

}//end getY

/\*

setMaxX

@param x: x-coordinate of character

@Post-condition: sets the maxX value of the character

\*/

public void setMaxX (int x) {

maxX = x;

}//setMaxX

/\*

getMaxX

@Post-condition: returns maxX value

@return maxX

\*/

public int getMaxX () {

return maxX;

}//end getMaxX

/\*

setMaxY

@param y: y-coordinate of projectile

@Post-condition: sets the maxY value of the projectile

\*/

public void setMaxY (int y) {

maxY = y;

}//end setMaxY

/\*

getMaxY

@Post-condition: returns the maxY value

@return maxY

\*/

public int getMaxY () {

return maxY;

}//end getMaxY

/\*

isColliding

@Post-condition: checks if collidee is colliding with projectile

@return true if character collides

@return false if character doesn't collide

\*/

public boolean isColliding(Character collidee) {

if (collidee.getMaxX() < x)

return false;

if (collidee.getX() > maxX)

return false;

if (collidee.getY() > maxY)

return false;

if (collidee.getMaxY() < y)

return false;

return true;

}//end isColliding

public abstract void move(int level, Character target);

public abstract void collisionReaction(Character attacker, Character defender);

public abstract void draw(Graphics gc);

}//end class

/\*

PlayerProjectile

Class represents the player's porjectile

@author Ken Tjie

@create 2015 - 07 - 01

@update 2015 - 18 - 01

@version 1.5

\*/

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Color;

public class PlayerProjectile extends Projectile{

/\*

PlayerProjetile

@param x: x-coordinate

@param y: y-coordinate

@Post-condition: instantiates an instance of the PlayerProjectile class

\*/

public PlayerProjectile (int x, int y) {

super(20);

this.x = x + 5;

this.y = y - radius;

maxX = this.x + 2\*radius;

maxY = this.y + 2\*radius;

}//end constructor

/\*

updateHP

@param level: game's current level

@param target: character to be targeted

@Post-condition: moves the projectile according to level

\*/

public void move(int level, Character target) {

x += 5;

maxX = x + 2\*radius;

if (level == 6) {

if ((target.getY() + target.getMaxY()) / 2 > (y + maxY) / 2 && maxX < 400) {

y += 4;

maxY += 4;

}//end if

else if ((target.getY() + target.getMaxY()) / 2 < (y + maxY) / 2 && maxX < 400) {

y -= 4;

maxY -= 4;

}//end else if

}//end if

}//end move

/\*

collisionReaction

@param attacker: character that attacks

@param defender: character that defends

@Post-condition: calculates damage to deal to the defending character

\*/

public void collisionReaction(Character attacker, Character defender) {

double hpDecrease = (((210)/250.0)\*((double)attacker.getAttack()/defender.getDefense())\*(80) + 2)\*(1.5\*(((Math.random()\*16) + 85) / 100));

defender.setHealth(defender.getHealth() - (hpDecrease/4));

}//end collisionReaction

/\*

draw

@param gc: place to draw

@Post-condition: draws the projectile on the screen

\*/

public void draw(Graphics gc) {

gc.setColor(Color.YELLOW);

gc.fillOval(x, y, 2\*radius, 2\*radius);

}//end draw

}//end class

/\*

BossProjectile

Class represents bosses' projectiles

@author Ken Tjie

@create 2016 - 07 - 01

@uodate 2016 - 18 - 01

@version 1.9

\*/

import java.awt.Graphics;

import java.awt.Color;

public class BossProjectile extends Projectile {

private String type;

private int velY = 2;

/\*

BossPorjectile

@param x: x-coordinate

@param y: y-coordinate

@param type: type of the projectile

@Pre-condition: type must not be null

@Post-condition: searches the scores array for the specified points

\*/

public BossProjectile (int x, int y, String type) {

super(20);

this.type = type;

this.x = x - 2 \* radius - 5;

this.y = y - radius;

maxX = this.x + 2 \* radius;

maxY = this.y + 2 \* radius;

}//end constructor

/\*

move

@param level: current level of the game

@param target: character to be targeted

@Post-condition: moves the projectiles movement according to the level

\*/

public void move(int level, Character target) {

x -= 2;

maxX -= 2;

if (level == 3 || level == 4) {

if (y < 0) {

y = 0;

maxY = y + 2 \* radius;

velY = -velY;

}//end if

else if (maxY > 650) {

maxY = 650;

y = 650 - 2\*radius;

velY = -velY;

}//end if

if (level == 4 && radius < 30 && x > 500 && maxX < 900) {

radius ++;

x -= 1;

maxX += 1;

y -= 1;

maxY += 1;

}//end if

y += velY;

maxY += velY;

}//end if

else if (level == 5) {

int randomNum = (int)(Math.random()\*10000);

if (y < 0 || maxY > 650 || randomNum < 20)

velY = -velY;

y += velY;

maxY += velY;

}//end else if

else if (level == 6) {

if ((target.getY() + target.getMaxY()) / 2 > (y + maxY) / 2 && x > 400) {

y += velY;

maxY += velY;

}//end if

else if ((target.getY() + target.getMaxY()) / 2 < (y + maxY) / 2 && x > 400) {

y -= velY;

maxY -= velY;

}//end else if

}//end if

}//end move

/\*

collisionReaction

@param attacker: character that attacks

@param defender: character that defends

@Post-condition: calculates damage to deal to the defending character

\*/

public void collisionReaction(Character attacker, Character defender) {

double hpDecrease = (((210) / 250.0) \* ((double)attacker.getAttack() / defender.getDefense()) \* 80 + 2)\*(1.5 \* (((Math.random() \* 16) + 85) / 100));

defender.setHealth(defender.getHealth() - hpDecrease);

}//end collisionReaction

/\*

draw

@param gc: place to draw

@Post-condition: draws a projectile accoridng to the type of the boss

\*/

public void draw(Graphics gc) {

//change color according to type

switch (type) {

case "Normal":

gc.setColor(Color.LIGHT\_GRAY);

break;

case "Fighting":

gc.setColor(Color.RED);

break;

case "Flying":

gc.setColor(new Color (0, 255, 255));

break;

case "Rock":

gc.setColor(new Color (102, 51, 0));

break;

case "Ground":

gc.setColor(new Color (204, 102, 0));

break;

case "Steel":

gc.setColor(Color.GRAY);

break;

case "Fire":

gc.setColor(Color.ORANGE);

break;

case "Water":

gc.setColor(Color.BLUE);

break;

case "Grass":

gc.setColor(Color.GREEN);

break;

case "Dark":

gc.setColor(Color.BLACK);

break;

case "Ghost":

gc.setColor(new Color(51, 0 , 102));

break;

case "Psychic":

gc.setColor(Color.MAGENTA);

break;

case "Bug":

gc.setColor(new Color(102, 155, 102));

break;

case "Poison":

gc.setColor(new Color(153, 51, 255));

break;

case "Electric":

gc.setColor(Color.YELLOW);

break;

case "Ice":

gc.setColor(new Color(153, 255, 255));

break;

case "Dragon":

gc.setColor(new Color(51, 51, 255));

break;

case "Fairy":

gc.setColor(Color.PINK);

break;

default:

gc.setColor(Color.WHITE);

break;

}//end switch

gc.fillOval(x, y, 2 \* radius, 2 \* radius);

}//end draw

}//end class

/\*

BattlePlayer

Class to instantiate BattleFrame and MusicPlayer classes

@author Ken Tjie

@create 2016 - 06 - 01

@update 2016 - 06 - 01

@version 1.3

\*/

import javax.swing.JFrame;

import java.io.FileNotFoundException;

public class BattlePlayer {

public static void main (String [] args) throws FileNotFoundException {

BattleFrame battleFrame = new BattleFrame();

battleFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//MusicPlayer music = new MusicPlayer();

}//end main

}//end class

/\*

BattleFrame

Class displays window for the game

@author Ken Tjie

@create 2016 - 06 - 01

@update 2016 - 18 - 01

@version 1.6

\*/

import javax.swing.JFrame;

import java.io.FileReader;

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.util.ArrayList;

import java.awt.Dimension;

public class BattleFrame extends JFrame {

private Boss[] chosenBosses = new Boss[6];

private Score[] scores;

/\*

BattleFrame

@Precondition al text files must exist

@Post-condition: instantiates an instance of the BattleFrame class

@Post-condition: fills the array of scores and bosses and contians the instance of the BattlePanel class to play

\*/

public BattleFrame () throws FileNotFoundException {

super("Battle!");

int count = 0;

String temp;

Scanner input = new Scanner (new FileReader("scores.txt"));

//count the lines in scores.txt

while (input.hasNextLine()) {

count ++;

temp = input.nextLine();

}//end while

input.close();

scores = new Score[count/3 + 1];

input = new Scanner (new FileReader("scores.txt"));

//store the scores in the array called scores

for (int i = 0; i < scores.length - 1; i++)

scores[i] = new Score(input.nextLine(), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine()));

input.close();

//fill the first two index positions with regular Pokemon

ArrayList<Boss> bosses = new ArrayList<Boss>();

input = new Scanner (new FileReader("bosses/regulars.txt"));

while (input.hasNextLine())

bosses.add(new Boss(input.nextLine(), input.nextLine(), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine())));

input.close();

for (int i = 0; i < 2; i++) {

int randomNum = (int)(Math.random()\*bosses.size());

chosenBosses[i] = bosses.remove(randomNum);

}//end for

//fill the next two index positions with mega Pokemon

bosses = new ArrayList<Boss>();

input = new Scanner (new FileReader("bosses/megas.txt"));

while (input.hasNextLine())

bosses.add(new Boss(input.nextLine(), input.nextLine(), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine())));

input.close();

for (int i = 2; i < 4; i++) {

int randomNum = (int)(Math.random()\*bosses.size());

chosenBosses[i] = bosses.remove(randomNum);

}//end for

//fill the last two index positions with legendary Pokemon

bosses = new ArrayList<Boss>();

input = new Scanner (new FileReader("bosses/legendaries.txt"));

while (input.hasNextLine())

bosses.add(new Boss(input.nextLine(), input.nextLine(), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine()), Integer.parseInt(input.nextLine())));

input.close();

for (int i = 4; i < 6; i++) {

int randomNum = (int)(Math.random()\*bosses.size());

chosenBosses[i] = bosses.remove(randomNum);

}//end for

add (new BattlePanel(chosenBosses, scores));

this.getContentPane().setPreferredSize(new Dimension(1300, 650));

this.pack();

setResizable(false);

setVisible(true);

}//end constructor

}//end class

/\*

BattlePanel

Class to control logic of the game

@author Ken Tjie

@create 2016 - 06 - 01

@update 2016 - 18 - 01

@version 2.9

\*/

import javax.swing.JPanel;

import javax.swing.Timer;

import javax.swing.ImageIcon;

import javax.swing.JOptionPane;

import javax.swing.JTextArea;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import java.awt.event.KeyListener;

import java.awt.event.KeyEvent;

import java.util.ArrayList;

import java.io.PrintWriter;

import java.io.FileNotFoundException;

public class BattlePanel extends JPanel implements ActionListener, KeyListener {

int velX = 0, velY = 0, bossMove = 1, playerPP = 10, level = 1, points = 0; //player's x and y velocity, boss's velocity, player power points, level and points

String userName; //user's name

boolean gameBeaten = false; //keeps track if the game has been beaten or not

Player player = new Player(); //Player's object

Boss boss; // boss to fight

Boss[] bosses; //array of 6 bosses for the battle

Score[] scores; //array of scores

ArrayList<Projectile> projectiles = new ArrayList<Projectile>(); //ArrayList for all projectiles

ArrayList<Item> items = new ArrayList<Item>(); //ArrayList of all the items

Timer timer = new Timer (5, this); //Timer for animation

/\*

BattlePanel

@param Boss[] bosses: array of bosses for the game

@param Score[] scores: array of scores

@Precondition: bosses must have 6 instances of Boss

@Post-condition: displays instructions, fills array of bosses and scores with given parameters, instantiates a Player

\*/

public BattlePanel (Boss[] bosses, Score[] scores) {

//display instructions and rules

JOptionPane.showMessageDialog(null, "Welcome to Battle!");

JOptionPane.showMessageDialog(null, "You've been challenged by a Pokemon Trainer! Guess what?");

JOptionPane.showMessageDialog(null, "You only have your Pikachu on you!");

JOptionPane.showMessageDialog(null, "Your job is to win the battle! \nIf you lose, you'll lose money and black out!");

JOptionPane.showMessageDialog(null, "It's a 6 vs. 1 battle, don't mess up!");

JOptionPane.showMessageDialog(null, "Arrow keys are to move Pikachu. Space is to shoot your attack.");

JOptionPane.showMessageDialog(null, "You start off with 10 PP (power points) which is how many times you can shoot.\n You must keep track of how many shot you have!");

JOptionPane.showMessageDialog(null, "Several items will pop up on your side of the field:\nThe Leppa Berry is red and will add 3 PP.\nThe Sitrus Berry is yellow and will add 50 HP to Pikachu.\nThe coins will add 10 to your score.");

JOptionPane.showMessageDialog(null, "the more points you gain, the higher you can be on the leaderboards so collect!");

JOptionPane.showMessageDialog(null, "IMPORTANT NOTE: Don't press another arrow key without letting go of the previous one \nor else Pikachu will stop until you press again.");

JOptionPane.showMessageDialog(null, "You can't go off screen or too far to the right");

JOptionPane.showMessageDialog(null, "Also, go to the music folder to play the music manually, it ups the action!");

JOptionPane.showMessageDialog(null, "", "!!!", JOptionPane.PLAIN\_MESSAGE, new ImageIcon("vs..png"));

timer.start(); //start the timer for animations

this.bosses = bosses; //give the array of bosses to the BattlePanel

this.scores = scores; // give the array of scores to the BattlePanel

boss = bosses[level - 1]; //set boss to the birst boss in the array

//give event listeners

addKeyListener(this);

setFocusable(true);

setFocusTraversalKeysEnabled(false);

//ask the user for a name

userName = JOptionPane.showInputDialog("Nickname your Pikachu! Type 'no' if you don't want to");

while (userName == null)

userName = JOptionPane.showInputDialog("Nickname your Pikachu! Type 'no' if you don't want to");

//create a Player depending on user's input

if (userName.equalsIgnoreCase("no"))

player = new Player();

else

player = new Player(userName);

JOptionPane.showMessageDialog(null, "Pokemon Trainer Red sent out "+ boss.getName() + "\nRound " + level + " : " + player.getName() + " Vs. " + boss.getName() + "\n" + player + "\n" + boss);

}//end constructor

/\*

paintComponent

@param gc: area to paint

@Post-condition: paints all instances of Item, Projectile, and Characters on the screen

\*/

public void paintComponent (Graphics gc) {

super.paintComponent(gc);

Graphics2D g2 = (Graphics2D) gc;

Font font = new Font("Comic Sans MS", Font.PLAIN, 18);

g2.setFont(font);

ImageIcon background = new ImageIcon("background.jpg");

background.paintIcon(this, gc, 0, 0);

int rng = (int)(Math.random()\*1000) + 1;

if (rng < 15)

projectiles.add(new BossProjectile(boss.getX(), (boss.getY() + boss.getMaxY()) / 2, boss.getType()));

if (rng > 998)

items.add(new Item("Leppa"));

if (rng < 5)

items.add(new Item("Sitrus"));

if (rng < 2)

items.add(new Item("Coin"));

for (Item currItem : items) {

if (currItem.getName().equalsIgnoreCase("Sitrus") && player.getHealth() < player.getMaxHealth())

currItem.spawn(gc);

else if (currItem.getName().equalsIgnoreCase("Sitrus"))

currItem.moveOffScreen();

else

currItem.spawn(gc);

}//end for

for (Projectile ball : projectiles)

ball.draw(gc);

gc.setColor(new Color(200, 0, 255));

gc.drawString( "Level: " + level + " " + "Score: " + points + " " + player.getName() + " Vs. " + boss.getName(), 0, 20);

player.updateHP(gc);

boss.updateHP(gc);

boss.draw(gc);

player.draw(gc);

}//end paintComponent

/\*

actionPerformed

@param e: event source

@Post-condition: animates all the objects and checks for collisons

\*/

public void actionPerformed (ActionEvent e) {

player.move(velX, velY);

for (Projectile ball : projectiles) {

ball.move(level, player);

}//end for

//change velocity of the boss when it hits the edge

if (boss.getY() < 0 || boss.getMaxY() > 650)

bossMove = -bossMove;

boss.move(bossMove);

//does collision reaction according to the item

for (Item currItem : items) {

if (currItem.isColliding(player)) {

currItem.moveOffScreen();

if (currItem.getName().equalsIgnoreCase("Leppa"))

playerPP += 3;

else if (currItem.getName().equalsIgnoreCase("Sitrus"))

player.setHealth(50 + player.getHealth());

else if (currItem.getName().equalsIgnoreCase("Coin"))

points += 10;

}//end if

}//end for

//check if any projectiles are colliding

for (Projectile ball : projectiles) {

if (ball.isColliding(boss)) {

ball.setX(1400);

ball.collisionReaction(player, boss);

}//end if

else if (ball.isColliding(player)) {

ball.setX(-100);

ball.collisionReaction(boss, player);

}//end else if

//check if the boss' HP is 0

if (boss.getHealth() <= 0) {

//end the game if the player is on the last level

if (level == 6) {

JOptionPane.showMessageDialog(null, "Yay! You beat the game! gg");

gameOver();

}//end if

nextLevel();

}//end if

//check if the player's HP is 0

if (player.getHealth() <= 0) {

player.setHealth(0);

int newPoints = (int)(points \* (((Math.random() \* 26) + 25) / 100)); //reduce the money by 25% - 50%

points = ((newPoints + 5) / 10) \* 10;

repaint();

JOptionPane.showMessageDialog(null, "You lost! You are left with " + points + " coins running to the Pokemon Center");

gameOver();

}//end if

}//end for

repaint();

}//end actionPerformed

/\*

keyPressed

@param e: key that is pressed

@Post-condition: sets velY and velX according to the arrow key pressed

\*/

public void keyPressed (KeyEvent e) {

int code = e.getKeyCode();

//move player up

if (code == KeyEvent.VK\_UP) {

velX = 0;

velY = -2;

}//end if

//move player down

if (code == KeyEvent.VK\_DOWN) {

velX = 0;

velY = 2;

}//end if

//move player right

if (code == KeyEvent.VK\_RIGHT) {

velX = 2;

velY = 0;

}//end if

//move player left

if (code == KeyEvent.VK\_LEFT) {

velX = -2;

velY = 0;

}//end if

}//end keyPressed

public void keyTyped (KeyEvent e) {}//end keyTyped

/\*

keyRekeased

@param e: key that is released

@Post-condition: creates player's projectiles, set velX and velY to 0 if keys are released

\*/

public void keyReleased (KeyEvent e) {

int code = e.getKeyCode();

//shoot a projectile if PP is greater than 0

if (code == KeyEvent.VK\_SPACE) {

if (playerPP > 0) {

playerPP --;

projectiles.add(new PlayerProjectile(player.getMaxX(), (player.getY() + player.getMaxY()) / 2));

}//end inner if

}//end outer if

//stop the player

else {

velX = 0;

velY = 0;

}//end else

}//end keyReleased

/\*

nextLevel

@Post-condition: updates all the variables to play the next level

\*/

public void nextLevel () {

//stop the player

velX = 0;

velY = 0;

boss.setHealth(0); //reduce boss' health to 0

player.setHealth(player.getMaxHealth()); //restore HP

repaint();

//update boss stats and level

JOptionPane.showMessageDialog(null, "Level complete!");

boss = bosses[level];

level ++;

repaint();

//empty old ArrayLists

projectiles = new ArrayList <Projectile>();

items = new ArrayList <Item>();

JOptionPane.showMessageDialog(null, "Pokemon trainer Red sent out "+ boss.getName() + "\nRound " + level + " : " + player.getName() + " Vs. " + boss.getName() + "\n" + player + "\n" + boss);

}//end nextLevel

/\*

gameOver

@Post-condition: adds a score to scores, writes data to files (bosses and scores), asks the user to search, displays leaderboard

\*/

public void gameOver () {

//add a new score to the array of scores

if (!gameBeaten)

scores[scores.length - 1] = new Score (player.getName(), points, level - 1);

else

scores[scores.length - 1] = new Score (player.getName(), points, level);

sort(); //sort scores

//write scores to a text file and all the bosses

try {

PrintWriter output = new PrintWriter("scores.txt");

for (int i = 0; i < scores.length; i++) {

output.println(scores[i].getName());

output.println(scores[i].getPoints());

output.println(scores[i].getLevels());

}//end for

output.close();

output = new PrintWriter("Previous Battle.txt");

output.println(player);

output.println("\nOpponant's Party: ");

for (int i = 0; i < bosses.length; i++)

output.println(bosses[i]);

output.close();

}//end try

catch (FileNotFoundException e) {}//end catch

int count = 0;

//display the leaderboard

String leaderboard = "Leaderboard:\n";

for (Score curr : scores) {

count++;

leaderboard += curr;

if (count == 10)

break;

}//end for

JOptionPane.showMessageDialog(null, leaderboard);

//ask the user if they want to search the leaderboards, even if the score is not in the top 10

String input = JOptionPane.showInputDialog("Would you like to search the leaderboards?\nOnly the top 10 players are shown!\nType name, points or don't search (type 'no')");

while (input == null || (!input.equalsIgnoreCase("name") && !input.equalsIgnoreCase("points") && !input.equalsIgnoreCase("no")))

input = JOptionPane.showInputDialog("Would you like to search the leaderboards?\nOnly the top 10 players are shown!\nType name, points or don't search (type 'no')");

if (input.equalsIgnoreCase("name")) {

input = JOptionPane.showInputDialog("Type in a name to search.");

while (input == null)

input = JOptionPane.showInputDialog("Type in a name to search.");

search(input); //search by name if the user says name

}// end if

else if(input.equalsIgnoreCase("points")) {

int numInput = Integer.parseInt(JOptionPane.showInputDialog("Type in points to search. It must be divisible by 10."));

while (input == null || numInput < 0 || numInput % 10 != 0)

numInput = Integer.parseInt(JOptionPane.showInputDialog("Type in points to search. It must be divisible by 10."));

search(numInput); //search by points if user says points

}// end if

JOptionPane.showMessageDialog(null, "Thanks for playing!");

System.exit(0); //close the program

}//end gameOver

/\*

sort

@Post-condition: sorts the scores array according to points in descending order

\*/

public void sort() {

int i, j;

Score currScore;

for (i = 1; i < scores.length; i++) {

currScore = scores[i];

j = i;

while (j > 0 && scores[j-1].getPoints() < currScore.getPoints()) {

scores[j] = scores[j -1];

j--;

}//end while

scores[j] = currScore;

}//end for

}//end sort

/\*

search

@param points number of points to be searched

@Pre-condition scores must have at least one Score

@Post-condition: searches the scores array for the specified points

\*/

public void search (int points) {

int left = 0, right = scores.length, mid = 0;

boolean found = false;

//use binary search to look for certain points

while (left != right - 1) {

mid = (left + right) / 2;

if (scores[mid].getPoints() > points) left = mid;

else if (scores[mid].getPoints() < points) right = mid;

else {

found = true;

left = right-1;

}//end else

}//end while

if (found) JOptionPane.showMessageDialog(null, scores[mid]);

else JOptionPane.showMessageDialog(null, "Sorry, not found!");

}//end search

/\*

search

@param name name to search

@Pre-condition scores must have at least one Score

@Post-condition: searches the scores array for the specified name

\*/

public void search (String name) {

int i;

boolean found = false;

//use linear search to search by name

for (i = 0; i < scores.length; i++) {

if (scores[i].getName().equalsIgnoreCase(name)) {

found = true;

break;

}//end if

}//end for

if (found)

JOptionPane.showMessageDialog(null, "You are number " + (i + 1) + ": " + scores[i]);

else

JOptionPane.showMessageDialog(null, "Sorry, not found!");

}//end search

}//end class