RBG - Final Project

Authors: Team RGB

I. Play Class (Driver)

Summary: This will be the driver class that will start the running of the game

A. Instantiates the Game class
B. Calls the play() method in the game class

II. Game Class

Summary: This will be the class to run the game

private Menu Menu;
private Environment environment;
private Player player;
private double cameraOffset;
private boolean up, down, left, right;
private AnimationTimer timer;
	//the menu object
	//the environment object
	//the player object
	//# of pixels to offset, to create scrolling effect
	
	//basically runs the whole game

handle() - intialized with AnimationTimer
double futureX, futureY;
 	
public void play()
private void doMove()
 	
private Wall checkWall(double nextX, double nextY)
 	
private int checkDirection(double currentX, double currentY, double nextX, double nextY)
 	
private void updateOffset();
	//method run every frame, so everything happens inside handle - for intro
	//calls
	
	//will return the value in the move arrays that represents what kind of move needs to be made
	//will return the direction of collision (0-up, 1-right, 2-down, 3-right (clockwise from up))
	
	if(playerXPos - cameraOffset > rightBoundary or < leftBoundary)
	cameraOffset += change in xPos

III. Movable Abstract Class

Summary: This will be an abstract interface that represents the methods of players and enemies

private double xPos;
private double yPos;
private double xVel;
private double yVel;
	//the horizontal position
	//the vertical position
	//the horizontal velocity
	//the vertical velocity

public void move()
private void updateAnimation()
 	
public void setXPos(double x)
public void setYPos(double y)
public void setXVelocity(double xVelocity)
public void setYVelocity(double yVelocity)
 	
public double getXPos()
public double getYPos()
public double getXVelocity()
public double getYVelocity()
 	
getWidth()
getHeight()
	//the method to move the object
	//the method to update the running/moving animation for a movable object
	
	//Sets the x position of the object
	//Sets the y position of the object
	//Sets the x velocity of the object
	//Sets the y velocity of the object
	
	//Returns the x position of the object
	//Returns the y position of the object
	//Returns the x velocity of the object
	//Returns the y velocity of the object
	
	//Returns the width of the image
	//Returns the height of the image

IV. Player Class extends Moveable - JavaFX class

Summary: The Player class will extend Moveable and will be a representation of the user playing the game.

private double xPos;
private double yPos;
private double xVel;
private double yVel;
 	
private static final double X_ACCEL, FRICT_ACCEL, GRAV_ACCEL, JUMP_ACCEL, MAX_VEL;
 	
private Image picStill;
private Image picRunning1;
	//the horizontal position
	//the vertical position
	//the horizontal velocity
	//the vertical velocity
	
	//constants needed for movement
	
	//character sprite when idle
	//1st running character sprite

private Image picRunning2;
private Image picJump;
private int animationState;
 	
private boolean isAlive;
 	
public boolean onGreenHorizontal;
public boolean onGreenVertical;
	//2nd running character sprite
	//character sprite when jumping
	//0 = still, 1 = running, 2= jumping
	
	//whether or not the user is alive (red + enemy)
	//whether or not you are on a green floor/ceiling
	//whether or not you are on a green wall

public void move()
private void updateAnimation()
 	
public boolean isAlive()
 	
public boolean isOnGreenHorizontal()
public boolean isOnGreenVertical()
	//the method to move the object
	//the method to update the running/moving animation for a movable object
	//decides if game will continue or if game is over
	//allows user right/left but not up down
	//allows user to use up/down but not right/left

V. Enemy Class extends Moveable - JavaFX class

Summary: The Enemy class will extends movable

private int xPos;
private int yPos;
private Image pic;
	//the horizontal position
	//the vertical position
	//the String that represented the png image that represents the enemy

public void move() private void updateAnimation()	//the method to move the object //the method to update the running/moving animation for a movable object, will be called by move

VI. Environment Abstract Class - JavaFX class

Summary: The Environment class will hold the map and the collision array. It is a superclass of the Intro and the Game environment

private Image map;
 	
private int[][] collisionsArray = {};
	//the String that holds the png file name of the map
	//the collisions array that holds the values of

	each element at each position

public Image getMapImage() //implemented
 	
public boolean isCollision(int col, int row)
 	
public int getType(int col, int row)
 	
private void createCollisionsArray(String txtFileName)
	//returns the map's image so that Game can display it
	//returns whether or not the array index in the collisions array is occupied
	//returns the tile type of the selected index
	
	//called by constructor, loads the .txt file into collisionsArray

VII. GameEnvironment extends Environment - JavaFX class

Summary: The GameEnvironment class will extend Environment and will actually implement the checkMove method and have its own helper methods

private Image map;
 	
private int[][] collisionsArray = {};
	//the String that holds the png file name of the map
	//the collisions array that holds the values of each element at each position

public Image getMapImage() public int checkMove(int nextX, int nextY);	//returns the map's png file name as a String so that Game can display it //checks and returns what kind of move can be made based on the future position in the collisions array. Ensures that collisions array is encapsulated

VIII. IntroEnvironment extends Environment

Summary: The IntroEnvironment class will extend Environment and will actually implement the checkMove method and have its own helper methods

private Image map;
private Image foreground;
private Image background;
 	
private int[][] collisionsArray = {};
	//map Image
	//foreground Image for parallax
	//background Image for parallax
	//the collisions array that holds the values of each element at each position

public Image getMapImage()	//returns the map's png file name as a String so that Game can display it

IX. Menu Abstract Class

Summary: Menu is an abstract class that

private Button options;
private Button credits;
private Button back;
private Button endGame;
	//the options button
	//the credits button
	//the back button
	//the button to end the game

private void createButtons() - abstract
private void displayButtons() - abstract
private void runMenu() - abstract
private void options()
private void credits()
 	
private void back(); - abstract
public void effects();
	//method to instantiate the buttons
	//method to actually display the methods
	//this method will actually run the menu
	//method is run when options is pressed
	//method to display the credits when credits is pressed
	//method when the back button is pressed
	// contains all effects for buttons when pressed

X. MainMenu class extends Menu

Summary: MainMenu is a type of Menu and it begins the game by having the user choose a character and then proceeding with the rest of the game

private Button play;
private Button options;
private Button endGame;
	//the button to play the game
	//the button for options
	//the button to end the game

private Player play()	//when play is called, it will call characterSelect then begin game

private Player characterSelect()
private void options()
//implemented in Menu
private void credits()
//implemented in Menu
	//to select a character to be used
	//when options is pressed
	
	//method to display credits when credits is pressed

XI. PauseMenu class extends Menu

Summary: PauseMenu is a type of Menu and it occurs when the escape key is pressed

private Button resume;
private Button options;
private Button endGame;
private Button restartSector;
private Button restartGame;
	//the button to resume the game
	//the button for options
	//the button to end the game
	//the button to restart the sector user is on
	//the button to restart the entire game

private void resume()
 	
private void options()
//implemented in Menu
private void credits()
//implemented in Menu
private void restartSector()
private void restartGame()
	//when resume is called, it will call go back to game
	//when options is pressed
	
	//method to display credits when credits is pressed
	//method to restart the sector the user is on
	//method to restart the game when user wants

XII. Wall Interface

Summary: Wall Interface is the interface that represents the walls

//no variables	//no variables

public void rightInteract(Player player);
public void leftInteract(Player player);
public void ceilingInteract(Player player);
public void floorInteract(Player player);
	//interaction with right-facing wall
	//interaction with left-facing wall
	//interaction with ceiling
	//interaction with floor

XIII. BlueWall implements Wall

Summary: Blue Wall (bounce)

private static final double HORI_ACCEL, FLOOR_ACCEL, CEIL_ACCEL;	//no variables

public void rightInteract(Player player);
 	
public void leftInteract(Player player);
 	
public void ceilingInteract(Player player);
 	
public void floorInteract(Player player);
	//when the wall is facing right
	Set xVelocity to -1 * HORI_ACCEL;
	//when the wall is facing left
	Set xVelocity to HORI_ACCEL;
	//when the wall is on the ceiling
	Set yVelocity to CEIL_ACCEL;
	//when the wall is on the ground
	Set yVelocity to FLOOR_ACCEL;

XIV. GreenWall implements Wall

Summary: Green Wall (stick)

//no variables	//no variables

public void rightInteract(Player player);
 	
public void leftInteract(Player player);
 	
public void ceilingInteract(Player player);
 	
public void floorInteract(Player player);
	//when the wall is facing right
	Set onGreenHorizontal to true;
	//when the wall is facing left
	Set onGreenHorizontal to true;
	//when the wall is on the ceiling
	Set onGreenVertical to true;
	//when the wall is on the ground
	Set onGreenVertical to true;

XV. NormalWall implements Wall

Summary: Normal Wall (nothing happens)

//no variables	

public void rightInteract(Player player);
 	
public void leftInteract(Player player);
 	
public void ceilingInteract(Player player);
 	
public void floorInteract(Player player);
	//when the wall is facing right
	Set xVelocity to 0;
	Set xPos to touching wall;
	//when the wall is facing left
	Set xVelocity to 0;
	Set xPos to touching wall;
	//when the wall is on the ceiling
	Set yVelocity to 0;
	Set yPos to touching wall;
	//when the wall is on the ground

	Set yVelocity to 0; Set yPos to touching wall;

XVI. RedWall implements Wall

Summary: Red Wall (death)

//no variables	

public void rightInteract(Player player);
 	
public void leftInteract(Player player);
 	
public void ceilingInteract(Player player);
 	
public void floorInteract(Player player);
	//when the wall is facing right
	Set xVelocity and yVelocity to 0;
	Set isAlive to false;
	//when the wall is facing left
	Set xVelocity and yVelocity to 0;
	Set isAlive to false;
	//when the wall is on the ceiling
	Set xVelocity and yVelocity to 0;
	Set isAlive to false;
	//when the wall is on the ground
	Set xVelocity and yVelocity to 0;
	Set isAlive to false;

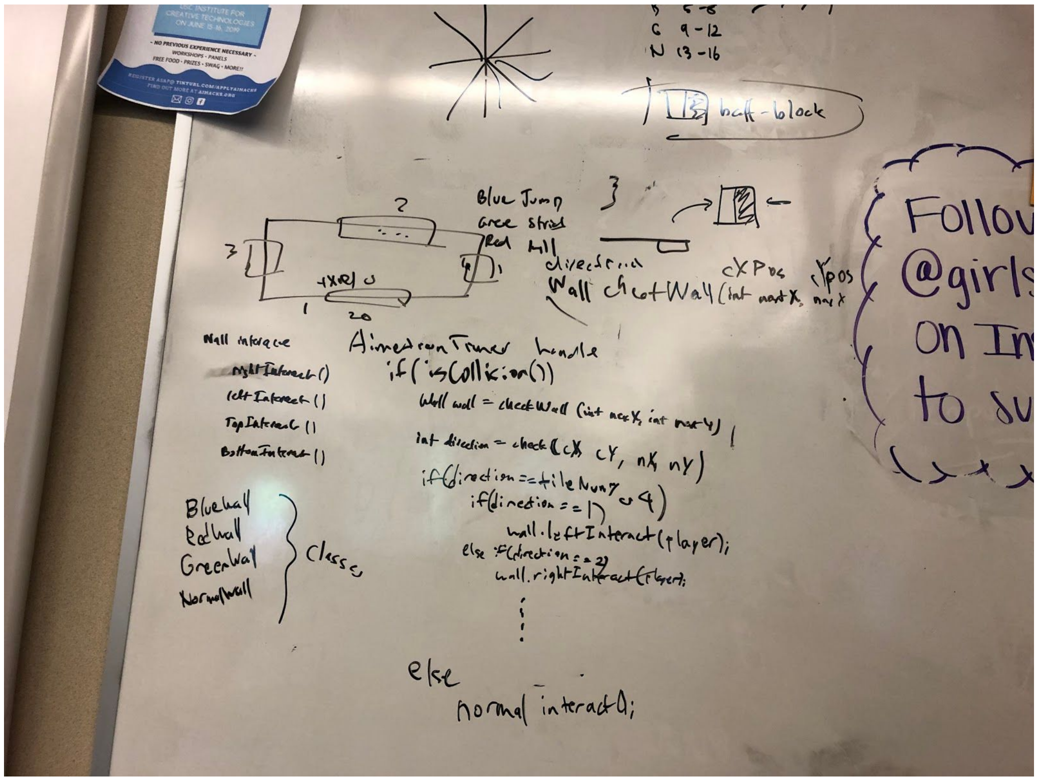
Packages needed: javafx.scene.Media, javafx.scene.MediaPlayer;

XVII. Music

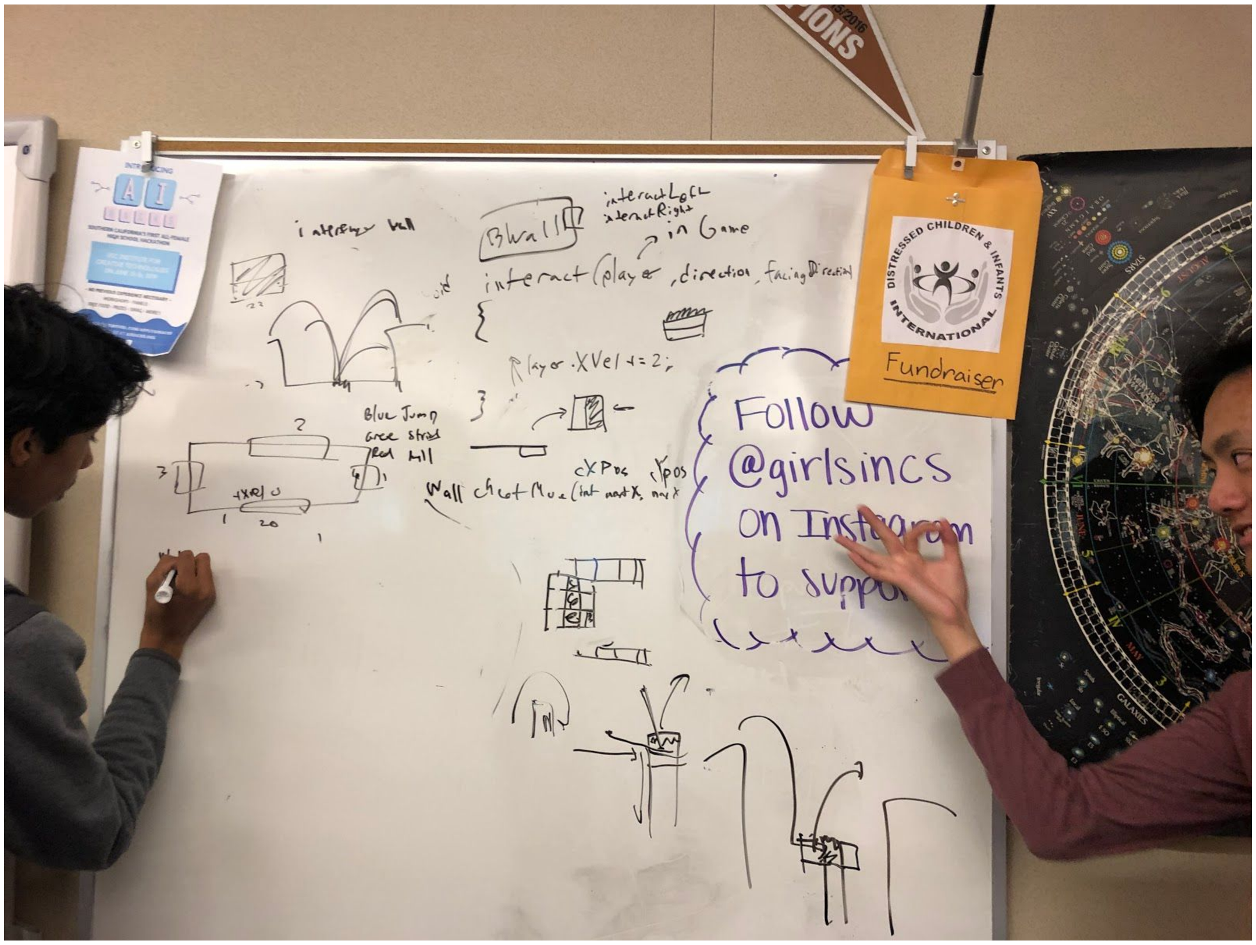
Summary: Plays different kinds of sound

String fileName;	//String of the music file that will play in the background throughout the entire game

public void loop(String file)
 	
public void play(String file)
	//Method will loop the music
	
	//Method will play the music file



```
Wall checkWall(int nextX, nextY
AimedrunTimer handle
if(isCollision())
    Wall wall = checkWall (int nextX, int nextY)
    int direction = check(cX, cY, nX, nY)
    if(direction==tileNum7 || 4)
        if(direction==1)
            wall.leftInteract(player);
        else if(direction==2)
            wall.rightInteract(player);
            :
else
    normal interact();
```



```
interact(player, direction, facingDirection)
{
    player.XVel += 2;
Wall checkMue(int nextX, nextY
```