|  |  |
| --- | --- |
| Egyptian cotton ™ | Yoshi’s Nightmare |



|  |
| --- |
| Developers | Sapkota, Subin | Turgeon, Trevor |

Table of Contents

[Project Description 2](#_Toc437868242)

[Accomplishments 3](#_Toc437868243)

[**Unique Aspects** 3](#_Toc437868244)

[Gantt Chart 5](#_Toc437868245)

[Flow Charts 6](#_Toc437868246)

[UML Diagram 10](#_Toc437868247)

[**Class Relationships** 10](#_Toc437868248)

[UML Class Diagram 11](#_Toc437868249)

[User Guide 12](#_Toc437868250)

[**Welcome Window** 12](#_Toc437868251)

[**Controls Window** 12](#_Toc437868252)

[**Main Window** 13](#_Toc437868253)

[**Points System** 14](#_Toc437868254)

[**Specials** 14](#_Toc437868255)

[Testing 15](#_Toc437868256)

[Snippets of Code 16](#_Toc437868257)

[**Read From Map** 16](#_Toc437868258)

[**Draw Maze** 17](#_Toc437868259)

[**Image Scaling** 18](#_Toc437868260)

[**Run Method:** 18](#_Toc437868261)

[**Ghosts Movement** 20](#_Toc437868262)

[**Yoshi Movement** 21](#_Toc437868263)

[**Ghosts** 22](#_Toc437868264)

[**Yoshi** 24](#_Toc437868265)

# Project Description

Yoshi’s Nightmare is a recreation of Pac-Man with a Mario themed twist to it. Yoshi is stuck in a nightmare where his head is floating around a maze. But wait! There’s a twist to the story. Yoshi is being chased by ghosts of Princess Peach, Mario, Bowser and Luigi. Help Yoshi escape this nightmare by eating all mushrooms, collecting all stars and defeating the ghosts. Yoshi earns points for every mushroom consumed and becomes invincible for every star. Catch the ghosts when you’re invincible and you earn even more points. Beware of the powers of the ghosts! Luigi will throw green shells at you, Bowser will roll around as a furious shell, Mario will throw fire balls, and Princess Peach will drop cute penguin traps. Don’t die more than once because the third death will mean an eternal nightmare for Yoshi! So get out there, collect all the mushrooms and stars and help Yoshi beat this nightmare!

# Accomplishments

Yoshi’s Nightmare was created from the ground up by a team of two student software developers in Java. The game uses a variety of Java libraries including:

* Swing
* Key & mouse event listeners
* File I/O
* TimerTask
* ImageIO

Our game contains 11 classes, all made from the scratch by the two developers. The class diagram in System Design section shows how the 11 classes work with each other. The game graphics were obtained from other sources and manipulated using photoshop to fit the design of our game. A file containing integers provides the data for the grid for our map. We have three files with maps for three worlds (mazes).

## **Unique Aspects**

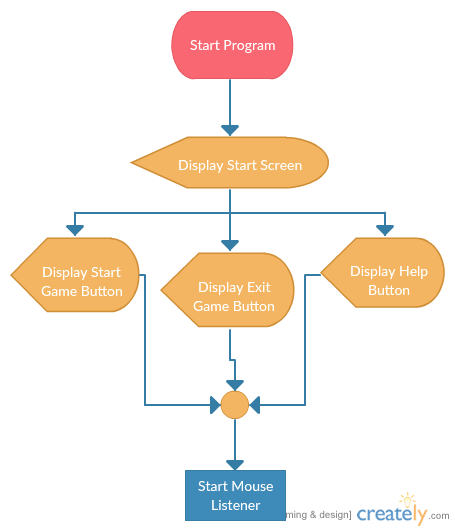
Yoshi’s Nightmare implements a PacMan engine and adds more functionality to it. Mushroom, stars and ghosts have same functionality as the original PacMan game. Mushroom is the same as the lights in PacMan, stars act as the berries and ghosts behave as ghosts of PacMan. Yoshi earns points for eating the mushrooms and stars, and loses life for getting eaten by a ghost. Along with these pre-existing rules, we added some unique aspects to our game, they are:

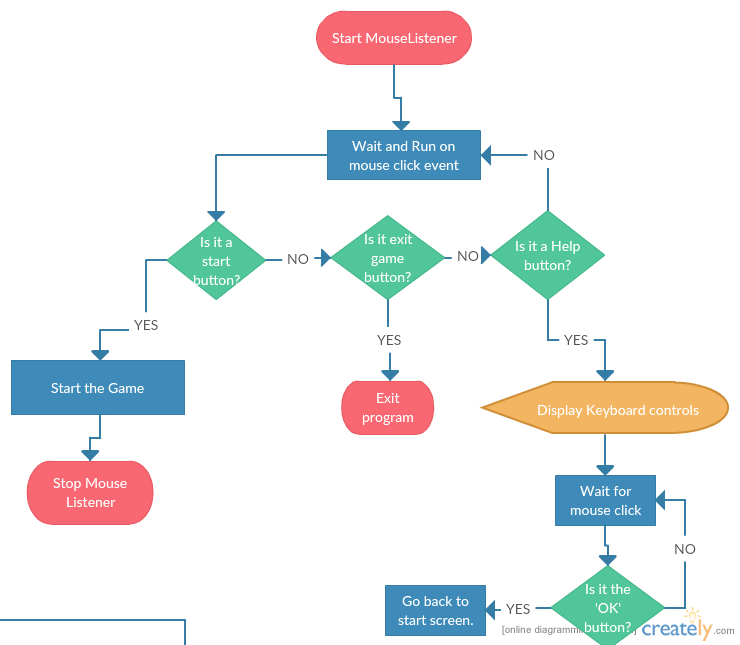
1. After a certain amount of time the ghosts will enable their special ability. These abilities include:
   1. Luigi throws a green shell. If Yoshi gets hit by the shell he will lose a life.
   2. Bowser turns into a shell and starts moving quicker than all the characters.
   3. Mario throws a fire ball. This will take a life away from Yoshi if he gets hit by it.
   4. Princess Peach will leave a penguin suit trap. Stepping into the trap will make Yoshi lose a life.
2. Along with the ghosts, we also added Bullet Bills into the map. A random number of bullets start at random positions in the maze and start moving. Yoshi loses a life every time it gets hit by the bullets too.
3. The game also includes three maps and four maze colours. The colour of the background is picked randomly every time the game runs from five possible colours.
4. All the images in the game will rescale so that they fit on any screen resolution.

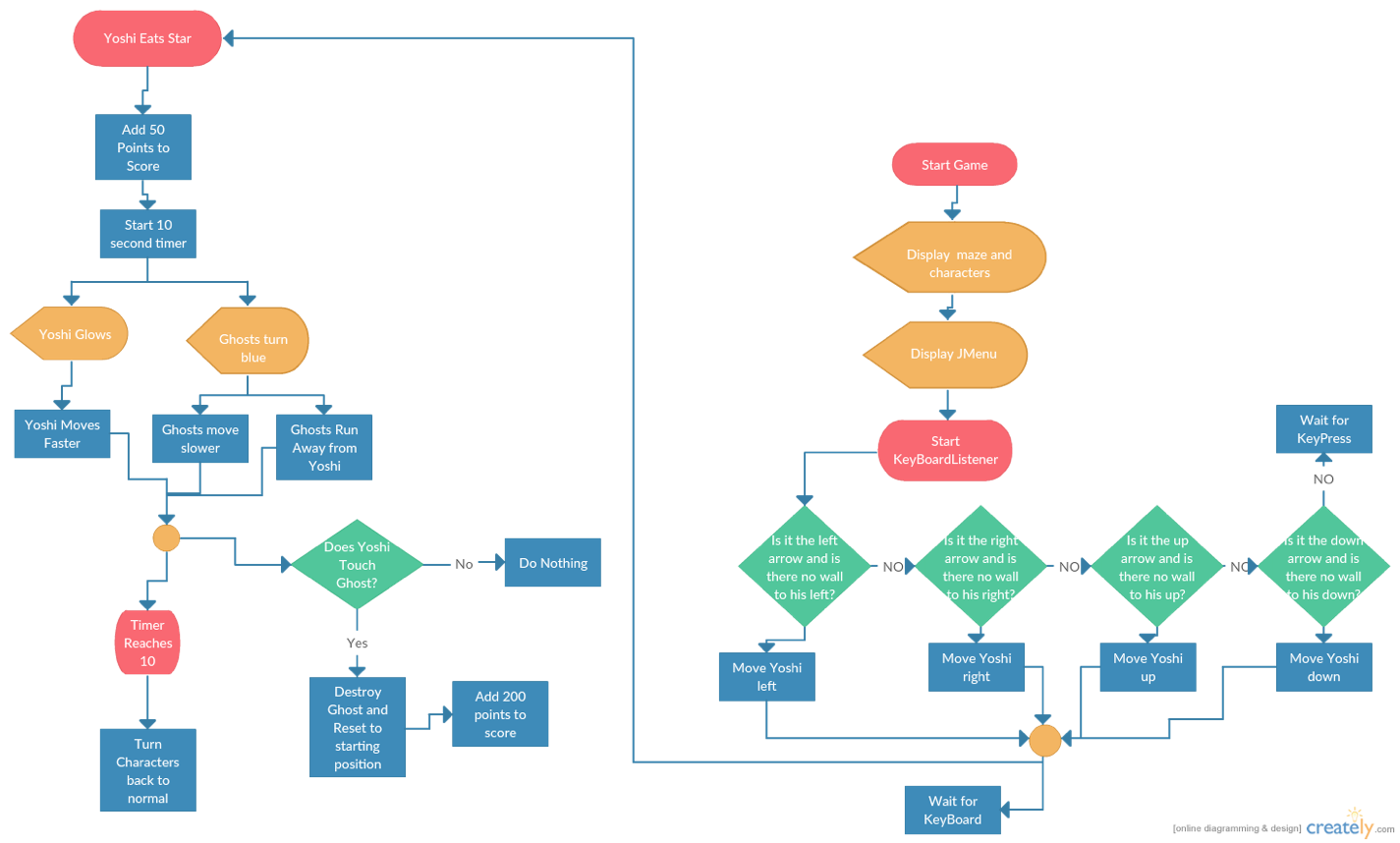
# Gantt Chart

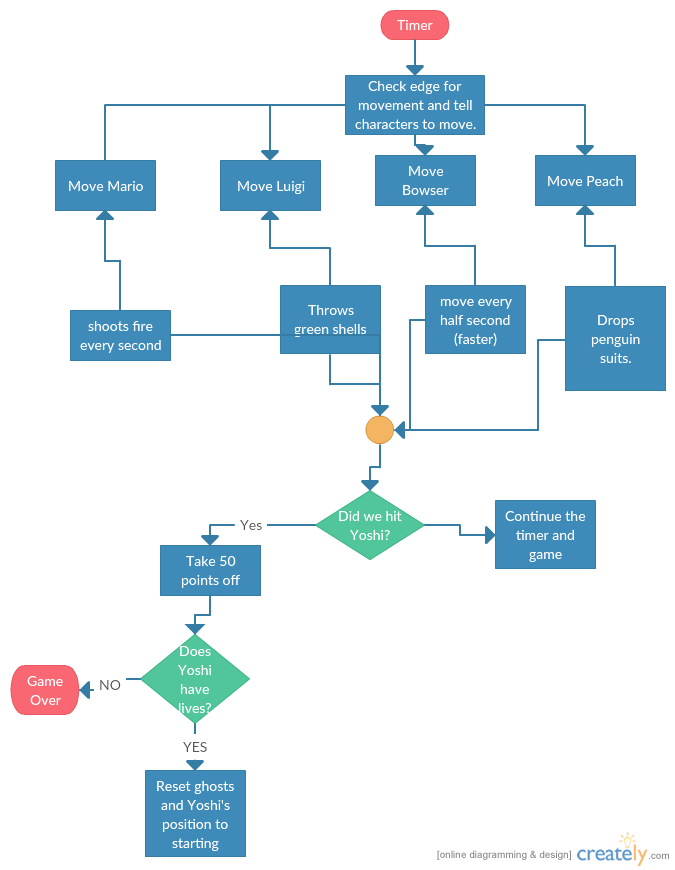


# Flow Charts









# UML Diagram

## **\\mark13\ssapkota$\Desktop\Yoshi_2.pngClass Relationships**

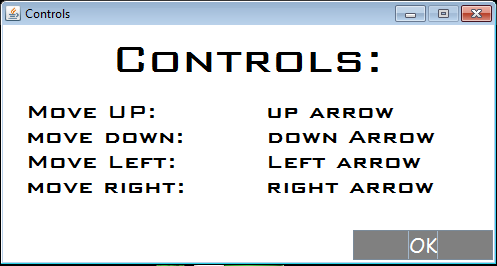
# UML Class Diagram

# User Guide

## **Welcome Window**

The game starts with a welcome window where the user has three choices. User can click on ‘Start’ or ‘Controls’ or choose to exit the game.

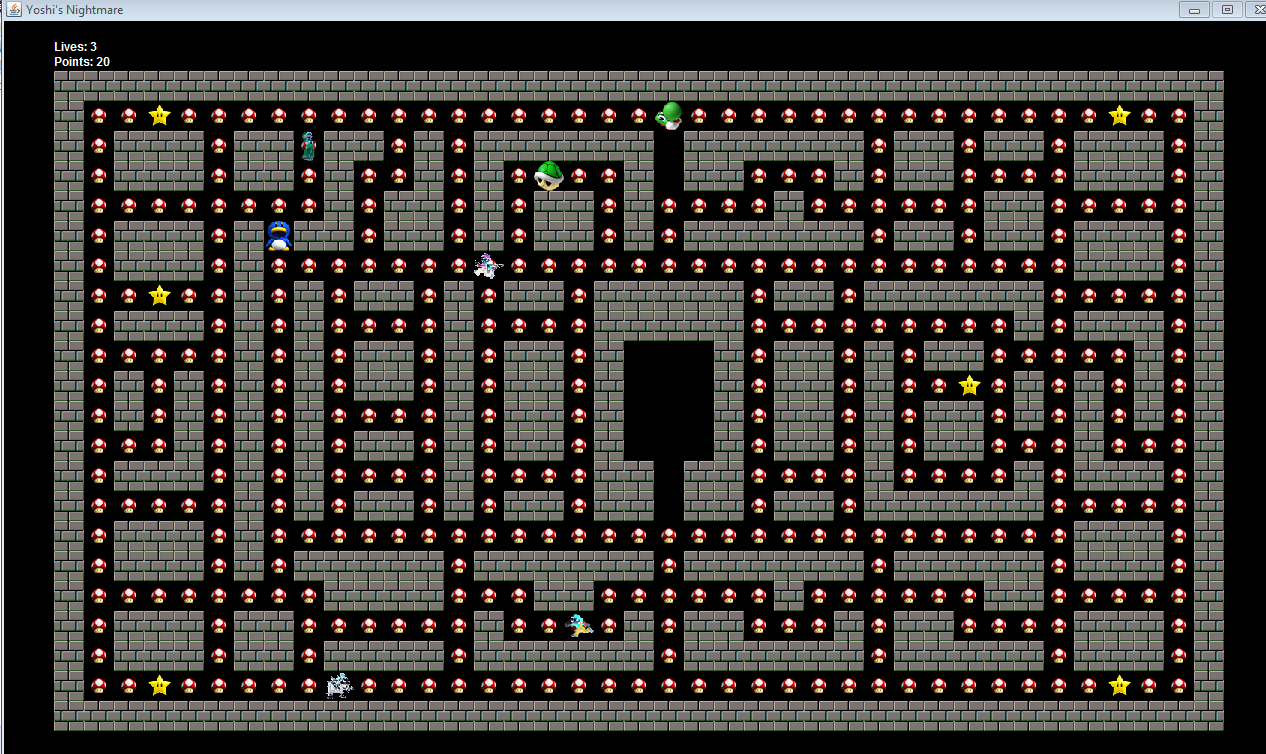
## **Controls Window**

When user clicks on ‘Controls’, the controls window will appear. This shows the manual for the game.

As shown in the controls window, Yoshi’s Nightmare has four possible controls change Yoshi’s direction as follows:

1. Up Arrow: Change direction to up.
2. Down Arrow: Change direction to down.
3. Left Arrow: Change direction to left.
4. Right Arrow: Change direction to right.

## **Main Window**

 The main window contains the maze with Yoshi and multiple ghosts. User can move Yoshi around and eat the mushrooms and stars. User must avoid all other characters including the ghosts, shells, penguins, bullets, and fire balls.

## **Points System**

User earns following amount of points for eating different elements in the game:

* Mushroom: 10 points
* Stars: 50 points
  + After eating the star, ghosts will turn blue and Yoshi will turn yellow. At this point user can eat the ghosts and this will earn the user 100 points.
  + User can’t eat the same ghost more than once.
  + User will turn back to normal after a certain amount of time at which point they can’t eat ghosts.

## **Specials**

Every ghost has a special that activates in the game and stays for a certain amount of time. If user is touched by any of these specials, then they will lose a life. A user only has 3 lives before the game is over! The ghost specials are:

* Fire Balls: Mario will throw fireballs that travel in one direction and disappear once they hit a wall.
* Green Shell: Luigi will throw a green shell that disappears after hitting intersections and walls 6 times.
* Bowser Shell: Bowser will turn into a shell and move twice as fast as other ghosts. He will only stop after hitting a wall head on.
* Penguin Suits: These adorable penuin suit traps will be dropped by Peach and won’t go away until Yoshi gets hit by them, or when Yoshi eats a star. The suits will move once Peach drops another one somewhere else in the maze.
* Bullet Bills: Bullet Bills will appear on the map and move in one direction until they get out of the map. Yoshi will lose a life if hit by a Bullet Bill. The only way to make a Bullet Bill disappear is by eating a star or by losing a life.

# Testing

Testing for the code was carried out by implementing user testing. Software engineering class of 2015 was allowed to play the game after it’s completion and any errors in the gameplay were debugged after the completion of testing. Java Debugger tool was used throughout the game development to check various breakpoints. The game was also built in small pieces and the team didn’t move on to another piece of the game unitl the one being worked on became bugs free.

# Snippets of Code

Here are some codes that are important for the game.

## **Read From Map**

//Read from the world map files and store the values in the maze array.

**private** **void** createMapArray(){

Scanner fileReader;

ArrayList<String> lineList = **new** ArrayList<String>();

**try**{

fileReader = **new** Scanner(**new** File(mapName[level])); //Read from file stored in mapName array.

**while** (**true**) {

String line = **null**;

**try** {

line = fileReader.nextLine();

} **catch** (Exception eof) {

}

**if** (line == **null**) {

**break**;

}

lineList.add(line);

}

//Initialize a map array with required number of rows and columns.

rowsOfBlocks = lineList.size();

columnsOfBlocks = lineList.get(0).length();

mazeArray = **new** **int**[rowsOfBlocks][columnsOfBlocks];

//Get a line, that's the row.

**for** (**int** row = 0; row < rowsOfBlocks; row++) {

String line = lineList.get(row);

//Get characters from line, that's the column for maze array.

**for** (**int** column = 0; column < columnsOfBlocks; column++) {

**char** type = line.charAt(column);

mazeArray[row][column] = Character.*getNumericValue*(type);

}

}

}**catch** (FileNotFoundException e) {

System.***out***.println("Maze map file not found");

}

}

## **Draw Maze**

//Following code draws the maze using the map obtained.

**public** **void** drawMap(**int** blocknumber){

mainPane.setLayout(**null**);

//Loop through the maze Array.

**for**(**int** i = 0; i < rowsOfBlocks; i++){

**for**(**int** j = 0; j < columnsOfBlocks; j++){

**switch**(mazeArray[i][j]){ //Check maze Array Component

**case** BLOCKS:

Image newImage = (**new** ImageIcon(blockImageName[blocknumber])).getImage().getScaledInstance(scale, scale, Image.***SCALE\_DEFAULT***); //Scale image

block[i][j] = **new** JLabel(**new** ImageIcon(newImage)); //Store image in block array.

block[i][j].setBounds(i\*scale + offset, j\*scale + offset, scale, scale); //Set image position for screen.

block[i][j].setVisible(**true**);

mainPane.add(block[i][j]); //add block to the panel used in screen.

**break**;

**case** MUSHROOMS:

Image mushrooms = (**new** ImageIcon("mushroom.png")).getImage().getScaledInstance(scale, scale, Image.***SCALE\_DEFAULT***);

block[i][j] = **new** JLabel(**new** ImageIcon(mushrooms));

block[i][j].setBounds(i\*scale+offset, j\*scale+offset, scale, scale);

block[i][j].setVisible(**true**);

mainPane.add(block[i][j]);

**break**;

**case** STARS:

Image stars = (**new** ImageIcon("star.png")).getImage().getScaledInstance(scale, scale, Image.***SCALE\_DEFAULT***);

block[i][j] = **new** JLabel(**new** ImageIcon(stars));

block[i][j].setBounds(i\*scale+offset, j\*scale+offset, scale, scale);

block[i][j].setVisible(**true**);

mainPane.add(block[i][j]);

**break**;

**case** CORNER\_SHROOMS:

Image cornerShrooms = (**new** ImageIcon("mushroom.png")).getImage().getScaledInstance(scale, scale, Image.***SCALE\_DEFAULT***);

mainPane.setLayout(**null**);

block[i][j] = **new** JLabel(**new** ImageIcon(cornerShrooms));

block[i][j].setBounds(i\*scale + offset, j\*scale + offset, scale, scale);

block[i][j].setVisible(**true**);

mainPane.add(block[i][j]);

**break**;

**case** EMPTY\_SPOTS:

**break**;

**case** YOSHI:

block[i][j] = **new** JLabel();

block[i][j].setBounds(i\*scale + offset, j\*scale + offset, scale, scale);

block[i][j].setVisible(**false**);

**break**;

}

}

}

}

## **Image Scaling**

//This will find the best scale for the images so that maze will fit on all screen resoultions.

Dimension screenSize = Toolkit.*getDefaultToolkit*().getScreenSize();

width = screenSize.getWidth();

height = screenSize.getHeight();

scale = (**int**) (screenSize.getWidth()-100)/39; //Scale for images in maze.

## **Run Method:**

// Runs every time the timer goes off (100ms for this game) inside the controller.

**public** **void** run() {

counter\_of\_timer\_runtime++;

//Ghosts special check every time timer runs.

marioSpecial();

bowserSpecial();

luigiSpecial();

**if**(counter\_of\_timer\_runtime == YOSHI\_RUN\_TIME){ //Move Yoshi.

//If User had pressed a direction before and it was stored, then check if there's a wall there now.

**if**(myYoshi.getLastSpecifiedDirection() != noDirection && isWallHitInDirection(myYoshi.getLastSpecifiedDirection())){

//No wall in last specified direction. Change direction.

myYoshi.changeDirection(myYoshi.getLastSpecifiedDirection());

moveYoshi();

myYoshi.setLastSpecifiedDirection(noDirection);

} **else** **if**(isWallHitInDirection(myYoshi.getDirection())){

//If user hasn't pressed a direction before.

myYoshi.changeDirection(myYoshi.getDirection());

moveYoshi();

}

counter\_of\_timer\_runtime = 0; //Reset counter.

//GHOSTS MOVEMENTS

} **else** **if**(counter\_of\_timer\_runtime == GHOSTS\_RUN\_TIME){

counter\_of\_ghosts\_special\_runtime++;//Add ghosts special runtime counter.

//Is it itme for ghosts special to activate?

**if**(counter\_of\_ghosts\_special\_runtime >= SPECIAL\_RUN\_TIME){

//If mario special hasn't already been activated and if yoshi isn't glowing.

**if**((!ghosts[MARIO].getSpecialValue()) &&!myYoshi.getYoshiGlow()){

((Mario)ghosts[MARIO]).setFireBallDirection(ghosts[MARIO].getDirection());

((Mario)ghosts[MARIO]).setFireXIndex(ghosts[MARIO].getXIndex());

((Mario)ghosts[MARIO]).setFireYIndex(ghosts[MARIO].getYIndex());

((Mario)ghosts[MARIO]).drawFireBall();

ghosts[MARIO].setSpecialInitiated();

}

//If BOWSER special hasn't already been activated and if yoshi isn't glowing.

**if**((!ghosts[BOWSER].getSpecialValue()) && !myYoshi.getYoshiGlow()){

ghosts[BOWSER].setSpecialInitiated();

}

//If LUIGI special hasn't already been activated and if yoshi isn't glowing.

**if** ((!ghosts[LUIGI].getSpecialValue()) && !myYoshi.getYoshiGlow()){

((Luigi)ghosts[LUIGI]).setShellXIndex(ghosts[LUIGI].getXIndex());

((Luigi)ghosts[LUIGI]).setShellYIndex(ghosts[LUIGI].getYIndex());

((Luigi)ghosts[LUIGI]).drawShell();

ghosts[LUIGI].setSpecialInitiated();

}

//If Yoshi isn't glowing activate Peach.

**if**(!myYoshi.getYoshiGlow()){

((Peach)ghosts[PEACH]).setTrapXIndex(ghosts[PEACH].getXIndex());

((Peach)ghosts[PEACH]).setTrapYIndex(ghosts[PEACH].getYIndex());

((Peach)ghosts[PEACH]).drawTrap();

}

//Reset special runtime counter.

counter\_of\_ghosts\_special\_runtime = 0;

}

**for**(**int** i=0; i<NUMBER\_OF\_GHOSTS; i++){

//Move individual ghosts

ghostsMovement(i);

}

}

//Check if glowing timer for ghosts has ended.

**if**(counter\_glow\_timer\_runtime > time\_to\_reset\_glow){

counter\_glow\_timer\_runtime = 0;

myYoshi.setYoshiGlow(); //Reset Yoshi's glow

**for**(**int** i = 0; i < NUMBER\_OF\_GHOSTS; i++){ //Reset Ghosts to have them stop glowing.

**if**(ghosts[i].getGhostsGlow()){

ghosts[i].setGhostsGlow();

}

}

}

//bullet bill, launch and move them

launchBullets();

**if** (counter\_of\_bullet\_runtime >= time\_between\_bullet\_bills){

**for**(**int** i = 0; i < NUMBER\_OF\_BULLETS; i++){

bullets[i].moveBullet(i);

didYoshiTouchBullet(); //Check if Yoshi touched Bullet.

}

}

}

## **Ghosts Movement**

//Handles all Ghosts Movement inside the Controller.

//Method to Move ghosts.

**private** **void** ghostsMovement(**int** ghostNumber){

// If Ghosts hit corner

**if**(maze.mazeArray[ghosts[ghostNumber].getXIndex()][ghosts[ghostNumber].getYIndex()] == CORNER\_SHROOMS){

Random randomGenerator = **new** Random();

**while** (**true**) {

**int** newDirection = randomGenerator.nextInt(NUMBER\_OF\_DIRECTIONS); //Random direction for ghosts.

//Is new ghost direction going backwards?

**if**((newDirection == DOWN && ghosts[ghostNumber].getDirection() == UP) || (newDirection == UP && ghosts[ghostNumber].getDirection() == DOWN)

|| (newDirection == LEFT && ghosts[ghostNumber].getDirection() == RIGHT) || (newDirection == RIGHT && ghosts[ghostNumber].getDirection() == LEFT)) {

//Loop again if ghosts is gowing backwards.

} **else** **if** (!willGhostHitWall(newDirection, ghostNumber)) {

//If ghost doesn't hit walls.

**if**(ghosts[ghostNumber].getGhostsGlow()){ //If ghost is glowing.

counter\_glow\_timer\_runtime++;

ghosts[ghostNumber].changeDirection(newDirection);

ghosts[ghostNumber].moveGhosts();

**break**;

} **else**{ //If ghost isn't glowing.

//If bowser has his special value activated.

**if**(ghostNumber == BOWSER && ghosts[BOWSER].getSpecialValue()){

ghosts[ghostNumber].changeDirection(newDirection);

((Bowser)ghosts[ghostNumber]).moveBowserShell();

} **else** { //If ghost being moved isn't bowser.

ghosts[ghostNumber].changeDirection(newDirection);

ghosts[ghostNumber].moveGhosts();

}

didYoshiTouchGhost();

**break**;

}

}

}

} **else** { // If corner is not hit.

**if**(ghostNumber == BOWSER && ghosts[BOWSER].getSpecialValue()){

((Bowser)ghosts[ghostNumber]).moveBowserShell();

didYoshiTouchGhost();

} **else**{

ghosts[ghostNumber].moveGhosts();

didYoshiTouchGhost();

}

}

}

## **Yoshi Movement**

//Handles Yoshi’s movement inside the Controller.

**private** **void** moveYoshi(){

myYoshi.moveYoshi(); //Move Yoshi.

updateScore(); //Update Score.

updateMaze(); //Update the Maze Array for the element Yoshi just moved to.

repaintPoints(); //Repaint points.

didYoshiTouchGhost();//Check if Yoshi touched Ghosts.

didYoshiTouchFire(); //Check if Yoshi touched Fire Ball.

didYoshiTouchTrap(); //Check if Yoshi touched Penguin Suit Trap.

didYoshiTouchShell(); //Check if Yoshi touched Green Shell.

didYoshiTouchBullet(); //Check if Yoshi touched Bullet Bills.

**if**(didYoshiWin()){ //Did yoshi win.

maze.setLevel(); //Increase level.

**if**(maze.getLevel() < maze.getNumberOfMaps()){ //If another level exists.

TIME\_TO\_MOVE\_CHARACTERS = TIME\_TO\_MOVE\_CHARACTERS - 15; //Move characters faster.

SPECIAL\_RUN\_TIME = SPECIAL\_RUN\_TIME + 5; //Ghosts enable special power faster

time\_between\_bullet\_bills = time\_between\_bullet\_bills - 5; //Bullet bills shoot faster

time\_to\_reset\_glow = time\_to\_reset\_glow - 5; // Glow time for Yoshi is shorter.

//Restrat maze, character and labels.

maze.startMaze();

startCharacters("ALL");

repaintLives();

repaintPoints();

} **else**{

//If Yoshi won then show winning screen.

mainFrame.setVisible(**false**);

JFrame winScreen = **new** JFrame("Congratulations!!");

winScreen.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

winScreen.getContentPane().add(**new** JLabel(**new** ImageIcon("win\_screen.png")));

winScreen.pack();

winScreen.setLocationRelativeTo(**null**);

winScreen.setResizable(**false**);

winScreen.setVisible(**true**);

}

}

}

## **Ghosts**

//This class is extended by all ghosts, so contains all the basic necessities for ghosts.

**public** **class** Ghosts {

//Initialize Variables here. Hidden for convience of reading code.

**public** Ghosts(Container panel){

gamePanel = panel;

ghostsLabel = **new** JLabel();

ghostsLabel.setBounds(10, 10, 10, 10);

ghostsLabel.setVisible(**true**);

gamePanel.add(ghostsLabel);

}

//Scale images to fit resolution.

**protected** **void** scaleImages(){

**for**(**int** i = 0; i < ***NUMBER\_OF\_IMAGES***; i++){

Image newImage = ghostsImage[i].getImage().getScaledInstance(*scale*, *scale*, Image.***SCALE\_DEFAULT***);

ghostsImage[i] = **new** ImageIcon(newImage);

}

}

//Draw ghosts on panel.

**public** **void** drawGhost(){

ghostsLabel.setIcon(ghostsImage[ghostDirection]);

ghostsLabel.setBounds(getXIndex()\**scale* + offset, getYIndex()\**scale*+offset, *scale*,*scale*);

ghostsLabel.setVisible(**true**);

}

//Draws glowing ghosts.

**public** **void** drawGlowingGhosts(){

ghostsLabel.setIcon(ghostsImage[glowDirection]);

ghostsLabel.setBounds(getXIndex()\**scale* + offset, getYIndex()\**scale* +offset,*scale*,*scale*);

ghostsLabel.setVisible(**true**);

}

//Change direction of ghosts.

**public** **void** changeDirection(**int** newDirection){

**if**(ghostsGlow){

ghostDirection = newDirection;

glowDirection = newDirection + ***BLUE\_UP***;

} **else** {

ghostDirection = newDirection;

}

}

//Change the glow characteristics of ghosts

**public** **void** setGhostsGlow(){

ghostsGlow = !ghostsGlow;

}

**public** **boolean** getGhostsGlow(){

**return** ghostsGlow;

}

**public** String getGhostName(){

**return** ghostsName;

}

//Change the special move value, ghosts either has initiated special or hasn't.

**public** **void** setSpecialInitiated(){

ghostsSpecialinitiated = !ghostsSpecialinitiated;

}

**public** **boolean** getSpecialValue(){

**return** ghostsSpecialinitiated;

}

//Move ghosts to the desired location.

**public** **void** moveGhosts(){

**switch**(ghostDirection){

**case** 0:

setYIndex(yIndex-1);

**break**;

**case** 1:

setYIndex(yIndex+1);

**break**;

**case** 2:

setXIndex(xIndex-1);

**break**;

**case** 3:

setXIndex(xIndex+1);

**break**;

}

**if**(ghostsGlow){

drawGlowingGhosts();

} **else** {

drawGhost();

}

}

**public** **int** getDirection(){

**return** ghostDirection;

}

//Set x position of ghost

**public** **void** setXIndex(**int** xPoint){

xIndex = xPoint;

}

//Set y position of ghost

**public** **void** setYIndex(**int** yPoint){

yIndex = yPoint;

}

**public** **int** getXIndex(){

**return** xIndex;

}

**public** **int** getYIndex(){

**return** yIndex;

}

//Reset special if it is initiated.

**public** **void** resetSpecial(){

**if**(ghostsSpecialinitiated){

ghostsSpecialinitiated = **false**;

}

}

//Check if the given element of array is a wall or empty spot.

**public** **boolean** isWallHit(**int** arrayComponent){

**if**(arrayComponent == BLOCKS || arrayComponent == EMPTY\_SPOTS){

**return** **true**;

} **else** {

**return** **false**;

}

}

}

## **Yoshi**

//This handles all the characteristics of Yoshi.

**public** **class** Yoshi {

//Initialize Variables here. Hidden for convience of reading code.

**public** Yoshi(Container panel, **int** maxLives, **int** scalar){

scale = scalar;

lives = maxLives;

gamePanel = panel;

initializeArray();

yoshiLabel = **new** JLabel();

gamePanel.add(yoshiLabel);

scaleImages();

}

//Initialize Yoshi's images array.

**public** **void** initializeArray(){

ImageName[UP] = "yoshi\_up.png";

Image[UP] = **new** ImageIcon(ImageName[UP]);

ImageName[DOWN] = "yoshi\_down.png";

Image[DOWN] = **new** ImageIcon(ImageName[DOWN]);

ImageName[LEFT] = "yoshi\_left.png";

Image[LEFT] = **new** ImageIcon(ImageName[LEFT]);

ImageName[RIGHT] = "yoshi\_right.png";

Image[RIGHT] = **new** ImageIcon(ImageName[RIGHT]);

ImageName[GLOW\_UP] = "yoshi\_up\_glow.png";

Image[GLOW\_UP] = **new** ImageIcon(ImageName[GLOW\_UP]);

ImageName[GLOW\_DOWN] = "yoshi\_down\_glow.png";

Image[GLOW\_DOWN] = **new** ImageIcon(ImageName[GLOW\_DOWN]);

ImageName[GLOW\_LEFT] = "yoshi\_left\_glow.png";

Image[GLOW\_LEFT] = **new** ImageIcon(ImageName[GLOW\_LEFT]);

ImageName[GLOW\_RIGHT] = "yoshi\_right\_glow.png";

Image[GLOW\_RIGHT] = **new** ImageIcon(ImageName[GLOW\_RIGHT]);

}

//Scale Images to fit screen.

**public** **void** scaleImages(){

**for**(**int** i = 0; i < ***NUMBER\_OF\_IMAGES***; i++){

Image newImage = Image[i].getImage().getScaledInstance(scale, scale, Image.length);

Image[i] = **new** ImageIcon(newImage);

}

}

//Draw Yoshi in map.

**public** **void** drawYoshi(){

yoshiLabel.setIcon(Image[yoshiDirection]);

yoshiLabel.setBounds(((xIndex)\*scale + offset), ((yIndex) \* scale + offset), scale,scale);

yoshiLabel.setVisible(**true**);

}

//Draw Glowing Yoshi in Map.

**public** **void** drawGlowingYoshi(){

yoshiLabel.setIcon(Image[glowDirection]);

yoshiLabel.setBounds(((xIndex)\*scale + offset),((yIndex) \* scale + offset),scale,scale);

yoshiLabel.setVisible(**true**);

}

//Change Yoshi's Direction.

**public** **void** changeDirection(**int** newDirection){

**if**(!yoshiGlow){ //If yoshi isn't glowing.

yoshiDirection = newDirection;

} **else**{ //If Yoshi is Glowing.

yoshiDirection = newDirection;

glowDirection = newDirection + GLOW\_UP;

}

}

//Change yoshi's glow.

**public** **void** setYoshiGlow(){

yoshiGlow = !yoshiGlow;

}

**public** **boolean** getYoshiGlow(){

**return** yoshiGlow;

}

//Check if the array Component is a wall or empy spaces.

**public** **boolean** isWallHit(**int** arrayComponent){

**if**(arrayComponent == BLOCK){

**return** **true**;

} **else** **if**(arrayComponent == EMPTY\_SPOTS){

**return** **true**;

} **else** {

**return** **false**;

}

}

//Decrement Yoshi's Lives.

**public** **void** changeLives(){

lives = lives - 1;

}

**public** **int** getLives(){

**return** lives;

}

//Change Yoshi's Last direction specified by user.

**public** **void** setLastSpecifiedDirection(**int** newDirection){

lastSpecifiedDirection = newDirection;

}

**public** **int** getLastSpecifiedDirection(){

**return** lastSpecifiedDirection;

}

//Change Yoshi's x and y index based on the direction he is moving in.

**public** **void** moveYoshi(){

**switch**(yoshiDirection){

**case** 0:

setYIndex(yIndex-1);

**break**;

**case** 1:

setYIndex(yIndex+1);

**break**;

**case** 2:

setXIndex(xIndex-1);

**break**;

**case** 3:

setXIndex(xIndex+1);

**break**;

}

**if**(yoshiGlow){

drawGlowingYoshi();

} **else** {

drawYoshi();

}

}

//Set X position of Yoshi

**public** **void** setXIndex(**int** xPoint){

xIndex = xPoint;

}

//Set Y position of Yoshi

**public** **void** setYIndex(**int** yPoint){

yIndex = yPoint;

}

**public** **int** getXIndex(){

**return** xIndex;

}

**public** **int** getYIndex(){

**return** yIndex;

}

**public** **int** getDirection(){

**return** yoshiDirection;

}

//Add user's score.

**public** **void** addScore(**int** newPoint){

score = score + newPoint;

}

**public** **int** getScore(){

**return** score;

}

}