Bad Ice Cream 4 Flowchart

LevellLoader Class

SecTimer class

Updates the time left on a level every second

If the time is 0s

BadIceCreamFour Class

Start

runGUI()

new BadIceCreamFour()

Initialize LevelLoader()

Create JFrame, add JLayeredPanel to JFrame and perform setVisible(true) on JFrame

set JFrame default close operation

Exits when the X button is pressed in the top right of the frame

Refresher Class

Create new Refresher Object in new thread

Keeps track of time in milliseconds

Create KeyListener

Create timer which updates game elements every 1ms

if a level is loaded

call refresh() on all game elements

Create ArrayList, common Buttons, and Sound objects

Create and show password fields and play non-level music

Create button actionPerformed() listener

if password is valid

show title screen

user presses button

pwdtotitle

totitle

what is the button command?

1 to 10

restart

tolevelselect

tocredits

tohowto

togglesfx

togglemusic

Show Level Selection screen

Show Credits Screen

show How To Screen

toggle the sfx enabler boolean flag

sfx will play/not play until the next toggle

toggle music enabler boolean flag

music will play/not play until the next toggle

set Level Number

begin reading text file and start recording at the line after the level number

read level tile dimensions and get list of fruits to acquire

load ArrayList of Tile objects, including the playable character

load ArrayList of Monster objects

load ArrayList of Fruit objects

Create secTimer object

add Tile objects, including playable character, to the frame

add Monster objects to the frame

add the next batch of fruit objects, or the first batch if the level is newly started

if the last batch has been finished

false

set number of fruits left to collect in the batch

show win display

show loss display

level number = 11

If the inputted secret code is valid

secret

Fruit Class

Initialize Fruit object with image, position, and size

detectWhenPicked() - checks if an ice block is covering a fruit

if a character is occupying the same tile as the fruit

remove fruit from Fruit ArrayList and window.

decrease the number of fruits left in the batch by 1

if there are no fruits left in the batch

checkIsIced() - checks if an ice block is covering a fruit and updates the fruit graphic accordingly

If the state the fruit being covered by ice has changed

if ice is now covering the player

true

false

coveredIce() - change the fruit's graphic and mobilize the fruit if applicable

uncoverIce() - change the fruit's graphic and immobilize the fruit if applicable

extends

empty refresh() function

Grapes Object

Lemon Object

Orange Object

Watermelon Object

Peach Object

refresh() - changes position randomly every second

refresh() - moves randomly and melts adjacent ice blocks

refresh() - moves randomly quickly

refresh() - moves randomly moderately fast

Avocado Class

Chili Class

Kiwi Class

Strawberry Class

Tile Class

Initialize Tile object with image, position, and size

extends

IceTile object

TransparentTile object

empty refresh function

Character Class

Character object

isDead() - the player is removed from the Tile ArrayList and a boolean flag marks the user dead

refresh() - increments the character horizontally or vertically with each refresh, causing the character to traverse tiles until the key is let go.

moveTilesHor(boolean isRight, double incr) - sets the horizontal increment for the character to move each time the refresh() function is called and makes changes accordingly in the Tile ArrayList

moveTilesVer(boolean isUp, double incr) - sets the vertical increment for the character to move each time the refresh() function is called and makes changes accordingly in the Tile ArrayList

KeyAction Class

faceDirection() - does not cause the character to move, but changes the direction being faced.

user presses key

If key pressed was an arrow key

true

false

If key pressed was Space

If the key pressed is up or down and the player is alive

If the key pressed is left or right and the player is alive

If the key has been held for over 50ms and the player's movement is not obstructed by ice

Monster class

destroyIce() - removes an ice block in front of the monster from the Tile ArrayList and the window

checkForKill() - checks if the player and the monster reside on the same tile

moveTilesHor(boolean isRight, double incr) - sets the horizontal increment for the character to move each time the refresh() function is called and makes changes accordingly in the Tile ArrayList

moveTilesVer(boolean isUp, double incr) - sets the vertical increment for the character to move each time the refresh() function is called and makes changes accordingly in the Tile ArrayList

Initialize Monster object with image, position, and size

extends

If the player has come into contact with the monster object

remove player from Tile ArrayList and window

refresh() - moves about randomly

refresh() - moves about randomly

refresh() - alternates between moving about randomly and burrowing underground and following the player every 5 seconds

refresh() - follows the player

refresh() - moves about randomly and can break one ice block in front of itself

refresh() - moves about randomly and can jump over ice blocks

Troll Object

Cow Object

BlueCow Object

YellowCow Object

OrangeSquid Object

Egg Object

Icing Class

destroy ice until a non-ice tile has been reached or the edge of the level has been reached

true

If there is ice directly adjacent to the player in the direction they're facing

Create a new Icing Thread

false

Create ice until a non blank-tile has been reached or the edge of the level has been reached

Create interface Update which contains the function refresh() - each fruit, monster, and character has their own unique refresh() function

Modifications

There are a few elements of the completed game that deviate from the specifications outlined in the Detailed Proposal.

The game's title is not "Dairy Free". It is simply "Bad Ice Cream 4".

Only one text file is used to load each level.

The monsters are not vegetables. They are monsters directly taken from the actual "Bad Ice Cream" series. Some of the monsters' abilities mentioned in the Proposal have also been modified.

The time limits on the levels do not range from 180 – 600 seconds. They start at 180 seconds and go to 345 seconds.

There is no scoring. There is also no multiplayer, though the keyEvent and program structure is able to support multiplayer. The feature just hasn't been implemented.

There is no wi-fi component, nor is there a LAN component.

There is no Arduino controller.

There are no save file features,

Testing and Integration

1. Syntax Error

The following code in the LevelLoader class produces a syntax error:

```
for(int h == 0; h < fruitIndicator.size(); h++) {

	fruitIndicator.get(h).setCenterX(50);
	fruitIndicator.get(h).setCenterY(90*(h+1) + 45);
	gamePane.add(fruitIndicator.get(h), 1);
}
```

The two equal signs at `int h == 0` perform a boolean comparison, rather than setting h to 0. Using only one equal sign corrects this issue:

```
for(int h = 0; h < fruitIndicator.size(); h++) {

	fruitIndicator.get(h).setCenterX(50);
	fruitIndicator.get(h).setCenterY(90*(h+1) + 45);
	gamePane.add(fruitIndicator.get(h), 1);
}
```

2. Runtime Error

The following code in the BadIceCream4 class produces a runtime error, and more specifically, NullPointerException:

```
public void setUpdateListener() {

	updateListener = new ActionListener() {

		@Override
		public void actionPerformed(ActionEvent e) {

			keyAction.useKeys();

			for(int i = 0; i < Constants.Y_TILEROWS; i++) {
				for(int j = 0; j < Constants.X_TILECOLS; j++) {
					lvlLoader.getTileList().get(i).get(j).refresh();
				}
```

```
                        }

                        for(int j = 0; j < Constants.numFruit; j++) {
                              lvlLoader.getFruitArray().get(Constants.fruitLvl).get(j).refresh();
                        }


                        for(int k = 0; k < lvlLoader.getMonsterList().size(); k++) {
                              lvlLoader.getMonsterList().get(k).refresh();
                        }

                        lvlLoader.getTimer().refresh();
                  }
            };
      }
```

This method is called by a timer to update the game every 1ms. However, the ArrayList called by getTileList() is empty when the game starts. Forcing the program to cycle through i*j elements in the Tile ArrayList obviously will not work. The ArrayLists are only filled when a level is loaded. Having a boolean flag that indicates when a level is loaded solves this problem:

```
public void setUpdateListener() {

            updateListener = new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent e) {

                        if(Constants.levelLoaded) {

                              keyAction.useKeys();

                              for(int i = 0; i < Constants.Y_TILEROWS; i++) {
                                    for(int j = 0; j < Constants.X_TILECOLS; j++) {
                                          lvlLoader.getTileList().get(i).get(j).refresh();
                                    }
                              }

                              for(int j = 0; j < Constants.numFruit; j++) {
                                    lvlLoader.getFruitArray().get(Constants.fruitLvl).get(j).refresh();
```

```
                        }


                        for(int k = 0; k < lvlLoader.getMonsterList().size(); k++) {
                              lvlLoader.getMonsterList().get(k).refresh();
                        }

                        lvlLoader.getTimer().refresh();
                  }
            }
      };
}
```

3. Logic Error

To center the level, two constants X_OFFSET and Y_OFFSET are used to make sure that the left and right edges of the level are equidistant from the left and right edges of the window, and that the same applies to the top and bottom edges of the level. Below is code from the Tile class which handles the constants:

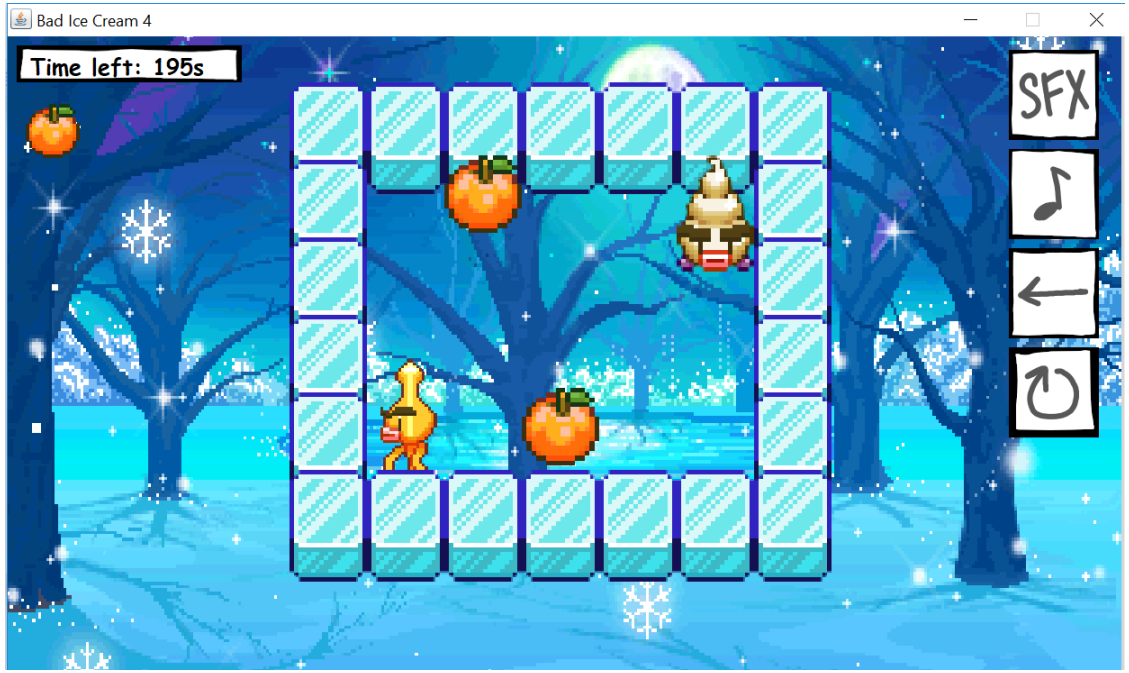
```
public void setCenterX(double xCoord) {

      centerX += xCoord + (int)Constants.X_OFFSET;
      setX((int) (centerX - getWidth()/2.0));
}

public void setCenterY(double yCoord) {

      centerY += yCoord + (int)Constants.Y_OFFSET;
      setY((int) (centerY - getHeight()));
}
```

Loading the level, the tiles are out of place and enemies move off-screen:

SetCenterX is repeatedly called every 1ms to update the locations of enemies, fruits, and characters when they move. Because it continually adds the constant to the coordinate instead of just once, the x and y coordinates of moving objects will act haphazardly. To remedy this, the constants are used only during the initialization of each tile, and during the goal setting of each tile, where the program determines where the tile should move to. Code starts on the next page.

```
	public void setCoordinates(boolean isFruit) {

		if(isFruit) {
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)(tileY*Constants.tileSize + Constants.tileSize/2.0 + Constants.Y_OFFSET));
		} else {
			setCenterX((int)(tileX*Constants.tileSize + Constants.tileSize/2.0 + Constants.X_OFFSET));
			setCenterY((int)((tileY + 1)*Constants.tileSize + Constants.Y_OFFSET));
		}

		setBounds((int)x, (int)y, getWidth(), getHeight());
		setRefresh(true, 0);
		goalX = getCenterX();
		goalY = getCenterY();
	}

	public void setCenterX(double xCoord) {

		centerX = xCoord;
		setX((int) (centerX - getWidth()/2.0));
	}

	public void setCenterY(double yCoord) {

		centerY = yCoord;
		setY((int) (centerY - getHeight()));
	}

	public void setGoalX(double xGoal) {
		goalX = xGoal + (int)Constants.X_OFFSET;
	}

	public void setGoalY(double yGoal) {
		goalY = yGoal + (int)Constants.Y_OFFSET;
	}
```

The fixed game looks like this:



Game Outcomes

Win:

Loss:

BAD ICE CREAM 4

user's manual

contents

introduction

Bad Ice Cream is a tile-based browser game where players, who control ice-cream characters, complete levels by collecting fruit while avoiding monsters. Some fruit are stationary, while others move about randomly. Likewise, some monsters move very simply, while others may actively chase the player. Players are aided by their ability to shoot and break rows of ice blocks to manipulate fruits or impede monsters. However, if a monster touches the player, they die, and the level must be restarted. Each level is also timed, and the player is required to complete that level before time runs out. If the player has collected all the fruits in a level, has not been touched by a monster, and has completed all tasks within the time limit, they are considered to have won the level. Otherwise, they are considered to have lost the level.

This assignment is a clone of the Bad Ice Cream game, titled Bad Ice Cream 4 The monster, fruit, and tile graphics are taken directly from the game, and are not my own. The code, however, is my own.

booting

Upon booting the program, the user is greeted with a prompted for a password. The password is **100%**. Enter **100%** into the text field and press the **Check Password!** button to start the game. Anything other than **100%** will not cause anything to happen.

title screen

After entering the correct password, the title screen will show, with **Bad Ice Cream 4** displayed in the middle of the screen.

fig 1. title screen

In the top right, there is a **music button** and an **SFX button**. They are used to turn music and sound effects, respectively, on and off.

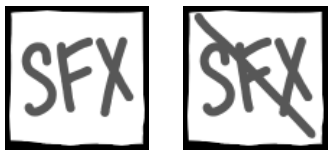


fig 2. SFX on and off

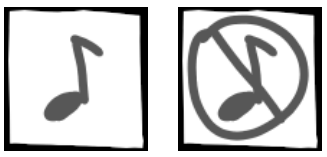


fig 3. music on and off

title screen

Underneath the Bad Ice Cream 4 logo, there are three buttons. The **Play** button on the left takes the user to the **Level Selection** screen. The **?** button in the middle takes the user to a **How To** screen with basic instructions on playing the game. The **CRED** button takes the user to the **Credits Screen**.

fig 3. Play, ?, and CRED buttons

how to screen

The **How To** screen shows a graphic with basic game instructions. The top right corner of the screen contains a left-facing arrow **Back** button in addition to the **SFX** and **Music** buttons. The **Back** button takes the user back to the **Title Screen**.

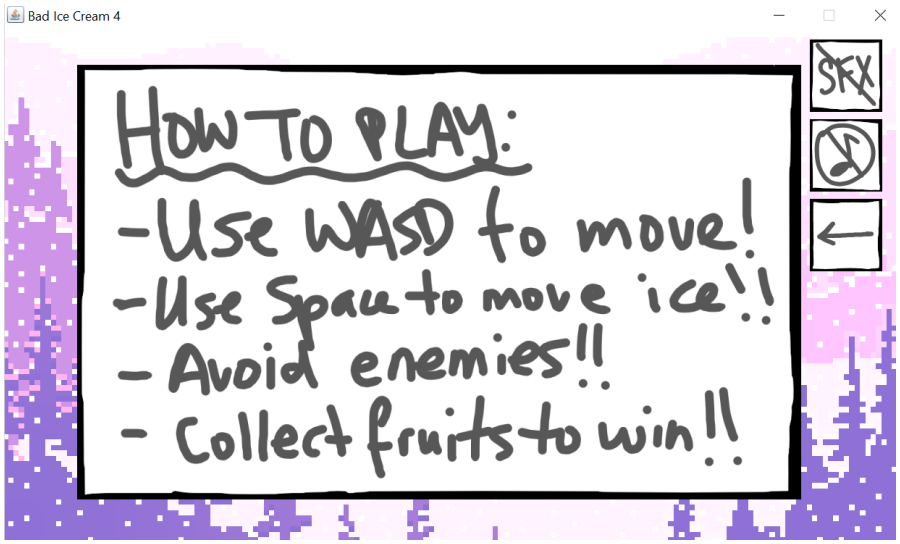


fig 4. How To Screen

credits

The **Credits Screen** shows a credits graphic. Like the **How To Screen**, the top right contains a left-facing arrow **Back** button in addition to the **SFX** and **Music** buttons. The **Back** button takes the user back to the **Title Screen**.

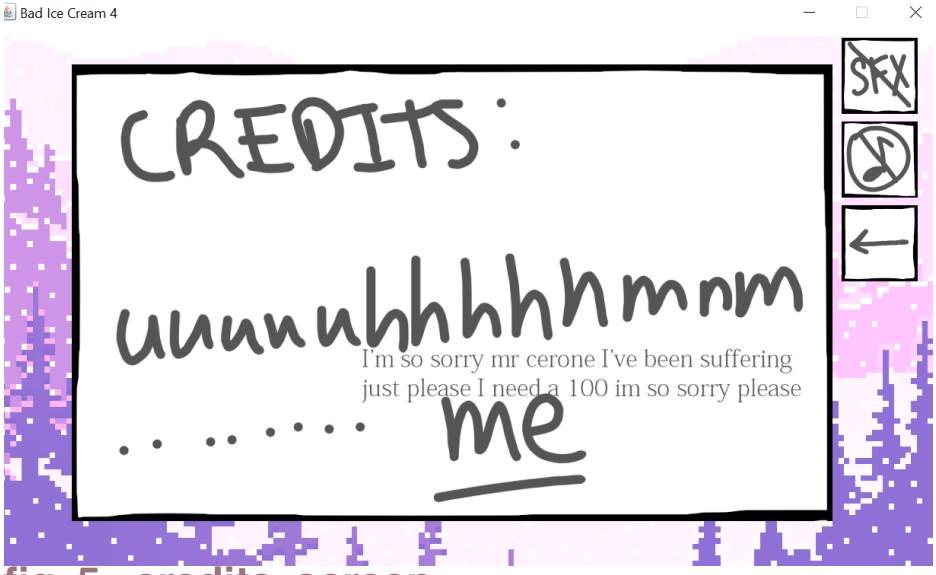


fig 5. credits screen

level selection

The **Level Selection Screen** contains 10 numbered buttons that take the user to the respective levels they indicate, as well as a field at the bottom for a **secret code**. When the user enters a code into the field and presses the **SEARCH** button, the program will search through its list of secret codes, looking for a match with what the user has entered. If what the user has entered is a secret code, they will be taken to a secret 11th level.

Valid secret codes are: **cerone i$ awesome**, **ICs$u roXXX**, or **PIS Assign 1 0 0**.

level selection

The **Level Selection Screen** also includes the **SFX** and **Music** buttons, as well as a back button which takes them back to the title screen

fig 6. level select screen

game screen

The **Game Screen** features a **timer** in the top left corner that indicates how long the player has left to finish the level.

Directly underneath the **timer**, there is a list of all the types of fruit the player needs to collect. After all of the first type of fruit has been collected, its icon will shrink and become grayscale. The next fruit on the list, if there is one, will then appear in the level and become collectable. This process repeats until all fruits have been collected and the player has won by extension. If the player touches a monster or the timer runs out, the level will immediately end.

game screen

In the top-right corner, there is an **SFX** button, **Music** button, and a **Back** button, which takes the user back to the **Level Selection** screen. There is also a **Replay** button, which allows the user to restart the current level.

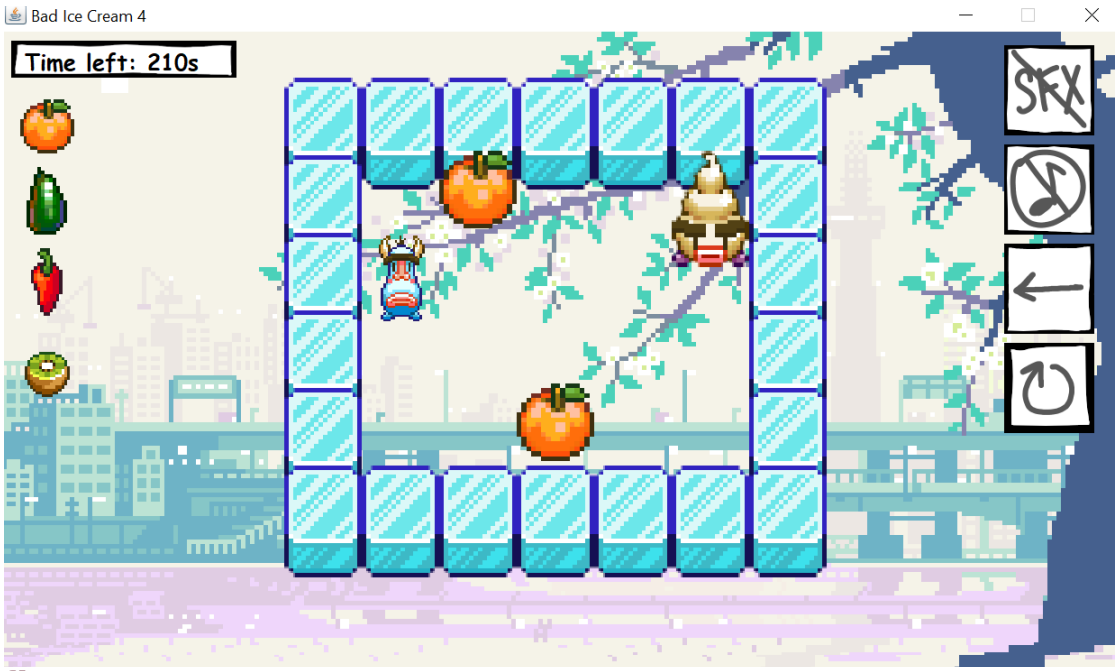


fig 7. game screen

gameplay

Each level features a grid framework of ice blocks, fruits, and monsters on top of a unique animated GIF background. The player moves around using the **Arrow** keys. Tapping an **arrow** key briefly will not cause the player to move, but will cause the player to face that direction. Note, however, that the character will still appear to be facing towards the player. Users must keep in mind what direction they are facing.

gameplay

Players can also shoot or destroy rows of ice blocks using the **space** key. If the player is not directly facing an ice block, pressing **space** will cause a row of ice blocks to be formed. Formation will continue in the direction being faced until another ice block is reached, a monster happens to be in the way, or the level's boundaries have been reached. Any fruit caught along the way will be trapped in the ice. Fruit cannot be collected while trapped in ice.

If the player is directly facing an ice block, pressing **space** will cause a row of ice blocks to be destroyed. Destruction will continue until a non-ice tile is reached, or the level's boundaries are reached. Fruit trapped in ice will then be released and become collectable.

fig 8. space is pressed twice while player faces down

fruit

There are many different types of fruits that may appear, some of which have unique properties.

Lemons, **oranges**, **peaches**, **watermelon**, and **grapes** are all stationary fruit. They do not move.

fig 9. the stationary fruits

Strawberry move about randomly at a moderate pace.

Kiwi move about very quickly and randomly.

Chili move about randomly and melt adjacent ice blocks on all sides. They can only remain trapped in an ice block for two seconds before breaking free.

Avocado change position randomly every second.

fig 10. the dynamic fruits

monsters

There are many different types of monsters that may appear, each with its own unique set of properties.

Green Trolls move about slowly. They move in a straight line until they meet an ice block. If the troll was originally facing right, it will try to turn right. Otherwise, it will try to turn left. If that is not possible, it will try to go up. If that is still not possible, it will stay still. Similarly, the order of precedence for a troll that was originally going up is left, right, down, then no movement. That of a troll originally going left is down, up, right, then no movement. That of a troll originally going right is up, down, left, no movement.

Cows move about slightly faster, albeit completely randomly.

Eggs move about randomly, but can jump over ice blocks, though only one ice block. If there are two or more in a row, an **Egg** cannot jump.

Orange Squids move about randomly, and can destroy an adjacent ice block in the direction they're facing.

Yellow Cows always follow the player.

Blue Cows alternate between moving randomly and burrowing underground and following the player every five seconds. When they are burrowed underground, a yellow dot is used to denote their location to the player.

fig 11. monsters

endgame

When the user has won, a display in the middle of the screen indicates to the player a win. If the user is not on the last level, a right-facing **Next** arrow appears below which allows the user to move onto the next level. The user may also restart the level, or go back to the **Level Selection Screen**.

fig 12. win screen

If the user loses, a display in the middle of the screen indicates to the player a loss. The user may restart the level, or go back to the **Level Selection Screen**.

fig 13. loss screen

bibliography

resize image in java. (n.d.). Retrieved from https://stackoverflow.com/questions/5895829/resizing-image-in-java

Convert color image into black and white using java created from scratch. (n.d.). Retrieved from https://stackoverflow.com/questions/41054525/convert-color-image-into-black-and-white-using-java-created-from-scratch

Playing Sounds in Java. (n.d.). Retrieved from https://www.cs.cmu.edu/~illah/CLASSDOCS/javasound.pdf

How to Play an Audio file using Java. (n.d.). Retrieved from https://www.geeksforgeeks.org/play-audio-file-using-java/

Java – Multithreading. (n.d.). Retrieved from https://www.tutorialspoint.com/java/java_multithreading.htm

Multithreading in Java (n.d.). Retrieved from https://www.javatpoint.com/multithreading-in-java

Bad Ice Cream – Nitrome Wiki. (n.d.). Retrieved from http://nitrome.wikia.com/wiki/Bad_Ice-Cream