

Sushi Rush

Nicole Ajoy

Version: 2.0

# Note from the Creator

## Intro

Hello! Thank you for downloading my game, *Sushi Rush*! This game currently runs on Processing version 3.X (library imports should work on any Processing 3.X versions).

## Library Imports

Before running the game, please make sure to download the libraries listed as they are imported into my program! They are necessary to properly run the game and to visualize everything.

|  |  |
| --- | --- |
| Library | Website |
| GifAnimation | https://github.com/akiljohnson1/GifAnimation  Download and unzip the “gifAnimation.zip” folder > put the extracted folder in the following directory:  **<PathToProcessingSketchFolder>/libraries** |
| Minim | http://www.cs.cmu.edu/~jar/cn21.pdf  Download from Processing > Sketch > Add Library > “Minim” |

# Game Description

## Overview

The genre of *Sushi Rush* is arcade/strategy game that follows the general game mechanics of the popular game, *Fruit Ninja*.

## Objective

The objective is to slice down as many pieces of sushi as possible, collecting enough points while avoiding bombs, to eventually beat the high score.

## UI/Controls & Gameplay

**TITLE SCREEN** There is currently one level implemented that can be set to an **EASY** or **HARD** difficulty from the title screen. By default, the level is set to **EASY**. Once the user is satisfied with the difficulty setting, they can select the **START** button to start the game (GAMEPLAY SCREEN). The user can exit the program by clicking on the **QUIT** button on the title screen. Likewise, the program can be terminated at any point by pressing the **ESC** key.

**GAMEPLAY SCREEN** In the gameplay screen, there is an **X** button on the upper left corner for when the user wishes to return to the title screen. The main UI is located on the top of the screen, including the current score, high score, and health (or lives). Both sushi and bombs will be thrown, allowing the player to chop (or avoid) them. Bombs must obviously be avoided and are worth -5 points (deduct from current score and remove one life). Sushi pieces are each worth 10 points (add to the current score). There are currently only two gameplay modes:

1. In **EASY** mode, the player begins with three lives. The speed of sushi and bombs is relatively slow, and there are no available powerups.
2. In **HARD** mode, the player only has one life. The speed of both sushi and bombs doubles, and powerups are made available. There are two powerups: ICE (slows down the game speed for 6 seconds) and STAR (worth twice a sushi’s value).

**HOW-TO-PLAY** The player must **click**, **hold**, and **drag** their mouse across the screen. If the mouse touches a sushi or a bomb, a collision is detected. If a sushi collision is detected, points are added to the current score, and the piece is cut in half. If a bomb collision is detected, points are deducted, and a life is lost. In both cases, the sushi/bomb becomes transparent to show that the collision was counted. But the game continues! The game goes on until one of the following:

1. Player loses their last life
2. The **X** button is clicked

**GAMEOVER SCREEN** This screen is obtained only when all lives are lost. Simply click the X button in the upper right corner to return to the title screen and try again!

# Development Difficulties

## What & How They Were Overcame

**TRACKING SCREEN LAYOUTS** I spent quite a decent amount of time on the screen layouts and tracking which screen (Title, Level, End) is active. I remembered a wise saying from one of my programming fundamentals professors: “If you are stuck, chances are you need to add another variable.” Surely, after adding a few more Boolean variables and if-else statements, I was able to solve all my problems.

**SPECIAL PROCESSING FUNCTIONS** Another difficulty I encountered was dealing with the **mousePressed()** and **mouseDragged()** functions when it came to button clicks and actual gameplay mechanics (mouse dragging/holding). My original implementation involved using the **mouseClicked** Boolean variable. The issue with this is that when I used it, it would count all the frames that the mouse was held, and my program would either keep adding values to the total score multiple times, or it would take away all the health/lives with just one bomb collision. To fix this, I changed the implementation to include **mousePressed()** and **mouseDragged()** functions and rearranged several lines of code.

# Resources

## Assets

|  |  |
| --- | --- |
| File (main/data folder) | Website |
| AUDIO | |
| audio\_bg.mp3 | https://opengameart.org/content/chiptune-loops-0 |
| audio\_chop.mp3 | https://www.fesliyanstudios.com/play-mp3/1215 |
| FONTS | |
| font\_gameplay.tff | https://www.dafont.com/gameplay.font |
| font\_rainyhearts.tff | https://www.dafont.com/rainyhearts.font |
| IMAGES | |
| bg\_end.png | http://i.imgur.com/V35BBN0.png |
| bg\_level.png | https://www.redbubble.com/people/stevenfholmes/works/27901396-pixel-art-wood-planks |
| bg\_title.png | https://twitter.com/woostarspixels/status/902548037226172416 |
| Image\_bomb.gif | http://pixelartmaker.com/art/cf8800006625ade |
| image\_cat.gif | made by me |
| Image\_paw.png | made by me |
| Image\_sushi.png (all sushi images) | https://www.deviantart.com/theskywaker/art/Sushi-Pixel-Art-630909109 |