Polygon and point utilities

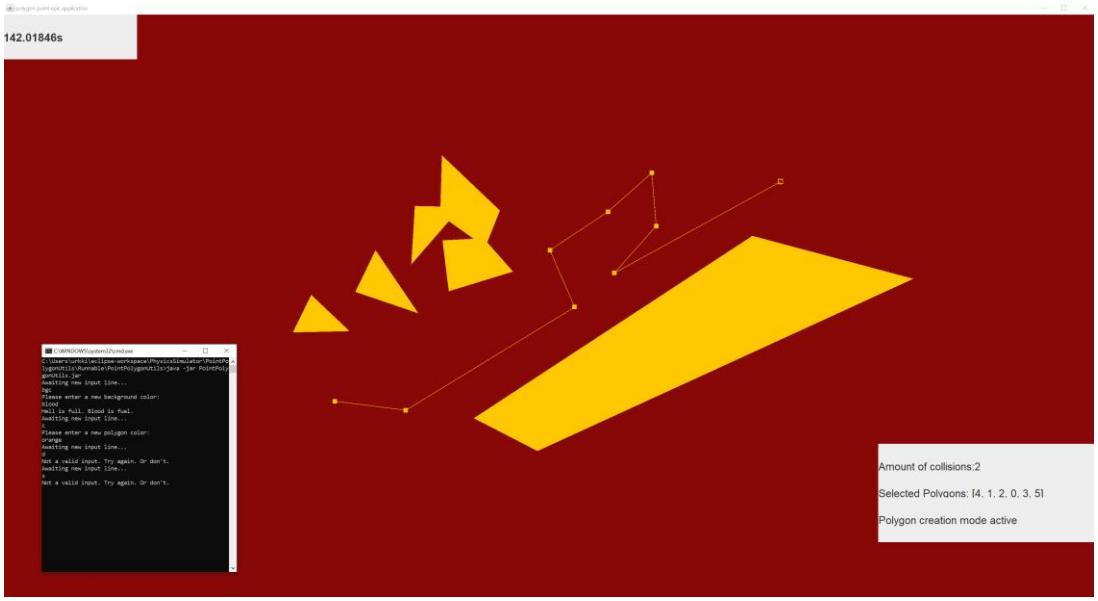

Table of Contents

Target assessment grade of this assignment is 3.

Specification

The application was written in java with the help of java swing, but includes custom polygon rendering and creation, lots of math and a cool interface. Master time, collision count, selected polygons and polygon creation state are stated in the UI. The window is scaled by screen resolution and supports scaling inside the original bounds of the window.

For windows, there is a run.bat containing an auto-run of the application (“~/Runnable/run.bat”) as also a java jar for the application. I strongly recommend using this jar, as the cmdlet on any OS is way better than the IntelliJ default console window. In the case that you run this on a Linux etc., you’ll need to invoke the jar via command prompt. I have not tested the program on Linux, as my Virtual Box boogied its way into dead code land. Also, the java jar is runnable, but **do launch it from a command prompt**, as the application is pretty heavily a command-line application.

Correctness and exception handling

Due to the complex nature of the application, there probably isn't an exception left from the possibilities. However, the most common exceptions are Typecast exceptions (for example, from String to Integer) which are handled with a try/catch. More exceptions, such as UnsupportedAudioFileException (audio file of type, for example, .mp3) are handled with try/catch.

Also, when searching for a file with a file path, FileNotFoundException and LineNotAvailableException are processed the same way. Other used-to-be-common exceptions in the application were ArrayIndexOutOfBoundsExceptions, which are ignored by the application in most cases due to runtime exception handling in the application.

Resource management

When inputting the command "Sing me a song"/ "Play for me" / "Play me the most beautiful piano piece" for the first time, a .wav size of a tremendous size is loaded and played (surrounded by try/catch). After inputting this again, the audio file and resource stream are closed because they aren't needed anymore, and the pianist inside your computer gets mad at you.

Runtime, polygons and other data are stored in ArrayLists often of type ArrayList<Polygon> for quick access of rendering and mathematical operations. This storage is not persistent.

UI

Master time (top left) – Time elapsed since creation of first polygon

13.139915s <-

Amount of collisions – amount of collisions happening **between the latest polygon and any other polygon on screen**. (Sometimes, two collisions need to be evaluated, thus showing 1-2 collisions per collision)

```
Amount of collisions:3 <-
```

Polygon creation state – displays if you are creating a polygon or not.

```
Polygon creation mode active <-
```

Selected polygon(s): Displays the index(es) of polygon(s) that you have selected.

```
Selected Polygons: [2, 1, 0]
```

Basic controls

G – toggle polygon creation mode

For the following actions, you must not be in polygon creation mode, which you are by default when launching the application.

Click and drag – move all the polygons via mouse.

Click on one polygon or multiple polygons consecutively with shift pressed – **select polygon(s).** The index(es) of the selected polygon(s) can be seen under “Selected Polygons”.

Selected Polygons: [1, 2, 0] <as so.

Things to do with selected polygons:

WASD – Move polygon(s).

Left arrow, Right arrow – Rotate polygon(s)

(rotations look really janky. Forgot to fix this.)

P - **Enter command input mode on command line**

(after creating first polygon. Interrupts the program when waiting for command input. Press multiple times for multiple consecutive operations. (“It’s not a bug, it’s a feature!”))

Commands

All Strings of commands are case-sensitive. If no command match was found, doesn’t do anything.

“Debug”, “toggle the got damn debugs!”, “enable debug plz”, etc.– enables printing of all the major mathematical and physics operations with a test value. See all debugged values and descriptions below:

```
MathOperations:
  Vector Length (int and float param): 8.13941 : 1.4142135
   Vector dot product: -16.0
    Length of vector cross product: -5.139642831806814
   Angle between vectors: 1.0000001rad / 40.514233°
   SmallestIntegerInArray: 0
   ConcatIntArray: [2, 10, 3, 2]
   Closest value to number from list:0.0
   Radians to angles: 305.335°
   Abstract sign of quadrant: 1
   Line intersection: [false]
   Depth of point polygon penetration: 0.0
   Closest two points: [[1920, 1080], [2, 3]]
Polygon holder:
  Base Polygons: 2
  First shape stats: [586, 542, 558], [573, 492, 305], 3
Physics:
    Collision (SAT): true
    Collision (repetitive point check): true
    Is polygon concave: false
Transform:
    Accurate cumulative float overflow: [[0.0, 0.0], [0.4001756482213068, -0.014444002855555027], [0.0, 0.0]]
```

“EXIT”, “/E”, “guh bye” – exits the application via a goodbye.

"Sing me a song", "Play for me", "Play me the most beautiful piano piece” – A pianist inside your computer starts playing Claude Debussy’s “Clair de Lune”. Whatever you do, do not dare to interrupt him.

"background", "bgc" – prompts you for a new background color from a pre-defined list. If no match is found, does nothing. For a custom color, type **"custom”.** The program will prompt you for R, G, and B values. Type “random” for a random color. Warns you if polygon color and background color are the same.

"color", "c" – prompts you for a new polygon color from a pre-defined list. If no match is found, does nothing. For a custom color, type **"custom”.** The program will prompt you for R, G, and B values. Type “random” for a random color. Warns you if polygon color and background color are the same.

"ConcaveDebug", "concave", “GEBUERJEIT” – Enables printing of concave checks and rendering of normals for polygon edge vectors, which are used for finding angles via dot product.

"FPS", "fps", “limitFPS”, “limit” – prompts the user for a new minimum time to wait between frames in milliseconds. (default = 18ms). Handy for combatting rendering issues on weaker computers. Warns you about rendering issues if you set the delay to less than 10. If the value is 0, nothing happens, and slanders the user If they typed a non-integer string as a prompt.

"indexes", "index", “polygonIndex” – toggles drawing of polygon indexes on top of polygons. Disabled by default, as this is performance heavy due to the way java.swing canvases are layered. Can help when selecting polygons.

"rainbow", "lsd" – toggles rainbow colors for polygons and background. Resource heavy.

"rect", "rc", "rectangle" – creates a randomized rectangle with prompt “random” or a rectangle with custom dimensions, position, and rotation.

"randomcolors", "randomtheme", "rndcolors", "rndtheme" – sets the primary and background color as random.

"scale" - scales a polygon a uniform or non-uniform amount according to the polygon index. You may need to toggle polygon indexes via e.g. **"index"** to find the index of the polygon you are trying to transform.

Cool algorithms used in the program

This contains only a very few of the algorithms used, but these are the most notable ones. They keep the program afloat. I hope.

Winding number algorithm – used to detect If a point is inside a polygon or not by the winding numbers sign determined by what direction polygon edges “wrap” around a point. Inside MathOperations and used extensively to detect if polygons intersect.

Concavity check – a polygon is concave, if any one of its interior angles is more than 180 degrees. This is checked by iterating through all the edges of a polygon and checking the dot product between the next edge vector and the normal vector of the previous vector. If the dot product is less than zero, the angle is more than 90 degrees -> more than 180 degrees in the interior of the two edges -> polygon is concave.

Separating axis theorem – a collision detection system for convex polygons. It bases this on the fact that if two polygons are projected as a “shadow” onto an axis and rotated in respect to the x and y-axis, their collision can be determined by their projection overlap at any point of the rotation. If the shadows overlap at any point of the rotation, the polygons collide/intersect.

Test case

A bunch of red polygons with all selected and indexes enabled via “indexes”. Debug is enabled and working via “enable the got damn debugs!” Selected polygons can be moved and rotated accordingly, and the master move via mouse drag works. The background is set via “bgc” -> “gold” and can be changed. The debug value printing is interrupted when P is pressed again, toggling command line input.