# Odyssey Now! v0.2.6

(See v0.2.3 for explanation of Version Numbers. Note v0.2.5 doesn’t exist due to laziness.)

# Launching Odyssey Now!

1. Double click BUILD.
2. Double click LAUNCH.

If this doesn’t work, you likely don’t have the Java Runtime Environment installed.

The Command Prompt may appear while this occurs. This is normal.

You may get security warnings from Windows saying it blocked the run. I can tell you they’re safe, but of course it’s your choice if you believe it or not.

# Running From the Command Line

I wanted to make sure that developers could still compile and run the code from the command line if they so choose, so I ensured this compatibility remained inside the code, but due to the JSSC problem outlined below, you’ll need to explicitly state the dependency during compilation and running, in the style of:

javac -cp "./jssc.jar;" ODSYRunner.java

java -cp "./jssc.jar;" ODSYRunner

# TO DO:

The compatibility of Odyssey Now! with Mac and Linux is under evaluation. Ideally, this project should run on all three operating systems. The operation of the program while the controller was plugged in was examined, and needs improved on.

Look into increasing the drawspeed of the program to paint more quickly and reduce flickering. Implementing some sort of DoubleBuffering or BufferStrategy might be useful for this.

We may also be able to reprogram each object’s paint function to only repaint the necessary area, instead of repainting the whole screen each time.

# Modifications from v0.2.4

Arduino

More modifications to the microcontroller were mad, plus the most recent files were added to the GitHib folder for easy access. The Arduino file is now programmed to ignore small (3 or 4) fluctuations in input, which greatly reduces the level of jitter reflected on the screen. It seems that the program still sends data even when it’s completely unchanged, which would require further work.

Multithreading

The concept of multithreading the program has been revived, with some success. Currently, the game operates in three threads: Game Logic, Player Input, and Graphics Draw (these names do not reflect the names in the code, but are rough estimates of what they do.) The first and main thread oversees the game’s logic, responding to keypresses that change the game card or ball speed and detecting if the ball collided with anything; the second thread focuses only on deriving the input from the analog controller and pushing it to where it goes; and the third thread focuses only on painting the screen. (Note: It may be beneficial to push the ball collide logic to the graphics draw, as they desync some with the current set-up.)

This seems to have helped with the lag problem, but introduced the problem that the player boxes seem to ‘teleport’ to their intended location, instead of having a smooth transition to where they belong, which resulted in the introduction of the next idea.

Intermediary Locations

The logic of the playerbox paint function was changed so that it keeps track of where the intended destination is, and draws a few points on the way there instead of drawing straight at that location.

This helped somewhat, but introduced a problem where the boxes tended to move at unnatural diagonals that didn’t reflect the true input. This led to the next patch.

Draw All Previous Inputs

Now, the code remembers all previous inputs and makes sure to draw the square hitting each one before heading to the next one, which helps round off the movements and eliminate the unnatural ‘stepping’ movements.