Martin Oka 12/6/2022

Game: Pogo Jump All work and art done alone.

Description:

The premise of this game is a simple 2d platformer where the main character is a man on a pogo stick in a medieval setting, trying to reach the star to win the game.

Disclaimer:

You need to set up java in a way that it will be able to recognize that it needs to use the gpu. If you do not do this the image will appear choppy and laggish. The utilization should be around 40% on both gpu and CPU (ik, java optimization is not great for games.) I personally have a i7 9700k and a 3070 and get around 40% on the cpu and 50% on gpu.

Starting off we have a simple repaint that runs every 2ms (which results in a 5% decrease in load compared to without any wait). **This calls the repaint function**

```
while (!player.intersects(5, 900, 1900, 20)) {
    /*-------------------------------------------------------------------------------------------------
    for people with weaker systems, you can have it check for inputs every 10 ms+,
    for an easier time on the cpu/gpu  2ms 40cpu  50 gpu and without the 2ms its runs at 45cpu 58gpu
    // */
    if ((System.currentTimeMillis() - timer) % 2 == 0) {
        repaint();
    }
```

Then we have the image splitter class which divides the player animation which is 9 separate images divided into 1 image. This class creates 9 buffered images each one referring to a different position of the pogo stick man.

Start Screen.

Then we have the code that displays the start screen, which are just some very simple image imports and drawstrings, and the start screen looks like this.

Once a mouse or keyboard event has been triggered, classes movement, mapgen and physics get called.

In **mapgen()** it chooses two random numbers which correspond to the width and height of the screen, int x1 = (int) Math.round(Math.random() * 2000); int y1 = (int) Math.round(Math.random() * 1000); adding these numbers to the array which will get displayed later on. There is also if statements that ensure that the platforms do not form in unreasonable locations, and thus increase the probability of the platforms being in key locations such as the sides, bottom and top. It also checks to see which platform is the highest, which is where the star will go.

In **movement()** it checks what the input is, if it is a valid input ie: no double jumping, and also checks if it is out of bounds. If out of bounds, it creates a new map in the next screen by setting Boolean mgen=true unless player is going to the left and map == 1 if going to the left and map=0, then stop player in tracks.

In **physics()**, if the player is going down and not touching() then the player goes down for I in range of z, so it isn't choppy and doesn't look like the player is going down -10 every 10ms, but rather 1 every 1ms. There is also a vertical transform, that works in the same way, as well as physics when the player is going up in the same way. Every if not touching() 10ms, **speed** will **increase** by .1ms.

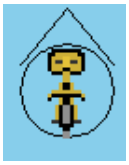
Boolean touching()

Returns true if player is touching the platform, and false if not. In here, I used two different means of detection, one which was player.intersects(….) and the other being a recursive function that ran through all of the platforms to see if the bottom of the player is touching the platform.

DISPLAYING ALL OF THIS

Finally in display, once events >= 1 and !win it displays the gif, which is just an image which changes its y every 250ms, making it the appearance of a looping gif. In this gif, the smoke from the chimney shifts upwards and the windmill runs.

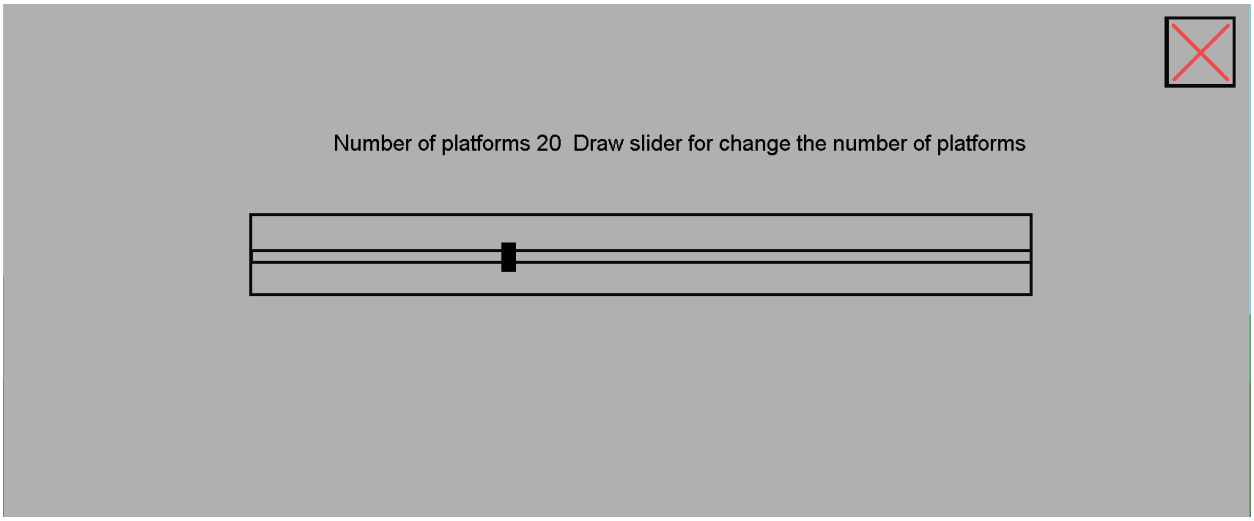
Then there is a for loop that displays all of the platforms, no matter how many there are.

If the player is above the screen then there will be a player icon pointing to where he is, and which direction he is going, by showing his animation but at the top of the screen. 

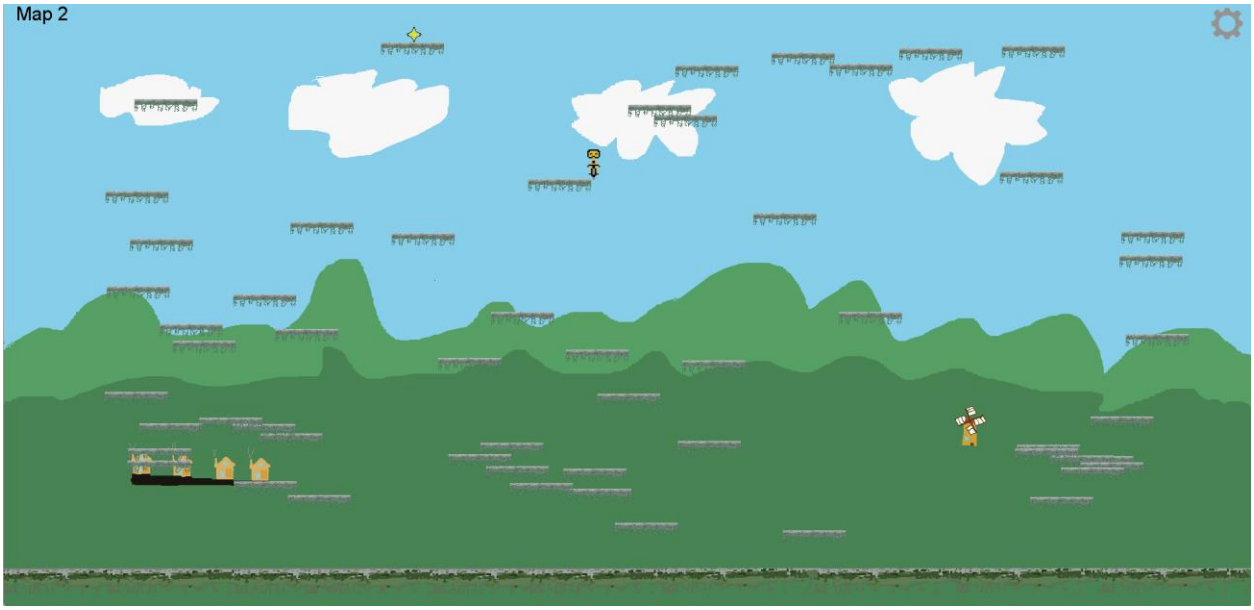
Then there is a line which tells you which map you are on, by dividing which platform you are on by the number of platforms +1. 

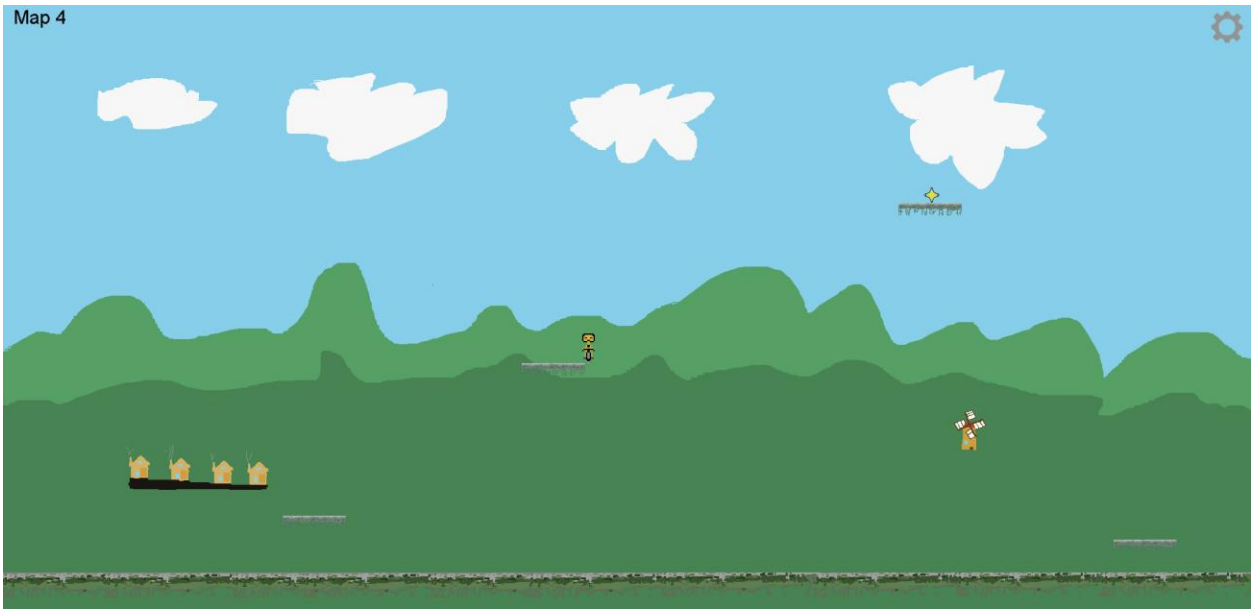
Settings

There is also a settings button market by  , which when pressed by the mouse brings up:

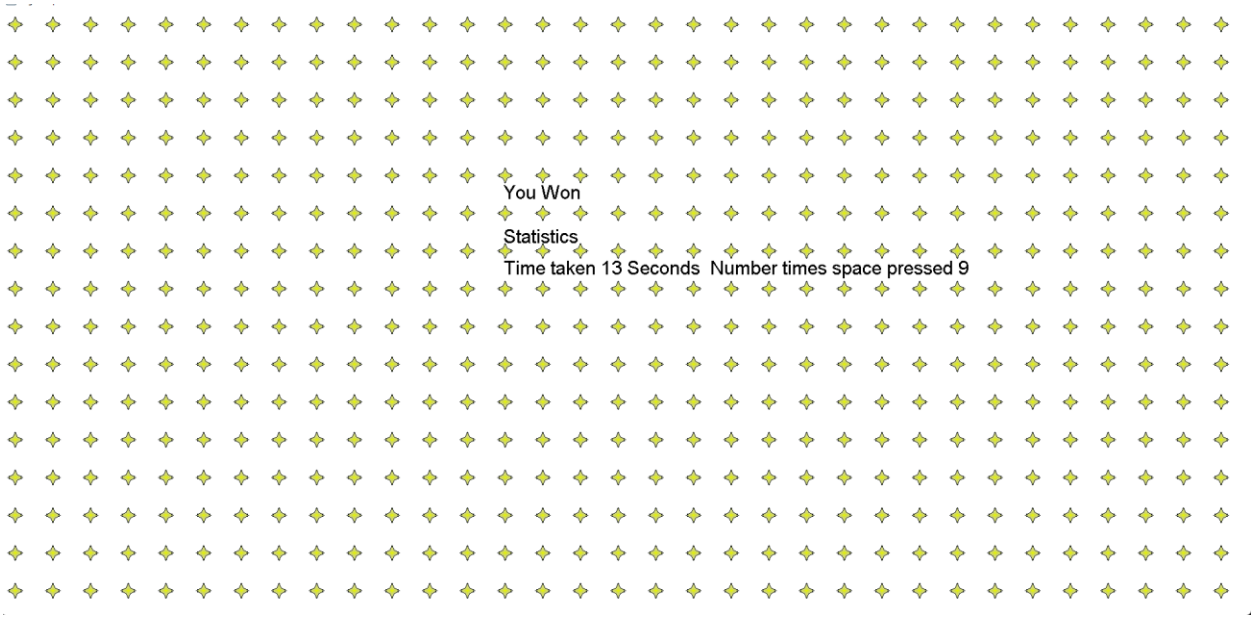


a slider which defaults to 21 platforms, which you can then chose between a max of 56 platforms and a minimum of 4 platforms to make the game easier or harder. You can then press the esc button and all of the platforms will be updated immediately. (examples shown below)





Finally there is the win condition which checks to see if the player is touching the star. If the player is touching the star, win=true, all of the other functions stop getting called, and stars get displayed as well as relevant statistics about the gameplay. (stars called recursively, 16 columns with 33 rows).



CONGRADULATIONS FOR READING THROUGH!

Hope you enjoyed my little run-through, please if you enjoyed it feel free to try out my game, though beware that it will run poorly on anything that isn't a desktop, or that doesn't utilize the gpu for java.

Again, all of these animations and graphics I made myself through a free software called piskel, and all work is my own except for some references on niche cases like importing images or sub splitting images, where I consulted outside help such as stack overflow.