Archeologitaire

This page intentionally left blank

Archeologitaire

developed by

Jason Miner

Matthew Smillie

for

CS 230

Software Engineering

Fall 2017

Dr. Harper

Carroll College

link

https://github.com/matt-xav/archeologitaire

This page intentionally left blank

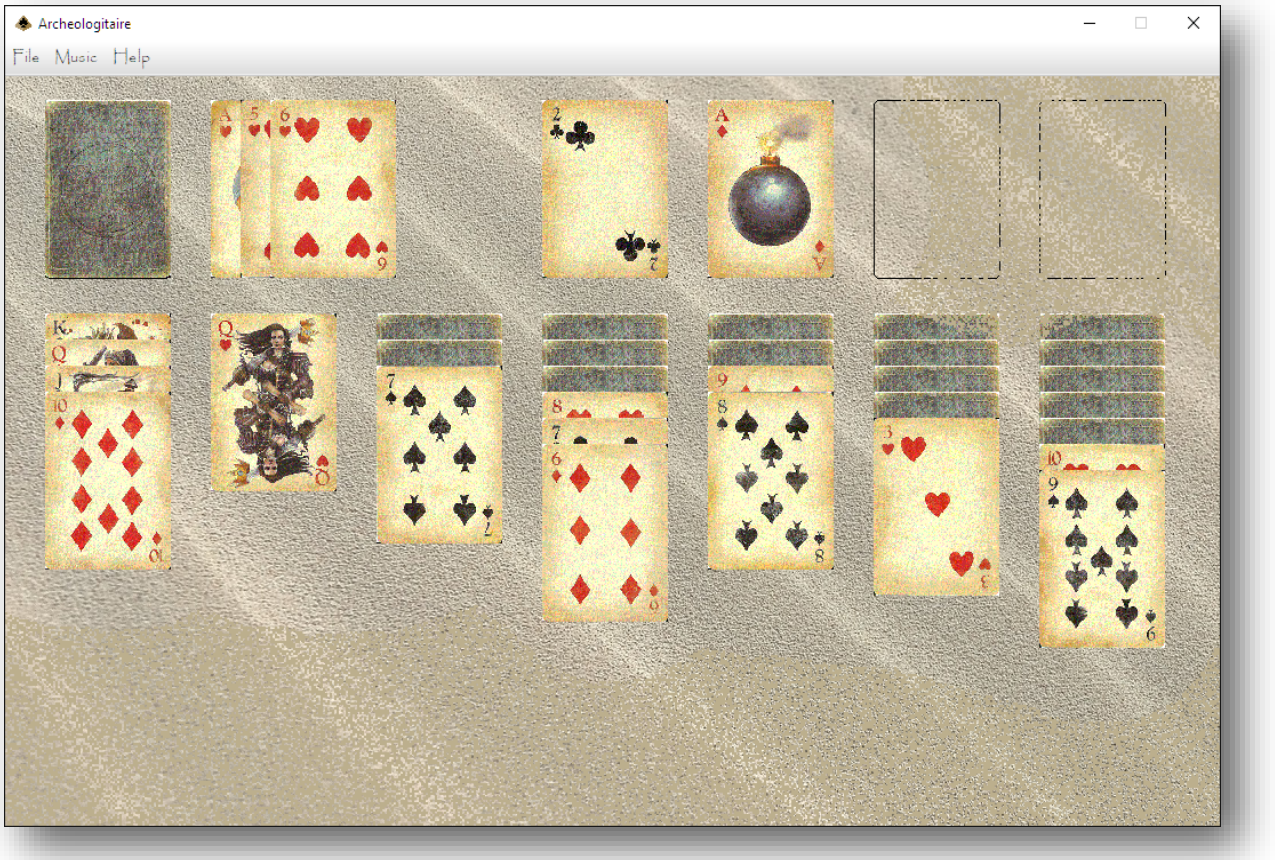
Contents

Introduction

Get immersed in the world of Archeologitaire! Play cards with the Pharaohs among the sands of Egypt, with blowing sand, crazy curses, and ancient hexes.

Accomplishments

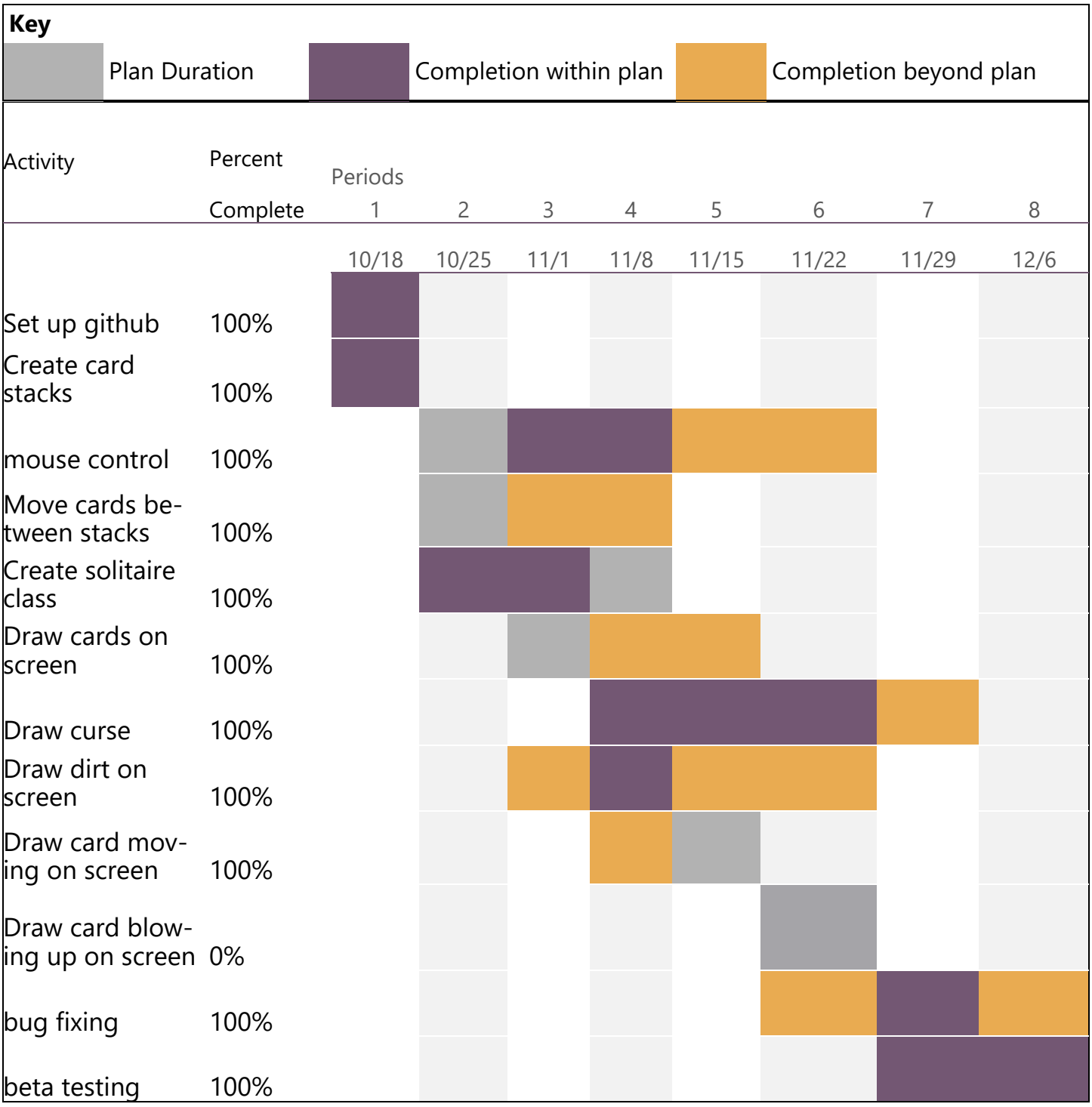
The program was built in Java. The chief accomplishment with this program involves nesting methods within classes to draw objects. The Solitaire class calls on Piles to draw themselves, the piles call on the cards to draw themselves. Jason Miner had previous experience building solitaire, and had help from Stanley Munson getting this off the ground. Matthew Smillie's contributions included adding a clearable dust board, and cursed cards that cause the player blindness.



Archeologitaire gameplay window

Project Schedule

Our Project's scheduling and fulfilment of said scheduling is described by the Gantt chart below.



Scrum Paperwork

Below is a sample of our Scrum Model for designing this project.

Vision
Create a game of solitaire, with some twists to it.

Road Map
Create Card Stacks

Create Visualization windows

Create Extra Features

Test Game

Product Backlog
Set up git / github

Create card stacks

Move cards between stacks

Create solitaire class

Draw cards on screen

Draw dirt on screen

Draw card moving on screen

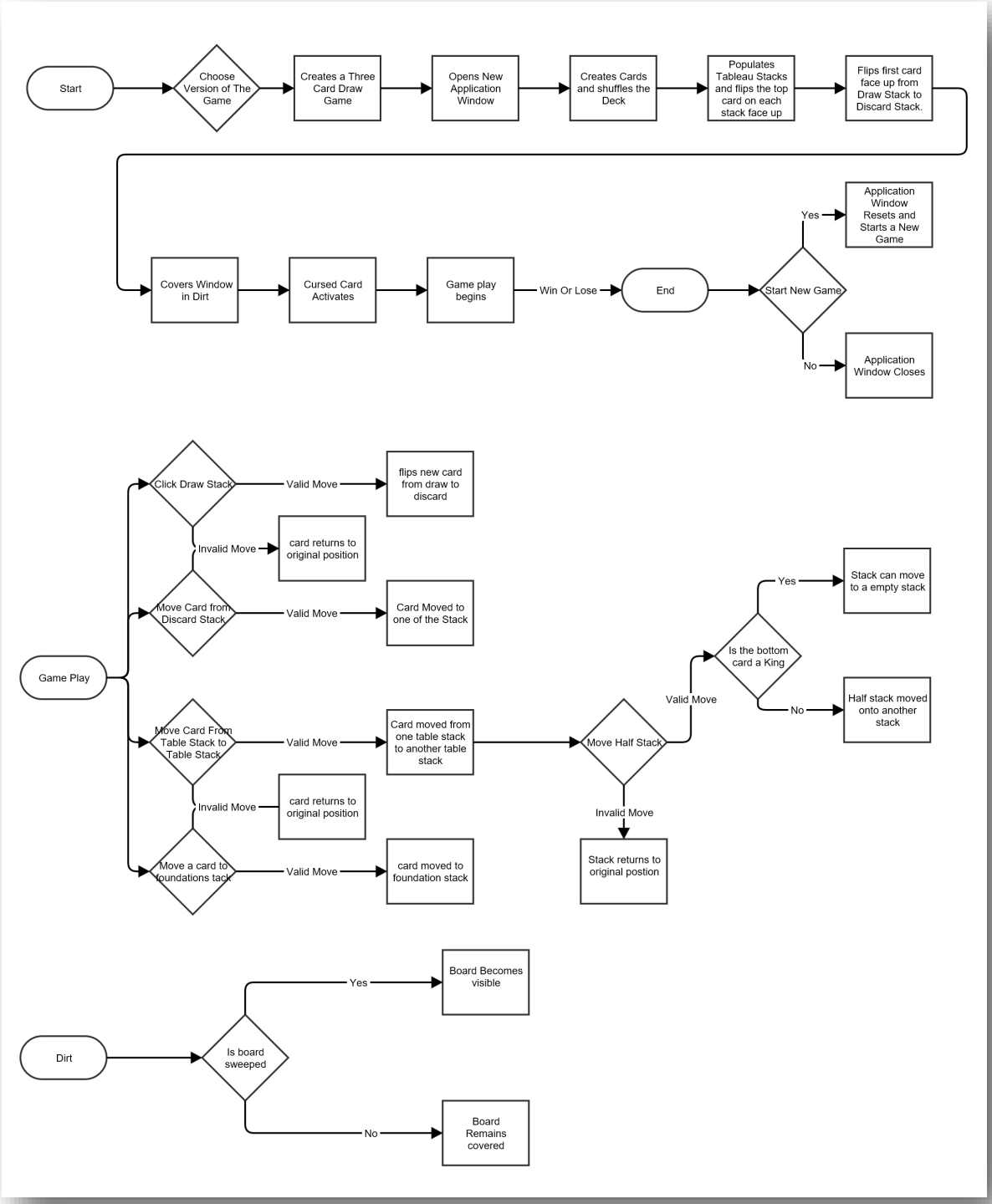
Draw card blowing up on screen

Sprint Backlog (sample)
Fully integrate cursed and dirt methods into solitaire

Fix card drawing issues

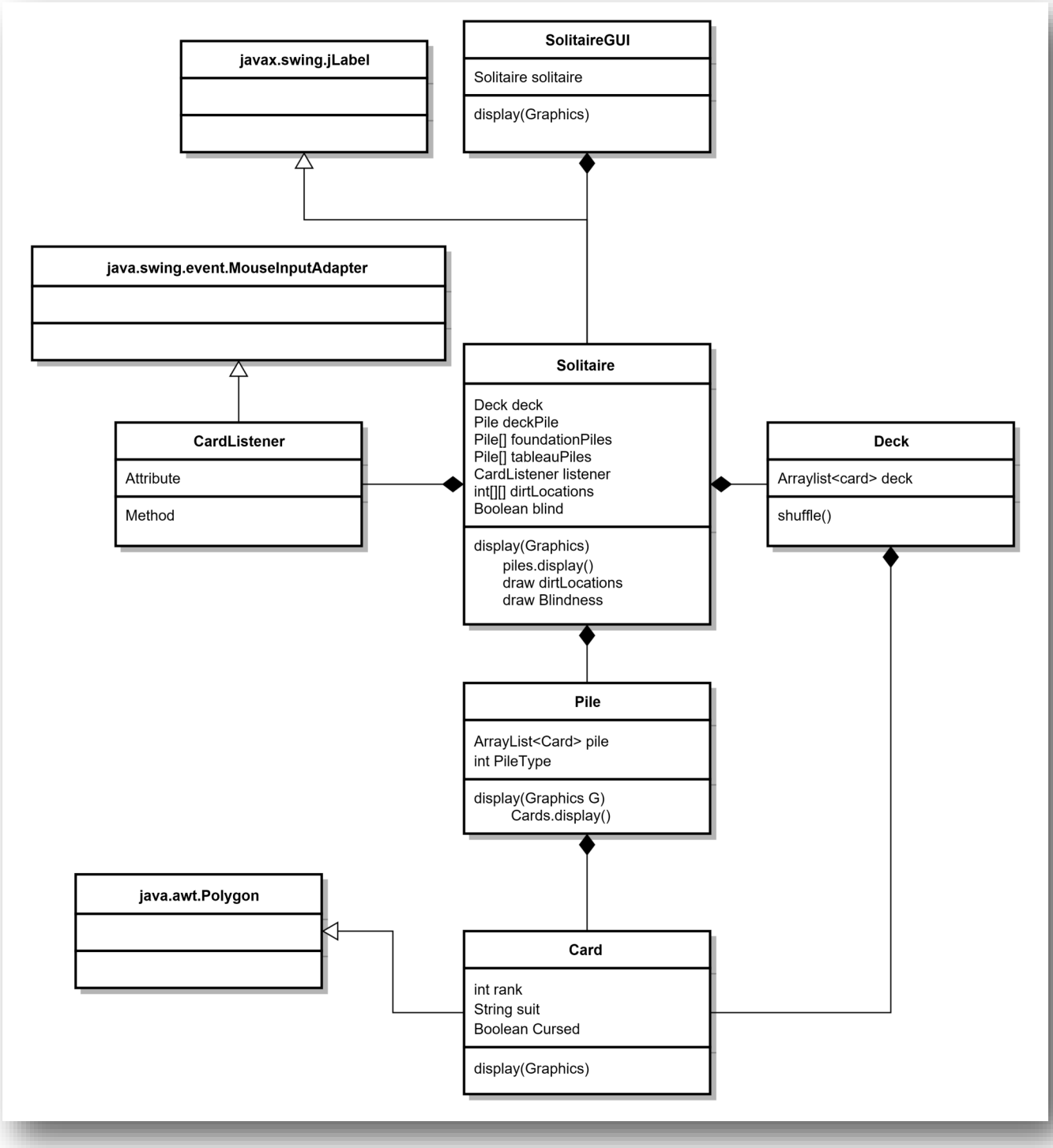
UML Activities Diagram

The following UML Flowchart describes the execution pattern of our program.



System Design

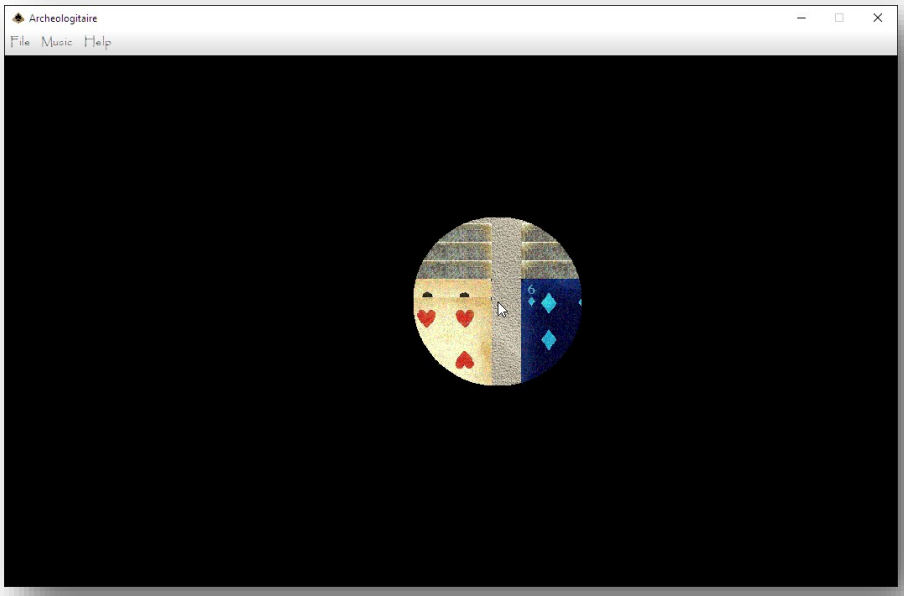
Below is a UML Class Diagram of our program.



Program Development

Drawing Blindness

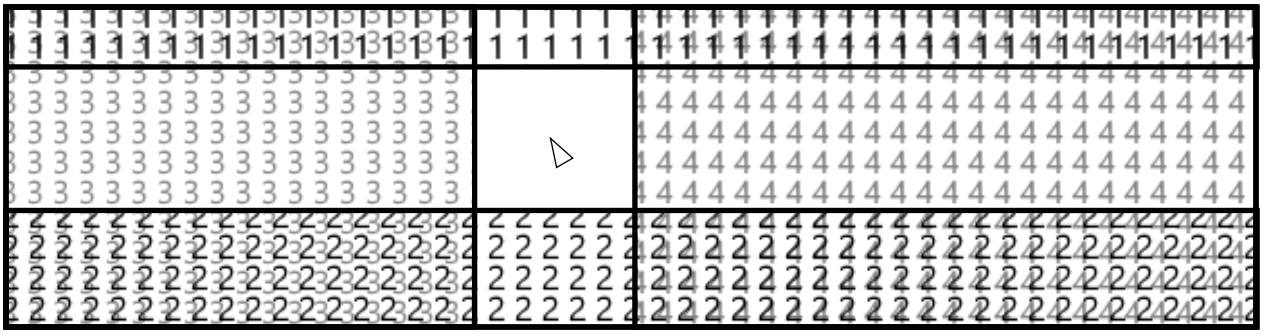
Part of the game includes a curse that gives the player blindness everywhere except within the area around their mouse.



All this should seemingly take is to draw a black rectangle over the whole area, then remove the circle around the cursor. However, Javax Graphics has no way of doing this, but the work around does the same thing in a bit more of a clever fashion.

```
g.setColor(Color.BLACK); // draw rectangles
g.fillRect(0, 0, 1064, (int) mouseLocation.getY() - blindRadius); // top
g.fillRect(0, (int) mouseLocation.getY() + blindRadius, 1064, 639 - (int) mouseLocation.getY());// bottom
g.fillRect(0, 0, (int) mouseLocation.getX() - blindRadius, 639);// left
g.fillRect((int) mouseLocation.getX() + blindRadius, 0, 1064 - (int) mouseLocation.getX(), 639);// right
```

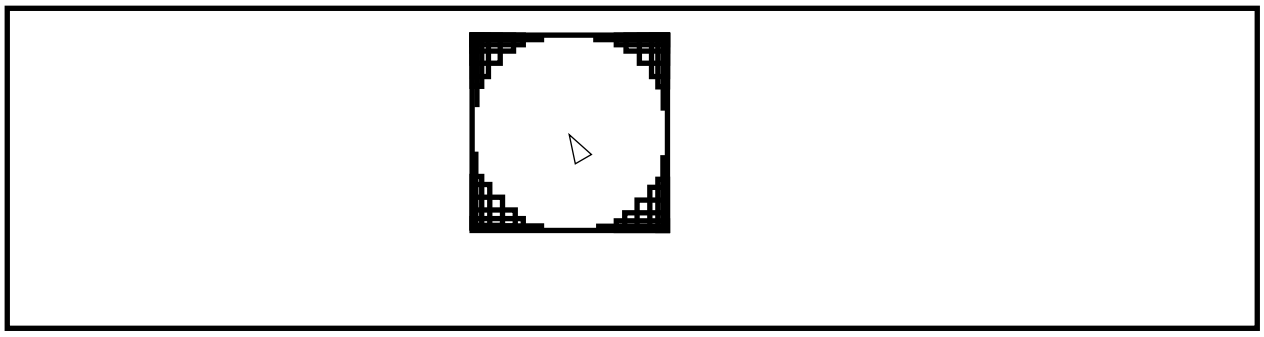
To begin with, rectangles are drawn around the cursor. The code above does this:



Next to draw the “circle”, we draw everything in the box region around the cursor — except for the circle:

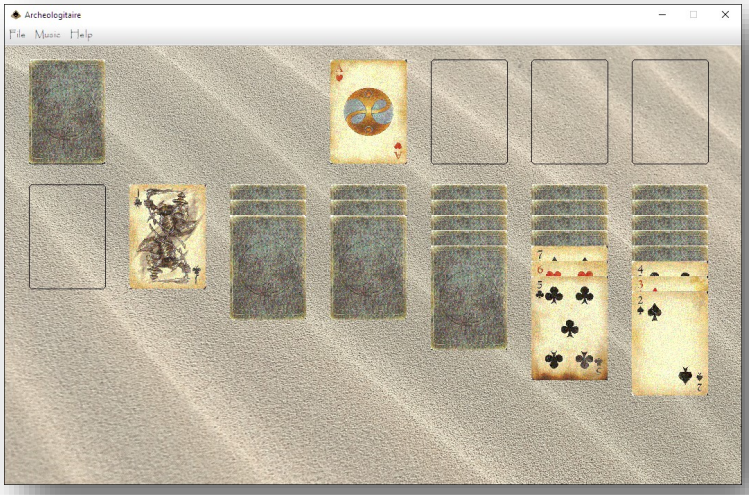
```
double angle;
double rangle;
for (int i = 0; i < 90; i++)
{
    angle = (double) i;
    rangle = Math.toRadians(angle);
    g.fillRect(// upper left
            (int) mouseLocation.getX() - blindRadius, (int) mouseLocation.getY() - blindRadius,
            (int) (blindRadius - (blindRadius * Math.sin(rangle))),
            (int) (blindRadius - (blindRadius * Math.cos(rangle))));
    g.fillRect(// upper right
            (int) (mouseLocation.getX() + (blindRadius * Math.sin(rangle)) + 1),
            (int) mouseLocation.getY() - blindRadius,
            (int) (blindRadius - (blindRadius * Math.sin(rangle))),
            (int) (blindRadius - (blindRadius * Math.cos(rangle))));
    g.fillRect(// lower left
            (int) mouseLocation.getX() - blindRadius,
            (int) (mouseLocation.getY() + (blindRadius * Math.cos(rangle)) + 1),
            (int) (blindRadius - (blindRadius * Math.sin(rangle))),
            (int) (blindRadius - (blindRadius * Math.cos(rangle))));
    g.fillRect(// lower left
            (int) (mouseLocation.getX() + (blindRadius * Math.sin(rangle)) + 1),
            (int) (mouseLocation.getY() + (blindRadius * Math.cos(rangle)) + 1),
            (int) (blindRadius - (blindRadius * Math.sin(rangle))),
            (int) (blindRadius - (blindRadius * Math.cos(rangle))));
}
```

To do this, we simply draw 90 rectangles in each corner with a width and length calculated with the sin and cosine of the angle. Something like this:



Testing

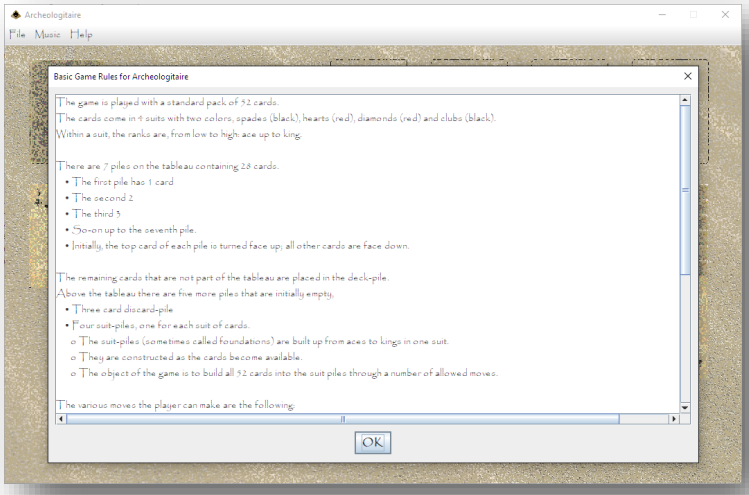
We had some of the veterans in the veterans room test the game at three major steps. They tested the game when we completed the basic solitaire game code, which us how we found out the cards where painting wrong. The code was then tweaked and tested until the bug was fixed.



We then continued to add new code and at each new feature it was crowd tested. The next two major tests came when we had them test the game after the Dirt code was

added in it's current configuration. And a third time when the cursed code was implemented.

After the game reached its final stage of development, Jason had his older brother, his uncle, and a couple people they know play and test the game. It was suggested we add some instructions. That is why we have the help menu.



Archeologitaire