Student Name: Congyao Zheng

In this project I made a game that allow users to choose an identity and to be able to fulfill two different tasks, which are 1. manipulate a polygon and cut it into a puzzle, 2. Solve a puzzle by observe matching edges.I first splited the project into several modules: 3 states (home, design, play), implement splitBy(), implement pieces creation, moving and operating pieces, achieve saving and loading all the needed pieces and shapes, select edges and deselect edges, spiral morph and animation, collision detector.
3 states: I use buttons under GUI library's help to connect the designer and player states. I numbered different states as home =0, design = 1, play = 2, with the variable *startGame* so that in draw() I can decide which state to appear.

SplitBy() and Piece Creation: *boolean LineStabsEdge()* return if two edge stabs using dot product *det(U(A, B), U(A, C)) * det(U(A, B), U(A, D )) < 0* and throw a ray return a coefficient <0(red) or {0, 1} or >1 to see the intersect points on edges. Those two functions combined let the arrow successfully decide whether it's a *goodSplit* or not. And then use *P(start, end, vector)* where the arrow start is start, the blue or red point returned by *ray* are the end, and the vector is the basically the splitting arrow. After everything I have an arraylist to store all the polygons.

Moving and operating pieces: I throw ray while user trying to move to decide which polygon the mouse is in and then set that flag that poly as selected.

Saving and Loading: I saved a counter file which recorded the number of pieces, which is also the iteration time for loading pieces. Also I called *savePts()* for each puzzle pieces in function *savePieces()* and before splitting the puzzles, call *savePts()* for each to save the ghost pieces. While loading, I first load counter from counter file and then called *loadPts()* counter times for loading puzzle pieces, original shape, and ghost pieces. I created different Arraylist to store them, so that it would be handy to operate later.

Select and deselect edges: every time the projection of mouse location onto any edge smaller than a distance, which I manually set as 3 here, the two vertex adjacent to mouse will be put into an arraylist. To do that, I check whether the projection is between vertices and the distance from mouse to the edge formed by the vertices. There are two list holding edges(two vertices), one on puzzle piece, one on the ghost. To deselect edges, I just clear the two arraylist.

Spiral morph and animation: I took the vertices in the two arraylists from edge selection step to determine which polygon is selected. And between every pair of matched edges, I use LERP to calculate spiral interpolation and update each vertex's coordinate of it. And then I manipulate a float t as time to make the updating shown smoothly. However, because of the angle will be NaN while the two edges are parallel, I separate the cases into another condition, where I also handled the situation where scale become NaN when the two edge are really close (distance = 0).

Collision detector: I first write a function to check if two edges are cross with checking whether they are splitting each other. And then for each time I move the polygon along spiral path, I run the collide() function on each edge with every other polygon pieces' edge. If any piece has reached its place, it's goal attribute will be set to true, so that no collide will be checked.