report

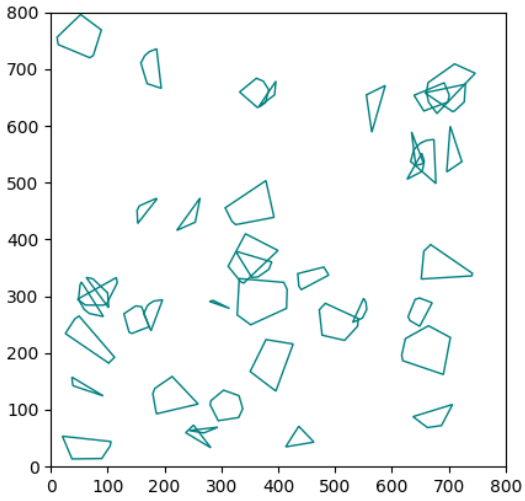
SEBASTIAN sjl214 and BENYAMIN bp535

October 2023

1 Generate, visualize store 2D polygonal scenes (20 pts)

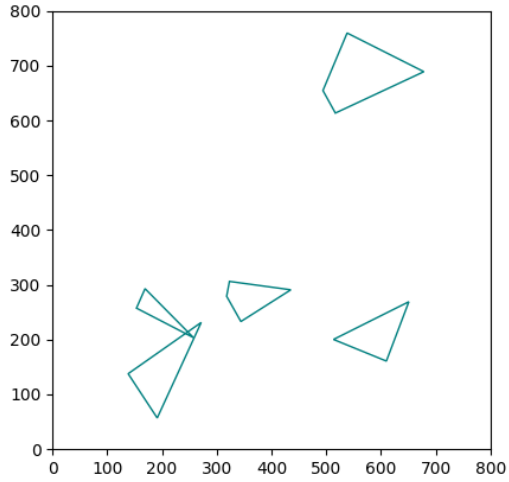
• 4 examples of maps that you have generated and the corresponding parameters you used for the generation of these maps. Use different parameters for the 4 different maps and discuss the choice of your parameters in the resulting environments you generated, i.e., try to generate some maps that are dense with obstacles and some that are sparse.

```
new file name, end it with .npy: test1.npy
Enter how many polygons: 40
radius minimum: 25
radius max: 50
vertex minimum: 3
vertex max: 7
```



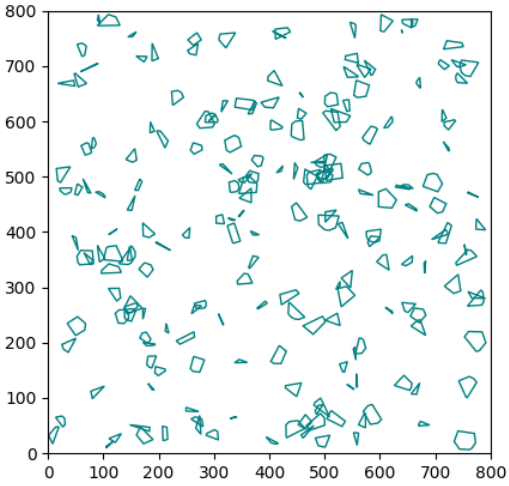
I thought that having 40 polygons would be perfect—a dense enough amount of polygons but not overly dense. I chose a range of 25-50 so that I could actually see the polygons without needing to zoom in. The range of vertex counts was also chosen carefully so that I could see that it actually produces polygons with vertex counts within the specified range.

```
new file name, end it with .npy: test2.npy
Enter how many polygons: 5
radius minimum: 50
radius max: 100
vertex minimum: 3
vertex max: 7
```



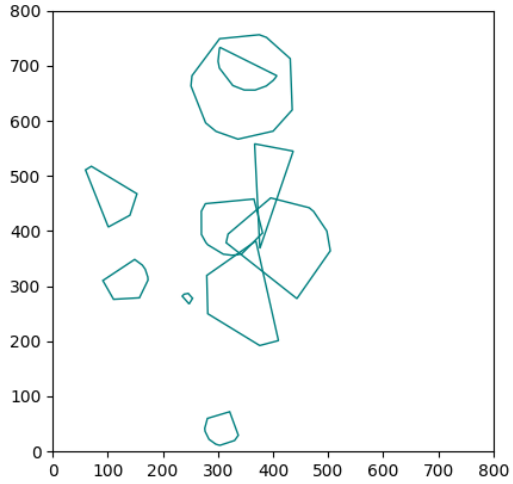
I decided to make it less dense with polygons and increase the range so the polygons still take up some space.

```
new file name, end it with .npy: test3.npy
Enter how many polygons: 200
radius minimum: 10
radius max: 20
vertex minimum: 3
vertex max: 7
```



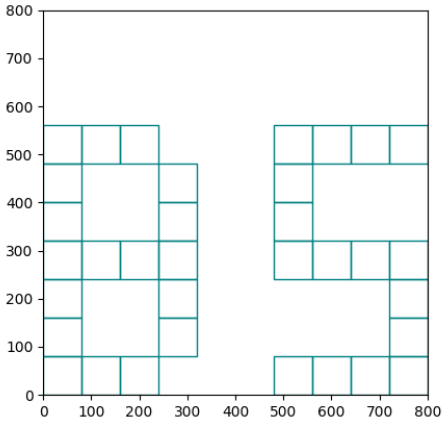
i made it denser in areas but more sparse because the amount of polygons increased by alot but the size decreased.

```
new file name, end it with .npy: test4.npy
Enter how many polygons: 10
radius minimum: 10
radius max: 100
vertex minimum: 3
vertex max: 12
```



I made a graph with a wide range of ranges and vertices

• A visualization of one additional map, which you have manually constructed, where you use convex polygons to form two letters inside the space. These two letters should correspond to the first letter of the first name of each team member (or the first letters of the first and last name for students working individually). You do not have to spend much time here to make the letters accurately represented and the convex polygons perfectly aligned. As long as the letters are recognizable when visualized, this is sufficient. See example Fig. 1 for letter "E".



• Describe the exact approach you followed in order to generate the convex polygons. If you followed the above mentioned strategy describe the operations you performed so as to i) rotate each polygon, ii) extrude vertices and iii) how

you determine/enforce the convexity of each polygon. If you deviated from the above strategy indicate so and explain your choices.

The code stays in border by making sure the center is always radius away from border.

The code does not explicitly rotate the polygon. Instead, it generates the vertices directly in a way that ensures convexity.

A loop was created that iterates for number of vertices, and inside it generates angles and polar coordinates.

After being generated, each vertex is ordered according to its polar angle to the inscribed circle's center.

Sorting is achieved using the vertices.sort() method with a custom key function that calculates the polar angle with math.atan2():

vertices.sort(key=lambda point: math.atan2(point[1] - center[1], point[0] - center[0]))

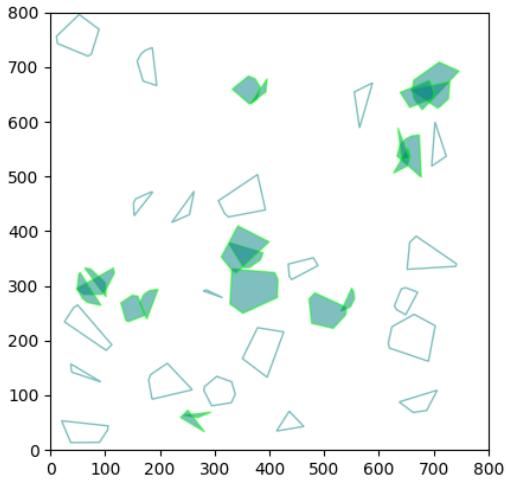
Sorting ensures that the vertices are ordered in a way that preserves the convexity of the polygon.

2 2D collision checking for convex polygons (20pts + 10 points for efficient implementation)

• Save the figure and put it in the report.

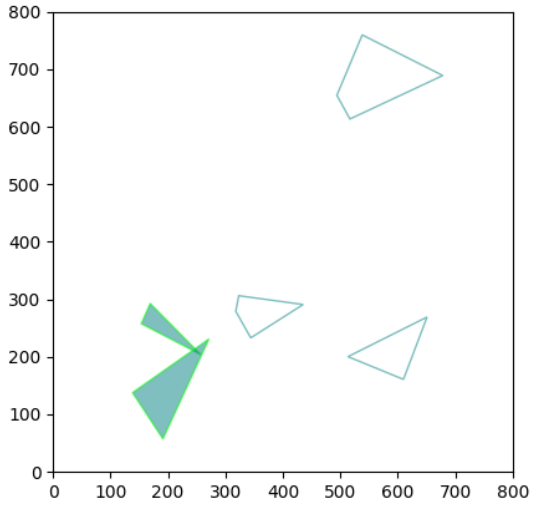
---

test1.npy

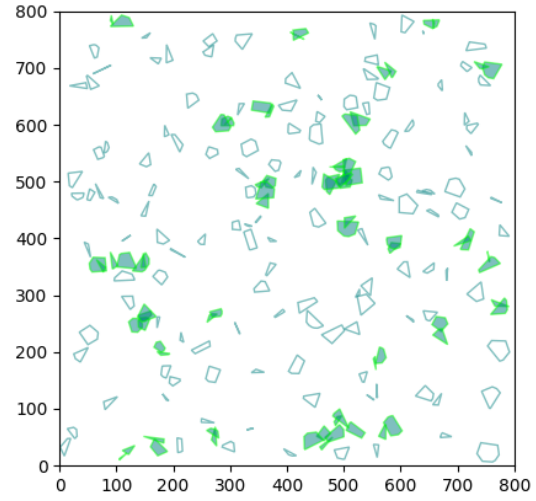


---

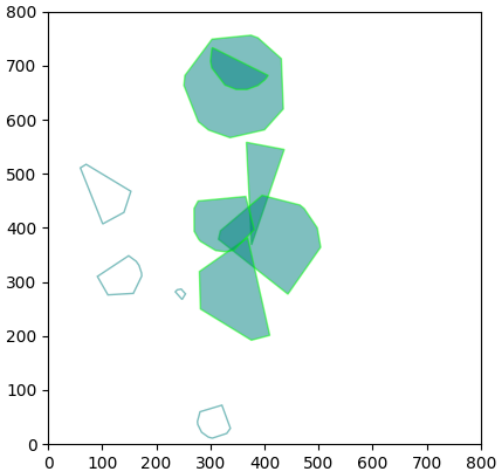
test2.npy



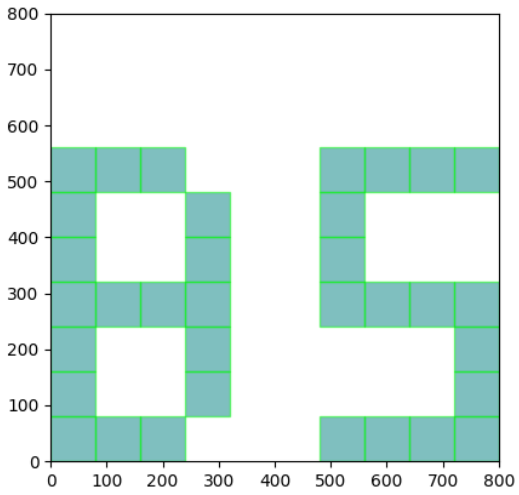
test3.npy



test4.npy



letters.npy



• Your report should also describe how you implemented your collision checking approach

When i was looking for a collision checking approach i stumbled upon the Separating Axis Theorem. So i decided to implement it myself due to the fact it is an efficient method for collision detection, especially when dealing with convex shapes. It does not require a detailed examination of the shapes' geometry and can quickly determine if they are in collision.

normalize(vector) divides a 2D vector by its length (magnitude) in order to normalize it. The objective is to maintain the vector's original direction while

ensuring that its length is 1. I return the input vector exactly as it was if it had a length of 0 (i.e., it was a zero vector).

Polygonminmax(polygon, axis) calculates the minimum and maximum values of the projections of a polygon onto an axis given a polygon expressed as a list of vertices and an axis (a 2D vector). I compute the projections onto the axis for each polygonal vertex as I iterate across the polygon, keeping track of the least and maximum projections.

isseparatingaxis(axis, polygon1, and polygon2): This function determines if an axis, specifically Polygon1 and Polygon2, serves as a separation axis between two polygons. I do this by mapping both polygons onto the axis, then determining if the resulting intervals—defined by minimum and maximum values—do not overlap. If the two polygons do not overlap, the axis does, in fact, divide them.

The primary function for checking whether two convex polygons, denoted by their vertices (vertices1 and vertices2), are colliding is checkpolygons(vertices1, vertices2). In order to make subsequent calculations simpler, I turn the vertices into NumPy arrays.

I determine the perpendicular axis to each edge of polygon 1 and use the isseparatingaxis function to determine whether it serves as a separating axis. I instantly return False, indicating that there was no collision, if it turns out that either of these axes is splitting.

I then carry out the same procedure along each polygon2 edge.

I return True, indicating that the polygons are truly colliding, if none of the axes for either of the two polygons is separating.

In collisiondetectionfull(numpyverticies) i use the the checkpolygons(vertices1, vertices2) on all the polygons in a loop. I make sure to check every polygon combination, and fill them in with a color if they collide.

• Report how much faster is the code due to your changes by evaluating the code on the 5 scenes from part 1.

I tried making it run faster by skipping polygons that we already knew collided run checkpolygon, but it ended up skipping polygon collisions when there were more than 2 polygons colliding. I didn't even get to check how much faster, but this only would speed it up on occasion, and it was skipping collisions.I left the code inside my project, if i wanted to try to make it work ,but didn't have time.

3 Collision-free Navigation for a 2D rigid body (30 pts)

• Report which 4 keys you have assigned to the corresponding control (e.g., up/down, left/right arrows)

use arrow keys to move

4 Collision-free Movement of a Planar Arm (30 pts)

•Use the keyboard to control the two joints individually (you can achieve unlimited rotation with each joint). Indicate in your report what keys you used in order to control the arm. use a and d and ¡- and -¿ to control the arm.