CS360 Artificial Intelligence Final Project Report

Pentagons Formation using Tangram Pieces

Students:
Man-Seui Chan

Lecture in Charge:
Tsang, Cheung-choy Eric

Macau University of Science and Technology
Faculty of Innovation Engineering

2022.05

Contents

1 Introduction

1.1 Question description

Tangram is one of the most popular games to play with. You put figures of 7 pieces together (five triangles, one square and one parallelogram). You must use all pieces. They must touch but not overlap.

All seven tangram pieces consist of many half squares(triangles), each with this shape:

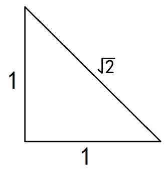


Figure 1: Basic shape

There are 32 half squares or 16 squares altogether.

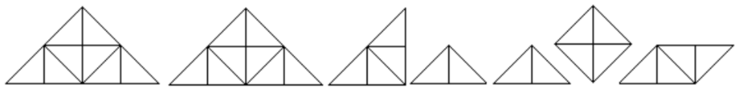


Figure 2: A set of tangram pieces

We could take the half-squared triangle as the basic form, because each square built for all seven tangram pieces has the simple length (perimeter) of 4 units.

There are many convex and non-convex figures/shapes of pentagons that you can build from all the 7 tangram pieces. Some of them are given below.

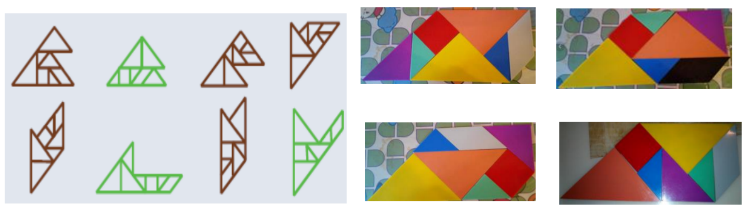


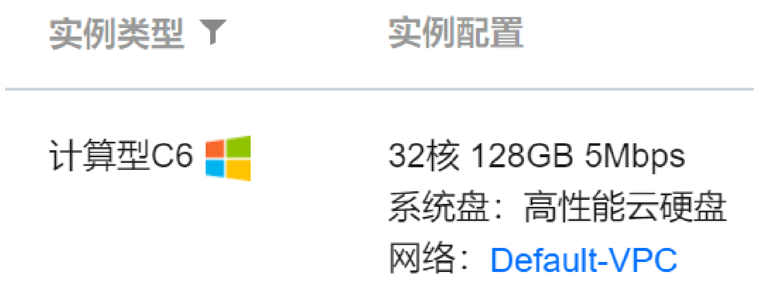
Figure 3: Convex and non-convex figures

In this project, you are required to use AI search methods (uninformed search methods such as Breadth First search, Depth First search, Iterative Deepening and Uniform Cost search) and (Informed or heuristic search methods such as Greedy search, A* algorithm and Iterative Deepening version of A* : IDA*) to perform pattern matching, recognition and pentagons formation.

1.2 A brief description of our method and the result

In this project, we use uninformed search methods - the breadth-first search(BFS) method and the depth-first search(DFS) method, and informed search method - Greedy search method to get formations of the pentagon through tangram pieces. After running our program, we finally get xxx formations of the pentagon through tangram pieces as the result.

Cloud server configuration: 32 cores 128GB 5Mbps, system disk: high performance cloud hard disk, network: Default-VPC.



实例类型	实例配置
计算型C6	32核 128GB 5Mbps 系统盘：高性能云硬盘 网络：Default-VPC

Figure 4: Cloud server

The following table shows the results and running time of each method

Table 1: Running Result

Method	Number of results	Running time
breadth-first search	55	14 hours
depth-first search	55	14 hours
Greedy search method	4	6 hours

2 Sub-problem formulation

2.1 An exmple

We stitch the tangram pieces according to the vertices of the polygon, here we give an example to explain the sub-problem formulation.

According to Fig.5, this polygon has 3 vertices.

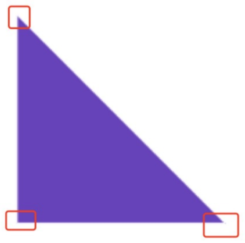


Figure 5: A tangram piece

Then, as Fig.6 shows, we propose to stitch the upper polygon with another polygon in the tangram pieces.

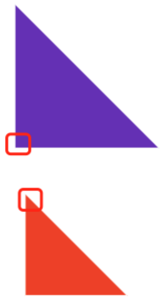


Figure 6: Two tangram pieces

The polygons in Tangram are all made up of square triangles with side length 1, so we can set the base rotation angle to $\frac{\pi}{4}$, so in two dimensions we have 8 angles($\frac{\pi}{4}$, $\frac{\pi}{2}$, ..., 2π) to rotate.

However, we will meet following three sub-cases:

a. Two sides of two polygons overlap (the case we want to get).

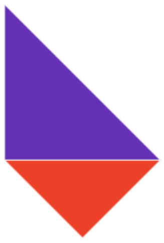


Figure 7: Overlap case

b. The two polygons do not have any sides overlapping.

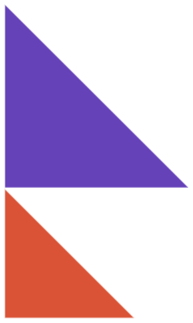


Figure 8: Do not have any sides overlapping case

c. Two polygon shapes with overlap inside.

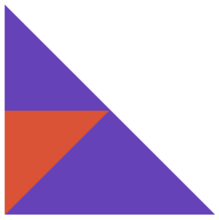


Figure 9: Overlap inside case

2.2 The relationship between points - segments, segments - segments

a. Point-line

So, we should determine if the point is on a line segment and if the sides of a polygon overlap.

The determination of whether a point is on a line segment is made by the inner product of the vector formed by the point and the endpoints of the line segment.

Figure 10: Point-line position relationship

Suppose $A(x_0, y_0, z_0)$ is a point on the line l and let a non-zero unit vector $\vec{l} = (l_1, l_2, l_3)$ (also knows as the unit-direction vector).

Then, let $P(x, y, z)$ stochastic point on the line l so that: $(x - x_0, y - y_0, z - z_0) = \lambda(l_1, l_2, l_3)$.

Finally, we have that: $|\vec{AP}| = |\vec{AP} * \vec{l}| = \lambda$.

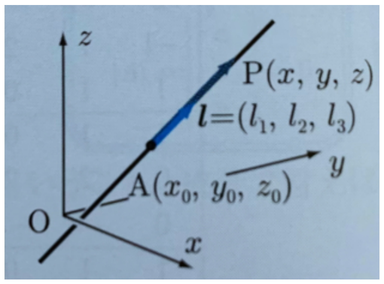


Figure 11: Inner product

b. Line-line

i. Overlapping

To determine whether line segment s_1 and line segment s_2 have overlapping parts is to determine whether s_1 on line segment s_2 and s_2 on line segment $s_1$hold at the same time, and if they hold at the same time, then the two line segments overlap.

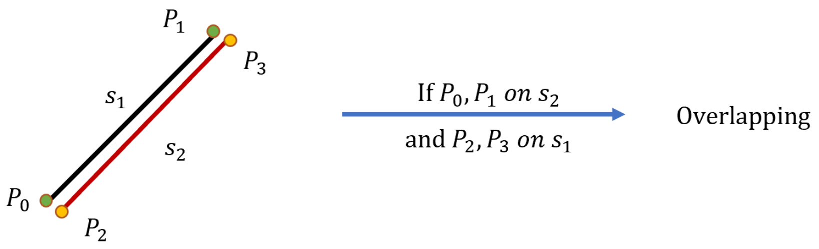


Figure 12: Two segments

ii. Intersection

To determine whether line segment s_1 and line segment s_2 have an intersection, we using cross product: Size: $|a \times b| = |a||b|sin\theta$; Direction: $a \times b \perp a, a \times b \perp b$; Pointing: $The \quad right - hand \quad rule.$

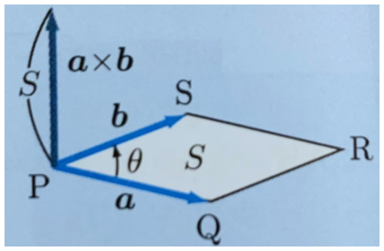


Figure 13: Cross product

Then, for the segment s_1 and the segment s_2, we verify $(\vec{v_1} \times \vec{l_1}) \cdot (\vec{v_2} \times \vec{l_1}) < 0$.

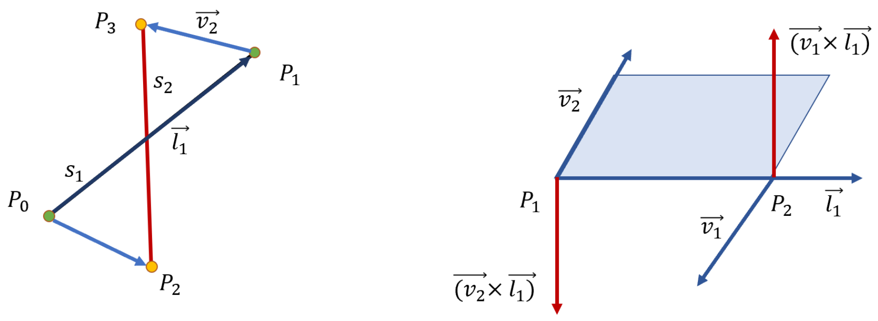


Figure 14: Segments intersection – 2D and 3D space

2.3 The specific code implementation

a. Define vertices and operations

```python
# Define vertices, rewrite, add, subtract and equal operations
class Point:
    instance = {}
    lock = Lock()

    def __init__(self, x=0., y=0.):
        self.x = float(x)
        self.y = float(y)

    def __eq__(self, other):
        """Define the equal operator between 2 points.
        Return True when the coordinates of 2 points are the same.
        """
        if not isinstance(other, type(self)):
            return False
        if self.x == other.x and self.y == other.y:
            return True
        return False

    def __add__(self, other):
        """Define the add operator between 2 points."""
        return Point.get_instance(x=self.x + other.x, y=self.y + other.y)

    def __sub__(self, other):
        """Define the sub operator between 2 points.
        Can be used to calculate the vector.
        """
        return Point.get_instance(x=self.x - other.x, y=self.y - other.y)

    def __hash__(self):
        return (hash(self.x) << 31) ^ hash(self.y)

    def get_coordinate(self):
        return [self.x, self.y]

    def point_is_coincide_point(self, another_point, threshold):
        """If the distance between 2 points is within an error threshold,
        then we say they coincide with each other.
        """
        if utils.get_distance_point_to_point(self, another_point) < threshold:
            return True
        return False

    def display(self):
        """For debug."""
        print('coordinate:', self.get_coordinate())
```

Since coordinates are float numbers, precision will be lost during the operation. All instances of points are created through the *get_instance* class method, so that there is only one instance of a point in the system, and it is possible to determine whether they are equal or not.

```python
  @classmethod
    def get_instance(cls, x=0., y=0.):
        x_rounded = round(x, 4)
        y_rounded = round(y, 4)
        assert cls.instance is not None
        key = (hash(x_rounded) << 31) ^ hash(y_rounded)
        with cls.lock:
            if key not in cls.instance:
                cls.instance[key] = cls(x_rounded, y_rounded)
            return cls.instance[key]
```

b. Define segments and operations

```python
# Define segments, equal, point on segment, segments overlaping,
# segment intersection, segments crossing operations
 class Segment:

    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2
        self.midpoint = Point.get_instance(x=(p1.x + p2.x) / 2.,
        y=(p1.y + p2.y) / 2.)
        self.vec = self.p2 - self.p1

    def __eq__(self, other):
        if not isinstance(other, type(self)):
            return False
        if self.p1 == other.p1 and self.p2 == other.p2:
            return True
        if self.p2 == other.p1 and self.p1 == other.p2:
            return True
        return False

    def __hash__(self):
        return hash(self.p1) ^ hash(self.p2)

    def point_is_on_segment(self, another_point, threshold, end_point=True):
        if end_point:
            return self.__point_is_on_segment(another_point, threshold)
        else:
            if self.p1 == another_point or self.p2 == another_point:
                return False
            return self.__point_is_on_segment(another_point, threshold)

    def __point_is_on_segment(self, another_point, threshold):
        """Judge whether a point is ON this segment.
        Endpoint of the segment is considered on here.
        threshold is a term that allows for tiny error
        (caused by non-integer of coordinates)
        """
        dist1 = utils.get_distance_point_to_point(self.p1, another_point)
        dist2 = utils.get_distance_point_to_point(self.p2, another_point)
        if dist1 < threshold or dist2 < threshold:
            # the point is one of the endpoints of this segment
            return True

        v1 = self.p1 - another_point
        v2 = self.p2 - another_point
        if abs(utils.inner_product(v1, v2) + dist1 * dist2) < threshold ** 2:
            return True
        return False

    def segments_has_overlap(self, another_segment, threshold):
        if self.point_is_on_segment(another_segment.p1, threshold)
        and self.point_is_on_segment(another_segment.p2,

            return True
        if another_segment.point_is_on_segment(self.p1, threshold)
        and another_segment.point_is_on_segment(self.p2,

            return True

    def segment_is_intersect_segment(self, another_segment, threshold):
        """Judge intersect using cross product.
        Won't be used outside the class so set it as private method.
        """
        # 2 segments are not parallel
        v1 = another_segment.p1 - self.p1
        v2 = another_segment.p2 - self.p1
        if utils.cross_product(v1, self.vec) * utils.cross_product(v2, self.vec)
        < -(threshold ** 4):
            return True
        return False

    def segment_is_cross_segment(self, another_segment, threshold):
        """Judge whether this segment is cross over another segment.
        Use mutual cross product to judge.
        Any parallel or even coinciding segments are not considered intersect
        here.
        __|__ is not considered intersect too.
        Only --|-- is considered intersect!
        threshold is a term that allows for tiny error
        (caused by non-integer of coordinates)
        """
        if self.point_is_on_segment(another_segment.p1, threshold) or \
                self.point_is_on_segment(another_segment.p2, threshold) or \
                another_segment.point_is_on_segment(self.p1, threshold) or \
                another_segment.point_is_on_segment(self.p2, threshold):
            # one point of one segment is on another segment
            return False

        if abs(self.vec.x * another_segment.vec.y -
        self.vec.y * another_segment.vec.x) < threshold ** 2:
            # 2 segments are parallel
            return False

        if self.segment_is_intersect_segment(another_segment, threshold) and \
                another_segment.segment_is_intersect_segment(self, threshold):
            return True
        return False

    def display(self):
        """For debug."""
        print('point1:', self.p1.get_coordinate())
        print('point2:', self.p2.get_coordinate())
```

3 Uninformed search methods

3.1 Initalize

We know that a set of tangram pieces is composed of two large triangles, one medium triangle, two small triangles, one square, and one parallelogram(the above polygons are defined in advance in the *TangramElementEnum* class as sets of points).

Then we set two enums, the *self.unused_elements_enum* and the *self.used_elements_enum* to store polygons.

So, during initialization, we put the above elements into the *self.unused_elements_enum* index sequence as an iterable object.

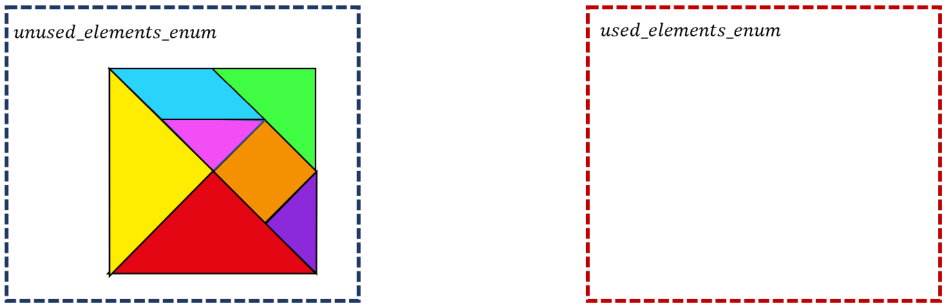


Figure 15: Uninformed search methods - Initalize

3.2 The BFS method

a. The First level of the BFS traversal tree

The First level of the BFS traversal tree is shown in Fig.16. For the green triangle, we define three vertices, p_1, p_2, and p_3. We overlap each of these three vertices with the origin and rotate them by $\frac{\pi}{4}$, $\frac{\pi}{2}$, ..., 2π to traverse all cases.

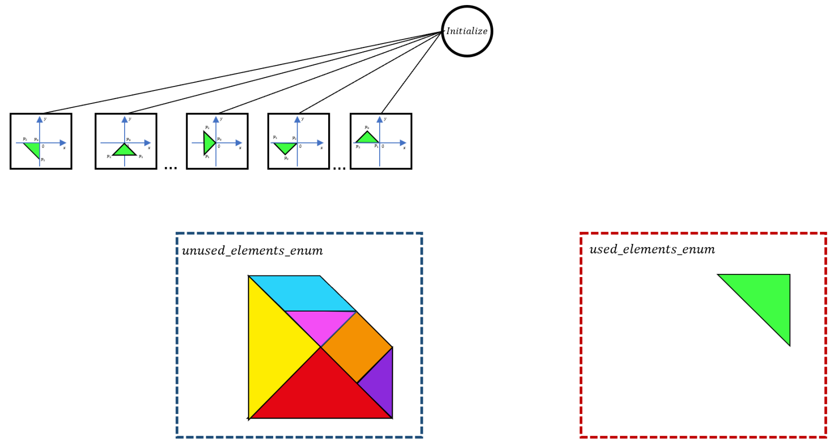


Figure 16: Iteration of the green triangle in the first layer of the BFS traversal tree

After that, we traverse all remaining peices in the *unused_elements_enum* in the same way.

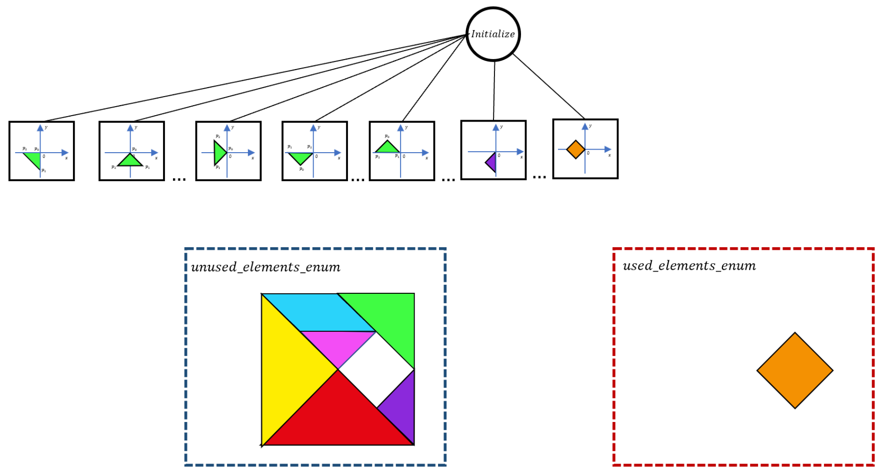


Figure 17: Traverse all remaining peices in the *unused_elements_enum*

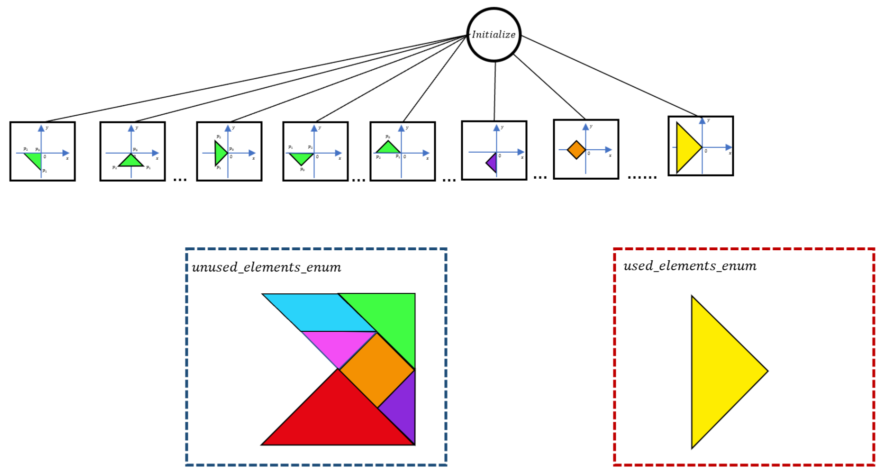


Figure 18: Traverse all remaining peices in the *unused_elements_enum*

b. The Second level and lower levels of the BFS traversal tree

When we traverse the Second layer, we are traversing the same iterative method as the First layer and so the vertices of the polygon in the layer above. As shown in Fig. 19, at this point, we can discard some cases that do not meet the requirements, until we find one that does.

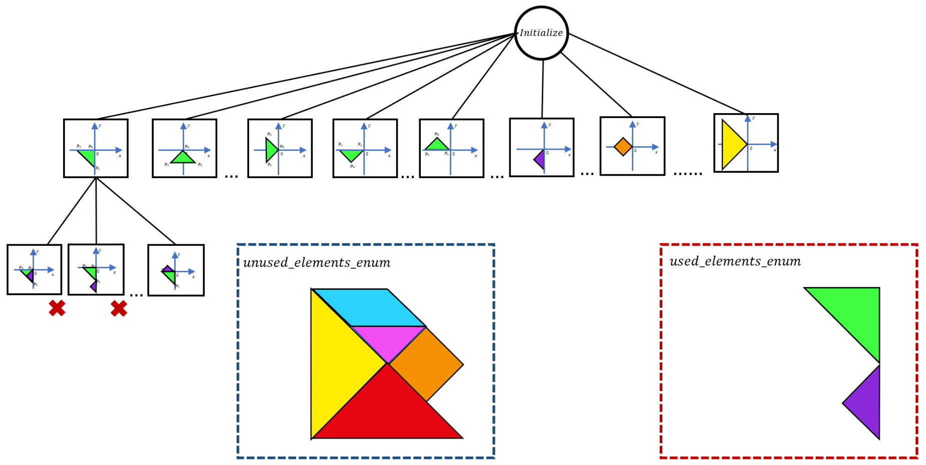


Figure 19: Traverse the Second layer, aborting cases

When we traverse the second layer, we are traversing the same iterative method as the first layer and so the vertices of the polygon in the layer above.

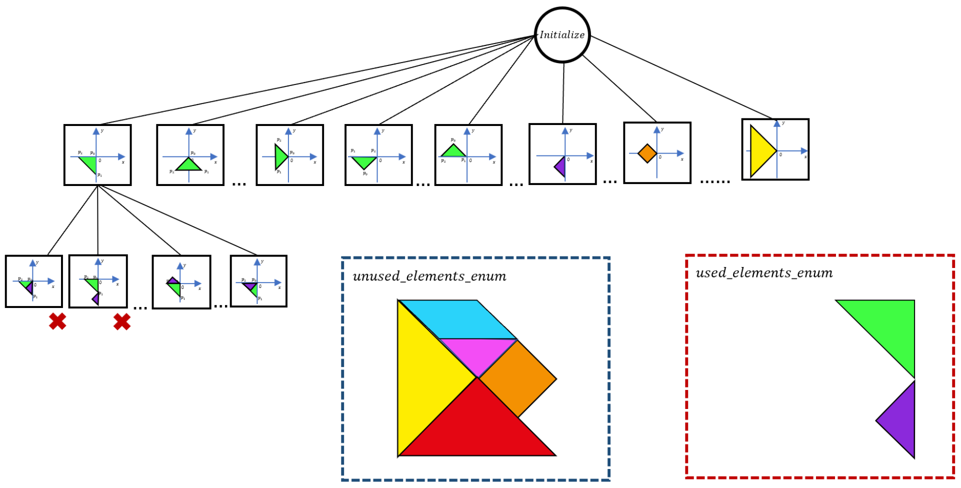


Figure 20: Iterate through all the cases of the Second layer of the First node of the First layer

After that, we finish traversing all the nodes in the Second layer as Fig.21 shows, and keep repeating the previous operations to finally complete the traversal.

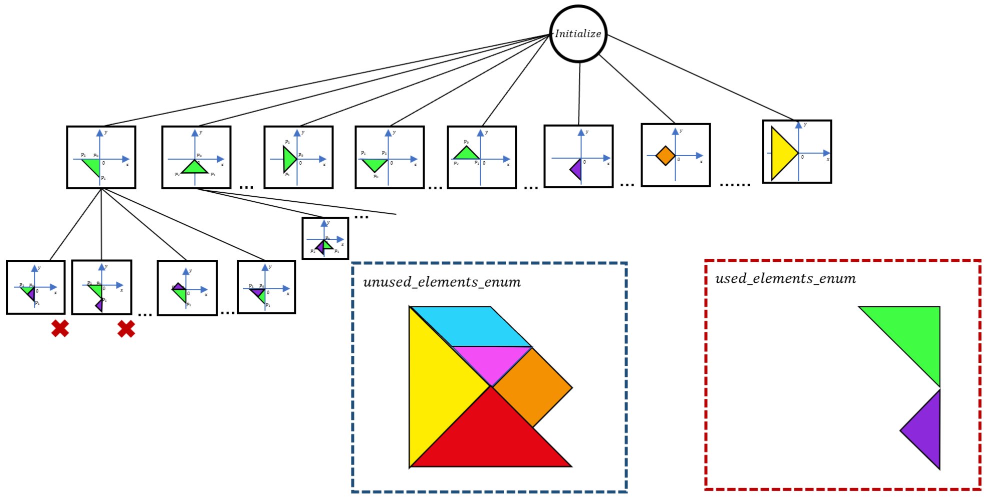


Figure 21: Traversing all the nodes in the Second layer

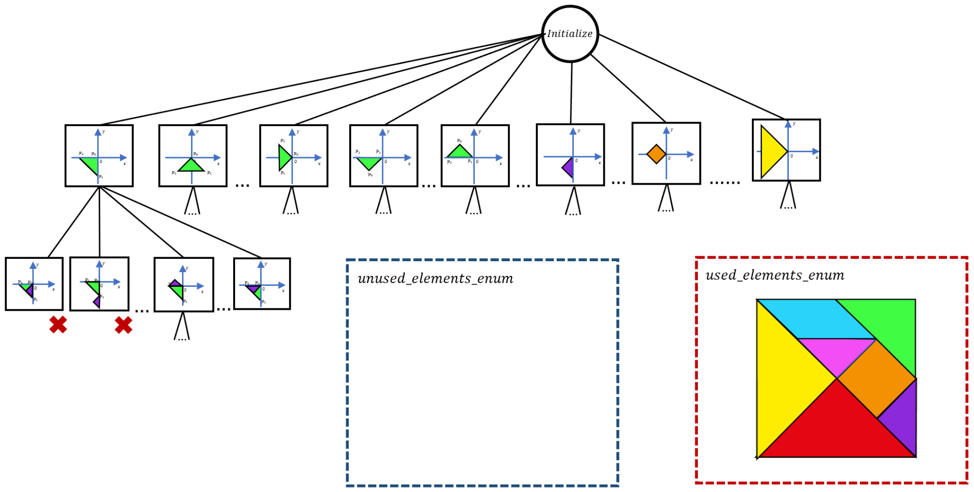


Figure 22: Complete the traversal

3.3 The DFS method

a. DFS traverses the subtree corresponding to the first node of the first level of the tree

First generate the First node of the First level of the traversal tree, i.e.: the rotation angle of the green triangle is 0 and the p_0 point coincides with the origin.

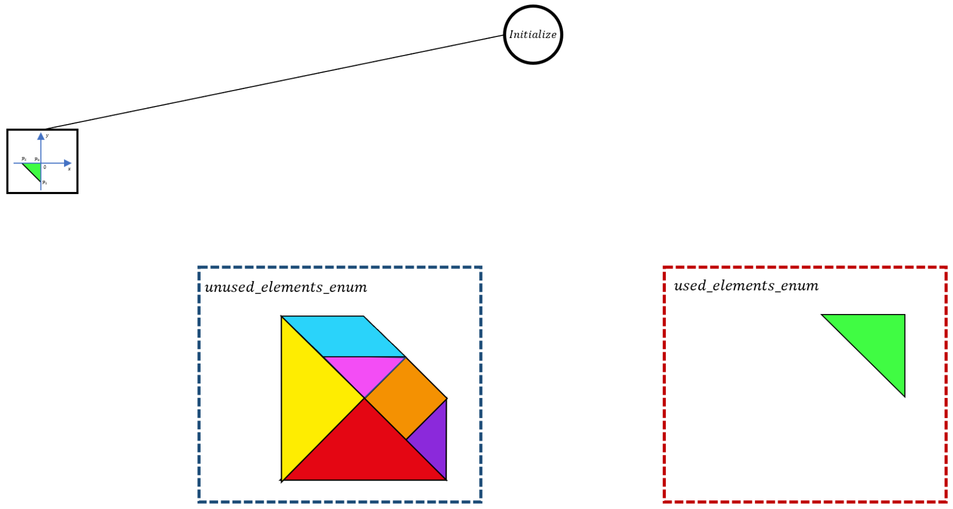


Figure 23: The First node of the First level of the traversal tree

Like the BFS method, we iterate through all the cases in the second layer of the first node of the first layer, aborting the cases that do not meet the requirements, until we find a case that meets the requirement.

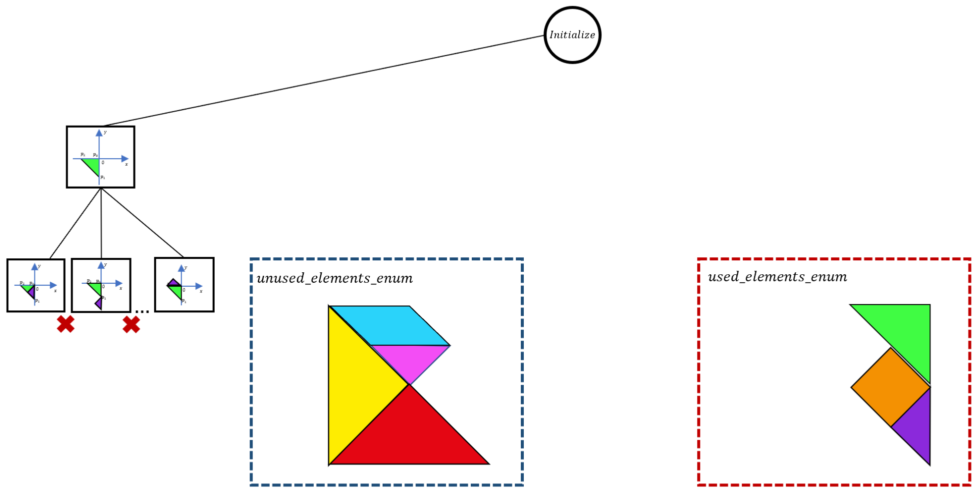


Figure 24: Traversing until find a case that meets the requirement

After that, we repeat the above method and search for nodes in the third level up to the bottom.

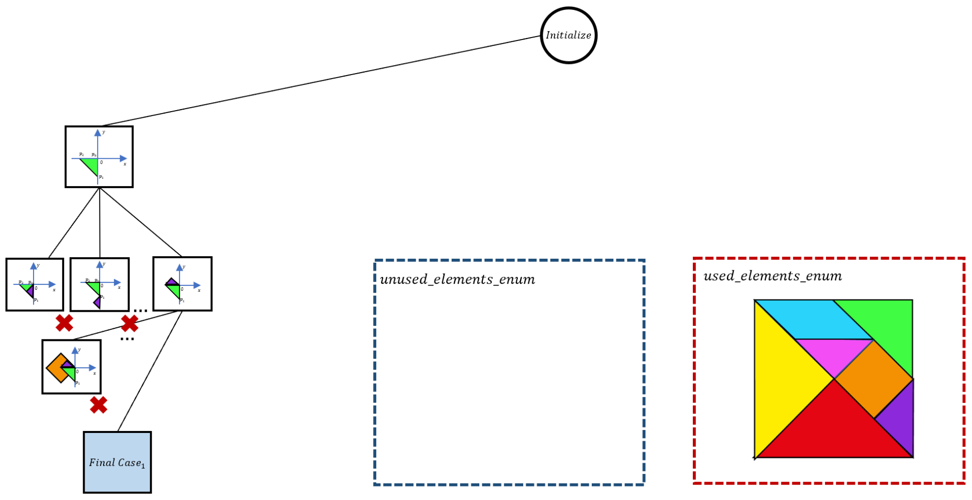


Figure 25: Search for nodes in the third level up to the bottom

Find all cases that match the node selected in the Second layer, and denote them as "$FinalCase_1$, $FinalCase_2$, ..., $FinalCase_n$".

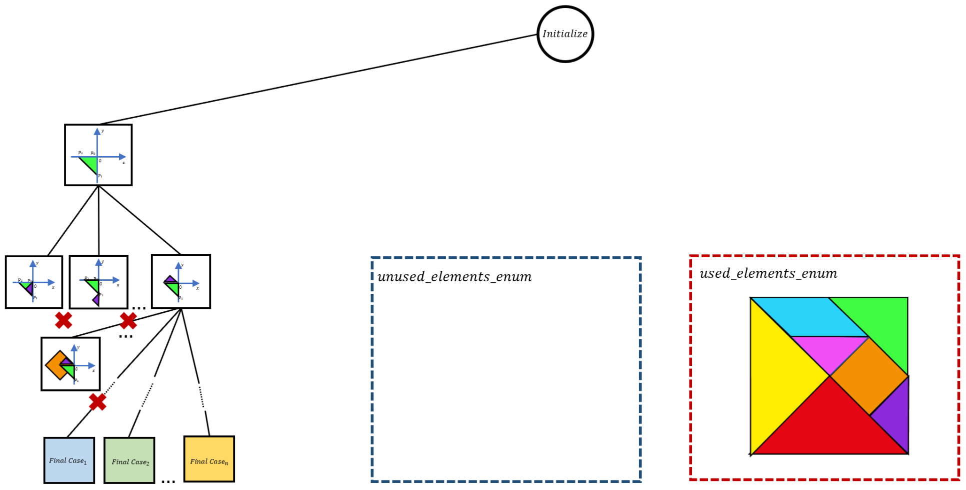


Figure 26: Traversing the all cases in the selected node in the Second layer

After that, all the nodes of the first node of the first layer are traversed as above.

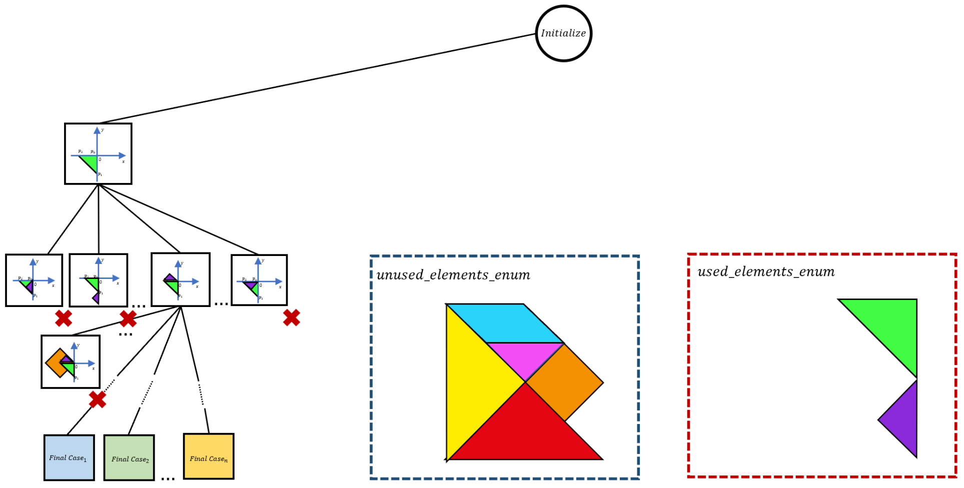


Figure 27: Traversing the all cases in the the nodes of the first node of the first layer

b. DFS traverses the subtree corresponding to other nodes of the tree

After that, we finish traversing all the nodes to the first node of the first level of the tree, and keep repeating the previous operations to finally complete the traversal.

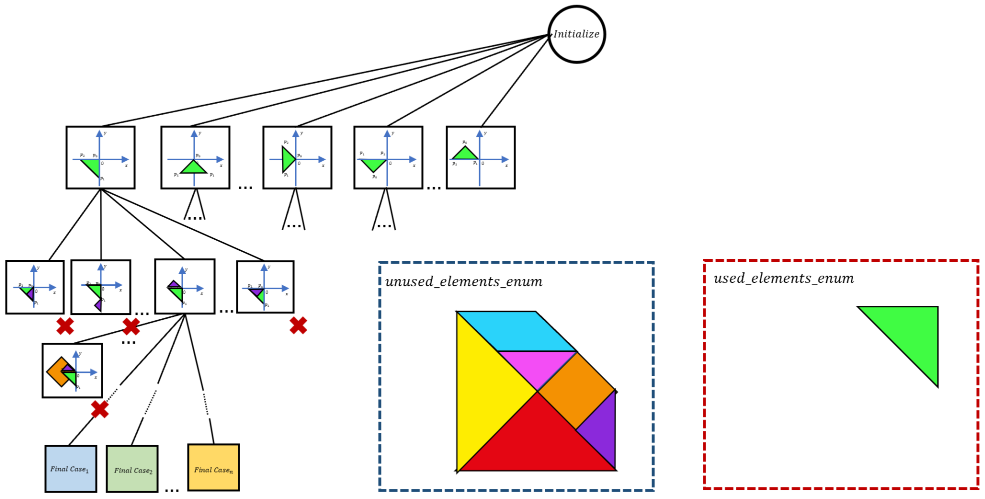


Figure 28: Iterate through all green triangle cases

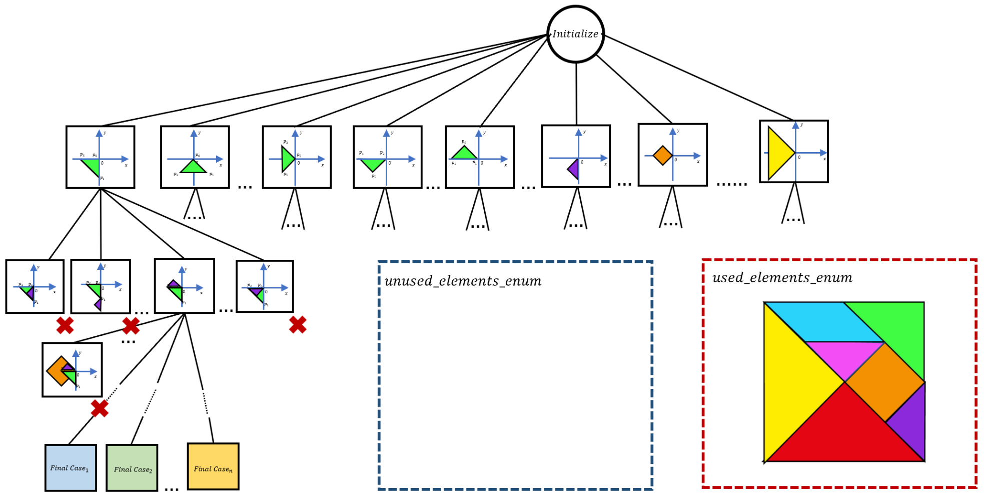


Figure 29: Iterate through all tangram pieces cases

3.4 The specific code implementation

a. Initalize

```
class Solve:

    def __init__(self):
        self.corners = []
        self.error_threshold = 0.01
        self.unit_length = 620 / 8
        self.concave_convex_enum = ConcaveConvexEnum.convex
        # mark some elements of Tangram
```

```python
        self.unused_elements_enum = [
            TangramElementEnum.medium_triangle,
            TangramElementEnum.square,
            TangramElementEnum.small_triangle,
            TangramElementEnum.large_triangle,
            TangramElementEnum.small_triangle,
            TangramElementEnum.large_triangle,
            TangramElementEnum.parallelogram
        ]
        self.used_elements = []
```

Next, we need to iterate over the vertices, so we define the object corner as the set of multiple polygon vertices.

```python
    def __update_corners(self):
        self.corners = []
        # corners --> multi-vertices
        all_element_edges = []
        angle_at_point = {}
        self.concave_convex_enum = ConcaveConvexEnum.convex
        for element in self.used_elements:
            all_element_edges.extend(element.edges)
            points = element.points
            for point in points:
                angle = element.get_angle_at_point(point)
                if point not in angle_at_point:
                    angle_at_point[point] = 0.
                angle_at_point[point] += angle
        for k, v in angle_at_point.items():
            if abs(v - PI) < self.error_threshold and abs(v - 2 * PI)
            < self.error_threshold:
                continue
            for e in all_element_edges:
                if e.point_is_on_segment(k, self.error_threshold, end_point=False):
                    angle_at_point[k] += PI
                    break

        for k, v in angle_at_point.items():
            if v > PI and abs(v - 2 * PI) > self.error_threshold:
                self.concave_convex_enum = ConcaveConvexEnum.concave
            if abs(v - PI) < self.error_threshold or abs(v - 2 * PI)
            < self.error_threshold:
```

b. The BFS method and the DFS method

In our algorithm, the BFS method uses the queue structure to be implemented, and the DFS method uses the stack structure to be implemented.

```
class BFS(Process):
    def __init__(self, pentagon_type, result_queue):
        super(BFS, self).__init__()
        self.__result = result_queue
        self.__pentagon_type = pentagon_type
        self.__result_queue = Queue(-1)
        self.__queue = Queue(-1)
        self.__remove_dup_queue = Queue(-1)
        self.__multiprocess_solver = []
        self.__remove_dup_process = None

    def run(self):
        begin = Solve()
        begin.init()
        self.__queue.put(begin)

        self.__remove_dup_process =
        BFS.RemoveDupProcess(self.__remove_dup_queue, self.__queue)
        self.__remove_dup_process.start()

        cpu_count = multiprocessing.cpu_count()-1
        if multiprocessing.cpu_count()<9 else multiprocessing.cpu_count()-3
        for i in range(cpu_count):
            solver_process = BFS.BFSSolverProcess
            (self.__queue, self.__result_queue, self.__remove_dup_queue)
            solver_process.start()
            self.__multiprocess_solver.append(solver_process)

        time.sleep(5)
```

```python

        while True:
            if not self.__queue.empty() or not self.__remove_dup_queue.empty():
                time.sleep(2)
                continue
            time.sleep(10)
            for x in self.__multiprocess_solver:
                x.terminate()
            self.__remove_dup_process.terminate()
            break

        for x in self.__multiprocess_solver:
            x.join()
        self.__remove_dup_process.join()

        count = 1
        while not self.__result_queue.empty():
            solve = self.__result_queue.get()
            if self.__pentagon_type == 0 and solve.concave_convex_enum
            == ConcaveConvexEnum.concave:
                continue
            if self.__pentagon_type == 1 and solve.concave_convex_enum
            == ConcaveConvexEnum.convex:
                continue
            image_path = os.path.join(config.RESULT_PATH, str(count) + ".jpg")
            image_array = get_final_result(solve.used_elements)
            cv2.imwrite(image_path, image_array, [cv2.IMWRITE_JPEG_QUALITY, 100])
            count += 1
        self.__result.put("BFS finish")
```

4 Informed search method

4.1 Initalize

Just as uninformed search methods, we set two enums, the *self.unused_elements_enum* and the *self.used_-elements_enum* to store polygons.

So, during initialization, we put the above elements into the *self.unused_elements_enum* index sequence as an iterable object.

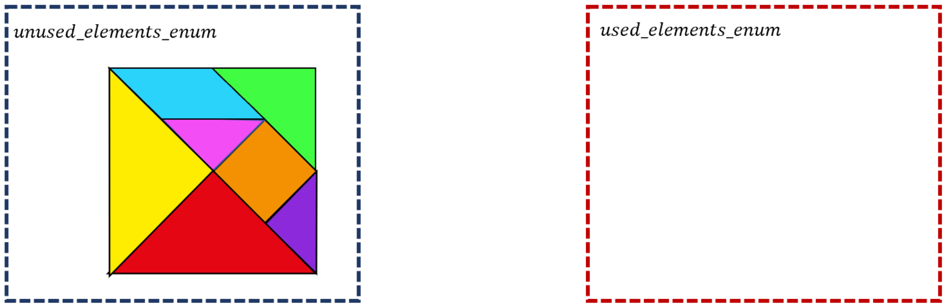


Figure 30: Informed search methods - Initalize

4.2 The Greedy method

We use the Greedy algorithm to iterate over all formations of tangram pieces.

In each iteration, the polygon whose side length is equal to the side length of the stitched polygon is selected for stitching among the unstitched polygons, after which the case with the least number of sides of the generated polygon is selected for further iterations.

We define our heuristic function, h(x)=Sides of the polygon.

a. The First level of the Greedy traversal tree

The First level of the Greedy traversal tree is shown in Fig.30. For the green triangle, we define three vertices, p_1, p_2, and p_3. We overlap each of these three vertices with the origin and rotate them by $\frac{\pi}{4}$, $\frac{\pi}{2}$, ..., 2π to traverse all cases.

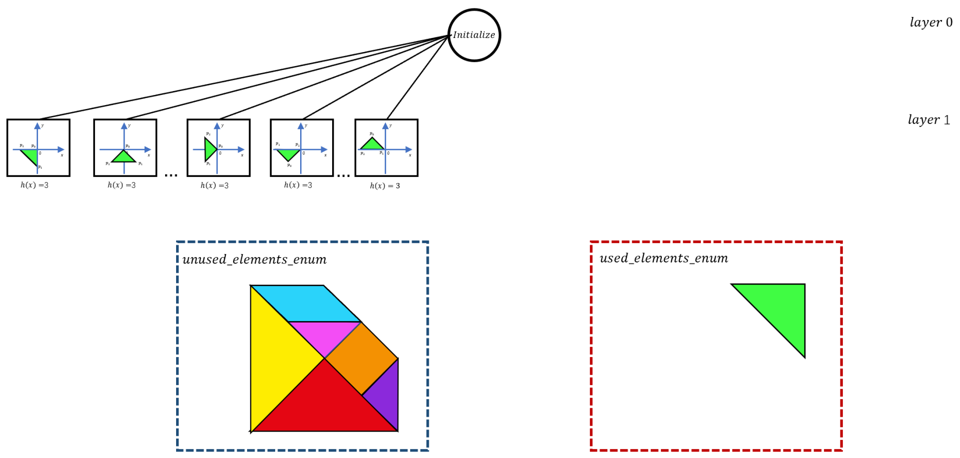


Figure 31: Iteration of the green triangle in the first layer of the Greedy traversal tree

After that, we traverse all remaining peices in the *unused_elements_enum* in the same way.

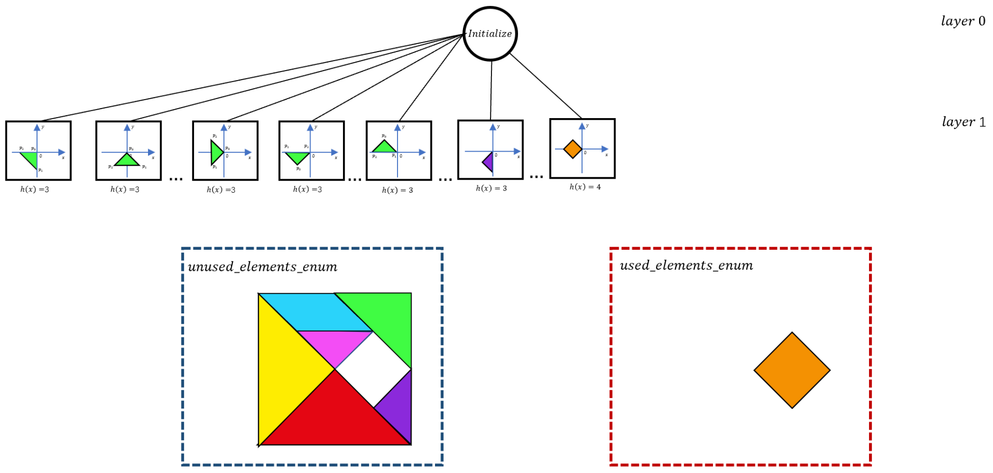


Figure 32: Traverse all remaining peices in the *unused_elements_enum*

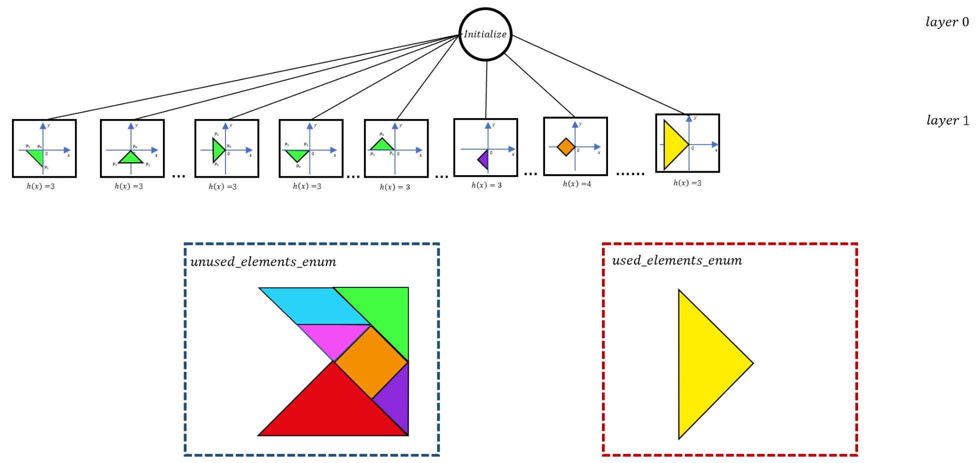


Figure 33: All have been traversed

b. The Second level and lower levels of the Greedy traversal tree

When we traverse the Second layer, we are traversing the same iterative method as the First layer and so the vertices of the polygon in the layer above. As shown in Fig. 34, at this point, we can discard some cases that do not meet the requirements, until we find one that does.

We only splice peices with the same number of sides as the current polygon.

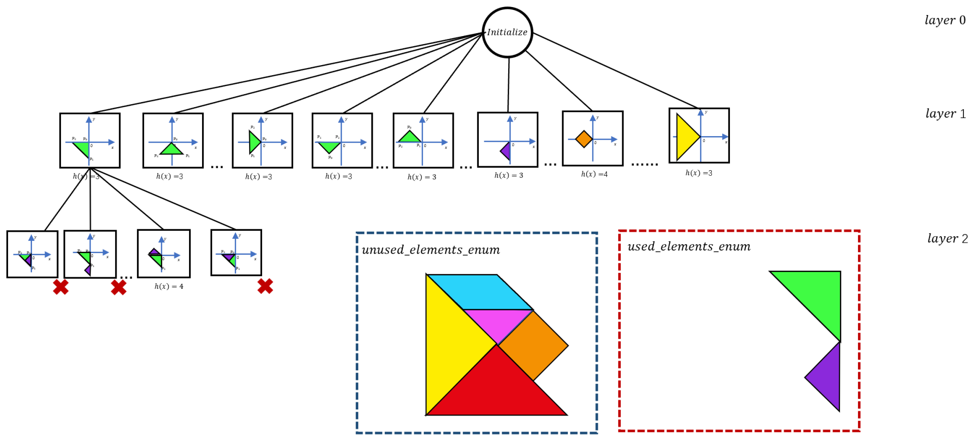


Figure 34: Iterate through all the cases of the Second layer of the First node of the First layer

After that we iterate over the Second node of the First layer in the same way.

And keep repeating the previous operations to finally complete the traversal.

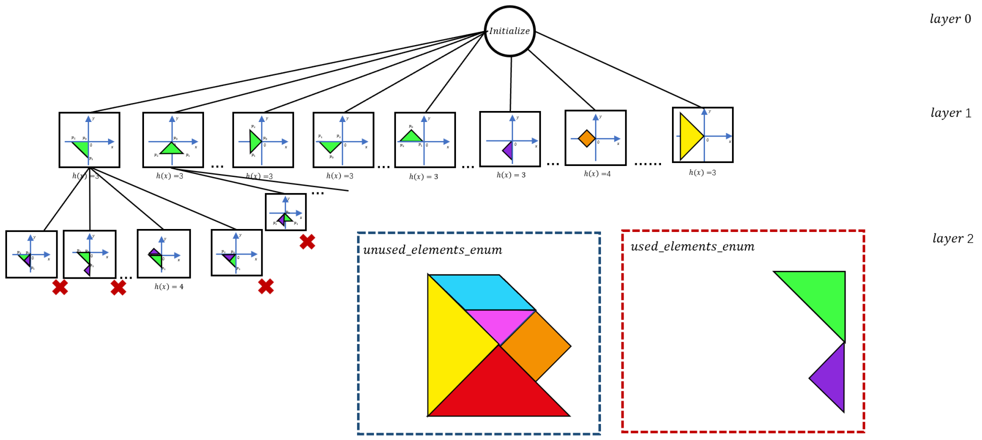


Figure 35: Iterate through all the cases of the Second layer of the Second node of the First layer

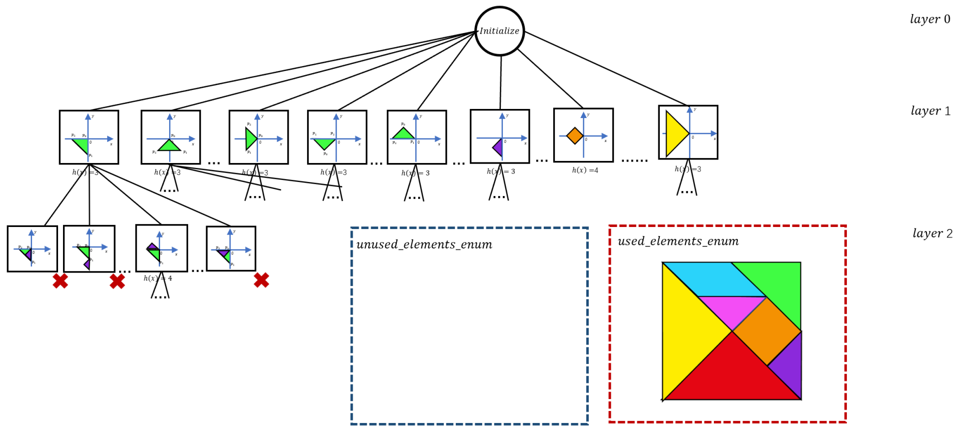


Figure 36: Iterate through all the nodes in Greedy Method

4.3 The specific code implementation

a. Initalize

```python
class Solve:

    def __init__(self):
        self.corners = []
        self.error_threshold = 0.01
        self.unit_length = 620 / 8
        self.concave_convex_enum = ConcaveConvexEnum.convex
        # mark some elements of Tangram
        self.unused_elements_enum = [
            TangramElementEnum.medium_triangle,
            TangramElementEnum.square,
            TangramElementEnum.small_triangle,
            TangramElementEnum.large_triangle,
            TangramElementEnum.small_triangle,
            TangramElementEnum.large_triangle,
            TangramElementEnum.parallelogram
        ]
        self.used_elements = []
```

b. The Greedy method

```
class Greedy(Process):
    def __init__(self, pentagon_type, result_queue):
        super(Greedy, self).__init__()
        self.__begin_element = [TangramElementEnum.medium_triangle,
                                TangramElementEnum.square,
                                TangramElementEnum.small_triangle,
                                TangramElementEnum.parallelogram]
        self.__queue = []
        self.__result = result_queue
        self.__pentagon_type = pentagon_type
        self.__solved = []

    def run(self):
        for element in self.__begin_element:
            begin = Solve()
            begin.init(element)
            self.__queue.append(begin)

        while len(self.__queue) > 0:
            solve = self.__queue.pop(0)
            proper_element = solve.get_proper_element()
            if len(solve.unused_elements_enum) == 0:
                self.__solved.append(solve)
                continue
            if len(proper_element)>0:
                for ele, e in proper_element:
                    temp_solve_list = solve.try_element_extra(ele, e)
                    for temp_solve in temp_solve_list:
```

```
                    if len(temp_solve.edges) > 6:
                        continue
                    self.__queue.append(temp_solve)
        else:
            if len(solve.used_elements)<5:
                continue
            for element_enum in solve.unused_elements_enum:
                for corner in range(len(solve.corners)):
                    for state in range
                    (TRIANGLE_ELEMENT_STATE_DIC[element_enum]):
                        for angle in range(ANGLE_NUM):
                            new_solve = solve.try_element
                            (element_enum=element_enum,
                            corner=corner, state=state,angle=angle)
                            if new_solve is None or \
                                    len(new_solve.corners) > 8 or \
                                    (len(new_solve.used_elements) > 3 and
                                     len(new_solve.corners)
                                     > len(new_solve.used_elements) + 1):
                                continue
                            self.__queue.append(new_solve)
```

5 Determine if the generated polygon is a pentagon

5.1 Judgment method

Directly calculate the number of vertices of the stitching result.

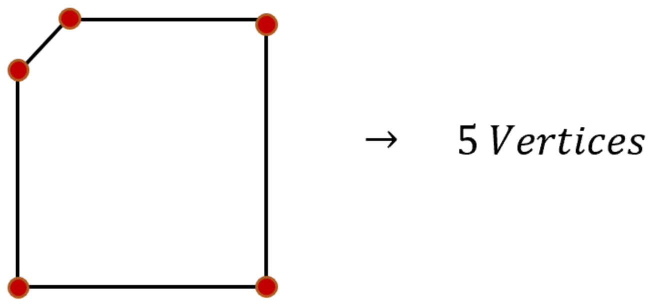


Figure 37: Pentagon

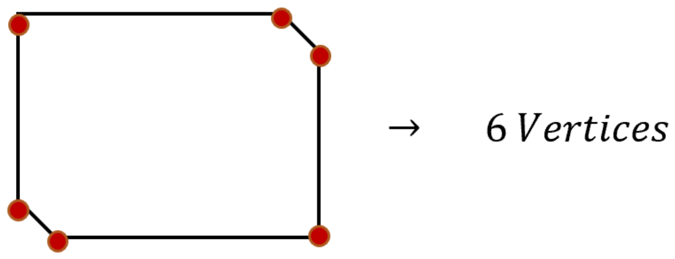


Figure 38: Hexagon

6 The result

Since uninformed requires too much memory, we used 4 pieces from Tangram pieces as a demo to prove the correctness of our algorithm.

6.1 Concave pentagon

i. Uninformed method

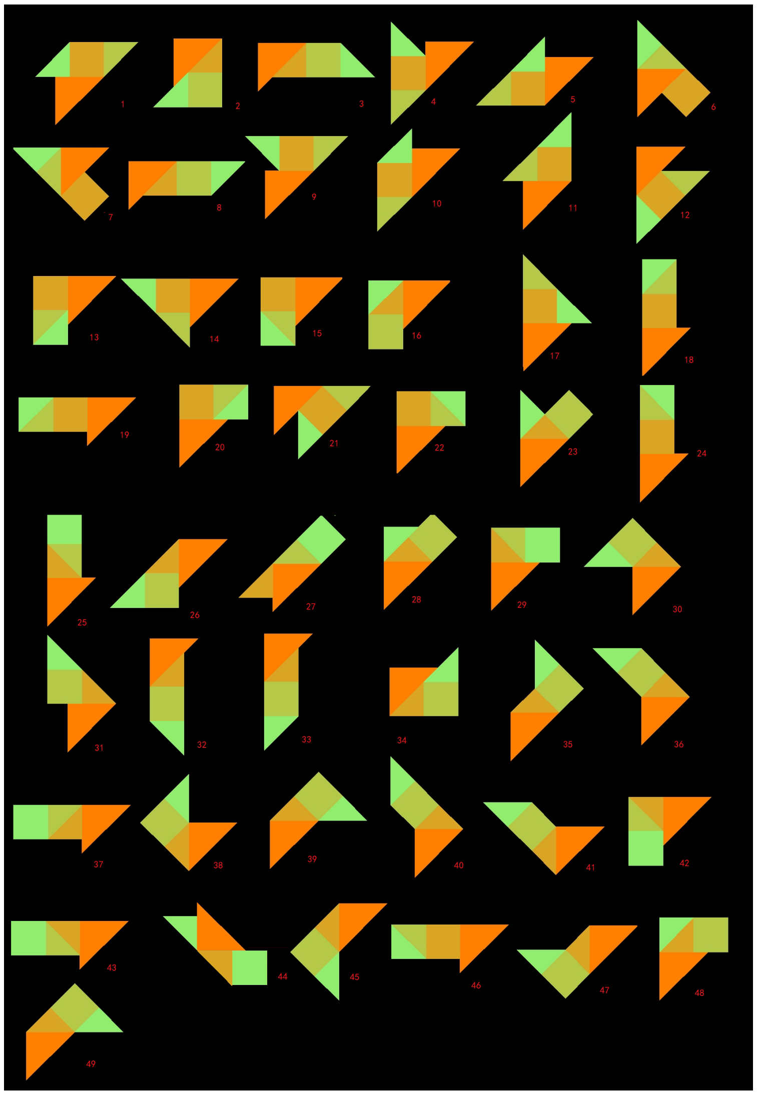


Figure 39: Uninformed method

ii. Informed method

6.2 Convex pentagon

i. Uninformed method

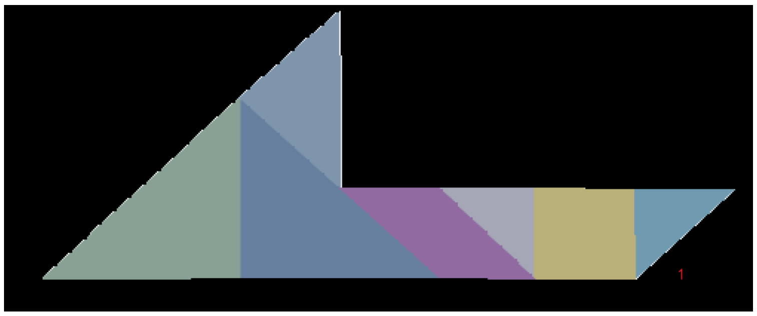


Figure 40: Informed method

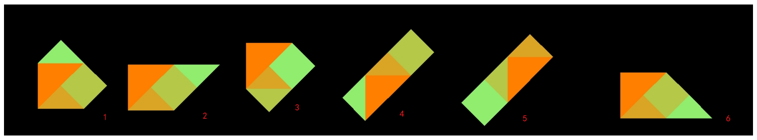


Figure 41: Uninformed method

ii. Informed method

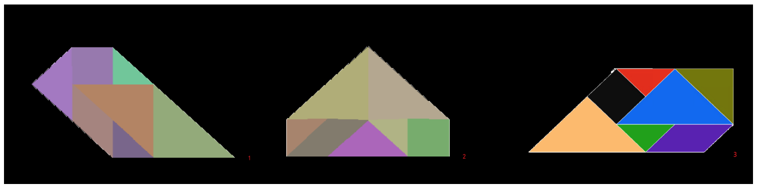


Figure 42: Informed method

7 Our system and contributions

7.1 Time Complexity Analysis

For BFS search, it's time complexity is: T(n) = O(n^7 * Number of rotation angles * Maximum multilateral length * Number of tangram pieces) = O(224 n^7).

For DFS search, it's time complexity is: T(n) = O(n^7 * Number of rotation angles * Maximum multilateral length * Number of tangram pieces) = O(224 n^7).

For Greedy search, it's time complexity is: T(n) = O(n^7 * Number of rotation angles * Maximum multilateral length * Number of tangram pieces) = O(224 n^7).

Table 2: Running Result

Method	Time Complexity
breadth-first search	O(224 n^7)
depth-first search	O(224 n^7)
Greedy search method	O(224 n^7)

7.2 Advantages

With the BFS, DFS algorithm, we can traverse all compositions of tangram peices, after which we can get the result we want without any leftovers.With the Greedy Search algorithm, we can quickly get the results we are looking for.

In addition, in terms of code, we used multi-threaded programming to implement it. Compared to single-threaded programming, it can significantly speed up our search in a multi-core CPU scenario.

7.3 Disadvantages

Our two Uninformed methods are still essentially violent search methods, which are not intelligent enough. The Informed method we use, Greedy Search, is fast enough but yields few results due to its nature of being obsessed with efficiency, it aborts many possible scenarios.

In addition, it is basically impossible to run our program on laptops and home PCs because the software takes up a huge amount of computing resources and memory causing the computer to crush.

8 The guideline for our software

8.1 Install Anaconda

a. Go to Anaconda [https://www.anaconda.com], click ”Download”.

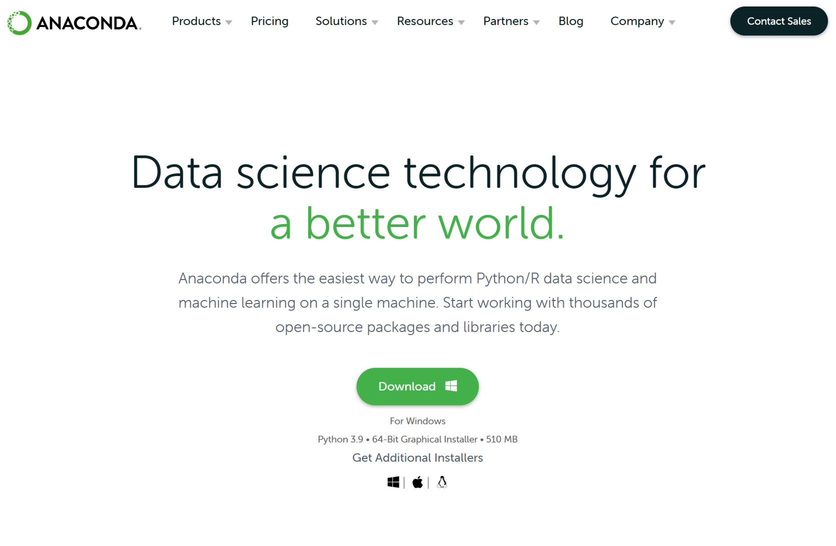
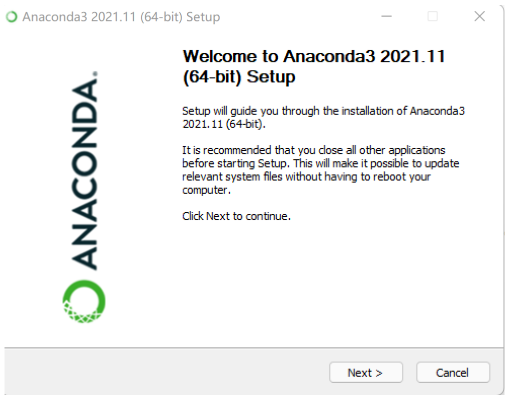

ANACONDA.
Products
Pricing
Solutions
Resources
Partners
Blog
Company
Contact Sales
Data science technology for
a better world.
Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.
Download
For Windows
Python 3.9 • 64-Bit Graphical Installer • 510 MB
Get Additional Installers

Figure 43: Anaconda

b. After that, open the downloaded installer and keep clicking the ”Next” button to complete the installation.

Figure 44: Installer



Anaconda3 2021.11 (64-bit) Setup

Welcome to Anaconda3 2021.11 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2021.11 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 45: Clicking the ”Next” button

8.2 Configuring the Python environment

a. Go to Anaconda Prompt

```
(base) C:\Users\HUAWEI>
```

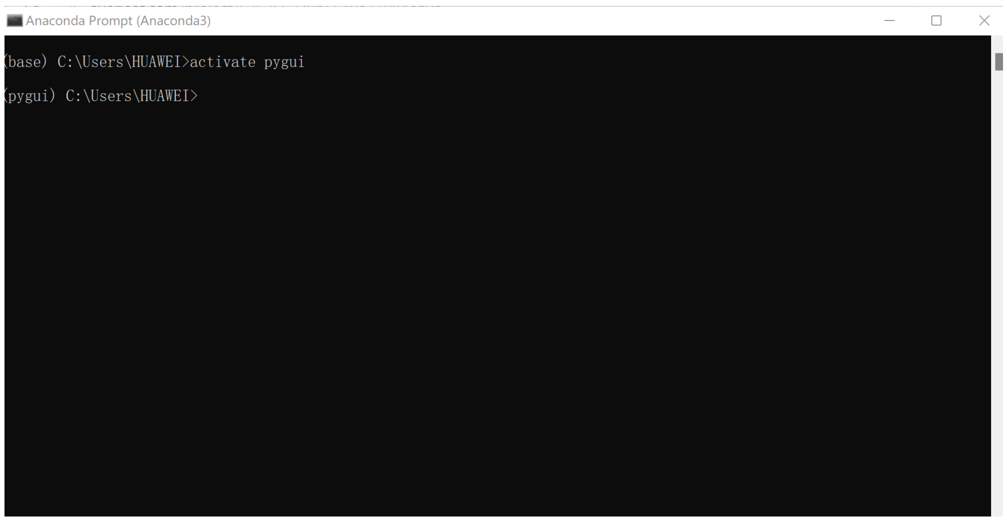
Figure 46: Anaconda Prompt

b. Enter the following commands

```
conda create -n pygui python=3.7.9
pip install PyQt5
pip install PyQt5-tools
pip install numpy==1.22.3
pip install opencv-python==4.5.5.64
pip install Pillow==9.1.0
```

8.3 Running software

a. Enter the ”activate pygui” in Anaconda Prompt to go into the pygui environment.

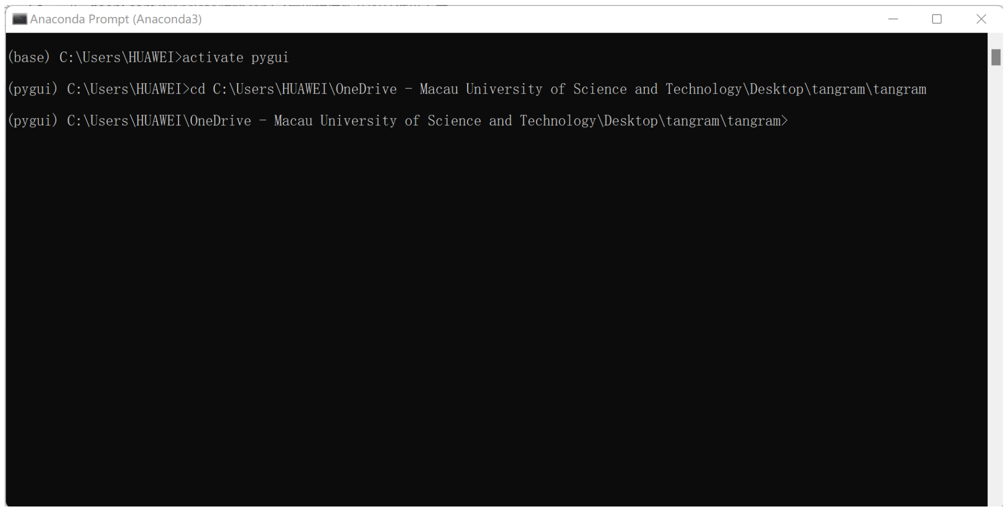


Anaconda Prompt (Anaconda3)

```
(base) C:\Users\HUAWEI>activate pygui

(pygui) C:\Users\HUAWEI>
```

Figure 47: Go into the pygui environment

b. Then go to the folder where the program is located

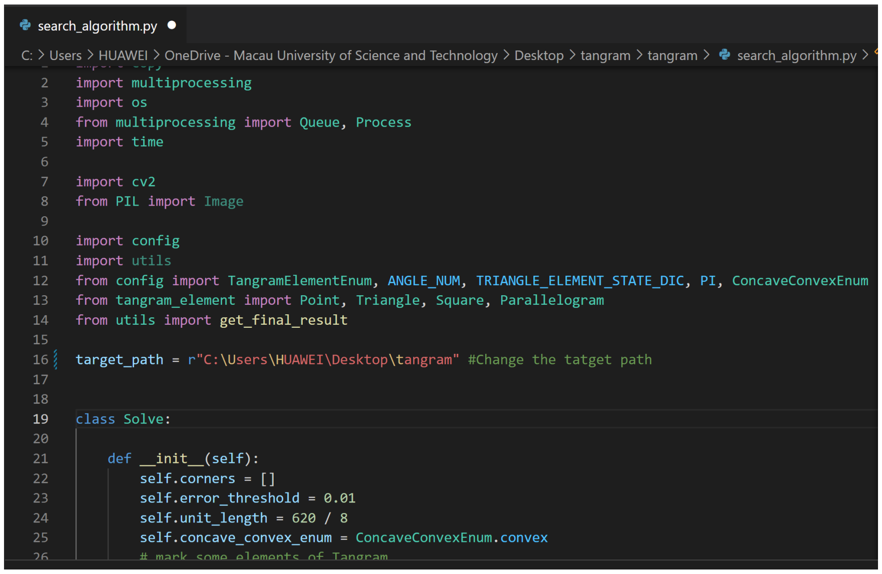


Anaconda Prompt (Anaconda3)

```
(base) C:\Users\HUAWEI>activate pygui

(pygui) C:\Users\HUAWEI>cd C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram

(pygui) C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram>
```

Figure 48: Go to the folder where the program is located

c.Go to the search_algorithm.py and config,py in the tangram folder and change the path to the current directory.

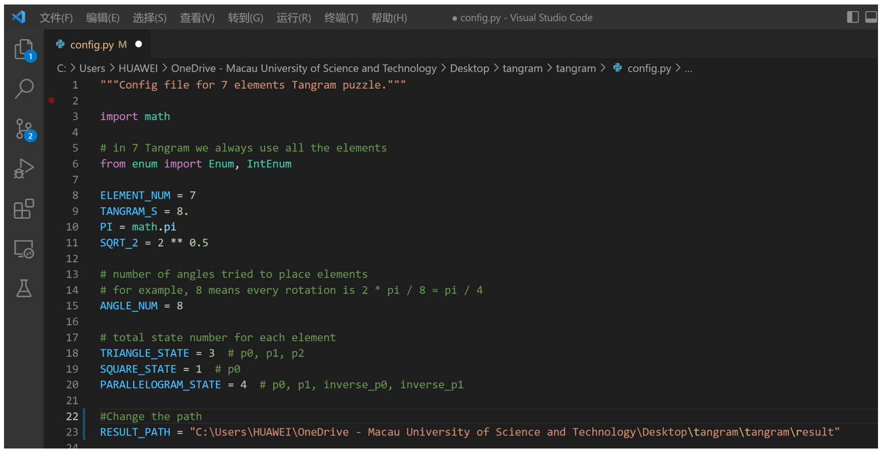


search_algorithm.py ●

C: > Users > HUAWEI > OneDrive - Macau University of Science and Technology > Desktop > tangram > tangram > search_algorithm.py >

```python
import multiprocessing
import os
from multiprocessing import Queue, Process
import time

import cv2
from PIL import Image

import config
import utils
from config import TangramElementEnum, ANGLE_NUM, TRIANGLE_ELEMENT_STATE_DIC, PI, ConcaveConvexEnum
from tangram_element import Point, Triangle, Square, Parallelogram
from utils import get_final_result

target_path = r"C:\Users\HUAWEI\Desktop\tangram" #Change the tatget path


class Solve:

    def __init__(self):
        self.corners = []
        self.error_threshold = 0.01
        self.unit_length = 620 / 8
        self.concave_convex_enum = ConcaveConvexEnum.convex
        # mark some elements of Tangram
```

Figure 49: Edit the TARGET path



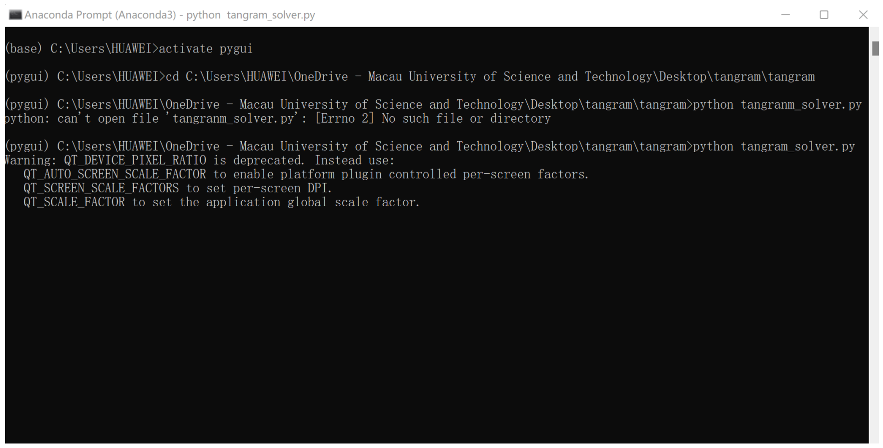
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
● config.py - Visual Studio Code
config.py M ●

C: > Users > HUAWEI > OneDrive - Macau University of Science and Technology > Desktop > tangram > tangram > config.py > ...

```python
"""Config file for 7 elements Tangram puzzle."""

import math

# in 7 Tangram we always use all the elements
from enum import Enum, IntEnum

ELEMENT_NUM = 7
TANGRAM_S = 8.
PI = math.pi
SQRT_2 = 2 ** 0.5

# number of angles tried to place elements
# for example, 8 means every rotation is 2 * pi / 8 = pi / 4
ANGLE_NUM = 8

# total state number for each element
TRIANGLE_STATE = 3  # p0, p1, p2
SQUARE_STATE = 1  # p0
PARALLELOGRAM_STATE = 4  # p0, p1, inverse_p0, inverse_p1

#Change the path
RESULT_PATH = "C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram\result"
```

Figure 50: Edit the RESULT path

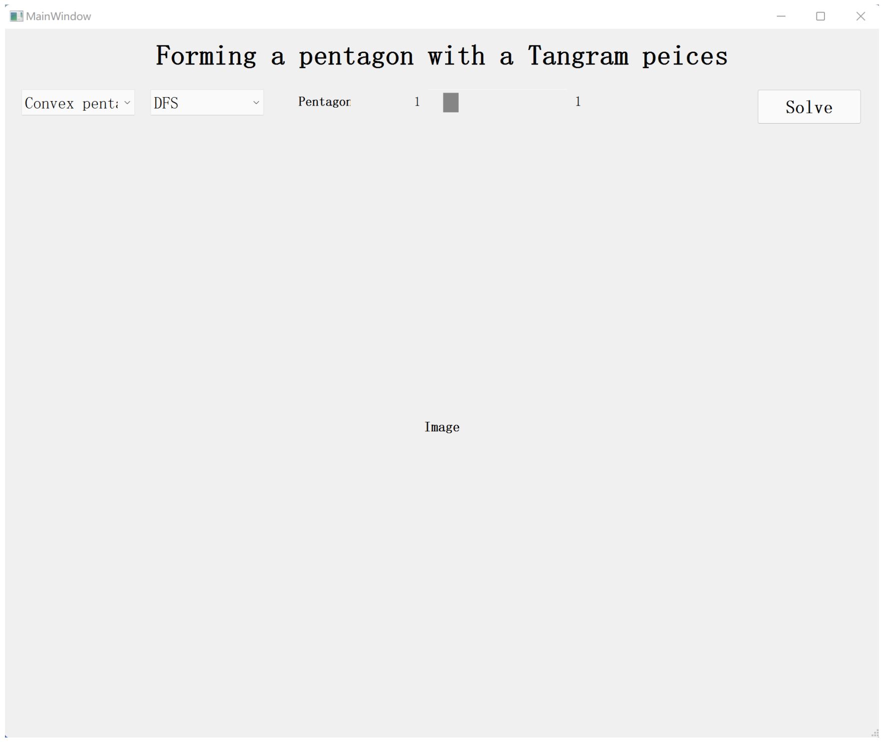
d. Enter the "python tangram_solver.py" in Anaconda Prompt to execute our program.



```
Anaconda Prompt (Anaconda3) - python  tangram_solver.py

(base) C:\Users\HUAWEI>activate pygui

(pygui) C:\Users\HUAWEI>cd C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram

(pygui) C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram>python tangranm_solver.py
python: can't open file 'tangranm_solver.py': [Errno 2] No such file or directory

(pygui) C:\Users\HUAWEI\OneDrive - Macau University of Science and Technology\Desktop\tangram\tangram>python tangram_solver.py
Warning: QT_DEVICE_PIXEL_RATIO is deprecated. Instead use:
   QT_AUTO_SCREEN_SCALE_FACTOR to enable platform plugin controlled per-screen factors.
   QT_SCREEN_SCALE_FACTORS to set per-screen DPI.
   QT_SCALE_FACTOR to set the application global scale factor.
```

Figure 51: Execute our program

f. After that you can choose the method on the UI interface for the formation of the pentagon.



MainWindow

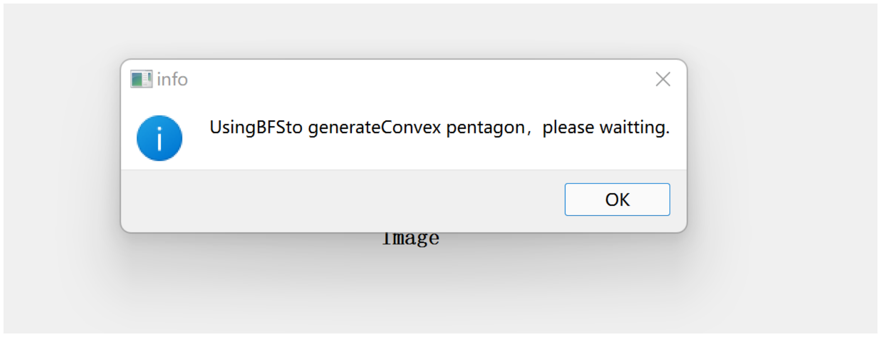
Forming a pentagon with a Tangram peices

Convex pent: | DFS | Pentagor 1 | 1 | Solve

Image

Figure 52: UI interface

g. Clicking the ”Solve” button to start searching.


info

UsingBFSto generateConvex pentagon，please waitting.

Figure 53: Start searching

References

[1]. 秋葉拓哉, 岩田陽一, 北川宜稔. (2012). プログラミングコンテストチャレンジブック: 問題解決のアルゴリズム活用力とコーディングテクニックを鍛える. マイナビ出版.

[2]. 寺田文行. (2012). 新版 演習線形代数. サイエンス社出版.

[3]. Saranli, A., Russel, S., Norvig, P. (2003). Artificial intelligence: a modern approach.

[4]. Solem, J. E. (2012). Programming Computer Vision with Python: Tools and algorithms for analyzing images. " O'Reilly Media, Inc.".

[5]. Summerfield, M. (2013). Python in practice: create better programs using concurrency, libraries, and patterns. Addison-Wesley.