CS464 Term Project Final Report

Date: December 22, 2019

Instructor: Abdullah Ercüment Çiçek

Assistant: Furkan Özden

Group 15

21600907 Taşçıoğlu Ayça Begüm

21602202 Tınaz Ege

21501975 Dikenelli Emre

21401310 Selver Emre

21300803 Savcı Can

1 Introduction and Problem Definition

Military vehicles are armoured by modular structures which contain ceramic tiles that have different shapes and geometries as shown in Figure 1 to enhance the protection. There are several different ceramic tile geometries, hexagonal or square are the most used ones, depending on the structure, vehicle module is covered by either hexagonal or square tile. The goal of this project to design a sequencing algorithm to place the ceramic tiles in the most optimized way on predetermined canvas.

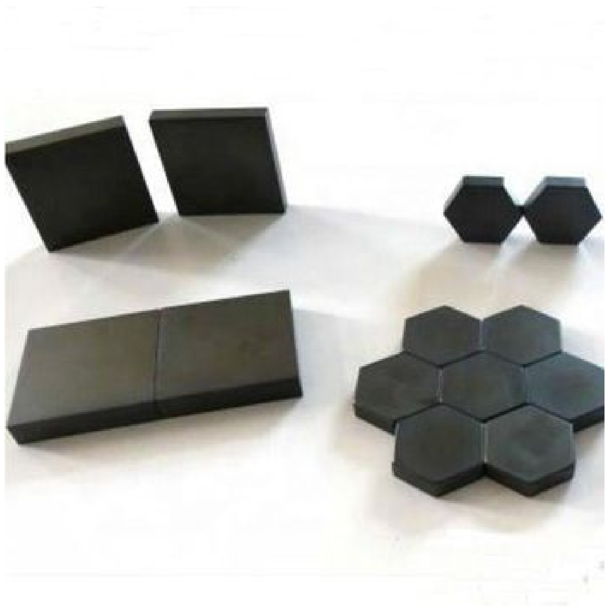

Figure 1. Hexagonal and Square Ceramic Tiles[1]

First, we write our program using TKinter library to display the tile arrangement and implement Q learning algorithm to find a pattern with minimum remaining area. Then to improve our program we use OpenAI Gym library to perform reinforcement learning on our custom environment. The custom environment created based on game of placing hexagons and squares using pygame library.

2 Methods

Reinforcement learning is the main method of this project, in the first step we implement the Q learning brute force to our concept and on the next step we use OpenAI Gym to solve the placement game on pygame environment.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a network that reacts to current state of the environment and controls the actions of a solver in order to maximize reward [2]. Reward is assigned in to obtain desired result. Fundamental concept of reinforcement learning is shown in Figure 1. Two main elements of RL is environment and agent. In the environment application is going to be created and environment engine is going to output a state and reward using an interpreter algorithm. Furthermore, the agent will perform actions in the environment and reward is yielded based on the actions of agent.

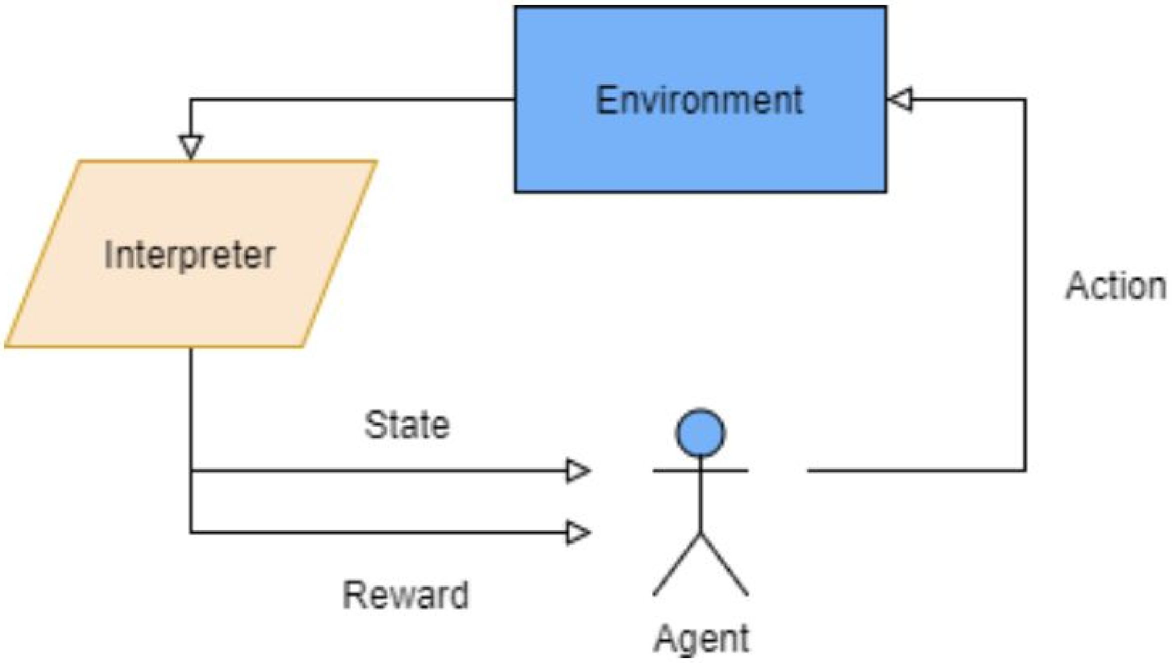

Figure 2. Reinforcement Learning schema [2]

Main goal of the agent is to process actions that yields the highest reward feedback. Agent will learn the interactions between steps by comparing the consecutive rewards. That is, reinforcement learning is based on feedback from previous steps. To implement this learning process a method called Q learning is going to used. Using reinforcement learning a network can learn to play games, solve physical problems etc.

This project’s goal is to cover an area using regular hexagons and squares. In this project environment is a canvas which hexagons and squares will be placed on. Agent take actions based on if consecutive point on the canvas is empty or not.
Our custom environment is randomly chosen shapes which are placed in randomly chosen coordinates(x,y) to cover the armor model.

- Our **agent** is the algorithm itself
- We have local **Q-Matrices** for each model, also a comprehensive Q-Matrix for keeping each models’ Q-Matrix.
- We took every **action** such as picking a shape and coordinate pair as an action, and appending every action taken in the model to the local matrix of that model.
- We used our remaining area as a score which is inversionally proportional to the **reward**.

After we finished our program and trained our algorithm without any library, we tried to expand our project. We tried to implement our project as a game and researched about it. . Reinforcement learning chosen to complete this task because we do not have dataset to perform supervised learning.

2.2 Q - Learning

Q learning is basically a value-based technique to supply information to the agent about which action should agents make. In a matrix where columns are actions and rows are states, one can keep track of which moves are the most advantageous [2]. So, when agent take random actions to explore the environment agent will choose the action which it learnt had outputs the highest reward in the previous instance.

The updating rule of Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

where s is current state, a is the possible action, r is the reward, γ value is delayed reward impact and maxQ(s’,a’) is the maximum possible Q value in the next step. Thus, agent starts in state s, takes action a (for instance move to left) and end up in state s’ and this rule calculates the max value of next state and update the value with the current value of state (Q(s,a)). Moreover, future action is multiplied by learning rate α.

In this project, OpenAI Gym is used as an agent to perform configuration tasks on the custom environment. Gym library implements the Q learning algorithm and builds a pattern composed of hexagons and squares with minimum empty area.

2.3 Brute Force Approach

We used tkinter library for this part, to prepare our GUI. We need a GUI to observe results, or testing our code is working. First of all, we defined our constant variables: EDGE, X_LIMIT, Y_LIMIT and TOTAL_AREA. EDGE is the edge of our ceramic tiles. Real lengths are multiplied by 5 to clearly see the shapes on the canvas. Also, the canvas which represents the door, or the hybrid armor in real life implementation, expanded at the same rate.

- Preprocessing for modeling:

```
EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425*650
window = Tk()
window.title("Work Place")
canvas = Canvas(window, width=X_LIMIT, height=Y_LIMIT)
# frame = Frame(width=500, height=200, bg='blue')
# canvas = Canvas(frame, bg='white')
canvas.pack(fill = BOTH, expand = YES)
```

After this, we define our functions to draw hexagons or the square with given coordinates(x,y) and calculating their areas.

```
def hex_points(x, y):
   points = (x+EDGE, y-EDGE, x,y, x+EDGE, y+EDGE, x+2*EDGE,y+EDGE,x+3*EDGE,y,x+2*EDGE,y-EDGE)
   return points


def hex_area(x):
   area = 0.0
   area = math.sqrt(3)*x*x/4
   return area


def square_points(x,y):
   points = (x, y, x, y+EDGE,x+EDGE,y+EDGE,x+EDGE,y)
   return points

def square_area(x):
   area = 0.0
   return x*x
```

-Design of a single Model

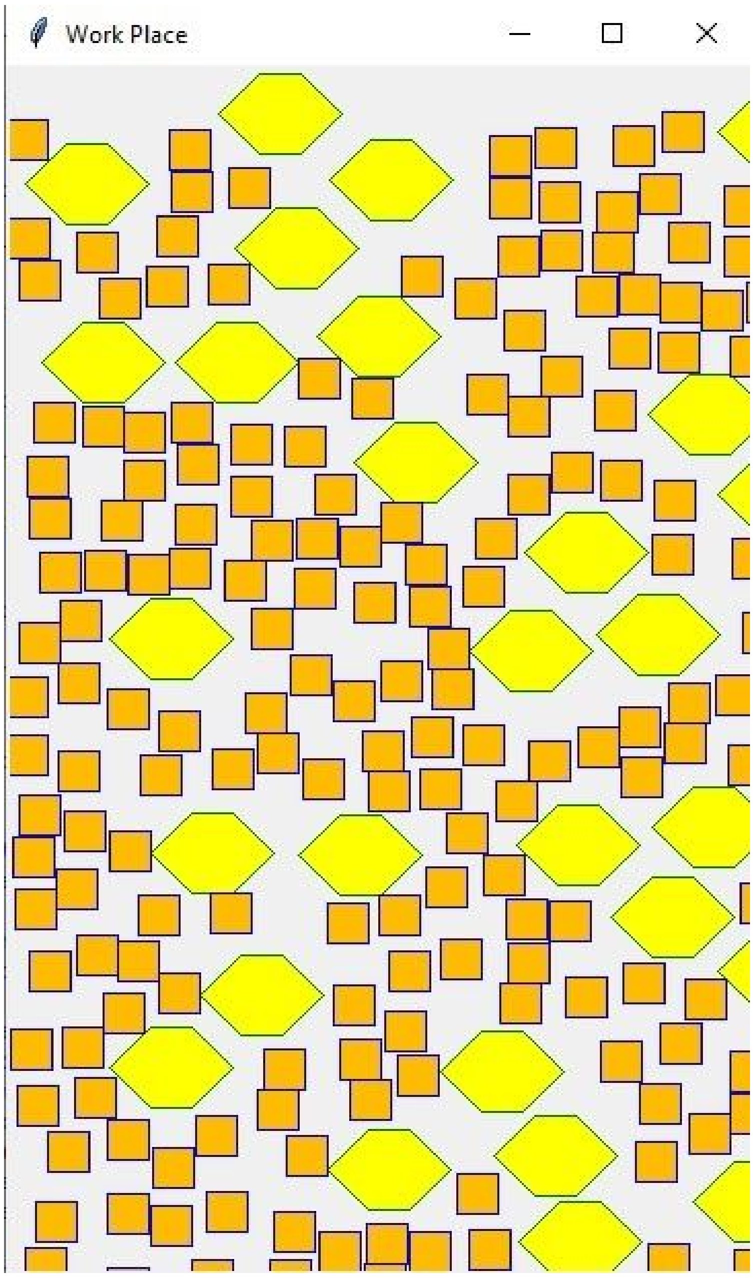

Figure 3: Model at the end

After preprocessing part, we first tried our model. Following steps are followed:
Step 1: Defining needed data structures for a single model.

```
count = 0  # number of iterations of the loop for a single model
appeared = False  # a controller flag to find whether random coordinate is already used or not
remaining_area = TOTAL_AREA  # the area which will represent the score variable for reinforcement learning
Q = []  # later process, Q matrix of the whole models for comparing actions and scores
```

```
moves = []  # keeps the data of the iteration count, shape selection and the coordinate selection
used_points = []  # keeps the data of used points(coordinates), to avoid overlapping
start_time = timeit.default_timer()  # timer initialization
```

Step 2: Selection of the shape(square or hexagon) randomly.

```
while remaining_area > 100000:
   choice = random.randint(0, 2)  # randomly selecting the shape. If choice == 1, draw a hexagon. Else, draw square
   appeared = False
   tmp_points = []
```

Step 3: Selection of the x and y coordinate randomly.

```
if choice == 1:  # If choice == 1, draw a hexagon
   print("choice 1")
   x = random.randint(0, X_LIMIT)
   y = random.randint(EDGE, Y_LIMIT)
```

Step 4: Drawing the randomly selected shape in randomly selected coordinate. If the random coordinates are not used. If there is a shape in the randomly selected coordinates, do nothing. Else draw and add these random coordinates to used_points list.

```
x = random.randint(0, X_LIMIT)
y = random.randint(EDGE, Y_LIMIT)

#rectangular area
for i in range(x+EDGE, x+2*EDGE):
   for j in range(y-EDGE, y+EDGE):
       tmp_points.append((i,j))

#triangular area-up left
tmp_k = x
for j in range(y,y+EDGE):
   for i in range(tmp_k, x+EDGE):
       tmp_points.append((i, j))
   tmp_k+=1

# triangular area-down left
tmp_k = x
for j in range(y,y-EDGE,-1):
   for i in range(tmp_k, x+EDGE):
       tmp_points.append((i, j))
   tmp_k+=1

# triangular area-up right
tmp_k = x+2*EDGE
for j in range(y, y + EDGE):
   for i in range(tmp_k, x + 3*EDGE):
       tmp_points.append((i, j))
   tmp_k += 1
```

```
# triangular area-down right
tmp_k = x + 2 * EDGE
for j in range(y, y -EDGE,-1):
    for i in range(tmp_k, x + 3 * EDGE):
        tmp_points.append((i, j))
    tmp_k += 1

for i in used_points:
    for j in tmp_points:
        if j == i:
            appeared = True
if not appeared:
    hexagon_points = hex_points(x, y)  # range of hexagon (x>=0, y>=20)
    a = canvas.create_polygon(hexagon_points, outline='green', fill='yellow', width=1)
    print("hex created")
    for i in tmp_points:
        used_points.append(i)
```

Step 5: Keep the data of the iteration counts, shape selections, coordinate selections moves, and the remaining are of a single model.

```
remaining_area -= hex_area(EDGE)
moves.append((count, choice, (x, y)))
print("remaining area", remaining_area)
```

Step 6: Use these shape and coordinate selections as actions in Q-Matrix, remaining are as the score.

-Design of Models and Comparison

```
import random
import math
import timeit

EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425 * 650


# frame = Frame(width=500, height=200, bg='blue')
# canvas = Canvas(frame, bg='white')
# frame.pack(fill = BOTH, expand = YES)


def hex_points(x, y):
    points = (
    x + EDGE, y - EDGE, x, y, x + EDGE, y + EDGE, x + 2 * EDGE, y + EDGE, x + 3 * EDGE, y, x + 2 * EDGE, y
- EDGE)
    return points


def hex_area(x):
    area = 0.0
```

```
  area = math.sqrt(3) * x * x / 4
  return area


def square_points(x, y):
  points = (x, y, x, y + EDGE, x + EDGE, y + EDGE, x + EDGE, y)
  return points


def square_area(x):
  area = 0.0
  return x * x


start_time = timeit.default_timer()  # timer initialization
# Q = []  # later process, Q matrix of the whole models for comparing actions and scores
# model_count = 0
# best_actions_states = []
# rem_areas = []  #dummy list just keptfor time efficiency
k = [2, 5, 10, 50, 100]
for j in range(10, 100, 10):
  Q = []  # later process, Q matrix of the whole models for comparing actions and scores
  model_count = 0
  best_actions_states = []
  rem_areas = []  # dummy list just keptfor time efficiency
  print("-------------------------------------------------------------------------------------------")
  print("k = ", j)
  for i in range(0, j):
    moves = []  # keeps the data of the iteration count, shape selection and the coordinate selection
    used_points = []  # keeps the data of used points(coordinates), to avoid overlapping
    count = 0  # number of iterations of the loop for a single model
    appeared = False  # a controller flag to find whether random coordinate is already used or not
    remaining_area = TOTAL_AREA  # the area which will represent the score variable for reinforcement
learning

    print("For Model number:", model_count + 1)
    if model_count > 1:
      index = rem_areas.index(min(rem_areas))  # find the position of bestmodel
      # print("index",index)
      # print("q-len",len(Q))
      # print("Q[index][0]",(Q[index])[0])
      moves.append((Q[index])[0:model_count - 1])
      print("Best model so far: ", index, "th model")

      # print("moves",moves)
      for i in range(0, model_count - 1):
        remaining_area = (moves[0])[-1][3]
      count += model_count
      print("Beginning from", count, "th iteration")
    while count < 100:
      choice = random.randint(0, 2)
      appeared = False
      tmp_points = []  # tmp points stores the randomly selected points and based on the shape, other
coordinates that will added to this list
```

```
        if choice == 1:
            # print("choice 1")
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)

            # rectangular area
            for i in range(x + EDGE, x + 2 * EDGE):
                for j in range(y - EDGE, y + EDGE):
                    tmp_points.append((i, j))

            # triangular area-up left
            tmp_k = x
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down left
            tmp_k = x
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-up right
            tmp_k = x + 2 * EDGE
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down right
            tmp_k = x + 2 * EDGE
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            for i in used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                hexagon_points = hex_points(x, y)  # range of hexagon (x>=0, y>=20)
                # print("hex created")
                for i in tmp_points:
                    used_points.append(i)
                remaining_area -= hex_area(EDGE)
                moves.append((count, choice, (x, y), remaining_area))

                # print("remaining area", remaining_area)
        elif choice == 0:
            # print("choice 0")
            x = random.randint(0, X_LIMIT)
```

```
                y = random.randint(EDGE, Y_LIMIT)
                for i in range(x, x + EDGE):
                    for j in range(y, y + EDGE):
                        tmp_points.append((i, j))
                for i in used_points:
                    for j in tmp_points:
                        if j == i:
                            appeared = True
                if not appeared:
                    s_points = square_points(x, y)
                    # print("square created")
                    for i in range(x, x + EDGE):
                        for j in range(y, y + EDGE):
                            used_points.append((i, j))
                    remaining_area -= square_area(EDGE)
                    moves.append((count, choice, (x, y), remaining_area))

                    # print("remaining area", remaining_area)
            count += 1
            # print("count",count)
        rem_areas.append(remaining_area)

        Q.append(moves)
        print("Q-Matrix (state,shape choice,coordinate pair choice, remaining area)\n", Q)
        # print("moves",moves)
        # print("Q",Q)
        # print("Q[0]",Q[0])
        # print("Q[0][1]",(Q[0])[1])
        # print("Q[0][2]",Q[0][2])
        # print("Q[0][3]",Q[0][3])
        model_count += 1
        print(("For given k = {} best model's remaining area: {}").format(k, rem_areas))
        # canvas.delete("all")

    # count = 1
    # for i in model_analytic:
    #     print(count, "th model:")
    #     print("Remaining Area:",i[0])
    #     count+=1
print("\n-----------------------------------------------END---------------------------------------------")
print("Best model remaining area:", rem_areas)
elapsed = timeit.default_timer() - start_time
print(elapsed)
```

2.4 OpenAI Gym

Gym is a library for developing RL algorithms, it is essentially an agent, as discussed in Reinforcement Learning section, that learns how to play the game of hexagon and square placement on the custom environment.

2.5 Building Custom Environment for Reinforcement Learning

2.5.1 Custom Environment

Our custom environment is randomly chosen shapes which are placed in randomly chosen coordinates(x,y) to cover the armor model.

- Our **agent** is the algorithm itself
- We have local **Q-Matrices** for each model, also a comprehensive Q-Matrix for keeping each models' Q-Matrix.
- We took every **action** such as picking a shape and coordinate pair as an action, and appending every action taken in the model to the local matrix of that model.
- We used our remaining area as a score which is inversionally proportional to the **reward**.
- After we finished our program and trained our algorithm without any library, we tried to expand our project. We tried to implement our project as a game and researched about it.

We tried to do a custom tetris environment implementation of the OpenAI's gym library. In that version the agent, reward function, Q matrix, and the action set can be taken from the library. However, we decided that we should develop our brute force approach well, because, if we used the library, we could finish our project with very few labour. Yet, we tried to implement both of these approaches.

After our discussions and implementation of our programs brute force approach, we decided to develop our custom environment with OpenAI-gym library as a game. A similar approach for our project that we found is, Tetris implementation. In that way, remaining area -which stands for the score (but negatively) - will be reduced too. First of all, before the tetris implementation, we tried to create our game engine. By using our code in the brute force approach:

```
import pygame,random,time,sys,math
from pygame.locals import *

EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425*650

def hex_points(x, y):
    points = [(x+EDGE, y-EDGE), (x,y), (x+EDGE, y+EDGE),
(x+2*EDGE,y+EDGE),(x+3*EDGE,y),(x+2*EDGE,y-EDGE)]
    return points


def hex_area(x):
    area = 0.0
    area = math.sqrt(3)*x*x/4
    return area
```

```
def square_points(x,y):
    points = [(x, y),( x, y+EDGE),(x+EDGE,y+EDGE),(x+EDGE,y)]
    return points


def square_area(x):
    return x*x


class GameState:
    def __init__(self):
        global FPSCLOCK,DISPLAYSURF, BASICFONT,BIGFONT
        pygame.init()
        FPSCLOCK = pygame.time.Clock()
        DISPLAYSURF = pygame.display.set_mode((425, 650))
        BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
        BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
        pygame.display.iconify()
        pygame.display.set_caption('Ceramic Tiles for Hybrid Armor')

        self.score = 0
        self.remaining_area = 0
        self.used_points = []
        self.moves = []
        self.coords = self.shape_choice()

        pygame.display.update()

    def reinit(self):
        self.score = 0
        self.remaining_area = 0
        self.used_points = []
        self.moves = []
        self.shape = self.shape_choice()

        pygame.display.update()

    def shape_choice(self):
        choice = random.randint(0, 2)
        tmp_points = [] #coordinate for the polygon
        appeared = False
        coords = []
        if choice == 0:
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)
            for i in range(x, x + EDGE):
                for j in range(y, y + EDGE):
                    tmp_points.append((i, j))
            for i in self.used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                s_points = square_points(x, y)
                coords = s_points
                #create polygon here?
                for i in range(x, x + EDGE):
                    for j in range(y, y + EDGE):
                        self.used_points.append((i, j))
                self.remaining_area -= square_area(EDGE)
                self.moves.append(( choice, (x, y)))
        elif choice == 1:
            print("choice 1")
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)

            # rectangular area
            for i in range(x + EDGE, x + 2 * EDGE):
                for j in range(y - EDGE, y + EDGE):
                    tmp_points.append((i, j))

            # triangular area-up left
            tmp_k = x
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down left
            tmp_k = x
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-up right
            tmp_k = x + 2 * EDGE
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down right
            tmp_k = x + 2 * EDGE
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            for i in self.used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                hexagon_points = hex_points(x, y)  # range of hexagon (x>=0, y>=20)
                coords = hexagon_points
                #a = canvas.create_polygon(hexagon_points, outline='green', fill='yellow', width=1)
                for i in tmp_points:
                    self.used_points.append(i)
                self.remaining_area -= hex_area(EDGE)
                self.moves.append((choice, (x, y)))
        return coords

    def getImage(self):
        image_data = pygame.surfarray.array3d(pygame.transform.rotate(pygame.display.get_surface(), 90))
        return image_data
# Surface = pygame.display.set_mode((800,600))
# BLUE = (80,208,255)
# PINK = (255,208,160)
# g = GameState()
# print(g.shape_choice())
# # pygame.draw.polygon(Surface, (BLUE), g.shape_choice())
# # run = True
# # while run:
# #     for event in pygame.event.get():
# #         if event.type == pygame.QUIT:
# #             run = False
# #     pygame.display.update()
# #     break
```

However, this engine was the biggest challenge that we encountered because, previous implementation of our program is based on random shape and coordinate selection, yet, we cannot implement this in a game engine because we want to program a game then an agent to play it. Hence, we researched and find tetris implementation and customize the “Tetris” Environment of the gym library[3]. (However, Tetris environment and engine is deprecated in gym library, we should have found the source code of gym-tetris and make minimal changes in this code. After these changes, we registered our custom environment with __init__ class.

- Changes for customizing gym-tetris

```
HEXAGON=['..OO.',
          'OOOO.',
          '..OO.',
          '.....',
          '.....']

SQUARE= ['.OO..',
         '.OO..',
     '.....',
     '.....',
     '.....']

SHAPES = {'S': SQUARE,
      'H': HEXAGON}
```

- Initialization of the Environment

```
import gym
#import pdb; pdb.set_trace()

for env in gym.envs.registration.EnvRegistry.env_specs:
    if 'Tetris' in env:
        print('Remove {} from registry'.format(env))
        del gym.envs.registration.EnvRegistry.env_specs[env]
```

3 Results

3.1 List Of Questions That Our Experiments are Designed to Answer

- How to reach the most optimized way to cover the canvas ?
- Which arrangement of tiles yields the smallest empty area ?
- How can we reach a better solution?
- Which approach would be better to implement our environment?

3.2 Details of Experiments

We did our experiment with different libraries, and designs. First of all, as written above in Chapter 2.3, we designed our model, which is our environment for a single model. The given real lengths were increased by multiplying by 5. And canvas is also increased in the same ratio.

The code in the demo:
- We tried to stop while loop for model drawing with remaining area, yet this approach was wrong since if we did in that way, models will be approximately the same. However, we want variation for training the agent and finding optimal solution with various data. Data size is quite different in Reinforcement Learning; we started with only the dimensions of the shapes and the armor. We created our own dataset by loop-which took too much time to get results-, and then tried to train our agent with best actions in our dataset.

- Action set was smaller since the control parameter was taken as the remaining area.

- We had a huge problem in our model which is that, we could not avoid overlapping, our shapes can be drawn onto another which is already drawn. After a hard discussion between Ayca Begum Tascioglu and Ege Tinaz, this problem solved.

- Constant variables such as increased dimensions of shapes and armor were same.

- We were advised to use library to avoid wasting too much time for running and collecting the outputs for data.

The last version of the code:

- After demo, we fixed the small but painful mistakes in our model such as overlapping.

- We decreased the number of loops that lead us to wait too much time for output.

- We took the state as the controller of our while loop which is implemented for models. Since we took exact lengths by increasing them with the same ratio, our random coordinate selection was harder. After we restrict our loop by state counts, our efficiency increased.

- In demo, another problem was that we could not tell clearly what is our collected data, such as which variable is Q-Matrix, score, and action. In that way, we fixed the output format.

- We tried to use OpenAI library to use agent that gym library provided, yet in the same time, we developed our last iteration of our brute force approach program too.

- For brute force part, both dynamic programming approach and greedy approach are tried to use. For dynamic programming approach, we iterated through 5000 models and tried to find the best. However, our best model, which is the set of the best actions, is not affected by the previous actions. In that way, dynamic programming approach is not used. We decided to be inspired by the greedy approach. For greedy approach, which is selecting the best/suitable solution in the dataset, and adding it to reach the next step is the way we have been looking for. Our program makes a model, and the first action in the best model found at the end, is always the first action of the first model. after that, it tries to find 2nd action by building actions on the first action (and this taking actions and building next action as an optimal solution process remains until the end of the given model count).

- In the last iteration of the code, the algorithm is tested for k = 10 to 1000 models. The best score for every step has been taken. (i.e first, we make 10 models and find optimal placement for these set which is the minimum remaining area left;

```
For given k = 10 remaining areas: [265078.71870789025,
261878.71870789025, 263693.0780618354, 261478.71870789025,
262439.4882233491, 263466.28314259223, 261185.89838486278,
260439.4882233491, 260612.69330410595, 260319.87298107846]

For given k = 10 best model’s remaining area: 260319.87298107846
```

265078.71870789025 is the first model’s remaining area, 261878.71870789025 for the second model, and 260319.87298107846 is the 10th model’s remaining area. Even for k=10 models and training remaining area could be increased by 5000.

```
For given k = 20 remaining areas: [265466.28314259223,
264105.5136271334, 266278.71870789025, 262451.9237886471,
264505.5136271334, 263705.5136271334, 262051.9237886471,
258612.6933041059, 258266.28314259215, 256653.84757729404,
254480.64249653716, 252068.206931239, 249895.0018504821,
247936.1561236701, 239828.97644669766, 235416.54088139944,
235977.3103968581, 233004.10531610122, 230538.07991231678,
224913.20878172034]

For given k = 10 best model’s remaining area: 224913.20878172034
```
Every model is built according to so far best model(best model’s actions) and then added to the data set. If the new model which is built according to so far best model is new best model, the next model will be built according to its state, shape and coordinate selections, and the actions. This process is repeated for k = 10,20,30,...,980,990,1000 models.

- Our trial for implementing our program with gym library is a continuing process for now, we needed to design our code from beginning if we want to write without any gym environments. We need to complete the engine of the game, create the environment and the __init__ function to register. However, if we use the gym environment for tetris, and just customize shapes, we could decrease the waiting time for output.

3.2.1 Output

k = 10’s output is taken exactly yet other iterations until k = 50 shortened for sake of simplicity. However included in the GitHub repository https://github.com/aeyc/CeramicTilesForHybridArmor/blob/master/out_10to50.txt

```
--------------------------------------------------------------------------------------------
k =  10
For Model number: 1
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)]]
For given k = 10 best model's remaining area: 265078.71870789025
For Model number: 2
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)]]
For given k = 10 best model's remaining area: 261878.71870789025
For Model number: 3
Best model so far:  1 th model
Beginning from 2 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)]]
For given k = 10 best model's remaining area: 263693.0780618354
For Model number: 4
Best model so far:  1 th model
Beginning from 3 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)]]
For given k = 10 best model's remaining area: 261478.71870789025
For Model number: 5
Best model so far:  3 th model
Beginning from 4 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)]]
For given k = 10 best model's remaining area: 262439.4882233491
For Model number: 6
Best model so far:  3 th model
Beginning from 5 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)], [[[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294)], (5, 0,
(268, 437), 274530.3847577294), (7, 1, (75, 111), 274357.17967697256),
(9, 1, (245, 308), 274183.9745962157), (10, 1, (167, 563),
274010.76951545884), (13, 1, (342, 541), 273837.564434702), (14, 0,
(334, 84), 273437.564434702), (15, 0, (128, 363), 273037.564434702),
(16, 0, (154, 41), 272637.564434702), (18, 1, (246, 391),
272464.3593539451), (20, 0, (264, 485), 272064.3593539451), (21, 1,
(83, 218), 271891.15427318827), (22, 0, (138, 529),
271491.15427318827), (23, 1, (139, 186), 271317.9491924314), (24, 1,
(75, 262), 271144.74411167455), (25, 0, (19, 519), 270744.74411167455),
(28, 1, (317, 253), 270571.5390309177), (29, 0, (79, 30),
270171.5390309177), (30, 1, (10, 66), 269998.3339501608), (37, 0, (142,
315), 269598.3339501608), (41, 1, (15, 395), 269425.12886940397), (45,
1, (324, 30), 269251.9237886471), (49, 0, (195, 486),
268851.9237886471), (54, 0, (163, 502), 268451.9237886471), (56, 1,
(226, 73), 268278.71870789025), (58, 1, (244, 161), 268105.5136271334),
(65, 1, (131, 414), 267932.30854637653), (68, 0, (293, 512),
267532.30854637653), (69, 0, (313, 563), 267132.30854637653), (70, 0,
(215, 459), 266732.30854637653), (71, 0, (53, 214),
266332.30854637653), (74, 1, (134, 477), 266159.1034656197), (75, 0,
(329, 370), 265759.1034656197), (77, 1, (14, 257), 265585.8983848628),
(80, 1, (45, 484), 265412.69330410595), (82, 0, (101, 568),
265012.69330410595), (84, 1, (287, 466), 264839.4882233491), (86, 0,
(279, 194), 264439.4882233491), (90, 1, (18, 579), 264266.28314259223),
(93, 0, (193, 47), 263866.28314259223), (99, 0, (112, 592),
263466.28314259223)]]
For given k = 10 best model's remaining area: 263466.28314259223
For Model number: 7
Best model so far:  3 th model
Beginning from 6 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)], [[[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294)], (5, 0,
(268, 437), 274530.3847577294), (7, 1, (75, 111), 274357.17967697256),
(9, 1, (245, 308), 274183.9745962157), (10, 1, (167, 563),
274010.76951545884), (13, 1, (342, 541), 273837.564434702), (14, 0,
(334, 84), 273437.564434702), (15, 0, (128, 363), 273037.564434702),
(16, 0, (154, 41), 272637.564434702), (18, 1, (246, 391),
272464.3593539451), (20, 0, (264, 485), 272064.3593539451), (21, 1,
(83, 218), 271891.15427318827), (22, 0, (138, 529),
271491.15427318827), (23, 1, (139, 186), 271317.9491924314), (24, 1,
(75, 262), 271144.74411167455), (25, 0, (19, 519), 270744.74411167455),
(28, 1, (317, 253), 270571.5390309177), (29, 0, (79, 30),
270171.5390309177), (30, 1, (10, 66), 269998.3339501608), (37, 0, (142,
315), 269598.3339501608), (41, 1, (15, 395), 269425.12886940397), (45,
1, (324, 30), 269251.9237886471), (49, 0, (195, 486),
268851.9237886471), (54, 0, (163, 502), 268451.9237886471), (56, 1,
(226, 73), 268278.71870789025), (58, 1, (244, 161), 268105.5136271334),
(65, 1, (131, 414), 267932.30854637653), (68, 0, (293, 512),
267532.30854637653), (69, 0, (313, 563), 267132.30854637653), (70, 0,
(215, 459), 266732.30854637653), (71, 0, (53, 214),
266332.30854637653), (74, 1, (134, 477), 266159.1034656197), (75, 0,
(329, 370), 265759.1034656197), (77, 1, (14, 257), 265585.8983848628),
(80, 1, (45, 484), 265412.69330410595), (82, 0, (101, 568),
265012.69330410595), (84, 1, (287, 466), 264839.4882233491), (86, 0,
(279, 194), 264439.4882233491), (90, 1, (18, 579), 264266.28314259223),
(93, 0, (193, 47), 263866.28314259223), (99, 0, (112, 592),
263466.28314259223)], [[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702), (12, 1,
(56, 457), 273264.3593539451), (15, 0, (235, 571), 272864.3593539451),
(17, 0, (107, 325), 272464.3593539451), (18, 0, (69, 503),
272064.3593539451), (23, 1, (177, 76), 271891.15427318827), (24, 1,
(24, 184), 271717.9491924314), (25, 0, (205, 33), 271317.9491924314),
(26, 0, (359, 559), 270917.9491924314), (28, 0, (92, 174),
270517.9491924314), (29, 0, (101, 56), 270117.9491924314), (31, 1, (9,
98), 269944.74411167455), (32, 0, (187, 506), 269544.74411167455), (34,
1, (285, 180), 269371.5390309177), (35, 1, (209, 354),
269198.3339501608), (37, 0, (145, 60), 268798.3339501608), (40, 0,
(318, 359), 268398.3339501608), (42, 0, (250, 423), 267998.3339501608),
(43, 1, (158, 284), 267825.12886940397), (44, 1, (311, 33),
267651.9237886471), (45, 1, (0, 240), 267478.71870789025), (46, 0, (99,
526), 267078.71870789025), (47, 0, (144, 253), 266678.71870789025),
(48, 0, (224, 378), 266278.71870789025), (51, 0, (325, 245),
265878.71870789025), (58, 0, (328, 225), 265478.71870789025), (62, 0,
(323, 437), 265078.71870789025), (64, 0, (33, 462),
264678.71870789025), (66, 0, (355, 90), 264278.71870789025), (69, 0,
(318, 539), 263878.71870789025), (81, 0, (309, 313),
263478.71870789025), (84, 1, (220, 142), 263305.5136271334), (86, 0,
(98, 547), 262905.5136271334), (92, 0, (306, 561), 262505.5136271334),
(93, 1, (358, 253), 262332.30854637653), (94, 1, (341, 376),
262159.1034656197), (95, 0, (354, 301), 261759.10346561967), (97, 0,
(30, 566), 261359.10346561967), (98, 1, (265, 514),
261185.89838486278)]]
For given k = 10 best model's remaining area: 261185.89838486278
For Model number: 8
Best model so far:  6 th model
Beginning from 7 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)], [[[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294)], (5, 0,
(268, 437), 274530.3847577294), (7, 1, (75, 111), 274357.17967697256),
(9, 1, (245, 308), 274183.9745962157), (10, 1, (167, 563),
274010.76951545884), (13, 1, (342, 541), 273837.564434702), (14, 0,
(334, 84), 273437.564434702), (15, 0, (128, 363), 273037.564434702),
(16, 0, (154, 41), 272637.564434702), (18, 1, (246, 391),
272464.3593539451), (20, 0, (264, 485), 272064.3593539451), (21, 1,
(83, 218), 271891.15427318827), (22, 0, (138, 529),
271491.15427318827), (23, 1, (139, 186), 271317.9491924314), (24, 1,
(75, 262), 271144.74411167455), (25, 0, (19, 519), 270744.74411167455),
(28, 1, (317, 253), 270571.5390309177), (29, 0, (79, 30),
270171.5390309177), (30, 1, (10, 66), 269998.3339501608), (37, 0, (142,
315), 269598.3339501608), (41, 1, (15, 395), 269425.12886940397), (45,
1, (324, 30), 269251.9237886471), (49, 0, (195, 486),
268851.9237886471), (54, 0, (163, 502), 268451.9237886471), (56, 1,
(226, 73), 268278.71870789025), (58, 1, (244, 161), 268105.5136271334),
(65, 1, (131, 414), 267932.30854637653), (68, 0, (293, 512),
267532.30854637653), (69, 0, (313, 563), 267132.30854637653), (70, 0,
(215, 459), 266732.30854637653), (71, 0, (53, 214),
266332.30854637653), (74, 1, (134, 477), 266159.1034656197), (75, 0,
(329, 370), 265759.1034656197), (77, 1, (14, 257), 265585.8983848628),
(80, 1, (45, 484), 265412.69330410595), (82, 0, (101, 568),
265012.69330410595), (84, 1, (287, 466), 264839.4882233491), (86, 0,
(279, 194), 264439.4882233491), (90, 1, (18, 579), 264266.28314259223),
(93, 0, (193, 47), 263866.28314259223), (99, 0, (112, 592),
263466.28314259223)], [[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702), (12, 1,
(56, 457), 273264.3593539451), (15, 0, (235, 571), 272864.3593539451),
(17, 0, (107, 325), 272464.3593539451), (18, 0, (69, 503),
272064.3593539451), (23, 1, (177, 76), 271891.15427318827), (24, 1,
(24, 184), 271717.9491924314), (25, 0, (205, 33), 271317.9491924314),
(26, 0, (359, 559), 270917.9491924314), (28, 0, (92, 174),
270517.9491924314), (29, 0, (101, 56), 270117.9491924314), (31, 1, (9,
98), 269944.74411167455), (32, 0, (187, 506), 269544.74411167455), (34,
1, (285, 180), 269371.5390309177), (35, 1, (209, 354),
269198.3339501608), (37, 0, (145, 60), 268798.3339501608), (40, 0,
(318, 359), 268398.3339501608), (42, 0, (250, 423), 267998.3339501608),
(43, 1, (158, 284), 267825.12886940397), (44, 1, (311, 33),
267651.9237886471), (45, 1, (0, 240), 267478.71870789025), (46, 0, (99,
526), 267078.71870789025), (47, 0, (144, 253), 266678.71870789025),
(48, 0, (224, 378), 266278.71870789025), (51, 0, (325, 245),
265878.71870789025), (58, 0, (328, 225), 265478.71870789025), (62, 0,
(323, 437), 265078.71870789025), (64, 0, (33, 462),
264678.71870789025), (66, 0, (355, 90), 264278.71870789025), (69, 0,
(318, 539), 263878.71870789025), (81, 0, (309, 313),
263478.71870789025), (84, 1, (220, 142), 263305.5136271334), (86, 0,
(98, 547), 262905.5136271334), (92, 0, (306, 561), 262505.5136271334),
(93, 1, (358, 253), 262332.30854637653), (94, 1, (341, 376),
262159.1034656197), (95, 0, (354, 301), 261759.10346561967), (97, 0,
(30, 566), 261359.10346561967), (98, 1, (265, 514),
261185.89838486278)], [[[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702)], (7, 1,
(153, 473), 273264.3593539451), (11, 0, (235, 443), 272864.3593539451),
(12, 0, (38, 277), 272464.3593539451), (13, 1, (59, 345),
272291.15427318827), (14, 0, (324, 185), 271891.15427318827), (16, 1,
(50, 139), 271717.9491924314), (17, 1, (95, 35), 271544.74411167455),
(19, 0, (26, 473), 271144.74411167455), (21, 1, (62, 265),
270971.5390309177), (22, 0, (343, 27), 270571.5390309177), (23, 0,
(271, 223), 270171.5390309177), (25, 1, (121, 215), 269998.3339501608),
(27, 1, (260, 290), 269825.12886940397), (28, 1, (189, 73),
269651.9237886471), (30, 0, (230, 205), 269251.9237886471), (31, 0,
(39, 339), 268851.9237886471), (32, 0, (113, 429), 268451.9237886471),
(33, 1, (307, 345), 268278.71870789025), (39, 1, (231, 542),
268105.5136271334), (42, 0, (176, 252), 267705.5136271334), (44, 0,
(106, 377), 267305.5136271334), (46, 0, (59, 382), 266905.5136271334),
(50, 0, (272, 355), 266505.5136271334), (52, 0, (261, 465),
266105.5136271334), (54, 1, (240, 593), 265932.30854637653), (58, 0,
(355, 162), 265532.30854637653), (61, 0, (98, 408),
265132.30854637653), (62, 1, (350, 258), 264959.1034656197), (65, 0,
(296, 34), 264559.1034656197), (66, 1, (115, 580), 264385.8983848628),
(68, 0, (120, 158), 263985.8983848628), (69, 0, (124, 482),
263585.8983848628), (70, 0, (167, 319), 263185.8983848628), (72, 1,
(278, 146), 263012.69330410595), (75, 1, (181, 29), 262839.4882233491),
(76, 0, (17, 495), 262439.4882233491), (81, 0, (91, 559),
262039.4882233491), (84, 0, (14, 579), 261639.4882233491), (92, 0,
(281, 472), 261239.4882233491), (96, 0, (196, 363), 260839.4882233491),
(99, 0, (59, 198), 260439.4882233491)]]
For given k = 10 best model's remaining area: 260439.4882233491
For Model number: 9
Best model so far:  7 th model
Beginning from 8 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)], [[[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294)], (5, 0,
(268, 437), 274530.3847577294), (7, 1, (75, 111), 274357.17967697256),
(9, 1, (245, 308), 274183.9745962157), (10, 1, (167, 563),
274010.76951545884), (13, 1, (342, 541), 273837.564434702), (14, 0,
(334, 84), 273437.564434702), (15, 0, (128, 363), 273037.564434702),
(16, 0, (154, 41), 272637.564434702), (18, 1, (246, 391),
272464.3593539451), (20, 0, (264, 485), 272064.3593539451), (21, 1,
(83, 218), 271891.15427318827), (22, 0, (138, 529),
271491.15427318827), (23, 1, (139, 186), 271317.9491924314), (24, 1,
(75, 262), 271144.74411167455), (25, 0, (19, 519), 270744.74411167455),
(28, 1, (317, 253), 270571.5390309177), (29, 0, (79, 30),
270171.5390309177), (30, 1, (10, 66), 269998.3339501608), (37, 0, (142,
315), 269598.3339501608), (41, 1, (15, 395), 269425.12886940397), (45,
1, (324, 30), 269251.9237886471), (49, 0, (195, 486),
268851.9237886471), (54, 0, (163, 502), 268451.9237886471), (56, 1,
(226, 73), 268278.71870789025), (58, 1, (244, 161), 268105.5136271334),
(65, 1, (131, 414), 267932.30854637653), (68, 0, (293, 512),
267532.30854637653), (69, 0, (313, 563), 267132.30854637653), (70, 0,
(215, 459), 266732.30854637653), (71, 0, (53, 214),
266332.30854637653), (74, 1, (134, 477), 266159.1034656197), (75, 0,
(329, 370), 265759.1034656197), (77, 1, (14, 257), 265585.8983848628),
(80, 1, (45, 484), 265412.69330410595), (82, 0, (101, 568),
265012.69330410595), (84, 1, (287, 466), 264839.4882233491), (86, 0,
(279, 194), 264439.4882233491), (90, 1, (18, 579), 264266.28314259223),
(93, 0, (193, 47), 263866.28314259223), (99, 0, (112, 592),
263466.28314259223)], [[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702), (12, 1,
(56, 457), 273264.3593539451), (15, 0, (235, 571), 272864.3593539451),
(17, 0, (107, 325), 272464.3593539451), (18, 0, (69, 503),
272064.3593539451), (23, 1, (177, 76), 271891.15427318827), (24, 1,
(24, 184), 271717.9491924314), (25, 0, (205, 33), 271317.9491924314),
(26, 0, (359, 559), 270917.9491924314), (28, 0, (92, 174),
270517.9491924314), (29, 0, (101, 56), 270117.9491924314), (31, 1, (9,
98), 269944.74411167455), (32, 0, (187, 506), 269544.74411167455), (34,
1, (285, 180), 269371.5390309177), (35, 1, (209, 354),
269198.3339501608), (37, 0, (145, 60), 268798.3339501608), (40, 0,
(318, 359), 268398.3339501608), (42, 0, (250, 423), 267998.3339501608),
(43, 1, (158, 284), 267825.12886940397), (44, 1, (311, 33),
267651.9237886471), (45, 1, (0, 240), 267478.71870789025), (46, 0, (99,
526), 267078.71870789025), (47, 0, (144, 253), 266678.71870789025),
(48, 0, (224, 378), 266278.71870789025), (51, 0, (325, 245),
265878.71870789025), (58, 0, (328, 225), 265478.71870789025), (62, 0,
(323, 437), 265078.71870789025), (64, 0, (33, 462),
264678.71870789025), (66, 0, (355, 90), 264278.71870789025), (69, 0,
(318, 539), 263878.71870789025), (81, 0, (309, 313),
263478.71870789025), (84, 1, (220, 142), 263305.5136271334), (86, 0,
(98, 547), 262905.5136271334), (92, 0, (306, 561), 262505.5136271334),
(93, 1, (358, 253), 262332.30854637653), (94, 1, (341, 376),
262159.1034656197), (95, 0, (354, 301), 261759.10346561967), (97, 0,
(30, 566), 261359.10346561967), (98, 1, (265, 514),
261185.89838486278)], [[[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702)], (7, 1,
(153, 473), 273264.3593539451), (11, 0, (235, 443), 272864.3593539451),
(12, 0, (38, 277), 272464.3593539451), (13, 1, (59, 345),
272291.15427318827), (14, 0, (324, 185), 271891.15427318827), (16, 1,
(50, 139), 271717.9491924314), (17, 1, (95, 35), 271544.74411167455),
(19, 0, (26, 473), 271144.74411167455), (21, 1, (62, 265),
270971.5390309177), (22, 0, (343, 27), 270571.5390309177), (23, 0,
(271, 223), 270171.5390309177), (25, 1, (121, 215), 269998.3339501608),
(27, 1, (260, 290), 269825.12886940397), (28, 1, (189, 73),
269651.9237886471), (30, 0, (230, 205), 269251.9237886471), (31, 0,
(39, 339), 268851.9237886471), (32, 0, (113, 429), 268451.9237886471),
(33, 1, (307, 345), 268278.71870789025), (39, 1, (231, 542),
268105.5136271334), (42, 0, (176, 252), 267705.5136271334), (44, 0,
(106, 377), 267305.5136271334), (46, 0, (59, 382), 266905.5136271334),
(50, 0, (272, 355), 266505.5136271334), (52, 0, (261, 465),
266105.5136271334), (54, 1, (240, 593), 265932.30854637653), (58, 0,
(355, 162), 265532.30854637653), (61, 0, (98, 408),
265132.30854637653), (62, 1, (350, 258), 264959.1034656197), (65, 0,
(296, 34), 264559.1034656197), (66, 1, (115, 580), 264385.8983848628),
(68, 0, (120, 158), 263985.8983848628), (69, 0, (124, 482),
263585.8983848628), (70, 0, (167, 319), 263185.8983848628), (72, 1,
(278, 146), 263012.69330410595), (75, 1, (181, 29), 262839.4882233491),
(76, 0, (17, 495), 262439.4882233491), (81, 0, (91, 559),
262039.4882233491), (84, 0, (14, 579), 261639.4882233491), (92, 0,
(281, 472), 261239.4882233491), (96, 0, (196, 363), 260839.4882233491),
(99, 0, (59, 198), 260439.4882233491)], [[[[[(1, 1, (96, 51),
276076.79491924314), (3, 0, (339, 495), 275676.79491924314)], (3, 1,
(95, 298), 275503.5898384863), (4, 0, (31, 178), 275103.5898384863),
(5, 1, (170, 560), 274930.3847577294), (6, 0, (353, 96),
274530.3847577294)], (6, 1, (243, 229), 274357.17967697256), (7, 1,
(122, 443), 274183.9745962157), (9, 0, (118, 360), 273783.9745962157),
(10, 1, (140, 163), 273610.76951545884), (11, 1, (295, 410),
273437.564434702)], (7, 1, (153, 473), 273264.3593539451), (11, 0,
(235, 443), 272864.3593539451), (12, 0, (38, 277), 272464.3593539451),
(13, 1, (59, 345), 272291.15427318827), (14, 0, (324, 185),
271891.15427318827), (16, 1, (50, 139), 271717.9491924314)], (9, 1,
(359, 499), 271544.74411167455), (10, 0, (299, 198),
271144.74411167455), (11, 0, (346, 281), 270744.74411167455), (12, 1,
(272, 45), 270571.5390309177), (15, 0, (73, 359), 270171.5390309177),
(16, 0, (93, 556), 269771.5390309177), (18, 0, (167, 306),
269371.5390309177), (20, 0, (186, 100), 268971.5390309177), (21, 1,
(60, 67), 268798.3339501608), (22, 1, (250, 144), 268625.12886940397),
(24, 0, (47, 358), 268225.12886940397), (25, 0, (250, 178),
267825.12886940397), (30, 0, (127, 108), 267425.12886940397), (31, 1,
(315, 588), 267251.9237886471), (34, 1, (20, 268), 267078.71870789025),
(38, 1, (62, 215), 266905.5136271334), (42, 1, (171, 536),
266732.30854637653), (49, 1, (105, 589), 266559.1034656197), (50, 0,
(23, 343), 266159.1034656197), (51, 0, (27, 158), 265759.1034656197),
(55, 0, (139, 500), 265359.1034656197), (60, 0, (82, 489),
264959.1034656197), (61, 0, (81, 585), 264559.1034656197), (63, 1,
(223, 225), 264385.8983848628), (66, 1, (185, 477),
264212.69330410595), (68, 0, (160, 55), 263812.69330410595), (80, 0,
(317, 247), 263412.69330410595), (81, 0, (238, 291),
263012.69330410595), (83, 0, (92, 107), 262612.69330410595), (85, 0,
(175, 572), 262212.69330410595), (90, 0, (64, 408),
261812.69330410595), (95, 0, (321, 469), 261412.69330410595), (97, 0,
(261, 453), 261012.69330410595), (98, 0, (288, 401),
260612.69330410595)]]
For given k = 10 best model's remaining area: 260612.69330410595
For Model number: 10
Best model so far:  7 th model
Beginning from 9 th iteration
Q-Matrix (state,shape choice,coordinate pair choice, remaining area)
 [[(1, 1, (332, 287), 276076.79491924314), (2, 0, (113, 144),
275676.79491924314), (4, 0, (351, 106), 275276.79491924314), (5, 1,
(22, 490), 275103.5898384863), (8, 1, (175, 56), 274930.3847577294),
(11, 0, (319, 330), 274530.3847577294), (13, 1, (32, 382),
274357.17967697256), (17, 0, (321, 523), 273957.17967697256), (18, 1,
(245, 166), 273783.9745962157), (20, 1, (268, 292),
273610.76951545884), (22, 1, (97, 571), 273437.564434702), (24, 0,
(326, 371), 273037.564434702), (25, 0, (101, 254), 272637.564434702),
(27, 0, (185, 394), 272237.564434702), (34, 1, (258, 111),
272064.3593539451), (36, 0, (210, 241), 271664.3593539451), (37, 0,
(228, 586), 271264.3593539451), (39, 0, (160, 595), 270864.3593539451),
(41, 0, (127, 44), 270464.3593539451), (42, 1, (248, 368),
270291.15427318827), (46, 0, (153, 44), 269891.15427318827), (48, 0,
(122, 66), 269491.15427318827), (51, 1, (10, 141), 269317.9491924314),
(52, 1, (92, 123), 269144.74411167455), (53, 1, (326, 35),
268971.5390309177), (54, 1, (76, 417), 268798.3339501608), (57, 1,
(165, 132), 268625.12886940397), (59, 0, (220, 86),
268225.12886940397), (65, 0, (262, 596), 267825.12886940397), (68, 0,
(149, 509), 267425.12886940397), (71, 0, (342, 328),
267025.12886940397), (79, 1, (203, 448), 266851.9237886471), (84, 0,
(99, 81), 266451.9237886471), (86, 0, (177, 508), 266051.9237886471),
(91, 1, (252, 410), 265878.71870789025), (93, 0, (236, 197),
265478.71870789025), (94, 0, (246, 56), 265078.71870789025)], [(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314),
(4, 0, (264, 132), 275276.79491924314), (5, 0, (338, 28),
274876.79491924314), (6, 0, (138, 133), 274476.79491924314), (8, 0,
(41, 501), 274076.79491924314), (10, 1, (308, 198), 273903.5898384863),
(11, 1, (159, 29), 273730.3847577294), (13, 1, (103, 566),
273557.17967697256), (14, 1, (333, 81), 273383.9745962157), (16, 0,
(208, 343), 272983.9745962157), (17, 0, (32, 108), 272583.9745962157),
(18, 0, (336, 578), 272183.9745962157), (19, 1, (57, 305),
272010.76951545884), (22, 0, (34, 230), 271610.76951545884), (23, 1,
(181, 114), 271437.564434702), (24, 0, (175, 508), 271037.564434702),
(26, 0, (23, 449), 270637.564434702), (28, 0, (166, 563),
270237.564434702), (29, 0, (112, 103), 269837.564434702), (30, 0, (339,
329), 269437.564434702), (31, 1, (89, 351), 269264.3593539451), (35, 0,
(351, 143), 268864.3593539451), (36, 1, (161, 263),
268691.15427318827), (38, 0, (185, 145), 268291.15427318827), (40, 0,
(24, 301), 267891.15427318827), (42, 1, (240, 44), 267717.9491924314),
(43, 0, (283, 539), 267317.9491924314), (49, 0, (168, 201),
266917.9491924314), (51, 0, (13, 43), 266517.9491924314), (55, 0, (78,
255), 266117.9491924314), (58, 0, (78, 411), 265717.9491924314), (60,
1, (16, 187), 265544.74411167455), (62, 0, (176, 368),
265144.74411167455), (63, 1, (232, 424), 264971.5390309177), (65, 1,
(28, 576), 264798.3339501608), (67, 0, (35, 262), 264398.3339501608),
(68, 1, (207, 309), 264225.12886940397), (72, 0, (1, 513),
263825.12886940397), (75, 1, (242, 210), 263651.9237886471), (77, 0,
(154, 323), 263251.9237886471), (79, 0, (109, 227), 262851.9237886471),
(89, 0, (281, 259), 262451.9237886471), (91, 1, (338, 422),
262278.71870789025), (93, 0, (207, 159), 261878.71870789025)], [[(1, 1,
(96, 51), 276076.79491924314)], (2, 0, (106, 451), 275676.79491924314),
(6, 1, (40, 258), 275503.5898384863), (7, 0, (154, 367),
275103.5898384863), (8, 0, (94, 27), 274703.5898384863), (9, 0, (16,
262), 274303.5898384863), (10, 1, (340, 488), 274130.3847577294), (11,
0, (26, 137), 273730.3847577294), (12, 1, (246, 528),
273557.17967697256), (13, 1, (329, 199), 273383.9745962157), (14, 1,
(158, 456), 273210.76951545884), (18, 1, (314, 151), 273037.564434702),
(19, 1, (321, 365), 272864.3593539451), (20, 1, (12, 67),
272691.15427318827), (22, 1, (198, 198), 272517.9491924314), (24, 0,
(198, 561), 272117.9491924314), (25, 1, (286, 89), 271944.74411167455),
(26, 1, (3, 344), 271771.5390309177), (29, 1, (164, 344),
271598.3339501608), (31, 0, (81, 351), 271198.3339501608), (32, 0,
(225, 360), 270798.3339501608), (33, 1, (269, 467),
270625.12886940397), (34, 1, (3, 492), 270451.9237886471), (37, 1,
(156, 109), 270278.71870789025), (41, 0, (40, 394),
269878.71870789025), (42, 0, (70, 543), 269478.71870789025), (44, 1,
(134, 192), 269305.5136271334), (46, 1, (318, 410),
269132.30854637653), (49, 0, (168, 25), 268732.30854637653), (53, 0,
(148, 277), 268332.30854637653), (54, 0, (204, 590),
267932.30854637653), (55, 1, (329, 278), 267759.1034656197), (56, 0,
(260, 47), 267359.1034656197), (57, 0, (8, 220), 266959.1034656197),
(59, 1, (226, 305), 266785.8983848628), (64, 0, (252, 238),
266385.8983848628), (65, 1, (79, 112), 266212.69330410595), (67, 0,
(280, 34), 265812.69330410595), (74, 0, (218, 271),
265412.69330410595), (75, 1, (167, 496), 265239.4882233491), (76, 1,
(62, 517), 265066.28314259223), (86, 1, (259, 202), 264893.0780618354),
(89, 0, (246, 378), 264493.0780618354), (90, 0, (129, 333),
264093.0780618354), (96, 0, (305, 273), 263693.0780618354)], [[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294), (6, 0, (353,
96), 274530.3847577294), (8, 0, (8, 371), 274130.3847577294), (10, 0,
(198, 181), 273730.3847577294), (11, 0, (354, 327), 273330.3847577294),
(12, 1, (88, 552), 273157.17967697256), (16, 0, (13, 265),
272757.17967697256), (17, 0, (247, 397), 272357.17967697256), (19, 1,
(355, 463), 272183.9745962157), (24, 1, (163, 248),
272010.76951545884), (27, 1, (349, 559), 271837.564434702), (28, 0,
(144, 155), 271437.564434702), (32, 0, (224, 47), 271037.564434702),
(34, 0, (151, 187), 270637.564434702), (35, 1, (283, 540),
270464.3593539451), (36, 0, (233, 308), 270064.3593539451), (38, 1,
(88, 222), 269891.15427318827), (42, 0, (124, 403),
269491.15427318827), (43, 0, (145, 484), 269091.15427318827), (47, 1,
(274, 67), 268917.9491924314), (50, 1, (287, 179), 268744.74411167455),
(51, 1, (298, 417), 268571.5390309177), (53, 0, (218, 476),
268171.5390309177), (55, 0, (52, 485), 267771.5390309177), (57, 0, (55,
355), 267371.5390309177), (58, 1, (70, 84), 267198.3339501608), (62, 1,
(72, 473), 267025.12886940397), (65, 0, (61, 559), 266625.12886940397),
(67, 0, (154, 418), 266225.12886940397), (68, 0, (30, 116),
265825.12886940397), (72, 0, (281, 472), 265425.12886940397), (73, 0,
(228, 218), 265025.12886940397), (74, 0, (26, 293),
264625.12886940397), (75, 1, (353, 504), 264451.9237886471), (77, 0,
(168, 129), 264051.9237886471), (80, 0, (1, 599), 263651.9237886471),
(83, 0, (60, 526), 263251.9237886471), (88, 1, (130, 49),
263078.71870789025), (89, 0, (235, 519), 262678.71870789025), (97, 0,
(178, 499), 262278.71870789025), (98, 0, (70, 107),
261878.71870789025), (99, 0, (104, 121), 261478.71870789025)], [[[(1,
1, (96, 51), 276076.79491924314), (3, 0, (339, 495),
275676.79491924314)], (3, 1, (95, 298), 275503.5898384863), (4, 0, (31,
178), 275103.5898384863)], (4, 0, (184, 292), 274703.5898384863), (5,
0, (35, 228), 274303.5898384863), (6, 1, (216, 26), 274130.3847577294),
(7, 0, (69, 536), 273730.3847577294), (9, 1, (245, 221),
273557.17967697256), (10, 1, (318, 149), 273383.9745962157), (11, 1,
(44, 411), 273210.76951545884), (12, 0, (2, 466), 272810.76951545884),
(13, 1, (58, 325), 272637.564434702), (15, 1, (179, 70),
272464.3593539451), (18, 1, (125, 243), 272291.15427318827), (19, 0,
(154, 403), 271891.15427318827), (20, 1, (333, 389),
271717.9491924314), (22, 1, (6, 572), 271544.74411167455), (23, 1,
(251, 126), 271371.5390309177), (27, 1, (315, 47), 271198.3339501608),
(28, 0, (335, 262), 270798.3339501608), (30, 1, (83, 37),
270625.12886940397), (34, 0, (233, 189), 270225.12886940397), (35, 0,
(181, 570), 269825.12886940397), (36, 0, (201, 496),
269425.12886940397), (37, 1, (162, 142), 269251.9237886471), (38, 1,
(243, 479), 269078.71870789025), (40, 0, (68, 192),
268678.71870789025), (41, 1, (295, 323), 268505.5136271334), (42, 1,
(344, 500), 268332.30854637653), (43, 0, (323, 467),
267932.30854637653), (44, 0, (31, 196), 267532.30854637653), (45, 1,
(355, 309), 267359.1034656197), (49, 0, (110, 196), 266959.1034656197),
(51, 0, (37, 490), 266559.1034656197), (54, 1, (3, 80),
266385.8983848628), (61, 0, (73, 588), 265985.8983848628), (62, 0,
(170, 183), 265585.8983848628), (64, 0, (3, 440), 265185.8983848628),
(71, 0, (239, 409), 264785.8983848628), (76, 0, (278, 557),
264385.8983848628), (78, 1, (70, 277), 264212.69330410595), (82, 0,
(49, 167), 263812.69330410595), (91, 0, (103, 88), 263412.69330410595),
(92, 0, (152, 40), 263012.69330410595), (94, 0, (151, 297),
262612.69330410595), (98, 1, (186, 351), 262439.4882233491)], [[[(1, 1,
(96, 51), 276076.79491924314), (3, 0, (339, 495), 275676.79491924314)],
(3, 1, (95, 298), 275503.5898384863), (4, 0, (31, 178),
275103.5898384863), (5, 1, (170, 560), 274930.3847577294)], (5, 0,
(268, 437), 274530.3847577294), (7, 1, (75, 111), 274357.17967697256),
(9, 1, (245, 308), 274183.9745962157), (10, 1, (167, 563),
274010.76951545884), (13, 1, (342, 541), 273837.564434702), (14, 0,
(334, 84), 273437.564434702), (15, 0, (128, 363), 273037.564434702),
(16, 0, (154, 41), 272637.564434702), (18, 1, (246, 391),
272464.3593539451), (20, 0, (264, 485), 272064.3593539451), (21, 1,
(83, 218), 271891.15427318827), (22, 0, (138, 529),
271491.15427318827), (23, 1, (139, 186), 271317.9491924314), (24, 1,
(75, 262), 271144.74411167455), (25, 0, (19, 519), 270744.74411167455),
(28, 1, (317, 253), 270571.5390309177), (29, 0, (79, 30),
270171.5390309177), (30, 1, (10, 66), 269998.3339501608), (37, 0, (142,
315), 269598.3339501608), (41, 1, (15, 395), 269425.12886940397), (45,
1, (324, 30), 269251.9237886471), (49, 0, (195, 486),
268851.9237886471), (54, 0, (163, 502), 268451.9237886471), (56, 1,
(226, 73), 268278.71870789025), (58, 1, (244, 161), 268105.5136271334),
(65, 1, (131, 414), 267932.30854637653), (68, 0, (293, 512),
267532.30854637653), (69, 0, (313, 563), 267132.30854637653), (70, 0,
(215, 459), 266732.30854637653), (71, 0, (53, 214),
266332.30854637653), (74, 1, (134, 477), 266159.1034656197), (75, 0,
(329, 370), 265759.1034656197), (77, 1, (14, 257), 265585.8983848628),
(80, 1, (45, 484), 265412.69330410595), (82, 0, (101, 568),
265012.69330410595), (84, 1, (287, 466), 264839.4882233491), (86, 0,
(279, 194), 264439.4882233491), (90, 1, (18, 579), 264266.28314259223),
(93, 0, (193, 47), 263866.28314259223), (99, 0, (112, 592),
263466.28314259223)], [[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702), (12, 1,
(56, 457), 273264.3593539451), (15, 0, (235, 571), 272864.3593539451),
(17, 0, (107, 325), 272464.3593539451), (18, 0, (69, 503),
272064.3593539451), (23, 1, (177, 76), 271891.15427318827), (24, 1,
(24, 184), 271717.9491924314), (25, 0, (205, 33), 271317.9491924314),
(26, 0, (359, 559), 270917.9491924314), (28, 0, (92, 174),
270517.9491924314), (29, 0, (101, 56), 270117.9491924314), (31, 1, (9,
98), 269944.74411167455), (32, 0, (187, 506), 269544.74411167455), (34,
1, (285, 180), 269371.5390309177), (35, 1, (209, 354),
269198.3339501608), (37, 0, (145, 60), 268798.3339501608), (40, 0,
(318, 359), 268398.3339501608), (42, 0, (250, 423), 267998.3339501608),
(43, 1, (158, 284), 267825.12886940397), (44, 1, (311, 33),
267651.9237886471), (45, 1, (0, 240), 267478.71870789025), (46, 0, (99,
526), 267078.71870789025), (47, 0, (144, 253), 266678.71870789025),
(48, 0, (224, 378), 266278.71870789025), (51, 0, (325, 245),
265878.71870789025), (58, 0, (328, 225), 265478.71870789025), (62, 0,
(323, 437), 265078.71870789025), (64, 0, (33, 462),
264678.71870789025), (66, 0, (355, 90), 264278.71870789025), (69, 0,
(318, 539), 263878.71870789025), (81, 0, (309, 313),
263478.71870789025), (84, 1, (220, 142), 263305.5136271334), (86, 0,
(98, 547), 262905.5136271334), (92, 0, (306, 561), 262505.5136271334),
(93, 1, (358, 253), 262332.30854637653), (94, 1, (341, 376),
262159.1034656197), (95, 0, (354, 301), 261759.10346561967), (97, 0,
(30, 566), 261359.10346561967), (98, 1, (265, 514),
261185.89838486278)], [[[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702)], (7, 1,
(153, 473), 273264.3593539451), (11, 0, (235, 443), 272864.3593539451),
(12, 0, (38, 277), 272464.3593539451), (13, 1, (59, 345),
272291.15427318827), (14, 0, (324, 185), 271891.15427318827), (16, 1,
(50, 139), 271717.9491924314), (17, 1, (95, 35), 271544.74411167455),
(19, 0, (26, 473), 271144.74411167455), (21, 1, (62, 265),
270971.5390309177), (22, 0, (343, 27), 270571.5390309177), (23, 0,
(271, 223), 270171.5390309177), (25, 1, (121, 215), 269998.3339501608),
(27, 1, (260, 290), 269825.12886940397), (28, 1, (189, 73),
269651.9237886471), (30, 0, (230, 205), 269251.9237886471), (31, 0,
(39, 339), 268851.9237886471), (32, 0, (113, 429), 268451.9237886471),
(33, 1, (307, 345), 268278.71870789025), (39, 1, (231, 542),
268105.5136271334), (42, 0, (176, 252), 267705.5136271334), (44, 0,
(106, 377), 267305.5136271334), (46, 0, (59, 382), 266905.5136271334),
(50, 0, (272, 355), 266505.5136271334), (52, 0, (261, 465),
266105.5136271334), (54, 1, (240, 593), 265932.30854637653), (58, 0,
(355, 162), 265532.30854637653), (61, 0, (98, 408),
265132.30854637653), (62, 1, (350, 258), 264959.1034656197), (65, 0,
(296, 34), 264559.1034656197), (66, 1, (115, 580), 264385.8983848628),
(68, 0, (120, 158), 263985.8983848628), (69, 0, (124, 482),
263585.8983848628), (70, 0, (167, 319), 263185.8983848628), (72, 1,
(278, 146), 263012.69330410595), (75, 1, (181, 29), 262839.4882233491),
(76, 0, (17, 495), 262439.4882233491), (81, 0, (91, 559),
262039.4882233491), (84, 0, (14, 579), 261639.4882233491), (92, 0,
(281, 472), 261239.4882233491), (96, 0, (196, 363), 260839.4882233491),
(99, 0, (59, 198), 260439.4882233491)], [[[[[(1, 1, (96, 51),
276076.79491924314), (3, 0, (339, 495), 275676.79491924314)], (3, 1,
(95, 298), 275503.5898384863), (4, 0, (31, 178), 275103.5898384863),
(5, 1, (170, 560), 274930.3847577294), (6, 0, (353, 96),
274530.3847577294)], (6, 1, (243, 229), 274357.17967697256), (7, 1,
(122, 443), 274183.9745962157), (9, 0, (118, 360), 273783.9745962157),
(10, 1, (140, 163), 273610.76951545884), (11, 1, (295, 410),
273437.564434702)], (7, 1, (153, 473), 273264.3593539451), (11, 0,
(235, 443), 272864.3593539451), (12, 0, (38, 277), 272464.3593539451),
(13, 1, (59, 345), 272291.15427318827), (14, 0, (324, 185),
271891.15427318827), (16, 1, (50, 139), 271717.9491924314)], (9, 1,
(359, 499), 271544.74411167455), (10, 0, (299, 198),
271144.74411167455), (11, 0, (346, 281), 270744.74411167455), (12, 1,
(272, 45), 270571.5390309177), (15, 0, (73, 359), 270171.5390309177),
(16, 0, (93, 556), 269771.5390309177), (18, 0, (167, 306),
269371.5390309177), (20, 0, (186, 100), 268971.5390309177), (21, 1,
(60, 67), 268798.3339501608), (22, 1, (250, 144), 268625.12886940397),
(24, 0, (47, 358), 268225.12886940397), (25, 0, (250, 178),
267825.12886940397), (30, 0, (127, 108), 267425.12886940397), (31, 1,
(315, 588), 267251.9237886471), (34, 1, (20, 268), 267078.71870789025),
(38, 1, (62, 215), 266905.5136271334), (42, 1, (171, 536),
266732.30854637653), (49, 1, (105, 589), 266559.1034656197), (50, 0,
(23, 343), 266159.1034656197), (51, 0, (27, 158), 265759.1034656197),
(55, 0, (139, 500), 265359.1034656197), (60, 0, (82, 489),
264959.1034656197), (61, 0, (81, 585), 264559.1034656197), (63, 1,
(223, 225), 264385.8983848628), (66, 1, (185, 477),
264212.69330410595), (68, 0, (160, 55), 263812.69330410595), (80, 0,
(317, 247), 263412.69330410595), (81, 0, (238, 291),
263012.69330410595), (83, 0, (92, 107), 262612.69330410595), (85, 0,
(175, 572), 262212.69330410595), (90, 0, (64, 408),
261812.69330410595), (95, 0, (321, 469), 261412.69330410595), (97, 0,
(261, 453), 261012.69330410595), (98, 0, (288, 401),
260612.69330410595)], [[[[[(1, 1, (96, 51), 276076.79491924314), (3, 0,
(339, 495), 275676.79491924314)], (3, 1, (95, 298), 275503.5898384863),
(4, 0, (31, 178), 275103.5898384863), (5, 1, (170, 560),
274930.3847577294), (6, 0, (353, 96), 274530.3847577294)], (6, 1, (243,
229), 274357.17967697256), (7, 1, (122, 443), 274183.9745962157), (9,
0, (118, 360), 273783.9745962157), (10, 1, (140, 163),
273610.76951545884), (11, 1, (295, 410), 273437.564434702)], (7, 1,
(153, 473), 273264.3593539451), (11, 0, (235, 443), 272864.3593539451),
(12, 0, (38, 277), 272464.3593539451), (13, 1, (59, 345),
272291.15427318827), (14, 0, (324, 185), 271891.15427318827), (16, 1,
(50, 139), 271717.9491924314), (17, 1, (95, 35), 271544.74411167455)],
(11, 0, (263, 134), 271144.74411167455), (13, 1, (70, 374),
270971.5390309177), (14, 1, (253, 561), 270798.3339501608), (16, 0,
(52, 235), 270398.3339501608), (17, 0, (246, 316), 269998.3339501608),
(18, 1, (336, 311), 269825.12886940397), (23, 0, (131, 482),
269425.12886940397), (24, 0, (153, 396), 269025.12886940397), (26, 0,
(61, 421), 268625.12886940397), (27, 0, (270, 250),
268225.12886940397), (28, 1, (345, 100), 268051.9237886471), (29, 1,
(169, 208), 267878.71870789025), (32, 1, (244, 396),
267705.5136271334), (34, 0, (11, 400), 267305.5136271334), (35, 0,
(324, 202), 266905.5136271334), (37, 1, (231, 447),
266732.30854637653), (40, 1, (125, 564), 266559.1034656197), (42, 0,
(2, 249), 266159.1034656197), (43, 0, (23, 188), 265759.1034656197),
(47, 0, (332, 390), 265359.1034656197), (48, 1, (149, 145),
265185.8983848628), (55, 0, (167, 25), 264785.8983848628), (57, 0,
(104, 272), 264385.8983848628), (58, 1, (358, 439),
264212.69330410595), (60, 0, (25, 120), 263812.69330410595), (65, 1,
(187, 42), 263639.4882233491), (66, 1, (271, 69), 263466.28314259223),
(68, 0, (323, 144), 263066.28314259223), (74, 0, (1, 91),
262666.28314259223), (76, 0, (288, 165), 262266.28314259223), (77, 0,
(136, 64), 261866.28314259223), (81, 1, (3, 534), 261693.07806183535),
(86, 0, (339, 338), 261293.07806183535), (90, 0, (130, 22),
260893.07806183535), (92, 1, (297, 270), 260719.87298107846), (99, 0,
(139, 155), 260319.87298107846)]]
For given k = 10 best model's remaining area: 260319.87298107846
-----------------------------------------------------------------------
--------------------------
k =  20
For given k = 20 remaining areas: [265466.28314259223,
264105.5136271334, 266278.71870789025, 262451.9237886471,
264505.5136271334, 263705.5136271334, 262051.9237886471,
258612.6933041059, 258266.28314259215, 256653.84757729404,
254480.64249653716, 252068.206931239, 249895.0018504821,
247936.1561236701, 239828.97644669766, 235416.54088139944,
235977.3103968581, 233004.10531610122, 230538.07991231678,
224913.20878172034]
For given k = 20 best model's remaining area: 224913.20878172034
```

```
------------------------------------------------------------------------
--------------------------
k =  30
For given k = 30 remaining areas: [263998.3339501608,
261025.1288694039, 261359.10346561967, 261759.10346561967,
263585.8983848628, 264039.4882233491, 262493.0780618354,
264959.1034656197, 262159.1034656197, 259025.1288694039,
259800.2577388077, 257693.07806183517, 256773.4628195646,
253334.23233502338, 251775.38660821138, 246736.15612367005,
240043.3358006425, 239750.51547761494, 235725.6443470185,
228340.0037009634, 223115.13257036696, 218488.33765112385,
214009.87668204118, 204904.6207937154, 197118.9801476603,
195613.72425933453, 184854.87853252253, 180656.8023211692,
169217.57183662787, 163312.3159483021]
For given k = 30 best model's remaining area: 163312.3159483021
------------------------------------------------------------------------
--------------------------
k =  40
For given k = 40 remaining areas: [261078.71870789022,
264105.5136271334, 264959.1034656197, 262732.30854637653,
262851.9237886471, 265132.30854637653, 263466.28314259223,
262212.69330410595, 262278.71870789025, 261239.48822334906,
261585.8983848628, 258666.28314259212, 258666.28314259215,
258546.66790032163, 255961.02725426655, 251841.41201199588,
246802.18152745455, 240670.13071988567, 240096.92563912878,
234018.46467004612, 231420.3884586928, 222434.74781263768,
218875.90208582568, 213424.23603598613, 207226.1598246328,
202401.28869403637, 196830.00740192615, 187271.16167511416,
183073.08546376083, 174021.41941392128, 164408.98384862306,
156730.5228795404, 147798.4720719715, 139320.01110288885,
130948.72981077863, 121377.44851866842, 113245.39771109953,
102940.14182277376, 94354.50117671865, 86036.80972309466]
For given k = 40 best model's remaining area: 86036.80972309466
------------------------------------------------------------------------
--------------------------
k =  50
For given k = 50 remaining areas: [261812.69330410592,
264398.3339501608, 263305.5136271334, 265532.30854637653,
260505.51362713333, 263359.1034656197, 261919.87298107852,
261932.3085463765, 263878.71870789025, 262039.48822334906,
260159.10346561953, 256093.07806183514, 254893.07806183514,
252307.43741578003, 247387.82217350937, 245495.0018504818,
239309.3612044267, 235123.7205583716, 232030.90023534404,
226925.64434701827, 220686.41386247694, 215927.56813566494,
212075.9020858254, 205304.62079371518, 200199.3649053894,
193720.90393630674, 185015.64804798097, 177337.1870788983,
172231.93119057253, 162846.29054451743, 156648.2143331641,
147876.93304105388, 138705.65174894367, 130734.37045683345,
122589.88408396635, 114911.42311488368, 107513.34690353036,
96861.6808536908, 88329.63004612192, 80704.75891552548,
71492.32335022726, 62321.04205811705, 53388.99125054816,
44790.915039194835, 36832.06931238284, 28700.018504813954,
18967.96769724507, 11862.711808919295, 4757.455920593527,
921.0050484891249]

For given k = 50 best model’s remaining area: 921.0050484891249
```

3.3 Observations

- Increasing k number:

k number in our implementation determines the number of models created. More models created, our remaining area decreased. Hence our reward in last step increases. To exemplify,

```
For given k = 10 best model's remaining area: 260319.87298107846
For given k = 30 best model's remaining area: 180656.8023211692
```

Even k(number of models) = 30 gave approximately half of the result of k(number of models) = 10

- Number of loops:

Worst case running time of our implementation is
= (# of models)*(# of states)*(# of coordinate pairs for randomly chosen shape)
Hence, for increasing number of models, or/and number of the states, or/and number of the coordinate pairs, our execution time increasing. However, for building our dataset, training our algorithm(agent) is a fundamental need for us to reach the best(optimal) solution at the end. We tried to change these affecting factors(for #of models and # of states) and took results for different combinations. Keeping # of states in an average value such as 100, yet increasing number of models and in that case, increasing Q-matrix gave us more educated results.

- Brute force approach vs using library:

For similar implementations in the future, we would definitely start with the library implementation for reinforcement learning. We spent more than 5 hours to execution of our code for even a single model, and took the outputs

- Sample Instances:

We might have given higher dimensions for shapes to get output fast.

4 Discussion

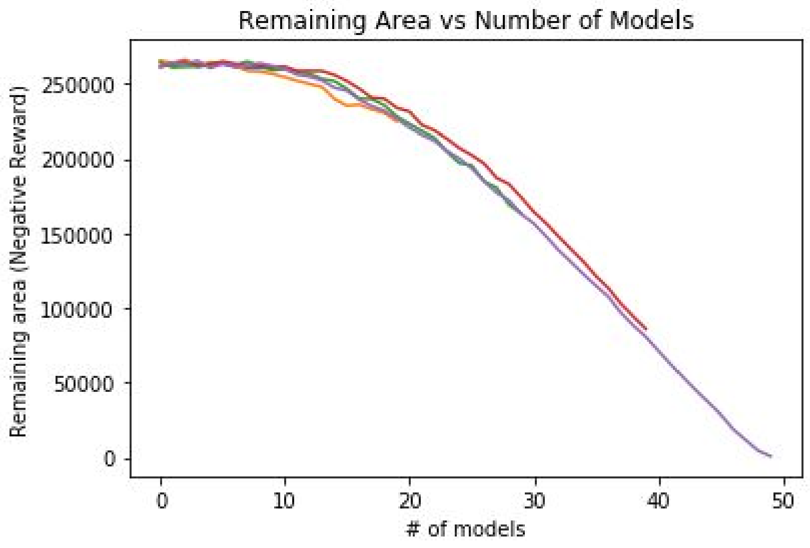

Figure 4: Remaining Area vs #of models

According to our output results, by increasing metrics such as model count and state count, we are getting closer to the optimal solution. By the comparison of the best results in output. The difference between training with data set size of 10 and training with the data set size of 100 is large. Hence, If we increase the number of models, we increase the size of our dataset. In that way we can reach the optimal solution.

In part 3.2 we also discussed our observations, results and interpretation of our results more clearly.

5 Conclusion

- How to reach the most optimized way to cover the canvas ?

By running our model, we found that in reinforcement learning best pattern with smallest remaining area can be reached by increasing the number of features that is, training the model with more data results better solutions.

In this project we learn the fundamental workings of reinforcement learning and Q learning rule by implementing a brute force algorithm and improve our RL model by creating a detailed pygame environment and run the OpenAI Gym agent.

- Which arrangement of tiles yields the smallest empty area ?
- How can we reach a better solution?

Since our design is dynamic, which is randomly coordinates differs in every model, there is no certain arrangement for smallest empty area yet, in the light of information which is mentioned in previous parts, we can reach optimal solution for this problem with increasing #of models or state size.

- Which approach would be better to implement our environment?

By analyzing our run time, implementing brute force part of this algorithm is valuable for understanding design and whole reinforcement learning concept. However, it took too much time to execute for various model counts. As a known fact, we should have created our own dataset and we created our own dataset by creating many models and finding best arrangement of tiles in them. Hence, we should # of models variable larger. In that way our run time increased linearly.

As a conclusion, if we have started with OpenAI gym library implementation, we can reach optimal solution faster. Hence, OpenAI gym library implementation would be better to implement our environment.

At the current stage of the project we find the pattern with minimum empty area, for future studies this remaining area can be filled with hexagons that are cut from different angles. Moreover, a user interface can be designed where user can select a different type of geometries for the canvas and change dimensions.

6 Appendix

6.1 Contribution of Team Members

Ege Tınaz	Project idea, problem definition, planning of the project, help coding (brute force and custom environment), report writing; introduction, methods, conclusion
Ayca Begum Tascioglu	Design of the project, coding and implementation of the idea, brute force algorithm, custom environment, results, discussion, conclusion, presentation
Emre Dikenelli	Nothing
Emre Selver	Left the group after demo. Nothing
Can Savci	Left the group after demo. Nothing

6.2 Source Files

For look further to the files please visit https://github.com/aeyc/CeramicTilesForHybridArmor

6.2.1 First_Version.py

```
from tkinter import*
import random
import math
import timeit
EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425*650
window = Tk()
window.title("Work Place")
canvas = Canvas(window, width=X_LIMIT, height=Y_LIMIT)
# frame = Frame(width=500, height=200, bg='blue')
# canvas = Canvas(frame, bg='white')
```

```
canvas.pack(fill = BOTH, expand = YES)
# frame.pack(fill = BOTH, expand = YES)


def hex_points(x, y):
   points = (x+EDGE, y-EDGE, x,y, x+EDGE, y+EDGE, x+2*EDGE,y+EDGE,x+3*EDGE,y,x+2*EDGE,y-EDGE)
   return points


def hex_area(x):
   area = 0.0
   area = math.sqrt(3)*x*x/4
   return area


def square_points(x,y):
   points = (x, y, x, y+EDGE,x+EDGE,y+EDGE,x+EDGE,y)
   return points

def square_area(x):
   area = 0.0
   return x*x


count = 0  # number of iterations of the loop for a single model
appeared = False  # a controller flag to find whether random coordinate is already used or not
remaining_area = TOTAL_AREA  # the area which will represent the score variable for reinforcement learning
Q = []  # later process, Q matrix of the whole models for comparing actions and scores
moves = []  # keeps the data of the iteration count, shape selection and the coordinate selection
used_points = []  # keeps the data of used points(coordinates), to avoid overlapping
start_time = timeit.default_timer()  # timer initialization
while remaining_area > 100000:
   choice = random.randint(0, 2)
   appeared = False
   tmp_points = []  # tmp points stores the randomly selected points and based on the shape, other coordinates
that will added to this list
   if choice == 1:
       print("choice 1")
       x = random.randint(0, X_LIMIT)
       y = random.randint(EDGE, Y_LIMIT)

       #rectangular area
       for i in range(x+EDGE, x+2*EDGE):
           for j in range(y-EDGE, y+EDGE):
               tmp_points.append((i,j))

       #triangular area-up left
       tmp_k = x
       for j in range(y,y+EDGE):
           for i in range(tmp_k, x+EDGE):
               tmp_points.append((i, j))
           tmp_k+=1

       # triangular area-down left
```

```python
        tmp_k = x
        for j in range(y,y-EDGE,-1):
            for i in range(tmp_k, x+EDGE):
                tmp_points.append((i, j))
            tmp_k+=1

        # triangular area-up right
        tmp_k = x+2*EDGE
        for j in range(y, y + EDGE):
            for i in range(tmp_k, x + 3*EDGE):
                tmp_points.append((i, j))
            tmp_k += 1

        # triangular area-down right
        tmp_k = x + 2 * EDGE
        for j in range(y, y -EDGE,-1):
            for i in range(tmp_k, x + 3 * EDGE):
                tmp_points.append((i, j))
            tmp_k += 1

        for i in used_points:
            for j in tmp_points:
                if j == i:
                    appeared = True
        if not appeared:
            hexagon_points = hex_points(x, y)  # range of hexagon (x>=0, y>=20)
            a = canvas.create_polygon(hexagon_points, outline='green', fill='yellow', width=1)
            print("hex created")
            for i in tmp_points:
                used_points.append(i)
            remaining_area -= hex_area(EDGE)
            moves.append((count, choice, (x, y)))
            print("remaining area", remaining_area)
    elif choice == 0:
        print("choice 0")
        x = random.randint(0, X_LIMIT)
        y = random.randint(EDGE, Y_LIMIT)
        for i in range(x, x + EDGE):
            for j in range(y, y + EDGE):
                tmp_points.append((i, j))
        for i in used_points:
            for j in tmp_points:
                if j == i:
                    appeared = True
        if not appeared:
            s_points = square_points(x,y)
            b = canvas.create_polygon(s_points, outline="blue", fill="#fb0")
            print("square created")
            for i in range(x, x + EDGE):
                for j in range(y, y + EDGE):
                    used_points.append((i, j))
            remaining_area -= square_area(EDGE)
            moves.append((count,choice,(x,y)))
            print("remaining area", remaining_area)
    count+=1
    print("count",count)
    #canvas.delete("all")

# count = 1
# for i in model_analytic:
#    print(count, "th model:")
#    print("Remaining Area:",i[0])
#    count+=1


print(moves)
elapsed = timeit.default_timer() - start_time
print(elapsed)
window.mainloop()
```

6.2.2 Last_Version.py

```
import random
import math
import timeit

EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425 * 650


# frame = Frame(width=500, height=200, bg='blue')
# canvas = Canvas(frame, bg='white')
# frame.pack(fill = BOTH, expand = YES)


def hex_points(x, y):
   points = (
   x + EDGE, y - EDGE, x, y, x + EDGE, y + EDGE, x + 2 * EDGE, y + EDGE, x + 3 *
EDGE, y, x + 2 * EDGE, y - EDGE)
   return points


def hex_area(x):
   area = 0.0
   area = math.sqrt(3) * x * x / 4
   return area
```

```
def square_points(x, y):
    points = (x, y, x, y + EDGE, x + EDGE, y + EDGE, x + EDGE, y)
    return points


def square_area(x):
    area = 0.0
    return x * x


start_time = timeit.default_timer()  # timer initialization
# Q = []  # later process, Q matrix of the whole models for comparing actions and
scores
# model_count = 0
# best_actions_states = []
# rem_areas = []  #dummy list just keptfor time efficiency
k = [2, 5, 10, 50, 100]
for j in range(10, 100, 10):
    Q = []  # later process, Q matrix of the whole models for comparing actions and
scores
    model_count = 0
    best_actions_states = []
    rem_areas = []  # dummy list just keptfor time efficiency

print("-----------------------------------------------------------------------------
---------------------")
    print("k = ", j)
    for i in range(0, j):
        moves = []  # keeps the data of the iteration count, shape selection and the
coordinate selection
        used_points = []  # keeps the data of used points(coordinates), to avoid
overlapping
        count = 0  # number of iterations of the loop for a single model
        appeared = False  # a controller flag to find whether random coordinate is
already used or not
        remaining_area = TOTAL_AREA  # the area which will represent the score
variable for reinforcement learning

        print("For Model number:", model_count + 1)
        if model_count > 1:
            index = rem_areas.index(min(rem_areas))  # find the position of
bestmodel
            # print("index",index)
            # print("q-len",len(Q))
            # print("Q[index][0]",(Q[index])[0])
            moves.append((Q[index])[0:model_count - 1])
            print("Best model so far: ", index, "th model")

            # print("moves",moves)
            for i in range(0, model_count - 1):
                remaining_area = (moves[0])[-1][3]
            count += model_count
            print("Beginning from", count, "th iteration")
        while count < 100:
            choice = random.randint(0, 2)
            appeared = False
            tmp_points = []  # tmp_points stores the randomly selected points and
based on the shape, other coordinates that will added to this list
            if choice == 1:
                # print("choice 1")
                x = random.randint(0, X_LIMIT)
                y = random.randint(EDGE, Y_LIMIT)

                # rectangular area
                for i in range(x + EDGE, x + 2 * EDGE):
                    for j in range(y - EDGE, y + EDGE):
                        tmp_points.append((i, j))

                # triangular area-up left
                tmp_k = x
                for j in range(y, y + EDGE):
                    for i in range(tmp_k, x + EDGE):
                        tmp_points.append((i, j))
                    tmp_k += 1

                # triangular area-down left
                tmp_k = x
                for j in range(y, y - EDGE, -1):
                    for i in range(tmp_k, x + EDGE):
                        tmp_points.append((i, j))
                    tmp_k += 1

                # triangular area-up right
                tmp_k = x + 2 * EDGE
                for j in range(y, y + EDGE):
                    for i in range(tmp_k, x + 3 * EDGE):
                        tmp_points.append((i, j))
                    tmp_k += 1

                # triangular area-down right
                tmp_k = x + 2 * EDGE
                for j in range(y, y - EDGE, -1):
```

```
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            for i in used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                hexagon_points = hex_points(x, y)  # range of hexagon (x>=0,
y>=20)
                # print("hex created")
                for i in tmp_points:
                    used_points.append(i)
                remaining_area -= hex_area(EDGE)
                moves.append((count, choice, (x, y), remaining_area))

                # print("remaining area", remaining_area)
        elif choice == 0:
            # print("choice 0")
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)
            for i in range(x, x + EDGE):
                for j in range(y, y + EDGE):
                    tmp_points.append((i, j))
            for i in used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                s_points = square_points(x, y)
                # print("square created")
                for i in range(x, x + EDGE):
                    for j in range(y, y + EDGE):
                        used_points.append((i, j))
                remaining_area -= square_area(EDGE)
                moves.append((count, choice, (x, y), remaining_area))

                # print("remaining area", remaining_area)
        count += 1
        # print("count",count)
    rem_areas.append(remaining_area)

    Q.append(moves)
    print("Q-Matrix (state,shape choice,coordinate pair choice, remaining
area)\n", Q)
```

```
        # print("moves",moves)
        # print("Q",Q)
        # print("Q[0]",Q[0])
        # print("Q[0][1]",(Q[0])[1])
        # print("Q[0][2]",Q[0][2])
        # print("Q[0][3]",Q[0][3])
        model_count += 1
        print(("For given k = {} best model's remaining area: {}").format(k,
rem_areas))
        # canvas.delete("all")

    # count = 1
    # for i in model_analytic:
    #    print(count, "th model:")
    #    print("Remaining Area:",i[0])
    #    count+=1
print("\n--------------------------------------------------END--------------------
-----------------------------")
print("Best model remaining area:", rem_areas)
elapsed = timeit.default_timer() - start_time
print(elapsed)
```

6.2.3 Engine.py (trial version without gym-tetris library)

```
import pygame,random,time,sys,math
from pygame.locals import *

EDGE = 20
X_LIMIT = 360
Y_LIMIT = 600
TOTAL_AREA = 425*650

def hex_points(x, y):
  points = [(x+EDGE, y-EDGE), (x,y), (x+EDGE, y+EDGE),
(x+2*EDGE,y+EDGE),(x+3*EDGE,y),(x+2*EDGE,y-EDGE)]
  return points


def hex_area(x):
  area = 0.0
  area = math.sqrt(3)*x*x/4
  return area


def square_points(x,y):
  points = [(x, y),( x, y+EDGE),(x+EDGE,y+EDGE),(x+EDGE,y)]
  return points
```

```
def square_area(x):
    return x*x


class GameState:
    def __init__(self):
        global FPSCLOCK,DISPLAYSURF, BASICFONT,BIGFONT
        pygame.init()
        FPSCLOCK = pygame.time.Clock()
        DISPLAYSURF = pygame.display.set_mode((425, 650))
        BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
        BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
        pygame.display.iconify()
        pygame.display.set_caption('Ceramic Tiles for Hybrid Armor')

        self.score = 0
        self.remaining_area = 0
        self.used_points = []
        self.moves = []
        self.coords = self.shape_choice()

        pygame.display.update()

    def reinit(self):
        self.score = 0
        self.remaining_area = 0
        self.used_points = []
        self.moves = []
        self.shape = self.shape_choice()

        pygame.display.update()

    def shape_choice(self):
        choice = random.randint(0, 2)
        tmp_points = [] #coordinate for the polygon
        appeared = False
        coords = []
        if choice == 0:
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)
            for i in range(x, x + EDGE):
                for j in range(y, y + EDGE):
                    tmp_points.append((i, j))
            for i in self.used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                s_points = square_points(x, y)
                coords = s_points
                #create polygon here?
                for i in range(x, x + EDGE):
                    for j in range(y, y + EDGE):
                        self.used_points.append((i, j))
```

```python
                self.remaining_area -= square_area(EDGE)
                self.moves.append(( choice, (x, y)))
        elif choice == 1:
            print("choice 1")
            x = random.randint(0, X_LIMIT)
            y = random.randint(EDGE, Y_LIMIT)

            # rectangular area
            for i in range(x + EDGE, x + 2 * EDGE):
                for j in range(y - EDGE, y + EDGE):
                    tmp_points.append((i, j))

            # triangular area-up left
            tmp_k = x
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down left
            tmp_k = x
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-up right
            tmp_k = x + 2 * EDGE
            for j in range(y, y + EDGE):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            # triangular area-down right
            tmp_k = x + 2 * EDGE
            for j in range(y, y - EDGE, -1):
                for i in range(tmp_k, x + 3 * EDGE):
                    tmp_points.append((i, j))
                tmp_k += 1

            for i in self.used_points:
                for j in tmp_points:
                    if j == i:
                        appeared = True
            if not appeared:
                hexagon_points = hex_points(x, y)  # range of hexagon (x>=0, y>=20)
                coords = hexagon_points
                #a = canvas.create_polygon(hexagon_points, outline='green', fill='yellow', width=1)
                for i in tmp_points:
                    self.used_points.append(i)
                self.remaining_area -= hex_area(EDGE)
                self.moves.append((choice, (x, y)))
        return coords
```

```
  def getImage(self):
      image_data = pygame.surfarray.array3d(pygame.transform.rotate(pygame.display.get_surface(), 90))
      return image_data
# Surface = pygame.display.set_mode((800,600))
# BLUE = (80,208,255)
# PINK = (255,208,160)
# g = GameState()
# print(g.shape_choice())
# # pygame.draw.polygon(Surface, (BLUE), g.shape_choice())
# # run = True
# # while run:
# #     for event in pygame.event.get():
# #         if event.type == pygame.QUIT:
# #             run = False
# #     pygame.display.update()
# #     break
```

6.2.4 Changes in gym-tetris

6.2.4.1 gym-tetris/Engine.py

```
SHAPES = {'S': SQUARE,
          'H': HEXAGON}
```

6.2.4.1 gym-tetris/__init__.py

```
import gym
#import pdb; pdb.set_trace()

for env in gym.envs.registration.EnvRegistry.env_specs:
    if 'Tetris' in env:
        print('Remove {} from registry'.format(env))
        del gym.envs.registration.EnvRegistry.env_specs[env]
```

7 Reference

[1] B. Wang, “Aluminum encapsulated ceramic tiles for hybrid armor,” *1761eac3b1967fa6e4cafb6af110abcc-1*, 07-Apr-2017. [Online]. Available: https://www.nextbigfuture.com/2011/01/aluminum-encapsulated-ceramic-tiles-for.html. [Accessed: 17-Oct-2019].

[2] “Reinforcement learning tutorial with TensorFlow,” *Adventures in Machine Learning*, 15-Mar-2019. [Online]. Available: https://adventuresinmachinelearning.com/reinforcement-learning-tensorflow/. [Accessed: 22-Dec-2019].

[3]“gym-tetris,” *PyPI*. [Online]. Available: https://pypi.org/project/gym-tetris/. [Accessed: 22-Dec-2019].