**数据结构（C++）**

**课程设计报告**

**设计题目: 基于强化学习的迷宫游戏AI训练**

**院 系: 信息工程学院**

**组 长: 刘文越**

**组 员: 符浩森、崔田雨、卢睿杰**

**指导教师: 安毅生**

**日 期: 2023.1.05**

# 题目

## 设计内容与要求

基于机器学习中的强化学习（Reinforcement Learning, 即RL）部分，训练机器人自发学习寻找迷宫终点。

## 实验环境

1）硬件环境：

笔记本电脑

* 处理器：英特尔 Core i7-10870H @ 2.20GHz 八核
* 主 板：微星 MS-17K2
* 内 存：32 GB
* 硬 盘：三星 MZVLB2T0HMLB-00000
* 显 卡：Nvidia GeForce RTX 3070 Laptop GPU

2）软件环境

* Windows 10 系统
* Visual Studio Code
* Python 3.11.1
* Qt 5.14.2

## 分工及组内得分

（介绍每位组员的角色划分，以及组内得分。组内得分需要符合正态分布）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 组长 | 组员一 | 组员二 | 组员三 |
| 姓名 | 刘文越 | 符浩森 | 崔田雨 | 卢睿杰 |
| 学号 | 2021902610 | 2021904170 | 2021901365 | 2021904516 |
| 组内得分 | A | B | B | C |
| 组员分工 | Runner类的编写、界面UI的编写、报告撰写 | Maze类的编写、PPT制作 | Runner类的编写、报告撰写 | Robot类的编写、PPT制作 |

## 系统分析与设计

强化学习是一种经典的机器学习方法，它关注智能体和环境之间的交互，目标一般为追求最大回报。换而言之，强化学习是一种学习如何从状态映射到行为以使得获取的奖励最大的学习机制。这样的一个主体需要不断地在环境中进行实验，通过环境给予的反馈（奖励）来不断优化“状态-行为”的对应关系。因此，反复实验（rial and error）和延迟奖励（delayed reward）是强化学习最重要的两个特征。

对于迷宫游戏，我们可以通过对迷宫的每个点位设置不同的回报值，而在强化游戏中，AI会自发追求最大回报，因此可以实现自发寻找终点，避开陷阱的效果。

强化学习系统一般包括四个要素：策略（policy），奖励（reward），价值（value），环境（environment）。而在本软件中其四要素对应的对象为：

* 策略（policy）：Q-learning算法
* 奖励（reward）：为终点赋予较高回报值
* 价值（value）：对于迷宫点位赋予的回报值
* 环境（environment）：生成的迷宫环境

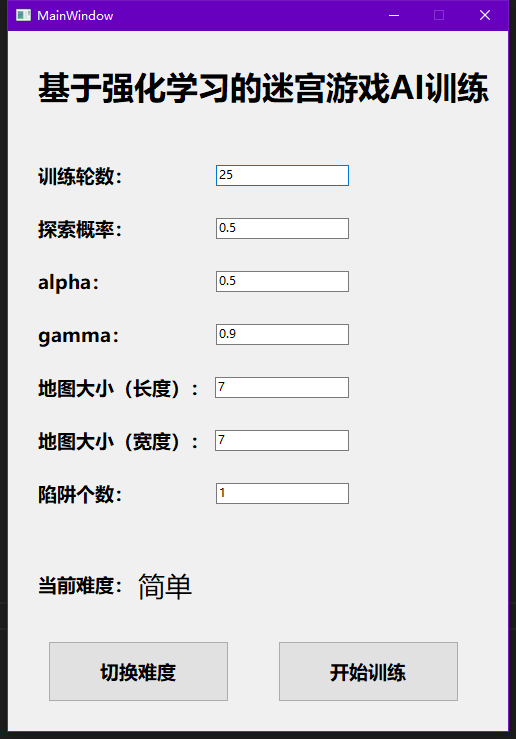
其中，Q-learning算法是一种基于价值的算法。在算法中，Q函数有两个参数（state, action），分别代表状态和决策，其值代表AI在当前状态下采取决策获得的价值。而算法的目标即为最大化价值来获取较高的回报值。而Q-learning算法采用Q table记录不同状态和不同决策下的价值，在初始化Q table后，训练主体在环境中不断探索，并使用贝尔曼方程（ballman equation）来迭代，随着迭代不断拟合Q函数，最终达到收敛或迭代结束。此方法在迷宫问题中效率极佳，故本系统采用Q-learning算法来训练AI。

针对问题分析，我们首先通过查阅资料的方式，确定采用强化学习中的Q-learning算法作为系统的核心算法。而对于训练后结果，可视化过程也非常重要， 对训练结果进行可视化操作，可以使用户更方便快捷的理解训练过程和训练结果，所以我们认为系统有必要进行可视化操作。最后，近些年来python语言在机器学习和可视化方面的优异表现，我们选用python作为系统的主要语言。

对于课设的实现过程，我们首先将问题进行模块化分析，由于python是强对象语言，于是我们将实现过程转化为几个关键类的构造，对于迷宫游戏，将其拆解为迷宫（Maze），机器人（Robot），程序框架（Runner），面板（UI）四个模块，分别进行编写。最后程序的实现由四个对象之间的交互完成。

## 运行结果

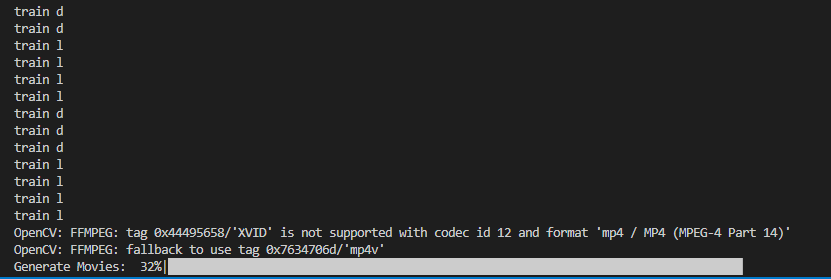
主界面：



进行机器人学习公式和地图的基本配置，可对参数直接修改。

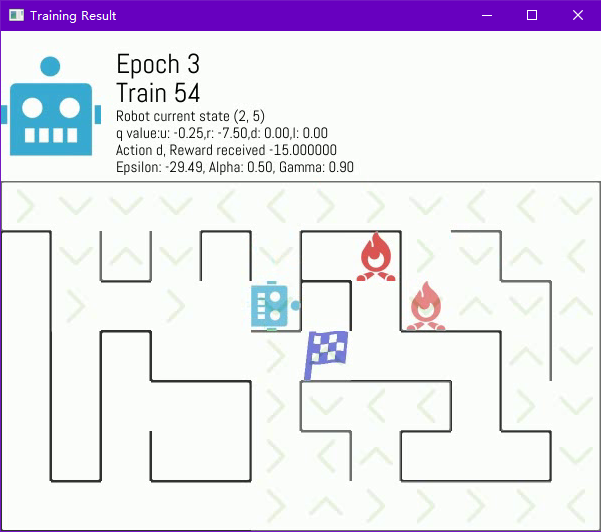
开始训练后，

进行强化学习，模拟迷宫游戏，并对结果进行可视化，生成结果视频并自动播放。



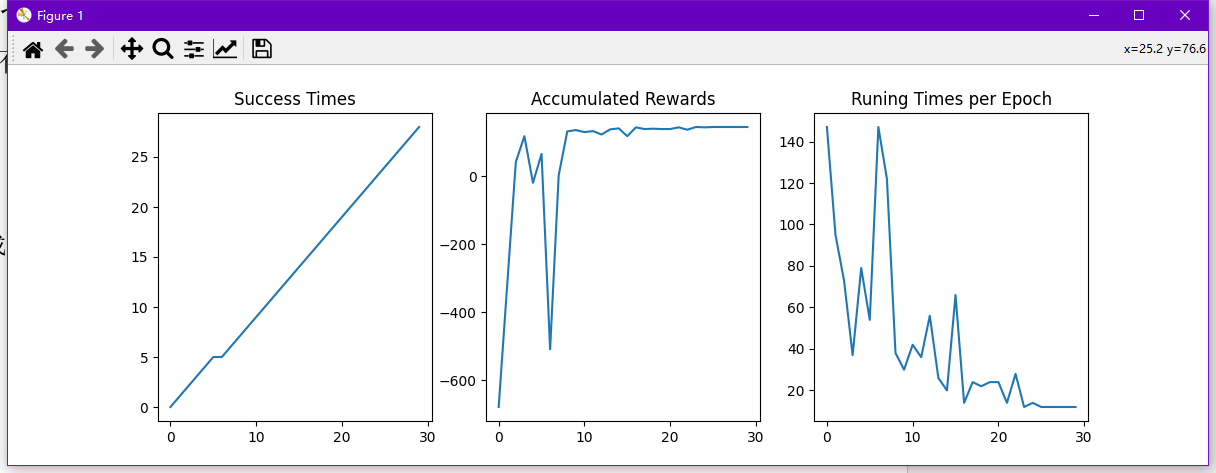
可视化界面：

上方左侧为训练轮数和决策数，右侧显示为机器人当前的状态



生成视频可在软件根目录下再次打开播放。

最后会根据训练过程中的轮次数据得到收敛图像并且呈现出来



## 总结

**6.1 课程设计支撑的毕业要求指标点**

数据结构课程设计支撑以下四项毕业要求指标点：

指标点2-4：能够从工程科学的角度，结合文献查阅及研究，对计算机领域复杂工程问题进行系统分析，并获得有效结论。

指标点7-1：知晓和理解环境保护和可持续发展的理念和内涵，正确认识计算机科学技术的发展与环境和可持续发展的关系。

指标点9-3：能够理解个人在团队中的角色划分，且胜任相应的角色职责。

指标点12-1：能在社会发展的大背景下，理解终身学习的必要性，具有自主学习和终身学习的意识。

**6.2心得**

1. 在程序编写过程中，《Python简明教程》提供了Python的基本使用方法和语言教程，帮助我们更快的理解并运用Python语言。机器学习和强化学习的博客帮助我们认识并学习强化学习的相关知识，对于课设的问题进行分析。此外，Python和第三方库的文档帮助规范代码，介绍函数接口的作用与调用。
2. Python是一门优秀的强对象语言，软件程序的实现实际上是我们编写的几个关键对象之间的交互，通过封装成类，再实例化为对象，有效优化了小组内合作的效率，并且降低了未来对软件进行维护的成本，做到了程序的可持续发展。而Q-learning算法在迷宫游戏中效率优异，大大减少了时间训练的成本，同样促进了程序的可持续发展。  
    通过本次软件的编写，我们深刻意识到了程序的编写需要规范化的代码语言，在封装类时要留好方便调用的接口函数，以便进行合作和未来维护。选取一个适合且优异的算法也同样重要，优秀的效率可以大大减少程序的成本。
3. 团队在组建时首先确定好小组长，以便划分任务和组织规划。其次小组开会讨论课设的需求，需要完成哪些功能，可能会出现哪些困难等，之后我们将软件依据功能和细节进行模块化划分，基于以上内容划分小组成员各自的内容，在代码书写过程中注意留好供别人调用的接口函数，描述好功能以直接使用，遇到问题及时在交流群内展开讨论。最后，由小组长汇总各自的代码，调试功能，分版本发布在群中方便大家进行bug测试。在此过程中，小组成员学习到了合作与分工的重要性，学会了如何将问题条理细化，认识到了个人与团体是紧密不可分割的，为未来学习与工作生活中的合作打下了良好的基础。
4. 在本次程序编写过程中，我们的主要自学内容有Python语言的使用、机器学习与强化学习、Qt语言的使用、Python各模板库的使用等。  
    在科学技术飞速发展的现在，自学能力显然是当代大学生必不可少的一项技能，而如何在生活中注重培养自学能力，需要我们明确自己的需求和目的、详细认真制定学习计划、妥善利用各种学习渠道、学会自我复盘平时学习的细节和错误、拥有创造性思维和研究新事物的思路。

## 程序源代码

程序由多个Python文件构成：（忽略编译过程中的生成文件）

* main.py

from Maze import Maze

from Robot import Robot

from Runner import Runner

from PySide6.QtWidgets import QMainWindow, QApplication

from ui import Ui\_MainWindow

import cv2

class MyMainWindow(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.ui = Ui\_MainWindow()

        self.ui.setupUi(self)

    def startTraining(self):

        """

        开始训练

        """

        g = Maze(

            maze\_size=(int(self.ui.lineEdit\_mazeLon.text()), int(self.ui.lineEdit\_mazeWid.text())),

            trap\_num=int(self.ui.lineEdit\_trapNumber.text())

            )

        r = Robot(

            maze = g,

            alpha = float(self.ui.lineEdit\_alpha.text()),

            epsilon0 = float(self.ui.lineEdit\_epsilon0.text()),

            gamma = float(self.ui.lineEdit\_gamma.text())

            )

        r.set\_status(learning=True)

        runner = Runner(r, g)

        runner.run\_training(int(self.ui.lineEdit\_epoch.text()), display\_direction=True)

        runner.generate\_movie(filename = "finalResult.mp4")

        runner.plot\_results()

        cap = cv2.VideoCapture('finalResult.mp4')

        while(cap.isOpened()):

            ret, frame = cap.read()

            if ret:

                cv2.imshow('Training Result', frame)

                cv2.waitKey(30)

            else:

                break

        cap.release()

        cv2.destroyAllWindows()

    def changeDifficulty(self):

        self.ui.stackedWidget.setCurrentIndex((self.ui.stackedWidget.currentIndex() + 1) % 3)

        if self.ui.stackedWidget.currentIndex() == 0:

            self.ui.lineEdit\_mazeLon.setText('7')

            self.ui.lineEdit\_mazeWid.setText('7')

            self.ui.lineEdit\_trapNumber.setText('1')

            self.ui.lineEdit\_epoch.setText('25')

        elif self.ui.stackedWidget.currentIndex() == 1:

            self.ui.lineEdit\_mazeLon.setText('7')

            self.ui.lineEdit\_mazeWid.setText('12')

            self.ui.lineEdit\_trapNumber.setText('2')

            self.ui.lineEdit\_epoch.setText('30')

        else:

            self.ui.lineEdit\_mazeLon.setText('11')

            self.ui.lineEdit\_mazeWid.setText('11')

            self.ui.lineEdit\_trapNumber.setText('3')

            self.ui.lineEdit\_epoch.setText('50')

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication()

    window = MyMainWindow()

    window.show()

    app.exec\_()

* Maze.py

import random

from os.path import join

import imageio.v2 as imageio

import numpy as np

from matplotlib import pyplot as plt

class Maze(object):

    valid\_actions = ['u', 'r', 'd', 'l']  # 机器人动作，包含向上，向右，向下，向左

    direction\_bit\_map = {'u': 1, 'r': 2, 'd': 4, 'l': 8}  # 设置位图

    move\_map = {'u': (-1, 0), 'r': (0, +1), 'd': (+1, 0), 'l': (0, -1)}  # 记录移动地图

    robot\_img = {d: imageio.imread(join("images/", "robot-" + d + ".jpg")) for d in valid\_actions}  # 读入图片

    def \_\_init\_\_(self, maze\_size=(6, 6), trap\_num=3):

        self.raw\_maze\_img = None  # 记录此时迷宫图片

        self.traps = None  # 记录陷阱

        self.destination = None  # 记录目标地点

        self.maze\_data = None  # 记录迷宫数据

        self.reward = None  # 记录奖惩

        self.robot = None  # 记录机器人，位置和朝向

        self.generate\_maze(maze\_size[0] \* 2 + 1, maze\_size[1] \* 2 + 1)  # 创建迷宫

        self.height, self.width = maze\_size  # 设置宽高

        self.default\_robot\_loc = {

            'loc': (1, self.width - 2),

            'dir': 'd',

        }  # 默认机器人位置，行进方向

        self.set\_destination()  # 设置目标地点

        self.generate\_trap(trap\_num=trap\_num)  # 制造陷阱

        self.draw\_raw\_maze\_img()  # 展示出图片

        self.place\_robot()  # 放置机器人

        self.set\_reward()  # 设置奖惩

    # 此处生成随机的迷宫

    def generate\_maze(self, height, width, complexity=.25, density=.25):

        # 调整形状大小为奇数

        shape = ((height // 2) \* 2 + 1, (width // 2) \* 2 + 1)

        # 调整复杂度

        complexity = int(complexity \* (5 \* (shape[0] + shape[1])))

        # 调整密集程度

        density = int(density \* ((shape[0] // 2) \* (shape[1] // 2)))

        # 创建迷宫

        amap = np.zeros(shape, dtype=bool)

        # 填充边界

        amap[0, :] = amap[-1, :] = 1

        amap[:, 0] = amap[:, -1] = 1

        # 制作出通道

        for i in range(density):

            x, y = random.randint(0, shape[1] // 2) \* 2, random.randint(0, shape[0] // 2) \* 2

            amap[y, x] = 1

            for j in range(complexity):

                neighbours = []

                if x > 1:

                    neighbours.append((y, x - 2))

                if x < shape[1] - 2:

                    neighbours.append((y, x + 2))

                if y > 1:

                    neighbours.append((y - 2, x))

                if y < shape[0] - 2:

                    neighbours.append((y + 2, x))

                if len(neighbours):

                    y\_, x\_ = neighbours[random.randint(0, len(neighbours) - 1)]

                    if amap[y\_, x\_] == 0:

                        amap[y\_, x\_] = 1

                        amap[y\_ + (y - y\_) // 2, x\_ + (x - x\_) // 2] = 1

                        x, y = x\_, y\_

        row, col = amap.shape

        # 转换迷宫格式以供记录

        maze\_data = np.zeros(((row - 3) // 2 + 1, (col - 3) // 2 + 1), dtype=np.uint8)

        for i in range(0, row - 2, 2):

            for j in range(0, col - 2, 2):

                maze\_data[i // 2, j // 2] = sum([1, 2, 4, 8][i] \* ~block for i, block in

                                                enumerate(np.ravel(amap[i:i + 3, j:j + 3], order='F')[[3, 7, 5, 1]]))

        self.maze\_data = maze\_data

    def set\_destination(self, destination\_coord=None):

        # 设置目地点坐标，默认处在中心位置

        if not destination\_coord:

            destination\_coord = (self.height // 2, self.width // 2)

        self.destination = destination\_coord

    def generate\_trap(self, trap\_num):

        # 随机制造陷阱

        if trap\_num > self.width \* self.height \* 0.1:

            raise ValueError('陷阱数太多')

        avoid\_location = int(self.default\_robot\_loc['loc'][0] \* self.width + self.default\_robot\_loc['loc'][1])

        destination = int(self.destination[0] \* self.width + self.destination[1])

        valid\_range = list(range(1, avoid\_location)) + list(range(avoid\_location + 1, destination)) + list(

            range(destination + 1, int((self.width - 1) \* (self.height - 1))))

        trap\_list = random.sample(valid\_range, trap\_num)

        self.traps = [(ele // self.width, ele % self.width) for ele in trap\_list]

    def draw\_raw\_maze\_img(self):

        # 加载图像

        grid\_images = []

        for i in range(16):

            grid\_images.append(imageio.imread(join("images/", str(i) + ".jpg")))

        maze = np.vstack((np.hstack((grid\_images[i] for i in row)) for row in self.maze\_data))

        # 添加陷阱和目标位置的图片

        trap\_img = imageio.imread(join("images", "trap.jpg"))

        destination\_img = imageio.imread(join("images", "destination.jpg"))

        grid\_size = 100  # 设置默认的大小

        for (row, col) in self.traps:

            maze[row \* grid\_size:(row + 1) \* grid\_size, col \* grid\_size:(col + 1) \* grid\_size, :] += trap\_img

        row, col = self.destination

        maze[row \* grid\_size:(row + 1) \* grid\_size, col \* grid\_size:(col + 1) \* grid\_size, :] += destination\_img

        self.raw\_maze\_img = maze

    def get\_raw\_maze\_img(self):

        # 获取迷宫图片，提供调用

        return self.raw\_maze\_img.copy()

    # 116-132行代码为单独展示迷宫图片部分

    def draw\_current\_maze(self):

        # 获取迷宫图片并添加进机器人图片

        grid\_size = 100  # 默认大小

        row, col = self.robot['loc']

        current\_maze\_img = self.raw\_maze\_img.copy()

        current\_maze\_img[row \* grid\_size:(row + 1) \* grid\_size, col \* grid\_size:(col + 1) \* grid\_size, :] += \

            self.robot\_img[self.robot['dir']]

        return current\_maze\_img

    def \_\_repr\_\_(self):

        # 展示生成好的迷宫

        plt.figure(figsize=(self.height, self.width))

        plt.imshow(self.draw\_current\_maze())

        plt.axis('off')

        plt.show()

        return 'Maze of size (%d, %d)' % (self.height, self.width)

    def is\_permissible(self, location, direction):

        # 判断接下来的一次行动是否会撞到墙壁

        return (self.maze\_data[location] & self.direction\_bit\_map[direction]) != 0

    def place\_robot(self):

        # 将机器人放置在迷宫中

        self.robot = self.default\_robot\_loc.copy()

    def set\_reward(self):

        # 对于不同的行为设置奖惩

        self.reward = {

            "hit\_wall": -10.,

            "destination": 50.,

            "trap": -30.,

            "default": -0.1,

        }

    def move\_robot(self, direction):

        # 根据不确定行为移动机器人，返回新位置和移动的奖励

        if direction not in self.valid\_actions:

            raise ValueError("Invalid Actions")

        if self.is\_permissible(self.robot['loc'], direction):

            self.robot['loc'] = tuple((i + di for i, di in zip(self.robot['loc'], self.move\_map[direction])))

            self.robot['dir'] = direction

            if self.robot['loc'] == self.destination:

                reward = self.reward['destination']

            elif self.robot['loc'] in self.traps:

                reward = self.reward['trap']

            else:

                reward = self.reward['default']

        else:

            self.robot['dir'] = direction

            reward = self.reward['hit\_wall']

        return reward

    def sense\_robot(self):

        # 调用后返回机器人位置

        return self.robot['loc']

    def reset\_robot(self):

        # 重置机器人的位置和行进方向

        self.robot = self.default\_robot\_loc.copy()

* Robot.py

import random

class Robot(object):

    def \_\_init\_\_(self,

                 maze,

                 alpha=0.5,

                 gamma=0.9,

                 epsilon0=0.5):  #设置默认松张变量，折扣因子，概率

        self.maze = maze  #机器人所在迷宫对象

        self.valid\_actions = self.maze.valid\_actions  #可进行的action

        #初始化机器人状态和动作

        self.state = None

        self.action = None

        self.alpha = alpha  #学习率

        self.gamma = gamma  #折扣因子

        self.epsilon0 = epsilon0  #初始探索概率

        self.epsilon = epsilon0  #探索概率，和动作选择有关

        self.t = 0  #已探索次数

        self.Qtable = {}  #Q table表

        self.reset()

    def reset(self):

        """

        初始化机器人

        """

        self.state = self.sense\_state()

        self.create\_Qtable\_line(self.state)

    def set\_status(self, learning=False, testing=False):

        """

        确定机器人是否在学习q表,

        或执行测试程序。

        """

        self.learning = learning

        self.testing = testing

    def update\_parameter(self):

        """

        q学习机器人的一些参数可以被改变,

        必要时更新这些参数。

        """

        if self.testing:

            #如果不是探索状态，则跳过

            pass

        else:

            self.t += 1

            if self.epsilon < 0.01:

                self.epsilon = 0.01

            else:

                self.epsilon -= self.t \* 0.1

                #根据已探索次数减少探索概率

        return self.epsilon

    def sense\_state(self):

        """

        返回机器人当前状态

        """

        return self.maze.sense\_robot()

    def create\_Qtable\_line(self, state):

        """

        在Qtable中添加新行

        """

        if state in self.Qtable: #机器人状态存在

            pass

        else:

            self.Qtable[state] = {'u': 0.0, 'r': 0.0, 'd': 0.0, 'l': 0.0}

    def choose\_action(self):

        """

        机器人进行动作选择，返回选择的动作

        """

        def is\_random\_exploration():

            #随机选择是否进行探索

            return random.random() < self.epsilon

        if self.learning:  #机器人进行探索情况

            if is\_random\_exploration():  #进行探索

                action = random.choice(self.valid\_actions)  #随机选择一个动作进行探索

                print("random: ", action)

                return action

            else:  #不进行探索

                action = max(

                    self.Qtable[self.state],

                    key=self.Qtable[self.state].get

                    )  #返回Qtable表中有着最大价值的action

                print("train", action)

                return action

        elif self.testing:  #机器人进行利用情况

            action = max(

                self.Qtable[self.state],

                key=self.Qtable[self.state].get

                )

            print("text:", action)

            return action

        else:

            action = random.choice(self.valid\_actions)

            print("non\_all", action)

            return action

    def update\_Qtable(self, r, action, next\_state):

        """

        依据当前规则更新Qtable

        """

        if self.learning:  #在探索情况下才更新

            current\_r = self.Qtable[self.state][action]  #获取当前的回报

            update\_r = r + (self.gamma \* float(

                max(self.Qtable[next\_state].values())))  #获取更新的回报

            self.Qtable[self.state][action] += self.alpha \* (

                update\_r - current\_r)  #写入Qtable表

    def update(self):

        """

        机器人Qtable表更新函数，返回现在的动作和奖励

        """

        self.state = self.sense\_state()  #获取现在的状态

        self.create\_Qtable\_line(self.state)  #如果当前状态不在Q table中，则添加进去

        action = self.choose\_action()  #针对当前状态选择动作

        reward = self.maze.move\_robot(action)  #根据选择的动作在迷宫中移动机器人

        next\_state = self.sense\_state()  #获取下一状态

        self.create\_Qtable\_line(next\_state)  #如果下一状态不在Q table中，则添加进去

        if self.learning and not self.testing:  #如果是学习状态

            self.update\_Qtable(reward, action, next\_state)  #更新Qtable表

            self.update\_parameter()  #减少探索概率，并更新该参数

        return action, reward

* Runner.py

import numpy as np

import random

import imageio.v2 as imageio

from multiprocessing import Pool, cpu\_count

import tqdm

from os.path import join

from copy import deepcopy

from PIL import Image

from PIL import ImageFont

from PIL import ImageDraw

import cv2

import matplotlib.pyplot as plt

#训练机器人，实现整个训练过程及可视化。在当前文件夹中生成一个名为 filename 的视频，记录了整个训练的过程。

class Runner(object):

    valid\_actions = ['u', 'r', 'd', 'l'] # 规定动作：上，右，下，左

    robot\_img = {d:imageio.imread(join("images/","robot-"+d+".jpg")) for d in valid\_actions}

    logo\_img = imageio.imread("images/logo.jpg")

    arrow\_img = {d:imageio.imread(join("images/","arrow-"+d+".jpg")) for d in valid\_actions}

    header\_font = ImageFont.truetype("abel-regular.ttf", 55)

    font = ImageFont.truetype("abel-regular.ttf", 30)

    def \_\_init\_\_(self, robot, maze):

        self.maze = maze#迷宫

        self.robot = robot#机器人

    #训练（训练次数，每次训练机器人最多移动的步数，显示方向

    def run\_training(self, training\_epoch, training\_per\_epoch=300, display\_direction=False):

        training\_per\_epoch = int(self.maze.height \* self.maze.height \* 3.0)

        #训练记录

        self.train\_robot\_record = {}  #type:dict

        #训练机器的状态记录，成功，奖励值，次数，这些参数

        self.train\_robot\_statics = {

            'success': [],

            'reward': [],

            'times': [],

        }#type:dict

        self.display\_direction = display\_direction

        #训练之前的记录

        def train\_logger\_before\_act(e, i):

            self.train\_robot\_record[(e,i)] = {}

            self.train\_robot\_record[(e,i)]['id'] = (e,i)

            self.train\_robot\_record[(e,i)]['success'] = False

            self.train\_robot\_record[(e,i)]['state'] = self.robot.sense\_state()

            self.train\_robot\_record[(e,i)]['qtable'] = self.robot.Qtable[self.robot.sense\_state()].copy()

            self.train\_robot\_record[(e,i)]['epsilon'] = self.robot.epsilon

            self.train\_robot\_record[(e,i)]['alpha'] = self.robot.alpha

            self.train\_robot\_record[(e,i)]['gamma'] = self.robot.gamma

            self.train\_robot\_record[(e,i)]['maze\_loc'] = self.maze.robot.copy()

            if self.display\_direction:

                self.train\_robot\_record[(e,i)]['Qtable'] = deepcopy(self.robot.Qtable)

        def train\_logger\_after\_act(e, i, action, reward):

            self.train\_robot\_record[(e,i)]['action'] = action

            self.train\_robot\_record[(e,i)]['reward'] = reward

        for e in range(training\_epoch):

            accumulated\_reward = 0

            run\_time = 0

            for i in range(training\_per\_epoch):

                train\_logger\_before\_act(e, i)

                action, reward = self.robot.update()

                train\_logger\_after\_act(e, i, action, reward)

                run\_time += 1

                accumulated\_reward += reward

                if self.maze.robot['loc'] == self.maze.destination:

                    i+=1

                    train\_logger\_before\_act(e,i)

                    self.train\_robot\_record[(e,i)]['success'] = True

                    break

            if self.maze.robot['loc'] == self.maze.destination:

                self.train\_robot\_statics['success'].append(1)

            else:

                self.train\_robot\_statics['success'].append(0)

            self.train\_robot\_statics['reward'].append(accumulated\_reward)

            self.train\_robot\_statics['times'].append(run\_time)

            self.maze.reset\_robot()

            self.robot.reset()

#测试机器人能否走出迷宫

    def run\_testing(self, testing\_per\_epoch):

        self.test\_robot\_statics = {}

        self.test\_robot\_statics['success'] = []

        self.test\_robot\_statics['reward'] = []

        self.test\_robot\_statics['times'] = []

        self.robot.set\_status(learning=False, testing=True)

        testing\_per\_epoch = int(self.maze.height \* self.maze.height \* 0.85)

        accumulated\_reward = 0.

        run\_time = 0

        for i in range(testing\_per\_epoch):

            run\_time += 1

            \_, reward = self.robot.update()

            accumulated\_reward += reward

            if self.maze.robot['loc'] == self.maze.destination:

                break

        if self.maze.robot['loc'] == self.maze.destination:

            self.test\_robot\_statics['success'].append(1)

        else:

            self.test\_robot\_statics['success'].append(0)

        self.test\_robot\_statics['reward'].append(accumulated\_reward)

        self.test\_robot\_statics['times'].append(run\_time)

#界面各项指标，图标的显示

    def draw\_header(self, base\_image):

        logo\_size = 200 # 默认logo大小200

        logo\_image = np.vstack((Image.new('RGB', (200,50), color=(255,255,255)),self.logo\_img))# 图像的设置

        logo\_image = np.vstack((logo\_image,Image.new('RGB', (200,50), color=(255,255,255))))

        header\_shape = (base\_image.shape[1]-logo\_size, logo\_size+100) # 形状的高度和宽度

        header\_img = np.hstack((logo\_image, Image.new('RGB', header\_shape, color=(255,255,255))))

        return header\_img

# 在迷宫中绘制机器人

    def draw\_robot(self, base\_image, parameters):

        img = base\_image.copy()

        robot = parameters['maze\_loc']

        grid\_size = 100

        r,c = robot['loc']

        img[r\*grid\_size:(r+1)\*grid\_size, c\*grid\_size:(c+1)\*grid\_size, :] += self.robot\_img[robot['dir']]

        if self.display\_direction:

            for state, q in parameters['Qtable'].items():

                r,c = state

                direction = max(q, key=q.get)

                img[r\*grid\_size:(r+1)\*grid\_size, c\*grid\_size:(c+1)\*grid\_size, :] = \

                    (0.3\*self.arrow\_img[direction] + 0.7\*img[r\*grid\_size:(r+1)\*grid\_size, c\*grid\_size:(c+1)\*grid\_size, :]).astype('uint8')

        return img

# 界面的文字记录

    def write\_on\_header(self, header\_img, parameters):

        header = Image.fromarray(header\_img.copy())

        draw = ImageDraw.Draw(header)

        line0 = "Epoch %d \nTrain %d \n"%(parameters['id'][0]+1, parameters['id'][1]+1)

        line1 = "Robot current state %s \n"%(str(parameters['state']))

        line2 = "q value:" + ','.join([a + ": %.2f"%q for a,q in parameters['qtable'].items()])

        if not parameters['success']:

            line3 = "\nAction %s, Reward received %f \n"%(parameters['action'],parameters['reward'])

            line4 = "Epsilon: %.2f, Alpha: %.2f, Gamma: %.2f"%(parameters['epsilon'],parameters['alpha'],parameters['gamma'])

        else:

            line3, line4 = '', ''

        draw.text((230, 30), line0, (0,0,0), font=self.header\_font)

        draw.text((230, 150), line1+line2+line3+line4, (0,0,0), font=self.font)

        return np.array(header)

#将训练结果生成视频

    def generate\_movie(self, filename):

        base\_image = self.maze.get\_raw\_maze\_img()

        header\_img = self.draw\_header(base\_image)

        def ensemble\_image(train\_id):

            new\_header = self.write\_on\_header(header\_img, self.train\_robot\_record[train\_id])

            current\_maze = self.draw\_robot(base\_image, self.train\_robot\_record[train\_id])

            return (train\_id, np.vstack((new\_header, current\_maze)))

        height, width, \_ = ensemble\_image((0,0))[1].shape

        writer = cv2.VideoWriter(filename, cv2.VideoWriter\_fourcc(\*"XVID"), 25.0, (width//2,height//2))

        #单线程导出视频

        # with tqdm.tqdm(sorted(self.train\_robot\_record.keys()), desc="Generate Movies") as pbar:

            #     for key in pbar:

            #         \_, img = ensemble\_image(key)

            #         writer.write(cv2.resize(img[:,:,::-1], (width//2, height//2)))

        # 多进程优化，内存使用优化：在工作流程之上构建的池之上模拟工作进程

        with Pool(processes=max(cpu\_count()-1,1)) as pool:

            with tqdm.tqdm(pool.imap(ensemble\_image, self.train\_robot\_record.keys()), total=len(self.train\_robot\_record.keys()), desc="Generate Movies") as pbar:

                for \_, img in pbar:

                    writer.write(cv2.resize(img[:,:,::-1], (width//2, height//2)))

        writer.release()

#以图表展示训练过程中的指标

    def plot\_results(self):

        plt.figure(figsize=(12,4))

        plt.subplot(131)

        plt.title("Success Times")

        plt.plot(np.cumsum(self.train\_robot\_statics['success']))

        plt.subplot(132)

        plt.title("Accumulated Rewards")

        plt.plot(np.array(self.train\_robot\_statics['reward']))

        plt.subplot(133)

        plt.title("Runing Times per Epoch")

        plt.plot(np.array(self.train\_robot\_statics['times']))

        plt.show()

* ui.py

from PySide6.QtCore import (QCoreApplication, QDate, QDateTime, QLocale,

    QMetaObject, QObject, QPoint, QRect,

    QSize, QTime, QUrl, Qt)

from PySide6.QtGui import (QBrush, QColor, QConicalGradient, QCursor,

    QFont, QFontDatabase, QGradient, QIcon,

    QImage, QKeySequence, QLinearGradient, QPainter,

    QPalette, QPixmap, QRadialGradient, QTransform)

from PySide6.QtWidgets import (QApplication, QHBoxLayout, QLabel, QLineEdit,

    QMainWindow, QPushButton, QSizePolicy, QSpacerItem,

    QStackedWidget, QVBoxLayout, QWidget)

class Ui\_MainWindow(object):

    def setupUi(self, MainWindow):

        if not MainWindow.objectName():

            MainWindow.setObjectName(u"MainWindow")

        MainWindow.resize(500, 700)

        sizePolicy = QSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)

        sizePolicy.setHorizontalStretch(0)

        sizePolicy.setVerticalStretch(0)

        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())

        MainWindow.setSizePolicy(sizePolicy)

        MainWindow.setMinimumSize(QSize(500, 700))

        MainWindow.setMaximumSize(QSize(500, 700))

        self.centralwidget = QWidget(MainWindow)

        self.centralwidget.setObjectName(u"centralwidget")

        sizePolicy.setHeightForWidth(self.centralwidget.sizePolicy().hasHeightForWidth())

        self.centralwidget.setSizePolicy(sizePolicy)

        self.centralwidget.setMinimumSize(QSize(500, 700))

        self.centralwidget.setMaximumSize(QSize(500, 700))

        self.verticalLayoutWidget = QWidget(self.centralwidget)

        self.verticalLayoutWidget.setObjectName(u"verticalLayoutWidget")

        self.verticalLayoutWidget.setGeometry(QRect(30, 110, 311, 391))

        self.verticalLayout = QVBoxLayout(self.verticalLayoutWidget)

        self.verticalLayout.setObjectName(u"verticalLayout")

        self.verticalLayout.setContentsMargins(0, 0, 0, 0)

        self.horizontalLayout = QHBoxLayout()

        self.horizontalLayout.setObjectName(u"horizontalLayout")

        self.label = QLabel(self.verticalLayoutWidget)

        self.label.setObjectName(u"label")

        font = QFont()

        font.setFamilies([u"Microsoft YaHei UI"])

        font.setPointSize(14)

        font.setBold(True)

        self.label.setFont(font)

        self.horizontalLayout.addWidget(self.label)

        self.horizontalSpacer = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)

        self.horizontalLayout.addItem(self.horizontalSpacer)

        self.lineEdit\_epoch = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_epoch.setObjectName(u"lineEdit\_epoch")

        self.horizontalLayout.addWidget(self.lineEdit\_epoch)

        self.verticalLayout.addLayout(self.horizontalLayout)

        self.horizontalLayout\_2 = QHBoxLayout()

        self.horizontalLayout\_2.setObjectName(u"horizontalLayout\_2")

        self.label\_2 = QLabel(self.verticalLayoutWidget)

        self.label\_2.setObjectName(u"label\_2")

        font1 = QFont()

        font1.setPointSize(14)

        font1.setBold(True)

        self.label\_2.setFont(font1)

        self.horizontalLayout\_2.addWidget(self.label\_2)

        self.horizontalSpacer\_2 = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)

        self.horizontalLayout\_2.addItem(self.horizontalSpacer\_2)

        self.lineEdit\_epsilon0 = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_epsilon0.setObjectName(u"lineEdit\_epsilon0")

        self.horizontalLayout\_2.addWidget(self.lineEdit\_epsilon0)

        self.verticalLayout.addLayout(self.horizontalLayout\_2)

        self.horizontalLayout\_3 = QHBoxLayout()

        self.horizontalLayout\_3.setObjectName(u"horizontalLayout\_3")

        self.label\_3 = QLabel(self.verticalLayoutWidget)

        self.label\_3.setObjectName(u"label\_3")

        self.label\_3.setFont(font1)

        self.horizontalLayout\_3.addWidget(self.label\_3)

        self.horizontalSpacer\_3 = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)

        self.horizontalLayout\_3.addItem(self.horizontalSpacer\_3)

        self.lineEdit\_alpha = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_alpha.setObjectName(u"lineEdit\_alpha")

        self.horizontalLayout\_3.addWidget(self.lineEdit\_alpha)

        self.verticalLayout.addLayout(self.horizontalLayout\_3)

        self.horizontalLayout\_4 = QHBoxLayout()

        self.horizontalLayout\_4.setObjectName(u"horizontalLayout\_4")

        self.label\_4 = QLabel(self.verticalLayoutWidget)

        self.label\_4.setObjectName(u"label\_4")

        self.label\_4.setFont(font1)

        self.horizontalLayout\_4.addWidget(self.label\_4)

        self.horizontalSpacer\_4 = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)

        self.horizontalLayout\_4.addItem(self.horizontalSpacer\_4)

        self.lineEdit\_gamma = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_gamma.setObjectName(u"lineEdit\_gamma")

        self.horizontalLayout\_4.addWidget(self.lineEdit\_gamma)

        self.verticalLayout.addLayout(self.horizontalLayout\_4)

        self.horizontalLayout\_5 = QHBoxLayout()

        self.horizontalLayout\_5.setObjectName(u"horizontalLayout\_5")

        self.label\_5 = QLabel(self.verticalLayoutWidget)

        self.label\_5.setObjectName(u"label\_5")

        self.label\_5.setFont(font1)

        self.horizontalLayout\_5.addWidget(self.label\_5)

        self.lineEdit\_mazeLon = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_mazeLon.setObjectName(u"lineEdit\_mazeLon")

        self.horizontalLayout\_5.addWidget(self.lineEdit\_mazeLon)

        self.verticalLayout.addLayout(self.horizontalLayout\_5)

        self.horizontalLayout\_7 = QHBoxLayout()

        self.horizontalLayout\_7.setObjectName(u"horizontalLayout\_7")

        self.label\_6 = QLabel(self.verticalLayoutWidget)

        self.label\_6.setObjectName(u"label\_6")

        self.label\_6.setFont(font1)

        self.horizontalLayout\_7.addWidget(self.label\_6)

        self.lineEdit\_mazeWid = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_mazeWid.setObjectName(u"lineEdit\_mazeWid")

        self.horizontalLayout\_7.addWidget(self.lineEdit\_mazeWid)

        self.verticalLayout.addLayout(self.horizontalLayout\_7)

        self.horizontalLayout\_8 = QHBoxLayout()

        self.horizontalLayout\_8.setObjectName(u"horizontalLayout\_8")

        self.label\_7 = QLabel(self.verticalLayoutWidget)

        self.label\_7.setObjectName(u"label\_7")

        self.label\_7.setFont(font1)

        self.horizontalLayout\_8.addWidget(self.label\_7)

        self.horizontalSpacer\_5 = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)

        self.horizontalLayout\_8.addItem(self.horizontalSpacer\_5)

        self.lineEdit\_trapNumber = QLineEdit(self.verticalLayoutWidget)

        self.lineEdit\_trapNumber.setObjectName(u"lineEdit\_trapNumber")

        self.horizontalLayout\_8.addWidget(self.lineEdit\_trapNumber)

        self.verticalLayout.addLayout(self.horizontalLayout\_8)

        self.PB\_start = QPushButton(self.centralwidget)

        self.PB\_start.setObjectName(u"PB\_start")

        self.PB\_start.setGeometry(QRect(270, 610, 181, 61))

        palette = QPalette()

        brush = QBrush(QColor(0, 0, 0, 255))

        brush.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.WindowText, brush)

        brush1 = QBrush(QColor(170, 255, 255, 255))

        brush1.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.Button, brush1)

        brush2 = QBrush(QColor(255, 255, 255, 255))

        brush2.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.Light, brush2)

        brush3 = QBrush(QColor(212, 255, 255, 255))

        brush3.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.Midlight, brush3)

        brush4 = QBrush(QColor(85, 127, 127, 255))

        brush4.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.Dark, brush4)

        brush5 = QBrush(QColor(113, 170, 170, 255))

        brush5.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.Mid, brush5)

        palette.setBrush(QPalette.Active, QPalette.Text, brush)

        palette.setBrush(QPalette.Active, QPalette.BrightText, brush2)

        palette.setBrush(QPalette.Active, QPalette.ButtonText, brush)

        palette.setBrush(QPalette.Active, QPalette.Base, brush2)

        palette.setBrush(QPalette.Active, QPalette.Window, brush1)

        palette.setBrush(QPalette.Active, QPalette.Shadow, brush)

        palette.setBrush(QPalette.Active, QPalette.AlternateBase, brush3)

        brush6 = QBrush(QColor(255, 255, 220, 255))

        brush6.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Active, QPalette.ToolTipBase, brush6)

        palette.setBrush(QPalette.Active, QPalette.ToolTipText, brush)

        brush7 = QBrush(QColor(0, 0, 0, 127))

        brush7.setStyle(Qt.SolidPattern)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette.setBrush(QPalette.Active, QPalette.PlaceholderText, brush7)

#endif

        palette.setBrush(QPalette.Inactive, QPalette.WindowText, brush)

        brush8 = QBrush(QColor(240, 240, 240, 255))

        brush8.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Inactive, QPalette.Button, brush8)

        palette.setBrush(QPalette.Inactive, QPalette.Light, brush2)

        brush9 = QBrush(QColor(227, 227, 227, 255))

        brush9.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Inactive, QPalette.Midlight, brush9)

        brush10 = QBrush(QColor(160, 160, 160, 255))

        brush10.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Inactive, QPalette.Dark, brush10)

        palette.setBrush(QPalette.Inactive, QPalette.Mid, brush10)

        palette.setBrush(QPalette.Inactive, QPalette.Text, brush)

        palette.setBrush(QPalette.Inactive, QPalette.BrightText, brush2)

        palette.setBrush(QPalette.Inactive, QPalette.ButtonText, brush)

        palette.setBrush(QPalette.Inactive, QPalette.Base, brush2)

        palette.setBrush(QPalette.Inactive, QPalette.Window, brush8)

        brush11 = QBrush(QColor(105, 105, 105, 255))

        brush11.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Inactive, QPalette.Shadow, brush11)

        brush12 = QBrush(QColor(245, 245, 245, 255))

        brush12.setStyle(Qt.SolidPattern)

        palette.setBrush(QPalette.Inactive, QPalette.AlternateBase, brush12)

        palette.setBrush(QPalette.Inactive, QPalette.ToolTipBase, brush6)

        palette.setBrush(QPalette.Inactive, QPalette.ToolTipText, brush)

        brush13 = QBrush(QColor(0, 0, 0, 128))

        brush13.setStyle(Qt.SolidPattern)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette.setBrush(QPalette.Inactive, QPalette.PlaceholderText, brush13)

#endif

        palette.setBrush(QPalette.Disabled, QPalette.WindowText, brush4)

        palette.setBrush(QPalette.Disabled, QPalette.Button, brush1)

        palette.setBrush(QPalette.Disabled, QPalette.Light, brush2)

        palette.setBrush(QPalette.Disabled, QPalette.Midlight, brush3)

        palette.setBrush(QPalette.Disabled, QPalette.Dark, brush4)

        palette.setBrush(QPalette.Disabled, QPalette.Mid, brush5)

        palette.setBrush(QPalette.Disabled, QPalette.Text, brush4)

        palette.setBrush(QPalette.Disabled, QPalette.BrightText, brush2)

        palette.setBrush(QPalette.Disabled, QPalette.ButtonText, brush4)

        palette.setBrush(QPalette.Disabled, QPalette.Base, brush1)

        palette.setBrush(QPalette.Disabled, QPalette.Window, brush1)

        palette.setBrush(QPalette.Disabled, QPalette.Shadow, brush)

        palette.setBrush(QPalette.Disabled, QPalette.AlternateBase, brush12)

        palette.setBrush(QPalette.Disabled, QPalette.ToolTipBase, brush6)

        palette.setBrush(QPalette.Disabled, QPalette.ToolTipText, brush)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette.setBrush(QPalette.Disabled, QPalette.PlaceholderText, brush13)

#endif

        self.PB\_start.setPalette(palette)

        self.PB\_start.setFont(font1)

        self.label\_8 = QLabel(self.centralwidget)

        self.label\_8.setObjectName(u"label\_8")

        self.label\_8.setGeometry(QRect(30, 30, 461, 51))

        font2 = QFont()

        font2.setPointSize(24)

        font2.setBold(True)

        self.label\_8.setFont(font2)

        self.PB\_change = QPushButton(self.centralwidget)

        self.PB\_change.setObjectName(u"PB\_change")

        self.PB\_change.setGeometry(QRect(40, 610, 181, 61))

        palette1 = QPalette()

        palette1.setBrush(QPalette.Active, QPalette.WindowText, brush)

        palette1.setBrush(QPalette.Active, QPalette.Button, brush1)

        palette1.setBrush(QPalette.Active, QPalette.Light, brush2)

        palette1.setBrush(QPalette.Active, QPalette.Midlight, brush3)

        palette1.setBrush(QPalette.Active, QPalette.Dark, brush4)

        palette1.setBrush(QPalette.Active, QPalette.Mid, brush5)

        palette1.setBrush(QPalette.Active, QPalette.Text, brush)

        palette1.setBrush(QPalette.Active, QPalette.BrightText, brush2)

        palette1.setBrush(QPalette.Active, QPalette.ButtonText, brush)

        palette1.setBrush(QPalette.Active, QPalette.Base, brush2)

        palette1.setBrush(QPalette.Active, QPalette.Window, brush1)

        palette1.setBrush(QPalette.Active, QPalette.Shadow, brush)

        palette1.setBrush(QPalette.Active, QPalette.AlternateBase, brush3)

        palette1.setBrush(QPalette.Active, QPalette.ToolTipBase, brush6)

        palette1.setBrush(QPalette.Active, QPalette.ToolTipText, brush)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette1.setBrush(QPalette.Active, QPalette.PlaceholderText, brush7)

#endif

        palette1.setBrush(QPalette.Inactive, QPalette.WindowText, brush)

        palette1.setBrush(QPalette.Inactive, QPalette.Button, brush8)

        palette1.setBrush(QPalette.Inactive, QPalette.Light, brush2)

        palette1.setBrush(QPalette.Inactive, QPalette.Midlight, brush9)

        palette1.setBrush(QPalette.Inactive, QPalette.Dark, brush10)

        palette1.setBrush(QPalette.Inactive, QPalette.Mid, brush10)

        palette1.setBrush(QPalette.Inactive, QPalette.Text, brush)

        palette1.setBrush(QPalette.Inactive, QPalette.BrightText, brush2)

        palette1.setBrush(QPalette.Inactive, QPalette.ButtonText, brush)

        palette1.setBrush(QPalette.Inactive, QPalette.Base, brush2)

        palette1.setBrush(QPalette.Inactive, QPalette.Window, brush8)

        palette1.setBrush(QPalette.Inactive, QPalette.Shadow, brush11)

        palette1.setBrush(QPalette.Inactive, QPalette.AlternateBase, brush12)

        palette1.setBrush(QPalette.Inactive, QPalette.ToolTipBase, brush6)

        palette1.setBrush(QPalette.Inactive, QPalette.ToolTipText, brush)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette1.setBrush(QPalette.Inactive, QPalette.PlaceholderText, brush13)

#endif

        palette1.setBrush(QPalette.Disabled, QPalette.WindowText, brush4)

        palette1.setBrush(QPalette.Disabled, QPalette.Button, brush1)

        palette1.setBrush(QPalette.Disabled, QPalette.Light, brush2)

        palette1.setBrush(QPalette.Disabled, QPalette.Midlight, brush3)

        palette1.setBrush(QPalette.Disabled, QPalette.Dark, brush4)

        palette1.setBrush(QPalette.Disabled, QPalette.Mid, brush5)

        palette1.setBrush(QPalette.Disabled, QPalette.Text, brush4)

        palette1.setBrush(QPalette.Disabled, QPalette.BrightText, brush2)

        palette1.setBrush(QPalette.Disabled, QPalette.ButtonText, brush4)

        palette1.setBrush(QPalette.Disabled, QPalette.Base, brush1)

        palette1.setBrush(QPalette.Disabled, QPalette.Window, brush1)

        palette1.setBrush(QPalette.Disabled, QPalette.Shadow, brush)

        palette1.setBrush(QPalette.Disabled, QPalette.AlternateBase, brush12)

        palette1.setBrush(QPalette.Disabled, QPalette.ToolTipBase, brush6)

        palette1.setBrush(QPalette.Disabled, QPalette.ToolTipText, brush)

#if QT\_VERSION >= QT\_VERSION\_CHECK(5, 12, 0)

        palette1.setBrush(QPalette.Disabled, QPalette.PlaceholderText, brush13)

#endif

        self.PB\_change.setPalette(palette1)

        self.PB\_change.setFont(font1)

        self.label\_9 = QLabel(self.centralwidget)

        self.label\_9.setObjectName(u"label\_9")

        self.label\_9.setGeometry(QRect(30, 530, 95, 48))

        self.label\_9.setFont(font1)

        self.stackedWidget = QStackedWidget(self.centralwidget)

        self.stackedWidget.setObjectName(u"stackedWidget")

        self.stackedWidget.setGeometry(QRect(120, 520, 111, 71))

        self.page = QWidget()

        self.page.setObjectName(u"page")

        self.label\_10 = QLabel(self.page)

        self.label\_10.setObjectName(u"label\_10")

        self.label\_10.setGeometry(QRect(10, 10, 71, 48))

        font3 = QFont()

        font3.setFamilies([u"\u5fae\u8f6f\u96c5\u9ed1"])

        font3.setPointSize(20)

        font3.setBold(False)

        self.label\_10.setFont(font3)

        self.stackedWidget.addWidget(self.page)

        self.page\_2 = QWidget()

        self.page\_2.setObjectName(u"page\_2")

        self.label\_11 = QLabel(self.page\_2)

        self.label\_11.setObjectName(u"label\_11")

        self.label\_11.setGeometry(QRect(10, 10, 71, 48))

        self.label\_11.setFont(font3)

        self.stackedWidget.addWidget(self.page\_2)

        self.page\_3 = QWidget()

        self.page\_3.setObjectName(u"page\_3")

        self.label\_12 = QLabel(self.page\_3)

        self.label\_12.setObjectName(u"label\_12")

        self.label\_12.setGeometry(QRect(10, 10, 71, 48))

        self.label\_12.setFont(font3)

        self.stackedWidget.addWidget(self.page\_3)

        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)

        self.stackedWidget.setCurrentIndex(0)

        self.lineEdit\_epoch.setText("25")

        self.lineEdit\_epsilon0.setText("0.5")

        self.lineEdit\_alpha.setText("0.5")

        self.lineEdit\_gamma.setText("0.9")

        self.lineEdit\_mazeLon.setText("7")

        self.lineEdit\_mazeWid.setText("7")

        self.lineEdit\_trapNumber.setText("1")

        self.PB\_start.clicked.connect(MainWindow.startTraining)

        self.PB\_change.clicked.connect(MainWindow.changeDifficulty)

        QMetaObject.connectSlotsByName(MainWindow)

    # setupUi

    def retranslateUi(self, MainWindow):

        MainWindow.setWindowTitle(QCoreApplication.translate("MainWindow", u"MainWindow", None))

        self.label.setText(QCoreApplication.translate("MainWindow", u"\u8bad\u7ec3\u8f6e\u6570\uff1a", None))

        self.label\_2.setText(QCoreApplication.translate("MainWindow", u"\u63a2\u7d22\u6982\u7387\uff1a", None))

        self.label\_3.setText(QCoreApplication.translate("MainWindow", u"alpha\uff1a", None))

        self.label\_4.setText(QCoreApplication.translate("MainWindow", u"gamma\uff1a", None))

        self.label\_5.setText(QCoreApplication.translate("MainWindow", u"\u5730\u56fe\u5927\u5c0f\uff08\u957f\u5ea6\uff09\uff1a", None))

        self.label\_6.setText(QCoreApplication.translate("MainWindow", u"\u5730\u56fe\u5927\u5c0f\uff08\u5bbd\u5ea6\uff09\uff1a", None))

        self.label\_7.setText(QCoreApplication.translate("MainWindow", u"\u9677\u9631\u4e2a\u6570\uff1a", None))

        self.PB\_start.setText(QCoreApplication.translate("MainWindow", u"\u5f00\u59cb\u8bad\u7ec3", None))

        self.label\_8.setText(QCoreApplication.translate("MainWindow", u"\u57fa\u4e8e\u5f3a\u5316\u5b66\u4e60\u7684\u8ff7\u5bab\u6e38\u620fAI\u8bad\u7ec3", None))

        self.PB\_change.setText(QCoreApplication.translate("MainWindow", u"\u5207\u6362\u96be\u5ea6", None))

        self.label\_9.setText(QCoreApplication.translate("MainWindow", u"\u5f53\u524d\u96be\u5ea6\uff1a", None))

        self.label\_10.setText(QCoreApplication.translate("MainWindow", u"\u7b80\u5355", None))

        self.label\_11.setText(QCoreApplication.translate("MainWindow", u"\u4e2d\u7b49", None))

        self.label\_12.setText(QCoreApplication.translate("MainWindow", u"\u56f0\u96be", None))

    # retranslateUi