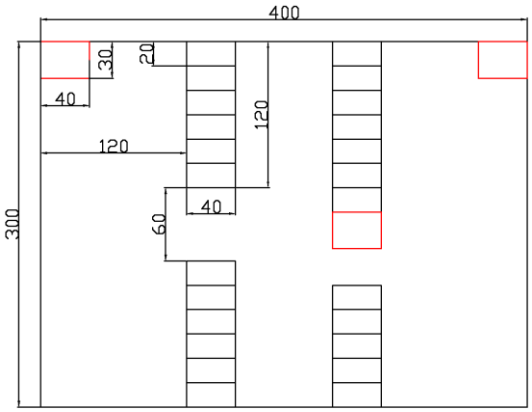
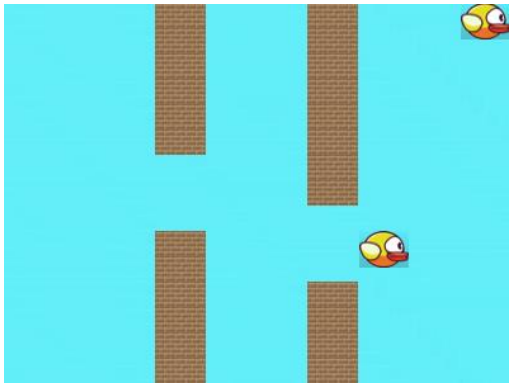
强化学习环境搭建

可以用 gym、pygame、pyglet 搭建自己的强化学习环境，也可以用 gym 的开源环境如 CartPole-v1 等。

Pygame

Pygame 是跨平台 Python 模块，专为电子游戏设计，包含图像、声音。我们用 Pygame 搭建一个迷宫环境 yuanyang。整个仿真画面的尺寸为 400x300，小鸟的尺寸为 40x30，每一块砖的尺寸为 40x20。上下障碍之间留出 60 的空隙，所以需要设置 12 块砖。小鸟一次移动的距离为 X:40, Y:30，共有 100 个状态。





1、加载图像

加载后的图像都是通过坐标固定在画布上的，每个图像块的坐标锚点都在其左上角，可建立数组存储其坐标值。初始化好所有坐标之后，就可以写一个渲染函数来渲染。

```
# 利用 os 模块得到得到当前工程目录
current_dir = os.path.split(os.path.realpath(__file__))[0]    # print(current_dir)
# 得到文件名
bird_file = current_dir + "/resources/bird.png"
obstacle_file = current_dir + "/resources/obstacle.png"
background_file = current_dir + "/resources/background.png"
# 加载图像
bird = pygame.image.load(bird_file).convert_alpha()
self.obstacle1_x = []   # 用于存放左边障碍物的坐标
self.obstacle1_y = []
```

2、初始化

该步初始化动作空间、状态空间、所有图元的初始坐标

```
    def __init__(self):   # 设置程序需要用到的各种基本参数
        self.viewer = None
# 控制渲染频率，体现在小鸟移动快慢，渲染时会有 self.FPSCLOCK.tick(30)
        self.FPSCLOCK = pygame.time.Clock()   # 控制渲染频率，体现在小鸟移动快慢
```

```
        self.actions = [0, 1, 2, 3]    # e,s,w,n  →↓←↑
        self.state = 0    # 设置当前状态
        self.states = [i for i in range(100)]
        # 屏幕大小
        self.screen_size = (400, 300)
        self.limit_distance_x = 40    # 用于限制小鸟与障碍物之间的距离
        self.limit_distance_y = 30
        # self.obstacle_size = [40, 20]    # 每一块砖的尺寸
        self.obstacle1_x = []    # 用于存放障碍物的坐标
        self.obstacle1_y = []
        self.obstacle2_x = []
        self.obstacle2_y = []
        # 设置障碍物和小鸟的坐标位置
        for i in range(12):
            # 左边上下障碍物
            self.obstacle1_x.append(120)
            if i <= 5:
                self.obstacle1_y.append(20 * i)
            else:
                self.obstacle1_y.append(20 * (i + 3))
            # 右边上下障碍物
            self.obstacle2_x.append(240)
            if i <= 7:
                self.obstacle2_y.append(20 * i)
            else:
                self.obstacle2_y.append(20 * (i + 3))

        self.bird_male_position = [0, 0]
        self.bird_female_init_position = [360, 0]
```

3、渲染函数

```
    def render(self):
        if self.viewer is None:    # 第一次初始化时
            pygame.init()
            # 画一个窗口
            self.viewer = pygame.display.set_mode(self.screen_size, 0, 32)
            # 下载图片
            self.bird_male = load_bird_male()
            self.bird_female = load_bird_female()
            self.background = load_background()
            self.obstacle = load_obstacle()
            # 在幕布上画图片，（图片，坐标）
            self.viewer.blit(self.bird_female, self.bird_female_init_position)
            self.viewer.blit(self.background, (0, 0))

        # 擦除
        self.viewer.blit(self.background, (0, 0))
        # 画终点
        self.viewer.blit(self.bird_female, self.bird_female_init_position)
        # 画障碍物
        for i in range(12):
            self.viewer.blit(self.obstacle, (self.obstacle1_x[i], self.obstacle1_y[i]))
            self.viewer.blit(self.obstacle, (self.obstacle2_x[i], self.obstacle2_y[i]))
```

```
        self.viewer.blit(self.bird_male, self.bird_male_position)
        pygame.display.update()
        time.sleep(0.1)    # 延时
        self.FPSCLOCK.tick(30)
```

4、状态与坐标之间的相互转换

```
    # 将状态转换为坐标值
    def state_to_position(self, state):
        i = int(state / 10)
        j = state % 10
        position = [0, 0]
        position[0] = 40 * j
        position[1] = 30 * i
        return position

    # 将坐标值转换为状态
    def position_to_state(self, position):
        i = position[0] / 40
        j = position[1] / 30
        return int(i + 10 * j)
```

5、编写判断是否结束的代码

```
    # 判断是否出界或撞墙
    def collide(self, state_position):
        flag = 1    是否相撞
        flag1 = 1    # 是否与第一个障碍物相撞
        flag2 = 1    # 是否与第二个障碍物相撞
        # 判断第一个障碍物
        dx, dy = [], []
        for i in range(12):    # 与每一个砖块计算距离
            dx1 = abs(self.obstacle1_x[i] - state_position[0])
            dx.append(dx1)
            dy1 = abs(self.obstacle1_y[i] - state_position[1])
            dy.append(dy1)
        mindx = min(dx)
        mindy = min(dy)
        if mindx >= self.limit_distance_x or mindy >= self.limit_distance_y:
            flag1 = 0
        # 判断第二个障碍物
        second_dx, second_dy = [], []
        for i in range(12):
            dx2 = abs(self.obstacle2_x[i] - state_position[0])
            second_dx.append(dx2)
            dy2 = abs(self.obstacle2_y[i] - state_position[1])
            second_dy.append(dy2)
        mindx = min(second_dx)
        mindy = min(second_dy)
        if mindx >= self.limit_distance_x or mindy >= self.limit_distance_y:
            flag2 = 0
        if flag1 == 0 and flag2 == 0:    # 与左右两边障碍物都未相撞时
            flag = 0
```

```
            if state_position[0] > 360 or state_position[0] < 0 or state_position[1] > 270 or
state_position[1] < 0:   # 如果出界了
                flag = 1
            return flag
    # 判断是否找到目标物
        def find(self, state_position):
            flag = 0
            if abs(state_position[0] - self.bird_female_init_position[0]) < self.limit_distance_x and
                abs(state_position[1] - self.bird_female_init_position[1]) < self.limit_distance_y:
                    flag = 1
            return flag
    # 当出界/撞墙/找到时都会结束本回合
        def is_terminal(self, s):
            flag = 0
            flag1 = self.collide(self.state_to_position(s))
            flag2 = self.find(self.state_to_position(s))
            if flag1 == 1 or flag2 == 1:
                flag = 1
            return flag
```

6、reset 函数

```
        def reset(self):   # 碰撞或者找到之后就初始化
            while 1:
                self.state = self.states[int(random.random() * len(self.states))]
                if not self.is_terminal(self.state):
                    break
            return self.state
```

7、step 函数

```
    # step()函数的输入是动作，输出是下一个时刻的状态、回报、是否终止和调试信息
        def step(self, action):
            # 将当前状态转化为坐标
            current_position = self.state_to_position(self.state)
            next_position = [0, 0]
            if action == 0:
                next_position[0] = current_position[0] + 40
                next_position[1] = current_position[1]
            if action == 1:
                next_position[0] = current_position[0]
                next_position[1] = current_position[1] + 30
            if action == 2:
                next_position[0] = current_position[0] - 40
                next_position[1] = current_position[1]
            if action == 3:
                next_position[0] = current_position[0]
                next_position[1] = current_position[1] - 30
            # 判断 next_state 是否与障碍物碰撞
            flag_collide = self.collide(next_position)
            # 如果碰撞，仍会回到当前状态并且回报为-1，并结束
            if flag_collide == 1:
                return self.state, -1, True

            self.state = self.position_to_state(next_position)
```

```
# 判断是否终点
flag_find = self.find(next_position)
if flag_find == 1:
        return self.state, 1, True
return self.state, 0, False
```