

Level 4 & 5 实验报告

姓名：蓝俊玮 学号：PB20111689

实验环境：Windows 10 Python 3.9.0 Pycharm Community Edition 2021

1. 实验选题

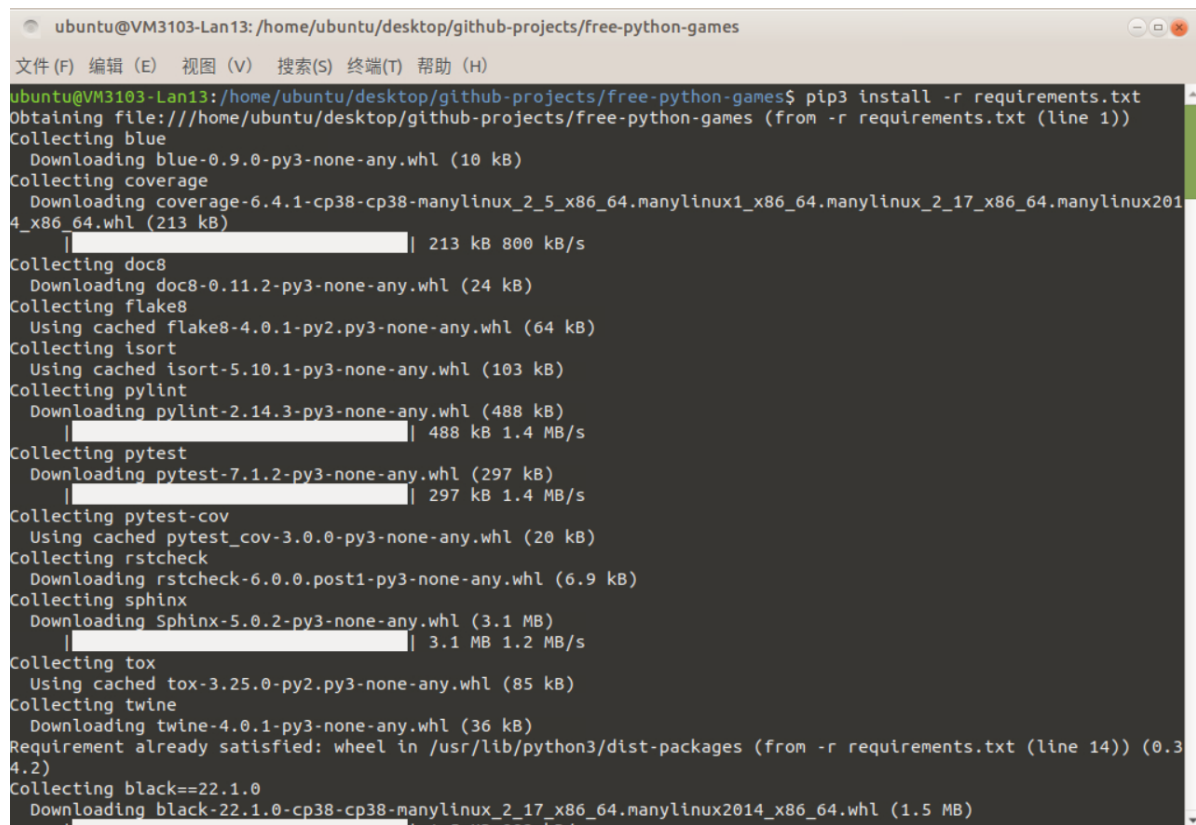
游戏应用 Free Python Games

Free Python Games 是一个包含了很多小游戏的开源平台，这些游戏是用简单的 Python 代码编写的，旨在进行实验和更改。包含的游戏有吃豆人，井字棋，Flappy 与一些简化的街机游戏等。

2. Level 4 部署环境与展示

直接将 github 仓库的文件都克隆到本地使用：

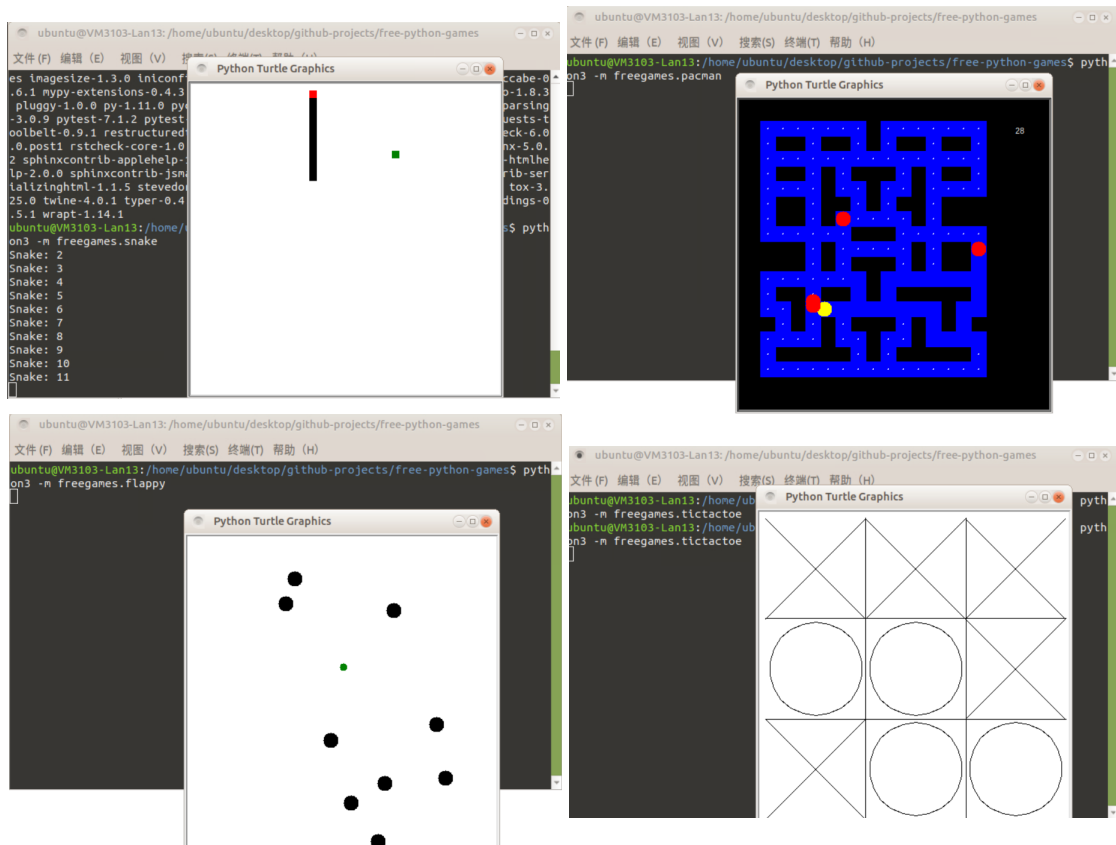
```
git clone https://github.com/grantjenks/free-python-games.git
cd free-python-games/
pip3 install -r requirements.txt
```

A terminal window screenshot showing the execution of 'pip3 install -r requirements.txt'. The terminal output lists various dependencies being collected and downloaded, including blue, coverage, doc8, flake8, isort, pylint, pytest, pytest-cov, rstcheck, sphinx, tox, twine, and black. Progress bars and download speeds are visible for several packages.

```
ubuntu@VM3103-Lan13: /home/ubuntu/desktop/github-projects/free-python-games
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
ubuntu@VM3103-Lan13:/home/ubuntu/desktop/github-projects/free-python-games$ pip3 install -r requirements.txt
Obtaining file:///home/ubuntu/desktop/github-projects/free-python-games (from -r requirements.txt (line 1))
Collecting blue
  Downloading blue-0.9.0-py3-none-any.whl (10 kB)
Collecting coverage
  Downloading coverage-6.4.1-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (213 kB)
  | 213 kB 800 kB/s
Collecting doc8
  Downloading doc8-0.11.2-py3-none-any.whl (24 kB)
Collecting flake8
  Using cached flake8-4.0.1-py2.py3-none-any.whl (64 kB)
Collecting isort
  Using cached isort-5.10.1-py3-none-any.whl (103 kB)
Collecting pylint
  Downloading pylint-2.14.3-py3-none-any.whl (488 kB)
  | 488 kB 1.4 MB/s
Collecting pytest
  Downloading pytest-7.1.2-py3-none-any.whl (297 kB)
  | 297 kB 1.4 MB/s
Collecting pytest-cov
  Using cached pytest_cov-3.0.0-py3-none-any.whl (20 kB)
Collecting rstcheck
  Downloading rstcheck-6.0.0.post1-py3-none-any.whl (6.9 kB)
Collecting sphinx
  Downloading Sphinx-5.0.2-py3-none-any.whl (3.1 MB)
  | 3.1 MB 1.2 MB/s
Collecting tox
  Using cached tox-3.25.0-py2.py3-none-any.whl (85 kB)
Collecting twine
  Downloading twine-4.0.1-py3-none-any.whl (36 kB)
Requirement already satisfied: wheel in /usr/lib/python3/dist-packages (from -r requirements.txt (line 14)) (0.34.2)
Collecting black==22.1.0
  Downloading black-22.1.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.5 MB)
  | 1.5 MB 883 kB/s
```

接着运行即可试玩小游戏：

```
python3 -m freegames.snake
python3 -m freegames.pacman
python3 -m freegames.flappy
python3 -m freegames.tictactoe
```



3. Level 5 实验思路

`free-python-games` 是使用 `turtle` 库实现的，根据 `free-python-games` 项目内的游戏，我选择用 `Pygame` 复现其中的贪吃蛇游戏，并且修复了 `free-python-games` 中 `snake.py` 设计中出现的问题，同时为其增加了本机双人同时竞技的功能，并且使用 `Pygame` 为其增添一个新的 2048 游戏。

3.1 贪吃蛇思路

3.1.1 贪吃蛇设计

```
def __init__(self, x, y, direction):
    self.Red = color("red") # 红色 用来表示蛇头部的颜色
    self.Black = color("black") # 黑色 用来表示蛇身体的颜色
    self.white = color("white") # 白色 背景颜色
    self.Blue = color("blue") # 蓝色 用来表示食物的颜色
    self.body = [[x, y]] # 蛇的整个身体的所有坐标点
    self.head = [x, y] # 蛇的头部坐标点
    self.direction = direction # 蛇当前移动的方向
    self.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
    while self.food in self.body: # 食物的位置，确保不会在蛇的身体内，修改了 free-
python-game 中食物出现的位置
        self.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
    self.eaten = False # 食物是否被吃掉
    self.score = 0 # 当前得分
```

- 用 `Red`, `Black`, `white`, `Blue` 分别来表示蛇头部，蛇身体，游戏背景，食物的颜色。
- 用 `body` 来存储蛇整个身体的所有坐标点
- 用 `head` 来记录蛇头部的坐标点
- 用 `direction` 来记录蛇当前移动的方向
- 用 `food` 来记录食物的坐标点
- 用 `eaten` 来表示当前移动过程中，食物是否被吃掉

其中，我将游戏界面大小设置为 800×600 ，将每个单位点的大小设置为 20，所以整个游戏界面就可以用 40×30 的坐标点来记录表示。无论是蛇身还是食物，都是以该单位进行表示的，同时蛇的移动偏量也是以该单位衡量的。

3.1.2 蛇身移动以及绘制

贪吃蛇中比较困难的逻辑就是蛇身移动以及绘制了，我采用的方法是每次移动时，将蛇移动后的头部 `head` 插入到蛇身体 `body` 的头部，如果在上次移动中，食物没有吃掉，便将蛇身体 `body` 的尾部去除，从而实现蛇身的移动；而当食物被吃掉时，则保留蛇身体 `body` 的尾部。

```
def draw_snake(self, display):
    self.body.insert(0, self.head)      # 把蛇移动后的头部加入到蛇的身体
    if not self.eaten:                  # 如果食物没有被吃掉的话
        self.body.pop()                 # 则身体最后一个点要被删除（即移动了）
    head = True                          # 只是用来标记输出头部的变量
    for segment in self.body:           # 对蛇身体每一部分进行绘图
        if head:
            pygame.draw.rect(display, self.Red, Rect(segment[0], segment[1], 20,
20))
            head = False
        else:
            pygame.draw.rect(display, self.Black, Rect(segment[0], segment[1],
20, 20))
```

3.1.3 贪吃蛇游戏结束

贪吃蛇游戏结束的主要判断依据是根据头部，因为蛇身体所经过的位置都是由蛇头部走过的位置。当蛇头碰到自身或者碰到墙壁时游戏结束，可以通过下面进行判断：

```
def game_over(self):
    if self.head in self.body:
        return True      # 如果移动后的头部碰到蛇的身体，游戏结束
    elif self.head[0] < 0 or self.head[1] < 0:
        return True      # 如果移动后的头部碰到上界和左界，游戏结束
    elif self.head[0] > 780 or self.head[1] > 580:
        return True      # 如果移动后的头部碰到下界和右界，游戏结束
    return False
```

3.1.4 双人贪吃蛇设计

用一个新的类封装双个贪吃蛇，设定 `[W, S, A, D]` 为左边用户的操作，`[K_UP, K_DOWN, K_LEFT, K_RIGHT]` 为右边用户的操作，左边用户贪吃蛇需要吃到蓝色的食物，而右边用户贪吃蛇需要吃到绿色的食物：

```
class Snake2:
    def __init__(self):
        self.snake1 = Snake(200, 300, "DOWN")
        self.snake2 = Snake(600, 300, "DOWN")

    def draw_food(self, display):
        pygame.draw.rect(display, color("blue"), Rect(self.snake1.food[0],
self.snake1.food[1], 20, 20))
        pygame.draw.rect(display, color("green"), Rect(self.snake2.food[0],
self.snake2.food[1], 20, 20))
```

并且改写单人贪吃蛇中游戏判断的方法，现在需要判断其是否碰到另一只蛇的身体。

```
def game_over(self):
    if self.snake1.head in self.snake1.body or self.snake1.head in self.snake2.body:
        return True, "user1" # 如果移动后的头部碰到蛇的身体，游戏结束
    # <-- snip -->

    if self.snake2.head in self.snake1.body or self.snake2.head in self.snake2.body:
        return True, "user2" # 如果移动后的头部碰到蛇的身体，游戏结束
    # <-- snip -->
```

同时食物生成时，除了不能生成到自己的身体内，也不能生成到另一只蛇的身体内：

```
def generate_food(self):
    if self.snake1.eaten: # 如果当前食物被吃了，则生成一个新的食物
        self.snake1.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
        # 食物的位置，确保不会在两只蛇的身体内
        while self.snake1.food in self.snake1.body or self.snake1.food in self.snake2.body:
            self.snake1.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
        self.snake1.eaten = False # 更新食物状态
    if self.snake2.eaten: # 如果当前食物被吃了，则生成一个新的食物
        self.snake2.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
        # 食物的位置，确保不会在两只蛇的身体内
        while self.snake2.food in self.snake1.body or self.snake2.food in self.snake2.body:
            self.snake2.food = [random.randint(1, 39) * 20, random.randint(1, 29) * 20]
        self.snake2.eaten = False # 更新食物状态
```

3.2 2048 思路

3.2.1 2048 设计

```
def __init__(self, display):
    self.font = pygame.font.SysFont("arial", 25) # 字体设置
    self.board = [[0, 0, 0, 0] for _ in range(4)] # 棋盘的初始化
    self.colors = { # 每个数字对应的颜色
        0: (200, 195, 180),
        2: (240, 230, 220),
        4: (235, 225, 200),
        8: (240, 175, 120),
        16: (245, 150, 100),
        32: (245, 125, 95),
        64: (245, 95, 60),
        128: (240, 210, 115),
        256: (235, 205, 100),
        512: (240, 200, 80),
        1024: (240, 200, 60),
        2048: (225, 188, 0)
    }
```

```

self.screen_left_gap = 50      # 棋盘到左边界的距离
self.screen_top_gap = 100     # 棋盘到上边界的距离
self.box_gap = 5              # 每个棋格之间的距离
self.score = 0                # 当前得分

```

- `board` 是 2048 游戏中最主要的部分，用来记录当前游戏界面的情况
- `score` 用来记录当前的得分

3.2.2 数字生成

```

def rand_board(self):
    blank = []      # 首先获取所有空白格子的左边点
    for i in range(4):
        for j in range(4):
            if self.board[i][j] == 0:
                blank.append((i, j))
    choice = random.choice(blank) # 随机从空白格子选取一个
    number = random.uniform(0, 1) # 采用均匀分布，按概率生成随机数
    if number < 0.01:
        number = 8
    elif number < 0.1:
        number = 4
    else:
        number = 2
    self.board[choice[0]][choice[1]] = number

```

数字生成方式采用双重随机的方式，即生成位置以及生成数字都是随机的方式。每次只在空白格的位置生成一个随机数字，并且增加了 8 的生成，可以再增加游戏的趣味性。

3.2.3 数字合并

2048 游戏中比较困难的就是移动时的方格合并了，采用枚举的方法，将移动时所有可能的情况都列举出来：

值得注意的是，将 `box_set` 参数传入时，需要根据当前合并的方向，对 `board` 进行有序的传入。

首先是计算需要移动的行或列中，有几个待合并的数字，然后根据合并后的情况，将其存储到 `result` 中返回：

```

def combine(self, box_set):
    result = [0, 0, 0, 0] # 合并后的情况
    number = []          # 该行或列中不为 0 的数字
    for box in box_set:
        if box != 0:
            number.append(box)

```

- 当待合并的数字中有四个数时，首先对第一个数字和第二个数字进行判断：

```

if number[0] == number[1]:      # 如果第一个和第二个相等
    result[0] = number[0] + number[1]    # 合并第一个和第二个
    self.score = self.score + result[0]
if number[2] == number[3]:      # 如果第三个和第四个还想等
    result[1] = number[2] + number[3]    # 合并第三个和第四个
    self.score = self.score + result[1]
else:
    result[1] = number[2]
    result[2] = number[3]

```

接着对第二个和第三个数字，第三个和第四个数字进行判断：

```

elif number[1] == number[2]:    # 如果第二个和第三个相等
    result[0] = number[0]
    result[1] = number[1] + number[2]    # 合并第二个和第三个
    self.score = self.score + result[1]
    result[2] = number[3]
elif number[2] == number[3]:    # 如果第三个和第四个相等
    result[0] = number[0]
    result[1] = number[1]
    result[2] = number[2] + number[3]    # 合并第三个和第四个
    self.score = self.score + result[2]

```

如果待合并的数字不能进行合并时，则依此复制到 `result` 当中：

```

else:      # 如果不能合并，则依此复制
    for i in range(len(number)):
        result[i] = number[i]

```

- 当待合并的数字中有三个数字，两个数字，一个数字，没有数字时，都是类似的枚举方式。

4. Level 5 实验展示

在终端输入即可运行：

```

pip install -r requirements.txt
python snake.py
python snake2.py
python 2048.py

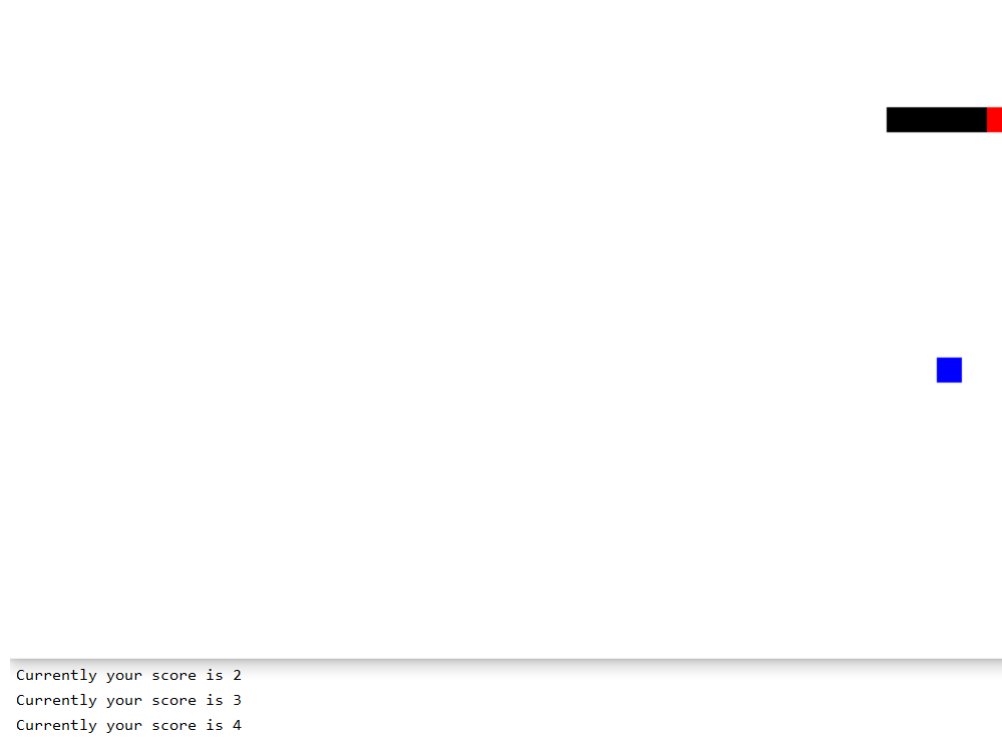
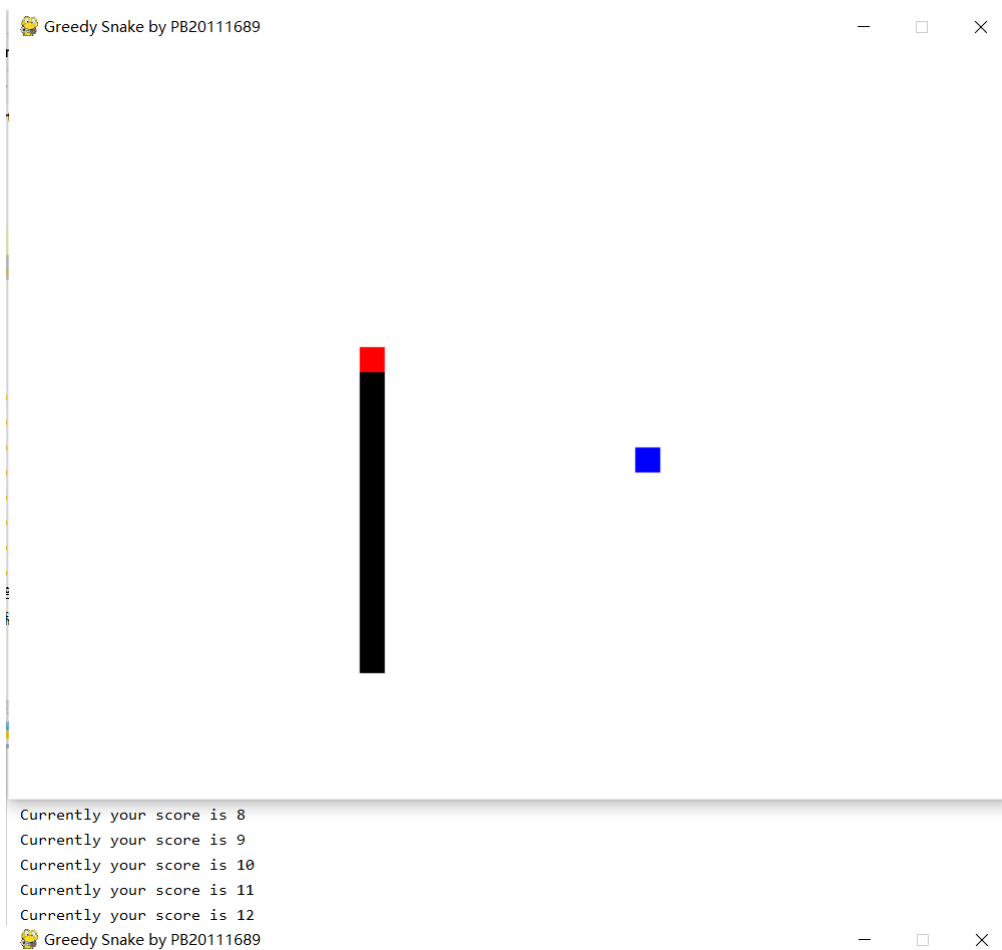
```

实验展示动画放在了百度云盘 https://pan.baidu.com/s/1fQnAoLcM_fl2Onx6OgtH1Q 提取码为 USTC

4.1 贪吃蛇游戏展示

单人游戏时通过 `[W, S, A, D]` 或者 `[K_UP, K_DOWN, K_LEFT, K_RIGHT]` 键控制贪吃蛇的移动（具体动画在附件视频中可看）。

游戏进行以及游戏结束：



双人游戏时，左边用户使用 [W, S, A, D] 键控制贪吃蛇的移动，右边用户使用 [K_UP, K_DOWN, K_LEFT, K_RIGHT] 键控制贪吃蛇的移动（具体动画在附件视频中可看）：



4.2 2048 游戏展示

通过 [W, S, A, D] 或者 [K_UP, K_DOWN, K_LEFT, K_RIGHT] 键控制 2048 游戏进行合并（具体动画在附件视频中可看）。

游戏进行以及游戏结束：

