Super Mario in Python World 作品实习报告

作者：吴悦欣 王旌 张启哲 包逸博

【摘要】本小组以马里奥为题材，Python 为编程语言，制作了内含 5 个关卡的过关小游戏，同时，作品内嵌了商城与卡牌收集系统，玩家可以使用在各关卡中收集到的金币在商城中抽取卡牌，作品也具有存档与重置整个游戏的功能，方便玩家反复游玩，提升游戏体验。

一、选题及创意介绍：

本小组的选题是以马里奥作为主要风格的系列型闯关游戏，主要构想是利用 Pygame Zero 库及其他内置库实现游戏框架、画面以及与玩家间的交互。项目的创意之处体现在作品内含了 5 个小关卡，利用外部框架整合后形成了系列闯关型游戏，全作中有致敬经典的部分，也有自己的创新；同时，作品内嵌了商城系统，玩家可利用在小关卡内获得的金币进行抽卡，丰富了游戏体验；作品还在隐藏了部分彩蛋等待玩家探索，并自己创作了许多背景、按钮与动画效果，完善了细节。

二、设计方案：

1. 外部框架：

游戏通过开始按钮进入主界面，在主界面展示了各关卡的通关情况，以及背包、商城的入口、重置游戏、查看帮助以及退出游戏的按钮。作为闯关式游戏，玩家需按照关卡的顺序依次通过，因此在最开始的游戏界面玩家只能进入第一关，其余关卡随着之前关卡的通关逐步开放。在 5 个小关卡中，包括了 3 个主关卡与 2 个奖励关卡，以主界面地图的主线和支线方式呈现。游戏中玩家在各关卡中获得的金币可累计，并用于在商城中抽取奖励（马里奥卡牌），抽到的奖励可在背包中查看。游戏在各关卡及抽卡后有自动存档功能，确保玩家在本地重新打开游戏时仍保留之前的游戏进度，玩家也可以选择点击主界面的 Reset 按钮重置整个游戏的进度与金币数。

2. 第一关——Mario_VS_Lakitu（简称 MVL）:

玩家通过键盘的向左、向右、Z 键控制马里奥的移动，来应对 Lakitu（刺猬云）的两种不同攻击方式：扔板栗仔和扔刺球。Lakitu 的平均攻击速度会随剩余时间的减少而小幅度上升。界面会由上至下随机掉落金币供玩家收集。游戏中每过一定时间并当马里奥受伤时会掉落蘑菇，拾取蘑菇可使马里奥恢复。玩家坚持 2min 即可过关。

3. 第二关——迷宫：

程序随机生成迷宫，保证一定有从起点到终点的通路。玩家通过键盘的上下左右键控制人物移动。游戏界面每次仅显示迷宫的四分之一，且在此基础上进一步增加局部可见的效果来提升游戏的难度和趣味。迷宫中随机放置一定数量的金币供玩家收集。玩家在 3min 内到达终点即可过关。

4. 第三关——Boss 战：

玩家可通过各方向键操作马里奥躲避 Boss 的攻击并通过 X 键攻击 Boss，

Boss 初始血量为 100，以随机的速度在岩浆上运动，且有两种攻击方式：喷射斜向运动的火焰或会被岩浆反弹一次的火球。马里奥共有 5 条生命来对 Boss 造成伤害，此间 Boss 的平均移动速度会随剩余血量的降低而上升，击败 Boss 即可过关。

5. 奖励关卡一——音游模拟：

程序随机生成带有不同方向键的方块或随机道具，玩家通过上下左右与空格键拾取对应方块或道具，不同的道具有不同的加成效果。游戏中物品的刷新间隔、密集程度和下落速度均会随分数增加而增加，开始时玩家共有 3 条生命，在漏击方块使生命减为 0 前尽可能获取高分，最终得分将会以 500:1 的方式折算为金币作为奖励。

6. 奖励关卡二——炸船：

程序在 10*10 的海面上随机生成 5 艘敌舰，玩家需要在 40 步之内通过鼠标点击摧毁隐藏的敌舰，每次点击投弹后均会有命中反馈（击中不消耗步数），玩家需要根据反馈和海面情况综合评估局势，尽快确定敌舰位置。击沉全部敌舰即获胜，玩家可在获胜后利用剩余步数探索随机出现的隐藏金币，最终所得金币作为奖励赠与玩家。

三、实现方案及代码分析：

1. 外部框架：

程序运行后首先会调用 read 函数进行读档操作，读取之前的通关状态、收集到的卡片和累积的积分，并将五个游戏的类对象都先初始化为 None，接下来的代码由 Mode 变量进行模式的选择。运行 pgzrun.go 后，程序不断调用 draw 和 update 函数；由 on_mouse_down 响应鼠标操作，生成需要的游戏类对象并进行游戏画面和模式之间的切换。Mode 及对应的模式如下：1-主菜单，2-迷宫，3-Boss 战，4-MVL，5-炸船，6-音游模拟，-1-游戏帮助，-2-商店，-3-背包。draw/update 函数会调用对应模式下各个对象内部的 draw/update 函数，或是直接展示对应模式界面所需元素。

```
2425  def update():
2426      if flag_pause:
2427          return
2428      if Mode == -2:
2429          store.draw_store()
2430      elif Mode == 2:
2431          maze_game.play_maze()
2432      elif Mode == 3:
2433          boss_game.update_boss()
2434      elif Mode == 4:
2435          MVL_game.update_MVL()
2436      elif Mode == 5:
2437          Ship_game.update_ships()
2438      elif Mode == 6:
2439          Music_game.update_music()
```

```python
# 根据Mode切换要展示的界面，调用对应的类的成员函数
def draw():
    global Game_Over, end_time, Mode, Win, maze_game, MVL_game, Ship_game, boss_game, Music_game
    hour = time.localtime()[3]
    if 5 <= hour < 7:
        time_mode = 1
    elif 7 <= hour < 18:
        time_mode = 2
    elif 18 <= hour < 20:
        time_mode = 3
    else:
        time_mode = 4
    if flag_pause:
        Main_menu.draw()
        Restart.draw()
        if Mode == 2:
            maze_game.start_time = time.time()
        return
    if Game_Over:
        # 要更换为显示通关、分数、下一关等的界面
        t = time.time() - end_time
        if t > 4:
            Game_Over = False
            music.stop()
            Mode = 1
            del maze_game, boss_game, MVL_game, Music_game, Ship_game
            maze_game = boss_game = MVL_game = Music_game = Ship_game = None
            music.play('main_bgm')
    if Win:
        t = time.time() - end_time
        if t > 4:
            save()
            Win = False
            Mode = 1
            del maze_game, boss_game, MVL_game, Music_game, Ship_game
            maze_game = boss_game = MVL_game = Music_game = Ship_game = None
            music.stop()
            music.play('main_bgm')
    if Mode == 0:
        screen.blit('cover', (0, 0))
        Cover_start.draw()
    if Mode == -1:
        Help_doc.draw()
        Main_menu2.draw()
        return
    if Mode == 1 and not Game_Over:
        if time_mode == 1:
            screen.blit("main_dawn", (0, 0))
        elif time_mode == 2:
            screen.blit("main_daytime", (0, 0))
        elif time_mode == 3:
            screen.blit("main_dusk", (0, 0))
        elif time_mode == 4:
            screen.blit("main_night", (0, 0))
        loading_menu()
    elif Mode == -3:
        screen.blit("main", (0, 0))
        for i in range(7):
            if not Collection[i]:
                Pics[i].image = Locked_pic[i]
            else:
                Pics[i].image = Orig_pics[i]
            Pics[i].draw()
        Main_menu2.draw()
```

除了这些函数外，外部还有：save 函数来实现在每次游戏通关后或是抽卡后保存记录，pause_menu 对应游戏内部的暂停操作，loading_menu 对应主界面的显示，on_key_down 响应部分键盘操作（包括 P 键暂停和继续，以及游戏中的部分键盘操作）。

附：存档与读档功能：

```
2245  def save():
2246      f = open("history.txt", "w")
2247      # 存档，包括每个关卡的过关情况，积分数，商场人物的购买、解锁情况
2248      for i in range(5):
2249          f.write(str(pass_through[i]))
2250          f.write("\n")
2251      for i in range(7):
2252          f.write(str(Collection[i]))
2253          f.write("\n")
2254      f.write(str(Coins))
2255      f.write("\n")
2256      f.close()
2257
2258
2259  def read():
2260      global Coins
2261      try:
2262          with open("history.txt", "r") as f:
2263              i = 0
2264              for a in f:
2265                  if i < 5:
2266                      if a == "False\n":
2267                          pass_through[int(i)] = False
2268                      else:
2269                          pass_through[int(i)] = True
2270                  elif i < 12:
2271                      if a == "False\n":
2272                          Collection[int(i - 5)] = False
2273                      else:
2274                          Collection[int(i - 5)] = True
2275                  else:
2276                      Coins = int(a)
2277                  i += 1
2278      except:
2279          pass
```

利用基本的文件读写操作，将游戏运行核心的参数（各关卡的通关情况、各卡牌的获取情况、金币数）通过文本文件进行存储，实现游戏的存档与读档功能。

2. 迷宫 MAZE 类：

2.1 随机 Prim 生成算法：

算法核心：初始化时使所有地方都是墙，即不可走通，之后选择一个单元格作为迷宫的通路，将与其相邻的四面墙放入列表中。当列表不为空的时候，不断从中选取一个：若该墙面两侧的单元格只有一个被访问过，则从列表中移除该墙，并使得两边打通；若都已经访问过，则可以直接移除该墙面。

在实际的算法实现过程中，采用了优化策略，将对墙的维护改成对于单元格四周的维护，即在已经能走通的单元格随机选择一面打通。随机 Prim 算法生成的墙面足够曲折和复杂，能够较好的满足游戏设计中随机生成迷宫的需求。

```
71      def prim_produce(self, no_wall, maps_vis, maps_content):
72          vis_lis = [(0, 0)]  # 存储待处理的单元格
73          while vis_lis:  # Prim随机生成迷宫
74              x, y = random.choice(vis_lis)
75              maps_vis[x][y] = 1
76              vis_lis.remove((x, y))
77              check = []
78              for i in range(4):  # 检测四个方向是否相通
79                  xx = x + dx[i]
80                  yy = y + dy[i]
81                  if xx < 0 or yy < 0 or xx >= self.W_1 or yy >= self.H_1:
82                      continue
83                  if maps_vis[xx][yy] == 1:
84                      check.append(i)
85                  elif maps_vis[xx][yy] == 0:
86                      vis_lis.append((xx, yy))
87                      maps_vis[xx][yy] = 2
88              if len(check):
89                  k = random.choice(check)
90                  no_wall[x][y][k] = 1
91                  k_ = -1
92                  if k == 0 or k == 2:
93                      k_ = k + 1
94                  else:
95                      k_ = k - 1
96                  no_wall[x + dx[k]][y + dy[k]][k_] = 1
97          for i in range(self.W_1):  # 将原本只有单元格的数组转变成包含墙的数组
98              for j in range(self.H_1):
99                  cell_data = no_wall[i][j]
100                 maps_content[i * 2 + 1][j * 2 + 1] = 1
101                 x = i * 2 + 1
102                 y = j * 2 + 1
103                 for k in range(4):
104                     if cell_data[k] == 1:
105                         maps_content[x + dx[k]][y + dy[k]] = 1
106     return maps_content
```

2.2 代码结构：

init 函数初始化加载规则、人物和遮挡画布的 Actor 类，并将游戏内部的 mode 置为 1，表示新建游戏后先进入规则界面。外部创建 MAZE 类的实例后，先调用其成员函数 main_maze 对人物的初始状态、游戏的开始时间、游戏积分进行初始化，并在 main_maze 中进一步调用 prim_produce 成员函数用随机 Prim 算法生成迷宫，存储在 numpy 数组中，之后再随机生成金币放入数组。

在外部函数 draw 不断调用 draw_maze 成员函数时，根据 mode 的值确定生成规则界面还是游戏界面，其中游戏界面除人物之外的场景在没有切换 1/4 迷宫区域的情况下不会重新对地图数组（Actor 类的数组）赋值，这一判断在 show_ground 函数中实现，节省了时间开销；而外部的 update 函数持续调用 play_maze 的成员函数来感应键盘的操作并判断操作的合法性，进而对人物的行动产生预判。人物之后主要由 move_player 函数进行控制并通过 animation 函数判断微小位移的次数，来实现画面的人物的动画效果。

游戏最终的胜负在 draw_maze 中判断，并调用 win 函数和 lose 函数来播放对应的音效，游戏的通关记录也将在 win 函数中进行更新。

3. MARIO_VS_LAKITU 类(简称 MVL)：

在生成 MVL 类对象后调用 init 函数初始化游戏规则界面，在点击 START 按钮后进入游戏，并调用 main_MVL 函数对成员变量进行赋值，其中包括但不限于金币、刺猬云、板栗仔、人物的生成及初始属性的设定。

后续运行过程中，外部的 draw 函数不断调用 draw_MVL 成员函数来显示或消除部分画面元素，并进行人物存货状态的判定。

```
def draw_MVL(self):
    if self.mode == 1:
        screen.clear()
        screen.blit("background", (0, 0))
        self.rule_MVL.pos = (WIDTH / 2 - 60, HEIGHT / 2 - 5)
        self.rule_MVL.draw()
        self.start_button.pos = (570, 420)
        self.start_button.draw()
        Pause_reminder.draw()
    elif self.mode == 2:
        global Game_Over, end_time
        screen.blit("background", (0, 0))
        screen.draw.text("TIME:%d" % (self.TIME / 60,), (10, 10), color="blue")
        screen.draw.text("COIN:%d" % (self.COIN,), (560, 10), color="blue")
        self.ground.draw()
        self.Lakitu.draw()
        if self.Mario.flag4 != 2:
            self.Mario.draw()
        for i in range(max(self.Goomba_num-10,0),self.Goomba_num):
            if self.Goomba[i].flag > 0:
                self.Goomba[i].draw()
        for i in range(max(self.spiny_num-10,0),self.spiny_num):
            if self.spiny[i].flag > 0:
                self.spiny[i].draw()
        for i in range(max(self.coin_num-10,0),self.coin_num):
            if self.coin[i].flag > 0:
                self.coin[i].draw()
        for i in range(max(self.mushroom_num-10,0),self.mushroom_num):
            if self.mushroom[i].flag > 0:
                self.mushroom[i].draw()
        if self.death.flag > 0:
            self.death.draw()
            if not Game_Over:
                Game_Over = True
                end_time = time.time()
```

外部的 update 函数则在调用 update_MVL 成员函数的过程中定时生成金币、板栗仔等对象、响应键盘操作和调用 pass0 进行游戏过关的判定。

```
def update_MVL(self):
    if self.mode == 1:
        return
    if not flag_pause and self.paused:
        MVL_game.paused = False
        clock.schedule_unique(self.Lakitu.Lakitu2, self.time)
        clock.schedule_unique(self.coin_create, 1.5)

    self.Lakitu.update()
    for i in range(max(self.Goomba_num - 10, 0), self.Goomba_num):
        if self.Goomba[i].flag > 0:
            self.Goomba[i].update()
    for i in range(max(self.spiny_num - 10, 0), self.spiny_num):
        if self.spiny[i].flag > 0:
            self.spiny[i].update()
    for i in range(max(self.coin_num - 10, 0), self.coin_num):
        if self.coin[i].flag > 0:
            self.coin[i].update()
    for i in range(max(self.mushroom_num - 10, 0), self.mushroom_num):
        if self.mushroom[i].flag > 0:
            self.mushroom[i].update()
    self.key()
    self.Mario.update()
    if self.death.flag > 0:
        self.death.update()
    self.mushroom_create()
    self.mushroom_time += 1
    if self.TIME > 0:
        self.TIME -= 1
    self.pass0()
```

其中，金币等对象由对应的内部类生成，分别为：_Goomba, _coin, _spiny, _Lakitu, _Mario, _mushroom 类，以及在人物死亡后出现的_death 类。这些类在各自的初始化中实现类对象属性的初始赋值，并通过函数互相调用的方式实现动画（下图为 coin 金币动画的实现方式，其他类对象是类似的）

```
1109        # 金币动画效果
1110        def coin1(self):
1111            self.image = "coin1"
1112            clock.schedule_unique(self.coin2, 0.1)
1113
1114        def coin2(self):
1115            self.image = "coin2"
1116            clock.schedule_unique(self.coin3, 0.1)
1117
1118        def coin3(self):
1119            self.image = "coin3"
1120            clock.schedule_unique(self.coin1, 0.1)
```

除上述内容外，MVL 类中还有的成员函数有：coin_create 控制金币的随机生成和初始属性；mushroom_create 控制蘑菇的定时生成；key 函数为 update_MVL 所调用来实现键盘的响应；pass0 函数进行成功过关的判定。

4. BOSS 类：

BOSS 类的代码结构与 MVL 类基本一致，也是在初始化后生成各内部类的对象，通过内部类的初始化和函数调用实现包括移动和动画在内的对画面元素的控制。

外部的 draw / update 函数通过调用 draw_boss / update_boss 成员函数实现画面的不断更新。

5. ImitateMusic 类:

游戏运行时，根据玩家得分实时更新物品下落间隔、密度与速度，在 update_music 函数中随机生成箭头方块或道具。

```
1937    def update_music(self):
1938        if self.mode == 1:
1939            return
1940        self.s += 1  # 更新时间
1941        # 随分数调整固定间隔
1942        d1 = 60 - self.score // 200
1943        if d1 < 10:
1944            d1 = 10
1945        # 随分数调整随机间隔
1946        d2 = 60 - self.score // 300
1947        if d2 < 20:
1948            d2 = 20
1949        # 随机生成新项目
1950        if self.s > d1 and random.randrange(d2) == 0 and self.lose == False:
1951            name = "unknown"
1952            if random.randrange(10) != 0:
1953                name = random.choice(self.dires)
1954            t = Actor(name, center=(random.randrange(16, 624), 10))
1955            t.name = name
1956            self.items.append(t)
1957            self.s = 0
```

随后通过 on_key_down_music 函数捕捉键盘响应，根据最低物品的状态判定响应正误，并更新拾取标记 get，连击数 combo，玩家得分 score 与拾取等级 condition，更新后交由得分处理函数 deal 统一处理。

```
1807    def deal(self):
1808        # 判断是否属于有效判定距离
1809        if 480 - self.items[0].bottom <= self.distance:
1810            self.get = True  # 更改拾取标记
1811            self.combo += 1  # 更新连击数
1812            if self.combo > 2:
1813                self.score += self.combo * 10
1814            if 480 - self.items[0].top <= self.distance / 3:  # Perfect
1815                self.score += 100
1816                self.condition = 5
1817            elif 480 - self.items[0].top <= self.distance - 10:  # Great
1818                self.score += 85
1819                self.condition = 4
1820            elif 480 - self.items[0].bottom <= self.distance / 3:  # Good
1821                self.score += 60
1822                self.condition = 3
1823            else:  # Normal
1824                self.score += 30
1825                self.condition = 2
1826        else:  # Miss
1827            self.get = False  # 更改拾取标记
1828            self.combo = 0  # 连击数归零
1829            self.score -= 100
1830            self.condition = 1
```

最后由 draw_music 函数根据各参数状态显示相应图像与文本。同时，拾取道具触发的效果由计时器定时调用 set_speed_normal 与 set_distance_normal 实现下落速度与判定距离的动态变化。

6. BattleShip 类:

游戏运行时，首先调用 generate_ships 函数，利用随机枚举加循环的方式实现敌舰的随机生成。

算法主要是利用循环随机选择方向优先填充海面，直至达到敌舰长度，再生成下一艘敌舰，否则更换方向重新填充，实现海面上随机生成预定长度敌舰的效果。

7. STORE 类：

STORE 类实现的主要功能是: 1、标记不断转动，显示抽卡的位置 2、点击抽卡键后，抽卡并且将抽到的图像闪烁并放在界面正中 3、对全局变量金币（coins）的处理

```
2178        if self.choose.move_flag == True:
2179            if self.choose.showtime < 20:
2180                self.choose.showtime += 1
2181            else:
2182                self.choose.showtime = 0
2183                self.choose.pos_i = (self.choose.pos_i + 1) % 14
2184                self.choose.pos = self.choose_card[self.choose.pos_i]
```

Choose 进行标记，当没有抽卡时（move_flag==True），标记正常转动，一共有 14 个位置，利用数组 choose.pos 记录位置，choose.pos_i 记录序号，利用数组的序号不断+1%14。为了使标记移动后有短暂停留，采用了 showtime 来记录停留时间，进行判断：如果停留时间超过 20/60s 那么就移动到下一个位置。

```
2166            if self.tick.flag == True:
2167                self.tick.draw()
2168                if self.prize.flag:
2169                    self.prize.image = self.prizes_bigger[self.choose.pos_i]
2170                    self.prize.pos = (320, 240)
2171                    self.prize.draw()
2172                    if self.prize.showtime < 70:
2173                        self.prize.showtime += 1
2174                        clock.schedule(self.hide_prize, 0.5)
2175                        clock.schedule(self.show_prize, 1)
2176                    else:
2177                        self.prize.flag = True
```

先是根据 choose.pos_i 来获取抽奖结果，并且用与前面类似的方式控制图像的显示时间，clock 函数实现了闪烁的效果

```
2214        if self.roll_button.collidepoint(pos) and Coins >= 50:  # 抽卡
2215            sounds.card.play()
2216            self.pressed = True
2217            x = random.randint(1, 15)
2218            Coins -= 50
2219            self.choose.move_flag = False
2220            self.tick.flag = False
2221            for i in range(1, x + 20):
2222                clock.schedule(self.move_choose, i / 4)
2223            clock.schedule(self.move_choose, x / 4 + 5)
2224            clock.schedule(self.move_choose, x / 4 + 6)
2225            clock.schedule(self.move_choose, x / 4 + 7.5)
2226            clock.schedule(self.move_choose, x / 4 + 9.5)
2227            clock.schedule(self.set_tick, x / 4 + 9.7)
2228            clock.schedule(self.show_prize, x / 4 + 9.8)
```

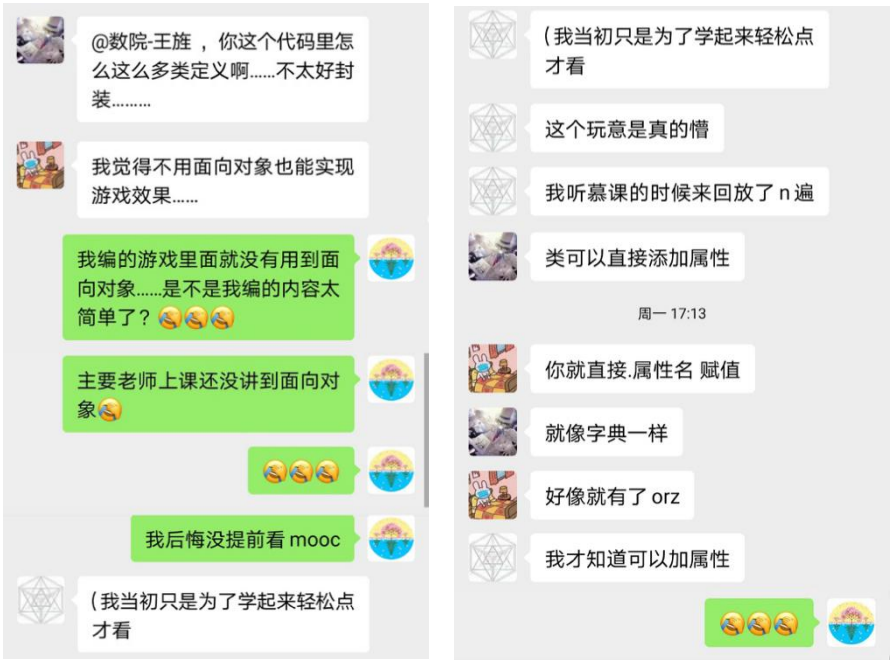
抽卡函数，当点击抽卡按钮且金币>=50 的时候可以抽卡，将抽卡状态记录下来（move_flag=False），使得显示标记进入抽卡状态的移动用多个 clock 函数实现标记逐渐减速，最终停下的动画效果，同时在函数中记录下抽奖的结果

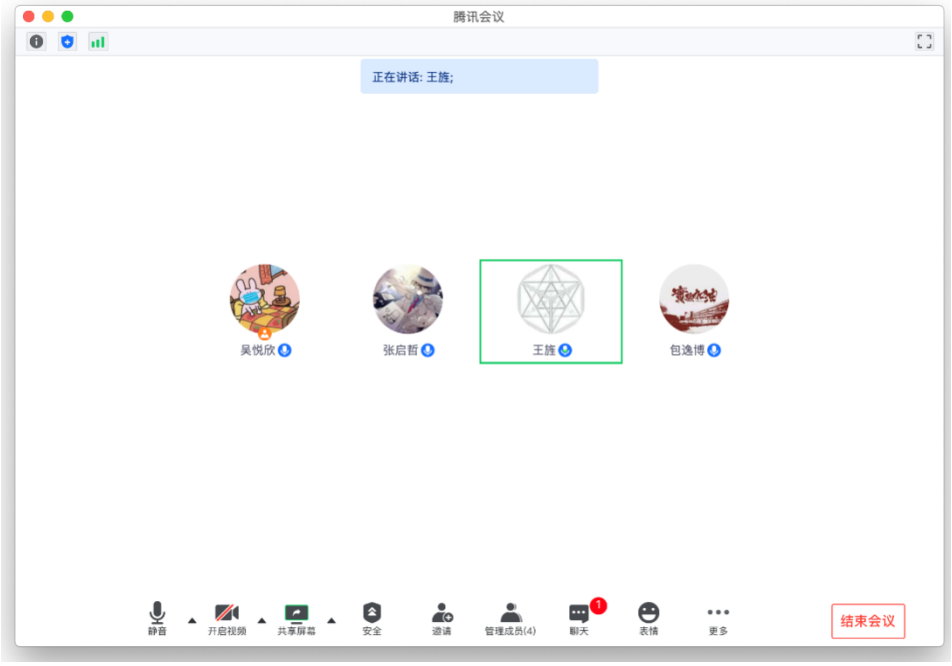
四、后续工作展望：

该项目后续仍有许多值得扩展改进的方面：内容上，可以增加更多类型的游戏关卡来丰富游戏体验；功能上，可以扩展商店的内容和更多由玩家自定义的部分，如音效与音乐的控制开关、游戏的难度选择等，提升玩家对于整个游戏的掌控性；画面上，可以增添更多的动画效果，增添转场动画、通关动画等，提高游戏的观赏性。

五、小组分工合作(含讨论照片)：

组长吴悦欣主要负责外部框架与主界面功能设计、迷宫关卡编写、代码改进、部分素材制作、视频剪辑与项目规划；王旌同学主要负责项目图片与音频素材提供、MVL 与 boss 关卡编写，项目海报制作与设计方案的提出；张启哲同学主要负责外部框架设计、两个奖励关卡编写、部分素材制作与代码改进；包逸博同学主要负责商店功能编写、部分素材制作、游戏介绍和帮助撰写与视频剪辑；实习报告的撰写所有成员均有参与。





六、致谢：

感谢 Python 语言基础与应用课程中陈斌老师与各位助教的讲解与帮助！