哈尔滨工业大学（威海）

“校长杯”学生学术科技作品竞赛

项目论文

大富翁游戏框架

——基于 Python 及 Pygame

第一作者姓名及学号：董成相 xxxxxxxxx

第一作者所在学院：计算机科学与技术学院

团队其他成员姓名：丁文琪、初征、付宽

填 写 日 期 ：2019 年 11 月 13 日

摘要

近年来，电子游戏正成为许多人们精神生活的很大一部分。我们通过对 Python 及 Pygame 的学习，并应用面向对象的思想设计并实现了大富翁游戏框架。

关键字：大富翁，电子游戏，Python，Pygame

Abstract

In recent years, video games are becoming a big part of the spiritual life of many people. By learning Python and Pygame, we used the Oriented-Object principle to design and implement the Monopoly's game framework.

Keywords

Monopoly, Video Game, Python, Pygame

目录

第一章 项目简介

1.1 项目背景

1.1.1 背景一

有人说二十一世纪是计算的时代，伴随着互联网的发展以及计算机的亲民化，电子游戏越来越为人们所接受，这其中有小游戏、大型网络游戏、也有中年人爱玩的纸牌游戏。无论哪种，电子游戏都极大的满足了我们的感官，给我们的生活带来了极大的乐趣。

1.1.2 背景二

随着电竞等行业的日益发展，电子游戏正在也必将成为未来的一个热点。因此我们认为学习如何实现一个小游戏是极具现实意义的，这不仅是对我们能力的一次检验，更是对我们能力的一次挑战。

1.2 项目制作的目的与意义

1. 在完成项目的过程中学习一种新的编程语言，增强我们小组成员的个人能力，培养小组成员的团队协作能力，为我们将来在软件开发这条路上走的更远助力；
2. 培养小组成员将一个项目具体实现的能力，为我们将来完成其他的项目打基础，积累语言的使用经验；培养艰苦创业，脚踏实地的实干精神；
3. 通过这个项目计划的实现，增强我们的自信心，为我们的语言学习增加源动力；
4. 通过这个计划切实体会游戏开发的过程，对游戏开发有一个明确的概念，也对软件工程有一个更深入的理解；
5. 培养我们以计算机科学家的角度思考问题的能力。

1.3 项目创新点

1. 装备升级：交通工具的升级；
2. 炸药：获得炸药时可以炸毁敌人的房子；
3. 资金互换：与敌人进行资金互换，增添了游戏的不确定性；
4. 开发者模式： NPC 的信息默认不显示，增添了游戏的不确定性；

第二章 方案选择与论证

2.1 方案一：使用 C 语言

优点：执行效率高，可延展功能多。此外，我们团队成员作为软件工程专业的学生，C 语言是我们接触及学习的第一门语言，也是使用最熟悉的语言，不需付出较多的额外时间去学习即可直接进行编程。

缺点：C 语言相比其他语言来说不适合编写游戏，将数据图形化表示需要较高的 C 语言水平并且相对以下两种方案来说更加不方便，而且我们目前还没有达到这一水平，想要编写具备图形界面的游戏，就需要更深入地学习，反而会抵消已知的优点。

2.2 方案二：使用 Python 语言

优点：我们已经具备了 C 语言的基础，学习 Python 的速度相对来说会快很多。掌握一门新的语言对于未来的学习与工作都有很多好处。Python 语法灵活，结构精简，代码量小，使用 python 编写游戏的图形界面比使用 C 语言编写简单方便很多。

缺点：需要付出较多的额外时间去学习一种新的语言。

2.3 方案三：使用 Python 语言并结合 Pygame 库

优点：Pygame 库是一个开源的游戏开发框架，其中自带图像与声音处理功能，游戏开发界面采用图形化表示，易理解、易上手、可玩性高。同时，Pygame 建立在 SDL 的基础上，允许实时电子游戏研发而不被低级语言束缚。

缺点：需要付出一定的时间去学习 Pygame，以及直接应用 Pygame 库会有比较高的侵入性，需要我们进行进一步的封装。

2.4 方案选择

经过上述论证并结合我们自身的实际情况及实际开发中的代码复用原则，我们最终决定使用方案三。方案三的技术难度适中，我们既可以达到学习新语言的目的，又可以达到切实体会游戏开发的目的，不至于将精力分散到太过底层的东西。

第三章 基础类设计与实现

3.1 角色设计

3.1.1 PC（Player Control）类

PC 类是由玩家操控的角色，用于维护玩家的状态：name、position、money、houseCounter、transportation、status、engine 等。如图 3-1 所示，PC 类拥有：move 方法，用于 PC 的坐标值的计算；swift_horse_move 方法，用于千里马移动的功能实现；incidents 方法用于处理 PC 遭遇的事件；messages 方法基于__base_messages 方法和__incidents_messages 方法生成相应事件的提示信息；buy 方法基于__buy_land 方法和__buy_horse 方法进行购买操作。

```
class PC:
    def __init__(self, name):...

    def move(self):...

    def swift_horse_move(self, forward):...

    def incidents(self, all_lands):...

    def messages(self, all_lands):...

    def __base_messages(self):...

    def __incidents_messages(self, all_lands):...

    def buy(self, land):...

    def __buy_land(self, land):...

    def __buy_horse(self):...
```

图 3-1 PC 类

3.1.2 NPC（Non-Player Control）类

NPC 类是由电脑操控的玩家，与 PC 类较为相似，具备一定的自我判断能力。如图 3-2 所示，NPC 类与 PC 类相比多了一个__money_left_line 方法，该方法用于限制 NPC 的金钱余额，确保其不会轻易破产；其他与 PC 中同名的方法，与 PC 中方法的功能相同，但是因 NPC 是有电脑操控的，所以内部实现的细节与 PC 会有不同。例如__buy_land 方法，如图 3-3 是 PC 的实现，如图 3-4 是 NPC 的实现，可以发现 NPC 的实现多出了一系列的判断条件。

```
class NPC:
    def __init__(self, name):...

    def move(self):...

    def swift_horse_move(self, all_lands, land_is_full):...

    def incidents(self, all_lands):...

    def messages(self, all_lands):...

    def __base_messages(self):...

    def __incidents_messages(self, all_lands):...

    def buy(self, land, land_is_full):...

    def __buy_land(self, land, land_is_full):...

    def __buy_horse(self):...

    def __money_left_line(self):...
```

图 3-2　NPC 类

```
def __buy_land(self, land):
    price = land.price(self.name)
    if price != 0:
        self.money -= price if self.free is False else 0
        self.free = not self.free if self.free is True else self.free
        land.change_property(self.name)
        self.houseCounter[land.level - 1] += 1
```

图 3-3　PC 的__buy_land 方法

```
def __buy_land(self, land, land_is_full):
    price = land.price(self.name)
    level = land.level
    if level == 0 or self.free or \
       (level == 1 and self.houseCounter[0] > 13) or \
       (level == 2 and self.houseCounter[1] > 10) or \
       (level == 3 and self.houseCounter[2] > 7) \
       or level >= 4 or land_is_full is True or self.money > 2500:
        self.money -= price if self.free is False else 0
        self.free = not self.free if self.free is True else self.free
        land.change_property(self.name)
        self.houseCounter[land.level - 1] += 1
```

图 3-4　NPC 的__buy_land 方法

3.1.3 PlayerTurn 类

PlayerTurn 类是一个枚举类，用于标记玩家的回合状态，如图 3-5 所示。

```
class PlayerTurn(Enum):
    start = 0        # 游戏开始
    PCMove = 1       # PC移动
    NPCMove = 2      # NPC移动
    PCAct = 3        # PC行动
    NPCAct = 4       # NPC行动
```

图 3-5 PlayerTurn 类

3.2 地图设计

3.2.1 OneLand 类

OneLand 类用于维护单个地块的状态：owner、position、level、incident。地图中共有三类地块：起点地块 1 个、事件地块 10 个、空地地块 39 个，其中只有空地地块可以被玩家购买。如图 3-6 所示，OneLand 类拥有：price 方法，用于获取地块价值及对于不同的角色来说该地块能否被购买；change_property 方法，用于变更地块所有者，实现地块的购买；bang 方法，用于炸毁地块。

```
class OneLand:
    def __init__(self, position, owner="系统", incident=Incidents.houseFiled):...

    def price(self, who):...

    def change_property(self, who):...

    def bang(self):...
```

图 3-6 OneLand 类

3.2.2 Landmasses 类

Landmasses 类是一个地块集，用于维护地图中所有的 50 个地块，两个角色的姓名以及玩家是否已获得地块被买满时的奖励和给予奖励的提示信息是否已展示到信息面板。如图 3-7 所示，Landmasses 类拥有：is_full 方法，用于获取地图中的所有地块是否已被买满，以便进行资金奖励。

```
class Landmasses:
    def __init__(self, PCName, NPCName):...

    def is_full(self, name):...
```

图 3-7 Landmasses 类

3.2.3 Incidents 类

Incidents 类是一个枚举类，用于标记地块上的事件类型，如图 3-8 所示。

```
class Incidents(Enum):
    start = 0                       # 起点
    horseField = 1                  # 马场
    encounterThief = 2              # 遭遇小偷
    involvedMurder = 3              # 卷入谋杀案
    changeMoney = 4                 # 资金互换
    explosive = 5                   # 炸药
    reachGoldMine = 6               # 到达金矿
    haveACold = 7                   # 感冒
    strongWind = 8                  # 遭遇大风
    friendshipWithLord = 9          # 与当地领主结好
    houseFiled = 10                 # 空地块
    house = 11                      # 房间
```

图 3-8　Incidents 类

3.3 交互设计

3.3.1 ShootDice 类

ShootDice 类维护 randomSeries 以及 finalPoints，用于为掷骰子做数据准备工作。如图 3-9，ShootDice 类拥有：__set_final_points 方法，用于设置最终展示的骰子的点数；__get_random_series 方法，用于获得一个随机数列，形成掷骰子的效果；set_dice 方法，用于基于从玩家获得的数据进行生成随机数列以及设置 finalPoints。

```
class ShootDice:
    def __init__(self):...

    def __set_final_points(self, points):...

    def __get_random_series(self):...

    def set_dice(self, points):...
```

图 3-9　ShootDice 类

3.3.2 MusicPlay 类

MusicPlay 类维护 music 及 isPlaying，用于控制背景音乐的播放状态。如图 3-10 所示，MusicPlay 类拥有一个 pause 方法，用于切换音乐播放状态。

```
class MusicPlay:
    def __init__(self, music):...

    def pause(self, pos):...
```

图 3-10　MusicPlay 类

3.3.3 GameManager 类

GameManager 类即时我们在前面提到的所谓的“进一步的封装”，其通过 Pygame 库[1][2]提供的一系列功能管理游戏所用到的所有素材[3]以及这些素材的正确展示。

```
class GameManager:
    def __init__(self):...

    def event_deal(self):...

    def post_space_key_down(self):...

    def get_character_name(self, PC_name, NPC_name):...

    @staticmethod
    def image_update():...

    def __image_load(self, name):...

    def draw_beginning(self):...

    def draw_map(self):...

    @staticmethod
    def __location_convert(position):...

    def draw_character(self, PC_pos, NPC_pos):...

    def draw_active_status(self):...

    def draw_lands(self, all_lands):...

    def draw_tips(self):...

    def __developer_pattern_check(self):...

    def __set_developer_pattern(self, event_key):...

    def __set_PC_act_key(self, event):...
```

图 3-11　GameManager 类第一部分

如图 3-11 及图 3-12 所示，Game Manager 类拥有一系列的方法，其中：event_deal 也就是前面提到的封装，它承担了事件处理的任务，将其从游戏的主循环中剥离了出来，如图 3-13 所示，现在在主循环中仅仅是一条语句，但如图 3-14 所示，在封装之前事件处理与游戏过程是混杂在一起的；以 draw 开头的一系列方法都是用来绘制相应的图案的；方法名内包含 developer 的是开发者模式相关的功能；__location_convert 方法是重要的坐标转换方法，它将角色以及单个地块的坐标转换为屏幕上的像素坐标；方法名包

含 music 的方法用于处理背景音乐播放的图像展示；方法名包含 dice 的方法用于处理掷骰子的图象展示；方法名包含 turn 的两个方法用于控制角色及游戏的回合切换，用于确保游戏正常进行以及在游戏进行过程中随时重新开始一轮游戏提供保障；

余下方法的功能可以通过方法名字很容易地知道。

```
def draw_messages(self, messages):...
def __music_pause(self, mouse_pos):...
def draw_music_button(self):...
def __draw_music_button_change(self, is_playing):...
def __set_dice_location(self):...
def draw_dice(self, final_points, random_series):...
def draw_fix_dice(self, final_points):...
def turn_change(self):...
def turn_change_space(self):...
def game_over_check(self, PC_money, NPC_money):...
def draw_game_over(self):...
def __play_again(self):...
def __quit(self):...
```

图 3-12　GameManager 类第二部分

```
while gameManager.gameStatus is not GameStatus.quit:
    gameManager.event_deal()

    if gameManager.gameStatus is GameStatus.start:...

    if gameManager.gameStatus is GameStatus.initial:...

    if gameManager.gameStatus is GameStatus.playing:...

    if gameManager.gameStatus is GameStatus.over:
        gameManager.draw_game_over()
```

图 3-13　封装后事件处理

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.KEYDOWN:
        # 开发者模式
        if enemy.check() is False:...
        # 掷骰子
        if heroTurn is True:
            if event.key == K_SPACE:...
        # 购买地块、交换金钱、使用炸药
        if event.key == K_b:...
        # 退出游戏
        elif event.key == K_e:...
        # 引擎移动
        if hero.transportation == "千里马":
            if event.key == K_a:...
            elif event.key == K_d:...
```

图 3-14　封装前事件处理

GameManager 类正像它的名字一样维护着游戏的进行，将素材的载入封装进了它的构造方法中，简化了游戏主循环的臃肿。如图 3-16 所示，封装过之后的游戏主循环的结构非常清晰；如图 3-15 所示，相比之下，封装前进入游戏主循环之前需要做大量准备，当然这些准备工作都是必须的，但这些准备不应当是显式地出现在游戏的主体结构中的，而且截图所展示的仅仅是一部分准备工作，在图 3-15 所示之后仍然有很多准备工作，臃肿程度可见一斑。

```
# pygame初始化
pygame.init()

# 窗口初始化
pygame.display.init()
screen = pygame.display.set_mode((1200, 825))
pygame.display.set_icon(pygame.image.load("./source/dog.ico").convert_alpha())
pygame.display.set_caption("大富翁")

# 载入图片
start = pygame.image.load("./source/start.png").convert_alpha()
gameOver = pygame.image.load("./source/gameover.png").convert_alpha()
gameWin = pygame.image.load("./source/gamewin.png").convert_alpha()
gameMap = pygame.image.load("./source/map.png").convert_alpha()
barrier = pygame.image.load("./source/barrier.png").convert_alpha()
PCImage = pygame.image.load("./source/PC.png").convert_alpha()
PCFixImage = pygame.image.load("./source/PCBeginning.png").convert_alpha()
NPCImage = pygame.image.load("./source/NPC.png").convert_alpha()
NPCFixImage = pygame.image.load("./source/NPCBeginning.png").convert_alpha()
```

图 3-15　封装前游戏主循环前代码

```
if __name__ == "__main__":
    gameManager = GameManager()
    hero, enemy, landmasses, shootDice = None, None, None, None
    while gameManager.gameStatus is not GameStatus.quit:
        gameManager.event_deal()

        if gameManager.gameStatus is GameStatus.start:...

        if gameManager.gameStatus is GameStatus.initial:...

        if gameManager.gameStatus is GameStatus.playing:...

        if gameManager.gameStatus is GameStatus.over:
            gameManager.draw_game_over()
```

图 3-16　封装后游戏主循环前代码

GameManager 类还有一项功能，它将游戏所用到的素材与游戏的数据分隔开了。尽管由相应的对象管理相应的素材这是一个很自然的想法，也更便于我们理解，但这并不是一个好的设计，某种程度上违反了单一职责原则。因此将数据与素材的展示分隔开就显得很必要了，这样一来游戏的数据层面便可专注于数据的运算，而不用关心游戏素材何时展示到屏幕、展示到屏幕的什么位置。

值得一提的还有 GameStatus 这个类，它也是一个枚举类，用于标记游戏阶段，如图 3-17 所示。

```
class GameStatus(Enum):
    start = 0       # 游戏开始界面
    waitIn = 1      # 等待进入游戏
    initial = 2     # 实例化角色对象、地块对象、掷骰子对象
    playing = 3     # 进入游戏
    over = 4        # 游戏结束界面
    end = 5         # 游戏结束界面绘制完毕
    quit = 6        # 退出游戏
```

图 3-17　GameStatus 类

第四章 后期规划

当前的框架已基本能满足进行大富翁游戏的功能，当然还有很多功能是我们与成熟的大富翁框架相比所欠缺的，在我们团队成员时间较宽裕的情况下，这些功能可以在后期经过调研逐步添加到我们的框架中，不管怎么说，值得期待。

结论

本文设计了一个基于 Python 及 Pygame 的大富翁游戏框架，尽管与成熟的大富翁有不小的差距，但我们相信对于在该项目进行时我们对相应的知识了解的比较少的情况下设计并实现该框架的挑战性是挺高的，对于我们来说也具有一定的研究价值。

参考文献

[1] David Beazley & Brian K. Jones. Python Cookbook[M]. 北京：人民邮电出版社，2015：1-679

[2]Pygame Documentation[OL].https://www.pygame.org/docs/

[3]爱给——专注免费素材[OL]. http://www.aigei.com