上海大学

SHANGHAI UNIVERSITY

《Python 计算》期末综合报告

题　　目　可视化麻将游戏

Mahjong

学　　号　**20123004**

姓　　名　钟浩文

日　　期　**2022 年 5 月 25 日**

一、实验目的与要求

本次实验希望能够借助 pygame 库设计一款用户友好的可视化麻将游戏。实验目的旨在熟悉 pygame 库中功能函数以及希望能够体验设计一款游戏的过程。

游戏规则简单介绍：游戏中采取日麻无赤宝牌规则。共计 136 张麻将牌，其中包含 1-9 万/条/筒与东南西北中发白各 4 张（无花牌）。基本规则与国标麻将一致，区别主要在于：每轮对局中存在“宝牌”，胡牌时手牌中若持有宝牌则会有对应奖励；日麻胡牌需要满足特定条件，即为“役”，无役无法胡牌。

本次实验中需要能够实现麻将中基本的吃、碰、杠，摸牌切牌理牌功能，同时对于玩家的手牌能够实时判胡。借助 pygame 库在游戏窗口中实现上述所有功能。

二、实验环境

Windows11 操作系统，使用 PyCharm 开发环境与 Python3.9 编译器。

实验过程中主要使用了 pygame 库实现，其功能如下：

可视化麻将游戏 Mahjong 中所使用的所有 pygame 功能函数解析	
函数名	函数功能
pygame.display 访问显示设备	
pygame.display.init	初始化显示模块
pygame.display.quit	取消初始化显示模块
pygame.display.set_mode	初始化窗口或屏幕以供显示
pygame.display.flip	将完整显示面更新到屏幕上
pygame.display.set_caption	设置当前窗口标题
pygame.surface 管理图像和屏幕	
pygame.Surface.blit	把一个图像画到另一个图像上
pygame.Surface.convert	更改图像的像素格式
pygame.Surface.fill	用固体颜色填充表面
pygame.rect 管理矩形区域	
pygame.Rect.clip	在另一个内种植一个长方形
pygame.event 管理事件	
pygame.event.get	从队列中获取事件
pygame.event.wait	等待队列中的单个事件
event 事件类型	
QUIT	用户按下窗口的关闭按钮
KEYDOWN	键盘按下

MOUSEBUTTONDOWN	鼠标按下
pygame.font 用于加载和呈现字体	
pygame.font.init	初始化字体模块
pygame.font.SysFont	从系统字体创建字体对象
pygame.font.Font	从文件中创建新的 Font 对象
pygame.image 加载和存储图片	
pygame.image.load	从文件中加载新图像
pygame.key 读取键盘按键	
pygame.key.get_focused	如果显示正在接收来自系统的键盘输入，则为 true
pygame.mouse 鼠标	
pygame.mouse.get_pos	获取鼠标光标的位置

三、实验内容

我对于可视化麻将游戏 Mahjong 的总体设计可参考下图：

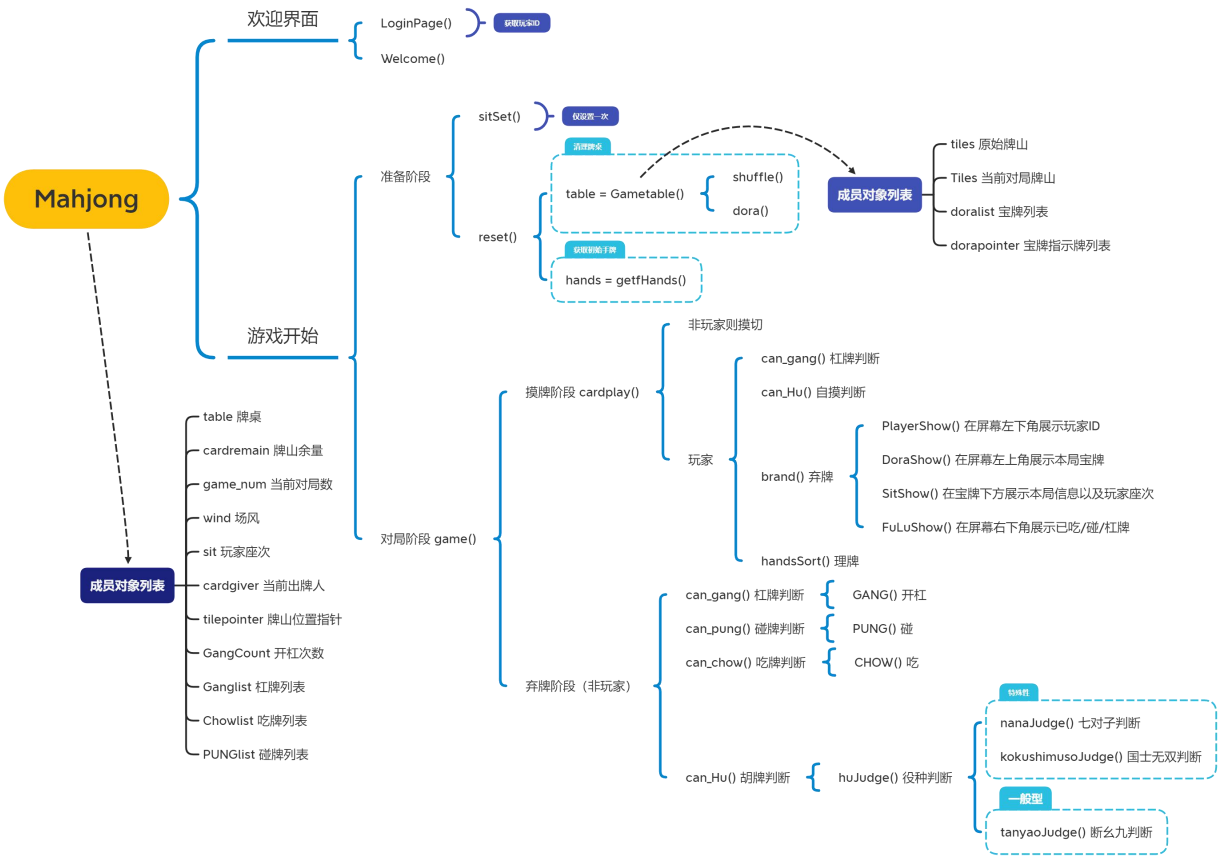


图 1 总体设计图

玩家进入游戏后，首先会来到登录界面，登录界面中有我为这个游戏所设计

的一个 Logo。此时玩家需要在 ID 栏中输入自己的 ID，并选择登录。登录后会来到欢迎界面，并提示玩家可以选择开始游戏。

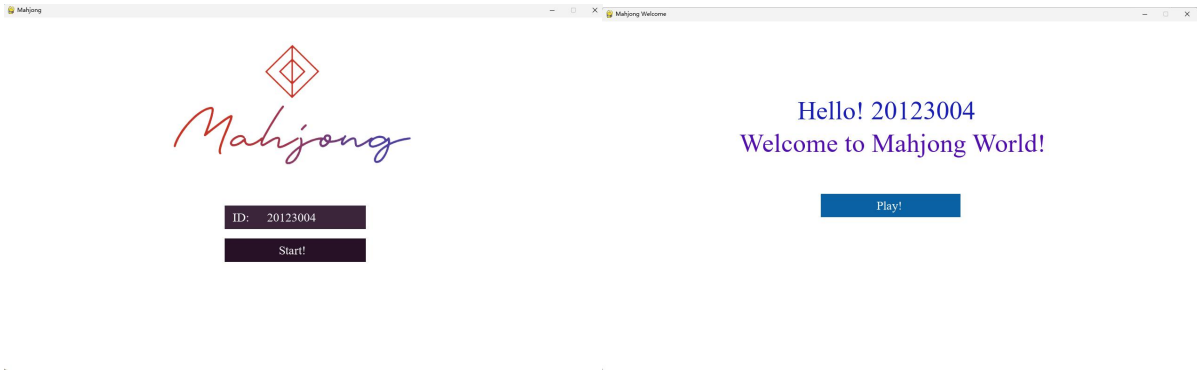


图 2、3 用户登录、欢迎界面

开始游戏后，游戏后台会自动进入准备状态，清理牌桌并生成此次对局的牌山，并为玩家分发初始手牌。玩家可以在界面中观察到如下信息：对局场次，玩家座次，玩家 ID，本局宝牌指示牌以及玩家手牌。管理员可以在控制台观测到本局的牌山以及宝牌情况，具体可以参考下图。

图 4 游戏界面信息

准备阶段结束后正式开始游戏。当一轮游戏开始时，玩家摸牌阶段，系统会自动为玩家分发此轮所摸到的牌。完成摸牌后，进入舍牌阶段。玩家可以使用鼠标点击自己手牌中的任意一张牌，并打出这张牌。系统会自动检测玩家鼠标位置并打出对应手牌。出牌后，系统会自动帮助玩家按照牌序进行理牌。

在非玩家出牌轮次中，系统会检测用户是否可以鸣牌（吃/碰/杠），当满足鸣牌条件时，系统会提示用户选择是否鸣牌。玩家鸣牌后，鸣牌手牌将不再出现在手牌中，而是会以组合形式摆放在屏幕右下角，表示此为副露牌，如图 5 所示。

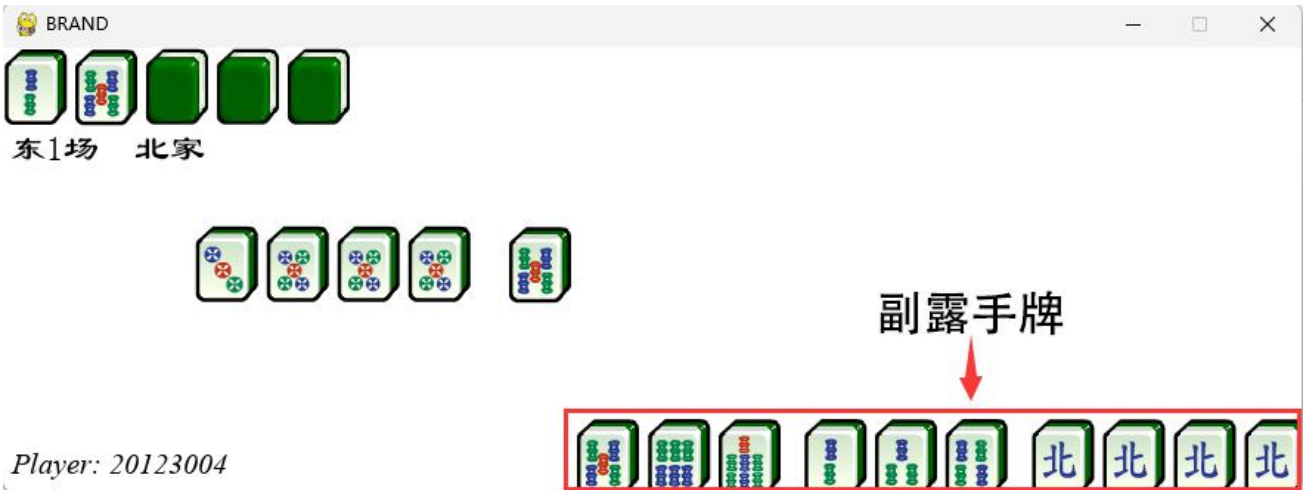


图 5 副露手牌显示

游戏后台会实时检测用户手牌发生变化时是否可以胡牌，一旦别家打出铳牌/自己自摸时，会提示用户可以胡牌。胡牌后系统会显示胡牌家手牌。

四、实验设计与实现

这一部分我主要想讲讲 Mahjong 图形化界面的设计与实现思路，以及吃碰杠和判胡算法的整体思路。

首先，我将会介绍图形化界面的相关设计与实现思路。由于麻将是一个益智休闲类的棋牌游戏，因此不会使用 pygame 中的 sprite 这类动作游戏常用模块，而是主要通过 display、surface、event、image 等模块创建一个静态的界面。

图形化界面中的所有内容都创建在一个 surface 对象 screen 之上，做个形象的类比，screen 就相当于一张纸，所有游戏的插图，绘画，图形都只能显示在这张纸上。游戏过程中利用 rect 模块中的 clip 函数创建按钮，并配合 event 模块中

的 get 函数与 mouse 模块中的 get_pos 函数，一旦检测到鼠标移动至我们创建的矩形区域内，并且还捕捉到鼠标左键按下的事件时，说明用户按下了按钮。此时便需要给予用户反馈，即按钮变色，并且执行按钮所对应的功能。

用户舍牌控制也是同样的原理，只需要将用户的所有手牌视作 14 个按钮即可，依旧是利用上述模块，检测到用户在哪张牌所在区域中点击时便可得知用户希望打出哪张牌，此时直接将其打出即可。

碰功能设计：碰牌是只有当其他三家打出牌的同时，我们手里还恰好有 2 张及以上相同牌的情况下可以触发的功能。但是由于其他三家每出一张牌我们都需要判断能否碰牌，因此如果能否碰牌的函数设计非常复杂则会导致整个游戏运行卡顿。所以为了避免这种情况，我在设计判断碰牌函数时首先会进行一轮遍历，如果手牌中没有那张别家打出的舍牌则直接不进行后续判断。如果存在，则记录下该牌第一张出现的位置，如果此位置已是这副牌的最后一张则返回。如过上述条件都满足且该位置的后一张牌也是我们所需要寻找的牌则提示用户是否需要碰，具体可以参考图 6。

图 6 碰牌提示

杠功能设计：杠功能的整体设计思路与碰一致，不同点为杠牌只有手牌中有 3 张相同牌的情况下才可以触发。而且杠牌后将会额外翻开一枚宝牌指示牌，

即场上会额外出现 4 枚宝牌。

图 7 杠牌提示

吃功能设计：吃功能是一个较为复杂的功能，主要体现在其多样性。由于吃牌时往往不一定只会有一种可以吃的牌型，因此需要进行多次判断。如图 8 所示，此时上家打出六万，且玩家手牌中同时包含四五七八万，就会瞬间产生 3 种可能的吃牌牌型。需要利用分支语句对这三种情况分别讨论，并在可视化界面中向玩家呈现所有的吃牌可能性，让玩家自行选择。

图 8 吃牌提示

判胡算法设计：判胡算法是整个实验中最难设计的一部分，因为麻将的胡牌牌型丰富且多变，因此这一块算法的设计着实下了不少功夫。

我将麻将中胡牌的牌型分为了两类：一般型与特殊型。特殊性的判断比较简

单，因此我将先介绍特殊性。特殊性的胡牌还可以分为两种：七对子与国士无双。七对子牌型的特点是整副手牌完全由对子构成，因此我们只需要设置一个以 2 为步长的变量遍历手牌，并判断其与其后的一张牌是否为同一张并且与前一张不一致即可，如果出现反例则不满足七对子胡牌条件，反之则满足条件。

图 9 七对子牌型解析

国士无双牌型的特点为手牌中包含全 13 张不同的幺九牌，并且剩余的那张为任意幺九牌。国士无双牌型的判断可以按照幺九牌的种类依次删除手牌中对应的牌，如果过程中发现手牌中缺少了哪种幺九牌则不满足国士无双胡牌条件。如果遍历完 13 种幺九牌后依然没有发生错误，则对剩余的 1 枚牌进行判断，如果为幺九牌则胡牌成功，具体可以参考图 10。

图 10 国士无双牌型解析

特殊性的牌型介绍完毕后，我将介绍最为复杂的一般性牌型胡牌判断条件。一般型指的是由 4 组顺子/刻子加一对雀头（任意两张相同的牌）组成的牌型，也是大多数情况下我们会优先考虑的牌型。我绘制了一般型胡牌判断函数执行的流程图如下：

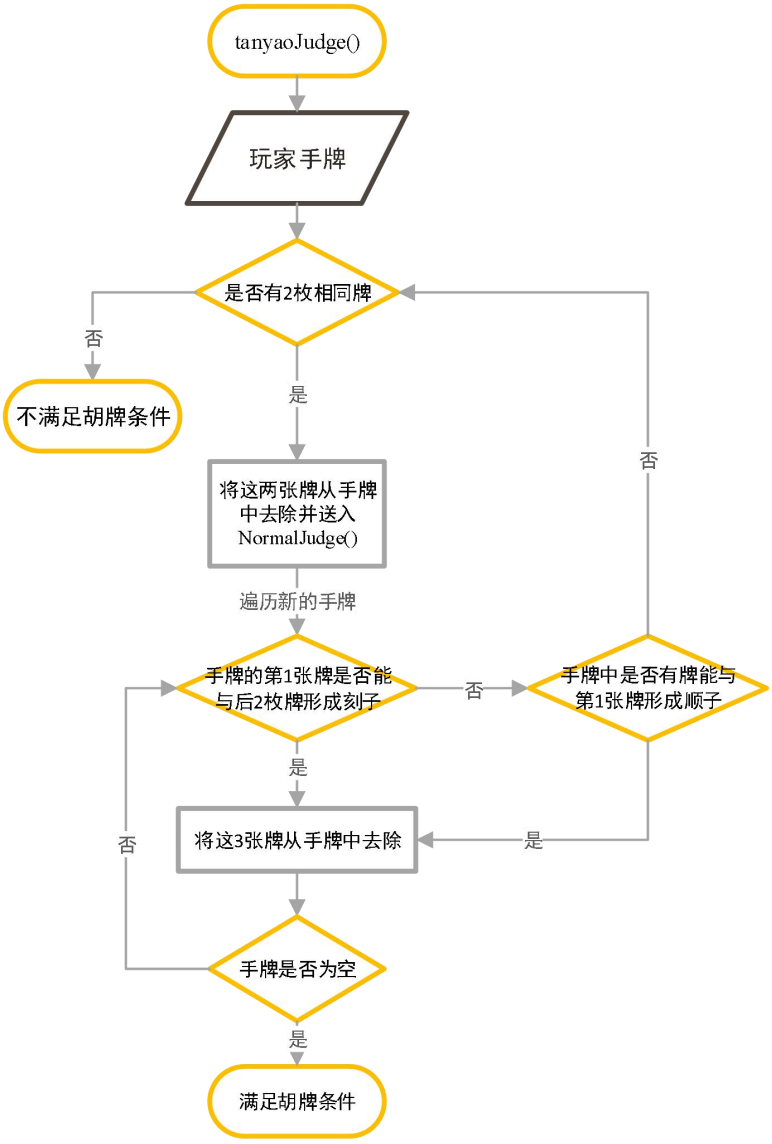


图 11 一般型胡牌函数判断流程图

接下来我将会使用文字来解析一遍判断流程。首先我们将手牌传入 huJudge 函数后会优先进行特殊性判断，如果不满足条件则正式进入一般型判断流程。

将玩家手牌传入 tanyaoJudge 函数后，首先遍历手牌寻找雀头候补。当发现

两张一样的手牌后，对整副手牌进行深拷贝复制，并去除刚刚发现的雀头，形成新的手牌并传入 NormalJudge 中进行进一步判断。

进行到这一步后我们可以知道传入 NormalJudge 中的这一副手牌中每一张牌都必定需要被一组刻字或顺子所容纳，否则将不满足胡牌条件。因此我们先判断相对简单的刻子。由于刻子是由 3 枚完全一致的牌组成的，而我们的手牌又是按照牌序摆放，因此同样的牌必定连续排列，我们只需要对该牌的后 2 枚牌进行判断即可。如果满足刻子条件则删除这 3 张牌并再一次进行判断；如果不满足条件则需要到手牌中寻找是否存在 2 枚能够与其组成顺子的牌；反之如果能够顺利找到一组顺子则删除这 3 枚牌并重复寻找下一组刻子。

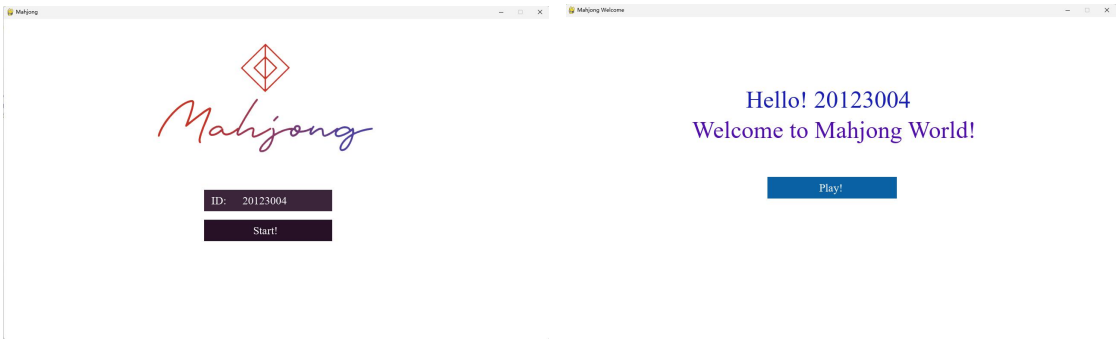
循环执行上述流程直至手牌为空/无法找到对应的顺子组合。若手牌为空则返回满足胡牌条件，若发生错误则返回 tanyaoJudge 中寻找下一组雀头候补，并循环这个流程。如果遍历了整副手牌的雀头候补都不能胡牌则说明不满足一般型胡牌条件。

以上便是我对整个实验过程中较为复杂的函数实现方法的介绍以及一些运行结果的展示。

五、测试用例（>2 个用例说明）

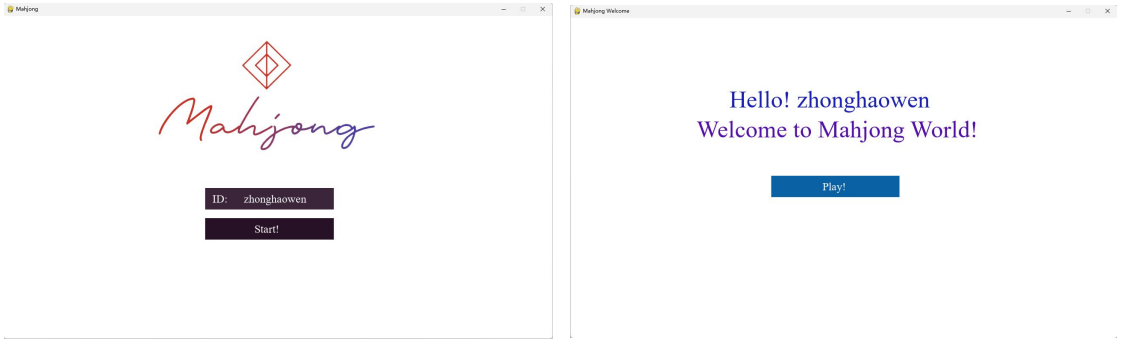
欢迎界面测试：

测试用例 1：20123004（纯数字）



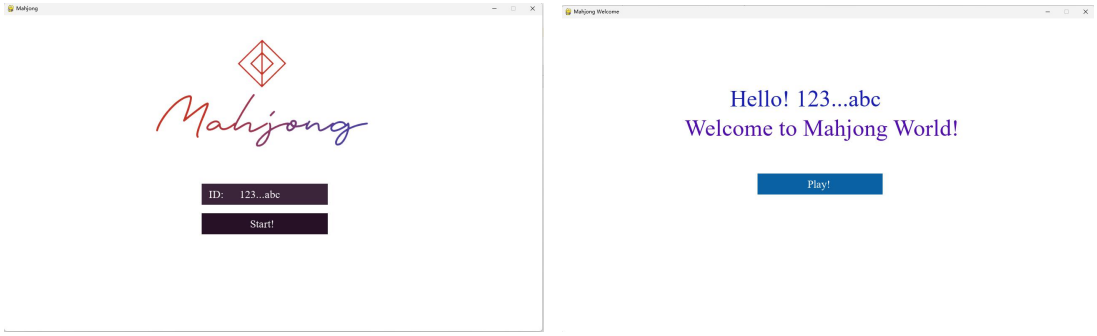
如测试结果图所示，20123004 被顺利输入。

测试用例 2：zhonghaowen（纯英语）



如测试结果图所示，zhonghaowen 被顺利输入。

测试用例 3：123//…abc（混合型，包含非法字符）



如测试结果图所示，123//…abc 未被顺利输入，仅显示得到 123…abc。

通过上面 3 个对于欢迎界面用户 ID 的测试，我们可以发现 ID 输入功能和预想的完全一致，仅有数字、字母以及“.”是程序中设置的合法字符，除了合法字符外，其余字符都无法输入。在 Welcome 界面中“Hello!”后也可以正确地显示刚刚用户输入的 ID 名称，并且根据名称的长度调整欢迎语句显示的位置。

舍牌功能 brand 测试：

测试用例 1：舍弃以下这副手牌中的第 1 张牌。

如图所示，八万被成功打出，三饼被理入手牌。

测试用例 2：舍弃以下这副手牌中刚刚摸到的五万。

如图所示，五万被摸切，手牌与副露牌没有发生变化。

测试用例 3：舍弃以下这副手牌中的五条实现听牌。

如图所示，打出五条后，下一巡自摸六万。

通过以上对切牌的 3 个测试样例可以得出：Mahjong 游戏的切牌功能完全可以仅仅依靠 pygame 的游戏窗口实现，无需借助控制台。而且运行精确流畅，出牌结果与鼠标点击的牌完全一致。

吃牌功能 CHOW 测试：

测试用例 1：吃下东家打出的这张六条。

如图所示，这一副手牌中只持有四条与五条，因此仅提示用户一种吃牌可能。吃下六条后玩家手牌长度缩短，并且四五六条以组合的形式出现在屏幕右下角。

测试用例 2：用五六饼吃下东家打出的这张四饼。

如图所示，这一副手牌中同时持有三五六饼，因此会产生 2 种吃牌可能。用五六饼吃下东家打出的这张四饼后，四五六饼以组合的形式出现在屏幕右下角。

测试用例 3：不吃西家打出的这张七饼。

如图所示，这一副手牌中同时持有五六八九饼，因此会产生 3 种吃牌可能。但是这里我们按照测试用例的要求选择点击“跳过”。可以发现手牌没有发生任何变化，也没有副露。

通过以上几个测试用例，我们认为吃牌功能是可以正常运行的。

碰牌功能 PUNG 测试：

测试用例 1：碰下东家打出的这张四万。

如图所示，玩家手上有两张四万，当有别家打出另一张四万时，系统便会提

示玩家可以碰，碰牌后，三张四万以组合形式出现在屏幕右下角。

测试用例 2：碰下南家打出的这张三万。

如图所示，玩家手上有两张三万，当有别家打出另一张三万时，系统便会提示玩家可以碰，碰牌后，三张四万以组合形式出现在屏幕右下角。此时 3 张三万与先前碰的 3 张四万都会显示在屏幕右下角。

测试用例 3：不碰北家打出的这张八条。

如图所示，玩家手上有两张八条，此时北家打出八条，系统提示玩家可以碰，碰牌。按照测试要求按下“跳过”。此时我们的手牌没有发生变化，直接进入了下一巡的摸牌环节。

通过以上三个测试样例的实验，可以认为 Mahjong 游戏的碰功能是完善的。

杠功能 GANG 测试：

测试用例 1：杠北家打出的这张一万。

如图所示，玩家手上有 3 张一万，当北家打出第 4 张一万时，玩家就可以选

择开杠。此时四张一万摆放在屏幕右下角，且屏幕上方翻开一张宝牌指示牌。

测试用例 2：杠自己摸到的这枚二万。

如图所示，玩家手上有 3 张二万，且自己摸到了第 4 枚二万，则可以立即选择开杠。此时四张二万与一万一同摆放在屏幕右下角，且屏幕上方额外翻开一张宝牌指示牌。

测试用例 3：不杠西家打出的这张九万。

如图所示，玩家手上有 3 张九，此时西家打出第 4 张九万，我们选择不开杠。此时并没有继续游戏，而是提示用户可以碰。因为开杠的判断条件其实比碰更加严苛，因此如果满足开杠条件则也一定满足了碰牌条件，所以此时会弹出窗口询问玩家是否需要碰牌。

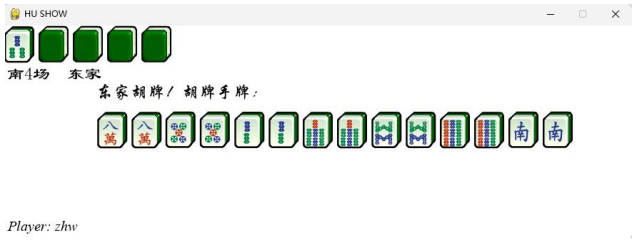
胡牌功能 HU 测试：

测试用例 1：胡东家打出的这张五饼。

如图所示，此时玩家已经听牌，听牌牌型为一般型。当别家打出的牌与玩

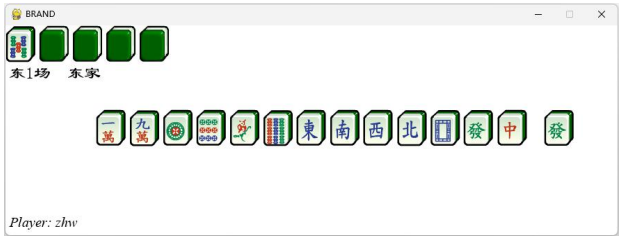
家手牌中的 13 枚牌可以组成满足胡牌条件的手牌时便会提示玩家可以胡牌。胡牌时会将所有已副露的手牌重新加入手牌中展示胡牌手牌。

测试用例 2：胡北家打出的这张九条。



如图所示，此时玩家已经听牌，听牌牌型为七对子。此时北家打出的九条恰好与玩家的手牌可以凑成胡牌牌型，因此可以直接胡牌。

测试用例 3：不自摸国士无双。



如图所示，此时玩家已经听牌，听牌牌型为国士无双。此时自摸了发财，但是由于选择了跳过，则直接进入弃牌环节，玩家需要打出一张不需要的手牌。

通过以上三个测试样例的实验，可以认为 Mahjong 游戏的胡牌判断以及胡牌功能是完善的。

六、收获与体会

本次综合实验我实现了一款可视化的麻将游戏 Mahjong。从开发到页面的设计都是我从零开始体验的一个过程，非常有意思，也让我体会到游戏开发的工程量其实相当大。最初选择麻将作为本次综合实验的主题其一是希望能体验一把开发游戏的感觉，另一点是希望能够选择一款不是特别复杂但是又比较有挑战的游戏进行实现。权衡之下选择了我比较熟悉的麻将。虽然实验中仅仅是对于麻将的基础功能进行了实现，但是整体代码量已经达到了 900 多行，整整写了将近两周，

非常地具有挑战性。

对于 pygame 这个库其实我在实验中只实践了很有限的一部分功能，包括其精髓“精灵”也没有机会体验。主要是受限于游戏本身的性质，麻将并非是动作类需要计算碰撞体积等等参数的游戏，但是我已经可以相当熟练地使用实验中所出现的这几个 pygame 模块。由于我在开发过程中是首先用控制台实现的麻将游戏功能，因此当看到一张张麻将牌出现在真正的游戏画面中时，内心还是有种难以压抑的激动。

本次实验中除了外部库的使用外另一点让我感觉比较有挑战性的便是胡牌算法的编写。由于我所采取的游戏规则为日麻规则，因此在网络上这方面的资料相对较少，基本是只能依靠自己来思考如何才能实现这个功能，尤其是一般型的胡牌判断。最后能够成功地实现也是相当不易。

当然这个项目目前还存在相当大的局限性，例如计分系统还没有实现；除真实玩家外的其余玩家可以编写 AI 来控制其出牌的规则，而不是直接摸切；以及整个游戏画面还有很大的美化空间。这些在课后我都会去一一实践尝试作出改进。

最后我还想谈谈我的一些感想。我觉得做实验，做项目都必须要投入自己的热情。我最终能够以一个自己相对满意的形式完成这个麻将的项目很大程度上是依靠着热情。实验过程不可避免地会遇到很多障碍，很多陌生的领域，甚至没有其他人的经验，只能自己摸索，但是只要仍然怀揣热情，困难总是可以克服的。我觉得能有一个机会去实现自己想体验的项目是一件非常幸运的事情，这次的实验过程中真的学习到了很多，体验到了很多，我认为十分有价值。

七、核心代码

以下是胡牌算法的具体代码，共 62 行：

```
def huJudge(hands, Chowlis, Punglis, Ganglis):
    count = len(Chowlis)/3 + len(Punglis) + len(Ganglis)
    if count == 0:
        if nanaJudge(hands) or kokushimusoJudge(hands):
            return True
    return tanyaoJudge(hands, count)

def nanaJudge(hands):
    i = 0
    temp = "0"
    while i < 12:
        if temp == hands[i]:
            return False
        if hands[i] != hands[i+1]:
            return False
        temp = hands[i]
        i += 2

    return True

def kokushimusoJudge(hands):
    handscopy = copy.deepcopy(hands)
    yaolist = ["1m", "9m", "1p", "9p", "1s", "9s", "d1", "d2", "d3",
"f1", "f2", "f3", "f4"]
    for i in yaolist:
        if i in handscopy:
            handscopy.remove(i)
    if len(handscopy) == 1 and handscopy[0] in yaolist:
        return True
    else:
        return False

def tanyaoJudge(hands, count):
    for i in hands:
        if i[0] in "19fd":
            return False
    i = 0
    while(i < len(hands)-1):
        handscopy = copy.deepcopy(hands)
```

```
        if hands[i] == hands[i + 1]:
            handscopy.pop(i+1)
            handscopy.pop(i)
            if NormalJudge(handscopy, count):
                return True
        i += 1
    return False

def NormalJudge(hands, count):  # 如果是一般型
    i = 0
    while count < 4 and i < len(hands) - 2:
        if hands[i+2] == hands[i+1] == hands[i]:  # 刻子判断
            count += 1
            for _ in range(3):
                hands.pop(i)
            i -= 1
        else:   # 顺子判断
            if str(int(hands[i][0])+1) + hands[i][1] in hands and
str(int(hands[i][0])+2) + hands[i][1] in hands:
                count += 1
                hands.remove(str(int(hands[i][0]) + 1) + hands[i][1])
                hands.remove(str(int(hands[i][0]) + 2) + hands[i][1])
                hands.pop(i)
                i -= 1
            else:
                return False
        i += 1
    return True
```