河北农业大学
理工系

Python 程序课程设计

题　　目：	台球
专业班级：	计算机科学与技术 2308 班
团队名称：	憧憬成为 Python 高手！
团队成员：	高鹤箫　耿延康　付宇轩
日　　期：	2025.1.5

目录

一、引言

1.1 设计背景

台球作为一项广受欢迎的娱乐和竞技运动，因其对技巧和策略的高要求而深受广大爱好者的喜爱。然而，由于场地、时间和设备等方面的限制，并不是每个爱好者都能随时随地享受台球带来的乐趣。在一次闲暇之际，与舍友一起前往台球厅打台球时，我们深深感受到这项运动的魅力。然而，频繁前往台球厅并不现实，这促使我们思考如何通过技术手段突破这些限制。

出于对台球运动的热爱和对编程技术的兴趣，我们决定利用 Python 语言编写一个台球模拟程序。这一设计不仅能够满足我们随时随地玩台球的需求，还能为更多台球爱好者提供一种便捷的娱乐方式。通过模拟真实的物理碰撞和运动轨迹，该程序能够让用户在虚拟环境中体验真实的台球游戏，从而在娱乐中提高自己的技能。

1.2 整体功能描述

1)发球功能

2)自由球功能

3)幕间提示功能

4)游戏逻辑功能

5)物理模拟功能

6)键盘操控功能

1.3 关键技术

（1）库：pygame 库、sdl 库、pymunk 库、flameprof 库、random 库、time 库

（2）技术：异步执行、状态机、滑动步长、性能计数器

二、总体设计

2.1 功能模块的总体设计

(1)class Ball: 定义了球的相关数据结构、相应的物理对象、绘制方法

(2)class Wall: 定义了库的相关数据结构、相应的物理对象、绘制方法

(3)class BilliardTable:定义了球桌的相关数据结构、其相应的球、库对象、绘制方法

(4)class GUI:定义了图形用户界面的相关元素、绘制、操作方法

(5)class Game:定义了游戏主函数包含所有的资源、窗体事件循环

2.2 功能模块的详细设计

(1)class Ball：

初始化时传入编号(index)、初始位置(position)、颜色(color)，定义质量为 20、半径 25、力矩、刚体、形状、弹性系数 0.98、从文件加载贴图、是否显示。

绘制方法：传入屏幕对象(screen)，检查是否显示，如显示则于屏幕对象上绘制贴图

施加冲量方法：传入冲量对象(impulse),调用刚体内置的于中心点施加冲量方法

(2)class Wall:

初始化时传入起始位置(start)、终止位置(end)、物理空间对象(space)，定义刚体形状、弹性系数 0.8，将其添加到物理空间对象中

绘制方法：传入屏幕对象，在屏幕上画线

(3)class BilliardTable:

初始化时传入屏幕对象(screen)和物理空间对象(space)，定义球列表、单幕掉落球列表、球权、库列表，调用创建球方法、创建墙方法，从文件加载标题贴图、球撞球音效、球撞库音效，调用混音器加

载背景音乐、重复 99999 次、音量 0.3，创建碰撞事件监听器并启动，注册回调函数。

重置方法：丢弃单幕掉落球列表、使用空列表代替、清除球权、创建行位置列表，从 0 到 15 遍历 index、将球刚体位置设为列表中对应值、设置球显示为真、设置球刚体速度为 0

碰撞监听器回调函数：接受碰撞监听器传入的 3 个参数(arbiter, space, data)，从 arbiter 中读出参与碰撞的两个物体的速度对象、计算相对速度的大小，从混音器获取新的声音通道将速度$\times \mathbf{0.0005} + \mathbf{0.1}$后设置为新获取到的声音通道对象的音量，并在该通道上播放相应的碰撞音效

创建球方法：创建球颜色列表、创建行位置列表，从 0 到 15 遍历 index、创建新球（位置为列表中对应值，颜色为列表中对应值），加入球列表，加入物理空间

创建库方法：创建库位置列表，遍历列表创建库（位置为列表中对应值）入库列表

绘制方法：绘制台球桌，创建袋口列表，遍历袋口列表绘制袋口，遍历库列表绘制库，遍历球列表绘制球，将标题绘制到屏幕

(4)class GUI:

初始化时传入屏幕对象，实例化幕间提示界面自由球界面，幕间计数器设置为 1，鼠标位置设置为$(\boldsymbol{0},\boldsymbol{0})$

图形用户界面元素(基类):初始化时传入屏幕对象，将开关设置为关、开率设置为 0、$\boldsymbol{\sigma}$设置为 0.1

插值方法：如果开关为开且开率小于 100，开率设置为

$$\boldsymbol{R_{open}(r_{open})=\begin{cases} r_{open}+(100.1-r_{open})\sigma, & r_{open}<100 \\ 100, & r_{open}\geq 100 \end{cases}}$$

否则如果开率大于 0，开率设置为

$$\boldsymbol{R_{open}(r_{open})=\begin{cases} r_{open}-(100.1-r_{open})\sigma, & r_{open}>0 \\ 0, & r_{open}\leq 0 \end{cases}}$$

绘制方法：调用插值方法

自由球界面：初始化时调用父类的初始化方法、创建与屏幕对象相同大小的平面对象，覆写$\boldsymbol{\sigma}$为 0.1

覆写插值方法：如果开关为开且开率小于 100，开率设置为

$$\boldsymbol{R_{open}(r_{open})=\begin{cases} r_{open}+(100.1-r_{open})\sigma, & r_{open}<100 \\ 100, & r_{open}\geq 100 \end{cases}}$$

否则如果开率大于 0，开率设置为

$$\boldsymbol{R_{open}(r_{open})=\begin{cases} r_{open}+(-0.1-r_{open})\sigma, & r_{open}>0 \\ 0, & r_{open}\le 0 \end{cases}}$$

检查位置是否可用函数：传入鼠标位置，从第二个成员开始遍历球列表，设置最小距离为 50，使用勾股定理判断距离，返回是否与球重叠

覆写绘制方法：传入鼠标位置、鼠标状态、球列表，调用父类的绘制方法。如果自身开率大于 0 将返回值设为假，颜色设为蓝色，将鼠标位置钳制在屏幕内，如果鼠标位置可用，颜色设置为青色；如果鼠标状态为按下且开关为打开状态，返回值设为真，关闭自身开关，否则颜色设为红色，清空平面，将平面透明度设置为

$$\boldsymbol{O_{sur}=\left(\frac{r_{open}}{100}\cdot 500\right)-256}$$

绘制圆形、绘制对象为自身平面、绘制颜色为先前生成的颜色、绘制位置为钳制过的鼠标位置、半径为

$$\boldsymbol{r=25+\left(1-\frac{r_{open}}{100}\right)\cdot 1280}$$

厚度为

$$\boldsymbol{\Delta r=int(125-r_{open})}$$

将表面绘制到屏幕，返回返回值。

幕间提示界面：初始化时调用父类的初始化方法、创建与屏幕

对象相同大小的平面对象，从文件加载字体，加载音效、加载背景图片、创建与屏幕对象相同大小的平面对象、覆写$\boldsymbol{\sigma}$为 0.08

切换方法：传入主要提示语和次要提示语，从混音器获取新的声音通道，音量设置为 1，并在该通道上播放相应的幕间提示音效，将自身开关设为打开，将背景图片绘制到表面上，渲染主要提示语，将位置调整为

$$\boldsymbol{pos_{t1}} = (\frac{\boldsymbol{x_{scr}}}{\boldsymbol{2}}, \frac{\boldsymbol{y_{scr}}}{\boldsymbol{2}} + \boldsymbol{5})$$

并绘制到表面，上渲染次要提示语，将位置调整为

$$\boldsymbol{pos_{t2}} = (\frac{\boldsymbol{x_{scr}}}{\boldsymbol{2}}, \frac{\boldsymbol{y_{scr}}}{\boldsymbol{2}} + \boldsymbol{100})$$

并绘制到表面上.

覆写绘制方法：调用父对象的绘制方法。如果自身打开且自身开率为 100，关闭自身开关。如果自身开率大于 0，将自身表面的透明度设置为

$$\boldsymbol{O_{sur}} = \left(\frac{\boldsymbol{r_{open}}}{\boldsymbol{100}} \cdot \boldsymbol{500}\right) - \boldsymbol{256}$$

宽度因子设置为

$$\boldsymbol{F_w} = \boldsymbol{2} - (\frac{\boldsymbol{r_{open}}}{\boldsymbol{100}})$$

高度因子设置为宽度因子。获取屏幕宽高$(\boldsymbol{H_{scr}}, \boldsymbol{W_{scr}})$，设置绘制起始位置为

$$\boldsymbol{Pos_{init}} = \left(\frac{\boldsymbol{F_w} \cdot \frac{\boldsymbol{r_{open}}}{\boldsymbol{100}} - \boldsymbol{W_{scr}}}{\boldsymbol{2}}, \frac{\boldsymbol{F_h} \cdot \frac{\boldsymbol{r_{open}}}{\boldsymbol{100}} - \boldsymbol{H_{scr}}}{\boldsymbol{2}}\right)$$

在屏幕上绘制表面，大小为

$$\boldsymbol{Rect_{sur} = (W_{scr} \cdot F_w, H_{scr} \cdot F_h)}$$

初始位置为$\boldsymbol{Pos_{init}}$

仲裁器:静态类，不会被实例化。

有黑(静态方法)：传入球列表，返回判断一个遍历球列表的元素 index 属性是否等于 8 的生成器生成的布尔值是否存在真

有白(静态方法)：传入球列表，返回判断一个遍历球列表的元素 index 属性是否等于 0 的生成器生成的布尔值是否存在真

首颜色(静态方法)：传入球列表，顺序遍历球列表，如果元素 index 属性大于等于 1 小于等于 7，立即返回"FullC"，如果元素 index 属性大于等于 8 小于等于 15，立即返回"HalfC"，否则返回 None

反颜色(静态方法)：传入 color 字符串，如果为"FullC"立即返回"HalfC"，如果为"HalfC"立即返回"FullC"。

绘制方法：传入球桌对象，现在的状态，上一个状态，鼠标位

置，鼠标状态，调用自由球界面对象实例中的绘制方法，传入鼠标位置、鼠标状态、球桌的球列表，如果返回真，球桌的球列表中的第一个成员的刚体的位置设置为鼠标位置，球桌的球列表中的第一个成员的刚体速度设置为$(\mathbf{0},\mathbf{0})$，隐藏球桌的球列表中的第一个成员。获取对单幕掉落球列表、球权、仲裁器中的有黑方法、仲裁器中的有白方法、仲裁器中的首颜色方法、仲裁器中的反颜色方法的引用。将提示语预设为"出现错误： 未经处理的异常"，如果上一个状态为空字典，则打印"scene start!"调用幕间提示界面的切换方法，传入两个字符串("第 1 幕","开球，分出球权")。否则如果现在的状态中键'ball_moving'的值不为真并且上一个状态中键'ball_moving'的值为真，打印"balls_dropped: "、一个遍历单幕掉落球列表的元素 index 属性的生成器转换而成的列表、"ball_right: "、球权。如果球权未定义如果单幕掉落球列表为空列表，提示词设置为"无球进洞， 交换球权"，否则如果调用有黑，传入单幕掉落球列表返回真，提示词设置为"黑 8 进洞，重新开局"，调用球桌的重置方法，场景计数器归零。否则如果调用有白，传入单幕掉落球列表返回真，提示词设置为"白球进洞， 交换球权， 自由球"，将自由球界面的开关设置为开。否则将第一个掉落的颜色类型设置为调用首颜色传入单幕掉落球列表的返回值，并将球权设置为第一个掉落的颜色。如果第一个掉落的颜色类型为"FullC",提示词设置为"全色球进洞， 球权归于该色球"。否则如果第一个掉落的颜色类型为"HalfC",提示词设置为"全色球进洞， 球权归于该色球"。否则提示词设置为"出现错误： 未经处理的异常"。否则如果球权等于

"FullC"或者球权等于"HalfC"，如果球权未定义如果单幕掉落球列表为空列表，提示词设置为"无球进洞， 交换球权"，否则如果调用有黑，传入单幕掉落球列表返回真，提示词设置为"黑 8 进洞， 重新开局"，调用球桌的重置方法，场景计数器归零。否则如果调用有白，传入单幕掉落球列表返回真，提示词设置为"白球进洞，交换球权，自由球"，将球桌的球权设置为调用反颜色、传入球权的返回值，将自由球界面的开关设置为开。否则如果调用首颜色、传入单幕掉落球列表的返回值等于球权，提示词设置为"同色球进洞， 球权保持"，否则提示词设置为"异色球进洞， 交换球权，自由球"，将球桌的球权设置为调用反颜色、传入球权的返回值，将自由球界面的开关设置为开。否则提示词设置为"出现错误： 未经定义的情形"。将自身的场景计数器加一。打印"scene change! \"{prompt_text}\""，其中 prompt_text 为提示词。调用幕间提示界面的切换方法，传入"第{self.scene_counter}幕"，self.scene_counter 为自身的场景计数，提示词两个字符串，清空球桌的单幕掉落球列表。调用幕间提示界面的绘制方法。

(5)class Game:

初始化时初始化 pygame，初始化混音器，启动 128 个阴道，启动屏幕对象，设置为 1280x720，启动 pygame 时钟，设置正在运行为真。设置$\boldsymbol{\Delta t = 0}$，创建 pymunk 物理空间对象，设置物理空间重力为$(\boldsymbol{0}, \boldsymbol{0})$，设置物理空间阻尼为 0.5，实例化 BilliardTable 和 GUI

，设置鼠标状态为[假,假]，鼠标上一个位置为$(\mathbf{0},\mathbf{0})$，鼠标位置为$(\mathbf{0},\mathbf{0})$,自身上一个状态为空字典。

运行时：循环直到自身正在运行为假：“对 pygame 的事件列表遍历，如果事件类型为退出，将自身正在运行设置为假。调用球桌的绘制方法，将现在的状态设置为调用更新物理方法，传入 BilliardTable 的单幕掉落球列表的返回值,调用 GUI 的绘制方法，传入 BilliardTable,现在的状态，上一个状态，鼠标位置，鼠标状态的第二个元素。调用处理输入方法，传入现在的状态、上一个状态。将上一个状态设置为这一个状态。翻转屏幕缓冲区。将屏幕填充为颜色$(\mathbf{0},\mathbf{28},\mathbf{0})$”,跳出循环后游戏退出。

处理输出方法：传入现在的状态，将自身的鼠标状态的第二个元素设置为第一个元素的值，将自身的鼠标状态的第一个元素设置为 pygame.mouse.get_pressed() 返回的列表的第一个元素。创建对 pygame.key.get_pressed() 返回的列表的引用，如果列表中存在 pygame.K_w，则调用 BilliardTable 中的球列表中的第一个元素的施加冲量方法，传入$(\mathbf{0},-\mathbf{1000})$如果列表中存在 pygame.K_s，则调用 BilliardTable 中的球列表中的第一个元素的施加冲量方法，传入$(\mathbf{0},\mathbf{1000})$如果列表中存在 pygame.K_a，则调用 BilliardTable 中的球列表中的第一个元素的施加冲量方法，传入$(-\mathbf{1000},\mathbf{0})$如果列表中存在 pygame.K_d，则调用 BilliardTable 中的球列表中的第一个元素的

施加冲量方法，传入$(\mathbf{1000},\mathbf{0})$。创建 pygame.mouse.get_pos()返回的元组的引用，存为自身的鼠标位置。如果鼠标状态的第一个成员为真并且鼠标状态的第二个成员为假，将自身的鼠标的上一个位置设置为自身鼠标位置的值。如果鼠标状态的第一个成员为真，并且现在的状态中键'ball_moving'的值为真，那么绘制线条，颜色为

$$(\boldsymbol{Rand_{int}}(\mathbf{0},\mathbf{255}),\boldsymbol{Rand_{int}}(\mathbf{0},\mathbf{255}),\boldsymbol{Rand_{int}}(\mathbf{0},\mathbf{255}))$$

起始位置为

$$\boldsymbol{pos_{start}} = (\boldsymbol{x_{pos_{ball}}} - \left(\boldsymbol{x_{pos_{mouse_{last}}}} - \boldsymbol{x_{pos_{mouse}}}\right), \boldsymbol{y_{pos_{ball}}} - \left(\boldsymbol{y_{pos_{mouse_{last}}}} - \boldsymbol{y_{pos_{mouse}}}\right))$$

结束位置为 BilliardTable 中的球列表的第一个元素的刚体的位置，宽度为 5。如果自身的鼠标状态的第一个元素为假并且自身的鼠标状态的第二个元素为真并且现在的状态中键'ball_moving'的值为假，创建元组$\boldsymbol{T_{line}}$为

$$\boldsymbol{T_{line}} = (\left(\boldsymbol{x_{pos_{mouse_{last}}}} - \boldsymbol{x_{pos_{mouse}}}\right), \left(\boldsymbol{y_{pos_{mouse_{last}}}} - \boldsymbol{y_{pos_{mouse}}}\right))$$

调用 BilliardTable 中的求列表的第一个元素的刚体的施加冲量方法，传入$\boldsymbol{T_{line}}$。

更新物理方法：设置最小速度$\boldsymbol{V_{min}} = \mathbf{100}$,球在移动为假，清场为真，遍历自身 BilliardTable 中的球列表，设置速度$\boldsymbol{V_i}$为球的刚体速度长度。如果$\boldsymbol{V_i} \leq \boldsymbol{V_{min}}$,则设置球的刚体速度为

$$\boldsymbol{V_i} = (\mathbf{0.9}\boldsymbol{x_{v_i}}, \mathbf{0.9}\boldsymbol{y_{v_i}})$$

设置位置$\boldsymbol{Pos_i}$为球的刚体位置，如果球显示为真且满足

$$x_{pos_i} < \frac{W_{scr}}{2} - 500$$

$$x_{pos_i} > \frac{W_{scr}}{2} + 500$$

$$y_{pos_i} < \frac{H_{scr}}{2} - 250$$

$$y_{pos_i} > \frac{H_{scr}}{2} + 250$$

中的任意一个表达式,那么设球的显示为假并追加到单幕掉落球列表中。如果球显示为真且满足$V_i > 100$, 设球在移动为真。如果如果球显示为真，设清场为假。设自身的Δt为

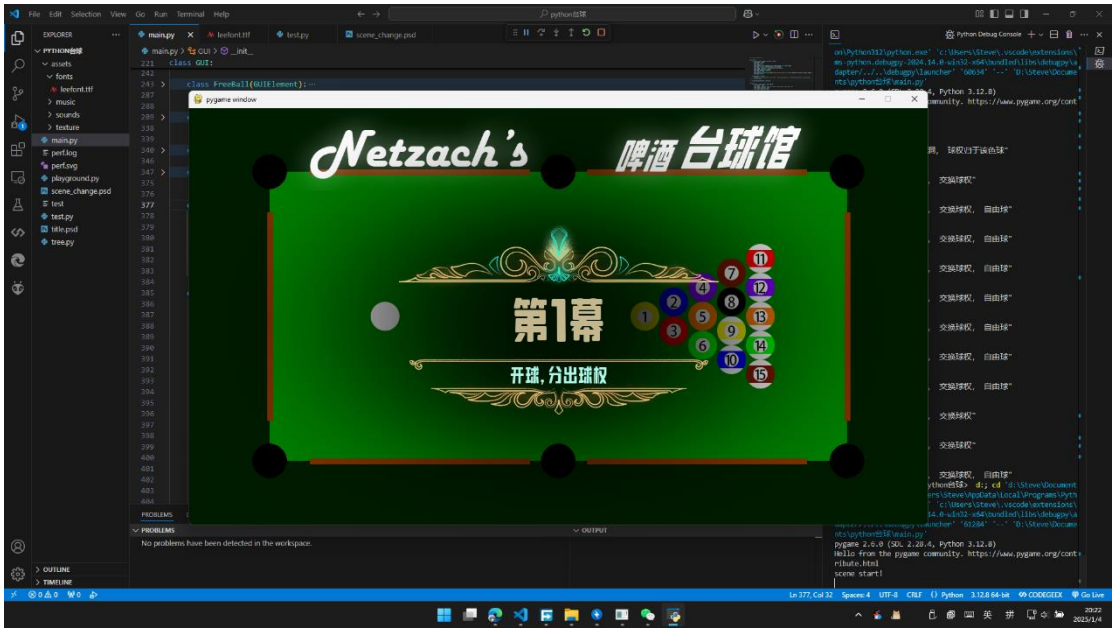
$$\Delta t(s) = \frac{\Delta t_{frame}(ms)}{1000(ms/s)}$$

调用自身物理空间对象的步进函数，步长为Δt。返回一个字典{"ball_moving": 球在移动, "stage_clear": 清场}

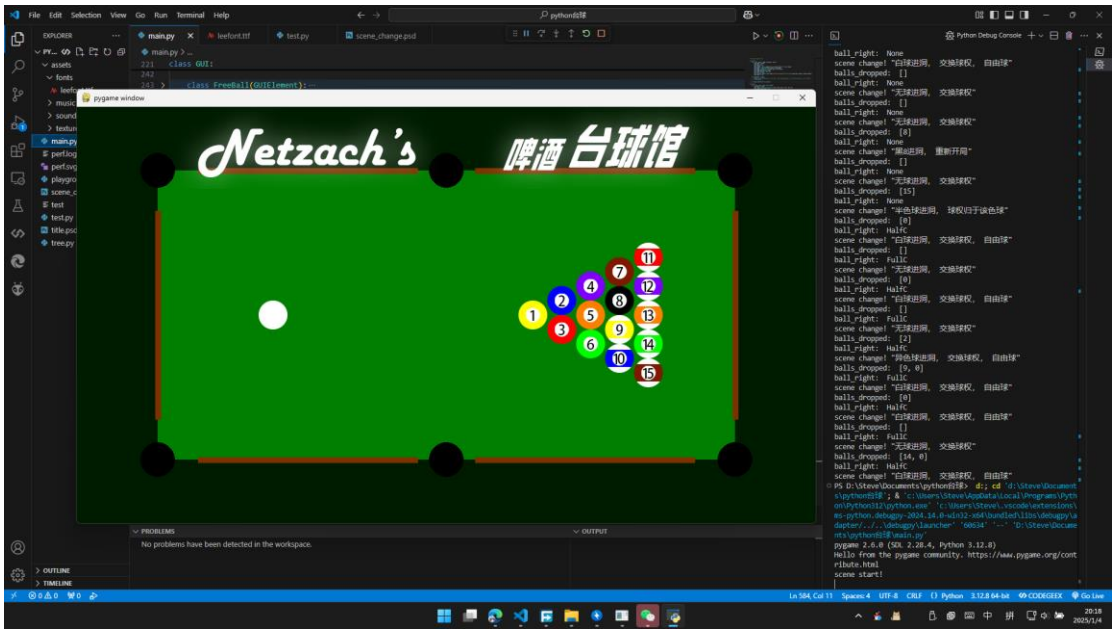
程序运行时实例化 Game，并调用 Game 的运行方法。

2.3 运行结果和测试

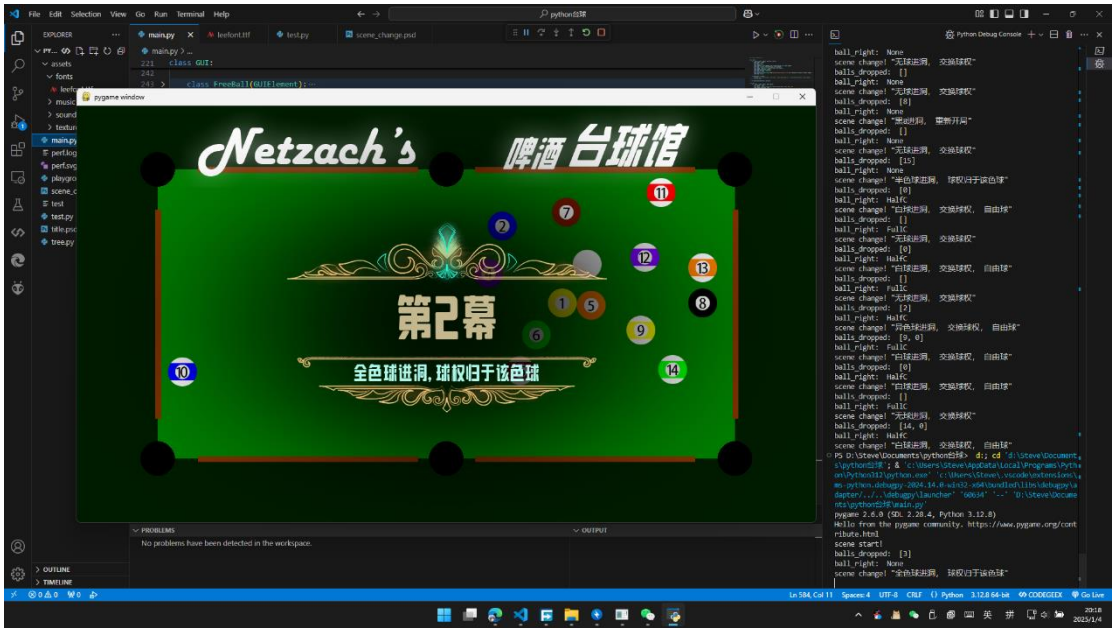
(1) 初始页面



(2)排布



(3)进行中



(4)自由球

2.4 结论和展望

(1)结论

这个台球模拟程序的设计背景是希望通过技术手段解决台球运动中由于场地、时间和设备限制带来的问题。通过使用 Python 语言

编写，程序能够模拟真实的台球游戏，让用户随时随地体验台球运动并提升技能。

功能方面，程序包括发球、自由球、幕间提示、游戏逻辑、物理模拟和键盘操控等模块。设计中运用了多个 Python 库（如 pygame、pymunk 等）和技术（如异步执行、状态机等）来实现物理碰撞、游戏控制和用户交互。

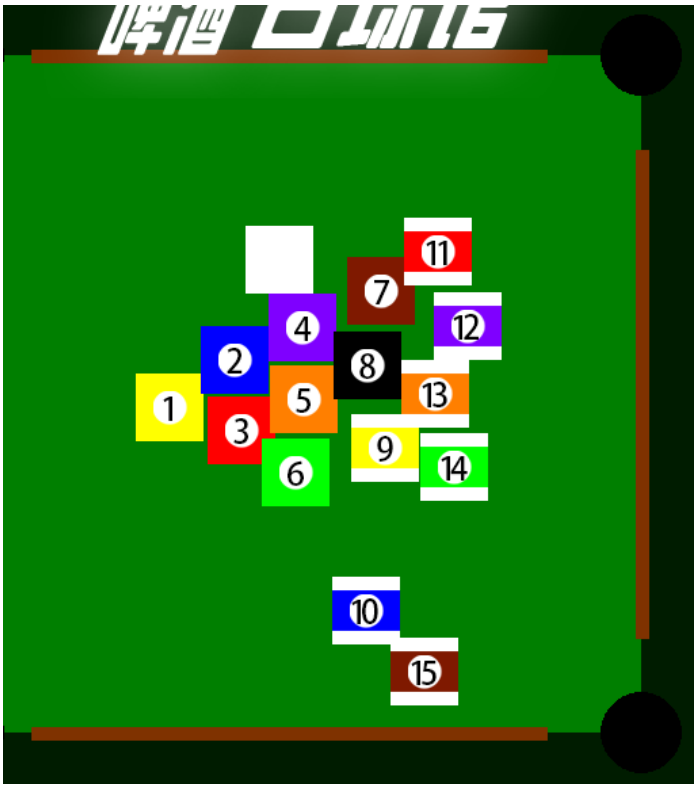
整体设计包括多个类（如 Ball、Wall、BilliardTable 等），每个类负责不同的功能模块，如球的物理属性、台球桌的设置、图形界面的呈现等，确保游戏的流畅性和真实感。

(2)展望

受制于 python 本身机能限制（GIL 全局解释器锁，GPU 接口等），许多预期效果未能实现。如抗锯齿，着色器效果，图像后处理等，未来可以考虑使用游戏引擎（如 Unity Engine，Godot Engine）等重构项目，以获得更好的性能和表现效果。受制于 pymunk 物理引擎本身局限性，以及采用了动态步长策略将导致计算机出现卡顿时物理仿真精度会下降，缺乏前向预测还会导致球体在长步长高速运动时出现 Noclip 现象。维度限制在二维，球体之间的摩擦力和角动量的模拟也未能实现，导致如旋转球，跳球等效果无法实现。未来可以让游戏在三维空间中仿真和渲染，从而增加沉浸感和真实性。

三、问题描述

1. 透明贴图渲染时，透明部分不透明



解决方案：通过 convert_alpha 将其转化为 RGBA 对象

```
self.texture= pygame.image.load("assets/texture/balls/{0:d}.png".format(self.index)).convert_alpha()
```

2. 出现严重的性能问题，操作卡顿延迟高，频繁出现 Noclip 现象

解决方案：

采用双缓冲区技术交替更新，使用更高效的屏幕填充方法。

```
#pygame.display.update()
pygame.display.flip()
#pygame.draw.rect(self.screen, (0, 28, 0, 1), (0, 0, self.screen.get_width(), self.screen.get_height()))
self.screen.fill((0, 28, 0))
```

3. 声音没办法并行播放，且重复播放的声音受到同一个音量控制

解决方案：使用 mixer.Channel，为每个声音事件创建独立的声音通道

```
#self.collision_sound.play()
channel = pygame.mixer.find_channel()
channel.set_volume(speed)
if(isinstance(arbiter.shapes[1],pymunk.shapes.Segment)):
    channel.play(self.collision_side_sound)

else:
    channel.play(self.collision_ball_sound)

return True
```

4.场景缩放时 pygame 默认使用最邻近插值算法，会导致画面扭曲和抖动。

解决方案：

使用 smoothscale，pygame 会使用双线性插值算法进行缩放，可以在不产生过大压力的情况下得到较好的渲染结果

```
self.screen.blit(pygame.transform.scale(self.surface, (screen_w * w_factor, screen_h * h_factor)), (x_p
self.screen.blit(pygame.transform.smoothscale(self.surface, (screen_w * w_factor, screen_h * h_factor))
```