可视化工具

代码分析报告

目录

---

0. 总览与简介

本项目使用 python 内置库 tkinter，对纸带圈地对局进行了可视化；其设计理念为读取运行比赛所用参数，以及 match_core 生成的标准格式的每帧画面，并根据此调用 tkinter.Canvas 相关接口进行可视化显示。

根据不同的应用场景，不同的可视化程序进行了相应的功能与界面扩展。本项目中包 AI 对战与比赛记录读取工具 vusalize.py 与人机对战工具 glory_of_mankind.py 两个文件。运行可视化程序需要包含 tkinter 库的 python3 环境，以及 match_core.py。

0.1. AI 对战与比赛记录读取工具 vusalize.py

可视化工具 visualize.py 集成了 AI 对战与比赛记录 zlog 文件读取的功能，界面元素如下图：

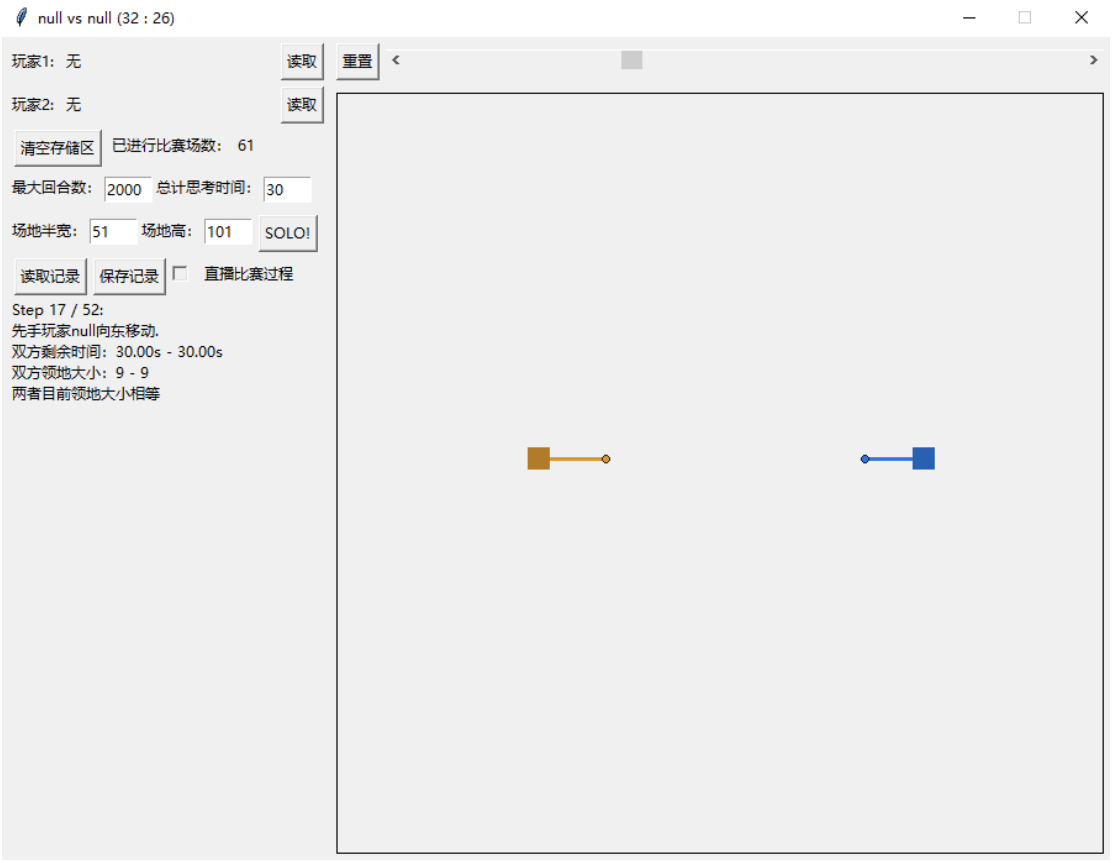

图 1 visualize.py 界面

界面左侧为控制栏，包含比赛双方 AI 选择模块，比赛自定义参数设置模块，比赛记录存取模块，比赛信息等，上述模块会在后文进行详细介绍。“清空存储区”按钮将重置 match_core.STORAGE 变量（即双方 AI 存储区）并重置比赛场数等统计信息；“SOLO!”按钮将启动一场比赛；“直播比赛过程”选框勾选时双方比赛过程内每帧都将直接显示，否则将在比赛结束后整局读取记录。

界面右侧为显示栏。右侧顶部为播放进度条，在读入比赛记录后可用，可以控制播放、暂停、时间定位等；主体部分为可视化的比赛界面。

0.2. 人机对战工具 glory_of_mankind.py

人机对战工具实现了手动控制比赛一方与选定的 AI 模块进行对战的功能，并支持比赛记录的保存，命名“Glory of Mankind”取自游戏《NieR:Automata》。其界面如下图：

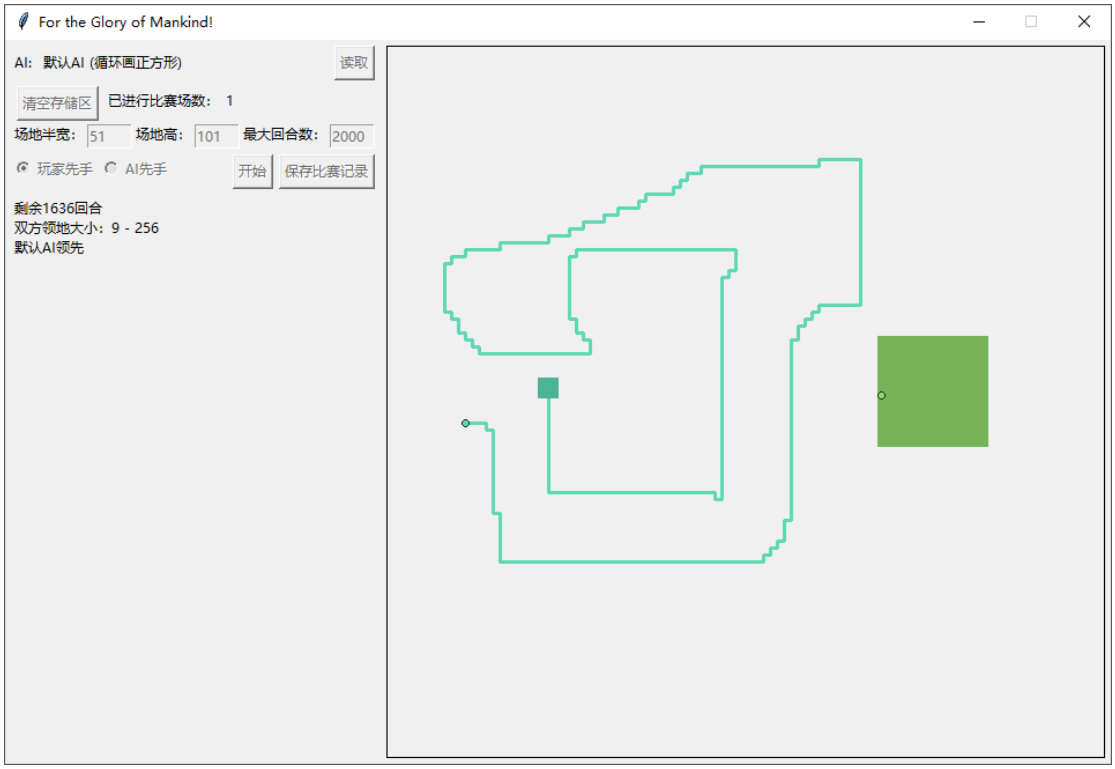

图 2 glory_of_mankind.py 界面

界面左侧为控制栏，包含对战 AI 选择模块、清空存储区等内容，与 visualize.py 类似。由于是人机对战，只需选择一个 AI 模块，并设置人类玩家为先手（左侧）或后手（右侧），另一方将使用特殊的 AI 模块（human_control）并使用“人类”作为名称；此外，由于人类思考时间较长，比赛设置内移除了时间限制并重写了 match_core 内相关模块，人机对战比赛将不会出现一方超时的比赛结果。

界面右侧为显示框，显示了比赛进行（或结束）时的场景。为保证窗口控制非阻塞，比赛与更新画面过程独立于主线程之外，故本程序与 visualize.py 在显示部分有少量区别。

1. 显示模块（display_frame 类）

display_frame 类为封装的帧显示模块，通过 tkinter.Canvas 相关方法显示比赛进程中的场地内容。其中玩家、纸带的显示与领地的显示采用了不同的策略，将详细介绍。

另外，由于涉及到 tkinter.Frame 布局，visualize.py 内的记录查看功能也内置于该类，将在后文相应部分进行介绍；该部分只以 glory_of_mankind.py 中的实现为例，介绍基础的显示设施涉及的对象与过程。

在两个可视化程序中，display_frame 类均实例化为 display 对象，布局于窗口右侧。

1.1. 设计理念与内容

Canvas 对象需要显示的内容包含如下两部分：一是总量、拓扑结构等都不定，但在有限个固定位置内只有固定状态的双方玩家领地，二是每帧最多存在一个的玩家纸带以及只存在一个的玩家本体。

在本项目实际实现中，采用提前布局网格的方法显示领地，每个坐标都放置一个矩形对象，领地所属变动只需调用 Canvas.itemconfig 设置相应坐标矩形的颜色即可；玩家与纸带分别使用椭圆对象与折线对象实现，可通过调用 Canvas.coord 改变其关键点更新位置。

display_frame 类包含的（与显示功能直接相关的）非函数对象如下表：

名称	功能描述
cv	Canvas 对象，用于显示及调用方法更新内容
names	该比赛参与双方名称
colors	根据比赛双方名称 hash 生成的对应领地颜色
bands	双方纸带 canvas 对象对应编号，用于 cv.coord 调用
players	双方玩家 canvas 对象对应编号，用于 cv.coord 调用
size	已生成的比赛场地网格宽高
pixels	记录比赛场地网格内容的列表，用于 cv.itemconfig 调用
last_frame	已渲染的帧内容，用于与新帧对照
default_frame	空白场景帧，在清屏后作为初始的 last_frame

函数对象如下表：

函数名称	接收参数	功能
__init__	窗口控件 root	将对象与 root 连接，并运行_init_screen 函数
_init_screen	无	将控件 cv 布局于 root 内，声明数个初始变量
_setup_grid	网格大小 size	根据输入更新 pixels，并运行_clear 函数
_setup_players	玩家名称 names	根据输入更新 bands 与 players
_clear	无	清除 pixels 内已有内容
_update_screen	当前帧 cur_frame	更新 cv 绘制输入帧内容

1.2. 初始化

显示模块在每次运行（开始比赛或读取比赛记录）时会首先进行初始化，依次运行 display._setup_grid 与 display._setup_players 函数，_setup_grid 内部又会调用_clear 函数。

其中，_setup_grid 函数会检测输入参数 size，在其不等于 self.size 时重新布局 pixels；随后运行_clear 初始化屏幕并运行 root.update 刷新窗口；_clear 函数会根据 last_frame 遍历 pixels，将所有包含内容的领地重设为 root 背景色清空，并将 last_frame 指向 default_frame。

_setup_players 函数在 names 不等于已写入名称时运行，记录名称于 names 变量并删除已有玩家、纸带对象；根据玩家名称 hash 生成主题颜色并记录，随后创建双方纸带与玩家对象并记录编号。

1.3. 领地绘制

领地、玩家、纸带绘制，及文字信息更新均在_update_screen 函数内实现。

在传入帧后显示新的领地状态具有 O(w*h)的复杂度，需要检查并设置场地内每个格点坐标对应矩形为相应颜色，其中双方领地颜色在初始化过程中记录在 colors 变量内，空位置则通过调用 root['bg']获取背景色。

由于更新 canvas 对象需要耗费较多时间，此处采用了缓存加速的策略，将更新前的帧内容保存在 last_frame 对象内，并在遍历网格时将当前帧与上一帧进行对比，只更新颜色有变动的坐标，相比全部更新节省了大量时间。

1.4. 玩家、纸带绘制

玩家通过 canvas 椭圆对象显示，每次更新帧时通过其游戏坐标与显示网格大小计算出椭圆对象在屏幕上实际的左上、右下坐标，并通过 cv.coord 更新。

绘制纸带过程使用了当前玩家坐标，并利用了 match_core 内额外传出的纸带方向序列，从玩家坐标开始回溯获得纸带上每个格点坐标；此外，由于纸带上常出现连续方向，通过记录上一步方向，当接收相同方向时删除对应的中间节点，可以简化纸带关键点数以加速。

1.5. 文字信息显示

在遍历检查领地的过程中，_update_screen 函数同时对比赛双方的总领地面积进行了计数，并将计数结果与其余所需的内容传递给生成文字信息的代码；后者将根据双方玩家名称、方向、占地等生成对当前局势的描述，并显示于窗口左侧下部的信息栏。

2. 通用功能模块

2.1. AI 读取模块（AI_selection 类）

AI_selection 类集成了玩家 AI 模块的读取与合法性检测，以及在 tkinter 窗口中布局的功能。初始化 AI_selection 对象时需要指定其父 tkinter 控件，与显示的名称（AI，玩家 1，玩家 2 等）；通过访问每个 AI_selection 实例的 AI_MODULE 与 AI_NAME 参数可以获得其成功读取的 AI 模块与对应名称。

其包含的 tkinter 控件包含可变显示模块名称的 Label 对象，与读取 AI 模块使用的按钮。点击按钮将调用 tkinter.filedialog.askopenfilename 获取 py 文件的路径，尝试读取其内容并使用 exec 函数声明于一个类内部，并将该类写入 AI_MODULE 参数。在成功读取后，AI_NAME 参数与相应的显示也将更新，同时将重新初始化 match_core 内部已有的存储内容。

2.2. 比赛参数设置（checked_entry 类）

checked_entry 类为对 tkinter.Entry 的一个封装，执行从中获取合法输入文字，自动检查合法性与转化为相应类型的功能，用于设置比赛参数。

初始化 checked_entry 对象需要指定父控件 root，输入数值类型 type（int，float 等），默认值 default 与该输入框描述 text。初始化函数会排列描述的 Label 与输入框 Entry 进父控件，并自动为 Entry 绑定合法性检查事件 ,通过调用 get 函数可以获取其中已经输入的数据。

2.3. 运行比赛（run_match 函数）

run_match 函数运行时会首先从几个输入框获取设置的双方参赛者模块与比赛参数，并更新一些简单的统计信息；随后向显示模块传入比赛参数与选手名称进行场地初始化，最后开始比赛。在比赛过程中使用了“控件锁”的机制防止误触造成逻辑错误。比赛结果将被记录于全局变量。

glory_of_mankind.py 在导入 match_core 时通过重写其 timer 函数移除了超时判定；此外，为防止 match_core 运行阻塞主窗口按键输入等，未将 match_core.match 函数直接在主线程中运行，而是放置于独立线程内。

2.4. 即时显示

即时显示功能应用于 visualize.py 勾选“直播比赛过程后”，以及 glory_of_mankind.py 全程。程序内通过将帧输出接口 match_core.FRAME_FUNC 指向 display._update_screen，实现即时显示功能。

visualize.py 在比赛运行期间是完全阻塞的，需要在 FRAME_FUNC 内增加调用 tk.update 以手动更新显示，封装为 update_hook 函数。

由于 glory_of_mankind.py 中调用帧输出接口也在外部线程内，在大块圈地时从主线程可以看到逐行遍历的“上色”过程。

2.5. 控件锁

窗口内所有可操作的控件，包括输入框、按钮等，均被加入全局内名为 OP_WIDGETS 的列表，用于统一启用/关闭控件。widget_off 函数在每场比赛开始前运行，遍历列表内控件并将其 state 设置为 DISABLED； widget_on 函数在比赛结束后运行，遍历列表内控件，由于 Entry 与 Button 启用状态不同，其通过 try except 结构启用每个控件。

visualize.py 内进度条 ScrollBar 控件无 state 属性，其通过在拖动事件中判断比赛是否正在运行来决定是否执行功能。

2.6. 记录保存

保存记录时调用了 tkinter.filedialog.asksaveasfilename 获取合法的保存路径，并将全局变量内的比赛记录对象使用 pickle 序列化后使用 zlib 进行压缩，并写入保存路径。

2.7. 垃圾清理

由于比赛记录文件占用大量的内存，为防止自动内存清理不及时造成内存溢出，在启动比赛、读取比赛记录等关键操作前会调用 gc.collect 强制回收内存。

3. 独有功能模块

3.1. 记录读取与进度条（visualize.py）

相比人机对战程序, visualize.py 的显示模块内增加了对查看比赛记录的支持, 在 Canvas 对象上方增加了播放控制按钮与滚动条

显示模块中新增的相关对象如下表：

名称	功能描述
playing_status	指示播放状态，0 为暂停/停止，1 为播放
frame_seq	帧序列，由 load_match_result 函数获取
frame_index	播放进度，指示 frame_seq 下标
match_result	比赛结果元组，由 load_match_result 函数获取

显示模块新增的相关函数如下表：

函数名称	接收参数	功能
button1_press	无	按钮事件；根据当前播放状态决定播放/暂停/返回第一帧（已播放至末尾）
scroll_option	可变参数*args	滚动条事件，包括滑块与前后按钮；只在非比赛运行且帧序列多于 1 帧时执行，运行时暂停播放事件，更新播放进度，渲染新的帧并更新滚动条位置
scroll_update	无	根据播放进度与帧序列总长度确定滚动条位置
update	无	播放事件；只在 playing_status 为 1 时运行，增加播放进度并渲染对应帧，若播放至末尾则自动停止, 同时通过调用 root.after 并传入 update 函数实现循环事件
load_match_result	比赛记录 log 初始化标志 init	从 log 读取比赛结果、帧序列，初始化播放进度与相关控件，在 init 为真（加载比赛记录）时读取比赛参数进行初始化，并渲染第一帧
_update_info	当前帧 frame 计数 field_count	封装的回合信息更新函数 ;根据比赛是否运行决定显示剩余回合数或播放进度, 其余类似人机对战程序

另外，可视化工具中为读取比赛记录功能新增了 load_log 函数以读取比赛记录文件；读取的对象将被写入全局变量并传递给

3.2. 人机对战（glory_of_mankind.py）

人机对战程序中，人类方 AI 模块为特殊的 human_control 类，可以接收的鼠标键盘输入控制玩家行动，同时也执行了帧率控制的功能。其内部的 control 参数指示了当前采用的控制模式（默认为键盘“key”）。

a. 键盘控制

键盘控制模式为自动逐帧行进，涉及到的类变量有指示方向 op，延时设置 delay 与延时控制变量 old_timer，全局变量有键位对相应方向的映射 key_mapping。tkinter 窗口中绑定了按键事件 key_control 函数。

key_control 函数读取按键事件的 keysym 并在 key_mapping 内查找，若该键存在（方向键或 WASD）则将控制模式设置为“key”并传入相应方向至 human_control.op。

在 human_control 内，load 函数将 old_timer 参数初始化为当前时间。play 函数在控制模式为“key”时，首先进行帧率控制，在当前时间距 old_timer 小于 delay 时 sleep 相应时间差，随后刷新 old_timer 至当前时间；随后，若指示方向存在，则通过计算指示方向与玩家当前方向之差决定输出转向方向，并将使用过的指示方向清空。

b. 鼠标控制

鼠标控制模式为阻塞模式，涉及的类变量有目标位置 pos 及鼠标状态 mouse_holding，并在显示窗口 display.cv 上绑定了鼠标左键按下事件 mouse_down，抬起事件 mouse_up 与拖动事件 set_position。

mouse_down 函数将控制模式切换为“mouse”，并将 mouse_holding 设为 True，随后执行一次 set_position 函数；mouse_up 函数将 mouse_holding 设为 False；set_position 函数通过显示参数计算出鼠标点选位置的游戏坐标并传入 pos 参数。

human_control.play 函数内，当控制模式为“mouse”时，首先在鼠标抬起或点选位置与玩家位置重合时保持阻塞（若中途切换为键盘控制则递归调用一次自身跳出循环），随后根据当前玩家方向分情况讨论决定前往点选位置的前进方向，以实现只在按下鼠标时前进的功能。