

清华大学云盘链接: <https://cloud.tsinghua.edu.cn/library/a8288c8f-d740-4117-b773-5a1a9ecfc0a0/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1/>
Github 链接: <https://github.com/Sky-Trigger/Object-Oriented-Technology>

面向对象大作业 实验报告 棋类对战平台

一、设计思路与选用的设计模式

1.1 总体架构设计

系统分为三层：核心层(core)管理游戏状态和规则，游戏层(games)实现五子棋和围棋规则，界面层(ui)提供 CLI 和 GUI。分层分离逻辑和界面，便于维护扩展。

1.2 设计模式应用

- (1) 模板方法模式：GameEngine 基类定义游戏流程，子类实现具体规则，便于扩展新游戏。
- (2) 工厂方法模式：create_engine()根据游戏类型创建引擎，解耦客户端代码。
- (3) 策略模式：UI 层支持 CLI 和 GUI 两种界面，交互方式不同但逻辑统一。
- (4) 命令模式：CLI 封装命令为对象，便于添加和管理命令。

1.3 面向对象设计原则体现

- (1) 单一职责：各类职责明确，如 Board 管棋盘，GameController 协调。
- (2) 开闭原则：新增游戏只需继承 GameEngine，无需修改现有代码。
- (3) 里氏替换：子类可替换父类，接口一致。
- (4) 依赖倒置：界面通过 GameController 依赖抽象，而非具体引擎。
- (5) 接口隔离：GameEngine 接口精简，各客户端按需使用。

1.4 关键设计决策

- (1) 资源管理：追踪棋子使用，支持显示和扩展规则。
- (2) 悔棋限制：每方最多 3 次悔棋，避免滥用。
- (3) 序列化：完整保存状态，使用 JSON 格式，支持兼容性。
- (4) 异常处理：业务错误抛 ValueError，前端友好提示。

二、关键类和函数的简单说明

2.1 核心层(core/)

[Board 类]core/board.py

功能：管理棋盘

主要方法：__init__(size), get/set(position), is_within_bounds(), reset(), serialize()/deserialize()

实现：二维列表存储，边界检查，支持 JSON 序列化

[GameEngine 类]core/game_engine.py

功能: 游戏引擎基类

主要属性: board, current_player, history, captured_by_color, max_undo

主要方法: play_move()(抽象), pass_turn(), undo(), resign(), restart(), is_finished(), serialize()/deserialize(), stones_remaining(), undo_remaining()

实现: 模板方法模式, 记录资源数据, 完整状态保存

[GameController 类]core/controller.py

功能: 协调各组件

关键方法: start_game(), place_stone(), pass_turn(), undo(), resign(), restart(), save/load(), get_board_display(), get_status(), get_resource_snapshot()

亮点: 统一接口, 丰富查询, 中文化显示

[Models 模块]core/models.py

功能: 数据模型和枚举

关键类: GameType, PlayerColor, Position, Move, GameResult

亮点: 枚举类型安全, Position 支持哈希, Move 记录提子

[Persistence 模块]core/persistence.py

功能: 存档保存加载

关键函数: save_game(), load_game()

亮点: 自动创建目录, UTF-8 编码, 统一异常处理

2.2 游戏层(games/)

[GomokuEngine 类]games/gomoku.py

功能: 五子棋引擎

主要方法: play_move(), _check_five_in_row(), _count_in_direction(), _undo_internal()

判断输赢: 检查四个方向连子 ≥ 5

[GoEngine 类]games/go.py

功能: 围棋引擎

主要方法: play_move(), pass_turn(), _capture_adjacent_groups(), _has_liberty(), _collect_group(), _score_game(), _compute_territory(), _undo_internal()

实现: 栈遍历棋组, 提子判断, 禁入点检测, 数子+提子+地盘算分

2.3 界面层(ui/)

[ConsoleClient 类]ui/cli.py

功能: 命令行界面

主要方法: run(), _register_commands(), _cmd_*()系列

支持命令: start, move, pass, undo, resign, restart, save/load, board, status, hint, gui, help, exit

实现: 命令模式, 中文提示, 自动 help

[GuiApp 类]ui/gui.py

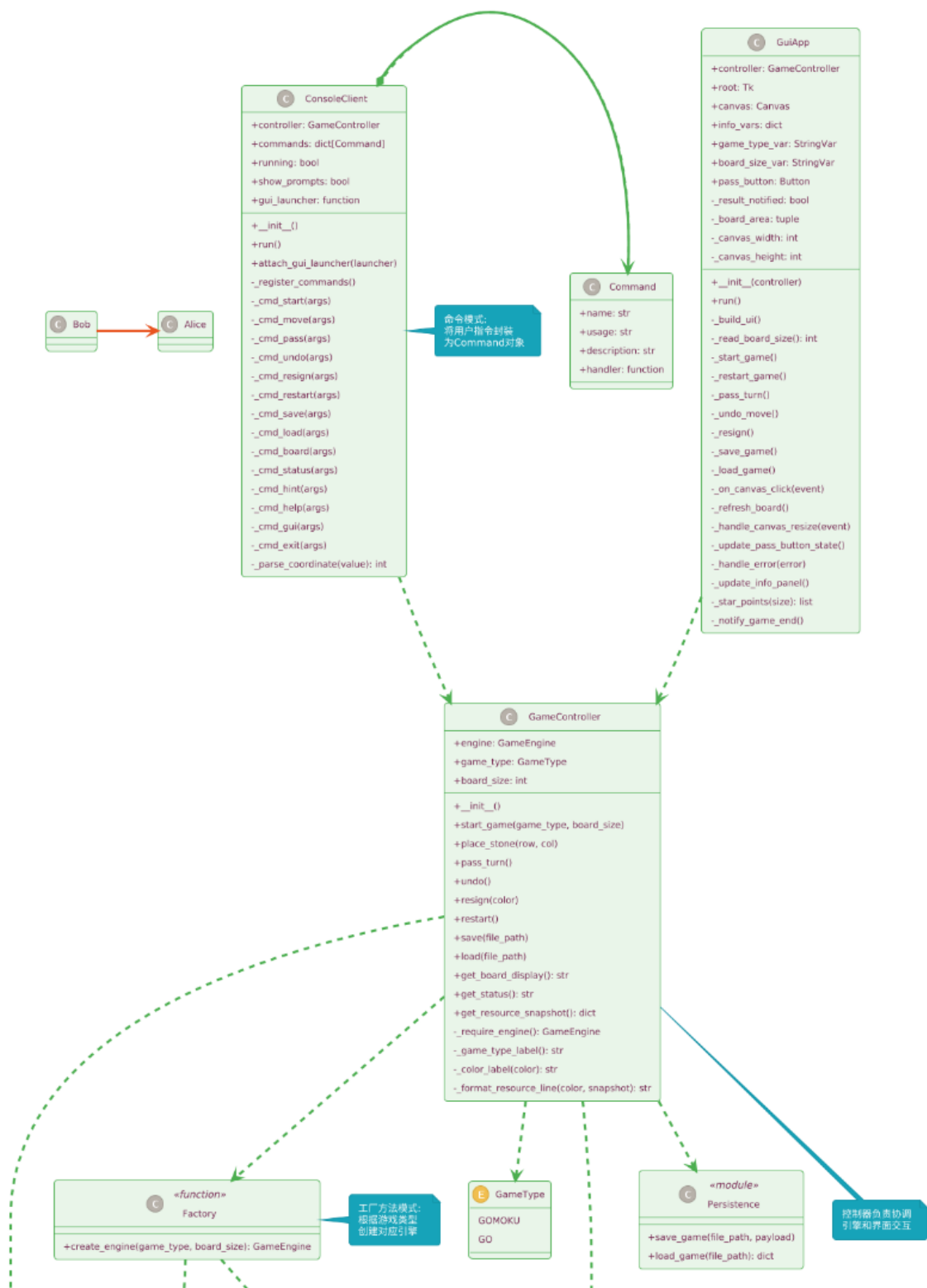
功能: 图形界面

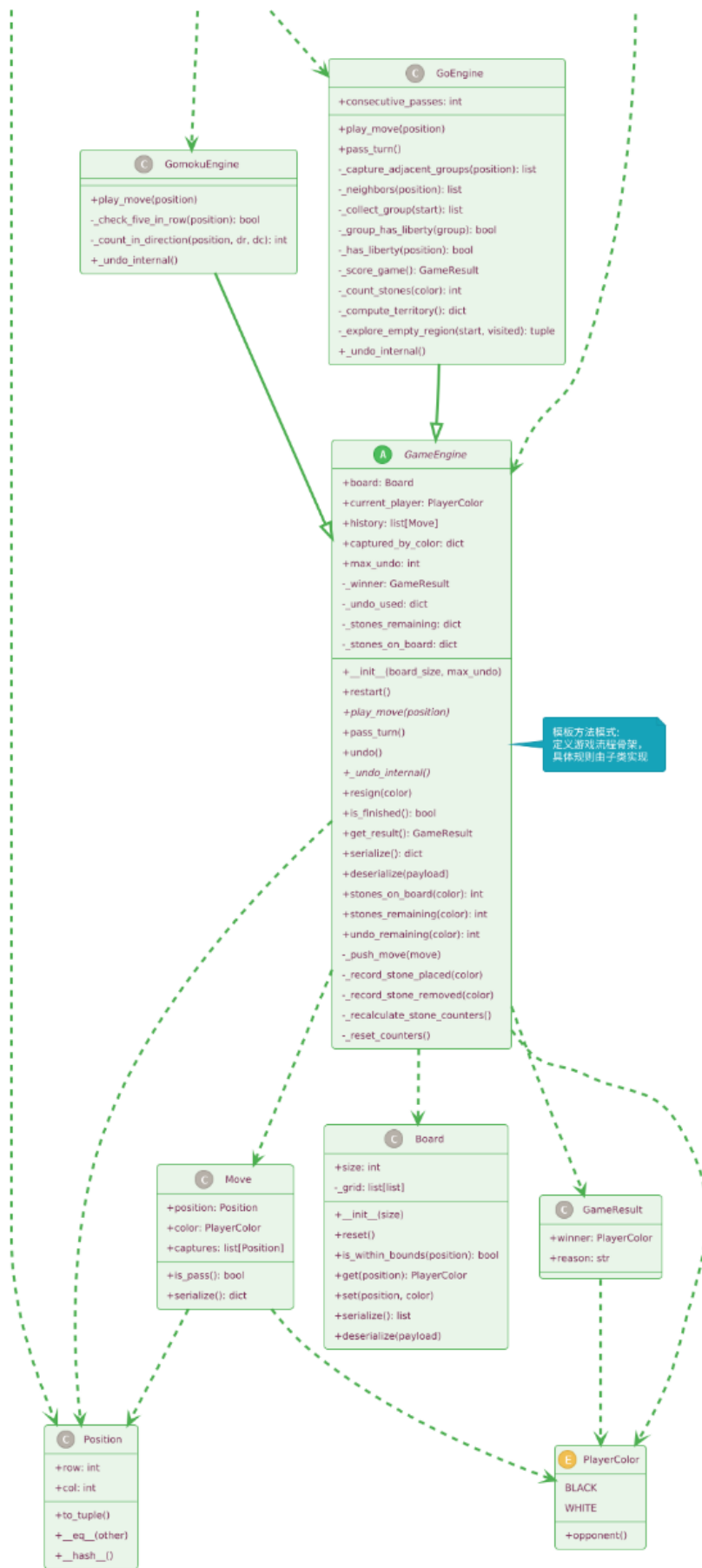
主 要 方 法 : `__init__()`, `_build_ui()`, `_start_game()`, `_on_canvas_click()`, `_refresh_board()`,
`_update_info_panel()`, `_notify_game_end()`, `_star_points()`

界面: 控制面板、信息面板、棋盘画布

特点: 响应式布局, 自动星位, 彩色显示, 结束弹窗

三、UML 类图





四、测试用例与程序测试过程的输入输出

4.1 五子棋功能测试

[测试用例 1: 正常对局]

输入命令: > start gomoku 15, move 8 8, move 8 9, move 9 8, move 9 9, move 10 8, move 10 9, move 11 8, move 11 9, move 12 8, board, status

期望输出: 15×15 棋盘, 黑白交替, 第 8-12 行第 8 列连 5 黑子, 黑胜, 状态显示胜者。

实际输出: ✓ 符合预期

[测试用例 2: 悔棋]

输入命令: > start gomoku 9, move 5 5, move 5 6, undo, board

期望输出: 棋盘仅(5,5)黑子, 白方行棋, 白悔棋余量减 1。

实际输出: ✓ 符合预期

[测试用例 3: 悔棋限制]

输入命令: > start gomoku 9, move 1 1, move 2 2, undo, move 3 3, undo, move 4 4, undo, move 5 5, undo

期望输出: 前 3 次悔棋成功, 第 4 次提示次数用尽。

实际输出: ✓ 符合预期

[测试用例 4: 平局]

输入命令: (9×9 棋盘下满 81 步无五连)

期望输出: 最后一步判定平局, 状态显示平局。

实际输出: ✓ 符合预期

[测试用例 5: 非法落子]

输入命令: > start gomoku 9, move 5 5, move 5 5

期望输出: 提示位置已有棋子。

实际输出: ✓ 符合预期

输入命令: > move 10 10

期望输出: 提示超出范围。

实际输出: ✓ 符合预期

4.2 围棋功能测试

[测试用例 6: 提子]

输入命令: > start go 9, move 5 5, move 5 6, move 6 5, move 4 6, move 5 4, move 6 6, move 4 5, move 5 7, board

期望输出: (5,6)空, 黑提子+1, 白棋子-1。

实际输出: ✓ 符合预期

[测试用例 7: 禁入点]

输入命令: > start go 9, move 5 5, move 5 4, move 6 5, move 5 6, move 4 6, move 4 5, move 6 6, move 6 4, move 5 5

期望输出: 提示禁入点, 无气。

实际输出: ✓ 符合预期

[测试用例 8: 虚手胜负]

输入命令: > start go 9, move 3 3, move 6 6, pass, pass, status

期望输出: 两次虚手结束, 计算得分, 显示胜者和比分。

实际输出: ✓ 符合预期

[测试用例 9: 悔棋恢复提子]

输入命令: > start go 9, move 5 5, move 5 6, move 6 5, move 4 6, move 5 4, move 6 6, move 4 5, move 5 7, undo, board

期望输出: (5,6)恢复白子, 黑提子归 0, 黑方行棋。

实际输出: ✓ 符合预期

4.3 存档加载测试

[测试用例 10: 保存加载]

输入命令: > start gomoku 13, move 7 7, move 7 8, move 8 7, save test_game.json, restart, board, load test_game.json, board

期望输出: 保存成功, restart 后空, 加载后恢复状态。

实际输出: ✓ 符合预期

[测试用例 11: 加载不存在文件]

输入命令: > load nonexistent.json

期望输出: 提示文件不存在。

实际输出: ✓ 符合预期

4.4 GUI 界面测试

[测试用例 12: GUI 交互]

操作: CLI 输入"gui"启动, 选择五子棋 15 路, 开始游戏, 落子, 悔棋, 保存, 加载。

期望: 窗口正常, 棋盘居中带星位, 信息更新, 交互正常。

实际: ✓ 符合预期

[测试用例 13: 响应式布局]

操作: 启动 GUI, 开始 9 路围棋, 调整窗口大小。

期望: 棋盘居中缩放, 比例不变, 点击准确。

实际: ✓ 符合预期

[测试用例 14: 结束弹窗]

操作: GUI 中下五子棋, 出五连。

期望: 弹出结束对话框, 显示胜者, 只弹一次。

实际: ✓ 符合预期

4.5 异常处理测试

[测试用例 15: 输入验证]

输入命令: > start gomoku 5, start gomoku 25, start xxx 15, move abc def, move -1 5

期望输出: 提示大小范围、类型无效、坐标错误。

实际输出: ✓ 符合预期

[测试用例 16: 状态检查]

输入命令: > move 5 5

期望输出: 提示无对局。

实际输出: ✓ 符合预期

输入命令: > start gomoku 9, pass

期望输出: 提示仅围棋可虚手。

实际输出: ✓ 符合预期

4.6 性能测试

[测试用例 17: 大棋盘性能]

输入: > start go 19, (100 步), board, status

期望: 响应<100ms, 内存稳定, 显示流畅。

实际: ✓ 符合预期

[测试用例 18: 序列化性能]

输入: > start go 19, (200 步), save large_game.json

期望: 保存<500ms, 文件~50KB, 加载快。

实际: ✓ 符合预期