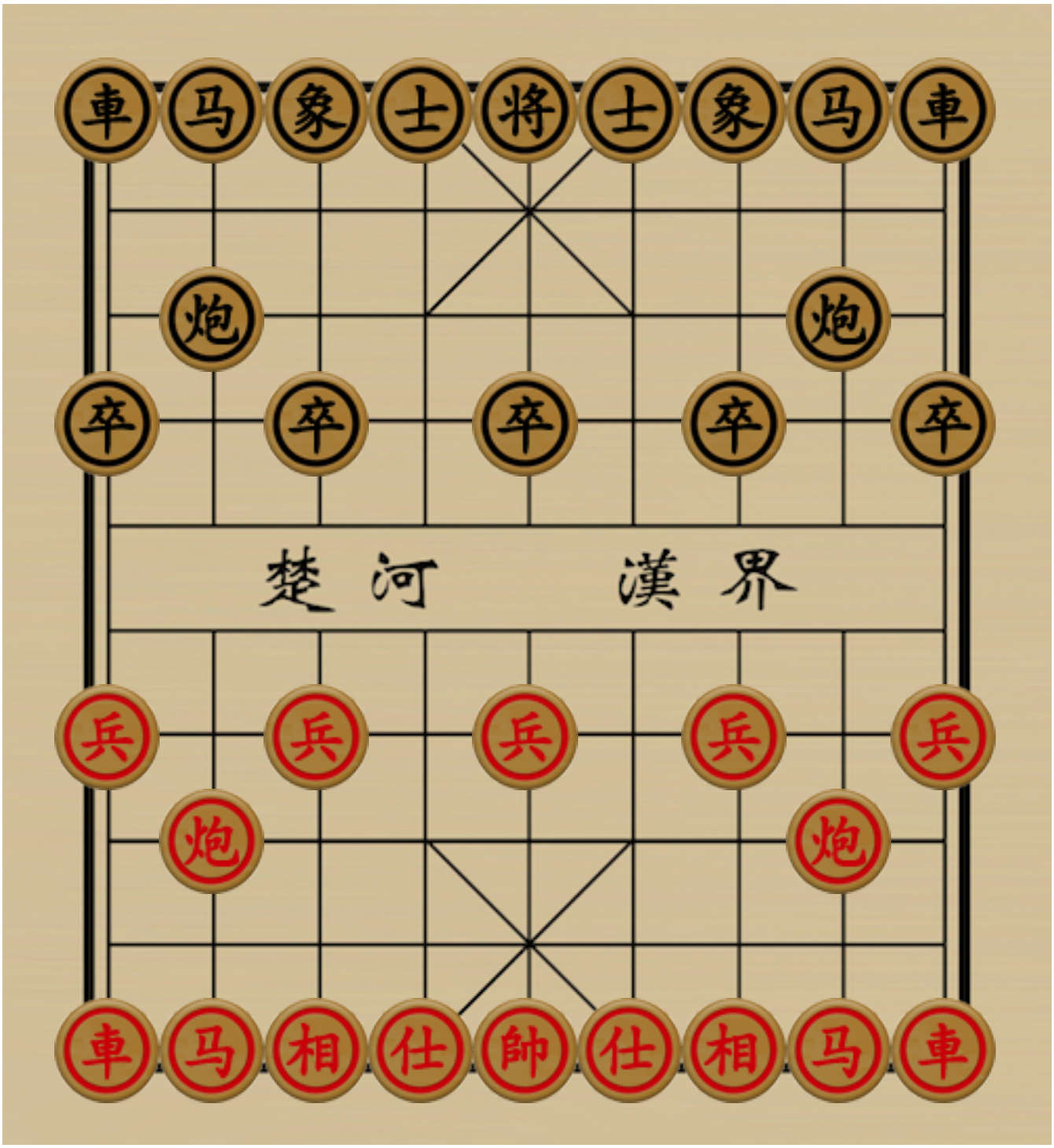
桌游类小游戏开发框架

南方小智

2020 年 8 月 8 日

目录


車 马 象 士 将 士 象 马 車
炮 炮
卒 卒 卒 卒 卒
楚 河 漢 界
兵 兵 兵 兵 兵
炮 炮
車 马 相 仕 帥 仕 相 马 車

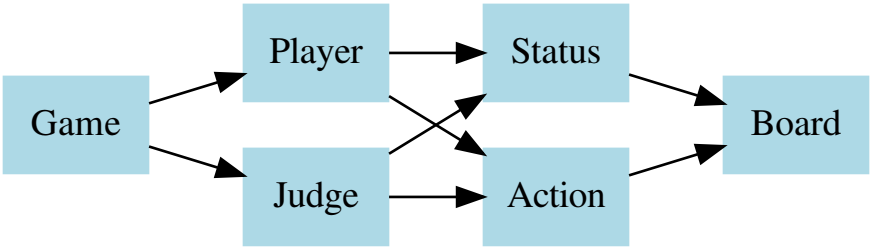
第一章　概述

本项目用于实现各类桌游小游戏的游戏流程，AI 玩家，对弈平台等。目前正在开发中国象棋的基本游戏流程和 AI 算法。

1.1　安装与设置

```
python3 -m pip install termcolor
```

第二章　基本元素



桌游小游戏基本元素

游戏中的基本元素有，Board，Status，Action，Player，Judge，Game 等。外部程序通过设置 Game，Players，和 Config，可以调用该框架。

2.1　Board

游戏中最基本的元素是 Board，相当与棋盘，牌桌。其中包含各类游戏物件以及他们的位置，比如棋子，纸牌。

Board 提供对这些游戏物件的访问和改变，比如改变某个物件的位置。同时也对 Board 上的基本状态进行检测，比如在中国象棋中判断当前局面上将帅是否照面。

2.2 Action

Action 代表某个玩家可以在 Board 上的一个操作。比如把某个子移动到另一个地方，称为 MoveAction。可以通过继承 Action 实现更高级的动作，比如悔棋操作。MoveAction 应当是可以撤销的（roll_back），这样可以支持悔棋的功能，以及在 AI 搜索算法中可以利用其实现回溯。

认输也可以是一个 Action。

2.3 Status

当前游戏的所有状态，包括 Board 的格局，当前玩家，获胜玩家，Action 的历史栈等。获得 Status 就可以知道游戏从初始到现在所有的需要的状态数据，也就是可以通过 Status 对整个游戏过程进行复盘。

2.4 Player

Player 的主要任务是，在自己为主的一轮当中，通过当前的 Status，按照顺序作出一个或多个 Action。

Player 可以是自动的 AIPlayer，也可以接收输入的真人玩家，网络玩家等。

2.5 Judge

Judge 负责判定 Player 产生的 Action 是否合法，并最终负责执行合法的 Action。Judge 还负责判断游戏是否已经结束，谁是胜利玩家等。Judge 通过 Rule 来进行这些判定，不同的 Rule 集合可以产生略微不同的游戏规则，比如，可以在象棋中去掉将帅不可照面的规则等。

某些特殊的中国象棋残局添加了额外的限定，这种 Judge 和 Rule 解耦分开处理的方式有利于实现这些残局游戏。

2.6 Game

Game 控制整个游戏的流程，每个游戏由准备阶段开始，然后经过若干轮，每轮以其中一位 Player 为主，并由 Judge 执行操作。最后利用 Judge 判断游戏结束和得出胜利玩家列表。

```python
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import abc
from typing import List


####################################
# 基本元素
####################################

# TODO: Items

class Action():
    def __init__(self):
        self.is_reversible = False


class Board():
    def __init__(self, name, level:int = 1):
        self.name = name
        self.set_level(level)

    @abc.abstractmethod
    def prepare(self) -> bool:
        self.print_info("Preparing")
        return True

    def set_config(self, config):
        self.config = config

    def set_level(self, level):
```

```
32          self.level = level
33          self.prefix = "\t" * level
34
35      def print_info(self, line: str):
36          if self.config.silent_mode:
37              return
38          print(f"{self.prefix}[{self.name}]: {line}")
39
40      @abc.abstractmethod
41      def print(self):
42          pass
43
44
45  class GameConfig():
46      def __init__(self, name, silent_mode: bool = False):
47          self.name = name
48          self.silent_mode = silent_mode
49
50
51  class GameStatus():
52      def __init__(self, board: Board, current_player_id: int):
53          self.board = board
54          self.action_stack = []
55          self.current_player_id = current_player_id
56          self.game_end = False
57          self.winner_names = []
58          self.turns_count = 0
59
60
61      def push(self, action: Action):
62          self.action_stack.append(action)
63
64      def switch(self, player_id: int):
65          self.current_player_id = player_id
66
67
68  class Player():
```

```
69      def __init__(self, name, level:int = 1):
70          self.name = name
71          self.set_level(level)
72
73      @abc.abstractmethod
74      def prepare(self) -> bool:
75          self.print_info("Preparing")
76          return True
77
78      def set_config(self, config):
79          self.config = config
80
81      @abc.abstractmethod
82      def play(self, status: GameStatus) -> Action:
83          pass
84
85      def set_level(self, level):
86          self.level = level
87          self.prefix = "\t" * level
88
89      def print_info(self, line: str):
90          if self.config.silent_mode:
91              return
92          print(f"{self.prefix}[{self.name}]: {line}")
93
94      def __str__(self):
95          return self.name
96
97
98  class Judge():
99      def __init__(self, config: GameConfig, name: str, level: int = 1):
100         self.config = config
101         self.name = name
102         self.set_level(level)
103
104     def set_level(self, level):
105         self.level = level
```

```
        self.prefix = "\t" * level

    def print_info(self, line: str):
        if self.config.silent_mode:
            return
        print(f"{self.prefix}[{self.name}]: {line}")

    @abc.abstractmethod
    def check_end(self, status: GameStatus, players: List[Player]) -> bool:
        pass


    @abc.abstractmethod
    def validate_action(self, action: Action, status: GameStatus) -> bool:
        pass

    # Return true is this the last Action of the turn
    # Help current player to run the action under the current status
    # Assume the action has been validated.
    @abc.abstractmethod
    def run(self, player: Player, action: Action, status: GameStatus) -> bool:
        pass


# 充当游戏流程与裁判的角色
class BoardGame():
    def __init__(
        self,
        players: List[Player],
        board: Board,
        config: GameConfig,
        level:int = 0
    ):
        self.name = config.name

        assert len(players) > 0, f"MUST have at least 1 player"
        self.players = players
```

```
        for player in self.players:
            player.set_level(level + 1)

        self.board = board
        self.config = config
        self.set_level(level)

    def set_level(self, level):
        self.level = level
        self.prefix = "\t" * level

    def print_info(self, line: str):
        if self.config.silent_mode:
            return
        print(f"{self.prefix}[{self.name}]: {line}")


    @abc.abstractmethod
    def init_status(self) -> GameStatus:
        pass

    @abc.abstractmethod
    def init_judge(self) -> Judge:
        pass

    @abc.abstractmethod
    def check_end(self) -> bool:
        pass

    def prepare(self) -> bool:
        self.board.set_config(self.config)
        self.board.prepare()
        for player in self.players:
            player.set_config(self.config)
            if player.prepare() is False:
                return False
        self.status = self.init_status()
```

```
180         self.judge = self.init_judge()
181         if self.config.silent_mode is False:
182             self.board.print()
183 
184         return True
185 
186     # TODO: Move to Judge
187     @abc.abstractmethod
188     def next_player(self) -> None:
189         pass
190 
191     def turn(self) -> None:
192         turn_end = False
193         current_player = self.players[self.status.current_player_id]
194         while(turn_end is False):
195             action = current_player.play(self.status)
196             # TODO: handle invalid steps
197             turn_end = self.judge.run(current_player, action, self.status)
198 
199     # return list of winner
200     @abc.abstractmethod
201     def result(self) -> List[Player]:
202         pass
203 
204     def start(self):
205         self.print_info("游戏开始")
206         self.prepare()
207 
208         # 每一轮的定义：从决定下一个player开始，到该player进行操作，最后到就判
    断游戏是否结束。
209         while(self.judge.check_end(self.status, self.players) is False):
210             self.print_info(f"第{self.status.turns_count+1}轮")
211             # for next turn
212             if self.status.turns_count > 0:
213                 self.next_player()
214             self.turn()
215             self.status.turns_count += 1
```

```
216
217         self.result()
218         self.print_info("游戏结束")
219
220
221 ###################################
222 # AI设计
223 ###################################
224
225
226 class StatusEvaluator():
227     def __init__(
228         self,
229         # level:int = 0
230     ):
231         pass
232
233     # Return value between [0, 1], 1 is the best
234     @abc.abstractmethod
235     def evaluateBoard(self, board: Board) -> float:
236         pass
```

第三章　AI 设计

AIPlayer 可以实现自动的游戏过程，一些通用的 AIPlayer 可以被不同的游戏重复利用。

3.1　AIPlayer

最基本的 AIPlayer 是随机的 AIPlayer。首先根据规则，获取所有可能的 Action，然后在其中随机选择一个 Action 返回。

3.2　Evaluator

Evaluator 是对当前 Board 局面的一个评估函数，返回一个 [0, 1] 的值，0 代表是最糟糕的局面，1 代表是最好的局面。

通过 Evaluator，我们可以获得稍好于随机操作的 AIPlayer。同样的，可以首先根据规则，获取所有可能的 Action。然后执行每个 Action，对新的局面调用评估函数，可以得出执行哪一个 Action 会得到对自己最有利的局面，然后返回这个 Action。

3.3　MaxMinAIPlayer

通过极大极小搜索和 AlphaBeta 剪枝实现多步的搜索。同样的会在搜索的尾部，调用 Evaluator 对局面进行评估。目前的单机搜索深度能够达到 4 到 5 层。

第四章　对弈平台

目前尚未开始这个阶段的工作。

第五章　TODO

- 基础设施
  - 学习和参照中国象棋标准，制定静态棋盘和动态棋局的文件格式。UCCI 中国象棋通用引擎协议　版本：3.0
  - 借助 latex 项目，实现静态棋盘的图片的生成，以及实现动态棋局 gif 图的生成。
- AI 设计
  - 实现 AI 的自动博弈，并计算每次结局的基本情况和整体胜率。
  - 实现训练框架对评估函数进行训练。
  - 将 MaxMinAIPlayer 通用化，使得在五子棋等其他游戏中也可以直接复用。
  - 学习通用的象棋 AI 算法，并制定下一步计划。
- 平台化
  - UCCI 中国象棋通用引擎协议　版本：3.0
  - 制作游戏 UI。
  - 参加中国象棋在线比赛，构建中国象棋在线比赛平台。
- Better Engineering
  - 完善 Readme 文档
  - 将项目添加到 git 上进行代码管理。
  - Modify all assert to exception