实例 2：井字棋

井字棋是一种在 3 * 3 格子上进行的连珠游戏，又称井字游戏。井字棋的游戏有两名玩家，其中一个玩家画圈，另一个玩家画叉，轮流在 3 * 3 格子上画上自己的符号，最先在横向、纵向、或斜线方向连成一条线的人为胜利方。如图 1 所示为画圈的一方为胜利者。

图 1 井字棋

本实例要求编写程序，实现具有人机交互功能的井字棋。

实例目标

- 理解面向对象的思想
- 能独立设计类
- 掌握类的继承和父类方法的重写

实例分析

根据实例描述的井字棋游戏的规则，下面模拟一次游戏的流程如图 2 所示。

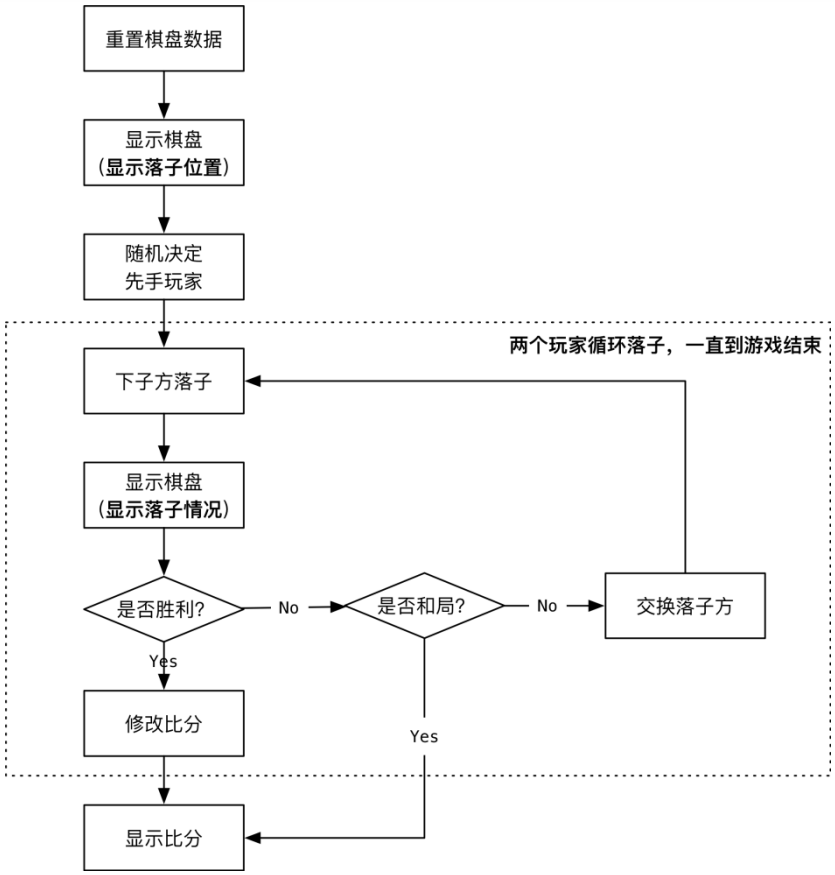

图 2 井字棋游戏流程

图 2 中的描述的游戏流程如下：

（1）重置棋盘数据，清理之前一轮的对局数据，为本轮对局做好准备。

（2）显示棋盘上每个格子的编号，让玩家熟悉落子位置。

（3）根据系统随机产生的结果确定先手玩家（先手使用 X）。

（4）当前落子一方落子。

（5）显示落子后的棋盘。

（6）判断落子一方是否胜利？若落子一方取得胜利，修改玩家得分，本轮对局结束，跳转至第（9）步。

（7）判断是否和棋？若出现和棋，本轮对局结束，跳转至第（9）步。

（8）交换落子方，跳转至第（4）步，继续本轮游戏。

（9）显示玩家当前对局比分。

以上流程中，落子是游戏中的核心功能，如何落子则是体现电脑智能的关键步骤，实现智能落子有策略可循的。按照井字棋的游戏规则：当玩家每次落子后，玩家的棋子在棋盘的水平、垂直或者对角线任一方向连成一条直线，则表示玩家获胜。因此，我们可以将电脑的落子位置按照优先级分成以下三种：

（1）必胜落子位置

我方在该位置落子会获胜。一旦出现这种情况，显然应该毫不犹豫在这个位置落子。

（2）必救落子位置

对方在该位置落子会获胜。如果我方暂时没有必胜落子位置，那么应该在必救落子位置落子，以阻止对方获胜。

（3）评估子力价值

评估子力价值，就是如果在该位置落子获胜的几率越高，子力价值就越大；获胜的几率越低，子力价值就越小。

如果当前的棋盘上，既没有必胜落子位置，也没有必救落子位置，那么就应该针对棋盘上的每一个空白位置寻找子力价值最高的位置落子。

要编写一个评估子力价值的程序，需要考虑诸多因素，这里我们选择了一种简单评估子力价值的方式——只考虑某个位置在空棋盘上的价值，而不考虑已有棋子以及落子之后的盘面变化。下面来看一下在空棋盘上不同位置落子的示意图，如图 3 所示。

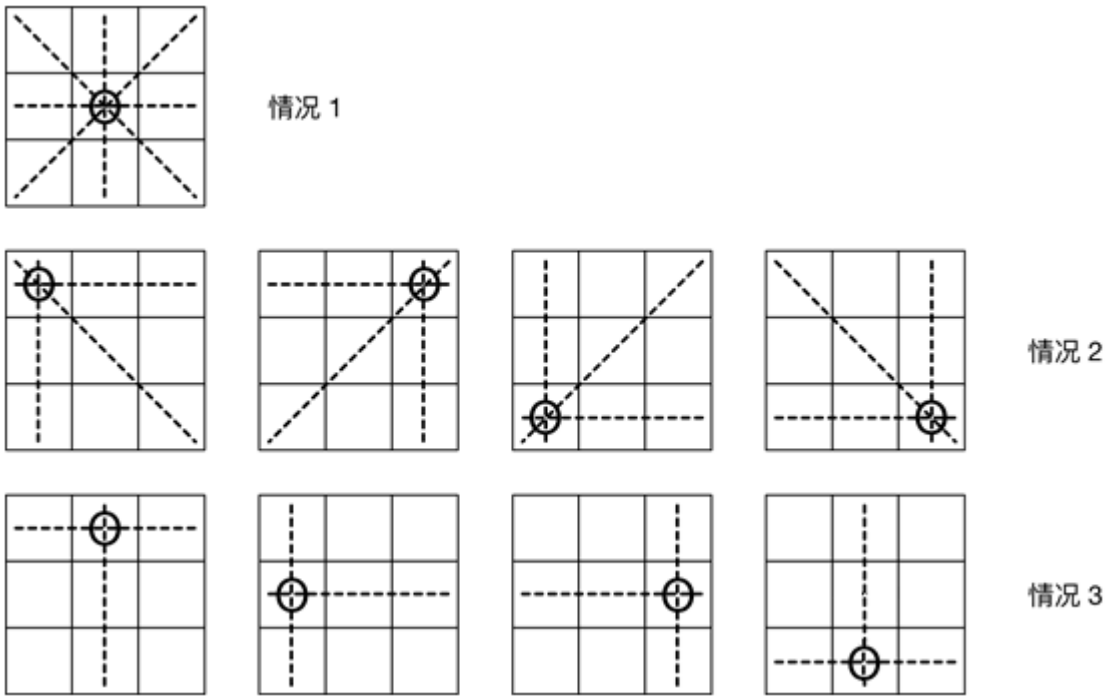

图 3 棋盘落子示意图

观察图 3 不难发现，玩家在空棋盘上落子的位置可分为以下 3 种情况：

（1） 中心点，这个位置共有 4 个方向可能和其它棋子连接成直线，获胜的几率最高。

（2） 4 个角位，这 4 个位置各自有 3 个方向可能和其它棋子连接成直线，获胜几率中等。

（3） 4 个边位，这 4 个位置各自有 2 个方向可能和其它棋子连接成直线，获胜几率最低。

综上所述，如果电脑在落子时，既没有必胜落子位置，也没有必救落子位置时，我们就可以让电脑按照胜率的高低来选择落子位置，也就是说，若棋盘的中心点没有棋子，则选择中心点作为落子位置；若中心点已有棋子，而角位没有棋子，则随机选择一个没有棋子的角位作为落子位置；若中心点和四个角位都有棋子，而边位没有棋子，则随机选择一个没有棋子的边位作为落子位置。

井字棋游戏一共需要设计 4 个类，不同的类创建的对象承担不同的职责，分别是：

（1）游戏类（Game）：负责整个游戏流程的控制，是该游戏的入口。

（2）棋盘类（Board）：负责显示棋盘、记录本轮对局数据、以及判断胜利等和对弈相关的处理工作。

（3）玩家类（Player）：负责记录玩家姓名、棋子类型和得分、以及实现玩家在棋盘上落子。

（4）电脑玩家类（AIPlayer）：是玩家类的子类。在电脑玩家类类中重写玩家类的落子方法，在重写的方法中实现电脑智能选择落子位置的功能。

设计后的类图如图 4 所示。

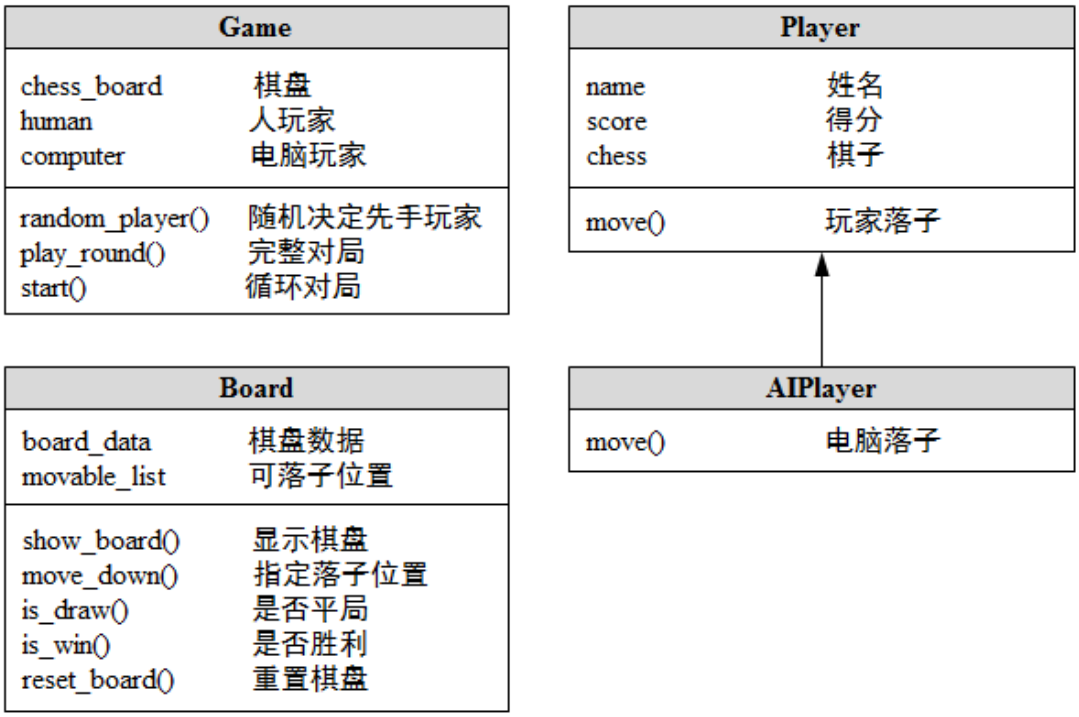

图 4 类结构图

本实例中涉及到多个类，为保证程序具有清晰的结构，可以将每个类的相关代码分别放置到与其同名的 py 文件中。另外，由于 Player 和 AIPlayer 类具有继承关系，可以将这两个类的代码放置到 player.py 文件中。

代码实现

本实例的实现过程如下所示。

1. 创建项目

使用 PyCharm 创建一个名为“井字棋 V1.0”的文件夹，在该文件夹下分别创建 3 个 py 文件，分别为 board.py、game.py 与 player.py，此时程序的目录结构如图 5 所示。

图 5 井字棋文件目录

由于棋盘类是井字棋游戏的重点，因此我们先开发 Board 类。

2. 设计 Board 类

（1）属性

井字棋的棋盘上共有 9 个格子落子，落子也是有位置可循的，因此这里使用列表作为棋盘的数据结构，列表中的元素则是棋盘上的棋子，它有以下三种取值：

- " " 表示没有落子，是初始值；
- "X" 表示玩家在该位置下了一个 X 的棋子；
- "O" 表示玩家在该位置下了一个 O 的棋子。

其中列表中的元素为" "的位置才允许玩家落子。为了让玩家明确可落子的位置，需要增加可落子列表。根据图 4 中设计的类图，在 board.py 文件中定义 Board 类，并在该类的构造方法中添加属性 board_data 和 movable_list，具体代码如下。

```
class Board(object):
    """棋盘类"""
    def __init__(self):
        self.board_data = [" "] * 9          # 棋盘数据
        self.movable_list = list(range(9))   # 可移动列表
```

（2）show_board()方法

show_board()方法实现创建一个九宫格棋盘的功能。游戏过程中显示的棋盘分为两种情况，一种是新一轮游戏开始前显示的有索引的棋盘，让玩家明确棋盘格子与序号的对应关系；另一种是游戏中显示当前落子情况的棋盘，会在玩家每次落子后展示。在 Board 类中添加 show_board()方法，并在该方法中传递一个参数 show_index，用于设置是否在棋盘中显示索引（默认为 False，表示不显示索引），具体代码如下。

```
def show_board(self, show_index=False):
    """显示棋盘
    :param show_index: True 表示显示索引 / False 表示显示数据
    """
    for i in (0, 3, 6):
        print("      |      |")
        if show_index:
            print("   %d  |  %d  |  %d" % (i, i + 1, i + 2))
        else:
            print("   %s  |  %s  |  %s" % (self.board_data[i],
                                           self.board_data[i + 1],
                                           self.board_data[i + 2]))
        print("      |      |")
        if i != 6:
            print("-" * 23)
```

（3）move_down ()方法

move_down ()方法实现在指定的位置落子的功能，该方法接收两个参数，分别是表示落子位置的 index 和表示落子类型（X 或者 O）的 chess，接收的这些参数都是落子前需要考虑的必要要素，具体代码如下。

```
def move_down(self, index, chess):
    """在指定位置落子
    :param index: 列表索引
    :param chess: 棋子类型 X 或 O
    """
    # 1. 判断 index 是否在可移动列表中
    if index not in self.movable_list:
        print("%d 位置不允许落子" % index)
        return
    # 2. 修改棋盘数据
    self.board_data[index] = chess
    # 3. 修改可移动列表
    self.movable_list.remove(index)
```

以上代码首先判断落子位置是否可以落子，如果可以就将棋子添加到 board_data 列表的对应位置，并从 movable_list 列表中删除。

（4）is_draw ()方法

is_draw ()方法实现判断游戏是否平局的功能，该方法会查看可落子索引列表中是否有值，若没有值表示棋盘中的棋子已经落满了，说明游戏平局，具体代码如下。

```
def is_draw(self):
    """是否平局"""
    return not self.movable_list
```

（5）is_win ()方法

is_draw ()方法实现判断游戏是否胜利的功能，该方法会先定义方向列表，再遍历方向列表判断游戏是否胜利，胜利则返回 True，否则返回 False，具体代码如下。

```
def is_win(self, chess, ai_index=-1):
    """是否胜利
    :param chess: 玩家的棋子
    :param ai_index: 预判索引，-1 直接判断当前棋盘数据
    """
    # 1. 定义检查方向列表
    check_dirs = [[0, 1, 2], [3, 4, 5], [6, 7, 8],
                  [0, 3, 6], [1, 4, 7], [2, 5, 8],
                  [0, 4, 8], [2, 4, 6]]
    # 2. 定义局部变量记录棋盘数据副本
    data = self.board_data.copy()
    # 判断是否预判胜利
    if ai_index > 0:
        data[ai_index] = chess
    # 3. 遍历检查方向列表判断是否胜利
```

```
    for item in check_dirs:
        if (data[item[0]] == chess and
            data[item[1]] == chess
                and data[item[2]] == chess):
            return True
    return False
```

注意，is_win()方法的 ai_index 参数的默认值为-1，表示无需进行预判，即提示玩家最有利的落子位置；若该参数不为-1 时，表示需要进行预判。

（6）reset_board ()方法

reset_board ()方法实现清空棋盘的功能，该方法中会先清空 movable_list，再将棋盘上的数据全部置为初始值，最后往 movable_list 中添加 0~8 的数字，具体代码如下。

```
def reset_board(self):
    """重置棋盘"""
    # 1. 清空可移动列表数据
    self.movable_list.clear()
    # 2. 重置数据
    for i in range(9):
        self.board_data[i] = " "
        self.movable_list.append(i)
```

3. 设计 Player 类

根据图 4 中设计的类图，在 player.py 文件中定义 Player 类，分别在该类中添加属性和方法，具体内容如下。

（1）属性

在 Player 类中添加 name、score、chess 属性，具体代码如下。

```
import board
import random
class Player(object):
    """玩家类"""
    def __init__(self, name):
        self.name = name     # 姓名
        self.score = 0       # 成绩
        self.chess = None    # 棋子
```

（2）move()方法

move()方法实现玩家在指定位置落子的功能，该方法中会先提示用户棋盘上可落子的位置，之后使棋盘根据用户选择的位置重置棋盘数据后进行显示，具体代码如下。

```
def move(self, chess_board):
    """在棋盘上落子
    :param chess_board:
    """
```

```
    # 1. 由用户输入要落子索引
    index = -1
    while index not in chess_board.movable_list:
        try:
            index = int(input("请 “%s” 输入落子位置 %s: " %
                (self.name, chess_board.movable_list)))
        except ValueError:
            pass
    # 2. 在指定位置落子
    chess_board.move_down(index, self.chess)
```

4. 设计 AIPlayer 类

根据图 4 中设计的类图，在 player.py 文件中定义继承自 Player 类的子类 AIPlayer。AIPlayer 类中重写了父类的 move()方法，在该方法中需要增加分析中的策略，使得计算机玩家变得更加聪明，具体代码如下。

```
class AIPlayer(Player):
    """智能玩家"""
    def move(self, chess_board):
        """在棋盘上落子
        :param chess_board:
        """
        print("%s 正在思考落子位置..." % self.name)
        # 1. 查找我方必胜落子位置
        for index in chess_board.movable_list:
            if chess_board.is_win(self.chess, index):
                print("走在 %d 位置必胜！！！" % index)
                chess_board.move_down(index, self.chess)
                return
        # 2. 查找地方必胜落子位置-我方必救位置
        other_chess = "O" if self.chess == "X" else "X"
        for index in chess_board.movable_list:
            if chess_board.is_win(other_chess, index):
                print("敌人走在 %d 位置必输，火速堵上！" % index)
                chess_board.move_down(index, self.chess)
                return
        # 3. 根据子力价值选择落子位置
        index = -1
        # 没有落子的角位置列表
        corners = list(set([0, 2, 6, 8]).intersection(
chess_board.movable_list))
```

```
        # 没有落子的边位置列表
        edges = list(set([1, 3, 5, 7]).intersection(
chess_board.movable_list))
        if 4 in chess_board.movable_list:
            index = 4
        elif corners:
            index = random.choice(corners)
        elif edges:
            index = random.choice(edges)
        # 在指定位置落子
        chess_board.move_down(index, self.chess)
```

5. 设计 Game 类

根据图 4 中设计的类图，在 game.py 文件中定义 Game 类，分别在该类中添加属性和方法，具体内容如下。

（1）属性

在 Game 类中添加 chess_board、human、computer 属性，具体代码如下。

```
import random
import board
import player
class Game(object):
    """游戏类"""
    def __init__(self):
        self.chess_board = board.Board()          # 棋盘对象
        self.human = player.Player("玩家")        # 人类玩家对象
        self.computer = player.AIPlayer("电脑")   # 电脑玩家对象
```

（2）random_player()方法

random_player()方法实现随机生成先手玩家的功能，该方法中会先随机生成 0 和 1 两个数，选到数字 1 的玩家为先手玩家，然后再为两个玩家设置棋子类型，即先手玩家为“X”，对手玩家为“O”，具体代码如下。

```
def random_player(self):
    """随机先手玩家
    :return: 落子先后顺序的玩家元组
    """
    # 随机到 1 表示玩家先手
    if random.randint(0, 1) == 1:
        players = (self.human, self.computer)
    else:
        players = (self.computer, self.human)
    # 设置玩家棋子
```

```
        players[0].chess = "X"
        players[1].chess = "O"
        print("根据随机抽取结果 %s 先行" % players[0].name)
        return players
```

（3）play_round ()方法

play_round ()方法实现一轮完整对局的功能，该方法的逻辑可按照实例分析的一次流程完成，具体代码如下。

```
    def play_round(self):
        """一轮完整对局"""
        # 1. 显示棋盘落子位置
        self.chess_board.show_board(True)
        # 2. 随机决定先手
        current_player, next_player = self.random_player()
        # 3. 两个玩家轮流落子
        while True:
            # 下子方落子
            current_player.move(self.chess_board)
            # 显示落子结果
            self.chess_board.show_board()
            # 是否胜利?
            if self.chess_board.is_win(current_player.chess):
                print("%s 战胜 %s" % (current_player.name, next_player.name))
                current_player.score += 1
                break
            # 是否平局
            if self.chess_board.is_draw():
                print("%s 和 %s 战成平局" % (current_player.name,
    next_player.name))
                break
            # 交换落子方
            current_player, next_player = next_player, current_player
        # 4. 显示比分
        print("[%s] 对战 [%s] 比分是 %d : %d" % (self.human.name,
                                        self.computer.name,
                                        self.human.score,
                                        self.computer.score))
```

从上述代码可以看出，大部分的功能都是通过游戏中各个对象访问属性或调用方法实现的，这正好体现了类的封装性的特点，即每个类分工完成各自的任务。

（4）start ()方法

start ()方法实现循环对局的功能，该方法中会在每轮对局结束之后询问玩家是否再来一局，若玩家选择是，则重置棋盘数据后开始新一轮对局；若玩家选择否，则会退出游戏，具体代码如下。

```
def start(self):
    """循环开始对局"""
    while True:
        # 一轮完整对局
        self.play_round()
        # 询问是否继续
        is_continue = input("是否再来一盘（Y/N）？").upper()
        # 判断玩家输入
        if is_continue != "Y":
            break
        # 重置棋盘数据
        self.chess_board.reset_board()
```

最后在 game.py 文件中通过 Game 类对象调用 start()方法启动井字棋游戏，具体代码如下。

```
if __name__ == '__main__':
    Game().start()
```

代码测试

运行程序，对战一局游戏的结果如下所示：

```
       |      |
   0   |  1   |  2
       |      |
-----------------------
       |      |
   3   |  4   |  5
       |      |
-----------------------
       |      |
   6   |  7   |  8
       |      |
根据随机抽取结果 电脑 先行
电脑 正在思考落子位置...
       |      |
       |      |
       |      |
```

```
----------------------
      |      |
      |  X   |
      |      |
----------------------
      |      |
      |      |
      |      |
请“玩家”输入落子位置 [0, 1, 2, 3, 5, 6, 7, 8]: 0
      |      |
  O   |      |
      |      |
----------------------
      |      |
      |  X   |
      |      |
----------------------
      |      |
      |      |
      |      |
电脑 正在思考落子位置...
      |      |
  O   |      |
      |      |
----------------------
      |      |
      |  X   |
      |      |
----------------------
      |      |
  X   |      |
      |      |
请“玩家”输入落子位置 [1, 2, 3, 5, 7, 8]: 2
      |      |
  O   |      |  O
      |      |
----------------------
      |      |
      |  X   |
```

```
      |      |
----------------------

      |      |
  X   |      |
      |      |
电脑 正在思考落子位置...
敌人走在 1 位置必输，火速堵上!
      |      |
  O   |  X   |  O
      |      |
----------------------

      |      |
      |  X   |
      |      |
----------------------

      |      |
  X   |      |
      |      |
请“玩家”输入落子位置 [3, 5, 7, 8]: 7
      |      |
  O   |  X   |  O
      |      |
----------------------

      |      |
      |  X   |
      |      |
----------------------

      |      |
  X   |  O   |
      |      |
电脑 正在思考落子位置...
      |      |
  O   |  X   |  O
      |      |
----------------------

      |      |
      |  X   |
      |      |
----------------------
```

```
      |      |
  X   |  O   |  X
      |      |
请“玩家”输入落子位置 [3, 5]：5
      |      |
  O   |  X   |  O
      |      |
----------------------
      |      |
      |  X   |  O
      |      |
----------------------
      |      |
  X   |  O   |  X
      |      |
电脑 正在思考落子位置...
      |      |
  O   |  X   |  O
      |      |
----------------------
      |      |
  X   |  X   |  O
      |      |
----------------------
      |      |
  X   |  O   |  X
      |      |
电脑 和 玩家 战成平局
[玩家] 对战 [电脑] 比分是 0 : 0
是否再来一盘（Y/N）？n
```