第21章井字棋

井字棋，是一种在3x3格子上进行的连珠游戏，英文名叫Tic^Tac^Toe,游戏规则和五子 棋类似。由于棋盘一般不画边框，格线排成井字故得名。本章将带领大家开发一个井字棋的游戏, 让大家对面向对象的封装、继承以及多态三大特性有更好的理解和体会。同时还引入一个简单 的算法，让计算机可以智能地选择落子位置，实现人机对战的效果。

2LI漩殲简命

21JJ Ii»i!

井字棋又称为井字游戏、圈圈叉叉；另外也有打井游戏、 o x棋的称呼。我国香港特别行政区多称井字过三关、过三关。

井字棋的游戏规则非常简单，两个玩家，一个打圈（O）， 一个打叉（X ）,轮流在3 X 3的格上打自己的符号（一般打X 的先走），最先在横向、竖向，或斜线方向连成一线的为胜。如 图214所示，这一盘打圈（O ）的玩家获得胜利。

21Jd2

井字棋游戏是一个相对简单的、有固定策略的游戏，如果游 戏双方都下得正确的话，必然会得到和局。这种游戏实际上是由 第一位玩家（假设是x玩家）所控制，第一位玩家是攻，第二位 玩家身段设是。玩家）是守。

x玩家（先手）有三种可能的开局方式角位，边位和中心位置， 这三种位置的示意图如图2卜2所示，其中三角形所在格子为角位，

圆形所在格子为边位，正中间的太阳形状所在格子为中心位。 图21-2井字棋的三种位置

x玩家在角位落子开局赢面最大。。玩家的第一步若是在边位落子，x玩家就可以用两 粒连线牵制着。玩家，然后制造分枝，俗称“两头蛇” o图2卜3所示为其中一种典型场景。

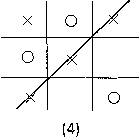
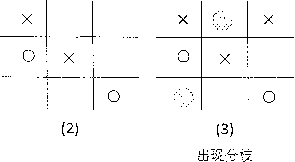
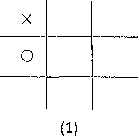
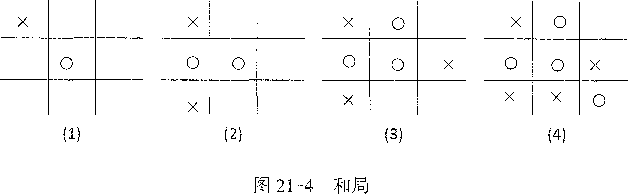


图 21-3

x玩家赢

O玩家第一步若是在中心落子,

则会成和局，图2卜4所示为其中一种典型场景。

其实，X玩家无论在哪个地方开局都有机会赢或至少逼平对手，然而选择角位作为开局时 留给。玩家的选择是最少的*，*也就是说这时候。玩家最容易走错。

对于。玩家来说，如果X玩家没有失误，那么。玩家的最好结果就是和局。其必须选择 中心位以应对选择角位的开局，选择角位来应对选择中心位的开局。当X开局选边时，。必 须选择中心、与X相邻的角或者与X相对的边，不然就会输。

那么在井字棋中正确的落子策略是什么呢？

其实，如果你按照以下所列的优先顺序选择落子位置，将会有最完美的表现：

（1）判断能否获胜。当你有两粒连子的时候，将其连成3个。

（2 ）如果己方不能获胜，则判断是否需要阻挡。如果对方有两粒连子，阻止它们构成三连子。

（3 ）如果不需要阻挡对方，则试图营造己方的分枝，即营造你可以通过两条路径获胜的机会。

（4） 如果无法营造己方的分枝，则试图阻止对方形成分枝：

-方法一：造成两粒连子迫使对方阻挡，前提是对方的阻挡不能使对方获胜。

•方法二M占据对方可以用来做成分枝的点。

（5） 如果对方没有形成分枝的机会，则选择落在中心点。

（6） 如果中心点被占据•而对方在角上歹则占据与之相对的角。

（7） 如果对方角上没有棋子，则占据一个空余的角落。

（8） 如果已经没有空余的角落可以落子，则选择占据空余的边侧。

ZL2 fit

本案例的设计目的是让开发者熟悉如何使用面向对象思想开发一个小游戏程序，并对面向 对象的封装、继承以及多态三大特性有更深的理解。在此基础上，对井字棋游戏进行流程分析、 类的设计，以及开发阶段设计的工作。

井字棋游戏程序的运行流程如图21-5所示。

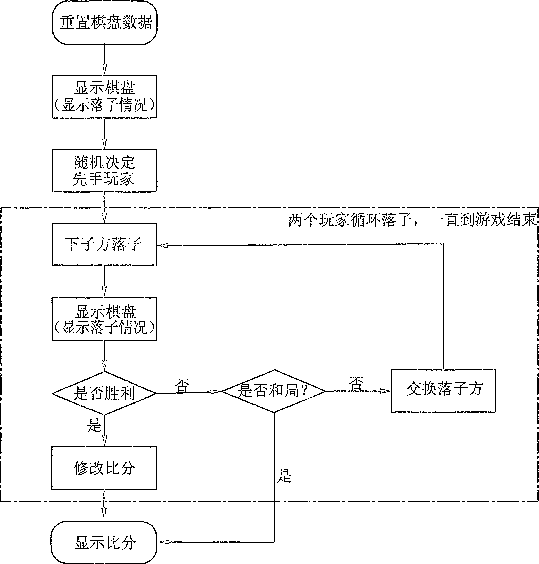


图21-5井字棋的流程

（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？如果胜利，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？如果和棋，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。

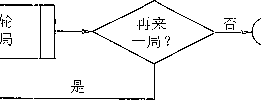
（9） 显示玩家当前对局比分。

为了增加游戏的可玩度，在井字棋游戏中还增加了循环对局功能（见图21・6）,即在每轮 对局结束之后，由玩家来选择是否再来一局。

«如果玩家选择“是”，则跳转到第1步，开始新一轮对局；

•如果玩家选择“否”，则退出游戏。

循环对局流程如图21-6所示。



对，

结束)

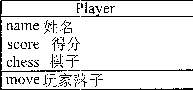
图21-6循环对局流程图

212.2 .

根据流程分析，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。
2. 棋(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. 计算机玩家类(AlPlayer):是玩家类的子类。在子类中重写父类的落子方法，在重 写的方法中实现计算机智能选择落子位置的功能，使得人机对战更有挑战，游戏更有乐趣！

设计后的类图如图2卜7所示。



chess borad 棋盘 human人玩家 computer电脑玩家

board\_data棋盘数据 movablejist 可落子索引列表

randomjplayer随机决定先手玩家 play\_round 一局完整对局 start 开始循环対局

show\_board显示棋盘 move\_down在棋盘指定位置落子 is\_draw是否平局 is\_win是否胜利 reset board 重置棋盘

图21-7类的设计图

212.3舞隱險殿谶谤 ，一

本案例的设计目的主要有以下两个：

1. 更好地体会面向对象的封装特性。
2. 引入一个简单的AI (人工智能)算法，实现人机对战。也借此让大家对继承和多态 特性有更好的体会。

为了更好地达成设计目的，我们将整个开发过程分为人人对战与人机对战两个阶段。每个 阶段的内容如下所示。

翁〜險殿：人人对战

-根据类的设计，实现游戏类、棋盘类和玩家类的所有功能。

•完整实现人人对战模式的游戏流程。

第二阶段；人凯对战

•由Play©『类派生一个AlPIa炉子类，在子类中重写move方法，让计算机玩家聪明地落子。

464 w. Pjthoo囊饑廳幽 映等雾Pjfthoo

。在Game类中创建计算机玩家，使用多态的方法实现人机对战的功能。

同时，为了简化程序设计，突出项目的逻辑功能，我们使用控制台程序来实现井字棋游戏 的功能。

I 2L3新建项目魇曳佛雑鑑

项目分析与设计都完成以后，就可以实施开发了。

之前章节介绍过，PyChaim是一个非常好用的工具，因此井字棋项目选择PyCharm来实现。 由于本书编写之时，Pythoii的最新稳定版本为36,所以项目选择基于Python 3.6进行开发。

项目开发的第一步就是创建项目和项目结构。首先，使用PyCh為mi新建一个项目，取名为“井 字棋”，然后在项目中依次建立3个模块，分别是：

• game.py：游戏主模块文件。

0 board.py：棋盘模块文件。

° player.py：玩家模块文件。

创建好的项目结构如图2「8所示。

w懿碧寧繼*口：5座5舞者""\*5"参*

board\*py

game.py player py

图21-8项目结构图

（☆注意：

在开发过程中要注意，每完成一个相对独立的功能，都建议对该功能进行测试，以保证实 现该功能的代码的正确性。避免将整个程序的代码完成以后再调试，这样很容易因为错误过多 而加大调试工作的难度。

2L4棋矗脆实现

在井字棋游戏中，棋盘对象负责以下职责：

（1） 显示棋盘的格子编号，以方便玩家选择落子位置；

（2） 记录本轮对局玩家轮流落子的完整记录；

（3） 显示玩家们的落子结果；

（4） 判断胜利;

（5） 判断平局；

（6） 重置棋盘数据，以方便新一轮游戏的开始。

根据职责的划分，不难看出棋盘类的设计是井字棋游戏的重点。因此，我们将棋盘类作为 井字棋开发的第一步。首先，需要明确棋盘的数据结构，也就是以什么数据类型来记录玩家的 落子。

21.4J纔蠢餉歡骚籍将

井字棋的棋盘上共有9个格子可以落子，因此要让棋盘记录这9个格子的落子情况，我们 可以使用一个列表来完成。列表中元素的取值有三种情况，其含义如下所示。

-""表示没有落子，是初始值；

•'成”表示玩家在该位置下了一个x的棋子；

• !S o 表示玩家在该位置下了一个O的棋子。

其中，只有记录H ”的项，才允许玩家落子，而其他两种情况表示该位置已经有棋子，不 再允许玩家落子。

t列畿繁引写纔廳位置

现在我们已经知道了，可使用一个列表来记录棋盘的落子数据。那么在游戏中，应该怎样 把列表中的数据以棋盘的形式显示在屏幕上呢？我们需要明确一下列表的索引和棋盘位置的对 应关系,如图21・9所示。

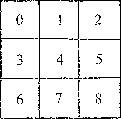
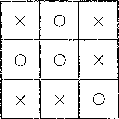


图21・9列表索引与棋盘位置的关系

把列表中的。〜2项的内容显示在第1行，3〜5项的内容显示在第2行，6~8项的内 容显示在第3行。这样，游戏在运行的时候，就可以看到一个棋盘形式的展现效果了。现在， 假设列表中保存了一些“x”和的数据，具体的展示效果如图2140所示。

列表数据 XOXOOXXX O

屏幕显示

图21-10棋盘的展现效果

1. 可落子索引列表

根据之前的分析，我们已经能够明确，只有列表值为-的位置，才允许玩家落子。

为了明确可以落子的位置，在棋盘对象中，可以再增加一个名为movableJist的属性（可 落子索引列表），用来记录当前所有值*为j*的索引。

例如,在新一轮游戏刚刚开始时,movablejist中保存的内容如下:

[0, 1, 2, 3, 4, 5, 6, 7, 8]

表示9个格子都可以落子。如果先手玩家在索引。的位置，放置了一个“x”，那么就把 数字0从列表中删除，结果如下：

丄 2, 3, 4, 5, 6, 7, 8]

根据列表的值，下一个玩家在落子时，只能选择在1〜8索引之间落子，而不能在索引0 的位置落子。

依此类推，当movableJist列表中没有任何值时，就表示棋盘上已经没有任何可以落子的 位置，由此可以用来判断平局。

21d4d2

按照之前的分析，棋盘对象中需要记录2个列表，分别是棋盘数据列表(取名为board\_ data)和可落子索引列表(取名*为*movable\_list)。

在board^py文件中定义Board类实现棋盘初始化功能，并且实现其方法，代码如下：

class Board(object):

"n n棋盘类° " n

def init (self):

self. board\_data = [" " ] \* 9 # 棋盘数据

self .movable\_list = list (range (9) ) # 可移动歹0表

接下来，在board^py的末尾增加以下代码，测试棋盘的初始化。

if name == ? main ?:

# 1.测试初始化

board = Board()

print(board«board\_data)

print(board.movable\_list)

在PyCharm中运行board^py文件进行测试,可以看到在PyCharm控制台中打印出以下信息， 说明棋盘初始化完成。

[0, lz 2, 3, 4, 5, 6/ 7, 8]

打印出来的信息是棋盘初始化时棋盘数据列表和可落子索引列表里存储的值。

21d4q3區彖鶴蘆

按照之前的分析，显示棋盘有两种情况：

°新一轮开始前，显示棋盘格子对应的索引值，让玩家明确棋盘格子与序号的对应关系；

。玩家每次落子之后，显示当前棋盘的落子结果，向玩家展示当前的棋局。

要实现显7K棋盘的功能，在Board类中定义show\_board方法,并接收一个参数show in- dex9用以区分上述两种情况,代码如下:

def show\_\_board (self z show\_index=False):

"n °显示棋盘

:param show\_index: True表示显示索引/ False表示显示数据

H队H

for i in (O’ 3Z 6):

print (° I I ") ‘

if show\_index:

print (° %d | %d

else :

print(" %s | %s

print (n | |")

if i != 6:

%dn % (i, i + 丄 i + 2))

%s° %

(self .board\_\_data [i] / self »board\_data[i + 1]z self oboard\_\_data [i + 2]))

print\* 23)

其中，show\_index True,表示显示棋盘格子的索引，False表示显示落子结果。

接下来，在board^py文件的末尾增加以下代码，用来测试显示棋盘的功能。 # 2.显示棋盘

print — ~显示棋盘研+ " \* 50)

board. show\_board (True) # 显示索引

print\* 50)

board. show\_board () # 显示数据

运行boardpy文件，可以看到，已经成功打印了棋盘，如下所示。

显 TF 棋盘 —

I I

0 | 1 | 2

214n4種鶴蠱上藉字

按照之前的分析，玩家落子包含两个必要要素？落子的位置(列表的索引)和落子的类型(X 或者。)O

要实现此功能，在Board类中定义move\_down方法，并接收两个参数，分别是落子的位 置index和落子的类型chess,代码如下:

def move\_\_down (self f index^ chess):

5"应指定位置落子

:param index: 歹U表索弓]

:param chess :棋子类型*乂或。*

n n n

* 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.movab1e\_li s t.remove(index)

在上述代码中，先判断落子位置是否可以落子，如果可以，则将棋子添加到board\_data列 表的对应位置,并从movable list列表中删除。

接下来,在board.py文件的末尾增加以下代码,用来测试落子功能:

# 3.测试落子

print 测试落子” + \* 50)

board B move\_down(0 f )

board.show\_board()

print(board.movable\_list)

board. move\_down (0, "X" ) #测试在已有棋子的位置落子

运行board.py文件,可以看到打印如下信息。

[1, 2, 3, 4, 5, 6, *L* 8]

0位置不允许落子

21.4.5剽嬲零圖

在上一小节中实现的落子方法中，每一次在棋盘上记录了玩家的落子之后歹就会修改可落 子索引列表。而按照之前的分析，当可落子索引列表中没有值时，就表示棋盘已满，我们可以 把这个作为判断平局的条件。

在Board类中定义is draw方法,如下所示。

def is\_draw(self):

° " ”是否平局° " ° return len(self.movable\_list) == 0

然后，在board^py文件的末尾增加以下代码，用来测试判断平局的功能。

# 4 .判断平局

print (°——扌寸®f平局叫+ \* 50)

print (° 是否平局 %d° % board. is\_draw ())

board.movable\_\_list。clear () # 清空可移动索引列表

print (° 是否平局 %d° % board. is\_draw ())

运行board^py文件，可以看到如下打印结果。

—一判断平局一一—— —

是否平局0

是否平局1

21.4.6

按照井字棋的游戏规则，当玩家每次落子后，如果在棋盘的水平、垂直或者对角线任意一 个方向上，玩家的棋子连成一条直线，则表示玩家获胜。

因此，在判断胜利时，只需要在各个方向上判断指定的棋子是否连成了一条直线，就能够 判断该类型的棋子是否获胜了。

在Board类中定义is win方法,并接收一个参数chess,用以传递玩家的棋子,代码如下:

def is\_win(selff chess):

"5是否胜利

:param chess :玩家的棋子

# 1.定义检查方向列表

check\_dirs

[[0, 1, 2],

[0z 3, 6]，

[3, 4, 5] , [6, 7, 8], 丄 4, 7] , [2, 5, 8],

# 2.

复制棋盘数据-

4, 8], 后续AI

[2, 4, 6]]

需要使用

data = self .board\_\_data. copy ()

# 3.遍历检查方向列裹判断是否胜利

for item in check\_dirs:

if (data[item[0]] == chess and

data[item[1]] == chess

and data[item[2]] == chess):

return True

return False

以上代码中的第2步——复制棋盘数据，是为后续引入AI铺垫的，后续会详细介绍。

接下来，在boardpy文件的末尾增加以下代码，用来测试判断胜利的功能。

# 5。判断胜利

print (" 判断胜利”

print ("是否胜利%d°

+ \* 50)

% board.is\_win(°X"))

board»move\_down(0 7 °Xn)

board.move\_down(1r *"X")*

board.move\_down(2, "X")

board <, show\_board ()

print (° 是否胜利 %dn % board. is\_\_win ())

需要注意的是，在测试判断胜利方法前，需要将测试平局的代码注释一下，否则无法落子。 并且在平时练习时，可以通过修改测试数据，从而测试判断胜利方法是否能够正确执行。

运行board.py文件,可以看到输出结果为:

是否胜利0

按照之前的分析，当一轮游戏结束后，进入下一轮对局之前，需要重置棋盘数据，也就是 把上一轮的对局数据清空。

要实现重置棋盘的功能，在Bcwd类中定义reset\_board方法，如下所示。

def reset\_board(self):

° ° °重置棋盘",,"

* 「清空可移动列表数据

self »movable\_list。clear()

* 2.重置数据一

for i in range (9):

self.board\_data[i] = " "

self.movable\_list。append(i)

然后，在board.py的末尾增加以下代码，用来测试重置棋盘：

# 6.测试重置棋盘数据

print (°——重置棋盘” + n-n \* 50)

board.reset\_board()

board。show\_board()

print(board omovable\_list)

需要注意的是，与初始化方法不同，在重置棋盘方法中，只是重新设置了列表中的数据, 而没有再次为数据列表分配空间。

运行程序，可以看到打印如下结果。

---重置棋盘 —

[0, 1, 2, 3, 4, 5, 6, 7, 8]

到目前为止,Board类中封装的方法已经全部完成并通过测试。后续的游戏功能实现过程中， 我们可以直接调用已经封装的方法*。*

2L5疏家碰能实现

在井字棋游戏中，玩家对象的主要职责就是负责在棋盘上选择一个位置落子。同时，为了 提高游戏的可玩性，我们还需要在玩家对象中封装三个属性，分别是：

•姓名(name ):玩家名称。

•得分(score ):玩家的总对局得分o

•棋子(chess):玩家在本轮对局中使用的棋子(x或者。)。

21.5J浏嬌徵巍豪 '「

按照之前的分析，玩家对象中需要封装3个属性：姓名、得分和棋子。但是，在创建玩家 对象时，并不知道当前玩家具体使用的棋子是什么。因为按照井字棋规则，先手玩家使x棋子,

后手玩家使用。棋子，而玩家的先手和后手是在每一轮对局开始前随机决定的。

所以，我们可以在初始化方法中，暂时将玩家的棋子属性设置为Non©。待后续开发到游 戏模块时，设计一个单独的方法来设置当前对局中两个玩家的棋子。

要实现玩家的初始化，可以在player.py模块中定义Player类，并且实现初始化方法，代码 如下：

class Player (object):

° °"玩家类° ° °

def init (selfz name)

self o name = name #

名绩子 姓成棋

self » score = 0 #

self.chess = None #

21n5H2歸蒙程鶴置蕎字

我们先来简单分析一下玩家在棋盘上落子的程序逻辑：

•玩家在落子前，首先需要知道棋盘上当前哪些位置可以落子；

。玩家自行决定在哪一个空闲位置上落子；

-玩家使用自己的棋子，在棋盘的指定位置上落子；

•由于我们目前的开发重点是实现人人对战，因此，玩家决定在哪一个空闲位置上落子这 一步骤，可以通过input函数由玩家自行输入。

同时，为了保证在玩家的落子方法内部能够调用到棋盘对象封装的方法，我们可以把棋盘 对象当作参数，传递给玩家的落子方法。

要使用Board类*，*我们首先需要在player.py模块的顶部导入board.py模块a代码如下:

import board

然后，在Pla炉类中定义move方法，接收一个棋盘对象参数chess\_board9代码如下？

def move(selfr chess\_board):

5，，在棋盘上落子

:param chess\_board:

n n n '

* 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.namez chess\_boardomovable\_list))) except ValueError:

pass

* 2。在指定位置落子

chess\_\_board.move\_down (indexf self . chess)

接下来，在piayerpy文件的末尾增加以下代码，用来测试玩家在棋盘落子的功能。

if name == \* main 1 :

#「创建棋盘对象

chess board = board.Board()

* 2.创建玩家对象

human = Player (° ")

human.chess = nXn

* 3。玩家在棋盘上循环落子，直到玩家胜利

while not chess\_board.is\_win(human.chess): human.move(chess\_board)

#显示棋盘 -

chess\_boardo show\_board()

要注意的是，在编写这段测试代码时，需要主动设置一下玩家的棋子。 此时,运行player.py文件,可以看见如下输出:

请“玩家”输入落子位置[0, 1, 2, 3, 4, 5, 6, *1,* 8]:

由于在前一个小节中9我们已经将棋盘相关的所有功能和数据全部封装到了 Board类中。 因此，在开发Player类时，只需要准备一个棋盘对象，然后由这个对象来调用之前准备好的方 法即可，而不再需要关心和棋盘相关的实现细节。这就是面向对象的封装特性成利用封装特性 可以提高软件开发的可重用性、灵活性和扩展性。

项目进行到此，我们已经准备好了棋盘类和玩家类。接下来，就可以开始游戏类的开发, 先来实现人人对战的功能。

首先9我们需要在game.py模块中导入已经开发好的boarck pla炉两个模块，代码如下:

import board

import player

21b6J邂瓣髓潮觸德

按照之前的分析，井字棋游戏的主体包括1个棋盘对象和2个玩家对象。因此，我们可以 在游戏类的初始化方法中完成这3个对象的创建。

要实现游戏的初始化,可以在game.py模块中定义Game类，并且实现初始化方法，代码如下:

class Game(object):

° ° "游戏类° n "

def init (self):

self . chess\_board = board.Board () # 棋盘对象

self s human = player. Player ("玩家 1 °) # 人类玩家对象 1

self. computer = player . Player ("玩家 2 n) # 人类玩家对象 2

由于这里先实现人人对战,所以我们暂时使用人类玩家的名称赋值给computer属性,作为 游戏的第二个人类玩家。在后续实现了人机对战以后，将会修改computer属性的玩家名称为计 算机，只保留人机对战的功能。

在每一轮新的游戏开始之前，需要随机决定先手玩家。同时，按照井字棋的游戏规则，先 手玩家使用的棋子是X,后手玩家使用的棋子是O。因此，在开发一轮完整对局的方法之前， 我们先做一个准备工作一一随机确定先手玩家。

首先，在gamely模块的顶部导入random模块，以方便使用随机数函数实现随机选择先 手玩家的功能，代码如下：

import random

然后,在Game类中定义random player方法，代码如下:

def random\_player(self):

ee随机先手玩家

:return :落子先后顺序的玩家元组

聞H H

#如果产生的随机数为1表示状家先手

if randomo randint(0 f 1) == 1:

players = (self.humanf self.computer)

else:

players = (self.computerf self.human)

#设置玩家棋子

players[0].chess = "X"

players[1].chess = °On

print ("根据随机抽取结果%s先行"% players [0] .name) return players

接下来，在gamely文件的末尾增加以下代码，用来测试随机决定先手玩家的功能。

if name == ' main ?: Game().random\_player()

运行game.py文件，可以看到如下测试结果：

根据随机抽取结果玩家1先行

或者为：

根据随机抽取结果玩家2先行

21d@o3『鰭嘉驚对晨

接下来就可以开始一轮对局了。我们回顾一下一轮完整对局的流程：

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. 当前一方落子。
5. 显示落子后的棋盘。
6. 判断落子一方是否胜利？如果胜利，修改玩家得分，跳转至第(9)步。
7. 判断是否和棋？如果是和棋，跳转至第(9)步。

(8 )交换落子方，跳转至第(1 )步.继续本轮游戏。

(9)显示玩家当前对局比分，本轮对局结束。

根据以上流程,我们在Game类中定义play\_round方法,实现一轮完整对局的控制。代码 如下：

def play\_round(self):

“5 —轮完整对局，顷，， #「显示棋盘落子位置 self.chess\_board.show\_board(True)

* 2.随机决定先手

current\_\_player f next\_player = self . random\_player ()

* 3.两车玩家轮流落子

while True:

#落子方落子 current\_player.move(self chess\_board)

#显示落孑结果 —

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

#是否为平局

if self.chess\_boardois\_draw(): print ("%s 和 %s 战成平局 "% (current\_player.namez next\_player.name)) break

#交换落子方 current\_playerf next\_player = next\_playerz current\_player

* 4.显示比分

print (" [%s] 对战[%s]比分是 %d : %d" % (self. human. namez self.computer.nameA self.human。scoref self o computer.score))

在play\_round方法的实现中可以看出，在一轮完整对局的方法中，除了交换落子方的代码 之外，其他代码基本上就是让各个对象调用方法，或者访问对象的属性，而这些方法和属性， 都是之前已经封装完成并且测试通过的。

接下来，修改ganiRpy文件末尾的代码，测试一轮完整对局，如下所示。

if name == f \_main ': Game().play\_round()

此时运行game..py文件就可以进行一场完整的对局了。

21q6q4籟醯繭屬

为了增加游戏的可玩性，要实现循环对局的功能。因此，在每轮对局结束之后，要询问玩 家是否再来一局。如果玩家选择是，需要先重置棋盘数据，然后开始新一轮对局；如果玩家选 择否，会退出游戏；

在上一小节中，我们已经在play\_round方法中完成了一轮完整对局。现在要想实现循环对 局，只需要再在Game类中定义一个start方法，并且实现以下代码。

def start(self):

while True:

# 一轮完整对局

self <, playground ()

#询问是否现续 is\_continue = input ( ° 是否再来一盘(Y/N ) ? ° ) , upper ()

#詞断玩家输入

if is\_continue != "Y":

break #重置棋盘数据 self.chess\_board.reset\_board()

接下来，修改gamepy文件末尾的代码，用来测试人人对战的完整过程。代码如下《

if name == \* main 1 :

Game() .start ()

2L7 ——

我们已经实现了人人对战,现在开始实现人机对战的功能。在已经完成的Player类中,是 通过inpirt函数让玩家对象输入落子位置的。但是，计算机显然是不需要输入的。因为，计算 机可以自主地选择一个合适的落子位置。

要实现这一功能，之前Player类中完成的move方法，显然无法胜任。因此，我们可以考虑, 在Player类的基础上派生一个子类AlPlayw。在保留已有三个属性的同时，针对计算机落子的 需求,重写move方法。

21.7J漉館闽Pfey"琴襲 ，.

先来实现一下最简单的随机落子，即在棋盘的可落子索引列表中，随机选择索引作为落子 位置。

首先，在player.py模块的顶部导入random模块，以方便使用随机数实现从可落子列表中 选择一个索引的功能，代码如下：

import random

然后在player.py模块Player类的下方,派生Player的子类AlPlayer,在AlPlayer类中重写 move方法，代码如下:

class AlPlayer(Player):

° n °智能玩家° ""

def move(selfz chess\_board):

5“在棋盘上落子

:param chess\_board:

W H

print ( n%s正在思考落子位置% self . name)

#从棋盘对象的可落子索引列表中随机选择一个索引

index = randomchoice (chess\_board.<,movable\_list)

#在指定位置落子 — —

chess board.move down(index7 self o chess)

21.7.2修教if戏襲听讲蠶諷碗豪鲍襲製

修改Game类的初始化方法，使用新派生的AlPlayer类创建computer对象，并修改玩家的 名称为“玩家”和“计算机“，代码如下:

def init (self):

self o chess\_board = board,Board()

#棋盘对象

#人类玩家对象

#计算机玩家对象

self . human = player. Player ("玩家 ")

self.computer = player oAlPlayer("计算2几 n)

运行程序，会发现计算机已经可以自主选择落子位置了。虽然落子的位置不是很好，但是 已经实现了人机对战的效果。

通过计算机随机落子这一步的实现，大家可以体会一下面向对象的继承和多态两个特性。 通过继承，子类可以拥有父类的所有属性和方法，还可以根据特有需求，对父类进行扩展。继 承能够提高代码的复用，相同的代码不需要重复编写。

通过多态，不同子类对象，可以调用相同父类方法，产生不同的结果。多态能够提高代码 的灵活性。

2L8人楓官寸龈——计算fl智能落于

在这一小节，我们就针对刚刚完成的计算机落子方法进行优化，让计算机在选择落子时， 能够稍微“聪明”一点。同时也给大家简单演示编写对弈类游戏的基本思路及实现过程。

首先，我们来分析计算机的落子策略，也就是按照什么套路落子胜率会更高。

按照井字棋的游戏规则，当玩家每次落子后，如果在棋盘的水平、垂直或者对角线任意一 个方向上，玩家的棋子连成一条直线，则表示玩家获胜。

那么，我们就可以把计算机的落子位置按照优先级分成以下三种:

t宓胜藩字位置

我方在该位置落子会获胜。一旦岀现这种情况，显然应该毫不犹豫在这个位置落子。

1. 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身那么应该在必救落子位置落 子，以阻止对方获胜。

1. 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概率越高，子力价值就越大;获胜的概率越低, 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。

21D8n2 ffWita

按照上述分析，无论是必胜落子还是必救落子，都需要能够对棋盘上某一个位置进行预判。

需要注意的是，预判有两个重要的特点：

'预判不会修改当前棋盘的数据；

•预判只是假设在棋盘的某个位置增加我方或者敌方棋子，用以判断我方或者敌方是否 胜利。

在这里,我们先回顾之前在棋盘类中封装的is\_win方法,代码如下:

def is\_win(self, chess):

"5是否胜利

:param chess :玩家的棋子

H H H

* 1.定义检查方向列表

check\_dirs = [ 1, 2] , [3, 4/ 5] 7 [6Z 7, 8]"

[0, 3, 6], [1, 4, 7], [2, 5, 8L

[0, 4, 8], [2, 4, 6]]

* 2»定义局部变量记录属性值-后续AI需要使阙

data = self.board\_data

* 3.遍历检查方向列裏判断是否胜利

for item in check\_dirs:

if (data[item[0]] == chess and

data[item[l]] == chess

and data[item[2]] == chess):

return True

return False

可以看到,在is\_win方法中,第1步定义了检查方向列表,第2步使用了一个局部变量 如&记录棋盘数据列表，第3步遍历了检查方向列表，判断参数chess的棋子是否获胜。

如果希望在is\_win方法的基础上增加预判功能，我们需要对该方法做三点调整： 僵给该方法增加一个参数，用来传递要预判的索引位置，默认值是T,表示不是预判。

。修改第2步代码，在给＜h伝变量赋值的同时做一个copy的动作，目的是产生一个数据 的复本data。这样，无论在方法内部对data的数据做怎样的修改，都不会影响到self board\_data列表中的数据。

•判断is\_win方法的参数，如果是预判，则在指定位置先记录chess,再检查是否胜利。 调整后的代码如下：

def is\_win(selfz chess, ai\_index=-l):

5匚是否胜利 -

: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 = [[0f 1r *2]r* [3\* 4 f 5]z [ 6 f 78]z

[0, 3, 6] ,, [1, 4, 7], [2, 5, 8] 7

[0, *4f* 8], [2, 4, 6]]

* 2。定义局部变量记录棋盘数据副本 data = self.board\_data.copy()

#判断是否预判胜利一

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

上述代码的第二步中，当aijudex不为-1时，表示要进行预判。此时，先将棋子存入复 本dat為中，再判断是否胜利。

21.8n3囈麗蘿書穎赂讖蕎字

现在，.Board类的is\_win方法已经具备了预判胜利的能力。接下来就调整一下AlPlayer的 move方法，按照之.前明确的落子策略，先实现必胜落子和必救落子的处理。

调整后的代码如下：

def move(self, chess\_board):

""f,在棋盘上落子

:param chess\_board:

H H H

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\_boardomovable\_list:

if chess\_board.is\_win(self.chess, index): print ("走在 %d 位置必胜！ ! ! " % index) chess\_board.move\_down(indexz self*。*chess) return

* 2.查找地方必胜落子位置-我方必救位置 other\_chess = "0° if self.chess == "Xn else "X" for index in chess\_board.movable\_\_list:

if chess\_\_board. is\_win (other\_chess f index):

print ("敌人走在%d位置必输,火速堵上！ ° % index) chess\_boardomove\_down(indexselfochess) return

#从棋盘对象的可落子索引列表中随机选择一个索引

index = random„choice(chess\_board.movable\_list)

#在指定位置落子 — —

chess\_board。move\_down (index# self .. chess)

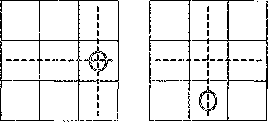
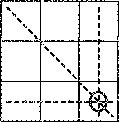
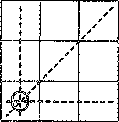
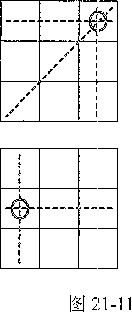
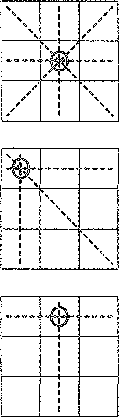
调整之后再运行程序会发现，现在计算机的落子已经有一定水平了，而且一不小心，我们 还会输给计算机。

21n8n4评傭宇刘盼値

接下来,再处理最后一种情况，就是当棋盘上既没有必胜落子位置?也没有必救落子位置时， 需要先评估所有可落子位置的子力价值，然后再选择子力价值最高的位置作为落子点。

通常要编写一个复杂的对弈类软件，在评估某个位置的子力时，需要考虑的因素还是很多的。不过，在这一小节，先给大家介绍一种最简单的评估方式——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。

我们先来看一下，在空棋盘上不同位置落子的示意图，如图所示。



棋盘落子示意图

观察图2141不难发现，在空棋盘上落子，一共有3种情况：

（1） 在中心落子，这个位置共有4个方向可能和其他棋子连接成直线，获胜的概率最高。

（2） 在角位落子，这4个位置各自有3个方向可能和其他棋子连接成直线，获胜概率中等。

（3） 在边位落子，这4个位置各自有2个方向可能和其他棋子连接成直线，获胜概率最低。 综上所述，如果计算机在落子时，既没有必胜落子位置，也没有必救落子位置时，就可以

让计算机按照胜率的高低来选择落子位置，即

（1 ）如果中心点没有棋子，则选择中心点作为落子位置；

（2） 如果中心点已有棋子，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；

（3） 如果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。

根据已经明确的思路，对代码进行调整，调整后的代码如下：

def move（selff chess\_board）:

5"在棋盘上落子

:param chess\_\_board:

n n n

print （°%s正在思考落子位置.…” % self . name）

# 「查找我方必胜落子位置 for index in chess\_board.»movable\_\_\_list: if chess\_board.is\_win（self.chessf index）:

print （n 走在 %d 位置必胜！ ! ! " % index） chess\_board.move\_down（index7 self.chess）

return

* 2。查找地方必胜落子位置-我方必救位置

other\_chess = "0" if self.chess == "X° else nX°

for index in chess\_board.movable\_list:

if chess\_board. is\_win (other\_ch.ess z index):

print 走右%d位置必臨，火速堵上！ n % index)

chess\_board.move\_down(indexz 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 ( [ 1z 37 5f 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)

调整之后再运行程序会发现，现在计算机的落子水平已经很不错了罗想要赢计算机已经相 当有难度了。

219顼同俸金

井字棋项目至此已经全部完成。我们先根据游戏性质、规则和策略进行了流程分析、类的 设计和开发阶段设计，然后逐步实现了游戏的功能。在实现过程中，又分为两个阶段：人人对 战和人机对战。通过井字棋项目的开发，达到了如下两个目的：

1. 让大家更好地体会面向对象的封装特性。封装是指按照信息屏蔽的原则，把对象同 类型的属性和操作结合在一起，构成一个独立单元模块,，通过限制对属性和操作的访问权限， 可以将属性“隐藏”在对象内部，对外提供一定的接口，在对象之外只能通过接口对对象进行 访问、操作。封装增加了对象的独立性，从而保证了数据的可靠性。
2. 弓|入一个简单的AI (人工智能)算法，实现人机对战。也借此让大家对继承和多态 特性有更好的体会。所谓继承，就是在已存在的类A的基础上建立一个新类B。类A称为基类 或父类，类B称作派生类或子类。子类从父类获得其已有的特性，这种现象称作类的基础。从 另一个角度看°从父类产生子类，称作类的派生。

.多态是基于继承的，是指一个父类的引用指向了不同的子类的实例。在父类中有的方法， 在子类中也存在，并且完成了重写，当在不同子类间进行切换的时候会有不同的结果。但要注 意的是，在Python中的应用比这个定义灵活得多，Python中的多态指在不考虑对象类型的情况 下使用对象。