A2 注释版

欢迎参加我的课！不给你答案，让你自己学会做出来！

CSC148, Assignment #2
due March 6th, 2018, 10 p.m.

overview

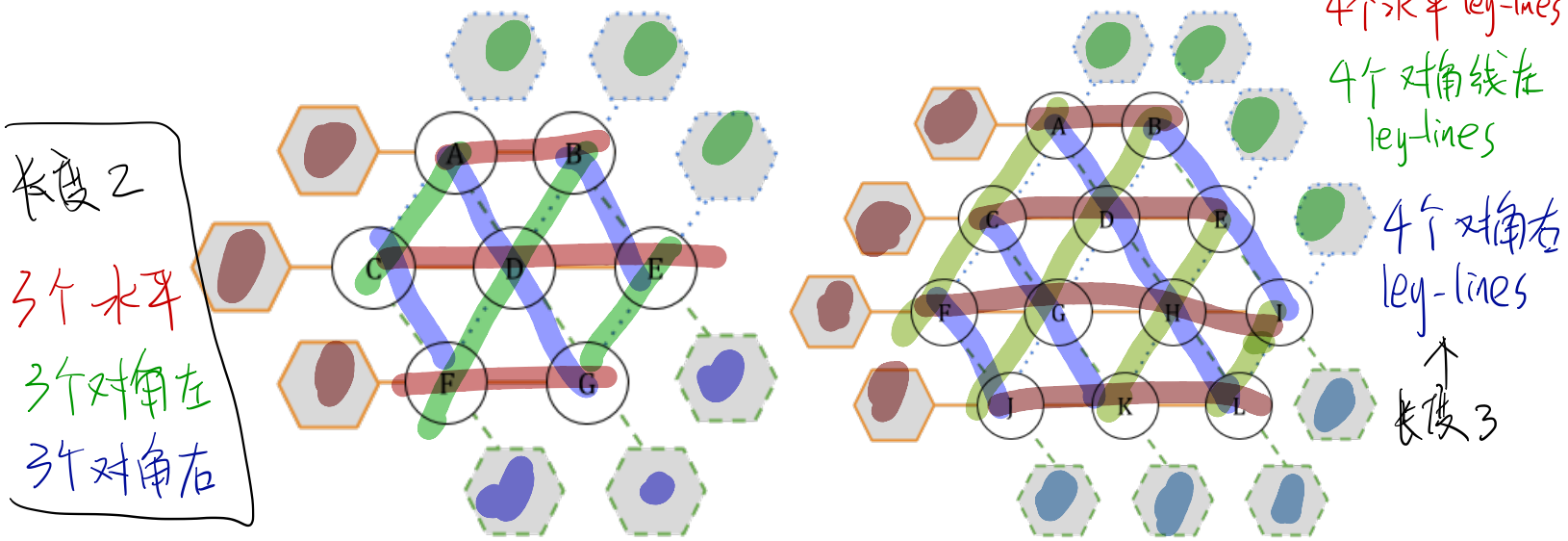
这个作业我们在A1基础上加一个新game，叫石阵

In assignment 2, you will build on the foundation laid out in assignment 1 in order to add a new game, called Stonehenge, and a new strategy, called minimax (for which you are to implement an iterative version and a recursive version). Starter code has been provided so that you should be able to "plug in" your new classes for games, and new strategies, without re-writing existing code. By the end of this assignment, you'll have a new game and a much stronger opponent to play against.

其实无论你A1做的如何，都不影响这个作业，这个作业你不用考虑design的问题，你只要写新游戏和写一个比较聪明的AI（strategy）即可，这种算法叫 minmax，说明了就是把所有可能性都尝试一遍，然后找到可以获胜的一个move

stonehenge

Stonehenge is played on a hexagonal grid formed by removing the corners from a triangular grid. Boards can have various sizes based on their side-length (the number of cells in the grid along the bottom), but are always formed in a similar manner: For side-length n, the first row has 2 cells, and each row after has 1 additional cell up until there's a row with $n + 1$ cells, after which the last row has only n cells in it.



长度2
3个水平
3个对角左
3个对角右

4个水平 ley-lines
4个对角线左 ley-lines
4个对角右 ley-lines
长度3

(a) A stonehenge grid with a side-length of 2

(b) A stonehenge grid with a side-length of 3

Players take turns claiming cells (in the diagram: circles labelled with a capital letter). When a player captures at least half of the cells in a ley-line (in the diagram: hexagons with a line connecting it to cells), then the player captures that ley-line. The first player to capture at least half of the ley-lines is the winner. A ley-line, once claimed, cannot be taken by the other player.

For your implementation of Stonehenge, you must fulfill the following specifications in order for us to test your code:

这个game有点类似于 tic-tac-toe 或者五子棋，只不过是看两个玩家轮流下了 1 以后，最后谁占领的 ley-line 更多，任何一个人首先占领了一个 ley-line 以后这个 ley-line 就归这个玩家了，看上面图

1, 2, 3, 4 各度的样子

```
length 1
01234567890
      @   @
     /   /
@ - A - B
     \ / \
  @ - C   @
       \
        @

length 2
012345678901234
        @   @
       /   /
  @ - A - B   @
     / \ / \ /
@ - C - D - E
     \ / \ / \
  @ - F - G   @
       \   \
        @   @
```

```
length 3
0123456789012345678
          @   @
         /   /
    @ - A - B   @
       / \ / \ /
  @ - C - D - E   @
     / \ / \ / \ /
@ - F - G - H - I
     \ / \ / \ / \
  @ - J - K - L   @
       \   \   \
        @   @   @

length 4
012345678901234567890123
            @   @
           /   /
      @ - A - B   @
         / \ / \ /
    @ - C - D - E   @
       / \ / \ / \ /
  @ - F - G - H - I   @
     / \ / \ / \ / \ /
@ - J - K - L - M - N
     \ / \ / \ / \ / \
  @ - O - P - Q - R   @
       \   \   \   \
        @   @   @   @
```

```
      @   @
  @   A   B   @
@   C   D   E   @
  @   F   G   @
@   @

 @@
@AB@
@CDE
@FG@
 @@
```

← 你的 git 不一定要出上面的样子，左边这种也可以

为了学校的 unittests 能正常运行，你要确保以下

- `str_to_move()` should take in a capital letter ("A", "B", etc.). 用户输入的 move 都是大写字母，正常人都不会这样做
- Cells should either be labelled with a capital letter ("A", "B", etc.) if they're unclaimed, or 1 or 2 if a player has claimed that cell. 用大写字母来代表没人占领的 cell，用 1，2 来代表被占领的
- Ley-lines should be marked with an @ if they're unclaimed, or 1 or 2 if it has been claimed. 用 @（at）来代表无人占领的 ley-line，有人占领就是 1，2
- Ley-line markers for rows must precede the cells in that row (e.g. @ - A - B as opposed to A - B - @) 水平的 ley line marker 要在这行前面
- Ley-line markers for down-left diagonals (/) are placed either: 又分两种，↙这个方向的，前2个在第一行，后面的在接下来的行最后
  - Before any rows of cells for 2 of the down-left diagonals (the ones connected to A and B)
  - Or in the row of cells above the row they?re connected to, following all of the cells in that row.
- Ley-line marks for down-right diagonals (\) are placed either: 对角右↘这个方向的，倒数第一行一个，其他的都在最后一行
  - In the very last row (after every row containing cells) in order of the cells they're connected to
  - Or in the last row of cells following the very last cell, below the row containing the cell it's connected to.

你的自己写的 str() 要满足的

看上一页的图

注意！学校没要求你一定要像 unittest 里面的 samples 那样输出也就是不用打 —，/，＼这些，空格也无所谓

minimax

Minimax is a strategy that picks the move that minimizes the possible loss for a player, working along the lines of "if my opponent were to play perfectly, which move of mine would provide the lowest score for them?." In the case of our games, where the only scores are 'winning' (i.e. a score of 1) or 'losing' (a score of -1), this becomes "is there a move that can guarantee a win no matter what the opponent does?"

For this assignment, you are to implement 2 versions of minimax: one implemented recursively, and the other implemented iteratively. minimax这个算法就是选择一个输的机率最低的 move，对于这个作业来讲，你选择的 move 应该基本是完美的，思想基本就是，假如我的对手玩的很完美，那么我走哪一步可以确保让对手分数最低，因为我们现在写的 game 没分数，只有输赢，所以分数就是 -1，1，0（0是平，其实也没有）

我们要分别用 recursion 和 loop 来做同样的事情

recursive minimax

For recursive minimax, we try to find a move that produces a "highest guaranteed score" at each step for the current player. For a game state that's over, the score is:

- 1 if the current player is the winner
- -1 if the current player is the loser
- 0 if the game is a tie

If the state isn't over, then we take a move that guarantees the "highest guaranteed score" accessible from the available moves. To do this, we get the scores that our **opponent** can get from each move, and get their **opposite** (multiply them by -1): after all, if your opponent can guarantee a score of 1 for themself (i.e a win), then that means you would be guaranteed a score of -1 (i.e. a loss).

For example, for some random game, suppose you have 3 possible moves: One that results in state A, the other in state B, and the last in state C.

Now suppose your opponent can guarantee a score of -1 for themselves from state A, a score of 1 from state B, and a score of 0 from state C. Then, equivalently, that means you can guarantee a score of 1 for yourself if you make the move to state A, -1 from state B, and 0 from state C. Thus, the "highest guaranteed score" for your current state would be 1, so you want the move that takes you to state A.

说白了，你就是把当前 state 的所有 possible moves 全都算出来一个分数（会赢 1，还是输 0），然后选择任何一个 1 的，make 那个 move 即可，当然你算一个 possible move 的分数时，要产生一个新的 state，然后这个 state 要继续算所有 possible moves 的分数，until 分出胜负，所以分出胜负的时候也就是你的 base case

用loop来做同样事情难度高不少，你肯定需要一个新的class来代表一个tree（一个possible move 连着这个move以后的每个state）

iterative minimax

你要明白，你用一个tree structure的目的不是为了真的给tree写code，你只是利用有个tree structure来记录你的children的ids（地址），也就是让每一个state知道我所有的接下来的states

For iterative minimax, the logic is similar to recursive minimax. However, you can't use recursion to go through various states: you have to use a loop of some sort, and also keep track of your states and the best values reachable from each of them. To do this, you will likely need to use a stack (to keep track of which states you've yet to explore) and a tree structure (to keep track of your various game states).

We start with our current game state and wrap it in a tree-node-like structure: this should keep track of the children (i.e. game states reachable from this one), and the "highest guaranteed score" reachable from this state. Initially, both the "highest guaranteed score" and the children should be unknown/not yet set. Then, we add it to our stack.

We progressively remove the tree-like structures from the stack. If the state is over, then the "highest guaranteed score" is:

算法详见下页中文版

- 1 if our player is the winner
- -1 if our player is the loser
- 0 if the game is a tie

If the state is not over, then we try the following:

- If there are no known children, then we make a tree-node-like structure for each of the states accessible from our available moves and set those as our children. We then add each of these to our stack **after** the parent node.
- If there are children already, then we've already seen this structure recently. Since we're in a stack, that means all of the children have been examined too, so they should all have found their "highest guaranteed score". So, we look at the scores of each of the children, and set the "highest guaranteed score" to the maximum of those.

Once the stack is empty, we've examined all possible states, and our original state should know its "highest guaranteed score" as well.

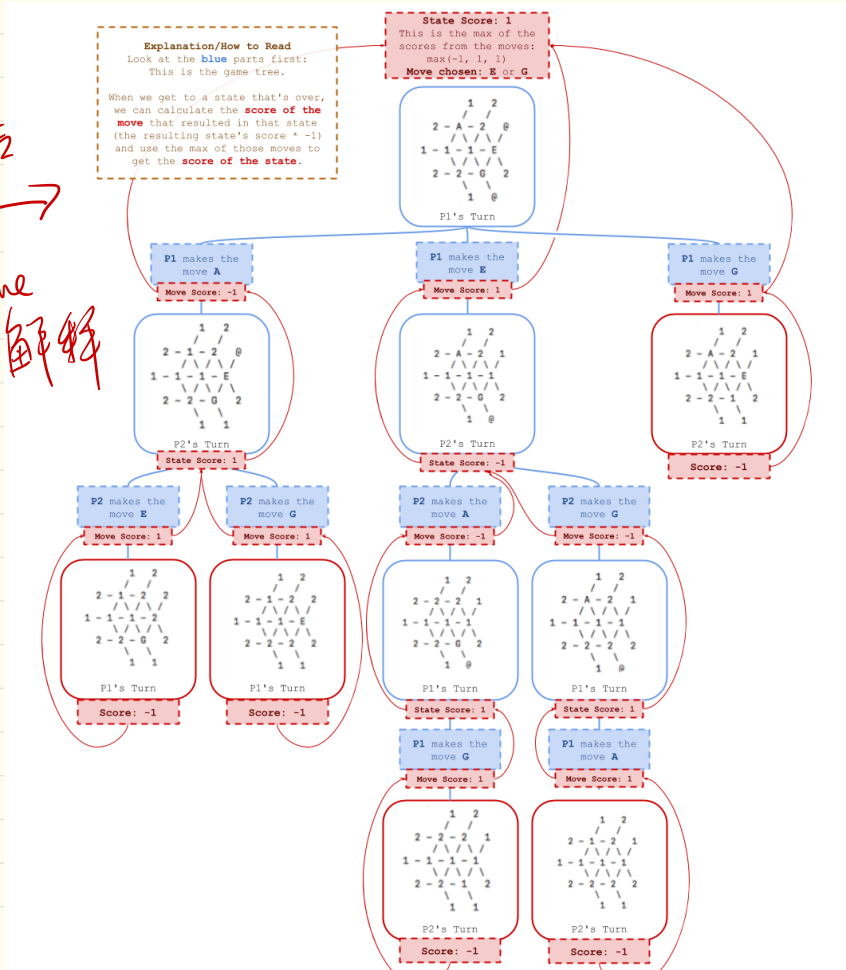
your job

- We have provided you with:
  - game.py, a superclass for games.
  - game_state.py, a superclass for game states.
  - game_interface.py, code to use your game classes.
  - strategy.py, a module for strategies.
  - subtract_square_game.py, a text-based subtract square game class, subclass of Game.
  - subtract_square_state.py, a text-based subtract square game state, subclass of GameState.
  - a2_pyta.txt, a configuration file for python_ta
  - stonehenge_unittest_basic.py, some unit tests to increase your confidence that you are on the right track with implementing the Stonehenge game.
  - minimax_unittest_basic.py, some unit tests to increase your confidence that you are on the right track implementing minimax.

loop 方法中文

以下就是把作业老师给的方法翻译成了中文，我稍微加了一点点注释

我们首先要把你的state包在(composition, wrap)一个好像tree一样的
structure里面（每一个tree node都有一个value，和children）
接下来我们把这第一个node扔到(push) stack里面（注意这个第一个node并没有children呢还）
然后我们反复的开始从stack里面pop，每一个pop出来的这个tree structure拿到了以后，
如果它包含的state已经是game over了，那就是要存上结果，
然后就可以不要了（也不是真的不要，它的id在它的parent那里呢），如果state不是结束的，
那就看看这个tree structure有没有children，有children
（也就是有一堆children的id）那就说明children都算完了，那我们就可以算这个node位置上的最大了，
如果还没有children，那就给这个state的
每一个下一个possible state都产生一个新的state，每一个新的state也都包在一个tree structure里面，
然后把所有这些children和parent都再放回
stack里面，parent要在前
一旦stack空了，就说明你已经把所有的children，所有的路线都走完了，
那你的结果也就拿到了



老师给的 Recursive 的解释

Copy these into a subdirectory of your project, where you will work on your code.

- Implement classes **StonehengeGame** (subclass of Game) and **StonehengeState** (subclass of GameState) to implement the game Stonehenge, and save them in **stonehenge.py**.
- Implement strategies **recursive_minimax_strategy(game)** and **iterative_minimax_strategy(game)** each of which provide a move that guarantees the highest possible score from the current position.
- Be sure to have:

```
if __name__ == "__main__":
    from python_ta import check_all
    check_all(config="a2_pyta.txt")
```

... at the bottom of each of the files you submit.

Submitting your work

Submit all your code on MarkUs by 10 p.m. March 6th. Click on the "Submissions" tab near the top. Click "Add a New File" and either type a file name or use the "Browse" button to choose one. Then click "Submit". You can submit a new version of a file later (before the deadline, of course).

A good strategy is to begin submitting the parts of your assignment that work **early**, and keep submitting improved versions as the deadline approaches. Only the last version of each file is graded.

appendix: more about minimax using subtract square example

Minimax is a strong strategy that assumes that both players (let's call them **A** and **B**) have all the information and time they need to make the strongest possible move from any game state (AKA position). In order to choose the strongest possible move, players need a way to evaluate the best possible outcome, called the player's score, they can guarantee from each position.

If the game is over, **A**'s score is:

- 1 if **A** wins
- -1 if **A** loses
- 0 if it's a tie

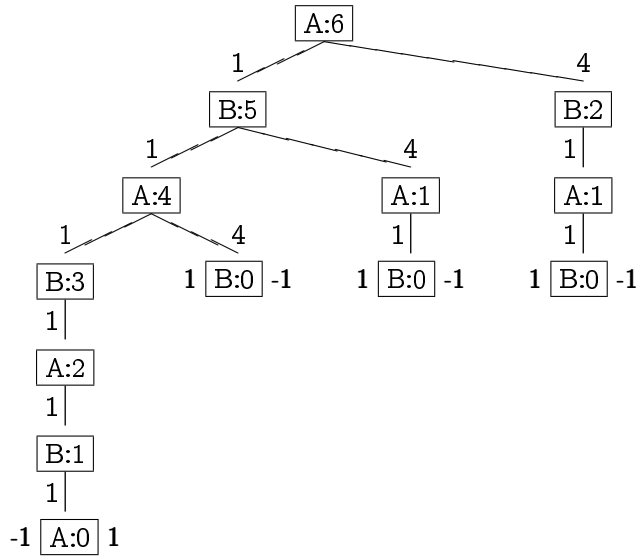
B's score is -1 times **A**'s score, since this is a zero-sum game: what's good for **A** is bad for **B**.

Suppose the game is not over, and the current player is **A**. Then **A**'s score is

- 1, if at least one **A**'s legal moves leads to a new position where **A**'s score is 1 (or **B**'s score is -1, since **B** is the current player in the new position)
- -1, if every legal move for **A** leads to a position where **A**'s score is -1 (or **B**'s score is 1, since **B** is the current player in the new position)
- 0, if at least one of **A**'s legal moves leads to a score of 0, but none lead to a score of 1.

This circular-sounding scoring process is guaranteed to provide an answer, assuming every sequence of moves eventually leads to the game ending. Indeed, it has a recursive solution, where the base case(s) corresponds to the end of the game.

Here's a diagram representing all sequences of moves for a game of Subtract Square, starting with value being 6 and current player **A**, so that position is labelled A:6. Leading out from that position are lines (edges) labelled with the possible moves, and then positions those moves lead to... and so on.



We have labelled the positions where the game is over with **A**'s score on the left, and **B**'s score on the right. Work up from those positions, writing **A**'s and **B**'s scores beside each position, until you reach the top.

You were on the right track if you ended up with a score of **1** for **A** at position A:6. You can score any position in the games we're working with using this technique. Here is a recursive algorithm for determining the score for the current position if you are the next player (the one about to play):

- If the game is over there are no moves available, your score is either 1, 0, or -1, depending on whether you win, tie, or lose.
- If the game is not over, there are legal moves available. Consider each available move, and the position it takes the game to. Determine **your opponent**'s score in the new position (after all, your opponent is the next player in the new position). Multiply that score by -1 to determine your score in that position.

  Your highest possible score among all the new positions is your score for the current position.

Of course, you will also want to record the move that gets you the highest score, so you might as well bundle them together in some suitable data structure.

Notice that you and your opponent are each solving different instances of the same problem, what your score is. That's what makes the process recursive.

其他常见问题

1. 不用考虑长度>5的游戏，你非要支持没人拦着你，学校也不test超过5的
2. str 函数你如果实在不会做，也可以用疯狂if大法，
   也就是根据长度1，2，3，4，5的不同，疯狂的列出来所有情况，疯狂的format
3. 先claim 一个ley line的人拿到这个leyline，就算后来另外一个玩家也拿到同样数量的cell
   leyline也不会易主
4. 你的strategy 和 game可以分开写，先后顺序也无所谓其实，你可以先写game，全用interactive，
   也可以先写strategy，你的strategy也可以给老师已经写好的substract square用
5. 如果你运行你的strategy时候时间很久。。别惊讶，挺正常的，但是如果1小时还停不下来，就有问题了
6. rough_outcome 不一定非要produce -1, 0, 1, 因为要的是个float，所以也可以是个在-1 和1之间的小数

王老师祝各位好运！

请自己完成作业，这是做人的基本！