This notebook is an exercise in the Intro to Game AI and Reinforcement Learning course. You can reference the tutorial at this link.

Introduction

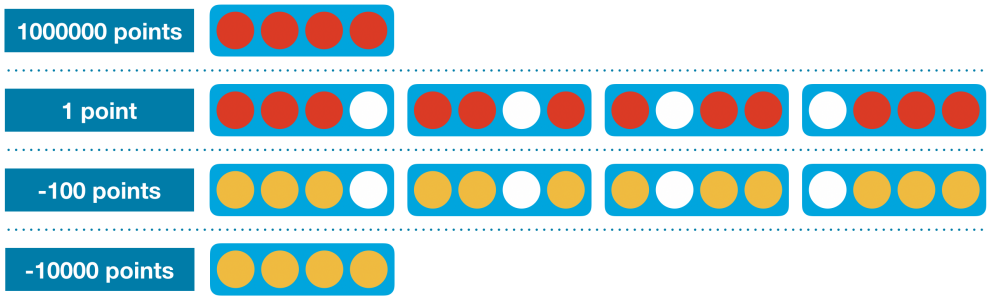
In the tutorial, you learned how to build a reasonably intelligent agent with the minimax algorithm. In this exercise, you will check your understanding and submit your own agent to the competition.

In [ ]:

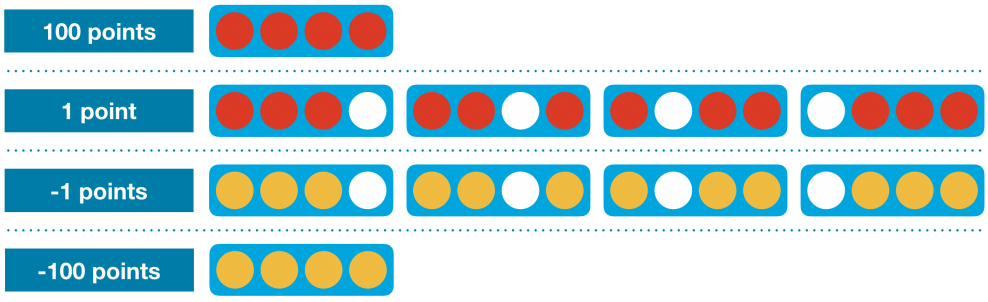
```
from learntools.core import binder
binder.bind(globals())
from learntools.game_ai.ex3 import *
```

1) A closer look

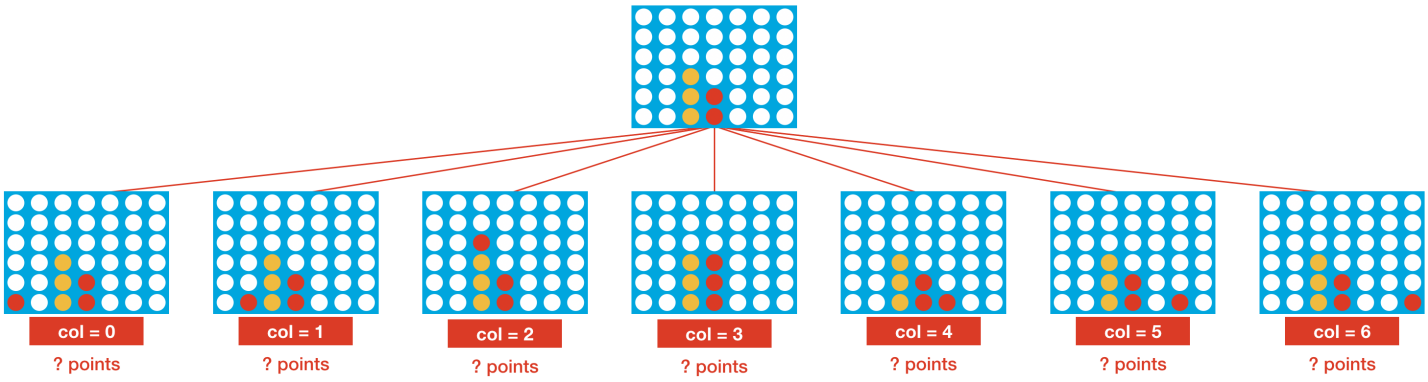
The heuristic from the tutorial looks at all groups of four adjacent grid locations on the same row, column, or diagonal and assigns points for each occurrence of the following patterns:



Is it really necessary to use so many numbers to define the heuristic? Consider simplifying it, as in the image below.



How would each heuristic score the potential moves in the example below (where, in this case, the agent looks only one step ahead)? Which heuristic would lead to the agent selecting the better move?



In [ ]:

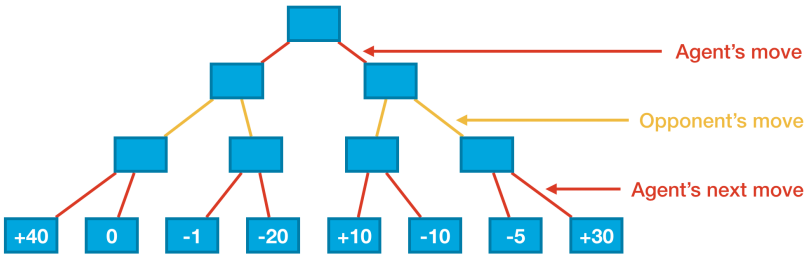
```
#q_1.hint()
```

In [ ]:

```
# Check your answer (Run this code cell to receive credit!)
q_1.solution()
```

2) Count the leaves

In the tutorial, we worked with a small game tree.



The game tree above has 8 leaf nodes that appear at the bottom of the tree. By definition, "leaf nodes" in a game tree are nodes that don't have nodes below them.

In the ConnectX competition, the game trees will be much larger!

To see this, consider a minimax agent that is trying to plan its first move, where all columns in the game board are empty. Say the agent builds a game tree of depth 3. How many leaf nodes are in the game tree?

Use your answer to fill in the blank below.

In [ ]:

```
# Fill in the blank
num_leaves = 7*7*7
# because there are 7 columns

# Check your answer
q_2.check()
```

In [ ]:

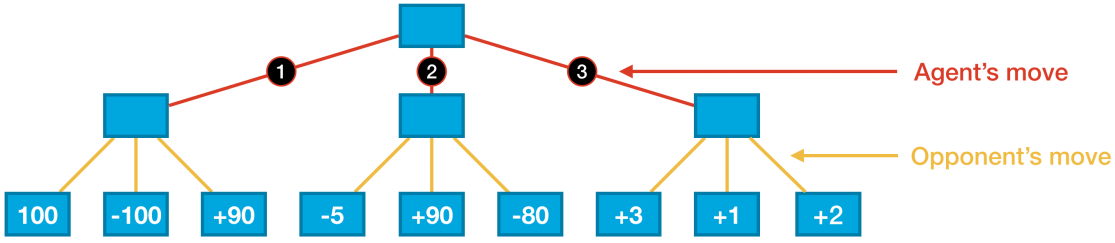
```
# Lines below will give you a hint or solution code
#q_2.hint()
#q_2.solution()
```

3) Which move will the agent select?

In this question, you'll check your understanding of the minimax algorithm. Remember that with this algorithm,

> **The agent chooses moves to get a score that is as high as possible, and it assumes the opponent will counteract this by choosing moves to force the score to be as low as possible.**

Consider the toy example below of a game tree that the agent will use to select its next move.



Which move will the agent select? Use your answer to set the value of the `selected_move` variable below. Your answer should be one of `1`, `2`, or `3`.

In [ ]:

```
# Fill in the blank
selected_move = 3

# Check your answer
q_3.check()
```

In [ ]:

```
# Lines below will give you a hint or solution code
#q_3.hint()
#q_3.solution()
```

4) Examine the assumptions

The minimax agent assumes that its opponent plays optimally (with respect to the heuristic, and using a game tree of limited depth). But this is almost never the case, in practice: it's far more likely for the agent to encounter a suboptimal (that is: worse than optimal) opponent.

Say the minimax agent encounters a suboptimal opponent. Should we expect the minimax agent to still play the game well, despite the contradiction with its assumptions? If so, why?

In [ ]:

```
#q_4.hint()
```

In [ ]:

```
# Check your answer (Run this code cell to receive credit!)
q_4.solution()
```

5) Submit to the competition

Now, it's time to submit an agent to the competition! Use the next code cell to define an agent. (You can see an example of how to write a valid agent in this notebook.)

If you decide to use the minimax code from the tutorial, you might like to add alpha-beta pruning to decrease the computation time (i.e., get the minimax algorithm to run much faster!). In this case, "alpha" and "beta" to refer to two values that are maintained while the algorithm is running, that help to identify early stopping conditions.

Without alpha-beta pruning, minimax evaluates each leaf node. With alpha-beta pruning, minimax only evaluates nodes that could provide information that affects the agent's choice of action. Put another way, it identifies nodes that could not possibly affect the final result and avoids evaluating them.

In [ ]:

```
# How deep to make the game tree: higher values take longer to run!
N_STEPS = 3

def agent(obs, config):
    # Get list of valid moves
    valid_moves = [c for c in range(config.columns) if obs.board[c] == 0]
    # Convert the board to a 2D grid
    grid = np.asarray(obs.board).reshape(config.rows, config.columns)
    # Use the heuristic to assign a score to each possible board in the next step
    scores = dict(zip(valid_moves, [score_move(grid, col, obs.mark, config, N_STEPS) for
col in valid_moves]))
    # Get a list of columns (moves) that maximize the heuristic
    max_cols = [key for key in scores.keys() if scores[key] == max(scores.values())]
    # Select at random from the maximizing columns
    return random.choice(max_cols)
```

In [ ]:

```
# Run this code cell to get credit for creating an agent
q_5.check()
```

In [ ]:

```
import inspect
import os

def write_agent_to_file(function, file):
    with open(file, "a" if os.path.exists(file) else "w") as f:
        f.write(inspect.getsource(function))
        print(function, "written to", file)

write_agent_to_file(my_agent, "submission.py")
```

Then, follow these steps to submit your agent to the competition:

1. **Begin by clicking on the blue Save Version button in the top right corner of the window. This will generate a pop-up window.**
2. **Ensure that the Save and Run All option is selected, and then click on the blue Save button.**
3. **This generates a window in the bottom left corner of the notebook. After it has finished running, click on the number to the right of the Save Version button. This pulls up a list of versions on the right of the screen. Click on the ellipsis (...) to the right of the most recent version, and select Open in Viewer. This brings you into view mode of the same page. You will need to scroll down to get back to these instructions.**
4. **Click on the Output tab on the right of the screen. Then, click on the file you would like to submit, and click on the blue Submit button to submit your results to the leaderboard.**

You have now successfully submitted to the competition!

If you want to keep working to improve your performance, select the blue Edit button in the top right of the screen. Then you can change your code and repeat the process. There's a lot of room to improve, and you will climb up the leaderboard as you work.

Go to "My Submissions" to view your score and episodes being played.

Keep going

Move on to learn how to use deep reinforcement learning to develop an agent without a heuristic!

Have questions or comments? Visit the Learn Discussion forum to chat with other Learners.