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

Introduction

You have seen how to define a random agent. In this exercise, you'll make a few improvements.

To get started, run the code cell below to set up our feedback system.

In [ ]:

```
from learntools.core import binder
binder.bind(globals())
from learntools.game_ai.ex1 import *
```

1) A smarter agent

We can improve the performance without devising a complicated strategy, simply by selecting a winning move, if one is available.

In this exercise, you will create an agent that:

- **selects the winning move, if it is available. (*If there is more than one move that lets the agent win the game, the agent can select any of them.*)**
- **Otherwise, it should select a random move.**

To help you with this goal, we provide some helper functions in the code cell below.

In [ ]:

```
import numpy as np

# Gets board at next step if agent drops piece in selected column
def drop_piece(grid, col, piece, config):
    next_grid = grid.copy()
    for row in range(config.rows-1, -1, -1):
        if next_grid[row][col] == 0:
            break
    next_grid[row][col] = piece
    return next_grid

# Returns True if dropping piece in column results in game win
def check_winning_move(obs, config, col, piece):
    # Convert the board to a 2D grid
    grid = np.asarray(obs.board).reshape(config.rows, config.columns)
    next_grid = drop_piece(grid, col, piece, config)
    # horizontal
    for row in range(config.rows):
        for col in range(config.columns-(config.inarow-1)):
            # `config.inarow` - number of pieces a player needs to get in a row in order
to win (`4` for Connect Four)

            window = list(next_grid[row,col:col+config.inarow])
            if window.count(piece) == config.inarow:
                return True
    # vertical
    for row in range(config.rows-(config.inarow-1)):
        for col in range(config.columns):
            window = list(next_grid[row:row+config.inarow,col])
            # bext_grid[range(row, row + config.inarow, col)]
            print(window)
            if window.count(piece) == config.inarow:
```

```
                return True
    # positive diagonal
    for row in range(config.rows-(config.inarow-1)):
        for col in range(config.columns-(config.inarow-1)):
            window = list(next_grid[range(row, row+config.inarow), range(col, col+config
.inarow)])
            if window.count(piece) == config.inarow:
                return True
    # negative diagonal
    for row in range(config.inarow-1, config.rows):
        for col in range(config.columns-(config.inarow-1)):
            window = list(next_grid[range(row, row-config.inarow, -1), range(col, col+co
nfig.inarow)])
            if window.count(piece) == config.inarow:
                return True
    return False
```

The `check_winning_move()` function takes four required arguments: the first two (`obs` and `config`) should be familiar, and:

- **`col` is any valid move**
- **`piece` is either the agent's mark or the mark of its opponent.**

The function returns `True` if dropping the piece in the provided column wins the game (for either the agent or its opponent), and otherwise returns `False` . To check if the agent can win in the next move, you should set `piece=obs.mark` .

To complete this exercise, you need to define `agent_q1()` in the code cell below. To do this, you're encouraged to use the `check_winning_move()` function.

The `drop_piece()` function (defined in the code cell above) is called in the `check_winning_move()` function. Feel free to examine the details, but you won't need a detailed understanding to solve the exercise.

In [ ]:

```
import random

def agent_q1(obs, config):
    valid_moves = [col for col in range(config.columns) if obs.board[col] == 0]
    print(valid_moves)
    for valid_move in valid_moves:
        print(valid_move)
        print("obs mark ", obs.mark)
        if check_winning_move(obs, config, valid_move,obs.mark):
            return valid_move
    # Your code here: Amend the agent!
    return random.choice(valid_moves)

# Check your answer
q_1.check()
```

In [ ]:

```
# Lines below will give you a hint or solution code
#q_1.hint()
#q_1.solution()
```

2) An even smarter agent

In the previous question, you created an agent that selects winning moves. In this problem, you'll amend the code to create an agent that can also block its opponent from winning. In particular, your agent should:

- **Select a winning move, if one is available.**
- **Otherwise, it selects a move to block the opponent from winning, if the opponent has a move that it can play in its next turn to win the game.**
- **If neither the agent nor the opponent can win in their next moves, the agent selects a random move.**

To help you with this exercise, you are encouraged to start with the agent from the previous exercise.

To check if the opponent has a winning move, you can use the `check_winning_move()` function, but you'll need to supply a different value for the `piece` argument.

In [ ]:

```
def agent_q2(obs, config):
    valid_moves = [col for col in range(config.columns) if obs.board[col] == 0]

    print(valid_moves)
    for valid_move in valid_moves: # if there is a winning turn
        print(valid_move)
        print('obs mark ',obs.mark)
        print('obs.mark%2+1 ', obs.mark%2+1)
        if check_winning_move(obs, config, valid_move,obs.mark) or \
        check_winning_move(obs, config, valid_move,obs.mark%2+1):
            return valid_move

    # Your code here: Amend the agent!
    return random.choice(valid_moves)

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

In [ ]:

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

3) Looking ahead

So far, you have encoded an agent that always selects the winning move, if it's available. And, it can also block the opponent from winning.

You might expect that this agent should perform quite well! But how is it still possible that it can still lose the game?

In [ ]:

```
#q_3.hint()
```

In [ ]:

```
# Check your answer (Run this code cell to receive credit!)
q_3.solution()
```

4) Play against an agent

Amend the `my_agent()` function below to create your own agent. Feel free to copy an agent that you created above.

Note that you'll have to include all of the necessary imports and helper functions. For an example of how this would look with the first agent you created in the exercise, take a look at this notebook.

In [ ]:

```
def my_agent(obs, config):
    valid_moves = [col for col in range(config.columns) if obs.board[col] == 0]

    print(valid_moves)
    for valid_move in valid_moves: # if there is a winning turn
        print(valid_move)
        print('obs mark ',obs.mark)
        print('obs.mark%2+1 ', obs.mark%2+1)
```

```
        if check_winning_move(obs, config, valid_move,obs.mark) or \
        check_winning_move(obs, config, valid_move,obs.mark%2+1):
            return valid_move

    # Your code here: Amend the agent!
    return random.choice(valid_moves)
```

In [ ]:

```
# Run this code cell to get credit for creating an agent
q_4.check()
```

Run the next code cell to play a game round against the agent. To select a move, click on the game screen in the column where you'd like to place a disc.

After the game finishes, you can re-run the code cell to play again!

In [ ]:

```
from kaggle_environments import evaluate, make, utils

env = make("connectx", debug=True)
env.play([my_agent, None], width=500, height=450)
```

5) Submit to the competition

Now, it's time to make your first submission to the competition! Run the next code cell to write your agent to a submission file.

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")

# Check that submission file was created
q_5.check()
```

Then, follow these steps:

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 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

Keep going

Learn how to use heuristics to improve your agent.

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