Gaming AI & Reinforcement Learning: 1st lesson – Play the Game

*Connect Four* is a game where two players alternate turns dropping colored discs into a vertical grid. Each player uses a different color (usually red or yellow), and the objective of the game is to be the first player to get four discs in a row.



In this course, you’ll build your own intelligent agents to play the game:

* In the first lesson, you’ll learn how to set up the game environment and create your first agent.
* The next two lessons focus on traditional methods for building game AI. These agents will be smart enough to defeat many novice players!
* In the final lesson, you’ll experiment with cutting-edge algorithms from the field of reinforcement learning. The agents that you build will come up with gameplay strategies much like humans do: gradually, and with experience.

***Getting started***

The game environment comes equipped with agents that have already been implemented for you. To see a list of these default agents, run.

from kaggle\_environments import make, evaluate

*# Create the game environment*

*# Set debug=True to see the errors if your agent refuses to run*

env = make("connectx", debug=True)

*# List of available default agents*

print(list(env.agents))

Loading environment lux\_ai\_s2 failed: No module named 'vec\_noise'

['random', 'negamax']

The "random" agent selects (uniformly) at random from the set of **valid moves**. In *Connect Four*, a move is considered valid if there's still space in the column to place a disc (i.e., if the board has seven rows, the column has fewer than seven discs).

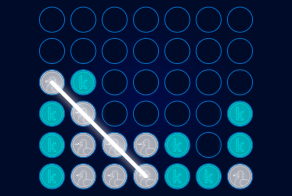
In the code cell below, this agent plays one game round against a copy of itself.

*# Two random agents play one game round*

env.run(["random", "random"])

*# Show the game*

env.render(mode="ipython")



You can use the player above to view the game in detail: every move is captured and can be replayed. Try this now! As you'll soon see, this information will prove incredibly useful for brainstorming ways to improve our agents.

***Defining agents***

To participate in the competition, you’ll create your own agents. Your agents should be implemented as a Python function that accepts two arguments: obs and config. It returns an integer with the selected column, where indexing starts at zero. So, the returned value is one of 0-6, inclusive.

We’ll start with a few examples, to provide some context. In the code cell below:

* The first agent behaves identically to the “random” agent above.
* The second agent always selects the middle column, whether it’s valid or not! Note that if any agent selects an invalid move, it loses the game.
* The third agent selects the leftmost valid column.

import random

import numpy as np

# Selects random valid column

def agent\_random(obs, config):

valid\_moves = [col for col in range(config.columns) if obs.board[col] == 0]

return random.choice(valid\_moves)

# Selects middle column

def agent\_middle(obs, config):

return config.columns//2

# Selects leftmost valid column

def agent\_leftmost(obs, config):

valid\_moves = [col for col in range(config.columns) if obs.board[col] == 0]

return valid\_moves[0]

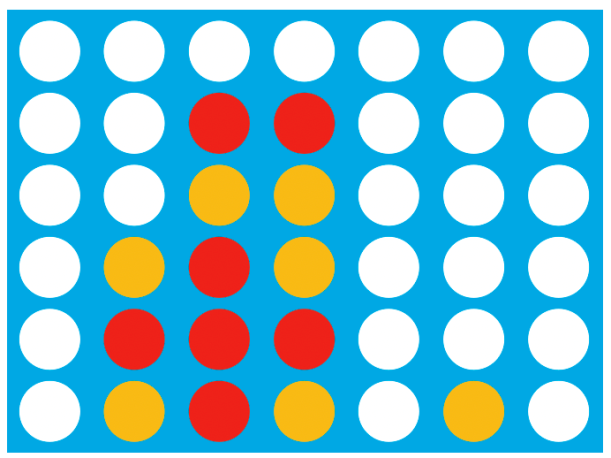
So, what are obs and config, exactly?

* obs

obs contains two pieces of information:

* obs.board

The game board (a Python list w/one item for each grid location). obs.board is a Python list that shows the locations of the discs, where the first row appears first, followed by the second row, and so on. We use 1 to track player 1's discs, and 2 to track player 2's discs. For instance, for this game board:



obs.board would be [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 1, 2, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 2, 1, 2, 0, 2, 0].

* obs.mark

The piece assigned to the agent (either 1 or 2).

* config

config contains three pieces of information:

* config.columns

Number of columns in the game board (7 for *Connect Four*).

* config.rows

Number of rows in the game board (6 for *Connect Four*).

* config.inarow

Number of pieces a player needs to get in a row in order to win (4 for *Connect Four*).

Take the time now to investigate the three agents we've defined above. Make sure that the code makes sense to you!

***Evaluating agents***

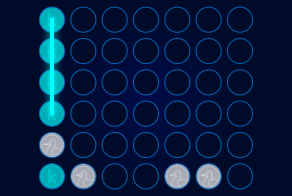
To have the custom agents play one game round, we use the same env.run() method as before.

*# Agents play one game round*

env.run([agent\_leftmost, agent\_random])

*# Show the game*

env.render(mode="ipython")



The outcome of a single game is usually not enough information to figure out how well our agents are likely to perform. To get a better idea, we'll calculate the win percentages for each agent, averaged over multiple games. For fairness, each agent goes first half of the time.

To do this, we'll use the get\_win\_percentages() function.

def get\_win\_percentages(agent1, agent2, n\_rounds=100):

*# Use default Connect Four setup*

config = {'rows': 6, 'columns': 7, 'inarow': 4}

*# Agent 1 goes first (roughly) half the time*

outcomes = evaluate("connectx", [agent1, agent2], config, [], n\_rounds//2)

*# Agent 2 goes first (roughly) half the time*

outcomes += [[b,a] for [a,b] **in** evaluate("connectx", [agent2, agent1], config, [], n\_rounds-n\_rounds//2)]

print("Agent 1 Win Percentage:", np.round(outcomes.count([1,-1])/len(outcomes), 2))

print("Agent 2 Win Percentage:", np.round(outcomes.count([-1,1])/len(outcomes), 2))

print("Number of Invalid Plays by Agent 1:", outcomes.count([None, 0]))

print("Number of Invalid Plays by Agent 2:", outcomes.count([0, None]))

Which agent do you think performs better against the random agent: the agent that always plays in the middle (agent\_middle), or the agent that chooses the leftmost valid column (agent\_leftmost)? Let's find out!

get\_win\_percentages(agent1=agent\_middle, agent2=agent\_random)

get\_win\_percentages(agent1=agent\_leftmost, agent2=agent\_random)

Agent 1 Win Percentage: 0.65

Agent 2 Win Percentage: 0.0

Number of Invalid Plays by Agent 1: 35

Number of Invalid Plays by Agent 2: 0

Agent 1 Win Percentage: 0.82

Agent 2 Win Percentage: 0.18

Number of Invalid Plays by Agent 1: 0

Number of Invalid Plays by Agent 2: 0

It looks like the agent that chooses the leftmost valid column performs best!