# Capstone Project Proposal

For Udacity's Machine Learning Engineer nanodegree program

## Reinforcement Learning for the ConnectX Competition at Kaggle

- The aim of this project is to design and submit a Machine Learning agent to the ConnectX competition at Kaggle.
- The final evaluation metric will be to place a reinforcement-learning model among the top 5% of participants in the on-going competition there.

### Domain Background

Reinforcement learning is a method of training neural network agents against similar neural network opponents within a commonly defined environment.
For this specific competition at Kaggle, a custom environment using OpenAI stable baselines is required for game play between agents.

### Problem Statement

The neural network agent must learn to play the classic board game "Connect 4", where two players drop different colored discs into a grid with six rows and seven columns until either they tie (when the grid is filled without a winner), or one player wins by placing four discs of their color in a row (horizontally, vertically, or diagonally)

There are some classic "heuristic" approaches to solving this game computationally, and a selection of these must be customized and applied to the game environment as it is defined for this competition.
The first problem will be to build heuristic agents that perform well against other such agents in open competition (to then be used as benchmarks for the performance of the reinforcement-learning model).
The next problem will then be to design and train a neural network model: first, just to learn to play the game; and then, to learn to play the game *well* in open competition against real-world opponents.
The final aim is to build (submitable) neural network models, training them with reinforcement learning until at least one model is competitive enough to place among the top 5% of other submitted agents.

## Datasets and Inputs

The input data will be the "score" for each step the model takes during training. These scores are calculated and presented to the model in the stable-baselines environment, which is at the moment opaque to me. But these are the data on which the model will actually be trained.
Data generated by the agents during competition will also be used as metrics for evaluation of the agents' performance.

## Solution Statement and Benchmark Model

To achieve competitive heuristic models for the NN to train against, the minimax algorithm with alphabeta pruning (and other feature implementations) will be developed and evaluated.

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cutoff *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if β ≤ α then
                break (* α cutoff *)
        return value
```

The NN itself will be a convolutional PP01 model, or possibly a keras model if it can be constructed to interface with the stable-baselines environment.
The final benchmark for the model's success will be how it performs in live competition against other agents.

## Evaluation Metrics

In the final evaluation:

- Placement among the top 5% on the leaderboard for this competition.

At least two other metrics will be used to that end:

- The scoring of each *move* made during training, customizable in the stable-baselines environment.
- Percentage of *wins* in evaluation against heuristic-agent opponents.

**Project Design**

1. Submit a heuristic agent to the game.
2. Implement optimizations to heuristic agents until they are competitive (top 20%).
3. Build a PPO1 convolutional model and train it in a customized learning environment against other agents, and against itself (to avoid "overfitting" it to the training agents).
4. Hand-tune the training hyperparameters and the model design.
5. Consider how to implement a keras model to work in this game environment.