W251 – Deep Learning in the Cloud and at the Edge

Q-Learning Our Way to Pentago Glory

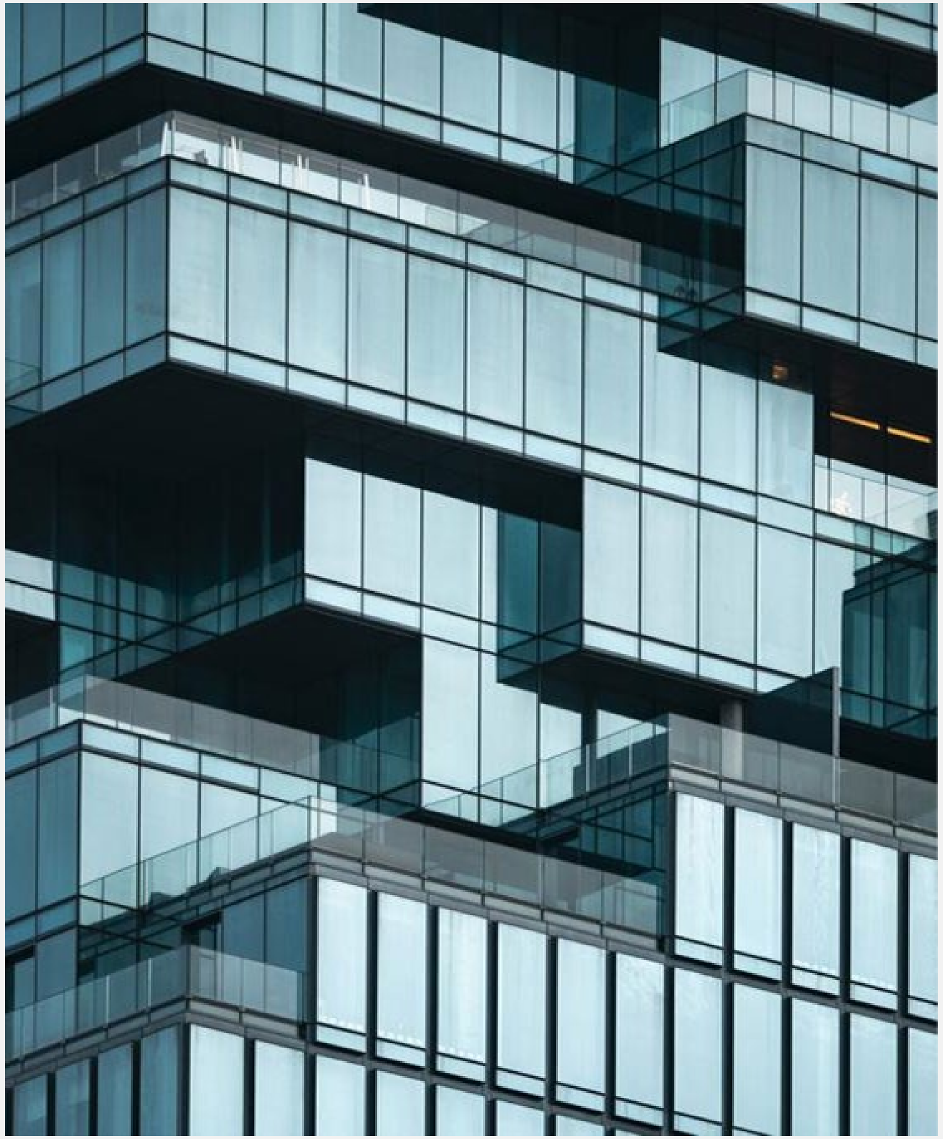
November 30, 2020

Gaustad, Oropeza, Robertson

Agenda

- Project Goal
- What exactly is Pentago?
- Q-Tables: The Simpleton’s Way to Winning
- Deep Q-Learning: A Glimpse of the Promised Land
- Closing

Project Goal



<u>Goal:</u> Use reinforcement learning to inform an agent that can consistently beat an opposing agent that plays randomly.

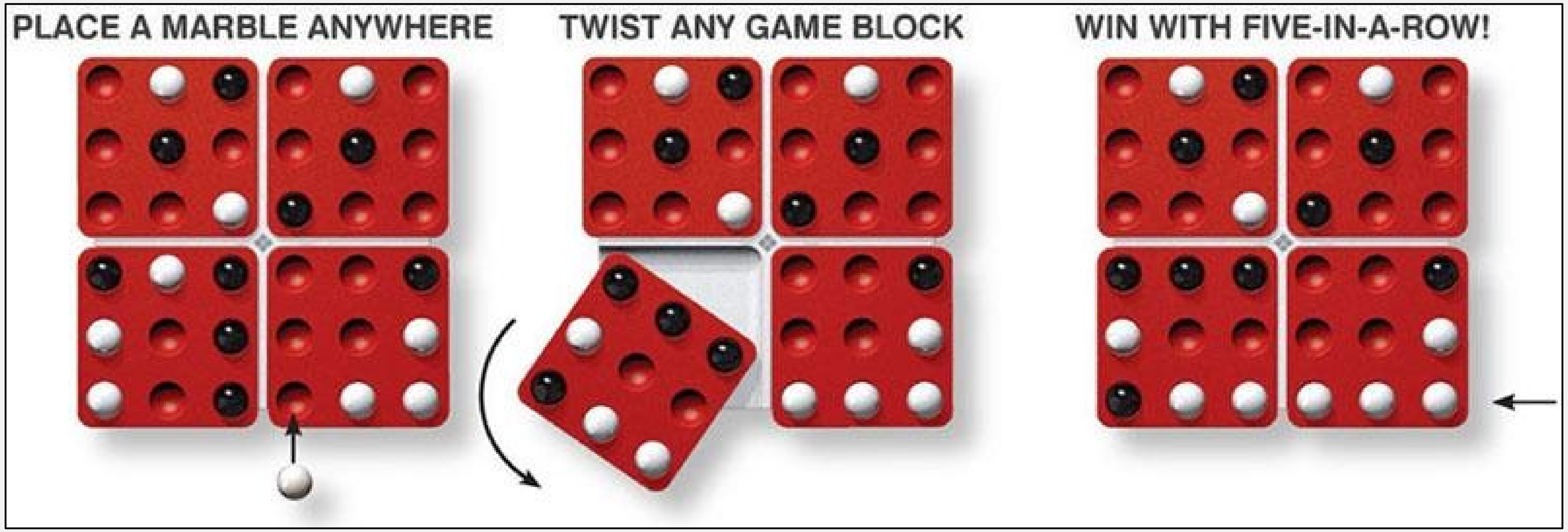
As a demonstration platform, we have chosen the game Pentago. It has similar strategy characteristics to other games (chess, go, etc.), but is less well-known and more amenable to our project.

What Exactly is Pentago?

Pentago

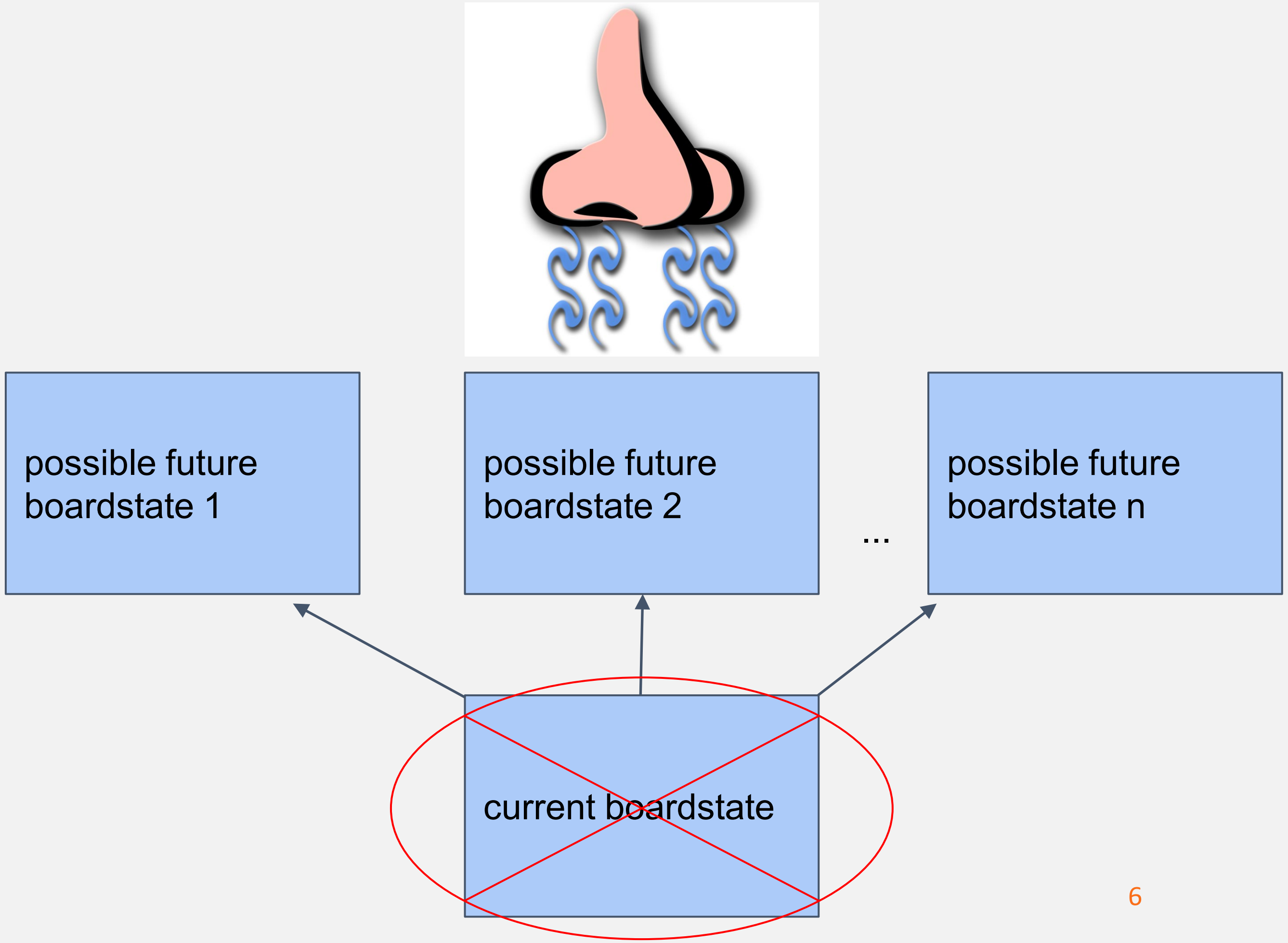
Pentago: Connect 4 for Adults

- 2 players
- 6 x 6 grid, made up of 4 board segments (3 x 3) that can rotate
- 3,009,081,623,421,558 (3×10^{15}) possible board states
- Players place one marble and rotate one section 90° each turn
- First player with 5 marbles in a row wins



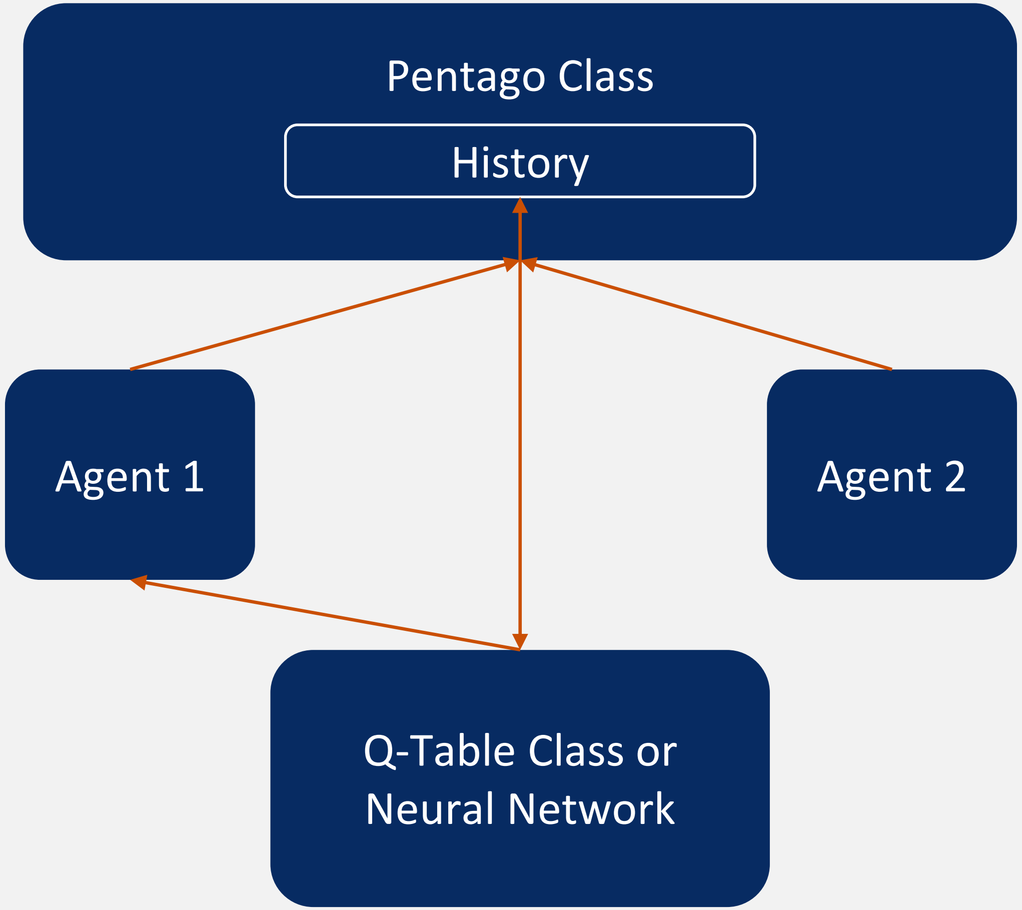
General Q-Learning Principles

- Robotics vs Strategy Game
  - Continuous vs discrete state spaces and action spaces
  - Q(s,a) > given from a Number of possible states x Number of possible actions q table.
- Abstracted away the action space, we only care about the future boardstates
  - Sniff Test or Q value
- Need to develop an intelligence for determining the best boardstate from a list



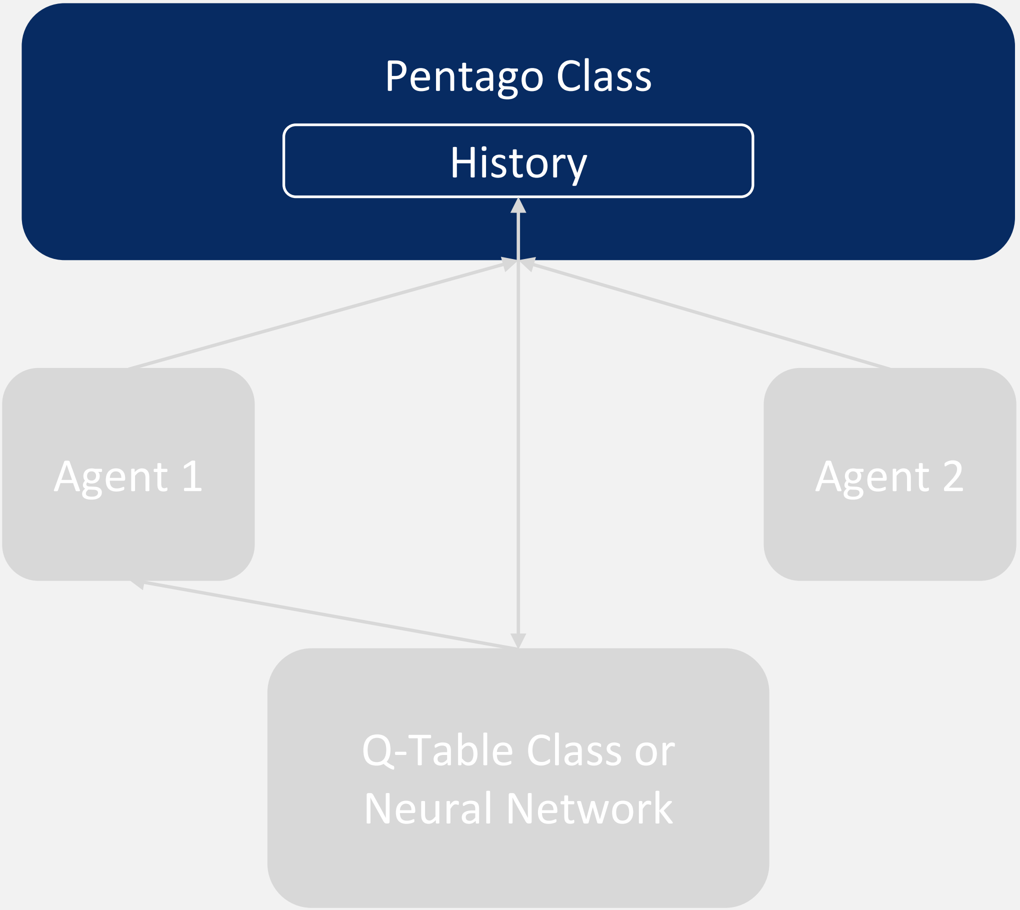
Overall Project Design

- **Elements:**
  - Pentago class – Simulates game movements
  - Agents – Entities that play the game
    - Begin playing randomly
    - After some time, Agent 1 will reference the Q-Table
  - Q-Table class – Uses history attribute of the Pentago class to update Q value associated with possible future board states



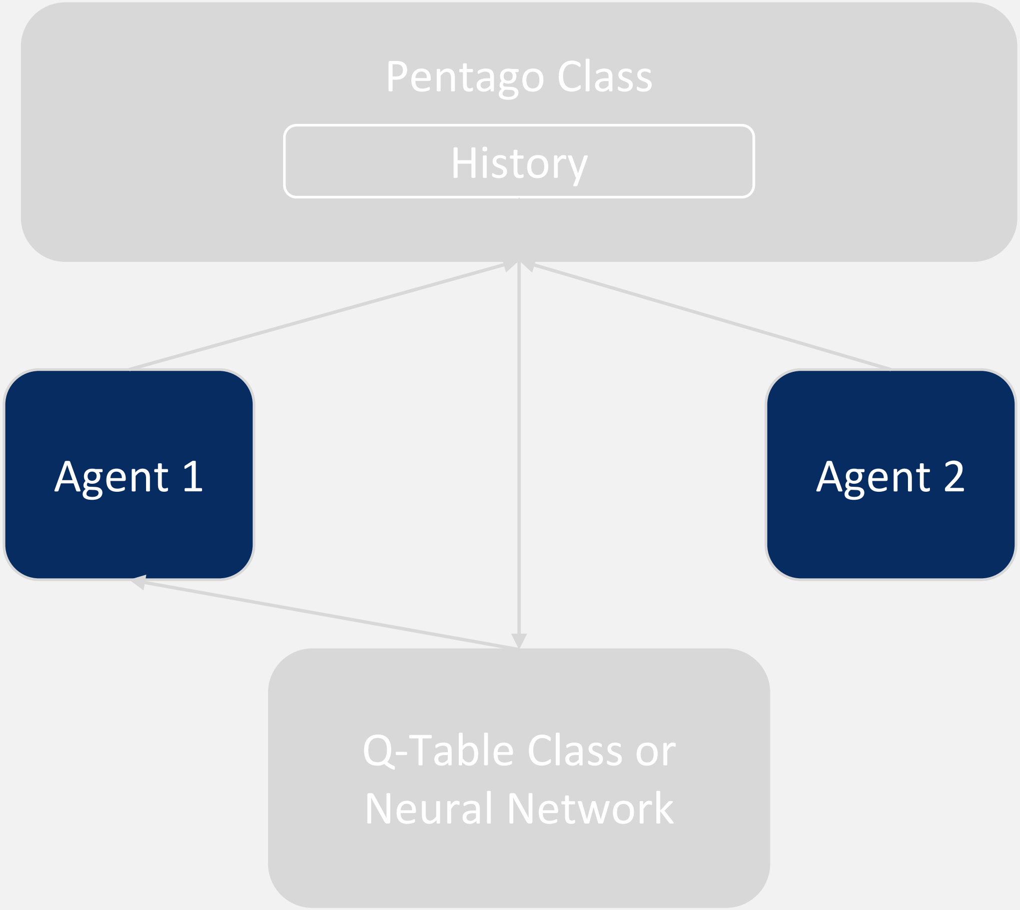
Step 1: Coding the Game

- <u>**Pentago Class**</u>
  - Keeps board state
  - Saves game history
  - Checks for a winner or a full board
  - Allows a “full move”
    - Placing a marble and rotating a quadrant
- Has robust error handling to ensure that moves are legal and that the board still has space to play



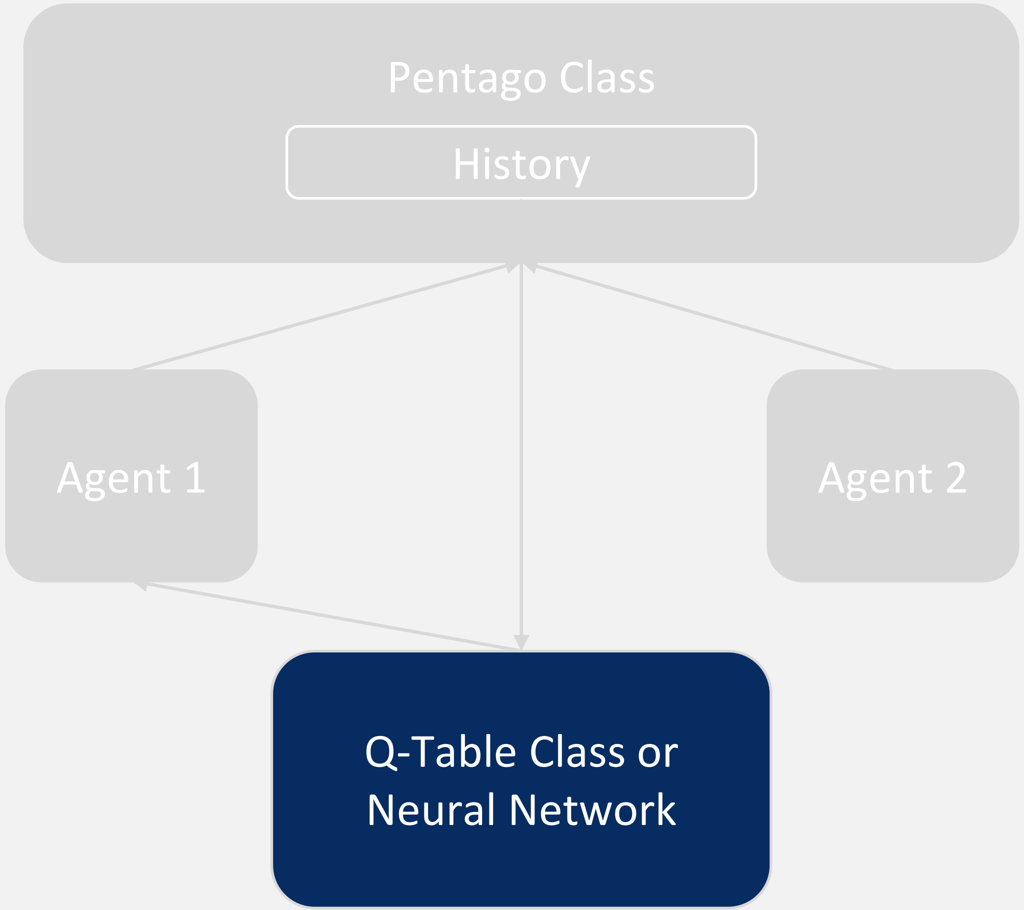
Step 2: Coding the Agent

- **<u>Agent Class</u>**
  - Gets available moves
  - Gets possible next board states
    - Can be quite long (~100 board states early in the game)
  - Makes a move
    - Initially picks a random placement and rotation
    - After a certain amount of time, Agent 1 will start referencing the Q-Table to assess the best next move



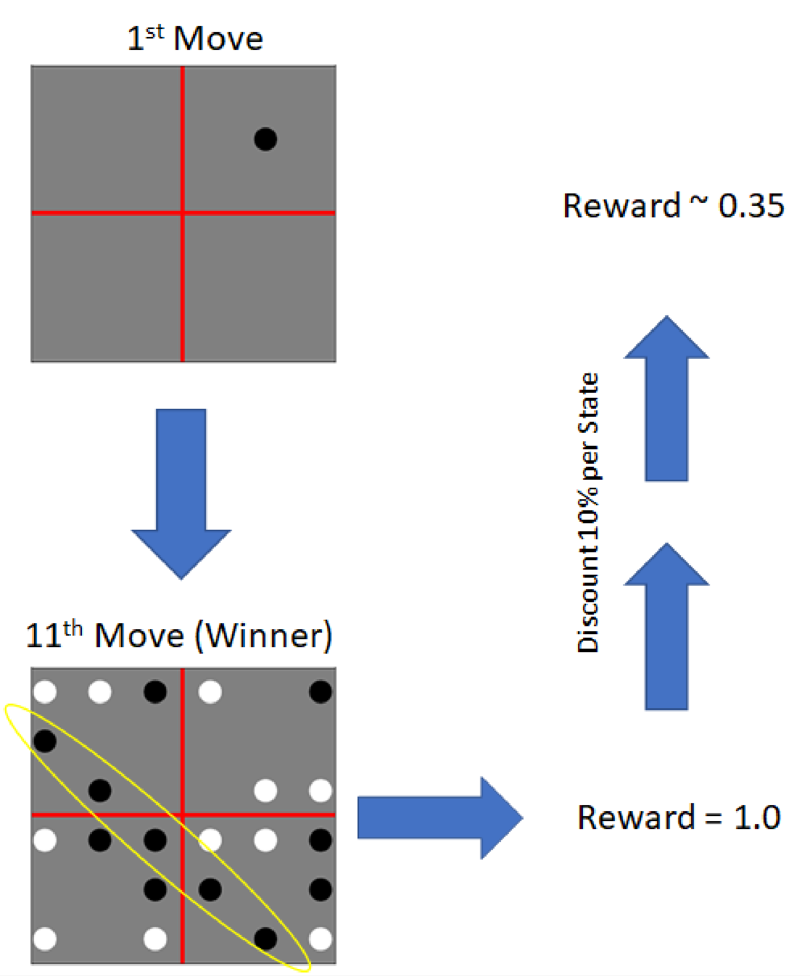
Step 3: Q-Table / Neural Network

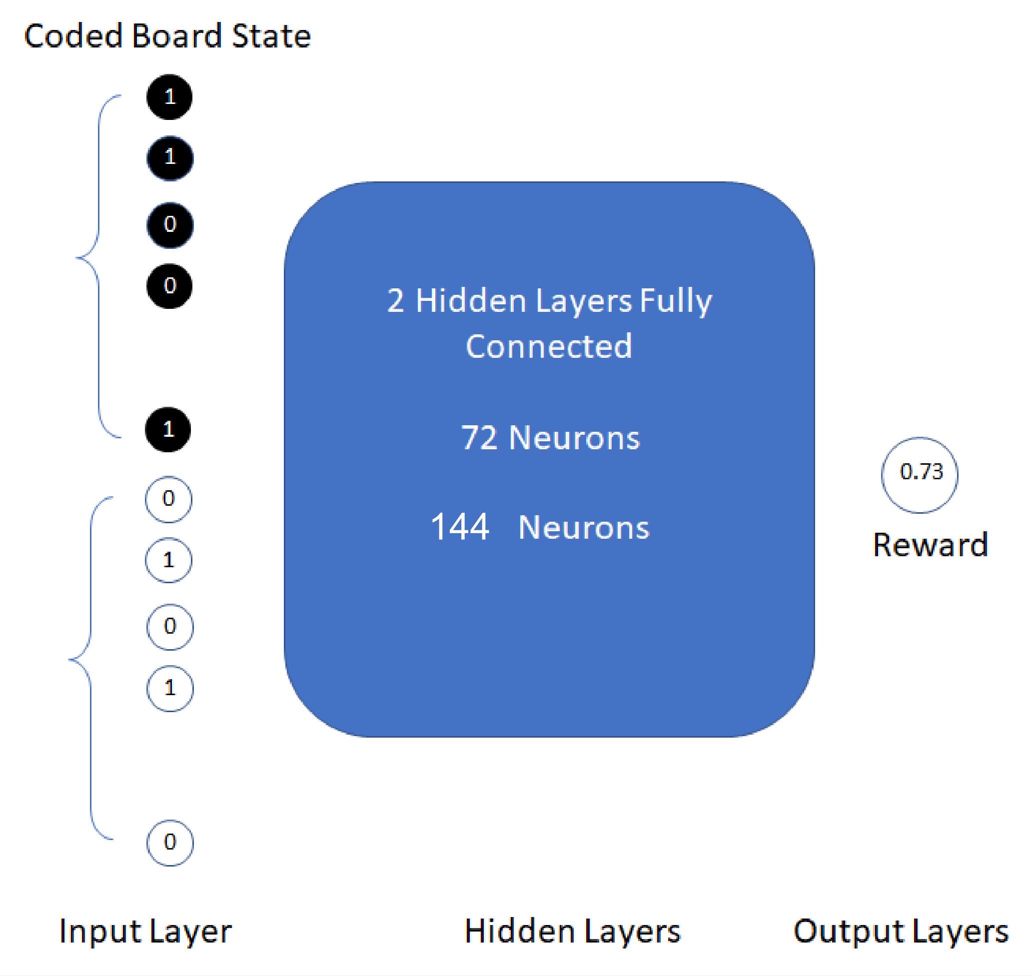
- **2 Possible Approaches to intelligent moves**
  - **<u>Q-Table Class</u>**
    - Saves each board state, then assigns a Q-Value for the next board state that follows
    - Q-Values are between -1 and 1
    - Updates Q-Values post game using the History attribute from the Pentago Class
      - For each game won, assigns a 1 (or -1, depending on the player who won) to the final board state, then decays the reward by 10% for each prior board state
        - Rewards are 1, .9. 81, etc.
  - **<u>Neural Network</u>**
    - Uses a neural network in place of a Q-table
    - Agent passes NN all possible future board states
    - NN determines the highest q-value of the possible future board states and makes the corresponding move



How we Implement Neural Networks

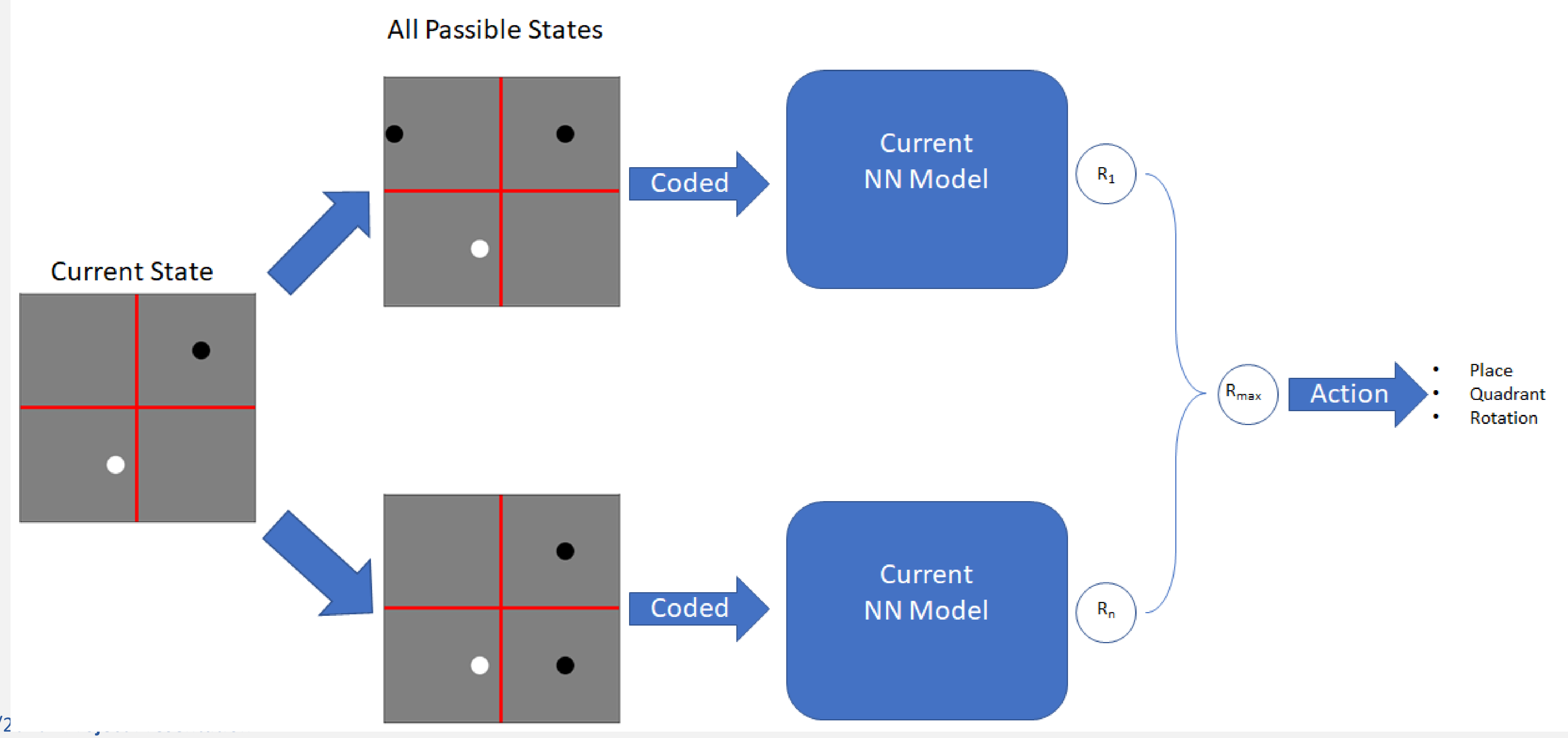
- For Neural Network model update, the input batch consist all states and rewards for 100 games





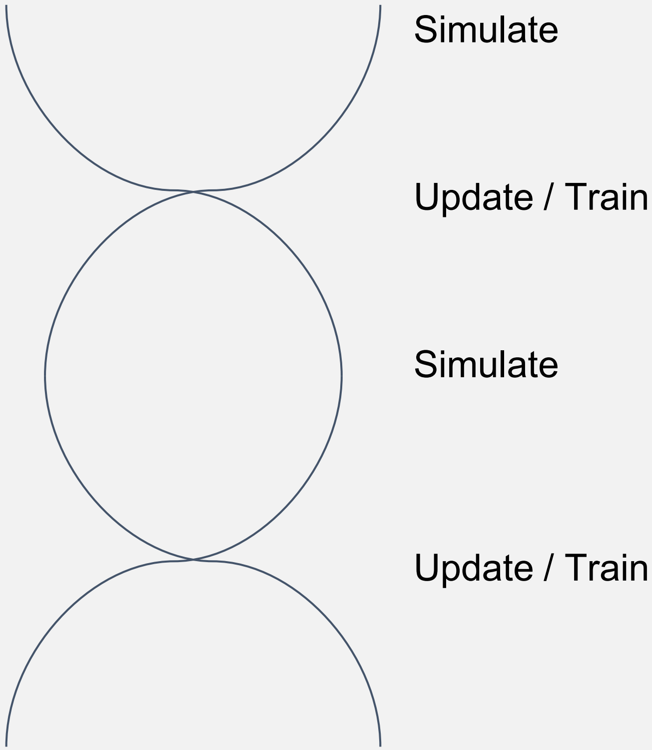
How we play with Neural Networks

- NN Model is used to calculate the reward for all resultant states from a current board



Training Optimization

- We have unlimited data, but we have to earn it.
- We can parallelize the simulation, but have to unify the Q_Table update or Neural Network Training
    - multiprocessing vs multithreading
- Methods to control state space
- What happens when both agents are learning at the same rate?
- How do we choose a reward function?



Conclusion

Conclusion

- Creating a game, using agents to create random games to use as data, and building Q-Tables is an effective way to use reinforcement learning to consistently beat a random agent

- Using neural networks to enact Deep Q-Learning makes this process much faster and less brittle, but requires a lot of data to do successfully