Derrick Lor

4/16/2024

Artificial Intelligence

Project Starter Gomoku AI

The environment for this project will be set up and implemented in python. Python is easy to code, flexible, and runs sufficiently fast enough to compute actions in real time. Inspiration was taken from OpenAI’s gymnasium library for basic structure and readability. Experienced users of the library will find my implementation of Gomoku MDP for reinforcement learning to be intuitive. Other dependencies include basic math library like numpy and a game engine like pygame to run. Strictly, pygame is used to provide human interaction and feedback to the game, but to facilitate training the rendering mode can be changed to off.

With pygame enabled, the user can interact with the environment to play a game of Gomoku against my AI. Using object-oriented programming, the AI looks at the position and states of the board to find the next best available action. The game window is a 900x900 pixel sized window divided into 9x9 playable grid world. There are only three self-created assets used in the game: white piece, black piece, and board background. Nothing fancy nor over the top. Visualization is simple and easy to read. At the back end, the board state and game rules are established in a board object. The agent or player can then interact with the board through a series of actions such as place piece. After each move, the board state is returned to the agent or player, the reward, and if the game has terminated. Very similar to the environment object in gymnasium.

Implementation of the model or MDP follows the Gomoku rules. It is a turn-based game where the agent is playing against the AI. The first to reach five in a row wins. The agent or player is first to start the game, followed by the AI. Rewards of each action are determined by an evaluation function. The more pieces that line up, the higher the reward. But also, the more pieces that get cutoff or the more pieces the opponent lines up reduce the agent reward. Rewards can be negative or stay the same. Negative if the action puts the agent in a worse state. A worse state can be defined if the action blocks possible future win conditions or allows opponent possible future win conditions. Rewards can stay the same if the action counteracts the action of the opponent. The goal is to find the action that gives the maximum reward, since reaching closer to the win condition will yield more rewards.

The AI implementation uses a minimax algorithm to find all possible sub states of the board position, then evaluates the position using the evaluation function defined in the earlier paragraph. The depth of recursion is currently set at depth 1. Calculating depth deeper than 1 is expensive and will take upwards of O(n2d). Where n2 is the action space and d is the depth of recursive iteration through all available actions. In testing, depth of 1 provides a competitive AI bot that provides quick runtime and more than 50%-win rate against me, but higher depths will exponentially increase the runtime, slowing the program dramatically.

The reinforcement learning implementation will use a linear function approximation to generate the next action based on the board’s state. Using a finite state space is not possible since the agent must reach each possible state space and then branch out in all possible action spaces. Then must continuously generate new episodes to fill in a Q table representing all possible states, and best actions in those states. A simpler approach is to use a function, input the state, and generate an action following gradient ascent to maximize rewards. Then iteratively change weights within the function approximation features.