알파제로로 해결하는 틱택토 (Tic-Tac-Toe)

세칸 승부사 팀

김현서, 안서연, 이승연, 이채연, 이지민, 장예원

GITHUB : https://github.com/KanghwaSisters/24-2-TicTacToe.git

[초록]

본 연구는 CNC 모델을 제어하여 AI와 인간이 현실에서 상호작용할 수 있는 시스템을 구축하는 것을 목표로 한다. 이를 위해 알파제로(AlphaZero)를 포함한 다양한 방법론(MCS, MCTS, Min-Max, AlphaBeta)을 구현하여 틱택토를 플레이하도록 설계하였다. 특히, 알파제로 기반의 강화학습 모델을 활용하여 최적의 틱택토 플레이어 신경망을 학습 시켰으며, 이를 통해 기존 방법론과의 성능을 비교·분석하였다.

연구 과정에서는 게임의 흐름을 이미지 상태 인식, 강화학습 모델의 예측, 로봇 행동 수행, 게임 상태 업데이트, 턴 반복의 5단계로 정의하여 AI가 현실에서 게임을 수행할 수 있도록 하였다. 실험 결과, 알파제로 기반의 AI가 다른 방법론보다 높은 승률을 기록하며 가장 우수한 성능을 보였다. 본 연구는 알파제로의 강화학습 기법이 틱택토 환경에서 효과적으로 적용될 수 있음을 입증하며, 향후 다양한 게임 및 로봇 제어 분야로의 확장 가능성을 제시한다.

핵심 주제어: 알파제로, 강화학습, 틱택토, AI vs 인간 상호작용, CNC 모델, 게임 AI

목차

그림 • 표 목차

Ⅰ. 서론

틱택토(Tic-Tac-Toe)는 단순한 규칙을 가진 보드 게임이지만, AI 알고리즘을 평가하는 데 유용한 실험 환경을 제공한다. 특히, AI가 게임 전략을 학습하고 최적의 수를 찾는 과정에서 다양한 탐색 기법과 강화학습 모델을 적용할 수 있다. 본 연구는 이러한 틱택토 환경에서 AI와 인간이 현실에서 상호작용할 수 있도록 CNC 모델을 제어하는 시스템을 구축하는 것을 목표로 한다.

이를 위해 다양한 AI 기법을 적용하여 틱택토를 플레이하는 시스템을 설계하였다. 기존의 탐색 알고리즘 Min-Max, AlphaBeta, Monte Carlo Search (MCS), Monte Carlo Tree Search (MCTS) 등의 방법론을 적용하고, 알파제로(AlphaZero) 기반의 강화학습 모델을 활용하여 최적의 플레이어 신경망을 학습시켰다. 이를 통해 각 방법론 간의 성능을 비교하고, 강화학습 기법이 게임 AI 개발에서 어떤 이점을 가지는지 분석하고자 한다. 본 연구에서는 기존의 탐색 알고리즘 중 MCTS와 알파제로의 성능 비교를 중점적으로 다루었다.

또한, 본 연구는 AI가 단순한 게임을 넘어서 현실의 물리적 시스템과 결합하여 동작할 수 있도록 하는 데에도 중점을 두었다. 틱택토 게임의 진행을 이미지 상태 인식 → 강화학습 모델의 예측 → 로봇 행동 수행 → 게임 상태 업데이트 → 턴 반복의 5단계로 정의하여, AI가 물리적 환경에서도 효과적으로 게임을 수행할 수 있도록 설계하였다.

본 논문에서는 기존 탐색 알고리즘과 강화학습을 비교하고, AI 모델의 학습 및 평가 과정을 상세히 설명한다. 이를 통해 알파제로 기반 AI가 틱택토 환경에서 최적의 성능을 달성할 수 있음을 검증하고, 향후 게임 AI 및 로봇 제어 분야로의 확장 가능성을 제시하고자 한다.

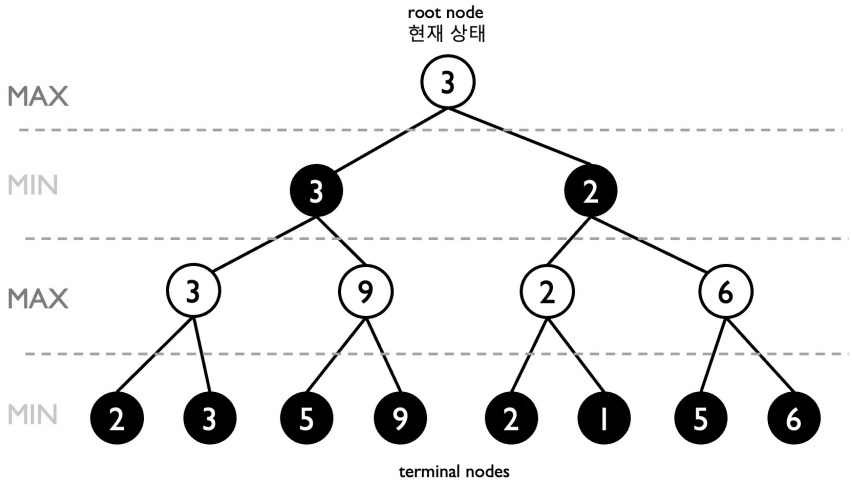
Ⅱ. 핵심 방법론

A. 알파제로 이전 방법론

1. Min-Max

최대 최소 전략을 사용하여 턴제 게임등과 같은 프로그램에서 의사결정에 주로 사용되는 알고리즘으로 two-player 게임에서 주로 사용된다. 이때 최대 최소 전략은 어떤 계획이 성공했을 때의 효과를 고려하는 것보다 실패했을 때의 손실을 고려해서, 손실이 최소가 되도록 계획을 세우는 전략을 의미한다. 구체적인 탐색 과정은 다음과 같다.

[그림 1] Min-Max 탐색 과정



Min-Max 알고리즘의 트리 탐색 과정은 4가지 단계로 이루어진다.

1) 현재 상태를 기준으로 한 층씩 내려가며 다음 수를 임의로 예상한다.
2) 상태함수를 통해 각 선택에 따른 최종 결과값을 수치화한다.
3) 수치화한 값들을 가지고, 한 층씩 올라가며 Min, Max를 반복한다.
    1. Min: 값을 비교하여 가장 작은 것을 선택
    2. Max: 값을 비교해 가장 큰 값을 선택

4) 마지막 최종 선택은 Max값을 계산한다.

이때 Maximizing player의 목적은 자신의 score를 최대화하는 것, Minimizing player의 목적은 Max player의 score를 최소화하는 것이다. 즉, Max 플레이어를 기준으로 좋고 나쁜 것을 판단하기 때문에, Min 플레이어가 자신의 score를 최대화하는 선택을 하는 것을 Max플레이어의 score를 최소화하는 선택을 한다고 표현한다.

Min-Max 알고리즘의 단점은 트리의 모든 노드를 탐색해야하기 때문에 트리의

깊이가 많아질수록 계산시간이 늘어난다는 것이다. 또한, 에이전트는 행동을 하고 다음 상태로 갈 때마다 모든 경우의 수에 대한 게임 트리를 계산하는 것을 반복하면서 최적의 행동을 하기 때문에 상태와 행동의 개수가 늘어나면 계산량이 기하급수적으로 늘어나게 된다.

2. AlphaBeta

알파베타법(Alpha-Beta algorithm)은 미니맥스법을 최적화한 게임 트리 탐색 방법이다. 미니맥스법은 게임 트리에서 최선의 수를 찾기 위해 완전 탐색을 수행하지만, 경우의 수가 기하급수적으로 증가하기 때문에 비효율적이다. 이를 개선하기 위해 알파베타 가지치기를 적용하면 불필요한 노드 탐색을 생략하여 속도를 크게 향상시킬 수 있다.

알파베타 탐색에서는 두 개의 값을 사용한다.

-알파(α): 최상의 최소값 (현재까지 발견된 최선의 Maximizer(최대화 측) 노드 값)

-베타(β): 최상의 최대값 (현재까지 발견된 최선의 Minimizer(최소화 측) 노드 값)

탐색 과정은 다음과 같다.

1) 깊이 우선 탐색(DFS) 수행

   루트 노드에서 시작하여 리프 노드까지 탐색을 진행한다.

2) Max 레벨

   가능한 수 중 최대값을 찾으며 진행한다. 현재까지 찾은 최고 점수를 알파(α) 값으로 저장한다. 더 좋은 선택지가 있으면 알파(α) 값을 갱신한다.

3) Mix 레벨

   가능한 수 중 최솟값을 찾으며 진행한다. 현재까지 찾은 최저 점수를 베타(β) 값으로 저장한다. 더 나쁜 선택지가 있으면 베타(β) 값을 갱신한다.

4) 가지치기

   불필요한 탐색을 생략하는 과정이다.

   - 알파컷(α-cut) : Max 레벨에서 수행되는 가지치기이다. 현재 노드의 자식들을 순차적으로 탐색하며 각 자식 노드의 값이 이전에 찾은 베타(β) 값보다 크면 가지치기한다. 이는 Min 레벨에서 절대 이 경로가 선택되지 않을 것이기 때문이다.
   - 베타컷(β-cut) : Min 레벨에서 수행되는 가지치기이다. 현재 노드의 자식들을 순차적으로 탐색하며 각 자식 노드의 값이 이전에

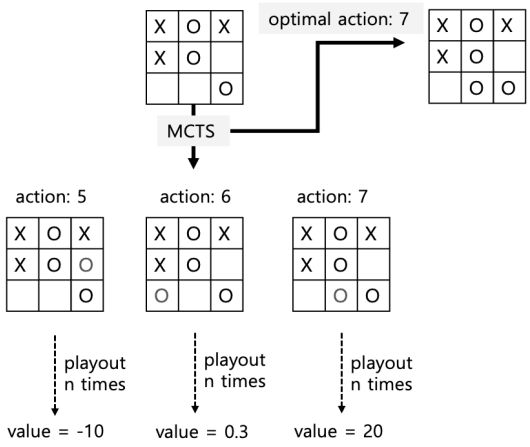
찾은 알파(α) 값보다 작으면 가지치기 한다. 이는 Max 레벨에서 절대 이 경로가 선택되지 않을 것이기 때문이다.

알파베타법은 불필요한 노드를 탐색하지 않도록 가지치기를 함으로써, 미니맥스법보다 훨씬 더 적은 수의 노드를 탐색할 수 있다. 이로 인해 게임 트리 탐색 속도가 빨라지고, 더 깊은 깊이까지 탐색할 수 있게 된다. 그러나 알파베타 가지치기의 성능은 탐색 순서에 크게 의존한다. 최적의 순서로 노드를 탐색할 경우 가지치기가 많이 발생해 탐색 효율이 크게 향상되지만, 잘못된 순서로 노드를 탐색하면 가지치기의 효과가 적어져 성능 향상이 크게 제한될 수 있다. 따라서 탐색 순서를 최적화하는 것이 매우 중요한 요소이다.

3. MCS (Monte Carlo Search)

몬테카를로 방법(Monte Carlo method)는 반복된 무작위 추출을 이용해 함수의 값을 수리적으로 근사하는 알고리즘이다. 컴퓨팅에서 몬테카를로 알고리즘(Monte Carlo algorithm)은 항상 유한한 시간에 멈추지만, 낮은 확률로 부정확할 수도 있는 결과를 도출하는 알고리즘이다. 본 프로젝트에서는 MDP(Markov Decision Process)문제를 해결하기 위해 몬테카를로 알고리즘을 활용한 시뮬레이션을 몬테카를로 시뮬레이션(Monte Carlo Simulation, 이하 MCS)이라고 정의하였다. MCS는 현재 상태에서 가능한 모든 행동에 대해 각각 일정 횟수만큼 무작위 게임을 진행해보고(playout), 각 게임의 결과값(value)을 누적해 가장 큰 값을 갖는 행동을 선택한다. 각 게임의 결과값(value)은 승리한 경우 +1, 무승부인 경우 0, 패배한 경우 −1로 한다.

[그림 2] MCS 탐색 과정



MCS는 고전적인 몬테카를로 알고리즘을 그대로 구현해 무작위 시뮬레이션만으로 MDP 문제를 해결한다는 의의가 있다. 따라서 모든 게임에 대해 아무런 부가정보 없이 문제를 해결할 수 있다는 것이 장점이다. 하지만 한 번 행동을 선택할 때마다 (playout 횟수)×(가능한 행동의 개수) 만큼 무작위 시뮬레이션을 진행해야 하므로 탐색 시간이 매우 길고, 게임의 크기가 커질 수록 연산량이 기하급수적으로 늘어난다는 단점이 있다.

4. MCTS (Monte Carlo Tree Search)

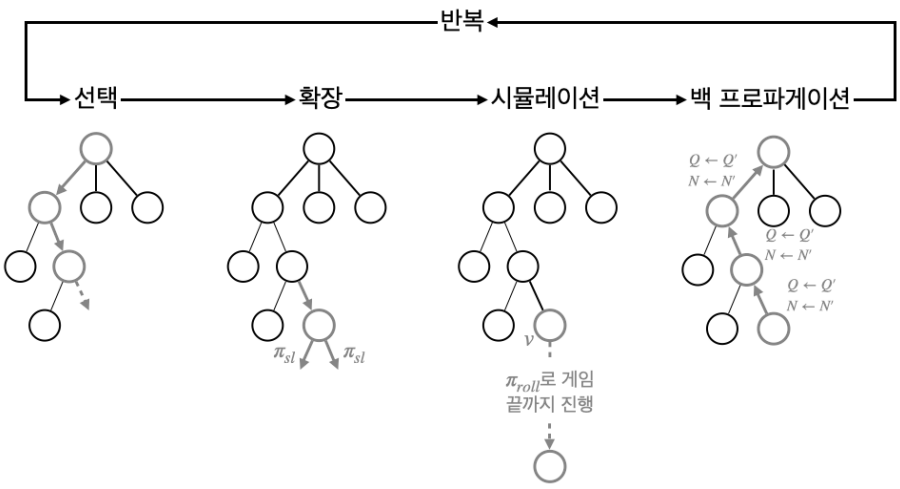
몬테카를로 트리 탐색 (Monte Carlo Tree Search, 이하 MCTS)는 MDP (Markov Decision Process)를 해결하는 방법의 한 종류이다. MCTS는 트리 서치 (tree search)에 몬테카를로 (Monte Carlo) 알고리즘을 응용한 것으로, 어떤 상태에서 게임이 종료될 때까지 모든 경우의 수를 탐색하지 않고, 몬테카를로 기반 시뮬레이션을 통해 랜덤한 수를 두어가면서 게임을 한번 끝까지 진행해본다. MCTS의 목적은 가장 최선의 움직임을 분석하는 데에 있으며, 무작위로 이동하면서 샘플을 얻은 것으로 트리를 확장한다.

MCTS에서 많이 사용되는 용어들은 다음과 같다.

- 루트 노드 (root node): 트리가 시작되는 맨 위의 노드.
- 리프 노드 (leaf node): 트리에서 가장 밑에 있는 노드.
- 플레이아웃 (playout): MCTS가 임의로 한번 끝까지 진행해보는 게임을 플레이아웃이라고 한다. 플레이아웃은 한 리프 노드가 만들어질 때, 즉 한 수가 둬졌을 때 실행된다. 플레이아웃의 목적은 이 수가 얼마나 좋은 수인지 확인하는 데에 있으며, 승패가 가려져서 플레이아웃이 끝나는 마지막 상태를 terminal state이라고 지칭한다. 이때 플레이아웃은 빠르게 수행되어야 하므로, 시간이 오래 걸리지 않는 정책이 사용된다. 이 정책을 MCTS의 rollout policy 또는 default policy이라 지칭한다.
- UCT (Upper Confidence Boundary of Tree): 선택 단계에서 확장을 이어나가기 위한 child node를 정할 때 사용되는 정책.

MCTS의 구조 및 연산 과정은 다음과 같다.

[그림 3] MCTS의 구조 및 연산 과정



1) 선택 (Selection)

게임 트리를 사용하는 루트 노드로부터 아래로 이동하기 위해 자식 노드를 순차적으로 선택하는 것을 말한다. 자식 노드를 선택할 때는 현재까지의 승률이 큰 것과 지금까지 방문횟수가 작은 것을 선호하도록 선택 우선순위를 정한다.

2) 확장 (Expansion)

선택 과정을 통해 마지막에 도달한 노드에 새로운 노드를 추가할 수 있다. 확장 단계로 진입하게 한 수가 특정 조건을 만족하면, 해당 수에 해당하는 노드를 추가한다.

3) 시뮬레이션 (Simulation)

노드의 형세 평가를 위해 승패가 결정될 때까지 무작위로 게임을 해보는 단계이다. 이러한 시뮬레이션은 시간이 많이 걸리기 때문에, 게임 규칙에 맞는 수를 무작위로 선택해서 하거나 약간 똑똑하지만 빠르게 계산되는 방법으로 게임을 한다. 시뮬레이션을 시작한 노드에는 가능한 첫 수의 시도 횟수 및 승률을 기록을 해두어서 나중에 선택 단계와 확장 단계에서 이 정보를 이용할 수 있도록 한다.

4) 역전파 (Backpropagation)

시뮬레이션 게임의 승패 정보를 현재 노드에서 루트 노드까지의 경로 상에 있는 노드들의 '이긴 횟수/전체 횟수' 정보에 반영한다. 시뮬레이션 게임의 승패 정보를 현재 노드에서 루트 노드까지의 경로 상에 있는 노드들의 시뮬레이션 단계에서 지면, 역전파 단계에서는 단말 노드부터 루트 노드까지 정보 상의 모든 노드의 이긴 횟수는 그대로 두고 전체 횟수 값만 1씩 증가시킨다.

B. 알파제로란?

알파제로는 딥마인드 (DeepMind)에서 개발한 범용 강화학습 기반 게임 AI로, 기존의 알파고 제로 (AlphaGo Zero)를 확장하여 특정 도메인에 구애받지 않는 알고리즘으로 발전된 모델이다. 알파고 제로와 동일한 자가 대국 (Self-Play) 및 몬테카를로 트리 탐색 (MCTS)을 기반으로 학습하지만, 체스, 쇼기, 바둑 등 다양한 보드게임에 적용할 수 있도록 일반화된 것이 가장 큰 특징이다.

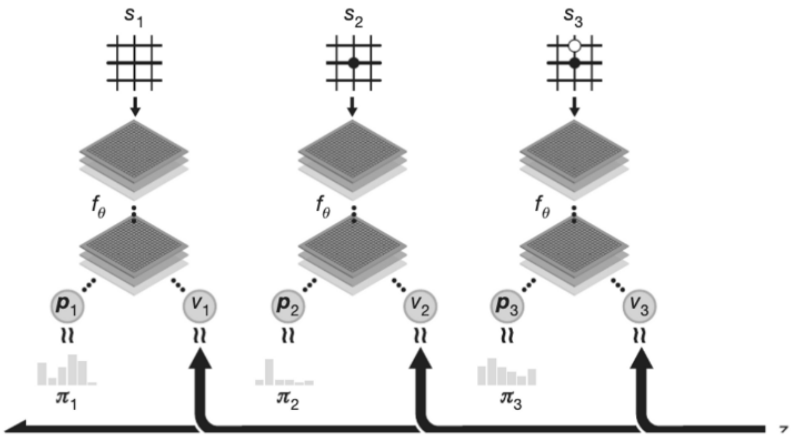
알파제로는 알파고 (AlphaGo)와 비교했을 때, 사전 전문가 지식 없이 순수한 자기 대국만으로 학습을 진행한다는 점에서 큰 차이를 보인다. 전문가 데이터 (Supervised Learning)가 존재하지 않기 때문에 Rollout Policy 및 SL Policy가 제거되었으며, 게임 도메인에 특화된 Handcrafted Feature 없이, 단순히 보드 상태만을 입력으로 사용하여 학습을 진행한다. 또한, 하나의 신경망이 강화학습 정책 (policy)과 가치 (value) 평가를 모두 수행하며, 학습 과정에서도 MCTS를 적극적으로 활용하여 탐색 효율성을 극대화한다.

이러한 방식 덕분에 알파제로는 특정 게임에 최적화된 별도의 모델 설계 없이도, 동일한 학습 구조를 활용하여 다양한 게임에서 초인적인 성능을 발휘할 수 있는 범용 AI로 자리 잡았다.

알파제로는 자가 대국과 MCTS를 결합하여 최적의 정책과 가치 함수를 학습하는 방식으로 동작한다. 학습 과정은 크게 (1) 자가 대국을을 통한 데이터 생성, (2) 신경망 학습, (3) 모델 업데이트의 반복적인 과정으로 이루어진다.

먼저, 자가 대국을 진행하며 각 상태 s_t에서 MCTS를 활용해 최적의 액션을 선택하고, 이를 통해 정책 분포 π_t를 얻는다. 게임이 종료되면 최종 승패 결과를 반영하여 보상 값 z_t를 결정한다. 승리 시 z = 1, 패배 시 z = −1이 부여되며, 이 과정을 통해 (s_t, π_t, z_t)형태의 학습 데이터가 축적된다.

[그림 4] 뉴럴 네트워크 트레이닝 (Neural Network Training)



이후, 자가 대국을 통해 수집된 데이터를 기반으로 신경망 $f\theta(s)=(p,v)$를 학습한다. 학습 과정에서는 이전에 저장된 (s_t, π_t, z_t) 데이터를 랜덤 샘플링(random sampling) 하여 손실 함수(loss function)를 최소화하는 방향으로 최적화가 진행된다. 알파제로의 손실 함수는 다음과 같다.

$$l = (z-v)^2 - \pi^T \log p + c \| \theta \|^2$$

- $(z-v)^2$: 신경망이 예측한 가치 v가 실제 게임 결과 z와 일치하도록 유도하는 항.
- $-\pi^T \log p$: MCTS를 통해 얻은 정책 π와 신경망이 예측한 정책 p가 일치하도록 학습하는 항.
- $c \| \theta \|^2$: 모델 가중치 θ가 과도하게 커지는 것을 방지하는 정규화(regularization) 항.

이러한 과정을 반복하며, 알파제로는 지속적으로 스스로 학습하여 점점 더 최적화된 정책과 가치 함수를 학습하게 된다. 이를 통해 알파제로는 사전 데이터 없이도 오직 자가 대국만으로 최적의 전략을 탐색하고 강화할 수 있는 능력을 갖추게 된다.

C. 탐험 (Explore)

강화학습에서는 에이전트가 최적의 행동을 학습하기 위해 탐색 (Exploit)과 탐험 (Explore) 사이의 균형을 조절하는 것이 핵심 과제가 된다. 일반적인 지도학습 (supervised learning)과 달리, 강화학습에서는 정답 레이블이 주어지지 않으며, 환경과의 상호작용을 통해 보상을 최대화하는 정책을 학습해야 한다. 이를 위해 가능한 많은 상태 (state)에 노출되어야 하며, 이를 위해 탐험이 필수적이다. 즉, 다양한 상태와 행동을 시도하며 환경에서 얻을 수 있는 보상을 최대한 탐색하는 과정이 필요하다.

탐험 기법 중 하나로는 볼츠만 탐색 (Boltzmann Exploration)이 사용되는데, 이는 볼츠만 분포 (Boltzmann Distribution)를 기반으로 행동을 선택하는 방식이다. 볼츠만 분포는 소프트맥스 (Softmax) 함수와 유사하지만, 추가적으로 온도 (Temperature, τ) 파라미터를 포함하며, 수식은 다음과 같다.

$$\pi(a) = \frac{e^{Q(a)/\tau}}{\sum_{j=1}^{N} e^{Q(j)/\tau}}$$

- Q(a): 행동 a의 기대 보상 값 (Q-value)
- τ: 온도 (Temperature) 파라미터로, 탐험과 탐색의 강도를 조절하는 항

온도의 개념은 맥스웰-볼츠만 분포 (Maxwell-Boltzmann Distribution)에서 유래하였으며, 물리학에서는 분자의 운동 속도와 온도의 관계를 설명하는 데 사용된다. 이를 강화학습에 적용하면,

1) 초기에는 높은 온도($\tau\uparrow$) → 무작위 탐험 증가
2) 학습이 진행될수록 온도를 낮춤($\tau\downarrow$) → 이미 학습한 방향으로 탐색 강화

즉, 초기에는 다양한 상태와 행동을 시도하며 최적의 정책을 찾기 위해 광범위한 탐험을 수행하고, 학습이 진행되면서 점차 학습된 지식에 기반하여 더 높은 확률로 좋은 행동을 선택하는 방식으로 변화해간다. 이를 통해 에이전트는 탐험을 통해 새로운 최적 정책을 발견하고, 궁극적으로는 안정적인 학습을 수행할 수 있다.

D. 핵심 구현부 및 하이퍼 파라미터

1. 핵심 구현부

1) State

강화학습에서 상태 부분을 각 상태에 대한 정보를 담고 있는 State 클래스로 구현하였다. State 클래스의 객체 하나는 한 상태를 나타내고, 게임이 한 턴 진행되면 다음 플레이어 기준의 다음 State 객체를 생성한다. 한 State 객체가 가지고 있는 attribute는 게임 보드 크기, 자신의 수와 상대의 수이다. 이 State 객체의 상태에 대한 정보를 제공하는 내장 메서드로 현재 상태의 lose, draw, done 여부를 계산하는 메서드와, 현재 상태에서 가능한 행동의 list를 반환하는 메서드가 있다. 게임 진행을 위해 현재 상태에서 입력받은 행동을 했을 때의 다음 State 객체를 반환하는 next 메서드가 있다. 그리고 신경망에 입력할 수 있는 형태로 변환하는 메서드와, 보드를 시각화하는 부가적인 메서드를 구현하였다.

학습 과정에서 이 클래스 객체를 여러 번 반복하여 호출하게 되고 내부 함수 또한 자주 사용되기 때문에, 코드 실행 속도를 고려해 구현하고자 하였다. 리스트 대신 최대한 Numpy 배열을 사용하고, Numpy 내장 함수를 활용해 코드를 최적화하였다. 전체 코드는 부록 ??에 첨부된 Github에 업로드 되어있다.

2) Net

알파제로에서는 정책 네트워크와 가치 네트워크를 통합한 듀얼 네트워크 (Dual Network)를 사용한다. 이때, 듀얼 네트워크를 구현하기 위해서는 잔차 학습 (Residual block)을 활용한 모델인 ResNet의 개념이 활용된다.

- ResidualBlock(nn.Module)

  잔차 연결을 통해 학습을 돕는 잔차 학습 클래스이다. 코드 진행 과정은 다음과 같다. 우선, 원래의 입력값을 잔차 연결용으로 보존하기 위한 변수인 shortcut에 입력값을 저장한다. 이로 인하여 실제 입력값 x를 자유자재로 변형하는 것이 가능해진다. 다음으로, 컨볼루션 및 배치 정규화, 그리고 ReLu 활성화 함수를 거쳐 나온 x는 이 과정을 다시 한 번 거치게 되어 출력이 된다. 마지막으로, 변형된 입력값 x에 원래 입력값 shortcut을 더한 후, 이에 ReLu 활성화 함수를 적용하면 최종 출력값이 된다.

- DualNetwork(nn.Module)

  알파제로에서 사용되는 정책과 가치 예측을 위한 듀얼 네트워크 구조를 구현한 클래스이다. 앞선 잔차 학습을 연속적으로 쌓은 레이어를 기반으로, 네트워크의 깊이를 증가시켜 복잡한 특징을 학습한다. 이때, 정책 네트워크와 가치 네트워크의 예측이 함께 이루어진다.

3) MCTS

MCTS는 알파제로의 핵심 요소로, 신경망이 예측한 정책과 가치를 기반으로 트리 구조를 탐색하고 게임에서 최적의 행동을 확률적으로 선택하는 알고리즘이다. UCB1 공식을 통해 탐험과 탐욕적 행동 간의 균형을 맞추며, 여러 시뮬레이션을 통해 최적의 정책을 도출한다.

MCTSNode 클래스에는 MCTS에서 사용되는 각 노드를 구현하였다. 각 노드는 하나의 게임 상태를 나타내며, 해당 상태에서 가능한 행동에 대한 정보를 담고 있다. 클래스 내부에는 현재 노드를 평가하는 메서드, UCB1 공식을 사용하여 자식 노드를 선택하는 메서드가 있다.

구현한 MCTS 알고리즘의 주요 흐름은 다음과 같다.

- 트리 탐색: 현재 상태에서 가능한 모든 수들을 자식 노드로 확장하고, UCB1 공식을 사용해 탐험과 탐욕적 행동 간의 균형을 맞추며 노드를 탐색한다.
- 정책 확률 분포 계산: 여러 번의 시뮬레이션을 통해 각 수에 대한 확률 분포를 계산한다. 이 확률 분포는 틱택토 게임에서 어떤 수를 둘지 선택하는 데 사용된다.
- 행동 선택: 각 상태에 대해 확률적으로 수을 선택하여, 틱택토 플레이에서 둘 수를 결정한다.

4) Self-play

self-play는 스스로 플레이하며 데이터를 생성하는 역할을 한다. MCTS를 활용해 스스로와의 대국을 진행하고, 각 상태에서 정책과 보상을 저장하여 상태, 정책, 가치 데이터를 도출해낸다.

self_play_game 메서드에 사용되는 매개변수 중 simulation은 한 턴에서 AI가 최적의 수를 찾기 위해 실행하는 MCTS 탐색 횟수를 의미한다. temperature는 탐색 분포를 조절하는 파라미터로 AI가 선택할 행동의 확률 분포를 조정하는 값이다. 만약 이 값이 0이라면 가장 확률이 높은 행동을 선택하게 되고 0보다 크다면 다양한 행동을 선택할 가능성이 높아진다.

self play의 주요 흐름은 다음과 같다.

- 게임 초기화: 새로운 게임 상태를 만들고 데이터를 저장할 리스트를 준비한다.
- MCTS 기반 확률 계산: 현재 상태에서 MCTS 점수를 얻고, 이를 정책 벡터로 변환하여 저장한다.
- 행동 선택 및 진행: MCTS 확률에 따라 랜덤하게 한 수를 선택하고, 다음 상태로 이동한다.
- 게임 종료 후 보상 부여: 게임이 끝난 후 승패에 따라 보상(1 혹은 -1)을 정하고 상태, 정책, 가치 데이터를 반환한다.

5) Train

알파제로의 신경망 모델을 학습하기 위해 train_model 메서드를 구현하였다. 해당 메서드에서는 self-play로 얻은 데이터를 활용해 정책 및 가치함수를 학습한다.

정책 손실함수는 CrossEntropyLoss를 사용해 MCTS 정책 분포와 예측값의 차이를 최소화, 가치함수는 MSELoss를 사용해 실제 보상과 예측된 가치 평가 차이를 최소화하고자 했다. 최종 loss를 구하는 과정에서 정책 loss와 가치 loss를 계산해봤을 때 정책이 상대적으로 크게 나온다는 점을 반영하여 정책 loss에 0.5를 곱해주었다. 데이터는 self-play에서 얻은 상태, 정책, 가치 3가지의 데이터를 tensor로 변환하여 모델 학습에 사용한다. 또한, 효율적인 학습을 위해 미니배치로 학습한다.

6) Evaluation

알파제로의 신경망 모델을 평가하기 위하여 evaluate_models()라는 메서드를 구현하였다. 해당 메서드는 새로 학습된 모델(new_model)과 기존에 학습된 모델(old_model)을 대결시켜 성능을 평가한다. 두 모델은 각

차례에서 pv_mcts_scores() 메서드를 이용해 MCTS을 수행하여 다음 수를 결정한다.

두 모델이 같은 환경에서 동일한 조건으로 대결하므로, 일반적인 평가 지표(정확도, 손실 함수 등)보다 실제 틱택토 게임에서 보이는 성능을 직접 비교할 수 있다. 또한 매 게임마다 new_model과 old_model이 번갈아 가며 첫 번째 플레이어가 되도록 설정하여, 특정 모델이 선공으로만 두는 불균형을 방지하고 공정한 성능 평가가 이루어지도록 한다. 일정 횟수만큼 게임을 진행하며, 두 모델이 이긴 횟수를 각각 new_wins와 old_wins에 기록한다. 두 변수를 비교하여 새로운 모델을 선택할지, 기존 모델을 유지할지 결정한다.

또한 사람와 AI가 대국하는 play_against_ai() 메서드를 구현하여 직접 대국을 하며 주관적인 평가에 활용하였다. 선공을 누가할지 먼저 선택하고 게임을 진행한다. 이를 통해 AI가 특정 상황에서 어떤 수를 두는지 직접 관찰할 수 있으며, AI가 예상 외의 이상한 수를 두거나, 약한 패턴이 있는지 확인하여 이를 모델 학습 개선에 활용할 수 있었다.

2. 하이퍼 파라미터

본 연구는 서로 다른 수치의 하이퍼 파라미터를 기저로 둔 알파제로 모델을 총 4개로 구현해보았다. 이 중 1개의 모델은 단순 CNN 신경망을 사용했고, 나머지 3개의 모델은 ResNet 신경망을 사용했다. 4개의 모델 중 마지막 모델의 성능이 가장 우수한 것으로 결론이 났으며, 그 이유를 알아가보고자 한다.

다음은 단순 CNN 신경망을 차용한 1번 모델 설계 시 적용된 하이퍼 파라미터이다.

[표 1] 1번 모델 (단순 CNN 신경망)

# 데이터 증강	LEARN_RATE = 0.0005
DATA_AGUMENTATION = True	
	L2 = 1e-4
REWARD_WIN = 1	GAMMA = 0.1
REWARD_LOSE = -1	TOTAL_SP_NUM = 300
REWARD_DRAW = 0	SP_NUM_TRAIN = 1 # 셀프 플레이를 수행할 게임 수(오리지널: 25,000)
# state	EXPLORE_REGULATION = 2
NUM_HISTORY = 0	
PLAYER_INFO = True	BATCHSIZE = 32
	TRAIN_EPOCHS = 5
CONV_UNITS = 64	MEM_SIZE = 50000
# mcts	# evaluation

C_PUCT = 5	EVAL_NUM_GAME = 20
EVAL_CNT = 300	TEST_NUM_GAME = 10
TEMPERATURE = 1.0	CRITERIA = 0.55
TEMPERATURE_DECAY = 0.8	
	EVAL_FREQUENCY = 20

1번 모델은 단순 CNN 신경망을 기반으로 구축되었으며, 64개의 합성곱 필터 (CONV_UNITS = 64)를 사용하여 보드 상태를 학습하도록 설계되었다. 해당 모델은 MCTS 탐색 깊이 (C_PUCT=5 및 EVAL_CNT=300)를 적절하게 선정하여 알파제로가 충분한 탐색을 수행할 수 있도록 설정되었으나, 비교적 작은 CNN 구조로 인해 복잡한 전략을 학습하는 데 한계가 있었을 가능성이 있다. 즉, ResNet을 적용한 모델들과 비교했을 때 상대적으로 단순한 구조로 인해 복잡한 전략을 충분히 학습하지 못한 점이 상대적인 성능 저하의 주요 원인으로 보인다. 첨언하자면, 해당 모델은 에포크가 수치를 늘리더라도 최신 모델이 이전의 best 모델을 뛰어넘는 성능을 보이지 못했다고 한다.

다음은 ResNet 신경망을 차용한 2번 모델 설계 시 적용된 하이퍼 파라미터이다.

[표 2] 2번 모델 (ResNet 신경망)

〈network〉	〈MCTS〉
DN_FILTERS = 128 # convolutional layer 수	PV_EVALUATE_COUNT = 20 # predict 1회당 시뮬레이션 횟수 50 -> 20
DN_RESIDUAL_NUM = 16 # residual block 수	
DN_INPUT_SHAPE = (3, 3, 2) # 3x3의 2차원 배열 2개	〈self-play〉
DN_OUTPUT_SIZE = 9 # 행동 수	SP_GAME_COUNT = 100 # self-play 수행할 게임 수
	SP_TEMPERATURE = 1.0
〈train〉	
RN_EPOCHS = 100 # 학습 횟수	〈Evaluate Network〉
	EVAL_GAME_COUNT = 10
	EVAL_TEMPERATURE = 1.0

2번 모델은 ResNet 구조를 적용하여 기존 CNN 모델보다 더 깊은 신경망을 구성했다. 특히, 128개의 필터 (DN_FILTERS=128)를 사용하여 더 많이 학습할 수 있도록 하였으며, MCTS 탐색 횟수 (PV_EVALUATE_COUNT=20)가 상대적으로 적어 계산량을 줄이는 대신 빠른 의사결정을 내릴 수 있도록 설계되었다. 또한 Self-play를 위한 SP_GAME_COUNT을 100으로 설정하여 충분한 학습 데이터를 확보하고자 했다. 이 모델은 CNN 기반 모델보다 더 깊은 네트워크로 인해 더 정교한 패턴 학습이 가능했으나, MCTS 탐색 깊이가 부족하여 후공 시 최적의 전략을 찾는 데 한계가 있었을 가능성이 있다.

다음은 ResNet 신경망을 차용한 3번 모델 설계 시 적용된 하이퍼 파라미터이다.

[표 3] 3번 모델 (ResNet 신경망)

# 신경망 구조 관련 N_KERNEL = 64 # 합성곱 레이어(ConvLayer)의 필터 수 / 클수록 더 많은 패턴을 학습 N_RESIDUAL_BLOCK = 16 # Residual Block 개수 / 클수록 더 깊은 네트워크를 만들 수 있음 # 학습 관련 BATCH_SIZE = 128 # 한 번의 학습에서 처리하는 데이터 샘플 개수 EPOCHS = 50 # 전체 학습 반복 횟수 -> 늘려보자! LEARNING_RATE = 0.001 # 학습률 (뉴런 가중치 업데이트 속도) LEARN_EPOCH = 100 # 학습률 감소 주기	# Self-play 관련 -> 한번 self-play 돌린 후 학습 SP_GAME_COUNT = 20 # self-play에서 생성할 게임 수 SP_TEMPERATURE = 1.5 # self-play에서 탐색 다양성을 조절하는 온도 파라미터 # 평가 관련 EVAL_COUNT = 100 # 평가할 때 MCTS 탐색을 몇 번 수행할지 결정 PV_EVALUATE_COUNT = 800 # MCTS가 얼마나 깊이 탐색할지를 결정 -> 400으로 수정 EVAL_GAME_COUNT = 20 # 평가 1회 당 게임 수(오리지널: 400) EVAL_TEMPERATURE = 1.0 # 온도 파라미터

3번 모델은 2번 모델과 동일한 ResNet 기반 신경망을 적용하였으나, 합성곱 필터 수 (N_KERNEL = 64)와 Residual Block 개수 (N_RESIDUAL_BLOCK = 16)를 최적화하여 학습 용량을 조절하고자 노력했음을 알 수 있다. 특히, MCTS 탐색 횟수 (PV_EVALUATE_COUNT = 800)를 크게 증가시켜 더욱 깊은 탐색을 수행할 수 있도록 하였으며, Self-play 게임 수 (SP_GAME_COUNT = 20)를 비교적 적게 설정하여 더 짧은 학습 주기 내에 빠르게 모델을 개선하는 방식으로 설계되었다. 또한, BATCH_SIZE = 128, LEARNING_RATE = 0.001을 적용하여 학습 속도를 높였지만, 너무 빠른 학습 속도로 인해 최적의 정책을 충분히 학습하지 못했을 가능성이 있다. 평가 시 EVAL_GAME_COUNT = 20을 설정하여, 보다 신뢰성 있는 모델 평가를 수행하였다. 그러나 후공 시 패배율이 높았던 점을 고려하면, Self-play 게임 수가 적고 빠른 학습률로 인해 정확한 탐색이 부족했을 가능성이 있다.

다음은 ResNet 신경망을 차용한 4번 모델 설계 시 적용된 하이퍼 파라미터이다.

[표 4] 4번 모델 (ResNet 신경망)

```
[MCTS]
C_PCUT = 1.0 # 탐험과 탐욕적 행동 선택 균형
PV_EVALUATE_COUNT = 100

[TRAIN]
BOARD_SIZE = 3
LEARNING_RATE = 2e-4
MOMENTUM = 0.9
WEIGHT_DECAY = 1e-4
SIMULATIONS = 120
GAMES_PER_ITERATION = 8
EVALUATION_GAMES = 140
ITERATIONS = 3
```

4번 모델은 앞선 ResNet 모델들과 마찬가지로 Residual Network 기반이지만, MCTS 탐색 및 학습 전략을 최적화하여 가장 뛰어난 성능을 기록했다. LEARNING_RATE = 2e-4 및 MOMENTUM = 0.9를 적용하여 모델의 수렴 속도를 안정적으로 유지하였고, SIMULATIONS = 120을 설정하여 MCTS가 충분한 탐색을 수행할 수 있도록 하였다. 무엇보다도 탐험과 탐욕적 행동 선택 균형을 조절하는 C_PCUT = 1.0을 적용하여 후공에서도 지나치게 보수적이거나 과감한 결정을 내리지 않도록 조정한 것이 성능 향상의 일부 요인으로 보인다. 또한 평가 시 EVALUATION_GAMES = 140을 수행하여 모델의 성능을 충분히 검증하였으며, 결과적으로 다른 모델들보다 후공 시 패배율이 낮아 가장 안정적인 성능을 기록했다.

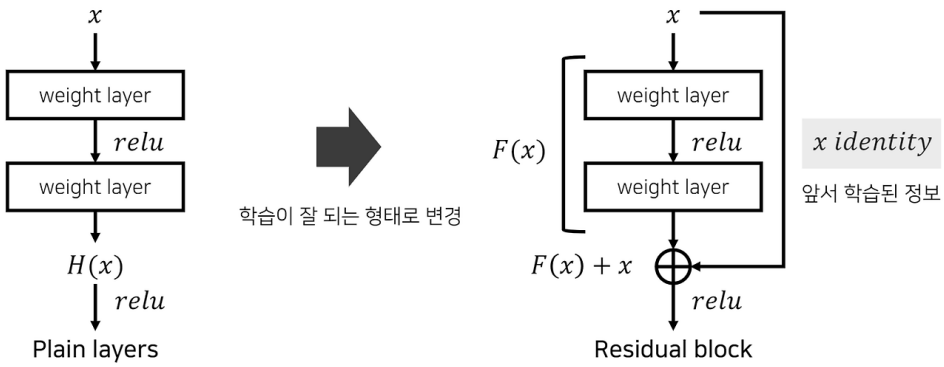
E. 신경망

1. 잔차 학습 (Residual Block)

딥러닝 모델에서 네트워크의 깊이가 증가하면 일반적으로 표현력은 향상되지만, 동시에 한 번에 학습해야 하는 Mapping의 복잡도가 기하급수적으로 증가하여 학습이 어려워지는 문제가 발생한다. 이와 같은 문제를 해결하기 위한 방법으로 잔차 학습 (Residual Block) 개념이 차용된 ResNet 신경망을 활용하는 방법이 있다.

Plain layer의 경우, 각 레이어는 입력 x로부터 완전히 새로운 Feature Map y=f(x)를 생성하도록 설계된다. 여기서 y는 기존의 정보를 모두 버리고, 오직 x로부터 새롭게 학습한 정보를 반영하는 결과이다. 이 방식은 네트워크가 고차원 Feature 공간으로 Mapping하는 부담이 커져, 층이 깊어질수록 학습해야 할 Mapping의 양이 많아지고, 결과적으로 최적화가 어려워지는 한계를 보인다.

[그림 5] Plane Layers와 Residual Block



이에 반해, 잔차 학습은 네트워크에 shortcut, 즉 입력값 x를 그대로 출력에 더해주는 지름길 연결을 도입한다. 이 구조에서는 각 레이어가 전체 Mapping H(x)를 직접 학습하는 대신, H(x)=F(x)+x라는 형태로 잔차 F(x)=H(x)−x만을 학습하도록 유도된다. 즉, 이미 학습된 x를 그대로 보존하고, 그 위에 추가적으로 학습해야 할 차이만을 보완하는 방식이다.

학습이 진행되어 네트워크의 깊이가 증가할수록 출력 H(x)는 점차 입력 x에 근접하게 되고, 이에 따라 추가 학습량 F(x)는 점점 작아져 이상적으로는 0에 수렴하게 된다. 즉, 네트워크가 충분히 학습되면 F(x)는 최소값에 도달하게 되고, 잔차가 거의 없어진 상태가 된다. 이러한 특성 덕분에 잔차 학습은 네트워크의 깊이를 크게 늘리더라도, 각 레이어가 학습해야 할 정보의 양을 줄여주어 최적화 과정에서의 어려움을 완화시킨다.

더불어, shortcut 연결은 단순히 입력 x를 출력에 덧셈으로 연결하는 방식이므로, 추가적인 파라미터를 필요로 하지 않으며 연산 비용에도 거의 영향을 주지 않는다. 따라서, 네트워크 구조 자체를 크게 변경할 필요 없이, 기존의 모델에 간단히 shortcut을 추가함으로써 효과적인 잔차 학습을 구현할 수 있다.

2. 듀얼 네트워크 (Dual Network)

앞서 정의한 잔차 학습이 적용된 듀엘 네트워크는 알파제로에서만 사용되어 알파고와 주된 차이점을 보인다. 알파고는 정책을 추론하는 정책 네트워크와 가치를 추론하는 가치 네트워크, 플레이아웃에 이용하는 롤아웃 정책의 3가지 뉴럴 네트워크를 사용한다. 이에 반면 알파제로는 Residual Block들을 연속적으로 쌓은 듀얼 네트워크를 사용하여, 정책 네트워크와 가치 네트워크를 함께 추론한다는 특징이 있다.

F. 성능

1. 새로운 모델과 기존 모델 간 대국 결과

모델 학습을 진행하면서 매 반복하다 evaluate_models() 메서드를 사용하여 새로운 모델과 기존 모델이 대국하여 승률을 평가하였다. 학습 반복 횟수는 3으로 설정하였고, 각 학습이 끝난 후 평가를 위해 진행하는 게임의 횟수는 140으로 설정하였다. 결과는 다음과 같다.

[표 5] 새로운 모델과 기존 모델 간 대국 결과

Iteration 1
New Model Wins: 83, Old Model Wins: 43, Loss: 0.13935230672359467, Value Loss: 0.00010535803448874503
New Model Win Rate: 65.87%
Iteration 2
New Model Wins: 69, Old Model Wins: 43, Loss: 0.29227298498153687, Value Loss: 2.382708953518886e-05
New Model Win Rate: 61.61%
Iteration 3
New Model Wins: 79, Old Model Wins: 39, Loss: 0.16655628383159637, Value Loss: 3.310593683636398e-06
New Model Win Rate: 66.95%

학습이 진행됨에 따라 승률이 60%대로 일정 수준 이상을 유지하며, 새로운 모델이 기존 모델보다 더 우수한 전략을 학습하고 있음을 보여준다. Iteration2에서 승률이 하락하고 손실이 증가하는 모습을 보이고 있지만, 이는 새로운 데이터 학습 과정에서 발생할 수 있는 일반적인 현상으로 보인다. 학습 후반부로 갈수록 승률이 높아지고 손실이 낮아지는 경향을 보이므로 모델이 안정적으로 최적화되고 있음을 알 수 있다. 또한 최종 Value Loss가 3.310593683636398e-06로 매우 낮아진 것으로 보아 모델이 승패 확률을 정확하게 예측할 수 있다고 판단할 수 있다.

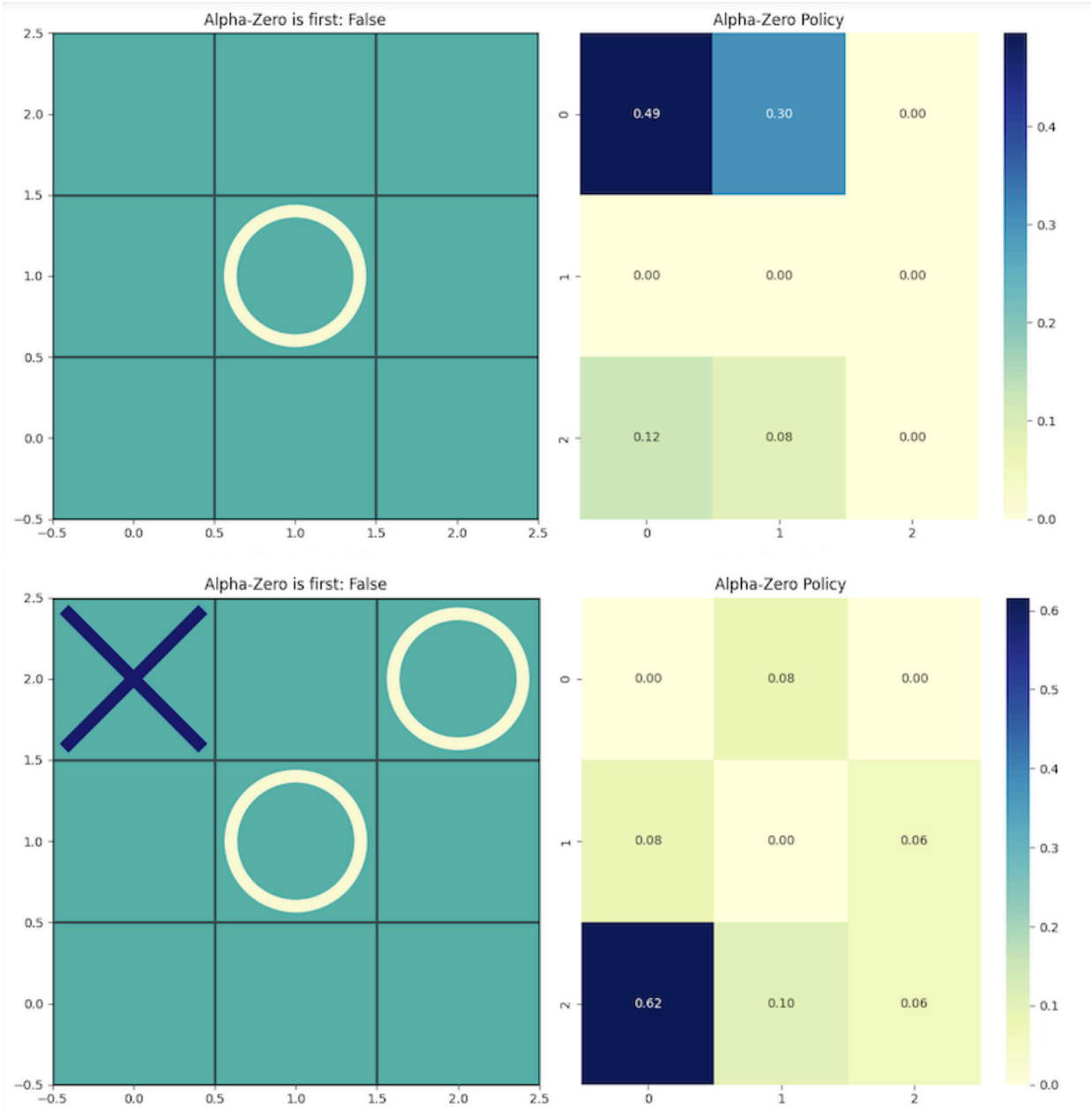
2. 대국 결과

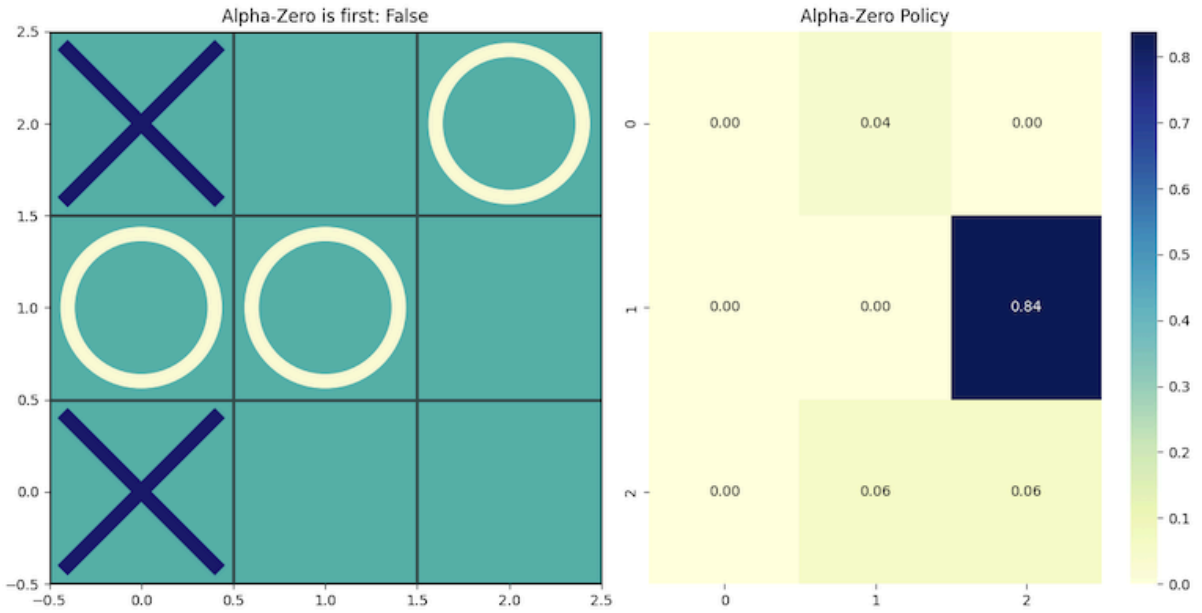
본 연구에서는 알파제로 기반의 AI가 후공 플레이할 때의 정책과 의사결정 과정을 시각적으로 분석하였다. O는 인간 플레이어의 수, X는 AI의 수를 의미한다. 오른쪽의 MCTS Policy는 AI가 각 위치를 선택할 확률을 나타내며, 색이 짙을수록 선택 확률이 높은 지점을 의미한다. AI는 탐색을 통해 후속 전략을 예측한 뒤, 승률이 높은 위치를 선택하여 착수하였다.

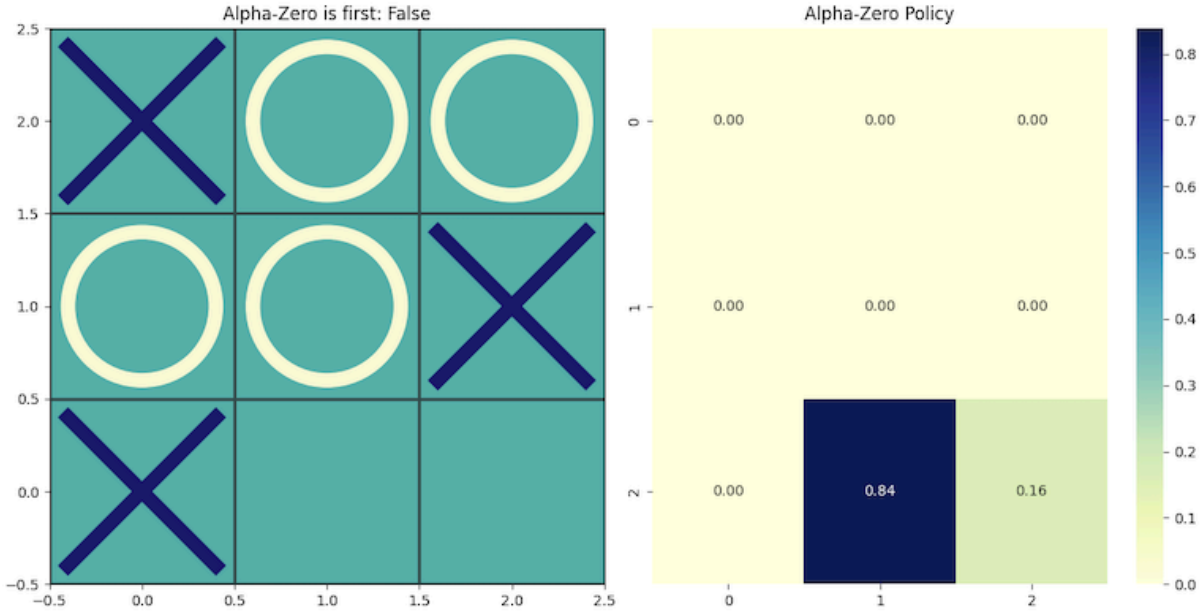
- **선공(인간, O)이 가운데, 후공(AI, X)가 모서리에 두어 무승부**

  [그림 6]은 선공(인간)이 중앙에 둔 후, 후공(AI)이 모서리에 둔 상황을 출발점으로 하여 AI가 최적의 수를 찾아 무승부를 이끌어낸 과정을 나타낸다. 다음은 알파제로와 대국을 시행했을 시, 현재 보드 상태에 대한 MCTS Policy를 담은은 시각화 자료이다.

[그림 6] Current Board State와 MCTS Policy (1)



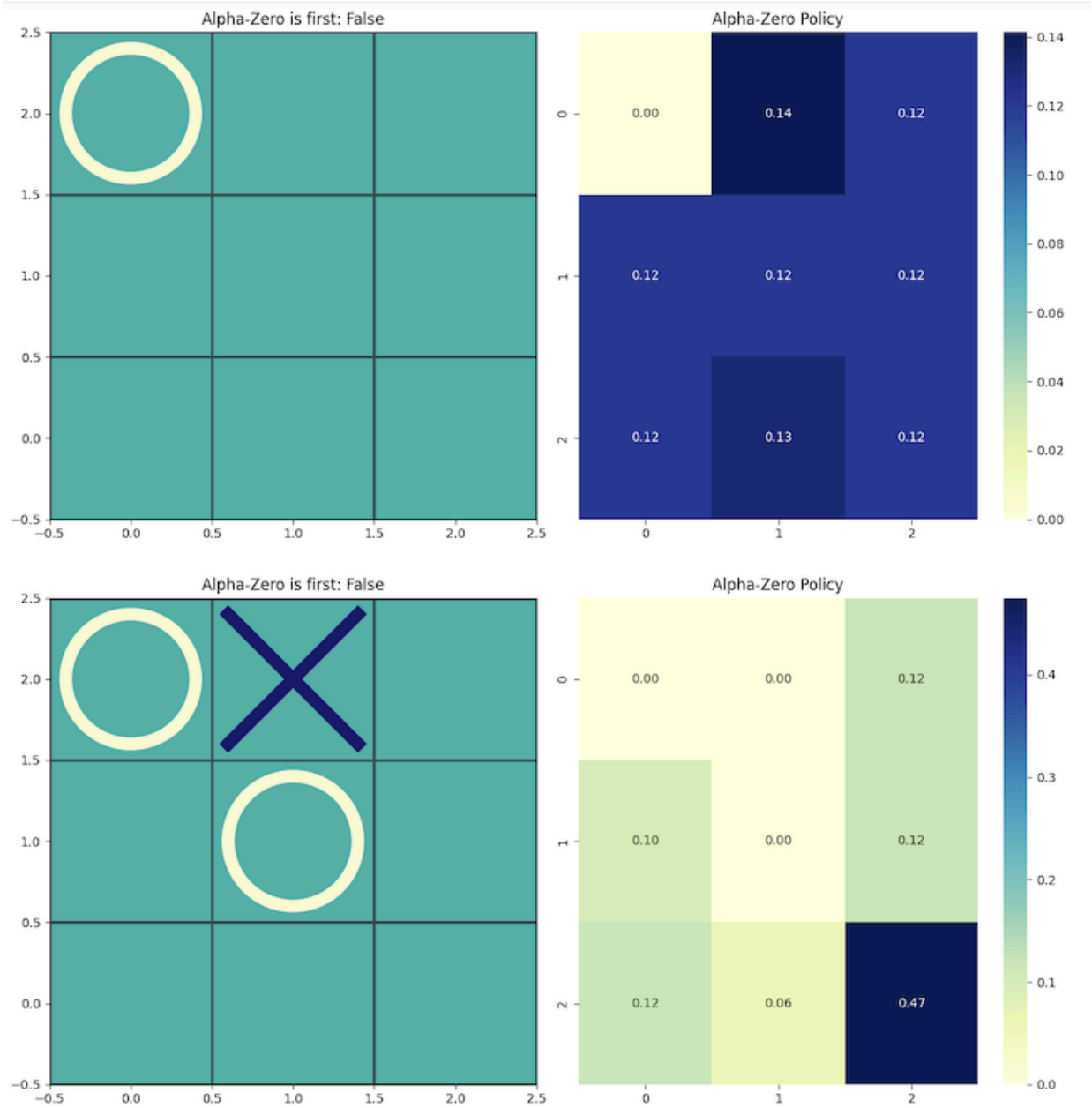


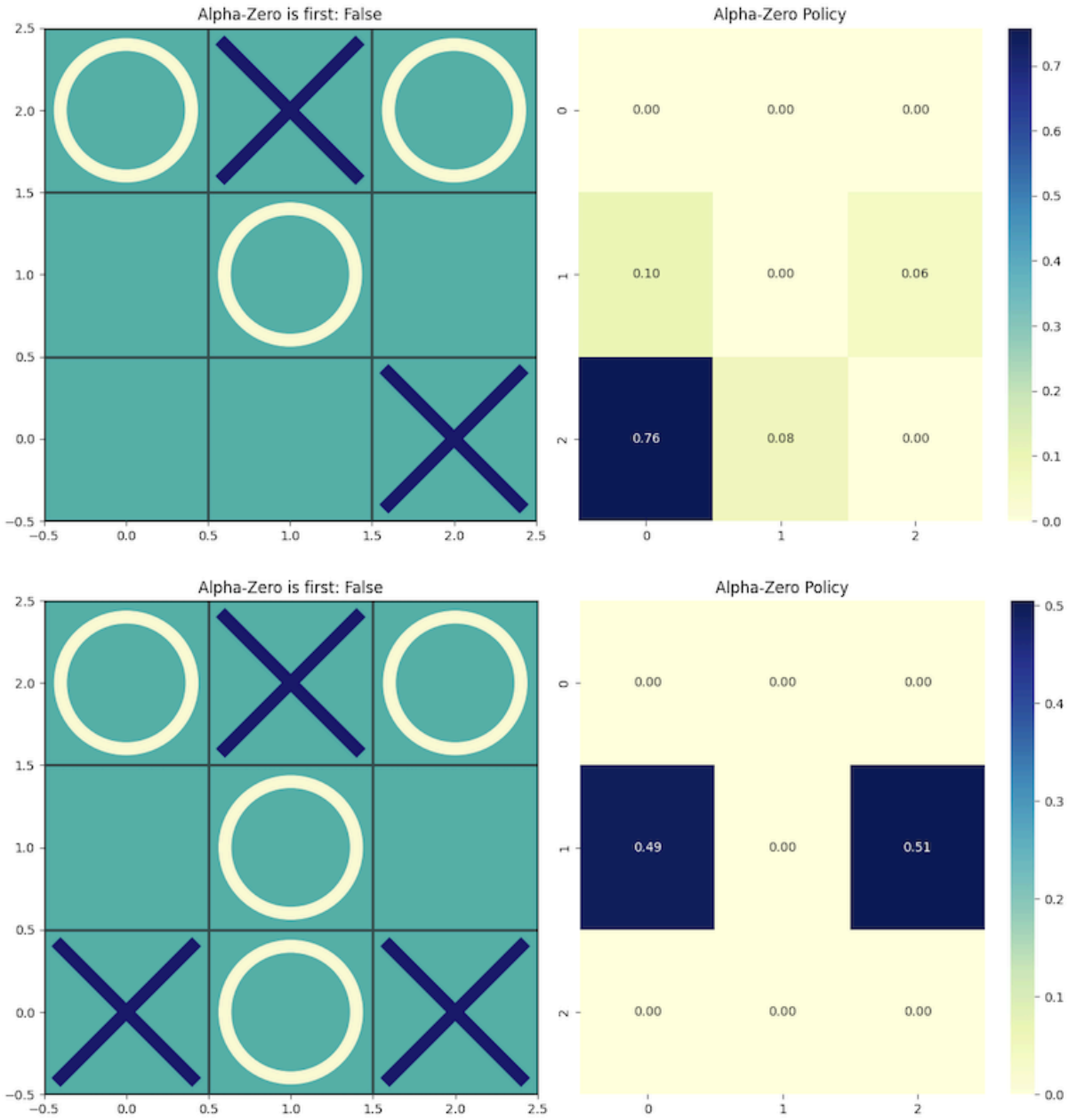


- **선공(인간, O)이 코너, 후공(AI, X)가 변에 두어 무승부**

  [그림 7]은 선공(인간)이 중앙에 둔 후, 후공(AI)이 변에 둔 상황을 출발점으로 하여 AI가 최적의 수를 찾아 무승부한 과정을 나타낸다.

[그림 7] Current Board State와 MCTS Policy (2)


Alpha-Zero is first: False
Alpha-Zero Policy
0.00
0.14
0.12
0.12
0.12
0.12
0.12
0.13
0.12
Alpha-Zero is first: False
Alpha-Zero Policy
0.00
0.00
0.12
0.10
0.00
0.12
0.12
0.06
0.47



일반적으로 틱택토에서 후공은 불리한 입장이지만, 본 연구에서 적용한 알파제로 기반 AI는 후공에서도 안정적인 성능을 보였다. 특히, 패배를 방지하기 위한 탐색 전략이 반영되어 후공 시 패배 확률이 낮아지는 경향을 보였다. 본 자료를 통해, AI가 탐색 기반으로 수를 선택하며 단순한 룰 기반 전략보다 뛰어난 적응력을 가질 수 있음을 시각적으로 확인할 수 있다.

3. CNC 모델 작동

다음은 현실에서 직접 작동할 수 있도록 알파제로를 CNC 모델과 연동하여 물리적 환경에서도 게임을 수행하는 실험을 진행한 사진이다.

[그림 8] CNC 모델 작동 사진

Ⅲ. 로보틱스

A. 구조 및 설계

CNC Plotter는 X, Y축의 정밀한 이동을 통해 2D 표면 위에 특정 도형이나 글씨를 그릴 수 있는 기계 장치이다. 본 연구에서는 틱택토 게임을 구현하기 위한 CNC Plotter의 구조와 설계를 다룬다. 해당 시스템은 프레임과 베이스, X·Y축 이동 장치, 펜 홀더, 제어 시스템 및 전원 공급 장치로 구성되며, 프레임은 주로 알루미늄 또는 3D 프린팅을 통해 제작된다. 본 프로젝트에서는 3D 프린팅을 활용하여 출력하였으며, 이를 통해 경량화와 정밀한 설계가 가능하도록 하였다.

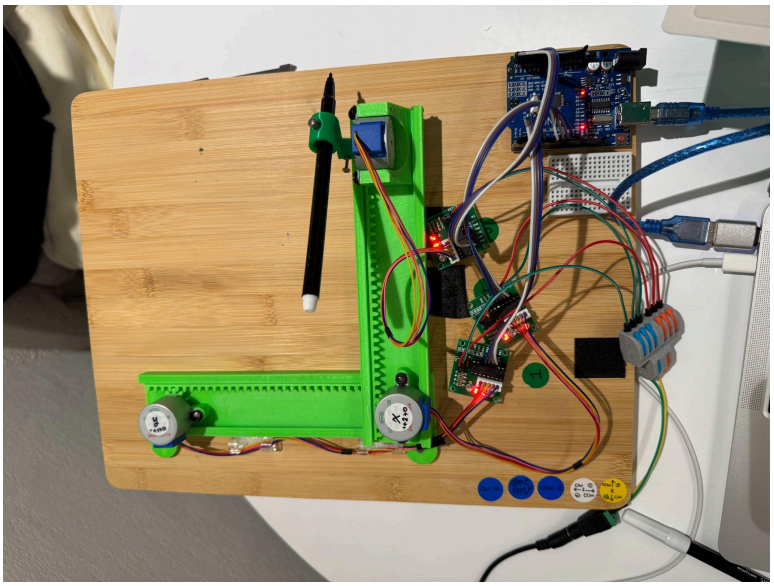
본 시스템의 주요 하드웨어는 Arduino Uno 한 개, Step Motor(28BYJ-48) 세 개, Motor Driver(ULN2003) 세 개로 구성되어 있으며, 이를 통해 X축과 Y축의 이동 및 펜의 위치를 정밀하게 제어한다. 각각의 스텝 모터는 해당 축의 움직임을 수행하며, 모터 드라이버는 아두이노에서 전달된 신호를 증폭하여 모터의 구동을 안정화하는 역할을 한다. 주요 하드웨어와 함께 사용한 부품은 [그림 6]과 같다.

[그림 9] 부품 목록표

부품명	개수
DM331 Mini BreadBoard	1
Arduino UNO	1
Step Motor 28BYJ-48	3
Step Motor Controller ULN2003	3
Barrel Jack with Terminal Block(5V)	1
Jumper Wires (F/M, M/M)	n
3D Printed Body	1

품목명	개수
5V Adapter	1
Super Glue	1
2M Bolts with large head	6 + 2 + 1
Pen	1
Chopping Board	1

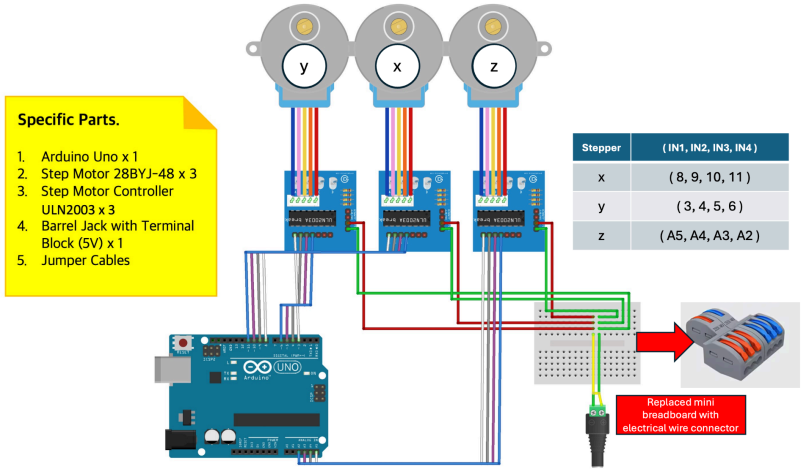
[그림 10] 실제 구현된 CNC Plotter



[그림 10]은 실제 구현한 CNC Plotter 모델과 하드웨어 배치를 나타내며, 최적의 가동 범위를 고려하여 설계되었음을 보여준다. 시스템의 상단에는 Arduino Uno가 위치하고 있으며, 하단에는 X축, Y축, Z축을 제어하는 모터 드라이버가 나란히 배치되어 있다. 각 모터는 지정된 축을 따라 움직이며, 아두이노는 사전에 입력된 명령을 바탕으로 모터 드라이버에 신호를 전달하여 각 축을 제어한다. 이러한 구성은 시스템의 정밀도를 유지하면서도 효율적인 동작을 보장하도록 설계되었다.

틱택토 보드의 좌표계는 3×3의 격자로 구성되며, 각 위치는 0~8까지의 번호로 표현된다. X축은 좌우 이동을, Y축은 상하 이동을 담당하며, 각 칸의 중심 좌표에 정확히 도달하기 위해 모터의 이동 거리의 정밀한 조정이 중요하다. 또한 틱택토 보드는 실시간으로 카메라를 통해 캡쳐되며 보드의 외곽 선들과 O, X를 인식하여 게임을 진행할 수 있도록 한다.

[그림 11] CNC Plotter의 하드웨어 회로도



[그림 8]은 CNC Plotter의 전자 회로도를 나타내며, 본 시스템의 제어는 Arduino Uno를 중심으로 이루어진다. CNC Plotter는 X, Y, Z축의 Step Motor(28BYJ-48)를 정밀하게 제어하기 위해 각각의 모터 드라이버(ULN2003)를 사용한다. 아두이노는 디지털 핀을 통해 모터 드라이버에 방향 및 스텝 신호를 전달하며, 모터 드라이버는 해당 신호를 증폭하여 모터가 안정적으로 동작하도록 한다. 이를 통해 CNC Plotter는 지정된 좌표로 이동하여 필기 도구를 조작할 수 있도록 설계되었다. 전원 공급은 5V Adapter를 이용하여 이루어지며, 각 모터 드라이버 및 아두이노에 안정적인 전원을 제공하기 위해 Barrel Jack with Terminal Block이 사용되었다. 기존의 미니 브레드보드 대신 전선 연결 단자(Wire Connector)를 사용함으로써 기존 브레드보드의 연결 불안정 문제를 해결하고, 보다 견고한 전기적 연결이 가능하도록 하였다. 각 스텝 모터의 연결 방식은 다음과 같다. X축 모터 드라이버는 Arduino Uno의 8, 9, 10, 11번 핀에 연결되며, Y축 모터 드라이버는 3, 4, 5, 6번 핀에, Z축(펜 조작) 모터 드라이버는 A5, A4, A3, A2번 핀에 각각 연결된다. 이를 통해 각 모터는 독립적으로 제어될 수 있으며, 지정된 좌표로 이동하는 기능을 수행한다.

B. 기능

본 프로젝트에서 사용되는 CNC Plotter 로봇은 틱택토 보드의 X와 O를 물리적으로 그려내는 역할을 수행한다. 강화학습 모델이 학습한 결과를 실제 물리적 환경에서 시각적으로 표현하기 위해, X축과 Y축을 조작하여 필기 도구(펜)를 특정 좌표로 이동시키는 방식으로 동작한다. 이를 통해 강화학습 모델의 성능을 직접적으로 평가하고, 시뮬레이션과 실제 환경 간의 차이를 분석할 수 있다.

1. 이동 원리

시스템은 세 개의 스텝 모터(28BYJ-48)와 ULN2003 모터 드라이버를 활용하여 정밀한 위치 제어를 수행한다. X축과 Y축 모터는 아두이노에서 발생한 신호를 기반으로 동작하며, 지정된 좌표로 이동하기 위해 스텝 단위로 회전한다. 스텝 모터는 일정한 각도로 회전하는 특성을 가지며, 이를 활용하여 기계적인 이동 거리를 정밀하게 조정할 수 있다. 각 축의 이동은 다음과 같이 이루어진다.

[그림 12] 스텝 모터의 주요 사양

Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)

스텝 모터는 내부에 여러 개의 코일이 배치되어 있으며, 이 코일들이 순차적으로 전류를 공급받음으로써 로터(Rotor)가 한 단계씩 회전한다. 이때, 스텝 모터의 회전은 지정된 각도(스텝 각, Step Angle)에 따라 이루어지며, 입력된 펄스 수에 따라 회전 각도가 결정된다. [그림 9] 스텝 모터의 data sheet을 보면, 한 스텝(step)당 5.625° 만큼 회전하며, 총 64개의 스텝으로 360° 회전을 한다. 하지만, 기어 감속 장치(1/64 비율)를 포함하면 실제로 4096 스텝(= 64 × 64) 후에 한 바퀴(360°)를 회전하게 된다. 즉, 4096번의 스텝을 수행해야 모터가 정확히 360° 를 회전하게 된다.

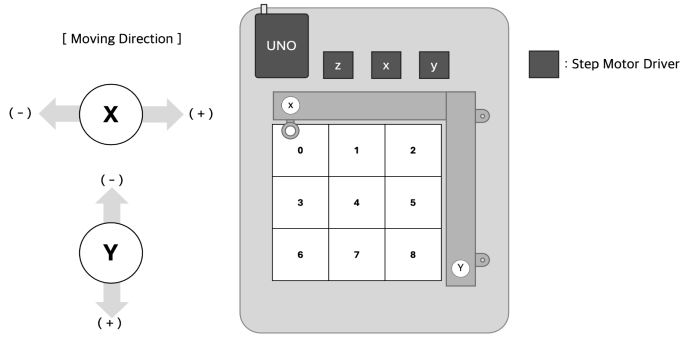
$$\text{총 스텝 수} = \frac{360^\circ}{\text{스텝 각}} \times \text{기어비} = \frac{360^\circ}{5.625^\circ} \times 64 = 4096$$

스텝 모터는 직접적으로 마이크로컨트롤러에 연결하여 구동할 수 없으며, 모터 드라이버를 통해 전류를 증폭한 후 제어해야 한다. ULN2003은 스텝 모터의 코일을 순차적으로 활성화하여, 정해진 방향으로 회전하도록 한다.

아두이노는 ULN2003 모터 드라이버를 통해 스텝 모터를 제어하는 역할을 수행한다. 먼저, 아두이노에서 특정 방향과 회전 단계를 지정하는 제어 신호를 생성하여 ULN2003 모터 드라이버로 전달한다. ULN2003은 이 신호를 증폭하여 스텝 모터에 충분한 전류를 공급하며, 각 코일에 순차적으로 전력을 인가함으로써 모터가 일정한 각도로 회전하도록 한다. 스텝 모터는 ULN2003에서 받은 신호에 따라 정확한 각도로 이동하며, 한 스텝(step)씩 회전하는 방식으로 목표한 위치로 이동하게 된다. 이러한 방식은 일반적인 DC 모터와 달리 지정된 위치에서 정밀하게 정지할 수 있도록 설계되었으며, CNC Plotter와 같은 정밀 제어가 필요한 시스템에서 필수적으로 사용된다.

2. 보드 인식 및 출력 방식

[그림 13] Arduino Uno 기반 CNC Plotter의 전체 배치도



틱택토 를 기반으로

특정한 좌표를 인식하고 출력해야 한다. 게임 보드는 0번부터 8번까지의 번호로 구분된 총 9개의 칸으로 나누어지며, 각 칸의 중심 좌표에 정확히 도달할 수 있도록 시스템이 설계되었다. 틱택토 보드 좌표를 CNC Plotter의 물리적 좌표로 변환하는 과정은 다음과 같다. 먼저, 실시간 감지 카메라를 통해서 보드를 캡쳐한 후 AI 모델이 결정한 수(0~8)에 해당하는 틱택토 보드의 좌표를 읽어들이고, 이를 CNC Plotter의 X, Y축 기준 좌표로 변환한다. 변환된 좌표 값은 모터의 스텝 수로 변환되며, 아두이노에서 계산된 이동 명령이 X, Y축 스텝 모터로 전달된다. X축 모터가 먼저 목표 위치로 이동한 후, Y축 모터가 해당 위치로 조정된다. 그 후 Z축 서보 모터가 펜을 내려 X 또는 O를 기록하는 방식으로 게임 결과가 출력된다. 이러한 방식으로 CNC Plotter는 AI 모델이 예측한 수를 바탕으로 자동으로 게임 진행 상황을 물리적으로 표현하는 기능을 수행한다.

C. CNC 로봇 제어

CNC 로봇이 틱택토 게임을 물리적으로 표현하는 기능을 수행하도록 제어하는 코드는 모두 C++ 클래스로 구현하였다. 아두이노 프로그램에서 해당 모듈들을 import하여 사용한다. CNC 로봇 제어 코드는 `Stepper`, `MultiStepper`, `TicTacToeArtist` 이렇게 총 세 개의 클래스로 구성되어 있다. Stepper 클래스는 CNC 로봇을 구성하는 세 개의 스텝모터 각각을 제어하는 역할을 한다. MultiStepper 클래스는 대각선을 긋기 위해 X축 스텝모터와 Y축 스텝모터를 동시에 작동하도록 하는 역할을 한다. TicTacToe Artist 클래스는 앞선 두 클래스를 이용해 스텝모터가 틱택토 게임을 표현하는 특정 명령을 수행하도록 각 명령을 모듈화한 클래스로, 게임보드 그리기, 특정 위치로 이동하기, O 혹은 X를 그리기 등의 행동을 하도록 명령할 수 있다. 본 프로젝트에서는 로봇이 정확한 길이만큼 움직일 수 있도록 '모눈력'이라는 새로운 길이 단위를 정의하였다. TicTacToeArtist 클래스에서 사용하는 모든 길이는 모눈력 기준으로 구현되어 있다.

1. Stepper

CNC 로봇의 X, Y, Z(펜)축을 담당하는 각 스텝모터를 제어하는 클래스이다. 각 스텝모터마다 아두이노 보드와 연결되어 있는 4개의 모터핀을 순서대로 입력받아 각 스텝모터를 제어한다.

step메서드에서 스텝모터가 움직일 스텝과 방향(시계 방향:1/반시계 방향:-1)을 입력받아 스텝모터에 알맞은 신호를 보낸다. 각 모터핀에 신호를 주는 순서는 〈그림 11〉에 명시하였다. 신호를 줄 때 한 스텝마다 1밀리초의 딜레이를 추가해 스텝모터가 안정적으로 움직이도록 하였다. 내부 메소드인 controlSignal에서 스텝모터가 움직일 방향에 맞춰 신호를 주는 순서를 정한다.

[그림 14] 스텝모터 회전 방향에 따른 신호 순서

clockwise

step	IN1	IN2	IN3	IN4
0	0	0	0	1
1	0	0	1	1
2	0	0	1	0
3	0	1	1	0
4	0	1	0	0
5	1	1	0	0
6	1	0	0	0
7	1	0	0	1

counter clockwise

.

2. MultiStepper

CNC 로봇의 X, Y축 스텝모터를 동시에 움직이도록 제어하는 클래스이다. X, Y축 스텝모터를 제어하는 각각의 Stepper 객체를 입력받아 두 스텝모터의 모터핀을 가져와서 동시에 제어한다.

Stepper 클래스의 step 메서드와 동일하게, 움직일 스텝 수와 X, Y축 스텝모터가 각각 움직일 방향을 입력받는다. X와 Y축이 같은 방향으로 움직이면 오른쪽 아래를 향하는 대각선을, 서로 다른 방향으로 움직이면 오른쪽 위를 향하는 대각선을 그린다. 위의 〈그림 11〉에서 볼 수 있는 모터핀 신호 전달 방향과 동일하게 신호를 전달하는데, 한 스텝마다 X, Y축 스텝모터의 8개 모터핀에 동시에 신호를 전달하는 것이 다른 점이다. 이외에는 Stepper 클래스와 동일하다.

3. TicTacToeArtist

CNC 로봇의 전체적인 움직임에 대한 메서드를 구현한 클래스이다. 각 스텝모터에 대한 Stepper 객체와 MultiStepper 객체를 입력받아 제어한다. 그리고 전체 게임보드의 크기를 입력받고, 내부적으로 3×3의 게임보드 한 칸의 길이와 게임보드 크기에 맞는 X, O 한 변의 길이를 계산한다.

총 9개의 좌표로 이동해 그림을 그릴 수 있다. 행동의 종류에는 크게 1) 게임보드 그리기, 2) O그리기, 3) X 그리기 가 있다. 게임보드는 입력값 없이 게임보드 크기에 맞춰서 게임보드를 그린다. O, X를 그리는 행동을 명령할 때는 어떤 좌표에 그릴지 좌표(0~8)를 입력하면, 해당 좌표로 이동한 후 그림을 그린다. 한 명령을 수행한 뒤에는 무조건 초기 위치(좌표 0)로 돌아온다.

D. 문제 해결 과정

1. 구조적 문제 및 해결 과정

본 프로젝트에서 구현한 CNC Plotter는 3D 프린팅을 활용하여 주요 부품을 제작하였으며, 이를 통해 경량화와 정밀한 설계를 가능하게 하였다. 그러나 3D 프린팅을 이용한 제작 과정에서 구조적 단차 및 조립 정밀도 문제가 발생하였으며, 이로 인해 시스템의 성능에 영향을 미치는 몇 가지 문제점이 발견되었다.

첫 번째 문제는 모터 구동 시 헛도는 현상이었다. 이는 3D 프린팅을 통해 출력된 부품 간의 미세한 단차로 인해 기계적 정렬이 완벽하게 이루어지지 않았기 때문이다. 특히, 스텝 모터와 X·Y축을 연결하는 구동 부위에서 발생한 단차로 인해 특정 위치에서 회전력이 충분히 전달되지 못하는 문제가 발생하였다. 이로 인해 모터가 정상적으로 동작하지 않고, 목표한 위치로의 이동이 불안정해지는 현상이 관찰되었다.

두 번째 문제는 X·Y축 레일 부품간의 미세한 규격 오차로 인해 발생한 문제였다. 주요 부품 간의 결합부에서 일정한 간격(틈)을 유지하지 못하였고, 이러한 비균일한 결합 상태로 인해, X·Y축을 따라 이동할 때 일부 구간에서는 비교적 원활하게 움직이지만, 다른 구간에서는 축의 움직임이 뻑뻑해지고 매끄러운 이동이 어려운 문제가 발생하였다. 특히, 일정한 속도로 움직여야 하는 스텝 모터가 특정 구간에서 저항을 받으며 출력이 불안정해지고, 반복적인 마찰로 인해 부품의 마모가 가속화될 가능성이 있었다. 이러한 문제를 해결하기 위해, 레일 부품의 하단 부분의 사포 작업(Sanding)을 통해 마찰력을 조정하여 부드러운 이동이 가능하도록 개선하였다. 이를 통해 레일과 프레임 간의 간격을 균일하게 유지하여 CNC Plotter가 보다 안정적이고 일정하게 동작할 수 있도록 하였다.

2. 소프트웨어적 문제 및 해결 과정

CNC Plotter를 움직이는 스텝모터를 제어하는 프로그램을 사용하는 과정에서 문제점이 발견되었다. 스텝모터를 제어하는데 주로 사용하는 Arduino 라이브러리 'Stepper'를 사용해 스텝모터를 제어하려고 하였으나, 해당 라이브러리에서 종종 발견되는 문제점으로 알려진 한 방향으로만 회전하는 현상이 관찰되었다. 따라서 라이브러리를 사용하지 않고 직접적인 제어 방법인 스텝모터의 모터핀에 순서대로 신호를 전달해 회전시키는 방법을 사용하였고, 이를 바탕으로 스텝모터를 제어하는 C++ 클래스를 직접 구현하였다. 자체적으로 구현한 스텝모터 제어 라이브러리는 시리얼 신호를 주고받는 것과 스텝모터를 작동하는 행동을 동시에 실행할 수 있는 동기식 신호전달이 불가능하지만, 비동기식으로 작동할 수 있으며 직접 모터핀에 신호를 주는 방식이므로 안정적이다. 또한 내부의 delay 시간을 조절해 회전 속도를 조절할 수 있다.

E. Arduino와 Python 통신

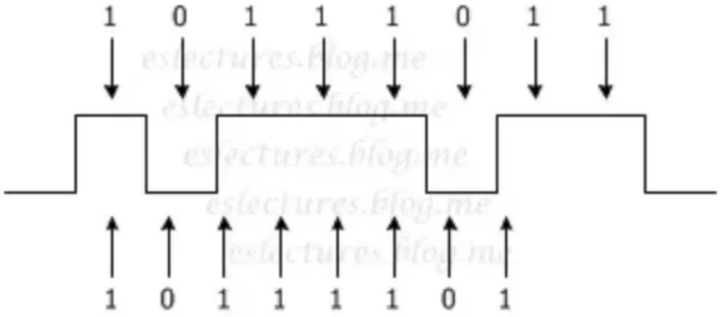
본 프로젝트는 로보틱스를 제외한 모든 부분을 Python으로 구현하였다. Python으로 구현한 Alpha-Zero 모델이 예측한 수를 CNC 로봇이 입력 받아 물리적으로 표현하기 위해 Python과 아두이노 프로그램(이하 아두이노)의 통신이 필요했고, 아두이노의 통신 방식을 고려해 USB(Universal Serial Bus)를 통한 시리얼 통신을 사용하였다. 전체 코드가 실행되는 Python 코드에서 AI가 행동을 예측한 후, 시리얼 통신을 통해 아두이노에 명령을 전달한다. 그리고 CNC 로봇이 행동을 마친 후에 아두이노에서 같은 방식으로 Python에 명령을 전달하고, Python은 명령을 받은 후에 다음 단계를 진행한다. 구체적인 작동 과정은 다음과 같다.

1) 아두이노는 Python으로부터 새로운 Serial 신호가 들어올 때까지 계속해서 대기하고, Serial 신호가 들어오면 실행한다. 명령을 모두 시행한 뒤에 다시 대기한다.
2) Python에서 아두이노로 Serial port를 통해 명령을 전송한다. 전송 후 아두이노로부터 완료 신호가 도착할 때까지 대기한다.
3) 아두이노에서 Python으로부터 도착한 명령을 읽고 실행한다. 실행 후 완료 신호를 Python에 전송한다.
4) 아두이노로부터 도착한 완료 신호를 받은 후, Python에서 다음 단계를 진행한다.

1. 시리얼 통신 (Serial communication)

시리얼 통신은 서로 다른 컨트롤러를 직렬로 연결하여 통신하는 방법 중 하나이다. 직렬 통신은 데이터를 한 번에 1bit씩 전송하는 방식으로, 데이터를 이진 펄스의 형태로 전송하고 이진 1을 논리 HIGH, 이진 0을 논리 LOW로 나타낸다. 송신부(Transmitter)가 이진 펄스 형태로 데이터를 전송하면, 수신부(Receiver)가 높고 낮음을 구분하여 의미있는 데이터로 읽는다. 이때 Transmitter와 Receiver 사이의 초당 전송 bits수인 Baud Rate를 양쪽 모두 동일하게 설정해야 한다. Baud Rate는 일반적으로 사용하는 숫자가 정해져 있으며, 본 프로젝트에서는 Baud Rate를 115200으로 설정하였다. 또한 Transmitter와 Receiver는 클럭(Clock)이 동기화되어 있어야 한다. Clock이란 어떠한 시점에서 이진 펄스를 논리정보로 읽을 것인지에 대한 시점을 정하는 기준이다. Clock이 상대적으로 느릴때와 빠를 때, 같은 이진 펄스를 다른 데이터로 읽게 되므로 Transmitter와 Receiver의 Clock을 맞춰주는 Clock Synchronization이 시리얼 통신에서 필수적이다.

[그림 15] 이진 펄스와 Clock



Clock Synchronization은 인터페이스 종류에 따라 동기와 비동기 방식으로 구분된다. 본 프로젝트에서는 시리얼 통신에 동기식 프로토콜인 USB를 사용하기 때문에 동기식 시리얼 통신을 사용하였다. 동기식 시리얼 통신과 비동기식 시리얼 통신에 대한 구체적인 설명은 다음과 같다.

1) 동기식 시리얼 통신: 각각의 컨트롤러가 이러한 신호 주기로 보내겠다는 Clock을 선언하고 데이터를 전송하는 방법이다. 따라서 데이터를 보내는 데이터선과 Clock 주기를 맞추는 클럭선이 필요하다. 보내는 Clock을 미리 맞춘 후 데이터를 주고 받기 때문에 오류가 적고 빠른 데이터를 통신할 수 있다. 동기식 프로토콜에는 USB 등이 있다.
2) 비동기식 시리얼 통신: 따로 주기를 정하지 않고 시작과 끝만 알려준 후 전송하는 방법이다. 데이터선만 필요하기 때문에 단순하고 빠르게 통신 회선을 구성할 수 있다. 하지만 동기 시리얼 통신에 비해 안정성이나 속도가 느리다. 비동기식 프로토콜에는 RS232 등이 있다.

아두이노 프로그램과 아두이노 보드가 통신하는 방식도 시리얼 통신인데, 이는 아두이노 프로그램에서 코드를 컴파일하고 컴퓨터의 가상 시리얼 포트(Serial port)를 거쳐 USB를 통해 아두이노 보드의 Serial port로 전송되는 구조이다.

2. 구현

Python에서 시리얼 통신을 하기 위한 라이브러리인 ‘pySerial’과 아두이노 내장 라이브러리 ‘Serial’을 사용하였다. 모든 코드가 loop() 안에서 실행되는 아두이노의 특성 상, Serial.available()를 사용해 새로운 시리얼 신호가 들어올 때까지 대기하고 시리얼 신호가 들어오면 코드를 실행하도록 하였다. 강제로 종료하지 않는 한 명령을 실행한 뒤에는 다시 대기한다. 아두이노로 전달할 수 있는 명령은 다음과 같다.

1) “S”: 게임보드를 그린다.
2) “X” or “O” + “0~8 구간의 정수” 조합: “X”또는 “O”를 해당 위치에 그린다.

아두이노에서는 파이썬이 보낸 시리얼 명령을 한 줄 읽고, 앞의 문자를 char 자료형으로 저장하고, 뒤의 문자를 int 형으로 변환하여 저장한다. 그리고 이에 따라 해당 명령을 수행한다.

Ⅳ. CV

A. 기능 및 성능

틱택토 보드 인식 시스템은 다음과 같은 주요 기능을 수행한다.

1. 카메라 영상 입력 처리: 실시간으로 카메라에서 틱택토 보드를 캡처한다.
2. 보드 영역 추출: 보드의 외곽선을 인식하고, 왜곡된 보드를 정사각형 형태로 보정한다.
3. CNN 기반 셀 분류: 틱택토 게임을 진행하는 데 있어 보드의 X와 O를 구분하는 역할을 수행한다. 다양한 형태의 X, O, 빈칸 이미지들을 CNN 기법을 활용해 학습한 모델로, 선의 두께나 위치와 관계없이 정확하게 인식할 수 있다. Fig ?.에서 볼 수 있듯이 training loss, validation loss 모두 0에 수렴할 정도까지 학습시킨 결과, 새로운 8개의 이미지 데이터 테스트에서 정확도 100%를 보였다. 이를 통해 틱택토 보드를 실시간으로 인식시킬 때 X와 O를 올바르게 판별해낸다.

[그림 16] Training Loss와 Validation Loss 시각화 지표

4. 상태 변화 감지: 보드의 상태 변화를 감지하고, 일정 시간동안 안정적인 상태를 유지하면 최종 보드 상태로 확정한다.

B. 핵심 구현부 및 트러블슈팅

1. 핵심구현부

1) 이미지 전처리 (Preprocessing)

틱택토 보드는 카메라로 촬영된 이미지에서 추출되므로, 안정적인 인식을 위해 여러 전처리 기법을 적용하였다. 먼저 cv2.cvtColor을 통해 입력 영상을 흑백 이미지로 변환하여 계산량을 줄이고, 엣지 검출을 효과적으로 수행할 수 있도록 하였다. cv2.Canny와 cv2.dilate를 사용하여 보드의 윤곽선을 감지하고 굵게 만들어 보드의 경계를 강조하였다.

2) 보드 영역 추출 및 투시 변환

cv2.findContours를 통해 이미지에서 윤곽선을 찾고, 윤곽선 내부의 면적을 비교하여 가장 큰 면적을 가진 영역을 보드 영역으로 인식하도록 하였다. cv2.approxPolyDP를 통해 검출된 보드 영역의 4개 모서리를 정렬하여 투시 변환의 기준점으로 설정하였다.

cv2.getPerspectiveTransform으로 비정형 사각형 모양으로 인식된 보드를 정사각형 형태로 변환하여 CNN 모델이 일관된 입력을 받을 수 있도록 하였다.

3) CNN 기반 셀 분류

CNN 모델을 학습시키는 데 있어 데이터 생성과 전처리, 데이터 증강, 모델 구조, 모델 학습 및 평가 4가지의 단계로 진행했다. 로봇과 사람이 펜을 이용해 번갈아 X, O를 그려야 한다는 점을 고려하여 아이패드에 직접 그리는 방식으로 이미지 데이터를 생성했다. 이때 보드판은 정확하게 9등분으로 나누어 그리되, X와 O는 최대한 다양한 모양이 될 수 있도록 그렸다. 지도 학습을 위해 각 그림을 보고 O는 0, X는 1, 빈칸은 2로 직접 라벨링하는 방식을 사용했다. 이후 3가지 증강기법을 적용해 데이터를 총 453개로 늘렸다.

모델은 이미지 처리에 유용한 합성곱 신경망(CNN, Convolutional Neural Network)을 기반으로, 입력으로 3채널을 갖는 3×3 크기의 보드를 받아 각 셀에 대해 3개의 클래스로 분류하는 방식이다. 총 3개의 합성곱과 2개의 완전연결 계층으로 이루어져 있으며 과적합을 방지하고자 배치 정규화와 드롭 아웃을 추가했다. 주요 하이퍼파라미터는 Fig ?.에 제시되어 있다.

데이터 개수가 총 453개로 비교적 적다는 점을 고려하여, 모델 학습 단계에서 데이터를 6:4 비율로 훈련(train)과 검증(validation) 세트로 다시 나누어 평가를 병행하였다. 배치 크기는 16으로 설정하고, 손실 함수로 CrossEntropyLoss를 사용했다. 또한, 성능 향상을 위해 RMSprop 옵티마이저와 StepLR 스케줄러를 함께 적용하여 보다 세밀한 조정을 하고자 했으며, 각 하이퍼파라미터 값은 학습률 0.00035, step size 5, gamma

0.95로 설정하였다. 총 280 epoch 동안 학습한 결과, train loss와 validation loss 모두 0에 매우 근접하는 값을 도출하였다. 이를 바탕으로 새로운 8개의 이미지 데이터를 테스트한 결과, 100%의 정확도를 기록하였으며, 추후 실시간으로 틱택토 보드를 인식할 때에도 매우 높은 정확도를 보였다.

[표 6] CNN Hyperparameter

계층	입력 크기	필터 수	커널 크기	패딩	활성화 함수	배치 정규화	풀링
합성곱1	(3, 3, 3)	64	3x3	1	ReLU	O	2x2(Max)
합성곱2	(64, 3, 3)	128	3x3	1	ReLU	O	2x2(Max)
합성곱3	(128, 3, 3)	256	3x3	1	ReLU	O	2x2(Max)
완전연결1	256x4x4	-	-	-	-	dropout (0.2)	-
완전연결2 (출력층)	128	-	-	-	-	-	-
출력크기	(9,3)	-	-	-	-	-	-

4) 안정적인 보드 상태 감지

틱택토 보드를 인식하는 과정에서 조명이나 카메라 설정 등 환경에 따라 잘못된 보드 인식이 일어날 수 있다. 또한 사람과 AI가 함께 플레이하기때문에 사람의 손이 프레임에 들어올 때 상태가 변동할 수 있다. 따라서 불안정한 인식을 방지하기 위해 일정 시간 동안 동일한 상태가 유지될 때만 확정된 보드 상태로 판단하도록 하였다.

- 이전 예측값과 비교 : 현재 예측값과 이전 프레임의 예측값이 일정 시간(stability_duration) 동안 동일하면 확정된 상태로 간주한다.
- 보드 변화 감지 : 변화가 감지될 경우 타이머를 초기화하여 새 상태를 반영할 시간을 확보한다.
- 최종 상태 출력 : 충분한 안정성이 확보된 보드 상태만 최종적으로 반환하여 에이전트에게 전달한다.

2. 트러블슈팅

1) 보드 인식 과정

보드 인식 과정에서 문제는 카메라가 수직에서 보드를 촬영하면 로봇이 보드 영역을 침범하여 보드 인식이 이루지지 않아 카메라 각도를 기울여서 보드를 촬영해야 하고, 기존에 이를 고려하지 않고 작성한 코드는 정사각형인 보드 윤곽선만 잘 인식한다는 것이었다. 따라서 기울어진 카메라 각도 때문에 비정형 사각형 모양으로 인식되는 보드를 정사각형으로 변환하는 것이 관건이었다.

이를 해결하기 위해서 투시변환을 사용하였다. 먼저 cv2.approxPolyDP를 사용해 비정형 사각형 보드의 꼭짓점을 인식하고, 만약 꼭짓점이 4개로 잘 인식이 되었다면 네 꼭짓점(sorted_points)들을 투시 변환의 대상으로 정한다. 정사각형의 크기를 설정하고 그 크기에 맞는 정사각형의 네 꼭짓점, 즉 목표 꼭짓점(dst_points)를 설정한다. cv2.getPerspectiveTransform(sorted_points, dst_points)를 통해 투시 변환 행렬을 계산한다. 마지막으로 cv2.warpPerspective 통해 카메라 프레임 이미지에 투시 변환을 적용하면 정사각형으로 보정된 보드 이미지가 반환된다.

이를 통해 틱택토 보드가 정사각형이 아니거나 카메라 각도로 인해 비뚤어진 사각형으로 인식되더라도 정사각형으로 보정되어 CNN 모델에 전달될 수 있게 하였다.

2) CNN 모델의 학습 과정

CNN 모델의 학습 과정에서 가장 큰 문제는 과적합이었다. 하이퍼파라미터를 조정하는 것만으로는 loss 값을 일정 수준 이상 감소시키는 것이 어려웠으며, 그 원인을 과적합으로 판단하였다. 이를 해결하기 위해 데이터 불균형을 해소하고, 데이터 양을 조정하는 것뿐만 아니라 배치정규화와 드롭아웃을 추가하는 방식을 적용하였다. X, O, 빈칸의 각 데이터 개수를 확인한 결과, 빈칸 데이터가 나머지에 비해 약 2배 많아 상대적으로 X와 O에 대한 학습이 부족할 가능성이 있었다. 이를 해결하기 위해 X 1,363개, O 1,380개, 빈칸 1,334개로 데이터 개수를 조정한 후 재학습을 진행한 결과, 정확도를 80%까지 향상시킬 수 있었다.

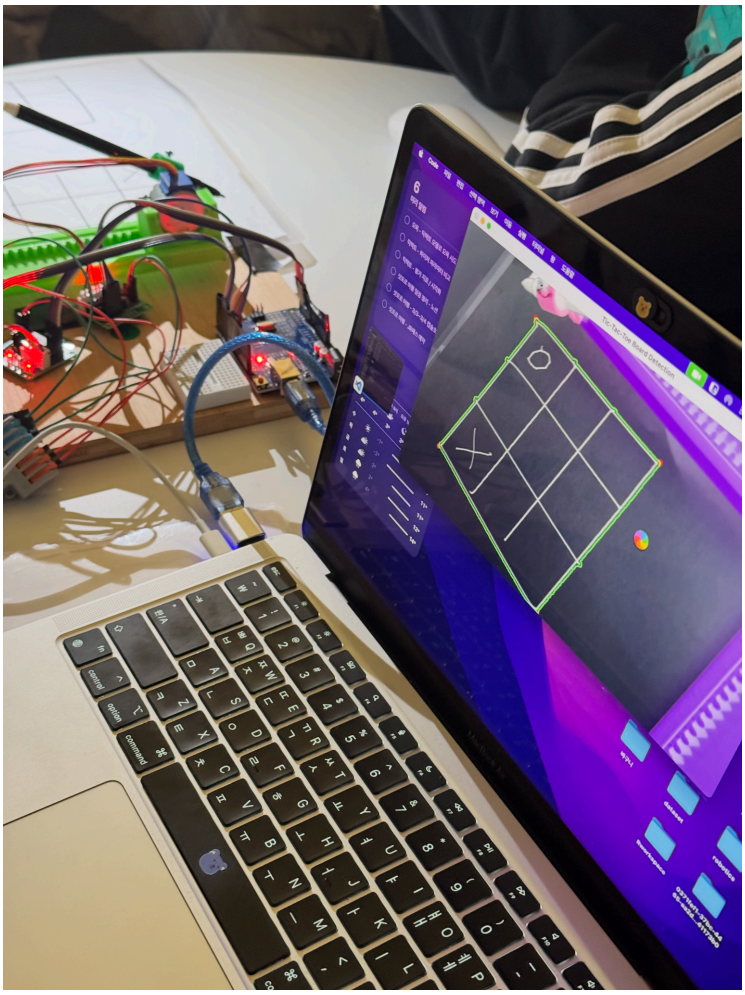
데이터 양을 늘리는 과정에서 직접 그림을 생성하는 방식은 결국 한정된 배경과 모양만을 포함할 수밖에 없다는 한계가 있었다. 이에 따라 데이터 증강 기법을 활용하는 방법을 선택하였다. 실시간으로 이미지를 촬영할 경우 다양한 변수가 발생할 수 있으므로, 불확실한 환경에서도 안정적으로 인식할 수 있도록 데이터를 변형하는 기법을 적용하였다. 이미지를 기울이는 Affine 변환, 흐리게 만드는 Blur, 안개와 같은 노이즈를 추가하는 Fog 기법을

활용하여, 60개의 데이터를 453개로 증가시켰다. 마지막으로, 드롭아웃값을 0.5에서 0.2로 줄여 데이터 손실을 최소화하는 동시에 과적합 문제를 완화하고자 하였다. 이러한 조정을 통해 최종적으로 정확도 100%를 달성할 수 있었다.

3. 이미지 인식 모습

다음 사진은 카메라로 실시간 틱택토 게임보드를 인식해 CNN 모델을 거쳐 state의 형태로 반환하는 함수를 작동해 로봇, 알파제로 에이전트와 상호작용하는 모습이다.

[그림 17] 이미지 인식 프로그램 작동 사진



V. 결론 및 시사점

본 연구에서는 틱택토 게임을 대상으로 다양한 AI 알고리즘을 적용하고, 알파제로(AlphaZero) 기반 강화학습 모델이 기존의 탐색 기법보다 더 효과적인 성능을 보일 수 있음을 실험적으로 검증하였다.

실험 결과, 알파제로 기반 AI는 기존 알고리즘인 MCTS보다 높은 승률을 기록하며, 강화학습 기법이 틱택토 환경에서 효과적으로 적용될 수 있음을 확인했다. 특히, 알파제로는 사전 지식 없이 스스로 학습하는 방식으로 작동하기 때문에, 다른 게임 환경이나 응용 분야로 확장할 가능성이 높다.

또한, 본 연구는 AI 모델이 현실에서 직접 작동할 수 있도록 CNC 모델과 연동하여 물리적 환경에서도 게임을 수행하는 실험을 진행하였다. 이를 통해 AI가 단순한 시뮬레이션을 넘어 현실 공간에서 인간과 상호작용할 수 있는 가능성을 제시했다.

향후 연구에서는 더 복잡한 게임 환경으로 확장하거나, 로보틱스와의 결합을 더욱 정교하게 설계하여 AI가 실제 환경에서 수행할 수 있는 응용 범위를 넓히는 것이 가능할 것이다. 또한, 강화학습 모델의 학습 속도를 개선하고, 학습된 AI의 의사결정을 더 효과적으로 해석하는 연구도 병행될 수 있다.

본 연구는 AI가 단순한 게임을 넘어 실제 물리적 환경에서도 효율적으로 작동할 수 있음을 보여주었으며, 강화학습 기반 AI의 실용적인 활용 가능성을 한 단계 확장하는 기초 연구로 의미를 가진다.

VI. 부록 및 참고문헌

[1] GITHUB : https://github.com/KanghwaSisters/24-2-TicTacToe.git

[2] Alpha-Zero code reference: https://github.com/Jpub/AlphaZero