Alpha Zero로 해결하는 오목

팀 AiGO

김도희, 김현서, 이승연, 이은나, 이정연, 이지민

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

초록

4축 로봇팔(end-effector : suction)을 제어해 학습한 인공지능과 인간이 현실에서 직접 상호작용하며 듀얼 에이전트 게임 '오목(Gomoku)'을 플레이하는 것을 목표로 한다. 오목 학습에 알파제로 방법론을 적용했으며, 로봇팔 제어를 위해 정방향•역방향 기구학을 사용했다. 오목 에이전트 성능 개선을 위해 데이터 증강 • 신경망 다양성 • 노이즈를 통한 추가적인 탐험을 시도했으며, 9*9 환경에서 평균 스텝수 25를 웃도는 에이전트를 개발했다.

핵심 주제어: 알파제로, 강화학습, 컴퓨터 비전, 오목, 듀얼 에이전트 게임

서론

인공지능 기술의 발전으로 다양한 게임 분야에서 인간의 수준을 뛰어넘는 성과가 보고되고 있다. 특히 2016년 알파고의 등장 이후, 강화학습을 활용한 게임 인공지능은 비약적인 발전을 이루었다. 그러나 대부분의 게임 AI 연구는 가상환경에서 이루어지며, 실제 물리적 환경에서 인간과 상호작용하는 연구는 상대적으로 제한적이다.

본 프로젝트는 강화학습으로 학습된 오목 AI가 흡착기가 부착된 4축 로봇팔을 통해 실제 환경에서 인간과 대국을 수행하는 통합 시스템을 구현하는 것을 목표로 했다. 이는 단순한 게임 AI의 구현을 넘어, 컴퓨터 비전을 통한 실시간 상태 인식과 로봇 제어를 포함하는 복합적인 과제이다.

오목은 바둑이나 체스에 비해 상대적으로 단순한 규칙을 가지고 있으나, 첫 수부터 승패가 갈리는 매우 전략적인 게임이다. 오목은 두 명의 플레이어가 번갈아가며 바둑판에 돌을 놓아 가로, 세로, 대각선 방향으로 정확히 5개의 돌을 연속으로 놓는 것을 목표로 한다. 본 프로젝트에서는 일반적인 15*15 오목판이 아닌 9*9 축소 환경을 사용했으며, 선공에게 적용되는 렌주 룰(renju rule)을 적용하지 않았다. 이러한 환경 설정은 학습의 효율성을 높이고 로봇팔의 물리적 제약을 고려한 것이다. 해당 환경에서 알파제로 방법론을 적용하여 효율적인 학습을 도모했으며, 데이터 증강, 신경망 다양성 확보, 노이즈를 통한 추가 탐험 등 다양한 기법을 통해 에이전트의 성능을 향상시켰다.

학습 결과, 본 프로젝트의 오목 AI는 제한적인 성능을 보여주었다. ResNet 구조와 4차원 상태 표현 방식이 학습 효율성 측면에서 가장 효과적이었으며, Base 모델은 일정 수준의 방어 전략을 습득했다. 인간과의 대국에서는 초반 16스텝 이내의 3목 상황은 효과적으로 방어하나, 후반부에는 대응력이 떨어졌다. 결과적으로 인간을 압도할 만한 성능에는 도달하지 못했으나, 초보자를 상대로는 승리를 거두는 등 일정 수준의 게임 이해도를 보여주었다.

실제 환경에서의 구현을 위해 컴퓨터 비전 시스템은 오목판의 상태를 실시간으로 인식하고, 6축 로봇팔은 정확한 위치에 착점을 수행해야 한다. 이를 위해 정방향 • 역방향 기구학을 활용한 로봇 제어 시스템을 구축했다.

또한 개발된 AI의 접근성을 높이기 위해 웹 플랫폼을 구현하여, 사용자가 온라인에서 직접 AI와 대국을 수행할 수 있도록 했다. 이는 물리적 환경의 제약 없이 더 많은 사용자가 개발된 AI와 상호작용할 수 있는 기회를 제공한다.

목차

그림 • 표 목차

I. 알파제로 (ALPHA ZERO)

알파제로는 MCTS를 기반으로 완전 비지도 학습을 사용해 바둑 문제를 효과적으로 해결한 최초의 방법론이다. 또한 이전 모델 알파고에 비해 훨씬 단순한 상태 정의만으로 문제를 해결했고, MCTS의 롤아웃(roll out; 현재 노드의 가치를 구하기 위해 종단 노드까지 탐험하는 과정)을 인공 신경망으로 대체해 효과적인 속도 개선 / 성능 최적화를 이루어냈다.

A. 구조 • 방법론

알파제로는 자가대국(self-play)을 통해 게임 데이터를 축적하고, 쌓인 데이터를 바탕으로 신경망을 학습시키는 사이클로 구성된다. 또한 학습 사이에 최고 성능 모델을 찾기 위한 평가를 진행한다.

1. 자가대국 (self-play)
2. 학습 (train)
3. 특정 시점마다 평가 (eval)

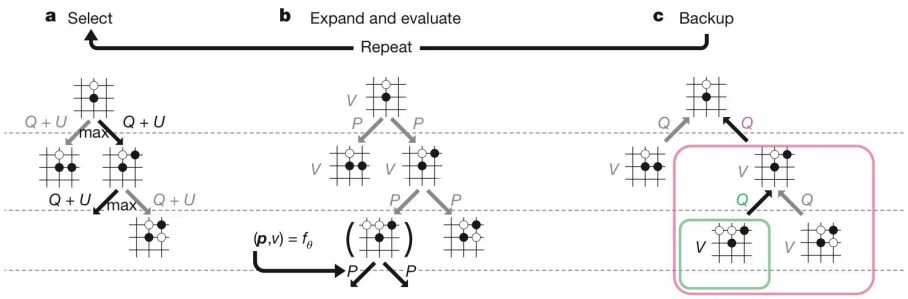
[표 1] 알파제로 학습 구조

1. 자가대국

자가대국이란, 하나의 모델이 두 플레이어 역할을 모두 수행하며 각 플레이어의 이익 극대화 행동을 선택하는 게임 진행 방식이다. 알파제로가 자가대국을 통해 축적하는 게임 데이터는 (상태 s_t - 정책 π_t - 가치 z_t) 쌍이다. 각 상태의 정책과 가치는 각각 MCTS 방법론과 자가 대국의 종단 상태에서 역전파로 구해진다.

1-1. 몬테카를로 트리 탐색 (MCTS)

MCTS는 시뮬레이션을 통해 현재 상태에서 최적의 정책을 찾는다. 알파제로의 MCTS는 선택 - 확장 및 평가 - 역전파 과정을 수행하는 플레이아웃(playout)을 반복해 얻은 자식 노드의 방문 빈도로 정책을 만든다.



[그림 1] MCTS playout 구조

플레이아웃은 자식 노드가 없는 리프 노드나 종단 노드에 다다를 때까지 자식 노드 **선택**을 반복한다. 이때 자식 노드는 Q(s_t, a) + U(s_t, a) 값을 기준으로 선택된다. U값은 탐험을 다루는 수식으로 PUCT 알고리즘으로 계산한다.

$$U(s_t, a) = C_{PUCT} * P(s, a) * \frac{\sqrt{\Sigma_b N(s,b)}}{1+N(s,a)}$$

- C_{PUCT} : 탐험의 정도를 지정하는 상수

초반에는 높은 사전 확률을 지니지만 방문 빈도가 낮았던 노드를 우선으로 탐색하며, 후반에는 높은 행동 가치를 가진 노드를 탐색해 탐험을 조절한다.

종단 노드가 아니지만 자식 노드가 없다면 **확장**하며 플레이 아웃을 완료한다. 기존 MCTS는 리프 노드의 가치를 구하기 위해 롤아웃을 진행한다. 하지만 롤아웃은 종단노드까지의 시뮬레이션이 필요하기 때문에 막대한 연산량을 필요로 한다. 알파제로는 롤아웃의 역할을 인공 신경망으로 대체해 문제를 해결한다. 인공신경망은 현재 상태에서 정책과 가치를 반환하는데 정책의 확률값은 각 자식 노드의 사전확률로, 가치는 현재 노드의 가치로 사용된다.

노드가 확장되거나, 종단 상태에 다다르면 **역전파**가 수행된다. 역전파는 자식노드의 가치를 부모노드로 보내는 과정으로, 부모는 자식노드들의 가치 평균으로 자신의 행동가치를 업데이트한다.

$$Q(s,a) = W(s,a) / N(s,a)$$

수 차례의 플레이아웃 수행 이후, 축적된 자식 노드들의 방문 수를 이용해 **볼츠만 분포를 따르는 정책**을 만든다. 볼츠만 분포는 타우가 1에 가까울수록 방문 빈도에 비례한 분포를, 0에 가까울수록 원핫인코딩 형식의 분포를 반환한다. 자가대국에서는 온도를 1로 유지하다 임계값 이후 0으로 낮추어 정책 분포를 형성한다. 이때 부족한 탐험을 보완하기 위해 Dirichlet 분포를 따르는 노이즈를 정책에 추가한다.

$$P(s,a) = (1-\varepsilon) * p_a + \varepsilon * \eta_a, \text{ where } \eta \sim \text{Dir}(0.03) \text{ and } \varepsilon = 0.25$$

1-2. 가치 역산

게임이 종료되면 플레이어의 승패를 기준으로 가치(z)를 역산한다. 이긴 플레이어라면 모든 상태의 가치는 1을 갖고, 반대로 패배한 플레이어라면 -1을 갖는다. 오목은 승패가 나지 않고 모든 행동 공간이 채워지는 무승부가 존재하는데, 이 가치를 0으로 설정했다.

2. 학습

알파제로에서 정책을 만드는 것은 MCTS고, 인공 신경망은 MCTS의 효율과 정확성을 높이기 위한 보조수단이다. 따라서 인공신경망은 모든 상태에서 MCTS의 정책과 그에 따른 가치의 기댓값을 근사하는 것을 목표로 한다. 학습에 사용되는 데이터 셋은 Queue로 누적되게 구현했으며 랜덤으로 샘플링해 시간 종속성을 없앴다.

2-1. 신경망 입출력 구조

듀얼 신경망으로, 현재 state를 입력값으로 받아 현재 state의 정책과 가치를 출력한다.

$$(\boldsymbol{p}, v) = f_\theta(s)$$

- $\boldsymbol{p}$: 현재 상태에서 모든 행동에 대한 정책 $\{p(s,a) \mid a \in A\}$
  - $A = \{$ total actions $\}$
- v : 현재 상태에서 가치 기댓값 $\{v \in [-1, 0, 1]\}$
- θ : 신경망의 파라미터
- f : 인공 신경망 함수
- s : 현재 상태

2-2. 신경망 타겟 값

정책 p는 MCTS가 만들어내는 정책 $\boldsymbol{\pi}$를, 가치 v는 현 상태의 결과를 나타내는 z를 목표값으로 학습한다.

2-3. 손실 함수 정의

p의 경우 $\boldsymbol{\pi}$ 사이의 유사도를 높이는 크로스 엔트로피를, v의 경우 z 사이의 오차를 줄이는 mse loss를 사용해 학습한다.

$$\text{loss} = (z-v)^2 - \boldsymbol{\pi}^T \log p + c \parallel \theta \parallel^2$$

- c : L2 정규화 파라미터로 과적합 방지에 이용한다.

3. **평가**

평가는 최고 성능 신경망과 현재 학습된 신경망끼리 대국하는 방식으로 진행된다. 오목은 선을 잡는 것이 유리하기에, 선공 후공에 영향을 받지 않도록 번갈아가며 게임을 진행한다. 평가는 학습과 달리 순수한 신경망의 성능을 테스트하는 것이 목적이기 때문에 탐험 온도를 0으로 설정하며, 노이즈 또한 제거한다. 만약 최신 모델의 승률이 55%를 넘기면 최고 성능 모델을 현재 모델로 갱신한다.

B. 탐험

알파제로의 탐험은 **PUCT 알고리즘, 볼츠만 분포의 온도 파라미터, 그리고 노이즈**를 통해 조절된다. 먼저, MCTS 노드 확장에 사용되는 PUCT 알고리즘의 탐험 상수 C는 탐색 방식에 직접적인 영향을 미친다. C 값이 크면 다양한 노드를 탐색하는 경향이 강해지고, 값이 작으면 자주 방문한 노드를 우선적으로 선택하게 된다. 또한, 볼츠만 분포를 기반으로 한 정책 결정에서는 온도 파라미터를 조정하여 탐험의 정도를 조절한다. 학습 초반에는 온도를 1로 설정해 충분한 탐험이 이루어지도록 하며, 일정한 임계 시점 이후에는 온도를 0에 가까운 작은 값으로 낮춰 결정론적인 선택을 강화한다. 그러나 온도를 지나치게 낮추면 정책이 원-핫 인코딩 형태로 수렴하여 이후 학습 과정에 부정적인 영향을 미칠 수 있다. 임계 시점 이후 탐험 부족 문제를 해결하기 위해 Dirichlet 분포를 따르는 노이즈를 정책에 추가하여 추가적인 탐색을 유지한다. 이러한 방식으로 알파제로는 탐색과 활용 사이의 균형을 조절하며 최적의 학습을 수행한다.

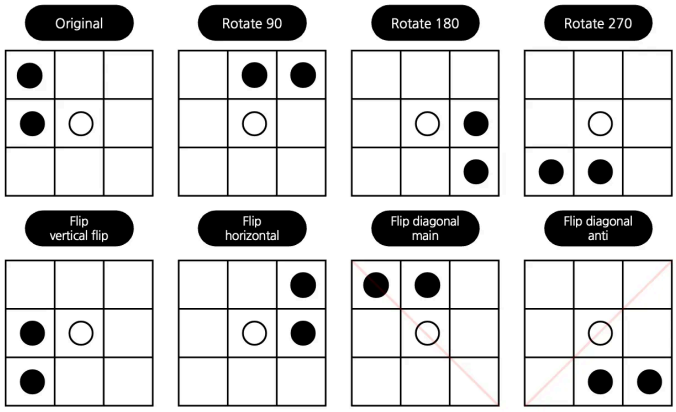
C. 핵심 구현부

1. **합법 행동** (legal action)

합법 행동이란 규칙을 위반하지 않으며 실행할 수 있는 행동을 의미한다. 오목에서 합법 행동은 아직 방문하지 않은 수다. 게임 환경을 구현할 때는 이미 방문한 행동을 아예 선택할 수 없도록 환경 단에서 제한할 것인지, 혹은 제약을 두지 않되 불가능한 행동임을 나타내는 지표를 반환할 것인지를 결정해야 한다. 알파제로는 MCTS를 이용하기 때문에 이미 방문한 행동에 제약을 두는 것이 좋다 판단했다. MCTS에서 환경 단에서 불가능한 행동을 차단하면 탐색 공간이 줄어들어 정책의 수렴이 빨라지고, 더 명확한 정책을 생성해 최적의 수를 더 효과적으로 찾을 수 있기 때문이다.

2. **대칭 불변 상태**

오목은 한 상태를 8가지 상황으로 바라볼 수 있다. 원본, 90도 회전, 180도 회전, 270도 회전, 상하반전, 좌우반전, 대각반전, 반대각반전을 해도 동일한 상태이기 때문이다. 이런 오목 상태 대칭 불변성은 상태 공간을 1/8로 획기적으로 줄일 수 있다는 장점이 있다.



[그림 2] 대칭 불변 상태 집합 8종

상태의 대칭 불변성을 **데이터 증강**과 MCTS **확장** 두 가지 방식으로 적용했다. 먼저 데이터 증강은 하나의 상태를 가능한 8개의 상태로 만들어 8배의 데이터를 만들어내는 방식이다. 한 자가대국마다 만들어지는 데이터의 양이 많기 때문에 충분한 히스토리 메모리 용량을 필요로 한다. MCTS 확장은 현재 상태를 랜덤으로 변형한 후, 그 상태를 기준으로 만들어진 정책과 가치를 확장에 사용해 상태 공간을 축소한다.

$$(d_i(\boldsymbol{p}), v) = f_\theta(d_i(s_L))$$

- d_i : 이미지 변환 공식

3. MCTS

MCTS는 여러 번의 플레이아웃을 통해 현재 상태에서 정책을 만들어내기 때문에, 막대한 양의 이터레이션을 수반한다. 따라서 알파제로 알고리즘 속도 향상을 위해서는 MCTS 속도 향상이 필수적이다. 이를 위해 MCTS 알고리즘을 C++로 구현한 뒤, 파이썬과 연동했지만 파이썬으로 구현된 상태 클래스와 MCTS 알고리즘 사이 오버헤드가 발생해 속도가 오히려 느려지는 문제가 존재했다.

II. 학습

A. 상태 (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 객체는 이 state에서 자신의 수와 상대의 수, 다음 행동, 게임 보드(state)의 크기, 전체 행동과 전체 행동의 개수, 이 게임의 승리 조건 정보를 가지고 있다. 이 상태에서 가능한 행동 리스트를 반환하는 메서드, 이 상태의 플레이어가 첫 번째 플레이어인지 반환하는 메서드, 이 상태의 플레이어 기준 게임 결과를 반환하는 메서드 등의 상태 정보를 표현하는 메서드가 존재한다. 그리고 게임을 진행하면서 다음 State 객체를 생성하는 next 메서드와 부가적으로 상태를 시각화 하는 메서드가 있다.

학습 과정에서 이 클래스 객체를 여러 번 반복하여 호출하고 내부 함수 또한 자주 사용되기 때문에, 코드 실행 속도를 고려해 구현하고자 하였다. 리스트 대신 Numpy 배열을 사용하고, Numpy 내장 함수를 활용해 코드를 최적화하였다. 특히 게임 결과를 매 턴마다 확인해야 하므로, 속도 개선을 위해 최대한 적은 반복으로 게임 결과를 정확하게 확인할 수 있도록 구현하였다. 마지막 행동과 연결된 부분에서만 게임 종료 조건을 만족할 가능성이 있다는 점을 이용해, 매번 전체 게임보드를 탐색하는 것이 아닌 마지막 행동이 존재하는 행, 열, 대각, 반대각 줄을 확인하도록 하였다. 또한 전체에서 나의 돌이 승리 조건 개수 미만인 경우 승리 조건 확인을 건너뛰고, 위와 같은 방식으로 승리 조건을 확인할 때도 한 줄에 돌이 승리 조건 개수 미만이라면 건너뛰는 방식으로 게임 종료 확인을 최대한 간결하게 하였다.

state를 신경망 입력으로 넣을 수 있는 모양으로 변환하는 함수는 State 클래스 외부에 `select_state` 함수로 구현하였다. 나의 state, 상대의 state를 기본으로 하고, 하이퍼 파라미터로 첫 번째 플레이어 여부와 직전 대국 state를 학습할 상태에 넣을 것인지 설정할 수 있도록 구현하였다.

B. 신경망

정책과 가치를 예측하는 단일 신경망을 구축할 때 기본적인 합성곱 신경망(CNN) 구조와 알파제로 논문에서 채택한 잔차 네트워크(Residual Network) 구조를 참고하여 구현하였다.

1. Basic CNN

Common layers
A convolution of 32 filters of kernel size 3×3 with stride 1, and padding of 1
Batch normalization
A rectifier nonlinearity
A convolution of 64 filters of kernel size 3×3 with stride 1, and padding of 1

Batch normalization
A rectifier nonlinearity
A convolution of 128 filters of kernel size 3×3 with stride 1, and padding of 1
Batch normalization
A rectifier nonlinearity

Policy head
A convolution of 4 filters of kernel size 1×1 with stride 1
Batch normalization
A rectifier nonlinearity
Flattening
A fully connected linear layer that outputs a vector of size 81
Softmax activation

Value head
A convolution of 2 filters of kernel size 1×1 with stride 1
Batch normalization
A rectifier nonlinearity
Flattening
A fully connected linear layer that outputs a vector of size 64
A rectifier nonlinearity
A fully connected linear layer that outputs a vector of size 1
Tanh activation

2. Residual Network

입력 상태가 초기 합성곱 층(convolutional block)을 통과한 후 10개의 연속된 잔차 블록들(residual block)을 통과하며 특징을 추출한다. 전역 평균 풀링(adaptive average pooling)으로 특징 맵을 축소하고 벡터로 평탄화하는 과정을 거쳐 정책 및 가치 헤드를 통해 최종 출력을 생성하는 구조이다. 정책 헤드의 출력 값은 9*9 바둑판에서 가능한 81개의 행동에 대한 정책이고, 소프트맥스 함수를 거쳐 확률 벡터로 변환된다. 그리고 가치 헤드의 출력 값은 현재 상태에서의 승리 가능성을 평가하는 가치 기댓값으로, 하이퍼볼릭 탄젠트 함수를 거치는 스칼라 값이다.

Convolutional block
A convolution of 128 filters of kernel size 3×3 with stride 1, and padding of 1
Batch normalization
A rectifier nonlinearity

Residual block
A convolution of 128 filters of kernel size 3×3 with stride 1, and padding of 1
Batch normalization
A rectifier nonlinearity
A convolution of 128 filters of kernel size 3×3 with stride 1, and padding of 1
Batch normalization
A skip connection that adds the input to the block
A rectifier nonlinearity

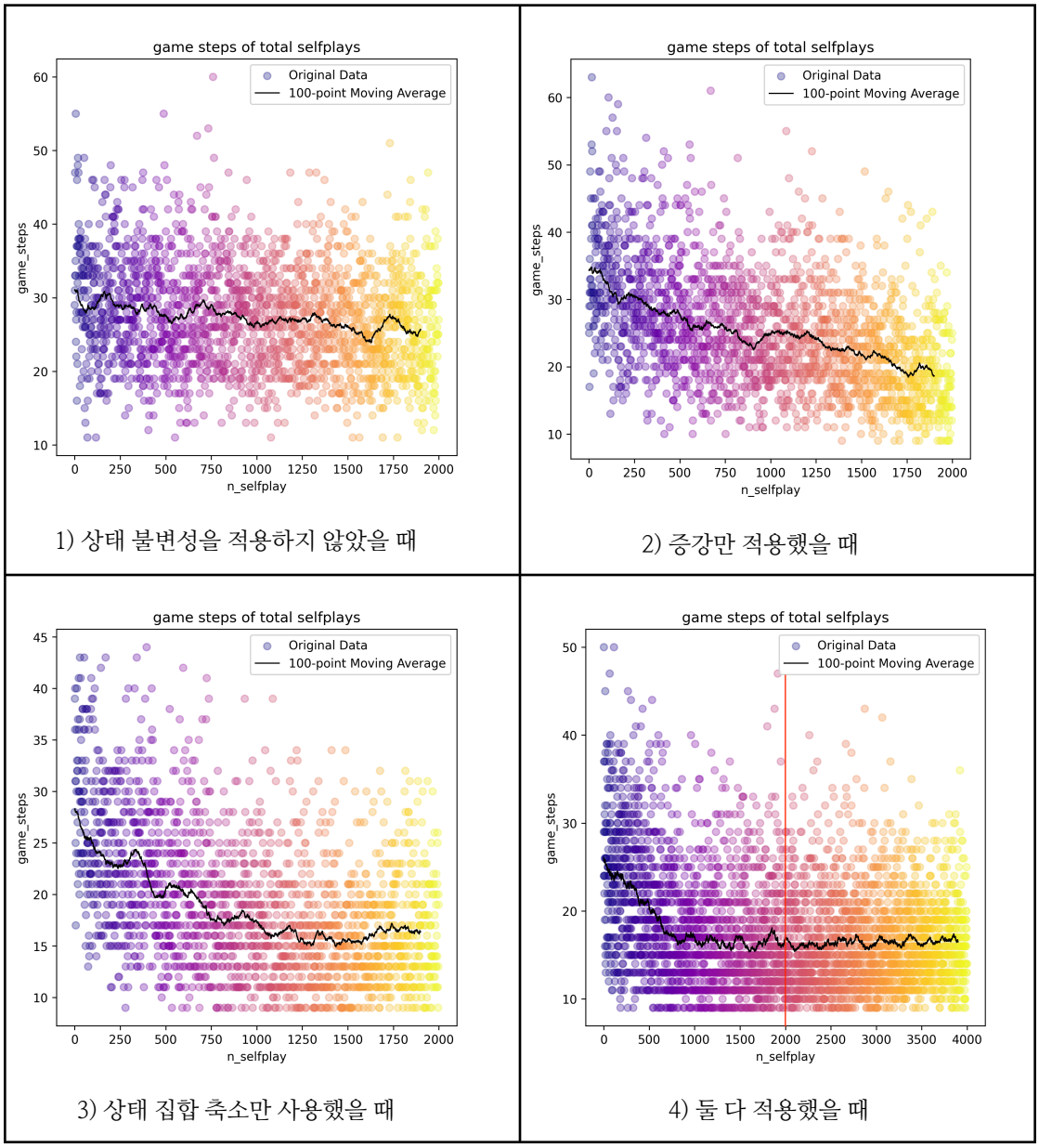
C. 최종 성능 • 하이퍼 파라미터

성능은 다양한 지표의 영향을 받는다. 아래 항목은 학습 도중 유의미한 차이를 만들었거나, 차이가 있으리라 생각해 실험해 본 것들을 정리한 것이다.

- 상태 불변성
- 신경망 / 상태 차이
- 핵심 파라미터 변경

1. 상태 불변성

오목 상태의 불변성은 에이전트의 성능 향상에 가장 큰 영향을 준 특성이다. 불변 상태는 증강과 MCTS 확장에서 랜덤 선택으로 상태 집합 축소 두 가지로 구현했으며, 각 각 학습 양상에서 차이가 있었다.



1) 상태 불변성을 적용하지 않았을 때

2) 증강만 적용했을 때

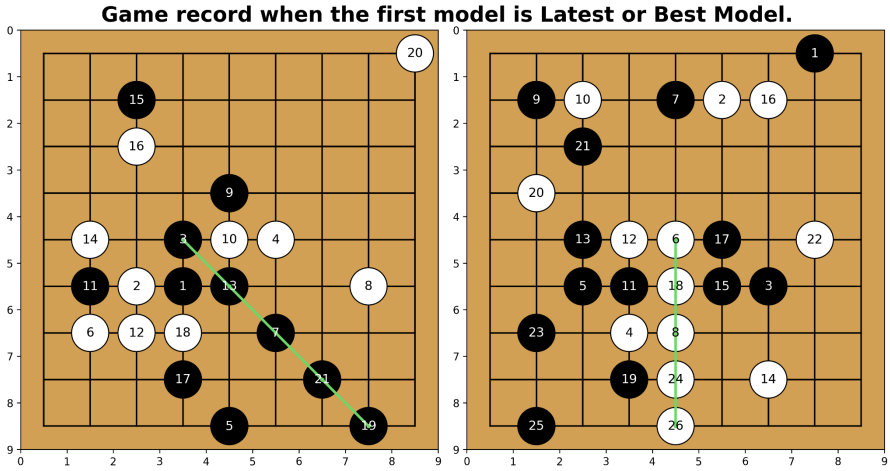
3) 상태 집합 축소만 사용했을 때

4) 둘 다 적용했을 때

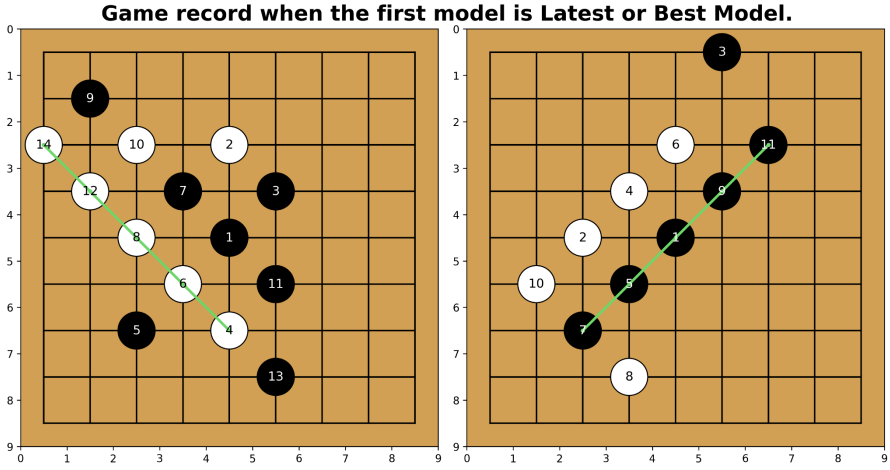
[그림 3] 2000 자가대국 시점에서 타입 별 스텝 수 그래프

스텝 수를 지표로 볼 때, 상태 불변성은 초기 학습에 영향을 미친다. 학습 초기에는 스텝

수가 줄어들며 학습이 진행되는데, 이를 정책 • 평가 단계의 기보(그림 4)를 통해 승리조건을 배우는 과정으로 해석했다. 상태 집합 축소는 단순 승리 조건 인지가 증강보다 빨랐다. 증강 방법론은 상태 집합 축소보다는 더 느리게 스텝 수가 줄었지만, 다양한 스텝 수 분포를 보이며, 보다 충분한 탐색이 이루어지고 있다고 해석할 수 있다.



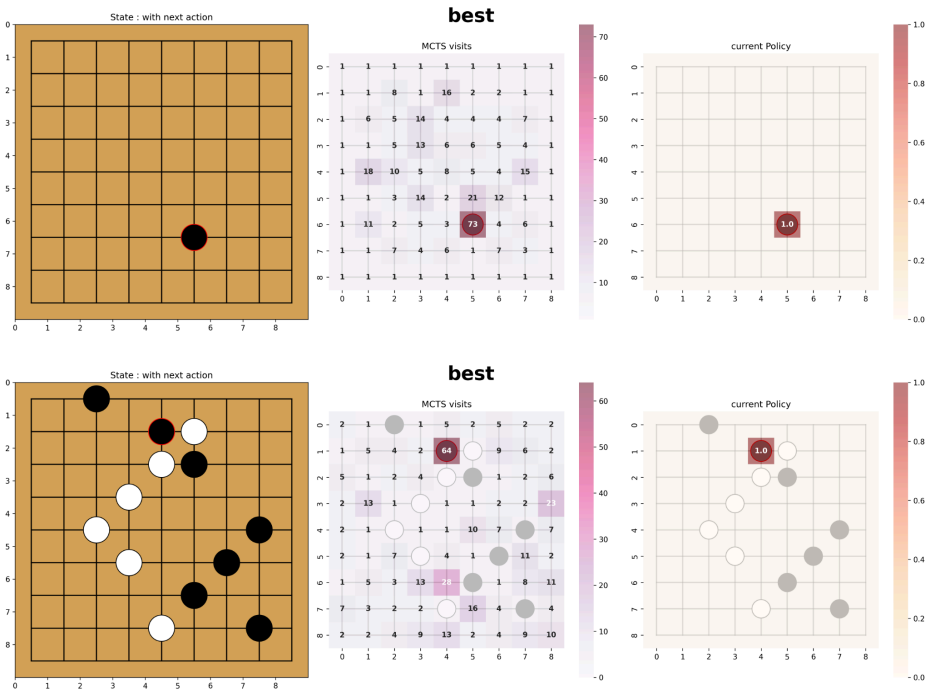
학습 초반 평가



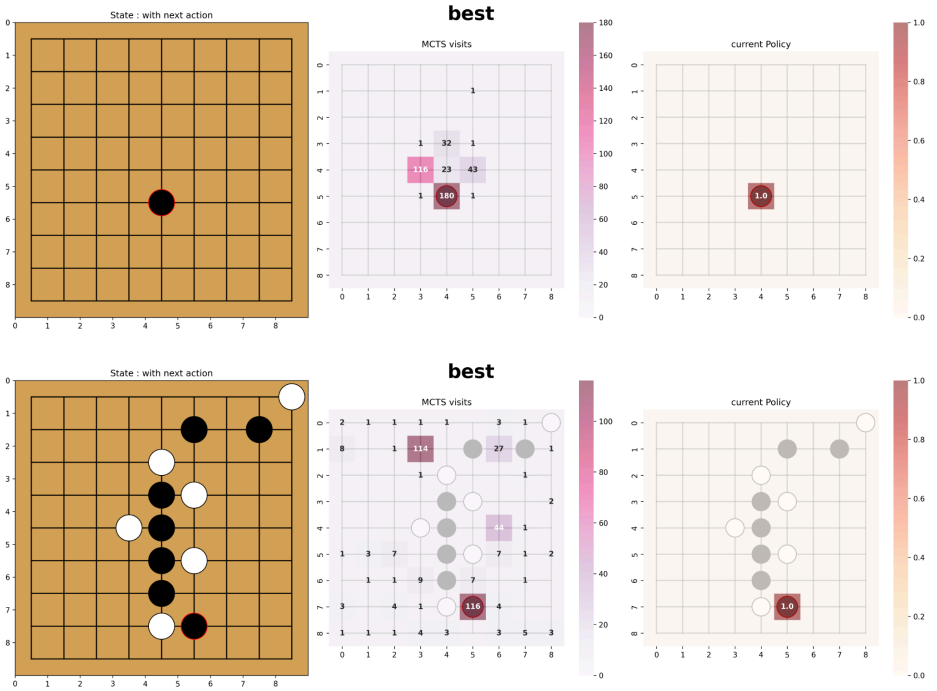
스텝 수가 유의미하게 떨어진 이후 평가

[그림 4] 평가 시점 기보

이 두 가지 방법론 중 **증강**을 학습에 더 많이 이용했다. 상태 집합 축소 방법론은 초반 수렴 속도는 빨랐으나, 이후 학습에서 방어를 잘 학습하지 못했고 학습이 진행될수록 정책이 수렴하지 못하는 모습을 보였다. 그에 반해 데이터 증강은 몇 번의 자가 대국 이후에도 정책 분포가 보다 상식적인 형태를 하고 있었으며, 정책이 상태 집합 축소 방법론에 비해 수렴되어가는 형태를 하고 있었다.



상태 집합 축소 방법론을 사용했을 때 6000 자가 대국 후 평가



데이터 증강 방법론을 사용했을 때 6000 자가 대국 후 평가

[그림 5] 6000 자가 대국 후 평가

2. 신경망 / 상태 차이

데이터 증강 + ResNet10을 기준 신경망으로 두고 100번씩 대국을 진행했다. 오목은 선을 잡는가가 게임의 승패에 영향을 주는 게임임으로, 번갈아가며 선을 잡게 만들었다. 기준이 되는 지표는 스텝 수의 평균 및 중앙값, 선공을 잡았을 때 승률이다.

- 4차원 상태 : [현재 내 판 – 현재 상대 판 – 이전 상대 행동(1개) – 선공인지]
- 5차원 상태 : [현재 내 판 – 현재 상대 판 – 이전 내 판 – 이전 상대 판 – 선공인지]

후공 | 선공	(Base) State 4dim + ResNet x 10	State 5dim + ResNet x 10	State 4dim + Basic CNN Net
(Base) State 4dim + ResNet x 10	0.58 (21.74 / 20)	0.88 (16.89 / 14)	0.55 (16.31/ 16)
State 5dim + ResNet x 10	0.97 (16.89 / 14)	0.68 (11.15 / 11)	0.78 (13.37 / 12)
State 4dim + Basic CNN Net	0.65 (16.31/ 16)	0.68 (13.37 / 12)	0.82 (17.81 / 16)

- 구조 : 선공 시점 승률 (스텝 수의 평균 / 스텝 수의 중앙값)
- 6천 번의 자가 대국 이후의 결과

[표 2] 승률-스텝 수 매트릭스

전반적으로 선을 잡았을 때의 승률이 후공을 잡을 때보다 월등히 높다. 평가 때는 정책을 만들 때 온도를 죽이기 때문에(0으로 수행), 여러 번 반복해도 동일한 사건이 중첩되어 나타날 가능성이 높다. 이는 승률에 영향을 미치므로, 추가적인 지표로 스텝 수를 추가했다.

스텝 수와 승률 지표에서 가장 많은 스텝이 진행되는 것은 Base 모델 사이의 자가대국이다. 이를 통해 Base 모델이 어느 정도 방어를 학습했다는 것을 추론할 수 있다. 그에 반해 5차원 상태로 학습한 모델은 자가 대국시 평균 11스텝에 머물러 방어를 거의 학습하지 못한데다, 승률이 그에 반해 높지 않은 것을 보아 선공이 확실한 승리를 가져가지 못함으로, 학습이 효과적이지 못했다고 해석할 수 있다. 기본 CNN 신경망을 사용한 경우는 자가 대국시 선공의 승률이 높지만, 스텝 수가 Base보다 적다.

이를 통해 기본 신경망보다는 ResNet이 동일한 시점에서 더 많은 방어방법을 학습했으며, 상태를 표현할 때는 하나의 행동이 원-핫으로 들어가 이전 상태를 표현하는 4차원 형태가 더욱 효과적임을 알 수 있다.

3. 핵심 파라미터

플레이 아웃 횟수는 적어도 전체 행동 공간의 n배는 되어야 종단까지의 탐험이 가능하리라 보았다. 따라서 약 5배인 **400번의 플레이 아웃**을 수행했고, 이보다 적은 수로 학습을 돌렸을 때는 유의미한 정책이 잘 발생하지 않았다.

9*9 보드에서 수행하는 오목의 경우, 게임 초기에 3목 방어 단계까지 오기 위해서는 **최소 4000번의 자가 대국**이 필요하다는 경험적인 값을 얻었다.

이외 깃에 올려둔 모델 학습에 사용한 하이퍼 파라미터들은 다음과 같다.

# state info # 	
STATE_DIM = 4
ALLOW_TRANSPOSE = False
DATA_AGUMENTATION = True
 	
# count # 	
TOTAL_SELFPLAY = 12000
EVAL_SELFPLAY = 20
N_PLAYOUT = 400
TRAIN_EPOCHS = 1000
MEM_SIZE = 80000
 	
# nn # 	
N_KERNEL = 128
N_RESIDUAL_BLOCK = 10
BATCH_SIZE = 512
	# learning rate #
	LEARNING_RATE = 0.0002
	LEARN_DECAY = 0.5
	
	L2 = 0.0001
	
	# temperture : Boltzman #
	TRAIN_TEMPERATURE = 1.0
	EVAL_TEMPERATURE = 0
	
	# selfplay : exploration #
	C_PUCT = 5.0
	EXPLORE_REGULATION = 6
	
	# frequency #
	TRAIN_FREQUENCY = 1
	EVAL_FREQUENCY = 50
	VISUALIZATION_FREQUENCY = 200
	LEARN_FREQUENCY = 6000

- 자세한 정보는 GitHub의 config.py에 기술되어 있다.

4. 최종 성능 평가

본 프로젝트는 기간 내 인간을 압도할 만큼 유의미한 성능의 모델을 개발하지는 못했다. 인간과 대국했을 때는 평균 25 스텝 게임이 진행되지만, 낮은 승률을 보인다. 초반(16 스텝 안)에 발생하는 3목은 높은 확률로 방어하나, 이후 발생하는 3목 상황에는 유연하게 대처하지 못한다. 그럼에도 대국했을 때 상식적인 수를 두는 경우가 많고, 오목에 익숙하지 않은 사람을 상대로는 꽤 높은 승률을 보인다.

[그림 6] 실제 인간과 대국한 보드, 흰 : 사람 검 : 알파제로

III. CV

CV는 오목 대국을 위해 실제 바둑판의 이미지를 입력받아 바둑판(19*19)에서 목표하는 9*9 바둑판의 상태를 반환한다. 이를 위해 OpenCV 라이브러리를 사용하여 카메라로 입력받은 영상을 처리하고, 돌의 위치를 효과적으로 추출해내는 과정이 필요하다. 다음은 CV 기반 코드에 대한 설명, 기능 및 성능 그리고 핵심구현부 및 트러블슈팅에 관한 내용이다.

A. 기능 • 성능

다음과 같은 기능 구현을 목표한다.

1. 카메라로부터 실시간 또는 스틸 이미지를 입력받아 바둑판 영역을 인식
2. 바둑판 위의 돌(흑, 백)을 판별하고 좌표계로 변환
3. 9*9 array의 형태로 반환해 활용할 수 있도록 제공

1. 바둑판 인식

카메라로부터 실시간 프레임 또는 캡쳐한 이미지를 수신하고 보드 영역을 찾는 기능으로 외곽선 검출, 모서리 검출, 투시 변환 등의 기법을 사용하여 실제 바둑판만을 잘라낸다.

2. 돌 위치 검출

보드 영역 내에서 흑/백 돌을 구분하여, 해당 좌표를 (row, col)형태로 변환한다. Color thresholding, Hough Circle Transform 등의 다양한 이미지 처리 기법을 사용하여 이를 구분하고 검출한다.

성능과 관련한 부분에 관해서는 일반 웹캠 30fps 정도가 가능하지만 해상도가 높아질수록 외곽 검출 및 돌 인식에 필요한 연산량이 증가한다는 점에서 한계가 발생할 여지가 있다. 정확하게 돌의 위치를 반환하지 못하는 경우가 많아, 실험은 이미지 상태에서 진행하였다. 인식의 정확도에 관해서는 조명, 돌 색깔 대비, 보드 디자인 등에 따라 달라질 수 있으며, 해당 상황에 맞게 파라미터를 조정해야 한다.

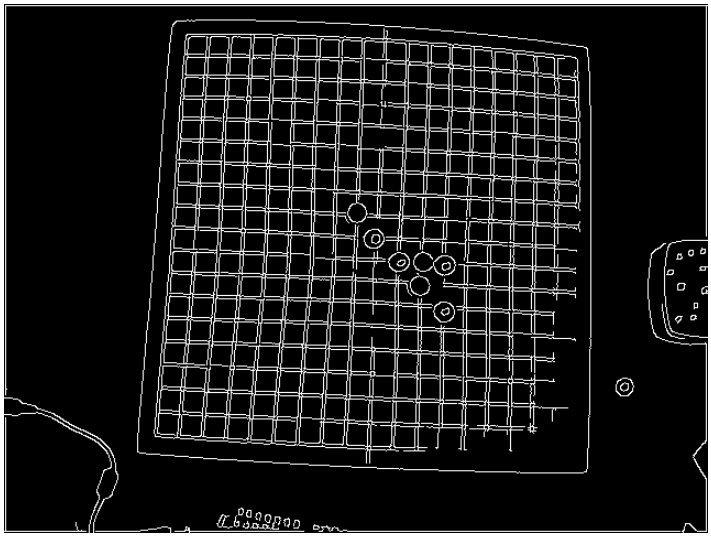
B. 핵심 구현부

1. 촬영 및 전처리

take_photo함수에서는 웹캠으로부터 이미지를 캡처하여 파일로 저장한 뒤, 최종 파일명을 반환한다. SPACE를 누르면 이미지를 캡처하고, ESC를 누르면 종료한다.

보드 이미지를 그대로 사용하면 배경이나 주변 사물이 포함되어 돌 인식 정확도가 떨어진다. 따라서 원근변환(Perspective Transform)이나 직교좌표=〉극좌표=〉변환=〉직교좌표 등의 변환을 통해 보드의 볼록 왜곡이나 오목왜곡을 보정해야한다. apply_lens_distortion 함수에서는 입력 이미지에 오목·볼록 렌즈 효과를 적용해 저장한다. exp 매개변수를 조정해 왜곡 정도를 설정한다. 이때 1미만이면 오목렌즈, 1이상은 볼록렌즈의 효과를 갖는다. scale 매개변수를 조정해 왜곡이 적용될 화면상의 범위(0~1)를 조정한다. 왜곡 후 결과 이미지를 파일로 저장한다.

crop_board_from_image 함수는 입력이미지에서 바둑판이나 사각형 영역을 찾아 크롭하고, 결과 이미지를 파일로 저장한다. 자르고자 하는 영역을 찾기위해 Canny 엣지 검출, 외곽선 검색, 사각형 추출과정을 따른다. Canny 엣지 검출은 이미지에서 밝기 변화 지점인 엣지를 찾아서, 윤곽선 검출이 더 정확하도록 도와준다. 우선 RGB에서 Grayscale로 바꾼 뒤 진행하고 가우시안 블러 등으로 작은 잡음을 완화해, 에지 검출이 노이즈에 덜 민감하도록 만든다. 각 픽셀에서의 엣지 강도와 방향을 구한다. 엣지 방향을 따라 인접 픽셀과 비교해, 국소적으로 가장 강도가 큰 픽셀만 남긴다. 이중 임계값을 사용하여 하한값, 상한값 두 가지 임계값을 두고, 픽셀이 엣지인지를 판별한다. 엣지 강도가 상한 임계값보다 크면, 해당 픽셀은 확실한 엣지로 분류되고, 하한 임계값 이하이면 엣지가 아닌것으로 간주된다. 하한값과 상한값 사이에 있는 경우에는 엣지가 아닌것으로 판단되는 것이 아니라 이웃 픽셀(주변 4방향이나 8방향) 중에 확실한 엣지가 하나라도 있으면 최종적으로 엣지로 인정한다. 이에 대한 결과로 엣지 부분은 흰색, 배경은 0인 바이너리 이미지가 생성된다.



[그림 7] 엣지와 배경을 인식한 이미지 상태

Canny로 얻은 이진화 이미지를 바탕으로, 연결된 엣지 픽셀을 연속된 곡선 또는 닫힌 도형으로 파악하기 위해 외곽선 검색을 실시한다. OpenCV함수 중 cv2.findContours를 사용한다. 입력은 Canny로 얻은 이진화 이미지, mode, method이다. 가장 외곽 윤곽만 나타내거나 모든 계층 구조를 추적하는 등의 mode가 있고 method는 윤곽점을 어느정도 간략화해서 저장할지를 결정한다. 검출된 각각의 윤관선을 리스트 형태로 반환하고 윤곽은 다각형 또는 곡선 형태로 구성된 좌표들의 집합이다.

사각형 추출과정을 통해 여러 윤곽선 중 사각형을 찾는다. 윤곽선 점이 많은 경우, cv2.approxPolyDP 함수로 다각형을 근사한다. 이 함수의 epsilon은 오차 허용정도이고 True이면 폐곡선으로 간주한다. 사각형인지 확정된 윤곽선에 대해서는, cv2.boundingRect(approx)함수를 사용해 (x,y,w,h)형태로 경계 상자를 얻을 수 있다. 이때, 영역의 넓이가 너무 작거나, 종횡비가 특정 기준과 크게 다르면 오검출일 가능성이 있다는 점에서 추가 필터링을 적용하기도 한다.

2. 이미지 분석

analyze_omok_board 함수의 입력으로는 바둑판 이미지 경로(image_path), 격자간격크기(cell_size), 바둑판 격자 수(grid_size), 시각화(show_grid), 돌 검출 정확도 향상 옵션(improve_detection)이다. 이에 대한 출력으로는 바둑판 상태배열(9*9 의 array), 흑돌의 좌표 리스트, 백돌의 좌표리스트이다. Canny 엣지 검출이나 모폴로지 연산은 전처리 시와 같은 기능을 한다. cv2.HoughLinesP로 선분을 검출하지만 이 코드에서 직선 추출은 주로 시각화용이다. 바둑한의 가정된 중앙을 기준으로, 격자 점들의 좌표를 계산해 놓는다. cv2.HoughCircles로 바둑돌을 찾고 찾아낸 돌의 좌표와 바둑한의 격자 점들을 비교해 가장 가까운 격자에 매핑한다. 매핑된 돌의 밝기 정보를 보고 흑돌, 백돌을 구분해 상태 배열을

구성하고, state와 흑돌 좌표, 백돌 좌표를 반환한다. show_grid=True 인 경우, 각 격자점과 검출된 돌을 시각화해 화면에 표시한다.

cv2.HoughCircles를 사용할 때, 대표적으로 조정하는 네가지 파라미터 param1, param2, minRadius, maxRadius가 있다. param1은 원 검출 과정에서 내부적으로 활용되는 엣지 검출 알고리즘의 상한 임계값으로, 보통 Canny는 하한 임계값과 상한 임계값 두 개를 사용한다. param1이 상한 임계값으로 설정되고, 하나한 임계값은 보통 그 절반갑으로 자동 설정된다. param1을 낮게 설정하면 엣지가 너무 많이 검출되고, 너무 높은 경우 약한 엣지를 놓쳐 원 검출이 누락된다. param2는 최종 원으로 인정되기 위한 조건을 결정하는 허프 누적 상의 임계값으로 허프 원 변환에서 투표로 누적된 원 후보마다 점수가 매겨지는데, 이 점수가 param2를 넘으면 실제 원으로 인식된다. 값이 낮을수록 원 후보 판정이 느슨해 지고 값이 높을수록 원 검출이 엄격해진다. minRadius, maxRadius는 검출할 원의 최소 반지름과 최대 반지름을 의미한다. 검출하고자 하는 원의 최소·최대 크기를 지정해 너무 작은 대상이나 너무 큰 겹침 등에서 오는 오검출을 줄이는 데 쓰인다.

C. *트러블슈팅*

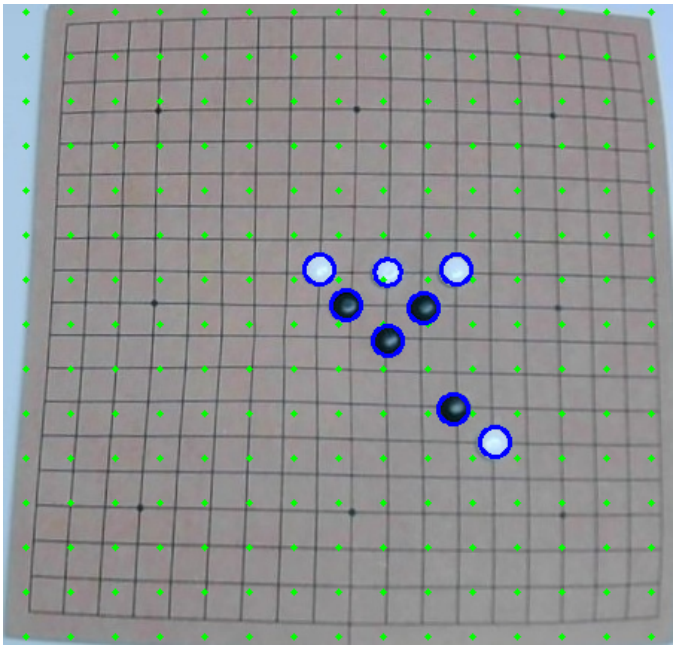
1. 조명문제

조명이 강하거나 불균일하여 이미지 분석이 어려워지는 경우가 발생한다. 특히과도한 빛으로 인해, 각도에 따라 바둑알·테이블 표면이 빛을 심하게 반사하는 경우나 그림자가 생기는 경우가 발생한다. 흰돌임에도 그림자가 강하게 생겨 검은돌로 인식하여 오류가 생긴다. 또한 그림자로 인해 바둑돌의 윤곽이 뭉개진다.

위의 문제를 해결하기 위해, 코드에서는 cv2.GaussianBlur로 노이즈를 줄이고 있고, 윤곽선 정제를 위해 엣지를 뚜렷하게 한다. 미세한 노이즈나 빛 번짐을 완화해서 엣지 검출 안정성을 높여주고, improve_detection=True 옵션을 사용하여 모폴로지 연산을 통해 엣지나 돌 형상이 그림자나 반사로 인해 끊기거나 희미해진 경우 엣지를 보정해줄 수 있다. 흰돌이 그림자 때문에 어둡게 찍혀 흑돌로 오인식 되는 경우를 방지하기 위해서는 밝기 임계값을 조정한다.

2. 일정 간격으로 나누어 판단하는 방식

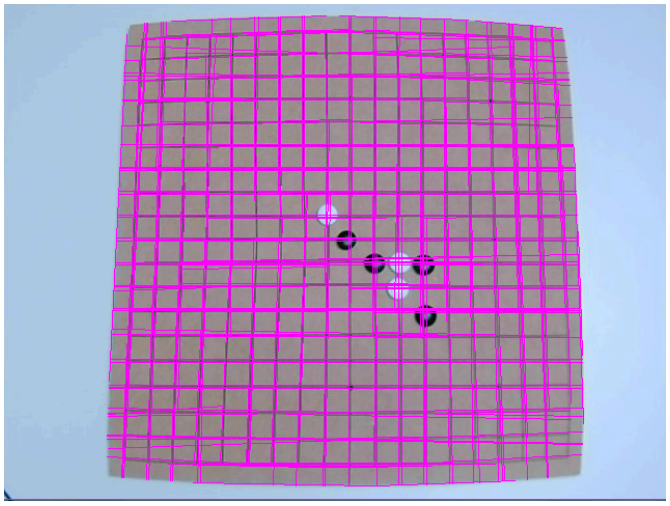
이미지 중심(center_x, center_y)을 기준으로, cell_size씩 일정 간격을 벌려 격자점을 찍는 방식을 사용한다. 이 방식은 바둑판이 정확히 가운데에 놓여있고, 각 칸의 크기가 일정하다는 전제를 충족해야만 제대로 작동한다. 만약 실제 이미지에서 바둑판이 조금만 치우쳐져 있거나 원근 왜곡으로 칸 간격이 일정하지 않거나 크게 회전된 상태라면 격자점이 실제 바둑판과 어긋나서 돌 검출 결과가 부정확해진다.



[그림 8] 일정하지 않은 격자와 돌을 인식한 이미지 상태

바둑판이 회전되거나 기울어진 상태에서 일정 간격을 가정해 격자를 생성하면 돌 인식에 오차가 발생할 수 있다. 이를 해결하기 위해 먼저 바둑판을 크롭하여 퍼스펙티브 보정하는 방법이 있다. cv2.findContours나 cv2.HoughLinesP 등을 사용해 바둑판 테두리 윤곽(사각형)을 검출하고, 검출된 네 모서리 좌표로 cv2.getPerspectiveTransform과 cv2.warpPerspective를 적용해 이미지를 정면에 가깝게 보정한다. 이렇게 회전·기울어짐 없이 수평·수직 상태가 된 이미지는 cell_size 방식으로 격자를 찍을 때도 훨씬 정확하다.

다른 접근으로, 바둑판의 가로선과 세로선을 각각 cv2.HoughLinesP로 찾아 교차점을 구하는 방법이 있다. 각 교차점을 (가로줄 i, 세로줄 j) 순서대로 정렬하면 실제 격자점 좌표를 얻을 수 있으며, 이미지가 회전되거나 기울어져 있어도 중심점이나 cell_size를 임의로 가정할 필요가 없다. 이러한 동적 격자 추출 방식은 기울기나 회전에 무관하게 돌 위치를 인식할 수 있다. 다만 조명과 노이즈 때문에 일부 선이 끊겨 누락될 수 있으므로, 블러나 모폴로지 연산 등을 적절히 적용하여 에지를 보완하고, minLineLength나 maxLineGap 파라미터를 조정해 선 검출이 더 안정적으로 이뤄지도록 해야 한다.

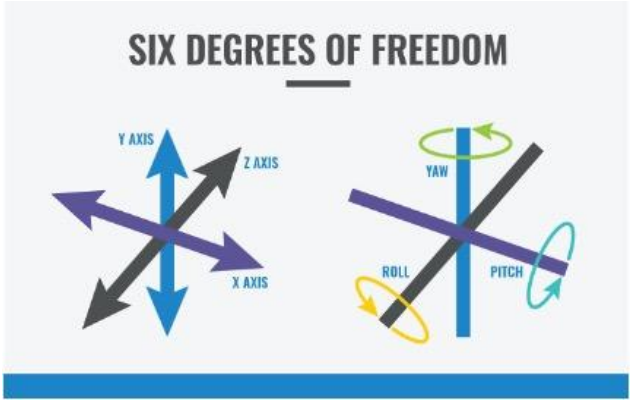


[그림 9] 격자 검출로 상태를 추출한 이미지

IV. 로보틱스

A. 구조 및 동작 원리

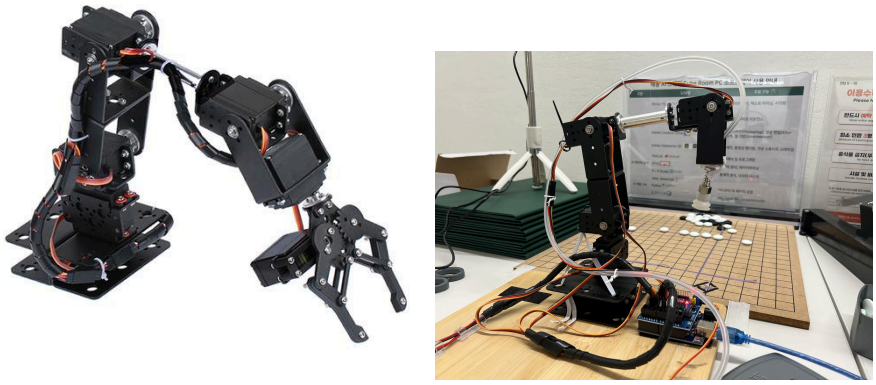
본 프로젝트는 알파제로 기반의 오목 대국 시스템 구축을 목표로 한다. 초기에는 6축 자유도(6-DOF) 로봇팔을 활용하는 방안을 고려했지만, 무게 부담과 가동 범위 제한 등의 문제로 인해 최종적으로 4축 제어 방식과 흡착기를 적용한 로봇으로 재설계했다.



[그림 10] 6 DOF

6축 자유도는 6개의 관절로 이루어져 있으며, 선형 변위(X, Y, Z축 이동)를 수행하는 3축의 관절과 회전(Roll, Pitch, Yaw)을 수행하는 3축의 관절로 구성된다. 선형 변위는 특정 축을 따라 직선적으로 이동하는 동작으로 X축(Surge)은 전후 이동, Y축(Sway)은 좌우 이동, Z축(Heave)은 상하 이동을 담당한다. 회전 변위는 특정 축을 중심으로 회전하는 동작을 포함하며, X축(Roll)은 기울기 조정, Y축(Pitch)은 앞뒤 기울기, Z축(Yaw)은 방향 전환을 수행한다. 로봇팔에서 6축에서 이동은 Surge : Shoulder, Sway : Elbow, Heave : Wrist로 대응되며, 회전은 Roll : Waist, Pitch : Shoulder Pitch, Yaw : Wrist Yaw로 대응된다.

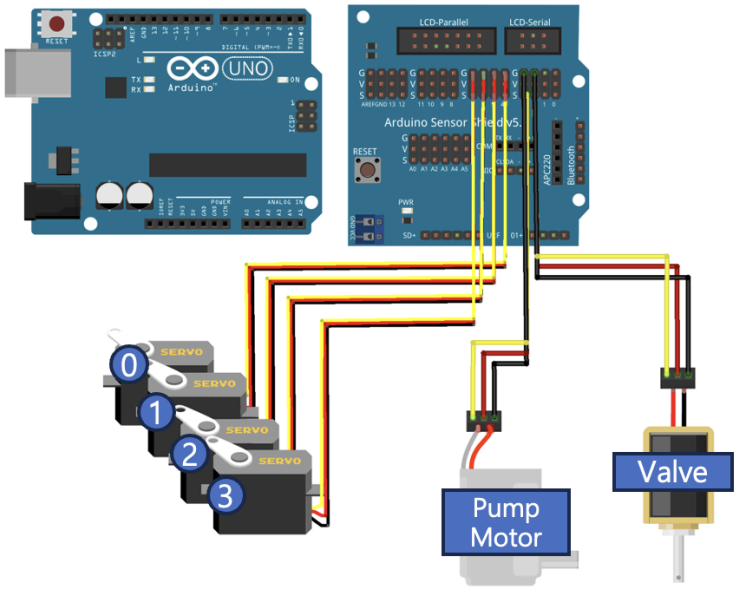
4축 제어 방식과 흡착기를 적용한 로봇은 6축 로봇의 회전을 최소한으로 하며, 제어를 단순화시킨 형태다. 6축의 중 3 선형 변위(Shoulder, Elbow, Wrist), 하나의 회전(Waist)만을 사용했으며 end-effector에 흡착기를 부착했다.



[그림 11] 6축 로봇팔 구조 | 4축 및 흡착기 로봇팔 구조

비용과 팔을 멀리 뻗었을 때의 하중 지지 능력을 고려하여, 경량성과 내구성을 갖춘 알루미늄 재질의 로봇팔 모델을 선정하였으며 구조적 이해와 제어의 용이성을 높이기 위해 각 모터별로 하단부터 0~3의 숫자를 지정하였다.

0번 모터는 허리 회전축으로, 로봇팔 전체의 방향을 조절하는 역할을 하며 수평 회전을 담당한다. 1번 모터는 어깨 회전축으로, 로봇팔의 메인 기둥을 앞뒤로 움직여 높이를 조절하는 기능을 수행한다. 2번 모터는 팔꿈치 회전축으로, 팔의 중간 부분을 접고 펼치는 동작을 통해 바둑돌을 가까이 놓거나 멀리 놓을 수 있는 전진 및 후진이 가능하도록 한다. 마지막으로 3번 모터는 손목 회전축으로, 바둑돌을 놓고 집을 수 있는 흡착기(gripper)의 정밀한 위치 조정을 담당하여 다양한 작업에 활용될 수 있도록 한다.



[그림 12] Arduino 기반 4축 서보모터 및 흡착기 제어 회로도

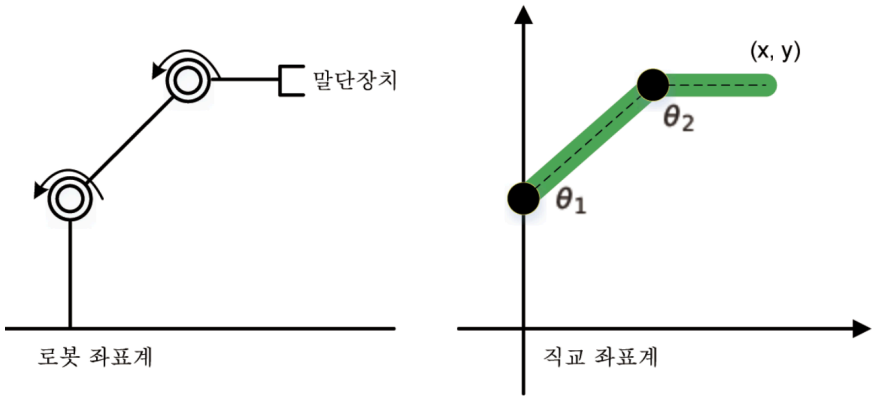
[그림 12]의 회로도는 Arduino Uno를 메인 컨트롤러로 사용하고 Arduino Sensor Shield V5.0을 통해 4개의 서보모터(MG996R)를 제어하며, 흡착기 시스템을 포함한 로봇팔 동작을 구현하는 구조로 설계되었다. Arduino Uno는 전체 시스템의 동작을 조정하며, 센서쉴드와 연결되어 서보모터에 필요한 PWM 신호를 제공하며. 쉴드는 다수의 서보모터를 쉽게 연결할 수 있도록 설계된 확장 보드로, 추가적인 PWM 컨트롤러 없이도 서보모터를 효율적으로 관리할 수 있다. 각 서보모터는 로봇팔의 주요 관절을 담당하며, [그림 13]과 같이 연결되어 있다. 흡착기 구성요소로인 석션 펌프(suction pump)와 밸브(valve)가 포함되어 있으며, Arduino의 디지털 신호를 통해 이들의 동작을 제어한다.

Arduino Sensor Shield V5.0	Valve	Pump	Servo Motor
2	○		
3		○	
4			0
5			1
6			2
7			3

[표 3] Arduino Sensor Shield V5.0의 핀 연결 구성

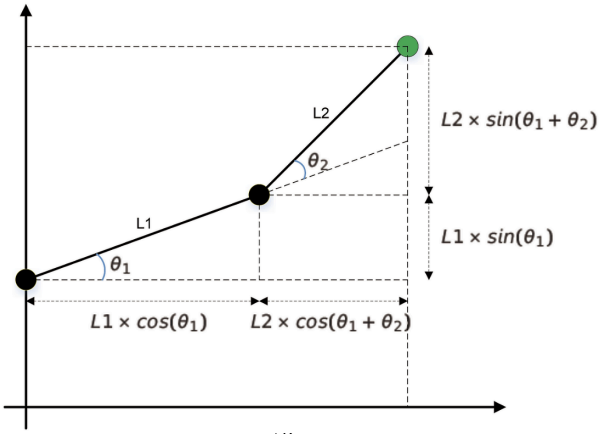
1. 본체 동작 원리 (정방향 – 역방향 기구학)

로봇팔의 동작은 정방향 기구학(Forward Kinematics, FK)과 역방향 기구학(Inverse Kinematics, IK)을 기반으로 하며, 각 관절의 움직임을 제어하고 최종적으로 원하는 위치에 바둑알을 배치하는 과정으로 이루어진다.



[그림 13] 관절의 회전에 대한 로봇 좌표계(왼쪽), 수학적 표현을 위한 직교 좌표계(오른쪽)

정방향 기구학(FK)은 각 관절의 회전각(θ)이 주어졌을 때, 이를 통해 로봇팔 말단(End-Effector)의 위치와 자세를 계산하는 방법이다. 즉, 각 축(조인트)의 회전값이 결정되면, 로봇팔이 어디에 위치할지를 예측할 수 있다. 로봇팔은 다수의 링크(Link)와 조인트(Joint)로 이루어져 있으며, 각 조인트는 특정한 회전각(θ)을 가지며 변환 행렬을 이용하여 위치를 결정한다. DH 파라미터(Denavit-Hartenberg Parameters)를 적용하여 로봇팔의 FK를 표현하면, 각 조인트의 회전 변환은 다음과 같은 4×4 변환 행렬(Transformation Matrix)을 사용하여 계산된다.

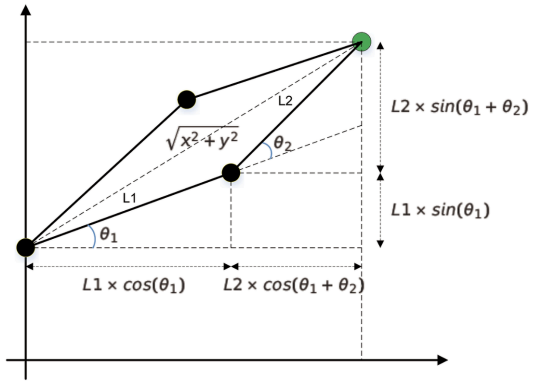


[그림 14] 2자유도 로봇팔의 좌표 변환 및 위치 계산

여기서, θ_i 는 해당 조인트의 회전각, d_i 는조인트 간 거리, a_i 는 링크의 길이, 그리고 α_i 는 축 사이의 회전각을 의미한다. 각 조인트의 변환 행렬을 곱하면 최종적으로 로봇팔 끝단의 위치 (x, y, z 좌표) 와 자세(Orientation)를 구할 수 있다.

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

역방향 기구학(IK)은 목표로 하는 끝단 위치가 주어졌을 때, 이를 도달하기 위한 각 조인트의 회전각을 계산하는 방법이다. 즉, "바둑알을 특정 좌표에 배치하고 싶다"라는 명령을 주었을 때, IK를 사용하여 로봇팔의 각 관절을 어떤 각도로 설정해야 하는지를 계산한다.



[그림 15] 2자유도의 역기구학 해석

K는 일반적으로 직접적인 해법(Closed-form solution)이 어려운 경우가 많아, 수치적 해법(Numerical Methods)이 활용된다. 이때, Jacobian 행렬(Jacobian Matrix)을 활용하여 목표 위치까지의 변화를 계산하는 방식이 사용된다. Jacobian 행렬을 이용한 속도 관계식은 다음과 같다. $\dot{x}$은 끝단의 속도 벡터 (위치 변화량)을 나타내며, $J(\theta)$ 는 Jacobian 행렬 (조인트 위치에 따른 변환 행렬), 그리고 $\dot{\theta}$ 는 조인트의 각속도 벡터를 나타낸다.

$$\dot{x} = J(\theta)\dot{\theta}$$

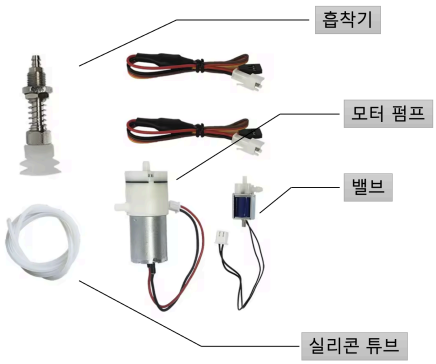
IK 문제를 해결하기 위해 Jacobian의 역행렬 또는 의사역행렬(Pseudo-Inverse)을 사용하여 역연산을 수행하며, 목표 좌표를 기준으로 조인트 회전각을 계산한다. 하지만

Jacobian의 역행렬이 존재하지 않는 특이점(Singularity) 문제가 발생할 수 있으므로, 보통 반복적인 수치 해법(Iterative Methods)이 적용된다. 목표 위치(예: 바둑알을 놓을 오목판의 좌표)가 주어지면, IK를 이용하여 조인트 각도를 자동 계산할 수 있다. 인간이 직접 각도를 조정할 필요 없이, 원하는 위치를 입력하면 로봇팔이 자동으로 조인트를 조정하여 목표 지점으로 이동할 수 있다.

$$\dot{\theta} = J^{-1}(\theta)\dot{x}$$

2. 흡착기 구성요소 및 동작 원리

흡착 시스템은 공압 펌프, 밸브, 흡착 패드, 실리콘 튜브로 구성되며, 압력을 조절해 물건을 집고 놓는 방식으로 동작한다. 흡착 패드가 바둑알 위에 위치하면, 아두이노가 공압 펌프를 작동시키고 밸브를 닫아 내부 공기를 제거한다. 공압 펌프는 내부의 모터를 회전시키고 원심력에 의해 압력이 감소하여 흡착 패드 내부가 저압(진공) 상태가 된다. 이 과정에서 흡착 패드 내부의 압력이 주변보다 낮아지고, 높은 압력에서 낮은 압력으로 공기가 이동하게 되면서 바둑알이 흡착 패드에 밀착된다. 입력된 위치에 도달하면 밸브를 다시 개방시켜 공기가 튜브를 통해 패드로 공급되도록 한다. 이로 인해 압력 차이가 사라지면서 내부 압력이 평형을 이루게 되고, 바둑알이 자연스럽게 떨어진다.



[그림 16] 흡착 시스템 구성요소

B. 코드 구현

신호를 받는 부분은 c++ 기반으로, 제어하는 부분은 python으로 구현했다. 파이썬으로 구현된 제어 코드는 시리얼 통신으로 아두이노에게 신호를 전송한다. 로봇팔은 알파제로가 예측한 수를 바둑판 위에 올려주는 목적으로 구현되었고, 다음과 같은 프로토콜을 따른다.

1. 선공, 후공을 정한다.
2. CV에서 받은 현 상태와 이전 상태를 비교해 행해진 행동을 추적한다.

3. 알파제로가 반환한 행동 좌표를 로봇팔을 이용해 바둑판에 업데이트한다.
4. 2-3 과정을 게임 종료까지 반복한다.

한계 및 보완점

- **알파제로**

  현재 모델은 3개가 이어졌을 때 방어하는 행동을 보이긴 하나, 모든 상황에서 이를 인지하지는 못한다. 게임 중간 중간 전혀 상관없는 사이드에 수를 두는 경우로 인하여 에이전트의 승률이 낮아지는 양상이 포착되었다. 학습이 되고 있는가를 판단하기까지 소요되는 시간이 커, 정작 성능 향상을 위한 시도를 충분히 진행하지 못해 아쉬움이 남는다. 또한 MCTS 최적화를 시도했음에도 효과적이지 않아 유의미한 속도 향상을 이뤄내지 못했다. MCTS 병렬화, cython, 멀티 스레딩 기법을 적용한 MCTS 속도 개선을 시도하고, 성능 향상을 위해 학습을 지속할 계획이다.

- CV

  각도나 빛의 위치에 따라 인식하는 것의 편차가 크다. 이에 대한 편차를 줄이기 위해 변환, 가우시안 노이즈를 추가하는 등의 보조적인 도구를 사용했지만 모든 상황에 대응하지는 못한다. 그리고 바둑판만 자르고 변환하여 특정 위치에서의 바둑돌을 탐지하는 방식을 시도했으나 바둑판을 수평이나 수직으로 정확하게 자르지 못해 탐지한 바둑돌의 위치가 맞지 않는 문제가 발생했다. 이 방식을 보완하기 위해 바둑판의 가로선과 세로선을 탐지하고 격자점 위에 바둑돌 정보를 반환하도록 설계했으나, 렌즈의 왜곡을 수정하는 과정과 선들을 군집화하는 과정에서 사람이 파라미터를 직접 수정해야한다는 문제가 있다. 두 가지의 아이디어 모두 바둑판을 상태로 변환할 때마다 파라미터를 직접 수정해야 한다는 한계가 존재한다.

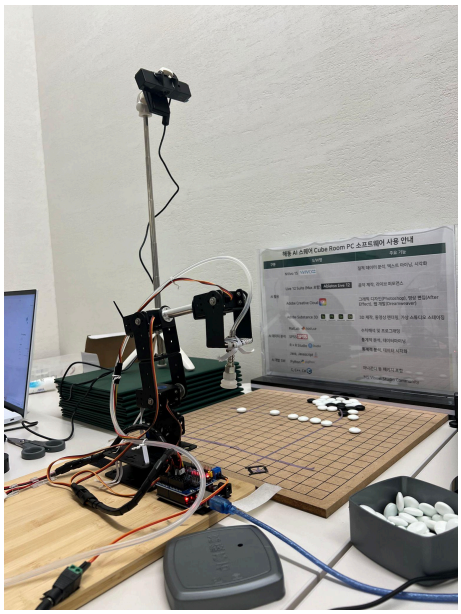
- **로보틱스**

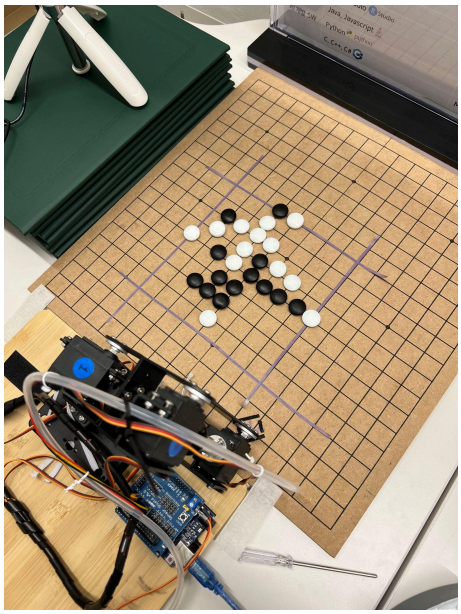
  본 프로젝트에서 적용한 정방향 기구학(FK)과 역방향 기구학(IK) 기반의 로봇 팔 제어 방식은 이론적으로 정확한 위치 제어를 가능하게 하지만, 실제 하드웨어의 한계로 인해 오차가 발생했다. 로봇 팔의 체결 오차 및 기어 유격으로 인해 회전각을 정확하게 추출하기 어려웠기 때문이다. 이를 해결하기 위해, 1차적으로 기구학을 통해 값을 계산한 후, 실제 움직임과 괴리가 큰 값들을 하드코딩으로 보정하여 보다 현실적인 값을 도출했다. 그러나 하드코딩 방식은 환경이 조금만 달라져도 원하는 결과를 얻기 어려우며, 다양한 상황에 유연하게 대처할 수 없다는 한계가 존재한다. 또한, 현재 시스템은 미리 지정된 위치에 사용자가 직접 바둑돌을 배치해야 한다는 불편함이 있다. 이를 개선하기 위해, 3축 자이로 센서를 활용하여 x, y, z 위치를 실시간으로 측정하고, 컴퓨터 비전(CV)을 활용해 설정한 값과 목표 위치 간의 오차를 조절하는 방안을 추가로 연구할 예정이다. 더불어, 바둑돌을 집는 과정에서 자유도를 높일 수 있는 추가적인 토이 프로젝트를 진행하여, 보다 범용적인 바둑용 로봇 팔 제어를 구현할 계획이다.

부록

실물 사진

- 실제 작동 영상은 깃허브에서 확인할 수 있다.



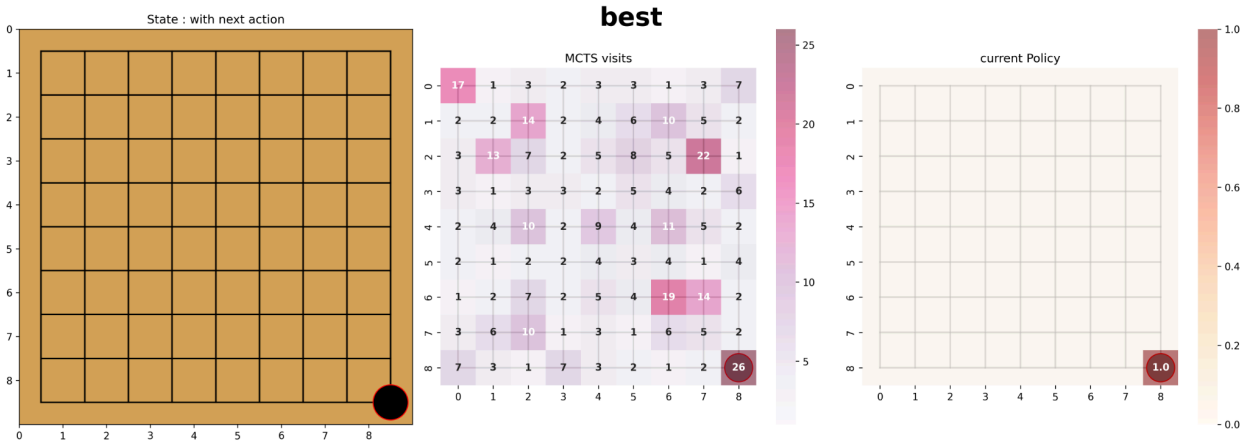


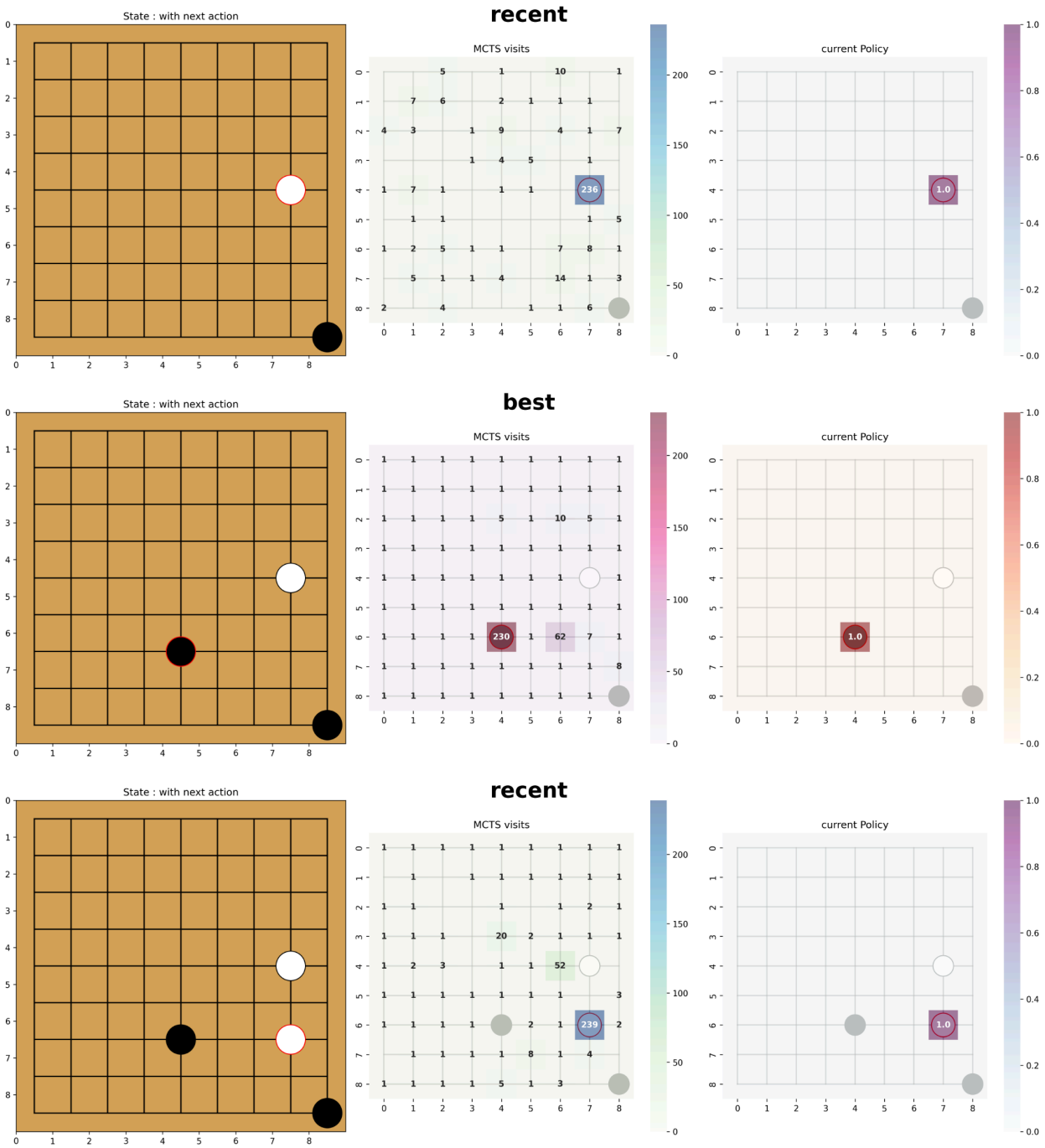
오른쪽의 사진은 CV-로봇-알파제로를 연동해 진행한 대국 결과다.

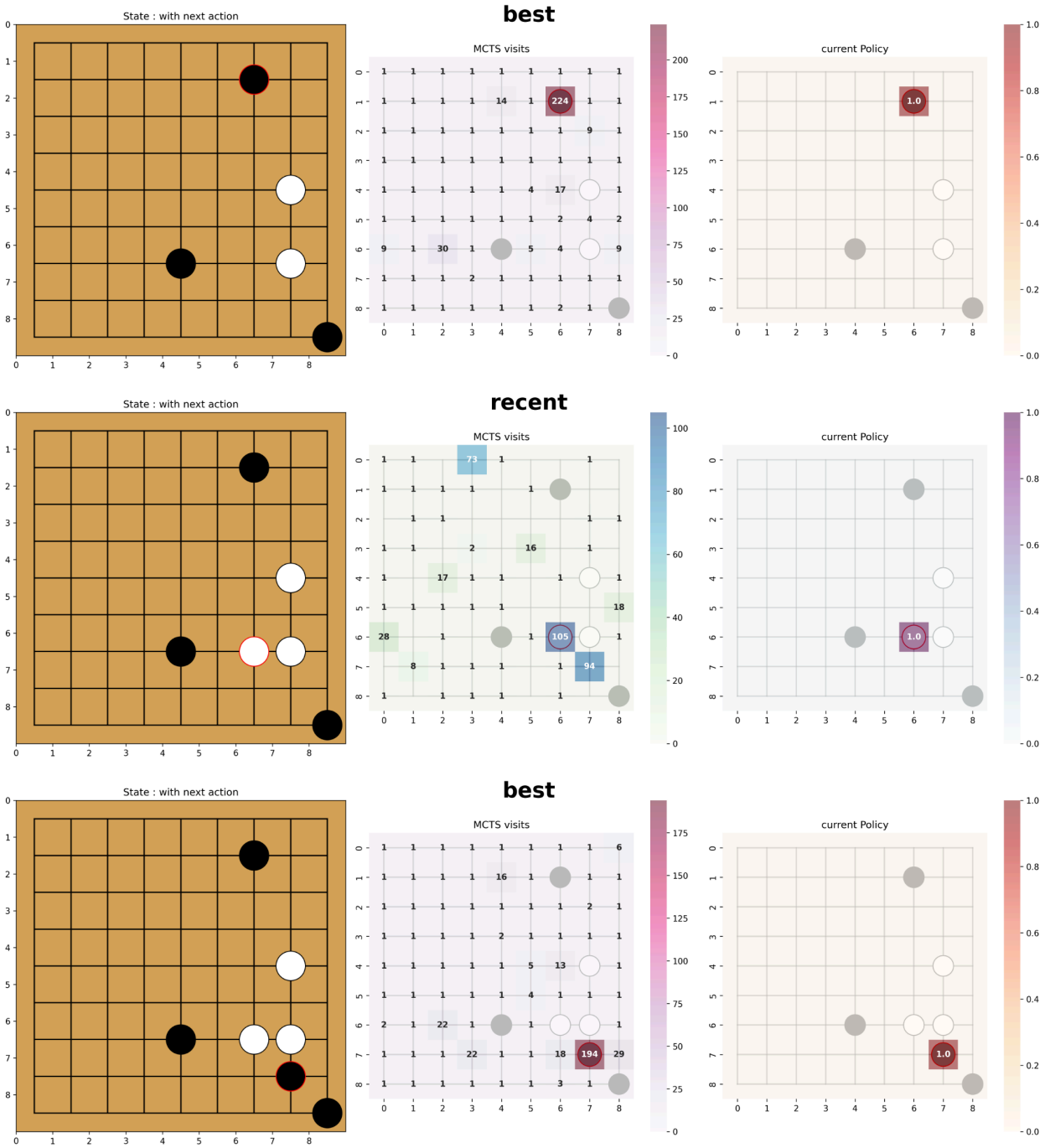
흑돌이 알파제로, 백돌이 인간이다.

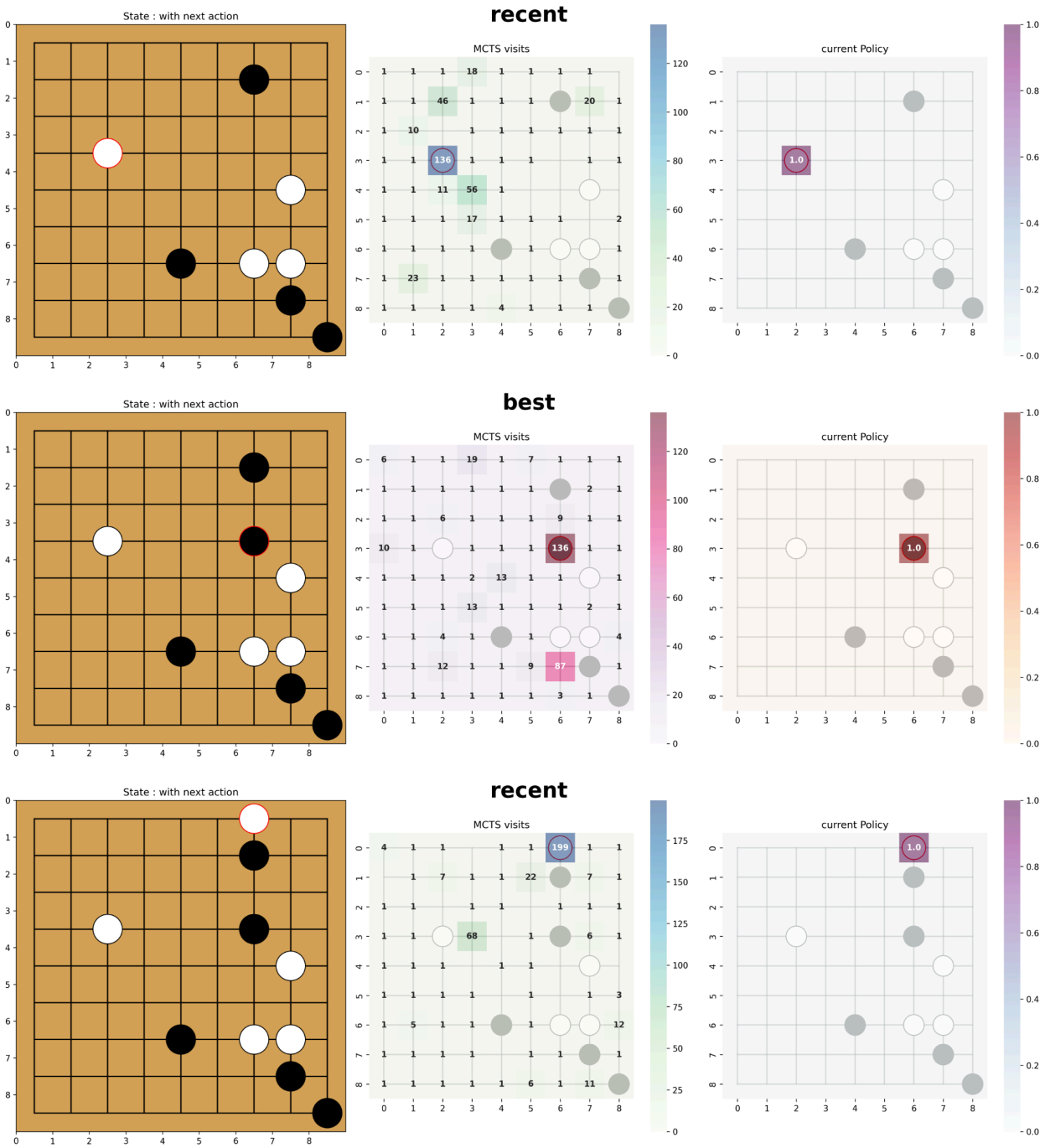
학습 과정 사진

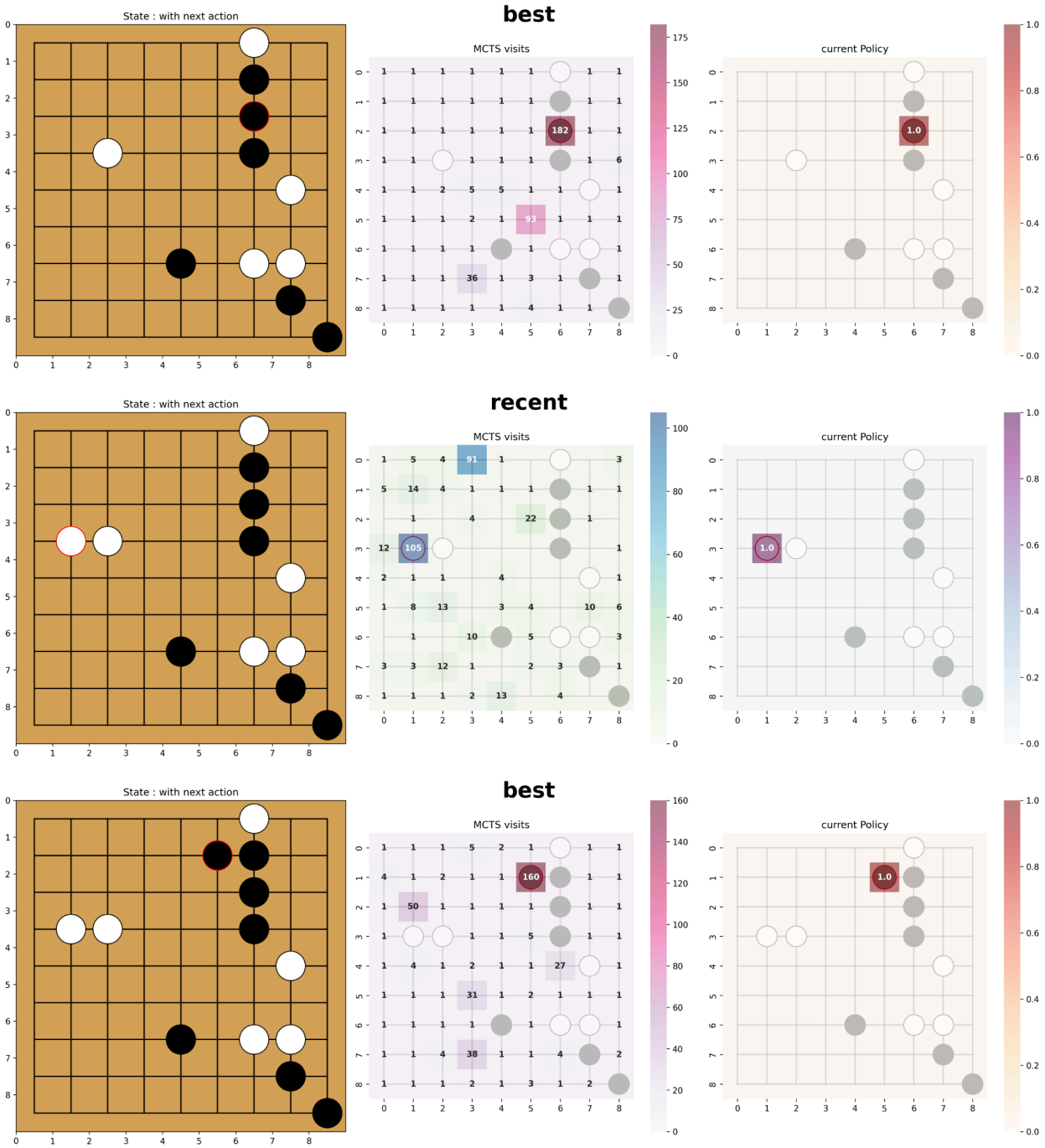
1. 초반 자가 대국 (1000 자가대국 이후 평가 : 당시 최고 모델이 선을 잡음)

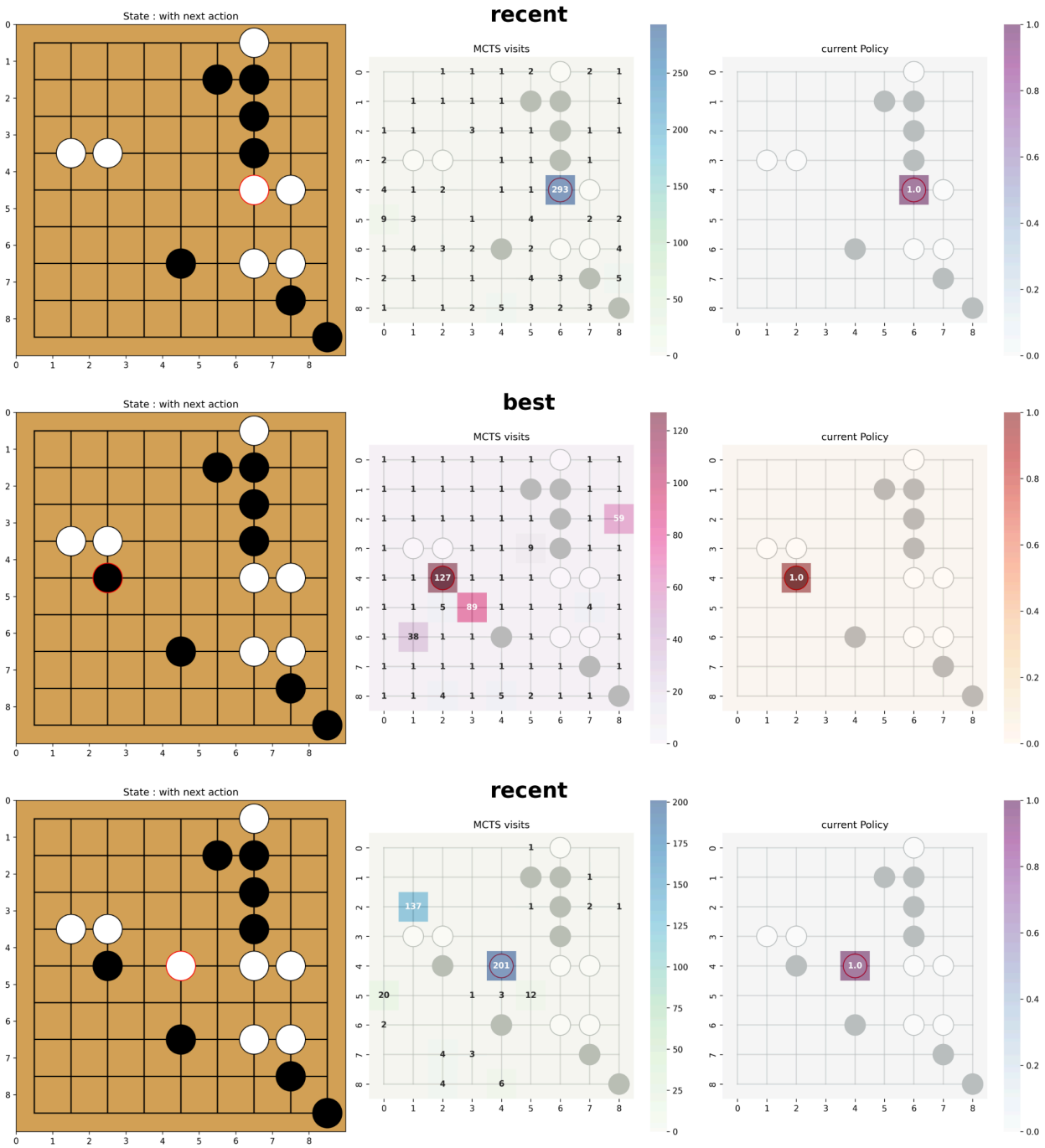


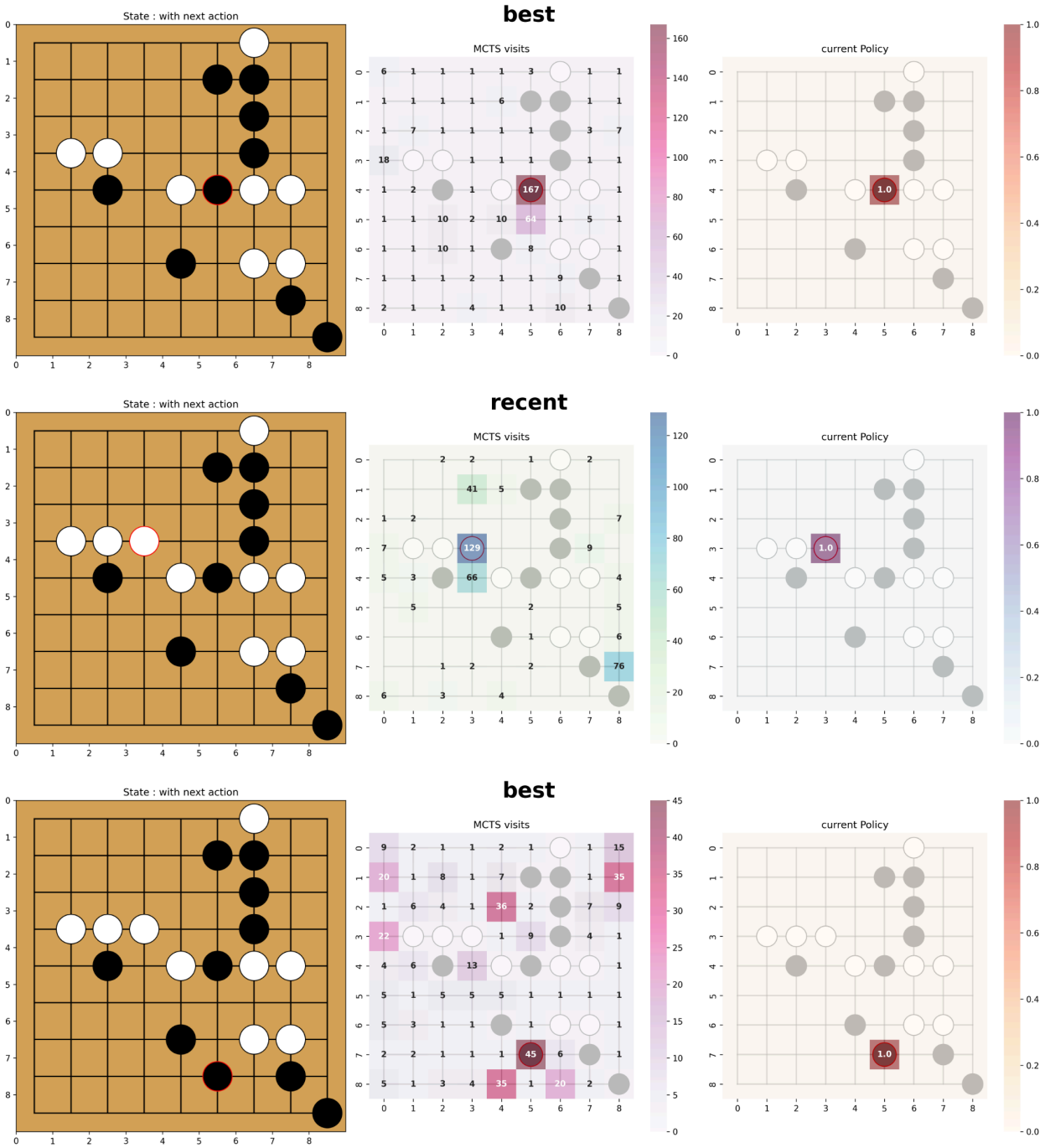

recent
State : with next action
MCTS visits
current Policy
best
State : with next action
MCTS visits
current Policy
recent
State : with next action
MCTS visits
current Policy

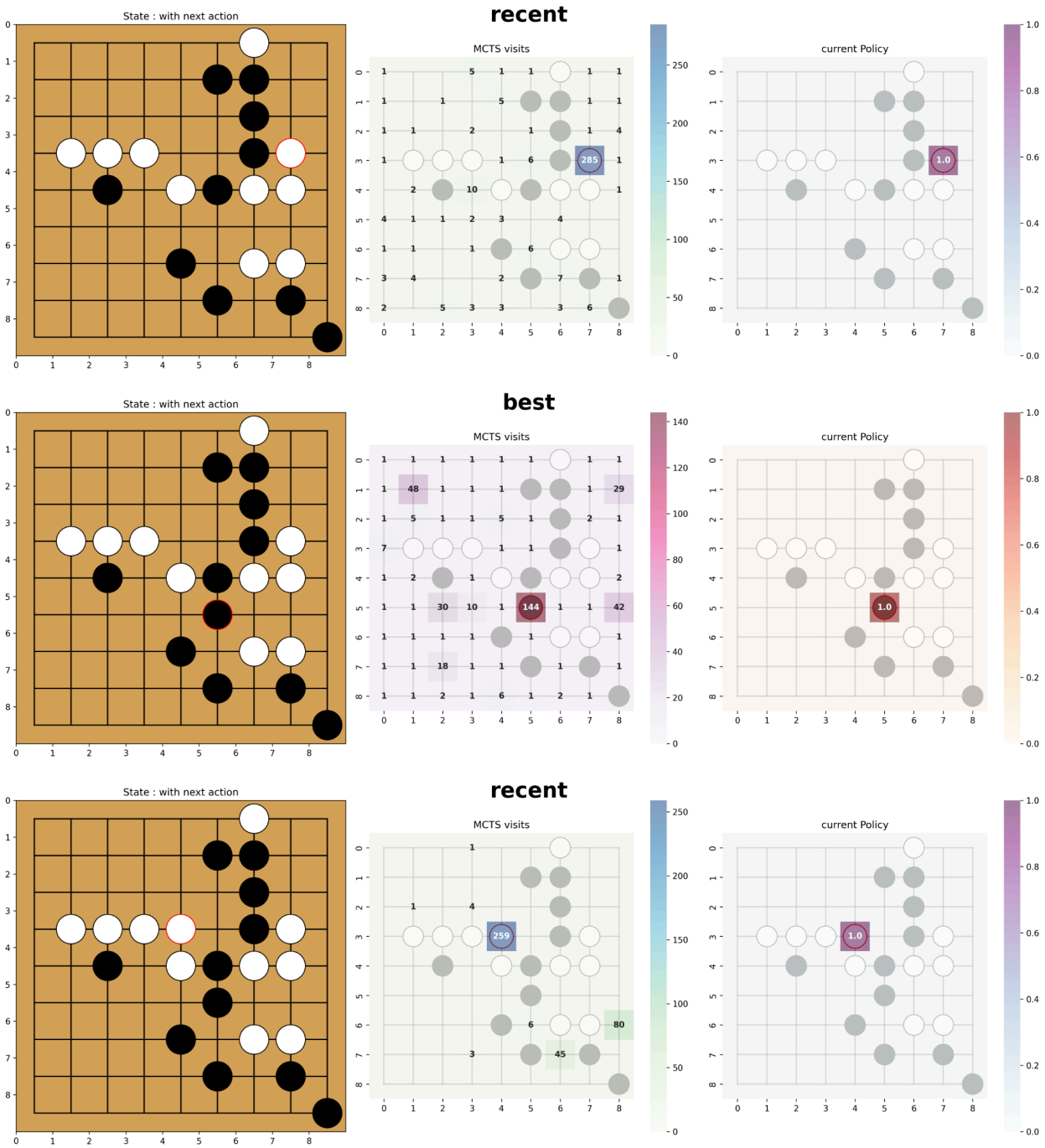

best
State : with next action
MCTS visits
current Policy
recent
State : with next action
MCTS visits
current Policy
best
State : with next action
MCTS visits
current Policy

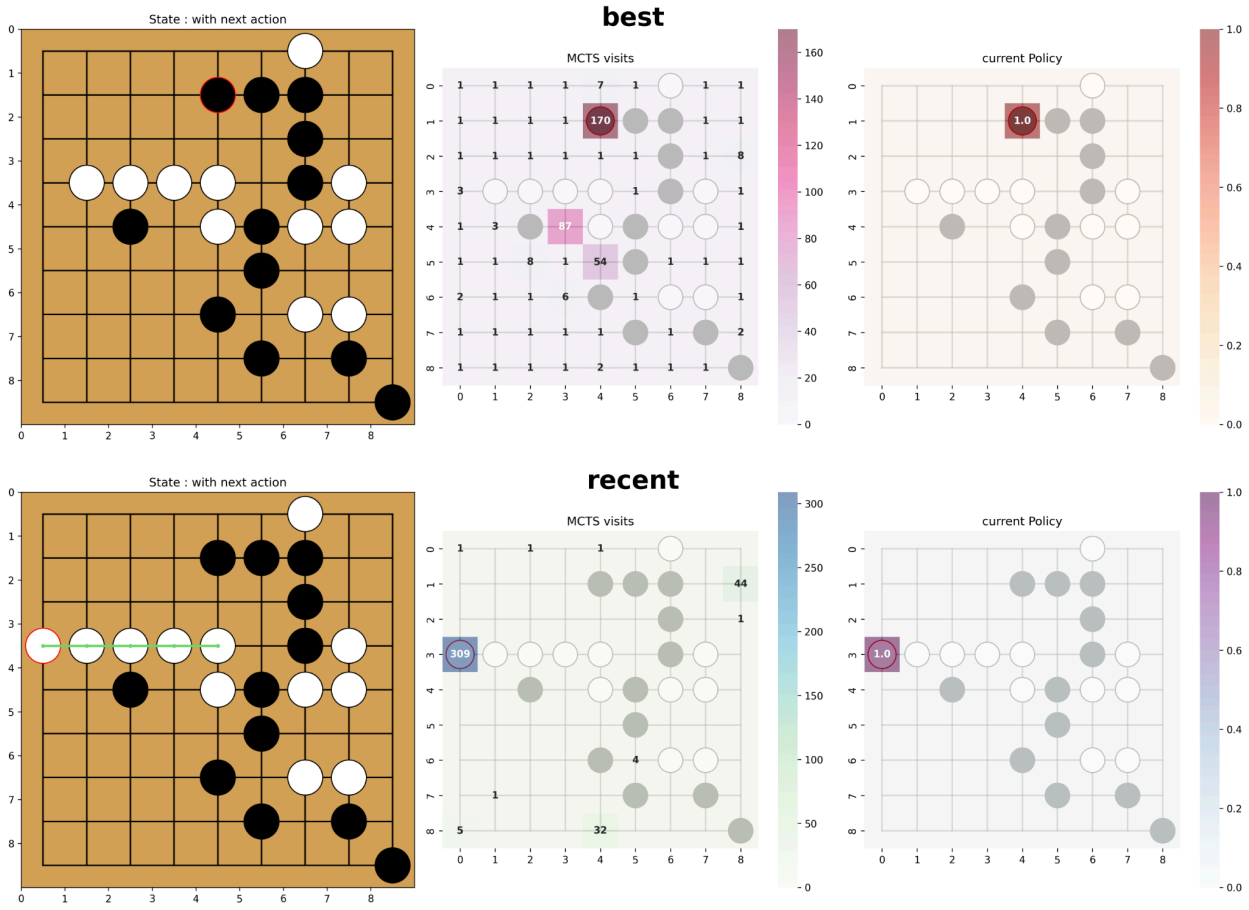

State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy


State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy

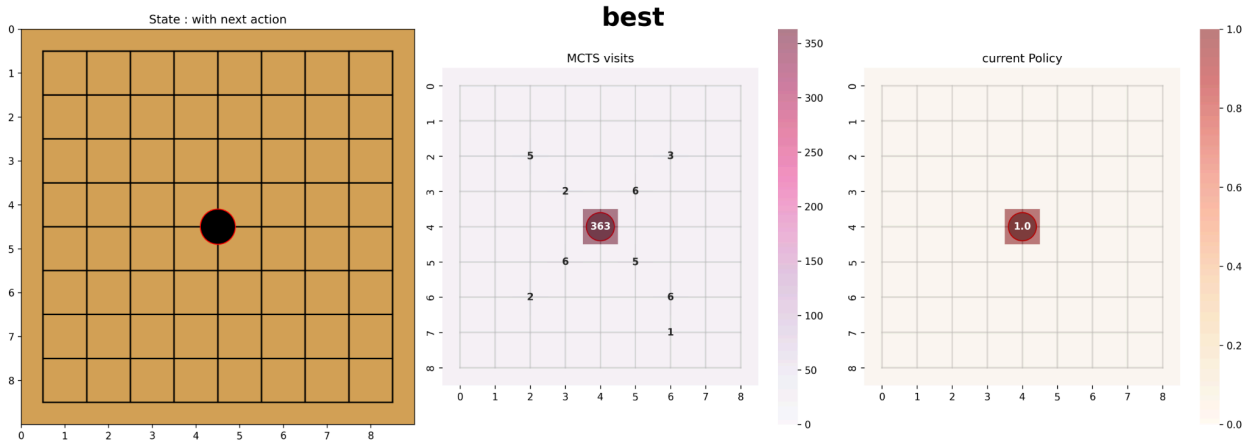

State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy

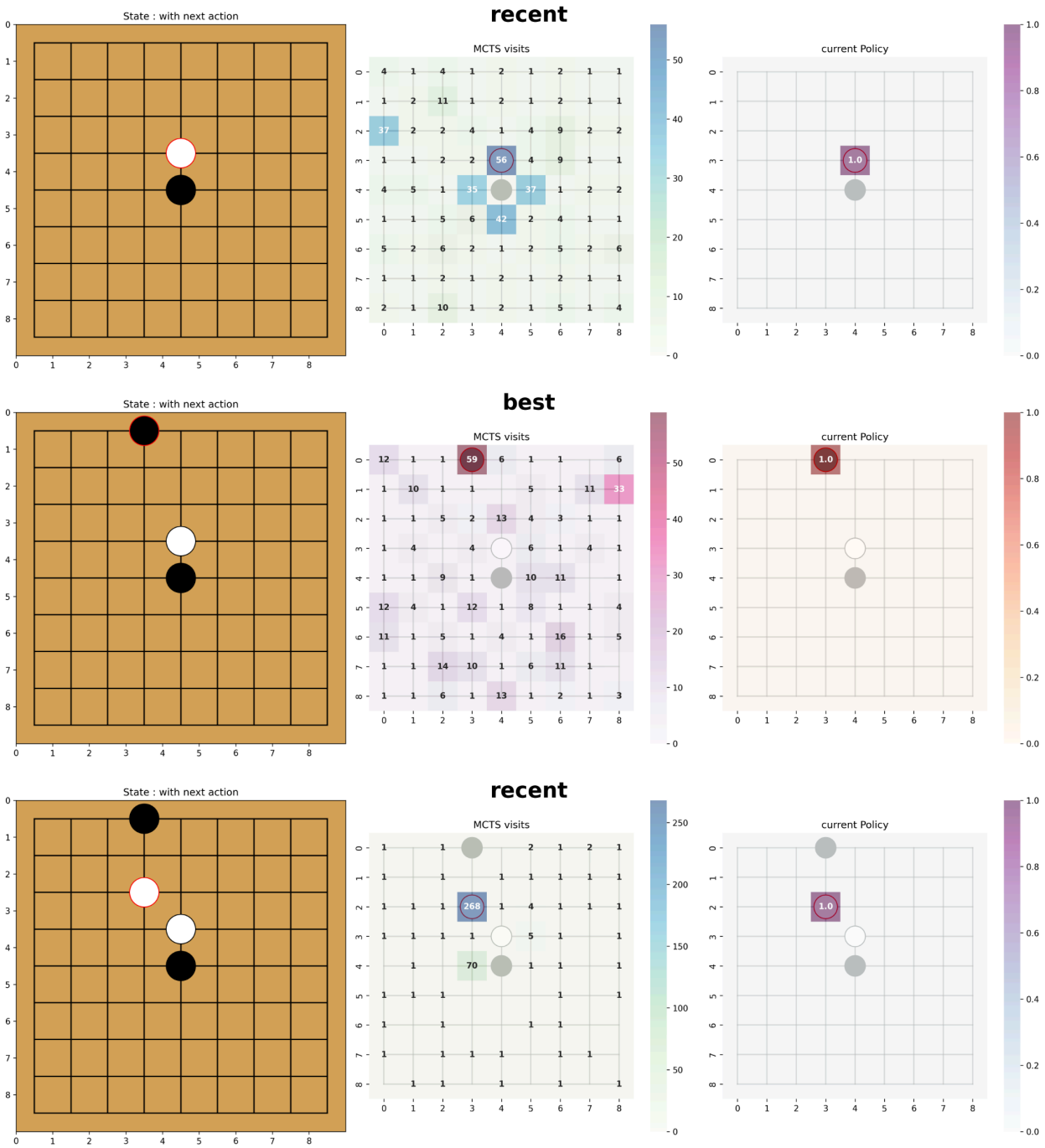

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy

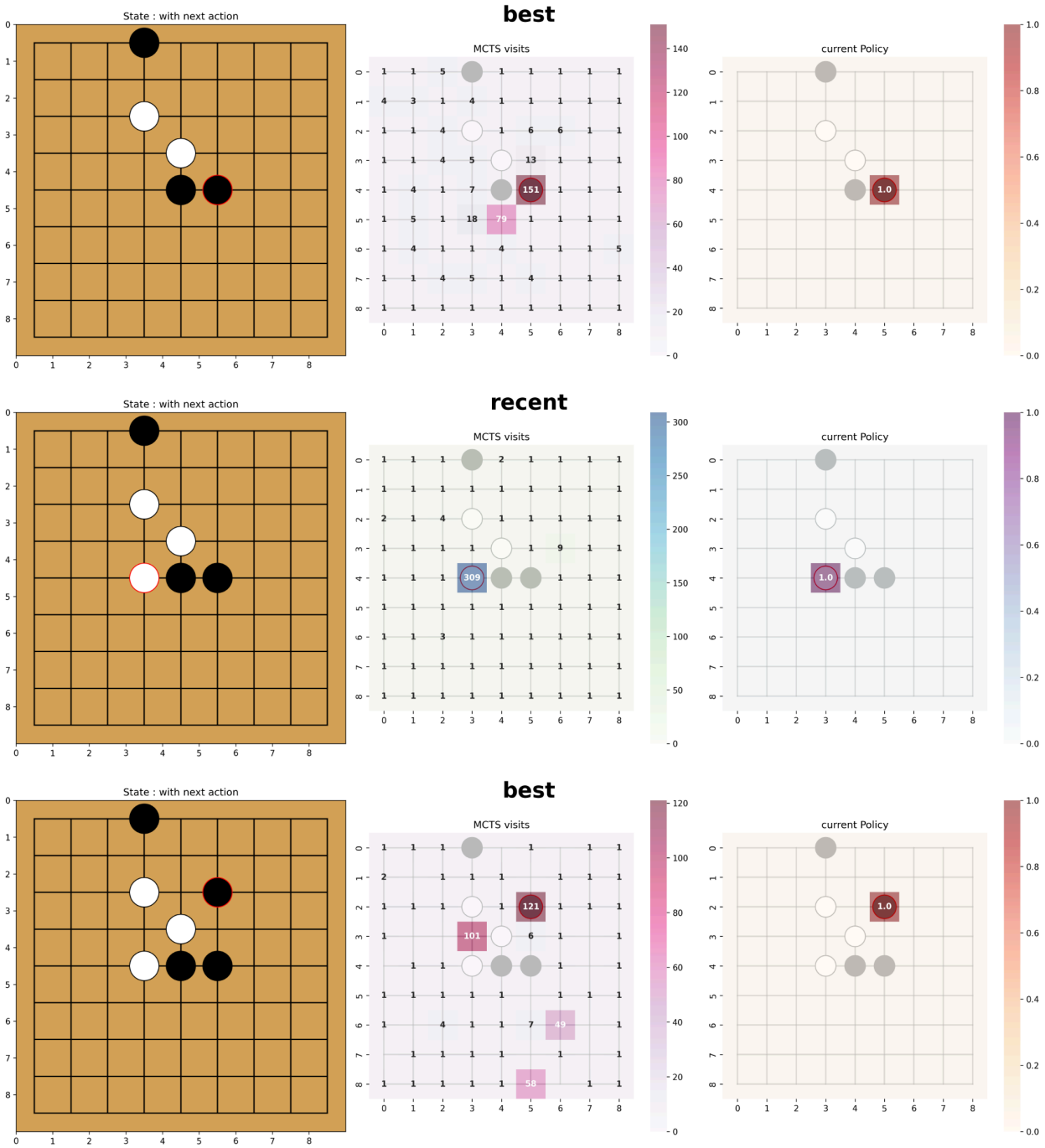

State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy

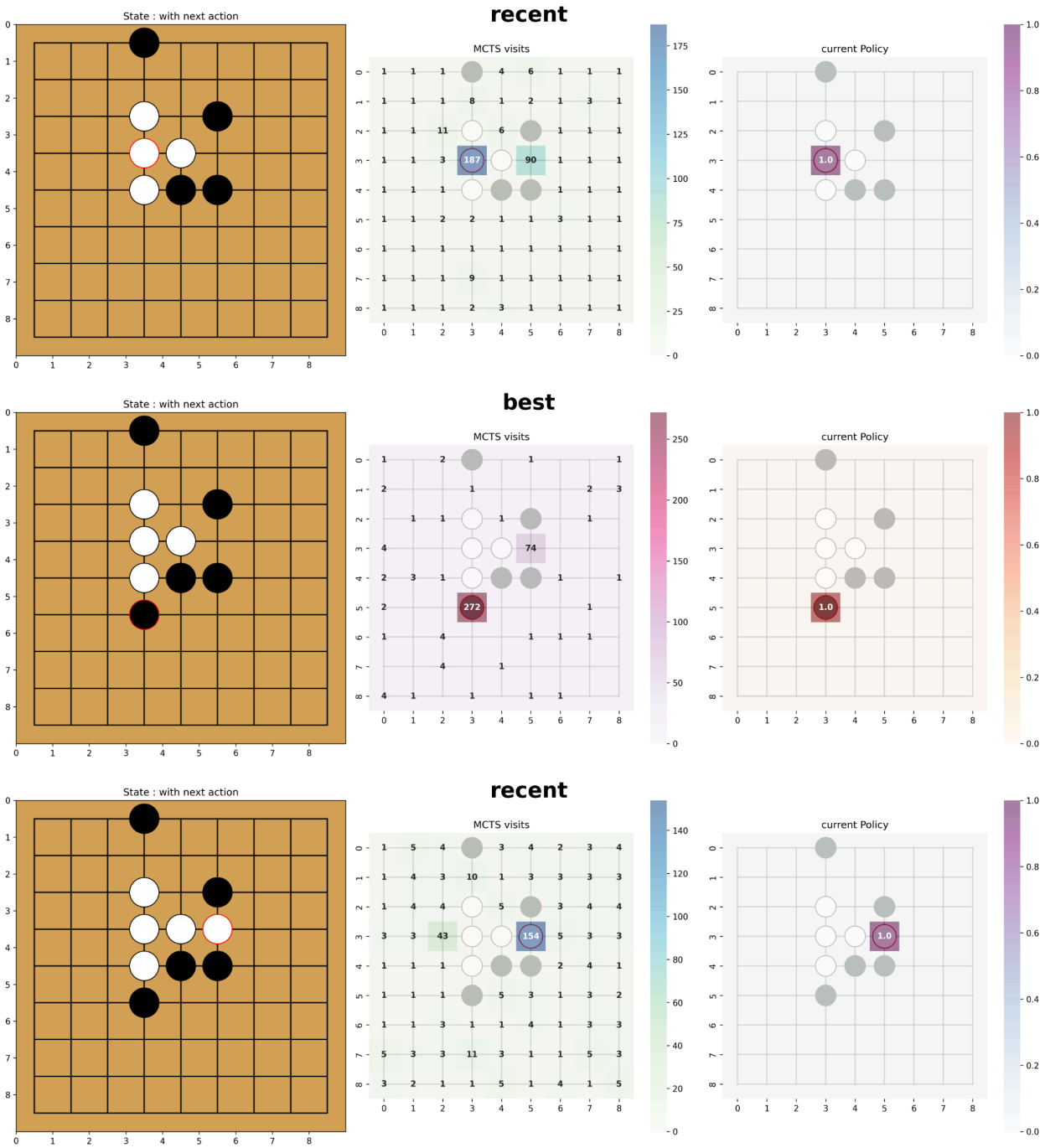


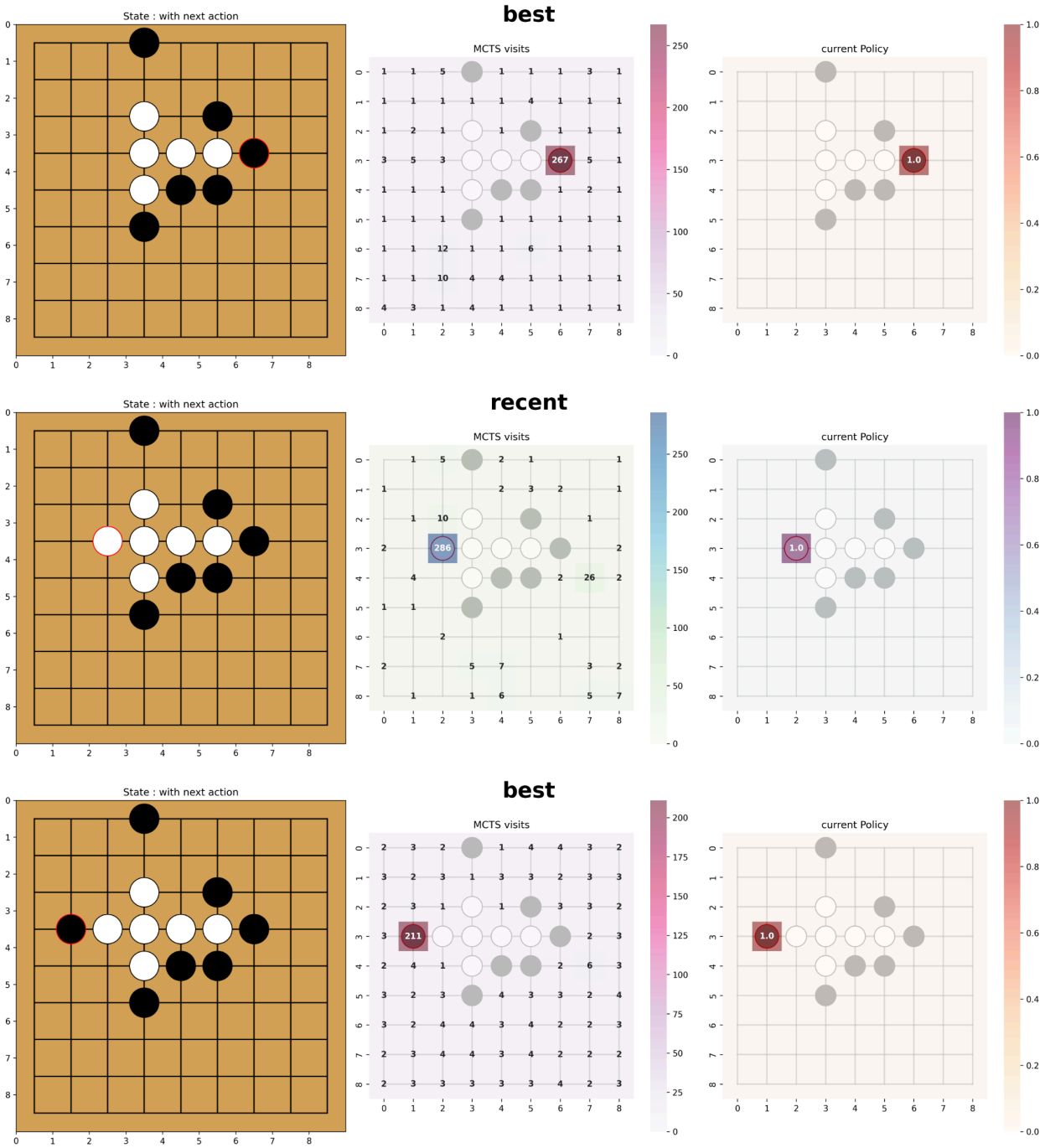
2. 중반 자가 대국 (자가 대국 5000번 이후)

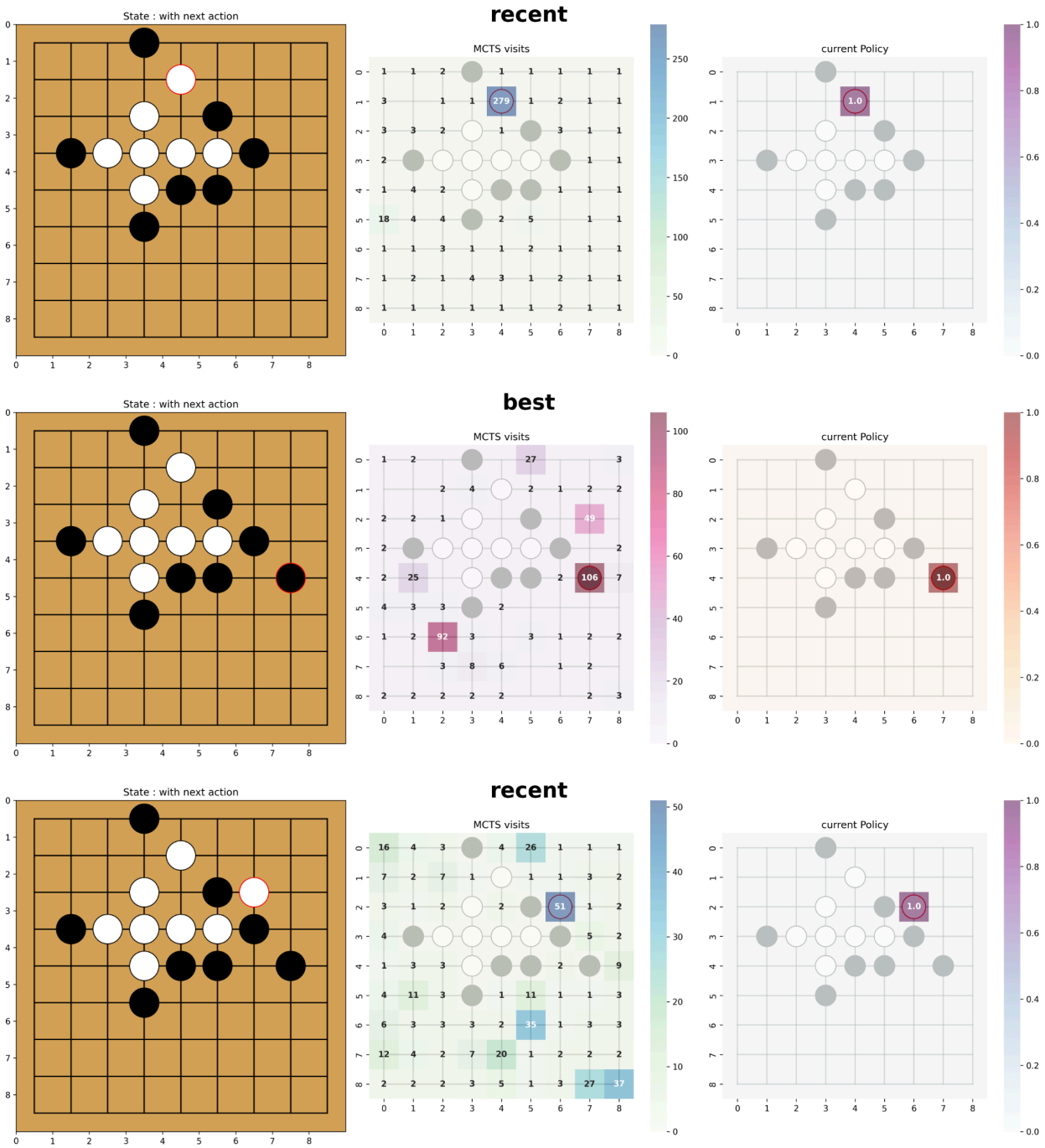


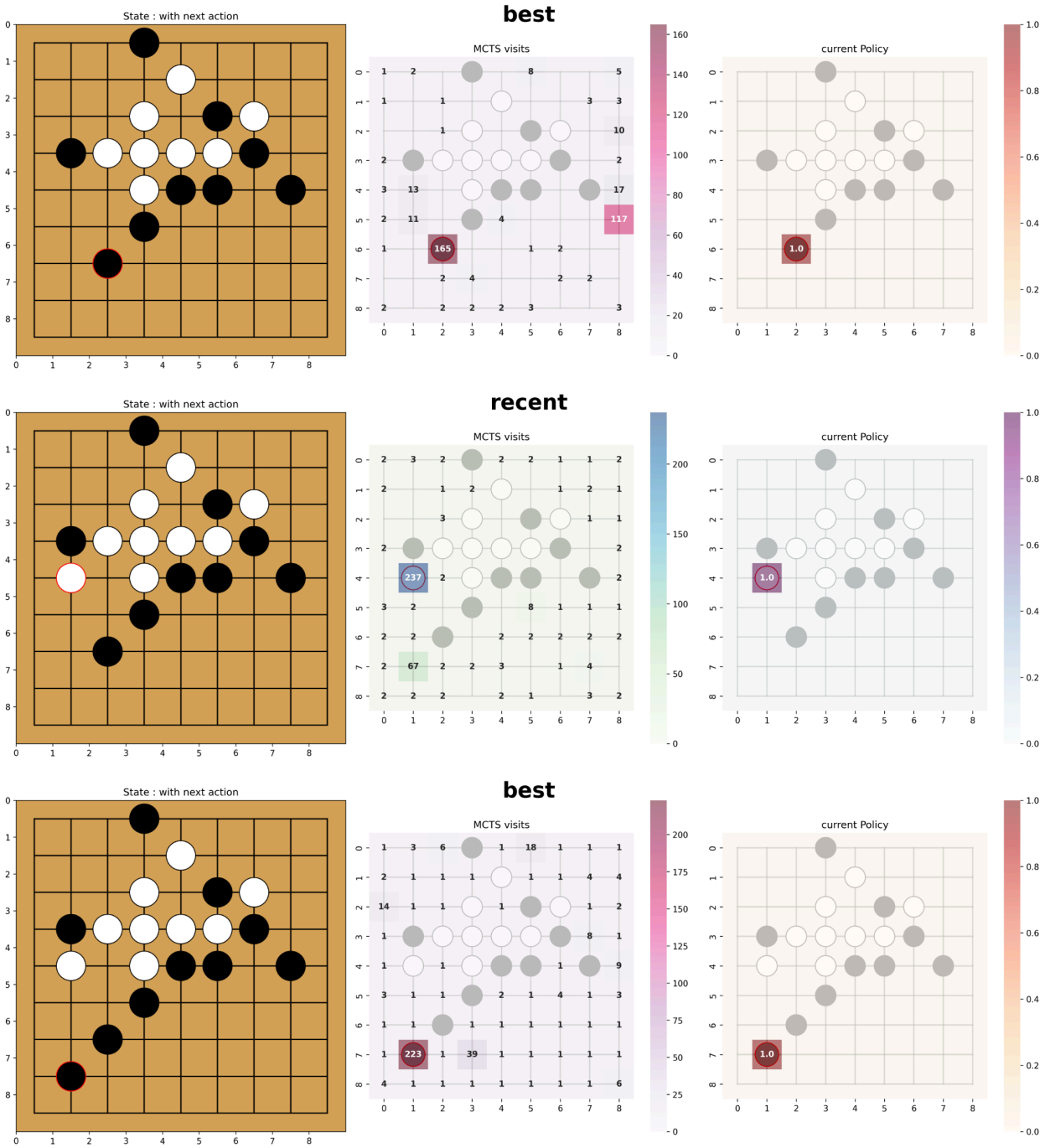

recent
State : with next action
MCTS visits
current Policy
best
State : with next action
MCTS visits
current Policy
recent
State : with next action
MCTS visits
current Policy

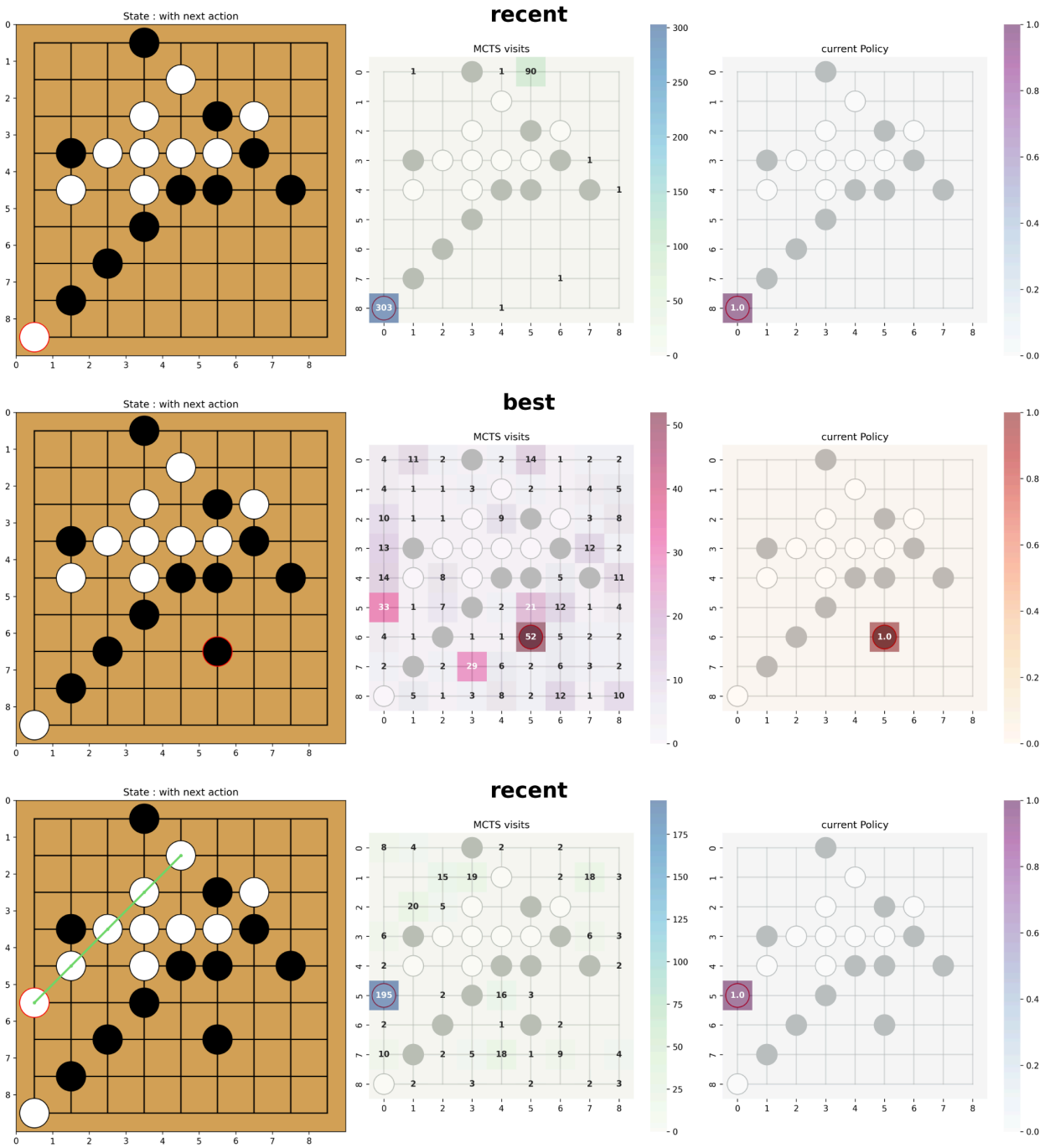

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy


State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy

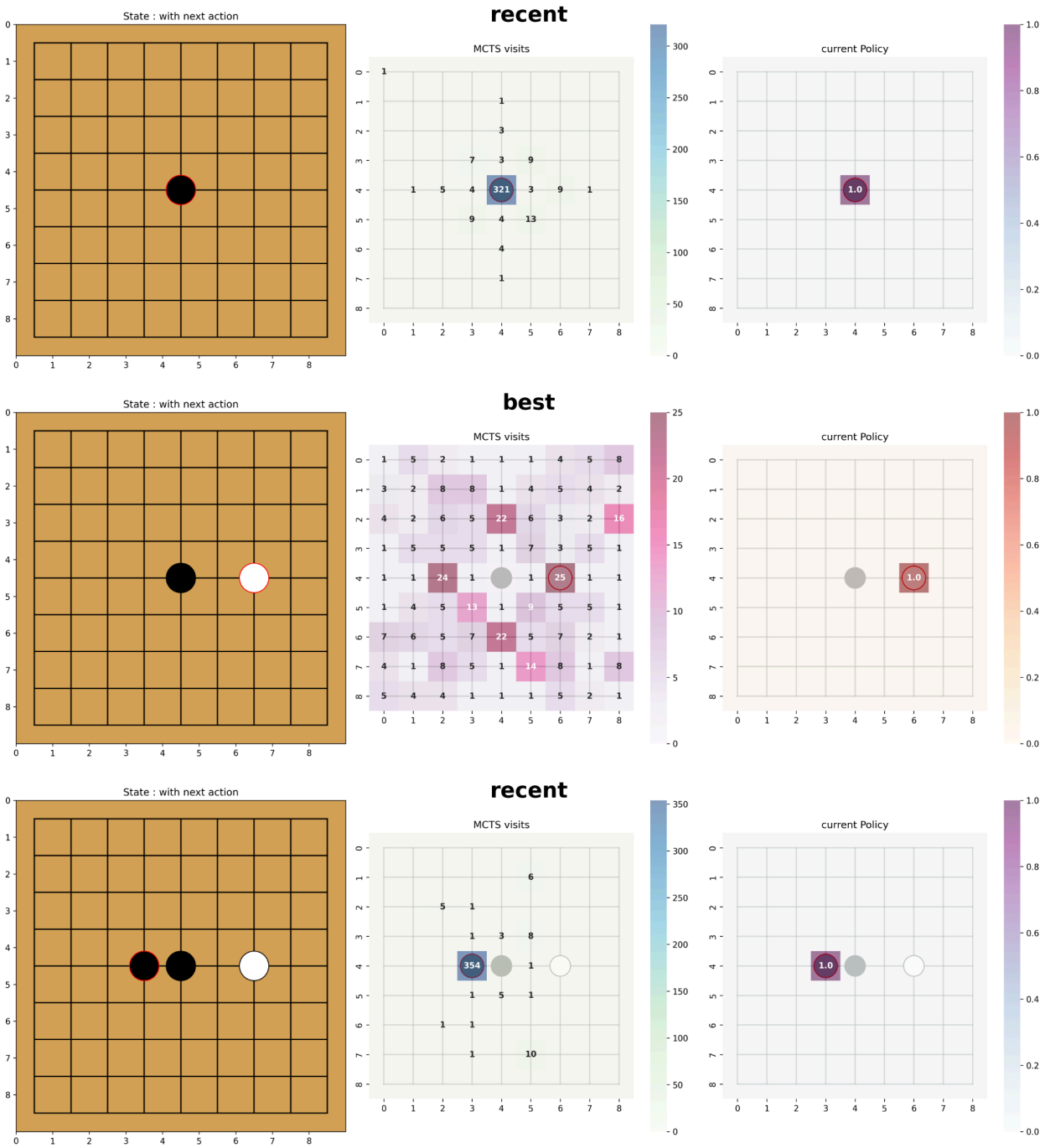

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy

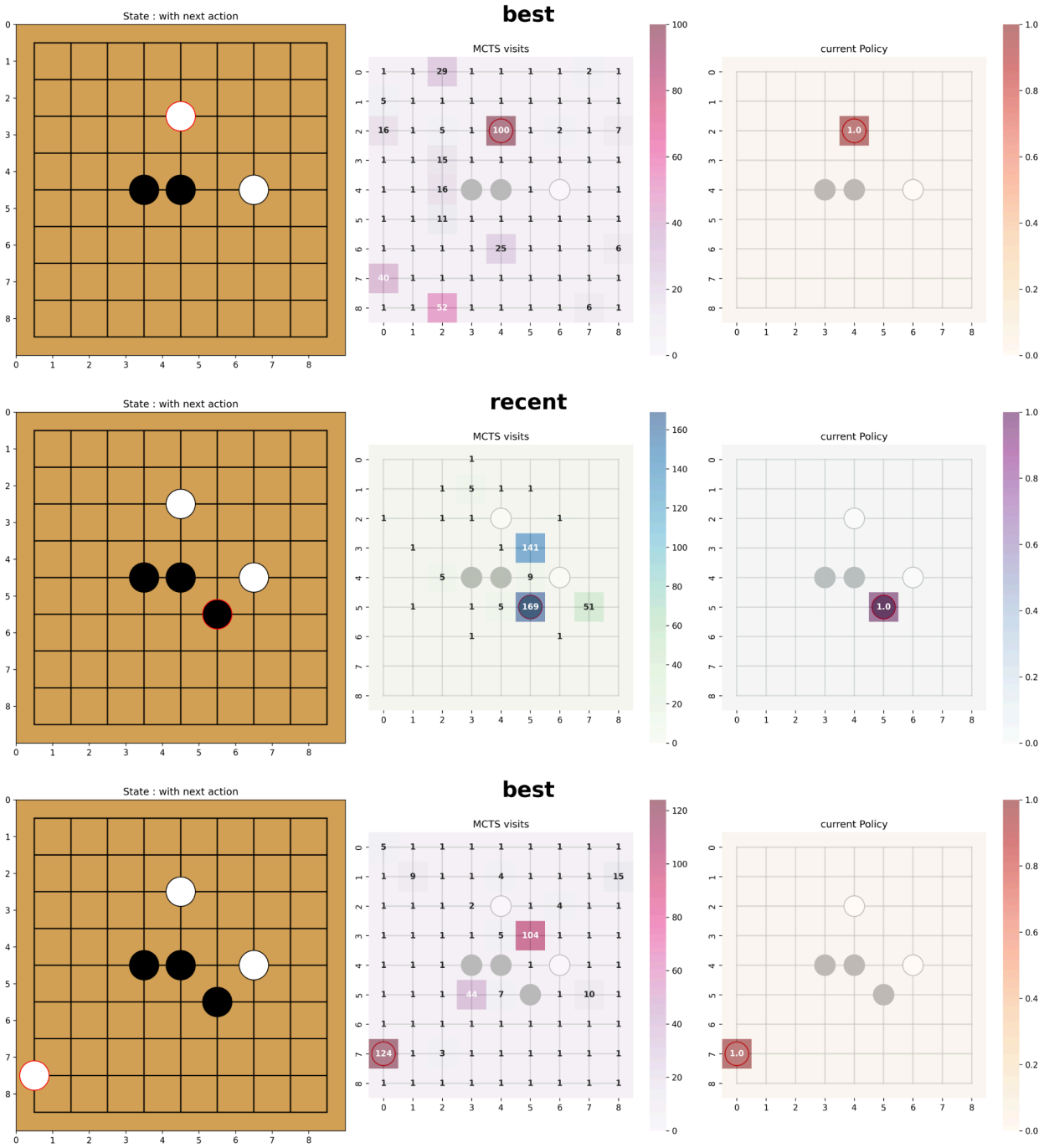

State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy

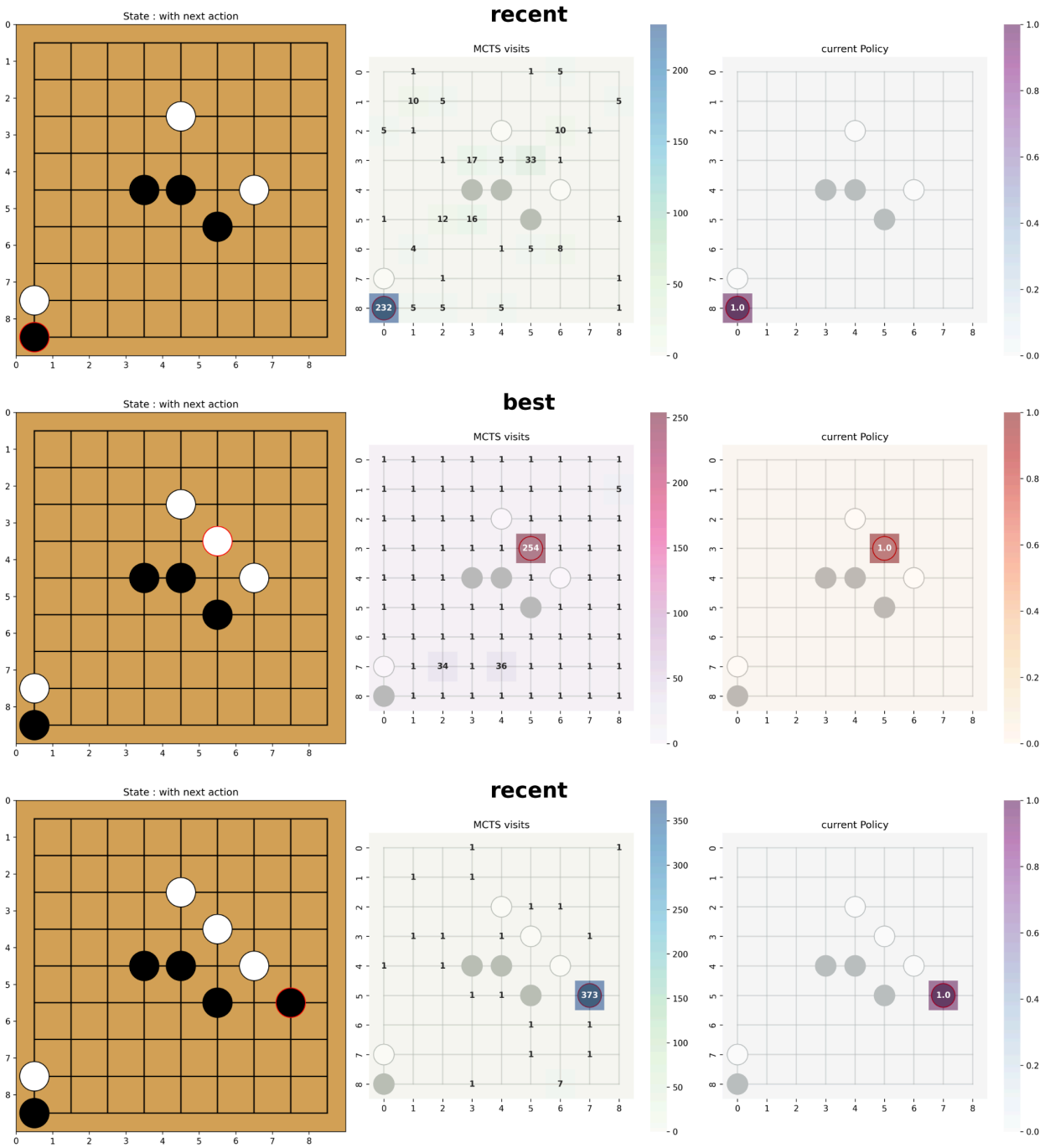

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy

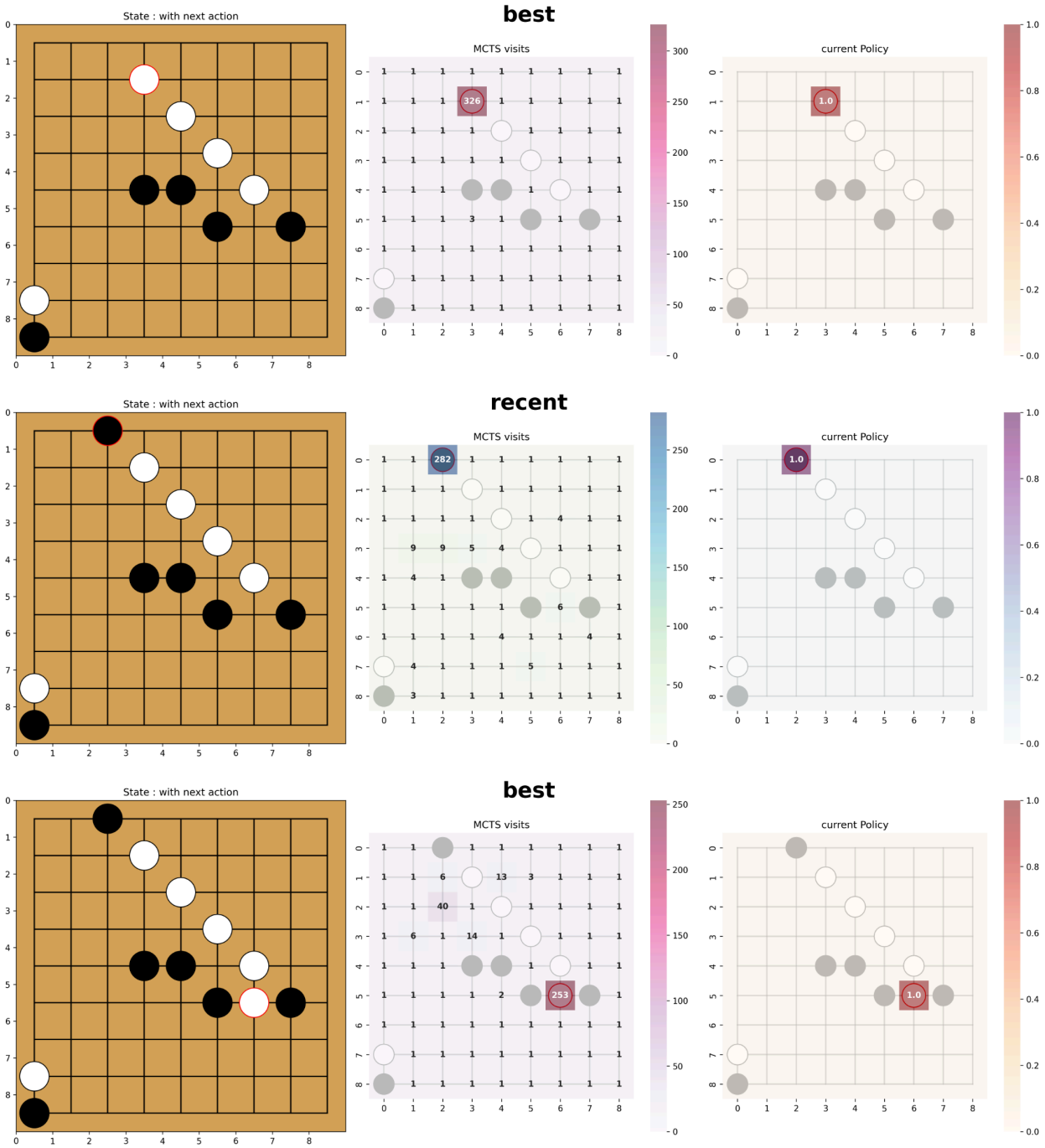


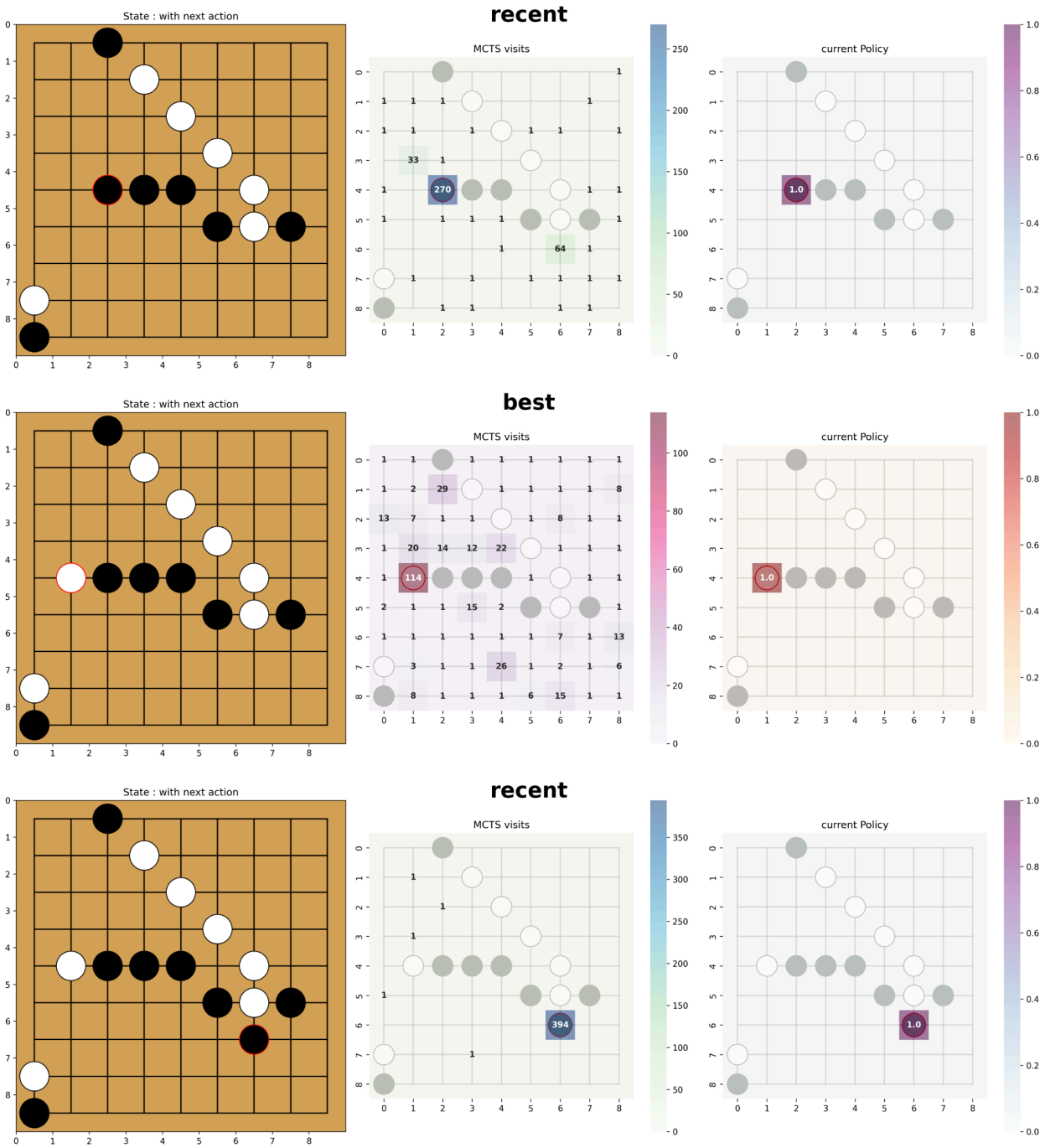
3. 후반 자가 대국 (11000번 이후)

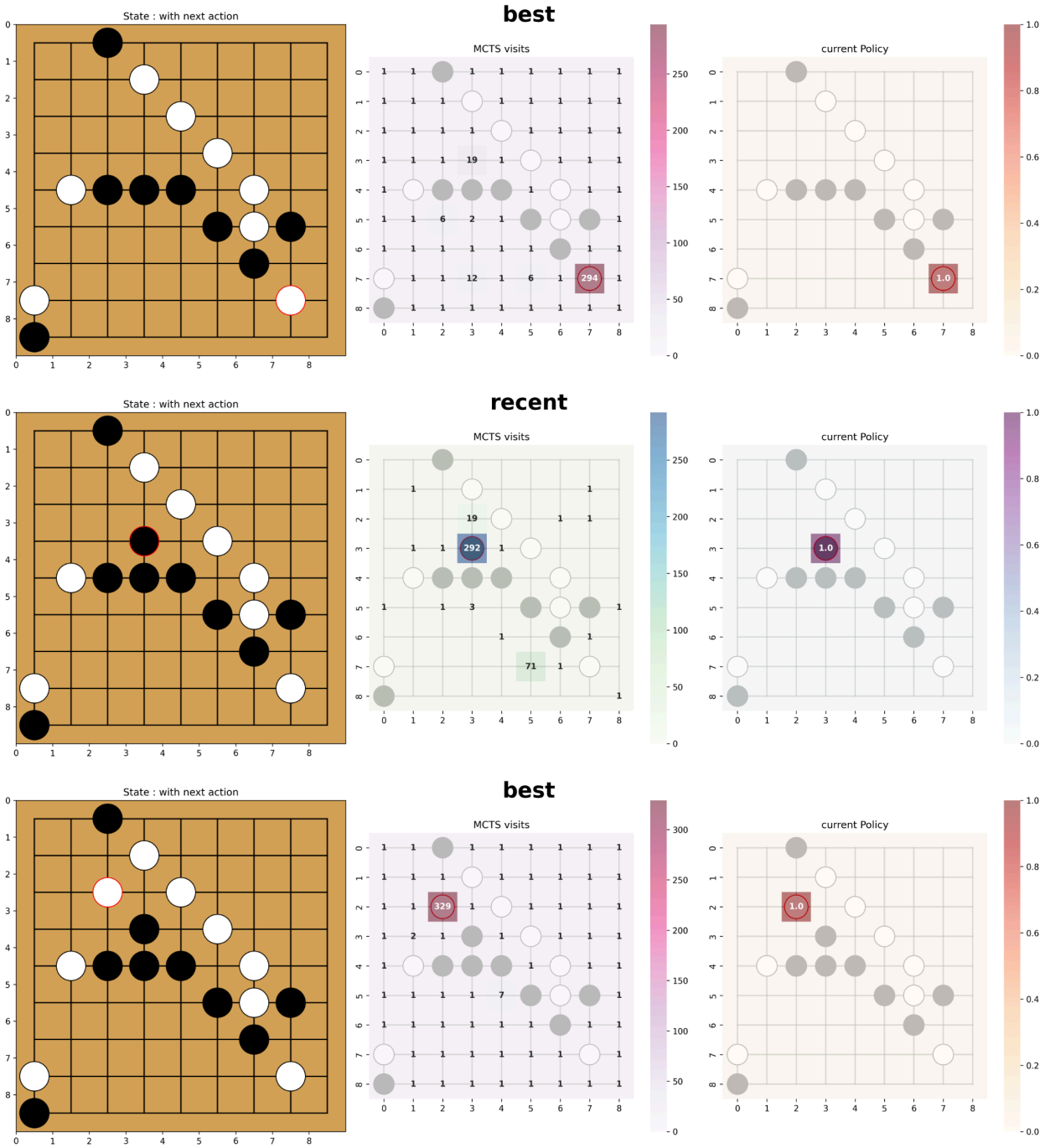

State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy

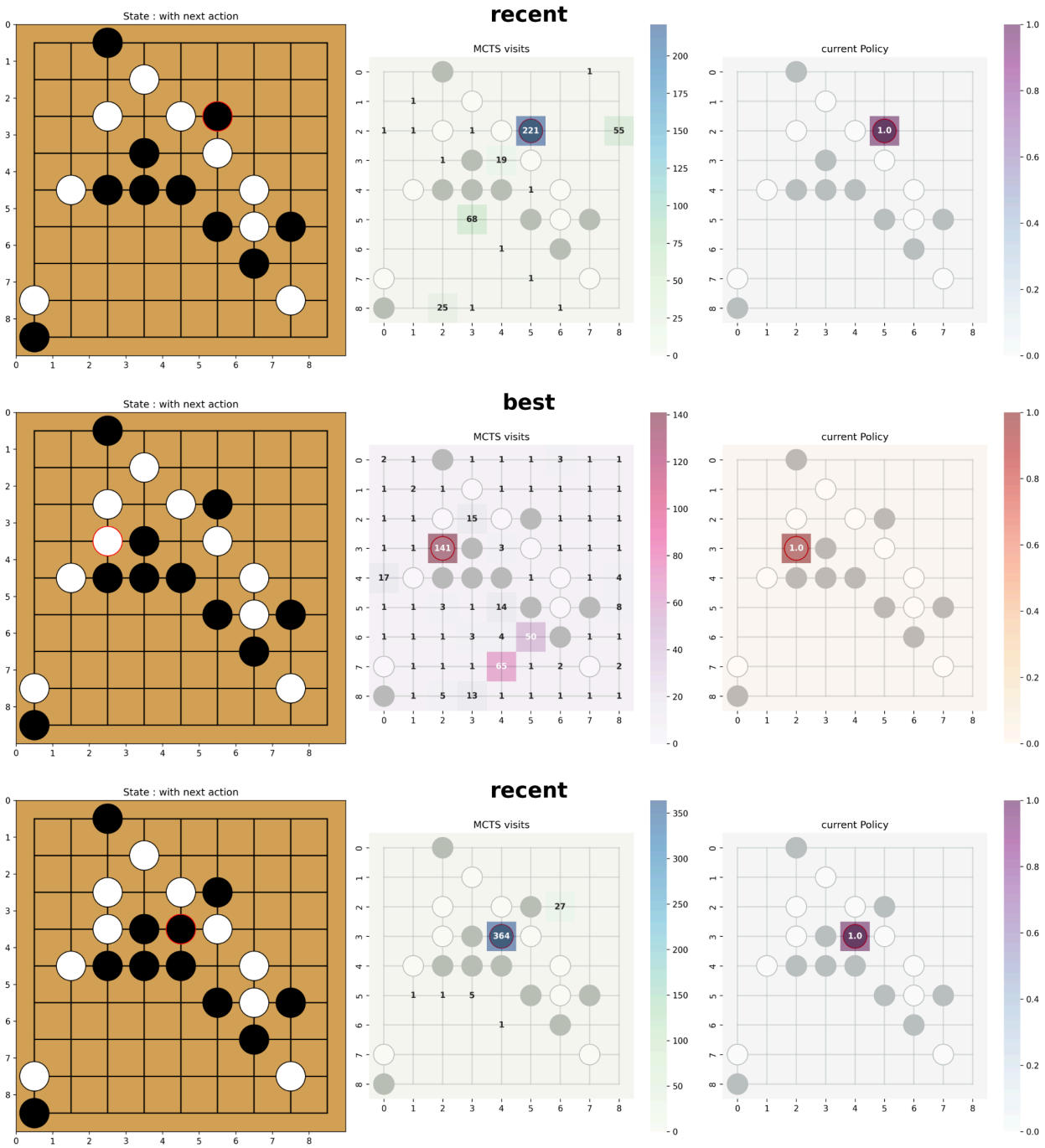

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy

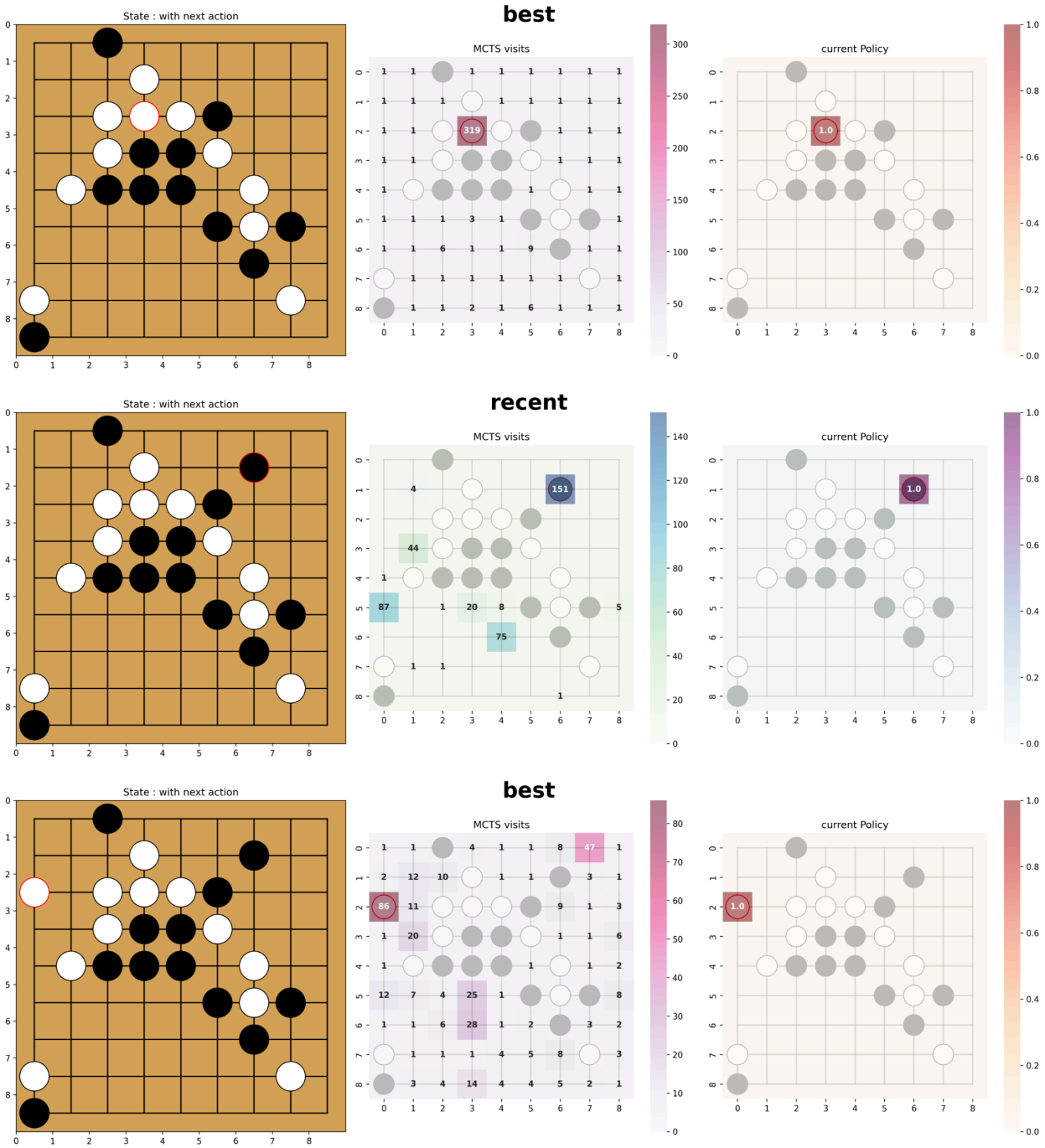

recent
State : with next action
MCTS visits
current Policy
best
State : with next action
MCTS visits
current Policy
recent
State : with next action
MCTS visits
current Policy

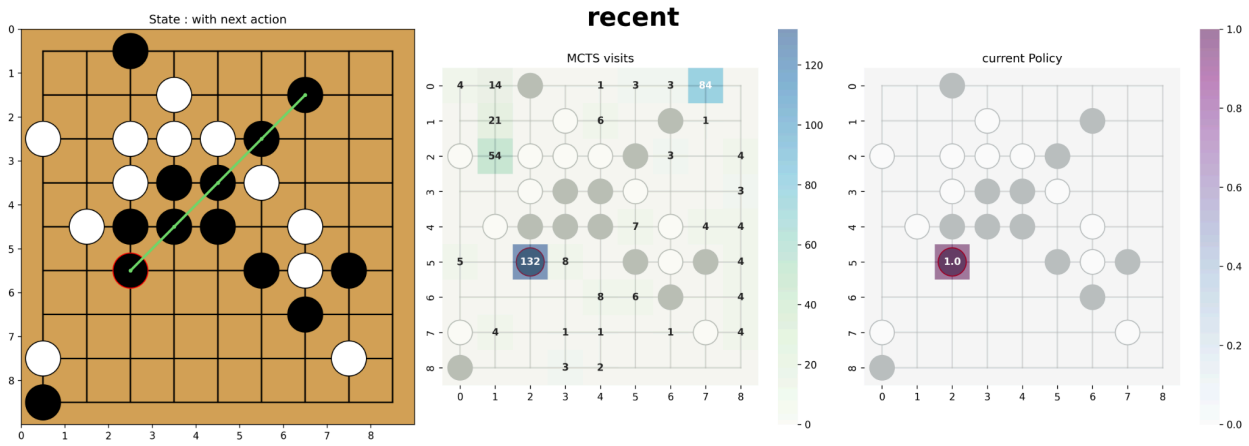

State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy


State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy


State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy


State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy


State : with next action
best
MCTS visits
current Policy
State : with next action
recent
MCTS visits
current Policy
State : with next action
best
MCTS visits
current Policy


State : with next action
recent
MCTS visits
current Policy
4 14 1 3 3 84
21 6 1
54 3 4
3
7 4 4
5 132 8 4
8 6 4
4 1 1 1 4
3 2
1.0

참고문헌

[1] Silver, D., Schrittwieser, J., Simonyan, K. et al. “Mastering the game of Go without human knowledge”. *Nature* 550, 354 – 359 (2017). https://doi.org/10.1038/nature24270

[2] Jpub, "AlphaZero," GitHub repository, Available: https://github.com/Jpub/AlphaZero.git. [Accessed: Feb. 27, 2025].

[3] Junxiaosong, "AlphaZero_Gomoku," GitHub repository, Available: https://github.com/junxiaosong/AlphaZero_Gomoku.git. [Accessed: Feb. 27, 2025].

[4] Reinforcement Learning KR, "Alpha Omok," GitHub repository, Available: https://github.com/reinforcement-learning-kr/alpha_omok.git. [Accessed: Feb. 27, 2025].