

# Reinforce Learning 프로젝트 -Yo-gi-ho Puzzle Duel Project-

소속 : 서강대학교 대학원 인공지능학과

팀명 : Re:Dash

학번:120230248

작성자 & 팀장 & 팀원 : 조재혁

## 0. 목차

- 서론
- 1. 문제제시
- 2. **Puzzle Rule**
- 3. 아이디어 발전과정?
- 4. 해결의 기초 idea
- 5. 학습 환경 과정 & 결과
- 6. 개선점 & 차후 계획
- 결론

# 서론 (Introduction)

- 강화학습(Reinforcement Learning, RL)은 환경(Environment)과의 상호작용을 기반으로 최적의 행동 정책을 학습하는 기계학습 분야로,  
특히 규칙 기반 게임, 퍼즐, 시뮬레이션 상황에서 뛰어난 성능을 보여왔다.  
*Go(바둑), Atari 게임, StarCraft II* 등에서 RL의 우수성이 이미 입증되었지만, TCG카드 게임은 게임 구조의 특성상 다른 RL 문제보다 훨씬 어렵다.
- 카드 게임은 다음과 같은 난제를 가진다.
- 카드 자체의 효과와 게임 자체의 둘 2가지 규칙의 제약을 갖는다.
- 상태 공간이 매우 크다 (패, 필드, 효과, 포지션 등)
- 행동이 순차적이며 의존성이 강하다
- 승리/ 패배 외 보상이 거의 없다는 희소 보상 문제이다(Sparse Reward)
- 정확한 규칙 엔진이 필요하다.
- 따라서 본 연구는 실시간 듀얼 AI가 아닌, 규칙이 고정된 환경에서 에이전트가 명확한 정답 시퀀스를 찾는 **Puzzle Duel**을 RL 환경으로 재구성하여 알고리즘 성능을 비교하는 데 초점을 맞춘다. 다양한 RL 알고리즘(DQN, Double DQN, REINFORCE, A2C)을 비교하는 실험을 수행하였다.

# 1. 문제제시

- Edo\_Pro라는 유희왕 카드 게임에서 강화학습 AI로 실시간으로 듀얼을 하게 할 수 있을까?에서 시작했다.(그러나 실시간으로 만들기엔 각 카드의 효과 및 순서로 인해 상호작용이 다른 점의 구현이 다른 점, 카드의 발동의 적절한 타이밍, 매번 바뀌는 환경에 대한 처리 등등, 제일 결정적인 문제는 시간상 구현이 불가능)
- 위의 문제로 인해 위의 계획은 불가하고 판단했다. → 그래서 puzzle duel로 변경했다.
- 2가지의 퍼즐듀얼을 대상으로 진행(Oh\_Jama, Ancient\_Kings)했다.

Oh\_Jama : 초반 Ojama Trio + Token + Zero Gravity를 어떻게 활용하는지가 핵심

목표: Ultimate Tyranno가 토큰 3장을 연속 공격하여 OTK

Ancient\_Kings: 카드 간 순서 의존성이 매우 높다.(난이도가 높다)

난이도는 훨씬 높고, 잘못된 순서 하나면 클리어 불가

- “강화학습으로 처음 보는 Puzzle duel을 풀면 많은 가능성 중에 1가지의 해법을 찾을 수 있을까?”에서 출발했다.
- Yu-Gi-Oh! 퍼즐 듀얼 문제는 에이전트가 환경과의 상호작용을 통해 최적의 행동 정책을 학습하는 마르코프 결정 과정(MDP)으로 공식화됩니다. 따라서 다음 섹션에서는 이 문제의 상태 공간, 행동 공간, 보상 체계를 정의하고 설계한 내용을 상세히 기술합니다.

## 2. PUZZLE DULE

- Puzzle Dule이란?

- 일반적인 퍼즐 듀얼과 같이, 세트 카드, 패, 텍의 순서 등 모든 것이 공개 정보이며, 이번 본인의 턴에 승리하면 성공, 패배하거나 해당 턴에 승리하지 못한 채로 턴을 종료하면 실패로 판정합니다.
- 본 프로젝트에서 사용한 퍼즐듀얼은 2008년 퍼즐듀얼입니다. 2008년의 현재의 듀얼 시스템으로 퍼즐듀얼을 할 경우, 강화학습의 이유는
- "복잡한 TCG 환경의 난제를 극복하고 강화학습의 효과를 검증하기 위해 퍼즐 듀얼을 선택했습니다.
- 1. 희박 보상 → 명확한 보상 구조: 일반 TCG의 보상과 달리, 승리(+100) 및 패배(-100)가 명확하며, \*\*잘못된 행동(-10~50)\*\*에 대한 Shaping이 용이합니다.
- 2. 거대한 상태 공간 → 고정된 환경: 필드, 패, 묘지, 텍이 변하지 않는 **Closed World**로 설정되어, 상태 공간 탐색의 복잡도를 획기적으로 낮췄습니다.
- 3. 정답 시퀀스 존재: Optimal Policy가 존재하므로, RL Agent가 해법을 찾았는지를 객관적으로 검증할 수 있습니다."

## 2-1 PUZZLEDUE 기반 설계

- Agent : player → DQN, Double\_DQN, Reinforce\_Baseline, Advantage Actor-critic
- Environment : 에이전트가 상호작용하는 세계 (게임 룰, 현 상황의 필드, 패, 묘지, 텍, 카드 위치 및 카드의 종류 등등), state, State Transition, 각각의 퍼즐듀얼에 환경을 분리해서 설정
- Episode : 1판의 게임이 끝날 때 까지가 1 Trajectory
- state: 필드/패/묘지/텍, 카드 상태, 턴, 페이즈
- action:
  - 카드 발동 : 세트, 발동, 소환등등
  - 몬스터 공격 : 5개의 몬스터 존 \* 5개의 상대의 몬스터 존 + 5곳의 직접공격=30
  - 페이즈 변경 : 매인페이즈, 배틀페이즈, 엔트 페이즈 3개가 존재
  - 총 42개 액션(유효 액션은 5~10개)

## 2-2 사용 알고리즘

- **Deep Q-Network(DQN)** — Atari 게임에서 인간 수준 능력을 보임
- **Double DQN** — 두 개의 네트워크를 활용하여 Q-값의 선택(Selection)과 평가(Evaluation)를 분리합니다.
- **Policy Gradient & REINFORCE** — 몬테카를로 방식의 확률 정책 학습
- **Actor-Critic(A2C)** — 가치 함수와 정책을 동시에 학습
- 카드 게임 RL 연구 — **Hearthstone**, **Magic: The Gathering** 등에서 시도되었지만 대부분 완전한 규칙 엔진과 매우 넓은 상태 공간 때문에 난이도가 높음
- 본 연구는 완전한 **Yu-Gi-Oh AI Agent** 대신, 퍼즐 해결에 필요한 최소한의 구조만 구현한 점이 특징이다.

## Policy Gradient / A2C / DQN Agent

### ● REINFORCE & Baseline

```
loss = -(log_probs * advantages).mean()  
value_loss = F.mse_loss(values, returns)  
•정책 기반 업데이트  
•Baseline( $V(s)$ )을 사용해 variance 감소
```

### ● A2C

```
actor_loss = -log $\pi(a|s)$  * A  
critic_loss = ( $V$ -target)**2  
entropy = -entropy  
•Actor–Critic 구조  
•탐험(Entropy)과 안정성 개선
```

### ● DQN (Value-based)

```
target = r +  $\gamma$  * target_q(next_state).max()  
loss = HuberLoss(current_q, target)  
•Replay Buffer + Target Network  
•Double DQN은 overestimation 해결  
•Dueling DQN은  $V(s)/A(s,a)$  분리
```

## 2-3 Environment

- **state:** 필드/패/묘지/덱, 카드 상태, 턴, 페이즈
- **action:** 행동 공간
  - 카드 발동 : 세트, 발동, 소환등 패 5장/필드 5존 중 1장 발동 → 10가지
  - 소환/세트 (**Summon/Set**): 패 5장 중 1장 소환/세트 → 5가지
  - 몬스터 공격 : 5개의 몬스터 존 \* 5개의 상대의 몬스터 존 + 5곳의 직접공격=30
  - 페이즈 변경 : 매인페이즈, 배틀페이즈, 엔트 페이즈 3개가 존재
  - 총 42개 액션(유효 액션은 5~10개)
- 기본적인 Reward 설계

액션/상태 변화	코드 기준 (Base Reward)	보상 (Shaped Reward)	설계 의도
성공적인 몬스터 소환/카드 발동	base_reward > 0 인 유효 액션 (진행)	base_reward * 0.5	긍정적 진행 보상. 에이전트가 카드를 활용하여 필드를 구성하는 **유의미한 행동**을 수행하도록 장려.
상대 LP 데미지 발생	lp_damage > 0	(lp_damage / Initial LP) * 100.0	최종 목표 보상. 승리 목표에 가까워지는 행동(데미지)에 가장 큰 공통 보상을 부여.
페이즈 전환	base_reward = 0 인 중립 행동	-0.1 (작은 페널티)	비효율적 행동 억제. 액션 없이 턴을 넘기거나 페이즈를 전환하는 행위에 작은 페널티를 부여하여, 에이전트가 최소 스텝으로 퍼즐을 해결하도록 유도.
무효한 액션 시도	base_reward < 0 (게임 진행 실패)	base_reward * 0.5	실패 페널티. 유효하지 않은 액션 선택을 피하도록 유도.

- Environment 핵심 코드
- def step(self, action): next\_state, reward, done = self.simulator.step(action)
- valid\_mask = self.get\_valid\_actions() return next\_state, reward, done, valid\_mask
- 역할: Simulator가 반환한 결과에 보상을 적용하고, 행동 마스킹을 통해 무효 행동을 제거하여 학습 안정성을 높인다.
- 2) Simulator (simul.py) 유희왕의 실제 룰(공격, 데미지 계산, 파괴, 반사 데미지, 포지션 변경)을 실행하는 핵심 엔진.
- 핵심 코드
- def attack(self, attacker, target):
  - if attacker.atk > target.atk:
  - target.destroy()
  - opp.lp -= attacker.atk - target.atk ...
- 역할: 규칙 엔진을 수행하고, Environment는 보상/ 종료 판정을 담당하는 구조적 분리.

- State Encoder (`state.py`) 핵심 코드
- 필드·패·몬스터 정보 등을 고정 길이 벡터로 인코딩하여 신경망 입력으로 전달한다.
- `state.append(player.ip / 10000)`
- `state.extend(self.encode_monster_zone(player.monster_zones))`
- `state.extend(self.encode_hand(player.hand))`
- 역할: 필드 상태를 Dense Vector로 변환하여 RL 알고리즘이 이해할 수 있는 형태 제공

- Action Space (actions.py) 핵심 코드
- 정수 인덱스를 실제 행동으로 변환하는 매핑 구조.
- ```
def index_to_action(idx):  
    return Action(type=ActionType.ATTACK, target=idx-ATTACK_OFFSET)
```
- 역할: 정책 네트워크가 선택한 `index`를 실제 카드 행동 (Summon/ Attack/ Spell Activation 등)으로 연결.

## 2-4 ACTION MASKING

Action Masking (행동 마스킹) 도입

### 1. 필요성 및 문제 정의

유희왕 듀얼 환경은 카드 자체의 효과와 게임의 룰이라는 이중 제약을 가지므로, 현재 게임 상태(**Phase**, 필드 카드 유무, 자원)에 따라 유효한 행동이 끊임없이 변합니다. 예를 들어, 배틀 페이즈가 아닌데 공격 액션을 선택하거나, 패에 카드가 없는데 카드 발동 액션을 시도하는 것은 무효한 행동입니다. 이러한 무효한 행동을 에이전트가 탐색하도록 허용하면, 학습 데이터의 대부분이 쓸모없는 경험으로 채워져 탐색 효율(**Sample Efficiency**)이 극도로 낮아지고, 정책이 안정적으로 수렴하지 못하는 문제가 발생합니다.

### 2. 구현 및 작동 원리

이러한 비효율을 제거하기 위해 Action Masking 기술을 도입했습니다.

마스크 생성: 시뮬레이터(**simul.py**)는 매 스텝마다 현재 상태(**state**)를 기반으로 42차원의 부울(**Boolean**) 마스크를 생성합니다. 이 마스크는 유효한 액션 인덱스만 **True**로 표시하고 나머지는 **False**로 설정합니다.

신경망 통합: A2C(혹은 DDQN)와 같은 신경망이 42개의 액션에 대한 최종 출력 값 (**Q-값** 또는 **Logits**)을 계산할 때, 이 마스크를 적용합니다.

확률 강제: 마스크를 통해 무효한 액션의 출력 값을 음의 무한대에 가까운 값으로 설정합니다. 이후 **Softmax** 함수를 적용하면, 해당 무효한 액션의 최종 선택 확률은 0으로 강제됩니다.

3. 기여도 및 효과 Action Masking은 다음과 같은 핵심적인 기여를 했습니다. 탐색 공간 축소: 유효하지 않은 액션을 원천적으로 배제하여, 에이전트가 최소한의 유효한 행동(5~10개) 내에서만 정책을 학습하도록 강제했습니다.

학습 안정성 증대: 불필요한 노이즈를 제거하고 정책 업데이트의 분산(**Variance**)을 줄여, 최적 정책으로의 수렴 속도와 안정성을 크게 향상시켰습니다. 룰 기반 통합: 복잡한 게임 룰의 제약 조건을 에이전트의 정책 결정 과정에 직접 통합하여 환경과의 상호작용을 더욱 현실적으로 만들었습니다.

## ACTION MASKING

```
def get_valid_actions(game_state):
    valid = []
    if game_state.phase == "main1":
        for i, card in enumerate(game_state.player.spell_trap_zones):
            if card: valid.append(Action(ACTIVATE_TRAP, zone_index=i))
        for i, card in enumerate(game_state.player.hand):
            if card and card.card_type == "spell":
                valid.append(Action(ACTIVATE_SPELL, card_index=i))
    valid.append(Action(CHANGE_PHASE))
    return valid
```

현재 phase, 필드, 손패를 보고 둘상 가능한 행동만 모은다.  
규칙 위반 행동은 여기서부터 자동 배제됨.  
유효 액션 리스트 → 42칸 boolean mask (True=선택 가능)에이전트는 이 벡터로 “할 수 있는 것만” 고르게 된다.

-42차원 Boolean Mask 변환 — Env  
def get\_valid\_actions(self):
 actions =
 ActionSpace.get\_valid\_actions(self.simulator
 .game\_state)
 mask = np.zeros(self.action\_size,
 dtype=bool)
 for a in actions:
 mask[ActionSpace.action\_to\_index(a)]  
= True
 return mask

# 마스크 적용 — AGENT (POLICY / DQN 공통 원리)

REINFORCE/A2C 적용

```
probs = self.policy_net.get_action_probs(state, valid_mask)  
dist = Categorical(probs)  
action = dist.sample()
```

DQN 적용

```
q = self.q_network(state)[0]  
q[~valid_mask] = -np.inf  
action = q.argmax()
```

Policy 계열: softmax 전에 invalid action 확률을 0으로 설정  
DQN 계열: Q값을  $-\infty$ 로 만들어 절대 선택 불가 상태로  
만듦 결과적으로 불가능한 행동은 모델이 “아예 선택할 수 없음”

-요약하면-

ActionSpace가 룰 기반으로 가능한 행동만 먼저 필터링한다. Env가 이를 42차원 action mask로 변환해 agent에게 전달한다. Agent는 mask로 불가능한 행동의 확률/Q값을 0 또는  $-\infty$ 로 만들어 완전 차단한다.

## 2-5 REWARD (보상) 설계

- 본 프로젝트의 목표인 1턴 킬(One-Turn Kill) 퍼즐은 성공 경로가 극도로 희소하여 보상이 거의 없는 희소 보상(Sparse Reward) 문제의 전형입니다. 에이전트가 탐색을 통해 우연히 정답 시퀀스를 찾기까지는 학습이 매우 비효율적입니다.
- 이러한 문제를 해결하고 학습 속도를 높이기 위해, 도메인 지식(Yu-Gi-Oh!) 듀얼 규칙 및 퍼즐 정답 시퀀스)을 활용하여 중간 단계의 목표 달성 시 보상을 제공하는 Reward Shaping (밀집 보상, Dense Reward)을 설계하고 적용하였습니다

# Reward Shaping

## OH JAMA 퍼즐의 REWARD SHAPING

| 시퀀스 단계 / 중간 목표        | 코드 기준<br>(env_oh_jama.py 참조)                                           | 보상 (Shaped Reward)                 | 설계 의도 및 기여                                                       |
|-----------------------|------------------------------------------------------------------------|------------------------------------|------------------------------------------------------------------|
| 순차적 시퀀스 보너스           | ojama_trio_used \$\to\$ zero_gravity_used \$\to\$ \$ big_evo_pill_used | \$+10.0\$ (누적)                     | Ojama Trio - Zero Gravity - Big Evolution Pill의 정확한 발동 순서 학습 유도. |
| Ojama Token 공격 표시 전환  | token_in_attack > prev_token_in_attack 체크                              | \$+50.0 \times \text{New Tokens}\$ | Zero Gravity 발동을 통해 공격력이 높은 토큰이 공격 표시로 전환되도록 유도 (데미지의 핵심).       |
| Reflect Bounder 수비 전환 | reflect_bounder_safe 체크                                                | \$+30.0\$                          | 상대 몬스터 Reflect Bounder의 전투 반사 데미지를 회피하도록 유도 (안전성 확보).            |
| 완벽한 준비 상태             | Bounder 수비 전환 및 토큰 3개 공격 표시                                            | \$+50.0\$                          | 1턴 퀄을 위한 최종 필드 상태 달성을 강조.                                        |

## ANCIENT KINGS 퍼즐의 REWARD SHAPING

| 시퀀스 단계 / 중간 목표 | 코드 기준<br>(env_ancient_kings.py 참조)     | 보상 (Shaped Reward)      | 설계 의도 및 기여                           |
|----------------|----------------------------------------|-------------------------|--------------------------------------|
| Kuriboh 제거     | self.kuriboh_removed 체크                | \$+10.0\$               | 상대의 직접 공격 방어 카드 제거 유도.               |
| LP 회복          | current_player_lp > 100 체크             | \$+50.0\$               | Mystik Wok 발동을 통한 LP 안전 확보 및 시퀀스 진행. |
| LP 지불          | Confiscation 발동 및 LP 감소 체크             | \$+20.0\$               | LP 지불을 통해 핵심 카드를 발동하도록 유도.           |
| 순서 보너스         | 5단계 시퀀스 스텝별 self.step_n_completed 체크   | \$+10.0 \times 5\$ (누적) | 정확한 5단계 시퀀스의 순차적 학습 유도.              |
| 완벽한 준비 상태      | Total ATK \$\geq\$ 4200\$ 및 Kuriboh 제거 | \$+30.0\$               | 최종 공격 직전의 필드 세팅 완료 강조.               |

# REWARD SHAPING (보상 설계) 파트 최종 구조

- 바로 다중적인 보상 설계가 이 프로젝트의 핵심적인 기여 중 하나입니다.
- 1. 희소 보상 문제 정의 및 Reward Shaping 도입 개요

1턴 퀄 퍼즐의 희소 보상(Sparse Reward) 문제를 지적하고, 이를 해결하기 위해 도메인 지식 기반의 \*\*밀집 보상(Dense Reward)\*\*을 다중적으로 향습니다.

- Level 1: 공통 기본 밀집 보상 (Base Shaped Reward)**
  - 목표:** 모든 퍼즐에 공통적으로 적용되며, 에이전트가 유효하고 효율적인 게임 진행을 하도록 유도하는 보상 체계입니다.
  - 핵심 내용 (표 활용):**
    - 몬스터 소환/카드 발동:** 양의 보상(예: `base_reward * 0.5`)을 주어 \*\*'의미 있는 행동'\*\*을 장려.
    - 페이즈 전환/중립 행동:** 작은 음의 페널티(예: `-0.1`)를 주어 \*\*'불필요한 스텝'\*\*을 억제하고 최단 경로 탐색 유도.
    - 상대 LP 데미지:** 가장 큰 공통 보상(예: `(lp_damage / Initial LP) * 100.0`)을 부여하여 최종 목표 달성을 가속화.
- Level 2: Ancient Kings 퍼즐 특화 핵심 Shaping**
  - 목표:** Ancient Kings의 정확한 5단계 LP 시퀀스를 순차적으로 달성을하도록 유도.
  - 핵심 내용 (표 활용):** LP 회복, LP 지불, Kuriboh 제거, 순서 보너스 등을 구체적인 보상 값과 함께 설명합니다.
- Level 2: Oh Jama 퍼즐 특화 핵심 Shaping**
  - 목표:** Oh Jama의 3단계 카드 시퀀스와 공격 표시 토큰 확보, 수비 전환 등 안전 조치를 유도.
  - 핵심 내용 (표 활용):** Ojama Trio 시퀀스 보너스, 토큰 공격 표시 전환 보너스, Reflect Bounder 수비 전환 보너스 등을 설명합니다.

# 공통 REWARD SHAPING (LP 데미지 기반)

- def calculate\_base\_shaped\_reward(self, base\_reward):
  - lp\_damage = self.prev\_op\_lp - self.game.opponent.lp
  - if lp\_damage > 0:
    - return (lp\_damage / self.initial\_lp) \* 100 # LP 감소 보상
  - if base\_reward > 0:
    - return base\_reward \* 0.5 # 진행 보상
  - if base\_reward < 0:
    - return base\_reward \* 0.5 # 무효 행동 패널티 완화
  - return -0.1 # 의미 없는 행동 패널티
- LP 감소를 가장 강하게 보상해 "전투 진전"을 명확히 인식하게 한다.
  - 의미 없는 행동은 작은 패널티로 탐색을 유도하면서도 불필요한 반복을 줄인다.
  - 무효 행동 패널티를 과하게 주지 않아 학습이 꺾이지 않도록 안정성을 확보한다.

## OH JAMA 퍼즐용 REWARD SHAPING (순서 + 필드 상태)

- def calculate\_shaped\_reward(self, action, base):
  - r = self.calculate\_base\_shaped\_reward(base)
  - if action == "OjamaTrio":     r += 20
  - if action == "ZeroGravity": r += 100 if self.ojama\_used else -30
  - if action == "BigEvoPill":    r += 50 if self.zero\_grav\_used else -50
  - # Token 공격표시/ Bounder 수비표시 보상
  - if new\_token\_in\_attack:    r += 50
  - if bounder\_in\_defense:    r += 30
  - # 퍼펙트 필드(공격 토큰 3 + 수비 Bounder)
  - if perfect\_field:          r += 50
  - return r
- Ojama Trio → Zero Gravity → Big Evolution Pill 정답 순서를 강하게 유도한다.
- Token 공격표시 & Bounder 수비표시 등 필드 상태의 정확한 구성에 보상을 준다.
- 잘못된 순서나 위험한 상태는 빠르게 페널티를 줘 탐색 공간을 줄이고 수렴을 가속한다.

# ANCIENT KINGS 퍼즐용 REWARD SHAPING (5단계 정답 시퀀스)

- def calculate\_shaped\_reward(self, action, base):
- r = self.calculate\_base\_shaped\_reward(base)
- # Step 1: Pill → Megazowler
- if action == "BigEvoPill":  
•         r += 50 if megazowler\_summoned else -50
- # Step 2: Mystik Wok (LP 확보)
- if action == "MystikWok":  
•         r += 80 if pill\_done else -50
- # Step 3: Confiscation → Kuriboh 제거
- if kuriboh\_removed:  
•         r += 40
- # Step 4/ 5: Ultimate Offering 1→2 (Beast → Tyranno)
- if mad\_sword\_summoned: r += 60
- if tyranno\_summoned: r += 70
- # 완벽한 시퀀스 보너스
- if sequence\_all\_done:  
•         r += 100
- # 직접 공격 보상/ 패널티
- if action == "DirectAttack":  
•         r += 20 if kuriboh\_removed else -30
- return r

- 5단계 콤보 시퀀스(1~5단)를 따라가도록 단계별로 큰 보상을 부여한다.
- Kuriboh 제거, LP 확보, Tyranno 소환 등 승리 필수 조건 확인 시 추가 보상을 준다.
- 순서가 틀리거나 위험한 행동(직접 공격 등)에 패널티를 줘 정답 루트만 빠르게 수렴하게 한다.

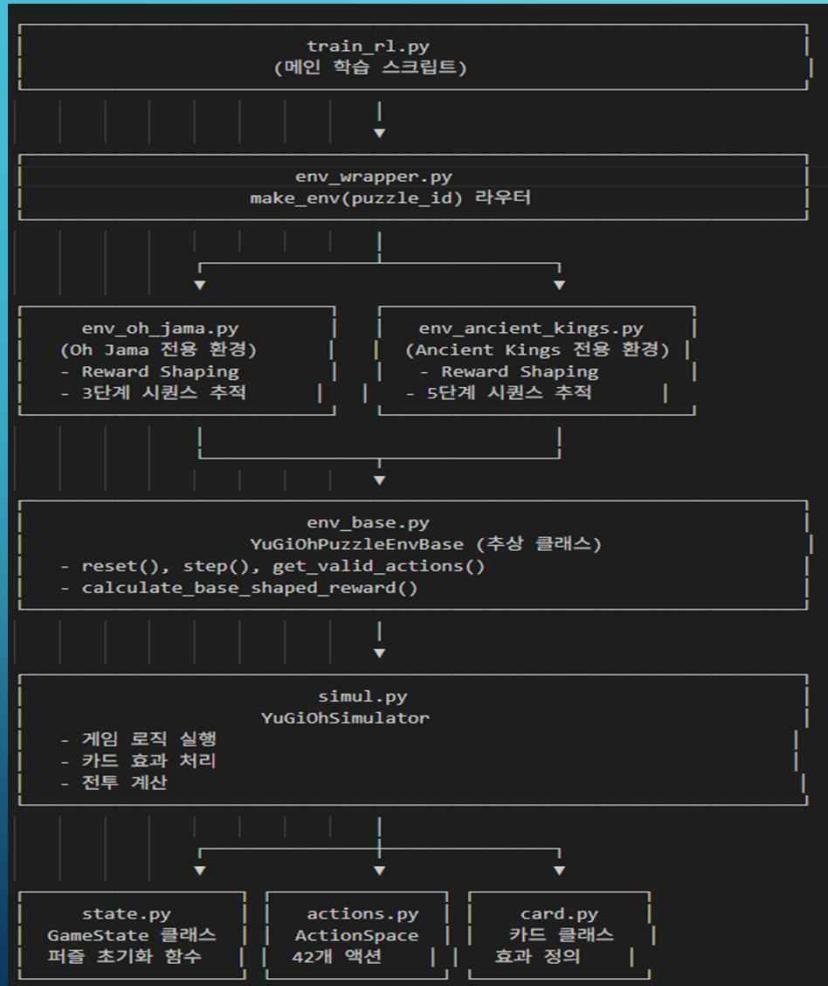
### 3. 아이디어 발전과정

- 룰&기초적인 환경의 조성 및 기초적인 agent로 학습을 진행  
(순수한 노멀 몬스터 만으로 진행→성공)
- 2차적으로 순서에 기반한 문제니까 limitation learning을 쓰자로 방향을 잡아서 해봤습니다.

→결과로는 너무 틀에 가둬둬서 하드코딩 같은 형태가 되어 버려서 실패했습니다.

- 세번째로 Deep RL을 사용해서 실행을 했다.
- Oh\_Jama는 reinforce\_baseline, Ancient\_Kings는 A2C알고리즘이 제일 정확히 풀었다.  
→결과적으로는 문제풀이 자체는 해결을 했으나 매번 실행 시 결과가 바뀌어서 해결의 결과가 애매했다.

# 3-1. 파일구조& 아키텍쳐



```
RL/
```

- Core Components (핵심 구성요소)
  - card.py # 카드 클래스 정의
  - state.py # 게임 상태 및 퍼즐 생성
  - actions.py # 액션 정의 및 ActionSpace
  - simul.py # 게임 시뮬레이터
- Environment (환경)
  - env\_base.py # 추상 기본 환경 클래스
  - env\_oh\_jama.py # Oh Jama 전용 환경
  - env\_ancient\_kings.py # Ancient Kings 전용 환경
  - env\_wrapper.py # 환경 라우터/팩토리
- Agents (에이전트)
  - dqn\_agent.py # DQN, Double DQN, Dueling DQN
  - policy\_gradient\_agent.py # REINFORCE, A2C
- Training (학습)
  - train\_rl.py # 메인 학습 스크립트
- Visualization (시각화)
  - play\_trained\_agent.py # 범용 플레이 시각화
  - play\_oh\_jama.py # Oh Jama 전용 시각화
  - play\_ancient\_kings.py # Ancient Kings 전용 시각화
- Results (결과물)
  - results/
    - best\_model\_\*.pth
    - learning\_curves\_\*.png
    - comparison\_\*.png
    - results\_\*.json

## 문제 해결의 과정

- Oh\_Jama, Ancient\_Kings 각각의 퍼즐듀얼을 limitation learning으로 진행 → 이건 성공
- 두개의 퍼즐을 전이학습으로 할 수 있는 타입으로 진화 시켜 봄 → 고정된 state에서 action이 고정이 아니라 다이나믹이란 것이 문제 였던 것을 나중에 깨달음
- 초반엔 DQN과 Double DQN로 접근 → 학습을 10000번을 시도 했지만 승률0을 기록하며, optimal policy에 도달이 불가(100000을 시도 해봐도 동일한 결과)
- 다음으로 reinforce\_baseline로 도전을 실행 → Oh\_Jama는 optimal에 도달했다. 반면, Ancient\_Kings에서 optimal에는 불가까지는 아니였으나 억지를 적용하며 되는 척을 했다. 이는 **Action Masking**을 사용해서 상당히 많이 줄였다.
- Ancient\_Kings에서는 A2C알고리즘이 4개의 에이전트중 A2C 에이전트의 승률이 잘 나왔다.

## 5. 학습 환경 과정 & 결과

- 학습환경
  - CPU : i9-13th
  - GPU : RTX4090 1장
  - RAM : 64GB
  - OS: 우분투 24.04(windows의 ws도 상과 없음)
- 
- 학습에 쓴 data(게임필드):  
[GX\_Spirit\_Caller]B03\_Oh\_Jama, [GX\_Spirit\_Caller]B04\_Ancient\_Kings

## 5-1. OH\_JAMA 퍼즐듀얼 필드(세트카드가 안보여서 2장을 넣었어요)



# 퍼즐의 풀이 비교 OH\_JAMA

## Original

- 해법 시퀀스:

### 1. Ojama Trio 발동

1. 상대 필드에 **Ojama Token** 3장 소환

### 2. Zero Gravity 발동

1. 필드 위 앞면 몬스터 전부 표시 형식 변경
2. Ojama Token들이 수비 ↔ 공격으로 전환(퍼즐 상황에선 공격 상태로 맞추는 용도)

### 3. Big Evolution Pill 발동,

1. **Black Stego**를 렐리스
2. 효과로 패에서 **Ultimate Tyranno** 특수 소환

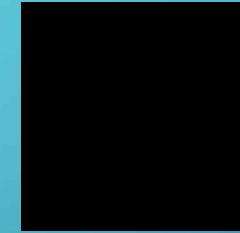
### 4. 배틀 페이즈 진입

### 5. Ultimate Tyranno로 3장의 Ojama Token 전부 공격

1. Tyranno는 몬스터를 연속으로 공격 가능
2. 토큰 3장을 전부 파괴하면서 누적 전투 데미지로 상대 LP 0 이하

## Rainforce\_Baseline

- 파일이 커서 txt파일을 올립니다.



solving.txt

**Rainforce\_baseline**의 풀이는 큰 메리트가 없는 **Reflect Bounder**를 공격하는 모션이 있는데 이는 몬스터를 없앨 때 공격 그 자체에 소량의 양의 보상(shaping)이 있기 때문이다. 즉, 알고리즘은 “승리 + 공격 성공 = 이득이면 OK”이지 “최소 행동 수 / 가장 예쁜 해법”을 목표로 하지 않는다.

## 5-2. ANCIENT\_KINGS 퍼즐듀얼 필드



# 퍼즐의 풀이 비교 ANCIENT\_KINGS

## Original A2C

- 해법 시퀀스:
  1. Big Evolution Pill 발동.
    1. 코스트로 Mammoth Graveyard를 릴리스
    2. 효과로 폐에서 Megazowler 특수 소환
  2. Mystik Wok 발동.
    1. Megazowler를 릴리스
    2. DEF 2000 선택 → LP 100 → 2100
  3. Confiscation 발동.
    1. LP 1000 지불 (2100 → 1100)
    2. 상대 폐의 유일한 카드 Kuriboh를 버린다
  4. Ultimate Offering 1회 발동.
    1. LP 500 지불 (1100 → 600)
    2. 폐에서 Mad Sword Beast 일반 소환
  5. Ultimate Offering 2회 발동.
    1. LP 500 지불 (600 → 100)
    2. (Pill의 “공룡 무제공 소환” 효과 활용)
    3. 폐에서 Ultimate Tyranno 소환
  6. 배틀 폐이즈 진입
  7. Mad Sword Beast + Ultimate Tyranno로 직접 공격
    1. Mad Sword Beast ATK 1400
    2. Ultimate Tyranno ATK 3000
    3. 합계 4400 → 상대 LP 4200을 초과, 원던 킥



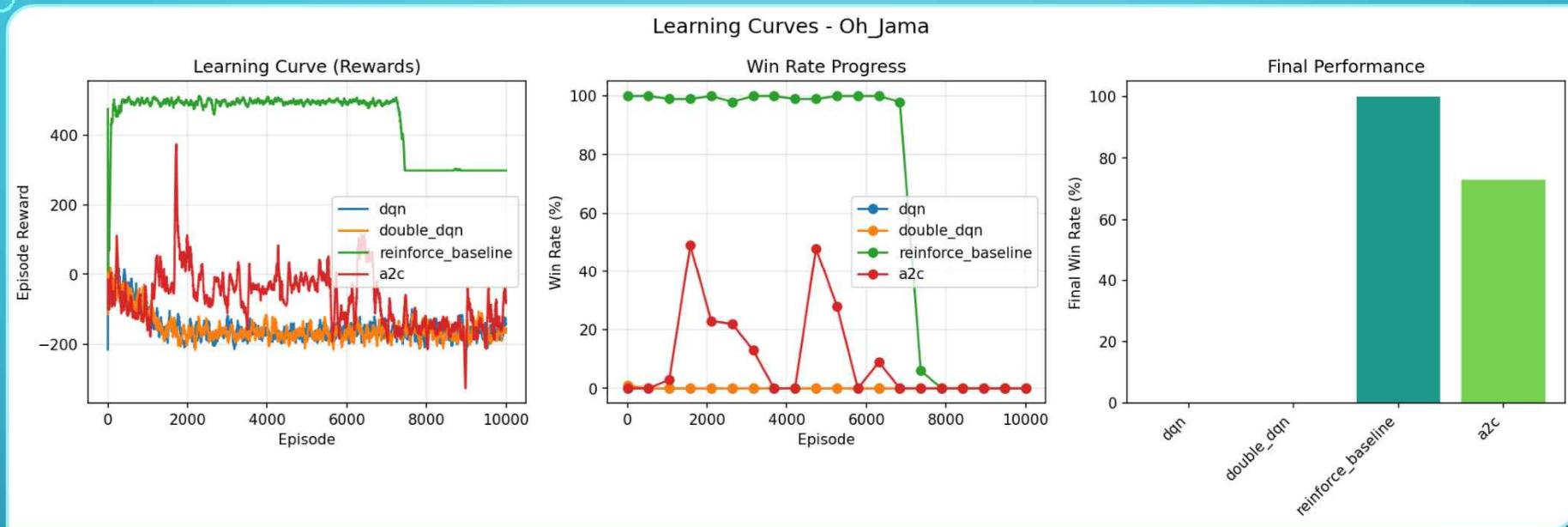
solve\_ancient.txt



잘못된 풀이.txt

- 잘못된 풀이의 경우, 초반에는 잘 했지만 중간에 무한루프에 빠진 것이 제일 큰 원인이 되었다

1. **Ultimate Offering**의 무한사용 & **Ultimate Tyranno**의 무한 공격(실제로는 이미 사용이 불가).
2. **Mad Sword Beast**의 막무가네 공격(1회 공격만 가능하게 했는데도 이렇네요.)
3. **Ultimate Offering** 사용의 부제 혹은 조건이 만족되지 않은 상태에서의 무한 사용(이 부분은 경험)



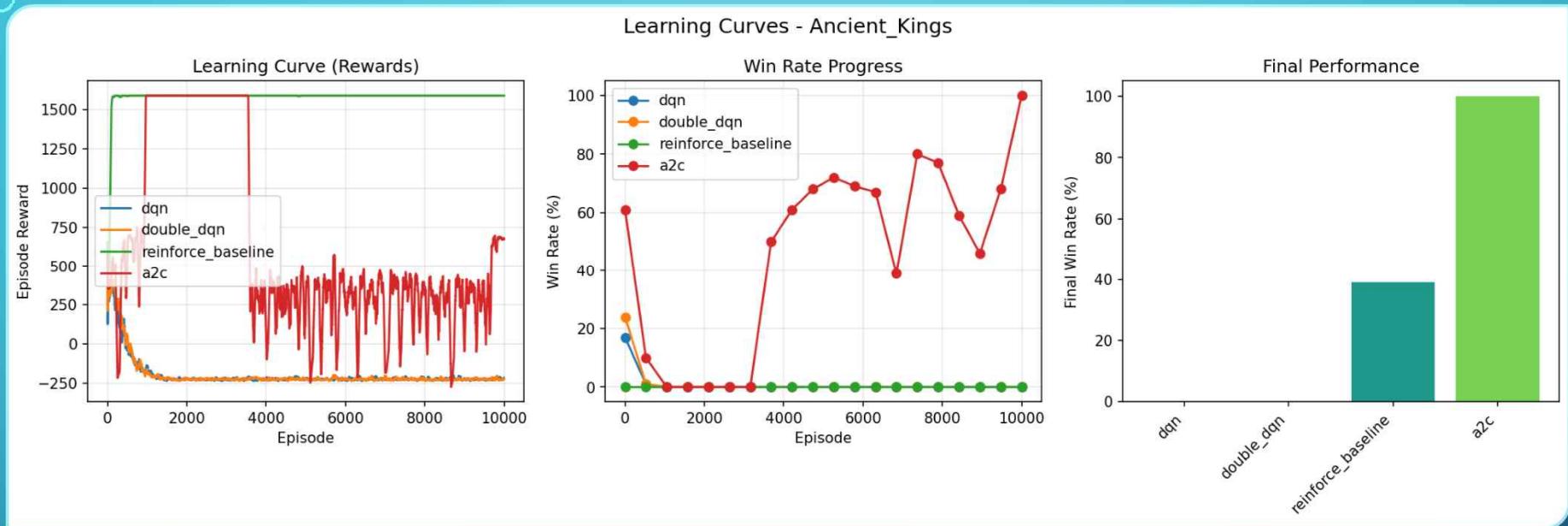
## Oh\_Jama 퍼즐을 풀 경우

Oh\_Jama의 경우에는 모든 agent중에 reinforce\_baseline과 A2C가 유일하게 성과를 내는 것을 보였다. 중간에 봉괴현상이 발생했지만 세 이브 포인트로 큰 영향을 없게 만들었다.

다른 agent들의 경우, 승률이 0되었다. 하지만 실제 퍼즐에 적용해본 결과 A2C는 억지스러운 승리를 했고, reinforce\_baseline은 룰, 효과 기반의 올바른 풀이법을 산출했다. 이를 미루어 보아 reinforce\_baseline은 optimal policy에 근접한 것으로 판단 된다.

그래서 퍼포먼스쪽에서는 100에 가까운 성능을 산출했다.

Oh\_Jama 퍼즐의 Reward Shaping 과정에서, '좋은 행동' (+10~20) 보상이 Ultimate Tyranno의 연속 공격 효과를 오직 토 큰 파괴에만 집중하도록 유도하는 데는 불완전했다. 에이전트는 승리에 도달할 수 있는 한도 내에서 불필요한 몬스터 파괴 행동을 수행했는데, 이는 최소 스텝 달성을 목표로 하는 퍼즐 듀얼 환경에서 보상 설계의 한계점을 보여준다.



## ANCIENT\_KINGS 퍼즐을 풀 경우

그래프와 같이 봉괴가 발생했으나, 제일 높이 나온 부분을 세이브 시켜두었다. 그래서 어느 정도의 승률은 보장이 됩니다. 이중에 승률은 a2c가 높았으나 실제 퍼즐에서는 둘 다 중요한 부분에서는 풀지 못 했다.

하지만 a2c에 **Action Masking**을 적용한 후 비교적 많이 올바른 풀이를 도출하여 **optimal policy**에 근접한 경향을 보인다고 판단했다. 하지만 무한루프에 빠지거나 억지스러운 승리를 만든 것 또한 사실이다. 다시 말하면, 정답은 이미 산출을 했지만 그 이후부터 연속적인 반복 에러가 발생했다.

이로인해서 개인적인 생각으로는 이런 2중 규칙의 경우, 리워드의 역할과 액션의 역할 설계가 더운 중요한 것이라 생각이 든다.

## 6. 개선점 & 차후 계획

- 현재의 개선점은 너무 명확합니다.
1. 제한된 **puzzle dule**, 환경에서만 최적화된 모델만 사용이 가능, 1개의 문제의 학습이 완료된 상태에서 다른 **puzzle dule**에서는 적용이 불가.  
→사용되는 카드들이 다르니 또 다시 학습을 해야만 한다.
  2. 카드자체의 효과, 룰 효과 2가지의 제약이 걸리니 서로 룰은 그대로 헤도 되지만 카드의 효과는 매번 변경이 필요하다.
  3. 매번 평가마다, 에이전트가 내놓는 답변이 달라서 불안정하다는 단점이 있다. 이 부분은 일반적인 **ML**과 같아서 세이브 파일만 존재하면 동일한 답변을 얻을 수 있다고 잘못 생각한 내 잘못 맞다.

## 6-1. 3번의 근본적인 문제 및 개선방안

- 특히 Policy Gradient 계열의 알고리즘에서 흔하게 관찰되는 매우 중요한 현상이므로, 보고서에 반드시 포함하여 실험 결과의 신뢰도를 분석하는 데 사용해야 합니다. 단순히 "결과가 달랐다"고 쓰는 대신, 이를 강화 학습의 특성과 연관 지어 전문적으로 기술하고 해결책을 제시함으로써 보고서의 완성도를 높일 수 있습니다.
- 높은 분산(High Variance) 문제
- 사용하신 Policy Gradient(REINFORCE, A2C) 알고리즘의 본질적인 한계를 설명하며 이 현상의 원인을 제시해야 합니다.
- "본 프로젝트에서 채택한 Policy Gradient 계열 알고리즘(REINFORCE 또는 A2C)은 Monte Carlo 추정 방식을 사용하거나(REINFORCE), Advantage 함수를 근사하는 과정에서 정책 경사(Policy Gradient) 추정치의 분산(Variance)이 높다는 근본적인 한계를 지닙니다.
- 불완전한 수렴과 미래 개선 방안
- 에이전트가 특정 퍼즐의 최적 해(Optimal Policy)에 완전히 수렴하지 못하고, 국소 최적점(Local Optima)에 머물렀을 가능성을 시사합니다. 일단 차후에 멀티시드 실험을 해보고 리워드 세이빙을 다시 손을 봐야 할거 같다.
- [보완 및 개선사항에 추가]
- 향후 연구에서는 이러한 분산을 더욱 효과적으로 줄이기 위해 GAE(Generalized Advantage Estimation)와 같은 기술을 적용하거나, 혹은 보다 분산이 낮고 안정적인 학습이 가능한 PPO(Proximal Policy Optimization) 알고리즘을 적용하여 성능의 안정성(Robustness)을 개선할 필요가 있습니다. 이 부분은 좀더 공부를 해보고 고쳐봐야 하겠습니다.

# 결론

- 본 프로젝트는 유희왕 퍼즐듀얼 환경을 대상으로 강화학습알고리즘인 **Reinforce\_Baseline**과 **A2C**를 적용해서 **Action\_Masking** → **Reward\_Shaping** → 환경 시뮬레이션 설계 → **train & evaluetion**의 전체 파이프라인을 구축을 했습니다.
- 행동 제약이 강한 카드게임의 특성에 맞춰서 **Action\_Masking**과 퍼즐정답 시퀀스를 유도하기위해 **Reward Shaping**을 구성했고, 이를 통해 **agent**가 정답을 안정적으로 재현하고 높은 보상에 수렴하게 노력했다.
- 본 과제는 퍼즐형 TCG환경에서 강화학습을 적용시 발생하는 불가능한 행동, 학습불량, 희귀한 목표달성을 같은 문제들을 구조적 설계로 해결해 보려고 노력 한 사례이며, 차후 더 복잡한 현재의 환경에 적용이 가능한 형태로 발전을 시킬 것이다.

## 프로젝트의 제작기간(프로토 타입 포함)

- 프로토 타입 구성 : 3주(일반적인 1회 공격 위주)
- Limitation Learning(알파타입) : 2, 3주(프로토에서 조금 더 추가해서 ohjama용정도로 제작)
- =====진 프로젝트=====
- 기본적인 환경구성 : 2주
- Agent's : 1.5주
- Reward : 6주

Oh\_Jama 퍼즐에 맞게 조정 → 퍼즐 자체는 단순해서 비교적 일찍 마무리 했습니다. (10번중 8~9번은 맞췄다.)

Ancient\_Kings 퍼즐에 맞게 조정 → 제일 투자를 많이 했는데 결과가 진짜 랜덤인 점이 아쉽습니다. (10번 정도 돌리면 6번 정도는 맞추는 거 같다.)