

강화학습 프로젝트

20201552 권현호

프로젝트 주제

다중 유닛 전투 환경에서 유닛의 행동 학습

- 제거 속도 극대화
- 아군 유닛 생존률 극대화
- 전투 승률 향상

환경 및 데이터셋

전장 공간: [256, 256]

유닛: [...아군 5, ...적군 5]

각 유닛은 한정된 budget에 따라 무작위 능력치를 분배받는다.

단일 agent가 아군 모든 유닛을 제어한다.

각 step마다 간단한 엔진 기반 이동 및 공격 시스템을 적용한다.

각 에피소드는 아군 또는 적군의 모든 유닛이 사망하면 종료된다.

환경 및 데이터셋

각 유닛은 한정된 budget에 따라 고유의 능력치를 가진다. 유닛은 다음 FSM에 따라 행동을 결정한다.

```
@dataclass
class Unit:
    team: int
    stats: UnitStat

    attack_target_idx: Optional[int] = None

    state: UnitState = UnitState.IDLE
    reaction_steps: int = 8

    pos: np.ndarray = field(default_factory=lambda: np.zeros(2))
    move_dir: np.ndarray = field(default_factory=lambda:
        np.zeros(2))
```

```
class StatType(Enum):
    MAX_HP = auto()
    HP = auto()
    ATTACK = auto()
    ATTACK_RANGE = auto()
    ATTACK_SPEED = auto()
    DEFENSE = auto()
    MOVE_SPEED = auto()
```

```
class UnitState(Enum):
    IDLE = auto()
    MOVE = auto()
    ATTACK = auto()
    DEAD = auto()
```

환경 및 데이터셋

각 유닛은 한정된 budget에 따라 능력치가 무작위로 분배된다.

```
def _generate_units(self, budget_per_unit=10.0):
    units = []

    for team in [0,1]:
        for i in range(ALLY_COUNT):
            w = np.random.dirichlet([1, 1, 1, 1, 1, 1])
            allocated = w * budget_per_unit

            stats = UnitStat({
                StatType.MAX_HP: 50 + allocated[0] * 10,
                ...
            })

    return units
```

state, action, reward

state observation space는 다음과 같이 생성된다.

```
def _build_obs(self):
    obs = []

    unit_vec = [
        team_flag,
        alive_flag,
        hp_ratio,
        pos_x_ratio, pos_y_ratio,
        *distances
    ]
    obs.extend(unit_vec)

    return np.array(obs, dtype=np.float32)
```

agent가 상호작용하는 전체 전장 공간 대신 유닛별 위치만 obs로 사용한다.

state, action, reward

action 행동 공간은 다음과 같다.

```
self.action_space = spaces.Dict({  
    "action_type": spaces.MultiDiscrete([3] * ALLY_COUNT),  
    "move_dir": spaces.Box(low=-1.0, high=1.0,  
shape=(ALLY_COUNT, 2), dtype=np.float32),  
    "attack_target": spaces.MultiDiscrete([NUM_UNITS] *  
ALLY_COUNT)  
})
```

stable-baseline3는 Dict action space를 받을 수 없기 때문에, 다음과 같이 flatten하여 사용한다.

```
self.action_space = spaces.Box(low=-1.0, high=1.0, shape=(20,),  
dtype=np.float32)
```

state, action, reward

reward

단기 보상을 기반으로 학습을 촉진하되,
최종 승패에 충분한 비중을 둔다.

집중 타격을 유도하기 위해
유닛이 사망했을 시 추가 보상을 제공한다.

```
curr_hp = np.array([u.stats.get(StatType.HP) for u in self.units])
hp_diff = self.prev_hp - curr_hp
```

```
for idx, u in enumerate(self.units):
    if u.team == 0: reward -= hp_diff[idx] * 0.1
    else: reward += hp_diff[idx] * 0.1
```

```
for idx, u in enumerate(self.units):
    if u.team == 1:
        if self.prev_hp[idx] > 0 >= curr_hp[idx]:
            reward += 10.0
    if u.team == 0:
        if self.prev_hp[idx] > 0 >= curr_hp[idx]:
            reward += 10.0
```

```
if all(u.state == UnitState.DEAD for u in self._enemy_units()):
    reward += 1000.0
if all(u.state == UnitState.DEAD for u in self._ally_units()):
    reward -= 1000.0
```

```
self.prev_hp = curr_hp.copy()
return reward
```

강화학습 알고리즘 및 hyperparameter

Proximal Policy Optimization 사용

더 공격적인 행동을 유도하기 위해 gamma 0.95 사용

```
def main():
    env = SubprocVecEnv([make_env() for _ in range(8)])

    model = PPO(
        "MlpPolicy",
        env,
        n_steps=2048,
        batch_size=64,
        learning_rate=3e-4,
        gamma=0.95,
        verbose=1,
    )

    model.learn(total_timesteps=1_000_000,
                callback=callback)
```

실험 셋업

실험 환경

i5-13600K

```
env = SubprocVecEnv([make_env() for _ in range(8)])
```

에피소드 10회마다 보상 로그를 수집한다.

```
callback = EpisodeLoggingCallback(
```

```
    log_dir="logs",
```

```
    record_every=10,
```

```
    verbose=1
```

```
)
```

실험 셋업

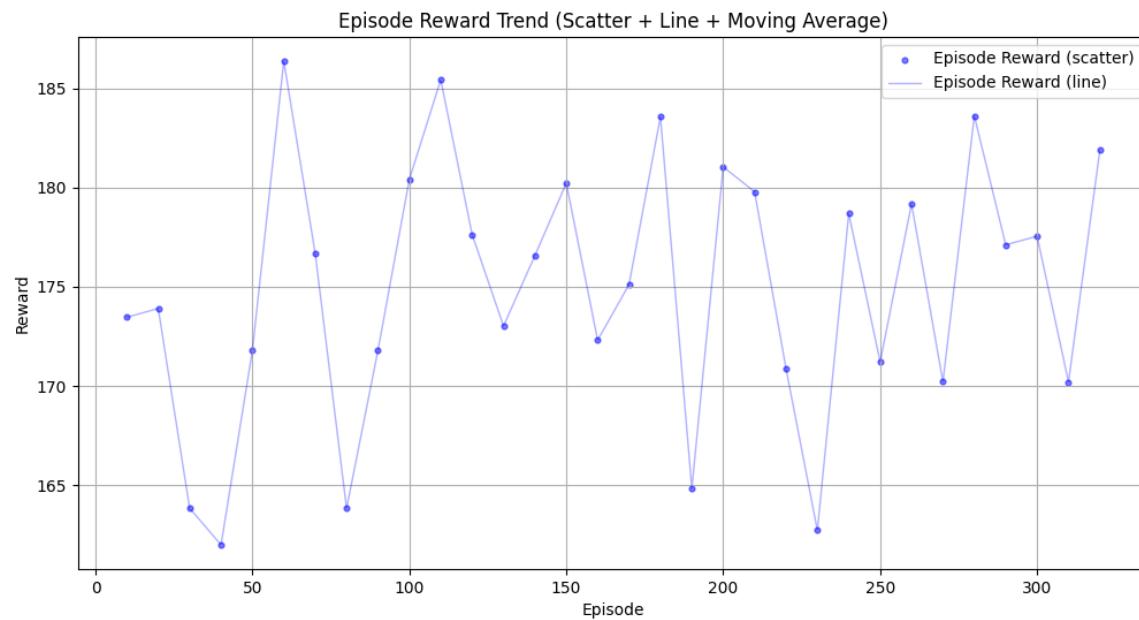
렌더링 함수를 정의하여 실제 전투 장면을 시각적으로 확인할 수 있다.

```
def render(self, mode="rgb_array"):
    # 메인 맵 캔버스
    # ----- 유닛 표시 -----
    # 팀 색상
    # ---- 사거리 표시 (연한 색) ----
    # ---- 유닛 원 ----
    # ---- 유닛 인덱스 표시 ----
    # ----- 전광판 HUD -----
    # ===== 팀 전체 체력 계산 =====
    # ===== HUD 최상단에 %만 표시 =====
    # 왼쪽: 아군 체력 % (Cyan)
    # 오른쪽: 적군 체력 % (Red)
    #
    # ----- 기존 4줄 HP 표시
    # -----
    # 줄1: 아군 현재 HP
    # 줄2: 아군 최대 HP
    # 줄3: 적군 현재 HP
    # 줄4: 적군 최대 HP
```

실험 결과

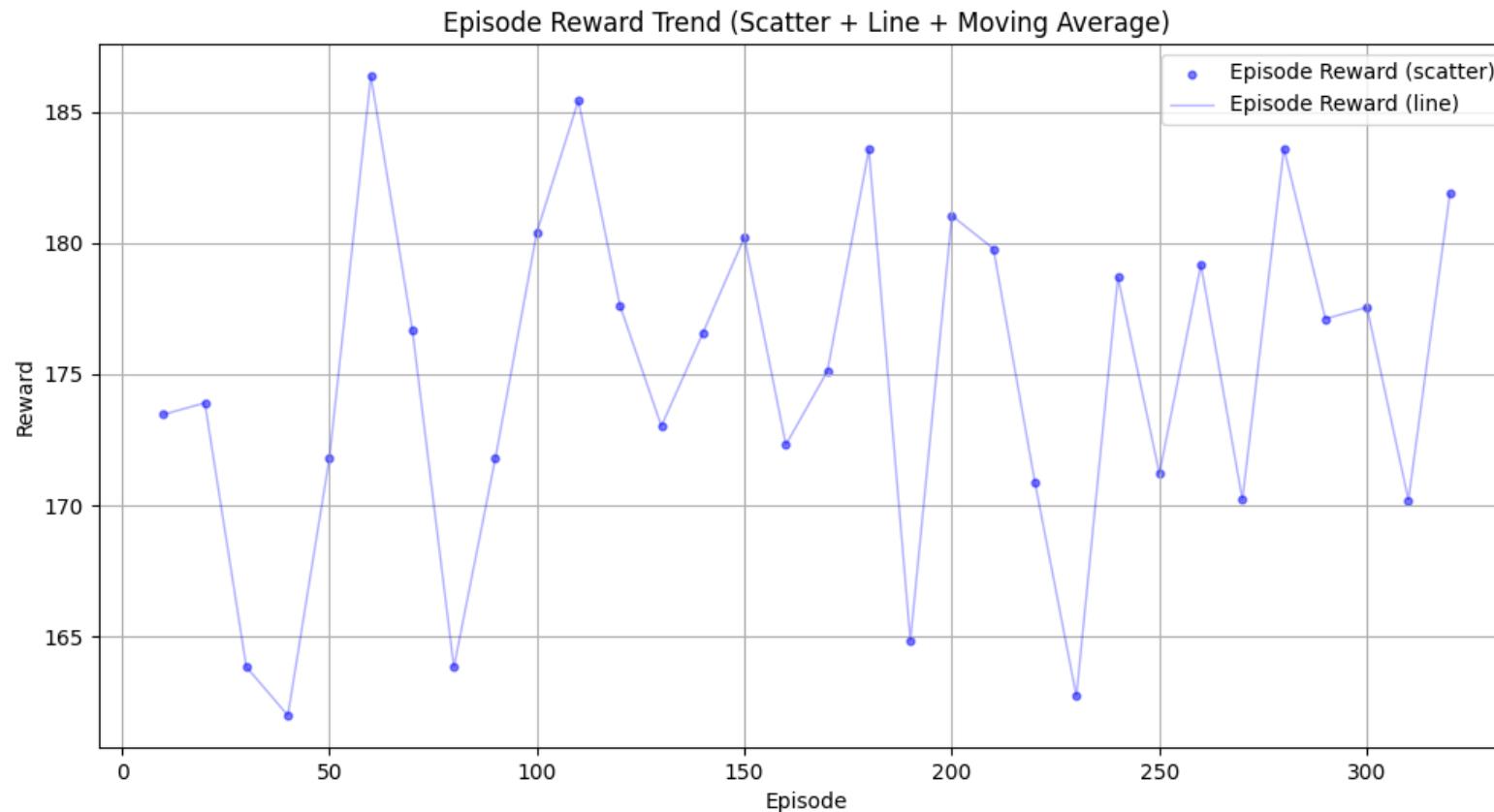
적을 빠르게 제거하도록 학습시키기 위해서 공격하지 않는 비저항 상태의 적을 생성하여 학습함.
300번 이상의 회차 후 전투가 안정됨을 확인할 수 있다.

아군을 오사하는 행동도 이 단계에서 개선되었다.



실험 결과

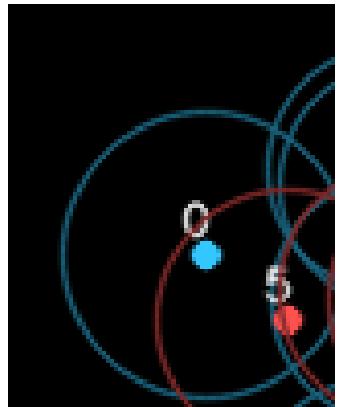
이후 공격하는 적 상대로 아군 유닛을 생존시키며 이기는 것을 학습하기 위해
미약한 적을 생성하여 학습함.



실험 결과

이후 공격하는 적 상대로 아군 유닛을 생존시키며 이기는 것을 학습하기 위해
미약한 적을 생성하여 학습함.

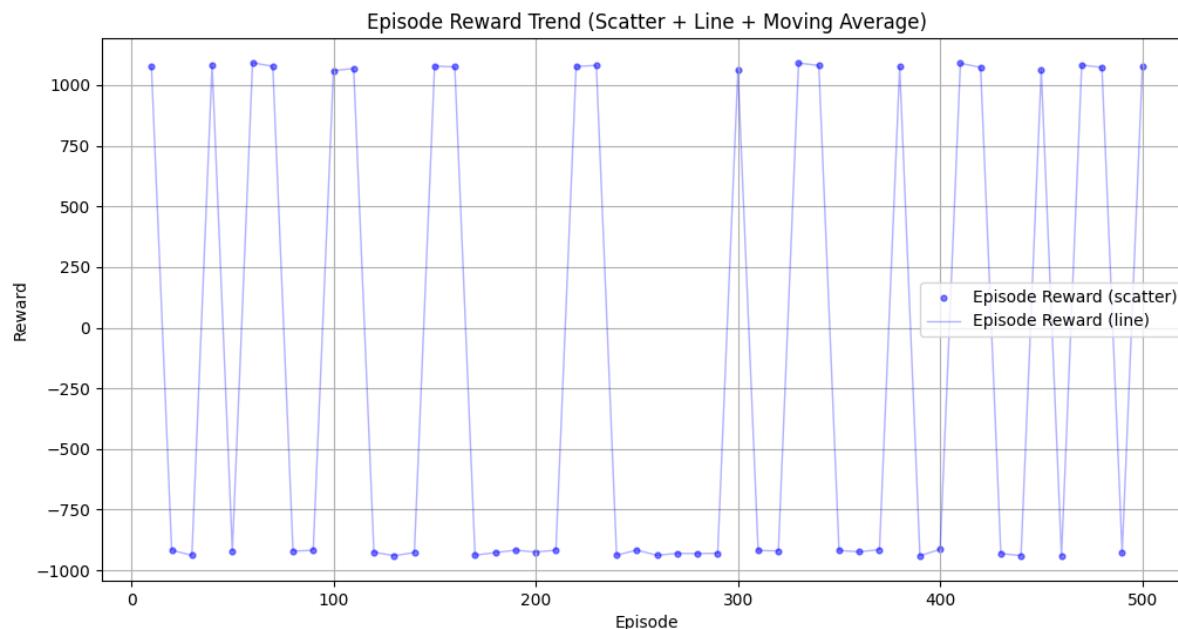
다음과 같이 상대와의 사거리를 유지하며 거리 조절(카이팅)을 하는 것을 확인할 수 있었다.



토의 및 결론

적 유닛이 많이 남지 않은 상황에서 일부 아군 유닛이 적 유닛에게 적극적으로 접근하지 않는 행동이 관찰되었다.

단기 체력 보상이 과도하게 설정되었다고 판단하여, 장기적 목표인 최종 승패에 대한 보상 비중을 상향하여 정책이 보다 안정적으로 수렴하도록 조정했다.



토의 및 결론

이번 프로젝트는 전투 환경에서 유닛의 행동을 결정하는 요소가 무엇인지 확인하는 것을 궁극적인 목표로 진행했다. 진행하면서 강화학습 기반의 AI라 하더라도 적절한 인간 피드백(human feedback)이 학습의 효율성을 크게 증대시킬 수 있다는 점을 확인할 수 있었다.

AI가 효과적으로 학습할 수 있는 명확한 판단 기준을 설계하는 것이 주요했다. 예를 들어, 아군 및 적 유닛의 위치뿐 아니라 유닛 간 거리를 직접적으로 관측 공간에 포함하는 것이 거리 조절과 관련된 행동의 학습을 유의미하게 촉진했다.

향후 프로젝트를 더 확장한다면, 기본적인 공격뿐 아니라 스킬을 추가하여 전술적 대응 범위를 넓힐 수 있다. 예를 들어, 상대 진영이 지나치게 앞으로 돌출되었거나 밀집했을 때 이를 효과적으로 응징하는 역할을 가진다.

최종적으로, 기존 덱빌딩 게임(Slay the spire, Nine Kings 등)에서 유닛을 추가하거나 강화하는 방식으로 덱을 성장시키는 구조를 넘어서, 유닛의 전투 능력 또한 플레이 방식에 따라 성장하는 기믹을 접목한다면 높은 차별점을 갖춘 게임 디자인으로 발전할 가능성이 높다고 예상된다.