이슈사항 및 해결방안

가상생태계 Environment

강화학습 알고리즘은 기본적으로 학습을 진행할 Environment가 필요하다. 따라서 우리만의 가상생태계 Environment를 구성해야했다. observation space, action space를 생태계에 맞게 설정한 후 step method를 실행하여 state, reward, done 을 반환받아 학습을 진행하도록 설정하였다.

DQN 알고리즘

DQN은 network 형식으로 state와 reward를 통해 action을 취하는 방식을 사용한다.

DQN 알고리즘의 핵심은 이전에 경험한 것을 replay buffer에 기억해 무작위로 뽑아 경험 간 상관관계 줄이는 것이다.

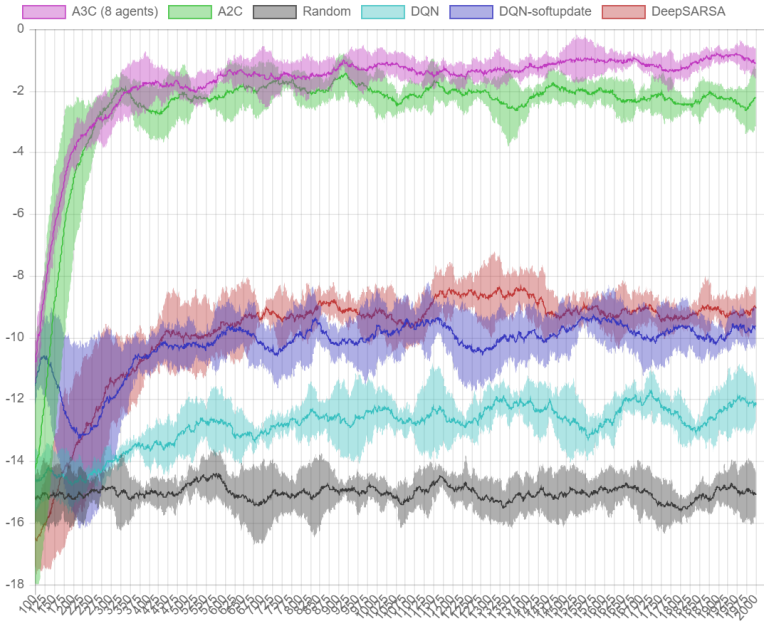
DQN 알고리즘을 통한 간단한 식 계산

- **배경 :** 시뮬레이션 코드가 완성되지 않아 작성한 Env.py가 적절한 Env 코드 인지, action space와 observation space이 DQN agent가 학습을 할 수 있는 환경인지 실험하였다.
- **실험 :** 예정된 simulation 과 비슷한 크기의 action_space(20)와 observation space(10차원 배열 * 최댓값(10))를 설정 한 후 simulation 함수 대신 간단한 식에 랜덤한 요소를 추가하여 목표치를 넘는 값을 구하는 학습을 진행하였다.

```
def simulate2(lists):
    tick = 0
    for i in range(0,10):
        tick += lists[i]*i
    tick = tick%17 * 10 + np.random.randint(0,31)- 15
    return tick
```

- **결과 :** 반복적인 학습에도 실패하는 경우가 굉장히 많았다.
- **분석 :** DQN agent는 강화학습의 방식 중 value based method 로 action space가 조금만 커져도 학습이 제대로 진행되지 않아 해당 프로젝트에는 다른 강화학습 방식을 사

용해야 한다고 판단하였다.



A2C 알고리즘

replay buffer를 사용하지 않고 on-policy방법으로 매 step마다 얻어진 상태, 행동, 보상, 다음 상태 이용해서 학습한다. Advantage Actor Critic으로 Actor nn과 Critic nn을 사용한다.

A2C 알고리즘을 통한 간단한 식 계산

- **배경 :** 시뮬레이션 코드가 완성되지 않아 작성한 Env.py가 적절한 Env 코드 인지, action space와 observation space의 크기를 A2C agent가 학습을 할 수 있는 환경인지 실험하였다.
- **실험 :** 위 실험과 같은 실험을 DQN 알고리즘 대신 A2C 알고리즘으로 대체하여 진행하였다.
- **결과 :** 랜덤한 요소가 비교적 작으면 빠르게 학습이 잘 되었으며 랜덤한 요소가 커져도 시간이 조금 더 걸렸지만 학습이 잘 되었다.
- **분석 :** Policy based method 인 만큼 20개의 action space에도 학습이 원활히 이루어지는 것을 확인 할 수 있었다.

A2C 알고리즘을 통한 시뮬레이션

Reward : if not done / sim_tick ≥ Goal / Overbound 순으로 기재

종료조건 : Goal 을 넘는 동일한 state가 6번 연속으로 등장

학습 첨부파일 :

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ca838198-7a39-469e-8e32-be17e7880f9f/강화학습실험결과.xlsx

1. 호성의 1번째 실험
    a. **Goal == 300 , Reward : sim_tick , sim_tick*10 , -1000**
        i. **결과**: 39번째 학습 이후 3번째 인자만 증가시켜보면서 300 넘으면 성공 아니면 overbound로 episode 종료를 하는 행동 반복
        ii. **분석** : Overbound 의 페널티가 너무 작아서 Overbound 로 넘어가는 것을 개의치 않고 한 개체 수만 증가 하는 것을 반복하는 것으로 예상됨
        iii. **개선방안** : Overbound 의 페널티를 증가
2. 호성의 2번째 실험
    a. **Goal = 300 , Reward : sim_tick , sim_tick * 10 , -10000**
        i. **결과:** 41번째 학습 이후 위와 동일하게 5번째 인자만 증가시켜보면서 300이 넘으면 성공 아니면 overbound로 episode 종료를 하는 행동 반복.
        ii. **분석 :** Overbound의 페널티와 별개로 Goal 을 넘지 못한 상황에 Reward 를 주었더니 episode 를 반복할 수록 가보지 않은 방향으로 가지 않고 Reward 가 누적된 길로만 가려고 하는것으로 예상된다.
        iii. **개선 방안 :** Goal을 넘지 못한 경우 페널티를 주어 새로운 방향으로 시도하게 만든다.
3. 호성의 3번째 실험
    a. **Goal = 300 , Reward : sim_tick - 299 , sim_tick - 299 , -3000**
        i. **결과** : 70번째 학습 까지 도달했다.
        ii. **분석 :** 위 학습들과 같이 overbound를 향해가는 학습을 하지는 않았고 다양한 결과가 눈에 띄었으나 12시간 정도의 학습이 진행되어도 눈에 띄는 결과가 도출 되지 않았다. 시뮬레이션 한번 한번이 너무 오래걸려 학습의 총 시간이 오래걸리는것으로 예상됨.
        iii. **개선방안 :** 시뮬레이션 코드를 최적화 하고 action space를 축소해보도록 한다.

4. 재영 1번째 실험
    a. **Goal = 300 , Reward : sim_tick/10 , sim_tick*5 , -1000**
        i. 결과 : 45번째 학습까지 도달
        ii. 분석 : 10시간의 학습을 진행했지만 실패한 경우가 너무 많았고 episode가 진행될수록 중간중간 학습된 것을 보여줬으나 성공률이 너무 적었고 최적화된 시뮬레이션 코드로 교체하기 위해 중단했다.
        iii. 개선방안 : Goal인 300tick을 달성하지 못하고 멸종한 경우에 대해 음수의 보상을 줘야 할 것 같다.

Threshold 에서 감소하는 방식으로 action space를 10으로 축소

시뮬레이션 첫번째 개선

시뮬레이션 코드를 개선하면서 전체적으로 simulation의 tick 이 줄어들어 Goal을 감소하였다.

5. 호성 4번째 실험
    a. **Goal = 150 Reward : sim_tick-149 , sim_tick-149 , -3000**
        i. **결과 :** 720번까지 학습을 진행하였으나 학습이 안됨.
        ii. **분석 :** 150초를 버티는 것이 쉽지 않은데 실패 보상이 성공 보상 보다 커 성공 보상을 trivial 하게 만드는 것으로 예상됨.
        iii. **개선방안 :** Goal에 도달하지 못한 페널티를 줄이고 성공한 케이스의 보상을 늘려 보아야겠다.
6. 호성 5번째 실험
    a. **Goal = 150 Reward : (sim_tick-149)/10 , (sim_tick-149)*10 , -3000**
        i. **결과 :** 성공률은 높아졌지만 학습이 되는 느낌은 아니다.
        ii. **개선방안 :** 멘토링 결과 action space가 증감이 모두 있어야 학습이 될 것 같다고 하여 action space를 원래대로 바꾸고 다시 돌려볼 예정이다.
7. 재영 2번째 실험
    a. **Goal = 150 Reward : (sim_tick-149)/10 , 3000 , -3000**
        i. **결과 :** 성공률이 아직도 너무 낮았다
        ii. **개선방안 :** 일정 목표를 넘으면 고정보상을 주려했으나 (6. 호성 5번째 학습)과 비교하여 목표를 훨씬 잘 넘는 것에 더 좋은 보상을 줘야하는 것을 알게 되었다.

(+action space 추가로 돌리기)

action space 20개로 복귀 , 초기 개체값은 (0~threshold) 사이의 중앙값에서 시작

8. 호성 6번째 실험
    a. **Goal = 150 Reward : (sim_tick-149)/10 , (sim_tick-149)*10 , -3000**
        i. **결과 :** episode 1000까지 도달하였으며 성공률은 매우 높지만 학습이 되지는 않는 것 같다.
        ii. **분석 :** 시뮬레이션 코드만 별도로 돌려본 결과 연산량을 줄인 시뮬레이션 코드는 똑같은 개체수를 넣고 돌려도 나오는 결과가 매우 다른 경우가 있었다.
        iii. **개선 방안 :** 시뮬레이션 코드 수정 후 다시 돌려 볼 예정
9. 재영 3번째 실험
    a. **Goal = 150 Reward : (sim_tick-149)/10 , (sim_tick-149)*10 , -3000**
        i. **결과 :** 위 학습과 동일 조건으로 실행 - episode 1000까지 도달했고 성공률은 높지만 학습은 안됨
        ii. **분석 :** 위와 동일
        iii. **개선 방안 :** 시뮬레이션 코드 수정 후 다시 돌려 볼 예정

시뮬레이션 두번째 개선

10. 호성 7번째 실험
    a. **Goal = 150 Reward : (sim_tick-149)/10 , (sim_tick-149)*10 , -3000**
        i. **결과 :** episode 약 380번 만에 학습 완료 후 종료되었다.
        ii. **분석 :** 처음으로 제대로 된 학습 결과가 도출 되었다.
        iii. **개선방안:** goal 을 변화 시키며 실험하면 될 것 같다.
11. 호성 8번째 실험
    a. **Goal = 180 Reward : (sim_tick-179)/10 , (sim_tick-179)*10 , -3000**
        i. **결과** : episode 약 720정도부터 하나의 action만을 선택함
        ii. **분석** : 해당 방향이 가장 성공률이 높게 나오는 루트인것 같다.
        iii. **개선방안** : goal 이 얼마나 떨어져있나 에 따라 이러한 현상이 일어나는것 같다. goal을 바꿔가며 실험해보면 될 것 같다.
12. 호성 9번째 실험
    a. **Goal = 160 Reward : (sim_tick-179)/10 , (sim_tick-179)*10 , -3000**

i. **결과 :** episode 약 270번째 부터 동일한 action을 취했으나 6번 연속으로 같은 값이 나오지 않아 이를 700번 정도 까지 반복함.

ii. **분석 :** goal 이 작아지니 한 action만을 취하는 현상이 더 빨리 이루어 졌다.

13. 재영 4번째 실험
    a. **Goal = 170 Reward : (sim_tick-169)/10 , (sim_tick-169)*10 , -3000**
        i. **결과** : 약 episode 290부터 동일한 action을 취하는 것을 확인할 수 있었지만 연속으로 같은 초기 개체수가 나오지 않았다.
        ii. **분석** : 160 tick에는 정확하게 학습된 것을 확인했었는데 일정하게 goal값을 버티는 개체 수가 나오지 않는 것으로 보아 개체 수가 버틸 수 있는 tick이 존재하는 것을 알 수 있었다.

14. 재영 5번째 실험
    a. **Goal = 170 Reward : (sim_tick-169)/10 , (sim_tick-169)*10 , -3000**
    b. **threshold 값 변화 /** [40, 150, 100, 90, 40, 90, 120, 100, 70, 1500]
        i. **결과** : episode 392부터 같은 action을 선택하고 episode 469번에서 종료조건을 만족해 학습이 완료되었다.
        ii. **분석** : 다른 개체 수를 사용하더라도 학습이 잘 되는 것을 확인할 수 있다.

시뮬레이션 코드의 개선

1. 동물 사이의 상호작용
    a. 육식동물 & 초식동물의 Class를 만들어서 동물의 특성에 따라 상속하도록 했다.
    b. 동물들은 기본적으로 Grid 위에서 활동한다. (2차원 좌표계 사용)
    c. 동물들은 상호작용 및 동작을 수행하기 위해 다음과 같은 특성들을 가진다.

    ▼ 동물의 특성들

        1. x, y 좌표
        2. 동물의 남은 에너지 (칼로리)
        3. 동물의 남은 수명 (자연사)

4. 동물이 먹혔을 때 포식자가 얻는 에너지 (칼로리)
5. 동물의 시야 범위
6. 동물의 사냥 성공 확률
7. 동물의 포식자 List
8. 동물의 먹이 List
9. 동물의 tick 당 칼로리 소모량
10. 최대, 최소 수명
11. 가지고 있을 수 있는 최대 칼로리
12. 동물의 번식 확률

d. 동물들은 기본적으로 자기의 시야 범위 내의 정보로 다음 행동을 결정하는데
   i. 포식자가 가장 가까이 있는 경우
      - 포식자의 사냥 확률의 따라서 발각 되면 포식자의 반대 방향에 존재하는 빈칸으로 도주
   ii. 먹잇감이 가장 가까이 있는 경우
      - 먹잇감의 방향에 존재하는 빈칸으로 이동
      - 먹잇감이 한칸 이내에 있으면 먹고 칼로리를 획득한다.
   iii. 둘 다 아닌 경우
      - 주변에 있는 가장 가까운 랜덤한 빈칸으로 이동
   iv. 이동할 수 있는 칸이 없는 경우
      - 제자리에서 대기
e. 이렇게 행동을 진행하고 나서 각자 tick 마다 정해진 칼로리를 소모한다.
f. 동물 입장에서 충분한 칼로리를 가지고 있다면 확률적으로 번식하고 칼로리를 잃는다.

2. 초기 시뮬레이션
   - 매 행동마다 시야 내를 검색하고, 이동하는 방향을 결정하고, 빈칸이 존재하는지 탐색했다.
   - Problem

    - 시뮬레이션 시간이 너무 오래 걸린다.
    - Grid로 각 동물들의 위치를 겹치지 않게 유지하기 위한 동물들의 충돌 검사(같은 Grid에 2마리 이상의 동물이 존재하지 않도록)이 복잡하다.
    - 풀도 시뮬레이션에서 하나의 동물 Class로 취급하는데, 풀 위에 다른 동물이 존재하지 못 한다는 건 자연스럽지 못하다.
    - 100*100 Grid에서 진행하지만, 동물 입장에서는 경계가 느껴지지 않는 것처럼 활동할 수 있게 자연스럽게 행동할 수 있도록 하고자 했다.

3. 개선 방안
    - 시야를 미리 검색하고, 포식자 정보, 먹이 정보, 빈칸 정보를 저장해두고 이를 활용해서 각 동물이 행동하는데 걸리는 연산 비용을 대략 절반 가까이로 내리도록
    - Grid의 충돌검사를 좀 더 정밀하게 수행하여, 한 Grid위에 2마리의 동물이 존재하지 못하도록 했다.
    - 풀의 경우, 따로 Grid_Grass를 따로 생성하여 관리하도록 했다.
    - 동물들의 탐지 및 이동 범위를 파이썬 list의 음수 인덱스를 이용하여 경계가 없는 것처럼 행동하도록 개선했다.
    - 동물들의 이동을 사분면 단위로 통제하여 연산량을 줄이려고 시도했다.
4. 개선된 시뮬레이션의 Problem
    - 시뮬레이션 한번에 드는 시간이 감소하기를 바랬지만, 피식 동물들이 너무 똑똑해져서 번식을 너무 잘하는 문제가 생겼다. 그에 따라 초식동물들의 수가 증가했고, 동물들의 수가 많아짐에 따라서 시뮬레이션 한번에 드는 시간이 늘어나는 문제가 생겼다.
    - 또, 초기에 랜덤하게 행동한 동물들의 결과의 차이에 따라 생태계의 유지 tick이 크게 달라지는 문제가 생겼다.
5. 결론(두번째 개선)
    - 결국 이전의 초기 시뮬레이션과 시야 검색이 개선된 시뮬레이션을 각각 사용해서 학습을 진행해보고, 성과를 비교했는데, 초기 시뮬레이션이 더 적절하다는 팀원들의 평가에 따라서 다시 초기 시뮬레이션을 채택했다.
    - 아쉽게도 어느정도 일정한 값을 제공하는 시뮬레이션을 만드는데 생각보다 오랜 시간이 들었고, 이는 로블록스 개발에도 영향을 끼쳤다.

로블록스

1. 계획의 변경
    - 로블록스 내부에 동물들의 시뮬레이션을 구현하여, FastAPI 혹은 Flask 웹서버를 통하여 강화학습 알고리즘과 데이터를 주고 받아 실시간으로 학습하는 모습을 보여준다.
    - 한계: 로블록스는 1분의 500회의 API 호출만을 지원하고, 무겁고 느린 플랫폼이기 때문에 로블록스에서 시뮬레이션을 진행하여 강화학습을 하는 데는 너무 오랜 시간이 걸린다는 결론이 나와서, 강화학습 프로젝트 내부에 시뮬레이션 코드를 작성해서 시뮬레이션을 수행하도록 했다.
    - 다음 계획: 로블록스에서 특성 값의 입력을 사용자로부터 받아서 해당 특성 값에 대해 학습하여, 적절한 초기값으로 구성된 생태걔를 로블록스에 시각화 시킨다.
    - 한계: 특성 값의 입력을 받고, 안정적인 초기값을 학습하는데 오랜 시간이 걸려서 로블록스가 대기하는 시간이 너무 길어진다.
    - 대안: 특성 값을 임의로 정하고, 강화학습을 진행한 뒤 해당 개체수를 로블록스에서 시뮬레이션 하는 걸로 제시
2. 현재 상태
    - 동물들 사이에 상호작용 및 시뮬레이션을 시각화하는건 성공했지만, 더 자연스러운 배경 혹은 정밀한 시뮬레이션 작성에는 도달하지 못 한 것 같아서 아쉽다.
    - coroutine 등을 이용해서 한번에 여러 동물들이 상호작용하는 모습을 원했으나, 그 기능까지는 도달하지 못했다.
3. Animal Model
    - 로블록스 스튜디오에서 지원하는 ToolBox에서 10종의 모델들을 사용
4. Problem
    - 로블록스 스튜디오의 무거운 환경
    - coroutine을 통한 자연스러운 동물들의 행동 구현 실패
    - 기간을 고려하지 못해 미처 다 구현하지 못한 부분들