인공지능

[Project#2]

Class : **[수 2]**

Professor : **[최상호 교수님]**

Student ID : **[2020202060]**

Name : **[홍왕기]**

Introduction

이번 프로젝트는 Dynamic Programming(DP)을 활용하여 주어진 grid world 환경에서 최적의 정책(policy)을 도출하는 것을 목표로 한다. DP 는 복잡한 문제를 작은 단위로 분할하여 점진적으로 해결하는 방법론으로, 여기서는 7x7 크기의 그리드 환경에서 Policy Iteration 과 Value Iteration 을 통해 최적 경로를 학습하고, 보상(reward)을 최대화하는 정책을 찾아내는 과제를 다룬다.

그리드 환경은 상하좌우 네 방향으로의 이동이 가능하며, 가장자리를 넘어서는 행동은 금지된다. 또한 그리드 내부에는 장애물과 함정이 위치하여 경로 탐색에 변수를 제공하며, 이러한 장애물은 에이전트의 경로 선택에 영향을 미친다. 보상 체계는 초기 설정으로 종료 지점에서 0, 장애물 또는 함정에서 -100, 그 외의 상태에서 -1 의 값을 가진다. 초기 정책은 상하좌우 네 방향에 대해 동일한 확률을 가지는 균일 무작위 정책(uniform random policy)으로 설정되며, 이는 특정 방향에 대한 편향 없이 모든 방향을 동일한 확률로 선택하는 방식이다.

이 프로젝트는 크게 세 단계로 진행된다. 첫 번째는 Policy Evaluation 단계로, 현재 주어진 정책에 따라 각 상태의 가치(value function)를 반복적으로 계산하여 수렴된 값을 도출한다. 이때, 에이전트는 주어진 정책에 기반해 보상을 누적하며 상태 가치의 변화를 관찰하고, 이를 통해 에이전트가 현재 정책을 따를 때 기대되는 총 보상을 파악할 수 있다. 두 번째는 Policy Improvement 단계로, 앞서 계산된 가치 함수를 바탕으로 정책을 개선하여 더 높은 보상을 얻을 수 있는 행동을 선택하도록 한다. 이는 greedy 정책 개선 방식을 이용해 각 상태에서 기대 보상이 최대가 되는 방향으로 정책을 업데이트하며 최적의 정책으로 점진적으로 나아간다. 마지막으로, Value Iteration 단계에서는 Bellman Optimality Equation 을 활용하여 각 상태에서 가능한 모든 행동의 보상을 비교하고, 가장 높은 보상을 얻는 방향으로 상태 가치를 업데이트해 나간다. 이 과정에서 정책을 따로 저장하지 않고 각 상태의 가치를 최대화하는 방향으로 직접 계산해 나가며 최적 정책을 찾아간다.

최종적으로, 프로젝트는 에이전트가 수렴된 최적의 정책에 따라 종료 지점까지 도달하는 경로와 이 과정에서의 보상을 시각적으로 확인하고 분석하는 것을 목표로 한다. 반복적인 정책 평가와 개선을 통해 Dynamic Programming 이 최적 경로를 학습하고 장애물을 피하며 목표 지점까지 이동하도록 효과적으로 작동함을 증명하며, 에이전트가 학습한 최적 경로와 정책을 평가하여 DP 의 유용성을 확인한다.

Policy iteration, value iteration 설명

Policy Iteration 과 Value Iteration 은 Dynamic Programming 의 핵심 알고리즘으로, 강화 학습에서 최적의 정책을 도출하기 위한 방법이다. 이 두 알고리즘은 환경의 모델(상태 전이 확률과 보상 함수)을 활용하여 최적의 상태 가치(value function)와 정책(policy)을 계산하며, Bellman Equation 을 기반으로 한다. Policy Iteration 은 정책 평가와 정책 개선을 반복하여 최적의 정책을 찾는 방식이고, Value Iteration 은 상태 가치를 최대화하는 방식으로 최적의 정책을 계산한다.

Policy Iteration 은 현재 정책의 상태 가치를 평가한 뒤, 이를 기반으로 더 나은 정책을 도출하는 과정을 반복한다. 첫 번째 단계인 Policy Evaluation 에서는 주어진 정책이 각 상태에서 얼마나 좋은지 계산한다. 초기 상태 가치를 임의의 값으로 설정하고 Bellman Expectation Equation 을 반복적으로 적용하여 상태 가치가 수렴할 때까지 갱신한다. 이 과정에서 각 상태의 가치는 해당 상태에서 정책에 따라 행동했을 때 얻을 수 있는 보상의 기대값으로 정의된다. 정책 평가가 완료되면, 두 번째 단계인 Policy Improvement 로 넘어가게 된다. 여기서 현재 상태 가치 함수에 기반하여 각 상태에서 기대 보상이 최대화되는 행동을 선택하고 정책을 업데이트한다. 이는 각 상태에서 가장 높은 보상을 가져오는 방향으로 정책을 수정하며, 새로운 정책이 이전 정책과 동일해질 때까지 반복한다. 이 과정을 통해 점진적으로 최적의 정책을 도출한다.

Value Iteration 은 Bellman Optimality Equation 을 활용하여 상태 가치를 직접 최적화하는 방식으로 동작한다. 초기 상태 가치를 임의로 설정한 뒤, 각 상태에서 가능한 모든 행동에 대해 기대 보상을 계산하고 최대 값을 선택하여 상태 가치를 갱신한다. Bellman Optimality Equation 은 특정 상태에서 가능한 행동 중 가장 높은 보상을 가져오는 행동을 탐색하는 데 사용되며, 이를 통해 정책을 따로 저장하지 않고도 상태 가치를 직접 계산할 수 있다. 상태 가치가 수렴하면, 각 상태에서 최대 가치를 주는 행동을 선택하여 최적의 정책을 도출한다. Value Iteration 은 상태 가치와 정책을 동시에 최적화하는 특징이 있어, 정책 평가와 개선이 분리된 Policy Iteration 에 비해 단순한 구조를 가진다.

두 알고리즘은 모두 보상과 상태 전이 확률을 활용하여 최적의 정책을 계산하지만, 접근 방식에서 차이가 있다. Policy Iteration 은 정책을 평가하고 개선하는 단계를 반복적으로 수행하며, 명시적으로 정책을 저장하고 갱신한다. 반면, Value Iteration 은 상태 가치를 직접 최적화하며, 정책을 별도로 저장하지 않고 상태 가치로부터 최적 행동을 계산한다. 결과적으로 두 알고리즘 모두 환경에 대한 충분한 정보가 주어진 상황에서 최적의 정책을 보장하며, 강화 학습의 기초를 이루는 중요한 방법론이다.

Algorithm(동작 과정과 구현 방법)

먼저, 상수 ACTION_UP, ACTION_DOWN, ACTION_LEFT, ACTION_RIGHT 는 에이전트의 네 가지 행동을 정의하고, 가능한 행동 집합인 ACTIONS 리스트와 행동을 상징하는 화살표를 매핑한 ACTION_SYMBOLS 를 설정하여 에이전트가 grid world 에서 상하좌우 방향으로 이동할 수 있도록 했다. 이후 설명할 각 함수는 이 상수를 활용해 상태 변이와 정책을 관리하는 것이다.

get_available_actions 함수 구현에서는 현재 상태에서 이동 가능한 방향을 반환하는 역할을 하도록 했다. 이 함수는 state 와 grid_reward 를 입력받아 해당 위치에서 상하좌우로 이동할 수 있는지를 검사하고, 이동 가능한 모든 방향을 리스트 형태로 반환하는 형식으로 구성하였다. 예를 들어 x > 0 일 때 위로 이동할 수 있다고 판단하여 ACTION_UP 을 actions 리스트에 추가하는 식으로, 가능한 모든 방향을 확인하여 반환하도록 했다.

get_transition 함수는 주어진 상태와 선택된 행동을 기반으로 에이전트가 이동하게 될 다음 상태와 해당 상태에서의 보상을 반환한다. 함수는 입력받은 현재 상태 state 와 행동 action 에 따라 가능한 방향으로 이동하며, 이동할 수 없는 경우 현재 상태를 유지하도록 하였다. 이후 이동한 위치를 next_state 로 설정하고, 보상은 grid_reward 에서 현재 상태의 보상 값을 가져와 반환하게 된다.

policy_generator 함수는 초기 정책을 설정하는 역할을 하며, 입력된 israndom 값에 따라 무작위 정책을 생성하거나 모든 가능한 행동을 포함한 정책을 생성하도록 한다. 무작위 정책을 설정할 때는 get_available_actions 함수에서 반환된 행동 중 하나를 무작위로 선택하고, 그렇지 않은 경우 가능한 모든 행동을 포함하는 방식으로 초기 정책을 구성하여 반환하도록 구현하였다.

policy_evaluation 함수는 주어진 정책에 따라 각 상태의 가치를 평가하고, 가치 함수 V 를 업데이트한다. 이 함수는 각 상태가 현재 정책을 따를 때 기대되는 총 보상을 계산하여 반환하며, 내부의 compute_value 보조 함수는 특정 상태와 가능한 행동들에 대해 가치 함수를 계산하는 역할을 한다. 각 행동으로 이동했을 때의 보상을 계산하고, 모든 가능한 행동에 대해 평균을 구해 해당 상태의 가치를 계산한다. update_value_if_needed 함수는 각 상태의 가치를 조건에 맞게 업데이트하며, 목표 상태에서는 가치 함수를 변경하지 않는다. policy_evaluation 메인 함수는 전체 상태에 대해 가치 함수를 반복적으로 업데이트하여 상태 가치의 변화가 특정 임계값 theta 이하로 떨어질 때까지 반복하고, 수렴된 가치 함수를 최종 반환하도록 구현했다.

policy_improvement 함수는 policy_evaluation 에서 계산된 가치 함수 V 를 기반으로 정책을 개선한다. 이 함수는 각 상태에서 가능한 행동 중 가장 높은 보상을 기대할 수 있는 행동을 선택하여 새로운 정책을 생성하며, 목표 지점에서는 정책을 설정하지 않고 None 으로 둔다. 이후 get_available_actions 와 get_transition 함수를 통해 가능한 행동을 평가하여 최적의 행동을 찾고, 개선된 정책을 반환하도록 구현했다.

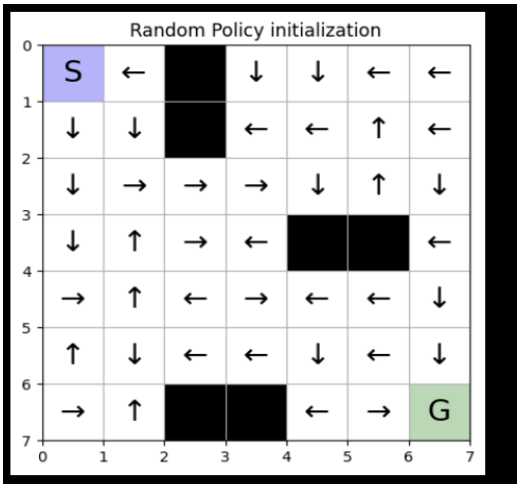
policy_iteration 함수는 Policy Iteration 알고리즘의 전체 흐름을 실행한다. 초기 정책을 policy_generator 로 무작위 생성하고, 수렴할 때까지 정책 평가와 정책 개선 단계를 반복하여 최적의 정책을 도출한다. 각 반복마다 policy_evaluation 을 통해 가치 함수를 평가하고, policy_improvement 를 통해 새로운 정책을 계산하여 갱신해 나간다. 이 과정에서 각 반복 단계의 가치 함수와 정책을 시각화하며, 정책이 수렴하거나 최대 반복 횟수에 도달하면 최종 정책과 가치 함수를 반환하여 구현했다.

마지막으로, value_iteration 함수는 Bellman Optimality Equation 을 활용하여 상태 가치를 최대화하는 방식으로 최적의 정책을 찾는다. 이 함수는 초기 가치 함수 V 와 정책을 grid 크기에 맞춰 초기화한 뒤, 각 상태에서 가능한 모든 행동에 대해 보상을 계산하여 가장 높은 보상을 기대할 수 있는 방향으로 상태 가치를 갱신한다. get_transition 을 호출해 이동할 다음 상태와 보상을 계산하고, 상태 가치의 변화량이 theta 이하가 될 때까지 반복하여, 최종 수렴된 가치 함수와 최적의 정책을 반환했다.

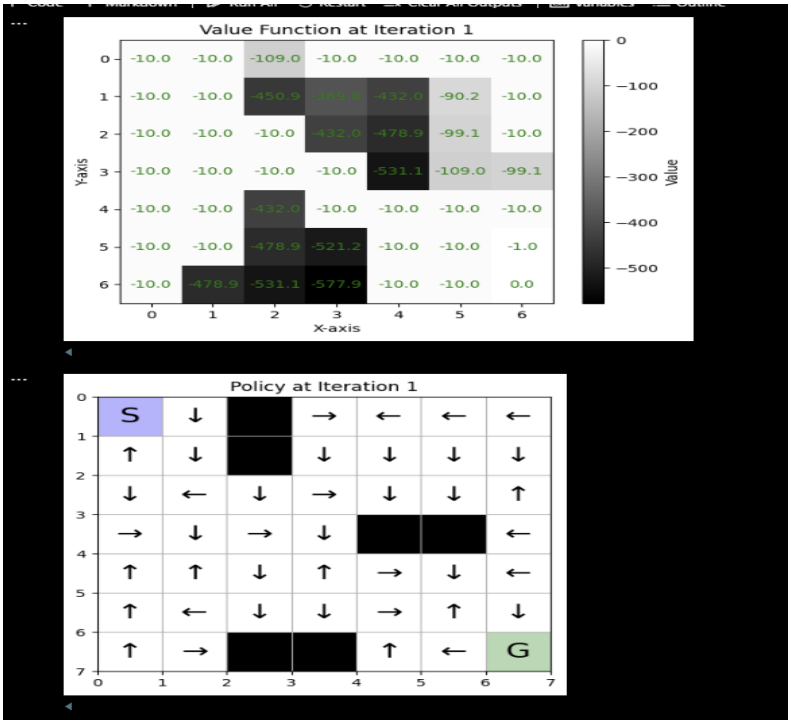
결과화면

Policy iteration 결과 단계별 캡처

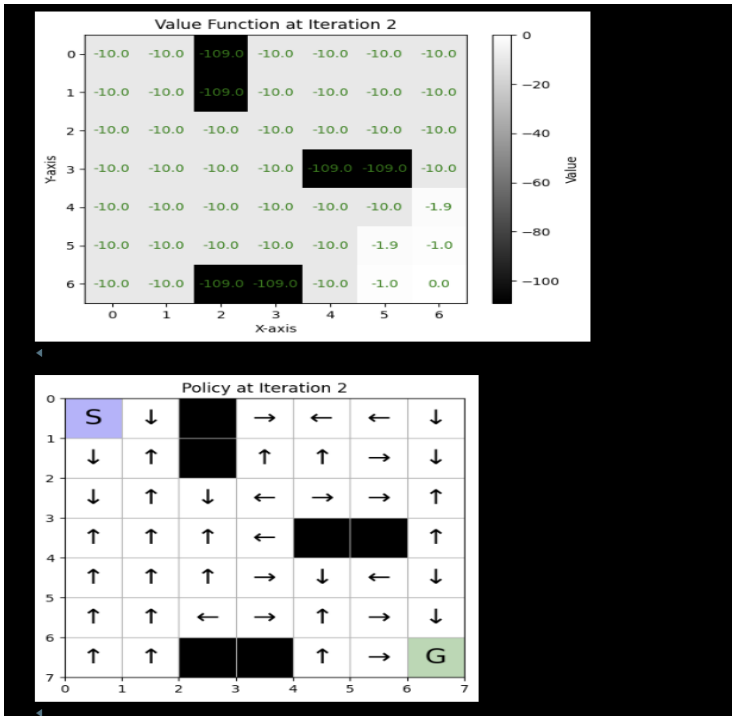
이 결과를 출력하면 다음과 같다.



먼저 랜덤한 방향에 대한 정책을 세우면 위와 같이 모두 랜덤한 방향이 출력되는 것을 볼 수 있다 이를 한번 평가하고 학습시킨다면



장애물이 있는 셀과 그 주변 셀들은 가치가 매우 낮으며, 특히 장애물 자체는 큰 음수 값을 가진다. 일반 셀들은 -10 에 가까운 값을 가지지만, 목표에 가까워질수록 가치가 조금씩 높아질 것이다. 하지만 반복이 첫번째 이므로 아직은 비슷한 것을 볼 수 있다. 이는 장애물과 목표 지점을 고려해 초기 정책을 기반으로 한 기대 가치가 반영된 것이다.

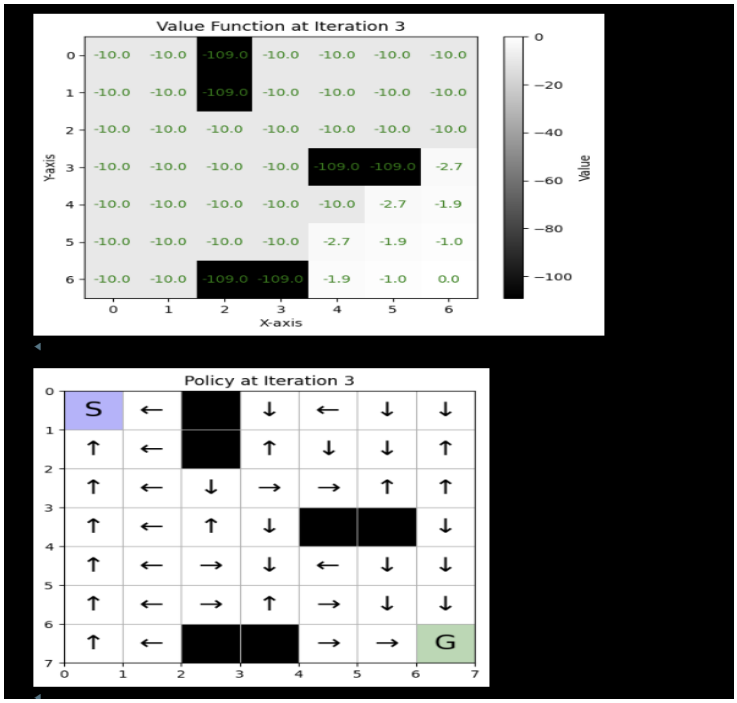


두 번째 반복은 첫 번째 반복과 비교해 가치 함수와 정책에 몇 가지 변화가 생겼으며 이 변화는 가치가 안정화되었음을 확인할 수 있다.

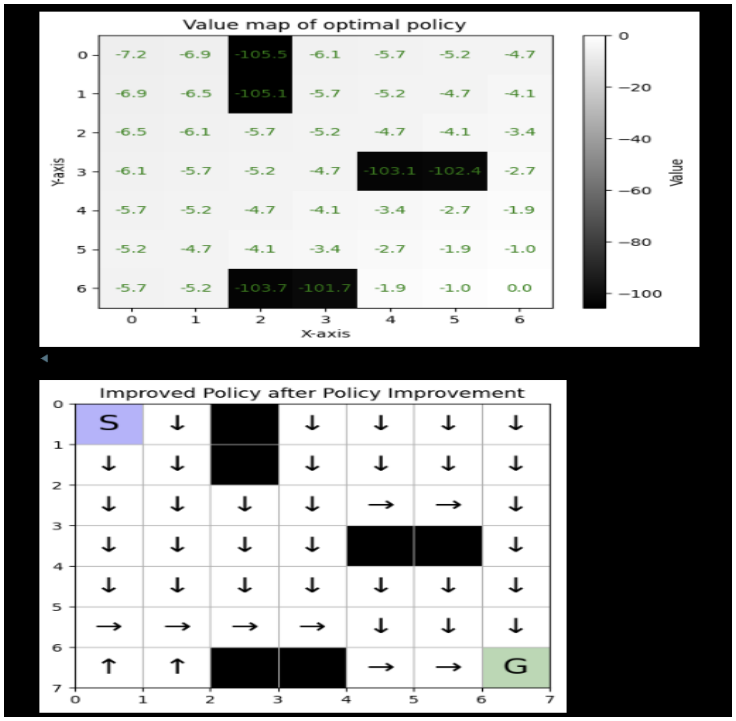
첫 번째 그래프의 가치 함수(Value Function)에서는 장애물 주위의 가치가 -109.0 으로 낮게 유지되며, 대부분의 일반 셀은 -10.0 의 가치를 그대로 가지고 있다. 목표 지점(G) 주변 셀은 0 에 가까운 값으로, 목표에 가까워질수록 점차적으로 가치가 높아지는 경향을 보이는 것을 볼 수 있다. 첫 번 째와 달리 두 번째 반복 후 가치가 안정적으로 유지되는 이유는, 정책 평가 단계에서 가치가 수렴하면서 목표 지점과 장애물에 대한 보상이 이미 반영되었기 때문이다. 이로 인해, 각 셀의 기대 가치가 장애물과 목표 지점의 영향을 받아 어느 정도 고정되었음을 볼 수 있다.

두 번째 그래프의 정책(Policy) 역시 첫 번째 반복과 달라진 부분을 볼 수 있다. 장애물 주변 셀들의 방향이 고정되면서 장애물을 피하는 경로가 형성되었고, 목표 지점을 향해 이동하는 방향이 점차 명확한 것으 볼 수 있다. 예를 들어, 목표 지점 근처의 셀들은 오른쪽으로 이동하고 시작 지점(S)에서 목표로 향하는 방향성이 형성되었다. 이는 두 번째 반복 후 정책이 어느 정도 안정화되었음을 보여준다.

위와 같은 방식으로 연속한 반복으로 인해 3 번째와 점차 수렴할 때까지 결과를 확인하면 다음과 같다.



(수렴할 때까지 반복)



(위는 수렴 결과)

위와 같은 방법을 통해 수렴할 때까지 진행된다면 위 그림 과 같은 결과를 얻을 수 있다

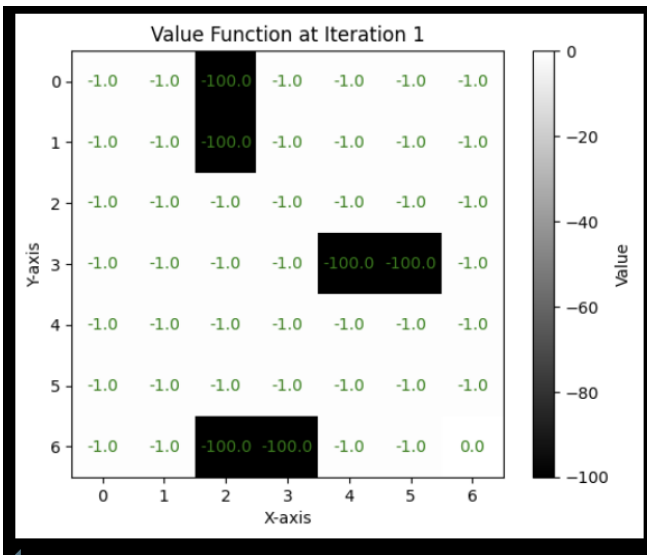
첫 번째 그래프의 가치 함수에서는 모든 셀이 안정된 값으로 수렴했음을 확인할 수 있다. 장애물 셀들은 약 -105 의 큰 음수 값을 가지며, 이는 장애물을 통과할 때 발생하는 큰 손실을 반영한 결과이다. 장애물 인근 셀들 역시 그 영향을 받아 낮은 가치를 가지며, 에이전트가 장애물 주위를

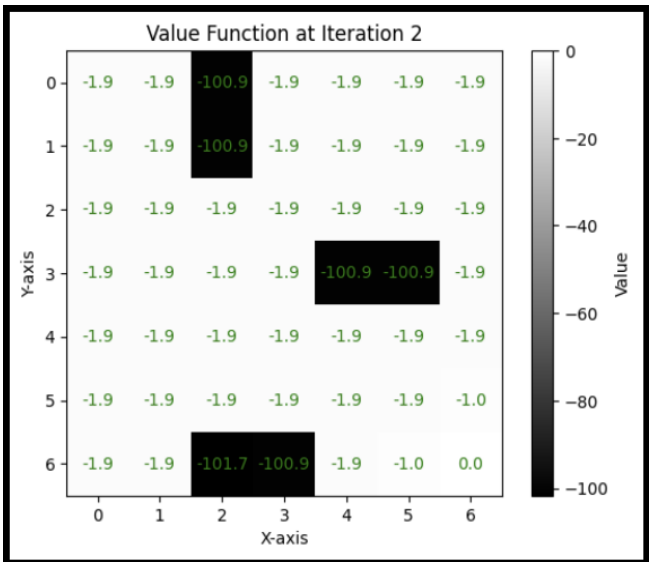
피하도록 유도하는 역할을 하고 있음을 알 수 있다. 일반 셀들은 목표 지점(G)과의 거리와 장애물 위치에 따라 가치가 다르게 형성되어 있으며, 목표 지점에 가까워질수록 점차 높은 가치를 가지는 경향을 보인다. 목표 지점(G) 자체는 가치가 0 으로 설정되어, 에이전트가 목표를 향해 이동할 동기를 제공함을 알 수 있다.

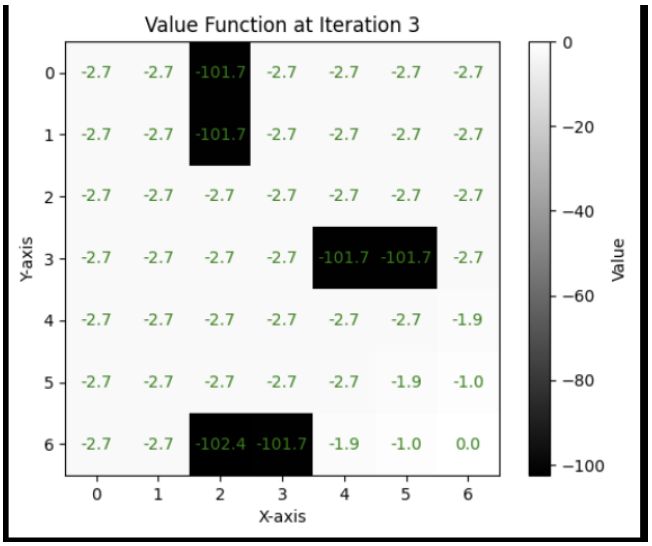
두 번째 그래프의 정책은 최적 경로가 화살표로 표시되어 있으며, 모든 셀이 목표를 향해 일관된 방향을 갖추고 있음을 확인할 수 있다. 시작 지점(S)에서 목표 지점(G)까지 장애물을 피하면서 최단 경로를 유지하도록 방향이 설정되어 있다. 이는 가치 함수에 기반하여 정책이 반복적으로 개선되며 수렴한 결과이다. 최적 정책은 장애물을 피하면서 목표 지점으로 이동하는 가장 효율적인 경로를 제공하고 있음을 볼 수 있다.

결과적으로, 이 수렴된 정책과 가치 함수는 에이전트가 장애물을 피하고 목표 지점으로 이동할 수 있도록 유도하며, 정책 반복 알고리즘이 성공적으로 최적 경로를 찾았음을 볼 수 있는 것이다.

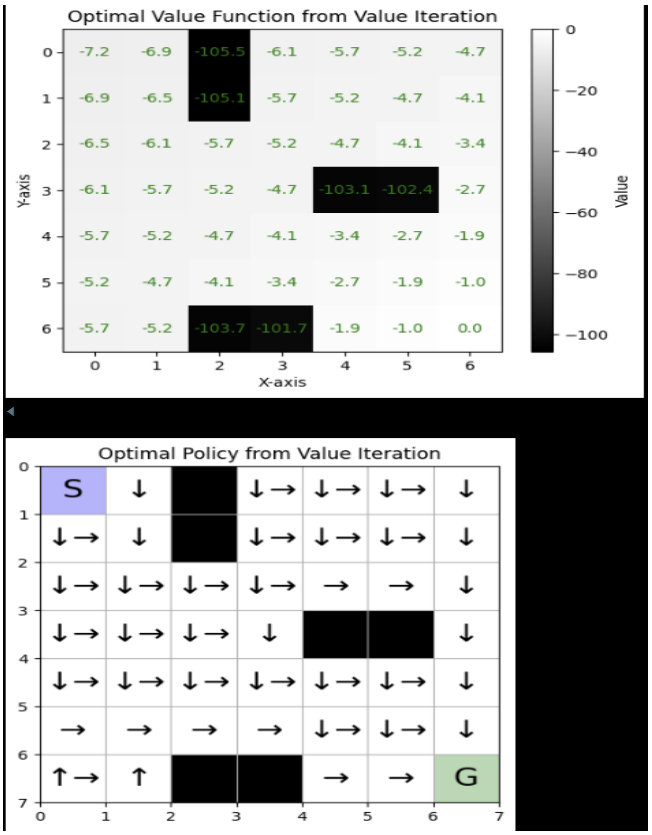
Value iteration 결과 단계별 캡처







(수렴할 때까지 반복)



(최종결과)

위를 확인하면 벨류 이터레이션의 초기 상태는 일반 상태가 -1, 장애물은 -100, 목표 상태 G 는 0 으로 설정된다. 첫 번째 반복에서는 모든 상태의 초기 값이 그대로 유지된다. 이후 반복을 통해 각 상태의 가치가 점차적으로 업데이트된다. 목표 상태 G 에 인접한 상태들의 가치가 갱신되기 시작하여, 목표 상태로부터 한 스텝 떨어진 상태들은 -1 이 되고, 두 스텝 떨어진 상태들은 약 -2 가 되는 것을 볼 수 있다.

반복이 진행됨에 따라 각 상태는 인접한 상태들의 갱신된 가치에 영향을 받아 점차적으로 업데이트된다. 장애물에 가까운 상태나 도착지와 멀면 큰 음수 값을 유지하지만, 목표 상태에 가까운 상태들은 점차 작은 음수 값으로 변화한다. 이 과정이 여러 번 반복되면서 각 상태의 가치는 목표 상태로부터의 거리와 장애물의 위치에 따라 점차적으로 안정화되는 것을 확인할 수 있다.

최종적으로 모든 상태의 가치가 수렴하면 더 이상 큰 변화가 없게 된다. 이 상태에서 각 상태의 가치는 목표 상태 G 까지의 최적 경로를 반영하게 된다. 목표 상태에 가까운 상태일수록 높은(덜 부정적인) 가치를 가지며, 먼 상태일수록 낮은(더 부정적인) 가치를 갖는다. 이를 통해 최선에 정책이 나온 결과를 확인할 수 있다.

고찰

이번 프로젝트를 통해 많은 것을 배웠고 다양한 경험을 할 수 있었다. 이론적으로 배우는 것과 실제로 코드를 구현해보는 것에는 큰 차이가 있다는 것을 배웠다. 특히, 상태 변이와 정책을 관리하는 함수들을 구현하면서 이론적으로 이해한 내용을 실제 코드로 작성하는 과정에서 많은 고민과 시행착오를 겪었다. 이론과 실습의 중요성을 다시 한번 느꼈고, 다양한 오류와 문제들을 마주했지만 강의 자료와 구글링을 통해 이론에 대해 깊이 있게 학습했다. 이를 통해 문제의 원인을 정확히 파악하고 해결하는 방법을 찾을 수 있었다.

각 상태에서 가능한 행동을 정의하고 이를 관리하는 것은 매우 까다로운 작업이었다. 특히, 에이전트가 grid world 에서 이동할 때 경계 조건을 처리하고, 이동 가능한 모든 방향을 반환하는 함수(get_available_actions)를 구현하는 과정에서 많은 시간을 소요했다. 또한, 정책을 평가하고 개선하는 과정에서 각 상태의 가치를 정확히 계산하는 것이 쉽지 않았다. 가치 함수 V 를 업데이트하고 수렴시키는 과정에서 많은 반복 작업이 필요했으며, 이를 효율적으로 처리하는 방법을 찾는 데 어려움이 있었다. 이론적으로는 쉽게 이해되던 내용도 실제로 구현하다 보면 다양한 문제들이 발생했다. 예를 들어, policy_evaluation 함수에서 가치 함수 V 를 업데이트하는 과정에서 임계값(theta)을 설정하고, 이를 기준으로 수렴 여부를 판단하는 과정이 생각보다 까다로웠다.

프로젝트를 진행하면서 가장 크게 배운 점은 문제 해결 능력의 중요성이었다. 각 단계에서 마주하는 문제들을 해결하기 위해서는 끈기가 필요했다. 문제의 본질을 정확히 파악하고, 이를 해결하기 위한 다양한 접근 방식을 시도하면서 해결 방안을 찾을 수 있었다. 코드의 효율성을 높이기 위해 끊임없이 고민하고 개선하는 과정이 중요함을 배웠다. 특히, 상태 변이와 정책을 관리하는 함수들을 구현하면서 코드의 가독성과 효율성을 높이기 위한 다양한 방법을 시도했다. 이를 통해 보다 효율적이고 유지보수가 용이한 코드를 작성할 수 있었다. 프로젝트를 통해 많은 것을 배웠지만, 여전히 부족한 점이 많다는 것을 깨닫았다. 지속적으로 학습하고 자기 개발을 통해 기술적 역량을 향상시키는 것이 중요하다는 것을 다시 한번 느꼈다.

Reference

Chap 5. Decision Making 의사결정 - Policy and Value Iteration Part2. — Saurus2