지속적 강화학습을 활용한
게임 난이도 조절 시스템 연구

광운대학교

소프트웨어학부

2020203023 이정원

2019203040 전귀로

요약: 이 논문에서는 플레이어를 추격하는 게임 시스템의 난이도 적응을 연구한다. 플레이어 추격 시스템이 경로를 예측하여 배치하는 동작을 강화학습을 통해 학습한 인공신경망 모델이 수행하게 하여 플레이어에 지속적으로 대응할 수 있는 시스템을 제작했다. 실험 결과, 지속적 강화학습을 사용하는 학습 기반 시스템이 정적 시스템보다 효과적으로 플레이어의 전략에 대응할 수 있음을 확인하였다.

Keywords: 게임 난이도 조절, 지속적 강화학습, 경로 예측

1. 서론

플레이어를 추격하는 NPC 는 비디오 게임에서 흔히 볼 수 있다. 이들은 일반적으로 특정 관제 시스템에 따라 플레이어 근처에 소환된 후 플레이어에게 접근하게 된다. 이때 플레이어는 추격을 피해 계속해서 이동하고 있기 때문에 플레이어의 이동 방향을 예측하여 앞을 가로막는 방식을 사용하기도 한다. 이러한 예측 소환 방식의 예시로는 오픈월드 액션 게임 Grand Theft Auto 시리즈가 있다. Grand Theft Auto 시리즈의 경우 플레이어가 탑승물을 사용해서 도주를 하고 있는 상황에서 플레이어의 이동 방향 앞에 경찰 바리케이드가 미리 설치되어 있는 경우를 볼 수 있다.

이러한 예측 기반 추격 시스템을 구현하는 일반적인 방법 중 하나는 플레이어의 현재 이동 경로를 취합해서 다음 이동 경로를 계산하는 것이 있다. 이 방법은 시스템이 플레이어의 이동을 그럴듯하게 예측하고 있는 것으로 보일 수 있다. 그러나 이러한 시스템은 주어진 상황이 동일하다면 항상 같은 결과를 내는 정적인 구조이기 때문에 플레이가 반복되다 보면 플레이어는 추격자의 생성 위치에 유사한 경향이 나타나는 것을 발견할 수 있다. 따라서 플레이어는 갈림길에서 바리케이드가 설치되지 않은 도로로 진입할 수 있을 것이다.

Somnuk Phon-Amnuaisuk 은 추격 NPC 의 지능적인 행동을 학습하기 위해 RL 을 사용하는 것은 효과적이라고 보고했다.[1] 그러나 일반적으로 게임 속의 추격자는 게임 내

[1] (Somnuk Phon-Amnuaisuk, 2011)

에서 항상 존재하는 것이 아니라, 필요할 때마다 플레이어의 근처에 소환된 후 동작이 이루어지기 때문에 추격 동작과는 별개로 소환되는 위치에 대한 연구가 필요하다.

이 논문에서는 플레이어의 이동 방향을 예측하는 시스템을 정적인 방법과 지속적 강화학습을 통한 학습 기반 방법 두 가지로 구현했다. 실험을 통해 두 시스템이 플레이어의 전략 연구에 기여하는 효과를 확인했을 때, 학습 기반 방법이 좋은 효과를 낼 것이라고 기대할 수 있었다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 2 장에서는 경로 예측 시스템이 공통적으로 작동하는 환경을 구성하고, 3 장에서 정적 시스템과 학습 기반 시스템을 각각 구현한다. 학습 기반 시스템의 경우 지속적 강화학습을 위한 강화학습 알고리즘 구성과 동작 방식에 대해 설명한다. 4 장에서는 시뮬레이션 실험을 통해 수집한 데이터를 분석하여 두 시스템을 비교하고 기대되는 효과를 기술한다. 마지막으로 본 논문의 기여와 한계 및 향후 연구 방향에 대해서는 5 장에서 기술하였다.

2. 환경 구성

이 장에서는 게임 내의 플레이어 이동 예측 기반 추격 시스템의 동작을 구현하기 위해 사용하는 환경을 설명한다. 환경은 유니티 엔진으로 구현되었다.

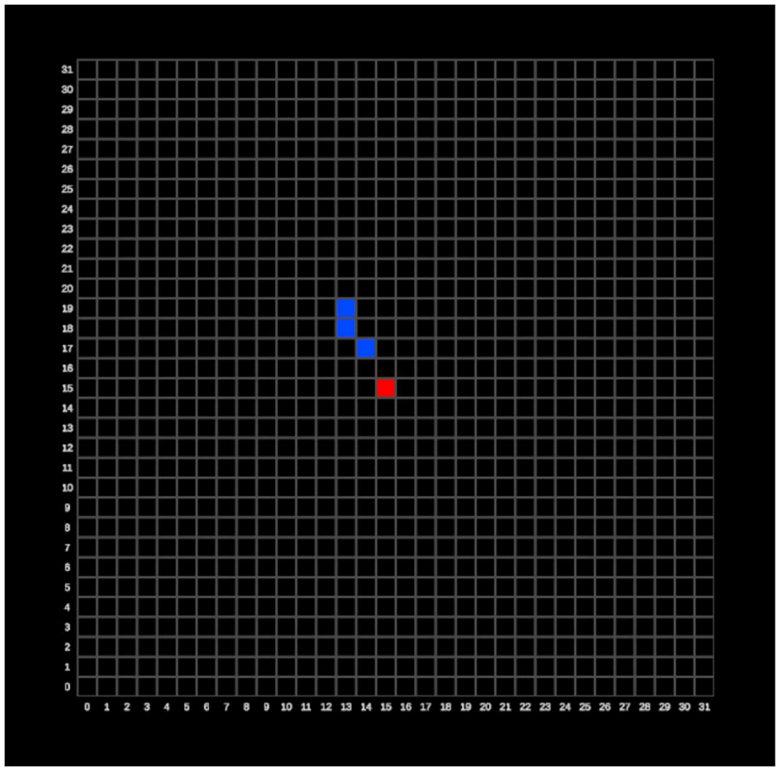

그림 1 32x32 그리드 환경

2.1. 32x32 그리드 보드

환경 내의 모든 개체는 32x32 크기의 그리드 보드 위에서 동작한다. 그리드 내의 좌표 공간은 정수 인덱스로만 대응되는 이산 공간이다. 보드는 Step 주기를 통제하며, 보드에 존재하는 각 개체의 동작은 Step 에 동기화된다.

2.2. 도둑 개체

보드에는 실제 게임에서의 플레이어에 대응하는 도둑 개체가 존재한다. 도둑 개체는 정해진 경로를 따라 보드 위를 이동한다. 도둑이 한 Step 당 이동할 수 있는 방향은 상, 하, 좌, 우로 한 칸 씩이며, 대각선 이동은 불가능하다.

2.3. 경찰 개체

경찰 개체는 실제 게임에서의 적대적 NPC 에 대응한다. 경찰 개체는 5 Step 마다 도둑 개체의 최근 5 Step 동안의 이동 경로를 통해 다음 이동 위치를 결정하며, 이동은 거리에 관계없이 즉시 이루어진다. 이동 경로 예측이 성공하면 이를 '잡았다'고 한다. 경찰 개체는 보드 위에 3 개가 존재하며, 이동 결정 주기와 경찰 개체 수는 조절 가능하다.

3. 이동 경로 예측 추격 시스템

3.1. 정적 이동 경로 예측 시스템

이 논문에서 사용하는 정적 이동 경로 예측 시스템은 그리드 환경에서 이동 경로가 주어졌을 때, 간단하게 목표 개체의 다음 이동 경로를 예측해볼 수 있는 시스템을 구상해본 것으로, 상용 게임에서 실제로 사용하는 이동 예측 시스템의 동작과는 다를 수 있다.

플레이어는 일반적으로 목표 지점을 향해 동일한 방향으로 이동하는 경향이 있을 것이라고 가정한다면, 이전 이동 방향이 있을 때, 미래에도 동일한 이동 방향으로 이동할 것이라고 예측할 수 있다. 따라서 도둑 개체의 이동 경로를 나타내는 N 크기의 좌표 리스트가 주어지면, 이를 N-1 크기의 이동 방향 리스트로 변환한다. 그리고 도둑 개체의 현재 위치에 이동 방향 리스트의 값을 하나씩 더하게 되면 도둑 개체의 예측 이동 경로

를 얻을 수 있다. 이제 만약에 경찰 개체가 3 개라면 이들을 각각 예측 이동 경로의 한 지점당 하나씩 배치하는 방식으로 이동 경로 예측 추격 시스템을 구현할 수 있다.

알고리즘 1: 정적 경로 예측

function dispatch_police
  recently_move_dir = convert(thief_path to move_direction)
  next_thief_position = current_thief_position
  for i in recently_move_dir:
  next_thief_position += i
  predict_thief_path.append(next_thief_position)
  for i, pos in enumerate(predict_thief_path):
  police[i].position = pos

3.2. 학습 기반 이동 경로 예측 시스템

학습 기반 이동 경로 시스템은 강화학습을 통해 학습된 인공 신경망 모델을 사용한다. 강화학습에는 유니티 ML-Agents[2]를 사용하였다.

3.2.1. 환경 및 보상 시스템 구성

강화학습 에이전트는 각 경찰 개체이며, 최근 5 Step 동안의 이동 방향 리스트와 각 경찰 개체의 번호를 뜻하는 정수 값을 관측 값으로 가진다.

경찰 개체 에이전트는 이동 위치 결정을 내린 후, 도둑 개체가 자신의 위치에 도착한다면 보상을 획득한다. 이때 보상은 보드 위의 모든 경찰 개체 에이전트가 공유하는 그룹 보상으로 제공된다. 그리고 Step 당 도둑이 이동한 위치가 경찰 개체가 있는 곳이 아니라면 도둑 개체와 각 경찰 개체 간의 거리의 합을 그룹 페널티로 얻는다. 모델은 각 경찰 개체가 배치될 이동 좌표를 뜻하는 두 개의 정수 값을 출력 값으로 가진다.

---

[2] Unity ML-Agents Toolkit documentation, available at: https://github.com/Unity-Technologies/ml-agents

3.2.2. 모델 구조

Layer	Shape	Input size	Output size
input	Linear	11	256
fc1	Linear	256	256
fc2	Linear	256	256
fc3	Linear	256	256
fc4	Linear	256	256
output	Linear	256	[20, 20]

강화학습 네트워크의 input 은 지난 5 칸의 이동 좌표 기록에 해당하는 2 차원 벡터 5 개(2 x 5)와 각 에이전트의 id(agent_id)에 해당하는 1 개의 실수를 합친 11 개의 실수가 입력된다. output 은 각 20 개씩의 실수 리스트 두 개가 출력되는데, 이때 각 리스트에서 최댓값이 있는 인덱스 두 개가 최종 행동 결정 벡터가 된다.

3.2.3. 강화학습 알고리즘 (DQN)

학습 과정에서는 이동 위치 결정 주기가 강화 학습 모델의 한 Step 이 된다. 학습 과정에서는 한 Step 마다 가져온 관측 값을 통해 각 에이전트의 행동을 결정한 후, 그룹 보상과 다음 관측 값을 획득한다.

에이전트의 행동은 epsilon-greedy 정책을 사용하여 결정한다[3]. 학습에는 Q-Learning 기반의 DQN 알고리즘을 사용했다. 훈련 과정에서 발생한 데이터는 Experience-Replay 메모리를 사용하여 배치 학습을 진행하였다. 학습 결과는 target model 에 먼저 반영된 후 128 step 마다 모델에 적용된다.[4]

이때 Q 의 업데이트는 개별 에이전트 단위로 이루어진다. 이는 각 에이전트가 개별적인 행동 결정을 내릴 수 있게 하기 위함이다. 자세한 동작 방식에 대한 코드는 다음과 같다.

---

[3] (Muhamad Ridzuan Radin Muhamad Amin & Mohd Fauzi Othman, 2021)

[4] (Volodymyr Mnih, et al., 2015)

알고리즘 2: train

```
input_size = 11
output_sizes = [20, 20]
max_replay_buffer_size = 50000
batch_size = 256
update_target_frequency = 128
epsilon = 1.0
epsilon_end = 0.01
learning_rate = 0.001
gamma = 0.99

epsilon_decay_interval = 100
learning_decay_interval = 100

function train:
  while episode < max_episode:
    state = env.get_observation()
    while not episode_end:
      action = get_action(state, epsilon)
      env.set_action(action)
      env.step()
      next_state = env.get_observation()
      group_reward = env.get_reward()

      replay_buffer.append((state, action, group_reward, next_state))

      train_step()

      step += 1
      if step % update_target_frequency == 0:
        target_model.load_state_dict(model.state_dict())

    episode += 1
```

```
    if episode % epsilon_decay_interval == 0:
      epsilon = max(epsilon_end, epsilon * 0.9)
    if episode % learning_decay_interval == 0:
      learning_rate *= 0.99
```

알고리즘 3: train_step (DQN)

```
function train_step:
  batch = replay_buffer.random_choice(batch_size)

  foreach sample in batch:
    for agent_id in range(agents_per_env):
      agent_state = sample.state[agent_id]
      agent_next_state = sample.next_state[agent_id]
      agent_action = sample.action[agent_id]
      group_reward = sample.group_reward

      q_value = sum(model(agent_state))
      next_q_value = sum(target_model(agent_next_state))

      expected_q_value = group_reward + gamma * next_q_value
      update_target_model(q_value, expected_q_value)
```

3.2.4. 모델 추론

모델은 11 크기의 input 을 입력으로 주었을 때, [20, 20] 크기의 output 과 각 20 개의 값 중 최댓값에 대한 인덱스 번호에 해당하는 두 개의 0~19 사이의 정수를 얻을 수 있다. 최종적인 경찰 개체의 이동 결정 위치는 각 정수에 대해서 10(20 / 2)을 빼 준 후, 도둑 개체의 현재 위치에 더해준다. 그리고 그리드 보드의 사이즈에 맞게 0~31 범위로 Clamp 하여 최종적인 경찰 개체의 이동 결정 위치를 얻을 수 있다.

알고리즘 4: 모델 사용

function dispath_police_with_inference:
  input.append(recently_5_move_direction)
  input.append(agent_id)
 	
  output = model.run(input)
 	
  action = output.indices
  move_position = current_thief_position - action - 10
  move_position = clamp(move_position, 0, 31)
 	
  move_to(move_position)

3.2.5. 지속적 강화학습

	0 회차	1 회차	2 회차	…	n 회차
학습 경로 데이터	기초 이동 경로	1 회차 플레이어 이동 경로	2 회차 플레이어 이동 경로	…	n-1 회차 플레이어 이동 경로

모델 최초 학습 시점에 도둑 개체가 사용하는 경로 데이터는 직선 또는 계단형 대각 이동만을 수행하는 경로로 구성된다. 이는 모델에 기초적인 경로에 대한 예측 능력을 부여하기 위함이다.

해당 시스템은 이후 플레이어의 실제 게임 플레이마다 플레이어의 이동 경로를 저장하게 된다. 이때, 플레이어는 플레이를 반복하면서 자신이 게임 내의 목적을 달성하기 쉬운 경로를 찾아내어 자주 해당 경로 패턴으로 움직이게 될 것이라고 가정한다.

플레이어 경로가 일정 이상 쌓이게 되면 해당 경로들을 도둑 개체의 이동 경로 데이터로 하여 강화 학습을 진행하게 된다. 이때 처음부터 학습을 진행하는 것이 아니라 기존 모델을 불러와서 가중치를 업데이트하는 지속적 강화 학습 방식을 사용한다.[5]

[5] (David Abel, et al., 2023)

이 시스템의 목적은 플레이어가 자신이 쉽게 목적을 달성할 수 있는 경로를 찾아내어 해당 경로를 자주 사용하게 된다면 플레이어가 체감하는 게임의 난이도가 낮아질 것이기 때문에 해당 경로에 대한 예측을 강화하여 플레이어가 계속해서 새로운 플레이 경로를 선택할 수 있도록 유도하여 체감하는 난이도를 유지할 수 있게 하기 위함이다.

4. 성능 비교 실험

이 장에서는 실험을 통해 두 가지의 이동 경로 예측 추격 시스템의 플레이어 체감 난이도 유지 성능을 비교하여 설명한다. 이 실험에서는 경로 예측 시스템에서 도둑 개체가 적게 붙잡힐수록 실제 게임에서 플레이어가 체감하는 난이도가 낮을 것이라고 가정한다. 이를 통해 각 이동 경로 예측 추격 시스템이 플레이어의 체감 난이도를 유지시킬 수 있는지 살펴볼 것이다.

4.1. 환경

실험 환경은 2 장에서 구성한 환경에 기반하고 있으며, 각 실험 단계에 매번 새로운 경로 데이터를 생성해주기 위한 경로 생성기가 추가되었다.

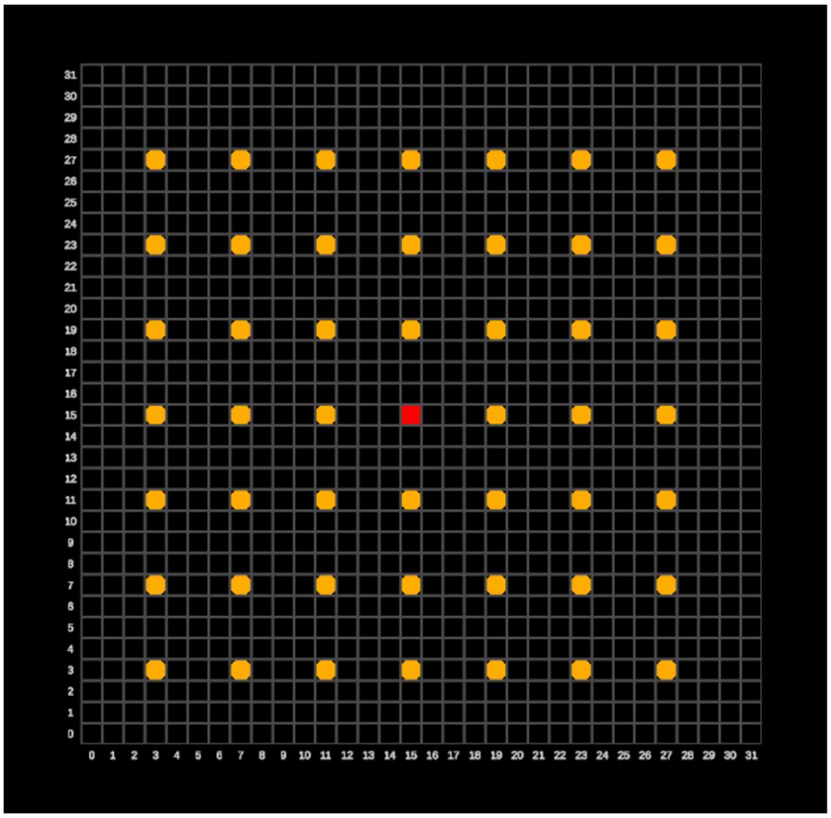

그림 2 경로 생성기

경로 생성기는 동전과 동전 수집기로 구성된다. 동전은 32x32 크기의 그리드에 4 칸씩 간격을 두고 총 49 개가 배치되어 있다. 동전 수집기는 (15, 15) 위치에서 시작하여 자신에 인접한 8 개의 동전 중에서 자신이 아직 방문하지 않는 동전을 무작위로 선택하여 해당 동전 위치로 이동한다. 목표 동전에 도착했다면 다시 인접한 동전 중 무작위로 한 동전을 선택한다. 이 과정을 반복하다가 동전 10 개를 모으면 여태까지 동전을 모으기 위해 이동한 자신의 경로 데이터를 내보낸다.

4.2. 방법

실험은 100 개의 스테이지로 구성된다. 각 스테이지는 경로 생성기를 사용해서 10 개의 경로를 생성하고, 경로 예측 추격 시스템을 사용해서 각 경로에서 도둑 개체가 잡힌 횟수를 기록한다.

이때 잡힌 횟수가 적은 경로는 실제 게임에서 플레이어에게 가장 쉬운 경로이기 때문에, 플레이어가 자신에게 가장 쉬운 방법으로 게임을 플레이하는 합리적인 주체라고 가정했을 때, 다음 플레이 회차에서도 잡힌 횟수가 적은 경로를 선택할 것이다. 따라서 잡힌 횟수가 적은 3 개의 경로를 선택해서 저장한다. 이때 붙잡힌 횟수에 대한 동점자가 있다면 하위 3 개의 경로만 선택할 수 없기 때문에, 동점자를 모두 포함한 3+개의 경로를 선택한다. 이 하위 3+개의 경로들은 다음 Stage 에서 사용하는 경로 데이터 셋에 포함된다.

학습 기반 경로 예측 추격 시스템의 경우에는 플레이어가 자주 선택하는 경로에 대해서 주기적으로 강화학습을 진행하며 모델을 업데이트하는 과정이 포함된다.

이 과정을 총 100 회 반복하며 데이터를 수집한다.

실험 방법	
1	경로 생성기로 생성한 10 개의 경로와 이전 회차의 하위 3+개의 경로를 불러온다.
2	경로 예측 추격 시스템으로 각 경로에서 도둑 개체가 잡힌 횟수 기록한다.
3	잡힌 횟수가 적은 3+개의 경로 저장한다.
4	1 번으로 돌아가 스테이지 카운터를 증가시킨다. 총 100 회 반복한다.

4.3. 결과

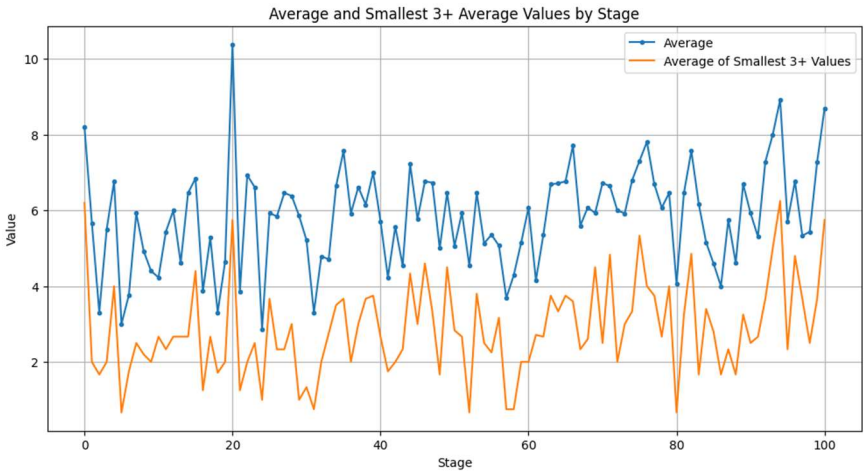

그림 3 학습 기반 경로 예측 추격 시스템 Stage 별 평균 잡힌 횟수, 하위 3+개의 평균 잡힌 횟수

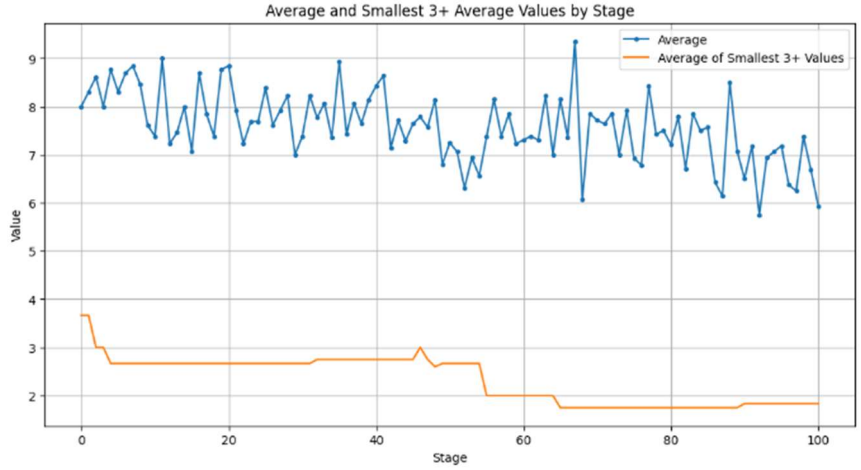

그림 4 정적 경로 예측 추격 시스템 Stage 별 평균 잡힌 횟수, 하위 3+개의 평균 잡힌 횟수

학습 기반 추격 예측 시스템과 정적 추격 예측 시스템의 평균 붙잡힌 횟수를 비교해 보았을 때, 학습 기반 시스템은 평균값이 크게 진동하는 반면, 정적 시스템은 비교적 안정적인 평균값을 가진다. 또한 평균값의 크기 또한 일반적으로 정적 시스템이 더 높다. 이를 통해 정적 시스템이 더 안정적이고 높은 예측 성능을 보임을 알 수 있다.

하지만 하위 3+개의 평균 잡힌 횟수를 보았을 때에 학습 기반 시스템은 감소하는 경향 없이 진동하는 평균값을 가지는 반면, 정적 시스템은 낮은 값으로 시작해서 점점 감소하는 경향이 나타난다. 이를 통해서 알 수 있는 것은 플레이어가 쉽게 선택할 수 있는 경로의 난이도 수준이 점점 낮아지고 있다는 것이다.

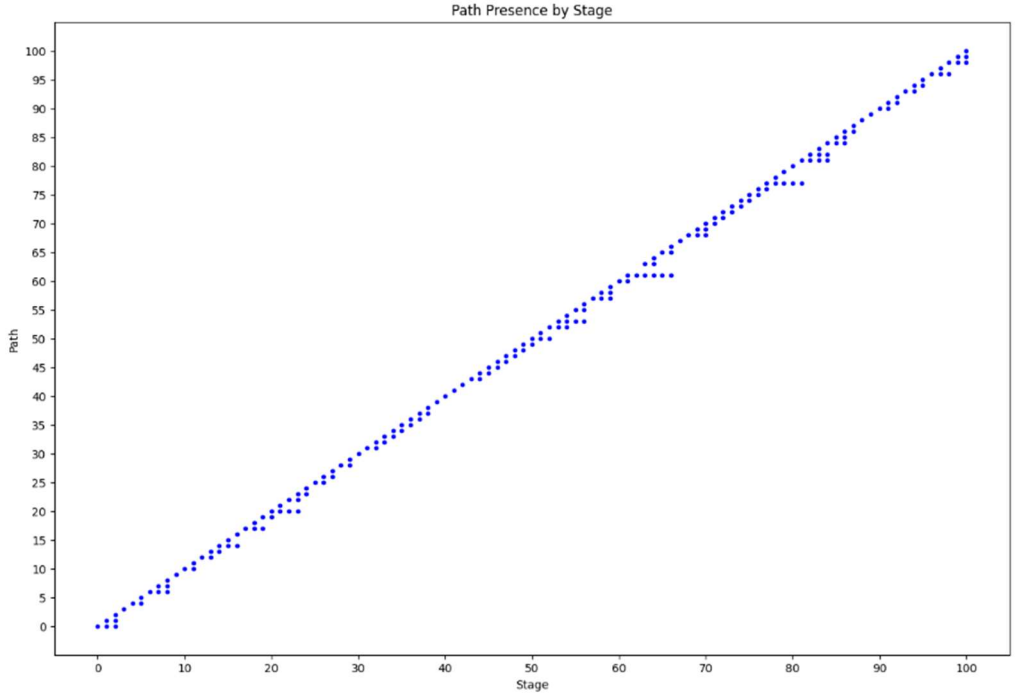

그림 5 학습 기반 경로 예측 추격 시스템 Stage 별 하위 3+개 경로 선택

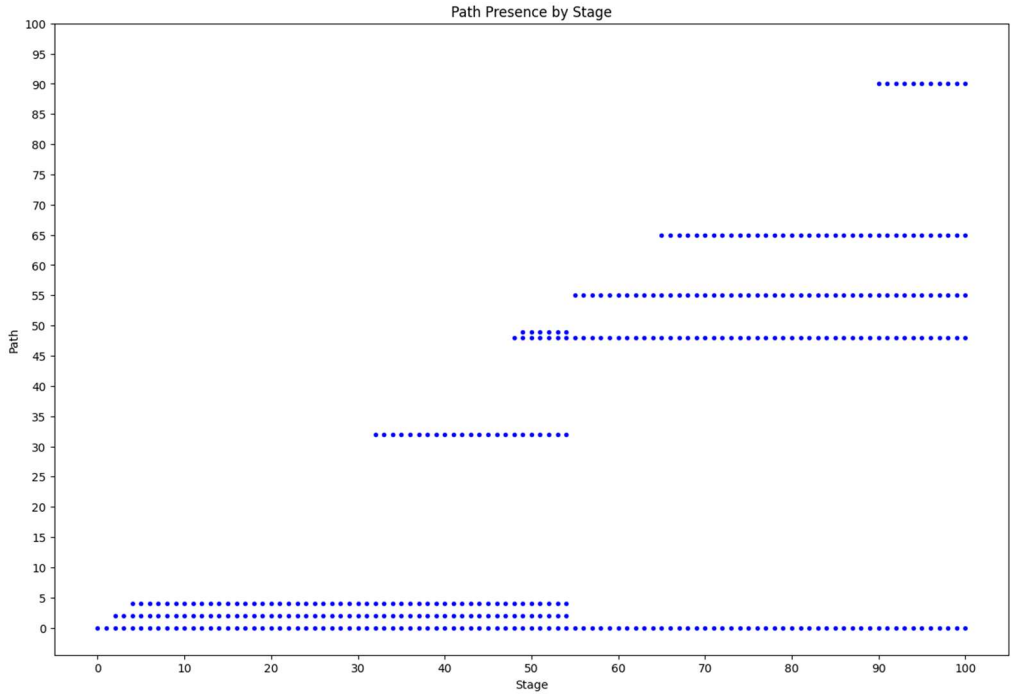

그림 6 정적 경로 예측 추격 시스템 Stage 별 하위 3+개 경로 선택

Stage 별로 각 시스템의 하위 3+개의 경로가 어떻게 선택되고 있는지 살펴보면 정적 시스템의 문제가 더 뚜렷하게 나타난다. 그림 3, 4 는 각 Stage 에서 생성된 경로가 붙잡힌 횟수 하위 3+개의 경로에 포함되는 여부를 보여준다.

학습 기반 시스템의 경우에는 각 스테이지에서 생성된 경로가 최소 0, 최대 6 Stage 동안만 하위 경로에 포함되지만, 정적 시스템의 경우에는 Stage 0 에서 생성된 경로가 Stage 100 까지 하위 3+개의 경로에 포함되는 것을 볼 수 있다. 그 밖에 선택되는 경로들의 수명도 아주 길고 대부분 Stage 100 까지 계속해서 유지된다. 이는 플레이어가 게임 플레이 초기 회차에 발견한 낮은 난이도의 게임 플레이 경로로 계속해서 게임 목표를 손쉽게 달성할 수 있다는 것을 의미한다. 이는 플레이어의 체감 난이도를 크게 낮추는 요인이 될 것이라고 볼 수 있다.

학습 기반 시스템의 각 회차에서 생성된 플레이 경로가 대부분 하나 이상 하위 3+개 경로에 포함되지만, 정적 시스템의 경우에는 9 개의 Stage 를 제외한 나머지 92 개의 Stage 에서 생성된 경로는 하위 3+개의 경로에 한 번도 포함되지 못했다. 따라서 플레이어가 계속해서 플레이 전략을 연구하기 위한 동기부여를 주기 힘들다.

5. 결론

본 연구는 플레이어의 체감 난이도를 유지하기 위한 목적으로 지속적 강화학습을 이용한 학습 기반 이동 경로 예측 추격 시스템을 제안하였다. 이 연구는 이산적 그리드 공간에서 경로를 나타낼 수 있는 상황에서 경로 강화학습 방법과 추격자 배치 시스템을 제공하였다.

플레이어는 계속해서 새로운 전략을 연구할 수 있을 때 체감하는 난이도가 유지되고, 게임에 대한 몰입을 얻을 수 있다. 따라서 게임 시스템은 플레이어가 발견한 전략에 대응하여, 플레이어가 기존의 전략을 포기하고 새로운 전략을 연구하도록 유도해야 한다.

본 연구는 지속적 강화학습 시스템을 통해 이 목표를 달성할 수 있음을 실험으로 보여주었다. 이전 단계에서 플레이어가 목표를 쉽게 달성하기 위해 사용하던 경로 전략들이 정적 시스템에 비해 짧은 수명을 보였으며, 이는 플레이어가 계속해서 새로운 경로 전략을 탐구하도록 유도하는 효과적인 방법이 될 것이다.

제시된 시스템의 한계 중 하나는 정적 시스템에 비해 예측 성능이 낮다는 것이다. 이로 인해 높은 수준의 도전을 원하는 플레이어들은 흥미를 잃을 수 있다. 이를 해결하기 위해 정적 시스템과 학습 기반 시스템을 혼합하여 하이브리드 시스템을 구성할 수 있다.

본 연구에서 사용한 환경보다 더 많은 수의 개체가 존재하는 환경에서는 경찰 개체의 그룹을 나누어 각각 학습 기반 시스템과 정적 시스템을 적용할 수 있다.

이 연구에는 실제 사용자를 대상으로 한 효용 측정이 포함되지 않아, 해당 부분에 대한 추가 연구가 필요하다. 또한, 이 시스템은 이산 그리드 공간을 가정하고 있기 때문에, 보다 일반적인 환경에 대한 추가 연구가 필요하다.

이 시스템은 게임 내의 난이도를 직접적으로 조정하지 않으면서도 플레이어의 체감 난이도를 유지할 수 있게 해주는 시스템을 탐구하였으며, 난이도의 직접적인 조정을 딥러닝을 통해 최적화하는 연구와 결합했을 때 효과적인 결과를 낼 수 있을 것으로 기대된다.[6]

[6] (Edwin A. Romero-Mendez, Pedro C. Santana-Mancilla, Miguel Garcia-Ruiz, & Osval A. Montesinos-López, 2023)

참고 문헌

[1] Somnuk Phon-Amnuaisuk. (2011). Learning Chasing Behaviours of Non-Player. *EvoApplications*, (pp. 133–142).

[3] Muhamad Ridzuan Radin Muhamad Amin, & Mohd Fauzi Othman. (2021). Re-exploration of ε-Greedy in Deep Reinforcement Learning. *RiTA 2020*, (pp. 264–272).

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, . . . Shane Legg. (2015). Human-level control through deep reinforcement learning. *Nature*, (pp. 529–533).

[5] David Abel, André Barreto, Benjamin Van Roy, Doina Precup, Hado van Hasselt, & Satinder Singh. (2023). *A Definition of Continual Reinforcement Learning.* NeurIPS.

[6] Edwin A. Romero-Mendez, Pedro C. Santana-Mancilla, Miguel Garcia-Ruiz, & Osval A. Montesinos-López. (2023). The Use of Deep Learning to Improve Player Engagement in a Video Game through a Dynamic Difficulty Adjustment Based on Skills Classification. MDPI.