캡스톤디자인 계획서

제목 : Unity를 활용한 강화학습 툴킷
Reinforcement Learning Toolkit with Unity

지도교수 : 강동완

제출일 : 2020년 10월 30일

색인번호 202010_0903

조장 : 14109347 송한솔

조원 : 14109353 유호균

: 14109377 조영익

서울과학기술대학교 컴퓨터공학과

1. 캡스톤 디자인 개요 및 계획

1.1 캡스톤 디자인 배경

여러 가지 강화 학습 사례들이 사람들에게 소개되고 있지만, 실질적으로 강화 학습에 대해 사람들은 강화 학습이 어떤 방식으로 작동 하는지, 어떤 효과를 보여주는지 직접 확인할 수 있는 방법은 제한 되어 있다. 강화 학습의 복잡한 이론이나, 실제 작동하는 코드의 내용을 모르더라도, 게임이라는 매체를 통해서 사람들이 실제로 강화학습을 통해서 에이전트가 어떻게 작동하는지, 어떠한 효과가 있는지 알 수 있을 것이다.

1.2 캡스톤 디자인 개발 목표

Unity 환경을 이용해서 주차게임을 구현하고, 그 속에서 인공지능이 스스로 최적의 경로를 찾아가게끔 학습하는 프로그램을 구현한다. 학습하는 과정을 사용자가 직접 보고, 학습한 인공지능과의 대결을 통해 학습의 결과물을 직접 체험해 볼 수 있는 기능들을 구현한다.

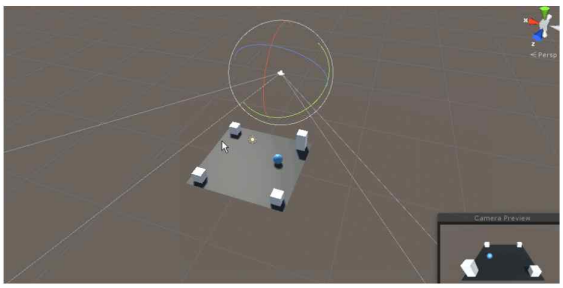
2. 캡스톤 디자인 관련 조사(연구)

2.1 시스템 관련 연구(ML-agent)

① 올바른 목표물을 획득하는 코드 작성

```
    public override void InitializeAgent()
    {
        ballRb = GetComponent<Rigidbody>();
        MakeRandomTarget();
    }

    public override void CollectObservations()
    {
        AddVectorObs(gameObject.transform.position);
        AddVectorObs(target[targetIdx].transform.position);

        AddVectorObs(ballRb.velocity.x);
        AddVectorObs(ballRb.velocity.z);
    }

    public override void AgentAction(float[] vectorAction, string textAction)
    {
        Vector3 velocity = Vector3.zero;
        velocity.x = vectorAction[0];
        velocity.z = vectorAction[1];

        ballRb.AddForce(velocity * 10);

        float distanceToTarget = Vector3.Distance(gameObject.transform.position, target[targetIdx].tr
ansform.position);

/*
        if (beforeDistance > distanceToTarget)
        {
            AddReward(0.001f);
        }

        beforeDistance = distanceToTarget;
*/
        if (distanceToTarget < 1.5f)
        {
            AddReward(1.0f);
            MakeRandomTarget();
            Done();
        }

        if (gameObject.transform.position.y < 0)
        {
            AddReward(-1f);
            Done();
        }
    }

    public override void AgentReset()
```

- agent가 올바른 목표물을 향할 수 있도록 보상 설정



– 유니티에서 목표물을 향해 제대로 이동하는지 확인

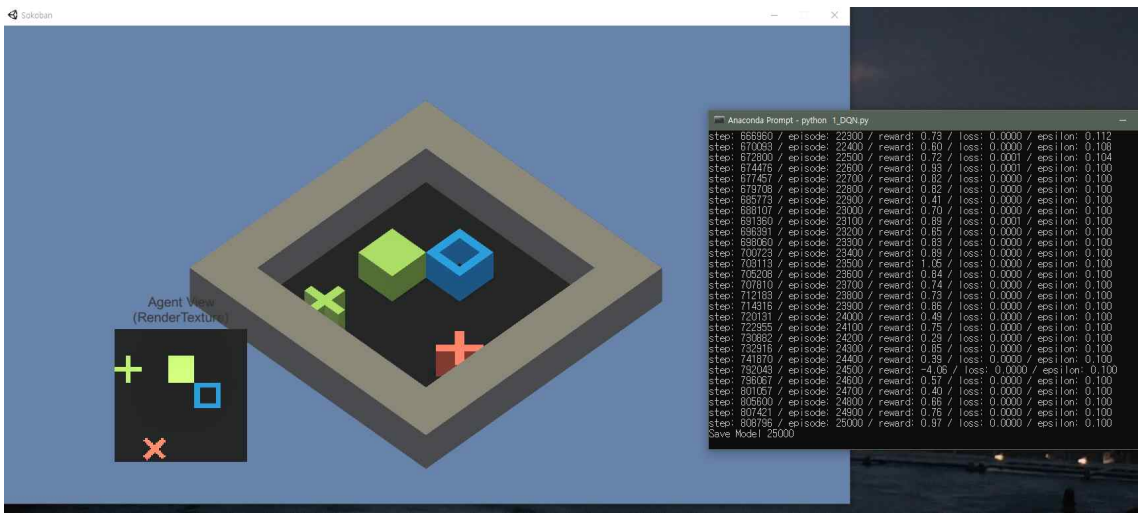
② 예제인 소코반 게임 코드를 통한 원리 이해

```
Collider[] blockTest = Physics.OverlapBox(targetPos, new Vector3(0.3f, 0.3f, 0.3f));
if (blockTest.Where(col => col.gameObject.CompareTag("wall")).ToArray().Length == 0)
{  // 가려는 곳이 벽이 아니면
    if((blockTest.Where(col => col.gameObject.CompareTag("pit")).ToArray().Length == 1)
        || (blockTest.Where(col => col.gameObject.CompareTag("goal")).ToArray().Length == 1))
    {// 에이전트가 골이나 구멍에 빠졌을때
        Done();
        SetReward(-1f);
    }
    else if (blockTest.Where(col => col.gameObject.CompareTag("box")).ToArray().Length == 1)
    { // 박스를 밀게됨
        GameObject box = blockTest[0].gameObject;
        Vector3 nextBoxPos = 2 * targetPos - transform.position;
        Collider[] boxBlockTest = Physics.OverlapBox(nextBoxPos, new Vector3(0.3f, 0.3f, 0.3f));
        if (boxBlockTest.Where(col => col.gameObject.CompareTag("pit")).ToArray().Length == 1)
        {//박스가 구멍에 들어감
            Done();
            SetReward(-1f);
        }
        else if ((blockTest.Where(col => col.gameObject.CompareTag("box")).ToArray().Length == 1)
        || (blockTest.Where(col => col.gameObject.CompareTag("wall")).ToArray().Length == 1))
        {//박스를 벽으로 밀음
            SetReward(-0.1f);
        }
        else if (boxBlockTest.Where(col => col.gameObject.CompareTag("goal")).ToArray().Length == 1)
        {// 박스를 골에 넣음
            GameObject goal = boxBlockTest[0].gameObject;
            transform.position = targetPos;
            if (academy.RemoveBoxGoal(box, goal) == 0) Done();
            SetReward(1f);
        }
        else
        {// 그냥 박스를 민다.
            box.transform.position = nextBoxPos;
            transform.position = targetPos;
            SetReward(0.1f);
        }
    }
    else
    {// 그냥 이동
        transform.position = targetPos;
    }
```

– 에이전트가 가능한 행동을 제한하고(벽으로 이동할 수 없음 등) 올바른 위치에 object를 옮기기 위해 각 행동(Action)에 대한 보상(Reward)값을 설정함.

```
# 게임 진행 반복문
for episode in range(run_episode + test_episode):
    if episode > run_episode:
        train_mode = False
        env_info = env.reset(train_mode=train_mode)[default_brain]

    # 상태, episode_rewards, done 초기화
    state = np.uint8(255 * np.array(env_info.visual_observations[0]))
    episode_rewards = 0
    done = False

    # 한 에피소드를 진행하는 반복문
    while not done:
        step += 1

        # 행동 결정 및 유니티 환경에 행동 적용
        action = agent.get_action(state)
        env_info = env.step(action)[default_brain]

        # 다음 상태, 보상, 게임 종료 정보 취득
        next_state = np.uint8(255 * np.array(env_info.visual_observations[0]))
        reward = env_info.rewards[0]
        episode_rewards += reward
        done = env_info.local_done[0]

        # 학습 모드인 경우 리플레이 메모리에 데이터 저장
        if train_mode:
            agent.append_sample(state, action, reward, next_state, done)
        else:
            time.sleep(0.01)
            agent.epsilon = 0.05

        # 상태 정보 업데이트
        state = next_state

        if episode > start_train_episode and train_mode:
            # 학습 수행
            loss = agent.train_model(done)
            losses.append(loss)

            # 타겟 네트워크 업데이트
            if step % (target_update_step) == 0:
                agent.update_target()

    rewards.append(episode_rewards)

    # 게임 진행 상황 출력 및 텐서 보드에 보상과 손실함수 값 기록
    if episode % print_interval == 0 and episode != 0:
        print("step: {} / episode: {} / reward: {:.2f} / loss: {:.4f} / epsilon: {:.3f}".format
              (step, episode, np.mean(rewards), np.mean(losses), agent.epsilon))
        agent.Write_Summray(np.mean(rewards), np.mean(losses), episode)
        rewards = []
        losses = []

    # 네트워크 모델 저장
    if episode % save_interval == 0 and episode != 0:
        agent.save_model()
        print("Save Model {}".format(episode))

env.close()
```

구한 모델에서 단계를 진행함에 따라 얻은 정보를 기반으로 학습을 진행



- 실제로 유니티를 이용해 ML-agents을 통하여 학습된 결과를 확인.

2.2 캡스톤 디자인 추진 계획

구 분	수행내용	수행일정						
		4월	5월	6월	7월	8월	9월	10월
분석	개발도구 및 개발환경 구비	■						
설계	개발일정 세분화 및 학습	■	■					
개발-게임	주차 시뮬레이션 환경 구현		■	■	■			
개발-AI	주차하는 AI 구현				■	■	■	
기타개발	추가기능 구현						■	
테스트 및 버그수정	구현된 주차게임 테스트 및 발견된 버그 수정							■

5월 중으로 개발에 필요한 Unity와 강화 학습에 대한 학습과 이해를 마치고, 본격적으로 게임 개발과, 학습 알고리즘을 나눠서 개발할 예정이다.

3. 캡스톤 디자인 구현 및 피드백

3.1 캡스톤디자인 구현 내용

학습한 내용을 기반으로 강화 학습을 통해 자동차가 목표 지점으로 가도록 하는 주차 학습 프로토타입 개발 진행

3.2 캡스톤디자인 구현 피드백 및 개선사항

1) 피드백 및 예정 사항

- 주차 환경 개발 미흡
  · Asset store를 통해 이미 구현되어있는 트랙과 차 모델을 사용
  · 7월 중 구현 목표. 이후 환경은 필요에 따라 적당히 수정하여 사용

- 주차 환경에 대한 학습 미흡 :현재 자동차가 학습을 진행해도 가로등에 박는 모션만 수행
  · 에이전트가 환경에서 원하는 동작을 할 수 있도록 구현
  · 9월 중 구현 목표

- 추가 기능 구현 및 테스트
  · 개발 결과 추가로 필요한 기능을 추가
  · 학습결과가 적당한지 확인하고 수정

- ML-agents version 호환성 문제
  · ML-agents가 출시된지 얼마 안된 기능이라 지원하는 기능이 이전 버전에서나 높은 버전에서 호환이 안되는 문제가 빈번하게 발생되었음. 각자 파트 부분을 개발했을 때, 환경적인 문제를 일치시키고 하지 않으면 개발한 부분을 모을때에 오류가 발생할 수도 있음.
  · 적절한 버전을 택해서 조원 모두가 일치시키고 개발 예정

4. 캡스톤 디자인 결과 및 효과

4.1 캡스톤디자인 예상 결과 및 효과

· 게임 뿐만이 아닌 지금 현실에서의 자율 주행의 한계에서 벗어나 자율 주차도 가능하게 하는 자동차에 도입 가능한 프로그램 구현

· 각종 레이싱 게임 에서의 주행이나 실제 현실 자율주행에서의 발생하는 버그나 오류를 찾아낼 수 있는 프로그램 구현.

· 현재 대두되고 있는 면허시험과 장롱면허 문제를 해소시킬 수 있는 시뮬레이션 게임 프로그램 개발

5. 참고자료, URL, 문헌

[1] Unity Learn https://learn.unity.com/

[2] Youtube, AI Learns to Park - Deep Reinforcement Learning https://www.youtube.com/watch?v=VMp6pq6_QjI

[3] ML-agents Document https://github.com/Unity-Technologies/ml-agents

[4] "텐서플로와 유니티 ML-Agents로 배우는 강화학습", 리디북스, pp.181-237 , 소코반 부분