졸업작품 보고서

2088033 이창현

01. 게임 소개

어떤 게임인가요?

‘기사도에 대하여’는 웹툰 작가 환댕의 시리즈 물, ‘애늙은이’와 ‘잔불의 기사’에 크게 감명받아 제작하게 된 수집형 카드 게임이다. 기존 CCG장르의 토대를 따라가는 동시에, 장르의 단점을 보안하기 위한 새로운 시도를 볼 수 있다.

게임명 및 개발 엔진	장르	플랫폼
'기사도에 대하여' 유니티	CCG (Collecting Card Game)	PC (차후 모바일)

02.
CCG란?

수집하고, 구축하자!

Collecting Card Game은 기존의 카드 게임과는 다르게 플레이어가 직접 카드를 수집해서, 다양한 조합을 구성해 다른 플레이어와 게임을 즐기는 형태의 카드 게임이다. 카드의 종류가 한정되어 있는 전통적인 카드 게임들과는 다르게, 무궁무진한 조합의 덱이 나올 수 있는 재미가 있다.
상위 장르로는 TCG(Trading Card Game)이 있다.

TCG/CCG 장르의
대표 게임들

하스스톤

매직 더 개더링

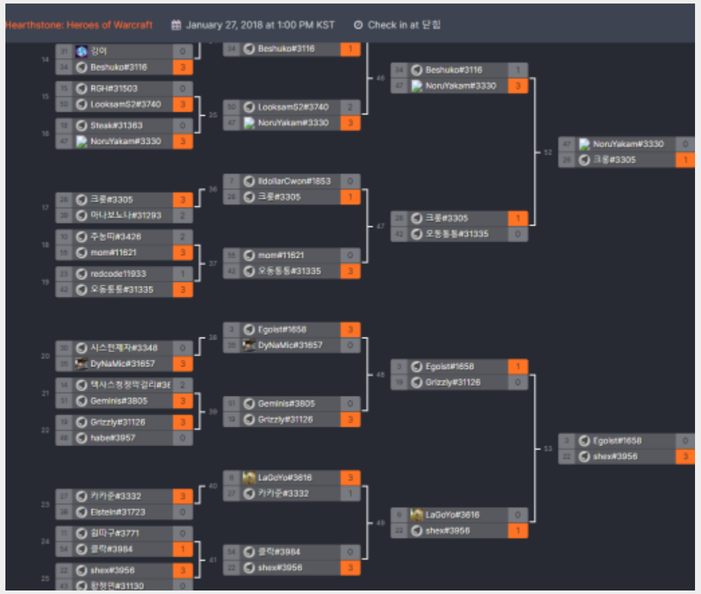
섀도우버스

유희왕 카드 게임

03.
개발 동기

인생을 갈았습니다.

카드 게임에 1만 시간이 넘는 세월을 녹인 만큼, 장르에 대해 이해도가 깊다고 자부합니다.
진짜 CCG, 이 이창현이 만들어보겠습니다.



하스스톤 프로게이머의 꿈을 꾸며, 아마추어 대회를 전전했습니다. 하스스톤 오프라인 이벤트 성 대회도 출전할 만큼, 하스스톤에 진심이었습니다.

유희왕 마스터듀얼이 출시하고 한달 만에 플레이 시간 200시간을 돌파했습니다. 코딩보다 더 많은 시간을 유희왕을 하는데 들였습니다.

04.
차별점

TCG/CCG 장르 게임 비교

게임 이름	덱 매수	턴 제한	턴 당 시간	게임 당 시간	슬러시 타임	판 당 플레이타임 (추정치)
유희왕 마스터 듀얼	40 ~ 60	없음	60초	420초	애니메이션 재생 시	5 ~ 40 분
하스스톤	30	없음	75초	없음	추정 15초	5 ~ 30 분
섀도우버스	40	없음	확인불가	없음	확인불가	10 분
발이 묶인 매	15	없음	30 ~ 60초	없음	없음	5분

기존의 TCG/CCG장르 게임들은 두꺼운 덱 매수, 긴 대기 시간, 오래 걸리는 한판 등 캐주얼함이 떨어지는 모습을 보였다.
이 부분을 해소하기 위해 파격적으로 최대 턴 수를 제한하고, 덱 매수를 줄여 게임을 보다 가볍고 빠르게 진행하고자 한다.

추진 일정

05.

추진 내용	3월	4월	5월	6월
기본 게임 시스템 개발				
네트워크 모듈 적용				
UI 개선				
이펙트 추가				
모델링				
추가 게임 시스템 개발				
피드백				

06.
수행도

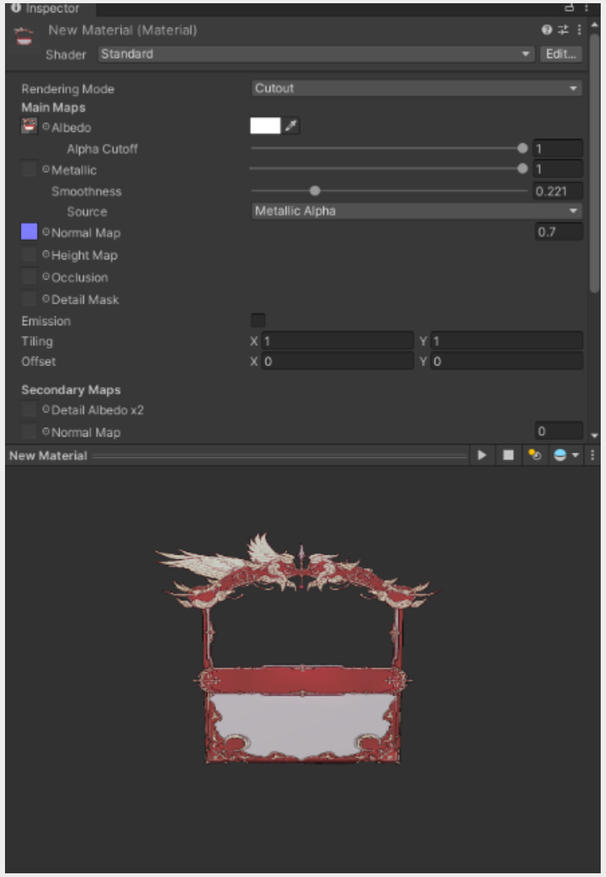
추진 내용	수행도
기본 게임 시스템 개발	100%
네트워크 모듈 적용	100%
UI 개선	100%
이펙트 추가	70%
모델링	100%
추가 게임 시스템 개발	50%
사용성 테스트	40%

07.

수행 내용

카드 디자인 및 모델링

- 카드 디자인은 외주로 진행
  이를 바탕으로 모델링 및
  UV 맵핑을 진행
  유니티에서 머테리얼 생성 후
  적용

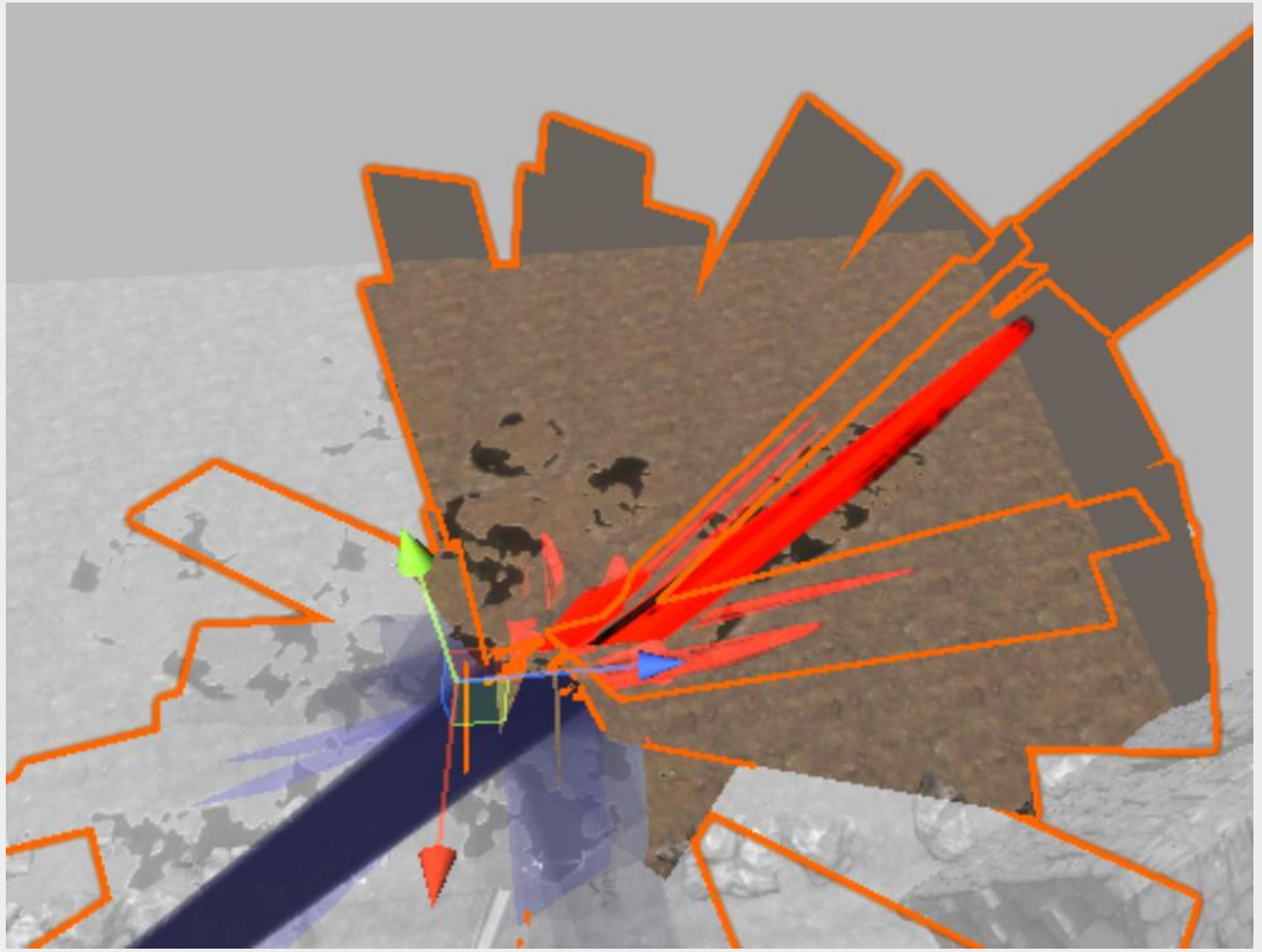


UI 및 배경

- 게임 진행 배경
- 턴을 표기하는 UI
- 방 생성 및 접속
- 채팅 UI
- 덱 생성 UI
- 설정

이팩트
& 사운드

- 공격 이팩트 추가
- 생성 이팩트 추가
- 배경음악 추가
- 공격 효과음 추가
- 생성 효과음 추가



08.

주요 기능

덱 빌딩

플레이어가 원하는 카드들로 덱을 구성해서 저장할 수 있게 해주는 시스템이다. 덱은 JSON파일로 저장된다. System.IO 네임스페이스를 사용해 구현했다.

```cs
using UnityEngine;
using System.IO;
using System.Collections.Generic;
using UnityEngine.UI;
using TMPro;

/// <summary>
/// 덱을 빌딩해주는 역할
/// </summary>
public class DeckManager : MonoBehaviour
{
    [SerializeField] ItemSO itemSO;
    [SerializeField] Card card;
    [SerializeField] Button addButton;
    [SerializeField] Button delButton;
    [SerializeField] GameObject[] children;
    PlayerData playerData = new PlayerData();
    int curCard = -1;


    public void Counting()
    {
        int num = 0;
        foreach(var i in children)
        {
            var j = i.GetComponentsInChildren<TMP_Text>();
            j[0].text = itemSO.items[num].cardName;
            int boool = 0;
            foreach (int k in playerData.list)
            {
                if (k == num) boool++;
            }
            num++;
            j[1].text = boool.ToString();
        }
    }
```

```cs
public void SelectCard(int i)
{
    card.init(itemSO.items[i]);
    curCard = i;
    int boool = 0;
    foreach(int num in playerData.list)
    {
        if (num == curCard) boool++;
    }
    if (boool >= 2)
    {
        addButton.interactable = false;
        delButton.interactable = true;
    }
    if ( boool == 1){
        addButton.interactable = true;
        delButton.interactable = true;
    }
    if (boool == 0)
    {
        addButton.interactable = true;
        delButton.interactable = false;
    }
}

public void AddCard()
{
    if (curCard == -1) return;
    playerData.list.Add(curCard);
    delButton.interactable = true;
    int boool = 0;
    foreach (int num in playerData.list)
    {
        if (num == curCard) boool++;
    }
    if (playerData.list.Count >= 15 || boool >=2) addButton.interactable = false;
    Counting();
}

public void DeleteCard()
{
    if (curCard == -1|| !playerData.list.Contains(curCard)) return;
    playerData.list.Remove(curCard);
    addButton.interactable = true;
    if (!playerData.list.Contains(curCard)) delButton.interactable = false;
    Counting();
}
```

```cs
public void SaveDeck()
{
    File.WriteAllText(Path.Combine(Application.dataPath, "Deck.json"), JsonUtility.ToJson(playerData, true));
}

public void LoadDeck()
{
    playerData = JsonUtility.FromJson<PlayerData>(File.ReadAllText(Path.Combine(Application.dataPath, "Deck.json")));
}
}

[System.Serializable]
public class PlayerData
{
    public List<int> list = new List<int>();
}
```

카드 업데이트

다양한 능력의 카드들이 존재한다. 구글 스프레드 시트를 DB로 사용해서 카드들의 정보를 담아둔다. 이를 Scriptable Object의 형태로 local에 저장하게 된다. UnityEngine.Networking 네임스페이스를 사용해 구현했다.

```
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

Unity 스크립트(자산 참조 1개)|참조 0개
public class GameManager : MonoBehaviour
{
    [SerializeField] ItemSO itemSO;
    const string URL = "https://docs.google.com/spreadsheets/d/1Yf_I11QO_boOBH3SYr3OcnhM1SdK47oNhZ8FdzpRaz4/export?format=tsv&range=A2:H33";

    Unity 메시지|참조 0개
    private void Start()
    {
        StartCoroutine(DownloadItemSOFromGoogleSheets());
        SpriteManager.Init();
    }

    참조 2개
    IEnumerator DownloadItemSOFromGoogleSheets()
    {
        UnityWebRequest www = UnityWebRequest.Get(URL);
        yield return www.SendWebRequest();
        SetItemSO(www.downloadHandler.text);
        yield return new WaitForSeconds(5f);
        StartCoroutine(DownloadItemSOFromGoogleSheets());
    }

    참조 1개
    void SetItemSO(string tsv)
    {
        int i = 0;
        Sprite[] sprites = new Sprite[32];
        foreach (string r in tsv.Split('₩n'))
        {
            string[] c = r.Split('₩t');
            itemSO.items[i] = new Item(i++, c[0], c[1]
        }
    }
}
```

	A	B	C	D	E	F	G	H
1	CardType	CardName	CardGrade	AttackPoint	HealthPoint	Cost	Ability	능력
2	기사카드	마을의 봄	수습기사	0	10	1	0	마을의 희망이다
3	기사카드	꺼지지 않은 잔불	수습기사	10	1	4	0	의지는 아직 꺼지
4	기사카드	잃어 버린 기억	수습기사	2	4	3	0	기억을 되찾으면
5	기사카드	신념있는 끈기	수습기사	1	3	1	0	*신념을 다하겠다*
6	기사카드	신속한 숙고	수습기사	1	4	2	0	깊고 빠른 고민.
7	기사카드	쓰레기의 가면	수습기사	3	7	4	0	가면 안, 누가 있
8	기사카드	눈물 많은 잠작	수습기사	1	7	3	0	누구보다 슬프게
9	기사카드	살눈	수습기사	1	5	2	0	앞은 잘 보인다.
10	기사카드	빡눈	수습기사	2	6	3	0	앞은 잘 보인다.
11	기사카드	무술의 후계자	수습기사	1	3	2	0	고대 무술을 다루
12	기사카드	팔자 주름	수습기사	2	7	4	0	나이에 비해 노안
13	기사카드	붉은 미역	수습기사	2	8	4	0	미역을 별로 안
14	기사카드	슬픈 눈	수습기사	3	9	5	0	눈물 자국이 남아
15	기사카드	최강의 소녀	수습기사	1	3	1	0	나, 강림
16	기사카드	최조	자유기사	8	8	8	0	그, 이전의 기사
17	기사카드	악마	자유기사	8	8	8	0	기사라 칭하기에
18	기사카드	하마	정식기사	5	10	4	0	모든 걸 먹어 치

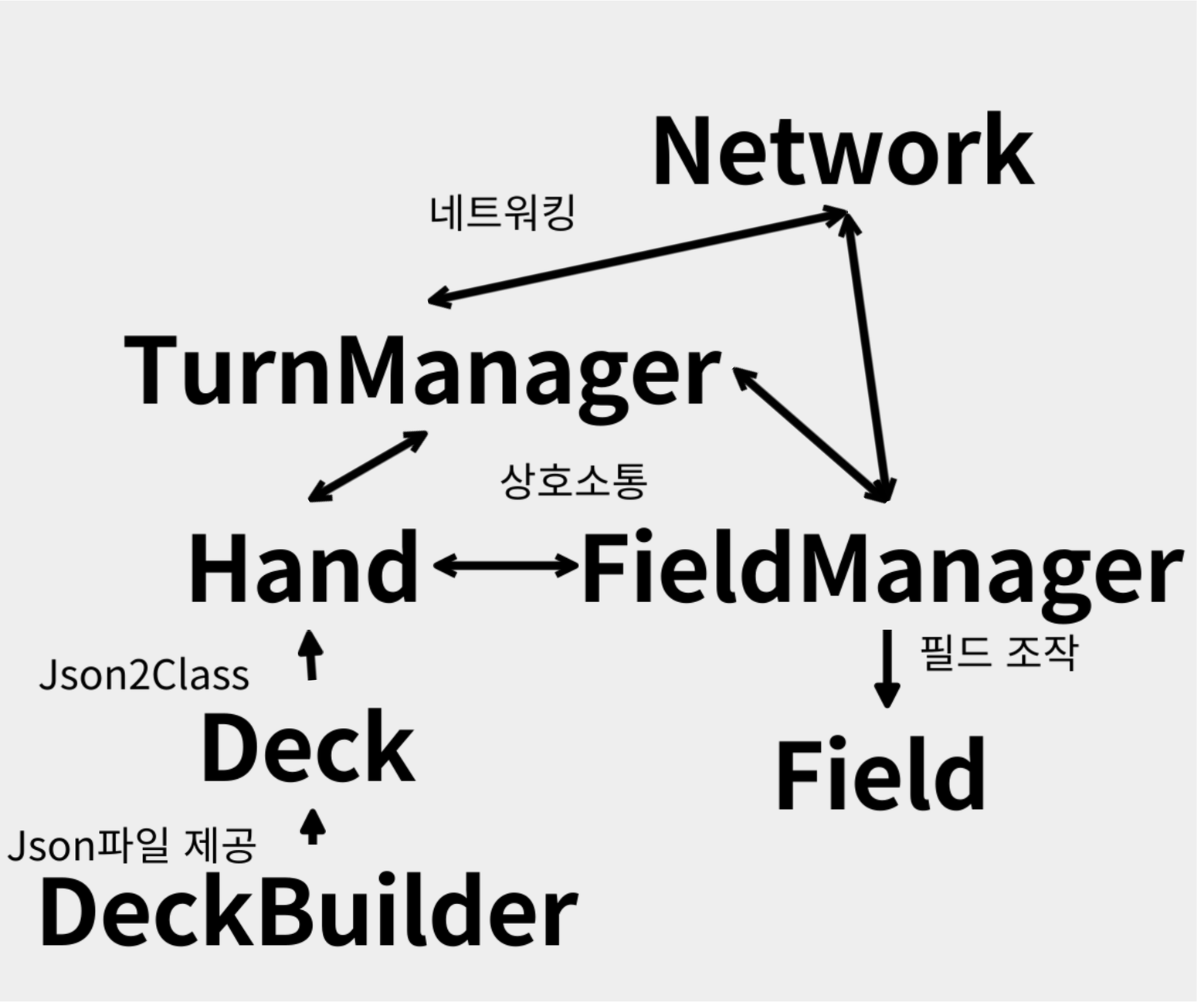
Texture lazy loading

- Texture가 많은 카드 게임 특성상, 게임을 처음 실행할 때, 많은 로딩 시간을 동반하는데, 이를 보안하기 위해서 Texture 파일들을 Start/Awake할 때가 아닌, 원하는 타이밍에 할 수 있게 만들었다. UnityEngine 네임스페이스를 사용해 구현했다.

```
using UnityEngine;
using UnityEngine.U2D;

참조 5개
public class SpriteManager : Singleton<SpriteManager>
{
    static SpriteAtlas spriteAtlas;
    static Sprite[] sprites;

    참조 0개
    public SpriteManager()
    {
        Debug.Log("SpriteManager 초기화");
    }

    참조 1개
    public static void Init()
    {
        spriteAtlas = Resources.Load<SpriteAtlas>("Sprites/CardSprites");
        sprites = new Sprite[spriteAtlas.spriteCount];
        spriteAtlas.GetSprites(sprites);
    }
```

객체지향 하고 싶었다

기능 별로 클래스를 나눠 클래스 끼리 소통을 하게 했다.
과도하게 나눈 감이 있지만, 오류를 고치기 쉬웠다.



감사 합니다!

게임공학과
2088033
이창현