논문 2021-06-16

Unity엔진을 이용한 턴제 카드 게임(Battle Tourment)

(Turn-based Game)

팀장 : 이규남 팀원 : 심재용, 안은규,
(Lee Gyu Nam, Shim Jea Jong, Ahn Eun Gyu)

Sunmoon University

Abstract : 2020년 코로나 19 발생 이후 자택에 있는 시간이 늘어남에 따라 자택에서 할 수 있는 여가활동이 조명을 받고 있다. 여러 여가활동중 게임도 조명을 받고 있는데 이러한 흐름에 따라 Unity 엔진을 활용하여 턴제 카드게임 개발을 진행하였다. 이 게임을 통해 30대 40대들의 관심도를 높이고 누구나 쉽게 할 수 있어 접근성을 높이고자 한다. 턴제 카드게임은 2D PC 플랫폼을 기반으로 한다. 이 논문에서는 카드 턴제게임의 전체적인 특징, 구성 및 기술 개발 등을 소개하며 개발자들의 개발 목록 및 개발과정에 대해 설명한다.

1. 서론

2020년부터 코로나19 확산으로 강제적으로 집에 있는 상황이 되었다. 하지만 일부분의 사람들이 코로나19 감염 여부에 상관없이 야외활동을 하여 코로나19 집단감염이 걸리는 경우가 많이 생겨났다.

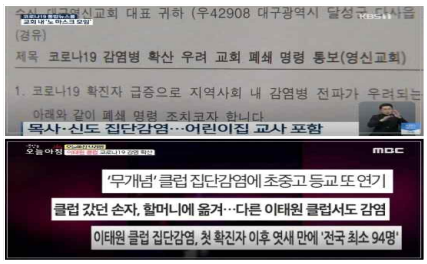

그림 1 야외활동 코로나19 감염

사람들이 야외활동을 하는 이유를 조사한 결과 '집에서의 활동이 지루해서', '밖에서의 활동이 더 재미있어서', '집에 있는게 답답해서' 라는 답변을 받았다. 이러한 답변을 받은 이유는 오랫동안 지속된 코로나19로 인해 집에서 버티기 힘들어서 라는 결과가 나왔다. 하지만 아직 코로나19가 종식되지 않은 상황이기에 집에서의 여가활동을 권장하는 상황이다.

2015년부터 집에서의 여가활동 시간이 증가하고 있다. 자택에서의 여가활동이 주목받고 있는데 비디오스트리밍, 독서, 음악감상, TV시청, 게임 등 여러 가지 여가활동이 있는데 그 중 게임의 여가활동 시간에 하는 사람들이 늘어나고 있다.

그림 2 게임의 여가활동 증가

여가활동으로 게임을 하는 사람들이 늘어나고 있지만 30대 40대들에게는 게임이 너무 복잡하여 게임을 여가활동으로 선택하지 않는 사람들이 있다. 게임의 접근성을 높여 게임에 대한 관심을 가질 수 있게 게임 개발을 하였다.

이 논문은 PC 플랫폼을 이용한 턴제 카드게임의 개발에 관한 연구를 기술하고 있다. 이 프로젝트의 목적은 게임에 익숙하지 않은 30대 40대에게 게임에 대한 관심도와 접근성을 높이며 10대 20대에게는 흥미를 느낄 수 있게 개발하였다. 2장에서는 게임 소개 및 시스템을, 3장은 개발도구 및 개발과정

을, 4장에서는 게임개발 과정 및 향후 방향을 설명한다.

Ⅱ. 게임 소개 및 시스템

본 연구에서는 턴제 카드게임으로 개발하였고 4명의 캐릭터와 20장의 카드로 구성된 랜덤한 덱으로 AI와 플레이어 간의 1:1 대결을 통해 승패가 결정되는 게임이다. 다음은 자세한 게임 플레이 시스템을 설명한다.

(1). 메인화면

그림 3 메인화면 타이틀

메인화면은 게임의 타이틀인 Battle Tournament와 Play버튼과 Quit으로 구성되어 있다. 각 버튼에 마우스를 올리면 버튼의 색이 변하게 된다. Play버튼을 누를 시 캐릭터 선택창으로 가고 Quit버튼을 누를 시 게임이 종료된다.

(2)캐릭터 선택 화면

그림 4 캐릭터 선택창

본 화면에서는 도적, 마법사, 전사, 궁수로 총 4가지 직업을 선택할 수 있다. 캐릭터를 선택하면 해당 캐릭터의 밝기가 밝아지고 선택을 누르게 되면 플레이 화면으로 넘어가서 해당 캐릭터가 생성되게된다.

(3) 플레이 화면

그림 4 플레이 화면

그림 5 게임시작 버튼클릭 후 시작화면

플레이 화면으로 넘어가게 되면 선택된 캐릭터가 생되고 게임시작 버튼이 나타난다. 게임시작을 누르게 되면 AI와 플레이어가 서로 4장을 카드를 받게 되고 나의 턴이나 상대 턴이 시작된다. 나의 턴이 되었을 때 카드한장을 받고 캐릭터 왼쪽에 현재마나와총마나가 표시되고 1턴마다 1씩 증가한다. 우측 상단에는 On/Off버튼과 게임종료버튼이 있으며 On/Off버튼 클릭시 게임BGM이 재생하고 한번 더 클릭시 꺼진다. 게임종료를 누르면 게임이 꺼지면서 종료된다.

그림 6 게임진행 화면(플레이어턴)

그림 6는 현재 플레이어의 턴이며 카드를 필드에 내면 해당 카드의 마나만큼 캐릭터 왼쪽의 현재 마나가 줄어들게 되고 마나를 모두사용하거나 카드를 2장을 냈으면 카드를 비활성화시켜 카드를 낼 수 없게된다. 현재마나보다 높은 비용의 카드 또한 비활성화 되어

있다. 카드를 내서 필드에 카드를 내면 나와있는 체력과 공격력의 유닛이 소환되고 한 턴이 지나게 되면 공격할 수 있는 상태가 된다. 한 턴이 지나고 공격할 수 있는 상태가 되면 유닛을 클릭하면 타켓표시가 생성되고 AI의 본체나 필드의 유닛을 공격할 수 있다.

그림 7 게임진행 화면(상대 AI턴)

그림 7은 AI가 진행되고 있는 모습으로 플레이어 같은 방식으로 카드를 소환하고 공격한다. 플레이어의 턴종료 버튼이 비활성화되고 회색의 버튼으로 바뀌게 된다.

그림 8 공격시 공격이펙트

그림 8에는 필드의 유닛이 적 유닛이나 적의 본체를 공격할 때 공격 이펙트가 발생하고 있는 모습이다.

그림 9 게임 승리

필드는 최대 6개의 유닛을 유지할 수 있으며 적의 본체의 체력을 0이하로 만들 경우에는 승리 텍스처가 뜨면서 다시하기 버튼이 생성된다. 다시하기를 누르게 되면 게임을 다시시작한다.

(4) 카드 드로우 및 정렬

그림 10 카드 드로우와 원형정렬

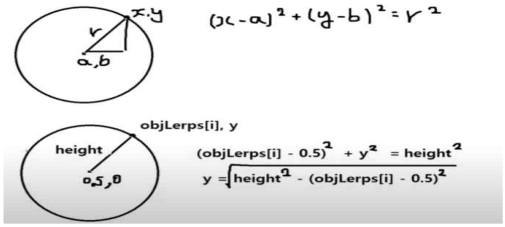

그림 11 원의 방정식

그림 10는 차례대로 카드를 정렬하고 있는 모습이며 원의 방정식을 정렬알고리즘에 적용하여 카드를 드로우하면 원형으로 카드를 정렬하게 된다.

Ⅲ.개발 도구 및 개발과정

1. Unity Engine

유니티 엔진은 3D 및 2D 비디오 게임의 개발 환경을 제공하는 게임 엔진이자, 3D 애니메이션과 건축 시각화, 가상현실(VR) 등 인터렉티브 콘텐츠 제작을 위한 통합 제작 도구이다.

유니티 엔진은 윈도우, 맥OS, IOS, 안드로이드, 플레이스테이션, 엑스박스, 닌텐도 스위치, 웹브라우저 등 27개의 플랫폼에서 사용 가능한 콘텐츠를 만들 수 있고, 제작 도구인 유니티 에디터는 윈도우와 맥OS를 지원한다.

엔진 자체에 라이트 매핑, 물리 엔진 등 미들웨어를 탑재했으며, 에디터에 내장된 에셋스토어를 통해 다양한 기능의 에셋을 다운로드하여 사용할 수 있다.

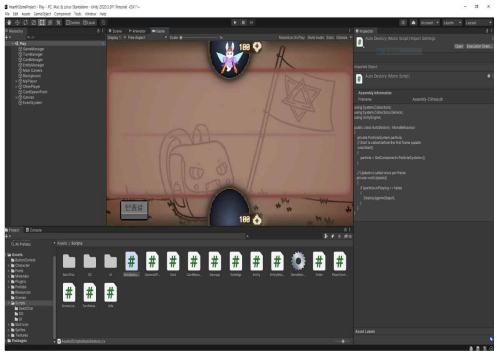

그림 12 Unity Engine 기본 화면

2. Visual Studio 2017

비주얼 스튜디오(Microsoft Visual Studio)는 마이크로소프트 윈도우에서 작동하며, 다양한 언어로 프로그래밍할 수 있는 마이크로소프트의 통합 개발 환경이다. 비주얼 스튜디오를 통해 Unity Engine에서 제공하는 monoscript에서 하지 못했던 코딩을 통해 게임의 생동감을 부여 해준다.

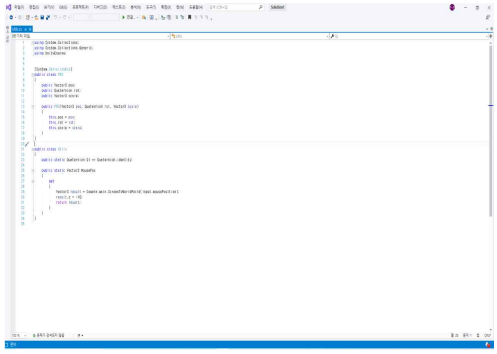

그림 13 Visual Studio 2019 기본 화면

3. photopea

온라인 웹버전으로 무료로 이미지편집할 수 있는 프로그램으로, 사진 이미지의 색상 보정, 오래된 사진 복원, 이미지 합성, 문자 디자인, 인쇄물 디자인, 웹디자인 등의 작업한다. 그림 3번의 게임 타이틀 글씨와 배경을 편집하는데 사용되었다.

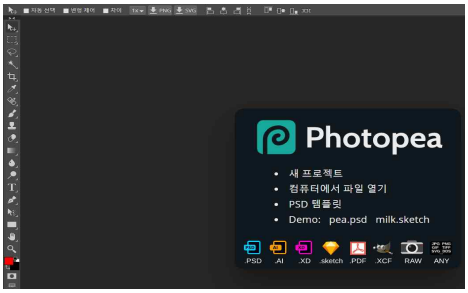

그림 14 photopea 웹화면

4. Unity Asset Store

유니티 에셋 스토어는 프로젝트를 위한 에셋을 구매하고 다운로드 받을 수 있게 에디터에 내장되어 있는 상점이다. 에셋 스토어를 통하여 다양한 게임 개발에 필요한 UI, 오브젝트, 캐릭터를 구매하여 쉽게 개발을 할 수 있게 해준다.

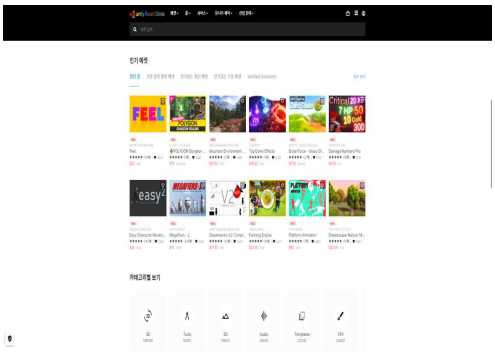

그림 15 Unity Asset Store 화면

Ⅳ. 게임개발 과정

1. 게임개발 과정

1) 게임 제작 과정

프로젝트 팀원과 회의를 통해 아이디어를 생각하여 연구과제 제안서를 작성하였다. 작성한 내용을 토대로 시장분석 조사를 하고, 게임개발 기획을 하였다. 첫 번째로 시나리오 작성, 두 번째로 필요한 요구사항 분석을 하였고, 마지막으로 요구사항 분석을 토대로 화면 설계를 하였다.

1) 화면 전환 스크립트

각각의 Scene을 화면전환을 하기위해 필요한 C#코드 이다. SceneManage.LoadScene (“화면명“)을 입력한다.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuSet : MonoBehaviour
{
    // Start is called before the first frame update

    public void OnclickNew()
    {
        SceneManager.LoadScene("Select");
    }
    public void OnclickQuit()
    {
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
#else
        Application.Quit();
#endif
    }
}
```

그림 16 화면 전환 스크립트

각각 OnclickNew를 클릭 시 다음 화면으로 전환되고 OnclickQuit를 클릭 시 게임이 종료함.

2) 캐릭터 선택 스크립트

그림 18에서 각 캐릭터를 열거형으로 선언하고 public static instance를 선언하여 다른 스크립트 에서 해당 함수나 변수를 접근할 수 있도록 허용하였다. 그림 17에서 각 캐릭터를 열거형으로 선언하고 GetComponent함수를 사용하여 캐릭터의 렌더링 이미지와 애니메이션을 불러오고 DataMgr의 캐릭터 변수를 불러왔다. 그리고 OnMouse UpAsButton함수를 작성하여 캐릭터를 선택하면 DataMgr에 해당 캐릭터가 저장되도록 하였습니다.

```
public class SelectChar : MonoBehaviour
{
    public Character character;
    Animator anim;
    SpriteRenderer sr;
    public SelectChar[] chars;

    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
        sr = GetComponent<SpriteRenderer>();
        if (DataMgr.instance.currentCharacter == character) OnSelect();
        else OnDeSelect();
    }

    private void OnMouseUpAsButton()
    {
        DataMgr.instance.currentCharacter = character;
        OnSelect();
        for (int i = 0; i < chars.Length; i++)
        {
            if (chars[i] != this) chars[i].OnDeSelect();
        }
    }
    void OnDeSelect()
    {
        anim.SetBool("run", false);
        sr.color = new Color(0.5f, 0.5f, 0.5f);
    }

    void OnSelect()
    {
        anim.SetBool("run", true);
        sr.color = new Color(1f, 1f, 1f);
    }
}
```

그림 17 캐릭터 선택 선택 불러오는 스크립트

```
public enum Character
{
    Archer, Knight, Wizard, Thief
}
public class DataMgr : MonoBehaviour
{
    public static DataMgr instance;
    private void Awake()
    {
        if (instance == null) instance = this;
        else if (instance != null) return;
        DontDestroyOnLoad(gameObject);
    }

    public Character currentCharacter;
}
```

그림 18 캐릭터 선택 스크립트(DataMgr)

```
public class Card : MonoBehaviour
{
    [SerializeField] SpriteRenderer card;
    [SerializeField] SpriteRenderer character;
    [SerializeField] TMP_Text nameTMP;
    [SerializeField] TMP_Text costTMP;
    [SerializeField] TMP_Text attackTMP;
    [SerializeField] TMP_Text healthTMP;
    [SerializeField] Sprite cardFront;
    [SerializeField] Sprite cardBack;

    public Item item;
    bool isFront;
    public PRS originPRS;
    public static int cost;

    public void Setup(Item item, bool isFront)
    {
        cost = item.cost;
        this.item = item;
        this.isFront = isFront;

        if (this.isFront)
        {
            character.sprite = this.item.sprite;
            nameTMP.text = this.item.name;
            costTMP.text = cost.ToString();
            //costTMP.text = this.item.cost.ToString();
            attackTMP.text = this.item.attack.ToString();
            healthTMP.text = this.item.health.ToString();
        }
        else
        {
            card.sprite = cardBack;
            nameTMP.text = "";
            costTMP.text = "";
            attackTMP.text = "";
            healthTMP.text = "";
        }
    }

    void OnMouseOver()
    {
        if (isFront)
            CardManager.Inst.CardMouseOver(this);
    }

    void OnMouseExit()
    {
        if (isFront)
            CardManager.Inst.CardMouseExit(this);
    }

    void OnMouseDown()
    {
        if (isFront)
        {
            CardManager.Inst.CardMouseDown(this);
        }
```

그림 19 카드 스크립트(Card)

```
void OnMouseUp()
{
    if (isFront)
    {
        CardManager.Inst.CardMouseUp();
    }
}
public void MoveTransform(PRS prs, bool useDotween, float dotweenTime = 0)
{
    if (useDotween)
    {
        transform.DOMove(prs.pos, dotweenTime);
        transform.DORotateQuaternion(prs.rot, dotweenTime);
        transform.DOScale(prs.scale, dotweenTime);
    }
    else
    {
        transform.position = prs.pos;
        transform.rotation = prs.rot;
        transform.localScale = prs.scale;
    }
}
```

그림 20 카드 스크립트(Card)

3) 카드 스크립트

그림 19-20에서는 카드의 체력, 공격력, 필요마나 텍스쳐와 카드UI 렌더링이미지를 불러오고 CardManager에서 해당 카드를 마우스로 클릭과 카드 드래그, 카드클릭 후 마우스에서 때는 동작의 함수를 불러온다.

```
public Item PopItem()
{
    if (itemBuffer.Count == 0)
        SetupItemBuffer();

    Item item = itemBuffer[0];
    itemBuffer.RemoveAt(0);
    return item;
}

void SetupItemBuffer()
{
    itemBuffer = new List<Item>(100);
    for (int i = 0; i < itemSO.items.Length; i++)
    {
        Item item = itemSO.items[i];
        for (int j = 0; j < item.percent; j++)
            itemBuffer.Add(item);
    }

    for (int i = 0; i < itemBuffer.Count; i++)
    {
        int rand = Random.Range(i, itemBuffer.Count);
        Item temp = itemBuffer[i];
        itemBuffer[i] = itemBuffer[rand];
        itemBuffer[rand] = temp;
    }
}
```

그림 21 카드관리 스크립트1 (CardManager)

3) 카드매니저 스크립트

카드매니저 스크립트에서는 카드의 정렬, 드래그,생성,추가등을 담당한다. SetupItem Buffer에서는 한장의 카드가 여러가지가 나오지 않도록 카드의 순서를 지정하고 Random.Range함수를 통해 무작위로 카드를 뽑도록 하였다.

```
void AddCard(bool isMine)
{
    var cardObject = Instantiate(cardPrefab, cardSpawnPoint.position, Utils.QI);
    var card = cardObject.GetComponent<Card>();
    card.Setup(PopItem(), isMine);
    (isMine ? myCards : otherCards).Add(card);

    SetOriginOrder(isMine);
    CardAlignment(isMine);
}

void SetOriginOrder(bool isMine)
{
    int count = isMine ? myCards.Count : otherCards.Count;
    for (int i = 0; i < count; i++)
    {
        var targetCard = isMine ? myCards[i] : otherCards[i];
        targetCard?.GetComponent<Order>().SetOriginOrder(i);
    }
}

void CardAlignment(bool isMine)
{
    List<PRS> originCardPRSs = new List<PRS>();
    if (isMine)
        originCardPRSs = RoundAlignment(myCardLeft, myCardRight, myCards.Count, 0.5f, Vector3.one * 1.9f);
    else
        originCardPRSs = RoundAlignment(otherCardLeft, otherCardRight, otherCards.Count, -0.5f, Vector3.one * 1.9f);

    var targetCards = isMine ? myCards : otherCards;
    for (int i = 0; i < targetCards.Count; i++)
    {
        var targetCard = targetCards[i];

        targetCard.originPRS = originCardPRSs[i];
        targetCard.MoveTransform(targetCard.originPRS, true, 0.7f);
    }
}
```

그림 23 카드관리 스크립트2

(CardManager)

AddCard함수는 IsMine일 경우에는 자신의 턴으로 카드 한장을 받고 IsMine이 아닐 경우에는 상대가 카드를 한 장을 받도록 하였다. SetOriginOrder에서는 카드를 정렬하도록 하였다.

```
public void CardMouseOver(Card card)
{
    if (eCardState == ECardState.Nothing)
        return;

    selectCard = card;
    //if (TurnManager.Inst.Currentcost < card.item.cost)
        //return;
    EnlargeCard(true, card);
}
public void CardMouseExit(Card card)
{
    EnlargeCard(false, card);
}

public void CardMouseDown(Card card)
{
    if (TurnManager.Inst.Currentcost < card.item.cost)
        return;
    if (eCardState != ECardState.CanMouseDrag)
        return;
    isMyCardDrag = true;
}

public void CardMouseUp()
{
    isMyCardDrag = false;

    if (eCardState != ECardState.CanMouseDrag)
        return;

    if (onMyCardArea)
        EntityManager.Inst.RemoveMyEmptyEntity();
    else
    {
        TryPutCard(true);
    }
}

void CardDrag()
{
    if (eCardState != ECardState.CanMouseDrag)
        return;
    if (!onMyCardArea)
    {
        selectCard.MoveTransform(new PRS(Utils.MousePos, Utils.QI, selectCard.originPRS.scale), false);
        EntityManager.Inst.InsertMyEmptyEntity(Utils.MousePos.x);
    }
}

void DetectCardArea()
{
    RaycastHit2D[] hits = Physics2D.RaycastAll(Utils.MousePos, Vector3.forward);
    int layer = LayerMask.NameToLayer("MyCardArea");
    onMyCardArea = Array.Exists(hits, x => x.collider.gameObject.layer == layer);
}
```

그림 24 카드관리 스크립트3

(CardManager)

eCardState를 통해서 카드의 활성화 또는 비활성화 시키고 CarMouseOver함수에는 카드에 마우스를올려놓았을 때 EnlargeCard를 True로 하여 카드를 확대하게하였다.

CarMouseExit함수에는 EnlargeCard를 false로 하여 카드 확대를 종료한다.

CardMouseDown에서는 마우스 클릭시 isMyCard Drag를 True로 하여 카드를 움직일 수 있게 하였고 CardMouseup에서는 isMyCardDrag를 false로 하여 내 카드의 드래그를 클릭 중인 손가락을 마우스에서 떨어뜨릴 경우에 카드 드래그를 비활성화였다.

```
void EnlargeCard(bool isEnlarge, Card card)
{
    if (isEnlarge)
    {
        Vector3 enlargePos = new Vector3(card.originPRS.pos.x, -4.8f, -10f);
        card.MoveTransform(new PRS(enlargePos, Utils.QI, Vector3.one * 3.5f), false);
    }
    else
        card.MoveTransform(card.originPRS, false);

    card.GetComponent<Order>().SetMostFrontOrder(isEnlarge);
}

void SetECardState()
{
    if (TurnManager.Inst.isLoading)
        eCardState = ECardState.Nothing;

    else if (!TurnManager.Inst.myTurn || myPutCount >= 2 || EntityManager.Inst.IsFullMyEntities)
        eCardState = ECardState.CanMouseOver;

    else if (TurnManager.Inst.myTurn && myPutCount == 0)
        eCardState = ECardState.CanMouseDrag;

    else if (TurnManager.Inst.Currentcost <= 0)
        eCardState = ECardState.CanMouseOver;
}
```

그림 25 카드관리 스크립트4

SetEcard State에서는 IsLoading일 경우 카드를 아무것도 할 수 없고 myPutCount는 카드를 낼 때마다 하나씩 증가해서 2장 이상을 냈을 경우에는 EcardState.CanMouseOver로 변경해서 카드를 확대해서 확인할 수 있게만 하였다. myPutCount==0 일경우에는 카드를 드래그 할 수 있고 Currentcost 현재마나가 0이하 일경우에도 확대만 가능하게 하였다.

```
void ShowTargetPicker(bool isShow)
{
    TargetPicker.SetActive(isShow);
    if (ExistTargetPickEntity)
        TargetPicker.transform.position = targetPickEntity.transform.position;
}

void SpawnDamage(int damage, Transform tr)
{
    if (damage <= 0)
        return;

    var damageComponent = Instantiate(damagePrefab).GetComponent<Damage>();
    damageComponent.SetupTransform(tr);
    damageComponent.Damaged(damage);
}
```

그림 26 유닛관리 스크립트

ShowTargetPicker는 보스나 적 유닛에 Target Picker를 활성화 하게 한다.SpawnDamage는 데미지가 0이상일때 데미지 텍스처를 불러온다.

```
public void EntityMouseDown(Entity entity)
{
    if (!CanMouseInput)
        return;

    selectEntity = entity;
}

public void EntityMouseUp()
{
    if (!CanMouseInput)
        return;

    // selectEntity, targetPickEntity 둘다 존재하면 공격한다. 바로 null, null로 만든다.
    if (selectEntity && targetPickEntity && selectEntity.attackable)
        Attack(selectEntity, targetPickEntity);

    selectEntity = null;
    targetPickEntity = null;
}

public void EntityMouseDrag()
{
    if (!CanMouseInput || selectEntity == null)
        return;

    // other 타겟엔티티 찾기
    bool existTarget = false;
    foreach (var hit in Physics2D.RaycastAll(Utils.MousePos, Vector3.forward))
    {
        Entity entity = hit.collider?.GetComponent<Entity>();
        if (entity != null && !entity.isMine && selectEntity.attackable)
        {
            targetPickEntity = entity;
            existTarget = true;
            break;
        }
    }
    if (!existTarget)
        targetPickEntity = null;
}
```

그림 27 유닛관리 스크립트(Entity 매니저)

Entity 스크립트는 카드를 내면 적장에 유닛이 소환되었을 때 들어가는 스크립트이다. OnTurnStarted에서는 IsMine == MyTurn으로 상대턴 상태를 내 턴으로 변경하고 카드를 내서 처음 소환되었을 때는 liveCount가 0으로 SleepParicle이 재생하면서 공격 할 수 없지만 한 턴이 지나 liveCount가 증가하만 공격할 수 있는 상태가 된다. IsMine일 경우에는 OnMouseDown, OnMouseUp, OnMouseDrag 함수를 사용할 수 있게 되고 유닛을 마우스로 드래그하거나 클릭 할 수 있게 된다.

```
void Attack(Entity attacker, Entity defender)
{
    // _attacker가 _defender의 위치로 이동하다 원래 위치로 온다, 이때 order가 높다
    attacker.attackable = false;
    attacker.GetComponent<Order>().SetMostFrontOrder(true);

    Sequence sequence = DOTween.Sequence()
        .Append(attacker.transform.DOMove(defender.originPos, 0.4f)).SetEase(Ease.InSine)
        .AppendCallback(() =>
        {
            attacker.Damaged(defender.attack);
            defender.Damaged(attacker.attack);
            SpawnDamage(defender.attack, attacker.transform);
            SpawnDamage(attacker.attack, defender.transform);
        })
        .Append(attacker.transform.DOMove(attacker.originPos, 0.4f)).SetEase(Ease.OutSine)
        .OnComplete(() => AttackCallback(attacker, defender));
}

void AttackCallback(params Entity[] entities)
{
    // 죽을 사람 골라서 죽음 처리
    entities[0].GetComponent<Order>().SetMostFrontOrder(false);

    foreach (var entity in entities)
    {
        if (!entity.isDie || entity.isBossOrEmpty)
            continue;

        if (entity.isMine)
            myEntities.Remove(entity);
        else
            otherEntities.Remove(entity);

        Sequence sequence = DOTween.Sequence()
            .Append(entity.transform.DOShakePosition(1.3f))
            .Append(entity.transform.DOScale(Vector3.zero, 0.3f)).SetEase(Ease.OutCirc)
            .OnComplete(() =>
            {
                EntityAlignment(entity.isMine);
                Destroy(entity.gameObject);
            });
    }
    StartCoroutine(CheckBossDie());
}

IEnumerator CheckBossDie()
{
    yield return delay2;

    if (myBossEntity.isDie)
        StartCoroutine(GameManager.Inst.GameOver(false));

    if (otherBossEntity.isDie)
        StartCoroutine(GameManager.Inst.GameOver(true));
}
```

그림 28 유닛관리 스크립트2(Entity 매니저)

Attack에서는 attacker가 공격하는 유닛 defender가 공격받는 유닛으로 attacker공격을 주면 해당 공격력만큼 데미지를 주고 defender의 공격력 만큼 데미지를 받는다. 공격 시에는 attacker의 위치가 defender의 위치로 이동후에 데미지를 주고 돌아오게 된다.

```
public void InsertMyEmptyEntity(float xPos)
{
    if (isFullMyEntities)
        return;

    if (!ExistMyEmptyEntity)
        myEntities.Add(myEmptyEntity);

    Vector3 emptyEntityPos = myEmptyEntity.transform.position;
    emptyEntityPos.x = xPos;
    myEmptyEntity.transform.position = emptyEntityPos;

    int _emptyEntityIndex = MyEmptyEntityIndex;
    myEntities.Sort((entity1, entity2) => entity1.transform.position.x.CompareTo(entity2.transform.position.x));
    if (MyEmptyEntityIndex != _emptyEntityIndex)
        EntityAlignment(true);
}

public void RemoveMyEmptyEntity()
{
    if (!ExistMyEmptyEntity)
        return;

    myEntities.RemoveAt(MyEmptyEntityIndex);
    EntityAlignment(true);
}

public bool SpawnEntity(bool isMine, Item item, Vector3 spawnPos)
{
    if (isMine)
    {
        if (isFullMyEntities || !ExistMyEmptyEntity)
            return false;
    }
    else
    {
        if (isFullOtherEntities)
            return false;
    }

    var entityObject = Instantiate(entityPrefab, spawnPos, Utils.QI);
    var entity = entityObject.GetComponent<Entity>();

    if (isMine)
        myEntities[MyEmptyEntityIndex] = entity;
    else
        otherEntities.Insert(Random.Range(0, otherEntities.Count), entity);

    entity.isMine = isMine;
    entity.Setup(item);
    EntityAlignment(isMine);

    return true;
}
```

그림 29 유닛관리 스크립트3(Entity 매니저)

EntityManager는 카드를 내서 적장이 소환될 수 있도록 하는 스크립트이다.

EntityAlignment는 유닛이 소환되면 중앙서부터 소환하도록 위치를 잡아주는 함수이다.

InsertMyEmptyEntity는 IsFullEntitles 필드에 화면에 찼을 경우에는 return하고 아닐경우에는 유닛 추가한다.

EmptyEntityIndex가 없을 경우에는 추가해준다. EntityAlignment를 True로 하여 소환위치를 지정한다.

RomoveMyEmptyEntity함수는 체력이 모두 소모되어 EmptyEntityIndex가 사라졌을 때 유닛 제거하고 내 필드에 다른 유닛이 있을 경우에 제거된 유닛 위치에 온다.

SpawnEntity는 유닛이 소환할 수 있도록 하는 함수로 IsMine이나 그 외 상대 턴일 경우 내 필드가 유닛으로 다 찼을 때나 MyEmptyEntity 가 없을 경우 return false하고 Item매개변수를 통해 해당 entityPrefab을 불러온다. otherEntities 상대 유닛을 경우에는 랜덤으로 불러온다.

```
public void Damaged(int damage)
{
    if (damage <= 0)
        return;

    GetComponent<Order>().SetOrder(1000);
    damageTMP.text = $"-{damage}";
    if(damage == 14)
    {
        Instantiate(skill1, transform.position, transform.rotation);
    }
    else if (damage == 15)
    {
        Instantiate(skill2, transform.position, transform.rotation);
    }
    else if(damage == 22)
    {
        Instantiate(skill3, transform.position, transform.rotation);
    }
    else if(damage == 25)
    {
        Instantiate(skill4, transform.position, transform.rotation);
    }
    else if(damage == 19)
    {
        Instantiate(skill5, transform.position, transform.rotation);
    }
    else if(damage == 20)
    {
        Instantiate(skill6, transform.position, transform.rotation);
    }
    else if(damage == 18)
    {
        Instantiate(skill7, transform.position, transform.rotation);
    }
    else if (damage == 21)
    {
        Instantiate(skill8, transform.position, transform.rotation);
    }
    else if (damage == 35)
    {
        Instantiate(skill9, transform.position, transform.rotation);
    }
    else
    {
        Instantiate(skill10, transform.position, transform.rotation);
    }
    Sequence sequence = DOTween.Sequence()
        .Append(transform.DOScale(Vector3.one + 1.8f, 0.5f).SetEase(Ease.InOutBack))
        .AppendInterval(1.2f)
        .Append(transform.DOScale(Vector3.zero, 0.5f).SetEase(Ease.InOutBack))
        .OnComplete(() => Destroy(gameObject));
}
```

그림 30 데미지 스크립트

Damaged함수에서는 스킬로 피해를 줄 때마다 스킬 Prefab을 읽어와서 스킬 이펙트를 발동하고 데미지 문구가 뜬 후에 데미지 문구 gameObject를 Destroy를 사용하여 파괴하였다.

```
void GameSetup()
{
    cost1 = 1;
    Currentcost = 1;
    OtherCurrentcost = 1;
    stack = 0;
    if (fastMode)
        delay05 = new WaitForSeconds(0.05f);

    switch (eTurnMode)
    {
        case ETurnMode.Random:
            myTurn = Random.Range(0, 2) == 0;
            break;
        case ETurnMode.My:
            myTurn = true;
            break;
        case ETurnMode.Other:
            myTurn = false;
            break;
    }
}

public IEnumerator StartGameCo()
{
    GameSetup();
    isLoading = true;

    for (int i = 0; i < startCardCount; i++)
    {
        yield return delay05;
        OnAddCard?.Invoke(false);
        yield return delay05;
        OnAddCard?.Invoke(true);
    }
    StartCoroutine(StartTurnCo());
}

IEnumerator StartTurnCo()
{
    isLoading = true;
    if (myTurn)
    {
        GameManager.Inst.Notification("나의 턴");
    }
    costTMP.text = Currentcost + "/" + cost1;
    yield return delay07;
    OnAddCard?.Invoke(myTurn);
    yield return delay07;
    isLoading = false;
    OnTurnStarted?.Invoke(myTurn);
}
```

그림 31 턴 매니저

GameSetup은 게임 시작으로 Cost1(총 마나), CurrentCost(현재 마나)로 각각 턴이 종료될 때마다 증가한다. 게임 시작 시에는 Switch에서 Case ETurnMode.Random에서 랜덤으로 Other 상대 턴부터 시작하거나 아니면 My 내 턴부터 시작하게 된다. IEnumerator는 void Start에 해당 함수를 넣을 경우에는 앞에 선언해야 한다. StartGamCo 게임 시작되면 카드를 배분받고 턴 종료 버튼이 활성화된다. StartTurnco는 나의 턴이 시작되고 현재 마나와 총 마나가 표시된다. 카드 배분이 끝나게 되면 OnTurnStared가 발동한다. 시작하게 된다.

```
public void EndTurn()
{
    myTurn = !myTurn;
    if (stack == 1)
    {
        cost1 += 1;
        Currentcost = cost1;
        stack = 0;
    }
    else if (stack == 0)
    {
        stack += 1;
    }
    if (cost1 >= 10)
    {
        cost1 = 10;
    }
    OtherCurrentcost = cost1;
    costTMP2.text = OtherCurrentcost + "/" + cost1;
    StartCoroutine(StartTurnCo());
}
```

그림 32 턴 매니저

EndTurn은 턴종료 클릭시 현재마나와 총마나 증가하게 된다.

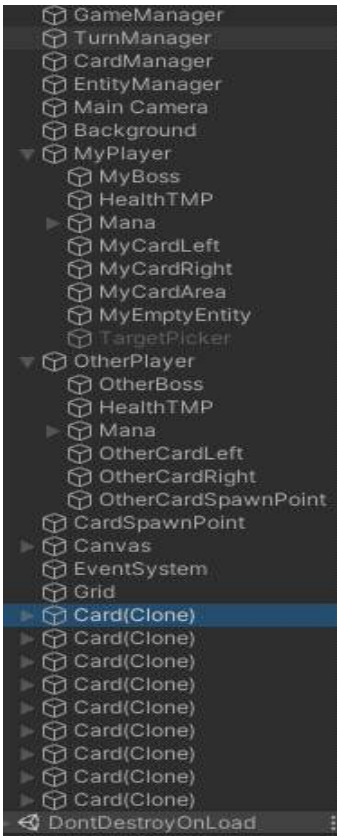

그림 33 게임시작시 오브젝트

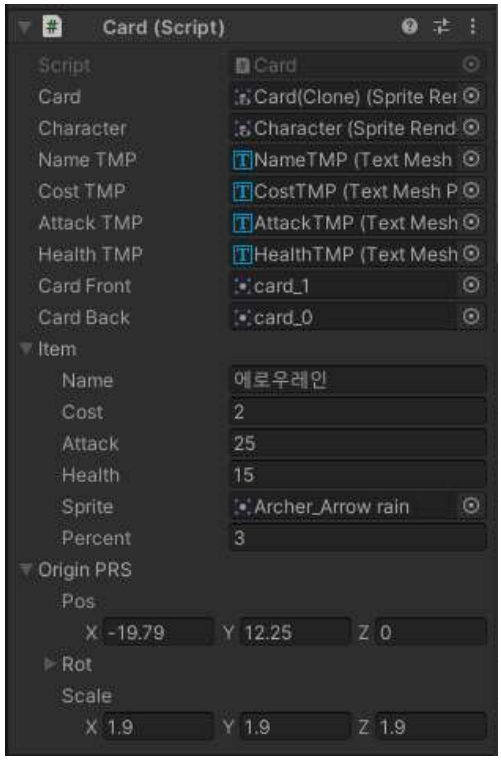

그림 34 카드 스크립트

Ⅴ. 향후 방향 및 결론

단판으로 구현되어 있는 스테이지와 카드의 종류를 추가하고 프로토타입으로 벨런싱 테스트 후 카드 종류를 증가시켜서 더 전략적으로 게임을 발전시키고 캐릭터마다 드로우 되는 카드를 다르게 설정하여 캐릭터의 특성을 살려 캐릭터 선택의 즐거움을 살리며 난이도 시스템을 도입하여 남녀노소 쉽게 접근할 수 있는 게임을 계획 하고 있다.

참고문헌

1. youtube / https://www.youtube.com/watch?v=Xo7EEegTUfE
2. asset / https://assetstore.unity.com/?locale=ko-KR
3. blog / Oct 25 2013 / https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=progagmer&logNo=197501681
4. Unity Engine
5. Visual Studio
6. https://www.photopea.com/