PORTFOLIO

문지현, Jihyun Moon

Github: github.com/solidcellaMoon
개인 블로그: star-crab.tistory.com

PORTFOLIO

CONTENTS

PORTFOLIO

001 갓버드 (경영 시뮬레이션 + 횡스크롤 게임)

001 프로젝트 개요

002 프로젝트 내의 역할

003 미니게임 구현

- 플레이어 움직임 로직
- 버튼 작동 과정
- Item, enemy 생성 & 관리

004 UI 배치 & 기능 구현 일부

005 전체 컨텐츠 Scene 연결 과정

PORTFOLIO

001 갓버드 (경영 시뮬레이션 + 횡스크롤 게임)

프로젝트 개요

게임개발 동아리에서 활동하며 제작한 프로젝트입니다.

장르: 간단한 경영 시뮬레이션 + 횡스크롤 게임

팀 인원: 5명

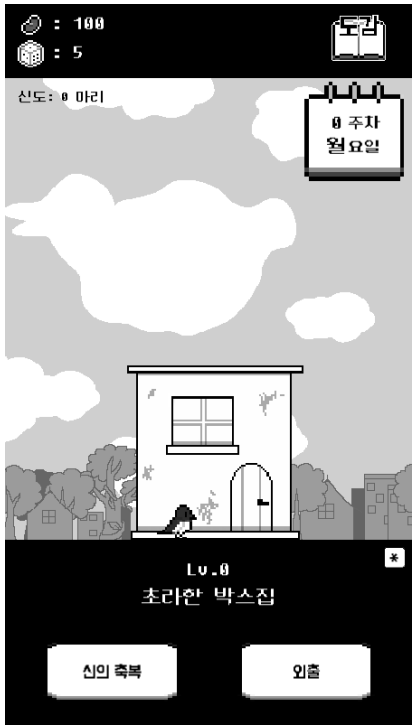
개발 환경: Unity 2D

개발 기간: 2019.05.23 ~ 2019.08.23

소개 영상: youtu.be/o3IrRYedrtE
Github: https://github.com/solidcellaMoon/GodBird

프로젝트 내의 역할

1. 횡스크롤 미니게임 전체 구현
2. 전체 컨텐츠 Scene 통합, UI 배치와 기능 구현
3. 팀장으로서 전체 개발 일정 관리



횡스크롤 미니게임 전체 구현

전체 컨텐츠의 UI 배치와 기능 구현

미니게임 – 플레이어 움직임 로직

PlayerMove.cs

```
//플레이어 화면 이탈 방지---
void moveRange(){
    Vector3 pos = Camera.main.WorldToViewportPoint(transform.position);

    if(pos.y < 0.2f) pos.y = 0.2f; // 화면 하단으로 이탈했을 때
    if(pos.y > 0.88f) pos.y = 0.88f; // 화면 상단으로 이탈했을 때

    // 위치 보정
    transform.position = Camera.main.ViewportToWorldPoint(pos);
}
```

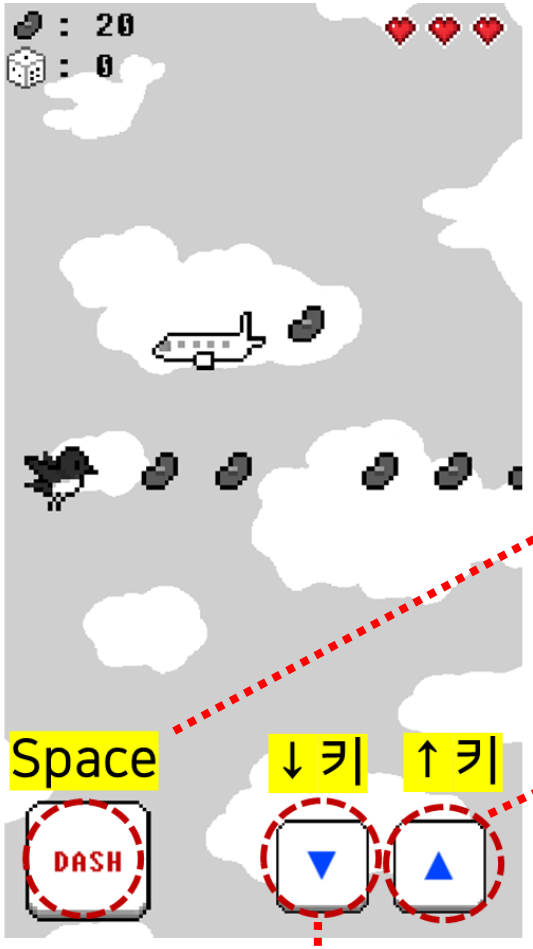
플레이어의 기본적인 움직임과
화면 이탈 여부를 제어합니다.

dashAttack.cs

```
void Update()
{
    // dash버튼 눌렀을 때 ---
    if(inputDashDwn || Input.GetKeyDown("space")){
        PlayerMove.superTime = true; //무적타임 시작
        transform.Translate(0.9f,0,0); //x축에서 우측으로 이동
        dashBgm.Play(); //효과음 실행
        StartCoroutine(waitTime(0.2f)); //대기 코루틴
        inputDashDwn = false; //DashDwn 끝
    }

    // dash버튼 떼었을 때 ---
    if(inputDashUp || Input.GetKeyUp("space")){
        transform.Translate(-0.9f,0,0); //x축 좌측으로 이동(좌표 원상복귀)
        PlayerMove.superTime = false; //무적타임 끝
        inputDashUp = false; //DashUp 끝
    }
```

Dash(공격) 버튼을 눌렀을 때
공격 모션과 충돌 판정을 시행합니다.

미니게임 – 버튼 작동 과정




```
public void Start(){ // 캐릭터 오브젝트, 동작 관리 스크립트 불러오기
    player = GameObject.Find("Player");
    playerScript = player.GetComponent<PlayerMove> ();
    dashScript = player.GetComponent<dashAttack> ();
}

public void Update(){ // 공격 - space 버튼이 눌렸는지 확인
    if(Input.GetKeyDown("space")) DashPressed();
    if(Input.GetKeyUp("space")) DashOff();
}
```

```
// 공격 모션 - space바가 눌렸는지 여부를 bool형으로 확인
public void DashPressed(){ // space 버튼을 눌렀을 때
    dashScript.inputDashDwn = true;
}

public void DashOff(){ // space 버튼을 떼었을 때
    dashScript.inputDashUp = true;
}
```

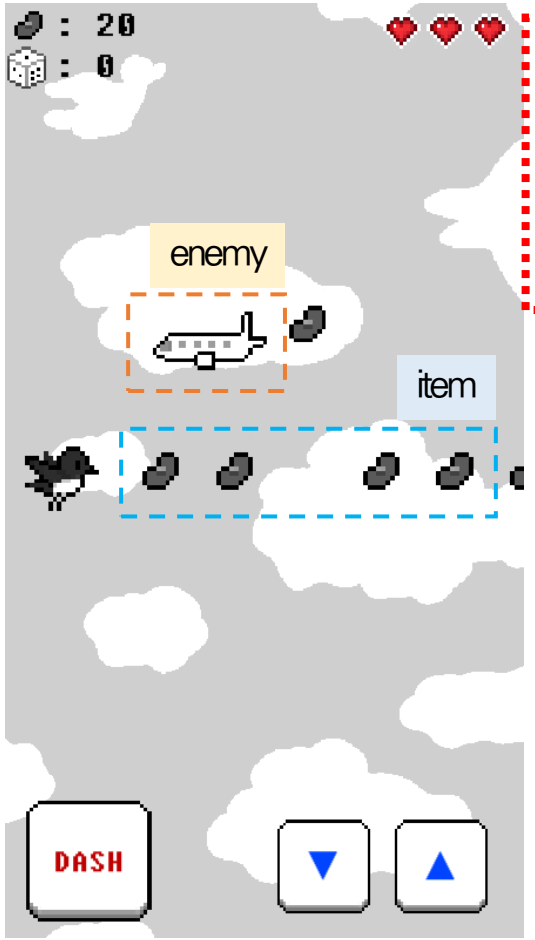
```
// 상승 모션 - up버튼을 눌렀는지 여부를 확인---
0 references
public void UpPressed(){…
0 references
public void UpOff(){…

// 하강 모션 - down버튼을 눌렀는지 여부를 확인---
0 references
public void DownPressed(){…
0 references
public void DownOff(){…
```

buttonManager.cs

Start()에서 필요한 오브젝트를 불러오고,
Update()에서 매 프레임마다
버튼이 눌렸는지 확인합니다.

해당 버튼이 눌렸을 때 정보를
Bool type으로 표현합니다.

미니게임 – item, enemy 생성



Item과 enemy 모두 Prefab으로 생성, 관리합니다.
Random을 활용해 확률적으로 특정 아이템을 생성합니다.

```
void makeItem(int num){
    for(int i = 0; i<num; i++){
        //  y축 좌표 조정
        if(py > 3.5f) py = (int) Random.Range(2,3);
        if(py < -3f) py = (int) Random.Range(-3,-1);

        // 0~1000까지의 수를 뽑아, 확률적으로 아이템 생성
        int itemType = Random.Range(0,1000);

        //주사위 생성 ---
        if(itemType > 970) item = Instantiate(prefab[1]) as GameObject;

        //콩 생성 ---
        else if(itemType > 1) item = Instantiate(prefab[0]) as GameObject;

        //하트 생성 ---
        else {
            if(heartLimit > 0){
                // 하트 제한 횟수(3번)까지만 하트를 생성
                item = Instantiate(prefab[2]) as GameObject;
                heartLimit --;
                }
            else item = Instantiate(prefab[1]) as GameObject;
            // 제한 횟수를 넘으면, 하트가 아니라 주사위 아이템을 생성
        }

        //아이템 표시 위치 조정
        itemPos();
```

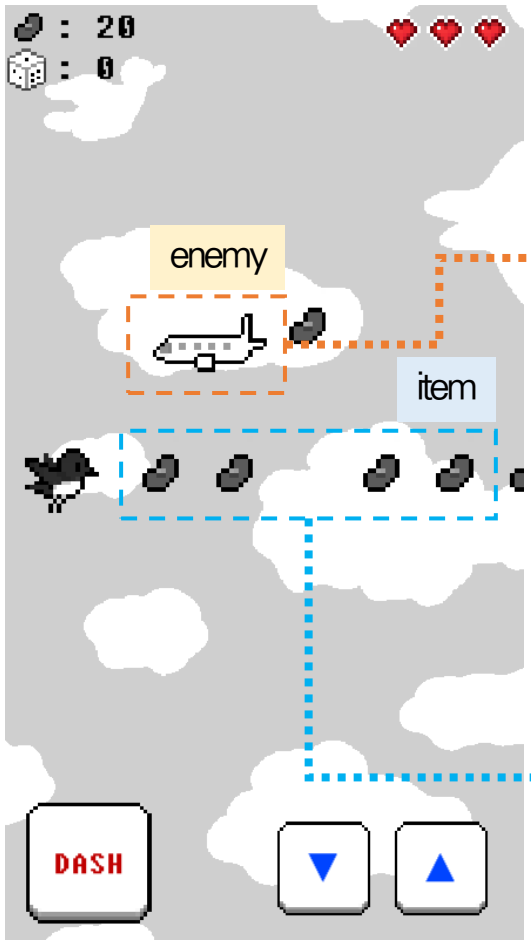
ItemGenerator.cs

```
//itemPos(): 생성좌표의 범위를 랜덤으로 조정 ---
void itemPos(){
    int pyType = Random.Range(0,141); // 랜덤으로 y좌표 위치를 선택

    px += 1; // x좌표는 한칸씩 뒤로

    if(pyType <= 90) py += 0; // 일직선으로 생성
    else if(pyType <= 105) py += 1; // 한칸 위에 생성
    else if (pyType <= 120) py -= 1; // 한칸 아래에 생성
    else if (pyType <= 130) py += 2; // 두칸 위에 생성
    else py -= 2; // 두칸 아래에 생성

}
```

ItemGenerator.cs

생성 위치도 랜덤적으로 조정합니다.
이렇게 하여 매 스테이지마다 아이템 배치가 바뀝니다.
Enemy 생성도 동일한 방식이며, 일부 코드를
기능에 맞추어 수정/추가 했습니다.

미니게임 – item, enemy 관리



enemyController.cs

```
void Update()
{
    // 게임 오버시 삭제
    if(lifeManager.lifeNum == 0) Destroy(gameObject);
    else{
        if(!isDie){
        switch(enemyType){
            case 1: // x축 방향으로 이동한다.
            transform.Translate(xSpeed*Time.deltaTime,0,0); break;

            case 2: // 지그재그로 이동한다.
            // delta, sinSpeed
            transform.Translate(xSpeed*Time.deltaTime,
            delta*Mathf.Sin(Time.time*sinSpeed),0); break;

            case 3: // 플레이어 방향으로 돌진한다.
            // ySpeed, playerPos, thisPos
            Type3Move(); break;
            }
        }
        else Die();
    }

// 범위 이탈시 삭제
    if(transform.position.x < -4.5f
    || transform.position.y < -8f) Destroy(gameObject);
}
```

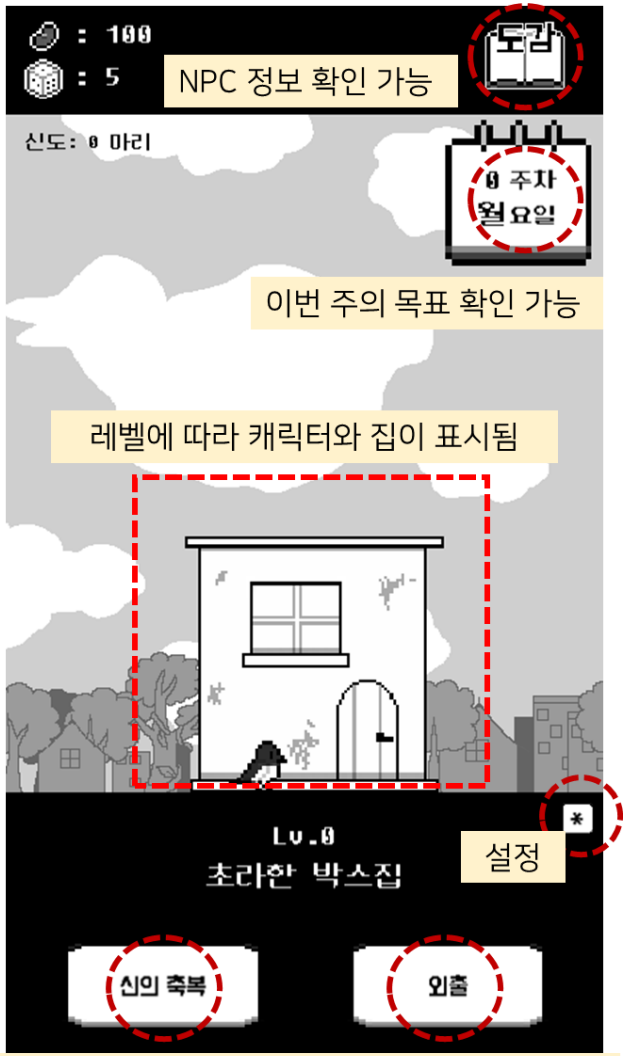
Update()에서 각 오브젝트의 동작을 관리합니다.
추가적인 기능은 함수형태로 구현한 후 불러옵니다.

onCollisionEnter2D/onTriggerEnter2D로
플레이어와 충돌 판정을 구현했습니다.

itemController.cs

```
void Update()
{
    if(lifeManager.lifeNum == 0) Destroy(gameObject); // 게임 오버시 삭제
    else{
    // 아이템은 화면에서 오른쪽->왼쪽 으로 이동
    transform.Translate(-1*speed*Time.deltaTime,0,0);

    // 화면 밖을 넘어가면 삭제
    if(transform.position.x < -4f) Destroy(gameObject);
    if(transform.position.y < -4f) Destroy(gameObject);
    }
}

0 references
void OnCollisionEnter2D(Collision2D collision){
    // 플레이어와 충돌하면 종류별 점수를 기록하고 삭제
        if(itemType == 0) scoreManager.beanScore += 1;
        if(itemType == 1) scoreManager.diceScore += 1;
        if(itemType == 2) lifeManager.lifeNum += 1;
        Destroy(gameObject);
}
```

메인화면 – UI 배치 & 기능 구현 일부



신의 축복(미니게임), 외출(대화): 메인 컨텐츠

각 UI를 표시하는 오브젝트를 디자인, 배치해둔 뒤
관련 Manager에서 필요한 정보를 받아와 표시합니다.

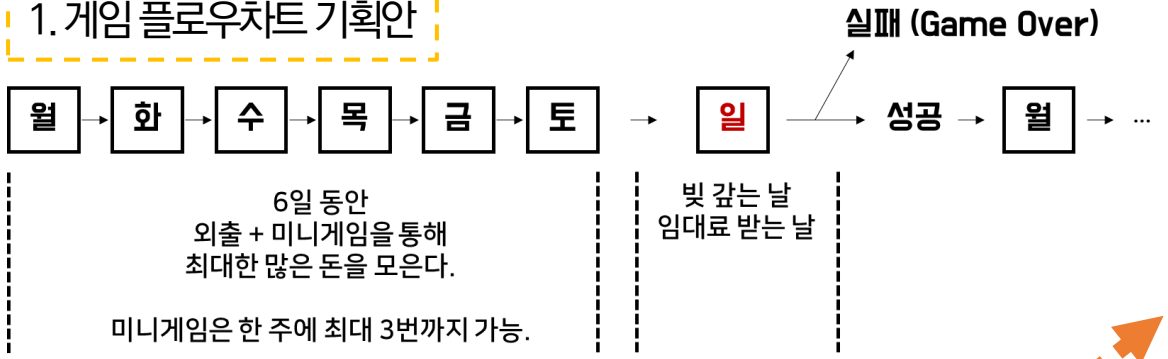
캐릭터별 능력, 레벨 등의 정보를 Manager 스크립트에 저장합니다.

npcManager.cs

```
public class npcManager : MonoBehaviour
{
    7 references
    public static bool[] npcList = new bool [6];
    // 0: 까마귀 / 1: 병아리 / 2: 비둘기 / 3: 펭귄 / 4: 앵무새 / 5: 어깨걸이
    1 reference
    public static int[] npcAbilty = {2,2,3,4,5,5};
    // 각 npc 별 신도수 증가 수치
    10 references
    public static int[] npcGage = {0,0,0,0,0,0};
    // 호감도 수치
    2 references
    public static int[] npcEnc = {0,0,0,0,0,0};
    // npc별 만남 횟수
```

전체 컨텐츠 Scene 연결 과정

1. 게임 플로우차트 기획안



2. 팀원과 협업하여 전체 컨텐츠를 Scene별로 구현

3. Scene 전환 스크립트로 컨텐츠 흐름 연결

SceneChange.cs

```
0 references
public void ChangeToIntro(){
    SceneManager.LoadScene("IntroScene");
}

0 references
public void ChangeToMiniGame(){
    if(dateManager.gameRetry > 0) SceneManager.LoadScene("MiniGame");
    else PanelOpen(0,true);
}

0 references
public void ChangeToMain(){
    if(SceneManager.GetActiveScene().name == "eventScene"
    && dateManager.dateNum % 7 == 6)  SceneManager.LoadScene("SundayScene");
    else SceneManager.LoadScene("MainScreen");
}

0 references
public void ChangeToDateEvt(){
    // 미니게임을 끝내고 난 뒤 실행
    if(SceneManager.GetActiveScene().name == "MiniGame") ScoreLoad();
    SceneManager.LoadScene("eventScene");
}

0 references
public void ChangeToList(){...

0 references
public void CloseList(){...

0 references
public void ChangeToTalk(){...
```

특정 Scene으로 이동하는 함수

Scene 전환을 관리하는 스크립트에
함수형태로 구현합니다.
각 컨텐츠 Scene에 필요한 시점/버튼에
해당 함수를 불러옵니다.

PORTFOLIO

002 Zepetarot (네이버 ZEPETO API 활용 운세 뽑기)

001 프로젝트 개요

002 프로젝트 내의 역할

003 뽑기 결과 화면 구현

- CSV 데이터 파일
- 애니메이션 구현

PORTFOLIO

002 Zepetarot (네이버 ZEPETO API 활용 운세 뽑기)

프로젝트 개요

Junction 2020 API 해커톤에서 제작한 프로젝트.
자신의 ZEPETO 캐릭터가 그려진 카드를 뽑을 수 있습니다.

장르: 2D 랜덤 뽑기 게임

팀 인원: 3명

개발 환경: Unity 3D

개발 기간: 2020.10.09 ~ 2020.10.11

소개 영상: youtu.be/B2TF5Y0mzic
Github: https://github.com/dahaelee/Zepetarot

프로젝트 내의 역할

1. 기획 참여, 그래픽 소스 제작
2. 뽑기 결과 화면 구현

기획 참여 & 그래픽 소스 제작

뽑기 결과 화면 구현

뽑기 결과 화면 – csv 데이터 파일

cardData.csv

API 호출에 필요한 정보를 포함한 데이터를 저장합니다.

cardNUM	cardName	boothName	posX	posY	script
0	마법사	287	0	0	오늘은 전반적으로 운이 좋을 것 같네요!</ br> (이하생
1	여사제	522	0	0	요즘 고민이 있으신가요? 차분히 생각하는 시간을 갖는게 좋겠네요.</ br> (이하생략)
2	황제	461	0	0	오늘은 무엇을 할지 계획을 세워보세요! </ br> (이하생
3	힘	400	0	0	오늘 당신의 키워드는 '외유내강'!</ br> (이하생략)
4	은둔자	525	0	0	요즘 피곤한 일들이 많지 않나요?</ br> (이하생략)

CSV 파일을 프로젝트 상에 불러옵니다.

api.cs

```
void Start()
{
    // 데이터를 저장한 csv 불러오기
    List<Dictionary<string,object>> data = CSVReader.Read("cardData");
    cardNum = res_CardManager.cardNum;
    //csv에 저장한 boothName의 이미지를 ZEPETO API에서 불러옴
    result = Post((int)data[cardNum]["boothName"]);
}
```

해당하는 API의 정보를 불러옵니다.

뽑기 결과 화면 – csv 데이터 파일

cardData.csv

API 호출에 필요한 정보를 포함한 데이터를 저장합니다.

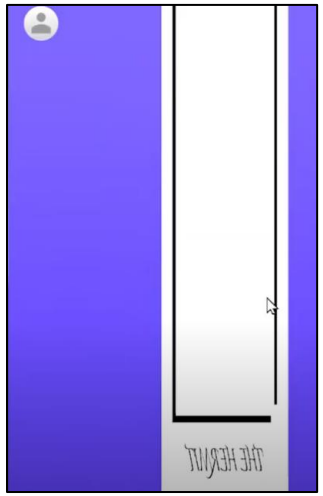
cardNUM	cardName	boothName	posX	posY	script
0	마법사	287	0	0	오늘은 전반적으로 운이 좋을 것 같네요!</ br> (이하생
1	여사제	522	0	0	요즘 고민이 있으신가요? 차분히 생각하는 시간을 갖는게 좋겠네요.</ br> (이하생략)
2	황제	461	0	0	오늘은 무엇을 할지 계획을 세워보세요! </ br> (이하생
3	힘	400	0	0	오늘 당신의 키워드는 '외유내강'!</ br> (이하생략)
4	은둔자	525	0	0	요즘 피곤한 일들이 많지 않나요?</ br> (이하생략)

Res_cardManager.cs

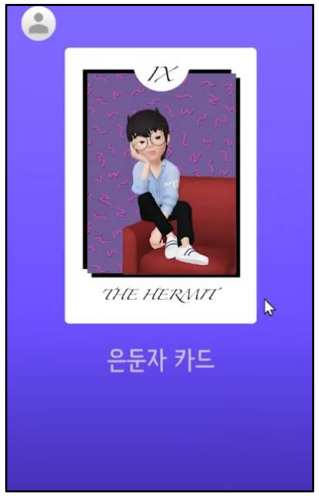
```
void Start()
{
    UIbackground.gameObject.SetActive(false);
    UImanager.allUIoff();
    hashcode.GetComponent<Text>().text = user.hashcode.ToString(); // 해시코드 변환

    //CSV 파일 불러오기
    List<Dictionary<string,object>> data = CSVReader.Read ("cardData");

    //랜덤으로 카드종류 정하고 배경 출력
    card_bgrd.sprite = bgrd_list[cardNum];

    // 설명 출력
    head.text = data[cardNum]["cardName"] + " 카드";
    script.text = (string)data[cardNum]["script"];

}
```

CSV 파일에 저장된 정보를 불러와 화면 상에 출력합니다.

뽑기 결과 화면 – 애니메이션

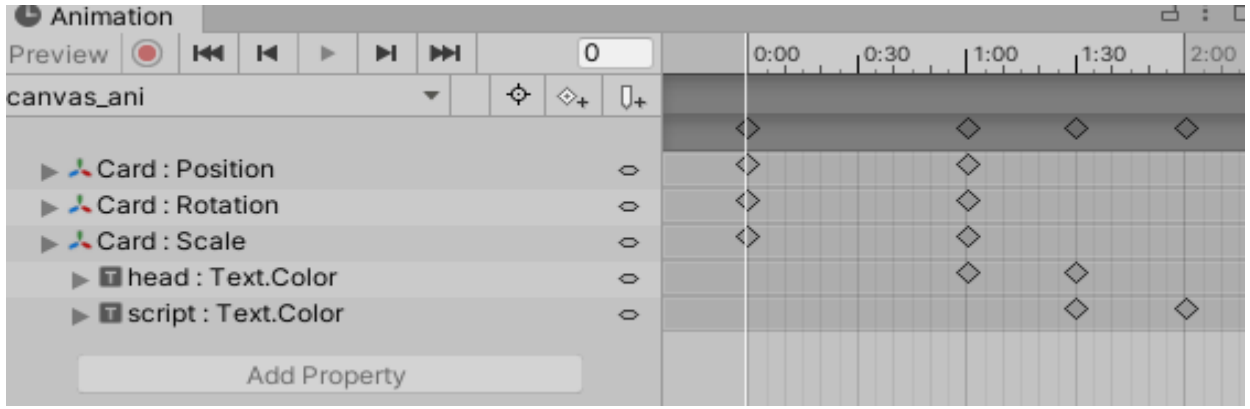


빈 화면에 카드가 날아오며 축소됨



카드 이미지, 이름 표시

운세 설명 표시

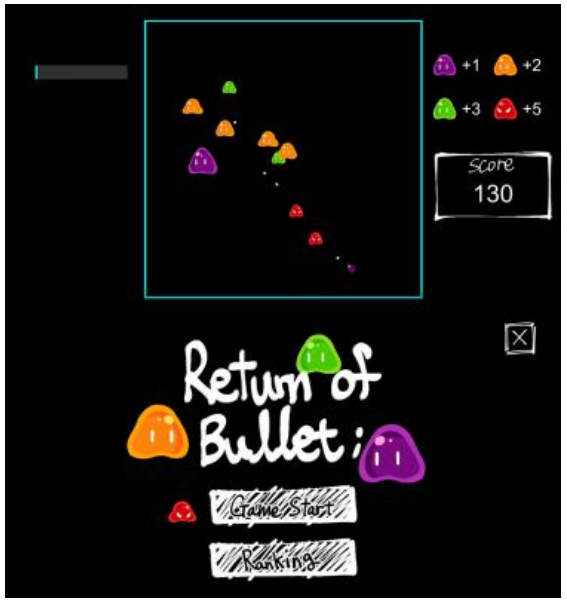


Animator를 사용하여 필요한
애니메이션 효과를 구현합니다.

PORTFOLIO

003 소규모 프로젝트 요약

2018년부터 약 2년간 교내 게임개발 동아리에서 활동하면서
게임 개발 관련 스터디, 소규모 프로젝트를 다수 진행했습니다.
아래 두 게임은 당시 제작했던 게임 중 일부입니다.



영상: youtu.be/5cfPvvJ-4rI

영상: youtu.be/2Hrf9bOrgjs

감사합니다

문지현, Jihyun Moon