게임 포트폴리오

윤찬호

CONTENTS

A 유니티 RPG

B PangPang

C 모두의 게임

CONTENTS

01. 게임소개

제작기간 : 2019.09.02 ~ 2019.11.14

1.유니티 RPG

이 포트폴리오는 유니티를 처음 접하고나서 유니티의 다양한 기능을 사용해보고 게임에 쓰이는 여러가지 기능을 직접 구현하는 것에 목적을 가지고 제작했습니다.

02. 구현설명

1) 미니맵

던전이나 필드로 이동할 경우 화면 상단에 미니맵이 표시 됩니다.
미로 형태를 띄는 씬은 미니맵이 표시되지 않도록 하였습니다.

미니맵을 위한 카메라를 생성한 후, Render Texture를 추가해 앞서 미니맵을 위해 만든 카메라가 보고있는 디스플레이를 Render Texture 에 그려줍니다. Raw Image를 생성 한 후 Render Texture에 그려진 그림을 Raw Image에 그려주게 했습니다.

02. 구현설명

2)아이템매니저

아이템 매니저는 게임 실행 시 외부 CSV파일을 읽어와 List로 미리 로드 해 놓습니다.

List에 저장 할 때는 따로 정의해 놓은 Item 클래스를 사용해 List<Item> 형태로 저장합니다.

플레이어의 인벤토리, NPC의 상점에 아이템을 추가할 때 List에 저장된 아이템 정보를 읽어와 오브젝트를 추가하도록 했습니다.

02. 구현설명

2)아이템매니저

```
public class ItemManage : MonoBehaviour
{
    public static ItemManage instance;
    private TextAsset ItemDataText;
    private PlayerInfo playerinfo;
    private GameObject Tab1;
    private GameObject Tab2;
    private List<Dictionary<string, object>> ItemTable;
    private List<Item> ItemDataList = new List<Item>();
    internal int ItemMax;

    private void Awake()
    {
        instance = this;
        SaveItemInfo();
        SaveJsonFile();
        Tab1 = GameObject.Find("UI Root").transform.Find("MENUButton")
             .Find("MenuChild").Find("InvenButton").Find("Inventory")
             .Find("Tab1").Find("Tab1Scroll").Find("Scroll View1").Find("Tab1Grid").gameObject;
        Tab2 = GameObject.Find("UI Root").transform.Find("MENUButton")
             .Find("MenuChild").Find("InvenButton").Find("Inventory")
             .Find("Tab2").Find("Tab2Scroll").Find("Scroll View2").Find("Tab2Grid").gameObject;
        ItemMax = ItemTable.Count;
        playerinfo = GameObject.Find("Player").GetComponent<PlayerInfo>();
    }

    private void SaveItemInfo()
    {
        ItemTable = CSVReader.Read("Data/ItemTable");
        for(int i =0; i < ItemTable.Count; i++)
        {
            ItemDataList.Add(new Item((int)ItemTable[i]["ID"],
                (string)ItemTable[i]["ItemName"],
                (int)ItemTable[i]["SellPrice"]));
        }
    }
```

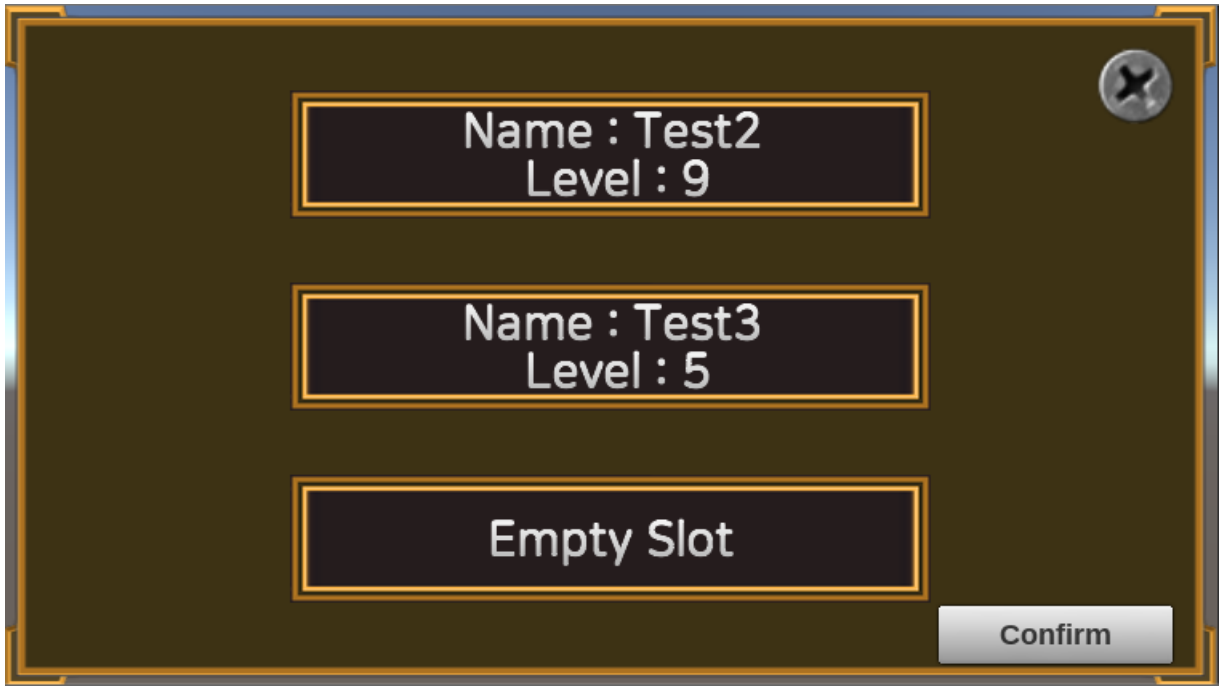
```
public class Item
{
    public int ID;
    public string ItemName;
    public int SellPrice;

    public Item(int id = 0, string itemname = "", int price = 0)
    {
        ID = id;
        ItemName = itemname;
        SellPrice = price;
    }
}
```

A	B	C
ID	ItemName	SellPrice
1	Gold	0
2	Shoulder	100
3	Boots	100
4	HPPortion	50
5	MPPortion	50

CSV 파일에서 아이템 정보를 읽어와 List<Item> 형태로 저장하도록 만들었습니다.

02. 구현설명

3)저장,불러오기 매니저



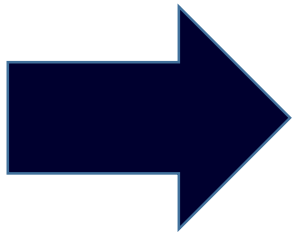
플레이 한 게임을 저장, 불러올 수 있는 기능입니다.
데이터를 저장, 불러오기 할 때는 Json을 사용해 관리하고,
게임을 처음 실행 할 때 미리 사용할 파일들을 생성한 후 저장이나 불러오기를 실행할 때 접근하도록 했습니다.
플레이어의 인벤토리에 있는 아이템들은 Save 파일 내에 슬롯 번호 별로 아이템 저장 파일의 경로를 읽어
각각 파일을 생성해 저장하도록 했습니다.

02. 구현설명

3) 저장,불러오기 매니저


```
{
  "number": 1,
  "SaveFlag": true,
  "ItemDataPath": "Data/SaveItemFile1",
  "data": {
    "Name": "Test2",
    "Class": "Warrior",
    "Level": 9,
    "Gold": 2686,
    "Exp": 750.0
  }
},
{
  "number": 2,
  "SaveFlag": true,
  "ItemDataPath": "Data/SaveItemFile2",
  "data": {
    "Name": "Test3",
    "Class": "Warrior",
    "Level": 5,
    "Gold": 1380,
    "Exp": 100.0
  }
},
{
  "number": 3,
  "SaveFlag": false,
  "ItemDataPath": "",
  "data": {
    "Name": "",
    "Class": "",
    "Level": 0,
    "Gold": 0,
    "Exp": 0.0
  }
}
```


사용중인 Json 파일




```
{
  "ID": 2,
  "ItemName": "Shoulder Armor",
  "SellPrice": 100
},
{
  "ID": 5,
  "ItemName": "MPPortion",
  "SellPrice": 50
},
{
  "ID": 5,
  "ItemName": "MPPortion",
  "SellPrice": 50
},
{
  "ID": 4,
  "ItemName": "HPPortion",
  "SellPrice": 50
},
{
  "ID": 5,
  "ItemName": "MPPortion",
  "SellPrice": 50
},
{
  "ID": 3,
  "ItemName": "Boots",
  "SellPrice": 100
},
```


2번 슬롯에 저장된
플레이어의 아이템 목록
Json 파일

02. 구현설명

3) 저장,불러오기 매니저

```
public void InitSaveFile()
{
    for(int index =1; index <= Constants.ListMax; index++)
    {
        PlayerDataList.Add(new SaveDataSlot(index));
    }

    JsonData SaveJson = JsonMapper.ToJson(PlayerDataList);
    File.WriteAllText(Application.dataPath + "/Resources/Data/SaveFile.json", SaveJson.ToString());

    for(int count = 1; count <= Constants.ListMax; count++)
    {
        SaveJson = JsonMapper.ToJson(SaveItemList);
        File.WriteAllText(Application.dataPath + "/Resources/Data/SaveItemFile" + count.ToString() + ".json", SaveJson.ToString());
    }
}

public void SavePlayerData(int SlotNumber)
{
    LatestSlotNumer = SlotNumber;
    PlayerDataList[SlotNumber].data = PlayerDataForSave;
    PlayerDataList[SlotNumber].SaveFlag = true;
    PlayerDataList[SlotNumber].ItemDataPath = "Data/SaveItemFile" + PlayerDataList[SlotNumber].number.ToString();
    JsonData SaveJson = JsonMapper.ToJson(PlayerDataList);
    if(File.Exists(Application.dataPath + "/Resources/Data/SaveFile.json"))
    {
        File.Delete(Application.dataPath + "/Resources/Data/SaveFile.json");
    }
    File.WriteAllText(Application.dataPath + "/Resources/Data/SaveFile.json", SaveJson.ToString());

    SaveJson = JsonMapper.ToJson(SaveItemList);
    if(File.Exists(Application.dataPath + " /Resources/Data/SaveItemFile" + PlayerDataList[SlotNumber].number.ToString() + ".json"))
    {
        File.Delete(Application.dataPath + " /Resources/Data/SaveItemFile" + PlayerDataList[SlotNumber].number.ToString() + ".json");
    }
    File.WriteAllText(Application.dataPath + " /Resources/Data/SaveItemFile" + PlayerDataList[SlotNumber].number.ToString() + ".json", SaveJson.ToString());
}
```

게임을 실행 할 시,

비어 있는 List<PlayerData>를 이용해

데이터를 쓸 파일을 생성합니다.

데이터를 저장 할 때에는

List에 데이터를 저장한 후,

플레이어 데이터 파일과 아이템 저장 파일을

갱신하도록 했습니다.

02. 구현설명

3) 저장,불러오기 매니저

```
public bool CheckSaveFile()
{
    TextAsset Data = Resources.Load("Data/SaveFile") as TextAsset;
    if(Data == null)
    {
        return false;
    }
    else
    {
        TextAsset ItemData;
        for(int count = 1; count <= Constants.ListMax; count++)
        {
            ItemData = Resources.Load("Data/SaveItemFile" + count.ToString()) as TextAsset;
            if(ItemData == null)
            {
                return false;
            }
        }
        SaveDataText = Data;
        SaveDataToJson = JsonMapper.ToObject(SaveDataText.text);
        for(int index = 0; index < SaveDataToJson.Count; index++)
        {
            bool flag = bool.Parse(SaveDataToJson[index]["SaveFlag"].ToString());
            if(flag == false)
            {
                PlayerDataList.Add(new SaveDataSlot(index + 1));
            }
            else
            {
                SaveDataSlot SlotTemp = new SaveDataSlot(index + 1, true, SaveDataToJson[index]["ItemDataPath"].ToString());
                PlayerData DataTemp = new PlayerData
                    (
                    SaveDataToJson[index]["data"]["Name"].ToString(),
                    SaveDataToJson[index]["data"]["Class"].ToString(),
                    int.Parse(SaveDataToJson[index]["data"]["Level"].ToString()),
                    int.Parse(SaveDataToJson[index]["data"]["Gold"].ToString()),
                    double.Parse(SaveDataToJson[index]["data"]["Exp"].ToString())
                    );
                SlotTemp.data = DataTemp;
                PlayerDataList.Add(SlotTemp);
            }
        }
        return true;
    }
}
```

저장된 파일을 불러올 때

파일이 존재하지 않는다면 false를 반환하고

존재한다면 임시적인 데이터 저장 변수에

파일을 읽어와서 저장 데이터를 가져오도록

구현 했습니다.

02. 구현설명

4)아이템 구매,판매,사용

인벤토리 내부 혹은 NPC의 상점에서 물품을 구매, 판매, 사용 등 을 구현 할 때

NGUI 를 사용해 구현하였습니다.

Sprite 와 Box Collider 컴포넌트를 사용해

클릭 또는 터치 시 OnClick() 함수를 통해 클릭된 아이템의 정보를 받아오고

비활성화돼 있던 구매, 판매, 사용 창을 활성화하도록 구현했습니다.

02. 구현설명

4)아이템 구매,판매,사용

```
private void OnClick()
{
    switch(CurSceneName)
    {
        case "Camp":
            PurchaseAndSell();
            GetNameForPurchaseSell();
            break;

        case "Field":
        case "Arena":
        case "Dungeon":
            PotionUse();
            break;
    }
}
```

OnClick()함수가

실행 될 시 구매,판매는

NPC가 존재하는

Camp씬 에서만 동작하고

포션 사용은 던전,필드 등

게임 플레이 씬에서 동작하도록

구현했습니다.

```
internal void GetNameForPurchaseSell()
{
    switch(tag)
    {
        case "Shop":
            purChaseitem.SelectItem(gameObject.name);
            break;

        case "Inven":
            sellitem.SelectItem(gameObject.name);
            break;
    }
}
```

선택된 아이템의 정보를
전달하는 함수

```
internal void PurchaseAndSell()
{
    if(gameObject.tag == "Shop")
    {
        purChaseitem.OpenPurchaseWindow();
    }
    else if(gameObject.tag == "Inven")
    {
        sellitem.OpenSellWindow();
        sellitem.SelectObject(gameObject);
    }
}
```

구매, 판매 시 창(Window)활성화

```
internal void PotionUse()
{
    if (gameObject.tag == "Inven")
    {
        if(potionUse == null)
        {
            potionUse = GameObject.Find("UI Root").transform.Find("PotionUseWindowPanel")
                .Find("PotionUseWindow").GetComponent<PotionUse>();
        }
        if(potionUse.IsSelectObject(gameObject))
        {
            potionUse.OpenPotionUseWindow();
            potionUse.SelectObject(gameObject);
            potionUse.SelectItem(gameObject.name);
        }
    }
}
```

포션 사용시 창(Window)활성화

02. 구현설명

5)몬스터, 아이템 HUD

몬스터의 체력바나 아이템이 드랍 됐을 때 어떤 아이템 이름이 적힌 HUD를 구현하기 위해
몬스터나 아이템 상자의 3D 월드 좌표를 Viewport 좌표로 변환 한 후,
체력바나 아이템 이름의 좌표를 다시 3D 좌표로 변환 한 위치에 자리하도록 했습니다.

02. 구현설명

5)몬스터, 아이템 HUD

```
private void SetPositionHUD()
{
    Vector3 Vposition;
    switch (tag)
    {
        case "HPBar":
            {
                Vposition = Camera.main.WorldToViewportPoint(TargetTransform.position
        + new Vector3(0, 2, 0));
                transform.position = uiCamera.ViewportToWorldPoint(Vposition);
            }
            break;

        case "MPBar":
            {
                Vposition = Camera.main.WorldToViewportPoint(TargetTransform.position
        + new Vector3(0, 2, 0));
                transform.position = uiCamera.ViewportToWorldPoint(Vposition);
            }
            break;

        case "ItemNameLabel":
            {
                Vposition = Camera.main.WorldToViewportPoint(TargetTransform.position
       + new Vector3(0, 0.3f, 0));
                transform.position = uiCamera.ViewportToWorldPoint(Vposition);
            }
            break;

        case "BossBar":
            {
                Vposition = Camera.main.WorldToViewportPoint(TargetTransform.position
        + new Vector3(0, 3, 0));
                transform.position = uiCamera.ViewportToWorldPoint(Vposition);
            }
            break;
    }
}
```

스크립트가 적용된 오브젝트가

체력 게이지 혹은 아이템 이름 인지에 따라

높이를 다르게 더해준 후,

Viewport 좌표로 변환하고

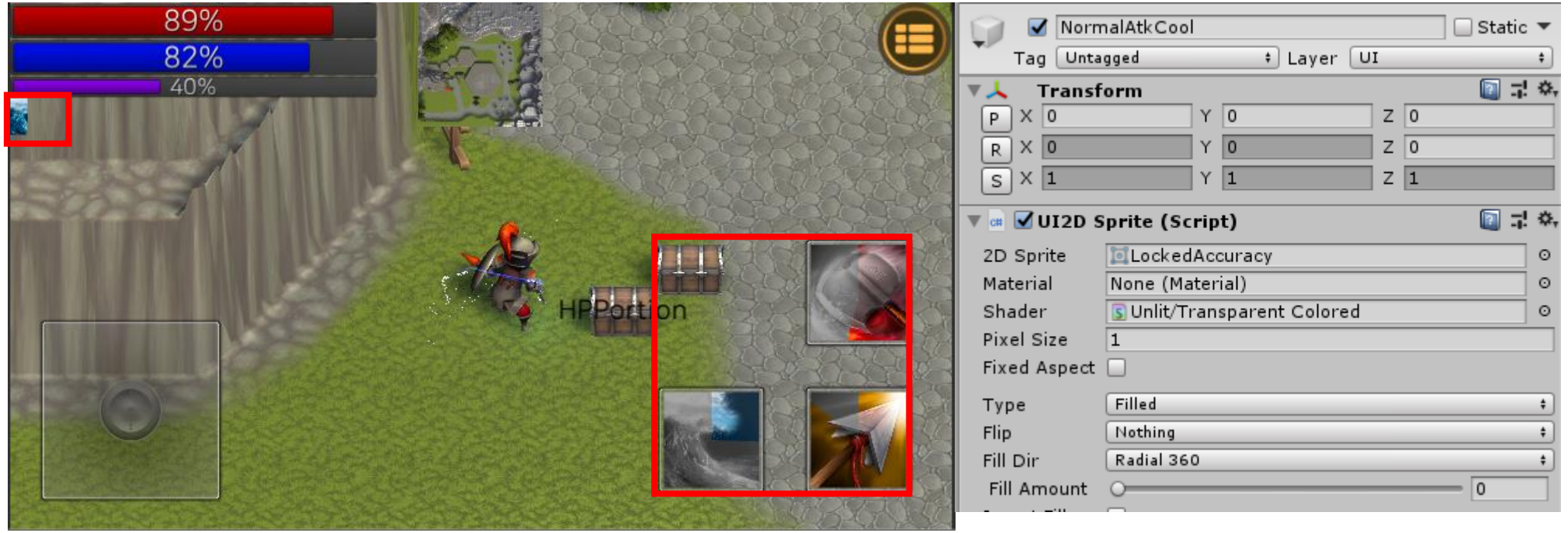
오브젝트의 좌표를 높이가 더해지고

Viewport로 변환된 좌표를 다시

3D월드 좌표로 변환해 오브젝트에 적용시켰습니다.

02. 구현설명

6)스킬 쿨 타임, 지속시간 HUD



스프라이트의 fill 항목을 Radial 방식을 사용해 현재 적용중인 버프의 시간이나,
쿨 타임 적용중인 스킬을 좀 더 시각적으로 볼 수 있게 구현했습니다

02. 구현설명

6)스킬 쿨 타임, 지속시간 HUD

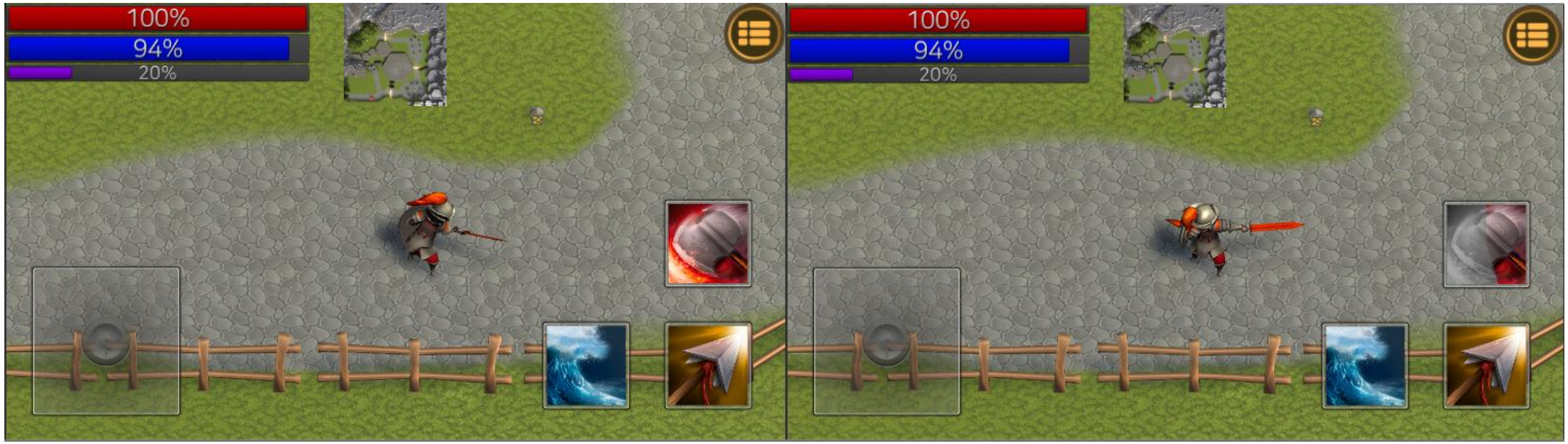
```
IEnumerator BuffCoolDown()
{
    while(true)
    {
        if(playerinfo.BuffCoolON == true)
        {
            if(BuffCoolObj.activeSelf == false)
            {
                BuffCoolObj.SetActive(true);
            }
            playerinfo.BuffCoolingDown(Time.deltaTime);
            BuffCool.fillAmount = 1 - playerinfo.BuffCoolRate();
            if(BuffCool.fillAmount == 1)
            {
                BuffCoolObj.SetActive(false);
            }
        }
        yield return null;
    }
}

IEnumerator BuffDurationDown()
{
    while (true)
    {
        if (playerinfo.BuffFlag == true)
        {
            if (BuffOnObj.activeSelf == false)
            {
                BuffOnObj.SetActive(true);
            }
            playerinfo.BuffDurationDown(Time.deltaTime);
            BuffOn.fillAmount = 1 - playerinfo.BuffDurationRate();
            if (BuffOn.fillAmount == 1)
            {
                BuffOnObj.SetActive(false);
            }
        }
        yield return null;
    }
}
```

스킬을 사용시 쿨 타임인 것을 표시하기위해
색깔이 없는 흑백 이미지를 활성화 한 후,
Time.deltaTime 을 사용해 쿨 타임 비율 만큼
흑백이미지가 시계방향으로 사라져 없어지도록
구현 했습니다.

지속 시간이 있는 버프 스킬의 경우
체력 게이지 아래에 이미지를 활성화 시켜
Time.deltaTime 을 사용해
줄어든 지속 시간 비율 만큼 이미지가 시계방향으로
사라져 없어지도록 구현했습니다.

02. 구현설명

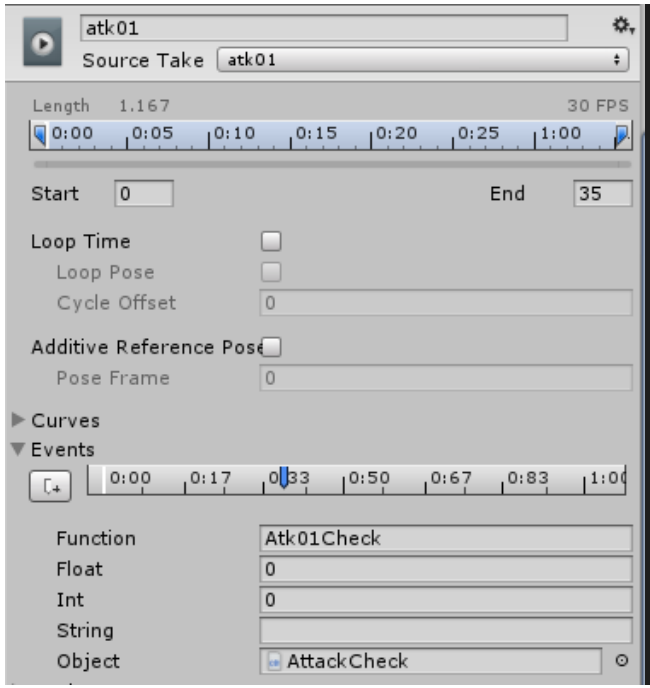
7)공격 체크



플레이어 또는 몬스터의 공격을 체크할 때 애니메이션의 특정 시간에서
Event를 생성, 함수를 추가 하고
함수 내부에서 Physics.OverlapSphere 를 사용해 공격 체크 여부를 확인했습니다.

02. 구현설명

7)공격 체크




```
public void Atk01Check()
{
    var col = Physics.OverlapSphere(tr.position + tr.transform.forward, 1.5f);
    if(tr.tag == "Player")
    {
        for(int index =0; index < col.Length; index++)
        {
            if(col[index].tag == "Enemy")
            {
                col[index].transform.GetComponent<EnemyInfo>().DecreaseHP(playerinfo.GetAtk01Damage());
                col[index].transform.GetComponent<FSMTest>().GetHurted();
            }
            else if(col[index].tag == "Boss")
            {
                col[index].transform.GetComponent<BossInfo>().DecreaseHP(playerinfo.GetAtk01Damage());
            }
        }
    }
```

공격 체크를 위해 작성된 스크립트가 적용된 오브젝트의 태그에 따라
Physics.OverlapSphere가 발생했을 때 반경 내의 오브젝트를 검출해 태그를 확인해
각각 데미지를 입도록 적용했습니다.

02. 구현설명

8)FSM

몬스터의 AI는 보스, 일반 몬스터 두 종류로 나누어 구현했습니다.
플레이어를 추적할 때는 NavMesh를 사용해 추적하도록 했습니다.
기본적인 구조는 같지만 보스는 마나 변수를 추가해 마나가 가득차면 스킬을 쓰도록 구현했습니다.

02. 구현설명

8)FSM

몬스터

```
switch(Enemystate)
{
    case ENEMY_STATE.IDLE:
        StopAttacking();
        nav.isStopped = true;
        anim.SetInteger("EnemyState", (int)ENEMY_STATE.IDLE);
        break;

    case ENEMY_STATE.TRACE:
        StopAttacking();
        nav.SetDestination(Player.transform.position);
        nav.isStopped = false;
        anim.SetInteger("EnemyState", (int)ENEMY_STATE.TRACE);
        break;

    case ENEMY_STATE.ATTACK:
        anim.SetInteger("EnemyState", (int)ENEMY_STATE.ATTACK);
        if (IsAtkCoroutine == false && MotionFlag == true)
        {
            StartCoroutine(Attacking());
            IsAtkCoroutine = true;
        }
        break;
}
yield return new WaitForSeconds(0.5f);

IEnumerator Attacking()
{
    while (Enemystate == ENEMY_STATE.ATTACK)
    {
        if(playerInfo.GetPSTATE() == PlayerInfo.PSTATE.ALIVE)
        {
            nav.isStopped = true;
            MotionFlag = false;
            anim.SetTrigger("atk01Trigger");
            yield return new WaitForSeconds(anim.GetCurrentAnimatorStateInfo(0).length);
            yield return new WaitForSeconds(2.0f);
            MotionFlag = true;
        }
        else
        {
            yield return new WaitForSeconds(1.0f);
        }
    }
}
```

보스

```
switch (Bossstate)
{
    case BOSS_STATE.IDLE:
        StopAttacking();
        nav.isStopped = true;
        anim.SetInteger("BossState", (int)BOSS_STATE.IDLE);
        break;

    case BOSS_STATE.TRACE:
        StopAttacking();
        nav.SetDestination(Player.transform.position);
        nav.isStopped = false;
        anim.SetInteger("BossState", (int)BOSS_STATE.TRACE);
        break;

    case BOSS_STATE.ATTACK:
        anim.SetInteger("BossState", (int)BOSS_STATE.ATTACK);
        if (IsAtkCoroutine == false && MotionFlag == true)
        {
            StartCoroutine(Attacking());
            IsAtkCoroutine = true;
        }
        break;

    case BOSS_STATE.SKILL:
        StopAttacking();
        nav.isStopped = true;
        StartCoroutine(UseSkill());
        break;
}
yield return new WaitForSeconds(2.0f);

IEnumerator UseSkill()
{
    MotionFlag = false;
    anim.SetTrigger("atk03Trigger");
    yield return new WaitForSeconds(1.0f);
    GameObject tmp = Instantiate(Resources.Load("Prefab/Effect/BossSkill"),
        tr.position, Quaternion.Euler(0, 0, 0)) as GameObject;
    bossinfo.DecreaseMP(100.0f);
    MotionFlag = true;
    yield return null;
}
```

공격 모션이나 스킬 모션이 동작하기 전에는 NavMesh의 작동을 정지시켜
모션이 나오는 동안에는 움직이지 않도록 구현했습니다.

01. 게임소개

2.PangPang

제작기간 : 2019.05.07 ~ 2019.05.30

API를 사용해 Pang을 모작한 게임입니다.
풍선이 바닥과 벽에 튕기는 모습을 구현하는데 중점을 두고 제작했습니다.

02. 구현설명

PangPang

```
void GameScene::BallonMove(int Level)
{
    for (list<Ballon>::iterator iter = m_BList.begin(); iter != m_BList.end(); ++iter)//풍선을 이동시키는 부분
    {
        if (iter->m_nBLevel == Level)
        {
            switch (iter->m_eDirState)
            {
            case LEFT:
                if (iter->m_ptfLoc.x - m_pArrRedBallon[iter->m_nBLevel]->GetWidth() / 2 > 14)//왼쪽벽에 부딪히지 않으면
                {
                    iter->m_ptfLoc.x -= 1.5f + (0.5f * Level);
                    iter->m_ptfLoc.y = iter->m_ptfOri.y - m_nArrHeightLmt[iter->m_nBLevel] * sinf(PI * iter->m_fDeg / 180);
                }
                else
                {
                    iter->m_eDirState = RIGHT;
                }
                break;

            case RIGHT:
                if (iter->m_ptfLoc.x + m_pArrRedBallon[iter->m_nBLevel]->GetWidth() / 2 < WIDTH - 14)//오른쪽 벽에 부딪히지 않으면
                {
                    iter->m_ptfLoc.x += 1.5f + (0.5f * Level);
                    iter->m_ptfLoc.y = iter->m_ptfOri.y - m_nArrHeightLmt[iter->m_nBLevel] * sinf(PI * iter->m_fDeg / 180);
                }
                else
                {
                    iter->m_eDirState = LEFT;
                }
                break;
            }
            //풍선의 RECT갱신
            iter->m_rcRect.Set(iter->m_ptfLoc.x - m_pArrRedBallon[iter->m_nBLevel]->GetWidth() / 3, iter->m_ptfLoc.y - m_pArrRedBallon[iter->m_nBLevel]->GetHeight() / 2,
                iter->m_ptfLoc.x + m_pArrRedBallon[iter->m_nBLevel]->GetWidth() / 3, iter->m_ptfLoc.y + m_pArrRedBallon[iter->m_nBLevel]->GetHeight() / 2);

            iter->m_fDeg += (Level * 0.5f) + 1;
            if (iter->m_fDeg >= 180)
            {
                iter->m_fDeg = 0;
            }
        }
    }
}
```

풍선의 움직임을 구현한 부분입니다.
풍선의 움직이는 방향과 분열된 단계에 따라 움직이는 거리와 높이가 다르도록 구현했습니다.

01. 게임소개

PangPang

3.모두의 게임

제작기간 : 2019.05.31 ~ 2019.06.28

API를 사용해 모두의 게임에서 '아슬아슬 비행기'와 '알록달록 색종이' 게임을 모작한 게임입니다.
비행기 부분에서 점수를 주는 별과 피해야 되는 총알이 방향,각도, 속력이 랜덤 하게
날아오는 부분을 구현하는데 중점을 두었습니다.

02. 구현설명

PangPang

```
void ColoredPaper::Update(float fETime)
{
    if (m_eChange == ON)
    {
        m_nSelectedPaper = rand() % 4;
        m_eChange = OFF;
        m_ptfPaperLoc = m_ptfCenterLoc;
    }

    switch (m_eDirstate)
    {
    case LEFT:
        if (m_ptfPaperLoc.x >= 17)
        {
            m_ptfPaperLoc.x -= fETime * 1000;
        }
        else
        {
            m_ptfPaperLoc.x = 17;
            m_eChange = ON;
            m_eDirstate = NOT;
            m_nPlayerScore += (m_nPaperScore * (m_nFeverLevel + 1));
            if (m_fFeverGaze + 10 <= 100)
            {
                m_fFeverGaze += 10;
            }
        }
        break;
```

색종이의 움직임을 구현한 코드의 일부분 입니다.

중앙 위치를 기준으로 누른 방향키의 방향에

있는 색종이와 중앙의 색종이가

같은 색의 색종이인 경우 정해진 위치까지

이동하도록해 위치에 도달할 경우 Fever레벨에

비례한 점수가 증가하고,

Fever게이지가 증가 하도록 구현했습니다

02. 구현설명

PangPang

```
if (iter->m_ptfLoc.y + fETime * 200 <= 608)
{
    if (iter->m_fOriX <= 0)
    {
        iter->m_ptfLoc.x += fETime * 70 * iter->m_nIncX;
        iter->m_ptfLoc.y += fETime * 70 * iter->m_nIncY;
    }
    else if (iter->m_fOriX >= WIDTH)
    {
        iter->m_ptfLoc.y += fETime * 70 * iter->m_nIncY;
        iter->m_ptfLoc.x -= fETime * 70 * iter->m_nIncX;
    }
    else
    {
        if (iter->m_nIncX % 2 == 0)
        {
            iter->m_ptfLoc.x -= fETime * 70 * iter->m_nIncX;
        }
        iter->m_ptfLoc.y += fETime * 100 * iter->m_nIncY;
    }
    iter->m_rcRect.Set(iter->m_ptfLoc.x - 6, iter->m_ptfLoc.y - 6, iter->m_ptfLoc.x + 6, iter->m_ptfLoc.y + 6);
}
else
{
    iter = List.erase(iter);
    if (List.size() == 0)
        break;
}
```

비행기 게임의 구현 코드 중 점수 별과 총알을 움직이는 코드 부분입니다.

별과 총알은 공통적으로 임의의 x 값과 y 값을 저장하는 구조체를 가지고있습니다.

이 값을 이용해 시간에 따라 움직이게 하여 방향(기울기)와 속력이 다르게 움직이도록 구현했습니다.

Thank you

윤찬호