[포트폴리오]
트릭스터 택틱스 기술문서

관련 동영상

https://youtu.be/zpo-eEXplFg

경일 게임 아카데미

게임 프로그래밍 16기

이 도 규

0. 목차

1. 게임 내용 및 설명	2. 구현 내역	3. 기술 코드	부록
1-1 개발 환경	INDEX	클래스 구조	
1-2 일정	1. CHARMANAGER	1. CHARMANAGER	
1-3 내용	2. MAP TOOL	2. MAP TOOL	
	3. 캐릭터 이동	3. 캐릭터 이동	
	4. 인터페이스	4. 인터페이스	

1. 게임 개요 및 설명 – 개발 환경

구분	환경
CPU	3세대 I5 3470 – 3.20GHZ
메모리	8GB
시스템	X64
하드	128GB
운영체제	WINDOW 10 PRO
해상도	1024 * 800
라이브러리	WINAPI / C++ / FMOD

1. 게임 개요 및 설명 – 일정

1주차 일정	내용
월	카메라 구현
화	캐릭터 인터페이스
수	캐릭터 매니저
목	Alpha Render 구현
금	Isomatric 맵 구현
토	통상 모드 테스트
일	디버그

2주차 일정	내용
월	디버그 문제 해결
화	씬 나누기 포탈 구현
수	UI 틀 작성
목	인터페이스 생성
금	맵 타일 오브젝트
토	맵 툴 작성
일	맵 툴 디버그

3주차 일정	내용
월	저장 / 로드 테스트
화	전투 / 통상 맵 디버그
수	Z order 구현
목	Z order 상호작용 디버그
금	A* 알고리즘 적용
토	A* 알고리즘 디버그
일	A* 알고리즘 캐릭터 적용

4주차 일정	내용
월	전투 장면 구현
화	전투 장면 UI 구성
수	맵 전환 디버그
목	기능 디버그
금	전체 디버그
토	
일	

1. 게임 개요 및 설명 – 내용

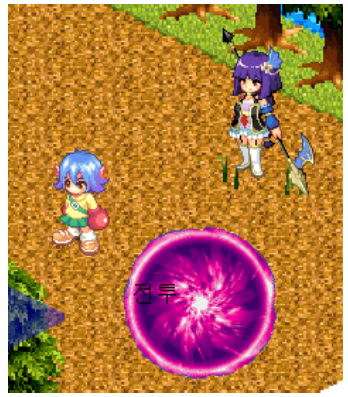
게임 기획 내용

((Motivation) 파랜드 택틱스 형식의 게임) – 트릭스터 리소스 사용

1. SRPG 형식의 택틱스 게임
2. 맵 상의 형태는 통상 모드 / 전투 모드로 나뉜다.
3. 통상 모드에서는 캐릭터의 제어권을 가지고 움직일 수 있다.
4. 상점, 던전 등의 특정 장소로 이동이 가능하도록 하며 여러 기능을 사용할 수 있다.
5. 전투 모드에서는 캐릭터의 제어권이 사라진다.
6. 플레이어는 현재 맵 상에 있는 캐릭터를 움직여 상대 캐릭터를 물리친다.

1. 게임 개요 및 설명 – 내용

PlayScene



Map Tool Scene

Battle Scene

2. 구현 내역 – INDEX

기본 구현 Base 설정

1. 타이머 설정
2. 더블 버퍼링
3. KEY 매니저
4. RANDOM
5. IMAGE MANAGER
6. TXT MANAGER
7. Animation
8. Ahlpa Channel
9. Sound
10. MOUSE MANAGER
11. SCENE MANAGER
12. CAMERA

게임 주 기능 설정

1 CHAR MANAGER

- NPC
- CHARACTER

3. 캐릭터 이동

- A* 알고리즘

2. MAP TOOL

- Z-ORDER
- ISOMatrix

4. 인터페이스

- 버튼 UI

더미데이터

- ITEM MANAGER
- BATTLE 턴
- BATTLE 캐릭터 선택
- 스테이터스 외 인터페이스

2. 구현 내역 – CHAR MANAGER

제작 내용

다양한 캐릭터의 능력, 위치, 움직임 등을 관리하기 위한 통합 시스템 제작

1. CHARMANAGER가 관리하는 것은 내용
    1. 캐릭터의 위치
    2. 현재 캐릭터가 바라보고 있는 방향 및 모션
    3. 캐릭터가 행동해야 할 다양한 애니메이션 및 스킬
2. 넘겨받은 정보를 바탕으로 각각의 캐릭터 클래스에게 그 클래스가 해야 할 행동 정보를 전달
3. 각각의 클래스는 자신의 TXT 정보를 읽어 들여 상태를 저장 및 로드 할 수 있음
4. 캐릭터, NPC, 이후 추가될 몬스터 또한 사용 가능

2. 구현 내역 – 맵툴

제작 내용

전투 맵에 사용될 맵을 제작하기 위한 툴 구현

1. 맵에 찍을 수 있는 성분은 크게 TILE과 OBJECT로 구성
   1. TILE은 보통과 강으로 구성, 속성값을 가짐
   2. 오브젝트는 크기에 맞게 화면 출력, 속성값을 가짐

2. 맵 화면은 오브젝트가 타일의 크기에 맞게 출력하도록 정렬 (Z – ORDER 구현)

3. 세이브, 로드, 전체 삭제, 개인 삭제 구현
4. 샘플 맵 타일의 모습은 BMP 파일을 세분화

2. 구현 내역 – 캐릭터 이동

제작 내용

A* 알고리즘 기반, 타일을 찾아 캐릭터를 이동하는 방식

1. 캐릭터는 자신이 바라보는 각도에 따라 회전이 가능
2. 캐릭터는 자신의 타일을 알고 있으며, 주변 타일에 대한 충돌 체크
3. 움직일 수 있는 타일 제한 및 A* 알고리즘으로 움직일 영역을 표시
4. 자신의 타일이 움직일 영역에 도달하는지를 확인하여, 캐릭터를 움직인다.

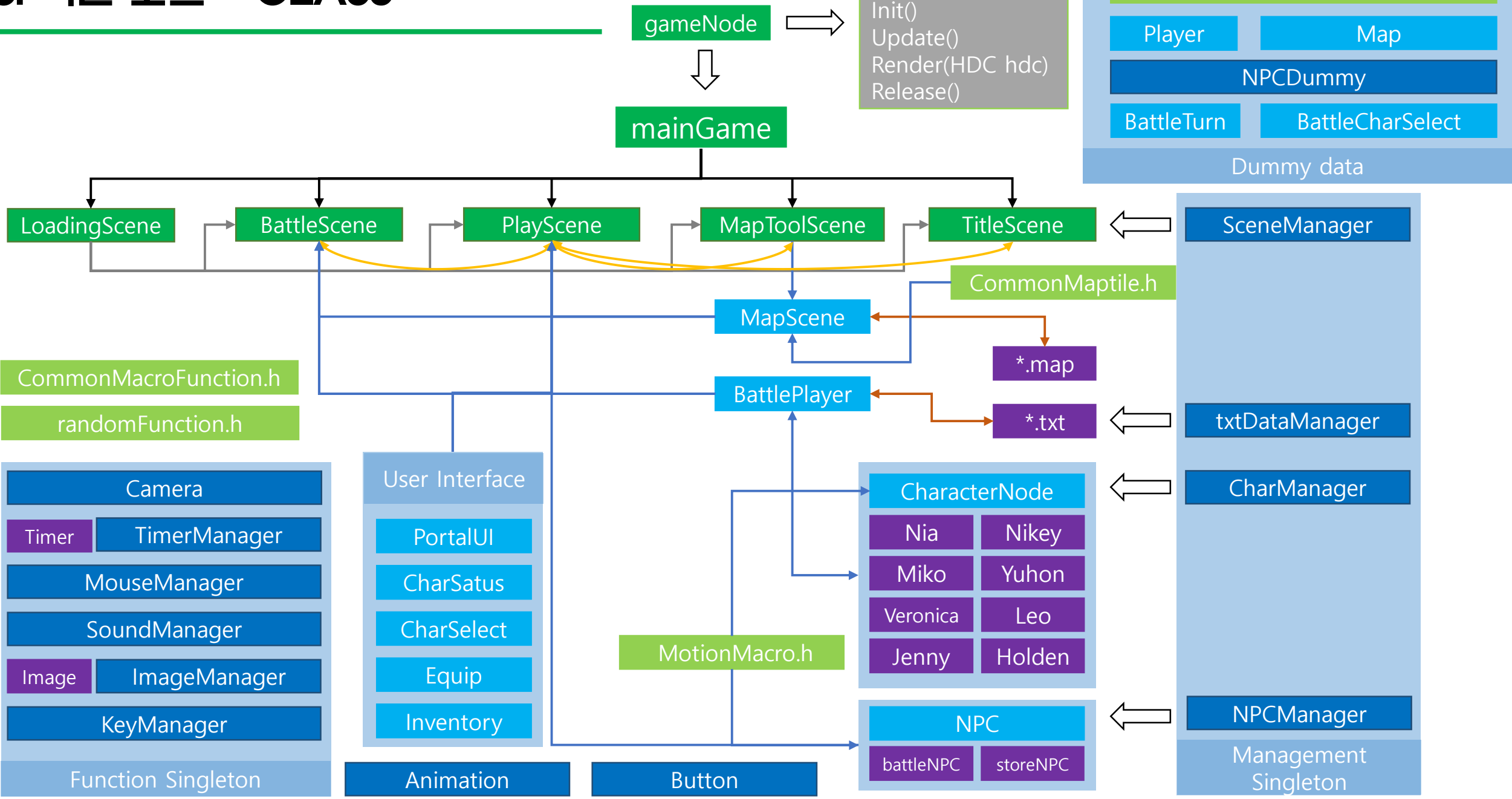
2. 구현 내역 – 인터페이스

제작 내용

전반적인 인터페이스 구성

1. 각각의 인터페이스는 이를 담당하는 클래스를 구성하도록 적용
2. 캐릭터 상태 창의 정보는 TXT 파일 기반 데이터를 받음
3. 전투 모션 및 상태를 주기적으로 확인

3. 기술 코드 – CLASS

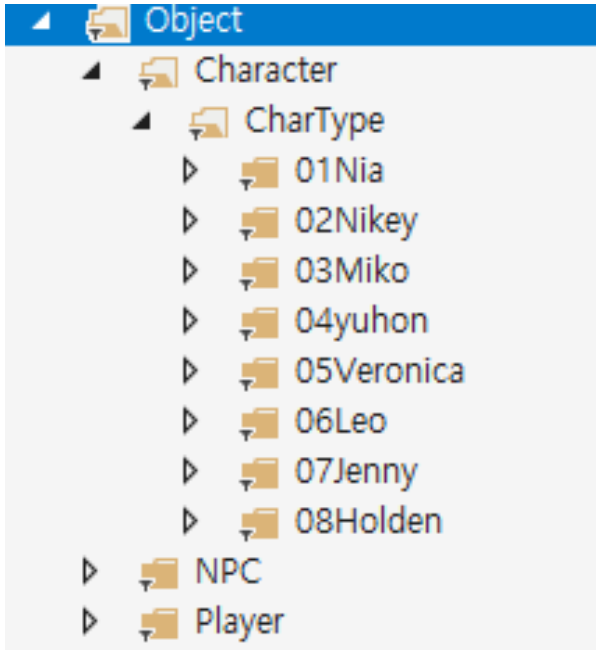

gameNode
Init()
Update()
Render(HDC hdc)
Release()
mainGame
UserInterface.h
Player
Map
NPCDummy
BattleTurn
BattleCharSelect
Dummy data
LoadingScene
BattleScene
PlayScene
MapToolScene
TitleScene
SceneManager
CommonMaptile.h
MapScene
*.map
CommonMacroFunction.h
randomFunction.h
BattlePlayer
*.txt
txtDataManager
User Interface
PortalUI
CharSatus
CharSelect
Equip
Inventory
CharacterNode
Nia
Nikey
Miko
Yuhon
Veronica
Leo
Jenny
Holden
CharManager
MotionMacro.h
NPC
battleNPC
storeNPC
NPCManager
Management Singleton
Camera
Timer
TimerManager
MouseManager
SoundManager
Image
ImageManager
KeyManager
Function Singleton
Animation
Button

3. 기술 코드 – CHAR MANAGER

CHARMANAGER 함수 구성

```
// 캐릭터 추가
CharacterNode* AddCharacter(string _key, CharacterNode* _char);
CharacterNode* FindCharacter(string _key);
// 캐릭터 체인지
HRESULT SelectChar(string _charName);

// 캐릭터 위치 좌표
void CharacterMove(float _posX, float _posY);₩
// 캐릭터 방향
void CharDirectState(int _motion);
// 캐릭터 애니메이션
void CharActionState(int _action);
```



캐릭터 관리는 자료구조 <map>을 사용하여 관리

MAP에서 관리하는 CLASS

3. 기술 코드 – MAP TOOL

```
struct tagSelectTile
{
    int showType;
    int showTileType;
    int showObjectType;
    int terrainFrameX;
    int terrainFrameY;
    int objectFrameX;
    int objectFrameY;
    int objectTileSizeX;
    int objectTileSizeY;
    int objectPosX;
    int objectPosY;
    int tileSizeX;
    int tileSizeY;
    bool objectTrue;
```

MAPTILE 하나가 가지고 있는 속성

```
list<ZORDER*> zoderData;
list<ZORDER*>::iterator itZorderData;
```

```
ZORDER* zorder = new ZORDER;
zorder->pos = MakePoint(x,y);
zorder->priority = x + y;
zorder->type = ZORDER_TILE;
zorderData.push_back(zorder);
```

```
for (itZorderData = zorderData.begin();itZorderData != zorderData.end();itZorderData++)
{
    if ((*itZorderData)->pos.x == playerPosition.x && (*itZorderData)->pos.y == playerPosition.y)
    {
        if (player)player->Render(gameMapDC);
    }
    if ((*itZorderData)->type == ZORDER_TILE)
    {
        if ((*itZorderData)->isPlayer == false)
        {
            mapScene->RenderObject(gameMapDC, (*itZorderData)->pos.x, (*itZorderData)->pos.y);
        }
    }
}
```

List로 우선순위(타일 x + y값) 적용
정렬 이후 출력

3. 기술 코드 – 캐릭터 이동

```
list<Node*>openList;
list<Node*>closeList;
list<Node*> pathList;
list<Node*>::iterator itPathList;

Node*       startNode;
Node*       destNode;
Node*       currentNode;

class Node
{
public:
    int idx;
    int idy;
    int F, G, H;       // COST
    Node* parentNode;  // 경로 추적
```

```
void PlayScene::AddOpenList(int _idx, int _idy)
{
    if (!(mapScene->MoveCheck(_idx, _idy)))
    {
        return;
    }

    Node* tempNode = new Node(_idx, _idy);
    tempNode->parentNode = currentNode;
    tempNode->H = fabs(_idx - destNode->idx) + fabs(_idy - destNode->idy);
    tempNode->G = currentNode->G + 1;
    tempNode->F = tempNode->G + tempNode->H;

    openList.push_back(tempNode);
}
```

```
    //// 경로
    Node* tempNode = currentNode;

    pathList.push_back(tempNode);
    while (tempNode->parentNode != NULL)
    {
        pathList.push_back(tempNode);
        tempNode = tempNode->parentNode;
    }
    return;
}

AddOpenList(currentNode->idx - 1, currentNode->idy);
AddOpenList(currentNode->idx, currentNode->idy - 1);
AddOpenList(currentNode->idx + 1, currentNode->idy);
AddOpenList(currentNode->idx, currentNode->idy + 1);

DeleteOpenList(currentNode->idx, currentNode->idy);
```

A* 알고리즘 주요코드

3. 기술 코드 – 캐릭터 이동

```
if (player->GetMoveCheck())
{
    if (!(pathList.empty()))
    {
        targetPosition.x = (*itPathList)->idx;
        targetPosition.y = (*itPathList)->idy;

        if (itPathList == pathList.begin())
        {
            targetPosition.x = (*itPathList)->idx;
            targetPosition.y = (*itPathList)->idy;
        }
    }
}

if (!pathList.empty())
{
    player->SetTargetPos(tiles[targetPosition.y][targetPosition.x].rc.left + TILESIZE_X_HALF + 5,
        tiles[targetPosition.y][targetPosition.x].rc.top + TILESIZE_Y_HALF - 5);
}
```

A* 알고리즘이 찾은 경로에 따라 캐릭터가 움직이는 소스코드

3. 기술 코드 – 인터페이스

```
// KEY ONCEKEYDOWN
if (KEYMANAGER->IsOnceKeyDown('P'))
{
    if (!openUI[UI_PARTY]){openUI[UI_PARTY] = true;}
    else if (openUI[UI_PARTY]) {openUI[UI_PARTY] = false; }
}
if (KEYMANAGER->IsOnceKeyDown('I'))
{
    if (!openUI[UI_ITEM]) { openUI[UI_ITEM] = true; }
    else if (openUI[UI_ITEM]) { openUI[UI_ITEM] = false; }
}
if (KEYMANAGER->IsOnceKeyDown('E'))
{
    if (!openUI[UI_EQUIP]) { openUI[UI_EQUIP] = true; }
    else if (openUI[UI_EQUIP]) { openUI[UI_EQUIP] = false; }
}
if (KEYMANAGER->IsOnceKeyDown('V'))
{
    if (!openUI[UI_STATUS]) { openUI[UI_STATUS] = true; }
    else if (openUI[UI_STATUS]) { openUI[UI_STATUS] = false; }
}
if (KEYMANAGER->IsOnceKeyDown(VK_ESCAPE))
{
    if (!openUI[UI_EXIT]) { openUI[UI_EXIT] = true; }
    else if (openUI[UI_EXIT]) { openUI[UI_EXIT] = false; }
}
```

```
if (openUI[UI_ITEM])inven->Update();
if (openUI[UI_STATUS])status->Update();
if (openUI[UI_EQUIP])equip->Update();

if (openUI[UI_ITEM])inven->Render(hdc);
if (openUI[UI_STATUS])status->Render(hdc);
if (openUI[UI_EQUIP])equip->Render(hdc);
```

```
switch (commandState)
{
case COMMAND_MOVE:
    AstarUpdate();
    break;
case COMMAND_READY:
    CommandCollisionCheck();
    break;
case COMMAND_BATTLE:
    AttackUpdate();
    break;
case COMMAND_SKILL:
    SkillUpdate();
    break;
case COMMAND_TURNOVER:
    TurnOver();
    break;
case COMMAND_CONFIRM:
    ConfirmUpdate();
    break;
}
```

UI 관련 클래스 관리 시스템

4. 부록

포트폴리오 개발에 대한 의의

1. 협동 콘텐츠 개발을 위한 클래스 작업의 맛보기
2. 하나의 오브젝트를 위해 구현되는 세세한 기능의 탐구 및 학습
3. 클래스의 구조화와 각종 자료구조 사용방법에 대한 의식 함양
4. 향후 개발 콘텐츠에 대한 사전 작업의 필수 작업
5. 디버그와 메모리 관리의 중요성 함양

읽어 주셔서 감사합니다.