

# 클 라이 언 트

N A M E : 권 성 호

T E L : 010-8874-6452

P A R T : PROGRAMING

**Efforts make all doors open**



# 목 차



I . 포트폴리오 개요 .....	1
1. 소화기를 쏘라 .....	1
2. BlackestNight .....	2
3. Babel .....	3
4. MiniMini .....	4
5. SpaceShooter .....	5
6. JAY's FACTORY .....	6
7. Guardians of the Galaxy .....	7
8. RunTurtle .....	8
9. Ogre기술데모 .....	9
10. DirectX 프레임워크 .....	10
11. Erica(졸업작품) .....	11
II . 기술요약 및 소스분석 .....	12
1. Erica(졸업작품) .....	12
2. BlackestNight .....	29
3. Babel .....	31
4. MiniMini .....	32
5. SpaceShooter, JAY's FACTORY,Guardians of the Galaxy	33
6. RunTurtle .....	33
7. Ogre기술데모,DirectX 프레임워크.....	33

# I. 포트폴리오 개요

## 1. 소화기를 쏴라!

<b>게 임 명</b>	소화기를 쏴라!
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	1. 불이 난 건물을 소화기를 쏴아 불을 끄는 게임 2. 간편한 인터페이스와 아기자기한 캐릭터로 재미를 부가 3. 이지모드와 하드모드로 난이도 조절
<b>개발 기간</b>	2014.3.(2주)
<b>개발 환경</b>	Unity3D 4.5.1f3 + C#
<b>개발 인원</b>	기획 : 황은영 프로그래밍 : 권성호 그래픽 : 이승균
<b>소감</b>	처음으로 기획, 프로그래밍, 그래픽이 모여 만든 게임이다. 처음으로 만든 게임은 아니지만 3명이 모여 2주라는 짧은 기간 내에 밤을 새며 만든 게임으로 나에게 의미가 크다. 이 게임을 통해 게임수학의 중요성을 알았으며 학과 수업에 더욱 집중 할 수 있었던 발판이 되었다. 또한 게임은 혼자가 아닌 사람과 사람이 모여 만드는 팀 프로젝트라는 것을 각인 시켰다. 항상 자만해오던 나에게 나의 부족함을 일깨워 준 게임이다.

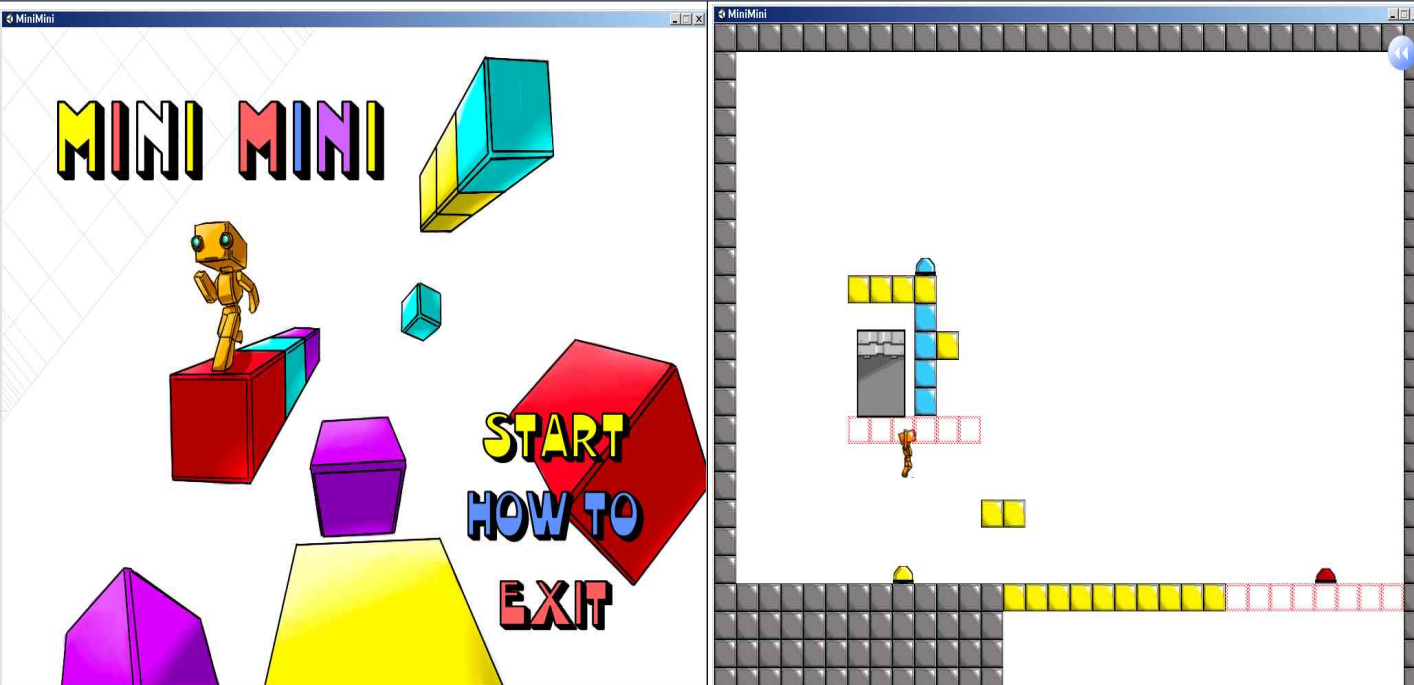
## 2. BlackestNight

게 임 명	BlackestNight
플 랫 폼	PC
게임화면	
	
게 임 소 개	<ol style="list-style-type: none"> <li>1. 세계제일의 도둑이 어떤 문서를 훔치기 위해 건물에 잠입하는 잠입액션 게임</li> <li>2. 빛에 닿게 되면 도둑은 죽는다.</li> <li>3. 구조물을 이용하여 빛을 피해라</li> </ol>
개발 기간	2014.4.(2주)
개발 환경	Unity3D 4.5.1f3 + NGUI + C#
개발 인원	기획 : 황은영 프로그래밍 : 권성호 그래픽 : 이승균
소 감	디버그에 없던 오류가 빌드 후에 나와 적지 않게 당황케 했던 게임이다. 게임 제출 날짜는 지났고 새벽에 일어난 일이어서 거의 이 게임을 포기하고 싶은 마음으로 손을 놓고 있었다. 하지만 같이 밤을 지새운 황은영양의 응원과 승균이 형이 내가 잡지 못한 오류를 잡아 주어서 이 게임을 완성하였다. 결국 게임프로젝트 수업에서 1등을 하게 되었다. 팀원들의 소중함을 알게 해준 게임이다.


### 3. Babel

게 임 명	Babel
플 랫 폼	PC
게임화면	
	
게 임 소 개	<ol style="list-style-type: none"> <li>1. BABEL이라는 탑을 정복하자</li> <li>2. 파츠를 맞춰가며 더욱 강한 아이템을 장착하자</li> <li>3. 끝이 없이 나오는 적을 해치우자.</li> </ol>
개발 기간	2014.5.(2주)
개발 환경	Unity3D 4.5.1f3 + NGUI + C#
개발 인원	기획 : 황은영 프로그래밍 : 권성호 그래픽 : 이승균
소 감	<p>인벤토리와 아이템 장착이라는 시스템을 만들어 본 첫 게임이다. 대략 2주라는 기간 동안 절반이상을 인벤토리시스템 제작과 머리, 팔, 몸통, 다리로 이루어진 주인공 오브젝트의 움직임 제어에 시간을 쏟게 되어 처음 기획과는 다른 게임이 되고 말았다. 모든 RPG게임에 존재하는 인벤토리 제작이 이렇게 어려울 줄 몰랐다. 단순히 이미지교체뿐만 아니라 데이터관리와 캐릭터와의 상호작용, 자료구조의 예외처리등 생각할 사항이 너무 많았다. 하지만 이 게임을 통해서 한층 더 객체지향적인 사고를 가지게 되었다.</p>

## 4. MiniMini

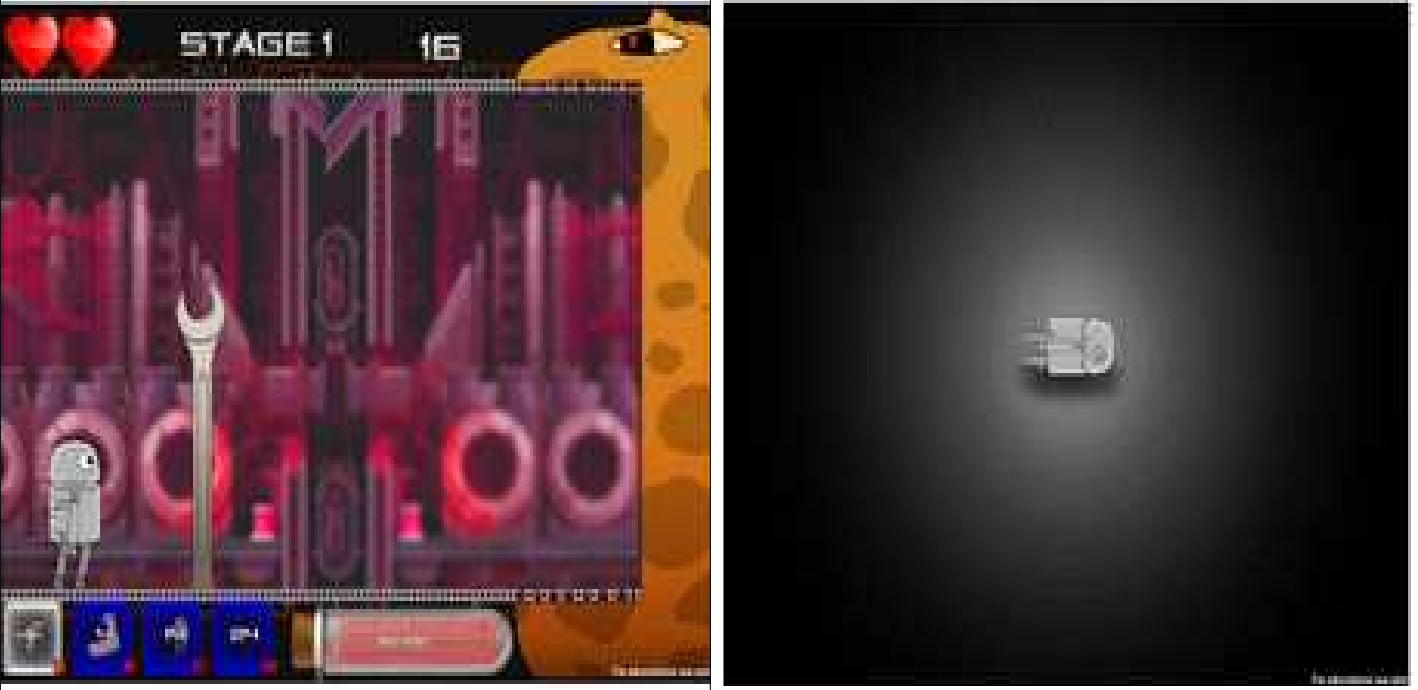
게 임 명	MiniMini
플 렷 폼	PC
게임화면	
	
게 임 소 개	<ol style="list-style-type: none"> <li>1. 퍼즐을 풀어가며 마지막 스테이지까지 도달 하자.</li> <li>2. 같은색깔의 버튼을 통해 블록을 사라지게 혹은 나타나게 하자.</li> <li>3. 스테이지가 진행될수록 게임은 어려워 진다.</li> </ol>
개발 기간	2014.6.(2주)
개발 환경	Unity3D 4.5.1f3 + NGUI + C#
개발 인원	기획 : 황은영 프로그래밍 : 권성호 그래픽 : 이승균
소 감	자료구조의 중요성을 일깨워 준 게임이다. 스택, 리스트, 해시테이블등 프로그래머의 판단에 따라 적재적소에 사용하여야 소스코드를 보다 쉽게 알아볼 수 있고 또한 사용하기가 편해진다. 배열을 통해 블록을 생성하고 각 색깔에 맞는 블록들을 관리하기 위해 리스트에서 관리를 하였다. 또한 동적으로 메모리를 생성하면 게임이 많이 느려지는 현상을 관찰 할 수 있었다. 이 게임을 통해 오브젝트 풀이라는 시스템을 왜 사용하는 지 이해할 수 있는 계기가 되었다.

## 5. SpaceShooter

<b>게 임 명</b>	SpaceShooter
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>- 유니티 기본게임인 SpaceShooter를 플레이메이커로 재구성</li> <li>- 오브젝트(FSM개수) <ul style="list-style-type: none"> <li>- Background(1)</li> <li>- Done_Player(2)</li> <li>- Boundary(1)</li> <li>- Done_Asteroid 01~03(Templates)</li> <li>- Done_Enemy Ship(3)</li> <li>- Done_Bolt, Done_Bolt_Enemy(1)</li> <li>- Game Controller(2)</li> </ul> </li> </ul>
<b>개발 기간</b>	2014.8.(2주)
<b>개발 환경</b>	Unity3D 4.5.1f3 + PlayMaker
<b>개발 인원</b>	프로그래밍 : 권성호
<b>소감</b>	<p>코딩없이도 게임을 만들 수 있는 시대를 실감 할 수 있었다. 유니티에서 기본으로 제공하는 SpaceShooter라는 게임을 플레이메이커라는 유니티 애셋을 통해 상태머신으로 게임을 바꾸었다. 단 한 줄의 소스코드 없이 게임을 만들 수 있다는 사실에 허망함을 느꼈다. 언리얼4라는 차세대 엔진에서는 블루프린트라는 유니티의 플레이메이커와 유사한 기능을 가지고 있다고 한다. 그 기능으로 만들어진 아이폰의 유명게임으로 알려진 인피니티블레이드는 소스코드 없이 블루프린트로만 만들어졌다. 앞으로 게임 프로그래머로써 살아남기 위해 남들과 다르게 더욱 뛰어난 실력을 가지겠다는 결심을 하였다.</p>



## 6. JAY's fActory

<b>게 임 명</b>	JAY's fActory
<b>플 랫 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<p>계속해서 함정을 생성하는 적 로봇을 무찔러, 아이템을 획득하여 최고 성능 로봇이 되어라!</p> <p>좌측으로 스크롤 되는 방식(우측으로 이동하는 느낌)</p> <p>적 로봇은 좌측에서 랜덤으로 3가지 장애물 중 하나씩 소환한다.</p> <p>로봇 JAY는 일정 간격으로 자동 공격을 한다.</p> <p>적 로봇은 HP가 존재하며, 적 로봇의 HP 0이 되면 다음 스테이지로 갈 수 있다.</p> <p>스테이지 클리어시 아이템을 획득한다.(능력치 상승)</p> <p>스테이지마다 보스의 능력치도 향상된다.</p>
<b>개발 기간</b>	2014.10.29 ~ 2014.11.25
<b>개발 환경</b>	Unity3D 4.5.1f3 + NGUI + C#
<b>개발 인원</b>	<p>기획 : 황은영</p> <p>프로그래밍 : 권성호</p> <p>그래픽 : 이수현</p>
<b>소감</b>	<p>이산수학과 C언어 기본 책에는 비트연산이 수록되어있다. 이 게임을 만들기 전에는 비트연산은 임베이드프로그래밍에서만 사용될 줄 알았다. 이 게임에서 스킬을 총 4가지이며 각각의 스킬은 중복효과를 가지게 된다. 만약 단순히 불리언변수로 처리하게 된다면 사용되는 예외처리는 스파게티소스처럼 꼬이게 되는 환경을 볼 수 있었다. 하지만 간단한 비트연산으로 스킬의 중복효과 및 메모리 절약 효과를 가질 수 있었다.</p>



## 7. Guardians of the Galaxy

<b>게 임 명</b>	Guardians of the Galaxy
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>• 태양계를 침략한 외계인으로 부터 지구 최종병기를 사용하여 태양계를 지켜라</li> <li>• 우주를 배경으로 최종병기(인공위성)를 조종한다.</li> <li>• 행성에 착륙한 우주선에서 외계인들이 생성된다.</li> <li>• SpaceBar로 레이저포를 발사한다. 총 5발 발사 가능하다.</li> <li>• 스테이지 클리어 시 모드 선택에서 행성에 맞는 모드를 선택한다.(힘, 스피드, normal)</li> <li>• 무중력 공간이므로 미세한 컨트롤이 필요하다.</li> <li>• 적 생성 한도 치 내에서 제한시간을 버텨라.</li> </ul>
<b>개발 기간</b>	2014.12. 1 ~ 2014.12.14
<b>개발 환경</b>	Unity3D 4.5.1f3 + NGUI + C#
<b>개발 인원</b>	기획 : 문영권 프로그래밍 : 권성호 그래픽 : 이루다
<b>소감</b>	각각의 생성되는 적들에게 능동적으로 3가지의 패턴을 넣고 싶었습니다. 단 순히 오른쪽, 왼쪽, 멈춤의 간단한 패턴이지만 실시간적으로 랜덤의 변수에 따라 결정하는 AI를 넣었습니다.

## 8. RunTurtle

<b>게 임 명</b>	RunTurtle
<b>플 렷 폼</b>	IOS 7.0
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>• 날라 오는 헬리콥터를 피해 최대한 멀리 도망가라</li> </ul>
<b>개발 기간</b>	2014.12. 1 ~ 2014.12.14
<b>개발 환경</b>	Xcode5 + Cocos2d 3.x.x + ObjectC
<b>개발 인원</b>	프로그래밍 : 권성호
<b>소감</b>	C언어를 기초로 하는 프로그래밍을 떠나 오브젝트-C기반의 Cocos2D 3을 다룰 수 있었던 좋은 경험이었습니다. 더 나아가 Cocos2d 3관련 책이 없어 기존 버전과의 차이를 혼자 공부하여 변환하는 연습을 통해 책이 아닌 레퍼런스를 보고 프로그래밍을 할 수 있는 넓은 시야를 가지게 하였던 게임입니다.


## 9. Ogre 기술데모

<b>게 임 명</b>	Ogre 기술데모
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>Ogre엔진을 통해 애니메이션 과 텍스트 파티클, UI등을 공부</li> </ul>
<b>개발 환경</b>	OgreSDK_vc11_v1-9-0 + C++ + VS2012
<b>개발 인원</b>	프로그래밍 : 권성호
<b>소감</b>	<p>기존의 상용화되었던 엔진이 아닌 Ogre라는 3D무료 엔진을 접할 수 있었던 기회였습니다. C++기반의 엔진이기 때문에 C++코딩실력을 늘릴 수 있었던 계기가 되었습니다. 또한 각각의 프로젝트 세팅의 중요성을 알 수 있었습니다. 렌더링 파이프라인을 다시 한번 복습할 수 있었으며 게임엔진의 사용법을 터득할 수 있었습니다.</p>

# 10. DirectX9 프레임워크 제작

<b>게 임 명</b>	DirectX9 프레임워크 제작
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>마우스, 키보드, 씬 전환, 카메라, 오브젝트, 사운드 관리 프레임워크 제작</li> </ul>
<b>개발 환경</b>	DirectX9.0 + C++ + STL + fMod + VS2012
<b>개발 인원</b>	프로그래밍 : 권성호
<b>소감</b>	<p>게임을 만들 때 무엇이 필요한지 다시 한 번 되새길 수 있었던 프로젝트였습니다. 입력장치와 출력장치, 게임의 사운드와 카메라, 충돌체 관리, 씬 관리 등 기본적으로 게임을 만들 때 필요한 사항들을 점걸 할 수 있었던 계기가 되었습니다. 또한 심도 있는 DirectX9의 공부로 3D게임 프로그래밍의 이론부터 실전까지 복습할 수 있었던 프로젝트 였습니다.</p>

# 11. Erica(졸업작품)

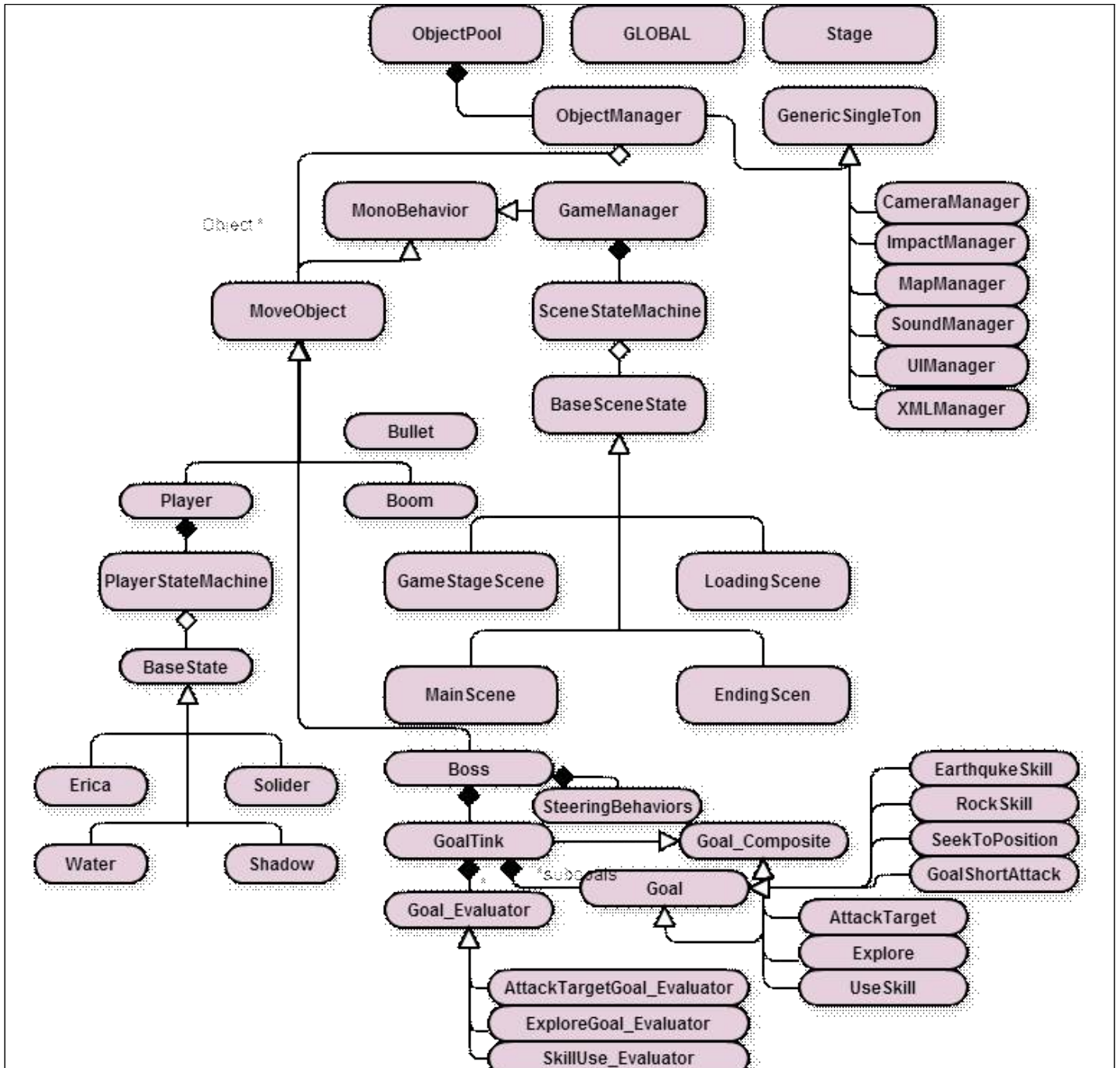
<b>게 임 명</b>	Erica(졸업작품)
<b>플 렷 폼</b>	PC
<b>게임화면</b>	
	
<b>게임 소개</b>	<ul style="list-style-type: none"> <li>• 변신을 활용하여 퍼즐을 풀고, 보스를 물리쳐 스테이지를 돌파하여 사라진 주인을 찾아라!</li> <li>• 모든 스테이지를 클리어 하여 주인을 찾아야 한다</li> </ul>
<b>개발 기간</b>	2014.7. 1 ~ 2014.11.20
<b>개발 환경</b>	Unity3D 4.5.1f3 + NGUI + C# + iTween
<b>개발 인원</b>	기획 : 황은영 프로그래밍 : 권성호 그래픽 : 김민아
<b>소감</b>	학교에서 마지막으로 만드는 게임인 만큼 저의 실력을 한층 더 업그레이드시키고 싶었습니다. 그리하여 지금까지 학교에서 배운 기술을 모두 다 넣고 싶은 마음 이었습니다. 단순 if문사용 이 아닌 목적중재를 통한 보스 AI제작에서부터 C#문법에 기초한 객체지향적 설계, 단순 스크립팅이 아닌 유니티 FrameWork로 제작 하였습니다.
<b>동영상 링크</b>	<a href="https://youtu.be/1VLCTr6Ni_Q">https://youtu.be/1VLCTr6Ni_Q</a>



## II. 기술요약 및 소스분석

### 1. Erica(졸업작품)

#### 가. 프로젝트 UML도표



UML 도표

## 나. 코드분석

### 1)템플릿 프로그래밍을 접목한 싱글톤 구조

<p><b>템플릿 싱글톤</b></p>	<pre> 1  using UnityEngine; 2  using System.Collections; 3 4  public class cs_GenericSingleton&lt;T&gt; : MonoBehaviour where T : new() 5  { 6      private static T _instance; 7 8      public static T Instance 9      { 10         get 11         { 12             if (_instance == null) 13             { 14                 _instance = new T(); 15             } 16             return _instance; 17         } 18     } 19 } 20 </pre>
<p><b>상속 예</b></p>	<pre> 1  using UnityEngine; 2  using System.Collections; 3 4  public class cs_CameraManager : cs_GenericSingleton&lt;cs_CameraManager&gt; 5  { 6 7      public float minPosX; 8      public float maxPosX; 9      public float minPosY; 10     public float maxPosY; 11     public bool showGizmo = true; 12 13     //카메라 흔들기 스킬시 14     private bool isShake; 15     private float shake; 16 } 17 </pre>
<p><b>코드분석</b></p>	<p>싱글 톤이란 인스턴스가 유일한 객체를 가지고 싶을 때 사용하는 패턴으로 프로세스에서 유일한 하나의 객체를 가지고 있기 때문에 관리가 편하다는 장점이 있다. 하지만 싱글 톤을 사용하고 싶을 때마다 동일한 코드를 반복해서 사용하기에는 프로그래머 입장으로써 귀찮은 일이지 않을까 생각한다. 동일 한 구조를 반복해서 사용한다면 템플릿(일반화 프로그래밍)을 사용하여 유일 한 인스턴스를 사용 할 수 있게 만들었다. 본 프로젝트의 모든 매니저 클래스는 템플릿 싱글톤 패턴을 상속받아 코드의 재 생산성을 키웠다.</p>



## 2)유한상태머신이 아닌 목적중재 보스 AI

<p><b>보스 목적중재 구조</b></p>	<div data-bbox="544 315 1337 862"> </div> <div data-bbox="427 969 520 1014"> <p><b>설 명</b></p> </div> <div data-bbox="592 898 1513 1088"> <ul style="list-style-type: none"> <li>- 탐색자들간의 우선순위를 판가름 하여 최적의 목표를 Goal_Think에 추가</li> <li>- 각 목표는 하위 목표를 가지고 있음</li> <li>- 하위 목표 완료시 상위목표 삭제</li> </ul> </div>
<p><b>cs_Goal</b></p>	<pre> 27 public enum status { active, inactive, completed, failed }; 28 29 public abstract class cs_Goal&lt;entity_type&gt; 30 { 31     protected GOALTYPE m_iType; 32     protected entity_type m_pOwner; 33     protected status m_iStatus; 34 35     protected void ActivateIfInactive() {...} 36     protected void ReactivateIfFailed() {...} 37 38     public cs_Goal(entity_type pE, GOALTYPE type) {...} 39 40     public abstract void Activate(); 41 42     public abstract status Process(); 43 44     public abstract void Terminate(); 45 46     public virtual void AddSubgoal(cs_Goal&lt;entity_type&gt; g) {...} 47 48     public virtual bool isActive() {...} 49 50     public virtual bool isInactive() {...} 51 52     public virtual bool isCompleted() {...} 53 54     public virtual bool hasFailed() {...} 55 56     public virtual GOALTYPE GetType() {...} 57 58     public virtual void Render() { } 59 }         </pre>

	설 명	<ul style="list-style-type: none"> <li>- cs_Goal클래스는 직접적으로 객체를 생성하지 않기 때문에 객체생성을 막기 위해 추상화 클래스로 제작되었다.</li> <li>- 이 클래스를 직접적으로 상속받는 객체(목적)은 하위 목적을 가질 수 없다.</li> <li>- 목적 완료 여부에 따라 active, inactive, completed, failed를 실행 끝에 반환한다.</li> </ul>
Goal_Compsite	<div data-bbox="403 539 1506 1402" data-label="Code-Block"> <pre> 1  using UnityEngine; 2  using System; 3  using System.Collections; 4  using System.Collections.Generic; 5 6  public abstract class cs_Goal_Composite&lt;entity_type&gt; : cs_Goal&lt;entity_type&gt; 7  { 8 9      protected List&lt;cs_Goal&lt;entity_type&gt;&gt; m_Subgoals; 10 11     protected status ProcessSubgoals() {...} 12 13     public cs_Goal_Composite(entity_type pE, GOALTYPE type) : base(pE,type) {...} 14 15     public override void Activate() {...} 16 17     public override status Process() {...} 18 19     public override void Terminate() {...} 20 21     public void AddSubgoal(cs_Goal&lt;entity_type&gt; g) {...} 22 23     public void RemoveAllSubgoals() {...} 24 25     ~cs_Goal_Composite() {...} 26 27     public virtual void Render() {...} 28 } </pre> </div>	<div data-bbox="403 1402 1506 1975" data-label="Text"> <p>설 명</p> <ul style="list-style-type: none"> <li>- cs_Goal_Think은 cs_Goal를 직접적으로 상속받는다.</li> <li>- cs_Goal_Think또한 객체를 직접적으로 생성하지 않기 때문에 추상화 클래스로 제작되었다.</li> <li>- cs_Goal과는 다르게 이 클래스를 상속받는 목적은 하위 목적을 가질 수 있으며 Process가 실행되는 동안 하위 목적의 프로세스를 실행하여 완료되거나 하위 목적 실패시 다음 목적으로 모두 완료를 하면 완료를 반환하게 되어있다.</li> </ul> </div>

## Goal\_Tink

```

1 using UnityEngine;
2 using System.Collections;
3 using System.Collections.Generic;
4 using GLOBAL;
5
6 public class cs_Goal_Think:cs_Goal_Composite<cs_Boss>
7 {
8     private List<cs_Goal_Evaluator> m_Evaluators;
9
10    private float ExploreBias;
11
12    private float SkillBias;
13
14    private float LongSkillBias;
15
16    private float AttackBias;
17
18    public cs_Goal_Think(cs_Boss pBoss):base(pBoss, GOALTYPE.goal_think){...}
19
20    public override status Process(){...}
21
22    public void Arbitrate(){...}
23
24    public bool notPresent(GOALTYPE GoalType){...}
25
26    public void AddGoal_MoveToPosition(Vector3 pos){...}
27
28    public void AddGoal_Explore(){...}
29
30    public void AddGoal_AttackTarget(){...}
31
32    public void AddGoal_UseSkill(){...}
33
34    public override void Terminate(){...}
35
36    public override void Activate(){...}
37
38    public override void Render(){...}
39

```

## 설 명

- 게임 에이전트의 두뇌에 해당된다.
- 탐색자로는 공격, 추격, 스킬사용 있다.
- 각각의 탐색자들은 캐릭터의 성향(0.0f~1.0)을 계산하여 각 상황에 맞는 우선순위 탐색자가 실행된다.
- 선택된 탐색자는 상위목적을 추가하게 된다.

탐색자 예시  
[공격 탐색자]

```

1 using UnityEngine;
2 using System.Collections;
3
4 public class cs_AttackTargetGoal_Evaluator : cs_Goal_Evaluator
5 {
6
7
8     public cs_AttackTargetGoal_Evaluator(float bias)
9     : base(bias)
10    {
11
12    }
13
14    public override float CalculateDesirability(cs_Boss pBoss)
15    {
16        float Desirability = 0.0f;
17
18        if (pBoss.isUseSkill())
19        {
20            m_dCharacterBias = Desirability;
21            return m_dCharacterBias;
22        }
23        if (pBoss.isTargetPresent())
24        {
25            Desirability = 1.0f;
26            m_dCharacterBias = Desirability;
27            return m_dCharacterBias;
28        }
29        m_dCharacterBias = Desirability;
30        return m_dCharacterBias;
31    }
32
33    public override void SetGoal(cs_Boss pEnt)
34    {
35        pEnt.GetBrain().AddGoal_AttackTarget();
36    }
37
38 }
39

```

설 명 [Goal\_Tink]의 Arbitrate멤버함수

```

public void Arbitrate()
{
    float best = 0;
    cs_Goal_Evaluator MostDesirable = null;

    foreach (cs_Goal_Evaluator curDes in m_Evaluators)
    {
        float desirability = curDes.CalculateDesirability(m_pOwner);

        if (desirability >= best)
        {
            best = desirability;
            MostDesirable = curDes;
        }
    }
    MostDesirable.SetGoal(m_pOwner);
}

```

- 각 탐색자는 Goal\_Think의 상태가 완료또는 실패일 때 실행 될 탐색자를 계산하게 된다. 위의 Goal\_Think의 멤버함수인 Arbitrate()에서 호출되며 성향이 더 강한 탐색자를 선별한 후 각 탐색자가 책임지고 있는 상위목적을 Goal\_Think에 추가하게 된다.

상위목적 예시  
(cs\_Goal\_Use\_Skill)

```

1 using UnityEngine;
2 using System.Collections;
3
4 public class cs_Goal_Use_Skill : cs_Goal_Composite<cs_Boss>
5 {
6
7     public cs_Goal_Use_Skill(cs_Boss pOwner)
8     : base(pOwner, GOALTYPE.goal_use_skill)
9     {
10    }
11
12     public override void Activate()
13     {
14         m_iStatus = status.active;
15
16         RemoveAllSubgoals();
17
18         if (m_pOwner.isEarthquakeSkill())
19         {
20             AddSubgoal(new cs_Goal_EarthquakeSkill(m_pOwner));
21         }
22         if (m_pOwner.isRockTrowSkill())
23         {
24             AddSubgoal(new cs_Goal_RockSkill(m_pOwner));
25         }
26     }
27
28     public override status Process()
29     {
30         ActivateIfInactive();
31
32         m_iStatus = ProcessSubgoals();
33
34         return m_iStatus;
35     }
36
37     public override void Terminate()
38     {
39     }
40 }

```

설 명

- 상위목적은 적절한 하위목적을 캐릭터의 상태에 따라 자신의 하위목적으로 추가한다.
- 하위목적의 완료여부에 따라 상위목적의 완료여부가 갈린다.



## 하위목적 예시 [cs\_Goal\_RockSkill]

```

1  using UnityEngine;
2  using System.Collections;
3
4  public class cs_Goal_RockSkill : cs_Goal<cs_Boss>
5  {
6      private float m_SkillTime;
7
8      public cs_Goal_RockSkill(cs_Boss pBoss)
9      : base(pBoss, GOALTYPE.goal_goal_rockskill)
10     {
11     }
12
13     public override void Activate()
14     {
15         m_iStatus = status.active;
16         m_pOwner.SetAnimation(BOSS_ANI_STATE.RockSkill);
17         m_SkillTime = 0.0f;
18         //m_pOwner.SetAniRockSkill();
19     }
20
21     public override status Process()
22     {
23         ActivateIfInactive();
24         m_SkillTime += Time.deltaTime;
25         if (m_SkillTime <= 1.0f)
26         {
27             m_iStatus = status.active;
28         }
29         else
30         {
31             m_pOwner.GetRockScript().Activate(cs_ObjectManager.Instance.GetGameObject("Rock").transform);
32             m_pOwner.GetRockScript().Initialize(m_pOwner,
33                 cs_ObjectManager.Instance.GetGameObject("Rock").transform);
34             m_pOwner.SetUseSkillIfFalse();
35             Debug.Log("RockSkill");
36             m_iStatus = status.completed;
37         }
38         return m_iStatus;
39     }
40
41     public override void Terminate()
42     {
43         m_SkillTime = 0.0f;
44         Debug.Log("TerminateRockSkill");
45     }
46

```

## 설 명

- 하위목적에서 직접적인 캐릭터의 행동을 수행하게 된다.
- 완료하지 못하면 실행상태를 반환하게 되며
- 행동을 완료하게 되면 완료를 반환하게 된다.

### 3)SteeringBehaviors

#### SteeringBehavior 멤버변수

```

3
4 public class cs_Boss_SteeringBehaviors {
5     [System.Flags]
6     private enum behavior_type
7     {
8         none = 1<<0,
9         seek = 1<<1,
10        arrive = 1<<2,
11        wander = 1<<3,
12        separation = 1<<4,
13        wall_avoidance = 1<<5
14    };
15
16    private Vector3 m_vWanderTarget;
17
18    private float m_dWanderJitter;
19    private float m_dWanderRadius;
20    private float m_dWanderDistance;
21
22    private float m_dWeightSeparation;
23    private float m_dWeightWander;
24    private float m_dWeightWallAvoidance;
25    private float m_dWeightSeek;
26    private float m_dWeightArrive;
27
28    private float m_dViewDistance;
29
30
31    private cs_Boss m_pBoss;
32
33    private Vector3 m_vSteeringForce;
34
35    private Transform m_pTargetPlayer;
36
37    private Vector3 m_vTarget;
38
39    private behavior_type m_iFlags;
40

```

#### 설 명

- 각 behavior\_type은 enum flag로 작성된다.
- enum flag로 작성되는 이유로는 상태가 한가지가 아니고 중복될 수 있기 때문에 비트연산으로 각각의 행동의 실행여부를 판단해야된다.
- 각 변수에는 움직임의 크기와 목표위치를 저장하기 위한 변수들로 이루어져 있다.



## Behavior 멤버 변수

```

49
50 public Vector3 Calculate()
51 {
52     m_vSteeringForce = new Vector3(0.0f, 0.0f, 0.0f);
53
54     //Debug.Log("CalculatePower");
55     m_vSteeringForce = CalculatePrioritized();
56
57     return m_vSteeringForce;
58 }
59
60 private Vector3 CalculatePrioritized()
61 {
62     Vector3 force;
63
64     if (On(behavior_type.seek))
65     {
66         force = Seek(m_pBoss.GetPlayerPosition()) * m_dWeightSeek;
67
68         if (!AccumulateForce(ref m_vSteeringForce, force)) return m_vSteeringForce;
69     }
70     else
71     {
72         m_vSteeringForce = new Vector3(0.0f, 0.0f, 0.0f);
73         return m_vSteeringForce;
74     }
75     return m_vSteeringForce;
76 }
77
78 private Vector3 Seek(Vector3 target)
79 {
80 }
81
82 private bool AccumulateForce(ref Vector3 sf, Vector3 ForceToAdd)
83 {
84 }
85
86 private enum Deceleration
87 {
88     slow = 3, normal = 2, fast = 1
89 }

```

## 설 명

- SteeringBehaviors는 행동 실행여부에 따라 힘을 계산한다.
- 계산된 힘은 최대힘을 넘지 않는 한에서 cs\_Boss에게 리턴된다.
- 주소에 대한 참조를 하지 않는다면 계산된 힘이 저장되지 않아 오류가 날 수 있다.
- 여기서 중요한 팁은 현재 Vector3를 사용하고 있지만 절대 사용하지 말아야한다. 이유인 즉슨 Vector3연산은 고비용을 요구하기 때문에 적시적소에 Vector2로 변환하여 프로세서의 비용을 줄이도록 노력해야 한다.

## SteeringBehavior 의 비트연산 부분

```

173
174     public void SeekOn() { m_iFlags |= behavior_type.seek; }
175     public void ArriveOn() { m_iFlags |= behavior_type.arrive; }
176     public void WanderOn() { m_iFlags |= behavior_type.wander; }
177     public void SeparationOn() { m_iFlags |= behavior_type.separation; }
178     public void WallAvoidanceOn() { m_iFlags |= behavior_type.wall_avoidance; }
179
180     public void SeekOff() { if(On(behavior_type.seek)) m_iFlags ^= behavior_type.seek; }
181     public void ArriveOff() { if(On(behavior_type.arrive)) m_iFlags ^= behavior_type.arrive; }
182     public void WanderOff() { if(On(behavior_type.wander)) m_iFlags ^= behavior_type.wander; }
183     public void SeparationOff() { if(On(behavior_type.separation)) m_iFlags ^= behavior_type.separation; }
184     public void WallAvoidanceOff() { if(On(behavior_type.wall_avoidance)) m_iFlags ^= behavior_type.wall_avoidance; }
185
186     public bool SeekIsOn() { return On(behavior_type.seek); }
187     public bool ArriveIsOn() { return On(behavior_type.arrive); }
188     public bool WanderIsOn() { return On(behavior_type.wander); }
189     public bool SeparationIsOn() { return On(behavior_type.separation); }
190     public bool WallAvoidanceIsOn() { return On(behavior_type.wall_avoidance); }
191

```

### 설 명

- 게임에서 비트연산은 패시브 스킬 뿐만 아니라 오브젝트의 움직임  
을 제어 할 때도 유용하게 사용된다.
- 만약 모든 플레그를 불리언으로 사용하게 된다면 최소 8배에서 그  
의 갑절의 메모리 손실이 일어나게 된다.
- 코드상에서도 if문으로 도배된 코드보다는 키고 끄고를 하여 더욱  
간편하게 코드를 작성할 수 있다.

### 3)ObjectPool

#### ObjectPool

```

1 using System.Collections;
2 using System.Collections.Generic;
3
4 public sealed class CGameObjectPool<T> where T : class
5 {
6     short count;
7
8     public delegate T Func();
9     Func create_fn;
10
11     Stack<T> objects;
12
13     public CGameObjectPool(short count, Func fn)
14     {
15         this.count = count;
16         this.create_fn = fn;
17
18         this.objects = new Stack<T>(this.count);
19         allocate();
20     }
21
22     private void allocate()
23     {
24         for (int i = 0; i < this.count; ++i)
25         {
26             this.objects.Push(this.create_fn());
27         }
28     }
29
30     public T pop()
31     {
32         if (this.objects.Count <= 0)
33         {
34             allocate();
35         }
36
37         return this.objects.Pop();
38     }
39
40     public void push(T obj)
41     {
42         this.objects.Push(obj);
43     }
44 }
45

```

#### 설 명

- ObjectPool이란 사용 될 오브젝트(메모리)를 게임 실행되기 전 메모리에 가지고 있는 개념이다.
- ObjectPool을 사용하는 이유로는 동적으로 메모리를 할당하게 되면 메모리의 장소의 선정과 메모리의 할당이라는 동작으로 인해 게임이 끊키는 형상이 보이게 된다.
- 하지만 정적으로 미리 메모리를 할당하게 되면 게임의 로딩의 시간은 걸리겠지만 부드러운 플레이를 할 수 있다.

## 사용 예시 (Object Manager)

```

1 using UnityEngine;
2 using System.Collections;
3
4 public sealed class cs_ObjectManager : cs_GenericSingleton<cs_ObjectManager>
5 {
6
7     private Hashtable mObject = new Hashtable();
8     private CGameObjectPool<GameObject> colliderCell;
9     private CGameObjectPool<GameObject> bullets;
10    private CGameObjectPool<GameObject> booms;
11
12    public void Initialize()
13    {
14        GameObject obj = Instantiate(Resources.Load("Prefabs/pfPlayer")) as GameObject;
15        obj.transform.parent = GameObject.Find("Game").transform;
16        obj.SetActive(false);
17        mObject.Add("Player", obj);
18
19        obj = Instantiate(Resources.Load("Prefabs/pfBoss")) as GameObject;
20        obj.transform.parent = GameObject.Find("Game").transform;
21        obj.SetActive(false);
22        mObject.Add("Boss", obj);
23
24        obj = Instantiate(Resources.Load("Prefabs/pfRock")) as GameObject;
25        obj.transform.parent = GameObject.Find("Game").transform;
26        obj.SetActive(false);
27        mObject.Add("Rock", obj);
28
29        this.booms = new CGameObjectPool<GameObject>(100, () =>
30        {
31            GameObject boom = Instantiate(Resources.Load("Prefabs/pfBoom")) as GameObject;
32            boom.transform.parent = GameObject.Find("Game").transform;
33            boom.SetActive(false);
34            return boom;
35        });
36        this.bullets = new CGameObjectPool<GameObject>(1000, () =>
37        {
38            GameObject bullet = Instantiate(Resources.Load("Prefabs/pfBullet")) as GameObject;
39            bullet.transform.parent = GameObject.Find("Game").transform;
40            bullet.SetActive(false);
41            return bullet;
42        });
43
44        this.colliderCell = new CGameObjectPool<GameObject>(1000, () =>
45        {

```

## 설 명

- ObjectManager는 sealed 클래스로 부모 클래스가 될 수 없게 만들었다.
- 자주사용하고 빈번히 삭제되는 오브젝트를 오브젝트풀이라는 자료구조에 넣어 사용하게 된다.
- 플레이어의 총알, 폭탄, 콜라이더 블록등은 생성과 삭제가 유동적이기 때문에 오브젝트 풀을 사용하지 않게 되면 위 오브젝트를 생성할 때마다 게임이 끊기게 되는 현상이 보일 수 있다.
- 추가적으로 람다식이라는 C#의 기능을 사용하여 함수포인터를 넘기고 있다.



### 3)유한상태머신

PlayerBaseState

```
using UnityEngine;
using System.Collections;

public class cs_PlayerBaseState
{
    protected Animator m_Anim;
    protected GameObject m_CurrentObj;
    protected GameObject m_ShadowObj;
    protected MOVESTATE m_CurrentMoveState;
    protected cs_Player m_pOwner;

    protected Vector3 m_GroundCheckRighLocalPosition;
    protected Vector3 m_GroundCheckLeftLocalPosition;
    protected Vector3 m_CeilingMiddleCheckLocalPosition;
    protected Vector3 m_CeilingRightCheckLocalPosition;
    protected Vector3 m_CeilingLeftCheckLocalPosition;
    protected Vector3 m_RightTopCheckLocalPosition;
    protected Vector3 m_RightMiddleCheckLocalPosition;
    protected Vector3 m_RightBottomCheckLocalPosition;

    protected bool BaseChangelsDone;

    public Vector3 GetCeilingMiddleCheckLocalPosition()
    public Vector3 GetCeilingRightCheckLocalPosition()
    public Vector3 GetCeilingLeftCheckLocalPosition()
    public virtual void Enter(cs_Player player)
    public virtual void Process(cs_Player player)
    public virtual void Exit(cs_Player player)
    protected virtual void SetAnimation(cs_Player player)
    protected virtual void Skill(cs_Player player) { }
}
```

설 명

- 게임의 엔티티는 유한한 상태를 가지고 있다. 각 상태에 따라 행동할 수 있는 움직임과 애니메이션이 존재한다. 위의 PlayerBaseState는 기본 상태로 기본적인 행동 및 프로세스를 구현하고 있으며 이 기본적인 상태는 더욱 체계화된 상태로 상속되어진다.

## 상태의 예 (PlayerSoliderState)

```

5 public class cs_PlayerSoliderState : cs_PlayerBaseState
6 {
7     private bool isTwoJumpEnable = true;
8
9     private float createBulletTime;
10    private float bulletTime;
11
12    private float createBoomTime;
13    private float BoomTime;
14
15    private static cs_PlayerSoliderState _instance;
16
17    private cs_PlayerSoliderState()...
39
40    public static cs_PlayerSoliderState Instance()...
48
49    public override void Enter(cs_Player player)
50    {
51        base.Enter(player);
52        createBulletTime = 0.5f;
53        bulletTime = 0.0f;
54
55        createBoomTime = 1.0f;
56        BoomTime = 0.0f;
57    }
58
59    public override void Process(cs_Player player)
63
64    public override void Exit(cs_Player player)
65    {
66        base.Exit(player);
67        bulletTime = 0.0f;
68        BoomTime = 0.0f;
69    }
70
71    protected virtual void SetAnimation(cs_Player player)
75
76    protected override void Skill(cs_Player player)
123 }

```

### 설 명

- 각각의 상태에서 가지는 행동이 존재한다.
- 진입과 상태를 벗어날 시 실행되는 고유의 행동이 존재한다.
- 싱글톤 패턴을 통해 유일하게 존재하는 상태로 만들어 바뀌어서는 안 되는 멤버변수 등을 저장 할 수 있다..

## PlayerStateMachine

```

4  public class cs_PlayerStateMachine
5  {
6      private cs_Player m_pOwner;
7      private cs_PlayerBaseState previousState;
8      private cs_PlayerBaseState currentState;
9      private cs_PlayerBaseState GlobalState;
10
11     public cs_PlayerStateMachine(cs_Player owner)
12     {
13         Process();
14     }
15
16     public void Exit()
17     {
18     }
19
20     public void Enter()
21     {
22         currentState.Enter(m_pOwner);
23     }
24
25     public void ChangeState(cs_PlayerBaseState pNewState)
26     {
27         if (currentState == pNewState) return;
28         previousState = currentState;
29         currentState.Exit(m_pOwner);
30         currentState = pNewState;
31         currentState.Enter(m_pOwner);
32     }
33
34     public void RevertToPreviousState()
35     {
36         this.ChangeState(previousState);
37     }
38
39     public void SetCurrentState(cs_PlayerBaseState s)
40     {
41         currentState = s;
42     }
43
44     public cs_PlayerBaseState GetCurrentState()
45     {
46         return currentState;
47     }
48
49     public void SetPreviousState(cs_PlayerBaseState s)
50     {
51         previousState = s;
52     }
53
54     public cs_PlayerBaseState GetPreviousState()
55     {
56         return previousState;
57     }
58 }

```

## 설 명

- PlayerStateMachine은 Player의 현재 상태와 출입조건을 실행하며 상태와 상태사이의 전환을 담당한다.
- Player은 상태를 직접적으로 가지고 있지 않으며 상태의 변환은 PlayerStateMachine을 통해 진행되며 현재 플레이어의 상태의 가지고 있다.
- 가지고 있는 상태의 Process는 PlayerStatemachine의 Process에서 호출된다.



## 시용예시 [Player의 Process]

```

326 public override void Process(Transform player)
327 {
328     Move(player);
329     Jump(player);
330     ChangePlayerState();
331     switch (this.m_PlayerStatus)
332     {
333         case PLAYERSTATUS.ERICA:
334             mPlayerStateMachine.ChangeState(cs_PlayerEricaState.Instance());
335             break;
336         case PLAYERSTATUS.SHADOW:
337             mPlayerStateMachine.ChangeState(cs_PlayerShadowState.Instance());
338             break;
339         case PLAYERSTATUS.SOLIDER:
340             mPlayerStateMachine.ChangeState(cs_PlayerSoliderState.Instance());
341             break;
342         case PLAYERSTATUS.WATER:
343             mPlayerStateMachine.ChangeState(cs_PlayerWaterState.Instance());
344             break;
345     }
346     mPlayerStateMachine.GetCurrentState().Process(this);
347
348     UpdateBullet();
349     UpdateBoom();
350 }

```

## 설 명

- Player의 상태(enum)가 변화게 되면 PlayerStateMachine은 상태를 바꾸게 된다.
- PlayerStateMachine은 현재의 상태를 Process하게 된다.

## 2. BlacketNight

원경과 근경	<pre> void Update () {     if(GameObject.Find ("Game").GetComponent&lt;Game_Script&gt;().isEnd == true)     {         return     }     GameObject Player = GameObject.Find ("Character");     if(Player.GetComponent&lt;Game_Character&gt;().isGameStart == false)     {         return     }     if(Input.GetKey(KeyCode.RightArrow))     {         float amtToMove = speed * Time.deltaTime;         transform.Translate(Vector3.left * amtToMove);     }     if(Input.GetKey(KeyCode.LeftArrow))     {         float amtToMove = speed * Time.deltaTime;         transform.Translate(Vector3.right * amtToMove);     }     Vector3 targetScreenPos =     UICamera.mainCamera.WorldToScreenPoint(gameObject.transform.position);     if(targetScreenPos.x + Screen.width * 0.5f &lt; 0 )     {         transform.position =         new Vector3(UICamera.mainCamera.transform.position.x + 2.6f,         transform.position.y,         transform.position.z);     }     if(targetScreenPos.x - Screen.width * 0.5f &gt; Screen.width )     {         transform.position =         new Vector3(UICamera.mainCamera.transform.position.x - 2.6f,         transform.position.y,         transform.position.z);     } } </pre>	
	설 명	- 원경과 근경효과를 위해 속도 조절

## Ray의 거리와 충돌판정을 통해 빛을 계산

```

if(gameObject.transform.FindChild("Light").transform.GetComponent<UISprite>().enabled == true)
{
    RaycastHit2D hit1 =
    Physics2D.Raycast(transform.position,transform.up,Vector3.Distance(gameObject.transform.position,trans
    form.FindChild("Light").transform.position)*1.9f);
    if(hit1.collider != null)
    {
        if(hit1.collider.name.Equals("Character"))
        {
            isPlayer = true
            return;
        }
    }

    Vector3 dir1 = new Vector3(0.9f, 7.0f, 0.0f);
    dir1.Normalize();
    RaycastHit2D hit2 =
    Physics2D.Raycast(transform.position,dir1,Vector3.Distance(gameObject.transform.position,transform.Fin
    dChild("Light").transform.position)*1.9f);
    if(hit2.collider != null)
    {
        if(hit2.collider.name.Equals("Character"))
        {
            isPlayer = true;
            return;
        }
    }

    Vector3 dir2 = new Vector3(-0.9f, 7.0f, 0.0f);
    dir2.Normalize();

    RaycastHit2D hit3 =
    Physics2D.Raycast(transform.position,dir2,Vector3.Distance(gameObject.transform.position,transform.Fin
    dChild("Light").transform.position)*1.9f);
    if(hit3.collider != null)
    {
        if(hit3.collider.name.Equals("Character"))
        {
            isPlayer = true;
            return;
        }
    }
    isPlayer = false;
}
isPlayer = false;
}

```

### 설 명

- Ray의 거리와 충돌판정을 통해 빛을 계산

### 3. Babel

<b>기술 요약</b>	<pre> public void replace(Item item) {     if((mItemSlot.Count - 1) == itemIndex)     {         return     }     print ("replace");     mItemSlot[itemIndex].transform.FindChild("Item").GetComponent&lt;UISprite&gt;().spriteName = item.ImgName;     mItemSlot[itemIndex].transform.FindChild("Item").transform.FindChild("Rank").GetComponent&lt;UILabel&gt;().text = item.Rank.ToString();     mItemSlot[itemIndex].GetComponent&lt;Babel_SlotCount&gt;().slot_Count = itemIndex;     mItemSlot[itemIndex].GetComponent&lt;Babel_SlotCount&gt;().isItem = true     mItemSlot[itemIndex].GetComponent&lt;Babel_SlotCount&gt;().mCharacter = Character;     mItemList.Add(item);     itemIndex++; } </pre>	
	<b>설 명</b>	- 아이템 인벤토리 제작

## 4. MiniMini

### 가)배열을 통한 블록생성

소스코드

```
public void MakeStage()
{
    float px = -4.95f;
    float py = 3.67f;
    float w = 0.32f;
    float h = 0.32f;

    string [] temp = Game_Stage.Stage[m_StageNum - 1];
    for(int i = 0; i < 24; i++)
    {
        string line = temp[i];
        float dx = px;
        for(int j = 0; j < line.Length; j++)
        {
            string ch = line.Substring(j, 1);
            if(ch == "0")
            {
                Character.transform.localPosition = new Vector3(dx, py, 10);
                dx += w;
                continue
            }
            I중략 .....
            if(ch == "6"||ch == "7"||ch == "8"||ch == "9")
            {
                GameObject Switch = Instantiate(Resources.Load("Prefabs/pfSwitch"+ch, typeof(GameObject))) as GameObject;
                Switch.transform.parent = transform;
                Switch.transform.localPosition = new Vector3(dx, py-0.055f, 10);
                Switch.GetComponent<Game_Switch>().mainObject = gameObject;
                m_SwitchList.Add (Switch);
                dx += w;
                continue
            }
            GameObject block = Instantiate(Resources.Load("Prefabs/pfBlock"+ch, typeof(GameObject))) as GameObject;
            block.transform.parent = transform;
            block.transform.localPosition = new Vector3(dx, py,10 );
            m_BlockList.Add(block);
            dx += w;
        };
        py -= h;
    }
}
```

설명

- 각 배열을 통해 블록을 생성하여 관리한다.

프로젝트 파일 및 실행파일 : [소화기를 쏘라, BlacketNight, Babel, MiniMini]

<https://drive.google.com/file/d/0BzusZZDBAoL3NEhTQWdvdlFITEE/view?usp=sharing>

5. SpaceShooter, JAY's fActory, Guardians of the Galaxy

포트폴리오 링크 :

[https://drive.google.com/open?id=1BeysUPY1Ge8rrnI49I3WSABwmPlekFzkifd4NHHLA\\_w](https://drive.google.com/open?id=1BeysUPY1Ge8rrnI49I3WSABwmPlekFzkifd4NHHLA_w)

8. RunTurtle

포트폴리오 링크 :

[https://docs.google.com/presentation/d/1GESY1RAGil82xCROboPi\\_FQ0jxusNhzh-f2ncMjYKMO/edit?usp=sharing](https://docs.google.com/presentation/d/1GESY1RAGil82xCROboPi_FQ0jxusNhzh-f2ncMjYKMO/edit?usp=sharing)

9. Ogre 기술데모, DirectX9 프레임워크 제작

프로젝트파일 링크 :

<https://drive.google.com/file/d/0BzusZZDBAoL3Y3Y5Q0RTNVVkaDQ/view?usp=sharing>

11. Erica[w졸업작품]

프로젝트파일 및 실행파일 :

<https://drive.google.com/file/d/0BzusZZDBAoL3YnFWTE9ZQIY5Z1E/view?usp=sharing>