碩士學位論文

게임 플레이 재생을 위한 키입력과 오브젝트 스테이트 기반의 저장 기법

濟州大學校 大學院

컴 퓨 터 工 學 科

李 承 元

2018 年 6月

게임 플레이 재생을 위한 키입력과 오브젝트 스테이트 기반의 저장 기법

指導敎授 李 尙 俊

李 承 元

이 論文을 컴퓨터工學 碩士學位 論文으로 提出함

2018 年 06 月

李承元의 工學 碩士學位 論文을 認准함

審査委員長 ㊞

委 員 ㊞

委 員 ㊞

濟州大學校 大學院

2018 年 06 月

Key Input and Object State Based Save Technic for Game Replay

SeungWon Lee
(Supervised by professor Sang-Joon Lee)

A thesis submitted in partial fulfillment of the requirement for the degree of Master of Science in Computer Engineering

2018. 06.

This thesis has been examined and approved.

Thesis director, Ho young Kwak

Thesis director, DoHyeun Kim

Thesis director, Sang Joon Lee

June 2018

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

목 차

그 림 목 차

국 문 초 록

게임 플레이 재생을 위한
키입력과 오브젝트 스테이트 기반의 저장 기법

컴퓨터공학과 이 승 원
지 도 교 수 이 상 준

기존에 게임 내에서 지원하는 동영상 녹화 기능은 단순히 플레이 내용을 저장하고 재생하는 기능만 지원한다. 본 논문에서는 동영상 재생 시에 플레이했던 화면뿐만이 아니라 다른 데이터도 볼 수 있도록 클라이언트를 이용하여 게임 상의 키 입력과 오브젝트 정보들을 저장해서 이를 이용한 영상 재생 방식을 제안한다. 제안한 방식을 이용하면 동영상 저장 공간이 줄어들어서 적은 용량으로 영상 리플레이가 가능하고 플레이했던 곳 이외의 내용까지 재생이 가능하므로 좀 더 풍부한 정보 공유가 가능해진다.

주제어 : 게임 리플레이, 클라이언트, 게임 스트리밍, 비디오 녹화, 키 입력 녹화, 오브젝트 상태 녹화

Abstract

Key Input and Object State Based Saving Technic for Game Replay

Lee, SeungWon
Department of Computer Engineering
Graduate School

Supervised by Professor Lee, Sang-Joon

Up to the present, played game contents have been just recorded and replayed for game streaming video recording function provided by the game company. This study deals with the method of recording game streaming video. By recording the key-input informations and object informations during game playing, the other data as well as played contents can be checked when streaming video is replayed. By the proposed method, it's possible to reduce the space to save streaming video, and therefore it is possible to replay streaming video in low capacity, and to share much more informations about other unplayed places as well as the played places.

Keywords : Game Replay, Game Streaming, Video Recoding, Key Input Recoding, Object State Recoding

I. 서 론

게임 유저가 게임플레이하는 이유는 게임에서 재미를 느끼기 때문이다 하지만 유저들은 게임을 플레이하는 것뿐만 아니라 다른 요소에서도 게임유저들은 즐거움을 찾는다 그 중 하나가 게임을 플레이하는 것을 녹화하는 것이다.

현재 게임관련 연구들을 보면 엔진이나 최적화에 대한 부분에 대한 연구가 많이 되고 있다[1][2]. 아니면 게임 엔진으로 특정장르에 게임을 만들거나 [3][4][5][6] 게임 중독성에 관한 인문학적 연구[7]가 주를 이루고 있다. 하지만 게임플레이를 저장하는 내용에 관한 연구는 연구된 바가 없기에 본 연구를 하게 되었다.

게임플레이 내용을 게임유저들이 녹화하는 이유는 여러 가지가 있다. 정보교환, 개인의 스킬을 높이기 위해 혹은 게임회사에 버그 제보를 위해 녹화를 한다.

게임을 플레이 하는 유저들은 좋은 아이템을 습득했을 때 스크린샷을 찍어서 공유하거나 어려운 컨텐츠를 클리어하는 영상들을 공유한다. SNS의 발달을 통해 알 수 있듯이 사람들은 자신의 행위들을 다른 사람에게 공유하는 것에도 재미를 느끼기 때문이다. 스크린샷의 경우에는 게임 내에서 직접 제공하는 경우가 많으나 영상의 경우에는 각 게임회사에서 제공하는 개임내의 기능을 사용하거나 직접 녹화 프로그램을 사용하는데 해당 기능은 항상 녹화하는 것이 아니라 미리 녹화버튼을 눌러 놓아야 하고 동영상 파일이기 때문에 용량이 굉장히 큰 경우가 대부분이다. 현재 온라인 게임에서 서비스하고 있는 게임들 중 동영상 녹화를 제공하고 있는 프로그램들은 많이 존재한다. 대표적으로 많이 사용하는 프로그램이 반디 캡쳐 라이브러리인데 영상 20초당 15MB정도의

용량을 가지게 된다. 따라서 해당 파일을 공유하기 위해서는 동영상 스트림 사이트에 직접 업로드 하거나 큰 용량의 파일을 공유하여야 한다.

그리고 게임 화면뿐만 아니라 다른 곳의 플레이 화면을 이용해 개임 내의 정보들을 더 알려주고 싶을 경우에 동영상의 정보만으로는 해당 내용을 표현 할 수가 없다.

게임을 제공하는 회사 입장에서도 각자 CS센터를 운영해 유저에게 버그를 제보 받게 되는데 텍스트를 가지고 버그 제보를 받게 된다면 어떤 내용인지 알아보기 힘들 수 있다. 동영상의 경우 용량이 크기 때문에 유저가 의도해서 저장한 영상을 올려야한다. 하지만 적은 용량의 리플레이 파일을 유저에게 공유 받는 시스템을 만들어 놓는다면 해당 버그의 내용을 훨씬 수월하게 공유 받고 쉽게 파악할 수 있게 된다.

그래서 본 연구에서는 해당 내용을 보여주게 될 주 대상인 게임을 플레이하는 다른 유저를 대상으로 게임 클라이언트가 가지고 있는 이미지 텍스쳐나 오브젝트 정보들을 이용해 플레이했던 내용을 좀 더 적은 용량으로 효율적으로 보이도록 하는 방법에 대해 연구한다.

플레이어가 입력하는 입력키 정보를 저장해서 유저의 플레이 내용을 다시 보여주는 입력키 레코딩 방식과 오브젝트들의 움직임 정보를 저장해서 플레이 내용을 보여주는 오브젝트 스테이트 저장 방식을 제안하였다.

이를 위해서 본 연구에서는 임의의 간단한 게임을 만들고 해당영상을 동영상 녹화 프로그램으로 녹화한 영상과 새로운 시도로 만들어진 데이터로 플레이한 영상을 비교해 정확도와 용량의 크기를 비교하고 추가적인 정보 제공이 가능함을 보인다.

II. 관 련 연 구

1. 게임 구성

게임을 제작하는 인원은 디자이너, 아티스트, 프로그래머로 구분할 수 있다. 디자이너가 게임의 방향성을 설정하고 아티스트가 게임 내에서 동작할 모델링이나 이미지를 제작하고 프로그래머가 해당 모델링이나 이미지를 게임 내에서 로직에 따라 보여지도록 하고 상황에 맞춰 이동, 변형해서 화면에 보여지도록 한다.

상용 서비스를 하고 있는 게임의 경우 추가로 새로운 시스템이나 모델링 이미지가 나올 경우에 시스템이 변경된 부분에서는 실행파일만 업데이트 하게 처리해주고 새로운 모델링이나 이미지의 경우에는 또 해당 데이터만 추가로 받아서 게임에 적용될 수 있도록 해준다. 해당 데이터의 경우에도 각자마다 로직을 가지고 압축해 용량을 줄여서 배포하게 되는데 업데이트 시간이 길어지게 되면 유저의 불만족 지수가 높아질 수 있기 때문이다.

하지만 그와는 별도로 유저들의 데이터 조작을 막기 위해서 클라이언트 실행할 때에 회사에서 배포한 파일과 같은지 데이터를 비교해 달라진 파일이 있으면 해당 파일을 재 배포하는 시스템을 갖추고 있다.

그래서 해당 게임을 플레이하는 유저들은 모델링이나 이미지를 같은 데이터를 가지고 있다고 보고 플레이 하게 될 수 있다.

일반적인 싱글 플레이의 게임은 <그림 1>과 같이 로직이 구성된다. 기본적으로 타이머가 돌면서 계속 유저의 입력을 받고 입력에 따라 상태가 변경되고

그 변경된 내용을 그려주는 방식으로 게임의 시스템은 구성된다. 멀티 플레이 게임의 경우엔 해당 입력이 오는 경우에 다른 컴퓨터에서 입력이 추가로 오게 될 수 있다. 멀티 플레이의 방식에는 가장 많이 사용되는 방식이 P2P 방식과 클라이언트 서버방식이 있는데 P2P방식에선 각자 본인의 클라이언트에서 상대방 오브젝트와 내 오브젝트를 모두 처리하고 입력이 있을 때 상대방에게서 그 정보를 받아온다. 그리고 중간 중간 각각의 클라이언트가 동기화를 해주는 방식으로 게임을 구성한다. 이를 간단히 그림으로 설명하면 <그림 2>와 같고 보통 FPS게임이나 RTS게임에 적합하다.

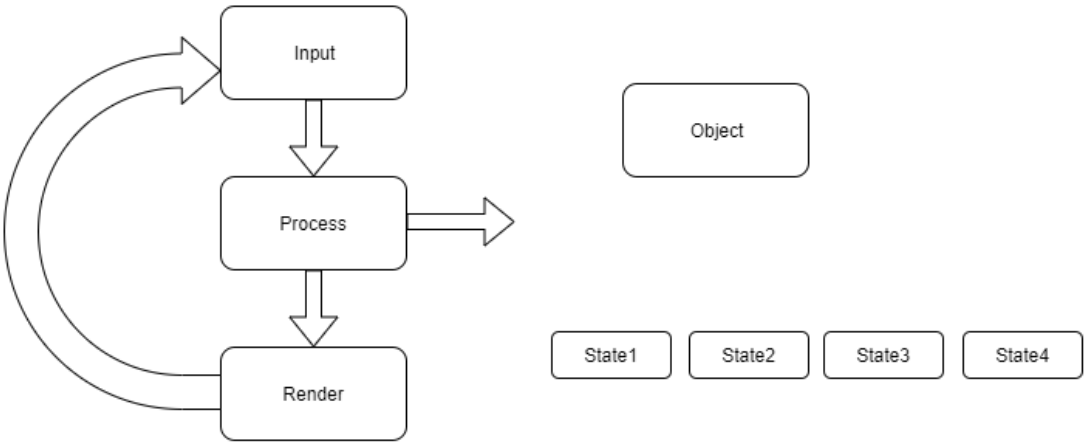

그림 1. 싱글플레이어 게임 처리 루프

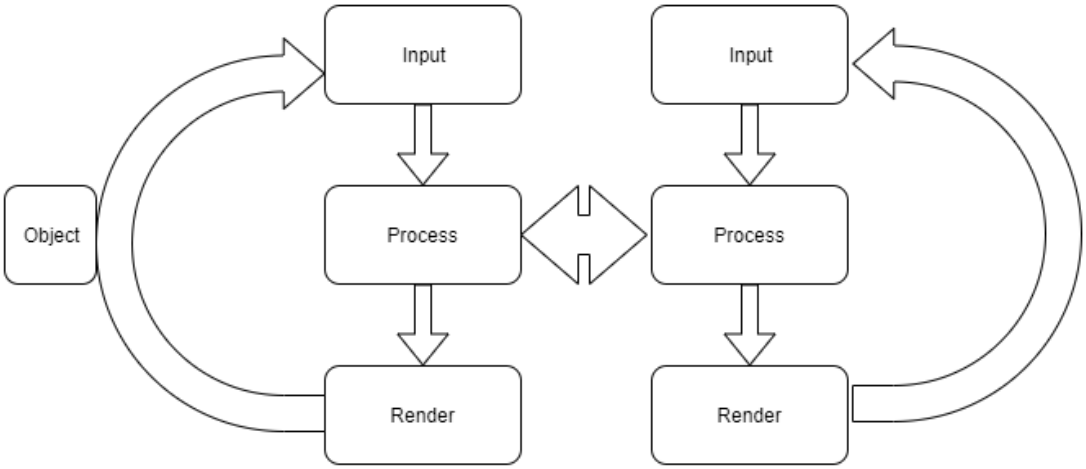

그림 2. P2P방식의 게임 처리 루프

또 다른 방식으로는 클라이언트 서버 방식이 있는데 해당 방식에서는 클라이언트는 입력과 렌더링 쪽에 치중하고 중요 로직은 다 서버에서 들고 있는 방식이다. 구현하는데 복잡도가 높고 서버 계산량이 많은편이라 MMORPG 게임에서 많이 사용한다. 이를 그림으로 표현하면 <그림 3>과 같이 표현할 수 있다.

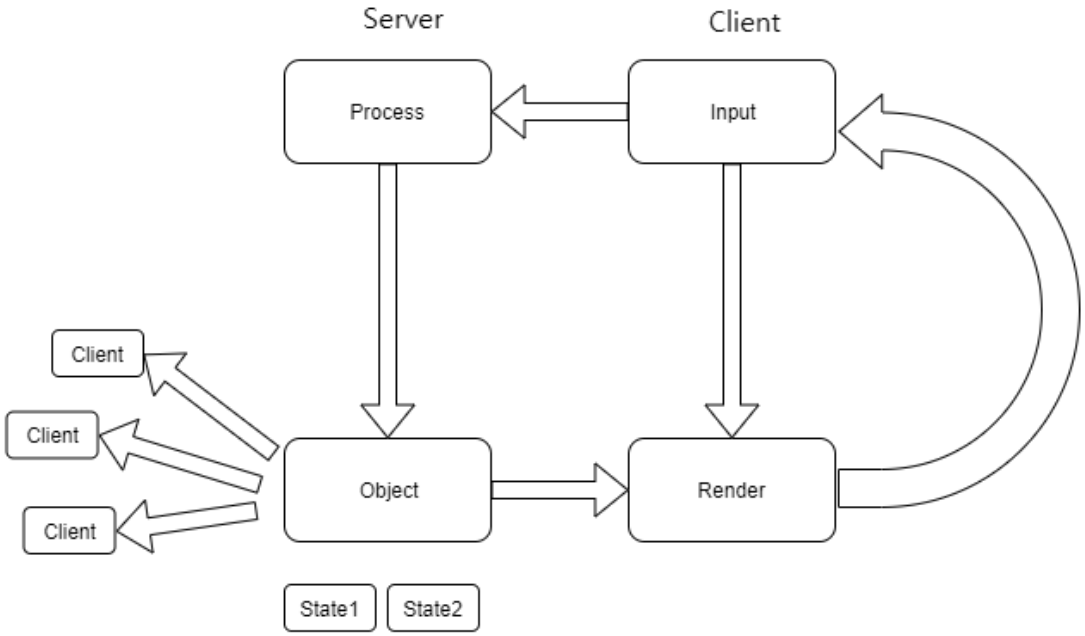

그림 3. Client-Server방식의 게임 처리 루프

클라이언트 서버방식의 게임처리에서는 해당 객체의 계산을 서버에서 처리하고 변경된 내용을 입력한 클라이언트뿐만 아니라 다른 클라이언트에게 보내줘서 동기화를 맞춰준다.

각자 게임을 구동하는 시스템이 다를 수 있어도 결국 대부분의 게임은 입력을 받아서 특정 대상의 상태를 변경하고 그려주는 방식이다. 그때 그려지는 대상의 상태에 대한 값과 입력 값을 이용해서 게임의 플레이를 대상에게 보여주는 방식으로 게임이 구성된다.

게임을 구성하기 위해서는 어느 장르의 게임이라도 보통 배경이 필요하고 캐릭터 혹은 오브젝트를 이동시켜서 게임을 진행해나간다. 비디오 게임의 종류

는 일반적으로 어드벤처 게임, 아케이드 게임, 스포츠 게임, MMORPG, 시뮬레이션 게임, 퍼즐게임, MOBA[8]으로 분류한다. 아케이드 게임은 게임의 장르라기보다는 오락실이나 코인을 이용해 하는 방식을 말하는 편이 많아 논외로 설정하고 어드벤처 게임, MMORPG, 롤플레잉 게임은 설계는 비슷하게 설계되어 지고 스토리를 풀어나가는 방식에 따라 게임을 분류하기에 같은 내용으로 로직을 설명하도록 하겠다.

해당 게임은 각자의 스토리가 있고 각자 특정한 캐릭터를 조작하면서 진행해 나간다. 어드벤처 게임은 이미 만들어진 상황에 따라 유저가 따라가지만 RPG는 캐릭터를 다르게 설정하거나 주인공이 스토리를 만들어나간다는 형식이 다르다. 간단하게 해당 방식의 내용의 게임에서 필요한 클래스 다이어그램을 그려보면 <그림 4>와 같이 표현 할 수 있다.

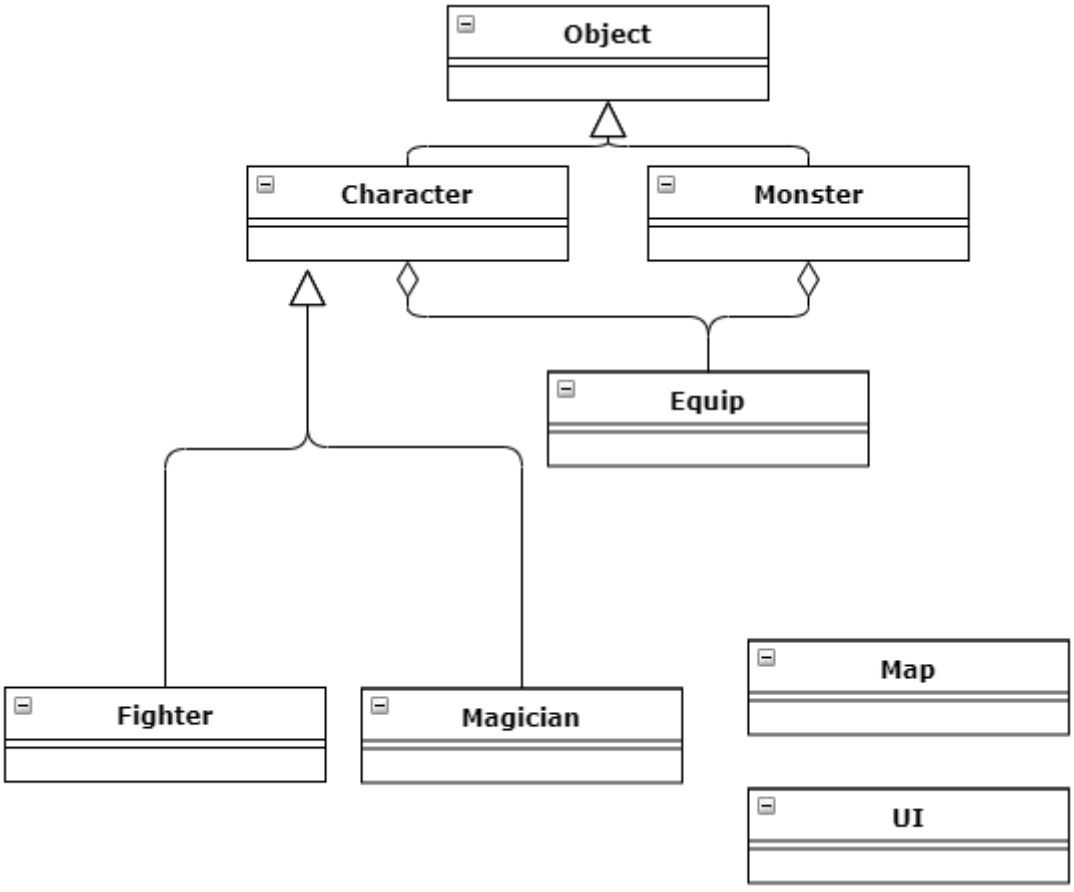

그림 4. 어드벤쳐, RPG 게임의 구성도

<그림 4>에서 유저는 입력을 통해 Character를 조작하면서 플레이 한다. 유저의 입력에 따라 Character 장비를 입고 전직을 하면서 직업을 바꾸어가며 몬스터를 사냥하며 게임을 플레이 해나간다. 실제의 게임에는 몬스터도 한 종류만 존재하는 것이 아니므로 몬스터 관련 클래스만 해도 굉장히 많은 수의 클래스가 생기고 직업도 많이 세분화 되겠지만 <그림 4>에서 확장된 방식으로 게임의 설계가 이루어 지게 된다.

스포츠게임은 축구, 농구, 야구의 구기류의 게임이나 레이싱게임도 스포츠에 포함하여 칭하는데 간단히 표현하기 위해 레이싱게임의 구성도를 그려보면 레이싱 게임의 클래스 구성도는 <그림 5>와 같다. 레이싱 게임의 경우에는 위의 그림과 다르게 Car라는 클래스를 유저가 조작해서 플레이한다. 입력에 따라 이동방향이 바뀌고 속력이 변경되며 맵에 있는 오브젝트와 혹은 상대방 자동차랑 충돌하면서 상호 반응하면서 플레이를 하게 된다.

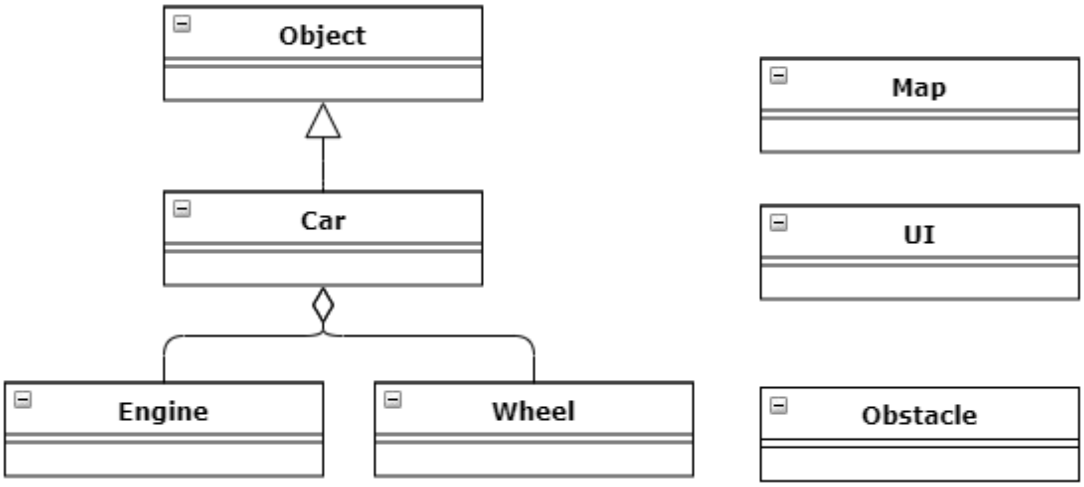

그림 5. 레이싱 게임의 구성도

그림의 예는 2개의 클래스 다이어그램만 간단히 표시했는데 대부분의 많은 게임들이 유저의 입력을 통해 특정 오브젝트가 조작이 되고 해당 오브젝트가 다른 오브젝트들과 상호반응해가면서 게임의 시간이 진행되고 유저는 그 결과를 보고 게임을 즐기게 되는 된다.

게임의 장르에 따라 약간씩의 차이는 있지만 기본적으로 배경을 구성하고 특정 오브젝트들이 유저의 행동에 의해 동작하고 로직이 구현되어 방식은 비슷하게 돌아간다고 할 수 있다. 그래서 본 논문에서는 이 공통적으로 동작하는 방식을 이용해 데이터를 저장해 게임의 플레이를 다시 재생 할 수 있는 방식에 대해 논한다.

2. 동영상 저장

동영상의 경우에는 코덱에 따라서 용량과 크기가 굉장히 많이 바뀌게 되는데 기본적으로 반디소프트의 캡쳐 라이브러리가 현재 게임회사에서 가장 많이 사용하는 라이브러리이기 때문에 반디 캡쳐 라이브러리에서 기본적으로 제공하는 방식으로 확인해보았다.

[BCL]

```
================================================================
파 일 기 본 정 보
----------------------------------------------------------------
파일명 : Capture 2008-09-10 09-37-43.avi
파일 포맷 종류 : AVI형 포맷
실제 파일 크기 : 16,452,884 Byte (15.7 MB)
순수 데이터 크기: 16,406,460 Byte (15.6 MB)
재생 시간 : 00:00:20 (20 Sec)
평균 전송률 : 6.46 MBits/Sec
----------------------------------------------------------------
비 디 오 스 트 림
----------------------------------------------------------------
코덱 종류 : MPEG4 (MP4V)
이미지 크기 : 1024 x 768 ( 1.33 : 1 )
평균 비트 전송률: 5.67 MBits/Sec
초당 프레임수 : 30.00 Frames/Sec
총 프레임수 : 611
----------------------------------------------------------------
오 디 오 스 트 림
----------------------------------------------------------------
코덱 종류 : PCM AUDIO (0x0001)
비트 전송률 : 768.00 KBits/Sec
샘플링 비율 : 24,000 Hz
채널 수 : 2 Channels (STEREO)
----------------------------------------------------------------
키 프 레 임
----------------------------------------------------------------
인덱스 상태 : 인덱스가 정상적으로 존재합니다.
총 키프레임 수 : 4
평균 거리 : 5.411 Sec
최대 거리 : 5.433 Sec
최소 거리 : 5.400 Sec
표준 편차 : 0.016 Sec
=========================[ By GomReader ]=========================
```

그림 6. Bandi Capture Library Benchmark[9]

반디소프트 공식 홈페이지 에서 제공하는 부분을 <그림 6>에서 보게 되면 20초의 영상을 제공하는데 약 15.7MB 정도의 용량이 필요하다. 한 프레임 당 약 28KB로 굉장히 높은 압축률을 지원하고 있지만 실제 게임플레이를 하게 되면 플레이 시간이 1시간이 넘어갈 수 있는데 1시간이 되면 2826MB나 된다. 더군다나 해당 인코딩 방식은 손실 압축이기 때문에 화질의 저하도 생길 수가 있다.

이러한 방식은 사용자가 많아지면 동영상을 저장을 위한 공간이 엄청나게 커져야 함을 의미한다. 이러한 방식은 사용자간이나 사용자와 게임회사간의 정보교환에 걸림돌이 될 것이다.

현재 용량을 중시하는 모바일관련 동영상에서 가장 많이 사용하고 있는 동영상코덱은 H.264 방식이다[10]. 해당 비디오 압축 원리는 크게 두 가지 방법을 이용해 영상의 압축을 하는데 Inter Prediction과 Intra Prediction의 방법을 이용해 영상을 압축한다. Inter Prediction 은 ME(Motion Estimation)을 적용하여, 이전 프레임(frame)과 현재 프레임의 동작 벡터(Motion Vector)로 다음 프레임의 움직임이 어떻게 될지 예측한다. Intra Prediction은 현재 프레임 내에서, 영역들의 연속성을 추측한다. H.264는 2003년 발표되어 현재까지 많이 사용되고 있고 많은 연구가 되고 있다[11]. 2013년에 H.265(HEVC)가 발표되었고 압축률은 뛰어나지만 H.264에 비해 5배 이상의 연산량을 요구하기에 높은 해상도의 영상을 재생, 편집을 위해서는 높은 사양의 CPU나 GPU를 필요로 한다. H.265에 관련해서도 많은 연구가 진행되고 있지만[12][13] 아직까지는 H.264가 더많이 사용된다. 따라서 본 논문에서는 H.264로 인코딩된 영상과 비교하도록 한다.

III. 이벤트 기반 게임플레이 관리방법

서론에 서술한 바와 같이 게임클라이언트는 입력을 받아서 해당 입력을 처리하고 그 입력에 따라 계산되어진 상태가 변경된 결과에 맞춰 화면 내에 오브젝트를 그려주는 방식으로 구현이 된다. 그 내용을 참고로 게임 내의 플레이를 입력 혹은 오브젝트 상태에 따라 저장하고 그에 맞춰 그려주는 방식을 본 논문에서 제안한다.

1. 키 인풋 레코딩 방식

제일 먼저 생각할 수 있는 방식이 유저가 입력한 게임의 패턴들의 데이터를 저장해 놓았다가 재생 시에 똑같은 패턴을 보여주는 방식이 있다. 해당 방식을 키 입력 레코딩 방식이라고 명한다.

키보드의 입력이 들어왔을 때 일반적인 캐릭터의 행동 패턴은 <그림 7>과 같다.

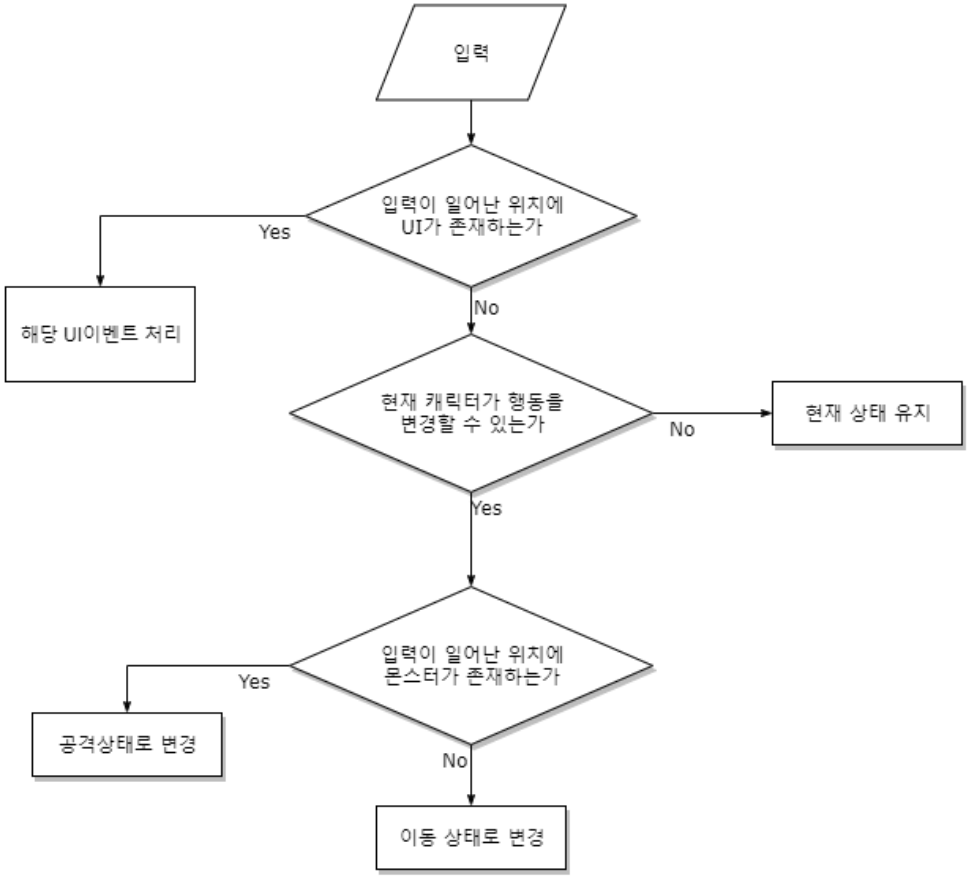

그림 7. Character State at KeyInput

하지만 키 입력방식을 위해서 <그림 8>과 같이 UI 입력이 일어나는 부분을 제외한 부분에서 현재 시간과 입력이 일어난 좌표를 저장해준다.

게임을 플레이 할 때에 키 입력 혹은 마우스 클릭이나 터치에 대한 값을 저장해서 타임라인에 해당 키 값을 저장해 두고 재생 할 때 해당 키 입력을 입력해주는 방식이다. 이 경우 키 입력 혹은 마우스 클릭 혹은 터치에 대한 값을 저장해서 타임라인에 해당 키 값을 설정해놓은 값을 영상 재생 시에 해당 키 입력을 입력해주는 방식이다. 아래의 <그림 9>는 해당 데이터를 저장한 파일의 저장 방식이다.

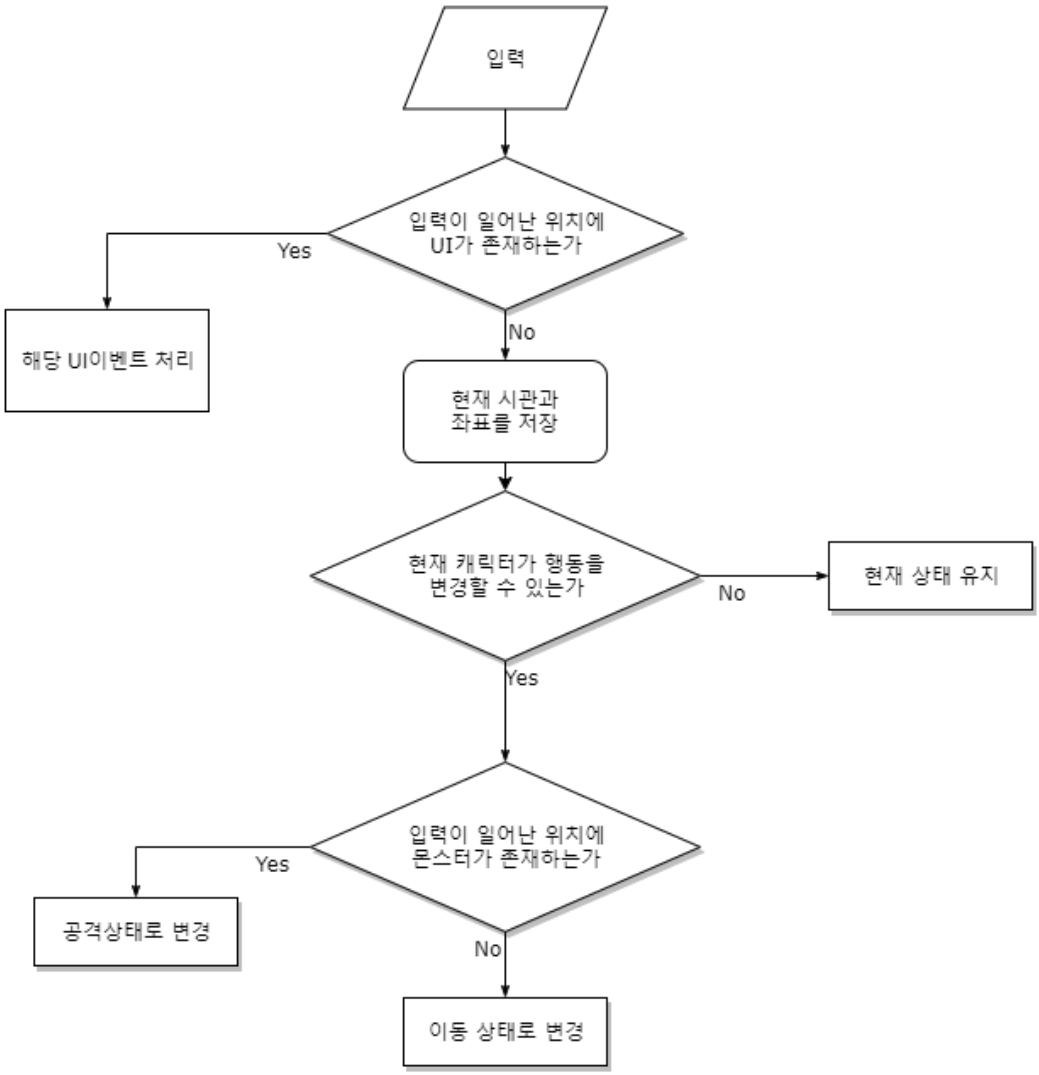

그림 8. Character State at KeyInput with Keyinput Recoding

```
<?xml version="1.0"?>
<touchDown>
  <timeline timeValue="10174" xPos="375" yPos="210">
  <timeline timeValue="12386" xPos="178" yPos="102">
  <timeline timeValue="13954" xPos="175" yPos="100">
  <timeline timeValue="17053" xPos="231" yPos="335">
</touchDown>
```

그림 9. Key Input Recoding FileFormat

샘플로 만든 데이터가 마우스 입력만 받도록 개발해놓은 게임이기 때문에 데이터는 타임 값과 위치 정보만 저장하게 해두었다. 혹시 키보드입력을 받는다면 키보드 입력도 추가해야 한다. 시간 값은 초당 여러 번 입력이 들어올 수 있으므로 밀리세컨드로 데이터를 저장하도록 한다.

해당 방식에서 플레이를 재생하는 방식은 리플레이시 유저의 키 입력은 캐릭터의 움직임과 상관없는 부분의 경우에만 동작하도록 설정하고 실제로 캐릭터의 움직임은 저장된 파일의 키 입력으로 처리하게 하는 방식이다.

2. 오브젝트 스테이트 방식

게임 내에서 오브젝트들이 구현되어 움직이는데 해당 움직임 패턴이 바뀌는 경우에 대한 값을 저장해서 재생할 때 이용하는 방식을 오브젝트 스테이트 방식이라 명하도록 한다.

입력키 레코딩 방식에 비해 오브젝트 수가 많아짐으로 인해 저장용량은 커질 수 있으나 정확도에서 훨씬 안정적인 상태가 된다.

플레이어가 컨트롤 하는 캐릭터의 경우 데미지를 입었는지 여부와 마우스 클릭 여부에 따라 캐릭터의 스테이트를 변경하고 몬스터의 경우 데미지를 입었는지 여부와 주변에 캐릭터가 존재하는지에 따라 공격이나 이동으로 스테이트를 변경한다. 스테이트가 변경될 때 변경된 스테이트와 현재 시간을 저장하도록 한다. <그림 7>의 상태에서 데이터를 저장하는 방식이 입력이 일어났을 때가 아닌 상태가 변경되었을 때에 데이터를 저장하도록 한다. <그림 10>과 같이 상태가 변경되었을 때 데이터를 저장하도록 한다. 상태가 변경될 시에 좌표도 저장하고 있는데 이는 혹시 모를 성능차이에 의해 잘못된 연산이 될

것에 대한 방어 작업으로 진행하였다[14].

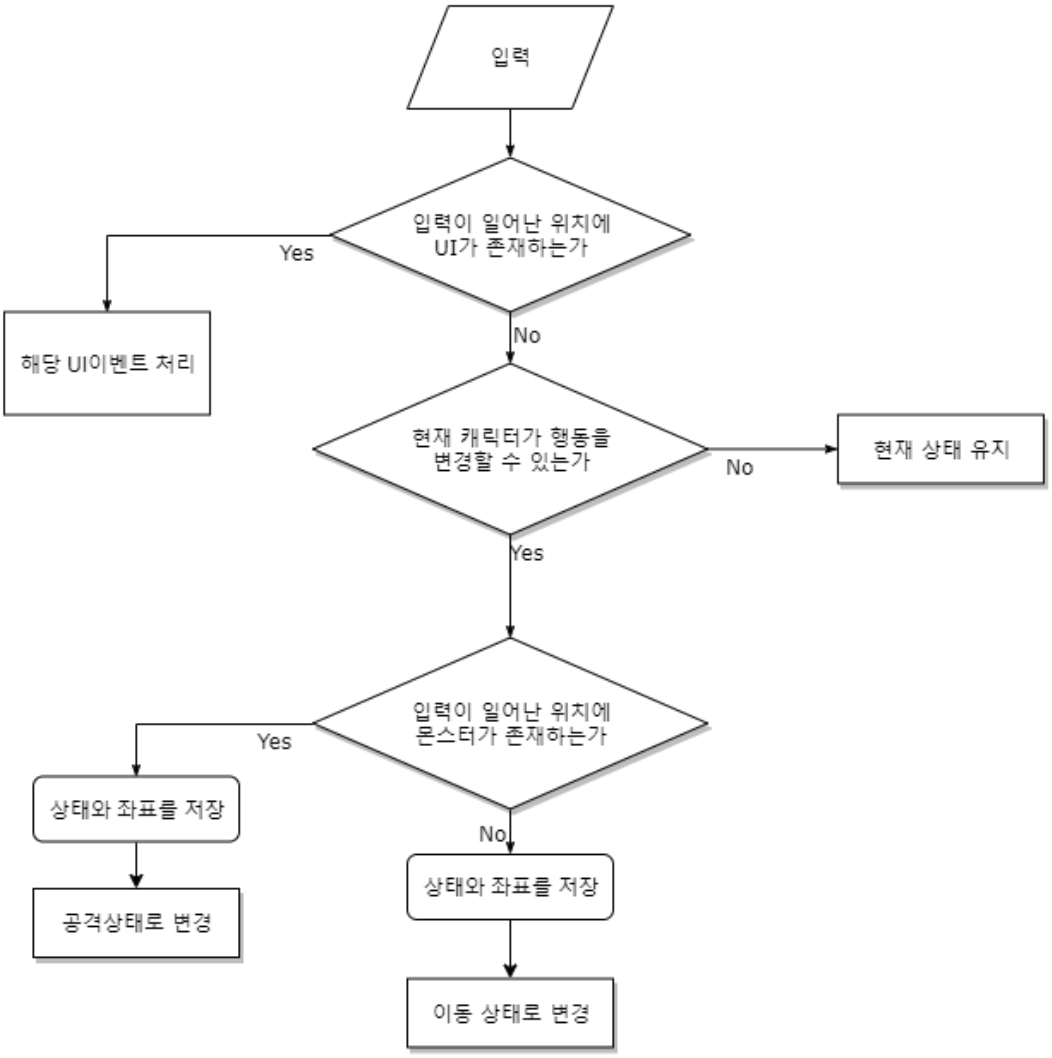

그림 10. Character State at KeyInput with Object State Recoding

캐릭터의 경우 키 입력에 의해 상태가 변경되지만 몬스터의 경우엔 유저의 입력에 직접 반응하는 것이 아니라 자체 로직에 의해 행동하고 반응한다. <그림 11>에서 보면 기본 상태에서 데미지를 입거나 캐릭터가 주변에 오게 될 경우 상황에 따라 몬스터의 상태가 변경되게 된다. 공격이 가능한 거리에 캐릭터가 존재한다면 해당 캐릭터를 공격하고 시야에는 들어왔으나 캐릭터가 공격거리 밖이라면 캐릭터에게로 이동한다. 캐릭터가 보이지 않는다면 주변을 순찰하는 로직으로 몬스터는 설계된다.

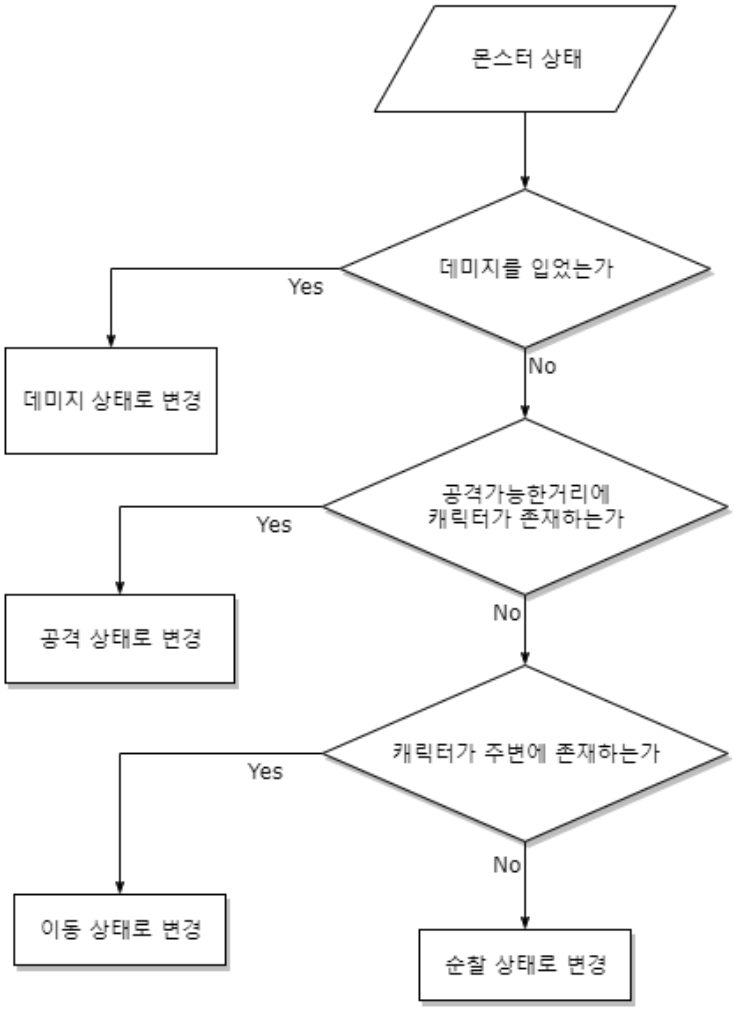

그림 11. Monster State Logic

몬스터도 상태가 변경될 때에 해당 데이터를 저장해 주도록 한다. 모바일 게임이나 PC게임이라고 하더라도 성능이 다르기에[15] 다른 위치, 다른 행동을 하게 될 수 있기에 몬스터의 값도 저장을 해준다.

<그림 12>는 몬스터의 상태 변경에 따라 데이터를 저장하는 타이밍을 설명해 놓았다.

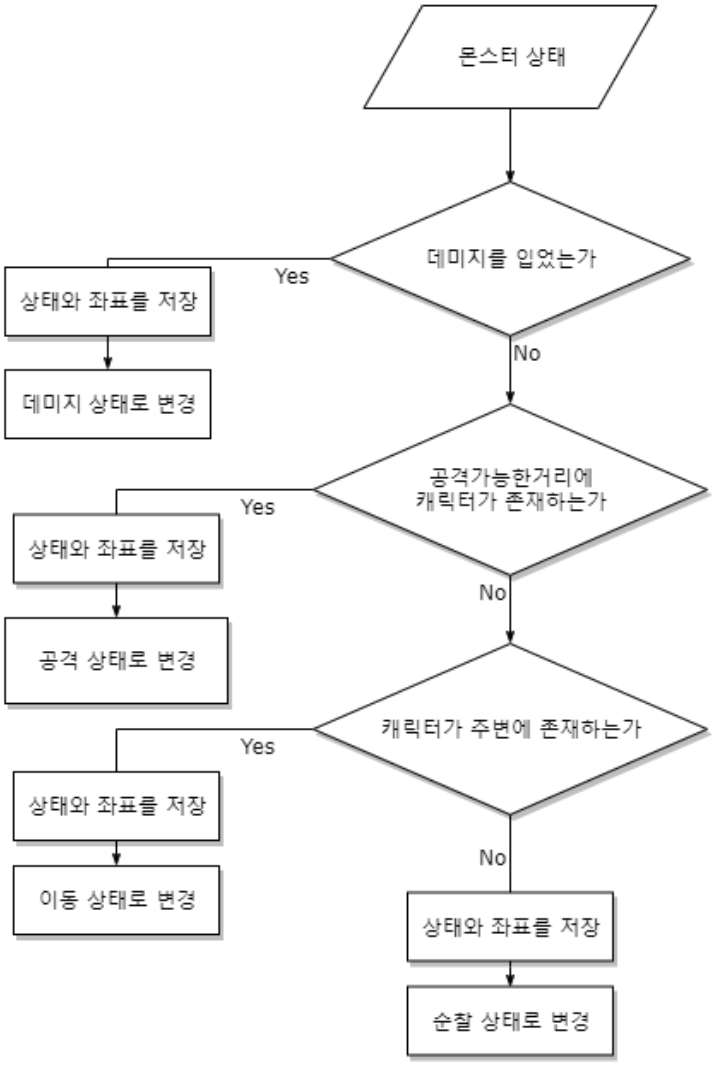

그림 12. Monster State with Object State Recoding

캐릭터의 상태와 몬스터의 상태를 변경해서 파일로 저장하는 방식은 <그림 13>과 같이 데이터를 저장하도록 했다. 입력키 레코딩 방식과 같이 밀리세컨드를 시간 단위로 설정하고 각각의 오브젝트들의 패턴이 변경되는 타이밍에 해당 스테이트의 내용을 저장하는 방식으로 구현하도록 한다.

```
<?xml version="1.0"?>
<objects>
  <monster type="goblin" spawnTime="5000"
        spawnXPos="210" spawnYPos="150">
    <timeline timeValue="7050" state="move"
        targetXPos="375" targetYPos="210">
    <timeline timeValue="12386" state="attack"
        targetXPos="385" targetYPos="210">
    <timeline timeValue="12389" state="damage"
        damageValue="50">
    <timeline timeValue="13715" state="attack"
         targetXPos="385" targetYPos="210">
    <timeline timeValue="12386" state="damage"
        damageValue="50">
    <timeline timeValue="12600" state="die">
  </monster>
  <character spawnTime="0"
        spawnXPos="300" spawnYPos="200">
    <timeline timeValue="10174" state="move"
        xPos="375" yPos="210">
    <timeline timeValue="12295" state="attack"
        xPos="178" yPos="102">
    <timeline timeValue="13954" state="move"
        xPos="175" yPos="100">
    <timeline timeValue="17053" state="move"
        xPos="231" yPos="335">
  </character>
</objects>
```

그림 13. Object State Recoding FileFormat

해당 방식에서 플레이를 재생하는 방법은 입력키 레코딩 방식과 같이 유저의 키 입력은 캐릭터의 움직임과 상관없는 부분의 경우에만 동작하도록 설정하고 게임플레이는 저장된 내용을 기반으로 오브젝트를 생성하고 오브젝트들이 움직이고 사라지도록 하는 처리를 해주었다. 오브젝트의 생성시간에 대한 정보를 'spawnTime'을 태그 앞부분에 설정해줘서 오브젝트들이 리젠 되는 시간을 설정해 주었고 하위 내용으로 게임에서 구현해준 스테이트인 'move', 'attack', 'damage' 태그로 분리해 두어 해당시간에 오브젝트의 움직임을 변동해 주도록 처리하고 'die' 태그가 나올 경우에 해당 오브젝트가 죽는 모션을 보여주고 이후에 오브젝트가 없어지도록 처리했다. 실제 라이브 서비스 중이거나 게임의 볼륨이 큰 게임의 경우엔 스킬과 상태의 가짓수만 해도 수십 개

가 넘어가고 공격이나 다른 스테이들도 많이 가지고 있게 되지만[16] 테스트를 위한 게임이기에 기본적인 이동, 공격, 데미지 상태의 스테이트만 가지고 있도록 개발 해 주었다. 추가로 구동장치 별로 계산하는데 걸리는 시간이 다를 수 있기에 오브젝트가 변경할 때에 위치 값을 넣어주어서 해당오류에 대한 방어 작업을 진행 해 주었다.

추후 플레이 재생할 경우에 각 오브젝트의 움직임이 기존 로직과 달라지는데 각 오브젝트의 동작을 보면 <그림 14>과 <그림 15>와 같이 표현할 수 있다.

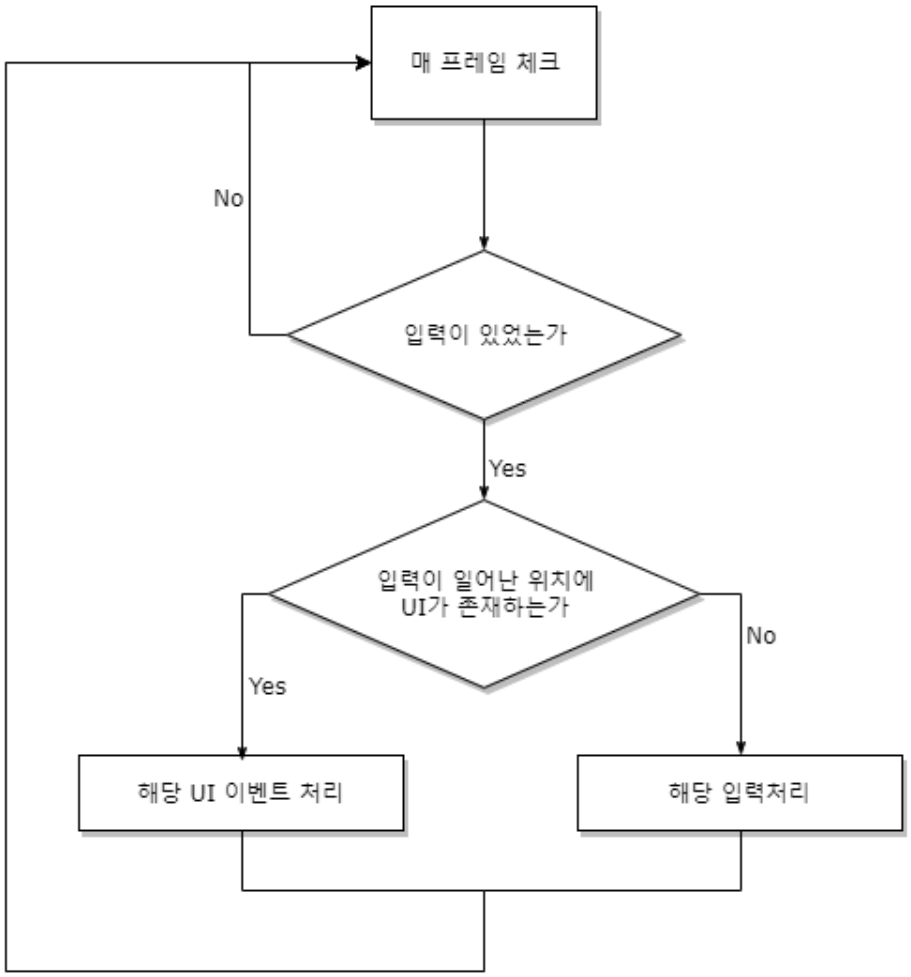

그림 14. Object State Change Situation Key Input Recoding

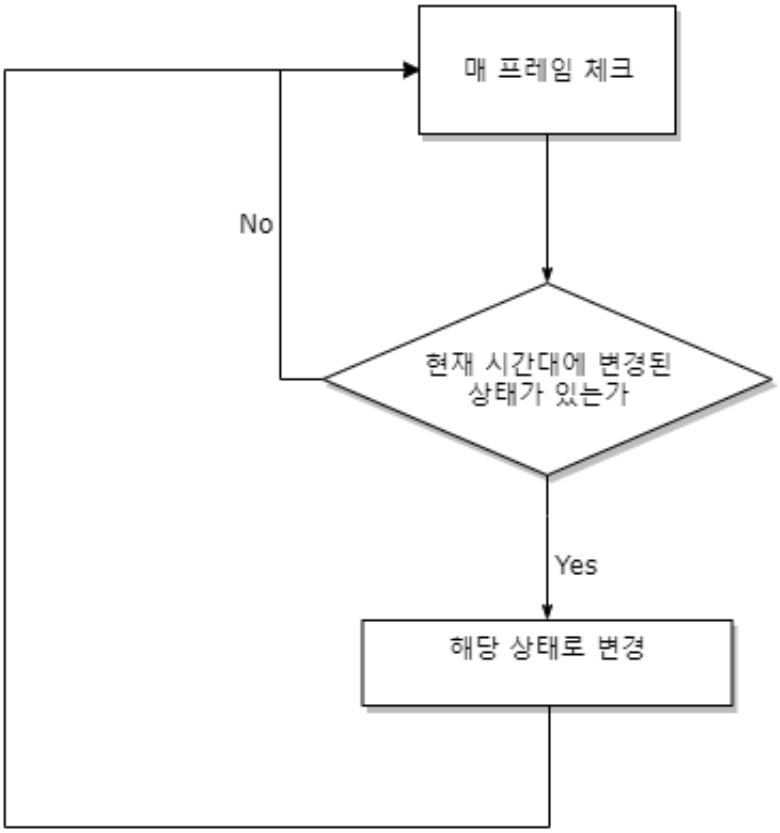

그림 15. Object State Change Situation Object State Recoding

<그림 14>는 키 입력 방식의 리플레이 재생 중 마우스 입력이 일어났을 때 하게 될 행동이다. 입력이 있었을 때가 아닐 때에는 계속 자동으로 게임 로직이 진행되고 입력이 있었을 타이밍이 되었을 때에만 해당 타이밍에 입력위치에 이벤트를 발생해준다. 현재 시간에 유저가 입력한 입력이 아니라 저장된 입력을 로딩해서 타이밍에 맞추어 입력을 진행해주는 방식으로 게임이 진행된다. <그림 15>는 오브젝트 스테이트 방식의 재생 방법으로 로직이 진행이 되다가 오브젝트가 변경된 내용이 있을 경우 해당 타이밍에 맞추어서 오브젝트를 저장되어 있는 값과 같은 상태로 변경해주고 위치를 맞추어 주도록 해주는 방식이다.

이와 같이 오브젝트에 대한 정보로 내용을 저장해서 해당내용으로 플레이 해주게 되면 추가적으로 플레이 재생 시에 아래의 <그림 16>과 같이 단순한 슬

라이드 바가 아닌 각각 플레이 했던 상황을 찾아서 재생을 시작할 수 있도록 처리할 수 있다.

그림 16. Replay Scene of Object State Recoding

IV. 실험 및 평가

1. 프로토타입 게임개발

본 논문에 사용 할 게임은 정식 서비스 할 게임이 아니기 때문에 간단히 만들 수 있는 게임엔진인 Unity3D를 이용하여 제작되었다. 엔진을 직접 만들어서 게임을 개발하는 연구들도 많이 진행되었으나[17] 가벼운 게임을 만드는 작업에는 Unity3D는 게임을 개발하기에 잘 만들어진 라이브러리로 언리얼 엔진과 함께 많은 회사에서 게임개발에 사용되는 라이브러리이다. 해당 엔진으로 많은 게임들이 개발되고 있다.

게임의 구성은 필드가 존재하고 해당 필드에 몬스터가 이동과 대기의 패턴으로 행동하다가 캐릭터가 주변에 나타나면 공격을 한다. 캐릭터는 몬스터를 사냥해 레벨업 해나가는 패턴의 게임이다.

현재 게임을 개발할 때에 많이 사용되는 디자인 패턴의 예로 싱글톤 패턴과 스테이트 패턴[18]이 있다. 본 연구에서는 스테이트 패턴에 대한 값을 저장해서 재생에 이용할 계획이기 때문에 스테이트 패턴을 이용해서 오브젝트의 상태를 조정하도록 한다.

스테이트 패턴은 게임 쪽에서 자주 사용하는 패턴이기 때문에 Unity3D State Machine[19]으로 자체적으로 해당 내용을 응용할 수 있도록 되어 있고, Unreal Engine State Machine[20]에서도 해당 패턴을 이용해 게임을 개발할 수 있도록 지원하고 있다.

몬스터는 Stand, Move, Attack, Damage의 패턴을 가지고 있고 캐릭터는

Dash, Move, Attack, Skill, Damage의 패턴을 가지고 행동한다. 캐릭터의 움직임은 화면을 클릭해서 컨트롤하는데 클릭한 위치에 몬스터가 존재할 경우 공격하고 클릭한 위치에 몬스터가 존재하지만 공격범위보다 멀리 있다면 Move패턴으로 이동한 후에 공격하도록 한다. 클릭한 곳에 아무것도 존재하지 않는다면 Dash로 빨리 이동하도록 처리해주었다. 우측하단에 있는 Skill을 사용하게 된다면 넓은 범위에 광범위 공격을 사용한다. 몬스터와 캐릭터 모두 공격을 받을 때 데미지 모션으로 변경되도록 작업해 주었다.

저장은 게임을 종료할 때 자동으로 플레이 할 때 입력했던 키 입력과 오브젝트 상태들을 자동으로 저장하도록 처리해 주었고 불러오기는 화면 우측상단에 있는 UI로 불러올 수 있도록 처리해 주었다.

게임을 구동한 후 1분가량 몬스터를 사냥하는 영상을 기존 동영상방식으로 녹화하고 저장된 용량과 비교해보고 다시 클라이언트를 구동한 후에 정상적으로 플레이가 저장되었는지를 확인해 비교해보도록 한다. <그림 17>와 <그림 18>은 구현한 게임의 스크린샷이다.

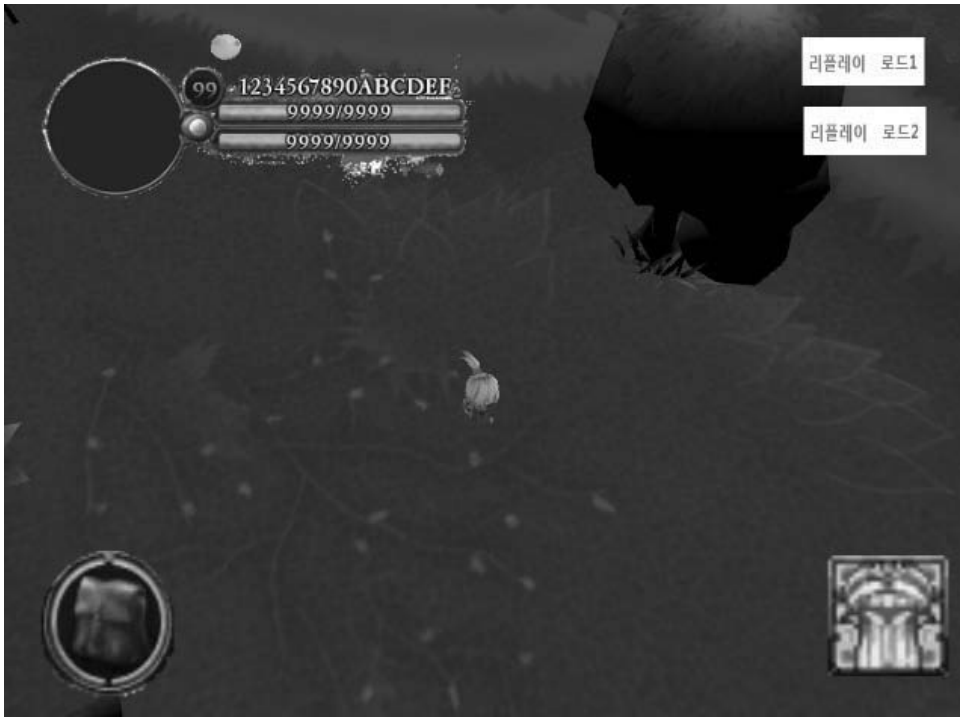

그림 17. Game ScreenShot1

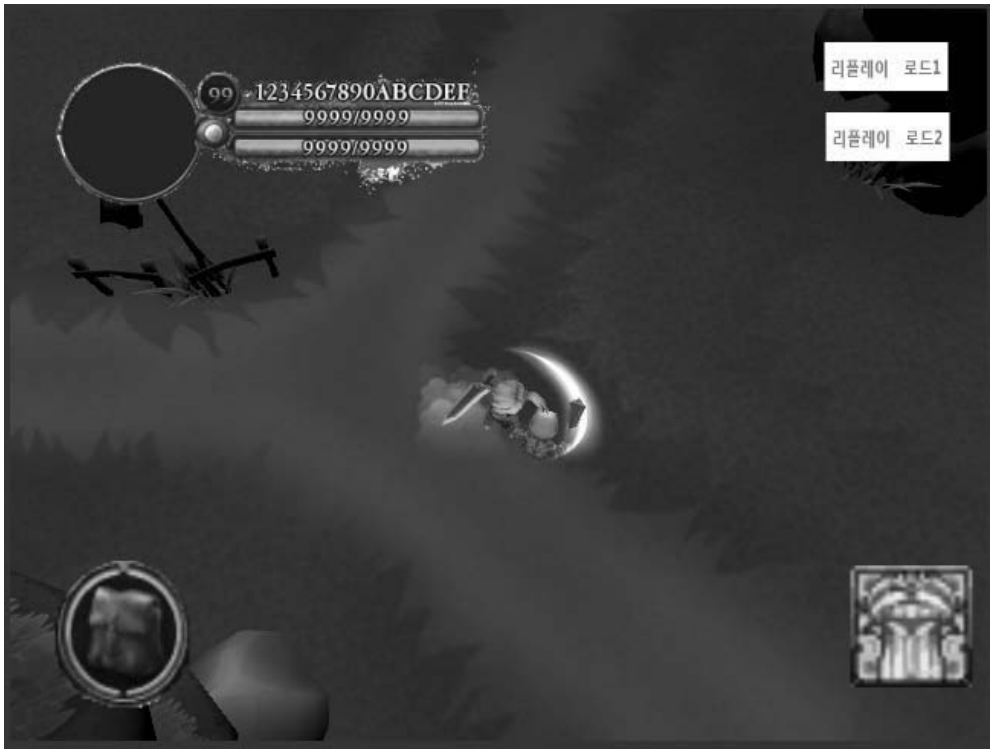

그림 18. Game ScreenShot2

2.영상 출력 결과 및 추가 기능 평가

게임을 플레이하는 영상은 640*480의 크기의 동영상으로 녹화했고 OhSoft의 Ocam[21] 이라는 외부 프로그램을 이용해서 동영상을 녹화했다. 플레이 영상은 게임을 켜고 캐릭터가 사망하기까지의 저장했고 1분간 키 입력은 초당 1~2회 가량 클릭해 캐릭터의 행동을 제어하도록 작업해주었다. 총 5회의 테스트를 진행했고 보통 40초에서 1분 이내에 캐릭터가 사망하였으며 그 결과로 나온 저장 파일의 크기는 <표 1>과 같다. Ocam의 경우 반디 캡쳐 라이브러리에서 제공한 벤치마크보다 작은 용량으로 저장되었는데 벤치마크에서 제공한 영상의 크기는 1024 x 768 이지만 테스트를 위한 영상은 480 x 320의 크기의 영상으로 저장하여 조금 더 작은 용량의 동영상이 저장되었다.

표 1. Result of GamePlay Recoding

파일크기	Video Recoding	Key Input Recoding	Object State Recoding
1회차	5,719,075 Byte	5,541 Byte	41,656 Byte
2회차	13,335,095 Byte	6,723 Byte	42,505 Byte
3회차	7,832,823 Byte	6,719 Byte	55,094 Byte
4회차	6,593,368 Byte	7,531 Byte	64,350 Byte
5회차	8,130,504 Byte	5,904 Byte	66,772 Byte
평균 값	8,322,173 Byte	6,483.6 Byte	54,075.4 Byte
파일 크기 비율	100%	0.08%	0.6%

비디오 녹화의 경우 화면이 얼마나 많이 변경되느냐에 따라 압축률이 달라지기에 영상의 크기가 많이 변경된다. 2회차에 진행한 테스트에선 마을의 먼 곳까지 이동을 했기에 화면이 많이 변동되어서 동영상의 크기가 커졌다. 하지만 키 입력 레코딩 방식과 오브젝트 스테이트 저장 방식의 경우 화면이 많이 변경되었다고 하더라도 저장 크기에는 차이가 없다. 화면을 직접 저장하는 방식

이 아니기 때문이다. 다만 오브젝트 스테이트 방식은 오브젝트들의 상태까지 저장하기에 맵에 위치한 오브젝트의 개수가 많아질수록 저장 크기가 커질 수 있다.

오브젝트 스테이트 저장 방식이 키 입력 레코딩 방식에 비해 저장 공간을 좀 더 사용하는데 게임의 경우 오브젝트들이 항상 정해진 위치에서 생성되는 것이 아니라 랜덤하게 맵에 배치되는 경우가 있을 수 있다. 그럴 경우에는 키 입력 방식의 경우 입력이 일어났더라도 해당 위치에 몬스터가 없거나 하게 되면 저장할 때의 내용과 플레이를 재생 했을 때 내용이 달라 질 수 있다. 또 게임이 고정적으로 한번 만들어지고 끝나는 것이 아니라 계속 변경이 될 수 있다. 몬스터의 체력이나 행동들이 변경될 수 있는데 그 경우에도 키 입력 레코딩 방식으로는 정확한 재현이 어려울 수 있기에 오브젝트 스테이트 저장 방식을 제안하였다.

따라서 퍼즐 게임이나 게임내의 오브젝트들이 랜덤하게 행동하는 일이 없을 경우에 키 입력 레코딩 방식을 사용하는 것이 나은 방법이고 오브젝트들이 랜덤하게 배치되고 행동하거나 업데이트가 자주 일어날 경우에는 오브젝트 스테이트 저장 방식을 사용하는 것이 더 좋은 방법이라 할 수 있다.

추가로 동영상이 아닌 새로 제시한 방식의 경우 몬스터의 움직임이나 키 입력에 따라서 저장 공간이 달라지는데 초당 1회의 키 입력을 해서 녹화를 진행한 결과이다. 해당 크기의 용량정도라면 유저가 서로 공유하기에도 굉장히 작은 용량으로 사용할 수 있고 해당 용량이라면 서비스하고 있는 각각의 회사에서도 리플레이로 각각의 회사에서 원하는 시스템도 구축할 수 있게 된다. 그리고 새로 제시한 기법을 이용하게 되면 용량이 훨씬 작아지는 것뿐만 아니라 그 주변 상황도 보이게 처리할 수 있는 장점이 있다.

<그림 19>을 보게 되면 리플레이가 재생되는 동안이지만 인벤토리 버튼을 클릭해 인벤토리 UI를 열어서 해당캐릭터가 어떤 장비를 착용하고 플레이했

는지에 대한 정보를 알아 볼 수 있도록 처리했다. 정식 서비스하는 게임이 아니기에 인벤토리 정보만 보여줄 수 있지만 각 회사에서 개발하기에 따라 어느 정도까지 추가 조작을 가능하게 할지의 여부를 정해서 추가적으로 개발할 수가 있다.

그림 19. Replay with UI Event

3. 추가데이터 압축

동영상의 경우 코덱과 기타 여러 가지 방법으로 데이터를 줄이기 위해 작업을 하기 때문에 새로 연구한 방식에도 결과물로 나온 텍스트 파일을 만들기 전에 파일용량을 줄이기로 한다. 압축 파일은 Unity3D엔진에서 가장 많이 사용하고 있는 LZF compression[22]을 사용하기로 한다. 실제로 키 입력 방식과 Object State Recoding 방식의 데이터를 5번 가량 저장해본 결과 <표 2>와 같이 80%의 데이터가 추가로 압축 되어 용량이 더 줄어드는 것을 확인 할 수 있었다. <표 1>의 데이터와 같은 데이터로 해당 녹화를 진행 할 때에 같이 압축해서 저장해 크기를 비교해 주었다.

표 2. 텍스트 추가 압축 결과

키입력방식 압축전	키입력방식 압축후	스테이트 방식 압축전	스테이트 방식 압축후
5,541 Byte	693 Byte	41,656 Byte	1,259 Byte
6,723 Byte	451 Byte	42,505 Byte	2,323 Byte
6,719 Byte	712 Byte	55,094 Byte	1,522 Byte
7,531 Byte	801 Byte	64,350 Byte	1755 Byte
5,904 Byte	450 Byte	66,772 Byte	1,907 Byte

V. 결론

입력키 레코딩 방식과 오브젝트 스테이트 저장 방식을 이용하게 되면 용량이 크게 줄어드는 것을 확인했다. 오브젝트가 무작위하게 배치되는 경우에는 오브젝트 스테이트 방식을 사용하는 것이 그렇지 않은 경우 입력키 레코딩 방식을 사용하는 것이 용이하다.

본 논문에서는 적은 용량으로 게임 플레이 내용을 다시 재생 하는 방법에 대해 키 입력과 오브젝트의 스테이트를 이용해서 재생하는 방법에 대해 연구했다.

유저가 버그를 제보해줄 때 텍스트만 적어서 제보해주거나 버그를 재현하는 방법을 유저가 정확히 알고 있는 경우 해당 버그가 생기는 부분에서 동영상 녹화를 한 후 본인 블로그나 유투브 같은 곳에 업로드한 후 회사에 제보한다. 하지만 해당 행동을 하기엔 번거로움이 많은 편이다.

해당 연구의 내용은 I/O System을 거치지 않고 메모리에만 데이터를 들고 있어 부하가 크지 않은 편이기에 유저가 버그를 제보하려고 할 때에 메모리에 들고 있던 데이터를 실제로 파일에 쓰는 방식이다. 그렇기에 현재 플레이 했던 내용을 유저가 제보하고 싶은 타이밍에 쉽게 해당 리플레이 파일을 첨부할 수 있어 유저가 버그를 제보하기에 도움이 된다. 각각 회사에서 사용하고 있는 이슈 트래킹 시스템에도 해당 리플레이 파일은 부담되지 않게 업로드 할 수 있게 된다.

해당 연구는 유저들끼리 공유할 단순히 작은 리플레이 용량에서 그치는 것이 아니라 해당 기능을 응용한 시스템 개발로 인해 유저들끼리의 커뮤니케이션을 만들어줌으로 인해 더 많은 유저들의 흥미를 이끌 수 있다. 또한 용량이 굉장히 작아졌기 때문에 해당내용을 항상 어느 정도 메모리에 저장해 놓고 유

저가 버그 제보를 할 때에 메모리에 저장되어 있던 내용을 같이 자동으로 전송하도록 시스템을 만들어 놓는다면 추후 버그 수정에도 굉장히 용이한 시스템을 구축할 수가 있게 된다.

또한 게임 내에서 콤보 연습 영상이나 스킬영상을 재생해주는 부분에도 사용할 수 있다. 해당 부분에 대해서는 앞으로 더 연구해 볼 가치가 있는 문제라고 본다.

참 고 문 헌

[1] 김혜영, 임영종, 실시간 전략 시뮬레이션 게임에서의 효율적인 동기화 기법, 한국게임학회 논문지 Vol.10 No.3 [2010] 83~92

[2] 방경운, 방정원, 언리얼4의 효과적인 라이팅 세팅에 관한 연구, 한국컴퓨터정보학회 학술발표논문집, Vol.26 No.1 [2018] 159~162

[3] 김수균, 강희조, 성경, 항공 응용 분야 : 언리얼 엔진을 통한 FPS 게임 개발, 韓國航行學會論文誌(The journal of Korea Navigation Institute) Vol.14 No.5 [2010] 718~724

[4] 김한호, 정형원, 유니티와 KUDAN 엔진을 활용한 MARKERLESS 방식의 증강현실 게임개발 - '우리동네히어로'의 개발사례 중심으로, 디지털융복합연구(Journal of Digital Convergence) Vol.15 No.4 [2017] 421~426

[5] 정서원, 김진모, Unity 3D를 활용한 1인 모바일 캐주얼 게임 제작, 한국디지털콘텐츠학회논문지 Vol.15 No.4 [2014] 501~512

[6] 배재환, Unity 3D 게임 엔진을 이용한 슈팅 게임 설계 및 개발, 한국컴퓨터게임학회논문지 Vol.29 No.1 [2016] 93~100

[7] 김현규, 김하균, 게임 과몰입 원인의 결정요인(게임특성, 심리적 특성, 환경특성)에 관한 연구, 예술인문사회융합멀티미디어논문지 Vol.8 No.2 [2018] 319~329

[8] 위키백과, 비디오 게임 장르, https://ko.wikipedia.org

[9] 반디 캡쳐 라이브러리 벤치마크, https://www.bandicam.co.kr

[10] 위키백과, H.264/MPEG-4 AVC, https://en.wikipedia.org

[11] H.264/SVC에서의 압축효율 향상을 위한 최적화 필터링 기법 및 디코더 구조 연구, 박민우, 중앙대학교 대학원 전자전기공학부 정보통신공학 전공 석사 학위논문 2010. 2

[12] HEVC 부호화 및 복잡도 성능 향상 기법, 이상용, 한국항공대학교 일반대학원 항공전자공학과 석사 학위논문 2012. 8

[13] HEVC 부호화기 고속화를 위한 코딩 블록 구조 결정방법의 간소화, 마종

현, 광운대학교 대학원 컴퓨터공학과 석사 학위논문 2015.8

[14] 모바일 실시간 전략 게임의 설계 및 구현, 김창효, 부산대학교 대학원 전자전기컴퓨터공학과 석사 학위논문 2015. 8

[15] 스마트폰의 플랫폼 구성에 따른 성능 비교, 박상민, 국민대학교 대학원 컴퓨터공학과 컴퓨터공학전공 석사 학위논문 2011. 2

[16] 모바일 아케이드 게임에서 게임 몬스터의 효율적인 행동 패턴, 김영배, 경북대학교 산업대학원 석사 학위논문 2007. 8

[17] 3차원 게임을 위한 물리엔진의 구현 및 성능분석, 세종대학교 대학원 컴퓨터공학과 석사 학위논문 2004. 2

[18] 정우철, 전문석, 최도현, 안드로이드 디바이스 최적화를 위한 GOF 디자인 패턴 적용 방법에 대한 연구, 한국인터넷방송통신TV학회 논문지 Vol.17 No.1 [2017] 89~97

[19] Unity3d StateMachine Basic, https://docs.unity3d.com/

[20] Unreal Engine State Machines, https://docs.unrealengine.com/

[21] OhSoft Ocam, http://ohsoft.net/

[22] LZF compression and decompression for Unity, https://forum.unity.com/threads/lzf-compression-and-decompression-for-unity.152579/