



Tmemo-2020-29

종합설계 프로젝트 수행 보고서

프로젝트명	VR 당구와 학습을 통한 가이드
팀 번호	S3-5
문서 제목	수행계획서(O) 2차발표 중간보고서(O) 3차발표 중간보고서(O) 4차발표 중간보고서(O) 최종결과보고서(O)

2020.12.04

팀원 : 정 준 영 (팀장)
김 준 기
정 호 길

지도교수 : 서대영 (인)



문서 수정 내역

작성일	대표작성자	버전(Revision)	수정내용	
2020.01.02	정준영(팀장)	1.0.0	수행계획서	최초작성
2020.01.06	김준기	1.1.0	수행계획서	서론, 본론 1차 수정
2020.01.07	김준기	1.1.1	수행계획서	서론, 본론 1차 수정 및 추가
2020.01.09	정호길	1.1.2	수행계획서	본론 2차 수정 및 추가
2020.01.10	정준영(팀장)	1.1.3	수행계획서	본론 3차 수정 및 추가
2020.01.15	정호길	1.2.0	수행계획서	시험시나리오 작성
2020.01.21	정준영(팀장)	1.2.1	수행계획서	문서 수정 및 최종 확인
2020.02.20	정호길	1.2.2	수행계획서	개발내용, 문제 및 해결방안 수정
2020.03.02	정준영(팀장)	2.0.0	수행계획서	개발 환경 변경 및 상세설계 추가
2020.05.01	정준영(팀장)	2.0.1	수행계획서	Prototype 구현 내용 추가
2020.05.27	정호길	2.1.0	수행계획서	시험/테스트 내용 추가
2020.06.14	김준기	2.1.1	수행계획서	코딩/데모 내용 추가
2020.06.25	정준영(팀장)	3.0.0	수행계획서	오타 수정 및 4차 보고서 검토
2020.08.21	정호길	3.1.0	수행계획서	가이드 설계 내용 보완
2020.10.14	정준영(팀장)	3.2.0	수행계획서	DB 설계 및 구현 내용 수정
2020.11.20	정준영(팀장)	4.0.0	수행계획서	결론부 내용 추가
2020.11.23	김준기	4.0.1	수행계획서	문단 내용 정리 및 목차 수정

문서 구성

진행단계	프로젝트 계획서 발표	중간발표1 (2월)	중간발표2 (4월)	학기말발표 (6월)	최종발표 (10월)
기본양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식	계획서 양식
포함되는 내용	I. 서론 (1~6)	I. 서론 (1~6)	I. 서론 (1~6)	I. 서론 (1~6)	I
	II. 본론 (1~3)	II. 본론 (1~4)	II. 본론 (1~5)	II. 본론 (1~7)	II
	참고자료	참고자료	참고자료	참고자료	III

이 문서는 한국산업기술대학교 컴퓨터공학부의
 “종합설계” 교과목에서 프로젝트
 “VR 당구와 학습을 통한 가이드” 을 수행하는
 (예시: S3-5, 정준영, 김준기, 정호길)들이 작성한 것으로
 사용하기 위해서는 팀원들의 허락이 필요합니다.

목 차

I . 서론

1. 작품선정 배경 및 필요성	4p
2. 기존 연구/기술동향 분석	4p
3. 개발 목표	6p
4. 팀 역할 분담	6p
5. 개발 일정	6p
6. 개발 환경	7p

II . 본론

1. 개발 내용	8p
2. 문제 및 해결방안	9p
3. 시험시나리오	9p
4. 상세 설계	11p
5. Prototype 구현	19p
6. 시험/ 테스트 결과	22p
7. Coding & DEMO	25p

III . 결론

1. 연구 결과	36p
2. 작품제작 소요재료 목록	38p

참고자료	38p
------------	-----

I . 서론

1. 작품선정 배경 및 필요성

당구는 2018년 기준 전국 당구장 수가 PC방 수를 넘을 만큼 인기가 있는 스포츠이다. 당구를 단지 어려운 스포츠라는 관념 때문에 시작에 어려움을 겪는 사람들이 많다. 또한, 당구대, 큐대 등의 물리적 제약사항이 존재하여 일상에서 즐기기에 까다로운 점이 존재한다.

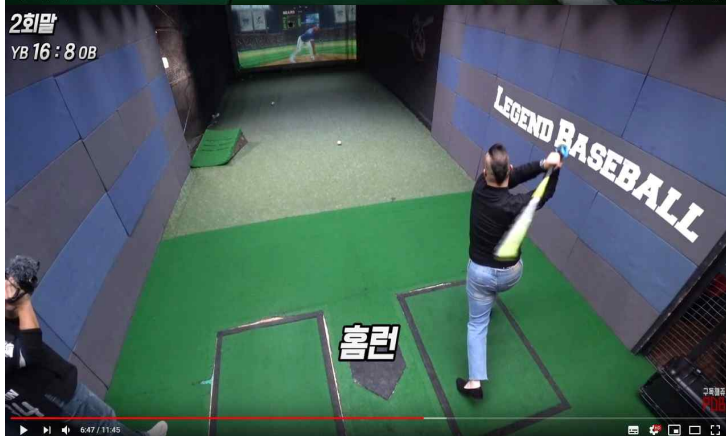
2. 기존 연구/기술동향 분석

- 스크린 스포츠(Sports)의 인기 배경

현재 많은 사람이 이용하는 스크린 스포츠로 대표적인 것이 야구, 골프이다. 스크린 스포츠가 인기 있게 된 배경에는 바쁜 업무 중 주변에 몇 없는 야구장, 골프장까지 가기에는 시간이 부족했었던 점, 장소는 예약하고 가더라도 넓은 장소에 비해 수용할 수 있는 인원수가 적은 점, 스크린 스포츠가 실제 스포츠보다 비용이 저렴했던 점 등등이 있다.



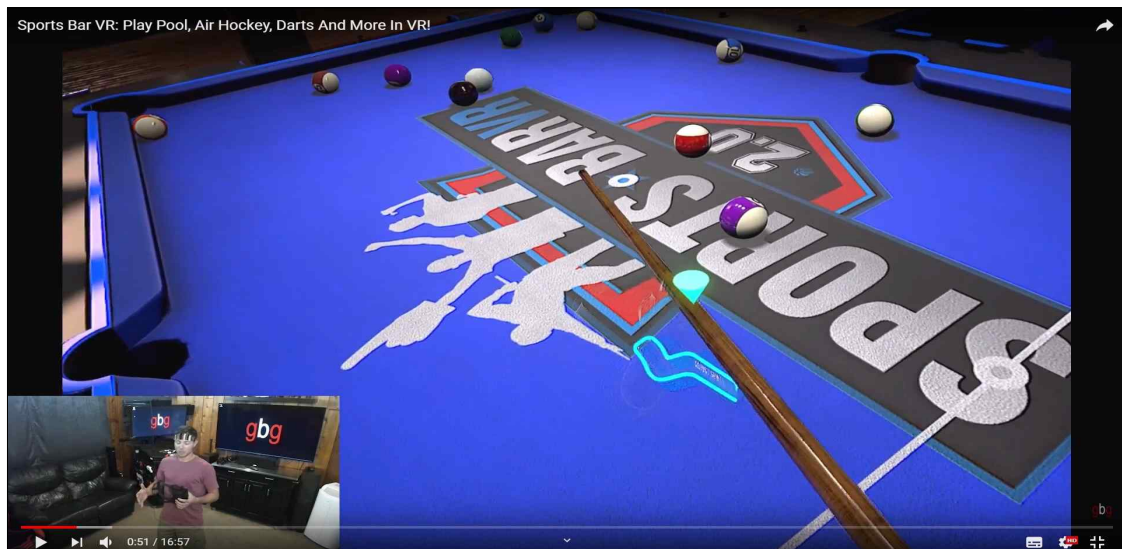
- 스크린 골프 -



- 스크린 야구 -

- VR 스포츠(Sports)의 등장

스크린 스포츠의 장점으로 쉽게 접근이 가능한 점, 비용이 저렴한 점, 공간의 제약이 적어진 점에서 더 나아가 이제는 VR 기기만 있다면 스포츠 종목에 따라 장비도 바꿀 필요가 없어졌다. 비용 측면에서도 게임과 기기를 한 번만 구매하면 추가적인 비용이 발생하지 않는다.



이름	특징
Pool Nation VR	<ul style="list-style-type: none"> - 멀티 플레이 - Pool(포켓볼) 이외의 볼링, 다트 게임이 가능 - 컨트롤러에서 여러 기능을 제공함(Teleport, Lock Shot) - 다음 타격할 공(Solids)을 UI를 통해 시각적으로 알려줌



이름	특징
Vr Super Sports	<ul style="list-style-type: none"> - 싱글 플레이 - 야구, 사격, 농구, 양궁 등등이 가능 - 야구는 공격만, 축구는 골키퍼만 플레이

3. 개발 목표

VR 기기를 이용하여 사용자 움직임을 인식하고 컨트롤러를 이용하여 가상의 당구공을 타격할 수 있으며, VR 환경 내 당구공의 움직임 구현, 리플레이 기능 및 성공 방법을 사용자에게 제공할 수 있는 소프트웨어를 개발하여 당구에 대한 어려움 및 공간에 대한 제약 해결을 목표로 한다.

4. 팀 역할 분담

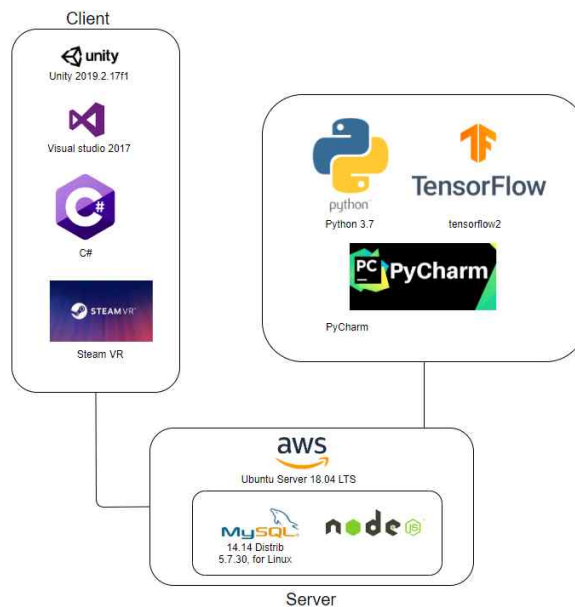
이름	김준기	정준영	정호길
자료수집	<ul style="list-style-type: none"> - 당구 물리학 논문 자료 수집 - Unity 사용과 C#스크립트 작성방법 조사 	<ul style="list-style-type: none"> - Unity 사용과 C#스크립트 작성방법 조사 - Unity와 AWS 연동 방법 조사 	<ul style="list-style-type: none"> - 당구 물리학 논문 자료 수집 - Tensorflow 기능과 학습 데이터 수집
설계	<ul style="list-style-type: none"> - 공 움직임 설계 - 게임 UI 설계 	<ul style="list-style-type: none"> - DB table 설계 - Unity와 DB연동 설계 - 리플레이 기능 설계 	<ul style="list-style-type: none"> - 머신러닝 학습 모델 설계 - Unity와 학습 모델 연동 설계
구현	<ul style="list-style-type: none"> - VR환경 및 공 움직임 구현 	<ul style="list-style-type: none"> - DB, 리플레이 기능 구현 	<ul style="list-style-type: none"> - 머신러닝 구현 - 가이드 기능 구현
테스트	<ul style="list-style-type: none"> - 게임 플레이 작동 테스트 - 가이드 기능 작동 테스트 - 리플레이 기능 작동 테스트 - 통합테스트/ 유지보수 		

5. 개발 일정

항목	추진사항	12	1	2	3	4	5	6	7	8	9	10
요구사항 정의 및 분석	<ul style="list-style-type: none"> - 요구사항 정의 및 분석 - 요구사항 명세서 											
시스템 설계 및 상세 설계	<ul style="list-style-type: none"> - 시스템 설계 - 상세 설계 											
구현	<ul style="list-style-type: none"> - VR 환경 및 공 움직임 구현 											

	- DB, 리플레이 기능 구현											
	- 머신러닝 구현											
시험 및 데모	- 테스트											
	- 딥러닝 학습효과 시험 및 조정											
	- 시스템 통합 시험											
문서화 및 발표	- 졸업작품 중간 보고서 작성(중간보고서, 사용자 매뉴얼 작성)											
	- 발표(전시회, 정보과학회, 산업기술대전)											
산업기술대 전	- 산업기술대전 참가											
졸업작품 최종 보고서 작성 및 패키징	- 졸업작품 최종 보고서 작성 - CD 패키징											

6. 개발환경



	구분	항목	적용 내역
S/W 개발환경	앱 개발	Unity Personal (2019.2.17.f1)	앱 개발을 위한 엔진
		Visual Studio community (2019)	앱 개발을 위한 툴
		Windows 10	개발 PC 운영체제
		C#	앱 개발을 위한 언어
	기능 개발	PyCharm (2019.2.1)	기능 개발을 위한 툴
		Python (3.6)	기능 개발을 위한 언어
		TensorFlow (2)	딥러닝 개발을 위한 라이브러리
H/W 개발환경	서버 개발	AWS EC2(Ubuntu 18.04 LTS)/ NodeJS	리플레이 저장을 위한 데이터베이스
		HTC VIVE	프로그램을 사용할 수 있는 장비
	VR	VIVE Controller	사용자의 조작을 입력할 수 있는 장비

II. 본론

1. 개발 내용

- 게임 플레이

게임 플레이시의 물리법칙은 사용자의 큐를 트리거로 설정하고 Raycast를 사용해 공의 타격을 체크한다. 또한 움직이는 공에서 Raycast를 통해 공과 쿠션의 Collider를 검출하여 충돌한 오브젝트의 정보를 얻어와 충돌한 물체와의 물리법칙을 스크립트에 수식으로 구현한다. 공 간의 충돌을 체크하여 3쿠션의 성공 여부를 결정한다. 공의 움직임은 공의 상태를 처리하는 스크립트를 통해 공의 상태를 매 프레임마다 업데이트시켜 움직임을 구현한다.

VR기기의 연동은 SteamVR API를 이용해서 사용자의 움직임을 입력받고 VR 컨트롤러의 버튼에 대한 기능을 정의한다.

- 가이드 기능

사용자는 게임 도중 버튼을 이용하여 가이드를 요청할 수 있다. 가이드 요청 시 현재 공의 위치 정보를 이용하여 학습 모델에 필요한 이미지를 생성하게 된다. 이는 서버의 학습모델에게 전달되어 학습된 결과가 도출되게 된다. 학습모델의 결과는 다시 클라이언트에게 전달되어 타법을 알 수 있게 된다.

- 리플레이 기능

사용자는 자신의 당구 실력 향상을 위해 리플레이 기능을 사용할 수 있다. 이를 구현하기 위해 필요한 DB데이터(공의 좌표, 공의 각도 등등)를 저장 및 관리하고 통신하기 위한 MySQL 서버 구축 및 테이블을 생성한다.

2. 문제 및 해결방안

Unity의 컴포넌트인 Rigidbody로 구현한 쿠션 충돌 및 공끼리의 충돌 처리는 불안정하고 공의 움직임이 실제 당구공의 움직임과는 큰 차이를 보였다. 충돌 처리를 Rigidbody와 Collider로의 체크가 아닌 트리거 역할로 전환하고 Raycast 방식을 이용해 충돌 유무의 확인과 충돌 처리를 스크립트로 작성하고자 한다.

학습 이미지 전처리 방안으로 OpenCV 함수 중 이미지에서 원을 찾는 함수를 알게 되어 예상 입력 이미지들에 적용해보게 되었다. 몇몇 이미지에서는 공을 찾았으나 글씨나 공이 가려져 있거나 하는 경우의 이미지가 많아 또 다른 전처리 작업이 필요하게 되었다. 많은 데이터가 필요하기에 다양한 이미지의 전처리가 가능한 프로그램을 만들고 있으나 입력 데이터의 규격이 너무 달라 자동화에 어려움을 겪었다. 해결방안으로 모든 이미지가 가능한 프로그램은 불가능하다고 판단하여 최대한 공이 정확히 나온 이미지를 가져와 경계를 뚜렷이 하여 원을 찾는 방법으로 바꾸게 되었다. 몇몇의 케이스를 제외하곤 당구공을 찾게 되었다.

AWS EC2 서버 모델 선택 시 가장 최근에 릴리즈된 모델을 선택하였으나, 개발 시 처음 다루어 보는 환경 때문에 서버 구축 및 DB설계에 어려움을 겪었다. 이를 해결하기 위해 다른 모델들 중 익숙하였던 Ubuntu Server 18.04 LTS를 선택하였다.

3. 시험시나리오

1. 사용자는 VR기기를 착용하여 사용자 인터페이스를 확인할 수 있다. 컨트롤러를 조작하여 메뉴 진입 등을 수행할 수 있다. 헤드 트래킹을 지원하여 사용자의 시선 방향에 따른 UI 변화를 확인할 수 있다.

2. 당구 플레이 시 사용자는 당구대와 초기 공의 위치를 확인하게 된다. 공의 타격은 컨트롤러를 이용하여 가능하게 된다. 한 손의 컨트롤러를 큐대의 손잡이와 같이 움직이게 된다. 이때, 반대 손으로는 큐대의 브릿지 역할을 버튼을 눌러 수행할

수 있다. 버튼을 누르면 큐대로 가르킨 타점의 상하좌우 움직임을 고정하여 사용자가 원하는 타점을 타격할 수 있다. 사용자는 타격한 공의 움직임을 확인하며 볼 수 있게 된다. 움직인 공에 따라 이동을 하여야 할 때는 신체를 움직여 이동할 수 있다. 하지만, 당구대의 반대편으로 넘어가야 하는 상황과 같이 실제 움직임의 길이가 길 경우 공간상의 제약을 해결하기 위해 컨트롤러의 조작으로 원하는 지점으로 텔레포트가 가능하다.

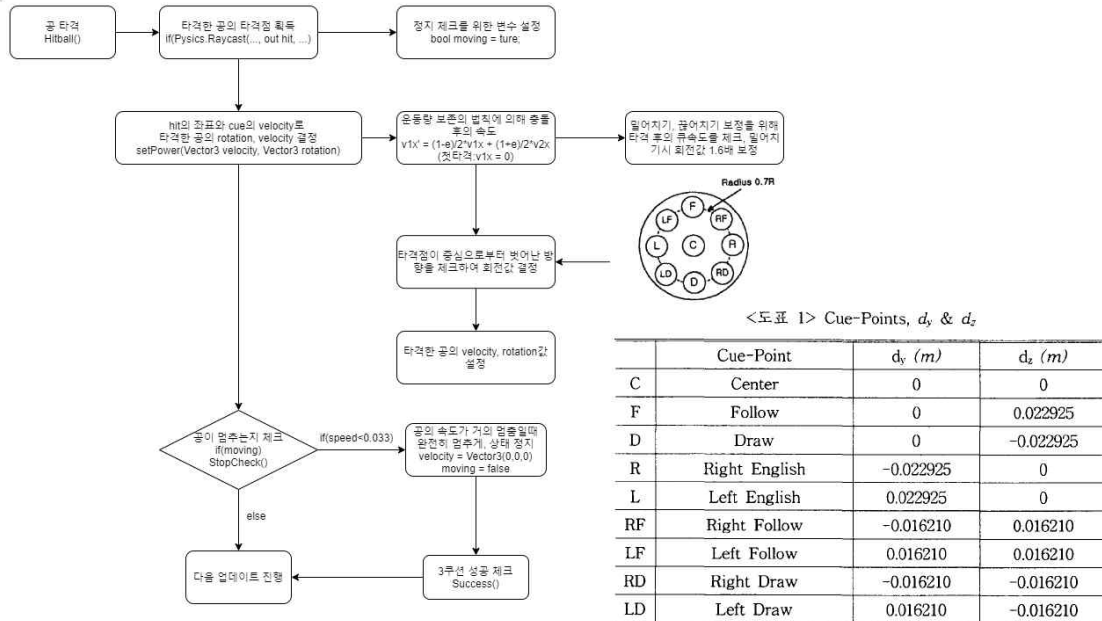
3. 사용자는 본인의 친 기록을 다시 확인할 수 있다. 서버에 저장된 본인이 쳤던 기록을 다양한 시점으로 확인할 수 있다. 사용자의 플레이 기록은 해당 타격을 했을 시 움직였던 공의 위치, 공의 각도, 타격 힘, 타격 점, 시각을 서버에 전달하여 저장된다. 이는 사용자 요청 시 제공된다.

4. 사용자는 타법을 알려주는 가이드 기능을 사용할 수 있다. 서버에 있는 학습 모델을 통하여 현재 공의 위치에 따른 타법을 알 수 있게 된다. 현재 공의 위치를 통해 만들어지게 되는 데이터가 학습 모델에게 입력되며 이에 따른 결과가 사용자에게 제공된다.

4. 상세 설계

4.1 공의 움직임 설계(타격, 움직임, 충돌)

4.1.1 공의 타격 기능 설계



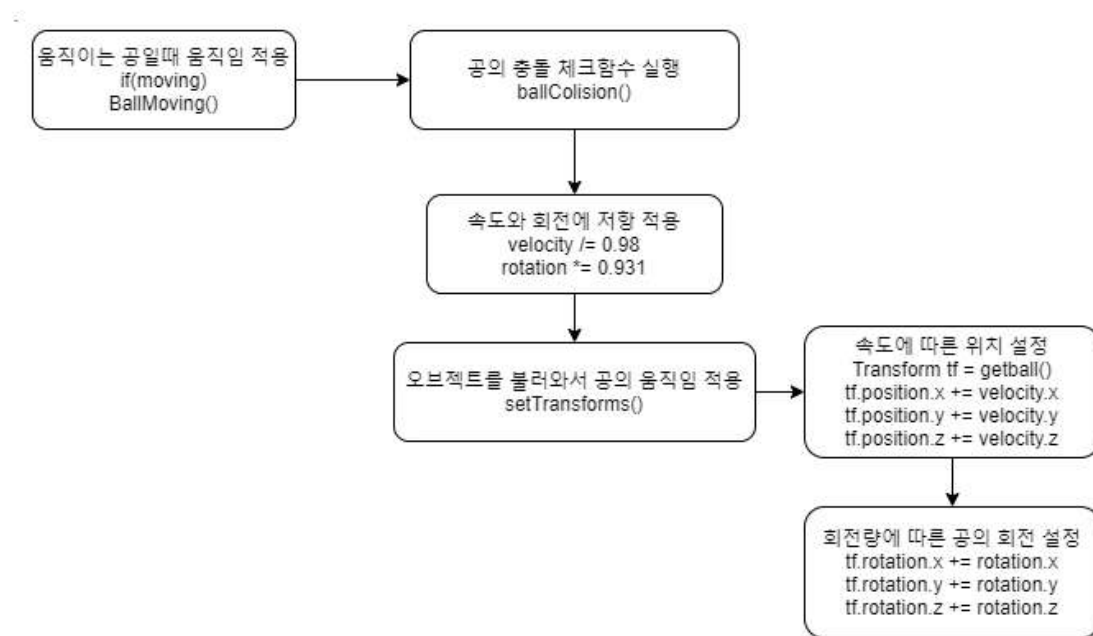
Hitball()함수를 통해 raycast로 타격한 공의 타격점을 hit라는 변수로 가져온다. 타격을 하였으므로 타격한 공의 움직임 상태를 true로 설정한다.

타격시의 플레이어 큐의 velocity 값을 운동량 보존법칙에 의거해 공의 velocity값을 설정한다. 이때, 타격 후 큐의 속도를 체크해서 밀어치기시에는 공의 회전 벡터의 보정값을 1.6배로 설정한다.

타격점이 공의 중심에서 어느 방향으로 얼마나 떨어져 있는지를 체크하여 도표의 값에 따라 회전 벡터값을 설정한다.

공이 멈추었는지를 판단하기 위한 함수 StopChecking()을 호출하여 정지상태에 가까운지를 확인한다. 해당하는 경우에 공의 상태를 완전한 정지상태로 바꾸어주고 타격에 대한 3쿠션의 성공 여부를 체크한다.

4.1.2 공 움직임 설계



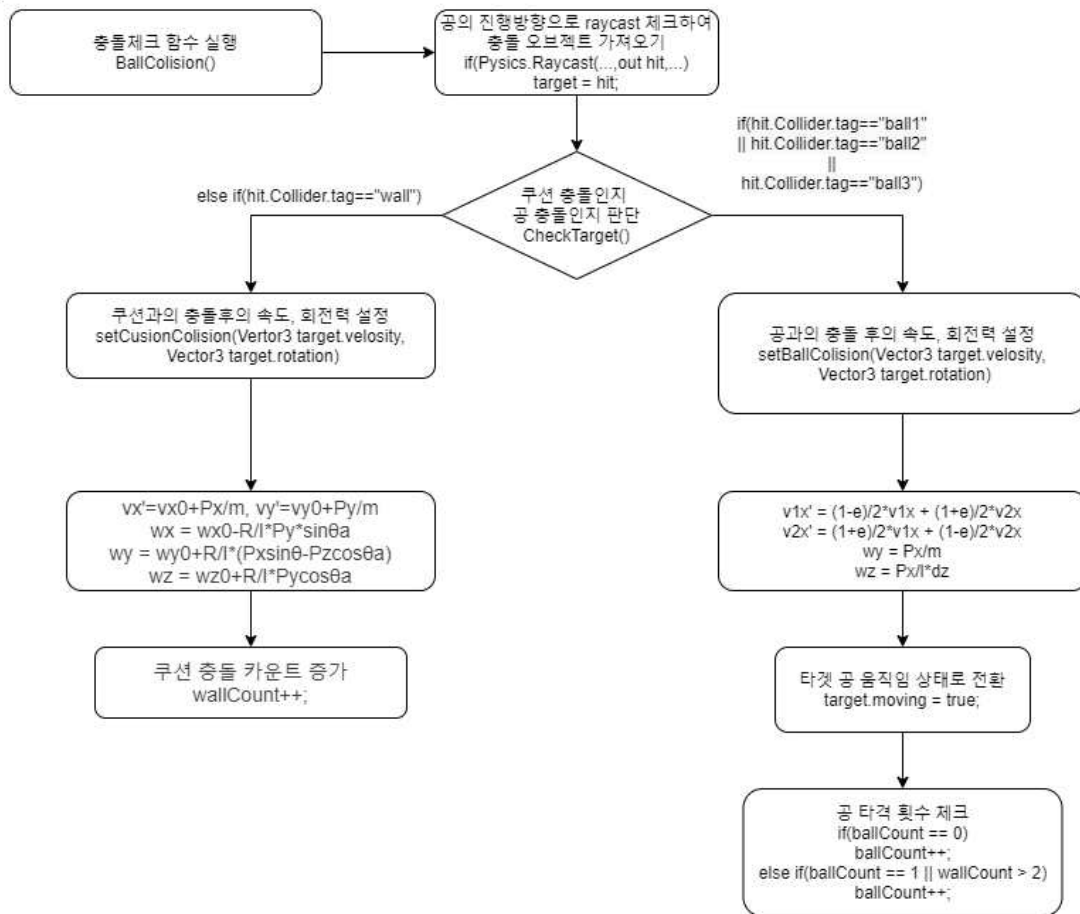
공이 움직이는 상태일 때 공에 적용되는 실질적인 물리법칙이 적용되는 함수이다. 속도와 회전에 대한 저항을 계산한다.

공의 움직임을 체크하면서 공이 움직이는 상태일 경우 BallMoving()함수를 실행시킨다.

처음으로는 공의 충돌 체크함수를 실행시켜 공이 충돌이 이루어지는지를 체크한다. 공의 속도벡터와 회전벡터에 저항값을 적용시킨다. 이 저항값은 참고한 공의 역학에 대한 논문의 <도표2>를 기반으로 설정하였고, 시뮬레이션을 통해 값의 조정이 필요하다. [1]

움직이는 상태인 오브젝트의 Transform 컴포넌트를 가져와서 설정한 속도벡터와 회전벡터를 적용시킨다. 이 기능은 Update()안에서 작동해서 매 프레임마다 실행되어 속도와 회전값에 따라 공의 상태를 변경시키는 방법으로 물리법칙을 적용하였다.

4.1.3 공 충돌 설계



공이 움직이는 상태일 때 실행되는 BallMoving()함수 안에서 사용되며 공이 충돌하는 오브젝트에 대한 정보를 알아내고, 충돌이 확인될 경우 충돌 후 공의 속도와 회전값을 설정하는 기능을 수행한다.

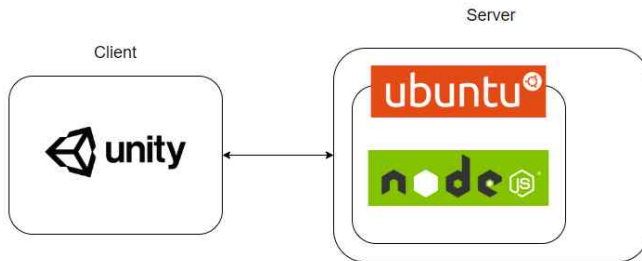
BallCollision()함수를 실행시키면 공의 진행방향을 향해 공의 velocity.magnitude 만큼의 길이의 raycast를 실행하여 충돌하는 물체를 감지하고 감지된 오브젝트를 가져온다.

오브젝트가 감지될 경우 감지된 오브젝트의 태그를 확인하여 cushion인지, ball인지를 판단한다.

cusion일 경우 쿠션과 충돌한 후의 속도와 회전값을 계산하여 적용한 후 쿠션 충돌 카운트 wallCount를 증가시킨다.

ball일 경우 공과 충돌한 후의 속도와 회전값을 계산하여 적용한 후 충돌한 타겟의 공의 움직임 상태를 true로 설정하고 3쿠션 성공 여부를 확인하기 위해 공의 타격 횟수를 쿠션과의 충돌횟수와 연계하여 체크하고 증가시킨다.

4.2 데이터 베이스(DB) 설계



4.2.1 DBMS 선정

대표적인 DBMS인 Oracle, Mysql, Microsoft SQL Server 중 Mysql를 선택했다. 라인센스가 무료이고, 기존에 학습했던 경험, 서버로 이용할 운영체제를 지원하는 점을 이유로 삼았다.

4.2.2 통신

Unity내에서 AWS EC2의 Nodejs 서버와 ScketIO을 이용해 통신이 가능하다. ScketIO 는 연결과 기본적인 프로토콜이 구현되어 있기 때문에 개발자는 컨텐츠를 주고 받는데만 집중할 수 있다는 장점이 있으며 JavaScript를 지원한다.

Unity의 C#스크립트와 JavaScript로 작성해둔 이벤트에 따라 DB를 제어하여 테이블을 추가, 삭제 가능하고 Replay 기록을 삽입하여 유저의 플레이 기록을 저장 및 로드 할 수 있다.

4.2.3 테이블 설계

테이블명	replayTable																				
목적	전체 replay 목록																				
테이블 구성	<div><div>Columns in table</div><table><tr><th>Column</th><th>Type</th><th>Nullable</th><th>Indexes</th></tr><tr><td>id</td><td>int(11)</td><td>NO</td><td>PRIMARY</td></tr><tr><td>title</td><td>varchar(30)</td><td>NO</td><td></td></tr><tr><td>user_id</td><td>varchar(30)</td><td>NO</td><td></td></tr><tr><td>save_time</td><td>varchar(30)</td><td>NO</td><td></td></tr></table></div>	Column	Type	Nullable	Indexes	id	int(11)	NO	PRIMARY	title	varchar(30)	NO		user_id	varchar(30)	NO		save_time	varchar(30)	NO	
Column	Type	Nullable	Indexes																		
id	int(11)	NO	PRIMARY																		
title	varchar(30)	NO																			
user_id	varchar(30)	NO																			
save_time	varchar(30)	NO																			

테이블명	replay								
목적	각 Replay별 공과 큐의 좌표 및 각도 값 저장 (리플레이 기록 저장)								
테이블 구성	Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI
	id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	ball1pos_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball1pos_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball1pos_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball1rot_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball1rot_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball1rot_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2pos_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2pos_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2pos_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2rot_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2rot_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball2rot_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball3pos_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball3pos_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	ball3rot_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	quepos_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	quepos_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	quepos_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	querot_x	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	querot_y	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	querot_z	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	deltatime	FLOAT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

4.2.4 통신 시기

-테이블 생성

클라이언트(VR Application)에서 Replay 버튼 클릭 시 새로운 리플레이 테이블이 생성된다.

생성한 테이블의 이름은 replay1, . . . , replay n 오름차순으로 자동생성 된다.

-Replay 저장

Unity Update 함수를 이용하여 맵프레임 마다 Replay 기록인 공의 좌표, 공의 각도, 큐의 좌표, 큐의 각도를 NodeJs 서버로 문자열 형태로 보내어 서버에서는 Replay기록을 미리 sql로 작성된 insert문에 각 Parameter로 사용하게 된다.

ex) SQL= 'insert into replay명 values(ball1pos_x, ball1pos_y, ...).'

공의 좌표, 공의 각도, 큐의 좌표, 큐의 각도는 과거의 타격을 재구현 하는데 이용된다.

-Replay 불러오기

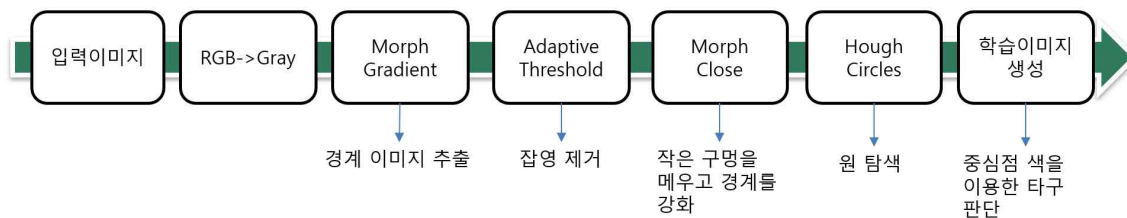
클라이언트(VR Application)에서 리플레이 선택 시 Unity에서 NodeJs서버로 리플레이명과 함께 이벤트 발생을 알린다. 해당 이벤트와 연결된 메소드의 sql인 “select * from replay명”을 이용하여 그 결과를 Unity로 다시 전송한다. Unity는 서버로부터 받은 Replay에 대한 정보를 GameObject인 당구공 3개와 큐의 좌표, 각도 값을 id의 순서대로 해당 값을 변경한다.

4.3 가이드 설계

4.3.1 학습이미지 생성 프로그램 설계

텐서플로우를 이용한 CNN 기법으로 공의 위치에 따른 타법 추천 모델을 제작하기 위하여 학습 이미지 생성에 필요한 파이썬 프로그램을 설계하였다. 해당 프로그램은 다양한 규격의 이미지에서 불필요한 데이터인 글씨나 배경을 없애고 필요한 데이터인 공의 위치를 표시한 이미지로의 변환을 수행하여야 한다. 따라서 이미지에서 공의 위치를 파악하기 위한 방안을 조사하여 예상 입력 이미지들에 적용하며 프로그램을 제작해보게 되었다.

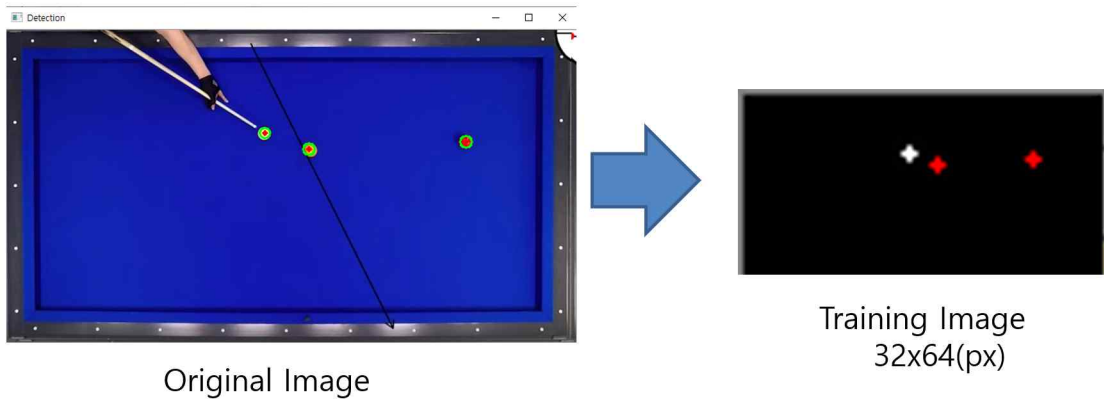
프로그램은 다음과 같이 동작하여 학습이미지를 생성한다.



학습 이미지 생성 순서

입력 이미지에서 경계를 찾기 위하여 RGB 컬러 이미지를 Gray 컬러 이미지로 변환한 후 Morph Gradient를 이용해 경계 이미지를 추출한다. 경계 이미지에서 잡영을 제거하기 위해 Adaptive Threshold를 적용한다. 그 후, 작은 구멍을 메우고 경계를 강화하기 위하여 Morph Close를 사용한 결과 이미지에서 원을 찾는 opencv 함수인 HoughCircles를 이용하여 원을 찾는다. 타구가 흰공인 경우와 주황공인 경우가 있기 때문에 이를 구분하게 되었다. 각 원의 중심점의 색을 추출하여 각 경우를 판단하였고, 변수를 이용해 각각의 경우에 맞게 학습 이미지에 타구를 흰색으로 다른 공을 빨간색으로 저장한다.[2]

프로그램의 수행 결과인 학습 이미지는 테이블의 장축, 단축 비율에 맞으며, 테이블의 단축인 1422mm와 실제 공 지름인 61.5mm의 비율을 토대로 공의 반지름은 1px이며, 전체 이미지의 크기는 64px X 32px의 크기로 선정하였다.

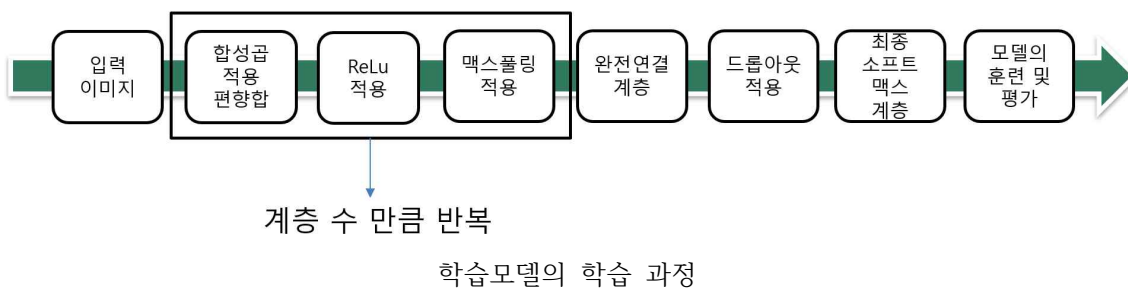


원본이미지의 학습이미지 변환

타법을 다음의 7개의 클래스로 분류하였다. 1. 뒤돌리기, 2. 앞돌리기, 3. 옆돌리기, 4. 빗겨치기, 5. 대회전, 6. 더블레일, 7. 빈쿠션. 앞의 분류에 따라 이미지에 라벨링을 하는 파이썬 프로그램을 이용하여 학습 이미지를 생성하게 된다.

4.3.2 학습모델 설계

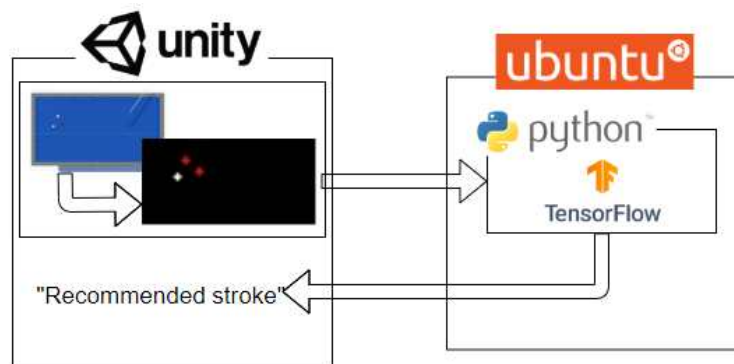
만들어진 데이터셋이 아닌 본인의 이미지로 데이터셋을 구성하여 이를 이용한 CNN 모델의 예시를 학습하였다. 영상 분류에 높은 성능을 보이고 있는 CNN 모델을 구성하게 되었다. 학습 모델의 구성은 영상 처리에 주로 사용되는 Conv2D 클래스를 사용한 컨볼루션 레이어, 컨볼루션 레이어의 출력에서 주요값만을 추출한 맥스 풀링 레이어, 오버피팅을 방지하기 위한 드롭아웃 레이어가 있다. 활성화함수는 간단하여 많이 사용되는 ReLu를 사용하도록 설계하였다. [3]



학습을 위한 계산 그래프를 구성하기 위해, 입력될 이미지와 각각의 출력 클래스에 해당하는 노드를 생성한다. 즉, 32x64의 크기를 가진 이미지를 한 줄로 펼친 크기인 2048과 8개의 타법 클래스에 해당하는 노드이다. 합성곱의 적용과 편향값의 합을 통하여 구성되는 합성곱 계층으로 이루어진 CNN 모델을 구성하기 위해서 많은 수의 가중치와 편향을 생성하는 함수를 이용한다. 이미지와 합성곱을 적용하고 편향을 더한뒤 ReLu 함수를 적용한다. 출력값을 구하기 위해 맥스풀링을 적용한다.

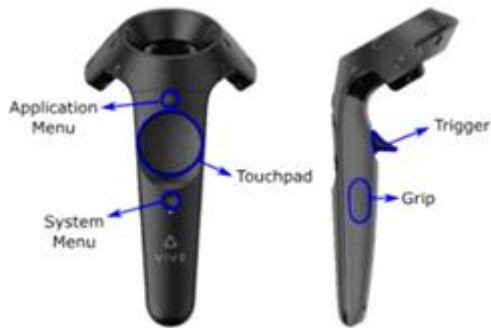
앞의 과정과 같으나 입,출력 채널 수를 다르게하여 다음 계층을 생성하게 된다. 마지막 소프트맥스 계층에 적용하기 위하여 완전연결 레이어를 추가한다. 이전 컨볼루션 레이어의 결과 텐서를 reshape하여 LeRu 활성화 함수에 전달한다. 드롭아웃 되지 않을 확률 값을 저장할 노드를 만들고 레이어를 추가한다. 마지막으로 소프트맥스 레이어를 추가한다. 크로스엔트로피와 최적화 알고리즘, 평가를 위한 연산을 정의하고 세션을 시작한 후 학습을 반복 수행해 학습모델을 생성하게 된다. 생성된 모델에 학습 이미지와 같은 이미지를 전달하여 결과를 확인할 수 있게 된다.

생성된 학습모델은 AWS에서 실행중인 ubuntu에 적재되어 unity 프로그램으로부터 필요 데이터를 수신하여 파이썬 프로그램을 이용해 실행된다. 해당 파이썬 프로그램은 결과를 다시 유니티 프로그램으로 송신하게 된다. 필요 데이터는 학습 이미지와 동일한 32x64 데이터이며, unity 프로그램에서 공의 좌표에 따른 해당 데이터를 생성한다.



타법 추천 서버 통신 과정

4.4 사용자 입력 및 상호작용 설계



좌, 우 컨트롤러가 존재한다. 사용 가능한 상호작용으로는 터치패드의 터치 액션과 방위별 클릭 액션, 트리거의 클릭 액션, 메뉴 버튼 액션, 그립 액션이 존재한다.

우측 컨트롤러는 터치패드의 아래를 클릭 시 On/Off로 큐를 잡을 수 있다. 터치패드 좌/우를 클릭 시 플레이어의 시야를 회전할 수 있다.

좌측 컨트롤러는 터치패드를 클릭 시 입력된 방향으로 플레이어가 이동한다. 큐를 든 상태에서 트리거를 클릭 시 왼손에 큐를 잡고 조준 상태가 된다.

좌, 우 동일하게 메뉴 버튼을 통해 메뉴에 접근할 수 있다. 메뉴는 왼손에서 출력되며 오른손 컨트롤러에서 RayCast를 통해 상호작용할 수 있다.

5. Prototype 구현

5.1 메인 화면

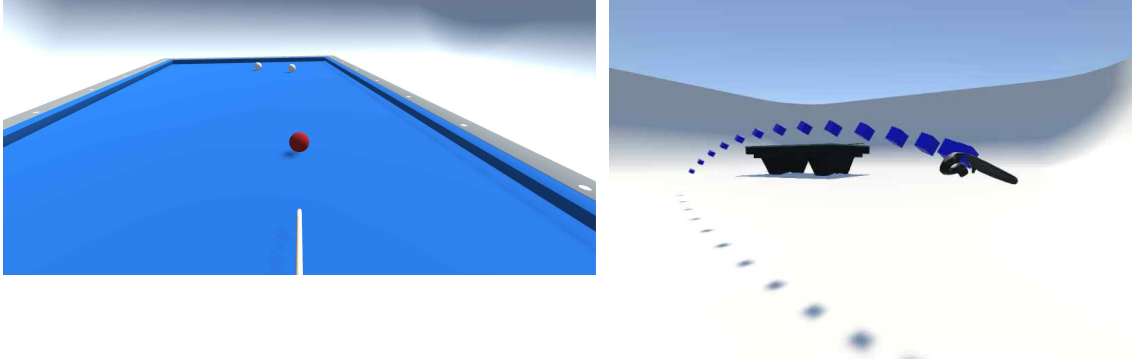


게임 시작 화면으로서 VR 컨트롤러의 Ray를 이용해 버튼을 선택할 수 있다.

버튼 선택은 콜리전 이벤트와 트리거 버튼 선택으로 가능하다.

현재 버튼은 1.게임 시작 2. 리플레이(다시보기) 3.게임 종료가 있다.

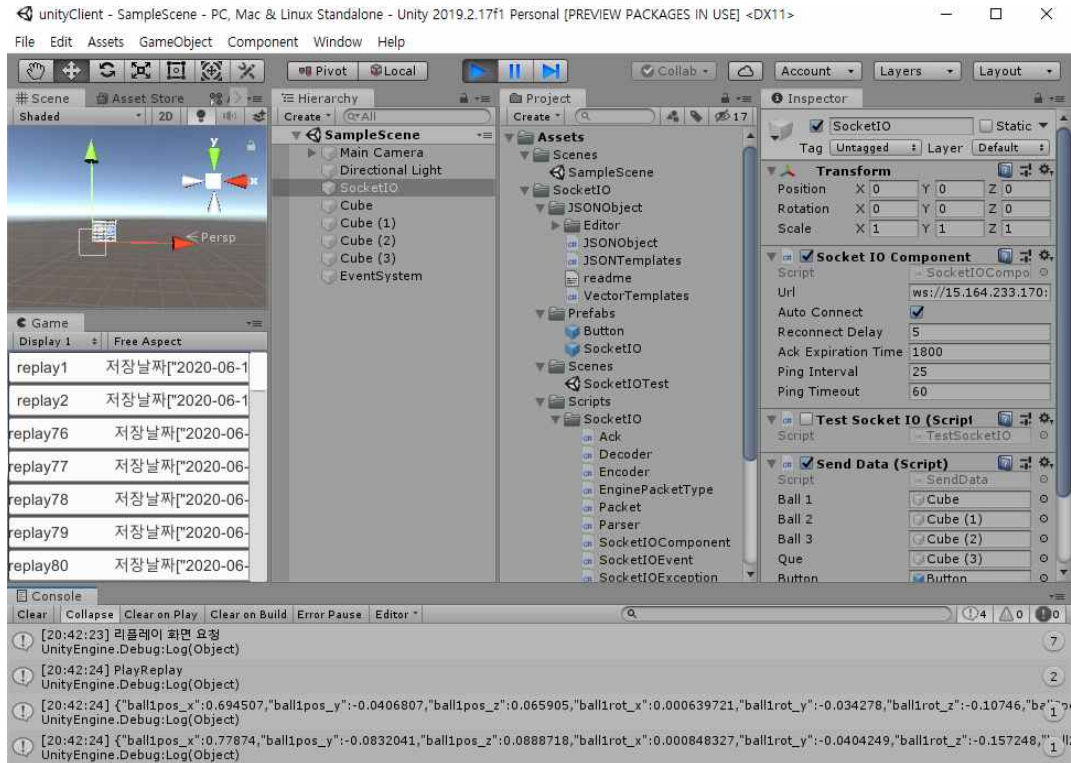
5.2 게임 모드



현재 관련 논문 공식을 통해 타격과 공 움직임을 구현하였다. 컨트롤러를 이용하여 큐를 움직이고, 공의 콜라이더와 큐의 콜라이더 충돌 체크를 통해 타격을 할 수 있다. 실제 당구와의 유사도를 높이기 위해 공과의 충돌 및 쿠션 충돌은 현재 구현 중이다. 또한, 카메라 전환을 통하여 시점을 변경할 수 있다. 시점은 플레이어 시점과 당구대를 위에서 바라본 시점이 있다.

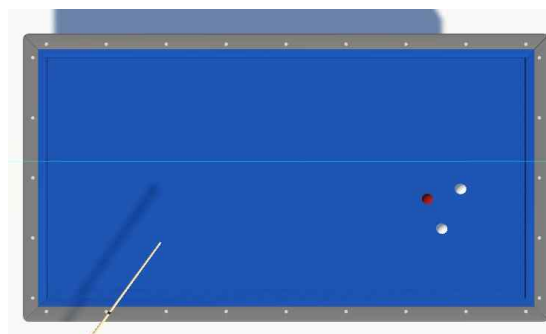
컨트롤러의 터치패드를 통하여 사용자의 위치를 이동시킬 수 있다.(텔레포트)

5.3 리플레이 구현

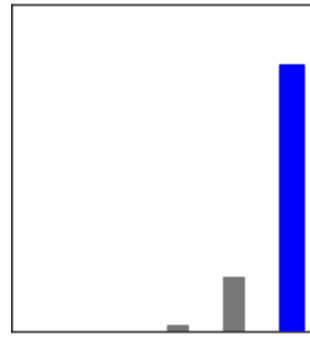
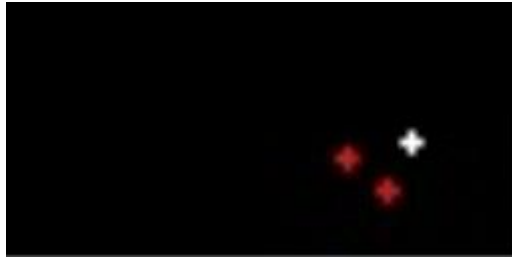


현재 AWS서버의 DB에 존재하는 테이블에 플레이한 정보를 삽입하고 게임에서 불러오는 것까지를 구현했다. [4] 게임에서 불러온 정보를 게임에서 다시 과거의 타격을 재현하는 것은 아직 구현중에 있다.

5.4 가이드 구현



왼쪽 이미지는 현재 게임의 공의 위치를 학습 이미지로 변환한 것이고, 오른쪽 그래프는 이를 학습 모델에게 주었을 때 나온 결과를 그래프로 변환하여 나타낸 것이다.

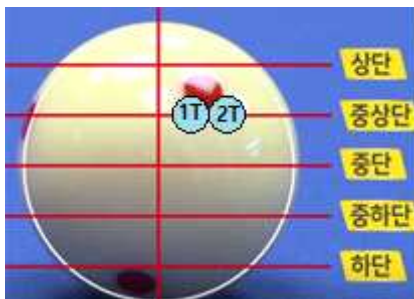


그래프의 x축은 타법, 즉 클래스이고, y축은 예측도이다. 현재 예시는 데이터양의 부족으로 정확한 예측 결과가 아니며, 수를 늘려 예측 결과의 정확도를 높이는 중이다. 또한, 이 결과를 게임 화면에서도 확인할 수 있도록 구현 중이다.


6. 시험/테스트 결과

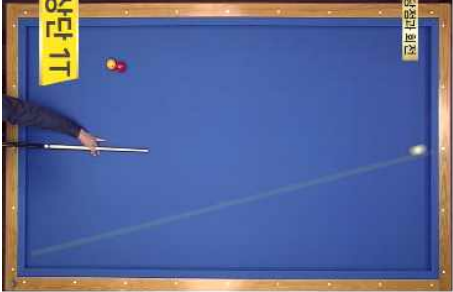
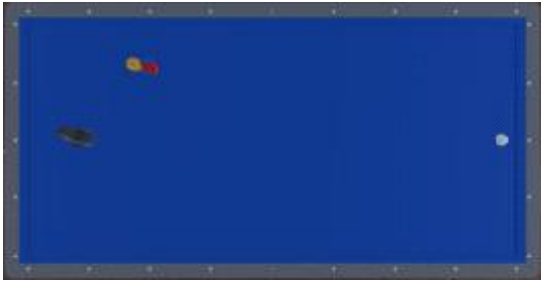
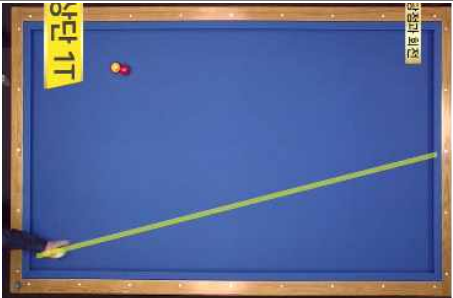
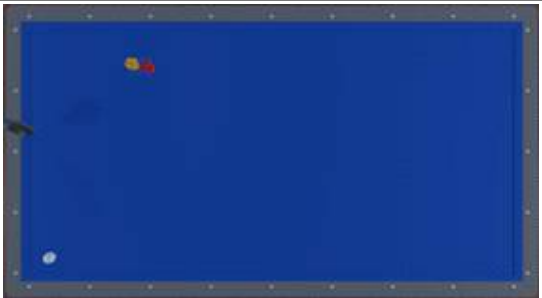

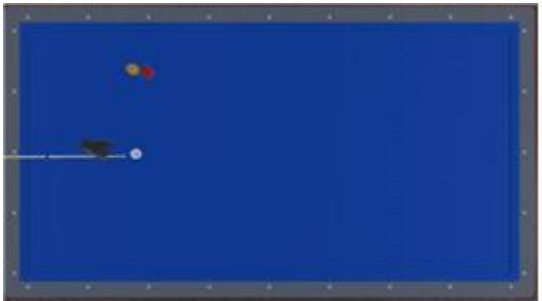

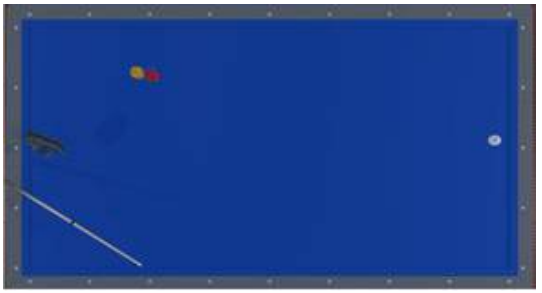
6.1 공의 회전에 따른 움직임 시험 및 결과

공의 중상단 1Tip과 2Tip은 다음 사진과 같다.



다음은 공의 중상단 1Tip, 2Tip을 타격하여 공이 회전 운동을 하며 직선으로 쿠션에 충돌했을 때 공의 움직임과 실제 타격 시의 유사도를 실험한 사진이다. 타격 후 처음, 중간, 마지막 순서이다.

실제 중상단 1Tip 타격	게임에서의 중상단 1Tip 타격
처음	
	

중간	
	
마지막	
	
실제 2Tip 타격	게임에서의 2Tip 타격
처음	
	
중간	
	
마지막	

7. Coding & DEMO

7.1 당구 물리

7.1.1 공 움직임에 대한 물리

```
private void FixedUpdate()
{
    Friction();    //저항
    CheckStop();  //멈춤상태 보정
}
```

물리 계산을 위해 FixedUpdate()에서 저항에 대한 계산과 공의 완전한 정지를 보정하고 판단하기 위한 함수를 사용한다.

```
private void Friction()
{
    //속도 저항
    if (rb.velocity.magnitude < 0.1)
    {
        rb.velocity *= 0.979f;
        rb.angularVelocity *= 0.9964f;
    }
    else if (rb.velocity.magnitude < 3)
    {
        rb.velocity *= 0.9964f;
        rb.angularVelocity *= 0.9964f;
    }
    else if (rb.velocity.magnitude < 10)
    {
        rb.velocity *= 0.9974f;
        rb.angularVelocity *= 0.9974f;
    }
    else if (rb.velocity.magnitude < 15)
    {
        rb.velocity *= 0.9984f;
        rb.angularVelocity *= 0.9984f;
    }
}
```

공의 움직임에 대한 저항함수 부분이다.

공의 속도 구간에 따라 속도의 감소와 구름 회전에 대한 저항을 테스트를 통해 최적화시킨 결과이다.

참고한 당구역학 논문을 통해서 대략적인 저항값이 0.029임을 알게 되었으나 unity에서는 FixedUpdate()에서 물리처리를 하기 때문에 논문과는 조금 다른 결과를 가져온다.

따라서 좀 더 세밀한 조정을 위해 구간별로 다른 값을 주어 공의 저항값을 최적화시켰다. 멈추기 직전의 경우 공의 속도가 매우 작아서 구름 운동이 사라지기 때문에 저항값이 커져서 약 10배의 저항을 주게 되었다.

```

private void CheckStop()
{
    //속도 체크
    if (rb.velocity.magnitude < 0.001)
        rb.velocity.Set(0, 0, 0);

    //회전 체크
    if (Mathf.Abs(rb.angularVelocity.x) < 0.1)
    {
        rb.angularVelocity.Set(0, rb.angularVelocity.y, rb.angularVelocity.z);
    }
    else if (Mathf.Abs(rb.angularVelocity.x) < 3)
    {
        rb.angularVelocity.Set(rb.angularVelocity.x * 0.8f, rb.angularVelocity.y, rb.angularVelocity.z);
    }

    if (Mathf.Abs(rb.angularVelocity.y) < 0.1)
    {
        rb.angularVelocity.Set(rb.angularVelocity.x, 0, rb.angularVelocity.z);
    }
    else if (Mathf.Abs(rb.angularVelocity.y) < 3)
    {
        rb.angularVelocity.Set(rb.angularVelocity.x, rb.angularVelocity.y * 0.8f, rb.angularVelocity.z);
    }

    if (Mathf.Abs(rb.angularVelocity.z) < 0.1)
    {
        rb.angularVelocity.Set(rb.angularVelocity.x, rb.angularVelocity.y, 0);
    }
    else if (Mathf.Abs(rb.angularVelocity.z) < 3)
    {
        rb.angularVelocity.Set(rb.angularVelocity.x, rb.angularVelocity.y, rb.angularVelocity.z * 0.8f);
    }
}

```

공의 정지를 체크하는 함수이다. 유니티상의 운동만으로는 완전 정지가 이루어지지 않고 아주 작은 값의 운동을 지속하기 때문에 공의 정지를 제어해 주어야 한다. 공의 저항에 의해 멈추기 직전의 큰 저항이 적용되는 구간에서 1초 안에 멈추기 위해 저항이 약 40번 계산된 후 속도가 멈추도록 정의하였다.

회전의 경우 이동 방향과는 다른 방향으로의 회전이 존재하기 때문에 멈춤에 대해 따로 정의해주어야 한다. 따라서 x, y, z에 대한 회전속도를 각각 따로 계산하여 멈춘 상태에 이르도록 회전 저항을 보정하고, 멈춤을 정의하였다.

7.1.2 타격에 대한 물리

```
private void Force(Collider other)
{
    //Debug.Log("trp" + ptrf.localPosition + "prep" + prePos + "V" + velocity);
    colrb = other.gameObject.GetComponent<Rigidbody>();
    colrb.velocity.Set(0, 0, 0);
    colrb.angularVelocity.Set(0, 0, 0);
    Physics.Raycast(ptrf.position, ptrf.forward, out hit, 2f);
    colrb.AddForceAtPosition(transform.forward * velocity.magnitude * 250f, hit.point);
    colrb.AddTorque(transform.forward * velocity.magnitude);
}

private void SetForce()
{
    if (cueGrap.IsGrap)
    {
        if (!trigger)
        {
            trigger = true;
            StartCoroutine("GetPrePos");
        }
    }
    else
    {
        if (trigger)
        {
            trigger = false;
            StopCoroutine("GetPrePos");
        }
    }
}

//감지
private void OnTriggerEnter(Collider other)
{
    //공일때
    if (other.tag.Equals("ball"))
    {
        //SetData0:
        Force(other);
        SoundManage.Instance.PlaySoundShot("shotStrong");
    }
}

IEnumerator GetPrePos()
{
    WaitForSeconds wait = new WaitForSeconds(0.03f);
    while (true)
    {
        prePos = ptrf.localPosition;
        //Debug.Log("prePos update");
        yield return wait;
        velocity = (ptrf.localPosition - prePos);
    }
}

//접촉중
private void OnTriggerStay(Collider other)
{
    //공일때
    if (other.tag.Equals("ball"))
    {
        colrb.AddForceAtPosition(transform.forward * velocity.magnitude, hit.point);
        //colrb.AddTorque(transform.forward * velocity.magnitude);
    }
}
```

공 타격시 큐에서 주는 힘을 계산하여 적용하는 코드이다. 타격을 위한 자세를 취하면 큐는 트리거로 설정되어 공을 타격시 OnTriggerEnter() 함수가 실행되며 타격한 공의 오브젝트를 가져올 수 있다. 그리고 큐의 타격 힘은 GetPrePos() 함수를 병렬실행하여 0.03초 이전의 위치와 현재 위치를 비교하여 속도를 설정한다. 이때 큐의 힘을 공에 적용시키는 force()함수를 실행시켜 공에 큐의 힘을 전달해준다. 이때 Raycast를 이용해 큐의 타격점을 알아내고 AddforceAtPosition() 함수를 이용해 타격점에 대해서 큐의 타격방향으로 힘을 주어 타격힘을 회전에 영향을 미치도록 한다. 그리고 AddTorque() 함수를 이용해 회전량을 보정 해준다. 그리고 밀어치기, 끌어치기의 적용을 위해 큐가 공과 접촉해있는 동안 실행되는 OnTriggerStay() 함수에서 큐가 공에 주는 힘을 보정한다.

```

if (laser.enabled)
    laser.SetPosition(0, transform.position);
if (Physics.Raycast(transform.position, transform.forward, out hit2, 2f))
{
    if (hit2.collider.tag.Equals("ball"))
    {
        if (!laser.enabled)
            laser.enabled = true;
        laser.SetPosition(1, hit2.point);
        rayDir.Set(transform.forward.x, 0f, transform.forward.z);
        rayDir.Normalize();
        if (Physics.SphereCast(hit2.collider.transform.position, 0.0308f, rayDir, out hit3, 2f))
        {
            predictPos = hit2.collider.transform.position + rayDir * hit3.distance;
            if (!shadowBall.activeSelf)
                shadowBall.SetActive(true);
            shadowBall.transform.position = predictPos;
        }
    }
    else
    {
        if (laser.enabled)
            laser.enabled = false;
        if (shadowBall.activeSelf)
            shadowBall.SetActive(false);
    }
}
else
{
    if (laser.enabled)
        laser.enabled = false;
    if (shadowBall.activeSelf)
        shadowBall.SetActive(false);
}
}

```

공 타격시의 예상 충돌지점을 알려주는 기능이다. 큐에서 시작되는 RayCast를 통해 수구의 타격점을 알아내서 이동 방향을 알아내고, 수구의 이동 방향에 SphereCast를 통해 공의 진행 방향 상에서 충돌할 거리를 알아낸다. 이 거리로 충돌지점을 알아내서 충돌지점에 투명한 공을 생성하여 플레이어에게 대략적인 예상 경로를 알려준다.

7.1.3 공의 충돌에 대한 물리

공이 벽에 충돌한 이후 공의 이동을 정의하는 코드이다. 벽에 대한 충돌은 각각의 벽마다 공의 회전이 적용되는 방향이 달라서 각각의 벽마다 방향을 올바르게 조정했다.

```
private void Reflect(Collision collision)
{
    Rigidbody rb = collision.collider.GetComponent<Rigidbody>();
    Vector3 incomingVector = rb.velocity;
    speed = incomingVector.magnitude;
    incomingVector = incomingVector.normalized;

    // 충돌한 면의 법선 벡터
    Vector3 normalVector = collision.GetContact(0).normal;

    // 법선 벡터와 입사벡터를 이용하여 반사벡터를 알아낸다.
    Vector3 reflectVector = Vector3.Reflect(incomingVector, normalVector); //반사각
    reflectVector = reflectVector.normalized;

    e = -Mathf.Pow(5f / 7f, speed + 3.12f) + 0.95f;
    mu = 0.471f - 0.241f * Vector3.Dot(rb.velocity, new Vector3(0, 0, rb.velocity.x));
    Debug.Log("e" + e + " mu " + mu);

    rb.velocity = reflectVector * speed * e + new Vector3(Mathf.Clamp(rb.angularVelocity.y * 0.018f, -10, 10), 0, 0);
    rb.angularVelocity = rb.angularVelocity * (1f - mu);
}
```

공과 쿠션의 충돌은 강체의 충돌이므로 OnCollisionEnter() 함수에서 실행되어 Reflect() 함수를 정의하여 충돌 후 공의 방향과 속도, 회전에 대한 이동 방향과 속도를 보정한다. 쿠션의 반발계수 e를 당구 역학 논문을 참고하여 공의 속도에 따라 변화함을 알아내고 그 관계를 수식을 통해 계산한다. 공의 반사 방향을 구하여 공의 이전 속도에 반발계수를 곱하여 이후 속도를 구하고 회전이 주는 추가적인 힘을 더해준다.

이때 실제 회전이 주는 힘에는 한계치가 존재하기에 Clamp함수를 통해 유효값 내로 한정한다. 그리고 공의 회전력을 감소시키는 정도인 쿠션의 마찰계수 mu는 공의 입사각이 크게 영향을 주므로 내적을 이용해서 구한다. 그 후 계산된 마찰계수를 이용하여 충돌 후 공의 회전속도를 계산한다.

```
private void Reflect(Collision collision)
{
    Rigidbody rb = collision.collider.GetComponent<Rigidbody>();
    Vector3 incomingVector = rb.velocity;
    speed = incomingVector.magnitude;
    incomingVector = incomingVector.normalized;

    // 충돌한 면의 법선 벡터
    Vector3 normalVector = collision.GetContact(0).normal;

    // 법선 벡터와 입사벡터를 이용하여 반사벡터를 알아낸다.
    Vector3 reflectVector = Vector3.Reflect(incomingVector, normalVector); //반사각
    reflectVector = reflectVector.normalized;

    rb.velocity = reflectVector * speed * 0.65f;
}
```

바닥의 충돌 처리 코드이다. 공의 움직임 처리에서 저항값을 설정하였기 때문에 바닥은 공이 Bounce 되었을 경우만 계산한다. 쿠션의 충돌과 같이 반사각을 구하고, 논문을 통해 알아낸 바닥면의 반발계수를 적용하여 충돌 후 공의 속도를 계산한다.

7.2 리플레이

7.2.1 리플레이 저장 코드

```
Dictionary<string, string> data = new Dictionary<string, string>();

data.Add("ball1pos_x", ball1trans.position.x.ToString());
data.Add("ball1pos_y", ball1trans.position.y.ToString());
data.Add("ball1pos_z", ball1trans.position.z.ToString());

data.Add("ball1rot_x", ball1trans.rotation.eulerAngles.x.ToString());
data.Add("ball1rot_y", ball1trans.rotation.eulerAngles.y.ToString());
data.Add("ball1rot_z", ball1trans.rotation.eulerAngles.z.ToString());

data.Add("ball2pos_x", ball2trans.position.x.ToString());
data.Add("ball2pos_y", ball2trans.position.y.ToString());
data.Add("ball2pos_z", ball2trans.position.z.ToString());

data.Add("ball2rot_x", ball2trans.rotation.eulerAngles.x.ToString());
data.Add("ball2rot_y", ball2trans.rotation.eulerAngles.y.ToString());
data.Add("ball2rot_z", ball2trans.rotation.eulerAngles.z.ToString());

data.Add("ball3pos_x", ball3trans.position.x.ToString());
data.Add("ball3pos_y", ball3trans.position.y.ToString());
data.Add("ball3pos_z", ball3trans.position.z.ToString());

data.Add("ball3rot_x", ball3trans.rotation.eulerAngles.x.ToString());
data.Add("ball3rot_y", ball3trans.rotation.eulerAngles.y.ToString());
data.Add("ball3rot_z", ball3trans.rotation.eulerAngles.z.ToString());

data.Add("quepos_x", quetrans.position.x.ToString());
data.Add("quepos_y", quetrans.position.y.ToString());
data.Add("quepos_z", quetrans.position.z.ToString());

data.Add("querot_x", quetrans.rotation.eulerAngles.x.ToString());
data.Add("querot_y", quetrans.rotation.eulerAngles.y.ToString());
data.Add("querot_z", quetrans.rotation.eulerAngles.z.ToString());

data.Add("deltatime", Time.deltaTime.ToString());

jsonData = new JSONObject(data);

socket.Emit("SendDataByUnity", jsonData);
```

SendFrameByUnity()함수로 Unity에서 NodeJS서버로 공 3개와 큐의 좌표값 x,y,z, 회전값 x,y,z을 매 프레임마다 보내게 된다.

다음 프레임간의 거리(지연시간)도 함께 보내어 프로그램 구동 환경마다 프레임이 달라지는 문제점 해결한다.

```
var sql = 'create table '+title+'(id INT not null auto_increment, '
        +'ball1pos_x float not null, ball1pos_y float not null, ball1pos_z float not null, '+
        +'ball1rot_x float not null, ball1rot_y float not null, ball1rot_z float not null, '+
        +'ball2pos_x float not null, ball2pos_y float not null, ball2pos_z float not null, '+
        +'ball2rot_x float not null, ball2rot_y float not null, ball2rot_z float not null, '+
        +'ball3pos_x float not null, ball3pos_y float not null, ball3pos_z float not null, '+
        +'ball3rot_x float not null, ball3rot_y float not null, ball3rot_z float not null, '+
        +'quepos_x float not null, quepos_y float not null, quepos_z float not null, '+
        +'querot_x float not null, querot_y float not null, querot_z float not null, '+
        +'deltatime float not null, constraint '+title+'_PK primary key(id));';

connection.query(sql, function(err, row, fields){
```

테이블 설계와 같이 각 게임 오브젝트(큐, 당구공)의 위치값, 회전값을 저장하는 테이블을 생성하고, 서버에서는 해당 sql문에 파라미터로 Unity에서 받은 데이터를 사용하여 해당 테이블에 insert 하게 될 예정이다.

7.2.2 리플레이 불러오기 코드

```
var sql = 'select * from '+data.title+'';
console.log("tablerows: "+tablerows);
connection.query(sql,function(err,rows, fields){
    if(err){
        console.log(err);
    }else{
        for(var i=0;i<tablerows;i++){
            msg.push({
                ball1pos_x: rows[i].ball1pos_x,
                ball1pos_y: rows[i].ball1pos_y,
                ball1pos_z: rows[i].ball1pos_z,
                ball1rot_x: rows[i].ball1rot_x,
                ball1rot_y: rows[i].ball1rot_y,
                ball1rot_z: rows[i].ball1rot_z,
                ball2pos_x: rows[i].ball2pos_x,
                ball2pos_y: rows[i].ball2pos_y,
                ball2pos_z: rows[i].ball2pos_z,
                ball2rot_x: rows[i].ball2rot_x,
                ball2rot_y: rows[i].ball2rot_y,
                ball2rot_z: rows[i].ball2rot_z,
                ball3pos_x: rows[i].ball3pos_x,
                ball3pos_y: rows[i].ball3pos_y,
                ball3pos_z: rows[i].ball3pos_z,
                ball3rot_x: rows[i].ball3rot_x,
                ball3rot_y: rows[i].ball3rot_y,
                ball3rot_z: rows[i].ball3rot_z,
                quepos_x: rows[i].quepos_x,
                quepos_y: rows[i].quepos_y,
                quepos_z: rows[i].quepos_z,
                querot_x: rows[i].querot_x,
                querot_y: rows[i].querot_y,
                querot_z: rows[i].querot_z,
                deltatime: rows[i].deltatime
            });
        }
    }
});
```

```
IEnumerator WaitFotit()
{
    Debug.Log("코루틴 실행");
    replaycue.SetActive(true);
    for (int i = 0, j = 0; i < temp.Count; )
    {
        yield return new WaitForSeconds(deltaList[j++]);
        ball1trans.position = temp[i++];
        ball1trans.rotation = Quaternion.Euler(temp[i++]);
        ball2trans.position = temp[i++];
        ball2trans.rotation = Quaternion.Euler(temp[i++]);
        ball3trans.position = temp[i++];
        ball3trans.rotation = Quaternion.Euler(temp[i++]);
        replaycue.transform.position = temp[i++];
        replaycue.transform.rotation = Quaternion.Euler(temp[i++]);
    }
    replaying = false;
    replaycue.SetActive(false);
    temp.Clear();
}
```

Unity에서 테이블 정보에 대한 이벤트 발생이 서버로 전달되면 서버는 테이블의 해당 테이블을 전체 select 하여 테이블의 모든 행을 다시 Unity로 전달 하게 된다.[5]

PlayReplaying 함수에서 게임 오브젝트에 대한 위치값,회전값을 수정하게 된다.

7.3 가이드 기능

7.3.1 학습모델 코드

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(32, 64, 1), kernel_size=(3, 3), filters=8),
    tf.keras.layers.MaxPool2D(strides=(2, 2)),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=16),
    tf.keras.layers.MaxPool2D(strides=(2, 2)),
    tf.keras.layers.Conv2D(kernel_size=(3, 3), filters=32),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(units=7, activation='softmax')
])
name="Strike_Classification"

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.5),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 62, 8)	80
max_pooling2d (MaxPooling2D)	(None, 15, 31, 8)	0
conv2d_1 (Conv2D)	(None, 13, 29, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 6, 14, 16)	0
conv2d_2 (Conv2D)	(None, 4, 12, 32)	4640
flatten (Flatten)	(None, 1536)	0
dense (Dense)	(None, 128)	196736
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 7)	903
Total params: 203,527		
Trainable params: 203,527		
Non-trainable params: 0		

설계대로 모델을 코딩하여 모델을 제작하게 되었다. 오른쪽 사진은 모델 구조를 출력한 결과이다.

활성함수는 Adam, 손실함수는 sparse_categorical_crossentropy로 적용했다. [6]

7.3.1 공 위치 좌표를 이용한 학습 이미지로의 변환 및 모델 불러오기 및 결과 출력

```
cv2.circle(img, (point1[0], point1[2]), 1, (255, 255, 255), -1)
cv2.circle(img, (point2[0], point2[2]), 1, (0, 0, 255), -1)
cv2.circle(img, (point3[0], point3[2]), 1, (0, 0, 255), -1)

model = load_model('strike_guide_model.h5')

np.argmax(model.predict_classes(img))
```

공 위치를 받아 이를 인풋 이미지로 변환하는 파이썬 코드이다. 유니티 프로그램에서 수신한 좌표값을 이용해 이미지를 생성하게 된다.

학습된 모델을 불러와 생성된 이미지를 전달하여 예측 결과를 얻어오게 된다.

7.3.2 서버 송수신 코드

```
socket.on('GetCommendedStroke', function(jdata){
    options.args[0] = jdata.point1;
    options.args[1] = jdata.point2;
    options.args[2] = jdata.point3;

    PythonShell.run("Stroke_Commend.py", options, function(err, results){
        if(err) console.log('err msg:', err);
        console.log('results: %j', results);

        io.emit('RecieveStrokeCommend', { result: results });
    })
})
})
```

서버 코드

```
// 입력 데이터 전송 함수
public void getRecommendedStroke(){
    Dictionary<string, string> data = new Dictionary<string, string>();
    data.Add("point1", ballPoints[0]);
    data.Add("point2", ballPoints[1]);
    data.Add("point3", ballPoints[2]);

    // 전송 데이터
    jdata = new JSONObject(data);
    socket.Emit("GetCommendedStroke", jdata);
}

// 결과 수신 및 변수 저장 함수
public void rcvRecommendedStroke(SocketIOEvent e){
    Debug.Log(e.data.Json.args[0] + ", " + e.data.Json.args[1]);
    // 예측 결과
    recommendedStroke = STROKE[Int32.Parse(e.data.Json.args[0])] + e.data.Json.args[1];
}
```

클라이언트 코드

유니티 프로그램과 서버간의 데이터를 송수신하는 코드다. 유니티 프로그램에서 가이드 요청 시 현재 공의 위치를 서버에 송신한다. 이 데이터를 이용하여 학습 이미지의 변환을 거쳐 모델에게 전달하여 결과를 얻게 된다. 이 결과를 유니티 프로그램으로 전송해 결과를 화면에 출력하게 되는 결과를 얻게 되었다.

7.3.3 유니티 프로그램에서의 가이드 기능 수행 과정 및 결과



위 사진은 가이드 기능 수행 과정 및 결과 이미지다. 메뉴에서 가이드 보기 버튼 클릭 후 수구를 선택할 수 있게 된다. 선택 후 공이 멈춰있거나 멈추게 될 때 해당 공의 위치에 따른 추천 타법이 수구 위에 출력되게 된다.

7.4 컨트롤러

컨트롤러의 기능을 정의하기 위해 앞서 플레이어의 현재 상태에 따라 사용 가능한 기능이 달라지기 때문에 mmode 변수를 통해 초기의 기본상태, 큐를 든 상태, 큐를 조준한 상태, 메뉴를 킨 상태, 물건을 들고 있는 상태로 나누어 실행되도록 작성했다.

7.4.1 우측 컨트롤러 주요 동작 코드

```
void Update()
{
    PadAction();
    MenuAction();

    switch (mmode.mode)
    {
        case 0: // 기본 상태
            CueAction(); //큐 꺼내기
            GrapAction(); //물건 집기
            break;
        case 1: // 큐 든 상태
            CueAction(); //큐 넣기
            Follow(); //큐 위치 지정
            break;
        case 2: // 메뉴 상태
            break;
        case 3: // 물건 든 상태
            GrapAction(); //물건 놓기/던지기
            break;
    }
}

switch (PlayerPrefs.GetInt("assist", 0))
{
    case 0: //어시스트모드 off
        handle.transform.position = transform.position;
        if (isGrp.IsGrp)
        {
            if (cueCol.IsTrigger)
            {
                cueCol.IsTrigger = true;
                handle.transform.LookAt(holdPos);
            }
            else
            {
                if (cueCol.IsTrigger)
                {
                    cueCol.IsTrigger = false;
                }
                if (cueGrp)
                {
                    cueGrp = false;
                    handle.transform.rotation = transform.rotation;
                    vHoldPos = handle.transform.position;
                }
                break;
            }
        }
    case 1: //어시스트모드 on
        if (isGrp.IsGrp)
        {
            if (cueCol.IsTrigger)
            {
                cueCol.IsTrigger = true;
                controllerMovePos = controllerPose.transform.position - vHoldPos;
                theta = Vector3.Dot(handle.transform.forward, controllerMovePos);
                degree = Mathf.Rad2Deg * theta;
                //Debug.Log("각도: " + degree);
                cuepos = (controllerPose.transform.position - vHoldPos).magnitude * degree * 0.08f;
                handle.transform.position = vHoldPos + cuepos * (handle.transform.forward);
            }
            else
            {
                Physics.Raycast(cue.transform.position, cue.transform.forward, out hit, 10f);
                handle.transform.position = transform.position;
                handle.transform.rotation = transform.rotation;
                vHoldPos = handle.transform.position;
                if (cueGrp)
                {
                    cueGrp = false;
                }
                if (cueCol.IsTrigger)
                {
                    cueCol.IsTrigger = false;
                }
            }
            break;
        }
}
```

컨트롤러의 이벤트는 Update() 함수에서 체크하며 컨트롤러의 각 버튼에 맵핑된 변수를 통해 코드를 실행한다.

우측 컨트롤러는 터치패드로 플레이어의 방향전환, 큐 들기/없애기, 점프가 가능하고, 추가로 메뉴 호출, 메뉴 호출 상태일 때 버튼과 상호작용, 물체 잡기 등의 기능을 수행할 수 있다.

큐를 든 상태일 때에는 follow() 함수를 통해 오른손이 큐의 손잡이를 잡은 상태가 된다.

이때, 일반 모드일 경우에는 조준시 왼손을 따라다녀서 실제 타자유롭게 타격이 가능하고, 어시스트 모드를 활성화 하면 조준시 큐가 현재 조준된 각도를 유지하여 처음 조준한 지점으로 흔들림 없이 타격할 수 있다.

7.4.2 좌측 컨트롤러 주요 동작 코드

```
void Update()
{
    // 왼손 터치패드 동작
    // 옵션으로 모드 조정해서 텔레포트, 방향이동 선택
    if (isTeleport.activeSelf)
        Teleporting();
    else
        Moving();

    MenuAction(); // 메뉴
    switch (mmode.mode)
    {
        case 0: // 기본 상태
            GrapAction(); // 물건 집기
            break;
        case 1: // 큐 든 상태
            GrapCue(); // 큐 고정
            break;
        case 2: // 메뉴 상태
            break;
        case 3: // 물건 든 상태
            GrapAction(); // 물건 놓기/던지기
            break;
    }
}
```

좌측 컨트롤러 또한 컨트롤러 이벤트는 Update() 함수에서 체크하며 컨트롤러의 버튼을 누르면 각 버튼에 맵핑된 변수를 조작하여 코드를 실행한다. 우측 컨트롤러는 컨트롤러의 터치패드 클릭으로 플레이어 이동/텔레포트가 가능하고, 추가로 큐를 든 상태일 때 조준, 메뉴 호출, 물체 잡기 등이 가능하다.

현재는 다음의 목표를 위해 프로그램 기능 시험, 서버와의 통신 테스트를 진행 중이다. 첫째, 성능 향상을 위한 코드 정제 및 발견하지 못한 에러에 대한 QA 진행을 통해 코드 완성도 향상. 둘째, 최초 사용자를 위한 기능, 동작에 대한 설명이 필요하다고 판단하여 튜토리얼과 같은 메뉴얼을 작성. 또한, QA를 진행하며 프로젝트에서 각자의 맡은 부분만을 아는 것보다 서로의 담당 부분을 팀원에게 설명을 해주며 본인은 복습을 다른 팀원들은 학습과 프로젝트에 대한 이해도의 향상을 하고 있다. 위의 과정을 수행하여 최종적으로 결과물에 대한 결론 작성을 진행할 예정이다.

III. 결론

1. 연구결과 및 발전 방향

- 당구 물리 구현

동역학 논문을 실제 Unity에 적용시켜 구현해본 결과 당구 물리를 구현하는 데에 **공의 회전과 저항에 대한 중요성**이 크다는 것을 알았다. 동역학 논문을 통해 구체의 운동방정식을 분석하여 적용할 식을 정의하고 이것을 Unity에 적용하였는데, 일반적인 타격에 대한 물리와 공과 공, 공과 쿠션의 충돌은 구현할 수 있었다. 그러나 **맞세이나 극회전같은 타격은 구현하기 어려웠다**. 맞세이는 당구 동역학 논문에서는 2D의 움직임으로 연구했기 때문에 타격점과 공이 충돌하여 회전이 적용받는 축이 달라지는 타격에는 또 다른 물리 계산이 필요해 보인다. 극회전은 공의 속도 대비 회전속도가 너무 커서 불안정한 움직임을 보여주어서 컴퓨터 자원을 많이 사용하지 않는 한 정확한 구현이 어려웠다.

VR 컨트롤러에서 한 버튼이 여러 가지 기능을 하면 초심자들에게 접근성이 낮다는 것을 알았다. 그래서 **사용자 이동 및 타격에 필요한 기능을 제외한 다른 기능은 메뉴에서 옵션으로 선택할 수 있게끔 하여 같은 루틴의 기능을 한 사이클로 구현**하였다. 여기에 더해서, 사용자가 VR컨트롤러 버튼에 대한 동작을 자신이 사용하기 편리하게 재할당 할 수 있는 기능과 같이 사용자의 환경을 더 편하게 조절할 수 있는 다른 옵션을 추가하면 접근성이 높아질 것으로 보인다.

- 가이드 기능의 완성 결과 및 발전 방안

가이드 기능은 공의 위치에 따른 클래스 분류를 수행하는 딥러닝 기술을 적용하였다. 클래스 분류에서의 정확도를 위해서는 각 클래스를 구분할 수 있는 특징을 선정함이 중요하다. 해당 프로젝트의 **학습모델은 이 특징을 공의 위치로 선정**하였다. 공의 위치에 따라 비슷한 타점, 방향으로의 타격이 있을 것이고 이를 타법으로 부르기에 **공의 위치에 따른 타법 분류에 성공할 수 있을 것으로 판단**하였다. 당구에는 많은 타법이 있다. 하지만 공 위치에 따른 각 타법 간의 구분이 어렵다고 판단되어 사용이 잦은 타법 중 공 위치에 따른 구분이 더 나은 7개의 타법을 임의 선정하여 데이터를 수집하였다. 후에 학습모델 제작에 시도, 7개의 타법에 대해 각 1000개의 데이터와 여러 parameter를 조정한 결과 정확도는 약 70%인 학습모델을 완성하였다.

3구의 성공 방법에는 여러 방법이 있다. 그러나 타점, 세기, 두께, 목적구와의 거리 등의 제약사항에 의해 실제로 치는 방법의 수는 줄게 된다. 따라서 실제 성공 타구에 대한 타점, 세기 등의 특징을 수집한 데이터를 학습시킨다면 타법보다 정확한 성공 방법을 추천해 줄 수 있을 것으로 생각한다.

- 리플레이 구현 방식 선택 향후 사용자 행동 분석

리플레이를 구현하는 방법에는 두 가지 방법이 있다. 첫 번째, 결과를 따라하기 위한 상태를 기록하는 방법이 있고, 두 번째, 과정을 따라하기 위해 사건을 기록하는 방법이 있다.[7] 하지만 Unity의 물리 처리가 FixedTime함수에서 실시간 연산으로 이루어져 렌더링이 끝난 시점에서 알 수 있고, 그 결과가 매번 다르다. 따라서 오차를 줄이기 위해 우리는 게임 오브젝트(큐대, 당구공)의 상태를 매 Frame 별로 저장하였다. 하지만, 실행해본 결과 프로그램 구동 사양에 따른 Frame 속도의 차이 발생 때문에 다른 리플레이 결과가 나왔다. 이 문제를 해결하고자 리플레이를 저장할 때 다음 프레임까지의 시간을 계산해주는 함수(Time.deltaTime)를 이용하여 다음 Frame까지의 지연시간을 주어 해결했다.

영상으로 보여지는 리플레이와 달리 매 Frame별 좌표값, 회전값 Float 데이터를 String으로 저장하여 용량이 비교적 작아지고 사용자가 원하는 시점으로 자신이 타격한 장면을 볼 수 있다. 향후에는 다른 스포츠 종목에 적용하여 사용자의 타격에 대한 정보를 기록해놓고 데이터 분석을 한다면 사용자의 주로 선호하는 타격 및 취약점을 분석하는 사용자 행동 분석으로도 활용해볼 수 있을 것으로 생각된다.

- WebSocket과 Socket.IO

Web Client-Server 방식에는 polling, long polling, socket 통신 등등이 존재한다, 하지만 게임과 같은 경우 실시간으로 데이터가 양방향으로 통신하는 경우와 HTTP를 사용하지 않고 TCP/IP Socket처럼 connection이 유지되어 전체 메시지가 줄어들기 때문에 데이터 전송으로 인한 오버헤드가 줄어들게 되어 자주 사용된다.[8]

2. 작품제작 소요재료 목록

- H/W

품목	용도	수량
HTC Vive	VR 프로그램 실행을 위한 H/W 패키지	1개
베이스 스테이션 삼각대	트래킹 센서의 원활한 동작을 위한 거치대	2개 (트래킹 센서 각 1개씩 거치)

참고자료

1	당구 게임에서의 동역학	http://m.riss.kr/search/detail/DetailView.do?p_mat_type=1a0202e37d52c72d&control_no=427cad0733796426ffe0bdc3ef48d419#redirect
2	영상처리	https://d2.naver.com/helloworld/8344782
3	Neural Network	텐서플로우 첫걸음 chapter5, Aug. 16. 2016
4	Socket.io-Client for Unity3D 소개	https://meetup.toast.com/posts/112
5	WebSocket과 Socket.IO	https://d2.naver.com/helloworld/1336
6	Tensorflow	https://www.tensorflow.org/tutorials/quickstart/beginner?hl=ko
7	리플레이 구현 방법	https://sunhyeon.wordpress.com/2014/12/01/1627
8	WebSocket을 사용하는 이유	https://ymcoder.tistory.com/281