Portfolio

Client Programmer
차종원

chajw950@gmail.com

index

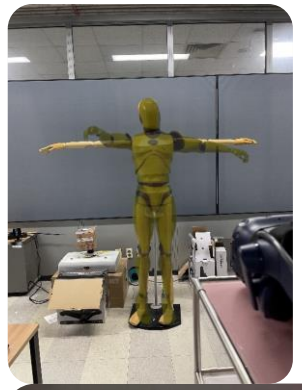
- 자기소개
- 메인 프로젝트
- 기타 프로젝트

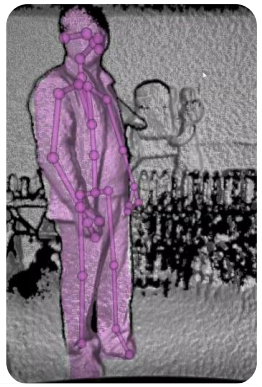
자기소개

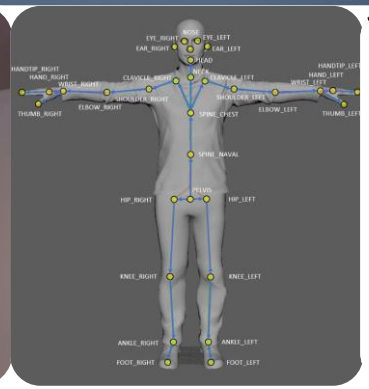
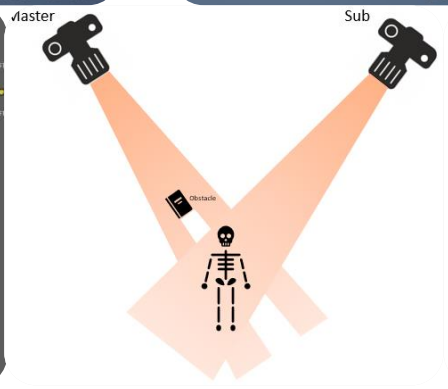
- 2014
  - 한국공학대학교 입학
  - C+/C++ 프로그래밍 수강
- 2015
  - 윈도우 프로그래밍 수강
    - 윈도우API를 사용한 **VVVVVV 모작**
- 2015 – 2017.08 군 휴학
- 2017.09
  - 2D게임 프로그래밍 수강
    - Python pico2d 엔진을 사용한 **황소 날리기** 모작
  - 컴퓨터 그래픽스 수강
    - Open GL을 사용한 3D 게임 **Slingshot** 제작
- 2018
  - 스크립트언어 수강
    - 공공API를 사용하여 **관광정보서비스 프로그램 개발**
  - STL 수강
  - 게임 소프트웨어 공학 수강
    - The Binding of Isaac의 물리 시스템을 모작하여 2.5D 게임 **Simple Isaac 개발**
  - 네트워크 게임 프로그래밍 수강
    - TCP 네트워크를 활용한 2D게임 **It's Mine** 개발
- 2019
  - 스마트폰 게임 프로그래밍 수강
    - 고전게임 Lunar Pool의 물리 시스템을 모작하여 2D 게임 **Pocketball 개발**
  - 게임서버프로그래밍 수강
    - IOCP를 활용하여 간단한 2D게임 제작
  - Unreal Engine4를 사용한 3D 비행 격투 게임 **Fly me to the air 개발**
- 2020
  - 한국공학대학교 일반대학원 입학
- 2021
  - 저가 모션 캡처 장치 Azure Kinect의 노이즈 감소와 다음 프레임을 예측하는 시스템을 개발하여 "**다중 영상기반 신체추적 신호의 칼리브레이션과 칼만필터 잡음 제거 기법**" 을 석사 논문으로 제출

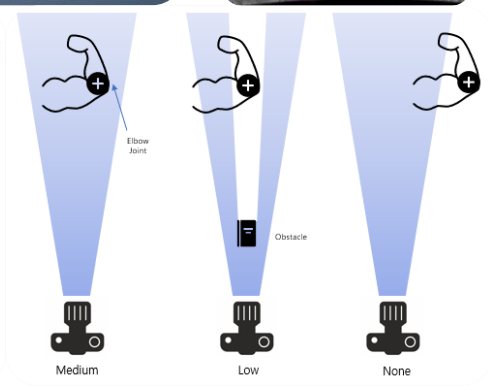
성명 : 차종원
010-9131-8286
chajw950@gmail.com

Optimizing Azure Kinect



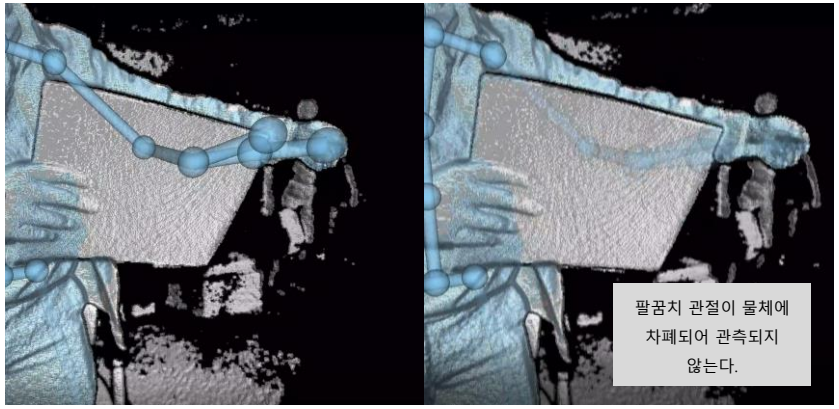


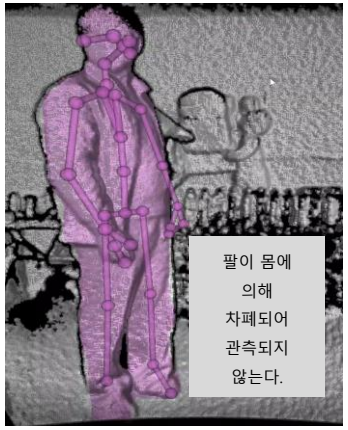
개발 목적	VR 기기 개발 방법 학습 Unreal Engine4 Plugin 개발 학습 Unreal Engine4 C++ 연동 학습 Azure Kinect의 최적화 기법 적용
개발 툴	Unreal Engine4, Visual studio 2017
개발 언어	C++
팀원	1인
개발 중점 사항	2대 이상의 Azure Kinect를 칼리브레이션 낮은 프레임의 Azure Kinect를 제어 공학을 통하여 프레임 개선

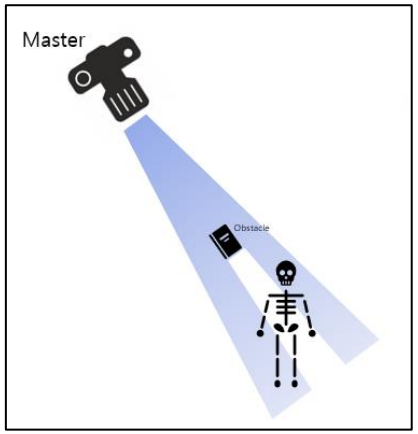
저가형 VR 기기 Azure Kinect의 낮은 Framerate와 노이즈를 최적화 하는 기법을 연구하였다.
여러 대의 Azure Kinect를 사용하여 Framerate를 높였고 제어 시스템 공학의 유명한 제어 기법인 Kalman Filter 기법을 사용하여 오차를 줄였고 나아가 다음 프레임의 정보를 예측하여 보여주는 효과를 가져왔다.

Optimizing Azure Kinect

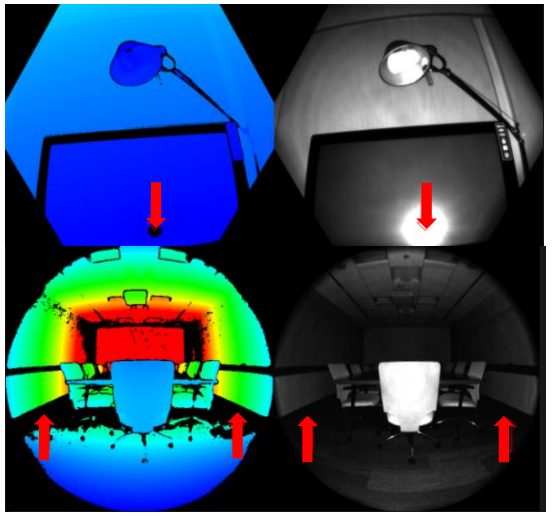
Azure Kinect Body Tracking 노이즈의 발생 원인

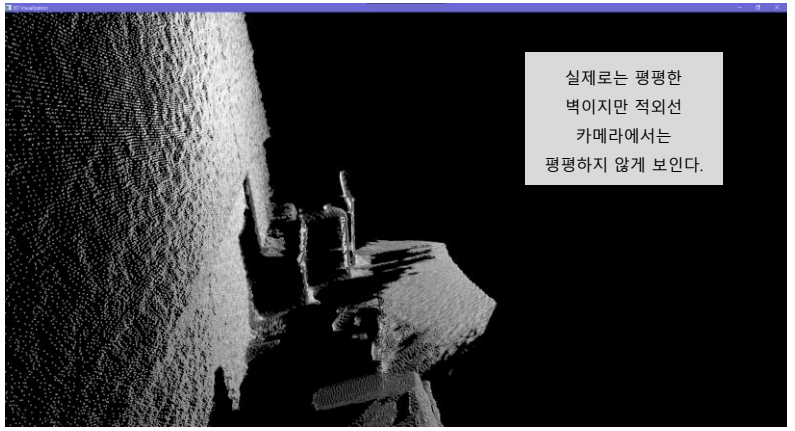






관절이 적외선 카메라에 관측되지 않을 시 잡음이 발생한다.

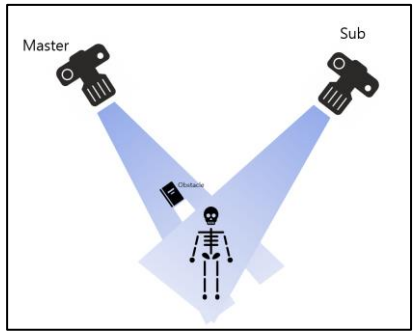




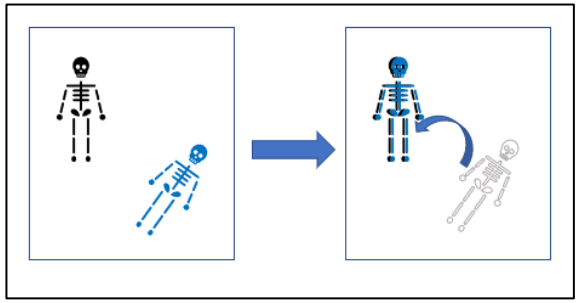
Azure Kinect 적외선 카메라의 성능 한계로 인하여 잡음이 발생한다.

Optimizing Azure Kinect

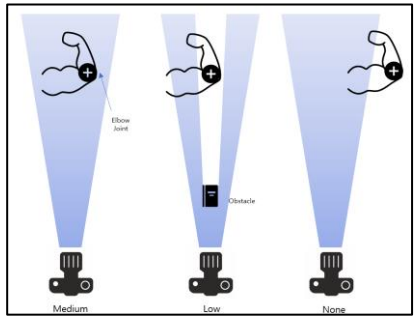
관절 차폐 문제 해결 방안



2대 이상의 Azure Kinect를 사용하여 가려진 부분을 또 다른 카메라로 관측한다.



Master 카메라를 기준으로 Sub 카메라의 관측된 관절 데이터를 Transform 하여 Skeleton을 통합한다.



관절이 차폐되어 있을 시 관절 Confidence Level이 Low로 설정되는데, 다른 카메라에서 해당 관절이 관측된다면 그 값을 관절 데이터에 업데이트한다.

```
VERIFY(k4a_device_open(0, &device), "Open K4A Device failed");
VERIFY(k4a_device_open(1, &Subdevice), "Open K4A Device failed - Sub");

k4a_device_get_sync_jack(device, &IsInConnected, &IsOutConnected);
if (IsInConnected)//device swap
{
    tempdevice = device;
    device = Subdevice;
    Subdevice = tempdevice;
    printf("DEVICE SWAPED\n");
}
```

0번, 1번 Azure Kinect 카메라를 open한 후 편의를 위해 마스터 카메라를 0번 카메라로 설정

```
//sub 스켈레톤
sub.pos = Eigen::Vector3f(position.xyz.x, position.xyz.y, position.xyz.z);
sub.quat = Quat(rotation.wxyz.w, rotation.wxyz.x, rotation.wxyz.y, rotation.wxyz.z);
sub.rot = quatToEuler(sub.quat);
if (head.confidence_level == K4ABT_JOINT_CONFIDENCE_MEDIUM)
{
    intergrated.pos = sub.pos - sub.transPos;
    intergrated.rot = sub.rot - sub.transRot;
    KF.predict(1);
    KF.update(intergrated.pos);
    //KF.xHat을 읽어 필터링된 관절 데이터를 사용.
}
```

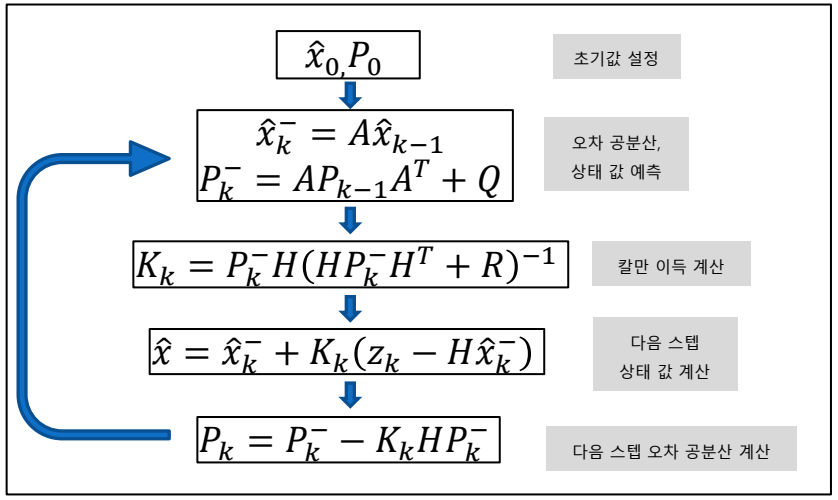
```
//sub 스켈레톤
sub.pos = Eigen::Vector3f(position.xyz.x, position.xyz.y, position.xyz.z);
sub.quat = Quat(rotation.wxyz.w, rotation.wxyz.x, rotation.wxyz.y, rotation.wxyz.z);
sub.rot = quatToEuler(sub.quat);
if (head.confidence_level == K4ABT_JOINT_CONFIDENCE_MEDIUM)
{
    intergrated.pos = sub.pos - sub.transPos;
    intergrated.rot = sub.rot - sub.transRot;
    KF.predict(1);
    KF.update(intergrated.pos);
    //KF.xHat을 읽어 필터링된 관절 데이터를 사용.
}
```

```
//master 스켈레톤
master.pos = Eigen::Vector3f(position.xyz.x, position.xyz.y, position.xyz.z);
master.quat = Quat(rotation.wxyz.w, rotation.wxyz.x, rotation.wxyz.y, rotation.wxyz.z);
master.rot = quatToEuler(master.quat);

LPFVal = (LPFVal * LPFalpha) + (position.xyz.x * (1 - LPFalpha));
if (head.confidence_level == K4ABT_JOINT_CONFIDENCE_MEDIUM) {
    intergrated.pos = master.pos;
    intergrated.rot = master.rot;
    KF.predict(1);
    KF.update(intergrated.pos);
    //KF.xHat을 읽어 필터링된 관절 데이터를 사용.
}
```

Optimizing Azure Kinect

관절 노이즈 발생 문제 해결 방안

Kalman Filter 알고리즘



제어 시스템 공학의 대표 필터링 기법인 Kalman Filter 기법을 사용하여 노이즈를 감소시킨다.

Kalman Filter는 측정 값에 포함된 잡음(Noise)를 최소화하여 신호를 추정하는 필터링 알고리즘이다.

KalmanFilter 싱글톤 클래스

```
class kalmanFilter {
public:
	Matrix<double, 6, 1> xHat;                    //추정값
	Matrix<double, 6, 6> covariance;    //오차공분산 매트릭스
	Matrix<double, 6, 6> A;                       //상태전이 매트릭스
	Matrix<double, 3, 6> H;             //측정 행렬
	Matrix<double, 6, 6> processNoise;  //Q값 - 모델 오차
	Matrix<double, 3, 3> measureNoise;  //R값 - 측정 오차

	kalmanFilter(double proNoise, double meaNoise, double initX, double initY) {
		A = Matrix<double, 6, 6>::Identity();
		H = Matrix<double, 3, 6>::Zero();
		H(0, 0) = 1;
		H(1, 1) = 1;
		H(2, 2) = 1;
		xHat = Matrix<double, 6, 1>::Zero();
		covariance = Matrix<double, 6, 6>::Identity();
		processNoise = Matrix<double, 6, 6>::Identity() * proNoise;
		measureNoise = Matrix<double, 3, 3>::Identity() * meaNoise;
		xHat(0, 0) = initX;
		xHat(1, 0) = initY;
	}

	static kalmanFilter& getInstance(double proNoise, double meaNoise, double initX, double initY) {
		static kalmanFilter instance(proNoise, meaNoise, initX, initY);
		return instance;
	}

	void predict(double dt) {
		A(0, 3) = A(1, 4) = A(2, 5) = dt;
		xHat = A * xHat;
		covariance = A * covariance * A.transpose() + processNoise;
	}

	void update(const Vector3d& z) {
		Vector3d y = z - H * xHat;
		Matrix3d temp = H * covariance * H.transpose() * measureNoise;
		Matrix<double, 6, 3> kalmanGain = covariance * H.transpose() * temp.inverse();

		xHat = xHat + kalmanGain * y;
		covariance = (Matrix<double, 6, 6>::Identity() - kalmanGain * H) * covariance;
	}

	kalmanFilter(const kalmanFilter&) = delete;
	kalmanFilter& operator=(const kalmanFilter&) = delete;  //싱글톤 보장
};
```

싱글톤 인스턴스 생성

상태 값, 오차 공분산 예측

상태 값, 오차 공분산 업데이트

싱글톤 인스턴스를 보장

통합 스켈레톤의 데이터를 읽어 칼만 필터를 통해 예측, 업데이트함으로써 데이터를 필터링함.

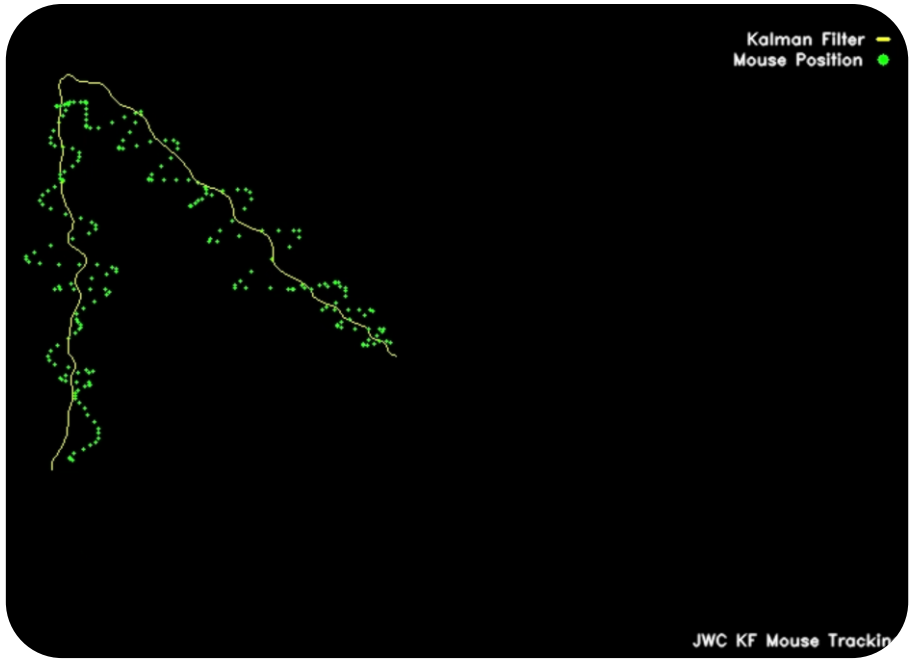
```
//sub 스켈레톤
sub.pos = Eigen::Vector3f(position.xyz.x, position.xyz.y, position.xyz.z);
sub.quat = Quat(rotation.wxyz.w, rotation.wxyz.x, rotation.wxyz.y, rotation.wxyz.z);
sub.rot = quatToEuler(sub.quat);
if (head.confidence_level == K4ABT_JOINT_CONFIDENCE_MEDIUM)
{
    intergrated.pos = sub.pos - sub.transPos;
    intergrated.rot = sub.rot - sub.transRot;
    KF.predict(1);
    KF.update(intergrated.pos);
    //KF.xHat을 읽어 필터링된 관절 데이터를 사용.
}
```

```
//master 스켈레톤
master.pos = Eigen::Vector3f(position.xyz.x, position.xyz.y, position.xyz.z);
master.quat = Quat(rotation.wxyz.w, rotation.wxyz.x, rotation.wxyz.y, rotation.wxyz.z);
master.rot = quatToEuler(master.quat);

LPFVal = (LPFVal * LPFalpha) + (position.xyz.x * (1 - LPFalpha));
if (head.confidence_level == K4ABT_JOINT_CONFIDENCE_MEDIUM) {
    intergrated.pos = master.pos;
    intergrated.rot = master.rot;
    KF.predict(1);
    KF.update(intergrated.pos);
    //KF.xHat을 읽어 필터링된 관절 데이터를 사용.
}
```

Optimizing Azure Kinect

칼만 필터 사용 개인 프로젝트

Kalman Filter Mouse Tracking



Kalman Filter를 사용하여 마우스 입력 값에 오차가 있다고 가정하고 올바른 데이터를 추정하는 프로그램.

(영상) https://youtu.be/a2FANg4i_Vo
(Github)https://github.com/JongWonCha/kalmanFilterBodyTracking

Kalman Filter Body Tracking

Kalman Filter를 사용하여 관절 데이터에 간헐적으로 오차가 발생하는 Azure Kinect의 오차를 줄여주는 프로그램. 2대의 Azure Kinect 카메라를 칼리브레이션 하는 기능 포함.

(영상) https://youtu.be/_OKpJgdF7eI
(Github)https://github.com/JongWonCha/kalmanFilterBodyTracking

Optimizing Azure Kinect

칼만 필터의 쓰임

Position 잡음

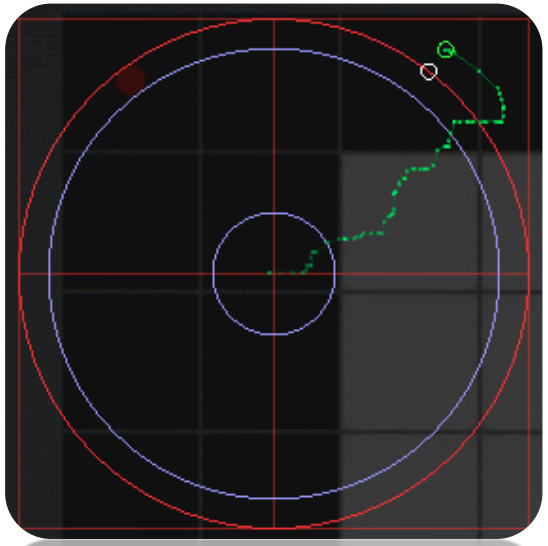
간헐적으로 플레이어의 Position이 흔들리는 현상을 완화시켜 움직임을 부드럽게 만든다.

(영상) https://youtu.be/zA9PrxpGGs8

모션캡처 애니메이션 제작

모션캡처 장비에 잡음이 포함될 경우 필터 알고리즘으로 보정 할 수 있다.

(영상) https://youtu.be/ufV7PlokoT0?t=612



아날로그 스틱 잡음 보정

게임패드로 플레이 할 시 아날로그 스틱의 잡음을 보정하여 더 쾌적한 플레이를 제공한다.

Fly me to the air

장르	3D 멀티 액션 게임
개발 목적	Unreal Engine4를 사용한 실시간 다중 접속 액션 게임 개발
개발 툴	Unreal Engine4, Visual studio 2017
개발 언어	Blueprint, C++
팀원	3인
담당 역할	클라이언트, 서버, 맵 제작
개발 중점 사항	Unreal Engine4 게임 개발 캐릭터, 투사체 움직임 구현 Dedicated Server를 이용한 온라인 게임 환경 구현 Unreal Material을 이용한 쉐이더 프로그래밍

상대 플레이어를 투사체로 맞춰 충격량 계수를 높인다.
충격량 계수가 높아지면 높아질 수록 야구 배트로 맞았을 때 더 멀리 날아간다.
플레이어가 맵 밖으로 떨어졌을 때에는 벌점 1점을 얻는다.
제한시간 내에 가장 벌점이 적은 플레이어가 승리.

(영상) SkyScraper Map : https://youtu.be/uEguNw0cTFc
Volcano Map : https://youtu.be/q9WIPdiToWs
FloatingIsland Map : https://youtu.be/gaGwgMuJ4Tg
(github) https://github.com/JongWonCha/FMTA

Fly me to the air – Characters, Maps

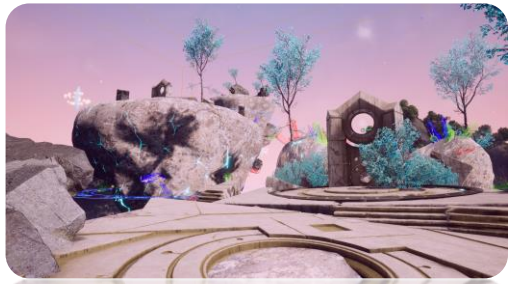
다양한 캐릭터와 맵

CHARACTERS

캐릭터 마다 능력을 다르게 함.
Power는 힘이 강하여 배트로 상대방을 히트 시 더 멀리 날아가게 한다. 반면 속도가 느리다.
Speed는 속도가 빨라 상대 플레이어의 투사체를 피하기 쉽다. 반면 힘이 약하다.
Balance는 속도와 힘이 모두 중간치이다.

MAPS

Floating Island

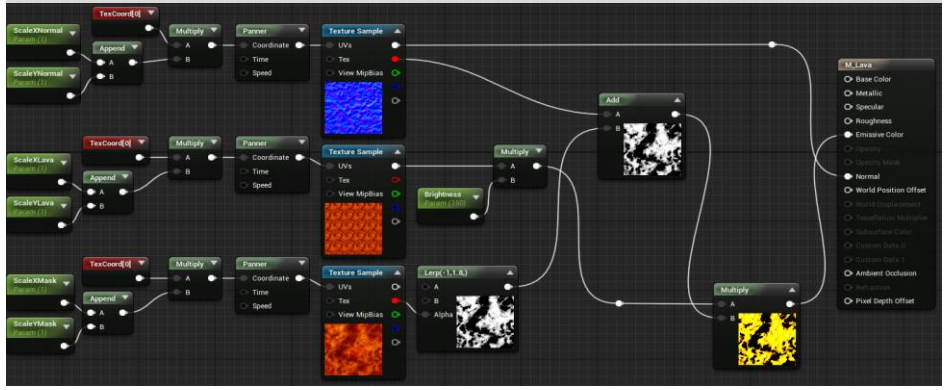


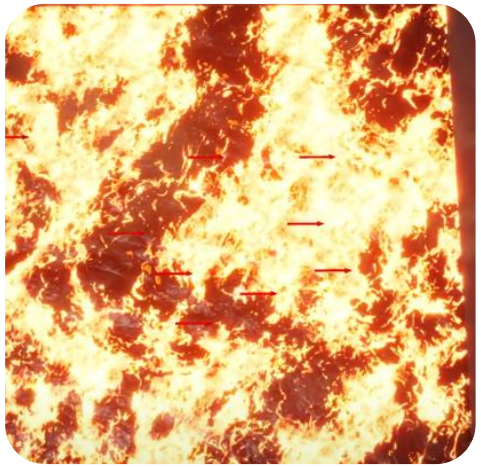
Skyscraper

Volcano

Fly me to the air – Shader programing

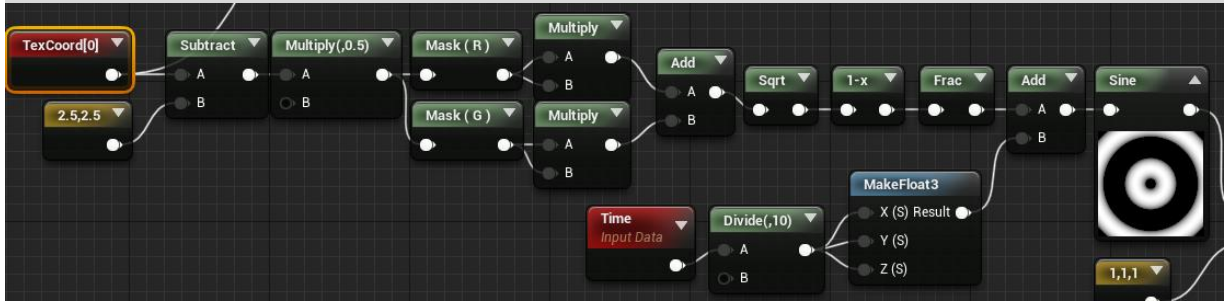
각 텍스처마다 다른 속도로 움직이게 하고 이들을 곱하고 더하여 생동감 있는 용암 움직임을 구현.

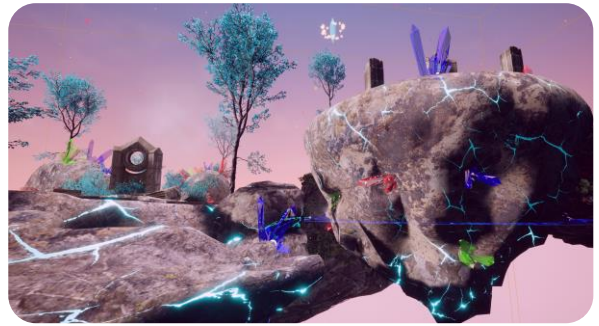




(영상) https://youtu.be/dyyJyZzB_XQ

텍스처 중점과의 거리 값과 Sin함수, Time값을 조합하여 중심으로부터 원이 퍼져 나가는 텍스처 구현

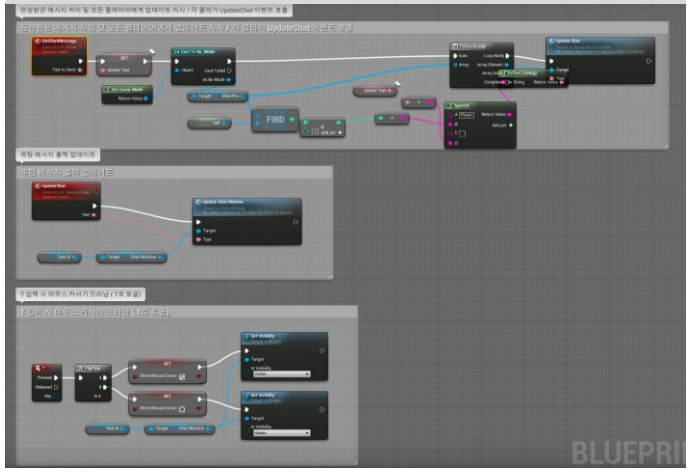




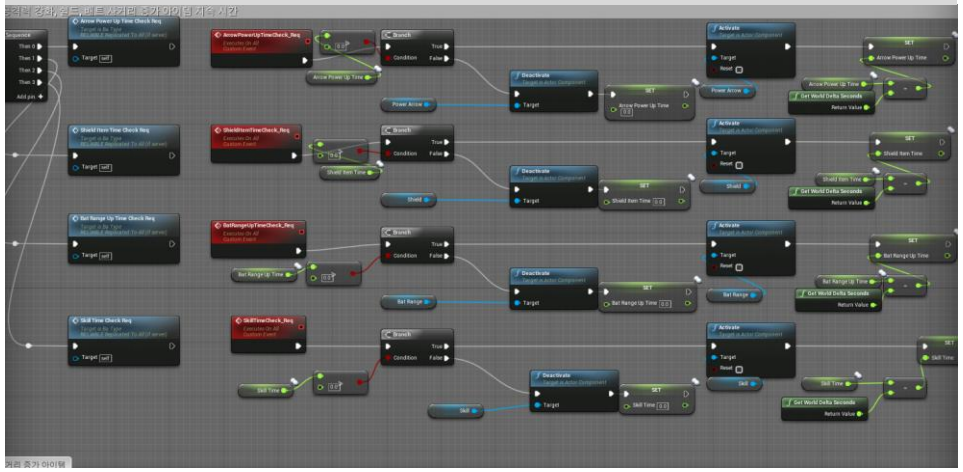
(영상) https://youtu.be/EKBNXuMttGo

Fly me to the air – Dedicated server

서버에 채팅 내용과 사용자 이름을 보내고 서버가 멀티 캐스팅하는 방식으로 채팅 시스템 구현



변수와 함수 리플리케이트 기능을 이용하여 각종 이펙트, 플레이어의 변수, Position, Rotation 동기화



Physics Project Series

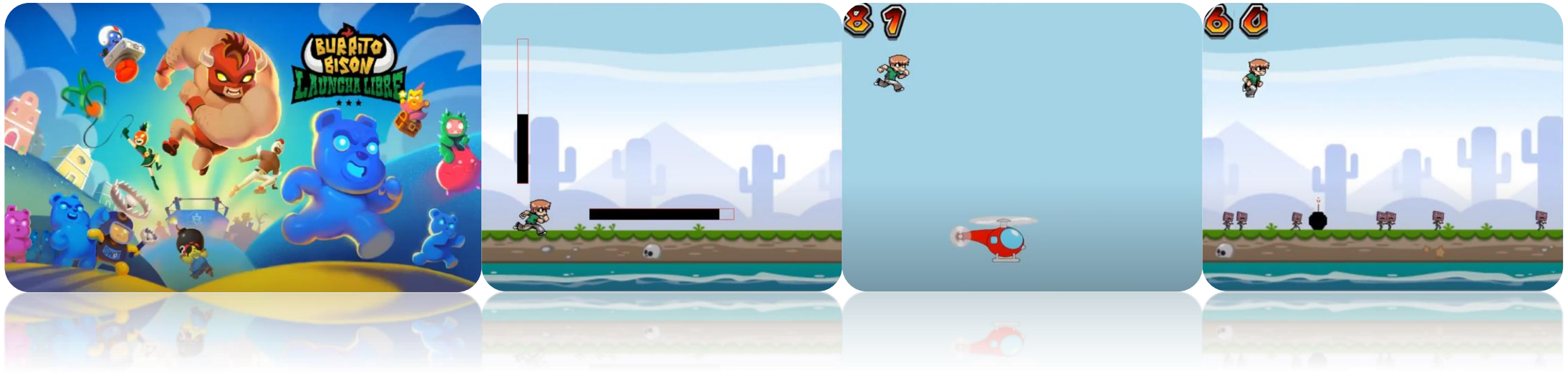
Burrito Bison : Launcha Libre를 모작한 게임 '**황소 날리기**' 제작

아이작의 번제의 물리를 모작한 게임 '**kill demons**' 제작

오리지널 게임 '**PocketBall 알까기**' 제작

게임에 다양한 물리 현상을 접목.
위치, 속도, 가속도, 질량, 마찰, 가해지는 힘을 고려하여 직접 구현한 물리 엔진을 이용한 프로젝트들을 소개.

Physics Project Series – 황소 날리기



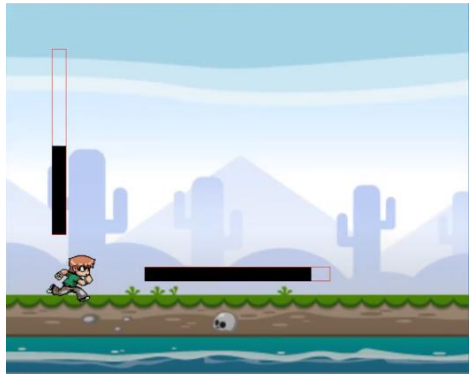
개발 목적	중력에 의한 등가속도 운동 구현
개발 툴	Pycharm
개발 언어	python
팀원	1인
개발 중점 사항	시차 스크롤링 기법을 이용한 배경의 원근감 표현 중력에 영향을 받아 자유 낙하 운동하는 객체 구현

캐릭터를 멀리 날리는 것이 목표인 게임.
적을 밟으면 X축 속도가 느려지지 않는다.
폭탄을 밟으면 X축 속도가 빨라지고 보다 높이 점프한다.
바닥에 닿으면 X축 속도가 느려짐.
스페이스바를 누르면 급속도로 땅으로 향해 돌진한다. 타이밍에 맞게
스페이스바를 누른다면 적을 연속해서 밟아 속도가 느려지지 않고 캐릭터를
멀리 날려보낼 수 있다.
속도가 0이 되면 게임 오버가 된다.

(영상) https://youtu.be/YOvjCKqmDMo
(Github) https://github.com/JongWonCha/2D-Programming

Physics Project Series – 황소 날리기

대표 구현 기술

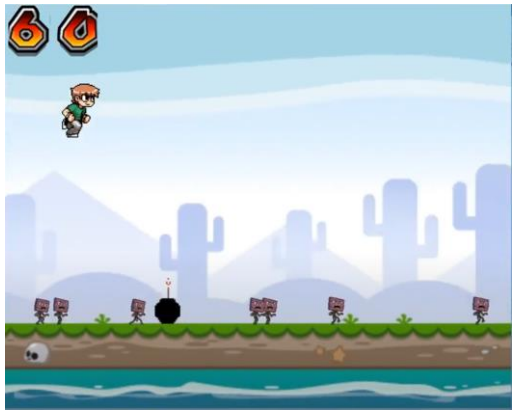


게임을 시작하기 전 게이지를 채워 가해지는 힘 x, y를 결정 후 날려보낸다.

```
elif event.type == SDL_KEYDOWN and event.key == SDLK_SPACE:
    power += 1
    if power == 2:
        global_state.x_acceleration = boy.x_acceleration
        global_state.y_accelertaion = boy.y_acceleration
        game_framework.change_state(main_state)
```

프레임 사이의 시간을 측정하여 이를 물리 현상에 적용함으로써 성능이 다른 컴퓨터에서도 같은 물리 현상이 일어나도록 최적화.

```
def update(self, frame_time):
    global SPEED, floor_enemy_speed, is_game_over, NORMALSTATE, HELISTATE, GRAVITIY, localmoney
    self.x += SPEED * frame_time
    if Boy.state == NORMALSTATE:
        self.y += self.acceleration * frame_time
        if self.acceleration > -2000:
            self.acceleration -= GRAVITIY * frame_time
        self.frame += 10 * frame_time
        if self.frame > 8: self.frame = 0
```



게임이 시작되면 매 프레임마다 중력에 영향을 받아 땅으로 떨어진다.

```
if Boy.state == NORMALSTATE:
    self.y += self.acceleration * frame_time
    if self.acceleration > -2000:
        self.acceleration -= GRAVITIY * frame_time
```

Front배경과 Back배경 객체를 따로 만들어 속도가 다르게 움직임으로써 시차 스크롤링 기법을 적용하였다.

```
front_background = None
back_background = None
```

```
front_background.update(frame_time)
back_background.update(frame_time)
```

Physics Project Series – Kill Demons

개발 목적	'아이작의 번제 ' 의 물리 엔진을 직접 구현하고 게임으로 만드는 것.
개발 툴	OpenGL, Visual studio 2017
개발 언어	C++
팀원	1인
개발 중점 사항	질량, 위치, 속도, 가속도, 가해지는 힘이 서로 상호작용하는 효과 구현. 마찰력 개념을 도입하여 물체가 서서히 속도가 감소하는 효과 구현. 객체의 속도 벡터를 분석하여 스프라이트가 쳐다보는 방향을 결정하는 기능 구현.

8방향의 적 스폰 구역에서 랜덤한 위치에 적이 스폰된다.
처음 스폰된 적은 체력이 낮아 죽이기 쉽지만 킬카운트가 올라갈 수록 체력이 높은 적이 생성된다.
투사체에 맞은 적은 투사체에 반대 방향으로 가해지는 힘을 받아 knockback된다.
플레이어의 체력이 다 소진되기 전에 60마리를 죽이면 클리어.
체력이 다 소진되면 게임 오버.

(영상) https://youtu.be/SUo_TAPqnH0
(Github) https://github.com/JongWonCha/killDemons

Physics Project Series – Kill Demons

대표 구현 기술

```
private:
	float m_PosX, m_PosY, m_PosZ;//position
	float m_SizeX, m_SizeY, m_SizeZ;//size
	float m_r, m_g, m_b, m_a;//color
	float m_VelX, m_VelY, m_VelZ;//velocity
	float m_AccX, m_AccY, m_AccZ;//accelaration
	float m_Mass;//mass
	float m_CoefFrict;//마찰계수
	int m_HP;//체력
	int m_State;//상태

	int m_kind;

	float m_InvinTime = 0; //무적시간, 0이 아니면 무적이다.
```

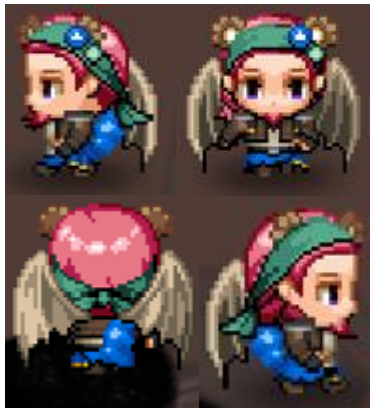
객체의 맴버 변수.
위치, 속도, 가속도,
마찰, 질량을 고려하여
물리 현상을 구현한다.

```
float friction = m_CoefFrict * m_Mass * GRAVITY;
if (magVel < FLT_EPSILON)//멈춰있을때 예외 처리,
{
	m_VelX = 0.f;
	m_VelY = 0.f;
	m_VelZ = 0.f;
}
else
{
	float accX = fricX / m_Mass;
	float accY = fricY / m_Mass;

	float afterVelX = m_VelX + accX * eTime;
	float afterVelY = m_VelY + accY * eTime;

	if (afterVelX * m_VelX < 0.f)
		m_VelX = 0.f;
	else
		m_VelX = afterVelX;
	if (afterVelY * m_VelY < 0.f)
		m_VelY = 0.f;
	else
		m_VelY = afterVelY;

	//gravity
	m_VelZ = m_VelZ - GRAVITY * eTime;
}
```

마찰력은 질량에 따라
힘이 달라진다.
객체의 속도 벡터에
반하는 힘을
지속적으로 줌으로써
마찰 현상을 구현.
속도 벡터가
입실론보다 작을 경우
마찰력을 주지 않고
강제로 속도 벡터를
0으로 set.

```
//cal Velocity
m_VelX = m_VelX + m_AccX * eTime;
m_VelY = m_VelY + m_AccY * eTime;
m_VelZ = m_VelZ + m_AccZ * eTime;
//cal Position
m_PosX = m_PosX + m_VelX * eTime;
m_PosY = m_PosY + m_VelY * eTime;
m_PosZ = m_PosZ + m_VelZ * eTime;
```

프레임 사이의 시간을
고려하여 업데이트.
성능이 다른
컴퓨터에서도 같은
물리 효과를 구현 할
수 있음.



```
if (abs(vel_x) >= abs(vel_y))
{
	if (vel_x < 0)
		Walk_Dir = 1;
	else
		Walk_Dir = 2;
}
else
{
	if (vel_y < 0)
		Walk_Dir = 0;
	else
		Walk_Dir = 3;
}
```

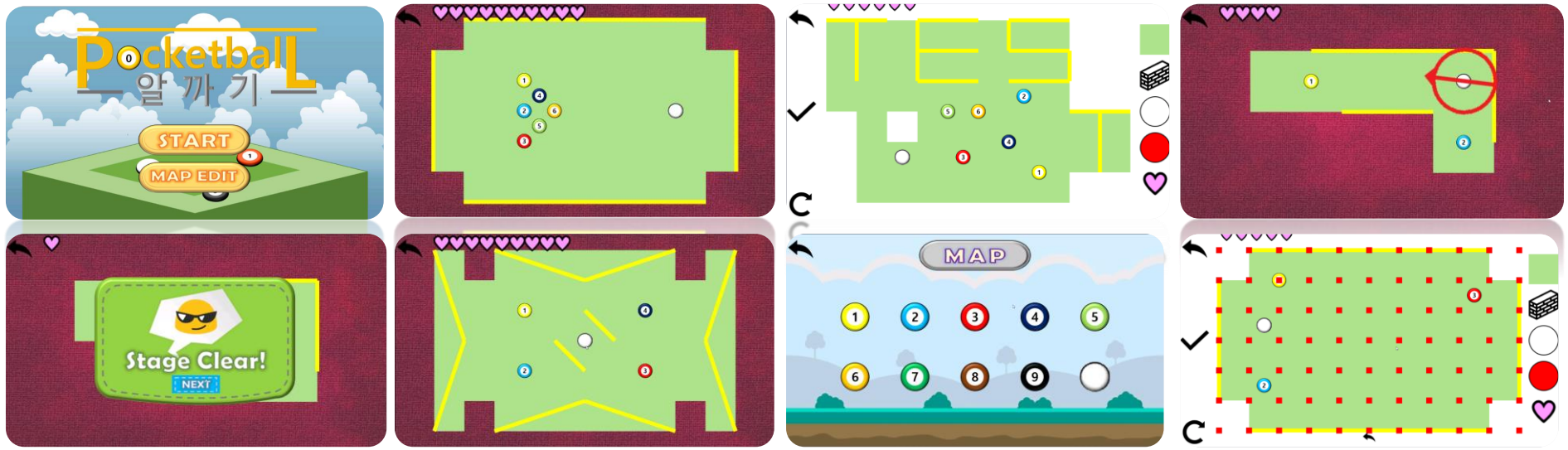
스프라이트가
바라보는 위치를
결정하는 간단한
코드.

```
quake_x = rand() % 100 - 50;
quake_y = rand() % 100 - 50;
temp = sqrtf(quake_x * quake_x + quake_y * quake_y);
quake_x /= temp;
quake_y /= temp;
quake_x *= invin * 5.f;
quake_y *= invin * 5.f;
```

```
m_Renderer->DrawTextureRectSeqXYHeight(50.f + quake_x, quake_y, 0, 50, 50, 1, 1, 1, 1, m_NumberTexture, g_Score % 10, 0, 10, 3, 0);
m_Renderer->DrawTextureRectSeqXYHeight(-50.f + quake_x, quake_y, 0, 50, 50, 1, 1, 1, 1, m_NumberTexture, g_Score / 10, 0, 10, 3, 0);
```

플레이어가 적에 맞을
시 quake 변수를
통하여 카메라가
흔들리는 효과 구현.

Physics Project Series – Pocketball 알까기



개발 목적	Kill Demons에서 구현하였던 움직임에 대한 연산 뿐 아니라 벽, 공과의 충돌 후 움직임을 실제와 같이 구현하려는 목적.
개발 툴	Android Studio
개발 언어	Java
팀원	3인
담당 역할	맵 저장 및 불러오기, 물리 구현
개발 중점 사항	파일 입출력을 통한 맵 에디터 기능 구현 공과 공의 충돌 후 가해지는 힘 부여. 공과 벽의 충돌 후 공에게 가해지는 힘 부여. 마우스 드래그, 릴리즈를 통하여 흰 공에 가해지는 힘 부여.

현실, 특히 당구의 물리 현상을 본떠 만든 모바일 게임.
흰 공을 클릭하고 원하는 방향으로 드래그 한 후 릴리즈 하면 흰 공이 특정 방향으로 나아간다.
흰 공으로 숫자 공을 쳐내서 움직이게 할 수 있다.
숫자 공을 1번 공부터 순서 대로 필드 밖으로 내보내야 한다.
시도 횟수를 모두 사용하거나 순서가 잘 못되면 스테이지 실패.
횟수 안에 순서대로 모든 숫자 공을 떨어뜨리면 스테이지 성공.

(영상) https://youtu.be/unQa8EOseX4
(Github) https://github.com/rlaghf555/Android_Game

Physics Project Series – Pocketball 알까기

물리 엔진 코드

```
public class Ball extends GraphicObject {
        public Rect m_rect;
        public int radius;
        public float m_VelX, m_VelY;
        public float m_AccX, m_AccY;
        public float m_CoefFrict;
        public boolean m_WallCollision;
        public int tile_i, tile_j;
        public int save_flag =0;
        public boolean draw= true;
        public boolean moving;
        //기타 등등 마찰, 어쩌구 저쩌구
        public Ball(Bitmap bitmap,int posx, int posy, int diameter) {   //위치 좌표, 지름
            super(bitmap);
            m_x = posx;
            m_y = posy;
            radius = diameter/2;
            m_rect = new Rect(m_x - radius, m_y - radius, m_x + radius, m_y + radius);
            m_CoefFrict = 15.f;
            m_WallCollision = false;
            draw= true;
            moving = false;
        }
```

공은 모두 같은 질량을 가지기에 질량을 맴버변수로 포함 시키지 않음.

마찰계수, 위치, 속도, 가속도를 고려하여 물리 현상을 구현.

```
public void UpDate(float eTime)
{
    float magVel = (float)Math.sqrt(m_VelX * m_VelX + m_VelY * m_VelY);
    float velX = m_VelX / magVel;
    float velY = m_VelY / magVel;
    float fricX = -velX;
    float fricY = -velY;

    float friction = m_CoefFrict * 9.8f;
    fricX *= friction;
    fricY *= friction;

    float epsilon = 1.f;

    if(magVel < epsilon)//epsilon
    {
        m_VelX = 0.f;
        m_VelY = 0.f;
        moving = false;
    }
    else
    {
        float afterVelX = m_VelX + fricX * eTime;
        float afterVelY = m_VelY + fricY * eTime;//eTime추가

        if(afterVelX * m_VelX < 0.f)
            m_VelX = 0.f;
        else
            m_VelX = afterVelX;
        if(afterVelY * m_VelY < 0.f)
            m_VelY = 0.f;
        else
            m_VelY = afterVelY;

    }
    m_VelX = m_VelX + m_AccX * eTime;
    m_VelY = m_VelY + m_AccY * eTime;
    //cal Position
    float afterPosX = (float)m_x + m_VelX * eTime;
    float afterPosY = (float)m_y + m_VelY * eTime;

    SetPosition((int)afterPosX, (int)afterPosY);
}
```

마찰력은 중력의 영향을 받음.

속도가 입실론 값 이하라면 마찰 연산을 중단한다.(부동 소수점 연산이라 부등호 연산)

프레임시간(eTime)을 고려하여 최적화.

Physics Project Series – Pocketball 알까기

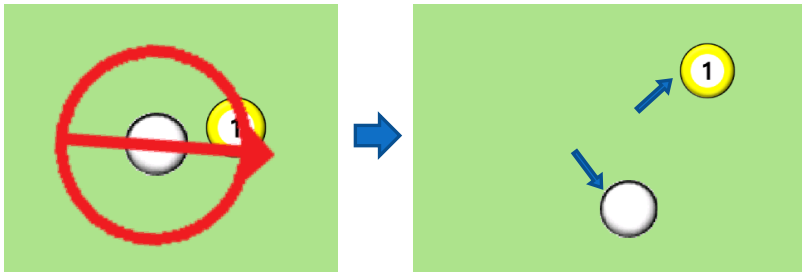
물리 충돌 구현

공과 공의 충돌

```
public void BallToBallCollision(Ball second, float elime)
{
    double len = Math.sqrt(Math.pow(this.m_x - second.m_x, 2) + Math.pow(this.m_y - second.m_y, 2));
    if(len < this.radius * 2)
    {
        float ForceX, ForceY;

        double v1X = ((double)this.m_VelX - (double)second.m_VelX) * 10.0;
        double v1Y = ((double)this.m_VelY - (double)second.m_VelY) * 10.0;
        double v2X = (double)second.m_x - (double)this.m_x;
        double v2Y = (double)second.m_y - (double)this.m_y;

        double DeltaNormalX = v2X / len;
        double DeltaNormalY = v2Y / len;
        float knockback = ((this.radius * 2) - (float)len) / 2.0f;

        int firstX = (int)((float)this.m_x + (-DeltaNormalX * knockback));
        int firstY = (int)((float)this.m_y + (-DeltaNormalY * knockback));
        int secondX = (int)((float)second.m_x + (DeltaNormalX * knockback));
        int secondY = (int)((float)second.m_y + (DeltaNormalY * knockback));

        this.SetPosition(firstX, firstY);
        second.SetPosition(secondX, secondY);

        double cos = ((v1X * v2X) + (v1Y * v2Y)) / (Math.sqrt(Math.pow(v1X, 2) + Math.pow(v1Y, 2)) * Math.sqrt(Math.pow(v2X, 2) + Math.pow(v2Y, 2)));
        double Force = Math.sqrt(Math.pow(second.m_VelX - this.m_VelX, 2) + Math.pow(second.m_VelY - this.m_VelY, 2));

        ForceX = (float)-DeltaNormalX * this.radius * 2 * (float)cos * (float)Force;
        ForceY = (float)-DeltaNormalY * this.radius * 2 * (float)cos * (float)Force;

        this.ApplyForce(ForceX, ForceY , 0.01f);
        second.ApplyForce(-ForceX, -ForceY, 0.01f);
    }
}
```

공이 충돌되어서 겹칠 경우 공이 서로 맞닿을 거리까지 강제로 위치를 조정.

조정 후 각 공에 가해지는 힘을 부여.



공과 벽의 충돌

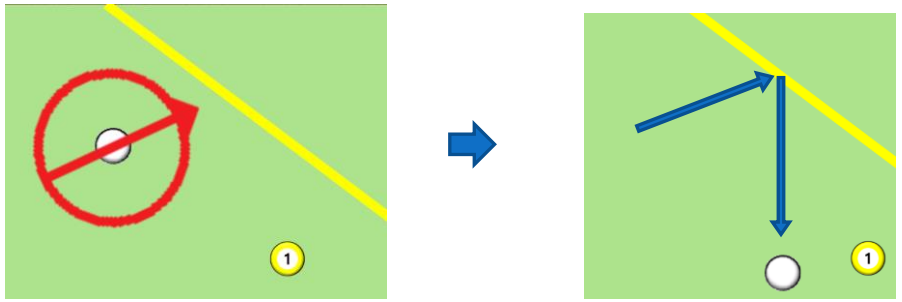
```
public boolean BallToWallCollision(Ball ball)
{
    double a, b, c ,A, B, R;
    double d;
    R = ball.radius;
    A = ball.GetX();
    B = ball.GetY();
    //(x - a)^2 + (y - b)^2 = r^2
    if(end.x == start.x) {//x = a꼴
        double T;
        T = end.x;
        a = 1.0;
        b = -2 * B;
        c = (T * T) - (2 * T * A) + (A * A) + (B * B) - (R * R);
        d = b * b - (4 * a * c);
        if(d < 0)
            return false;
        else {
            double yposp = (-b + Math.sqrt(d)) / (2 * a);
            double yposm = (-b - Math.sqrt(d)) / (2 * a);
            if(start.y > end.y) {
                if((end.y < yposp && yposp < start.y) || (end.y < yposm && yposm < start.y)) {
                    ApplyReflectVel(ball, 1.0, 0.0, -start.x);
                    return true;
                }
            }
            else {
                if((start.y < yposp && yposp < end.y) || (start.y < yposm && yposm < end.y)) {
                    ApplyReflectVel(ball, 1.0, 0.0, -start.x);
                    return true;
                }
            }
        }
    }
```

판별식을 통해 벽과 충돌하였는지 판단.

```
if(cos > 0)//노말 그대로 사용
{
    ReflectX = ball.m_VelX - 2 * (ball.m_VelX * normalX + ball.m_VelY * normalY) * normalX;
    ReflectY = ball.m_VelY - 2 * (ball.m_VelX * normalX + ball.m_VelY * normalY) * normalY;
    ball.SetPosition(ball.GetX() - (int)(normalX * length * 2.0), ball.GetY() -(int)(normalY * length* 2.0));
}
else if (cos <= 0)//노말 반대로 사용
{
    ReflectX = ball.m_VelX - 2 * (ball.m_VelX * -normalX + ball.m_VelY * -normalY) * -normalX;
    ReflectY = ball.m_VelY - 2 * (ball.m_VelX * -normalX + ball.m_VelY * -normalY) * -normalY;
    ball.SetPosition(ball.GetX() - (int)(-normalX * length* 2.0), ball.GetY() - (int)(-normalY * length* 2.0));
}
ball.SetVel((float)ReflectX, (float)ReflectY);
```

벽의 앞, 뒤에서 충돌 할 모든 경우를 고려.

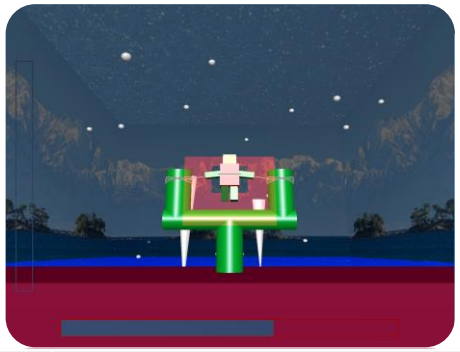
반사 벡터를 구하는 공식(R = v – 2(v · n)n을 사용하여 공의 속도 벡터 변수 업데이트.



other projects

이름	It’s mine
개발 툴	Visual Studio 2017
개발 언어	C++
개발 사항	TCP를 이용한 실시간 온라인 게임 구현.
아이템을 먹고 미사일을 날려 적의 체력을 소진 시키는 게임. (영상) https://youtu.be/GHeQNc9InJY (Github) https://github.com/natsnatsmon/AENG-GYUNG	

이름	아빠 어디가?(관광정보서비스)
개발 툴	Pycharm
개발 언어	python
개발 사항	공공 데이터 포털 API를 활용한 관광 어플리케이션 제작. 보고싶은 정보를 이메일, 텔레그램으로 받을 수 있도록 구현.
지역을 검색하면 그 지역에 있는 관광지의 정보들이 나오고, 그 정보들을 이메일과 텔레그램으로 볼 수 있는 어플리케이션. (영상) https://youtu.be/AqBnEZzaGZ8 (Github) https://github.com/JongWonCha/Script-Language	



이름	SlingShot
개발 툴	Visual Studio, OpenGL
개발 언어	C++
개발 사항	눈이 내리는 파티클 제작 Plane의 버텍스를 Sin함수로 이용하여 파도가 출렁이는 효과 제작 중력 가속도를 이용하여 포물선 운동을 하는 움직임 구현 마우스 드래그를 통한 카메라 시점 변환 구현
마우스를 이용하여 카메라 시점을 바꾸어 방향을 설정하고 키를 눌러 날려보낼 힘을 설정하면 날아간다. 사이클론을 피하고 낙하산을 이용하여 무사히 과녁 안에 착지하면 클리어. (영상) https://youtu.be/pm0XSl27eXE (Github) https://github.com/JongWonCha/Slingshot	