1945 Strikers 모작

한융희

목차

1) 개발환경
2) 게임소개
3) 프로그래밍 이슈 설명 목차
4) 프로그래밍 이슈 설명
5) 시연 영상

개발환경(도구)

비주얼 스튜디오 2017 Communinty

개발환경(일정)

총 4주

2020-12-28 ~ 2021-01-22

1주차 : 프로토타입

2주차 : 알파버전

3주차 : 베타버전

4주차 : 최종버전

게임 소개

제목 : 1945 Strikers 모작

장르 : 슈팅 게임

플랫폼 : PC

플레이 방법 : 화살표를 이용해 이동하고 스페이스로 탄환을

발사하며 아이템을 먹어 주인공 기체를 강화해가며

적기를 격추시킨다.

주요 프로그래밍 이슈 목차

1. 객체의 클래스화
2. 매크로 상수
3. enum문을 이용한 유한상태기개
4. 클리핑 처리
5. 무한스크롤을 이용한 배경화면
6. 함수의 재귀호출
7. 벡터를 이용한 탄환 방향조정
8. 객체간의 충돌표현

1. 객체의 클래스화

1.객체들을 클래스로 나누어 관리하기 용이하게 했다.

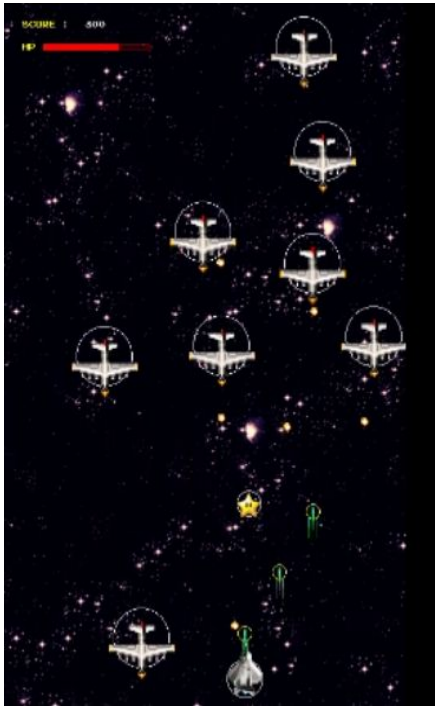
#include "olcPixelGameEngine.h"
#include "CScrollBg.h"
#include "CUnit.h"
#include "CActor.h"
#include "CBullet.h"
#include "CEnemy.h"
#include "CItem.h"

```
CScrollBg mBg_0;                    // 배경
CActor mActor;                      // 주인공 기체
CEnemy mEnemy[AMOUNT_ENEMY];        // 적기(잡몹)
CItem mItem[AMOUNT_ITEM];           // 아이템
CEnemy mBoss;                       // 적기(보스)
```

```
CBullet mBullet_Actor[BULLET_AMOUNT_ACTOR];
CBullet Bullet_Enemy[AMOUNT_ENEMY][BULLET_AMOUNT_ENEMY];
CBullet mBullet_Boss_left[BULLET_AMOUNT_BOSS];
CBullet mBullet_Boss_right[BULLET_AMOUNT_BOSS];
```

CActor 헤더파일의 클래스 예시

```
#include "olcPixelGameEngine.h"
#include "CUnit.h"

class CBullet; // 클래스 전방선언

class CActor : public CUnit
{
private:
        olc::Sprite *mpActorMoveSprite[7] = { nullptr };
        int nActorMoveIndex = 3; // 초기 스프라이트 인덱스
        int m_nActorDamage = 20;
        int m_nActorLife = 100;
        int m_nActorScore = 0;
        int m_nActorBulletPower = 0;
        bool m_IsActorAlive = true;

public:
        virtual void BuildInfo(float fX, float fY, float fSpeed, int nRadius, int nActorLife);
        virtual void DoFire(CBullet CBullet[]) override;

        virtual void UpdateMoveLeft(float fElpsedTime) override;
        virtual void UpdateMoveRight(float fElpsedTime) override;
        virtual void UpdateMoveUp(float fElpsedTime) override;
        virtual void UpdateMoveDown(float fElpsedTime) override;

        int GetActorLife();
        int GetActorDamage();
        int GetActorScore();
        int GetActorBulletPower();
        bool GetActorIsAlive();

        //void DisplayMoveActor(olc::PixelGameEngine *tpDc, olc::Sprite *mpActorMoveSprite, int nMoveIndex);
        void DisplayMoveActor(olc::PixelGameEngine *tpDc);
        void DisplayActorLife(olc::PixelGameEngine * tpDc);
        void SetActorLife(int nActorLife);
        void SetActorScore(int nScore);
        void SetActorIsAlive(bool IsAlive);
        void SetActorBulletPower(int nBulletPower);
        void SetActorDamage(int nDamage);

        void CreateActor();
        void DestroyActor();

        CActor();
        ~CActor();
};
```



객체 : 총알, 적기, 주인공 기체, 아이템, 배경 등

2. 매크로 상수

2.매크로 상수를 써 코드관리를 편하게하고 가독성을 높였다.

```
#define BOSS_Y 45.0f
#define MAPSIZE_ENEMY_Y -1600
#define RADIUS_ENEMY 33
#define BULLET_TIME_ENEMY 2.0f
#define BULLET_TIME_BOSS 0.5f
#define BULLET_TIME_BOSS_2 0.04f
#define BULLET_AMOUNT_ACTOR 100
#define BULLET_AMOUNT_ENEMY 100
#define BULLET_AMOUNT_BOSS 100
#define AMOUNT_ENEMY 15
#define AMOUNT_ITEM AMOUNT_ENEMY
#define DISPLAY_STAGE_FRAME 30
#define SPEED_ENEMY 100.0f

#define BULLET_ANGLE_ENEMY 1
#define BULLET_BOSS_ANGLE 5.0f
```

사용법을 예로들면 매크로 상수의 (AMOUNT_ENEMY)의 값(현재:15)만 바꿔주면
게임 스테이지당 나오는 적기의 수를 조절할수 있다.

CEnemy mEnemy[AMOUNT_ENEMY];// 적기(잡몹)

또는 , (BULLET_TIME_ENEMY) 의 값을 바꿔 적기의 총알 발사간격을 조절할수있다.

```
for (int i = 0; i < AMOUNT_ENEMY; i++)
{
	// 적기가 화면안에 있을때만 Dofire()
	if ((mEnemy[i].mX >= 0) && (mEnemy[i].mX <= 480) && (mEnemy[i].mY >= 0))
	{
		// (BULLET_TIME_ENEMY - fBulletTimeMinus)초 간격으로 탄환 발사
		// fBulletTimeMinus는 스테이지가 오를수록 커짐 --> 적 탄환발사 간격이 짧아짐
		if (mEnemy[i].GetTimeTick() >= BULLET_TIME_ENEMY - fBulletTimeMinus)
		{
		// (BULLET_TIME_ENEMY - fBulletTimeMinus)초마다 탄환을 발사하는구문.
			mEnemy[i].DoFire(mBullet_Enemy, &mActor, i);
			mEnemy[i].SetTimeTick(0.0f);
		}

		else
		{
			// 프레임당 시간 (실제 시간)을 누적해간다.
		mEnemy[i].SetTimeTick(mEnemy[i].GetTimeTick() + fElapsedTime);
		}
	}
}
```

3. enum문을 이용한 유한상태기개

3. enum문을 이용해 상태기개를 만들고 전이조건등을 설정해 씬화면과 스테이지 관리를 편리하게 했다.

```
enum SCENE
{
	TITELE = 0,
	READY_PLAYGAME,
	PLAYGAME,
	CLEAR,
	ENDGAME
};

enum STAGE
{
	STAGE_1 = 1,
	STAGE_2,
	STAGE_3,
	STAGE_4,
	STAGE_BOSS
};

SCENE mSceneState;
STAGE mStageState = STAGE::STAGE_1;
```

```
bool OnUserUpdate(float fElapsedTime) override
{
	switch (mSceneState)
	{
	case TITELE:
		mBg_0.Display_Start_Bg(); // 시작 배경 Display

		if (GetKey(olc::Key::SPACE).bReleased)
		{
			mSceneState = SCENE::READY_PLAYGAME;
		}

		if (GetKey(olc::Key::ESCAPE).bReleased)
		{
			mSceneState = SCENE::ENDGAME;
		}

		break;

	case READY_PLAYGAME:
		CreatePlayGame(); // 게임 데이터 설정
		mSceneState = SCENE::PLAYGAME;
		break;

	case PLAYGAME: // 게임중
	if (mActor.GetActorIsAlive() == true)
	{
	SetPixelMode(olc::Pixel::MASK);

	UpdatePlayGame(fElapsedTime); // 게임 구현부

	SetPixelMode(olc::Pixel::NORMAL);
```

```
if (GetKey(olc::Key::ESCAPE).bReleased)
{
		mSceneState = SCENE::ENDGAME;
}
}

// 주인공이 죽으면 R을 누르면 데이터를 초기화하고 다시시작
// ESC를 누르면 종료
else
{
		DrawString(ScreenWidth() / 5 - 75.0f, ScreenHeight() / 2, "Retry R / Exit ESC
if (GetKey(olc::Key::R).bReleased)
{
		ResetGame();
		mSceneState = SCENE::READY_PLAYGAME;
}

if (GetKey(olc::Key::ESCAPE).bReleased)
{
		mSceneState = SCENE::ENDGAME;
}
}
break;

case CLEAR: // 보스를 죽이고 게임 클리어시
GameClear();
break;

case ENDGAME: // 게임 종료
ExitGame();
break;
}
		return true;
}
```

3. 배열을 이용한 애니메이션 구현

3.스프라이트를배열로 생성해 애니메이션을구현했다.

```
class CActor : public CUnit
{
private:
		olc::Sprite *mpActorMoveSprite[7] = { nullptr };
		int nActorMoveIndex = 3; // 초기 스프라이트 인덱스

CActor::CActor()
{
		mpActorMoveSprite[0] = new olc::Sprite("resources/test/tile000.png");
		mpActorMoveSprite[1] = new olc::Sprite("resources/test/tile002.png");
		mpActorMoveSprite[2] = new olc::Sprite("resources/test/tile004.png");
		mpActorMoveSprite[3] = new olc::Sprite("resources/test/tile006.png");
		mpActorMoveSprite[4] = new olc::Sprite("resources/test/tile008.png");
		mpActorMoveSprite[5] = new olc::Sprite("resources/test/tile010.png");
		mpActorMoveSprite[6] = new olc::Sprite("resources/test/tile012.png");
}

// 주인공 기체가 움직일때마다 스프라이트를 다르게해 애니메이션 효과를 냈다.
void CActor::DisplayMoveActor(olc::PixelGameEngine * tpDc)
{
		tpDc->DrawSprite(mX - mRadius - 5.0f, mY - 50.0f, mpActorMoveSprite[nActorMoveIndex], 2);
}

void CActor::UpdateMoveLeft(float fElpsedTime)
{
		mX = mX - mSpeed * fElpsedTime;

		if (nActorMoveIndex > 0)
		{
			//cout << "nActorMoveIndex = " << nActorMoveIndex << endl;
			nActorMoveIndex--;
		}

		else
		{
			//cout << "nActorMoveIndex = " << nActorMoveIndex << endl;
			nActorMoveIndex = 0;
		}
}
```

[0] [1] [2] [3] [4] [5] [6]

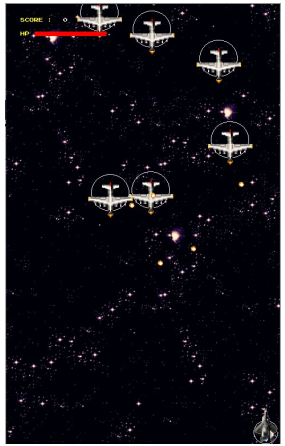
주인공 기체가 좌우 움직일때마다 **nActorMoveIndex**의 값을 바꿔줘서 출력되는 스프라이트를 다르게 한다.

4. 클리핑 처리

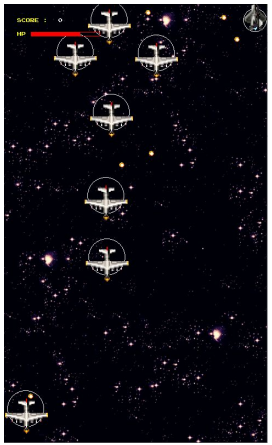
4. 주인공 기체를 클리핑 처리해 화면밖으로 못나가게 했다.

```
void CUnit::Clipping(float fElpsedTime)
{
	if (CUnit::mX - mRadius <= 0) // 왼쪽 끝
	{
		mX = mX + mSpeed * fElpsedTime;

		if (CUnit::mY - mRadius <= 0) // 위 끝
		{
			mY = mY + mSpeed * fElpsedTime;
		}

		if (CUnit::mY + mRadius >= mpPGE->ScreenHeight()) // 아래 끝
		{
			mY = mY - mSpeed * fElpsedTime;
		}
	}
	else if (CUnit::mX + mRadius >= mpPGE->ScreenWidth()) // 오른쪽 끝
	{
		mX = mX - mSpeed * fElpsedTime;

		if (CUnit::mY - mRadius <= 0) // 위 끝
		{
			mY = mY + mSpeed * fElpsedTime;
		}

		if (CUnit::mY + mRadius >= mpPGE->ScreenHeight()) // 아래 끝
		{
			mY = mY - mSpeed * fElpsedTime;
		}
	}
	else if (CUnit::mY - mRadius <= 0) // 위 끝
	{
		mY = mY + mSpeed * fElpsedTime;
	}

	else if (CUnit::mY + mRadius >= mpPGE->ScreenHeight()) // 아래 끝
	{
		mY = mY - mSpeed * fElpsedTime;
	}
}
```

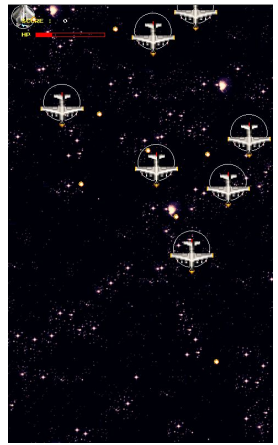
아래-왼쪽



아래-오른쪽



위-오른쪽



위-왼쪽

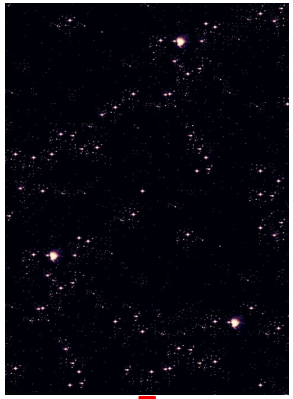
주인공 기체가 화면밖으로 나가려한다면 반대방향으로 이동시켜 기체가 멈추게 하는 효과를 냄

5. 무한스크롤을 이용한 배경화면

5. 무한스크롤 개념을 이용해 두개의 이미지로 무한한 배경이미지를 구현했다.

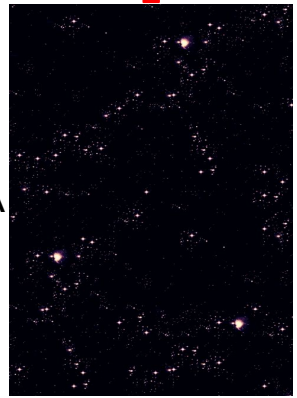
```
void CScrollBg::CreateR(float tScrollSpeed)
{
		mpSpriteA = new olc::Sprite("resources/back01.png");
		mpSpriteB = new olc::Sprite("resources/back01.png");

		//
		mA_Y = 0.0f;
		mB_Y = -800.0f;
		mHeight = 800.0f;
		mScrollSpeed = tScrollSpeed;
}

void CScrollBg::UpdateR(float tDelta)
{
		// update
		mA_Y = mA_Y + mScrollSpeed * tDelta;
		mB_Y = mB_Y + mScrollSpeed * tDelta;

		// 저글링
		if (mA_Y >= mHeight)
		{
			mA_Y = - mHeight + (mA_Y - mHeight);
		}

		if (mB_Y >= mHeight)
		{
			mB_Y = - mHeight + (mB_Y - mHeight);
		}
}

void CScrollBg::Display_Start_Bg()
{
		mpPGE->DrawSprite(0, mA_Y, mpSpriteA);
		mpPGE->DrawSprite(0, mB_Y, mpSpriteB);

		mpPGE->Clear(olc::BLACK);

		mpPGE->SetPixelMode(olc::Pixel::MASK);

		mpPGE->DrawSprite(-20, -40, pSpriteTitleBG, 2);
		mpPGE->DrawSprite(mpPGE->ScreenWidth() / 7, mpPGE->ScreenHeight() / 4, pSpriteTitleSTRING, 2);
		mpPGE->DrawString(mpPGE->ScreenWidth() / 3 - 5, mpPGE->ScreenHeight() / 2 + 170, "PRESS SPACE",
olc::WHITE, 2);

		if (nButton <= 1)
		{
			mpPGE->DrawSprite(mpPGE->ScreenWidth() / 3, mpPGE->ScreenHeight() / 2 + 120, pSpriteTitleButton[0],
2);
			nButton++;
		}

		else if (nButton > 1)
		{
			mpPGE->DrawSprite(mpPGE->ScreenWidth() / 3 - 2, mpPGE->ScreenHeight() / 2 + 120,
pSpriteTitleButton[1], 2);
			nButton--;
		}

		mpPGE->SetPixelMode(olc::Pixel::NORMAL);
}

void CScrollBg::Display_Game_Bg()
{
		mpPGE->DrawSprite(0, mA_Y, mpSpriteA);
		mpPGE->DrawSprite(0, mB_Y, mpSpriteB);
}
```

배경B



배경이미지의Y축 좌표가 화면의 크기(800) 만큼 이동하면Y축 좌표를 반대로 바꿔서 저글링 하듯이 배경화면A,B가 전환된다.

배경A



6. 함수의 재귀호출

6. 재귀적으로 함수를 호출해 적기체의 시작위치를 재조정했다.

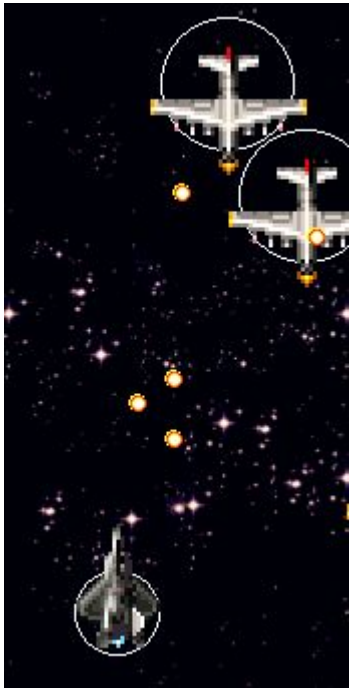
```
// 겹쳐지는 적기체 위치조정 (겹치는 경우가 있는경우 계속 위치조정)
// + 적기체 초기화
while (true == CollisionEnemytoEnemyAdjust())
{
	CollisionEnemytoEnemyAdjust();
}
bool CollisionEnemytoEnemyAdjust()
{
	float fRandXvalue = 0; // 적기의 x축 위치
	float fRandYvalue = 0; // 적기의y축 위치

	bool IsCollison_EE = false;

	for (int i = 0; i < AMOUNT_ENEMY; i++)
	{
		mEnemy[i].SetEnemySpriteIndex(0);
		mEnemy[i].mDirX = 1.0f; // 방향 아래로
		mEnemy[i].mIsAlive = true;
		mEnemy[i].SetIsAliveEnemy(true);
		mEnemy[i].SetEnemyHP(100);
		mEnemy[i].SetRandomItemValue(-1);

		// 적기체 x좌표 y좌표 랜덤설정
		fRandXvalue = rand() % ScreenWidth() + RADIUS_ENEMY;
		fRandYvalue = rand() % MAPSIZE_ENEMY_Y;

		// 적기체가 화면 x축 밖이랑 겹치면 적기체의 x좌표를 화면 안으로 재조정
		if (fRandXvalue + RADIUS_ENEMY >= ScreenWidth())
		{
			fRandXvalue -= 2 * (RADIUS_ENEMY);
		}

		mEnemy[i].BuildInfo(fRandXvalue, -fRandYvalue, SPEED_ENEMY, RADIUS_ENEMY);

for (int k = 0; k < AMOUNT_ENEMY; k++)
{
		if (i != k) // 자기 자신의 충돌원은 제외
{
		// 루트함수써도 되지만 연산비용이 높아서 제곱으로 함
		float tEEAdd = (RADIUS_ENEMY + RADIUS_ENEMY)*(RADIUS_ENEMY + RADIUS_ENEMY);
		float tEEDistance = (mEnemy[i].mX - mEnemy[k].mX) * (mEnemy[i].mX - mEnemy[k].mX)
		+ (mEnemy[i].mY - mEnemy[k].mY) * (mEnemy[i].mY - mEnemy[k].mY);

		// 적기끼리 충돌원이 겹치는 경우
		if (tEEDistance <= tEEAdd)
		{
			IsCollison_EE = true;
			//cout << "collision_ETOE --> 위치 재조정" << endl;

			// 적기체 x좌표 y좌표 랜덤설정
			fRandXvalue = rand() % ScreenWidth() + RADIUS_ENEMY;
			fRandYvalue = rand() % MAPSIZE_ENEMY_Y;

			// 적기체가 화면 x축 밖이랑 겹치면 적기체의 x좌표를 화면 안으로 재조정
			if (fRandXvalue + RADIUS_ENEMY >= ScreenWidth())
			{
			fRandXvalue -= 2 * (RADIUS_ENEMY);
			}
mEnemy[i].BuildInfo(fRandXvalue, -fRandYvalue, SPEED_ENEMY, RADIUS_ENEMY); }}}} return IsCollison_EE;}
```

적 기체의 시작 x,y좌표는 난수값으로 설정했는데, 난수가 발생하면서 적기체끼리 겹치거나 화면밖으로 나가는 경우가 생긴다. 이러한 경우, 난수를 재발생을 시켜 적기체의 x,y좌표를 재설정한다. 만약 설정된 x,y위치가 겹치거나 화면밖으로 나가지 않는다면 탈출조건 (IsCollison_EE == false)로 루프를 탈출한다.

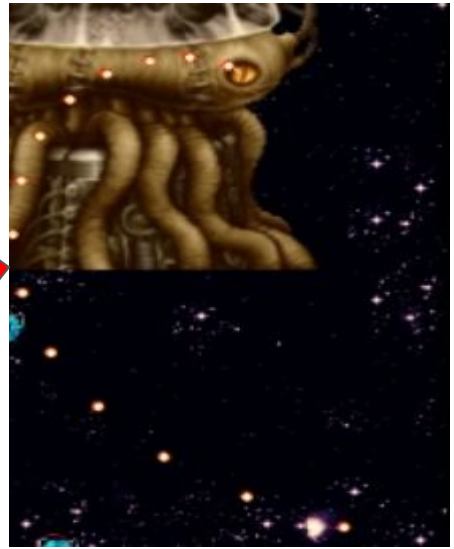
7. 벡터를 이용한 탄환 방향조정

7. 벡터(기하학의)를 이용해 적기 탄환이 주인공 기체를 조준하게 한다.

```
void CEnemy::DoFire(CBullet CBullet[][100], CUnit * tpTarget, int nIndex)
{
	if (m_IsAliveEnemy == true)
	{
		//cout << "DoFire (ENEMY)" << endl;

		// 발사 시작 위치 지정
		if (nullptr == CBullet) return;
		CBullet[nIndex][mCurIndexBullet].mX = this->mX;
		CBullet[nIndex][mCurIndexBullet].mY = this->mY;

		// 조준한 방향을 구하기.
		// 방향 벡터 = (목적지점 - 시작지점) 정규화
		// 임의의 크기에 방향 벡터를 구함
		float tDirX = tpTarget->mX - this->mX;
		float tDirY = tpTarget->mY - this->mY;

		// 벡터의 크기를 구함
		float tMagnitude = std::sqrtf((tpTarget->mX - this->mX)*(tpTarget->mX - this->mX)
			+ (tpTarget->mY - this->mY)*(tpTarget->mY - this->mY));

		// 정규화 (해당 벡터의 크기를 1로 만든다)
		tDirX = tDirX / tMagnitude;
		tDirY = tDirY / tMagnitude;

		// 탄환의 방향을 설정한다.
		CBullet[nIndex][mCurIndexBullet].DirX = tDirX;
		CBullet[nIndex][mCurIndexBullet].DirY = tDirY;

		// 발사
		CBullet[nIndex][mCurIndexBullet].mIsAlive = true;

		if (mCurIndexBullet < BULLET_AMOUNT_BOSS - 1)
		{
			mCurIndexBullet++;
		}
		else
		{
			mCurIndexBullet = 0;	}}
}
```



삼각함수를
이용해
탄환발사방향을
일정각도마다로
할수 있다.

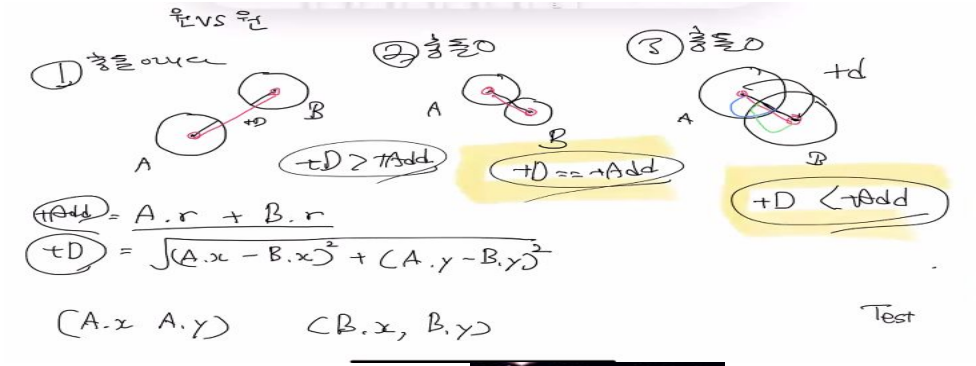


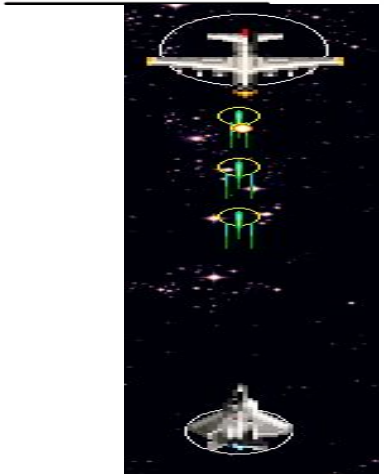
```
// 조준한 방향을 구하자.
float fDirX = 1.0f * std::cosf(fDirY_Plus * (3.14159f / 180.0f));
float fDirY = 1.0f * std::sinf(fDirY_Plus * (3.14159f / 180.0f));

// 10도씩 더해감
fDirY_Plus = fDirY_Plus + 10.0f;

// 탄환의 방향을 설정한다.
CBullet[mCurIndexBullet].DirX = fDirX;
CBullet[mCurIndexBullet].DirY = fDirY;
```

8. 객체간의 충돌표현

8. 원(객체)간의 중점사이 거리와 반지름을 이용해 원의 충돌을 표현했다.

```
void Collision_Setting()
{
// collision
for (int k = 0; k < AMOUNT_ENEMY; k++)
		{
		for (int i = 0; i < BULLET_AMOUNT_ACTOR; i++)
		{
			// 적기가 살아있는 경우만
			if (mEnemy[k].GetIsAliveEnemy() != false)
			{
			// 주인공 기체에서 나오는 탄환 충돌설정
			if (true == mBullet_Actor[i].mIsAlive)
			{
			// 루트함수써도 되지만 연산비용이 높아서 제곱으로 함
			float tAdd = (mBullet_Actor[i].mRadius + mEnemy[k].mRadius)*(mBullet_Actor[i].mRadius + mEnemy[k].mRadius);
			float tDistance = (mBullet_Actor[i].mX - mEnemy[k].mX) * (mBullet_Actor[i].mX - mEnemy[k].mX)
					+ (mBullet_Actor[i].mY - mEnemy[k].mY) * (mBullet_Actor[i].mY - mEnemy[k].mY);

			if (tDistance <= tAdd)
			{
			cout << "collision_ENEMY" << endl;

			mBullet_Actor[i].mIsAlive = false;

			// 적기가 주인공기체의 탄환에 맞으면
			// 적기의 현재체력 = 적기의 체력 - 주인공 기체의 공격력

			mEnemy[k].SetEnemyHP(mEnemy[k].GetEnemyHP() - mActor.GetActorDamage());
			//cout << "적 (" << k << ") 의 HP : " << mEnemy[k].GetEnemyHP() << endl;

			break;}}}}}
```





시연 영상

링크

https://youtu.be/JRGj2pEtx_E

마무리

감사합니다.