CircleShootor_3week 프로젝트 설명문서

작성자 이름: 강현구

제목: CircleShootor_3week

게임 소개

장르 – RPG
게임 플레이 – 1. 상(W), 하(S), 좌(A), 우(D) 이동,
2. 조준(마우스 좌 클릭) 탄환 모드 지원 – 마우스 커서 방향으로 발사
3. 적기 공격 2초에 1번
4. 적기 재생성 10초
5. 적기 이동 방향 재설정 1초
6. 게임 시작 – 스페이스바, 게임 중 리셋 – R , 게임 중 종료 – ESC
7. Enemy를 처치할 때마다 200점
8. Actor의 체력이 0이 되면 게임 오버

개발환경과 도구

1. 비주얼 스튜디오 community 2019

게임에 쓰인 프로그래밍 기술 주요 이슈

1. 마우스를 이용한 조준 사격

마우스 클릭을 통해 마우스 좌표를 가져와 벡터의 개념을 활용하여 탄환을 발사하게 하였습니다.

```
void cUser::DoFire(cBullet tpBullet[], sMouse_Point tMouse_Point)
		float tX_Size = mX - tMouse_Point.tX;
		float tY_Size = mY - tMouse_Point.tY;
		float tH_Size = sqrtf(powf(tX_Size, 2) + powf(tY_Size, 2));

		//시작점
		tpBullet[mCurIndexBullet].SetBulletPosition(mX - (mSize + tpBullet[mCurIndexBullet].GetSize()) * (tX_Size / tH_Size), mY - (mSize + tpBullet[mCurIndexBullet].GetSize()) * (tY_Size / tH_Size));


		//방향
		tpBullet[mCurIndexBullet].SetDirX(tMouse_Point.tX - this->mX);
		tpBullet[mCurIndexBullet].SetDirY(tMouse_Point.tY - this->mY);


		float tMagnitude = sqrtf(powf(tpBullet[mCurIndexBullet].GetDirX(), 2) + powf(tpBullet[mCurIndexBullet].GetDirY(), 2));


		tpBullet[mCurIndexBullet].SetDirX(tpBullet[mCurIndexBullet].GetDirX() / tMagnitude);
		tpBullet[mCurIndexBullet].SetDirY(tpBullet[mCurIndexBullet].GetDirY() / tMagnitude);
```

```
tpBullet[mCurIndexBullet].ShootFire();

if (mCurIndexBullet < BULLET_MAX - 1)

        mCurIndexBullet++;

else
        mCurIndexBullet = 0;
```

2. Enemy의 AI

Enemy는 살아있을 경우 불규칙적인 움직임을 가지며 일정 시간마다 정해진 패턴에 따라 사격합니다.
Enemy는 죽어있을 경우 일정 시간 후 부활합니다.

```
for (int ti = 0; ti < mLevel->GetUnitCount(); ti++)
        if (tEnemy[ti]->GetIsAlive())
                for (int tj = ti + 1; tj < mLevel->GetUnitCount(); tj++)
                        if (tEnemy[tj]->GetIsAlive())
                                if (powf(tEnemy[ti]->GetX() - tEnemy[tj]->GetX(), 2) + powf(tEnemy[ti]->GetY() -
tEnemy[tj]->GetY(), 2) <= powf(tEnemy[ti]->GetSize() + tEnemy[tj]->GetSize(), 2))
                                        tEnemy[ti]->ChangeAI(tEnemy[tj]);//Enemy끼리 충돌 시 방향 전환



                if (tEnemy[ti]->GetTimeTick() >= mLevel->GetBulletFireTime())

                        tEnemy[ti]->DoFire(tEnemyBullet[ti], tActor, mLevel->GetAccuracy());
                        tEnemy[ti]->SetTimeTick(0.0f);
                        //사격 및 사격 쿨타임

                else
                        tEnemy[ti]->SetTimeTick(tEnemy[ti]->GetTimeTick() + fElapsedTime);
                        //사격 쿨타임

                tEnemy[ti]->UpdateAI(fElapsedTime, tpCircle);
                //AI의 움직임

        else
                if (tEnemy[ti]->GetTimeTick() >= mLevel->GetResponeTime())

                        if (rand() % 100 > 70)
                                mEnemyState[ti] = DoFireCircle::CreateInstance();
                                tEnemy[ti]->SetState(mEnemyState[ti]);
                                tEnemy[ti]->SetUnitClolor(olc::GREEN);
                                //8방향 사격 Enemy

                        else
                                mEnemyState[ti] = DoFireAim::CreateInstance();
                                tEnemy[ti]->SetState(mEnemyState[ti]);
                                tEnemy[ti]->SetUnitClolor(olc::RED);
                                //조준 사격 Enemy


                        tEnemy[ti]->SetUnitSize(UNIT_SIZE);
                        tEnemy[ti]->SetUnitPosition(rand() % (tpCircle->ScreenWidth() - 2 * UNIT_SIZE) + UNIT_SIZE, rand()
% (tpCircle->ScreenHeight() - 2 * UNIT_SIZE) + UNIT_SIZE);
                        tEnemy[ti]->SetVelocity(50.0f);
                        tEnemy[ti]->SetHP(3);

                        tEnemy[ti]->SetIsAlive(true);

                        tEnemy[ti]->SetTimeTick(0.0f);
                        //Enemy 정보의 초기화

                else
                        tEnemy[ti]->SetTimeTick(tEnemy[ti]->GetTimeTick() + fElapsedTime);
                        //Enemy의 부활 시간
```

==========

Enemy의 움직임을 설정하는 함수 - AIMODE에 관한 정보는 DEFINE.h에 저장되어있습니다.

```
void cEnemy::UpdateAI(float fElapsedTime, cCircleShootor* tpCircle)
	if (mIsAlive)
		if (mAI == AIMODE::STOP)
			mAI = AIMODE::XX;
			SetTheta();
			mTimeTickAI = 0.0f;


		if (mTimeTickAI < 1.0f)
			mTimeTickAI += fElapsedTime;

		else
			mAI = AIMODE::XX;
			SetTheta();
			mTimeTickAI = 0.0f;


		if (this->GetY() <= this->GetSize())
			if (this->GetX() <= this->GetSize() && mAI != AIMODE::UL)
				mAI = AIMODE::UL;
				SetTheta();
				mTimeTickAI = 0.0f;


			else if (this->GetX() >= tpCircle->ScreenWidth() - this->GetSize() && mAI != AIMODE::UR)
				mAI = AIMODE::UR;
				SetTheta();
				mTimeTickAI = 0.0f;

			else if (this->GetX() < tpCircle->ScreenWidth() - this->GetSize() && GetX() > this->GetSize() && mAI !=
AIMODE::UX)
				mAI = AIMODE::UX;
				SetTheta();
				mTimeTickAI = 0.0f;


		else if (this->GetY() >= tpCircle->ScreenHeight() - this->GetSize() && mAI != AIMODE::DL)
			if (this->GetX() <= this->GetSize())
				mAI = AIMODE::DL;
				SetTheta();
				mTimeTickAI = 0.0f;

			else if (this->GetX() >= tpCircle->ScreenWidth() - this->GetSize() && mAI != AIMODE::DR)
				mAI = AIMODE::DR;
				SetTheta();
				mTimeTickAI = 0.0f;

			else if (this->GetX() < tpCircle->ScreenWidth() - this->GetSize() && this->GetX() > this->GetSize() && mAI
!= AIMODE::DX)
				mAI = AIMODE::DX;
				SetTheta();
				mTimeTickAI = 0.0f;


		else if (this->GetY() < tpCircle->ScreenHeight() - this->GetSize() && this->GetY() > this->GetSize() )
			if (this->GetX() <= this->GetSize() && mAI != AIMODE::XL)
				mAI = AIMODE::XL;
				SetTheta();
				mTimeTickAI = 0.0f;

			else if (this->GetX() >= tpCircle->ScreenWidth() - this->GetSize() && mAI != AIMODE::XR)
				mAI = AIMODE::XR;
				SetTheta();
				mTimeTickAI = 0.0f;

			else if (this->GetX() < tpCircle->ScreenWidth() - this->GetSize() && GetX() > this->GetSize() && mAI !=
AIMODE::XX)



		this->UpdateMoveRight(fElapsedTime * cosf(PI * mTheta / 180.0f));
		this->UpdateMoveDown(fElapsedTime * sinf(PI * mTheta / 180.0f));
```

```
=============================================================================
Enemy끼리의 충돌 시 반대 각도로 이동하게 하는 함수
void cEnemy::ChangeAI(cEnemy* tEnemyB)
	int tHorizontal_A;
	int tVertical_A;
	int tHorizontal_B;
	int tVertical_B;


	int tTable[3][3] =	AIMODE::UL, AIMODE::UX, AIMODE::UR,
								AIMODE::XL, AIMODE::XX, AIMODE::XR,
								AIMODE::DL, AIMODE::DX, AIMODE::DR ;

	if (this->mX - tEnemyB->mX == 0)			//A와 B의 X 좌표가 동일
		tHorizontal_A = 1;
		tHorizontal_B = 1;

	else if (this->mX - tEnemyB->mX > 0)			//A와 B의 X 좌표가 A가 B보다 크다.
		tHorizontal_A = 2;
		tHorizontal_B = 0;

	else
		tHorizontal_A = 0;					//A와 B의 X 좌표가 A가 B보다 작다.
		tHorizontal_B = 2;


	if (this->mY - tEnemyB->mY == 0)			//A와 B의 Y 좌표가 동일
		tVertical_A = 1;
		tVertical_B = 1;

	else if (this->mY - tEnemyB->mY > 0)		//A와 B의 Y 좌표가 A가 B보다 크다.
		tVertical_A = 2;
		tVertical_B = 0;

	else
		tVertical_A = 0;					//A와 B의 Y 좌표가 A가 B보다 작다.
		tVertical_B = 2;


	if (tHorizontal_A == 1 && tVertical_A == 1)
		tHorizontal_A = rand() % 3;
		tVertical_A = rand() % 3;

		tHorizontal_B = rand() % 3;
		tVertical_B = rand() % 3;

		this->mAI = tTable[tHorizontal_A][tVertical_A];
		this->SetTheta();
		tEnemyB->mAI = tTable[tHorizontal_B][tVertical_B];
		tEnemyB->SetTheta();

	else
		this->mAI = tTable[tHorizontal_A][tVertical_A];
		this->SetTheta();
		tEnemyB->mAI = tTable[tHorizontal_B][tVertical_B];
		tEnemyB->SetTheta();


	this->mTimeTickAI = 0.0f;
	tEnemyB->mTimeTickAI = 0.0f;


=============================================================
Enemy의 이동 방향(각도)를 설정하는 함수
void cEnemy::SetTheta()

	switch (mAI)
	case AIMODE::UL:

		mTheta = rand() % 90;

	break;
	case AIMODE::UX:

		mTheta = rand() % 180;
```

```
		break;
		case AIMODE::UR:

			mTheta = rand() % 90 + 90;


		break;
		case AIMODE::XL:

			mTheta = rand() % 180 - 90;

		break;
		case AIMODE::XX:

			mTheta = rand() % 360;

		break;
		case AIMODE::XR:

			mTheta = rand() % 180 + 90;

		break;
		case AIMODE::DL:

			mTheta = rand() % 90 + 270;

		break;
		case AIMODE::DX:

			mTheta = rand() % 180 + 180;

		break;
		case AIMODE::DR:

			mTheta = rand() % 90 + 180;

		break;
		case AIMODE::STOP:



		break;
```

3. 상태 패턴을 통해 Enemy의 사격 방식의 상태를 설정

상태 패턴을 사격 방식의 추가, 삭제 및 수정이 쉬워지며, 코드의 가독성이 올라간다.

```
============================================================
Enemy의 상태 패턴
class cEnemy : public cUnit
private:
		= 생략 =
		cState* mpState = NULL;

public:
		= 생략 =

		cEnemy()  mpState = DoFireAim::CreateInstance();

		void SetState(cState* tpState)

			mpState = tpState;


		void DoFire(cBullet tpBullet[], cUnit* tpTarget, int tAccuracy)
			mpState->SetEnemy(this);
			mpState->DoFire(tpBullet, tpTarget, tAccuracy);

;


============================================================
State 클래스를 통한 가상함수 정의
```

```
class cState

protected:
	cEnemy* mEnemy = nullptr;

public:

	static cState* CreateInstance()  return NULL;

	void SetEnemy(cEnemy* tEnemy)
		mEnemy = tEnemy;


	virtual void DoFire(cBullet tpBullet[], cUnit* tpTarget, int tAccuracy) = 0;

protected:
	cState() ;
	virtual ~cState() ;
;
```

==========

상태 클래스

```
class DoFireAim : public cState
private:
	static DoFireAim* mpInstance;

protected:
	DoFireAim()
	virtual ~DoFireAim()

public:
	static DoFireAim* CreateInstance()

		if (NULL == mpInstance) mpInstance = new DoFireAim();

		return mpInstance;


	virtual void DoFire(cBullet tpBullet[], cUnit* tpTarget, int tAccuracy) override;
;

class DoFireCircle : public cState
private:
	static DoFireCircle* mpInstance;

protected:
	DoFireCircle()
	virtual ~DoFireCircle()

public:
	static DoFireCircle* CreateInstance()

		if (NULL == mpInstance) mpInstance = new DoFireCircle();

		return mpInstance;


	virtual void DoFire(cBullet tpBullet[], cUnit* tpTarget, int tAccuracy) override;
;
```

4. 동적 할당을 사용하여 게임 종료 후 불필요하게 사용되는 메모리를 거의 없앴습니다.

게임의 타이틀 SCENE에서 로딩SCENE에서 사용될 변수들을 동적 할당하여 게임 플레이 SCENE에서 게임이 진행된다.
게임이 종료되는 시점에 동적 할당 되어있는 변수들을 동적 할당을 해제하는 것으로 게임 종료 후 불필요하게 사용되는 메모리를 거의 없앱니다.

5. SCENE 구현

SCENE을 활용하여 게임 상태를 구분하였습니다.

```
switch (mSceneState)
	case SCENE::TITLE:

		mGameTitle.UpdateTitle(fElapsedTime, this);
		mGameTitle.DisplayTitle(this);

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


	break;
	case SCENE::READY_PLAYGAME:

		mGamePlay.CreatePlayGame(this);
		mGameTitle.DestroyTitle();
		mSceneState = SCENE::PLAYGAME;

	break;
	case SCENE::PLAYGAME:

		if (GetKey(olc::Key::ESCAPE).bReleased)
			mSceneState = SCENE::TITLE;
			mGamePlay.DestroyPlayGame();
			mGameTitle.CreateTitle(this);

		else if (GetKey(olc::Key::R).bReleased)
			mGamePlay.DestroyPlayGame();

			mGamePlay.CreatePlayGame(this);

		else
			mGamePlay.UpdatePlayGame(fElapsedTime, this);
			mGamePlay.DisplayPlayGame(this);


		if (mGamePlay.GetGameOver())
			mGameEnd.CreateTitle(this, mGamePlay.GetScore());
			mGamePlay.DestroyPlayGame();
			mSceneState = SCENE::ENDGAME;


	break;
	case SCENE::ENDGAME:

		mGameEnd.UpdateTitle(fElapsedTime, this);
		mGameEnd.DisplayTitle(this);
		if (GetKey(olc::Key::SPACE).bReleased)
			mSceneState = SCENE::TITLE;
			mGameEnd.DestroyTitle();


	break;
```

6. 파일 분할

헤더의 이름 앞에 c를 붙이면 클래스명

CircleShootor.h	: 게임 루프 부분을 담당하는 헤더
GamePlay.h	: 게임 진행 부분을 담당하는 헤더
GameTitle.h	: 게임의 타이틀 부분을 담당하는 헤더
GameEnd.h	: 게임 오버 부분을 담당하는 헤더
Unit.h	: 게임의 유닛을 담당하는 헤더 (자식 클래스 - cEnemy, cUser)
Enemy.h	: 게임의 적 유닛을 담당하는 헤더 (부모 클래스 - cUnit)
State.h	: 게임의 적 유닛의 상태에 대한 정보를 담당하는 헤더
User.h	: 게임의 유저 유닛을 담당하는 헤더 (부모 클래스 - cUnit)
Bullet.h	: 게임의 탄환을 담당하는 헤더
Tile.h	: 게임의 타일을 담당하는 헤더
Map.h	: 게임의 맵(타일 맵)을 담당하는 헤더
Scroll.h	: 게임의 구름 스크롤을 담당하는 헤더
Define.h	: 게임 내의 enum, struct, define에 관한 정보들을 담아놓은 헤더
PROJECT_CS.cpp	: 게임의 메인 함수를 가지고 있는 파일

7. Y SORT를 통해 Enemy와 Actor의 표시를 재정렬

```
void cGamePlay::UpdatePlayGame(float fElapsedTime, cCircleShootor* tpCircle)
	- 생략 -
	mUnitVec.push_back(tActor);

	for (int ti = 0; ti < mLevel->GetUnitCount(); ti++)
		if (tEnemy[ti]->GetIsAlive())
			mUnitVec.push_back(tEnemy[ti]);



	sort(mUnitVec.begin(), mUnitVec.end(), DoCompared);


void cGamePlay::DisplayPlayGame(cCircleShootor* tpCircle)
	for (int ti = 0; ti < mUnitVec.size() ;  ti++)
		if (mUnitVec[ti]->GetIsAlive())
			mUnitVec[ti]->Display(tpCircle);


	- 생략 -
```

8. 게임 시간에 따른 난이도 조정

5초마다 난이도가 증가합니다.
시간에 따라 Enemy의 등장 개수가 1 ~ 10개까지 1씩 증가합니다.
시간에 따라 Enemy의 조준 오차가 100 ~ 0까지 5씩 감소합니다.
시간에 따라 Enemy의 Bullet 발사 쿨타임이 4.0 ~ 2.0까지 0.25초씩 감소합니다.
시간에 따라 Enemy의 리스폰 시간이 20.0 ~ 10.0까지 0.5초씩 감소합니다.

```
============================================================
시간에 따라 Enemy의 난이도가 변경되는 함수
void cLevel::UpdateLevel(float fElapsedTime, cEnemy* tEnemy)
	mGame_Play_Time += fElapsedTime;

	if (mGame_Play_Time > 30.0f)
```

```
		if (mUnit_Count < ENEMY_MAX)
			tEnemy[mUnit_Count].SetIsAlive(true);

			mUnit_Count++;


		if (mRespone_Time > 10.0f)
			mRespone_Time -= 0.5f;


		if (mBullet_Fire_Time > 2.0f)
			mBullet_Fire_Time -= 0.25f;


		if (mAccuracy > 0)
			mAccuracy -= 5;


		mGame_Play_Time -= 5.0f;
```

```
============================================================
Enemy의 조준 함수가 tAccuracy에 따라 조준 각이 변하도록 변경하였습니다.
tAccuracy는 조준 오차

void cEnemy::DoFireAim(cBullet tpBullet[], cUnit* tpTarget, int tAccuracy)
	float tTheta = (rand() % 90) - 45;
	tTheta = tTheta * (PI / 180.0f) * ((float)tAccuracy / 100.0f);

	float tX_Size = mX - tpTarget->GetX();
	float tY_Size = mY - tpTarget->GetY();

	//시작점
	tpBullet[mCurIndexBullet].SetBulletPosition(mX - (mSize + tpBullet[mCurIndexBullet].GetSize()) * cosf(atan2f(tY_Size, tX_Size) + tTheta), mY
- (mSize + tpBullet[mCurIndexBullet].GetSize()) * sinf(atan2f(tY_Size, tX_Size) + tTheta));

	//방향
	tpBullet[mCurIndexBullet].SetDirX(cosf(atan2f(-tY_Size, -tX_Size) + tTheta));
	tpBullet[mCurIndexBullet].SetDirY(sinf(atan2f(-tY_Size, -tX_Size) + tTheta));

	tpBullet[mCurIndexBullet].ShootFire();

	if (mCurIndexBullet < BULLET_MAX - 1)

		mCurIndexBullet++;

	else
		mCurIndexBullet = 0;
```

 # 잘한 점

- 동적 할당을 통해 게임 진행 중에만 메모리를 사용하게 하였습니다.
- 파일 분할을 통해 관리가 쉽게 하였습니다.
- AI의 움직임을 단순한 8방향이 아닌 원의 각도를 이용한 이동으로 하였습니다.

 # 잘못한 점

1. 타일 맵의 스크롤이 불가능합니다.

기타