

SFML을 이용한 벽치기 게임

임베디드시스템공학과 202201658 박소윤

유튜브 링크: <https://www.youtube.com/watch?v=nReuD26gmaw>

1. 직접 짠 코드 (프로그래밍 필수조건 구현 설명은 보라색으로 하겠습니다!)

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <time.h>
#include <string>
#include <iostream>
#include <vector>
using namespace sf;
using namespace std;

// 부모 클래스
class In {
protected: //상속받는 클래스에서 쓰기 위해서 protected 사용
    Texture t;
    double x, y;
public:
    Sprite s; //public에 있는 이유는 한 클래스에서 다른 클래스의 객체를 사용하기 위해
    존재, 예를 들어 부딪힘 확인 위해서(intersects)
    SoundBuffer b;
    Sound sound;
    void move(double dx, double dy) { //이동하는 함수
        x += dx;
        y += dy;
        s.setPosition(x, y);
    }
    void draw(RenderWindow& window) { //그리는 함수
        window.draw(s);
    }
};

// 상속받은 패들 클래스
class Paddle :public In {
private:
    bool die;
public:
    Paddle(double _x, double _y): //패들 생성자
    void gostop(double _x): // 패들 좌우 이동 함수(끝에 도착할시 return을 사용해서 밖
    화면을 나가지 못하게 하기)
};
```

```

Paddle::Paddle(double _x, double _y) {
    t.loadFromFile("paddle.png");
    s.setTexture(t);
    s.setScale(0.1, 0.7);
    s.setPosition(_x, _y);
    die = false;
}

void Paddle::gostop(double _x) {
    //_x 와 0의 크기를 비교해야 화면 끝에 도착했을 때 반대방향으로 움직일 수 있음.
    if (s.getPosition().x <= 0 && _x < 0) {
        return;
    }
    if (s.getPosition().x >= 370 && _x > 0) {
        return;
    }
    s.move(_x, 0);
}

//상속 받은 ball 클래스
class Ball : public In {
private:
    bool die;
    double dx;
    double dy;
public:
    Ball(); //볼 생성자
    void update(Paddle& _paddle); //볼 업데이트 함수(벽과 패들에 닿으면 방향 바꾸기,
이동과 소리 플레이, 죽음 판단)
    bool not_ball_paddle_hit(); //볼과 패들에 닿지 않는 상황을 저장하는 함수 (죽음)
};

Ball::Ball() {
    x = 200;
    y = 300;
    t.loadFromFile("ball.png");
    s.setTexture(t);
    s.setPosition(x, y);
    die = false;
    dx = 5.0;
    dy = -5.0;
    if (!b.loadFromFile("ball_hit.wav")) {

```

```

        cout << "ball_hit.wav 존재 X\n";
    }
    sound.setBuffer(b);
}

void Ball::update(Paddle& _paddle) { //==> 게임 필수 조건: 사운드 효과 (볼이 벽과 패들에 닿으면 사운드)
    if (s.getPosition().x <= 0 || s.getPosition().x >= 390) {
        dx = -dx;
        sound.play();
    }
    if (s.getPosition().y <= 0) {
        dy = -dy;
        sound.play();
    }

    if (FloatRect(_paddle.s.getGlobalBounds()).intersects(s.getGlobalBounds()) == true) {
        dy = -dy;
        sound.play();
    }
    else if (s.getPosition().y >= 600) {
        die = true;
        s.setPosition(200, 0);
        x = 200;
        y = 0;
    }
    s.move(dx, dy);
}

bool Ball::not_ball_paddle_hit() {
    //그냥 die만 return 하면 계속 죽어있는 상태여서 목숨이 바로 0으로 줄어든다.
    bool temp = die;
    die = false;
    return temp;
}

//상속받은 폭탄 클래스
class Bomb : public In {
private:
    bool die;
    double under_x;
    double speed;
public:

```

```

Bomb(); //폭탄 생성자

shared_ptr<Sprite>bombs[5]; //----->프로그래밍 필수 조건: 스마트 포인터, 객체
배열

//==> shared_ptr이 class이기에 bombs라는 객체를 만들었다.
//==> 하나의 객체에 대해 여러 포인터가 가리킬 수 있는 shared_ptr 사용 -> bombs
이라는 객체로 배열[5]개를 만들어
// 하나씩 포인터가 각각의 객체를 가리켜 스마트 포인터 이용

void under(Paddle& _paddle); //폭탄이 떨어지는 함수 (속도와 개수를 랜덤으로 설정,
최대 폭탄이 5개)
bool paddle_bomb_hit(); //패들과 폭탄이 부딪힌 상황을 저장하는 함수
void draw(RenderWindow& window); //폭탄 그리는 함수(여러 개의 폭탄이라 따로따로
그려줘야함.)=> 상속관련으로 overriding
};

Bomb::Bomb() {
    x = 210;
    y = 10;
    die = 0;
    speed = rand() % 10;
    t.loadFromFile("bomb.png");
    s.setTexture(t);
    s.setPosition(x, y);
    s.setScale(0.5, 0.5);
    for (int i = 0; i < 5; i++) {
        under_x = (rand() % 20) * 20;
        bombs[i] = make_shared<Sprite>(); //--->프로그래밍 필수 조건: 객체 동적
생성

        //==> make_shared 안에 new가 있기 때문에 동적 생성 가능
        *bombs[i] = s;
        bombs[i]->setPosition(under_x, 0);
    }
    if (!b.loadFromFile("crash.wav")) {
        cout << "crash.wav 존재 X\n";
    }
    sound.setBuffer(b);
}

void Bomb::under(Paddle& _paddle) {

    for (int i = 0; i < 5; i++) { //게임 필수 조건==>랜덤으로 n>1 동시 출현
        under_x = (rand() % 20) * 20;

```

```

        if (bombs[i]->getPosition().y >=600) {
            bombs[i]->setPosition(under_x, 0);
        }

        bombs[i]->move(0, speed);
        i
    f
    (FloatRect(_paddle.s.getGlobalBounds()).intersects(bombs[i]->getGlobalBounds()) == true) {
//==>게임 필수 조건: 닿으면 bar 폭발
        //패들의 경계선과 폭탄의 경계선이 닿을 때 ==> 공은 상관없음!
        sound.play(); //==>게임 필수 조건:사운드 효과 1개 (패들과 폭탄이
        닿으면!)
        bombs[i]->setPosition(under_x, 0); //폭탄이 떨어지면 위로 올라가
        서 다시 시작
        die = true;
    }
}

bool Bomb::paddle_bomb_hit() {
    bool temp = die;
    die = false;
    return temp;
}

void Bomb::draw(RenderWindow &window) {
    for (int i = 0; i < 5.; i++) {
        window.draw(*bombs[i]);
    }
}

//상속받은 목숨 클래스
class Heart : public In {
private:
    int heart;
    bool game;
    Font font;
    Text text1;
    string status;

    Text text2;
    string gameover;

public:

```

```

Heart(); //목숨 생성자
void death(Paddle &paddle); //죽음 판별 함수(목숨 3개에서 0이 되면 패들은 화면 밖
으로 나가고 게임 오버)
void draw(RenderWindow& window); //그리는 함수(게임오버 text가 끝나고 실행되야
하기 때문에 따로 만들기) => 상속관계로 overriding
bool gameend(); //게임 중단 하는 함수
};

```

```

Heart::Heart() {
    heart = 3; //==> 게임 필수 조건: bar 생명 개수 3개
    if (!font.loadFromFile("OpenSans-Bold.ttf")) {
        cout << "폰트 파일을 오픈할 수 없음!" << endl;
    }
    text1.setFont(font);
    text2.setFont(font);
    status = "heart:" + to_string(heart);
    text1.setString(status);
    text1.setCharacterSize(20);
    text1.setPosition(0, 0);
    game = false;
    if (!b.loadFromFile("gameover.mp3")) {
        cout << "gameover.mpe 존재 X\n";
    }
    sound.setBuffer(b);
}

```

```

void Heart::death(Paddle &paddle) {
    heart -= 1; //==> 하나씩 카운트하기
    status = "heart:" + to_string(heart);
    text1.setString(status);
    text1.setCharacterSize(20);
    text1.setPosition(0, 0);

    gameover = "Game over";
    text2.setString(gameover);
    text2.setCharacterSize(50);
    text2.setPosition(60, 250);
    if (heart == 0) { //==> 목숨이 끝날 때
        sound.play(); //=> 게임 필수 조건: 사운드 효과(게임 오버)
        paddle.s.setPosition(-11111, -1111); //=> 죽으면 bar 는 화면 밖으로!!
        game = true;
    }
}
}

```

```

void Heart::draw(RenderWindow& window) {
    window.draw(text1);
    if (heart == 0) {
        window.draw(text2);
    }
}

bool Heart::gameend() {
    return game;
}

//메인
int main() {
    srand(time(NULL));
    RenderWindow window(VideoMode(400, 600), "A_wall_hitting_game");
    window.setFramerateLimit(30);
    Texture Td;
    Sprite background;
    Td.loadFromFile("background.png");
    background.setTexture(Td);
    Paddle paddle(200, 590);
    Ball ball;
    Bomb bomb;
    Heart heart;

    while (window.isOpen()) {
        Event e;
        while (window.pollEvent(e)) {
            if (e.type == Event::Closed) {
                window.close();
            }
        }

        if (heart.gameend()) { //게임 오버
            continue; //밑에 각 객체의 업데이트와 그리기를 멈추기 위해
        }

        if (Keyboard::isKeyPressed(Keyboard::Left)) {
            paddle.gostop(-3.0); //왼쪽 키 누르면 -3만큼 이동
        }
        if (Keyboard::isKeyPressed(Keyboard::Right)) {
            paddle.gostop(3.0); //오른쪽 키 누르면 3만큼 이동
        }
    }
}

```

```

    }

    ball.update(paddle); //패들과 볼의 부딪힘을 봐야하기 때문에 아규먼트에 패들
을 넣음.

    bomb.under(paddle); //패들과 폭탄의 부딪힘을 봐야하기 때문에 아규먼트에
패들을 넣음.

    if (ball.not_ball_paddle_hit() || bomb.paddle_bomb_hit()) {
        heart.death(paddle); //죽으면 패들을 밖으로 내보내기 위해 아규먼트
에 패들을 넣음.
    }

    window.clear();
    window.draw(background);
    paddle.draw(window);
    ball.draw(window);
    bomb.draw(window);
    heart.draw(window);
    window.display();
}

return 0;
}

```

2. 소스코드 주요 루틴 설명 (주석 참고)

(1) 목숨에 관한 주요 루틴

- 목숨은 총 3개이다.
- 패들이 공을 못 받아서 공이 바닥에 떨어지거나 패들이 폭탄에 닿아서 폭발할 때 목숨이 하나 씩 줄어든다.
- 목숨 3개가 끝나면 게임오버라는 글씨와 함께 사운드가 재생한다.

(2) 게임 진행 주요 루틴

볼:

- 볼이 x,y축으로 5씩 움직인다.
- 움직이면서 화면의 맨 왼쪽과 맨 오른쪽에 부딪히면 x에 -를 붙이고 화면의 맨 위쪽과 패들이 없는 아래쪽에 부딪히면 y에 -y를 설정해서 각 방향을 반대방향으로 바꿔준다.

패들:

- 사용자가 좌우 키를 눌러서 좌우 방향으로 이동을 한다.
- 패들이 폭탄에 닿으면 사운드와 함께 목숨 하나가 줄어든다.
- 패들이 공을 놓치면 목숨 하나가 줄어든다.

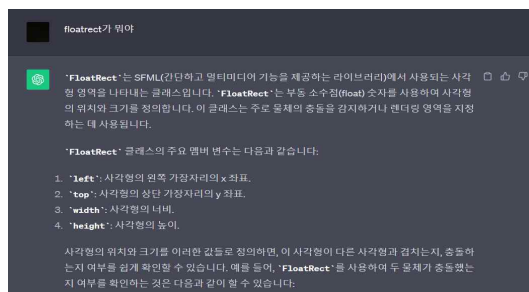
- 사용자가 왼쪽이나 오른쪽 키를 눌러 화면 밖 범위에 나가게 하면 나가지 못하게 설정했다.
- 폭탄:
- 폭탄이 1개 이상 여러 개가 랜덤으로 떨어진다.
- 폭탄이 바닥과 패들에 닿으면 화면 맨 위로 올라가서 다시 떨어지게 한다.
- 폭탄은 랜덤 x좌표로 설정된다.

3. chatGPT 사용한 부분 설명

(1) 패들과 볼, 패들과 폭탄 닿을 때 사용

두 객체의 스프라이트끼리 닿을 때의 경우를 알고 싶었는데 책에 나온 클래스와 함수를 이해하지 못 했기 때문에 FloatRect, intersects, getGlobalBounds에 대해 질문했다. 덕분에 무엇을 의미하고 무슨 일을 하는지 알게 돼서 활용하여 잘 적용할 수 있었다. 예시도 적혀있었기 때문에 쉽게 진행할 수 있었다.

- FloatRect 질문



FloatRect rect1(10.0f, 10.0f, 50.0f, 50.0f); // (10, 10) 좌표에서 시작하고 너비와 높이가 각각 50인 사각형
FloatRect rect2(30.0f, 20.0f, 60.0f, 40.0f); // (30, 20) 좌표에서 시작하고 너비와 높이가 각각 60인 사각형

```
if (rect1.intersects(rect2)) {
    // 두 사각형이 충돌함
    // 여기에 처리 코드 추가
}
```

위 코드에서 'intersects' 메서드는 두 개의 'FloatRect' 객체 간에 충돌이 있는지 여부를 검사합니다. 만약 충돌이 있다면, 두 사각형은 어딘가에서 겹치는 부분이 있습니다. 이것은 게임 프로그래밍 및 그래픽 처리에서 충돌 감지에 매우 유용한 도구입니다.

- intersects 질문



FloatRect rect1(10.0f, 10.0f, 50.0f, 50.0f); // (10, 10) 좌표에서 시작하고 너비와 높이가 각각 50인 사각형
FloatRect rect2(30.0f, 20.0f, 60.0f, 40.0f); // (30, 20) 좌표에서 시작하고 너비와 높이가 각각 60인 사각형

```
if (rect1.intersects(rect2)) {
    // 두 사각형이 충돌함
    // 여기에 처리 코드 추가
}
```

위 코드에서 `'rect1'`과 `'rect2'` 사이에 `'intersects'` 메서드가 호출되고, 두 사각형이 서로 겹치기 때문에 `'if'` 문이 `'true'`를 반환합니다. 이것은 충돌 검사에 사용되며, 게임에서 물체 간의 충돌을 감지하거나 처리하는 데 유용합니다.

📄 🌟 🗣️

(3) getGlobalBounds에 대한 질문

sfml에서 getGlobalBounds 뭐야?

SFML(Simple and Fast Multimedia Library)은 C++로 작성된 멀티미디어 및 게임 개발을 위한 라이브러리입니다. SFML을 사용하면 그래픽, 사운드, 네트워킹 등 다양한 멀티미디어 기능을 손쉽게 활용할 수 있습니다.

`*getGlobalBounds`는 SFML에서 주로 사용되는 함수 중 하나로, SFML의 그래픽 요소인 `sf::Drawable`을 상속한 객체(예: `sf::Sprite` 또는 `sf::Text`)의 전역 경계 상자(global bounding box)를 얻는 데 사용됩니다. 이 함수는 해당 객체가 화면 상에 어떤 위치에 그려져 있는지, 크기가 어떻게 되는지를 고려하여 전역 경계 상자를 계산합니다.

주로 충돌 감지나 객체 위치/크기에 관한 연산에 사용됩니다. 예를 들어, 두 개의 객체가 충돌했는지 여부를 확인하거나, 화면에 특정 객체를 그리는 위치와 크기를 조절하기 위해 `*getGlobalBounds` 함수를 사용할 수 있습니다.

다음은 `*getGlobalBounds` 함수의 일반적인 사용 예시입니다.

```
cpp
sf::Sprite sprite;
// 스프라이트 설정 (텍스처, 위치, 크기 등)

sf::FloatRect bounds = sprite.getGlobalBounds();
// 스프라이트의 전역 경계 상자를 얻음

// 경계 상자의 정보를 사용하여 작업을 수행할 수 있음 (예: 충돌 감지, 위치 설정 등)
```

위의 예시에서 `*sprite.getGlobalBounds()` 호출은 스프라이트 객체의 전역 경계 상자 정보를 반환합니다. 이 정보는 `*sf::FloatRect` 형식으로 반환되며, 객체의 위치, 크기 및 회전을 모두 고려합니다.

(2) 폭탄 랜덤으로 만들기

폭탄 5개가 랜덤위치에서 떨어지고 다시 위에서 시작하는 것을 위해서 질문을 했다.

- 폭탄 랜덤으로 떨어지는 질문

5개의 폭탄이 아래로 랜덤으로 떨어지는 c++ 코드 알려줘

아래는 C++를 사용하여 5개의 랜덤한 폭탄이 아래로 떨어지는 예제 코드입니다. 이 코드는 SFML 라이브러리를 사용하여 그래픽을 처리하고, 5개의 폭탄을 떨어뜨리는 간단한 시뮬레이션을 구현합니다. 먼저 SFML 라이브러리를 설치하고 환경을 설정해야 합니다.

```
#include <SFML/Graphics.hpp>
#include <vector>
#include <cstdlib>
#include <ctime>
using namespace sf;
int main() {
    // 윈도우 생성
    RenderWindow window(VideoMode(800, 600), "Falling Bombs");
    // 폭탄 텍스처 로딩
    Texture bombTexture;
    bombTexture.loadFromFile("bomb.png"); // "bomb.png" 파일은 폭탄 이미지 파일입니다.
    // 폭탄 스프라이트를 저장하는 벡터
    std::vector<Sprite> bombs;
    // 랜덤 시드 설정
    std::srand(static_cast<unsigned int>(std::time(nullptr)));
    while (window.isOpen()) {
        Event event;
        while (window.pollEvent(event)) {
            if (event.type == Event::Closed)
                window.close();
        }
        // 화면 클리어
```

```

window.clear();
// 5개의 폭탄을 생성하고 랜덤한 위치에서 떨어뜨립니다.
for (int i = 0; i < 5; i++) {
    // 랜덤한 x 좌표 선택
    float x = static_cast<float>(std::rand() % 800);

    // 새로운 폭탄 스프라이트 생성
    Sprite bomb;
    bomb.setTexture(bombTexture);
    bomb.setPosition(x, 0); // 화면 위쪽에서 시작
    // 폭탄을 아래로 떨어뜨리기
    float fallSpeed = 2.0f;
    bomb.move(0, fallSpeed);
    // 벡터에 폭탄 추가
    bombs.push_back(bomb);
}
// 벡터에 있는 모든 폭탄을 화면에 그립니다.
for (const auto& bomb : bombs) {
    window.draw(bomb);
}
// 화면 업데이트
window.display();
}
return 0;
}

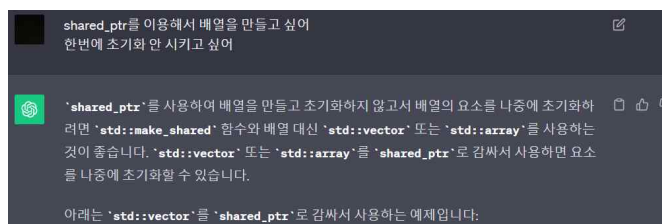
```

=> 폭탄이 화면 위쪽에서부터 떨어지는 도중 for문으로 다시 들어가서 bomb.setPosition(x, 0); 이 실행된다. 그렇게 되면 떨어졌다가 다시 y축이 0으로 돼서 다시 화면 위쪽으로 간다는 문제점을 발견해서 이 부분을 참고해서 코드 작성하였다. 그래서 나는 if문을 사용하여 폭탄이 바닥 밑에 닿거나 패들에 닿으면 화면 위로 가게 구현하였다.

(3) 게임 필수 조건 만족하기 위해 검색

객체 동적 생성과, 스마트 포인터, 객체 배열을 이용하여 한 번에 bomb을 만들고 싶어서 질문을 하였다.

- shard_ptr, 배열 질문



```

#include <iostream>
#include <memory>
#include <vector>
int main() {
    // int 형 요소를 가지는 std::vector를 생성합니다.
    std::shared_ptr<std::vector<int>> sharedVector = std::make_shared<std::vector<int>>();
    // 나중에 요소를 추가할 수 있습니다.
    sharedVector->push_back(1);
    sharedVector->push_back(2);
    sharedVector->push_back(3);
    // 요소 출력
    for (const int& value : *sharedVector) {
        std::cout << value << " ";
    }
    std::cout << std::endl;
    return 0;
}

```

이 예제에서 `std::make_shared<std::vector<int>>()`를 사용하여 `int` 요소를 가지는 `std::vector`를 `shared_ptr`로 생성합니다. 그런 다음 `sharedVector`를 통해 나중에 요소를 추가할 수 있습니다.

`std::array`의 경우 크기가 고정된 정적 배열이므로 배열의 크기를 지정하여 초기화합니다. 아래는 `std::array`를 `shared_ptr`로 감싸서 사용하는 예제입니다:

```
#include <iostream>
#include <memory>
#include <array>

int main() {
    // int 요소 5개를 가지는 std::array를 생성합니다.
    std::shared_ptr<std::array<int, 5>> sharedArray = std::make_shared<std::array<int, 5>>();
    // 나중에 요소를 초기화할 수 있습니다.
    (*sharedArray)[0] = 1;
    (*sharedArray)[1] = 2;
    (*sharedArray)[2] = 3;
    // 요소 출력
    for (int value : *sharedArray) {
        std::cout << value << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

이렇게 하면 나중에 요소를 초기화하거나 변경할 수 있으며, `shared_ptr`가 배열 또는 벡터를 효율적으로 관리합니다.

=> 동적 생성하려면 `make_shared`를 써야한다는 사실을 알게 됐다.

하지만 `shared_ptr`를 초기화를 안 시키고 싶었던 것에 대한 답변이 제대로 이루어지지 않아 아쉬웠다. 왜냐하면 난 `shared_ptr`를 클래스에서 사용하려고 해서 초기화를 한 번에 시키고 싶지 않았기 때문이다. 질문의 유도를 좀 더 잘 해야겠다는 생각이 들었다.

그래서 나는 코드를 클래스에 `shared_ptr<Sprite>bombs[5];`을 써서 초기화를 안 시키는 동시에 객체 배열과 스마트 포인터 조건을 만족하고 생성자에 `bombs[i] = make_shared<Sprite>();` 써서 객체 동적 생성을 만족하였다.