```cpp
#include "Bottle.h"
#include "Window.h"
#include "Player.h"
#include "math.h"
#include "Mixer.h"
#include "ObservableCollisionDetection.h"

Bottle::Bottle(int x, int y, bool headLeft) {
    appearance.x = x;
    appearance.y = y;
    appearance.w = 64;
    appearance.h = 20;
    this->headLeft = headLeft;
    bottle_tx = NULL;
    bottle_shattered_tx = NULL;
    curr_tx = NULL;
    ground_y_coordinate = Window::getInstance()->getWindowSizeH() - Window::getInstance()->getWindowSizeH()/8;
    ticks_to_self_destruction = 30;
    harmful = 6;
    sinus_ticks = 0;
    reached_max_height = false;
    turn = 0.0;
};

bool Bottle::loadMedia() {
    bool success = true;

    //Load Texture
    bottle_tx = loadTexture("assets/sprite_sheets/bottle/Bottle.png");
    if (bottle_tx == NULL) {
        printf("Failed to create texture. SDL Error: %s\n", SDL_GetError());
        success = false;
    }

    //Load Shattered Texture
    bottle_shattered_tx = loadTexture("assets/sprite_sheets/bottle/BottleShattered.png");
    if (bottle_tx == NULL) {
        printf("Failed to create texture. SDL Error: %s\n", SDL_GetError());
        success = false;
    }
    curr_tx = bottle_tx;

    //Attach to CD System
    ObservableCollisionDetection::getInstance()->attach(this);

    return success;
};

void Bottle::render() {
    //Depending on where the character looks at the moment, the texture gets flipped before rendering or not
    if (headLeft) {
        SDL_RenderCopyEx(Window::getInstance()->getRenderer(), curr_tx, NULL, &appearance, turn, NULL, SDL_FLIP_HORIZONTAL);
    }
    else {
        SDL_RenderCopyEx(Window::getInstance()->getRenderer(), curr_tx, NULL, &appearance, turn, NULL, SDL_FLIP_NONE);
    }

    if (false) { renderCollider(); }
};

void Bottle::tick() {
    /*for the first few ticks the bottle is not harmful,
      this done to prevent the player from hitting himself with a bottle */
    if (harmful > 0) {
        harmful--;
    }

    //Move if not collided
    //FLYING WHEN THROWN
    int max_sinus_ticks = 60;
    if (sinus_ticks >= ((max_sinus_ticks)/2)) {
        reached_max_height = true;
    }

    //(ticks_to_self_destruction >= 30) means the bottle has't collided yet
    if ((ticks_to_self_destruction >= 30)) {
        sinus_ticks++;

        const double PI = 3.14159265359;
        double amplitude = 15;
        //The normal sinus function does not look natural, therefore the reversed sinus curve (1-sinus) is used.
        double sinus = amplitude - amplitude * sin((double)sinus_ticks*(PI) / (double)max_sinus_ticks);
        /* The derivate of a sinus function is a cosine, function therefore this is used to determine
            the pitch of the bottle */
        double cosine = cos((double)sinus_ticks*(PI) / (double)max_sinus_ticks);
        double max_angle = 30.0;
        int speed = 10;
        double one_degree = 1.0 / max_angle;

        //Move y and set pitch
        if (reached_max_height == false) {
            appearance.y -= (int)(sinus);
            turn = cosine / one_degree;
        }
        else {
            /*It looks more natural if the bottle turns and falls down faster in the end
            than it turns and rises in the beginnning, therefore this is a bit adjusted here*/
            appearance.y += (int)(3 * sinus);
            turn = (1.5*cosine) / one_degree;
        }

        //If Bottle reaches the ground
        if (/*((sinus_ticks >= max_sinus_ticks) && (reached_max_height == true)) ||*/ ((appearance.y >= ground_y_coordinate) && (reached_max_height == true))) {
            reached_max_height = false;
            shatter();
            sinus_ticks = 0;
        }

        //Move x
        if (headLeft == true)
            {appearance.x -= speed;}
        else
        {
            appearance.x += speed;
            turn = -turn;
        }
    }

    //Wait for self-destruction
    if ((ticks_to_self_destruction < 30) && (ticks_to_self_destruction > 0)) {
            ticks_to_self_destruction--;
    }
};

std::string Bottle::isDead() {
    bool success = false;
    std::string isDead = "NOTDEAD";

    //IF WE RETURN "BROKENBOTTLE", THE BOTTLE WON'T BE TICKED ANYMORE, THEREFORE WE ONLY RETURN
    //"BROKENBOTTLE" WHEN THE COUNTDOWN TO SELFDESTRUCTION IS EXPIRED OR BOTTLE OUT OF THE WINDOW

    //If Bottle is in the screen
    if ((appearance.x < Window::getInstance()->getWindowSizeW()) && (appearance.x > 0)) {
        //If Bottle is already destroyed
        if (ticks_to_self_destruction <= 0) {
            close();
            ObservableCollisionDetection::getInstance()->detach(this);
            isDead = "BROKENBOTTLE";
        }
    }
    else {
        close();
        ObservableCollisionDetection::getInstance()->detach(this);
        isDead = "BROKENBOTTLE";
    }

    return isDead;
}

void Bottle::restart() { shatter(); };

std::string Bottle::getType() { //"PLAYERRIGHT", "PLAYERLEFT", "BOTTLE", "BACKGROUND"
    return "BOTTLE";
};

void Bottle::checkInput() { return; };

Bottle* Bottle::spawnBottle() { return NULL; };

bool Bottle::shatter() {
    bool success = false;

    if (curr_tx != bottle_shattered_tx) {
        //Play sound effect
        Mixer::getInstance()->play(Mixer::BOTTLE_SHATTERING);

        //Change texture
        curr_tx = bottle_shattered_tx;

        //Start countdown to self-destruction
        ticks_to_self_destruction--;
        success = true;
    }

    return success;
};

void Bottle::update(int collided_with, int own_collider, SDL_Rect rec) {
    /*  If the bottle itself is harmless or the bottle that it collided with is harmless
    (can mean shattered as well in this case) it is not supposed to react at all */
    if ((harmful > 0) || (collided_with == Player::BOTTLEHARMLESS)) {
        return;
    }

    shatter();
};

void Bottle::renderCollider() {
```

```
    //Saving the old rendercolor
    SDL_Color old_color;
    SDL_GetRenderDrawColor(Window::getInstance()->getRenderer(), &old_color.r, &old_color.g, &old_color.b, &old_color.a);

    //Set color
    SDL_Color color;
    color.r = 0xFF;
    color.g = 0x00;
    color.b = 0x00;
    color.a = 0x70;

    //Draw
    SDL_SetRenderDrawColor(Window::getInstance()->getRenderer(), color.r, color.g, color.b, color.a);
    SDL_RenderFillRect(Window::getInstance()->getRenderer(), &getColliders().at(0));

    //Reset the old rendercolor
    SDL_SetRenderDrawColor(Window::getInstance()->getRenderer(), old_color.r, old_color.g, old_color.b, old_color.a);
};

int Bottle::getColliderType(int index_in_vector) {
    int type;

    //Bottle can only hurt after the first few ticks and only once
    if ((harmful > 0) || (curr_tx == bottle_shattered_tx)) {
        type = Player::BOTTLEHARMLESS;
    }
    else {
        type = Player::BOTTLE;
    }

    return type;
};

std::vector<SDL_Rect> Bottle::getColliders() {
    //Create Collider Rectangle
    SDL_Rect collider_rec;
    collider_rec.x = appearance.x;
    collider_rec.y = appearance.y;
    collider_rec.w = appearance.w;
    collider_rec.h = appearance.h;

    //Create vector and fill him
    std::vector<SDL_Rect> collider;
    collider.push_back(collider_rec);

    return collider;
};

void Bottle::close() {
    //Destroying all textures
    SDL_DestroyTexture(bottle_tx);
    bottle_tx = NULL;

    SDL_DestroyTexture(bottle_shattered_tx);
    bottle_shattered_tx = NULL;

    SDL_DestroyTexture(curr_tx);
    curr_tx = NULL;
};

Bottle::~Bottle() {
    close();
};
```