MAIN.CPP

```cpp
#include <algorithm>
#include "Animate.h"
#include "Draw.h"
#include <ctime>
#include <iostream>
#include <math.h>
#include "Network.h"
#include "Player.h"
#include <stdlib.h>
#include "Object.h"
#include <vector>
#include <windows.h>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <SFX.h>

#define N 13
#define iniX 300
#define iniY 150

HWND xpos, ypos, xvalue, yvalue, pointsDisplay, applesDisplay, pointsVal, applesVal, startButton, resetButton;

float xlimit, ylimit;
int points      = 0;
int eatenApples = 0;
bool start      = false;
bool reset      = false;
bool gameOver   = false;
bool toggle     = true;
bool server     = true;

SFX sfx;

void createApple(std::vector<Object*>&apples,sf::RenderWindow&window,Player&info,Draw&player,int i){
    apples.push_back(new Object("sprites/apple.png"));
    xlimit = window.getSize().x-player.gfx.getLocalBounds().width;
    ylimit = window.getSize().y-player.gfx.getLocalBounds().height;
    apples[i]->loadSprite(xlimit,ylimit,info,player);
}

void isColliding(Player& info,Draw& player,std::vector<Object*>&apples,Object* object,int i,SFX& sfx){
    const float guy_left        = player.gfx.getPosition().x;
    const float guy_right       = player.gfx.getPosition().x+player.gfx.getLocalBounds().width;
    const float guy_top         = player.gfx.getPosition().y;
    const float guy_bottom      = player.gfx.getPosition().y+player.gfx.getLocalBounds().height;
    const float apple_left      = object->gfx.getPosition().x;
    const float apple_right     = object->gfx.getPosition().x+object->gfx.getLocalBounds().width;
    const float apple_top       = object->gfx.getPosition().y;
    const float apple_bottom    = object->gfx.getPosition().y+object->gfx.getLocalBounds().height;
    if(guy_left<apple_right&&guy_right>apple_left&&guy_top<apple_bottom&&guy_bottom>apple_top){
        object->eaten = true;
        sfx.sfx_collect.play();
        eatenApples++;
        points += 50;
        apples.erase(apples.begin()+i);
    }
}

```

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam){
    switch(msg){
        case WM_CREATE:{
        xpos          = CreateWindowEx(0,"Static","X Coordinate: ",WS_VISIBLE | WS_CHILD,30,525,100,20,hwnd,0,0,0);
        ypos          = CreateWindowEx(0,"Static","Y Coordinate: ",WS_VISIBLE | WS_CHILD,30,545,100,20,hwnd,0,0,0);
        xvalue        = CreateWindowEx(0,"Static","0",WS_VISIBLE | WS_CHILD,130,525,100,20,hwnd,0,0,0);
        yvalue        = CreateWindowEx(0,"Static","0",WS_VISIBLE | WS_CHILD,130,545,100,20,hwnd,0,0,0);
        pointsDisplay = CreateWindowEx(0,"Static","Points: ",WS_VISIBLE | WS_CHILD,280,525,100,20,hwnd,0,0,0);
        applesDisplay = CreateWindowEx(0,"Static","Apples: ",WS_VISIBLE | WS_CHILD,280,545,100,20,hwnd,0,0,0);
        pointsVal     = CreateWindowEx(0,"Static","0",WS_VISIBLE | WS_CHILD,340,525,100,20,hwnd,0,0,0);
        applesVal     = CreateWindowEx(0,"Static","0",WS_VISIBLE | WS_CHILD,340,545,100,20,hwnd,0,0,0);
        startButton   = CreateWindowEx(0,"Button","Start",WS_VISIBLE | WS_CHILD,707,25,80,30,hwnd,(HMENU)2,0,0);
        resetButton   = CreateWindowEx(0,"Button","Reset",WS_VISIBLE | WS_CHILD | WS_DISABLED,707,60,80,30,hwnd,(HMENU)1,0,0);
        break;
        }
        case WM_COMMAND:{
             switch (LOWORD(wParam)){
                case 1:{
                    gameOver    = false;
                    reset       = false;
                    toggle      = true;
                    points      = 0;
                    eatenApples = 0;
                    sfx.vgm.play();
                    EnableWindow(resetButton, false);
                    break;
                }
                case 2:{
                    start = true;
                    sfx.vgm.play();
                    EnableWindow(startButton, false);
                    break;
                }
             }
        break;
        }
        case WM_CLOSE:
        DestroyWindow(hwnd);
        break;
        case WM_DESTROY:
        PostQuitMessage(0);
        break;
        default:
        return DefWindowProc(hwnd,msg,wParam,lParam);
    }
    return 0;
}

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR cmdLine,int shwCmd){
    /* Define objects and variables. */
    MSG msg;
    Animate Movement;
    WNDCLASS WindowClass;
    sf::Clock timing;
    sf::Texture letters;
```

```cpp
    letters.loadFromFile("data/gfx/title.png");
    sf::Sprite logo(letters);
    logo.setPosition(sf::Vector2f(155,50));
    /* Create parent window's class. */
    WindowClass.cbClsExtra    = 0;
    WindowClass.cbWndExtra    = 0;
    WindowClass.hbrBackground = (HBRUSH)COLOR_WINDOW;
    WindowClass.hCursor       = LoadCursor(0, IDC_ARROW);
    WindowClass.hIcon         = LoadIcon(0, IDI_WINLOGO);
    WindowClass.hInstance     = hInstance;
    WindowClass.lpfnWndProc   = WndProc;
    WindowClass.lpszClassName = "Game";
    WindowClass.lpszMenuName  = 0;
    WindowClass.style         = 0;
    /* Register the class */
    RegisterClass(&WindowClass);
    /* Create the Win32 window & the child window that will contain the game. */
    HWND GameWindow = CreateWindowEx(0,"Game","Game Demo",WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU | WS_VISIBLE,100,100,800,600,0,0,hInstance,0);
    HWND ChildSFML  = CreateWindowEx(0,"Static",0,WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS,0,0,700,500,GameWindow,0,hInstance,0);
    if(GameWindow==NULL)
        return 1;
    /* Render the Win32 parent & the SFML child. */
    ShowWindow(GameWindow,shwCmd);
    UpdateWindow(GameWindow);
    sf::RenderWindow SFMLFrame(ChildSFML);
    SFMLFrame.setFramerateLimit(60);
    /* Create objects that will interact with each other. */
    Player P1;
    Draw Guy(P1,"sprites/6464.png", iniX, iniY);
    std::vector<Object*> apples;
    for(int i = 0;i<N;i++){
        createApple(apples,SFMLFrame,P1,Guy,i);
    }
    /* Create the window (game) loop. */
    msg.message = ~WM_QUIT;
    while(msg.message!=WM_QUIT){
ResetJump:
        if(reset){
            EnableWindow(resetButton, true);
        }
        /* Set up detection of keyboard inputs */
        P1.moving.up    = (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))    ? true: false;
        P1.moving.down  = (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))  ? true: false;
        P1.moving.left  = (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))  ? true: false;
        P1.moving.right = (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) ? true: false;
        /* Update the player, only if game is not over. */
        if(!gameOver&&start)
            P1.update(P1.moving.up,P1.moving.down,P1.moving.left,P1.moving.right,Guy,Movement,SFMLFrame);
        for(int i = 0;i<(int)apples.size();i++)
            isColliding(P1,Guy,apples,apples[i],i,sfx);
        /* Track some of the information. */
        char xv[100];
        char yv[100];
        std::string xx = std::to_string(eatenApples);
        std::string yy = std::to_string(points);
        sprintf(xv,"%f",roundf((P1.xpos * 100) / 100));
        sprintf(yv,"%f",roundf((P1.ypos * 100) / 100));
        SetWindowText(xvalue, xv);
        SetWindowText(yvalue, yv);
        SetWindowText(applesVal, xx.c_str());
        SetWindowText(pointsVal, yy.c_str());
        /* Handle WIN32 message and SFML events. */
        if(PeekMessage(&msg,NULL,0,0,PM_REMOVE)){
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }else{
            SFMLFrame.clear();
            if(!start)
                SFMLFrame.draw(logo);
            if(((int)apples.size()==0)&&toggle){
                sfx.vgm.stop();
                gameOver         = true;
                reset            = true;
                toggle           = false;
                time_t now       = time(0);
                std::string dt   = ctime(&now);
                dt.erase(std::remove(dt.begin(), dt.end(), '\n'), dt.end());
                if(server){
                    Network* net = new Network();
                    net->transmitData(dt,points,eatenApples);
                    delete net;
                }
                Guy.gfx.setPosition(sf::Vector2f(iniX, iniY));
                P1.xpos = Guy.gfx.getPosition().x;
                P1.ypos = Guy.gfx.getPosition().y;
                for(int i = 0;i<N;i++)
                    createApple(apples,SFMLFrame,P1,Guy,i);
                goto ResetJump;
            }
            else if(!gameOver&&start){
                SFMLFrame.draw(Guy.gfx);
                for(int i = 0;i<(int)apples.size();i++)
                    SFMLFrame.draw(apples[i]->gfx);
                Movement.Animation(P1,Guy,timing);
            }
            SFMLFrame.display();
        }
    }
    return (int)msg.wParam;
}
```

PLAYER.H

&

PLAYER.CPP

```cpp
class Animate;
class Draw;
#ifndef PLAYER_H
#define PLAYER_H

#include "Animate.h"
#include "Draw.h"
#include <SFML/Graphics.hpp>

class Player{
    public:
        struct{
            bool left;
            bool right;
            bool up;
            bool down;
        }facing;
        struct{
            bool left;
            bool right;
            bool up;
            bool down;
        }moving;
        float xvel;
        float yvel;
        float xpos;
        float ypos;
        void update(bool up,bool down,bool left,bool right,Draw& sprite,Animate&
Movement,sf::RenderWindow& window);
        Player(){
            this->facing.left   = false;
            this->facing.right  = false;
            this->facing.up     = false;
            this->facing.down   = true;
            this->moving.left  = false;
            this->moving.right = false;
            this->moving.up    = false;
            this->moving.down  = false;
            this->xvel = 0;
            this->yvel = 0;
            this->xpos = 0;
            this->ypos = 0;
        }
    protected:
    private:
};

#endif
```

```cpp
#include "Animate.h"
#include "Draw.h"
#include <iostream>
#include <math.h>
#include "Player.h"
#include <stdlib.h>
#include "Object.h"
#include <windows.h>
#include <SFML/Graphics.hpp>

void Player::update(bool up,bool down,bool left,bool right,Draw& sprite,Animate&
Movement,sf::RenderWindow& window){
    if(up){
        this->yvel   = -2.0f;
        sprite.frame.top = 64;
        this->facing.left  = false;
        this->facing.right = false;
        this->facing.up    = true;
        this->facing.down  = false;
    }
    if(down){
        this->yvel   = 2.0f;
        sprite.frame.top = 0;
        this->facing.left  = false;
        this->facing.right = false;
        this->facing.up    = false;
        this->facing.down  = true;
    }
    if(left){
        this->xvel     = -2.0f;
        sprite.frame.top = 128;
        this->facing.left  = true;
        this->facing.right = false;
        this->facing.up    = false;
        this->facing.down  = false;
    }
    if(right){
        this->xvel     = 2.0f;
        sprite.frame.top = 192;
        this->facing.left  = false;
        this->facing.right = true;
        this->facing.up    = false;
        this->facing.down  = false;
    }
    if(!(this->moving.left||this->moving.right))
        xvel = 0;
    if(!(this->moving.up||this->moving.down))
        yvel = 0;
    if(!this->moving.up&&!this->moving.down&&!this->moving.left&&!this->moving.right)
        Movement.IsAnimated = false;
    if(this->xpos+sprite.gfx.getLocalBounds().width>=window.getSize().x)
        this->xvel = -1;
    if(this->xpos<(window.getSize().x-window.getSize().x))
        this->xvel = 1;
    if(this->ypos+sprite.gfx.getLocalBounds().height>=window.getSize().y)
        this->yvel = -1;
    if(this->ypos<(window.getSize().y-window.getSize().y))
        this->yvel = 1;
    this->xpos += xvel;
    this->ypos += yvel;
    sprite.gfx.move(this->xvel, this->yvel);
}
```

DRAW.H

&

DRAW.CPP

```cpp
#ifndef DRAW_H
#define DRAW_H

#include <Player.h>
#include <SFML/Graphics.hpp>

class Draw{
    public:
        sf::Texture object;
        sf::Sprite gfx;
        sf::IntRect frame = sf::IntRect(0,0,64,64);
        Draw(Player& P1, const char * path, float x, float y){
            this->object.loadFromFile(path);
            sf::Sprite gfx(this->object, this->frame);
            gfx.setPosition(sf::Vector2f(x, y));
            P1.xpos = gfx.getPosition().x;
            P1.ypos = gfx.getPosition().y;
            this->gfx = gfx;
        }
    protected:
    private:
};

#endif
```

```cpp
#include "Animate.h"
#include "Draw.h"
#include <iostream>
#include "Player.h"
#include <windows.h>
#include <SFML/Graphics.hpp>
```

OBJECT.H

&

OBJECT.CPP

```cpp
#ifndef OBJECT_H
#define OBJECT_H

#include <random>

class Object{
    public:
        sf::Texture image;
        sf::Sprite gfx;
        const char * path;
        float xpos;
        float ypos;
        bool eaten;
        Object(const char* path){
            this->path = path;
            this->eaten = false;
        }
        void loadSprite(float x,float y,Player&info,Draw&player);
    protected:
    private:
};

#endif
```

```cpp
#include "Animate.h"
#include "Draw.h"
#include <iostream>
#include <random>
#include "Player.h"
#include <stdlib.h>
#include "Object.h"
#include <windows.h>
#include <SFML/Graphics.hpp>

void Object::loadSprite(float x,float y,Player&info,Draw&player){
    sf::Clock clock;
    this->image.loadFromFile(this->path);
    sf::Sprite gfx(this->image);
    std::mt19937 rng(clock.getElapsedTime().asMicroseconds());
RecalculateXPosition:
    std::uniform_real_distribution<float>xposcor(1,x);
    if((xposcor(rng)>=info.xpos)&&(xposcor(rng)<info.xpos+player.gfx.getLocalBounds().width))
        goto RecalculateXPosition;
    std::uniform_real_distribution<float>yposcor(1,y);
    this->xpos = xposcor(rng);
    this->ypos = yposcor(rng);
    gfx.setPosition(this->xpos, this->ypos);
    this->gfx = gfx;
}
```

ANIMATE.H

&

ANIMATE.CPP

```cpp
#ifndef ANIMATE_H
#define ANIMATE_H

#include "Draw.h"
#include "Player.h"
#include <SFML/Graphics.hpp>
#include <SFX.h>

class Animate{
    public:
    bool IsAnimated;
    void Animation(Player& P1,Draw& Guy,sf::Clock& timing);
    Animate(){
        this->IsAnimated = false;
    }
    protected:
    private:
};

#endif
```

```cpp
#include "Animate.h"
#include "Draw.h"
#include <iostream>
#include "Player.h"
#include <SFML/Graphics.hpp>
#include <windows.h>

void Animate::Animation(Player& P1,Draw& Guy,sf::Clock& timing){
    if((P1.moving.left||P1.moving.right||P1.moving.up||P1.moving.down)&&timing.getElapsedTime().asSeconds()>0.07){
        if(Guy.frame.left==192)
            Guy.frame.left = 0;
        else
            Guy.frame.left += 64;
        Guy.gfx.setTextureRect(Guy.frame);
        timing.restart();
        this->IsAnimated = true;
    }
}
```

SFX.H

&

SFX.CPP

```cpp
#ifndef SFX_H
#define SFX_H

#include <SFML/Audio.hpp>

class SFX{
    public:
        sf::Music vgm;
        sf::SoundBuffer buffer_collect;
        sf::Sound sfx_collect;
        SFX(){
            this->buffer_collect.loadFromFile("data/sfx/collect.wav");
            this->sfx_collect.setBuffer(this->buffer_collect);
            this->vgm.openFromFile("data/music/friday13.wav");
            this->vgm.setLoop(true);
        }
    protected:
    private:
};

#endif
```

```
#include "SFX.h"

```

NETWORK.H

&

NETWORK.CPP

```cpp
#ifndef NETWORK_H
#define NETWORK_H


class Network{
    public:
        void transmitData(std::string time,int points,int eaten);
    protected:
    private:
};

#endif
```

```cpp
#define NTDDI_VERSION NTDDI_VISTA
#define WINVER _WIN32_WINNT_VISTA
#define _WIN32_WINNT _WIN32_WINNT_VISTA

#include <iostream>
#include "Network.h"
#include <string>
#include <winsock2.h>
#include <ws2tcpip.h>

void Network::transmitData(std::string time,int points,int eaten){
    /* Initialise all the variables, structures, etc. */
    auto pointsStr = std::to_string(points);
    auto eatenStr  = std::to_string(eaten);
    WSADATA ini;
    sockaddr_in Server;
    WORD ver = MAKEWORD(2,2);
    SOCKET output;
    /* Boot up library. */
    int instance = WSAStartup(ver,&ini);
    if(instance!=0)
        std::cout << "WinSock initialisation failed." << std::endl;
    else
        std::cout << "WinSock initialisation successful!" << std::endl;
    /* Fill out "Server" structure. */
    Server.sin_family   = AF_INET;
    Server.sin_port     = htons(54000);
    InetPton(AF_INET,"127.0.0.1",&Server.sin_addr);
    /* Create the socket. */
    output = socket(AF_INET,SOCK_DGRAM,0);
    /* Send out the data. */
    for(int i = 0;i<3;i++){
        if(i==0)
            if(sendto(output,time.c_str(),strlen(time.c_str()),0,(sockaddr*)&Server,
sizeof(Server))!=SOCKET_ERROR)
                std::cout << "Sent \"" << time << "\" to server." << std::endl;
        if(i==1)
            if(sendto(output,pointsStr.c_str(),strlen(pointsStr.c_str()),0,(sockaddr
*)&Server,sizeof(Server))!=SOCKET_ERROR)
                std::cout << "Sent \"" << pointsStr << "\" to server." << std::endl;
        if(i==2)
            if(sendto(output,eatenStr.c_str(),strlen(eatenStr.c_str()),0,(sockaddr*)&
Server,sizeof(Server))!=SOCKET_ERROR)
                std::cout << "Sent \"" << eatenStr << "\" to server." << std::endl;
    }
    /* Close the socket & cleanup. */
    closesocket(output);
    WSACleanup();
}
```