2024

TIMBER!!!

PROGRAMMING

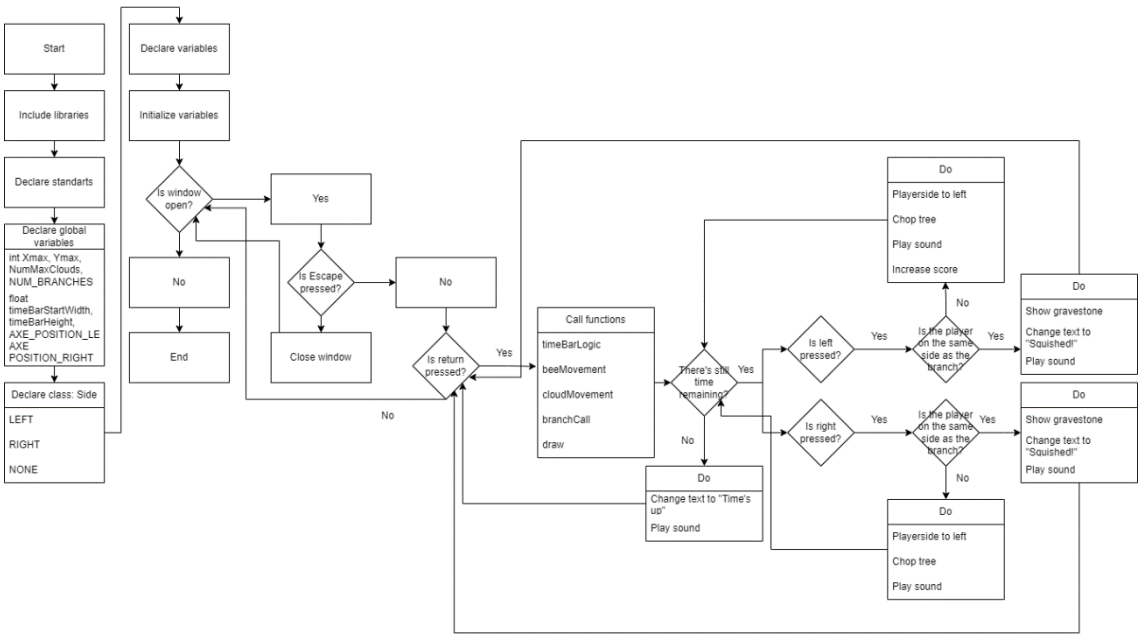
ÀLEX PALMADA

Table of contents

ABSTRACT

In this practice, we did the game called “Timber!”, it’s a game wher you have to chop the trees and if you are on a side where the branch falls, you will lose, so you have to chop the tree where the branch won’t fall.

Flux diagram



Code

```
//Including necessary libraries

#include <iostream>

#include <SFML/Audio.hpp>

#include <SFML/Graphics.hpp>

#include <sstream>

#include <Windows.h>


//Using the standarts

using namespace std;

using namespace sf;


//Global variables that never change

const int Xmax = 1920, Ymax = 1080, NumMaxClouds = 3, NUM_BRANCHES = 6;

const float timeBarStartWidth = 400, timeBarheight = 80, AXE_POSITION_LEFT = 850,
AXE_POSITION_RIGHT = 1050;


//Class to asssing possitions

enum class Side { LEFT, RIGHT, NONE };


//Function init, it loads the assets

void init(int score, stringstream& ss, FloatRect& textRect, Text& messageText, Text& scoreText,
Font& font, Texture& textureBackground, Texture& textureTree, Texture& textureBee,
Texture& textureCloud, Sprite& spriteBackground, Sprite& spriteTree, Sprite& spriteBee, Sprite
spriteCloud[], RectangleShape& timeBar, Time& gameTimeTotal, float& timeRemaining, float&
timeBarWidthPerSecond, Texture& textureBranch, Sprite branches[], Texture& texturePlayer,
Sprite& spritePlayer, Texture& textureAxe, Sprite& spriteAxe, Texture& textureRIP, Sprite&
spriteRIP, Texture& textureLog, Sprite& spriteLog, SoundBuffer& chopBuffer, Sound& chop,
SoundBuffer& deathBuffer, Sound& death, SoundBuffer& ootBuffer, Sound& oot) {


    //Loads the font and assigns it to variable. This variable has size, color and positioning

    font.loadFromFile("Fonts/komisan.ttf");

    messageText.setFont(font);

    messageText.setString("Press ENTER to start");
```

```
messageText.setCharacterSize(75);
messageText.setFillColor(Color::White);
textRect = messageText.getLocalBounds();
messageText.setOrigin(textRect.width / 2.0f, textRect.height / 2.0f);
messageText.setPosition(Xmax / 2.0f, Ymax / 2.0f);
scoreText.setCharacterSize(100);
scoreText.setFillColor(Color::White);
scoreText.setFont(font);

//It will concatenate the score with the default text
ss << "Score = " << score;

//Sets the color, size and positioning
scoreText.setString(ss.str());
scoreText.setPosition(20, 20);
timeBar.setSize(Vector2f(timeBarStartWidth, timeBarheight));
timeBar.setFillColor(Color::Red);
timeBar.setPosition((Xmax / 2) - timeBarStartWidth / 2, 980);
timeRemaining = 6.0f;
timeBarWidthPerSecond = timeBarStartWidth / timeRemaining;

//Loads the textures and assigns it to the sprites
texturePlayer.loadFromFile("./graphics/player.gif");
spritePlayer.setTexture(texturePlayer);
textureAxe.loadFromFile("./graphics/axe.png");
spriteAxe.setTexture(textureAxe);
textureRIP.loadFromFile("./graphics/rip.png");
spriteRIP.setTexture(textureRIP);
textureLog.loadFromFile("./graphics/log.png");
spriteLog.setTexture(textureLog);
textureBackground.loadFromFile("./graphics/background.png");
```

```
	spriteBackground.setTexture(textureBackground);
	spriteBackground.setPosition(0, 0);
	textureTree.loadFromFile("./graphics/tree.png");
	spriteTree.setTexture(textureTree);
	spriteTree.setPosition(810, 0);
	textureBee.loadFromFile("./graphics/bee.gif");
	spriteBee.setTexture(textureBee);
	spriteBee.setPosition(0, 800);
	textureBranch.loadFromFile("./graphics/branch.png");
	for (int r = 0; r < NUM_BRANCHES; r++) {
		branches[r].setTexture(textureBranch);
		branches[r].setPosition(-2000, -2000);
		branches[r].setOrigin(220, 20);
	}
	textureCloud.loadFromFile("./graphics/cloud.png");
	for (int i = 0; i < NumMaxClouds; i++) {
		spriteCloud[i].setTexture(textureCloud);
		spriteCloud[i].setPosition(-325, i * 250);
	}

	//Loads the buffer and assigns it to the sound
	chopBuffer.loadFromFile("./sound/chop.wav");
	chop.setBuffer(chopBuffer);
	deathBuffer.loadFromFile("./sound/death.wav");
	death.setBuffer(deathBuffer);
	ootBuffer.loadFromFile("./sound/out_of_time.wav");
	oot.setBuffer(ootBuffer);
}

//Function to move the Bee
void beeMovement(Clock& watch1, Sprite& spriteBee, bool& beeActive, float& beeSpeed) {
```

```
	//Everytime it runs, it restarts the variable
	Time dt = watch1.restart();

	//It checks if the bee is on screen, if it isn't, assigns a random speed, random height and turns it on
	if (!beeActive) {
		beeSpeed = (rand() % 200) + 200;
		float height = (rand() % 200) + 500;
		spriteBee.setPosition(2000, height);
		beeActive = true;
	}

	//If it is active, it displaces the position of the bee to the left
	else
	{
		spriteBee.setPosition(spriteBee.getPosition().x - (beeSpeed * dt.asSeconds()), spriteBee.getPosition().y);

		//If it reaches the border, the bee deactivates
		if (spriteBee.getPosition().x < -100)
		{
			beeActive = false;
		}
	}
}

//Function to move the clouds
void cloudMovement(Clock& watch2, Sprite spriteCloud[], bool cloudActive[], float height, float lastheight, float cloudSpeed[]) {

	//Everytime it runs, it restarts the variable
```

```
        Time dt = watch2.restart();


        //A loop that loads to all clouds
        for (size_t u = 0; u < NumMaxClouds; u++)
        {
                //If the current cloud isn't active, assigns a random height depending on the height of the last cloud, random speed and actives the cloud
                if (!cloudActive[u])
                {
                        cloudSpeed[u] = (rand() % 60 + 30);
                        height = (rand() % 300 + 1);
                        lastheight = height;
                        spriteCloud[u].setPosition(-325, lastheight);
                        cloudActive[u] = true;
                }

                //If the current cloud is active, it displaces the position of the cloud to the right
                else
                {
                        spriteCloud[u].setPosition(spriteCloud[u].getPosition().x + (cloudSpeed[u] * dt.asSeconds()), spriteCloud[u].getPosition().y);

                        //If the current cloud reaches the border, it deactivates
                        if (spriteCloud[u].getPosition().x > Xmax)
                        {
                                cloudActive[u] = false;
                        }
                }
        }
}

//Function to load the size of the bar
```

```
void timeBarLogic(RectangleShape& timeBar, float& timeRemaining, float&
timeBarWidthPerSecond, Clock& watch3) {

	//Everytime it runs, it restarts the variable
	Time dt = watch3.restart();

	//Substracts the time remaining every second
	timeRemaining -= dt.asSeconds();

	//It assigns the current size to the bar
	timeBar.setSize(Vector2f(timeBarWidthPerSecond * timeRemaining, timeBarheight));
}

//Function to assing a side randomly
void updateBranches(Side branchPositions[], int& seed) {

	//For every branch, displaces one to the right
	for (int j = NUM_BRANCHES - 1; j > 0; j--) {
		branchPositions[j] = branchPositions[j - 1];
	}

	//Generates a random number
	int r = rand() % 3;

	//Depending on what number selects, the position of the branch is assigned.
	switch (r) {
	case 0:
		branchPositions[0] = Side::LEFT;
		break;
	case 1:
		branchPositions[0] = Side::RIGHT;
```

```
			break;
		case 2:
			branchPositions[0] = Side::NONE;
			break;
		}
		seed++;
}

//Function to load the branches positions
void branchCall(int& seed, Sprite branches[], Side branchPositions[], Clock& watch5) {

		//For every branch, assigns a position according to the possition is assigned in another function
		for (int i = 0; i < NUM_BRANCHES; i++) {
			float height = i * 150;
			if (branchPositions[i] == Side::LEFT) {
				branches[i].setPosition(610, height);
				branches[i].setRotation(180);
			}
			else if (branchPositions[i] == Side::RIGHT) {
				branches[i].setPosition(1330, height);
				branches[i].setRotation(0);
			}
			else {
				branches[i].setPosition(3000, height);
			}
		}
}

//Decides when to draw on screen the textures and the texts and handles some mechanics
void draw(bool& paused, Text& messageText, Text& scoreText, RenderWindow& window, Sprite& spriteBackground, Sprite& spriteTree, Sprite& spriteBee, Sprite spriteCloud[], float&
```

```
timeRemaining, FloatRect& textRect, RectangleShape& timeBar, Texture& textureBranch,
Sprite branches[], int& score, Side branchPositions[], Sprite& spritePlayer, Sprite& spriteAxe,
Sprite& spriteRIP, bool& acceptInput, Side& playerSide, Sprite& spriteLog, float& logSpeedY,
float& logSpeedX, bool& logActive, int& seed, bool positioned[], Clock& watch4, stringstream&
ss, Sound& chop, Sound& death, Sound& oot, Event& event, bool& oSound, bool& dSound) {

	//Everytime it runs, it restarts the variable
	Time dt = watch4.restart();

	//Cleans the frame
	window.clear();

	//Starts and draws the background, tree, branches and the message to start the gamne
	window.draw(spriteBackground);
	window.draw(spriteTree);
	for (int e = 0; e < NUM_BRANCHES; e++)
	{
		window.draw(branches[e]);
	}
	if (paused) {
		window.draw(messageText);
	}

	//Detects if the return key is pressed, whe is it pressed, restarts the game loading by
default the assets
	if (Keyboard::isKeyPressed(Keyboard::Return) && paused && timeRemaining > 0.0f) {
		paused = false;
		score = 0;
		timeRemaining = 6;
		ss.str("");
		ss.clear();
		ss << "Score = " << score;
		scoreText.setString(ss.str());
```

```
            for (int i = 0; i < NUM_BRANCHES; i++)
            {
                branchPositions[i] = Side::NONE;
            }
            spriteRIP.setPosition(675, 2000);
            spritePlayer.setPosition(580, 720);
            acceptInput = true;
            dSound = true;
            oSound = true;
        }

        //When the game isn't paused, draws on screen the assets
        if (!paused) {
            for (int i = 0; i < NumMaxClouds; i++) {
                window.draw(spriteCloud[i]);
            }
            window.draw(spriteTree);
            for (int e = 0; e < NUM_BRANCHES; e++)
            {
                window.draw(branches[e]);
            }
            window.draw(spriteBee);
            window.draw(timeBar);
            window.draw(scoreText);
            window.draw(spritePlayer);
            window.draw(spriteAxe);
        }

        //Conditional that checks when a when a key is pressed or not, when is it active, it will
accept inputs
        if (acceptInput)
```

```
{

        //When the key is pressed, assigns the possition to the player and axe sprite, makes the log fly and increases the score.
        if (Keyboard::isKeyPressed(Keyboard::Right))
        {
                playerSide = Side::RIGHT;
                score++;
                timeRemaining += (2 / score) + .15;
                //cout << int(timeRemaining);
                spriteAxe.setPosition(AXE_POSITION_RIGHT, 800);
                spritePlayer.setPosition(1125, 720);
                spriteAxe.setScale(1.f, 1.f);
                spriteLog.setPosition(810, 720);
                updateBranches(branchPositions, seed);
                logSpeedX = -5000;
                logActive = true;
                acceptInput = false;
                window.draw(scoreText);
                ss.str("");
                ss.clear();
                ss << "Score = " << score;
                scoreText.setString(ss.str());
                chop.play();
        }

        //When the key is pressed, assigns the possition to the player and axe sprite, makes the log fly and increases the score.
        else if (Keyboard::isKeyPressed(Keyboard::Left))
        {
                playerSide = Side::LEFT;
                score++;
```

```
                timeRemaining += (2 / score) + .25;
                //cout << int(timeRemaining);
                spriteAxe.setPosition(AXE_POSITION_LEFT, 800);
                spriteAxe.setScale(-1.f, 1.f);
                spritePlayer.setPosition(580, 720);
                spriteLog.setPosition(810, 720);
                updateBranches(branchPositions, seed);
                logSpeedX = +5000;
                logActive = true;
                acceptInput = false;
                ss.str("");
                ss.clear();
                ss << "Score = " << score;
                scoreText.setString(ss.str());
                chop.play();
            }
        }
        while (window.pollEvent(event))
        {
            //An event that checks
            if (event.type == Event::KeyReleased && !paused)
            {
                acceptInput = true;
                spriteAxe.setPosition(4000, spriteAxe.getPosition().y);
            }
        }
        //When the log is choped, it will fly
        if (logActive)
        {
            spriteLog.setPosition(spriteLog.getPosition().x + (logSpeedX * dt.asSeconds()),
spriteLog.getPosition().y + (logSpeedY * dt.asSeconds()));
```

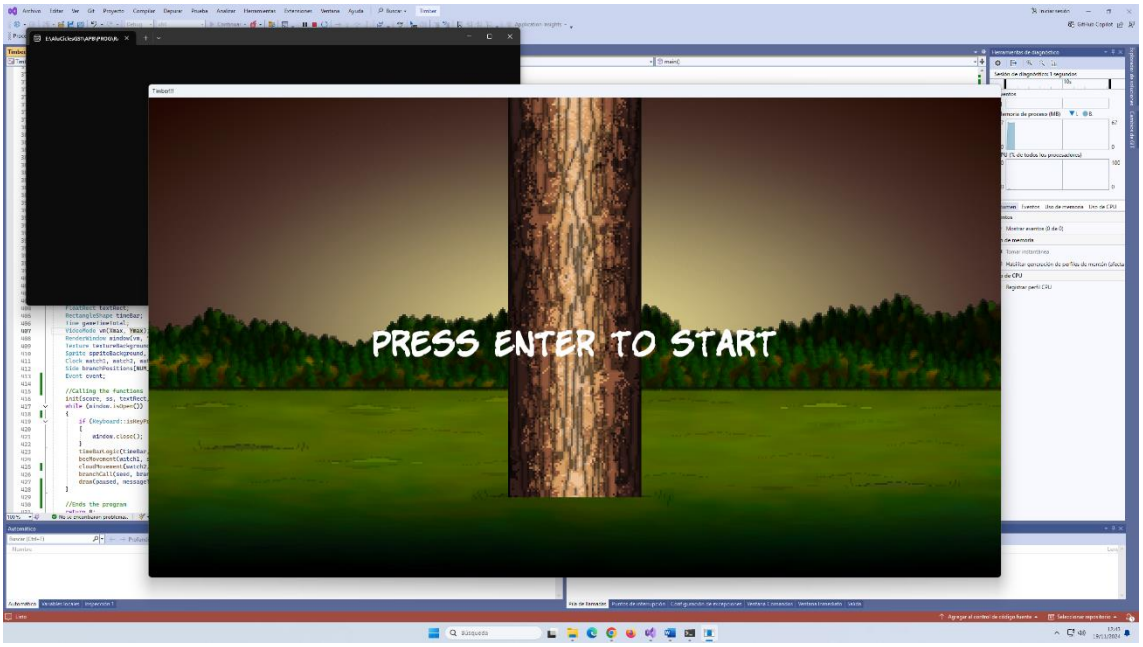
```
		if (spriteLog.getPosition().x < -100 || spriteLog.getPosition().x > 2000)
		{
			logActive = false;
			spriteLog.setPosition(810, 720);
		}
		window.draw(spriteLog);
	}

	//If the position of the branch is the same as the side of the player, it game over to the player
	if (branchPositions[5] == playerSide)
	{
		paused = true;
		acceptInput = false;
		spriteRIP.setPosition(525, 760);
		spritePlayer.setPosition(2000, 660);
		messageText.setString("SQUISHED!!");
		textRect = messageText.getLocalBounds();
		messageText.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top + textRect.height / 2.0f);
		messageText.setPosition(1920 / 2.0f, 1080 / 2.0f);
		if (playerSide == Side::LEFT)
		{
			spriteRIP.setPosition(580, 740);
		}
		else if (playerSide == Side::RIGHT)
		{
			spriteRIP.setPosition(1200, 740);
		}
		if (dSound)
		{
			death.play();
```

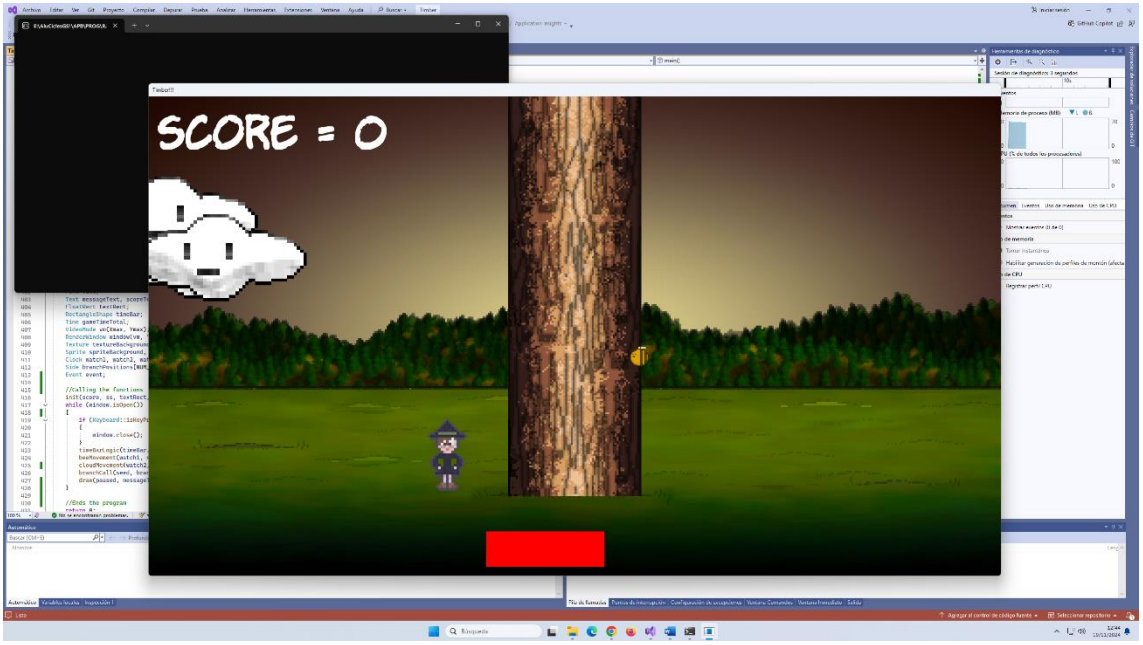
```
                dSound = false;
                oSound = false;
            }
            spriteAxe.setPosition(4000, spriteAxe.getPosition().y);
            window.draw(spriteRIP);
        }

        //If the time remaining reaches 0, it game over to the player
        if (timeRemaining <= 0.0f)
        {
            paused = true;
            messageText.setString("Out of time");
            textRect = messageText.getLocalBounds();
            messageText.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top +
textRect.height / 2.0f);
            messageText.setPosition(1920 / 2.0f, 1080 / 2.0f);
            if (oSound)
            {
                oot.play();
                oSound = false;
            }
            timeRemaining = 6.0f;
            spriteAxe.setPosition(4000, spriteAxe.getPosition().y);
        }

        //If the time remaining is above 6, it will cap to 6
        if (timeRemaining > 6.0f)
        {
            timeRemaining = 6.0f;
        }
```

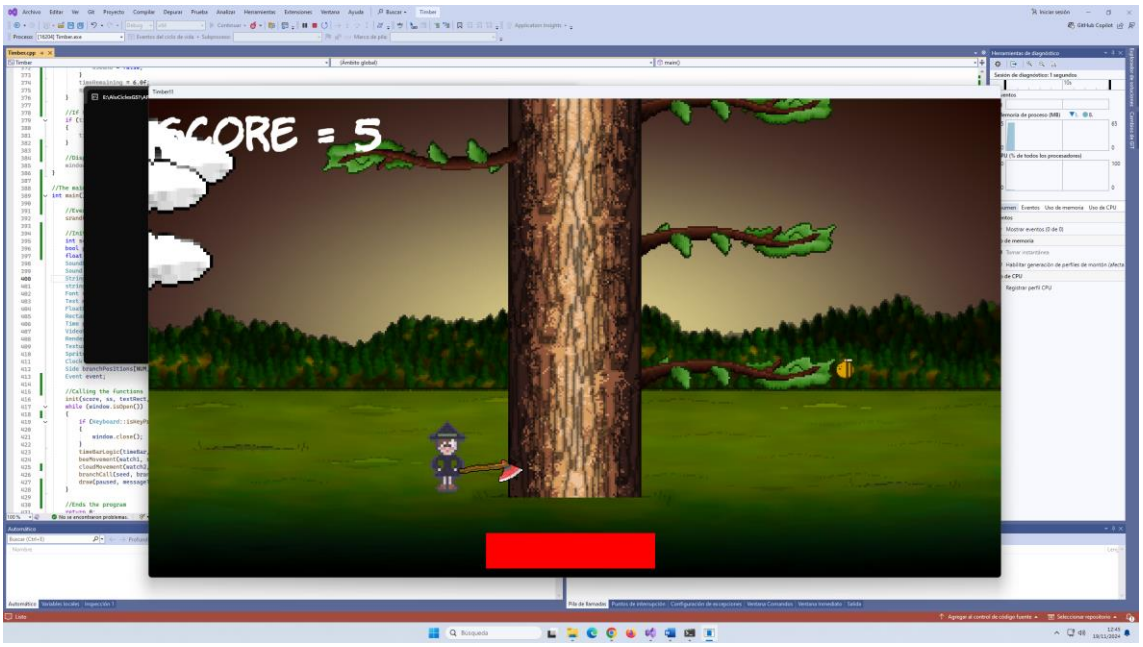
```
	//Displays everything on scren
	window.display();
}

//The main where the variables are created
int main() {

	//Every time the game starts, it will start randomly
	srand((int)time(0) * 10);

	//Initialization of scores
	int score = 0, seed;
	bool paused = true, beeActive = false, cloudActive[NumMaxClouds] = { false }, positioned[NUM_BRANCHES] = { false }, logActive = false, acceptInput = false, dSound, oSound;
	float height = 0, lastheight = (rand() % 300 + 1), beeSpeed = 0.0f, cloudSpeed[NumMaxClouds] = { 0.0f }, timeRemaining, timeBarWidthPerSecond, logSpeedX = -1000, logSpeedY = -1500;
	SoundBuffer chopBuffer, deathBuffer, ootBuffer;
	Sound chop, death, oot;
	String levelName = "DastardlyCave", playerName = "Jhon Carmack";
	stringstream ss;
	Font font;
	Text messageText, scoreText;
	FloatRect textRect;
	RectangleShape timeBar;
	Time gameTimeTotal;
	VideoMode vm(Xmax, Ymax);
	RenderWindow window(vm, "Timber!!!", Style::Titlebar);
	Texture textureBackground, textureTree, textureBee, textureCloud, textureBranch, texturePlayer, textureAxe, textureRIP, textureLog;
	Sprite spriteBackground, spriteTree, spriteBee, spriteCloud[NumMaxClouds], branches[NUM_BRANCHES], spritePlayer, spriteAxe, spriteRIP, spriteLog;
	Clock watch1, watch2, watch3, watch4, watch5;
```

```
	Side branchPositions[NUM_BRANCHES] = {Side::NONE, Side::NONE, Side::NONE,
Side::NONE, Side::NONE, Side::NONE}, playerSide = Side::LEFT;

	Event event;


	//Calling the functions

	init(score, ss, textRect, messageText, scoreText, font, textureBackground, textureTree,
textureBee, textureCloud, spriteBackground, spriteTree, spriteBee, spriteCloud, timeBar,
gameTimeTotal, timeRemaining, timeBarWidthPerSecond, textureBranch, branches,
texturePlayer, spritePlayer, textureAxe, spriteAxe, textureRIP, spriteRIP, textureLog, spriteLog,
chopBuffer, chop, deathBuffer, death, ootBuffer, oot);

	while (window.isOpen())

	{

		if (Keyboard::isKeyPressed(Keyboard::Escape))

		{

			window.close();

		}

		timeBarLogic(timeBar, timeRemaining, timeBarWidthPerSecond, watch3);

		beeMovement(watch1, spriteBee, beeActive, beeSpeed);

		cloudMovement(watch2, spriteCloud, cloudActive, height, lastheight,
cloudSpeed);

		branchCall(seed, branches, branchPositions, watch5);

		draw(paused, messageText, scoreText, window, spriteBackground, spriteTree,
spriteBee, spriteCloud, timeRemaining, textRect, timeBar, textureBranch, branches, score,
branchPositions, spritePlayer, spriteAxe, spriteRIP, acceptInput, playerSide, spriteLog,
logSpeedY, logSpeedX, logActive, seed, positioned, watch4, ss, chop, death, oot, event, dSound,
oSound);

	}


	//Ends the program

	return 0;

}
```

Execution tests






SCORE = 5


SQUISHED!!