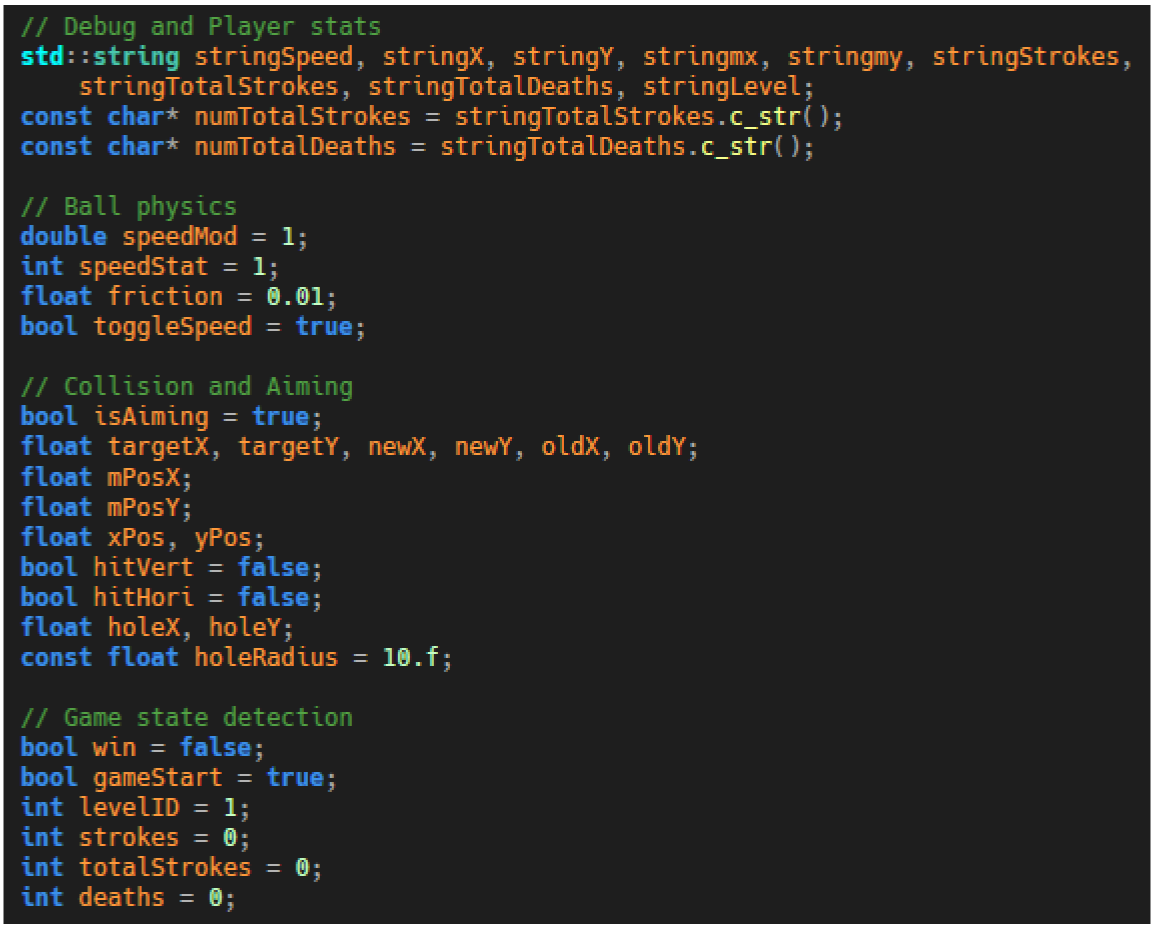
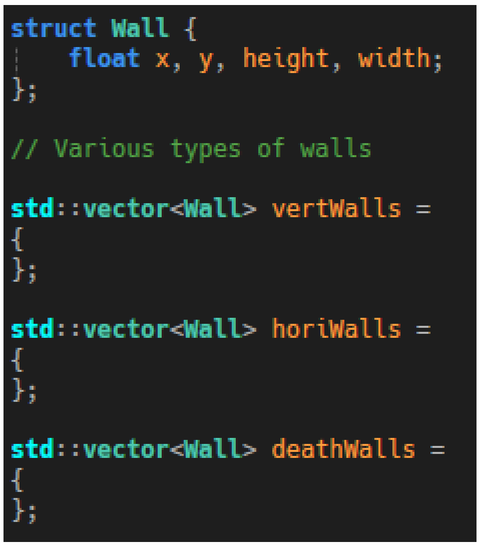


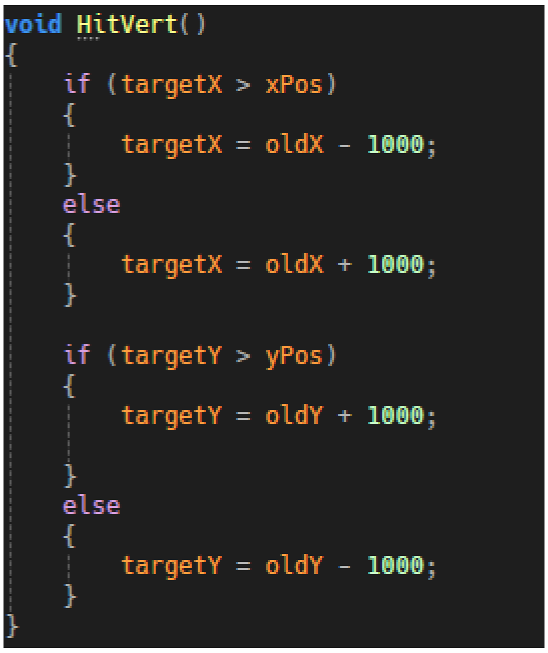
```cpp
// Debug and Player stats
std::string stringSpeed, stringX, stringY, stringmx, stringmy, stringStrokes,
    stringTotalStrokes, stringTotalDeaths, stringLevel;
const char* numTotalStrokes = stringTotalStrokes.c_str();
const char* numTotalDeaths = stringTotalDeaths.c_str();

// Ball physics
double speedMod = 1;
int speedStat = 1;
float friction = 0.01;
bool toggleSpeed = true;

// Collision and Aiming
bool isAiming = true;
float targetX, targetY, newX, newY, oldX, oldY;
float mPosX;
float mPosY;
float xPos, yPos;
bool hitVert = false;
bool hitHori = false;
float holeX, holeY;
const float holeRadius = 10.f;

// Game state detection
bool win = false;
bool gameStart = true;
int levelID = 1;
int strokes = 0;
int totalStrokes = 0;
int deaths = 0;
```

Various variables that I used to define coordinates, integers, and debugging



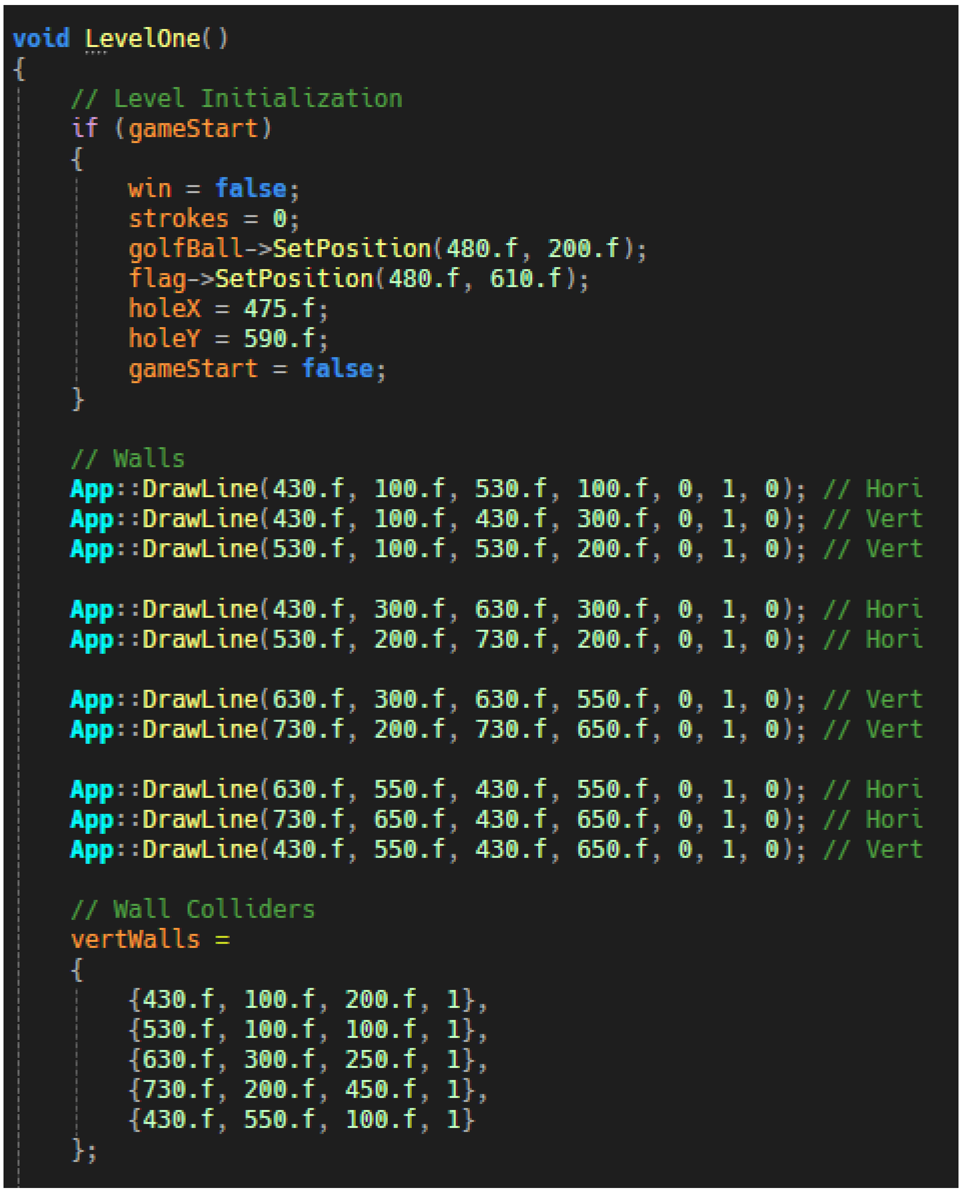
```cpp
struct Wall {
    float x, y, height, width;
};

// Various types of walls

std::vector<Wall> vertWalls =
{
};

std::vector<Wall> horiWalls =
{
};

std::vector<Wall> deathWalls =
{
};
```

How I define walls, walls are added during the level making process



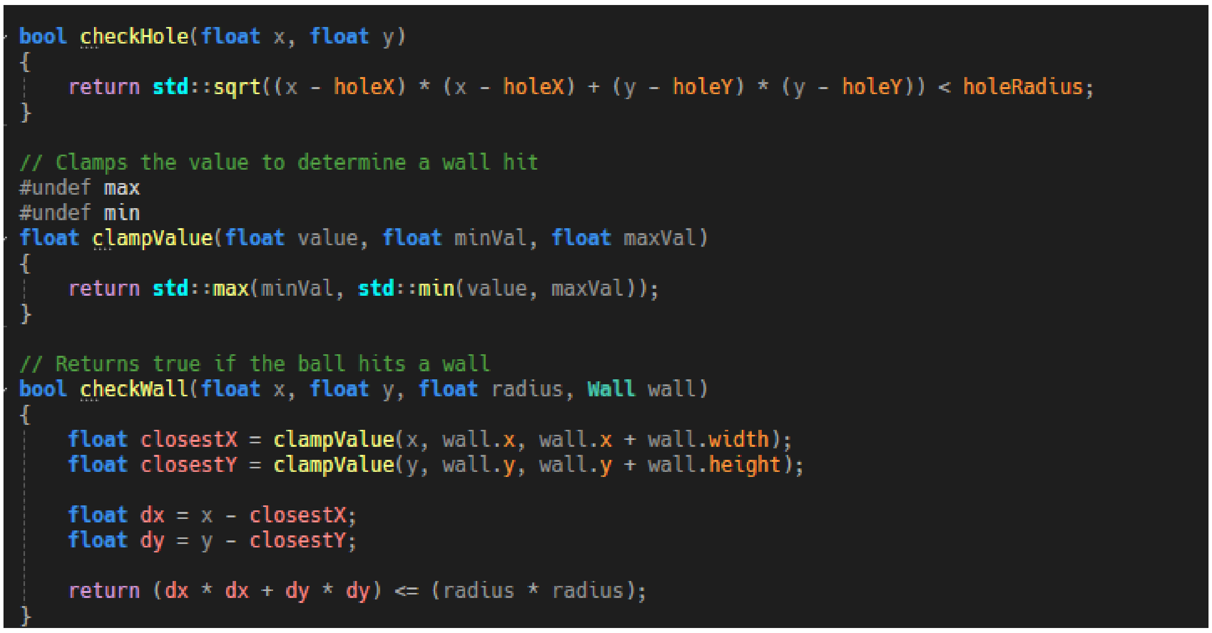
```
void HitVert()
{
    if (targetX > xPos)
    {
        targetX = oldX - 1000;
    }
    else
    {
        targetX = oldX + 1000;
    }

    if (targetY > yPos)
    {
        targetY = oldY + 1000;

    }
    else
    {
        targetY = oldY - 1000;
    }
}
```

How the game calculates where to send the ball after hitting a wall, one for each type of wall that does slightly different things



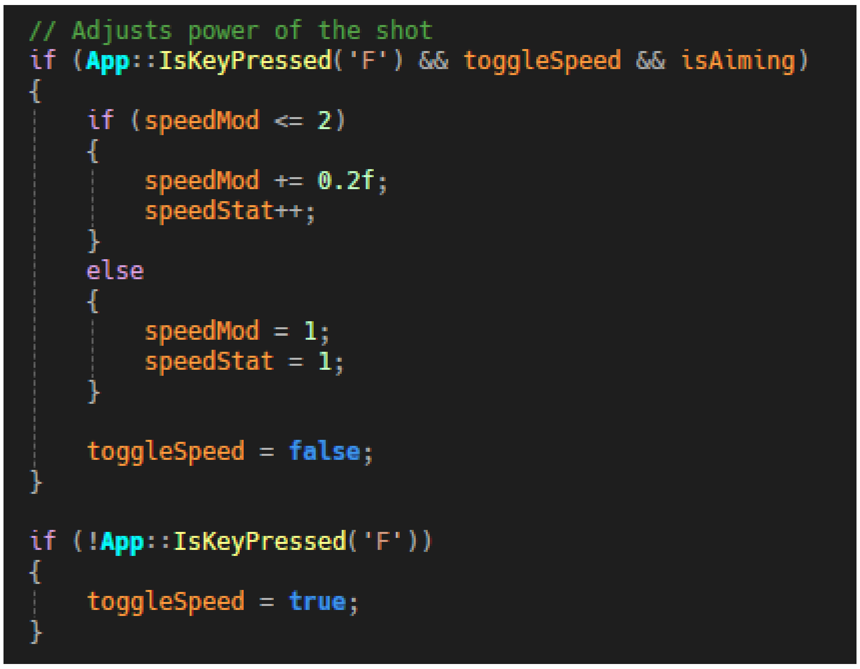
```
void LevelOne()
{
    // Level Initialization
    if (gameStart)
    {
        win = false;
        strokes = 0;
        golfBall->SetPosition(480.f, 200.f);
        flag->SetPosition(480.f, 610.f);
        holeX = 475.f;
        holeY = 590.f;
        gameStart = false;
    }

    // Walls
    App::DrawLine(430.f, 100.f, 530.f, 100.f, 0, 1, 0); // Hori
    App::DrawLine(430.f, 100.f, 430.f, 300.f, 0, 1, 0); // Vert
    App::DrawLine(530.f, 100.f, 530.f, 200.f, 0, 1, 0); // Vert

    App::DrawLine(430.f, 300.f, 630.f, 300.f, 0, 1, 0); // Hori
    App::DrawLine(530.f, 200.f, 730.f, 200.f, 0, 1, 0); // Hori

    App::DrawLine(630.f, 300.f, 630.f, 550.f, 0, 1, 0); // Vert
    App::DrawLine(730.f, 200.f, 730.f, 650.f, 0, 1, 0); // Vert

    App::DrawLine(630.f, 550.f, 430.f, 550.f, 0, 1, 0); // Hori
    App::DrawLine(730.f, 650.f, 430.f, 650.f, 0, 1, 0); // Hori
    App::DrawLine(430.f, 550.f, 430.f, 650.f, 0, 1, 0); // Vert

    // Wall Colliders
    vertWalls =
    {
        {430.f, 100.f, 200.f, 1},
        {530.f, 100.f, 100.f, 1},
        {630.f, 300.f, 250.f, 1},
        {730.f, 200.f, 450.f, 1},
        {430.f, 550.f, 100.f, 1}
    };
```

Example of a level being made, App::Drawline is used to show players where walls are, actual walls are invisible



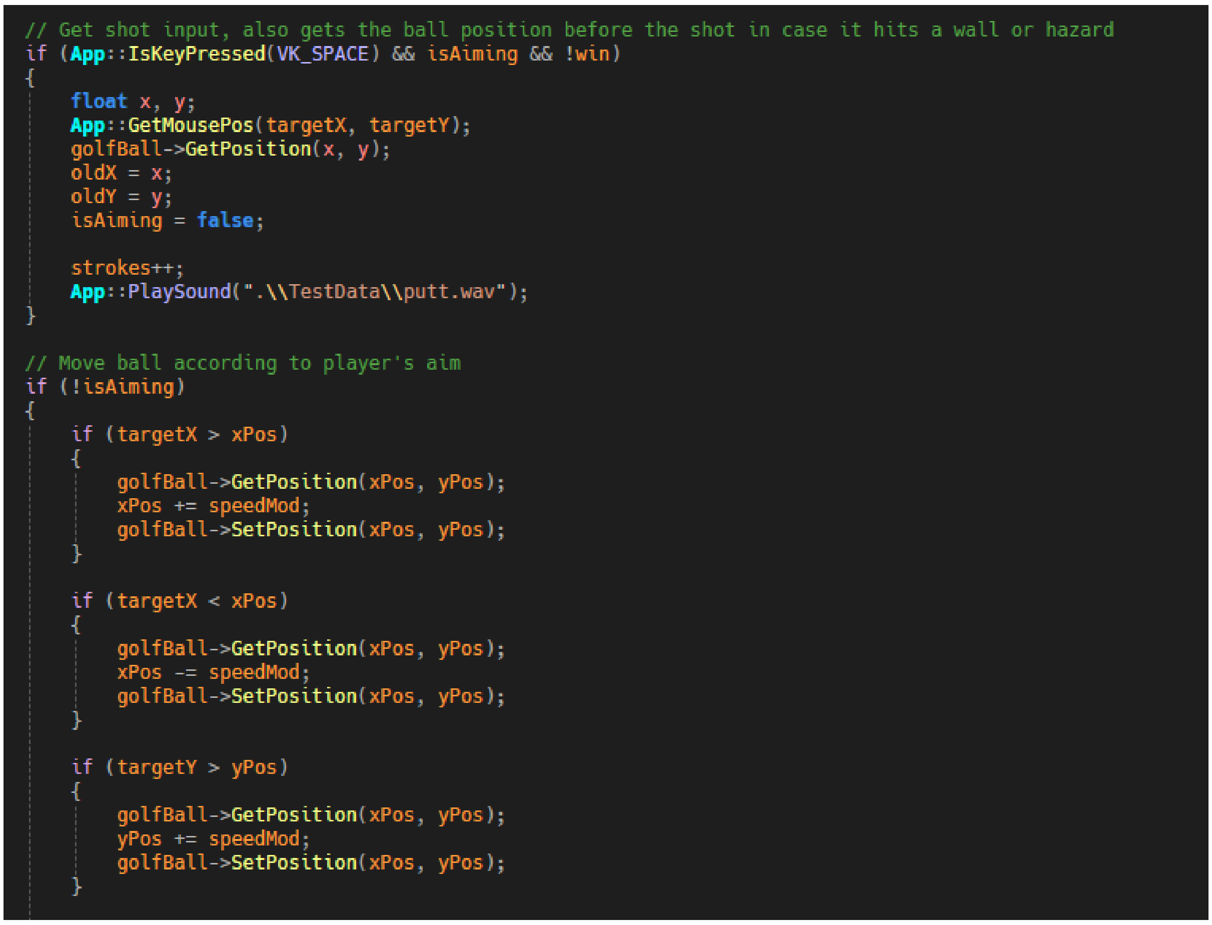
```cpp
bool checkHole(float x, float y)
{
    return std::sqrt((x - holeX) * (x - holeX) + (y - holeY) * (y - holeY)) < holeRadius;
}

// Clamps the value to determine a wall hit
#undef max
#undef min
float clampValue(float value, float minVal, float maxVal)
{
    return std::max(minVal, std::min(value, maxVal));
}

// Returns true if the ball hits a wall
bool checkWall(float x, float y, float radius, Wall wall)
{
    float closestX = clampValue(x, wall.x, wall.x + wall.width);
    float closestY = clampValue(y, wall.y, wall.y + wall.height);

    float dx = x - closestX;
    float dy = y - closestY;

    return (dx * dx + dy * dy) <= (radius * radius);
}
```

How the game checks to see if the player hits anything



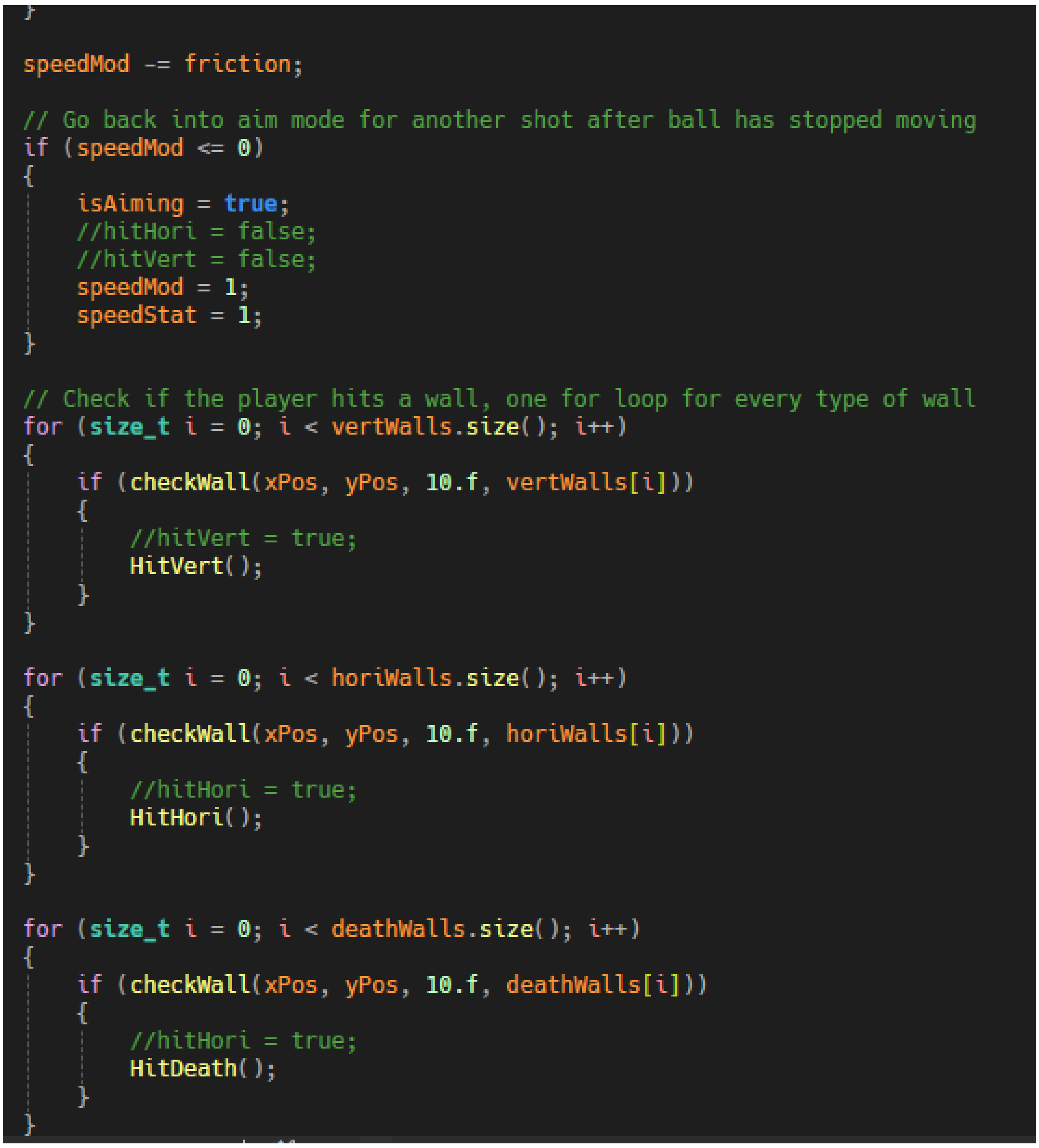
```cpp
// Adjusts power of the shot
if (App::IsKeyPressed('F') && toggleSpeed && isAiming)
{
    if (speedMod <= 2)
    {
        speedMod += 0.2f;
        speedStat++;
    }
    else
    {
        speedMod = 1;
        speedStat = 1;
    }

    toggleSpeed = false;
}

if (!App::IsKeyPressed('F'))
{
    toggleSpeed = true;
}
```

Adjusting power of shots



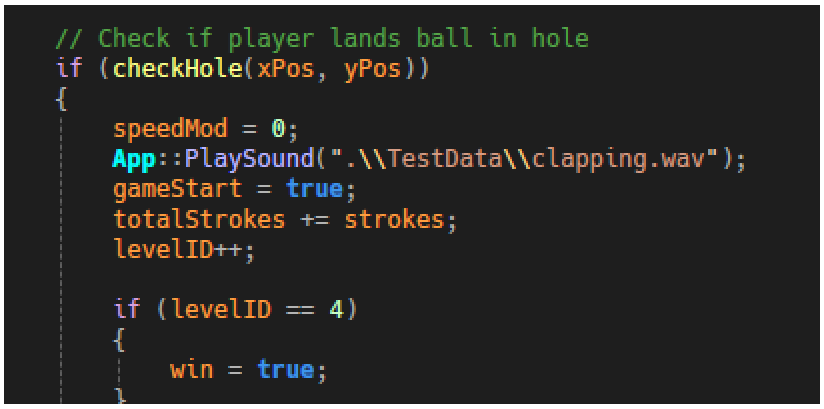
```
// Get shot input, also gets the ball position before the shot in case it hits a wall or hazard
if (App::IsKeyPressed(VK_SPACE) && isAiming && !win)
{
    float x, y;
    App::GetMousePos(targetX, targetY);
    golfBall->GetPosition(x, y);
    oldX = x;
    oldY = y;
    isAiming = false;

    strokes++;
    App::PlaySound(".\\TestData\\putt.wav");
}

// Move ball according to player's aim
if (!isAiming)
{
    if (targetX > xPos)
    {
        golfBall->GetPosition(xPos, yPos);
        xPos += speedMod;
        golfBall->SetPosition(xPos, yPos);
    }

    if (targetX < xPos)
    {
        golfBall->GetPosition(xPos, yPos);
        xPos -= speedMod;
        golfBall->SetPosition(xPos, yPos);
    }

    if (targetY > yPos)
    {
        golfBall->GetPosition(xPos, yPos);
        yPos += speedMod;
        golfBall->SetPosition(xPos, yPos);
    }
```

Aiming and launching the ball



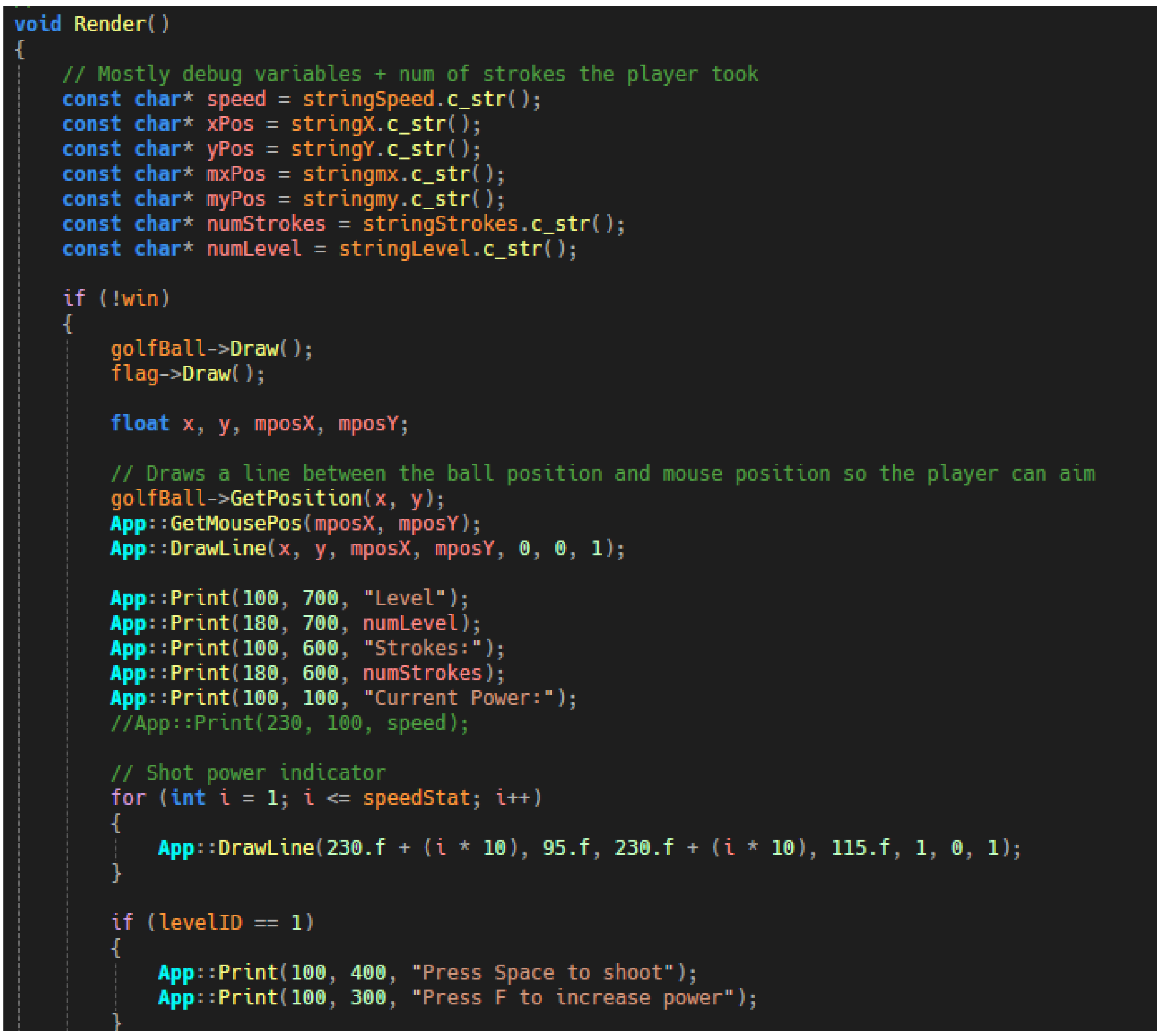
```
}

speedMod -= friction;

// Go back into aim mode for another shot after ball has stopped moving
if (speedMod <= 0)
{
    isAiming = true;
    //hitHori = false;
    //hitVert = false;
    speedMod = 1;
    speedStat = 1;
}

// Check if the player hits a wall, one for loop for every type of wall
for (size_t i = 0; i < vertWalls.size(); i++)
{
    if (checkWall(xPos, yPos, 10.f, vertWalls[i]))
    {
        //hitVert = true;
        HitVert();
    }
}

for (size_t i = 0; i < horiWalls.size(); i++)
{
    if (checkWall(xPos, yPos, 10.f, horiWalls[i]))
    {
        //hitHori = true;
        HitHori();
    }
}

for (size_t i = 0; i < deathWalls.size(); i++)
{
    if (checkWall(xPos, yPos, 10.f, deathWalls[i]))
    {
        //hitHori = true;
        HitDeath();
    }
}
```

Reducing speed as time goes on and for loops to check for collisions with walls



```
// Check if player lands ball in hole
if (checkHole(xPos, yPos))
{
    speedMod = 0;
    App::PlaySound(".\\TestData\\clapping.wav");
    gameStart = true;
    totalStrokes += strokes;
    levelID++;

    if (levelID == 4)
    {
        win = true;
```

Check if the player lands in the hole



```
void Render()
{
    // Mostly debug variables + num of strokes the player took
    const char* speed = stringSpeed.c_str();
    const char* xPos = stringX.c_str();
    const char* yPos = stringY.c_str();
    const char* mxPos = stringmx.c_str();
    const char* myPos = stringmy.c_str();
    const char* numStrokes = stringStrokes.c_str();
    const char* numLevel = stringLevel.c_str();

    if (!win)
    {
        golfBall->Draw();
        flag->Draw();

        float x, y, mposX, mposY;

        // Draws a line between the ball position and mouse position so the player can aim
        golfBall->GetPosition(x, y);
        App::GetMousePos(mposX, mposY);
        App::DrawLine(x, y, mposX, mposY, 0, 0, 1);

        App::Print(100, 700, "Level");
        App::Print(180, 700, numLevel);
        App::Print(100, 600, "Strokes:");
        App::Print(180, 600, numStrokes);
        App::Print(100, 100, "Current Power:");
        //App::Print(230, 100, speed);

        // Shot power indicator
        for (int i = 1; i <= speedStat; i++)
        {
            App::DrawLine(230.f + (i * 10), 95.f, 230.f + (i * 10), 115.f, 1, 0, 1);
        }

        if (levelID == 1)
        {
            App::Print(100, 400, "Press Space to shoot");
            App::Print(100, 300, "Press F to increase power");
        }
```

Changing ints and floats into printable strings and drawing levels and objects