tools.cpp

```
#include "headers.hpp"
#include "tools.hpp"



v2i min(const v2i& first, const v2i& second)
{
    return v2i(std::min(first.x, second.x), std::min(first.y,
    second.y));
}

v2i max(const v2i& first, const v2i& second)
{
    return v2i(std::max(first.x, second.x), std::max(first.y,
    second.y));
}

float mag(const v2f& vec)
{
    return sqrtf(vec.x * vec.x + vec.y * vec.y);
}

v2f norm(const v2f& vec)
{
    return vec / mag(vec);
}

float lerp(float a, float b, float t)
{
    return a * (1.0 - t) + (b * t);
}

sf::Color lerp(sf::Color c1, sf::Color c2, float t)
{
    return sf::Color(
        t * c1.r + (1 - t) * c2.r,
        t * c1.g + (1 - t) * c2.g,
        t * c1.b + (1 - t) * c2.b
    );
}
float angleBetweenVectors(const v2f& v1, const v2f& v2)
{
    float nigga = (v1.x * v2.x + v1.y * v2.y) / (mag(v1) * mag(v2));

    cout << "nig: " << (nigga) << " angle: ";
    if (nigga > 0)
        return acos(nigga);

    return acos(nigga);
}



bool inBounds(const v2f& box_pos, const v2f& box_size, const v2i& pos)
```

```
{
    if (pos.x < box_pos.x || pos.x > box_pos.x + box_size.x) {
        return false;
    }

    // Check if the point's y coordinate is within the box's vertical
    bounds.
    if (pos.y < box_pos.y || pos.y > box_pos.y + box_size.y) {
        return false;
    }

    // If both checks pass, the point is within the box.
    return true;
}


vector<string> split(const string& str)
{

    std::istringstream iss(str);
    vector<string> tokens;
    string token;

    // Split the string by spaces and store each token in a vector
    while (std::getline(iss, token, '~')) {
        tokens.push_back(token);
    }

    return tokens;
}


TextBox::TextBox(const v2f& pos, const v2f& size, const string& str,
const sf::Font& font)
    : position(pos), size(size), text_string(str), text(str, font, 30),
    shadow(size), cursor(v2f(1.5f, 30))
{
    text.setFillColor(sf::Color::Black);
    text.setPosition(position + text_offset);

    shadow.setPosition(position);
    shadow.setFillColor(sf::Color(0, 0, 0, 70));

    cursor.setFillColor(sf::Color::Black);
    cursor.setSize(v2f(1.5, 30));
}

string TextBox::getString()
{
    return text_string;
}

void TextBox::draw(sf::RenderWindow& window, bool is_focused)
```

```
{
    text.setString(text_string);

    if (hidden)
    {
        string hashed(text_string.size(), '*');
        text.setString(hashed);
    }


    window.draw(text);


    if (!is_focused)
    {
        window.draw(shadow);
        return;
    }

    if (cursor_visible)
    {
        cursor.setPosition(text.getPosition() +
            v2f(text.getGlobalBounds().getSize().x + 6, 4));
        window.draw(cursor);


    }


    if (cursor_timer.getElapsedTime().asMilliseconds() > 500)
    {
        cursor_visible ^= true;
        cursor_timer.restart();
    }


}

void TextBox::turnOnCursor()
{
    cursor_visible = true;
    cursor_timer.restart();
}

void TextBox::addText(const string& added_text)
{
    turnOnCursor();

    text_string = text_string + added_text;
    if (text_string.size() > 15)
    {
        text_string = text_string.substr(0, 15);
    }
```

```
}

void TextBox::backspace(int backspace_counter)
{
    turnOnCursor();

    if (backspace_counter < 0) // ctrl backspace
    {
        text_string = "";
        return;
    }


    text_string = (text_string.substr(0, text_string.size() -
    backspace_counter));
}

void TextBox::clearText()
{
    text_string = "";
}

bool TextBox::inBox(const v2i& pos)
{
    return inBounds(position, size, pos);
}
```