```cpp
#include "ColorChooser.h"

/**
 * @brief Construct a new Color Chooser:: Color Chooser object
 *
 * @param screen Pointer to the screen object to be used
 * @param ts Pointer to the touchscreen object to be used
 * @param bg Current background color
 * @param box Current box color
 * @param text Current text color
 */
ColorChooser::ColorChooser(Elegoo_TFTLCD *screen, TouchScreen *ts, uint16_t bg,
    uint16_t box, uint16_t text) {
    _screen = screen;
    _ts = ts;
    _bg = bg;
    _box = box;
    _text = text;
}

/**
 * @brief
 * Draws the color chose menu on the display and let's the user choose one of them by
     clicking it.
 *
 * @return int Chosen color
 */
int ColorChooser::choose() {
    // Set variable for selection
    uint16_t selection = -1;

    // Draw the menu to the screen
    draw();

    // Wait for input
    while (true) {
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If there was a user input, analyse it.
        if (p.z > 20) {
            // Parse coordinates from touchscreen to screen
            Coords parse;
            parse.x = (int) ((p.x - 120) / 3.5);
            parse.y = (int) ((p.y - 70) / 2.66);

            // Get selection from parsed coordinates
            selection = getSelection(parse);

            // If valid selection occured, break
            if (selection != -2) break;
        }

        // Delay, in order not to get the same input more than once.
        delay(5);
    }

    // Return the selection
    return selection;
}

/**
 * @brief
 * Analyses coordinates passed into function and returns selected color
 *
 * @param coords Coordinates of touchpoint, parsed to screen coordinates
 * @return uint16_t chosen Color or 0, if cancelled or -2, if selection was not valid
 */
uint16_t ColorChooser::getSelection(Coords coords) {
    int column, row;

    Serial.println("CC: X: " + String(coords.x) + ", Y: " + String(coords.y));

    // Check, if touchpoint is at height of cancel button
    if (320 - coords.y > 250) return 0;

    // If coordinates are out of valid ranges, return failure marker
    if (coords.x < 35 || coords.x > 215) return -2;
    if (coords.x < 135 && coords.x > 100) return -2;

    // If x is less than 100, it has to be first column (index 0), otherwise second
        column (index 1)
    column = coords.x < 100 ? 0 : 1;

    // Invert coordinates (320-y, because screen is 320 high) and divide by 60,
        because that's the
    // height of the boxes, to get the row
    row = (int)((320 - coords.y) / 60);

    // Variable basically only exists, so that we don't try to reach indices the array
        does not have
    // This works, because the first column has indices 0,2,4,6 and second has 1,3,5,7
    int safeHandler = column + (row*2);

    // If selection > 7, set selection to failure, otherwise set selection to selected
        color
    uint16_t selection = (safeHandler > 7 ? -2 : _colors[safeHandler]);

    switch (selection) {
        case BLUE: Serial.println("CC Selection: BLUE"); break;
        case RED: Serial.println("CC Selection: RED"); break;
        case GREEN: Serial.println("CC Selection: GREEN"); break;
        case CYAN: Serial.println("CC Selection: CYAN"); break;
        case MAGENTA: Serial.println("CC Selection: MAGENTA"); break;
        case YELLOW: Serial.println("CC Selection: YELLOW"); break;
        case WHITE: Serial.println("CC Selection: WHITE"); break;
        case BLACK: Serial.println("CC Selection: BLACK"); break;
        default: Serial.println("CC No selection"); break;
    }

    return selection;
}

/**
 * @brief
 * Draws the menu to the screen
 *
 */
void ColorChooser::draw() {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    _screen->fillScreen(_bg);

    // Box Size: 70. Boxes start at y = 10 and end at y = 240, box offset 10
    // Draw boxes, fill with corresponding color
    for (int i = 0; i < 8; i++) {
        _screen->fillRect((i % 2 == 0 ? 35 : 135),
            ((int) (i / 2))*50 + 10,
            70,
            50,
            _colors[i]
        );
    }

    // Draw cancel button
    _screen->drawRect(10,250,220,60, _box);
    _screen->setCursor(50,270);
    _screen->setTextSize(3);
    _screen->setTextColor(_text);
    _screen->print("Zurueck");
}


/**
* @class ColorChooser
* @author Gerrit Koppe, Thilo Drehlmann
*
* @brief
* Class to implement a Menu for choosing a color.
* Public method ColorChooser::choose prints the menu to display and returns the color
    the user clicked on.
*
* @version 1
* @date 2022
*/

#ifndef colorchooser
#define colorchooser

#include <Elegoo_GFX.h> // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>
#include "Messenger.h"

#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN   0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE  0xFFFF

typedef struct Coords {
    int x;
    int y;
} Coords;
```

```
class ColorChooser {
    public:
        ColorChooser(Elegoo_TFTLCD *screen, TouchScreen *ts, uint16_t bg, uint16_t
            box, uint16_t text);

        int choose();
    private:
        uint16_t _bg, _box, _text;


        uint16_t _colors[8] = {
            BLUE,
            RED,
            GREEN,
            CYAN,
            MAGENTA,
            YELLOW,
            WHITE,
            BLACK
        };

        void draw();
        uint16_t getSelection(Coords coords);

        Elegoo_TFTLCD *_screen;
        TouchScreen *_ts;
};

#endif


#include "Messenger.h"

/**
    * @brief Construct a new Messenger:: Messenger object
    *
    * @param screen Pointer to the LCD
    * @param ts Pointer to the touchscreen
    * @param keys pointer to the virtual keys
    */
Messenger::Messenger(Elegoo_TFTLCD *screen, TouchScreen *ts, VKeys *keys, Radio *r) {
    _screen = screen;
    _ts = ts;
    _keys = keys;
    _radio = r;

    // Set defaults
    _background = BLACK;
    _textColor = WHITE;
    _boxColor = WHITE;
    _keyColor = WHITE;
    _textSize = TEXTMEDIUM;
    _menuBorderOffset = 10;
    _minTouch = 3;

    // Set Defaults on the keys
    _keys->setKeyColor(_keyColor, _background);
    _keys->setTextColor(BLACK, _background);
}

/**
    * @brief
    * Initializes the Messenger object and starts the infinite loop
    *
    */
void Messenger::init(void) {
    int selection = MAINMENU;
    String message = ""; // Message to send

    Serial.println("Screen Width: " + String(_screen->width()) + ", Screen Height: " +
        String(_screen->height()));

    while (true) {
        // Pinmodes because TS and Screen share these pins
        pinMode(A2, OUTPUT);
        pinMode(A3, OUTPUT);
        _screen->fillScreen(_background);

        // Start the corresponding menu
        switch (selection) {
            case MAINMENU: selection = mainMenu(); break;
            case WRITEMESSAGE: message = writeMessage(); selection = MAINMENU; break;
            case OPTS: optsMenu(); selection = MAINMENU; break;
            case READ: readMenu(); selection = MAINMENU; break;
            default: selection = mainMenu();
        }

        if (message != "") {
            // TODO get real message here and send it, this stuff is just for testing!
            _radio->sendMessage(message);
            message = "";
        }
    }
}

/**
    * @deprecated
    * @brief
    *
    */
void Messenger::reset(void) {
    _screen->fillScreen(WHITE);
    init();
}



/************************* PRIVATE ***************************/

/************************* PRIVATE MENU FUNCTIONS **************************/

/**
    * All menu functions work the exact same, therefore not all of them are commented.
        This comment
    * shows the function of menu methods
    *
    *  1. Configure the menu with a menu struct
```

```
 *      1.1 menuStart: Y-Value where the menu starts on display
 *      1.2 menuThickness: How thick are the menu entries
 *      1.3 menuOffset: how far are entries apart
 *      1.4 header: Header text of the menu
 *      1.5 extraText: smaller text below menu header
 *      1.6 entries[5]: Text shown in the menu entries
 *  2. Draw the menu
 *  3. Infinite loop
 *      3.1 Check, if messages are available on the air
 *      3.2 Get touchpoint
 *      3.3 analyse touchpoint
 *      3.4 Call method or return value according to selection and break
 */

/**
 * @brief
 * Shows the main menu and handles the input
 *
 * @return int Selection in menu
 */
int Messenger::mainMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);

    // Build menu defintion
    Menu menu;
    menu.menuStart = 100;
    menu.menuThickness = 60;
    menu.menuOffset = 10;
    menu.header = "Messenger";
    menu.extraText = checkCache();
    menu.entries[0] = String("Schreiben");
    menu.entries[1] = String("Lesen");
    menu.entries[2] = String("Optionen");
    menu.entries[3] = String("\0");
    menu.entries[4] = String("\0");

    // Draw menu
    drawMenu(menu);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") {
            Serial.println(inc);
            return -42;
        }
        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 3, parseCoords(p));;
            Serial.println("Selection: " + String(selection));

            // Return the seection
            switch (selection) {
                case -1: break;
                case 1: return WRITEMESSAGE;
                case 2: return READ;
                case 3: return OPTS;
                default: break;
            }
        }
        delay(10);
    }

    Serial.println("Returning from mainMenu");
}

/**
    * @brief
    * Options menu
    *
    */
void Messenger::optsMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 40;
    menu.menuOffset = 20;
    menu.header = "Optionen";
    menu.entries[0] = String("Farben");
    menu.entries[1] = String("Tastatur");
    menu.entries[2] = String("Distanz");
    menu.entries[3] = String("Zurueck");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);
        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 4, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            switch (selection) {
                case -1: break;
                case 1: colorMenu(); return;
                case 2: keysMenu(); return;
                case 3: distanceMenu(); return;
                case 4: return;
                default: break;
            }
        }
        delay(10);
    }

    Serial.println("Returning from optsMenu");
}

void Messenger::distanceMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 80;
    menu.menuThickness = 30;
    menu.menuOffset = 10;
    menu.header = "Distanz";
    menu.entries[0] = String("Sehr nah");
    menu.entries[1] = String("Nah");
    menu.entries[2] = String("Weit");
    menu.entries[3] = String("Sehr weit");
    menu.entries[4] = String("Zurueck");
    menu.entries[5] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);
        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 5, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            switch (selection) {
                case -1: break;
                case 1: _radio->setPALevel("MIN"); return;
                case 2: _radio->setPALevel("LOW"); return;
                case 3: _radio->setPALevel("HIGH"); return;
                case 4: _radio->setPALevel("MAX"); return;
                case 5: return;
                default: break;
            }
        }
        delay(10);
    }

    Serial.println("Returning from Distance");
}

/**
    * @brief
    * Submenu for all things keyboard related
    *
    */
void Messenger::keysMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 40;
    menu.menuOffset = 20;
    menu.header = "Tastatur";
    menu.entries[0] = String("Tastenfarbe");
    menu.entries[1] = String("Buchs.-Far.");
    menu.entries[2] = String("Stil");
    menu.entries[3] = String("Zurueck");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);

        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 4, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            ColorChooser cc(_screen, _ts, _background, _boxColor, _textColor);

            switch (selection) {
                case -1: break;
                case 1: _keys->setKeyColor(cc.choose(), _background); return;
                case 2: _keys->setTextColor(cc.choose(), _background); break;
                case 3: _keys->setStyle(keyStyleMenu());
                case 4: return;
                default: break;
            }
        }
        delay(10);
    }

    Serial.println("Returning from optsMenu");
}

/**
    * @brief
    * Menu for selecting layout of keys
    *
    * @return String Layout of the keys
    */
String Messenger::keyStyleMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 40;
    menu.menuOffset = 20;
    menu.header = "Layout";
    menu.entries[0] = String("QWERTZ");
    menu.entries[1] = String("QWERTY");
    menu.entries[2] = String("ABCDE");
    menu.entries[3] = String("Zurueck");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);

        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 4, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            switch (selection) {
                case -1: break;
                case 1: return String("QWERTZ");
                case 2: return String("QWERTY");
                case 3: return String("ABCDE");
                case 4: return String("\0");
                default: break;
            }
        }
        delay(10);
    }
    Serial.println("Returning from optsMenu");
}

/**
    * @brief
    * Submenu for different color menus
    *
    */
void Messenger::colorMenu(void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 40;
    menu.menuOffset = 20;
    menu.header = "Farben";
    menu.entries[0] = String("Hintergrund");
    menu.entries[1] = String("Text");
    menu.entries[2] = String("Boxen");
    menu.entries[3] = String("Zurueck");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);

        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 4, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            ColorChooser cc(_screen, _ts, _background, _boxColor, _textColor);

            switch (selection) {
                case -1: break;
                case 1: setBackground(backGroundColorMenu()); return;
                case 2: setTextColor(cc.choose()); return;
                case 3: setBoxColor(cc.choose()); return;
                case 4: return;
                default: break;
            }
        }
        delay(10);
    }
    Serial.println("Returning from colorMenu");
}
```

```
/**
    * @brief
    * Menu to set either light or dark mode
    *
    * @return uint16_t WHITE or BLACK
    */
uint16_t Messenger::backGroundColorMenu(void) {
    Menu menu;
    menu.menuStart = 60;
    menu.menuThickness = 70;
    menu.menuOffset = 20;
    menu.header = "Hintergrund";
    menu.entries[0] = String("Dark");
    menu.entries[1] = String("Light");
    menu.entries[2] = String("Zurueck");
    menu.entries[3] = String("\0");
    menu.entries[4] = String("\0");

    drawMenu(menu);
    delay(100);

    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);

        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        // If touch was recognized
        if (p.z > _minTouch) {
            // Get selected menu poin
            selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 3, parseCoords(p));

            Serial.println("Selection: " + String(selection));

            switch (selection) {
                case -1: break;
                case 1: return BLACK;
                case 2: return WHITE;
                case 3: return 0;
                default: break;
            }
        }
        delay(10);
    }

    Serial.println("Returning from bgMenu");
}

/************************ PRIVATE MENU DRAW ***************************/

/**
    * @brief
    * Draws the menu from definition passed in argument
    *
    * @param menu Definition of the menu
    */
void Messenger::drawMenu(Menu menu) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    _screen->fillScreen(_background);

    // Print header
    _screen->setTextColor(RED);
    _screen->setTextSize(_textSize);
    _screen->setCursor(20,10);
    _screen->print(menu.header);

    _screen->setTextColor(_textColor);

    // Print menu boxes and texts
    for (int i = 0; i < 5; i++) {
        if (menu.entries[i] == "\0") continue;

        _screen->drawRect(_menuBorderOffset,
            (int16_t) menu.menuStart + i*(menu.menuThickness + menu.menuOffset),
            _screen->width() - 2*_menuBorderOffset,
            (int16_t) menu.menuThickness,
            _boxColor
        );

        _screen->setCursor(20,menu.menuStart + i*(menu.menuThickness +
            menu.menuOffset) + ((int) (menu.menuThickness / _textSize)));
        _screen->print(menu.entries[i]);
    }

    // Print extra text, if it exists
    if (menu.extraText != "") {
        int y = (10 + menu.menuStart) / 2;
        _screen->setTextSize(_textSize - 1);
        _screen->setTextColor(_textColor);
        _screen->setCursor(20,y);
        _screen->print(menu.extraText);
    }
}

/***************************** PRIVATE READ MENU STUFF ***************************/

/**
    * @brief
    * Menu for reading received messages
    */
void Messenger::readMenu(void) {
    // Couunter for which message screen is on
    int currentMessage = 0;

    // Set pinmodes
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);

    // Draw the menu
    drawReadMenu();
    // Set text config
    _screen->setCursor(0,10);
    _screen->setTextSize(_textSize);
    _screen->setTextColor(_textColor);

    // If there are no new messages, return
    if (checkCache() == "") {
        _screen->print("Keine neuen\nNachrichten");
        delay(3000);
        return;
    }

    // Print first message
    _screen->print(_messages[currentMessage]);
    while (true) {
        String inc = receiveMessage();
        if (inc != "\0") Serial.println(inc);

        int selection = -1;

        // Get touchpoint
        digitalWrite(13, HIGH);
        TSPoint p = _ts->getPoint();
        digitalWrite(13, LOW);

        if (p.z > _minTouch) {
            // Get selected menu poin
            // selection = getSelection(menu.menuStart, menu.menuThickness,
                menu.menuOffset, 3, parseCoords(p));
            selection = readMenuSelection(parseCoords(p));

            Serial.println("Selection: " + String(selection));

            switch (selection) {
                case -1: break;
                case 1: switchMessageToRead(&currentMessage, false, false); break; //
                    Message back
                case 2: switchMessageToRead(&currentMessage, true, false); break; //
                    Message forward
                case 3: deleteMessage(currentMessage);
                    switchMessageToRead(&currentMessage, false, true); break; // Delete
                    current
                case 4: return;
                default: break;
            }
            delay(10);
        }

        continue;
    }
}

/**
    * @brief
    * Method for drawing the read menu on screen
    */
void Messenger::drawReadMenu(void) {
    int msgNo = 0;
    for (int i = 0; i < 3; i++) {
        if (_messages[i] != String("\0")) msgNo++;
    }
    // Set pinmodes
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);

    // Fill screen and set text config
    _screen->fillScreen(_background);
    _screen->setTextSize(_textSize -1);
    _screen->setTextColor(_textColor);

    // Draw function buttons
    _screen->drawRect(_menuBorderOffset, 180, 100, 50, _boxColor);
    _screen->drawRect(_screen->width() - _menuBorderOffset - 100, 180, 100, 50,
        _boxColor);
    _screen->drawRect(_menuBorderOffset, 240, 100, 50, _boxColor);
    _screen->drawRect(_screen->width() - _menuBorderOffset - 100, 240, 100, 50,
        _boxColor);

    // Fill function buttons
    _screen->setCursor(_menuBorderOffset, 160);
    if (msgNo > 0) _screen->print(String(msgNo) + " Nachr.; Nr. 1");
    _screen->setCursor(_menuBorderOffset + 40, 195);
    _screen->print("<");

    _screen->setCursor(_screen->width() - _menuBorderOffset - 100 + 48, 195);
    _screen->print(">");

    _screen->setCursor(_menuBorderOffset + 10, 255);
    _screen->print("DEL");

    _screen->setCursor(_menuBorderOffset + 140, 255);
    _screen->print("BACK");
}

/**
    * @brief
    * Writes new message to display, after it has been switched
    *
    * @param msgCounter Pointer to the current messagenumber
    * @param plus Message +1 or -1?
    * @param afterDelete Triggered after a message has been deleted?
    */
void Messenger::switchMessageToRead(int *msgCounter, bool plus, bool afterDelete) {
    // Set pinmodes
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);

    // Overprint old message and set text configuration
    _screen->fillRect(0,0,_screen->width(), 175, _background);
    _screen->setCursor(0,0);
    _screen->setTextColor(_textColor);
    _screen->setTextSize(_textSize);

    // Set tempCounter for later comparison
    int tempCounter = *msgCounter;
    int noMessages = 0;
    // Check, how many empty messages are in cache
    for (int i = 0; i < 3; i++) {
        if (String(_messages[i]) == String("\0")) noMessages++;
    }

    // If message cache is empty, set msgCounter back to 0 and print, that there are
        no messages
    if (noMessages == 3) {
        *msgCounter = 0;
        _screen->print("Keine neuen\nNachrichten");
        return;
    }

    // Modify tempcounter
    plus ? tempCounter++ : tempCounter--;
    if (tempCounter > 2) {
        tempCounter = 2;
    } else if (tempCounter < 0) {
        tempCounter = 0;
    }

    Serial.println("Tempcounter: " + String(tempCounter));
    // If method was called after deletion, do it a little differently
    if (afterDelete) {
        for (int i = 0; i < 3; i++) {
            if (String(_messages[i]) == String("\0")) {
                tempCounter = i-1;
                if (tempCounter > 2 || tempCounter < 0) tempCounter = 0;
                break;
            }
        }
    }

    Serial.println("Tempcounter after deletecheck: " + String(tempCounter));

    // Handle tempcounter being outside of array indices
    if (tempCounter > 2 || tempCounter < 0) return;

    if (String(_messages[tempCounter]) == String("\0")) tempCounter = 0;

    Serial.println("Tempcounter after boundcheck: " + String(tempCounter));

    // Set msg counter and print message
    *msgCounter = tempCounter;

    if (_messages[*msgCounter] == String("\0")) return;
    else {
        _screen->print(_messages[*msgCounter]);
        _screen->setCursor(_menuBorderOffset, 160);
        _screen->setTextSize(_textSize - 1);
        _screen->print(String(3 - noMessages) + " Nachr.; Nr. " + String(tempCounter +
            1));
    }
}

/**
    * @brief
    * Gets selection in the read menu
    *
    * @param parse x and y coordinates of pressed point
    * @return int 1: top left, 2: top right, 3: bottom left, 4: bottom right
    */
int Messenger::readMenuSelection(ScreenParse parse) {
    int selection = 1;
    int y = 320 - parse.y;

    Serial.println("Y: " + String(y) + "; parse.y: " + String(parse.y));

    // Check, if point is outside of menu
    if (y < 180) return -1;
    if (y > 290) return -1;
    if (parse.x < _menuBorderOffset) return -1;
    if (parse.x > _screen->width() - _menuBorderOffset) return -1;

    // Get row and add 2, if second row. Works because of numbering of menu entries
    selection += (y > 235 ? 2 : 0);

    // Get column and add 1, if second column
    selection += (parse.x > (_screen->width() / 2) ? 1 : 0);

    return selection;
}

/************************ PRIVATE SPECIALS ***************************/

/**
    * @brief
    * Parses Coordinates from touchpoint to screen coordinates
    *
    * @param p TSPoint, Point the user touched on display
    * @return ScreenParse Struct containing parsed screen coordinates
    */
ScreenParse Messenger::parseCoords(TSPoint p) {
    ScreenParse parse;

    // Touch X: [120,940], Screen X: [0,240]
    // Parse X
    parse.x = (int) ((p.x - 120) / 3.5);

    // Touch Y: [70,920], Screen Y: [0,320]
    // Parse Y
    parse.y = (int) ((p.y - 70) / 2.66);
    return parse;
}

/**
    * @brief
    * Gets number of menupoint selected
    *
    * @param menuStart y-Value, where the menu starts
    * @param menuThickness how thick one menupoint is
    * @param menuOffset how far the menu entries are apart
    * @param entries number of entries
    * @param parse x and y values of the pressed point
    * @return int number of the menu entry selected
    */
int Messenger::getSelection(int menuStart, int menuThickness, int menuOffset, int
    entries, ScreenParse parse) {
    int x, y, selection;

    // Screen and touchscreen have different values, parse ts to screen
    // Get x and y coordinates from point
    x = (int) parse.x;
    // Invert display
    y = (int) 320 - parse.y;

    Serial.println("X: " + String(x) + ", Y: " + String(y));

    // Check if touch occured out of menu
    if (y < menuStart || y > menuStart + (entries*menuThickness) +
        (entries*menuOffset)) return -1;
    if (x < _menuBorderOffset || x > (_screen->width() - _menuBorderOffset)) return -1;

    // translate y to the start of the menu and get selection
    y -= menuStart;
    selection = (int) (y / (menuThickness+menuOffset)) + 1;
    selection = selection;

    if (selection < 1 || selection > entries) return -1;
    return selection;
}

/**
    * @brief
    * Sets private field _background to color that you want
    *
    * @param color Color you want to set the background to
    */
void Messenger::setBackground(uint16_t color) {
    if (color == _textColor) _textColor = (color == WHITE ? BLACK : WHITE);
    if (color == _boxColor) _boxColor = (color == WHITE ? BLACK : WHITE);

    _background = color;
}

/**
    * @brief Prints the current message to the display
    *
    * @param msg {String} Current message
    */
void Messenger::printMessageOnDisplay(String msg) {
    // Set pinmodes
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);

    // Print a filled rect over old message
    _screen->fillRect(0,0,_screen->width(), 140, _background);

    // Set config values for text and print it
    _screen->setCursor(0,0);
    _screen->setTextSize(_textSize);
    _screen->setTextColor(_textColor);
    _screen->print(msg);
}
```

```
/**
   * @brief
   * Function to write a message that you want to send
   *
   * @return String Written message
   */
String Messenger::writeMessage(void) {
   // Initialize the keypad (draw it on screen)
   if (_keys->getSpecial()) _keys->switchKeys();
   else _keys->init();

   // Initialize variables and get initial touchpoint
   String msg = "", oldChar = "";
   TSPoint oldP = _ts->getPoint();

   // Clicks: Counter for how long since current char has been pressed. It works, I
       forgot how
   int clicks = 0;

   while (true) {
      TSPoint p;

      do {
         // Get touchpoint, until the point differs from last touched point on
             display
         digitalWrite(13, HIGH);
         p = _ts->getPoint();
         digitalWrite(13, LOW);
      } while (oldP.x == p.x && oldP.y == p.y);

      // Save new point in old point for later comparison
      oldP.x = p.x;
      oldP.y = p.y;

      if (p.z > _minTouch) {
         if (msg.length() >= 80) {
            printMessageOnDisplay("Maximal 80\nZeichen!");
            delay(1000);
            printMessageOnDisplay(msg);
            continue;
         }
         // hasChanged: records, whether the new char differs from last char
         bool hasChanged = false;

         // Get character from pressed point
         String newChar = String(_keys->getInputChar(p));

         // Check, if pressed char differs from last pressed char
         if (newChar == oldChar) {
            // if they don't differ, count how many times we have been at this point
            if (clicks < 3) {
               clicks++;
               continue;
            } else {
               // else reset all variables
               oldChar = "";
               newChar = "";
               clicks = 0;
            }
            } else {
                // Unless a special key was pressed, add new char to message
                if (newChar != "~" && newChar != "s" && newChar != "DONE" && newChar !=
                    "BACK") {
                    msg += newChar;
                }

                // record, that something has changed
                oldChar = newChar;
                hasChanged = true;
            }

            // If delete was pressed, delete last char from message
            if (newChar == "~") {
                msg = msg.substring(0,msg.length()-1);
                printMessageOnDisplay(msg);
            } else if (newChar == "s") {
                // small s means, that we want to switch to special keys (numbers and
                    non-alpha-numeric)
                _keys->switchKeys();
            } else if (newChar == "DONE") {
                // If done, break infinite loop to return the message
                break;
            } else if (newChar == "BACK") {
                // If cancelled, reset message and break infinite loop
                msg = "";
                break;
            } else {
                if (hasChanged) {
                    // If the input char has changed, print message to display
                    printMessageOnDisplay(msg);
                }
            }

            Serial.println(msg);
        }

        // Without delay, this thing notices touches 3-5 Times per Touch
        delay(10);
    }

    return msg;
}

/**
    * @brief
    * Gets message that was sent from other device
    *
    * @return String received message
    */
String Messenger::receiveMessage() {
    // Try to get message and return, if message is \0
    String msg = _radio->receiveMessage();
    if (msg == "\0") return "\0";

    // Cache that message and return it
    Serial.println("Received message: " + String(msg));
    cacheMessage(msg);
    return msg;
}

/********************************** PRIVATE CACHE METHODS
    *********************************************/

/**
    * @brief
    * Caches incoming messages in message cache
    *
    * @param msg String to be cached
    */
void Messenger::cacheMessage(String msg) {
    // Check, how many messages are in cache
    int counter = 0;
    for (int i = 0; i < 3; i++) {
        if (_messages[i] != "\0") {
            counter++;
            continue;
        } else {
            break;
        }
    }

    // If message already is in cache, return
    for (int i = 0; i < counter; i++) {
        if (msg == _messages[i]) return;
    }

    // If there are less than 3 messages, just cache the newest message on last
        position
    if (counter < 3) {
        _messages[counter] = msg;
        return;
    } else {
        // If there are 3 messages, clear the last message
        _messages[0] = "\0";

        // Push every other message one to the back
        for (int i = 1; i < 3; i++) {
            _messages[i - 1] = _messages[i];
        }

        // Put new Message as first in the array
        _messages[2] = msg;
        return;
    }
}

/**
    * @brief
    * Clears the whole message cache. Every message is deleted!
    *
    */
void Messenger::clearMessageCache() {
    for (int i = 0; i < 3; i++) {
        _messages[i] = "\0";
    }
}
```

```
/**
    * @brief
    * Deletes a certain message from cache.
    *
    * @param num Number of message to be deleted. Has to be between 0 and 2!
    */
void Messenger::deleteMessage(int num) {
    if (num < 0 || num > 2) return;

    // Delete requested message from cache
    _messages[num] = String("\0");

    // For every message, that is listed after the requested message: move upwards in
        array
    for (int i = num; i < 3; i++) {
        if (i != 2) {
            if (_messages[i + 1] != String("\0")) {
                _messages[i] = String(_messages[i+1]);
                _messages[i+1] = String("\0");
            }
        }
    }
}

/**
    * @brief
    * Checks, whether there are messages in cache and returns a String for the main
        Menu screen.
    *
    * @return String
    */
String Messenger::checkCache() {
    // Counter for the number of messages
    int number = 0;

    // Iterate through all messages
    for (int i = 0; i < 3; i++) {
        // Increment counter for every message, that isn't \0
        if (String(_messages[i]) != String("\0")) {
            Serial.println("Cache Analysis, Message " + String(i) + ": " +
                String(_messages[i]));
            number++;
        }
    }

    if (number == 0) return "";

    // Return message
    return String(number) + String(" Nachrichten");
}

#ifndef messenger
#define messenger

#include <Elegoo_GFX.h> // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>
```

```
#include "VKeys.h"
#include "ColorChooser.h"
#include "Radio.h"

#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN   0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE  0xFFFE

#define MAINMENU 0
#define WRITEMESSAGE 1
#define READ 2
#define OPTS 3

#define COLOR_OPTS 21
#define CHANNEL_OPTS 22
#define TSIZE_OPTS 23
#define BACK_OPTS 24

#define TEXT_COLOR_OPTS 211
#define BG_COLOR_OPTS 212
#define KEY_COLOR_OPTS 213

#define TEXTSMALL 2
#define TEXTMEDIUM 3
#define TEXTLARGE 4

typedef struct ScreenParse {
    int x;
    int y;
} ScreenParse;

typedef struct Menu {
    int menuStart;
    int menuThickness;
    int menuOffset;
    String entries[5];
    String header;
    String extraText;
} Menu;

class Messenger {
    public:
        Messenger(Elegoo_TFTLCD *screen, TouchScreen *ts, VKeys *keys, Radio *r);

        // MAIN FUNCTION
        void init(void);

        // RESET
        void reset(void);

        // GETTERS
        uint16_t getKeyColor(void) { return _keyColor; };
        uint16_t getTextColor(void) { return _textColor; };
        uint16_t getTextSize(void) { return _textSize; };
        uint16_t getBoxColor(void) { return _boxColor; };
        uint16_t getBackground(void) { return _background; };

    private:
        // CONFIG FIELDS
        uint16_t _keyColor, _textColor, _background, _textSize, _boxColor;
        int _menuBorderOffset, _minTouch;

        byte adress[6];

        // PARTS
        VKeys *_keys;
        Elegoo_TFTLCD *_screen;
        TouchScreen *_ts;
        Radio *_radio;

        String _messages[3] = {
            "\0", "\0", "\0"
        };

        // MENU FUNCTIONS
        int mainMenu(void),
            getSelection(int menuStart, int menuThickness, int menuOffset, int
                entries, ScreenParse parse);

        uint16_t backGroundColorMenu(void);

        void    optsMenu(void),
                colorMenu(void),
                keysMenu(void),
                distanceMenu(void);

        String keyStyleMenu(void);

        // READ
        void    readMenu(void),
                drawReadMenu(void),
                switchMessageToRead(int *msgCounter, bool plus, bool afterDelete);

        int readMenuSelection(ScreenParse parse);

        // MENU DRAW FUNCTIONS
        void drawMenu(Menu menu);

        // SETTERS
        void setKeyColor(uint16_t color) { _keyColor = (color == _background ?
            _keyColor : color); };
        void setTextColor(uint16_t color) { _textColor = (color == _background ?
            _keyColor : color); };
        void setTextSize(uint16_t size) { _textSize = size; };
        void setBoxColor(uint16_t color) { _boxColor = (color == _background ?
            _keyColor : color); };
        void setBackground(uint16_t color);

        // SPECIALS
        ScreenParse parseCoords(TSPoint p);

        String writeMessage(void);
```

```
        void    printMessageOnDisplay(String msg);

        String receiveMessage();

        // CACHE
        void    cacheMessage(String msg),
                clearMessageCache(),
                deleteMessage(int num);

        String checkCache();
};

#endif


#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Arduino.h>

// Local includes
#include "Radio.h"

/************************* PUBLIC *****************************/
Radio::Radio(uint16_t ce, uint16_t csn) {
    Serial.println("We in Radio Constructor");
    _ce = ce;
    _csn = csn;
    _readingPipe = 1;

    _listening = true;
    _level = RF24_PA_MIN;

    _antenna = RF24(_ce, _csn);
    Serial.println("We out of Radio Constructor");
}

/**
    * @brief
    * Initializes the object. Should be called before doing anything else!
    *
    */
void Radio::init() {
    _antenna.begin();
    byte _adress[6] = "00001";
    _antenna.openReadingPipe(_readingPipe, _adress);
    _antenna.setPALevel(_level);
    _antenna.startListening();
}

/**
    * @brief
    * Sets PA-Level / Reach of antenna
    *
    * @param level MIN, LOW, HIGH or MAX, depending on the level that is supposed to
        be used
    */
void Radio::setPALevel(String level) {
    if (level == "MIN") {
        _level = RF24_PA_MIN;
    } else if (level == "LOW") {
        _level = RF24_PA_LOW;
    } else if (level == "HIGH") {
        _level = RF24_PA_HIGH;
    } else if (level == "MAX") {
        _level = RF24_PA_MAX;
    } else {
        _level = RF24_PA_MIN;
    }

    _antenna.setPALevel(_level);
    Serial.println("Internal PA-Level: " + String(getPALevel()));
}

/**
    * @brief
    * Switches from listening to sending and the other way around
    *
    */
void Radio::switchState() {
    byte _adress[6] = "00001";
    // Invert listening
    _listening = !_listening;

    // If listening: Prepare antenna for listening. Otherwise: Prepare antenna for
        sending
    if (_listening) {
        _antenna.openReadingPipe(_readingPipe, _adress);
        _antenna.startListening();
    } else {
        _antenna.stopListening();
        _antenna.openWritingPipe(_adress);
    }

    _antenna.setPALevel(_level);
}

bool Radio::available(void) {
    return _antenna.available();
}

/**
    * @brief
    * Checks, whether there are devices available nearby
    *
    * @return true If there are devices available nearby
    * @return false otherwise
    */
bool Radio::checkNearbyDevices(void) {
    char buffer[128];
    String msg;

    // Switch from listening to sending
    if (_listening) switchState();

    // Convert _acknowledge message to char array
    convertStringToCharArray(_acknowledge, buffer);

    // Send acknowledge message 3 times
    for (int i = 0; i < 3; i++) {
        _antenna.write(buffer, sizeof(buffer));
        delay(10);
    }

    // Switch back to listening
    switchState();

    // Wait for payload being available
    int counter = 0;
    while (!_antenna.available()) {
        counter++;
        if (counter > 9) return false;
        delay(10);
        continue;
    }

    // Read payload and convert it back to String
    _antenna.read(buffer, sizeof(buffer));
    msg = convertCharArrayToString(buffer);

    // If payload was the acknowlege String, that means there are devices nearby.
    if (msg == _acknowledge) return true;

    return false;
}

/**
    * @brief
    * Receives message from radio frequency
    *
    * @return String received message or \0, if nothing was gotten
    */
String Radio::receiveMessage(void) {
    char buffer[128] = "";
    String msg = "";

    // Switch from sending to listening
    if (!_listening) switchState();

    // If no message is available, return universal break character
    if (!_antenna.available()) return "\0";
    Serial.println("Receiving: Antenna available");

    // Read available payload
    _antenna.read(&buffer, sizeof(buffer));

    for (int i = 0; i < 128; i++) {
        msg += String(buffer[i]);
        if (buffer[i] == '\0') break;
    }

    Serial.println("Receiving: Received char Array: " + msg);

    // Convert received message to String
    // msg = convertCharArrayToString(buffer);

    // THIS IS DEPRECATED AND USELESS
    if (msg == String(_acknowledge)) {
        return "\0";
    }

    // THIS MOST PROBABLY IS DEPRECATED AND USELESS AS WELL
    if (msg == String(_jam)) {
        acknowledge();
        return String(_jam);
    }

    return msg;
}

/**
    * @brief
    * Sends acknowledgement message
    *
    */
void Radio::acknowledge(void) {
    char buffer[128];

    // Switch from listening state to sending state
    if (_listening) switchState();

    // convert the acknowledge payload to a char array
    convertStringToCharArray(_acknowledge, buffer);

    // Send acknowledgement 3 times, to make sure partner gets the message
    for (int i = 0; i < 3; i++) {
        _antenna.write(buffer, sizeof(buffer));
        delay(10);
    }
}

/**
    * @brief
    * Sends message given in param
    *
    * @param msg Message to be sent
    * @return true If sending was successful
    * @return false
    * TODO return false somewhere, why should it be bool otherwise
    */
bool Radio::sendMessage(String msg) {
    String sendTest = "";
    bool tx_ok, tx_fail, rx_ready, test = true;
    char buffer[128];
    msg.toCharArray(buffer, msg.length() + 1);

    Serial.println("Sending message: " + msg);
    Serial.println("Listening State: " + String((_listening ? "listening" :
        "writing")));

    if (_listening) switchState();

    Serial.println("Listening State: " + String((_listening ? "listening" :
        "writing")));

    // convertStringToCharArray(msg, buffer);
    for (int i = 0; i < msg.length(); i++) {
        sendTest += String(buffer[i]);
    }
    Serial.println("Sending: Char Arr after conversion: " + sendTest);

    test = _antenna.write(&buffer, sizeof(buffer));

    if (!test) {
        Serial.println("Sending unseccessful");
        _antenna.whatHappened(tx_ok, tx_fail, rx_ready);
        Serial.println("OK: " + String((tx_ok ? "Yes" : "No")) + ", FAIL: " +
            String((tx_fail ? "Yes" : "No")) + ", READY: " + String((rx_ready ? "Yes"
            : "No")));
    }

    Serial.println("Returning from sending");
    switchState();
    return true;
}

/**
    * @brief
    * Converts a given String s to a char array and saves it to given pointer a
    *
    * @param s Strung to be converted to char array
    * @param a Pointer to char array
    */
void Radio::convertStringToCharArray(String s, char a[128]) {
    int counter = 0;

    // Iterate through entire String and add character to char array
    for (int i = 0; i < s.length(); i++) {
        a[i] = s.charAt(i);
        counter++;
    }

    // Fill up char array with empty values
    for (int i = counter; i < 128; i++) {
        a[i] = '\0';
    }

    Serial.println("Char Arr after conversion: " + String(a));
}

/**
    * @brief
    * Converts a given char array to a String and returns it
    *
    * @param a Char array to be converted to a String
    * @return String
    */
String Radio::convertCharArrayToString(char a[128]) {
    String s = "";

    // Iterate through the whole array and add every character to empty string
    for (int i = 0; i < 128; i++) {
        if (a[i] == '\0') continue;
        s += String(a[i]);
    }
    Serial.println("String after conversion: " + s);

    return s;
}


#ifndef radio_h
#define radio_h

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Arduino.h>

class Radio {
    public:
        Radio(uint16_t ce, uint16_t csn);

        uint16_t   getCE (void) { return _ce; };
        uint16_t   getCSN (void) { return _csn; };

        uint8_t    getPALevel(void) { Serial.println("PA Level: " +
            String(_antenna.getPALevel()));return _level; };
        void       setPALevel(String level);

        int        getReadingPipe (void) { return _readingPipe; };

        void       setCE (uint16_t ce) { _ce = ce; };
        void       setCSN (uint16_t csn) { _csn = csn; };
        void       setListening (bool listen) { _listening = listen; };

        bool       getListening (void) { return _listening; };

        void       switchState(void);

        String     receiveMessage(void);
        bool       available(void);
        void       init();

        bool       sendMessage(String msg);

    private:
        uint16_t   _ce, _csn, _readingPipe, _paLevel;

        uint8_t    _level = RF24_PA_MIN;

        bool       _listening = true;

        String     _jam = "1337";
        String     _acknowledge = "4269";

        RF24       _antenna;

        bool       checkNearbyDevices(void);
        void       acknowledge(void);

        void       convertStringToCharArray(String s, char a[256]);
        String     convertCharArrayToString(char a[256]);
};
```

```
#endif


#include "VKeys.h"
#include <Elegoo_GFX.h> // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>

/**
    * @brief Construct a new VKeys::VKeys object
    *
    * @param style Style of the keyboard. Possible Configurations: QWERTZ, QWERTY,
        ABCDE
    * @param keyColor Color of the keys.
    * @param textColor Color of the text on the keys
    * @param tScreen Pointer to the touchscreen to be used
    */
VKeys::VKeys (String style, uint16_t keyColor, uint16_t textColor, Elegoo_TFTLCD
    *tScreen) {
    _style = style;
    _screen = tScreen;
    _keyColor = keyColor;
    _textColor = textColor;
    _textSize = 4;
    _special = false;

    // Set rows to passed configuration
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            if (_style == "QWERTZ") _rows[i][j] = _QWERTZ[i][j];
            else if (_style == "QWERTY") _rows[i][j] = _QWERTY[i][j];
            else if (_style == "ABCDE") _rows[i][j] = _ABCDE[i][j];
        }
    }
}

/**
    * @brief
    * Initializes and draws the keyboard on the touchscreen
    *
    */
void VKeys::init (void) {
    pinMode(A2, OUTPUT);
    pinMode(A3, OUTPUT);
    _screen->setTextColor(_textColor);
    _screen->setTextSize(_textSize - 1);

    // Spacebar, Done and Back
    _screen->fillRect(0, _screen->height() - (_kHeight - 1), (_screen->width() / 4) -
        1, _kHeight - 1, _keyColor);
    _screen->fillRect((_screen->width() / 4), _screen->height() - (_kHeight - 1),
        2*(_screen->width() / 4) - 1, _kHeight - 1, _keyColor);
    _screen->fillRect(3*(_screen->width() / 4), _screen->height() - (_kHeight - 1),
        (_screen->width() / 4) - 1, _kHeight - 1, _keyColor);
    _screen->setCursor((int16_t)(_screen->width() / 4), (int16_t)(_screen->height() -
        (_kHeight - 4)));

    _screen->setCursor((_screen->width() / 4) + 15, 291);
    _screen->print((char*)"SPACE");
    _screen->setTextSize(_textSize - 2);
    _screen->setCursor(3*(_screen->width() / 4) + 8, 291);
    _screen->print((char*)"Send");

    _screen->setCursor(8, 291);
    _screen->print((char*)"Back");

    // Draw the keyboard
    _screen->setTextSize(_textSize);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            // Draw the button
            _screen->fillRect(j*(_kWidth + 1), (int) _screen->height() -
                (4-i)*(_kHeight + 1), _kWidth, _kHeight, _keyColor);

            // Set cursor and draw character into current button
            _screen->setCursor((int16_t)(j*(_kWidth + 1) +2), (int16_t)((int)
                _screen->height() - (4-i)*(_kHeight + 1) + 2));

            // If special is selected, draw the special characters, otherwise draw
                alphabetic characters from current config
            if (_special) {
                if (_specialChars[i][j] != '\0') {
                    _screen->print(_specialChars[i][j]);
                }
            } else {
                _screen->print(_rows[i][j]);
            }
        }
    }
}

/**
    * @brief
    * Resets virtual keyboard. Useful if you want to change the color of the keys.
    */
void VKeys::reset (void) {
    // Default is QWERTZ
    if (_style == "QWERTY") {
        _rows[0][5] = 'Y';
        _rows[2][1] = 'Z';
    } else {
        _rows[0][5] = 'Z';
        _rows[2][1] = 'Y';
    }
}

/**
    * @brief
    *
    * @param point
    * @return char
    */
String VKeys::getInputChar(TSPoint point) {
    int16_t x,y;

    x = point.x;
    y = point.y;
    return String(getCharFromCoords(x, y));
}

void VKeys::setStyle (String style) {
    if (style == "\0") return;
    _style = style;

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 10; j++) {
            if (_style == "QWERTZ") _rows[i][j] = _QWERTZ[i][j];
            else if (_style == "QWERTY") _rows[i][j] = _QWERTY[i][j];
            else if (_style == "ABCDE") _rows[i][j] = _ABCDE[i][j];
        }
    }
}

void VKeys::switchKeys() {
    _special = !_special;
    init();
}


/**************************************************** PRIVATE
    ***************************************************/

/**
    * @brief
    * Returns selected char by calculating, which column and row was pressed from
        passed x and y coords
    *
    * @param x X coordinate from touchscreen selection
    * @param y Y coordinate from touchscreen selection
    * @return String Selected character parsed to a string, BACK if back button was
        selected and DONE, if done button was selected
    */
String VKeys::getCharFromCoords(int16_t x, int16_t y) {
    //TODO Check, what the hell is returned in this method and why. For Some reason, a
        fucking 2 is returned at one point.
    int row, column;

    // Complete Y: [70,920], range 850, keyboard [70,475], keyboard range of 405, 4
        rows so 405/4 = 101.25
    if (y > 475) return String('\0');
    row = (int) ((y - 70)/ 101.25);

    // row 0 is spacebar, therefore return escaped space immediately, as nothing else
        is there.
    if (row == 0) {
        int tempX = (int)((x - 120) / 3.5);

        if (tempX < (_screen->width() / 4)) {
            return String("BACK");
        } else if (tempX > 3*(_screen->width() / 4)) {
            return String("DONE");
        }
        return String(" ");
    }

    if (row > 3) return String('2');
    // X Range ca. 120 - 920 for 10 columns, therefore 800 / 10 = 80
    column = (int) ((x - 120)/ 80);

    if (column < 11 && column >= 0 && row < 4 && row >0) return String(_special ?
        _specialChars[(3-row)][column] : _rows[(3-row)][column]);

    return String('-');
}


#ifndef vkeys
#define vkeys

#include <Elegoo_GFX.h> // Core graphics library
#include <Elegoo_TFTLCD.h> // Hardware-specific library
#include <TouchScreen.h>

#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN   0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE  0xFFFE

const char _QWERTZ[3][10] = {
           {
               'Q','W','E','R','T','Z','U','I','O','P'
           },
           {
               'A','S','D','F','G','H','J','K','L','~'
           },
           {
               's','Y','X','C','V','B','N','M','?','#'
           }
};

const char _QWERTY[3][10] = {
           {
               'Q','W','E','R','T','Y','U','I','O','P'
           },
           {
               'A','S','D','F','G','H','J','K','L','~'
           },
           {
               's','Z','X','C','V','B','N','M','?','#'
           }
};

const char _ABCDE[3][10] = {
           {
               'A','B','C','D','E','F','G','H','I','J'
           },
           {
               'K','L','M','N','O','P','Q','R','S','~'
           },
           {
               's','T','U','V','W','X','Y','Z','?','#'
```

```
          }
};

class VKeys {
    public:
        VKeys (String style, uint16_t keyColor, uint16_t textColor, Elegoo_TFTLCD
            *tScreen);

        void setKWidth (int width) { _kWidth = width; };
        int getKWidth () { return _kWidth; };

        void setKHeight (int height) { _kHeight = height; };
        int getKHeight () { return _kHeight; };

        void setSpecial (bool special) { _special = special; };
        bool getSpecial () { return _special; };

        void setStyle (String style);
        String getStyle () { return _style; };

        void setKeyColor (uint16_t color, uint16_t _background) { _keyColor = (color
            == _background ? _keyColor : color);};
        uint16_t getKeyColor (void) { return _keyColor; };

        void setTextColor (uint16_t color, uint16_t _background) { _textColor = (color
            == _background ? _textColor : color); };
        uint16_t getTextColor (void) { return _textColor; };

        void   init (void),
               reset (void);

        void switchKeys();

        String getInputChar(TSPoint point);

    private:
        String _style;
        uint16_t _keyColor;
        uint16_t _textColor;
        int _textSize = 1;

        int _kHeight = 40 - 2;
        int _screenX = 240;
        int _kWidth = (_screenX / 10) - 1;
        int _screenY = 300;

        bool _special = false;

        char _rows[3][10] = {
            {
                'Q','W','E','R','T','Z','U','I','O','P'
            },
            {
                'A','S','D','F','G','H','J','K','L','#'
            },
            {
                's','Y','X','C','V','B','N','M','?','*'
            }
        };
        char _specialChars[3][10] = {
            {
                ’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,’0’
            },
            {
                ’@’,34,’#’,’$’,’%’,’&’,39,’\0’,’\0’,’~’
            },

            {
                ’s’,’\0’,’\0’,’\0’,’\0’,’\0’,’\0’,’\0’,’\0’,’\0’
            }
        };

        String getCharFromCoords (int16_t x, int16_t y);

        Elegoo_TFTLCD *_screen;
};

#endif
```