ЗВІТ

з лабораторної роботи **№4**

з дисципліни “Автоматизоване проектування комп’ютерних систем”

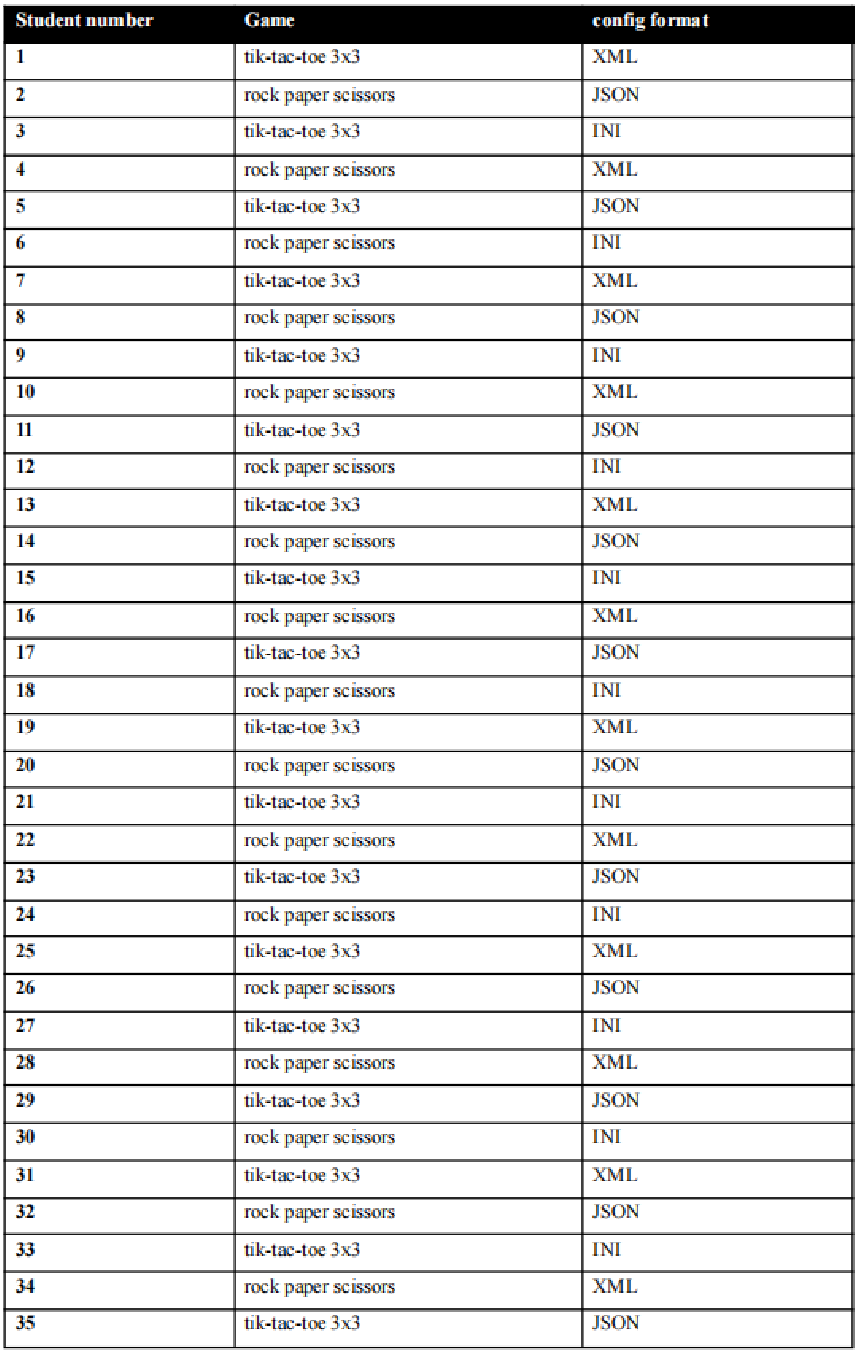
Виконала:
ст.гр. КІ-401
Кріль Катерина
Перевірив:
Федак П. Р.

Львів - 2024

Ініціалізація Git репозиторію

Хід роботи:

1. Додайте коментарі doxygen для всіх публічних функцій, класів, властивостей, полів...
2. Згенеруйте документацію на основі коментарів doxygen
3. Обов'язкові кроки



Student number	Game	config format
1	tik-tac-toe 3x3	XML
2	rock paper scissors	JSON
3	tik-tac-toe 3x3	INI
4	rock paper scissors	XML
5	tik-tac-toe 3x3	JSON
6	rock paper scissors	INI
7	tik-tac-toe 3x3	XML
8	rock paper scissors	JSON
9	tik-tac-toe 3x3	INI
10	rock paper scissors	XML
11	tik-tac-toe 3x3	JSON
12	rock paper scissors	INI
13	tik-tac-toe 3x3	XML
14	rock paper scissors	JSON
15	tik-tac-toe 3x3	INI
16	rock paper scissors	XML
17	tik-tac-toe 3x3	JSON
18	rock paper scissors	INI
19	tik-tac-toe 3x3	XML
20	rock paper scissors	JSON
21	tik-tac-toe 3x3	INI
22	rock paper scissors	XML
23	tik-tac-toe 3x3	JSON
24	rock paper scissors	INI
25	tik-tac-toe 3x3	XML
26	rock paper scissors	JSON
27	tik-tac-toe 3x3	INI
28	rock paper scissors	XML
29	tik-tac-toe 3x3	JSON
30	rock paper scissors	INI
31	tik-tac-toe 3x3	XML
32	rock paper scissors	JSON
33	tik-tac-toe 3x3	INI
34	rock paper scissors	XML
35	tik-tac-toe 3x3	JSON

Табл.1 Завдання

Виконання роботи:

Прокоментований файл game.h:

```
#ifndef GAME_H
#define GAME_H

#include <stdio.h>
#include <stdlib.h>

#include "project.h"
#include "types.h"
#include "constants.h"
#include "communication.h"

//! Indicates if the game is running
static u8 game_running = 1u;

//! Indicates if it's the man's turn
static u8 man_turn = 1u;

//! Array to store figure positions on the board
static u8 figures[CELLS_NUMBER * CELLS_NUMBER];

//! Type of the game being played
static game_type type = UNKNOWN;

//! Packet for game type
static u8 game_packet[GAME_TYPE_PACKET_LENGTH];

//! Packet for man vs man game type
static u8 man_vs_man_packet[GAME_TYPE_PACKET_LENGTH];

//! Packet for man vs AI game type
static u8 man_vs_ai_packet[GAME_TYPE_PACKET_LENGTH];

//! Packet for AI vs AI game type
static u8 ai_vs_ai_packet[GAME_TYPE_PACKET_LENGTH];

//! Packet for resetting the game
static u8 reset_packet[GAME_TYPE_PACKET_LENGTH];

//! Packet to indicate a win
static u8 win_packet[CELLS_NUMBER * CELLS_NUMBER];

//! Acknowledgement packet
static u8 ack_packet[CELLS_NUMBER * CELLS_NUMBER];

/**
 * @brief Clears the game board.
 */
static inline void clear_board(void)
{
    for (u8 i = 0u; i < CELLS_NUMBER; i++)
        for (u8 j = 0u; j < CELLS_NUMBER; j++)
        {
            figures[i * CELLS_NUMBER + j] = EMPTY;
        }
}

/**
 * @brief Sets a figure on the game board.
 *
 * @param x The x-coordinate on the board.
 * @param y The y-coordinate on the board.
 */
static inline void set_figure(u8 x, u8 y)
{
    static u8 figure_index = 0u;

    if (figures[x * CELLS_NUMBER + y] == EMPTY)
    {
        figures[x * CELLS_NUMBER + y] = figure_index;
        figure_index = !figure_index;
    }
}

/**
 * @brief Initializes the packets for different game states.
 */
static inline void init_packets(void)
{
    for (u8 index = 0u; index < CELLS_NUMBER * CELLS_NUMBER; index++)
    {
        ack_packet[index] = ACK_PACKET_VALUE;
        win_packet[index] = NO_WINNER_PACKET_VALUE;
    }

    for (u8 index = 0u; index < GAME_TYPE_PACKET_LENGTH; index++)
    {
        game_packet[index] = UNKNOWN_PACKET_VALUE;
        man_vs_man_packet[index] = MAN_VS_MAN_PACKET_VALUE;
        man_vs_ai_packet[index] = MAN_VS_AI_PACKET_VALUE;
        ai_vs_ai_packet[index] = AI_VS_AI_PACKET_VALUE;
        reset_packet[index] = RESET_PACKET_VALUE;
    }
}

/**
 * @brief Sends an acknowledgement packet.
 *
 * @param packet The packet to acknowledge.
 */
static inline void send_ack(u8 *packet)
{
    return;

    send_message(ack_packet, CELLS_NUMBER * CELLS_NUMBER);
    receive_message(game_packet, GAME_TYPE_PACKET_LENGTH);

    while (memcmp(game_packet, packet, GAME_TYPE_PACKET_LENGTH) == 0)
    {
        send_message(ack_packet, CELLS_NUMBER * CELLS_NUMBER);
        receive_message(game_packet, GAME_TYPE_PACKET_LENGTH);
    }
}

/**
 * @brief Receives the game type from the communication channel.
```

```c
 */
static inline void recieve_game_type(void)
{
    u8 recieved = FALSE;

    while (!recieved)
    {
        receive_message(game_packet, GAME_TYPE_PACKET_LENGTH);

        if (memcmp(game_packet, man_vs_man_packet,
GAME_TYPE_PACKET_LENGTH) == 0)
        {
            type = MAN_VS_MAN;

            send_ack(man_vs_man_packet);
        }
        else if (memcmp(game_packet, man_vs_ai_packet,
GAME_TYPE_PACKET_LENGTH) == 0)
        {
            type = MAN_VS_AI;

            send_ack(man_vs_ai_packet);
        }
        else if (memcmp(game_packet, ai_vs_ai_packet,
GAME_TYPE_PACKET_LENGTH) == 0)
        {
            type = AI_VS_AI;

            send_ack(ai_vs_ai_packet);
        }
        else
            continue;

        recieved = TRUE;
    }
}

/**
 * @brief Resets the game to its initial state.
 */
static inline void game_reset(void)
{
    type = UNKNOWN;
    game_running = 1u;

    clear_board();
    init_packets();
    recieve_game_type();
}

/**
 * @brief Starts the game.
 */
static inline void game_start(void)
{
    communication_start();
    clear_board();
    init_packets();
    recieve_game_type();
}
```

```
/**
 * @brief Checks for events and handles them accordingly.
 */
static inline void check_events(void)
{
    receive_message(game_packet, GAME_TYPE_PACKET_LENGTH);

    if (memcmp(game_packet, reset_packet, GAME_TYPE_PACKET_LENGTH) ==
0)
    {
        send_ack(game_packet);
        game_reset();

        return;
    }
    if (!game_running)
    {
        send_message(win_packet, CELLS_NUMBER * CELLS_NUMBER);

        CyDelay(50u);

        return;
    }

    switch (type)
    {
        case MAN_VS_MAN:
        {
            if ((game_packet[0u] - 1u < CELLS_NUMBER) &&
(game_packet[1u] - 1u < CELLS_NUMBER))
            {
                send_ack(game_packet);
                set_figure(game_packet[0u] - 1u, game_packet[1u] -
1u);
                send_message(figures, 9u);
            }

            break;
        }
        case MAN_VS_AI:
        {
            if ((game_packet[0u] - 1u < CELLS_NUMBER) &&
(game_packet[1u] - 1u < CELLS_NUMBER) && man_turn)
            {
                send_ack(game_packet);
                set_figure(game_packet[0u] - 1u, game_packet[1u] -
1u);

                man_turn = !man_turn;
            }
            else if (!man_turn)
            {
                send_ack(game_packet);

                u8 x = rand() % 3;
                u8 y = rand() % 3;

                while (figures[x * CELLS_NUMBER + y] != EMPTY)
                {
                    x = rand() % 3;
                    y = rand() % 3;
                }

                set_figure(x, y);
                man_turn = !man_turn;

                CyDelay(50u);
            }

            send_message(figures, 9u);

            break;
        }
        case AI_VS_AI:
        {
            send_ack(game_packet);

            u8 x = rand() % 3;
            u8 y = rand() % 3;

            while (figures[x * CELLS_NUMBER + y] != EMPTY)
            {
                x = rand() % 3;
                y = rand() % 3;
            }

            set_figure(x, y);
            send_message(figures, 9u);

            CyDelay(50u);

        }
        case UNKNOWN:
        default:
        {
            send_ack(game_packet);
            break;
        }
    }
}

/**
 * @brief Checks if there is a winner in the game.
 */
static inline void check_win(void)
{
    for (u8 index = 0u; index < CELLS_NUMBER * CELLS_NUMBER; index++)
    {
        if (figures[index] == EMPTY)
        {
            game_running = 1u;

            break;
        }
        else
        {
            game_running = 0u;
        }
    }

    for (u8 i = 0u; i < 8u; i++)
    {
        u32 accumulator = 0u;

        for (u8 j = 0u; j < CELLS_NUMBER * CELLS_NUMBER; j++)
            accumulator += figures[j] & win_masks[i][j];

        if (accumulator == 0u)
        {
            game_running = 0u;

            for (u8 index = 0u; index < CELLS_NUMBER * CELLS_NUMBER;
index++)
                win_packet[index] = WIN_CROSS_PACKET_VALUE;
        }
        else if (accumulator == 3u)
        {
            game_running = 0u;

            for (u8 index = 0u; index < CELLS_NUMBER * CELLS_NUMBER;
index++)
                win_packet[index] = WIN_NOD_PACKET_VALUE;
        }
    }
}

/**
 * @brief Main game loop function.
 */
static inline void game_run(void)
{
    for (;;)
    {
        check_events();
        check_win();
    }
}

#endif
```

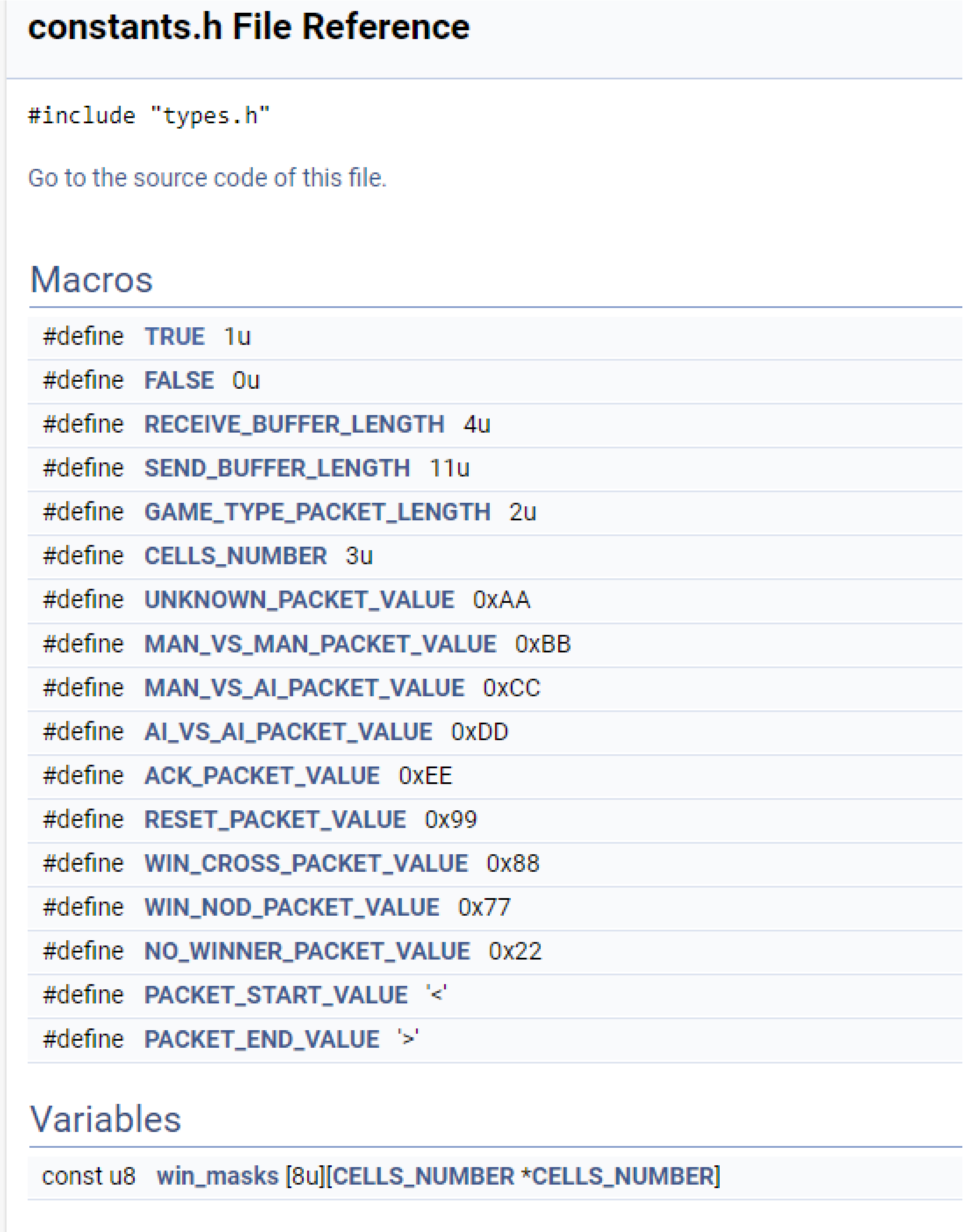
Прокоментований файл communication.h:

```
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <stdio.h>
#include <string.h>

#include "project.h"
#include "types.h"
#include "constants.h"

//! Buffer to receive communication data
static u8 communication_receive_buffer[RECEIVE_BUFFER_LENGTH];

//! Buffer to send communication data
static u8 communication_send_buffer[SEND_BUFFER_LENGTH];

/**
 * @brief Initializes the communication system.
 */
static inline void communication_start(void)
{
    UART_Start();
    setvbuf(stdin, NULL, _IONBF, 0);
}

/**
 * @brief Receives a message through the communication channel.
 *
 * @param buffer The buffer to store the received message.
 * @param length The length of the message to be received.
 */
static inline void receive_message(u8 *buffer, u8 length)
{
    if ((length + 2u > RECEIVE_BUFFER_LENGTH) || buffer == NULL)
return;

    UART_GetArrayBlocking(communication_receive_buffer, length + 2u);

    if ((communication_receive_buffer[0] == PACKET_START_VALUE)
        && (communication_receive_buffer[length + 1u] ==
PACKET_END_VALUE))
    {
        for (u8 index = 0u; index < length; index++)
            buffer[index] = communication_receive_buffer[index + 1u];

        memset(communication_receive_buffer, 0u,
RECEIVE_BUFFER_LENGTH);
    }
}

/**
 * @brief Sends a message through the communication channel.
 *
 * @param buffer The buffer containing the message to be sent.
 * @param length The length of the message to be sent.
 */

static inline void send_message(u8 *buffer, u8 length)
{
    if ((length + 2u > SEND_BUFFER_LENGTH) || buffer == NULL) return;

    communication_send_buffer[0] = PACKET_START_VALUE;
    communication_send_buffer[length + 1u] = PACKET_END_VALUE;

    for (u8 index = 1u; index < length + 1u; index++)
        communication_send_buffer[index] = buffer[index - 1u];

    UART_PutArrayBlocking(communication_send_buffer,
SEND_BUFFER_LENGTH);

    memset(communication_send_buffer, 0u, SEND_BUFFER_LENGTH);
}

#endif // COMMUNICATION_H
```

Прокоментований файл constants.h:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#include "types.h"

//! Boolean true value
#define TRUE                     1u

//! Boolean false value
#define FALSE                    0u

//! Length of the receive buffer
#define RECEIVE_BUFFER_LENGTH    4u

//! Length of the send buffer
#define SEND_BUFFER_LENGTH       11u

//! Length of the game type packet
#define GAME_TYPE_PACKET_LENGTH  2u

//! Number of cells in the game board
#define CELLS_NUMBER             3u

//! Value for unknown packets
#define UNKNOWN_PACKET_VALUE     0xAA

//! Value for man vs man game type packets
#define MAN_VS_MAN_PACKET_VALUE  0xBB

//! Value for man vs AI game type packets
#define MAN_VS_AI_PACKET_VALUE   0xCC

//! Value for AI vs AI game type packets
#define AI_VS_AI_PACKET_VALUE    0xDD

//! Value for acknowledgement packets
#define ACK_PACKET_VALUE         0xEE

//! Value for reset packets
#define RESET_PACKET_VALUE       0x99

//! Value for win packets indicating a cross win
#define WIN_CROSS_PACKET_VALUE   0x88

//! Value for win packets indicating a nod win
#define WIN_NOD_PACKET_VALUE     0x77

//! Value for packets indicating no winner
#define NO_WINNER_PACKET_VALUE   0x22

//! Start value for packets
#define PACKET_START_VALUE       '<'

//! End value for packets
#define PACKET_END_VALUE         '>'

//! Win masks used to determine winning conditions
const u8 win_masks[8u][CELLS_NUMBER * CELLS_NUMBER] = {
    {0xFF, 0xFF, 0xFF, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0xFF, 0xFF, 0xFF, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0xFF, 0xFF, 0xFF},
    {0xFF, 0, 0, 0xFF, 0, 0, 0xFF, 0, 0},
    {0, 0xFF, 0, 0, 0xFF, 0, 0, 0xFF, 0},
    {0, 0, 0xFF, 0, 0, 0xFF, 0, 0, 0xFF},
    {0xFF, 0, 0, 0, 0xFF, 0, 0, 0, 0xFF},
    {0, 0, 0xFF, 0, 0xFF, 0, 0xFF, 0, 0},
};

#endif // CONSTANTS_H
```

Приклад документації



constants.h File Reference

```
#include "types.h"
```

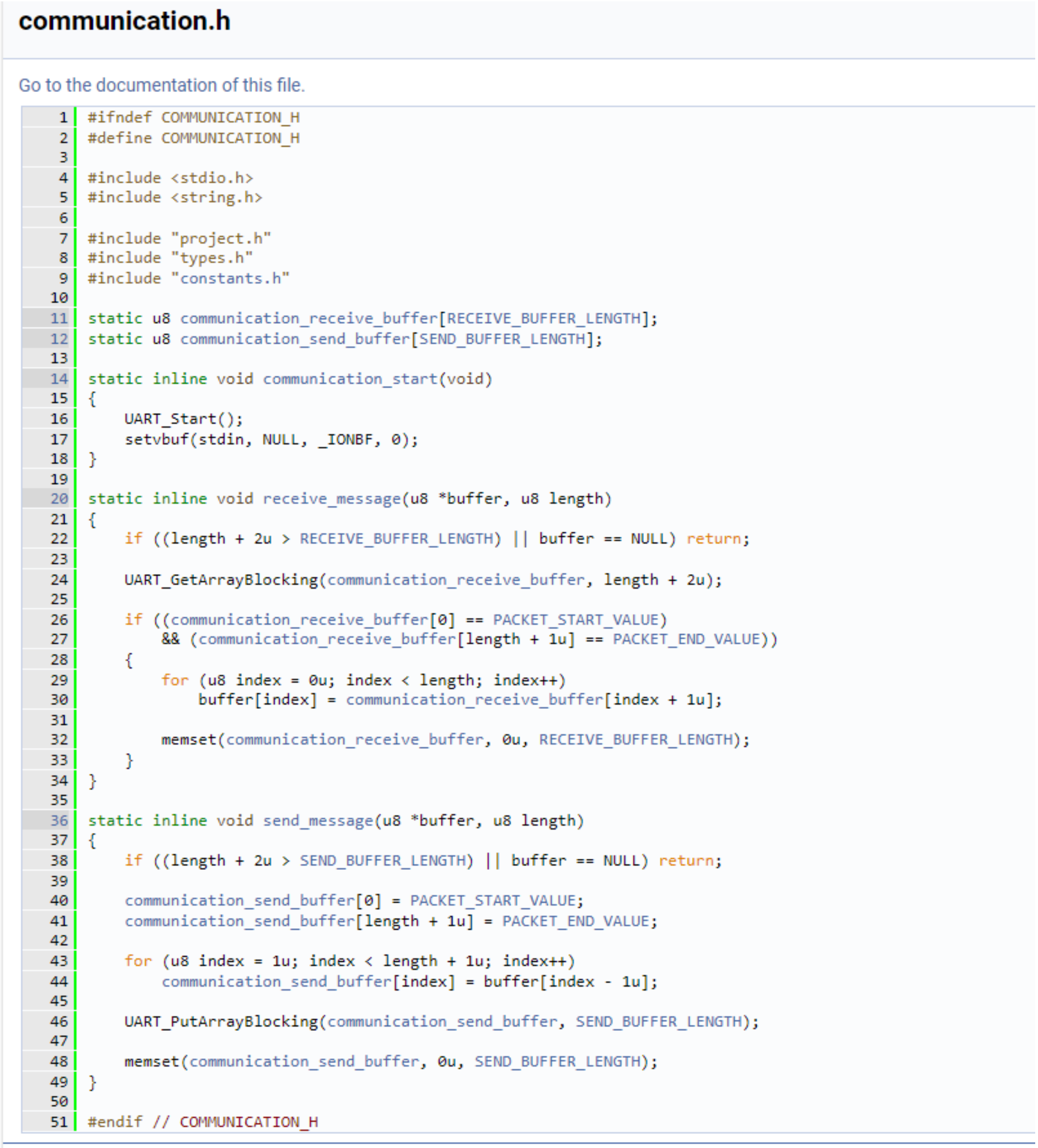
Go to the source code of this file.

Macros

#define	TRUE	1u
#define	FALSE	0u
#define	RECEIVE_BUFFER_LENGTH	4u
#define	SEND_BUFFER_LENGTH	11u
#define	GAME_TYPE_PACKET_LENGTH	2u
#define	CELLS_NUMBER	3u
#define	UNKNOWN_PACKET_VALUE	0xAA
#define	MAN_VS_MAN_PACKET_VALUE	0xBB
#define	MAN_VS_AI_PACKET_VALUE	0xCC
#define	AI_VS_AI_PACKET_VALUE	0xDD
#define	ACK_PACKET_VALUE	0xEE
#define	RESET_PACKET_VALUE	0x99
#define	WIN_CROSS_PACKET_VALUE	0x88
#define	WIN_NOD_PACKET_VALUE	0x77
#define	NO_WINNER_PACKET_VALUE	0x22
#define	PACKET_START_VALUE	'<'
#define	PACKET_END_VALUE	'>'

Variables

const u8	win_masks [8u][CELLS_NUMBER *CELLS_NUMBER]



communication.h

Go to the documentation of this file.

```c
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <stdio.h>
#include <string.h>

#include "project.h"
#include "types.h"
#include "constants.h"

static u8 communication_receive_buffer[RECEIVE_BUFFER_LENGTH];
static u8 communication_send_buffer[SEND_BUFFER_LENGTH];

static inline void communication_start(void)
{
    UART_Start();
    setvbuf(stdin, NULL, _IONBF, 0);
}

static inline void receive_message(u8 *buffer, u8 length)
{
    if ((length + 2u > RECEIVE_BUFFER_LENGTH) || buffer == NULL) return;

    UART_GetArrayBlocking(communication_receive_buffer, length + 2u);

    if ((communication_receive_buffer[0] == PACKET_START_VALUE)
        && (communication_receive_buffer[length + 1u] == PACKET_END_VALUE))
    {
        for (u8 index = 0u; index < length; index++)
            buffer[index] = communication_receive_buffer[index + 1u];

        memset(communication_receive_buffer, 0u, RECEIVE_BUFFER_LENGTH);
    }
}

static inline void send_message(u8 *buffer, u8 length)
{
    if ((length + 2u > SEND_BUFFER_LENGTH) || buffer == NULL) return;

    communication_send_buffer[0] = PACKET_START_VALUE;
    communication_send_buffer[length + 1u] = PACKET_END_VALUE;

    for (u8 index = 1u; index < length + 1u; index++)
        communication_send_buffer[index] = buffer[index - 1u];

    UART_PutArrayBlocking(communication_send_buffer, SEND_BUFFER_LENGTH);

    memset(communication_send_buffer, 0u, SEND_BUFFER_LENGTH);
}

#endif // COMMUNICATION_H
```

ВИСНОВОК:

В ході роботи No4 створено doxygen коментарі для кожного файла в проекті та згенеровано HTML doxygen документацію.