坦克射擊遊戲

組員:許育瑋、謝瑞峰、曾喜煌

大綱

- 專題構想與簡介
- 網路程式設計課堂內容相關性
- 核心程式碼與程式流程說明
- 遭遇之問題與解決方案
- 心得與結論
- 影片展示

專題構想與簡介

.io 遊戲

- 靈感來自於 io 遊戲，如 Agar.io 與 slither.io
- 特點：
  - 操作簡單
  - 可隨時進入或離開遊戲
  - 成長系統

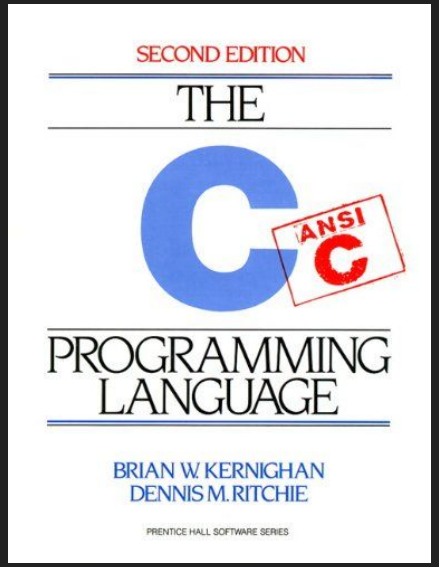
tank-game

- 一個多人的線上射擊遊戲
- 在終端機內就可以執行
- 操作簡單:
  - 上下左右:方向鍵
  - 射擊:x
  - 填充子彈:z
- 可隨時加入或離開遊戲
- 沒有成長系統

https://github.com/9501sam/tank-game

使用的語言及環境

- C 語言
- 使用者介面：ncurses（不必按 enter 就可以讀入鍵盤輸入、方便在終端機內用文字畫圖）



+

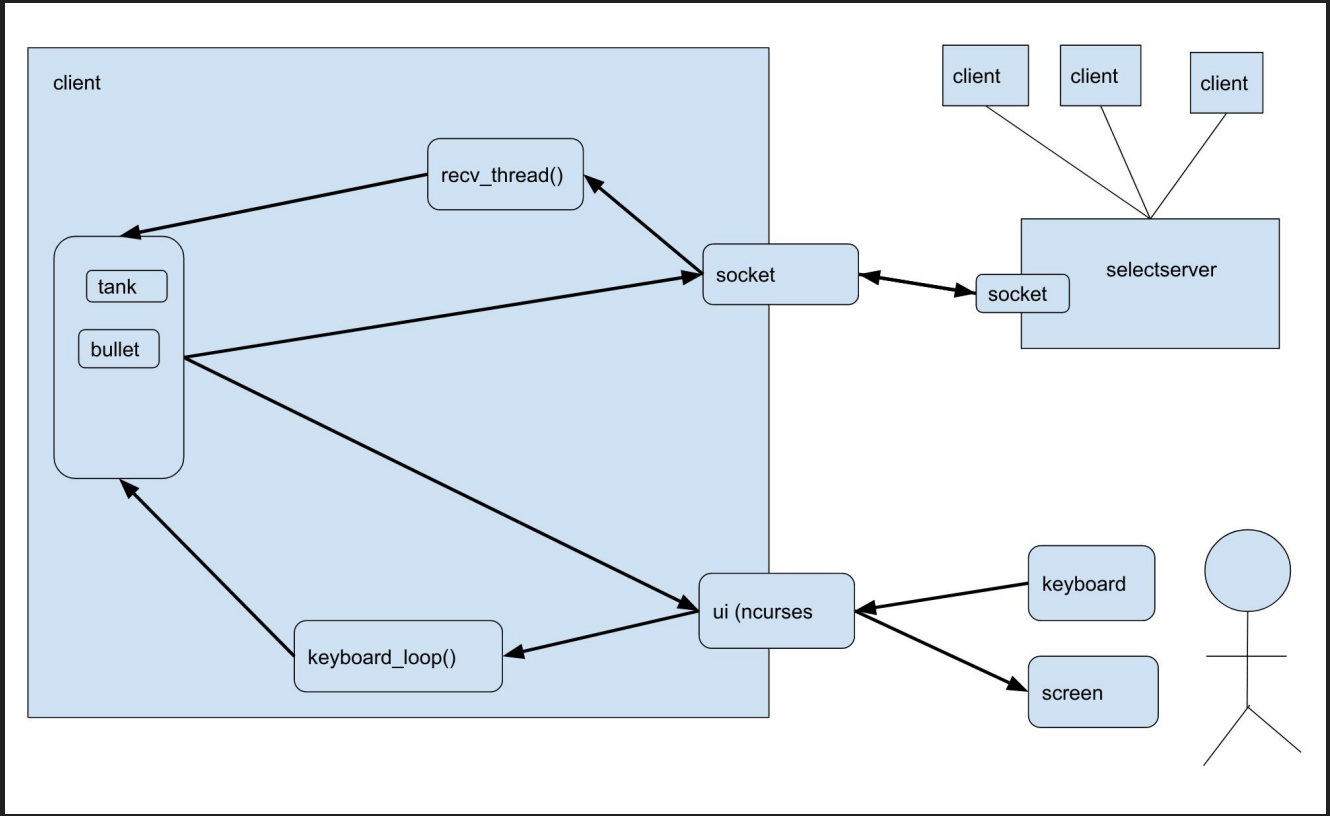
網路程式設計課堂內容相關性

網路程式設計課堂內容相關性

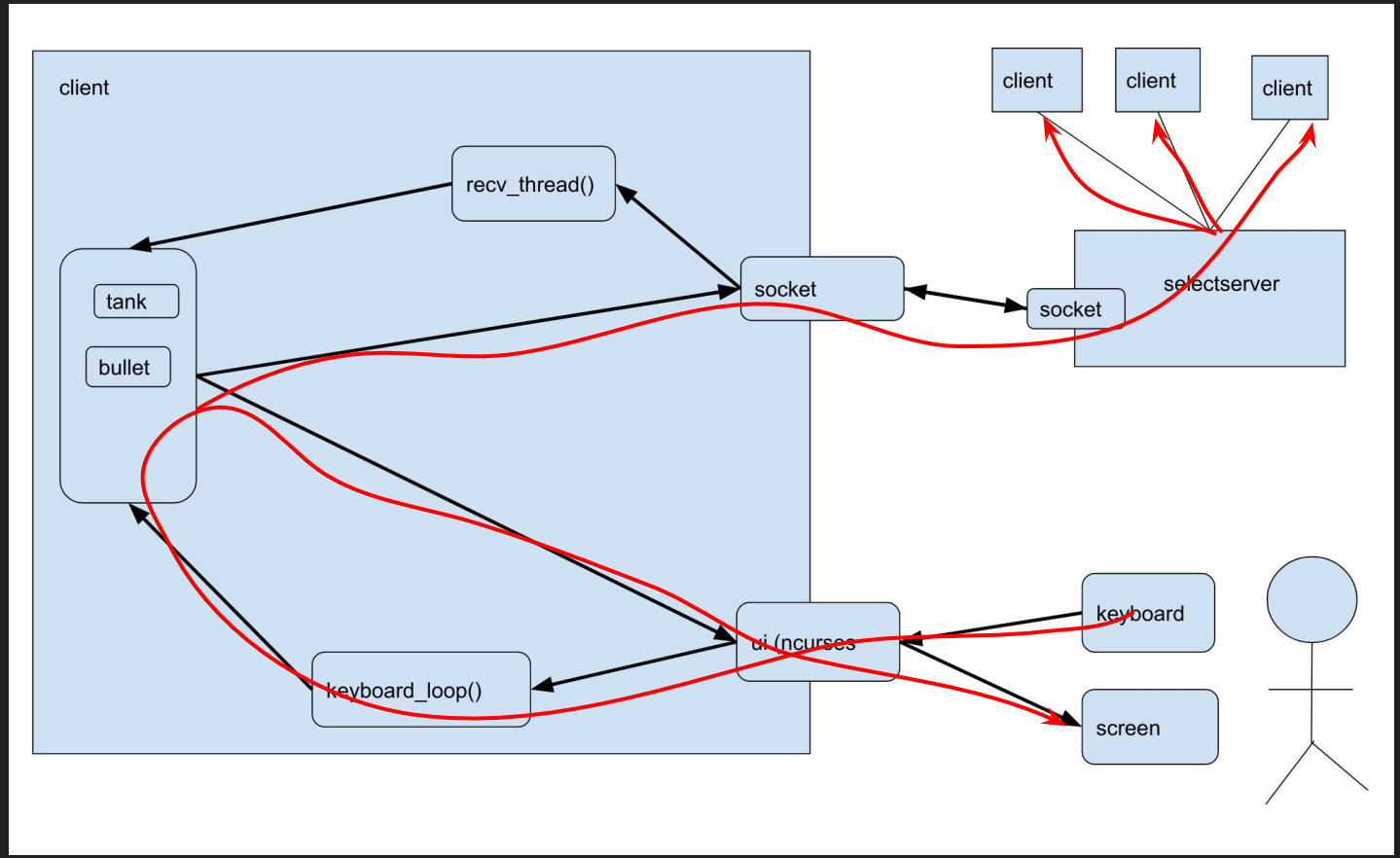
- server 端是由 lab06 的 selectserver.c 修改而來
- 本質上是一個多人網路聊天室

核心程式碼與程式流程說明

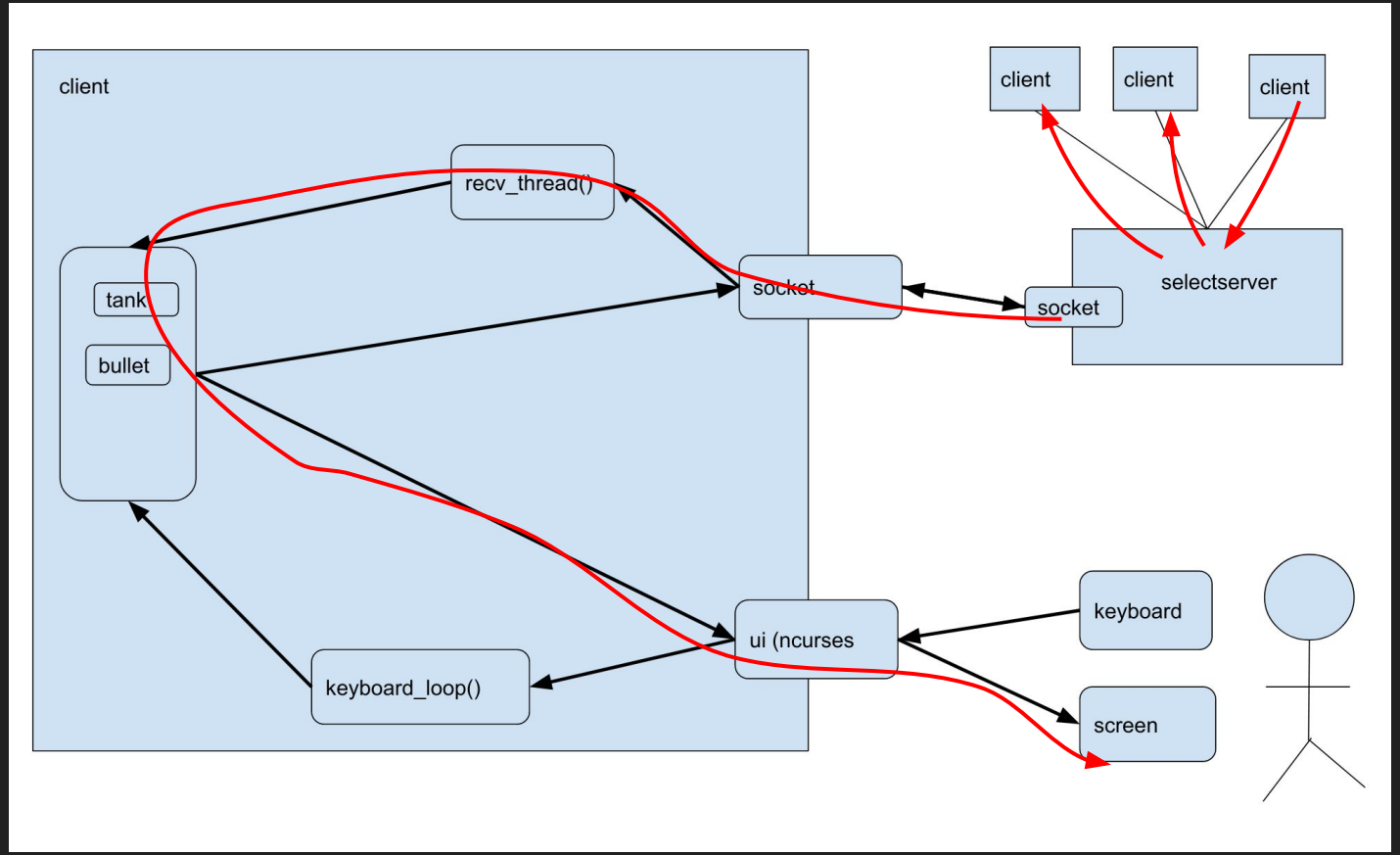
程式架構圖


client
recv_thread()
tank
bullet
socket
client
client
client
selectserver
socket
keyboard_loop()
ui (ncurses
keyboard
screen

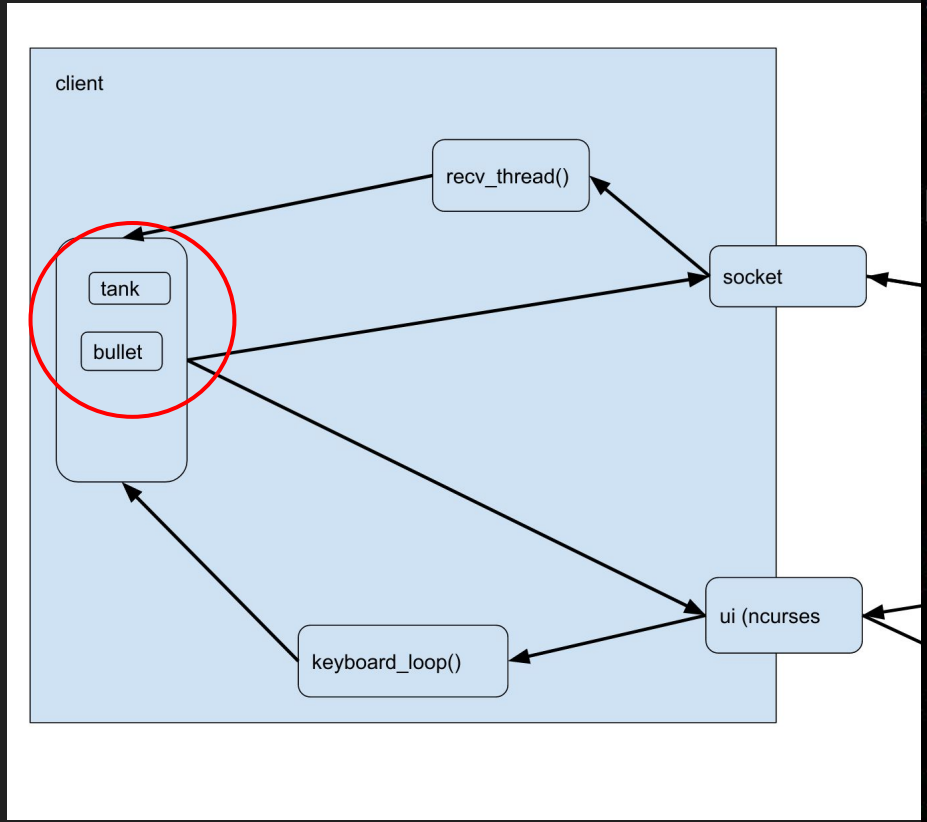
當使用者輸入時


client
recv_thread()
tank
bullet
socket
keyboard_loop()
ui (ncurses
client
client
client
socket
selectserver
keyboard
screen

當接收到來自 server 的資料時

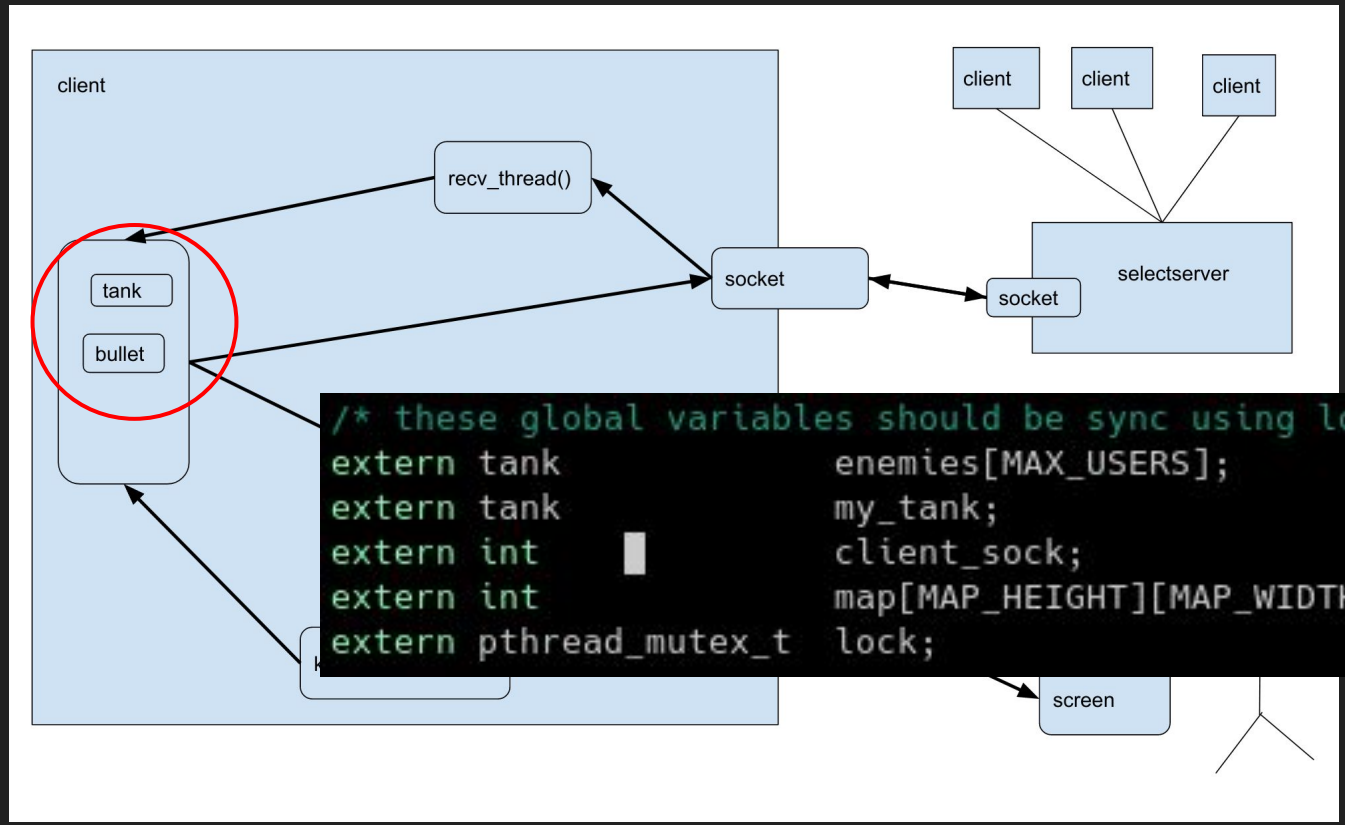


程式碼 : 資料結構

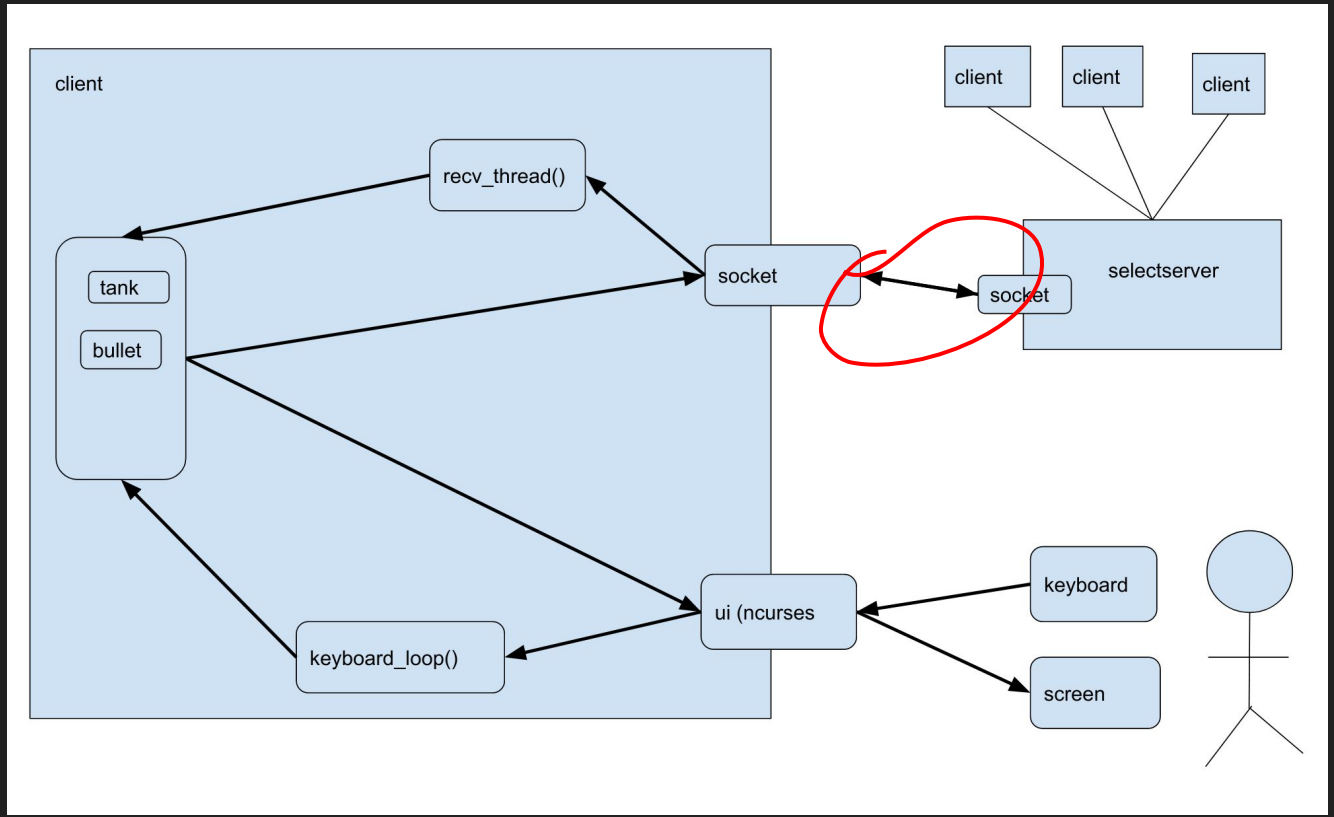



```
// direction
typedef uint8_t direction_t;
#define LEFT    (direction_t) 0
#define RIGHT   (direction_t) 1
#define UP      (direction_t) 2
#define DOWN    (direction_t) 3
#define NUM_DIR (direction_t) 4

///*** tank ***///
typedef struct {
    uint16_t x;
    uint16_t y;
    direction_t dir;
    int8_t hp;
    int8_t nblts;
    int8_t id;
} tank;

bool my_tank_move(direction_t);
void my_tank_attacked(void);
void my_tank_refill(void);

///*** bullet ***///
typedef struct {
    int x;
    int y;
    direction_t dir;
} bullet;

void *shoot(void *);
void shoot_thread_create(int id);
void my_tank_shoot(void);
```

程式碼 : 資料結構


client
recv_thread()
tank
bullet
socket
client
client
client
selectserver
socket
screen
/* these global variables should be sync using lock */
extern tank enemies[MAX_USERS];
extern tank my_tank;
extern int client_sock;
extern int map[MAP_HEIGHT][MAP_WIDTH];
extern pthread_mutex_t lock;

程式碼:傳遞資料


client
recv_thread()
tank
bullet
socket
keyboard_loop()
ui (ncurses
client
client
client
selectserver
socket
keyboard
screen

程式碼：傳遞資料 (struct packet)

- NEW_TANK (data 是 tank)
  - 當新連線建立時，server 送給所有的 client
- DIE (data 是 id)
  - 當有 client 斷線時，server 送給所有剩下的 client
- TANK (data 是 tank)
  - client 移動時送給 server，server 再送給其他 client
- SHOOT (data 是 id)
  - client 射擊時送給 server，server 再送給其他 client
- ATTACKED (data 是 id)
  - client 被攻擊時送給 server，server 再送給其他 client

```
///*** network ***///
typedef uint8_t packet_kind;
#define NEW_TANK (packet_kind) 0
#define TANK     (packet_kind) 1
#define SHOOT    (packet_kind) 2
#define REFILL   (packet_kind) 3
#define ATTACKED (packet_kind) 4
#define DIE      (packet_kind) 5

struct packet {
    packet_kind kind;
    union {
        tank   tk;
        int8_t id;
    } data;
};

int recv_packet(int fd, struct packet *pkt);
int send_packet(int fd, struct packet *pkt);
```

程式碼:傳遞資料(錯誤寫法)

- 原本的寫法
- 沒有考慮到 endianness

```
struct packet pkt;
pkt.kind = REFILL;
pkt.data.id = 3;
send(fd, pkt, sizeof(pkt), 0)
```

```
struct packet pkt;
recv(fd, pkt, sizeof(pkt), 0)
```

```
///*** network ***///
typedef uint8_t packet_kind;
#define NEW_TANK (packet_kind) 0
#define TANK     (packet_kind) 1
#define SHOOT    (packet_kind) 2
#define REFILL   (packet_kind) 3
#define ATTACKED (packet_kind) 4
#define DIE      (packet_kind) 5

struct packet {
    packet_kind kind;
    union {
        tank   tk;
        int8_t id;
    } data;
};

int recv_packet(int fd, struct packet *pkt);
int send_packet(int fd, struct packet *pkt);
```

程式碼:傳遞資料

```c
///*** network ***///
typedef uint8_t packet_kind;
#define NEW_TANK (packet_kind) 0
#define TANK     (packet_kind) 1
#define SHOOT    (packet_kind) 2
#define REFILL   (packet_kind) 3
#define ATTACKED (packet_kind) 4
#define DIE      (packet_kind) 5

struct packet {
    packet_kind kind;
    union {
        tank   tk;
        int8_t id;
    } data;
};

int recv_packet(int fd, struct packet *pkt);
int send_packet(int fd, struct packet *pkt);
```

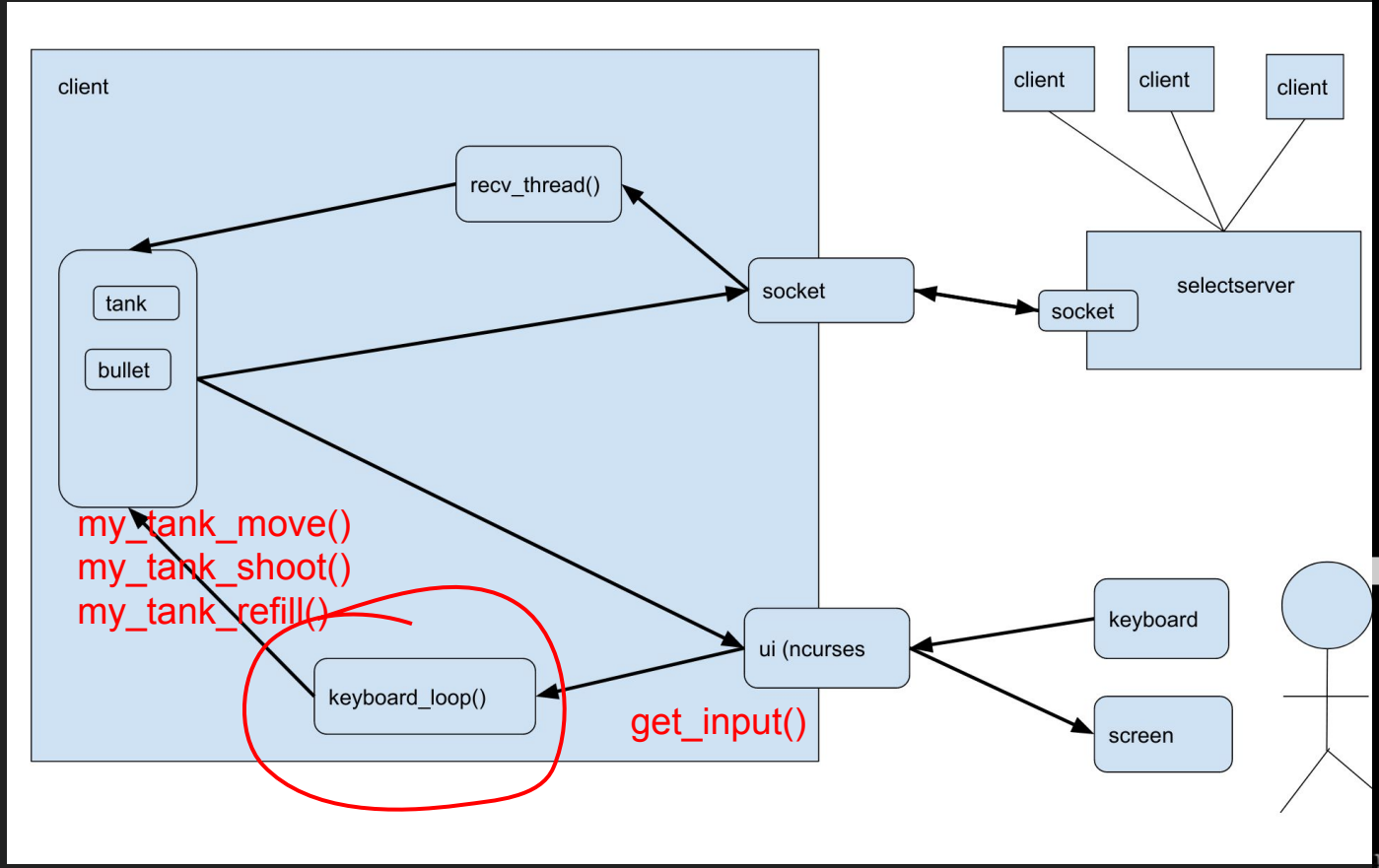
```c
int send_packet(int fd, struct packet *pkt)
{
    char buffer[BUF_SIZE];
    packet_kind k = pkt->kind;
    // uint8_t kind
    buffer[0] = k;
    switch (k) {
    case NEW_TANK:
    case TANK:
        // uint16_t x;
        buffer[1] = pkt->data.tk.x >> 8;
        buffer[2] = pkt->data.tk.x;
        // uint16_t y;
        buffer[3] = pkt->data.tk.y >> 8;
        buffer[4] = pkt->data.tk.y;
        // uint8_t dir;
        buffer[5] = pkt->data.tk.dir;
        // int8_t hp;
        buffer[6] = pkt->data.tk.hp;
        // int8_t nblts;
        buffer[7] = pkt->data.tk.nblts;
        // int8_t id;
        buffer[8] = pkt->data.tk.id;
        return send(fd, buffer, sizeof(buffer), 0);
        break;
    case SHOOT:
    case REFILL:
    case ATTACKED:
    case DIE:
        buffer[1] = pkt->data.id;
        return send(fd, buffer, 2 * sizeof(char), 0);
        break;
    default:
        return -1;
    }
    return -1;
}
```

程式碼:傳遞資料

```c
///*** network ***///
typedef uint8_t packet_kind;
#define NEW_TANK (packet_kind) 0
#define TANK     (packet_kind) 1
#define SHOOT    (packet_kind) 2
#define REFILL   (packet_kind) 3
#define ATTACKED (packet_kind) 4
#define DIE      (packet_kind) 5

struct packet {
    packet_kind kind;
    union {
        tank   tk;
        int8_t id;
    } data;
};

int recv_packet(int fd, struct packet *pkt);
int send_packet(int fd, struct packet *pkt);
```

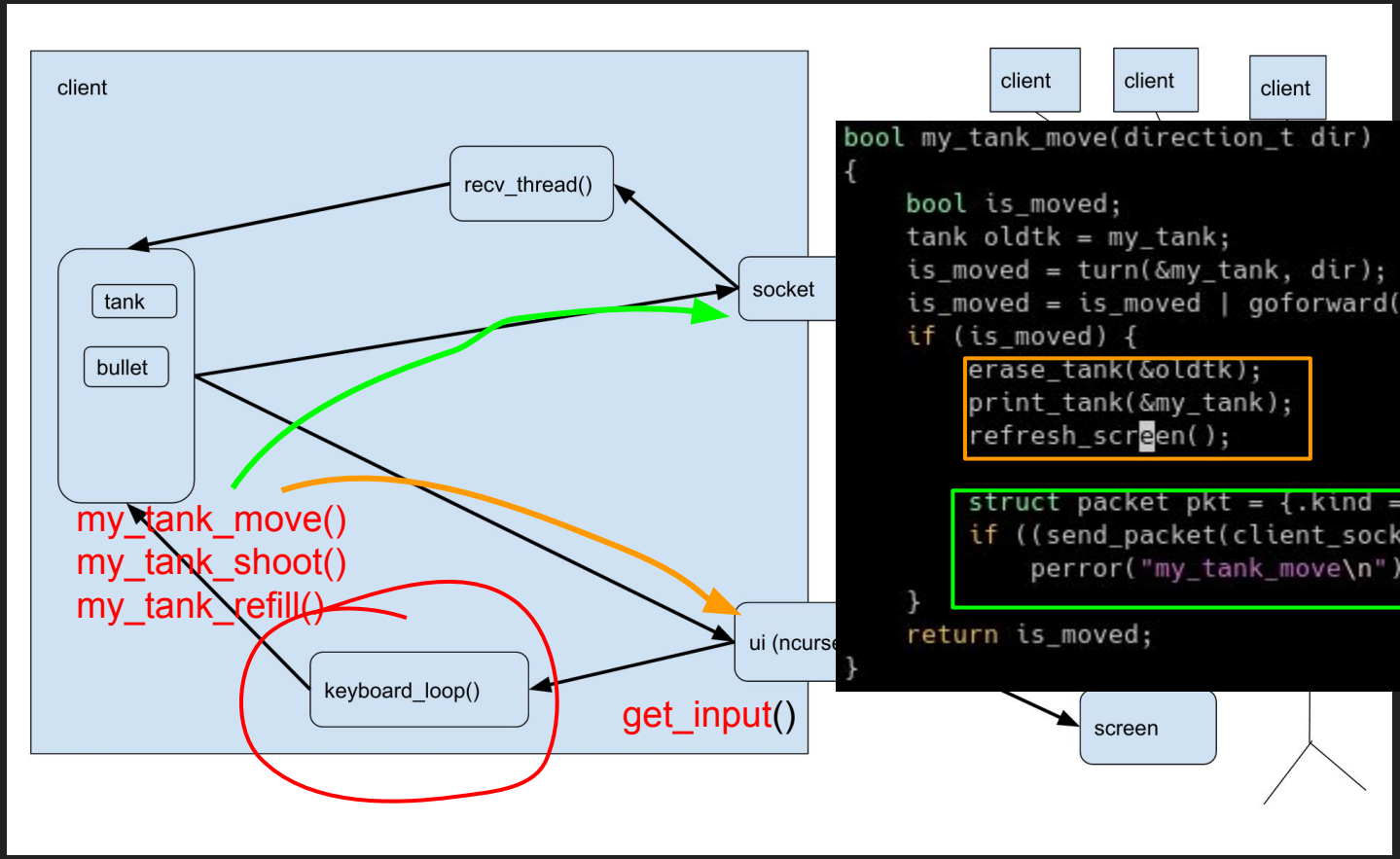
```c
int recv_packet(int fd, struct packet *pkt)
{
    char buffer[BUF_SIZE];
    int ret;
    packet_kind k;
    if ((ret = recv(fd, buffer, sizeof(buffer), 0)) <= 0)
        return ret;
    k = buffer[0];
    pkt->kind = k;
    switch (k) {
    case NEW_TANK:
    case TANK:
        // uint16_t x;
        pkt->data.tk.x = (((uint16_t) buffer[1]) << 8) | (uint16_t) buffer[2];
        // uint16_t y;
        pkt->data.tk.y = (((uint16_t) buffer[3]) << 8) | (uint16_t) buffer[4];
        // uint8_t dir;
        pkt->data.tk.dir = buffer[5];
        // int8_t hp;
        pkt->data.tk.hp = buffer[6];
        // int8_t nblts;
        pkt->data.tk.nblts = buffer[7];
        // int8_t id;
        pkt->data.tk.id = buffer[8];
        break;
    case SHOOT:
    case REFILL:
    case ATTACKED:
    case DIE:
        pkt->data.id = buffer[1];
        break;
    default:
        return -1;
    }
    return ret;
}
```

程式碼:當使用者輸入時

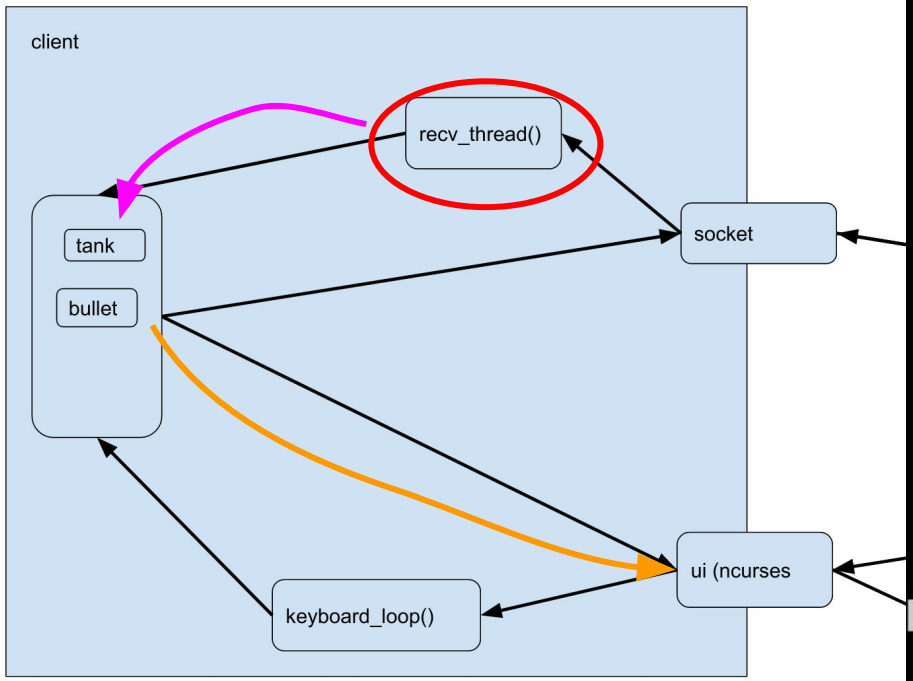



```c
static void keyboard_loop(void)
{
    input_t in;
    while (1) {
        in = get_input();
        if (in == INPUT_INVALID)
            continue;
        pthread_mutex_lock(&lock);
        switch (in) {
        case INPUT_LEFT:
            my_tank_move(LEFT);
            break;
        case INPUT_RIGHT:
            my_tank_move(RIGHT);
            break;
        case INPUT_UP:
            my_tank_move(UP);
            break;
        case INPUT_DOWN:
            my_tank_move(DOWN);
            break;
        case INPUT_SHOOT:
            my_tank_shoot();
            break;
        case INPUT_REFILL:
            my_tank_refill();
            break;
        case INPUT_QUIT:
            exit(EXIT_SUCCESS);
            break;
        case INPUT_INVALID:
            break;
        }
        pthread_mutex_unlock(&lock);
    }
}
```

程式碼:當使用者輸入時(以按下方向鍵為例)

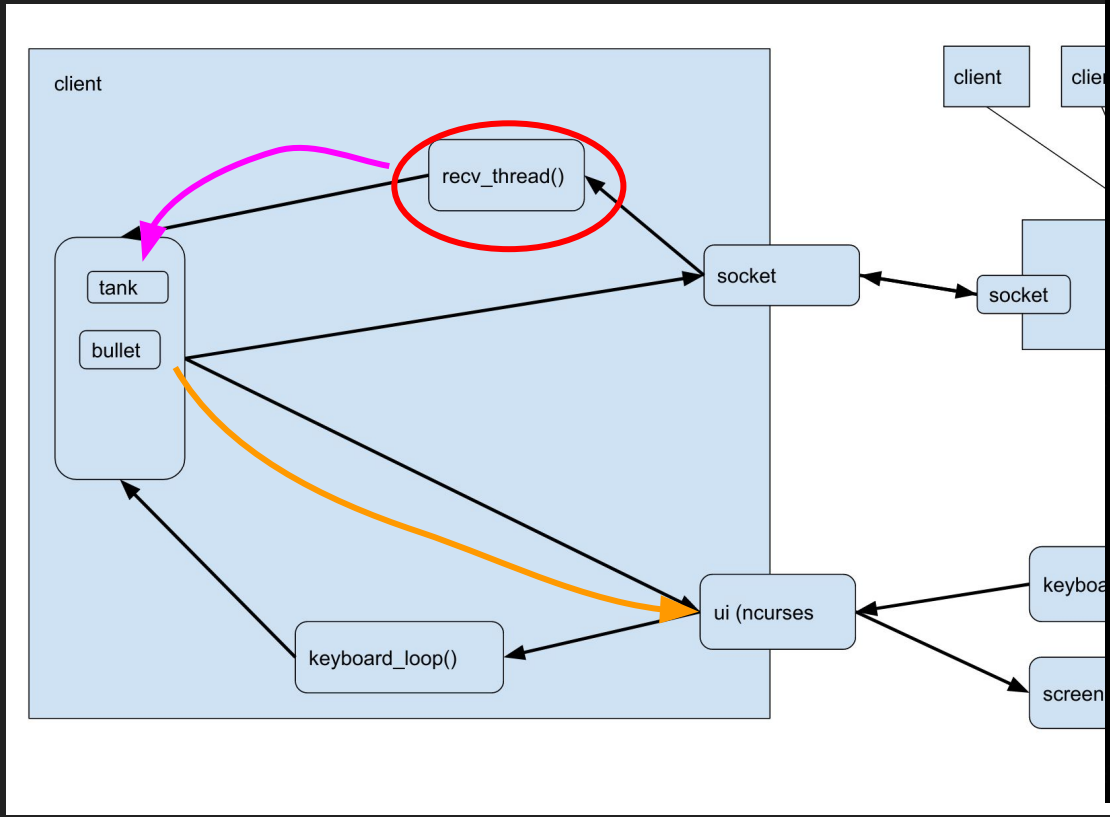


程式碼:接收到資料時




```c
static void *recv_thread(void *arg)
{
    struct packet pkt;
    int id, nbytes;

    while (1) {
        if ((nbytes = recv_packet(client_sock, &pkt)) <= 0)
            exit(EXIT_SUCCESS);
        pthread_mutex_lock(&lock);
        switch (pkt.kind) {
        case NEW_TANK:
            id = pkt.data.tk.id;
            add_enemy(&pkt.data.tk);
            print_tank(&pkt.data.tk);
            refresh_screen();
            break;
        case TANK:
            id = pkt.data.tk.id;
            tank oldtk = enemies[id];
            enemies[id] = pkt.data.tk;
            erase_tank(&oldtk);
            print_tank(&enemies[id]);
            refresh_screen();
            break;
        case SHOOT:
            id = pkt.data.id;
            enemies[id].nblts--;
            erase_tank_info(&enemies[id]);
            print_tank_info(&enemies[id]);
            refresh_screen();
            shoot_thread_create(id);
            break;
```

程式碼：接收到資料時



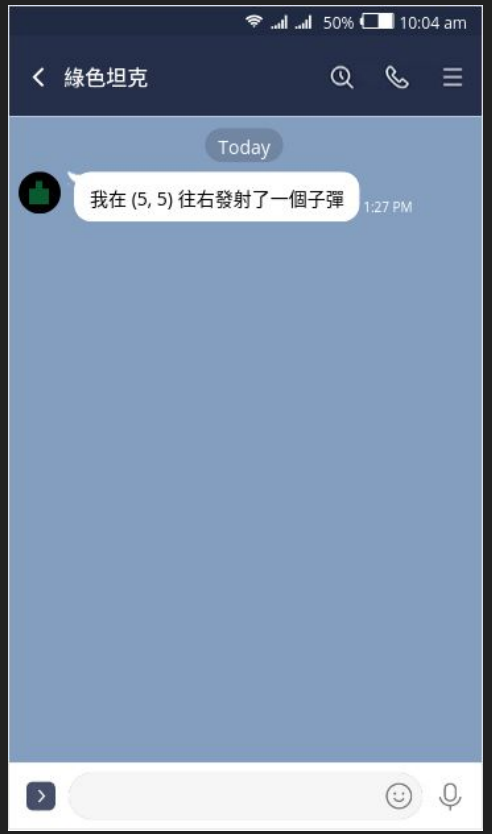

```
        case REFILL:
            id = pkt.data.id;
            enemies[id].nblts = NUM_BULLETS;
            erase_tank_info(&enemies[id]);
            print_tank_info(&enemies[id]);
            refresh_screen();
            break;
        case ATTACKED:
            id = pkt.data.id;
            enemies[id].hp--;
            print_tank_info(&enemies[id]);
            refresh_screen();
            break;
        case DIE:
            id = pkt.data.id;
            tank dietk = enemies[id];
            if (dietk.id != NOT_USED) {
                erase_tank(&dietk);
                refresh_screen();
            }
            del_enemy(id);
            break;
        }
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}
```

遭遇之問題與解決方案

- 讓子彈飛
- 發射子彈的紅色坦克與躲避子彈的綠色坦克

讓子彈飛:傳遞射擊事件

- 第一種方式太煩了
- 第二種方式比較不煩
  - 如何處理這個訊息成為下一個問題



讓子彈飛:處理射擊事件

1. 開一個執行緒製造 clock，搭配一個子彈的集合，每一個 clock 就把集合內的子彈往前推進一個單位
2. 一個子彈就個別用一個執行緒處理
    - 採用原因:程式碼比較方便寫

讓子彈飛:程式碼

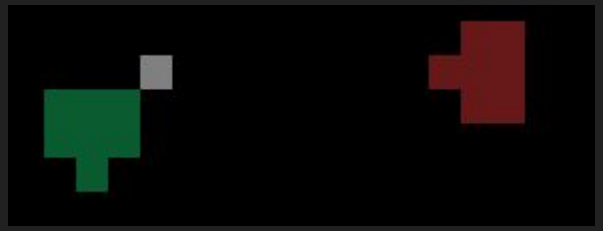
```c
void my_tank_shoot(void)
{
    if (my_tank.nblts <= 0)
        return;
    my_tank.nblts--;

    struct packet pkt = {
        .kind = SHOOT,
        .data.id = my_tank.id,
    };
    if (send_packet(client_sock, &pkt) == -1)
        perror("send");
    shoot_thread_create(my_tank.id);
}
```

```c
void shoot_thread_create(int id)
{
    pthread_t tid;
    tank *tk = NULL;
    if (id == my_tank.id)
        tk = &my_tank;
    else
        tk = &enemies[id];
    erase_tank_info(tk);
    print_tank_info(tk);
    refresh_screen();

    int *arg = malloc(sizeof(int));
    *arg = id;
    pthread_create(&tid, NULL, shoot, (void *) arg);
}
```

發射子彈的紅色坦克與躲避子彈的綠色坦克 : 問題

- 紅色坦克的視角 :
- 由於網路延遲的關係, 紅色坦克認為綠色坦克還在原地
- 紅色坦克認為子彈已經打到了綠色坦克

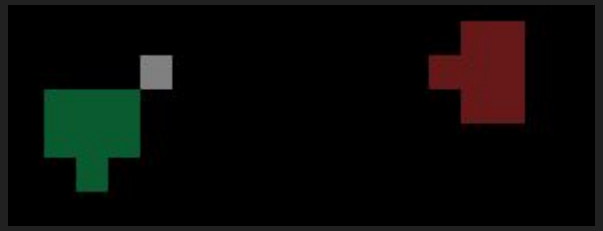
- 綠色坦克的視角 :
- 綠色坦克在快要被打到的瞬間往下閃 躲
- 綠色坦克認為自己閃過了子彈



發射子彈的紅色坦克與躲避子彈的綠色坦克:解決方案

- 紅色坦克的視角:
- 子彈會因為擊中綠色坦克而消失
- 但因為綠色坦克沒有傳遞他被攻擊的訊息，所以綠色坦克的生命 值不會減少

- 綠色坦克的視角:
- 子彈會繼續往左飛行



解決方案:定義有沒有攻擊到是受攻擊的坦克 說了算

心得與結論

- 好很玩
- 透過這次專題了解如何寫連線的應用程式

參考資料

- Beej's Guide to Network Programming: 7.3 select()—Synchronous I/O Multiplexing, Old School
- NCURSES Programming HOWTO
- Passing a structure through Sockets in C
- The Linux Programming Interface 第 29 章 Threads: Introduction 以及第 30 章 Threads: Thread Synchronization
- C Programming: A Modern Approach, 2/e, p355: Sharing Variable Declarations

影片展示