Algorithms and Data Structure
SET08122

Yifu Lai
40413832@live.napier.ac.uk

1.Introduction

This report will show how to complete a c language game called tic tac toe. This game contains four basic functions, first, the game has the game board (1-1). It is stored in a two-dimensional array. Second, the game can be played by two players at the same time. In this section, a loop is used that allows the player to take turns playing chess based on the odds of the turn. Third, this game uses ‘noughts’ and ‘crosses’ to represent two players (1-2). Fourth, the game board can record which pieces have been placed in which position by which player. In addition, the program also can recode the history of the each step that is played by players and automatically replayed them from this record (1-3). On the other hand, the game still has some problems, such as, it cannot support the undo function, which players can un-apply their actions, returning the game state to the immediate previous state.

2. Design

2.1 Representation of game state

In order to implement this game, I think the representation method should be the first to consider. Because this is a turn-based game, I need to store the game state for each round. I used a structure named ‘state’ to represent the state of a round of tic-tac-toe (2-1-1). It includes a game board that is stored in a ‘3x3’ two-dimensional array. The reason for using a two-dimensional array is because it is easier to achieve a win-loss decision. In addition, the number of the turn is also included in this structure. Because, in this game, I used an algorithm to determine whether the round is odd or even, to choose which player to play chess . Then add function to initialize the board (2-1-2). In the function, I use the nested loop to enter the number ‘-1’ into each position in the two-dimensional array, ‘-1’ means that the position is empty, player can place the piece. The number of rounds is also assigned a value of 0.

2.2 Show game status

When the board is set, I think the game state should be displayed first, and then add other rules. Because this can be easily tested. I filled the nine slots on the board with numbers so that the player can enter the chess position. Because the game state is displayed read-only without changing the game state, the parameter type is const. In the ‘display’ function, I use the switch statement to detect the number stored in each array element (2-2-1). If it is ‘-1’, it means that the position has not been placed yet, so break the loop and the number corresponding to it is displayed. When it detects that ‘0’ or ‘1’ is stored in the array, it outputs ‘O’ or ‘X’, indicating the chess pieces of each player. Finally, use two judgment sentences to print the checkerboard of Tic Tac Toe (2-2-2).

2.3 Chess function

Next I realized the function of playing chess. In the ‘Move’ function, there is a legality judgment statement. If the position has been played a chess, the function will return 0 means input failed. On the other hand, if there can be played the chess, according to the number of rounds, the parity is judged. If it is an odd round, assign the value of the array element to 0, otherwise the value is 1 (2-3-1). Because in display function that has already specified that printf ‘O’ when the array element is ‘0’, and printf ‘X’ when it is ‘1’. In addition, the number of rounds starts from 0. At the end, the number of turn plus ‘1’ at each end of round.

2.4 Processing input

At this part, allow players to take turns playing chess and display the player's pieces on the board. According to the parity of the round, the player turns chess in the order of ‘O’‘X’. In order to prevent the player from entering multiple numbers incorrectly, the program ignores other characters until the newline is read after reading the first input character (2-4-1). Then convert the input number into an index into a two-dimensional array (2-4-2).

2.5 Winning and losing judgment

In order to determine the outcome, I implemented an evaluate() function to evaluate the state of the game. If O wins, it returns 1. If X wins, it returns -1. If no win or loss, it returns 0. The function ‘evaluate’ uses a check() macro to detect if the three connected pieces are the same (2-5-1).

In the end, in the main() , using a switch statement to judge the value returned by evaluate() function. In addition, after each time you play chess and display the status, it is determined whether or not the winner is present. If the ninth round is reached (starting at 0), the result of the tie is output (2-5-2).

3. Enhancements

In the general chess game, there are undo functions. When I started writing program, I planed to add the undo function. However, I have no idea that how to break the loop of program and make the undo operation. In addition, I would like to add a function that players can chose the two kinds of pieces what they want by themselves at the begin of game.

4. Critical evaluation

4.1 Round and player relationship

Considering that this is a turn-based game, I choose the odd number of the round to determine which player to play chess. Its advantages are simple and clear, no redundant features. On the other hand, This also makes it impossible for players to choose the pieces they want. I think this is a disadvantage of this program.

4.2 Playing game

While play the game, player may input the wrong number of the position. In order to solve the trouble, I use the getchar (). The program ignores other characters until the newline is read after reading the first input character. I think this is a positive feature to prevent game errors.

4.3 Winning and losing judgment

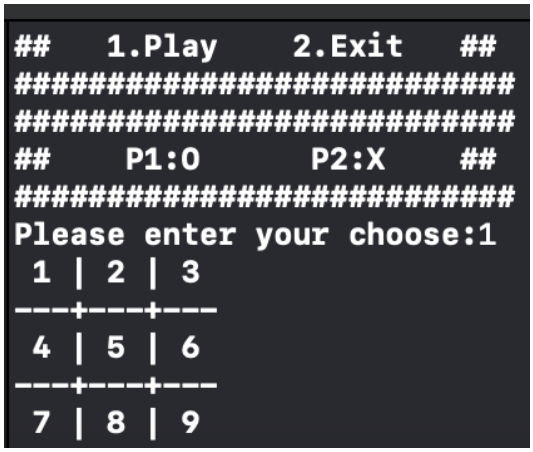
In this function, I think I did well. At the beginning, I am going to write a loop structure that will write the winnings for each row and column. Then I found that it is too redundant. I found some tutorial on the internet, learn a function that can greatly shorten the code.

5. Personal evaluation

Through this coursework, I was more impressed with the c language. When I started writing programs, I was confused. Because the hardest thing is from 0 to 1, not from 1 to 100. During the coursework, I have encountered many problems. For example, I have no idea that how to store information into a two-dimensional array and return different values. However, I got a lot of help by referring to the lab of this semester. In addition, I also found a lot of tutorials through the network to help me build this program. Now, I fully understand what they mean and how to use them. I think my program is not bad, although it looks a bit rudimentary, but the function is complete. But I still have a lot of room for improvement. In the rest of the semester, I will continue to learn about data structures.

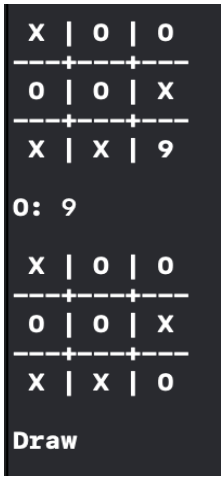
appendices

1.Introduction



game board(1-1)

'noughts' and 'crosses'(1-2)



record history(1-3)

2. Design

```c
typedef struct {
    int board[3][3];     // -1 = empty, 0 = O, 1 = X
    int turn;            // O first
} state;
```

state structure(2-1-1)

```c
void init(state* s) {
    int i, j;
    for (j = 0; j < 3; j++)
        for (i = 0; i < 3; i++)
            s->board[j][i] = -1;
    s->turn = 0;
}
```

void init(2-1-2)

```c
void display(const state* s) {
    int i, j;
    for (j = 0; j < 3; j++) {
        for (i = 0; i < 3; i++) {
            switch (s->board[j][i]) {
                case -1: printf(" %d ", j * 3 + i + 1); break;
                case  0: printf(" O "); break;
                case  1: printf(" X "); break;
            }
            if (i < 2)
                printf("|");
            else
                printf("\n");
        }
        if (j < 2)
            printf("---+---+---\n");
        else
            printf("\n");
    }
}
```

void display(2-2-1)

```c
        if (i < 2)
            printf("|");
        else
            printf("\n");
    }
    if (j < 2)
        printf("---+---+---\n");
    else
        printf("\n");
```

printf game board(2-2-2)

```
int Move(state* s, int i, int j) {
    if (s->board[j][i] != -1){
        printf("this place has been played\n");
        return 0;}
    else s->board[j][i] = s->turn++ % 2;
    return 1;
}
```

chess function(2-3-1)

```
void human(state* s) {
    char c;
    do {
        printf("%c: ", "OX"[s->turn % 2]);
        c = getchar();
        while (getchar() != '\n');
        printf("\n");
    } while (c < '1' || c > '9' || !Move(s, (c - '1') % 3, (c - '1') / 3));
}
```

processing input(2-4-1)

```
    } while (c < '1' || c > '9' || !Move(s, (c - '1') % 3, (c - '1') / 3));
```

conver number into index(2-4-2)

```
#define CHECK(j1, i1, j2, i2, j3, i3) \
if (s->board[j1][i1] != -1 && s->board[j1][i1] == s->board[j2][i2] && s->board[j1][i1] == s->board[j3][i3]) \
return s->board[j1][i1] == 0 ? 1 : -1;

int evaluate(const state* s) {
    int i;
    for (i = 0; i < 3; i++) {
        CHECK(i, 0, i, 1, i, 2);    // horizontal
        CHECK(0, i, 1, i, 2, i);    // vertical
    }
    CHECK(0, 0, 1, 1, 2, 2);        // diagonal
    CHECK(0, 2, 1, 1, 2, 0);        // diagonal
    return 0;
}
```

Winning and losing judgment(2-5-1)

```c
int main() {
    int a;
    menu();
    printf("Please enter your choose:");
    scanf("%d",&a);
    if(a!= 1){ return 0; }
    while(1)
    {
        state s;
        init(&s);
        display(&s);
        while (s.turn < 9) {
            human(&s);
            display(&s);
            switch (evaluate(&s)) {
                case  1: printf("O is winner\n\n"); return 0;
                case -1: printf("X is winner\n\n"); return 0;
            }
        }
        printf("Draw\n\n"); break;}

}
```

main() (2-5-2)