LAB

Place tokens & Play game

Pre-Requisites

In this lab we assume that the tasks requested in the previous lab are completed.

- The data structure representing the players of the game has been created.
- The function to initalize players (`initialize_players`) has been implemented and players can now be created from the command line.

Objectives

Implement the following functions inside file *game.c*

- *place_tokens*
- *play_game*

Assumption (Only for This Week)

- You can start implement those functions just assuming that only 1 token at a time can be placed on each square of the board.

- If a token A is moved to a square that already contains a token B, Token A “overwrites” Token B.

Function *place_tokens* (1/3)

```
/*
 * Place tokens in the first column of the board
 *
 * Input: board - a 6x9 array of squares that represents the board
 *        players - the array of the players
 *        numPlayers - the number of players
 */
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){

    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;

    // TO BE IMPLEMENTED

}
```

- Should allow players to put all their tokens only on the first column of the board.
- Should ensure that players do not put their token on top of a token of the same colour.

Function *place_tokens* (2/3)

```
/*
 * Place tokens in the first column of the board
 *
 * Input: board - a 6x9 array of squares that represents the board
 *        players - the array of the players
 *        numPlayers - the number of players
 */
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){

    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;

    // TO BE IMPLEMENTED

}
```

- This function should ensure that players put their token in a square that has the minimum number of tokens in it.
  - **Hint**: Each square in the board needs to keep track of how many tokens are placed in it. This may require you to modify *struct square* in *game_init.h*

Function *place_tokens* (3/3)

```
/*
 * Place tokens in the first column of the board
 *
 * Input: board - a 6x9 array of squares that represents the board
 *        players - the array of the players
 *        numPlayers - the number of players
 */
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){

    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;

    // TO BE IMPLEMENTED

}
```

- Should keep track of the minimum number of tokens placed on each square in the first column of the board.
  - **Hint:** This number should be incremented by one every time 6 (= *NUM_ROWS*) or a multiple of 6 tokens are placed in the squares of the first column.

Skeleton Solution

Structs to represent game entities

```
//defines a token.
//Note each token can be associated with a color
typedef struct token{
   enum color col;
};

//Defines a square of the board.
typedef struct square{
    //A square can be a NORMAL or an OBSTACLE square
    enum stype type;
    //the stack of tokens that can be placed on the board square
    token * stack;
    //the number of tokens of a square
    int numTokens;
};

/*
 * You need to fill this data structure
 * with the information about the player
 * such as a name and a color.
 */
typedef struct player{
    char name[20];
    enum color col;
    int numTokensLastCol;

};
```

Structs to represent game entities

```
//defines a token.
//Note each token can be associated with a color
typedef struct token{
   enum color col;
};

//Defines a square of the board.
typedef struct square{
    //A square can be a NORMAL or an OBSTACLE square
    enum stype type;
    //the stack of tokens that can be placed on the board square
    token * stack;
    //the number of tokens of a square
    int numTokens;
};

/*
 * You need to fill this data structure
 * with the information about the player
 * such as a name and a color.
 */
typedef struct player{
    char name[20];
    enum color col;
    int numTokensLastCol;

};
```

The struct representing a square should also include the number of tokens in it.

Structs to represent game entities

```
//defines a token.
//Note each token can be associated with a color
typedef struct token{
   enum color col;
};

//Defines a square of the board.
typedef struct square{
    //A square can be a NORMAL or an OBSTACLE square
    enum stype type;
    //the stack of tokens that can be placed on the board square
    token * stack;
    //the number of tokens of a square
    int numTokens;
};

/*
 * You need to fill this data structure
 * with the information about the player
 * such as a name and a color.
 */
typedef struct player{
    char name[20];
    enum color col;
    int numTokensLastCol;

};
```

The struct representing a square should also include the number of tokens in it.

A player is characterized by a name and a color s/he choses and the number of his/her token in the last column

Function *place_token* (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }

}
```

Min number of tokens placed on a square in the first column of the board

Function *place_token* (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }

}
```

Min number of tokens placed on a square in the first column of the board

Example Function place_token (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }

}
```

The for cycles ensure that each player places all his/her tokens on the first column of the board

Example Function place_token (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }

}
```

The function should ask the current player where to place the next token

Example Function place_token (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }
}
```

Here you should verify whether the square selected by the user has the minimum number of tokens and whether it does not contain a token of the same color selected by the player

Example Function place_token (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

          //updates the minimum number of Tokens
          if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
              minNumOfTokens++;

        }
    }

}
```

Example Instructions to add a token to a square.

If a token is already placed on the square it will be overwritten.

Example Function place_token (Implementation is not Completed)

```
void place_tokens(square board[NUM_ROWS][NUM_COLUMNS], player players[], int numPlayers){
    //the min number of tokens placed on a square in the first column of the board
    int minNumOfTokens = 0;
    int selectedSquare = 0;

    for(int i =0; i< 4; i++){
        for (int j=0; j< numPlayers; j++){

            printf("Player %d please select a square\n", j);
            scanf ("%d", &selectedSquare);

            /* TO BE IMPLEMENTED: if the square contains the minimum number of tokens and
             does not have a token of the same color of the player */

            board[selectedSquare][0].stack = (token *) malloc(sizeof(token));
            board[selectedSquare][0].stack->col = players[j].col;
            board[selectedSquare][0].numTokens++;

            //updates the minimum number of Tokens
            if(((numPlayers * i) + j +1)%NUM_ROWS ==0)
                minNumOfTokens++;

        }
    }
}
```

This min number of tokens on the first column of the board is incremented by one every time 6 (= *NUM_ROWS*) tokens or a multiple of 6 tokens are placed on the first column.

Function *play_game*

Should manage the turns of the game

Assumption (Only for This Week)

- You may ignore the case when a player lands on an obstacle square

Function *play_game*

Should manage the turns of the game

For each player this function should

1. roll the dice
2. optionally ask the player to move one of his/her tokens up or down.
3. move right any token placed in a row number which is equal to the number rolled with the dice.
   - If a player moves his/her own token to the last column, the *numTokensLastCol* of the player should be incremented. If that is equal 3 the game finishes and the function should return the number of the winning player.