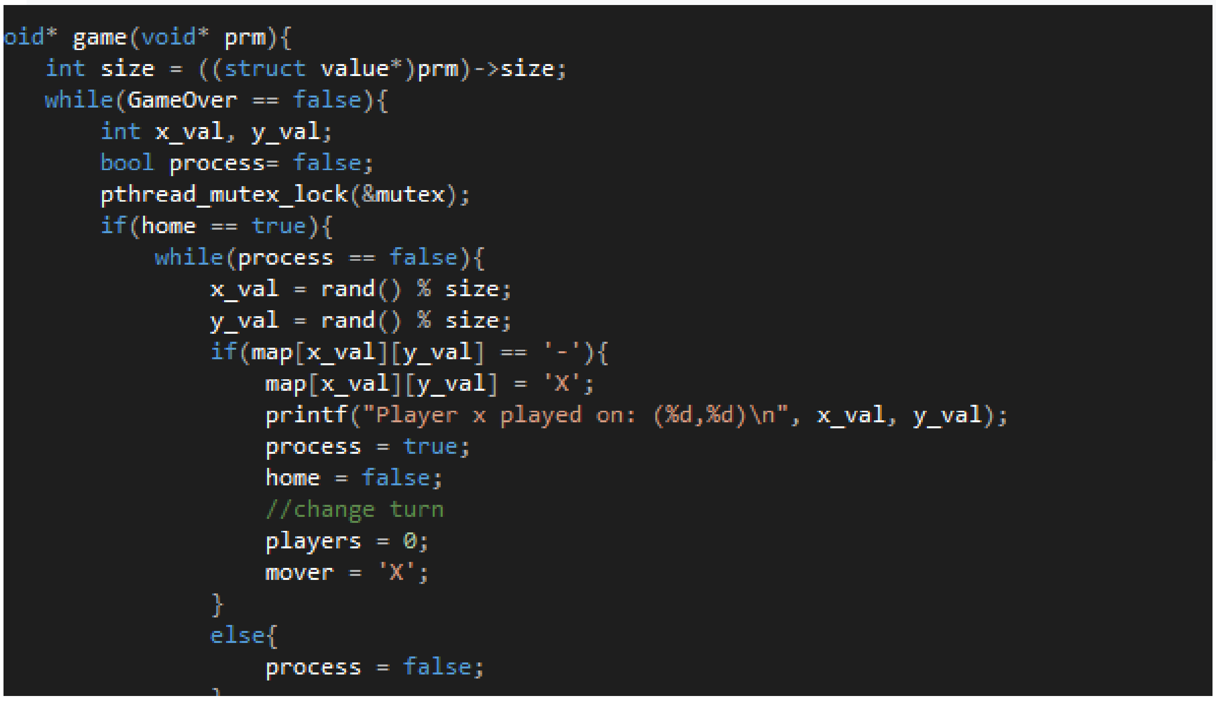
ORKUN ERDEM 20701 Summer 2021 CS307 OPERATING SYSTEMS PA2 REPORT

I used mutex threads in this project. For each resource shared by different Threads, a Mutex is created to regulate access to the resource. The code region where the shared resource is accessed is called the Critical Section. Thread with resource tries to take ownership (Acquire) of Mutex. If the Mutex is not currently held by another Thread, it takes the Thread Mutex, enters the Critical Section and uses the relevant resource. In the other case, that is, if the Mutex is currently being used by another Thread, the second Thread is put on hold by the processor. I used this thread structure because it makes sense in a turn-based game. Threads are working with high efficiency. I thought it was efficient in this project since it is a data input that changes sequences and threads.

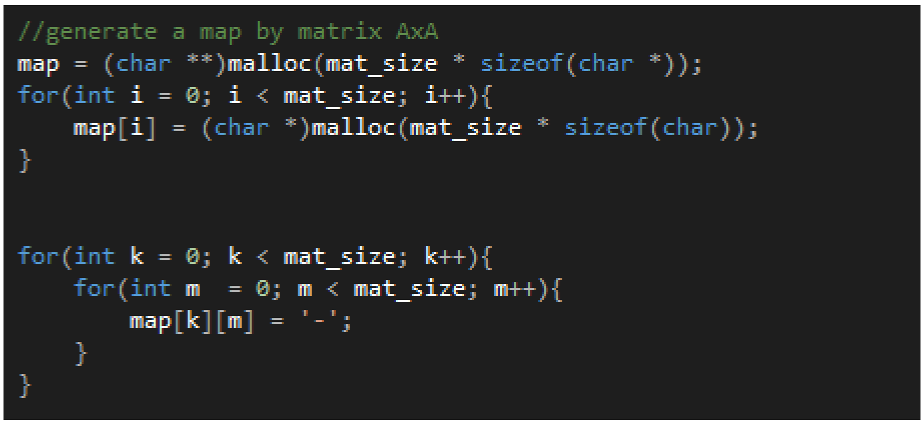
In this system, when the queue is 1, number 2 will be put to sleep. As soon as the round turns, the first thread will be put to sleep and the transition to the 2nd thread will be provided. Excessive logging to the same log file in a multithreaded application adversely affects application performance



```
oid* game(void* prm){
   int size = ((struct value*)prm)->size;
   while(GameOver == false){
       int x_val, y_val;
       bool process= false;
       pthread_mutex_lock(&mutex);
       if(home == true){
           while(process == false){
               x_val = rand() % size;
               y_val = rand() % size;
               if(map[x_val][y_val] == '-'){
                   map[x_val][y_val] = 'X';
                   printf("Player x played on: (%d,%d)\n", x_val, y_val);
                   process = true;
                   home = false;
                   //change turn
                   players = 0;
                   mover = 'X';
               }
               else{
                   process = false;
```

We create 2 threads generally and use them to player's moves.

I used matrix to keep game map so win lose conditions tracking is easy.



```
//generate a map by matrix AxA
map = (char **)malloc(mat_size * sizeof(char *));
for(int i = 0; i < mat_size; i++){
    map[i] = (char *)malloc(mat_size * sizeof(char));
}


for(int k = 0; k < mat_size; k++){
    for(int m  = 0; m < mat_size; m++){
        map[k][m] = '-';
    }
}
```