

Developer Doc

OUTLINE

1. Development Environment	-----	Page 1
2. Solution	-----	Page 1
3. Modules	-----	Page 1
4. Data Structures	-----	Page 2
5. Data Type Declarations	-----	Page 3
6. Algorithms	-----	Page 4
7. List of Functions	-----	Page 5 – Page 7
8. Testing doc	-----	Page 8

Development Environment

The program is developed with XCode and Visual Studio code.

Solution

The solution mainly based on two 2D integer type arrays of the size the user wanted the minefield to be. The progress of the game (change of the field by user) will be stored in one of the two arrays. The solution of the game will be stored in the other one. All data (e.g. Mine or not, covered or revealed) will be stored as a specific number as an element of the arrays. Two arrays will be compared to decide the end of game.

User changes the array (making progress) by enter the coordinates of the position of the field. Time of game is recorded as well as the last running time.

Everything will be running in the terminal.

Modules

The program is divided into three modules(main.c, game.c, game.h) along with a makefile.

- main.c
Include the main function (the flow of the entire game).
- game.c
Include every function declaration which will be used in main function.
- game.h
Include every function definitions of function in game.c.
Include every h files required for the build of program.
Include the declaration of the structures and enumeration.
- makefile
Include the commands to build up the program from previous modules. Provide a shortcut to build up.

Data Structures

2 **dynamic 2D integer type arrays** are used to store the state of elements within the minefield.

Declaration	Explanation
int **userfield	For the User's side (User-field). Be displayed before the user during the process of the game. (it stores states, for example, whether the elements in the field are covered(unrevealed)).
int **datafield	Stores the actual states (the solution of field. e.g. The position where there are mines) (data-field).

(TO BE CONTINUED ON NEXT PAGE)

Data Type Declarations

Structures

Four new structures are defined to store necessary data while one default structure (tm in time.h) is used to store time.

Structure Name:	Field
Declaration:	<code>typedef struct {int **userfield; int **datafield; FieldProperty property; } Field;</code>
Explanation:	Stores two pointers points to user-field and data-field, as well as a FieldPorperty structure. It should be used as a parameter in function calls when both pointers are required and any members of FieldProperty is required.

Structure Name:	FieldProperty
Declaration:	<code>typedef struct { int width; int height; int num_of_mines; } FieldProperty;</code>
Explanation:	Stores width of array, height of array and the number of mines decided by the user. This structure could be used as a parameter itself when any member of it is required but only one pointer of field is required.

Structure Name:	Coordinate
Declaration:	<code>typedef struct { int x; int y; } Coordinate;</code>
Explanation:	Stores x-coordinate and y-coordinate entered by user indicate the position of one element in the field.

Structure Name:	TimeComsumption
Library:	<code>#include <time.h></code>
Declaration:	<code>typedef struct { time_t str, end; } TimeComsumption;</code>
Explanation:	Stores starting and ending time of the game. It should be used when the calculation of how much time used by user in one game is involved in a function call.

Enumeration

Enumeration name:	state
Declaration:	<code>enum state {FREE = 0, FLAGGED = 9, COVERED = 10, EXPLODED = 11, MINE = 12} ;</code>
Explanation:	FREE: no mine, FLAGGED: is set to flag by user, COVERED: not yet revealed, EXPLODED: the mine selects by user which leads to failure, MINE: elements that contains mines. They represent the numbers store in the integer array.

Enumeration name:	action
Declaration:	<code>typedef enum {TOSWEEP = 1, TOFLAG = 2, TODEFLAG = 3, TOEXPAND = 4} action;</code>
Explanation:	They represent the four operation for users to choose.

Algorithms

1. Greet the user and display the last running time (reading from time.txt)
2. Ask the user whether he or she wants to start a new game.
 - ➔ If yes,
 - i. Ask the user for the size of minefield and the number of mines.
 - ii. Allocate memory to the user-field pointer and initialize all its elements as COVERED.
 - iii. Allocate memory to the data-field pointer and initialize all its elements as FREE.
 - iv. Record the system time (str).
 - v. Print the user-field and the number of mines remained.
 - vi. Ask the user for the first coordinate to sweep.
 - vii. Randomly set the elements of data-field to MINE according to the specified mine number and exclude the first coordinate.
 - viii. Reveal the first coordinate entered by the user and auto reveal its surroundings.
 - ix. If all mine-free elements are revealed (cleared = width*height – number of mines), jump to "xiii."
 - x. Print the user-field and the number of mines remained.
 - xi. Ask the user for the next operation. (sweep/flag/de-flag/expand)
 - xii. Ask the user for the coordinate where the operation should apply to. (at this point there should be a check to ensure the coordinate is within the field.)
 - a. If user choose to sweep,
 - 1) Check the coordinate according to data-field.
 - I. If FREE, count the number of mines surrounds the coordinate.
 - If the number is zero, set the coordinate as FREE in user-field and for every coordinate surround it, go back to "1") .
 - If the number is non-zero, set the coordinate as the number of mines of surrounding in user-field.
 - Count the cleared element (width*height – flags – covered)
 - Go back to "ix."
 - II. If MINE, Set the coordinate to EXPLODED in data-field and jump to "xiii."
 - III. If neither, print "error", go back to "ix."
 - b. If user choose to flag,
 - If the coordinate is COVERED in user-field, set the corresponding element to FLAGGED. Deduct the number of remain mines by 1.
 - Otherwise, print "a".
 - Go back to "ix."
 - c. If user choose to de-flag,
 - If the coordinate is FLAGGED in user-field, set the corresponding element to COVERED. Increment the number of remain mines by 1.
 - Otherwise, print "a".
 - Go back to "ix."
 - d. If user choose to expand,
 - If the coordinate is COVERED, do "a." for every coordinate surrounds the coordinate.
 - Otherwise, print "a".
 - Go back to "ix."
 - xiii. Record the system time (end).
 - xiv. - If jumped from "ix", print "succeed" and print the time difference (end – str).
- If jumped from "II", print "failed" and print the time difference (end – str).
 - xv. Free the memory allocated to both user-field and data-field.
 - xvi. Go back to "2."
 - ➔ If no, continue to "3."
3. Print "goodbye".
4. Record the system time and write into the time.txt file.

List of Functions

(The following list of functions are arranged in alphabetical order regardless their order of appearance in the main function.)

ArrayFree	
Library:	#include<stdlib.h>
Declaration:	void ArrayFree(int **field, FieldProperty f)
Parameter:	Pointer points to a 2D dynamic array, Structure contains size of array
Explanation:	Frees the memory allocated to a 2D dynamic array.
Return:	No return values.

AutoReveal	
Library:	
Declaration:	void AutoReveal (Field f, Coordinate coor)
Parameter:	Structure contains pointers points to both arrays, size of array and the number of mines, Structure contains the coordinates entered by user.
Explanation:	Automatically reveals the states of the surroundings if a coordinate is free of mine , until no more can be revealed.
Return:	No return values.

CoorCheck	
Library:	#include<stdbool.h>
Declaration:	bool CoorCheck(Coordinate coor, FieldProperty f)
Parameter:	Structure contains x/y-coordinates, Structure contains field properties.
Explanation:	Checks if a given coordinate is within the scope of field.
Return:	Returns TRUE if within. Return FALSE if not in the scope.

CountChosenElements	
Library:	
Declaration:	int CountChosenElements(int **user_f, FieldProperty f, int checkwhat)
Parameter:	Pointer points to the user-side-array, Structure contains field properties, Enumeration expression of the kind of states to be checked.
Explanation:	Counts the occurrences of one type of element in the entire array.
Return:	Returns the number of occurrences.

CountSurroundingMine	
Library:	
Declaration:	int CountSurroundingMine(int **data_f, Coordinate coor);
Parameter:	Pointer points to the data-array, Structure contains x/y-coordinates.
Explanation:	Counts the number of mines around a chosen coordinate.
Return:	Returns the number of mines around a chosen coordinate.

DisplayField	
Library:	#include<stdio.h>
Declaration:	void DisplayField(int **field, FieldProperty f);
Parameter:	Pointer points to an array, Structure contains field properties,
Explanation:	Prints the 2D array in field manner(with x/y-axis).
Return:	No return values.

DisplayLastRunningTime	
Library:	#include<stdio.h>
Declaration:	void DisplayLastRunningTime(void);
Parameter:	No parameter needed.
Explanation:	Reads and prints the latest running time from file.
Return:	No return values.

DisplayResults	
Library:	#include<stdio.h>
Declaration:	void DisplayResults(Field f, bool result, TimeConsumption t);
Parameter:	Structure contains both array pointers and the field properties, Bool type indicate whether user has failed or succeeded, Structure stores the starting and ending time of the game.
Explanation:	Displays the (results and time) of the game.
Return:	No return values.

InitializeField	
Library:	#include< stdlib.h>
Declaration:	int **InitializeField(FieldProperty f, enum state state);
Parameter:	Structure contains field properties, The state in enum state which you want to set all elements of the array to.
Explanation:	Allocates the memory for the 2D array of appropriate size. Initializes every element with chosen state. Memory must be freed (by the ArrayFree function) at the end of the program.
Return:	Returns the address of the array.

RandomlyInsertMine	
Library:	#include< stdlib.h>
Declaration:	void RandomlyInsertMine(int **field, FieldProperty f, Coordinate coor);
Parameter:	Pointer points to an array, Structure contains field properties, Structure contains x/y-coordinates.
Explanation:	Randomly sets appreciate number of elements in the field to mine. The coordinate provided in argument will be excluded.
Return:	No return values.

ReadAction	
Library:	#include<stdio.h>
Declaration:	action ReadAction (void);
Parameter:	No parameter needed.
Explanation:	Reads what the user wants to do with the field next.
Return:	Returns what the user wants to do in enum action.

ReadFieldProperty	
Library:	#include<stdio.h>
Declaration:	FieldProperty ReadFieldProperty(void);
Parameter:	No parameter needed.
Explanation:	Reads size of field and number of mines from the user and
Return:	Returns data received in structure format.

ReadTheCoordinate	
Library:	#include<stdio.h>
Declaration:	Coordinate ReadTheCoordinate(FieldProperty f);
Parameter:	Structure contains field properties.
Explanation:	Reads from coordinates from user.
Return:	Returns the coordinates in struct form.

RecordLastRunningTime	
Library:	#include<stdbool.h>, #include<stdio.h>
Declaration:	bool RecordLastRunningTime(void);
Parameter:	No parameter needed.
Explanation:	Writes the system time into an external file.
Return:	Returns FALSE if failed to save.

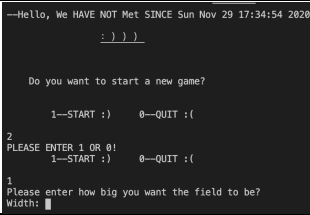
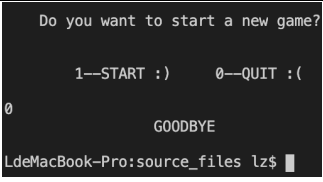
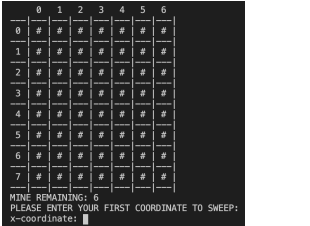


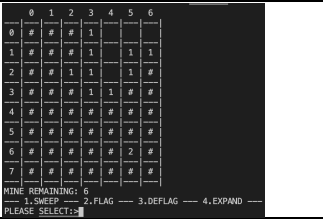

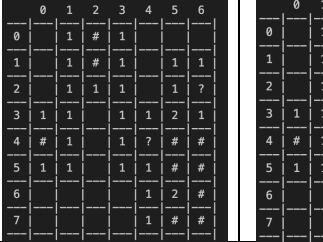
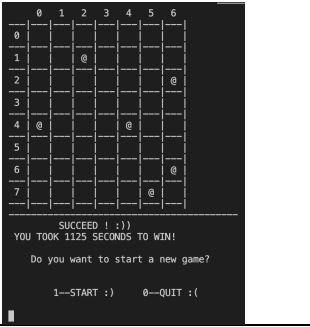
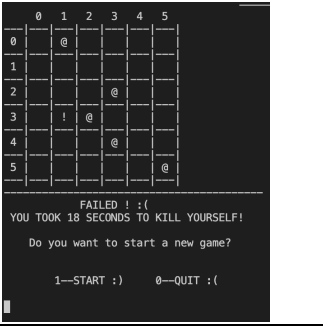
StartOrNot	
Library:	#include<stdio.h>
Declaration:	int StartOrNot(void);
Parameter:	No parameter needed.
Explanation:	Asks if the user want to start or quit.
Return:	Returns 1 if to start. Return 0 if to quit.

Sweep	
Library:	#include<stdbool.h>
Declaration:	bool Sweep(Field f, Coordinate coor, int* cleared);
Parameter:	Structure contains both array pointers and the field properties, Structure contains x/y-coordinates, Address of a variable to save the number of cleared elements.
Explanation:	Sweeps a chosen coor.
Return:	Returns FALSE if failed, TRUE if not yet.

SweepingFlaggingEnding	
Library:	#include<stdbool.h>, #include<stdio.h>, #include<stdlib.h>
Declaration:	bool SweepingFlaggingEnding(Field f);
Parameter:	Structure contains both array pointers and the field properties.
Explanation:	The main process of the game. User can select operation and coordinates, to sweep, to flag, to de-flag, to expand, until either failed (sweep the mine) or succeed (reveal all mine free place).
Return:	Returns TRUE if user succeed. Return FALSE if user failed.

Testing

- The following lists a series of screen shots confirm some basic realization of desired function.

<p>1. Start</p> 	<p>-Last running time -Correction of invalid input. -1 to start.</p>	<p>2. Quit</p> 	<p>-Quit if select 0</p>
<p>3. initialization</p> 	<p>-Printing format. -remain mines shown -correct size and mines</p>	<p>4. Sweep</p> 	<p>-Sweep (4,2);(5,6);(2,2) -Auto reveal. -Number of mines around.</p>
<p>5. Flag</p> 	<p>-Flag (2,4) -Remain mines decrement by one.</p>	<p>6. De-flag</p> 	<p>-De-flag (2,4) -Remain mines increment by one.</p>
<p>7. Expand</p> 	<p>-Expand (4,2) -elements around it are swept.</p>		<p>-Expand (4,5) -Flagged not swept.</p>
<p>8. Success</p> 	<p>- Succeed when all mine-free revealed. -Display the solution. -Time consumed displayed. -Re-ask whether to start.</p>	<p>9. Fail</p> 	<p>-Fail when sweep mine. -the exploded mine is indicated with solution. -Time consumed displayed. - Re-ask whether to start.</p>

(* The test just provides a brief outlook. Not every detail is tested.)

- The memory consumption after repeatedly new games. (No leakage)

