Curtin University — Discipline of Computing

Unix & C Programming (COMP1000)

Semester 1, 2021

Assignment One

Due: Friday, April 16th, 11:59 PM (GMT+8)

Weight: 15% of the unit

Your task for this assignment is to design, code (in C89 version of C) and test a program. In summary, your program will:

- Read the parameters from command line and utilize them for the game configuration.
- Able to create dynamically-allocated 2D char array to make a simple ASCII-based game.
- Receive user input to control the flow of the game.
- Write a suitable makefile. Without the makefile, we will assume it cannot be compiled.

1 Code Design

You must comment your code sufficiently in a way that we understand the overall design of your program. Remember that you cannot use "//" to comment since it is C99-specific syntax. You can only use "/*.....*/" syntax.

Your code should be structured in a way that each file has a clear scope and goal. For example, "main.c" should only contain a main function, "map.c" should contain functions that handle and draw the map, and so on. Basically, functions should be located on reasonably appropriate files and you will link them when you write the makefile. DO NOT put everything together into one single source code file. Make sure you use the header files and header guard correctly.

Make sure you free all the memories allocated by the malloc() function. Use valgrind to detect any memory leak and fix them. Memory leaks might not break your program, but penalty will still be applied if there is any.

Please be aware of our coding standard (can be found on Blackboard under "Resources") Violation of the coding standard will result in penalty on the assignment. Keep in mind that it is possible to fail the assignment with a fully-working program that is written messily, has no clear structure, full of memory leaks, and violates many of the coding standards. The purpose of this unit is to teach you a good habit of programming.

2 Academic Integrity

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from anyone else on completing the tasks. You must not show your work at any time to another student enrolled in this unit who might gain unfair advantage from it. These things

are considered plagiarism or collusion. To be on the safe side, keep your source code private until next year.

Do NOT just copy some C source codes from internet resources. If you get caught, it will be considered plagiarism. If you put the reference, then that part of the code will not be marked since it is not your work.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for you to solve them on your own, so that you learn from it. Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to attend quick interview to answer questions about any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by systems to detect plagiarism and/or collusion.

3 Task Details

3.1 Quick Preview

First of all, please watch the supplementary videos on the Assignment link on Blackboard. These videos demonstrates what we expect your program should do. You will implement a simple game inspired from a classical puzzle game "Laser Tank". Further details on the specification will be explained on the oncoming sections.

3.2 Command Line Arguments

Please ensure that the executable is called "laserTank". Your executable should accept eight (8) command-line parameters/arguments:

./laserTank <row_size> <col_size> <player_row> <player_col> <player_direction> <enemy_row> <enemy_col> <enemy_direction>

- <row_size> and <col_size> determines the size of the playable map. The minimum and maximum value for <row_size> and <col_size> are 5 and 25 respectively. You need these numbers for dynamic memory allocations of 2D char array. (use malloc() function)
- <player_row> and <player_col> are the coordinates of the player tank. You can use these numbers as the array index of the map. Coordinate <0,0> is located at top left of the map.
- <player_direction> is the direction the player is facing upon starting the game. There are 4 choices: u, d, l, r, (lowercase letter) which is derived from the word up, down, left, right.
- <enemy_row>, <enemy_col>, and <enemy_direction> have the same meaning as the previous explanations. However, these arguments configure the enemy tank.

This is one example: ./laserTank 10 20 1 1 r 5 10 u

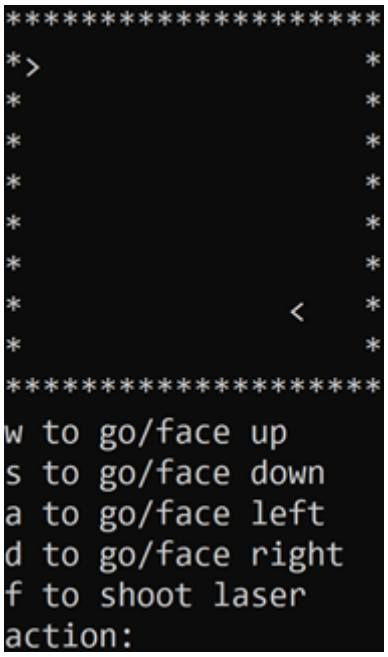
Note: Remember, the name of the executable is stored in variable argv[0].

3.3 Map

The map can be derived from a 2D char array, which has been dynamically allocated based on the command line arguments. For more information about the characters you can use on the map, please watch the supplementary video.

3.4 Main Interface

Once you run the program, it should clear the terminal screen (watch the video about "clear and newSleep" functionality) and print the map along with the instructions:



This is the interface where the user plays the game. The user can type the command after the sentence "action:". Every time the user inputs the command, the program should update the map accordingly and refresh the screen. (clear the screen and reprint everything)

3.5 User Input

The user only need to type 1 character for each command (lower case). Here is the list of the possible commands:

- **'w'** moves the player one block above. If the player tank is not facing upwards, this command will only cause the tank to face upwards.
- **'s'** moves the player one block below. If the player tank is not facing downwards, this command will only cause the tank to face downwards.
- **'a'** moves the player one block to the left. If the player tank is not facing leftwards, this command will only cause the tank to face leftwards.

- **'d'** moves the player one block to the right. If the player tank is not facing rightwards, this command will only cause the tank to face rightwards.
- **'f'** causes the player tank to shoot the laser on the direction it is facing. Please refer to the next section on "Laser Animation" for more details. Additionally, please watch the supplementary video on "Assignment Demonstration".
- Any other character will be ignored. The program still need to refresh the interface to enter new command. You can add warning message if you want.

Note: If the player attempts to move to the same location as the map border OR the enemy tank, it should not do anything. (player location stays the same)

3.6 Laser Animation

When the user provides the command 'f', there will be an additional character representing the laser on the map. You can use the character '-' if the tank is facing horizontal direction, or the character '|' if the tank is facing the vertical direction. When the laser is being shot, the program should trigger a simple animation of the laser moving from the front of the player tank. The laser will keep moving forward until it hits the border OR the enemy tank. When the laser hits the enemy tank, its character should be replaced with 'X'.

In order to make a simple animation, you can use the newSleep function provided in the Blackboard (and also watch the supplementary video). The idea is to move the position of the laser one block at a time with a very brief sleeping period in between. We recommend sleeping time less than 0.5 seconds for reasonable speed. When the animation is still running, the program should NOT show the game instructions and NOT prompting the user for any command.

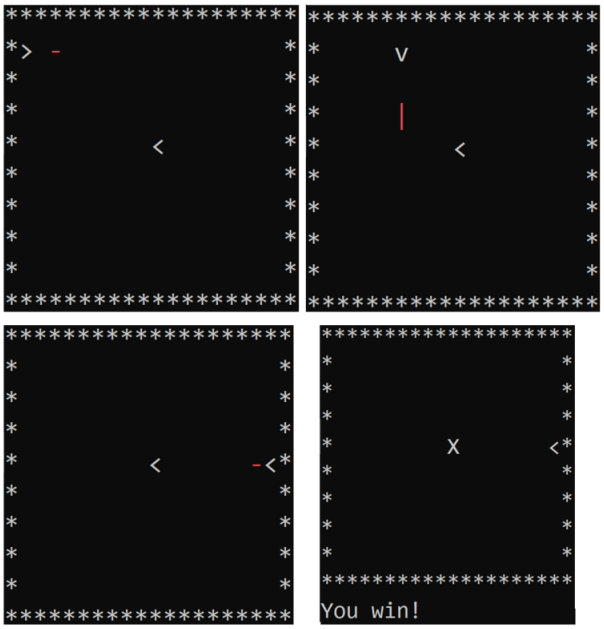
The laser should be printed on the map using font color other than the default color. We have provided a small example on the Blackboard for this task. Additionally, please utilize search engine to search the term "ANSI color in c" for more details.

Note: If the enemy tank is exactly in front of the player (no distance), then there is no need to show the laser, and the enemy tank should get destroyed immediately and the game should conclude.

3.7 Winning/Losing Conditions

The game will continue until the player either win or lose. This is the condition for each result:

- **WIN**: The player tank manage to hit the enemy tank with the laser. This can only happen when the player tank shoots the laser while facing the enemy tank, regardless of the distance between them. The enemy tank's character will change into a character 'X' (uppercase X).
- **LOSE**: The player moves in front of the enemy tank (facing the player). The enemy tank will immediately shoots the laser to the player (with the laser animation). The player cannot move when the animation is happening. The player's character will change into a character 'X'.



When this occurs, please do not forget to **free** all the memories allocated via malloc() function before the program ends.

3.8 Makefile

You should write the makefile according to the format explained in the lecture and practical. It should include the Make variables, appropriate flags, appropriate targets, and clean rule. We will compile your program through the makefile. If the makefile is missing, we will assume the program cannot be compiled, and you will lose marks.

3.9 Assumptions

For simplification purpose, these are assumptions you can use for this assignments:

- The coordinates of the player tank and enemy tank will not overlap from the provided command line arguments. But, you still need to check whether the coordinates are inside the map boundary.
- The coordinate of the tanks will not overlap with the border of the map as well.
- The player tank will NOT be in front of the enemy tank immediately upon starting the game. (NO instant lose)
- Any command or argument is always on correct datatype and in lower case (for letters).

4 Marking Criteria

This is the marking distribution for your guidance:

- Properly structured makefile (10 marks)
- Program can be compiled with the makefile and executed successfully without immediate crashing (5 marks)
- Usage of header guards and sufficient in-code commenting (5 marks)
- The whole program is readable and has reasonable framework. Multiple files are utilized with reasonable category (5 marks). If you only have one c file for the whole assignment, you will get zero mark on this category.
- Avoiding bad coding standard. (10 marks) Please refer to the coding standard on Blackboard. Some of the most common mistakes are:
  - Using global variables
  - Calling exit() function to end the program abruptly
  - Using "break'' not on the switch case statement
  - Using "continue''
  - Having multiple returns on a single function.
- No memory leaks (10 marks) Please use valgrind to check for any leaks. If your program is very sparse OR does not use any malloc(), you will get zero mark on this category.
- Functionalities:
  - Able to read and interpret command line arguments correctly. This includes clarifying the argument amount and values. If there is any issue, then the error message is printed on terminal and end the program. Otherwise, the map array should be allocated accordingly. (10 marks)
  - Correct characters usage for the map and its components. (5 marks)
  - Player tank moves and changes direction correctly based on player command. (10 marks)
  - The laser is animated correctly on the map with the newSleep function. (15 marks)
  - (BONUS mark) The laser is changing color every time it moves by one block. (hint: use static variable to keep track.) (5 marks) (total mark is still capped at 100 marks.)
  - Able to refresh the screen with the updated map/instructions visualization every time the user enters the command. (5 marks)
  - Winning/Losing condition is implemented correctly. (make sure to free the memory before wrapping up the program) (10 marks)

5 Short Report

If your program is incomplete/imperfect, please write a brief report explaining what you have done on the assignment. This report is not marked, but it will help us a lot to mark your assignment. Please ensure your report reflects your work correctly (no exaggeration). Dishonest report will lead to academic misconduct.

6 Final Check and Submission

After you complete your assignment, please make sure it can be compiled and run on our Linux lab environment. If you do your assignment on other environments (e.g on Windows operating system), then it is your responsibility to ensure that it works on the lab environment. In the case of incompatibility, it will be considered "not working'' and some penalties will apply. You have to submit a tarball (see practical worksheet for the tarball creation instruction) containing:

- **Declaration of Originality Form** – Please fill this form digitally and submit it. You will get zero mark if you forget to submit this form.
- **Your essential assignment files** – Submit all the .c & .h files and your makefile. Please do not submit the executable and object (.o) files, as we will re-compile them anyway.
- **Brief Report** (if applicable) - If you want to write the brief report about what you have done on the assignment, please save it as a PDF or TXT file.

The name of the tarball should be in the format of:

<student-ID>_<full-name>_Assignment1.tar.gz
Example: 12345678_Antoni-Liang_Assignment1.tar.gz

Please make sure your submission is complete and not corrupted. You can re-download the submission and check if you can compile and run the program again. Corrupted submission will receive instant zero. You can submit it multiple times, but only your latest submission will be marked.

End of Assignment