COMP 201 - Spring 2022
Assignment 3 - Heap Management - WoW: Alliance vs. Horde

Assigned: 24 March 2022 23:59, Due: 7 April 2022 23:59

Mert Cokelek (`mcokelek21@ku.edu.tr`) is the lead person for this assignment.

1. Introduction

You are expected to implement a board game version of ”World of Warcraft” in this assignment. Your program should take two input files as command-line arguments for initializing and playing the game. In World of Warcraft, there are two factions: Alliance and Horde. Your game is expected to simulate the spawning, movement, and attacking of the characters of these sides. The corresponding messages should be printed to the terminal based on the input scenario.

2. Background

The sides of this war and the characters are inspired by the well-known role-playing game World of Warcraft. In our version of the game, all characters will have health and damage points. Unlike the Horde, the Alliance side will have additional experience points. At the beginning of your program, the game map will be created, and the characters will be spawned in the given positions on that map. Later, based on a given sequence of commands, the warriors of Alliance will be able to move across the map, and the characters will have combat.

Horde characters are initially spawned in different locations and do not move. At some point, if any two characters from different sides are adjacent, they can attack each other. In this stage, the defender's health point (HP) will be decreased by the damage point of the attacker. If a character's HP reaches 0, it dies. If an Alliance kills a Horde, it earns one experience point (XP). Recall that only Alliance can have XPs. The details will be in the following sections.

3. Handout Instructions:

Accept the GitHub Classroom assignment using the link: https://classroom.github.com/a/0gfppA3w. Clone the GitHub repository created for you to a Linux machine in which you plan to do your work (We advice you to do your work on our linux servers [linuxpool.ku.edu.tr].

```
git clone https://github.com/COMP201-Spring22/assignment-3-USER.git
```

(Replace USER with your GitHub username that you use to accept the assignment).

- main.c: The main file of the assignment that should be submitted to be graded.
- chars_<input_number>.txt
- commands_<input_number>.txt

The details for the input and outputs are in the following sections.

4. Tasks

4.1 Spawning Characters

In the beginning of the game, you will read **2 input files**. The first one is: **chars_<input_number>.txt**. This input file contains the properties of Alliance and Horde characters. These properties are: name, HP, damage, in the given order, separated by comma. Each line contains information of a character, with the format: **<type>,<name>,<hp>,<damage>**.

A sample for the first input file is given below, and all the explanations in this handout will be based on this example:

```
ALLIANCE,ELF,8,3
HORDE,VULPERA,10,2
ALLIANCE,IRON,6,3
HORDE,TROLL,4,3
HORDE,GOBLIN,1,3
```

Here are some important points you should keep in mind:

- There can be any number of Alliance and Horde characters.
- **The attacking order** will be the same as spawning order of the characters.
  - For this example, when its Alliance's turn to attack, the order of attacking will be: **"ELF, IRON"**.
  - For the Horde, it will be: **"VULPERA, TROLL, GOBLIN"**.

4.2 Playing the Game

The second input file argument **commands_<input_number>.txt**, contains a sequence of commands, to play a game. Since we first need to load the map and spawn the characters, the game will always begin with the following three lines. Keep in mind that the parameters (size of the map, location of the characters) can and will be different.

```
LOADMAP,5 5
PUT,ALLIANCE,ELF,0,0,IRON,4,1
PUT,HORDE,TROLL,1,1,GOBLIN,4,0,VULPERA,0,1
```

Apart from these default commands in the beginning, there will be other commands for playing. They are described as the following:

- **LOADMAP**

  Loads a map with the specified size (row, column). They will be positive integers. **The map should be implemented using a 2 dimensional array, and created dynamically.**

  Input:

  ```
  LOADMAP,<number_of_rows>,<number_of_columns>
  ```

Output:
There will not be output messages for this command.

- **PUT**

  Puts the given characters on the map. <type> will be "ALLIANCE" or "HORDE". Characters can be given in any order for this command. While attacking, you should take into account the spawning order, not this.

  Input:

  ```
  PUT,ALLIANCE,<name_1>,<row>,<column>,<name_2>,<row>,<column> ...
  PUT,HORDE,<name_1>,<row>,<column>,<name_2>,<row>,<column> ...
  ```

  Output:
  There will not be output messages for this command.

- **SHOW (30 points)**

  Prints the current status of either Alliance, Horde, or the map.

  ***Input** (e.g. in the beginning of the game):*

  ```
  SHOW,ALLIANCE
  ```

  Output:

  ```
  ALLIANCE STATUS
  ELF HP: 8 XP: 0
  IRON HP: 6 XP: 0
  ```

  ***Input**: (e.g. in the beginning of the game)*

  ```
  SHOW,HORDE
  ```

  Output:

  ```
  HORDE STATUS
  VULPERA HP: 10
  TROLL HP: 4
  GOBLIN HP: 1
  ```

  ***Input**: (e.g. in the beginning of the game)*

  ```
  SHOW,MAP
  ```

  Output:

```
MAP STATUS
E V . . .
. T . . .
. . . . .
. . . . .
G I . . .
```

- **ATTACK (30 points)**

  Executes the attack command for the given <**type**>. The characters for the given type should attack the other types. There are some rules you should consider:

  - Characters attack with their spawn order.
  - Characters can only attack their adjacent enemies. There are a total of 8 directions. These directions have a priority as well.

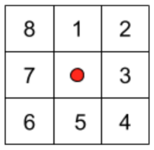


Figure 1: Attacking order for a character in the red cell is given numerically.

  - For the example in the above map, for **ATTACK,ALLIANCE** command, ELF will attack the Hordes before IRON. ELF will attack VULPERA first, and then TROLL. Some other cases for attacking orders are illustrated below:

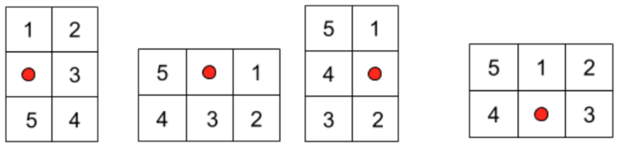


Figure 2: Attacking orders for the characters in the red cells, in the map borders.

  - The same types of characters do not attack each other.
  - To simulate the strike, you should decrease the HP of the damaged character, by the damage value of the attacker. So, for the same example, after a Horde attack, ELF will have HP=3 (8-2-3), and IRON will have 3 (6-3).
  - A character's HP can not be less than 0. If a character reaches HP=0, it should be removed from the map. However, its status should still be printed in the SHOW command.
  - If an Alliance kills a Horde, its XP should be increased by 1 point. If a Horde kills an Alliance, nothing happens.

Input:

```
ATTACK,<type>
```

Output:

```
<type>s ATTACKED
```

- **MOVE (30 points)** This command will be only for the Alliance. There are some important points here:
  - You should check if the new position is valid (unoccupied, and inside the boundaries of the map).
  - A dead character can not be moved.
  - A character can move only one unit at a time. (You don't need to handle this case, the inputs will be given accordingly.)
  - Characters can be given in any order for this command.

  Input:

  ```
  MOVE,ALLIANCE,<name_1>,<row>,<column>,<name_2>,<row>,<column> ...
  ```

  Output:

  ```
  ALLIANCES MOVED
  # If the target position is occupied by another character:
  <character_name> can't move. Place is occupied.
  # If the target position exceeds the map:
  <character_name> can't move. There is a wall.
  # If the character is dead:
  <character_name> can't move. Dead.
  ```

- **ENDING (10 Points)** After each ATTACK command, you should check whether a type of characters are all dead. In that case, you should print a message and terminate the program, even there are more commands in the commands file. Message is:

  ```
  ALL <type>s ARE DEAD!
  ```

  Other than that, the game may not be finished according to the given scenario. In other words, at the end of the game, there still might be alive Alliance and Hordes. In that case, you don't need to print anything.

5. Important Points

- You should use **dynamic arrays** and **structs**, for the map and the characters. Other methods will not be accepted.
- The game map should be created as a 2D dynamic array. Other methods will not be accepted.
- Do not forget tor free the allocated memory. You can check the memory leaks using **valgrind**.
- Your outputs should be printed to the terminal, and they should comply with the given output formats.

6. Submission

As with the previous assignments, we use GitHub for the submissions as follows. Note that we want you to get used to using a version management system (Git) in terms of writing good commit messages and frequently committing your work so that you can get most out of Git.

1. Commit all the changes you make:

   ```
   git commit -a -m "commit message"
   ```

   Note: please use meaningful commit messages because

2. Push your work to GitHub servers:

   ```
   git push origin main
   ```

Please submit your `main.c` file, with the untouched **Makefile** and input files. A starter code for the basic skeleton will be provided. However, feel free to make any changes as you wish, taking into account the important points mentioned in the previous sections.

7. Evaluation:

Your score will be computed out of a maximum of 100 points based on the following distribution:

- SHOW (30%)
- ATTACK (30%)
- MOVE (30%)
- END (10%)

Testing on the inputs: You are provided a Makefile for the given sample inputs. Just type

```
make compile_and_test1
         or
make compile_and_test2
```

to test your game.

Effective use of version control: You are required to push your changes to the repository frequently. If you only push the final version, even if it is implemented 100% correctly, you will lose a fraction of the grade because you are expected to learn to use Version Control Systems effectively. You do not have to push every small piece of change to GitHub but every meaningful change should be pushed. For example, each of the tasks coded and tested can be one commit. For each task, there should be at least one commit (with proper commit message) that includes just modifications on that function.

Important Note: We use automated plagiarism detection to compare your assignment submission with others and also the code repositories on GitHub and similar sites.

Oral Assessment: As usual, we are going to ask randomly selected 10% of students to explain their approach verbally after the assignments are graded. And one may lose full credit due to a failure o this oral part.

8. How to use linuxpool.ku.edu.tr linux servers

I Connect to KU VPN (If you are connected to the KU network, you can skip this step.)
See for details: https://confluence.ku.edu.tr/kuhelp/ithelp/it-services/network-and-wireless/vpn-access

II Connect to linuxpool.ku.edu.tr server using SSH (Replace USER with your Koç University username):

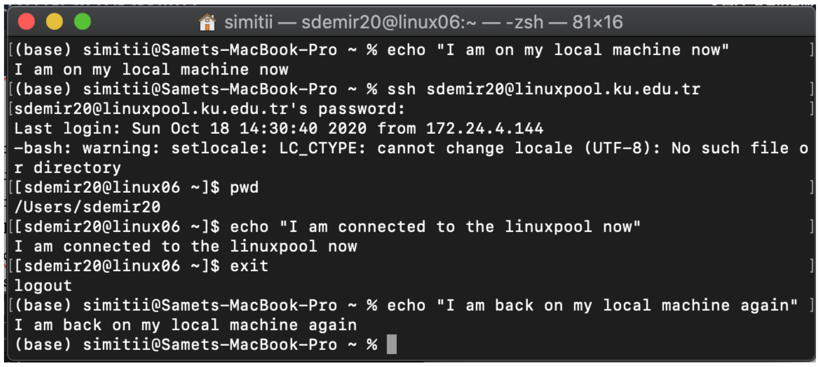
```
ssh USER@linuxpool.ku.edu.tr
```

(It will ask your password, type your Koç University password.)

III When you are finished with your work, you can disconnect by typing: `$ exit`

Your connection to the server may drop sometimes. In that case, you need to reconnect.

We advise you to watch the following video about the usage of SSH, which is used to connect remote servers, and SCP, which is used to transfer files between remote servers and your local machine: https://www.youtube.com/watch?v=rm6pewTcSro



```
[(base) simitii@Samets-MacBook-Pro ~ % echo "I am on my local machine now"        ]
 I am on my local machine now
[(base) simitii@Samets-MacBook-Pro ~ % ssh sdemir20@linuxpool.ku.edu.tr            ]
[sdemir20@linuxpool.ku.edu.tr's password:                                           ]
 Last login: Sun Oct 18 14:30:40 2020 from 172.24.4.144
 -bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file o
 r directory
[[sdemir20@linux06 ~]$ pwd                                                          ]
 /Users/sdemir20
[[sdemir20@linux06 ~]$ echo "I am connected to the linuxpool now"                  ]
 I am connected to the linuxpool now
[[sdemir20@linux06 ~]$ exit                                                         ]
 logout
[(base) simitii@Samets-MacBook-Pro ~ % echo "I am back on my local machine again" ]
 I am back on my local machine again
 (base) simitii@Samets-MacBook-Pro ~ %
```

Figure 3: How to connect and disconnect using SSH

9. Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the web material as everything on the web has been written by someone else. See Koç University - Student Code of Conduct.

10. Late Submission Policy

You may use up to 7 grace days (in total) over the course of the semester for the assignments. That is, you can submit your solutions without any penalty if you have free grace days left. Any additional unapproved late submission will be punished (1 day late: 20% off, 2 days late: 40% off) and **no submission after 2 days will be accepted.**

Acknowledgement

This assignment is adapted from a C Programming Assignment in Hacettepe University.