Tecnológico de Monterrey

Advanced Programming

Final Project Proposal

“The Skyscraper of Doom”

Professor: Gilberto Echeverría

Isabel Maqueda Rolon	A01652906
Eduardo Badillo	A01020716

9/12/2019

PROBLEM DESCRIPTION

For the final project we will make a “Zork” style, text-based, video game. Various players will be able to connect simultaneously to a game. There will be no connection between players. Once connected, they get to choose one of the next three roles: Warrior, mage or healer. Each player will receive a scenario. Players will control their respective characters writing simple commands like: “ATTACK, DEFEND, etc”. The game starts the players with a random selected scenario. The players must go up, trying to get to the highest possible level until they are defeated. Each floor will have a randomly generated hazard/situation for the players. Hazards include: enemies, ambiguous situations that involve decision-making, and loot to equip or help the players. Enemies and other hazards can harm the player. The order and effectivity of the actions taken is determined by the stats of the player, which can improve or worsen, depending on luck and player’s choice. If a player gets defeated the game will end for the player.

PROPOSED SOLUTION

- We will use threads to handle each player and the random generated scenarios. The players will have no communication between them. We will use mutexes to handle the global variable of levels played in one instance.

- There will be several structs to handle the stages, and the global variables. We will use sockets to connect to the server.

- Situations in each floor will be randomly generated, the description of the situations will be stored in a text file with identificators next to them to select them.

EXAMPLE (in situations.txt): _____
SIT 2 TYPE: 0 ENEMIES: 2 DESC: “The party members found themselves surrounded by evil, vicious marshmallow people ready to attack them.”

SIT : 3 TYPE:1 ENEMIES:0 DESC: “The party found themselves in an empty room.”

In this example, the random function will have selected the situation number 2. Then it will read that it is a type 0 situation, meaning that the party will need to confront a group of enemies to clear the level. The next identificator will determine the number of enemies.

- Once a battle starts, we will have pointers to each structure (enemies and players), to determine how many enemies we need to create and how can the player act.

- We will use threads to allow the multiple players simultaneously, each on their own instance of the game.

- Each floor will instantiate a set of structs to convey the different elements happening in a floor. Each time the player clears a floor, we will have a procedure to delete from memory the elements of the last floor and make space for the next.

- The most crucial library in our project will be the “string.h” library, given that we are storing everything in text files.

- We will create functions for recovering info from text files, to clear memory and to call certain elements from the structs.

- Battles and situations will be handled in functions that randomly determine the loot that the player will receive.

- A cycle will be implemented in the client program that will continue receiving situations from the server until the player gets eliminated by one of them. At that point the client will send a DEAD message to the server indicating the game has concluded.

TOPICS USED

- POINTERS: We use pointers to utilise less memory and to avoid creating unnecessary variables for objects that already exist elsewhere. Pointers to structures are passed to different functions to alter the stats of characters.

- THREADS: We use threads to create multiple games at the same time and host a different player per game. We share between players the number of floors they all traversed, which is made easy with thread handling.

- DYNAMIC MEMORY: Each time a floor is cleared, wel free memory of every pointer to struct used. And the arrays created . The memory will also be cleared each time the server closes.

- SIGNALS: The server will block all signals except SIGINT, and the program will detect the signal and close all the threads created and send a goodbye message.

FUNCTIONS AND LIBRARIES USED:

For the program, for the connection of the server and the client and the sending and receiving of information, we used the Socket Library created by Guilberto Echeverria, which contains the following functions:

Sockets:

void printLocalIPs():

Shows the local IP address

int initServer(char * port, int max_queue):

Initializes the socket and opens and returns the file descriptor for the socket

int connectSocket(char * address, char * port):

Connects the socket to the server

int recvString(int connection_fd, void * buffer, int size):

Receives a stream of data from the socket

void sendString(int connection_fd, void * buffer, int size):

Send a stream of data to the socket

ERROR HANDLING:

For error detection and handling we used a Library called fatal_error

void fatalError(const char * message):

exits the program if it detects errors and prints the error for easier error recognition.

SERVER:

The server reads the text file where the stages are stored and connects to clients. The server blocks all signals except SIGINT and has a signal handler to store changes before it quits. It has the structures used for storing and managing the structures used in the game and the connection to the socket.

The structures used for the server are the following:

- stage_struct: It stores the stage id, the type, the number of enemies and its description.
- game_struct: it stores an array of stages and the total levels played in one iteration.
- locks_struct: it stores the locks used for the managing of information.
- data_struct: it stores the id of the connection as well as a pointer to the game_struct and locks_struct

void usage(char * program)

Explanation to the user of the parameters required to run the program

void setupHandlers();

it sets up the signal handlers to manage the SIGINT signal.

void onInterrupt(int signal);

Handler for SIGINT

sigset_t setupMask();

Blocks all signals except SIGINT

void unsetMask(sigset_t old_set);

Restores the mask to the original

void initGame(game_t * game_data, locks_t * data_locks);

Initializes all structs with the memory space and the mutexes. And calls the function to read the text file to the struct

void readStageFile(game_t * game_data);

Reads a text file and stores it in an array of structs.

void waitForConnections(int server_fd, game_t * game_data, locks_t * data_locks);

Manages the connections to the socket. Checks if the program catches SIGINT and if so, finishes it and send the signal to the client. Checks for connections to the server and

creates a new thread to manage new clients, passing as parameters the structs with the game. Uses polling and threads.

void closeGame(game_t * game_data, locks_t * data_locks);

Frees the memory used by the server and the program.

void * attentionThread(void * arg);

Receives the parameters passed by the creation of a thread, and sends the clients a random stage for them to complete.

CLIENT:

The client first connects to a server, and then allows for the user to choose which character to use, with the stats defined. Each character has a weapon, and a number indicating speed, damage, and health. The client receives a stage from the server and depending of the type, with the number of enemies it receives it creates the enemies with random stats for health, speed and damage. The other type creates the loot inside the chests with random potions, that could restore or steal health, deal damage or an upgrade weapon.

the structs used in the client are:

- CharacterInfo: Stores the info of the character and villains, with weapon, health, damage and speed.
- EnemyParty: is an pointer to an array of characters type villain, with random stats, and the number of enemies.

void usage(char * program);

void initPlayer(char_st * player);

Initialize player character with the role that the player chooses, the player can choose three characters, the warrior, the mage and the healer.

void initEnemies(eparty_st *vil, int enemies);

Initialize enemies with randomized stats. The enemies each have different stats.

void freeEnemies(eparty_st * vil);

Frees the memory allocated for the enemies

void printInstructions();

Print the instructions of the game.

int fleeFunction(char_st * player, eparty_st * enemies);

Function that controls how characters react when player chooses to flee, if you flee but the enemy has a speed greater than yours, you lose health points. If you flee but you have more speed than the enemy you win that round.

bool attackFunction(char_st * player, eparty_st * enemies);
Function that controls how characters react when player chooses to attack. If the enemy has a greater speed, the enemy attacks firs and the player has to respond until either is defeated. If player has greater speed, the player attacks first and the enemy responds until either are defeated.

bool defendFunction(char_st *player, eparty_st * enemies);
Function that controls how characters react when player chooses to defend. If the enemies damage is greater than yours, you lose points in health. If the players damage is greater than the enemy, the player successfully defended and the enemy loses speed.

void printEnemies(eparty_st * enemies);
Function that prints some useful indicators about the enemies in the current stage

void reviewSelf(char_st * player);
Give the player an indication of how he is performing in the game

bool battleFunction(char_st * player, eparty_st * enemies);
Function that iterates and controls battle functions until all enemies in stage are defeated. It gives options for the player to play.

bool generateChestCont(char_st *player);
Function that randomly generates loot that give buffs or debuffs to the player

void applyBuffs(char_st *player);
Apply buff to the player according to the selected role

void gameOperations(int connection_fd);
Mother function that iterates and receives server messages until the player is defeated. It receives the stages from the server and depending on stage, gives the options to play.

THE TEXT FILE USED:

SITUATION TYPE ENEMIES DESCRIPTION
0 0 2 You found yourself surrounded by evil, vicious marshmallow people ready to attack them.
1 1 0 You found yourself in a room full of water, in the middle is a wooden chest with an iron lock. You find the key

INSTRUCTIONS:
Getting Started
These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See deployment for notes on how to deploy the project on a live system.

Prerequisites
You will need to install the GNU compiler and a Make build automaton
```
**sudo apt instal ggc**
```
```
**sudo apt instal make**
```
Installing
A step by step series of examples to run the project

Install ggc

```
**sudo apt instal ggc**
```
Install make
```
**sudo apt instal make**
```
and then to run
```
**gcc client.c -o client**
```
Compile using the MakeFile
run the make build automaton

```
**make**
```
Run the Server
use ./server {port_number}

```
**./server 8989**
```
Run the Client
Open another tab in terminal and run
./client {server_address} {port}

```
**./client localhost 8989**
```

Deployment

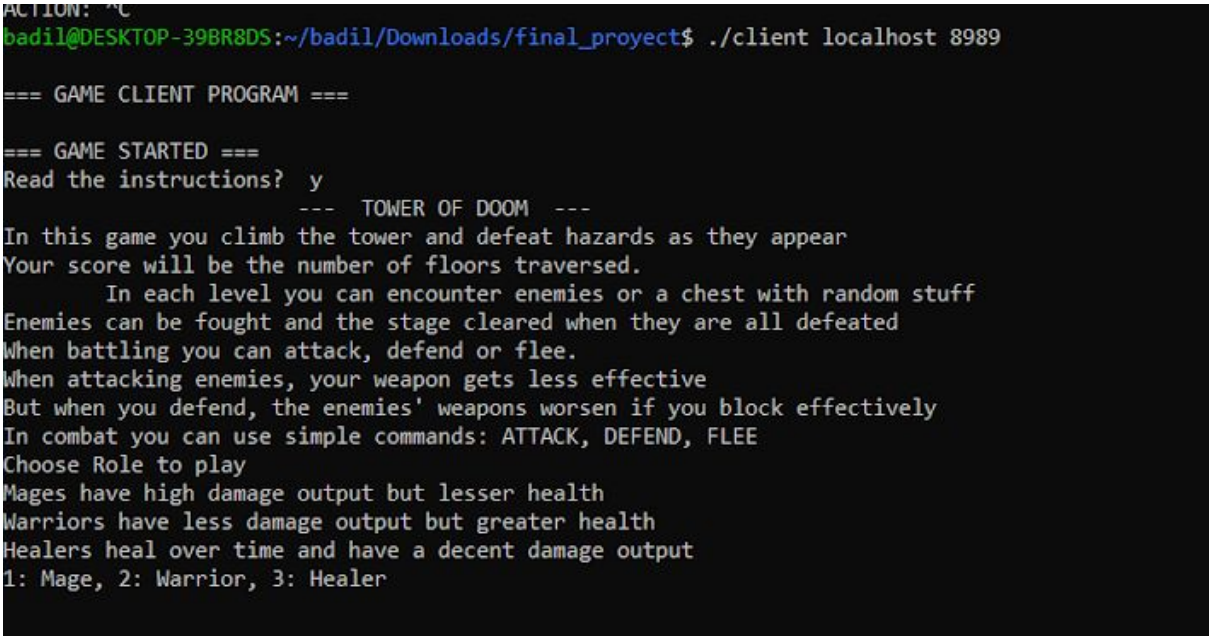
Run the program. Further instructions will follow.

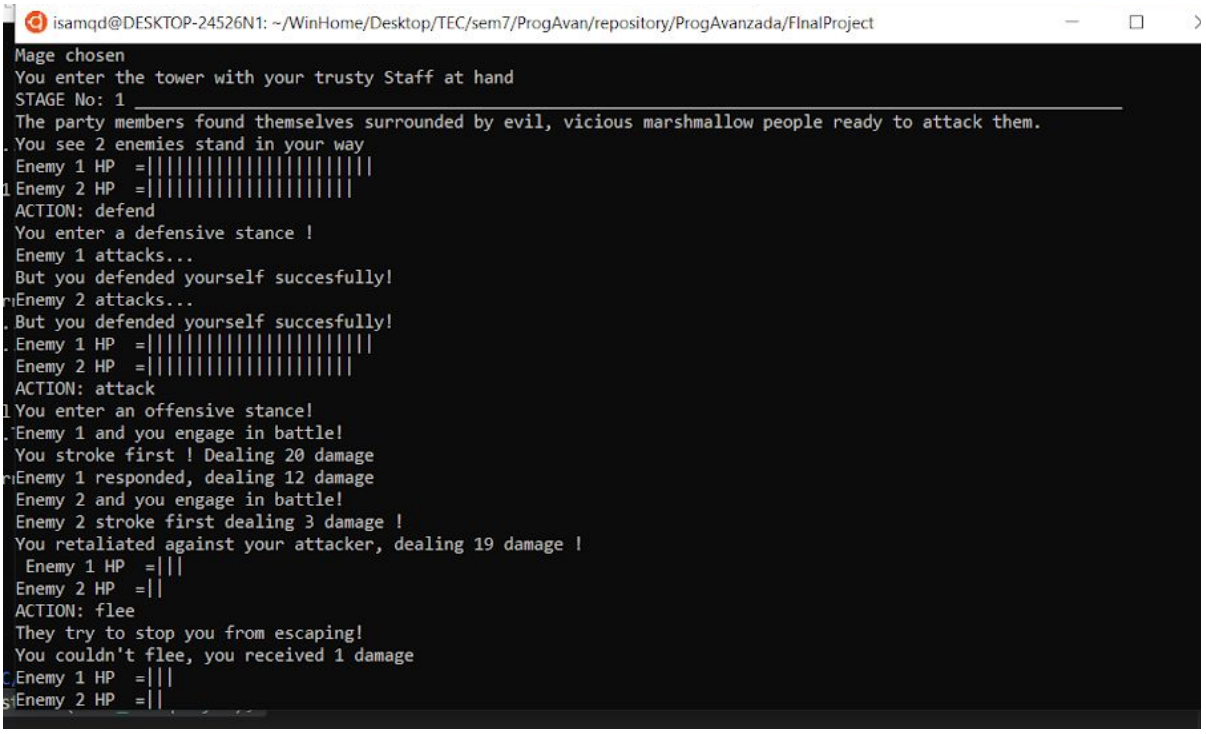
To play:
run the server
run the client

In the Client:

The program will let you choose to read the instructions or not, if you choose yes:

Then the program will let you choose a character, depending on which character you choose you have different weapons, damage, speed and health.





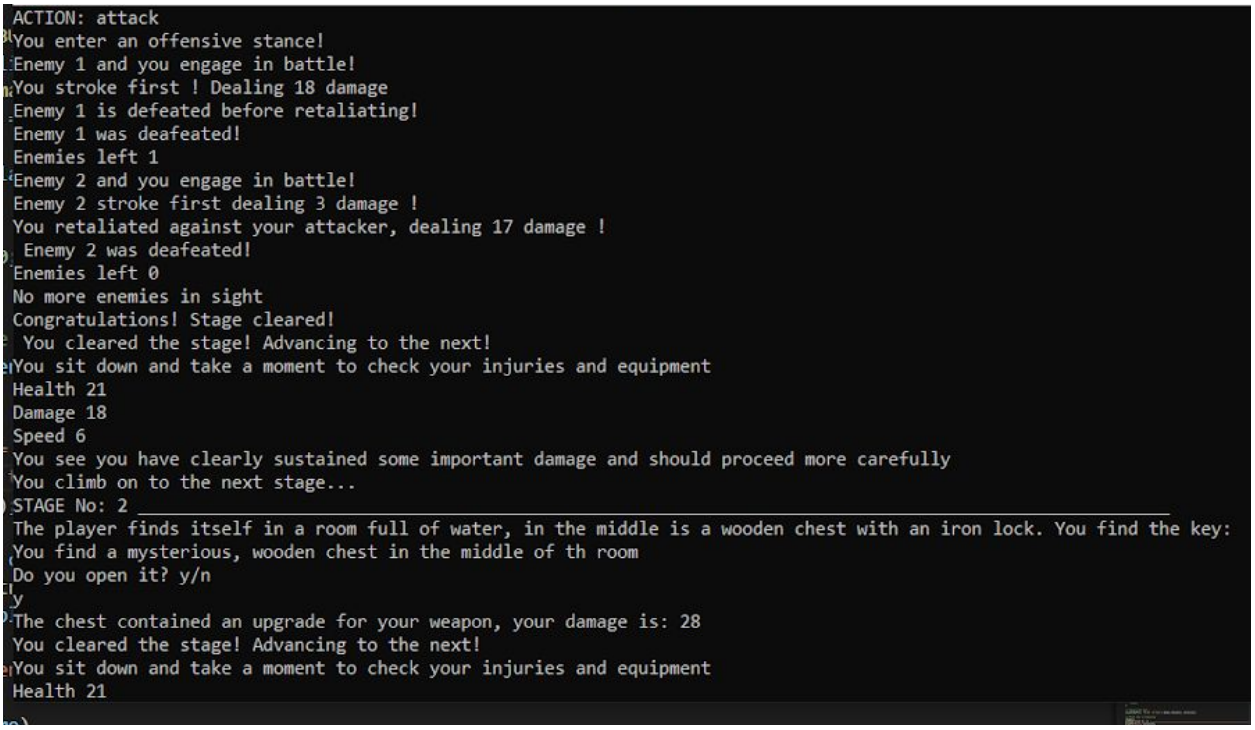
After you choose your character, the program will receive a stage by the server. Depending on the type you have to defeat enemies or open a chest

If you have enemies, you have three options, ATTACK, DEFEND or FLEE. Each action has a consequence. If you flee and the enemy is faster than you, then you lose health points. If you flee and the enemy is slower than you, then you can escape.
If you choose to defend, and you have a greater damage than your enemy, the enemy loses points in health but if the enemy has a greater damage, then you lose health points. Also, every time you attack or defend you lose a damage points. If you choose to attack, depending on your speed and the speed of the enemy, the one with greater speed attacks first. If the one attacked loses all health points, the battle with the enemy does not continue. If not, the enemy retaliates and deals damage.

This repeats until all enemies are defeated or until the player dies.

If the type has you opening a chest, the chest generates random content that can heal you, give you a new weapon, augment your damage or the opposite.

You keep playing until you die. Each stage is different because the enemies can change it´s stats and the chest contents are random.


ACTION: attack
You enter an offensive stance!
Enemy 1 and you engage in battle!
You stroke first ! Dealing 18 damage
Enemy 1 is defeated before retaliating!
Enemy 1 was deafeated!
Enemies left 1
Enemy 2 and you engage in battle!
Enemy 2 stroke first dealing 3 damage !
You retaliated against your attacker, dealing 17 damage !
Enemy 2 was deafeated!
Enemies left 0
No more enemies in sight
Congratulations! Stage cleared!
You cleared the stage! Advancing to the next!
You sit down and take a moment to check your injuries and equipment
Health 21
Damage 18
Speed 6
You see you have clearly sustained some important damage and should proceed more carefully
You climb on to the next stage...
STAGE No: 2
The player finds itself in a room full of water, in the middle is a wooden chest with an iron lock. You find the key:
You find a mysterious, wooden chest in the middle of th room
Do you open it? y/n
y
The chest contained an upgrade for your weapon, your damage is: 28
You cleared the stage! Advancing to the next!
You sit down and take a moment to check your injuries and equipment
Health 21