

EE6411/ED5021 Programming Project

1. Overview

This programming project is to be undertaken in groups of three to five students – please add your group details to the module’s SULIS page (Wiki section). This page also provides a facility to look for additional group members or to look for a group. Any **queries** regarding the group project should be raised via the module’s **SULIS page Forum facility**.

Your task is to write a simple Fantasy Game played through a text interface. The aim of the game is to collect as much gold as possible by defeating enemy characters. A (partially) implemented executable can be found on the module’s webpage to give you an idea of what is expected.

2. Rules of the Game

2.1 Environment:

The game is played on a rectangular board consisting of up to 200 by 200 squares (positions) – for both values the player can select the actual limits at the beginning of each game (if you wish to make it more general by omitting the upper limit of 200x200 you are welcome). Each square can either hold an item, an enemy or be empty. At the beginning of a game the board is initialised, i.e. each square is assigned either an enemy, an item or stays empty. After choosing a character the player begins the game at the starting square. The game starts on daytime. Daytime and Night-time alternate every 5 commands. The game is based on simple commands, entered via the keyboard. The following commands are available (use the first character for input):

- North, South, East, West – character moves to the square in the indicated direction (north=up, south=down, east=right, west=left)
- pick up – character attempts to add item on square to inventory (if the resulting total weight exceeds the character’s strength, this will fail)
- drop – drops one item (prompt user for the item to be dropped) to the current square (if the square already contains an item, request is ignored)
- attack – character attacks enemy on current square (if this results in the character’s health dropping to below zero, it is defeated)
- look – prints out information about the current location (information about any enemies or items present)
- inventory – prints out the list of items the player is currently carrying and the amount of gold that the player has earned.
- exit – ends the game

2.2 Characters:

Characters have the following properties:

- Attack (A) – indicates the attack-force of the character
- Attack Chance – indicates the probability of a successful attack, e.g. Hobbits have only a 1 in 3 probability of an successful attack.
- Defence (D) – indicates the defensive abilities of the character

- Defence Chance – indicates the probability to defend against a successful attack, e.g. Hobbits can defend against 2 out of 3 successful attacks.
- Health (H) – indicates the health status of a character. Once the health level has reached 0 the character is defeated.
- Strength (S) – indicates the weight a character can carry.

2.3 Race:

The race defines the abilities of a character. The following table shows the available races and their basic properties:

Race	Attack	Attack Chance	Defence	Defence Chance	Health	Strength
Human	30	2/3	20	1/2	60	100
Elf	40	1/1	10	1/4	40	70
Dwarf	30	2/3	20	2/3	50	130
Hobbit	25	1/3	20	2/3	70	85
Orc*	25(45)	1/4(1/1)	10(25)	1/4(1/2)	50	130

*Value is divided into day(night), i.e. Orcs are very good at night, but poor at day

2.4 Special Race Abilities:

- Human: Successful defences never cause damage.
- Elf: Successful defences always increases health by 1.
- Dwarf: Successful defences never cause damage.
- Hobbit: Successful defences cause 0-5 damage regardless of attack value.
- Orc: During day-time, successful defences cause 1/4 of adjusted damage (ie. 1/4 of attacker's attack – defender's defence). During night-time, successful defences cause increase of health by 1.

2.5 Items:

Each character can carry items up to a weight indicated by its strength. Items modify the basic abilities of characters. A character can carry only one item of each category, except rings, of which a player can carry as many as the character's strength allows. The following items are available (category is indicated in parentheses):

- Sword (Weapon), weight 10: increases attack by 10.
- Dagger (Weapon), weight 5: increases attack by 5.
- Plate Armour (Armour), weight 40: increases Defence by 10, decreases Attack by 5.
- Leather Armour (Armour), weight 20: increases Defence by 5.
- Large Shield (Shield), weight 30: increases Defence by 10, decreases Attack by 5.
- Small Shield (Shield), weight 10: increases Defence by 5.
- Ring of Life (Ring), weight 1: increases Health by 10.
- Ring of Strength (Ring), weight 1: increases Strength by 50, decreases Health by 10.

2.6 Combat Rules:

Enemies have the same characters/types as players. Whenever a player attacks an enemy, the enemy will immediately afterwards attack the player (unless the enemy was defeated on the last attack). For each attack the following rules apply:

1. The attacker resolve the attack success determined by the attack chance of the attacker's race. If the attack fails, terminate this round (if this is player's round move on to enemy, otherwise wait for the player's next command).
2. On successful attack, the defender resolves the defence success determined by the defence chance of the defender's race.
3. If the defence fails, the attack amount of the attacker (race + weapon) is adjusted by the defence value of the defender (race+weapon-defence) and then subtracted from the defender's health (defender new health = defender old health - (race+weapon-defence))
4. On a successful defence, the damage is determined by the character's race (see special race abilities).
5. If the health of the player reaches 0 (or below), the game is over. Your game should display the result (amount of gold collected) of the game.
6. If the health of an enemy reaches 0 (or below), the enemy is defeated: Remove the enemy from the board and add the enemy's defence value (adjusted by any carried items) in gold to the player's inventory.

Combat example: Hobbit Frodo (A20D20H70) is attacked during daytime by Orc Gargoyle (A25(50)D10(30)H50). Assuming Gargoyle succeeds with his attack (1/4 chance) and Frodo's defence succeeds (2/3 chance), then Frodo's health is reduced by random value 0-5 (race ability). If Gargoyle's attack fails, Frodo's health stays unchanged and if Frodo's defence fails his health is reduced by the attack value minus his defence (A25-D20 = 5).

Combat example 2: Hobbit Frodo (A20D20H70) attacks during daytime Orc Gargoyle (A25(50)D10(30)H50). Assuming Frodo's attack succeeds and Gargoyle's defence fails, then Gargoyle's health is reduced by 10 (A20-D10).

Above calculations assume that all characters are carrying no items. Any worn items will change the outcome of the attacks according to their modifications. If the health of any character drops to zero or below, that character is defeated. If the player defeats an enemy he will gain the enemy's defence value (adjusted by any items the enemy is carrying) in gold, i.e. if a player defeats a basic human enemy that player will gain 20 gold pieces.

Note: If you want to extend the list of characters and/or items, feel free to do so. However, above character/item lists are the minimum for a successful project.

3. Instructions

Please follow these instructions carefully:

- Use either a plain text editor (emacs or any other) or the Code::Blocks IDE.
- Your solutions **must** use object oriented programming (i.e. classes & inheritance, polymorphism is recommended, but not essential) – a solution in an imperative programming style (e.g. just main and a collection of functions, but no classes) will be **deemed a failed project (0 marks)**.
- Your solution needs to compile on an Ubuntu Linux machine (using g++). A solution that **does not compile will be deemed a failed project (0 marks)**.

- Design your classes and implement your game – make sure to comment your code, to format your code properly and to spread your code over suitable files.
- Create a Makefile to simplify compilation of your code (see comments on Makefile in Section 3 Submission & Marking).
- You can use static arrays for game board & list of items. However, feel free to explore use of STL classes for this project.
- You must **properly comment** your code (top of each file, before each function/method (including description of parameter & return value) and within your code) – as discussed in Unit1. **Insufficient comments will lead to significant loss of marks.**
- Your code **must also be properly formatted** (follow K&R coding style as discussed in the Unit1). **Insufficient poorly formatted code will lead to significant loss of marks.**
- There is no need to include a graphical user interface or any kind of artificial intelligence (the player is the only active character).
- Only standard C++ libraries are allowed – if you wish to use any other library you must first seek permission from me.

3. Submission & Marking

The project is worth 40% of the module. Deadline for the project is Friday, 18.12.2020 (week 12 – solutions will be accepted until Monday, 21.12.2020 without penalty). Solutions are to be submitted electronically (as a single tar archive) via the module's SULIS page. Only one group member should submit the group's solution. To create a tar archive, use the following command (please **do not** use compression as this sometimes causes problems on SULIS):

```
tar cvf yourTarfile.tar files
```

where should replace *yourTarfile.tar* with the name you choose for your tar file and *files* with the files/folders you want to include in the tar file. I strongly recommend to check afterwards that your tar file contains all files (copy it to another folder and extract it with `tar xvf yourTarfile.tar` and confirm that all files are present).

A complete solution consists of the following:

- Well documented and formatted source code for the game implementation using a suitable file structure.
- A Makefile written by yourself (do not use an IDE generated Makefile). Your Makefile must have the following features:
 - Provide separate targets for compilation and linking. Make sure to compile each source file individually with its own target.
 - Use at least one Makefile variable.
 - Running make without a target should execute the compile targets, the link target and also automatically start your game executable.
 - Provide a "clean" target that removes temporary files, object files and executable files.
 - Comment your Makefile.
 - **Not use** implicit rules.

The marking scheme for this project is as follows:

Description	Marks
Class Design	10
Game Implementation	20
Makefile	10
Penalties:	
Not using classes	-100%
Not using inheritance	-50%
Poor file structure	Up to -30%
Insufficient Comment	Up to -30%
Poor Code Format	Up to -30%
Bad Coding Style (e.g. global variables, goto statements, etc.)	Up to -50%
Compile time warning	-10% each
Compile time error	-100%
Total:	40