**Project 2**

Title

**Dungeons and Dragons**

**Fifth Edition Combat Simulator**

Course

CIS-5

Section

41595

Due Date

Feb 11, 2023

Author

Julian Eastman

Introduction 3

Development Summary 4

* Stage 7 4
* Stage 8 6
* Stage 9 8
* Stage 10 9
* Stage 11 10

Pseudocode and Flowcharts 11

Constructs and Concepts Used 18

Proof of a Working Project 22

References 27

Introduction

Overall project size: 2721 lines

Number of variables: over 30

Number of functions: 16

Combat in Dungeons and Dragons, unlike the rest of the game’s aspects, follows a very well-defined set of rules. At the very beginning of a campaign, players come into the game having rolled “ability scores” beforehand for their characters, these being the physical attributes of strength, constitution, and dexterity, as well as the mental attributes of wisdom, intelligence, and charisma. In combat and elsewhere throughout the game, these scores determine how good the players are at performing certain tasks, such as magic, or melee fighting. Players then choose one of many humanoid species, referred to as “races,” many of which apply additional ability score bonuses and features. Last, and most importantly, players choose a “class,” this being the specific skill set and expertise of their character. For instance, a Fighter may excel at archery and swordsmanship, but lack the knowledge necessary to cast spells, while a Wizard exemplifies the exact opposite of this. Due to the way each specific class works, for each one it is necessary to organize one’s stat rolls into a different arrangement for the best results. While a Fighter prioritizes strength in order to be better able to utilize martial weapons, a Wizard prioritizes intelligence in order to be able to learn and memorize spells.

When the players are attacked, or when they initiate an attack themselves, the very first thing to happen is the initiative roll. When initiative is rolled, each combatant adds their dexterity modifier to a roll of the d20, or 20-sided die. They then take their turns in descending order of their rolls, and repeat the order until the combat is over, usually resulting in the death of one or more combatants. On a turn, each player can perform any action that their class allows, but unless stated otherwise, they only get one action. Any monsters in the battle can make attacks or actions based on a preset list, referred to as a stat block, and they are controlled by the Dungeon Master, the person who runs the game. During the fight, the ability of each character to consistently perform well on attacks, or to dodge attacks directed at them, is more or less predetermined by the character’s six stats, as the ability modifiers which result from them are added to almost every roll.

In most cases, combat doesn’t happen in a vacuum. Instead, it takes place as part of a larger encounter or adventure. Maybe the players want to explore an ancient ruin, or infiltrate an enemy keep. Sometimes the script is flipped and they need to escape from some place. In the final version of my game, players must seek refuge from winter’s wrath by completing a short 3-room dungeon, in which they fight a zombie, collect loot from one of four chests which may or may not be monsters in disguise, and face off against a final boss.

Development Summary: Stage 7

In stage 7 of development, I was mainly focused on making the sorting algorithms for the score rolls more streamlined. As we did not have access to functions or arrays before this project, each roll had to be a separate variable, and the sorter was a while loop that had ifs in it to switch around each individual pair of numbers. The new chapters introduced arrays, and also introduced two new strategies for organizing them, those being the bubble sort and selection sort.



Text

Description automatically generated

As can be seen, a third function “sortRolls” is included here. I had originally made this function using my old method of sorting, but since I needed to include bubble and selection sorts, both of which are likely more efficient and most definitely more versatile, I ended up throwing my old design out the window pretty early on. On the next page is a picture of my implementation of bubble and selection sort. Disregard the second comment about strength, I fixed that to say “constitution” later on.

Graphical user interface, text, application

Description automatically generated

As can be observed, for each of these I create a random array of d6 dice rolls, sort it, and add the largest three to get the ability score.

Development Summary: Stage 8

In stage 8 of development, I added a more streamlined set of dice. In Project 1, as we could not use functions, each dice roll required me to write out a full expression to get a random number, mod it with the appropriate number, and add one. The lengthier rolls, especially that of Fireball, became difficult to read and tedious to edit. So, I created a set of dice using functions, and replaced the original dice-rolling code with calls to these functions. I also created a boolean “coin flip” function for randomly choosing between two options.

Graphical user interface, text, application

Description automatically generated

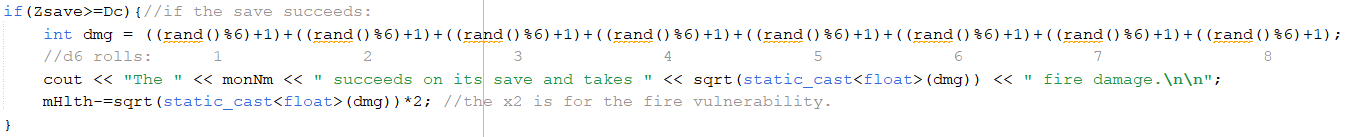
On lines 39 and 40, I overloaded the d20 function. This is due to the fact that in D&D, d20 rolls are often made with advantage or disadvantage, where you take two rolls and use the higher or lower of the two. The function on line 39 is a regular roll, while the one on line 40 accepts the character ‘a’ or ‘d’ to roll with advantage or disadvantage, respectively. Here is the code for each of these.

Graphical user interface, text, application

Description automatically generated

Shown below is a before/after shot of the code for Fireball’s damage roll, to illustrate how much simpler the dice functions make things.

Before:



After:

Text

Description automatically generated

Lastly, I included vectors in the project by making a selection sort function that works on a vector input. I then used this function in a stat roll in stage 9. This function has the added benefit of putting the rolls in descending order, and in version 8, I had also used a cout statement to test if it was working correctly. This statement has been removed in later iterations.

Graphical user interface, text

Description automatically generated with medium confidence

Development Summary: Stage 9

In stage 9 of development, I implemented the previously mentioned vector function to sort one of the stat rolls. I also designed a better way to sort the ability scores themselves once they are all rolled. Beforehand, I had used six individual variables “s1” through “s6” for the scores. But now that we can use arrays, I was able to make an array s[] of size 6. Since I used the array selection sorter to sort it, s[] ends up in ascending order, which is not ideal for my purposes. We want the highest scores first. So, I created a parallel array sd[], or “scores descending,” into which I put the values of s[], but backwards.

Graphical user interface, text, application, email

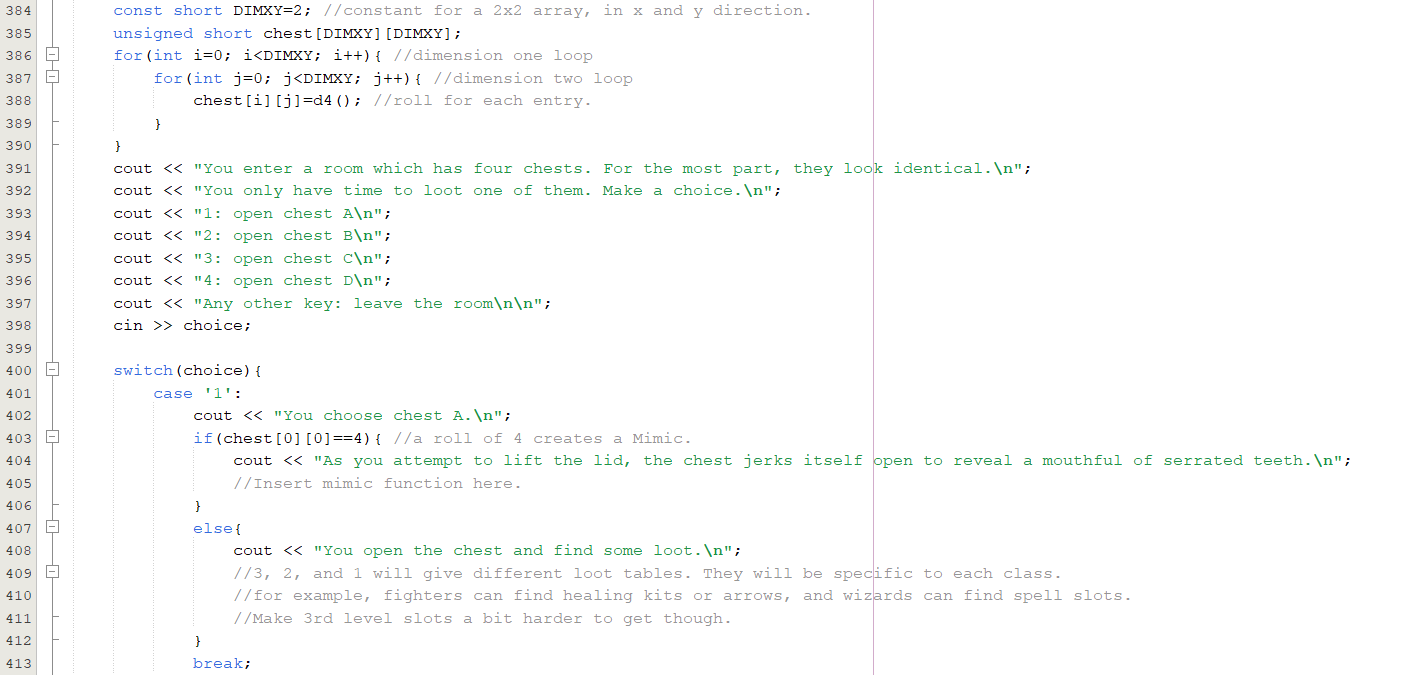
Description automatically generated

After getting the score sorter set up, I then decided to put the code for the zombie fight into a function. The function intakes everything necessary to run the combat code (about 18 variables), and it passes the player health and the “consumable” items by reference, so later sections of combat can know what items and how much health the player has left by then.

Development Summary: Stage 10

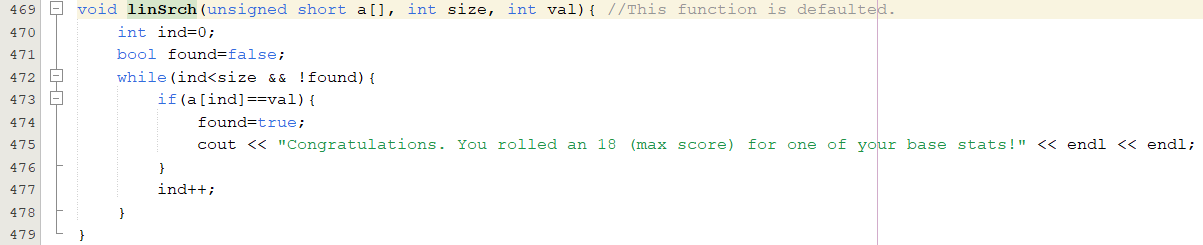
In stage 10 of development, I began work on the chest room. The room is intended to have four chests, with randomized loot tables. The loot includes magical crystals, to recharge Wizard spell slots, and also arrows, to prepare the Fighter for the boss fight, in which all combat will be at range and a sword will be useless. There is also a 25 percent chance for each chest that the chest will actually be a Mimic, a type of shapeshifting monster that disguises itself as loot or furniture and attacks if disturbed.



In the chest room, I utilized a 2-dimensional array for the loot table. As shown above, I randomized each coordinate with a d4 roll. Then I have the player pick a chest or choose to leave the room.

Afterwards, I also implemented a linear search that looks through your initial ability scores and congratulates you if you rolled an 18 (max score) on any of them. This function is also defaulted to look for the number 18 if not told what to look for.





Development Summary: Stage 11

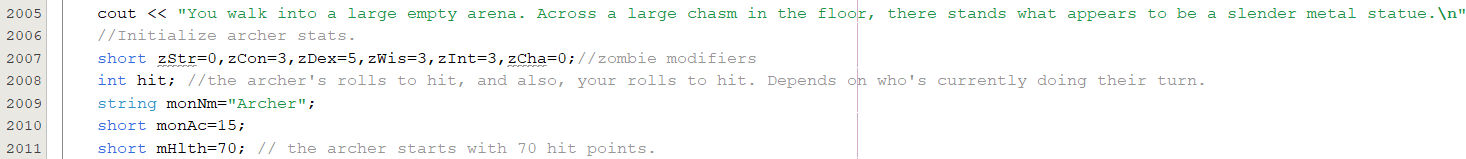
In stage 11 of development, I added the last two sections of combat, completing the project. In stage 9, I made the zombie fight into a function to make it easier to copy and modify. All I had to do for each separate monster was change the monster name and stats, change the methods of attack it can use, and apply any damage vulnerabilities or resistances into the player’s attack list. Since each monster is slightly different, I could not have them all use exactly the same code. So, this more than doubled the length of my program. I also created a function that decides what loot the chests give you, assuming they don’t just fight you instead.

Graphical user interface, text

Description automatically generated

Application

Description automatically generated with low confidence



In version one, it likely stuck out like a sore thumb that the Dwarf option, with its CON save advantage and poison resistance, was not at all suited for the zombie fight. However, the Archer uses poisoned arrows, and also uses bomb arrows that force a CON save, both of which allow the Dwarf to shine during the boss fight.

Pseudocode and Flowcharts

void function sSrtVec, accepts a vector of integers “a” by reference.

Declare integers minDex, minVal

For all but the last vector element, denoted by index “s”:

minDex=s

minVal=a[s]

For each element “i” that is one index ahead of element “s”:

if a[i] is less than minVal:

minVal=a[i]

minDex=i

Swap a[s] with a[minDex]

Declare integers strt=0, end=the size of vector a, minus one

While strt is less than end

Swap a[strt] with a[end]

Add one to strt

Subtract one from end

Diagram

Description automatically generated

void function bubSort, accepts unsigned short array a[] and int size

(arrays are always passed by reference by default)

For integer maxEl=size-1, maxEl is greater than zero, maxEl reduces by 1 each run:

For integer index=0, index is less than maxEl, index++

If a[index] is greater than the index immediately after it, swap the two of them.

Diagram

Description automatically generated

If player still lives:

declare constant short DIMXY=2;

declare unsigned short array: chest[DIMXY][DIMXY] (2 dimensional)

for the two indices “i”

for the two indices “j”

chest[i][j] equals a random d4 roll

say: "You enter a room which has four chests. For the most part, they look identical.\n";

"You only have time to loot one of them. Make a choice.\n";

"1: open chest A\n";

"2: open chest B\n";

"3: open chest C\n";

"4: open chest D\n";

"Any other key: leave the room\n\n";

Intake their choice, as a char.

switch(choice)

case '1':

say "You choose chest A.\n"

call function chestRm(chest[0][0],sp1,sp2,sp3,arrow), which generates the loot table.

if(chest[0][0]==4), call the Mimic function with all the player’s variables.

Break out of switch

case '2':

say "You choose chest B.\n"

call function chestRm(chest[0][1],sp1,sp2,sp3,arrow), which generates the loot table.

if(chest[0][1]==4), call the Mimic function with all the player’s variables.

Break out of switch

case '3':

say "You choose chest C.\n"

call function chestRm(chest[1][0],sp1,sp2,sp3,arrow), which generates the loot table.

if(chest[1][0]==4), call the Mimic function with all the player’s variables.

Break out of switch

case '4':

say "You choose chest D.\n";

call function chestRm(chest[1][1],sp1,sp2,sp3,arrow), which generates the loot table.

if(chest[1][1]==4), call the Mimic function with all the player’s variables.

Break out of switch

default:

say "You proceed down the corridor.\n";

Flowchart on next page:

Diagram

Description automatically generated Diagram

Description automatically generated

Void function linSrch(unsigned short a[], int size, int val)

Variable “val” is defaulted to 18, the highest score possible.

Declare integer ind=0;

Declare Boolean found=false;

While ind is less than size and found = false

If a[ind] equals val

found=true;

say "Congratulations. You rolled an 18 (max score) for one of your base stats!" << endl << endl;

Add one to ind

Diagram

Description automatically generated

Constructs and Concepts Used

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Chapter** | **Section** | **Topic** | **Where Line #''s** | **Pts** | **Notes** |
| 2 | 2 | cout | 82 |  |  |
|  | 3 | libraries | 16 | 5 | iostream, iomanip, cmath, cstdlib, fstream, string, ctime |
|  | 4 | variables/literals | 75 |  | No variables in global area, failed project! |
|  | 5 | Identifiers | 64 |  |  |
|  | 6 | Integers | 65 | 1 |  |
|  | 7 | Characters | 72 | 1 |  |
|  | 8 | Strings | 74 | 1 |  |
|  | 9 | Floats No Doubles | 64 | 1 | Using doubles will fail the project, floats OK! |
|  | 10 | Bools | 1394 | 1 |  |
|  | 11 | Sizeof \*\*\*\*\* |  |  |  |
|  | 12 | Variables 7 characters or less | TRUE |  | All variables <= 7 characters |
|  | 13 | Scope \*\*\*\*\* No Global Variables | TRUE |  |  |
|  | 14 | Arithmetic operators | 1406 |  |  |
|  | 15 | Comments 20%+ | TRUE | 2 | Model as pseudo code |
|  | 16 | Named Constants | 387 |  | All Local, only Conversions/Physics/Math in Global area |
|  | 17 | Programming Style \*\*\*\*\* Emulate | TRUE |  | Emulate style in book/in class repositiory |
|  |  |  |  |  |  |
| 3 | 1 | cin | 401 |  |  |
|  | 2 | Math Expression | 1796 |  |  |
|  | 3 | Mixing data types \*\*\*\* |  |  |  |
|  | 4 | Overflow/Underflow \*\*\*\* | FALSE |  |  |
|  | 5 | Type Casting | 1796 | 1 |  |
|  | 6 | Multiple assignment \*\*\*\*\* | FALSE |  |  |
|  | 7 | Formatting output | 355 | 1 |  |
|  | 8 | Strings | 174 | 1 |  |
|  | 9 | Math Library | 1796 | 1 | All libraries included have to be used |
|  | 10 | Hand tracing \*\*\*\*\*\* | FALSE |  |  |
|  |  |  |  |  |  |
| 4 | 1 | Relational Operators | 1793 |  |  |
|  | 2 | if | 2667 | 1 | Independent if |
|  | 4 | If-else | 1764 | 1 |  |
|  | 5 | Nesting | 2670 | 1 |  |
|  | 6 | If-else-if | 2694 | 1 |  |
|  | 7 | Flags \*\*\*\*\* | FALSE |  |  |
|  | 8 | Logical operators | 2492 | 1 |  |
|  | 11 | Validating user input | 269 | 1 |  |
|  | 13 | Conditional Operator | 363 | 1 |  |
|  | 14 | Switch | 1524 | 1 |  |
|  |  |  |  |  |  |
| 5 | 1 | Increment/Decrement | 2525 | 1 |  |
|  | 2 | While | 480 | 1 |  |
|  | 5 | Do-while | 630 | 1 |  |
|  | 6 | For loop | 490 | 1 |  |
|  | 11 | Files input/output both | 447, 467 | 2 |  |
|  | 12 | No breaks in loops \*\*\*\*\*\* | TRUE |  | Failed Project if included |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| \*\*\*\*\*\* Not | required to | show | Total | 30 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Chapter** | **Section** | **Topic** | **Where Line #''s** | **Pts** | **Notes** |
| 6 |  | Functions |  |  |  |
|  | 3 | Function Prototypes | 37 | 4 | Always use prototypes |
|  | 5 | Pass by Value | 49 | 4 |  |
|  | 8 | return | 545 | 4 | A value from a function |
|  | 9 | returning boolean | 582 | 4 |  |
|  | 10 | Global Variables | not used | XXX | Do not use global variables -100 pts |
|  | 11 | static variables | 146 | 4 |  |
|  | 12 | defaulted arguments | 39 | 4 |  |
|  | 13 | pass by reference | 49 | 4 |  |
|  | 14 | overloading | 41 | 5 |  |
|  | 15 | exit() function | 460 | 4 |  |
| 7 |  | Arrays |  |  |  |
|  | 1 to 6 | Single Dimensioned Arrays | 90 | 3 |  |
|  | 7 | Parallel Arrays | 147 and 148 | 2 |  |
|  | 8 | Single Dimensioned as Function Arguments | 37 | 2 |  |
|  | 9 | 2 Dimensioned Arrays | 388 | 2 | Emulate style in book/in class repositiory |
|  | 12 | STL Vectors | 105 | 2 |  |
|  |  | Passing Arrays to and from Functions | 92 | 5 |  |
|  |  | Passing Vectors to and from Functions | 109 | 5 |  |
|  |  |  |  |  |  |
| 8 |  | Searching and Sorting Arrays |  |  |  |
|  | 3 | Bubble Sort | 532 | 4 |  |
|  | 3 | Selection Sort | 515 | 4 |  |
|  | 1 | Linear or Binary Search | 477 | 4 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| \*\*\*\*\*\* Not | required to | show | Total | 70 | Other 30 points from Proj 1 first sheet tab |

Proof of a Working Project

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text, email

Description automatically generated

References

**Class Features, Races (The races were tweaked a little here for better balance)**

<http://dnd5e.wikidot.com/>

**Frost Zombie stat block**

<https://www.dandwiki.com/wiki/Frost_Zombie_(5e_Creature)>

**Mimic stat block (Hit points reduced for balance)**

<https://www.dndbeyond.com/monsters/16957-mimic>

**The Archer (the final boss) is of my own design.**