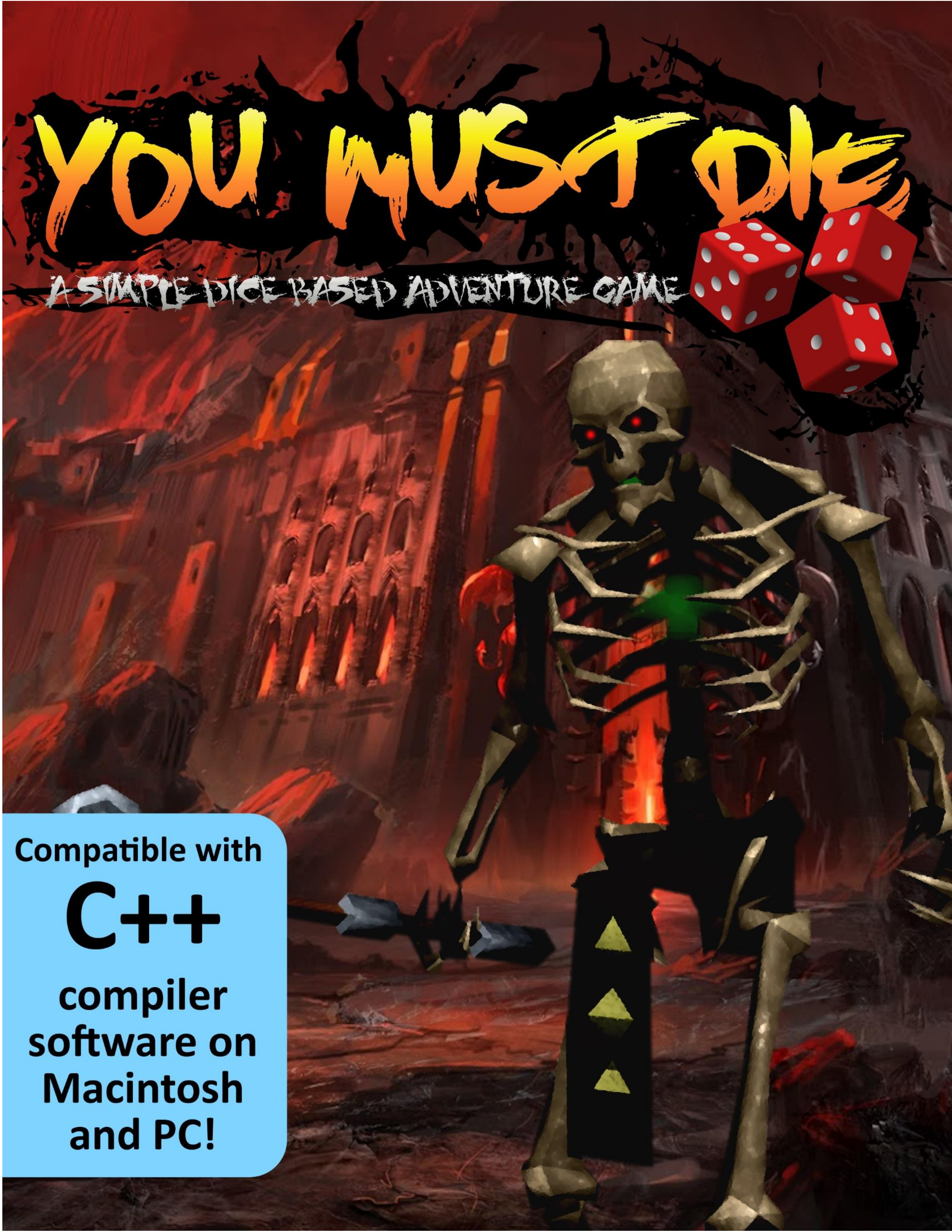
YOU MUST DIE

A SIMPLE DICE BASED ADVENTURE GAME



Compatible with

C++

compiler software on Macintosh and PC!

PROJECT 1

(You Must Die – A Simple Dice Based Adventure Game)

CIS – 5 47471

Yokley, Jacob

11/05/2017

The Story

Eons ago, there ruled one, mighty empire over multiple, puny colonies. This empire—once known as Grayshire—was known for its everlasting peace, equity, and self-sustainment—but only at the cost of all the colonies it ruled over. Weary of the constant infraction of human rights by the Empire, the colonies banded together an army of rags and bones to demolish Grayshire and build a truly unified kingdom of peace upon the rubble known as Sylvania.

As Sylvania celebrates its 100th annual Festival of Peace marking 100 years since the destruction of Grayshire, a massive structure has shot up into the middle of the town square resembling the Empire's former castle. Many warriors have entered, but none have returned.

It is now up to you, a young knight seeking an end to this injustice (and perhaps some loot along the way), to enter the dungeon and see an end to its army of undead. Choose how many rooms you want to traverse and roll a die in each one—will it earn you some hidden treasures, or a fight with Grayshire's ghostly warriors? Only the highest rolls and the quickest wits will earn you the victory you seek in these battles—so enter, young hero, and gamble 'til you drop.

Introduction

In You Must Die, all of the events that the player faces and how he or she responds to them is entirely based on the roll of a die. Players choose how many rooms they want to traverse in the dungeon, and then each room randomly selects if the player encounters an enemy, a treasure chest, or a trap. During these encounters, the player chooses what action they want to gamble on—whether the player attacks the enemy, defends against it, or chooses to open the treasure chest they stumbled across, the computer decides at random if these choices end up being beneficial to the player. The result is a simple adventure game that involves a basic amount of strategy to decide whether or not the odds are for or against the player.

Development

Project size: 427 lines

This project took about 18 hours to fully develop. Although I toiled with a few different ideas, it did not take long for me to settle on the concept for an adventure game. There was a lot of opportunity for passion to be put into this project because of my own love for adventure games and a lot of ways that the lessons we have covered in class could be utilized to make the gameplay more flavorful.

I began by jotting down some notes on what I wanted to see in the game and what it needed to achieve those goals. This was a good way for me to draw inspiration for the project and to cut my work out for myself before I actually began coding.

Once I was content with my ideas, I began working on version 1 of the game. This version was focused solely on creating what I believed would be the most complex and important part of the game: the battle sequence. This version only featured a title screen that immediately sent the user into a battle. I was hoping this would be the best way to start since I could iron out bugs from inside-out the program, but in retrospect, I think I might have had an easier time keeping track of everything if I built the "shell" of the game and then inserted all of the actual gameplay components where they needed to be. The main reason I believe this is because by the final versions, whenever I wanted to implement something ambititous outside of the projects requirements, I had a very difficult time finding all the issues to get rid of that would help me update the game.

This is a perfect segway to version 2, which implemented almost everything else I envisioned the game to be originally. I added a functioning menu that allowed the user to choose whether they wanted to play, read about the game, or quit. I also implemented all of the other gameplay features I wanted, such as scaling difficulty, treasure chests, traps, and custom dungeon size. I also took this time to fix some glaring bugs from the previous version.

Version 3 added everything else on the checklist that I missed with version 2 that I believed I could reasonably fit into the game. I added a score system using constants and allowed the player to name their hero. These both were implemented into the new file output system that created a file with the player's previous game results in a formatted table.

The development wasn't faced with any hugely frustrating struggles. There were some issues that I simply had to walk away from for a bit to finally think of the solution to. I was very pleased about this because I was worried about choosing a project that would be a bit overly ambitious, since I tend to do so in other classes and did not want to run that risk with my first computer science class. The result was a very rewarding development process that taught me a lot and provided a challenge without being a total pushover. I already have ideas for what I want to implement when updating this project for project 2, and some of these ideas can even be seen in the changelogs of each version of the game. I included these so I could list the issues that I wanted to fix and to show the issues I have overcome, and all I can really say in retrospect is that I have a lot to learn, but I'm excited to learn it so that I can make a more polished product.

Simple Adventure Game

- Dice based battle game
- Player skills increase with each kill (# of dice increases to increase odds of a kill)
- Player can set number of rooms
- Random enemy encounters
- Random treasure encounters

~~[illegible]~~

- On victory/death, character stats saved to a file
- if on final ~~[illegible]~~ room, boss battle starts(?)
    (How would you scale this for lower difficulties?)
- ASCII Character art?
- (Character classes?)
- Potions with different effects
    - while potion A = true, etc etc etc

Needs

- Random numbers
- if statements

Room traversing — traps? Items? Enemies?

- Menu
- switch cases

~~while/for loops~~

- file ~~output~~ input/output
- Boolean statements
- void functions

```
switch(choice){
    case 1: {

        break;
    }
    default: {
    }
```

Changelogs

Version 1

List of known bugs:

- Does not display "You have been slain!" when player health drops to 0 or below 0

- I'm not yet sure how to cap the player's health to a maximum so that the maximum can be upgraded in game, but can't be crossed with healing during battle

- Default doesn't work properly (entering a number forces you to enter a correct one, just doesn't display the error message. Entering a character starts an infinite loop)

- If rolls are equal in defense, heal, and run, the game has no idea what to do and just awkwardly restarts the loop without saying anything

- Successfully running away doesn't actually do anything

Version 2

List of known issues:

- Having a hard time getting a special "final room sequence" begin. Setting conditional statements just causes the game to return to the title screen. Must be an issue with the for loop? Might have to implement in a future version

- Had a problem with cin.get() for waiting for the user to hit enter to continue, so I added cin.ignore() to flush the entry, but now the program sometimes requires the user to hit enter twice, which is a bit annoying but nothing game breaking

- Had a problem implementing voids to help clean up the main chunk of code behind the game. I wanted to have the battle sequence, title sequence, and room sequence be different bools that would run whenever they are true and then call to the code behind each sequence when those bools are true. The game works as intended without it, but it lacks polish

- General balancing could be better

Changes from V1:

- Added a working title menu with an option to start the game, read about the game and its story, and to quit

- Added the option to choose how many rooms the user wants to traverse

- Added an increase in enemy health with every fight to add to difficulty

- Lowered player base health for balancing and to give cmath a fair purpose when healing the player

- Added booby traps

Version 3

List of known issues:

- Room score is terribly incorrect and causes an overflow of some kind. I really have no idea what causes this and I've done everything I can to try to mitigate this, so I'm just going to very slyly interpret this as an example of an overflow since I was able to identify it properly and since my file output at least works properly

Changes from V2:

- Added a working file system for outputting the score of the players last game

- Added the ability to name your hero

- Added the use of iomanip for formatting the score table

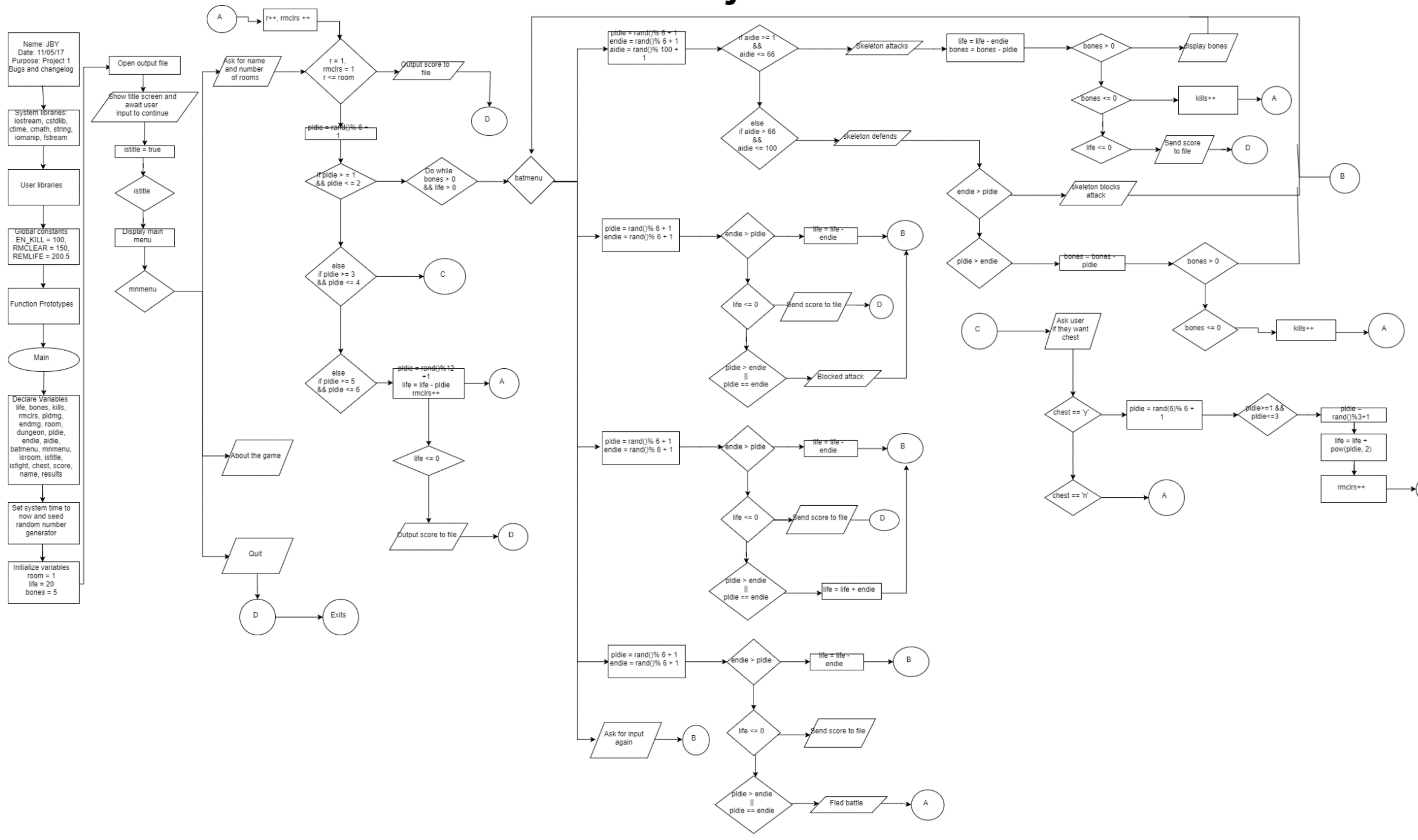
- Added counters and variables for keeping track of kills, rooms, score, etc.

Cross Reference for Project 1

Chapter	Section	Topic	Where in Code Line number (based on first appearance)
2	2	cout	78
	3	libraries	iostream, iomanip, cmath, cstdlib, fstream, string, ctime
	4	variables/literals	45
	5	Identifiers	68
	6	Integers	45
	7	Characters	56
	8	Strings	58
	9	Floats No Doubles	57
	10	Bools	55
	11	Sizeof *****	
	12	Variables 7 characters or less	All variables at 7 characters max
	13	Scope ***** No Global Variables	No global variables
	14	Arithmetic operators	125
	15	Comments 20%+	1
	16	Named Constants	36
	17	Programming Style ***** Emulate	
3	1	cin	89
	2	Math Expression	329
	3	Mixing data types ****	
	4	Overflow/Underflow ****	File output (results.txt) shows an overflow in rmclrs variable
	5	Type Casting	
	6	Multiple assignment *****	Some variables are categorized together, but not assigned together
	7	Formatting output	176
	8	Strings	118
	9	Math Library	26
	10	Hand tracing ******	

4	1	Relational Operators	124
	2	if	126
	4	If-else	334
	5	Nesting	160
	6	If-else-if	170
	7	Flags *****	
	8	Logical operators	160
	11	Validating user input	304
	13	Conditional Operator	
	14	Switch	111
5	1	Increment/Decrement	124
	2	While	95
	5	Do-while	140
	6	For loop	124
	11	Files input/output both	Output only - 177
	12	No breaks in loops ******	
****** Not required to show			

Flowchart for Project 1


Name: JBY
Date: 11/05/17
Purpose: Project 1
Bugs and changelog
System libraries: iostream, cstdlib, ctime, cmath, string, iomanip, fstream
User libraries
Global constants EN_KILL = 100, RMCLEAR = 150, REMLIFE = 200.5
Function Prototypes
Main
Declare Variables life, bones, kills, rmclrs, pldmg, endmg, room, dungeon, pldie, endie, aidie, batmenu, mnmenu, isroom, istitle, isfight, chest, score, name, results
Set system time to now and seed random number generator
Initialize variables room = 1 life = 20 bones = 5
Open output file
Show title screen and await user input to continue
istitle = true
istitle
Display main menu
mnmenu
About the game
Quit
D
Exits
A
r++, rmclrs ++
Ask for name and number of rooms
r = 1, rmclrs = 1 r <= room
Output score to file
D
pldie = rand()% 6 + 1
if pldie > = 1 && pldie < = 2
Do while bones > 0 && life > 0
batmenu
else if pldie >= 3 && pldie <= 4
C
else if pldie >= 5 && pldie <= 6
pldie = rand()%12 +1 life = life - pldie rmclrs++
A
life <= 0
Output score to file
D
pldie = rand()% 6 + 1 endie = rand()% 6 + 1 aidie = rand()% 100 + 1
if aidie >= 1 && aidie <= 66
Skeleton attacks
life = life - endie bones = bones - pldie
bones > 0
display bones
bones <= 0
kills++
A
life <= 0
Send score to file
D
else if aidie > 66 && aidie <= 100
skeleton defends
endie > pldie
skeleton blocks attack
B
pldie > endie
bones = bones - pldie
bones > 0
bones <= 0
kills++
A
pldie = rand()% 6 + 1 endie = rand()% 6 + 1
endie > pldie
life = life - endie
B
life <= 0
Send score to file
D
pldie > endie || pldie == endie
Blocked attack
pldie = rand()% 6 + 1 endie = rand()% 6 + 1
endie > pldie
life = life - endie
B
life <= 0
Send score to file
D
pldie > endie || pldie == endie
life = life + endie
pldie = rand()% 6 + 1 endie = rand()% 6 + 1
endie > pldie
life = life - endie
B
life <= 0
Send score to file
pldie > endie || pldie == endie
Fled battle
A
Ask for input again
B
C
Ask user if they want chest
chest == 'y'
pldie = rand(6)% 6 + 1
pldie>=1 && pldie<=3
pldie = rand()%3+1
life = life + pow(pldie, 2)
rmclrs++
chest == 'n'
A

Pseudo Code

```
/*
* File:           main.cpp
* Author:         Jacob Yokley
* Created on November 3, 2017, 2:45 PM
* Purpose: Project 1: “You Must Die – A Simple Die-Based Adventure Game” V3
*/

//System libraries
//User libraries
//Global Constants – Used as score multipliers: enemies killed, rooms cleared, life
//remaining
//Function prototypes
//Main - Execution begins here!

    //Variable declaration

    //Set system time to now

    //Seed random number generator

    //Variable initialization

    //Open output file

    //Process Mapping – Inputs to Outpus

    //Title Screen with lovely ascii art

    //Main menu

    //Loop switch case for menu

        //Case 1 starts game

            //Ask for users name and number of rooms

            //For loop ends once max number of rooms is reached

                //Random number generator for dice

                    //if dice is 1 or 2 a battle starts
```

```
//do a switch for the battle menu while
//player life and enemy life are greater than
//0

//case 1 is attack

        //Both player and enemy roll a new
        //number on 6 sided die

        //Simple AI picks from a random
        //number from 1- 100 to emulate
        //percentage

        //If AIDie lands on anything from 1-
        //66 enemy attacks

        //Rolls determine damage

        //Killing enemy increases kill count

        //If you are killed, game ends and
        score is sent to file

        //Exits switch if enemy dies

        //AI rolls from 67 – 100 for defense

        //enemy die must be greater than
        //player’s to defend

//case 2 is defend

        //Player die must be greater than
        //enemy’s to defend

//case 3 is heal

        //Player die must be greater than
        //enemy’s to heal

        //Absorbs enemy’s roll as health if
        //successful

//case 4 is run
```

```
                            //Player die must be greater than
                            //enemy’s to run
                        //After battle ends, skeleton health is reset and
                        //increases by number of rooms traversed * 2
                    //if player die is 3 or 4
                        //Finds an item in the room
                            //if player chooses to open chest
                                //if player die rolls a 1 – 3
                                    //die rolls from 1-3
                                    //heals by die roll^2
                                    //Uses cmath
                                //else nothing is found
                            //chooses not to open chest moves
                            //onward
                    //if player die is 5 or 6
                        //Encounters booby trap
                            //player die rolls from 1 to 12 and
                            //takes as much damage as the amount
                            //rolled
        //When the game ends, score is sent to file
    //Case 2 for main menu
        //About the game
    //Case 3 for main menu
        //Quits the game
//Closes score file at end of program
```

```
//Exit function main, end of program
```