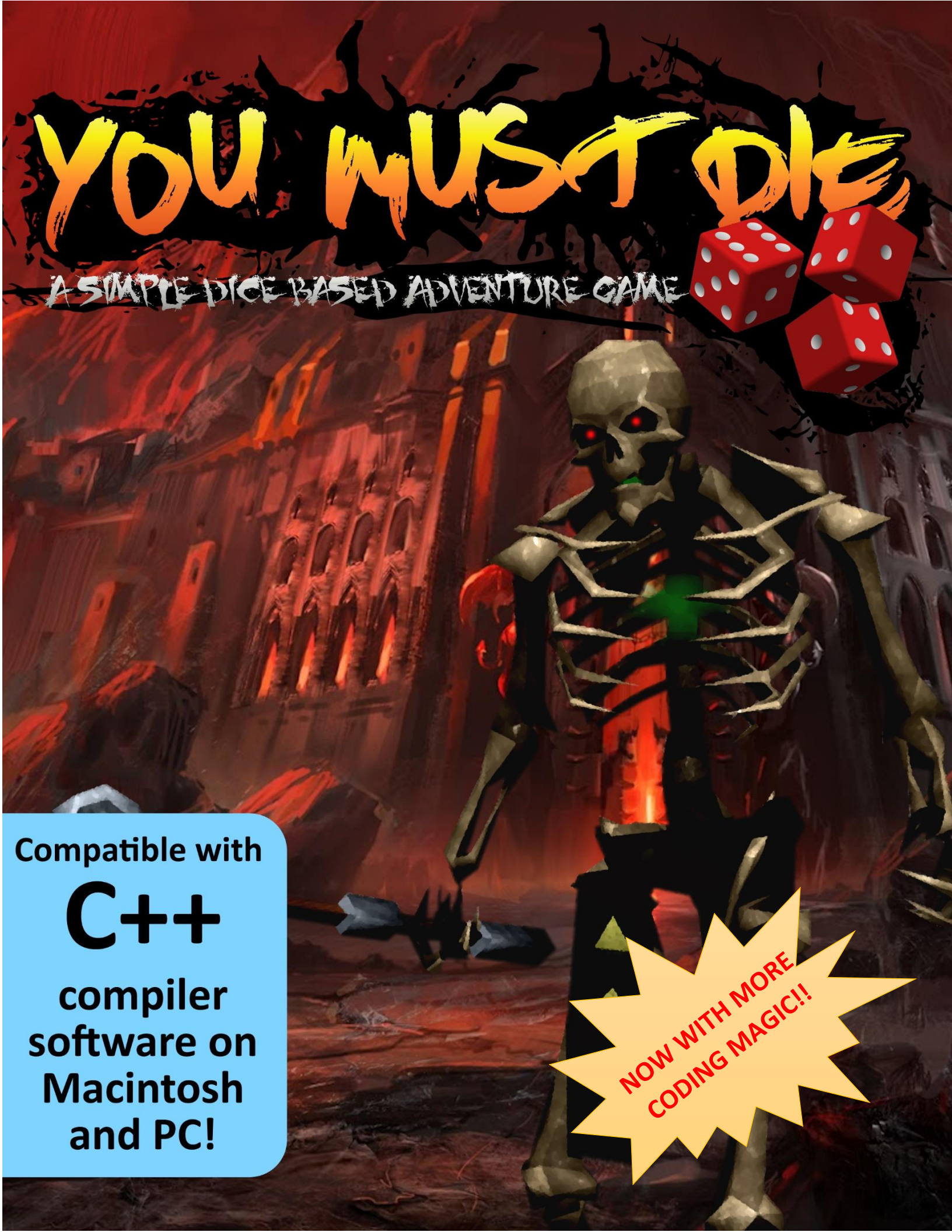
YOU MUST DIE

A SIMPLE DICE BASED ADVENTURE GAME



Compatible with

C++

compiler software on Macintosh and PC!

NOW WITH MORE CODING MAGIC!!

PROJECT 2

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

CIS – 5 47471

Yokley, Jacob

12/13/2017

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.

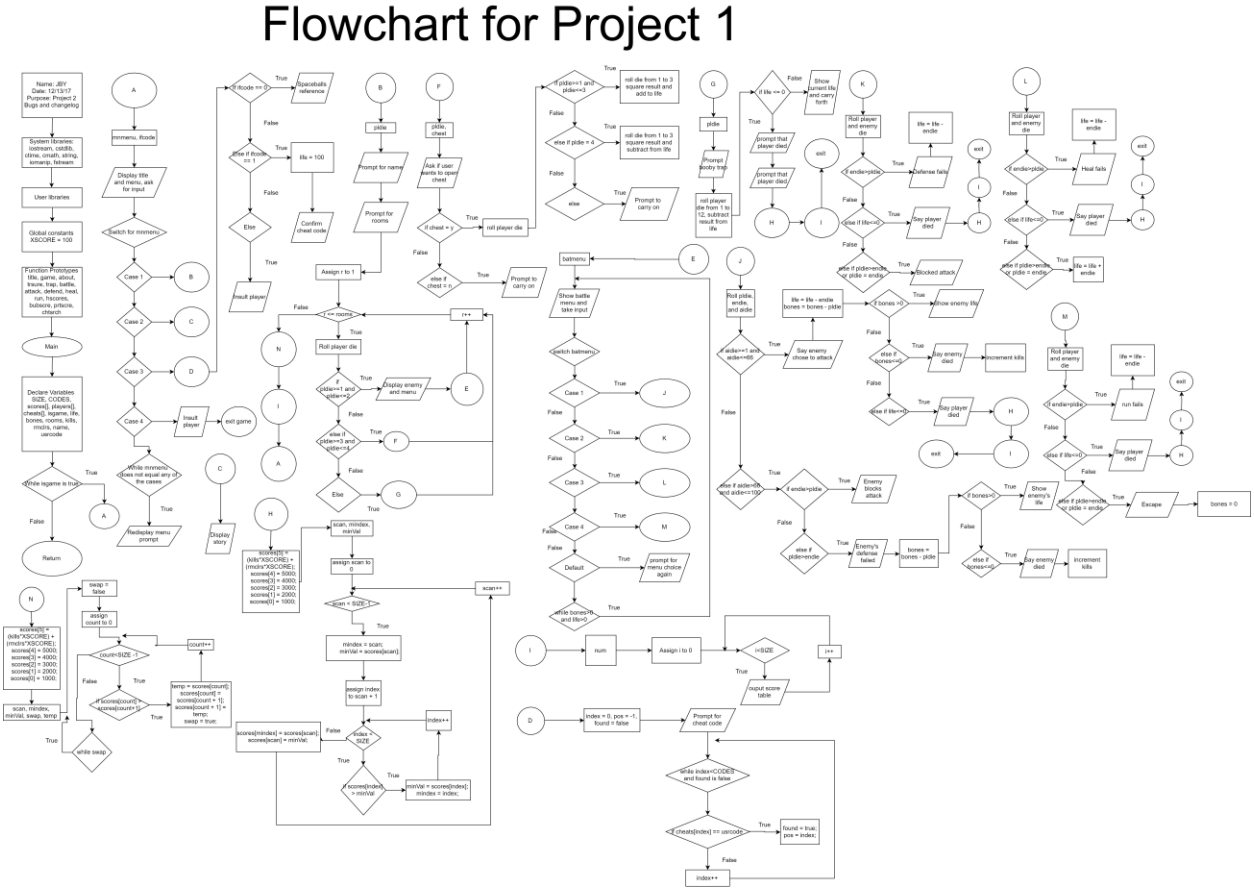
It is now up to you, a young knight seeking an end to this injustice (and perhaps some loot along the way), to enter the dungeon and see an end to its army of undead. Choose how many rooms you want to traverse and roll a die in each one—will it earn you some hidden treasures, or a fight with Grayshire's 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: 682 lines



Here it is: one of my proudest creations so far. This program is a culmination of all the things I've learned—and what I've yet to learn—in my first ever programming course. This wasn't the worst thing in the world to program, but it certainly had its challenges. Though I'm happy with what I have so far since it meets the criteria of how I wanted the game to function and because I'm proud of how far I've come, I'm also a bit puzzled about some features we learned that I couldn't get functioning properly in the code.

I began trying to simply modify the original code from version 3 with everything new that we've learned, but that proved to be a total disaster. I'd fix one problem and another would pop up with an error that Google could not help me understand. After exhausting every possible solution I could think of, I finally decided the best solution would be to start from scratch. I rewrote version 4 completely from the ground up, restructuring the entire game with functions in a manner that reminds

me of Russian nesting dolls. The more specific the action in the game, the deeper the function is nested within other functions.

Once I got this version working, I began adding arrays, sorting, and searching. I had some problems getting string arrays and string sorting to work properly with selection sorts, so I had to scrap the concept of having names in the high score table completely. I didn't have any ideas for implementing searching, so I made an array for number based cheat codes that the player can enter to increase the character's health or find easter eggs. Once I was satisfied with everything I could get working, I added some extra polish with balance and design tweaks.

This was definitely a harder project than the first one, but that's what made it more rewarding. Figuring out how to clean up my code and make the game more like how I originally envisioned it felt hugely rewarding and satisfying, and the sense of accomplishment I get from overcoming these hurdles is how I know I enjoy this field.

Simple Adventure Game

- Dice based battle game
- Player skills increase with each level (# of dice increases to increase odds of a kill)
- Player can set number of rooms
- Random enemy encounters
- Random treasure encounters
- ~~Shops~~
- On victory/death, character stats saved to a file
- if on final ~~Boss~~ room, boss battle starts(?)
  (How would you scale this for lower difficulties?)
- ASCII Character art?
- Character classes?
- Potions with different effects
  - white potion A = true, etc etc etc

Needs

- Random numbers
- if statements
- Menu
- switch cases
- ~~while/for loops~~
- file ~~output~~ input/output
- Boolean statements
- void functions

Room traversing — traps? Items? Enemies?

```
Switch(choice) {
    case 1: {
        ~~~~~~~~~~~~~~~~~~~~
        ~~~~~~~~~~~~~~~~~~~~
        break;
    }
    default: {
        ~~~~~~~~~~~~~~~~~~~~
    }
```

Changelogs

Version 4

List of known issues:

- Life falling below 0 doesn't seem to work properly
- Needs more comments (deleted a ton as a result of prioritizing getting the code to work first)

Changes from V3:

- Rewrote the entire code (excluding some reused code) to restructure the program in terms of functions
- Removed unnecessary variables
- MUCH cleaner looking

Version 5

Changes from V4:

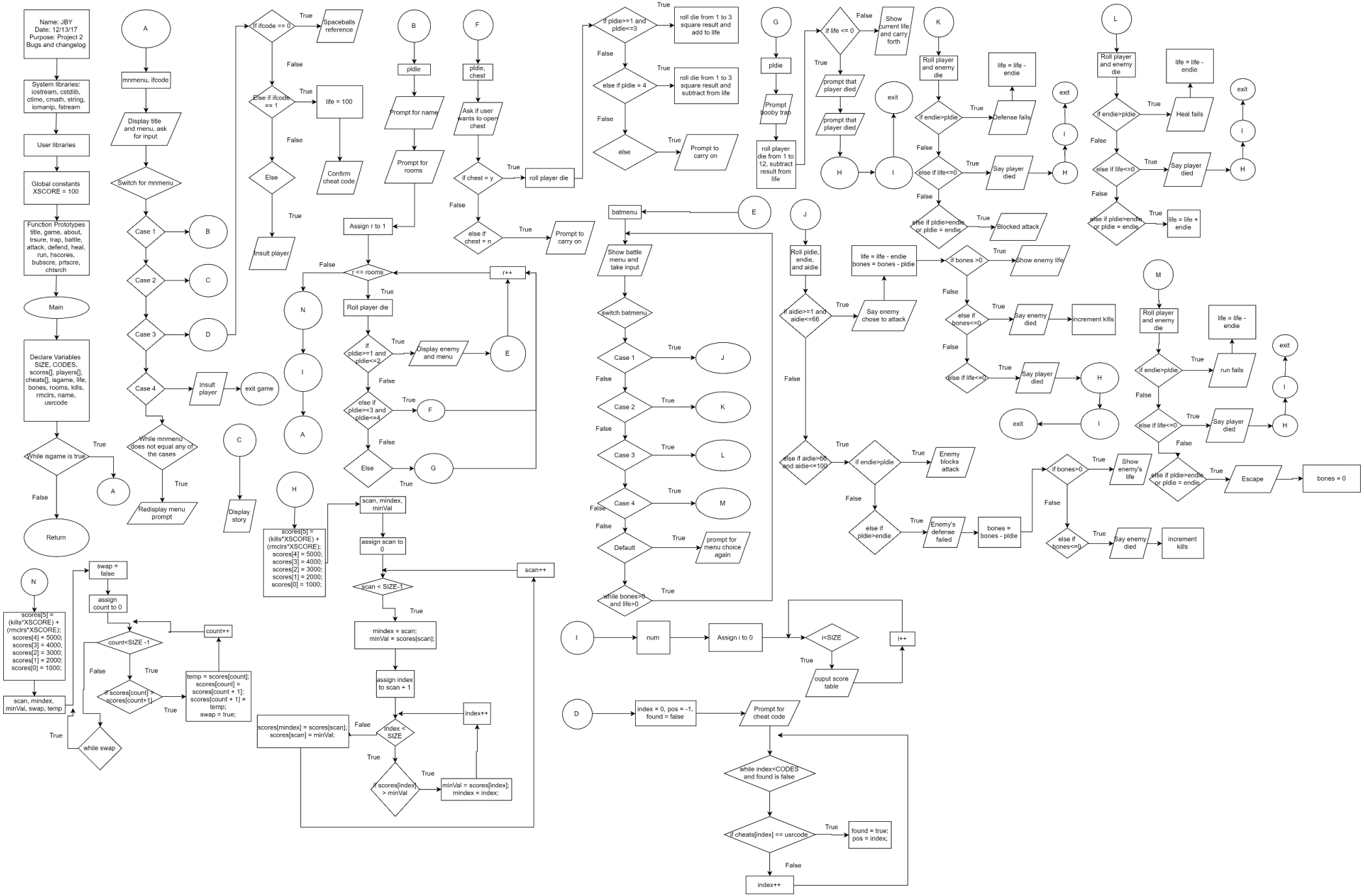
- Re-implemented a scoring system using arrays and sorting. Some parts of the game implement a bubble sort and others use a selection sort
- Upped player life to make game more balanced
- Added cheat codes with an implemented search function
- Added traps in treasure chests
- Added comments

Cross Reference for Project 2

			Where in Code
Chapter	**Section**	**Topic**	**Line number of first appearance in code**
2	2	cout	96
	3	libraries	18 iostream, iomanip, cmath, cstdlib, fstream, string, ctime
	4	variables/literals	53
	5	Identifiers	59
	6	Integers	59
	7	Characters	249
	8	Strings	64
	9	Floats No Doubles	None
	10	Bools	58
	11	Sizeof *****	
	12	Variables 7 characters or less	Yes - main variables end at 65
	13	Scope ***** No Global Variables	Yes
	14	Arithmetic operators	y
	15	Comments 20%+	Yes
	16	Named Constants	y
	17	Programming Style ***** Emulate	Yes?
3	1	cin	110
	2	Math Expression	223
	3	Mixing data types ****	
	4	Overflow/Underflow ****	
	5	Type Casting	None
	6	Multiple assignment *****	
	7	Formatting output	650
	8	Strings	64
	9	Math Library	21
	10	Hand tracing ******	
4	1	Relational Operators	140
	2	if	204
	4	If-else	219
	5	Nesting	254
	6	If-else-if	216
	7	Flags *****	670
	8	Logical operators	140
	11	Validating user input	140
	13	Conditional Operator	None

	14	Switch	111
5	1	Increment/Decrement	201
	2	While	70
	5	Do-while	337
	6	For loop	201
	11	Files input/output both	None
	12	No breaks in loops ******	Yes
6	3	Function Prototypes	33
	5	Passing by value	33
	8	Returning values from functions	681
	9	Returning a boolean ******	*None*
	10	No Global Variables Allowed	No global variables present
		Only Global Constants	
		Meaning Conversions,Physical Constants only	
	11	Static Local	193
	12	Default arguments	None
	13	Reference Parameters	90
	14	Overloading functions	None
	15	Exit function *******	137
7	4	Array Initialization	55
	6	Processing Arrays	576
	7	Parallel Arrays	None
	8	Arrays as function arguments	91
	9	2-D Arrays	None
	12	STL Vector	None
8	1	Linear and Binary Search	Linear - 667
	3	Bubble and Selection Sort	Bubble - 610 Selection - 574
	5	Search/Sorting Vectors ******	
****** Not required to show			

Flowchart for Project 2



Pseudo Code

```
//System libraries
//User libraries
//Global Constants – Used as score multipliers: enemies killed, rooms cleared, life
//remaining
//Function prototypes
//Main - Execution begins here!
        //Variable declaration
        //Process Mapping – Inputs to Outputs
                //While isgame is true
                        //Run title function
        //Exit function main, end of program
//Title Screen
        //Show title art and menu
        //Prompt user for menu input
        //Switch for menu
                //Case 1
                        //Run game function
                //Case 2
                        //Run about function
                //Case 3
                        //Run chtsrch function and assign value to ifcode
                        //If ifcode equals 0
                                //Display Spaceballs reference
                        //Else if ifcode equals 1
                                //Player life equals 100
```

```
                    //Display confirmation of player life
                //Else
                    //Prompt player to quit trying to be a cheater
            //Case 4
                //Quit game, exit
            //While main menu input does not equal one of the options
                //Prompt player to enter options again
//About function
    //Display story of game
//Game function
    //Ask for players name
    //Ask for number of rooms the player wants to traverse
    //Assign r to 1
    //For as long as r is less than or equal to the number of rooms
        //Roll player die
        //If player die is 1 or 2
            //Display enemy
            //Enter battle function
        //Else if player die is 3 or 4
            //Enter treasure function
        //Else
            //Enter trap function
        //Set enemy health back to 5 and then add 2 times the number of
        rooms passed as health
    //Enter bubble sort score function
    //Enter print score function
```

```
        //Enter title function
//Treasure function
        //Ask user if they want to open treasure chest
        //If user says yes
                //Roll player die
                //If player die is from 1 to 3
                        //Roll player die from 1 to 3, add that number squared to health
                        //Prompt user of health gain and ask to carry forth
                //Else if player die is 4
                        //Tell player it's a trap
                        //Roll player die from 1 to 3, subtract that number squared from
health
                        //Prompt user of health loss and ask to carry forth
                //Else
                        //Nothing is in chest, ask to carry forth
        //Else if user says no
                //Carry forth
//Trap function
        //Roll from 1 to 12
        //Subtract number from life
        //If life falls to 0
                //Player is killed, high score and print function run, game exits
        //Prompt current life
        //Carry forth
//Battle function
        //Do
```

```
            //Switch for battle menu
                //Case 1
                    //Enter attack function
                //Case 2
                    //Enter defend function
                //Case 3
                    //Enter heal function
                //Case 4
                    //Enter run function
                //Default
                    //Prompt user for valid input
        //While both player and enemy are alive
//Attack function
        //roll player die, enemy die, and AI die
        //If AI die is from 1 to 66
            //Enemy attacks and player takes the amount of damage enemy rolled
            //player attacks enemy using player die
            //If enemy is still alive
                //Display enemy health
            //Else if enemy is dead
            //Increase kills and display that enemy is dead
            //Else if player is dead
                //Display player is slain, high score and print score functions
                //Exit game
        //Else if AI die is from 67 to 100
```

```
        //Prompt that enemy chooses to defend
        //If enemy die is greater than player die
            //Enemy takes no damage
        //Else if player die is greater than enemy die
            //enemy takes damage
            //If enemy is still alive
                //Display enemy health
            //Else if enemy is dead
                //Display enemy is dead and increase kills
//Defend function
    //Roll player and enemy die
    //If enemy roll is greater than player die
        //Defense fails, player takes damage
    //Else if player dies
        //Displays player is dead, high score and print score functions, exit
    //Else if player roll is greater than enemy die or equal to it
        //Player takes no damage
//Heal function
    //Roll player and enemy die
    //If enemy roll is greater than player die
        //Heal fails, player takes damage
    //Else if player dies
        //Displays player is dead, high score and print score functions, exit
    //Else if player roll is greater than enemy die or equal to it
        //Player gains health
```

```
//Run function
    //Roll player and enemy die
    //If enemy roll is greater than player die
        //Escape fails, player takes damage
    //Else if player dies
        //Displays player is dead, high score and print score functions, exit
    //Else if player roll is greater than enemy die or equal to it
        //Player escapes, enemy health set to 0 to end game
//High score function
    //Set highest score in score array to number of kills and rooms times score multiplier
    //Assign scan to 0
    //For as long as scan is less than size of the score array minus 1
        //set mindex to scan and then minVal to scores[scan]
        //Assign index to scan + 1
        //For as long as index is less than size of the score array
            //if scores[index] is greater than minVal
            //Set minVal equal to scores[index] and mindex equal to index
        //Set scores[mindex] equal to scores[scan]
        //Set scores[scan] equal to minVal
//Bubble sort high score function
    //Set highest score in score array to number of kills and rooms times score multiplier
    //Do
        //Set swap equal to false
        //Assign count to 0
```

```
                    //For as long as count is than size of score array minus 1
                        //If scores[count] > scores[count + 1]
                        //Set temp equal to scores[count]
                        //Set scores[count] equal to scores[count + 1]
                        //Set scores[count + 1] equal to temp
        //While swap is true
//Print score function
        //Assign i to 0
        //For as long as i is less than the size of the scores array
            //assign num to 0
            //Print high score table and sorted score array
            //increment num by 1
//Cheat search function
        //Set index to 0
        //Set pos to -1
        //Set found to false
        //Ask player to enter a cheat code
        //While index is less than size of cheat code array size and found isn't true
            //if cheats[index] equals the user's code
                //found is true
                //pos equals index
            //increment index
        //return pos
```