Hangman Adventures: A Love Story in Beta

User Guide

Welcome to the land of the WordKeepers where just like love, one half is timing, and the other half is luck! Journey through the gates and guess the words of these tortured souls. Work is never done in the game, there will always be more Keepers to free.

You are shoved into the role of the hero where you must balance stats involving: health, strength, luck and charm to earn **10 coins** from these WordKeepers before the **30 day limit** is up!

You will experience hardships and love throughout your journey and it all relies on your wits and strategy to balance these 4 elements.

Skills

Health: This represents the max number of hearts you al allowed to have at any given time. The minimum it can be is 8

Strength: Although it is a game of wits, to get noticed, you're going to need to be strong. Maybe you'll even find love if this stat is high enough

Charm: Although wits are important, sometimes if your charm is high enough, you can just get what you want. The higher this stat, the higher the chance of you being able to bypass the word. This is especially helpful in situations where you are just utterly stumped.

Luck: It's what this whole game revolves around. The probability of you getting extra items, or getting an effective hint to guess a word is based on this trait.

Days

The game works on a day system, where every 4 encounters with a WordKeeper means you must sleep. You get a skill point every day, but don't get greedy, you must be wise in your assignment. Every 5 days, the theme of the WordKeepers words changes too!

- Up to Day 5: **Animals**
- Up to Day 10: **Instruments**
- Up to Day 15: **Countries**
- Up to Day 20: **TV Shows**
- Up to Day 25: **Singers**
- Up to Day 30: **Capital Cities**

Items

Health Pot, **Greater Health Pot** and **bread** all deal with replenishing your hearts. Each of them replenish 1 heart, with the exception of the Greater Health Pot which heals for 2.

The **Stat Reset Scroll** allows you to reassign your skill points, be strategic because they aren't as common as healing items.

Lucky Guess is an item which gives you a letter of the hidden word, essentially a hint. But be careful, because like most aspect of this game, you need a good amount of luck for the hint to be 100% effective.

Love Encounter

Love encounters occur spontaneously based on your stats, they could be every 4 days, or once in 15 days. It all depends on your style throughout the game, and really if your stats are high enough. Remember, a good start will most likely lead to better love encounters, first impressions really do last.

Endings

The endings of the game is based on your style of gameplay, whether you relied primarily on charm, or your wits were good enough to get through the game with pure guessing.

Probability and Percent Chances

The game is mainly based on luck and probability as well as certain encounters at fixed intervals. A lottery type system has been coded where the probability of success or an extra item, is based on your stats.

Hangman

Just like the classic hangman, this portion is played by guessing letters of a hidden word. You will not be penalized for repeated guesses, or accidentally entering a number.

Modifications from Original Plan

I originally wanted to the probability to yield a coin to be based on luck, however, since luck is based on skill points and a random number generator to be a pseudo lottery, the player could end up just never earning a coin, making it unfair. I changed that, and a lot of other aspects to be earned 'every 2 turns'. This was done by taking the number of turns, and seeing if the remainder is 0. If the remainder was 0, then that would mean it was indeed the 2nd turn, or whatever number turns was being divided by -> if (turns % 3 == 0), then the tern number is divisible by 3.

I also wanted a 'lucky beginning' which I altered to lucky guess, which essentiality just became a hint for the player. I did not know how to search through each word in the array and find the word with most instances of that letter. I tried (.contains), but it proved much too difficult for time constraints.

I also changed the skills, vitality became health, but it stuck with the same definition I gave it in my plan. Health is the max number of hearts allowed at any time. However, stamina was excluded and the player was never required to sleep for more than 1 day.

The entirety of the boss fight was excluded as adding attacks was much too farfetched for a bang man based game. The love aspect of the game was added where lover encounters essentially just gives the player free items and benefits. This is to say, the more aware the player is of balancing stats, the more likely they will be rewarded for doing so.

I added in multiple endings, which are all just slightly different from each other.

Testing and Integration

May 30 – June 1

I created hangman base but when a letter was guessed, it wouldn't output the underscores with the guessed letters AND previous guesses.

```
String words = "yes";
char guess;
String newGuess = ""; //SAVE NEW WORD W/ GUESSES AS A NEW WORD
// prints out word as astericks

// player enters letter, have loop to cycle through each index until the
// end is reached
// once a letter is found @any point, replace that indexof w/ the letter
// guessed

for (int i = 0; i < words.length(); i++) {
  System.out.print("_");
}

while (true) {
  System.out.println("\n");
  guess = sn.next().charAt(0);
  int i = 0;
  while (i < words.length()) {

    if (guess == words.charAt(i)) {
      System.out.print(guess);

    }
    else {
      System.out.print("_");
    }
    i++;
```

```
___

y
y__

e
_e_

s
__s

s
__s
```

output

I tried to great a new variable to save the 'newGuess', but the output kept adding on.

```
for (int i = 0; i < words.length(); i++) {
  System.out.print("_");
}

while (true) {
  System.out.println("\n");
  guess = sn.next().charAt(0);
  int i = 0;
  while (i < words.length()) {

    if (guess == words.charAt(i)) {
      newGuess = newGuess + words.charAt(i);
    }
    else {
      newGuess = newGuess + "_";
    }
    i++;
  }

  System.out.println(newGuess);
```

```
___
y
y__

e
y__e_

s
y__e__s

s
y__e__s__s

s
y__e__s__s__s

s
y__e__s__s__s__s

s
y__e__s__s__s__s__s

s
y__e__s__s__s__s__s__s

s
y__e__s__s__s__s__s__s__s

s
y__e__s__s__s__s__s__s__s__s
```

Attempting to use string builder below. Top 2 pictures are problems, bottom 2 is FIX.

```
sb.append("yes"); //TEMPORARY

for (int i = 0; i < words.length(); i++) {
  System.out.print("_");
}

while (true) {
  System.out.println("\n");
  guess = sn.next().charAt(0);
  int i = 0;
  while (i < words.length()) {

    if (words.charAt(i) != '_'){
      sb.setCharAt(i, '_');
     break;
    }

    else if (guess == words.charAt(i) && newGuess.char) {
      // newGuess = newGuess + words.charAt(i);
      //ABOVE DIDNT WORK
      sb.setCharAt(i, guess);

    }
    else {
        sb.setCharAt(i, '_');
      // newGuess = newGuess + "_";
      //ABOVE DIDNT WORK
    }
    i++;
  }
  System.out.println(sb);
  //System.out.println(newGuess);
```

```
---

y
_es


e
_es


s
_es


s
_es


s
_es


sj
_es


j
_es


j
_es
```

```
// this fills the StringBuilder with the appropriate # of underscores
for (int i = 0; i < words.length(); i++) {
  sb.append("_");
}
System.out.println(sb);

while (true) {
  System.out.println("\n");
  letter = sn.next().charAt(0);
  int i = 0;
  while (i < words.length()) {

    if (letter == words.charAt(i) && (sb.charAt(i) == '_')) {
      // newGuess = newGuess + words.charAt(i);
      //ABOVE DIDNT WORK
      sb.setCharAt(i, letter);

    }
    i++;
  }
  System.out.println(sb);
  //System.out.println(newGuess);
```

```
---


y
y__


e
ye_


s
yes
```

June 2

Created array for word bank, random number generator to output a word. Realized variables are local to methods. FIX: global local variables. Fixed underscore output to accommodate multiple words. FIX: if statements. Added game menu with arrays, gave more options.

June 3

Added backpack (began to add items)

Made a lot of if statements, may consider revising.

June 4

Tested out code, game menu only appeared on first interaction. After an option was chosen, the game menu was not offered again. When fixed, the guesses (underscores with correctly guessed letters) appeared twice in a row.

June 5

```
public static void assignSkillPoints() {

    System.out
            .println("\nPlease assign points to your desired skills, keep in mind that your total points must sum to "
                    + skillPoints);

    do {
        System.out.println("Enter strength: ");
        strength = sn.nextInt();
    } while (strength > skillPoints || strength < 0);
    System.out.println("Skill points left: " + (skillPoints -= strength));

    do {
        System.out.println("Enter health: ");
        health = sn.nextInt();
    } while (health > skillPoints || health < 0);
    System.out.println("Skill points left: " + (skillPoints -= health));

    do {
        System.out.println("Enter luck: ");
        luck = sn.nextInt();
    } while (luck > skillPoints || luck < 0);
    System.out.println("Skill points left: " + (skillPoints -= luck));

    do {
        System.out.println("Enter charm: ");
        charm = sn.nextInt();
    } while (charm > skillPoints || charm < 0);
    System.out.println("Skill points left: " + (skillPoints -= charm));

    // System.out.println("Are you sure of these stats? (y/n)");
    System.out.println("Character information is as follows for " + name);
    System.out.println("Strength: " + strength);
    System.out.println("Health: " + health);
    System.out.println("Luck: " + luck);
    System.out.println("Charm: " + charm);

    // MAKE BELOW SO THAT, WHEN SKILLPOINTS ADDED DAILY, ITLL ADD TO HERE
    skillPoints = strength + health + luck + charm;
```

Thought it would just be a better idea to have GUI for stat assignment, because if player enters a double or string, it crashes. Having a GUI with a + and – counter would make things easier.
NOTE: Ended up not using GUI, see difference between Proposal and Game Document for greater detail.

June 6

Looped hangman, but the new word would just append every time. Problem was fixed by removed all content from both hearts and word string builder.

```
sb.delete(0, word.length());
sbHearts.delete(0, hearts);
```

June 7

Tried to add the item ‘Lucky Guess’ which gives the player a letter to guess.

```
if (choiceBackpack == 5) {
    if (luckyGuess > 0) {
        luckyGuess--;

letter = sn.next().charAt(0);

//      letter = word.charAt(rando.nextInt(word.length()));

        for (int i = 0; i < word.length(); i++) {
            if (sb.charAt(i) == '_') {
                letter = word.charAt(i);
            }
        }

         for (int i = 0; i < word.length(); i++) {

            if (letter == word.charAt(i) && (sb.charAt(i) == '_')) {
                sb.setCharAt(i, letter);
            }
        }


        System.out.println(sb);


    } else {
        System.out.println("You have none of this item");
    }
}
```

First tried to use math.random, but it would not work if the character at that integer had already been guessed. Decided that random would only be used if the characters luck was high enough.

June 8

Runtime error appeared when one enters a string when asking for a number. Used ASCI conditions in a while statement to prevent error. However, this did not work for when there is more than one character.

```
do {
    System.out.println("\nChoose an option by typing the number next to your desired choice.");
    choice = sn.next().charAt(0);
} while (!(choice <= 57 && choice >= 48));
// above is 0 to 9 in ASCI
choiceNumber = Character.getNumericValue(choice);
```

June 9

Found the method -`sn.hasNext()`- which checks if an input can be read as an int or not. Much easier than method used previous day, but kept both because one is for reading 1 char, other is for reading a string. This method took a few tries to understand as infinite loops were the result the majority of the time. The solution was to break if the string could be read as an int, and return **true** then.

```
633
634     public static boolean isNumeric(String tempTrait) {
635
636         Scanner sn = new Scanner(tempTrait);
637
638         while (sn.hasNext()) {
639
640             if (sn.hasNextInt() == true) {
641                 break;
642             } else {
643                 return false;
644             }
645         }
646         return true;
```

Variables were created accordingly for parameters for the Boolean method.

June 10

Realized a ‘back’ option was needed in the backpack as player was trapped if they accidentally typed the wrong input. Sections of code would crash if the user didn’t enter anything and just pressed ‘enter’ were fixed, usually by creating do while loop to not proceed unless some type of input was entered. Ex.

```
do {
    tempTrait = sn.nextLine();
} while (tempTrait.equals(""));
isNumeric(tempTrait);
```

Without the mini do while, the method would have nothing to read and a runtime error would occur. This method was used in many areas.

June 11

The text seemed to go too quickly, majority of the time one would need to scroll up to see what happened. Added a lot of -`sn.nextLine();`- which required the user to press 'enter' to proceed. This made it a lot more convenient, while also becoming more like a real adventure game where acknowledgement by the user is necessary.

Timers were learned and put in areas to build suspense. Discovered –new Timer at Fixed Rates-, it worked but a code which just pauses the console for a specified amount of time did the same thing and didn't require a method.

June 12

All dialogue written for game since technical parts of code were completed. This just involved adding to arrays.

TESTING IN-DEPTH

Throughout production of the game, many runs were completed to check if a section of code seemed to work the way it was intended or not. The three main types of errors are syntax errors, logical errors and run-time errors. The production was documented above

Syntax Errors

The most common syntax error was undeclared variables and variables that could not be reached because variables within a method could not be accessed outside of it without being included in the parameters. As I was new to methods, this occurred quite a lot. Since a great deal of my variables were used in multiple methods, I wanted to make them 'global'. Upon research, I found how to declare them, and made almost every variable global to avoid picking and choosing which variables I used in more than one method.

Another thing I found was that, even if a global variable was declared, a variable with the **same name** could be created inside a method and no error would occur. Then I would try to use that variable outside the method (hangman method) and wonder why the value wasn't 'carrying over'.

```
// for hangman, global because it is cleared outside of the Hangman method
public static char letter;
public static String word;
```

```
// take letter input as string
// do all condition for letter.charAt(0)
String letter;
```

(in hangman method)

In the end, I only ended up using the global variable of letter in a different method (encounterMenu) which essentially gives the play a hint, without altering the stringbuilder of the hidden word.

Another common error was trying to call methods on a char as if it were a string. Originally, the guessed letter was a ‘char’ because I did not think ahead and accommodate for runtime errors (errors occurring when entering a string for int and vice versa). These errors will be discussed in the ‘runtime error analysis’, but essentially I tried to compare the value of a char to ASCI values. Although this worked when trying to prevent string from being entered when an int value is needed, it proved inconvenient as hangman also needed to append to **incorrectGuesses**.

At first, I just changed **char letter** -> **String letter.** Below is the difference in code (first is wrong, second is fixed).

```
//take letter input as string
//do all condition for letter.charAt(0)
String letter;

letter = sn.next().charAt(0);

for (int i = 0; i < word.length(); i++) {

    if (letter == word.charAt(i) && (sb.charAt(i) == '-')) {
        sb.setCharAt(i, letter);
        isGuessCorrect = true;
    }
}
```

```
String letter;

System.out.println(sb);
letter = sn.next().toUpperCase();

for (int i = 0; i < word.length(); i++) {

    if (letter.charAt(0) == word.charAt(i) && (sb.charAt(i) == '-')) {
        sb.setCharAt(i, letter.charAt(0));
        isGuessCorrect = true;
    }
}
```

This chance was necessary, because when appending the char to **incorrectGuesses (stringbuilder),** the **numeric** value was appended (first is wrong, second is fixed).

```
// if the guess this round wasn't correct,
if (!isGuessCorrect) {

    if ((String.valueOf(incorrectGuesses)).contains((String.valueOf(letter)).toUpperCase())) {
        System.out.println("You have already guessed this letter");
    } else {
        incorrectGuesses.append((String.valueOf(letter)).toUpperCase() + ' ');
        System.out.println("Incorrect guesses: " + incorrectGuesses);
        removeHeart();
        System.out.println(sbHearts + "\n");
        sn.nextLine();
    }
```

```
// if the guess this round wasn't correct,
if (!isGuessCorrect) {

    // checks if letter has already been guessed
    if ((String.valueOf(incorrectGuesses)).contains((String.valueOf(letter.charAt(0))).toUpperCase())) {
        System.out.println("You have already guessed this letter\n");
    }

    // checks if guessed letter is a number
    else if (letter.charAt(0) <= 57 && letter.charAt(0) >= 48) {
        System.out.println("Guess a LETTER, not a number\n");
        sn.nextLine();
    }

    // if neither true, then incorrect guess is valid and player
    // loses a heart
    else {
        incorrectGuesses.append((String.valueOf(letter.charAt(0))).toUpperCase() + ' ');
        System.out.println("Incorrect guesses: " + incorrectGuesses);
        removeHeart();
        System.out.println(sbHearts + "\n");
        sn.nextLine();
    }
```

Since letter is a string, it can NOT be compared using ==.

Small syntax errors were not included since they largely consist of ‘forgetting semicolons’, and easy to fix problems such as declaring a variable.

Logical Errors

Logical errors consisted of the code not going exactly where I wanted it to go. Many consisted of > and < being in the wrong direction, having conditions opposite of what I want, or even doing || instead of &&.

Infinite loops were mainly a problem when trying new things, such as the ‘sn.hasNext()’ method which checks if an input can be read as an int or not. While trying to use the method, an infinite loop occurred as seen below.

```
String s = "yes";

// create a new scanner with the specified String Object
Scanner scanner = new Scanner(s);

while (scanner.hasNext()) {

System.out.println("" + scanner.hasNextInt());


}

// close the scanner
scanner.close();
}
```

For the code on the previous page (I test code in coderpad.io and then try it on my own code), false kept repeating. I then tried to create a method which checks if an input is an integer or not, and the problem did not occur, but text kept duplicating. A common test I do, is to print out a variable to try and see the logic of the code.

```
public static boolean isNumeric(String tempTrait) {

    Scanner sn = new Scanner(tempTrait);

    while (sn.hasNext()) {

        if (sn.hasNextInt() == true) {
            System.out.println("It works");
            return true;
        } else {
            return false;
        }
    }
//  sn.close();
    return true;
}
```

```
It works
It works
Enter health:
1234431
It works
It works
Enter health:
234
It works
It works
Enter health:
```

The line kept repeating twice, this was due to the scanner reading ‘next’ and not ‘nextLine’. Next takes input all the way up to the \n, but doesn’t delete the \n, causing it to be read again and being repeated. This was a difficult process, so I decided to try and code outside of the method depending on the output of the Boolean method. The above code was simplified as much as possible to the following.

```
public static boolean isNumeric(String tempTrait) {

    Scanner sn = new Scanner(tempTrait);

    while (sn.hasNext()) {

        if (sn.hasNextInt() == true) {
            return true;
        } else {
            return false;
        }
    }
    sn.close();
    return true;
}
```

If the above method outputs true, then the integer ‘strength’ will be set to the string ‘tempTrait’ by parsing it. This proceeded without any errors, but is in the test plan because the problem was more so diverted by using different techniques, rather than being ‘solved.

```
                "Health must be at least 8. You will begin your journey with this
        do {
//          do {
                System.out.println("Enter health: ");
                tempTrait = sn.nextLine();
//          } while (tempTrait.equals(""));
            isNumeric(tempTrait);
            if (isNumeric(tempTrait) == true) {
                health = Integer.parseInt(tempTrait);
            } else {
                health = -1;
                // random invalid output forces do while to loop again
            }
        } while (health > skillPoints || health < 8);
        System.out.println("Skill points left: " + (skillPoints -= health));
```

The 2 comments on the do while show a before and after. Without the do while which prevents using from entering in nothing, the user pressing ‘enter’ would just keep going to the next line. This would repeat until the user actually entered a value. The do while ensures that the user is prompted to enter another value.

Another logical error was during skill assignment. Health must be at least 8 since you cannot play the game without any life points. However, the input for health was prompted for AFTER a skill (strength), meaning that if the player chose to input all skill points into strength, then skillPoints -= strength means 0 skill points left, and the code was set so that the player must enter between skillPoints left and 0, however, health minimum was 8, meaning player would be stuck in there forever.

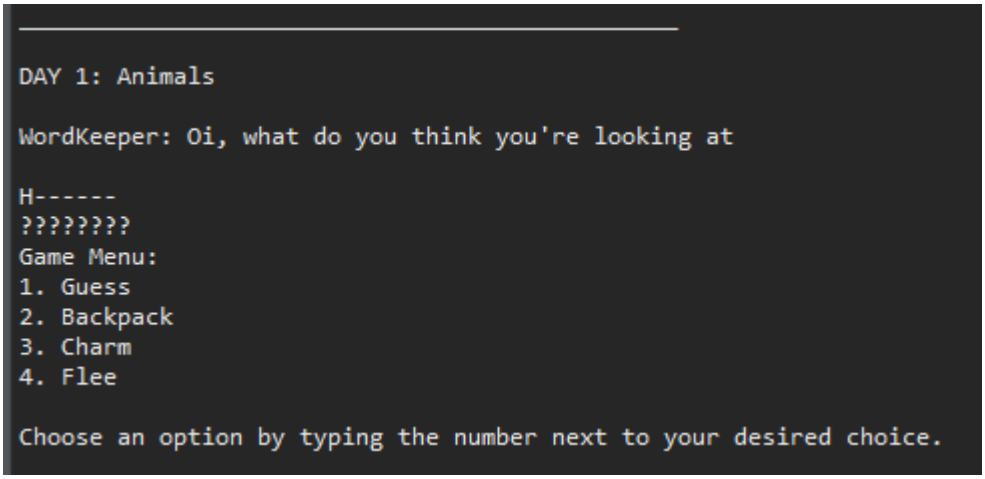
```
do {
    System.out.println("\nIt's mysterious why we value power so much...");
    do {
        do {
            System.out.println("Enter strength: ");
            tempTrait = sn.nextLine();
        } while (tempTrait.equals(""));
        isNumeric(tempTrait);
        if (isNumeric(tempTrait) == true) {
            strength = Integer.parseInt(tempTrait);
        } else {
            strength = -1;
            // random invalid output forces do while to loop again
        }

    } while (strength > skillPoints || strength < 0);
    System.out.println("Skill points left: " + (skillPoints -= strength));

    System.out.println(
            "Health must be at least 8. You will begin your journey with this
    do {
        do {
            System.out.println("Enter health: ");
            tempTrait = sn.nextLine();
        } while (tempTrait.equals(""));
        isNumeric(tempTrait);
        if (isNumeric(tempTrait) == true) {
            health = Integer.parseInt(tempTrait);
        } else {
            health = -1;
            // random invalid output forces do while to loop again
        }
    } while (health > skillPoints || health < 8);
    System.out.println("Skill points left: " + (skillPoints -= health));
```

```
It's mysterious why we value power so much...
Enter strength:
25
Skill points left: 0
Health must be at least 8. You will begin your jour
Enter health:
0
Enter health:
0
Enter health:
8
Enter health:
```

The player is stuck in an infinite loop, the solution was to have the skill points be decided for health first. No picture is required as strength and health input prompting was just switched around.

Another logical error was stringbuilder mishaps, at first it was because the srtingbuilder was never cleared. The word would be overwritten with the next word, however, the underscores for the next word would just append to the end of the old word. The stringbuilder had to be cleared, no picture given for previous code because previous code did not exist. The solution was to clear the stringbuilder (both for hearts and word, and later on incorrectGuesses).

My first attempt at fixing this problem, I deleted stringbuilder contents at the beginning of the hangman code, but then when it was time for the next word, the stringbuilder would only delete to the index of the word just guessed as show below.



The first word was ‘CHEETAH’, notice how the ‘H’ still remains, it is because the word was cleared based on the capacity of the next word, which has a length of 6. Therefore, only the first 5 indexes of ‘CHEETAH’ is cleared, leaving just ‘H’. The new word is then appended. The solution was to clear the stringbuilder after the WHOLE HANGMAN method was run.

```
    playHangman(word);

    sb.delete(0, word.length());
    sbHearts.delete(0, hearts);
    incorrectGuesses.delete(0, incorrectGuesses.length());
    // clears information for the next word

    WordKeeperNotCharmed = true;

}
```

The curly bracket at the bottom loops the game, this means that the information is cleared for every word.

Another common error would be when the user is asked a yes or no question. A do while with conditions was the simple fix. This would loop the program until the user entered a valid input (y/n).

```
//  while (adventureStart == true) {
//      do {
            System.out.println("Enter the name of your character:");
            name = sn.nextLine();
//      } while (name.equals(""));

        System.out.println("Young hero, are you happy with your name? (y/n)");
        System.out.println(name);
        goodName = sn.nextLine().charAt(0);

        // code will only accept y or n w/ 'while'
//      while (!(goodName == 'y') && (!(goodName == 'n'))) {
//          System.out.println("Hopeless romantic, is your name to your liking? yes OR no");
//          goodName = sn.nextLine().charAt(0);

//      }
        if (goodName == 'y') {
            System.out.println("THEN ONWARDS TO LOVE AND POWE-- I MEAN LITERATURE");
            adventureStart = false;
        }

//  }
```

```
Welcome to Hangman Adventures: A Love Story in Beta~
Enter the name of your character:
sdfg
Young hero, are you happy with your name? (y/n)
sdfg
agr
Please assign points to your desired skills, keep in mind that your total points must sum to 20
Enter strength:
```

No matter what was typed, it would just proceed to the next section of code. The code below fixed this.

```
    while (adventureStart == true) {
        do {
            System.out.println("Enter the name of your character:");
            name = sn.nextLine();
        } while (name.equals(""));

        do {
            // this do while loop won't take anything except 'y' or 'n'
            // as an answer, it is used mans times throughout the code
            // for yes and no questions
            System.out.println("\nYoung hero, are you happy with your name? (y/n)");
            System.out.println(name);
            goodName = sn.nextLine();

        } while (goodName.equals("") || !(goodName.equals("y")) && (!(goodName.equals("n"))));

        if (goodName.equals("y")) {
            System.out.println("THEN ONWARDS TO LOVE AND POWE-- I MEAN LITERATURE");
            adventureStart = false;
            // boolean is easiest way to keep track of 'yes' or 'no' as
            // they both only have 2 values
        }

    }
```

A Boolean was very useful here, and was reused in every case where the user was asked a ‘yes or no’ question. This code also ensures that the user can’t enter ‘nothing’.

Run-time Errors

The main runtime error I focused on was when a string was entered for a sn.nextInt(), or an int entered for an sn.nextLine(). This would always crash the code, so I had to take every input as a String. If the variable needed an int, I would check if it was an int, if not, the program would not proceed or loop a prompt until the user enters the appropriate value.

I will begin by discussing variables requiring an int.

```
public static void assignSkillPoints() {
    System.out
            .println("Please assign points to your desired skills, keep in mind that your total points must sum to "
                    + skillPoints);
    do {
        System.out.println("Enter strength: ");
        strength = sn.nextInt();
    } while (strength > skillPoints || strength < 0);
    System.out.println("Skill points left: " + (skillPoints -= strength));

    do {
        System.out.println("Enter health: ");
        health = sn.nextInt();
    } while (health > skillPoints || health < 0);
    System.out.println("Skill points left: " + (skillPoints -= health));

    do {
        System.out.println("Enter luck: ");
        luck = sn.nextInt();
    } while (luck > skillPoints || luck < 0);
    System.out.println("Skill points left: " + (skillPoints -= luck));

    do {
        System.out.println("Enter charm: ");
        charm = sn.nextInt();
    } while (charm > skillPoints || charm < 0);
    System.out.println("Skill points left: " + (skillPoints -= charm));

    // System.out.println("Are you sure of these stats? (y/n)");
    System.out.println("Character information is as follows for " + name);
    System.out.println("Strength: " + strength);
    System.out.println("Health: " + health);
    System.out.println("Luck: " + luck);
    System.out.println("Charm: " + charm);

    // MAKE BELOW SO THAT, WHEN SKILLPOINTS ADDED DAILY, ITLL ADD TO HERE
    skillPoints = strength + health + luck + charm;
}
```

```
Please assign points to your desired skills, keep in mind that your total points must sum to 20
Enter strength:
asrdg
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at hangman.hangmanBasic.assignSkillPoints(hangmanBasic.java:335)
        at hangman.hangmanBasic.main(hangmanBasic.java:85)
```

To fix this, the input was taken as a string, created a method which checks if the input can be read as an int, and then parse the string to an int. The int checker code was created on **June 9th**, and shows my process in more depth. Solution code below:

```
do {
    // must do health first because it has a min. requirement
    System.out.println(
            "Health must be at least 8. You will begin your journey with t
    do {
        do {
            System.out.println("Enter health: ");
            tempTrait = sn.nextLine();
        } while (tempTrait.equals(""));
        isNumeric(tempTrait);
        if (isNumeric(tempTrait) == true) {
            health = Integer.parseInt(tempTrait);
        } else {
            health = -1;
            // random invalid output forces do while to loop again
        }
    } while (health > skillPoints || health < 8);
    System.out.println("Skill points left: " + (skillPoints -= health));
```

This code was repeated for the other 3 traits.

This method was researching because it was easier to read multiple digit ints this way. For situations such as the backpack, where the user needs to type the number associated with the option they want to choose, only a one digit int was required, the check for this was much easier. Instead of reading choice as an int, it was read as a char (because only 1 digit), and then checked through its ASCI values to see if it was really an int.

```
System.out.println("\nChoose an option by typing the number next to your desired choice.");
choice = sn.nextInt();
if (choice == 1) {
```

Above is original code, below is the FIX.

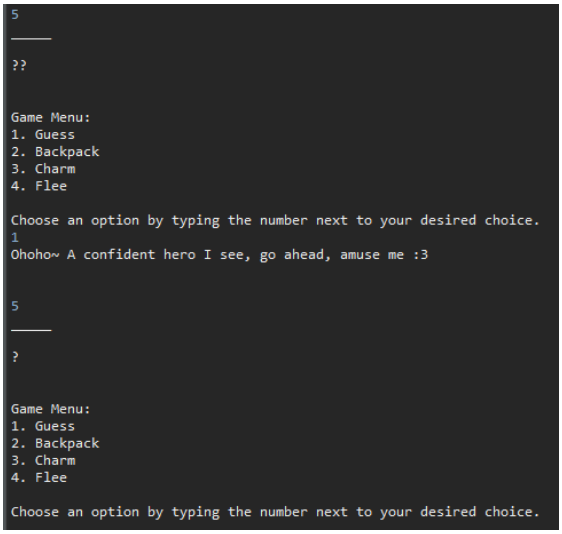
```
char choice;
int choiceNumber;
// made a char because it's easier to ensure that the input is actually
// a number using ASCI
// turned into an int after anyways
```

```
do {
    System.out.println("\nChoose an option by typing the number next to your desired choice.");
    choice = sn.next().charAt(0);
} while (!(choice <= 57 && choice >= 48));
// above is 0 to 9 in ASCI
choiceNumber = Character.getNumericValue(choice);
```

This code sufficed for all sections where only a one-digit number was required, as there are only 10 digits all next to each other. 0 -> 9 is 48->57.

Although a run-time error did not occur, the hangman method accepted numbers as guesses because numbers can be read as strings. Also, the player was penalized for repeated guesses, and it was fixed by creating a stringbuilder for incorrect guesses (problems discussed in syntax).

This was done by checking if the guess contained a letter from the inCorrectGuess stringbuilder. There is not really a before and after code, because there was no 'before', but I will compare the output.



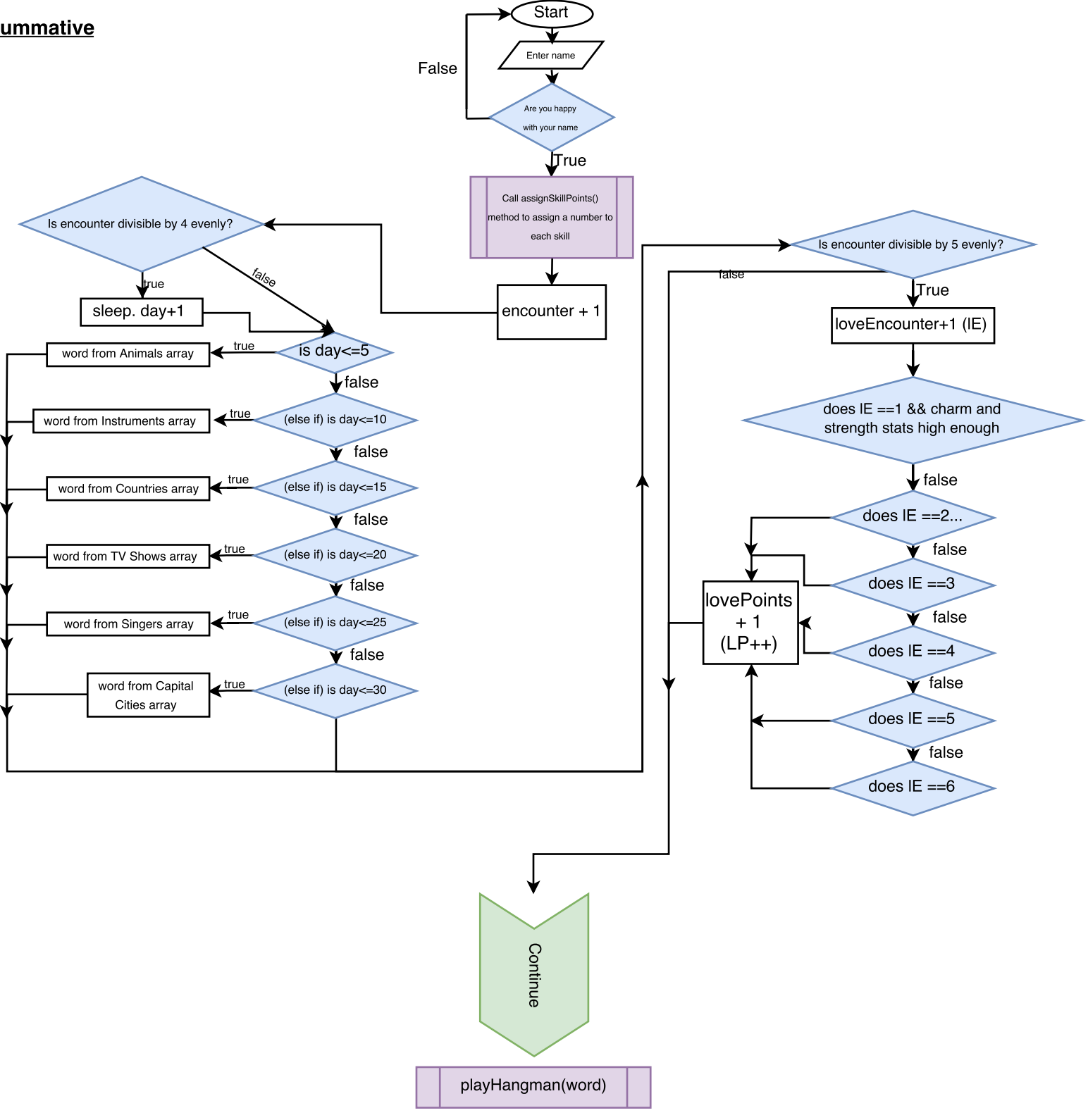
As you can see on the left side, 5 is guessed twice, and a heart (shown as question mark) is taken off BOTH times. On the right side, guesses are saved to prevent this from happening.

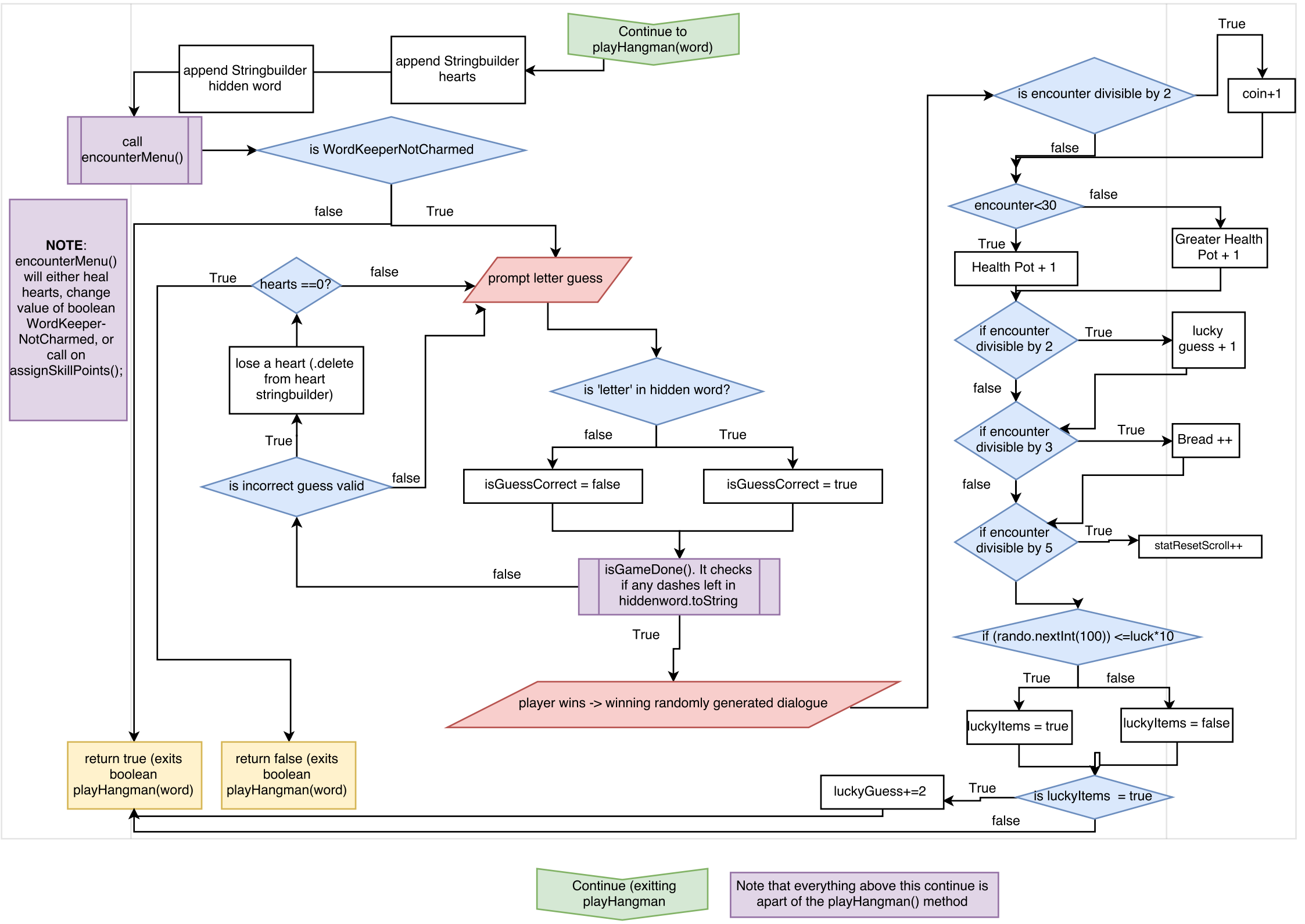
The code was run through multiple tests constantly to check that all input was allowed without a run-time error. The tests consisted of:

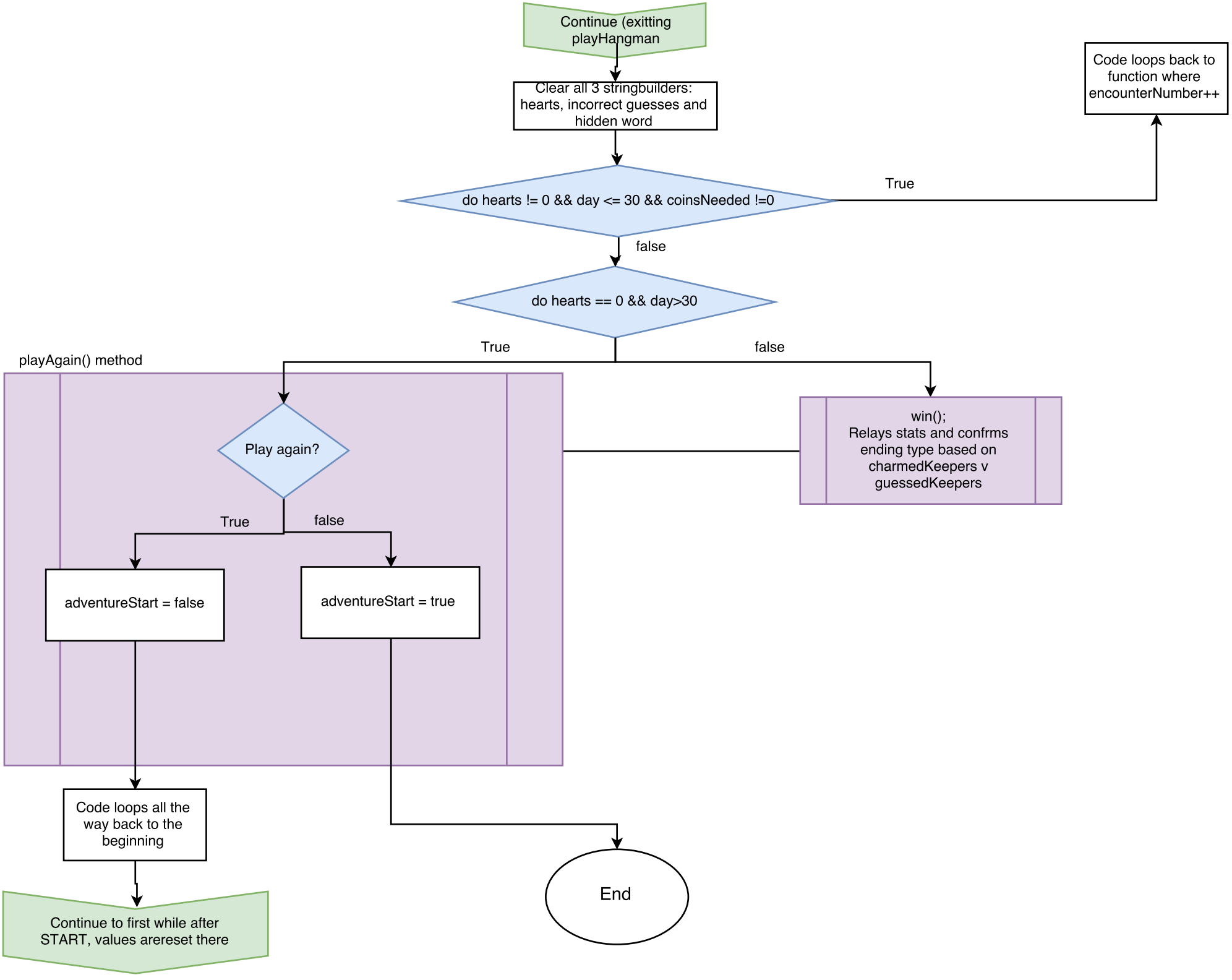
- Entering string when asked for int
- Entering int when asked for string
- Entering nothing (results: either prompt again or code would not proceed until input was entered)
- Constantly changing skill points to see if probability increased with skill points
    - Ex. Increasing charm and strength resulting in encounters with lover
    - Ex. Increasing luck (to 10 which is 100% probability) ensured extra items and LuckyGuess success rate
    - Ex. Increasing health increased max. number of hearts
- As days increased, the array from which the word was pulled from changed
    - Every 5 days was a new array
- At 30 days, I obtained losing scenario, winning brought me to win scenario

Flowchart for Final Summative

Michelle Watson




Continue to playHangman(word)
append Stringbuilder hearts
append Stringbuilder hidden word
call encounterMenu()
is WordKeeperNotCharmed
false
True
NOTE: encounterMenu() will either heal hearts, change value of boolean WordKeeper-NotCharmed, or call on assignSkillPoints();
True
hearts ==0?
false
prompt letter guess
lose a heart (.delete from heart stringbuilder)
True
is incorrect guess valid
false
is 'letter' in hidden word?
false
True
isGuessCorrect = false
isGuessCorrect = true
false
isGameDone(). It checks if any dashes left in hiddenword.toString
True
player wins -> winning randomly generated dialogue
return true (exits boolean playHangman(word)
return false (exits boolean playHangman(word)
True
is encounter divisible by 2
coin+1
false
encounter<30
false
Greater Health Pot + 1
True
Health Pot + 1
if encounter divisible by 2
True
lucky guess + 1
false
if encounter divisible by 3
True
Bread ++
false
if encounter divisible by 5
True
statResetScroll++
if (rando.nextInt(100)) <=luck*10
True
false
luckyItems = true
luckyItems = false
is luckyItems = true
True
luckyGuess+=2
false
Continue (exitting playHangman
Note that everything above this continue is apart of the playHangman() method


Continue (exitting playHangman
Clear all 3 stringbuilders: hearts, incorrect guesses and hidden word
do hearts != 0 && day <= 30 && coinsNeeded !=0
True
Code loops back to function where encounterNumber++
false
do hearts == 0 && day>30
True
false
playAgain() method
Play again?
True
false
adventureStart = false
adventureStart = true
win();
Relays stats and confrms ending type based on charmedKeepers v guessedKeepers
Code loops all the way back to the beginning
Continue to first while after START, values arereset there
End