Unit 2 Programming Assignment

Function Parameters and Return Values, Arrays, String User Inputs, String Methods, Random Numbers

* There are 2 different options for this assignment.
* Fully document your program for communication marks. See the rubric at the end of the document to see what will be evaluated.
* Make sure to program the **Required Functions** for your option for full marks.

# Option 2: Battle Game

1. Create a program that lets the human player battle against a computer opponent.
2. There are 2 different characters that the player can choose to use.
3. From the 2 choices, the human picks one, and then the computer automatically chooses the other character to use in the battle.
4. The in-battle user interface should show both the human player’s and computer player’s characters’ stats:
   1. Name
   2. School-appropriate image.
   3. Current Health.
   4. Maximum health. Choose an integer between 200 and 300, unique for each character.
   5. Normal attack power. This must be an integer between 5 and 10, unique for each character.
   6. Special attack power. This must be an integer between 50 and 100, unique for each character.
      * Special attacks are special because they can do lots of damage, but also because they cannot be used every turn. If a special attack is used by a character, then they cannot use the special attack again until they have waited at least 15 seconds.
5. Set unique values for each of the available characters’ maximum health points, normal attack power, special attack power, and healing power. Each player’s current health points starts off at their maximum.
6. The character selection user interface, which happens before the battle itself, allows the human player to choose one of the available characters. The user can see only 1 character’s picture and stats at a time.

The user has buttons that they can click:

* 1. See next character.
  2. See previous character.
  3. Choose the current character.

1. For the TypeRacer part of this program, use 20 different words
   * Use different lengths of words - some 2 letter words, some 3 letter words, all the way up to a couple of 10 letter words.
   * Put these words into an **array**, sorted with the shorter words at the beginning, and the longest words at the end.
2. When the human user chooses to use a normal attack by clicking a button:
   1. Create a random number to get an index in the words array. The element at that index is the “target” word.
   2. The user must type in that “target” word correctly to do the attack. Use a timer to automatically check their input one letter at a time to see if it matches the next letter in the “target” word.
   3. The user can backspace to correct any errors in the textbox.
   4. The amount of damage done is a random number between the character’s **normalAttackPower** and **(normalAttackPower \* (randomIndex + 2) / 2)**, where the randomIndex is the index of the word that the user typed in. Therefore, harder words give stronger attacks.
3. When the human user chooses to use a special attack by clicking a button:
   1. Create a random number to get an index in the words array. The element at that index is the “target” word.
   2. The user must type in that “target” word correctly to do the attack. Use a timer to automatically check their input one letter at a time to see if it matches the next letter in the “target” word. If they get any letter wrong, then the special attack immediately fails.
   3. The amount of damage done is a random number between the character’s **specialAttackPower** and **(specialAttackPower \* (randomIndex + 2) / 2)**, where the randomIndex is the index of the word that the user typed in. Therefore, harder words give stronger attacks.
4. A timer repeatedly checks the textbox to see if the user typed in the next letter in the word.
   * Use 2 <p> elements in the interface to do the following:
   * One <p> shows all the letters that the user has typed in correctly.
   * The other <p> shows the part of the word that the user still needs to type for the attack.
   * As the user types in letters correctly, the remaining word gets shorter and shorter.
5. The computer should do 2 normal attack every 5 seconds, and then a special attack after the 15th second. The game ends once one of the users has no more health points.

You must declare and properly **use** (at least) the following functions with parameters

|  |
| --- |
| 1. A function named createRandom() that calculates and returns a random number within a certain range. Parameters required: 2   1. The minimum number in the range 2. The maximum number in the range |
| 2. A function named getStringFront() that returns the first letter of a string. This function is used to figure out the next letter of the word that the user is supposed to type into the textbox.  For example, a word might be “left”.  This function would remove the “l” and return it.  Parameters required: 1   1. the string to remove the letter from. |
| 3. A function named getStringBack() that returns a string, with the first letter removed. This function is used to figure out what is remaining for the user to type into the textbox.  For example, a word might be “left”.  This function would remove the “l” and return “eft”.  Parameters required: 1   1. the string to remove the letter from. |
| 4. A function named displayCharacter() to display the information for a character in a <p>. Parameters required: 1   1. the array index of the player to display   Hint: You should use a paragraphs[] array to store each player’s UI paragraph. |

# Marking Rubric

|  |  |  |
| --- | --- | --- |
| Communication – Documentation  Weight 4% /100 | Application – Coding  Weight 9% /100 | Thinking  Weight 2% /10 |
| **Level 4: Clearly documents most to all of the following:**  Header,  Subprograms,  Parameters,  Variable, arrays, and constants,  Mathematical expressions,  Conditional statements,  Loops.  **Level 4: Properly names most to all**  Variable, arrays, and constants,  Subprograms  **Level 4: Correctly indents**  The entire program | **Level 4: Very efficient and correct use of**  Variables and constants,  Conditional statements,  Functions,  Mathematical expressions,  Arrays,  Loops,  Random numbers,  Textbox input,  Data type conversions. | **Level 4:** Creates program backups at least once every 1-2 days on your Google Drive submission. |
| **Level 3: Documents many of the following:**  Header,  Subprograms,  Parameters,  Variable, arrays, and constants,  Mathematical expressions,  Conditional statements,  Loops.  **Level 3: Properly names many**  Variable, arrays, and constants,  Subprograms,  **Level 3: Correctly indents**  Almost all of the program | **Level 3: Mostly efficient and mostly correct use of, while missing one of:**  Variables and constants,  Conditional statements,  Subprograms,  Mathematical expressions,  Arrays,  Loops,  Random numbers,  Textbox input,  Data type conversions. | **Level 3:** Creates program backups every 3-4 days on your Google Drive submission. |
| **Level 2: Documents numerous of the following:**  Header,  Subprograms,  Parameters,  Variable, arrays, and constants,  Mathematical expressions,  Conditional statements,  Loops.  **Level 2: Properly names numerous**  Variable, arrays, and constants,  Subprograms  **Level 2: Correctly indents**  Much of the program | **Level 2: Somewhat efficient and somewhat correct use of, or missing some of:**  Variables and constants,  Conditional statements,  Subprograms,  Mathematical expressions,  Arrays,  Loops,  Random numbers,  Textbox input,  Data type conversions. | **Level 2:** Creates program backups only twice on your Google Drive submission before the due date. |
| **Level 1: Documents some of the following:**  Header,  Subprograms,  Parameters,  Variable, arrays, and constants,  Mathematical expressions,  Conditional statements,  Loops.  **Level 1: Properly names some**  Variable, arrays, and constants,  Subprograms  **Level 1: Correctly indents**  Some of the program | **Level 1: Mostly inefficient and mostly incorrect use of, or missing many of:**  Variables and constants,  Conditional statements,  Subprograms,  Mathematical expressions,  Arrays,  Loops,  Random numbers,  Textbox input,  Data type conversions. | **Level 1:** Creates program backups on your Google Drive submission only once before the due date. |
| **Level R: Documents few of:**  Header,  Subprograms,  Variable, arrays, and constants,  Mathematical expressions,  Conditional statements,  Loops.  **Level R: Properly names few of**  Variable, arrays, and constants,  Functions  **Level R: Correctly indents**  Little of the program | **Level R: Missing most of:**  Variables and constants,  Conditional statements,  Subprograms,  Mathematical expressions,  Arrays,  Loops  Random numbers,  Textbox input,  Data type conversions. | **Level R:** Does not make any backups during the project before its due date. |