SUTD 10.014 Computational Thinking for Design 2020
1D Project

F03
Group 6
Sim Yu Hui, Kellie (1004204)
Wang Xilun (1004877)
M S Subesh Kumar (1005141)
Cheong Cher Lynn (1005458)
Muhammad Zulfiqar Bin Bakar (1005023)

Last Updated: 8 December 2020 (Tuesday)

Description

This game takes the user through a series of scenarios, simulating a Freshmore student's life through Term 1. At the start, the user is randomly assigned a character. While all characters have 4 attributes (Health, Money, Social and Intelligence), the initial values of each of these attributes are dependent on the assigned character. Throughout the game, in each scenario, the user's responses determine the final values of their attributes. Once the game is run, the name of the assigned character, attributes and their respective values are displayed on the top right corner of the screen.

A scenario is marked as such when the user is prompted with a question on screen, and is required to input their choice through the pop-up input box. As mentioned above, the attribute values are updated based on the user's choice. Some of the choices will result in the activation of mini-games, which serve as bonus opportunities to boost the user's attribute values. If the values of any attributes fall below 0 points, a particular set of revival questions will be activated, allowing the user to continue with the game if they meet certain conditions (e.g. having a certain other attribute value to be more than half of the initial attribute value). Should the user fail these revival conditions, they will be considered dead and the game will be over.

At the end of the game, the user's attribute values will be displayed on the screen, acting as a gauge of the user's decision making skills.

The main objective of the game is to not die before the final question pops up. Ideally, the user should have a balanced set of attribute values, showing that he/she is an all-rounded individual that is healthy, financially stable, decently sociable, and equipped with the right skills (i.e. high intelligence).

When coming up with this game, we took into consideration existing games such as Reigns and Episode / Choices. We wanted to incorporate the choice-based element of these games into the game we designed, while making it more fun and relevant for SUTD students.

Documentation

This game requires the **random**, **time** and **turtle libraries** to be imported.

Functions

assign_values(default_attributes, character)

This function updates the default attributes of the character each time the game is run. The default attributes are contained in the dictionary named **default_attributes**. This dictionary has *keys* corresponding to the 4 attributes (Health, Money, Social, Intelligence) and *values* corresponding to the relevant attribute values. The global **assigned_character** variable, which is a randomised value between 1 and 8, is passed as an argument for the parameter **character**. The randomised value was generated using the **randint()** function from the **random** library. Based on the value of **assigned_character**, the values of the dictionary **default_attributes** are updated, where the values of **Health, Money, Social, Intelligence** are increased or decreased by 25 points (decided by using modulo).

bar_fill(bar, value)

This function contains the turtle commands required to fill the bars (drawn within the both the **bar_initial_setup()** function and **bar_update()** function) with colours.

bar_initial_setup(attributes, char_name)

This function takes in the variables **attributes** and **char_name** as parameters. The *values* of the global dictionary **default_attributes** are passed into the **attributes** parameters, where the *keys* are **Health, Money, Social** and **Intelligence** respectively. This function primarily contains the turtle commands required to draw the 4 bars displayed on the top right corner of the screen when the game is run. Within this function, the **bar_fill()** function is run.

bar_update(attributes)

This function is called whenever the bar values need updating. The *values* of the global dictionary **default_attributes** are passed into the **attributes** parameters, where the *keys* are **Health, Money, Social** and **Intelligence** respectively. This function primarily contains the turtle commands required to draw the 4 bars displayed on the top right corner of the screen when the game is run. Within this function, the **bar_fill()** function is run. If the values of the attributes are less than 0, they will be set forcefully to 0. If the values of the attributes are more than 100, they will be forcefully set to 100.

get_attribute_value()

This function stores the *values* of the global dictionary **default_attributes** into separate variables and returns a tuple of these variables in the following order: **(h_value, m_value, s_value, i_value)**.

guess_num_game()

This function calls the mini-game that is run when *option b* is selected for the question where your classmate asks you to help him with a question (3rd question). It assigns a variable **target** a value using the **randint()** function from the **random** library. The user has to guess the value of **target** within 5 tries. Guessing the value correctly results in an increase in points; otherwise, there will be a decrease in points. The variable **guess_num_score**, which is initially assigned 0, will be updated to 1 if **guess == target.** Finally, **guess_num_score** is returned.

hangman_game(char_name)

As the name describes, this function is meant to model the hangman game that we played as kids. This function takes in the parameter **char_name**, which denotes the name of the character assigned at the start. Within the function, string inputs were accepted via **turtle.textinput()** from the **turtle** library and assigns it to the variable **guess**. In the previous function **guess_num_game()**, we observe that the variable name **guess** was also used, and we used the same name in this function **hangman_game()** as this variable was a local variable and hence unaffected by having the same name. For this game, the user needs to guess a hidden word within 8 tries. For every wrong guess, the amount of beer in the mug (displayed on screen) increases. If **guess** is in the variable **word** (initialised with the value of the word to be guessed by the user), the guessed letter is appended to the list **lst**, and displayed on the screen. On the 8th try, the user is required to guess the entire word, and the result is then displayed. If the player has guessed the entire word (i.e. win the game), the outcome of the game will be denoted by the number 1. Otherwise, if the player is unable to guess the entire word, the outcome of the game would be 0. This outcome is later assigned to the variable **hangman_score**. The **default_attributes** dictionary *values* are then updated accordingly.

draw_exam_paper()

This function is called when either *option b* or *option c* are chosen for the last question of the game. It displays 3 questions on the screen (one at a time), and takes input from the user via **turtle.textinput()** from the **turtle** library. If the correct options are selected for each question, the **finals_score** variable is updated from 0 to 1. This function returns the **finals_score** variable, of which will be iteratively added to itself until all 3 questions have been answered. From there, the user's final score can be obtained.

display_finale(attributes)

This function is called at the end of the game. It displays the final statistics of the user's attribute values (**Health, Money, Social** and **Intelligence**). The *values* of the global dictionary **default_attributes** are passed into the **attributes** parameters, where the *keys* are **Health, Money, Social** and **Intelligence** respectively. This function primarily contains the turtle commands required to draw the 4 bars displayed on the top right corner of the screen when the game is run. Within this function, the **bar_fill()** function is run. If the values of the attributes are less than 0, they will be set forcefully to 0. If the values of the attributes are more than 100, they will be forcefully set to 100.

thank_you()

This function is called at the end of the game, displaying the names of the creators of this game.

check_attributes(default_attributes)

This function is called throughout the game. It checks if any of the attributes of the player has dipped below 0, which are stored in the values of the dictionary **default_attributes**. If the current **Health** value has dipped to 0, we will allow the user to be revived should their current **Money** value be greater than or equal to 50. Else, the game will end there for the player. If the current **Money** value has dipped to 0, we will use the **randint()** function and **%** (modulo) to determine the fate of the player. After generating a random integer in the range of 1 to 100, we will use the modulo (i.e. random_integer mod 2) function. Should the result be 1, the player's **Money** value will be replenished to 50. Else, the game will end there for the player. If the current **Social** value has dipped to 0, we will allow the user to be revived by using some of their current **Health** and **Money** values. These values will be generated using the **randint()** function. If the current **Intelligence** value has dipped to 0, we will give the user a chance at revival only if their current **Social** or **Money** value is greater than or equal to 50. Following that, we will use the **randint()** function and **%** (modulo) to determine the fate of the player. After generating a random integer in the range of 1 to 100, we will use the modulo (i.e. random_integer mod 2) function. Should the result be 1, the player's **Intelligence** value will be replenished based on the current **Social** or **Money** values. If the result is 0, the game will end there for the player. The game will also end right away if the player's current **Social** or **Money** value is less than 50.

game_over()

This function is called whenever any attribute dips to 0 and the player cannot be revived (i.e. dies), and whenever the player manages to run through the entire game till the credits scene.