CC 10 group 08 - dating simulator

Team members

Goh Ying Ming, Bryce	1005016
Melodie Chew En Qi	1005319
Atisha Teriyapirom	1005244
Mohamad Arman Bin Mohamad Nasser	1005135

Our game

Our game resembles a dating simulator.

You have 3 days to make a girl fall in love with you, and on each day you have 3 chances to either increase your intelligence, physical strength, or deepen your relationship with the girl. You play minigames to increase your intelligence and strength stat, while having dialogues with the girl can increase her attraction towards you, but choose when you want to talk to her as only certain response options will be unlocked only after you attain a certain stat level. By the end of the 3 days, you will need to hit a certain level of intelligence, physical strength and influence in order to win her heart and thus win the game.

You as the player will go through a total of 3 days with 3 events each for each timeframe: morning, afternoon and night. You will begin with randomized stats each ranging from 75 - 130.
Each event will only have 2 possibilities, either a dialogue or a minigame.
For the dialogue, multiple options will be given to you and awards for each option will be different.

Running the main.py file will start the game.

Dependencies: random and time modules.

Overview of file system

Our file system incorporates a modular and scalable design principle. Folders are organized in a way that editing or adding more features will be easier. Each folder contains functions and data needed for the same category of usage as follows:

events.py will hold a list of dictionaries in which every dictionary will hold all information required to run the event.

eventFunctions.py holds all functions required by each specific event. Therefore, the two main categories of functions are for either specific minigames or to trigger the dialogue.

tictactoeFunctions.py contains all functions needed to run the tictactoe mini game.

stages.py holds all functions that are related to the stage in which the player is at. The functions then invoke the specific event function to start the event.

main.py contains the main flow of the game which we use to invoke functions in stages.

character.py holds the class in which represents our player and holds the initial stats and player chosen name.

util.py holds all functions that are for general usage.

textFiles Folder stores .txt files which store data for either each event dialogue or information needed for the games.

hangmanPrints.py contains all the preset hangman ASCII art to be printed while playing the hangman minigame.

Documentation

File: events.py

Type: List with nested dictionaries
Dependencies: Functions from eventFunctions.py
Schema for our event dictionary:

```
'''
Template of the dictionary is as such

{
    "title": STRING,      | describes the event

    "start": FUNCTION REFERENCE FROM eventFunctions ,        | reference to the function to start the event

    "awards": [{
        STRING : INT       | List of dictionaries which corresponds to the number of options
    }],

    "completionStatus": BOOLEAN,     | Event complete: True or False -> DEFAULT IS FALSE

    "timeframe": STRING,        | morning , afternoon , night

    "dialogueFile": STRING,        | filepath to the .txt file -> "./textFiles/yourFileName.txt"

    "optionsRequirement": [
        {
            STRING: INT             | requirement for each option
        }
    ]
}
'''
```

title describes the event, it is also what the user sees
start is to link the function which triggers the event.

- This function will be from the eventFunctions.py file
- It will either be triggerDialogue or the specific minigame function to invoke.

awards is a list of dictionaries in which each dictionary contains key value pairs of the stats and points to add or delete.

- In the case of a dialogue based event, the length of the award list must be the same as the length of the optionsRequirement list. This is because each award and requirement must correspond to each option given in the txt dialogue file.
- In the case of a minigame, the length of the award must be 1 as the player either gets the award or not.

completionStatus will store the boolean state of if the event has been completed by the player or not. It defaults to False and only changes to true when the eventFunction at **start** gets invoked. This is to ensure that the player cannot repeatedly choose the same previous events.
timeframe stores when this event will be given to the player. It will only be either morning, afternoon or night.
dialogueFile is a string that contains the file path where the dialogue .txt file is located at. This is used by the triggerDialogue event function to open the .txt file and read it.
optionsRequirement is a list of dictionaries in which each dictionary represents the stats requirements needed for each option that is given to the player when a dialogue event is triggered.

File: character.py

Type: Class
Dependencies:

1. **randint from the random module**

Contains the character class, in which contains the following initial key value pairs:

1. day = 1
2. stats which is a dictionary which contains initial stats of the player
    a. intelligence : random integer from 75 - 130
    b. strength : random integer from 75 - 130
    c. influence : random integer from 75 - 130

File: hangmanPrints.py

Type: List of Strings
Dependencies: No module or function dependencies
ASCII art copied from https://gist.github.com/chrishorton/8510732aa9a80a03c829b09f12e20d9c
Is a list of strings that is used by the hangmanMiniGame function in eventFunctions.py, to display the hangman art.

File: historyQuizQuestionBank.py

Type: List of dictionaries
Dependencies: No module or function dependencies
Contains a list of dictionaries in which each dictionary has the following key value pairs:

- Query [STRING] : The question and options [STRING]
- Answer [STRING]: The correct answer [STRING]

File: /textFiles/words.txt

Type: text
Dependencies: No module or function dependencies
Words copied from https://www.mit.edu/~ecprice/wordlist.10000
This .txt file contains words which will be used in the hangman and typing test minigames.

File: /textFiles/_____.txt (all other files except words.txt)

Type: text
Dependencies: No module or function dependencies
Dialogue to display for each specific event.
Syntax for our dialogue text files:

--NAME-- is used to represent the user's name.
--options-- is used to represent the start of the options.
--end-- is used to represent the end of the options.

File: util.py

Function Name: intergerInput() & floatInput()

Dependencies: No module or function dependencies
Argument(s) : inputString [string]
Return: Single [int] or [float] respectively.
These 2 functions take in 1 STRING argument which will be passed into the input function to display on the user's terminal.
It then uses try, except to ensure user input can be converted into an integer or float respectively. If try returns a ValueError, except will use recursion to re-execute the function, prompting the user to reinput a valid input. If no ValueError then user input is returned as a string or float respectively.

Take note that both functions only ensure a valid integer or float input respectively and **DOES NOT** ensure the integer or float is of any other conditions.

Function Name: validateString()

Dependencies: No module or function dependencies
Argument(s) : userInput [string]
Return: [boolean]
This function takes in a string and returns a True if the string passes a set of conditions.
The set of conditions are as follows:

1. String length of between 7 to 14 characters
2. At least 2 numbers
3. Contains no spaces
4. Contains no special characters

It will then validate the string with those conditions and return it if all the conditions are True. If not, it will print an error message stating what exactly is wrong and prompt the user to input again.

File: eventFunctions.py

Function Name: distributeRewards()

Dependencies: No module or function dependencies
Argument(s) : player [object] , awards [dictionary] , pointsToMinusOff (defaults to int 0) [int]
Return: None
This function takes in the player object and awards dictionary, then loops through the dictionary to award the player the stat points accordingly.

Function Name: startHistoryQuiz()

Dependencies: questions [list] from historyQuizQuestionBank.py
Argument(s) : player [object] , event [dictionary]
Return: None
Function takes in the player object and event dictionary in which will be used to change the event's completionStatus to True and distribute awards to the player based on whether the player wins or loses the game. The function also takes the list of questions from historyQuizQuestionBank.py and loops over it, each time printing the question and querying the player for the answer. The function also ensures that the player's input is either "a", "b" or "c". After the user inputs the answer, it will then check against the correct answer and add marks accordingly.

Take note that the function also strips away any trailing or leading whitespace in the player's input.

Function Name: tictactoeGame()

Dependencies:

1. **tictactoeStart function from tictactoeFunctions.py**

Argument(s) : player [object] and awards [dictionary]
Return: None
This function creates an empty tictactoe board using a string with a single space to signify an empty slot. It then invokes the tictactoeStart function from tictactoeFunctions.py. It then distributes the awards if the player wins then changes the event dictionary key of completionStatus to True.

Function Name: mathClassMinigame()

Dependencies:

1. **randint, choice functions from random module**
2. **floatInput from util.py**

Argument(s) : player [object] and awards [dictionary]
Return: None
This function runs the arithmetic math test game. It takes 2 random numbers ranging from 1 to 100, then chooses a random operator from a list [+ , - , * , //]. Using the 2 numbers an operator, it then prints out the expression and prompts the user for an input. It then changes the event's completionStatus to True and checks if the user's input is correct and distributes the awards accordingly.

Function Name: hangmanMiniGame()

Dependencies:

1. **choice function from random module**
2. **words.txt file from textFiles folder**

Argument(s) : player [object] and awards [dictionary]
Return: None
This function starts the hangman mini game. It chooses 1 random word from the words.txt file. It then prompts the user to guess the word. It limits the errors up to 5 wrong guesses. During the user input, the function convert the alphabet to lowercase and will check for the following:

1. Length of guessed alphabet is 1
2. Guessed alphabet is actually an alphabet
3. Guessed alphabet has not been guessed before

Any conditions not met, the function will print an error message.
Once the word is guessed, the awards are distributed and the event's completionStatus is set to True.

Function Name: typingTestMiniGame()

Dependencies:

1. **Sample from random module**
2. **Words.txt from textFiles folder**
3. **Sleep and time from time module**

Argument(s) : player [object] and awards [dictionary]
Return: None
This function starts the typing test mini game. It selects 5 random words from the words.txt file, then it gets the current time in epoch time before printing each word and prompting the user to type. It will also strip the strings before comparing with the actual word. It tracks the alphabets in which the user typed correctly and how many words the user gets wrong. Once the 5 words have been iterated over, it stops the time and compares it to the previously stored time. Using that, it prints out how many alphabets the user typed correctly per second, rounded off to 2 decimal places. It will then distribute awards but also pass in the amount of errors which will be used to minus off the actual award when awarding to the player.

Take note that the function also strips away any trailing or leading whitespace from the player's input.

Function Name: askForOptions()

Dependencies:

1. **integerInput from util.py**

Argument(s) : player [object] , awards [dictionary], lengthOfOptions [integer]
Return: userChoice integer
This function is used to ensure that the user input is one of the options provided in the dialogue. It also checks if the option selected matches the optionsRequirement in the event dictionary. If the player's current stats are not enough for the option, it will then print what stats the user is missing.

Function Name: triggerDialogue()

Dependencies: txt file from textFiles.py
Argument(s) : player [object] , awards [dictionary]
Return: None
This function reads the dialogue txt file and prints out the dialogue while replacing the syntax placeholders as explained in the previous section. It will then distribute the rewards according to the option chosen and change the event's completionStatus to True.

File: main.py

Dependencies:

1. **triggerEventChoice and welcome function from stages.py**

This file controls the main player game flow. It welcomes the user with the welcome function from stages.py which asks the user to input a custom name which will create and initialize the player object.
It then runs the player through 3 days worth of events. After the 3 days, it then determines if the player's stats is enough to win the girls heart.

File: stages.py

Function Name: welcome()

Dependencies:

1. **validateString from util.py**
2. **createCharacter from character.py**

Argument(s) : None
Return: character/player [object]
Asks the user to input a custom name that is then validated by the validateString from util.py. It then uses createCharacter to initialize the player/character object and returns it.

Function Name: getAllPossibleEvents()

Dependencies: None
Argument(s) : timeframe [string]
Return: list of possible events for user to do
This function filters through all events and returns a list of events which fulfills the following conditions:

1. Event's completionStatus is False
2. Event's timeframe is the timeframe passed into function

Function Name: showCurrentStats()

Dependencies: None
Argument(s) : player [object]
Return: None
Using the player object passed in. Iterate through the player's stats and display them.

Function Name: optionInputLoop()

Dependencies: integerInput from util.py
Argument(s) : maxNumber [integer] , player [object]
Return: user's choice of event to play [int]
maxNumber represents the number of event options given to the player, in which maxNumber + 1 is the option to show current stats.This function ensures that the user's input is an integer and that it is one of the options. It then returns it once the user has inputted a valid option.

Function Name: askUserToChooseOption()

Dependencies: None
Argument(s) : events [list], player [object]
Return: user's choice of event to play [INT]
Print out all events in the list and trigger optionInputLoop to prompt the user to choose an event to play.

Function Name: triggerEventChoice()

Dependencies: None
Argument(s) : player [object] , timeframe [string]
Return: None
Get all possible events using getAllPossibleEvents, then ask the user to choose an option using askUserToChooseOption. It then triggers the function that starts the event which is stored in the event object under the key, "start".

File: tictactoeFunctions.py

Function Name: boardTemplatePrint()

Dependencies: None
Argument(s) : None
Return: None
This function prints the board template in accordance to the different print statements in the function. This is to show the player which key to press to access the different boxes in the Tic Tac Toe mini game.

Function Name: boardPrint()

Dependencies: None
Argument(s) : None
Return: None
Code was taken/modified from:
https://www.codegrepper.com/code-examples/python/how+to+code+a+tic+tac+toe+game+with+ai+python

Serves to initially print an empty Tic Tac Toe board. The board is actively modified each time the user or computer plays a move in which a 'X' or 'O' replaces the single space string in the board.

Function Name: isWinner()

Dependencies: None
Argument(s) : bo [list] , le [string]
Return: [boolean]
Code was taken/modified from:
https://www.codegrepper.com/code-examples/python/how+to+code+a+tic+tac+toe+game+with+ai+python

The purpose of this function is to check if a user has won the tic tac toe game. If the three strings (either 'X' or 'O') in any of the specific positions given are the same (==) then the function will return True and the winner is shown. If the function returns False then no winner is named and the game continues on, or it is a tie.

Function Name: randomComputerMove()

Dependencies: choice from random module
Argument(s) : board [list]
Return: move [int]
The use of this function is to return the position that the computer wishes to place its 'O'. First, the function evaluates all the possible positions that the computer can place its 'O' by appending the empty list possibleMoves with the index of where there is a single spaced string in the board. All the indices are then put into the possibleMoves list and the computer makes a random choice from this list where the 'O' is to be placed.

Function Name: tictactoeStart()

Dependencies: integerInput from util.py
Argument(s) : board [list]
Return: move [int]
This function starts the Tic Tac Toe game. It first prints the template board then an empty board. It then prompts the player to input any number between 1-9 to place their 'X'. As the board fills up, there are less strings with a single space and therefore, the possibleMoves list for both the user and the computer decreases in length. Each time the user inputs the value, the function will check whether the value (index + 1) is in the possibleMoves list. If it is, the function will allow the placement of 'X' on the board. If not, the function will ask the player to input another value. In addition the function also checks that the value the user puts in is also an integer by using the function imported from util.py file. Lastly, it will also check whether there has been a winner or not by referencing the isWinner function previously.