Computational Thinking for Design
1D Text-based Game

F09 TEAM 04

Name	Student ID
Yang Haocheng	1004883
Brandon Ong Jian Xun	1004997
Lee Cheng Xin	1005056
Cassandra Dana Chin Kaiwen	1005324
Georgia Tan Hui Yi	1005532

Description

Professor Jerry, a well-respected neuroscientist and computer scientist in his university, has been mysteriously found dead due to unknown reasons. His high profile case has been reported to the town's Police Division and they have assigned you, one of their finest detectives (aka user), onto the case. As you progress through this game, you will uncover the truth behind Jerry's death and piece together your findings to solve the mystery. Perhaps more.

Overview

Flow	Section
1	Character Create
2	Prologue
3	Hacking into Professor Jerry's laptop Username: jerry Password: 14021051
4	1st short game: Math Games Answer: 47
5	2nd short game: Battleship
6	3rd short game: Riddle Answer: Time
7	Ending

Documentation

main.py

The executable file for the game. Simply runs the state machine.

states.py

The states file implements a state machine which is used to smoothly navigate through multiple "events" in the game. Since the game is fairly simple, there are only 3 states defined by the game: Prologue, the starting phase, FileMode, the main part of the game and End the ending of the game

Each state class contains a run function and a next file. The run function executes upon moving into the state, and the next function dictates the next state the function will move into, depending on the current progress of the game.

The State Machine class is responsible for the execution of the game itself. It starts the state at the prologue and runs the current state function. It then checks the next state to move into, and continues until the game ends.

prologue.py

Combines both character_create.py and computer_access.py as they depend on the same input.

prologue_readtext.py

Opens *Prologue.txt*, reads it line by line with a delay by putting **time.sleep** in the **for** loop.

character_create.py

character_create():
Arguments: name, sex_num

Description:
character_create contains the function **character_profile**(*name,sex_num*), which takes in 2 arguments.

This function first defined *counter = 0*, and *sex as an empty string*.

The function then uses a **while** loop to check if the user input the correct option for *sex_num*, if a wrong option is made, the user will have to input the correct option again, which will then be passed onto an **if** function that will stop the **while** loop.

While incorrect input is given:

1) Prompt user for correct input until a correct input is given through a nested while loop
2) Only if input is within the list, function proceeds to the next line.

A small easter egg is added in the form of option 4, '*Yes please'* which will trigger a **while** loop to prompt the user to pick the options again. When triggered, the *counter* will increase by 1, and trigger respective **prints** depending on the **if** conditions.

While correct input for '*sex_num*' is given:

1) Defines '*sex*' respectively
2) If '4' is chosen, loop continues and ask for new input
3) If '4' is repeatedly chosen, *counter* increases by 1, eventually '*Yes please*' will be given
4) If '4' is chosen, then the user inputs out of range, '*sex*' will be defined as '*Burden*'

Finally the function **prints** your *name* and *sex*, followed by checking the **if-else** condition of your *sex*, which will trigger it's respective **prints** that will run through a **for** loop that prints the text character by character, from left to right. This is achieved by doing a *print (x,end='',flush=True)* where *end=''* appends an empty string (Thus giving the left to right effect), and *flush=True* flushes the buffer every time the loop triggers.

computer_access.py

Arguments: user_id and pwd

Description:
computer_access contains the function check_userid(user_id) and check_password(pwd). This is the part of the game when the user needs to "hack" into the professor's computer. The functions ensure that the input by the user is the correct user iD and password before proceeding.

check_userid(user_id):

This function is the main function in this section. If the user ID is not correct, the user will not progress to the password segment.

Before executing the function, the user is required to input what they think is the correct user ID. This input is a string.
The function checks if the user_id entered is == to the correct_id (determined by us), through an **if else** function as there are only two possible outcomes - user gets it correct or wrong.

If the user_id matches up to the correct_id, it then moves on to enter check_password(pwd):.
Else, meaning the user_id does not match up to the correct_id, it enters a **while** loop such that the user can keep trying. In order to engage the user and to drop hints, after each attempt, a slightly different response is produced. The response lets the user know the user_id they have entered is incorrect, then **prints** other phrases at the back.

To break out of the **while** loop, the user needs to get the user_id correct. Once user_id is correct, we go into the next function.

check_password(pwd):

Similar to the check_userid(user_id): function, what the user enters as the presumed password is taken in as a string.

The function then checks if the pwd entered is == to the correct_pwd (determined by us), through an **if else** function. If the user gets the password correct, they are given a response to tell them their login is a success. Else, they enter a **while** loop to allow them to try as many times as it takes to get it right. Just like the check_userid(user_id): function, the user gets a response that informs them the password entered is incorrect and **prints** different phrases to keep it exciting and gives them hints to help them along. The only way to break out of the while loop is to get the password correct.

fileexplorer.py

The game is based around simulating a real cmd prompt to navigate the files in the professors laptop. filexplorer.py contains the FileExplorer class, which represents the files in the system.

The following properties in the class are initialized on creation of the explorer:
Commands: a dictionary list containing the cmd commands that the program accepts. The key is the name of the command, and the value is the resulting function that will be executed.

Files: Represents all the files in the laptop. As the user progresses through the game, the database will add more files.

Filenames: The list of filenames in the laptop, for the ls command.

Current Files: Files in the current directory.

Current Directory: List containing the folders that the user is currently inside. This allows the program to use the cd command easily.

Event: Keeps track of any events that are triggered by the player. The state machine (see states file details) reads this to check if there is a need to transition to a new mode.

In the class, files are represented by a key value pair. The key is the filename (eg. Hello World.txt), and the value is the file content. If the file content is a string, then the file is a txt file and is meant to be read using the read function. If the content is a dictionary, then the file is a folder and can be navigated in and out of using the cd function. If the content is a function, then the file is an executable file or the database file using the run function.

There are a lot of functions in the file explorer class, most of which are the actual commands (cd, read, run, ls, etc.). Those are fairly intuitive to understand, but at its core the parse_text() function is responsible for reading the user input and executing the correct command.

The parse_text command reads the input, and splits the input into words. The first word is taken as the command, and the rest of the words entered is the context. If the command is executable the function executes the command, passing in the context.

At its core, this class is important, and it keeps track of the current stage the player has progressed through and acts as the main medium which the game takes place in.

Game2_remaster.py: Math Game

random.choice(*replies*): gives a random response to wrong answers to add variation in the text interface.

random.seed(*parameter*): gives a number of responses that are returned back to the user.

response(*user_input*): This function allows the user to input an answer which is either a yes or a no. It will then return a response based on the answer that is being input. If the answer provided by the user is yes, then the while loop is considered to be false and the returning statement in the if statement is returned. The question is then given to the user to solve and the game progresses. If no is being inputted by the user, it will enter the while loop where, in order to break out of it, would require the user to respond with a yes such that the game commences.

answer(*user_answer*): This function guides the user through the question, providing the hints required for the user to progress through the game. The user will input his answer and it will thus return a response depending on whether the answer is correct. If the answer is wrong, it will enter a while loop where, after a certain number of tries (3 tries), would provide the user a hint to solve the question. If the answer is correct, the while loop is broken and it will return the statement affirming the user of his correct answer and the game can progress.

battleship .py

In this game, there is a 4*4 board and the ship is randomly set on the board.
The player has 8 chances in total to guess the coordinate of the ship.

print_board(*board*):This function uses a for-loop to generate a 4*4 game board.

random_row(*board*):randomly set the row coordinate of ship.

random_col(*board*):randomly set the column coordinate of ship.

for Turn in range(8):In this for-loop, the user input the guess for the row and the column of the ship in turn. And all the situations are lconsidered in this for-loop. If the user guessed the coordinate of the ship correctly, the for-loop will break and it will return a statement that the user has won the game. If the user guessed a coordinate which is off the board, a statement that says the guess is not on the board will be shown and a chance is wasted.If the user made a guess that has already be made before, a statement corresponds to it will be shown too.After trying for 8 times, the for-loop ends and shows ‘game over’.

Game3_remaster.py: Riddle

random.choice(*replies*): gives a random response to wrong answers to add variation in the text interface.

random.seed(*parameter*): gives a number of responses that are returned back to the user.

answer(*user_response*): This function allows the user to input an answer which is either a yes or a no. It will then return a response based on the answer that is being input. If the answer provided by the user is yes, then the while loop is considered to be false and the returning statement in the if statement is returned. The question is then given to the user to solve and the game progresses. If no is being inputted by the user, it will enter the while loop where, in order to break out of it, would require the user to respond with a yes such that the game commences. The main purpose of this code is to promote the interactivity of the game to the user and it makes it as interactive as possible.

string_answer(*response_back*): This function guides the user through the question, providing the hints required for the user to progress through the game. The user will input his answer and it will thus return a response depending on whether the answer is correct. If the answer is wrong, it will enter a while loop where, after a certain number of tries (3 tries), would provide the user some hints to solve the question. Unlike the previous game where the answer is more straightforward, this question's answer is harder to derive and thus, more hints are provided. If the answer is correct, the while loop is broken and it will return the statement affirming the user of his correct answer and the tip that he should take note of the answer and the game can progress.

database.py

The database file implements the myDiary.db, which roughly simulates a real life database, giving clues to the user when they search the right keywords.

Like other executable files, the database function takes in the explorer object.

Entries is a dictionary variable that contains the keys and values in the database. When a key is entered by the user, the database will add the file to the file explorer. Since the amount of data in the database is not too much, we used a variable to store all the key value pairs.

diary.py

The diary file contains the diary entries of the professor.

end.py

Argument: -
Description: Contain a function end_game() that runs what Jerry is about to say to the user.

The function **prints** a list of text from left to right using *print(c, end="", flush=True),* which was explained in character_create.py.

The function then runs a **while** loop that checks if the **input** prompt was answered correctly.

The function executes it's last **print** from left to right, followed by a *sys.exit()* which exits the game for the user.

Resources

1. Math Game: https://www.rd.com/list/math-riddles/
2. Riddle Game: https://www.rd.com/list/history-famous-riddles/