**Part B: Documentation (50%)**

Documentation supporting your software development journey must be created using Microsoft Word and include the following sections:

|  |  |
| --- | --- |
| Section | Description |
| Planning | Task Definition |
| Structure charts |
| Flow chart |
| Data Dictionary |
| Implementation | Link to your GitHub repository containing source code |
| Testing | Test table |
| Evaluation | Project reflection |

**Software Engineering Year 11: Programming Documentation**

**Project Documentation: ‘Connections.py’**

Table of Contents

[Task Definition 3](#_Toc162708302)

[Data Dictionary 6](#_Toc162708303)

## Task Definition

I have been assigned the task of developing ‘connections.py’, a word puzzle driven python application for the 2023 game of connections, ensuring a functional and fun game play that reflects the game's challenging nature. I aim to execute a will programmed, bug-free application that is user-friendly and more importantly engaging to the user.

The functional requirements I will need to implement include:

* Randomly select 4 categories and in each of those categories it will select there 4 words.
* Display those 12 words in a 4x4 words grid.
* There will be 4 lives.
* For every wrong guess the live will deplete by 1 and for every right answer it will display how many live you have left and congratulate you for guessing all the words correctly with their categories.
* If all your live are less, then 1 so 0 it will display game over.
* Have a restart game function.

**Structure Chart**

This is the approach that I used throughout the development of ‘Connections.py’, this structure chart decomposes the game logic into a mainline and the individual functions within and help visualise the data/parameters that will be passed around.

The following structure chart maps out the functions within my program, a simple run down is here:A diagram of a diagram

Description automatically generated

*This diagram was generated using* draw.io.

**Connections** will be the top-level mainline that starts the game.

**Word categories** is an array of all the categories and words related to the category they are in.

**Create empty grid** will make an empty 4 x 4 grid for the words to be put in.

**Select at random the categories** will be a function that selects 4 categories with 4 words in each for the game.

**Populated grid with four categories** will put the 16 words in the empty 4 x 4 grid.

**Play Game** is the main game loop where the gameplay occurs, including getting the display grid function, get guesses, check if valid and updating the game state. It continues until the player runs out of attempts or puts all the words with their chosen category.

**Update game stats** this will then display the amount of guessed categories (out of 4), and if the lives greater then 0, you won the games if not then Game over.

**Display grid from populated grid** will display the populated grid function.

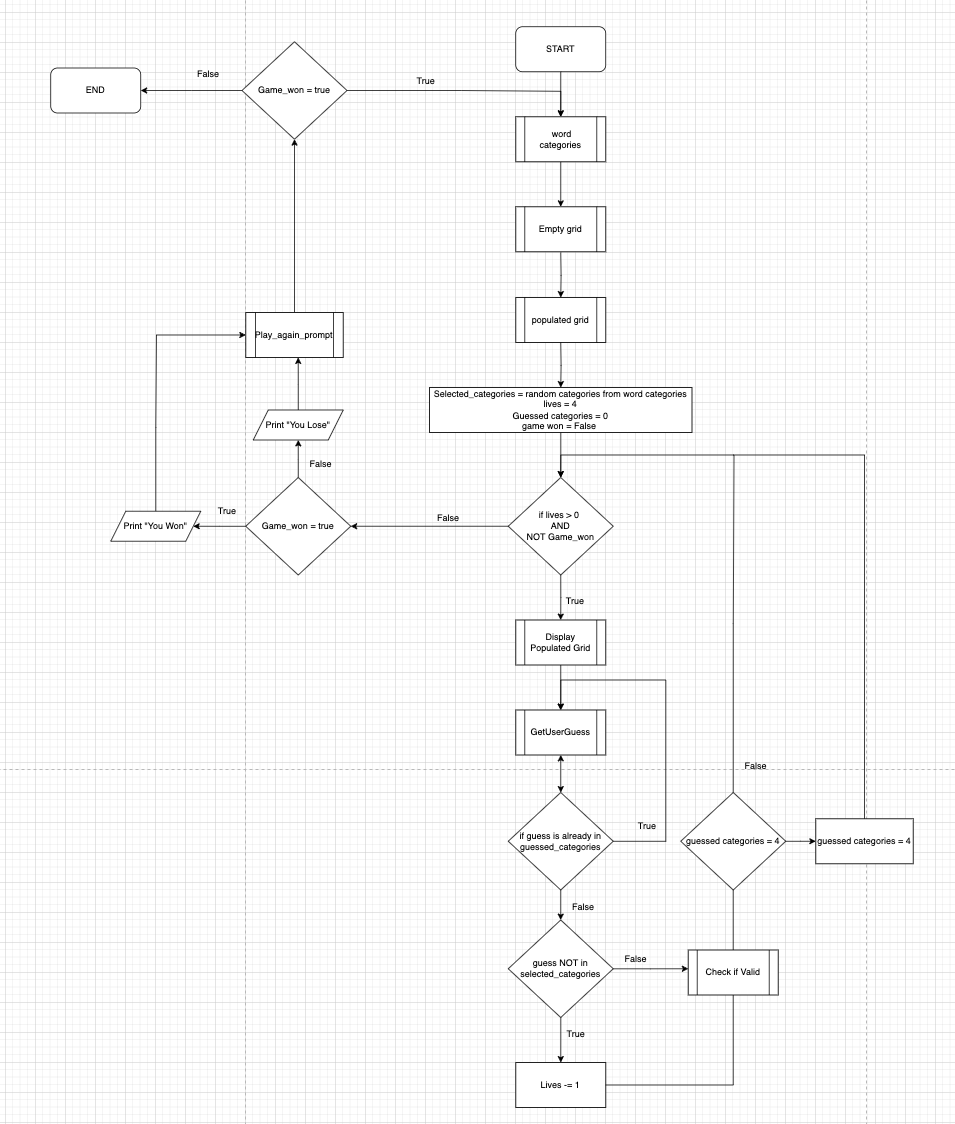
**Get Guess**: This function simply gets 4 words guessed by the player.

**Check if it is valid** this will check if the guess is valid.

**Play Again Prompt**: After the game concludes, this prompts the player to start a new game or exit.

**Flow Chart**

This Flowchart effectively illustrates the way my game “connections.py” uses detailed operations of subfunctions in a simplified way, this overview should perfectly demonstrate the workings of the game, “Connections.py”.

*This diagram was generated using* [*.draw.io*](https://www.drawio.com/)

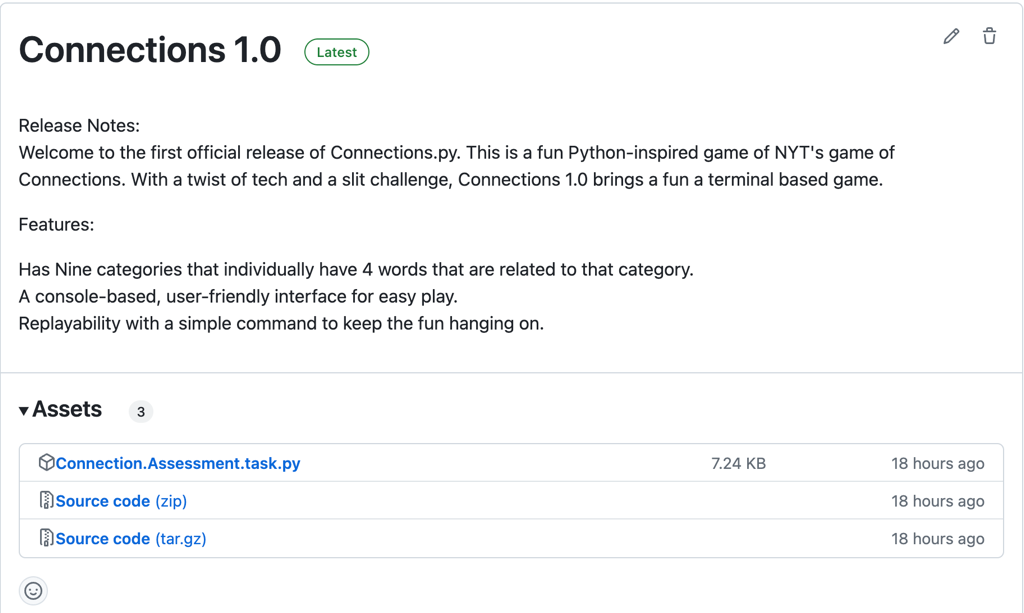
## Data Dictionary

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data Type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
| Word Categories | List[String] | List of strings | - | - | List of potential Categories for the game | "linking\_word":"body\_parts", "words": ["Legs", "Arms", "Fingers", "Feet"] | Must not be empty |
| Selected Categories | String | Text | - | - | The word to be guessed by the player | “word”, “word” , ”word”, “word” | Must be from Word categories |
| Play again prompt | char | characters | - | Y/N | Asking if you want to play again | “Y” of “y” for yes  “N” or “n” for No | Yes or NO only |
| Lives | Integer | Numeric | - | 1-2 digits | Number of lives left for incorrect guesses | Live = 4 | 0 to max number of Lives |
| Check if valid | Boolean | True/False | - | True/False | Flag to determine if the game has been won | True/False | True or False only |
| Getuserguess | string | multiple characters | - |  | Current letter guessed by the player | “input” | Multiple characters |

## GitHub repository containing source code:

This is the 1.0 release of the game Connections.py:

<https://github.com/DeMass199/Task-1-term-1-Assessment/releases>



This is the 1.1 update of Connections.py:

<https://github.com/DeMass199/Task-1-term-1-Assessment/releases>

## Test Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Category | Test Case Description | Input to Provide | Expected Output | Actual Output | Pass/Fail |
| Test 1 | Normal | Verify attempts increment on multiple failures | An incorrect combination of words 4 times | Lives counter reaches 0 and game over | “You Lost”  “Play Again (Y/N): ” | Pass |
| Test 2 | Normal | Check behaviour on last attempt | Correct guess after 3 incorrect guesses and an additional 3 more correct guesses | You have won.  Play again | “You have won”  “Play Again (Y/N): ” | Pass |
| Test 3 | Normal | Validate win condition with minimum guesses | get 4 correct guesses | You have won the game | “You have won the game”  “Play Again (Y/N): ” | Pass |
| Test 4 | Extreme | Input non-alphabetic characters as guess | '1', '@', '-' | incorrect | Incorrect | Fail |
| Test 5 | Extreme | Enter an already guessed word | Correct word guessed twice | Incorrect! | You've already guessed that word. Please enter a different one. | Pass |
| Test 6 | Normal | Check game restart functionality | 'y' after game concludes | Game restarts with initial conditions | Game restarted with initial conditions as expected | Pass |
| Test 7 | Exceptional | Check how game reacts to no words | “”, ””,” ”, ”” | Incorrect! | You've already guessed that word. Please enter a different one.  Incorrect! | Fail |

**Project Reflection**

During the planning phase of Connections, I was unsure on how to start my structured approach but as soon as I got the main idea of how to structure my approach it really helped me through the planning phase and the start of my coding process as I had an idea of what types of functions I would need in throughout my code like a checklist. Even though the planning phase felt useless at the start because it was so tedious I help me to ensure that my code was precise and functional.

Initially, I encountered difficulties with array manipulation, particularly with just displaying the words from the categories and not the linking words with the words. But over time I got familiar with the categories and how they worked and how I could use the categories in my array.

The GitHub repository management proved to be a very tedious but a rewarding aspect of the project, particularly providing me with my code as a backup so when I tried new code I had a reliable backup. The other thing GitHub help with was facilitating efficient documentation for example the updated versions of the game.

The creation of the testing table was a painful moment in the document process, that brought to light serval significant, overlooked bug that could have undermined the entire game.

My proficiency in Python has enhance the way that I have coded this task. As I went through this task, I feel that I have mastered the use of a basic array and how it can be formatted and used throughout a game. Something that I felt I improved in was the use of while and for loops and the method of breaking down complex problems with the use of notes throughout my code. With a basic understanding of object-oriented programming principles, I am looking anxious to exploring and use the skills I have learnt in terms task.