**Saint Augustine’s College, Sydney**

**Software Engineering Year 11: Programming Fundamentals**

**Project Documentation: ‘Connections Py JK’**

Table of Contents

[Planning 3](#_Toc150971422)

[Task Definition 3](#_Toc150971423)

[Structure Chart 4](#_Toc150971424)

[Algorithm Design 5](#_Toc150971425)

[Flowchart 6](#_Toc150971426)

[Data Dictionary 7](#_Toc150971427)

[Implementation 8](#_Toc150971428)

[GitHub Repository 8](#_Toc150971429)

[Testing 9](#_Toc150971430)

[Test Table 9](#_Toc150971431)

[Release and Patch Notes 10](#_Toc150971432)

[Release 1.0.0 10](#_Toc150971433)

[Release 1.1.0 11](#_Toc150971434)

[Release 1.1.1 11](#_Toc150971435)

[Project Reflection 13](#_Toc150971436)

# Planning

## Task Definition

I have been assigned the task of developing ‘Connections.Py’, a command-line driven Python application for the New York Times developed game ‘Connections, ensuring an engaging play that aims to mirror the games nature. I aim to create an application that is highly unlikely to fail and is bug-free with a user friendly and engaging command-line interface.

The functional requirements I will need to implement include:

Randomly select 4 categories with 4 words corresponding to each category from a predefined list at the start of each game.

Generate a 4x4 grid that displays the 16 words shuffled to be out of order.

Use a command line system involving coordinates in the gid to capture player’s guesses.

Validate guesses and reveal correct guess by floating correct category guesses to the top and colourising categories.

Track incorrect guesses, update lives left, and end the game upon reaching guess limits.

When game has ended, reveal the answer by floating category words to their correct spot within the category and colourise categories.

Allow a new game to start once current game concludes and correct answer are shown.

## Structure Chart

As I will be taking a functional approach to the development of ‘Connections Py’, it is appropriate to create a structure chart that will decompose the game logic into a mainline and the individual functions within, and help visualise the data/parameters that will be passed around.

The following flowchart 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* [*.drawio*](https://www.drawio.com/)*. It can be viewed as a template* [*here*](https://drive.google.com/file/d/1uzQsjF8thjtgjTTYEHFJa-khEq4BfrPz/view?usp=sharing)*.*

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

**Choose Categories** will be a function that selects four random categories for the game

**Generate Grid** will be a function that populates a grid with words from the chosen categories randomised

**Play Game** is the main game loop where the gameplay occurs, including getting guesses and updating the game state. It continues until the player runs out of lives or guesses the word.

**Display Game State** will show the current status of the categories being guessed and the remaining lives.

**Get Guess**: This function gets a 4 word guess from the player.

**Update Categories** will be the function that updates the categories the player has guessed

**Check Win** will determine whether all the categories have been guessed indicating a win.

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

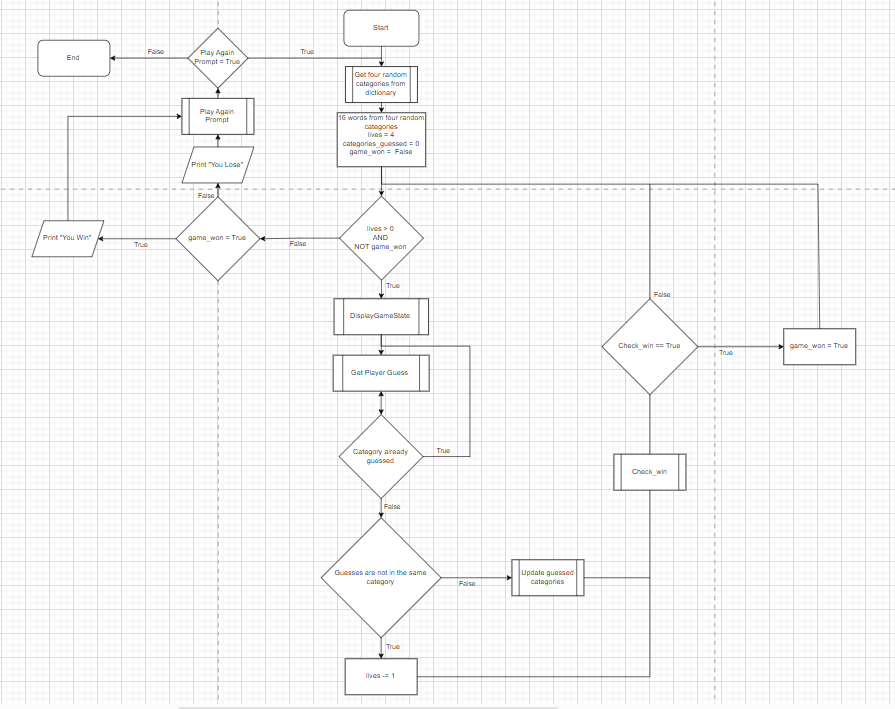
## Algorithm Design

The mainline logic of the 'Connections Py JK' game proceeds as follows:

1. **Start**:
   * Start the game by initializing the list of dictionaries, the number of lives, and other necessary game states.
2. **Gameplay**:
   * Selects 4 random categories using the ‘Choose categories’ function.
   * Generates the grid with words from the categories randomised using the ‘Generate grid’ function
   * Begin the main game loop which continues until the player guesses the word or runs out of attempts.
     + **Game Loop**:
       - Display the current game state using the ‘Display game state’ function.
       - Capture the player's 4 word guess with the ‘Get Guess’ function.
       - If the guess is new, use the ‘Update categories’ function to check if the guess is correct and then update the number of lives and categories guessed.
       - Determine if the player has won with the ‘Check win’ function.
3. **Win/Loss Screen and Replay**:
   * Once out of the loop, display a win or loss message.
   * Prompt the player to play again using the ‘Play Again Prompt’ function.
     + If the player chooses to replay, reset the game variables and restart the game.
     + End the game if the player decides not to continue.

## Flowchart

This algorithm's logic can be effectively illustrated through a flowchart to visually add to understanding of the algorithm. While the detailed operations of the subfunctions are simplified, this overview should adequately convey the workings of the ‘Connections Py JK’ game.



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

## Data Dictionary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable | Data Type | Format for display | Description | Example | Validation |
| flat\_list | List[String] | List of strings | List of words within the chosen categories randomised | ["Block", “Nail”, “Thumb”, “Roomie” | Must not be empty |
| correct | Boolean | True/False | Flag to determine if the guess is correct | True/False | True or False only |
| multiple\_dictionaries | List[Dictionaries] | List of dictionaries | List of dictionaries to choose from | [example\_one, example\_two, example\_3] | Must not be empty |
| lives | Integer | Numeric | Number of lives left before the game is over | 4 | 0 to max number of lives |
| categories\_guessed | Integer | Numeric | Number of categories guessed by the user | 1 | 0 to 4 (max number of categories) |
| game\_categories | List[Dictionaries] | List of dictionaries | List of the chosen dictionaries for the game | [example\_five, example\_two, example\_eight, example\_six] | Must not be empty |

# Implementation

## GitHub Repository

<https://github.com/fong-a/11-SE-Assessmnent-1-Examplar>

**A screenshot of a computer

Description automatically generated**

*This GitHub README.md was created using* [*https://readme.so/*](https://readme.so/)

# Testing

## Test Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Category | Test Case Description | Input to Provide | Expected Output | Actual Output | Pass/Fail |
| Test 1 | Normal | Check if game ends when lives go to 0 | An incorrect guess 4 times | Lives counter reaches 4 and the game ends | “You Lose, you suck” | Pass |
| Test 2 | Extreme | Check if you can win on the last life | Game won after 3 incorrect guesses | Game should indicates win condition | “You have successfully guessed 4 categories”  “You win” | Pass |
| Test 3 | Extreme | Check if you can win without losing a life | 4 correct categories | Game should indicate win before a life is lost | “You have successfully guessed 4 categories”  “You win” | Pass |
| Test 4 | Exceptional | Input uses non alphabetic character as guess for category | “1 5 @ %” | Game should prompt for correct input format | “Type in four words with a space seperating them. Eg. Red Green Blue Yellow:”  Game accepted non-alphabetic characters and took life off. | Fail |
| Test 5 | Exceptional | Enter an already guessed correct category | Correct category guessed twice | Game should indicate that the category was already guessed | “You guessed the category: Minecraft”  “You have successfully guessed 1 categories”  “You guessed the category: Minecraft”  “You have successfully guessed 2 categories”  Game accepted both answers as correct and did not indicate that the category was already guessed. | Fail |
| Test 6 | Exceptional | Enter nothing and press enter 4 times | ‘enter’ key 4 times | Game should prompt for correct input format | [' ']  [' ']  [' ']  [' ']  “You Lose, you suck”  Game accepted non-alphabetic character/nothing as an answer and took life off resulting in the end of the game. | Fail |
| Test 7 | Normal | Check if game ends if all categories are guessed | N/A | “You have successfully guessed 4 categories”  “You win” | “You have successfully guessed 4 categories”  “You win” | Pass |
| Test 8 | Normal | Check if game registers correct guess | “Skate Board Body Ski” | “You guessed the category: Things that can go with the word surf”  “You have successfully guessed 1 categories” | “You guessed the category: Things that can go with the word surf”  “You have successfully guessed 1 categories” | Pass |
| Test 9 | Extreme | All dictionaries are included including first and last when choosing a random category | N/A | First and last dictionary can be one of the random categories in the game | First and last dictionaries appear as categories in the game | Pass |

# Release and Patch Notes

## Release 1.0.0

<https://github.com/fong-a/11-SE-Assessmnent-1-Examplar/releases>

A screenshot of a computer

Description automatically generated

## Release 1.1.0

<https://github.com/fong-a/11-SE-Assessmnent-1-Examplar/releases/tag/v1.1.0-difficulty-mode>

Patch 1.1.0 is a feature update, whereby I introduced different game difficulties, The game now prompts the user to select a difficulty level at the start of the game and will keep prompting them until they enter a valid input. It then selects a word from the appropriate list based on the chosen difficulty level. I have implemented the new words using a dictionary, rather than a simple array of strings.

A screenshot of a computer

Description automatically generated

## Release 1.1.1

<https://github.com/fong-a/11-SE-Assessmnent-1-Examplar/releases/tag/v1.1.1-difficulty-modes>

I noticed after adding the difficulty modes, there was a new bug whereby I had introduced a run-time error. I fixed this bug, and fixed the issue picked up in Test 4, to properly validate player guesses.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test 4** | Faulty Data | Input non-alphabetic characters as guess | '1', '@', '-' | Game should prompt for correct input format | Game rejected invalid characters and prompted for letters | Fail |

A screenshot of a video game

Description automatically generated

# Project Reflection

The planning phase of the algorithms, albeit initially met with skepticism due to my preference for direct coding, taught me the value of a structured approach. Although it extended the time required to accomplish tasks, it ensured the achievement of the set objectives with greater precision.

Initially, I encountered difficulties with array manipulation, particularly with iterating through them. Over time, familiarity with the indexing system grew, simplifying the process.

The GitHub repository management proved to be a rewarding aspect of the project, particularly with the utilization of readme.io, which facilitated efficient documentation formatting. The culmination of development efforts into the initial v1.0 release was a gratifying milestone.

The creation of the testing table was a pivotal moment, underscoring the critical nature of thorough testing. It brought to light a significant, overlooked bug that could have undermined the entire game.

My proficiency in Python has advanced considerably through this first project. I have mastered the structure of a basic game loop and the method of breaking down complex problems into manageable segments. With an understanding of object-oriented programming principles, I am looking forward to exploring beyond the confines of a functional approach next term.

e.g.

Be honest!

What went well?

* Learned a bunch of Python!!!!

- Dictionaries!

- For While Loops

3 of each

Normal: Normal tests are test whereby you are checking that the game works as intended

Exceptional: Exceptional tests are when you are checking how the game deals with incorrect/erroneous inputs

Extreme: Extreme tests involve your arrays, or any variables that involve integers, and whether the program handles the far ends of each range