**Saint Augustine’s College, Sydney**

**Software Engineering Year 11: Programming Fundamentals**

**Example Project Documentation: ‘HangPy’**

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.

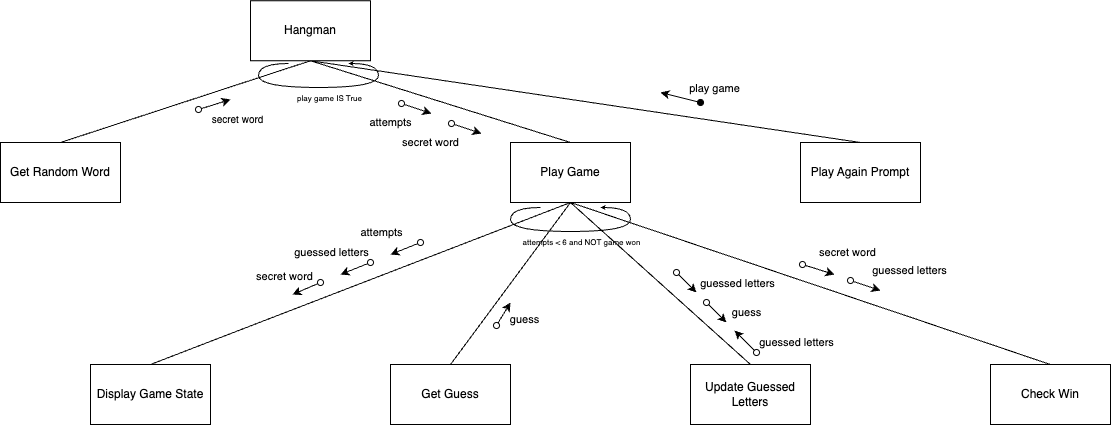
When game has ended, reveal the answer by floating category words to their correct spot within the category and colourise categories.

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 ‘HangPy’, 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:



*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)*.*

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

**Get Random Word** will be a function that selects a secret word for the game.

**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 attempts or guesses the word.

**Display Game State** will show the current status of the word being guessed and the remaining attempts.

**Get Guess**: This function simply gets a letter guess from the player.

**Update Guessed Letters** will be the function updates the set of letters that the player has guessed.

**Check Win** will determine whether the guessed letters match the secret word, 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 'HangPy' game proceeds as follows:

1. **Start**:
   * Start the game by initializing the list of words, the number of attempts, and other necessary game states.
2. **Gameplay**:
   * Select a secret word using the Get Random Word 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 guess with the Get Player Guess function.
       - If the guess is new, use the Update Guessed Letters function to add it to the list of guessed letters.
       - Decrement attempts if the guess is incorrect.
       - 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 augment comprehension. While the detailed operations of the subfunctions are simplified, this overview should adequately convey the workings of the HangPy game.

A diagram of a flowchart

Description automatically generated

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

## Data Dictionary

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data Type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
| word\_list | List[String] | List of strings | - | - | List of potential secret words for the game | ["apple", "banana"] | Must not be empty |
| secret\_word | String | Text | - | - | The word to be guessed by the player | "apple" | Must be from word\_list |
| guessed\_letters | Set[Char] | Set of characters | - | - | The set of letters that have been guessed | {'a', 'e'} | Unique characters only |
| attempts | Integer | Numeric | 4 | 1-2 digits | Number of attempts left for incorrect guesses | 6 | 0 to max number of attempts |
| game\_won | Boolean | True/False | 1 | True/False | Flag to determine if the game has been won | True/False | True or False only |
| guess | Char | Single character | 1 | 1 character | Current letter guessed by the player | 'a' | Single character |

# 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 | Path Coverage | Verify attempts increment on multiple failures | An incorrect letter six times | Attempts counter reaches 6 and game ends | “You Lost”  “Play Again (Y/N): ” | Pass |
| Test 2 | Boundary Value | Check behavior on last attempt | Correct letter after 5 incorrect guesses | Game indicates a win condition | “You Lost”  “Play Again (Y/N): ” | Pass |
| Test 3 | Path Coverage | Validate win condition with minimum guesses | Correct letters of the word in order | Game should indicate win before max attempts | “You Lost”  “Play Again (Y/N): ” | Pass |
| 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 |
| Test 5 | Abnormal Data | Enter an already guessed letter | Correct letter guessed twice | Game notifies letter was already guessed | “You already guessed that letter!  Guess a letter:” | Pass |
| Test 6 | Path Coverage / Replayability | Check game restart functionality | 'y' after game concludes | Game restarts with initial conditions | Game restarted with initial conditions as expected | Pass |
| Test 7 | Boundary Value | Attempt to start game with invalid difficulty | '0', then '4' for difficulty level | Game prompts for valid difficulty input | “Invalid input. Please enter 1, 2, or 3.  Select a difficulty level (1, 2, or 3): 1” | 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.