**Saint Augustine’s College, Sydney**

**Software Engineering Year 11: Programming Fundamentals**

**Cruz Leung Project “assessment.py”**

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 ‘assesseent.py’ (also known as ultimate knowledge quiz), a command-line driven Python application designed for an engaging and competitive quiz experience suitable for all ages in high school or above. The goal is to create an intuitive and easy-to-use quiz game that balances fun, randomness and challenge while maintaining a smooth user experience for a productive learning experience.

The functional requirements I will need to implement include:

* A question system that includes multiple categories: Random, Sports, and Everyday
* A scoring system that tracks correct answers and rewards players accordingly.
* A hint function to assist players when they are stuck
* A leaderboard to increase competitiveness by displaying top 5 high scores and rankings.
* Randomized questions to enhance and engagement and ability to replay.
* A user-friendly command-line interface that makes playing the quiz simple and enjoyable.
* The ability to start a new quiz session once the current game concludes.

## Structure Chart

As I will be taking a functional approach to the development of ‘assessment.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 structure chart maps out the functions within my program, a simple run down is here:

A diagram of a company

AI-generated content may be incorrect.

*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)*.*

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

**Welcome page** will be a function that welcomes the user and creates a banner

**Choose Category** will be the function where the user chooses what category of questions they want to play, it is also looped through after each section. Leaderboard is also displayed in this function when they are no categories remaining for the user to play

**Print questions** will be the main question function. It displays the questions and the choices depending on the user’s chosen category. It is then broken down in the following sequences:

* **Get user input** simply asks user for their answer of the question
* **Check for hint** will be used to check if user entered “hint” in the input get a hint displayed
* **Check user answer** simply checks if user has entered the correct answer
* **Apply and display consequences** will be a set of consequences that will apply to user depending on user input (e.g. score, hint, displaying retry attempts remaining)
* **Update section score to main score** simply adds the score tally to main after each section. Then loops through print questions until all categories are played

## 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

AI-generated content may be incorrect.

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

## Data Dictionary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable | Data Type | Format for display | Description | Example | Validation |
| score | Integer | Numeric | Accumulates the score of the player throughout the all sections of the quiz | score = 2 | Can’t be below 0 |
| section\_score | Integer | Numeric | Accumulates the score of the player in the specific section of questions. | section\_score = 1 | Can’t be below 0 and over 12 |
| name | String | Text | Name that user inputted | name = Peter | Input cannot be empty |
| categories | List of strings | Strings | Used as a tool to display all categories correctly without duplicates | categories = [“random”, “everyday”] | Cannot be topics that are not in categories |
| played\_categories | List of strings | Strings | Categories that the user has already played | played\_category = [“sport”, “everyday” ] | Cannot be topics that are not in categories |
| hint\_count | integer | Numeric | Count to keep the hint function functional and accurat. (Displays different hints depending on counts) | hint\_count = 3 | Can’t be below 0 |
| incorrect\_attempt | integer | Numeric | Keeps count of user’s incorrect attempts per question. | Incorrect\_attempt = 3 | Can’t be below 0 and over 3 |
| imported\_question | Dictionary | Dictionary of strings | Question bank | Imported\_question = ["Topic": "sport", "Question": "What is the national sport of Japan?"] | Dictionary data doesn’t change |

# 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.