This document will instruct and direct you to complete all of the evidence you will need to meet the Merit level criteria for the following standard.

| **Number** | **Version** | **Title** | **Credits** | **Assessment** |
| --- | --- | --- | --- | --- |
| AS91896 | 1 | Use advanced programming techniques to develop a computer program | 6 | Internal |
| **Achievement Level Statement** | | | | |
| Use advanced programming techniques to develop an informed computer program. | | | | |

Please enter the requested evidence in the areas provided.

**PLEASE NOTE : COMPLETION OF THIS WORKSHEET DOES NOT GUARANTEE SUCCESS AT THIS LEVEL. IT ONLY ACTS TO PROVIDE A FRAMEWORK FOR WHICH A JUDGEMENT CAN BE MADE.**

# READ readme.md for extra information about the project

# 2.1

Evidenced within code

# 2.2 - Conventions Used

Each programming language has a set of conventions that should be followed. For Python the current conventions are documented in the [PEP 8](https://www.python.org/dev/peps/pep-0008/) style guide and contain guidance such as:

* Variable and function names should all be lower case with words separated by underscores.
* Lines of code should not be longer than 79 characters
* Lines of comments should not be longer than 72 characters
* Functions should always contain a docstring

Other languages such as [C#](https://docs.godotengine.org/en/3.1/getting_started/scripting/c_sharp/c_sharp_style_guide.html) and [JavaScript](https://www.w3schools.com/js/js_conventions.asp) also have their own style guides.

In the table below identify the conventions you have followed and provide evidence that they have been applied.

|  |  |
| --- | --- |
| Selected Programming Language | Style Guide Used |
| Python | PEP8 |
| Evidence that Conventions have been applied | |
| *e.g.1 - A screen shot of the output from an online checking tool like* [**http://pep8online.com**](http://pep8online.com/)*and / or* [**https://www.pythonchecker.com**](https://www.pythonchecker.com/)  *e.g. 2 - A description of all of the conventions you have taken with examples.* | |
| games/\_\_init\_\_.py    games/game.py    This one issue is just because the action variable is used in a match statement that I had to remove because the checker does not support match statements  games/singleplayer.py    resources/config/\_\_init\_\_.py    flagged unused imports but these are intentionally reexported for other module’s use, which is allowed (afaik)  resources/config/botconfig.py    resources/config/config.py    File required truncation to fit into checker. Also required removing match statement from newer python as checker does not support. The one flagged issue is this:  where the continuation line of the if statement is in line with the inner block. while this checker flags this, PEP8 explicitly states that this is allowed:    <https://peps.python.org/pep-0008/#code-lay-out>  resources/config/gamemodeconfig.py    resources/resourcemanager.py    resources/serverlistmanager.py    resources/servermanager.py    😭😭😭😭😭😭😭😭😭😭😭😭😭  umm this is just way too long it isn’t really shortening it to test, as removing any actual code would cause compilation failures if any required functions were missing, etc. This makes it difficult to test. I, regardless, have tried my best to adhere to the standards while writing this code, so it should be alright  resources/wordlistmanager.py    hangmanbot.py    logger.py    main.py    school has explicitly stated that main file does not require module docstring  parserutil.py    wordproviders.py     * Variable and function names should all be snake case (😭). * Class names are camel case with uppercase first letter * Lines of code should not be longer than 79 characters * Lines of comments should not be longer than 72 characters * Functions, classes, modules should always contain a docstring * Good spacing and indentation * Spaces not tabs (😭) * Surround top-level function and class definitions with two blank lines. * Method definitions inside a class are surrounded by a single blank line.     Etc  Please see <https://peps.python.org/pep-0008/> | |

**NOTE: This section is intended to demonstrate that you are aware of and have adhered to the conventions for your selected language. A screen shot alone does not demonstrate that you have been successful throughout your entire project so make sure that you thoroughly check each aspect of your work.**

**2.3 Testing and debugging the program effectively to ensure that it works on a sample of both expected cases and relevant boundary cases.**

Retest your new code for **both expected and boundary inputs**. Boundary tests are the maximum and minimum values your program should allow for each user input. This means that for each input there is likely to be 2 boundaries that can be tested.

**3.6 Comprehensively testing and debugging the program. (ensure that it captures and manages invalid user inputs).**

Invalid tests are the tests made to make sure your program continues to operate when invalid data is entered. i.e. when the program asks for a number and the user enters some text, you want the program to manage this and continue as expected.

To fully test your program for invalid inputs you need to consider all the possible invalid entries a user or input system could make and test each appropriately (there could be quite a few). If you have built good error capture facilities then everything should work as expected!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test (include test data if necessary) | Expected Result | Actual Result | Test Result |
| 1 | Non-int value in number\_of\_lives field in config file | Log error to server console and use default value (8) | <- | Pass |
| 2 | Negative or zero value for number\_of\_lives field in config file | Should not be allowed | Allowed, player starts with that many lives and dies immediately after first guess, even if it was correct guess | Fail |
| 3 | Deleted gamemode config file while bot running | Next time config file is needed, a default config file is created in it’s place | <- | Pass |
| 4 | Gamemodes in server | All gamemode config files in the server dir should show up as options in /play command, as well as in the gamemode selector (with descriptions) | <- | Pass |
| 5 | Server logs have logger name | Logger name should be based of name of the class that created it, if not explicitly named otherwise | <- | Pass |
| 6 | Non-existing file in word\_list\_paths of gamemode config | Should log error in console and continue without crashing | <- | Pass |
| 7 | Path to directory in word\_list\_paths of gamemode config | Should log error in console and continue without crashing | Tries to .open() dir as if it was file, crashes | Fail |
| 8 | When game of hangman is over with close\_thread\_action set to lock in config file | Should close and lock the thread | <- | Pass |
| 9 | When game of hangman is over with close\_thread\_action set to archive in config file | Should close the thread | <- | Pass |
| 10 | When game of hangman is over with close\_thread\_action set to delete in config file | Should delete the thread | <- | Pass |
| 11 | close\_thread\_action set to value which is not a valid option in config file | Should log error to console and continue using old/default value | <- | Pass |
| 12 | Running main.py | Should actually start the bot, etc | <- (it used to be cyclic dependency hell) | Pass |
| 13 | The WordListManager for a gamemode | should only be (re)loaded the first time that gamemode is played, or on changes to the config, or after restarting the bot (to minimize performance hit) | <- (debug logs only appears in console on first play of game) | Pass |
| 14 | Using /play without a gamemode | Should open the gamemode selector | <- | Pass |
| 15 | Selecting gamemode in selector | Should start the game and remove the gamemode selector | <- | Pass |
| 16 | Using /play WITH gamemode | Should immediately start that gamemode | <- | Pass |
| 17 | create\_thread option True in config file when starting gamemode | Should create a thread to play the gamemode in | <- | Pass |
| 18 | create\_thread option False in config file when starting gamemode | Should not create a thread, and start game in current channel | <- | Pass |
| 19 | Start a game with create\_thread True INSIDE of a thread | Should not crash, just start game inside current thread | Crashes (tries to create thread inside of thread) | Fail |
| 20 | guessers set to public in config file | Anyone can make guesses | <- | Pass |
| 21 | guessers set to private in config file | Only the user who started the game can make guesses | <- | Pass |
| 22 | Capitals in letters in guess | Should be treated identical to lowercase | <- (thanks Wolfi for noticing this in testing, forgot to .lower() guess) | Pass |
| 23 | Extra whitespace around guess | Should be stripped, ignored | User warned that guess can only contain letters (need to .strip() guess) | Fail |
| 24 | No internet, or invalid discord bot token | Log error and fail gracefully | Crash (atleast we don’t try to run anyways and end up in corrupt/invalid state) | Fail |
| 25 | Bot reconnects (short internet outage?) | Continue running as normal | Minor errors with resyncing commands to discord (clashes with already existing commands), causes error, but error is ignored thanks to asyncio logging errors and continuing like nothing is wrong (seems like we end up in potentially valid state, but not very graceful) | Kind of pass but in reality fail |

# Your Code

Please copy the code from the second version of your program into the space below

|  |
| --- |
| V2 Code |
| Its 1655 lines across 16 files so plz no 😭  <https://github.com/wntiv-main/Assessment> if u want it |

**3 Ensuring that the program is a well-structured, logical response to the task**

In the table below explain how you have addressed each of the refined characteristics of the assessment and where further evidence can be found.

|  |  |  |
| --- | --- | --- |
| Refinements | Explanation | Where is it seen |
| 3.1 The program code is efficiently and effectively organised | This makes it easier to read and maintain. quote from PEP8 “One of Guido’s key insights is that code is read much more often than it is written” also PEP20 “Readability counts” it also improves the development process by making it easier to extend and abstract away functionality that can be applied in many contexts | I use an organised class structure to handle the large amount of data that needs to be processed as there could be multiple discord servers each with many gamemodes etc., so there is a clear need to maybe group info about a gamemode into a class (GamemodeConfig), and many gamemodes belong to a server (ServerManager), and the discord bot (HangmanBot) has many servers (ServerManagers) in a (ServerListManager) |
| 3.2 The program has features to help manage user input | Make sure that the user can enter a set of allowed inputs, and any invalid inputs should be reported to the user WITHOUT crashing the game or making an invalid state etc. | Invalid input handling in SingleplayerGame.\_update\_inner()  To make sure that the user guesses something valid  also  ServerManager.\_escaped\_name()  To make sure gamemode names are valid for file names |
| 3.3 The program has user friendly output | This is important as it makes the user’s experience much better by increasing the aesthetic of the game | I use discord as a user interface as my game is targeted towards discord users and they will feel familiar with the UI. I (optionally) use discord threads for organisation and to avoid complete spam in main channels |
| 3.4 The program includes error capture features | This is important as many thing about the environment the code is running in could change at any point – e.g. the users filesystem could be hostile, or an invalid value could be passed into a function (e.g non-integral string passed to int()) | I have advanced handling for all my file interfaces. This is mostly just the Config class (resources/config/config.py), which handles all cases where a file might not exist, or might be in use by another process. Additionally, the config parser (in class Config$Entry) uses error capture to capture any errors caused by invalid values from the AbstractParser, as an invalid value here should not crash the program, but instead just log an error and continue with the old or default value for that config entry |
| 3.5 The program report unexpected behaviour to the user | This allows the user to fix the error (if it is in their power to do so), or report is to the developer. This is important so that the user knows that something is gone wrong, as opposed to them just thinking that it is taking a long time or something. | Any invalid values from the user are reported to them, as these are likely fixable by them by re-entering a valid value, or something. Visible in SingleplayerGame, where invalid guesses send a relevant message to the user telling them what exactly is wrong and prompting them to try again. Other errors, such as invalid internal states or errors in the server backend, are logged in the server, as these are things that are likely unfixable by the end user, unless they have access to the server, in which case they would be able to see the server logs, which is where these kinds of errors are logged to instead, so the sysadmin or developer or me can handle them. |