This document will instruct and direct you to complete all of the evidence you will need to meet the Achieved 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 a computer program. | | | | |

**Submission Checklist (Achieved)**

To meet the achievement level criteria it is important that your code meet the minimum expectations of the assignment. Before submitting, please complete the checklist below to ensure you have not missed anything significant in your submission.

|  |  |
| --- | --- |
| Checklist | Done? Y/N |
| My program uses variables storing at least two types of data (e.g. numeric, text, Boolean) | Y |
| My program uses sequence, selection (IF) and iteration (LOOP) control structures | Y |
| My program uses Input from a user, sensors or another external source & produced output | Y |
| My program uses two or more advanced programming techniques. | Y |
| My code is set out clearly using suitable whitespace | Y |
| I have included comments to document and explain what the code is doing | Y |
| I have completed the testing table below to show that I have tested my program works for expected inputs (plural) | Y |
| I have copied my complete code into the space provided in this document | N |
| I have prepared a video recording of my program working and will submit it with this document. | y |

# 1.1 - 1.5

Evidenced within code

# 1.6 Advanced Tools

In the table below identify the advanced tools you have used, why they have been used and where the evidence of their use can be found.

|  |  |  |
| --- | --- | --- |
| Advanced Tool | Why was it used | Where is it used |
| *e.g. Functions using parameters* | *To generate new ….* | *This is called from the …* |
| **modifying data stored in collections** | Store the word as it is going to be displayed, and updating it with the correct letters in the right places as the user makes correct guesses | Singleplayer.py, in class Player in method turn() |
| **creating methods, functions, or procedures that use parameters and/or return values** | Classes with relevant methods, requiring or returning relevant data, are very useful to keep the codebase organised and make it clear which parts of the code are performing what task | Literally everywhere |
| **using non-trivial string manipulation** | Finding indices of a char in a string, to accurately update the displayed word/underscores with the users guessed char | Singleplayer.py, in class Player in method turn() |

# 1.7 Expected (Valid) Input Testing

Expected (Valid) tests are tests that show your program operates as expected if the inputs received are as you expect them to be. For each input test that your program works using valid inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test (include test data if necessary) | Expected Result | Actual Result | Test Result |
| 1 | Lives drops below zero at once  Enter a few guesses then a whole word guess | dies | dies | Pass |
| 2 | Lives drops to 0 | dies | dies | Pass |
| 3 | Lives above zero | Remains alive | <- | Pass |
| 4 | Single letter guess in word | Reveals all instances of that letter | <- | Pass |
| 5 | Single letter guess not in word | Removes a life | <- | Pass |
| 6 | Substring guess in word  Guesses a part of the word like ‘ing’ | Reveals the substring AND all other letters that appear in the substring | <- | Pass |
| 7 | Substring guess not in word | Removes lives based on length of guess | <- | Pass |
| 8 | Full word guess correct | Immediate win | <- | pass |
| 9 | Incorrect full word guess | Removes lives based on length of guess | <- | Pass |
| 10 | Guess contains non-alphabetic char | No lives removed, warn user | <- | pass |
| 11 | Full-word guess longer than word | No lives removed, warn user | <- | pass |
| 12 | Config file does not exist | Default file is created with default options | <- | Pass |
| 13 | Invalid int/float literal in config file option that expects one | Error logged and old/default value used | <- | Pass |
| 14 | Invalid enum name in config file option that expects one | Error logged and old/default value used | <- | pass |
| 15 | Config file number of lives | Game should start with number of lives stated in config file | <- | Pass |
| 16 | Config file should only reload if it has been changed since last read  Change file and enter guess (check\_config\_changes() is called between each turn) | Config file should be re-parsed (check logs for print) | <- | Pass |
| 17 | Config file should only reload if it has been changed since last read  DO NOT change file and enter guess (check\_config\_changes() is called between each turn) | Config file should NOT be re-parsed (check logs for print) | <- | Pass |
| 18 | Changing dictionary\_location in config should reload RandomWordProvider (callback listener)  Change dictionary\_location in config file then guess as check\_config\_changes() is called between each turn | RandomWordProvider should reload | <- | Pass |
| 19 | Changing dictionary\_location in config should reload RandomWordProvider (callback listener)  DO NOT change dictionary\_location in config file then guess as check\_config\_changes() is called between each turn | RandomWordProvider should NOT reload | <- | Pass |
| 20 | All letters guessed (no more underscores in displayed word) | Game should end and print win condition | <- | Pass |
| 21 | Not all letters guesssed | Game should continue (until run out of lives) | <- | Pass |
| 22 | Word blacklist – blacklist files in dictionary\_location in config | no words in the blacklisted files should be displayed | <- | Pass |

# Your Code

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

|  |
| --- |
| V1 Code |
| There are multiple files so please don’t make me do this  The code can be found attached in zip |