| **Number** | **Version** | **Title** | **Credits** | **Assessment** |
| --- | --- | --- | --- | --- |
| AS91906 | 1 | Use complex programming techniques to develop a computer program | 6 | Internal |

|  |
| --- |
| **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 complex 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 prepared a video recording of my program working and will submit it with this document. | Y |
| I have completed the appropriate testing table(s) to show how effective my program is. | N |
| I have copied my completed code into the space provided in this document | Y |
| I have prepared a video recording of my program working and will submit it with this document. | Y |

|  |
| --- |
| **About Your Project** |

In the space below, tell me anything about your program that I may need to know.

Include elements such as…

* What features did you add beyond the basic brief?
* Were there any changes to the brief that were agreed by your teacher?
* What environment is needed for your code to run?
* Etc.

In short, explain anything beyond hitting run button for a standard python environment and expecting the program to act as the example program provided.

|  |
| --- |
| Any windows computer should be able to run this program as it is a windows executable, a windows 10 computer would be preferred |

|  |
| --- |
| **1. Achieved Level Documentation** |

# 1.1 - 1.6

Evidenced within the code

# 1.7 Complex Tools

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

|  |  |  |
| --- | --- | --- |
| Complex Tool | Why was it used | Where is it used |
| *e.g. Student created classes/objects* | *To generate new ….* | *These are created …* |
| Classes/objects | To reduce the amount of repeated code having to be written by the programmer, thereby increasing efficiency. | most .cc documents and .h documents are separate classes used by the program to make the game run. |
| GUI | To give the end user of my program a better interface to interact with my program, graphics make a much more versatile way of outputting data to an end user than a text based program. | The GUI is used to display the game to the end user, in both the menu and the game itself. |
| External storage | So that the user may close the program and then re-open it as a later date, having their progress saved | A save .txt file is included in the assets folder which is update data the end of execution, and then is loaded in at the start of each execution. |

|  |
| --- |
| **2.Merit Level Documentation** |

# 2.1 – Variable Names & Comments

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 |
| C++ | [Google style guide](https://google.github.io/styleguide/cppguide.html) |
| Evidence that Conventions have been applied | |
| *e.g.1 - A screen shot of the output from an online checking tool like* [*https://www.codewof.co.nz/*](https://www.codewof.co.nz/)  *e.g. 2 - A description of all of the conventions you have taken with examples.* | |
| Examples   |  |  |  | | --- | --- | --- | | convention | Where it is seen | Stylegiude reference | | Indentation | Each block of code is indented by two spaces, NOT TABS for readability and compactness. | [Spaces vs. Tabs](https://google.github.io/styleguide/cppguide.html#Spaces_vs._Tabs) | | Variable naming | All variables are named in snake case (all lowercase letters separated with underscores between the words), apart from constants, which are named using pascal case (capitalization of the first letter of the first word in each word) and have a leading k at the start of the name. Class member variables also have a trailing underscore to mark them as member variables. | [Constant Names](https://google.github.io/styleguide/cppguide.html#Constant_Names), [Variable Names](https://google.github.io/styleguide/cppguide.html#Variable_Names) | | Function naming | All functions are named in pascal case (capitalization of the first letter of the first word in each word). | [Function Names](https://google.github.io/styleguide/cppguide.html#Function_Names) | | File naming | My files all conform to the naming conventions in my style guide, all my files are named snake case (all lowercase letters separated with underscores between the words). All header files end in .h and all C++ files end in .cc | [File Names](https://google.github.io/styleguide/cppguide.html#File_Names) | | Line length | Line must not exceed 80 charters for readability purposes, read more on like to left. All of my code is organised so that no line exceeds 80 charter long, and if it does exceed 80 characters, then the line is split in the most easily readably way. | [Line Length](https://google.github.io/styleguide/cppguide.html#Line_Length) | | Copyright notice | Each of my files starts with the copyright notice and licence boilerplate, as my code is not open source, this licence is not needed but the copyright notice is. | [File Comments](https://google.github.io/styleguide/cppguide.html#File_Comments) | | Function comments | All function declarations in my code have a function comment immediately preceding them which should describe the purpose of the function, and the definition should have comments regarding the behaviour of the function, in the style guide obvious functions, such as mutators and accessors (set get and other similar methods), but I have chosen to include function comments for these as well as the standard demands this. | [Function Comments](https://google.github.io/styleguide/cppguide.html#Function_Comments) | | Define header guard | All .h files have an ifndef statement at the start of them to prevent multiple inclusion, the macro names for this guard are formatted in all caps with the project name, then a underscore, followed by the file name (I only have one directory under the src folder) and then \_H\_. | [The define Guard](https://google.github.io/styleguide/cppguide.html#The__define_Guard) | | Non-ASCII charters | The only non ascii charter in my code are the ¢ symbol to (in this game’s case) designate credits, these are all formatted as UTF charters (\u00A2 in this case).  see [menu\_items.h](../inculde/menu_items.h). | [Non-ASCII Characters](https://google.github.io/styleguide/cppguide.html#Non-ASCII_Characters) | | |

**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.**

|  |
| --- |
| **3.Excellence Level Documentation** |

In the table below explain how you have addressed each of the refined characteristics of the assessment and where this can be seen in your program.

|  |  |  |
| --- | --- | --- |
| Refinements | Explanation | Where is it seen |
| 3.1 The program code is efficiently and effectively organised | The code should be organised in a way so that anyone looking at my code, whether it be another contributor, or someone else, should be able to read and understand my code, and even be able to build upon it. | By using a style guide, commenting, separating each of my classes into separate documents, and organising my directories based on convention I have ensured that my code is organised well. |
| 3.2 The program has features to help manage user input | This is all about guiding the player into making valid inputs into my program. | By using a GUI for a menu, I am able to restrict the users possible inputs so that the are more likely to pick valid ones, I also have a tutorial section on how to play the game so that the player knows what controls do what. |
| 3.3 The program has user friendly output | The better the player is able to understand what is going in in the game, the more of an enjoyable experience they will have in the playing of the game | My program uses a GUI as a way to output data to an end user. |
| 3.4 The program includes error capture features | So that the user is informed of any abnormalities and or errors in the code and can respond to them efficiently. | In library initialisation, dynamic casting and asset loading, the program checks and reports back any abnormalities in the code. |
| 3.5 The program report unexpected behaviour to the user | when an error does occur, the player is informed about it, then the player is able to diagnose the problem. | I have created custom errors in the terminal that will help guide the player into being able to diagnose or fix the problem. |

|  |
| --- |
| **4.Final Submission & Testing** |

# 4.1 - Your Video

Record a video showing your game in operation with valid inputs. If you wish to show any error capturing you have included in your program, record these in a separate video.

Be sure to take your time and give the viewer of this video and opportunity to fully see your game in action. Save your video recording in a suitable file format (e.g. wmv or mp4 – NOT an iSpring file)

|  |
| --- |
| Below tell us the name of this file and where it is stored. |
| <video.wmv> |

# 4.2 – Demonstrate Authenticity

Evidenced by Teacher throughout project development & after submission

# 4.3 Valid “Expected” Input Testing (Achieved)

Valid “Expected” 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 | Key w Pressed while in game | Currently selected robot will move forward by an amount scaled by their speed. | As expected, | pass |
| 2 | Key a pressed while in game | Currently selected robot will rotate anticlockwise by an amount scaled by their speed. | As expected, | pass |
| 3 | Key d pressed while in game | Currently selected robot will rotate clockwise by an amount scaled by their speed. | As expected, | pass |
| 4 | Key space pressed in game | Currently selected robot will shoot a bullet in the direction it is facing. | As expected, | pass |
| 5 | Key v pressed in game | A bar will appear and fill us as lone as v is pressed. | As expected, | pass |
| 6 | If above stated bar is full, | The robot the player is currently controlling will switch | As expected, | pass |
| 7 | When the play button in the menu is pressed, | The menu will hide and the game window will open | As expected, | pass |
| 8 | When the quit button, the close button on the menu or the escape key in the menu is pressed | The game will quit, closing all windows | As expected, | pass |
| 9 | As above. | The game is saved to the text document | As expected, | pass |
| 10 | When an upgrade is purchased, | The amount of credits that upgrade cost is deducted from the total credits and the display in the menu is updated to reflect that | As expected, | pass |
| 11 | If a repair button is pressed | The robot relating to that repair button is repaired, decreasing the damage taken by 5 | As expected, | pass |
| 12 | When a robot upgrade is bought, | The corresponding robot is upgraded | As expected, | pass |
| 13 | When a weapon upgrade is successful purchased | The corresponding weapon is upgraded | As expected, | pass |
| 14 | When a weapon button is pressed in the menu for either robot | The selected weapon changes color, and when the game is loaded in, the corresponding robot is welding that weapon | As expected, | pass |
| 15 | The How to button is pressed | A window will open displaying a brief tutorial of the game. | As expected, | pass |

# 4.4 Boundary Input Testing (Merit)

Boundary tests are tests that show your program operates as expected if the inputs received are at the extreme ends of the allowed input range. It is also worth including the other side of each boundary to show that the boundaries are actually operating correctly. E.g. To test an input asking for a number between 1 and 10, apply 4 tests: 1 then 10, but also 0 and 11!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test (include test data if necessary) | Expected Result | Actual Result | Test Result |
| 1 | A robot attempt to shoot another robot | The shot robot takes damage | As expected | pass |
| 2 | An enemy attempts to shoot another enemy | The shot enemy takes damage | As expected | pass |
| 3 | Robot is repaired at damage = 5 | Damage goes to 0 | As expected | pass |
| 4 | the player attempts to purchase an upgrade for a robot without the sufficient funds | A popup display that the funds are insufficient appears | As expected, | pass |
| 5 | the player attempts to purchase an upgrade for a weapon without the sufficient funds | A popup display that the funds are insufficient appears | As expected, | pass |
| 6 | the player attempts to repair a robot without the sufficient funds | A popup display that the funds are insufficient appears | As expected, | pass |
| 7 | the player attempts to repair a robot where that repair would put the robot at a negative damage | A popup display that the health is already full appears | As expected, | pass |
| 8 | A robot tries to move within a wall or closed door | The robot’s motion is cancelled | As expected, | pass |
| 9 | the player tries to move a robot that has been deactivated | The robot will not respond | As expected, | pass |
| 10 | Either robot is inactive when they player presses play | A popup opens displaying that both robots must be active to play | As expected, | pass |
| 11 | Player is outside of 384 px of an enemy | Enemy is inactive and will not pathfind towards a robot | As expected, | pass |
| 12 | Player is inside of 384 px of an enemy | Enemy will pathfind towards a robot | As expected, | pass |
| 13 | The enemy has above quarter of their health and is inside range mentioned above | Enemy will pathfind towards a robot | As expected, | pass |
| 14 | The enemy has below quarter of their health and is inside range mentioned above | Enemy will pathfind away from a robot with increased speed | As expected, | pass |

# 4.5 Comprehensive Invalid Input Testing (Excellence)

**Comprehensive testing needs to show testing of all areas of the program.**

Invalid tests are tests that ensure the program operates as expected regardless of the inputs made by the user. This will be the biggest of all of the testing sections.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test (include test data if necessary) | Expected Result | Actual Result | Test Result |
| 1 | Any .png game image file is missing from assets or renamed | An error message is displayed in the terminal | As expected, | pass |
| 2 | The program has two instances running and then closes both | The save file will be correct for the last instance of the program run | As expected, | pass |
| 3 | Any .dll file is missing | The program will not start, displaying an error message to the user | As expected, | pass |
| 4 | A font file is missing from assets or renamed | An error message is displayed in the terminal | As expected, | pass |
| 5 | The player tries to interact with the main menu when a popup is displayed | The main menu will not react, and the popup will stay in front | As expected, | pass |
| 6 | Any keyboard input is made during the game phase that does not relate to the keys listed in valid input testing | The game will not react | As expected, | pass |
| 7 | Save file is missing | An error message is displayed in the terminal | Program crashes | pass |
| 8 | Player leaves the program running for an extended period of time | The window remains open and program does not crash | As expected, | pass |
| 9 | Save file is modified in a way that makes it so that it modifies the length of the file | The program aborts, and an error message is displayed in the terminal | As expected, | pass |
| 10 | Any menu .png image is missing or renamed | An error message is displayed in the terminal | As expected, | pass |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
| 16 |  |  |  |  |
| 17 |  |  |  |  |
| 18 |  |  |  |  |
| 19 |  |  |  |  |
| 20 |  |  |  |  |

# Your Code

Please copy the final code from your program into the space below.

|  |
| --- |
| Final Code |
| <allcode.txt> |