# Assignment 1 Marking Criteria

Your program/code/software will be assessed against the following criteria:

* **Readability**: The ease with which your code/software can be understood by a human. Usually helped by: an abundance of clear, concise, informative source code comments; the use of naming conventions and sensible names for variables, functions and classes; a consistent and standard source code layout (achieved via standard and consistent code indentation, blank lines and the adherence of standardised documentation syntax); and producing appropriate auxiliary documentation as part of the submission.
* **Structure**: The degree to which the code has been organised into relevant blocks, files and other structures as appropriate.
* **Validity**: The severity and quantity of any logical or functional errors and the presence of appropriate tests and internal checks. Evidence of testing can be provided in auxiliary documentation.
* **Efficiency**: The extent to which the code is efficient in terms of the time taken to do some processing, the responsiveness of user interaction elements, and the demands for computation and data storage resources.
* **Functionality**: The overall functionality and usability of the software and any associated documentation.

Guidelines:

* **Comments and documentation**:
  + Include a comment at the top of each file that provides a brief description of the file, and details authorship and version.
  + Organise the code into blocks separated by blank lines and start each code block with a comment that describes what that code block does.
  + Comment out print statements that were used for testing rather than deleting them. You will want to comment them out if they do not produce useful information for the user (as essentially these just slow the program down). If you comment them out, consider providing a brief explanation if it is not obvious what these were used for. (It is common to remove many such print statements used to test code especially if unit tests or doc tests have been developed but having commented out print statement used to test code is evidence that you have tested code as you have developed it, and it is better these are there as evidence, especially if no other evidence is provided.)
  + Produce standard function documentation that describes what each function does and what any arguments are that are passed into the function and what if anything is returned.
* **Testing**:
  + If the software works as expected, then this is evidence that some testing has been done, but you are strongly advised to provide additional evidence of testing. This can be provided in auxiliary documentation.
* **Lay out**:
  + Place all import statements under the comment at the top of the file.
  + Place all functions below the import statements.
  + Where it is reasonable to do so, split code over multiple lines to restrict it to a normal page width.
  + Keep indentation aligned.
  + Have blank lines to help separate out parts of the code that do different things.
* **Structure**:
  + Generalise code into functions as appropriate. Defining and calling functions can make code a lot more concise and therefore easier to read, use, maintain and develop further.

The code is expected in two files and it should be clear in the README which file should be run by the user.

If you have attempted to add some functionality but have not got this working fully and want credit and feedback on this, then include details in the submission. Try to be clear about what is not working and what you have tried to resolve the issue. Provide also the code for the last working version you have before trying to add the feature that somehow failed.

The final mark will be based on a combination of elements from across the marking scheme. The Python source code and auxiliary documentation will be inspected and the software will be run. The following rubric will be used to assess the work and provide feedback:

* For 50 points, the program should instantiate a collection of agents that interact with the environment, and there should be both visual and textual output from running it.
* For 60 points, the agents should interact with each other, and one of the outputs should be an animation.
* For 70 points, there is expected to be a GUI and/or some way for a user to input or modify parameters without having to modify the source code.
* For 80 points, the work should be exceptionally good and this may include a significant enhancement as suggested in the practical documentation, e.g. having the agents move in a more realistic way, add predator agents, implement a fairer/more realistic way of resource sharing, limit the amount that the environment can be eaten and perhaps control have fast the edible layer regrows, model agents duplicating somehow, produce additional visualisations or statistical outputs from the model. Not much in addition is expected to gain these extra marks, but taking a step without instructions is a significant step. Even if you do not successfully implement this step, but outline what you tried and how far you got, then credit for trying to take this step can be given.

A qualitative judgement may assign a different number of points where some elements are included and other not.

The initial amount of awarded points will increase or decrease by 5 depending on the assessed quality judgement of each of the following:

* **Auxiliary documentation:** This should include a [README](https://en.wikipedia.org/wiki/README) file, a [LICENCE](https://en.wikipedia.org/wiki/Software_license) file and may also include user and developer documentation. The README should either be a simple [Text](https://en.wikipedia.org/wiki/Text_file) ASCII file, an [AsciiDoc](https://asciidoc-py.github.io/) file or a [Markdown](https://en.wikipedia.org/wiki/Markdown) file. Ideally this should provide: a contents list (detailing what all the files/directories are); and, an outline what the software is, how it can be run and what is to be expected when it is run. Any additional document might provide more details about how to run the software, what to expect when it is run, outline any ‘known issues’, outline any testing done and may outline ideas for further development. All of these details may be included in the README, but you may prefer to have multiple documents.
* **Source code comments and documentation.** Ideally, these will be as outlined in the above guidelines.
* **Code structure, the software efficiency and the model validity.** Ideally, these will be as outlined in the above guidelines.
* **Website for the software**. This is expected to link to an online repository of the source code. The validity of HTML will be tested using the W3C validator: <https://validator.w3.org/nu/>

The typical criteria below will be used to provide marks in the feedback sheet boxes and should help you gauge what is expected of you for Assignment 2. So, if you get a high mark for this assignment, but the overall feedback in the boxes is at a lower level, the specific feedback should indicate what areas to strengthen for Assignment 2.

|  |  |
| --- | --- |
| **Grade** | **Typical Criteria** |
| **High Distinction** | **Readability**: The code/software will have excellent documentation. The software source code will contain appropriate useful comments that explain what the code does. The code will be laid out clearly and consistently. The program itself may come with structured help files which may be embedded throughout the system.  **Structure**: The Python code will have a main file that is run and a separate class file for dealing with agents. It may optionally have other classes for reading and writing data files. There will be useful functions that avoid repetitive code and provide generality. Any Graphical User Interface will be clear. Any website or auxiliary documentation will be well structured.  **Validity**: The code will show an appropriate level of testing and internal checks, both to ensure the program’s validity and to prevent fatal issues during running. There will be checks to help prevent the system exiting in error (especially as a result of user providing unusual input values or clicking buttons and pressing keys not in accordance with user documentation). Auxiliary documentation may detail testing. There may be doc tests or unit tests.  **Efficiency**: The code will have close to optimal efficiency in terms of the time taken to run the program.  **Functionality**: The code will include the core functionality, enhanced with additional functionality. The software will be designed to produce informative/useful output.  Overall, the code will be of a professional release standard, or close to it. |
| **Distinction** | **Readability**: The code/software will have excellent documentation. The software source code will contain appropriate useful comments that explain what the code does. The code will be laid out clearly and consistently.  **Structure**: As for **High Distinction**.  **Validity**: The code will show an appropriate level of testing and internal checks, both to ensure the program’s validity and to prevent fatal issues during running. The software will be designed to produce informative error messages that advise users if an error is encountered.  **Efficiency**: As for **High Distinction**.  **Functionality**: The code will include the core functionality.  Overall, the code will be of a near professional release standard with only minor usability issues. |
| **Merit** | **Readability**: The software source code will contain appropriate useful comments that explain what the code does. The source code will be laid out with only minor inconsistencies. The program/software will have some user and developer documentation that helps users understand what the program does and how to run it, and that details the thought processes behind the code functionality and development.  **Structure**: Where appropriate, the code will show some thought into the development of structural units (like classes) resulting in clear code, but perhaps not really facilitating code reuse. Any Graphical User Interface and website will show thought in their structuring, with appropriate separation of functionality.  **Validity**: There will be some evidence of testing.  **Efficiency**: The code may be inefficient with no evidence that work has been done to make the code efficient.  **Functionality**: The code will include most of the core functionality, with only minor issues. |
| **Pass** | **Readability**: The code/software will have limited documentation. There may be little evidence that thought has gone into code structuring or standards. There may only be minimal inline comments. There will be an attempt to lay out the code neatly, but there may be some inconsistency. Ancillary documentation may lack detail, may be unclear in parts and may be of only limited use.  **Structure**: The Python source code may only follow broad structures associated with the coding language. Where there is an expectation of code separation, much of the code may nevertheless be in a single file or block. The website may not function well as a consequence of poor structure.  **Validity**: Some errors may be encountered at runtime without this being documented.  **Efficiency**: The code may be inefficient.  **Functionality**: There will be evidence of an attempt to include the core functionality, although numerous issues may have been encountered. |
| **Fail** | **Readability**: The code/software may have little or no documentation. Inline comments will be minimal or non-existent, or may contain major errors or misunderstandings. The code layout may be confusing and difficult to read. Any ancillary documentation may be of very limited or no use. There may be little or no attempt to articulate issues encountered in developing the code/software.  **Structure**: The code will be poorly structured and may, for example, be in a single block.  **Validity**: Many errors may be encountered at runtime without this being documented.  **Efficiency**: The code may be very inefficient.  **Functionality**: There may be little evidence of an attempt to include the core functionality. |

# Feedback Sheet

In general, your overall mark will reflect the average of the categories below, however please note that the grades for the individual components are nevertheless indicative only and the overall mark may vary from this. For example, the assessor may decide that a very nice but minor piece of code warrants a higher mark overall, despite not making a huge difference to any one of the individual categories. Please read the detailed feedback in order to understand your mark.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | High Distinction | Distinction | Merit | Pass | Fail |
| Readability |  |  |  |  |  |
| Structure |  |  |  |  |  |
| Validity |  |  |  |  |  |
| Efficiency |  |  |  |  |  |
| Functionality |  |  |  |  |  |
| Overall |  |  |  |  |  |

Feedback

Areas to especially prioritise

|  |  |
| --- | --- |
| Marker | Provisional mark |
|  |  |

*Any mark given here is provisional and subject to moderation by the School’s Board of Examiners. This is to ensure comparable marking standards for all students. In a minority of individual cases moderation can lead to either the raising or lowering of the provisional marks.*