**Testing**

Throughout the process dynamic white box testing was carried out. Due to the nature of the agile development Test Driven Development was used to ensure that at each milestone in development the program worked as expected and bugs were minimised. This was done in part by enforcing strict rules on how GitHub was used for version management. Each time a team member wanted to merge their code addition a review was requested from another team member, requiring that the other member check that the code ran, and review the code to make sure no obvious bugs were present and conventions were kept. Code could not be merged into the project until this review was given, and the merge was approved. This meant that at every point no broken code was placed into the repository.

Dynamic black box testing was carried out on the produce to ensure all parts worked as specified and the Moscow analysis criteria were met. This was done with no access to the code and it was both tested in a ‘test to pass’ and ‘test to fail’ approach to maximise the bug and problem reporting.

|  |  |
| --- | --- |
| **MOSCOW ANALYSIS** | **Testing notes** |
| **Must** |  |
| Organism life cycle | Organisms are created at first by the user. After that they need food to survive and die if their hunger reaches 0. Once the organism reaches its maximum age it dies. |
| Genetic crossover algorithm | They can mate producing an offspring of the same species and their individual attributes appear to crossover to produce the offspring attributes. |
| Organism attributes |  |
| Live edit of organisms | This is possible and editing the organisms results in them being affected while on screen. For instance, setting hunger to 0 slowly kills the organism and setting the speed increases the organisms moving speed around the map. |
| Simple 2D graphics | The graphics are good quality with 2D particle effects when food or an organism is placed into the world. The organisms do not overlap and occupy the same grid tile and when they die their sprite is replaced with meat. |
| Herbivores and natural food sources | Plant material can be added to the map and organisms will consume it, increasing their own hunger and removing the plant from the world.  When organisms die they are turned into a meat source which other organisms can eat. |
| **Should** |  |
| Weather/disease system | If the environment becomes too cold or hot, some organisms will start to die from the environment and if the temperature does not change then this can kill many the organisms on the map. |
| Advanced path finding algorithm | During testing and observance, when searching for food or a mate the organisms will avoid paths with take them over mountains or water. They will also avoid moving to the same tile as other organisms. |
| Terrain variation, e.g. grass, mountains, water |  |
| Ability to pause, speed up and slow down simulation | Implemented and seems to work. |
| **Could** |  |
| Natural disasters | Not implemented |
| Speciation (new species forming from heavily mutated organisms over time) | The species do change over time and form new organisms through reproduction. However, there is no clustering of organisms to make set populations around the map. |
| A game log with charts and text output | Not implemented |
| Sprite sheet animation | Not implemented |
| Particle effects, e.g. weather effects, running water, blood | There is blood |
| **Won’t** |  |
| 3D graphics | Not implemented |
| Scale realism | Not implemented |

Visual Studio metrics

In visual studio tools are provide to analyse a project. These analyse the maintainability, coupling, Cyclomatic complexity.

**Maintainability**

It measures how difficult the code is to fix and maintain by how long sections of code are and how much they interact with other classes and methods. The project has a maintainability score of 81 out of 100 which presents a positive green symbol on visual studio. Individual classes within the project report scores between 56 and 96. Fundamental differences between the functions of these classes explain these differences, however, any score above 50 is considered good in the visual studio maintainability metric.

**Complexity**

Cyclomatic complexity displays the number of decisions a block of code contains plus 1. This generally correlates to how easy the code is to comprehend and therefore fix or add to. Namespaces in the project range from 136 to 28. 136 is the score for the Namespace that contains the state machine and therefore warrants a high complexity score and the lowest is pathfinding, which is highly optimised resulting in a low score. Overall for the project complexity ranges and given time changes to the project would reduce the complexity score if possible. This would increase the maintainability and coherency of the code also, resulting in a more robust and better system.

**Coupling**

In the metrics, coupling is measured. This was of high importance throughout the project and effort was placed to reduce coupling where possible. Efforts were made to limit communication between namespaces however, Organism, StateMachine and Grid are highly coupled, due to their nature and the time available for the project. Given more time the coupling within these Namespaces would be reduced to improve the code base.