Change Report

Approach to Change Management

After first selecting Team Fractal's project. We had a formal meeting in which we went over and reviewed their architecture, plan and state of the game. During this meeting we divided the workload between us and gave each person doing that area of the project the power to make any calls require provided the group was informed and at the very least a small discussion was had regarding it. This distribution of control over the project ensured each change was considered and implemented to the best effect, and the frequent discussions over changes ensure our team acted as one team and not disjoint changes. Documentation updates as opposed to code updates have been handled as a team first discussing any changes we wanted to make, then modifying the documents to represent the team's decision.

Alongside the changes made we kept a changelog of any additions or modifications and what requirements they affected or if they affected architecture etc. This ensured that the document updates would be complete with our changes.

Changes from the initial code:
https://github.com/NotKieran/DRTN-Fractal/compare/Fractal_Initial...master

Alongside all of these summarised documentation changes, smaller changes have been made to documents like the Requirements. We have assembled all documentation for this assessment here in one location.

GUI Report

Updated Document
Original Document

The GUI of *Fractal's* game was one of the focal areas of development in this assessment, having been drastically restructured and heavily rewritten in places to provide more utility and visual appeal. For this reason, we also decided to entirely rewrite *Fractal's* GUI report from scratch so that we could more accurately cover the new design that we put in place, though we still took care to mention everything that *Fractal* did in their original report.

The biggest change that we made to the game's UI (which we expanded upon in our new report) saw the roboticon- and resource-markets' interfaces be integrated into the main map-screen via the use of the Overlay class that we developed for our first project, *Sabbaticoup*. This in itself necessitated the complete recreation of those interfaces to ensure that they didn't completely cover said map-screen, hence providing excuses for us to make improvements to *Fractal's* admittedly space-inefficient layouts. Condensations of stock-labels into purchase quantity indicators, reductions in the sizes of roboticons' images and reductions in the usage of table objects provided the biggest gains in useful real-estate, whereas more varied colour schemes and a transition towards using Montserrat as the interface's primary font helped the game to look far more presentable than it ever did before.

The report itself was also written slightly differently in that we chose to show how the game's new GUI satisfied *Fractal's* requirements by dropping annotative identifiers instead of describing relevant requirements outright. If a sentence in the new report refers to something that satisfies a requirement, an emboldened tag containing the ID-number of the requirement that it supposedly satisfies will instead be present in the middle of that sentence.

Methods and Plans

Updated Document
Original Document
Original Plan

There was only one significant change made to the Method and Plans document. The previous team used a tool called Trello in order to organise tasks and their individual progress; our team has used a GitHub extension called ZenHub to do this. ZenHub, like Trello, allows separation of tasks where they can be placed in different categories such as Product Backlog and Sprint Backlog. Given that our team has been using ZenHub continually and that we are familiar with it, there was no reason to acquire and learn Trello. Other tools such as Slack, mentioned in the documents, are still in use.

The plan itself was suitable to continue with and therefore did not require any changes. The subdivision of tasks and documents was appropriate for our team size and the time periods given to do them were appropriate for our work pace. This was also the case for the Assessment 4 plan.

The software engineering method used, Agile, was continued along with a scrum methodology. The frequency and planned content of scrum meetings was suitable for the project and our team. Overall the methodology and approach to team organisation was very similar to our's in previous assessments and therefore could be continued easily with no changes.

Risk Assessment

Updated Document
Original Document

As a team, we felt that all of the current risks within the risks and mitigation document still applied to our project. Therefore there wasn't any need to edit or delete any of the risks that were currently there as none of them have been made redundant.

The most substantial issue that we have had to deal with is the process of taking ownership of another team's project. As a result we thought it was necessary to document the associated risks that this process posed. Most of the risks derived from the fact that we would have a great deal of unfamiliarity with the code that we have decided to extend despite reading over it before choosing it. Also, we have to presume that all current code is bug free and as functional as described in the documentation. This, in itself, creates further risks that we have to deal with.

Summary of Changes

- Added a risk relating to requirements being inaccurately reported as being met
- Added a risk relating to code being poorly documented
- Added a risk regarding refactoring code
- Added a risk regarding insufficient testing

Testing Report

Updated Document
Original Document

The test report has been completely updated and rewritten into one document. In particular we have removed their manual unit test recording as IntelliJ is able to export JUnit tests as HTML and this is more reliable, time efficient and maintainable than manually recording test results.
The tests are all either included in the inbuilt Test suite as described in our previous assessment or clearly described in our updated Test3 document in the case of the manual tests. Any executable tests are obviously clearly repeatable. And all of our manual tests have been described in a way such that they two can be replicated by any developers who wish to.

The automatic JUnit test results can be found on the website. These tests are all essentially unit tests with a few acting more as integration tests due to the necessity of that class requiring other classes to be tested. Our methodology for these tests closely mirrored our previous assessment.

We had to undertake manual testing for classes which require a game to be running. Justified here:

Following a discussion with Fiona after realising Fractal had not made their application testable with either their included headless application or the variant we used last assessment. We decided instead to forgo the efficiency of automatic testing of certain elements because the game simply did not allow for it in that state. Therefore some elements have been tested manually, the testing methodology and results for these exceptions can be found listed bĕlow. Any class which requires the main game to be functioning, cannot be tested with the standard JUnit config.
-Team DRTN, Test 3 (Updated document linked above)

This decision was an unfortunate one to have to make however as mentioned above we have been as descriptive as possible when detail the methods employed to test these components.