Formal approaches to change management

To ensure any changes to the codebase are adequately thought out before being implemented, we have developed a system whereby any change which is not directly related to completing the game to the requirements is reviewed through our change management process. The process we used had the following elements:

- **Change request is made**
  A team member suggests a change to the codebase that they would like to see. For our project this was mainly done either in person at team meetings or on our team slack chat.
- **Change request reviewed**
  The change is reviewed by other team members to identify if the change would fit within the requirements of the project and to try to identify any potential side effects that the change would have to other functionality.
- **Request is accepted/rejected**
  The request is then either accepted, rejected, or accepted with additional changes which also need to be made to avoid any broken functionality.
- **Change assigned to developer**
  If the request was accepted, the change is assigned to a developer to be implemented
- **Implemented change is audited**
  The change as implemented is reviewed by another team member, the unit tests are run, and the whole game is tested to ensure that the change hasn't caused any unforeseen side effects.

Throughout the development process for this project our team has been using the version control system, Git. This is beneficial to change management as it provides traceability of all changes throughout the history of the project (through tools such as git-diff and git-blame). This also means we can very easily revert to a previous commit to roll back a change which accidentally breaks some piece of functionality.

For our deliverables and documentation, we created and edited them all in Google Docs, which automatically keeps track of all changes that are made. This means that all changes are traceable and we can easily see what has changed. We also coloured any text we added green, and any we took away red so we could see the overall changes to the files.

When making a new class we marked it at the beginning with a comment saying "new for assessment 3". When extending existing classes we add a comment saying either "new" or "extended" to individual methods that are new/extended.

GUI report changes

The GUI report is based on the one we inherited from Duck Related Team Name. We updated it to include new GUI elements implemented in Assessment 3. We also changed the requirement references and discussion about requirements to match our requirements. We have also introduced definitions for usability and playability and used them consistently throughout the report.

Updated deliverable

https://teamfractal.github.io/assessment3/GUI3.pdf

Risk Assessment and mitigation

The risk Assessment and mitigation document has remained the same as in Assessment 2, this is because throughout Assessment three we have had no problems using it, we have faced no problems which the risk table did not discuss and upon review were unable to think of any other risks which needed discussion in the risk table.

Testing Report

The Testing report is based on the testing report we inherited from duck related team name. Most of the changes to the report were updates for Assessment 3. However our requirements testing was based upon our own requirements so the document now links to a document which shows that.

We have also slightly updated the discussion of various testing techniques to incorporate our requirements testing as it was more precise than the previous testing.

Updated deliverable:
https://teamfractal.github.io/assessment3/Test3.pdf

Methods and Plans

Method and plans did not require many changes as we feel the method section covers our methodology sufficiently. We have updated the plan for Assessment 4 with more detail as we now have the context in which to do so. On the plan red represents the critical path.

Plan URL:

https://teamfractal.github.io/assessment3/plan.pdf