Continuous Integration Report

Tom Rosewell
Sam Redgewell
Tom Keeler
Jake Stogden
John Limb
Adham Noureldin

Methods and Approaches

Our continuous integration will be running alongside our software testing plan. As discussed in our software testing report, there are three phases of testing we plan to go through. Between these testing phases, we will be implementing our continuous integration methods to remove bugs and implement new features. Initially, no new features will be implemented into the code, only bug fixes found in the original code. For the other 2 stages, we will be implementing new features and then fixing bugs that may come as a consequence of said implementation.

Our continuous integration will have a very ‘hands-on’ approach. Our programmers will be completing a list of tasks that have been posted on our GitHub repository. This list of tasks has contributions from every member of our team, who were encouraged to add whatever they felt needed attention, and will continue to do so throughout the project. During the time we have allocated to integration in our plan, our programmers will have ‘sessions’ where they will take on tasks and iteratively attempt to implement them. During this time they will be doing mild white box testing to avoid obvious errors in the code. This is not to be confused with the white box testing done as a part of our software testing plan.

To ensure our project is delivered on time in good condition, we have also introduced risk management strategies into the continuous integration plan. The process we use to handle risks is dependent on their type.

Project risks have already been identified and discussed heavily during team meetings so they can be avoided or mitigated. An example of an avoided project risk is a possible lack of available team members during implementation. This issue was tackled using good team management, ensuring high priority tasks do not have a ‘bus factor’ less than 2. An example of a mitigated project risk could be the removal of ‘single points of failure’. One way we did this was by making sure essential files are kept on cloud storage accessible by all team members.

Product risks will have to be managed throughout the project. The identification of product risks will be done during software testing as this pipelined approach saves valuable time. Software testers can then add their findings to our ‘risk register’ where risks have their severity and likelihood rated. During our regular team meetings, these risks will need to be discussed in detail with our programmers where we can discuss avoidance and mitigation strategies. Something that will apply to all risks is ‘risk ownership’. Owners of risks will be in charge of monitoring their likelihood and severity. Since these factors will change over the course of implementation and testing, risk owners will be anyone on the software testing team as they will have a more intimate understanding of how the system works. Our programmers have been told to make sure all code is modular so that if a risk has a high likelihood of catastrophic severity, features causing that risk can be easily removed or changed.

Continuous Integration Infrastructure
We have organised the implementation of features using GitHub's 'issues'. Anytime there is a feature or bug that needs adding to the project our team members can add issues with the appropriate labels to describe the issue such as 'bug' or 'enhancement'. Team members are also able to comment on issues to communicate extra details.

We will also be taking advantage of GitHub's 'branches' feature. Branches are a form of risk management that allows each of our programmers to develop and experiment with areas of the code on their local machine without affecting the overall project. Once the subject of a given branch has been dealt with, team members can work together and merge branches to bring new areas of the code together and test whether they are ready to be implemented fully. A lot of the testing done will be automated. This is a fast and effective way to find severe errors in the code without spending the time testing ourselves. Despite this, human-controlled testing will also have to be done to find logical errors in the programming that affect things like gameplay.

The priority of these tasks will be recorded in our shared google drive in a google sheets file. These priorities are labelled as 'may/should/shall' and will affect the order in which the tasks are handled. This way if we find we are behind schedule, less important tasks can be left unfinished to allow us to work on other areas of the project and a working final product can still be delivered.

Similarly to our task priority ratings, our risk register will also be stored on the same google drive in a google sheets file. The file will contain information about risks such as likelihood, severity, owner, mitigation strategies and descriptions. The existence of this file ensures that our team can stay coordinated. Despite this, a risk register is not enough alone to ensure this. As a part of our risk management, it is integral we stay in good communication about our tasks and their risks. Because of this tasks and risks will be regularly discussed on our team's Discord server, where members can use text and voice chats to communicate their needs from any location. Discussions will include changes in priorities, likelihoods, severities, risk ownership and general talks of implementation to make sure teams are aware of each overs work and how their work affects themselves and the overall project.