Continuous Integreation

Windows 007
Cohort 3 Team 7

Christopher Eames
Stephen Lavender
Sam Leach
Daron Lepejian
Ding Lim
Hasan Majid
Joshau Marshall-Law

0.1 Initial CI Approach

Initially, our project utilised a minimalistic continuous integration strategy centered around manual verification through print statements embedded within the codebase. This approach allowed us to monitor the execution flow and verify the functionality of various game components by reviewing console outputs during development. Each code modification was followed by running the project to ensure that changes behaved as expected, helping us quickly identify and address any issues. This method was simple and resource-efficient, enabling the team to focus on game development without the overhead of setting up automated CI tools. However, it had limitations in automation, scalability, and comprehensive coverage, making it less effective as the project grew.

0.2 Evolved CI Approach

Regognising the need for a more robust CI infrastructure, we transitioned to a combined strategy that incorporates both manual gameplay testing and automated test coverage using IntelliJ. Manual testing now involves actively engaging with the game to assess functionalities, evaluate user experience, and identify bugs that might not be evident through code inspection alone. This hands-on approach ensures that features operate as intended and maintains the game's quality from a player's perspective. To complement manual testing, we integrated IntelliJ's automated test coverage tools. These tools provide insights into which parts of the codebase are exercised by tests, allowing us to identify areas that require additional testing. Automated unit tests are written and maintained to verify the correctness of individual components, facilitating early bug detection. Regular automated test runs offer continuous feedback on code changes, ensuring that new developments do not compromise existing functionalities. This combination enhances code quality, increases efficiency by reducing the time spent on manual verification, and improves the overall reliability of the project. Further details of our pipelines can be seen below alongside our complete integrated infrastructure report.

0.3 Suitability for the Project

This evolved CI approach aligns well with our project's complexity and scope. . The team's familiarity with manual debugging and IntelliJ's testing tools made the transition smooth and efficient, leveraging existing skills without requiring significant additional training. Balancing manual and automated testing supports our development pace and project timeline by preventing bottlenecks and ensuring thorough verification of code changes. Additionally, the flexibility of this CI infrastructure allows us to adapt testing strategies as the project evolves, maintaining alignment with our project needs.

0.4 Pipeline 1: Dual Testing Approach

At the core of our continuous integration infrastructure is a balanced approach between both manual and automated testing. Manual tests play a significant role in ensuring a human-centric understanding of the project, allowing us to assess complete functionality and interaction with new or redesigned features. These tests, documented in our software testing report under "Manual Tests", were conducted whenever modifications or changes were made to relevant segments of the code. While manual testing introduces the potential for human error, it has been invaluable for capturing issues that automated tests might overlook. To mitigate the risk of human error and streamline processes, we incorporated automated tests for key methods in the project. Using GitHub Actions, every commit automatically triggers a build process. This setup also executed a suite of automated tests to ensure that new changes

did not break existing functionality, addressing common pitfalls like incorrect class names or method calls. Additionally, we utilized IntelliJ IDEA to generate code coverage reports, ensuring that our test suite provided adequate coverage and that critical parts of the code were thoroughly tested. Whenever writing new functionality or features we always ensured that both critical functions and at least 7% of our project in addition to complex parts of the new code were thoroughly tested.

0.5 Pipeline 2: Hierarchical Structure

Our 2nd Pipeline was a hierarchical development process, headed by Stephen Lavender. This approach allowed for seamless integration of visual and functional elements, as Stephen's comprehensive understanding of the project enabled him to oversee multiple aspects of development while ensuring cohesion across the entire project. Key to this hierarchical structure was the collaboration between Stephen Lavender and Joshua Marshall-Law, who led the architecture team. Joshua and Stephen designed the game framework from the project brief, breaking it into modular strands that could be developed in parallel. This parallelisation allowed different teams to work simultaneously on distinct areas of the game while minimizing the risk of conflicts during integration.

0.6 Pipeline 3: Iterative Checks and References

To maintain alignment with the project's requirements, regular checks were conducted between the development teams and the architecture team. These checks ensured that the evolving game structure adhered to the initial specifications and that the implementation remained consistent with the overall design objectives. For example, changes to core game mechanics or visual elements were reviewed against the architecture charts, which provided a clear representation of system dependencies, module interactions, and overall game flow. By continuously referencing architecture charts and conducting iterative reviews, the team could identify and address potential discrepancies early in the development process. This reduced rework, prevented integration bottlenecks, and ensured that the final product met the project's goals.