**DevOps Toolchain for Godot Game Development**  
*By Team Number six*  
*Date: 4/4/2025*

**1. Introduction**

**Introduction**  
DevOps is a modern software development approach that integrates the efforts of both development and operations teams to improve collaboration and efficiency across the entire software lifecycle. It encompasses various stages, including planning, coding, building, testing, deployment, and monitoring. Whether working with an agile or waterfall methodology, the goal remains the same: to deliver a functional application with minimal manual intervention. This toolchain is focused on automating processes, ensuring smooth application releases, and providing continuous monitoring to maintain the health of the app after deployment. By using a set of free, beginner-friendly tools, we have created a basic DevOps pipeline for our Godot game project, enabling us to collaborate effectively, streamline our development workflow, and ensure a reliable, tested final product.

**2. Issue Tracking**

We track all bugs, feature requests, and game updates using two tools: **GitHub Issues** and **Trello**. GitHub Issues is tightly integrated with our code and allows us to link specific problems to code commits. This helps when tracking bugs that appear after specific changes.

Meanwhile, Trello serves as our visual task board. We organize cards under columns like “To Do,” “In Progress,” and “Done.” Each card includes a description, checklist, and deadline (if needed). This system helps us divide work evenly and stay updated on the project status.

**3. Communication & Planning**

Our team communicates primarily through **Discord**, which allows real-time discussions, screen sharing, and quick feedback. We’ve set up a few dedicated channels—for bug discussions, progress updates, and game design ideas.

For planning, we use **Trello** to outline project milestones and weekly goals. When it comes to technical tasks, **GitHub Projects** allows us to link planning directly to code commits and pull requests, which is helpful when preparing game releases.

**4. Infrastructure Automation**

Although our project is small, we implemented a bit of automation using **GitHub Actions**. This tool runs a script to automatically build our game for different platforms (Windows and Linux) whenever someone pushes code to the main branch.

This ensures that no matter who updates the project, the build process remains consistent and manual errors are reduced. Automation might seem advanced, but GitHub Actions makes it accessible—even to beginners like us.

**5. Continuous Integration (CI)**

CI means regularly integrating code into a shared repository and testing it automatically. With GitHub Actions, we have a basic CI pipeline set up. Every push triggers a new build of the game. If something fails—like a missing file or a broken script—we get notified quickly, so we can fix it before it causes more problems.

This process prevents last-minute surprises during game demos or when preparing for submission deadlines.

**6. Source Code Management**

We use **GitHub** to host and manage all our code and game assets. Version control helps us avoid conflicts, especially when multiple teammates are working on the same scene or script.

Each team member works on a separate branch. When a feature is complete, we open a pull request and review the changes before merging. We also configured .gitignore to exclude unnecessary files like temporary Godot exports and editor logs.

**7. Monitoring**

While we don’t use professional monitoring tools like New Relic or Prometheus, we created a basic feedback system using **log files** and **Discord webhooks**. Godot’s built-in logging lets us trace problems when the game crashes or behaves unexpectedly.

We also set up Discord to notify the team if a GitHub Action build fails. This helps us respond faster and know who’s available to investigate.

**8. Quality Assurance (QA)**

Our main QA method is **manual playtesting**. Each team member plays through new builds and records any bugs or gameplay issues. We track these issues in Trello or GitHub.

In addition, we started using **GDUnit3**, a unit testing framework for Godot. With this, we can test parts of our code automatically (e.g., checking if health points decrease correctly after damage). We’re still learning how to use it, but it’s been helpful for catching small bugs early.

**9. Conclusion**

Our DevOps toolchain combines essential tools and practices tailored for a student-level game development project. While we don’t use every advanced DevOps feature, our toolchain covers all the basics: version control, task management, automation, CI, QA, and communication.

Here’s a cleaned-up version of your first explanation, reworded into a clear introductory paragraph for your DevOps toolchain document:

