Quality Assurance Document

Quality Assurance Plan

1. **Coding Standards**

   One of the ways we plan to maintain high quality in our codebase is to have strict coding standards. Static typing is an available option inside the Godot editor which requires all variables to be statically typed. This will assist with code clarity as well as help prevent bugs caused by unexpected type switching.

   Another way we will use to maintain strict coding standards is by utilising a linter. GDLint is a linter that provides static analysis of GDScript code to ensure it abides to the Godot coding standards. We will use GDLint as part of our static test in our Github page to run on every commit to ensure compliance with the Godot coding standards which we will abide by.

   Lastly we will use a branch and approval system to manage our Github development. Each new development will be made into an issue and have an associated branch. This could be new features, bug fixes, or any other change in the project. When the new feature is finished a pull request will be made to review and merge the new feature. This pull request must be reviewed and approved by at least one other team member before it can be merged into main. This will help keep our codebase clean as well as helping other members of the team stay informed about different areas of the codebase.

2. **Testing**

   Testing is another way we intend to keep a high level of quality in our codebase. We will implement Godot Unit Tests (GUT) which will run on each commit on our Github to ensure our new features or changes haven't broken our systems that are already implemented. Unit testing is not very common in the game development industry but we have decided to include them regardless as a way to safeguard our state machines specifically.

   A second form of testing we will use is integration testing. This will test multiple systems together where we can assure that the new features as well as existing features interact with each other correctly and continue to do so through development. This will be the majority of our code testing.

   Lastly, we will be conducting user testing at various stages to receive feedback regarding the user experience of various portions of the game. This could include: user interface usability, knowledge base, as well as other aspects of the game like enjoyability and fairness.

Results

1. **Coding Standards**

   Throughout development we adhered very closely to the standards we set in the beginning of the project. Static typing was enabled for the entirety of development and helped us avoid using vague declared variables. We also made use of GDLint at every commit to maintain a consistent format and naming scheme in all files. Each individual team member has different coding tendencies. During implementation, tasks like naming variables, classes, etc, it forced us to maintain a fixed style.

   The branch approval system proved to be an effective way of managing development in many areas at once. This allowed the team to get up to date on all the new features and systems introduced from various branches and understand them. This was also an opportunity for suggestions or concerns to be brought up and discussed with the team. This also kept the project relatively conflict free across most merges. Even through this process, there were still features that were complicated and hard to understand by other team members. For better understandability, we ultimately decided the systems had to be refactored and rewritten so the rest of the team could understand and work with the system confidently.

2. **Testing**

   GUT was applied exclusively on the two most prominent state machines being the player state machine (living, dying, dead) and the ghost state machine (waiting, moving, possessing, attacking, stunned). This was enforced to ensure these core systems were not broken by new features or fixes. Unit testing is not commonly used in game development in favour of integration testing.

   Integration testing was extensively used throughout development. This was done by making “test scenes”, separate areas to debug and test code where specific scenarios and situations can be set up. An example of this is our test called the “many ghosts single possessable” test. This test loads a room with a single object for ghosts to possess and four ghosts. The goal of this test is to ensure that only a single ghost can possess the object at any given time.

   User testing was very important to better understand how a player is feeling or what they need in the game. Since the team has been developing the game we can leave information out that we see as intuitive without realizing we are keeping information from the player. During a phase of user testing we did not have any UI elements to tell the player what items did or how to use them. This led to confusion with the testers as they did not know what or how to use the items in the game. This brought on change where a popup would notify the player of what they had just picked up and how to use it.

3. Summary

To conclude, the plan we had set up to maintain a high level of quality in both our codebase as well as a quality user experience was successful. The strict coding standards we set at the beginning were effective at helping us maintain an organized and cohesive code base throughout development. Internal testing also helped us ensure a high quality of our systems as well as their integration with each other in the final product. Lastly, user testing helped guide development in the right direction in terms of giving the player what they want need to be successful and most importantly to have fun.

Links

Full Test Document

User Testing Spreadsheet