DM Tools Test Plan

Prepared by: SEP Team 5

Introduction

DMs, with our project, will now be able to easily generate and track details of their campaign encounters in one single location and be able to save and share their creations with others with ease. This project will specifically be focusing on alleviating the DM's burden, and thus will not focus on things like player characters or providing interactive maps to replace the in-person experience. Our main focus is rather on the most tedious part of being a DM, which is managing the combat aspect of these encounters. This translates to the features of quick-viewing a creature's information, tracking their condition, and managing turn order, as well as creating the combat encounter itself. These will be the focus of our application, which means that we will also need to provide a simple catalogue of creatures and items for the DM to use to populate such encounters.

Testing Strategy

As part of this project, a comprehensive testing mechanism and workflow is defined to carefully determine all potential defects as well as to assure the software integrity and completeness. All developers are responsible to write extensive unit tests for the features they developed. Every code change are required to be reviewed and approved by at least one project member different from the developer who took role in respective code change. The developer is responsible to choose and assign reviewers that the developer believes are suitable and capable of evaluating the said change. If the reviewer finds issues, the developer is informed to address the reviewer's concerns. After another developer approves of the changes, the product owner reviews the pull request for acceptance testing. Feedback and possible needs for revision will be provided at this time. After all issues are addressed, the product owner approves the requested code change and thus allows the pull request to update the master repository.

There will be seven non-features branches as part of this project; master and release branches [1 - 6]. The master branch of this project reflects the alpha testing stage in development. Any changes made to master repo are followed by the execution of automated tests to ensure a safe integration and adaptation with base code and other changes. The release branches document each of the 6 releases at the end of each sprint, allowing for an easily referenceable history to be kept. The specifics of the test implementations are detailed in the following section.

Alpha Testing (Unit Testing)

Definition:

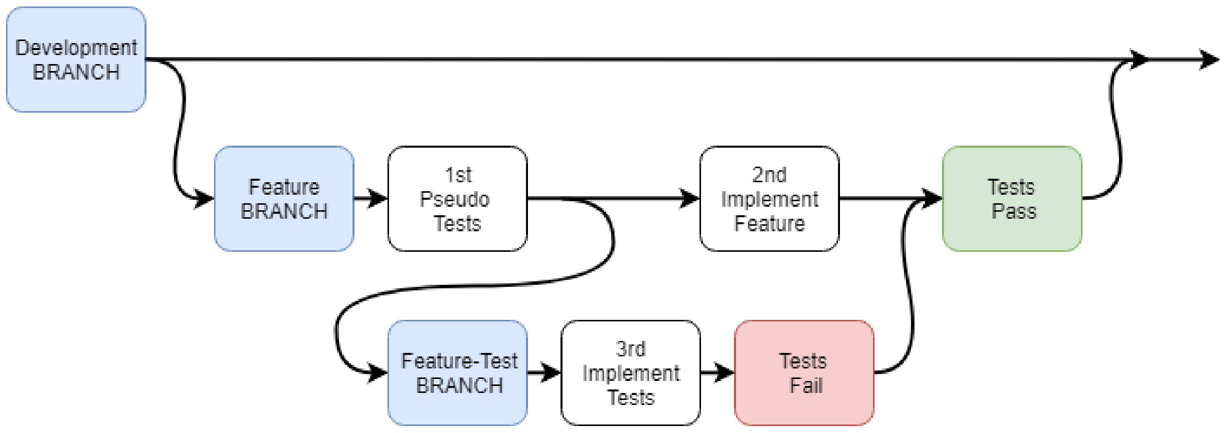
This project requires at least 80% of all code have to be covered in terms of unit testing. An automated testing tool will be used to assess the comprehensiveness of the test cases. All test cases will also be subject to code review, as detailed above, by the a developer and/or the product owner to ensure its semantic completeness (i.e. “does what we intend it to”) in addition to the correctness of functionality (i.e. “does it correctly”).

Participants:

All members on the team are responsible for implementing the unit testing for their own features.

Methodology:

Every feature has to be thoroughly tested via unit tests. Every developer who worked on a particular commit is responsible for writing respective test cases considering happy and edge cases. This project will follow the “red -> green -> refactor” approach to divide testing flow into clear steps for each developer. When a task is taken on, the developer will first a write a test scenario covering the eventual goal, which will initially fail (i.e. red) as no implementation has been done yet. Next step will be implementing the minimum functionality that will allow the unit test to pass (i.e. green). At this point, the developer will examine the implementation and the test to discover potential optimizations and edge cases, and then refactor the code accordingly. Every feature, and along with its unit testing process, will be developed in a separate branch (namely, feature branch) to ensure a safe development environment without affecting the development branch. Once the feature and its unit tests are implemented and successfully run, the feature branch will be merged with the development branch. The libraries that will be used in this process are detailed in the Tools section.



Integration Testing

Definition:

Integration testing for DM Tools aims to verify that all modules within the project work in a harmonical way. In other words, the goal for system and integration testing is to verify that the communication between low-level modules within the project work as expected while the entity they form by coming together fulfills the project requirements.

Participants:

All members on the team are responsible for implementing the integration testing for their own features.

Methodology:

For each module to be implemented, the integration tests will be written for every interaction that module has with other modules. Before implementing any module, integration tests for that module need to be written. There should be independent tests for each interaction medium the intended module is planned to have with other modules. Once two modules that needs to communicate with each other is implemented, it is expected to have written tests to pass.

Automated Regression Testing

Definition:

Regression testing is the selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still works as specified in the requirements. This process insures that all implemented features are thoroughly tested prior to acceptance tests.

Participants:

All developers will perform regression testing after each merge of the implemented feature to the development branch.

Methodology:

At the end of each merge request all unit tests are run to ensure that all the new features works with all other features. At all times we want our code coverage to be above 80%.

User Acceptance Testing

Definition:

The purpose of our acceptance test is to confirm that the system is ready for operational use, and, beyond that, achieves its mission statement of easing the burden on the DM. During acceptance test, end-users (stakeholders and customers, ideally experienced DMs) of the system compare the system to its initial requirements.

Participants:

Alic Szecsei, as the product owner, will be responsible for spearheading the user acceptance testing. He will be testing the product in his own D&D campaign as well as consulting other DM’s in the local community.

Methodology:

Once a feature is done, Alic will test out the feature on a personal level to ensure that it is up to spec for our users to use. He will be comparing this to the user story as well as providing feedback on the implementation based on his own experience. At the end of each sprint, Alic will holistically review the new application while integrating it into his current DMing experience. He will also provide it to any DMs in his network that are willing to test out the application at that stage in development. At the end of their D&D session and the acceptance testing period for that sprint, the DMs will provide feedback on what can be improved and how next features should be approached.

Control Procedures

Problem Reporting

Any bug that is found in production will be made into an issue in GitHub by the person that discovered it. Once it is brought to the team's attention, the first responder to the issue will assign labels to the issue and craft a user story for the bug. It will then be pulled in to the icebox and integrated into the next available sprint. If the issue is critical to the success to the project altogether, then it will be prioritized to the next sprint, or possibly even the current sprint depending on team availability.

Feature acceptance criteria and process

Feature acceptance will be decided by the product owner. We will have internal reviews of a feature before contacting the product owner.

Once a developer has finished a feature they make a pull request. A separate developer is responsible for pulling their feature and testing that it works, adheres to the code style, and is sufficiently addressing the features and testing requirements. Once both developers are confident that the feature is acceptable they will reach out to the product owner. The product owner will then pull down the feature and complete the acceptance testing process by merging it in to master once they approve of the changes.

Resources, Roles, and Responsibilities

All members of the development team will be involved in managing, designing, preparing, executing, and resolving unit, integration, and regression testing and any related issues for their owned features. The entire developmental team will be knowledgeable on the test environment and know how to set up and work with the environments on team systems.

Alic Szecsei (the product owner) will be responsible for compiling the test cases for acceptance testing and executing them either in his own D&D campaign or having other DMs execute these test cases and gather feedback accordingly to report to the team.

Schedules and Dependencies

Luckily, since we are a single developmental team, we do not have external dependencies on development tasks. However, consequently, if one member is missing or fails to complete their task, then the whole team will be at risk for falling behind on our releases. As a result, there is a lot of pressure for us all to be able to accomplish our work by the ends of the sprints.

Sprint 1 End: February 15th
(TDB/Tentative) Sprint 2 End: March 1st
(TBD/Tentative) Sprint 3 End: March 15th
(TBD/Tentative) Sprint 4 End: March 29th
(TBD/Tentative) Sprint 5 End: April 12th
(TBD/Tentative) Sprint 6 End: April 26th

Risks and Assumptions

A risky assumption for this project are that everyone will be able to keep up with the demand of the project for a whole semester while there are other projects and commitments flying about. There is, unfortunately, not a lot we can do about that, given that we are not actually in a workplace environment where the product is our only focus. To mitigate this, we'll be encourage team members to inform the others at the start of the sprint what their capacity will be based on their schedule, and if a sudden impactful event occurs, they'll be tasked with finding someone to pick up their extra tasks to complete the sprint on time.

Another risky assumption for this project is that our product owner will be able to pilot test the application every sprint. While his D&D group meets every week, sometimes people are sick or holidays take over, meaning that we could, in theory, have a sprint release and not receive feedback until either the middle or the end of the next sprint. Our plans to mitigate this solely revolve around the fact that our product owner is very knowledgeable about DMing and thus could provide quick wholistic feedback at the end of the sprint, even without having used it in an actual campaign.

A final risk is losing the atmosphere of discussion and falsely believing that Alic's knowledge and experiences are universal to all DMs. If we're not keeping a healthy skepticism and discussion about our product and features, we may end up making a product that solely tailors to Alic's experiences, which, while valid, aren't the only focus of the product.

Tools

Several different tools will be utilized throughout the course of this project. The first is Jest, a JavaScript testing framework "designed to ensure correctness of any JavaScript codebase." It will allow us to write easily understood tests while testing our JavaScript files, especially our

API. The second is Enzyme, another JavaScript testing framework that integrates smoothly with React components on our front end. This will allow us to implement tests for our front end.

For implementation, we will be using https://reactjs.org/ React JS for reusable component based architecture with https://redux.js.org/ Redux for managing the states of our application. We will also be using Hapi Js to allow our RESTful API to be reusable and adaptable while still being easy to start up. By using Hapi, this also allows us to use Joi JS for object validation in our API and Boom JS for http-friendly error objects

For project management, we will be using Travis CI to ensure that not only every pull request is tested before it is created, but also grant for automated deployments, should we go the deployment route.We will also be using Zenhub with GitHub Issues to add task boards, epics, estimates, and reports directly into GitHub while allowing our tasks and issues to be publicly visible.