s

Table 1: Revision History

Date	**Developer(s)**	**Change**
09/23/2016	Or	Created outline .tex
09/25/2016	Ian	Added Meeting Plan, Team Roles, and Proof of Concept Plan
09/25/2016	Mikhail	Added Communication Plan and Technology
09/26/2016	Mikhail	Added Git Workflow Plan
09/30/2016	Mikhail	Edited and Proofread Document
...	...	...

SE 3XA3: Development Plan
Rogue Reborn

Group #6, Team Rogue++

Ian Prins	prinsij
Mikhail Andrenkov	andrem5
Or Almog	almogo

Friday, September 30, 2016

The Rogue Reborn project aims to rewrite the classic video game *Rogue* in a modern programming language using contemporary software development techniques. The purpose of this document is to outline the development plan of the project. Included below are the strategies for team coordination and work partitioning, details of the technology and development process, and an overview of the project schedule including details of the requirements for the proof of concept deadline.

1 Team Meeting Plan

Team meetings will be held on a weekly basis in Thode library at 3:30 PM every Wednesday. These weekly meetings will be chaired by the team leader whose will be responsible for developing and enforcing a rough meeting agenda. Although this agenda will not be posted, it will be briefly outlined for the participants of the meeting. In addition, full meeting minutes will not be recorded; however, the meeting scribe will be responsible for recording the outcomes of the meeting discussions. These transcripts will be posted to the team Git repository. If a team member cannot attend the meeting, a brief summary of the meeting (including a reference to the Git commit) will be posted to the team Slack channel by the meeting scribe. Any changes to the meeting format, location, or time will also be posted to the team Slack channel by the meeting chair.

2 Team Communication Plan

Team communications will be distributed across several social platforms. This way, conversations can be grouped together according to the degree of formality and desired visibility. Specifically, Slack will be used to exchange informal messages for purposes such as conveying meeting times or briefly discussing topics. Team members will be required to check this service at least once per day in order to facilitate quicker response times and encourage more open communication. Next, the GitLab ITS will serve as an official means of tracking bugs, reporting issues, and announcing any other code correction requests. All other communications will be performed in person during the lab sessions as well as the weekly group meeting. Currently, there is no plan to include any other communication streams as this will most likely dilute the conversations to a point where accessibility and traceability is sacrificed.

3 Team Member Roles

As of the initial submission of this document, Mikhail is the team leader. As such, he will be responsible for chairing the meetings, allocating work across the team, and ensuring that all of the team members are up-to-date with respect to the project status and deadlines. The other team members will alternate between fulfilling the role of the meeting scribe. Outside of meetings, various team members will assume the roles of experts in the project technology area. In particular, Mikhail will be the Git and LaTeX expert, Ori will be the testing and Linux environment expert, and Ian will be the C++ and *libtcod* expert (more on this library in the Technology section). Expert roles do not constitute work allocations; rather, they assign accountability for certain portions of the project and provide a contact for internal questions related to a specific domain.

4 Git Workflow Plan

To minimize the overhead associated with creating, updating, and managing project files, the Rogue++ team will collaborate using several Git integration flows. In general, all project source files and documents associated with the main (stable) development branch will be hosted in a central GitLab repository. Any developers (or distinguished stakeholders) may clone, view, and modify this code, given that their changes do not compromise any tested functionality. Note that it is still acceptable to commit a change to this branch that is not fully integrated; however, the application must be able to successfully compile and run. In the event that a prototype demonstration is required, a new branch will be created to host any temporary changes that will not be merged back into the main branch. This way, developers are free to adapt existing source code in any manner they choose in order to showcase their progress to a stakeholder without violating the integrity of the stable branch.

To create a semantic history of the development process, labels will be used extensively throughout the course of the Rogue Reborn project. This will enable stakeholders to oversee the progress of the project more clearly and developers to gauge their own productivity with greater ease. Milestones will also be incorporated as a means of measuring the progress of the application against the goals of the stakeholders. These milestones may also be used internally within the Rogue++ team to coordinate feature implementation or debug deadlines. If done correctly, this system will allow for more efficient communication between the involved parties and improve the visibility of the entire development cycle.

5 Proof of Concept Demonstration Plan

To demonstrate the feasibity of the project, a proof of concept (PoC) will be developed. The PoC will demonstrate the following features:

- Basic dungeon generation, including rooms, corridors, and placement of gold, items, monsters, and traps
- Line of sight and pathfinding implementation
- Non-functional items and traps
- Minimum viable monster AI
- Basic movement and very simple environmental interaction (acquiring items, basic combat)

If there are no issues implementing the features above, it shall be assumed that there are no fundamental flaws with the requirements or architectural design of the project. Several major features of the project have been excluded from this demonstration (advanced item manipulation and traps, hidden passageways, complex monster AI, etc.) because the PoC is otherwise too ambitious. As long as the underlying code is well-architectured, the more sophisticated features of the application should be able to flow out of the PoC foundation.

On a positive note, it is unlikely that a straight-forward implementation of the PoC features will prove to be unusually difficult. However, implementing these features in a sufficiently extensible manner (so that they can be reused without readjustment) will undoubtedly be more challenging. It is also important to realize that the reverse-engineering process of the algorithms for various features from the original source will also incur significant effort. From a testing perspective, the Rogue++ development team is relatively new to testing frameworks. As such, writing unit and integration tests may initially take some additional time. At this point in time, the project is planned to be solely developed in a Linux environment; all of the required library dependencies have been installed and a test application of the game is successfully compiling and on all of the developers' machines.

6 Technology

The technology behind the Rogue Reborn project was selected to facilitate a productive development process and a powerful user experience. At its very core, C++ will serve as the primary programming language for this application. This decision was heavily influenced by the superior performance benefits and community support behind the language, not to mention its prevalence in the professional game developer industry. Another factor that motivated the use of C++ was its compatibility with *libtcod*: a lightweight graphics library that offers a simple interface to draw ASCII-style art and collect user input. For these reasons, the development team believes that developing a C++ project would yield the best experience from both a technical and practical perspective.

With respect to the environment of the technology, the Rogue++ team agreed to use Linux as the primary development platform (attempting to achieve cross-platform portability from the start could significantly hinder progress). Otherwise, every team member is free to use a text editor of their choice: imposing a constraint on an IDE could result in unnecessary complications and may interfere with productivity during the early coding stages. To gain confidence in the correctness and versatility of the code, the Boost Test framework will be heavily utilized to perform unit testing across the project. This library was selected on the merits of its superb documentation, simple approach to test creation, and robust assertion support. On a final note, the Rogue Reborn documentation will be generated using two tools. Specifically, all design documentation will be generated using LaTeX, while all source code documentation will be delegated to the Doxygen tool.

7 Coding Style

In any large-scale project, it is vital for all members to be on the same page. This is especially true for a software project, where every team member must be able to read, understand, and analyze each other's code. The Rogue++ project will be utilizing a modified version of Google's C++ Style Guide to format and organize the source code. Google's style guides are a professional standard and are utilized in virtually every corner of the software industry. The team has decided to implement some changes to it, however, primarily due to the nature of the project and the past programming experience of the development team. As with any style guide, what is most important is that the code is consistent and persistent, not that it conforms to one system or another. The team has decided to differ on two matters from Google's style guide:

- Opening and closing curly braces ({}) shall be on the same column. The reason for this is simple: curly braces are used to separate code blocks and to designate to whom or what a piece of code belongs. In a trivial case, this makes a negligible difference, but with multiple layers of nested for-loops, if-statements, and function definitions, opening and closing curly braces on new lines will yield code that is arguably easier to read and debug.

- Google's C++ Style Guide calls for inline comments describing classes, functions, methods, and file contents. While this is a noble goal (and is definitely necessary in most circumstances), this job has been delegated elsewhere. As mentioned before, the Rogue++ team will be using the Doxygen tool for documenting the source code structure. Doxygen will encapsulate the design-documentation sphere, stepping into a fair portion of Google's style guide. The team will still, of course, leave in-line comments explaining the functional aspects of the code.

8 Project Schedule

Over the course of a one-hour meeting, the Rogue++ team decided on an overarching timeline for the development of the project. The timeline consists of several components, tasks, and deadlines that occasionally overlap. This timeline exists as a way of benchmarking the project's completeness; a guideline, not a rulebook. It will be wise to refer back to this schedule at least once a week (most likely during the weekly meetings), update the project status accordingly, and assess any complications that could arise. The schedule will be created in the form of a Gantt chart (soon to be made available). Listed below are the deadlines that were established in the previous team meeting:

Table 2: **Project Deadlines**

Requirements Document	Sept 26	Oct 7
Milestone 1 (Rev -1)	Oct 10	Oct 14 (Break)
Technology Exploration	Oct 17	Oct 21
Test Plan + Boost Integration	Oct 24	Oct 28
Design Document	Oct 31	Nov 11
Revision 0 (Milestone 2)	Oct 21	Nov 19
Revision 1 Fixes (Milestone 3)	Nov 18	Nov 30
Final Design Document	Nov 26	Dec 7

9 Project Review