Requirements and Specifications

Bryan Ceberio-Lucas Ethan Eldridge Peter LeBlanc Phelan Vendeville

***Abstract*—This document contains the specifications of CS 205 Software Engineering's final project, an implementation of Rat-a-tat Cat. These standards and requirements will be followed by all team members. The following terms and descriptions must be clear to all members so that the system is a cohesive and comprehendable system.**

CONTENTS

I. TERMS AND DEFINITIONS

Must If a specification uses the word Must, it is mandatory that all team members follow this requirement. E.g. *The System **must** handle all possible URLs and direct the user's to an appropriate page.*

Shall If a specification uses the word Shall, then the System must respond to the specification in the detailed way. E.g. *The system **shall** perform operations in a timely manner and no operation will take more than 10 seconds*

Gantt A bar graph used to visualize a project schedule

Glow To glow is to surround an object with a faint highlight that indicates that the User may interact with this object.

State The System's internal state is kept using a Stack of strings that indicate the current and next state of the System, this collection is refered to as the State and can be pushed, popped, and peeked.

Magic Numbers Hard-coded numerical constants

II. INTRODUCTION

Phelans awesome introduction goes here

III. INTRODUCTION

IV. SCOPE AND PURPOSE

A. Scope

What is the scope of our project

B. Purpose

This is where the purpose of our project goes

V. FUNCTIONAL REQUIREMENTS

The functional requirements of this project are specified by the Coding Standards in §V-A, Version Control standards in §V-B, directory and game Architecture in §V-C, Artificial Intelligence logical overview in §V-D, and Database Design images and naming conventions in §V-E.

A. Coding Standards

The following standards must be followed by all team members. By defining these standards all code will be readable for all members, and no discrepencies between conventions will occur. Each team member is responsible for keeping to these standards, and submission of code not keeping to these standards will come under review and the format shall be adjusted accordingly.

Naming conventions

- Variable names must be camelcase, descriptive, and self documenting
- Class names must begin with a capital letter and use camelcase
- Database table names must begin with a capital letter and use camelcase
- Database table names should be short, one word where ever possible
- Directories must be lowercase and without spaces
- File names must be lowercase and without spaces
- File names for card images must be the value of the card, or 10-12 for power cards.
- All images should end in .png and be of that format
- CSS class names must be self-documenting
- CSS class names must be camel case
- Constants in any form must be all uppercase with underscores between natural breaks
- Git tagging must follow the convention of version_x.y, x must be the major release number, y the minor release number
- The team leaders repository should be refered to as mainline during remote declaration

Commenting Conventions

At the beginning of each function or class there must be a comment section within triple qoutes defining the following:

- Description of function or class
- A list of parameters and types of each
- A brief description of the return type of the function

At the top of each code file there must be a comment section with the following information:

- A description of the file's purpose and intent
- A list of the functions or classes defined within the file
- The date the file was made
- The date of the most recent revision
- A list of authors or modifiers of the file.

Within HTML each ending $< div >$ should have a comment indicating the id of the opening tag. CSS comments should be used to partition style sheet files into managable and well ordered blocks of style. It must be easy to determine which content is affected by the style by simpling reading through the comments.

General Conventions and Guidelines

- Conditional statements that involve more than a single variable must use parenthesis
- All sensitive information should be passed through posting whenever possible
- HTML/CSS should pass validation tests and be well formed and self documenting
- Global Variables should only be used when neccesary
- Magic Numbers should be avoided whenever possible
- Formal specifications should be made available using the shared Google Drive or through the mainline Git repository

B. Version Control

The version control used to maintain the source code for this System is Git. The following standards must be followed by all team members in order to maintain proper source code management.

- Git commits must be descriptive and verbose
- When merging feature and component branches to dev or mainline the option –no-ff must be used
- The Git tagging system must be used to maintain stable release checkpoints
- The master branch of the mainline repository must be functional
- Rebasing commit history is forbidden if the history has been pushed to a remote repository
- A team member resolving merge conflicts must ensure the merge is agreeable to both their and the incoming code
- All members must have their global config setup with email and name for source code tracking purposes

C. Architecture and Structure

The System shall use the Google App Engine (GAE) and Jinja templating systems to function.

The model-view-controller paradigm will be implemented, in this instance the model will be the database backend from GAE. The view will be comprised of Jinja templates, Javascript/JQuery, and CSS. The python files used by the GAE will handle all control information and dictate the flow control of the System. The project must be organized, and the directory structure will be as follows:

- `python/`
- `config/`
- `templates/`
- `css/`
- `scripts/`
- `images/`
- `userimages/`
- `sounds/`

These directory names are self explanatory besides the difference between `images` and `userimages`. The `userimages` directory is a location where user uploaded images may be stored, this directory is kept seperate for security purposes.

Each URL handled by the GAE framework and our configuration files is mapped to a python controller in a many to one relationship. The URL to python controller mapping is defined as follows:

URL	Handler Name
`/`	MasterControlProgram
`/scores`	ScoresHandler
`/game`	GameHandler
`/playerinfo`	PlayerInfoHandler
`/characterchoice`	CharacterHandler
`/difficulty`	DifficultyHandler

D. Artificial Intelligence

E. Database Design

VI. Non-Functional Requirements

A. User Interface

pretty pictures and descriptions galore

B. Game Play

This is where the storyboarding stuff goes

C. Character Design and Concept Art

D. Timeline and Delivery

This is where timeline and due dates go as well as what has to go into each part

VII. Test Cases

VIII. Summary

Overall summary