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

Knock The button press that determines a round is over and the cards should be overturned when gameplay returns to the user who knocked.

II. INTRODUCTION

Phelans awesome introduction goes here

III. SCOPE AND PURPOSE

A. Scope

What is the scope of our project

B. Purpose

This is where the purpose of our project goes

IV. FUNCTIONAL REQUIREMENTS

The functional requirements of this project are specified by the Coding Standards in §IV-A, Version Control standards in §IV-B, directory and game Architecture in §IV-C, Artificial Intelligence logical overview in §IV-D, and Database Design images and naming conventions in §IV-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
- Database field names must be camelcase, beginning with a lowercase letter except for foreign keys
- Foreign key fields are prefixed by the foreign tables name, and therefore begin with uppercase letters
- 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

The artificial intelligence (AI) within the System is stochastic in nature. A difficulty modifier passed via the games start up parameters influences the reliability of the AI's memory via a decay rate. Each level of difficutly effects the decay rate of the memory as well as the baseline of remembrance. The AI is not all knowing, and keeps estimates of both its and the opponents cards. Using these estimates a strategy can be used to determine when to knock and when to use power cards. For each of the four cards in the AI's hand a rate of memory decay is kept. This value determines how well the AI remembers the value of the card and is decremented each round according to the decay rate value stored within the AI.

Swap Strategy Using an internal estimate, the AI selects it's highest known card and switches it with the players lowest known card. The AI can determine what the players cards are based on if the Player has drawn from the discard pile or swapped with one of the AI's known cards. In the case of no cards being known – whether in selection of its own highest card or the users – a random card is chose.

Peek Strategy The AI will always select a card it does not know to peek. If all the values are known, the AI will select the card with the lowest decayed value, and that value will be reset to 1.

Knocking The AI will knock when it has reasonable confidence that its internal estimate of its own score is higher than the players.

The AI will also keep track of internal statistics to send to the database later. This is primarily a logging function and is an optional part of the AI implementation.

E. Database Design

The database backend is composed of two tables: Players and Games. The field names are shown in figure 1

V. 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

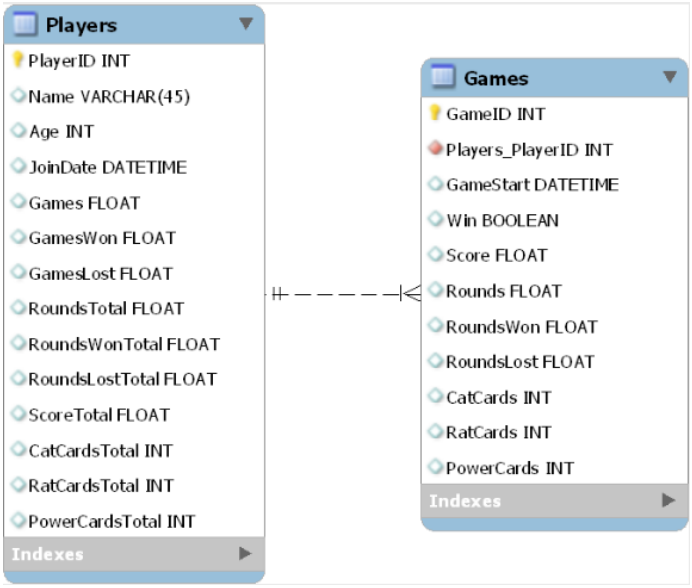

Figure 1. Database Schema of the Players and Games tables.

VI. Test Cases

VII. Summary

Overall summary