Moonwake Code Inspection Report

Developers: Cedric Fahey, Matt Virgin, David DiFrumolo, Tristan Zippart, Landon Thibodeau.

Figure 1: Moonwake Development Logo

Product: M^3 A front end for Frog God Games' Creature Creator

07 March 23

Purpose of This Assignment

- To achieve group consensus on developing or adopting practical coding and commenting conventions.
- To learn how to perform code inspections among "egoless" team members.
- To appreciate how error correction costs can be significantly reduced by discovery during the coding phase rather than later during testing or after installation.

Required Activities

- Define project-specific commenting and coding conventions (or adopt/modify other popular conventions).
- Compile and build all code without fatal errors and with only minimal warnings.
- Inspect together, as a full team, all significant pieces of code written to date. (Code inspection meetings may be virtual rather than physical, although this may be more difficult.) You will need to take copious notes during inspection meetings in order to fulfill the content requirements of the Code Inspection Report. Understand what information you will need to record before each meeting.
- The CIR should document that a thorough Code Inspection has been completed by the team. Inspection reveals flaws and inconsistencies, but it does not necessarily fix them.

M[3]
Code Inspection Report

Table of Contents

1. Introduction

This is a capstone project for Frog God Games, in partial fulfillment of the Computer Science BS degree for the University of Maine. Frog God Games are accomplished publishers of role-playing games, adventures, and add-ons for both modern and legacy gaming systems (Frog God Games Staff 2020). The company requires a front end for its database of creatures, M3. The front end of M3 will allow users to search, filter, and create creatures. The main objective of this application is to provide users with a way to access the database and utilize it effectively for its game. The product also enables Frog God Games to bolster their sales by introducing users to its website. Moonwake Development is a UMaine capstone team consisting of Cedric Fahey, Tristan Zippert, David DiFrumolo, Matthew Virgin, and Landon Thibodeau.

1.1 Purpose of This Document

The purpose of this document is to outline the status of the code that Moonwake Development has created. This includes the information gathered during code inspection. This document will contain current bug-tracking, our static analysis, existing errors, violations of development standards, diversions from our common style, and our other programming conventions. The outcome of our Source code review as well as style and our style checker will be included as well. Modules implemented and inspected, and the defects found with them will be listed also. If a developer were to be put on the project now this is the document they should reference.

Overview:

- Bug-tracking
- Code inspections are a form of static analysis, code review often without running the code. Find errors, violations of development standards, diversions from our common style and programming conventions
- Checkstyle
- PMD: Source code review
- Style checking: Webstorm built-in functionality, IntelliJ, Database tools, and SQL for Webstorm
- Code Inspection Progress
- Modules Inspected
- Defects

References

1. Moonwake Development (2022). System Requirements Specification. Retrieved October 20, 2022.
2. Moonwake Development (2022). System Design Document. Retrieved November 30, 2022.

1.2 Coding and Commenting Conventions

Moonwake Development opted for an object-oriented approach to the project through Typescript. Object-oriented design serves as the foundation of our coding conventions, but we also deviate when needed if another design style suits our needs more for a different part of the project. For example, in the back end of the project, there is a connection object that serves as the link to Frog God Games' database that is referenced as needed in other files. Front-end code deviates somewhat from Object Oriented design practices, as such a style is not as necessary for website design. Whitespace and blank lines are used to make the code more readable. If a comment is in-line, it will be indented to separate from the actual code.

Moonwake Development created its own commenting conventions. In general, every file should have a header explaining the file's purpose. The only case where this isn't true is where the file is short enough and the file's name is clear enough that its purpose is self-explanatory. Within each file, important blocks of code should be commented on. Any piece of code that can be even slightly confusing shall be sufficiently commented on to explain its purpose. The only cases where comments are not required are those where the code is self-explanatory to a novice-level programmer.

1.3 Defect Checklist

Figure 2 below is a comprehensive tabular checklist of all of the defects that the Moonwake team found during the inspection process. The defects were labeled as being a part of one or many categories. The categories are listed following: “Coding Conventions”, “Logic Errors”, “Security Oversights”, and “Commenting” A “X” in the column conveys that the type of defect was found in the part of the system, named in the leftmost row.

Figure 2: Defect Checklist

File	Coding Conventions	Logic Errors	Security Oversights	Commenting
models/index.js	**X**		**X**	**X**
Husky pre-commit		**X**		
creature-db-modal.tsx	**X**			**X**
creature-database.tsx	**X**	**X**		**X**
routes.test.js	**X**	**X**		**X**
fgg-front-end/server.ts	**X**	**X**	**X**	**X**
fgg-front-end/src/controllers/google-controller.ts	**X**			
fgg-front-end/src/components/Home-Demo.tsx		**X**		**X**
fgg-back-end/src/GASYMO_GAMESYSTEM_MONSTER Associations		**X**		**X**
fgg-back-end/src/GASYMO_GAMESYSTEM_MONSTER Listings		**X**		**X**
fgg-back-end/src/GASYMO_GAMESYSTEM_MONSTER Draw.io				**X**
fgg-back-end/srs/GASYMO_ARMOR_RATING		**X**		
Index.txt				**X**

fgg-front-end/src/pages/creature-creator.tsx		**X**	**X**	
fgg-front-end/src/pages/search_and_filter.tsx			**X**	**X**
fgg-back-end/srs/procedures	**X**		**X**	**X**

2. Code Inspection Process

Code inspection is a systematic and structured process of reviewing and analyzing source code to identify potential defects, errors, and areas for improvement. The goal of a code inspection is to improve the quality and reliability of software by identifying coding standards violations, logic errors, security vulnerabilities, and other issues that could negatively impact the software's performance or functionality. Code inspections are typically performed by a team of experienced software engineers and testers, who examine the code against established coding standards, best practices, and common programming errors. The findings and recommendations from the code inspection are documented in a report which serves as a valuable tool for improving the software and enhancing its quality.

Moonwake development conducted a code inspection while several features were still under development. Despite this, the application currently includes a display of all creatures in the database with their associated stat block, and the creature creator and filtering features have the majority of their development work completed. The application is capable of accessing and processing connections to the database, and delivering requested information to the user interface. Additionally, Moonwake is currently developing user authentication and is working on refining the styling of the information returned from the database. Work is also being done on making the connection to the database more efficient.

2.1 Description

Moonwake Development followed a modified version of a Round Robin Review during the process of the code inspection. An ideal Round Robin Review gives each participant an equal opportunity to examine and review each piece of code presented by the review leader. Moonwake Development assigned an initial review leader, recorder, and reviewers which rotated after each item. On each topic item, the review leader presented and discussed the code and features that they worked on while the recorder wrote down comments and notes. Once the review leader was finished with their presentation, the next leader presented their work. When it was time for the recorder to become the review leader, a new recorder took their role for that topic item. Each reviewer had time to discuss questions and comments during the review leader's presentation.

To provide structure to the review, each review leader was required to answer eight questions regarding their work. These questions included the current state of the code, past and current work, the best and weakest modular units they worked on, how the planned module interface compared to the actual, trade offs faced during development, and any defects encountered. The review leader's answers provided valuable insights and helped to identify potential issues and areas for improvement in the codebase.

Overall, the Round Robin Review process used by Moonwake Development was an effective approach to conducting a thorough code inspection and identifying areas for improvement in the codebase. It allowed each participant to contribute their expertise and provided a structured framework for evaluating the code. By answering the eight questions, the review leaders provided a clear and concise summary of their work, which helped to streamline the review process and ensure that all important aspects of the code were evaluated.

2.2 Impressions of the Process

The first Round Robin was a good start for the group. It was a good view of the landscape of our project and the hills we still have to climb. The structure of the round robin was logical, with some people bowed out of their position another member filled the gap for the turn. It revealed to us the highlights of the project and seemingly the lowlights became a little brighter to us.

The product's search and filter right now is the worst unit due to its build percentage. They understandably can have the most flaws as a major part of the build is not finished. The best modular unit is the Procedures which are near completion (98%) and have few flaws. Associations are not far behind and also have few flaws.

2.3 Inspection Meetings

The First Code inspection meeting was held Sunday, February 19th, 2023. The meeting was held in an online environment of Discord, in our Capstone channel. Cedric and Tristan acted as the moderators, while Matt and David acted as the scribes. Tristan and Cedric showcased the back-end work completed so far. Matt, David, and Landon showcased the front end/UI work completed so far.

3. Modules Inspected

Figure 3 below describes the modules of our code that we have inspected, as well as their current state and projected completion date. Below that, there will be a more detailed description of how each module differs from the initial plan set down in the SRS/SDD.

Figure 3 (Modules not implemented)

Name of Module	Percentage of Completion	Functionality of Module	Projected Date of Completion
Search Functionality	30%	This module will allow users to search for creatures in the library by name.	3/19/23
Filter Functionality	30%	This will allow users to filter parts of the data by block. Filters will be stackable.	3/19/23
Creature Creator	65%	This will allow users to create their own creatures following the FGG Model.	3/18/23
Routes	75%	This allows for the front end to locate the models.	3/19/23
Associations	70%	Allows for data from different tables to be used together. This is necessary for Filter.	3/12/23
Procedures	98%	Procedures for routes are near completion as what's currently needed for the connection to the backend is ready to go, however, future functionality will need to be added.	3/12/23
Log In	50%	This will allow users to log into an account associated with the system.	3/16/23
Export of .pdf	10%	Some code exists for exporting to a PDF, however it's not currently hooked up to the user interface and is not currently working.	3/19/23
Database Display	50%	Displays pagination for database connection displays basic popup modal when a creature is selected. Excel like display is in progress	3/19/23

Creature Database Display:

Displays the creatures currently in the database and allows users to select and print out creatures they like by a pdf. This uses the connection to the database to get and display every single creation to the user, along with the associated information including stat blocks and the company that created it. It displays this information in a user-friendly way, allowing the user to easily browse through interesting creatures that they might want to use for their next game.

Models:

Models are what allow the Oracle Database Tables to be readable in JavaScript (through Sequelize) and convert the data to JSON. The models are in every part of the database, for every single table. It's for every component that needs information from the database, except for the external login system.

Routes:

Routes are what allow the front end to locate the models for the database. The routes are a part of every module except for the login module. Routes are in every part of the database, they are necessary for every table. It has to do with every component that needs information from the database, that is all modules except for the external login system.

Associations:

Associations are what allow for the front end to pull parts of different tables in conjunction. To have all of the information of one creature 18 tables need to be accessed. This is fundamental for creature search and filter modules. It does not appear in the SDD. It was added because we needed to implement search and filter. It was not added before because we were under the impression that all of the associations were completed for us 10 Nov. It was not until 6 December we had access to the database and in January that we found there were 0 associations. The process was further hindered due to there being absolutely no documentation for the pre-existing database.

Procedures:

Procedures are what allow for the alteration of the database by users via the front end. The procedures make sure that the needed tables and associated information is created before user data is added. This makes sure that there are no blank or error values when data is inserted into the database.

Creature Creator:

The creature creator allows the user to conveniently and intuitively create a new creature based on information provided by the database. Once a user creates a creature, that creature is added to that user's private library, and upon review added to

the public library. The database provides available values to the creature creator that guide a user through the process of creating a new creature.

Creature Search and Filter:

Creature Search and Filter allows a user to search each library and tools that allow the user to filter the library through narrow search criteria. The search and filter tool allows a user to filter their search by selecting ranges of values from categories also found in the creature creator.

Log in:

- The log in system will allow users to log in with a google account or an admin account (assuming they know the username and password - meant for the client only). This module matches the Log In System design component of the SDD. However, the scope of log-in has been greatly reduced from the SDD. There were different levels of users planned, but in actuality, there will only be google users and the client log-ins. Because the initial plan would require a restructuring of the database, Moonwake development opted for a simpler implementation to save time.

4. Defects

Figure 4 below lists our modules alongside their defects and the categories those defects fall under.

Figure 4: Modules and Defects

Module	Defect + Description	Category
Log-In System	-Does not store user info -Back end files not in correct folder	-Correctness -Coding Convention
Creature Creator	- Values tied to Armor class confused with Armor rating - Display for Skill value displays wrong value - Improve Commenting	- Correctness - Commenting
Connection	-a class that takes the API URL for the front end to fetch data from, however, security concern is that the	-Security Concern

	URL can be viewed from web development consoles easily.	
GASYMO Associations	-no commenting -Some associations that GASYMO_GAMESYSTEM_MONSTER feed into are not listed properly.	-commenting -logic error
GASYMO Listings	-Some associations have “null” pulled. This should not be pulled.	-commenting -logic error
GASYMO Draw.io	This documentation is discolored and there is no explanation for the colors.	-commenting
ArmorDisplay	-armor class is displayed instead of armor rating in the front end. I think that this is because of the back end GASYMO_GAMESYSTEM_MONSTER_ARMOR_RATING table.	-logic error
Husky pre-commit	-Husky pre-commit is not setup correctly and constantly errors out when committing code	-Correctness
Search	-Lack of user input checking -Doesn’t pull from the database -Lacks comments	-User friendliness -Correctness -Commenting

Appendix A - Coding and Commenting Conventions

Coding Conventions:

- Figure 5 below shows how the project split into front and back end, each with its own separate folder

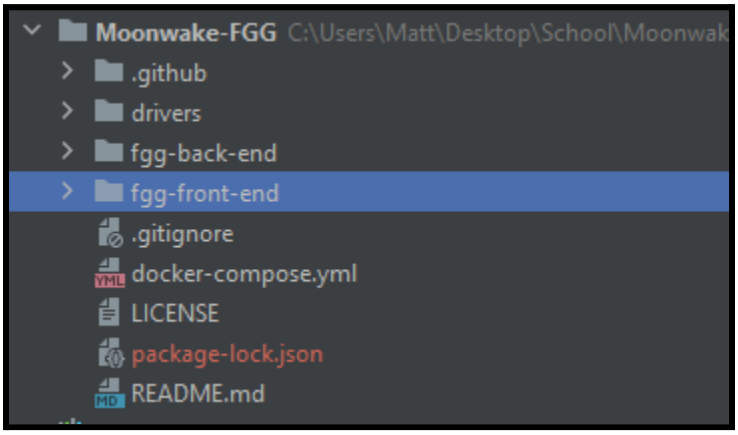

Figure 5: Folder Setup

  - All code pertaining to website design should go in the fgg-front-end folder, and all code pertaining to the database and connections to it should go in the fgg-back-end folder
- Figure 6 below shows an example of Object Oriented Design. Where the paradigm makes sense, the project should utilize Object Oriented Design Principles

```
export class Connection{
    Matthew Virgin
    private baseURL: string;

    5+ usages  Matthew Virgin
    constructor(baseURL: string) {
        this.baseURL = baseURL
    }
    5+ usages  Matthew Virgin +1
    public async getData(): Promise<Data>{
        try {
            const res = await axios.get(this.baseURL);
            return res.data;
        }catch(error){
            console.error(error)
            return {}
        }
    }
}
```

Figure 6: Connection.ts Connection Class

Commenting Conventions:

- Figure 7 below demonstrates an example of a header comment. All important files should have a header comment explaining their purpose:

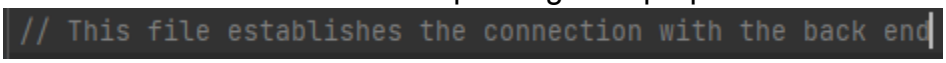

Figure 7: Connection.ts header comment

- Figure 8 below illustrates an example of a commented function. Important code segments and functions should be commented to explain their purpose:

Figure 8: Home-Demo.tsx function

- In-line comments should be indented to keep both code and comments readable
  - Example:
    - console.log(data) // prints data to ensure accuracy

Appendix B – Agreement Between Customer and Contractor

By signing this document you are agreeing that all of the content above is accurate. You agree that the code has been inspected.

- In the future, if there are any updates or changes to this document that are agreed upon by the team the following procedure will take place. A review will be held with the client and the team members with the updated document. If, after the review, both parties agree upon the changes then the document must be resigned by each member before official use.

PRINT SIGNATURE DATE

Customer:

Edwin Nagy _____ X _____ Date 8 Mar 2023

Comments: Feedback provided to team via Discord. By signing, I agree that I read the material and that to the best of my knowledge it is accurate. There are many things I do not understand and do not feel that I need to. I do believe that the team inspected the code.

Team:

Cedric Fahey _____ x Cedric Fahey _____ Date 7Mar23

David DiFumolo _____ x David DiFumolo _____ Date 7Mar23

Landon Thibodeau _____ x Landon Thibodeau _____ Date 7Mar23

Tristan Zippert _____ x Tristan Zippert _____ Date 7Mar23

Matthew Virgin _____ x Matthew Virgin _____ Date 7Mar23

Appendix C – Team Review Sign-off

By signing this document you are agreeing that all of the content above is accurate. You agree that the code inspection is complete. No member wishes to alter or change how it currently exists and there are no major points of contention within the team.

PRINT	SIGNATURE		DATE
X	x Cedric Fahey	Date	**3 / 7 / 23**
Comments: N/A			
X David DiFrumolo	x David DiFrumolo	Date	**3 / 7 / 23**
Comments: N/A			
X Landon Thibodeau	x Landon Thibodeau	Date	**3 / 7 / 23**
Comments: N/A			
X Tristan Zippert	x Tristan Zippert	Date	**3 / 7 / 23**
Comments: N/A			
X Matthew Virgin	x Matthew Virgin	Date	**3 / 7 / 23**
Comments: N/A			

Appendix D – Document Contributions

Landon Thibodeau - 2, 2.1
David DiFrumolo - Cover, Sections 2.1, 2.2 2.3, Appendices 2,3
Tristan Zippert - Sections 1.3
Cedric Lahey - Cover, Sections 1.1, 3
Matt Virgin - Sections 1.0, 1.2, 1.3