COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Salesforce

Tylor Bray
Seth Motta
Bryson Smith

Revised December 3, 2024

CSCI 370 Fall 2024

Prof. Donna Bodeau

Table 1. Revision history

Revision	Date	Comments
New	8/29/2024	Requirements Draft
Design	9/13/2024	Wrote design document, added team profiles, updated requirements section
Quality & Ethics	10/20/2024	Added quality testing and ethics sections
Results	11/10/2024	Added results and lessons learned sections; these sections reflect the status of the project today. These were updated as things were finished. Made updates to software quality, testing, and ethics sections.
Acknowledgments and test results	11/21/2024	Added Acknowledgements section, test results table, and revised other sections.
Feedback Revisions	12/3/2024	Made updates and revisions based on feedback from the paper swap. Rubrics included at the bottom.

Table of Contents

I. Introduction

The XIV Rotation Optimizer aims to improve how players of the popular massively multiplayer online game Final Fantasy XIV (FFXIV) optimize their in-game job rotations. The project envisions an advanced, intelligent tool that automates the process of determining the optimal sequence of actions (rotations) for various jobs, considering parameters such as total duration, available actions, and action speed.

Currently, players rely on manual theory-crafting, often using multi-page spreadsheets to compare different rotations. This process is time-consuming and lacks the precision that modern optimization algorithms can offer. The XIV Rotation Optimizer leverages Monte Carlo Tree Search, a heuristic search algorithm, to perform effective optimizations.

The ultimate vision for the project is to create a website that finds the most effective rotation. The tool will be benchmarked and compared to different optimization algorithms to evaluate different approaches. By doing so, the XIV Rotation Optimizer will provide a robust solution that empowers players to achieve peak performance with minimal manual effort, and Matt will win his raids.

Matt Buland, a lead software engineer at Salesforce, is our client for this project. Matt is an avid player of Final Fantasy, which he plays at a high level. Playing at this high level requires highly precise gameplay wherein all party members need to be doing nearly optimal damage for the entirety of fights that can last up to 15 minutes. This is understandably extremely mechanically difficult, so Matt hopes to take away some of the mechanical complexity by ensuring that the rotations used by himself and his party members are as optimal as possible. A previous field session group began to address this problem by creating the XIV Potency Calculator, which can be used to construct rotations and check their potency. Our project, the XIV Rotation Optimizer, further Matt's goals by expanding upon the existing calculator with optimization functionality.

II. Functional Requirements

The following list details the functional requirements for our project. Overall, the optimizer requires modifications to the existing code and data, as well as the specification of its input variables and functional output.

1. Takes parameters including job, duration of rotation, optionally a baseline user-constructed rotation that the algorithm can use as a starting point.
2. Produces a valid rotation as output
    a. Obviously, invalid rotations are of no use to us, as Matt cannot execute them in-game. The optimizer must produce valid rotations in order for Matt's goals to be realized.
3. Refactor existing Javascript codebase to Typescript
    a. The previous team encountered several issues regarding type safety that proved extremely difficult to diagnose, and one of their lessons learned was that Typescript would likely have solved these issues before they had to spend time fixing them. Thus, our client decided to have us refactor the entire existing codebase to Typescript to capitalize on its static typing, enhanced IDE support, and improved code quality.
4. Update the jobs data
    a. The game underwent a major content expansion in the time between the last group finalizing their portion of the project and our group beginning work on the project. Thus, the project's files storing data about jobs (in-game character classes) need to be updated to reflect the changes made in the expansion.

III. Non-Functional Requirements

The non-functional requirements for this project include the final UI design, the expected output, and additional features, along with testing following bug fixes.

1. Runs in the browser
2. Runs in less than 15 minutes
3. Produces an optimal or close to optimal rotation
4. Compare the output of the algorithm to benchmarks
5. Cover edge cases

IV. Risks

There are several risks associated with developing our optimizer, particularly regarding trade-offs to ensure its viability. These include adapting to external changes, the need to learn new technologies, and various unknown factors that could impact future functionality and testing.

1. The calculator built by the previous field session group had many bugs, this led to both large amounts of reactive work and incorrect results from our optimizer.
2. The time complexity of finding optimal is very high, we make the trade off between time, compute power, or optimality.
3. The game could update, rendering the project non-functional.
4. Finding all edge cases may be very complex, if edge cases remain in the calculator then it will be very difficult to create optimal outputs. Only Seth, due to his knowledge of FFXIV, will be able to easily identify these edge cases (but not all of them).
5. LLM Benchmarks may be hard to create due to issues with creating valid rotations with an LLM.
6. There was an unknown amount of time for us to become accustomed with TS and refactor the codebase.

V. Definition of Done

The definition of done for this project includes the integration of the optimizer, comprehensive testing, and an initial release, with a potential stretch goal of deploying the website to production for wider feedback.

1. Add a new tab to the website created by the previous group for the optimizer that accepts input parameters and produces valid output rotation.
2. Tests: comparison with existing rotation data, user testing done by Matt and his friends.
3. Basic delivery via Github, with a stretch goal of putting the website into production to get more feedback, and ease of use.

Each of these are complete with the exception of the stretch goal, the website is not in production.

VI. System Architecture

The architecture of this solution leverages a conventional yet robust web application framework, meticulously designed for scalability and efficiency. The foundation of the application is already established utilizing Salesforce's cutting-edge Lightning Web Runtime (LWR), ensuring seamless integration within the Salesforce ecosystem while maximizing responsiveness and flexibility. Moving forward, continued to capitalize on this advanced runtime environment to maintain consistency and streamline the development process.

For the development and testing phases, our team locally hosted the web app via the command line interface. A stretch goal for our deployment strategy includes migrating the application to a full production environment. In this scenario, we would leverage a standard Amazon EC2 instance, equipped with Nginx as the web server to manage traffic and handle static assets efficiently, alongside PM2 for enhanced process management and system reliability. Furthermore, integrating CI/CD pipelines into this architecture would automate deployment workflows, ensuring rapid and reliable updates to production with minimal downtime.

As part of our long-term vision, an additional stretch goal involves the strategic use of AWS Lambda functions to offload computationally intensive tasks, such as the optimization logic, to the cloud. This would not only reduce latency but also improve the overall performance and scalability of the web application by leveraging serverless architecture for on-demand, event-driven computation.

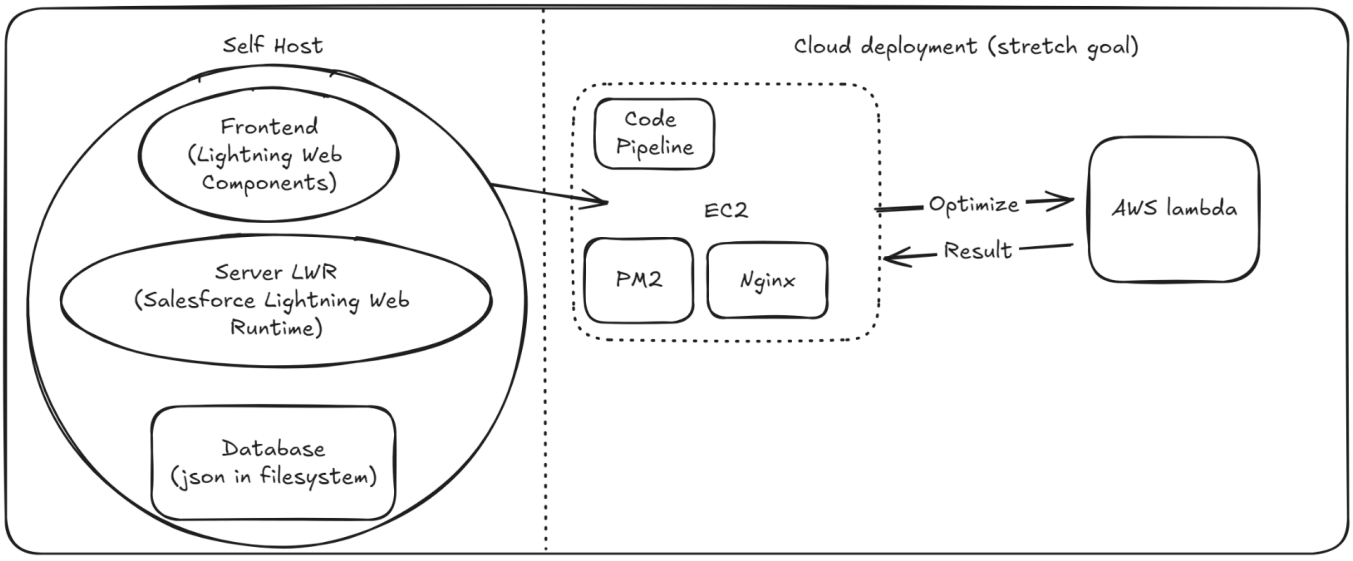

Figure 1. Visualization of application architecture

A decision graph optimization approach works by building a graph of actions, and then testing the PPS using the pre-built calculator. The graph itself will be far too large to search in its entirety, as seen in Figure 2. Thus, we settled on a heuristic to make the problem feasible to solve. The algorithm used to build and search this tree for optimal rotations relies on a tradeoff of the deviance from optimal, and computation time / power.

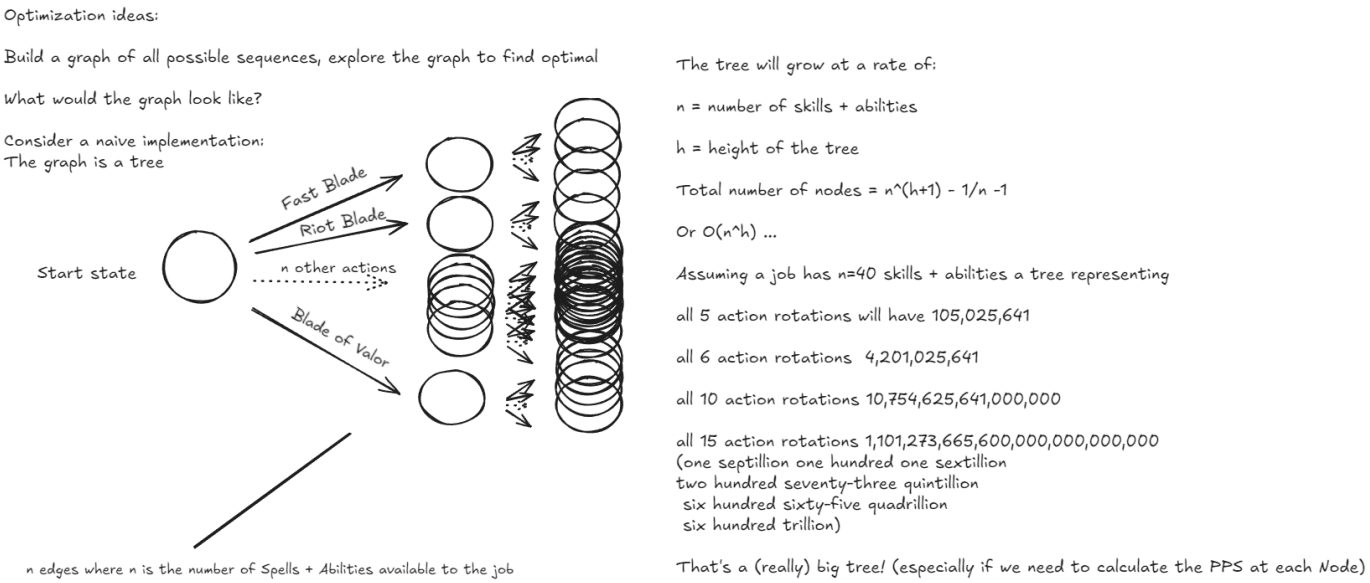

Figure 2. Demonstration of the scale of the optimization problem.

Figure 3 details the inputs our algorithm accepts and the outputs it produces. Given the assumptions that the fight is against a single target, the only buffs that are relevant are those that are casted by the person creating the rotation, critical hits are irrelevant, and that the player is both max level and can execute the rotation perfectly in-game, the algorithm should base its calculations off of the inputted potency, duration, and GCD time and output a valid rotation for the user to execute. The structure of such a rotation is also displayed at the bottom of Figure 3.

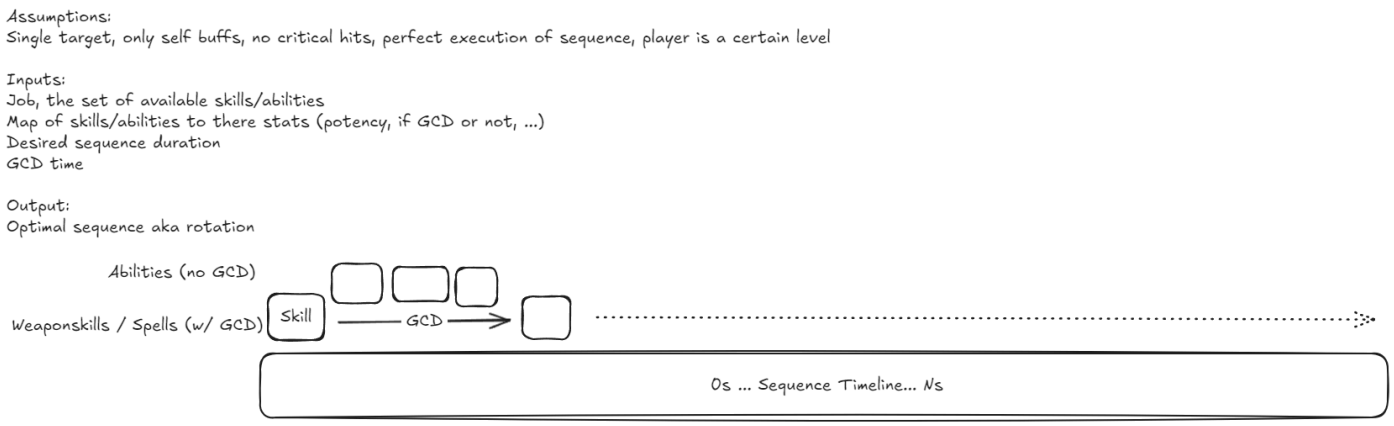

Figure 3. Visualization of algorithm inputs and outputs

```
{
    "paladin": {
        "actions": [
            {
                "icon": "https://lds-img.finalfantasyxiv.com/d/8325a7bb54f039c5bd3cd2af4430dccfd525e0b9.png",
                "name": "Fast Blade",
                "level": "1",
                "type": "Weaponskill",
                "cast": "Instant",
                "recast": "2.5s",
                "cost": "-",
                "effect": "Delivers an attack with a potency of 200."
            },
            {
                "icon": "https://lds-img.finalfantasyxiv.com/d/87405b9b9f00d9957df252ea0116e4137bc4dbd1.png",
                "name": "Fight or Flight",
                "level": "2",
                "type": "Ability",
                "cast": "Instant",
                "recast": "60s",
                "cost": "-",
                "effect": "Increases damage dealt by 25%.<br>Duration: 20s"
            },
            {
                "icon": "https://lds-img.finalfantasyxiv.com/d/bdda04f3b284e3cbf7d07e50d9bffb21b74ce869.png",
                "name": "Riot Blade",
                "level": "4",
                "type": "Weaponskill",
                "cast": "Instant",
                "recast": "2.5s",
                "cost": "-",
                "effect": "Delivers an attack with a potency of 140.<br>Combo Action: Fast Blade<br>Combo Potency: 300<br>Combo Bonus: Restores MP"
            },
```


Figure 4. JSON data storage within the project

Data is primarily stored in JSON files within the project, such as the snippet of XIVFinal.json displayed above. This file contains information about every job and skill to be leveraged by various components of the calculator and optimizer.

VII. Technical Design

The technical design of the project is centered around the optimizer component. When designing the optimizer we needed to select an optimization algorithm. We considered several approaches, including brute force, deep learning, and simulated annealing, before deciding on Monte Carlo Tree Search (MCTS). Brute force was immediately ruled out due to the astronomical number of possible rotations (far exceeding a googolplex) making exhaustive search computationally infeasible. Deep learning, while powerful in some domains, was unsuitable for this project because it would require retraining after every game update, a frequent occurrence in FFXIV that introduces new abilities, rebalances existing ones, or changes job mechanics. Simulated annealing, another optimization approach, was considered but ultimately rejected due to its tendency to struggle with problems involving highly interdependent variables, such as rotation sequences with intricate cooldown and combo mechanics.

We selected MCTS because it is particularly well-suited for tackling large, complex problems with many possible outcomes. One of its key advantages is that it does not require domain-specific knowledge of FFXIV mechanics to function—it only needs to interact with the potency calculator to evaluate potential rotations. This made it both easier to implement and highly adaptable to updates or extensions. Additionally, MCTS can produce useful results quickly, while its performance improves with additional iterations, offering flexibility in time-constrained scenarios. Historically, MCTS has been chosen to solve similarly complex problems, such as game-playing strategies in Go and chess, making it a reliable and proven choice for the complex task of rotation optimization.

MCTS operates through four iterative steps: select, expand, simulate, and backpropagate, refining its understanding of the optimal sequence with each iteration. This is visualized in Figure 5.

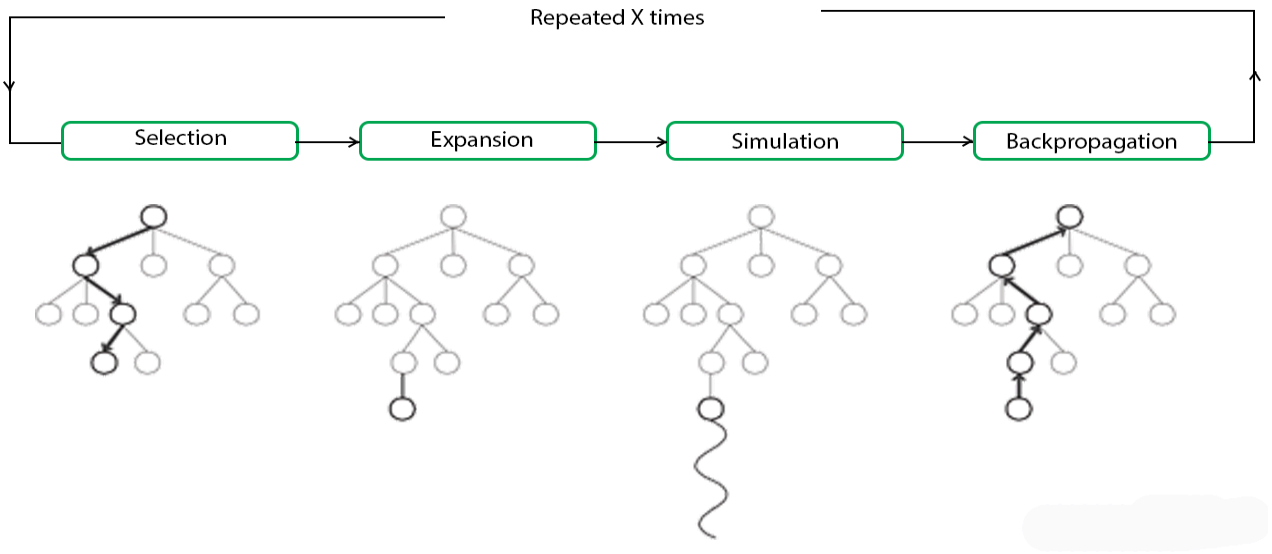

Figure 5. MCTS Visualization

The process begins with the select phase, where the optimizer navigates the decision tree starting from the root. Nodes are chosen based on their visit count and score, balancing the need to explore unvisited sequences with the potential of revisiting promising paths. This ensures that both new possibilities and high-scoring rotations receive attention. Each selected node represents a specific action in the rotation and acts as a baseline for further improvements.

In the expand phase, the optimizer adds new child nodes to the selected node. Each child represents a valid follow-up action from the current sequence. Validity is determined by the calculator written by the previous group. This systematic expansion of possible next steps mirrors how a player might manually consider logical progressions in their rotation. Essentially, in this phase, the optimizer builds out the entirety of the tree in preparation for future steps.

The simulate phase once again utilizes the calculator written by the previous group to evaluate the effectiveness of action sequences. Starting from the selected node's rotation, the optimizer appends random but valid actions until the total duration is reached. The calculator verifies the validity of the sequence and computes its potency. This phase directly ties into the project's goal by automating the labor-intensive process of manually constructing and testing rotations, as only valid and effective combinations are considered.

Finally, in the backpropagate phase, the optimizer updates the scores and visit counts of all nodes leading back to the root. Each node's score reflects the potency of the sequence it represents, allowing the algorithm to prioritize nodes that contribute to higher damage. This feedback loop allows the optimizer to prioritize paths that are more effective over paths that are not.

The optimizer's effectiveness is determined by the number of iterations. With each iteration, the algorithm explores new paths or refines existing ones, progressively improving its understanding of the tree. Early iterations offer a basic understanding of viable options, while longer runs allow the optimizer to refine its results and find rotations that are either optimal or very close to it. This is uniquely suited to our project, as it's quite easy to find an acceptably powerful rotation during the 15 minute time period described in the functional requirements, but hardcore players could potentially input a massive number of iterations and leave it running overnight to find something that approaches optimal or truly is optimal.

As stated above, one of the most notable advantages of MCTS over similar optimization algorithms is the fact that the algorithm itself does not encode any of the game rules. The optimizer relies on the calculator solely for validating action sequences and computing their potency. This separation guarantees that updates to the calculator, such

as adjustments for new game patches, balance changes, or expanded functionality, can be implemented independently without requiring changes to the optimizer's core logic. This modular approach enhances maintainability, reduces the risk of breaking functionality when the game is updated, and is overall the best approach to guarantee flexibility and continued functionality.

VIII. Software Test and Quality

Our software testing and quality is built upon three core pillars: unit testing, code reviews, and code standards. These core pillars are essential to ensure we build a high quality, maintainable app.

1. **Unit Testing:**
    a. As part of our software quality plan, we implemented unit testing to an existing part of the codebase. Originally this component was developed without any unit tests, which could allow for regression or unexpected bugs. In order to test the functionality of our app's calculator which is ultimately at the core of the program we implemented unit tests for it to ensure it works as expected. These tests utilize a variety of scenarios such as manually created rotations, single ability or skill execution, and combo demonstration to ensure our app's calculator is functioning as it should be and outputting the expected potency for such scenarios.
    b. Future features will be developed alongside appropriate unit tests to ensure that things are working correctly.
    c. Tests are automatically run via a Github Action whenever a pull request is made, and changes cannot be made to the core codebase until all tests pass.
2. **Code Reviews:**
    a. An additional, ongoing part of our software quality plan is to review new code whenever possible and especially prior to it being included into the main project. This ensures that everyone within the group is on the same page as well as ensuring the code being implemented into the project is of the highest quality functionally.
    b. This is done through Github pull requests, which are reviewed by our client and relevant team members.
3. **Code Standards:**
    a. A major part of our software quality plan involved refactoring the entire existing application from JavaScript to TypeScript. TypeScript allows for much higher code standards, with support for autocompletion, easier refactoring, early error detection, etc.
    b. Alongside the Type safety and linting provided by TypeScript we are using LWC (Lightning Web Components) instead of raw HTML whenever possible, ensuring we are using modern components and not re-inventing the wheel.
    c. Further code standards include leaving meaningful comments, using sensible variable names, and writing clean code.

IX. Project Ethical Considerations

Because of the nature of our project, it is difficult to say whether we would have to consider the public interest because the software is intended to be self-hosted and primarily for our client and those of their choosing. However, as it stands, our software is public and therefore accessible to anyone, so it's important that we still consider the ethics of our project. To help us be ethically sound, we are considering many rules from the ACM Code of Ethics and Professional Conduct. These principles guide our approach to the project, including the following key considerations:

1. **Public Impact:** Despite our intended usage scope, it's important to consider how our project could affect all FFXIV players. We need to be aware of any possible misuse or unintended effects and ensure the tool is helpful for everyone. Relevant principles include ACM 3.1, IEEE 1.01, and IEEE 1.02.
2. **Client and Employer Responsibilities:** It's important that we meet our client's goals while considering the public interest and our own as well. While we want to meet our client's goals for the optimizer—creating a powerful tool to help players perform better—we also need to keep in mind our responsibility to the public. This means being clear about what the software can and cannot do. Specifically, reference IEEE 2.09.
3. **Management:** As we work closely with our client, it's important to reflect and consider the management of the software we create. Specifically, reference IEEE 5.01.
4. **Colleagues:** As part of a team, it's important that we are supportive of each other and fair in our interactions and collaboration. By sharing and discussing our work, we can catch mistakes early and maintain high standards. Relevant principles include IEEE 6.01 and 6.04.
5. **Quality Assurance:** It's important to make sure our work is of high quality. We carried out thorough testing to ensure the optimizer works well and provides good results. Relevant principles include ACM 2.1, ACM 2.2, and IEEE 3.10.

X. Project Completion Status

The project is complete! Major refactoring and improvements were made to the previous team's code. The TS refactor was completed. Testing and automation improvements were implemented. The optimizer is finished and works as expected. Most importantly our client is happy with the current status of the project!

1. The core functionality of the optimizer is dependent on the existing calculator, which was not well evaluated and entirely untested. To ensure that the optimizer will produce accurate results, we wrote unit tests for the existing rotation calculator. We understood that edge cases we found would not be in the scope of this project. We have completed the unit tests, giving validation to some parts of the calculator and revealing edge cases it does not handle.
2. Another functional requirement of the project was to refactor the codebase from Javascript to Typescript, which we have completed.
3. We have updated the optimizer's input data to reflect the changes made by a game update by writing scripts to scrape the FFXIV job guide website [1].
4. We developed the optimizer interface to take parameters, including job, duration of rotation, optimization strategy, and GCD.
5. The core optimizer code is complete, using MCTS to find the best rotation possible under the non functional requirements. The optimizer runs in the browser, in less than 15 minutes, and produces an optimal or close to optimal rotation given the state of the calculator. The optimizer is shown in Figure 6.

6. Our work was documented for the client and the documentation was delivered alongside the project.

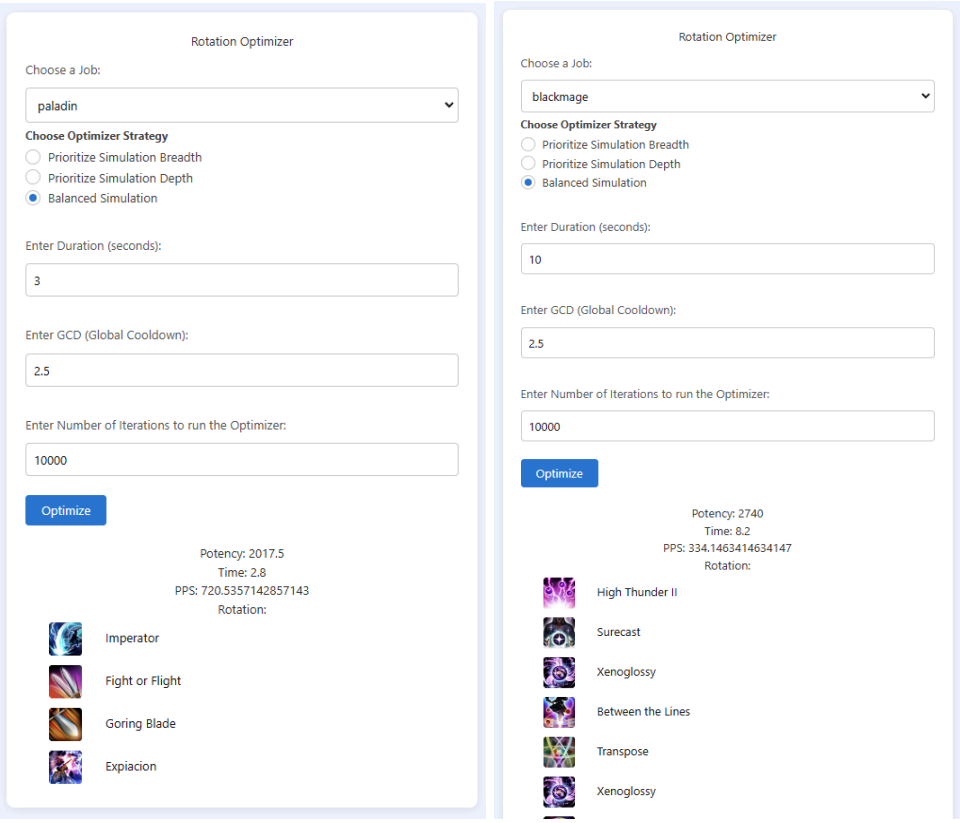

Figure 6. Example Optimizer Outputs

XI. Test Results

There are two core components to test, the rotation calculator, and the optimizer. Our test results for the calculator are purely an evaluation of the calculator we were given, and **this project accepted the risk that we would find issues with the calculator.**

Table 2. Calculator test results

Test	Pass / Fail	Explanation	Work needed
Calculator identifies invalid rotations	Fail	Most invalid rotations are found, however some edge cases are not found.	In the future, edge cases need to be fixed before the optimizer will work for all jobs.
Calculator returns accurate potency	Pass	Correct potency is found.	
Calculator correctly works for combo actions	Pass	Combos are handled correctly.	
Calculator handles XIV casting mechanics	Fail	Some new updates to casting mechanics are not handled by the calculator.	Each time FFXIV is updated the calculator will need to be updated to reflect

			changes.

Table 3. Optimizer test results

Test	Pass / Fail	Explanation	Work needed
Optimizer produces valid rotations, given the calculator	Pass	Some invalid rotations are found, however this is because of the calculator, and the optimizer function as expected.	In the future, edge cases need to be fixed before the optimizer will work for all jobs.
Optimizer produces near optimal results	Pass	The design of the optimizer is such that the longer it is run for the closer to optimal it will find. It is impossible to test all rotations so we will never know if results are truly optimal.	
Optimizer results produces outputs on par with manually built rotations	Pass	This is only true for some game jobs, specifically Paladin, Matt's main Job.	Update the calculator for other Jobs.
Optimizer is easy to use	Pass	The client likes the interface.	
Optimizer runs in less than 15 minutes in the browser	Pass	How long it runs for is adjustable, the longer it runs the better it will perform.	

Overall the optimizer works as expected and both the functional and nonfunctional requirements are met. Tables two and three show the results of the optimizer and calculator tests. Some tests are failing; however, this was a risk we accepted when we started developing the optimizer. The future work section reflects on what is needed to improve the application.

XII. Future Work

Much of the project's future work includes functionality that would develop the app further by reducing error, increasing usability, and implementing quality of life features for potential users.

1. Producing the optimal output would be ideal, however some issues with edge cases in the existing calculator make this impossible. We accepted this risk when beginning the project, but future work to fix these issues would ensure the output is always correct and close to optimal.

2. For users that already have a rotation they would like to use and expand upon, we would like to add an option to allow a customizable “starting point” of the optimizer algorithm. This would facilitate increased usability of the app, diverging from the current, rigid state of the optimizer in which there is no customizability.
3. Testing to compare the output of the MCTS optimizer to other benchmarks, including various player developed rotations for different jobs would be a great way to compare our results to those of players manually created rotations.
4. Using advanced web development techniques such as using Web Assembly would help to optimize the optimizer itself. This would help reduce the runtime increasing usability for users. Other approaches such as memoization could help optimize the user experience by quickly serving pre-computed data.
5. A blocker in the app’s usability is how it’s run. Currently, users have to download the source code and run it themselves in the local browser. A future implementation to reduce this slog in usability however, is to put the app into production, hosting a site that anyone can visit without needing to download the project.
6. After the app is put into production the greater FFXIV player base would be able to use it and help us collect user feedback. This would allow for new ideas for ways we can improve the app.

XIII. Lessons Learned

In the process of developing the FFXIV rotation optimizer, we encountered quite a few challenges and roadblocks that provided us with valuable insights regarding code standards, client interaction, and project planning.

1. Legacy code is difficult to work with. This project was worked on by another group in a previous field session, and their work was entirely untested and largely lacked documentation. This led to frequent confusion regarding what code did and why it was done that way, necessitating large, time-intensive refactors. If the code had been tested, it may have been written well the first time, and if it had been written well the first time, there would have been no need for us to redo large swaths of the existing codebase.
2. Frequent meetings with the client are essential. Throughout the course of the semester, we held weekly meetings with our client. These meetings allowed frequent and early feedback on the state of our work, assuring that all time spent was furthering the goals of the client and also preventing blockers due to the technical assistance provided by our client’s expertise.
3. Estimating the amount of time a task will require is difficult. When we were assigning stories in our sprint planning meetings, we would frequently underestimate the amount of work a task would require, as it is very easy for the complexity of a story to rapidly increase as it is worked on. We learned that it is better to plan somewhat pessimistically (i.e. expect the worst case) and then work from there in order to avoid missing deadlines due to unexpectedly large stories.
4. Planning ahead for potential risks can help keep expectations reasonable and ensure that goals can be met. When we first started with the project, we identified and discussed potential risks, including the unknown accuracy of the calculator. By accepting that, until the calculator works, the optimizer may produce incorrect results, we were able to stay on track to finish the project. By communicating this risk to the client, we were able to ensure that we were aligned on expectations and that delivered exactly what was expected.

XIV. Acknowledgments

We would like to thank Matt Buland for proposing this project and for the amazing experience it has been developing the XIV rotation optimizer. As Matt works as a software engineer we are very grateful for the incredible feedback and advice we have gotten throughout this semester. We would also like to thank our advisor Donna Bodeau for the advice and guidance through the various challenges we have encountered. Finally we would like to thank Kathleen Kelly for facilitating and organizing the field session projects, this has been a fantastic experience and one of the most impactful in our time at Mines.

XV. Team Profile

Tylor Bray

- Senior in Computer Science
- In the 4+1 masters program
- Born in Denver
- Interested in Software Engineering and Cybersecurity

Seth Motta

- Senior in Computer Science
- Born in Thornton
- Work Experience: King Soopers, AMS Department Main Office Assistant
- Interested in reading, writing, game development, SWE, and graphics engineering

Bryson Smith

- Senior studying Computer Science
- From Colorado Springs
- Currently working at Teradyne as a software engineering intern
- Interested in mountain biking, video games, mechanical keyboards, and software

References

[1]

SQUARE ENIX Inc, “FINAL FANTASY XIV Job Guide,” FINAL FANTASY XIV, 2024. https://na.finalfantasyxiv.com/jobguide/battle/ (accessed Dec. 03, 2024).

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
rotation	*A set of skills and abilities used in a particular order while playing FFXIV*
FFXIV	*Final Fantasy 14*
potency	*The effectiveness of the damage of skills and spells in FFXIV*

MCTS	*Monte Carlo Tree Search, an algorithm to explore large trees*
TS	*TypeScript*