Mariam Ahmad 202202568

Reflection Document: Enhancing React Tetris with Aider

Introduction

This reflection document summarizes my experience using Aider to enhance the React Tetris project. Over the course of this assignment, I used Aider as an AI pair programming assistant to implement new features, understand the codebase, and solve various challenges. This document outlines the techniques I discovered, limitations I encountered, and compares this workflow to traditional coding methods.

Most Effective Aider Techniques

1. Codebase Understanding Through Targeted Questions

One of the most effective techniques I discovered was using Aider to understand specific parts of the codebase through targeted questions. By adding only relevant files to the context and asking specific questions about functionality, I could quickly grasp how different components interacted.

For example, when I added `src/control/states.js` and asked about the game state management, Aider provided a comprehensive breakdown of the state transitions and game logic. This was significantly faster than manually tracing through the code and would have taken hours to understand traditionally.

2. Incremental Feature Implementation

Breaking down complex features into smaller, manageable chunks and implementing them incrementally proved highly effective. Rather than attempting to add the entire power-up system at once, I:

1. First had Aider explain the existing state management
2. Then added the base power-up data structure
3. Followed by implementing detection logic
4. Finally added the activation and effects

This incremental approach allowed Aider to focus on specific tasks rather than getting overwhelmed by the entire feature scope.

3. Context Management and File Selection

Being selective about which files to add to the context was crucial for getting quality assistance from Aider. I found that adding 3-5 related files at a time produced better results than:

- Adding too few files (lacking context)
- Adding too many files (overwhelming the context)

The `/add` command followed by targeted questions with the `/ask` command created an effective workflow for understanding and modifying specific parts of the application.

4. Command Chaining for Workflow Efficiency

I discovered that certain command sequences were particularly effective:

- `/add` → `/ask` → Code modification request → `/diff`
- `/add` → `/edit` (for small changes) → `/diff`
- `/run` → Identify issues → `/ask` for solutions → Implement fix

This created a natural and efficient workflow that resembled pair programming with a human.

Limitations Encountered

1. Context Window Constraints

Despite careful file selection, Aider occasionally struggled with understanding relationships between components that weren't explicitly included in the current context. The need to constantly add and remove files from context to stay within token limits was time-consuming.

2. Complex State Management Challenges

When implementing features that touched the Redux state management (particularly the power-up system), Aider sometimes generated code that didn't properly account for all edge cases. This required additional manual intervention and debugging.

3. UI and CSS Modifications

Aider had difficulty visualizing the actual UI, resulting in some CSS changes that didn't achieve the desired visual effect. This required multiple iterations and manual tweaking of the styling code.

4. Limited Testing Capabilities

While Aider could generate test cases, it couldn't actually observe the running application to detect visual or interaction bugs. This meant that all implementations required manual testing, which reduced some of the time-saving benefits.

Comparison to Traditional Coding Workflow

Speed and Efficiency

Aider significantly accelerated certain aspects of development:

- **Codebase understanding**: 2-3x faster than manual code review
- **Feature implementation planning**: Much more comprehensive and faster
- **Boilerplate code generation**: Nearly instantaneous vs. manual typing
- **Bug identification**: Faster in some cases, though required manual verification

Quality and Reliability

The quality of Aider-generated code was:

- **Generally solid**: Syntactically correct and followed project patterns
- **Sometimes incomplete**: Edge cases occasionally missed
- **Integration challenges**: Required more careful review than hand-written code

Learning and Skill Development

Using Aider created an interesting learning dynamic:

- The explanations helped me understand the codebase faster
- I gained insights from seeing alternative implementation approaches
- However, I felt less connected to some parts of the code I didn't write

Suggestions for Improving Aider for Similar Tasks

1. **Better Persistence of Project Understanding**: A mechanism for Aider to maintain a more comprehensive understanding of the codebase beyond the current context window would reduce the need for repetitive explanations.
2. **Visual Preview Integration**: The ability to see screenshots or previews of UI changes would greatly improve the CSS and styling workflow.
3. **Interactive Testing Capabilities**: Adding the ability for Aider to understand the results of running tests or the application would create a more complete feedback loop.
4. **Workflow Templates**: Pre-defined sequences of commands for common tasks (e.g., "add feature," "fix bug") would streamline the development process.

5. **Enhanced Refactoring Support**: Improved capabilities for suggesting and implementing refactoring across multiple files would be valuable for larger code changes.

Conclusion

Working with Aider to enhance the React Tetris project was an enlightening experience that demonstrated both the current capabilities and limitations of AI pair programming. The tool excelled at accelerating understanding and implementation but still required significant human guidance and verification.

For projects with well-structured codebases like React Tetris, Aider proved to be a valuable assistant that could significantly accelerate development tasks. The most effective approach was using Aider as a collaborative tool rather than a replacement for human development skills, combining AI assistance with human creativity and judgment.

I found that the key to success was developing a workflow that played to Aider's strengths (code understanding, generation, and pattern recognition) while compensating for its limitations through careful human oversight and testing. As the technology continues to improve, I expect the balance of responsibilities to shift, but the collaborative nature will likely remain the optimal approach for complex development tasks.