Homework 7: Reflection

1. **In retrospect, what could you have done better to reduce the time you spent solving this assignment?**

   Looking back, a more methodical approach towards the testing infrastructure, especially concerning the hidden instructor tests, would have significantly reduced the time spent. A substantial amount of effort went into troubleshooting test failures that stemmed not only from environmental factors, like line ending inconsistencies, but also from subtle edge cases concerning input handling and program termination revealed only by the instructor suite. Establishing a detailed verification system for test outputs early on, ensuring consistency, would have been beneficial.

   More critically, developing a strategy for diagnosing failures specific to the hidden tests would have been very helpful. Unlike local JUnit tests where behavior could be directly examined and debugged step-by-step, resolving these hidden failures required interpreting minimal diff output (‘Expected‘ vs. ‘Actual‘). This process of deduction was necessary to understand the specific edge cases being tested, particularly the intricate scenarios involving End-Of-File (EOF) conditions interacting with invalid user input or partial commands. Implementing a more robust local testing framework designed to simulate these specific EOF conditions and input sequences could have reduced the reliance on the slower feedback cycle from the grading server.

   While examining the graph structure and planning the MVC architecture were important, the most challenging aspect remained dealing with tests demanding exact output matches. This included whitespace, line endings, and, crucially for the hidden tests, the precise timing of error messages relative to input prompts and EOF events. Developing local tests that specifically mimic the inferred logic of challenging hidden tests, like EOF occurring immediately after partial command input, would have been a more effective use of time than relying solely on the provided test feedback. Automating the capture of expected outputs for basic cases could have provided a baseline, but wouldn't have covered the hidden edge cases. Finally, better separation between the graph model and the console interaction logic might have simplified debugging by allowing isolated testing of pathfinding algorithms.

2. **What could the Principles of Software staff have done better to improve your learning experience in this assignment?**

   While the assignment was instructive, the learning experience could have been enhanced with clearer guidance on certain aspects. Explicit instructions on handling potential cross-platform testing issues like line endings would be helpful. Furthermore, clearer specifications or examples regarding the expected handling of complex edge cases, particularly those involving End-Of-File (EOF) interacting with invalid input as tested by the hidden suite, would have been beneficial. Understanding the exact expected output sequence and program behavior during various EOF events was a significant hurdle.

   Additional explanation of how the testing framework, especially the hidden component, evaluates output and handles program termination would provide valuable context. A starter test suite demonstrating not only basic functionality but also some common edge cases, like partial input before EOF, could set clearer expectations from the outset. While visualization tools might be complex to provide, even a detailed guide on creating and validating test files for console applications, addressing EOF simulation nuances, would be useful.

More detailed examples of the expected console output format, specifically under error conditions and various EOF scenarios, would have reduced the time spent aligning application output with hidden test expectations. Given that some hidden test failures seemed due to exact string matching and precise output timing rather than core algorithmic errors, slightly more descriptive feedback or hints for these specific types of failures (beyond just the diff) could have guided the debugging process more effectively. Alternatively, explicitly stating any non-standard assumptions made by the test harness (e.g., the expected behavior upon encountering an empty line before EOF) would clarify requirements.

3. **What do you know now that you wish you had known before beginning the assignment?**

   This assignment provided several valuable insights that would have been beneficial earlier. I now understand the critical importance of consistent line endings and anticipating subtle platform differences in testing. More profoundly, I appreciate the necessity of meticulous attention to control flow and output timing when dealing with interactive console applications and End-Of-File (EOF) conditions. It became clear that seemingly minor differences in when an error message (like Ünknown buildingör Ünknown option ̈) was printed relative to detecting EOF during subsequent input reads were critical for satisfying specific hidden test requirements.

   I gained experience in debugging issues manifested only in hidden test suites, learning to rely on careful analysis of diff outputs to hypothesize and iteratively test potential edge-case behaviors. This often required a process of elimination and educated guesses, like speculating on how the test harness might use empty lines before EOF, to finally match the test's specific expectations. This iterative process, where fixing one subtle issue sometimes revealed another, underscored the need for careful, systematic debugging and the ability to adapt code to meet precise external specifications, even if those specifications required logic that seemed counter-intuitive based on standard practice.

   Beyond the testing challenges, I have a better grasp of techniques for designing cleaner MVC separations within the constraints of a text UI, the importance of structuring 'build.gradle' effectively, strategies for testing applications reliant on 'System.in'/'System.out', methods for calculating geometric directions in specific coordinate systems, and approaches for reusing code while maintaining good design. Most significantly, I now understand the importance of anticipating and designing for subtle edge cases in input handling and program termination, especially when interacting with automated test harnesses that may have very specific, non-obvious requirements. The experience of debugging the EOF and error-reporting interactions provided valuable, albeit challenging, lessons in robust program design and testing strategy.