Title: ComplianceChecker: A Java-Based Simulation Inspired by 'Papers, Please'

Lab Section: EQ8

Team Members: Team Green Lantern

Date: Nov 11, 2023

ComplianceChecker: A Java-Based Simulation Inspired by 'Papers, Please'

B. R. Bastasa[1] , Xi, Hongxia[2] And Talicol, Matt[3]

[1]*Department of Engineering, Gokongwei College of Engineering*
[2]*Department of Engineering, Gokongwei College of Engineering*
[3]*Department of Engineering, Gokongwei College of Engineering*
De La Salle University, Taft Avenue, Manila, Philippines
LBYCPA2, EQ8, GROUP1

ABSTRACT

This document serves as a template for the ComplianceChecker project, a Java-based simulation inspired by the game "Papers, Please." The project aims to enhance the understanding and practical application of various data structures and algorithms among students in a LAB Java class. The primary focus is on dynamic queue management using queue data structures, efficient data handling through tree data structures like Binary Search Trees (BST) and AVL trees, and the analysis of algorithmic complexity. This approach not only aligns with the course's learning objectives but also provides a hands-on experience in applying theoretical concepts in a simulated environment. The project's success will be evaluated based on functionality, efficiency, and adherence to the learning outcomes of the course.OI

Commented [1]: (Arial, 10, Italic) A condensed version of the entire paper; summarizes the four essential aspects of the paper, namely, purpose of the experiment, results (% error, % difference, etc.), significance of the study, and major conclusions. It should give one an idea of the scope of the study. It includes a very brief summary of the major steps/techniques used in the experiment. Do not include too much background information. Typically, 100-250 words.

Keywords: Java programming, data structures, algorithms, simulation, educational tool, queue management, tree data structures, algorithmic complexity.

I. Introduction

The ComplianceChecker project is an innovative Java-based simulation, inspired by the popular game "Papers, Please." This project is designed to align with the learning objectives of the LAB Java class at our college, particularly focusing on the understanding and application of various data structures and algorithms. The primary goals of this project are to deepen students' understanding of Java programming concepts and to provide hands-on experience in implementing complex data structures and algorithms in a practical setting.

Gameplay Aspects of ComplianceChecker

To adapt the core gameplay mechanics of the ComplianceChecker for a 3-week development period by a team of 4-5 people, the features are streamlined while maintaining the game's essence.

1. Basic Document Verification:

- Players act as compliance officers, verifying a limited set of documents, such as identification cards and basic permits.

- The verification process focuses on checking key elements like expiry dates, names, and photo matching.

- The variety of documents is limited to reduce complexity in coding and design.

2. Static Rule Set:

- The game uses a fixed set of rules for document verification, eliminating the need for programming dynamic scenarios.

- This rule set is simple yet challenging, such as verifying document validity based on a specific date.

3. Basic Queue Management:

- A simple queue system is in place where characters line up for document verification.

- Basic queue operations like adding and removing characters are implemented without complex time-based penalties.

4. Simplified Decision Making:

- Players make decisions to approve or reject documents based on their validity, with fewer consequences.

- The impact of decisions is limited to the player's score, simplifying the narrative and branching storylines.

5. Relaxed Time Management:

- Each game day is timed but with more lenient time constraints to reduce pressure and complexity.

- The focus is on processing a set number of characters per day, rather than balancing speed and accuracy.

Streamlined Interactive Elements

1. User Interface:

- The interface includes basic drag-and-drop functionality for document comparison.

- Essential tools like a basic rulebook and a simple document viewer are provided.

2. Character Interaction:

- Characters have basic backgrounds, with limited interaction to reduce narrative complexity.

- The emphasis is on the verification process rather than deep character-driven storylines.

3. Basic Performance Metrics:

- Simple metrics such as the number of correctly verified documents and basic errors are tracked.

- These metrics are used for player feedback and scoring, avoiding complex analytics.

Objectives

To further refine the objectives for the ComplianceChecker project, especially considering the streamlined approach for a 3-week development cycle, additional goals can be set to ensure a focused and productive development process. These objectives are designed to align with the educational aspects of the project while also ensuring that the game is engaging and achievable within the given timeframe.

1. Reinforce Fundamental Java Programming Skills:

- Emphasize the application of basic Java programming concepts, such as loops, conditionals, and data structures, in the game's development.

2. Introduce Basic Game Development Principles:

- Provide team members with an understanding of game development cycles, including planning, design, coding, testing, and deployment.

3. Develop Efficient Code and Algorithmic Thinking:

- Focus on writing clean, efficient code, and encourage the use of algorithms for basic game mechanics like document verification and queue management.

4. Foster Team Collaboration and Project Management Skills:

- Enhance team collaboration skills through coordinated development efforts, regular meetings, and effective use of version control systems like Git.

5. Implement a User-Friendly Interface:

- Design a simple yet intuitive user interface that allows easy navigation and interaction within the game.

6. Incorporate Basic Error Handling and Debugging Techniques:

- Teach basic error handling and debugging practices to ensure the game runs smoothly and is free of major bugs.

7. Achieve a Balance Between Educational Value and Entertainment:

- Ensure that the game is not only a learning tool but also engaging and enjoyable for players.

8. Adapt to Feedback and Iterative Improvement:

- Encourage an iterative development process, where feedback is used to make continuous improvements to the game.

9. Develop Documentation and Reporting Skills:

- Create comprehensive documentation detailing the game's design, implementation, and user instructions, enhancing technical writing skills.

10. Explore Basic Artificial Intelligence (AI) for Character Behavior:

- If time permits, introduce basic AI concepts for character behavior, making the game more dynamic and interesting.

11. Ensure Scalability and Maintainability of Code:

- Write code that is scalable and maintainable, allowing for future enhancements or modifications.

12. Incorporate Basic Sound and Graphics:

- Include simple sound effects and graphics to enhance the gaming experience, introducing basic multimedia integration in Java.

Scope

The project will feature dynamic queue management using queue data structures, profile and requirement repositories using tree data structures (such as Binary Search Trees and AVL trees), and an analysis of algorithmic complexity. Constraints include the time frame for development, the current skill level of team members, and the resources available.

II. Methodology

The methodology for the ComplianceChecker project is designed to ensure a structured and efficient approach to developing a Java-based simulation. Given the educational context and the project's alignment with the learning objectives of a LAB Java class, the methodology encompasses several key phases, from initial planning to final implementation and testing. The choice of the Integrated Development Environment (IDE) is crucial, and while IntelliJ IDEA is a strong candidate, alternative game development IDEs are also being considered for their simplicity and efficiency.

1. **Design Phase:** We will design the overall architecture of the program, including the user interface and the underlying data structures. This phase will involve creating diagrams such as IPO, flowcharts, and UML diagrams to visualize the project structure.

Requirement Analysis: Identify and document the specific requirements of the ComplianceChecker simulation, including the core features and functionalities that align with the learning objectives.

Tool Selection: Evaluate IntelliJ IDEA and other potential IDEs for game development, such as Eclipse with WindowBuilder, NetBeans, or lightweight alternatives like BlueJ or Greenfoot, which are known for their simplicity and educational focus.

IDE Selection: Finalize the choice of IDE based on factors like ease of use, compatibility with

Java, support for game development, and educational value.

Environment Configuration: Set up the chosen IDE with necessary plugins and tools for Java development and, if applicable, game development.

2. **Implementation Phase**: In this phase, we will code the designed structures in Java, focusing on implementing dynamic queues, tree structures, and efficient algorithms.

Architecture Design: Develop a high-level architecture for the ComplianceChecker, outlining the main components such as the dynamic queue system, profile and requirement repositories, and the user interface.

Coding: Begin coding in Java, focusing on implementing the core functionalities. Utilize appropriate data structures (queues, trees) and algorithms, ensuring that the code is well-documented and adheres to best practices.

Iterative Development: Adopt an iterative approach, where features are developed, tested, and refined in cycles, allowing for continuous improvement and integration.

3. **Testing and Optimization Phase:** The program will undergo rigorous testing to ensure functionality and performance. Algorithmic efficiency will be a key focus, with optimizations made as necessary.

Unit Testing: Conduct thorough unit tests for individual components to ensure they function as intended.

Integration Testing: Test the integration of different components to verify the overall functionality of the simulation.

Debugging: Identify and fix any bugs or issues that arise during testing.

4. Documentation and Reporting:

Technical Documentation: Prepare comprehensive documentation detailing the design, implementation, and usage of the ComplianceChecker.

User Manual: Develop a user manual to guide users through the features and functionalities of the simulation.

5. Final Review and Submission:

Peer Review: Conduct peer reviews to gather feedback and make final refinements.

Project Submission: Prepare the final version of the project, including the code, documentation, and user manual, for submission.

Major Milestones:

1. Completion of the Design and Architecture (Week 11)

- Finalize the overall architecture of the ComplianceChecker, including user interface and data structure design.

- Complete and approve all necessary diagrams (IPO, flowcharts, UML diagrams).

- Ensure that the design aligns with the project's objectives and learning outcomes.

2. Implementation of Core Functionalities (Week 12)

- Begin coding the designed structures in Java, focusing on dynamic queues, tree structures (Binary Search Trees, AVL trees), and efficient algorithms.

- Ensure that the code is well-documented, adhering to best practices and coding standards.

- Implement core functionalities, ensuring they align with the project's requirements and objectives.

3. First Round of Testing and Feedback (Week 12)

- Conduct initial unit tests to validate the functionality of individual components.

- Perform integration testing to ensure all components work cohesively.

- Gather initial feedback from peers or instructors to identify areas for improvement.

4. Final Optimizations and Second Round of Testing (Week 13)

- Refine and optimize the code based on feedback and testing results.

- Focus on enhancing algorithmic efficiency and addressing any identified bugs or performance issues.

- Conduct a comprehensive second round of testing, including stress testing and user experience testing.

5. Project Submission (Week 13)

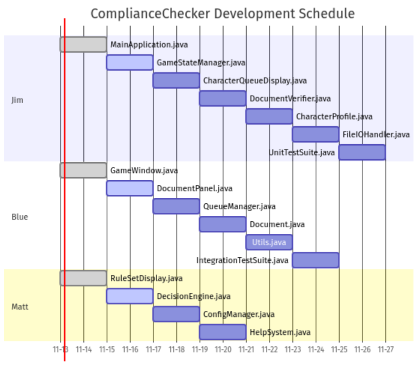
- Finalize all documentation, including technical documentation and user manual.

- Ensure that the project meets all evaluation criteria: functionality, efficiency, usability, and code quality.

- Submit the complete project package, including code, documentation, and any additional resources.

These milestones are critical for the structured and efficient progression of the ComplianceChecker project. Each milestone is designed to build upon the previous one, ensuring a cohesive development process that aligns with the educational objectives of the LAB Java class. The focus on iterative development and continuous testing is essential to achieve a high-quality, functional, and efficient final product.

III. Deliverables



Gant chart, detailing the timeline and responsibilities of each team member.

Deliverables include:

1. The fully functional ComplianceChecker program.

Core Application Files

a) MainApplication.java
   a. Purpose: Acts as the entry point of the application, initializing the game environment and managing the main game loop.

b) GameStateManager.java
   a. Purpose: Manages different states of the game (e.g., start screen, in-game, game over) and transitions between them.

User Interface and Interaction Files

c) GameWindow.java
   a. Purpose: Manages the main game window and rendering of the game's graphical elements.

d) DocumentPanel.java
   a. Purpose: Handles the display and interaction with the documents within the game.

e) CharacterQueueDisplay.java
   a. Purpose: Visualizes the queue of characters awaiting document verification.

f) RuleSetDisplay.java
   a. Purpose: Displays the current set of rules and regulations that players must follow.

Gameplay Mechanics Files

g) DocumentVerifier.java

    a. Purpose: Contains logic for verifying the authenticity and compliance of documents.

h) QueueManager.java
    a. Purpose: Manages the queue of characters, implementing efficient queue algorithms.

i) DecisionEngine.java
    a. Purpose: Processes player decisions on document approval or rejection and calculates consequences.

    b. Data Management and Structures Files

j) CharacterProfile.java
    a. Purpose: Represents the profiles of characters, including their documents and background stories.

k) Document.java
    a. Purpose: Represents different types of documents that players will inspect.

Utility and Support Files

l) FileIOHandler.java
    a. Purpose: Manages file input/output operations, such as saving and loading game states.

m) Utils.java
    a. Purpose: Contains utility functions used across the application (e.g., date and time utilities, string processing).

n) ConfigManager.java
    a. Purpose: Manages game configuration settings.

    b. Testing and Debugging Files

o) UnitTestSuite.java
    a. Purpose: Contains unit tests for individual components and functionalities.

p) IntegrationTestSuite.java
    a. Purpose: Contains tests for the integration of different components and overall game functionality.

Documentation and Help Files

q) HelpSystem.java
    a. Purpose: Provides in-game help, tutorials, and FAQs.

Additional Considerations

Graphics and Resource Management: Files for managing graphical assets, animations, and other resources.

SoundManager.java: Manages audio effects and background music (if time permits).

This structure provides a comprehensive yet manageable framework for the ComplianceChecker game, ensuring that each aspect of the game is adequately addressed while keeping the development process within the constraints of the 3-week timeline. Each file is designed to fulfill specific roles within the game, from managing game states and user interactions to implementing core gameplay mechanics and educational content.

2. A comprehensive user manual.

3. A technical document detailing the design, implementation, and testing processes.

IV. Evaluation

Evaluation Criteria:

1. Functionality: Does the program meet the specified requirements?

2. Efficiency: Are the data structures and algorithms implemented efficiently?

3. Usability: Is the user interface intuitive and user-friendly?

4. Code Quality: Is the code well-organized, commented, and adhering to best practices?

V. Conclusion

The ComplianceChecker project represents a significant opportunity to apply theoretical knowledge in a practical, engaging manner. It addresses the need for hands-on experience in software development, particularly in the areas of data structures and algorithms, and aims to bridge the gap between academic learning and real-world application.

VI. References

Forster, D. (n.d.). Philosophical Problems in Artificial Intelligence. PhilArchive. Retrieved from https://philarchive.org/archive/FORPPA-13

Papers Please Wiki. (n.d.). General tips. Fandom. Retrieved from https://papersplease.fandom.com/wiki/General_tips

GameCentral. (2019, November 30). The genius gameplay mechanics of Papers, Please. Metro. Retrieved from https://metro.co.uk/2019/11/30/genius-gameplay-mechanics-papers-please-11244037/

SteamLists. (n.d.). Papers, Please Walkthrough Guide: Basic Mechanics & Gameplay Tips. Retrieved from https://steamlists.com/papers-please-walkthrough-guide-basic-mechanics-gameplay-tips/