2805ICT/7805ICT System and Software Design

3815ICT Software Engineering

Assignment Specification: Final Submission

- Assignment Title: Enhanced Tetris Game Development
- Weight: 50%
- Submission Deadline: End of Week 10 (29/09/2024)

Objective

The primary objective of this assignment is to utilize Software Engineering principles and the Java programming language to develop an enhanced version of the classic Tetris game. This final submission will build upon the foundation established in Milestone 1, focusing on the completion and refinement of the project.

Background

Tetris is a classic puzzle video game originally designed and programmed by Alexey Pajitnov in 1984. The game involves players manipulating tetrominoes, shapes composed of four square blocks each, which descend onto a playing field. The goal is to fit these shapes into complete rows, which then disappear, earning points for the player. The game ends when the tetrominoes stack up to the top of the playing field and no more shapes can enter.

General Functions

To understand the general requirements and functionalities expected in the enhanced Tetris game, please study the provided demo game released in Week 6. Download the game and the server from our course website and explore all the functions. Detailed requirements will be provided in the later **Requirements and Marking Criteria** section.

Submission

The submission for this final milestone is a Word or PDF document based on the template provided on the course website. The document must include the following:

- **GitHub Project Link**:
  - Provide the link to your GitHub project repository.
  - Ensure that you have invited the course convenor to your project so they can access it.
- **Demo Video Link**:
  - Include a link to a video that demonstrates your final game. Detailed instructions for the video will be provided later in this specification.

Note: Each group only needs to make one submission. A student in the group will submit the document on behalf of the entire group.

Requirements and Marking Criteria

Subsection 1: Project Management and Reflection (10 points)

1.1. Project Plan (5 points)

Students are required to provide a comprehensive project plan specifically for the final submission. The project plan should adhere to the provided template and must include the following elements:

- **Task Definition and Assignment**:
    - Clearly define all tasks necessary for the completion of the final project.
    - Assign tasks to individual group members.
- **Time Estimation and Tracking**:
    - Estimate the time required to complete each task.
    - Record the actual time spent on each task.
    - Include both the planned task finish date and the actual completion date.
- **Group Meetings**:
    - Document the frequency of group meetings.
    - Specify the software tools used for project management and communication.
- **Effort Summary Table**:
    - Complete the effort summary table to provide an overview of the work distribution and contributions of each group member.

Note: This project plan should focus solely on the tasks required for the final submission. There is no need to include any tasks or details from Milestone 1.

1.2. GitHub Repository (0 points)

Students are required to provide the link to their GitHub repository. Ensure that the course convenor has access to the repository:

- GC students, please ask Elizabeth for her GitHub account.
- Online and Nathan students, please add `LarryAtGU` as a team member in your GitHub project.
- **If you use the same repository for the final submission and the course convenor is already in your project team, you don't need to reinvite them.**

Additionally, include a screenshot of the commit history **post-Milestone 1** in your submission. Ensure the screenshot is cropped so that the text is easy to read. Marks will be deducted for the following:

- Poor image resolution.
- Too few commits.
- Insufficient explanations in commit messages.
- Most commits occurring within a very short period.

- Too few group members contributing to the code.

Note: While this section carries no points, failure to comply may result in a deduction of up to 10 points.

1.3. Peer Review (5 points)

Peer review is an essential part of the collaborative development process, helping to improve the quality of the project through constructive feedback. You are required to conduct both a self-review and a peer-review based on the provided template. For this section, please include the following:

- **Self-Review**:
  - Complete a self-review using the provided template.
  - Reflect on your contributions, the challenges you faced, and areas for improvement.
- **Peer-Review**:
  - Conduct peer-reviews for your group members based on the provided template.
  - If you completed the project as a solo member group, you are exempt from the peer-review section but must still complete the self-review.
- **Practical and Constructive Feedback**:
  - Ensure that the peer-review process is practical and provides constructive feedback aimed at improving the quality of the project.
  - Focus on how peer feedback can be used positively to enhance collaboration and the final product.

1.4. Project Reflection (Master Student Only, 0 points)

If your group includes master students, each master student must provide a one-page reflection on their learning experiences from this project and suggest ways to improve the work in future projects.

Requirements:

- **Content of the Reflection**:
  - Reflect on the key learnings gained from participating in this project.
  - Identify specific areas where the project work could be improved in the future.

Guidelines:

- **Length and Detail**:
  - The reflection should be one page in length.
  - Provide thoughtful and detailed insights into your learning experiences and suggestions for improvement.

Note: Failure to provide a reflection or submitting a poorly written reflection may result in a deduction of up to 10 points from the overall project score.

Subsection 2: Architectural Level Design (15 points)

2.1. Class Diagram (5 points)

You are required to draw a class diagram that represents the overall design of your project. While it is not necessary to include every class from your project, focus on the most important ones that effectively reflect the design. The class diagram must adhere to UML notation and should include the following elements:

- **Relationships**:
  - **Association**: Show how different classes are related and interact with each other.
  - **Generalization**: Illustrate inheritance relationships where applicable.
  - **Composition**: Represent strong ownership relationships between classes.
- **Correct Multiplicity**:
  - Ensure that all associations in the diagram have the correct multiplicity, accurately representing the relationships between classes.
- **Interfaces/Abstract Classes**:
  - Include interfaces or abstract classes to show how your design supports abstraction and polymorphism.
- **Attributes and Methods**:
  - List important attributes and methods for the classes included in the diagram.
  - Ensure that these attributes and methods align with your actual implementation.
- **Implementation Alignment**:
  - The class diagram should accurately reflect your project's implementation.

You may include a brief explanation of up to half a page to describe your class diagram, highlighting key design decisions, the use of relationships, and how the diagram represents the overall structure of your project.

2.2. MVC Architectural Design Pattern (5 points)

You are required to apply the Model-View-Controller (MVC) architectural design pattern in your project. For this section, please provide the following:

- **Description of MVC**:
  - Provide a brief description (up to half a page) explaining the MVC architectural design pattern.
  - Discuss how you have implemented MVC in your project, highlighting how the pattern is applied in your design.
- **Class Diagram**:

    - Include a class diagram that shows only the classes relevant to the MVC pattern in your project.
    - Ensure that the diagram clearly illustrates the roles of the Model, View, and Controller components and how they interact with each other.

- **Code Screenshot**:

    - Provide a screenshot of relevant code that demonstrates the implementation of the MVC pattern in your project.
    - Ensure that the code screenshot is clear and properly highlights key aspects of your MVC implementation.

- **Alignment**:

    - Make sure that the code and the class diagram are fully aligned, with each component in the diagram accurately represented in the code.

2.3. Allocation Diagrams (5 points)

You are required to provide both an implementation style view and a deployment style view in this section. Please refer to the Week 5 lecture slides for guidance. The following elements must be included:

- **Implementation Style View**:

    - Provide a diagram that shows the structure of your project, including all folders and the file names of important files.
    - Include a brief introduction for each folder, explaining its purpose within the project.
    - Highlight some of the important files, providing a brief description of their roles and how they contribute to the overall project.

- **Deployment Style View**:

    - Provide a diagram that illustrates the deployment of your project.
    - This view should include your runtime `.jar` files, `.json` files, and the `TetrisServer.jar`.
    - Clearly depict the working environment for each of these deployment files and describe the relationships among them.
    - Ensure that the deployment view accurately represents how the components interact and operate within the runtime environment.

Subsection 3: Design Patterns (15 points)

3.1. Singleton Design Pattern (5 points)

The Singleton Design Pattern is a common design pattern used in many software projects. You are required to apply the Singleton Design Pattern in your project. For this section, please include the following:

- **Code Implementation**:
  - Provide the relevant code that demonstrates your implementation of the Singleton Design Pattern.
  - Ensure the code is clear and highlights the key aspects of the Singleton pattern.
- **Explanation**:
  - Provide a short explanation of your implementation, discussing how the Singleton pattern is applied in your project.
- **Thread Safety and Lazy Initialization**:
  - Discuss whether your implementation is thread-safe and whether it uses lazy initialization.
  - If applicable, explain how you ensured thread safety in your Singleton implementation or why it was not necessary.

3.2. Factory Design Pattern (5 points)

The Factory Design Pattern is another widely used design pattern in software development. You are required to implement the Factory Design Pattern in your project. For this section, please include the following:

- **Code Implementation**:
  - Provide the relevant code that demonstrates your implementation of the Factory Design Pattern.
  - Ensure the code is clear and highlights the key aspects of the Factory pattern.
- **Class Diagram**:
  - Include a class diagram that represents the structure of your Factory Design Pattern implementation.
  - The diagram should clearly show the relationships between the factory class and the products it creates.
- **Sequence Diagram**:
  - Provide a sequence diagram that illustrates the process flow of your Factory Design Pattern implementation.
  - The diagram should depict how objects are created through the factory method and how the different components interact.
- **Explanation**:

  - Provide a brief explanation of your implementation, discussing how the Factory Design Pattern is applied in your project.
- **Alignment**:
  - Ensure that both the class diagram and sequence diagram are fully aligned with your code implementation, accurately reflecting the structure and behavior of your design.

3.3. Facade Design Pattern (5 points)

The Facade Design Pattern is a structural pattern that simplifies the interaction with a complex system. You are required to implement the Facade Design Pattern in your project. For this section, please include the following:

- **Code Implementation**:
  - Provide the relevant code that demonstrates your implementation of the Facade Design Pattern.
  - Ensure the code is clear and highlights the key aspects of the Facade pattern.
- **Class Diagram**:
  - Include a class diagram that represents the structure of your Facade Design Pattern implementation.
  - The diagram should clearly show the relationship between the facade class and the subsystems it interacts with.
- **Explanation**:
  - Provide a brief explanation of your implementation, discussing how the Facade Design Pattern is applied in your project.
- **Alignment**:
  - Ensure that the class diagram is fully aligned with your code implementation, accurately reflecting the structure and behavior of your design.

Subsection 4: Good Programming Practices (25 points)

4.1. Using Sorting (5 points)

You are required to demonstrate the proper use of sorting within your project. For this section, please include the following:

- **Explanation of Sorting Usage**:
  - Provide a brief explanation (up to half a page) detailing how sorting is utilized in your project.
  - Highlight where and why sorting is applied, such as sorting top scores or optimizing movements in the AI section.
- **Implementation with Comparable or Comparator**:
  - You must implement sorting using either the `Comparable` or `Comparator` interface.
  - Include a relevant screenshot of the code that demonstrates your sorting implementation.
- **Code Demonstration**:
  - The code screenshot should clearly show how you've successfully applied sorting in the project, with a focus on the correct use of `Comparable` or `Comparator`.

4.2. Lambda Expression (5 points)

Lambda expressions are a significant extension in Java that support functional programming. You are required to apply lambda expressions in your project. For this section, please include the following:

- **Code Screenshot**:
  - Provide a screenshot of the code that includes the use of a lambda expression.
  - Ensure the screenshot is clear and highlights the lambda expression in your code.
- **Explanation**:
  - Provide a short explanation of the purpose of the code where the lambda expression is applied.
  - Discuss the benefits of using lambda expressions in this context, such as improved code readability, conciseness, or the ability to pass behavior as parameters.

4.3. Multiple Threads (5 points)

Java supports multi-threading, an essential feature for developing responsive and efficient applications. You are required to utilize this feature in your project. For this section, please include the following:

- **Code Implementation**:
  - Provide relevant code that demonstrates your implementation of multi-threading in your project.
  - Ensure the code clearly shows the creation, management, and coordination of multiple threads.

- **Explanation**:
    - Provide a proper explanation of how multi-threading is implemented in your project.
    - Discuss the purpose of using multiple threads, the tasks they perform, and any considerations for thread safety or synchronization.

4.4. Exception and Argument Validation Handling (5 points)

Proper exception handling and argument validation are crucial for building robust and error-resistant applications. You are required to demonstrate these practices in your project. For this section, please include the following:

- **Code Implementation**:
    - Provide relevant code that demonstrates how you handle exceptions and validate arguments in your project.
    - Ensure the code showcases key aspects of exception handling, such as try-catch blocks, custom exceptions, and input validation.
- **Explanation**:
    - Provide an explanation of your approach to exception handling and argument validation.
    - Discuss why these practices are important in your project, how they contribute to the stability and reliability of your application, and any specific scenarios where they are applied.

4.5. Testing (5 points)

Testing is a crucial aspect of software development to ensure the correctness and reliability of your code. You are required to use JUnit to create at least 5 unit tests for your project. For this section, please include the following:

- **Test Cases**:
    - Provide the code for at least 5 unit test cases written using JUnit.
    - Ensure that each test case is clearly linked to the method it is testing.
- **Tested Methods**:
    - Show the code of the methods being tested by the unit tests.
    - Explain briefly what each method is supposed to do and why it's important to test it.
- **Test Report**:
    - Include a test report that shows the results of your unit tests, indicating which tests passed and which (if any) failed.
    - Provide a code coverage report generated by the system, demonstrating how much of your code is covered by the tests.
    - Both reports should be created by JUnit, you only need to provide the screenshot of their report.

Subsection 5: Video Demonstration (35 points)

You are required to create a video demonstration of your project, focusing on specific functionalities. Please adhere to the following guidelines:

- **Maximum Length**:
  The video must not exceed 8 minutes. Any content beyond 8 minutes will not be viewed or assessed.
- **Video Quality**:
  Ensure the video is of high quality, with clear visuals and audio.
- **Voice Annotation**:
  Provide voice annotations throughout the video to explain and demonstrate the functionalities of your project.

5.1. Save and Load Files in JSON (5 points)

In your video, you need to demonstrate the following:

- **Updating JSON Files**:
  - Show that when you change the configuration of the game or update the high scores, the corresponding JSON files are updated accordingly.
- **Reading JSON Files**:
  - Demonstrate that when the game is restarted, these JSON files are read, and the data is correctly reflected in the high score and configuration screens.

5.2. Automatically Frame Size and Location Adjustment (5 points)

In your video, you need to demonstrate the following:

- **Dynamic Frame Adjustment**:
  - Show that each time a new play screen is loaded, the frame automatically adjusts its size according to the game settings, such as different field sizes and one or two-player modes.
- **Return to Default Size**:
  - Demonstrate that when a game is finished and the user returns to the main menu, the frame size automatically returns to the default size.
- **Centering the Frame**:
  - Ensure that during any size change, the frame remains centered on the screen. The frame should adjust smoothly without shifting away from the center.
- **Demo App Reference**:

- You might download the latest demo app from the course website and run it to check the desired size change operation. Use this as a reference to ensure your implementation meets the required functionality.

5.3. Music and Sound (5 points)

In your video, you need to demonstrate the following:

- **Music and Sound Implementation**:
  - Show that the game has implemented both sound effects and background music features.
- **Configuration Screen**:
  - Demonstrate that users can change the music and sound settings in the configuration screen.
- **In-Game Toggle**:
  - Show that during gameplay, users can toggle the music and sound settings by pressing the 'M' key for music and the 'S' key for sound.
  - The current setting status (on/off) for both music and sound should be visibly displayed on the gameplay screen.
- **Real-Time Update**:
  - When the user presses the 'M' or 'S' key to change the settings, the screen should immediately update to reflect the new setting status.
- **Audibility**:
  - Ensure that both the sound effects and background music are audible in the video, clearly demonstrating the functionality.

5.4. Score and Display (5 points)

In your video, you need to demonstrate the following:

- **Game Status Display**:
  - The gameplay screen should display key game status information, including:
    - Type of player (AI/Human/External)
    - Initial level
    - Current level
    - Current score
    - Lines erased
  - Ensure these elements are clearly visible and accurate during gameplay. Refer to the demo game for the exact details and layout.
- **Scoring System**:
  - Implement and demonstrate the following scoring rules:

        - Erasing one row earns 100 points.
        - Erasing two rows in a batch earns 300 points.
        - Erasing three rows in a batch earns 600 points.
        - Erasing four rows in a batch earns 1,000 points.
    - For every 10 rows erased, the player should advance one level.
    - Show how the score and level are updated in real-time during gameplay.

- **Game Completion and Top Scores**:
    - When a game is finished, if the player's score is within the top 10, an input dialogue should appear for the player to enter their name.
    - After confirmation, the new score should be added to the top scores and immediately visible on the top score screen.
    - The updated top scores should be saved in a JSON file.

- **Top Score Management**:
    - In the top score screen, demonstrate the functionality of a button that allows the player to clear the top scores.
    - After clicking the button and confirming, the top scores should be deleted from both the display and the JSON file.
    - Only positive scores can be added to the top scores.
    - Ensure the top scores are displayed in the format shown in the demo game.

5.5. AI Play (5 points)

In your video, you need to demonstrate the following:

- **AI Implementation**:
    - Show that you have successfully implemented an AI that can play the game.

- **AI Performance**:
    - To achieve full marks for this requirement, your AI must be able to acquire at least 500 points during gameplay.

- **Demonstration**:
    - Run the game in AI mode and display the gameplay in your video, clearly showing the AI playing and accumulating points.
    - Ensure the scoring system is visibly updating as the AI plays, and highlight when the AI reaches the 500-point mark.

5.6. External Player (5 points)

In your video, you need to demonstrate the following:

- **External Player Implementation**:

  - Show that your game can be controlled through an external player by connecting to a separate application, `TetrisServer.jar`.
  - The communication protocol for connecting to `TetrisServer.jar` will be provided soon, but your game should be prepared to connect through `localhost:3000`.

- **Gameplay with External Player**:

  - Demonstrate that when you select "External" as the player type, your game successfully connects to the `TetrisServer` and allows it to control the gameplay.

- **Handling Connection Issues**:

  - Show that if the `TetrisServer` is not running on your local computer when "External" is selected as the player type, your game gives a warning message and the blocks enter a state of no control.
  - Demonstrate that if you start the `TetrisServer` after the game has entered this state, your game immediately resumes proper gameplay, controlled by the `TetrisServer`.

- **Demo Reference**:

  - You may run the demo game to observe and verify this feature before implementing it in your project.

5.7. Extension Mode (5 points)

In your video, you need to demonstrate the following:

- **Extension Mode Implementation**:

  - Implement the extension mode in your game. By default, the extension mode should be a two-player mode. However, if you have received approval from the course convenor, you may implement a different extension mode.

- **Two-Player Mode**:

  - In the default two-player mode, when you enter the gameplay screen, the display should show two games running simultaneously, along with two separate game information panels.
  - Each game can be independently played by a human, AI, or external player, as defined in the configuration screen.

- **Fair Play**:

  - Ensure that in two-player mode, both games receive the same sequence of tetrominoes to allow for a fair comparison of performance between the players.

- **Demonstration Scenarios**:

  - In your video, demonstrate the two-player mode with the following scenarios:
    - Two humans playing against each other.
    - Two AIs playing against each other.

- Two external players playing against each other (if the external player feature has been successfully implemented).
- A player vs. AI match.