UC Software Engineering Project Report by group 081

Devana Monguchi (dmo118, 82172512)
Janice Leung (wcl29, 39126312)

Structure of the Application

The application source itself is divided into 3 main parts, the model, view (the UI) and last but not least the view manager. **1.** The model code is independent of the UI and does not have direct knowledge of how the data is presented to the user. It encapsulates the data structures, data access logic, validation rules, and business rules related to the application's functionality. Components in the Model layer include data models, data access objects (DAOs),and repositories. **2.**The view code represents the user interface (UI) elements that are visible to the user, it is passive and does not contain any business logic. Its sole responsibility is to display the data to the user and capture user input. The View observes the ViewModel to receive updates and binds UI elements to the ViewModel's properties. **3.** The view manager code acts as an intermediary between the Model and the View.It exposes data and commands that the View can bind to and interact with.The ViewModel retrieves data from the Model and prepares it in a format suitable for the View to display. It contains presentation logic, formatting data, performing transformations, and adapting the data to the requirements of the View. The ViewModel also handles user actions or commands triggered from the View, updating the Model accordingly.

We start from the main menu. The main menu is the page that connects all other pages. Six options are provided on the main menu, including Club, Stadium, Market, Save Game, Load Game, and Skip Week. Each Club, Stadium, and Market button is linked to another page. Save Game would allow the user to save the game's progress to avoid any accidents, and the user can press Load Game to stop the game at any point. The user can also Skip a Week if they do not want to do anything for the current game week.

We chose inheritance over interfaces for our components to efficiently handle shared functionality by grouping methods in a superclass. For example, our shoes and racket classes are inherited by the item class. Override

Design Decisions

A big decision we had to make is the implementation of the battle system, contained in the CompetitionPage class. In order to enhance user enjoyment and increase interactivity, our team has decided to take a calculated risk and introduce more interactive elements to the game. We implemented a timer component that enables the execution of a specific method, responsible for continuously monitoring and updating the player points throughout each round of the tournament. The timer component ensures a dynamic gameplay experience by regularly assessing the outcome of each round, as well as the overall progress of the tournament. It evaluates various conditions, such as determining whether a player has won or lost a particular round, and checks for situations where both players have accumulated an equal number of points, commonly known as a "Duce" scenario.By incorporating this looping mechanism, our aim is to provide players with a more engaging and immersive gaming experience. The introduction of these interactive elements not only adds a strategic dimension to the game but also encourages active participation and decision-making from the players themselves.

We have also added equipment options to enhance player customization and gameplay experience. These equipment choices boost the athletes' statistics and empower players to strategically tailor their playstyle. By considering the unique characteristics of each equipment and their synergies with athletes' strengths, players can optimize their team composition and gain a competitive edge. This addition adds depth, personalization, and strategic decision-making to the game, fostering an immersive and dynamic gameplay environment.

We ran JUnit test cases on Eclipse on important methods for Athlete, Game, and Game Master classes. After developing the GUI, integration testing was performed to make sure the GUI ran well with different classes as a module. Finally, the game was run with system testing, ensuring project specification requirements were met.

Brief retrospective of the project

This game has limitations regarding its scope and GUI design. Due to time constraints, we focused on men's doubles matches instead of the originally planned broader gameplay. The GUI could be more visually appealing, but the use of royalty-free images limited our options. The game lacks time limits for activities and lacks match details. Users' gameplay experience may be influenced by the time they spend on the game.

Improvement for the future

In future projects, we aim to enhance our planning and preparation. As novice game developers, we lacked prior knowledge and experience, leading to rushed work and unforeseen setbacks. To overcome these challenges, we will prioritize improved project management, allocate ample time for research and planning, and establish clear communication channels. We will also focus on acquiring necessary skills and learning from our experiences to ensure smoother development in future endeavors.

Thoughts and Feedback

Devana Monguchi
“In my opinion, this project served as a commendable initiation into project management and program design. One aspect that I found dissatisfying, though, was the timing of the project management, design, and architecture course material, which was primarily covered in the latter half of the semester. This knowledge would have been invaluable for project initiation and planning. While I acknowledge the priority given to learning Java coding skills, I can't

shake the feeling that there could have been an opportunity to introduce these concepts earlier."

Janice Leung

"This project provided an enjoyable avenue for the practical application of the course materials. There are many critical useful topics in the second half of the semester that can help us develop this project, so we changed some of our early decisions about game design after learning those topics. The labs focused on GUI-related issues are very helpful, so it would be great if lectures could also address how specifically we can apply those GUI-related skills in our project."

Division of Tasks and Effort Spent

Devana Monguchi: 185 hours (65%)

- Game design and development
- UML class diagram
- Command line application developing and coding
- Javadoc documenting
- GUI application developing and coding
- README writing

Janice Leung: 100 hours (35%)

- Game design
- UML class and use case diagrams
- Command line application initial coding
- GUI application design and initial developing
- Unit test writing
- Report writing

Agreed Contribution

We didn't establish a rigid percentage allocation explicitly. Instead, we mutually decided to strive for a substantial contribution, aiming for approximately 40% to 60% (preferably 50%) for each project component. We didn't enforce a strict 50-50 workload division as it would have been challenging to split the tasks precisely evenly, and we believed it would be unnecessary and hinder productivity.