Phase	Task	Assigned To	Start Date	End Date	Status	Comments
1. Initial Planning & Setup	Establish coding conventions and best practices for the project (e. g., naming conventions, code formatting, commit message rules).		Mar, 24, 2025	Mar, 25, 2025		
	Set up a code review process to maintain code quality.		Mar, 24, 2025	Mar, 25, 2025		
	Finalize system architecture and class design.		Mar, 24, 2025	Mar, 25, 2025		
	Review system architecture and identify missing components		Mar, 24, 2025	Mar, 25, 2025		
2. GeneralStats Class Implementation	Implement the __init__ method to initialize player-specific statistics (e.g., wins, losses, ties, games played, and MMR).		Mar, 25, 2025	Mar, 27, 2025		
	Implement methods like .win(), .lose(), and .tie() to update the statistics accordingly.		Mar, 25, 2025	Mar, 27, 2025		
	Write unit tests to ensure that the win(), lose(), and tie() methods properly update the statistics.		Mar, 28, 2025	Mar, 29, 2025		
	Define an abstract method update_mmr(self, win) that must be implemented by subclasses to update MMR based on game-specific rules		Mar, 28, 2025	Mar, 29, 2025		
	Implement the _update_rank() method to automatically update a player's rank based on their MMR.		Mar, 29, 2025	Mar, 30, 2025		
	Implement a display_stats() method to return a string or dictionary showing the current stats for the player.		Mar, 30, 2025	Mar, 31, 2025		
3. Matchmaking System Implementation	Outline the components and interactions of the matchmaking system (e.g., player queues, ranking system, pairing logic). Define data structures to hold player data, including their MMR and status (e.g., waiting, playing).		April, 1, 2025	April, 2, 2025		
	Implement logic that pairs players based on their Matchmaking Rating (MMR). Ensure players are matched with opponents of similar skill levels (e.g., within a range of 100 MMR points). Account for potential edge cases (e.g., when there are no available opponents with similar MMR).		April, 1, 2025	April, 2, 2025		
	Develop a system that adds players to a matchmaking queue when they are waiting for an opponent. Ensure players are automatically removed from the queue once they have been paired.		April, 3, 2025	April, 4, 2025		
	Establish thresholds for matchmaking, such as the minimum and maximum acceptable MMR difference for pairing players. Adjust matchmaking behavior based on these thresholds (e.g., narrow the acceptable MMR difference during high player volume times).		April, 3, 2025	April, 4, 2025		
	Implement timeout handling when players wait too long for a match (e.g., retrying the matchmaking process, adjusting MMR ranges).		April, 3, 2025	April, 4, 2025		
	Write unit tests and integration tests to ensure that the matchmaking system functions correctly. Test the system with a variety of player MMRs to ensure it returns appropriate pairings and works under different conditions (e.g., low or high traffic).		April, 4, 2025	April, 5, 2025		
4. Leaderboard System Implementation	Outline the structure of the leaderboard, specifying how player data will be stored and sorted. Define the data types (e.g., player ID, MMR, wins) that need to be stored for each player.		April, 1, 2025	April, 2, 2025		
	Develop the system to read from and write to a CSV file to store player stats. Ensure that player stats are updated in real time, reflecting wins/losses after each match.		April, 2, 2025	April, 3, 2025		
	Create a method for displaying the leaderboard in a user-friendly format (e.g., top 10 players with their MMR). Implement sorting options for the display (e.g., by MMR, by wins).		April, 4, 2025	April, 5, 2025		
	Implement error handling for missing or incomplete player data (e. g., missing MMR, incomplete games). Ensure that the leaderboard does not display incomplete player entries or misrank players due to missing data.		April, 4, 2025	April, 5, 2025		
	Ensure the leaderboard system is resistant to tampering or manipulation (e.g., fraudulent MMR manipulation). Implement security measures to ensure only authorized users can modify leaderboard data.		April, 4, 2025	April, 5, 2025		
5. Game-Specific Stats Classes Implementation	Each game-specific class (e.g., ChessStats, GoStats) should inherit from GeneralStats.		April, 5, 2025	April, 6, 2025		
	Implement the game-specific MMR adjustment logic in the update_mmr(self, win) method.		April, 5, 2025	April, 6, 2025		
6. Final Testing and Documentation	Test the entire system end-to-end to ensure all components (e.g., matchmaking, ranking, leaderboard, etc.) work together seamlessly. Verify that player stats, MMR, and game outcomes are correctly reflected across all components.		April, 6, 2025	April, 7, 2025		
	Identify and test edge cases such as incomplete player data, missing stats, or invalid input. Ensure the system gracefully handles unexpected inputs and errors, preventing crashes or incorrect behavior.		April, 6, 2025	April, 7, 2025		
	Ensure all code is properly commented to explain the logic behind complex functions or algorithms. Clean up any unnecessary or commented-out code before final submission.		April, 6, 2025	April, 7, 2025		