About Hitwicket

Greetings from Hitwicket**!**

Very rarely will you come across a role that is Global, Challenging and Mission driven to add 'fun' to the world! Imagine running into users from London to Dubai to Kochi using a product that you had a SIGNIFICANT role to play in.

We are here to provide an opportunity that brings out your creativity and make work as fun as playing a game.

The Global Gaming industry is worth $206 Billion, with Mobile Gaming itself accounting for more than **$100 Billion**! Mobile gaming is one of the few sectors that has continued the growth through the pandemic.

Gaming is a youth focused industry, where people like YOU are the target segment, so who better than you all to understand the needs and aspirations of the audience?

Therefore, unlike most sectors, in gaming you'll get to be a part of the decision making process even this early in your career.

✨ Hitwicket is a Series A funded Technology startup based in Hyderabad and co-founded by VIT alumni.

✨ Hitwicket won the First prize in **Prime Minister's AatmaNirbhar Bharat App Innovation Challenge**, a nation-wide contest to identify the top homegrown startups who are building for the Global market; Made in India, for India & the World!

✨ We recently onboarded **Mr. Harsha Bhogle** as our strategic investor.

✨ Cherry on the top? Even **Tim Cook** himself believes in our app. We were the **only gaming app** presented with the opportunity to meet and demo our app to him during his recent visit to India! This recognition further solidifies our position as a groundbreaking and influential force in the industry.

Our work culture is driven by speed, innovation and passion, not by hierarchy. Our work philosophy is centered around building a company like a 'Sports Team' where each member has an important role to play for the success of the company as a whole.

Learn, have fun and build a Game that millions of people love to play! Join us on this epic journey!

Note: *We are looking to evaluate your problem-solving skills. Please do not use AI tools to generate your responses. We employ a GenAI plagiarism detector that flags AI-generated content. Use of such tools may result in disqualification*

Need Help? Feel free to contact talent@hitwicket.com

Placements can be stressful and unfair, we've been through this ourselves back in the day. We'll do our best to give everyone a fair chance. We promise to listen and read EVERY application that is submitted.

Very few get an opportunity to work in a high-impact role early in their career. If this is what matters to you, then give it your best shot!

Turn-based Chess-like Game with Websocket Communication

Objective

Develop a turn-based chess-like game with a server-client architecture, utilizing websockets for real-time communication and a web-based user interface.

Components

1. Server

- Implement the game logic in any server-side language of your choice.
- Set up a websocket server to handle real-time communication with clients.
- Process game moves and maintain the game state.

2. Websocket Layer

- Implement a websocket communication layer between the server and clients.
- Handle events for game initialization, moves, and state updates.

3. Web Client

- Create a web-based user interface to display the game board and controls.
- Implement websocket communication with the server.
- Render the game state and provide interactive controls for players.

Game Rules

Game Setup

1. The game is played between two players on a 5x5 grid.
2. Each player controls a team of 5 characters, which can include Pawns, Hero1, and Hero2.
3. Players arrange their characters on their respective starting rows at the beginning of the game.

Characters and Movement

There are three types of characters available:

1. Pawn:
    - Moves one block in any direction (Left, Right, Forward, or Backward).
    - Move commands: L (Left), R (Right), F (Forward), B (Backward)

2. Hero1:
    - Moves two blocks straight in any direction.
    - Kills any opponent's character in its path.
    - Move commands: L (Left), R (Right), F (Forward), B (Backward)

3. Hero2:
    - Moves two blocks diagonally in any direction.
    - Kills any opponent's character in its path.
    - Move commands: FL (Forward-Left), FR (Forward-Right), BL (Backward-Left), BR (Backward-Right)

All moves are relative to the player's perspective.

Move command format:

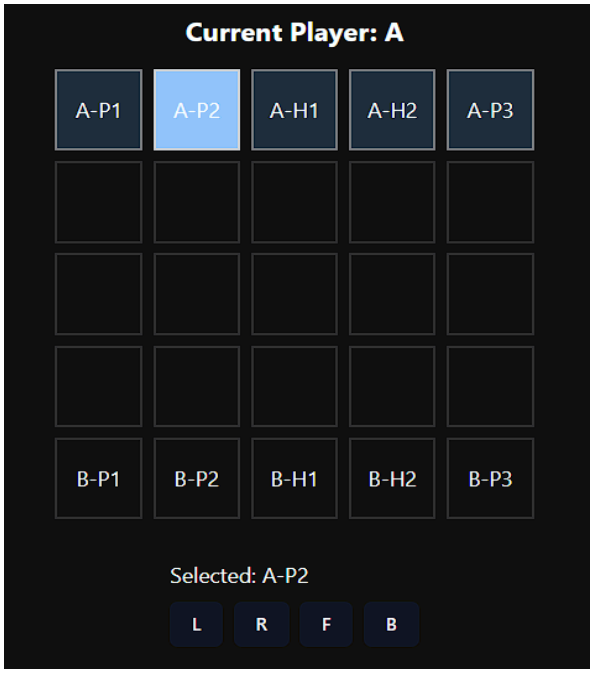
- For Pawn and Hero1: <character_name>:<move> (e.g., P1:L, H1:F)

- For Hero2: <character_name>:<move> (e.g., H2:FL, H2:BR)

Game Flow

Initial Setup:

- Players deploy all 5 characters on their starting row in any order.
- Character positions are input as a list of character names, placed from left to right.
- Players can choose any combination of Pawns, Hero1, and Hero2 for their team.



Turns:

- Players alternate turns, making one move per turn.

Combat:

- If a character moves to a space occupied by an opponent's character, the opponent's character is removed from the game.
- For Hero1 and Hero2, any opponent's character in their path is removed, not just the final destination.

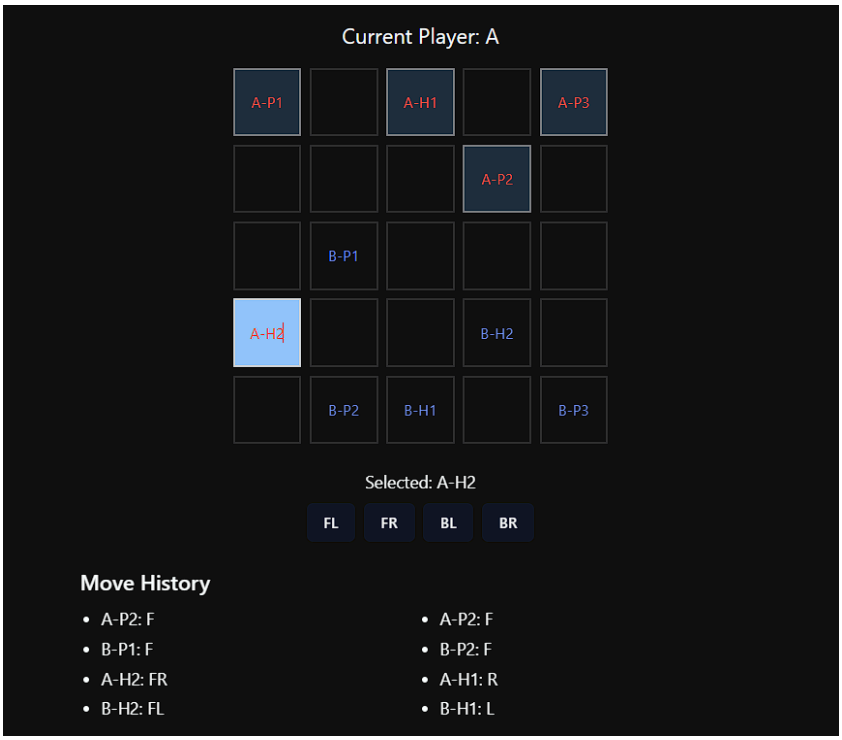
Invalid Moves:

- Moves are considered invalid if: a. The specified character doesn't exist. b. The move would take the character out of bounds. c. The move is not valid for the given character type. d. The move targets a friendly character.

- Players must retry their turn if they input an invalid move.

Game State Display:

- After each turn, the 5x5 grid is displayed with all character positions.
- Character names are prefixed with the player's identifier and character type (e.g., A-P1, B-H1, A-H2).



Winning the Game:

- The game ends when one player eliminates all of their opponent's characters.
- The winning player is announced, and players can choose to start a new game.

Technical Requirements

Server

- Implement the core game logic as described in the original rules.
- Set up a websocket server to handle client connections and game events.
- Process move commands received from clients and update the game state accordingly.
- Broadcast updated game state to all connected clients after each valid move.

Websocket Communication

- Implement the following event types:
    - Game initialization
    - Player move
    - Game state update
    - Invalid move notification
    - Game over notification

Web Client

- Create a responsive web page that displays:
    - A 5x5 grid representing the game board
    - Current game state with characters positioned on the board

    - Player turn indication
    - Move history (Optional)
- Implement interactive features:
    - Clickable character pieces for the current player
    - Display valid moves as buttons when a character is selected below the grid.
    - Send move commands to the server when a valid move is chosen
- Handle and display server responses, including invalid move notifications and game over states

User Interface Requirements

- Display the 5x5 game board with clear differentiation between empty cells and character positions.
- Use distinct visual representations for each player's characters (e.g., different colors or prefixes as in the original requirements).
- When a player selects their character, highlight valid move options as clickable buttons below the game board.
- Show the current player's turn prominently.
- Display a move history or log.
- Implement a game over screen showing the winner and offering an option to start a new game.

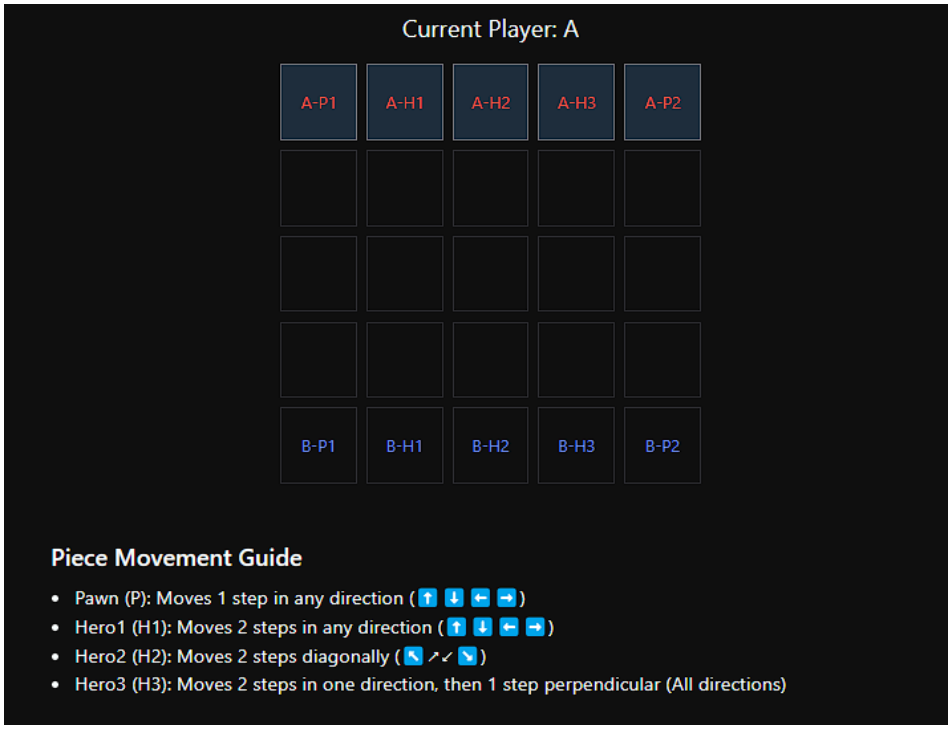
Implementation Steps

1. Set up the server with the core game logic.
2. Implement the websocket server and define the communication protocol.
3. Create the web client interface with any web framework.
4. Implement websocket communication in the client.
5. Integrate the game logic with the websocket layer on the server.
6. Develop the interactive features of the web client.
7. Implement game state rendering and updates on the client side.
8. Add final touches such as move validation, game over conditions, and the option to start a new game.

Bonus Challenges

1. Implement additional character types with unique move patterns.
    - Add Hero3:
        - Movement: Moves 2 steps straight and one to the side in a single turn.
        - Attack: Kills only the character at its final landing position (if occupied by an opponent).
        - Move commands:

        - FL: 2 steps Forward, 1 step Left
        - FR: 2 steps Forward, 1 step Right
        - BL: 2 steps Backward, 1 step Left
        - BR: 2 steps Backward, 1 step Right
        - RF: 2 steps Right, 1 step Forward
        - RB: 2 steps Right, 1 step Backward
        - LF: 2 steps Left, 1 step Forward
        - LB: 2 steps Left, 1 step Backward
    - Example moves: H3:FR (2 front, 1 right), H3:RF (2 right, 1 front)
2. Implement a dynamic team composition feature:
    - Allow players to choose their team composition at the start of each game.
    - Ensure the game logic can handle any combination of character types.
3. Add a spectator mode for other clients to watch ongoing games.
4. Implement a chat feature for players to communicate during the game.
5. Create an AI opponent that can play the game using basic strategy.
6. Implement a replay system that allows players to review past games move by move.
7. Add a ranking system that tracks player performance across multiple games.



When implementing Hero3, consider the following:

1. Movement Validation:
    - Ensure that the entire path of Hero3's movement is within the bounds of the 5x5 grid.
    - Check for any obstructions (friendly characters) along the path.
2. Combat Mechanics:

    - Only check for and eliminate opponent characters at the final landing position.
    - Do not eliminate characters that Hero3 passes over during its movement.
3. User Interface:
    - Update the UI to clearly show the available moves for Hero3 when selected.
    - Consider using arrow indicators or highlighting the possible end positions.
4. Game Balance:
    - Assess how the addition of Hero3 affects overall game balance.
    - You may need to adjust the number of each character type allowed per team.

By implementing Hero3, students will gain experience in handling more complex movement patterns and expanding their game logic to accommodate new rules. This challenge encourages students to think about modular design and extensibility in their code.

Acceptance Criteria and Guidelines

1. Code Organization:
    - The project must be in a single repository with clear separation between client and server code.
    - Include a README with setup and run instructions for both client and server.
2. Server-side Implementation:
    - Implement the core game logic as described in the Game Rules section.
    - Set up a websocket server to handle client connections and game events.
    - Process move commands and update the game state accordingly.
    - Implement thorough server-side move validation.
3. Client-side Implementation:
    - Create a basic web interface that displays the 5x5 game board.
    - Implement websocket communication with the server.
    - Implement client-side move validation (mirroring server-side validation).
    - Display valid moves for the selected character.
    - Send move commands to the server and handle responses.
4. Websocket Communication:
    - Implement event handling for game initialization, player moves, and game state updates.
    - Ensure real-time synchronization of game state between server and all connected clients.
5. Move Validation (both client and server-side):
    - Prevent selection or movement of opponent's pieces.
    - Ensure moves are within the 5x5 grid boundaries.
    - Validate moves according to each character type's movement rules.
    - Prevent friendly fire (moving onto or through spaces occupied by friendly characters).
    - Handle and communicate invalid move attempts to the user.

6. Edge Cases to Handle:
    - Simultaneous move attempts by multiple clients.
    - Disconnection and reconnection of clients during an ongoing game.
    - Attempts to make moves out of turn.
    - Handling of game state when a player quits mid-game.
    - Proper game termination when all opponent's pieces are eliminated.
7. Game Flow:
    - Implement turn-based gameplay with clear indication of current player's turn.
    - Correct handling of piece elimination upon valid capture moves.
    - Proper game end detection and winner announcement.
8. Code Quality:
    - Write clean, well-commented code following best practices for chosen technologies.
    - Implement error handling and logging for both client and server.

Note: While a polished UI is not the focus of this assignment, the client interface should be functional enough to demonstrate all required game interactions. The primary focus should be on implementing correct game logic, efficient network communication, and proper move validation on both client and server sides.

Bonus features (like Hero3 or AI opponents) are encouraged but not required for meeting the base acceptance criteria.

Create a repository in GitHub with your Name and VIT registration number and upload the link in the below attached form

---

And once you are done with the test, **CLICK HERE TO SUBMIT**

Deadline for submission: 12pm (noon), 26th Aug 2024

Feel free to reach out to us for any further queries on talent@hitwicket.com

---

Think Out of the Box, all the best!