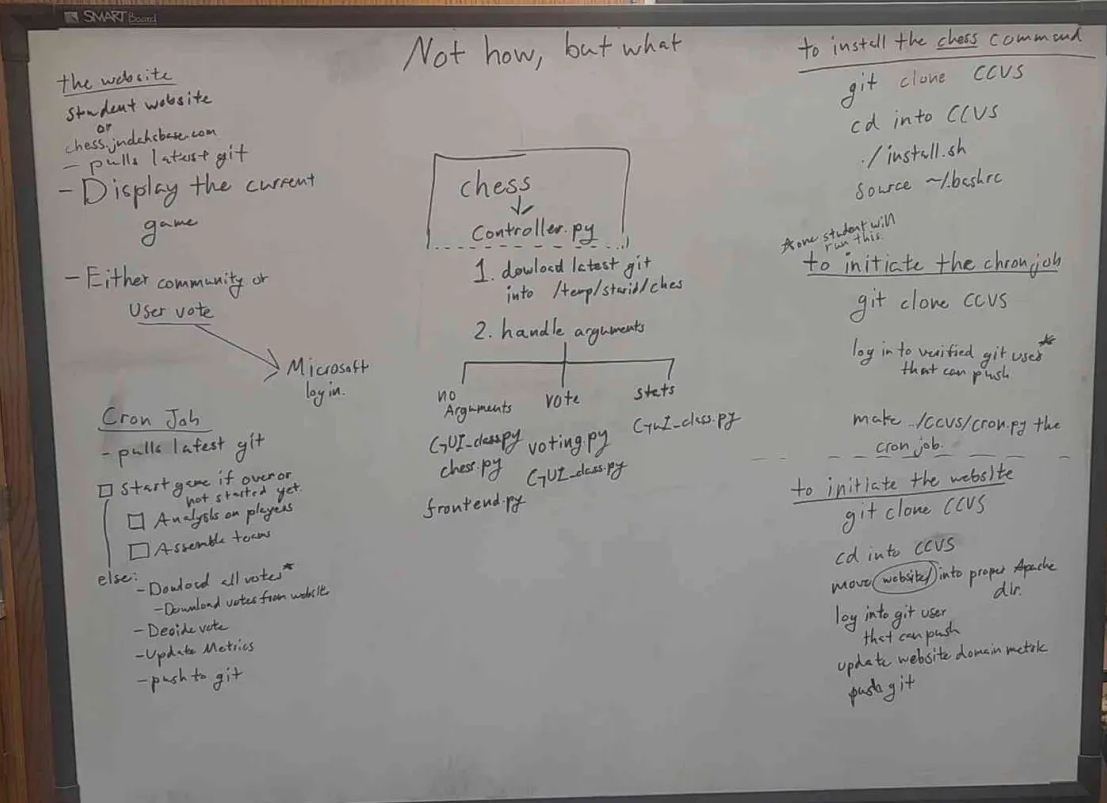
For our project, we have decided to document the architecture on a whiteboard, and here is the photo:

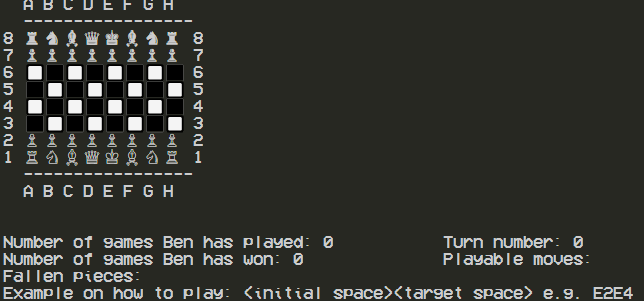


To explain it, we are using a website to display the current game and to let the people vote on there while being logged into their student account. For the Cron job, it pulls the latest git from our github, while ensuring it starts a new game or continues the old one with the moves made that people voted on.

For the chess game itself, it's a pretty simple chess implementation that makes use of a prebuilt chess engine in python. It has a custom-made display function for the board in the command line, and the game loop allows players to play until a lose condition is met. It uses the chess engines built-in function for checking if a game is over and also uses its built-in function for checking legal moves. For now, the interface displays the board, legal moves, whose turn it is, and some stats about the person logged in which will eventually tie into the statistics part of the software. All you need to run this part is a python interpreter and the chess library installed.

We will also be using authentification for players joining, so we can keep track of the most involved players, and to keep track of voting.

Here is a photo of our UI for the game with the statistics that will be displayed:



This is in our rough draft of the to-do list:

# Authentication:

1. Determine the integration method (using OAuth2 protocols maybe).
2. Build a simple module that authenticates a user and validates if its a student account.

# Game Logic:

1. Build the core classes (Board and Pieces) and game logic.
2. Render the board using Unicode (or something more visually friendly).
3. Document commands and how to vote/make moves (--help).

# Voting System Implementation:

1. Create an API or command interface for vote submissions.
2. Write the logic to count votes and select the winning move.
3. Decide on a tie-break mechanism (maybe random selection).

# Team Management & User Activity:

1. Choose between static and dynamic approach for teams (whether users will stay on the same team or switch to rebalance every day).
2. Implement a system to track the user activity and assign points based on participation (encourage activity).
3. save login times to check for “inactive after 48 hours” rule.

# Data Storage & GitHub Integration for Game Data:

1. Create a module to commit game state at 4:29 to GitHub (we can use a library like PyGitHub)
2. Store game data in a JSON file or a database (like SQLite?).
3. Implement a system to restore game state from the last commit (for restoring after changing game state or maybe a crash).

# Statistics and Analytics Module:

1. Create a module to track game statistics.
2. Implement a system to display user rankings and game statistics (turn number, team sizes, participation frequency).
3. Create a leaderboard to show the top players.
4. allow toggling of different stat panels (Optional).

Here is our architecture on the moving parts:

