EECS 448: Project 3 Design Paradigms

Team 20

Identify the design paradigm you (i.e., the team) chose for your prototype and explain why you chose that paradigm (250-350 words – 5%).

A few of the design paradigms we utilized during our project 3, were Object-Oriented Design, Component-Level Design, Event-Driven Design, and a Top-Down Functional Decomposition. To start, we used Object-Oriented Design in order to divide up our code into digestible pieces. For instance, we utilize a pieces.js file and what this accomplishes is containing all of the needed functions and variables needed for each piece on the board. This also ties into Component-Level Design given the fact that, the pieces class/file is used multiple times to create each piece and connect it to the board. This allows our project to be somewhat modular! By utilizing the pieces.js file we are able to not only interact with the pieces but as well as the board creating allowing this file to the bridge between our pieces object and the board on the screen. When it comes to Event-Driven design, we are checking for win conditions based on the event that one play no longer has any pieces left or no possible moves. Another aspect of this paradigm is capturing pieces, we are checking if captures are possible as well as if they can capture more than one piece! Finally connecting everything together is how we utilize the Top-Down Functional decomposition, everything that has been previously stated plays a role into this paradigm. The big picture is to be able to play checkers, so how do we do that? We break it up into subcategories such as pieces and boards in order to spread the work out. Once that is done, we can connect the pieces to complete the big picture of our program.

Within the context of your chosen design paradigm, describe the software architecture of your prototype (250-350 words – 5%).

Overall, I think the most fitting software architecture would be Peer-to-Peer for our prototype. I say this since we are currently not relying on a third-party server to hold any information P2P is the only architecture that fits. Within the context of our design paradigms, it makes sense that we would utilize a P2P architecture since it would allow for the fastest results. Since our project is checkers there is no need for an N-tier architecture or client-server communication since the project will be played on one laptop. All of this would allow for the most concise and fastest results from our project.

Use one or more of the UML modeling diagrams to design your prototype (5%).

Diagram

Description automatically generated

Identify the design patterns you used in your design and explain how you applied them (250-350 words - 5%).

Some of the design patterns we used throughout project 3 would consist of, Builder, Singleton, Composite, Mediators, and states! Starting with the Creation design patterns we utilized builders and singletons. The builders helped set up the board and make sure all of the pieces were accounted for, on top of that it would also validate moves and make sure errors wouldn’t be made. On the side of the singleton, we utilized it for containing the positions of the pieces, instead of having each piece be an individual class we have one class that has storage for the positions of the pieces! This combination was perfect for our project since, overall, it is not entirely too complex! Regarding the Structural design patterns, our team used composites to store needed functions and classes! To go more in depth about this pattern, especially in the pieces.js file is where this was used the most, that file kept track of variables and other important functions that are a requirement for this game to run smoothly. Finally, inside the Behavioral design patterns, we utilize mediators and states. In terms of mediators, we are checking for win conditions, pieces being captured, and even if a piece can become a queen if it reaches the other side. It simply stays and relays information that is received from the game to the other components of the project. Lastly, we use states to determine who’s turn it is as well as who won, if the state is on player one and he captures the player two’s last piece then player one would have to win. By utilizing states we are able to keep the game straightforward and without error.