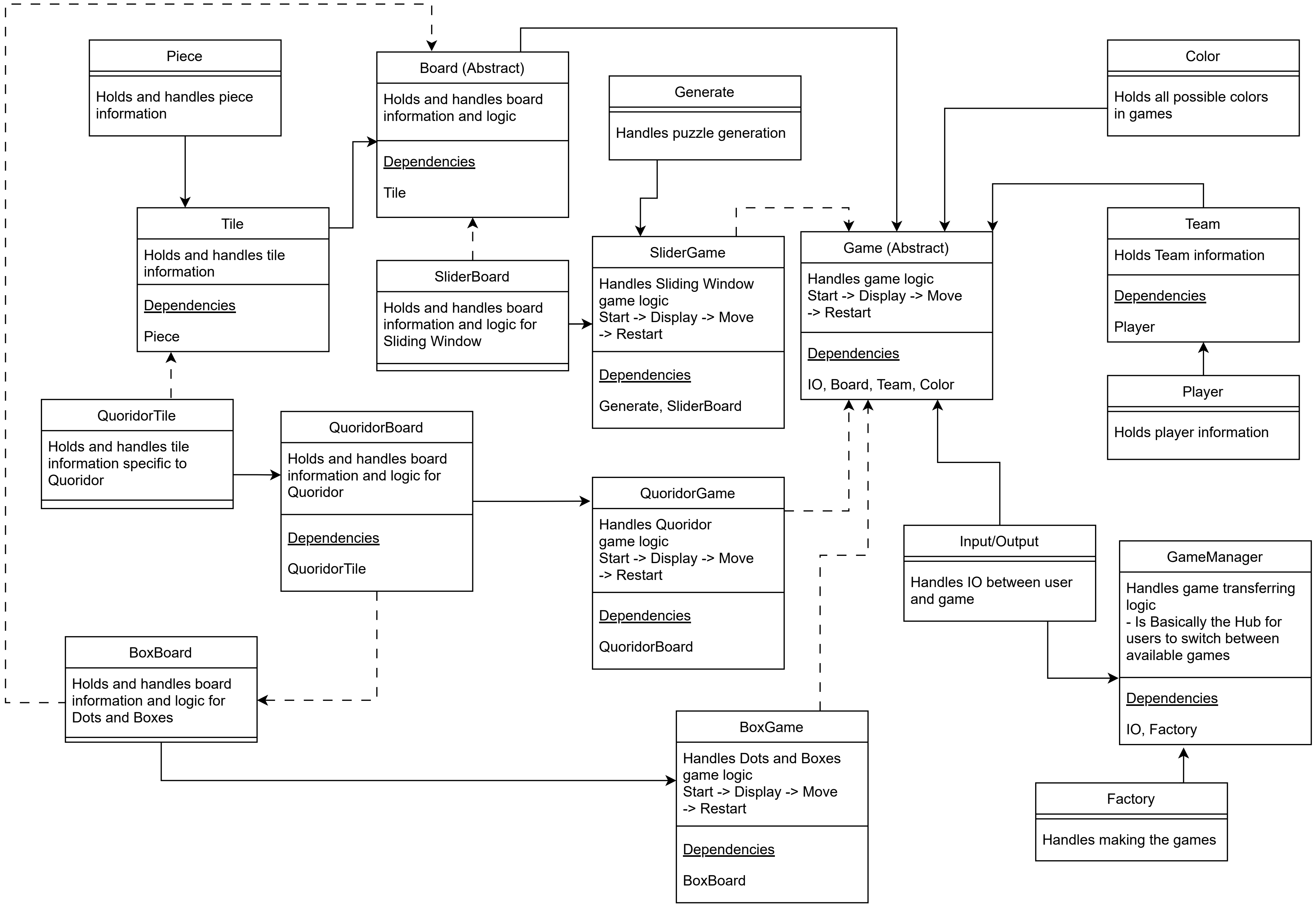
Piece
Holds and handles piece information
Board (Abstract)
Holds and handles board information and logic
Dependencies
Tile
Generate
Handles puzzle generation
Color
Holds all possible colors in games
Tile
Holds and handles tile information
Dependencies
Piece
SliderBoard
Holds and handles board information and logic for Sliding Window
SliderGame
Handles Sliding Window game logic
Start -> Display -> Move -> Restart
Dependencies
Generate, SliderBoard
Game (Abstract)
Handles game logic
Start -> Display -> Move -> Restart
Dependencies
IO, Board, Team, Color
Team
Holds Team information
Dependencies
Player
Player
Holds player information
QuoridorTile
Holds and handles tile information specific to Quoridor
QuoridorBoard
Holds and handles board information and logic for Quoridor
Dependencies
QuoridorTile
QuoridorGame
Handles Quoridor game logic
Start -> Display -> Move -> Restart
Dependencies
QuoridorBoard
Input/Output
Handles IO between user and game
GameManager
Handles game transferring logic
- Is Basically the Hub for users to switch between available games
Dependencies
IO, Factory
BoxBoard
Holds and handles board information and logic for Dots and Boxes
BoxGame
Handles Dots and Boxes game logic
Start -> Display -> Move -> Restart
Dependencies
BoxBoard
Factory
Handles making the games

Our class structures all starts from an abstract Board and Game class which this extends to all combinations of board games. We implement a GameMangager class that is connected to a factory class who based on the board game chosen by the users via the displayed menu options. In this way, we can make changes in the Factory without disrupting another other component of the program. Hence, adding the Quoridor board game to the games doesn't affect the existing structure of the program. We decided to have a Constants class which houses all general constants the overall program needed in order to keep from having redundant constants floating around. Note, a board's width and height are not part of this class, as given the users' decisions, these variables changes indefinitely. In order to minimize dependencies on specific IO interactions with users, we decided to change the IO class into three parts: more general integer, String, and Boolean queries that have given prompts and bounds; prompts for team creation such as color and names; ending stats displays. This way, there is no need to create special instances for the IO, and the IO instance object can be passed throughout the entire program, starting from the GameManager who directs users to their chosen game.

In Dots and Boxes, a turn variant was already implemented that relied on an index which incremented every other turn to cycle through the list of Players in each Team, prompting them to play their turn of the board game. Overall, because the actions of the program vary between the indexing, we believe this is no great need to abstract a single game-dependent variable out of each individual game the user decides to play. However, the components surrounding a multiple piece, multiple player game can be useful in multiple different board games. Hence the reason for extending QuoridorBoard out of BoxBoard in order to use the same method to color vertical and horizontal lines on the board.

The only changes we've made to the base program structure are cleaning out special IO methods built up from previous game implementations into a more clean and general IO structure that all games can use without a need for modification. We also expanded the amount of methods for the Game abstract class so that the entire game logic is not just in the Start method. Specifically, we pulled out the inner game loop, with calls to move checking logic with in the Board abstract class, into a separate gameLoop method. Within the QuoridorBoard class, we abstracted the move checking logic from the move choice logic. Meaning we use the base methods to make a forked path that allows for two different methods to check for a valid pawn move verus a valid wall placement. Lastly, for aesthetic and user clarity reasons, we added in Tile numbering and Tile coloring to indicate Team turns and possible Player moves.