- Iterator
    - Checkers_iterator
        - Iterates through non captured pieces of a specific color. When called checkers_iterator, the iterator returns only checks if a piece is captured and does not check the identity (man, king) of the piece. Thus, this iterator can be extended to work with chess pieces as long as each piece implements a boolean indicating whether they are captured. Also makes finding all moves for a player easy (check moves for each piece returned).
- Abstract Factory
    - Player_factory
        - Creates a player based on command line arguments. Simplifies the process of creating players, and as each player only interacts with move command and not pieces directly, both the player class and the player_factory are game agnostic.
    - Checkers_factory
        - The board.set_up() method calls the piece factory on every single board position. The piece factory handles the logic of where to place checker pieces and of what color, allowing the board class to remain game agnostic.
- Command
    - Move_command
        - This class is the only class that directly interacts with the methods of the checkers_piece class, but even then, most of the behavior of the pieces are left up to the piece class. Subclasses allow specific types of moves by each type of player. This keeps both the board and players game agnostic. Undo/redo is also simplified by maintaining a stack of move_commands.
- Template:
    - Checker_piece:
        - This class handles most of the actual checkers game rules such as where pieces can move, promotions, demotions(for undo), etc. Localizing most game-specific behavior to the pieces keeps most of the rest of the code game agnostic. This class uses a template pattern as only a few specific methods are overwritten for piece vs man, but this number is expected to be much higher for chess. Thus, implementing with a template pattern early allows for more flexibility later on. Additionally, using a template pattern allows the classes that interact with checkers_piece to use the same interface regardless of whether a piece is a man or a king.
    - Players:
        - All players are very diverse in their abilities while keeping the same “core” function: “making a move”. As only the way in which the move is made differs from player to player, it makes sense to use a template pattern and selectively override the way each move is made in subclasses. Thus,

interfaces with players in the main driver loop can be kept consistent regardless of players.