- **15 points. Submit a brief write-up (upper limit of 600 words) describing how you made use of design patterns we talked about in class. I expect you will use at least 4 distinct design patterns (and maybe some more than once), but you're welcome to include more as long as they are appropriate to the task. Iterator counts, but only if you have created your own custom iterator that does something more than just stepping through a list.**

1) The Strategy class is an abstract base class that declares the generate_make_and_build method, which all concrete strategy classes must implement. This method is used to generate the moves and builds in the game.

   There are three concrete strategy classes that inherit from Strategy:

   HumanStrategy: strategy for a human player interacting with the game
   RandomStrategy: strategy for a random AI interacting with the game
   HeuristicStrategy: strategy for a random AI interacting with the game but based on heuristic values that determines the actions

   The Context class holds a reference to a Strategy instance and delegates the execution of the behavior to the generate_make_and_build method of the strategy.

   The context_interface method is called by the clients, which in turn calls the strategy's method to generate moves and builds.

   The Player class, from which HumanPlayer, RandomPlayer, and HeuristicPlayer are derived, using the Context class to set its strategy according to the type of player. Then, the player_move method uses the current strategy context to execute the move and build actions during the game.

2) The FactoryofPlayers class acts as a factory for creating Player instances with the appropriate strategy based on the player type. It would use these strategy classes to instantiate player objects with the correct behavior.

   The FactoryofPlayers class encapsulates the logic required to create different types of Player objects. This allows for the creation of player objects without exposing the creation logic to the client and without the client needing to know the exact class that needs to be instantiated.

   This class defines a method initiate_player that determines the type of player to instantiate based on the specified player_type. By passing parameters like board, colour, and the type of player, the factory method abstracts the instantiation logic and returns a new Player object suited to the game's needs. This pattern allows the game to introduce new types of players with unique strategies without altering the underlying system that utilizes them.

3) The Memento pattern in the Caretaker class allows the game to record its state at various points, enabling the player to move back and forth within these states without compromising the internal state of the game components.

   In Caretaker class, the Memento pattern is applied to manage the states of a game, which allows players to undo or redo a turn

- Initialization: The Caretaker class is initialised with the initial state of the game stored in _game_state_mementos list.
- Save: The save method is used to store a memento of the game's current state. Before saving a new state, the method checks if any 'future' states need to be discarded, which happens if the player has used the undo function and then makes a new move.
- The undo method lets a player step back to an earlier game state, while redo permits advancing to a state previously undone, navigating through stored states using _current_index.

4) Command Patttern:

   MoveandBuild class, the Command pattern is used to encapsulate all the information needed for moving and building actions in the game into a single object. This object (MoveandBuild) defines the move and build methods that encapsulate the actions and the corresponding data:

- Initialization: The constructor initializes the object with the board, the active worker, and the directions for moving and building.
- The Move method in the MoveandBuild class represents the execution of a command, handling the worker's movement on the board.
- The Build method follows suit, managing the construction of a structure on the board. Both methods execute their respective actions based on the given directions, updating the game state accordingly.
- Execution: The move and build methods represent the execution of the command. When called, they carry out the necessary operations to alter the game's state