Explanation of Design Patterns Used

Stratego Project, OOP w/ Dr. Basnet

State machine (GUI)

Depending on the current state of the GUI and the user input provided, the GUI will transition to a new state. The state of the GUI is provided by the `screen` property. The values of this property are as follows:

- ” (empty string) - The GUI has not initialized yet
- ‘HOME’ - The title screen, where you can choose to host or join a game.
- ‘INFO’ - Displays info about the project, as well as the authors.
- ‘WIN’ - The screen which is shown when you win the game.
- ‘LOSE’ - The screen which is shown when you lose the game.
- ‘ERROR’ - The screen which is shown upon a networking failure.
- ‘SETUP’ - The screen where the user can setup their pieces.
- ‘HOST_GAME’ - Asks the user about hosting information, like IP address and port number.
- ‘JOIN_GAME’ - Asks the user about the host machine, like IP address and port number.
- ‘YOUR_TURN’ - Waits for the user to make a move. If this move is valid, it will send the updated game state to the other player. Otherwise, the game state will be reverted and “invalid move” will be displayed.
- ‘THEIR_TURN’ - Awaits the reception of the other player’s move.

Logical transitions occur depending on this state and the input, as well as the logical rules of the game.

Singleton (GUI, Board, Networking)

It does not make sense for there to be more than one instance of the GUI or the Stratego game board operating at once. This is especially true for the networking manager object; If this were to be duplicated it could cause confusion and failures in networking, as one instance could receive data meant for the other. Thus, the GUI, networking manager and board classes are all defined as singletons. Since we need to have several instances of the GUI for manual testing, our testing process involves launching two instances of python running the program, rather than two GUI’s running in the same instance of python.

Decorator (type checking, class management)

The hypothesis method of unit testing requires the use of decorators, as does the designation of class and static class methods. This pattern, then, is used throughout our code. Additionally, we used many `property` methods in our classes, each of which is described via the use of the `@property` decorator.

Command (Board Square Callbacks)

The `tkinter` module provides only a basic callback system, so in order to implement a functional board system in the GUI we needed to implement a command structure. This structure, `StrategoGUI.ButtonCallbackWrapper`, is a callable object which stores information about the button it is associated with. This allows the GUI to know where the click occurred on the board, and modify the game state accordingly.