**Board and Display Testing:**

**Board Component tests:**

**Description:**

The central component of the architecture for the actual play of the game is the Board component. It is responsible for maintaining the overall board state, directing the flow of the game, and communicating that state and flow to the other components. It is expected that most other components communicate with this component. As such, the Board component is responsible for tracking the teams and their pieces and ensuring that each play is valid according to the game rules, and that the change to the state of the board is propogated to the Display and Logger components. The Board component is also responsible for representing the Board and expressing the state of the Board according to this representation. Looked at from an MVC architectural perspective, the Board has some of the functions of a Controller, but also some aspects of the Model. One of the primary inputs to the Board component comes from the Event Handler component.

The Board component will be tested to ensure that a piece is correctly moved, all pieces occupying a shot space are dealt the correct damage, the next active piece is correctly determined, scanning a space or area is performed correctly, spaces are correctly converted between their absolute and relative representations, and a space is correctly determined to be inside or outside of the board’s boundaries. These tests includes testing the correct piece is moved, pieces not occupying a shot space remain undamaged, the next active piece is alive, only spaces within range are scanned, converting to one representation and back gives the correct result, and testing spaces just within bounds, the bounds themselves, and spaces considered to be out of bounds. Extensive tests will also be conducted on the reduce method of the board class, which takes a hexagon coordinate and reduces it to its minimal representation. Testing the reduce method will include testing correct and incorrect hexagon coordinate inputs and testing sensitive hexagon coordinate inputs.

**Board Class tests:**

**Description:**

The Board class is designed to model the game board. The Board class stores an array of the coordinates of the hexagon spaces at which the game pieces are located, an array of the coordinates of the hexagon spaces that are the boundary limits of the game board, an array of the game pieces, and array of the teams, the enumeration of the current game piece that is active, the logger, and the display. The Board class not only stores the necessary coordinates and pieces, but also manipulates the position of each piece according to direct instruction from the event handlers while also checking that each action is valid. The board component notifies the Logger and the Display after each action so that each may update according to how the game board state has changed and which actions have been performed.

**Significance:**

The Board class is significant because it directs the flow of the game, stores the overall state of the game board, and updates the Display and Logger. Also, the Board class is centralized in that it is the communication link between most of the other components for game updates within the system. It is important to assert that the Board class functions correctly to ensure the state of the game board remains stable and correct, other components receive correct updates, the flow of the game is directed correctly and smoothly, and the game rules are adhered to.

**Test Cases:**

* **Board():**
  + **Test Case:** Initialize the Board.

**Expected Result:** The correct number of teams and pieces are initialized and Each team has the correct number and type of pieces. The Logger is initialized and present. The Display is initialized and present.

* **movePiece(PieceEnum piece, HexCoord coord):**
  + **Test Case:** The coordinate represents a valid coordinate within bounds.

**Expected Result:** The piece is moved to the coordinate.

* + **Test Case:** The coordinate represents a valid coordinate not within bounds.

**Expected Result:** The piece is not moved at all and the error is handled.

* + **Test Case:** The coordinate does not represent a valid coordinate.

**Expected Result:** The piece is not moved at all and the error is handled.

* + **Test Case:** The piece parameter represents a valid piece.

**Expected Result:** The correct piece is moved and no other piece is.

* + **Test Case:** The piece parameter does not represent a valid piece.

**Expected Result:** No piece is moved and the error is handled.

* + **Test Case:** The travel distance is less than the piece’s remaining mobility points.

**Expected Results:** The piece is moved to the coordinate.

* + **Test Case:** The travel distance is equal to the piece’s remaining mobility points.

**Expected Results:** The piece is moved to the coordinate.

* + **Test Case:** The travel distance is greater than the piece’s remaining mobility points.

**Expected Results:** The piece is not moved and the error is handled.

* + **Test Case:** A valid move is performed.

**Expected Results:** The distance is subtracted from the piece’s mobility points.

* + **Test Case:** An invalid move is performed.

**Expected Results:** The piece’s mobility points remain unchanged.

* + **Test Case:** Perform two valid move operations follow by a third on the same piece.

**Expected Results:** The first two moves are performed correctly, but the third is not and the error is handled.

* **shootSpace(PieceEnum piece, HexCoord coord):**
  + **Test Case:** Shoot a space inside the range of the piece’s range points.

**Expected Results:** The shot is performed correctly.

* + **Test Case:** Shoot a space outside the range of the piece’s range points.

**Expected Results:** No piece’s are damaged and the error is handled.

* + **Test Case:** Shoot an unoccupied space.

**Expected Results:** No piece is dealt any damage.

* + **Test Case:** Shoot a space occupied by one piece.

**Expected Results:** Only the piece occupying the space is dealt the correct damage.

* + **Test Case:** Shoot a space occupied by more than one piece.
  + **Expected Results:** All pieces occupying the space are dealt the correct damage.
  + **Test Case:** The piece shoots are second time following a first shot.

**Expected Results:** The shot is not performed and the error is handled.

* **nextPiece():**
  + **Test Case:** Two rounds of game play are simulated for 2, 3, and 6 teams.

**Expected Results:** The next active piece is correctly determined each turn.

* + **Test Case:** A piece no longer alive is next in line to be active.

**Expected Results:** The piece is skipped and the next piece is correctly determined.

* + **Test Case:** The next team does not posses any live pieces.

**Expected Results:** The next active team and piece is correctly determined.

* **scanArea(PieceEnum piece, HexCoord coord):**
  + **Test Case:** The coordinate represents a valid coordinate within bounds.

**Expected Results:** The correct scan is performed.

* + **Test Case:** The coordinate represents a valid coordinate outside bounds.

**Expected Results:** The scan is not performed and the error is handled.

* + **Test Case:** The coordinate represents a valid coordinate outside bounds.
  + **Expected Results:** The scan is not performed and the error is handled.
  + **Test Case:** The coordinate represents an invalid coordinate.

**Expected Results:** The scan is not performed and the error is handled.

* + **Test Case:** Perform a valid scan.

**Expected Results:** The scan returns coordinates that are occupied.

* + **Test Case:** Perform a valid scan.

**Expected Results:** The scan does not return coordinates that are not occupied.

* + **Test Case:** Perform a valid scan.

**Expected Results:** The scan does not return coordinates outside the piece’s range.

* **scanSpace(HexCoord coord):**
  + **Test Case:** The coordinate represents a valid coordinate within bounds.

**Expected Results:** The correct scan is performed.

* + **Test Case:** The coordinate represents a valid coordinate outside bounds.

**Expected Results:** The scan is not performed and the error handled.

* + **Test Case:** The coordinate represents an invalid coordinate.

**Expected Results:** The scan is not performed and the error handled.

* + **Test Case:** The space is unoccupied.

**Expected Results:** The scan returns an empty list.

* + **Test Case:** The space is occupied by one piece.

**Expected Results:** The scan returns a list of only the piece occupying the space.

* + **Test Case:** The space is occupied by more than one piece.

**Expected Results:** The scan returns a list of only all pieces occupying the space.

* **absoluteToRelative(HexCoord coord)** and **relativeToAbsolute(HexCoord coord):**
  + **Test Case:** The coordinate represents a valid coordinate within bounds.

**Expected Results:** The correct conversion is performed.

* + **Test Case:** The coordinate represents a valid coordinate outside bounds.

**Expected Results:** The conversion is not performed and the error handled.

* + **Test Case:** The coordinate represents an invalid coordinate outside bounds.

**Expected Results:** The conversion is not performed and the error handled.

* + **Test Case:** Convert a relative coordinate to an absolute coordinate.

**Expected Result:** The correct absolute coordinate is returned.

* + **Test Case:** Convert an absolute coordinate to a relative coordinate.

**Expected Result:** The correct relative coordinate is returned.

* + **Test Case:** Convert a relative coordinate to an absolute coordinate and back to a relative coordinate.

**Expected Result:** The original relative coordinate is returned.

* + **Test Case:** Convert an absolute coordinate to a relative coordinate and back to an absolute coordinate.

**Expected Result:** The original absolute coordinate is returned.

* **isInBounds(HexCoord coord):**
  + **Test Case:** The cooordinate represents a valid coordinate within bounds.

**Expected Result:** Returns true.

* + **Test Case:** The cooordinate represents a valid coordinate outside bounds.

**Expected Result:** Returns false.

* + **Test Case:** The cooordinate represents an invalid coordinate.

**Expected Result:** The error is handled.

* **PieceEnum:**
  + Each enumeration correctly represents a unique piece.
  + Each piece is enumerated.
  + The enumerations are well defined.

**HexCoord Class tests:**

**Description:**

The HexCoord class is designed to model the hexagon space elements on the game board. The HexCoord class is used to refer to the actual hexagon spaces on the game board and for storing the coordinate positions of the game pieces. The HexCoord class will also be used throughout the system when it is necessary to refer to a hexagon space on the game board, such as within the display component to update the visual display of a game piece.

**Significance:**

The HexCoord class is significant because it represents the coordinates of the hexagon spaces on the game board, but moreso because the reduce method of the HexCoord class is designed to reduce the x, y, and z coordinates down to their minimal representation for simplification, which must function correctly for the rest of the implementation to work.

**Test Cases:**

* **Setters and Getters:**
  + **Test Case:** Perform each getter method.

**Expected Result:** Each getter returns its correct value.

* + **Test Case:** Perform each setter method.

**Expected Result:** Each setter sets its respective value.

* **reduce():**
  + **Test Case:** Peform the reduce method on a coordinate representing the center.

**Expected Result:** The center is correctly reduced.

* + **Test Case:** Peform the reduce method on coordinates just beyond the center.

**Expected Result:** The coordinates are correctly reduced.

* + **Test Case:** Peform the reduce method on coordinates between the center and boundaries.

**Expected Result:** The coordinates are correctly reduced.

* + **Test Case:** Peform the reduce method on coordinates just before the boundaries.

**Expected Result:** The coordinates are correctly reduced.

* + **Test Case:** Peform the reduce method the boundary coordinates.

**Expected Result:** The cpordinates are correctly reduced.

* + **Test Case:** Peform the reduce method on a coordinate outside the boundries.

**Expected Result:** The coordinate is not reduced and the error handled.

**Display Component Tests:**

**Description:**

The Display component is responsible for the user interface (UI) elements of the game. It primarily takes information from the Board and displays that information to the user. The Display, along with the Flow, would be the View component of an MVC architecture. Even more than the Pieces component which appears next, the Display is a passive component and does not send information out to other components.

The Display component will be tested to determine that it correctly initializes the various screens, correctly switches between screens, that each screen is effective and accurate, and that each of the screen buttons performs their intended operation.

**Display Class tests:**

**Description:**

The Display class is designed to model the Display component. The Display class stores an array of all the screens, which can be easily indexed using the ScreenEnum enumeration that is assigned to each of the screens of the Screen class. The public switchToScreen() method takes in the enumeration of the screen to switch to and then updates the Display component to display the specified screen. The private initializer methods are used to initialize the various screens held by the Display. The Display also provides an accessor to retrieve the GameScreen to allow easier updating of the GameScreen.

**Significance:**

The Display component is significant because it is responsible for visualizing the state and actions of the system for the user. It is important to ensure correct visualization of the system so the user can interact with and use the system. The Display is also the only visually appealing component within the system. If the Display is not visually appealing, and does not display the data and actions adequately, the system will not succeed. As such, it is imperative to determine that each Screen within the Display is correctly initialized, is visually appealing, and displays the data and actions in a way that is easily interpreted.

**Test Cases:**

* **Display(Logger logger) and initializer methods for each Screen:**
  + **Test Case:** Inititialize the Display when the logger is null.

**Expected Result:** The Display is not initialized and the error handled.

* + **Test Case:** Inititialize the Display when the logger is not null.

**Expected Result:** The Display is initialized smoothly and correctly.

* + **Test Case:** Initialize each Screen with its respective initializer method.

**Expected Result:** The Screen is initialized correctly and smoothly.

* + **Test Case:** View each Screen after initialization.

**Expected Result:** Each Screen is appealing and conveys its data well.

* + **Test Case:** Click the buttons of each Screen.

**Expected Result:** Each button performs the intended operation.

* **switchTo(ScreenEnum screen):**
  + **Test Case:** screen does not represent a valid screen.

**Expected Result:** The Display does not change and the error handled.

* + **Test Case:** screen represents a valid screen.

**Expected Result:** The Display changes to the correct screen.

* + **Test Case:** Peform the method for each enumeration with a time delay between invocations.

**Expected Result:** The Display switchs to each Screen in the correct sequence with the correct delay.

* **getGameScreen():**
  + **Test Case:** A game is in progress.

**Expected Result:** The game screen is returned.

* + **Test Case:** A game is not in progress.

**Expected Result:** Null is returned.

* **ScreenEnum:**
  + Each enumeration correctly represents a unique screen.
  + Each Screen is enumerated.
  + The enumerations are well defined.

**Screen Class tests:**

**Description:**

The Screen class has been designed to model a screen within the system. Each screen will have a specific layout, but the layout will be initialized by the display component. Each screen will be assigned an enumeration that will be stored and for which there is an accessor. Each screen will use the Swing library for its display components, and will inherit from JPanel and holds all of the display elements for that screen, including buttons, images, and other control widgets.

**Significance:**

The Screen class is significant because it defines the basis of each screen. The Screen class must work correctly and guarantee to be displayed or no Screen within the Display component will be displayed correctly.

**Test Cases:**

* **Screen(ScreenEnum screen)**
  + **Test Case:** Perform the initialization.

**Expected Result:** The Screen is correctly initialized with the correct enumeration.

* **getEnum():**
  + **Test Case:** Perform the method for each Screen.

**Expected Result:** The correct enumeration for each Screen is returned.

**GameScreen Class tests:**

**Description:**

The GameScreen class inherits from the Screen class and has been designed to model the game screen. It contains all elements that a Screen does, including its ScreenEnum value and the Panels it contains. A collection of images for the game pieces will be stored in an array that will be indexed using enumerations where applicable. The x and y offsets of the board will be stored to allow the correct portion of the board to display when the Player pans the board. Also, the log screen and the context menu that will be shown to the Player to allow the Player to select their action are be stored.

**Significance:**

The GameScreen is significant because it displays the game’s state and actions to the players. This Screen is the most important of all the Screens because this Screen is where the game is visualized. Every action within the game is visualized here. Much of the user’s time and concern will be focused in this Screen, so this Screen must function correctly and effectively. If the game is not visualized correctly or effectively, the system will not succeed.

**Test Cases:**

* **GameScreen():**
  + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The necessary images are initialized.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The teams-bar is correctly positioned an initialized with the correct ordering and coloring of teams.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The log screen is correctly initialized and positioned.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The context menu screen is correctly initialized.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The game board is correctly initialized and positioned.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The game piece images are correctly initialized and positioned.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The side buttons are correctly initialized and positioned.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The end turn button is correctly initialized and positioned.

* + **Test Case:** Perform the construction for 2, 3, and 6 teams.

**Expected Result:** The quit button is correctly initialized and positioned.

* + **Test Case:** Perform click each button on the GameScreen.

**Expected Result:** Each button performs its intended operation.

* **update():**
  + **Test Case:** Change the location of a piece.

**Expected Result:** The piece is updated to the correct position.

* + **Test Case:** Pan the board.

**Expected Result:** The board is correctly panned.

* + **Test Case:** Switch the active team and piece.

**Expected Result:** The top teams-bar, active piece, board focus, and board visibility is updated correctly.

* **movePiece(PieceEnum piece, HexCoord coord):**
  + **Test Case:** A valid move is performed.

**Expected Result:** Only the piece parameter is visually moved to the coordinate and no other pieces are moved.

* + **Test Case:** The coordinate represents a valid hexagon coordinate within bounds.

**Expected Result:** The piece is visually moved to the coordinate.

* + **Test Case:** The coordinate represents a valid hexagon coordinate outside bounds.

**Expected Result:** The piece is not visually moved and the error handled.

* + **Test Case:** The coordinate represents an invalid hexagon coordinate.

**Expected Result:** The piece is not visually moved and the error handled.

* **shootSpace(PieceEnum piece, HexCoord coord):**
  + **Test Case:** A valid shot is performed.

**Expected Result:** Only the piece parameter shoots the coordinate and no other shots occur.

* + **Test Case:** The coordinate represents a valid hexagon coordinate within bounds.

**Expected Result:** A shot is visually fired from the piece to the coordinate.

* + **Test Case:** The coordinate represents a valid hexagon coordinate outside bounds.

**Expected Result:** A shot is not visually fired and the error handled.

* + **Test Case:** The coordinate represents an invalid hexagon coordinate.

**Expected Result:** A shot is not visually fired and the error handled.

* **switchPlayer(PlayerEnum player, PieceEnum piece):**
  + **Test Case:** Perform a valid switch.

**Expected Result:** The GameScreen is blacked-out, initializes itself for the new player, prompts the player to begin their turn, and the GameScreen is correct when the player acknowledges its their turn.

* + **Test Case:** The piece parameter belongs to the team parameter.

**Expected Result:** The switch visually occurs correctly.

* + **Test Case:** The piece parameter does not belong to the team parameter.

**Expected Result:** The switch does not occur and the error handled.

* + **Test Case:** Perform a valid switch for a live team.

**Expected Result:** The top teams-bar is updated correctly.

* + **Test Case:** Perform a valid switch for a dead team.

**Expected Result:** The GameScreen is not updated and the error handled.

* + **Test Case:** Perform a valid switch for a live piece.

**Expected Result:** The game board focus and visibility is updated correctly.

* + **Test Case:** Perform a valid switch for a dead piece.

**Expected Result:** The GameScreen does not update and the error handled.

* + **Test Case:** Perform a valid switch for a live piece and dead piece.

**Expected Result:** The game board and piece are positioned correctly.

* + **Test Case:** Perform a valid switch.

**Expected Result:** The game board and active piece are positioned correctly.

* + **Test Case:** Perform a valid switch.

**Expected Result:** The log is correctly updated for the new team.

* + **Test Case:** Perform an invalid switch.

**Expected Result:** The log is not updated and the error handled.

* **updateLog(TeamEnum team):**
  + **Test Case:** The team is a correct team.

**Expected Result:** The log is updated for the correct team.

* + **Test Case:** The team is not a correct team.

**Expected Result:** The log is not updated and the error is handled.

* + **Test Case:** Perform a valid log.

**Expected Result:** The log data belongs to the team.

* **pan(int deltaX, int deltaY):**
  + **Test Case:** The deltaX and deltaY keep the board position within the limits.

**Expected Result:** The pan is correctly performed according to the changes.

* + **Test Case:** The deltaX and deltaY push the board position outside the limits.

**Expected Result:** The pan is correctly performed only up to the position limits.

* + **Test Case:** Perform a valid pan.

**Expected Result:** The change in the board’s visual position remains after panning.

* **showContextMenu(), promptEndTurn(), promptQuit(), and showTransition():**
  + **Test Case:** Invoke showContextMenu().

**Expected Result:** The context menu is shown, sends the correct event upon a choice being clicked, and disappears.

* + **Test Case:** Invoke promptEndTurn().

**Expected Result:** The end turn prompt is shown, sends the correct event upon a choice being clicked, and disappears.

* + **Test Case:** Invoke promptQuit().

**Expected Result:** The quit game prompt is shown, sends the correct event upon a choice being clicked, and disappears.

* + **Test Case:** Invoke showTransition().

**Expected Result:** The GameScreen turns black for the transition and the prompt prompting the next team to acknowledge it is their turn appears and disappears after the team acknowledges.

* + **Test Case:** Click each button.

**Expected Result:** Each button performs its intended operation.

**Proposed changes:**