API Documentation

Stratego project, OOP w/ Dr. Basnet

GUI

`ScreenType`

The state of the GUI state machine. This, along with user input, will be used to determine the next thing to do. Each of these options is a screen.

- “” (empty string)
- HOME
- INFO
- WIN
- LOSE
- ERROR
- SETUP
- HOST_GAME
- JOIN_GAME
- YOUR_TURN
- THEIR_TURN

`ButtonCallbackWrapper`

A wrapper class for board button callbacks. This is able to store x, y, and a callable for any given square on the board.

`__init__(self, x: int, y: int, c: Callable[[int, int], None]) -> None`

Initialize with the given data.

`__call__(self) -> None`

Call the callback function with the x and y position of this board square.

`StrategoGUI`

`resize_image(img: tkinter.PhotoImage, w: int, h: int) -> tkinter.PhotoImage`

Returns the given image, re-scaled to the given dimensions.

`get_instance(cls) -> 'StrategoGUI'`

Returns any existing instance of the singleton `StrategoGUI` class. If none exists, this method creates an instance.

`clear_instance(cls) -> None`

Erases any existing instance of the `StrategoGUI` singleton class. If none exists, does nothing.

`press_key(self, key: str) -> None`

Simulates the given keypress. This is a wrapper function on top of `tkinter`, so the keypress must follow their formatting.

`__init__(self) -> None`

Initializes the GUI. If a GUI currently exists, this will raise an error. Otherwise, it will launch the home screen and enter the main app loop.

`property screen(self) -> ScreenType`

Gets the current screen.

`property screen(self, to: ScreenType) -> None`

Transitions the GUI to the given screen.

`property color(self) -> Literal['BLUE', 'RED']`

Gets the current color of the player.

`property color(self, to: Literal['BLUE', 'RED']) -> None`

Sets the GUI's player color to the given value.

`property board(self) -> Board`

Gets the underlying Stratego board from the GUI.

`quit(self) -> None`

Shuts down the app.

`clear(self) -> None`

Clears the GUI screen without moving to any other screen.

`__get_image(self, piece: b.Square) -> tk.PhotoImage`

Loads the image which represents the given piece.

`__refresh_board(self, callback: Callable[[int, int], None]) -> None`

Refreshes the board which is currently on screen, given that it exists. The given callback function will be called when a button on the board is pressed.

`__display_board(self, callback: Callable[[int, int], None]) -> None`

Creates and displays a board. The given callback function will be called when a button on the board is pressed.

`__bind(self, sequence: str, event: Callable[[], None]) -> None`

Binds the given keypress sequence to the given function.

`__quit(self) -> None`

Internal function for quitting the app.

`__clear(self) -> None`

Internal function for clearing the GUI screen.

`__home_screen(self) -> None`

Displays the home screen, which can move to the info, host, or join screen.

`__info_screen(self) -> None`

Displays information about the team and the project.

`__host_game_screen(self) -> None`

Allows the player to enter an IP and port to host a game of Stratego on. This will display the game password, IP and port upon hosting.

`__join_game_screen(self) -> None`

Allows the player to enter an IP, port and password to join a game of Stratego on.

`__setup_left_to_place(self) -> None`

Internal function which sets up the pieces which still need to be placed.

`__randomize_all(self) -> None`

Randomly places all remaining pieces.

`__first_sync(self) -> None`

Synchronizes the game state with the other player. This immediately moves on upon completion.

`__setup_screen(self) -> None`

Screen where the player is to set up their pieces. This will wait for the other player to complete before moving on.

`__your_turn_screen(self) -> None`

Waits for the user to make a move selection, then sends it to the other computer.

`__check_move(self) -> None`

Internal function for validating a user move.

`__their_turn_screen(self) -> None`

Waits for the other user to decide on a move.

`__win_screen(self) -> None`

Displayed if our player captures the other player's flag.

`__lose_screen(self) -> None`

Displayed if the other player captures our flag.

`__error_screen(self) -> None`

Displayed upon networking failure.

Networking

`StrategoNetworker`

This class handles transmission of the game state for the GUI.

`is_terminal_state(cur_state: str) -> bool`

Returns true if the given state warrents halting the game, false otherwise.

`clear_instance(cls) -> None`

Clears any existing networker.

`get_instance(cls) -> 'StrategoNetworker'`

Yields the existing networker instance if there is one, creates one otherwise.

`__init__(self) -> None`

Initializes the networker.

`host_game(self, ip: str, port: int) -> str`

Hosts a game on the given IP address and port number. Returns a randomly generated password.

`host_wait_for_join(self) -> None`

Assuming that this is the host networker, waits for a client networker to join the connection.

`join_game(self, ip: str, port: int, password: str) -> int`

Joins a game on the given IP and port using the given password. Returns 0 on success, nonzero on error.

`close_game(self) -> None`

Stops hosting a game.

`send_game(self, board: Board, state: str) -> None`

Sends the given board and game state to the other computer.

`recv_game(self) -> Tuple[Board, str]`

Receives the other computer's board and game state.

`__send_board(self, to_send: Board) -> None`

Sends the board over a socket.

`__recv_board(self) -> Board`

Receives a board over a socket.

`__send_game_state(self, state: str) -> None`

Sends the game state over a socket.

`__recv_game_state(self) -> str`

Receives the game state over a socket.

Board

`LakeSquare`

A unit class to represent a lake on the board.

`InvalidMoveError: Exception`

An exception class raised when an invalid move is provided.

`Square`

This is a union of several types. All of the types listed below are valid as an instance of `Square`

- `Piece`
- `None` (`NoneType`)
- `LakeSquare`

`Board`

Handles game logic and holds pieces.

`property height(self) -> int`

Returns the height of the board.

`property width(self) -> int`

Returns the width of the board.

`clear(self) -> None`

Resets the board to defaults.

`fill(self, start: Tuple[int, int], end: Tuple[int, int], to: Union[Square, Callable[[int, int], Square]]) -> None`

Fills the given region with the given pieces. If `to` is a callable instead of a piece, it calls it with the coordinates of each spot and places the return value in that spot.

`get(self, x: int, y: int) -> Square`

Gets the square at the given coordinates.

`set_piece(self, x: int, y: int, what: Square) -> None`

Sets the square at the given coordinates.

`move(self, color: Literal['BLUE', 'RED], from_pair: Tuple[int, int], to_pair: Tuple[int, int]) -> Literal['RED', 'BLUE', 'GOOD']`

Attempts to move the piece at `from_pair` to `to_pair`, given that the color `color` is trying to make the move. If the move is determined to be invalid, raises an `InvalidMoveError`. Otherwise, returns the new game state.

`__init__(self) -> None`

Initializes the board to defaults.

`__move_is_inside_board(cls, from_x: int, to_x: int, from_y: int, to_y: int) -> bool`

Internal function which verifies that the given move is inside the board.

`__move_is_logical(from_x: int, to_x: int, from_y: int, to_y: int) -> bool`

Internal function which verifies that the given move is logical.

`__types_are_legal(from_piece: Square, to_piece: Square) -> bool`

Internal function which verifies that the given move deals with legal types.

`__move_makes_sense_for_type(self, from_x: int, to_x: int, from_y: int, to_y: int) -> bool`

Internal function which verifies that the given move makes sense for the piece being moved.

`__is_valid_move(self, from_x: int, from_y: int, to_x: int, to_y: int) -> bool`

Internal function which verifies that the given move is valid. This calls all the other validity checking internal methods, and returns true if and only if they all return true.

Piece

`Piece: abc.ABC`

Represents an abstract base class from which all the usefull Stratego pieces inherit.

`__init__(self, color: Literal['BLUE', 'RED]) -> None`

Initializes the piece with the given color.

`__eq__(self, rhs: object) -> bool`

Returns true if and only if this object is equal to the other.

`__hash__(self) -> int`

Returns the hash of this object.

`property color(self) -> str`

Returns the color of this object.

`__repr__(self) -> str`

Returns the string representation of this object.

`confront(self, other: 'Piece') -> Optional['Piece']`

Returns the piece which should remain (if any) when this piece confronts another.

`property rank(self) -> int`

Returns the rank of this piece.

`Bomb: Piece`

Represents a bomb on the board.

`__repr__(self) -> str`

Returns the string representation of this piece.

`property rank(self) -> int`

Returns the rank of this piece.

`confront(self, _: Piece) -> Optional[Piece]`

Returns the piece which should remain (if any) when this piece confronts another. This is usually `self` for a bomb, except if the other piece is a miner.

`Flag: Piece`

A flag piece in Stratego. This is the goal of the game.

`__repr__(self) -> str`

Returns the string representation of this piece.

`property rank(self) -> int`

Returns the rank of this piece.

`confront(self, _: Piece) -> Optional[Piece]`

Returns the piece which should remain (if any) when this piece confronts another. This is always `self` for a flag piece.

`Troop: Piece`

A standard troop piece, from which the special troops inherit.

`__init__(self, color: Literal['BLUE', 'RED], rank: int) -> None`

Initializes the troop with the given color and rank.

`__repr__(self) -> str`

Returns the string representation of this object.

`property rank(self) -> int`

Returns the rank of this object.

`confront(self, other: Optional[Piece]) -> Optional[Piece]`

Returns the piece which should remain (if any) when this piece confronts another.

`Spy: Troop`

A special spy piece. This can defeat marshals if it attacks them.

`__init__(self, color: Literal['BLUE', 'RED']) -> None`

Initialize this piece with the given color.

`confront(self, other: Optional[Piece]) -> Optional[Piece]`

Returns the piece which should remain (if any) when this piece confronts another. This is either `other` or `None` for all pieces except the marshal, when it is `self`.

`Miner: Troop`

A miner piece. These pieces can disarm bombs.

`__init__(self, color: Literal['BLUE', 'RED']) -> None`

Initialize this object with the given color.

`confront(self, other: Optional[Piece]) -> Optional[Piece]`

Returns the piece which should remain (if any) when this piece confronts another. If `other` is an opponent's bomb, returns `self`.

`Scout: Troop`

A special piece which can move like a castle in chess.

`__init__(self, color: Literal['BLUE', 'RED']) -> None`

Initializes this object with the given color.

`Marshal: Troop`

A 10 piece which can be killed by spies.

`__init__(self, color: Literal['BLUE', 'RED']) -> None`

Initializes this object with the given color.