Name: Zane Miller

Date: 11/20/2021

Description: Halfway Progress Report (CS162)

Currently, I am using one class for the Hasami Shogi game. I believe I could further modularize this program into 3 classes (HasamiShogiGame, Board, Pieces). Below is the code for the current class HasamiShogiGame and its methods.

1. class HasamiShogiGame:  
    *"""*

*This is the main class for the board game Hasami Shogi Variant 1. The board*

*is represented as a 9x9 array with a numbered x-axis [1, 9] and a lettered y-*

*axis [a, i].*

*"""*

* 1. def \_\_init\_\_(self, width=9, length=9):  
      *"""  
      Initializes width and length of board, dictionaries for red and black   
      pieces, a board array, game\_state to "UNFINSIHED", active player to   
      "BLACK", and the number of captured pieces for red and black to 0.* ***:param*** *width: initialized to 9* ***:param*** *length: initialized to 9  
      """*
  2. def initialize\_board\_array(self):  
      *"""  
      Initializes the board array for the start of the current game. Adds the  
      x- and y-coordinates, initializes the header row [1, 9] for printing  
      the board, and initializes each space on the board as "NONE" (empty)  
      and assigns each space the symbol "." to represent an empty space.  
      returns: the self.\_board since this method is called in \_\_init\_\_ to   
      initialize self.\_board at the creation of the object.   
      """*
  3. def get\_board\_array(self):  
      *"""  
      Gets the current board array.* ***:return****: self.\_board  
      """*
  4. def convert\_coordinates\_xy(self, coordinate):  
      *"""  
      Converts single string coordinate to x- and y- individual coordinate   
      values.* ***:param*** *coordinate: takes a single string coordinate ("a1").* ***:return****: returns the string value for coordinate y and the integer   
      value for coordinate x.   
      """*
  5. def get\_square\_occupant(self, coordinate):  
      *"""  
      Gets the current occupant, if any, of the given space. Extracts the* ***:param*** *coordinate: takes a single string coordinate* ***:return****: returns the occupant (RED, BLACK, NONE) at that coordinate.   
      """*
  6. def set\_square\_occupant(self, x\_coord, y\_coord, symbol):  
      *"""  
      Updates the symbol at a given x- and y-coordinate based on the symbol it   
      receives.* ***:param*** *x\_coord: integer value of the x-coordinate.* ***:param*** *y\_coord: string value of the y-coordinate* ***:param*** *symbol: symbol to be represented in the coordinate.* ***:return****: none  
      """*
  7. def get\_game\_state(self):  
      *"""  
      Gets the current state of the game (UNFINISHED, RED\_WON, BLACK\_WON)* ***:return****: self.\_game\_state  
      """*
  8. def set\_game\_state(self):  
      *"""  
      Sets the current state of the game (UNFINISHED, RED\_WON, BLACK\_WON)* ***:return****: none  
      """*
  9. def get\_active\_player(self):  
      *"""  
      Gets the current active player (RED, BLACK)* ***:return****: self.\_active\_player  
      """*
  10. def set\_active\_player(self, player):  
       *"""  
       Sets the active player* ***:param*** *player: player to assign as the new current player* ***:return****: none  
       """*
  11. def get\_piece\_object(self, color, coordinate):  
       *"""  
       Gets the object id of a piece at a given location* ***:param*** *color: the color of the piece we are looking for* ***:param*** *coordinate: the location of the piece we are looking for.* ***:return****: object\_id which is the object id of the specific color piece   
       at the given coordinate.   
       """*
  12. def get\_num\_captured\_pieces(self, color):  
       *"""  
       Gets the number of captured pieces by color* ***:param*** *color: the color of the piece we are looking for* ***:return****: the number of captured pieces for black or red depending on   
       the received color.   
       """*
  13. def set\_num\_captured\_pieces(self, color, num):  
       *"""  
       Increments the number of captured pieces for a specific color* ***:param*** *color: color of piece we are incrementing* ***:param*** *num: number of pieces removed* ***:return****: none  
       """*
  14. def print\_board(self):  
       *"""  
       Prints the board in its current state. Initializes the index of the   
       self.\_alphabet string to 0. Initializes a temporary row string for   
       printing. Prints the header row from self.\_board[0]. Prints the rest of   
       the board, row by row.* ***:return****: none.  
       """*
  15. def create\_game\_pieces(self):  
       *"""  
       Initializes the dictionary for the red and black pieces with  
       piece\_object as the key, and piece color, location, and status as keys   
       of a dictionary.   
       Initializes the starting coordinates for each piece   
       (red: a1 - a9 and black i1 - i9).   
       Initializes the arrays for each piece north, south, east, west with  
       self.define\_arrays\_nsew.   
       Updates the square occupants in self.\_board with   
       self.set\_square\_occupant.* ***:return****: none  
       """*
  16. def check\_valid\_move(self, move\_from, move\_to):  
       *"""  
       Verifies that a move is valid. First checks if the move is invalid if   
       a) the x or y coordinates are not the same (move is not strictly   
       horizontal or vertical); b) if the player has a piece in the current   
       spot; c) the player is trying to move to an occupied spot. If the   
       user's move is not invalidated, the method then gathers an array of   
       all spaces between where the user is moving from to where the user is   
       moving to. The method then sets the range to reflect the appropriate   
       move direction. Then validates that none of the spaces in-between   
       move\_from and move\_to are already occupied.* ***:param*** *move\_from: yx coordinate of where the user is trying to move   
       from.* ***:param*** *move\_to: yx coordinate of where the user is trying to move to.* ***:return****: False if the user move is invalid. Returns True otherwise  
       """*
  17. def make\_move(self, move\_from, move\_to):  
       *"""  
       Makes a move for the active player if the move is verified as valid   
       (True). If the move is valid, the method will update the location of   
       the piece depending on who is moving. It will then update the NSEW   
       arrays for the piece that was moved. Checking for trapped pieces will   
       be done after the current players turn before the next player is set,   
       so there is no need to update other arrays. Next, the method will check   
       for trapped pieces before finally setting the next active player.* ***:param*** *move\_from: coordinates of where the user is moving from* ***:param*** *move\_to: coordinates of where the user is moving to.* ***:return****: False if user move is invalid.   
       """*
  18. def define\_arrays\_nsew(self, move\_to, color):  
       *"""  
       Defines/initializes arrays for each piece that store the occupants in   
       spaces to the North, South, East, and West of every piece. Updates   
       those arrays after a piece has been moved. Reverses the order of the   
       arrays for North and West since default direction is read East and   
       South.* ***:param*** *move\_to: yx coordinate of the location piece moved to or is   
       starting at* ***:param*** *color: the color of the piece to be updated so correct   
       dictionary is updated.* ***:return****: none  
       """*
  19. def check\_trapped\_pieces(self):  
       *"""  
       Checks if the player sandwiched any opposing pieces, to the North,  
       South, East, West, or corner sandwiches, after a move. If the player  
       sandwiched opposing players, they are added to a   
       remove\_occupants\_list\_direction, and then removed by calling   
       self.remove\_occupants().* ***:return****: none  
       """*
  20. def remove\_occupants(self, remove\_occupants\_list):  
       *"""  
       Removes occupants from list of occupants to remove, then updates the  
       num\_captured\_pieces for the color just captured, the status of the  
       captured pieces in its dictionary.* ***:param*** *remove\_occupants\_list: list of occupants to remove.* ***:return****: none  
       """*

1. class Pieces:  
    def \_\_init\_\_(self):  
    *""" Currently only used to initialize object\_ids for each piece. """*

**DETAILED TEXT DECRIPTIONS OF HOW TO HANDLE THE FOLLOWING SCENARIOS**

1. **Determining how to store the board**

The board will be stored as an array (self.\_board). The array will be a list of lists with self.\_board[0] being the header row list (['1', '2', '3', '4', '5', '6', '7', '8', '9']). Each subsequent element will be a list [y-coordinate, x-coordinate, occupant, symbol]. For example (['a', 1, 'NONE', '.']). There are 82 elements in self.\_board - the first is the header row and 81 elements for each of the 81 spaces on the board. The list is updated after each move, to reflect any changes to occupant status and symbol.

1. **Initializing the board**

The board is initialized using the initialize\_board\_array() method. This method is called in \_\_init\_\_ as:

self.\_board = self.initialize\_board\_array()

1. **Determining how to track which player’s turn it is to play right now**

The players turn is tracked by first initializing self.\_active\_player = “BLACK”. The method set\_active\_player() takes “player” as a parameter and changes self.\_active\_player to the opposite player. The method is called at the end of make\_move() after any pieces have been removed and updated. A method get\_active\_player() is used to get the current active player in check\_valid\_move(). In check\_valid\_move, if the current player color is not equal to the color of piece in the location the player is trying to move from, it returns false.

1. **Determining how to validate a piece movement**

Movements are validated in check\_valid\_move() which is called from the make\_move() function. If the x- or y- coordinates the player is moving from do not match the x- or y- coordinates the player is moving to (i.e. the player is not moving exactly vertical or horizontally), the function returns false and make\_move() then returns false. Also, if the player is trying to move to a space that is currently occupied, or does not have a piece in the space they are trying to move from, check\_valid\_move() also returns false.

If the piece passes these initial tests, an array of occupants is compiled for all the spaces between where its moving from to where its moving to. This list of occupants is then checked to verify that all the occupants are “None” ensuring the player is not trying to “jump” any pieces.

1. **Determining when pieces have been captured**

Determining when pieces have been captured is done in the check\_trapped\_pieces() method which is called by the make\_move() method. Checking for trapped pieces is done from the most recently moved piece.

First, each piece has a North, South, East, and West array. These arrays are added to each piece’s dictionary as a separate key value. Each array consists of a list of all the spaces between it and the edge of the board in its respective direction. This array is updated after each move in the define\_arrays\_nsew() method, called by the make\_move() method.

The method check\_trapped\_pieces() checks each of the recently moved piece’s North, South, East, and West arrays. If the first value of any array is a space that contains another piece of its own, or an empty space, the method moves to the next direction. If the first value of any direction array consist of a space occupied by the opposing player, it continues checking until it runs into an empty space or a space containing its own piece. If empty, it breaks to the next direction. If its contains its own color, it adds the list to a remove\_occupants\_list for that direction.

1. **Determining when the game has ended**

The method set\_game\_state() is used to update the game state after each move. It checks the number of pieces captured for each player. If any player has 8 or more pieces captured, the game state is updated to reflect the winner (other player).