Colton Woodruff
CS 162
08/09/2023

Halfway Progress Report

I am using six classes - Rook, Bishop, Knight, and King, which inherit from Piece, and are used in ChessVar. Below is the structure of classes and methods including docstrings explaining how they’re used.

```
class Piece:
    “””Represents a generic chess piece object to be inherited by piece specific classes”””

    def __init__(self, color, pos):
        “””The constructor for the Piece class. Takes color and position on board as
        parameters and assigns them to class data members, as well as initializing
        others.”””

        pass

    def get_color(self):
        “””Takes no parameters and returns color of game piece.”””

        pass

    def get_symbol(self):
        “””Takes no parameters and returns symbol of game piece.”””
        # symbol is a two-letter representation of the piece used when printing the game
        board to console.

        pass

    def get_available moves(self):
        “””Takes no parameters and returns list of spaces the piece can move to.”””

        pass

    def get_type(self):
        “””Takes no parameters and returns type of game piece.”””
        # types are ‘rook’, ‘knight’, etc.

        pass
```

Colton Woodruff
CS 162
08/09/2023

```
    def get_pos(self):
        “””Takes no parameters and returns position of game piece.”””
        # position not stored in chess notation. Rather, the game board is treated as a
        single list and the position is the index (0 - 63).

        pass

    def set_pos(self, pos):
        “””Updates the position attribute when a piece is moved. Returns nothing.”””

        pass


class Rook(Piece):
    “””Represents a rook chess piece object that inherits Piece class. Piece is used by the
    ChessVar class”””

    def __init__(self, color, pos):
        “””The constructor for the Rook class. Takes color and position on board as
        parameters and assigns them to class data members, as well as updating others
        that are initialized by Piece class.”””

        pass

    def update_moves(self, piece_locations, test_location = None):
        """Returns available spaces rook can move (before testing for legal takes or
        checking king. Takes a list of piece locations (as list indexes) and a hypothetical
        location of the piece that’s different from the actual position.

        Method has four loops that check squares to the left, right, up, and down,
        appending those locations to a list of possible_moves, until either the board edge
        or another piece is hit in each respective direction.

        If no test_location was passed, the piece updates it’s _available_moves attribute.
        If a test_location was passed, it returns the possible moves without updating.
        """

        pass

class Bishop(Piece):
    “””Represents a bishop chess piece object that inherits Piece class. Piece is used by the
    ChessVar class”””
```

```
        def __init__(self, color, pos):
                “””The constructor for the Bishop class. Takes color and position on board as
                parameters and assigns them to class data members, as well as updating others
                that are initialized by Piece class.”””

                pass

        def update_moves(self, piece_locations, test_location = None):
                """Returns available spaces rook can move (before testing for legal takes or
                checking king. Takes a list of piece locations (as list indexes) and a hypothetical
                location of the piece that’s different from the actual position.

                Method has four loops that check squares to the up-left, up-right, down-left, and
                down-right diagonals, appending those locations to a list of possible_moves, until
                either the board edge or another piece is hit in each respective direction.

                If no test_location was passed, the piece updates it’s _available_moves attribute.
                If a test_location was passed, it returns the possible moves without updating.
                """

                pass

class Knight(Piece):
        “””Represents a knight chess piece object that inherits Piece class. Piece is used by the
        ChessVar class”””

        def __init__(self, color, pos):
                “””The constructor for the Knight class. Takes color and position on board as
                parameters and assigns them to class data members, as well as updating others
                that are initialized by Piece class.”””

                pass

        def update_moves(self, piece_locations, test_location = None):
                """Returns available spaces knight can move (before testing for legal takes or
                checking king. Takes a list of piece locations (as list indexes) and a hypothetical
                location of the piece that’s different from the actual position.

                Method checks each of the eight possible moves a rook can make and appends
                them to the _available_moves attribute if they are not beyond any board edge .

                If no test_location was passed, the piece updates it’s _available_moves attribute.
                If a test_location was passed, it returns the possible moves without updating.
                """

                pass

class King(Piece):
        “””Represents a king chess piece object that inherits Piece class. Piece is used by the
        ChessVar class”””

        def __init__(self, color, pos):
                “””The constructor for the King class. Takes color and position on board as
                parameters and assigns them to class data members, as well as updating others
                that are initialized by Piece class.”””

                pass

        def update_moves(self, piece_locations, test_location = None):
                """Returns available spaces king can move (before testing for legal takes or
                checking king. Takes a list of piece locations (as list indexes) and a hypothetical
                location of the piece that’s different from the actual position.

                Method checks each of the eight possible moves a rook can make and appends
                them to the _available_moves attribute if they are not beyond any board edge .

                If no test_location was passed, the piece updates it’s _available_moves attribute.
                If a test_location was passed, it returns the possible moves without updating.
                """

                pass

class ChessVar:
        “””An object representing a chess board, on which a variant of chess is played. Contains
        methods that manipulate chess piece objects, tracks game state, which color turn it is.
        Uses objects of each piece class (all but Piece class) to play game to track location on
        board and interact with other chess piece objects.”””

        def __init__(self, color, pos):
                “””The constructor for the ChessVar class. Takes no parameters and initializes
                the game state, turn, a dictionary of pieces in play, and sets for possible moves
                either color can make.

                The game pieces and their starting locations are initialized and added to the
                dictionary of pieces on the board.
```

Colton Woodruff
CS 162
08/09/2023

I chose to represent the chess board as a single, linear list, rather than a list of lists, or dictionary, or other data type. The ChessVar class has a list of spaces in chess notation saved as a variable that is used to convert from chess notation to linear index. All game pieces do not move in two directions, but rather left and right along a line as if each row was lined end to end.”””

pass

def get_game_state(self):

"""Takes no parameters and returns the current game state (‘UNFINISHED’, ‘WHITE WINS’, ‘BLACK WINS’, or ‘TIE’ """

pass

def get_turn(self):

"""Takes no parameters and returns the current color turn"""

pass

def game_update(self):

"""Takes no parameters and updates game state and color turn. Returns nothing.

Method checks which row either team’s king is in and then checks if white reached the end without black reaching it, if both reached the end on consecutive turns, or if black reached the end first, and then updates the game_state to the corresponding end-game goal reached.

If the game is not yet over, this method advances to next teams turn."""

pass

def update_moves(self):

"""Takes no parameters and updates all pieces available moves. Returns nothing.

Method iterates through all pieces in play and calls their respective update_moves() methods, and saving adding them to the corresponding team set. Each piece has a list of its own available moves saved, and the ChessVar class has two sets, one for either team, that contain all possible moves at the moment, which is used for checking if a king is entering check."""

pass

Colton Woodruff
CS 162
08/09/2023

def check_checker(self, piece, to_index):

"""Takes two parameters, a piece being moved and the list index of the square it's moving to, then checks if that move will cause a king to enter check. Returns True if a king is in check.

Method performs two operations. If the piece being moved is not a king, it hypothetically moves it, then calls update_moves() on all pieces and checks if any piece on the board has put an opposing king in check.

If the piece being moved is a king, it checks if the square being moved to exists in the set of squares reachable by the opposing color."""

pass

def make_move(self, current_square, to_square):

"""Takes two parameters, a square to move a piece from a square to move the piece to, then checks if that move is legal. If it is legal, it makes the move and returns nothing.

Method first checks if the selected square has a piece at it, then checks if the piece matches the color for the current turn. Then it checks if the to-square is in the piece's available_moves. The next check is if check_checker() returns if the move will put a king in check. Finally, it checks if the to_square has a piece already assigned to it and if that piece is the same color as the piece moving. If all these checks are passed, the piece is moved and its ._piece attribute is updated.

After a move is completed, this method then calls game_update() and update_move()."""

pass

def print_board(self):

"""Takes no parameters and prints out the game board to console. For testing and debugging

Method iterates through list of board squares and prints either empty space or piece symbols eight spaces at a time to represent the board rows."""

pass

Colton Woodruff
CS 162
08/09/2023

Handling Listed Scenarios

Initializing the ChessVar class

An object of the ChessVar class does not need any parameters to initialize. When initialized, the starting state of the game is set, such as the turn being set to white and game state set to 'Unfinished'. The board representation (a single list), a dictionary that tracks the piece objects in play, and two sets that represent all available moves of either colored pieces are created as empty.

The initialization then creates the game pieces and adds them to the dictionary with their position, in chess notation, as the key. After the pieces are created, the initialization then iterates through them while calling their update_moves() method and adding those moves to the color move sets.

Keeping track of turn order

After a move is performed by the make_move() method, the same method then calls the game_update() method. This method looks at which color just moved and the location of both color kings. If no end-game status was reached, it sets ChessVar._turn to the opposite color.

```
If turn is white:
    Turn is black
Elif white king in row 8 and black king in row <8
    Game state is 'WHITE_WON'
Elif black king in row 8 and white king in row 8
    Game state is 'TIE'
Elif black king in row 8
    Game state is 'BLACK_WON'
Else
    Turn is white
```

Keeping track of the current board position

The board is represented as a single, linear list, as if the rows were laid end to end. The locations of pieces on the board are stored in two places - a dictionary in the ChessVar class that has piece objects tied to keys in chess notation. The pieces themselves store their position as a list index, rather than chess notation. When a move is made, both the dictionary and the piece's position attribute are updated in their respective format.

```
If conditionals are all met:
    Board dictionary[to_space] equals Board dictionary.pop(current_space)
    Piece._pos equals space list index
```

Colton Woodruff
CS 162
08/09/2023

Determining if a regular move is valid

Each move is checked through a series of if - elif checks.

```
If current_square is not in piece dictionary.keys()
    Return False
Elif piece color does not equal current turn
    Return False
Elif to_square is not in piece available moves
    Return False
Elif check_checker(piece, to_square) # checks if move puts either king in check
    Return False
# check if space being moved to is occupied by piece of same color
Elif to_square in piece dictionary.keys() and other piece color equals move piece
    Return False
Else
    Make move
```

Determining if a capture is valid

This check occurs in the if - elif checks that happen before any move is made. Each piece has a ._color attribute that is checked. If they do not match, then the capture is allowed. Whether or not the piece can reach the space where the capture is happening is tested prior to this.

```
Previous conditional checks
Elif to_square in piece dictionary.keys() and other piece color equals move piece
    Return False
```

Determining whether a move places either king into check

Each piece class has a method to find all possible spaces it can possibly reach based on its specific movement. Whenever a new move is occurring, the move is passed to the check_checker() method which simulates the move and then iterates through every piece, calling the piece.update_moves() method. Then it checks whether any of the spaces returned match the location of a king. If a piece can reach the location of a king of the opposite color, check_checker() returns True, which causes the move conditional to return False. If the piece being moved is a king, it checks the the space its moving to cannot be reached by any piece of the other color.

```
For pieces in game piece dictionary
    If piece is king and piece is opposite color and piece location is somewhere the current piece can reach
    Return True

If piece is king and to_space is in opposite team moves set
    Return True
```

Colton Woodruff
CS 162
08/09/2023

Determining the current state of the game

After each move, the game_update() method is called. Part of this method's operation is checking which color just completed its turn and the row location of both kings. If white just finished its turn, there is no change to the game state, because black always has the opportunity to tie on the next turn. If black just finished its turn, there are four options:

```
If turn is white:
    Turn is black
Elif white king in row 8 and black king in row <8
    Game state is 'WHITE_WON'
Elif black king in row 8 and white king in row 8
    Game state is 'TIE'
Elif black king in row 8
    Game state is 'BLACK_WON'
Else
    Turn is white
```