Daniel Reid Nelsen
Intro to Computer Science II
CS 162

Halfway Progress Report

class OutOfTurn(Exception):
"""Raise exception if a player moves out of turn"""
pass

Used if the next player is going out of turn

class InvalidSquare(Exception):
"""Raise exception if a player doesn't have a piece in the square"""
Pass

Used if the player is moving the wrong piece of if there is nothing in the square to move. There are other scenarios explained below.

class InvalidPlayer(Exception):
"""Raise exception if the player doesn't exist"""
Pass

Raised if player doesn't exist in the created objects.

class Player():
"""The player class takes a player name and piece color."""

def __init__(self, player_name, piece_color):

This will initialize the name of the player, the piece color they are, the number of kings, triple kings, and captured pieces. Last three members will star with 0

def get_king_count(self):
"""Returns king count of player"""

def get_tripe_king_count(self):
"""Returns the triple king count of player"""

def get_captured_pieces(self):
"""Returns the number of captured pieces"""

def get_player_name(self):
"""Method to get the players name for various methods and determine who won. Returns the players name"""

class Checkers:
"""The checker class"""

def __init__(self):
"""Init method for the checkers class"""

Set up board using dictionary tuple of grid as a key and values are the current piece on the grid.

Need to set whose turn is first, the players in a dictionary with key as player name and then the Player object as the value. Initialize the number of captures pieces, which piece is moving, last piece moved and if a piece was captured.

def create_player(self, player_name, piece_color):
"""Create players for the game. Creates an object from the Player class. Puts these objects in a dictionary."""

Player is added by calling the Player class and then put into a dictionary of self._players. Name is the key and object is the value

def get_turn(self, checker_type = None):
"""Method to return who's turn it is"""

Get whose turn it is for checking that the correct player is playing.

def set_turn(self):
"""Method to set who's turn it is. Called after a successful move is made and no pieces are captured. This method is not called after a captures, because a capture means the player has another turn"""

Get whose turn it is for checking that the correct player is playing. Also need to reset the if an opponent's piece was captured

def set_king(self, location, player_name):
"""Sets the piece to king after it reaches the opposite side of the board. Nothing is returned from this method"""

If a piece reaches either side of the board set it to King. Also set how many kings are available for each player.

def set_triple_king(self, location, player_name):
"""Set king to triple king after a king reaches its home row. Nothing is returned from this method"""

If a king reaches it's home row then make it a Triple King. Also set how many kinds are available for each player.

def set_moving_piece(self, start):
“””Helper method for the class. This sets what piece is moving, regular, king, or triple king. Nothing is returned from this method”””

Set which piece is moving, regular, king, or triple king

def get_moving(self):
“””Helper method for the class to see which piece we are moving. This is needed by play_game to determine if the piece is able to move backwards or forwards based on color. This method returns the game piece type, regular, king, or triple king”””

return which piece is moving

def play_game(self, player_name, start, finish):

"""Method for moving pieces on the checkers board. See description below for explanation. Final return is how many pieces were captured during the move"""

- Play_game takes player name and start and finish. Start and finish are tuples. Can unpack the tuples to determine wich way the player is moving.
- Each time the play_game is called need to initialize the following.
- # of captured
- Moving forward or backwards
- Moving left or right.
- Figure out what type of checker is moving, regular, king or triple king
- Also need to set if the piece is jumping to capture
- Set the coordinates to compare against when making kings or triple kings. These would be the end rows.

After initialization do the following (not necessarily in this order after further testing is done):

- unpack coordinates tuple

- determining which way the piece moves by subtracting the start and finish row and start and finish column if they are greater or less than 0, then we can determine which way the piece is moving. From there assigned variables with the movements.

- Detect if player is jumping if the subtraction of start and finish unpacked tuples is greater than 2.

- Check to see if the player exists. If a player isn't in the list raise InvalidPlayer

- Check is the checker type is valid if not raise InvalidSquare

- Checks to see if the square is in the list for the player
    - check that the player is moving the right piece otherwise raise InvalidSquare

- Check to see if it is the players turn otherwise raise OutOfTurn

- Check to see if the starting square is on the board otherwise raise InvalidSquare

- Check that the piece is moving in the correct direction based on which player is playing and the type of piece that is moving

- checks that the destination is empty and move the piece to that spot otherwise raise InvalidSquare

- If the player is jumping and capturing pieces then fix the board dictionary based on a successful move. Need to determine the direction moving and make adjustments based on that. Going to have forward – left, forward – right, backwards – left, backwards – right. These are directions are based on the black facing the white pieces. Forward means that the black is moving from for 7 to row 0 and backwards is row 0 to row 7. Raise InvalidSquare is something is wrong. This may need to change based on the README

- Movement for king and triple king with jumping and capturing. Use the same as above need to scan further because of the rules that are being used. May need to write a helper function in the class to determine number of captures pieces. If something is wrong then raise InvalidSquare.

- After moves are done then check if the piece is against the edge and make it a king or triple king if required using the initialized values

- Finally return the number of captured pieces if any. If there are no pieces captured then return 0

def set_last_piece_moved(self, finish_location):

"""Method to record the finishing location of the last piece that captured an opponent piece. Used in determining whether or not the correct player is playing next. Nothing is returned from this method"""

This is used to determine if the next player is moving the correct piece. If a player captures a piece and they have another turn with an opportunity to capture a piece then they need to make that jump and capture the piece. This will ensure that the rules are followed each time the play_game is called.

def count_captured(self, start, finish):

"""Method to see how many pieces will be captured. This is called from the play_game method and will return how many pieces will be captured after the move is finished. This returns the number of captured pieces to play_game"""

This takes the tuples of the start and finishing positions. Need to unpack the tuples for later comparisons. Figure out which way the pieces are moving. Then start counting down the diagonal how many pieces will be captured. This is especially useful for triple kings, but can be used for regular pieces and kings. Correct direction of movement is the same as was determined in the play_game method. Set the number of captured pieces here and return that value to play_game method. There will be a while loop that utilizes the start position and continues the loop until the tuple is the same as the finish position. From there we can count on each iteration how many opposite color pieces will be captured after the jump.

def get_checker_details(self, location):

"""Gets the details about a square on the checker board. This will return what piece is occupying a valid square. Returns None if no piece is on the square."""

Return which piece is on the board otherwise return None. Lookup the location from board dictionary and return the value. Also need to check that the coordinates exist on the board. If they don't raise InvalidSquare

def print_board(self):

"""Prints the current board layout. Returns the current board layout in the form of a list for each row and then those lists are in a list for the entire board """

This will turn the board dictionary into a list of lists. The lists will be the rows with either None or the piece type in the square. This will iterate over the dictionary with a counter to put each row of 7 squares in a row. Then it will append that row to the final board. Finally, the method will return the current board setup

def game_winner(self):

"""Returns either the name of the player who won the game or "Game has not ended"""

If the number of pieces captured is 12 then declare the winner. Going to get this from a private member variable. Otherwise return “Game has not ended”. This logic may need to change because there are cases where not all the pieces will be captured. However, the ReadMe states that we don’t need to check for that.

Final thoughts

Basic premise for Checkers class is that each coordinate on the board is a tuple, starting with (0,0) on the white side moving left and down to (7,7). From that tuple the class creates a dictionary with the tuple being the key and the value is the piece currently occupying the square. Empty black square will have None as the value and white unused squares will have an empty tuple.

The play_game class in the Checkers class will determine the following:

- how far the piece is moving
- the starting position
- the finishing position
- if the piece is the correct piece to move based on color
- if the piece is moving correctly based on the rules

The methods in Checkers need to be able to check the get the direction based on the supplied tuple of start and finish. The arbitrarily chosen nomenclature is from the perspective of the black pieces, because they start first. Forward means row 7 to 0, backwards means 0 to 7. Moving left means going from row 0 to 7 and moving right means moving from 7 to 0. So black will generally be forward and which will be moving backwards.

The class uses the following helper methods to help with the gameplay. See above for descriptions and why they are utilized.

- **count_captured**
- **set_last_piece_moved**
- **get_moving**
- **set_moving_piece**
- **set_triple_king**
- **get_turn**
- **set_turn**
- **set_king**