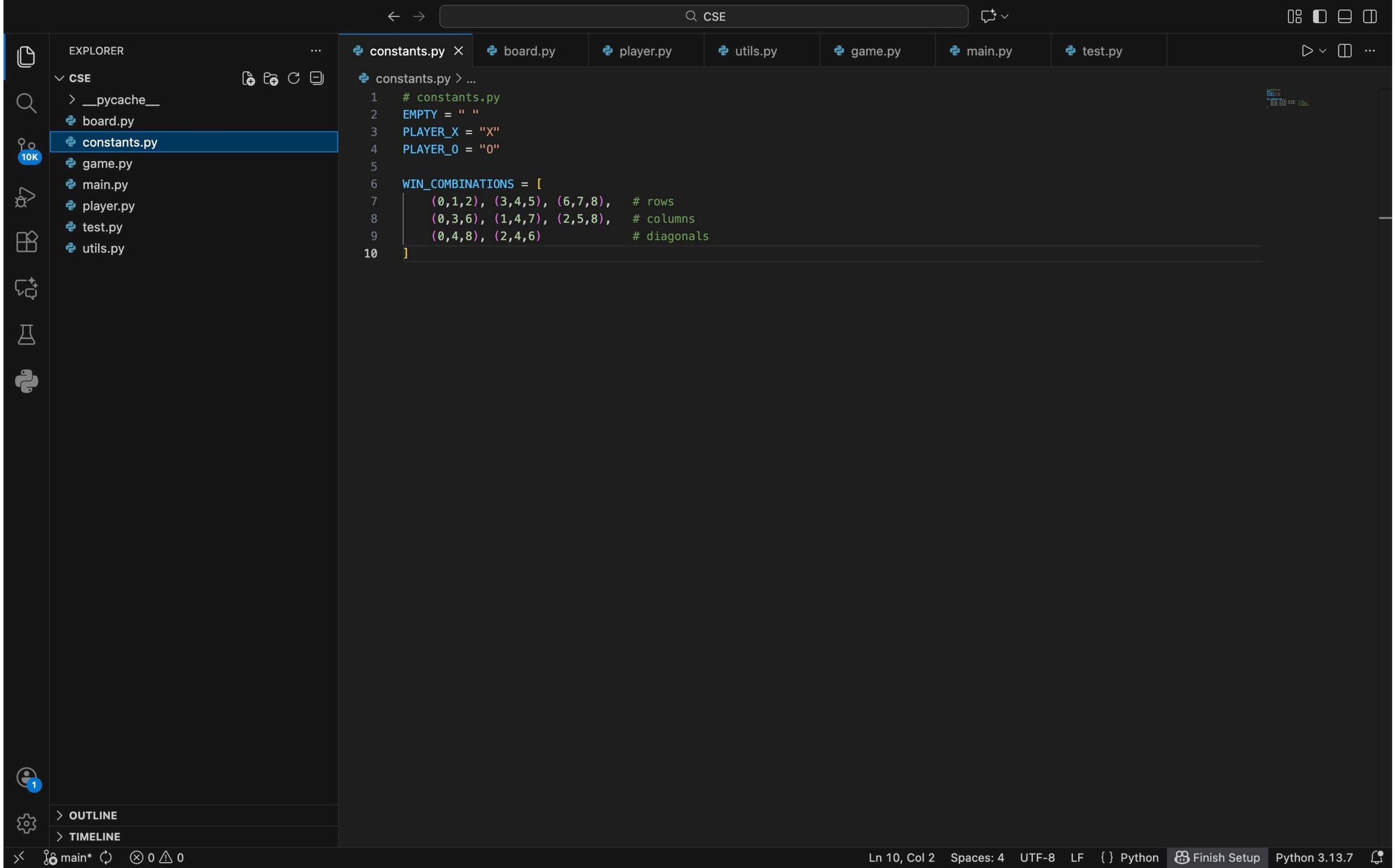
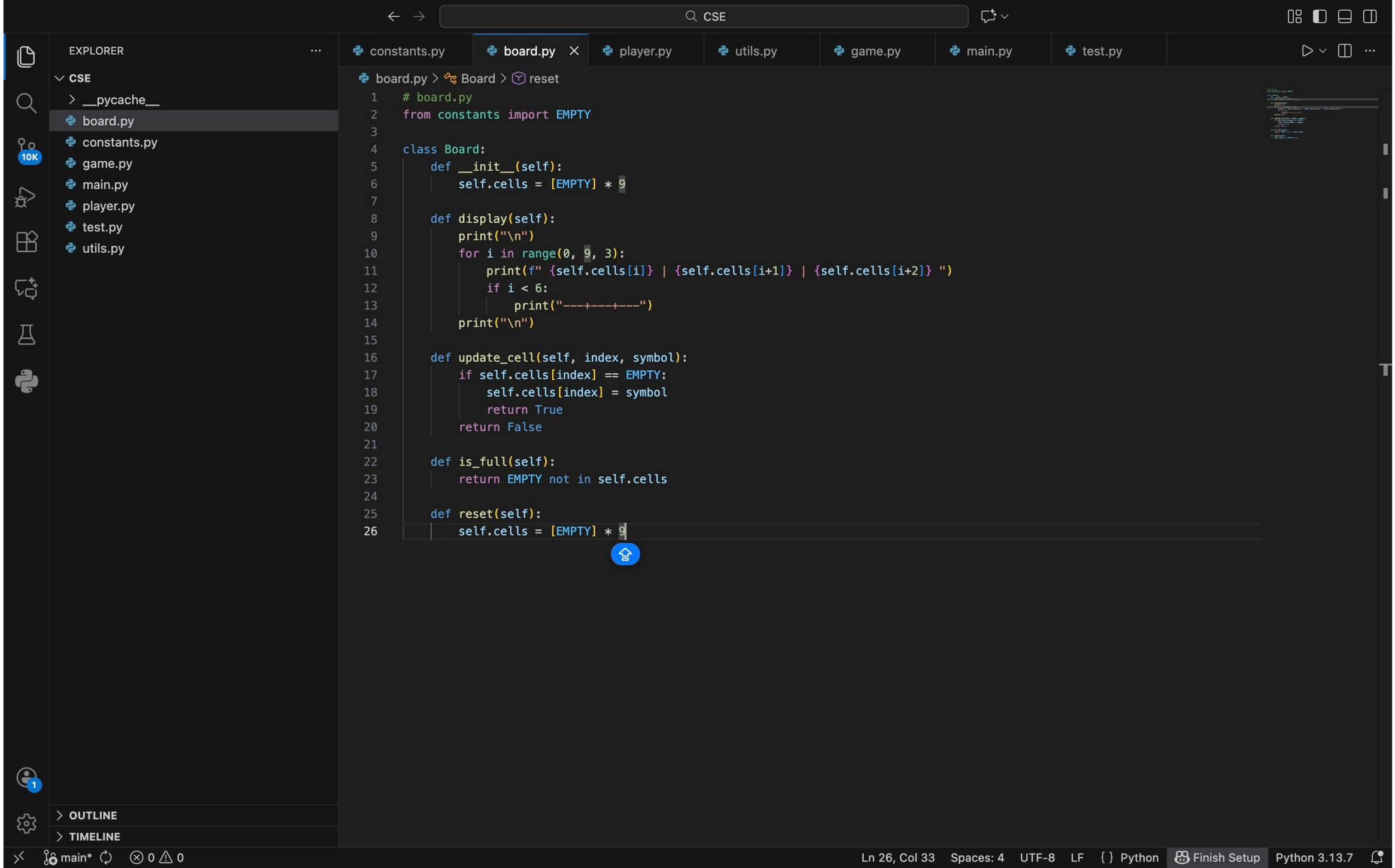
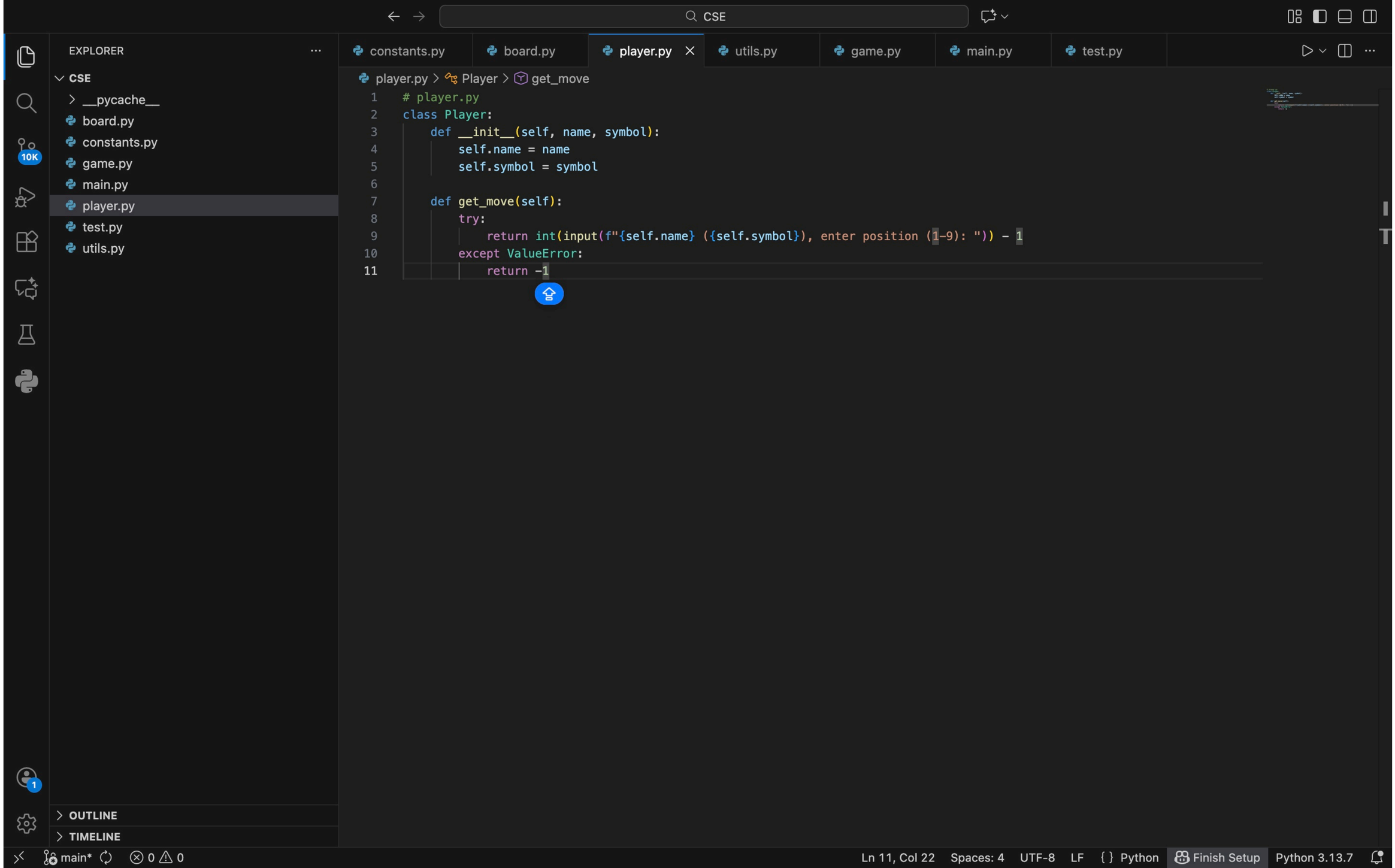
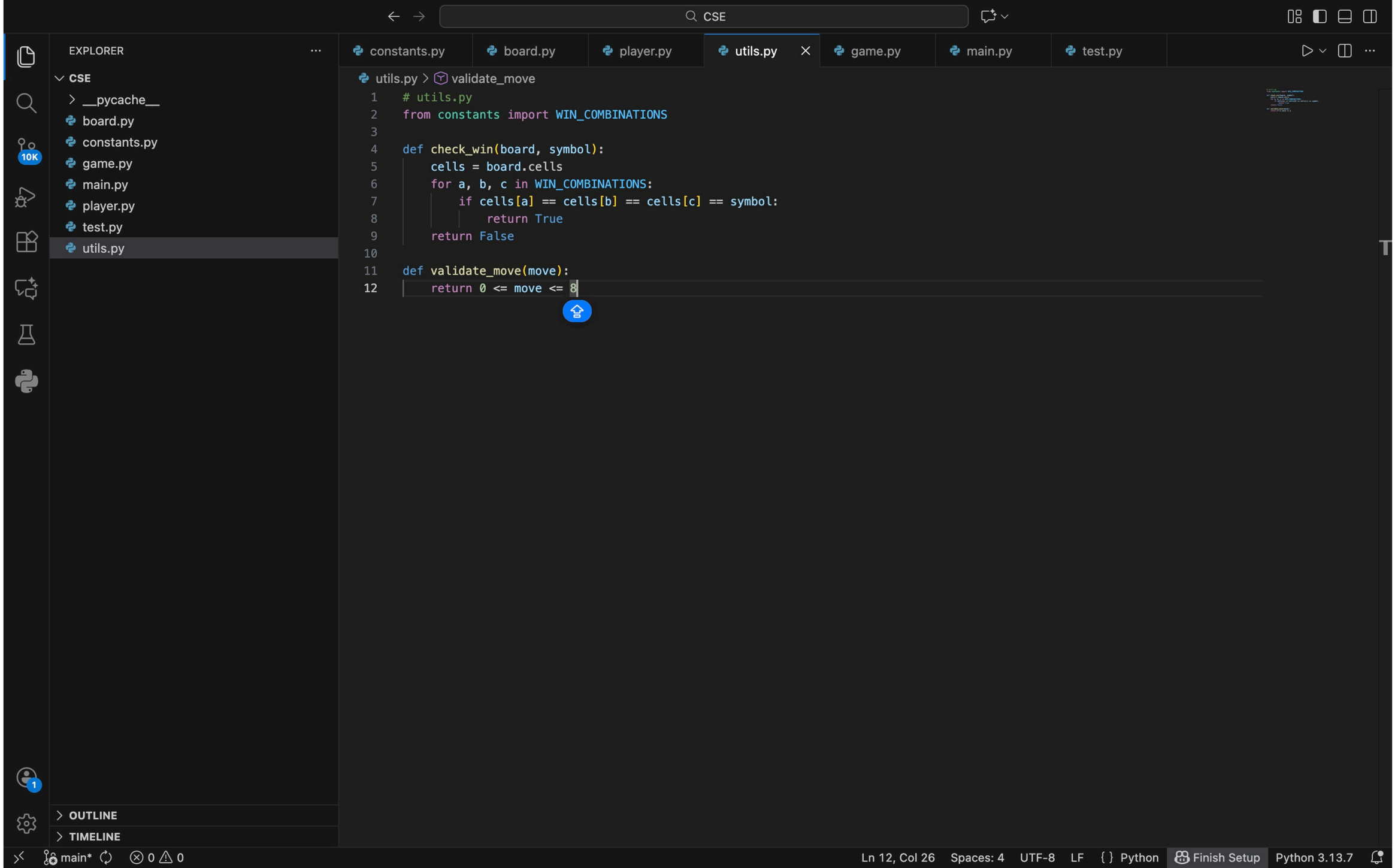
SOURCE CODE


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
constants.py > ...
1 # constants.py
2 EMPTY = " "
3 PLAYER_X = "X"
4 PLAYER_O = "O"
5
6 WIN_COMBINATIONS = [
7 (0,1,2), (3,4,5), (6,7,8), # rows
8 (0,3,6), (1,4,7), (2,5,8), # columns
9 (0,4,8), (2,4,6) # diagonals
10]
OUTLINE
TIMELINE
main*
0 0
Ln 10, Col 2
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
board.py > Board > reset
# board.py
from constants import EMPTY
class Board:
def __init__(self):
self.cells = [EMPTY] * 9
def display(self):
print("\n")
for i in range(0, 9, 3):
print(f" {self.cells[i]} | {self.cells[i+1]} | {self.cells[i+2]} ")
if i < 6:
print("---+---+---")
print("\n")
def update_cell(self, index, symbol):
if self.cells[index] == EMPTY:
self.cells[index] = symbol
return True
return False
def is_full(self):
return EMPTY not in self.cells
def reset(self):
self.cells = [EMPTY] * 9
main*
0 0
Ln 26, Col 33
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
player.py > Player > get_move
# player.py
class Player:
def __init__(self, name, symbol):
self.name = name
self.symbol = symbol
def get_move(self):
try:
return int(input(f"{self.name} ({self.symbol}), enter position (1-9): ")) - 1
except ValueError:
return -1
main*
0 0
Ln 11, Col 22
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
utils.py > validate_move
# utils.py
from constants import WIN_COMBINATIONS
def check_win(board, symbol):
cells = board.cells
for a, b, c in WIN_COMBINATIONS:
if cells[a] == cells[b] == cells[c] == symbol:
return True
return False
def validate_move(move):
return 0 <= move <= 8
main*
0 0
Ln 12, Col 26
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7

CSE

EXPLORER

CSE

- __pycache__
- board.py
- constants.py
- game.py
- main.py
- player.py
- test.py
- utils.py

OUTLINE

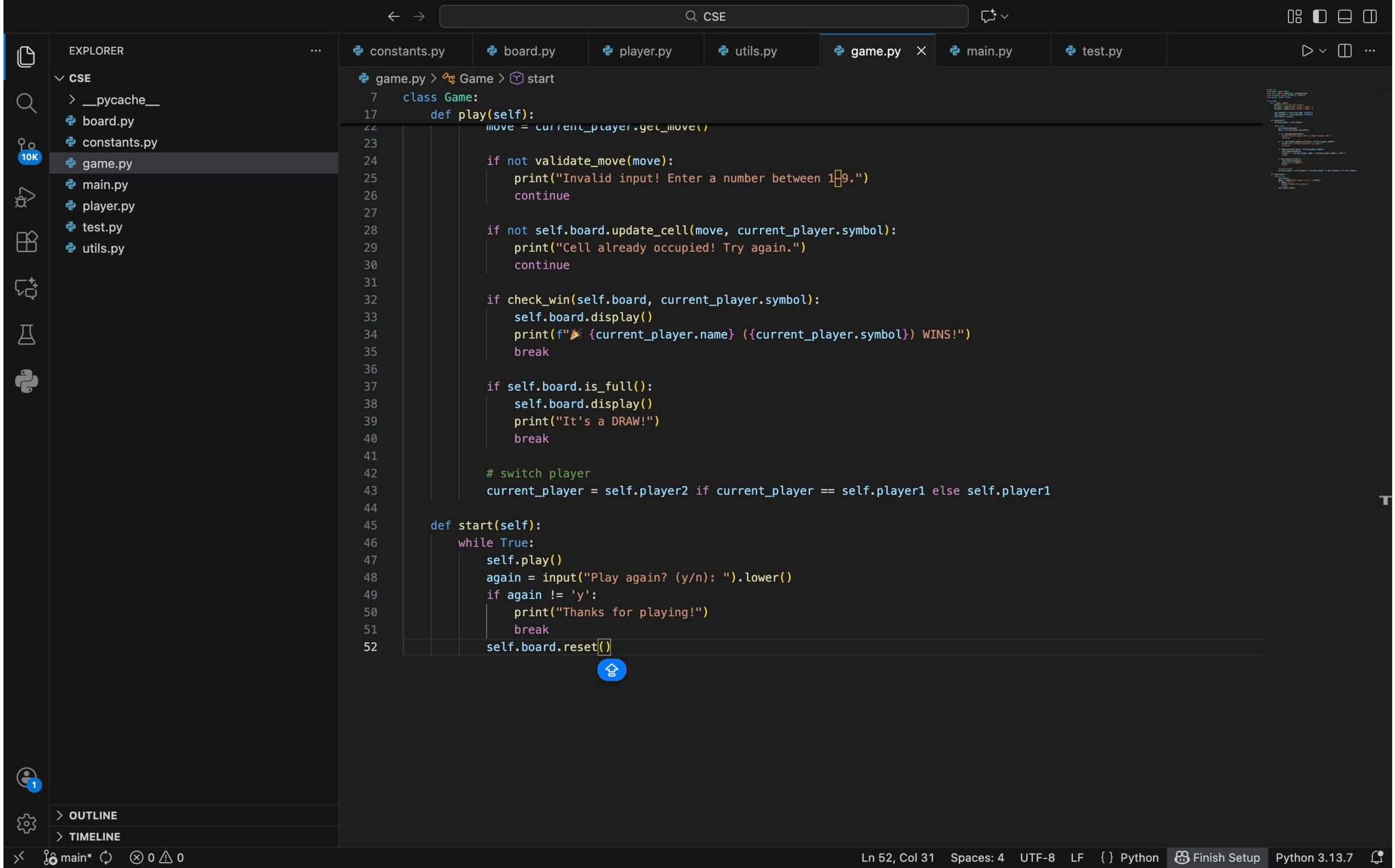
TIMELINE

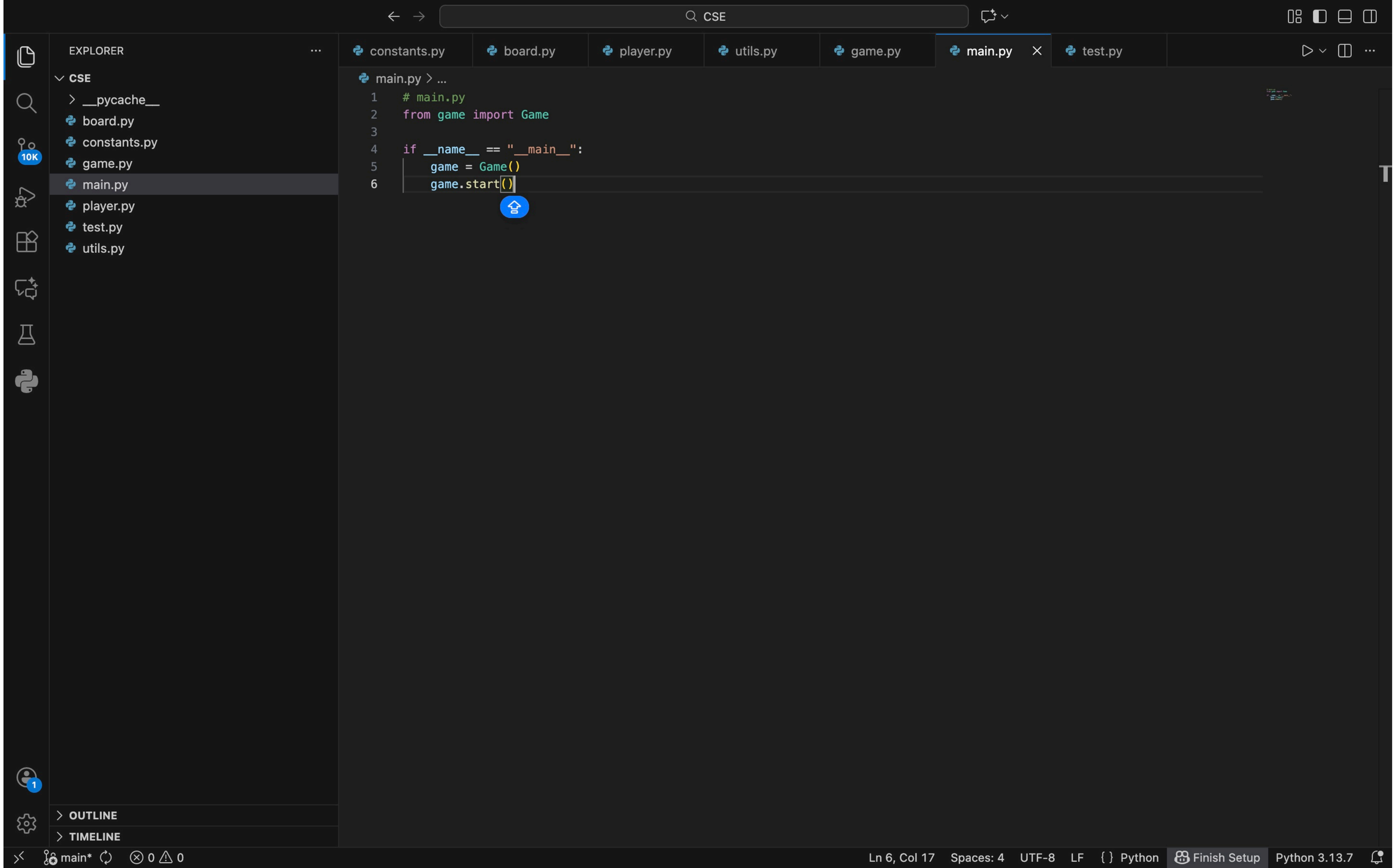
constants.py | board.py | player.py | utils.py | game.py | main.py | test.py

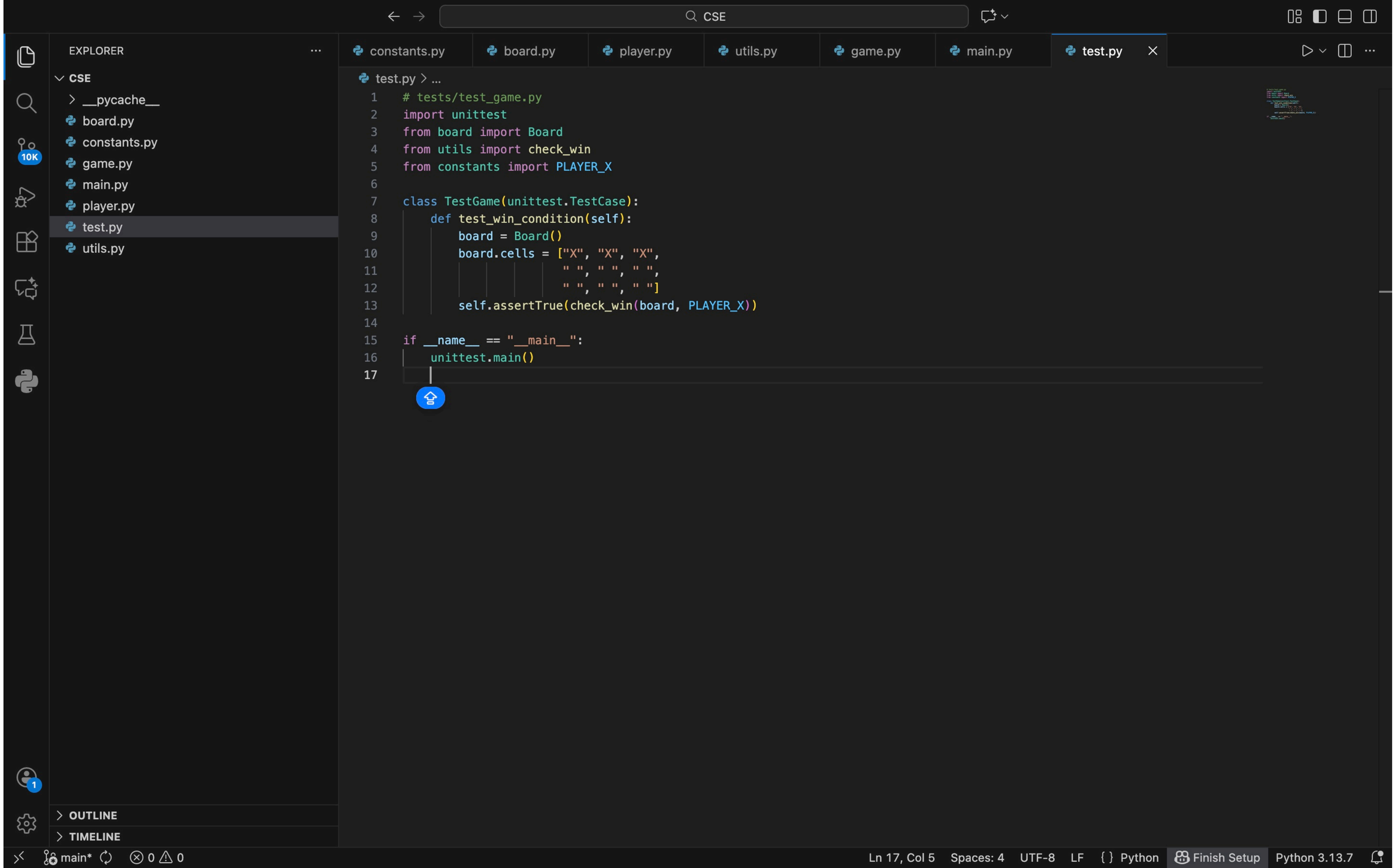
game.py > Game > start

```python
# game.py
from board import Board
from utils import check_win, validate_move
from constants import PLAYER_X, PLAYER_O
from player import Player

class Game:
    def __init__(self):
        print("=== Tic Tac Toe ===\n")
        p1_name = input("Enter Player 1 Name: ")
        p2_name = input("Enter Player 2 Name: ")

        self.player1 = Player(p1_name, PLAYER_X)
        self.player2 = Player(p2_name, PLAYER_O)
        self.board = Board()

    def play(self):
        current_player = self.player1

        while True:
            self.board.display()
            move = current_player.get_move()

            if not validate_move(move):
                print("Invalid input! Enter a number between 1 9.")
                continue

            if not self.board.update_cell(move, current_player.symbol):
                print("Cell already occupied! Try again.")
                continue

            if check_win(self.board, current_player.symbol):
                self.board.display()
                print(f"🎉 {current_player.name} ({current_player.symbol}) WINS!")
                break

            if self.board.is_full():
                self.board.display()
                print("It's a DRAW!")
                break

            # switch player
            current_player = self.player2 if current_player == self.player1 else self.player1
```

main* 0 0 | Ln 52, Col 31 | Spaces: 4 | UTF-8 | LF | { } Python | Finish Setup | Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
game.py > Game > start
class Game:
def play(self):
move = current_player.get_move()
if not validate_move(move):
print("Invalid input! Enter a number between 1-9.")
continue
if not self.board.update_cell(move, current_player.symbol):
print("Cell already occupied! Try again.")
continue
if check_win(self.board, current_player.symbol):
self.board.display()
print(f"🎉 {current_player.name} ({current_player.symbol}) WINS!")
break
if self.board.is_full():
self.board.display()
print("It's a DRAW!")
break
# switch player
current_player = self.player2 if current_player == self.player1 else self.player1
def start(self):
while True:
self.play()
again = input("Play again? (y/n): ").lower()
if again != 'y':
print("Thanks for playing!")
break
self.board.reset()
main*
0 0
Ln 52, Col 31
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
main.py > ...
1 # main.py
2 from game import Game
3
4 if __name__ == "__main__":
5 game = Game()
6 game.start()
main*
0 0
Ln 6, Col 17
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7


CSE
EXPLORER
CSE
__pycache__
board.py
constants.py
game.py
main.py
player.py
test.py
utils.py
OUTLINE
TIMELINE
constants.py
board.py
player.py
utils.py
game.py
main.py
test.py
test.py > ...
# tests/test_game.py
import unittest
from board import Board
from utils import check_win
from constants import PLAYER_X
class TestGame(unittest.TestCase):
def test_win_condition(self):
board = Board()
board.cells = ["X", "X", "X",
" ", " ", " ",
" ", " ", " "]
self.assertTrue(check_win(board, PLAYER_X))
if __name__ == "__main__":
unittest.main()
main*
0 0
Ln 17, Col 5
Spaces: 4
UTF-8
LF
Python
Finish Setup
Python 3.13.7