פרויקט סיום שנה ב-Python

**תאריך הגשה**: 14/05/18

**שם**: דן קצוב-פייגין

**תעודת זהות**: 323002915

**שם המשחק**: ארבע בשורה

**מוגש ל**דלית אורן

תוכן

[חוקי ומהלך המשחק 3](#_Toc513895968)

[תכנות המשחק 3](#_Toc513895969)

[תקציר מהלך המשחק 3](#_Toc513895970)

[קבצי ומחלקות הפרויקט 3](#_Toc513895971)

[קובץ board 3](#_Toc513895972)

[קובץ game 4](#_Toc513895973)

[תיעוד פעולות המשחק 4](#_Toc513895974)

[קובץ board 4](#_Toc513895975)

[מחלקה Board 4](#_Toc513895976)

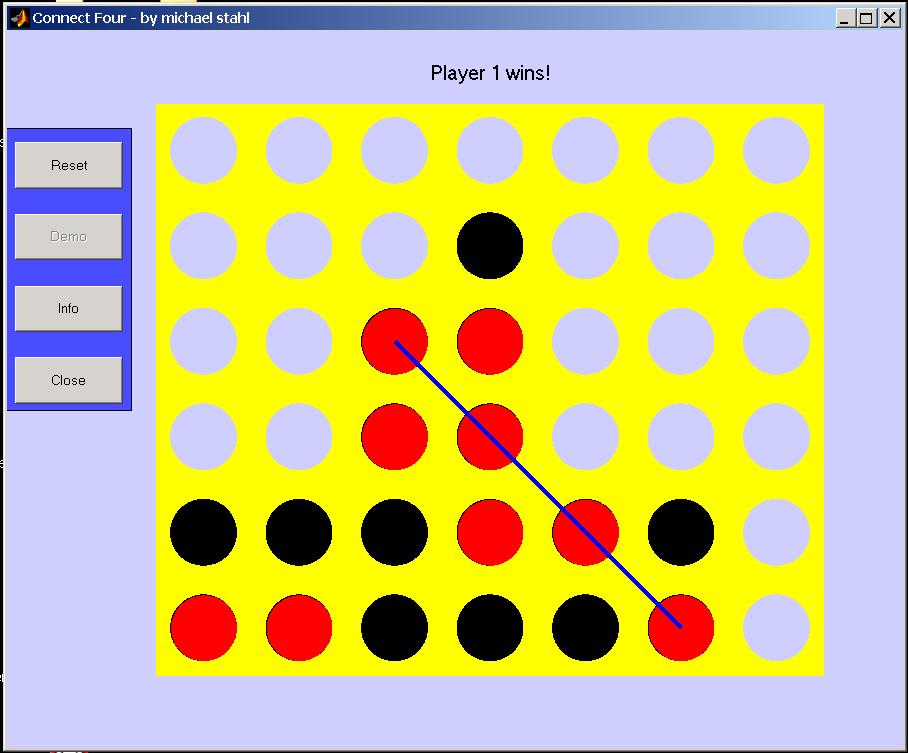
[מחלקת ColumnOutOfBoundsError 5](#_Toc513895977)

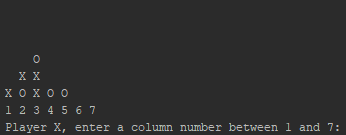
[מחלקת FullColumnError 5](#_Toc513895978)

[קובץ game 5](#_Toc513895979)

# חוקי ומהלך המשחק

המשחק "ארבע בשורה" הוא משחק חשיבה לשני שחקנים המשוחק על לוח אנכי בעל שש שורות, כאשר כל שורה מכילה שבעה תאים. לכל שחקן יש אסימונים בצבע אחד. המשחק מנוהל בתורות. כל שחקן בתורו מפיל אסימון בצבע שלו לאחד משני הטורים. המנצח הוא הראשון ליצור רצף של ארבעה אסימונים בצבע שלו בשורה, בטור או באלכסון.





*לוח ארבע בשורה במשחק*

*משחק ארבע בשורה בו ניצח השחקן האדום*

# תכנות המשחק

## תקציר מהלך המשחק

המשחק משוחק בין שני שחקנים. בין כל תור מודפס מצבו הנוכחי של הלוח. כל שחקן בתורו מכניס מספר של טור אליו הוא רוצה להכניס את האסימון שלו. אם השחקן מכניס מספר של טור שלא קיים בלוח או שהטור שהוכנס מספרו כבר התמלא, השחקן יתבקש להכניס מספר של טור כל עוד הוא לא יכניס מספר תקין. אם השחקן ניצח במשחק, תודפס הודעה מתאימה. אם הלוח התמלא לפני שהוכרז מנצח (ואי אפשר להכניס עוד אסימונים), תוצאת המשחק תהיה תיקו ותודפס הודעה מתאימה.

## קבצי ומחלקות הפרויקט

פרויקט המשחק מורכב משני קבצים. קובץ אחד בשם board וקובץ שני בשם game.

### קובץ board

קובץ זה מכיל שלוש מחלקות: מחלקה אחת מרכזית בשם Board ושתי מחלקות נוספות המייצגות exceptions, או חריגות בעברית. חריגה אחת היא ColumnOutOfBoundsError והיא נזרקת כאשר מוכנס מספר של טור שלא קיים בלוח. החריגה השנייה היא FullColumnError והיא נזרקת כאשר מוכנס מספר של טור שכבר התמלא.

לב המשחק הוא המחלקה Board והיא מייצגת לוח של משחק "ארבע בשורה". הלוח מיוצג על ידי רשימה של רשימות בשם rows. בכל רשימה פנימית יש מספר תאים כמספר הטורים בלוח, וכל תא מאותחל בתו רווח בודד. (' '). השחקן הראשון והשני מיוצגים על ידי תווי 'X' ו-'O' בהתאמה. בכל פעם המוכנס מספר טור, השיטה insert מכניסה את האסימון של השחקן הנוכחי לתא הפנוי הגבוה ביותר בטור שהכניס השחקן את מספרו ומחזירה את מספר השורה שהוכנס אליה האסימון.

אחרי כל מהלך, השיטה is\_winner תבדוק האם המהלך האחרון הביא לניצחון. זאת על ידי בדיקת ניצחון באחת משלוש דרכי הניצחון האפשריות במשחק: יצירת רצף של ארבעה אסימונים בשורה, בטור, או באלכסון. השיטה is\_winner קוראת לארבע שיטות נוספות, הבודקות אם קיים ניצחון בשורה, בטור, באלכסון העולה או באלכסון היורד הנוכחיים. כל אחת מהשיטות הללו הופכות את השורה, הטור, האלכסון הראשי או האלכסון היורד הנוכחיים לרצף בפורמט יחיד. כל אחת מהשיטות הללו קוראת לשיטה is\_winner\_in\_sequence הבודקת האם יש ניצחון ברצף כללי.

בנוסף, קיימות בתוך המחלקה Board פונקציות הזורקות את החריגות המפורטות לעיל.

קובץ game

קובץ זה הוא הקובץ המריץ את המשחק ואת מהלכיו. המשחק עצמו רץ בתוך הפעולה main, ובתוכה בתוך לולאה המחליפה בין השחקנים בכל תור. אם יש ניצחון, מודפסת הודעה מתאימה והמשחק נגמר. אם לא, השיטה Board.is\_full בודקת האם התמלא הלוח, כלומר המשחק הגיע למצב של תיקו. אם כן, מודפסת הודעה מתאימה והמשחק נגמר. אם אינם מתקיימים ניצחון של אחד השחקנים או תיקו, התור עובר לשחקן הבא, וכך חוזר חלילה עד שמתקיימים ניצחון של אחד השחקנים או תיקו.

בנוסף, קיימות בקובץ זה ארבע פעולות קטנות יותר, המשמשות לקליטת מספרי הטורים ולהדפסות שונות.

# תיעוד פעולות המשחק

כעת נעבור אל תיעוד מחלקות המשחק. נתחיל מהקובץ board עבור כל מחלקה ולאחר מכן נעבור אל הקובץ game.

משמעות שני קווים תחתונים לפני פונקציה או משתנה היא שהפונקציה או המשתנה הם פרטיים של המחלקה. פונקציה המוקפת בשני קווים תחתונים משני הצדדים היא פונקציה מיוחדת של המחלקה.

## קובץ board

קובץ זה מכיל שלוש מחלקות: Board, ColumnOutOfBoundsError, ו-FullColumnError.

### מחלקה Board

|  |  |
| --- | --- |
| **תיאור השיטה** | **שיטה** |
| הפעולה הבונה של המחלקה. בנה עצם של המחלקה, ומאתחלת מערך רשימה של רשימות המייצגת את הלוח. | **\_\_init\_\_**() |
| החזר מחרוזת המתארת את הלוח. | **\_\_str\_\_**() |
| החזר האם הלוח מלא. | **is\_full**() |
| פרמטרים: מספר הטור, שחקן.  הכנס אסימון של שחקן אל הטור שמספרו הוא מספר הטור באמצעות insert והחזר האם התקיים ניצחון. | **move**(*column*, *player*) |
| פרמטרים: מספר הטור, שחקן.  הכנס אסימון של שחקן אל הטור שמספרו הוא מספר טור והחזר את מספר השורה אליה הוכנס האסימון – זו תהיה השורה הפנויה הגבוהה ביותר. | **\_\_insert**(*self*, *column*, *player*) |
| פרמטרים: שורה אליה הוכנס האסימון, טור אליו הוכנס האסימון, שחקן.  החזר אם מתקיים ניצחון בשורה על ידי הפיכת השורה לרצף וקריאה לפעולה is\_winner\_in\_sequence. | **\_\_is\_winner\_in\_row**(*row*, *column*, *player*) |
| פרמטרים: טור אליו הוכנס האסימון, טור אליו הוכנס האסימון, שחקן.  החזר אם מתקיים ניצחון בטור על ידי הפיכת הטור לרצף וקריאה לפעולה is\_winner\_in\_sequence. | **\_\_is\_winner\_in\_column**(*row*, *column*, *player*) |
| פרמטרים: שורה אליה הוכנס האסימון, טור אליו הוכנס האסימון, שחקן.  החזר אם מתקיים ניצחון באלכסון היורד על ידי הפיכת האלכסון היורד לרצף על ידי מציאת כל התאים בלוח שההפרש בין מספר הטור לבין מספר השורה הוא ההפרש בין מספר השורה הנוכחית למספר הטור הנוכחי וקריאה לשיטה is\_winner\_in\_sequence. | **\_\_is\_winner\_in\_descending\_diagonal**(*row*, *column*, *player*) |
| פרמטרים: שורה אליה הוכנס האסימון, טור אליו הוכנס האסימון, שחקן.  החזר אם מתקיים ניצחון באלכסון העולה על ידי הפיכת האלכסון היורד לרצף על ידי מציאת כל התאים בלוח שהסכום של מספר הטור ושל מספר השורה הוא הסכום בין מספר השורה הנוכחית למספר הטור הנוכחי וקריאה לשיטה is\_winner\_in\_sequence. | **\_\_is\_winner\_is\_ascending\_diagoanl**(*row*, *column*, *player*) |
| פרמטרים: רצף, אינדקס אחרון, שחקן.  החזר אם מתקיים ניצחון ברצף. אינדקס הוא המקום האחרון ברצף בו הוכנס אסימון. | **\_\_is\_winner\_in\_sequence**(*sequence*, *index*, *player*) |
| פרמטרים: מחלקה, מספר טור  זרוק ColumnOutOfBoundsError אם הוכנס מספר טור האינו קיים בלוח. | **\_\_raise\_if\_out\_of\_bounds**(*cls*, *column*) |
| פרמטרים: מספר טור  זרוק FullColumnError אם הוכנס מספר טור שהתמלא כבר. | **\_\_raise\_if\_column\_is\_full**(*column*) |

### 

### מחלקת ColumnOutOfBoundsError

|  |  |
| --- | --- |
| **תיאור השיטה** | **שיטה** |
| פרמטרים: מספר טור  אתחל חריגת ColumnOutOfBoundsError. | **\_\_init\_\_**(*column*) |

### מחלקת FullColumnError

|  |  |
| --- | --- |
| **תיאור השיטה** | **שיטה** |
| פרמטרים: מספר טור  אתחל חריגת ColumnOutOfBoundsError. | **\_\_init\_\_**(*column*) |

### 

### קובץ game

קובץ זה אינו מכיל מחלקות, אך כן מכיל פעולות.

|  |  |
| --- | --- |
| **תיאור הפעולה** | **פעולה** |
| המשחק עצמו רץ בתוך הפעולה main, ובתוכה בתוך לולאה המחליפה בין השחקנים בכל תור. בתחילת כל מהלך מודפס הלוח הנוכחי. לאחר מכן אנו נכנסים ללולאה הקולטת מספרי טור מהשחקן עד שלא הוכנס מספר תקין. אם יש ניצחון, מודפסת הודעה מתאימה והמשחק נגמר. אם לא, השיטה Board.is\_full בודקת האם התמלא הלוח, כלומר המשחק הגיע למצב של תיקו. אם כן, מודפסת הודעה מתאימה והמשחק נגמר. אם אינם מתקיימים ניצחון של אחד השחקנים או תיקו, התור עובר לשחקן הבא, וכך חוזר חלילה עד שמתקיימים ניצחון של אחד השחקנים או תיקו. | **main**() |
| פרמטרים: לוח  הדפס את הלוח הנוכחי ושורה המאתרת את מספרי הטורים | **print\_board**(*board*) |
| פרמטרים: שחקן  הדפס ששחקן ניצח את המשחק. | **print\_win**(*player)* |
| הדפס שהלוח מלא, המשחק נגמר במצב תיקו. | **print­\_game\_over**() |
| פרמטרים: שחקן  קלוט מספר טור מהשחקן. | **column\_input**(*player*) |

# קבצי המשחק

כאמור, המשחק מורכב משני קבצים המצורפים כאן: הקובץ board והקובץ game.

## הקובץ board

**class** Board(object):  
 *"""Class that represents a Connect Four board."""* ROWS = 6  
 COLUMNS = 7  
 EMPTY = **' '** TARGET = 4  
  
 **def** \_\_init\_\_(self):  
 *"""Initialize a Connect Four board."""* self.\_\_rows = [[self.EMPTY **for** \_ **in** xrange(self.COLUMNS)] **for** \_ **in** xrange(self.ROWS)]  
  
 **def** \_\_str\_\_(self):  
 *"""* **:return***: a string describing the current board* **:rtype***: str  
 """* **return '\n'**.join(**' '**.join(row) **for** row **in** self.\_\_rows)  
  
 **def** is\_full(self):  
 *"""* **:return***: whether the board is full.* **:rtype***: bool  
 """* **return** all(self.EMPTY **not in** row **for** row **in** self.\_\_rows)  
  
 **def** move(self, column, player):  
 *"""  
 Make one move in the game and return whether player won the game.* **:param** *column: column the player inputted* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game* **:rtype***: bool  
 """* row = self.\_\_insert(column, player)  
 **return** self.\_\_is\_winner(row, column, player)  
  
 **def** \_\_insert(self, column, player):  
 *"""  
 Insert a player's token into the board.* **:param** *column: column number the player entered* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: row the token fell into* **:rtype***: int  
 """* self.\_\_raise\_if\_out\_of\_bounds(column)  
 self.\_\_raise\_if\_column\_is\_full(column)  
 **for** i, row **in** reversed(list(enumerate(self.\_\_rows))):  
 **if** row[column] != self.EMPTY:  
 **continue** row[column] = player  
 **return** i

**def** \_\_is\_winner(self, row, column, player):  
 *"""* **:param** *row: row player inserted his last token into* **:type** *row: int* **:param** *column: column player inserted his last token into* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game* **:rtype***: bool  
 """* **return** self.\_\_is\_winner\_in\_row(row, column, player) \  
 **or** self.\_\_is\_winner\_in\_column(row, column, player) \  
 **or** self.\_\_is\_winner\_in\_descending\_diagonal(row, column, player) \  
 **or** self.\_\_is\_winner\_in\_ascending\_diagonal(row, column, player)  
  
 **def** \_\_is\_winner\_in\_row(self, row, column, player):  
 *"""* **:param** *row: row player inserted his last token into* **:type** *row: int* **:param** *column: column player inserted his last token into* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game by creating a sequence of four tokens in a row* **:rtype***: bool  
 """* **return** self.\_\_is\_winner\_in\_sequence(self.\_\_rows[row], column, player)  
  
 **def** \_\_is\_winner\_in\_column(self, row, column, player):  
 *"""* **:param** *row: row player inserted his last token into* **:type** *row: int* **:param** *column: column player inserted his last token into* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game by creating a sequence of four tokens in a column* **:rtype***: bool  
 """* **return** self.\_\_is\_winner\_in\_sequence([r[column] **for** r **in** self.\_\_rows], row, player)

**def** \_\_is\_winner\_in\_descending\_diagonal(self, row, column, player):  
 *"""* **:param** *row: row player inserted his last token into* **:type** *row: int* **:param** *column: column player inserted his last token into* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game by creating a sequence of four tokens in a descending diagonal* **:rtype***: bool  
 """* **if** row >= column:  
 difference = row - column  
 **return** self.\_\_is\_winner\_in\_sequence(  
 [self.\_\_rows[i][i - difference]  
 **for** i **in** xrange(difference, min(self.ROWS, difference + self.COLUMNS))],  
 column, player)  
  
 difference = column - row  
 **return** self.\_\_is\_winner\_in\_sequence(  
 [self.\_\_rows[i - difference][i] **for** i **in** xrange(difference, min(difference + self.ROWS, self.COLUMNS))],  
 row, player)  
  
 **def** \_\_is\_winner\_in\_ascending\_diagonal(self, row, column, player):  
 *"""* **:param** *row: row player inserted his last token into* **:type** *row: int* **:param** *column: column player inserted his last token into* **:type** *column: int* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game by creating a sequence of four tokens in a ascending diagonal* **:rtype***: bool  
 """* addition = row + column  
 **if** addition < self.ROWS:  
 **return** self.\_\_is\_winner\_in\_sequence([self.\_\_rows[addition - i][i] **for** i **in** xrange(addition + 1)], column,  
 player)  
 **return** self.\_\_is\_winner\_in\_sequence(  
 [self.\_\_rows[addition - i][i] **for** i **in** xrange(addition - (self.ROWS - 1), min(addition + 1, self.COLUMNS))],  
 column - addition + (self.ROWS - 1), player)  
  
 @classmethod  
 **def** \_\_is\_winner\_in\_sequence(cls, sequence, index, player):  
 *"""* **:param** *sequence: sequence of tokens* **:type** *sequence: list* **:param** *index: index of the latest token in the list* **:param** *player: current player* **:type** *player: str* **:return***: whether player won the game by creating a sequence of four tokens* **:rtype***: bool  
 """* target\_slice = [player] \* cls.TARGET  
 **return** any(sequence[start:start + cls.TARGET] == target\_slice  
 **for** start **in** xrange(max(0, index - (cls.TARGET - 1)), index + 1))

@classmethod  
 **def** \_\_raise\_if\_out\_of\_bounds(cls, column):  
 *"""  
 Raise a ColumnOutOfBoundsError exception if the inputted column is out of board boundaries.* **:param** *column: column number inputted* **:type** *column: int* **:raise***: ColumnOutOfBoundsError  
 """* **if not** 0 <= column < cls.COLUMNS:  
 **raise** ColumnOutOfBoundsError(column)  
  
 **def** \_\_raise\_if\_column\_is\_full(self, column):  
 *"""  
 Raise a FullCommonError exception if the inputted column is already full.* **:param** *column: column number inputted* **:type** *column: int* **:raise***: FullColumnError  
 """* **if** self.\_\_rows[0][column] != self.EMPTY:  
 **raise** FullColumnError(column)

**class** ColumnOutOfBoundsError(IndexError):  
 *"""  
 Exception for handling columns numbers inputted for columns that are out of board boundaries.  
 """* **def** \_\_init\_\_(self, column):  
 *"""  
 Initialize a ColumnOutOfBoundsError exception.* **:param** *column: column inputted for checking if it is out of board boundaries* **:type** *column: int  
 """* super(ColumnOutOfBoundsError, self).\_\_init\_\_()  
 self.column = column  
  
  
**class** FullColumnError(Exception):  
 *"""  
 Exception for handling columns numbers for columns that are already full.  
 """* **def** \_\_init\_\_(self, column):  
 *"""  
 Initialize a FullColumnError exception.* **:param** *column: column inputted for checking if it is already full* **:type** *column: int  
 """* super(FullColumnError, self).\_\_init\_\_()  
 self.column = column

## הקובץ game

**from** itertools **import** cycle  
  
**from** board **import** Board, ColumnOutOfBoundsError, FullColumnError  
  
  
**def** main():  
 *"""  
 Main function running the game.* **:except***: ColumnOutOfBoundsError, FullColumnError, Exception* **:return***: when the game has ended  
 """* board = Board()  
  
 **for** player **in** cycle([**'X'**, **'O'**]):  
 print\_board(board)  
 **while** True:  
 **try**:  
 column = column\_input(player)  
 **if** board.move(column, player):  
 **print** board  
 print\_win(player)  
 **return  
 break  
 except** ColumnOutOfBoundsError **as** error:  
 **print 'Column {} is not in board'**.format(error.column + 1)  
 **except** FullColumnError **as** error:  
 **print 'Column {} is full'**.format(error.column + 1)  
 **except** Exception **as** error:  
 **print** error  
  
 **if** board.is\_full():  
 **print** board  
 print\_game\_over()  
 **return  
  
  
def** print\_board(board):  
 *"""  
 Print the current board.* **:param** *board: current board* **:type** *board: Board  
 """* **print** board  
 **print ' '**.join(str(column) **for** column **in** xrange(1, board.COLUMNS + 1))  
  
  
**def** print\_win(player):  
 *"""  
 Print the winner's sign.* **:param** *player: the winner* **:type** *player: str  
 """* **print 'Player {} won! :)'**.format(player)  
  
  
**def** print\_game\_over():  
 *"""  
 Print that the board is full and the game is over.  
 """* **print 'Board is full, tie.'  
  
  
def** column\_input(player):  
 *"""  
 Input a column number from player.* **:param** *player: player to ask the input for* **:type** *player: str* **:return***: column number player inputted* **:rtype***: int  
 """* **return** int(raw\_input(**'Player {}, enter a column number between 1 and {}: '**.format(player, Board.COLUMNS))) - 1  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()