עבודה מס' 2: פונקציות, מערכים

הוראות מקדימות :

1. העבודה כתובה בלשון זכר (מטעמי נוחיות), אך פונה לשני המינים.
2. ניתן ומומלץ לבצע את העבודה בזוגות בהתאם להוראות המופיעות בסילבוס הקורס.
3. יש להגיש את קבצי ה- java הערוכים בלבד, מכווצים כקובץ ZIP יחיד. אין לשנות את שמות הקבצים, להגיש קבצים נוספים, ליצור תיקיות, להגיש מספר קבצים עבור אותה המשימה, או להגיש קובץ יחיד למספר משימות. שם הקובץ יכול להיות על פי שיקולכם ובאנגלית בלבד. **קבצים שיוגשו בפורמט שונה מzip לא ייבדקו**.
4. את הקובץ יש להגיש ב Submission System.
5. העבודה תיבדק **באופן אוטומטי**. לכן, יש להקפיד על ההוראות ולבצע אותן **במדויק**. כל חוסר מעקב אחר ההוראות **יגרור פגיעה משמעותית בציון**.
6. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד ברור, מתן שמות משמעותיים למשתנים, הזחות (אינדנטציה), **והוספת הערות בקוד** המסבירות את תפקידם של מקטעי קוד שונים. אין צורך למלא את הקוד בהערות סתמיות אך חשוב לכתוב הערות בנקודות קריטיות המסבירות קטעים חשובים בקוד. כתיבת קוד אשר אינו עומד בסטנדרטים אלו תגרור הפחתה בציון העבודה. בנוסף, הערות יש לרשום אך ורק ב**אנגלית** (כתיבת הערות בכל שפה אחרת שקולה לאי כתיבת הערות).
7. בכדי לוודא שהגשתם את התרגיל באופן תקין – הורידו את קובץ הZIP שהגשתם ב Submission System למחשב שלכם, חלצו אותו ונסו להדר ולהריץ את התוכנית. הליך זה הוא חשוב, עקב מקרים שקרו בעבר בהם הקובץ שהועלה למערכת ההגשות היה פגום.
8. במידה ואינכם בטוחים מהו הפירוש המדויק להוראה מסוימת, או בכל שאלה אחרת הקשורה **בתוכן** העבודה, אנא היעזרו בפורום או בשעות הקבלה של האחראים על העבודה (פרוט שעות הקבלה מופיע באתר הקורס). בכל בעיה **אישית** הקשורה בעבודה (מילואים, אשפוז וכו'), אנא צרו את הפניה המתאימה במערכת הגשת העבודות כפי שמוסבר בסילבוס שבאתר הקורס (כאן).
9. שימו לב כי בעבודה ניתנו 5 נקודות "תמריץ" עבור מעקב אחר הוראות העבודה. במידה וההוראות לא מולאו (שכחתם להוסיף שותף במידה ובחרתם להגיש בזוגות, חוסר בדיקה של פורמט הקבצים, שמות לא נכונים וכדומה) הדבר יגרור הורדה של נקודות אלו.
10. בעבודה זו ניתן לצבור עד 140 נקודות:

10.1. 5 נקודות עבור מעקב אחר הוראות (סעיף 9)

10.2. 15 נקודות עבור הבדיקה הידנית - כתיבת הערות (7 נק'), הזחה (4 נק') ושמות משתנים (4 נק').

10.3. 80 נקודות על כתיבת הקוד בהתאם לדרישות התרגיל.

10.4. 0-40 נקודות עבור חלק הבונוס (משימה 13).

בתרגיל זה נצבור ניסיון בשימוש בפונקציות ובמערכים.
התרגיל עוסק במשחק הלוח המיועד לשני משתתפים, הקרוי רוורסי (Reversi). במשחק, לכל שחקן דיסקיות בצבע אופייני, (למשל, כחול ואדום). המשחק נפתח כשלכל שחקן שתי דיסקיות בצבע שלו, והן מסודרות במרכז הלוח בהצלבה (ראו דוגמה בקישור שלמטה). המטרה היא לכבוש את דיסקיות היריב. כיבוש מתאפשר ע"י הצבת דיסקית כך שבינה ובין דיסקית אחרת של השחקן נמצאת לפחות דיסקית אחת של היריב. לאחר שמשבצות הלוח מתמלאות, נקבע המנצח על פי הצבע השליט על הלוח (הצבע בו צבועות רוב הדיסקיות).
תוכלו לקרוא עוד על המשחק בלינק הבא, וכן להתנסות במשחק בלינק הבא.

עבודה זו מחולקת לשלושה חלקים, בחלק הראשון תממשו קטעי קוד שיעזרו לכם לבדוק את תקינות הקוד שלכם ואת נכונותו. בחלק השני תממשו את הפונקציות הנדרשות בכדי לאפשר משחק Reversi ובחלק השלישי של העבודה תממשו אסטרטגיות משחק עבור סוגי שחקנים שונים, כולל האפשרות להוסיף אסטרטגיה משלכם.

לתרגיל מצורפים 2 קבצים:

1. **ReversiPlay.java:** קובץ זה מכיל תבניות עבור הפונקציות אותן תצטרכו להשלים. בנוסף, מכיל הקובץ פונקציות נוספות:
   - פונקציות בדיקה חלקיות עבור חלק מהמשימות של חלק ב'. משימות אלו מתחילות תמיד בשמות testPart2Assignment. שימו לב כי פונקציות אלו מסתמכות על נכונות הפונקציות שכתבתם בחלק א'.
   - פונקציית main המכילה קריאות לפונקציות הבדיקה.
2. **ReversiGUI.java:** מטרתו של קובץ זה היא לאפשר לכם לבחון את הקוד שלכם. הוא מאפשר לכם לשחק אחד מול השני או מול המחשב, וכך לבחון את מימושי הפונקציות שלכם.

על מנת להשתמש בפונקציה זו, יש לקמפל את הקובץ ReversiGUI.java ליצירת ReversiGUI.class. קבצים אלו יימצאו בספריה בה נמצא הקובץ ReversiPlay.java.
שימו לב: אין צורך לקרוא ולהבין את הקוד בקובץ ReversiGUI.java.

- עליכם להגיש קובץ אחד בלבד: ReversiPlay.java. השתמשו בתבניות שניתנו לכם עבור הפונקציות השונות. במידה ובחרתם להגיש את הפונקציה myPlayer, יש להוסיף אותה לקובץ המוגש ReversiPlay.java. כמו כן, יש להוסיף קובץ PDF כמתואר בתרגיל.
- תוכלו להוסיף פונקציות עזר נוספות לבחירתכם. שימו לב: קריאות לפונקציות עזר שכתבתם, ייעשו **רק** מתוך הפונקציות אותן אתם מממשים, ולא מתוך פונקציית ה-main שנועדה לבדיקה. במידה ואכן כתבתם פונקציות עזר נוספות, יש לצרף אותן לקובץ ההגשה: ReversiPlay.java.

חלק א' - מימוש קוד לבדיקת התרגיל (10 נק')

בחלק זה של התרגיל תממשו מספר פונקציות אשר תעזורנה לכם בהמשך העבודה. המטרה של פונקציות אלו היא להקל עליכם **לבדוק את נכונות הקוד** אשר תכתבו. אחד הדברים הקשים (והכי פחות מתוגמלים) בתכנות היא היכולת לבדוק את הקוד שרשמנו. נניח שרשמנו פונקציה אשר מטרתה לבצע שינוי כלשהו על מטריצה (שינוי ערכים מסויימים), וברצוננו לבדוק את תקינות הקוד. בפנינו עומדות מספר שאלות:

1. האם הקוד שלנו עובד גם על מטריצה ריקה, ערך null (טיפול במקרי קצה)
2. כיצד עלינו לוודא שהקוד שלנו נכון (בדיקת פלט)

מתן מענה לשאלות אלו נותן מענה ראשוני לבדיקת הקוד שלנו. לכן, עלינו לדעת גם לספק לקוד שלנו מקרים רבים שיכסו כמה שיותר אפשרויות קלט מ"העולם האמיתי" וגם לרשום קטעי קוד שמטרתם להקל עלינו בבדיקת נכונות הפלט. מכיוון שבעבודה זו רוב העבודה נעשית על קלטים של מטריצות, נרשום מספר פונקציות אשר מטרתן לבדוק פלט מצורה זו.

יש להשלים את תבניות הקוד המצורף בקובץ ReversiPlay.java.

משימה ראשונה (0 נק')

השלימו את הפונקציה

public static void printMatrix(int[][] matrix)

המקבלת כקלט מערך דו מימדי matrix, ומדפיסה את תוכן המערך למסך בצורה הנוחה לאדם לקרוא אותה. שימו לב שנוחות היא דבר אינדיבידואלי, לכן איננו מבקשים צורה זו או אחרת של הדפסה. המטרה היא שהפלט של הפונקציה יספק צורה נוחה שבה תוכלו לראות ולהבחין בערכים הנמצאים בכל תא במטריצה.

משימה שנייה (5 נק')

השלימו את הפונקציה

public static boolean isEqual (int[][] matrix1, int[][] matrix2)

המקבלת כקלט שני מערכים דו מימדיים ומחזירה אמת אם ורק אם שני המערכים מקיימים:

1. גדלי המערכים זהים
2. תוכן כל תא i,j במערך הראשונה זהה לתוכן התא המקביל במערך השני

משימה שלישית (5 נק')

השלימו את הפונקציה

public static int[][] copyMatrix (int[][] matrix)

המקבלת כקלט מערך דו מימדי matrix ומחזירה העתק חדש של המערך המכיל את אותו המידע **בדיוק** (משימה 2 תחזיר עליהם אמת) אך הם נמצאים במקומות שונים בזכרון.

חלק ב' - מימוש המשחק (55 נק')

יש להשלים את תבניות הקוד המצורף בקובץ ReversiPlay.java.

במשחק הקלאסי, לוח המשחק הינו לוח במימדים 8*8. אנו נרחיב הגדרה זו ונאפשר לוח במימדים size*size, כאשר size הינו פרמטר שניתן על ידי המשתמש. על size להיות מספר **זוגי** בטווח 4-40 (כולל).

זכרו להשתמש בפונקציות שרשמתם בחלק הקודם בכדי לבדוק את נכונות הקוד שלכם!

משימה ראשונה – יצירת לוח משחק התחלתי (3 נק')

כתבו את הפונקציה:

public static int [][] createBoard (int size)

בעת קריאה לפונקציה זו, נוצר לוח משחק חדש, המתאים למצב ההתחלתי במשחק: עמדת הפתיחה היא 4 דיסקיות במרכז הלוח, 2 לכל שחקן, בהצלבה.
לוח התחלתי במשחק, עבור size=8, נראה כך:

	0	1	2	3	4	5	6	7
0								
1								
2								
3				2	1			
4				1	2			
5								
6								
7								

הפונקציה מחזירה את הלוח החדש. דיסקיות שחקן א' תיוצגנה על ידי המספר 1, דיסקיות שחקן ב' תיוצגנה על ידי המספר 2, והמשבצות הריקות בלוח תיוצגנה על ידי המספר 0.
בכדי לעזור להבין את נושא בדיקת הקוד, נסביר עבור משימה זו כיצד ניתן לבדוק את הקוד שרשמתם. התבוננו בקטע הקוד הבא:

```
int[][] expectedBoard = new int[][] {{0,0,0,0},{0,2,1,0},{0,1,2,0},{0,0,0,0}};
int[][] answerBoard = createBoard(4);
boolean check = isEqual(expectedBoard,answerBoard);
System.out.println("The two boards should be the same, the result we got is: " + check);
```

קטע הקוד מאתחל מטריצה בשם expectedBoard, בגודל 4 על 4 אשר מכילה את הערכים עבור לוח חוקי. לאחר מכן נקראת הפונקציה createBoard עם הערך 4 והתוצאה נשמרת במשתנה בשם answerBoard. לבסוף מופעלת הפונקציה שרשמנו בחלק א' (משימה שנייה) בכדי לבדוק האם המטריצות זהות אחת לשנייה.
באופן דומה תוכלו לבדוק את כל אחת מהמשימות הבאות.

משימה שניה – בדיקת חוקיות צעד במשחק (12 נק')

בכל שלב במשחק, מניח אחד השחקנים דיסקית שלו על הלוח, באופן הבא:
שחקן יכול להניח דיסקית שלו במשבצת מסוימת אם מתקיימים התנאים:

1. המשבצת ריקה.
2. קיים לפחות ישר אחד (אופקי, אנכי או אלכסוני) בין הדיסקית החדשה לדיסקית אחרת מאותו צבע, כך שבין דיסקיות אלו ישנו רצף בגודל של לפחות אחד של דיסקיות היריב. כל הדיסקיות הנ"ל צריכות להיות צמודות זו לזו על הישר הרלוונטי (כאמור, אופקי, אנכי או אלכסוני).

כתבו את הפונקציה:

```
public static boolean isLegal (int [][] board, int player, int row, int column)
```

המקבלת כקלט את הלוח, מספר השחקן והקואורדינטות המבוקשות, ומחזירה ערך אמת אם מותר לשחקן להניח דיסקית במשבצת המבוקשת, וערך שקר אחרת. קואורדינטת ה-row מייצגת את השורה בלוח וקואורדינטות ה-column מייצגת את העמודה.

לדוגמא, עבור הקלטים הבאים, כאשר board הינו לוח המשחק ההתחלתי (שתואר קודם), יוחזרו הפלטים הבאים:

```
isLegal(board, 1, 3, 2) → true
isLegal(board, 2, 3, 2) → false
isLegal(board, 1, 6, 4) → false
```

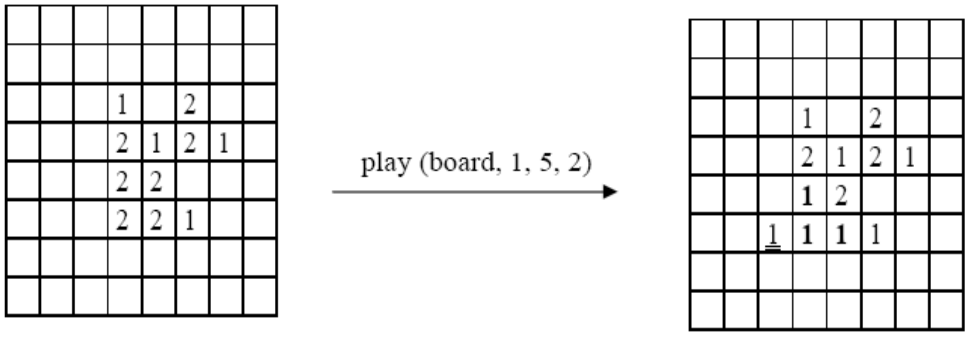
משימה שלישית – שינוי הלוח בעקבות מהלך במשחק (10 נק')

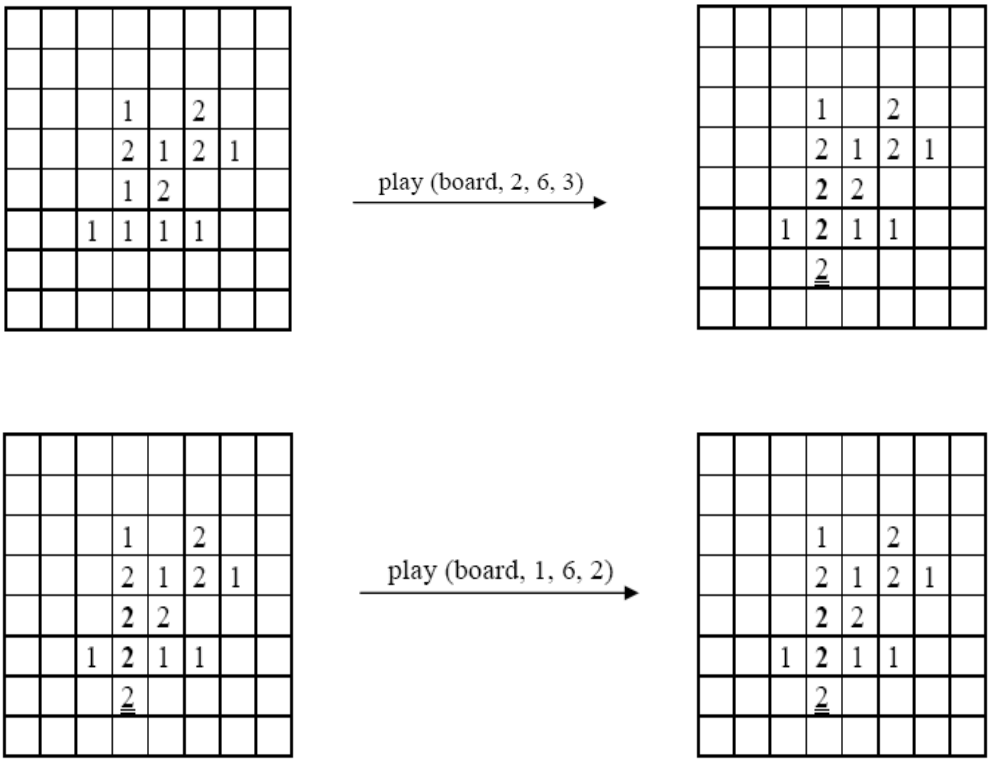
בעקבות הוספה חוקית של דיסקית ללוח המשחק (תחימת שורה, טור או אלכסון של עיגולים בצבע היריב, משני הקצוות, בעיגולים מהצבע שלך), הופכות דיסקיות היריב, המצויות על ישר כלשהו (אפקי אנכי או אלכסוני) בין הדיסקית החדשה לדיסקית אחרת של השחקן, לדיסקיות של השחקן.
כתבו את הפונקציה:

```
public static int [][] play (int [][] board, int player, int row, int column)
```

המקבלת כקלט את הלוח, מספר השחקן והקואורדינטות המבוקשות, ובאם הצעד המבוקש הינו חוקי, הפונקציה מחזירה את הלוח לאחר השינויים. במידה והצעד אינו חוקי, יוחזר הלוח ללא כל שינוי.

לדוגמא (זכרו – האינדקסים מתחילים תמיד מ-0 !!) (הדוגמה השלישית תוקנה) :





משימה רביעית – חישוב תועלת המהלך (10 נק')

כתבו את הפונקציה:

public static int benefit (int [][] board, int player, int row, int column)

המונה ומחזירה את תועלת המהלך, דהיינו את מספר דיסקיות היריב המשתנות בעקבות המהלך.
לדוגמא, עבור שלושת הלוחות והמהלכים שתוארו במשימה הקודמת, יוחזרו התועלות:
3
2
0

משימה חמישית – מציאת כל הצעדים האפשריים (5 נק')

כתבו את הפונקציה:

public static int [][] possibleMoves (int [][] board, int player)

המוצאת את כל המהלכים האפשריים (החוקיים) עבור שחקן ולוח נתון.
הפונקציה תחזיר את רשימת המהלכים הנ"ל במערך דו-מימדי, כך שהמימד הראשון של המערך מייצג את המהלכים החוקיים האפשריים, והמימד השני מייצג את הקואורדינטות של כל מהלך, כלומר: [i][0] מכיל את קואורדינטת השורה של מהלך

אפשרי מסויים, [1][i] מכיל את קואורדינטת העמודה של אותו מהלך, ומתקיים ש-i קטן ממש ממספר המהלכים האפשריים.
אין חשיבות לסדר המהלכים במערך המוחזר.
לדוגמא, עבור שלושת הלוחות ההתחלתיים והשחקנים המתאימים בדוגמת המשימה השלישית, יוחזרו המערכים הבאים:

2	3	5	6	6	1	1
2	2	2	3	4	6	4

4	6	6	6	6	6	4	3	2	1
2	2	1	3	4	6	7	7	4	3

2	3	7	7	7	1	1
2	2	2	3	4	6	4

<u>שימו לב:</u> גודלו של המימד הראשון במערך המוחזר הוא כמספר הצעדים האפשריים! לכן, **במידה ולא קיימים מהלכים אפשריים עבור השחקן המבוקש, יוחזר מערך עם מימד ראשון בגודל אפס ומימד שני בגודל 2.**

משימה שישית – האם קיימים צעדים אפשריים עבור שחקן? (5 נק')

כתבו את הפונקציה:

```
public static boolean hasMoves (int [][] board, int player)
```

המקבלת לוח משחק נוכחי ומספר השחקן ומחזירה ערך אמת בתנאי שקיים לפחות מהלך אחד אותו יכול השחקן לבצע.

משימה שביעית – מציאת המנצח במשחק (5 נק')

כתבו את הפונקציה:

```
public static int findTheWinner(int [][] board)
```

המקבלת לוח משחק נוכחי ומחזירה את מספר השחקן המוביל בשלב זה של המשחק: השחקן בעל מספר הדיסקיות הרב יותר על הלוח. במידה ויש שוויון בין מספר הדיסקיות, הפונקציה תחזיר 0.

משימה שמינית – הגדרת סוף המשחק (5 נק')

המשחק מסתיים כאשר מתקיימים אחד משלושת התנאים באים:

1. הלוח מלא
2. כל הדיסקיות על הלוח הן מצבע אחד (לאחד השחקנים אין דיסקיות על הלוח כלל)
3. לא קיימים צעדים אפשריים לביצוע עבור אף אחד מהשחקנים

כתבו את הפונקציה:

```
public static boolean gameOver (int [][] board)
```

המחזירה ערך אמת במידה וכל משבצות הלוח board מלאות, או במידה וכל הדיסקיות על הלוח הן של אותו שחקן, או במידה ששני השחקנים לא יכולים להתקדם במשחק: לשניהם אין צעדים אפשריים.

חלק ג' - כתיבת אסטרטגיות משחק (30 נק')

יש להשלים את תבניות הקוד המצורף בקובץ ReversiPlay.java.

במשימות הבאות תממשו שחקנים בעלי אסטרטגיות משחק שונות:

בכל אחת מארבע המשימות הבאות, מהלך חוקי מיוצג ע"י מערך בעל שני תאים, הראשון מייצג את השורה והשני מייצג את העמודה.

משימה תשיעית – אסטרטגיית משחק אקראית (6 נק')

כתבו את הפונקציה:

public static int[] randomPlayer (int [][] board, int player)

המקבלת לוח משחק ומספר שחקן, ומחזירה מהלך חוקי **אקראי** אפשרי עבור השחקן player מתוך כל המהלכים האפשריים עבור שחקן זה. במידה ולא קיים אף מהלך אפשרי, יוחזר null.

משימה עשירית – אסטרטגיית משחק חמדנית (8 נק')

כתבו את הפונקציה:

public static int[] greedyPlayer (int [][] board, int player)

המקבלת לוח משחק ומספר שחקן, ומחזירה מהלך חוקי בעל תועלת מקסימלית (במהלך הנוכחי) עבור השחקן. במידה וקיים יותר ממהלך אחד כזה (כלומר ישנם מספר מהלכים אפשריים שונים שתועלתם שווה ומקסימלית), יש לבחור מבין כל המהלכים האפשריים אחד, שיוחזר באופן אקראי. במידה ולא קיים אף מהלך אפשרי, יוחזר null.

משימה אחת עשרה – אסטרטגיית משחק הגנתית (8 נק')

כתבו את הפונקציה:

public static int[] defensivePlayer (int [][] board, int player)

המקבלת לוח משחק ומספר שחקן, ומחזירה מהלך חוקי בעל תועלת מקסימלית (בשני המהלכים הבאים) עבור השחקן: תועלת זו הינה ההפרש בין מספר הדיסקיות שיהפכו לצבע השחקן **במהלך הנוכחי** ומספר הדיסקיות שעשויות להפוך לצבע היריב בעקבות **מהלכו הבא** של היריב, לו ינקוט היריב באסטרטגיה **חמדנית**.

לדוגמא, אם בעקבות מהלך מסוים יהפכו 4 דיסקיות לצבע השחקן, אך בעקבות מהלך זה יכול היריב להפוך 6 דיסקיות לצבע שלו, אזי תועלת המהלך עבור השחקן הינה 2-. במידה וקיימת יותר מאפשרות אחת יש לבחור את האפשרות שתוחזר באופן אקראי. במידה ולא קיים אף מהלך אפשרי, יוחזר null.

משימה שתיים עשרה – אסטרטגיית משחק לפי מיקום (8 נק')

כתבו את הפונקציה:

public static int[] byLocationPlayer (int [][] board, int player)

המקבלת לוח משחק ומספר שחקן, ומחזירה מהלך חוקי לפי העדיפויות הבאות:

1. באם ניתן למקם דיסקית באחת מפינות הלוח, ייבחר מהלך זה (באם ישנן מספר אפשרויות כאלה, הבחירה ביניהן תהיה אקראית)
2. אחרת, ייבחר המהלך הקרוב ביותר למרכז הלוח (באם ישנן מספר אפשרויות בעלות עדיפות זהה, הבחירה ביניהן תהיה אקראית). **מרחק נמדד בצעדים שאינם אלכסוניים, ממרכז הלוח**

מרחקים ממרכז הלוח, לדוגמא:

7	6	5	4	4	5	6	7
6	5	4	3	3	4	5	6
5	4	3	2	2	3	4	5
4	3	2	1	1	2	3	4
4	3	2	1	1	2	3	4
5	4	3	2	2	3	4	5
6	5	4	3	3	4	5	6
7	6	5	4	4	5	6	7

<u>שימו לב:</u> במשימות 8-12, במידה וקיימת יותר מאפשרות אחת יש לבחור את האפשרות שתוחזר באופן אקראי. במידה ולא קיים אף מהלך אפשרי, יוחזר null.

משימה שלוש עשרה – משימת רשות: אסטרטגיית משחק טובה יותר (עד 40 נקודות)

תוכלו לנסות ולתכנן אסטרטגיות נוספות ולבחון אותן מול אסטרטגיות המשחק שמימשתם, ומול אסטרטגיות נוספות שמימשו חבריכם.

ניתן להגיש בנוסף למשימות הקודמות, פונקציה נוספת:

public static int[] myPlayer (int [][] board, int player)

בעלת אסטרטגיית משחק כלשהי לבחירתכם.

בהערכת התרגיל, נקיים "תחרות" של 1000 משחקים בין השחקן לבין ארבעת השחקנים (האסטרטגיות) שתוארו במשימות 9-12, על לוחות משחק בגדלים שונים כך שזמן כל 1000 המשחקים לא יעלה על **10 דקות**. שחקן שינצח בלפחות 80% מהמשחקים, יזכה את כותבו ב15 נקודות בונוס לציון התרגיל. מצורף לעבודה זו קובץ בשם **ReversiTournament** שמריץ משחקים אלו.

במידה ותבחרו להגיש שחקן זה, חובה לצרף להגשה קובץ טקסט נוסף (בפורמט pdf, בעברית או באנגלית) המתאר בקצרה

(עד עמוד אחד) את האסטרטגיה שמימשתם. תיאור האסטרטגיה יילקח בחשבון בעת ציון התרגיל.

בין כל האסטרטגיות שיוגשו ויקיימו את הדרישות הבאות נקיים טורניר נוסף:

1. יצליחו לעבור את הטורניר המתואר למעלה.
2. העבודה של הסטודנטים קיבלה את מלוא הנקודות בחלק ג' - משימות 9-12.
3. לעבודה סופק קובץ PDF המכיל מידע על האסטרטגיה.

הטורניר יתבצע באופן הבא: נחלק את העבודות ל"בתים" כך שבכל בית יהיו בין 4-8 עבודות (תלוי במספר העבודות שתוגשנה ושתעמודנה בדרישות) ונריץ 1000 משחקים בין כל הזוגות בתוך כל בית (כל נצחון מקנה שתי נקודות וכל תיקו מקנה נקודה).
המקום השני בכל בית יזכה ב10 נקודות (נוספות על 15 הנקודות שקיבל) ומקום ראשון יזכה ב15 נקודות.
בין המקומות הראשונים יתקיים טורניר נוקאאוט. טורניר זה יזכה את הזוכה במקום הראשון ב10 נקודות, מקום שני 6 נקודות ומקומות שלישי ורביעי ב3 נקודות.

בהצלחה!!!