# תרגיל מספר 2 (בול-פגיעה)

**תאריך הגשה**: 3.12.2017
**מתרגלת אחראית**: יעל מה-טוב

תרגיל זה מכסה את הנושאים הבסיסיים שנלמדו על מערכים.
בחלק הראשון תממשו אוסף שיטות המטפלות במערכים של מספרים שלמים (מטיפוס int)
בחלק השני תשתמשו בשיטות אלו על-מנת לממש אסטרטגיה למשחק בול-פגיעה (Mastermind).
שימו לב כי כל שיטה במשימות הנ"ל ניתנת למימוש ב- 5-15 שורות של קוד Java נאה.

## הוראות

**יש לקרוא את כל ההוראות ולוודא שאתם מבינים אותם לעומק!**

- במידה ואינכם בטוחים מהו הפירוש המדויק להוראה מסוימת או אם יש לכם שאלה אחרת הקשורה **לתוכן העבודה**, אנא היעזרו בפורום או בשעות הקבלה של האחראית על העבודה (פרוט שעות הקבלה מופיע באתר הקורס).
- בכל **בעיה אישית** הקשורה בעבודה (מילואים, אשפוז וכו') אנא צרו את הפניה המתאימה למייל של הקורס (ise.intro.181@gmail.com).
- **שאלות על העבודה יופנו אך ורק למתרגלת האחראית ולא לשאר סגל הקורס.**

### הוראות ודגשים לפתרון

מרבית תהליך הבדיקה מבוצע ע"י כלים אוטומטיים. כל חריגה מהנהלים להלן עלולה לגרום לתהליך הבדיקה להיכשל, וכתוצאה מכך לפגיעה משמעותית בציון התרגיל!

לכן, יש להקפיד למלא אחר כל הנהלים הבאים באופן מדויק:

- **ניתן להיעזר אך ורק בדברים שנלמדו במסגרת הקורס עד כה** (שיעורים, תרגולים, בחנים וכדומה).
- יש לדייק באופן כתיבת שמות השיטות וחתימותיהן כפי שתואר כאן. הבדיקה הינה אוטומטית ומסתמכת על כך שהשמות הינם כפי שתוארו.
- יש לוודא שהקוד שלכם מתנהג **בדיוק** כפי שמתואר בתרגיל. אין להוסיף שום דבר מעבר! הבדיקה הינה אוטומטית ומסתמכת על כך שהתוצאה היא לפי הדרישה.
- בכדי להבטיח שלא יהיו תקלות בשמות, סיפקנו לכם קובץ שלד Mastermind.java. חובה להשתמש בו.
- המשתנים `BASE`, `N` ו-`MAX_ROUND_NUMBER` הם משתנים מיוחדים אשר מוגדרים ב-scope של כל הפונקציות. <u>כל פונקציה בתוך המחלקה יכולה לגשת למשתנה כזה ולקרוא את הערך שלו</u>. **אין לשנות את ערכם של משתנים אלו מתוך הפונקציות בתרגיל**. ערכים של המשתנים האלו יתעדכנו בתחילת התרגיל מתוך החלון הגרפי.
- יש לוודא שהערכים של המשתנים `BASE`, `N` ו-`MAX_ROUND_NUMBER` זהים לאלו שבשלד התוכנית בעת ההגשה!
- בכל סעיף ניתן להיעזר בפונקציות שכתבתם בסעיפים הקודמים.
- ניתן להוסיף פונקציות עזר אך אין בכך צורך! אפשר לפתור את התרגיל עם קוד קריא וברור גם ללא פונקציות עזר. במידה ובחרתם לכתוב פונקציות נוספות אנא ודאו כי הן עומדות בדרישות והקורס ולא פוגעות בריצת הקוד. אם הטסטים שלכם לא עוברים אנא וודאו כי הפונקציות שכתבתם הן לא הסיבה. זו חובתכם לוודא שהפונקציות לא פוגעות בריצת הטסטים!
- לעבודה מצורף קובץ נוסף Frame.java האחראי על הגרפיקה בעבודה שלכם. **אין לשנות את הקובץ זה ואין לשנות את פונקציית ה-main אשר קוראת לפונקציה מתוך מחלקה זו**.

- הסבר מפורט על החלק הגרפי של העבודה וכיצד התוכנית שלכם אמורה לעבוד מובא בסוף המסמך. החלק הגרפי יכול לסייע לכם בבדיקת התוכנית שלכם אך אינו קובע כי התוכנית שלכם תקינה.
- פונקציית main מכילה מספר דוגמאות לבדיקות עבור הפונקציות שאתם נדרשים לכתוב (בדיקות אלו מסומנות כרגע בהערה). כדי לבדוק את הקוד בעזרתן יש להוציא אותן מהערה ולסמן בהערה רק את השורה `Frame.play();`.
- הבדיקות שסיפקנו הן רק דוגמה, עליכם לכתוב בדיקות נוספות כדי לבדוק שהתוכנית שלכם עובדת כנדרש. כתיבת הבדיקות הינה חלק חשוב בתכנות שמטרתו לבדוק שהתוכנית עובדת כמצופה.
- לחלק מהפונקציות הוספנו פקודת return על מנת לאפשר לקוד להתקמפל. עליכם לשנות את הערכים הללו בהתאם למימוש הפונקציות.
- לאורך כתיבת הפתרון חובה להקפיד על קוד קריא: הערות, הזחות ושמות בעלי משמעות. **פתרונות שלא יעמדו בסטנדרטים הללו יקבלו ניקוד חלקי בלבד.** אם אינכם בטוחים אם הקוד שלכם קריא, אנא פנו למתרגלת אחראית על התרגיל.

## הוראות הגשה

- יש לבצע ולהגיש את העבודה כיחידים, כלומר כל סטודנט יבצע ויגיש את עבודתו שלו.
- ההגשה מתבצעת במערכת ההגשה בלינק https://subsys.ise.bgu.ac.il/submission/login.aspx.
- יש להגיש את הקובץ Mastermind.java (בלבד) מכווץ כקובץ ZIP ששמו הוא תעודת הזהות של הסטודנט/ית, לדוגמה: 123456789.zip
- אין לשנות את שמות הקבצים, אין להגיש קבצים נוספים ואין ליצור תיקיות.
- ניתן להגיש את העבודה מספר פעמים עד לתאריך ההגשה, ההגשה האחרונה היא זו שתיבדק.
- העבודה תיבדק באופן אוטומטי לפי הפלט אשר התוכניות שלכם תדפיס למסך. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק. כל הדפסה אשר אינה עונה באופן מדויק לדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק, או כל תו אחר מיותר, חסר, או מופיע בסדר שונה מהנדרש), תגרור פגיעה משמעותית בציון.
- ניתן לפתור את העבודה במעבדות המחשבים של האוניברסיטה או במחשבים האישיים. **העבודות יבדקו במחשבי האוניברסיטה ולכן חובה לוודא שהעבודה עובדת כמצופה עליהם!** הציון של עבודות שלא יתקמפלו ו/או לא ירוצו על מחשבי האוניברסיטה יפגע.

# חלק 1

בתרגיל זה תייצגו N-יה של ספרות הניתנות בבסיס BASE באמצעות מערך של int באורך N.
גם עבור BASE>10 המספרים מיוצגים כמספרים שלמים! למשל הספרה הגדולה ביותר עבור BASE=11 תיוצג על ידי int שערכו הוא 10.

שימו לב! ערכי משתנים BASE, N ומשתנים נוספים שיוגדרו בתכנית (אשר יפורטו בהמשך), ישונו לצורך הבדיקה.

## משימה 1 (areArraysEqual)

כתבו את השיטה:

```
public static boolean areArraysEqual(int[] firstArr, int[] secArr)
```

אשר מחזירה true אם ורק אם שני מערכים firstArr ו-secArr מטיפוס int מכילים את אותם ערכים בסדר זהה. הקוד צריך להתייחס לאפשרות שמערך (אחד או יותר) יהיה null וכן לאפשרות ששני המערכים אינם בעלי אותו אורך – במקרה כזה יוחזר false.

דוגמאות הרצה:

| firstArr | secArr | areArraysEqual(firstArr, secArr) |
|---|---|---|
| {2,1,2,4} | {2,1,3,4} | **False** |
| {2,1,3,4} | {2,1,3,4} | **True** |
| {2,1,4,3} | {2,1,3,4} | **False** |
| {2,1,3} | {2,1,3,4} | **False** |

## משימה 2 (randomizeSequence)

כתבו את השיטה:

```
public static int[] randomizeSequence()
```

אשר מחזירה רצף אקראי של N **ספרות שונות** מן ה-BASE הנתון.
ניתן לייצר רצף אקראי ע"י הגרלת מספר בין 0 ל-BASE (כולל 0 ולא כולל BASE). כל בחירה נוספת של מספר תתבצע ע"י הגרלת מספר בין 0 ל-BASE מבין המספרים שטרם נבחרו.
**בהגדרה** מתקיים $N \leq BASE$ (מדוע?).

**שימו לב**: המשתנים N ו-BASE הם משתנים <u>סטטיים</u> שהוגדרו בתוך המחלקה וכל פונקציה יכולה להשתמש ולקרוא את הערך של משתנה כזה ע"י ציון שמו.

## משימה 3 (incrementOdometer)

נוכל להתייחס למערך בן m ספרות כ**מד-מרחק** אשר תוכנו מייצג את **מספר הקילומטרים אותו נסעה מכונית מרגע הייצור**.

כתבו את השיטה:

```
public static int[] incrementOdometer(int[] odometer)
```

אשר מעלה את ערך מד המרחק המיוצג
ע"י odometer ב-1, כאשר הערך מיוצג בבסיס BASE. את התוצאה יש להחזיר כמערך חדש.
הניחו כי הספרה המשמעותית ביותר (most significant digit) נמצאת באינדקס ה-0 במערך והספרה המשמעותית פחות (least significant digit) נמצאת באינדקס ה-(odometer.length-1) במערך.

שימו לב כי הספירה הינה מעגלית במובן זה שאחרי ערך מקסימאלי שניתן לייצג בבסיס BASE, חוזרים ל-0.

דוגמאות:

| BASE | odometer | incrementOdometer(odometer) |
|---|---|---|
| 10 | {0,0,9,9} | **{0,1,0,0}** |
| 5 | {2,0,1,4,4} | **{2,0,2,0,0}** |
| 10 | {9,9,8} | **{9,9,9}** |
| 10 | {9,9,9} | **{0,0,0}** |

אתם מתבקשים להשתמש בשלד הבא:

```
public static int[] incrementOdometer(int[] odometer) {
      int[] increasedOdometer = new int[odometer.length];

      // ENTER YOUR CODE HERE

      return increasedOdometer;
}
```

בכתיבת השיטה ניתן להניח כי odometer אינו null וכי הוא מכיל רק ספרות חוקיות (לפי בסיס BASE).
באחריות המשתמש בשיטה לוודא תנאים אלו.

**שימו לב**: משתנה BASE הוא משתנה סטטי שהוגדר בתוך המחלקה וכל פונקציה יכולה לקרוא את הערך של משתנה זה ע"י ציון שמו.

## משימה 4 (isRightfulGuess)

במשחק בול-פגיעה, אותו נממש בחלק 2 של התרגיל, "ניחוש חוקי" הינו מערך של integers באורך N המכיל **ספרות שונות** בבסיס BASE.

כתבו את השיטה:

```
public static boolean isRightfulGuess(int[] guessArray)
```

אשר מחזירה true אם מערך guessArray הינו ניחוש חוקי.

דוגמאות הרצה:

| BASE | N | guessArray | isRightfulGuess(guessArray) |
|---|---|---|---|
| 10 | 4 | {2,1,3,2} | **False** |
| | | {2,1,3,11} | **False** |
| | | {0,2,3,4,9} | **False** |
| | | {0,2,3,9} | **True** |

בכתיבת השיטה ניתן להניח כי guessArray אינו null. באחריות המשתמש בשיטה לוודא תנאי זה.

**שימו לב**: המשתנים N ו-BASE הם משתנים סטטיים שהוגדרו בתוך המחלקה וכל פונקציה יכולה להשתמש ולקרוא את הערך של משתנה כזה ע"י ציון שמו.

## משימה 5 (getNextRightfulGuess)

על-מנת ליעל את החיפוש אחר ניחוש חוקי, כתבו את השיטה:

```
public static int[] getNextRightfulGuess(int[] guessArray)
```

אשר מקבלת מערך guessArray של int בבסיס BASE ומחזירה ניחוש חוקי בבסיס BASE שערכו הינו הקטן ביותר מבין הניחושים החוקיים שערכם גדול מ-guessArray. במקרה בו guessArray מכיל את הניחוש החוקי הגדול ביותר יוחזר הניחוש החוקי הקטן ביותר בבסיס BASE עם N ספרות.

בכתיבת השיטה ניתן להניח כי guessArray אינו null וכי אורכו N. כמו כן, ניתן להניח כי guessArray מכיל ספרות חוקיות בבסיס BASE. באחריות המשתמש בשיטה לוודא תנאי זה.

**רמז**: השתמשו בפונקציה incrementOdometer שכתבתם בסעיף 3.

**שימו לב**: משתנה BASE הוא משתנה סטטי שהוגדר בתוך המחלקה וכל פונקציה יכולה לקרוא את הערך של משתנה זה ע"י ציון שמו.

## משימה 6 (judgeBetweenGuesses)

יהיו guess1 ו-guess2 ניחושים חוקיים במשחק בול פגיעה.
אם באינדקס כלשהו i מתקיים guess1[i]==guess2[i] נאמר כי i הינו פגיעה (hit).
אם באינדקסים i ו-j כך ש- i!=j מתקיים guess1[i]==guess2[j] נאמר שהזוג i,j הינו התאמה (fit).
כתבו את השיטה:

```
public static int[] judgeBetweenGuesses(int[] guess1, int[] guess2)
```

אשר מוצאת את מספר הפגיעות ומספר ההתאמות בין guess1 ו-guess2 ומחזירה אותם כמערך באורך 2 (איבר ראשון מספר פגיעות, ואיבר שני מספר התאמות).

לדוגמה:
עבור guess1 = {2,1,3,4} ,guess2 = {4,1,3,2}
נקבל 2 פגיעות (guess1[1]==guess2[1] ו-guess1[2]==guess2[2])
ו-2 התאמות (guess1[3]==guess2[0] ו-guess1[0]==guess2[3])
כך ש-judgeBetweenGuesses(guess1, guess2) יחזיר מערך שאיברו הראשון הוא 2 (מספר הפגיעות) ואיברו השני הוא 2 (מספר ההתאמות).

ניתן להניח כי guess1 ו-guess2 ניחושים חוקיים. באחריות המשתמש בשיטה לוודא תנאים אלו.

## חלק 2

במשחק בול-פגיעה שחקן א' בוחר קוד סודי שהינו N-יה בעלת ספרות שונות בבסיס BASE, שחקן ב' מנסה לנחש את הקוד הסודי. בכל תור שחקן ב' מנחש N-יה של ספרות שונות בבסיס BASE.
שחקן א' מודיע לשחקן ב' מה מספר הפגיעות ומה מספר ההתאמות. שחקן ב' מנצח במשחק אם הוא מנחש את הקוד הסודי (דהיינו, מספר הפגיעות בניחוש שלו הוא N) תוך מספר סיבובים שאינו עולה על מספר מסוים MAX_ROUND_NUMBER. אחרת א' מנצח.

תוכלו להניח כי התכנית מכילה את ההגדרה הבאה בתחילת המחלקה:

```
static int MAX_ROUND_NUMBER; // maximal number of rounds in a game
```

משתנה MAX_ROUND_NUMBER הוא משתנה <u>סטטי</u> שהוגדר בתוך המחלקה וכל פונקציה יכולה לקרוא את הערך של משתנה זה ע"י ציון שמו. ערך של משתנה זה יתעדכן מתוך חלון גרפי (הסבר בהמשך).

בתרגיל זה נייצג את ה"היסטוריה" של משחק כמערך תלת-ממדי אותו נגדיר ע"י:

```
int[][][] gameHistory = new int[MAX_ROUND_NUMBER][2][];
```

הניחוש של שחקן ב' בסיבוב round ($0 \le round < MAX\_ROUND\_NUMBER$)
מוחזק ב-gameHistory[round][0] וכזה הינו באורך N. התשובה הניתנת ע"י שחקן א' לניחוש זה (מס' הפגיעות ומס' ההתאמות) מוחזקת ב-gameHistory[round][1] והינה באורך 2.

דוגמה להיסטוריה עבור הקוד הסודי 4 2 6 5 כאשר N=4 ו-BASE=10:

| round | gameHistory[round][0] | gameHistory[round][1] |
|---|---|---|
| 0 | {0,1,2,3} | {1,0} |
| 1 | {0,4,5,6} | {0,3} |
| 2 | {4,1,6,5} | {0,3} |
| 3 | {5,6,2,4} | {4,0} |

ישנה אסטרטגיה פשוטה לשחקן המנחש במשחק:
בכל סיבוב שחקן ב' עובר על קבוצת הניחושים החוקיים האפשריים **ובוחר את הראשון אשר *מתיישב* עם כל שאר הניחושים והתשובות הקודמים לסיבוב זה** (הסיקו מזה מה הניחוש שיבחר בסיבוב הראשון).
ניחוש אפשרי נקרא *מתיישב* עם הניחושים והתשובות הקודמים אם הוא יכול לשמש בפוטנציאל כקוד הסודי המקורי. כלומר, אילו הוא היה הקוד הסודי המקורי אזי כל הניחושים הקודמים היו מקבלים בדיוק את אותן תשובות (מספר הפגיעות ומספר ההתאמות) שניתנו במציאות לקוד הסודי האמיתי.

שימו לב כי בקביעת התיישבות אין צורך להתייחס לקוד הסודי הניתן (או אפילו לדעת מהו) **אלא רק לניחוש אל מול ההיסטוריה של המשחק**.

דוגמה למציאת ניחוש מתיישב. נניח כי נתונה ההיסטוריה:

| round | gameHistory[round][0] | gameHistory[round][1] |
|---|---|---|
| 0 | {0,1,2,3} | {1,0} |
| 1 | {0,4,5,6} | {0,3} |
| 2 | {4,1,6,5} | {0,3} |

אזי:

| האם מתיישב עם הסטוריית המשחק? | ניחוש |
|---|---|
| אינו מתיישב עם round={1,2} | {4,1,6,7} |
| אינו מתיישב עם round={0,1,2} | {4,1,7,0} |
| אינו מתיישב עם round={2} | {4,5,6,3} |
| מתיישב עם כולם | {6,5,4,3} |
| מתיישב עם כולם | {5,6,2,4} |

נדגיש שוב כי אילו הקוד הסודי האמיתי היה 6543 או 5624 הניחושים שבהיסטוריה היו נותנים בדיוק את התשובות המצויות בהיסטוריה, ולכן שני ניחושים אלו הם פוטנציאליים בסיבוב הבא.

## משימה 7 (settleGuessInHistory)

כתבו את השיטה:

```
public static boolean settleGuessInHistory
      (int round, int[] currentGuess, int[][][] gameHistory)
```

אשר בודקת האם ניחוש בסבב ה-round מתיישב עם היסטוריה לכל i<round.

בכתיבת השיטה ניתן להניח כי:

1. לכל i<round הערכים ב- gameHistory[i][0] ו- gameHistory[i][1] אינם null.
2. currentGuess אינו null.

באחריות המשתמש בשיטה לוודא תנאים אלו.

## משימה 8 (update)

כתובו את השיטה update, אשר מוסיפה את נתוני סיבוב round (כאשר newGuess הניחוש ו-score היא תוצאת ה-hit\fit עבור הניחוש) להיסטוריה של המשחק.

```
public static void
      update(int[][][] gameHistory, int round, int[] newGuess, int[] score) {
      // adds a newGuess and answer (hits/fits) to the gameHistory at
      // "line" round. We assume that round < MAX_ROUND_NUMBER.
}
```

תוכלו להניח כי כל הפרמטרים תקינים. באחריות המשתמש בשיטה לוודא תנאים אלו.

## משימה 9 (play)

כעת עליכם לממש את האסטרטגיה למשחק. כתבו את השיטה:

```
public static int[][][] play(int[] secret)
```

אשר מחזירה את ההיסטוריה המיוחסת למשחק בול פגיעה המנוהל לפי האלגוריתם שהצגנו, ולסוד secret.
אתם מתבקשים להשתמש בשלד הבא:

```
public static int[][][] play(int[] secret) {

      boolean found = false;
      int[][][] gameHistory = new int[MAX_ROUND_NUMBER][2][];
      int[] currentGuessArray = new int[N];

      for (int i=0; i<N; i=i+1) {
            currentGuessArray[i] = 0;
      }

      int round = 0;
      while (round < MAX_ROUND_NUMBER && !found) {
            // ENTER YOUR CODE HERE
      }
      return gameHistory;
}
```

**רמז**: למימוש האלגוריתם השתמשו בשיטות שכתבתם בסעיפים 1-8.

משחק לדוגמה עבור MAX_ROUND_NUMBER=6 כאשר N=4 ו-BASE=10:
הקוד הסודי secret הוא:

| 0 | 1 | 7 | 8 |
|---|---|---|---|

ההיסטוריה לאחר הסיבוב הראשון:

| round | gameHistory[round][0] | gameHistory[round][1] |
|---|---|---|
| 0 | {0,1,2,3} | {2,0} |
| 1 | null | null |
| 2 | null | null |
| 3 | null | null |
| 4 | null | null |
| 5 | null | null |

ההיסטוריה לאחר הסיבוב הרביעי (והאחרון למשחק זה):

| round | gameHistory[round][0] | gameHistory[round][1] |
|---|---|---|
| 0 | {0,1,2,3} | {2,0} |
| 1 | {0,1,4,5} | {2,0} |
| 2 | {0,1,6,7} | {2,1} |
| 3 | {0,1,7,8} | {4,0} |
| 4 | null | null |
| 5 | null | null |

## משימה 10 (printGame)

משימתכם האחרונה היא מימוש השיטה:

```
public static void printGame(int[] secret, int[][][] gameHistory)
```

שתייצר את הפלט בפורמט הבא, כאשר התו '_' מסמל רווח (שימו לב – הקוד הסודי אקראי ולכן המספרים יהיו שונים בין ריצה לריצה אך יש להקפיד על הפורמט):

להצלחה (נניח כי MAX_ROUND_NUMBER=20)

```
The_secret_is_1_4_0_6_
The_guess_0_1_2_3_gives_(h/f):_0_2_
The_guess_1_0_4_5_gives_(h/f):_1_2_
The_guess_1_2_5_4_gives_(h/f):_1_1_
The_guess_1_4_0_6_gives_(h/f):_4_0_
You_guessed_the_secret_within_4_rounds
```

לכישלון (נניח כי MAX_ROUND_NUMBER=4)

```
The_secret_is_3_8_9_5_
The_guess_0_1_2_3_gives_(h/f):_0_1_
The_guess_1_4_5_6_gives_(h/f):_0_1_
The_guess_2_5_7_8_gives_(h/f):_0_2_
The_guess_3_6_8_7_gives_(h/f):_1_1_
You_failed_guessing_the_secret_within_MAX_ROUND_NUMBER=4_rounds
```

שימו לב כי gameHistory אינו בהכרח מלא.

## ממשק משתמש גרפי (GUI - Graphical User Interface)

ממשק משתמש גרפי (או בשמו המקובל GUI) הוא למעשה החלק של התוכנה אשר חשוף למשתמש ובעזרתו נוצר קשר בין התוכנית למשתמש. הממשק הגרפי עוזר למשתמש להפעיל את התוכנה בצורה קלה ויעילה יותר ומאפשר למשתמש לבצע פעולות (כגון שימוש בעכבר ולחיצה על כפתורים) אשר ללא השימוש בממשק לא היו מתאפשרות.

למידע נוסף על GUI:

http://en.wikipedia.org/wiki/Graphical_user_interface

לאחר שתכתבו (ותבדקו) את כל הפונקציות שלכם תוכלו לדמות משחק בול פגיעה, הריצו את הקובץ Mastermind.java בתוכו נמצאת פונקציית main הבאה:

```
public static void main(String[] args) {
      Frame.play();
}
```

בשלב הזה יפתח חלון נוסף עם הכותרת `MASTERMIND`. זהו הממשק הגרפי אשר נכתב בשבילכם.
כעת בחרו `MAX_ROUND_NUMBER`, `N` ו-`BASE` ולחצו על כפתור ה-start.
בעזרת הפונקציה `randomizeSequence` שכתבתם יבחר סוד (הסוד אינו חשוף עדין ובמקומו ישנן כוכביות).
לאחר מכן לחצו על כפתור ה-next guess אשר בוחר ניחוש (שוב בעזרת הפונקציות שכתבתם) עד אשר ניחשתם את הסוד או שהגעתם למספר הניחושים המקסימאלי. הודעות על הניחושים יופיעו בכל לחיצה.

### הערות חשובות

1. יתכן והממשק הגרפי יעבוד בצורה חלקה גם אם לא סיימתם את העבודה או אם לא פתרתם נכונה את כל המשימות.
2. ההדפסות לממשק הגרפי אינן תלויות במשימה 10. שימו לב כי הפלט שכתבתם במשימה 10 יופיע ב-Console וניתן להיעזר בממשק הגרפי כדי לבדוק את פלט זה.

# בהצלחה!