



## בס"ד שלמי תודה-

בסיעתא דשמיא,

עם השלמת בנית הפרויקט הננו להביע את תודתנו לכל המסייעים למלאכה:

**לריבון העולמים** - שהכל שלו והוא הנותן לאדם חכמה, בינה ודעת.

**לרב קוק שליט"א** - מנהל הסמינר, על השקעתו בפיתוח המגמה ובמשאבי המעבדה.

**לגב' אלישבע קצנלנבוגן תחי'** - מרכזת המגמה, שדאגה לכל מה שקשור למסלול ותרמה לנו רבות.

**לגב' רות רבין תחי'** - שהנחתה אותנו במהלך הפרויקט, על העזרה הרבה, יעוץ והכוונה לאורך כל הדרך.

לחברות הצוות שלנו – שליוו אותנו ואת הפרוייקט תקופה ארוכה. על עידוד וגם עזרה מקצועית. הקוד נכתב לא מעט בהשראתכן...

ומעל לכל לבני משפחותינו היקרים שרק בזכות התמיכה, ההתעניינות והסיוע הבלתי פוסקים יכולות אנו עתה לברך על המוגמר.

ולכל מי שהטה שכם וכרה אוזנו לקידום הפרויקט.

לכולם תודה!



## מבוא

בבואנו לבחור פרויקט התלבטנו רבות. רצינו ליצור תוכנה שתתן לנו התנסות עם קוד מעניין, מאתגר, לוגי. משהו שונה מהפרויקט הממוצע. בנוסף רצינו לרכוש באמצעות הפרויקט ידע מעשי החשוב לשוק העבודה. העלנו רעיונות שונים ומגוונים, פנינו לאנשים שיתכן ויצטרכו תכנה למקום עבודתם. שקלנו ברצינות כל רעיון, הקפנו אותו מכל היבטיו והתייעצנו עם אנשי מקצוע. לבסוף החלטנו לפתח משחק לוגי מעניין- הן מהסיבה שעולם המשחקים הוירטואלי מתפתח כל הזמן וכל משחק חדש יתקבל בברכה, והן מהסיבה שבכתיבת קוד זה היינו צריכות להכיר את טכניקת חדשות רבות, כגון טכניקת בניה של צורות תלת מימדיות במחשב, טכניקת הגרירה והשחרור ועוד טכניקות רבות שלא התנסנו בהן עד כה. המשחק מורכב מהמון שלבים כשבכל שלב נצרך המשתמש לפתור חידה, ברמת קושי עולה. החידה עצמה מורכבת מעשר צורות שונות, שחלקן כבר מובנות על הלוח ואין אפשרות להזיזם, והשאר ניתנות להזזה וסיבוב לכל הכיוונים, ואיתם על המשתמש להשלים לוח שלם. כל המשחק כולו, על כל שלביו וחלונותיו השונים, מעוצב בצורה יפה ומעניינת, וכולל יתרונות רבים על פני משחק הקופסה המקביל, כמו לדוגמה מתן זמן מוגבל לכל שלב, צבירת ניקוד ואפשרות השוואה לניקוד של השחקנים האחרים, מתן רמזים ועוד. דברים אלו מושכים את עינו של המשתמש ונותנים לו חשק להמשיך. במהלך פיתוח התוכנה הכרנו טכנולוגיות רבות, שיטות עבודה, ולאורך כל שלבי הפיתוח נפגשנו עם נושאים חדשים, טכניקות עבודה חדשות ורבות. שמנו גם דגש על כתיבת קוד על פי מתודולוגיות פיתוח חדישות ביותר, כדי שהקוד יהיה בהיר, יעיל וגמיש ככל הניתן.



## תוכן עניינים

1. הגדרת דרישות ותיאור כללי עמוד 4
- 1.1 תאור כללי עמוד 4
- 1.2 חומרת המערכת עמוד 5
- 1.3 תוכנת המערכת עמוד 5
- 1.4 תאור פונקציות המערכת עמוד 6
- 1.5 זרימת מידע עמוד 9
2. ממשקים חיצוניים עמוד 10
3. ממשק אדם מכונה עמוד 10
- 3.1 תרשים מסכים עמוד 10
- 3.2 תאור מסכים עמוד 11
4. מבנה נתונים וארגון קבצים עמוד 20
- 4.1 ארגון קבצים עמוד 20
- 4.2 מבנה נתונים עמוד 21
5. תכנון עמוד 23
- 5.1 מבנה כללי של הפרויקט עמוד 23
- 5.2 עקרונות תכנות עמוד 23
- 5.3 תאור אלגוריתמים מרכזיים- פרוט עבודה עמוד 24
- 5.4 בדיקות תקינות עמוד 56
6. מה הקנה הפרויקט עמוד 57
7. בבליאוגרפיה עמוד 58



## הגדרת דרישות ותאור כללי

### 1. הגדרת דרישות ותאור כללי

#### 1.1 תאור כללי

##### 1.1.1 מטרת המערכת

מטרת המערכת להציג משחק חשיבה לוגי מעניין. המשתמש יכול לבחור שלב, לפתור את החידה המסוימת שבחר, ולהמשיך לשלב הבא. בנוסף המשתמש יכול לעקוב אחר התקדמותו ע"י צבירת נקודות, השוואה לשחקנים אחרים, והמשך המשחק במועד מאוחר יותר כאשר השלבים שהוא כבר פתר נשארים שמורים לו כפתורים, וזאת על מנת לעורר מוטיבציה אצל ותחושת התקדמות תמידית אצל השחקן.

##### 1.1.2 היקף עבודה

מספר השעות המוקדש לפרויקט זה הוא 700 שעות.

##### 1.1.3 מבנה וארגון

השתמשנו בקבצי XML לצורך שמירת הנתונים על בניית המשחק, מכיוון שהם לא מכבידים על המערכת, ונוחים מאד לקריאה, כתיבה ועדכון. לצורך שמירת הנתונים האישיים של כל שחקן השתמשנו בטבלאות SQL, מכיוון שהם בטוחות יותר לשימוש, גם הן נוחות לקריאה, כתיבה ועדכון, והיתרון שלהם הוא שניתן להציג בעזרתן טבלאות בקלות (דבר שבא לידי שימוש במשחק).

##### 1.1.4 משימות המערכת

- פתיחת משחק.
- פתיחת שלב.
- ניהול משחק.
- מתן עזרה.
- סיום שלב.
- צפיה בטבלת שיאים.



בס"ד  
**1.2. חומרת המערכת:**

**1.2.1. כללי**

אין המערכת זקוקה למרכיבי חומרה מיוחדים.

**1.2.2. מרכיבי המערכת**

מחשב תואם Intel® Pentium® 4, מעבד Pentium III ומעלה

**1.3. תוכנת המערכת**

**1.3.1. מערכת הפעלה**

מערכת הפעלה Windows

**1.3.2. תוכנות**

Visual Studio 2017

**1.3.3. כלי התוכנה לפיתוח המערכת**

הפרוייקט נכתב בטכנולוגיית WPF בשפת C#.NET ומשתמשת ב-SQL לצורך שמירת נתונים.



בס"ד

## **1.4. תאור פונקציות המערכת**

### **1.4.1 פונקציות פתיחת משחק**

#### **1.4.1.1 פתיחת משחק חדש**

- הזנת שם שחקן

#### **1.4.1.2 פתיחת משחק שמור**

- שליפת הנתונים של השחקן

#### **1.4.1.3 פתיחת משחק**

- בחירת רמת קושי
- בניית לוח השלבים
- קליטת השלב הנבחר ע"י השחקן.

### **1.4.2 פונקציות פתיחת שלב**

- שליפת נתונים מה- xml
- בניית לוח החידה.
- בניית הצורות הנותרות ומיקומם בלוח המתאים.

### **1.4.3 ניהול משחק**

#### **1.4.3.1 תצוגת משחק**

#### **1.4.3.2 שינוי צורה**

- בעת לחיצה על צורה קליטת מספר הצורה והכיוון
- עפ"י מספר הצורה והכיוון בניית צורה תלת מיימדית שחופפת לצורה המקורית בדף חדש
- אפשרות מעבר בין הכיוונים של הצורה ושינוי הצורה המקורית



### **1.4.3.3 גרירת צורה**

- גרירה ללוח המשחק
- בדיקת הגרירה

### **1.4.3.4 מתן רמזים לפתרון**

### **1.4.3.5 ספירת זמן**

## **1.4.4 פונקציות לסיום שלב**

- בדיקה סיום שלב
- חישוב ניקוד
- עדכון הנתונים בהתאם
- פתיחת השלב הבא

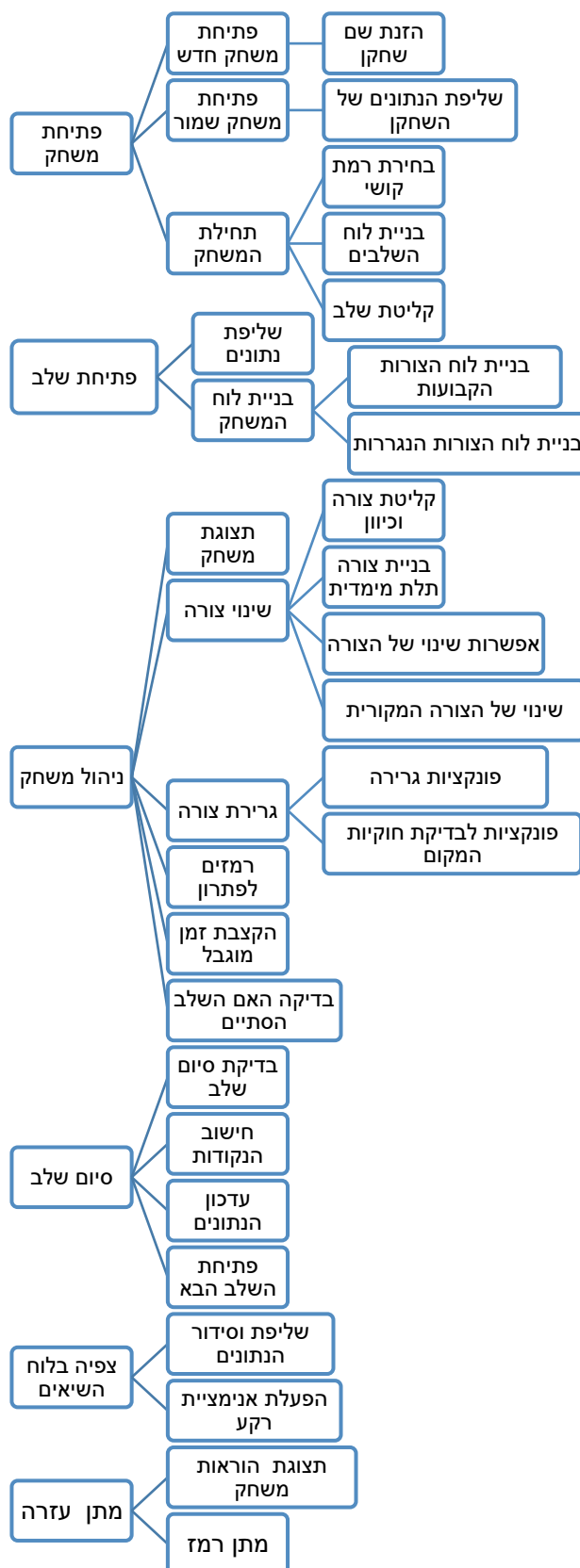
## **1.4.5 צפייה בלוח השיאים**

- שליפת נתוני השחקנים וסידור לפי גובה הניקוד
- הפעלת אנימצית רקע - צורות זזות המזכירות את הצורות במשחק

## **1.4.6 מתן עזרה**

- תצוגת הוראות משחק
- מתן רמז

### 1.5.1. תרשים תהליכים





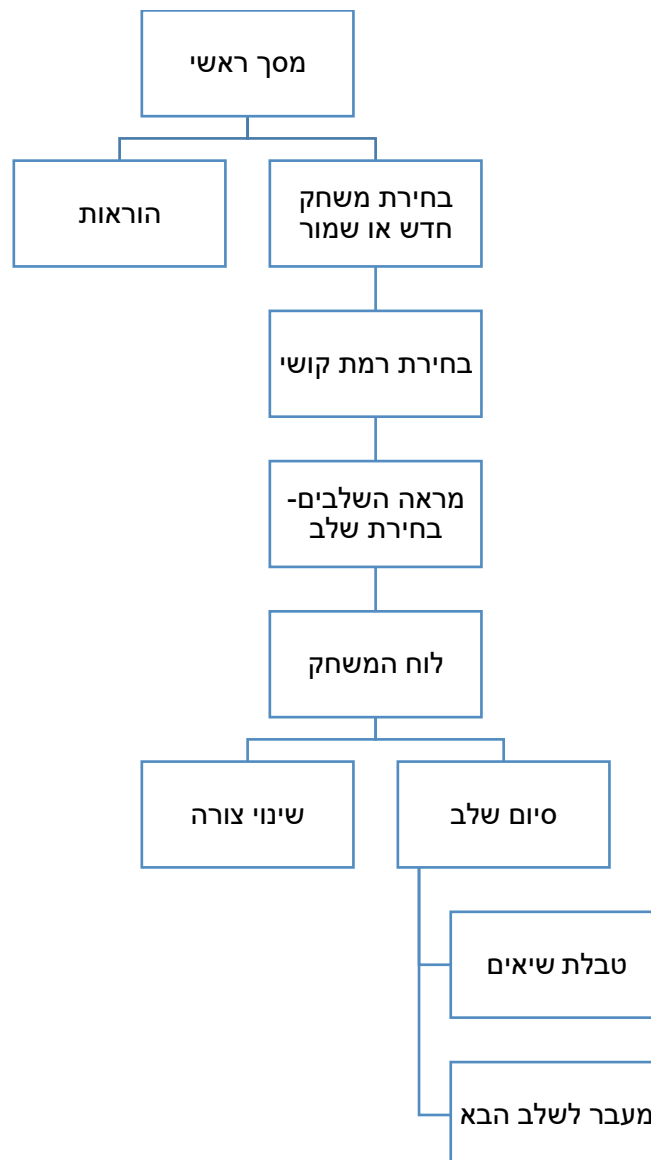
## ממשקים חיצוניים

### 2. ממשקים חיצוניים

לא רלוונטי

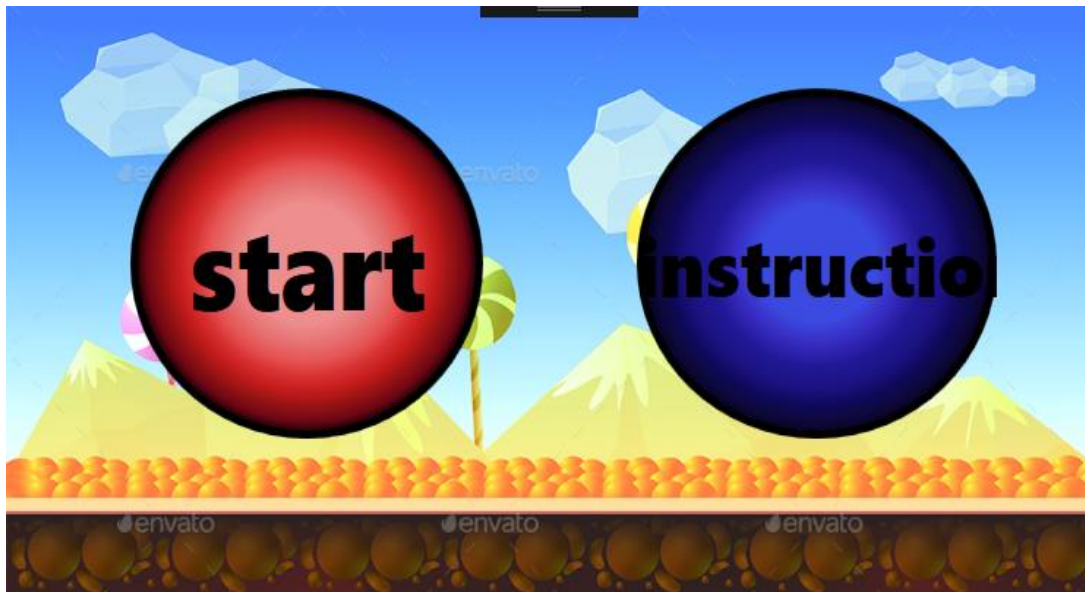
## ממשק אדם מכונה

### 3. ממשק אדם מכונה 3.1 תרשים מסכים-



### 3.2.1 תפריט פתיחה

מראה המסך:



**משימות המסך:** כניסה למשחק והוראות המשחק

**רשימת הפקדים ותפקידם:**

1. Instruction- הצגת הוראות לשחקן
2. Start -פתיחת מסך לבחירת משחק חדש או שמור

## 3.2.2 מסך הוראות

### מראה מסך:



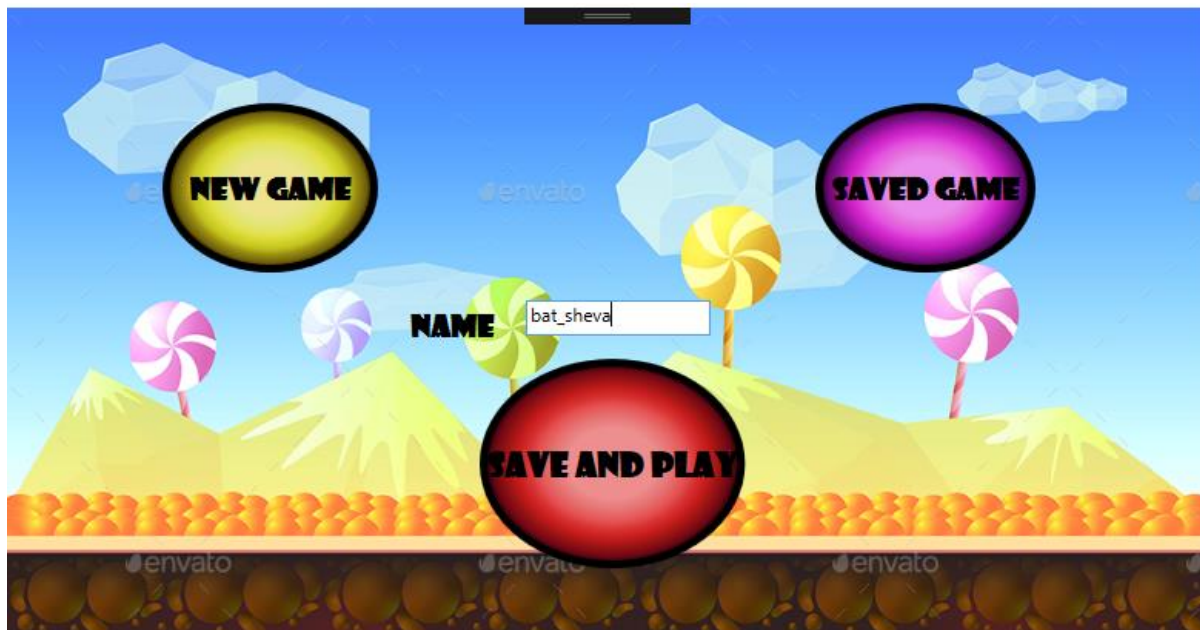
### משימות המסך: הוראות עבור המשחק

### רשימת פקדים ותפקידם:

Let's start - פתיחת מסך לבחירת רמת משחק

### 3.2.3 בחירת משחק חדש או שמור

מראה מסך:



#### רשימת הפקדים ותפקידם:

- NEW GAME - בחירת משחק חדש
- SAVED GAME - בחירת משחק שמור
- SAVE AND PLAY - שמירה ופתיחה של משחק חדש
- PLAY - שליפה ופתיחה של משחק שמור (לא מופיע במסך, נראה בדיוק כמו כפתור ה-SAVE AND PLAY, אך נראה רק כאשר לוחצים על כפתור ה-SAVED GAME)

### 3.2.4 בחירת רמת קושי

מראה מסך:



**משימות המסך:** בחירת רמת הקושי של המשחק

#### רשימת פקדים ותפקידים:

1. I'm ready - פתיחת מסך לבחירת השלב, לפי הרמה שנבחרה
2. List - רשימה של רמות קושי (1-12)



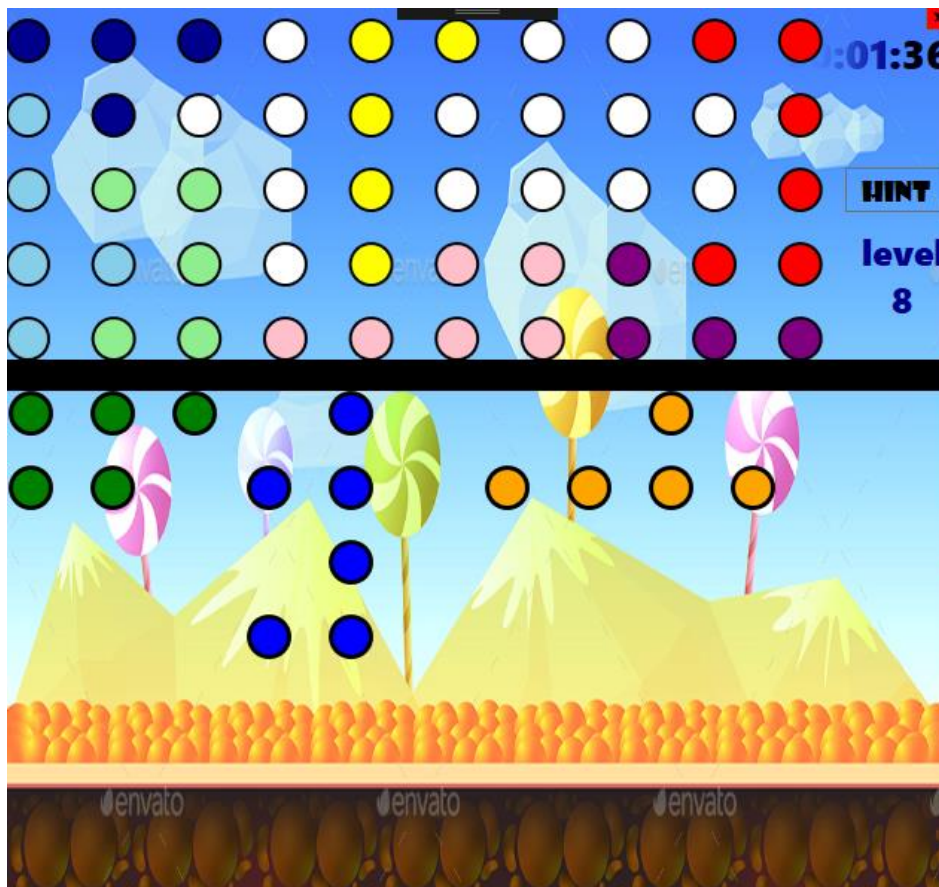
**משימות המסך:** בחירת שלב ותחילת המשחק

**רשימת הפקדים ותפקידם:**

1. לחצנים ימינה-שמאלה: מעבר בין רמות.
2. לחצנים עם מספרי השלבים: אפשרות של בחירת שלב.  
כאשר נמצאים באמצע שלב הלחצן שלו נצבע בצבע אדום, וכאשר מסיימים שלב בהצלחה הלחצן שלו נצבע בירוק.  
בעת פתיחת משחק שמור השלבים שכבר פותרו ע"י השחקן צבועים בירוק.

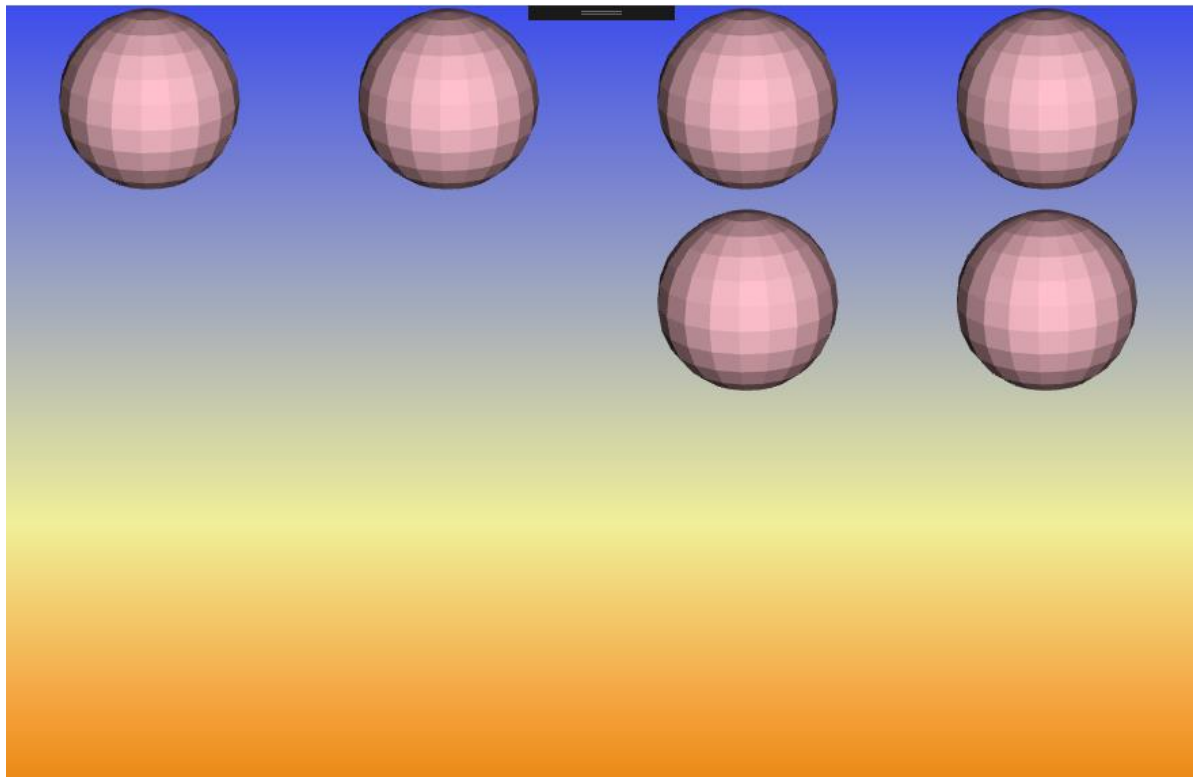


### 3.2.6 לוח המשחק מראה מסך:



**משימות המסך:** יש להשלים את הלוח העליון ע"י הצורות שנמצאות למטה  
**רשימת הפקדים ותפקידם:**

1. HINT-מתן רמז לפתרון
2. בלחיצה ימנית כפולה על צורה מהחלק התחתון של הלוח ניתן לשנות את כיוון הצורה
3. ע"י לחיצה רגילה על צורה מהחלק התחתון של הלוח ניתן לגרור אותה
4. כאשר הזמן נגמר לוח השלב נסגר
5. בלחיצה על "X" לוח השלב נסגר



**משימות המסך:** במסך זה ניתן לשנות את כיוון הצורה לכיוון הרצוי

**רשימת הפקדים ותפקידם:**

1. ע"י הלחצנים ימינה שמאלה שעל המקלדת ניתן לשנות את כיוון הצורה
2. ע"י לחיצה על enter או יציאה מהלוח הצורה המקורית משתנה



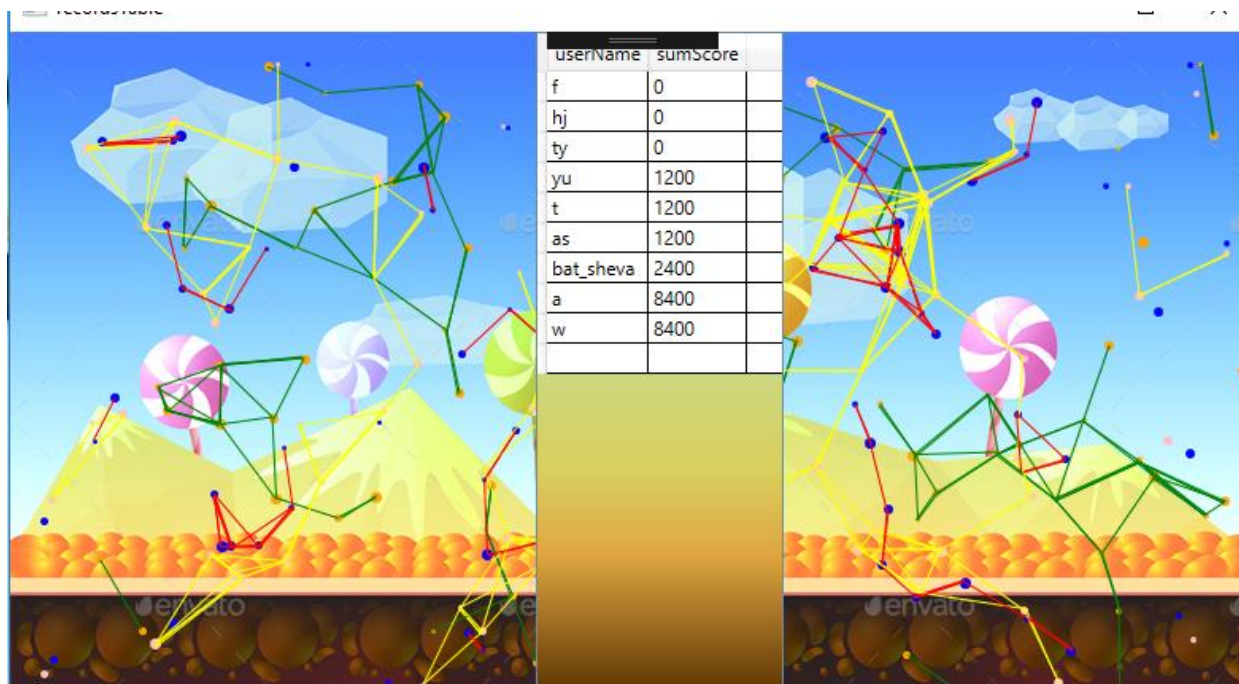
### 3.2.8 סיום המשחק

מראה מסך:



**משימות המסך:** סיום שלב, מראה את זמן פתירת השלב, מספר "הצעדים"-הזזת הצורות, עדכון נתוני הניקוד של השחקן ופתיחת השלב הבא  
**רשימת פקדים:**

Let's continue-מעבר לשלב הבא  
to the records table-מעבר ללוח השיאים



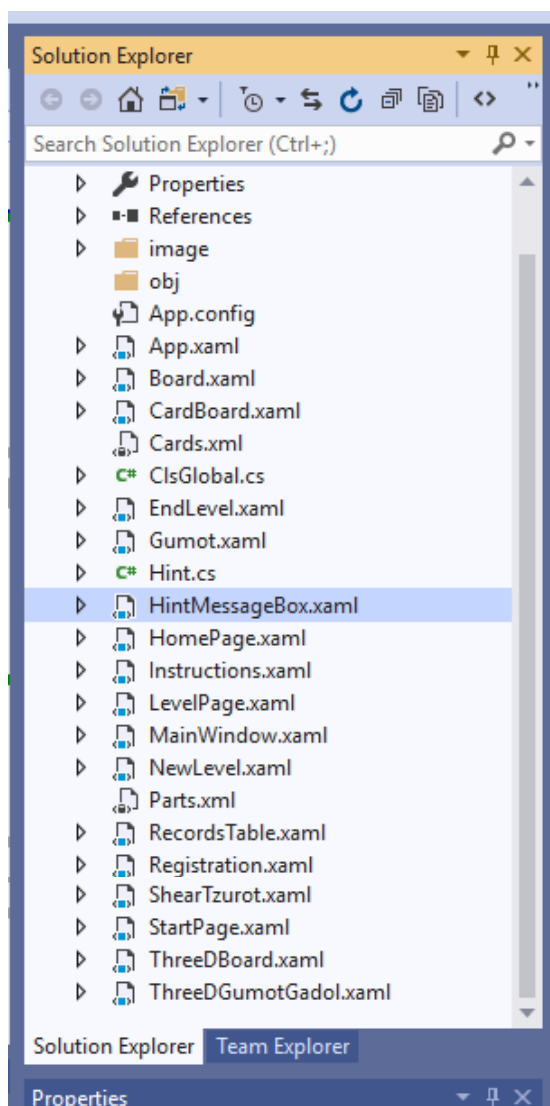
userName	sumScore
f	0
hj	0
ty	0
yu	1200
t	1200
as	1200
bat_sheva	2400
a	8400
w	8400

### משימות המסך:

1. הצגת טבלת הניקוד של השחקנים השונים, מסודרים לפי גובה הניקוד
2. הפעלת אנימצית רקע- צורת צבעוניות (המזכירות את הצורות שבמשחק) שזזות ומשתנות

## מבנה נתונים וארגון קבצים

### 4. מבנה נתונים וארגון קבצים



#### 4.1 ארגון קבצים

1. Image- שבה שמורות תמונות הרקע של הפרויקט
2. Board- הגדרת לוח המשחק העליון
3. CardBoard- הגדרת לוח המשחק התחתון
4. ClsGlobal- מחלקה שמגדירה משתנים סטטיים עבור כל התכנית
5. EndOfLevel- הגדרת מסך סיום שלב
6. Gumot- הגדרת עיגול (גומה) בודד



בס"ד

7. מחלקה שיוצרת מופע עבור צורה הנבדקת לרמז-Hint
8. מסך הרמז-HintMessageBox
9. מסך מראה השלבים-HomePage
10. מסך הוראות לשחקן-Instruction
11. מסך לבחירת רמת משחק-LevelPage
12. מסך המשחק-MainWindow
13. הגדרת הכפתור לבחירת השלב הרצוי-NewLevel
14. הגדרת המסך שמראה את טבלת השלבים-RecordsTable
15. מסך פתיחת משחק חדש או שמור-Registration
16. הגדרת הצורות שבלוח התחתון-ShearTzurot
17. מסך הפתיחה-StartPage
18. מסך הגדלת ושינוי הצורות-ThreeDBoard
19. הגדרת עיגול תלת מימדי-ThreeDGumotGadol

## 4.1.2 קבצי XML

- קובץ PARTS להגדרת עשרת הצורות וכיווני הסיבוב (הווריאציות השונות) של כל צורה
- קובץ CARDS להגדרת 120 שלבי המשחק, הכולל עבור כל שלב אילו צורות נמצאות בלוח העליון, באיזה כיוון באיזה מיקום.

## 4.2 מבנה נתונים

### 4.2.1 כללי

המשחק מורכב מ-2 לוחות- עליון ותחתון, כאשר על העליון יש צורות שלא ניתנות להזזה ומקומות ריקים, ומהלוח התחתון צריך לגרור את הצורות הנותרות עד להשלמת הלוח העליון

הלוחות מוגדרים באופן הבא:

### 4.2.2 הלוח העליון-

הגדרנו מחלקת user control בשם BOARD שהיא מגדירה את צורת הלוח. הלוח מורכב מ Grid המחולק למטריצה של 5 על 10 עיגולים (גומות). כל גומה נצבעת בצבע המתאים על פי המידע השמור בקובץ ה-XML, בסדר הבא:

עבור כל שלב שמורים מספרי הצורות שבלוח העליון, מספר הכיוון של כל צורה ונקודת ההתחלה של הצורה על פני הלוח (הנקודה שמורה כמספר דו ספרתי, כאשר מספר היחידות הוא מספר השורה, ומספר העשרות הוא מספר העמודה), לדוגמא:

קטע מתוך קובץ ה- CARDS

```
<card number="110">
  <part number="1" directionNum="6" place="50"></part>
  <part number="2" directionNum="7" place="03"></part>
</card>
```



בס"ד

שלב מספר 110, בו יש 2 צורות על פני הלוח העליון: צורה מספר 1 בכיוון מספר 6, מתחילה בנקודה (0,5), וצורה מספר 2 בכיוון מספר 7 המתחילה בנקודה (3,0).

מראה של צורה מסוימת בכיוון מסוים מוגדר בקובץ PARTS באופן הבא (נמשיך בדוגמא של שלב מספר 110):

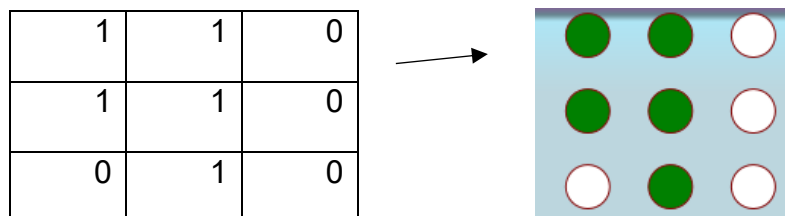
קטע מתוך קובץ ה-PARTS:

```
<part number="1" color="dark green" >
<direction directionNum="1" dr="8010111000"></direction>
<direction directionNum="2" dr="8100110100"></direction>
<direction directionNum="3" dr="8111010000"></direction>
<direction directionNum="4" dr="8001011001"></direction>
<direction directionNum="5" dr="8011111000"></direction>
<direction directionNum="6" dr="8110110010"></direction>
<direction directionNum="7" dr="8111110000"></direction>
<direction directionNum="8" dr="8100110110"></direction>
</part>
```

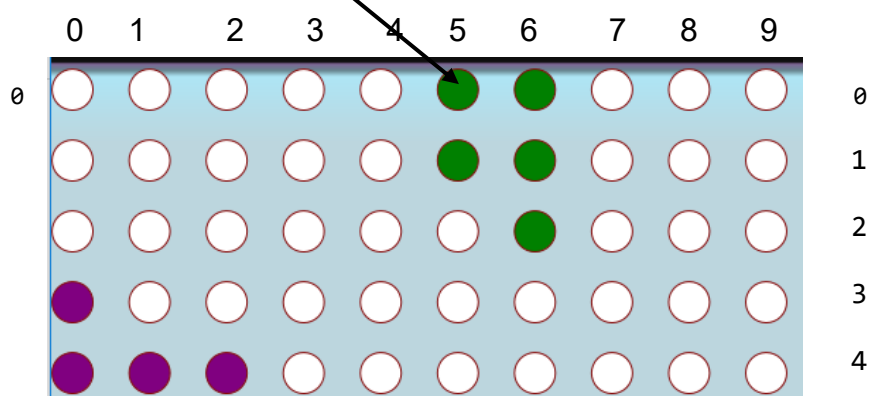
צורה מס' 1, צבעה הוא ירוק כהה, ויש לה 8 כיוונים, עבור הדוגמא שלנו ניקח את כיוון מספר 6:

```
<direction directionNum="6" dr="8110110010"></direction>
```

הצורה מוגדרת כמטריצה בגודל של 3 על 3, כאשר המספר 0 מבטא שהמקום ריק, והמספר 1 מבטא שבאותו המקום יש עיגול. (המספר 8 בא כדי לעזור למערכת להבחין בסוף המחרוזת) המחרוזת נקראת משמאל לימין, לא כולל את הספרה 8, מראה הצורה במקרה שלנו:



ונקודת ההתחלה של המטריצה הזו היא בשורה אפס עמודה חמש.



## 4.2.3 הלוח התחתון- בס"ד

במחלקה CIsGlobal שמור מערך בגודל 10 (בשביל עשרת הצורות) השומר מידע לגבי מספרי הצורות שמוקמות בלוח העליון (באופן הבא: המערך מלא באפסים, וכאשר צורה מסוימת ממוקמת בלוח ספרת האפס שהאינדקס שלה שווה למספר הצורה משתנה לספרה אחד), לאחר שנבנה הלוח העליון כל הצורות שלא מוקמו בו ממוקמות בלוח התחתון. צורת הבניה של הצורות זהה לצורת הבניה בלוח העליון, מלבד מספר הכיוון שנקבע באופן רנדומלי, ונקודת ההתחלה שנקבעת לפי התכנית.

## 4.2.4 טבלאות-

בחרנו לשמור את פרטי השחקנים בטבלאות SQL על מנת להראות שימוש בכלי נוסף בעולם התכנות.

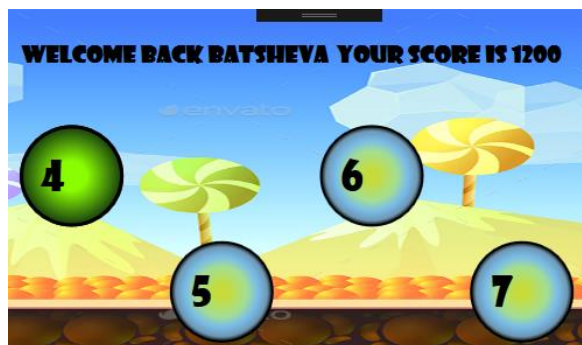
יצרנו טבלה בשם USERS המכילה 3 שדות :

**USER NAME** - שם השחקן,

**SUM SCORE** - סך הניקוד שצבר השחקן (ניתן לצפות בו במסך של טבלת השיאים, שם מופיע סך הניקוד של כל השחקנים ששמורים בטבלה, וכן בלוח בחירת השלב יש הודעה שמראה את גובה הניקוד),

**DONE LEVELS** - בשדה הזה שמורים מספרי השלבים שהשחקן עבר בהצלחה, ובכל בניה של לוח מראה השלבים, השלבים האלו צבועים בצבע ירוק,

לדוגמה-



השחקן BATSHEVA עבר את שלב 4 בהצלחה, והוא נצבע בצבע ירוק, גם בפתיחה מאוחרת יותר של המשחק לחצן מספר 4 יישאר ירוק.

בכל פתיחה של משחק חדש יש להזין שם משתמש, המערכת בודקת ששם כזה עדיין לא קיים בטבלה, ובמידה וכן היא מוציאה הודעה מתאימה. במידה ולא- השם נשמר בטבלה, ובשאר השדות נשמרים אפסים. לאחר כל שלב שעובר בהצלחה השדות מתעדכנים בהתאם.

כאשר פותחים משחק שמור בתחילה המשתמש מזין את השם, ולפי השם נשלפים מהטבלה הנתונים על השלבים שנעשו ועל גובה הניקוד שהשחקן צבר בעבר, ובזמן בניית הלוח ניתן לראות אותם.

## 5. תכנון

### 5.1. מבנה כללי של הפרויקט

הפרויקט נכתב בסביבת 201 2.net בשפת #c הנחשבת לסביבה חזקה, עדכנית ומבוקשת ובטכנולוגית wpf המתאימה לסוג אפליקציה זו. הפיתוח הקנה ניסיון רב ותרם לנו רבות.

### 5.2. עקרונות תכנות

**כללי:** הגישה התכנותית איתה עבדנו היא תכנות מונחה עצמים.

לגישה זו יתרונות רבים ביניהם: קוד קצר יותר, מתומצת, קוד הניתן לשימוש חוזר, איתור שגיאות לוגיות בצורה קלה יותר ועוד...

**כתיבת קוד תוכנה תקני ומובן:**

- קריאת שמות משמעותיים לפונקציות ולפרמטרים.
- חלוקת הקוד ליחידות קצרות ולפונקציות מחולקות (תפקיד ברור ומוגדר לכל פונקציה) על מנת להקל על שינויים עתידיים.
- ולידציות ובדיקות תקינות

## 5.3. תיאור אלגוריתמים מרכזיים- פרוט העבודה

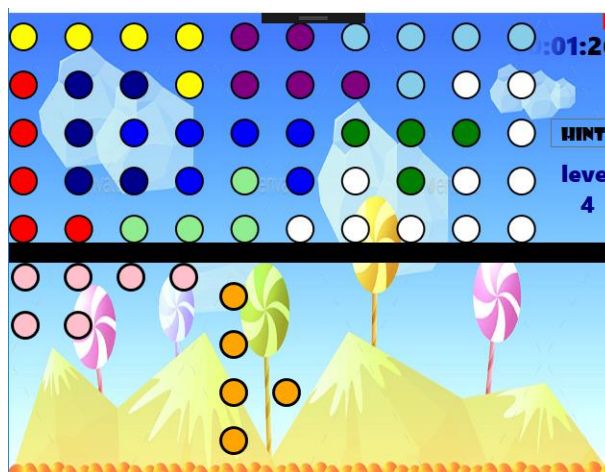
### 5.3.1 אלגוריתם מתן רמז

בעת לחיצה על כפתור ה-HINT שבלוח המשחק מופעלת פונקציית הרמז שמציאה לשחקן רמז בצורה הבאה- באופן רנדומלי נבחרת צורה מבין הצורות הנמצאות על הלוח התחתון, ונוצר בשבילה מופע של המחלקה HINT, שבו יש משתנים ששומרים את מראה הלוח העליון איך שהיה נראה בתחילת הבדיקה. כמו שפרטנו לעיל, לכל צורה יש 8 וריאציות שונות של כיוונים. הפונקציה עוברת עליהם לפי הסדר וסורקת כל פעם את הלוח העליון לראות האם יש מקום פנוי עבור הצורה הנוכחית עם בכיוון הנוכחי (אין אפשרות שלא יהיה לפחות כיוון אחד אפשרי, אחרת למשחק לא היה פתרון).

ברגע שהפונקציה מצאה מקום מתאים היא שומרת את מראה הלוח איך שהוא נראה אם המקום הזה תפוס, ושולחת לעצמה (באופן רקורסיבי) את הצורה הבאה שנמצאת על הלוח התחתון. גם עבור הצורה הבאה נוצר מופע של המחלקה HINT, ובו צורת הלוח בכניסה שווה לצורת הלוח שביציאה של הצורה הקודמת. גם עבור הצורה הבאה הפונקציה מחפשת כיוון אפשרי, כך שיהיה לו מקום על פני הלוח, אם היא מצאה היא שומרת את מראה הלוח ועוברת לצורה הבאה, וכך עד שיגמרו כל הצורות שבלוח התחתון. אם הפונקציה לא מוצאת שום כיוון אפשרי לצורה כלשהי היא חוזרת לצורה שהיתה קודם ומחפשת בשבילה כיוון אחר, ולאחר שהיא מוצאת היא מחפשת שוב עבור הצורות הבאות. אם הפונקציה הצליחה למצוא מקום עבור כל הצורות שבלוח התחתון היא מחזירה בתור רמז היכן ובאיזה כיוון כדאי למקם את הצורה שנבדקה ראשונה, שכאמור נבחרה באופן רנדומלי.

על מנת לפשט את ההסבר נסיף דוגמת הרצה:

נבחר את שלב מספר 4, יש בו 2 צורות על פני הלוח התחתון:



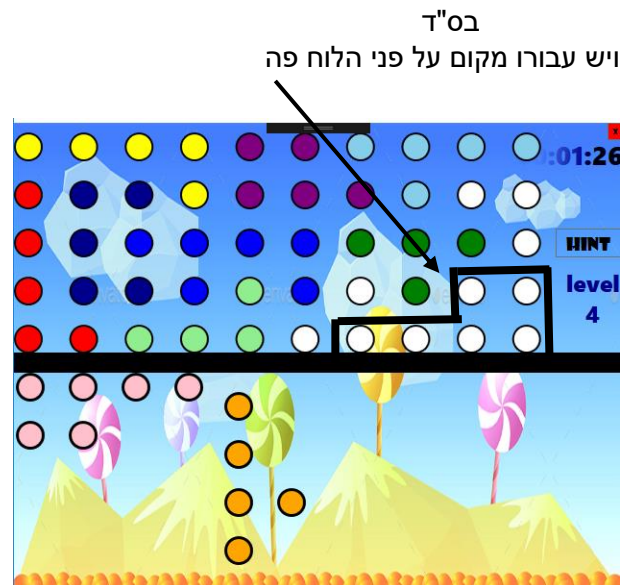
בלחיצה על HINT נבחרה באופן רנדומלי הצורה הורודה, היא שמורה בדפי ה-XML כצורה מספר 8, והכיוונים שלה הם כדלהלן:

```
<part number="8" color="pink" >
<direction directionNum="1" dr="8111111000000000"></direction>
<direction directionNum="2" dr="81100110001000100"></direction>
<direction directionNum="3" dr="80011111100000000"></direction>
<direction directionNum="4" dr="81000100011001100"></direction>
<direction directionNum="5" dr="80100111100000000"></direction>
<direction directionNum="6" dr="81000110010001000"></direction>
<direction directionNum="7" dr="81111001000000000"></direction>
<direction directionNum="8" dr="80100010011000100"></direction>
```

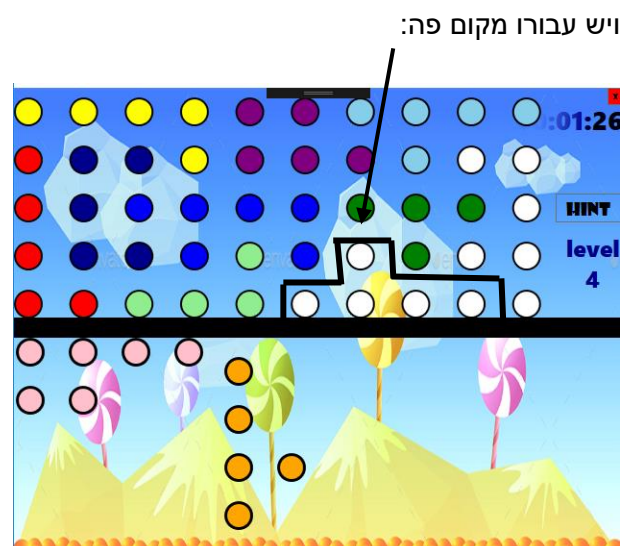
הפונקציה מחפשת כיוון שיהיה עבורו די מקום מתאים על פני הלוח, ונעצרת בכיוון מספר 3, שנראה כך,







הפונקציה שומרת את מראה הלוח כאשר המקום הנ"ל תפוס, ועוברת לצורה הבאה, הצורה הכתומה, ומנסה למצוא עבורה מקום פנוי על פני הלוח, אך כמובן שהיא לא מוצאת שום כיוון שיש עבורו מקום פנוי בלוח כזה, ולכן, לאחר שהיא בדקה את כל שמונת הכיוונים של הצורה הכתומה ולא מצאה שום כיוון מתאים היא חוזרת לצורה הורודה ומחפשת בשבילה כיוון מתאים אחר. לאחר חיפוש היא תעצר על כיוון מספר 5, שנראה כך,

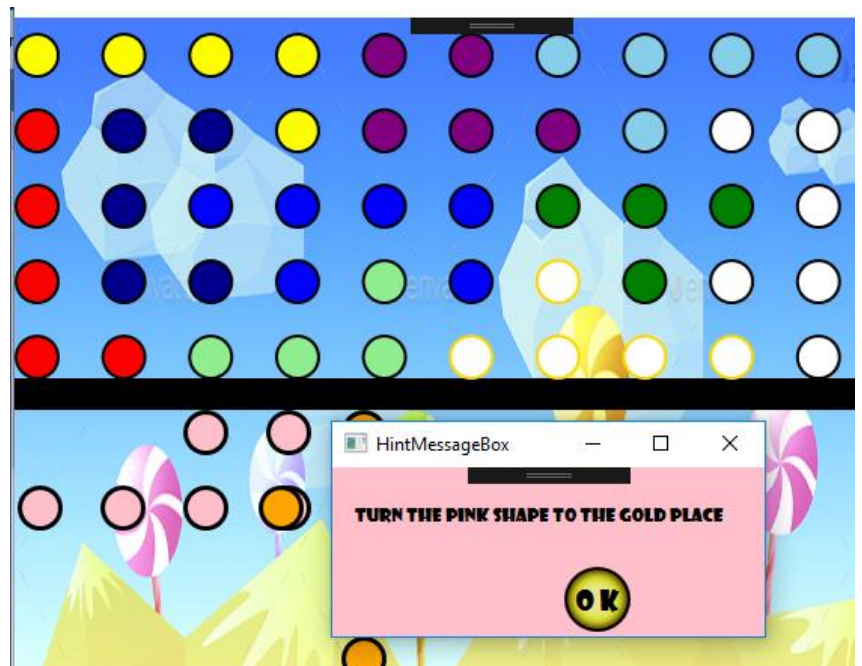


לאחר שמירת מראה הלוח הפונקציה תבדוק שוב את הכיוונים האפשריים של הצורה הכתומה, ותמצא לה מקום מתאים במקום של הגומות הלבנות שנשארו בכיוון מספר 7 של הצורה הכתומה, שנראה כך:



בס"ד  
`<direction directionNum="7"  
dr="81100010011000100"></direction>`

מכיוון שהפונקציה סיימה לבדוק את הצורות שבלוח התחתון ומצאה עבור כולם מקום, היא תחזיר בתור רמז שכדאי לשים את הצורה הורודה במקום הנ"ל, אופן הצגת הרמז הוא ע"י סימון המקום המתאים במסגרת בצבע זהב מסביב לגומות, והופעת חלון שבו רשום איזה צורה כדאי לשים במקום הזה, בדוגמה שלנו:



צבע הרקע של ההודעה שווה לצבע הצורה שעליה רומזים. לאחר 3 שניות צבע המסגרת של הגומות חוזר להיות שחור.

להלן הפונקציות:  
הפונקציה HINT שבוחרת צורה באופן רנדומלי, מאתחלת את המשתנים המתאימים, ומפעילה את הפונקציה הבאה (חילקנו ל-2 פונקציות כדי להקל על כתיבת הקוד)

```
public void Hint()
{
    int[] arrToHint = new int[10];
    int part;
    for (int i = 0; i < 10; i++)//הרמז, ולא בפונקציה שמחשבת את הרמז
    {
        if (ClsGlobal.shapesUse[i] != 0)// כדי שלא
        {
            נעשה רמז עליהם, לצורך זה משתמשים במערך הגלובלי
            arrToHint[i] = 1;
        }

        Random random = new Random();
        int randomS = random.Next(0, 10);

        while (arrToHint[randomS] != 0)//הבדיקה את נבצע עליה נבצע את הבדיקה
    }
```



בס"ד

```
{
    randomS = random.Next(0, 10);
}
part = randomS + 1;
ClsGlobal.previosShapeToHint[randomS] = new iq_project.Hint();// שבו , איפוס מערך העזר, שבו
שמורות הצורות שהיו קודם ברמז
ClsGlobal.previosShapeToHint[randomS].PreviosPart = 0;
ClsGlobal.previosShapeToHint[randomS].IsFirstShape = true;

for (int i = 0; i < 5; i++)
    for (int j = 0; j < 10; j++)
    {
        if (GOfBoard[i, j].ellipse.Fill != Brushes.White)
        {
            ClsGlobal.previosShapeToHint[randomS].BoardOnEnter[i, j] = true;
            ClsGlobal.previosShapeToHint[randomS].BoardOnExit[i, j] = true;
        }
        else
        {
            ClsGlobal.previosShapeToHint[randomS].BoardOnEnter[i, j] = false;// מין
            העתק של הלוח איך שהוא נראה כשהפונקציה מתחילה לבדוק צורה כלשהיא
            ClsGlobal.previosShapeToHint[randomS].BoardOnExit[i, j] =
            false;//בשביל הצורה הראשונה אנחנו מסמנים ככה גם בביל הלוח של היציאה
        }
    }
    GoodHint(1, arrToHint, part, true);//ועוד אחד כי מערך
}
```

הפונקציה GOOD HINT , זו פונקציה רקורסיבית, שמקבלת מס' צורה, מס' כיוון, מערך עזר שמסמן אילו צורות כבר בדקנו, ומשתנה בוליאני שמסמן אם זו הצורה הראשונה. אופן החישוב של הפונקציה הוסבר בפירוט לעיל.

```
public void GoodHint(
    int dir, int[] arrToHint, int part, bool isFirstShape)// משתנה ראשון- בא לסמן את כיוון
    הצורה, מתחילים מהכיוון הראשון, וממשיכים עד שמוצאים משהו מתאים
    (המשתנה השני בא במקום המשתנה הגלובלי (נראה לי עדיף))
    // המשתנה השלישי בא לסמן אם עובדים שוב על אותה צורה, א שכבר על צורה אחרת (אם הוא שווה אפס אז על
    צורה אחרת, אחרת עליו)
    bool IsShape = false;//בא לבדוק האם סיימנו לחפש את כל הצורות שהיו למטה, או שעוד לא
    bool flagForThisPlace = false;
    ShearTzurot tzurot = null;

    for (int i = 0; i < 10 && !IsShape; i++)
    {
        //בדיקה האם לא סיימנו לעבור על כל הצורות שיש, אם סיימנו והכל טוב, מחזירים את הצורה שהיתה ראשונה
        if (arrToHint[i] == 0)
            IsShape = true;
    }
    if (!IsShape)// זה אומר שסיימנו לעבור על כל המערך! ברכותי! צריך להחזיר את הצורה הנכונה, עם הכיוון
    שעליו נעצרנו, ועם המקום ששם נעצרנו
    {
        FindMatrixSize(firstPartForHint);
        long DRForHint = FindDr(firstPartForHint, dirForHint);
        for (int i = xForHint; i < xForHint + matrixSize; i++)
            for (int j = yForHint; j < yForHint + matrixSize; j++)
            {
                long y = (DRForHint / slicer) % 10;
                if (y == 1)

```



בס"ד

```
GOfBoard[i, j].ellipse.Stroke = Brushes.Gold; // אז צובעים את מסגרת הגומות שמתאימות לרמז בצבע זהב (לשלוש שניות)
    slicer /= 10;
}

HintMessageBox HMB = new HintMessageBox(Col, "Turn the ", " shape to the gold place");
HMB.ShowDialog();
for (int i = 0; i < 10; i++)
    ClsGlobal.previosShapeToHint[i] = null; // איפוס המערך ששומר בשביל כל צורה את הפרטים על הצורה הקודמת ברקורסיה
return;
}

FindMatrixSize(part); // מוצאים את סדר הגודל של הצורה (3*3 או 4*4 וזה נשמר במשתנה גלובלי
long DR = FindDr(part, dir); // מוצאים את הקוד של הצורה+הכיוון שלה
long slicer1 = slicer;

for (int i = 0; i < 5; i++) // סריקה של הלוח העליון, על מנת למצוא מקום מתאים לצורה
{
    for (int j = 0; j < 10; j++)
    {
        for (int k = i; k < (i + matrixSize); k++)
        {
            for (int l = j; l < (j + matrixSize); l++)
            {
                flagForThisPlace = true;
                long y = (DR / slicer) % 10;
                if (y == 1) // משווים את הצורה שמלטה ואת המקום על פני הלוח למעלה
                {
                    if (k >= 5 || l >= 10) // אם יצאנו מגבול המטריצה, יוצאים מהלולאה ישר, כי
                    {
                        flagForThisPlace = false;
                        k = i + matrixSize;
                        l = j + matrixSize;
                    }
                    else
                    {
                        if (ClsGlobal.previosShapeToHint[part - 1].BoardOnEnter[k, l] == true)
                        {
                            flagForThisPlace = false; // אם המקום לא מתאים מהלולאה,
                            k = i + matrixSize;
                            l = j + matrixSize;
                        }
                    }
                    if (flagForThisPlace)
                        slicer /= 10;
                    else
                        slicer = slicer1;
                }
            }
        }
    }
    slicer = slicer1;

    if (flagForThisPlace) // אם המקום הזה טוב, כלומר הוא מתאים לצורה שאותה בדקנו
    {
        arrToHint[part - 1] = 1; // כדי
        // מסמנים במערך שאנו סיימנו כעת לעבוד על הצורה הזו (כדי לא לחפש אותה שוב במקרה שלא נמצא לה מקום ונצטרך לחפש צורה אחרת)
        if (isFirstShape)
        {
            אין טעם להמשיך לבדוק
            וממשיכים לחפש מהמקום הבא
        }
    }
}
```



בס"ד

```
firstPartForHint = part;
dirForHint = dir;
xForHint = i;
yForHint = j;
}

for (int k = i; k < (i + matrixSize); k++)
{
    for (int l = j; l < (j + matrixSize); l++)
    {
        long y = (DR / slicer1) % 10;
        if (y == 1 && k < 5 && l < 10)
            ClsGlobal.previosShapeToHint[part - 1].BoardOnExit[k, l] =
true; // מסמנים את הלוח המתאים כך שישמר לנו המקום של הצורה+הכיוון שבדקים עכשיו/
        slicer1 /= 10;
    }
}
i = 5;
j = 10;
int nextPart = 0;
for (int p = 0; p < 10 && nextPart == 0; p++)
{
    if (arrToHint[p] == 0)
        nextPart = p;
}
ClsGlobal.previosShapeToHint[nextPart] = new Hint();
ClsGlobal.previosShapeToHint[nextPart].PreviosPart = part; // שמירת

הצורה הקודמת
ClsGlobal.previosShapeToHint[nextPart].BoardOnEnter =
ClsGlobal.previosShapeToHint[part - 1].BoardOnExit; // הלוח שביציאה של הצורה הראשונה זה הלוח בכניסה של
הצורה הבאה
ClsGlobal.previosShapeToHint[nextPart].BoardOnExit =
ClsGlobal.previosShapeToHint[part - 1].BoardOnExit; // הלוח שביציאה של הצורה השניה צריך בהתחלה להיות
שווה ללוח שבכניסה, ואח"כ עוד מוסיפים לו את הבדיקות העכשוויות
ClsGlobal.previosShapeToHint[part - 1].LastDir = dir; // שמירת הכיוון האחרון
GoodHint(1, arrToHint, nextPart + 1, false);
}
}
if (!flagForThisPlace)
{
    if (dir < 8) // יש סה"כ 8 כיוונים, אז אם עברנו אותם ולא מצאנו כלום צריך לבדוק מה הצורה שהיתה
קודם, ולחפש בשבילה כיוון אחר
    {
        ClsGlobal.previosShapeToHint[part - 1].LastDir = dir + 1; // שמירת הכיוון האחרון
        GoodHint(dir + 1, arrToHint, part, isFirstShape);
    }
    else
    {
        ClsGlobal.previosShapeToHint[part - 1].PreviosPart =
1].PreviosPart - 1].BoardOnExit =
ClsGlobal.previosShapeToHint[part - 1].PreviosPart -
1].BoardOnEnter; // משנים את לוח היציאה של הצורה כך שיהיה שווה ללוח הכניסה, כדי שאפשר יהיה להתייחס לצורה כאילו
לא נבדק הכיוון הקודם
        GoodHint(ClsGlobal.previosShapeToHint[part - 1].PreviosPart - 1].LastDir + 1, arrToHint, ClsGlobal.previosShapeToHint[part -
1].PreviosPart, ClsGlobal.previosShapeToHint[part -
1].PreviosPart - 1].IsFirstShape); // זה מקרה שצריך לחזור אחורה ברקורסיה, ולבדוק שוב צורה קודמת
    }
}
}
```



3 שניות לאחר מתן הרמז מופעלת הפונקציה הבאה, ומחזירה את מסגרות הגומות לצבע שחור:

```
public void backToBlack()
{
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 10; j++)
            if (GOfBoard[i, j].ellipse.Stroke == Brushes.Gold)
                GOfBoard[i, j].ellipse.Stroke = Brushes.Black;
}
```

### 5.3.2 פתיחת משחק

בעת פתיחת משחק יש אפשרות לבחירה- משחק חדש או משחק שמור. בשני המקרים אנו נעזרים בפונקציה SELECT המוגדרת במחלקה :REGISTRATION

```
public DataTable Select(string selectSQL) // חיבור למסד הנתונים
{
    DataTable dataTable = new DataTable("dataBase"); // יצירת טבלה
    SqlConnection sqlConnection = new SqlConnection(@"Data Source
=(LocalDB)\MSSQLLocalDB; AttachDbFilename = C:\Users\Avraham\AppData\Local\Microsoft\Microsoft
SQL Server Local DB\Instances\MSSQLLocalDB\New Database.mdf "); // חיבור למסד הנתונים
    sqlConnection.Open(); // פתיחת מסד הנתונים
    SqlCommand sqlCommand = sqlConnection.CreateCommand(); // הפקודה
    sqlCommand.CommandText = selectSQL; // הפיכה לטקסט
    SqlDataAdapter sqlDataAdapter = new SqlDataAdapter(sqlCommand);
    sqlDataAdapter.Fill(dataTable); // החזרת הטבלה עם
    הפקודה שנשלחה
    return dataTable;
}
```

כאשר למשתמש חדש משתמשים בפונקציה SavedAndPlayButton\_Click שבה מוסיפים בדיקה ששם המשתמש שהוא רשם לא קיים עדיין במערכת, ואם הוא כן אז יוצאת הודעה מתאימה. וכמו כן נוצר משתנה חדש, DONELEVEL, מסוג מחרוזת שבו ישמרו מספרי השלבים שהשחקן יעבור בהצלחה. בהתחלה הוא שמור כמחרוזת של אפסים כמספר השלבים הקיימים במשחק, וכל פעם שהמשתמש עובר שלב בהצלחה האפס שבמחרוזת שהאינדקס שלו שווה למספר השלב הופך להיות ל-1.

```
private void SavedAndPlayButton_Click(object sender, RoutedEventArgs e)
{
    string doneLevel = "0"; // בשביל שמירת המשחק
    for(int i=1; i<120; i++)
        doneLevel += "0"; //1-ל הופך ל-0
    ClsGlobal.DoneLevels = doneLevel;
    try
    {

```



בס"ד

```
Select("INSERT INTO [dbo].[Users] VALUES ('" +
newNameTextBox.Text + "','0','"+doneLevel+"')"); //עדכון משתנים של שחקן חדש
ClsGlobal.userName = newNameTextBox.Text;
ClsGlobal.userNameString = "hello " + newNameTextBox.Text;
LevelPage 1 = new LevelPage();
Close();
l.ShowDialog();
}
catch (Exception ex)
{
    MessageBox.Show("name like this exists in sistem");
}
}
```

PLAYBUTTONCLICK

עבור שחקן קיים- חיפוש האם במסד הנתונים קיים שחקן בשם הזה, אם כן הודעה מתאימה ושלילת הנתונים שלו (מה הניקוד שלו ואילו שלבים הוא סיים בהצלחה בעבר), ואם לא אז הודעה שמבקשת לבדוק את השם.

```
private void PlayButton_Click(object sender, RoutedEventArgs e)
{
    // חיפוש משתמש עם שם כזה
    DataTable dt_user = Select("SELECT * FROM [dbo].[Users] WHERE [userName] = '" +
savedNameTextBox.Text + "'");
    if (dt_user.Rows.Count > 0) // אם הוא קיים
    {
        MessageBox.Show("welcome back!"); // יוצאת הודעה מתאימה
        ClsGlobal.userNameString = "welcome back " + savedNameTextBox.Text;
        ClsGlobal.userName = savedNameTextBox.Text;
        DataTable dt = new DataTable();
        dt = Select("SELECT [sumScore] FROM [dbo].[Users] WHERE [userName] = '" +
savedNameTextBox.Text + "'"); // שולפים את הפרטים של השחקן
        ClsGlobal.sumScore = dt.Rows[0].Field<int>(0); // ושומרי מכמספר
        dt = Select("SELECT [doneLevels] FROM [dbo].[Users] WHERE [userName] = '" +
savedNameTextBox.Text + "'"); // שולפים את הפרטים של השחקן
        ClsGlobal.DoneLevels = dt.Rows[0].Field<string>(0); // ושומרים כמחרוזת
        LevelPage 1 = new LevelPage();
        Close();
        l.ShowDialog();
    }
    else notCorrectNameLabel.Visibility = Visibility.Visible;
}
}
```

### 5.3.3 כתיבת מספרי השלבים על הלחצנים: num\_of\_level-

הפונקציה נעזרת במידע מהמחלקה ClsGlobal – כדי לדעת איזו רמה בחר השחקן, ואיזה שלבים הוא כבר עשה בעבר (רלוונטי עבור שחקן לא חדש) לפי זה רושמת את מספרי השלבים על הלחצנים, וצובעת אותם בצבע ירוק אם הם נפתרו בעבר. כמו כן הפונקציה מגדירה עבור כל לחצן את הלחצן הבא אחריו, וזאת על מנת שהלחצנים יצבעו בצבעים





בס"ד

המתאימים תוך כדי המשחק (אדום כאשר פותרים שלב, וירוק אם פתרו אותו בהצלחה).

```
int[] rowSet = { 0, 1, 0, 1, 2, 1, 2, 2, 1, 0 }; // יעזור לסדר בלוח את הלחצנים
int[] columnSet = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; // יעזור לסדר בלוח את הלחצנים

void num_Of_Level(int Cls) // פונקציה שכותבת את מספרי השלבים על הלחצנים
{
    NewLevel[] levels = new NewLevel[10];
    levels[0] = new NewLevel();
    for (int i = 0; i < 10; i++)
    {
        levels[i].Num_level.Content = Cls * 10 + i + 1; // מספר השלב נכתב על הלחצן
        Grid.SetRow(levels[i], rowSet[i]);
        Grid.SetColumn(levels[i], columnSet[i]);
        g.Children.Add(levels[i]);
        if (i < 9)
        {
            levels[i + 1] = new NewLevel();
            levels[i].Set_Level(levels[i + 1]);
        }
        if (ClsGlobal.DoneLevels[Cls * 10 + i] == '1') // זה בא לסמן את השלבים שהשחקן עשה
        {
            levels[i].onWinn.Visibility = Visibility.Visible;
        }
    }
}
```

במשחקים קודמים

**5.3.4. בניית המשחק, פונקציית בניית הלוח העליון: Board** הגדרת מטריצה בגודל של 5 על 10 גומות, וצביעה של כל אחת בצבע המתאים לפי המידע השמור בקבצי XML, לצורך זה משתמשים במספר פונקציות: דבר ראשון בונים 5 על 10 גומות לבנות, שולפים מתוך דף XML את המידע על השלב, (לפי מספר השלב השמור במחלקה clsGlobal), ועוברים על כל צורה שבו בנפרד:

```
public Board() // את מיקום הצורות והצבעים xml-פונקציה שמחשבת ע"פ ה
{
    InitializeComponent();
    for (int i = 0; i < 5; i++) // הצבה של הגומות שיצרנו קודם בלוח
    {
        for (int j = 0; j < 10; j++)
        {
            GOfBoard[i, j] = new Gumot();
            Grid.SetRow(GOfBoard[i, j], i);
            Grid.SetColumn(GOfBoard[i, j], j);
            GOfBoard[i, j].ellipse.Fill = Brushes.White; // וצביעה בצבע לבן
            gBoard.Children.Add(GOfBoard[i, j]);
        }
    }
    XDocument doc = XDocument.Load(Environment.CurrentDirectory +
    "\\..\\..\\Cards.xml"); // שבו שמורים פרטי הכרטיסים xml-שליפה של דף ה
    foreach (XElement item in doc.Descendants("card"))
    {

```





בס"ד

```
var numOfCard =
int.Parse(item.Attribute("number").Value); // שלילת מספר כרטיס
if (numOfCard == ClsGlobal.numOfCard) // אם מספר הכרטיס שווה למספר
    הכרטיס שאנחנו צריכים
{
    foreach (var item1 in item.Descendants("part")) // שלילת מספר צורה
    {
        int numPart = int.Parse(item1.Attribute("number").Value);
        ClsGlobal.shapesUse[numPart - 1]++; // כדי שנדע שהשתמשנו
        בצורה הזו כבר
        int numDirection =
int.Parse(item1.Attribute("directionNum").Value); // שלילת מספר הכיוון הרצוי של הצורה
        int place = int.Parse(item1.Attribute("place").Value); // שלילת המיקום
        הרצוי של על פני הלוח שבו הצורה צריכה להיות
        NewMatInTheGrid(numPart, numDirection, place); // שליחה לפונקציה שבונה את
        הצורה במדויק בתוך המטריצה הנ"ל
    }
}
}
```

NewMatInTheGrid

- כל צורה נשלחת לפונקציה הזו, ששולפת את המידע עליה ומסדרת אותה במקום המתאים על פני הלוח

```
public void NewMatInTheGrid(int numG, int drG, int place) // פונקציה שמקבלת מספר צורה ומספר כיוון
ומסדרת במטריצה
{
    long dr = FindDr(numG, drG);

    // בדיקה אם זו צורה מסדר 4 על 4 או 3 על 3
    FindMatrixSize(numG);

    int i = place % 10;
    int j = place / 10;

    for (int k = 0; k < matrixSize; k++) // הצבה בלוח
    {
        for (int l = 0; l < matrixSize; l++)
        {
            long y = (dr / slicer) % 10; // ה"פ" או לאחד ע"פ
            if (y == 1) // אם הוא שווה אחד
            {
                switch (Col)
                {
                    case "dark green":
                        GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Green; // אם זה
                        צבע ירוק
                        break;
                    case "purple":
                        GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Purple; // אם זה
                        צבע סגול
                        break;
                    case "light azure":
                        GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.SkyBlue; // אם זה
                        צבע תכלת
                        break;
                }
            }
        }
    }
}
```



בס"ד

```
case "light green":
    GOfBoard[i + k, j + 1].ellipse.Fill =
Brushes.LightGreen; // אם זה צבע ירוק בהיר
    break;
case "azure":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Blue; // אם זה צבע
    break;
case "blue":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.DarkBlue; // אם
    break;
case "yellow":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Yellow; // אם זה
    break;
case "pink":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Pink; // אם זה
    break;
case "orange":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Orange; // אם זה
    break;
case "red":
    GOfBoard[i + k, j + 1].ellipse.Fill = Brushes.Red; // אם זה צבע
    break;
    }
    }
    slicer = slicer / 10;
    }
    }
```

כחול  
זה צבע כחול כהה  
צבע צהוב  
צבע ורוד  
צבע כתום  
אדום

שמוצאת את הקוד של הצורה + הכיוון הרצוי FIND DR הפונקציה שלעיל משתמשת בפונקציה

```
public long FindDr(int part, int dir) // ע"י מספר XML פונקציה ששולפת מתוך דפי
{
    XDocument doc = XDocument.Load(Environment.CurrentDirectory +
"\\..\\..\\Parts.xml"); // שבו שמורים פרטי הכרטיסים xml-שליפה של דף ה
    foreach (var item in doc.Descendants("part")) // xml-ריצה על דף ה
    {
        var num = int.Parse(item.Attribute("number").Value); // בדיקה האם זו הצורה שאנחנו
        if (num == part)
        {
            Col = item.Attribute("color").Value.ToString(); // שמירה של הצבע
            foreach (XElement item1 in item.Descendants("direction")) // ריצה על כל
            {
                var str = int.Parse(item1.Attribute("directionNum").Value); // בדיקה האם
                if (str == dir)
                {
                    DR = long.Parse(item1.Attribute("dr").Value); // שמירה של פרטי הכיוון
                }
            }
        }
    }
}
```

צורה ומספר כיוון  
צריכים  
הכיוונים האפשריים של הצורה  
זה הכיוון שאנחנו צריכים  
של הצורה - שמופיעים בצורה של אפסים ואחדות



בס"ד

```
        }  
    }  
}  
return DR;  
}
```

### 5.3.5. פונקציית בניית הלוח התחתון :OzherGumot

כל צורה שלא הוגדרה בלוח העליון יש להגדיר בלוח התחתון (כי בכל שלב במשחק משתתפות כל עשרת הצורות). הפונקציה עוברת על מערך עזר שבו שמורים מספרי הצורות שנבנו על בלוח העליון כדי לדעת אילו צורות צריך להגדיר, ולאחר מכן שולחת את מספר הצורה הרצוי לפונקציה שבונה צורה באותו האופן כמו בלוח העליון. מספר הכיוון מוגרל באופן רנדומלי, והמיקום על פני הלוח נקבע מראש, כדי שלא ייווצר מצב של חפיפת צורות על פני הלוח

```
public void OzherGumot(int[] help)//בניית שאר הצורות שלא היו בלוח  
{  
    int position = 0;  
    for (int u = 0; u < 10; u++)  
    {  
        if (help[u] == 0)//בדיקה במערך העזר האם הצורה המסויימת נמצאת בלוח או לא  
        {  
            doc = XDocument.Load(Environment.CurrentDirectory +  
                "\\..\\..\\Parts.xml");//עם פרטי הצורות xml-שליפה של דף ה  
            foreach (var item in doc.Descendants("part"))//xml-ריצה על הצורות שבדף ה  
            {  
                var num = int.Parse(item.Attribute("number").Value);  
                if (num - 1 == u)//אם זו הצורה שאנחנו צריכים  
                {  
                    var col = item.Attribute("color").Value.ToString();//שמירת הצבע של  
                    הצורה  
                    foreach (var item1 in item.Descendants("direction"))  
                    {  
                        var str = int.Parse(item1.Attribute("directionNum").Value);  
                        if (str == ClsGlobal.numDirection)//הכיוון של הצורה נקבע כל פעם  
                        ClsGlobal-מחדש בעזרת ה  
                        {  
                            ShearTzurot tsura = new ShearTzurot();  
                            tsura = tsura.newMat(u + 1,  
                                ClsGlobal.numDirection);//בניית הצורה  
                            for (position = 0; ClsGlobal.orderShapes[position] != 0 &&  
                                position < 7; position++) ;  
                            tsura.Tag = num + "" + str;//הכיוון ומס' הצורה  
                            switch (position)//בדיקה לאן אפשר להכניס את הצורה על פני הלוח, כך  
                            {  
                                case 0:  
                                    שלא תעלה על צורה אחרת  
                                }  
                                case 0:
```



בס"ד

```
canv1.Children.Add(tsura);
Canvas.SetTop(tsura, 0);
Canvas.SetLeft(tsura, 0);
break;
```

```
case 1:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 0);
    Canvas.SetLeft(tsura, 160);
    break;
case 2:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 0);
    Canvas.SetLeft(tsura, 320);
    break;
case 3:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 0);
    Canvas.SetLeft(tsura, 480);
    break;
case 4:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 173);
    Canvas.SetLeft(tsura, 0);
    break;
case 5:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 173);
    Canvas.SetLeft(tsura, 160);
    break;
case 6:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 173);
    Canvas.SetLeft(tsura, 320);
    break;
case 7:
    canv1.Children.Add(tsura);
    Canvas.SetTop(tsura, 173);
    Canvas.SetLeft(tsura, 480);
    break;
default:
    break;
```

```
}
ClsGlobal.orderShapes[position]++; // סימון המקום, כדי לא לשים
```

שם עוד צורה

```
    }
    }
    }
    ClsGlobal.numDirection++; // ע"פ ה"פ כל פעם נקבע הצורה כיוון ה"פ
    if (ClsGlobal.numDirection > 8) // אין יותר משמונה כיוונים לכל צורה
    {
        ClsGlobal.numDirection = 1; // ואין פחות מאחד
    }
}

for (int i = 0; i < 8; i++)
{
    ClsGlobal.orderShapes[i] = 0;
}
```

}

### 5.3.6. שעון (טיימר):

עבור כל שלב מוגדר זמן מוגבל שניתן בשביל לפתור אותו, אורך הזמן תלוי ברמת הקושי של השלב, עבור שלב קל ניתן פחות זמן, ועבור שלב קשה יותר זמן. אם הזמן נגמר לוח השלב נסגר.

```
int s = (ClsGlobal.numLevel + 1) * 100;
time = TimeSpan.FromSeconds(s);
timer = new DispatcherTimer(new TimeSpan(0, 0, 1), DispatcherPriority.Normal,
delegate
{
    if (time == TimeSpan.Zero)
    {
        timer.Stop();
        End();
    }
    labeltimer.Content = time.ToString();
    time = time.Add(TimeSpan.FromSeconds(-1));
}, Application.Current.Dispatcher);

timer.Start();
```

### 5.3.7. גרירת צורה ושחרורה על גבי הלוח:

לחיצת עכבר על צורה (מהלוח התחתון) מפעילה את הפונקציה canv1\_PreviewMouseLeftButtonDown ומאפשר את גרירת הצורה על גבי כל לוח המשחק.

```
private void canv1_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    if (e.Source == canv1)//נגררה הצורה מהיכן בדיקה
        return;

    if (!IsDragging)
    {
```



בס"ד

```
startPoint = e.GetPosition(canv1); // הצורה של ההתחלה נקודת שמירת (נגררה היא בדיקת מאיפה)
selectedElement = e.Source as UIElement;
canv1.CaptureMouse();

IsDragging = true;

selectedElementOrigins =
    new Point(
        Canvas.GetLeft(selectedElement),
        Canvas.GetTop(selectedElement));
}
e.Handled = true;
}
```

בזמן הגרירה מופעלת הפונקציה canv1\_PreviewMouseMove אשר מראה את תזוזת הצורה על גבי הלוח בזמן הגרירה, לנוחות השחקן.

```
private void canv1_PreviewMouseMove(object sender, MouseEventArgs e)
{
    if (canv1.IsMouseCaptured) // העכבר מצב על בדיקה
    {
        if (IsDragging)
        {
            currentPosition = e.GetPosition(canv1);
            elementLeft = (currentPosition.X - startPoint.X) +
                selectedElementOrigins.X;
            elementTop = (currentPosition.Y - startPoint.Y) +
                selectedElementOrigins.Y;
            Canvas.SetLeft(selectedElement, elementLeft); // הגרירה זמן כל הצורה מיקום שינוי
            Canvas.SetTop(selectedElement, elementTop);
        }
    }
}
```

בעת שחרור הצורה מופעל הפונקציה canv1\_PreviewMouseLeftButtonUp שמקבלת מידע על הצורה ועל נקודת השחרור, ומבצעת סדרת בדיקות (ע"י פונקציות עזר) כדי לדעת האם המקום שאליו הצורה נגררה הוא חוקי, כלומר האם יש לצורה מספיק מקום ריק כדי להשאר על פני הלוח. במידה וכן, הצורה תשאר במקום שאליו נגררה. במידה ולא, היא תחזור למקומה המקורי.

```
private void canv1_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)
{
    ShearTzurot tsura = new ShearTzurot();
    foreach (UIElement child in canv1.Children)
    {
        if (child.Equals(selectedElement))
            tsura = (ShearTzurot)(child);
    }
    if (canv1.IsMouseCaptured)
```

```

    {
        Point pBoard = new Point();
        Point pCanv1 = new Point();
        Board b1 =
Application.Current.Windows.OfType<Board>().FirstOrDefault();
        Board b2 = new Board();
        pBoard = e.GetPosition(b1);
        pCanv1 = e.GetPosition(tsura);
        if (pBoard.Y < 250) //בדיקה שנגחתנו בדיקה
        {
            b2.inBoard(canv1, pBoard, pCanv1); //עזר לפונקציית שליחה
            if (!ClsGlobal.flagForTheDrag)
            {
                Canvas.SetLeft(selectedElement, selectedElementOrigins.X); //לא המקום אם
חוקי,
                Canvas.SetTop(selectedElement, selectedElementOrigins.Y); //תחזור הצורה
נגררה שממנו למקום
            }
        }
        IsDragging = false; //ועוד העכבר שיחרור
        canv1.ReleaseMouseCapture();
        e.Handled = true;
    }
}
}
}
}

```

פונקציית העזר נקראת inBoard היא מקבלת מידע על הצורה ובודקת האם המקום מכיל עיגולים לבנים כך שיהיה די שטח ריק בשביל הצורה

```

public void InBoard(int partDir, Point mikum, Point mikumPnimi, Point startPlace) // פונקציית
שבודקת האם הצורה נגררה למקום חוקי או לא
{
    mw = Application.Current.Windows.OfType<MainWindow>().FirstOrDefault();

    bool toCardBoardFlag = mikum.Y > mw.limitLabel;

    int j = (int)(mikum.X / mw.spaceRelativelyBoard_X); //הצורה של הנחיתה
    int i = (int)(mikum.Y / mw.spaceRelativelyBoard_Y);
    if (j < 10)
    {
        Gumot g;
        if (toCardBoardFlag) //אם הצורה נגררה ללוח התחתון המטרה של הבדיקה היא רק לבקוד האם יש
צורך לשנות את הסימון של הגומות, ולצורך כך צריך "לאסוף" פרמטרים על הצורה
        {
            g = new Gumot();
        }
        else
            g = GOfBoard[i, j];

        int part = partDir / 10; //מחלצים את מספר הצורה
        int dir = partDir % 10; //מחלצים את מספר הכיוון של הצורה

        long DR = FindDr(part, dir);
        FindMatrixSize(part);

        j = j - (int)(mikumPnimi.X / mw.spaceBetweenGumot);
    }
}

```



בס"ד

```
i = i - (int)(mikumPnimi.Y / mw.spaceBetweenGumot);
long slicer1 = slicer;
bool flag = true; // צריך לצאת מהפונקציה
long y;

if (mikum.Y < mw.limitLabel)
{
    for (int k = 0; k < matrixSize; k++)
    {
        for (int l = 0; l < matrixSize; l++)
        {
            y = (DR / slicer1) % 10;
            if (y == 1 && flag)
            {
                if (i + k < 0 || j + l < 0 || i + k >= 5 || j + l >= 10 ||
GOfBoard[i + k, j + l].ellipse.Fill != Brushes.White || GOfBoard[i + k, j + l].IsTaken == true)
                {
                    flag = false;
                    k = matrixSize;
                }
            }
            slicer1 /= 10;
        }
    }
}
bool fromBoard = false;
if (!flag)
{
    ClsGlobal.flagForTheDrag = false; // לא, היא תחזור בחזרה
    if (startPlace.Y < mw.limitLabel) // צריך לשנות את
    המקור שלה ללבן רגיל
    {
        j = (int)(startPlace.X / mw.spaceRelativelyBoard_X) -
(int)(mikumPnimi.X / mw.spaceBetweenGumot); // נקודת המקור של הצורה, מאיפה היא נגררה
        i = (int)(startPlace.Y / mw.spaceRelativelyBoard_Y) -
(int)(mikumPnimi.Y / mw.spaceBetweenGumot);
        fromBoard = true;
        IsTaken(matrixSize, i, j, DR, slicer, fromBoard);
        // ClsGlobal.shapesUse[part-1] = 0;
    }
    // זה במקרה שנגרר מלמעלה ולא נשאר - לא צובעים כלום
    // או במקרה שנגרר מלמעלה ולמעלה ולא נשאר במקום החדש-צובעים את המקום הישן בלבן
}
else
{
    ClsGlobal.flagForTheDrag = true; // הצורה תשאר היכן שגררו אותה
    if (startPlace.Y < mw.limitLabel) // צריך לשנות את
    המקור שלה ללבן רגיל
    {
        // במקרה שנגרר מלמעלה ולמעלה ונשאר צובעים את המקום החדש, ומחזירים את הישן להיות כמו קודם
        int jj = (int)(startPlace.X / mw.spaceRelativelyBoard_X) -
(int)(mikumPnimi.X / mw.spaceBetweenGumot); // נקודת המקור של הצורה, מאיפה היא נגררה
        int ii = (int)(startPlace.Y / mw.spaceRelativelyBoard_Y) -
(int)(mikumPnimi.Y / mw.spaceBetweenGumot);
        fromBoard = true;
        IsTaken(matrixSize, ii, jj, DR, slicer, fromBoard);
        fromBoard = false;
    }
    // במקרה שנגרר מלמעלה ולמעלה וכן נשאר הישן חוזר להיות כמו
    קודם ושום דבר חדש לא נצבע
    {
        IsTaken(matrixSize, i, j, DR, slicer, fromBoard); // זה במקרה שנגרר מלמעלה
        ללמעלה וכן נשאר - צובעים את המקום החדש
    }
}
```





```
    }  
}
```

במידה והמקום חוקי הצורה נשארת במקום שאליו נגררה, והעיגולים הלבנים שמאחוריה מסומנים ע"י משתנה מיוחד שנוצר לצורך זה ע"י הפונקציה `isTaken`, כדי שלא יהיה ניתן לגרור עוד צורה מעליה

```
public void IsTaken(int godelMatriza, int i, int j, long DR, long slicer, bool fromBoard)  
{  
    for (int k = 0; k < godelMatriza; k++)  
    {  
        for (int l = 0; l < godelMatriza; l++)  
        {  
            var y = (DR / slicer) % 10;  
            if (y == 1)  
            {  
                if (fromBoard)  
                    GOfBoard[i + k, j + l].IsTaken = false;  
                else  
                    GOfBoard[i + k, j + l].IsTaken = true;  
            }  
            slicer /= 10;  
        }  
    }  
}
```

}



## בס"ד 5.3.8 שינוי הצורה

בעת לחיצה ימנית כפולה על צורה מהלוח התחתון מופעלת הפונקציה Tsuru\_MouseDoubleClick אשר פותחת מסך נוסף, ובו נראית הצורה שעליה לחצו, אך בהגדלה ובתלת מימד. ע"י החיצים <-> יש אפשרות לסובב את הצורה לכל הכיוונים, ולבחור את מצב הצורה שהכי מתאים ללוח

```
private void Tsuru_MouseDoubleClick(object sender, MouseButtonEventArgs e)//לחיצה על הצורה
{
    int part = 0, dir = 0;
    ShearTzurot st = sender as ShearTzurot;

    int partdir = Convert.ToInt32(st.Tag);
    part = partdir / 10;
    dir = partdir % 10;
    ThreeDBoard newthreedboard = new ThreeDBoard();
    newthreedboard.newMat(part, dir);
    newthreedboard.ShowDialog();
    newMat(part, newthreedboard.drNumToCardBoard);
}
```

אשר newMat הפונקציה הנ"ל שומרת את מספר הצורה ומספר הכיוון של הצורה שנלחצה, ושולחת לפונקציה בונה צורה דומה מעיגולים תלת מימדיים.

```
public ThreeDBoard NewMat(int prNum, int drNum)//המטריצה בניית
{
    partNumber = prNum;
    directionNumber = drNum;
    simun++; //כלום שאין בדיקה
    if (simun != 0)
        gridush.Children.Clear();
    ThreeDGumotGadol[,] Boal = new ThreeDGumotGadol[4, 4]; //על 4 מטריצה יצירת

    foreach (var item in doc.Descendants("part"))//המתאים החלק מציאת
    {
        int num = int.Parse(item.Attribute("number").Value);
        if (num == prNum)
        {
            string col = item.Attribute("color").Value.ToString(); //הצורה צבע שמירת
            foreach (var item1 in item.Descendants("direction"))//המתאים הכיוון מציאת
            {
                int str = int.Parse(item1.Attribute("directionNum").Value);
                if (str == drNum) //המתאים הכיוון את כשמצאנו
                {
                    long dr = long.Parse(item1.Attribute("dr").Value);
                    long ez_dr = dr;

                    //על 4 או 3 על 3 מסדר צורה זו האם בדיקה
                    int len = (int)Math.Log10(ez_dr);
                    int ezi_j = (int)Math.Sqrt(len);
                }
            }
        }
    }
}
```



```
long slicer;

if (ezi_j == 3) // על 3 מסדר זה אם
{
    //בהתאם העזרים עדכון
    slicer = 100000000;
    dr = dr % 1000000000;
}
else // על 4 מסדר זה אם
{
    //בהתאם העזרים עדכון
    slicer = 10000000000000000;
    dr = dr % 10000000000000000;
}
for (int i = 0; i < ezi_j; i++)
{
    for (int j = 0; j < ezi_j; j++)
    {
        long y = dr / slicer;

        if (y == 1) //לוח מציבים אחד יש xml-ה לפי אם
        {
            Boal[i, j] = new ThreeDGumotGadol(); //גומה יצירת

            switch (col) //המתאים בצבע צביעה
            {
                case "dark green":
                    Boal[i, j].cadur.Color = Brushes.Green.Color;
                    break;
                case "purple":
                    Boal[i, j].cadur.Color = Brushes.Purple.Color;
                    break;
                case "light azure":
                    Boal[i, j].cadur.Color = Brushes.Azure.Color;
                    break;
                case "light green":
                    Boal[i, j].cadur.Color =

Brushes.LightGreen.Color;

                    break;
                case "azure":
                    Boal[i, j].cadur.Color = Brushes.Blue.Color;
                    break;
                case "blue":
                    Boal[i, j].cadur.Color =

Brushes.DarkBlue.Color;

                    break;
                case "yellow":
                    Boal[i, j].cadur.Color = Brushes.Yellow.Color;
                    break;
                case "pink":
                    Boal[i, j].cadur.Color = Brushes.Pink.Color;
                    break;
                case "orange":
                    Boal[i, j].cadur.Color = Brushes.Orange.Color;
                    break;
                case "red":
                    Boal[i, j].cadur.Color = Brushes.Red.Color;
                    break;
            }
            Grid.SetRow(Boal[i, j], i);
            Grid.SetColumn(Boal[i, j], j);
        }
    }
}
```



```

        בס"ד
        gridush.Children.Add(Boal[i, j]);
    }
    dr = dr % slicer;
    if (slicer / 10 != 0)
        slicer = slicer / 10;
    }
    }
    }
    }
    }
    }
    return this; // הבנויה הצורה החזור
}
}
}
;
}
```

הפונקציות newnewMatRight newnewMatLeft בזמן לחיצה על החיצים <- -> שעל המקלדת מופעלות אשר משנות את כיוון הצורה, לכל צורה יש כאמור 8 כיוונים, ולכן אם נלחץ חץ אחד 8 פעמים הצורה תחזור לכיוון שהיה לפני השינוי

```

public void newnewMatLeft(int num, int dr) // 8 מספר - האחרון הכיוון לא שווה בדיקה
{
    if (dr != 8) // הבא לכיוון מעבר, לא זה ואם
    {
        newMat(num, dr + 1);
        drNumToCardBoard = dr + 1;
    }
    else
    {
        newMat(num, 1);
        drNumToCardBoard = 1;
    }
}

public void newnewMatRight(int num, int dr) // 1 מספר - הראשון הכיוון לא שווה בדיקה
{
    if (dr != 1) // הקודם לכיוון מעבר, לא זה ואם
    {
        newMat(num, dr - 1);
        drNumToCardBoard = dr - 1;
    }
    else
    {
        newMat(num, 8);
        drNumToCardBoard = 8;
    }
}
}
```

לאחר שינוי הצורה-בעת לחיצה על או יציאה מהדף הצורה שנלחצה על הלוח משנה את כיוונה בהתאם למה שנבחר ע"י השחקן



לאחר כל גרירה מופעלת פונקציה שבודקת האם השלב פתור ע"י סריקה של הלוח העליון ובדיקה האם נשארו שם נקודות לבנות פנויות, אם השלב נגמר, שומרים את הזמן שלקח לשחקן לפתור ואת מספר הגרירות שהוא עשה

```
public void EndLevel()
{
    bool flag = true;
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 10; j++)
            if (GOfBoard[i, j].ellipse.Fill == Brushes.White && GOfBoard[i, j].IsTaken
== false)
                flag = false;
    if (flag)
    {
        mw.endTime = mw.startTime - mw.time;
        EndLevel end = new EndLevel(mw.endTime.Seconds, mw.endTime.Minutes,
mw.allTimeInSeconds, ClsGlobal.numOfSteps);
        ClsGlobal.numOfSteps = 0;
        ClsGlobal.sumScore += mw.score;
        end.Show();
        mw.End();
    }
}
```

אופן חישוב הניקוד של השחקן הוא פשוט:

```
//פונקציה לחישוב ניקוד: כמה שפחות צעדים ופחות זמן (יחסית לזמן שניתן) ככה הניקוד יותר גבוה//
מה שעושים- מכפילים את מס' הצעדים ביחס שבין הזמן שלקח לפתור לזמן שניתן מלכתחילה, ומחסירים את התוצאה
מ1200 (רמז ל-120 שלבי במשחק)
private int Score(int steps, int seconds)
{
    return 1200 - steps * seconds;//שאלה- לבדוק- שאלה-
}
```

לאחר סיום השלב הלחצן של אותו שלב נצבע בצבע ירוק, ונתוני השחקן המתאימים (מספר הנקודות והשלבים שהוא פתר) מתעדכנים, צורת העדכון זהה לצורה שבה שלפנו ושמרנו נתונים בעת הכניסה למשחק, רק שהפקודה כעת היא שונה:

```
try
{
    Select("UPDATE [dbo].[Users] SET sumScore='" + ClsGlobal.sumScore + "',
doneLevels='" + ClsGlobal.DoneLevels + "' WHERE [userName]= '" + ClsGlobal.userName +
"'");
}
catch (Exception ex)
{ MessageBox.Show(ex.Message); }
```



## בס"ד 5.3.10. הצגת לוח השיאים

אופן שליפת וסידור הנתונים זהה לאופן שבו שלפנו ועדכנו את מסד הנתונים כבר קודם, רק עם פקודה שונה:

```
try
{
    Select("SELECT userName,sumScore FROM[dbo].[Users] ORDER BY [sumScore] ");
    //שליפת פרטי השחקנים+הנקודות, בסדר יורד/
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

## 5.3.11 אנימצית הרקע

בעת מראה טבלת השיאים מופעלת ברקע אנימציה של קווים ועיגולים צבעוניים היוצרים יחד צורות המזכירות את הצורות שבלוח המשחק. העיגולים זזים כל הזמן בקצב איטי, ואם המרחק בין 2 עיגולים קטן מאורך מסוים נוצר בין העיגולים קשר- קו צבעוני. והעכבר נחשב גם הוא כ"עיגול" לצורך זה וגם בינו לבין שאר העיגולים נוצרים קשרים, מה שיוצר אשליה של "מעקב" אחר העכבר. הקווים והעיגולים נוצרים ע"י 3 פונקציות שיוצרות אותם (ב-3 צבעים שונים):

העיגולים נוצרים ע"י הפונקציות:

CreateParalax

CreateParalaxBackground

CreateParalaxBackground2

להלן הפונקציות:

```
public void CreateParalax()
{
    Random random = new Random();

    for (int i = 0; i < count; i++)
    {
        ellipseInfo newElement = new ellipseInfo();
        newElement.x = random.Next(20, (int)Width - 20);
        newElement.y = random.Next(20, (int)Height - 20);

        Ellipse ellipse = new Ellipse();
        Canvas.SetLeft(ellipse, newElement.x);
        Canvas.SetTop(ellipse, newElement.y);

        int widthEllips = random.Next(3, maxSize);

        ellipse.Width = widthEllips;
    }
}
```



ט"ו

```
ellipse.Height = widthEllips;  
ellipse.Fill = new SolidColorBrush(Colors.Blue);
```

```
newElement.ellipse = ellipse;  
canvas.Children.Add(newElement.ellipse);  
allEllipse.Add(newElement);  
}
```

```
ellipseInfo mouseElement = new ellipseInfo();  
mouseElement.x = (int)Mouse.GetPosition(canvas).X;  
mouseElement.y = (int)Mouse.GetPosition(canvas).Y;
```

```
Ellipse mouseEllipse = new Ellipse();  
mouseEllipse.Width = 2;  
mouseEllipse.Height = 2;  
Canvas.SetLeft(mouseEllipse, mouseElement.x);  
Canvas.SetTop(mouseEllipse, mouseElement.y);
```

```
mouseElement.ellipse = mouseEllipse;  
canvas.Children.Add(mouseElement.ellipse);  
allEllipse.Add(mouseElement);
```

```
public void CreateParalaxBackground()  
{  
    // יוצר רנדום  
    Random random = new Random();  
  
    // יצירת נקודות  
    for (int i = 0; i < count; i++)  
    {  
        // יצירת האלמנט  
        ellipseInfo newElement = new ellipseInfo();  
        // יצירת קואורדינטות רנדומליות  
        newElement.x = random.Next(20, (int)Width - 20);  
        newElement.y = random.Next(20, (int)Height - 20);  
  
        // יצירת עיגול  
        Ellipse ellipse = new Ellipse();  
        // קואורדינטות  
        Canvas.SetLeft(ellipse, newElement.x);  
        Canvas.SetTop(ellipse, newElement.y);  
  
        // גובה+רוחב רנדומלי  
        int widthEllips = random.Next(3, maxSize);  
  
        // שמירת הגובה והרוחב כמשתנים של הצורה  
        ellipse.Width = widthEllips;  
        ellipse.Height = widthEllips;  
        // צביעה  
        ellipse.Fill = new SolidColorBrush(Colors.Orange);
```



```
// שומרים את הנתונים
newElement.ellipse = ellipse;
// מוסיפים את העיגול לקנבס
canvas.Children.Add(newElement.ellipse);
// שומרים במערך העיגולים
allEllipseBackground.Add(newElement);
}

// מידע על העיגול בשביל העכבר
ellipseInfo mouseElement = new ellipseInfo();
// הקואורדינטות של העיגול
mouseElement.x = (int)Mouse.GetPosition(canvas).X;
mouseElement.y = (int)Mouse.GetPosition(canvas).Y;

// יצירת עיגול
Ellipse mouseEllipse = new Ellipse();
// גובה+רוחב של העיגול
mouseEllipse.Width = 2;
mouseEllipse.Height = 2;
Canvas.SetLeft(mouseEllipse, mouseElement.x);
Canvas.SetTop(mouseEllipse, mouseElement.y);

// שומרים את הנתונים
mouseElement.ellipse = mouseEllipse;
// מוסיפים את העיגול לקנבס
canvas.Children.Add(mouseElement.ellipse);
// שומרים במערך העיגולים
allEllipseBackground.Add(mouseElement);
} }
```

```
public void CreateParallaxBackground2()
{
    Random random = new Random();

    for (int i = 0; i < count; i++)
    {
        ellipseInfo newElement = new ellipseInfo();
        newElement.x = random.Next(20, (int)Width - 20);
        newElement.y = random.Next(20, (int)Height - 20);

        Ellipse ellipse = new Ellipse();
        Canvas.SetLeft(ellipse, newElement.x);
        Canvas.SetTop(ellipse, newElement.y);

        int widthEllips = random.Next(3, maxSize);

        ellipse.Width = widthEllips;
        ellipse.Height = widthEllips;
        ellipse.Fill = new SolidColorBrush(Colors.Pink); //it was Gray
        //GetEllipseColor(ellipse);

        newElement.ellipse = ellipse;
    }
}
```





7"01

```
        canvas.Children.Add(newElement.ellipse);
        allEllipseBackground2.Add(newElement);
    }

    ellipseInfo mouseElement = new ellipseInfo();
    mouseElement.x = (int)Mouse.GetPosition(canvas).X;
    mouseElement.y = (int)Mouse.GetPosition(canvas).Y;

    Ellipse mouseEllipse = new Ellipse();
    mouseEllipse.Width = 2;
    mouseEllipse.Height = 2;
    Canvas.SetLeft(mouseEllipse, mouseElement.x);
    Canvas.SetTop(mouseEllipse, mouseElement.y);

    mouseElement.ellipse = mouseEllipse;
    canvas.Children.Add(mouseElement.ellipse);
    allEllipseBackground2.Add(mouseElement);
}
}
```



בס"ד  
ליצור יצירת הקווים משתמשים ב-3 פונקציות:

UpdateParallax

UpdateParallaxBackground

UpdateParallaxBackground2

להלן הפונקציות:

```
private void UpdateParallax(object sender, EventArgs e)
{
    // יצירת רנדום
    Random random = new Random();

    // קליטת קואורדינטות העכבר
    allEllipse[count].x = (int)Mouse.GetPosition(canvas).X;
    allEllipse[count].y = (int)Mouse.GetPosition(canvas).Y;

    // עוברים על הנקודות שברמה המתאימה
    for (int i = 0; i < allEllipse.Count; i++)
    {
        // אם הנקודה הספציפית ביניהם
        if (i < count)
        {
            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipse[i].x < 20)
            {
                // מזיזים את הנקודה
                allEllipse[i].x += random.Next(0, maxTranform);
            }
            else if (allEllipse[i].x > Width - 20)
            {
                // מעבירים אותה למקום החדש
                allEllipse[i].x += random.Next(-maxTranform, 0);
            }
            else
            {
                // מעבירים למקום החדש
                allEllipse[i].x += random.Next(-maxTranform, maxTranform + 1);
            }

            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipse[i].y < 20)
            {
                // מזיזים את הנקודה
                allEllipse[i].y += random.Next(0, maxTranform);
            }
            else if (allEllipse[i].y > Height - 20)
            {
                // מעבירים אותה למקום החדש
                allEllipse[i].y += random.Next(-maxTranform, 0);
            }
            else
            {
                // מעבירים אותה למקום החדש
                allEllipse[i].y += random.Next(-maxTranform, maxTranform + 1);
            }
        }
    }
}
```



בס"ד

```
// מעבירים את הנקודה על פני הקנבס
Canvas.SetLeft(allEllipse[i].ellipse, allEllipse[i].x);
Canvas.SetTop(allEllipse[i].ellipse, allEllipse[i].y);

// מבטלים את הקווים שהיו קודם
for (int j = 0; j < allEllipse[i].lines.Count; j++)
{
    canvas.Children.Remove(allEllipse[i].lines[j]);
    allEllipse[i].lines.Remove(allEllipse[i].lines[j]);
}

// עוברים על שאר העיגולים
for (int j = i + 1; j < allEllipse.Count; j++)
{
    // קואורדינטות נקודה מס 1
    double x1 = allEllipse[i].x + allEllipse[i].ellipse.Width / 2;
    double y1 = allEllipse[i].y + allEllipse[i].ellipse.Width / 2;

    // קואורדינטות נקודה מס 2
    double x2 = allEllipse[j].x + allEllipse[j].ellipse.Width / 2;
    double y2 = allEllipse[j].y + allEllipse[j].ellipse.Width / 2;

    double distention = Math.Sqrt(Math.Pow((x2 - x1), 2) + Math.Pow((y2 - y1),
2));

    if (distention < minDistention)
    {
        Line line = new Line();
        line.X1 = x1;
        line.Y1 = y1;
        line.X2 = x2;
        line.Y2 = y2;
        line.Stroke = new SolidColorBrush(Colors.Red);

        line.StrokeThickness = 1;

        canvas.Children.Add(line);
        allEllipse[i].lines.Add(line);
    }
}
}
```



```
private void UpdateParallaxBackground(object sender, EventArgs e)
{
    Random random = new Random();

    allEllipseBackground[count].x = (int)Mouse.GetPosition(canvas).X;
    allEllipseBackground[count].y = (int)Mouse.GetPosition(canvas).Y;

    for (int i = 0; i < allEllipseBackground.Count; i++)
    {
        if (i < count)
        {
            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipseBackground[i].x < 20)
            {
                allEllipseBackground[i].x += random.Next(0, maxTranform);
            }
            else if (allEllipseBackground[i].x > Width - 20)
            {
                allEllipseBackground[i].x += random.Next(-maxTranform, 0);
            }
            else
            {
                allEllipseBackground[i].x += random.Next(-maxTranform, maxTranform +
1);

            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipseBackground[i].y < 20)
            {
                allEllipseBackground[i].y += random.Next(0, maxTranform);
            }
            else if (allEllipseBackground[i].y > Height - 20)
            {
                allEllipseBackground[i].y += random.Next(-maxTranform, 0);
            }
            else
            {
                allEllipseBackground[i].y += random.Next(-maxTranform, maxTranform +
1);
            }
        }

        Canvas.SetLeft(allEllipseBackground[i].ellipse, allEllipseBackground[i].x);
        Canvas.SetTop(allEllipseBackground[i].ellipse, allEllipseBackground[i].y);

        for (int j = 0; j < allEllipseBackground[i].lines.Count; j++)
        {
            canvas.Children.Remove(allEllipseBackground[i].lines[j]);
            allEllipseBackground[i].lines.Remove(allEllipseBackground[i].lines[j]);
        }

        for (int j = i + 1; j < allEllipse.Count; j++)
        {
            double x1 = allEllipseBackground[i].x +
allEllipseBackground[i].ellipse.Width / 2;
            double y1 = allEllipseBackground[i].y +
allEllipseBackground[i].ellipse.Width / 2;
```



7"01

```
2));  
  
        double x2 = allEllipseBackground[j].x +  
allEllipseBackground[j].ellipse.Width / 2;  
        double y2 = allEllipseBackground[j].y +  
allEllipseBackground[j].ellipse.Width / 2;  
  
        double distention = Math.Sqrt(Math.Pow((x2 - x1), 2) + Math.Pow((y2 - y1),  
  
        if (distention < minDistentionBackground)  
        {  
            Line line = new Line();  
            line.X1 = x1;  
            line.Y1 = y1;  
            line.X2 = x2;  
            line.Y2 = y2;  
            line.Stroke = new SolidColorBrush(Colors.Green);  
  
            line.StrokeThickness = 1;  
  
            canvas.Children.Add(line);  
            allEllipseBackground[i].lines.Add(line);  
        }  
    }  
}
```



```
private void UpdateParallaxBackground2(object sender, EventArgs e)
{
    Random random = new Random();

    allEllipseBackground2[count].x = (int)Mouse.GetPosition(canvas).X;
    allEllipseBackground2[count].y = (int)Mouse.GetPosition(canvas).Y;

    for (int i = 0; i < allEllipseBackground2.Count; i++)
    {
        if (i < count)
        {
            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipseBackground2[i].x < 20)
            {
                allEllipseBackground2[i].x += random.Next(0, maxTranform);
            }
            else if (allEllipseBackground2[i].x > Width - 20)
            {
                allEllipseBackground2[i].x += random.Next(-maxTranform, 0);
            }
            else
            {
                allEllipseBackground2[i].x += random.Next(-maxTranform, maxTranform +
1);
            }

            // מגבילים את התזוזה כדי שלא יצא מהמסך
            if (allEllipseBackground2[i].y < 20)
            {
                allEllipseBackground2[i].y += random.Next(0, maxTranform);
            }
            else if (allEllipseBackground2[i].y > Height - 20)
            {
                allEllipseBackground2[i].y += random.Next(-maxTranform, 0);
            }
            else
            {
                allEllipseBackground2[i].y += random.Next(-maxTranform, maxTranform +
1);
            }
        }

        Canvas.SetLeft(allEllipseBackground2[i].ellipse, allEllipseBackground2[i].x);
        Canvas.SetTop(allEllipseBackground2[i].ellipse, allEllipseBackground2[i].y);

        for (int j = 0; j < allEllipseBackground2[i].lines.Count; j++)
        {
            canvas.Children.Remove(allEllipseBackground2[i].lines[j]);
            allEllipseBackground2[i].lines.Remove(allEllipseBackground2[i].lines[j]);
        }

        for (int j = i + 1; j < allEllipse.Count; j++)
        {
            double x1 = allEllipseBackground2[i].x +
allEllipseBackground2[i].ellipse.Width / 2;
            double y1 = allEllipseBackground2[i].y +
allEllipseBackground2[i].ellipse.Width / 2;
        }
    }
}
```



7"01

```
2));  
  
        double x2 = allEllipseBackground2[j].x +  
allEllipseBackground2[j].ellipse.Width / 2;  
        double y2 = allEllipseBackground2[j].y +  
allEllipseBackground2[j].ellipse.Width / 2;  
  
        double distention = Math.Sqrt(Math.Pow((x2 - x1), 2) + Math.Pow((y2 - y1),  
  
        if (distention < minDistentionBackground2)  
        {  
            Line line = new Line();  
            line.X1 = x1;  
            line.Y1 = y1;  
            line.X2 = x2;  
            line.Y2 = y2;  
            line.Stroke = new SolidColorBrush(Colors.Yellow); //it was gray  
            //GetLineColor(line);  
            line.StrokeThickness = 1;  
  
            canvas.Children.Add(line);  
            allEllipseBackground2[i].lines.Add(line);  
        }  
    }  
}
```



בס"ד

## 5.4. בדיקת המערכת:

לאחר שעות רבות של השקעה, חשיבה ומאמצים מרובים התיישבנו לבדוק את

המערכת שבנינו. ערכנו תצפיות רבות על תקינות המערכת,

ערכנו בדיקות מדוקדקות על מנת לוודא שאכן המערכת פועלת כראוי.

בדקנו את נכונות האלגוריתמים והפונקציות השונות, ווידאנו שכל הנתונים באים לידי ביטוי

בפונקציות השונות בצורה נכונה, בזמן ובמקום המתאימים.

על מנת לוודא כל אלה ועוד, השתמשנו בBreakPoint.

להלן עיקרי הבדיקות:

הפונקציה הנבדקת	תאור הבדיקה	התוצאה
פונקציית הרמז	בדיקה שהרמז שהפונקציה מחזירה אכן מספק פתרון אפשרי של החידה	הרמז שסופק היה נכון.
פונקציית הרמז	בדיקה שהלוח חוזר לצבע המקורי לאחר 3 שניות	הלוח אכן חזר לצבע המקורי.
שמירת פרטי השחקן בטבלת SQL	שמירה של שחקנים חדשים, ובדיקת הפרטים בטבלה, ולאחר מכן פתיחת משחק שמור עם אותו השם, ובדיקת הפרטים שנשלפו	פרטי השחקן נשמרים ונשלפים כשורה
בניית לוח משחק	השוואה של הלוח העליון והלוח התחתון שנבנו ע"י הפונקציות למה שמצופה ע"י המידע השמור בקובץ ה-XML	הלוחות נבנו כמצופה
גרירה ושחרור	גרירת צורות למקומות חוקיים ולמקומות לא חוקיים	כאשר הצורה נגררה למקום חוקי היא נשארה במקום, וכאשר היא נגררה למקום לא חוקי היא חזרה למקומה המקורי
שינוי צורה	בדיקת כל שלבי שינוי הצורה- מהגדלתה, שינויי כיווניה, ושינוי הצורה המקורית שבלוח	הכל עבד כנדרש
סיום שלב	בדיקה שהטיימר עובד כנדרש, שהשלב נסגר כאשר נגמר הזמן, ושפרטי השחקן מתעדכנים כאשר הוא מסיים שלב בהצלחה	הכל עבד כנדרש
לוח השיאים	בדיקה שכל הפרטים נשלפו מהטבלה בצורה מדוייקת, ושאינמציות הרקע עובדת כנדרש	הפרטים נשלפו מהטבלה במדוייק, ואינמציות הרקע עבדה כנדרש





בס"ד

## 6. מה הקנה הפרוייקט

מבחינה לוגית - המשחק מורכב מפרטים רבים ודרש הרבה מחשבה בתכנון האלגוריתם ליישומם.

מבחינה טכנולוגית: את הפרוייקט כתבנו בשפת C#, שפה מעניינת ומלאת פונקציונליות, תוך שימוש נרחב ומקיף ברכיבי התוכנה ובאפשרויות ש- .net. נותנת. כמו כן הפרוייקט משתמש בהרבה רכיבים ומחלקות והיה עלינו להכיר ולהתמצא בהרבה מאוד טכנולוגיות חדשות, כגון: קריאה מקובץ xml.

שליפה ועדכון של טבלאות SQL

עבודה עם טכנולוגית תלת המימד

עבודה עם טכנולוגיית גרירה ושחרור עצמים על פני הלוח

טכנולוגית ה WPF שזוהי טכנולוגיה חדישה לניהול ממשק גרפי מתקדם ללא בזבז מיותר של משאבי מערכת.

לסיכום, הפרוייקט הקנה לנו :

✓ ידע נרחב בשפת C#.

✓ ידע נרחב בטכנולוגית WPF.

✓ נסיון בשפת SQL

✓ התמודדות עם פרויקט בהיקף גדול.

✓ התמודדות עם פיתוח משחק קיים בצורה מעניינת ומאתגרת ביותר.



## ביבליוגרפיה

### 7. ביבליוגרפיה

#### 7.1. אתרי אינטרנט

Visual c# 2012 □

[www.corner.co.il](http://www.corner.co.il) □

<http://www.wpftutorial.net> □

<http://msdn.microsoft.com/es-es/library/ms750612.aspx> □

<http://dotnetslackers.com> □

<http://en.csharp-online.net> □

<http://www.go4answers.com/Example/wpf-poligon-shape-bitmap-memory-stream-72267.aspx> □

[http://www.codeproject.com/KB/WPF/rounded\\_corner\\_polygon.aspx](http://www.codeproject.com/KB/WPF/rounded_corner_polygon.aspx) □

<http://switchonthecode.com/tutorials/wpf-the-basedon-style-property>

<https://you-hands.ru/2018/09/25/wpf-sozdanie-effekta-sozvezdiya/>

<https://you-hands.ru/2018/08/31/wpf-podklyuchenie-k-baze-dannyx-ms-sql-server/>

<https://you-hands.ru/2018/08/31/wpf-vypolnenie-zaprosov-k-baze-dannyx-ms-sql-server/>

<https://stackoverflow.com/>