****

פרויקט ב-5 יח"ל בינה מלאכותית

****

**שם המבצעת:** דנה בידה.

**שם המורה המנחה:** אסתי מאסטרסי.

**שם בית הספר:** בית הספר הריאלי העברי בחיפה.

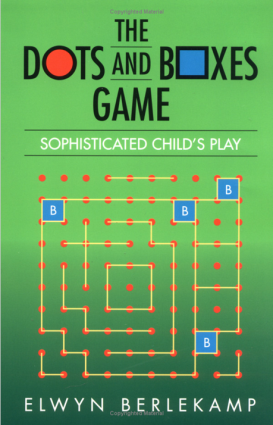
**תאריך ההגשה:** 28.4.12

**תוכן עניינים:**

|  |  |
| --- | --- |
| נושא | עמוד |
| **הקדמה** | 3 |
| **אפיון המערכת + מדריך להפעלתה** | 2-3 |
| **ייצוג בסיס הידע כתכנית פרולוג**  1. תיאור העובדות  2. סוגי נתונים מופשטים לייצוג הידע  3. תיאור החוקים (מלבד חוקי הבינה המלאכותית) \*החוקים שפותחים את חלון המשחק \*החוקים שמאתחלים את העובדות הדינאמיות של המשחק \*החוקים שמציגים על לוח המשחק את הנתונים \*החוקים שבונים את לוח המשחק | 6-8  9-10  11-16 16-18 18-23 24-29 |
| **תיאור מעטפת המערכת המומחית**  1. הבינה המלאכותית- הסבר  2. חוקי הבינה המלאכותית- יישום | 30  31-44 |
| **הממשק למשתמש** | 45-46 |
| **סיכום אישי (בעיות, הצעות לשיפורים ומשו"ב אישי)** | 47 |
| **נספחים**  1. ביבליוגרפיה  2. חוקי המשחק  3. קוד התוכנית (כולל תיעוד) | 49  50  51-70 |

**הקדמה:**

המשחק "Dots and Boxes" (הידוע גם כ: Boxes, Squares, Paddocks, Square-it,   
Dots and Dashes, Dots, Smart Dots, Dot Boxing, Dot Game) הוא משחק "עפרון ודף" לשני שחקנים.   
את המשחק פרסם לראשונה ב-1889 אדוארד לוקאס - מתמטיקאי צרפתי.   
משחק זה הוא משחק שתלוי ברובו בסדר התורות במשחק.  
משחק כזה נקרא באנגלית: ""Impartial game', כלומר "פניות המשחק". הכוונה היא שאם ישנם שני שחקנים באותה הרמה (מבחינת ידע), אזי הסיכוי של שניהם לנצח תלוי רק במי שיקבל ראשון את התור ויוכל לסגור את הריבועים שעל הלוח. כמובן שזה קורה רק לאחר שהשחקנים נקטו בכל המהלכים האפשריים בהם שחקן אחד חוסם את השחקן השני בכדי שהיריב לא יסגור ריבוע.  
משחק זה כמה שהוא נשמע בתיאוריה פשוט, יכול להיות גם מסובך.  
המון אנשים ניסו לחשוב על אסטרטגיה - שיטה לנצח את היריב. אחד מאלו היה אלווין ברלקאמף (Elwyn R. Berlekamp) שכתב את הספר:   
The Dots and Boxes Game: Sophisticated Child's Play שכולל את התיאוריה השלמה של המשחק ואת האסטרטגיות בהם יש לנקוט על מנת לנצח במשחק.

 כיום, ניתן לשחק במשחק זה (כמו פעם בעיפרון ודף) בעיתון השבועי.  
בנוסף, יש גם מספר תוכנות למשחק זה באינטרנט אך רק באפשרות משחק של שחקן אנושי מול שחקן אנושי.  
  
המטרה שלי בכתיבת הפרויקט הייתה ליצור משחק בו שחקן אנושי יוכל לשחק גם מול המחשב ולא רק מול שחקן אנושי. באופן זה לשחקן האנושי יש יותר אתגר ויותר קושי לנצח.  
  
תחום הידע: בכתיבת התוכנה הסתמכתי על הניסיון שלי אשר נצבר עם הזמן- ככל ששיחקתי יותר הבנתי יותר למה יש לשים לב במשחק ואיך אוכל לנצח את היריב. בנוסף לאמור לעיל, מבדיקה מעמיקה הבנתי שמשחק זה תלוי גם בעיקר בסדר התורות והשחקן שתורו לשחק במצב בו יש ריבועים על הלוח- הוא זה שיזכה לסגור את הריבועים שעל הלוח.

**אפיון המערכת:**

שם המערכת: משחק "Dots and Boxes"

קהל היעד: המשחק מיועד לכל הגילאים ובעיקר לאלו שאוהבים משחקים תחרותיים שבכדי לנצח בהם יש להפעיל את הראש ולחשוב קצת מעבר ולפתח אסטרטגיה במשחק.

מומחיות המערכת: את המשחק ניתן לשחק ב-3 דרכים שונות:

1. משחק של *שחקן אנושי מול שחקן אנושי*.

2. משחק של *שחקן אנושי מול המחשב- ברמת קושי קלה*:  
המחשב שם בתורו קו אנכי או אופקי באופן רנדומאלי בלבד! הוא לא חושב איזה מהלך חכם.

3. משחק של *שחקן אנושי מול המחשב- ברמת קושי קשה*:  
המחשב בוחר באופן חכם את הקו אותו הוא ישים בתורו על הלוח: המחשב בודק איזה קו יוכל לסגור ריבוע ואם אין כזה הוא שם קו תוך בדיקה שהקו הזה לא יגרום לשחקן האנושי לסגור ריבוע. אם אין לו ברירה הוא מגריל אחד מן הקווים שעל הלוח. ברמה זאת מומחיות המערכת היא לנצח את השחקן האנושי במשחק.

**מדריך להפעלת המערכת:**

בכדי להריץ את המשחק, ראשית יש להוריד את התוכנה המכונה "SWI-Prolog". לאחר התקנת התוכנה יש להעביר את כל התמונות (הגרפיקה) שבתוך העבודה לתיקייה שנמצאת בתוך התיקייה בכונן בו התקנתם את הפרולוג. תיקייה זו מכונה bitmaps והיא נמצאת בתוך   
התיקייה xpce שבתוך תיקיית הפרולוג שאת שמה אתם בחרתם עם התקנת התוכנה.  
דוגמה למיקום: . C:\Program Files\pl\xpce\bitmaps

F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\playvsfriend.bmpלאחר שהתקנתם את התוכנה, פתחו אותה ורשמו על המסך שיופיע לכם עם פתיחת התוכנה (הנקרא מסך השאילתות) .emacs ואז יפתח חלון. אל חלון זה תגררו את קובץ העבודה. חיזרו לחלון הקודם- חלון השאילתות וכתבו start. .עכשיו נפתח המשחק !  
  
בכדי לשחק משחק של *שחקן אנושי מול שחקן אנושי* לחץ על הכפתור:

F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\playvscomputer_easy.bmpבכדי לשחק משחק של *שחקן אנושי מול המחשב- ברמת קושי קלה*, לחץ על הכפתור:

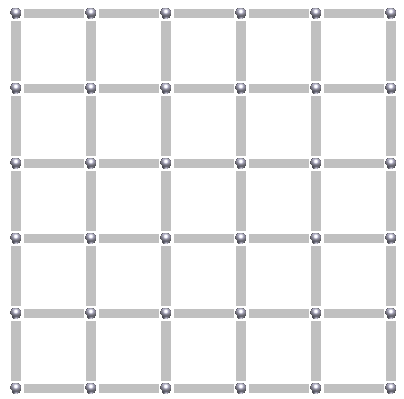
F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\playvscomputer_hard.bmpובכדי לשחק משחק של *שחקן אנושי מול המחשב- ברמת קושי קשה*, לחץ על הכפתור:

F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\About.bmpF:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\Rules.bmp  
בכדי לראות את חוקי המשחק לחץ על הכפתור:   
  
בכדי לראות את אודות המשחק לחץ על הכפתור:

בכל פעם שתרצה לחזור לתפריט בו מופיעים כל הלחצנים האלו, לחץ על הכפתור:   
חשוב לציין!! עם לחיצה על כפתור זה, המשחק בו אתם נמצאים יימחק  
ותתחילו משחק חדש בהתאם ללחיצה מחדש על המשחק אותו תבחרו.



**תרשים המשחק:**

**F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\hightLine_green.bmpF:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\widthLine_red.bmp**

**שחקן אנושי לחץ כבר על קו אנכי זה- קו זה כבר תפוס ולא ניתן ללחוץ עליו שוב!**

**לוח המשחק:  
בכל תור השחקן לוחץ על הקו האפור שהוא בחר, עד שכבר לא נותרו קווים אפורים ובכך הסתיים המשחק.**

**כאן יופיע מספר הנקודות (המתעדכן בכל תור) של כל שחקן: הריבועים הירוקים שייכים לשחקן מס' 2 והריבועים האדומים לשחקן מס' 1.**

**כאן יופיע התור הנוכחי של השחקן , כאשר במשחק מול המחשב: השחקן האדום הוא המחשב והירוק הוא השחקן האנושי.**

**F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\widthLine_red.bmpF:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\hightLine_green.bmpF:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\hightLine_green.bmp**

**F:\פרוייקט במחשבים\תמונות לפרוייקט\תמונות לפרוייקט\widthLine_red.bmp**

**המחשב סגר פה ריבוע וקיבל נקודה.  
(X אדום – נק' למחשב  
X ירוק – נק' לשחקן האנושי )**

**המחשב בחר כבר את הקו האופקי הזה- קו זה כבר תפוס ולא ניתן ללחוץ עליו שוב!**

* **הגראפיקה של המשחק** היא כל התמונות שמציגות את נתוני המחשב הכתובים על המסך. בפרולוג הגראפיקה מתבצעת בעזרת פעולות שמורות שמאפשרות לפתוח חלון משחק ועליו להציג תמונות שנשמרות בתיקייה ספציפית במחשב על מסך המשחק בכל מיקום שאנו נבחר. המיקום מיוצג על ידיי שני מספרים שגודלם מתאר את רוחב ואורך המסך.

**כיצד התוכנה בפרולוג בנויה?**  
**ייצוג בסיס הידע:**

בפרולוג הייצוג של המשחק מתבצע בעזרת עובדות.

העובדות הקבועות: (העובדות שלא משתנות במהלך המשחק)

התמונות של הנקודות של כל משתמש :   
 scorePlayer(NumofScore,TheAppropriatePicture)

( scoreRed(sumScoreRed,Pic - תמונת הנקודות של השחקן האדום  
scoreGreen(sumScoreGreen,Pic) – תמונת הנקודות של השחקן הירוק

מספר הנקודות שיש לשחקן בתור הנוכחי (Green/sumScoreRed) NumofScore :   
התמונה שמתאימה למספר הנקודות הנ"ל TheAppropriatePicture : (Pic)

מאחר והמחשב קרס כאשר העליתי בכל פעם מחדש את המשחק עם עדכון הנקודות על הלוח כטקסט חדש, בשל עומס במקום שיש לו לאחסון המידע/הנתונים, אזי החלטתי לייצג את הנקודות של השחקנים בצורה גראפית על ידי תמונה ולא על ידי טקסט. בכדי שיהיה לי יותר קל להעלות את התמונות בכל פעם שמתעדכנות הנקודות, עשיתי עובדות שמצמידות את מספר הנקודות עם התמונה המתאימה ואז אני פשוט מזמנת את התמונה המתאימה למספר הנקודות הנוכחי.   
לדוגמא: scoreRed(0,'0r.bmp') –0r זוהי התמונה שעל הלוח עבור 0 נקודות לשחקן האדום.

העובדות הדינאמיות: (העובדות שמשתנות במהלך הרצת הקוד)

1. הנקודות על הלוח : (p((X,Y),Up,Down,Right,Left.  
   
 הנקודות נקבעו לפי המצב ההתחלתי של לוח המשחק, הן אלו שמייצגות את הלוח.

Up,Down,Right,Left:  
מייצגים מיקום הקווים ביחס לנק', כלומר, עובדות על הקו שמעל לנקודה, מתחת לנקודה, מימין לנקודה ומשמאל לנקודה.

מספר הטור בו נמצאת הנק'X: queue number-

מספר השורה בו נמצאת הנק'Y: line number-

אין במקום זה קו אפור, כלומר מקום זה מחוץ לתחום הלוחNo exist- :(-1)

יש במקום זה קו אדום או ירוק, כלומר, לחצו כבר על קו זה: Full- 1

יש במקום זה קו אפור, כלומר, לא לחצו עוד על קו זה0: empty-

לדוגמא: (p((1,1),-1,0,0,-1 – נק' זו נמצאת בשורה 1 ובטור 1.  
 מעליה ומשמאלה לא קיים קו, מתחתיה ומימינה קיים קו אפור.

הנקודות משתנות במהלך המשחק דרך שינוי איברי רשימת הנקודות.

רשימת הנקודות שעל הלוח : pointsList([])  
  
 ברשימה זו מופיעות עובדות כל הנקודות והיא משתנה בהתאם למהלכים במשחק.  
 השינוי הוא באיבריה, למשל- אם נלחץ קו כלשהו שהיה מעל לנקודה כלשהי,   
 אז בנקודה זו שמהווה איבר ברשימה, ה-UP הופך מ-0 ל-1.

איבר ברשימה זו יהיה מהצורה: (p((X,Y),Up,Down,Right,Left

(על תוכנו פורט בהסבר על עובדת הנקודות שעל הלוח)

2. רשימת השחקנים : players([])

דוגמה: [p(human,6,green),p(aiEasy,0,red)]

ברשימה זו מופיעים שני השחקנים במשחק, כאשר האיבר הראשון תמיד יהיה השחקן הנוכחי שתורו לשחק. כלומר, אחרי סיום כל תור, נקרא לחוק שיחליף בין מיקומי שני השחקנים ברשימה.

איבר ברשימה יראה מהצורה: p(PlayerType , Score, SignColor)

סוג השחקן: שחקן מס' 1 או 2PlayerType:   
מספר הנקודות הנוכחי שיש לשחקןScore:   
צבע השחקן: אדום או ירוקSignColor:

3. רשימת הריבועים שעל הלוח : squaresList([])

דוגמה: [sq(1,2,red),sq(2,2,green),sq(5,2,red)]

ברשימה זו מופיעים כל הריבועים שנוצרים על הלוח- של שני השחקנים יחדיו. בכל פעם שנסגר ריבוע, נוסף איבר לרשימה ובו צבע השחקן שסגר את הריבוע והמיקום של הנקודה דרכה נסגר הריבוע (שלחצו על קו מתחתיה/מעליה/מימינה/משמאלה ואז נסגר ריבוע).

איבר ברשימה הוא מהצורה: sq(X,Y,Color)

מספר הטור בו נמצאת הנק'X: queue number-

מספר השורה בו נמצאת הנק'Y: line number-

צבע השחקן שסגר את הריבוע :Color

סוגי נתונים מופשטים לייצוג הידע:

**רשימה של מארזים:**

\*את לוח המשחק הכולל את הקווים והנקודות (לא כולל הריבועים) ומשתנה מתור לתור, ייצגתי ברשימה של מארזים אשר כל מארז הוא נקודה על הלוח שמכילה: מארז (X,Y) של מיקום הנק' (הטור- X והשורה- Y בהם נמצאת הנקודה) ועובדות על הקו שמעל לנקודה, מתחת לנקודה, מימין לנקודה ומשמאל לנקודה. (כמו שהסברתי בתיעוד העובדה של הנקודות שעל הלוח)  
רשימת הנקודות ההתחלתית: [....,(p((2,1),-1,0,0,0,(p((1,1),-1,0,0,-1]  
עובדות על הנקודות משמען- הקווים שהם חלק מהלוח(כך ניתן לראות מהן גבולות הלוח), הקווים שעל הלוח שכבר לחצו עליהם והקווים הפנויים שעל הלוח נכון לאותו תור נוכחי.

ייצגתי את הנקודות ברשימה בכדיי שיהיה לי קל לשנות את העובדות על כל נקודה שלחצתי על קו הקשור אליה בתור מסוים, תוך נגישות למקום ברשימה בו נמצאת הנקודה. בנוסף, בכל פעם שאני משנה את העובדות על נקודה בלוח, אני מוחקת את כל הרשימה הקודמת שבזיכרון ומעדכנת רשימה חדשה ומעודכנת. לכן, רשימת הנקודות היא עובדה דינאמית ולא רגילה.

\*את הריבועים שעל לוח המשחק גם החלטתי לייצג באמצעות רשימה של מארזים, הנפרדת מהרשימה של הנקודות. כל מארז כולל את: מיקום הנקודה שדרך לחיצה על קו הקשור אליה נסגר ריבוע (הטור- X והשורה- Y בהם נמצאת הנקודה) ואת הצבע של השחקן שסגר את אותו ריבוע. (עם תחילת המשחק רשימת הריבועים ריקה)

דוגמה לרשימת ריבועים בתור נוכחי: [sq(1,2,red),sq(2,2,green),sq(5,2,red)]   
עם כל תור מופעל חוק הבודק אם היה שחקן שסגר ריבוע- אם כן, הריבוע נוסף לרשימת הריבועים, מתעדכנת הרשימה והרשימה הישנה נמחקת מזיכרון המחשב. לכן, בדומה לעובדה של רשימת הנקודות, גם זאת עובדה זו היא דינאמית.  
נוסף על כך, בשונה לרשימת הנקודות, ייצוג הריבועים חייב להיות על ידי רשימה, מאחר וכל פעם נוסף ריבוע חדש לרשימה (ע"י append) ואין כלים שדרכם נוכל להוסיף איבר למארז. לכן, ייצוג על ידי מארז של מארזים לא מתאים כאן.

\*המידע על שני השחקנים במשחק מיוצג גם כן על ידי רשימה של שני מארזים, אחד עבור כל שחקן. בכל מארז שמורים הנתונים על: סוג השחקן (שחקן מס' 1 או 2), מספר הנקודות הנוכחי שיש לשחקן וצבע השחקן (אדום או ירוק- במקרה של משחק בין שחקן אנושי למחשב: המחשב הוא הצבע אדום והשחקן האנושי מיוצג על ידי הצבע ירוק).  
לדוגמא, במצב של התחלת משחק בין שחקן אנושי למחשב ברמה הקלה הרשימה תראה כך: [p(aiEasy,0,red),p(human,0,green)]

בכל סיום תור, יתבצע חוק שיחליף בין איברי הרשימה על ידי הפעולה : reverse (כמובן שאם יש שחקן שסגר ריבוע אזי לפי חוקי המשחק הוא מקבל תור נוסף, במקרה כזה יופעל חוק שיעדכן את הנקודות של אותו שחקן, מבלי להחליף את מיקומי האיברים ברשימה), הצורך להחליף בכל פעם את תור המשתמשים דורש ייצוג רשימתי של השחקנים ולכן בחרתי בייצוג זה.

Assert **ו-** Retract**:**

אלו מתוארים מוכנים בתוכנת הפרולוג. מתוארים אלו מאפשרים להוסיף עובדות לזיכרון של המחשב בזמן הרצת התוכנית וגם למחוק עובדות קודמות מהזיכרון:  
: assert הפעולה שיוצרת עובדה חדשה עם זמן הרצת התוכנית.  
: retract הפעולה שמוחקת את העובדה הקודמת מהזיכרון.

השתמשתי בעבודה במתוארים אלה בכדיי שלא יהיה עומס על מערכת הפרולוג עקב זיכרון מלא ובכך התוכנית עלולה להיתקע ולרוץ לאט מדיי.   
בנוסף, אין צורך ברשימות של התורות הקודמים במשחק שהתבצעו לפני התור הנוכחי בו התעדכנו הרשימות, מאחר ובכל פעם שמעדכנים את הרשימה שומרים את הנתונים הקודמים הנחוצים.

(העובדות שמוספות לזיכרון ונמחקות מהזיכרון עם הרצת התוכנית, נקראות עובדות דינאמיות)

תיאור החוקים: (מלבד חוקי הבינה המלאכותית)

החוקים שפותחים את חלון המשחק :

1. התחלת המשחק : % start/0

מטרת החוק: זהו החוק המייצר מצב התחלתי של משחק ובונה את העובדות הדינאמיות הבסיסיות לתוכנית. זה גם החוק שכותבים במסך ההרצה (מסך השאילתות) בכדיי לפתוח את המשחק.   
  
הצגת החוק:

**start** : -  
 initGame,  
 openWindow.

הסבר החוק: start קורא לחוק- initGame, שבו נוצרות העובדות הדינאמיות של המשחק (עובדות אלו נוצרות עוד לפני שהמשחק התחיל), אחריו הוא קורא לחוק openWindow שפותח את חלון המשחק.

2. פתיחת חלון המשחק : 0/ openWindow%

מטרת החוק: החוק פותח את חלון המשחק ואת תפריט המשחק.

הצגת החוק:

**openWindow** - :  
 new(W,window('dots and boxes', size(900,800))),  
 menu(W).

הסבר החוק: openWindow פותח את חלון המשחק הריק שנקרא 'dots and boxes' ועליו ירוץ המשחק, לאחר מכן הוא קורא לחוק menu(W) שיוצר את התפריט הראשי של המשחק.

3. התפריט הראשי של המשחק : menu/1%

מטרת החוק: החוק שם את רכיבי התפריט (תמונות) על חלון המשחק והופך את כפתורי התפריט (התחלת המשחק, חוקי המשחק...) ללחיצים.

הצגת החוק:

**menu**(W):-  
 new(B,bitmap('reka.bmp')),  
 send(W,background,B),  
 new(HftH,bitmap('playvsfriend.bmp')), send(W,display,HftH,point(300,310)), new(HftC\_easy,bitmap('playvscomputer\_easy.bmp')),  
 send(W,display,HftC\_easy,point(210,410)),  
 new(HftC\_hard,bitmap('playvscomputer\_hard.bmp')),  
 send(W,display,HftC\_hard,point(210,510)),  
 new(R,bitmap('Rules.bmp')),  
 send(W,display,R,point(385,610)),  
 new(A,bitmap('About.bmp')),  
 send(W,display,A,point(379,705)),  
 send(W,open),  
 send(R,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,openRules,W)))), send(A,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,openAbout,W)))),  
 send(HftH,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,openPlay\_HftH,W)))), send(HftC\_easy,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,openPlay\_HftCEasy,W)))), send(HftC\_hard,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,openPlay\_HftCHard,W)))).

הסבר החוק: החוק שם את רקע התפריט לחלון המשחק ועליו, שם את רכיבי התפריט (תמונות) הכוללים: כפתור לתחילת משחק של שחקן אנושי מול שחקן אנושי, כפתור לתחילת משחק של שחקן אנושי מול מחשב ברמה הקלה, כפתור לתחילת משחק של שחקן אנושי מול מחשב ברמה הקשה, כפתור לחוקי המשחק וכפתור לאודות המשחק.  
החוק הופך את כל כפתורי התפריט ללחיצים, ועם לחיצה על כל אחד מהם מופעל חוק מתאים, אשר פותח את החלון המתאים ואם צריך- מבצע את הפעולות המתאימות.

4. חזרה לתפריט המשחק : 1/ back\_deleteOld%

מטרת החוק: החוק מופעל עם לחיצה על: 'back'. הכפתור 'back' מופיע בכל חלון שנפתח עם לחיצה על כפתור בתפריט הראשי.   
כפתור זה מאפשר לחזור לתפריט הראשי ולהתחיל משחק חדש וכל זה מתבצע ע"י החוק .back\_deleteOld

\*הכפתור 'back' נעשה לחיץ בכל חוק שמופעל עם לחיצה על כפתור מהתפריט הראשי. (לא אפרט בכל פעם שהחוק ישים את הכפתור 'back' כל המסך ויהפוך אותו ללחיץ)

הצגת החוק:

**back\_deleteOld**(W):-  
 send(W,clear),  
 initGame,  
 menu(W).

הסבר החוק: החוק back\_deleteOld(W) מנקה את החלון הנוכחי, כלומר, מוחק את כל התמונות שהושמו על חלון המשחק, מאתחל את העובדות הדינאמיות של המשחק ע"י קריאה לחוק: initGame ובסוף קורא לחוק התפריט הראשי שפותח את התפריט.

5. התחלת משחק לפי בחירת המשתמש :

**א.** שחקן אנושי מול שחקן אנושי:

1/ openPlay\_HftH%

מטרת החוק: החוק מופעל עם לחיצה על הכפתור: 'play Vs friend' שבתפריט הראשי. חוק זה פותח חלון מתאים ומאתחל משחק של שחקן אנושי מול שחקן אנושי.

הצגת החוק:

**openPlay\_HftH**(W):-  
 initPlayers(humanfthuman),  
 send(W,clear),  
 new(PR,bitmap('play\_reka\_cleen.bmp')),  
 send(W,background,PR),  
 new(D,bitmap('dotsNboxes.bmp')),  
 send(W,display,D,point(16,3)),  
 new(Back,bitmap('backTOmenu.gif')), send(W,display,Back,point(700,660)), send(Back,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,back\_deleteOld,W)))), addPointsToBored(W,1),  
 drawHLines(W,1),  
 drawVLines(W,1),  
 showPlayerTurn(W),  
 showScores(W),  
 squaresList(FullSquaresList),  
 showSquares(W,FullSquaresList),  
 send(W,open),  
 playMove(W).

**ב.** שחקן אנושי מול מחשב- ברמה הקלה:

1/ openPlay\_HftCEasy%

מטרת החוק: החוק מופעל עם לחיצה על הכפתור: 'play Vs computer – easy level' שבתפריט הראשי. חוק זה פותח חלון מתאים ומאתחל משחק של שחקן אנושי מול המחשב ברמה הקלה.

הצגת החוק:

**openPlay\_HftCEasy**(W):-  
 initPlayers(humanftcomputer\_easy),  
 send(W,clear),  
 new(PR,bitmap('play\_reka\_cleen.bmp')),  
 send(W,background,PR),  
 new(D,bitmap('dotsNboxes.bmp')),  
 send(W,display,D,point(16,3)),  
 new(Back,bitmap('backTOmenu.gif')), send(W,display,Back,point(700,660)), send(Back,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,back\_deleteOld,W)))), addPointsToBored(W,1),  
 drawHLines(W,1),  
 drawVLines(W,1),  
 showPlayerTurn(W),  
 showScores(W),  
 squaresList(FullSquaresList),  
 showSquares(W,FullSquaresList),  
 send(W,open),  
 playMove(W).

**ג.** שחקן אנושי מול מחשב- ברמה הקשה:

1/ openPlay\_HftCHard%

מטרת החוק: החוק מופעל עם לחיצה על הכפתור: 'play Vs computer – hard level' שבתפריט הראשי. חוק זה פותח חלון מתאים ומאתחל משחק של שחקן אנושי מול המחשב ברמה הקשה.

הצגת החוק:

**openPlay\_HftCHard** (W):-  
 initPlayers(humanftcomputer\_hard),  
 send(W,clear),  
 new(PR,bitmap('play\_reka\_cleen.bmp')),  
 send(W,background,PR),  
 new(D,bitmap('dotsNboxes.bmp')),  
 send(W,display,D,point(16,3)),  
 new(Back,bitmap('backTOmenu.gif')), send(W,display,Back,point(700,660)), send(Back,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,back\_deleteOld,W)))), addPointsToBored(W,1),  
 drawHLines(W,1),  
 drawVLines(W,1),  
 showPlayerTurn(W),  
 showScores(W),  
 squaresList(FullSquaresList),  
 showSquares(W,FullSquaresList),  
 send(W,open),  
 playMove(W).

הסבר החוקים א',ב' ו-ג': כול חוקים אלו קוראים לחוק initPlayers(X), כאשר X הוא סוג המשחק שנבחר, שמעדכן את העובדה הדינאמית של רשימת השחקנים בהתאם לסוג המשחק שנבחר,   
לאחר מכן, פותחים חלון חדש עם לוח המשחק:   
1. קוראים לחוק addPointsToBored(W) שמוסיף נקודות ללוח המשחק  
2. קוראים לחוקים drawHLines(W,1) ו- drawVLines(W,1) שמוסיפים את הקווים הלחיצים ללוח המשחק.   
3. קוראים לחוקים: showPlayerTurn(W) ו- showScores(W) שמראים על חלון המשחק את הנקודות ההתחלתיות כל שחקן ואת שם השחקן שתורו לשחק ראשון.  
4. קוראים לחוק showSquares(W,FullSquaresList) אשר מקבל את רשימת הריבועים (שנסגרו) הנוכחית, ושם את הריבועים שברשימה במקום המתאים על הלוח, אם הרשימה ריקה- הוא לא עושה כלום.  
ולבסוף, קוראים לחוק playMove(W) אשר בבחירה במשחק של שחקן אנושי מול המחשב, מבצע את תור המחשב ובבחירה במשחק של שחקן אנושי מול שחקן אנושי- לא עושה כלום.

6. פתיחת החוקים : 1/ openRules%

מטרת החוק: החוק מופעל עם לחיצה על הכפתור: 'Rules' שבתפריט הראשי. חוק זה פותח חלון עם חוקי המשחק.

הצגת החוק:

**openRules**(W):-  
 send(W,clear),  
 new(RR,bitmap('rulesreka.bmp')),  
 send(W,background,RR),  
 send(W,open),  
 new(Back,bitmap('backTOmenu.gif')), send(W,display,Back,point(705,660)), send(Back,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,back\_deleteOld,W)))).

הסבר החוק: החוק מנקה את החלון הנוכחי, כלומר, מוחק את כל התמונות שהושמו על חלון המשחק, ושם על החלון רקע חדש עם חוקי המשחק.

7. פתיחת האודות : 1/ openAbout%

מטרת החוק: החוק מופעל עם לחיצה על הכפתור: 'About' שבתפריט הראשי. חוק זה פותח חלון עם אודות המשחק.

הצגת החוק:

**openAbout**(W):-  
 send(W,clear),  
 new(AR,bitmap('aboutreka.bmp')),  
 send(W,background,AR),  
 send(W,open),  
 new(Back,bitmap('backTOmenu.gif')), send(W,display,Back,point(705,660)), send(Back,recogniser,click\_gesture(left,'',single,  
 and(message(@prolog,back\_deleteOld,W)))).

הסבר החוק: החוק מנקה את החלון הנוכחי, כלומר, מוחק את כל התמונות שהושמו על חלון המשחק, ושם על החלון רקע חדש עם אודות המשחק.

החוקים שמאתחלים את העובדות הדינאמיות של המשחק :

\*אתחול הרשימות הדינאמיות: % initGame/0

מטרת החוק: חוק זה מופעל עם פתיחה/חזרה לתפריט הראשי ותחילת משחק חדש.   
החוק מאתחל את כל הרשימות הדינאמיות של המשחק, כלומר יוצר אותן ריקות בתוכנן בשביל המצב ההתחלתי של המשחק.

הצגת החוק:

**initGame**:-  
 retractall(pointsList(\_)),  
 retractall(players(\_)),  
 retractall(squaresList(\_)),  
 assert(squaresList([])),  
 assert(players([])),  
 assert(pointsList([])),  
 listPoints.

הסבר החוק: ראשית, החוק משתמש בפעולה: retractallשמוחקת את כל העובדות הדינאמיות הקודמות ששמורות בזיכרון מהמשחק הקודם (אם היה כזה), כלומר את הרשימות הדינאמיות: pointsList, players ו- squaresList . לאחר מכן החוק יוצר את אותן הרשימות הדינאמיות אך חדשות ומאותחלות, וזאת בעזרת הפעולה: assert. לבסוף החוק מזמן את החוק: listPoints, שמכניס לרשימת הנקודות את עובדות הנקודות של הלוח: p((X,Y),Up,Down,Right,Left), שנכתבו מראש.

מילוי רשימת הנקודות: % listPoints/0

מטרת החוק: חוק זה ממלא את רשימת הנקודות ההתחלתית של המשחק בעובדות הנקודות שנכתבו מראש.

הצגת החוק:

**listPoints**:-  
 findall(p((X,Y),Up,Down,Right,Left),p((X,Y),Up,Down,Right,Left),L), retract(pointsList(\_)),  
 assert(pointsList(L)).

הסבר החוק: חוק זה מכניס לרשימה אחת, בעזרת הפעולה findall, את עובדות כל הנקודות שנכתבו מראש לפי גבולות הלוח ההתחלתי של המשחק. לאחר מכן, החוק מוחק את הרשימה הריקה שאותחלה בחוק: initGame ויוצר רשימה דינאמית חדשה שתוכנה כתוכן הרשימה הנ"ל.

מילוי רשימת השחקנים: % initPlayers/1

מטרת החוק: חוק זה מופעל עם לחיצה על אחד משלושת הכפתורים לתחילת משחק בתפריט הראשי. חוק זה ממלא את רשימת השחקנים ההתחלתית של המשחק בהתאם לסוג המשחק שהמשתמש בחר.

הצגת החוק:

**initPlayers**(humanfthuman):-  
 retract(players(\_)), append([p(human,0,red)],[p(human,0,green)],PlayersList), assert(players(PlayersList)).

**initPlayers**(humanftcomputer\_easy):-  
 retract(players(\_)), append([p(aiEasy,0,red)],[p(human,0,green)],PlayersList), assert(players(PlayersList)).

**initPlayers**(humanftcomputer\_hard):-  
 retract(players(\_)), append([p(aiHard,0,red)],[p(human,0,green)],PlayersList), assert(players(PlayersList)).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר  **initPlayers**–   
החוק מקבל את סוג המשחק שהמשתמש בחר- יש 3 סוגים ולכן יש גם 3 חוקים חילופיים המתאימים לכל סוג משחק. החוק מוחק את רשימת השחקנים הריקה שאותחלה בחוק initGame ויוצר רשימה דינאמית חדשה שתוכנה מהצורה:   
[p(PlayerType1,Score1,SignColor1),p(PlayerType2,Score2,SignColor2)] , כאשר:  
סוג המשתמש הראשון יבחר לפי סוג המשחק שהמשתמש בחר לשחק: במשחק קל מול מחשב: aiEasy, במשחק קשה מול מחשב: aiHard ובמשחק מול שחקן אנושי: human,  
צבע המחשב תמיד יהיה אדום וצבע השחקן האנושי ירוק,  
והנקודות של כל שחקן יאותחלו בהתחלה ל-0.

החוקים שמציגים על לוח המשחק את הנתונים השמורים בעובדות :

1. הצגת תור השחקן הנוכחי: % showPlayerTurn/1

מטרת החוק: בהתחלה, חוק זה פועל עם לחיצה על כפתור של 'תחילת משחק' בתפריט הראשי, במהלך המשחק, החוק מופעל בכל פעם ששחקן כלשהו לוחץ על קו.  
חוק זה מציג על לוח המשחק את שם השחקן הנוכחי (player1/player2) שתורו לשחק.

הצגת החוק:

**showPlayerTurn**(W):-  
 players(PlayersList),  
 nth1(1,PlayersList,p(\_,\_,Sign)),  
 showPlayerTurn1(W,Sign).

הסבר החוק: החוק בודק מיהו השחקן שנמצא במקום הראשון ברשימת השחקנים (האיבר הראשון ברשימה הוא השחקן שתורו הנוכחי לשחק). לאחר מכן, הוא שולח את סוג שחקן זה (צבע השחקן- אדום/ירוק) לחוק:   
, showPlayerTurn1(W,Sign)  
שמציג על לוח המשחק את התמונה המתאימה של שם השחקן שתורו לשחק.

שיבוץ התמונה של שם השחקן שתורו לשחק: % showPlayerTurn1/2

מטרת החוק: החוק שם על הלוח את התמונה של שם השחקן שתורו לשחק.

הצגת החוק:

**showPlayerTurn1(**W,red):-  
 new(Ba,bitmap('player1red.bmp')),  
 send(W,display,Ba,point(435,76)).

**showPlayerTurn1**(W,green):-  
 new(Ba,bitmap('player2green.bmp')), send(W,display,Ba,point(435,76)).

**- showPlayerTurn1**הסבר החוק: אלו הם שני חוקים חילופיים למתאר: החוק שם על לוח המשחק במיקום המתאים, את התמונה המתאימה לצבע השחקן שקיבל, לכן יש שני חוקים חילופיים: אחד עבור צבע השחקן הראשון ואחד עבור צבע השחקן השני.

2. הצגת מס' הנקודות של כל שחקן בתור הנוכחי: % showScores/1

מטרת החוק: בהתחלה, חוק זה פועל עם לחיצה על כפתור של 'תחילת משחק' בתפריט הראשי, במהלך המשחק, החוק מופעל בכל פעם ששחקן כלשהו לוחץ על קו.  
חוק זה מציג על לוח המשחק את מספר הנקודות שיש לכל שחקן בכל תור במשחק.

הצגת החוק:

**showScores**(W):-  
 players(PlayersList),  
 writeln(PlayersList),  
 member(p(\_,ScoreRed,red),PlayersList),  
 member(p(\_,ScoreGreen,green),PlayersList), showScoresBoxesRed(W,ScoreRed),  
 showScoresBoxesGreen(W,ScoreGreen),  
 squaresList(SquaresList),  
 writeln(SquaresList).

הסבר החוק: החוק שולח לחוק: showScoresBoxesRed(W,ScoreRed) ולחוק-  
 showScoresBoxesGreen(W,ScoreGreen) את מספר הנקודות של השחקן האדום ברשימת השחקנים ואת מספר הנקודות של השחקן הירוק בהתאמה. חוקים אלו מציגים על לוח המשחק את התמונות המתאימות למספר הנקודות הנוכחי של כל שחקן.  
לאחר מכן, החוק מדפיס את רשימת הריבועים על מסך השאילתות.

שיבוץ תמונת מספר הנקודות של השחקן האדום:   
 % showScoresBoxesRed/2  
הצגת החוק:

**showScoresBoxesRed**(W,ScoreRed):-  
 scoreRed(ScoreRed,Pic),  
 new(S,bitmap(Pic)),  
 send(W,display,S,point(665,300)).

שיבוץ תמונת מספר הנקודות של השחקן הירוק:  
 % showScoresBoxesGreen/2  
הצגת החוק:

**showScoresBoxesGreen**(W,ScoreGreen):-  
 scoreGreen(ScoreGreen,Pic),  
 new(S,bitmap(Pic)),  
 send(W,display,S,point(665,375)).

מטרת החוקים: החוקים שמים על הלוח את התמונות המתאימות למספר הנקודות שיש לכל שחקן.

הסבר החוקים: החוקים שמים על לוח המשחק במיקום המתאים, את התמונה המתאימה למספר הנקודות של סוג השחקן אשר קיבל החוק. זאת הם עושים בעזרת קריאה עם scoreRed(ScoreRed,Pic) או scoreGreen(ScoreGreen,Pic)מספר הנקודות לעובדות:   
אשר נקבעו בתחילת המשחק, ובהן שמורה התמונה המתאימה למספר הנקודות של כל שחקן, ואותה מציגים על הלוח.

3. הצגת הריבועים שסגרו השחקנים: % showSquares /2

מטרת החוק: בהתחלה, חוק זה פועל עם לחיצה על כפתור של 'תחילת משחק' בתפריט הראשי, במהלך המשחק, החוק מופעל בכל פעם ששחקן כלשהו לוחץ על קו.  
במקרה שמשתמש סגר ריבוע בתורו, חוק זה שם על לוח המשחק את אותו הריבוע בצבע השחקן שסגר אותו ובמיקום המתאים. כך, השחקנים רואים בבירור למי יש יותר ריבועים ואיפה כבר נסגר ריבוע. בנוסף, החוק בודק בכל תור נוכחי אם המשחק הסתיים.

הצגת החוק:

**showSquares**(\_,[]).  
  
**showSquares**(W,[sq(X,Y,Sign)|RestOfSquares]):-  
 showSingleSquare(W,X,Y,Sign),  
 showSquares(W,RestOfSquares),  
 endGame(W).

הסבר החוק: חוק זה הוא חוק רקורסיבי, לכן אלו הם שני חוקים חילופיים למתאר-**showSquares**: אחד מהווה את תנאי העצירה, והשני מבצע את מטרת החוק.   
  
החוק הראשון: חוק זה פועל אם השחקן לא סגר בתור הנוכחי שום ריבוע ואז אין צורך להציג ריבוע על לוח המשחק. בנוסף, זהו תנאי העצירה לחוק השני, כלומר, אם כבר הוצגו כל הריבועים שסגר המשתמש על הלוח ורשימת הריבועים התרוקנה, אזי הרקורסיה מפסיקה לפעול.  
החוק השני: קורא לחוק: showSingleSquare(W,X,Y,Sign) עם מיקום אחד מהריבועים הנוכחיים שנסגרו ועם צבע השחקן שסגר את הריבוע. את מידע זה החוק שואב מהאיבר הראשון ברשימת הריבועים שנשלחה אליו. החוק קורא לעצמו שוב, עם שאר איברי הרשימה (מלבד האיבר הראשון, אותו הפרדנו מהרשימה בעזרת הקו המפריד: | ). אם יש עוד איברים ברשימה, כלומר אם השחקן סגר בתור הנוכחי יותר מריבוע אחד, אזי החוק מתבצע שוב. אחרת, כלומר אם הרשימה ריקה, החוק אינו עושה דבר. חוק זה מתבצע עד שהוצגו כל הריבועים שסגר השחקן בתור הנוכחי. בסיום הרקורסיה וביצוע כל השורות בחוק, מלבד השורה האחרונה, החוק בודק אם בתור זה המשחק הסתיים. זאת הוא עושה בעזרת קריאה לחוק: endGame(W).

שיבוץ תמונת הריבוע שסגר השחקן: % showSingleSquare/4

מטרת החוק: החוק שם על הלוח את תמונת הריבוע שנסגר במיקום המתאים ובצבע השחקן המתאים (צבע השחקן שסגר את הריבוע).

הצגת החוק:

**showSingleSquare**(W,X,Y,red):-  
 new(Xg,bitmap('Xred.bmp')),  
 LocX is 188 + (X-1) \* 75,  
 LocY is 258 + (Y-1) \* 75,  
 send(W,display,Xg,point(LocX,LocY)).

**showSingleSquare**(W,X,Y,green):-  
 new(Xr,bitmap('Xgreen.bmp')),  
 LocX is 188 + (X-1) \* 75,  
 LocY is 258 + (Y-1) \* 75,  
 send(W,display,Xr,point(LocX,LocY)).

הסבר החוק: אלו הם שני חוקים חילופיים למתאר **showSingleSquare**:   
חוק אחד שם ריבוע אדום, במקרה שהשחקן האדום הוא זה שסגר את הריבוע והחוק האחר שם ריבוע ירוק, במקרה שהשחקן הירוק הוא זה שסגר את הריבוע.  
סדר כתיבת החוקים אינו משנה. מה שיקבע אם החוק הראשון או החוק השני יפעל, יהיה צבע השחקן שישלח אל החוק. החוק מקבל בנוסף את מיקום הריבוע ובכך הוא יודע איפה למקם את תמונת הריבוע המתאימה בעזרת הנוסחה לאיבר בסדרה חשבונית, כאשר הפרשי מיקומי הריבועים קבועים והם 75 והאיברים הראשונים של סדרת הטור וסדרת השורה מהווים את מיקום הנקודה על הלוח: (188,258), בה יהיה צריך להיות מודבק הריבוע שבקצה העליון השמאלי אשר ממנו מתחיל הלוח.

4. סיום משחק: % endGame /1

מטרת החוק: חוק זה פועל בכל פעם שהחוק showSquares מופעל ורשימת הריבועים אינה ריקה. החוק בודק אם נגמר המשחק בתור זה (בו שחקן כלשהו סגר ריבוע), אם כן- מציג את שם המנצח על חלון המשחק ואם לא- לא עושה דבר.

הצגת החוק:

**endGame**(\_):-  
 totalScore(Total),  
 Total \= 25.

**endGame**(W):-  
 totalScore(Total),  
 Total is 25,  
 winner(W).

הסבר החוק: אלו הם שני חוקים חילופיים למתאר  **endGame**:  
שני החוקים קוראים לחוק: totalScore(Total) אשר מחזיר להם את מספר הנקודות הכולל של שני השחקנים ביחד. אם מספר זה שווה ל-25 אזי נגמר המשחק, מאחר ונסגרו כל הריבועים על הלוח. במקרה זה, מתבצע החוק השני והוא מזמן את החוק: winner(W)אשר מדפיס על חלון המשחק את שם המנצח (player 1/2). במקרה האחר, בו מספר הנקודות הכולל של שני השחקנים יחדיו אינו 25, החוק הראשון מתבצע והוא לא עושה דבר.

שיבוץ תמונת שם השחקן שניצח : % winner/1

מטרת החוק: החוק שם על הלוח תמונה עם שם השחקן שניצח במשחק זה. (player 1/2)

הצגת החוק:

**winner**(W):-  
 players(PL),  
 nth1(1,PL,p(\_,S1,green)),  
 nth1(2,PL,p(\_,S2,red)),  
 S1>S2,  
 new(D,bitmap('player2Win.bmp')),  
 send(W,display,D,point(237,155)).

**winner**(W):-  
 players(PL),  
 nth1(1,PL,p(\_,S1,red)),  
 nth1(2,PL,p(\_,S2,green)),  
 S1>S2,  
 new(D,bitmap('player1Win.bmp')),  
 send(W,display,D,point(237,155)).

**winner**(W):-  
 players(PL),  
 nth1(1,PL,p(\_,S1,green)),  
 nth1(2,PL,p(\_,S2,red)),  
 S2>S1,  
 new(D,bitmap('player1Win.bmp')),  
 send(W,display,D,point(237,155)).

**winner**(W):-  
 players(PL),  
 nth1(1,PL,p(\_,S1,red)),  
 nth1(2,PL,p(\_,S2,green)),  
 S2>S1,  
 new(D,bitmap('player2Win.bmp')),  
 send(W,display,D,point(237,155)).

הסבר החוק: אלו הם ארבעה חוקים חילופיים למתאר  **winner**: החוקים שמים על לוח המשחק במיקום המתאים את התמונה המתאימה לשם השחקן שניצח במשחק, כלומר זה שיש לו יותר נקודות. זאת הם עושים בעזרת קריאה לרשימה players(PL) שבה יש 2 איברים(כמספר השחקנים) ובכל איבר יש את מספר הנקודות של כל שחקן. החוקים בודקים לאיזה שחקן יש מספר נקודות גבוהה יותר ואת שמו הם מציגים על חלון המשחק.  
כל חוק מבין ארבעת החוקים בודק מצב שונה (נוצרים ארבעה מצבים שונים עקב מיקום השחקנים ברשימה שמשתנה כל תור) :

החוק הראשון: השחקן שניצח במשחק נמצא במקום הראשון ברשימה והוא השחקן הירוק.  
החוק השני: השחקן שניצח במשחק נמצא במקום הראשון ברשימה והוא השחקן האדום.  
החוק השלישי: השחקן שניצח במשחק נמצא במקום השני ברשימה והוא השחקן האדום.  
החוק הרביעי: השחקן שניצח במשחק נמצא במקום השני ברשימה והוא השחקן הירוק.

החוקים שמציירים/בונים את לוח המשחק :

1. הוספת הנקודות ללוח המשחק: % addPointsToBored/2

מטרת החוק: חוק זה פועל עם לחיצה על כפתור של 'תחילת משחק' בתפריט הראשי.  
חוק זה מצייר על חלון המשחק את הנקודות הבונות את לוח המשחק.

הצגת החוק:

**addPointsToBored**(\_,7).

**addPointsToBored**(W,Line):-  
 putPoints(W,1,Line),   
 NewLine is Line + 1,  
 addPointsToBored(W,NewLine).

הסבר החוק: חוק זה הוא חוק רקורסיבי שעובר על כל השורות בלוח המשחק ושם בכל שורה את מספר הנקודות הדרוש ברווחים שווים, לכן אלו הם שני חוקים חילופיים למתאר- **addPointsToBored**: הראשון מהווה את תנאי העצירה של הרקורסיה והשני שם את הנקודות במיקום המתאים.

החוק הראשון: מצב העצירה- הגענו לשורה מספר 7 בלוח המשחק שמחוץ לגבולות הלוח, לכן החוק לא עושה דבר.  
החוק השני: קורא לחוק putPoints(W,1,Line): חוק רקורסיבי שמקבל את ערכי X ו-Y של נקודה על הלוח כאשר X=1. בכך, החוק מקבל את כל נקודות הטור הראשון ורץ על כל טורי לוח המשחק עד שמגיע לטור השביעי שמחוץ לגבולות הלוח.   
החוק putPoints שולף את הנקודה המתאימה מרשימת הנקודות, קורא לחוק drawPoint(W,X,Y) ששם את הנקודה במיקום המתאים על הלוח וקורא לעצמו שוב עם השורה הבאה על הלוח (עד שמגיע לשורה השביעית ואז מתבצע החוק ה-1 ונגמרת הרקורסיה).

2. הצגת הקווים האופקיים: % drawHLines/2

מטרת החוק: בהתחלה, חוק זה פועל עם לחיצה על כפתור של 'תחילת משחק' בתפריט הראשי במטרה לשים במיקום המתאים על לוח המשחק את הקווים האופקיים האפורים שבונים את לוח המשחק ועושה אותם לחיצים- כאלו שניתן ללחוץ עליהם ועם לחיצה עליהם יפעל החוק: hLinePressed, אשר דואג לעדכון כל נתוני המשחק ולביצוע הרקורסיה של המשחק כולו.  
לאחר בניית הלוח ההתחלתי של המשחק, החוק מופעל שוב דרך החוק: hLinePressed והוא דואג שכאשר שחקן לחץ על קו אופקי כלשהו, קו זה יהפוך מצבע אפור לצבע השחקן שלחץ עליו ולא יעשה אותו קו לחיץ, כלומר לא יהיה ניתן ללחוץ על קו זה שוב כי כבר לחצו עליו.

הצגת החוק:

**drawHLines**(\_,7)**.**

**drawHLines**(W,Line):-  
 putHorizontaLines(W,1,Line),  
 NewLine is Line + 1,  
drawHLines(W,NewLine).

הסבר החוק: חוק זה הוא חוק רקורסיבי שעובר על כל השורות בלוח המשחק ושם בכל שורה את מספר הקווים הדרוש ברווחים שווים, לכן אלו הם שני חוקים חילופיים למתאר- **drawHLines**: הראשון מהווה את תנאי העצירה של הרקורסיה והשני שם את הקווים במיקום המתאים על הלוח.

החוק הראשון: מצב העצירה- הגענו לשורה מספר 7 שמחוץ לגבולות הלוח, לכן החוק לא עושה דבר.  
החוק השני: קורא לחוק putHorizontalLines(W,1,Line): חוק רקורסיבי שמקבל את ערכי X ו-Y של נקודה על הלוח כאשר X=1. בכך, החוק מקבל את כל נקודות הטור הראשון שמימינם יש לשים קו אפור. החוק רץ על כל טורי לוח המשחק עד שמגיע לטור השישי, אשר אם נשים קו מימין לנקודה בטור זה, הוא יהיה מחוץ לגבולות הלוח.   
החוק putHorizontalLines שולף את הנקודה המתאימה מרשימת הנקודות ו"שולח" אותה ואת מצבה הנוכחי לחוק drawHLine(W,X,Y,LineStatus) שכאשר עוד לא לחצו על הקו(המצב הנוכחי הוא 0), הוא שם קו אפור במיקום המתאים על הלוח וכאשר שחקן לחץ על קו כלשהו(המצב הנוכחי הוא 1), הוא שם קו בצבע השחקן הראשון ברשימת השחקנים שכרגע ביצע את התור, זאת הוא עושה בעזרת שליחת הצבע לחוק החילופי: (drawHLineColor(W,X,Y,1 ,SignColor ששם את תמונת הקו המתאים. (לכן יש שני חוקים חילופיים למתאר **– drawHLineColor** אחד לצבע האדום והאחר לצבע הירוק).  
לאחר שהחוק: putHorizontalLines(W,1,Line) הסתיים, החוק: (drawHLines(W,Line קורא לעצמו שוב עם השורה הבאה על הלוח (עד שמגיע לשורה השביעית ואז מתבצע החוק ה-1 ונגמרת הרקורסיה).

3. הצגת הקווים האנכיים: % drawVLines/2

מטרת החוק: אותה מטרה כמו של החוק: drawHLines, רק שחוק זה שם את הקווים האנכיים ולא האופקיים ועם לחיצה עליהם יפעל החוק: vLinePressed ולא- hLinePressed. בנוסף, החוק יופעל שוב במהלך המשחק על ידי החוק: vLinePressed ולא על ידי- hLinePressed מאחר והשחקן ילחץ על קו אנכי ולא אופרי.

הצגת החוק:

**drawVLines**(\_,6)**.**

**drawVLines**(W,Line):-  
 putVerticalLines(W,1,Line),  
 NewLine is Line + 1,  
drawVLines(W,NewLine).

הסבר החוק: חוק זה הוא חוק רקורסיבי שעובר על כל השורות בלוח המשחק ושם בכל שורה את מספר הקווים הדרוש ברווחים שווים, לכן אלו הם שני חוקים חילופיים למתאר- **drawVLines**: הראשון מהווה את תנאי העצירה של הרקורסיה והשני שם את הקווים במיקום המתאים על הלוח.

החוק הראשון: מצב העצירה- הגענו לשורה מספר 6 בלוח המשחק שהקו מתחתיה מחוץ לגבולות הלוח, לכן החוק לא עושה דבר.  
החוק השני: קורא לחוק putVerticalLines (W,1,Line): חוק רקורסיבי שמקבל את ערכי X ו-Y של נקודה על הלוח כאשר X=1. בכך, החוק מקבל את כל נקודות הטור הראשון שמתחתם יש לשים קו אפור. החוק רץ על כל טורי לוח המשחק עד שמגיע לטור השביעי שמחוץ לגבולות הלוח.   
החוק putVerticalLines שולף את הנקודה המתאימה מרשימת הנקודות ו"שולח" אותה ואת מצבה הנוכחי לחוק drawVLine(W,X,Y,LineStatus) שכאשר עוד לא לחצו על הקו(המצב הנוכחי הוא 0), הוא שם קו אפור במיקום המתאים על הלוח וכאשר שחקן לחץ על קו כלשהו(המצב הנוכחי הוא 1), הוא שם קו בצבע השחקן הראשון ברשימת השחקנים שכרגע ביצע את התור, זאת הוא עושה בעזרת שליחת הצבע לחוק החילופי: (drawVLineColor(W,X,Y,1 ,SignColor ששם את תמונת הקו המתאים. (לכן יש שני חוקים חילופיים למתאר **– drawVLineColor** אחד לצבע האדום והאחר לצבע הירוק).  
לאחר שהחוק: putVerticalLines(W,1,Line) הסתיים, החוק: (drawVLines(W,Line קורא לעצמו שוב עם השורה הבאה שעל הלוח (עד שמגיע לשורה השישית ואז מתבצע החוק ה-1 ונגמרת הרקורסיה).

4. עדכון הנתונים והרקורסיה של המשחק- החוק מתבצע לאחר לחיצה על קו:

\*לאחר לחיצה על קו אופקי: % hLinePressed /3

מטרת החוק: חוק זה מופעל בכל פעם ששחקן לוחץ על קו אופקי אפור שעל הלוח.  
חוק זה מעדכן את הנתונים של המשחק בעקבות המהלך של אותו שחקן שביצע את התור וממשיך את הרקורסיה של המשחק.

הצגת החוק:

**hLinePressed**(W,X,Y):-  
 %check: append([X],[Y],Z), writeln(Z), % pointsList(PointsList), member(p((X,Y),Up,Down,0,Left),PointsList), XPlus1 is X+1, member(p((XPlus1,Y),Up1,Down1,Right1,0),PointsList), select(p((X,Y),Up,Down,0,Left),PointsList,NL), select(p((XPlus1,Y),Up1,Down1,Right1,0),NL,NL2), append([p((X,Y),Up,Down,1,Left)],NL2,NL3), append([p((XPlus1,Y),Up1,Down1,Right1,1)], NL3,NewPointsList), retract(pointsList(\_)), assert(pointsList(NewPointsList)), member(p((X,Y),\_,\_,LineStatus,\_),NewPointsList), drawHLine(W,X,Y,LineStatus), totalScoreForHorizontalLine(X,Y,Score), playTurn(Score), showPlayerTurn(W), showScores(W), squaresList(FullSquaresList), showSquares(W,FullSquaresList), playMove(W).

הסבר החוק: חוק זה מעדכן את עובדת רשימת הנקודות בעקבות הלחיצה על קו אופקי, כאשר הוא משנה את איבר הנקודה שלחצו על הקו האופקי שמימינה, כלומר הוא משנה את המשתנה שמייצג את הקו הימיני מ-0 ל-1 ואת איבר הנקודה שלחצו על הקו האופקי שמשמאלה, כלומר הוא משנה את המשתנה שמייצג את הקו השמאלי מ-0 ל-1.   
לאחר מכן, החוק מזמן את החוק:drawHLine(W,X,Y,LineStatus)עם המצב החדש של הנקודה שקיבל, כלומר LineStatus=1, והחוק drawHLineמחליף את הקו האפור שעל הלוח במיקום המתאים בקו שצבוע בצבע השחקן שלחץ על אותו הקו.  
אחרי זה החוק קורא לחוק: totalScoreForHorizontalLine(X,Y,Score), אשר מעדכן את רשימת הריבועים (במידה ועם לחיצה על קו זה נסגר ריבוע) ומחזיר את מספר הנקודות שהשחקן שלחץ על הקו האופקי צבר באותו תור. החוק hLinePressed שולח את מספר נקודות זה לחוק:playTurn(Score), שבמקרה שהשחקן לא סגר ריבוע ומספר הנקודות הוא 0: הוא מחליף את סדר איברי רשימת השחקנים (מאחר והשחקן הראשון הוא זה שתורו לשחק ועכשיו עובר התור לשחקן השני), ובמקרה שהשחקן כן סגר ריבוע וצבר נקודות: הוא מעדכן את מספר הנקודות של אותו שחקן ברשימת השחקנים.בנוסף, החוק קורא לחוקים: showPlayerTurn(W), showScores(W) והחוק: squaresList(FullSquaresList) עם רשימת הריבועים. חוקים אלו מציגים את הנתונים העדכניים על מסך המשחק.לבסוף, החוק קורא לחוק:playMove(W)אשר במידת הצורך דואג לביצוע תור המחשב, אחרת (אם זה תור השחקן האנושי) לא מבצע דבר, כלומר חוק זה ממשיך את הרקורסיה של המשחק.

\*לאחר לחיצה על קו אנכי: % vLinePressed /3

מטרת החוק: חוק זה מופעל בכל פעם ששחקן לוחץ על קו אנכי אפור שעל הלוח.  
חוק זה מעדכן את הנתונים של המשחק בעקבות המהלך של אותו שחקן שביצע את התור וממשיך את הרקורסיה של המשחק.

הצגת החוק:

**vLinePressed**(W,X,Y):-  
 %check: append([X],[Y],Z), writeln(Z), % pointsList(PointsList), member(p((X,Y),Up,0,Right,Left),PointsList), YPlus1 is Y+1,  
 member(p((X,YPlus1),0,Down1,Right1,Left1),PointsList), select(p((X,Y),Up,0,Right,Left),PointsList,NL), select(p((X,YPlus1),0,Down1,Right1,Left1),NL,NL2), append([p((X,Y),Up,1,Right,Left)],NL2,NL3), append([p((X,YPlus1),1,Down1,Right1,Left1)],  
 NL3,NewPointsList),  
 retract(pointsList(\_)), assert(pointsList(NewPointsList)),   
 member(p((X,Y),\_, LineStatus,\_ ,\_),NewPointsList),   
 drawVLine(W,X,Y,LineStatus), totalScoreFor Vertical Line(X,Y,Score), playTurn(Score), showPlayerTurn(W), showScores(W), squaresList(FullSquaresList), showSquares(W,FullSquaresList), playMove(W).

הסבר החוק: חוק זה מעדכן את עובדת רשימת הנקודות בעקבות הלחיצה על קו אנכי, כאשר הוא משנה את איבר הנקודה שלחצו על הקו האנכי שמתחתיה, כלומר הוא משנה את המשתנה שמייצג את הקו שמתחת מ-0 ל-1 ואת איבר הנקודה שלחצו על הקו האנכי שמעלייה, כלומר הוא משנה את המשתנה שמייצג את הקו שמעלייה מ-0 ל-1.   
לאחר מכן, החוק מזמן את החוק:drawVLine(W,X,Y,LineStatus)עם המצב החדש של הנקודה שקיבל, כלומר LineStatus=1, והחוק drawVLineמחליף את הקו האפור שעל הלוח במיקום המתאים בקו שצבוע בצבע השחקן שלחץ על אותו הקו.  
אחרי זה החוק קורא לחוק: totalScoreForVerticalLine(X,Y,Score), אשר מעדכן את רשימת הריבועים (במידה ועם לחיצה על קו זה נסגר ריבוע) בעזרת החוק: addToFullSquareList(X,Y,Score) שמקבל את מס' הנקודות (הריבועים) שנבדקות בעזרת החוק checkFullSquareAtXYasTopLeftCorner(X,Y,Score). לבסוף החוק totalScoreForVerticalLine מחזיר את מספר הנקודות שהשחקן שלחץ על הקו האנכי צבר באותו תור והחוק vLinePressed שולח את מספר נקודות זה לחוק:- playTurn(Score) שני חוקים חילופיים למתאר **playTurn**, שבמקרה שהשחקן לא סגר ריבוע ומספר הנקודות הוא 0: מתבצע החוק הראשון שמחליף את סדר איברי רשימת השחקנים (מאחר והשחקן הראשון הוא זה שתורו לשחק ועכשיו עובר התור לשחקן השני), ובמקרה שהשחקן כן סגר ריבוע וצבר נקודות: מתבצע החוק השני שמעדכן את מספר הנקודות של אותו שחקן ברשימת השחקנים (מבלי לשנות את סדר השחקנים).בנוסף, החוק קורא לחוקים: showPlayerTurn(W), showScores(W) והחוק: squaresList(FullSquaresList) עם רשימת הריבועים. חוקים אלו מציגים את הנתונים העדכניים על מסך המשחק.לבסוף, החוק קורא לחוק:playMove(W)אשר במידת הצורך דואג לביצוע תור המחשב, אחרת (אם זה תור השחקן האנושי) לא מבצע דבר, כלומר חוק זה ממשיך את הרקורסיה של המשחק.

**תיאור מעטפת המערכת המומחית:**

הבינה המלאכותית - לימוד המחשב לחשוב כמו מוח של בן אדם, כי בעצם המחשב מבצע את כל מה שהאדם רושם לו לעשות. המטרה היא שהמחשב יקבל את מירב הידע על המשחק ובכך יוכל לנצח את השחקן האנושי (ככלי שיכול לתפקד יותר במהרה).   
מעבר לנתוני המשחק הבסיסיים, יש את החוקים שלפיהם המחשב בוחר במהלך אותו יבצע בתורו. המהלך צריך להיות המהלך החכם ביותר שיוביל כמה שיותר מהר לניצחון, לכן החוק שקובע את מהלך המחשב ברמה הקשה הוא בעצם 3 חוקים חילופיים למתאר: **chooseMoveForHardAI** שמסודרים לפי סדר עדיפויות- מהמהלך החכם והנכון ביותר לעשות ועד המהלך שנעשה בלית ברירה, אשר לא עומדת מאחוריו שום חשיבה (החוק האחרון מייצג את המהלך שמבצע המחשב במשחק ברמה הקלה).

סדר החוקים החילופיים הוא החשוב ביותר- כמו שהאדם מחשב בראש איזה מהלך חכם ביותר לעשות באותו התור, כך גם המחשב צריך לעשות כאשר מוצגים לו העדיפויות של כל מהלך. המחשב ילך לחוק הראשון שיבצע את המהלך החכם ביותר שיוביל לניצחון ורק אם מהלך זה לא אפשרי הוא ילך למהלך החכם בדרגה שנייה וכך הלאה...

כמובן שאת תהליך בחירת מהלך המחשב הארוך אין אנו רוצים לראות מודפס על לוח השאילתות כשרץ המשחק, לכן מספיק לנו לדעת כמה נקודות המחשב הרוויח בכל תור ולראות את המהלך שהוא ביצע. כאשר נראה שהמחשב מוביל ומשחק כראוי, נבין שהמחשב קרוב לניצחון והוא מצליח במשחק, אך אם המחשב לא משיג הרבה נקודות ולא מבצע מהלכים שקולים, המשתמש יבין שהוא מתעלה בביצועיו על ביצועי המחשב.

המטרה הכללית של המחשב במשחק זה היא לסגור כמה שיותר ריבועים ובכך להשיג כמה שיותר נקודות. בנוסף, עליו לחשב את כל המהלכים שאם יעשה אותם בתורו אזי השחקן שמולו- היריב, יוכל גם הוא להרוויח נקודה בתור שאחריו, ולא לבצע מהלך מסוג זה אלא אם אין לו ברירה.

בעמודים הבאים פירטתי על החוקים שאחראיים על ביצוע תור המחשב ועל החוקים שנעזר בהם המחשב בכדיי לבחור את המהלך החכם ביותר לעשות אשר יישרת את המטרה בצורה הטובה ביותר.

חוקי המהלכים של המחשב :

1. תור המחשב : 1/ playMove%

מטרת החוק: החוק מופעל עם פתיחת משחק חדש ואחרי לחיצה על קו אנכי או אופקי. חוק זה מבצע את תור המחשב (אם זה תור שחקן האנושי הוא לא עושה דבר) ודואג שהתור יתבצע בהתאם לרמת המשחק (סוג שמשחק) שנבחרה.

הצגת החוק:

**playMove**(W):-  
 send(timer(0.3), delay),  
 players(PlayersList),  
 writeln(['playmove - the players : ', PlayersList]),  
 players([p(aiEasy,\_,\_)|\_]),  
 chooseMoveForEasyAI(W).  
  
**playMove**(W):-  
 send(timer(0.3), delay),  
 players(PlayersList),  
 writeln(['playmove - the players : ', PlayersList]),  
 players([p(aiHard,\_,\_)|\_]),  
 chooseMoveForHardAI(W).  
  
**playMove**(\_).  
 %player human.  
 %do nothing - wait for press

הסבר החוק: 3 חוקים חילופיים למתאר **playMove** -  
ראשית, החוק מבצע השהייה של 0.3 שניות מהתור הקודם לתור הנוכחי, בכדי שהמשתמש האנושי יוכל לראות בבירור את מהלכי המחשב. לאחר מכן החוק בודק של מי התור הנוכחי (שחקן אנושי או מחשב) אם הוא של השחקן האנושי הוא לא עושה דבר, ואם הוא של המחשב הוא בודק איזה סוג שחקן (בהתאם לסוג המשחק) נבחר- שחקן קשה או קל  
ולפי זה הוא מבצע את תור המחשב, על ידי קריאה לחוק: chooseMoveForHardAI(W) או chooseMoveForEasyAI(W) בהתאמה.

2. בחירת מהלך המחשב- ברמה הקלה : 1/chooseMoveForEasyAI%

מטרת החוק: חוק זה בוחר את הקו עליו המחשב ילחץ בהתאם לרמה הקלה.

הצגת החוק:

**chooseMoveForEasyAI**(W):-  
 createListOfAllPossibleMoves(ML),  
 length(ML,Len),  
 Index is random(Len)+1,  
 nth1(Index,ML,move(Dir,X,Y)),  
 makeAIMove(W,Dir,X,Y).

הסבר החוק: חוק זהקורא לחוק: createListOfAllPossibleMoves(ML) שמחזיר לו רשימה עם כל הקווים הפנויים- הקווים שאף אחד עוד לא לחץ עליהם. לאחר מכן, החוק מבצע את פעולת ה- random ומגריל קו מתוך רשימה זאת. לבסוף החוק קורא לפעולה: makeAIMove(W,Dir,X,Y) עם הקו שנבחר ופעולה זאת לוחצת על קו זה ומעדכנת את נתוני המשחק.

3. בחירת מהלך המחשב- ברמה הקשה : 1/chooseMoveForHardAI%

מטרת החוק: חוק זה בוחר את הקו עליו המחשב ילחץ בהתאם לרמה הקשה, את קו זה הוא בוחר דרך מציאת המהלך החכם ביותר לעשות.

הצגת החוק:

**chooseMoveForHardAI** (W):-  
 closeSquaresList(CloseList),  
 length(CloseList,Len),  
 Len\=0,  
 Index is random(Len)+1,  
 nth1(Index, CloseList,close(Dir,X,Y)),  
 makeAIMove(W,Dir,X,Y).  
  
**chooseMoveForHardAI** (W):-  
 not\_line3List(Not\_line3List),  
 length(Not\_line3List,Len),  
 Len\=0,  
 Index is random(Len)+1,  
 nth1(Index,Not\_line3List,not\_line3(LineType,X,Y)), makeAIMove(W,LineType,X,Y).

**chooseMoveForHardAI** (W):-  
 createListOfAllPossibleMoves(ML),  
 length(ML,Len),  
 Index is random(Len)+1,  
 nth1(Index,ML,move(Dir,X,Y)),  
 makeAIMove(W,Dir,X,Y).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר **chooseMoveForHardAI** שמסודרים לפי סדר עדיפות בחירת המהלך הנכון (והחכם) לעשות –

החוק הראשון: בודק אם יש קו כלשהו שעל ידי בחירתו המחשב יסגור ריבוע- מה שהכי כדאי לו לעשות, כי בכך הוא יקבל נקודה ועוד תור, לכן, אם יש כזה קו, הוא בוחר את קו זה ולא מבצע את 2 החוקים האחרים.  
החוק קורא לפעולה: closeSquaresList(CloseList) שמחזירה את רשימת כל הקווים שיוכלו לסגור ריבוע. אם רשימה זאת לא ריקה- הוא מגריל קו ממנה וקורא לפעולה: makeAIMove(W,Dir,X,Y) עם הקו שנבחר ופעולה זאת לוחצת על קו זה ומעדכנת את נתוני המשחק. אם הרשימה ריקה- התוכנה עוברת לחוק השני.

החוק השני: בודק אם יש קו כלשהו (קו שלישי בריבוע) שעל ידי בחירתו, המשתמש האנושי יוכל בתור הבא לסגור ריבוע, מה שלא טוב למחשב מאחר ובכך המשתמש האנושי יקבל נקודה ועוד תור. לאחר מכן, הוא בודק אם יש קווים פנויים אחרים על הלוח ואם כן, הוא בוחר קו מהם ולא מבצע את החוק השלישי.  
החוק קורא לפעולה: not\_line3List(Not\_line3List) שמחזירה את רשימת כל הקווים הפנויים שעל ידי בחירה בהם המשתמש האנושי לא יסגור ריבוע. אם רשימה זאת לא ריקה, הוא מגריל קו ממנה וקורא לפעולה: makeAIMove(W,Dir,X,Y) עם הקו שנבחר ופעולה זאת לוחצת על קו זה ומעדכנת את נתוני המשחק. אם הרשימה ריקה- התוכנה עוברת לחוק השלישי.

החוק השלישי: זהו החוק שהמחשב מבצע אם אין לו ברירה אחרת, מאחר והוא לא מרוויח מתור זה שום תועלת- זאת בחירה רנדומאלית ולא חשיבתית.   
חוק זה פועל כמו החוק chooseMoveForEasyAI(W) .

4. ביצוע תור המחשב : 1/makeAIMove%

מטרת החוק: חוק זה שם את הקו שנבחר על ידי המחשב במקום המתאים ומעדכן את הנתונים של המשחק.

הצגת החוק:

**makeAIMove**(W,horizontal,X,Y):-  
 hLinePressed(W,X,Y).

**makeAIMove**(W,vertical,X,Y):-  
 vLinePressed(W,X,Y).

הסבר החוק: אלו הם 2 חוקים חילופיים למתאר **makeAIMove** -  
חוק זה מקבל את הקו שבחר המחשב וקורא לחוק: hLinePressed(W,X,Y) אם נבחר קו אנכי, ולחוק: vLinePressed(W,X,Y) אם נבחר קו אופקי.

רשימות עזר :

\*רשימת הקווים הפנויים : 1/createListOfAllPossibleMoves%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים הפנויים על הלוח- הקווים ששום שחקן לא לחץ עליהם עדיין.

הצגת החוק:

**createListOfAllPossibleMoves**(MovesList):-  
 pointsList(Points), findall(move(horizontal,X,Y),member(p((X,Y),\_,\_,0,\_),Points),Lhorizontal), findall(move(vertical,X,Y) ,member(p((X,Y),\_,0,\_,\_),Points),Lvertical), append(Lhorizontal,Lvertical,MovesList),  
 writeln(['moves list : ',MovesList]).

הסבר החוק: החוק משתמש בפעולה findall בכדיי ליצור רשימה של כל הקווים האנכיים שפנויים וזאת הפעולה עושה בעזרת איברי רשימת כל הנקודות (הפעולה מכניסה לרשימה את כל הקווים שמימין לנקודה כלשהי ברשימה, אשר קיימים בלוח ועדיין לא לחצו עליהם), לאחר מכן הוא משתמש באותה פעולה אך בכדיי ליצור רשימה של כל הקווים האופקיים שפנויים (הפעולה מכניסה לרשימה את כל הקווים שמתחת לנקודה כלשהי ברשימה, אשר קיימים בלוח ועדיין לא לחצו עליהם). החוק מאחד את שתי רשימות אלו לרשימה אחת בעזרת הפעולה append, ומחזיר את רשימת האיחוד.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

\*רשימת הקווים שסוגרים ריבוע : 1/closeSquaresList%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים הפנויים על הלוח שאם המחשב יבחר אותם הוא יסגור ריבוע ויקבל נקודה.

הצגת החוק:

**closeSquaresList**(CloseList):-  
 horizontalCloseSquaresList(CloseHlist), verticalCloseSquaresList(CloseVlist),  
 append(CloseHlist,CloseVlist,CloseList),  
 writeln(['close list : ',CloseList]).

הסבר החוק: החוק קורא לחוקים: horizontalCloseSquaresList(CloseHlist) ו- verticalCloseSquaresList(CloseVlist) שמחזירים את רשימת הקווים האופקיים והאנכיים בהתאמה, שאם המחשב יבחר בהם הוא יסגור ריבוע. לאחר מכן, הוא מאחד את שתי רשימות אלו לרשימה אחת בעזרת הפעולה append, ומחזיר את רשימת האיחוד.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

\*רשימת הקווים האופקיים שסוגרים ריבוע: 1/horizontalCloseSquaresList%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים האנכיים הפנויים על הלוח שאם המחשב יבחר אותם הוא יסגור ריבוע ויקבל נקודה.

הצגת החוק:

**horizontalCloseSquaresList**(CloseHlist):-  
 createListOfAllPossibleMoves(ML),  
 findall(close(horizontal,X,Y),( member(move(horizontal,X,Y),ML),  
 horizontalAtXYasTopLeftCorner(X,Y,1);  
 horizontalAtXYasDownLeftCorner(X,Y,1) ),  
 CloseHlist),  
 writeln(['close Hori list : ',CloseHlist]).

הסבר החוק: החוק קורא לחוק: (createListOfAllPossibleMoves(ML שמחזיר את רשימת כל הקווים הפנויים שעל הלוח, ובעזרת הפעולה findall הוא יוצר ומחזיר רשימה של כל הקווים האופקיים שהם איברי הרשימה ושהם גם קווים שמקיימים את אחד מחוקים אלו:horizontalAtXYasTopLeftCorner(X,Y,1) / horizontalAtXYasDownLeftCorner(X,Y,1) כלומר שאם המחשב יבחר את קווים אלו, הוא יסגור ריבוע ואז החוק יחזיר 1.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

החוקים שבודקים אם הקו האופקי סוגר ריבוע:

הקווים האופקיים שסוגרים ריבוע, אשר מימין לנקודה השמאלית העליונה: horizontalAtXYasTopLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אופקי, שקיבל את שיעורי הנקודה שמשמאלו, יסגור ריבוע.  
חוק זה בודק את הקווים האופקיים שמימין לנקודה השמאלית העליונה שבריבוע.

הצגת החוק:

**horizontalAtXYasTopLeftCorner**(X,Y,Score):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,1,0,\_),PointsList),   
 XPlus1 is X+1,  
 YPlus1 is Y+1,   
 member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),  
 Score is 1.  
**horizontalAtXYasTopLeftCorner**(\_,\_,0).

כתום= קו תפוס  
אפור= קו פנוי

הסבר החוק: החוק בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה והקו שמשמאלה כבר תפוסים.  
אם כל זה מתקיים- הקו שמימין לנקודה יסגור ריבוע ואז החוק יחזיר 1. אחרת- החוק יחזיר 0.

הקווים האופקיים שסוגרים ריבוע, אשר מימין לנקודה השמאלית התחתונה: horizontalAtXYasDownLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אופקי, שקיבל את שיעורי הנקודה שמשמאלו, יסגור ריבוע.  
חוק זה בודק את הקווים האופקיים שמימין לנקודה השמאלית התחתונה שבריבוע.

הצגת החוק:

**horizontalAtXYasDownLeftCorner**(X,Y,Score):-   
 pointsList(PointsList),  
 member(p((X,Y),1,\_,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y-1,  
 member(p((XPlus1,YPlus1),\_,1,\_,1),PointsList),  
 Score is 1.  
**horizontalAtXYasTopLeftCorner**(\_,\_,0).

כתום= קו תפוס  
אפור= קו פנוי

הסבר החוק: החוק בודק שהנקודה של הקו שקיבל מקיימת: הקו שמעליה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה שלפניה מקיימת: הקו שמתחתיה והקו שמשמאלה כבר תפוסים.  
אם כל זה מתקיים- הקו שמימין לנקודה יסגור ריבוע ואז החוק יחזיר 1. אחרת- החוק יחזיר 0.

\*רשימת הקווים האנכיים שסוגרים ריבוע: 1/ verticalCloseSquaresList%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים האופקיים הפנויים על הלוח שאם המחשב יבחר אותם הוא יסגור ריבוע ויקבל נקודה.

הצגת החוק:

**verticalCloseSquaresList(**CloseVlist):-  
 createListOfAllPossibleMoves(ML),  
 findall(close(vertical,X,Y),( member(move(vertical,X,Y),ML),  
 verticalAtXYasTopLeftCorner(X,Y,1);  
 verticalAtXYasTopRightCorner(X,Y,1) ),  
 CloseVlist),  
 writeln(['close Verti list : ',CloseVlist]).

הסבר החוק: החוק קורא לחוק: (createListOfAllPossibleMoves(ML שמחזיר את רשימת כל הקווים הפנויים שעל הלוח, ובעזרת הפעולה findall הוא יוצר ומחזיר רשימה של כל הקווים האנכיים שהם איברי הרשימה ושהם גם קווים שמקיימים את אחד מחוקים אלו: verticalAtXYasTopLeftCorner (X,Y,1) / verticalAtXYasTopRightCorner (X,Y,1) ,כלומר שאם המחשב יבחר את קווים אלו, הוא יסגור ריבוע ואז החוק יחזיר 1.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

החוקים שבודקים אם הקו האנכי סוגר ריבוע:

הקווים האנכיים שסוגרים ריבוע, אשר מתחת לנקודה השמאלית העליונה: verticalAtXYasTopLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אנכי, שקיבל את שיעורי הנקודה שמעליו, יסגור ריבוע.  
חוק זה בודק את הקווים האנכיים שמתחת לנקודה השמאלית העליונה שבריבוע.

הצגת החוק:

**verticalAtXYasTopLeftCorner**(X,Y,Score):-  
 pointsList(PointsList),  
 member(p((X,Y),\_,0,1,\_),PointsList),   
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),  
 Score is 1.  
**verticalAtXYasTopLeftCorner**(\_,\_,0).

כתום= קו תפוס  
אפור= קו פנוי

הסבר החוק: החוק בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק ושהקו שמימינה תפוס כבר, כלומר כבר בחרו בו. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה והקו שמשמאלה כבר תפוסים.  
אם כל זה מתקיים- הקו שמתחת לנקודה יסגור ריבוע ואז החוק יחזיר 1. אחרת- החוק יחזיר 0.

הקווים האנכיים שסוגרים ריבוע, אשר מתחת לנקודה הימנית העליונה: verticalAtXYasTopRightCorner /3 %

מטרת החוק: החוק בודק אם קו אנכי, שקיבל את שיעורי הנקודה שמעליו, יסגור ריבוע.  
חוק זה בודק את הקווים האנכיים שמתחת לנקודה הימנית העליונה שבריבוע.

הצגת החוק:

**verticalAtXYasTopRightCorner**(X,Y,Score):-  
 pointsList(PointsList),   
 member(p((X,Y),\_,0,\_,1),PointsList),  
 XMinus1 is X-1,  
 YPlus1 is Y+1,  
 member(p((XMinus1,YPlus1),1,\_,1,\_),PointsList),  
 Score is 1.  
**verticalAtXYasTopRightCorner**(\_,\_,0).

כתום= קו תפוס  
אפור= קו פנוי

הסבר החוק: החוק בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק ושהקו שמשמאלה תפוס כבר, כלומר כבר בחרו בו. בנוסף, הוא בודק שהנקודה שבטור לפניה ובשורה אחריה מקיימת: הקו שמעליה והקו שמימינה כבר תפוסים.  
אם כל זה מתקיים- הקו שמתחת לנקודה יסגור ריבוע ואז החוק יחזיר 1. אחרת- החוק יחזיר 0.

\*רשימת הקווים ה-3 שאם המחשב יבחר בהם המשתמש האנושי יסגור בתור אחריו ריבוע: 1/ line3List%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים הפנויים על הלוח שאם המחשב יבחר אותם המשתמש האנושי יסגור בתור שאחריו ריבוע ויקבל נקודה.

הצגת החוק:

**line3List**(Line3List):-  
 line3horizontalList(Line3\_hList),  
 line3verticalList(Line3\_vList),  
 append(Line3\_hList,Line3\_vList,Line3List),  
 writeln(['line 3 list : ',Line3List]).

הסבר החוק: החוק קורא לחוקים: line3horizontalList(Line3\_hList) ו-(line3verticalList(Line3\_vList שמחזירים את רשימת הקווים האופקיים והאנכיים בהתאמה, שאם המחשב יבחר בהם המשתמש האנושי יסגור בתור אחריו ריבוע. לאחר מכן, הוא מאחד את שתי רשימות אלו לרשימה אחת בעזרת הפעולה append, ומחזיר את רשימת האיחוד.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

\*רשימת הקווים שאם המחשב יבחר בהם המשתמש האנושי לא יסגור בתור אחריו ריבוע: 1/ not\_line3List%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים הפנויים על הלוח שאם המחשב יבחר אותם המשתמש האנושי לא יסגור בתור שאחריו ריבוע.

הצגת החוק:

**not\_line3List**(Line3List):-  
 createListOfAllPossibleMoves(ML),  
 line3List(Line3List),  
 findall(not\_line3(LineType,X,Y),( member(move(LineType,X,Y),ML),  
 not(member(line3(LineType,X,Y),Line3List)) ),  
 Not\_line3List),  
 writeln(['not line 3 list : ',Not\_line3List]).

הסבר החוק: החוק קורא לחוק: createListOfAllPossibleMoves(ML) שמחזיר את הרשימה הראשונה: רשימת כל הקווים הפנויים על הלוח- שאף משתמש עוד לא לחץ עליהם. לאחר מכן החוק קורא לחוק: line3List(Line3List) שמחזיר את הרשימה השנייה: רשימת כל הקווים שאם המחשב יבחר אותם המשתמש האנושי יסגור בתור שאחריו ריבוע. החוק משתמש בפעולה: findall שבעזרתה יוצר ומחזיר רשימה של כל הקווים שהם איברי הרשימה הראשונה ולא נמצאים ברשימה השנייה.  
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

\*רשימת הקווים האופקיים שאם המחשב יבחר בהם המשתמש האנושי יסגור בתור אחריו ריבוע: 1/ line3horizontalList%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים האופקיים הפנויים על הלוח שאם המחשב יבחר אותם המשתמש האנושי יסגור בתור שאחריו ריבוע ויקבל נקודה.

הצגת החוק:

**line3horizontalList**(Line3\_hList):-  
 createListOfAllPossibleMoves(ML),  
 findall(line3(horizontal,X,Y),( member(move(horizontal,X,Y),ML),  
 line3HorizontalAtXYasTopLeftCorner(X,Y,1);  
 line3HorizontalAtXYasDownLeftCorner(X,Y,1) ),  
 Line3\_hList),  
 writeln(['line 3 horizontal list : ',Line3\_hList]).

הסבר החוק: החוק קורא לחוק: (createListOfAllPossibleMoves(ML שמחזיר את רשימת כל הקווים הפנויים שעל הלוח, ובעזרת הפעולה findall הוא יוצר ומחזיר רשימה של כל הקווים האופקיים שהם איברי הרשימה ושהם גם קווים שמקיימים את אחד מחוקים אלו: line3HorizontalAtXYasTopLeftCorner(X,Y,1) או: line3HorizontalAtXYasDownLeftCorner(X,Y,1) ,  
כלומר שאם המחשב יבחר את קווים אלו, המשתמש האנושי יסגור בתור שאחריו ריבוע ואז החוק יחזיר 1.   
בנוסף, חוק זה מדפיס את הרשימה המתקבלת על מסך השאילתות.

החוקים שבודקים אם הקו האופקי הוא קו 3 – קו שאם המחשב יבחר בו המשתמש האנושי יסגור בתור אחריו ריבוע:

הקווים האופקיים שסוגרים ריבוע, אשר מימין לנקודה השמאלית העליונה: line3HorizontalAtXYasTopLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אופקי שהמחשב יבחר, ושהחוק קיבל את שיעורי הנקודה שמשמאלו, יגרום למשתמש האנושי לסגור ריבוע בתור שאחרי.  
חוק זה בודק את הקווים האופקיים שמימין לנקודה השמאלית העליונה שבריבוע.

הצגת החוק:

  
**line3HorizontalAtXYasTopLeftCorner**(X,Y,Flag):-  
 pointsList(PointsList),  
 member(p((X,Y),\_,1,0,\_),PointsList),   
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),1,\_,\_,0),PointsList),  
 Flag is 1.  
  
**line3HorizontalAtXYasTopLeftCorner**(X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,1,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),0,\_,\_,1),PointsList),  
 Flag is 1.

כתום= קו תפוס  
אפור= קו פנוי

כתום= קו תפוס  
אפור= קו פנוי

**line3HorizontalAtXYasTopLeftCorner**(X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),  
 Flag is 1.  
 **line3HorizontalAtXYasTopLeftCorner** (\_,\_,0).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר **line3HorizontalAtXYasTopLeftCorner** -  
כל החוקים הם למטרה אחת- לבדוק אם בחירת הקו שמימין לנקודה שקיבלו יגרום למשתמש האנושי לסגור בתור אחרי המחשב ריבוע. אם כן- החוק יחזיר 1, אם לא- החוק יחזיר 0. (סדר כתיבתם לא משמעותי)

החוק הראשון: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה כבר תפוס והקו שמשמאלה פנוי.

החוק השני: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה פנוי והקו שמשמאלה תפוס.

החוק השלישי: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה ומימינה פנויים וקיימים, כלומר אף שחקן לא בחר אותם והם חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה ומשמאלה תפוסים, כלומר כבר בחרו אותם.

הקווים האופקיים שסוגרים ריבוע, אשר מימין לנקודה השמאלית התחתונה: line3HorizontalAtXYasDownLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אופקי שהמחשב יבחר, ושהחוק קיבל את שיעורי הנקודה שמשמאלו, יגרום למשתמש האנושי לסגור ריבוע בתור שאחרי.  
חוק זה בודק את הקווים האופקיים שמימין לנקודה השמאלית התחתונה שבריבוע.

הצגת החוק:

  
**line3HorizontalAtXYasDownLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),1,\_,0,\_),PointsList),   
 XPlus1 is X+1,  
 YPlus1 is Y-1,  
 member(p((XPlus1,YPlus1),\_,1,\_,0),PointsList),  
 Flag is 1.

כתום= קו תפוס  
אפור= קו פנוי

** line3HorizontalAtXYasDownLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),1,\_,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y-1,  
 member(p((XPlus1,YPlus1),\_,0,\_,1),PointsList),  
 Flag is 1.

 **line3HorizontalAtXYasDownLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),0,\_,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y-1,  
 member(p((XPlus1,YPlus1),\_,1,\_,1),PointsList),  
 Flag is 1.  
 **line3HorizontalAtXYasDownLeftCorner** (\_,\_,0).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר-**line3HorizontalAtXYasDownLeftCorner** כל החוקים הם למטרה אחת- לבדוק אם בחירת הקו שמימין לנקודה שקיבלו יגרום למשתמש האנושי לסגור בתור אחרי המחשב ריבוע. אם כן- החוק יחזיר 1, אם לא- החוק יחזיר 0. (סדר כתיבתם לא משמעותי)

החוק הראשון: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמעליה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה לפניה מקיימת: הקו שמתחתיה כבר תפוס והקו שמשמאלה פנוי וקיים.

החוק השני: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמעליה כבר תפוס, כלומר כבר בחרו אותו, הקו שמימינה שברצוננו לבחור פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה לפניה מקיימת: הקו שמתחתיה פנוי וקיים והקו שמשמאלה תפוס.

החוק השלישי: בודק שהנקודה של הקו שקיבל מקיימת: הקווים שמעלייה ומימינה פנויים וקיימים, כלומר אף שחקן לא בחר אותם והם חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה לפנייה מקיימת: הקווים שמתחתיה ומשמאלה תפוסים, כלומר, כבר בחרו אותם.

\*רשימת הקווים האנכיים שאם המחשב יבחר בהם המשתמש האנושי יסגור בתור אחריו ריבוע: 1/ line3verticalList%

מטרת החוק: החוק בונה ומחזיר רשימה של כל הקווים האנכיים הפנויים על הלוח שאם המחשב יבחר אותם המשתמש האנושי יסגור בתור שאחריו ריבוע ויקבל נקודה.

הצגת החוק:

**line3verticalList**(Line3\_vList):-  
 createListOfAllPossibleMoves(ML),  
 findall(line3(vertical,X,Y),( member(move(vertical,X,Y),ML),  
 line3VerticalAtXYasTopLeftCorner(X,Y,1);  
 line3VerticalAtXYasTopRightCorner(X,Y,1) ),  
 Line3\_vList),  
 writeln(['line 3 vertical list : ',Line3\_vList]).

הסבר החוק: החוק קורא לחוק: (createListOfAllPossibleMoves(ML שמחזיר את רשימת כל הקווים הפנויים שעל הלוח, ובעזרת הפעולה findall הוא יוצר ומחזיר רשימה של כל הקווים האופקיים שהם איברי הרשימה ושהם גם קווים שמקיימים את אחד מחוקים אלו: line3VerticalAtXYasTopLeftCorner(X,Y,1) / line3VerticalAtXYasTopRightCorner(X,Y,1) כלומר שאם המחשב יבחר את קווים אלו, המשתמש האנושי יסגור בתור שאחריו ריבוע ואז החוק יחזיר 1.

החוקים שבודקים אם הקו האנכי הוא קו 3 – קו שאם המחשב יבחר בו המשתמש האנושי יסגור בתור אחריו ריבוע:

הקווים האנכיים שסוגרים ריבוע, אשר מתחת לנקודה השמאלית העליונה: line3VerticalAtXYasTopLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אנכי שהמחשב יבחר, ושהחוק קיבל את שיעורי הנקודה שמעליו, יגרום למשתמש האנושי לסגור ריבוע בתור שאחרי.  
חוק זה בודק את הקווים האנכיים שמתחת לנקודה השמאלית העליונה שבריבוע.

הצגת החוק:

  
**line3VerticalAtXYasTopLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,1,\_),PointsList),   
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),0,\_,\_,1),PointsList),  
 Flag is 1.

כתום= קו תפוס  
אפור= קו פנוי

** line3VerticalAtXYasTopLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,1,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),1,\_,\_,0),PointsList),  
 Flag is 1.

כתום= קו תפוס  
אפור= קו פנוי

**line3VerticalAtXYasTopLeftCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,0,\_),PointsList),  
 XPlus1 is X+1,  
 YPlus1 is Y+1,   
 member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),  
 Flag is 1.  
 **line3VerticalAtXYasTopLeftCorner** (\_,\_,0).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר **line3VerticalAtXYasTopLeftCorner** -  
כל החוקים הם למטרה אחת- לבדוק אם בחירת הקו שמתחת לנקודה שקיבלו יגרום למשתמש האנושי לסגור בתור אחרי המחשב ריבוע. אם כן- החוק יחזיר 1, אם לא- החוק יחזיר 0. (סדר כתיבתם לא משמעותי)

החוק הראשון: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק והקו שמימינה תפוס, כלומר כבר בחרו אותו. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה פנוי וקיים והקו שמשמאלה תפוס.

החוק השני: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק והקו שמימינה תפוס, כלומר כבר בחרו אותו. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה תפוס והקו שמשמאלה פנוי וקיים.

החוק השלישי: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה ומימינה פנויים וקיימים, כלומר אף שחקן לא בחר אותם והם חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור אחריה ובשורה אחריה מקיימת: הקו שמעליה ומשמאלה תפוסים, כלומר כבר בחרו אותם.

הקווים האנכיים שסוגרים ריבוע, אשר מתחת לנקודה הימינית העליונה: line3VerticalAtXYasTopLeftCorner/3 %

מטרת החוק: החוק בודק אם קו אנכי שהמחשב יבחר, ושהחוק קיבל את שיעורי הנקודה שמעליו, יגרום למשתמש האנושי לסגור ריבוע בתור שאחרי.  
חוק זה בודק את הקווים האנכיים שמתחת לנקודה השמאלית העליונה שבריבוע.

הצגת החוק:

  
**line3VerticalAtXYasTopRightCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,\_,1),PointsList),   
 XPlus1 is X-1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),0,\_,1,\_),PointsList),  
 Flag is 1.

כתום= קו תפוס  
אפור= קו פנוי

** line3VerticalAtXYasTopRightCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,\_,1),PointsList),  
 XPlus1 is X-1,  
 YPlus1 is Y+1,  
 member(p((XPlus1,YPlus1),1,\_,0,\_),PointsList),  
 Flag is 1.

**line3VerticalAtXYasTopRightCorner** (X,Y,Flag):-   
 pointsList(PointsList),  
 member(p((X,Y),\_,0,\_,0),PointsList),  
 XPlus1 is X-1,  
 YPlus1 is Y+1,   
 member(p((XPlus1,YPlus1),1,\_,1,\_),PointsList),  
 Flag is 1.  
 **line3VerticalAtXYasTopRightCorner** (\_,\_,0).

הסבר החוק: אלו הם 3 חוקים חילופיים למתאר **line3VerticalAtXYasTopRightCorner** -  
כל החוקים הם למטרה אחת- לבדוק אם בחירת הקו שמתחת לנקודה שקיבלו יגרום למשתמש האנושי לסגור בתור אחרי המחשב ריבוע. אם כן- החוק יחזיר 1, אם לא- החוק יחזיר 0. (סדר כתיבתם לא משמעותי)

החוק הראשון: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק והקו שמשמאלה תפוס, כלומר כבר בחרו אותו. בנוסף, הוא בודק שהנקודה שבטור לפנייה ובשורה אחריה מקיימת: הקו שמעליה פנוי וקיים והקו שמימינה תפוס.

החוק השני: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה פנוי וקיים, כלומר אף שחקן לא בחר אותו והוא חלק מלוח המשחק והקו שמשמאלה תפוס, כלומר כבר בחרו אותו. בנוסף, הוא בודק שהנקודה שבטור לפנייה ובשורה אחריה מקיימת: הקו שמעליה תפוס והקו שמימינה פנוי וקיים.

החוק השלישי: בודק שהנקודה של הקו שקיבל מקיימת: הקו שמתחתיה ומשמאלה פנויים וקיימים, כלומר אף שחקן לא בחר אותם והם חלק מלוח המשחק. בנוסף, הוא בודק שהנקודה שבטור לפנייה ובשורה אחריה מקיימת: הקו שמעליה ומימינה תפוסים, כלומר כבר בחרו אותם.

**ממשק למשתמש:**

את המדריך להפעלת המערכת prolog, המדריך להרצת המשחק דרך מסך השאילתות והסבר על הכפתורים שבתפריט ובלוח המשחק – הצגתי בעמודים 2-3 שבתחילת העבודה.

בכדיי לראות יותר טוב את מהלכי כל שחקן ועדכון הנתונים במהלך כל תור, חוץ מהגראפיקה שעוזרת לנו לראות וויזואלית את הנתונים על המסך, הדפסתי , על מסך השאילתות עצמו בעזרת הפעולה: writeln() את הרשימות החשובות שמאחסנות את נתוני המשחק.

החוקים בהם יודפסו הנתונים הם:

1. createListOfAllPossibleMoves(MovesList)- מדפיס את רשימת כל הקווים הפנויים על לוח המשחק, כלומר הקווים האפורים שעוד לא לחצו עליהם.

לדוגמא:

[moves list : ,[move(horizontal,5,3),move(horizontal,5,1),move(horizontal,4,4),move(horizontal,1,3),move(horizontal,2,5),move(horizontal,2,4),move(horizontal,3,1),move(horizontal,3,5),move(horizontal,5,5),move(horizontal,4,5),move(horizontal,4,3),move(horizontal,1,2),move(horizontal,4,1),move(horizontal,1,5),move(horizontal,1,6),move(vertical,2,3),move(vertical,6,2),move(vertical,4,4),move(vertical,5,4),move(vertical,1,4),move(vertical,6,4),move(vertical,1,1),move(vertical,4,2),move(vertical,4,1),move(vertical,1,5)]]

2. (closeSquaresList(CloseList- מדפיס את רשימת כל הקווים הפנויים על לוח המשחק שאם ילחצו עליהם ייסגר ריבוע.

לדוגמא: [close list : ,[close(horizontal,2,6)]]

3. (not\_line3List(Not\_line3List- מדפיס את רשימת כל הקווים הפנויים על לוח המשחק שאם המחשב יבחר בהם בתורו, אזי בתור הבא המשתמש האנושי לא יסגור ריבוע.

לדוגמא:

[not line 3 list : ,[not\_line3(horizontal,4,6),not\_line3(horizontal,4,4),not\_line3(horizontal,5,5),not\_line3(horizontal,5,2),not\_line3(horizontal,3,1),not\_line3(horizontal,5,3),not\_line3(horizontal,4,1),not\_line3(horizontal,1,4),not\_line3(horizontal,5,4),not\_line3(horizontal,1,5),not\_line3(horizontal,1,6),not\_line3(horizontal,5,6),not\_line3(vertical,6,1),not\_line3(vertical,5,1),not\_line3(vertical,2,3),not\_line3(vertical,1,3),not\_line3(vertical,3,3),not\_line3(vertical,5,5),not\_line3(vertical,4,3),not\_line3(vertical,5,3),not\_line3(vertical,4,1),not\_line3(vertical,6,2),not\_line3(vertical,6,3),not\_line3(vertical,1,4),not\_line3(vertical,5,4),not\_line3(vertical,6,4),not\_line3(vertical,1,5),not\_line3(vertical,6,5)]]

4. playMove(W)- מדפיס את רשימת השחקנים המתעדכנת בכל תור ותור.

לדוגמא: [p(human,0,green),p(aiHard,3,red)]

5. showScores(W)- מדפיס גם כן את רשימת השחקנים המתעדכנת בכל תור ותור, בכדי שיהיה לנו קל להתאים את הנקודות של כל שחקן לרשימת הריבועים שמודפסת מיד אחריה.

לדוגמא: [p(human,0,green),p(aiHard,3,red)]  
 [sq(2,5,red),sq(2,4,red),sq(4,2,red)]

6. (hLinePressed(W,X,Y ו- (vLinePressed(W,X,Y- מדפיסות בכל פעם את הנקודה של הקו שבחר השחקן בהתאמה: אם אופקי- את הנקודה שמשמאל לקו, אם אנכי- את הנקודה שמעל לקו.

לדוגמא: [5,4]

**סיכום אישי:**

פרויקט זה יוצר תוכנה שהיא בין התוכנות היחידות, אם לא היחידה, שבונות את המשחק הזה כך שלשחקן האנושי תהיה את האופציה לשחק מול המחשב, כלומר לא צריכים שני שחקנים אנושיים בכדי לשחק במשחק, אלא ניתן לשחק בו גם לבד מול המחשב.

פרויקט זה נבנה בזמן מוגבל, לכן המחשב לא יודע לחשוב על כל האופציות (מצבים) שמובילות לניצחון כי לא הכנסתי לו חוקים שמעבר למהלכים הבסיסיים שמובילים לניצחון.  
יש לציין שמשחק זה יכול להיות מאוד מורכב, למשל אם נסתכל בתיאוריית "אלווין ברלקאמף" על אסטרטגיות המשחק וננסה להכניס את כולן לקוד התוכנה, ייקח לנו המון זמן ומקום וכנראה שגם הזמן שייקח למחשב לבצע תור יהיה איטי יותר. לכן הסתפקתי בכתיבת קוד בסיסי המסתמך על 3 חוקים חילופיים בלבד למהלך המחשב.  
דבר נוסף שניתן להוסיף לפרויקט, הוא לעשות כפתור של 'תחילת משחק' (סוג משחק) נוסף: משחק של מחשב אחד מול מחשב אחר, שמאחורי כל מחשב יהיה שחקן אנושי שיבצע את המהלכים. זוהי עוד דרך בה ניתן לשחק במשחק זה ולא חייבים למצוא עוד יריב אנושי למשחק (כי הרשת תמצא לבד).

למדתי מפרויקט זה הרבה דברים שלא ידעתי שניתן לעשות בעזרת המחשב.   
בעיקר התנסיתי בכתיבת ויצירת משחק משלי- הן בבחינה הגראפית והן מבחינת התוכנה הכתובה.

מבחינת הגראפיקה: יצרתי את התמונות שמרכיבות את המשחק ברובן לבד ותכננתי איך המשחק יראה ואיך המשתמש האנושי יוכל לדעת בכל פעם שתורו לשחק או לדעת מה ניקודו בכל זמן נתון.   
הגראפיקה היא כלי מדהים, יחסית חדשני ומאוד שימושי ומועיל. בנוסף, כאשר אתה רואה על מסך המחשב וויזואלית את התוכנה שלך עובדת - אין סיפוק גדול יותר.

מבחינת התכנות וההתמודדות עם הכתיבה: למדתי שלפני שכותבים את העבודה עצמה יש לחשוב איך נאחסן את הנתונים של המשחק בצורה הטובה והנוחה ביותר, למדתי שלסדר כתיבת החוקים יש משמעות רבה על מהלכי המחשב ושלפני שכותבים חוק צריך לחשוב על מטרת החוק וניתן גם להיעזר בגראפיקה בכדי לראות יותר טוב את כל מה שצריך לכלול בחוק על מנת לממש את מטרתו.

בנוסף, פרויקט זה לימד אותי שאם נתקעים בבאג בתוכנה או שהתכונה לא עובדת כראוי, אסור לוותר ולדחות את התיקון, יש לתקן את העבודה במיידית (אחרי אותו חוק שכתבנו וגרם לבעיה), ולחשוב מה הוביל לבאג כאשר ניתן להיעזר במעקב על פעולות התוכנה, או שניתן להדפיס על מסך השאילתות את מה שעדכננו או שינינו לאחר כל שורה בחוק.

לסיום, בתחילה היה לי קשה להתחיל לכתוב את הפרויקט, אך אחרי שהתחלתי ופתרתי בעיות נכנס בי יותר ביטחון והצלחתי להמשיך את הפרויקט במהירות יחסית להתחלה ויותר בקלות. דבר זה לימד אותי לא לוותר לעצמי ונתן לי בטחון ביכולות שלי.

**ביבליוגרפיה:**

אתר ויקיפדיה:

<http://en.wikipedia.org/wiki/Dots_and_Boxes>

**חוקי המשחק:**

מס' המשתתפים: במשחק "Dots and Boxes" משחקים שני שחקנים, אשר לכל שחקן צבע משלו (אין לצבע משמעות במשחק, הוא רק בכדי שנוכל לראות בבירור איזה מהלך עשה כל משתמש).

כלי המשחק: עפרון ודף, בכדי שנוכל לצייר על הדף בעצמנו את לוח המשחק (הנקודות), להעביר את הקווים בכל תור ואם נסגר ריבוע- לסמן זאת במיקום המתאים על הלוח.  
*כל זה עושה המחשב בעזרת הגראפיקה*, ואנו מקבלים לוח משחק מוכן עם נקודות וקווים שניתן ללחוץ עליהן, לכן אין צורך בעיפרון ודף אלא רק במחשב, מסך ועכבר.

מהלך המשחק: כל משתמש בתורו צריך ללחוץ על קו אפור (שעוד לא לחצו עליו) אופקי או אנכי.   
\*ריבוע- מצב בו יש 4 קווים המסודרים בצורה הנ"ל: 2 אופקיים המקבילים זה לזה ו-2 אנכיים, הסמוכים לקווים האופקיים ומקבילים זה לזה גם כן.  
כאשר עם לחיצה על קו נסגר ריבוע, השחקן שסגר את הריבוע מקבל נקודה וחייב לבצע עוד תור.  
**חשוב לציין:** מי שסוגר ריבוע בתורו הוא זה שמקבל את הנקודה, גם אם הקווים האחרים שיוצרים את הריבוע היו חלקם או כולם בצבע של היריב! (כלומר, כפי שציינתי בהתחלה הצבע לא משמעותי לניקוד המשחק)

סיום המשחק: המשחק מסתיים כאשר אין יותר קווים אפורים פנויים ללחיצה על המסך, ובכל לא ניתן לבצע עוד תורות.

מנצח המשחק: המשתמש שסגר יותר ריבועים, ובכך צבר יותר נקודות.  
מכאן, מטרת המשחק היא לסגור כמה שיותר ריבועים ולא לתת ליריב לסגור ריבוע.

ראה את תרשים המשחק בכדי להבין את ההצגה הגראפית.

**קוד התוכנית (כולל תיעוד) :**

:-dynamic pointsList/1.

:-dynamic squaresList/1.

:-dynamic players/1.

%%% scorePlayer(NumofScore,TheAppropriatePicture) %%%

%scoreRed(sumScoreRed,Pic).

scoreRed(0,'0r.bmp').

scoreRed(1,'1r.bmp').

scoreRed(2,'2r.bmp').

scoreRed(3,'3r.bmp').

scoreRed(4,'4r.bmp').

scoreRed(5,'5r.bmp').

scoreRed(6,'6r.bmp').

scoreRed(7,'7r.bmp').

scoreRed(8,'8r.bmp').

scoreRed(9,'9r.bmp').

scoreRed(10,'10r.bmp').

scoreRed(11,'11r.bmp').

scoreRed(12,'12r.bmp').

scoreRed(13,'13r.bmp').

scoreRed(14,'14r.bmp').

scoreRed(15,'15r.bmp').

scoreRed(16,'16r.bmp').

scoreRed(17,'17r.bmp').

scoreRed(18,'18r.bmp').

scoreRed(19,'19r.bmp').

scoreRed(20,'20r.bmp').

scoreRed(21,'21r.bmp').

scoreRed(22,'22r.bmp').

scoreRed(23,'23r.bmp').

scoreRed(24,'24r.bmp').

scoreRed(25,'25r.bmp').

%scoreGreen(sumScoreGreen,Pic).

scoreGreen(0,'0g.bmp').

scoreGreen(1,'1g.bmp').

scoreGreen(2,'2g.bmp').

scoreGreen(3,'3g.bmp').

scoreGreen(4,'4g.bmp').

scoreGreen(5,'5g.bmp').

scoreGreen(6,'6g.bmp').

scoreGreen(7,'7g.bmp').

scoreGreen(8,'8g.bmp').

scoreGreen(9,'9g.bmp').

scoreGreen(10,'10g.bmp').

scoreGreen(11,'11g.bmp').

scoreGreen(12,'12g.bmp').

scoreGreen(13,'13g.bmp').

scoreGreen(14,'14g.bmp').

scoreGreen(15,'15g.bmp').

scoreGreen(16,'16g.bmp').

scoreGreen(17,'17g.bmp').

scoreGreen(18,'18g.bmp').

scoreGreen(19,'19g.bmp').

scoreGreen(20,'20g.bmp').

scoreGreen(21,'21g.bmp').

scoreGreen(22,'22g.bmp').

scoreGreen(23,'23g.bmp').

scoreGreen(24,'24g.bmp').

scoreGreen(25,'25g.bmp').

%%% the points in the board %%%

%p((X,Y),Up,Down,Right,Left).

%X: queue number

%Y: line number

%(-1): no exist

%1: full

%0: empty

p((1,1),-1,0,0,-1).

p((2,1),-1,0,0,0).

p((3,1),-1,0,0,0).

p((4,1),-1,0,0,0).

p((5,1),-1,0,0,0).

p((6,1),-1,0,-1,0).

p((1,2),0,0,0,-1).

p((2,2),0,0,0,0).

p((3,2),0,0,0,0).

p((4,2),0,0,0,0).

p((5,2),0,0,0,0).

p((6,2),0,0,-1,0).

p((1,3),0,0,0,-1).

p((2,3),0,0,0,0).

p((3,3),0,0,0,0).

p((4,3),0,0,0,0).

p((5,3),0,0,0,0).

p((6,3),0,0,-1,0).

p((1,4),0,0,0,-1).

p((2,4),0,0,0,0).

p((3,4),0,0,0,0).

p((4,4),0,0,0,0).

p((5,4),0,0,0,0).

p((6,4),0,0,-1,0).

p((1,5),0,0,0,-1).

p((2,5),0,0,0,0).

p((3,5),0,0,0,0).

p((4,5),0,0,0,0).

p((5,5),0,0,0,0).

p((6,5),0,0,-1,0).

p((1,6),0,-1,0,-1).

p((2,6),0,-1,0,0).

p((3,6),0,-1,0,0).

p((4,6),0,-1,0,0).

p((5,6),0,-1,0,0).

p((6,6),0,-1,-1,0).

start:-

initGame,

openWindow.

openWindow:-

new(W,window('dots and boxes', size(900,800))),

menu(W).

menu(W):-

new(B,bitmap('reka.bmp')),

send(W,background,B),

new(HftH,bitmap('playvsfriend.bmp')),

send(W,display,HftH,point(300,310)),

new(HftC\_easy,bitmap('playvscomputer\_easy.bmp')),

send(W,display,HftC\_easy,point(210,410)),

new(HftC\_hard,bitmap('playvscomputer\_hard.bmp')),

send(W,display,HftC\_hard,point(210,510)),

new(R,bitmap('Rules.bmp')),

send(W,display,R,point(385,610)),

new(A,bitmap('About.bmp')),

send(W,display,A,point(379,705)),

send(W,open),

send(R,recogniser,click\_gesture(left,'',single,

and(message(@prolog,openRules,W)))),

send(A,recogniser,click\_gesture(left,'',single,

and(message(@prolog,openAbout,W)))),

send(HftH,recogniser,click\_gesture(left,'',single,

and(message(@prolog,openPlay\_HftH,W)))),

send(HftC\_easy,recogniser,click\_gesture(left,'',single,

and(message(@prolog,openPlay\_HftCEasy,W)))),

send(HftC\_hard,recogniser,click\_gesture(left,'',single,

and(message(@prolog,openPlay\_HftCHard,W)))).

% the back button pressed - return to the menu.

back\_deleteOld(W):-

send(W,clear),

initGame,

menu(W).

% the playVSfriend button pressed - open this game kind.

openPlay\_HftH(W):-

initPlayers(humanfthuman),

send(W,clear),

new(PR,bitmap('play\_reka\_cleen.bmp')),

send(W,background,PR),

new(D,bitmap('dotsNboxes.bmp')),

send(W,display,D,point(16,3)),

new(Back,bitmap('backTOmenu.gif')),

send(W,display,Back,point(700,660)),

send(Back,recogniser,click\_gesture(left,'',single,

and(message(@prolog,back\_deleteOld,W)))),

addPointsToBored(W,1),

drawHLines(W,1),

drawVLines(W,1),

showPlayerTurn(W),

showScores(W),

squaresList(FullSquaresList),

showSquares(W,FullSquaresList),

send(W,open),

playMove(W).

% the playVScomputer\_easylevel button pressed - open this game kind.

openPlay\_HftCEasy(W):-

initPlayers(humanftcomputer\_easy),

send(W,clear),

new(PR,bitmap('play\_reka\_cleen.bmp')),

send(W,background,PR),

new(D,bitmap('dotsNboxes.bmp')),

send(W,display,D,point(16,3)),

new(Back,bitmap('backTOmenu.gif')),

send(W,display,Back,point(700,660)),

send(Back,recogniser,click\_gesture(left,'',single,

and(message(@prolog,back\_deleteOld,W)))),

addPointsToBored(W,1),

drawHLines(W,1),

drawVLines(W,1),

showPlayerTurn(W),

showScores(W),

squaresList(FullSquaresList),

showSquares(W,FullSquaresList),

send(W,open),

playMove(W).

% the playVScomputer\_hardlevel button pressed - open this game kind.

openPlay\_HftCHard(W):-

initPlayers(humanftcomputer\_hard),

send(W,clear),

new(PR,bitmap('play\_reka\_cleen.bmp')),

send(W,background,PR),

new(D,bitmap('dotsNboxes.bmp')),

send(W,display,D,point(16,3)),

new(Back,bitmap('backTOmenu.gif')),

send(W,display,Back,point(700,660)),

send(Back,recogniser,click\_gesture(left,'',single,

and(message(@prolog,back\_deleteOld,W)))),

addPointsToBored(W,1),

drawHLines(W,1),

drawVLines(W,1),

showPlayerTurn(W),

showScores(W),

squaresList(FullSquaresList),

showSquares(W,FullSquaresList),

send(W,open),

playMove(W).

openRules(W):-

send(W,clear),

new(RR,bitmap('rulesreka.bmp')),

send(W,background,RR),

send(W,open),

new(Back,bitmap('backTOmenu.gif')),

send(W,display,Back,point(705,660)),

send(Back,recogniser,click\_gesture(left,'',single,

and(message(@prolog,back\_deleteOld,W)))).

openAbout(W):-

send(W,clear),

new(AR,bitmap('aboutreka.bmp')),

send(W,background,AR),

send(W,open),

new(Back,bitmap('backTOmenu.gif')),

send(W,display,Back,point(705,660)),

send(Back,recogniser,click\_gesture(left,'',single,

and(message(@prolog,back\_deleteOld,W)))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%% the computer moves %%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create a list of all the possible lines- horizontal and vertical

%that the computer could choose (no one choose them yet).

createListOfAllPossibleMoves(MovesList):-

pointsList(Points),

findall(move(horizontal,X,Y),member(p((X,Y),\_,\_,0,\_),Points),Lhorizontal),

findall(move(vertical,X,Y) ,member(p((X,Y),\_,0,\_,\_),Points),Lvertical),

append(Lhorizontal,Lvertical,MovesList),

writeln(['moves list : ',MovesList]).

%%%%player AI%%%%

%the move that the computer will do:

%for easy level - choose random move.

chooseMoveForEasyAI(W):-

createListOfAllPossibleMoves(ML),

length(ML,Len),

Index is random(Len)+1,

nth1(Index,ML,move(Dir,X,Y)),

makeAIMove(W,Dir,X,Y).

%for hard level

%%%for close square%%%

%for horizontal close return Score=1.

horizontalAtXYasTopLeftCorner(X,Y,Score):-

pointsList(PointsList),

member(p((X,Y),\_,1,0,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),

Score is 1.

horizontalAtXYasTopLeftCorner(\_,\_,0).

horizontalAtXYasDownLeftCorner(X,Y,Score):-

pointsList(PointsList),

member(p((X,Y),1,\_,0,\_),PointsList),

XPlus1 is X+1,

YMinus1 is Y-1,

member(p((XPlus1,YMinus1),\_,1,\_,1),PointsList),

Score is 1.

horizontalAtXYasDownLeftCorner(\_,\_,0).

%for vertical close return Score=1.

verticalAtXYasTopLeftCorner(X,Y,Score):-

pointsList(PointsList),

member(p((X,Y),\_,0,1,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),

Score is 1.

verticalAtXYasTopLeftCorner(\_,\_,0).

verticalAtXYasTopRightCorner(X,Y,Score):-

pointsList(PointsList),

member(p((X,Y),\_,0,\_,1),PointsList),

XMinus1 is X-1,

YPlus1 is Y+1,

member(p((XMinus1,YPlus1),1,\_,1,\_),PointsList),

Score is 1.

verticalAtXYasTopRightCorner(\_,\_,0).

%make a list of all the lines that will close a squares.

closeSquaresList(CloseList):-

horizontalCloseSquaresList(CloseHlist),

verticalCloseSquaresList(CloseVlist),

append(CloseHlist,CloseVlist,CloseList),

writeln(['close list : ',CloseList]).

%make a list of all the horizontal lines that will close a squares.

horizontalCloseSquaresList(CloseHlist):-

createListOfAllPossibleMoves(ML),

findall(close(horizontal,X,Y),( member(move(horizontal,X,Y),ML),

horizontalAtXYasTopLeftCorner(X,Y,1);

horizontalAtXYasDownLeftCorner(X,Y,1) ),

CloseHlist).

%make a list of all the vertical lines that will close a squares.

verticalCloseSquaresList(CloseVlist):-

createListOfAllPossibleMoves(ML),

findall(close(vertical,X,Y),( member(move(vertical,X,Y),ML),

verticalAtXYasTopLeftCorner(X,Y,1);

verticalAtXYasTopRightCorner(X,Y,1) ),

CloseVlist).

%%%for 3 Line- like this, the human player will close a square%%%

%for horizontal 3 line return Flag=1.

line3HorizontalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,1,0,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,0),PointsList),

Flag is 1.

line3HorizontalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,1,0,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),0,\_,\_,1),PointsList),

Flag is 1.

line3HorizontalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,0,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),

Flag is 1.

line3HorizontalAtXYasTopLeftCorner(\_,\_,0).

line3HorizontalAtXYasDownLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),1,\_,0,\_),PointsList),

XPlus1 is X+1,

YMinus1 is Y-1,

member(p((XPlus1,YMinus1),\_,1,\_,0),PointsList),

Flag is 1.

line3HorizontalAtXYasDownLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),1,\_,0,\_),PointsList),

XPlus1 is X+1,

YMinus1 is Y-1,

member(p((XPlus1,YMinus1),\_,0,\_,1),PointsList),

Flag is 1.

line3HorizontalAtXYasDownLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),0,\_,0,\_),PointsList),

XPlus1 is X+1,

YMinus1 is Y-1,

member(p((XPlus1,YMinus1),\_,1,\_,1),PointsList),

Flag is 1.

line3HorizontalAtXYasDownLeftCorner(\_,\_,0).

%for vertical 3 line return Flag=1.

line3VerticalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,1,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),0,\_,\_,1),PointsList),

Flag is 1.

line3VerticalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,1,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,0),PointsList),

Flag is 1.

line3VerticalAtXYasTopLeftCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,0,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),

Flag is 1.

line3VerticalAtXYasTopLeftCorner(\_,\_,0).

line3VerticalAtXYasTopRightCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,\_,1),PointsList),

XMinus1 is X-1,

YPlus1 is Y+1,

member(p((XMinus1,YPlus1),0,\_,1,\_),PointsList),

Flag is 1.

line3VerticalAtXYasTopRightCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,\_,1),PointsList),

XMinus1 is X-1,

YPlus1 is Y+1,

member(p((XMinus1,YPlus1),1,\_,0,\_),PointsList),

Flag is 1.

line3VerticalAtXYasTopRightCorner(X,Y,Flag):-

pointsList(PointsList),

member(p((X,Y),\_,0,\_,0),PointsList),

XMinus1 is X-1,

YPlus1 is Y+1,

member(p((XMinus1,YPlus1),1,\_,1,\_),PointsList),

Flag is 1.

line3VerticalAtXYasTopRightCorner(\_,\_,0).

%make a list of all the 3 lines that if the computer choose them,

%the human player will close a square.

line3List(Line3List):-

line3horizontalList(Line3\_hList),

line3verticalList(Line3\_vList),

append(Line3\_hList,Line3\_vList,Line3List).

%make a list of all the possible moves that if the computer choose them,

%the human player wont close a square.

not\_line3List(Not\_line3List):-

createListOfAllPossibleMoves(ML),

line3List(Line3List),

findall(not\_line3(LineType,X,Y),( member(move(LineType,X,Y),ML),

not(member(line3(LineType,X,Y),Line3List)) ),

Not\_line3List),

writeln(['not line 3 list : ',Not\_line3List]).

%make a list of all the horizontal lines that will close a squares.

line3horizontalList(Line3\_hList):-

createListOfAllPossibleMoves(ML),

findall(line3(horizontal,X,Y),( member(move(horizontal,X,Y),ML),

line3HorizontalAtXYasTopLeftCorner(X,Y,1);

line3HorizontalAtXYasDownLeftCorner(X,Y,1) ),

Line3\_hList).

%make a list of all the vertical lines that will close a squares.

line3verticalList(Line3\_vList):-

createListOfAllPossibleMoves(ML),

findall(line3(vertical,X,Y),( member(move(vertical,X,Y),ML),

line3VerticalAtXYasTopLeftCorner(X,Y,1);

line3VerticalAtXYasTopRightCorner(X,Y,1) ),

Line3\_vList).

%check if he can close a square this turn.

% and choose this line!!

chooseMoveForHardAI(W):-

closeSquaresList(CloseList),

length(CloseList,Len),

Len\=0,

Index is random(Len)+1,

nth1(Index,CloseList,close(Dir,X,Y)),

makeAIMove(W,Dir,X,Y).

%check if there is a line that if he choose it,

%the human player will close a square,

% and choose another line!!

chooseMoveForHardAI(W):-

not\_line3List(Not\_line3List),

length(Not\_line3List,Len),

Len\=0,

Index is random(Len)+1,

nth1(Index,Not\_line3List,not\_line3(LineType,X,Y)),

makeAIMove(W,LineType,X,Y).

%else- random a move.

chooseMoveForHardAI(W):-

createListOfAllPossibleMoves(ML),

length(ML,Len),

Index is random(Len)+1,

nth1(Index,ML,move(Dir,X,Y)),

makeAIMove(W,Dir,X,Y).

%make the comuter move according to the line that chosen.

makeAIMove(W,horizontal,X,Y):-

hLinePressed(W,X,Y).

makeAIMove(W,vertical,X,Y):-

vLinePressed(W,X,Y).

%%%%playMove(W)%%%%

%doing the computer move according to the gameType.

playMove(W):-

send(timer(0.3), delay),

players(PlayersList),

writeln(['playmove - the players : ',PlayersList]),

% players([p(GameType, Score, Color)|\_]),

players([p(aiEasy,\_,\_)|\_]),

chooseMoveForEasyAI(W).

playMove(W):-

send(timer(0.3), delay),

players(PlayersList),

writeln(['playmove - the players : ',PlayersList]),

% players([p(GameType, Score, Color)|\_]),

players([p(aiHard,\_,\_)|\_]),

chooseMoveForHardAI(W).

playMove(\_).

%player human.

%do nothing - wait for press ....

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%% init facts of game%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%init all the dynamic lists in the game.

initGame:-

retractall(pointsList(\_)),

retractall(players(\_)),

retractall(squaresList(\_)),

assert(squaresList([])),

assert(players([])),

assert(pointsList([])),

listPoints.

%init the points list.

listPoints:-

findall(p((X,Y),Up,Down,Right,Left),p((X,Y),Up,Down,Right,Left),L),

retract(pointsList(\_)),

assert(pointsList(L)).

%initPlayers(GameType)

%init the Players list according to the GameType- the player choose.

initPlayers(humanfthuman):-

% type , score, signColor

retract(players(\_)),

append([p(human,0,red)],[p(human,0,green)],PlayersList),

assert(players(PlayersList)).

initPlayers(humanftcomputer\_easy):-

% type , score, signColor

retract(players(\_)),

append([p(aiEasy,0,red)],[p(human,0,green)],PlayersList),

assert(players(PlayersList)).

initPlayers(humanftcomputer\_hard):-

% type , score, signColor

retract(players(\_)),

append([p(aiHard,0,red)],[p(human,0,green)],PlayersList),

assert(players(PlayersList)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%% show the facts on the board %%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%someone just closed a square- put a red square in this place.

showSingleSquare(W,X,Y,red):-

new(Xg,bitmap('Xred.bmp')),

LocX is 188 + (X-1) \* 75,

LocY is 258 + (Y-1) \* 75,

send(W,display,Xg,point(LocX,LocY)).

%someone just closed a square- put a green square in this place.

showSingleSquare(W,X,Y,green):-

new(Xr,bitmap('Xgreen.bmp')),

LocX is 188 + (X-1) \* 75,

LocY is 258 + (Y-1) \* 75,

send(W,display,Xr,point(LocX,LocY)).

%no one closed a square in this turn

showSquares(\_,[]).

%someone just closed a square, check if the game end.

showSquares(W,[sq(X,Y,Sign)|RestOfSquares]):-

showSingleSquare(W,X,Y,Sign),

showSquares(W,RestOfSquares),

endGame(W).

%show the scores of the players in this turn

showScores(W):-

players(PlayersList),

writeln(PlayersList),

member(p(\_,ScoreRed,red),PlayersList),

member(p(\_,ScoreGreen,green),PlayersList),

showScoresBoxesRed(W,ScoreRed),

showScoresBoxesGreen(W,ScoreGreen),

squaresList(SquaresList),

writeln(SquaresList).

%put the appropriate picture to the red scores in this turn.

showScoresBoxesRed(W,ScoreRed):-

scoreRed(ScoreRed,Pic),

new(S,bitmap(Pic)),

send(W,display,S,point(665,300)).

%put the appropriate picture to the green scores in this turn.

showScoresBoxesGreen(W,ScoreGreen):-

scoreGreen(ScoreGreen,Pic),

new(S,bitmap(Pic)),

send(W,display,S,point(665,375)).

%put the name of the player -player1/2, that his turn is now.

showPlayerTurn(W):-

players(PlayersList),

nth1(1,PlayersList,p(\_,\_,Sign)),

showPlayerTurn1(W,Sign).

showPlayerTurn1(W,red):-

new(Ba,bitmap('player1red.bmp')),

send(W,display,Ba,point(435,76)).

showPlayerTurn1(W,green):-

new(Ba,bitmap('player2green.bmp')),

send(W,display,Ba,point(435,76)).

%the game is not finish in this turn.

endGame(\_):-

totalScore(Total),

Total \= 25.

%the game is finish in this turn.

endGame(W):-

totalScore(Total),

Total is 25,

winner(W).

%check who is the winner and put the appropriate player name.

winner(W):-

players(PL),

nth1(1,PL,p(\_,S1,green)),

nth1(2,PL,p(\_,S2,red)),

S1>S2,

new(D,bitmap('player2Win.bmp')),

send(W,display,D,point(237,155)).

winner(W):-

players(PL),

nth1(1,PL,p(\_,S1,red)),

nth1(2,PL,p(\_,S2,green)),

S1>S2,

new(D,bitmap('player1Win.bmp')),

send(W,display,D,point(237,155)).

winner(W):-

players(PL),

nth1(1,PL,p(\_,S1,green)),

nth1(2,PL,p(\_,S2,red)),

S2>S1,

new(D,bitmap('player1Win.bmp')),

send(W,display,D,point(237,155)).

winner(W):-

players(PL),

nth1(1,PL,p(\_,S1,red)),

nth1(2,PL,p(\_,S2,green)),

S2>S1,

new(D,bitmap('player2Win.bmp')),

send(W,display,D,point(237,155)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%% draw the board game %%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%add all the points to the board.

addPointsToBored(\_,7).

addPointsToBored(W,Line):-

putPoints(W,1,Line),

NewLine is Line + 1,

addPointsToBored(W,NewLine).

%put one point on the board.

drawPoint(W,X,Y):-

LocX is 172 + (X-1) \* 75,

LocY is 241 + (Y-1) \* 75,

new(P,bitmap('dot.bmp')),

send(W,display,P,point(LocX,LocY)).

putPoints(\_,7,\_).

putPoints(W,X,Y):-

pointsList(PointsList),

member(p((X,Y),\_,\_,\_,\_),PointsList),

drawPoint(W,X,Y),

NewX is X + 1,

putPoints(W,NewX,Y).

%draw the horizontal lines on the board- till line 7.

drawHLines(\_,7).

drawHLines(W,Line):-

putHorizontalLines(W,1,Line),

NewLine is Line + 1,

drawHLines(W,NewLine).

%draw the horizontal lines on the board- till queue 6.

putHorizontalLines(\_,6,\_).

putHorizontalLines(W,X,Y):-

pointsList(PointsList),

member(p((X,Y),\_,\_,LineStatus,\_),PointsList),

drawHLine(W,X,Y,LineStatus),

NewX is X + 1,

putHorizontalLines(W,NewX,Y).

% 0 means a gray line - no one chose this line yet ....

% make the lines pressed- if someone will press them

% the hLinePressed action will be work.

drawHLine(W,X,Y,0):-

LocX is 188 + (X-1) \* 75,

LocY is 245 + (Y-1) \* 75,

new(B,bitmap('widthLine\_gray.bmp')),

send(W,display,B,point(LocX,LocY)),

send(B,recogniser,click\_gesture(left,'',single,

and(message(@prolog,hLinePressed,W,X,Y)))).

% 1 means a red\green line - someone already chose this line ....

drawHLine(W,X,Y,1):-

players(PlayersList),

nth1(1,PlayersList,p(\_,\_,SignColor)),

drawHLineColor(W,X,Y,1,SignColor).

drawHLineColor(W,X,Y,1,red):-

LocX is 188 + (X-1) \* 75,

LocY is 245 + (Y-1) \* 75,

new(B,bitmap('widthLine\_red.bmp')),

send(W,display,B,point(LocX,LocY)).

drawHLineColor(W,X,Y,1,green):-

LocX is 188 + (X-1) \* 75,

LocY is 245 + (Y-1) \* 75,

new(B,bitmap('widthLine\_green.bmp')),

send(W,display,B,point(LocX,LocY)).

%someone just pressed on horizontal line.

hLinePressed(W,X,Y):-

%check:

append([X],[Y],Z),

writeln(Z),

%

pointsList(PointsList),

member(p((X,Y),Up,Down,0,Left),PointsList),

XPlus1 is X+1,

member(p((XPlus1,Y),Up1,Down1,Right1,0),PointsList),

select(p((X,Y),Up,Down,0,Left),PointsList,NL),

select(p((XPlus1,Y),Up1,Down1,Right1,0),NL,NL2),

append([p((X,Y),Up,Down,1,Left)],NL2,NL3),

append([p((XPlus1,Y),Up1,Down1,Right1,1)],

NL3,NewPointsList),

retract(pointsList(\_)),

assert(pointsList(NewPointsList)),

member(p((X,Y),\_,\_,LineStatus,\_),NewPointsList),

drawHLine(W,X,Y,LineStatus),

totalScoreForHorizontalLine(X,Y,Score),

playTurn(Score),

showPlayerTurn(W),

showScores(W),

squaresList(FullSquaresList),

showSquares(W,FullSquaresList),

playMove(W).

%draw the vertical lines on the board- till line 6.

drawVLines(\_,6).

drawVLines(W,Line):-

putVerticalLines(W,1,Line),

NewLine is Line + 1,

drawVLines(W,NewLine).

%draw the vertical lines on the board- till queue 6.

putVerticalLines(\_,7,\_).

putVerticalLines(W,X,Y):-

pointsList(PointsList),

member(p((X,Y),\_,LineStatus,\_,\_),PointsList),

drawVLine(W,X,Y,LineStatus),

NewX is X + 1,

putVerticalLines(W,NewX,Y).

% 0 means a gray line - no one chose this line yet ....

% make the lines pressed- if someone will press them

% the vLinePressed action will be work.

drawVLine(W,X,Y,0):-

LocX is 175 + (X-1) \* 75,

LocY is 257 + (Y-1) \* 75,

new(B,bitmap('hightLine\_gray.bmp')),

send(W,display,B,point(LocX,LocY)),

send(B,recogniser,click\_gesture(left,'',single,

and(message(@prolog,vLinePressed,W,X,Y)))).

% 1 means a red\green line - someone already chose this line ....

drawVLine(W,X,Y,1):-

players(PlayersList),

nth1(1,PlayersList,p(\_,\_,SignColor)),

drawVLineColor(W,X,Y,1,SignColor).

drawVLineColor(W,X,Y,1,red):-

LocX is 175 + (X-1) \* 75,

LocY is 257 + (Y-1) \* 75,

new(B,bitmap('hightLine\_red.bmp')),

send(W,display,B,point(LocX,LocY)).

drawVLineColor(W,X,Y,1,green):-

LocX is 175 + (X-1) \* 75,

LocY is 257 + (Y-1) \* 75,

new(B,bitmap('hightLine\_green.bmp')),

send(W,display,B,point(LocX,LocY)).

%someone just pressed on vertical line.

vLinePressed(W,X,Y):-

%check:

append([X],[Y],Z),

writeln(Z),

%

pointsList(PointsList),

member(p((X,Y),Up,0,Right,Left),PointsList),

YPlus1 is Y+1,

member(p((X,YPlus1),0,Down1,Right1,Left1),PointsList),

select(p((X,Y),Up,0,Right,Left),PointsList,NL),

select(p((X,YPlus1),0,Down1,Right1,Left1),NL,NL2),

append([p((X,Y),Up,1,Right,Left)],NL2,NL3),

append([p((X,YPlus1),1,Down1,Right1,Left1)],

NL3,NewPointsList),

retract(pointsList(\_)),

assert(pointsList(NewPointsList)),

member(p((X,Y),\_,LineStatus,\_,\_),NewPointsList),

drawVLine(W,X,Y,LineStatus),

totalScoreForVerticalLine(X,Y,Score),

playTurn(Score),

showPlayerTurn(W),

showScores(W),

squaresList(FullSquaresList),

showSquares(W,FullSquaresList),

playMove(W).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% calculate the scores and the squares %%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% retuen the score

% Score will be 1 if the square is closed and 0 if not.

checkFullSquareAtXYasTopLeftCorner(X,Y,Score):-

pointsList(PointsList),

member(p((X,Y),\_,1,1,\_),PointsList),

XPlus1 is X+1,

YPlus1 is Y+1,

member(p((XPlus1,YPlus1),1,\_,\_,1),PointsList),

Score is 1.

checkFullSquareAtXYasTopLeftCorner(\_,\_,0).

%no one closed a square in this turn - do nothing.

addToFullSquareList(\_,\_,0).

%someone just closed a square - add to the squres list that square.

addToFullSquareList(X,Y,1):-

players(PlayersList),

nth1(1,PlayersList,p(\_ , \_, SignColor)),

squaresList(OldSquareList),

append([sq(X,Y,SignColor)],OldSquareList,NewSquareList),

retract(squaresList(\_)),

assert(squaresList(NewSquareList)).

%if someone just closed a square - add to the squres list that square,

%else- do nothing.

%and return the Total- the scores that need to be add.

%if someone press on Horizontal line.

totalScoreForHorizontalLine(X,Y,Total):-

checkFullSquareAtXYasTopLeftCorner(X,Y,Score1),

addToFullSquareList(X,Y,Score1),

YMinus1 is Y - 1,

checkFullSquareAtXYasTopLeftCorner(X,YMinus1,Score2),

addToFullSquareList(X,YMinus1,Score2),

Total is Score1 + Score2.

%if someone press on vertical line.

totalScoreForVerticalLine(X,Y,Total):-

checkFullSquareAtXYasTopLeftCorner(X,Y,Score1),

addToFullSquareList(X,Y,Score1),

XMinus1 is X - 1,

checkFullSquareAtXYasTopLeftCorner(XMinus1,Y,Score2),

addToFullSquareList(XMinus1,Y,Score2),

Total is Score1 + Score2.

%calculate the total Scores = the num of squares

%of player1+player2 together.

totalScore(Total):-

players(PL),

nth1(1,PL,p(\_,S1,\_)),

nth1(2,PL,p(\_,S2,\_)),

Total is S1+S2.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%% Main game - handle single turn functions %%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%no one close a square - just switch the players turn.

playTurn(0):-

players(PlayersList),

reverse(PlayersList,NewPlayersList),

retract(players(\_)),

assert(players(NewPlayersList)).

%someone just closed a square !!!

%switch the players turn and updating the score of this player.

playTurn(Score):-

players(PlayersList),

nth1(1,PlayersList,p(PlayerType1 , Score1, SignColor1)),

nth1(2,PlayersList,P2),

NewScore is Score1 + Score,

retract(players(\_)),

append([p(PlayerType1, NewScore ,SignColor1 )],[P2],NewPlayersList),

assert(players(NewPlayersList)).