**שש בשBackgammon**



מגיש: שחר ברנסון  
ת.ז*: 208608414*  
מורה: אורנה אברך  
בית הספר: ריאלי בית בירם

תוכן עניינים

[פרולוג 3](#_Toc447136158)

[מהלך המשחק 3](#_Toc447136159)

[סיום המשחק 4](#_Toc447136160)

[קהל היעד 5](#_Toc447136161)

[ייצוג בסיס הידע בתוכנית פרולוג 6](#_Toc447136162)

[נתונים 6](#_Toc447136163)

[חוקי אתחול ומחיקה 7](#_Toc447136164)

[חוקים ראשיים הקשורים בניהול המשחק 7](#_Toc447136174)

[חוקי עזר כלליים 8](#_Toc447136200)

[חוקים גרפיים 9](#_Toc447136222)

[חוקים הקשורים לאלגוריתם 9](#_Toc447136238)

[האלגוריתם 10](#_Toc447136251)

[היורסטיקה 10](#_Toc447136238)

[גרפיקה וממשק משתמש 11](#_Toc447136252)

[הקוד 12](#_Toc447136253)

[שימוש במערכת והרצתה 58](#_Toc447136254)

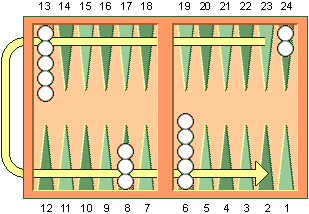
[נספחים 63](#_Toc447136263)

פרולוג

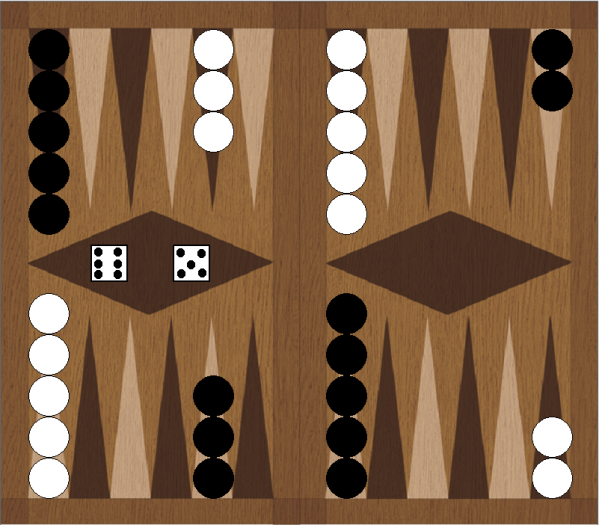
המשחק **שש בש** הוא משחק מזל, חשיבה ואסטרטגיה לשני שחקנים המכיל לוח המחולק לצד עליון ותחתון שבכל צד 12 טורים שעליהם מצויירים משולשים המתחלפים כל פעם בין בהיר לכהה, 15 כלים לכל שחקן ושתי קוביות של מס' מ 1-6.

מטרת המשחק היא להוציא את כל הכלים שברשותכם מהלוח בשביל לנצח.

**מהלך המשחק**

כל שחקן זורק קובייה לראות מי מתחיל, השחקן שקיבל את המס' הגבוה ביותר בקובייה הוא זה שמתחיל. במקרה של תיקו, זורקים את הקוביות עוד פעם.  
כל שחקן זורק בתורו שתי קוביות ומזיז את הכלים שלו. השחקן מזיז את הכלים שלו על פי כל קובייה בנפרד. השחקן חייב להזיז את הכלים בכל מצב שהלוח מאפשר לו והוא אינו רשאי לנוע אחורה. לוח המאפשר הזזה הוא לוח שבו יש טור ריק, כלים שחורים אחרים או כלי בודד של שחקן היריב בו יוכל לבצע אכילה.

בסיס שחקן 2

  
  
  
  
  
  
  
  
  
  
  
כדי לנצח במשחק צריך השחקן להכניס את כל הכלים שלו לבסיס. לאחר שהגיע למצב זה יכול השחקן להתחיל להוציא את כליו מחוץ ללוח, במידה ולאחר שהתחיל שחקן להוציא את כליו מחוץ ללוח בוצעה אכילה על אחת מכליו לא יוכל השחקן להוציא כלים מחוץ ללוח עד אשר יוחזר הכלי לבסיס.

בסיס שחקן 1

צד הבסיסים נקבע לפני המשחק איפה השחקנים מעדיפים לשחק: שמאל או ימין.

כאשר שחקן מבצע אכילה על כלי של שחקן יריב, יזרוק השחקן היריב את הקוביות בתורו, בהתאם לתוצאות הקוביות יוכל השחקן להכניס את הכלי לבסיס השחקן שאכל, הספירה מבוצעת החל מהטור הראשון עד השישי או לסירוגין מטור 24 עד טור 19, כלומר אם יזרוק השחקן 1:3 (בקובייה אחת 1 ובקובייה שנייה 3) שחקן 2 יוכל להכניס את הכלי לטור 1 או 3 ושחקן 1לטורים 22 ו- 24. שחקן שלא הצליח לשחרר את חיילו הלכוד, יוותר על תורו. אין הגבלה על מספר החיילים הלכודים.

במקרה של קוביות זהות (דאבל), השחקן מבצע 4 הזזות באותו תור במקום 2.

**סיום המשחק**

המשחק יכול להיגמר אך ורק בניצחון של אחד מהשחקנים.  
שחקן מנצח אם הגיע לבסיס והוציא ראשון את כל הכלים שברשותו מהלוח.

קהל היעד

העבודה פונה לכל משתמש, למעשה המשחק מיועד לכל אחד שאוהב את המשחק ויכול לשחק מול המחשב ללא צורך לשחק מול יריב אנושי.

ייצוג בסיס הידע בתוכנית פרולוג

**נתונים**

הנתונים בתוכנה נשמרים ע"י אובייקטים דינמיים שנוצרים בכל פעם שהמשחק מופעל מחדש. האובייקטים השונים מתוארים בטבלה המצורפת.

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| pieces | Pos,Amount,Color,Top | האובייקט שומר את מיקום הכלים, כמות בעמודה, צבע הכלי והכלי העליון |
| dice1, dice2 | Pic,Num | האובייקט שומר את תמונת הקובייה והמספר על הקובייה |
| piecepic | Circle,Height,Pos,Color | האובייקט שומר את הכלי, גובה שלו בעמודה, מיקום הכלי וצבעו |
| marked | Circle,Amount | האובייקט שומר את העיגול המסומן וכמות המסומנים |
| turn | Flag | האובייקט שומר את תור השחקן או המחשב |
| double | Flag,Flag | האובייקט שומר את מצב הדאבל |
| Check, check1, check2, check3, check4 | Flag | האובייקט שומר דגל כדי שהתוכנה לא תיכנס לאותה פעולה |
| grade | Grade,Pos,Dice | האובייקט שומר את ציון המהלך,מיקום הכלי ומספר קובייה |
| side | Side | האובייקט שומר את צד הבסיס |
| colors | Color,EColor | האובייקט שומר את צבע הכלים של השחקן וצבע הכלים של המחשב |

**חוקי אתחול ומחיקה**

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| retractor | \_ | מחיקת כל האובייקטים הדינמים |
| game | Side,Colors | אתחול כל האובייקטים הדינמיים ומצב התחלתי של לוח |
| changing\_turn\_to\_computer | \_ | למחיקת תור של שחקן ומחיקת הקוביות בשביל להעביר לתור של המחשב |

**חוקים ראשיים הקשורים בניהול המשחק**

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| settings | \_ | חלון הגדרות המשחק, איזה צד הבסיס ומה צבע הכלים |
| game | Side,Colors | חלון המשחק,אתחול כל האובייקטים הדינמיים ומצב התחלתי של לוח |
| moveU | \_ | התחלת תור של שחקן |
| moveComputer | \_ | התחלת תור של מחשב |
| win |  | מצב מנצח של שחקן |
| lose |  | מצב מפסיד של שחקן |
| play\_again | Color | השחקן נשאל אם לשחק שוב או לצאת מהמשחק. מקבל צבע. |
| again | \_ | מחיקת חלון המשחק וקריאה למשחק חדש בהגדרות settings |
| exit | \_ | קריאה ל-retractor ומחיקת חלון המשחק |

**חוקי עזר כלליים**

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| diceroll | \_ | הגרלת שתי קוביות |
| starter | \_ | בודק מי מתחיל |
| mark | Top | מסמן את הכלי העליון בעמודה שלחצת עליו על מנת להזיזו |
| getNewPos | Dice | נותן את המיקום החדש של הכלי המסומן עם הקובייה שלחצת עליה |
| possible | Top Circle ,Amount, Pos,Cube, Other\_Cube, Dice Pic, Num | בודק את המצבים האפשריים במיקום החדש |
| options | Top Circle,Dice | נותן את המיקום החדש של הקובייה ושולח ל-states אם המצב אפשרי ול-showit כדי להזיז אותו |
| states | Top Circle, New Pos, Phase | בודק את המצבים האפשריים של המיקום החדש. כש- Phase הוא 1 אז הזזה רגילה וכשהוא 2 אז הוצאה החוצה |

**חוקים גרפיים**

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| game | \_ | אתחול מצב ההתחלתי של הלוח ומיקום הכלים בו |
| roll | Num,Dice | יוצר את הקוביות על חלון המשחק |
| possible | Top Circle ,Amount, Pos,Cube, Other\_Cube, Dice Pic, Num | בדיקה אם המצב אפשרי והזזה למיקום החדש של הכלי בתור השחקן |
| showit | Pos | הזזה למיקום החדש של הכלי בתור המחשב |
| nomove | \_ | הצגת הטקסט אין הזזה כשאי אפשר לזוז |

**חוקים הקשורים לאלגוריתם**

|  |  |  |
| --- | --- | --- |
| **שם אובייקט** | **נתונים** | **הסבר** |
| dragon | \_ | ניקוד מהלכים וביצוע הזזה |
| hiroshima | List,Dice | קבלת רשימה של מיקומים ושליחה ל-kraken בשביל ניקוד המיקום החדש ומספר קובייה |
| kraken | \_ | ניקוד המיקום החדש |
| nikud(1,2,3,4) | List,Grade | קבלת רשימה של מיקומים והחזרת הניקוד לפי קטגוריה מסוימת |

האלגוריתם

הבינה המלאכותית במשחק נכתבה באמצעות האלגוריתם ההיורסטיקה (Heuristic).

**היוריסטיקה**

פונקציית ההיוריסטית היא שיטת ניקוד לכל מיקום בלוח. ע"פ כמה קטגוריות הפונקציה מנקדת את המיקום החדש ולוקחת את המיקום החדש עם הציון הכי גבוה כדי להזיז אותו. בעקבות הסיבוכיות והמורכבות של משחק השש-בש, ההיורסטיקה יכולה לנקד רק את המהלך הבא של המחשב ולא לראות כמה צעדים קדימה. הפונקציה ההיורסטית בודקת את סכום הניקוד של כל מיקום ובודקת האם הסכום גבוה מקודמו כדי לקבל את המהלך הכי טוב לביצוע.  
  
הקטגוריות של הפונקציה ההיורסטית:  
ההפרש בין כמות הכלים הפתוחים בין השחקן למחשב \* 300 נק'  
ההפרש בין כמות הכלים האכולים בין השחקן למחשב \* 1000 נק'   
ההפרש בין המרחקים של כל הכלים בין השחקן למחשב \* 50 נק' –   
קטגוריה זו בשביל ליצור חומה ולראות כמה מהלכים קדימה ולחסום את השחקן מלצאת.  
ההפרש בין כמות הכלים שבתוך הבסיס בין השחקן למחשב \* 100 נק'

ישנם עוד קטגוריות הנוגעות לגבי מצבים מסויימים:  
במצב של הוצאת הכלים החוצה ישנו ניקוד שונה לגבי מס' העמודה שממנה יוצא 300 או 600 נק'.  
במצב של סגירת בתים או אכילה ע"י שתי הקוביות ניקוד הנוסף הוא 600 נק'.  
במצב של סגירת בית ע"י קובייה אחת הניקוד הוא 500 נק'.  
  
המהלך עם הניקוד הכי גבוה הוא המהלך שיבוצע ע"י המחשב לפי שיטת ניקוד זו.  
החוקים המפעילים את האלגוריתם מפורטים בפרק "ייצוג הידע בתוכנית הפרולוג".

גרפיקה וממשק משתמש

* חלון הגדרות: בחלון ההגדרות תוכל להגדיר איך תרצה לשחק נגד המחשב. השחקן מתבקש לבחור באיזה צד ירצה לשחק (ימין, שמאל או שהמחשב יבחר באקראי), ולבחור את צבע הכלים של השחקנים מהרשימה.  
  אם השחקן אינו בוחר הוא יכול ללחוץ על Ok והברירת מחדל תהיה המצב הקיים של ההגדרות: צד ימין וצבעים שחור נגד לבן.
* חלון המשחק: במסך המשחק יש את הלוח, 15 הכלים של כל שחקן ושתי הקוביות. במסך זה מתבצע כל המשחק, ההזזות של הכלים ושתי הקוביות.

דוגמאות להמחשת חלון ההגדרות והמשחק בפרק "שימוש במערכת והרצתה".

הקוד

:-dynamic pieces/4. %pieces(Pos,Amount,Color,Top), Pos=Column,line ;Top=contains top circle in column

:-dynamic dice1/2. %dice1(Pic,Num) Num between 1 to 6

:-dynamic dice2/2. %dice2(Pic,Num)

:-dynamic piecepic/4. %piecepic(Circle,Height,Pos,Color)

:-dynamic marked/2. %marked(Circle,Amount)

:-dynamic turn/1. %turn(Turn) Turn=player,computer

:-dynamic double/2. %double(Flag,Flag), Flag=false,true

:-dynamic check/1. %check(Flag), Flag=false,true %checks if enters only 1 state in computer move

:-dynamic check1/1. %check1(Flag), Flag=false,true ;enters only one term in computer moves

:-dynamic check2/1. %check2(Flag), Flag=false,true ;checks if no move

:-dynamic check3/1. %check3(Flag), Flag=false,true ;if score to move is possible

:-dynamic check4/1. %check4(Flag), Flag=false,true ;for not clicking while no move

:-dynamic grade/3. %grade(Grade,Pos,Dice)

:-dynamic side/1. %side(Side), Side=right,left,random

:-dynamic colors/2. %colors(Color1,Color2)

%cube(Num,X,Y).

cube(1,120,325).

cube(2,230,325).

retractor:-

retractall(pieces(\_,\_,\_,\_)),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

retractall(piecepic(\_,\_,\_,\_)),

retractall(marked(\_,\_)),

retractall(turn(\_)),

retractall(check(\_)),

retractall(check1(\_)),

retractall(check2(\_)),

retractall(check3(\_)),

retractall(check4(\_)),

retractall(double(\_,\_)),

retractall(grade(\_,\_,\_)),

retractall(side(\_)),

retractall(colors(\_,\_)).

%bmp(Pic, PicName).

bmp(b, 'bgs7\_bitmaps/board.bmp').

bmp(1, 'bgs7\_bitmaps/dice1.bmp').

bmp(2, 'bgs7\_bitmaps/dice2.bmp').

bmp(3, 'bgs7\_bitmaps/dice3.bmp').

bmp(4, 'bgs7\_bitmaps/dice4.bmp').

bmp(5, 'bgs7\_bitmaps/dice5.bmp').

bmp(6, 'bgs7\_bitmaps/dice6.bmp').

%lineX(Pos,X,Side).

lineX(1,710,right).

lineX(2,655,right).

lineX(3,600,right).

lineX(4,545,right).

lineX(5,490,right).

lineX(6,435,right).

lineX(7,312,right).

lineX(8,257,right).

lineX(9,202,right).

lineX(10,147,right).

lineX(11,92,right).

lineX(12,37,right).

lineX(13,37,right).

lineX(14,92,right).

lineX(15,147,right).

lineX(16,202,right).

lineX(17,257,right).

lineX(18,312,right).

lineX(19,435,right).

lineX(20,490,right).

lineX(21,545,right).

lineX(22,600,right).

lineX(23,655,right).

lineX(24,710,right).

lineX(25,380,right).

lineX(0,360,right).

lineX(1,37,left).

lineX(2,92,left).

lineX(3,147,left).

lineX(4,202,left).

lineX(5,257,left).

lineX(6,312,left).

lineX(7,435,left).

lineX(8,490,left).

lineX(9,545,left).

lineX(10,600,left).

lineX(11,655,left).

lineX(12,710,left).

lineX(13,710,left).

lineX(14,655,left).

lineX(15,600,left).

lineX(16,545,left).

lineX(17,490,left).

lineX(18,435,left).

lineX(19,312,left).

lineX(20,257,left).

lineX(21,202,left).

lineX(22,147,left).

lineX(23,92,left).

lineX(24,37,left).

lineX(25,360,left).

lineX(0,380,left).

%lineY(Pos,Y).

lineY(1,1,610).

lineY(1,2,555).

lineY(1,3,500).

lineY(1,4,445).

lineY(1,5,390).

lineY(1,6,380).

lineY(1,7,370).

lineY(1,8,360).

lineY(1,9,350).

lineY(1,10,340).

lineY(1,11,330).

lineY(1,12,320).

lineY(1,13,310).

lineY(1,14,300).

lineY(1,15,290).

lineY(2,1,37).

lineY(2,2,92).

lineY(2,3,147).

lineY(2,4,202).

lineY(2,5,257).

lineY(2,6,267).

lineY(2,7,277).

lineY(2,8,287).

lineY(2,9,297).

lineY(2,10,307).

lineY(2,11,317).

lineY(2,12,327).

lineY(2,13,337).

lineY(2,14,347).

lineY(2,15,357).

lineY(3,1,265).

lineY(3,2,275).

lineY(3,3,285).

lineY(3,4,295).

lineY(3,5,305).

lineY(3,6,315).

lineY(3,7,325).

lineY(3,8,335).

lineY(3,9,345).

lineY(3,10,355).

lineY(3,11,365).

lineY(3,12,375).

lineY(3,13,385).

lineY(3,14,395).

lineY(3,15,405).

game(Side,Colors):-

retractor,

new(@window,window('backgammon',size(800,730))),

send(@window,background,colour(white)),

bmp(b, BMP), %calling bitmap of board

new(Chessboard,bitmap(BMP)),

send(@window,display,Chessboard,point(0,0)),

side\_decide(Side), %inserting side to dynamic

side(M), %calling it for circles positions

coloring(Colors), %inserting the colors to dynamic

colors(Color,EColor), %Color for player 1 and EColor for player 2

%starting positions of circles of the 2 players

new(B1,circle(55)),

send(B1,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B1,point(37,37));

(M=left, send(@window,display,B1,point(710,37))))),

new(B2,circle(55)),

send(B2,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B2,point(37,92));

(M=left, send(@window,display,B2,point(710,92))))),

new(B3,circle(55)),

send(B3,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B3,point(37,147));

(M=left, send(@window,display,B3,point(710,147))))),

new(B4,circle(55)),

send(B4,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B4,point(37,202));

(M=left, send(@window,display,B4,point(710,202))))),

new(B5,circle(55)),

send(B5,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B5,point(37,257));

(M=left, send(@window,display,B5,point(710,257))))),

new(W6,circle(55)),

send(W6,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W6,point(257,37));

(M=left, send(@window,display,W6,point(490,37))))),

new(W7,circle(55)),

send(W7,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W7,point(257,92));

(M=left, send(@window,display,W7,point(490,92))))),

new(W8,circle(55)),

send(W8,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W8,point(257,147));

(M=left, send(@window,display,W8,point(490,147))))),

new(B6,circle(55)),

send(B6,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B6,point(257,610));

(M=left, send(@window,display,B6,point(490,610))))),

new(B7,circle(55)),

send(B7,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B7,point(257,555));

(M=left, send(@window,display,B7,point(490,555))))),

new(B8,circle(55)),

send(B8,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B8,point(257,500));

(M=left, send(@window,display,B8,point(490,500))))),

new(W1,circle(55)),

send(W1,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W1,point(37,610));

(M=left, send(@window,display,W1,point(710,610))))),

new(W2,circle(55)),

send(W2,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W2,point(37,555));

(M=left, send(@window,display,W2,point(710,555))))),

new(W3,circle(55)),

send(W3,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W3,point(37,500));

(M=left, send(@window,display,W3,point(710,500))))),

new(W4,circle(55)),

send(W4,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W4,point(37,445));

(M=left, send(@window,display,W4,point(710,445))))),

new(W5,circle(55)),

send(W5,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W5,point(37,390));

(M=left, send(@window,display,W5,point(710,390))))),

new(B9,circle(55)),

send(B9,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B9,point(710,37));

(M=left, send(@window,display,B9,point(37,37))))),

new(B10,circle(55)),

send(B10,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B10,point(710,92));

(M=left, send(@window,display,B10,point(37,92))))),

new(W9,circle(55)),

send(W9,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W9,point(710,610));

(M=left, send(@window,display,W9,point(37,610))))),

new(W10,circle(55)),

send(W10,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W10,point(710,555));

(M=left, send(@window,display,W10,point(37,555))))),

new(B11,circle(55)),

send(B11,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B11,point(435,610));

(M=left, send(@window,display,B11,point(312,610))))),

new(B12,circle(55)),

send(B12,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B12,point(435,555));

(M=left, send(@window,display,B12,point(312,555))))),

new(B13,circle(55)),

send(B13,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B13,point(435,500));

(M=left, send(@window,display,B13,point(312,500))))),

new(B14,circle(55)),

send(B14,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B14,point(435,445));

(M=left, send(@window,display,B14,point(312,445))))),

new(B15,circle(55)),

send(B15,fill\_pattern,colour(Color)),

((M=right, send(@window,display,B15,point(435,390));

(M=left, send(@window,display,B15,point(312,390))))),

new(W11,circle(55)),

send(W11,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W11,point(435,37));

(M=left, send(@window,display,W11,point(312,37))))),

new(W12,circle(55)),

send(W12,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W12,point(435,92));

(M=left, send(@window,display,W12,point(312,92))))),

new(W13,circle(55)),

send(W13,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W13,point(435,147));

(M=left, send(@window,display,W13,point(312,147))))),

new(W14,circle(55)),

send(W14,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W14,point(435,202));

(M=left, send(@window,display,W14,point(312,202))))),

new(W15,circle(55)),

send(W15,fill\_pattern,colour(EColor)),

((M=right, send(@window,display,W15,point(435,257));

(M=left, send(@window,display,W15,point(312,257))))),

%asserting pieces to dynamic with their top circle of column

assert(pieces(1,2,EColor,W10)),

assert(pieces(6,5,Color,B15)),

assert(pieces(8,3,Color,B8)),

assert(pieces(12,5,EColor,W5)),

assert(pieces(13,5,Color,B5)),

assert(pieces(17,3,EColor,W8)),

assert(pieces(19,5,EColor,W15)),

assert(pieces(24,2,Color,B10)),

%asserting all piecepics and their positions in column

assert(piecepic(B1,1,13,Color)),

assert(piecepic(B2,2,13,Color)),

assert(piecepic(B3,3,13,Color)),

assert(piecepic(B4,4,13,Color)),

assert(piecepic(B5,5,13,Color)),

assert(piecepic(B6,1,8,Color)),

assert(piecepic(B7,2,8,Color)),

assert(piecepic(B8,3,8,Color)),

assert(piecepic(B9,1,24,Color)),

assert(piecepic(B10,2,24,Color)),

assert(piecepic(B11,1,6,Color)),

assert(piecepic(B12,2,6,Color)),

assert(piecepic(B13,3,6,Color)),

assert(piecepic(B14,4,6,Color)),

assert(piecepic(B15,5,6,Color)),

assert(piecepic(W1,1,12,EColor)),

assert(piecepic(W2,2,12,EColor)),

assert(piecepic(W3,3,12,EColor)),

assert(piecepic(W4,4,12,EColor)),

assert(piecepic(W5,5,12,EColor)),

assert(piecepic(W6,1,17,EColor)),

assert(piecepic(W7,2,17,EColor)),

assert(piecepic(W8,3,17,EColor)),

assert(piecepic(W9,1,1,EColor)),

assert(piecepic(W10,2,1,EColor)),

assert(piecepic(W11,1,19,EColor)),

assert(piecepic(W12,2,19,EColor)),

assert(piecepic(W13,3,19,EColor)),

assert(piecepic(W14,4,19,EColor)),

assert(piecepic(W15,5,19,EColor)),

%setting start values to the dynamics

assert(marked(\_,0)),

assert(double(false,false)),

assert(check(false)),

assert(check1(false)),

assert(check2(true)),

assert(check3(false)),

assert(check4(false)),

assert(grade(-10000,-10000,-10000)),

new(Text,text('Start!')),

send(Text, font, font(times, bold, 36)),

send(Text, colour(white)),

send(@window,display,Text,point(140,320)),

send(Text,recogniser,click\_gesture(left,'',single,message(@prolog,diceroll,Text))),

send(@window,open).

%rolling the dices

diceroll(Text):-

((dice1(L1,\_), dice2(L2,\_), free(L1), free(L2), retractall(dice1(\_,\_)), retractall(dice2(\_,\_)));true),

free(Text),

%rolling numbers of dices, E is Dice1 G is Dice2

random\_between(1,6,E),

random\_between(1,6,G),

write('Dices of who starts: ['), write(E:G), writeln(']'),

%keep rolling for the effect of rolling dices, rolling 5 times

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

roll(E,1),

roll(G,2),

starter.

%Player starts

starter:-

dice1(L1,E),

dice2(L2,G),E > G, new(Text1,text('Player starts!')),

send(Text1, font, font(times, bold, 18)),

send(@window,display,Text1,point(0,700)),

send(L1,recogniser,click\_gesture(left,'',single,message(@prolog,makelist\_of\_circles,1))),

send(L2,recogniser,click\_gesture(left,'',single,message(@prolog,makelist\_of\_circles,1))).

%Draw, reroll

starter:-

dice1(L1,E),

dice2(L2,G),

E is G,

new(Text2,text('Draw, reroll')),

send(Text2, font, font(times, bold, 18)),

send(@window,display,Text2,point(0,700)),

send(L1,recogniser,click\_gesture(left,'',single,message(@prolog,diceroll,Text2))),

send(L2,recogniser,click\_gesture(left,'',single,message(@prolog,diceroll,Text2))).

%Computer starts

starter:-

dice1(L1,E),

dice2(L2,G),

G > E, new(Text3,text('Computer starts!')),

send(Text3, font, font(times, bold, 18)),

send(@window,display,Text3,point(0,700)),

send(timer(0.5), delay),

free(L1), free(L2),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

makelist\_of\_circles(2).

%roll(Num,Cube) R can be 1 or 2 depends on which dice

roll(N,R):-

bmp(N, BMP),

cube(R,X,Y),

(R is 1, new(Dice1,bitmap(BMP)),

send(@window,display,Dice1,point(X,Y)),

assert(dice1(Dice1,N));

(R is 2, new(Dice2,bitmap(BMP)),

send(@window,display,Dice2,point(X,Y)),

assert(dice2(Dice2,N)))).

%makelist\_of\_circles(R) R is the one who starts

makelist\_of\_circles(R):-

colors(Color,\_),

findall(Circle1, piecepic(Circle1,\_,\_,Color), L1),

recogniser\_pieces(L1),

((R is 1, moveU);(R is 2, moveComputer)).

%adding recogniser to player's pieces (circles)

recogniser\_pieces([]).

recogniser\_pieces([H|T]):-

send(H,recogniser,click\_gesture(left,'',single,message(@prolog,mark,H))),

recogniser\_pieces(T).

%player's turn

moveU:-

%reseting dynamics for player's turn

retractall(turn(\_)),

assert(turn(player)),

double(false,false),

dice1(Dice1,\_),

dice2(Dice2,\_),

%removing dices pics and its former info

free(Dice1), free(Dice2),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

retractall(marked(\_,\_)),

%asserting no circle has been marked for movement

assert(marked(\_,0)),

%rolling numbers of dices, F is Dice1 D is Dice2

random\_between(1,6,F),

random\_between(1,6,D),

write('Player Dices: ['), write(F:D), writeln(']'),

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

roll(F,1),

roll(D,2),

dice1(\_,Cube1),

dice2(\_,Cube2),

%incase of double

((Cube1=Cube2, retractall(double(\_,\_)), assert(double(true,true)));

(Cube1\=Cube2)),

colors(Color,\_),

%incase of no movement

findall(Pos, pieces(Pos,\_,Color,\_), L),

(((marked(\_,0), not(isnt\_at\_home1(Color)),no\_move\_u(L,Cube1), no\_move\_u(L,Cube2), nomove, changing\_turn\_to\_computer);

(isnt\_at\_home1(Color),is\_there\_move));true).

moveU.

%checking if all pieces in base of player

isnt\_at\_home1(Color):-

not(pieces(7,\_,Color,\_)),

not(pieces(8,\_,Color,\_)),

not(pieces(9,\_,Color,\_)),

not(pieces(10,\_,Color,\_)),

not(pieces(11,\_,Color,\_)),

not(pieces(12,\_,Color,\_)),

not(pieces(13,\_,Color,\_)),

not(pieces(14,\_,Color,\_)),

not(pieces(15,\_,Color,\_)),

not(pieces(16,\_,Color,\_)),

not(pieces(17,\_,Color,\_)),

not(pieces(18,\_,Color,\_)),

not(pieces(19,\_,Color,\_)),

not(pieces(20,\_,Color,\_)),

not(pieces(21,\_,Color,\_)),

not(pieces(22,\_,Color,\_)),

not(pieces(23,\_,Color,\_)),

not(pieces(24,\_,Color,\_)),

not(pieces(25,\_,Color,\_)).

%checking if all pieces in base of computer

isnt\_at\_home2(EColor):-

not(pieces(0,\_,EColor,\_)),

not(pieces(1,\_,EColor,\_)),

not(pieces(2,\_,EColor,\_)),

not(pieces(3,\_,EColor,\_)),

not(pieces(4,\_,EColor,\_)),

not(pieces(5,\_,EColor,\_)),

not(pieces(6,\_,EColor,\_)),

not(pieces(7,\_,EColor,\_)),

not(pieces(8,\_,EColor,\_)),

not(pieces(9,\_,EColor,\_)),

not(pieces(10,\_,EColor,\_)),

not(pieces(11,\_,EColor,\_)),

not(pieces(12,\_,EColor,\_)),

not(pieces(13,\_,EColor,\_)),

not(pieces(14,\_,EColor,\_)),

not(pieces(15,\_,EColor,\_)),

not(pieces(16,\_,EColor,\_)),

not(pieces(17,\_,EColor,\_)),

not(pieces(18,\_,EColor,\_)).

%no\_move\_u(List of pos,Cube), checks if player can move in his turn

no\_move\_u([],\_).

no\_move\_u([H|L1],Cube):-

H1 is H-Cube,

colors(\_,EColor),

((pieces(H1,X,EColor,\_),

X>1,

H1>0);

(H1=<0)),

no\_move\_u(L1,Cube).

%mark(Top) Top circle in column so you can mark it for movement

mark(Top):-

check4(false), %inorder to block pressing circles while displaying no move msg

turn(player), %checking who's turn that is

pieces(25,\_,\_,Top), %25 is the captive piece of player so must choose that on

marked(\_,0),

retractall(marked(\_,\_)),

assert(marked(Top,1)),

%asserting marked(Top Circle,Amount of marked circles)

send(Top,fill\_pattern,colour(red)),

%checking player's movement with the 2 dices

((not(dice1(\_,\_)),

dice2(Dice2,Cube2),

Pos1 is 25-Cube2,

Pos2 is 25-Cube2);

(not(dice2(\_,\_)),

dice1(Dice1,Cube1),

Pos1 is 25-Cube1,

Pos2 is 25-Cube1);

(dice1(Dice1,Cube1),

dice2(Dice2,Cube2),

Pos1 is 25-Cube1,

Pos2 is 25-Cube2)),

colors(\_,EColor),

%checks if there's column of computer's pieces

(are\_there\_any\_pie(EColor,Pos1,2);

are\_there\_any\_pie(EColor,Pos2,2)),

%reseting check2

retractall(check2(\_)),

assert(check2(false)),

%clicking the dice you'd like for moving the marked circle piece

((dice1(\_,\_), send(Dice1,recogniser,click\_gesture(left,'',single,message(@prolog,getNewPos,1))));true),

((dice2(\_,\_), send(Dice2,recogniser,click\_gesture(left,'',single,message(@prolog,getNewPos,2))));true).

%for clicking circles that are not in top column

mark(Top):-

check4(false),

not(pieces(\_,\_,\_,Top)).

mark(Top):-

check4(false),

not(pieces(25,\_,\_,\_)), %no captive pieces

turn(player),

marked(\_,0),

retractall(marked(\_,\_)),

colors(Color,\_),

pieces(\_,\_,Color,Top), %checking if the selected circle is top circle in column from player

assert(marked(Top,1)), %asserting marked(Top Circle,Amount of marked circles)

send(Top,fill\_pattern,colour(red)),

dice1(Dice1,\_),

dice2(Dice2,\_),

%clicking the dice you'd like for moving the marked circle piece

send(Dice1,recogniser,click\_gesture(left,'',single,message(@prolog,getNewPos,1))),

send(Dice2,recogniser,click\_gesture(left,'',single,message(@prolog,getNewPos,2))).

% click other top piece M instead of your previous Top, changing

% selected cirlce

mark(M):-

check4(false),

turn(player),

not(pieces(25,\_,\_,\_)),

marked(Top,1),

M\=Top,

colors(Color,\_),

send(Top,fill\_pattern,colour(Color)),

pieces(\_,\_,Color,M),

retractall(marked(\_,\_)),

assert(marked(M,1)),

send(M,fill\_pattern,colour(red)),

retractall(check2(\_)),

assert(check2(false)).

%selected circle is captive and can not move

mark(Top):-

check4(false),

turn(player),

check2(true),

pieces(25,\_,\_,Top), %25 is captive line

nomove, %nomove msg

marked(Top,1),

colors(Color,\_), %Color is player's color

send(Top,fill\_pattern,colour(Color)),

%if you got any dice, remove them

((dice1(Dice1,\_), free(Dice1), retractall(dice1(\_,\_)));true),

((dice2(Dice2,\_), free(Dice2), retractall(dice2(\_,\_)));true),

retractall(double(\_,\_)),

assert(double(false,false)),

moveComputer.

%cant click on other circles that are captive

mark(Top):-

check4(false),

turn(player),

pieces(25,\_,\_,\_),

marked(\_,0),

pieces(25,\_,\_,M),

M\=Top.

mark(\_).

%getNewPos(Dice)

%getting new pos for dice 1

getNewPos(1):-

dice1(Dice,Main),

(dice2(\_,Second);true),

marked(Top,1),

colors(Color,\_),

pieces(Pos,A,Color,Top),

Pos1 is Pos-Main,

(

%when not all pieces in base moves normally

((pieces(7,\_,Color,\_);

pieces(8,\_,Color,\_);

pieces(9,\_,Color,\_);

pieces(10,\_,Color,\_);

pieces(11,\_,Color,\_);

pieces(12,\_,Color,\_);

pieces(13,\_,Color,\_);

pieces(14,\_,Color,\_);

pieces(15,\_,Color,\_);

pieces(16,\_,Color,\_);

pieces(17,\_,Color,\_);

pieces(18,\_,Color,\_);

pieces(19,\_,Color,\_);

pieces(20,\_,Color,\_);

pieces(21,\_,Color,\_);

pieces(22,\_,Color,\_);

pieces(23,\_,Color,\_);

pieces(24,\_,Color,\_);

pieces(25,\_,Color,\_)),

((Pos1>=1, Pos1=<24,

possible(Top,A,Pos1,Pos,Main,Second,Dice,1));

(Pos1<1, send(Top,fill\_pattern,colour(Color)),

retractall(marked(\_,\_)),

assert(marked(\_,0)))));

(%when all pieces in base moving them out or closer to 1

not(pieces(7,\_,Color,\_)),

not(pieces(8,\_,Color,\_)),

not(pieces(9,\_,Color,\_)),

not(pieces(10,\_,Color,\_)),

not(pieces(11,\_,Color,\_)),

not(pieces(12,\_,Color,\_)),

not(pieces(13,\_,Color,\_)),

not(pieces(14,\_,Color,\_)),

not(pieces(15,\_,Color,\_)),

not(pieces(16,\_,Color,\_)),

not(pieces(17,\_,Color,\_)),

not(pieces(18,\_,Color,\_)),

not(pieces(19,\_,Color,\_)),

not(pieces(20,\_,Color,\_)),

not(pieces(21,\_,Color,\_)),

not(pieces(22,\_,Color,\_)),

not(pieces(23,\_,Color,\_)),

not(pieces(24,\_,Color,\_)),

not(pieces(25,\_,Color,\_)),

(%when new pos isn't getting them out, move normally

Pos1>0, possible(Top,A,Pos1,Pos,Main,Second,Dice,1));

(%when new pos is 0, that means getting them out from the column is correct

Pos1=0,

pieces(Main,A,\_,Top),

NewA is A-1,

dice1(Dice1,\_),

((double(true,T),

retractall(double(\_,\_)),

assert(double(false,T)));

(double(false,\_),

free(Dice1),

retractall(dice1(\_,\_)))),

isEmpty(NewA,Main,Color),

free(Top),

retractall(piecepic(\_,A,Main,\_)),

retractall(marked(\_,\_)),

assert(marked(\_,0)),

(win;is\_there\_move;moveComputer) );

(%when new pos is below 0, that means getting them out when you don't have higher columns of pieces

Pos1<0,

((pieces(Poss,A,\_,Top),

findall(F, (pieces(F,\_,Color,\_),(F>Poss,F<7)), L1),

length(L1,Length),

Length is 0,

NewA is A-1,

dice1(Dice1,\_),

((double(true,T),

retractall(double(\_,\_)),

assert(double(false,T)));

(double(false,\_),

free(Dice1),

retractall(dice1(\_,\_)))),

isEmpty(NewA,Poss,Color),

free(Top),

retractall(piecepic(\_,A,Poss,\_)),

retractall(marked(\_,\_)),

assert(marked(\_,0)),

(win;is\_there\_move;moveComputer));

(send(Top,fill\_pattern,colour(Color)),

retractall(marked(\_,\_)),

assert(marked(\_,0))))))).

%getting new pos for dice 2

getNewPos(2):-

dice2(Dice,Main),

(dice1(\_,Second);true),

marked(Top,1),

colors(Color,\_),

pieces(Pos,A,Color,Top),

Pos1 is Pos-Main,

(((pieces(7,\_,Color,\_);

pieces(8,\_,Color,\_);

pieces(9,\_,Color,\_);

pieces(10,\_,Color,\_);

pieces(11,\_,Color,\_);

pieces(12,\_,Color,\_);

pieces(13,\_,Color,\_);

pieces(14,\_,Color,\_);

pieces(15,\_,Color,\_);

pieces(16,\_,Color,\_);

pieces(17,\_,Color,\_);

pieces(18,\_,Color,\_);

pieces(19,\_,Color,\_);

pieces(20,\_,Color,\_);

pieces(21,\_,Color,\_);

pieces(22,\_,Color,\_);

pieces(23,\_,Color,\_);

pieces(24,\_,Color,\_);

pieces(25,\_,Color,\_)),

((Pos1>=1, Pos1=<24,

possible(Top,A,Pos1,Pos,Main,Second,Dice,2));

(Pos1<1, send(Top,fill\_pattern,colour(Color)),

retractall(marked(\_,\_)),

assert(marked(\_,0)))));

(not(pieces(7,\_,Color,\_)),

not(pieces(8,\_,Color,\_)),

not(pieces(9,\_,Color,\_)),

not(pieces(10,\_,Color,\_)),

not(pieces(11,\_,Color,\_)),

not(pieces(12,\_,Color,\_)),

not(pieces(13,\_,Color,\_)),

not(pieces(14,\_,Color,\_)),

not(pieces(15,\_,Color,\_)),

not(pieces(16,\_,Color,\_)),

not(pieces(17,\_,Color,\_)),

not(pieces(18,\_,Color,\_)),

not(pieces(19,\_,Color,\_)),

not(pieces(20,\_,Color,\_)),

not(pieces(21,\_,Color,\_)),

not(pieces(22,\_,Color,\_)),

not(pieces(23,\_,Color,\_)),

not(pieces(24,\_,Color,\_)),

not(pieces(25,\_,Color,\_)),

(Pos1>0,

possible(Top,A,Pos1,Pos,Main,Second,Dice,2));

(Pos1=0,

pieces(Main,A,\_,Top),

dice2(Dice2,\_),

((double(T,true),

retractall(double(\_,\_)),

assert(double(T,false)));

(double(\_,false),

free(Dice2),

retractall(dice2(\_,\_)))),

NewA is A-1,

isEmpty(NewA,Main,Color),

free(Top),

retractall(piecepic(\_,A,Main,\_)),

retractall(marked(\_,\_)),

assert(marked(\_,0)),

(win;is\_there\_move;moveComputer));

(Pos1<0,

((pieces(Poss,A,\_,Top),

findall(F, (pieces(F,\_,Color,\_),F>Poss,F<7), L1),

length(L1,Length),

Length is 0,

dice2(Dice2,\_),

((double(T,true),

retractall(double(\_,\_)),

assert(double(T,false)));

(double(\_,false),

free(Dice2),

retractall(dice2(\_,\_)))),

free(Top),

NewA is A-1,

isEmpty(NewA,Poss,Color),

retractall(piecepic(\_,A,Pos1,\_)),

retractall(marked(\_,\_)),

assert(marked(\_,0)),

(win;moveComputer));

(send(Top,fill\_pattern,colour(Color)),

retractall(marked(\_,\_)),

assert(marked(\_,0))))))).

getNewPos(\_).

%win msg for player

win:-

colors(Color,EColor),

not(pieces(\_,\_,Color,\_)),

new(Win,text('Player wins!')),

send(Win,colour(Color)),

send(Win, font, font(times, bold, 36)),

send(@window,display,Win,point(100,330)),

((dice1(Dice1,\_),

free(Dice1), retractall(dice1(\_,\_)));true),

((dice2(Dice2,\_),

free(Dice2), retractall(dice2(\_,\_)));true),

writeln('win'),

((pieces(P,\_,EColor,\_), (P=0;P=1;P=2;P=3;P=4;P=5;P=6), write('turkish '));true),

(((not(isnt\_at\_home2(EColor))), writeln('march'));true),

play\_again(Color).

%lose msg for player

lose:-

colors(Color,EColor),

not(pieces(\_,\_,EColor,\_)),

new(Lose,text('Computer wins!')),

send(Lose,colour(EColor)),

send(Lose, font, font(times, bold, 36)),

send(@window,display,Lose,point(100,330)),

dice1(Dice1,\_),

dice2(Dice2,\_),

free(Dice1), free(Dice2),

retractall(dice1(\_,\_)), retractall(dice2(\_,\_)),

writeln('lose'),

((pieces(P,\_,Color,\_), (P=25;P=24;P=23;P=22;P=21;P=20;P=19), write('turkish '));true),

(((not(isnt\_at\_home1(Color))), writeln('march'));true),

play\_again(EColor).

play\_again(C):-

new(Again,text('Again?')),

send(Again,colour(C)),

send(Again, font, font(times, bold, 26)),

send(@window,display,Again,point(100,400)),

send(Again,recogniser,click\_gesture(left,'',single,message(@prolog,again))),

new(Exit,text('Exit?')),

send(Exit,colour(C)),

send(Exit, font, font(times, bold, 26)),

send(@window,display,Exit,point(220,400)),

send(Exit,recogniser,click\_gesture(left,'',single,message(@prolog,exit))).

again:-

send(@window,destroy),

settings.

exit:-

retractor,

send(@window,destroy).

% updown(Pos,Amount,Pixel)

%checks if your new pos in the upper or lower part of the board

updown(Pos,A,Pixel):-

Pos>=1, Pos=<12, lineY(1,A,Pixel).

updown(Pos,A,Pixel):-

Pos>=13, Pos=<24, lineY(2,A,Pixel).

%captive piece in middle

updown(Pos,A,Pixel):-

(Pos=25;Pos=0), lineY(3,A,Pixel).

%possible(Top Circle,Amount,Pos,Cube,Other\_Cube,Dice Pic,Num)

%possible movement for dice 1

possible(Top,A,Pos1,Pos,Cube,Cube2,Dice,1):-

dice1(Dice,Cube),

colors(Color,EColor),

((Cube=Cube2,

((not(pieces(Pos1,X,EColor,\_)));

(pieces(Pos1,1,EColor,\_))),

double(T1,T2),

((T1=false,

free(Dice),

retractall(dice1(\_,\_)));

(T1=true,retractall(double(\_,\_)),

assert(double(false,T2)))));true),

(%when player moves to empty column

(check1(false),not(pieces(Pos1,\_,EColor,\_)), not(pieces(Pos1,\_,Color,\_)),

assert(pieces(Pos1,1,Color,Top)),

NewA is A-1,

isEmpty(NewA,Pos,Color),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(Top,1,Pos1,Color)),

side(M), lineX(Pos1,X,M), updown(Pos1,1,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube2,

free(Dice),

retractall(dice1(\_,\_)));true) ,

retractall(check1(\_)),

assert(check1(true)),

retractall(marked(\_,\_)),

assert(marked(\_,0))

);

%when player enters his column

(check1(false),not(pieces(Pos1,\_,EColor,\_)), pieces(Pos1,B,Color,\_),

NewB is B+1,

NewA is A-1,

retractall(pieces(Pos1,\_,\_,\_)),

assert(pieces(Pos1,NewB,Color,Top)),

isEmpty(NewA,Pos,Color),

retractall(piecepic(\_,A,Pos,\_)),

((not(piecepic(Top,\_,Pos1,Color)),assert(piecepic(Top,NewB,Pos1,Color)));true),

side(M), lineX(Pos1,X,M), updown(Pos1,NewB,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube2,

free(Dice),

retractall(dice1(\_,\_)));true) ,

retractall(marked(\_,\_)),

assert(marked(\_,0))

);

(%when player eats computer's piece

pieces(Pos1,1,EColor,\_),

piecepic(WPart,1,Pos1,EColor),

retractall(pieces(Pos1,\_,\_,\_)),

assert(pieces(Pos1,1,Color,Top)),

((not(pieces(0,\_,\_,\_)),

assert(pieces(0,1,EColor,WPart)),

retractall(piecepic(WPart,1,Pos1,\_)),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(WPart,1,0,EColor)),

assert(piecepic(Top,1,Pos1,Color)));

(pieces(0,R,\_,\_),

R>=1,

R1 is R+1,

retractall(pieces(0,\_,\_,\_)),

assert(pieces(0,R1,EColor,WPart)),

retractall(piecepic(\_,1,Pos1,\_)),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(WPart,R1,0,EColor)),

assert(piecepic(Top,1,Pos1,Color)))),

NewA is A-1,

isEmpty(NewA,Pos,Color),

pieces(0,C,\_,WPart),

side(M), lineX(0,D,M), updown(0,C,F),

send(WPart,x(D)),

send(WPart,y(F)),

lineX(Pos1,X,M), updown(Pos1,1,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube2, free(Dice), retractall(dice1(\_,\_)));true) ,

retractall(marked(\_,\_)),

assert(marked(\_,0))

);

(pieces(Pos1,X,EColor,\_),

X>1,

retractall(marked(\_,\_)),

assert(marked(\_,0)))

;true

),

retractall(check1(\_)),

assert(check1(false)),

retractall(check2(\_)),

assert(check2(true)),

send(Top,fill\_pattern,colour(Color)),

(is\_there\_move;true).

%possible movement for dice 2

possible(Top,A,Pos1,Pos,Cube,Cube1,Dice,2):-

dice2(Dice,Cube),

colors(Color,EColor),

((Cube=Cube1,

((not(pieces(Pos1,X,EColor,\_)));

(pieces(Pos1,1,EColor,\_))),

double(T1,T2),

(((T2=false,

free(Dice),

retractall(dice2(\_,\_)));

(T2=true,retractall(double(\_,\_)),

assert(double(T1,false))));free(Dice)));true),

(

(not(pieces(Pos1,\_,EColor,\_)), not(pieces(Pos1,\_,Color,\_)),

assert(pieces(Pos1,1,Color,Top)),

NewA is A-1,

isEmpty(NewA,Pos,Color),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(Top,1,Pos1,Color)),

side(M), lineX(Pos1,X,M), updown(Pos1,1,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube1,

free(Dice),

retractall(dice2(\_,\_)));Cube=Cube1) ,

retractall(check1(\_)),

assert(check1(true)),

retractall(marked(\_,\_)),

assert(marked(0,\_))

);

(check1(false),not(pieces(Pos1,\_,EColor,\_)), pieces(Pos1,B,Color,\_),

NewB is B+1,

NewA is A-1,

retractall(pieces(Pos1,\_,\_,\_)),

assert(pieces(Pos1,NewB,Color,Top)),

isEmpty(NewA,Pos,Color),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(Top,NewB,Pos1,Color)),

side(M), lineX(Pos1,X,M), updown(Pos1,NewB,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube1,

free(Dice),

retractall(dice2(\_,\_)));true) ,

retractall(marked(\_,\_)),

assert(marked(0,\_))

);

( pieces(Pos1,1,EColor,\_),

piecepic(WPart,1,Pos1,EColor),

retractall(pieces(Pos1,\_,\_,\_)),

assert(pieces(Pos1,1,Color,Top)),

((not(pieces(0,\_,\_,\_)),

assert(pieces(0,1,EColor,WPart)),

retractall(piecepic(WPart,1,Pos1,\_)),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(WPart,1,0,EColor)),

assert(piecepic(Top,1,Pos1,Color)));

(pieces(0,R,\_,\_),

R>=1,

R1 is R+1,

retractall(pieces(0,\_,\_,\_)),

assert(pieces(0,R1,EColor,WPart)),

retractall(piecepic(\_,1,Pos1,\_)),

retractall(piecepic(\_,A,Pos,\_)),

assert(piecepic(WPart,R1,0,EColor)),

assert(piecepic(Top,1,Pos1,Color)))),

NewA is A-1,

isEmpty(NewA,Pos,Color),

pieces(0,C,\_,WPart),

side(M),lineX(0,D,M), updown(0,C,F),

send(WPart,x(D)),

send(WPart,y(F)),

lineX(Pos1,X,M), updown(Pos1,1,Y),

send(Top,x(X)),

send(Top,y(Y)),

write('Pl moving: '), writeln(Pos+Cube=Pos1),

((Cube\=Cube1,

free(Dice),

retractall(dice2(\_,\_)));true) ,

retractall(marked(\_,\_)),

assert(marked(0,\_))

);

(pieces(Pos1,X,EColor,\_),

X>1,

retractall(marked(\_,\_)),

assert(marked(0,\_)))

;true

),

retractall(check1(\_)),

assert(check1(false)),

retractall(check2(\_)),

assert(check2(true)),

send(Top,fill\_pattern,colour(Color)),

(is\_there\_move;true).

%checks if there's a move for player in 2 dices

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

dice1(Dice11,Cube11),

dice2(Dice22,Cube22),

no\_move\_u(NL,Cube11),

no\_move\_u(NL,Cube22),

nomove,

free(Dice11),

free(Dice22),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

moveComputer.

%checks if there's a move for player in dice 2 in base

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

not(dice1(\_,\_)),

dice2(Dice22,Cube22),

no\_move\_u(NL,Cube22),

nomove,

free(Dice22),

retractall(dice2(\_,\_)),

moveComputer.

%checks if there's a move for player in dice 1 in base

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

dice1(Dice11,Cube11),

not(dice2(\_,\_)),

no\_move\_u(NL,Cube11),

nomove,

free(Dice11),

retractall(dice1(\_,\_)),

moveComputer.

%checks if there's a move for player in dice 1 in base

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

dice1(Dice11,Cube11),

not(dice2(\_,\_)),

no\_move\_us(NL,Cube11),

nomove,

free(Dice11),

retractall(dice1(\_,\_)),

moveComputer.

%checks if there's a move for player in dice 2 in base

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

not(dice1(\_,\_)),

dice2(Dice22,Cube22),

no\_move\_us(NL,Cube22),

nomove,

free(Dice22),

retractall(dice2(\_,\_)),

moveComputer.

%checks if there's a move for player in 2 dices in base

is\_there\_move:-

colors(Color,\_),

not(isnt\_at\_home1(Color)),

findall(Pos, pieces(Pos,\_,Color,\_), NL),

dice1(Dice11,Cube11),

dice2(Dice22,Cube22),

no\_move\_us(NL,Cube11),

no\_move\_us(NL,Cube22),

nomove,

free(Dice11),

free(Dice22),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

moveComputer.

%move to computer's turn

is\_there\_move:-

moveComputer.

%no\_move\_us(List of Pos,Cube) checks if computer can move

no\_move\_us([],\_).

no\_move\_us([H|L],Cube):-

Pos1 is H-Cube,

colors(Color,EColor),

((Pos1>0,

pieces(Pos1,X,EColor,\_),

X>1);

(Pos1<0,

findall(F, (pieces(F,\_,Color,\_),F>H,H<7), L1),

length(L1,Length),

Length\=0)),

no\_move\_us(L,Cube).

%computer's turn

moveComputer:-

not(dice1(\_,\_)), not(dice2(\_,\_)),

retractall(turn(\_)),

assert(turn(computer)),

double(false,false),

random\_between(1,6,F),

random\_between(1,6,D),

write('Computer Dices: ['), write(F:D), writeln(']'),

retractall(grade(\_,\_,\_)),

assert(grade(-10000,-10000,-10000)),

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

keep\_rolling,

roll(F,1),

roll(D,2),

send(timer(1), delay),

((F is D, retractall(double(\_,\_)), assert(double(true,true))); F\=D),

dragon,

(check4(true), nomove, retractall(check4(\_)), assert(check4(false));true),

retractall(double(\_,\_)),

assert(double(false,false)),

retractall(check3(\_)),

assert(check3(false)),

retractall(check(\_)),

assert(check(false)),

dice1(Dice1,\_),

dice2(Dice2,\_),

send(Dice1,recogniser,click\_gesture(left,'',single,message(@prolog,moveU))),

send(Dice2,recogniser,click\_gesture(left,'',single,message(@prolog,moveU))).

%when isn't computer's turn

moveComputer.

%dragon, not double dices, 2 movements

dragon:-

double(false,false),

colors(\_,EColor),

((pieces(0,\_,\_,\_), B1=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos1, pieces(Pos1,\_,EColor,\_), B1))),

%hiroshima(List of Pos,Dice) gets best grade for all columns with dice 1

hiroshima(B1,1),

((pieces(0,\_,\_,\_), B2=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos2, pieces(Pos2,\_,EColor,\_), B2))),

%hiroshima(List of Pos,Dice) gets best grade for all columns with dice 2

hiroshima(B2,2),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_),

grade(\_,NPos1,R),

pieces(NPos1,\_,\_,H),

%option(Top Circle, Dice) does the best movement with the selected dice

options(H,R),

retractall(grade(\_,\_,\_)),

assert(grade(-10000,-10000,-10000)),

%getting the other dice

R1 is 3-R,

(((( pieces(0,\_,\_,\_),

B3=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos3, pieces(Pos3,\_,EColor,\_), B3))),

%hiroshima(List of Pos,Dice) gets best grade of 2nd dice

hiroshima(B3,R1),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_), %if there are pieces and there was better grade than initial grade

grade(\_,NPos2,R1),

pieces(NPos2,\_,\_,NH),

send(timer(1), delay),

%option(Top Circle, Dice) does the movement with the 2nd dice

options(NH,R1));nomove).

%dragon, double dices, 4 movements

dragon:-

double(true,true),

colors(\_,EColor),

((pieces(0,\_,\_,\_),

B1=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos1, pieces(Pos1,\_,EColor,\_), B1))),

hiroshima(B1,1),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_),

grade(\_,NPos1,1),

pieces(NPos1,\_,\_,H1),

options(H1,1),

retractall(grade(\_,\_,\_)),

assert(grade(-10000,-10000,-10000)),

(((

pieces(0,\_,\_,\_),

B2=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos2, pieces(Pos2,\_,EColor,\_), B2))),

hiroshima(B2,1),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_),

grade(\_,NPos2,1),

pieces(NPos2,\_,\_,H2),

send(timer(1), delay),

options(H2,1);true),

retractall(grade(\_,\_,\_)),

assert(grade(-10000,-10000,-10000)),

((((

pieces(0,\_,\_,\_),

B3=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos3, pieces(Pos3,\_,EColor,\_), B3))),

hiroshima(B3,1),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_),

grade(\_,NPos3,1),

pieces(NPos3,\_,\_,H3),

send(timer(1), delay),

options(H3,1));true),

retractall(grade(\_,\_,\_)),

assert(grade(-10000,-10000,-10000)),

((((

pieces(0,\_,\_,\_),

B4=[0]);

(not(pieces(0,\_,\_,\_)),findall(Pos4, pieces(Pos4,\_,EColor,\_), B4))),

hiroshima(B4,1),

not(grade(-10000,-10000,-10000)), pieces(\_,\_,EColor,\_),

grade(\_,NPos4,1),

pieces(NPos4,\_,\_,H4),

send(timer(1), delay),

options(H4,1));nomove).

dragon:-

nomove.

hiroshima([],\_).

hiroshima([H|L],R):-

retractall(check3(\_)),

assert(check3(false)),

kraken(H,R),

hiroshima(L,R).

%when you're at base and you want pieces out

kraken(H,R):-

check3(false),

colors(Color,EColor),

isnt\_at\_home2(EColor),

((R is 1,dice1(\_,Cube));(R is 2, dice2(\_,Cube))),

H1 is H+Cube,

((H1<25, are\_there\_any\_pie(Color,H1,2) ,G is 300);

(H1 is 25,G is 300);

(H1>25,

findall(F, (pieces(F,\_,EColor,\_),F<H,F>18), L1),

length(L1,Length),

Length is 0,

G is 600);false),

((grade(BestG,\_,\_),

G>BestG,

retractall(grade(\_,\_,\_)),

assert(grade(G,H,R)));true),

retractall(check3(\_)),

assert(check3(true)).

%when there's empty column

kraken(H,R):-

check3(false),

colors(\_,EColor),

not(isnt\_at\_home2(EColor)),

((R is 1,dice1(\_,Cube));(R is 2, dice2(\_,Cube))),

H1 is H+Cube,

H1=<24,

not(pieces(H1,\_,\_,\_)),

findall((Pos,Amount1,Color), (pieces(Pos,Amount,Color,\_),((Pos is H,Amount\=1,Amount1 is Amount-1);(Pos\=H,Amount1 is Amount))), Board),

append(Board,[(H1,1,EColor)],NewBoard),

nikud1(NewBoard,G1),

nikud2(NewBoard,G2),

nikud3(NewBoard,G3),

nikud4(NewBoard,G4),

((

(dice1(\_,\_),dice2(\_,\_)), R1 is 3-R, ((R1 is 1,dice1(\_,Cube2));(R1 is 2, dice2(\_,Cube2))),

findall(Pos3, pieces(Pos3,\_,EColor,\_), LL),

DPos is H1-Cube2,

member(DPos, LL),

G5 is 600);G5 is 0),

((H>=0, H=<6, G6 is 400);(H>=7, H=<12, G6 is 300);(H>=13, H=<18, G6 is 200);(H>=19, G6 is 100)),

G is G1+G2+G3+G4+G5+G6,

((grade(BestG,\_,\_),

G>BestG,

retractall(grade(\_,\_,\_)),

assert(grade(G,H,R)));true),

retractall(check3(\_)),

assert(check3(true)).

%when there's 1 piece to eat to player

kraken(H,R):-

check3(false),

colors(\_,EColor),

not(isnt\_at\_home2(EColor)),

((R is 1,dice1(\_,Cube));(R is 2, dice2(\_,Cube))),

H1 is H+Cube,

H1=<24,

pieces(H1,1,Color,\_),

findall((Pos,Amount1,Color),(pieces(Pos,Amount,Color,\_),((Pos is H,Amount\=1,Amount1 is Amount-1);(Pos\=H,Pos\=H1,Pos\=25,Amount1 is Amount))),Board),

append(Board,[(H1,1,EColor)],NewBoard),

((pieces(25,X,\_,\_),

X1 is X+1,

append(NewBoard,[(25,X1,Color)],NNewBoard));

(not(pieces(25,\_,\_,\_)),

append(NewBoard,[(25,1,Color)],NNewBoard))),

nikud1(NNewBoard,G1),

nikud2(NNewBoard,G2),

nikud3(NNewBoard,G3),

nikud4(NNewBoard,G4),

((

(dice1(\_,\_),dice2(\_,\_)), R1 is 3-R, ((R1 is 1,dice1(\_,Cube2));(R1 is 2, dice2(\_,Cube2))),

findall(Pos3, pieces(Pos3,\_,EColor,\_), LL),

DPos is H1-Cube2,

member(DPos, LL),

G5 is 600);G5 is 0),

( (H>=0, H=<6, G6 is 400);(H>=7, H=<12, G6 is 300);(H>=13, H=<18, G6 is 200);(H>=19, G6 is 100)),

G is G1+G2+G3+G4+G5+G6,

((grade(BestG,\_,\_),

G>BestG,

retractall(grade(\_,\_,\_)),

assert(grade(G,H,R)));true),

retractall(check3(\_)),

assert(check3(true)).

% when computer adds 1 piece to column

kraken(H,R):-

check3(false),

colors(\_,EColor),

not(isnt\_at\_home2(EColor)),

((R is 1,dice1(\_,Cube));(R is 2, dice2(\_,Cube))),

H1 is H+Cube,

H1=<24,

pieces(H1,\_,EColor,\_),

findall((Pos,Amount1,Color),(pieces(Pos,Amount,Color,\_),((Pos is H,Amount1 is Amount-1);(Pos is H1,Amount1 is Amount+1);(Pos\=H,Pos\=H1,Amount1 is Amount))),Board),

nikud1(Board,G1),

nikud2(Board,G2),

nikud3(Board,G3),

nikud4(Board,G4),

(((pieces(H1,1,\_,\_)), G5 is 500);G5 is 0),

((H>=0, H=<6, G6 is 400);(H>=7, H=<12, G6 is 300);(H>=13, H=<18, G6 is 200);(H>=19, G6 is 100)),

G is G1+G2+G3+G4+G5+G6,

((grade(BestG,\_,\_),

G>BestG,

retractall(grade(\_,\_,\_)),

assert(grade(G,H,R)));true),

retractall(check3(\_)),

assert(check3(true)).

% when there's column of player's pieces and can not move there and make

% grade

kraken(\_,\_).

%changing\_turn\_computer

changing\_turn\_to\_computer:-

dice1(Dice1,\_),

dice2(Dice2,\_),

free(Dice1),

free(Dice2),

retractall(dice1(\_,\_)),

retractall(dice2(\_,\_)),

moveComputer.

%nikudN(List of Pos,Grade)

%difference between black and white of number of open pieces

nikud1(Board,G1):-

colors(Color,EColor),

findall(Pos1, member((Pos1,1,EColor), Board), L1),

length(L1,Length1),

findall(Pos2, member((Pos2,1,Color), Board), L2),

length(L2,Length2),

Diff is Length2-Length1,

G1 is Diff\*300.

%difference between black and white of captive pieces

nikud2(Board,G2):-

colors(Color,EColor),

findall(X1, member((0,X1,EColor), Board), L1),

length(L1,Length1),

findall(X2, member((25,X2,Color), Board), L2),

length(L2,Length2),

Diff is Length2-Length1,

G2 is (Diff\*1000).

% difference between black and white between distance of columns from

% eachother

nikud3(Board,G3):-

colors(Color,EColor),

findall(Pos1, member((Pos1,\_,EColor), Board), L1),

sort(L1,NL1),

reverse(NL1,NNL1),

rec(NNL1,Sum1),

findall(Pos2, member((Pos2,\_,Color), Board), L2),

sort(L2,NL2),

reverse(NL2,NNL2),

rec(NNL2,Sum2),

Diff is Sum2-Sum1,

G3 is (Diff\*50).

% difference between black and white of in-base pieces

nikud4(Board,G4):-

colors(Color,EColor),

findall(Pos, (member((Pos,\_,EColor), Board),Pos>18, Pos<25), L1),

length(L1,Length1),

findall(Pos, (member((Pos,\_,Color), Board),Pos>0, Pos<7), L2),

length(L2,Length2),

Diff is Length2-Length1,

G4 is (Diff\*100).

%checking distance of columns in nikud3 for making a wall

rec([\_|[]],0).

rec([H|T],Sum1):-

nth1(1,T,Part),

Diff is H-Part,

rec(T,Sum),

Sum1 is Diff+Sum.

%options(Top Circle,Dice1) when pieces aren't in base

options(H,1):-

colors(\_,EColor),

not(isnt\_at\_home2(EColor)),

dice1(\_,Cube),

pieces(Pos,\_,EColor,H),

Pos1 is Pos+Cube,

Pos1>=1, Pos1=<24,

check(false),

states(H,Pos1,1),

showit(Pos1),

write('AI moving: '), writeln(Pos+Cube=Pos1).

%options(Top Circle,Dice1) when pieces aren in base

options(H,1):-

colors(\_,EColor),

isnt\_at\_home2(EColor),

dice1(\_,Cube),

pieces(Pos,\_,EColor,H),

Pos1 is Pos+Cube,

states(H,Pos1,2),

write('AI moving out: '), writeln(Pos+Cube=Pos1).

%options(Top Circle,Dice2) when pieces aren't in base

options(H,2):-

colors(\_,EColor),

not(isnt\_at\_home2(EColor)),

dice2(\_,Cube),

pieces(Pos,\_,EColor,H),

Pos1 is Pos+Cube,

Pos1>=1, Pos1=<24,

check(false),

states(H,Pos1,1),

showit(Pos1),

write('AI moving: '), writeln(Pos+Cube=Pos1).

%options(Top Circle,Dice2) when pieces are in base

options(H,2):-

colors(\_,EColor),

isnt\_at\_home2(EColor),

dice2(\_,Cube),

pieces(Pos,\_,EColor,H),

Pos1 is Pos+Cube,

states(H,Pos1,2),

write('AI moving out: '), writeln(Pos+Cube=Pos1).

% states(Top Circle,New Pos,Phase)

% Phase=1 of getting circles to base, 2 getting them out of base

% state when computer goes into empty column

states(H,Pos1,1):-

check(false),

colors(Color,EColor),

pieces(Pos,A,EColor,H),

not(pieces(Pos1,\_,EColor,\_)), not(pieces(Pos1,\_,Color,\_)),

NewA is A-1,

isEmpty(NewA,Pos,EColor),

retractall(piecepic(H,\_,\_,EColor)),

assert(piecepic(H,1,Pos1,EColor)),

assert(pieces(Pos1,1,EColor,H)),

retractall(check(\_)),

assert(check(true)).

%state when computer goes to column of his pieces

states(H,Pos1,1):-

check(false),

colors(Color,EColor),

pieces(Pos,A,EColor,H),

not(pieces(Pos1,\_,Color,\_)), pieces(Pos1,B,EColor,\_),

NewB is B+1,

NewA is A-1,

retractall(pieces(Pos1,\_,EColor,\_)),

assert(pieces(Pos1,NewB,EColor,H)),

isEmpty(NewA,Pos,EColor),

retractall(piecepic(H,\_,\_,EColor)),

assert(piecepic(H,NewB,Pos1,EColor)),

retractall(check(\_)),

assert(check(true)).

%state when computer can eat player's piece

states(H,Pos1,1):-

check(false),

colors(Color,EColor),

pieces(Pos,A,EColor,H),

pieces(Pos1,1,Color,BPart),

piecepic(BPart,1,Pos1,Color),

retractall(pieces(Pos1,\_,\_,\_)),

((not(pieces(25,\_,\_,\_)),

assert(pieces(25,1,Color,BPart)),

retractall(piecepic(BPart,\_,\_,\_)),

assert(piecepic(BPart,1,25,Color)));

(pieces(25,R,\_,\_),

R1 is R+1,

retractall(pieces(25,\_,\_,\_)),

assert(pieces(25,R1,Color,BPart)),

retractall(piecepic(BPart,\_,\_,\_)),

assert(piecepic(BPart,R1,25,Color)))),

NewA is A-1,

isEmpty(NewA,Pos,EColor),

pieces(25,C,\_,\_),

side(M), lineX(25,D,M), updown(25,C,F),

send(BPart,x(D)),

send(BPart,y(F)),

assert(pieces(Pos1,1,EColor,H)),

retractall(piecepic(H,\_,\_,EColor)),

assert(piecepic(H,1,Pos1,EColor)),

retractall(check(\_)),

assert(check(true)).

%state of computer getting pieces out of base

states(H,Pos1,2):-

colors(\_,EColor),

((Pos1<25,

((states(H,Pos1,1),showit(Pos1));

(send(H,fill\_pattern,colour(EColor)),

retractall(marked(\_,\_)),

assert(marked(\_,0))),false));

(Pos1 is 25,

pieces(Pos,A,\_,H),

NewA is A-1,

isEmpty(NewA,Pos,EColor),

free(H),

retractall(piecepic(\_,A,Pos,\_)),

not(lose));

(Pos1>25,

((pieces(Poss,A,\_,H),

findall(F, (pieces(F,\_,EColor,\_),F<Poss,F>18), L1),

length(L1,Length),

Length is 0,

free(H),

NewA is A-1,

isEmpty(NewA,Poss,EColor),

retractall(piecepic(\_,A,Pos1,\_)))),

not(lose))).

%incase of a loss

states(\_,\_,2):-

lose.

%showit(New Pos), Shows the new pos of the computer's piece

showit(Pos1):-

check(true),

colors(\_,EColor),

pieces(Pos1,A,EColor,H),

retractall(check(\_)),

assert(check(false)),

side(M), lineX(Pos1,X,M), updown(Pos1,A,Y),

send(H,x(X)),

send(H,y(Y)).

% isEmpty(NewAmount,Pos,Color), checks if previous Pos has any pieces

isEmpty(0,Pos,\_):-

retractall(pieces(Pos,\_,\_,\_)).

isEmpty(NewA,Pos,Color):-

NewA\=0,

piecepic(NPart,NewA,Pos,Color),

retractall(pieces(Pos,\_,\_,\_)),

assert(pieces(Pos,NewA,Color,NPart)).

% are\_there\_any\_pie(Color,Pos,Amount), checks if there's column of

% pieces and cant move there

are\_there\_any\_pie(\_,\_,16).

are\_there\_any\_pie(Color,Pos,T):-

not(pieces(Pos,T,Color,\_)),

T1 is T+1,

are\_there\_any\_pie(Color,Pos,T1).

%nomove msg

nomove:-

colors(\_,EColor),

pieces(\_,\_,EColor,\_),

retractall(check4(\_)),

assert(check4(true)),

new(NoMove,text('No Move!')),

send(NoMove, font, font(times, bold, 40)),

send(NoMove, colour(white)),

send(@window,display,NoMove,point(130,230)),

send(timer(0.5),delay),

retractall(check4(\_)),

assert(check4(false)),

free(NoMove).

%settings dialog msg

settings:-

new(Dialog,dialog('Settings')),

send\_list(Dialog,append,

[

new(S, menu(side)),

new(C, menu(colors,cycle)),

button(ok,and(message(@prolog,

game,

S?selection,

C?selection),

message(Dialog,destroy))),

button(instructions,and(message(@prolog,

instructions),

message(Dialog,destroy)))

]),

send\_list(S,append,[right,left,random]),

send\_list(C,append,[black\_vs\_white,

white\_vs\_black,

black\_vs\_red,

red\_vs\_black,

black\_vs\_blue,

blue\_vs\_black,

black\_vs\_green,

green\_vs\_black,

black\_vs\_orange,

orange\_vs\_black,

white\_vs\_red,

red\_vs\_white,

white\_vs\_blue,

blue\_vs\_white,

white\_vs\_green,

green\_vs\_white,

orange\_vs\_red,

red\_vs\_orange

]),

send(Dialog,open).

%side\_decide(Side), Side=right,left,random

side\_decide(Side):-

Side=right, retractall(side(\_)), assert(side(right)).

side\_decide(Side):-

Side=left, retractall(side(\_)), assert(side(left)).

side\_decide(Side):-

Side=random, random\_between(1,2,M), random\_side(M).

%random\_side(Num), Num=1,2 for right or left

random\_side(1):-

retractall(side(\_)), assert(side(right)).

random\_side(2):-

retractall(side(\_)), assert(side(left)).

%coloring(Text), Text tells the two colors of player and computer

coloring(black\_vs\_white):-

retractall(colors(\_,\_)), assert(colors(black,white)).

coloring(white\_vs\_black):-

retractall(colors(\_,\_)), assert(colors(white,black)).

coloring(black\_vs\_red):-

retractall(colors(\_,\_)), assert(colors(black,brown)).

coloring(red\_vs\_black):-

retractall(colors(\_,\_)), assert(colors(brown,black)).

coloring(black\_vs\_blue):-

retractall(colors(\_,\_)), assert(colors(black,blue)).

coloring(blue\_vs\_black):-

retractall(colors(\_,\_)), assert(colors(blue,black)).

coloring(black\_vs\_green):-

retractall(colors(\_,\_)), assert(colors(black,green)).

coloring(green\_vs\_black):-

retractall(colors(\_,\_)), assert(colors(green,black)).

coloring(black\_vs\_orange):-

retractall(colors(\_,\_)), assert(colors(black,orange)).

coloring(orange\_vs\_black):-

retractall(colors(\_,\_)), assert(colors(orange,black)).

coloring(white\_vs\_red):-

retractall(colors(\_,\_)), assert(colors(white,brown)).

coloring(red\_vs\_white):-

retractall(colors(\_,\_)), assert(colors(brown,white)).

coloring(white\_vs\_blue):-

retractall(colors(\_,\_)), assert(colors(white,blue)).

coloring(blue\_vs\_white):-

retractall(colors(\_,\_)), assert(colors(blue,white)).

coloring(white\_vs\_green):-

retractall(colors(\_,\_)), assert(colors(white,green)).

coloring(green\_vs\_white):-

retractall(colors(\_,\_)), assert(colors(green,white)).

coloring(orange\_vs\_red):-

retractall(colors(\_,\_)), assert(colors(orange,brown)).

coloring(red\_vs\_orange):-

retractall(colors(\_,\_)), assert(colors(brown,orange)).

%keep\_rolling(Times), the rolling simulation of dices

keep\_rolling:-

random\_between(1,6,E),

random\_between(1,6,F),

bmp(E, BMP1),

bmp(F, BMP2),

cube(1, X1, Y1),

cube(2, X2, Y2),

new(Dice1,bitmap(BMP1)),

send(@window,display,Dice1,point(X1,Y1)),

new(Dice2,bitmap(BMP2)),

send(@window,display,Dice2,point(X2,Y2)),

send(timer(0.05), delay),

free(Dice1), free(Dice2).

%instructions for backgammon

instructions:-

new(@ins,window('Instructions',size(300,300))),

new(Title,text('Instructions:')),

send(Title, font, font(ariel, bold, 20)),

send(@ins,display,Title,point(20,10)),

new(T1,text('Each player has 15 pieces.')),

new(T2,text('Your goal is to get your pieces out')),

new(T3,text('from your base, In your turn, you roll')),

new(T4,text('the dices and move your pieces')),

new(T5,text('according to the numbers on the dices.')),

new(T6,text('The first to get his pieces out - wins!')),

send(T1, font, font(ariel, bold, 14)),

send(T2, font, font(ariel, bold, 14)),

send(T3, font, font(ariel, bold, 14)),

send(T4, font, font(ariel, bold, 14)),

send(T5, font, font(ariel, bold, 14)),

send(T6, font, font(ariel, bold, 14)),

send(@ins,display,T1,point(0,50)),

send(@ins,display,T2,point(0,75)),

send(@ins,display,T3,point(0,100)),

send(@ins,display,T4,point(0,125)),

send(@ins,display,T5,point(0,150)),

send(@ins,display,T6,point(0,175)),

new(Back,text('back to settings')),

send(Back, font, font(ariel, bold, 20)),

send(Back, colour(red)),

send(@ins,display,Back,point(30,230)),

send(Back,recogniser,click\_gesture(left,'',single,message(@prolog,transition))),

send(@ins,open).

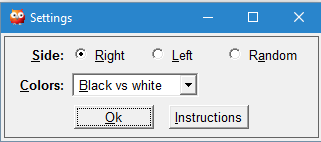
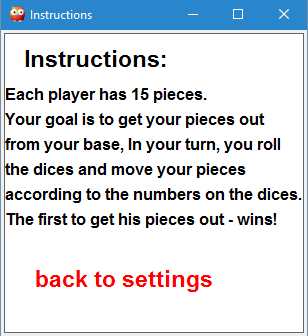
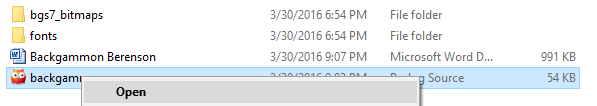
%transition from instuctions to settings

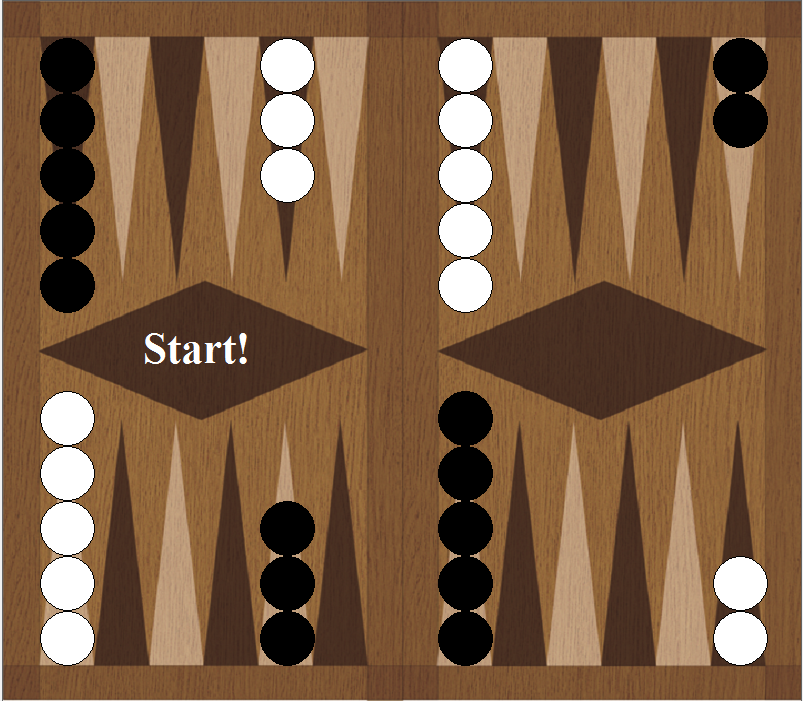
transition:-

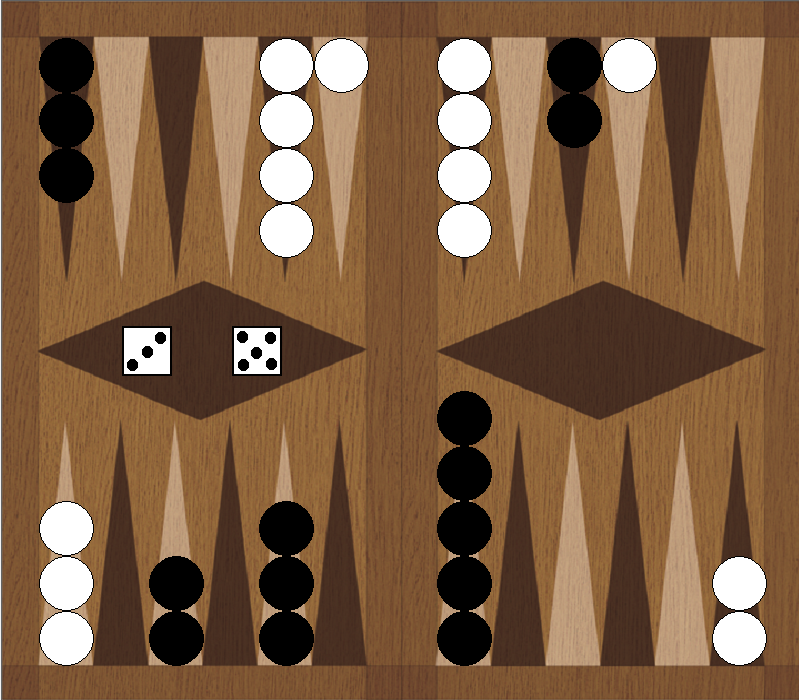
free(@ins),

settings.

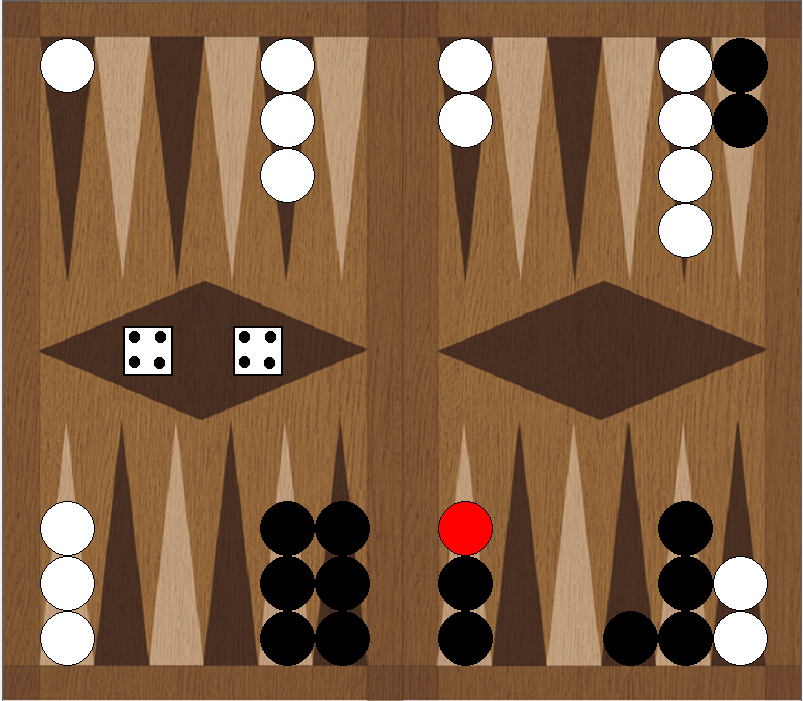
שימוש במערכת והרצתה

C:\Users\shahar\Desktop\Screenshot_13.pngראשית יש לפתוח את הקובץ backgammon.pl כאשר הקובץ עצמו נמצא בתיקייה שבתוכה יש תיקייה בשם bgs7\_bitmaps המכילה את הקבצים הגרפיים.  
  
  
  
  
  
  
כעת יש להקליד settings. על מנת לפתוח את חלון ההגדרות.   
  
  
  
  
  
  
  
  
בחלון ההגדרות מופיע בחירת צד הבסיס, רשימה של צבעים בשביל השחקן נגד המחשב והוראות המשחק.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
לאחר לחיצה על הלחצן Ok, נפתח חלון המשחק עם הלוח, 15 כלים לכל שחקן והטקסט Start! כדי להתחיל.

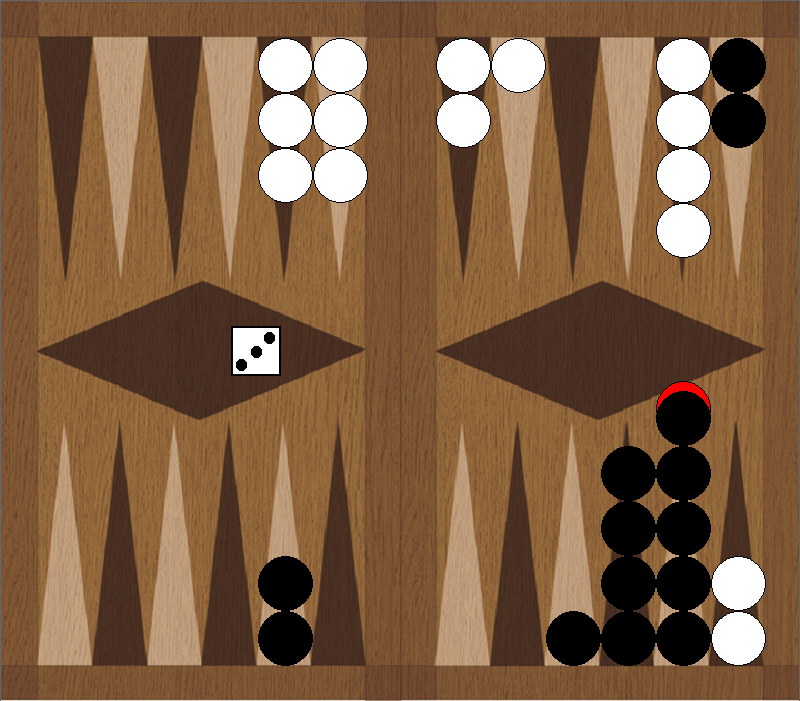
  
  
  
  
  
  
  
  
  
  
  
  
  
  
לאחר לחיצה על Start! יופיעו שתי קוביות שיחליטו מי מתחיל, השחקן או המחשב.   
לאחר מכן, אפשר להתחיל לשחק! לחיצה על הקוביות לאחר תור המחשב או הטלת הקוביות למי מתחיל תוביל לקבלת קוביות חדשות לתור של השחקן שבהן תצטרך ללחוץ על כלי עליון בעמודה ואז על הקובייה שתרצה להזיז את הכלי איתה.  
השחקן תמיד יהיה עם הבסיס התחתון והמחשב תמיד יהיה עם הבסיס העליון.



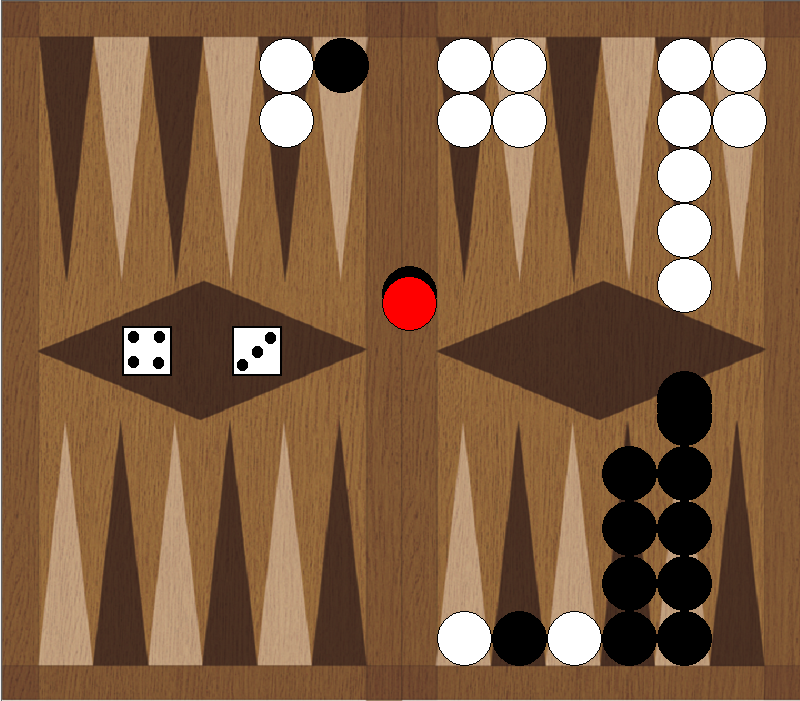
דוגמאות לסימון הכלי העליון שלך בעמודה



סימון רגיל לעמודה של פחות מ-   
 6 כלים

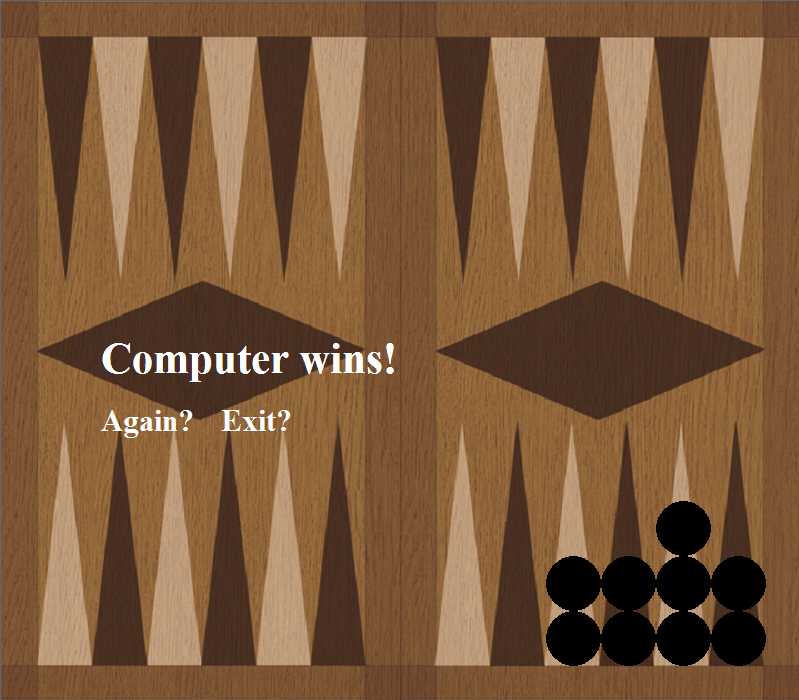


סימון כשבעמודה יש יותר מ-5 כלים



סימון של כלי אכול- חייב   
 לסמן אותו כדי להוציאו ולא  
 יכול ללחוץ על כלים אחרים

דוגמא לניצחון של מחשב כשהבסיסים בצד ימין והצבעים הם שחור נגד לבן;   
הגדרות המצב של ברירת מחדל. תוכל ללחוץ על Again? כדי לשחק עוד פעם מה שמחזיר אותך לחלון ההגדרות או לצאת מן המשחק ע"י לחיצה על Exit?.



הצעות לשיפור

ישנן אפשרויות רבות לשיפור המשחק.

* שחקן מול שחקן
* שיפור הגרפיקה
* ייעול התוכנית והחוקים למינימום שורות
* שיפור הניקוד
* אפשרות לשחק שש בש מקסיקני.

נספחים

**ביבליוגרפיה**

* האתר הרשמי של השפה פרולוג

[**http://www.swi-prolog.org/**](http://www.swi-prolog.org/)