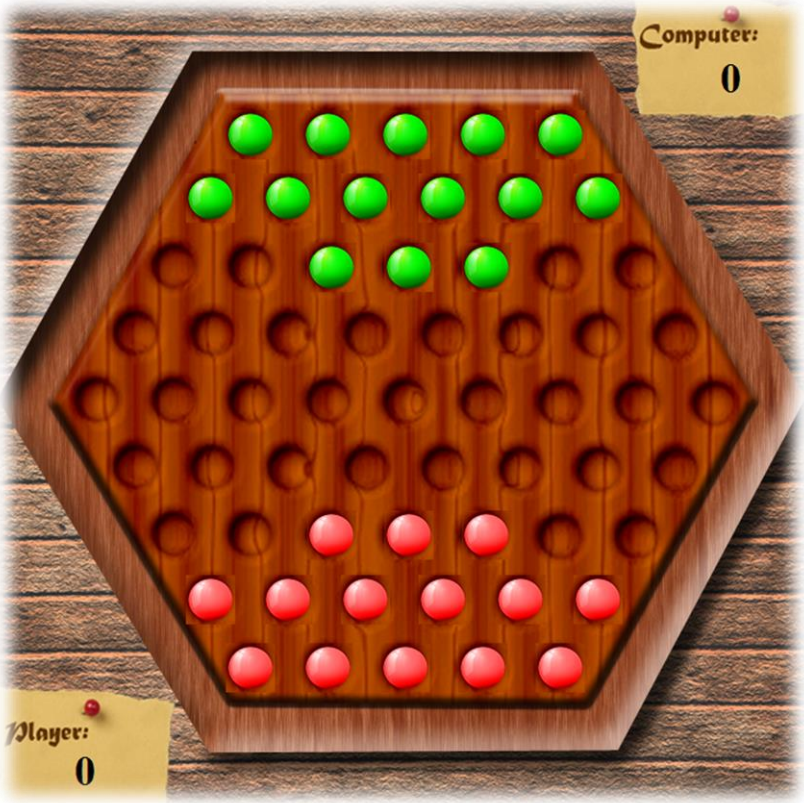
פרויקט ב-5 יח"ל בינה מלאכותית

אבאלון

Abalone



מאת: ערן פלשין
ת"ז: 314626334
כיתה: י"ב 3
מורה מנחה: אסתי מאסטרסי
תאריך: אפריל 2016

בית הספר העברי בחיפה, בית בירם.

תוכן עניינים

הקדמה

המשחק אבאלון הוא משחק קופסא אסטרטגי המיועד לשני אנשים שהומצא ב-1987. שמו של המשחק מרמז על כלל האצבע הנחוץ לניצחון ab משמעו-לא ו- alone משמעו-לבד, כלומר יש לשמור על כלי המשחק ביחד. השחקנים מיוצגים באמצעות גולות משני צבעים שונים. המשחק נערך בזירה שצורתה מתומן. אבאלון עלה למדפים ב-1990 ונמכר מאז ביותר מ-30 מדינות.
כל שנה נערכות תחרויות בינלאומיות במשחק אבאלון בהן מתחרים אנשים מכל העולם. במקביל מתקיימות תחרויות בין אלגוריתמים שונים שמפותחים ע"י קבוצות סטודנטים מאוניברסיטאות שונות בעולם. האלגוריתמים השונים מתבססים על שתי שיטות עיקריות:

אלגוריתם MIN-MAX- המחשב בהגיע תורו, מנסה למקסם מצד אחד את פונקציית ההערכה (זוהי פונקציה שנותנת ציון מספרי למערך הכלים בלוח במצב הנתון) של עצמו ומצד שני מנסה לצמצם את השיפור בערך פונקציית ההערכה של היריב. באלגוריתם זה, נבנה עץ המתאר את המהלכים האפשריים ומערכי הלוח לאחר ביצוע המהלכים השונים המיוצגים ע"י ציון מספרי הניתן ע"י פונקציית ההערכה. האלגוריתם מתמרן את מהלכיו לפי העץ כך שבסופו של דבר ידו של המחשב תהיה על העליונה ואילו היריב יאלץ לברור במהלך המשחק בין מהלכים גרועים.

אלגוריתם היוריסטי- מתבסס על צורת המחשבה של האדם כאשר הוא משחק במשחק קופסא כלשהו. למשחק קיימים מספר כללי אצבע המבטיחים ניצחון (או לפחות שאיפה אליו).
האלגוריתם פועל לפי כללי האצבע הללו ובדרך זו מקבל החלטות בדבר מהלכיו במשחק.
באלגוריתם זה המחשב בודק איזה מהלך לבחור לפי סדר עדיפויות קבוע הקובע כללי אצבע שונים המתאימים למצבים שונים של מערך הכלים בלוח.

את המשחק הכרתי בבית הספר היסודי. לדעתי חוקי המשחק ייחודים- מצד אחד פשוטים ומובנים ואינם מצריכים ידע תיאורטי וניסיון רב כפי שאלו נחוצים בשחמט למשל ומצד שני מאפשרים לפתח טקטיקות רבות ומגוונות.

מטרת העבודה שלי היא יצירת משחק שהמשתמש בו יוכל לשחק נגד המחשב. המחשב משתמש בבינה מלאכותית על מנת להוות יריב לשחקן.

אפיון המערכת

שם המערכת: משחק "אבאלון".

מומחיות המערכת: ניתן לשחק את המשחק ב-2 דרכים שונות:

1. שחקן מול שחקן-כלומר התוכנית מהווה את הפלטפורמה למשחק בין זוג שחקנים. התכנית מציגה לאור בחירת השחקן בכלים עמם רוצה לנוע או לתקוף, את אפשרויותיו למהלך חוקי.

2. שחקן מול מחשב- המחשב מנתח באמצעות פונקציות היוריסטיות את מצבו מול היריב ובוחר במהלך שישפר את מעמדו במשחק תוך שיפור מינימלי של מצב היריב. בחירת המהלך נעשית לפי כללים מוגדרים מראש המכסים מצבים שונים בהם קיימת דרך תגובה הגיונית ומוסכמת.
המהלכים השונים כוללים תגובות הגנתיות ותגובות התקפיות.

קהל יעד: ילדים, בני נוער ומבוגרים ובאופן כללי מי שרוצה לפתח את מיומנויות האסטרטגיה שלו.

תמצית תחום הידע: הצגת המשחק, החוקים והמטרה מפורטים בנספחים.[1]

[1] ראו נספחים, סעיף 1 "הצגת המשחק, החוקים והמטרה".

ייצוג בסיס הידע כתכנית פרולוג

ייצוג המערכת מורכב יחסית. רוב העובדות המיוצגות הן דינמיות-כלומר משתנות במהלך הרצת הקוד. כך למשל: רשימות שתוכנן משתנה בתום פעולת השחקן, עובדות דינמיות שמייצגות את מספר השחקן שכעת תורו לשחק והניקוד הנוכחי של השחקנים. עם זאת, קיימות גם עובדות קבועות כמו: אינדקסים קואורדינטות גרפיות המגדירים את גבולות התאים בלוח ואינדקסי תאים חוקיים. כמו כן ישנן עובדות נוספות אשר אינן משתנות לאורך המשחק אך לא ניתן לייצגן מראש כעובדה פשוטה שכן הן תלויות בבחירת המשתמש. כך למשל בחירת סוג המשחק- שחקן מול שחקן או שחקן מול מחשב.

סוגי נתונים מופשטים לייצוג הידע

רשימת מארזים- בעבודה זו הידע החשוב ביותר לייצוג הוא מצב הלוח של המשחק המשתנה בסיומו של כל תור. בייצוג שבחרתי-משמעות "מצב הלוח" היא המקומות (תאים) על הלוח שתפוסים על ידי כל משתתף- אם זה על ידי המחשב ואם על ידי השחקן. בחרתי שלא לייצג את המקומות הפנויים בלוח (דהיינו מקומות ריקים מנוכחות כלי אחד השחקנים) וזאת משום שניתן להסיק מה הם המקומות הריקים על סמך ידיעת המקומות התפוסים ותוך שימוש בייצוג המגדיר את גבולות הלוח.

בחרתי לייצג את מצב הלוח על ידי שתי רשימות של מארזים: [(X,Y),(X,Y)...], כך שכל מארז מציין כדור אחד (כלי משחק) שמונח על הלוח באינדקס X (בעל ערך מספרי) המציין טור אלכסוני, בשורה Y (בעלת ערך תווי).[2] ישנה רשימה אחת עבור כל שחקן ובא מארז נפרד לכל כדור של אותו השחקן.

לייצוג הרשימתי-דינאמי של כלי המשחק שתי מטרות. האחת להתמודד בנוחות עם הצורך המתמיד לעדכן את מצב הלוח תוך מחיקת כלי משחק מהרשימות או החלפת האינדקס שלהם המייצג את מיקומם. השנייה- שימוש בפעולות מוכנות בשפת Prolog, אשר בעזרתם ניתן לבדוק למשל את מספר האיברים ברשימה, למיין את הרשימה, לבצע פעולות איחוד וחיתוך עם רשימות אחרות וכדומה.

שתי הרשימות מופיעות בהתאמה במשתנים הדינמיים:

```
blacklist(list).
whitelist(list).
```

מארזים- מלבד הצורך בייצוג "מצב הלוח", ייצגתי גם את המצב "המנהלי" של המשחק. במילים אחרות ייצגתי גם באמצעות משתנה דינמי המהווה מארז, את מספר השחקן שכעת חל תורו. בחרתי לייצג את קבוצת הכדורים האדומים במספר 1 ואילו את מספר הכדורים הירוקים במספר 2:

[2] ראו נספחים, תרשים 1: "ייצוג האינדקסים של כל תא בלוח".

turn/1. %Indicates whose the turn is: Black(1) or White (2) . [3]

השתמשתי במארז דינמי במקרה זה שכן ערך התור משתנה כל פעם לאחר מהלך של אחד השחקנים.

כמו כן, ישנו ייצוג דינמי כמארז לניקוד כל שחקן. ניקוד זה של שחקן מייצג את מספר הכלים של היריב אותם הצליח להוציא מחוץ לגבולות הלוח.

השימוש במארז דינמי במקרה זה גם כן נחוץ לאור הנסיבות הדומות- ישנו צורך מתמיד לעדכן את ניקוד השחקנים לאורך המשחק.

scoring/2. %Indicates the scoring of the black and the white players.

Assert and Retract- מדובר במתארים מוכנים בתוכנת פרולוג המשמשים להסרה\הוספה של אותן עובדות דינמיות בהן דנתי לעיל וזאת על ידי הוספה או מחיקה של עובדות בזיכרון המחשב במהלך הרצת התוכנה. השימוש במתוארים אלו נוח ומונע העברת משתנים לחוקים שמסתעפים ולחוקים רקורסיביים. באופן זה נמנעת העמסה על זיכרון המחשב.

על מנת לייצג את המשחק כהלכה, השתמשתי בכמה משתנים דינמיים נוספים מסוג רשימת מארזים ומארזים רגילים, עליהם אפרט במידת הצורך בהמשך שכן הופעתם בתוכנית אינה ראשונית ובסיסית כל כך ואינה רלוונטית לשלב הראשון של ייצוג הידע.

ייצגתי גם את בסיס הידע באמצעות עובדות סטטיות שנשארות קבועות לאורך המשחק ואינן תלויות במהלך השחקן או המחשב:

רשימה- ייצגתי באמצעות רשימה את אינדקסי ה-Y החוקיים בלוח המשחק שהם כאמור תווים. כך אוכל לבדוק האם תא או כדור נמצאים בתחומי הלוח או שמא מחוצה לו.

legal_Y_Indexes(['A','B','C','D','E','F','G','H','I']).

רשימת מארזים-על מנת לממש פונקציה היוריסטית מסוימת באמצעותה מגיב המחשב, ייצגתי ברשימת מארזים את התאים בלוח המשתייכים לכל טבעת בלוח[4] וכן את אינדקס הטבעת. כך אוכל לבדוק עבור תא מסוים באיזו טבעת ממוקם-קרוב לשוליים או שמא במרכז הלוח.

ring(cells_in_ring_list, ring_val).

לדוגמא:

ring([(1,'A'),(2,'A'),(3,'A'),(4,'A'),(5,'A'),(1,'B'),(6,'B'),(1,'C'),(7,'C'),(1,'D'),(8,'D'),(1,'E'),(9,'E'),(2,'F'),(9,'F'),(3,'G'),(9,'G'),(4,'H'),(9,'H'),(5,'I'),(6,'I'),(7,'I'),(8,'I'),(9,'I')],4).

[3] הקוד מכנה את הכדורים שצבעם ירוק בממשק הגרפי כ"לבנים" (white), בעוד שאת אלו שצבעם אדום כ"שחורים" (black).

[4] ראו ערך "Ring" במילון המושגים המקצועיים.

מארזים-ייצגתי באמצעות מארזים את המידע אודות גבולות ערך ה-X של שורה וכן הקואורדינטות הגרפיות של התא הראשון בשורה. באופן זה, אקשור בין ערך השורה לערכי הטור האלכסוני האפשריים בה וכן אוכל לקבוע היכן נקודת ההתחלה הגרפית של השורה.

line_info(line_index, initial_X_val , final_X_val, CoordX_init_cell, CoordY_init_cell).

לדוגמא:

line_info('A',1,5,199,100).

הצגת החוקים העיקריים

חוקים הקשורים בהתחלת המשחק וניהולו

1)

```
intro:-
        new(P,window('Abalone',size(700,700))),
        new(Background,bitmap('intro.bmp')),
        send(P,display,Background,point(0,0)),

        new(TwoPlayers,bitmap('TwoPlayers.bmp')),
        new(Instructions,bitmap('Instructions.bmp')),
        new(AI,bitmap('Ai.bmp')),

        send(P,display,TwoPlayers,point(143,430)),
        send(P,display,Instructions,point(1,342)),
        send(P,display,AI,point(217,0)),

        send(AI,recogniser,click_gesture(left,'',single,and(message(@prolog,startAi,P)))),
        send(TwoPlayers,recogniser,click_gesture(left,'',single,and(message(@prolog,startTwoPlayers,P)))),
        send(Instructions,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage)))),

    send(P,open).
```

הפעולה פותחת את החלון הגרפי של התפריט בו ניתן לבחור את סוג המשחק או קריאת הוראות.
ניתן לבחור לשחק במצב "שני שחקנים (אנושיים)" או "שחקן מול מחשב".

2)

```
startAi(P):-
        send(timer(0.2),delay),
        free(P),
        init_dyn_par,
        init_board(W),
        init_balls,
        drawWithRecBlack(W),
        drawNoRecWhite(W),
        print_scoring(W).
```

הפעולה מטפלת בבחירת המשתמש במשחק "שחקן מול מחשב". הפעולה מקבלת את אובייקט מסך התפריט, סוגרת אותו, מאתחלת את המשתנים הדינמיים השונים שנחוצים לריצת התכנית, בונה חלון לוח משחק, מייצרת את רשימות כלי המשחק ומאתחלת את מיקומם על המסך, מציגה את הכדורים על המסך ומדפיסה את הניקוד.

3)

```
startTwoPlayers(P):-
        send(timer(0.2),delay),
        free(P),
        init_dyn_par,

        retract(aiOrTwoPlayers(_)),
        assert(aiOrTwoPlayers(1)),

        init_board(W),
        init_balls,
        drawWithRecBlack(W),
        drawNoRecWhite(W),
        print_scoring(W).
```

הפעולה מטפלת בבחירת המשתמש במשחק "שחקן מול שחקן". הפעולה מבצעת את אותן פעולות ההכנה שתוארו בסעיף 2, אך גם קובעת את ערך הפרמטר aiOrTwoPlayers ל-1, כדי שבהמשך תנותב ריצת התוכנית לפעולות שלא מערבות בינה-מלאכותית, כלומר לפעולות "שחקן מול שחקן".

4)

```
init_dyn_par:-
        retractall(turn(_)),
        assert(turn(1)),
        retractall(moveList(_)),
        assert(moveList([])),
        retractall(buttonFlag(_)),
        assert(buttonFlag(0)),
        retractall(scoring(_,_)),
        assert(scoring(0,0)),
        retractall(graphicScoring(_,_)),
        new(ScoreB,text(0,center,font(times,bold,30))),
        new(ScoreW,text(0,center,font(times,bold,30))),
        assert(graphicScoring(ScoreB,ScoreW)),
        retractall(whiteList(_)),
        retractall(blackList(_)),
        retractall(arrangementType(_)),
        assert(arrangementType(0)),
        retractall(tempCell(_,_)),
        assert(tempCell(0,0)),
        retractall(broadSideList(_)),
        assert(broadSideList([])),
        retractall(pushList(_)),
        assert(pushList([])),
        retractall(tempWin(_)),
        assert(tempWin(0)),
        retractall(aiOrTwoPlayers(_)),
        assert(aiOrTwoPlayers(0)).
```

הפעולה מאתחלת את כל המשתנים הדינמיים הנחוצים לתוכנית. נשים לב לכמה משתנים חשובים:

התור (turn) מאותחל ל-1, כלומר הכלים האדומים מתחילים (לפי חוקי המשחק). הניקוד (scoring)

מאותחל ל-0:0, אובייקטי הטקסט שמייצגים את הניקוד גם הם מאותחלים ל-0:0, רשימת הכלים האדומים והירוקים (whiteList,blackList) מאותחלים לרשימה ריקה, בחירת המשתמש לשחק מול שחקן אדם או מחשב (aiOrTwoPlayers) מאותחלת כ-0 (ברירת המחדל היא כי מדובר במשחק-"שחקן מול מחשב").

5)

```
init_board(W):-
        new(W,window('Abalone',size(700,700))),

        ((aiOrTwoPlayers(0),new(Background,bitmap('boardAi.bmp')));(aiOrTwoPlayers(1),new(Background,bit
        map('boardTwoPlayers.bmp')))),
        send(W,display,Background,point(0,0)),

        send(W,open).
```

הפעולה מייצרת ומציגה את חלון המשחק. בהתאם לסוג המשחק ("שני שחקנים" או "שחקן מול מחשב"), נקבע הרקע התואם.

6)

```
init_balls:-
        createBallList(1,5,'A',WhiteList1),
        createBallList(1,6,'B',WhiteList2),
        createBallList(3,5,'C',WhiteList3),
        append(WhiteList1,WhiteList2,WhiteAppendedList),
        append(WhiteAppendedList,WhiteList3,WhiteList),

        createBallList(5,9,'I',BlackList1),
        createBallList(4,9,'H',BlackList2),
        createBallList(5,7,'G',BlackList3),
        append(BlackList1,BlackList2,BlackAppendedList),
        append(BlackAppendedList,BlackList3,BlackList),

        assert(whiteList(WhiteList)),
        assert(blackList(BlackList)).
```

הפעולה מייצרת את רשימות הכלים (כדורים) של כל קבוצה-אדומים וירוקים ומאתחלת את המשתנים הדינמיים של שתי הרשימות.

7)

```
drawNoRecWhite(W):-
        whiteList(WL),
        drawNoRec(W,WL,white).

drawWithRecWhite(W):-
        whiteList(WL),
        drawWithRec(W,WL,white).

drawNoRecBlack(W):-
        blackList(BL),
        drawNoRec(W,BL,black).

drawWithRecBlack(W):-
        blackList(BL),
        drawWithRec(W,BL,black).
```

מטרת צמד הפעולות הראשון והשני זהה והשוני נובע רק מסט הכדורים עליו מתבצעת הפעולה.

פעולות drawNoRec מקבלות את חלון המשחק ומציגות בו את הכדורים מהקבוצה הנידונה כך שהכדורים לא יהיו לחיצים-לחיצה עליהם לא תפעיל שום פעולה אחרת.

פעולות drawWithRec מקבלות את חלון המשחק ומציגות בו את הכדורים מהקבוצה הנידונה כך שהכדורים לחיצים-לחיצה עליהם מפעילה פעולה אחרת קבועה (מאפשרת למעשה למשתמש האנושי לבחור את הכדורים עמם רוצה לנוע).

8)

```
stuck(W):-
        turn(1),
        open_error_dialog_stuck,
        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawNoRecWhite(W),
        drawWithRecBlack(W),
        reset_moveList.

stuck(W):-
        turn(2),
        open_error_dialog_stuck,
        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawWithRecWhite(W),
        drawNoRecBlack(W),
        reset_moveList.
```

פעולה זו מופעלת כאשר השחקן אינו יכול לנוע בשום מהלך חוקי עם סט הכדורים שבחר להזיז.

הפעולה מתריעה על מצב תקיעה, מאתחלת את רשימת בחירת הכדורים לרשימה ריקה ומאפשרת לבחור מחדש את הכדורים.

9)

```
switchPlayers(W):-
        turn(2),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecWhite(W),
        drawWithRecBlack(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(1)).
```

```
switchPlayers(W):-
        turn(1),aiOrTwoPlayers(1),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecBlack(W),
        drawWithRecWhite(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(2)).
```

```
switchPlayers(W):-
        turn(1),aiOrTwoPlayers(0),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecBlack(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(2)),
        aiReaction(W).
```

הפעולה מחליפה בין תורות השחקנים: (משאל לימין):

-מתור של שחקן אדום כאדם לזה של שחקן ירוק כמחשב (כאן משתלבות פעולות הבינה המלאכותית).

-מתור של שחקן אדום כאדם לזה של שחקן ירוק כאדם.

-מתור של שחקן ירוק כאדם לזה של שחקן אדום כאדם.

הפעולה בודקת האם התקבל ניצחון של אחד הצדדים.

10)

הפעולה בודקת האם אחד השחקנים ניצח- כלומר צבר 6 נקודות. במידה והתקבל ניצחון מציגה מסך ניצחון שמתאים למנצח.

```
checkWin(W):-
        scoring(_,6),
        send(W,clear),
        new(GreenWins,bitmap('greenWins.bmp')),
        send(W,display,GreenWins,point(0,0)).


checkWin(W):-
        scoring(6,_),
        send(W,clear),
        new(RedWins,bitmap('redWins.bmp')),
        send(W,display,RedWins,point(0,0)).
```

חוקים הקשורים בבחירת השחקן האנושי את מהלכו

בתוכנית, מופיעות עבור רוב החוקים מסוג זה, שתי וריאציות סימטריות-עבור מהלך של שחקן אדום ועבור מהלך של שחקן ירוק. אציג רק וריאציה אחת שכן השנייה סימטרית בלבד-מטרתה, הקלט והפלט שלה זהים.

1)

```
select_balls(W,_,_):-
        view_control_buttons(W),fail.

select_balls(W,X,Y):-
        turn(1),moveList(CurrList),

        ((member((X,Y,HlBall),CurrList),
        select((X,Y,HlBall),CurrList,NNList),
         free(HlBall),
         retract(moveList(_)),
         assert(moveList(NNList)));

        (new(HLBall,bitmap('hlRedBall.bmp')),
         findGraphicalCoords(X,Y,CoordX,CoordY),
         send(W,display,HLBall,point(CoordX,CoordY)),
         append([(X,Y,HLBall)],CurrList,NList)),
         retract(moveList(_)),
         assert(moveList(NList))).
```

מטרת הפעולה לאתחל את הרשימה הדינמית moveList ברשימת הכדורים אותם השחקן רוצה להזיז. הפעולה מופעלת עם בחירת השחקן באחד מהכדורים (X,Y) מקבוצת הכדורים שלו. ראשית כל מופיע על המסך כפתור בקרה שבעזרתו יאשר השחקן את סט הכדורים הסופי שבוחר להזיז. החלק השני של הפעולה מאפשר לערוך את רשימת moveList: השחקן בוחר בכדור מסוים, הכדור מסומן גרפית ומצטרף לרשימה הקיימת. במידה והשחקן בוחר בכדור שכבר נבחר על ידו (כלומר, מסומן כבר גרפית ונמצא ברשימה), תבוטל בחירת הכדור-הכדור יצא מרשימת הנבחרים והסימון ימחק. ניתן לערוך את רשימת הכדורים הנבחרים כל עוד מקש הבקרה שהופיע לא נלחץ.

(2

```
show_move_options(Button,W):-
        free(Button),
        retract(buttonFlag(_)),
        assert(buttonFlag(0)),

        check_legality,

        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawNoRecWhite(W),
        drawNoRecBlack(W),
        restart_hlBalls_maat(W),

        mark_move_and_push_options(W).

show_move_options(_,_):-
        open_error_dialog_choice,

        moveList(ML),
        clear_hlBalls(ML),
        reset_moveList.
```

פעולה זו מופעלת עם אישורו של השחקן על בחירתו בסט הכדורים שברצונו להזיז, זאת ע"י לחיצה על המקש המתואר לעיל. הפעולה מקבלת את אובייקט החלון ואובייקט המקש. המקש נמחק, נבדקת חוקיות הסט הנבחר (מספר כדורים חוקי, צורת ארגון חוקית-אופקי או אלכסוני) ובמידה והסט חוקי מתבטלת האפשרות לבחור בכדורי הקבוצה שכעת תורה, מסומנים הכדורים שנבחרו ומופעלת פעולת הצגת המהלכים האפשריים. במידה והסט לא חוקי, מופיע חלון שגיאה, moveList מתאפס, סימון הכדורים הנבחרים מתבטל וניתנת אפשרות לבחור מחדש.

(3

```
check_legality:-
        moveList(MList),
        sort(MList,SMList),
        retract(moveList(_)),
        assert(moveList(SMList)),

        length(MList,Len),
        Len>0,Len=<3,

        ((horiz_leg(SMList),retract(arrangementType(_)),assert(arrangementType(1)));
        (numDiag_leg(SMList),retract(arrangementType(_)),assert(arrangementType(2)));
        (mixDiag_leg(SMList)),retract(arrangementType(_)),assert(arrangementType(3))).
```

הפעולה בודקת האם סט הכדורים שנבחר חוקי. הפעולה מאתחלת את moveList ברשימת הבחירה הממוינת. נבדק גודלה של רשימה זו וכן צורת סידור הכדורים-אופקי או num diagonal או mix

diagonal[5] תוך צמידות הכדורים אחד לשני. כמו כן, המשתנה הדינאמי arrangementType מאותחל באחד הערכים 1,2,3 בהתאם לצורת סידור הכדורים וזאת לשימוש עתידי.

(4

```
mark_move_and_push_options(W):-
        moveList([(X,Y,_)]),
        list_of_free_cells_around_ball(X,Y,FCL),

        ((length(FCL,0),stuck(W));
        (highLightAvailCell(W,FCL))).

mark_move_and_push_options(W):-
        scan_inline(W,SBroadsideMove,InLineListLen),
        scan_broadside(W,SBroadsideMove,CheckValBM),
        scan_push(W,CheckValPush),
        writeln('Results'+InLineListLen+CheckValBM+CheckValPush),
        (((InLineListLen \= 0);(CheckValBM \= 0);(CheckValPush \= 0));(stuck(W))).
```

הפעולה תואמת לשני מצבים: מצב בו נבחר כדור אחד או מצב בו נבחרו מספר כדורים לתזוזה. מטרת הפעולה להציג את כל המהלכים האפשריים ולאפשר למשתמש לבחור אחד מהם.

במידה ומדובר בכדור אחד-השחקן יכול להזיז את הכדור לכל אחד מו התאים הצמודים והפנויים. הפעולה מתריעה אם הכדור תקוע (אין אפשרויות למהלך) ומאפשרת לשחקן לבחור כדורים מחדש.

במידה ומדובר בסט כדורים- מריצים את פעולת אפשרויות מהלכי ה-inline[5], מריצים את פעולת אפשרויות מהלכי ה-broadside[5] ומריצים את פעולת אפשרויות ההדיפה (scan_push). כל אחת מהפעולות שתיארתי לעיל מחזירה ערך דגל המצביע על כך שקיימים מהלכים אפשריים. במידה ואין אפשרויות למהלך-הפעולה מתריעה על מצב היתקעות ומאפשרת לשחקן לבחור כדורים מחדש.

(5

```
scan_inline(W,SBroadsideMove,InLineListLen):-
        moveList(ML),
        list_of_free_cells_around_set(ML,FCL),

        ((arrangementType(1),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(CML,SCML),
        horiz_leg(SCML)),InlineMove));
        (arrangementType(2),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(CML,SCML),
        numDiag_leg(SCML)),InlineMove));
        (arrangementType(3),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(CML,SCML),
        mixDiag_leg(SCML)),InlineMove))),
        highLightAvailCell(W,InlineMove),
        subtract(FCL,InlineMove,BroadsideMove),
        sort(BroadsideMove,SBroadsideMove),
        length(InlineMove,InLineListLen).
```

הפעולה מוצאת תחילה את כל המקומות (תאים) הפנויים מסביב לסט הכדורים הנבחר. בהתאם לצורת סידור הכדורים הנבחרים ורשימת המקומות הפנויים מסביב לתאים, בודקים מה הם כל

[5] ראו "מילון מושגים מקצועיים".

המצבים בהם ניתן להזיז את סט הכדורים בתנועת inline (בפועל, בונים רשימה לכל מצב שכזה. כל מצב מיוצג ברשימה ממוזגת של סט הכדורים והתא הפנוי שבאותו הכיוון).
הפעולה מציגה ויזואלית את אפשרויות הבחירה השונות לתנועת inline, עליהם יכול ללחוץ המשתמש. כמו כן, הפעולה מחסירה מרשימת המקומות הפנויים מסביב לסט את התאים הפנויים שמאפשרים תנועת inline ובכך מכינה את רשימת התאים הפנויים שבצדי הציר עליו מונח הסט, זאת לשימוש בפעולת broadSide. הפעולה מחזירה את מספר האפשרויות לתנועה שנמצאו וכן את הרשימה החדשה.

6)

```
scan_push(W,CheckVal):-
        turn(1),
        moveList(ML),
        whiteList(WL),

        ((arrangementType(1),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CML,SCML),
        horiz_leg(SCML)),PushOpt));
        (arrangementType(2),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CML,SCML),n
        umDiag_leg(SCML)),PushOpt));
        (arrangementType(3),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CML,SCML),
        mixDiag_leg(SCML)),PushOpt))),

        ((length(PushOpt,0),CheckVal is 0);
        (build_push_list_maat(PushOpt,PushList),
        fail_Push_List_combined_colour(PushList,NPushList),
        fail_Push_List_too_long(ML,NPushList,NNPushList),
        ((length(NNPushList,0),CheckVal is 0);
        (retract(pushList(_)),
         assert(pushList(NNPushList)),
         view_Push_options(W,NNPushList,1),CheckVal is 1)))).
```

הפעולה בודקת את אפשרויות דחיפת הכדורים הירוקים ע"י סט כדורים אדומים (קיימת פעולה נוספת סימטרית). בהתאם לצורת הסידור של סט הכדורים, מחפשים כדור ברשימת הירוקים שאם לצרפו לסט הכדורים הדוחף, ייווצר סידור כדורים חדש חוקי. במידה ואין כדורים כאלו (אין אפשרות לדחוף) יוחזר ערך 0, אחרת בודקים האם רשימת הכדורים הפוטנציאליים לדחיפה (אוסף הכדורים שמתחיל בכדור שנמצא ומסתיים בכדור שבאותו הציר לפני תא ריק או שולי הלוח) כולל כדורים בצבעים שונים (ניסיון לדחוף כדורים בצבעים מעורבים לא חוקי) או אורכו גדול שווה לאורך רשימת הכדורים הדוחפים (לפי חוק "סומיטו"-ניתן לדחוף רק מספר קטן יותר של כדורים). פוסלים מקרים כאלו-שנזכרו לעיל. במידה ונשארו אפשרויות דחיפה חוקיות מציגים אותן גרפית ומחזירים ערך דגל מספרי 1 המציין כי נמצאו אפשרויות דחיפה.

7)

```
push(W,N):-
        turn(1),
        tempWin(TW),
        free(TW),
        moveList(ML),
        pushList(PL),
        blackList(BL),
        whiteList(WL),

        nth1(N,PL,ChosenPushOpt),
        findall((MLX,MLY),member((MLX,MLY,_),ML),NML),
        sort(ChosenPushOpt,SCPO),
        last(NML,LNML),last(SCPO,LSCPO),
        findHigherValCell(LNML,LSCPO,Max),
        ((member(Max,NML),(reverse(NML,RNML),reverse(SCPO,RSCPO),pushedList1(RNML,PNML),
        pushedList1(RSCPO,PSCPO)));
        (member(Max,SCPO),pushedList2(NML,PNML),pushedList2(SCPO,PSCPO))),

        subtract(BL,NML,NBL),append(NBL,PNML,FinalBL),
        subtract(WL,SCPO,NWL),append(NWL,PSCPO,FinalWL),

        subtract(FinalBL,[('*','*')],NFinalBL),
        subtract(FinalWL,[('*','*')],NFinalWL),

        retract(blackList(_)),
        assert(blackList(NFinalBL)),
        retract(whiteList(_)),
        assert(whiteList(NFinalWL)),
        switchPlayers(W).
```

הפעולה מופעלת עם לחיצת המשתמש על כדורי האויב אותם רוצה לדחוף לאור בחירתו. האפשרות שבחר מסומנת כבר בפעולת הצגת אפשרויות הדחיפה (view_Push_options) במספר סידורי, כך שניתן כעת לברור מתוך כלל אפשרויות הדחיפה השמורות ברשימת PL, את האפשרות שמספרה N שנשלחת מפעולת הצגת אפשרויות הדחיפה, היא זו שנבחרה ע"י המשתמש. הפעולה מוצאת את הרשימה החדשה של כדורים דוחפים ונדחפים, שלאחר דחיפת כדורי האויב (יש שתי פעולות שמוצאות את הרשימה החדשה - בהתאם לכיוון הדחיפה-ימינה או שמאלה). כדורים שנדחפו מחוץ ללוח סומנו בכוכביות מבעוד מועד בפעולת pushedList1 או pushedList2 ואותם הפעולה מוחקת. הפעולה מעדכנת את רשימת הכדורים האדומים והירוקים ומחליפה את התור.

8)

```
scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(1),
        moveList([(_,Y,_)|_]),
        next_char(Y,NY),
        previous_char(PY,Y),
        findall((ValX,PY),member((ValX,PY),SBroadsideMove),RightList),
        findall((ValX,NY),member((ValX,NY),SBroadsideMove),LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).

scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(2),
        moveList([(X,_,_)|_]),
        NX is X+1,
        PX is X-1,
        findall((NX,ValY),member((NX,ValY),SBroadsideMove),RightList),
        findall((PX,ValNY),member((PX,ValNY),SBroadsideMove),LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).

scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(3),
        moveList(ML),
        findall((ValX,ValY),(member((ValX,ValY),SBroadsideMove),((next_char(ValY,YML),member((ValX,Y
        ML,_),ML));(XML is (ValX-1),member((XML,ValY,_),ML)))),RightList),
        subtract(SBroadsideMove,RightList,LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).
```

הפעולה בהתאם לצורת הסידור של קבוצת הכדורים שנבחרה, מחלקת את רשימת SbroadsideMove (שכוללת את התאים הפנויים שמסביב לסט הכדורים הנבחר ומונחים על הצירים הצמודים והמקבילים לציר עליו מונחים הכדורים הנבחרים) לשתי רשימות- רשימת תאים שמצד ימין לסט הכדורים ורשימת תאים שמצד שמאל לסט הכדורים. שתי רשימות אלו נשלחות להמשך טיפול בפעולת broadside_option_list.

9)

```
broadside_option_list(W,RightList,LeftList,CheckVal):-
        sort(RightList,SRightList),
        sort(LeftList,SLeftList),

        moveList(ML),
        length(ML,MLlen),
        subList(SRightList,MLlen,Opt1),
        subList(SLeftList,MLlen,Opt2),

        reverse(SRightList,RevSRightList),
        reverse(SLeftList,RevSLeftList),

        subList(RevSRightList,MLlen,Opt3),
        subList(RevSLeftList,MLlen,Opt4),

        sort(Opt1,SOpt1),
        sort(Opt2,SOpt2),
        sort(Opt3,SOpt3),
        sort(Opt4,SOpt4),

        append([SOpt1],[SOpt2],List1),
        append(List1,[SOpt3],List2),
        append(List2,[SOpt4],FinalList),

        sort(FinalList,SFinalList),
        delete(SFinalList,[],SSFinalList),

        find_only_valid(SSFinalList,SSSSFinalList),

        ((not(highLight_broadside_options_maat(W,SSSSFinalList)),CheckVal is 0);(CheckVal is 1)).
```

הפעולה מקצה משתי רשימות התאים המקבילות והצמודות לסט הכדורים הנבחר ארבע רשימות תאים שגודלן כגודל סט הכדורים הנבחר. כל רשימה שכזו מייצגת אפשרות אליה יכול השחקן להזיז את הסט הנבחר ב-"תנועה-הצידה"[6]. הפעולה בונה רשימה המכילה את הרשימות הנ"ל ומוחקת את הרשימות הריקות ואת אלו שאינן יוצרות צורת סידור תאים חוקית. האפשרויות השונות ל"תנועה-הצידה" מוצגות באופן גרפי בפעולת highLight_broadside_options_maat ומוחזר ערך דגל 1 המסמן כי קיימות אפשרויות לתנועה שכזו. במידה ואין אפשרויות כאלו מוחזר ערך דגל 0.

[6] ראו "מילון מושגים מקצועיים".

10)

```
broadSide(W,TempWin,N):-
        turn(1),
        free(TempWin),
        broadSideList(FinalList),
        blackList(BL),
        nth0(N,FinalList,BSList),
        moveList(ML),
        findall((X,Y),member((X,Y,_),ML),NML),
        subtract(BL,NML,NBL),
        append(NBL,BSList,NNBL),
        retract(blackList(_)),
        assert(blackList(NNBL)),
        switchPlayers(W).
```

לאחר בחירת אפשרות כלשהי מבין אפשרויות לתזוזה ב-"תנועה-הצידה" שמוצגות למשתמש בחלון גרפי נפרד, פעולה זו מופעלת. הפעולה סוגרת את חלון הבחירה, מסירה את רשימת הכדורים שבחר המשתמש להזיז מרשימת הכדורים של השחקן ומוסיפה לרשימה זו את רשימת התאים אליהם מזיז השחקן את הכדורים שבחר ב"תנועה הצידה". כמו כן, הפעולה מחליפה את התור.

11)

```
inline(W,NX,NY):-
        turn(1),
        blackList(BL),
        moveList([(X,Y,_)]),
        select((X,Y),BL,NBL),
        append([(NX,NY)],NBL,NNBL),
        retract(blackList(_)),
        assert(blackList(NNBL)),
        switchPlayers(W).

inline(W,NX,NY):-
        turn(1),
        tempWin(TempWin),
        free(TempWin),
        blackList(BL),
        moveList(ML),
        append([(NX,NY,_)],ML,Temp),
        sort(Temp,STemp),
        nth1(1,STemp,(FX,FY,_)),
        last(STemp,(LX,LY,_)),
        append([(NX,NY)],BL,NBL),
        (((not((FX is NX, FY == NY)),select((FX,FY),NBL,NNBL));(not((LX is NX, LY ==
        NY)),select((LX,LY),NBL,NNBL))),
        retract(blackList(_)),
        assert(blackList(NNBL))),
        switchPlayers(W).
```

לאחר שבחר השחקן את התא הפנוי אליו ינוע כדור אחד או סט כדורים (כדור אחר כדור בתנועת inline) מופעלת פעולה זו לכדור אחד או לסט בהתאמה. הפעולה משנה את רשימת הכדורים של

השחקן כך שיכילו את שיעורי הכדורים במקומם החדש (לאחר שהוזזו בתנועת inline). לאחר מכן הפעולה מחליפה את התור. יש לשים לב כי במקרה של סט כדורים, הפעולה סוגרת את מסך אפשרויות ה-"תנועה-הצידה" TempWin במידה וקיימות אפשרויות כאלו וחלון העזר נפתח.

תיאור האלגוריתם-בינה מלאכותית

כפי שתואר בהקדמה, שתי הגישות העיקריות בהם משתמשים לבניית משחק ה-"אבלון" הן:

א) אלגוריתם MINMAX ב) פונקציות היוריסטיות שמתבססות על כללי אצבע

המשחק מתבסס על אפשרות ב). אמנם אלגוריתם MINMAX מוביל ליחס ניצחונות:הפסדים גבוה יותר אך עץ המצבים במשחק גדול במיוחד בשל המספר הרב של המהלכים אפשריים בכל תור. עץ מצבים שכזה מעמיס על הזיכרון ומפריע לריצה החלקה של המשחק. מצד שני ניתן אמנם להגביל את מספר הרמות בעץ-כלומר האלגוריתם ינתח רק מספר קבוע וקטן של צעדים מראש אך הסיכוי להוביל לניצחון המחשב יקטן. על כן בעבודה זה מיושמות פונקציות היוריסטיות.
חשוב לציין כי באלגוריתם שיוצג הפונקציות ההיוריסטיות נעזרות בפעולות שמנתחות את מצב הלוח ומחזירות כפלט ערך מספרי-באופן כזה הפונקציה ההיוריסטית מעריכה באיזה מהלך לבחור מבין האפשרויות השונות.

להלן פעולות הערכת מצב הלוח:

1)

```
manhatten_dist_from_center_calc_maat(Val):-
        whiteList(WL),
        manhatten_dist_from_center_calc(WL,Val).

manhatten_dist_from_center_calc([],0).
manhatten_dist_from_center_calc([(X,Y)|Other],Val):-
        ring(List,Dist),
        member((X,Y),List),
        manhatten_dist_from_center_calc(Other,NVal),
        Val is NVal+Dist.
```

אחת הטקטיקות שמובילות לניצחון במשחק היא לכבוש את אזור מרכז הלוח כבר בשלבים המוקדמים של המשחק. באופן זה, היריב נאלץ לפרוש את כליו בקרבת שולי הלוח במקומות בהם קל יחסית להדוף את כדוריו אל מחוץ ללוח. מצד שני, השחקן שממקם את כליו במרכז הלוח מוגן מהדיפת כדוריו אל מחוץ ללוח.
הפעולה מסכמת את כל "מרחקי מנהטן"[7] שבין כל כדור בקבוצה הירוקה (המחשב) לתא המרכזי בלוח (5,'E'). כך ניתן לקבוע עד כמה כדורי הקבוצה הירוקה קרובים למרכז הלוח. **ככל שהערך המספרי גדול יותר, כך הכדורים מרוחקים יותר מהמרכז.**

[7] ראו "מילון מושגים מקצועיים".

2)

```
compactness_calc_maat(Val):-
        whiteList(WL),
        compactness_calc(WL,0,0,DoubleVal),Val is DoubleVal/2,!.

compactness_calc_maat(WL,Val):-
        compactness_calc(WL,0,0,DoubleVal),Val is DoubleVal/2,!.

compactness_calc(WL,I,_,0):-
        length(WL,I).

compactness_calc(WL,I,J,Val):-
        length(WL,J),
        NI is I+1,
        compactness_calc(WL,NI,0,Val).

compactness_calc(WL,I,J,Val):-
        nth0(I,WL,(X1,Y1)),
        nth0(J,WL,(X2,Y2)),
        manhatten_min_distance_maat((X1,Y1),(X2,Y2),Dist),
        NJ is J+1,
        compactness_calc(WL,I,NJ,OtherDist),
        Val is Dist+OtherDist.
```

בנוסף לחשיבות שבשמירת הכדורים בקרבת מרכז הלוח, חשוב גם לשמור על הכדורים במבנה קומפקטי ככל האפשר-כלומר רצוי שהכדורים יהיו מסודרים כגוש ולא בטור או כיחידים מפוזרים. במבנה גושי, הכדורים יכולים למעשה להדוף את היריב בכל כיוון אפשרי (במידה ומספר הכדורים ההודף גדול ממספר הכדורים הנהדף, כמובן).

הפעולה compactness_calc_maat מתאימה לשני מצבים: בדיקת ערך הקומפקטיות של מערך הכדורים הירוקים במצב הקיים בלוח-לפי הרשימה שבמשתנה whiteList וכן למצב בו ניתנת לפעולה רשימה של כדורים שאינה רשימת הכדורים בפועל במשחק.

הפעולה compactness_calc מסכמת את "מרחקי מנהטן"[7] שבין כל זוג כדורים ירוקים (הערך הסופי כולל חזרות בין זוגות ולכן ערך הקומפקטיות הוא הערך שנמצא מחולק ב-2). **ככל שהערך המספרי קטן יותר, כך הקומפקטיות גדולה יותר.**

(3

```
list_and_num_of_threatened_pieces_maat(SThreatenedList,Val):-
	whiteList(WL),
	blackList(BL),
	findall([(HostileLen),(X,Y)],(member((X,Y),WL),is_threatened(WL,BL,X,Y,HostileLen)),ThreatenedList)
,
	sort(ThreatenedList,SThreatenedList),
	length(ThreatenedList,Val).

list_and_num_of_threatened_pieces_maat(WL,BL,SThreatenedList,Val):-
	findall([(HostileLen),(X,Y)],(member((X,Y),WL),is_threatened(WL,BL,X,Y,HostileLen)),ThreatenedList)
,
	sort(ThreatenedList,SThreatenedList),
	length(ThreatenedList,Val).
```

כלי מאוים הוא כלי שמוקף ביותר כלים עוינים מכלים מקבוצתו. הפעולה בונה רשימה של כל הכלים המאוימים מקבוצת הכדורים הירוקים שנמצאים בשולי הלוח (טבעות מספר 3 ו-4). לצד כל קוארדינטת כלי מאוים מצורף גם מספר הכדורים העוינים שסביבו. הפעולה מחזירה את הרשימה ומספר הכדורים המאוימים (אורך הרשימה). גם במקרה זה הפעולה מתמודדת עם רשימות הכדורים שמתארות את מצב הלוח הקיים אך גם רשימות שנשלחות לפעולה ואינן רשימות הכדורים שמונחים על הלוח.

(4

```
list_of_horiz_pairs(List,BallList):-
	findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),horiz_leg([(X1,Y1,_),(X
	2,Y2,_)])),List).

list_of_numDiag_pairs(List,BallList):-
	findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),numDiag_leg([(X1,Y1,
	_),(X2,Y2,_)])),List).

list_of_mixDiag_pairs(List,BallList):-
	findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),mixDiag_leg([(X1,Y1,_
	),(X2,Y2,_)])),List).

list_of_horiz_triple(TripList,BallList):-
	list_of_horiz_pairs(List,BallList),
	findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),compatibilit
	yConvertion(TripleList,TempTripleList),horiz_leg(TempTripleList)),TripList).

list_of_numDiag_triple(TripList,BallList):-
	list_of_numDiag_pairs(List,BallList),
	findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),compatibilit
	yConvertion(TripleList,TempTripleList),numDiag_leg(TempTripleList)),TripList).

list_of_mixDiag_triple(TripList,BallList):-
	list_of_mixDiag_pairs(List,BallList),
	findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),compatibilit
	yConvertion(TripleList,TempTripleList),mixDiag_leg(TempTripleList)),TripList).
```

הפעולות הללו מקבלות רשימה של כלי משחק אדומים או ירוקים ומחזירות את רשימת כל זוגות הכדורים בצורות סידור שונות (אופקי או אלכסון "\ " או אלכסון "/") ורשימת כל שלישיות הכדורים בצורות סידור שונות.

הפעולות הבאות עוסקות במציאת סטים חוקיים שונים של כדורים וזאת על מנת למצוא מאוחר יותר את המהלכים השונים האפשריים. **פעולות אלו סימטריות עבור הקבוצה האדומה והירוקה ולכן אציג רק אפשרות אחת מכל פעולה-כלומר פעולה מכל סוג המתמקדת בצבע אחד**:

5)

```
all_avail_groups_white(List):-
        whiteList(WL),
        list_of_horiz_pairs(List1,WL),
        list_of_numDiag_pairs(List2,WL),
        list_of_mixDiag_pairs(List3,WL),
        list_of_horiz_triple(TripList1,WL),
        list_of_numDiag_triple(TripList2,WL),
        list_of_mixDiag_triple(TripList3,WL),
        findall([Item],member(Item,WL),WLList),
        append([WLList,List1,List2,List3,TripList1,TripList2,TripList3],List).
```

הפעולות הללו מחזירות רשימה של כל זוג ושלישיות כדורים בכל צורת סידור חוקית עבור רשימת הכדורים הירוקים או האדומים. כמו כן ניתן לשלוח רשימה של כדורים שאינם הכדורים שמונחים על הלוח במצב הקיים במשחק.

6)

```
whiteBall_push_list(List):-
        all_avail_groups_white(White),
        all_avail_groups_black(Black),
        findall((Set),(member(Group1,White),member(Group2,Black),append(Group1,Group2,Set),(length(Group
1,Len1),length(Group2,Len2),Len1>Len2),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_l
eg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List1),
        findall((Set),(member(Group2,White),member(Group1,Black),append(Group1,Group2,Set),(length(Group
1,Len1),length(Group2,Len2),Len1<Len2),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_l
eg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List2),
        append(List1,List2,List).
```

הפעולות הללו יוצרות רשימות שבכל אחת מהן סט כדורים הכולל את סט הכדורים הדוחף מצבע אחד וסט כדורים נדחף מצבע שני. הפעולה בודקת האם יש אפשרות לקבוצה האחת לדחוף את הכדורים של הקבוצה השניה-כלומר בודקת האם יש תא ריק או שולי לוח אחרי הכדור הנדחף האחרון. שתי רשימות הכדורים-כדורים דוחפים וכדורים נדחפים, ממוזגות לרשימה אחת.

7)

```
whiteBall_inline_list(List):-
	allCellsList(AllCellsList),
	appended_black_and_white(AL),
	all_avail_groups_white(White),
	findall((Set),(member(Group,White),member(Item,AllCellsList),not(member(Item,AL)),append([Item],Gr
oup,Set),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvS
et))),List1),
	findall((Set),(member(Group,White),member(Item,AllCellsList),not(member(Item,AL)),append(Group,[Ite
m],Set),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet
))),List2),
	append(List1,List2,List).
```

הפעולה הזו (אין פעולה נוספת סימטרית לכדורים אדומים) בונה רשימה ובה כל סטי הכדורים בצבע ירוק שיכולים לנוע בתנועת inline (מבלי לדחוף כדורים מצבע אדום). כלומר, מדובר ברשימת כל סטי הכדורים הירוקים שאחריהם או לפניהם תא ריק.

8)

```
push_out_push_in_maat_white(PushOutBoardList,PushInBoardList):-
	whiteBall_push_list(List),
	push_out_push_in_white(List,PushOutBoardList,PushInBoardList).
```

פעולות אלו מחלקות את הרשימה List שבה תת-רשימות של סטי כדורים דוחפים מצבע אחד ונדחפים מצבע שני לרשימת סטים בהם הדחיפה מבוצעת בתחום הלוח (PushInBoardList) ולרשימת סטים בהם הדחיפה מבוצעת אל מחוץ ללוח (PushOutBoardList).

9)

```
pushedList_white(List,PL):-
	arrangementType(List,Type),
	sort(List,SList),
	blackList(BL),
	((nth1(1,SList,Item),member(Item,BL),reverse(SList,Rev),pushedList_1(Rev,PL,Type));
	(pushedList_2(SList,PL,Type))),!.
```

פעולות אלו מקבלות רשימה שבה סט כדורים דוחף מצבע אחד וסט כדורים נדחף מצבע שני כך שהצבע בכותרת הפעולה תואם לצבע הקבוצה הדוחפת. הפעולה מחזירה את רשימת הכדורים לאחר שהדחיפה בוצעה.

בהתבסס על הפעולות שמנתחות את מצב הלוח ומפורטות לעיל, יוצגו כעת פעולות הבינה המלאכותית. **חשוב לציין: בעת ריצת התוכנית, כל פעולת בינה נסרקת ואם התנאים בה תואמים למצב הלוח, הפעולה מיושמת וכך למעשה מיושמת תגובת המחשב. במידה והפעולה לא מתאימה למצב הלוח, נסרקת פעולת הבינה הבאה בתור בסדר הופעתה וכך הלאה. כל פעולת בינה מייצגת כלל אצבע אחר. סדר הופעת הפעולות תואם לחשיבות כלל האצבע והלוגיקה האנושית. סדר זה נשמר גם בפירוט שיבוא כעת:**

1) כלל אצבע: במידה וקיימת סכנה מידית להדיפת כדור מחוץ ללוח, יש להזיזו כך שתמנע ההדיפה. כאשר מול היריב ניצבות אפשרויות הדיפת כדור רבות, יש לסכל את ההדיפה שאילו התבצעה הייתה גורמת לנזק הגדול ביותר למחשב.

```
aiReaction(W):-
        whiteList(WL),
        push_out_push_in_maat_black(PushOutBoardList,_),
        length(PushOutBoardList,Len),Len\=0,

         writeln('1'),

         chooseWorstBlackPush(PushOutBoardList,WorstPush),
        ((nth1(1,WorstPush,(X,Y)),member((X,Y),WL),findBestOptToMoveCell2(X,Y,ResultWL,ResultBL));
        (last(WorstPush,(X,Y)),findBestOptToMoveCell2(X,Y,ResultWL,ResultBL))),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).
```

הפעולה מוצאת בעזרת push_out_push_in_maat_black את הסטים שבהם הכדורים האדומים דוחפים אל מחוץ ללוח כדורים ירוקים. במידה ויש סטים כאלו, כלומר במידה וקיימת סכנה של הדיפת כדורים ירוקים מחוץ ללוח, הפעולה chooseWorstBlackPush בוחרת את ההדיפה הגרועה ביותר עבור המחשב. הפעולה מוצאת את שיעורי הכדור שנמצא בסכנת הדיפה ומריצה את findBestOptToMoveCell2 אשר בודקת מהו המהלך הטוב ביותר להזזת הכדור המאוים כך שהסיכול אף יועיל לשיפור מצבו של המחשב. הפעולה מציגה את המצב החדש של הלוח, לאחר מהלך הסיכול ומחליפה תורות.

כיצד בוחר אפוא המחשב את ההדיפה הגרועה ביותר אותה יסכל?

```
chooseWorstBlackPush(PushOutBoardList,WorstPush):-
        findall((Val1,Val2,Item),(member(Item,PushOutBoardList),boardAfterPushOut_black(Item,NWL,NBL),c
        omparePotentialPushes(NWL,NBL,Val1),compareThreatenedPieces(NWL,NBL,Val2)),Options),
        sort(Options,SOptions),
        last(SOptions,(_,_,WorstPush)).
```

הפעולה מקבלת את רשימת ההדיפות האפשריות, עבור כל הדיפה שכזו בודקת מה הן הרשימות החדשות של הכדורים הירוקים והאדומים אילו ההדיפה הייתה מבוצעת. כעת מושווה מספר ההדיפות מחוץ ללוח שבאפשרותו של היריב לבצע במצב החדש לעומת במצב הקיים. ההפרש בין ערכים אלו מושם בVal1. השוואה זו חשובה שכן נעדיף לסכל הדיפה שאילו התבצעה הייתה גוררת יצירת הדיפות פוטנציאליות נוספות, לרעת המחשב. כמו כן עבור כל הדיפה שברשימת ההדיפות מושווה מספר הכדורים המאוימים במצב הקיים מול מספר הכדורים המאוימים לאחר ההדיפה הנבדקת. ההפרש בין הערכים מושם בVal2. גם כאן, נעדיף לסכל הדיפה שאילו בוצעה הייתה גוררת יצירת כדורים מאוימים נוספים בקבוצת הכדורים של המחשב. הפעולה ממיינת את האפשרויות לפי Val1 ו-Val2 ומחזירה את האפשרות הגרועה ביותר עבור המחשב.

כיצד בוחר המחשב את המהלך להזזת הכדור המאוים (על מנת לסכל את ההדיפה הגרועה ביותר)?

```
findBestOptToMoveCell2(X,Y,ResultWL,ResultBL):-
        whiteBall_inline_list(InlineList),
        push_out_push_in_maat_white(PushOutBoardList,PushInBoardList),


        findall((Val1,Val2,NWL,NBL),(member(Item,InlineList),member((X,Y),Item),boardAfterInline(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maat_black(NWL,NBL,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),InlineOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushInBoardList),member((X,Y),Item),boardAfterPushIn_white(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maat_black(NWL,NBL,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),PushInOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushOutBoardList),member((X,Y),Item),boardAfterPushOut_white(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maat_black(NWL,NBL,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),PushOutOpts),

        append([InlineOpts,PushInOpts,PushOutOpts],BestOptList),
        sort(BestOptList,SBestOptList),
        nth1(1,SBestOptList,(_,_,ResultWL,ResultBL)).
```

הפעולה מקבלת את שיעורי כדור המאוים. הפעולה בעזרת whiteBall_inline_list ו-push_out_push_in_maat_white, מוצאת את מהלכי התנועה האפשריים של הקבוצה הירוקה. עבור כל אחד מהמהלכים שמערבים את הזזת הכדור המאוים, משווים בין הקומפקטיות במצב הנוכחי לבין הקומפקטיות שלאחר ביצוע המהלך הנבדק. הפרש בין ערכים אלו (Val2) מצביע על טיב המהלך תוך מתן תשומת לב לשיפור מצבו של המחשב. כמו כן, עבור כל מהלך שכזה בודקים מהו המספר החדש של אפשרויות הדיפת הכדורים הירוקים מחוץ ללוח כתוצאה מביצוע המהלך הנבדק. מספר זה מושם ב-Val1. נעדיף את המהלך שגורר את מספר אפשרויות היריב הנמוך ביותר להדוף כדורים ירוקים החוצה. הפעולה מחזירה את רשימות הכדורים הירוקים והאדומים שלאחר המהלך שמניב רווח טוב ביותר למחשב.

2) כלל אצבע: במקרה שבו כדור שנמצא באזור הטבעות החיצוניות, בקרבת השוליים והוא מאוים- כלומר מספר הכדורים העוינים סביבו גדול ממספר הכדורים מאותה הקבוצה שסובבים אותו, יש לנסות להזיז את הכדור במטרה לשפר את מצב קבוצת המחשב-צמצום מספר הכדורים המאוימים תוך הרחקתם מהשוליים.

```
aiReaction(W):-
        list_and_num_of_threatened_pieces_maat(SThreatenedList,Val),Val\=0,

        writeln('2'),

        last(SThreatenedList,([_,(ThX,ThY)])),
        findBestOptToMoveCell1(ThX,ThY,ResultWL,ResultBL),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).
```

הפעולה מוצאת בעזרת list_and_num_of_threatened_pieces_maat את רשימת הכדורים המאוימים של הקבוצה הירוקה המונחים בטבעות החיצוניות. במידה וקיימים כאלו כדורים, הפעולה בוחרת ל"הציל" את הכדור שסביבו מספר הכדורים העוינים הגדול ביותר. עבור כדור זה הפעולה findBestOptToMoveCell1 בוחרת את המהלך הטוב ביותר שבו המחשב ישפר את מצבו ומחזירה את רשימת הכדורים האדומים והירוקים במצב החדש של הלוח. הפעולה מציגה את המצב החדש של הלוח ומחליפה תורות.

```
findBestOptToMoveCell1(X,Y,ResultWL,ResultBL):-
        whiteBall_inline_list(InlineList),
        push_out_push_in_maat_white(PushOutBoardList,PushInBoardList),

        findall((Val1,Val2,NWL,NBL),(member(Item,InlineList),member((X,Y),Item),boardAfterInline(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCompactness(NWL,Val2)),InlineOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushInBoardList),member((X,Y),Item),boardAfterPushIn_white(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCompactness(NWL,Val2)),PushInOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushOutBoardList),member((X,Y),Item),boardAfterPushOut_white(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCompactness(NWL,Val2)),PushOutOpts),

        append([InlineOpts,PushInOpts,PushOutOpts],BestOptList),
        sort(BestOptList,SBestOptList),
        nth1(1,SBestOptList,(_,_,ResultWL,ResultBL)).
```

הפעולה מקבלת את שיעורי הכדור המאוים. הפעולה מוצאת את כל אפשרויות המהלך של השחקן הירוק ומשווה עבור כל מהלך בו הכדור המאוים מעורב את מספר הכדורים המאוימים לאחר המהלך כנגד מספרם במצב הנוכחי. ההפרש בין הערכים מושם ב-Val1. כמו כן, הפעולה בודקת עבור כל מהלך שכזה את הקומפקטיות של הכדורים הירוקים לפני ואחרי, ההפרש הזה מושם ב-Val2. הפעולה בוחרת במהלך הטוב ביותר לשיפור מצבו של המחשב.

חשוב:

פעולות ההפרש שמוצגות (הפעולות שעוזרות לקבוע האם מצב המחשב לאחר המהלך הנידון משתפר או ניזוק) מחשבות את ההפרש כך:

ערך הפעולה שווה לערך הנבדק לאחר המהלך פחות הערך הנבדק לפני הפעולה. על כן אם ערך הפעולה שלילי, ישנו שיפור בפרמטר הנבדק, אם ערך הפעולה אפס אז אין שיפור אך אין גם נזק ואילו ערך הפעולה חיובי נוצר נזק בפרמטר הנבדק. שימוש במיון רשימה לפי ערכי ההפרש של פרמטרים שונים מאפשרת לבחור באפשרות הטובה ביותר שקיימת. נעדיף כמובן שיתקיים שיפור בפרמטרים השונים אך במידת הצורך נסתפק גם במהלכים ניטרליים ואף במקרים קיצוניים מהלכים שאינם מטיבים עם מצבו של המחשב במשחק.

פירוט ההרחבה או השכלול של המערכת

1) לשחקן ניתנת האפשרות לבחור בכדורים אותם רוצה להזיז ואף לבטל את הבחירה באמצעות לחיצה חוזרת על הכדור. בסיום בחירת הכדורים, הוספתי מקש עמו יכול השחקן לסיים את תהליך בחירת הכדורים ולהמשיך לתהליך בחירת המהלך מתוך המהלכים האפשריים.

```
view_control_buttons(W):-
        turn(1),buttonFlag,(0)
        retract(buttonFlag(_)),
        assert(buttonFlag(1)),
        new(ConfirmStep,bitmap('confirmStepRight.bmp')),
        send(W,display,ConfirmStep,point(580,568)),
        send(ConfirmStep,recogniser,click_gesture(left,'',single,and(message(@prolog,show_move_opt
        ions,ConfirmStep,W)))).

view_control_buttons(W):-
        turn(2),buttonFlag,(0)
        retract(buttonFlag(_)),
        assert(buttonFlag(1)),
        new(ConfirmStep,bitmap('confirmStepLeft.bmp')),
        send(W,display,ConfirmStep,point(0,0)),
        send(ConfirmStep,recogniser,click_gesture(left,'',single,and(message(@prolog,show_move_opt
        ions,ConfirmStep,W)))).
```

הפעולה מתאימה לשני מצבים: בחירת כדורים ע"י שחקן אדום ובחירת כדורים ע"י שחקן ירוק. ההבדל מתבטא בקואורדינטות הגרפיות בה מופיע הלחצן. הפעולה משתמשת במשתנה דינאמי buttonFlag שאינו מאפשר לפעולה להציג את המקש פעם נוספת אם כבר מוצג, כך שלא ייווצרו מספר מקשים שפועלים בו זמנית ולחיצה עליהם גוררת הפעלה מרובה של הפונקציה show_move_options.

2) על מנת להציג את אפשרויות ההזזה במקביל-"תנועה הצידה" של סט כדורים מסוים לנוחיות המשתמש, לאחר בחירת השחקן בסט, נפתחת חלונית ובה מסומנים בצורה מוקטנת הכדורים שנבחרו על ידיו וצבעם כצבע השחקן וכן המקומות אליהם הכדורים יכולים לנוע ב"תנועה הצידה" בצבע אחר.

```
highLight_broadside_options_maat(W,FinalList):-
                length(FinalList,N),
                not(N is 0),
                WinLen is 231*N,
                new(TempWin,window('Choose An Option:',size(WinLen,231))),
                send(TempWin,open),
                retract(tempWin(_)),
                assert(tempWin(TempWin)),
                retract(broadSideList(_)),
                assert(broadSideList(FinalList)),
                highLight_broadside_options(W,TempWin,FinalList,0).
```

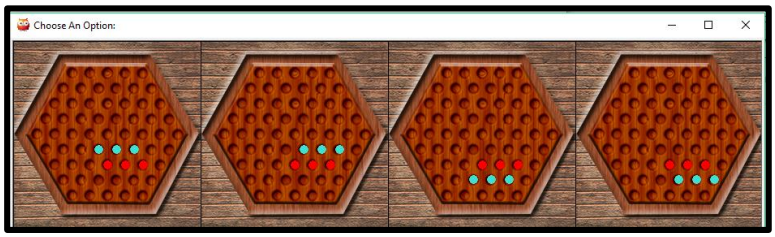
הפעולה בודקת כמה אפשרויות להזזת הסט במקביל קיימות לפי הרשימה המוכנה שהפעולה מקבלת. לפי מספר האפשרויות נקבע גודל חלונית העזר ונוצר אובייקט החלונית. אובייקט החלונית מושם במשתנה דינמי כדי שפעולות אחרות יוכלו לגשת לאובייקט ולסגרו במידת הצורך. כעת נשלחים החלונית, המסך הראשי, רשימת האפשרויות ואינדקס מסוכם 0 לפעולת הצגת האפשרויות באופן גרפי (highLight_broadside_options).

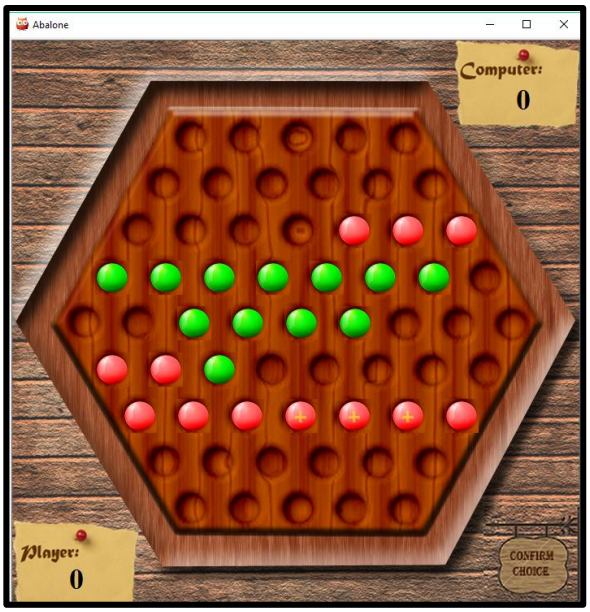
3)

```
highLight_broadside_options(_,_,[],_).

highLight_broadside_options(W,TempWin,[CurrList|OtherLists],N):-
        ((turn(1),Colour = 'red'); (turn(2),Colour = 'green')),
        moveList(ML),
        ValX is 231*N,
        NN is N+1,
        new(Board,bitmap('minBoard.bmp')),
        send(TempWin,display,Board,point(ValX,0)),
        new(Box,box(232,232)),
        send(TempWin,display,Box,point(ValX,0)),
        draw_minimized(TempWin,ML,Colour,N),
        draw_minimized(TempWin,CurrList,'turquoise',N),

send(Box,recogniser,click_gesture(left,'',single,and(message(@prolog,broadSide,W,TempWin,N)))),
        highLight_broadside_options(W,TempWin,OtherLists,NN).
```

הפעולה מציגה בחלונית שנוצרה את האפשרויות השונות. הפעולה עובדת בצורה רקורסיבית וכך עוברת על כל האפשרויות של ההזזה במקביל שרלוונטיות לסט הכדורים הנבחר ומציגה אותן בחלונית. המספר N, בפעם הראשונה מוגדר כ-0 וגדל ב-1 בכל פעם שהתוכנית נכנסת מחדש לפעולה עם אפשרות הזזה שונה. מספר זה מוכפל בקבוע גרפי 231 וקובע את קואורדינטות תמונת הלוח המוקטן, עליו מוצגת אפשרות ההזזה. קואורדינטה זו גדלה כל פעם ולכן תמונות המצב של האפשרויות השונות מוצגות אחת לצד השנייה. התמונה מוצגת בחלונית ותוחמים אותה באובייקט מלבני box וזאת כדי שתהיה גם לחיצה. הפעולה draw_Minimized מציגה על הלוח הקטן את הכדורים שנבחרו ואת מקומם האפשרי החדש (לאחר ההזזה במקביל). לחיצה על האובייקט התחום הכולל בתוכו את תמונת הרקע המוקטן והכדורים תזמין את פעולת broadSide שמבצעת את הזזת הכדורים למיקומם החדש.

כך למשל:


Choose An Option:


Abalone
Computer:
0
Player:
0
CONFIRM
CHOICE

הממשק למשתמש

בעבודה נעשה שימוש בגרפיקה באמצעות Xpce, כלי ליצירת ממשק גרפי בתוכניות פרולוג המאפשר תכנות מונחה עצמים ובאמצעותו ניתן ליצור חלונות, תפריטים, ותיבות דיאלוג.

הממשק במשחק מצריך אך ורק שימוש בעכבר.

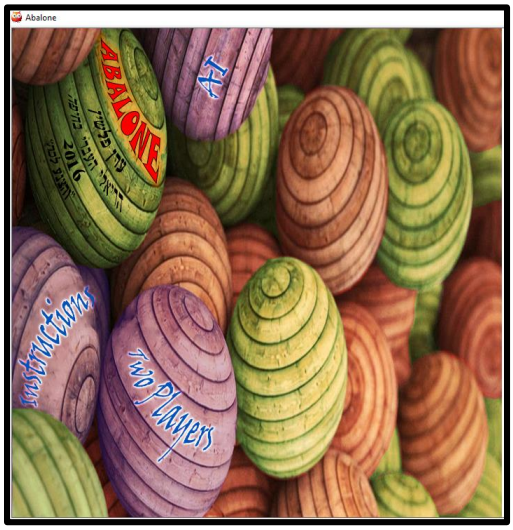
להלן מסכים גרפיים שונים בהם ניתן להיתקל במהלך המשחק:

1) מסך הפתיחה:

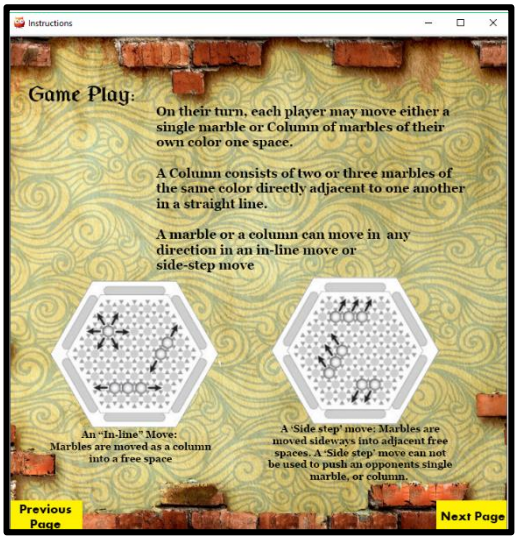
-AI –משחק שחקן מול מחשב.

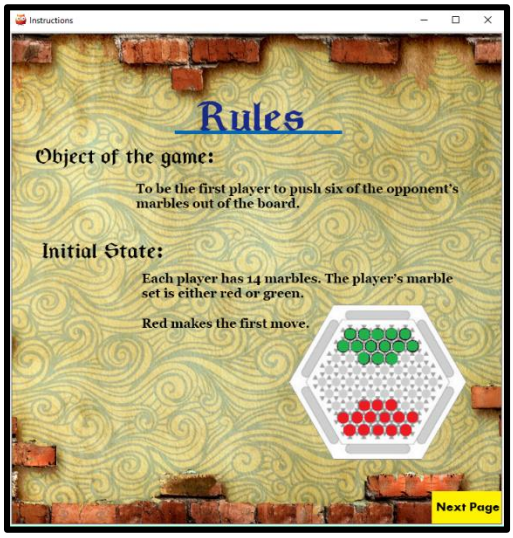
-Two Players-משחק שחקן מול שחקן.

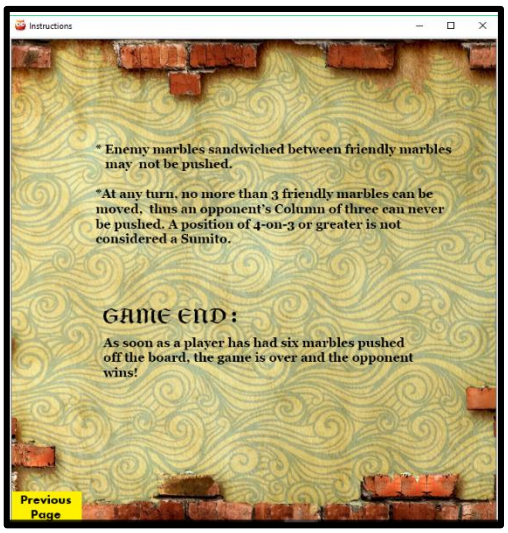
-instructions- הוראות המשחק.



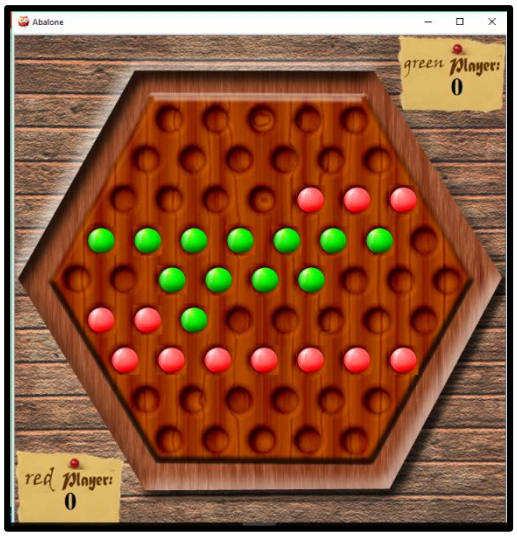
2) מסכי הוראות המשחק:



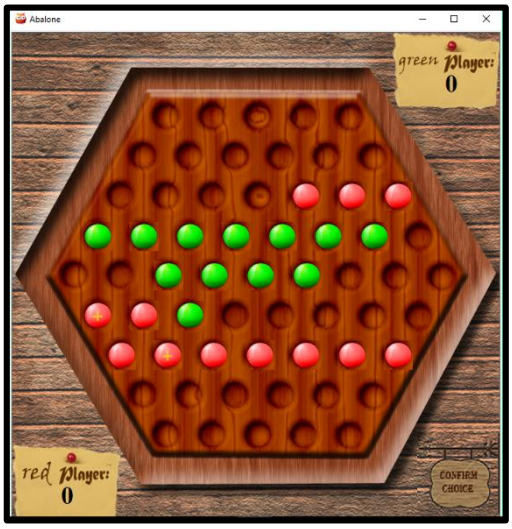




בכל מסך הוראות ניתן להמשיך לדף הבא או לחזור לדף הקודם ולצאת מההסבר באמצעות לחצן סגירת החלון.



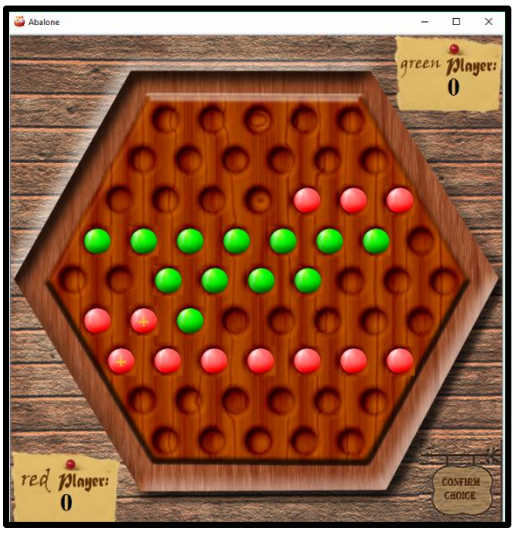
3) מצב מייצג כלשהו במשחק "שחקן מול שחקן".
זכות בחירת הכדורים היא של הקבוצה האדומה, כלומר כעת תור האדומים.
האפשרויות במצב זה:
-בחירת כדורים וביטול בחירתם.



4) מצב מייצג של בחירה לא חוקית וההודעה שמתלווה אליה לאחר לחיצה על "confirm choice":

האפשרות היא:
-לצאת מתיבת דיאלוג זו ע"י לחיצה על מקש QUIT ובחירה מחודשת.

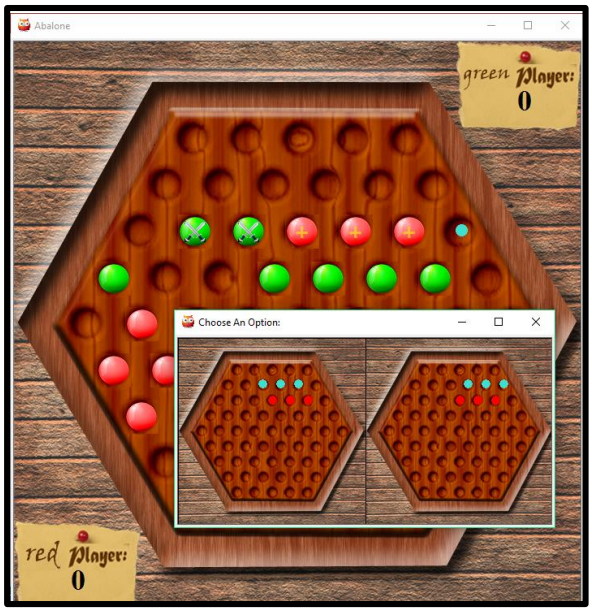
5) מצב מייצג של בחירה חוקית אך כזו המניבה מצב תקיעה וההודעה שמתלווה אליה לאחר לחיצה על "confirm choice":



האפשרות היא:

-לצאת מתיבת דיאלוג זו ע"י לחיצה על מקש QUIT ובחירה מחודשת.

6) מצב מייצג של כל 3 האפשרויות למהלך והביטוי הגרפי שלהן:



אפשרויות הלחיצה הן:

-נקודה כחולה בודדה-הזזת שלוש הכדורים לימין.

-אחת החרבות-דחיפת הכדורים הירוקים לשמאל.

-לחיצה על אחד המצבים בחלונית-תנועה במקביל הצידה כמתואר במצב הנבחר.

בחירת כל אחת משלוש האפשרויות תסגור גם את חלונית העזר הגרפית.

7) מסכי ניצחון:

אפשרויות לחיצה:

-סגירת המשחק.

מדריך להתקנת והפעלת המשחק

על מנת להפעיל את המשחק יש לפעול לפי ההוראות הבאות:

1) יש להעתיק את תיקיית התמונות המצורפות לתיקייה:
תיקיית ההתקנה של פרולוג\xpce\bitmaps

2) יש לפתוח את תוכנת פרולוג ולכתוב את השאילתא:

?-emacs.

3) בחלון שנפתח יש ללחוץ על file ו-open , לאחר מכן על browse ולבחור:

abaloneFinal.pl

4) יש לקמפל את הקובץ: ללחוץ על compile ועל compile buffer.

5) יש לכתוב את השאילתא:

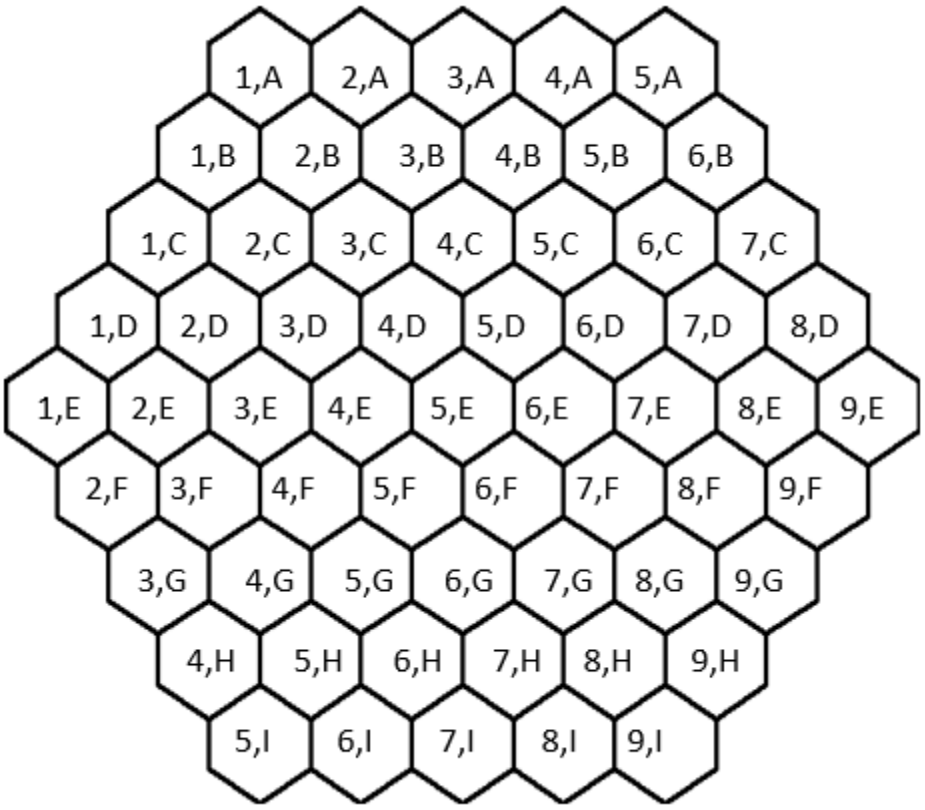
?-intro.

בעיות פתוחות והצעות לשיפורים

-יש להוסיף פעולות התקפיות עמן יכול להגיב המחשב.
-יש להרחיב את פעולת ההגנה שמסכלת את דחיפת כדורי המחשב מחוץ ללוח ע"י הריסת סט הכדורים התוקף ולא רק ע"י הזזת הכדור הנדחף.
-למנוע מהמחשב להגן על כדור מאוים בכך שיזיז אותו לתא שבו חשוף לדחיפה מחוץ ללוח.

נספחים

תרשים 1: "ייצוג האינדקסים של כל תא בלוח"

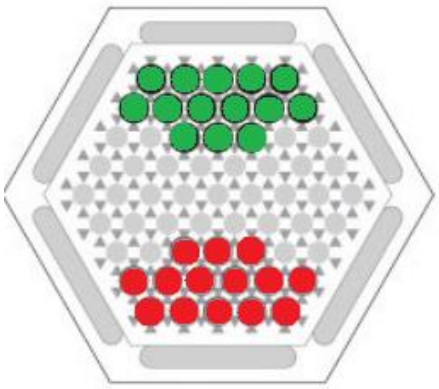


סעיף 1: "הצגת המשחק, החוקים והמטרה":

חוקים

מספר משתתפים-משחק האבלון מיועד לשני משתתפים.

מטרת המשחק- להיות הראשון להדוף 6 כלי שחקן יריב מחוץ לגבולות הלוח.



פתיחת המשחק-לכל שחקן סט של 14 כדורים בעלי צבע זהה (אדום או ירוק) הממוקמים במצבם ההתחלתי כפי שמתואר בתרשים. השחקן האדום הוא זה שתמיד פותח את המשחק. המשחק מתנהל בתורות.

מהלך המשחק- בכל תור ניתן לבצע את אחד המהלכים הבאים:

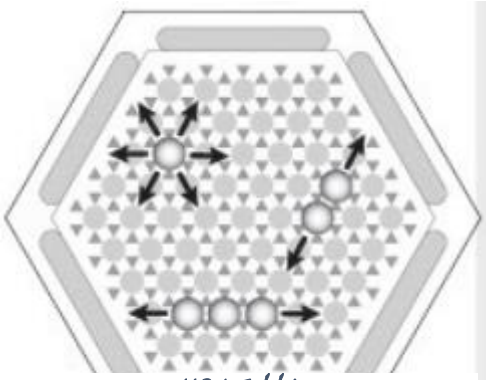
1. תנועה
2. "סומיטו"-הדיפת יריב.

תנועה- ישנן שלוש אפשרויות לתנועה:

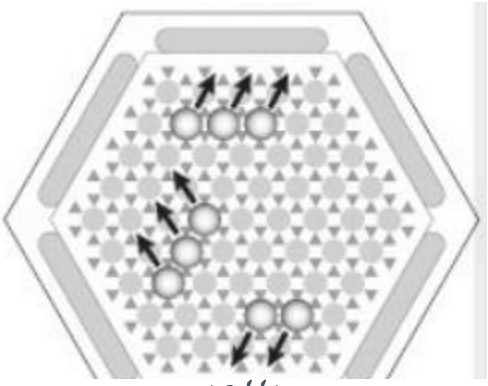
1. הזזת כדור **אחד בלבד** בכל אחד משש הכיוונים האפשריים למקום פנוי הצמוד לו.
2. ניתן להזיז בבת אחת, כקבוצה ולאותו כיוון **שניים** או **שלושה** כדורים הצמודים זה לזה ומהווים שורה (אופקית או אלכסונית) כך שבסיום התנועה כל הכדורים ימצאו שוב בשורה אחת. כלל זה ניתן ליישם בשני אופנים:

2.א. תנועה "בתוך השורה"-הכדורים זזים יחד בקו ישר, כדור אחר כדור.

2.ב. תנועה "הצידה"- תנועה לצד השורה, לשורה המקבילה. הכדורים זזים כמקשה אחת הצידה למקום פנוי חדש המקביל לשורה בה היו קודם לכן. במצב החדש הכדורים עדיין מהווים שורה.



כלל 1 ו-2א.



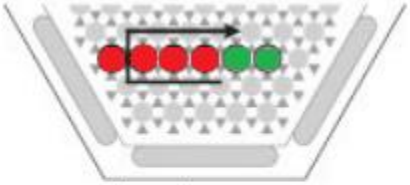
כלל 2ב

"סומיטו"-הדיפת יריב- ניתן להדוף (להזיז את כדורי היריב הצמודים לתא אחר בתחומי הלוח או אל מחוץ לגבולותיו) רק כאשר לשחקן ההודף יתרון מספרי על כדורי היריב. כאשר ישנו שוויון כוחות-לשני השחקנים אותו מספר כדורים לא ניתן להדוף.

"סומיטו" אפשרי בתנועה "בתוך שורה" בלבד ובתנאי שקיים מקום פנוי, אליו ניתן להדוף את כדורי היריב.

קיימים רק שלושה מצבי "סומיטו" אפשריים:

גם במהלך "סומיטו" ניתן להדוף באמצעות **שניים** או **שלושה** כדורים. במידה ובשורה מונחים יותר משלושה כדורים, נשתמש אך ורק בשניים או שלושה למטרת הדיפה:



במצב זה למשל:
3 כדורים הודפים כדור אחד.
הכדור האדום הרביעי נותר במקומו.

במידה וכדור נהדף מחוץ לתחומי הלוח כתוצאה מ-"סומיטו"-זוהי הדיפה סופית והכדור הנהדף יוצא מהמשחק.

ניצחון-השחקן הראשון שמצליח להדוף 6 מכדורי יריבו מחוץ לתחומי הלוח מנצח במשחק!

מילון מושגים מקצועי

מושגי משחק כלליים

כדור- כל אחד מחיילי המשחק. כל אחד מהשחקנים מקבל 14 כדורים בצבע אדום או ירוק.

קבוצה דו-כדורית- 2 כדורים צמודים בעלי צבע זהה.

קבוצה תלת-כדורית- 3 כדורים צמודים בעלי צבע זהה.

תנועה הצידה-תנועה שבה 2 או 3 כדורים נהדפים בו זמנית הצידה לשורה המקבילה לשורה בו נמצאו קודם לכן.

תנועה בתוך השורה- תנועה שבה 2 או 3 כדורים נעים כטור בו זמנית לאורך קו ישר.

מצב שוויון- מצב יציב בו שתי קבוצות כדורים יריבות בעלות אותו כוח נמצאות אחת מול השנייה: 3 כדורים אדומים מול 3 כדורים ירוקים, 2 כדורים אדומים מול 2 כדורים ירוקים וכו'.

סומיטו-מצב של יתרון מספרי על כדורי היריב.

מושגי ייצוג בקוד

Horizontal (horiz.)-אופקי. כלומר סט תאים או כדורים אופקיים. לאור הגדרת ייצוג האינדקסים של כל תא בלוח בעבודה זו, מדובר בסט תאים או כדורים צמודים בעלי אינדקס Y זהה ואינדקסי X עוקבים. לדוגמא- [(1,A),(2,A),(3,A)].

Num Diagonal (numDiag)-סט תאים או כדורים צמודים הממוקמים באלכסון " / " (מימין לשמאל). לאור הגדרת ייצוג האינדקסים, מדובר בסט תאים או כדורים צמודים בעלי אינדקס X זהה ואינדקסי Y עוקבים. לדוגמא- [(1,A),(1,B),(1,C)].

Mix Diagonal (mixDiag)- סט תאים או כדורים צמודים הממוקמים באלכסון " \ " (משמאל לימין). לאור הגדרת ייצוג האינדקסים, מדובר בסט תאים או כדורים צמודים בעלי אינדקסי X עוקבים ואינדקסי Y עוקבים. לדוגמא- [(1,A),(2,B),(3,C)].

Inline- המונח "תנועה בתוך השורה" באנגלית.

Broadside-המונח "תנועה הצידה" באנגלית.

Push-הדיפת כדור יריב (בתחומי הלוח או מחוצה לו).

Ring-בייצוג חלקתי את הלוח לטבעות כך שהתא המרכזי בלוח נחשב טבעת מספר 0, 6 התאים הצמודים לו מוגדרים כטבעת 1, התאים המקיפים את טבעת 1 מוגדרים כטבעת 2, וכן הלאה.

Manhatten-distance (מרחק מנהטן)- מרחק זה מוגדר באופן שונה ממרחק בין שתי נקודות בגיאומטריה האוקלידית. מרחק מנהטן הוא המרחק הפסיעות הקצר ביותר מתא על הלוח לתא אחר על הלוח. צורת המסלול אינה חשובה. זהו לא "מרחק אווירי"!

Threatened piece (כלי מאוים)- כדור אשר מוקף ביותר כדורי יריב מאשר כדורים מקבוצתו נחשב ככדור מאוים.

ביבליוגרפיה

1) רקע כללי למשחק:
https://en.wikipedia.org/wiki/Abalone_(board_game)

2) כללי המשחק:
http://www.foxmind.com/media/29695/abalone_en.pdf

3) רקע תיאורטי אודות שיטות יישום הבינה המלאכותית במשחק "אבלון":

http://geneura.ugr.es/cig2012/papers/paper51.pdf

http://www.cs.cornell.edu/~hn57/pdf/AbaloneFinalReport.pdf

4) טקטיקות ואסטרטגיות:
http://entertainment.howstuffworks.com/leisure/brain-games/abalone2.htm

קוד התכנית

```
%Eran Flashin 2016, Abalone, Reali Hebrew School

%dynamic variables
:-dynamic blackList/1. %Black played balls list.
:-dynamic whiteList/1. %White played balls list.
:-dynamic turn/1. %Indicates whose the turn is: Black(1) or White(2).
:-dynamic scoring/2. %Indicates the scoring of the black and the white players.
:-dynamic graphicScoring/2.%keeps the scoring text.
:-dynamic moveList/1.%keeps list of chosen balls to move.
:-dynamic buttonFlag/1.%indicates whether the "ConfirmStep" button is on(1) or off(0).
:-dynamic tempCell/2.%Variable used as a temporal cell
:-dynamic arrangementType/1.%keeps the ball-set arrangement type
:-dynamic tempWin/1.%Graphic small window
:-dynamic broadSideList/1.
:-dynamic pushList/1.
:-dynamic aiOrTwoPlayers/1.%indicates whether the game is player vs. player or player vs. comp.


%assisting functions and facts:

   %finds the following or the previous char.
next_char(Char,NChar):-
        char_code(Char,CharCode),
        NCharCode is CharCode+1,
        char_code(NChar,NCharCode).

previous_char(Char,NChar):-
        char_code(NChar,NCharCode),
        CharCode is NCharCode-1,
        char_code(Char,CharCode).

   %information for each line
(line_Index,initXVal,FinalXVal,graphical_CoordX_of_init_cell,graphical_CoordY_of_init_cell)
line_info('A',1,5,199,100).
line_info('B',1,6,165,154).
```

```
line_info('C',1,7,136,213).
line_info('D',1,8,102,271).
line_info('E',1,9,71,327).
line_info('F',2,9,102,384).
line_info('G',3,9,136,441).
line_info('H',4,9,168,496).
line_info('I',5,9,199,554).

    %converts board cell indexes to graphical coordinates.
findGraphicalCoords(X,Y,CoordX,CoordY):-
        line_info(Y,X0,_,CoordX0,CoordY),
        DeltaX is X-X0,
        CoordX is CoordX0+DeltaX*66.

    %prints error messages. 1) choice illegal 2)the choice leads to a dead end.

open_error_dialog_choice:-
        new(D,dialog('')),
        send(D,background(bitmap('dialogue.bmp'))),
        send(D, append, text('Your choice is illegal!',center,font(times,bold,30))),
        send(D, append, button(quit, message(D, destroy))),
        send(D,open).

open_error_dialog_stuck:-
        new(D,dialog('')),
        send(D,background(bitmap('dialogue.bmp'))),
        send(D, append, text('You are stuck!',center,font(times,bold,30))),
        send(D, append, button(quit, message(D, destroy))),
        send(D,open).

%given List and size N, the function finds the sublist 1-->N.

subList(List,Len,[]):-
        length(List,LLen),
        LLen<Len.
```

```
subList(_,0,[]).
subList(List,Len,[Elem|B]):-
        nth1(Len,List,Elem),
        select(Elem,List,NList),
        NLen is Len-1,
        subList(NList,NLen,B).

    %retrieves appended list of black and white balls.
appended_black_and_white(AL):-
        blackList(BL),
        whiteList(WL),
        append(BL,WL,AL).

%given two cells (X,Y), it finds the one having the higher value.
findHigherValCell(Cell1,Cell2,Max):-
        append([Cell1],[Cell2],List),
        sort(List,SList),
        last(SList,Max).

%list of all legal Y indexes
legal_Y_Indexes(['A','B','C','D','E','F','G','H','I']).

% converts List of cells: [(X1,Y1),(X2,Y2),...] into
% [(X1,Y1,_),(X2,Y2,_)....]
compatibilityConvertion(L,NL):-
        findall((X,Y,_),member((X,Y),L),NL).

%checks whether a cell is valid or not (within the board borders).
validCell(ValX,ValY):-
        legal_Y_Indexes(LY),
        member(ValY,LY),
        line_info(ValY,MinX,MaxX,_,_),
        ValX>=MinX,ValX=<MaxX.

%checks whether a cell is both valid and vacant.
```

```
availCell(ValX,ValY,AL):-
        legal_Y_Indexes(LY),
        member(ValY,LY),
        line_info(ValY,MinX,MaxX,_,_),
        ValX>=MinX,ValX=<MaxX,
        not((member((X,Y),AL),(ValX is X, ValY == Y))).

% given a cell (X,Y), the function retrieves all vacant cells around
% (X,Y)
list_of_free_cells_around_ball(X,Y,FreeCellList):-
        appended_black_and_white(AL),
        retractall(tempCell(_,_)),

        PrevXVal is X-1,
        NextXVal is X+1,
        previous_char(PrevYVal,Y),
        next_char(Y,NextYVal),

        assert(tempCell(NextXVal,Y)),
        assert(tempCell(NextXVal,NextYVal)),
        assert(tempCell(X,NextYVal)),
        assert(tempCell(PrevXVal,Y)),
        assert(tempCell(PrevXVal,PrevYVal)),
        assert(tempCell(X,PrevYVal)),

        findall((ValX,ValY),(tempCell(ValX,ValY),availCell(ValX,ValY,AL)),FreeCellList).

%% given a set of cells, the function retrieves all vacant cells
%% around the set.

list_of_free_cells_around_set([],[]).

list_of_free_cells_around_set([(X,Y,_)|B],FreeCellList):-
        list_of_free_cells_around_ball(X,Y,FreeCellListAroundSingleBall),
        list_of_free_cells_around_set(B,C),
```

```
        union(FreeCellListAroundSingleBall,C,FreeCellList).



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


%the root function: creates the game menu
intro:-
        new(P,window('Abalone',size(700,700))),
        new(Background,bitmap('intro.bmp')),
        send(P,display,Background,point(0,0)),

        new(TwoPlayers,bitmap('TwoPlayers.bmp')),
        new(Instructions,bitmap('Instructions.bmp')),
        new(AI,bitmap('Ai.bmp')),

        send(P,display,TwoPlayers,point(143,430)),
        send(P,display,Instructions,point(1,342)),
        send(P,display,AI,point(217,0)),

        send(AI,recogniser,click_gesture(left,'',single,and(message(@prolog,startAi,P)))),
        send(TwoPlayers,recogniser,click_gesture(left,'',single,and(message(@prolog,startTwoPlayers,P)))),
        send(Instructions,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage)))),

      send(P,open).

% Each function InstructionPage_,1,2,3,4 displays an instruction and
% allows to browse pages.
instructionsPage:-
                new(Inst,window('Instructions',size(700,700))),
                send(Inst,open),
                instructionsPage1(Inst).


instructionsPage1(Inst):-
        send(Inst,clear),
```

```
        new(Page,bitmap('InstructionsPage1.bmp')),
        new(Next,bitmap('nextPage.bmp')),

        send(Inst,display,Page,point(0,0)),
        send(Inst,display,Next,point(599,652)),

        send(Next,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage2,Ins
t)))).

instructionsPage2(Inst):-
        send(Inst,clear),
        new(Page,bitmap('InstructionsPage2.bmp')),
        new(Next,bitmap('nextPage.bmp')),
        new(Previous,bitmap('previousPage.bmp')),

        send(Inst,display,Page,point(0,0)),
        send(Inst,display,Next,point(599,652)),
        send(Inst,display,Previous,point(0,652)),

        send(Next,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage3,Ins
t)))),
        send(Previous,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage1,
Inst)))).


instructionsPage3(Inst):-
        send(Inst,clear),
        new(Page,bitmap('InstructionsPage3.bmp')),
        new(Next,bitmap('nextPage.bmp')),
        new(Previous,bitmap('previousPage.bmp')),

        send(Inst,display,Page,point(0,0)),
        send(Inst,display,Next,point(599,652)),
        send(Inst,display,Previous,point(0,652)),

        send(Next,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage4,Ins
t)))),
```

```
        send(Previous,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage2,
Inst)))).

instructionsPage4(Inst):-
        send(Inst,clear),
        new(Page,bitmap('InstructionsPage4.bmp')),
        new(Previous,bitmap('PreviousPage.bmp')),

        send(Inst,display,Page,point(0,0)),
        send(Inst,display,Previous,point(0,652)),

        send(Previous,recogniser,click_gesture(left,'',single,and(message(@prolog,instructionsPage3,
Inst)))).


% the root function for Player vs Comp game. it initializes and displays
% the basic graphic platform.
startAi(P):-
        send(timer(0.2),delay),
        free(P),
        init_dyn_par,
        init_board(W),
        init_balls,
        drawWithRecBlack(W),
        drawNoRecWhite(W),
        print_scoring(W).

% the root function for Player vs Player game. it initializes and
% displays the basic graphic platform.
startTwoPlayers(P):-
        send(timer(0.2),delay),
        free(P),
        init_dyn_par,

        retract(aiOrTwoPlayers(_)),
        assert(aiOrTwoPlayers(1)),
```

```
        init_board(W),
        init_balls,
        drawWithRecBlack(W),
        drawNoRecWhite(W),
        print_scoring(W).

%initialize dynamic parameters.
init_dyn_par:-
        retractall(turn(_)),
        assert(turn(1)),
        retractall(moveList(_)),
        assert(moveList([])),
        retractall(buttonFlag(_)),
        assert(buttonFlag(0)),
        retractall(scoring(_,_)),
        assert(scoring(0,0)),
        retractall(graphicScoring(_,_)),
        new(ScoreB,text(0,center,font(times,bold,30))),
        new(ScoreW,text(0,center,font(times,bold,30))),
        assert(graphicScoring(ScoreB,ScoreW)),
        retractall(whiteList(_)),
        retractall(blackList(_)),
        retractall(arrangementType(_)),
        assert(arrangementType(0)),
        retractall(tempCell(_,_)),
        assert(tempCell(0,0)),
        retractall(broadSideList(_)),
        assert(broadSideList([])),
        retractall(pushList(_)),
        assert(pushList([])),
        retractall(tempWin(_)),
        assert(tempWin(0)),
        retractall(aiOrTwoPlayers(_)),
        assert(aiOrTwoPlayers(0)).
```

```
%initialize graphic board.
init_board(W):-
        new(W,window('Abalone',size(700,700))),

        ((aiOrTwoPlayers(0),new(Background,bitmap('boardAi.bmp')));(aiOrTwoPlayers(1),new(Bac
kground,bitmap('boardTwoPlayers.bmp')))),
        send(W,display,Background,point(0,0)),

        send(W,open).

%Displays the graphic board.
view_board(W):-
        ((aiOrTwoPlayers(0),new(Background,bitmap('boardAi.bmp')));(aiOrTwoPlayers(1),new(Bac
kground,bitmap('boardTwoPlayers.bmp')))),
        send(W,display,Background,point(0,0)).

% set both scoring value and scoring text objects according to the sent
% values.
set_scoring(Black,White):-
        retract(scoring(_,_)),
        retract(graphicScoring(_,_)),

        assert(scoring(Black,White)),
        new(ScoreB,text(Black,center,font(times,bold,30))),
        new(ScoreW,text(White,center,font(times,bold,30))),
        assert(graphicScoring(ScoreB,ScoreW)).

%prints the current scoring.
print_scoring(W):-
        graphicScoring(ScoreB,ScoreW),
        send(W,display,ScoreB,point(70,640)),
        send(W,display,ScoreW,point(620,50)).

%deletes the previous scoring.
delete_scoring:-
```

```
        graphicScoring(PrevScoreB,PrevScoreW),
        free(PrevScoreB),
        free(PrevScoreW).

%displayes the current scoring.
view_scoring(W):-
        scoring(Black,White),
        delete_scoring,
        set_scoring(Black,White),
        print_scoring(W).

% Initializes both ball Lists-Black and White with their initial
% coordinates.
init_balls:-
        createBallList(5,9,'I',BlackList1),
        createBallList(4,9,'H',BlackList2),
        createBallList(5,7,'G',BlackList3),

        append(BlackList1,BlackList2,BlackAppendedList),
        append(BlackAppendedList,BlackList3,BlackList),

        createBallList(1,5,'A',WhiteList1),
        createBallList(1,6,'B',WhiteList2),
        createBallList(3,5,'C',WhiteList3),

        append(WhiteList1,WhiteList2,WhiteAppendedList),
        append(WhiteAppendedList,WhiteList3,WhiteList),


        assert(whiteList(WhiteList)),

        assert(blackList(BlackList)).


% given X start index, X end index, and Y index, the function builds a
```

```
% list: [(XStart,Y),...,(XEnd,Y)]
createBallList(Start,End,_,[]):-
        Start is End+1.

createBallList(Start,End,Index,[(Start,Index)|B]):-
        NextStart is Start+1,
        createBallList(NextStart,End,Index,B).


% Displays either black balls or white ones, creating an
% unclickable bitmap or an interactive button.

drawNoRecWhite(W):-
        whiteList(WL),
        drawNoRec(W,WL,white).

drawWithRecWhite(W):-
        whiteList(WL),
        drawWithRec(W,WL,white).

drawNoRecBlack(W):-
        blackList(BL),
        drawNoRec(W,BL,black).

drawWithRecBlack(W):-
        blackList(BL),
        drawWithRec(W,BL,black).


% Both functions display the white or black bitmaps clickable or
% uncklicable.

drawNoRec(_,[],_).
drawNoRec(W,[(X,Y)|NL],Colour):-
```

```
        ((Colour=='white',new(Pic,bitmap('greenball.bmp')));(Colour=='black',new(Pic,bitmap('red
Ball.bmp')))),
        findGraphicalCoords(X,Y,CoordX,CoordY),
        send(W,display,Pic,point(CoordX,CoordY)),
        drawNoRec(W,NL,Colour).

drawWithRec(_,[],_).
drawWithRec(W,[(X,Y)|NL],Colour):-
        ((Colour=='white',new(Pic,bitmap('greenball.bmp')));(Colour=='black',new(Pic,bitmap('red
Ball.bmp')))),
        findGraphicalCoords(X,Y,CoordX,CoordY),
        send(W,display,Pic,point(CoordX,CoordY)),
        send(Pic,recogniser,click_gesture(left,'',single,and(message(@prolog,select_balls,W,X,Y)))),
        drawWithRec(W,NL,Colour).


% the function handles with the player's balls set choice. It displays a
% choice confirmation button as well. First click means a choice, Second
% click on the same tool means cancellation.

select_balls(W,_,_):-
        view_control_buttons(W),fail.

select_balls(W,X,Y):-
        turn(1),moveList(CurrList),

        ((member((X,Y,HlBall),CurrList),
      select((X,Y,HlBall),CurrList,NNList),
         free(HlBall),
         retract(moveList(_)),
         assert(moveList(NNList)));

        (new(HLBall,bitmap('hlRedBall.bmp')),
         findGraphicalCoords(X,Y,CoordX,CoordY),
         send(W,display,HLBall,point(CoordX,CoordY)),
         append([(X,Y,HLBall)],CurrList,NList)),
```

```
            retract(moveList(_)),
            assert(moveList(NList))).

select_balls(W,X,Y):-
          turn(2),moveList(CurrList),

          ((member((X,Y,HlBall),CurrList),
        select((X,Y,HlBall),CurrList,NNList),
            free(HlBall),
            retract(moveList(_)),
            assert(moveList(NNList)));

          (new(HLBall,bitmap('hlGreenBall.bmp')),
            findGraphicalCoords(X,Y,CoordX,CoordY),
            send(W,display,HLBall,point(CoordX,CoordY)),
            append([(X,Y,HLBall)],CurrList,NList)),
            retract(moveList(_)),
            assert(moveList(NList))).

%displays the confirmation button.
view_control_buttons(W):-
          turn(1),buttonFlag(0),
          retract(buttonFlag(_)),
          assert(buttonFlag(1)),
          new(ConfirmStep,bitmap('confirmStepRight.bmp')),
          send(W,display,ConfirmStep,point(580,568)),
          send(ConfirmStep,recogniser,click_gesture(left,'',single,and(message(@prolog,show_move_
options,ConfirmStep,W)))).

view_control_buttons(W):-
          turn(2),buttonFlag(0),
          retract(buttonFlag(_)),
          assert(buttonFlag(1)),
          new(ConfirmStep,bitmap('confirmStepLeft.bmp')),
          send(W,display,ConfirmStep,point(0,0)),
```

```
        send(ConfirmStep,recogniser,click_gesture(left,'',single,and(message(@prolog,show_move_
options,ConfirmStep,W)))).

% After the player has chosen the tools he wants to move, the function
% shows all possible valid moves. It cancells the interactive buttons as
% well.
show_move_options(Button,W):-
        free(Button),
        retract(buttonFlag(_)),
        assert(buttonFlag(0)),

        check_legality,

        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawNoRecWhite(W),
        drawNoRecBlack(W),
        restart_hlBalls_maat(W),

        mark_move_and_push_options(W).

% In case of check_legality>false. Shows error message and provides the
% player another choice option.
show_move_options(_,_):-
        open_error_dialog_choice,

        moveList(ML),
        clear_hlBalls(ML),
        reset_moveList.

reset_moveList:-
        retract(moveList(_)),
        assert(moveList([])).
```

```
%clears graphic highlighted balls that have been chosen before.
clear_hlBalls([]).

clear_hlBalls([(_,_,CurrBall)|B]):-
        free(CurrBall),
        clear_hlBalls(B).

%displays the highlighted chosn balls.
restart_hlBalls_maat(W):-
        moveList(ML),
        restart_hlBalls(ML,NML,W),
        retract(moveList(_)),
        assert(moveList(NML)).

restart_hlBalls([],[],_).

restart_hlBalls([(X,Y,_)|B],[(X,Y,HLBall)|C],W):-
        turn(1),
        new(HLBall,bitmap('hlRedBall.bmp')),
        findGraphicalCoords(X,Y,CoordX,CoordY),
        send(W,display,HLBall,point(CoordX,CoordY)),
        restart_hlBalls(B,C,W).

restart_hlBalls([(X,Y,_)|B],[(X,Y,HLBall)|C],W):-
        turn(2),
        new(HLBall,bitmap('hlGreenBall.bmp')),
        findGraphicalCoords(X,Y,CoordX,CoordY),
        send(W,display,HLBall,point(CoordX,CoordY)),
        restart_hlBalls(B,C,W).

%checks whether the chosen balls set is legal.
check_legality:-
        moveList(MList),
        sort(MList,SMList),
        retract(moveList(_)),
```

```
        assert(moveList(SMList)),


        length(MList,Len),
        Len>0,Len=<3,

        ((horiz_leg(SMList),retract(arrangementType(_)),assert(arrangementType(1)));
        (numDiag_leg(SMList),retract(arrangementType(_)),assert(arrangementType(2)));
        (mixDiag_leg(SMList)),retract(arrangementType(_)),assert(arrangementType(3))).

%checks whether the set is arranged horizontally.

horiz_leg([_|[]]).

horiz_leg([(X,Y,_)|B]):-
        nth1(1,B,(NX,Y,_)),
        NX is X+1,
        horiz_leg(B).


%checks whether the set is arranged num axis Diagonally.

numDiag_leg([_|[]]).

numDiag_leg([(X,Y,_)|B]):-
        nth1(1,B,(X,NY,_)),
        next_char(Y,NY),
        numDiag_leg(B).

%checks whether the set is arranged mix axis Diagonally.

mixDiag_leg([_|[]]).

mixDiag_leg([(X,Y,_)|B]):-
        nth1(1,B,(NX,NY,_)),
```

```
        NX is X+1,
        next_char(Y,NY),
        mixDiag_leg(B).

%$$$$$$$$$$$$$$$$$$$$$$$$$$$$

%marking move options for a single ball: all adjecant free cells.
% The player is stuck when none moves are available.
mark_move_and_push_options(W):-
        moveList([(X,Y,_)]),
        list_of_free_cells_around_ball(X,Y,FCL),

        ((length(FCL,0),stuck(W));
        (highLightAvailCell(W,FCL))).

% marking move and push options for set of balls. prints the available
% moves' status. The player is stuck when none moves are available.

mark_move_and_push_options(W):-
        scan_inline(W,SBroadsideMove,InLineListLen),
        scan_broadside(W,SBroadsideMove,CheckValBM),
        scan_push(W,CheckValPush),
        writeln('Results'+InLineListLen+CheckValBM+CheckValPush),
        (((InLineListLen \= 0);(CheckValBM \= 0);(CheckValPush \= 0));(stuck(W))).

% Displayes interactively (highlighted buttons) all inLine move options
% and prepares in advance a list of adjecant free cells for BroadSide
% move.

scan_inline(W,SBroadsideMove,InLineListLen):-
        moveList(ML),
        list_of_free_cells_around_set(ML,FCL),

        ((arrangementType(1),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(C
ML,SCML),horiz_leg(SCML)),InlineMove));
```

```
        (arrangementType(2),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(CM
L,SCML),numDiag_leg(SCML)),InlineMove));

        (arrangementType(3),findall((X,Y),(member((X,Y),FCL),append([(X,Y,_)],ML,CML),sort(CM
L,SCML),mixDiag_leg(SCML)),InlineMove))),

        highLightAvailCell(W,InlineMove),

        subtract(FCL,InlineMove,BroadsideMove),

        sort(BroadsideMove,SBroadsideMove),

        length(InlineMove,InLineListLen).


% Displays interactively all push options when the pushing set is

% moveList. The push options are reached gradually-by failing unsuitable

% options-too long and sandwiched illegal sets.


scan_push(W,CheckVal):-

        turn(1),

        moveList(ML),

        whiteList(WL),


        ((arrangementType(1),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CM
L,SCML),horiz_leg(SCML)),PushOpt));

        (arrangementType(2),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CM
L,SCML),numDiag_leg(SCML)),PushOpt));

        (arrangementType(3),findall((X,Y),(member((X,Y),WL),append([(X,Y,_)],ML,CML),sort(CM
L,SCML),mixDiag_leg(SCML)),PushOpt))),


        ((length(PushOpt,0),CheckVal is 0);

        (build_push_list_maat(PushOpt,PushList),

        fail_Push_List_combined_colour(PushList,NPushList),

        fail_Push_List_too_long(ML,NPushList,NNPushList),

        ((length(NNPushList,0),CheckVal is 0);

        (retract(pushList(_)),

         assert(pushList(NNPushList)),

         view_Push_options(W,NNPushList,1),CheckVal is 1)))).


scan_push(W,CheckVal):-

        turn(2),

        moveList(ML),
```

```
        blackList(BL),

        ((arrangementType(1),findall((X,Y),(member((X,Y),BL),append([(X,Y,_)],ML,CML),sort(CML
,SCML),horiz_leg(SCML)),PushOpt));
        (arrangementType(2),findall((X,Y),(member((X,Y),BL),append([(X,Y,_)],ML,CML),sort(CML,
SCML),numDiag_leg(SCML)),PushOpt));
        (arrangementType(3),findall((X,Y),(member((X,Y),BL),append([(X,Y,_)],ML,CML),sort(CML,
SCML),mixDiag_leg(SCML)),PushOpt))),

        ((length(PushOpt,0),CheckVal is 0);
        (build_push_list_maat(PushOpt,PushList),
        fail_Push_List_combined_colour(PushList,NPushList),
        fail_Push_List_too_long(ML,NPushList,NNPushList),
        ((length(NNPushList,0),CheckVal is 0);
        (retract(pushList(_)),
         assert(pushList(NNPushList)),
         view_Push_options(W,NNPushList,1),CheckVal is 1)))).

% creates a list of several "push_list" for each available suspected
% push option.
%
build_push_list_maat([],[]).
build_push_list_maat([Opt|OtherOpts],[CurrList|B]):-
        appended_black_and_white(AL),
        moveList(ML),
        findall((X,Y),member((X,Y,_),ML),NML),
        subtract(AL,NML,SubList),

        build_push_list(SubList,Opt,PushList),
        append(PushList,[Opt],CurrList),
        build_push_list_maat(OtherOpts,B).


build_push_list(SubList,(X,Y),[(NX,NY)|B]):-
        ((arrangementType(1),member((NX,NY),SubList),append([(NX,NY,_)],[(X,Y,_)],Check),sort
(Check,SCheck),horiz_leg(SCheck));
```

```
        (arrangementType(2),member((NX,NY),SubList),append([(NX,NY,_)],[(X,Y,_)],Check),sort(
Check,SCheck),numDiag_leg(SCheck));
        (arrangementType(3),member((NX,NY),SubList),append([(NX,NY,_)],[(X,Y,_)],Check),sort(
Check,SCheck),mixDiag_leg(SCheck))),
          select((X,Y),SubList,NSubList),
          build_push_list(NSubList,(NX,NY),B).

build_push_list(_,_,[]).

% fail those potential options which include sandwiched balls> white
% black white or black white black.
fail_Push_List_combined_colour(PushList,NPushList):-
        findall(List,(member(List,PushList),not_comb_colour(List)),NPushList).

not_comb_colour(List):-
        blackList(BL),
        whiteList(WL),
        intersection(BL,List,Int1),
        intersection(WL,List,Int2),
        ((length(Int1,0),length(Int2,Len2),Len2\=0);(length(Int2,0),length(Int1,Len1),Len1\=0)).

% fail those potential options which include more pushed balls than
% pushing balls.
fail_Push_List_too_long(ML,NPushList,NNPushList):-
        length(ML,MLLen),
        findall(List,(member(List,NPushList),length(List,Len),Len<MLLen),NNPushList).

%Handles with displaying the pushing options as interactive buttons.
view_Push_options(_,[],_).
view_Push_options(W,[Opt|OtherOpt],N):-
        (member((X,Y),Opt),
        ((turn(1), new(BMP,bitmap('PushGreenBall.bmp')));
        (turn(2), new(BMP,bitmap('PushRedBall.bmp')))),
        findGraphicalCoords(X,Y,CoordX,CoordY),
        send(W,display,BMP,point(CoordX,CoordY)),
        send(BMP,recogniser,click_gesture(left,'',single,and(message(@prolog,push,W,N)))),
```

```
        fail);
        NN is N+1,
        view_Push_options(W,OtherOpt,NN).

% when pushing option is chosen, the function operates: it finds the
% pushed list according to the arrangment- right: pushing left: pushed
% or vice versa. the pushed out board balls get (*,*) coords and are
% subtracted.

push(W,N):-
        turn(1),
        tempWin(TW),
        free(TW),
        moveList(ML),
        pushList(PL),
        blackList(BL),
        whiteList(WL),

        nth1(N,PL,ChosenPushOpt),
        findall((MLX,MLY),member((MLX,MLY,_),ML),NML),
        sort(ChosenPushOpt,SCPO),
        last(NML,LNML),last(SCPO,LSCPO),
        findHigherValCell(LNML,LSCPO,Max),
        ((member(Max,NML),(reverse(NML,RNML),reverse(SCPO,RSCPO),pushedList1(RNML,P
NML),pushedList1(RSCPO,PSCPO)));
        (member(Max,SCPO),pushedList2(NML,PNML),pushedList2(SCPO,PSCPO))),

        subtract(BL,NML,NBL),append(NBL,PNML,FinalBL),
        subtract(WL,SCPO,NWL),append(NWL,PSCPO,FinalWL),

        subtract(FinalBL,[('*','*')],NFinalBL),
        subtract(FinalWL,[('*','*')],NFinalWL),

        retract(blackList(_)),
        assert(blackList(NFinalBL)),
```

```
        retract(whiteList(_)),
        assert(whiteList(NFinalWL)),
        switchPlayers(W).

push(W,N):-
        turn(2),
        tempWin(TW),
        free(TW),
        moveList(ML),
        pushList(PL),
        blackList(BL),
        whiteList(WL),

        nth1(N,PL,ChosenPushOpt),
        findall((MLX,MLY),member((MLX,MLY,_),ML),NML),
        sort(ChosenPushOpt,SCPO),
        last(NML,LNML),last(SCPO,LSCPO),
        findHigherValCell(LNML,LSCPO,Max),

        ((member(Max,NML),(reverse(NML,RNML),reverse(SCPO,RSCPO),pushedList1(RNML,P
NML),pushedList1(RSCPO,PSCPO)));
        (member(Max,SCPO),pushedList2(NML,PNML),pushedList2(SCPO,PSCPO))),

        subtract(WL,NML,NWL),append(NWL,PNML,FinalWL),
        subtract(BL,SCPO,NBL),append(NBL,PSCPO,FinalBL),

        subtract(FinalBL,[('*','*')],NFinalBL),
        subtract(FinalWL,[('*','*')],NFinalWL),

        retract(blackList(_)),
        assert(blackList(NFinalBL)),
        retract(whiteList(_)),
        assert(whiteList(NFinalWL)),
        switchPlayers(W).
```

```
% these functions retrieve the pushed list of a given list. when a ball
% is thrown out of the board it gets the (*,*) coords.

pushedList1([(X,Y)],[(Xn,Yn)]):-
        ((arrangementType(1),ValX is X-1,ValY=Y);
        (arrangementType(2),ValX is X,previous_char(ValY,Y));
        (arrangementType(3),ValX is X-1,previous_char(ValY,Y))),

        ((not(validCell(ValX,ValY)),Xn = '*',Yn = '*',scoring(BS,WS),
        ((turn(1),NBS is BS+1,set_scoring(NBS,WS));(turn(2),NWS is
WS+1,set_scoring(BS,NWS))));
        (Xn is ValX, Yn = ValY)).



pushedList1([_|Other],[(NX,NY)|B]):-
        nth1(1,Other,(NX,NY)),
        pushedList1(Other,B).

pushedList2([(X,Y)],[(Xn,Yn)]):-
        ((arrangementType(1),ValX is X+1,ValY=Y);
        (arrangementType(2),ValX is X,next_char(Y,ValY));
        (arrangementType(3),ValX is X+1,next_char(Y,ValY))),

        ((not(validCell(ValX,ValY)),Xn = '*',Yn = '*',scoring(BS,WS),
        ((turn(1),NBS is BS+1,set_scoring(NBS,WS));(turn(2),NWS is
WS+1,set_scoring(BS,NWS))));
        (Xn is ValX, Yn = ValY)).

pushedList2([_|Other],[(NX,NY)|B]):-
        nth1(1,Other,(NX,NY)),
        pushedList2(Other,B).

% the function looks for available broadside moves of the chosen balls
% set and displays the options using an additional window.
% It brakes the adjecant free cellls around the move list into right
% left groups.
```

```
scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(1),
        moveList([(_,Y,_)|_]),
        next_char(Y,NY),
        previous_char(PY,Y),
        findall((ValX,PY),member((ValX,PY),SBroadsideMove),RightList),
        findall((ValX,NY),member((ValX,NY),SBroadsideMove),LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).

scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(2),
        moveList([(X,_,_)|_]),
        NX is X+1,
        PX is X-1,
        findall((NX,ValY),member((NX,ValY),SBroadsideMove),RightList),
        findall((PX,ValNY),member((PX,ValNY),SBroadsideMove),LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).

scan_broadside(W,SBroadsideMove,CheckVal):-
        arrangementType(3),
        moveList(ML),
        findall((ValX,ValY),(member((ValX,ValY),SBroadsideMove),((next_char(ValY,YML),memb
er((ValX,YML,_),ML));(XML is (ValX-1),member((XML,ValY,_),ML)))),RightList),
        subtract(SBroadsideMove,RightList,LeftList),
        broadside_option_list(W,RightList,LeftList,CheckVal).

% the function, uses the broken left right free cells lists in order to
% build tripples or couples of cells where the chosen balls can be
% moved.
broadside_option_list(W,RightList,LeftList,CheckVal):-
        sort(RightList,SRightList),
        sort(LeftList,SLeftList),

        moveList(ML),
```

```
        length(ML,MLlen),
        subList(SRightList,MLlen,Opt1),
        subList(SLeftList,MLlen,Opt2),

        reverse(SRightList,RevSRightList),
        reverse(SLeftList,RevSLeftList),

        subList(RevSRightList,MLlen,Opt3),
        subList(RevSLeftList,MLlen,Opt4),

        sort(Opt1,SOpt1),
        sort(Opt2,SOpt2),
        sort(Opt3,SOpt3),
        sort(Opt4,SOpt4),

        append([SOpt1],[SOpt2],List1),
        append(List1,[SOpt3],List2),
        append(List2,[SOpt4],FinalList),

        sort(FinalList,SFinalList),
        delete(SFinalList,[],SSFinalList),

        find_only_valid(SSFinalList,SSSSFinalList),

        ((not(highLight_broadside_options_maat(W,SSSSFinalList)),CheckVal is 0);(CheckVal is 1)).

%the function sifts illegal, discontinuous sets.
find_only_valid(List,OnlyValidOptList):-
        findall(L,(member(L,List),compatibilityConvertion(L,NL),(horiz_leg(NL);numDiag_leg(NL);
mixDiag_leg(NL))),OnlyValidOptList).


% the function creates a window in which the valid broadside moves are
% displayed as buttons.
highLight_broadside_options_maat(W,FinalList):-
```

```
        length(FinalList,N),
        not(N is 0),
        WinLen is 231*N,
        new(TempWin,window('Choose An Option:',size(WinLen,231))),
        send(TempWin,open),
        retract(tempWin(_)),
        assert(tempWin(TempWin)),
        retract(broadSideList(_)),
        assert(broadSideList(FinalList)),
        highLight_broadside_options(W,TempWin,FinalList,0).

highLight_broadside_options(_,_,[],_).

highLight_broadside_options(W,TempWin,[CurrList|OtherLists],N):-
          ((turn(1),Colour = 'red');(turn(2),Colour = 'green')),
          moveList(ML),
          ValX is 231*N,
          NN is N+1,
          new(Board,bitmap('minBoard.bmp')),
          send(TempWin,display,Board,point(ValX,0)),
          new(Box,box(232,232)),
          send(TempWin,display,Box,point(ValX,0)),
          draw_minimized(TempWin,ML,Colour,N),
          draw_minimized(TempWin,CurrList,'turquoise',N),

send(Box,recogniser,click_gesture(left,'',single,and(message(@prolog,broadSide,W,TempWin,N)))),
          highLight_broadside_options(W,TempWin,OtherLists,NN).


%the function displays the different options over the new window.
draw_minimized(_,[],_,_).

draw_minimized(TempWin,[(X,Y)|B],Colour,N):-
        findGraphicalCoords(X,Y,CoordX,CoordY),
        NCoordX is (CoordX/3)+(231*N),
```

```
        NCoordY is (CoordY/3),
        new(Circle,circle(11)),
        send(Circle,fill_pattern(colour(Colour))),
        send(Circle,colour,Colour),
        send(TempWin,display,Circle,point(NCoordX,NCoordY)),
        draw_minimized(TempWin,B,Colour,N).

draw_minimized(TempWin,[(X,Y,_)|B],Colour,N):-
        findGraphicalCoords(X,Y,CoordX,CoordY),
        NCoordX is (CoordX/3)+(231*N),
        NCoordY is (CoordY/3),
        new(Circle,circle(11)),
        send(Circle,fill_pattern(colour(Colour))),
        send(Circle,colour,Colour),
        send(TempWin,display,Circle,point(NCoordX,NCoordY)),
        draw_minimized(TempWin,B,Colour,N).


% the function operates when the player chooses broadside option. It
% updates the board according to the player's choice.
broadSide(W,TempWin,N):-
        turn(1),
        free(TempWin),
        broadSideList(FinalList),
        blackList(BL),
        nth0(N,FinalList,BSList),
        moveList(ML),
        findall((X,Y),member((X,Y,_),ML),NML),
        subtract(BL,NML,NBL),
        append(NBL,BSList,NNBL),
        retract(blackList(_)),
        assert(blackList(NNBL)),
        switchPlayers(W).
```

```
broadSide(W,TempWin,N):-
        turn(2),
        free(TempWin),
        broadSideList(FinalList),
        whiteList(WL),
        nth0(N,FinalList,BSList),
        moveList(ML),
        findall((X,Y),member((X,Y,_),ML),NML),
        writeln(NML),
        subtract(WL,NML,NWL),
        append(NWL,BSList,NNWL),
        retract(whiteList(_)),
        assert(whiteList(NNWL)),
        switchPlayers(W).

%the function displayes as buttons all available inline options.

highLightAvailCell(_,[]).
highLightAvailCell(W,[(X,Y)|B]):-
        findGraphicalCoords(X,Y,CoordX,CoordY),
        NCoordX is CoordX+12,
        NCoordY is CoordY+12,
        new(Circle,circle(15)),
        send(Circle,fill_pattern(colour(turquoise))),
        send(Circle,colour,turquoise),
        send(W,display,Circle,point(NCoordX,NCoordY)),
        send(Circle,recogniser,click_gesture(left,'',single,and(message(@prolog,inline,W,X,Y)))),
        highLightAvailCell(W,B).

% choosing an inline option operates the function. the function updates
% the balls coords according to the player's choice. It handles 1)one
% ball inline 2) a set of balls performing an inline.
inline(W,NX,NY):-
        turn(1),
        blackList(BL),
```

```
        moveList([(X,Y,_)]),
        select((X,Y),BL,NBL),
        append([(NX,NY)],NBL,NNBL),
        retract(blackList(_)),
        assert(blackList(NNBL)),
        switchPlayers(W).

inline(W,NX,NY):-
        turn(1),
        tempWin(TempWin),
        free(TempWin),
        blackList(BL),
        moveList(ML),
        append([(NX,NY,_)],ML,Temp),
        sort(Temp,STemp),
        nth1(1,STemp,(FX,FY,_)),
        last(STemp,(LX,LY,_)),
        append([(NX,NY)],BL,NBL),
        (((not((FX is NX, FY == NY)),select((FX,FY),NBL,NNBL));(not((LX is NX, LY ==
NY)),select((LX,LY),NBL,NNBL))),
        retract(blackList(_)),
        assert(blackList(NNBL))),
        switchPlayers(W).


inline(W,NX,NY):-
        turn(2),
        whiteList(WL),
        moveList([(X,Y,_)]),
        select((X,Y),WL,NWL),
        append([(NX,NY)],NWL,NNWL),
        retract(whiteList(_)),
        assert(whiteList(NNWL)),
        switchPlayers(W).
```

```
inline(W,NX,NY):-
        turn(2),
        tempWin(TempWin),
        free(TempWin),
        whiteList(WL),
        moveList(ML),
        append([(NX,NY,_)],ML,Temp),
        sort(Temp,STemp),
        nth1(1,STemp,(FX,FY,_)),
        last(STemp,(LX,LY,_)),
        append([(NX,NY)],WL,NWL),
        (((not((FX is NX, FY == NY)),select((FX,FY),NWL,NNWL));(not((LX is NX, LY ==
NY)),select((LX,LY),NWL,NNWL))),
        retract(whiteList(_)),
        assert(whiteList(NNWL))),
        switchPlayers(W).

% the function switches between players when the move was chosen and was
% done. If the human player has chosen to play against ai, the function
% routes the program to aiReaction set of functions. Otherwise simetric
% manual functions are operated in a loop.

switchPlayers(W):-
        turn(1),aiOrTwoPlayers(0),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecBlack(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(2)),
        aiReaction(W).

switchPlayers(W):-
```

```
        turn(1),aiOrTwoPlayers(1),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecBlack(W),
        drawWithRecWhite(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(2)).


switchPlayers(W):-
        turn(2),
        send(W,clear),
        view_board(W),
        not(checkWin(W)),
        view_scoring(W),
        drawNoRecWhite(W),
        drawWithRecBlack(W),
        reset_moveList,
        retract(turn(_)),
        assert(turn(1)).

%the function hanndles with stuck players.

stuck(W):-
        turn(1),
        open_error_dialog_stuck,
        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawNoRecWhite(W),
        drawWithRecBlack(W),
        reset_moveList.
```

```
stuck(W):-
        turn(2),
        open_error_dialog_stuck,
        send(W,clear),
        view_board(W),
        view_scoring(W),
        drawWithRecWhite(W),
        drawNoRecBlack(W),
        reset_moveList.

%the function ends the game when a victory is achieved.

checkWin(W):-
        scoring(_,6),
        send(W,clear),
        new(GreenWins,bitmap('greenWins.bmp')),
        send(W,display,GreenWins,point(0,0)).



checkWin(W):-
        scoring(6,_),
        send(W,clear),
        new(RedWins,bitmap('redWins.bmp')),
        send(W,display,RedWins,point(0,0)).


%%      ################
%%      Ai functions:


%ring(List_of_cells,ManhattenDistFromCenter).
ring([(1,'A'),(2,'A'),(3,'A'),(4,'A'),(5,'A'),(1,'B'),(6,'B'),(1,'C'),(7,'C'),(1,'D'),(8,'D'),
    (1,'E'),(9,'E'),(2,'F'),(9,'F'),(3,'G'),(9,'G'),(4,'H'),(9,'H'),(5,'I'),(6,'I'),(7,'I'),(8,'I'),(9,'I')],4).
```

```
ring([(2,'B'),(3,'B'),(4,'B'),(5,'B'),(2,'C'),(6,'C'),(2,'D'),(7,'D'),(2,'E'),(8,'E'),
        (3,'F'),(8,'F'),(4,'G'),(8,'G'),(5,'H'),(6,'H'),(7,'H'),(8,'H')],3).

ring([(3,'C'),(4,'C'),(5,'C'),(3,'D'),(6,'D'),(3,'E'),(7,'E'),(4,'F'),
        (7,'F'),(5,'G'),(6,'G'),(7,'G')],2).

ring([(4,'D'),(5,'D'),(4,'E'),(6,'E'),(5,'F'),(6,'F')],1).

ring([(5,'E')],0).

%the function retrieves list of all cells on the board.
allCellsList(List):-
            findall(A,ring(A,_),ListOfList),
            append(ListOfList,List).

% the function checks whether two given cells on the same horizontal
% axis.
onSameHorizAxis((X1,Y1),(X2,Y2),Dist):-
            Y1 ==Y2,
            Dist is abs(X1-X2).

%same for numDiag axis
onSameNumDiagAxis((X1,Y1),(X2,Y2),Dist):-
            X1 == X2,
            char_code(Y1,CY1),
            char_code(Y2,CY2),
            Dist is abs(CY2-CY1).

%same for mixDiag axis.
onSameMixDiagAxis((X1,Y1),(X2,Y2),Dist):-
            append([(Y1,X1)],[(Y2,X2)],List),
            sort(List,SList),
            nth1(1,SList,(Yval1,Xval1)),
            nth1(2,SList,(Yval2,Xval2)),
```

```
        Dx is Xval2-Xval1,
        Dx>=0,
        char_code(Yval1,CY1),
        char_code(Yval2,CY2),
        Dy is CY2-CY1,
        Dy is Dx,
        Dist is Dx.


%manhatten distance from center calculator

manhatten_dist_from_center_calc_maat(Val):-
        whiteList(WL),
        manhatten_dist_from_center_calc(WL,Val),!.

manhatten_dist_from_center_calc_maat(List,Val):-
        manhatten_dist_from_center_calc(List,Val),!.

% the function sums up all distances from each cell in the given list to
% the center.
manhatten_dist_from_center_calc([],0).
manhatten_dist_from_center_calc([(X,Y)|Other],Val):-
        ring(List,Dist),
        member((X,Y),List),
        manhatten_dist_from_center_calc(Other,NVal),
        Val is NVal+Dist.


%compactness calculator

compactness_calc_maat(Val):-
        whiteList(WL),
        compactness_calc(WL,0,0,DoubleVal),Val is DoubleVal/2,!.

compactness_calc_maat(WL,Val):-
```

```
        compactness_calc(WL,0,0,DoubleVal),Val is DoubleVal/2,!.

compactness_calc(WL,I,_,0):-
        length(WL,I).

compactness_calc(WL,I,J,Val):-
        length(WL,J),
        NI is I+1,
        compactness_calc(WL,NI,0,Val).

% the function calculates and sums up all distances between every pair
% among the balls. As the result will be twice as big, the maat function
% divides the result into two.

compactness_calc(WL,I,J,Val):-
        nth0(I,WL,(X1,Y1)),
        nth0(J,WL,(X2,Y2)),
        manhatten_min_distance_maat((X1,Y1),(X2,Y2),Dist),
        NJ is J+1,
        compactness_calc(WL,I,NJ,OtherDist),
        Val is Dist+OtherDist.

% these functions calculate the manhatten distance between each two
% cells on the board.

manhatten_min_distance_maat((X1,Y1),(X2,Y2),Dist):-
        findall(Distance,manhatten_min_distance((X1,Y1),(X2,Y2),Distance),DistList),
        sort(DistList,SDistList),
        nth1(1,SDistList,Dist).


manhatten_min_distance((X1,Y1),(X2,Y2),Dist):-
        onSameHorizAxis((X1,Y1),(X2,Y2),Dist);
        onSameNumDiagAxis((X1,Y1),(X2,Y2),Dist);
        onSameMixDiagAxis((X1,Y1),(X2,Y2),Dist).
```

```
manhatten_min_distance((X1,Y1),(X2,Y2),Dist):-
        ring(List,_),
        member((X,Y),List),
        onSameNumDiagAxis((X1,Y1),(X,Y),Val1),
        onSameHorizAxis((X2,Y2),(X,Y),Val2),
        Dist is Val1+Val2.

manhatten_min_distance((X1,Y1),(X2,Y2),Dist):-
        ring(List,_),
        member((X,Y),List),
        onSameNumDiagAxis((X1,Y1),(X,Y),Val1),
        onSameMixDiagAxis((X2,Y2),(X,Y),Val2),
        Dist is Val1+Val2.

manhatten_min_distance((X1,Y1),(X2,Y2),Dist):-
        ring(List,_),
        member((X,Y),List),
        onSameMixDiagAxis((X1,Y1),(X,Y),Val1),
        onSameHorizAxis((X2,Y2),(X,Y),Val2),
        Dist is Val1+Val2.

% the function creates a list of all white threatened pieces (noting
% the amount of hostile balls surrounding the white ones) and retrieves
% the list's length as well.

list_and_num_of_threatened_pieces_maat(SThreatenedList,Val):-
        whiteList(WL),
        blackList(BL),
        findall([(HostileLen),(X,Y)],(member((X,Y),WL),is_threatened(WL,BL,X,Y,HostileLen)),Threa
tenedList),
        sort(ThreatenedList,SThreatenedList),
        length(ThreatenedList,Val).
```

```
list_and_num_of_threatened_pieces_maat(WL,BL,SThreatenedList,Val):-
        findall([(HostileLen),(X,Y)],(member((X,Y),WL),is_threatened(WL,BL,X,Y,HostileLen)),Threa
tenedList),
        sort(ThreatenedList,SThreatenedList),
        length(ThreatenedList,Val).

% the function checks whether a ball is threatened i.e-placed on outer
% rings and surrounded by more enemies than friends.
is_threatened(WL,BL,X,Y,HostileLen):-
        retractall(tempCell(_,_)),

        ring(Ring4,4),
        ring(Ring3,3),

        (member((X,Y),Ring3);member((X,Y),Ring4)),

        PrevXVal is X-1,
        NextXVal is X+1,
        previous_char(PrevYVal,Y),
        next_char(Y,NextYVal),

        assert(tempCell(NextXVal,Y)),
        assert(tempCell(NextXVal,NextYVal)),
        assert(tempCell(X,NextYVal)),
        assert(tempCell(PrevXVal,Y)),
        assert(tempCell(PrevXVal,PrevYVal)),
        assert(tempCell(X,PrevYVal)),

        findall((ValX,ValY),(tempCell(ValX,ValY),member((ValX,ValY),BL)),HostileBallList),
        findall((ValX,ValY),(tempCell(ValX,ValY),member((ValX,ValY),WL)),FriendlyBallList),
        length(HostileBallList,HostileLen),
        length(FriendlyBallList,FriendlyLen),
        HostileLen>FriendlyLen.

% the functions beneath find pairs of horizontal\numDiag\mixDiag balls
```

```
% out of a given list

list_of_horiz_pairs(List,BallList):-

        findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),horiz_leg([(X1,Y
1,_),(X2,Y2,_)])),List).


list_of_numDiag_pairs(List,BallList):-

        findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),numDiag_leg([(
X1,Y1,_),(X2,Y2,_)])),List).


list_of_mixDiag_pairs(List,BallList):-

        findall([(X1,Y1),(X2,Y2)],(member((X1,Y1),BallList),member((X2,Y2),BallList),mixDiag_leg([(X
1,Y1,_),(X2,Y2,_)])),List).



% the functions beneath find tripples of horizontal\numDiag\mixDiag

% balls out of a given list


list_of_horiz_triple(TripList,BallList):-

        list_of_horiz_pairs(List,BallList),

        findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),c
ompatibilityConvertion(TripleList,TempTripleList),horiz_leg(TempTripleList)),TripList).


list_of_numDiag_triple(TripList,BallList):-

        list_of_numDiag_pairs(List,BallList),

        findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),c
ompatibilityConvertion(TripleList,TempTripleList),numDiag_leg(TempTripleList)),TripList).


list_of_mixDiag_triple(TripList,BallList):-

        list_of_mixDiag_pairs(List,BallList),

        findall(TripleList,(member(Pair,List),member((X,Y),BallList),append(Pair,[(X,Y)],TripleList),c
ompatibilityConvertion(TripleList,TempTripleList),mixDiag_leg(TempTripleList)),TripList).


% The functions beneath find every available group out of a given

% list-single balls, pairs and tripples.


all_avail_groups_white(List):-

        whiteList(WL),
```

```
        list_of_horiz_pairs(List1,WL),
        list_of_numDiag_pairs(List2,WL),
        list_of_mixDiag_pairs(List3,WL),
        list_of_horiz_triple(TripList1,WL),
        list_of_numDiag_triple(TripList2,WL),
        list_of_mixDiag_triple(TripList3,WL),
        findall([Item],member(Item,WL),WLList),
        append([WLList,List1,List2,List3,TripList1,TripList2,TripList3],List).

all_avail_groups_black(List):-
        blackList(BL),
        list_of_horiz_pairs(List1,BL),
        list_of_numDiag_pairs(List2,BL),
        list_of_mixDiag_pairs(List3,BL),
        list_of_horiz_triple(TripList1,BL),
        list_of_numDiag_triple(TripList2,BL),
        list_of_mixDiag_triple(TripList3,BL),
        findall([Item],member(Item,BL),BLList),
        append([BLList,List1,List2,List3,TripList1,TripList2,TripList3],List).

all_avail_groups_white(WL,List):-
        list_of_horiz_pairs(List1,WL),
        list_of_numDiag_pairs(List2,WL),
        list_of_mixDiag_pairs(List3,WL),
        list_of_horiz_triple(TripList1,WL),
        list_of_numDiag_triple(TripList2,WL),
        list_of_mixDiag_triple(TripList3,WL),
        findall([Item],member(Item,WL),WLList),
        append([WLList,List1,List2,List3,TripList1,TripList2,TripList3],List).

all_avail_groups_black(BL,List):-
        list_of_horiz_pairs(List1,BL),
        list_of_numDiag_pairs(List2,BL),
        list_of_mixDiag_pairs(List3,BL),
        list_of_horiz_triple(TripList1,BL),
```

```
        list_of_numDiag_triple(TripList2,BL),
        list_of_mixDiag_triple(TripList3,BL),
        findall([Item],member(Item,BL),BLList),
        append([BLList,List1,List2,List3,TripList1,TripList2,TripList3],List).

% the function finds all available push options list of a given list
% when the whites attack.

whiteBall_push_list(List):-
        all_avail_groups_white(White),
        all_avail_groups_black(Black),
        findall((Set),(member(Group1,White),member(Group2,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1>Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List1)
,
        findall((Set),(member(Group2,White),member(Group1,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1<Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List2)
,
        append(List1,List2,List).

whiteBall_push_list(WL,BL,List):-
        all_avail_groups_white(WL,White),
        all_avail_groups_black(BL,Black),
        findall((Set),(member(Group1,White),member(Group2,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1>Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List1)
,
        findall((Set),(member(Group2,White),member(Group1,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1<Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_white(Set)),List2)
,
        append(List1,List2,List).

% the function finds all available push options list of a given list
% when the blacks attack.

blackBall_push_list(List):-
        all_avail_groups_white(White),
        all_avail_groups_black(Black),
```

```
        findall((Set),(member(Group1,White),member(Group2,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1<Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_black(Set)),List1)
,
        findall((Set),(member(Group2,White),member(Group1,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1>Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_black(Set)),List2)
,
        append(List1,List2,List).


blackBall_push_list(WL,BL,List):-
        all_avail_groups_white(WL,White),
        all_avail_groups_black(BL,Black),
        findall((Set),(member(Group1,White),member(Group2,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1<Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_black(WL,BL,Set
)),List1),
        findall((Set),(member(Group2,White),member(Group1,Black),append(Group1,Group2,Set),(
length(Group1,Len1),length(Group2,Len2),Len1>Len2),compatibilityConvertion(Set,ConvSet),(horiz
_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet)),free_pushing_space_black(WL,BL,Set
)),List2),
        append(List1,List2,List).


% the function, given a list to push, checks whether there is a
% vacant space (cell) to push the list to.


free_pushing_space_white(List):-
        appended_black_and_white(AL),
        whiteList(WL),
        subtract(List,WL,Sub),
        not(((member((X,Y),AL),append([(X,Y)],Sub,List1),append([(X,Y)],List,List2),msort(List1,SLis
t1),msort(List2,SList2),compatibilityConvertion(SList1,ConvList1),compatibilityConvertion(SList2,Co
nvList2)),((horiz_leg(ConvList1),horiz_leg(ConvList2));(numDiag_leg(ConvList1),numDiag_leg(ConvL
ist2));(mixDiag_leg(ConvList1),mixDiag_leg(ConvList2))))).


free_pushing_space_black(WL,BL,List):-
        append(WL,BL,AL),
        subtract(List,BL,Sub),
        not(((member((X,Y),AL),append([(X,Y)],Sub,List1),append([(X,Y)],List,List2),msort(List1,SLis
t1),msort(List2,SList2),compatibilityConvertion(SList1,ConvList1),compatibilityConvertion(SList2,Co
nvList2)),((horiz_leg(ConvList1),horiz_leg(ConvList2));(numDiag_leg(ConvList1),numDiag_leg(ConvL
ist2));(mixDiag_leg(ConvList1),mixDiag_leg(ConvList2))))).
```

```
free_pushing_space_black(List):-
        appended_black_and_white(AL),
        blackList(BL),
        subtract(List,BL,Sub),
        not(((member((X,Y),AL),append([(X,Y)],Sub,List1),append([(X,Y)],List,List2),msort(List1,SList1),msort(List2,SList2),compatibilityConvertion(SList1,ConvList1),compatibilityConvertion(SList2,ConvList2)),((horiz_leg(ConvList1),horiz_leg(ConvList2));(numDiag_leg(ConvList1),numDiag_leg(ConvList2));(mixDiag_leg(ConvList1),mixDiag_leg(ConvList2))))).

% the function creates a list of all available inline moves when the
% whites move.

whiteBall_inline_list(List):-
        allCellsList(AllCellsList),
        appended_black_and_white(AL),
        all_avail_groups_white(White),
        findall((Set),(member(Group,White),member(Item,AllCellsList),not(member(Item,AL)),append([Item],Group,Set),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet))),List1),
        findall((Set),(member(Group,White),member(Item,AllCellsList),not(member(Item,AL)),append(Group,[Item],Set),compatibilityConvertion(Set,ConvSet),(horiz_leg(ConvSet);numDiag_leg(ConvSet);mixDiag_leg(ConvSet))),List2),
        append(List1,List2,List).

% the functions beneath split the various push moves of black or white
% tools into two groups: push moves that lead to a throw off the board
% of the opponent's tool or a push within the borders.

push_out_push_in_maat_white(PushOutBoardList,PushInBoardList):-
        whiteBall_push_list(List),
        push_out_push_in_white(List,PushOutBoardList,PushInBoardList).

push_out_push_in_maat_white(WL,BL,PushOutBoardList,PushInBoardList):-
        whiteBall_push_list(WL,BL,List),
        push_out_push_in_white(BL,List,PushOutBoardList,PushInBoardList),!.

push_out_push_in_maat_black(PushOutBoardList,PushInBoardList):-
```

```
        blackBall_push_list(List),
        push_out_push_in_black(List,PushOutBoardList,PushInBoardList),!.

push_out_push_in_maat_black(WL,BL,PushOutBoardList,PushInBoardList):-
        blackBall_push_list(WL,BL,List),
        push_out_push_in_black(WL,List,PushOutBoardList,PushInBoardList),!.

push_out_push_in_white([],[],[]).
push_out_push_in_white([Item|A],[Item|B],C):-
        pushedList_white(Item,PL),
        member((X,Y),PL),
        not(validCell(X,Y)),
        push_out_push_in_white(A,B,C).

push_out_push_in_white([Item|A],B,[Item|C]):-
        push_out_push_in_white(A,B,C).


push_out_push_in_white(_,[],[],[]).
push_out_push_in_white(BL,[Item|A],[Item|B],C):-
        pushedList_white(BL,Item,PL),
        member((X,Y),PL),
        not(validCell(X,Y)),
        push_out_push_in_white(BL,A,B,C).

push_out_push_in_white(BL,[Item|A],B,[Item|C]):-
        push_out_push_in_white(BL,A,B,C).


push_out_push_in_black([],[],[]).
push_out_push_in_black([Item|A],[Item|B],C):-
        pushedList_black(Item,PL),
        member((X,Y),PL),
        not(validCell(X,Y)),
        push_out_push_in_black(A,B,C).
```

```
push_out_push_in_black([Item|A],B,[Item|C]):-
        push_out_push_in_black(A,B,C).


push_out_push_in_black(_,[],[],[]).
push_out_push_in_black(WL,[Item|A],[Item|B],C):-
        pushedList_black(WL,Item,PL),
        member((X,Y),PL),
        not(validCell(X,Y)),
        push_out_push_in_black(WL,A,B,C).

push_out_push_in_black(WL,[Item|A],B,[Item|C]):-
        push_out_push_in_black(WL,A,B,C).

% the functions beneath given a push list find the pushed list. 1)When
% whites attack 2) wheb blacks attack.

pushedList_white(List,PL):-
        arrangementType(List,Type),
        sort(List,SList),
        blackList(BL),
        ((nth1(1,SList,Item),member(Item,BL),reverse(SList,Rev),pushedList_1(Rev,PL,Type));
        (pushedList_2(SList,PL,Type))),!.

pushedList_white(BL,List,PL):-
        arrangementType(List,Type),
        sort(List,SList),
        ((nth1(1,SList,Item),member(Item,BL),reverse(SList,Rev),pushedList_1(Rev,PL,Type));
        (pushedList_2(SList,PL,Type))),!.

pushedList_black(List,SPL):-
        arrangementType(List,Type),
        sort(List,SList),
        whiteList(WL),
```

```
        ((nth1(1,SList,Item),member(Item,WL),reverse(SList,Rev),pushedList_1(Rev,PL,Type),sort(P
L,SPL));
        (pushedList_2(SList,PL,Type),sort(PL,SPL))),!.

pushedList_black(WL,List,SPL):-
        arrangementType(List,Type),
        sort(List,SList),
        ((nth1(1,SList,Item),member(Item,WL),reverse(SList,Rev),pushedList_1(Rev,PL,Type),sort(P
L,SPL));
        (pushedList_2(SList,PL,Type),sort(PL,SPL))),!.



pushedList_1([(X,Y)],[(ValX,ValY)],Type):-
        ((Type is (1),ValX is X-1,ValY=Y);
        (Type is(2),ValX is X,previous_char(ValY,Y));
        (Type is(3),ValX is X-1,previous_char(ValY,Y))).

pushedList_1([_|Other],[(NX,NY)|B],Type):-
        nth1(1,Other,(NX,NY)),
        pushedList_1(Other,B,Type).

pushedList_2([(X,Y)],[(ValX,ValY)],Type):-
        ((Type is(1),ValX is X+1,ValY=Y);
        (Type is (2),ValX is X,next_char(Y,ValY));
        (Type is (3),ValX is X+1,next_char(Y,ValY))).

pushedList_2([_|Other],[(NX,NY)|B],Type):-
        nth1(1,Other,(NX,NY)),
        pushedList_2(Other,B,Type).

% the function finds the arrangement type of a list 1) horiz 2)numDiag
% 3)mixDiag.

arrangementType(List,Type):-
        (((member((X1,_),List),not((member((X2,_),List),X1 \= X2))),Type is 2);
        ((member((_,Y1),List),not((member((_,Y2),List),Y1 \= Y2))),Type is 1);
```

```
        (Type is 3)).

% ^^^aiReaction functions react according to their adjustment to the
% board's state. these functions attack black balls and defend white
% ones.

% In case of immediate push out of board, the function prevents the
% worst push.

aiReaction(W):-
        whiteList(WL),
        push_out_push_in_maat_black(PushOutBoardList,_),
        length(PushOutBoardList,Len),Len\=0,

         writeln('1-immediate push prevention'),

         chooseWorstBlackPush(PushOutBoardList,WorstPush),
        ((nth1(1,WorstPush,(X,Y)),member((X,Y),WL),findBestOptToMoveCell2(X,Y,ResultWL,Resu
ltBL));
        (last(WorstPush,(X,Y)),findBestOptToMoveCell2(X,Y,ResultWL,ResultBL))),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).

% Out of the available push out options (white attacks) the function
% chooses the best option. No new push out (black attacks) options are
% created and, the more options to push blacks in board and out the
% better.
%
aiReaction(W):-
        push_out_push_in_maat_white(PushOutBoardList,_),
        length(PushOutBoardList,Len),Len\=0,

         writeln('2-best option to push out a red ball'),

         bestRedGroupToPushOut(PushOutBoardList,Best),
```

```
        boardAfterPushOut_white(Best,ResultWL,ResultBL,_),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).

% the function chooses out of all push in board options (white attacks),
% that option in which the red ball which is being pushed is the nearest
% to the board borders and that makes the red ball get even further from
% the center/

aiReaction(W):-
        push_out_push_in_maat_white(_,PushInBoardList),
        length(PushInBoardList,Len),Len\=0,

        writeln('3-best option to push a red ball within the board'),

        find_furthest_pushed_red_ball_Option(PushInBoardList,Option),
        boardAfterPushIn_white(Option,ResultWL,ResultBL,_),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).


% In case of specific threats (i.e-white surrounded by more red than
% green and placed in the outer rings.) the threatened ball is saved.

aiReaction(W):-
        list_and_num_of_threatened_pieces_maat(SThreatenedList,Val),Val\=0,

        writeln('4-when ball is thretened-surrounded by more red than green and is placed on the
outer rings'),

        last(SThreatenedList,([_,(ThX,ThY)])),
        findBestOptToMoveCell1(ThX,ThY,ResultWL,ResultBL),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).
```

```
% If the conditions above fail, the function gathers the balls to
% the center, after a minimal distance structure has been built, the
% moves are random.

aiReaction(W):-
        writeln('5-if nothing suits-gathering and randomizing'),
        whiteBall_inline_list(InLineList),
        findall((Val1,Val2,Item),(member(Item,InLineList),boardAfterInline(Item,NWL,_,_),compar
eDistToCenter(NWL,Val1),length(Item,Len),Val2 is (Len*(-1))),Options),
        sort(Options,SOptions),
        nth1(1,SOptions,(_,_,Best)),
        boardAfterInline(Best,ResultWL,ResultBL,_),
        refresh_pieces(ResultWL,ResultBL),
        switchPlayers(W).

% the function checks which of the options among the pushOutBoard (black
% pushes whites) options is the most dangerous for the state of the
% whites and must be prevented. If the worst push was made-the potential
% amount of additional pushes of black balls against whites would be the
% largest and same about threatened white pieces.

chooseWorstBlackPush(PushOutBoardList,WorstPush):-
        findall((Val1,Val2,Item),(member(Item,PushOutBoardList),boardAfterPushOut_black(Item,
NWL,NBL),comparePotentialPushes(NWL,NBL,Val1),compareThreatenedPieces(NWL,NBL,Val2)),
Options),
        sort(Options,SOptions),
        last(SOptions,(_,_,WorstPush)).

% the function checks which of the pushoutboard options when the white
% pushes is the best for the white state-ie after the push the potential
% black against white attacks will be minimal while the potential push
% whites against blacks will be maximal.
bestRedGroupToPushOut(PushOutBoardList,Best):-
        findall((Val1,Val2,Val3,Item),(member(Item,PushOutBoardList),boardAfterPushOut_white(
Item,NWL,NBL,_),comparePotentialPushes(NWL,NBL,Val1),Val1=<0,push_out_push_in_maat_wh
ite(NWL,NBL,NPushOutBoardList,NPushInBoardList),length(NPushOutBoardList,LenOut),length(
NPushInBoardList,LenIn),Val2 is (LenOut*(-1)),Val3 is (LenIn*(-1))),Options),
```

```
        sort(Options,SOptions),
        writeln(SOptions),
        nth1(1,SOptions,(_,_,_,Best)).


find_furthest_pushed_red_ball_Option(PushInBoardList,List):-
        blackList(BL),
        findall((Dist,Val,Item),(member(Item,PushInBoardList),intersection(Item,BL,RedBall),manha
tten_dist_from_center_calc_maat(RedBall,Dist),boardAfterPushIn_white(Item,_,NBL,_),manhatten
_dist_from_center_calc_maat(NBL,Val)),Options),
        sort(Options,SOptions),
        writeln(SOptions),
        last(SOptions,(_,_,List)).


findBestOptToMoveCell1(X,Y,ResultWL,ResultBL):-
        whiteBall_inline_list(InlineList),
        push_out_push_in_maat_white(PushOutBoardList,PushInBoardList),

        findall((Val1,Val2,NWL,NBL),(member(Item,InlineList),member((X,Y),Item),boardAfterInli
ne(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCompactness(NWL,Val
2)),InlineOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushInBoardList),member((X,Y),Item),boardA
fterPushIn_white(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCompact
ness(NWL,Val2)),PushInOpts),

        findall((Val1,Val2,NWL,NBL),(member(Item,PushOutBoardList),member((X,Y),Item),board
AfterPushOut_white(Item,NWL,NBL,_),compareThreatenedPieces(NWL,NBL,Val1),compareCom
pactness(NWL,Val2)),PushOutOpts),

        append([InlineOpts,PushInOpts,PushOutOpts],BestOptList),
        sort(BestOptList,SBestOptList),
        nth1(1,SBestOptList,(_,_,ResultWL,ResultBL)).


findBestOptToMoveCell2(X,Y,ResultWL,ResultBL):-
        whiteBall_inline_list(InlineList),
```

```
        push_out_push_in_maat_white(PushOutBoardList,PushInBoardList),



        findall((Val1,Val2,NWL,NBL),(member(Item,InlineList),member((X,Y),Item),boardAfterInli
ne(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maat_black(NWL,NBL
,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),InlineOpts),


        findall((Val1,Val2,NWL,NBL),(member(Item,PushInBoardList),member((X,Y),Item),boardA
fterPushIn_white(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maat_bl
ack(NWL,NBL,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),PushInOpts),


        findall((Val1,Val2,NWL,NBL),(member(Item,PushOutBoardList),member((X,Y),Item),board
AfterPushOut_white(Item,NWL,NBL,_),compareCompactness(NWL,Val2),push_out_push_in_maa
t_black(NWL,NBL,NPushOutBoardList,_),length(NPushOutBoardList,Val1)),PushOutOpts),


        append([InlineOpts,PushInOpts,PushOutOpts],BestOptList),

        sort(BestOptList,SBestOptList),

        nth1(1,SBestOptList,(_,_,ResultWL,ResultBL)).


% given a list that can be moved in inline, it returns the board's state

% after the inline.


boardAfterInline(ListToInline,NWL,NBL,Result):-

        sort(ListToInline,SListToInline),

        appended_black_and_white(AL),

        whiteList(WL),

        blackList(NBL),

        ((last(SListToInline,(X,Y)),availCell(X,Y,AL),nth1(1,SListToInline,First),select(First,SListToInli
ne,Result));

        (reverse(SListToInline,Rev),nth1(1,Rev,First),select(First,Rev,Result))),

        subtract(WL,ListToInline,SubList),

        append(SubList,Result,NWL),!.


% given a list that can be pushed within the board, it returns the

% board's state after the push.


boardAfterPushIn_white(ListToPush,NWL,NBL,NWhiteIntersected):-

        pushedList_white(ListToPush,PushedList),
```

```
        whiteList(WL),
        blackList(BL),
        intersection(WL,ListToPush,WhiteIntersected),
        intersection(BL,ListToPush,BlackIntersected),
        length(WhiteIntersected,WhiteNum),
        subList(PushedList,WhiteNum,NWhiteIntersected),
        subtract(PushedList,NWhiteIntersected,NBlackIntersected),

        subtract(WL,WhiteIntersected,WhiteRemaining),append(WhiteRemaining,NWhiteIntersect
ed,NWL),
        subtract(BL,BlackIntersected,BlackRemaining),append(BlackRemaining,NBlackIntersected,
NBL),!.

% given a list that can be pushed out of the board, it returns the
% board's state after the push.



boardAfterPushOut_white(ListToPush,NWL,NBL,NWhiteIntersected):-
        boardAfterPushIn_white(ListToPush,NWL,BL,NWhiteIntersected),
        findall((X,Y),(member((X,Y),BL),validCell(X,Y)),NBL),!.

boardAfterPushIn_black(ListToPush,NWL,NBL):-
        pushedList_black(ListToPush,PushedList),
        whiteList(WL),
        blackList(BL),
        intersection(WL,ListToPush,WhiteIntersected),
        intersection(BL,ListToPush,BlackIntersected),
        length(WhiteIntersected,WhiteNum),
        subList(PushedList,WhiteNum,NWhiteIntersected),
        subtract(PushedList,NWhiteIntersected,NBlackIntersected),

        subtract(WL,WhiteIntersected,WhiteRemaining),append(WhiteRemaining,NWhiteIntersect
ed,NWL),
        subtract(BL,BlackIntersected,BlackRemaining),append(BlackRemaining,NBlackIntersected,
NBL),!.
```

```
boardAfterPushOut_black(ListToPush,NWL,NBL):-
        boardAfterPushIn_black(ListToPush,WL,NBL),
        findall((X,Y),(member((X,Y),WL),validCell(X,Y)),NWL),!.

% compare functions: they compare the state now versus the state after a
% specific move when the new lists are NWL and NBL. The val is the
% difference that shows whether the move contributes or damages the
% white's state.

compareThreatenedPieces(NWL,NBL,Val):-
        list_and_num_of_threatened_pieces_maat(_,OrigVal),
        list_and_num_of_threatened_pieces_maat(NWL,NBL,_,NVal),
        Val is NVal-OrigVal.


compareCompactness(NWL,Val):-
        compactness_calc_maat(OrigVal),
        compactness_calc_maat(NWL,NVal),
        Val is NVal-OrigVal.

comparePotentialPushes(NWL,NBL,Val):-
        push_out_push_in_maat_black(PushOutBoardListBef,_),
        push_out_push_in_maat_black(NWL,NBL,PushOutBoardListAft,_),
        length(PushOutBoardListBef,Bef),
        length(PushOutBoardListAft,Aft),
        Val is Aft-Bef.

compareDistToCenter(NWL,Val):-
        manhatten_dist_from_center_calc_maat(ValBef),
        manhatten_dist_from_center_calc_maat(NWL,ValAft),
        Val is ValAft-ValBef.

%given new list, the function updates the lists WL and BL.

refresh_pieces(NWL,NBL):-
```

```
        appended_black_and_white(AL),
        ((append(NWL,NBL,NAL),length(NAL,Len1),length(AL,Len2),Len2>Len1,
          scoring(BS,WS),NWS is WS+1,set_scoring(BS,NWS),fail);(fail)).


refresh_pieces(NWL,NBL):-
        retract(whiteList(_)),
        retract(blackList(_)),
        assert(whiteList(NWL)),
        assert(blackList(NBL)).
```