**פרויקט גמר - מבוא לבינה מלאכותית בשפת פרולוג**

**משחק קופסא "אבאלון"**

מגישים: שוברט יובל 302599485, ברוך שושי 066070236

**תיאור המשחק**

בחרנו לבנות את המשחק קופסא "אבאלון", משחק אסטרטגיה וחשיבה המיועד לשני שחקנים. המשחק בנוי מלוח בצורת משושה. לכל שחקן 14 כדורים (שחורים או לבנים) המסודרים בקצוות הלוח. כל שחקן מזיז בתורו בין כדור אחד לשלושה כדורים מכדוריו. בנוסף ישנה האופציה לדחוף את כדורי היריב, וכן אם אחד מהכדורים יצא מהלוח כתוצאה מהדחיפה הוא יוצא מהמשחק. השחקן הראשון שדחף שישה כדורים של יריבו מחוץ ללוח הוא המנצח.

**מטרה**

ליצור מערכת מומחית שמבוססת על המשחק אבאלון, תוך יצירה של שחקן מחשב בבינה מלאכותית, כך שהמחשב יהווה יריב שקול וטוב לשחקן אנושי.



**סביבת עבודה – הוראות הרצה**

את פרוייקט הגמר כתבנו בשפת פרולוג בשילוב עם פייתון. בשפת פרולוג השתמשנו כמנוע הבינה מלאכותית של המשחק תוך שימוש באלג' האלאפאבטא, וכן לוגיקת המשחק כתבנו בפרולוג. בפייתון מימשנו את ה GUI למשחק, והשתמשנו בספריית [pyswip](https://pypi.org/project/pyswip/) על מנת להריץ את קוד הפרולוג מהפיתון.

נכתב בפייתון 3, ומצורף קובץ requirements.txt המכיל את הpyswip על מנת להתקין את התלות יש להריץ את הפקודה:

pip3 install -r requirements.txt

על מנת להריץ את התוכנית יש להריץ את הפקודה:

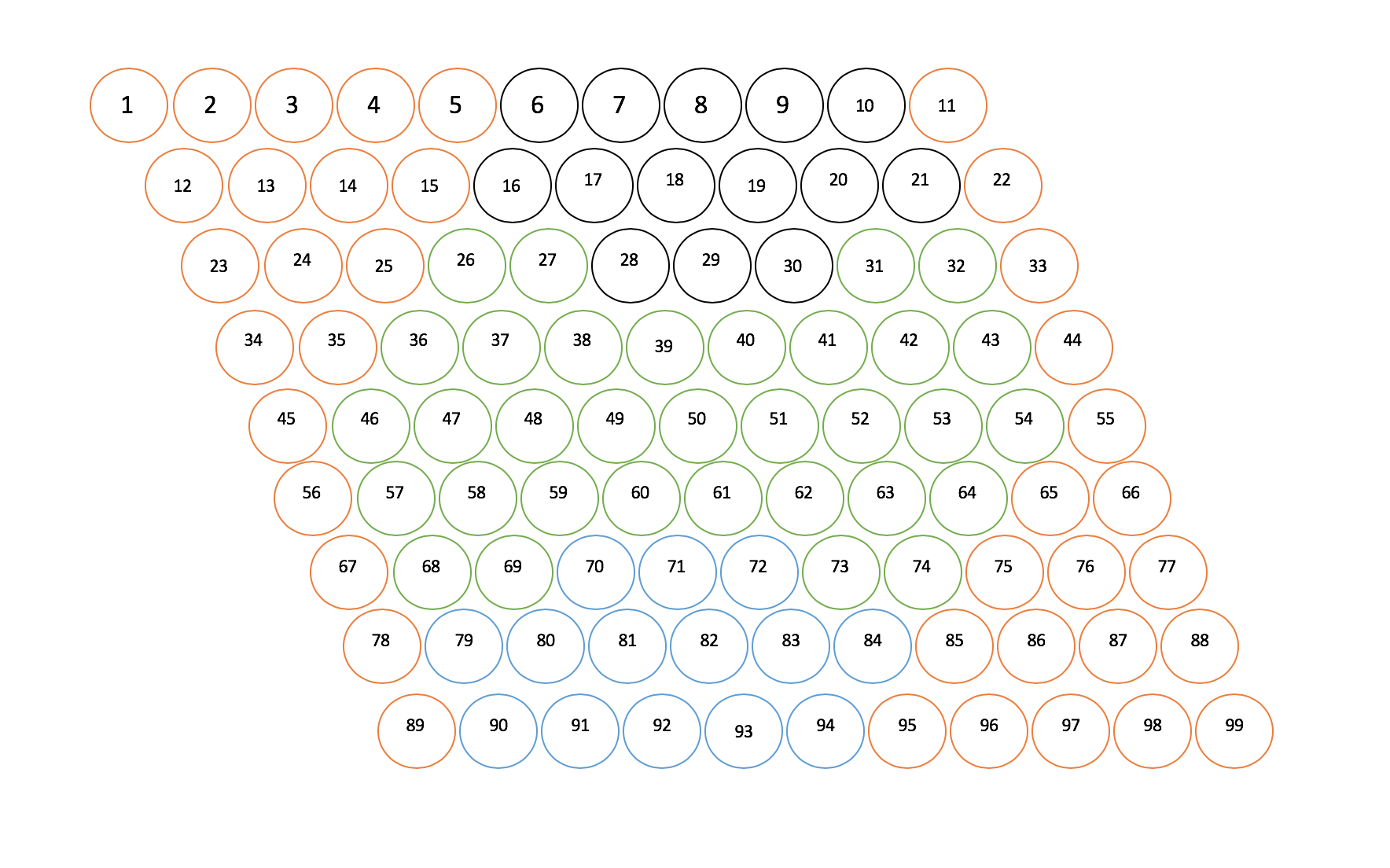
python3 abalone\_gui.py

נבדק במערכת MAC, על מנת להריץ זו בMAC הייתי צריך להגדיר משתנה סביבה - SWI\_LIB\_DIR

os.environ['SWI\_LIB\_DIR'] = '/usr/local/Cellar/swi-prolog/8.0.3/libexec/lib/swipl/lib

**ייצוג הלוח – מבנה הנתונים.**

לוח ה"אבאלון" הוא לוח משושה, אשר לכל משבצת פנימית יש שישה שכנים. זהו לוח מעט יותר מורכב מלוח ריבועי סטאנדרטי. ולא ניתן לייצגו באמצעות מטריצה רגילה. לכן ביצענו הרחב של הלוח המשושה והרחבנו אותו למקבילית ע"י הוספת משבצות (משבצות כתומות באיור), משבצות הלוח ממוספרות מ1 עד 99. והמשבצות שהוספנו יהיו "לא פעילות". עכשיו שהלוח מיוצג בצורת מטריצה יהיה יותר קל לבצע עליו פעולות בתוכנית שבנינו.



כל תא בלוח מיוצג על ידי העובדה הבאה בפרולוג

@NumberCell – Cell Number

@Value – The Value currently in cell can be {black, white, empty}

cell\_value(NumCell, Value)

בניית הלוח מתבצעת על ידי קריאה ליחס build\_board(6), פרמטר 6 מייצג את התא הראשון הפעיל במטריצה. (תאים 1-5 אינם פעיליים)

כיוונים אפשריים למהלך

ההפרש בין מספרי תאים סמוכים הוא יכול להיות מתוך הרשימה

[1, -1, 10, -10, 11, -11]

ולכן זו הרשימה של הכיוונים האפשריים של שחקן לבצע מהלך במשחק.

העובדה בפרולוג המייצגת זו היא:

directions([-1, 10, 11, 1, -10, -11]).

**מרחב מצבים**

מצב התחלתי: לוח הפתיחה כפי שמתואר בתרשים.

מצב סופי: כשאר שחקן מצליח להדוף אל מחוץ ללוח 6 מכדורי היריב.

מעבר: כל צעד חוקי – מהלך של הליכה בין כדור אחד לשלושה כדורים, מהלכי דחיפה, ומהלכי הוצאה החוצה תקניים.

מרחב מצבים: כל משבצת יכולה להיות בשלושה מצבים אפשריים: ריקה, עם כדור שחור ועם כדור לבן. הלוח מכיל 61 משבצות, לכן מרחב המצבים הוא 361, לכן לא נוכל לסרוק את כל מרחב המצבים עד המצב הסופי, ונצטרך להקטין את גודל מרחב המצבים באמצעות:

* הגבלת חיפוש לעומק – הגבלת עומק עץ אלפאבטא בהתאם לרמת המשחק.
  + רמת משחק hard – עומק 4
  + רמת משחק moderate – עומק 3
  + רמת משחק easy – עומק 2
* הגבלת חיפוש לרוחב - קטימת רשימת המהלכים האפשריים שמהם נפתח את עץ המשחק. כדי לעשות זו ניתן ציון לכל סוג מהלך, צעד של הזזה פשוט – כלומר נגביל את מספר המצבים עבור כל שחקן מספר אפשרויות של הצעדים שלו במהלך הוא מאוד גודל. לביצוע הקטימה נדרג את המהלכים. צעד של הזזה פשוט יקבל ערך נמוך, צעד של דחיפה יקבל ערך גבוה יותר, ואילו צעד של דחיפה החוצה יקבל ציון מקסימלי, ונבחר את 20 המהלכים עם הדירוג הגבוה ביותר שמהם נפתח את עץ המשחק בעומק הראשון, וכל עומק לאחריו כמות המהלכים שנסרוק לרוחב תקטן ב4.

המימוש הוא ביחס all\_moves\_for\_player.

%all\_moves\_for\_player rule  
% The relation gets the CurrentState - (Player, \_, \_, \_, \_), The Current Depth Level in the tree  
% return a SortedMoves List of the top ranked moves  
% 1. calculate the new Player - the next turn player  
% 2. find all cells which belongs to the new player  
% 3. find all all moves from those cells.  
% 4. return the top MovesLen moves.  
all\_moves\_for\_player((Player, \_, \_, \_, \_), CurrDepth, SortedMoves):-  
 player(Player, Color),  
 NewPlayer is Player\* -1,  
 findall((NewPlayer, MoveType, Direction, FromCell, ToCell),  
 (cell\_value(FromCell, Color),  
 can\_make\_move(MoveType, Color, Direction, FromCell, ToCell)),  
 Moves),  
 level(Depth),  
 MovesLen is 20 - (4 \* (Depth - CurrDepth)),  
 sort\_moves\_with\_limit(Moves, SortedMoves, MovesLen).

**פונקציה היוריסטית**

הפונק' היוריסטית משקפת את מצב הלוח, שחקן הMAX ישאף שהניקוד יהיה מקסימלי, ושחקן הMIN ירצה ניקוד מינימלי.

הפרמטרים המשפיעים על הפונק' היוריסטית שבחרנו:

* מספר החיילים בשחור\לבן – ככל שיש לשחקן יותר חיילים כך מצבו במשחק טוב יותר. לכן נתנו לפרמטר זה משקל 100
* המרחק מהאמצע של החיילים בשחור\לבן – כל צד מצבו טוב יותר אם חייליו מקובצים במרכז הלוח, כך הם רחוקים מהדפנות שמהם ניתן לדחוף את החיילים החוצה.

static\_val(\_ , Val):-  
 amount(black, BlackNum),  
 amount(white, WhiteNum),  
 board\_distance\_from\_center(black, BlackDist),  
 board\_distance\_from\_center(white, WhiteDist),  
 Val is ((((BlackNum \* 100) - (WhiteNum \* 100)) - BlackDist + WhiteDist)).

\*אופטימיזציה של יעילות – פונ static\_val נקראת עבור כל צומת בעץ המשחק של alphabeta לכן נרצה שעלותה תיהיה O(1) לכן הוספנו עובדה board\_distance\_from\_center(Color, Dist) כדי לשלוף את המרחק מהאמצע של שחקן מסויים בO(1). לאחר כל צעד אנו מעדכנים את המרחק מן האמצע של הצדדים.

**מבנה הקבצים:**

* abalone.pl – קובץ המכיל את כל הלוגיקה בפרולוג של המשחק וכן של הalphabeta.
* abalone\_gui.py – קובץ main של המשחק, מכיל את כל הלוגיקה של הGUI.
* - abalone\_prolog\_ai\_engine.pyקובץ המאתחל את קוד הפרולוג (consult), ומבצע queries לתוכנית הפרולוג.

abalone\_prolog\_ai\_engine.py

**יחסים מרכזיים בפרולוג**

\*ישנו תיעוד על כל קוד הפרולוג, וקוד הפייתון בקבצים.

% start\_game Rule  
% build the board  
% assert the number of cells for each player  
% calculate and assert the dist from center for each of the player

start\_game:-

%%%%%%%% build\_board %%%%%%%%%  
% recursive function which get a cell, and assert it with the value of the cell in the initial state of the board,  
% then it calls build\_board with the next cell.  
% end condition is when we reach the last cell.

build\_board(Number)

% get\_best\_move  
% Params: get Color of player  
% returns the next best move for player.  
% 1. validate the game is not ended - there are more than 8 cells for each player.  
% call the alpha\_beta for getting the best move ( we limit the Depth level of the tree)  
get\_best\_move(Color, MoveType, Direction, From, To):-

%all\_moves\_for\_player rule  
% The relation gets the CurrentState - (Player, \_, \_, \_, \_), The Current Depth Level in the tree  
% return a SortedMoves List of the top ranked moves  
% 1. calculate the new Player - the next turn player  
% 2. find all cells which belongs to the new player  
% 3. find all all moves from those cells.  
% 4. return the top MovesLen moves.  
all\_moves\_for\_player((Player, \_, \_, \_, \_), CurrDepth, SortedMoves):-

% can\_make\_move Rules  
% All those moves rules validates if a move is valid and possible  
% Params: MoveType, Color, Direction, FromCell, ToCell

can\_make\_move(MoveType, Color, Direction, FromCell, ToCell):-

% make\_move  
% Params: get the curr state,  
% undo the move by calling the undo\_move relation.  
undo\_move((NextPlayer, MoveType, Direction, From, To)):-

% make\_move  
% Params: get the curr state,  
% execute the move by calling the make\_move relation.  
make\_move((NextPlayer, MoveType, Direction, From, To)):-