

תוכן עניינים

1	מבוא	2
2	מבני נתונים ואלגוריתמים	3
2.1	ייצוג לוח המשחק, המשבצות והכלים	3
2.1.1	משבצות	3
2.1.2	לוח המשחק	4
2.1.3	מצב נתון במשחק	4
2.2	אלגוריתמים ושגרות מרכזיות	5
2.2.1	מודול ניהול מבני-נתונים	5
2.2.2	מודול אלפא-ביתא	8
2.2.3	מודול ניהול המשחק וממשק המשתמש	11
2.3	סיכום מבני נתונים ופרדיקטים מרכזיים	13
3	ממשק המשתמש	14
3.1	התנעת האפליקציה	14
3.2	בהירות ונוחות הממשק	14
3.3	טיפול בשגיאות	16
3.4	שיפורים ותוספות	17
4	רשימה ביבליוגרפית	18

1 מבוא

בפרויקט זה פותח המשחק Othello בשפת פרולוג, במתכונת של מחשב מול שחקן וכן מחשב מול מחשב, תוך מימוש שיטת חיפוש של אלגוריתם אלפא-בתא לטובת חישוב המהלכים האופטימאליים לביצוע בתור המחשב.

לגבי המשחק – חוקי המשחק הם החורים הקלאסיים של Othello (לא [Reversi](#)), משמע עמדת

הפתיחה היא תמיד הצלבה של זוגות כלים מנוגים במרכז לוח ריבועי מסדר זוגי, דהיינו עבור n נתון, הכלים יוצבו בקואורדינטות

$(\frac{n}{2}, \frac{n}{2}), (\frac{n}{2}+1, \frac{n}{2}), \frac{n}{2}, \frac{n}{2}+1, (\frac{n}{2}+1, \frac{n}{2}+1)$. ראה למשל

איור 1 עבור $n=8$. כמו כן, להבדיל מ-Reversi, באותו המשחק לא בהכרח מסתיים כאשר שחקן לא יכול לזוז בתורו. אם שחקן תקוע ללא מהלכים חוקיים, התור עובר ליריב ורק אם גם הוא תקוע המשחק המסתיים. לפירוט נוסף על הוראות המשחק, ראה [2] וכן

הפרויקט פותח והורץ על מהדר ו-Interpreter של SWI Prolog, גרסה 8.0.3, מחשב נייד בעל 8 GB RAM עם מעבד i7-7700 בעל מהירות של 3.60GHz.

הערות לגבי הפיתוח והפרויקט באופן כללי –

אופן ייצוג מבנה הנתונים עוצב במטרה למטב ביצועים ויעילות.

הפיתוח בוצע Top-Down, כאשר תוך מימוש פרדיקט High-Level נולד הצורך למימוש יחסי עזר, ויחסים בעלי עבודה משותפת עוצבו ונכתבו מחדש כדי להוציא את הכפילות ולרכז אותה בשגרה ייעודית אחת ייעודית לביצוע מטלה משותפת זו.

המשחק פותח עבור לוח כללי $n \times n$.

המשחק תומך במכה דרגות קושי, הקובעות הן את עומק עץ החיפוש של האלגוריתם, והן את אופן שערך העמדות (פונקציות ההערכה).

מומשו 3 פונקציות היוריסטיות להערכת עמדה נתונה.

התכנית אינה מצפה לקלט תקין בהכרח, מוודאת תקינות, ומפיקה חייוי ברור על שגיאות קלט.

ממשק המשתמש ברור ונח, בכל שלב מודפסות הנחיות במפורטות למשתמש היכן שהוא נדרש להכניס קלט, והיכן שהוא צופה מצד מודפס לו חייוי מלא של התרחשות העניינים.



2 מבני נתונים ואלגוריתמים

בסעיף זה נציג את מבני הנתונים המרכזים והשיטות ואלגוריתם שנעשה בהם שימוש בפרויקט.

2.1 ייצוג לוח המשחק, המשבצות והכלים

2.1.1 משבצות

באופן אבסטרקטי, ניתן לייצג לוח-משחק של אות'לו ע"י מטריצה ריבועית מסדר $n \times n$ כדלקמן, שהרכיב ה- a_{ij} שלה מייצג את המשבצת הממוקמת בשורה ה- i ובעמודה ה- j של לוח המשחק -

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

בכדי לממש הלכה למעשה ייצוג שכזה בשפת תכנות עילית, טבעי להשתמש במערך דו-ממדי (או למעשה "מערך של מערכים"), המאפשר גישה ישירה (RAM – Random Access Memory) לכל תא במערך (מטריצה) לפי האינדקס שלו. לא כך בפרולוג, שאינו מכיר מערכים. אזי כיצד בכל זאת ניתן לייצג מטריצה?

אלטרנטיבה אחת בפרולוג, המזכירה במקצת את רעיון המערך הדו-ממדי היא רשימה של רשימות, כאשר כל אחת מהרשימות ברשימה מייצג שורת בודדת במטריצה. ייצוג שכזה אף יכול לתמוך בממד n כללי כלשהו, לכן נראה אולי טבעי, אך לאחר מעט מחשבה ניסיון נאיבי זה נפסל מיד, מטעמי ביצועים: עבור לוח בגודל n , בכדי להגיע לרכיב ה- a_{ij} , יידרשו במקרה הגרוע n^2 קריאות רקורסיביות, ובתוחלת בערך $n^2 / 2$ קריאות (פחות או יותר), וזאת רק בשביל לגשת לרכיב יחד. זה אולי סביל אם רוב הסריקות הן סדרתיות ברצף מהרכיב הראשון בפניה השמאלית העליונה (הרכיב ה- a_{11}) ועד לרכיב האחרון בפניה הימנית התחתונה (הרכיב ה- a_{nn}), אך לא כך באות'לו: במשחק זה, בכל תור השחקן צריך לסרוק שמונה כיוונים שונים, בדיוק כמו תמנון השולח את 8 הזרועות שלו לכל עבר, כמו מלך על לוח השחמט, כך גם תכנית המחשב צריכה לבדוק שמונה כיוונים שונים לבדיקת תקפות מסעי השחקן והפיכת החיילים על הלוח, כלומר הגישה אל הרכיבים השונים במטריצה אקראית עד מאוד, מה שמצריך גישה מהירה תכופה שאינה מוגבלת לסריקות לינאריות סדרתיות.

בכדי לתת מענה לצורך זה, החלטתי לתת למשבצת הבודדת את הכבוד המגיע לה ולייצג אותה כאובייקט עצמאי, שיהיה אפשר לגשת אליו ישירות. כל משבצת שכזו, תכיל את הקואורדינטות שלה על הלוח, ואת הערך שיש כרגע על המשבצת (האם המשבצת ריקה, או האם מכילה חייל לבן/שחור). בכדי לאגד את כל המשבצות של לוח משחק נתון יחד, נתייג על משבצת עם מזהה של הלוח שאליו היא שייכת. ובאופן פורמלי, נגדיר יחס slot המייצג את אובייקט של משבצת, כדלקמן –

`slot(GridIdentifier, coordinate(I,J), CurrentValueOnSlot)`

כאשר GridIdentifier הוא מספר מזהה (מפתח) של לוח משחק מסוים, דהיינו עמדה מסוימת, coordinate(I,J) הוא יחס דינאמי לייצוג קואורדינטות המשבצת על הלוח (I) מייצג את אינדקס השורה, ו-J את אינדקס העמודה), ו-CurrentValueOnSlot מייצג את אכלוס המשבצת הנוכחי כדלקמן – 0 = משבצת ריקה, 1 = חייל של השחקן הפותח (מודפס כ-x) ו-2 חייל של השחקן השני (מודפס כ-o). למשל, הפרדיקט ה-`slot(13,coordinate(5,2),0)` משמעו שבלוח שמספרו 13, המשבצת הנמצאת בשורה החמישית על הטור השני כרגע ריקה.

תחת הנחה שהמימוש בפרולוג לניהול היחסים הדינאמיים בזיכרון הוא בטבלת גיבוב, ייצוג זה ע"י פרדיקטים, מאפשר לגשת לכל משבצת בתוחלת של $\Theta(1)$ בלבד!

2.1.2 לוח המשחק

באשר ללוח המשחק עצמו, במימוש שנבחר בפרויקט זה אין לו ייצוג ממשי כמבנה נתונים אלא תגית עם ערך מזהה (Grid Id), מה שמאפשר להתייחס אליו באופן לוגי כאילו הוא קיים, בעוד שבפועל מדובר אך ורק בתגית המאפשר בו בעת לגשת לכל המשבצות השייכות ללוח מסוים ולזהות את הלוח באופן חד-חד ערכי. בכדי לנהל מפתחות, מוגדר נומרטור ע"י הפרדיקט `next_idle_grid_id` שדואג "להצביע" על הערך המספרי הפנוי הבא בשביל הלוח (מצב משחק) הבא שייווצר לאחר ביצוע איזשהו מסע.

הערה: נבחין כי הלוח לא מכיל מידע לגבי הממדים שלו. הסיבה לכך, היא שכל הלוחות, דהיינו במשחק נתון כולם מאותו סדר, לכן אין טעם לבזבז זיכרון ולתחזק ערך זה שוב ושוב בכל עמדה/לוח מחדש. במקום זאת, סדר הלוחות נשמר במשתנה גלובלי $\text{dimension}(N)$, כאשר N הוא סדר הלוח של כלל העמדות במשחק, והוא נקבע ע"י המשתמש בשלב האתחול טרם המשחק.

2.1.3 מצב נתון במשחק

מצב נתון במשחק מיוצג ע"י היחס `pos` (קיצור של position) שלהלן –

`pos(GridId, Turn, coordinate(I,J))`

כאשר `GridId` הוא מספר מזהה של לוח המשחק בעמדה הנתונה, `Turn` מייצג תור מי לשחק כרגע – 1, השחקן הראשון (Max) או 2, השחקן השני (Min), ו-`coordinate(I,J)` המכילה את קואורדינטות המשבצת שהונח עליה כלי במסע האחרון שהוביל למצב הנוכחי.

הערות והרחבות –

- הקשר בין הדמיון לייצוג ערכי המשבצת לערכי תורות השחקנים אינו מקרי: במכוון העיצוב בוצע כך שניתן יהיה לשייך ערך של כלי נתון על משבצת ישירות לשחקן אליו הוא שייך (שחור או לבן במשחק ממשי), כאשר בתכנית המחשב 1 מייצג גם את תור השחקן הראשון וגם את ערך הכלים שלו על הלוח, ובאופן דומה 2 מייצג גם את תור השחקן השני וגם את ערך כליו על הלוח.
- לעיתים בתכנית במקום `Turn` נעשה שימוש במשתנה `Player` או וריאציות דומות לו.
- הקואורדינטה האחרונה אמנם אינה חיונית לייצוג המצב, אך שימושית בהקשר של תכנית מחשב לצורך חיווי למשתמש מהו המסע האחרון שבוצע שהוביל לעמדה העדכנית.
- כאן אולי ניכר חסרון בשיטת הייצוג: ביחס המייצג מצב נתון כמעט ואין מידע, אלא רק "קצה של חוט", הלוא הוא מזהה הלוח (`GridId`) שעל-פיו יש למשוך את המידע. עם זאת, לאור הביצועים הטובים והקלות שבה ניתן להעביר "עמדה" כארגומנט בין יחסים תוך "כיווצה" לכדי תגית בודדת מטים את המשקל לטובת בחירה זו.

נסכם את ייצוג הלוח, המשבצות והכלים –

- במימוש זה, **לוח המשחק** אין ייצוג עצמאי כמבנה נתונים פיזי, אלא לוגי בלבד: לכל לוח יש מספר מזהה `Grid Id`, המופיע כתגית בפרדיקטים של כל המשבצות המאוגדות תחת לוח זה.
- **משבצת** אינדיווידואלית בלוח משחק מיוצגת ע"י היחס `slot`, המכיל שלושה רכיבים: מזהה הלוח, מצייני מיקום על הלוח (קואורדינטות), וערך נוכחי שמאוכסל על המשבצת: 0 עבור ערך ריק, 1 עבור ערך של כלי בצבע השחקן הראשון (מודפס כ-x) ו-2 עבור ערך של כלי בצבע השחקן השני (מודפס כ-o).
- **מצב נתון** במשחק מיוצג ע"י היחס `pos`, המכיל אף הוא שלושה רכיבים: מספר מזהה של הלוח, תור מי לשחק (שחקן ראשון או שני, וערך זה מקביל לערכים התואמים של החיילים של השחקנים על הלוח) וקואורדינטה של המשבצת שהונח עליה כלי במסע האחרון שהוביל לעמדה המיוצגת ע"י המצב הנתון. בכדי לגזור את מצב העמדה בפועל יש להשתמש ב- `Grid Id` ולנתח את היחסים של המשבצות המאוגדות תחתיו.

2.2 אלגוריתמים ושגרות מרכזיות

בסעיף קודם הוצגו מבני הנתונים המרכזיים המשמשים לייצוג וניהול המשחק. בסעיף זה נציג את השגרות ואלגוריתמים האחראים לאתחול, לתחזוק ולעבד מבני נתונים אלו, וכן שגרות נוספות כגון פונקציות היוריסטיות ותכניות אינטראקטיביות מול משתמש. סעיף זה מחולק למודולים לוגיים, בהתאם לנושא/קטגוריה/אחריות אליהם ניתן לסווג/לשייך את השגרה/אלגוריתם:

- **מודול ניהול מבנה-נתונים** – מודול זה אחראי על אתחול, תחזוקה ועיבוד מבני הנתונים הפנימיים של הלוח, משבצות, כלים, שחקנים וכד' בהתאם ללוגיקה שתוארה לעיל.
- **מודול אלפא-ביתא** – מודול זה אחראי על מימוש אלגוריתם החיפוש אלפא-ביתא, לרבות מימוש פונקציות היוריסטיות להערכת עמדות.
- **מודול ממשק המשתמש** – מודול זה אחראי הן על ניהול מהלך המשחק, והן על ממשק האינטראקציה מול המשתמש, הכולל מימוש פונקציות קלט/פלט, טיפול בשגיאות והצגה גראפית רציפה של מצב המשחק העדכני בכל שלב.

2.2.1 מודול ניהול מבני-נתונים

2.2.1.1 סריקות הלוח

בהרבה מהשגרות נדרש לסרוק את הלוח או חלקו, לעיתים מדובר בסריקה הלוח כולו מההתחלה ועד הסוף (למשל באתחול והדפסה) ולעיתים נדרש רק את חלקו אך בכל 8 הכיוונים ממשבצת נתונה (למשל באיתור אחר משבצות שיש להפוך לאחר הנחת חייל במשבצת ריקה). לפיכך, בכדי לא לחזור על אותה לוגיקת סריקה שוב ושוב, נכתבו שתי שגרות עזר. האחת מסייעת בסריקה סדרתית של הלוח מתחילתו וסופו ע"י החזרת האינדקס העוקב לאינדקס נתון, הקריאה הראשונית היא עם אינדקס (1,1) והממד N של סדר הלוחות המשמש כתנאי עצירה ואינדיקציה לעבור משורה אחת לזו שאחריה –

`get_next_sequential_index(I, J, Next_I, Next_J, Dimension_N)`

השגרה השנייה מסייעת בסריקה אוקטט-כיוונית ע"י החזרת האינדקס העוקב לאינדקס נתון בכיוון מבוקש, למשל דרום-מערב –

`get_next_directional_index(I, J, Next_I, Next_J, Direction)`

שמונת הכיוונים האפשריים נקבעים מראש כאטומים ברשימה ע"י היחס שלהלן –

`direction_list([north,northEast,east,southEast,south,southWest,west,northwest])`

כנגד אטומים אלו מוגדרים הוראות ניווט אריתמטיות ביחס לערכי הקואורדינטות (למשל צפונה משמעו למעלה, דהיינו העלאת האינדקס של I, ומערה משמעו שמאלה, דהיינו הפחתת הערך של J. בכדי לאפשר סריקות לכל שמונה הכיוונים גם יחד מבלי הצורך לציין את כולם במפורש, בחלק מהשגרות המפעילות את שגרת העזר הכיוונית נעשה שימוש ביחס member לבחירה לא-דטרמיניסטית של כיוון מתוך מרשימה, כך שבכל כישלון מנגנון ה-backtracking בוחר בכיוון הבא.

2.2.1.2 אתחול הלוח

אתחול הלוח הראשוני מבוצע ע"י השגרה `initialize_starting_pos_grid`, המקבלת כקלט את סדר הלוח ואת קואורדינטת הפינה השמאלית העליונה, וע"י שימוש בסריקה סדרתית עוברת אינדקס אחר אינדקס, ורושמת בזיכרון משבצת אחר משבצת באופן דינאמי עם `assert`. הלוח הראשוני נבנה עם מספר מזהה אפס. יחד איתו מוגדר נומרטור `next_idle_grid_id` שכאמור לעיל דואג לנהל מעקב על המספר המזהה הפנוי הבא.

2.2.1.3 שכפול לוח

שגרה זו שימושית לאלגוריתם החיפוש, שבדוק מלא מצבים אפשריים, ונדרש לשמר בזיכרון רשימה של מצבים (לוחות). לשם כך, תוך שימוש בנומרטור שהוזכר לעיל הוגדר היחס `duplicate_grid` המקבל כקלט את המספר המזהה של היחס המבוקש לשכפול, ומחזיר כפלט את המספר המזהה של הלוח החדש שנוצר (הלוח המשוכפל). על לוח זה אפשר לבצע מניפולציות ושינויים מבלי שאלו יבואו לידי ביטוי בלוח המקורי. במונחים של OOP, זהו שכפול "עמוק" ולא "רדוד" (shallow copying).

2.2.1.4 הדפסת לוח

השגרה `print_grid` מקבלת כקלט מספר מזהה של לוח ומדפיסה אותו ואת תוכנו על תורים הפלט הנוכחי (קונסול כברירת מחדל) בצורה של מטריצה. כדי להבחין בין חיילי שני השחקנים ומשבצות ריקות, משבצות ריקות מודפסות עם `[_]`, משבצות עם חיילים של השחקן הראשון (שחקן max) מודפסים עם 'x', וחיילים של השחקן השני (שחקן min) מודפסים עם 'o'. בנוסף ללוח עצמו מודפסים גם הקואורדינטות על שני הצירים בכדי לספק למשתמש האנושי ממשק נח שיאפשר לו לנתח את העמדה ולבחור את המסע הבא בקלות.

2.2.1.5 ולידציות מסע חוקי

הפרדיקט הבא מקבל כקלט מספר מזהה לוח, קואורדינטה של משבצת מבוקשת וערך של הכלי שאותו הוא רוצה להציב במשבצת זו –

`validate_coordinate(GridId,coordinate(I,J),RequestedVal)`

הפרדיקט מצליח אם המסע המבוקש חוקי ונכשל אחרת. הבדיקה עצמה מבוצעת ע"י בחירה לא-דטרמיניסטית של כיוון מתוך אחד משמונת הכיוונים האפשריים כפי שהוסבר [לעיל](#), ובדיקה שלפחות עבור אחד מכיוונים אלו, המשבצת הסמוכה (מהכיוון הנוכחי) למשבצת המבוקשת מכילה כלי של היריב. בדיקה זו פשוטה לאור התאימות בין קידוד ערכי השחקנים והכלים שלהם על הלוח, כך שמספיק לבדוק שהערך המוחלט הוא 1. במידה ובדיקה זו נכשלת, מנגנון ה-Backtracking עובר לכיוון הבא או שהפרדיקט נכשל אם מוצו כל הבחירות. אם הבדיקה הראשונית הזו מצליחה, מופעלת שגרת עזר נפרדת שאחראית להמשיך ולבדוק את הכיוון הספציפי הזה שיש לגביו "lead", ולוודא שברצף המשבצות הסמוכות בכיוון זה יש לפחות משבצת אחת עם כלי של השחקן הנוכחי, הקודמת למשבצת ריקה. שגרת העזר היא:

`validate_direction_recursively(GridId,coordinate(I,J),RequestedVal,Direction)`

על בסיס שגרת הוולידציה עבור משבצת יחידה, הוגדר יחס מעטפת האחראי להחזיר אוסף (רשימה) של קואורדינטות חוקיות למסע אפשרי עבור מצב לוח נתון. הרעיון דומה באופיו ל-moves באלפא-ביתא, אלא ששם מוחזרת רשימה של פוזיציות, וכאן רשימה של קואורדינטות, כלומר המהלכים שמובילים אל הפוזיציות. רשימה זו שימושית הן לצורך אכיפת קלט לא חוקי ממשתמש, קיצור חישובים באלפא-ביתא, בדיקת מצב משחק (האם לפחות לאחד השחקנים יש מסע חוקי בעמדה נוכחית?) ועוד. שגרת המעטפת היא –

`get_legal_coordinates(+GridId,+Player,-LegalCoordinatesList)`

2.2.1.6 היפוך צבעי כלים

השגרה `flip_pieces` היא השגרה המרכזית האחראית להיפוך צבעי כלים על הלוח, ופועלת בהנחה שאכן כבר בוצעה ולידציה והמסע אכן חוקי. השגרה פועלת ע"פ עקרונות דומים של שגרת הוולידציה, מבחינת הבחירה האי-דטרמיניסטית של שמונת הכיוונים בסריקה. להבדיל ממנה, בוולידציה די היה מספיק בלפחות כיוון אחד שיצליח, כדי לוודא שהמסע חוקי. כאן, יש בכל מקרה לבדוק את כל שמונת הכיוונים בכדי להחליט מה להפוך ומה לא. לפיכך, לאחר גילוי כיוון "טוב"

ורלוונטי" מבוצע כישלון יזום ע"י fail, שיבטיח את הפעלת מנגנון ה-Backtracking וסריקת הכיוון הבא, ורק לאחר שכל שמונת הכיוונים נסרקים ומטופלים מבוצעת ההסתעפות להצלחה וסיום.

מאחר שעד תום סריקת כיוון מסוים עוד לא תמיד ידוע בכל המקרים אם הכלים המיועדים להפיכה אכן ייהפכו (תלוי בשאלה אם בהמשך הכיוון הנוכחי ניתקל בכלי שלנו לפני שניתקל במשבצת ריקה), נשמרים כל הקואורדינטות של המשבצות הפוטנציאליות להפיכה על רצף הכיוון הנוכחי ברשימה זמנית. אם בהמשך יתגלה שהכיוון כוזב, ההסתעפות נכשלת, הרשימה נזרקת, וממשיכים לכיוון הבא. אחרת, מעבירים את הרשימה ליחס עזר נפרד שייעודו לבצע את היפוך הערכים בפועל.

יחס העזר לאיסוף רשימת המשבצות המיועדות להפיכה –

`getListOfCoordinatesToFlip(Id,coordinate(I,J),Val,Direction,CoordinatesList)`

יחס העזר שאחראי לבצע את ההיפוכים בפועל –

`flip_list(GridId,Val, CoordinatesListToFlip)`

ההיפוך של כלים בתכנית מבוצע ע"י שינוי ערכם לזה של השחקן היריב. בכדי לעדכן ערך של משבצת נתונה, מאחזרים אותה מהזיכרון עם `retract` וכותבים בחזרה עם `assert` עם הערך המעודכן.

2.2.1.7 ביצוע מסע על הלוח

היחס הבא מהווה את אבן הבניין ליחס `moves` של אלגוריתם אלפא-ביתא –

`makeLegalMove(+coordinate(I,J), +pos(Player1,Grid1Id), -pos(Player2,Grid2Id))`

יחס זה מקבל כקלט מצב נתון (ארגומנט ה-`pos` הראשון) וקואורדינטה של משבצת על הלוח במצב זה, מריץ את המהלך (בהנחה שהוא חוקי), ומחזיר את המצב העדכני (פרמטר `pos` השני) המתקבל לאחר ההרצה, לרבות היפוך צבעי הכלים הרלוונטיים, ועדכון תור מי לשחק בעמדה החדשה. במידה והמסע המבוקש בכלל לא חוקי, הפרדיקט נכשל. הוולידציה על חוקיות המהלך המבוקש והיפוך צבעי הכלים מבוצעים באמצעות היחסים הייעודיים שהוצגו לעיל.

בכדי שהיחס יהיה אכן רלוונטי עבור `moves` ויאפשר לבצע "את כל המהלכים האפשריים" מתוך עמדה נתונה, ולא רק מהלך מסוים עבור איזושהי קואורדינטה קונקרטית, היחס תומך גם בקבלת קואורדינטה לא מאותחלת, במקרה כזה, היא מאותחלת באופן לא-דטרמיניסטי לאחת מהמשבצות הפנויות על הלוח, מה שמאפשר הרצת כל המהלכים האפשריים מעמדה נתונה תוך שימוש ב-`backtracking`.

2.2.2 מודול אלפא-ביתא

2.2.2.1 נקודות דמיון ושוני

אלגוריתם אלפא-ביתא בפרויקט זה מבוסס אמנם על [שבלונת האלגוריתם שבספר של ברטקו](#) [2], אך מכיל בנוסף למימוש השגרות האבסטרקטיות גם תוספת של כמה שיפורים –

1. **ניהול רמת עומק** – שליטה במספר הרמות של עץ החיפוש באלגוריתם.
2. **שמירת תוצאות חישוב קודמות** בטבלת גיבוב בכדי לחסוך חישובים כפולים.
3. **תאימות וגמישות פונקציית ההערכה** בהתאם לרמת דיוק (קושי) מבוקשת ושימוש בפונקציות היוריסטיות.
4. **אתחול אוטומטי** של ערכי אלפא-בתא ל- $-\infty$ ו- ∞ , בהתאמה, בקריאה ראשונית.

נסביר בפירוט את השינויים –

- **ניהול רמת עומק:** בכדי לנהל רמת עומק, נוספו לאלגוריתם שני פרמטרים, האחד MaxDepth המייצג את רמת העומק המקסימאלית לחיפוש, והשני, DepthLevel המייצג את רמת העומק הנוכחית. רמת העומק המקסימאלית נקבעת בהתאם לרמת הקושי שהשחקן מבקש: ככל שהשחקן דורש רמת קושי גבוהה יותר, כך גם רמת העומק החיפוש של האלגוריתם תעלה. בקריאה ראשונית של האלגוריתם DepthLevel מאותחל לרמה 0, ובכל הפעלה רקורסיבית הרמה מועלית ב-1. בכדי לחלחל לכל רמות הרקורסיה פרמטרים אלו מועברים גם לשגרות הפנימיות שקוראות לאלפא-ביתא באופן רקורסיבי (goodenough ו-boundedbest).

- **שמירת תוצאות חישוב קודמות:** בכדי לחסוך בביצועים ע"י מניעת צלילה לרמות עומק רק לשם הערכת עמדות שכבר ממילא חושבו בסריקות קודמות של עץ החיפוש, בפרויקט זה מומש ניהול טבלת גיבוב לשמירת תוצאות ביניים של הערכה של כל עמדה. המימוש אינו טריוויאלי כלל –

א. ראשית, בכדי שניתן יהיה לאחזר תוצאות בדיעבד עבור עמדות זהות שחוזרות על עצמן בהמשך נדרש איזשהו מפתח משותף שזהה לכל העמדות הזהות, אולם אין כזה בייצוג הנוכחי, שכן הממספר המזהה של לוח בייצוג הנוכחי לא תורם מאומה באשר לעמדה עצמה, וייתכנו הרבה פוזיציות זהות עם מספרי לוח שונים. לפיכך, הוגדרה שגרת עזר שבונה לכל לוח מפתח-גיבוב שיזהה אותו ויתאים לכל העמדות הזהות ללא תלות במספר הלוח. המפתח נבנה משרשור ערכי תכני כל המשבצות באופן סדרתי מההתחלה לסוף והמרתם למחרוזת אוטומית, שהינה חד-ערכית לעמדות זהות. שגרת העזר לבניית מפתח הגיבוב היא `get_hash_key`.

ב. שנית, הערך שניתן ע"י פונקציות ההערכה מתייחס לעמדה תחת חיפוש לרמת עומק מסוימת. בחיפוש אחר עבור אותה עמדה, שכולל יותר רמות עומק, החיפוש עלול להגיע לתוצאות פחות גסות ויותר ומדויקות (ולחפך בריצות רדודות יותר), לפיכך אפילו שיש בדינו לזהות את העמדה ע"י מפתח גיבוב חד-ערכי אין זה מספיק כשלעצמו. לפיכך, נזהה כל רשומה בטבלת גיבוב לא רק ע"פ מפתח הגיבוב שלה אלא גם ע"פ רמת העומק הממשית שלפיו בוצעה ההערכה בפועל. רמת עומק זו מחושבת כהפרש בין רמת העומק המקסימאלית לרמת העומק הנוכחית. השמירה בטבלת הגיבוב מבוצעת ע"י מבנה נתונים דינאמי המיוצג ע"י רשומות של היחס `pos_evaluation` –

`pos_evaluation(HashKey, ActualDepth, BestSuccessorPos, Val)`

כאשר לכל רשומה ביחס נשמרים כאמור מפתח הגיבוב ורמת העומק האקטואלית לצרכי אחזור, וכמובן, המסע העוקב הטוב ביותר (לפחות ע"פ פונקציית ההערכה) והערך שניתן בהערכה זו, שבהם אנחנו מעוניינים מלכתחילה. נבחין כי מהמצב ניתן גם לגזור את המהלך (קואורדינטה) שהובילה אליו.

- **תאימות וגמישות פונקציית ההערכה:** בכדי לשפר את התמיכה במספר דרגות הקושי במשחק, ולא להגביל אותה אך רק להבדל ברמות העומק, מועבר פרמטר נוסף לאלגוריתם אלפא-ביתא, Level, שבתורו מועבר לפונקציית ההערכה staticval בכדי שתוכל לקבוע בהתאם לרמת הקושי המבוקשת איך לחשב ועם איזה פונקציות לחשב את ההערכה של עמדה נתונה.

להלן סיכום הדמיון והשוני – לבן = דמיון, צהוב = שוני, ורדרד/כתום = הערות שלי –

```

alphabetalpha(CurrentPos,Alpha,Beta,BestSuccessorPos,Val,DepthLevel,MaxDepth,Level): -
    % check if results already exists on database
    ActualDepth is MaxDepth - DepthLevel,
    CurrentPos = pos(Grid,_,_),
    get_hash_key(Grid,HashKey),
    ((pos_evaluation(HashKey,ActualDepth,BestSuccessorPos,Val),!)
    ;
    % if not, calculate them from scratch
    (((nonvar(Alpha),nonvar(Beta)) ; (Alpha is -999999, Beta is 999999)), % init +-infinity
    moves(CurrentPos,PosList,DepthLevel,MaxDepth),!,
    NewDepthLevel is DepthLevel+1, % update depth level

boundedbest(PosList,Alpha,Beta,BestSuccessorPos,Val,NewDepthLevel,MaxDepth,Level),
    % save result to database for saving calculation runtime in the future
    assert(pos_evaluation(HashKey,ActualDepth,BestSuccessorPos,Val))
    ;
    staticval(CurrentPos,Val,Level)). % terminal seatchtree node - evaluate directly

boundedbest([Pos|PosList],Alpha,Beta,GoodPos,GoodVal,DepthLevel,MaxDepth,Level): -
    alphabetalpha(Pos,Alpha,Beta,_ ,Val,DepthLevel,MaxDepth,Level),
    goodenough(PosList,Alpha,Beta,Pos,Val,GoodPos,GoodVal,DepthLevel,MaxDepth,Level
    ).

goodenough([],_ ,_ ,Pos,Val,Pos,Val,_ ,_ ,_):- !.% No other candidate

goodenough(_ ,Alpha,Beta,Pos,Val,Pos,Val,_ ,_ ,_):-
    min_to_move(Pos),Val > Beta, !      % Maximizer attained upper bound
    ;
    max_to_move(Pos),Val < Alpha, !.    % Minimizer attained lower bound

goodenough(PosList,Alpha,Beta,Pos,Val,GoodPos,GoodVal,DepthLevel,MaxDepth,Level): -
    newbounds(Alpha,Beta,Pos,Val,NewAlpha,NewBeta), % Refine bounds
    boundedbest(PosList,NewAlpha,NewBeta,Pos1,Val1,DepthLevel,MaxDepth,Level),
    betterof(Pos,Val,Pos1,Val1,GoodPos,GoodVal).

/* define new interval by 2 last arguments that is narrower or equal to old interval */
newbounds(Alpha,Beta,Pos,Val,Val,Beta):-
    min_to_move(Pos),Val > Alpha, !.    % Maximizer increased lower bound

newbounds(Alpha,Beta,Pos,Val,Alpha,Val):-
    max_to_move(Pos),Val < Beta, !.    % Minimizer decreased upper bound

newbounds(Alpha,Beta,_ ,_ ,Alpha,Beta). % Otherwise bounds unchanged

% betterof(Pos,Val,Pos1,Val1,Pos,Val) % Pos better than Pos1
betterof(Pos,Val,_ ,Val1,Pos,Val):- % Pos better than Pos1
    min_to_move(Pos),Val > Val1, !
    ;
    max_to_move(Pos), Val < Val1, !.

betterof(_ ,_ ,Pos1,Val1,Pos1,Val1). % Otherwise Pos1 better

```

2.2.2.2 פונקציות היוריסטיות

בכדי להעריך עמדה נתונה במשחק אות'לו, מומשו שלושה פונקציות ההערכה בהשראת מאמר של סנידיהנאם [2] –

1. **ספירת כלים על הלוח** – בדיקה כמה כלים מהלוח שייכים לשחקן הראשון, כמה מהם לשחקן השני, והערכה ע"י חישוב היחס שבין ההפרש בין שחקן המקסימום ומינימום לבין סך הכלים של שניהם –

$$(1) \frac{\max_player_pieces - \min_player_pieces}{\max_player_pieces + \max_player_pieces}$$

פונקציה זו ממומשת ע"י הפרדיקט שלהלן –

`pieces_count_evaluation(GridId,Val,MaxCount,MinCount)`

ההערכה זו מחזירה תוצאה בין אפס לאחת. זוהי הערכה בסיסית יחסית, חמדנית בטיבה, ולכן תלויה במידה רבה ברמת העומק של עץ החיפוש, אחרת, החמדנות שלה עלולה להביא ליתרון בטווח הקצר אך הפסד בהמשך. לכן הוגדרו שתי פונקציות נוספות יציבות יותר לטווח הארוך.

2. **ניידות** – בדיקה כמה מהלכים חוקיים כל יריב יכול לבצע מהעמדה הנוכחית לו היה זה תור, וחישוב היחס שבין ההפרש לשה"כ בדומה לנוסחה (1). הרעיון מאחורי היוריסטיקה זו הוא להגביל את צעדי היריב במידת האפשר, ואולי אף להצליח להשיג להביא למצב שאין לו מסעים חוקיים, ואז נזכה לתורות נוספים עד אשר הוא יוכל לשחק או עד אשר ייגמר המשחק (המוקדם מביניהם). פונקציה ממומשת ע"י היחס –

`mobility_evaluation(Grid,Val)`

3. **פינות** – בדיקה כמה פינות מאוכלסות ע"י כל שחקן: מתן 0 לפינה שאינה מאוכלסת, 0.25 לפינה שמאוכלסת ע"י השחקן הראשון (שחקן המקסימום) ו-0.25 לפינה שמאוכלסת ע"י השחקן השני (שחקן המינימום), כלומר כל הפינות מתחלקות שווה (רבע-רבע-רבע-רבע). הפינות הן המשבצות היחידות ששומרות על צבען לאורך כל המשחק החל מהרגע שבו אוכלסו, ולכן מביא ליציבות יחסית לשחקן שכובש אותן. יתר על כן, יש להן שליטה על אלכסון ארוך שעשוי להיות בכל השפעה חזקה על שורות וטורים לאורך ורוחב כל הלוח. פונקציה זו ממומשת ע"י היחס –

`corners_evaluation(GridId,Val)`

לבסוף, כלל הפונקציות מנוהלות ונקראות מתוך הפונקציה המקורית להערכת עמדות טרמינליות – `staticval`. בהתאם לרמת הקושי, מתקבלת החלטה בכמה ואיזה פונקציות הערכה להשתמש, תוך מתן משקל יחסי ע"פ החלוקה הבאה ויצירת סכום ההערכה ממושקל כדלקמן –

פונקציה/רמה	מתחיל	בינוני	מתקדם
ספירת כלים	100%	40%	25%
ניידות	---	60%	35%
פינות	---	---	40%

למשל עבור שחקן ברמה מתקדמת נקבל –

$$staticval \leftarrow \sum_{i=1}^3 w_i f_i = 0.25 \times f(pieces_count) + 0.35 \times f(mobility) + 0.4 \times f(corners)$$

2.2.3 מודול ניהול המשחק וממשק המשתמש

2.2.3.1 פרדיקט התכנית הראשית

האפליקציה כולה מופעלת ע"י הרצת הפרדיקט של התכנית הראשית `run/0`. מתוך תכנית זו מבוצעת הסתעפות יחסי-משנה לאתחול והפעלת המשחק באחד משני מצבים (אינטראקטיבי או אוטומטי), שמהם נקראים לבסוף כל יתר האלגוריתמים, ולבסוף מוחזרת השליטה לתכנית הראשית, להדפסת תוצאות המשחק וניקוי "שאריות". עם הפרדיקט `cleanup` שדואג להפעיל את `retractall` על כל יחסי הזיכרון הדינאמי בכדי לנקות את כל חישובי הביניים, הלוחות וכו'.

2.2.3.2 פונקציות לעיבוד קלט משתמש

הקלט האינטראקטיבי מטופל ע"י הפרדיקט שלהלן –

`get_user_input(InputString)`

פרדיקט זה אחראי לעבד את השורה הבאה שהמשתמש מזין (עד ה-Enter), לגזור מקלט זה את המילה הראשונה (עד הרווח הראשון) ולהחזיר אותה בלבד, תוך התעלמות מיתר השורה. האפליקציה עוצבת כך שלא יידרש יותר ממילה אחת בכל הזנה, כך שאם בהזנה מסוימת הוזנה יותר ממילה אחת, ממילה אין צורך ביתר ואפשר להתעלם ממנו.

לטובת מימוש פרדיקט זה, ראשית מבוצע דילוג "אוטומטי" על רווחים, שורות ריקות ו-"רעש לבש מיותר" עם `get`, שמתעלם מתווים "שקופים" בהדפסה, ולאחר קריאת תו ראשון, מואצלת הסמכות לשגרת `parse_rest_of_line` עזר האחראית לפרסר אחר יתר השורה ולאסוף את כל התווים עד הרווח הראשון (או סוף שורה – המוקדם מבניהם) ולהחזיר אותם כרשימה, ובכל מקרה השורה נקראת בשלמותה ככדי לנקות את ה-Buffer עבור הקריאות הבאות.

על בסיס הפרדיקט `get_user_input` מושתתים כל הפרדיקטים בתכנית לאיסוף פרמטרים מהמשתמשים –

- איסוף פרמטר `n` כללי לקביעת סדר הלוח - `get_board_dimension`
- איסוף רמת קושי (Level) - `get_game_level`
- איסוף מצב משחק מבוקש - `get_game_mode`

שלושתם פועלים באופן דומה: איסוף קלט עם `get_user_input`, ביצוע ולידציה על תקינות הקלט, בדיקה אם המשתמש ביקש לעזוב את המשחק והמשך עיבוד בהתאם (חיווי על שגיאה ודרישת קלט מחודש / העברת הקלט הלאה / יציאה מהמשחק וכו'...).

החריגה בנוף שגרות איסוף הקלט היא השגרה האחראית על איסוף הקואורדינטות – `get_coordinate_from_user (+Player, +ValidCoordinates, -Coordinate)`

הקלט המצופה מהמשתמש הוא בפורמט `(I,J)`. להבדיל מיתר הפרמטרים שנאספו, קלט זה הוא לא מספרי (יש פסיק באמצע, ויש גם סוגריים), לכן לא ניתן לקלוט אותו איך שהוא בצורתו הגולמית. ניתן אולי היה לשנות את הפורמט כך שהקלט יוזן בצורה אחרת, למשל `I.J` אולם העדפתי שהקלט ייראה כמו מה שהוא מייצג (כפי שרכיב במטריצה מסומן במתמטיקה. לפיכך, הפורמט נשאר כך ובכדי לעבד אותו בוצע ירידה ופירוק לרמת התו הבודד והשוואתו מול ערך ה-Ascii שלו בשביל לפענח את הערך המקורי ולקבל החלטה בהתאם.

2.2.3.3 פרדיקטים לניהול המשחק

יש שני פרדיקטים נפרדים לניהול המשחק בהתאם למצב שנבחר ע"י המשתמש :
משחק אינטראקטיבי, או צפייה במשחק אוטומטי –

1. ניהול משחק אינטראקטיבי שחקן מול מחשב –

`play_interactive_game(Mode,Level,pos(GridId,Player1,_))`

2. ניהול משחק אוטומטי בין מחשב לבין עצמו –

`play_automatic_game(Level,pos(Grid1,Computer1((_,_)))`

בשניהם הרעיון הבסיסי הדומה : כל הפעלה של הפרדיקט שקולה לניהול מסע אחד במשחק, על כל הכרוך בכך : ביצוע תור אחד, הדפסת לוח עדכני, שמירת תוצאות ביניים וכו'. בשביל להעביר את השליטה על המשחק לשחקן הבא שיבצע הוא את תורו, יש להפעיל את היחס שוב עם העמדה העדכנית שכוללת בין היתר את המידע של מי התור הנוכחי לשחק. לפיכך שני הפרדיקטים שלעיל מבצעים מסע של שחקן אחד, מקבלים את המצב החדש המתקבל לאחר מכן, וקוראים לשגרה רקורסיבית עם העמדה העדכנית, וע"י כך בקריאה החדשה השליטה תועבר לשחקן השני לביצוע התור.

ההפרדה לשני יחסים נפרדים בוצע בכדי לפשט את הבקרה ולא לערבב בין המשחק האינטראקטיבי שכולל גם אינטראקציה וחיווי קלט/פלט שוטף מול השחקן האנושי, לעומת המשחק האוטומטי, שרק מריץ אלפא-ביתא על העמדה שוב ושוב עם הדפסת הלוח העדכני בין הרצה להרצה.

יחסים אלו משתמשים באלגוריתמים שנדונו קודם לכן לבדיקת תקינות קלט, חישוב קואורדינטות חוקיות, היפוך צבעי כלים על הלוח, הדפסת הלוח וכד'. באותו לוח, אם לשחקן אחד אין מסע חוקי, המשחק אינו מסתיים, אלא עובר לשחקן השני. לפיכך לזיהוי מצב כזה ומניעת מצב שכאשר שני השחקנים תקועים התור עובר בניהם לסירוגין עד אינסוף (או עד אשר ה-CPU יאבד שליטה מצריכת זיכרון גבוהה...), מנוהלים דגלי עזר לשמירת סטטוסים –

- אחד השחקנים תקוע ללא מסעים חוקיים : `player_stuck/1`
- שני השחקנים תקועים ללא מסעים חוקיים : `no_legal_move/0`
- המשחק הגיע לסיומו : `end_of_game/1`
- משתמש ביקש לעזוב באמצע המשחק : `user_exited_game/0`

דגלים אלו מודלקים ונכבים ע"פ הצורך מתוך שגרות ניהול המשחק. בשגרת השחקן הבודד התקוע מועבר גם מידע מיהו השחקן התקוע, ובשגרת סיום המשחק מועבר מידע על עמדת הסיום בכדי שניתן יהיה לבצע ניתוח מצב, לגזור סטטיסטיקה ולקבוע מי ניצח/הפסיד או אם יש תיקו.

2.3 סיכום מבני נתונים ופרדיקטים מרכזיים

מודול/שיוך	תפקיד/אחריות הפרדיקט	פרדיקט
מבני נתונים וייצוג (לוגיקה פנימית של המשחק בהתאם לאופן הייצוג ומבני הנתונים שנבחרו)	ייצוג משבצת בודדת על לוח : מזהה לוח, קואורדינטה, ערך מאוכלס על המשבצת : $0 = \text{ריק}, x = 1, 2$	<i>slot(Id, coordinate(I,J), Val)</i>
	ייצוג עמדה נתונה במשחק	<i>pos(GridId, Turn, coordinate(I,J))</i>
	אחזור ממד סדר לוח (משתנה גלובאלי)	<i>get_board_dimension</i>
	אינדקס עוקב סריקה עבור סדרתית	<i>get_next_sequential_index</i>
	החזרת אינדקס עוקב בכיוון ספציפי	<i>get_next_directional_index</i>
	אתחול לוח ראשוני	<i>initialize_starting_pos_grid</i>
	שכפול לוח	<i>duplicate_grid</i>
	ניהול נומרטור למספרי לוחות	<i>next_idle_grid_id</i>
	בדיקת תקינות על הנחת חייל על משבצת נתונה	<i>validate_coordinate</i>
	החזרת רשימת כל הקואורדינטות החוקיות לביצוע במסע הנוכחי	<i>get_legal_coordinates</i>
	היפוך צבעים חיילים לאחר מסע	<i>flip_pieces</i>
	ביצוע מסע מבוקש, החזרת עמדה מעודכנת לאחר הביצוע	<i>makeLegalMove</i>
	שמירת הערכת עמדה שחושבה באלפא-בתא בטבלת גיבוב	<i>pos_evaluation</i>
	היוריסטיקה להערכה ע"י השוואה בין כמות הכלים על הלוח	<i>pieces_count_evaluation</i>
אלפא-בטא	היוריסטיקה להערכה ע"י השוואת מספר מהלכים אפשריים לכל שחקן	<i>mobility_evaluation</i>
	היוריסטיקה להערכה ע"י השוואת מספר פינות שנלכדו ע"י כל שחקן	<i>corners_evaluation</i>
	התנעת אפליקציה	<i>run</i>
	התנעת משחק אינטראקטיבי שחקן מול מחשב	<i>play_interactive_game</i>
	התנעת משחק אוטומטי מחשב מול מחשב	<i>play_automatic_game</i>
	אחזור רמת קושי מבוקשת	<i>get_game_level</i>
	אחזור מצב משחק	<i>get_game_mode</i>
	אחזור קואורדינטה מיועדת לתור	<i>get_coordinate_from_user</i>

3 ממשק המשתמש

3.1 התנעת האפליקציה

האפליקציה כולה מופעלת ע"י הרצת הפרדיקט של התכנית הראשית `run/0` :

```
SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- run.
Welcome to this super cool prolog othello game application!
What is your name? Skywalker
```

3.2 בהירות ונוחות הממשק

מיד לאחר התנעת האפליקציה, התכנית מדפיסה למשתמש **הסבר ברור** בכל שלב ושלב עד לסוף המשחק בדיוק מה מצופה ממנו כקלט, מה שיחק היריב (המחשב), מה המצב הנוכחי של המשחק, ומה הוא צריך להזין אם ברצונו לצאת מהמשחק באמצע. יתר על כן, בכדי להקל עליו להזין קואורדינטות, הלוח מודפס עם מסגרת אינדקסים מתאימה משמאל ומלעיל הלוח, והמהלכים החוקיים האפשריים מוכתבים לו מלכתחילה מתחת ללוח, לדוגמה –

```
SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl
File Edit Settings Run Debug Help
3  [ ] [ ] [ ] [ ] [x] [ ] [ ] [ ]
4  [ ] [ ] [ ] [x] [x] [ ] [ ] [ ]
5  [ ] [ ] [ ] [x] [o] [o] [ ] [ ]
6  [ ] [ ] [ ] [x] [ ] [ ] [ ] [ ]
7  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
8  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

Wow, you are pretty good

My artificial intelligence guides me to play (5,3).
current position after placing a piece on this slot -
  [1] [2] [3] [4] [5] [6] [7] [8]
1  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
2  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
3  [ ] [ ] [ ] [ ] [x] [ ] [ ] [ ]
4  [ ] [ ] [ ] [x] [x] [ ] [ ] [ ]
5  [ ] [ ] [o] [o] [o] [o] [ ] [ ]
6  [ ] [ ] [ ] [x] [ ] [ ] [ ] [ ]
7  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
8  [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]

It's your turn to move.
You need to place an "x" on the board
Enter the requested slot in the format (I,J),
where I & J stand for the row & column coordinate indices, respectively.
The possible options are: [(4,2),(4,6),(6,2),(6,3),(6,5),(6,6),(6,7)]
If you feel like cutting early, just type "exit" or "EXIT"
|:
```

יתר על, הממשק לא רק ברור, אלא גם קריא. כדי לוודא שלמשתמש יהיה זמן לקרוא מבלי להצטרך לגלול את המסך אחורנית, מוכנסות מעת לעת שהיות קצרצרות יזומות ע"י היחס ¹sleep/1 המקבל כארגומנט מספר ממשי המייצג כמה שניות יש להשהות את הריצה.

הממשק מאפשר נקודת יציאה בכל נקודה אינטראקטיבית, דהיינו בכל נקודה שבה מצפים לקלט. פקודות היציאה הן exit או EXIT (יש שתיים מטעמי נוחות למשתמש). לאחר יציאה, התכנית דואגת לנקות אחריה את כל הלוחות בזיכרון ויתר שאריות כך שניתן להתחיל משחק מחדש ללא חשש. התכנית גם דואגת לברך השחקן לשלום לאחר יציאתו, למשל –

```
5  [ ][ ][o][o][o][o][ ][ ]
6  [ ][ ][x][ ][ ][ ][ ]
7  [ ][ ][ ][ ][ ][ ][ ]
8  [ ][ ][ ][ ][ ][ ][ ]
```

It's your turn to move.

You need to place an "x" on the board

Enter the requested slot in the format (I,J),

where I & J stand for the row & column coordinate indices, respectively.

The possible options are: [(4,2),(4,6),(6,2),(6,3),(6,5),(6,6),(6,7)]

If you feel like cutting early, just type "exit" or "EXIT"

|: exit

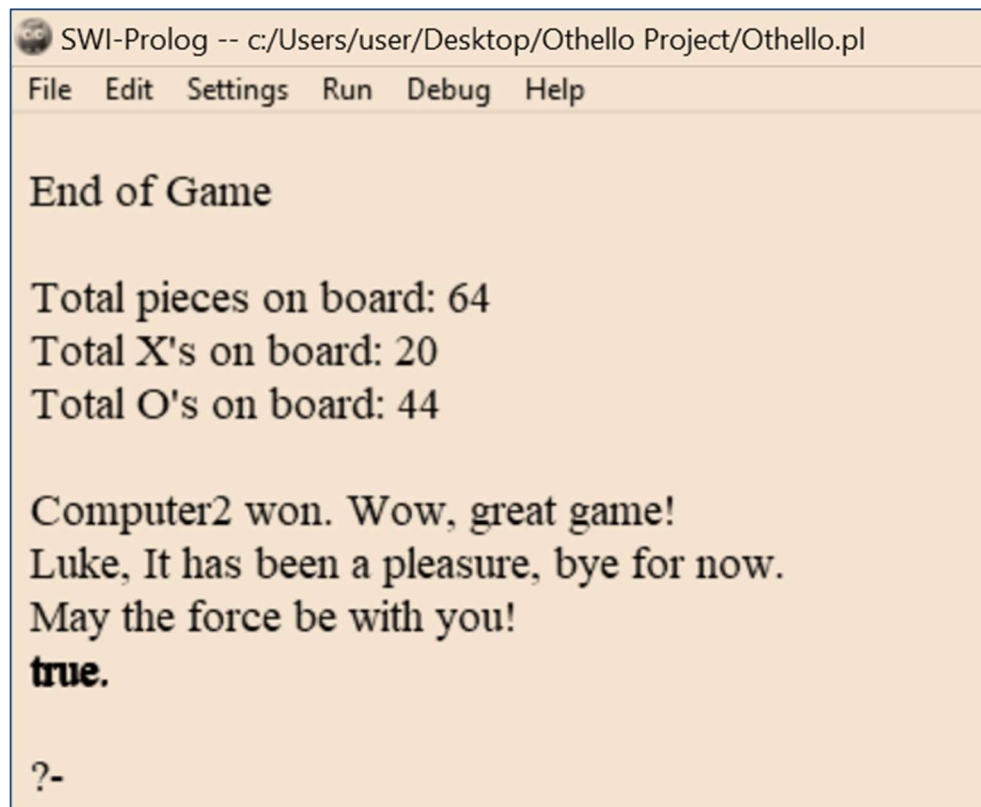
Skywalker, It has been a pleasure, bye for now.

May the force be with you!

true.

?- |

- בסיום של משחק שלם מוצגת התוצאה הסופית –



```
SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl
File Edit Settings Run Debug Help

End of Game

Total pieces on board: 64
Total X's on board: 20
Total O's on board: 44

Computer2 won. Wow, great game!
Luke, It has been a pleasure, bye for now.
May the force be with you!
true.

?-
```

¹ יחס זה אמנם אולי אינו נמצא בספר [2], אך הוא משמש כאן לטובת ממשק המשתמש הגראפי כספרייה חיצונית.

3.3 טיפול בשגיאות

התכנית אינה מניח שהקלט שמגיע מהמשתמש תקין, ודואגת לוודא זאת לאחר כל הזנה. במקרה של קלט שגוי, התכנית מדפיס למשתמש חיווי ברור על כך, ומבקשת ממנו לנסות שנית, תוך הסבר ברור של פורמט הקלט המצופה ממנו. דוגמאות –

```
Okay. Let's set the game's level -
Please enter a number between 1 to 3 as follows:
1 = Beginner
2 = Intermediate
3 = Advanced
|: 4
```

Sorry, the input "4" is invalid. Let's try again -

```
Please enter a number between 1 to 3 as follows:
1 = Beginner
2 = Intermediate
3 = Advanced
|: 3
```

SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl

File Edit Settings Run Debug Help

```
[1] [2] [3] [4]
1  [ ] [ ] [x] [o]
2  [ ] [x] [x] [o]
3  [ ] [o] [o] [o]
4  [ ] [ ] [ ] [ ]
```

It's your turn to move.

You need to place an "x" on the board

Enter the requested slot in the format (I,J),

where I & J stand for the row & column coordinate indices, respectively.

The possible options are: [(4,1),(4,2),(4,3),(4,4)]

If you feel like cutting early, just type "exit" or "EXIT"

```
|: (1,1)
```

Sorry, the input "(1,1)" is invalid. Let's try again -

Enter the requested slot in the format (I,J),

where I & J stand for the row & column coordinate indices, respectively.

The possible options are: [(4,1),(4,2),(4,3),(4,4)]

If you feel like cutting early, just type "exit" or "EXIT"

```
|: 3-CPO
```

Sorry, the input "3-CPO" is invalid. Let's try again -

Enter the requested slot in the format (I,J),

where I & J stand for the row & column coordinate indices, respectively.

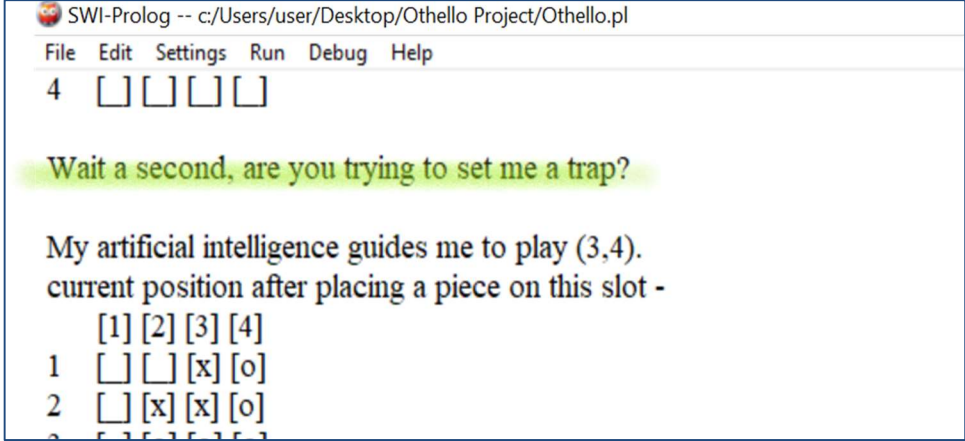
The possible options are: [(4,1),(4,2),(4,3),(4,4)]

If you feel like cutting early, just type "exit" or "EXIT"

```
|:
```

3.4 שיפורים ותוספות

- המימוש הוא ללוח מסדר n כללי.
- בכדי להוסיף קצת הומור למשחק, לאחר כל מסע של המשתמש המחשב מדפיס מחמאה באופן אקראי מסט מוכן ראש של מחמאות, למשל –



```

SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl
File Edit Settings Run Debug Help
4  [ ] [ ] [ ] [ ]

Wait a second, are you trying to set me a trap?

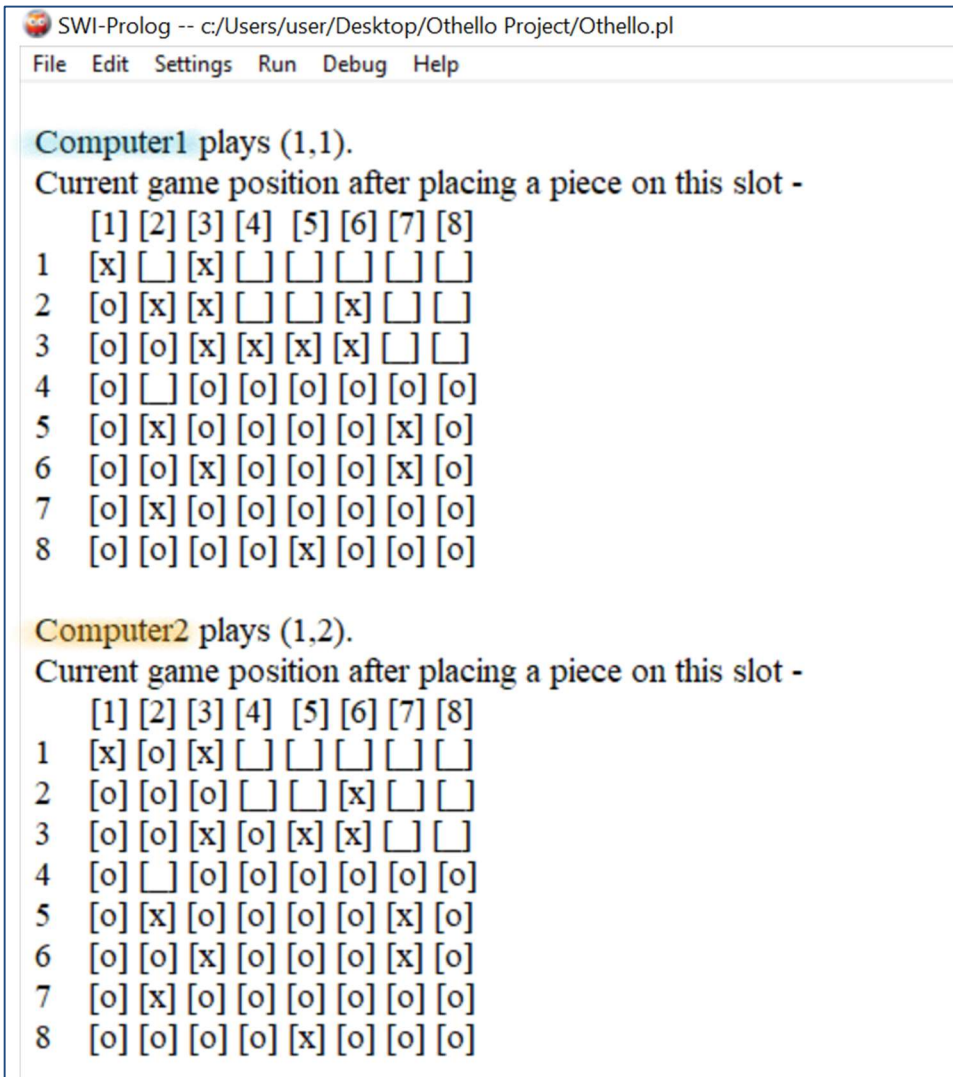
My artificial intelligence guides me to play (3,4).
current position after placing a piece on this slot -
  [1] [2] [3] [4]
1  [ ] [ ] [x] [o]
2  [ ] [x] [x] [o]
3  [ ] [ ] [ ] [ ]
4  [ ] [ ] [ ] [ ]
5  [ ] [ ] [ ] [ ]
6  [ ] [ ] [ ] [ ]
7  [ ] [ ] [ ] [ ]
8  [ ] [ ] [ ] [ ]

Computer1 plays (1,1).
Current game position after placing a piece on this slot -
  [1] [2] [3] [4] [5] [6] [7] [8]
1  [x] [ ] [x] [ ] [ ] [ ] [ ] [ ]
2  [o] [x] [x] [ ] [ ] [x] [ ] [ ]
3  [o] [o] [x] [x] [x] [x] [ ] [ ]
4  [o] [ ] [o] [o] [o] [o] [o] [o]
5  [o] [x] [o] [o] [o] [o] [x] [o]
6  [o] [o] [x] [o] [o] [o] [x] [o]
7  [o] [x] [o] [o] [o] [o] [o] [o]
8  [o] [o] [o] [o] [x] [o] [o] [o]

Computer2 plays (1,2).
Current game position after placing a piece on this slot -
  [1] [2] [3] [4] [5] [6] [7] [8]
1  [x] [o] [x] [ ] [ ] [ ] [ ] [ ]
2  [o] [o] [o] [ ] [ ] [x] [ ] [ ]
3  [o] [o] [x] [o] [x] [x] [ ] [ ]
4  [o] [ ] [o] [o] [o] [o] [o] [o]
5  [o] [x] [o] [o] [o] [o] [x] [o]
6  [o] [o] [x] [o] [o] [o] [x] [o]
7  [o] [x] [o] [o] [o] [o] [o] [o]
8  [o] [o] [o] [o] [x] [o] [o] [o]

```

- קיימת אפשר לצפות במחשב משחק נגד עצמו! למשל –



```

SWI-Prolog -- c:/Users/user/Desktop/Othello Project/Othello.pl
File Edit Settings Run Debug Help

Computer1 plays (1,1).
Current game position after placing a piece on this slot -
  [1] [2] [3] [4] [5] [6] [7] [8]
1  [x] [ ] [x] [ ] [ ] [ ] [ ] [ ]
2  [o] [x] [x] [ ] [ ] [x] [ ] [ ]
3  [o] [o] [x] [x] [x] [x] [ ] [ ]
4  [o] [ ] [o] [o] [o] [o] [o] [o]
5  [o] [x] [o] [o] [o] [o] [x] [o]
6  [o] [o] [x] [o] [o] [o] [x] [o]
7  [o] [x] [o] [o] [o] [o] [o] [o]
8  [o] [o] [o] [o] [x] [o] [o] [o]

Computer2 plays (1,2).
Current game position after placing a piece on this slot -
  [1] [2] [3] [4] [5] [6] [7] [8]
1  [x] [o] [x] [ ] [ ] [ ] [ ] [ ]
2  [o] [o] [o] [ ] [ ] [x] [ ] [ ]
3  [o] [o] [x] [o] [x] [x] [ ] [ ]
4  [o] [ ] [o] [o] [o] [o] [o] [o]
5  [o] [x] [o] [o] [o] [o] [x] [o]
6  [o] [o] [x] [o] [o] [o] [x] [o]
7  [o] [x] [o] [o] [o] [o] [o] [o]
8  [o] [o] [o] [o] [x] [o] [o] [o]

```


4 רשימה ביבליוגרפית

- [1] Bratko, I. *[Prolog Programming for Artificial Intelligence](#)*. Pearson, Canada, 2011.
- [2] Sannidhanam, V. and Muthukaruppan, A. "An Analysis of Heuristics in Othello", 2015. Retrieved November 12, 2019, from The University of Washington: https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf