ספר פרויקט – שחמט

מכללה: רב תחומי עמל ב

שם פרויקט: שחמט

שם מלא: מהדב איתי

ת.ז: 215300807

שם מנחה: נילי נווה

תאריך הגשה:

תוכן עניינים

1. תקציר

ספר פרויקט זה עוסק בבעיית מציאת המהלך הטוב בשחמט ומציע פתרון באמצעות ניתוח המשחק והגדרת המצבים כמו ששחקן בשר ודם עושה. ספר זה מתעמק בהיסטורייה של משחק השחמט, מקורו ובנוסף אסטרטגיות ואלגוריתמים אשר מאחורי מנועי השחמט ברמות הכי גבוהות ביותר.

לאחר מכן יפורט הפתרון המוצא, את הדפוסים שיש במשחק השחמט היכולים לעזור לנו לגרום למכונה לבצע מהלכים אינטלגנטיים כמו בני אדם.

היו הרואים את משחק השחמט כאל משחק שבו נדרשת אינטואיציה גבוהה וזיהוי דפוסים ולכן מכונה לא יכולה לנצח בן אדם, אך כיום אנו יודעים שזה לא נכון והמכונות לא רק הצליחו לבצע מהלכים אינטלגנטיים ולזהות דפוסים הן גם מצליחות לנצח כל שחמטאי גם אם הוא אלוף העולם בקלות רבה.

יש לציין כי כעת משחק השחמט אינו פתור ואי אפשר לכפות תיקו או מט ממצב התחלתי של משחק שחמט, ולכן זוהי מהווה בעיה אלגוריתמית שספר זה מנסה לפתור.

2. תיאור הנושא

הנושא הנבחר הוא משחק השחמט, משחק לוח המשוחק על לוח בעל 64 משבצות.

לכל שחקן ישנם 16 כלים בעלי 6 סוגים שונים. כל סוג כלי יכול לזוז בהתאם לכללי התזוזה שלו.

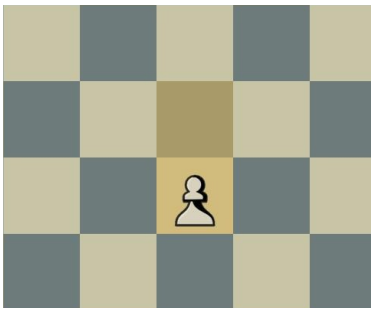
מטרתו של כל שחקן להזיז את כליו באופן שיגיע למט על היריב מבלי שיבצע מט עליו.

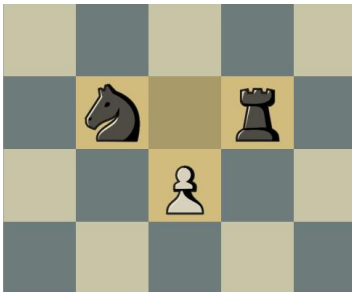
שחמט הוא משחק חשיבה וספורט תחרותי בין שני אנשים או יותר.

כלי המשחק

- חייל

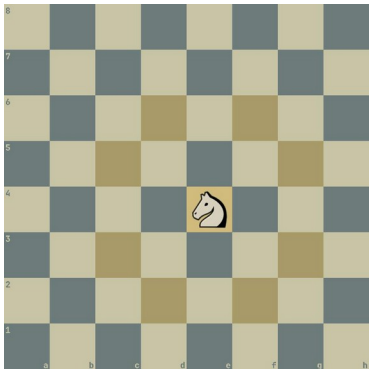
החייל יכול לזוז משבצת אחת קדימה בכל תור, למעט אם הוא עדיין לא זז ובמקרה כזה הוא יכול לזוז עד 2 משבצות קדימה. החייל יכול לאכול כל כלי הנמצא מלפניו באלכסון במרחק של משבצת אחת.





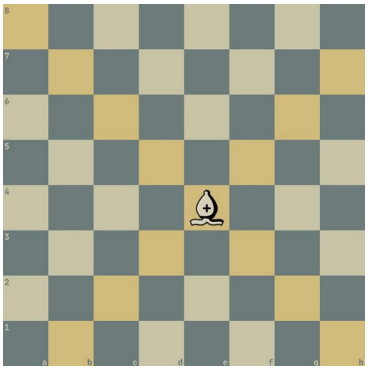
- פרש

הפרש יכול לזוז שתי משבצות בקו ישר ולאחר מכן משבצת אחד בפניה והוא השחקן היחיד שיכול לדלג מעל כלים אחרים. יכולת האכילה זהה ליכולת התזוזה.



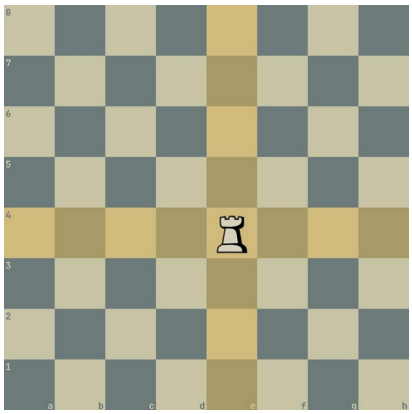
- רץ

הרץ זז באלכסונים לכל כיוון ואינו יכול לדלג על אף כלי. יכולת האכילה זהה.



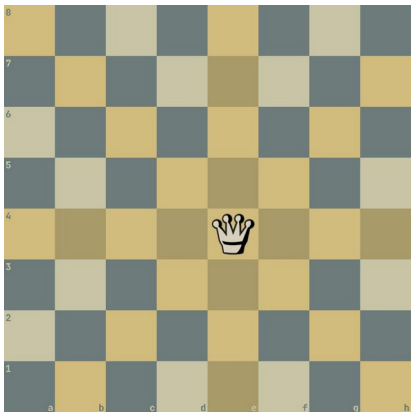
- צריח

הצריח נע בקווים ישרים לכל הכיוונים ואינו יכול לדלג על אף כלי. יכולת האכילה זהה.



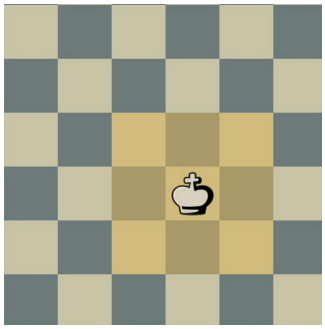
- מלכה

המלכה יכול לזוז או באלכסון לכל כיוון או בקו ישר לכל כיוון. אפשר להתייחס למלכה כאל חיבור של צריח ורץ. המלכה אינה יכולה לדלג על אף כלי .יכולת האכילה זהה.

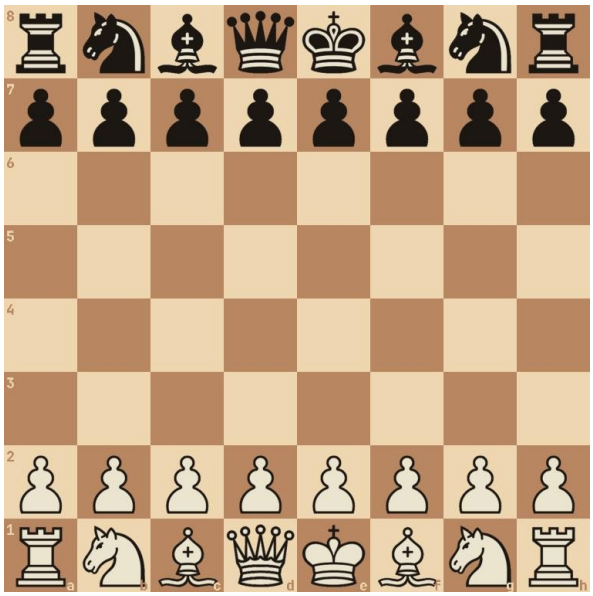


- מלך

המלך יכול לזוז משבצת אחת לכל הכיוונים כל עוד היא לא מאוימת ע"י השחקן השני. יכולת האכילה זהה.



מצב התחלתי של הלוח:



הfen של המצב ההתחלתי נראה כך:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w

כאשר:

- '/' מייצג סוף שורה בלוח
- כל אות מייצגת כלי וצבעו (אות גדולה כלי לבן, אות קטנה כלי שחור)
- מספר מייצג את כמות המשבצות הריקות מאותו מיקום
- w כאשר תורו של השחקן הלבן, b כאשר תורו של השחקן השחור

לfen הרשמי חלקים נוספים כמו האם שחקן יכול להצריח, ולחוק "הכאה אגב הלוכו" שמתאר אכילה מיוחדת בין שני כלי חייל.

3. רקע תיאורטי

ככל הנראה שחמט מתבסס על משחק לוח ההודי צטורגנה אך אינינו בטוחים. המשחק התפשט באזור המאה השביעית ורק לאחר מכן במאה ה15 חוקי השחמט המוכרים לנו כיום הומצאו.

לאחר שהשחמט שאנו מכירים היום התפשט, שחקנים החליטו לארגן טורנירים שבהם שחקן אחד מתחרה עם שחקן אחר ומטרתו של כל אחד הוא לנצח. כיום אנו קוראים לשחקן שחמט מקצועי "שחמטאי". יש לציין כי השחמט המקצועי משוחק לצד שעון כך כשלכל שחקן יש זמן מוגבל לחשוב ולבצע מהלך. השחמט נחשב למשחק חשיבה מורכב שעל השחקנים להסתכל כמה צעדים קדימה כדי להחשיל את האויב ולנצח. שם המשחק נקרא ע"י כיתור המלך - שח וע"י כיתור המלך שאין לו אפשרות לבצע לברוח – "מט". ברגע ששחקן מבצע שחמט הוא מנצח.

לשחמט ומדעי המחשב ישנה היסטוריה מעניינת, ב1948 אלן טיורינג ושמפרנאון יצרו את תכנית השחמט "*Turochamp*" המסוגלת לנצח בני אדם ברמה נמוכה ע"י חישוב האפשרויות ובחירת מהלך בהתאם, אלגוריתם זה נקרא minimax והוא משמש בלמציאת המהלך הטוב ביותר במגבלות המכונה. מאוחר יותר בשנות התשעים פותח"Deep Blue" מחשב מיוחד ע"י IBM כשמטרתו לשחק שחמט ברמה הגבוהה ביותר, המחשב ניצח בשני משחקים מתוך שישה נגד "גרי קספרוב", שחמטאי ברמה עולמית, וכעבור שנה פותח המחשב וניצח את גרי בשני משחקים מתוך שלושה כשאר במשחק השלישי היה תיקו. "Deep Blue" הוא המחשב הראשון שהביס שחמטאי ברמה עולמית.

כיום "stock fish" נחשב למנוע השחמט החזק ביותר.חלק מחשיבים את "alpha zero" שפותח ע"י גוגל למנוע שחמט הממוחשב החזק ביותר כאשר הוא ניצח את stockfish, אבל יש הטוענים שמכיוון שalpha zero מבוסס על למידת מכונה שיושבת על מחשב על בעוד שstockfish היא תוכנה שניתן להריץ על כמעט כל מחשב הנמצא בשוק זהו לא קרב הוגן. יש לציין כי ל stockfish יש לא מעט גרסאות שמחולקות לפי תוכנה לוקאלית (על המחשב) ותוכנה בענן (על שרת), כאשר לתוכנה בענן היכולת לשחק שחמט ברמה גבוהה יותר כי יש לה גישה ליותר כוח עיבוד וגישה לנתונים רבים. למשל כשאר ישנם שבעה כלים על הלוח (לא משנה איזה סוג הכלים) לstockfish יש בסיס נתונים בעל 20 טרה המכיל את כל האפשרויות של המשחק ובמצב זה הוא יכול לנצח ב100 אחוז מהפעמים.

4. מושגים

לוח המשחק 8*8 – מערך של מאסקים

כלי - ביט במאסק

Fen – מחרוזת המתארת את מצב הלוח ואת תורו של השחקן הנוכחי

פעולות:

מהלך - הזזה של כלי ממשבצת מקור למשבצת יעד בהתאם לכללי התזוזה של הכלי. כאשר במשבצת היעד אין כלי בעל אותו הצבע של הכלי המוזז – מהלך ייוצג ע"י ביט מקור וביט יעד.

כלי מחליק - כלי הנע אינסוף משבצות לכיוונים מסוימים עד לסוף הלוח או עד למשבצת של כלי יריב או עד משבצת לפני של כלי לא יריב, הכלים המחליקים הם צריח, רץ ומלכה.

חוסם - כלי הנמצא במשבצת שמגבילה את יכולת התזוזה של **כלי מחליק**

אכילה - **מהלך** הנגמר בהוצאת כלי מהמשחק.

קידום - **מהלך** הנגמר בהחלפת חייל למלכה כאשר משבצת היעד היא בשורה האחרונה של הלוח בהתאם לצד של כל שחקן

מצבים:

איום – מצב כאשר כלי בעל היכולת **לאכול** כלי אחר **במהלך** אחד.

שח – **איום** על המלך. במצב זה לשחקן בעל המלך המאוים מותרים רק מהלכים המסירים את האיום.

מט – **מהלך** הגורם ל**שח** ובלתי אפשרי להסירו לאחר **מהלך** אחד.

תיקו – כאשר אין **שח** ואין **מהלכים** חוקיים לאחד מהשחקנים או כאשר אין כלים על הלוח מלבד שני המלכים.

מזלג – שני **איומים** או יותר המתבצעים ע"י כלי אחד.

הצמדה – **איום** על **חוסם** שכאשר יזוז יחשף **חוסם** נוסף.

תמיכה – כאשר כלי יכול להגיע במהלך אחד למיקום של כלי אחר מאותו הצבע.

פיתוח – מהלך הגורם לכלי לעלות את **דרגת ההשקפה** שלו.

דרגת השקפה – מספר **המהלכים** שכלי יכול לבצע ברגע נתון

החלפה – מהלך או יותר הנגמר בהורדת סכום משקל כלים שווה בין שני השחקנים.

5. תיאור הבעיה האלגוריתמית

Model

- ייצוג הלוח
- ביצוע מהלך
- בדיקה האם מהלך חוקי
- זיהוי קידום
- ביצוע קידום
- זיהוי מצב התחלה
- זיהוי שח
- זיהוי מט
- זיהוי תיקו
- זיהוי מצב הגנה
- זיהוי מצב התקפה
- זיהוי מצב פיתוח
- זיהוי מצב מזלג
- זיהוי מצב הצמדה
- זיהוי איום
- זיהוי כלי מחליק
- מיפוי הלוח למצבים ותיעדופם
- תגובה למצב התקפה
- תגובה למצב הגנה
- תגובה למצב התקפה
- תגובה למצב פיתוח
- תגובה למצב מזלג
- תגובה למצב הצמדה
- תגובה למצב איום
- אתחול הלוח לפי fen
- אתחול הלוח לפי מצב התחלתי
- חישוב מהלך אופטימלי

viewer

- ייצוג הלוח
- קליטת מהלך מהשחקן
- ביצוע מהלך
- ביצוע קידום
- קליטת fen מהמשתמש
- בדיקה האם fen חוקי
- אתחול הלוח לפי fen

6. סקירת אלגוריתמים בתחום הבעיה

ישנם מספר אלגוריתמים ושיטות לבניית מנוע שחמט. מנוע שחמט טוב מנצל את כל האלגוריתמים האלה ומגיע להחלטה הכי נכונה ככל האפשר. אינו משתמש באלגוריתמים האלה

יעילות

אלגוריתם minimax:

המטרה במשחק השחמט היא למצוא את המהלך היקרב את השחקן לניצחון ככל שניתן. באלגוריתם זה אנו בונים עץ אפשרויות המכיל את כל האפשריות של הלוח כאשר כל רמה בעץ היא חצי מהלך (מהלך של שחקן אחד). לאחר מכן מבצעים ניתוח של הלוח ובוחרים במהלך שהוביל ללוח בעל סיכוי הכי גבוה לניצחון בשבילנו בהנחה שהשחקן היריב יבחר במהלך הכי טוב לטובתו. ככל שגובה העץ (עומק החיפוש) יגדלו, האלגוריתם ימצא מהלכים טובים יותר או יוודא שהמהלך שבחר הוא מהלך טוב.

Alpha beta pruning:

אלפה בטא פרונינג הינו תוספת לאלגוריתם המינימקס שבא למטב את החיפוש כך שנבדוק פחות מהלכים עבור אותו עומק או עבור אותו מספר מהלכים נגיע לעומק גדול יותר. כדי לצמצם את האפשרויות האלגוריתם גוזם את תתי העץ שלא יובילו למהלך טוב ובכרממטב את החיפוש ומעלה את גובה העץ עבור אותה יכולת עיבוד.

Algorithm Based-Heuristic:

אלגוריתם זה מעריך את לוח השחמט ופולט לאיזה שחקן יש יתרון ובכמה. אלגוריתם זה הוא קריטי בשיטת הminimax כדי להכריע האם המהלך הוא הוסיף לנו יתרון או הוריד. אלגוריתם זה עושה זאת ע"י ספירת הכלים על הלוח כפול משקל של כל כלי. משקל של מלכה יהיה יותר ממשקל של חייל. בנוסף האלגוריתם מנתח את הפוזיציה של כל כלי ונותן להם ערך. למשל המלך יקבל ניקוד גבוה אם הוא יהיה בקצה של הלוח שם הוא יותר מוגן (אלא אם זה סוף משחק ואז המלך יקבל ניקוד גבוה כאשר הוא קרוב לאמצע הלוח), הפרש יקבל ניקוד גבוה כאשר הוא באמצע הלוח כי אז יש לו דרגת השקפה גבוהה יותר וכך הלאה. ככל שהאלגוריתם של הערכה יהיה יותר מדויק, אנו נוכל להסתפק בגובה נמוך יותר של עץ האפשרויות. ברמה תיאורטית אם הפונקציה מושלמת אנו נוכל לחפש בגובה עץ 1 ולא יותר כדי למצוא את המהלך הכי טוב.

Monte Carlo Tree Search:

בדומה לאלגוריתם מינימקס גם במונטי קרלו אנו בונים עץ אפשרויות אך טיפה שונה. במקום לבצע brute force לכל האפשרויות כדי למצוא את המהלך הטוב ביותר אנו נשתמש בהסתברויות. תחילה נבנה עץ אפשרויות בעומק 1, לאחר מכן עבור כל בן נבצע סימולציית משחק רנדומלי שלם ונוסיף את תוצאת המשחק לצומת. האלגוריתם יבחר איזה מהלכים באים לחקור ע"פ נוסחה מתמטית בין הערך שבכל צומת לבין מספר הפעמים שחקרנו אותו ואז נבצע את אותו תהליך עד מספר חזרות מסויים. בסוף האלגוריתם נבחר את המהלך שבעל שילוב של הערך הכי גבוה וכמות הסימולציות שהורצו עבור תת העץ של אותו מהלך (ככל שחקרנו צומת יותר והסתברות הניצחון שלו גבוהה אז הוא כנראה מהלך טוב).

יש לציין שאלגוריתם זה אינו זקוק לפונקציה המעריכה את הלוח והוא מתאים למשחקים שבהם יש קושי לכתוב פונקציה המעריכה את הלוח כמו שחמט.

בסיס נתונים:

שמירת מצבי לוח והמהלך הנכון ביותר למצב בבסיס נתונים, ושליפת המהלך מבסיס הנתונים לפי מצב הלוח במהלך המשחק.

כיוון שיש יותר מדי מצבי לוח, בלתי אפשרי לשמור את כולם. נהוג לשמור מצבי פתיחה ומצבי סוף. כאשר מתחילת המשחק ועד ל5 מהלכים, ומ5 כלים על הלוח עד לסיום המשחק בהתאמה.

יתרונות: בחירת המהלכים הכי נכונים בדיוק של 100% בסיבוכיות לינארית התלויה בגודל בסיס הנתונים.

חסרונות: אלגוריתם זה פותר רק את מצב הפתיחה ומצב הסוף.

מכונת מצבים:

בשיטה זו אנו ננתח את משחק השחמט, נמצא אסטרטגיות שמעלות את הסיכוי לניצחון ונבחר את האחת המתאימה לפי המצב בלוח, ישנו דמיון בין שיטה זו לבין האופן שבו שחמטאי בוחר מהלך, הרי שחמטאי אינו מחשב בראש עץ אפשרויות ואין לו מבנה נתונים בגודל של נאסא בגישה של (1)o. הוא מזהה דפוסים על הלוח ולפיכך מגיע להכרעה.

כדי לממש את השיטה הזו, יש לנתח את משחק השחמט ולמצוא דפוסי משחק והתגובה המתאימה אליהם.

דפוסים בשחמט לדוגמה: מזלג, הצמדה, שח, מט, תמיכה בכלי, פיתוח, החלפה, התקפה, הגנה.

7. מבנה נתונים

גרף:

אפשר לייצג את הלוח ע"י גרף מכוון, כך שכל משבצת בלוח היא קודקוד, לכל קודקוד יהיו שכנים כך שכל שכן הוא נמצא במרחק של מהלך אחד ממנו לפי חוקי התזוזה של כל כלי. זאת אומרת שיהיה סוג לקשת, בין אם היא קשת של מלכה או קשת של חייל.

יעילות המבנה לצורך בדיקה האם מהלך חוקי:

בדיקה יכולת התזוזה: O(V) בממוצע עבור כלים מחליקים, ו(O(1 עבור כלי לא מחליק. בהנחה שבדיקה האם שני קודקודים שכנים היא ב(O(1.

בדיקת האם שח: כמות השכנים לקודקוד * בדיקת יכולת התזוזה = O(V*E).

בדיקת מהלך חוקי כוללת בדיקה האם הכלי יכול להגיע למשבצת היעד, האם אין אף כלי בדרך לשם אם זה כלי מחליק, והאם השחקן אינו גורם לאיום על המלך שלו.

הגדרה בסביבת העבודה:

```
typedef char Squares;

typedef struct {
	Squares vertexs[8][8];
	int rook_edges[8][8];
	int bishop_edges[8][8];
	int pawn_edges[8][8];
	int king_edges[8][8];
	int knight_edges[8][8];
} Board;
```

Bitboard:

ייצוג הלוח ע"י 15 מספרים בעלי 64 סיביות. כאשר כל מספר מתאר את הלוח 'מימד' אחר של הלוח בעזרת 1 המייצג תפוסה, ו0 שמייצג אין תפוסה.

- חיילים לבנים/שחורים

- פרשים לבנים/שחורים

- רצים לבנים/שחורים

- צריחים לבנים/שחורים

- מלכות לבנות/שחורים

- מלך לבן/שחור

- כל הכלים של לבן/שחור

- כל הכלים על הלוח

בדיקת האם כלי יכול לזוז למשבצת מסוימת היא ב(O(1

בדיקה האם שח - (מספר הכלים)O

בדיקת האם מט - (מספר המהלכים החוקיים * בדיקה האם שח)O

הגדרה בסביבת העבודה:

```
typedef enum Player {
	BLACK,
	WHITE,
} Player;

enum BoardType {
	WHITE_PAWNS, WHITE_KNIGHTS, WHITE_BISHOPS,
	WHITE_ROOKS, WHITE_QUEENS, WHITE_KINGS,
	BLACK_PAWNS, BLACK_KNIGHTS, BLACK_BISHOPS,
	BLACK_ROOKS, BLACK_QUEENS, BLACK_KINGS,
	WHITE_ALL, BLACK_ALL, ALL, NUM_OF_BOARD_TYPES,
};

typedef struct {
	u64 board[NUM_OF_BOARD_TYPES];
	Player current_player;
} Bitboard;
```

8. תיאור האלגוריתם הנבחר

האלגוריתם הנבחר הוא מכונת מצבים כדי למצוא את המהלך האופטימלי עבור השחקן הממוחשב. אלגוריתם זה נבחר כיוון שהוא אלגוריתם יעיל ומינימלי שביצועיו לא תלויים בחומרה שהוא רץ עלייה.

נרכיב גרף מצבים כאשר כל קודקוד מכיל פונקציה של מהלך המחזירה הצלחה או לא. אם המצב בפונקציה מתקיים והיא מצליחה לפתור את המצב היא תחזיר צלחה, אחרת כישלון. בגרף זה הקשתות מכוונות ולא ממושקלות.

מצב פתיחה

- בחר מהלך מספר הפתיחות

מצב שח

- אם השח הוא מ2 כלים ומעלה הזז את המלך אחרת הסר את האיום עם הכלי בעל החשיבות הנמוכה ביותר

מצב הגנה

- אכילת מאיים במידה ומשבצת המאיים אינה מאויימת
- הסר מזלג
- הסר הצמדה
- ברח

מצב התקפה

- בצע מט
- בצע שח
- תאכל בעזרת כלי בעל ערך נמוך כלי בעל ערך גבוה
- בצע מזלג
- בצע הצמדה

מצב פיתוח

- קידום חייל
- איזון חיילים
- פיתוח פרש
- פיתוח רץ
- פיתוח מלכה

כעת כדי למצוא את המהלך האידיאלי לפי מכונת המצבים הנתונה, האלגוריתם יבצע DFS ואם יגיע לעלה הוא יבצע את תוכנו של הקודקוד.

הגרף:

9. האלגוריתם הראשי בפסאוד קוד

הפונקציה DFS:

מקבלת: גרף.

מחזירה: מהלך.

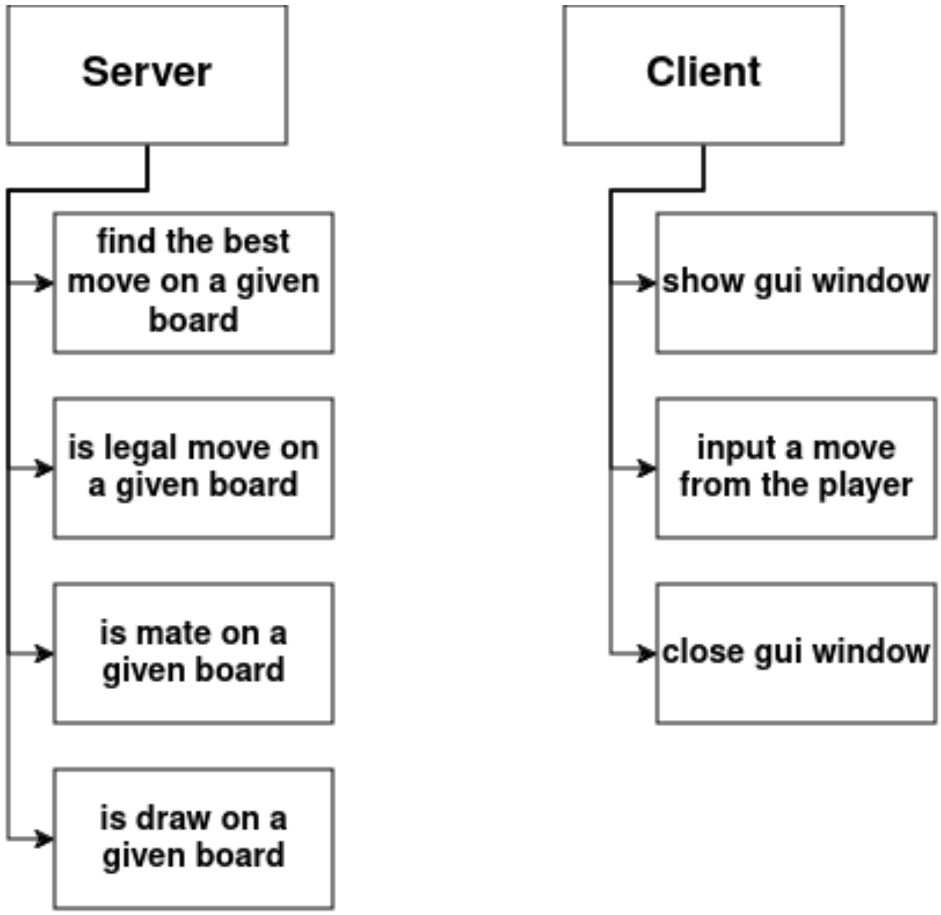
סיבוכיות: O(V+E).

תיאור הפונקציה: האלגוריתם מתחיל בקודקוד ההתחלה ומבצע DFS כאשר הוא מתקדם בענף רק אם הוא מקיים את התנאי שבקודקוד.

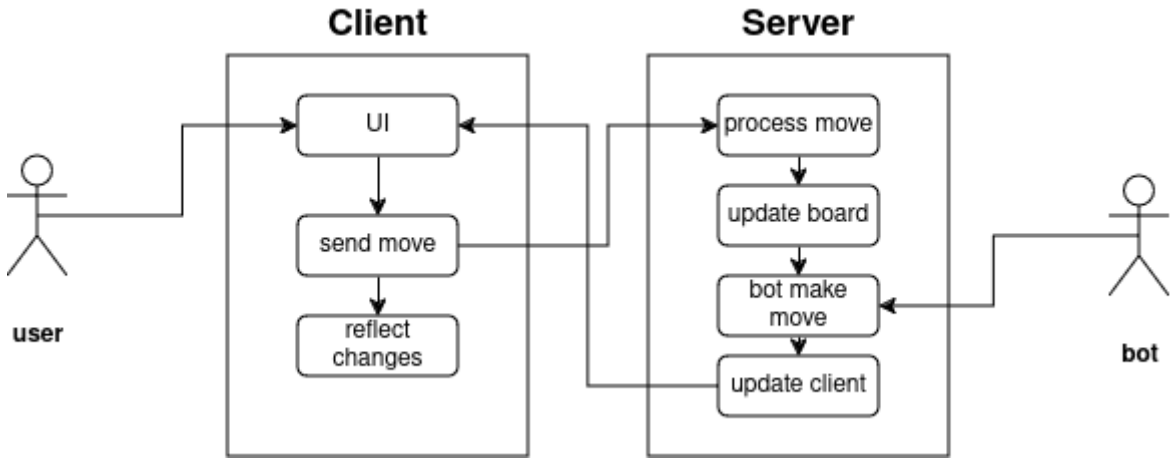
```
Function DFS(Graph G, board)
{
	bool done = false
	create stack s
	create visited array
	push start node to s

	while s not empty and not done {
		node = pop from s

		if visited[node] == false {
			visited[node] = true
			if node is leaf {
				done = node.function(board)
			}

			for v in node.adjucents {
				if visited[v] == false && v.function(board) == true {
					push v to s
				}
			}
		}
	}
}

Function get_best_move(Graph G, board)
{
	return DFS(G, board)
}
```

10. ארכיטקטורה של הפתרון המוצע בפורמט Top-Down Level Design

במצב אידיאלי פרויקט זה ימומש ע"י ארכיטקטורה שרת-לקוח, אך פרויקט זה הינו אבטיפוס ולכן הוא מומש בארכיטקטורה standalone.



11. תרשים מקרי שימוש UML Use cases



12. תרשים סביבת העבודה ושפת התכנות

שפת תכנות: c

סביבת עבודה: gcc

13. התכנית הראשית

```
Bitboard bitboard
bool draw = false
bool mate = false
GuiInitWindow(fen)
BitboardInit(bitboard, fen)
mate = BitboardIsMate(&bitboard)
draw = BitboardIsDraw(&bitboard)

while (!WindowShouldClose() && !draw && !mate) {
    GuiDrawWindow()
    if (current_player == BLACK_PLR) {
        play_computer_move(&bitboard)
        switch current player
    } else {
        play_user_move(&bitboard)
        switch current player
    }
    mate = BitboardIsMate(bitboard)
    draw = BitboardIsDraw(bitboard)
}

GuiCloseWindow();
```

14. תיאור ממשק גרפי

ספריית raylib הינה ספריית גרפיקה פשוטה הכתובה בc.

השתמשתי בספרייה לצורך:

- הצגת החלון
- ניגון שמע של אכילה ותזוזה
- הצגת הלוח
- הצגת הכלים
- קליטת מהלך מהמשתמש

החלון מורכב ממטריצה של ריבועים כאשר כל משבצת יכולה להכיל כלי.

הפונקציות העיקריות מספריית raylib:

תיאור הפונקציה	שם הפונקציה
הפונקציה מאתחלת את החלון	void InitWindow(int width, int height, const char *title);
הפונקציה סוגרת את החלון	void InitWindow(int width, int height, const char *title);
הפונקציה טוענת תמונה ממערכת הקבצים	Image LoadImage(const char *fileName);
הפונקציה משנה את גודלה של תמונה	void ImageResize(Image *image, int newWidth, int newHeight);
הפונקציה טוענת קובץ צליל ממערכת הקבצים	Sound LoadSound(const char *fileName);

15. תיאור הפונקציות הראשיות בפרויקט

הפונקציה balance_pawns:

מקבלת: לוח.

מחזירה: משתנה בוליאני.

סיבוכיות: O(מספר החיילים).

תיאור הפונקציה: הפונקציה מאזנת את החיילים במידה ואפשר.

```
Funtion balance_pawns(board, player)
{
    pawn_mask = all player pawns mask
    for pawn in player pawns {
        if move_mask(pawn) & pawn_mask {
            make_move(board, pawn,  move_mask(pawn) & pawn_mask)
            return true
        }
    }
    return false
}
```

הפונקציה count_check_threats:

מקבלת: לוח, שחקן.

מחזירה: מספר.

סיבוכיות: O(מספר הכלים ליריב).

תיאור הפונקציה: הפונקציה מונה את מספר המאיימים על המלך.

```
Funtion count_check_threat(board, player)
{
    king_mask = player king mask
    count = 0
    for attack_mask in every opponent piece attack masks{
        if king_mask & attack_mask != 0 {
            count++
        }
    }
    return count
}
```

הפונקציה get_check_threat:

מקבלת: לוח, שחקן.

מחזירה: כלי.

סיבוכיות: O(מספר הכלים ליריב).

תיאור הפונקציה: הפונקציה מחזירה את המאיים על המלך.

```
Funtion get_check_threat(board, player)
{
    king_mask = player king mask
    for piece in opponent pieces {
        if king_mask & attack_mask(piece) != 0 {
            return piece
        }
    }
    return 0
}
```

הפונקציה get_threat:

מקבלת: לוח, שחקן.

מחזירה: כלי.

סיבוכיות: O(מספר הכלים ליריב).

תיאור הפונקציה: הפונקציה מוצאת מאיים על כלי של השחקן שהוא בעל חשיבות קטנה יותר מאשר הכלי שעליו הוא מאיים.

```
Funtion get_threat(board, player)
{
        my_mask = player pieces mask
        for piece in opponent pieces {
                if attack_mask(piece) & my_mask &&
                 value(piece) >= value( attack_mask(piece) & my_mask) {
                        return piece
                }
        }
        return 0
}
```

הפונקציה run:

מקבלת: לוח, כלי, כלי, שחקן.

מחזירה: משתנה בוליאני.

סיבוכיות: O(1).

תיאור הפונקציה: הפונקציה מסירה את האיום על הכלי הנתון ע"י בריחה למקום לא מאויים.

```
Funtion run(board, piece, player)
{
    move_mask = MoveMask(by_piece)
    oponent_mask = AttackMaskAll(oponent(player)
    if move_mask & oponent_mask ) {
        play_move(by_piece, bsf(move_mask & oponent_mask))
        return true
    }
    return false
}
```

הפונקציה can_mate:

מקבלת: לוח, שחקן.

מחזירה: משתנה בוליאני.

סיבוכיות: O(מספר המהלכים לשחקן).

תיאור הפונקציה: הפונקציה בודקת האם קיים מט במהלך אחד.

```
Funtion can_mate(board, player)
{
    moves = getAllValidMoves(board, player)
    for move in moves {
        tmp = board
        makeMove(tmp, move)
        if is_mated(board_player) {
            return true
        }
    }
    return false
}
```

הפונקציה is_safe_move:

מקבלת: לוח, שחקן, מהלך.

מחזירה: משתנה בוליאני.

סיבוכיות: O(1).

תיאור הפונקציה: הפונקציה בודקת האם מהלך בטוח ולא חושף את המלכה.

```
Funtion is_safe_move(board, player, move)
{
    tmp = board
    makeMove(tmp, move)
    queen = player queen
    attack_mask = AttackMaskAll(tmp, oponent(player))
    if attack_mask & queen {
        return false
    }
    return true
}
```

הפונקציה get_fork:

מקבלת: לוח, שחקן.

מחזירה: כלי.

סיבוכיות: O(1).

תיאור הפונקציה: הפונקציה מחזיקה את הפרש הגורם למצב מזלג אם קיים.

```
Funtion get_fork(board, player)
{
    my_mask = player pieces
    for knight in opponent knights {
        if count_bits(attack_mask(knight) & my_mask) > 1 {
            return knight
        }
    }
    return false
}
```

הפונקציה AttackMaskBishop:

מקבלת: לוח, ביט הרץ.

מחזירה: מאסק.

סיבוכיות: O(1).

תיאור הפונקציה: הפונקציה מחזירה את המאסק של המהלכים האפשריים של רץ, היא עושה זאת ע"י ביצוע not עם הקרן של החוסם הכי קרוב לכלי לכל כיוון בנפרד. פונקציה זו תיראה כמעט זהה לפונקציה שמחזירה את המאסק של המהלכים לצריך כיוון ששניהם כלים מחליקים.

```
static inline u64 AttackMaskBishop(Bitboard *b, u8 k, Player player)

{

	u64 mask = 0;

	mask |= (~bishop_moves_north_east[bsf(b->b[ALL] & bishop_moves_north_east[k+1])]) &
bishop_moves_north_east[k+1];

	mask |= (~bishop_moves_north_west[bsf(b->b[ALL] & bishop_moves_north_west[k+1])])
& bishop_moves_north_west[k+1];

	mask |= (~bishop_moves_south_east[bsb(b->b[ALL] & bishop_moves_south_east[k+1])])
& bishop_moves_south_east[k+1];

	mask |= (~bishop_moves_south_west[bsb(b->b[ALL] & bishop_moves_south_west[k+1])])
& bishop_moves_south_west[k+1];



	mask &= ~(b->b[player ? WHITE_ALL : BLACK_ALL]);



	return mask;

}
```

פעולות על טיפוס Bitboard:

שם הפונקציה	תיאור הפונקציה
void BitboardInit(Bitboard *b, const char *fen);	הפונקציה מאתחלת את הלוח בעזרת מחרוזת fen או עם הלוח ברירת מחדל.
void BitboardMakeMove(Bitboard *b, u8 srck, u8 destk);	הפונקציה מבצעת מהלך בלוח
bool BitboardIsChecked(Bitboard *b, Player player);	הפונקציה בודקת האם שחקן נמצא בשח
bool BitboardIsMated(Bitboard *b, Player player);	הפונקציה בודקת האם שחקן נמצא במט
bool BitboardIsDraw(Bitboard *b);	הפונקציה בודקת האם הלוח במצב תיקו
bool BitboardIsValidMove(Bitboard *b, u8 srck, u8 destk, Player player);	הפונקציה בודקת האם מהלך חוקי
bool getAllValidMoves(Bitboard *b, u8 moves[][2], u32 *len, u32 n, Player player);	הפונקציה מחזירה את כל המהלכים החוקיים

16. מדריך למשתמש

לבניית הפרויקט במערכות מבוססות unix

```
$ chmod +x build.sh
$ ./build.sh
```

כדי להריץ את המשחק

```
$ ./chess
```

כעת יפתח חלון המשחק:

אתה השחקן הלבן והמחשב הוא השחקן השחור, כדי לבצע מהלך יש ללחוץ על כלי לבן ולאחר מכן על משבצת היעד.

כעת הכלי יזוז למשבצת שבחרת במידה והמהלך חוקי, ואחר כך המחשב יבצע מהלך נגדי.

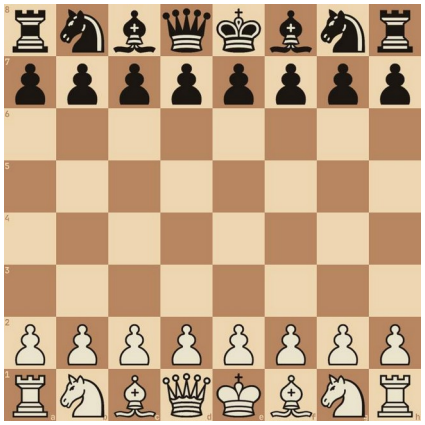
חזור על פעולות אלה עד שיש נצחון או תיקו ואז לא תוכל לבצע מהלכים יותר.

כדי לשנות את הגדרות המשחק יש לפתוח את הקובץ config.h ולשנות את ערכי המשתנים לפי הצורך.

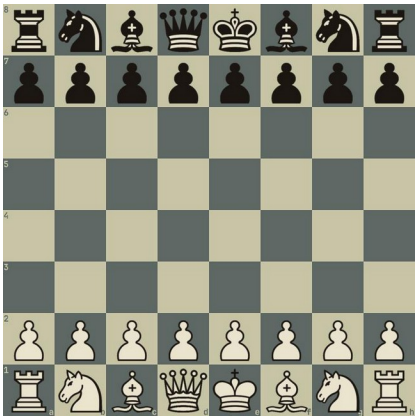
לאחר השינוי יש לבצע קומפילצייה מחדש לפי ההנחיות למעלה.

דוגמאות לערכות נושא:

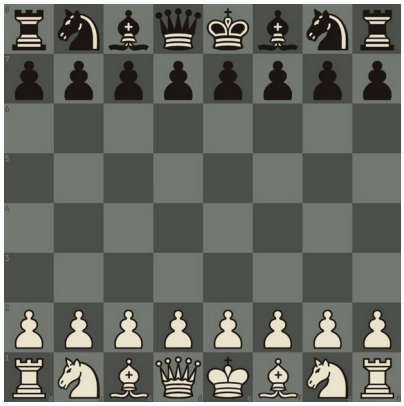
#define THEME_1



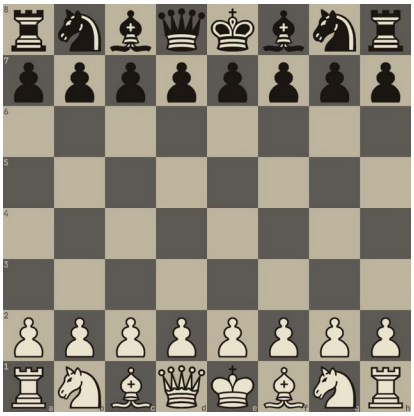
#define THEME_2



#define THEME_3



#define THEME_4



כדי לשנות את הצלילים של הזזת שחקן ואכילה או את התמונות של החיילים עיין בתיקיית assets.

פורמט תמונה חייב להיות png או jpg. ופורמט הצלילים חייב להיות mp3.

17. רפלקציה

בשבילי מחשב נגד בן אדם זהו תחום שעניין אותי נורא, איך המחשב יודע מה המהלך הנכון?, איך המחשב מתכנן?, איך הוא בוחר בין אסטרטגיות שונות?.

התחלתי את הפרויקט בגרפיקה עוד לפני שחקרתי את האלגוריתמים והאסטרטגיות, ולאחר מכן התחלתי את העבודה בספר הפרויקט.

במהלך הפרויקט פיתחתי דעות רבות לגבי השפה c ואף ניסיתי שפות אחרות אך לא מצאתי שפה כה שהתאימה לי יותר. לדעתי c היא שפה טובה ללמידה כי היא פשוטה והתרגום שלה לאסאמבלי הוא צפוי. מספר זה מספר, מחרוזת זו מחרוזת. אין טריקים ושטיקים, מה שרואים זה מה שקורה במחשב. אך זו גם הבעיה, כיוון שזו שפה פשוטה למדי, ישנו קושי והמון עבודה כדי לממש אפליקציות מסוימות.

במהלך הפרויקט החלפתי אינספור אסטרטגיות. בתחילת הפרויקט התעניינתי מאוד באלגוריתם אבולוציוני שכן הוא מדמה אבולוציה וכן מגיע לפתרונות יצירתיים, שכן קשה ללמד מחשב לחשוב בייצירתיות.

באכזבה לא הצלחתי למצוא דרך לשלב את האלגוריתם האבולוציוני עם בעיית המהלך הטוב בשחמט, ולכן בחרתי במכונת מצבים.

אני מאוד התלהבתי מייצוג לוח השחמט בbitboard ואיך שאפשר לבדוק האם מהלך חוקי בזמן קבוע, ללא לולאות, וללא שאלות.

אני שמח שלא דחיתי את הפרויקט לרגע האחרון וכך יצא לי לעבוד עליו קרוב לחצי שנה, כך התהליך היה מהנה ולא הרגשתי שאני בלחץ להספיק, בשונה מהגישה של כל העולם כיום...

בפרויקט זה הבנתי את המשמעות של לתכנן לפני ואז לבצע, שכן עשינו את הספר ואז כתבנו קוד, ולא ישר קוד, קודם כל פסאודו קוד. כי אם זה לא מסתדר בפסאודו קוד זה לא יסתדר בקוד אמיתי. כך אפשר להימנע משגיאות ותסכול רב.

18. ביבילוגרפיה

Analog Hors (2022)

Magical Bitboards and How to Find Them: Sliding move generation in chess

https://analog-hors.github.io/site/magic-bitboards/

Rhys Rustad-Elliott (2019)

Fast Chess Move Generation With Magic Bitboards

https://rhysre.net/fast-chess-move-generation-with-magic-bitboards.html

Omer Madmon, (2023)

איך מחשבים משחקים שחמט?

https://typefully.com/OmerMadmon/tc0DnJO

אבי רוזנטל (2018)

האם לאינטליגנציה מלאכותית יכולה להיות אינטואיציה?

https://jokopost.com/it/16321/