

# דמקה עם אינטליגנציה מלאכותית ממוטבת

**פרויקט גמר - י"ג הנדסת תוכנה**

**פרטים אישיים**

שם: דניאל קנבסקי

ת.ז: 206629776

אימייל: dk19dk07@gmail.com

מוסד: מכללת הכפר הירוק

מנחה: יהודה אור

שפת תכנות: Java

תאריך פיתוח: 10.3 – 21.4

2018 תשע"ח

3.....	מבוא
3.....	ראשית דבר
4.....	תמונות
9.....	רקע תאורטי
9.....	חוקי המשחק
13.....	מינימקס
16.....	גיזום אלפא-באטא
19.....	אלגוריתם תוך-מקומי
19.....	תכנות מונחה עצמים
20.....	פיתוח הפרויקט
20.....	סביבת עבודה
21.....	לוח המשחק
22.....	מימוש תהליך המשחק
24.....	פיתוח האינטליגנציה המלאכותית
27.....	הערכת מצב
29.....	קוד הפרויקט
29.....	MyProject
31.....	DamkaTile
35.....	Damka
56.....	Computer
66.....	רפלקציה
67.....	ביבליוגרפיה

# מבוא

## ראשית דבר

### \* מה יצרתי?

יצרתי משחק של דמקה רוסית לפי החוקים הרשמיים שפורסמו בברית המועצות בשנת 1884, אך אף מעבר. המשתמש יכול לבחור את **גודל הלוח** שעליו ישוחק המשחק ואת **מספר שורות החיילים** שיש לכל צד. למשתמש נתונה הבחירה לשחק **מול בן אדם** אחר או **מול אינטליגנציה מלאכותית ממוטבת**.

### \* למה דמקה?

תמיד אהבתי לשחק דמקה. למרות פשטות החוקים, המשחק כלל לא פשוט! כדי לשלוט במשחק צריך יכולת של חשיבה אנליטית ואסטרטגית, יכולת לחשוב קדימה ולנתח מצבים שעלולים להופיע בעתיד, והבנה עמוקה של אופי המשחק. בנוסף, רציתי ליצור אינטליגנציה מלאכותית ברמה גבוהה מאוד, כך שלבני אדם לא יעמוד סיכוי לנצח אותה.

### \* איך יצרתי?

יצרתי את התוכנית בסביבת העבודה NetBeans באמצעות שפת Java תוך שימוש בכלים של swing. את האריחים ציירתי באמצעות MS Paint.

### \* תודה רבה!

תודה רבה ליהודה אור, מנחה הפרויקט, שהנחיתו המקצועית ועצותיו המועילות עזרו לי בעשיית הפרויקט ושיפורו. ארצה להודות לחבריי לכיתה שתרמו לי מהידע ומהניסיון שלהם. תודה רבה לאחותי ולאבא שלי ששיחקו איתי, ובכך עזרו לי למצוא באגים.

### \* שימו לב!

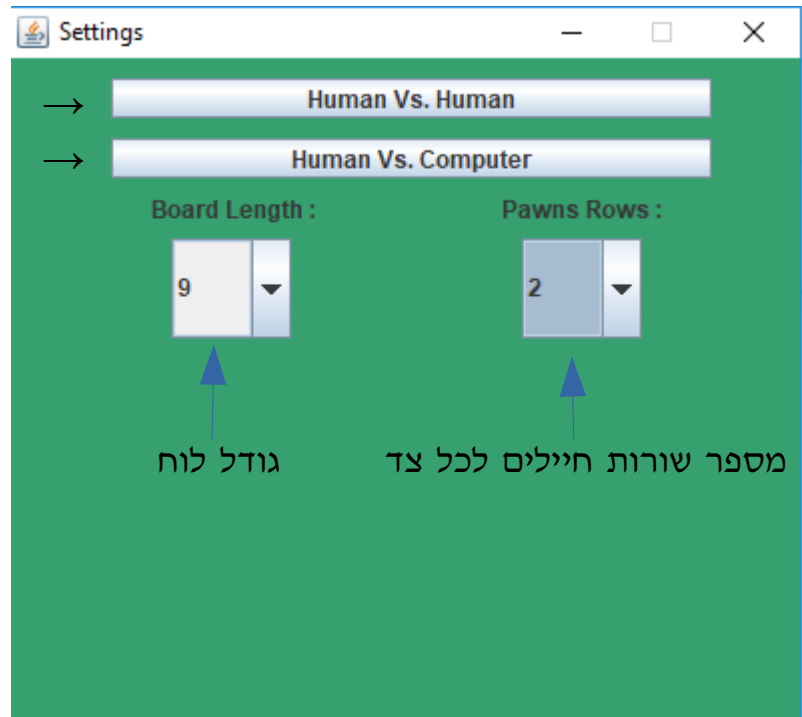
הפרויקט והספר הם עבודתם האישית והבלעדית. כל פרט בהם שייך ונעשה ב 100% על ידי ואין אישור להשתמש בהם ללא רשותי.

# מבוא

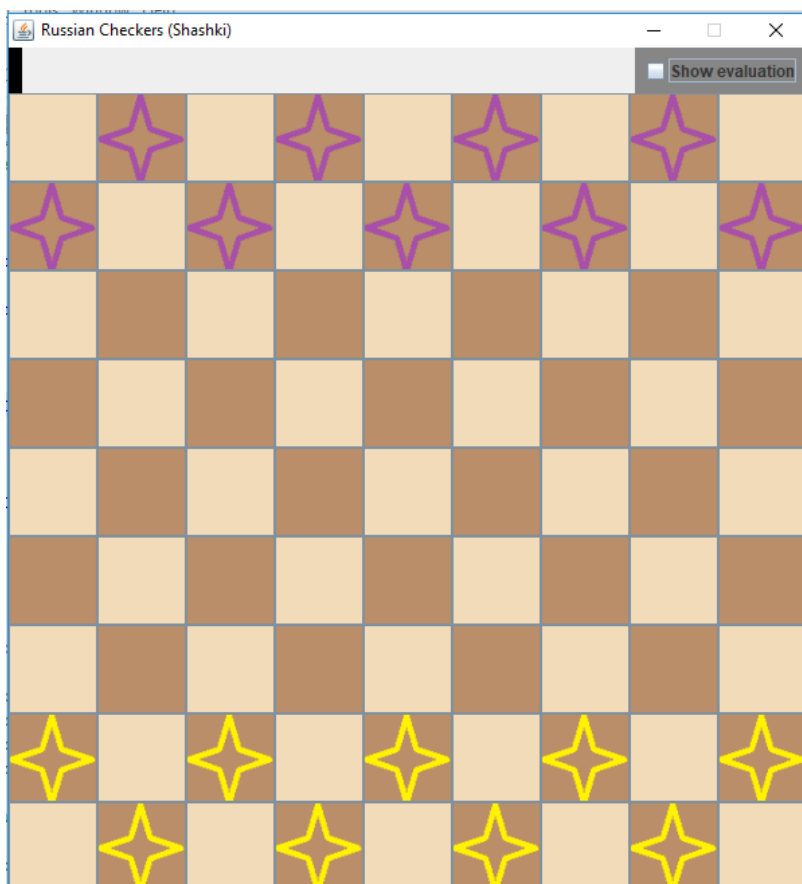
## תמונות

מסך הגדרות ראשי:

בן אדם מול בן אדם  
בן אדם מול מחשב



לאחר התחלת המשחק יראה הלוח כך:

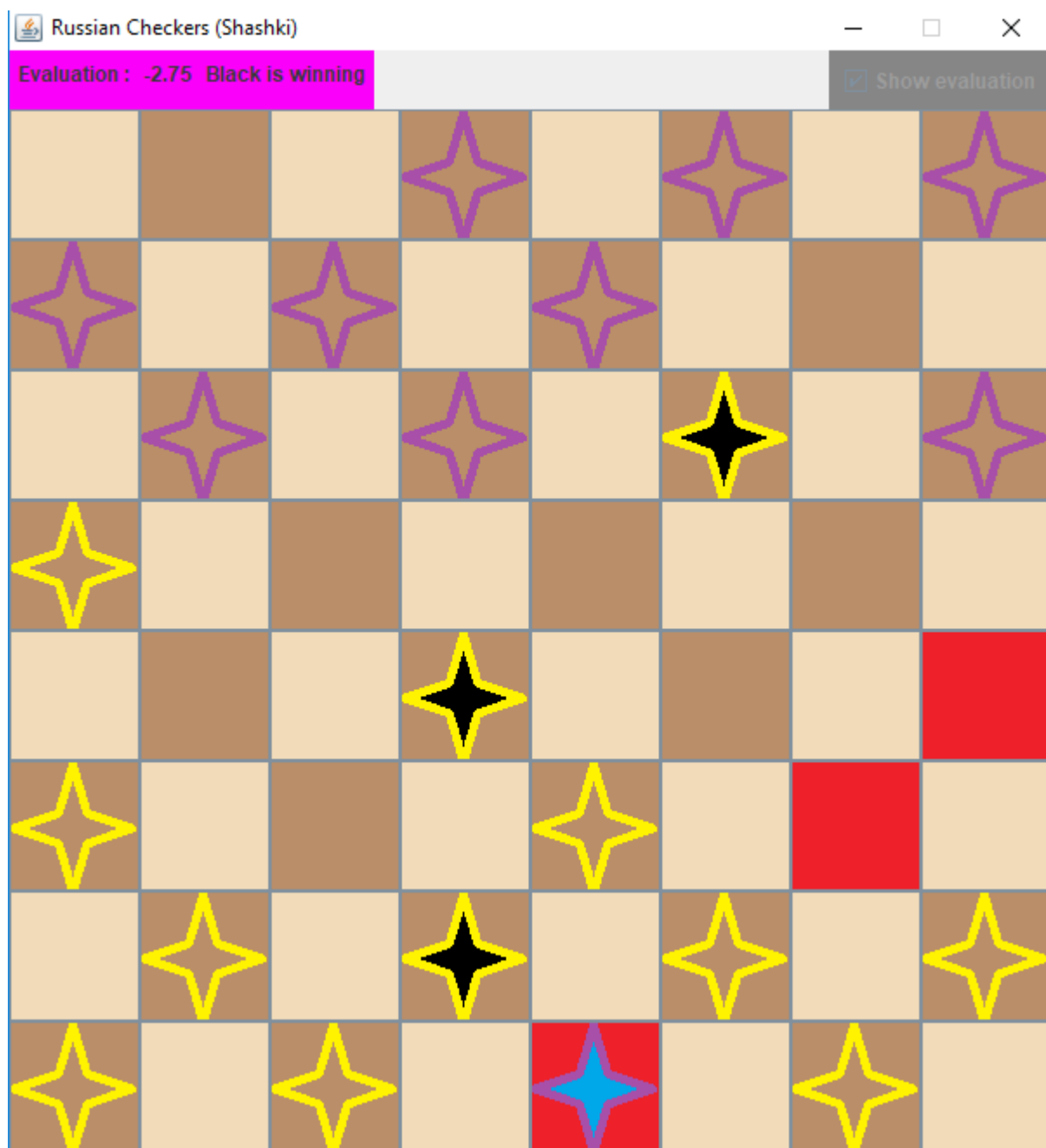


כפי שניתן לראות, גודל הלוח הוא 9X9 ולכל צד (הצד ה"לבן" משחק עם החיילים הצהובים והצד ה"שחור" משחק עם החיילים בצבע סגול כהה) יש שני שורות של חיילים. גודל הלוח הוא קבוע לכל אורך המשחק. מספר החיילים יורד בהדרגה במהלך המשחק.

בהמשך מוצגות תמונות ממשחקים שונים.

# מבוא

מהלך המשחק:

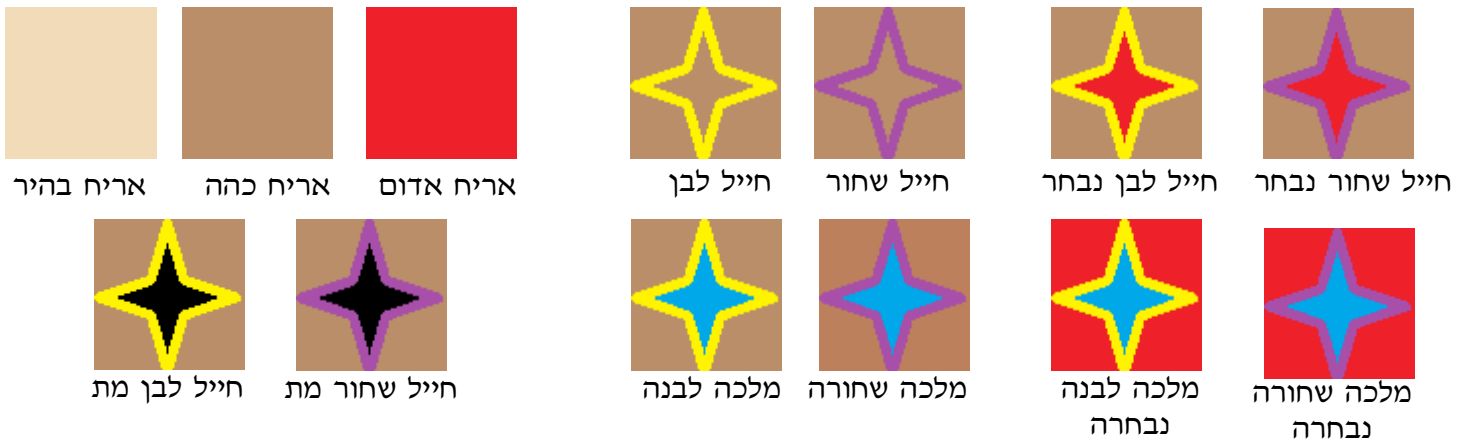


תמונה זו מראה הרבה אלמנטים במשחק.

בהמשך יוסבר בפירוט רב מה מתרחש כאן.

# מבוא

לוח המשחק מורכב מ-13 האריחים הבאים:



המשחק יכול להסתיים בשלושה דרכים.

ניצחון על ידי "לבן":



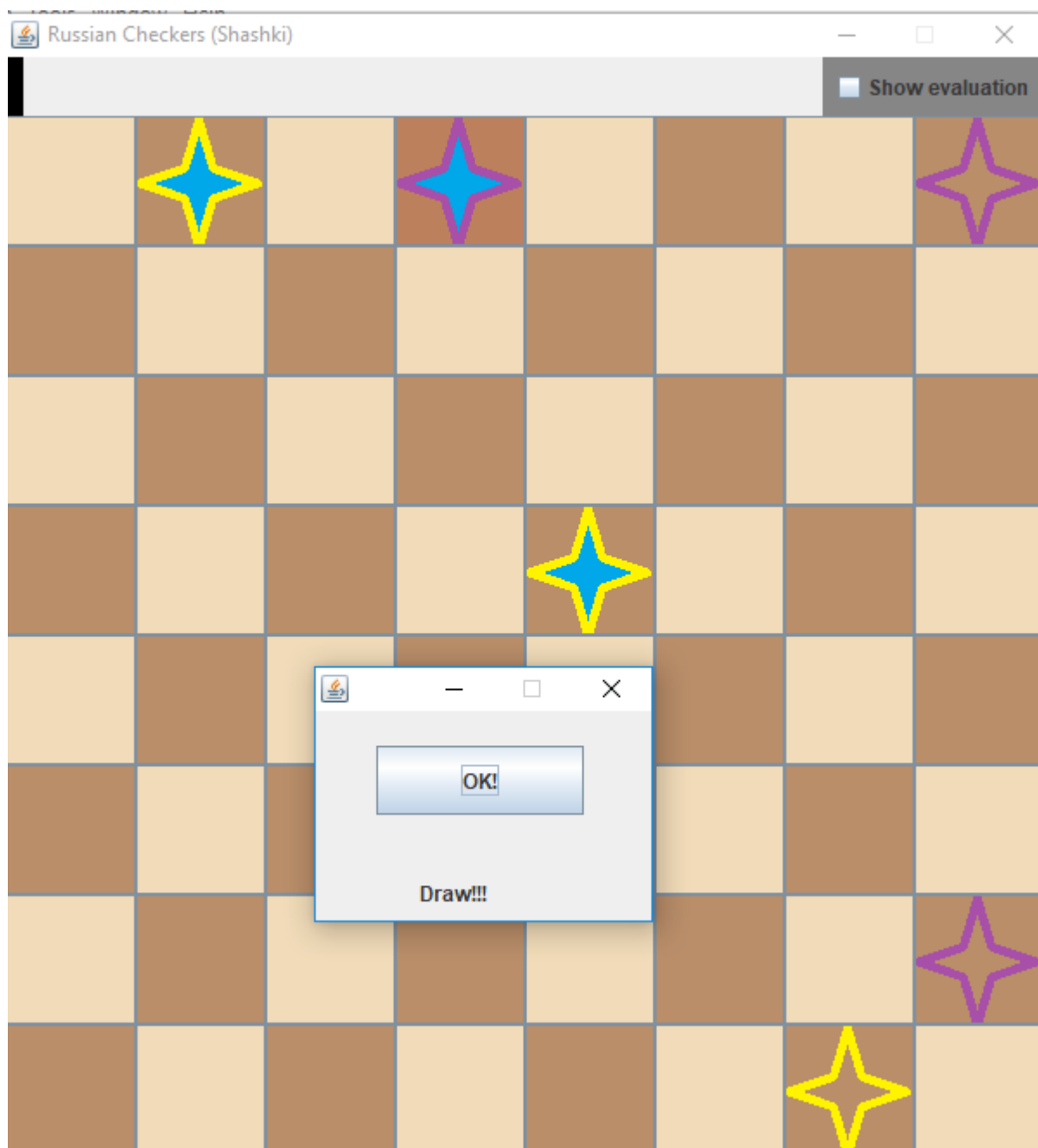
# מבוא

ניצחון על ידי "שחור":



# מבוא

תיקו:





# רקע תאורטי

## חוקי המשחק

מכיוון שדמקה הוא משחק פופולרי בכל העולם, יש לו גרסאות שונות מאזור לאזור. הגרסה המשוחקת בישראל, והגרסה שלפיה מימשתי פרויקט זה היא **דמקה רוסית**. החוקים המוצגים בהמשך הם החוקים הרשמיים כפי שפורסמו בברית המועצות בשנת 1884.

המשחק משוחק על ידי שני שחקנים "לבן" ו"שחור" כך שלבן משחק בחיילים הבהירים, ושחור משחק בחיילים הכהים, הצד הלבן מתחיל במשחק והתורות מתחלפים לסירוגין. החיילים ממוקמים על הריבועים הכהים כך שהריבועים הבהירים נשארים ריקים, ובהתאם, המשחק משוחק אך ורק על הריבועים הכהים. המשחק מתחיל בלוח של  $8 \times 8$  ולכל צד 3 שורות חיילים (בתוכנה של גודל הלוח הוא  $N \times N$  ו  $M$  שורות חיילים לכל צד, כאשר  $N$  ו  $M$  נתונים לבחירת המשתמש, ו  $N = 8, M = 3$  משמשים כערכי ברירת מחדל) כך:

תזוזת החיילים היא ריבוע אחד קדימה (מנקודת המבט של השחקן המשחק) וריבוע אחד לצד\* (ימין או שמאל) (כמובן, כל עוד המהלך מתבצע בגבולות הלוח) לתוך ריבוע ריק. ברגע שחייל מגיע לשורה האחרונה (מנקודת מבטו של השחקן המשחק) הוא מקודם ל"מלכה" (ידוע גם בתור, "מלך" ו-"דמקה").

מלכה יכולה לזוז באלכסון ללא הגבלה של כמות משבצות ( = ריבועים). להלן המחשה על ידי תמונה:

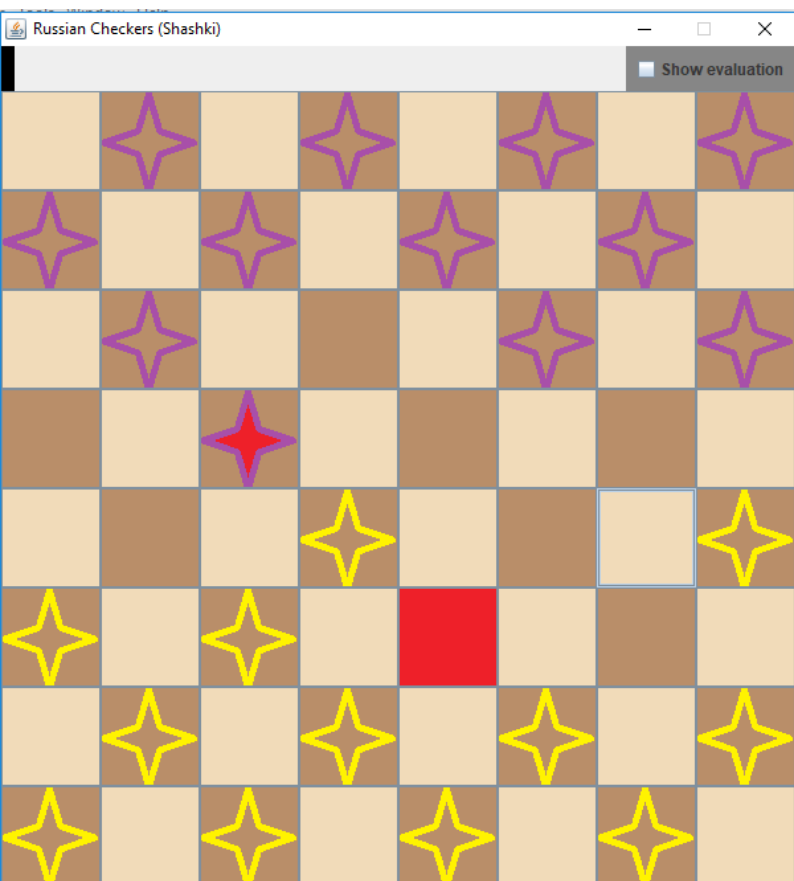
\*תזוזת חייל

כפי שניתן לראות, המלכה נעצרת על אך ורק על ידי גבולות הלוח או על ידי חייל/מלכה מהצבע היריב.

# רקע תאורטי

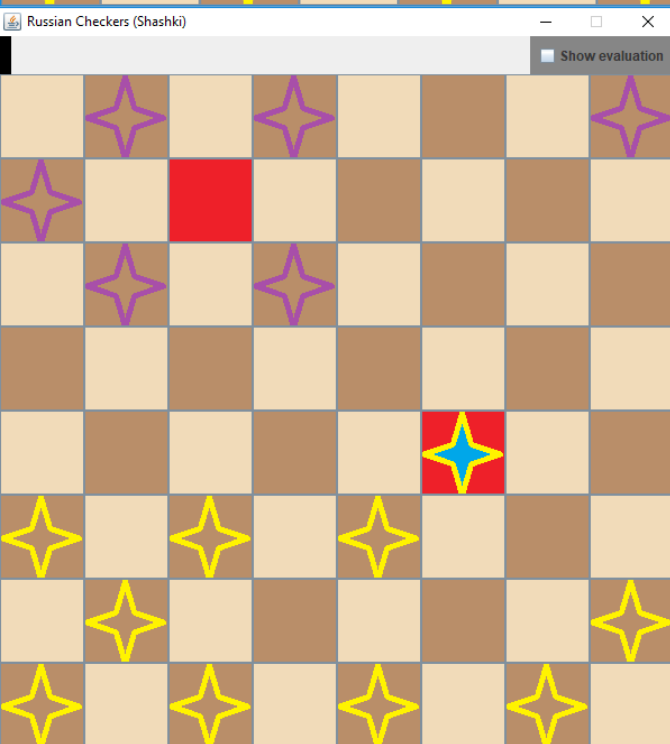
**אכילה** בדמקה רוסית הינה תזוזה מעל אריח של שחקן יריב, ובכך אותו אריח "נאכל" ויוצא מהמשחק. **חובה לאכול**, כלומר אם קיימת אפשרות לאכול אז באותו מהלך תבוצע אכילה. בכל הנוגע לאכילות בדמקה רוסית, גם לחיילים רגילים וגם למלכות, **מוותר לאכול אחורה**. להלן פירוט:

אכילה על ידי חייל אפשרית במידה ועומד בצמוד לחייל אריח יריב, ובאותו כיוון, משבצת לאחריו, נצמא אריח ריק שאליו "יקפוץ" החייל האוכל.



כפי שניתן לראות, לחייל המסומן יש רק מהלך חוקי אחד, שכן, מכיוון שקיימת במצב הלוח אפשרות לאכול, אין לבצע מהלך שאינו אכילה. לכן, לשחור יש רק מהלך חוקי אחד. לאחר אותו מהלך, ללבן יש שני מהלכים אפשריים, מכיוון שהוא חייב לאכול את אותו אריח שחור, אך קיימות מספר דרכים לבצע אכילה.

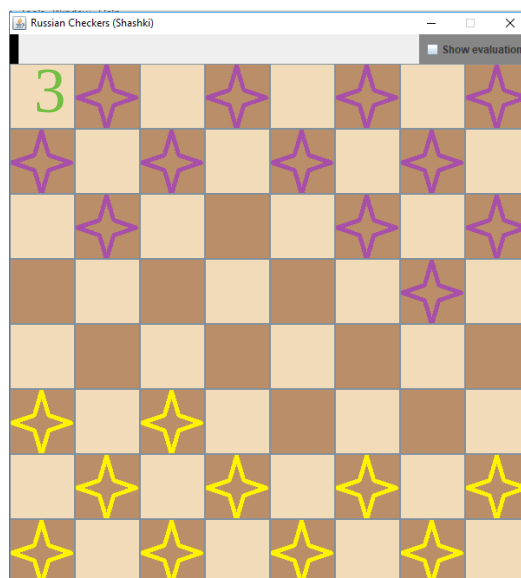
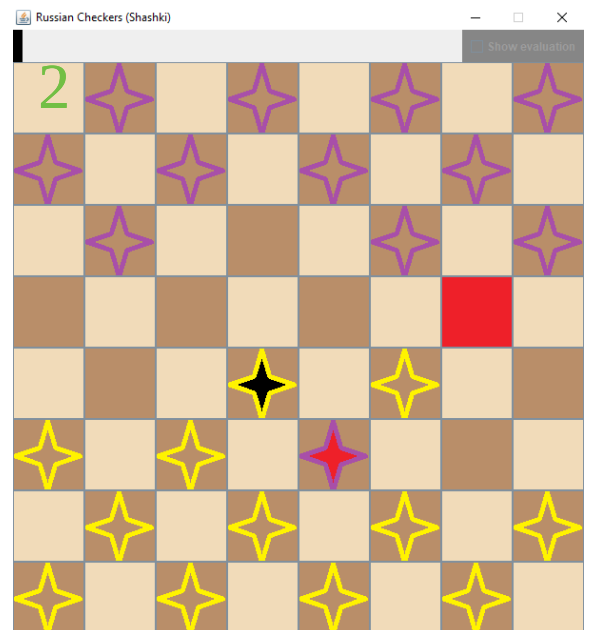
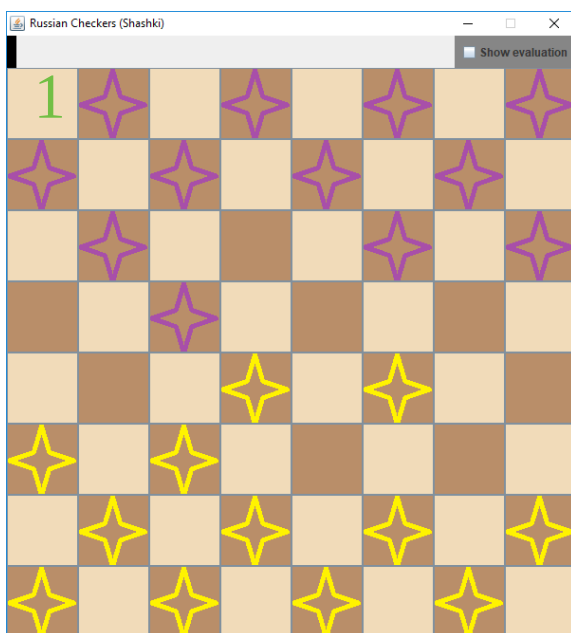
זו דוגמה למצב נפוץ, בו לאחר אכילה של אריח, היריב מחזיר באותו המטבע. בכך בעצם היתרון הזמני מתבטל. כלומר, מתבצעת החלפה, ואף צד לא נהנה מקבלת יתרון מבחינת כמות חיילים בסופו של דבר.



אכילה על ידי מלכות מתבצעת באופן דומה, כל המשבצות הריקות לאחר האריח הנאכל באותו כיוון הן נקודות יעד אפשריות. אין דרישה לכך האריח הנאכל ימצא בצמוד למלכה (להבדיל מחיילים).

# רקע תאורטי

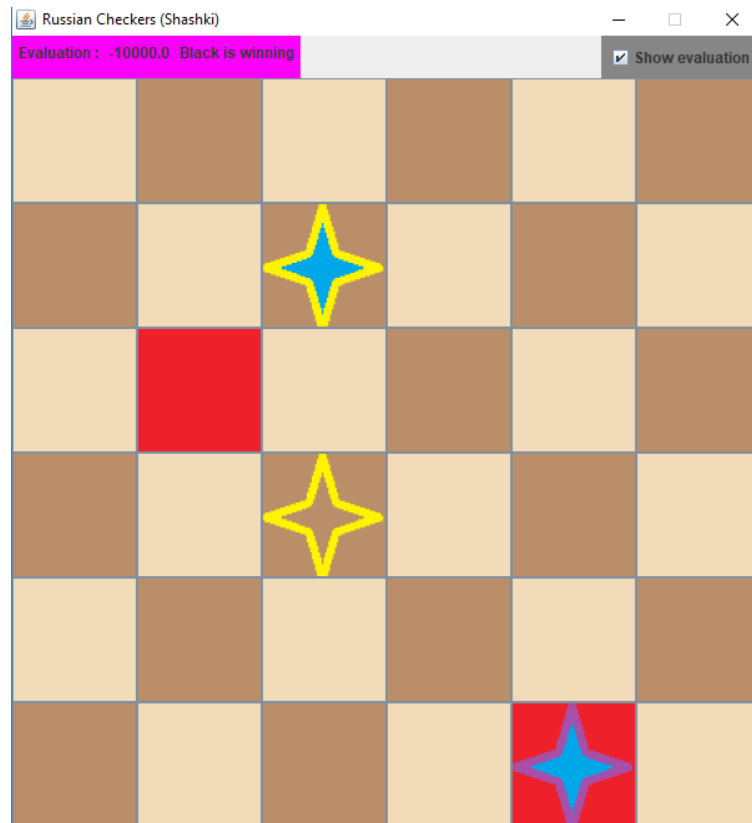
לאחר אכילה באמצעות חייל או מלכה, יכול להיות שקיימת לאותו אריח אפשרות נוספת לאכול. במצב כזה הצד האוכל מקבל מהלך נוסף לתורו בו הוא מחויב לאכול אם החייל שזה עתה אכל, ובכך ליצור **רצף**. ניתן לאכול אריח ספציפי רק פעם אחת, האריחים שנאכלו יישארו על הלוח עד לסיום התור של הצד המשחק, אך לא ניתן יהיה לאכול אותם שוב והם ייחשבו בתור **חיילים מתים**. אם יש אפשרות לאכול ברצף לאחר אכילה מסוימת עם אריח, חובה לאכול אם אותו אריח עד לסיום התור, גם אם יש עוד אריח שיכול לאכול. להלן דוגמה לאכילה ברצף באמצעות חייל.



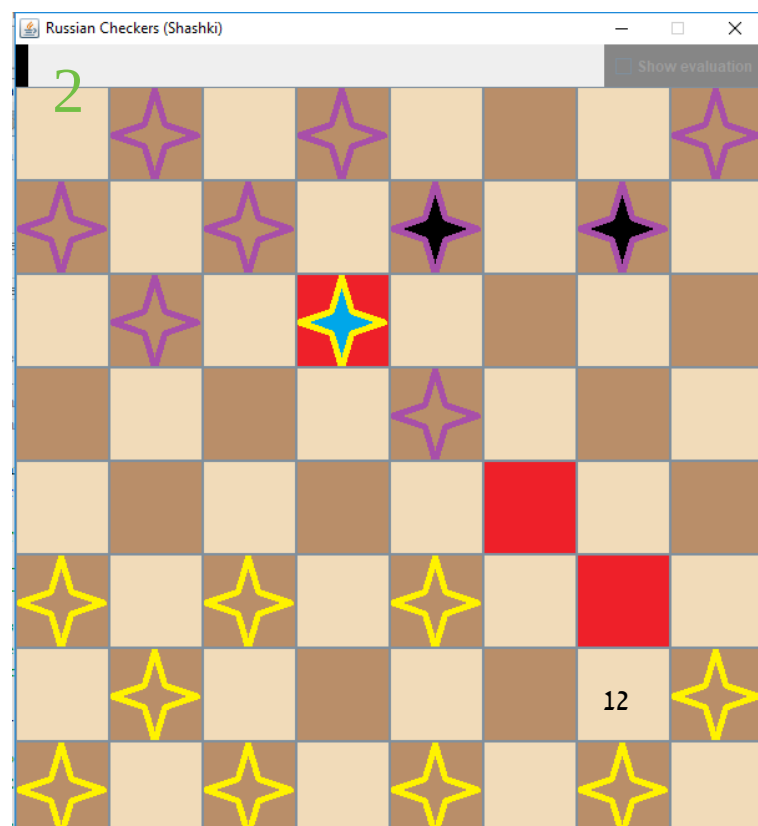
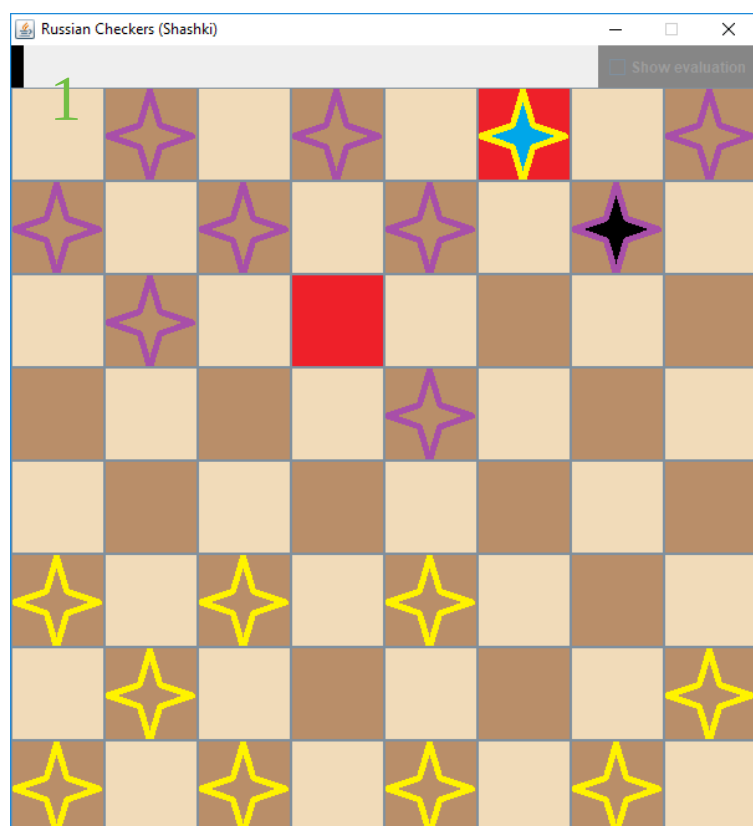
אין הגבלה לגודל הרצף, כלומר ניתן לאכול אם האריח אף יותר מ 2 חיילים, אם המצב בלוח מאפשר זאת.

# רקע תאורטי

אם לאחר אכילת אריח עם מלכה ישנן כמה משבצות יעד, וחלקן (1 לפחות) מובילות לרצף או להמשך הרצף, משבצות היעד הקבילות הם אלו שמובילות לרצף או להמשכו, כפי שניתן לראות כאן:



להלן זוג תמונות נוספות שממחישות את החוקים שהוסברו לעיל:



## רקע תאורטי

המשחק מסתיים בניצחון של אחד הצדדים והפסדו של האחר או בתיקו. ניצחון נוצר כאשר ליריב אין אפשרות לבצע מהלך חוקי. זה יכול לקרות לאחר שכל אריחיו נאכלו או שאריחיו על הלוח חסומים על ידי אריחים אחרים. תיקו יכול להיות לפי הסכמה בין השחקנים, או כאשר התבצעו 15 מהלכים ברצף על הלוח בהם לא זז אף חייל ולא התבצעה אכילה (כלומר רק מהלכים שהם תזוזה של המלכה). תמונות המתארות מצבים בהם המשחק הסתיים ניתן למצוא בעמודים 6-8.

## מינימקס

בדמקה, יריב א' משחק נגד יריב ב' כך שיתרון של אחד מהם הוא חסרונו של האחר. למשל אם יריב א' ניצח אז יריב ב' הפסיד, אם לב' 3 חיילים יותר, אז לא' יש 3 חיילים פחות. באופן כללי יותר, בהינתן מצב לוח מסוים במהלך המשחק (State) ופונקציה Evaluate המחזירה הערכה עבור אותו מצב לאחד מ-2 השחקנים (A, B), אז אם  $Evaluate(State, A) = X$  אז  $Evaluate(State, B) = -X$ . לכן:  $Evaluate(State, A) + Evaluate(State, B) = 0$

מכאן נובע שסכום הרווחים/ההפסדים מנקודת המבט של השחקנים הוא 0. לכאן דמקה קרויה "**משחק סכום אפס**" (המושג לקוח מתורת המשחקים). דמקה הוא לא משחק סכום האפס היחיד, גם שחמט, 4 בשורה, איקס עיגול ואפילו פוקר הם דוגמאות מעולות למשחקי סכום אפס.

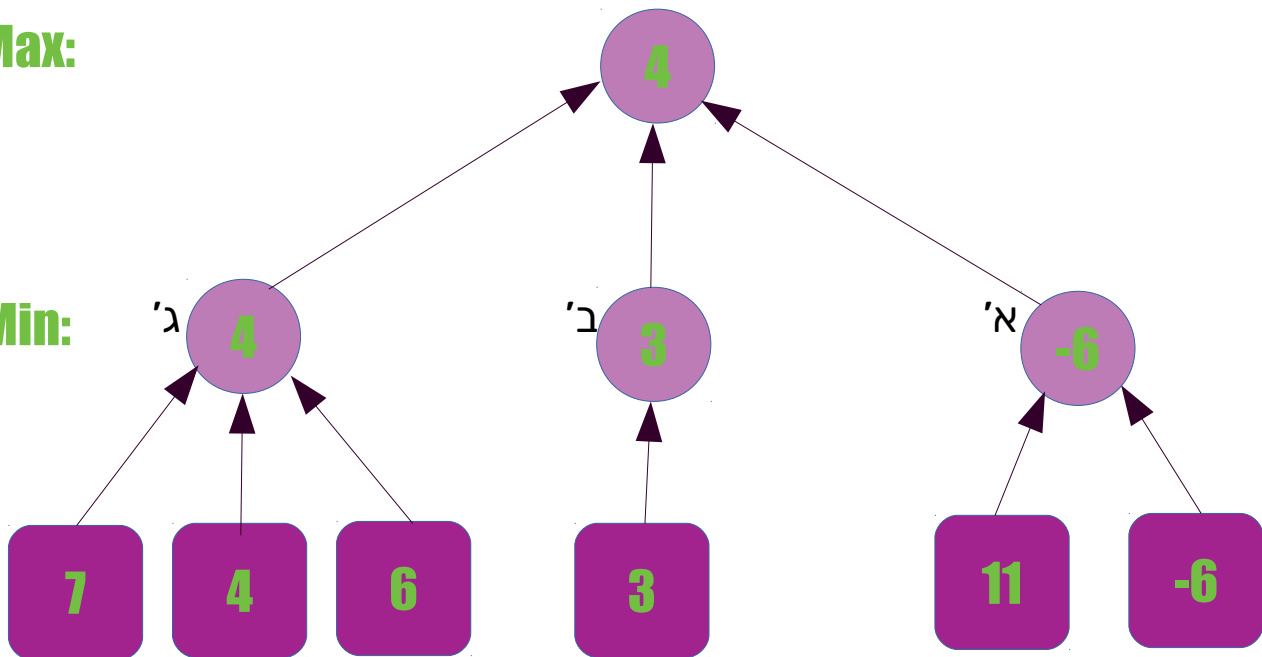
מאחר שדמקה הוא משחק סכום אפס הוא מקיים את **משפט המינימקס** (לפי ניסוח שלי): מבין כל סדרות המהלכים האפשריים ממצב כלשהו בלוח, קיימת דרך אופטימלית לשחק כך שהרווח (מבחינת הערכה אחידה) המינימלי באותו מצב לוח לא תלוי ביריב. אלגוריתם המינימקס מתבסס על רעיון זה.

**אלגוריתם המינימקס** מניח שהיריב משחק בצורה אופטימלית, וכך גם אנחנו נשחק בעתיד. מבין כל המהלכים האפשריים, גם אנחנו וגם היריבים נבחר את המהלך שיוביל לתוצאה הטובה ביותר עבורנו. מכאן נובע השם, אני אנסה להגיע לתוצאה המקסימלית (שחקן המקס) ויריבי ינסה להגיע לתוצאה המינימלית (שחקן המין) (מנקודת המבט של שחקן המקס). שנינו משחקים בהתחשבות בשיקוליו של השחקן האחר. {פירוט והמחשה בעמוד הבא}

# רקע תאורטי

Max:

Min:



התרשים המוצג לעיל מתאר עץ מינימקס בעומק 2 עבור מצב לוח ספציפי בו תורו של שחקן המקס, ויש לו שלושה מהלכים חוקיים (א', ב' וג')

## רעיון האלגוריתם:

איזה מהלך נבצע?

נבצע את המהלך שהערכה שלו היא מקסימלית, שכן תורו של שחקן המקס.

כיצד נדע מהי ההערכה של כל אחד מהמהלכים?

נבצע כל אחד מהמהלכים, כך שלאחר ביצוע מהלך כלשהו, נסתכל על הלוח מנקודת מבטו של מין (שכן, לאחר ביצוע המהלך תורו של מין) ונחשב את ההערכה של הלוח באופן רקורסיבי, כך שכאשר עץ הקריאות הרקורסיבי מגיע לעומק כלשהו שהוגדר על ידי המתכנת, תיקרא פונקציה נפרדת שמקבלת לוח כקלט ומחזירה הערכה כפלט. מבין הערכים שמצאנו באופן רקורסיבי נבחר את הערך המינימלי שכן תורו של מין. לדוגמה. לאחר ביצוע מהלך ג' יוחזר 4 מכיוון שזה הערך המינימלי. ובתהליך של חזרה לאחור (**Backtracking**) נמצא את כל ההערכות של כל אחד מהמהלכים, ולבסוף נבחר לבצע את מהלך ג'.

איך הפונקציה החיצונית נותנת הערכת מצב?

זה תלוי מטרה. במקרה של דמקה יש כמה פרמטרים שונים, כגון מספר חיילים לכל צד, מספר מלכות לכל צד, מספר חיילים בשטח אויב, תור ועוד. (הפונקציה מחזירה ערך חיובי אם מקס מוביל ושלילי אם מין מוביל. ככל שההובלה יותר משמעותית כך הערך המוחלט של ההערכה גדול יותר).

# רקע תאורטי

(הצעה ומימוש אישי)

## פסאודו-קוד של מינימקס:

**משתנה גלובלי:** מהלך\_לביצוע

**פונקציית\_מינימקס**(לוח\_משחק, שחקן\_משחק, עומק\_רקורסיה):

אם עומק\_רקורסיה == עומק\_מקסימלי:

החזר הערכת\_מצב(לוח\_משחק)

רשימת\_מהלכים\_אפשריים[] = השג\_מהלכים\_אפשריים()

אם שחקן\_משחק == מקס:

הערכה\_נוכחית =  $-\infty$

עבור מהלך\_נוכחי מבין רשימת\_מהלכים\_אפשריים:

לוח\_חדש = בצע\_מהלך(לוח\_משחק, מהלך\_נוכחי)

הערכת\_מהלך = פונקציית\_מינימקס(לוח\_חדש, מין, עומק\_רקורסיה + 1)

אם הערכת\_מהלך < הערכה\_נוכחית:

הערכה\_נוכחית = הערכת\_מהלך

אם עומק == 0:

מהלך\_לביצוע = מהלך\_נוכחי

אחרת: \* שחקן\_משחק: מין \*

הערכה\_נוכחית =  $\infty$

עבור מהלך\_נוכחי מבין רשימת\_מהלכים\_אפשריים:

לוח\_חדש = בצע\_מהלך(לוח\_משחק, מהלך\_נוכחי)

הערכת\_מהלך = פונקציית\_מינימקס(לוח\_חדש, מקס, עומק\_רקורסיה + 1)

אם הערכת\_מהלך > הערכה\_נוכחית:

הערכה\_נוכחית = הערכת\_מהלך

אם עומק == 0:

מהלך\_לביצוע = מהלך\_נוכחי

החזר הערכה\_נוכחית

**קריאה לפונקציה:** פונקציית\_מינימקס(לוח\_משחק\_נוכחי, השחקן\_שזהו\_תורו, 0)

לוח\_משחק\_נוכחי = בצע\_מהלך(לוח\_משחק\_נוכחי, מהלך\_לביצוע)

מטרתו של הפסאודו-קוד לעיל הוא להעביר את הרעיון של האלגוריתם. הוא עדיין לא ממוטב(ניתן לשפר אותו, מבחינת זכרון וזמן ריצה) וחסרים פרטים טכניים. בהמשך הספר אציג שיפורים משמעותיים לאלגוריתם, מכמה בחינות (אופטימיזציות). הפסאודו-קוד כתוב בסגנון שפת Python.

# רקע תאורטי

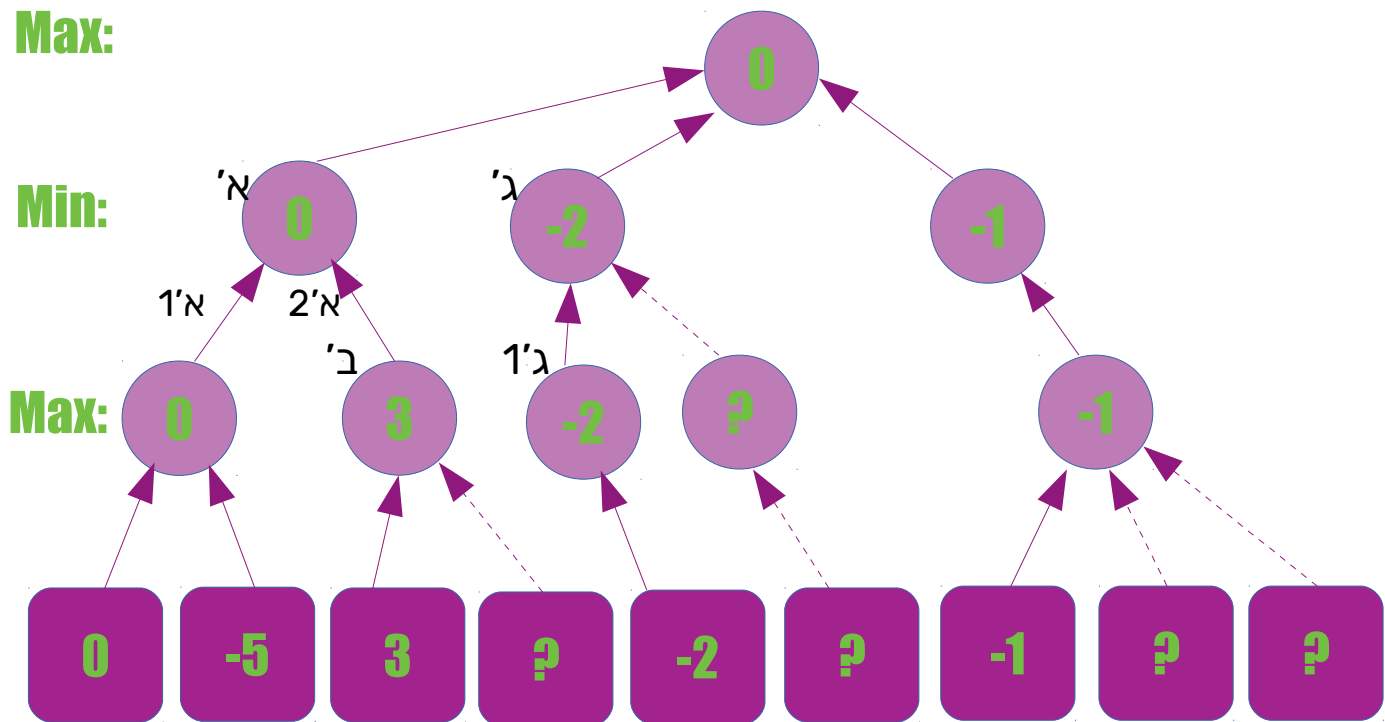
## גיזום אלפא-באטא

גיזום אלפא-באטא היא אופטימיזציה (מיטוב) עבור אלגוריתם מינימקס מבחינת זמן ריצה. סיבוכיות זמן הריצה של אלגוריתם מינימקס (במקרה הגרוע ביותר) היא  $O(m^d)$  כאשר  $m$  הוא מספר המהלכים המקסימלי שקיימים במצב לוח כלשהו, ו  $d$  הוא עומק הרקורסיה. ניתן להוכיח לפי אינדוקציה שזוהי סיבוכיות זמן הריצה מאחר שאם נגדיל את העומק ב 1 נצטרך לחקור פי  $m$  יותר מצבי לוח. מכיוון שבתחילת המשחק לכל צד יש 7 מהלכים אפשריים ומספר המהלכים האפשריים גדל ככל שהמשחק מתקדם, ומגיע לשיאו כשיש מלכה/מלכות על הלוח, האלגוריתם ללא הגיזום מאפשר לתוכנה שלי להגיע לעומק של 5-6 מהלכים בלבד בזמן סביר. היתרון המשמעותי ביותר באופטימיזציית גיזום אלפא-באטא (לדעתי) הוא שהשיפור המשמעותי שהרעיון מספק מבחינת זמן ריצה לא בא על חשבון הדיוק של אלגוריתם המינימקס בניגוד למיטובים אחרים, כלומר המהלך שישוחק בסוף החישוב יהיה אותו מהלך עם או בלי גיזום אלפא-באטא, ההבדל היחיד הוא שמהירות האלגוריתם תהיה קטנה יותר (או אותה מהירות בערך במקרים נדירים). המינימקס שלי עם גיזום מאפשר לתוכנה להגיע לעומק של כ 7-8 מהלכים בזמן סביר (בהחלט ניתן להרגיש בהבדל בזמן המשחק).

אם נסתכל על מצבי הלוח שניתן להגיע אליהם תוך כמה מהלכים, נבחין שחלק מהם אין טעם לחקור, למשל אם כבר גילינו שאם נשחק את מהלך א' ננצח את המשחק ללא תלות בתגובתו של יריבינו, מה הטעם לבצע חקירה יקרת משאבים של מהלך ב', כמובן שזהו מקרה קצה, באופן כללי אם נוכל להיווכח שלא נגיע למצב לוח כלשהו נוכל להפסיק לחקור אותו. מכאן משמעות המילה "גיזום", כפי שגנן גוזם ענפים עץ בגינה כך גם המתכנת "גוזם" קשתות מעץ המינימקס. הגיזום יתבצע על ידי שמירת 2 ערכים; אלפא-הערכה המינימלית שמובטחת לשחקן המקס (מאותחל ל  $-\infty$ ) ובאטא-ההערכה המקסימלית שמובטחת לשחקן המין (מאותחל ל  $\infty$ ). אם אלפא יהיה גדול או שווה מבאטא, נוכל להסיק שאנחנו מסתכלים על מצב שלעולם לא ייבחר, שכן מקס או מין יבחרו להגיע למצב לוח אחר. בעמוד הבא יש המחשה.



# רקע תאורטי



הצמתים נבדקים משמאל לימין. "?" מסמל צומת שלא נבדק כי לא היה צורך.

נסתכל על מצב לוח א', לאחר שחישבנו שמין גילה שאם ישחק א' הוא יגיע למצב עם הערכה 0, נחקור מה קורה אם מין ייבחר במהלך א' 2 למצב ב'. מהלך אחד אפשרי מב' מוביל למצב לוח בעל ההערכה של 3. כעת יש לשאול את השאלה; האם אכפת לנו מהמצב/ים האחר/ים? התשובה היא לא. יש לזכור שבמצב ב' תורו של מקס לשחק והוא יירצה להשיג ההערכה מקסימלית, לכן אנחנו יודעים שהוא ישיג לפחות הערכה של 3 מאותו מצב ואולי אף יותר, מצד שני ידוע שממצב א' ניתן לבחור במהלך שיוביל להערכה של 0. ברור שאין טעם יותר לחקור את מהלך א' 2. מכיוון שבמצב א' תורו של מין והוא שואף לתוצאה הנמוכה ביותר, הוא יכול להשיג 0 אם ירצה אבל ממצב ב' הוא לא ישיג פחות מ 3. לכן יש לבצע גיזום ולהפסיק לחשב את מצב ב'. במילים אחרות, במצב ב' אלפא = 3, באטא = 0, אלפא < באטא, לכן יש לבצע גיזום. במצב ג', לאחר שחושב שערכו של המהלך ג' 1, מכיוון שאז אלפא = 0, באטא = -2, אלפא < באטא ולכן לא נגיע למצב ב' (בהינתן ששני השחקנים משחקים את המהלכים הטובים ביותר). במקרה הגרוע ביותר, לא נמצא קשתות לגזום והסיבוכיות תישאר  $O(m^d)$ , אך במקרה הטוב ביותר (בו נבחן את המהלך הטוב ביותר על הפעם הראשונה כל פעם), הסיבוכיות תהיה  $O(\sqrt{n})$  כאשר  $n = m^d$ .

# רקע תאורטי

**פסאודו-קוד של מינימקס עם גיזום אלפא-באטא:** (הצעה ומימוש אישי)

**משתנה גלובלי:** מהלך\_לביצוע

**פונקציית מינימקס אלפא-באטא** (לוח\_משחק, שחקן\_משחק, עומק\_רקורסיה, אלפא, באטא):

אם עומק\_רקורסיה == עומק\_מקסימלי:

החזר הערכת\_מצב(לוח\_משחק)

רשימת\_מהלכים\_אפשריים[] = השג\_מהלכים\_אפשריים()

אם שחקן\_משחק == מקס:

הערכה\_נוכחית =  $-\infty$

עבור מהלך\_נוכחי מבין רשימת\_מהלכים\_אפשריים:

לוח\_חדש = בצע\_מהלך(לוח\_משחק, מהלך\_נוכחי)

הערכת\_מהלך = פונקציית\_מינימקס\_אלפא\_באטא(לוח\_חדש, מין,

עומק\_רקורסיה + 1,

אלפא, באטא)

אם הערכת\_מהלך < הערכה\_נוכחית:

הערכה\_נוכחית = הערכת\_מהלך

אם הערכה\_נוכחית < אלפא:

אלפא = הערכה\_נוכחית

אם עומק == 0:

מהלך\_לביצוע = מהלך\_נוכחי

**אם אלפא ≤ באטא:**

**שבור**

אחרת: \* שחקן\_משחק: מין \*

הערכה\_נוכחית =  $\infty$

עבור מהלך\_נוכחי מבין רשימת\_מהלכים\_אפשריים:

לוח\_חדש = בצע\_מהלך(לוח\_משחק, מהלך\_נוכחי)

הערכת\_מהלך = פונקציית\_מינימקס\_אלפא\_באטא(לוח\_חדש, מקס,

עומק\_רקורסיה + 1,

אלפא, באטא)

אם הערכת\_מהלך > הערכה\_נוכחית:

הערכה\_נוכחית = הערכת\_מהלך

אם הערכה\_נוכחית > באטא:

באטא = הערכה\_נוכחית

אם עומק == 0:

מהלך\_לביצוע = מהלך\_נוכחי

**אם אלפא ≤ באטא:**

**שבור**

החזר הערכה\_נוכחית

**קריאה לפונקציה:** פונקציית\_מינימקס(לוח\_משחק\_נוכחי, השחקן\_שזהו\_תורו, 0,  $-\infty$ ,  $\infty$ )

לוח\_משחק\_נוכחי = בצע\_מהלך(לוח\_משחק\_נוכחי, מהלך\_לביצוע)

# רקע תאורטי

## אלגוריתם תוך-מקומי

אלגוריתם תוך-מקומי הוא אלגוריתם שמשתמש בכמות קבועה וקטנה (שאינה תלויה בגודל הקלט) של הקצאת זכרון. כלומר אין שימוש במבנים זמניים משמעותיים והשינויים שיש לבצע בקלט (למשל, מערך) נעשים על גבי הקלט, ולא בנפרד. במילים אחרות: סדר גודל של  $O(1)$  זכרון נוסף. החיסכון של אלגוריתמים תוך-מקומיים מובן מאליו, אך לעתים מתלווה אליהם גם חיסכון בזמן הריצה של התוכנית. ראשית, לא יחופש מקום על גבי הערימה (שטח בזיכרון ה-RAM שעליו מוקצים משתנים שלא בזמן הידור, אלא בזמן ריצה). וכמו כן, גם הזמן שבו הזיכרון משוחרר/מנוקה נחסך. שנית, עם אלגוריתם תוך-מקומי אין צורך להעתיק את הנתונים הקיימים למקום אחר. החיסרון שמתלווה לאלגוריתמים מסוג זה הוא שהקלט משתנה ואין גישה (ישירה) לקלט כפי שהיה בעבר (למשל, מערך לא ממין לפני שעבר מיון מקומי).

## תכנות מונחה עצמים

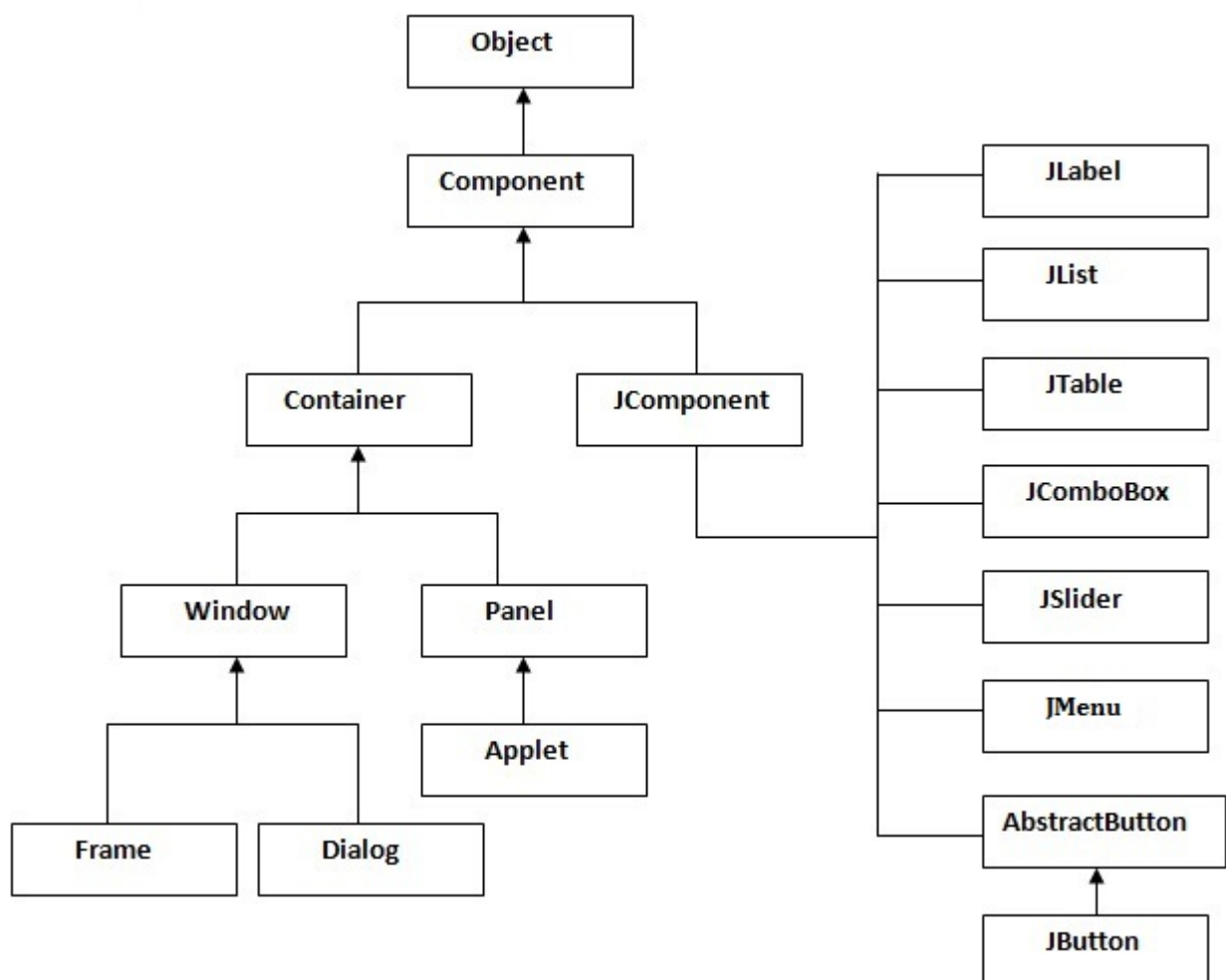
- תוכנית מונחה עצמים מספק גישה נוחה וטבעית למתכנת על ידי חלוקת עצמים לקבוצות. תוכנית מונחה עצמים ב-Java מאופיין על ידי כמה תכונות.
1. **חלוקה למחלקות** - לכל מחלקה יש תפקיד כללי ומחלקות יכולות להיעזר אחת בשנייה.
  2. **הורשה** - מחלקה יכולה "לרשת" את התכונות של מחלקה אחרת. למשל, המחלקה עט יכולה לרשת מהמחלקה כלי-כתיבה.
  3. **פולימורפיזם** - זה בעצם הסתכלות על קבוצה של עצמים שונים בתור נגזרת מהתכונות שלו. למשל ניתן להסתכל על עטים, עפרונות וטושים בתור כלי-כתיבה.
  4. **הפשטה** - רעיון מופשט שלא ניצור מופע שלו ללא פירוט ספציפי. למשל, נוכל להגדיר כלי-כתיבה כרעיון מופשט שלא ניתן ליצור מופעים שלו, אלא רק מופעים שיוורשים ממנו.
  5. **כימוס** - איגוד של כמה תכונות לעצם אחד. למשל עט מכיל תכונות של כלי כתיבה וגם יש לו תכונה של כמות דיו.

# פיתוח הפרויקט

## סביבת עבודה

כתבתי את הפרויקט בסביבת הפיתוח NetBeans באמצעות שפת Java תוך שימוש בכלים של swing. את האריחים ציירתי באמצעות MS Paint. Swing זה GUI- Graphical User Interface עבור שפת Java שמאפשר ליצור אפליקציות שמבוססות על מערכת ההפעלה של Windows תוך שימוש בעקרונות של OOP. הוא כולל הרבה כלים שימושיים כגון פאנלים, כפתורים, תפריטים ועוד הרבה.

### היררכיית Swing





# פיתוח הפרויקט

## לוח המשחק

תחילה פיתחתי ויצרתי את מסך ההגדרות, שמכיל כפתור למשחק של בן-אדם מול בן-אדם וכפתור אחר למשחק של בן-אדם מול מחשב. בנוסף, מסך ההגדרות מכיל שני Checkboxes שתפקידם לקבוע את הגדרות הלוח כפי שמוצג בעמוד 4. אחר כך הגיע חלק מאתגר שהציב בפניי דילמה: כיצד לממש את לוח המשחק.

אפשרות 1 - ניתן להסתכל על הלוח כצירוף של שתי שכבות: שכבת לוח ושכבת חיילים, הלוח יהיה תמונה סטטית אחת גדולה שתשמש כרקע לחיילים שמיקומם יקבע ביחס ללוח.

אפשרות 2 - ניתן להסתכל על הלוח בתור מטריצה של אריחים, כך שכל אריח יכול להיות במצב כלשהו מתוך מספר קבוע של מצבים (למשל חייל לבן או מלכה שחורה וכו').

אפשרות 1	אפשרות 2	
<b>יתרונות</b>  (1) ניתן לממש תזוזה הדרגתית של אריחי המשחק (2) גישה אינטואיטיבית. לוח הדמקה בנוי כשכבות במציאות, תינתן תחושת תלת-ממד.	(1) ניתן לפקח בקלות על מהלכי המשתמש, ולוודא שהקלט תקין (שהמהלכים חוקיים) (2) החישוב יתבצע על גבי הלוח ולא בנפרד. נוח מאוד!	
<b>חסרונות</b>  (1) נצטרך לחשב כל פעם מחדש מיקומים על הלוח - לא פרקטי ולא יעיל. (2) יש קושי בקבלת הקלט מהמשתמש והמרה ללוח המשחק	(1) השינוי במצבם של האריחים יקרה בבת אחת. (2) המשבצות הבהירות תופסות זיכרון מיותר.	

בחרתי באפשרות 2 מטעמים פרקטיים - גם למתכנת וגם למשתמש.

בעת לחיצה על אריח תקרא פעולה actionPerformed במסגרת ה actionListener שישנה את מצבו של האריח ואת מצב הלוח בהתאם למצבו הנוכחי של האריח. ובהתאם למצב הלוח.

# פיתוח הפרויקט

## מימוש תהליך המשחק

לאחר מימוש הלוח כפי שהצגתי, מימשתי את אופן המשחק. כל משבצת תוכל להימצא במצב אחד בכל רגע נתון מתוך 13 מצבים אפשריים כפי שמוצג בעמוד 6. הלוח ישתנה בהתאם ללחיצה על האריחים.

\* בעת לחיצה על אריח כהה או אריח בהיר או חייל מת, לא ישתנה דבר.  
\* לחיצה על חייל או מלכה שלא נבחרו, ובתנאי שהלוח לא נמצא בתהליך רצף, ובתנאי שהתור עולה בקנה אחד עם הבחירה יסומן האריח כ"נבחר" ויצבעו אריחי יעד אפשריים באדום בהתאם למצב הלוח.

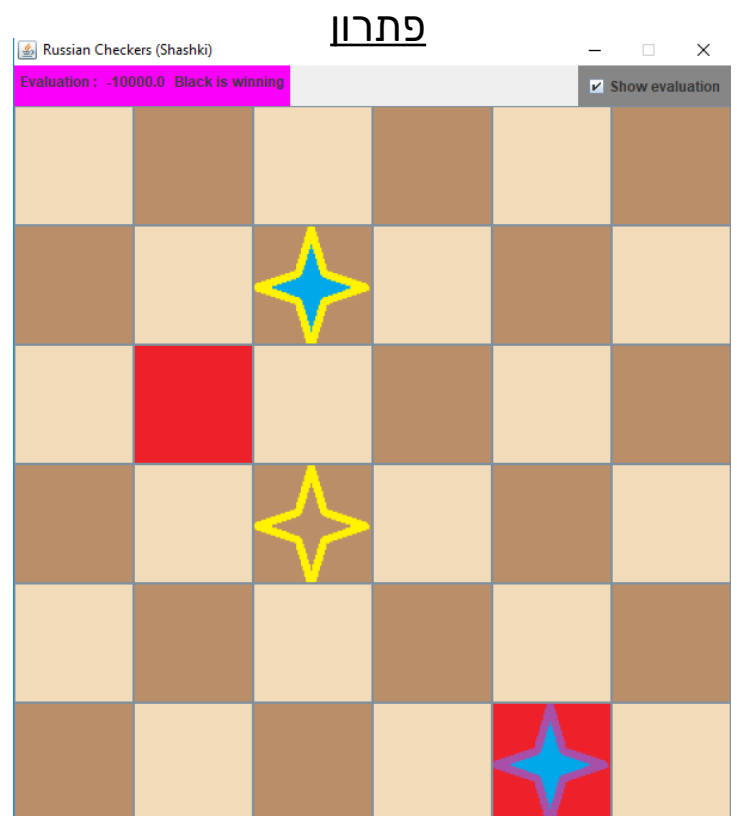
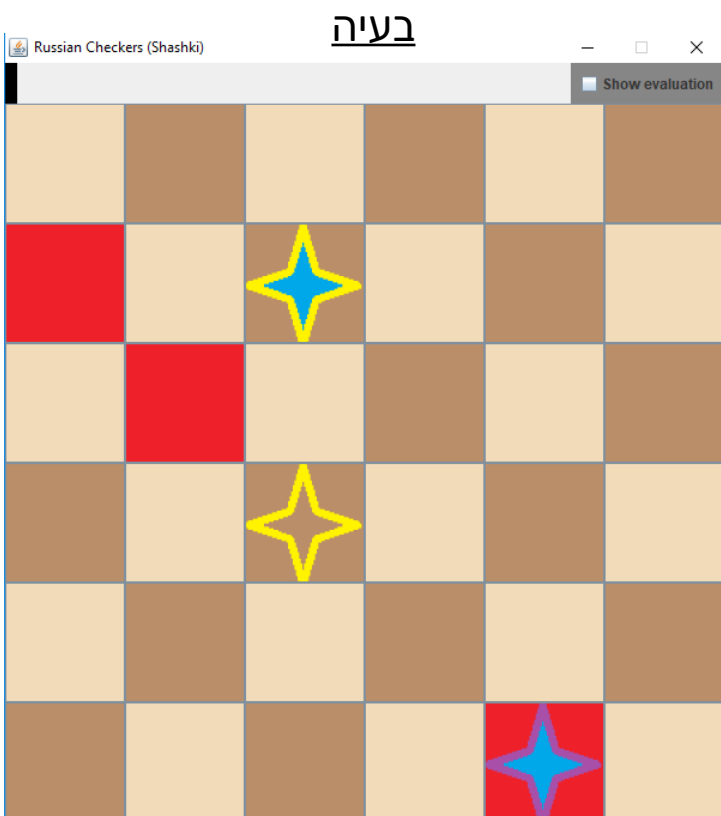
\* לחיצה על חייל או מלכה שנבחרו תבטל את בחירתם, בתנאי שהלוח לא נמצא במצב של רצף.

\*בלחיצה על אריח אדום, ישתנה הלוח כך שהאריח הנבחר יהפוך לאריח כהה וצבעו של האריח הנבחר ייקבע לצבע חדש בהתאם לצבע הריח שנבחר ובהתאם למיקומו על הלוח (למשל, תוך התחשבות בכך שחייל יכול להפוך למלכה). בנוסף, ייבדק האם נוצר רצף או שרצף קיים נמשך, ואם לא יוחלף התור, והחיילים המתים (אם התבצעה אכילה) יוסרו מהלוח.

את האמור לעיל מימשתי באמצעות פעולות, ועל ידי אחסון מידע חשוב במשתנים של המחלקה, כפי שניתן לראות בקוד המצורף. בנוסף, מומשו הבדיקות לסיום המשחק. כלומר, נבדק אם אחד הצדדים ניצח, או שיש תיקו (15 מהלכים ללא התקדמות). האריחים מומשו כמחלקה שיורשת ממחלקת כפתור. כלומר אריח הוא כפתור בעל תכונות מוספות. לוח המשחק מיוצג באמצעות מחלקה אחת שמכילה, בין היתר, מטריצה של אריחים. מטריצת האריחים מיושמת בלוח המשחק באמצעות JPanel עם GridLayout. כך שאין צורך לבצע חישובים במטרה לקבוע גודל ומיקום של כפתור. כל הפעולות מתחשבות בחוקים הרשמיים, כך שהמשתמש (או המחשב) לא יוכל לבצע מהלך שאינו חוקי. שמתי דגש על תכנות נכון- יש שימוש בקבועים והפעולות יכולות לזמן אחת את השנייה כך שאין שכפול קוד. לאחר סיום המשחק תופיע הודעה שתוכנה מכיל את תוצאת המשחק, ולאחריה המשחק ייסגר.

# פיתוח הפרויקט

כמובן, שלא הכל הלך חלק, וחווייתי במהלך פיתוח תהליך המון באגים שחלקם היו גלויים מאוד וחלקם היו נסתרים מן העין. את רובם לא תיעדתי אז אוכל לספר עליהם בעל פה. למשל התור הוחלף במהלך רצף, חלק המחילים המתים נשארו על הלוח, חיילים הופיעו על הלוח משום מקום, התוכנה לא שמה לקיומה של אכילה ואפשרה למשתמש לבצע מהלך רגיל ועוד. בעיה ספציפית שהתקשתי לפתור הייתה הוספת ריבועים אדומים למלכה שמאיימת על אריח יריב. משמאל ניתן לראות את הבעיה שניסיתי לפתור, ומימין את הפיתרון. הפיתרון הצריך Debugging. כלומר הרצתי את הקוד בשלבים בעוד שעקבתי אחר המשתנים ואחר מצב הלוח והשווייתי בין מצב הלוח לבין הציפיות שלי. כך מצאתי את המקום בקוד בו הייתה הבעיה.



# פיתוח הפרויקט

## פיתוח האינטליגנציה המלאכותית

במהלך פיתוח ה-A.I (Artificial Intelligence) חוויתי קשיים והתלבטויות. לעתים ה-A.I פשוט ביצע מהלכים גרועים, ולעתים הוא ניצח, אך בדרך לא חוקית. אני שמח שבסופו של דבר, הצלחתי לפתח A.I חכם ותקין, שאני וכל מי שניסה לא מסוגלים לנצח, ושתיקו מולו מרגיש כמו ניצחון ענק. ה-A.I מומש באמצעות פונקציית חיפוש רקורסיבית מסוג Minimax עם גיזום אלפא-באטא עם אלגוריתמיקה תוך-מקומית (הסבר מפורט על המושגים מופיע בפרק על הרקע התאורטי). תחילה מימשתי את פונקציית ההערכה. שיניתי אותה בהדרגתיות במהלך פיתוח האלגוריתם. למשל, ראיתי שהאלגוריתם מאפשר לי "לשתול" חייל עמוק בתוך שטח האויב בשלבים מוקדמים של המשחק, כך שבשלבים מאוחרים של המשחק הערך שלו גדל, ומאפשר לי לנצח. דוגמה נוספת היא שהאלגוריתם הזיז כמה חיילים מהשורה האחורית בתחילת המשחק ובעצם, יצר "דרך" שבתוך כ-10 מהלכים יתאפשר לי להפוך חייל למלכה, ושאין לו אפשרות למנוע את זה. לבסוף נותרתי עם פונקציית הערכה בעלת מספר שיקולים שנותנים לה דיוק רב, כאשר לכל שיקול משקל שונה. כל שיקול מחושב בנפרד עבור כל שחקן כך שהסימן מתחלף. למשל אם לי יש 5 חיילים וליריבי יש 3. אז:  $positionValue = 5 * 1.0 - 3 * 1.0 = 2.0$ .

### השיקולים של פונקציית ההערכה ללוח המשחק ומשקלם

שיקול	משקל
אם אין מהלכים חוקיים:	-10000
מספר חיילים	1.0 לכל חייל
מספר מלכות בתחילת המשחק	2.6 לכל מלכה
מספר מלכות בסוף המשחק	2.1 לכל מלכה
תור המשחק	0.25
אם קיימת אכילה:	0.35
חיילים בשטח אויב בתחילת משחק	-0.15 לכל חייל
חיילים בשטח אויב בסוף משחק	0.25 לכל חייל
חיילים בשורה האחורית בתחילת משחק	0.2 לכל חייל



## פיתוח הפרויקט

פיתוח פונקציית החיפוש הרקורסיבי מינימקס עם גיזום אלפא-באטא נעשה בשלבים. לאחר בניית השלד של הפונקציה, ולאחר טיפול בפונקציות מחלקת לוח המשחק, כך שיתאימו גם לשימוש על ידי ה A.I, בחנתי אפשרויות שונות לשיפור האלגוריתם, את חלקן שללתי, וחלקן מצאו חן בעיני. לגבי גיזום האלפא-באטא לא הייתה שאלה בכלל כי אין בו שום חסרון. אפשרות ששקלתי הייתה Zobrist hashing, keys - אלגוריתם זה מתחיל במתן מספר ערכים אקראיים לכל משבצת. ובעבור מצב לוח מסוים שלא נתקלנו בו, האלגוריתם נותן לו ערך על ידי ביצוע פעולת xor בין הערכים המתאימים בלוח (לפי איזה כלי נמצא על איזה משבצת), הערך הניתן משמש כמפתח בטבלת גיבוב (Hash Table) והערכים הנשמרים הם המהלך שיש לשחק, הערכת המצב באמצעות פעולת החיפוש, וערכי האלפא והבאטא. ואם ניתקל במצב לוח זהה באחד החישובים בעתיד נוכל לגשת לטבלת הגיבוב ולדלות את המידע הרלוונטי.

### האם להשתמש ב Zobrist hashing?

יתרונות	חסרונות
1) קל למימוש- האלגוריתם פשוט ו Java מספקת טבלאות גיבוב מובנות ופעולת xor (^) כחלק מפעולות ה Bit Manipulation. 2) יש פוטנציאל לחסכון רב בזמן הריצה של התוכנית.	1) טבלת הגיבוב בוודאות תיקח זכרון רב שלרובו לעולם לא יהיה שימוש. 2) יש הסתברות נמוכה להתנגשות בתהליך ה hashing ובכך יוחזרו פרטים לא נכונים על מצב הלוח, דבר שיכול לגרום למחשב לבצע מהלך שגוי ובכך להפסיד את המשחק!

בחרתי שלא לממש Zobrist hashing כי לדעתי החסרונות גוברים על היתרונות. ישנם עוד אלגוריתמים שבעצם ממשים טבלאות גיבוב, אך בכולן קיימים החסרונות שהצגתי, ובחלקם עלול להתבזבז יותר זמן ממה שיכול להיחסך. בנוסף, אם המצבים שמופיעים פעמיים הם חלק מאותו חיפוש לעומק אז רובו של החיפוש של המצב השני ייחסך באמצעות גיזום אלפא-באטא.

## פיתוח הפרויקט

בשלב זה של הפרויקט חיפשתי דרכים יצירתיות וחדשניות לשפר את הפרויקט שלי. בדיקה בגוגל מגלה שאני לא הראשון שמימש דמקה באמצעות מינימקס עם גיזום אלפא-באטא. לאחר חיפוש מעמיק אחר חסרונות שניתן לשפר באלגוריתם, הגעתי למסקנה שהאלגוריתם, שבאופן מסורתי הוא חוץ-מקומי מבזבז הרבה זכרון וגם זמן. עבור כל מהלך ליצור לוח חדש, שרובו זהה בדיוק, וגם מהלוח הזה ליצור עוד לוח באופן רקורסיבי מצריך זכרון רב וזמן להעתקה. בנוסף, גורם ל Garbage Collector (שחרור זכרון שהוקצה באופן דינמי מהערמה אוטומטית) לעבוד שעות נוספות. נראה כי אין מה לעשות, שהרי עם נבצע את האלגוריתם באופן תוך-מקומי, כלומר על גבי הלוח, לוח המשחק ייהרס, והתוכנה תקרוס. הפתרון שאני הגיתי הוא לשמור אך ורק את המהלכים שנעשו, ולאחר חישוב ערכו של לוח משחק כלשהו נוכל לשחזר את הלוח למצב שהיה בו טרם שוחק המהלך. תהליך השחזור לא פשוט כלל, וכנראה זו הסיבה שאף אחד אחר לא הצליח לעשות פעולת חיפוש מסוג **מינימקס באופן תוך-מקומי** עבור דמקה למיטב ידיעתי. שמחתי שיש מקום בפרויקט שבו אני יכול לתת ביטוי ליצירתיות ולחדשנות. לאחר חיפוש אחר מבנה נתונים שישמור על המהלכים שביצעתי, חשבתי ליצור **עץ** שכל שכבה בו מתארת את מכלול המהלכים שיש להריץ. ולאחר ההרצה איתור ההורה של אותו צומת וחזרה אליו. פתרון זה לא הרגיש לי נכון ולאחר חשיבה מעמיקה גיליתי את הפתרון המושלם: **מחסנית**. תתבצע דחיפה (Push) עבור כל מהלך שעושים, ולאחר שחושב, נבצע שליפה (Pop) ונעשה לאותו מהלך שחזור. זהו פתרון טבעי מאוד מכיוון שגם הקריאות הרקורסיביות ממושות על ידי מחסנית (מחסנית הקריאות). הפתרון הצריך ליצור מבנה חדש (מומש על ידי מחלקה פנימית), שישמש איבר למחסנית, שייצג מהלך, ומסתבר שצריך לשמור על המהלך פרטים כלל לא מובנים מאליהם כמו למשל, האם המהלך היה שינוי של חייל למלכה, או, האם המהלך היה חלק מרצף, וגם רשימת כל האריחים המתים שהיו על הלוח לפני ביצוע המהלך, כדי שהשחזור יהיה מדויק. אחרת הלוח יהיה מלא אריחים מתים ומלכות שהופיעו משום מקום. לאחר הרבה ניסיונות וכישלונות הצלחתי לממש את חיפוש המינימקס באופן תוך-מקומי בעזרת מחסנית. זה עובד כל כך טוב שלא נותר אלא לתהות איך לא עשו את זה לפני.

# פיתוח הפרויקט

## הערכת מצב

בתור חובב שחמט שמשחק לפעמים עם יריבים דרך האינטרנט, שמתי לב שכאשר מנתחים את המשחק לאחר סיומו יש אפשרות לקבל את חוות דעתה של אינטליגנציה מלאכותית לגבי מצב לוח כלשהו. אם המשתמש רוצה בזאת, יופיע בפינת המסך מספר המסמל את מצב הלוח, ככל שהמספר גדול יותר כל ה"לבן" מוביל וככל שהמספר קטן יותר ה"שחור" מוביל. כאשר עלתה בראשי המחשבה על להוסיף את הפיצ'ר למשחק, מיד הבנתי שאני יכול להשתמש באותה פעולה שכבר מימשתי על מנת למצוא את המהלך הטוב ביותר (חיפוש המינימקס הרקורסיבי), ושאני כמעט לא צריך לכתוב קוד בשביל להוסיף את האופציה. מצד שני דאגתי שמא המשתמש לא ירצה לדעת מה מצבו לפי דעת המחשב, לכן הוספתי Checkbox שרק אם יסומן תינתן הערכת מצב עבור הלוח הנוכחי. הפיצ'ר השתלב מעולה במשחק ומצא חן בעיני כל מי ששיחק. בנוסף, על מנת שהתוכנית תהיה ברורה ומובנת לכל משתמש, עד כמה שאפשר, הוספתי ליד המספר משפט באנגלית המתאר את מצב הלוח, ברקע צבע מתאים (לבן, צהוב כהה, צהוב בהיר, טורקיז, סגול). המשפט נקבע בהתאם להערכה, כפי שמתואר בתמונות בעמוד זה ובעמוד הבא.

 Show evaluation

Evaluation : 0.6 White is slightly leading

Evaluation : -0.75 Black is slightly leading

Evaluation : 1.85 White is winning

Evaluation : -10000.0 Black is winning

Evaluation : 0.25 Position is roughly equal

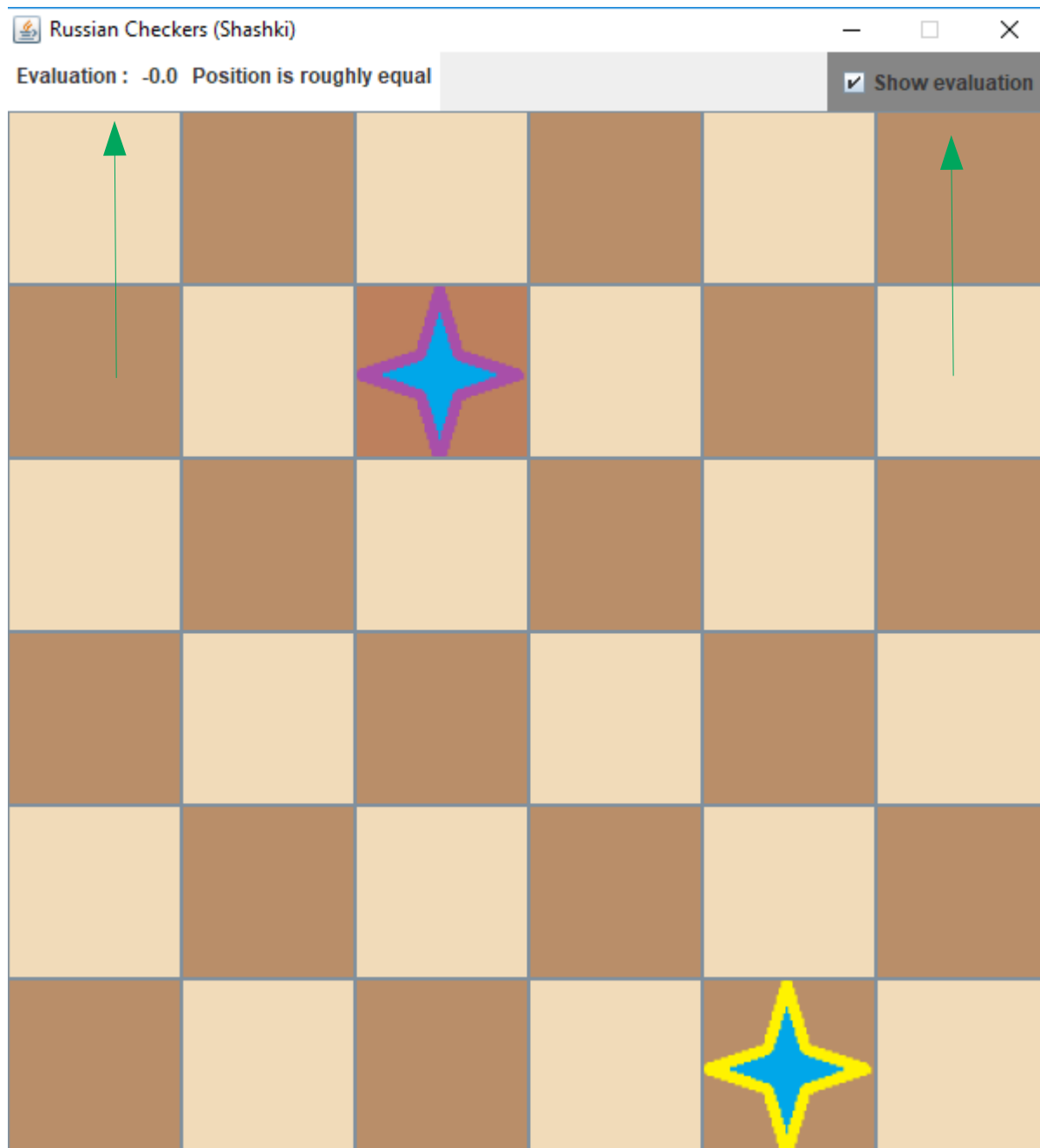
# פיתוח הפרויקט

נסמן את ההערכה ב  $v$ .

$|v| < 0.6 \rightarrow$  Position is roughly equal (לאף צד אין יתרון משמעותי)

$0.6 \geq |v| < 1.5 \rightarrow$  Someone is slightly leading (יתרון קל לצד כלשהו)

$|v| \geq 1.5 \rightarrow$  Someone is winning (יתרון משמעותי לצד כלשהו)



# קוד הפרויקט

## MyProject

<https://github.com/DanielK1907/MyProject>

```
1 package myproject;
2
3 import java.awt.Color;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import javax.swing.*;
7 import static javax.swing.JFrame.EXIT_ON_CLOSE;
8
9
10 /**
11  * "Abstract strategy game"
12  * @author Daniel Kanevsky
13  */
14 public class MyProject {
15
16     /**
17      * @param args the command line arguments
18      */
19     public static void main(String[] args) {
20
21         //<editor-fold defaultstate="collapsed" desc="Settings JFrame">
22
23         // Instantiate the comboboxes
24         JFrame f = new JFrame("Settings");
25         JComboBox lengthBox = new JComboBox();
26         JComboBox pawnRowsBox = new JComboBox();
27         lengthBox.setBounds(80, 90, 60, 50);
28         pawnRowsBox.setBounds(255, 90, 60, 50);
29
30         // Instantiate the labels
31         JLabel lengthLabel = new JLabel("Board Length :");
32         JLabel pawnRowsLabel = new JLabel("Pawns Rows :");
33         lengthLabel.setBounds(70, 60, 130, 30);
34         pawnRowsLabel.setBounds(245, 60, 120, 30);
35
36         // instantiate the buttons
37         JButton humanB = new JButton("Human Vs. Human");
38         humanB.setBounds(50, 10, 300, 20);
39         humanB.addActionListener(new ActionListener()
40         {
41             @Override
42             public void actionPerformed(ActionEvent ae) {
43                 Damka d;
44                 if (lengthBox.getSelectedItem() != null & pawnRowsBox.getSelectedItem() != null)
45                     d = new Damka((int)lengthBox.getSelectedItem(), (int)pawnRowsBox.getSelectedItem());
46                 else // Go for the classic variation
47                     d = new Damka(8, 3);
48                 Computer.comp.board = d;
49                 d.isComputer = false;
50                 d.start();
51             }
52         });
53     }
54 }
```

# קוד הפרויקט

```
51         f.dispose();
52     }
53
54 });
55
56 JButton computerB = new JButton("Human Vs. Computer");
57 computerB.setBounds(50, 40, 300, 20);
58 computerB.addActionListener(new ActionListener()
59 {
60     @Override
61     public void actionPerformed(ActionEvent ae) {
62         Damka d;
63         if (lengthBox.getSelectedItem() != null & pawnRowsBox.getSelectedItem() != null)
64             d = new Damka((int)lengthBox.getSelectedItem(), (int)pawnRowsBox.getSelectedItem());
65         else // Go for the classic variation
66             d = new Damka(8, 3);
67         Computer.comp.board = d;
68         d.isComputer = true;
69         d.start();
70         f.dispose();
71     }
72
73 });
74
75 for (int i = 4; i <= 12; i++)
76     lengthBox.addItem(i);
77
78 lengthBox.addActionListener(new ActionListener() {
79     @Override
80     public void actionPerformed(ActionEvent ae) {
81         pawnRowsBox.removeAllItems();
82         for (int i = 1; i < ((int)lengthBox.getSelectedItem()+1)/2; i++)
83             pawnRowsBox.addItem(i);
84     }
85 });
86
87
88 f.add(lengthBox);
89 f.add(pawnRowsBox);
90 f.add(lengthLabel);
91 f.add(pawnRowsLabel);
92 f.setSize(400, 360);
93 f.add(humanB);
94 f.add(computerB);
95 f.setLayout(null);
96 f.setDefaultCloseOperation(EXIT_ON_CLOSE);
97 f.setResizable(false);
98 f.setLocationRelativeTo(null); // Passing null centers the form!
99 f.setVisible(true);
100 f.getContentPane().setBackground(new Color(50, 155, 105));
101 //</editor-fold>
102
103 }
104 }
```

# קוד הפרויקט

## DamkaTile

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package myproject;
7
8  /**
9   *
10   * @author Daniel Kanevsky
11   */
12  import javax.swing.*;
13  import java.awt.event.*;
14  import java.awt.*;
15
16  public class DamkaTile extends JButton implements ActionListener{
17
18      public enum TilePawn
19      {
20          WHITE,
21          BLACK,
22          WHITE_PAWN,
23          BLACK_PAWN,
24          WHITE_PAWN_CHOSEN,
25          BLACK_PAWN_CHOSEN,
26          RED,
27          WHITE_QUEEN,
28          BLACK_QUEEN,
29          WHITE_QUEEN_CHOSEN,
30          BLACK_QUEEN_CHOSEN,
31          DEAD_WHITE,
32          DEAD_BLACK
33      }
34
35
36      private static final int TOTAL_ICONS = 13;
37      private final Damka board;
38      public final int row;
39      public final int col;
40      public int color;
41
42      public static Dimension buttonSize= new Dimension(Damka.TILE_SIZE, Damka.TILE_SIZE);
43      public static ImageIcon[] Images = new ImageIcon[TOTAL_ICONS];
44
45      /**
46       * Initialize the images array with images.
```

# קוד הפרויקט

```
47 */
48 public void initialize()
49 {
50     //<editor-fold defaultstate="collapsed" desc="load content">
51     Images[TilePawn.WHITE.ordinal()] =
52         new ImageIcon(this.getClass().getResource("Images/white.png"));
53     Images[TilePawn.BLACK.ordinal()] =
54         new ImageIcon(this.getClass().getResource("Images/black.png"));
55     Images[TilePawn.WHITE_PAWN.ordinal()] =
56         new ImageIcon(this.getClass().getResource("Images/white pawn.png"));
57     Images[TilePawn.BLACK_PAWN.ordinal()] =
58         new ImageIcon(this.getClass().getResource("Images/black pawn.png"));
59     Images[TilePawn.WHITE_PAWN_CHOSEN.ordinal()] =
60         new ImageIcon(this.getClass().getResource("Images/white pawn chosen.png"));
61     Images[TilePawn.BLACK_PAWN_CHOSEN.ordinal()] =
62         new ImageIcon(this.getClass().getResource("Images/black pawn chosen.png"));
63     Images[TilePawn.RED.ordinal()] =
64         new ImageIcon(this.getClass().getResource("Images/red.png"));
65     Images[TilePawn.WHITE_QUEEN.ordinal()] =
66         new ImageIcon(this.getClass().getResource("Images/white queen.png"));
67     Images[TilePawn.BLACK_QUEEN.ordinal()] =
68         new ImageIcon(this.getClass().getResource("Images/black queen.png"));
69     Images[TilePawn.WHITE_QUEEN_CHOSEN.ordinal()] =
70         new ImageIcon(this.getClass().getResource("Images/white queen chosen.png"));
71     Images[TilePawn.BLACK_QUEEN_CHOSEN.ordinal()] =
72         new ImageIcon(this.getClass().getResource("Images/black queen chosen.png"));
73     Images[TilePawn.DEAD_WHITE.ordinal()] =
74         new ImageIcon(this.getClass().getResource("Images/dead white.png"));
75     Images[TilePawn.DEAD_BLACK.ordinal()] =
76         new ImageIcon(this.getClass().getResource("Images/dead black.png"));
77     //</editor-fold>
78
79     Image buffer;
80
81     // Stretch the image across the button
82     for (int i = 0; i < TOTAL_ICONS; i++)
83     {
84         buffer = Images[i].getImage();
85         buffer = buffer.getScaledInstance(Damka.TILE_SIZE, Damka.TILE_SIZE, Image.SCALE_SMOOTH);
86         Images[i].setImage(buffer);
87     }
88
89 }
90
91 public DamkaTile(){row = -1; col = -1; board = null;} // Technical only
92 public DamkaTile(int row, int col, Damka board)
```



# קוד הפרויקט

```
93 {
94     setPreferredSize(buttonSize);
95     this.board = board;
96     this.row = row;
97     this.col = col;
98
99     addActionListener(this);
100 }
101
102 @Override
103 public void actionPerformed(ActionEvent e)
104 {
105     if (color == TilePawn.WHITE.ordinal() ||
106         color == TilePawn.BLACK.ordinal() ||
107         color == TilePawn.DEAD_BLACK.ordinal() ||
108         color == TilePawn.DEAD_WHITE.ordinal())
109         return;
110
111     if (color == TilePawn.WHITE_PAWN.ordinal() ||
112         color == TilePawn.BLACK_PAWN.ordinal())
113     {
114         if (board.isOnStreak)
115             return;
116
117         if (board.isPawnChosen)
118         {
119             board.turnPawnOff();
120         }
121         board.turnPawnOn(this);
122         return;
123     }
124
125     if (color == TilePawn.WHITE_PAWN_CHOSEN.ordinal() ||
126         color == TilePawn.BLACK_PAWN_CHOSEN.ordinal())
127     {
128         if (board.isOnStreak)
129             return;
130         board.turnPawnOff();
131         return;
132     }
133
134     if (color == TilePawn.WHITE_QUEEN_CHOSEN.ordinal() ||
135         color == TilePawn.BLACK_QUEEN_CHOSEN.ordinal())
136     {
137         if (board.isOnStreak)
138             return;
139         board.turnQueenOff();
140     }
141 }
```

# קוד הפרויקט

```
132
133     if (color == TilePawn.WHITE_QUEEN_CHOSEN.ordinal() ||
134         color == TilePawn.BLACK_QUEEN_CHOSEN.ordinal())
135     {
136         if (board.isOnStreak)
137             return;
138         board.turnQueenOff();
139         return;
140     }
141
142     if (color == TilePawn.RED.ordinal())
143     {
144         if (!board.isForced)
145         {
146             int tileColor = board.tiles[board.chosenPawnRow][board.chosenPawnCol].color;
147             if (tileColor == TilePawn.WHITE_QUEEN_CHOSEN.ordinal() ||
148                 tileColor == TilePawn.BLACK_QUEEN_CHOSEN.ordinal())
149             {
150                 board.moveQueen(this);
151                 return;
152             }
153
154             board.movePawn(this);
155             return;
156         }
157
158         board.Capture(this);
159         return;
160     }
161
162     // if reached here: color must be black or white queen
163     if (board.isOnStreak)
164         return;
165
166     if (board.isPawnChosen)
167         board.turnPawnOff();
168
169     board.turnQueenOn(this);
170 }
171
172
173 public void setColor(int color)
174 {
175     this.color = color;
176     setIcon(Images[color]);
177 }
178
179
```

# קוד הפרויקט

## Damka

```
1 package myproject;
2
3 import java.awt.*;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ItemEvent;
7 import java.awt.event.ItemListener;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 import javax.swing.*;
11
12 /**
13  * In this frame the game play takes place, including evaluation
14  * @author Daniel Kanevsky
15  */
16 public class Damka extends JFrame {
17
18     //<editor-fold defaultstate="collapsed" desc="Constants">
19     // Height and width of the board
20     public final int LENGTH;
21     // Number of rows filled with pawns for each side
22     public final int PAWN_ROWS;
23
24     // Tile length in pixels
25     public static int TILE_SIZE;
26
27     // The color purple - used in evaluation
28     public static final Color PURPLE = new Color(250, 0, 250);
29
30     // Number of moves without captures or pawn pushes needed for a draw
31     public static final int MOVES_FOR_DRAW = 15;
32
33     // Maximum amount of same-color pawns in a position where player can't move
34     public static final int MAX_PAWNS_IN_STALEMATE = 5;
35
36     /**
37      * Direction vector for possible direction the queen can move
38      */
39     public static final int[][] QUEEN_DIRS = {
40         {1, 1},
41         {1, -1},
42         {-1, 1},
43         {-1, -1}
44     };
45
46     // Possible messages for evaluation
47     private static final String[] STATES = new String[]
48     {
49         " Position is roughly equal",
50         " White is slightly leading",
51         " Black is slightly leading",
52         " White is winning",
53         " Black is winning"
54     };
```

# קוד הפרויקט

```
55 //</editor-fold>
56
57 //<editor-fold defaultstate="collapsed" desc="Variables">
58 // All the game board tiles
59 public DamkaTile[][] tiles;
60
61 public int whitePawnsLeft;
62 public int blackPawnsLeft;
63 public int whiteQueens = 0;
64 public int blackQueens = 0;
65 public int chosenPawnRow;
66 public int chosenPawnCol;
67 public int movesWithoutProgress = 0;
68 public boolean isPawnChosen = false;
69 public boolean isForced = false; // Is the player forced to make a capture
70 public boolean isOnStreak = false; // Is the player in capture streak
71 public boolean isComputer; // Does a computer play in this game
72 public boolean turn = false; // false - white to play; true - black to play
73 public boolean isComputerPlaying = false; // The computer isn't playing in the first move
74
75 //<editor-fold defaultstate="collapsed" desc="JPanels">
76 public final JPanel gamePanel = new JPanel(true); // Contains the board
77 private final JPanel evalPanel = new JPanel(); // Contains the evaluation
78 private final JPanel showEvalPanel = new JPanel(); // Contains the checkbox
79 private final JPanel buffer = new JPanel(); // A buffer between "evalPanel" & "showEvalPanel"
80 //</editor-fold>
81
82
83 //<editor-fold defaultstate="collapsed" desc="Other JComponents">
84 private final JCheckBox showEval = new JCheckBox("Show evaluation");
85
86 private final JLabel evalMsg = new JLabel("Evaluation : ");
87 private JLabel evaluation = new JLabel();
88 private JLabel stateMsg = new JLabel();
89 //</editor-fold>
90
91 private Color winningColor = Color.WHITE; // Background color for evaluation
92 //</editor-fold>
93
94 /**
95  * Create a new Damka object (Board game as a frame)
96  * Initialize the frame with panels and JComponents
97  * @param length: Length of the board (width and height)
98  * @param pawnRows: Number of pawn rows each side has to begin with
99  */
100 public Damka(int length, int pawnRows) {
101     super("Russian Checkers (Shashki)");
102
103     LENGTH = length;
104     PAWN_ROWS = pawnRows;
105     TILE_SIZE = 600/LENGTH;
106
107     tiles = new DamkaTile[LENGTH][LENGTH];
108     whitePawnsLeft = LENGTH * PAWN_ROWS / 2;
```

# קוד הפרויקט

```
109 blackPawnsLeft = LENGTH * PAWN_ROWS / 2;
110
111 setSize(TILE_SIZE * LENGTH, TILE_SIZE * LENGTH + 60);
112
113 gamePanel.setLayout(new GridLayout(LENGTH, LENGTH, 0, 0)); // default value is 0 anyway
114 showEvalPanel.setBackground(Color.GRAY);
115
116 showEval.setBackground(Color.GRAY);
117 showEval.setSelected(true);
118 showEval.addItemListener(new ItemListener() {
119     @Override
120     public void itemStateChanged(ItemEvent ie) {
121         evalMsg.setVisible(!evalMsg.isVisible());
122         evaluation.setVisible(!evaluation.isVisible());
123         stateMsg.setVisible(!stateMsg.isVisible());
124         if (ie.getStateChange() == 1)
125             evaluate();
126         else
127             evalPanel.setBackground(Color.BLACK);
128     }
129 });
130
131 // Add JComponents to panels
132 evalPanel.add(evalMsg);
133 evalPanel.add(evaluation);
134 evalPanel.add(stateMsg);
135 showEvalPanel.add(showEval);
136
137 // Add panels to the frame
138 add(buffer);
139 add(evalPanel, BorderLayout.WEST);
140 add(showEvalPanel, BorderLayout.EAST);
141 add(gamePanel, BorderLayout.SOUTH);
142 }
143
144 /**
145  * Initialize the game panel with pawns in the corrent colors
146  */
147 public void start()
148 {
149     //<editor-fold defaultstate="collapsed" desc="create tiles and add them to the game panel">
150     for (int i = 0; i < LENGTH; i++) {
151         for (int j = 0; j < LENGTH; j++) {
152             tiles[i][j] = new DamkaTile(i, j, this);
153             gamePanel.add(tiles[i][j]);
154         }
155     }
156     //</editor-fold>
157
158     // Initialize Images array with the possible images
159     (new DamkaTile()).initialize();
160
161     //<editor-fold defaultstate="collapsed" desc="set pawn colors">
162     // set white tales
```

# קוד הפרויקט

```
163 for (int i = 0; i < LENGTH; i++) {
164     for (int j = i % 2; j < LENGTH; j += 2) {
165         tiles[i][j].setColor(DamkaTile.TilePawn.WHITE.ordinal());
166     }
167 }
168
169 // set the white pawns
170 for (int i = LENGTH - PAWN_ROWS; i < LENGTH; i++) {
171     for (int j = 1 - i % 2; j < LENGTH; j += 2) {
172         tiles[i][j].setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
173     }
174 }
175
176 // set the black pawns
177 for (int i = 0; i < PAWN_ROWS; i++) {
178     for (int j = 1 - (i % 2); j < LENGTH; j += 2) {
179         tiles[i][j].setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
180     }
181 }
182
183 // set the black remaining slots
184 for (int i = PAWN_ROWS; i < LENGTH - PAWN_ROWS; i++) {
185     for (int j = 1 - i % 2; j < LENGTH; j += 2) {
186         tiles[i][j].setColor(DamkaTile.TilePawn.BLACK.ordinal());
187     }
188 }
189 //</editor-fold>
190
191
192 // Frame settings
193 setDefaultCloseOperation(EXIT_ON_CLOSE);
194 setResizable(false);
195 setLocationRelativeTo(null); // center the form
196 pack();
197 setVisible(true);
198
199
200 //evaluate(); // evaluate initial value of position
201 }
202
203 /**
204  * Turn the chosen pawn off, as well as all the red squares.
205  * If the pawn is A queen, call turnQueenOff()
206  */
207 public void turnPawnOff() {
208     if (tiles[chosenPawnRow][chosenPawnCol].color == DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal() ||
209         tiles[chosenPawnRow][chosenPawnCol].color == DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal())
210     {
211         turnQueenOff();
212         return;
213     }
214
215     if (turn)
216         tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
```

# קוד הפרויקט

```
217     else
218     |   tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
219
220     turnRedSquaresOff();
221     isPawnChosen = false;
222   }
223
224   /**
225   * Turn the chosen queen off, as well as all the red squares
226   */
227   public void turnQueenOff()
228   {
229     |   if (!turn) {
230     |   |   tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
231     |   } else {
232     |   |   tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
233     |   }
234
235     turnRedSquaresOff();
236     isPawnChosen = false;
237   }
238
239   /**
240   * Turn a pawn on:
241   * 1) set chosen pawn properties to tile properties
242   * 2) change the pawn color
243   * 3) turn on the red pawn squares
244   * @param tile: the pawn to turn on
245   */
246   public void turnPawnOn(DamkaTile tile)
247   {
248     |   chosenPawnRow = tile.row;
249     |   chosenPawnCol = tile.col;
250     |   isPawnChosen = true;
251
252     // turn the pawn on if it is the correct turn
253     if (tile.color == DamkaTile.TilePawn.WHITE_PAWN.ordinal()) {
254       |   if (!turn) {
255       |   |   tile.setColor(DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal());
256       |   |   turnRedPawnSquaresOn();
257       |   } else
258       |   |   isPawnChosen = false;
259     } else {
260       |   if (turn) {
261       |   |   tile.setColor(DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal());
262       |   |   turnRedPawnSquaresOn();
263       |   } else {
264       |   |   isPawnChosen = false;
265       |   }
266     }
267   }
268
269   /**
270   *
```

# קוד הפרויקט

```
271 * @param tile: the queen to turn on.
272 * 1) set chosen pawn properties to tile properties
273 * 2) change the pawn color
274 * 3) turn on the red queen squares
275 */
276 public void turnQueenOn(DamkaTile tile)
277 {
278     chosenPawnRow = tile.row;
279     chosenPawnCol = tile.col;
280     isPawnChosen = true;
281
282     // turn the queen on if it is the correct turn
283     if (tile.color == DamkaTile.TilePawn.WHITE_QUEEN.ordinal()) {
284         if (!turn) {
285             tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal());
286             turnRedQueenSquaresOn();
287         } else {
288             isPawnChosen = false;
289         }
290     } else {
291         if (turn) {
292             tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal());
293             turnRedQueenSquaresOn();
294         } else {
295             isPawnChosen = false;
296         }
297     }
298 }
299
300 /**
301  * Turn the red pawn squares according to the chosen pawn.
302  * Function considers the state of the board (forced or not)
303  */
304 public void turnRedPawnSquaresOn()
305 {
306     if (!isForced) {
307         int direction = (turn) ? 1 : -1;
308         if (tiles[chosenPawnRow][chosenPawnCol].col != 0)
309             if (tiles[chosenPawnRow + direction][chosenPawnCol - 1].color ==
310                 DamkaTile.TilePawn.BLACK.ordinal())
311                 tiles[chosenPawnRow + direction][chosenPawnCol - 1].setColor(DamkaTile.TilePawn.RED.ordinal());
312         if (tiles[chosenPawnRow][chosenPawnCol].col != LENGTH - 1)
313             if (tiles[chosenPawnRow + direction][chosenPawnCol + 1].color ==
314                 DamkaTile.TilePawn.BLACK.ordinal())
315                 tiles[chosenPawnRow + direction][chosenPawnCol + 1].setColor(DamkaTile.TilePawn.RED.ordinal());
316     }
317     else
318     {
319         int colorEaten1;
320         int colorEaten2;
321
322         // Determine the colors that are searched
323         if (tiles[chosenPawnRow][chosenPawnCol].color == DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal())
324         {
```



# קוד הפרויקט

```
325         colorEaten1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
326         colorEaten2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
327     }
328     else
329     {
330         colorEaten1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
331         colorEaten2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
332     }
333
334     // Make sure this index isn't out of bounds and compare the tile color to the colors searched
335     if (tiles[chosenPawnRow][chosenPawnCol].col >= 2 && tiles[chosenPawnRow][chosenPawnCol].row >= 2)
336     {
337         if (tiles[chosenPawnRow - 2][chosenPawnCol - 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
338             ( tiles[chosenPawnRow - 1][chosenPawnCol - 1].color == colorEaten1 ||
339               tiles[chosenPawnRow - 1][chosenPawnCol - 1].color == colorEaten2))
340             tiles[chosenPawnRow - 2][chosenPawnCol - 2].setColor(DamkaTile.TilePawn.RED.ordinal());
341     }
342     if (tiles[chosenPawnRow][chosenPawnCol].col >= 2 && tiles[chosenPawnRow][chosenPawnCol].row <= LENGTH - 3)
343     {
344         if (tiles[chosenPawnRow + 2][chosenPawnCol - 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
345             (tiles[chosenPawnRow + 1][chosenPawnCol - 1].color == colorEaten1 ||
346               tiles[chosenPawnRow + 1][chosenPawnCol - 1].color == colorEaten2))
347             tiles[chosenPawnRow + 2][chosenPawnCol - 2].setColor(DamkaTile.TilePawn.RED.ordinal());
348     }
349     if (tiles[chosenPawnRow][chosenPawnCol].col <= LENGTH - 3 && tiles[chosenPawnRow][chosenPawnCol].row >= 2)
350     {
351         if (tiles[chosenPawnRow - 2][chosenPawnCol + 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
352             (tiles[chosenPawnRow - 1][chosenPawnCol + 1].color == colorEaten1 ||
353               tiles[chosenPawnRow - 1][chosenPawnCol + 1].color == colorEaten2))
354             tiles[chosenPawnRow - 2][chosenPawnCol + 2].setColor(DamkaTile.TilePawn.RED.ordinal());
355     }
356     if (tiles[chosenPawnRow][chosenPawnCol].col <= LENGTH - 3 && tiles[chosenPawnRow][chosenPawnCol].row <= LENGTH - 3)
357     {
358         if (tiles[chosenPawnRow + 2][chosenPawnCol + 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
359             (tiles[chosenPawnRow + 1][chosenPawnCol + 1].color == colorEaten1 ||
360               tiles[chosenPawnRow + 1][chosenPawnCol + 1].color == colorEaten2))
361             tiles[chosenPawnRow + 2][chosenPawnCol + 2].setColor(DamkaTile.TilePawn.RED.ordinal());
362     }
363 }
364
365 /**
366  * Turn the red queen squares according to the chosen pawn.
367  * Function considers the state of the board (forced or not)
368  * Moreover, the function checks wheter A streak is possible.
369  * If a streak is possible the function removes all red squares which
370  * do not lead to a streak. (Based on the official rules)
371  */
372 public void turnRedQueenSquaresOn()
373 {
374     DamkaTile enemy = null; // The enemy tile in the current iteration
375     int enemyColor = -999; // The color of the enemy pawn found (used as a 'temp' variable)
376     int currentColor; // Current color index of tile checked
377     int currentRowIndex; // Current row index of tile checked
378     int currentColIndex; // Current col index of tile checked
379     int colorEaten1; // Pawn color to eat
380     int colorEaten2; // Queen color to eat
381     boolean enemyFound; // Did we came across enemy tile in the current iteration
382     boolean doubleFound; // Did we find double option in the current iteration
383
384     // Determine the "pray" colors in case position is "must-capture"
385     if (turn)
```

# קוד הפרויקט

```
379 {
380     colorEaten1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
381     colorEaten2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
382 }
383 else
384 {
385     colorEaten1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
386     colorEaten2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
387 }
388
389 for (int i = 0; i < 4; i++)
390 {
391     doubleFound = false;
392     enemyFound = false;
393     currentRowIndex = chosenPawnRow + QUEEN_DIRS[i][0];
394     currentColIndex = chosenPawnCol + QUEEN_DIRS[i][1];
395     if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
396         currentColIndex == -1 || currentColIndex == LENGTH)
397         continue;
398     currentColor = tiles[currentRowIndex][currentColIndex].color;
399     if ((currentColor == colorEaten1 || currentColor == colorEaten2) && isForced)
400     {
401         enemyFound = true;
402         enemy = tiles[currentRowIndex][currentColIndex];
403         enemyColor = enemy.color;
404         currentRowIndex += QUEEN_DIRS[i][0];
405         currentColIndex += QUEEN_DIRS[i][1];
406
407         if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
408             currentColIndex == -1 || currentColIndex == LENGTH)
409             continue;
410         currentColor = tiles[currentRowIndex][currentColIndex].color;
411     }
412
413     // Loop will be broken when there is no possibility of finding a red square
414     // in the current direction.
415     while (true)
416     {
417         if (!isForced && currentColor != DamkaTile.TilePawn.BLACK.ordinal())
418             break;
419         if (isForced && currentColor != DamkaTile.TilePawn.BLACK.ordinal() && enemyFound)
420             break;
421
422         // If The current color is black and either a capture exists and enemy found
423         // or enemy isn't found and there is no capture (using XOR as boolean operator),
424         // then the square is valid option, hence I turn that black square to red.
425         if (!(isForced ^ enemyFound) && currentColor == DamkaTile.TilePawn.BLACK.ordinal())
426         {
427             // Check if a double is possible
428             if (isForced && !doubleFound)
429             {
430                 enemy.setColor(DamkaTile.TilePawn.BLACK.ordinal());
431                 if (canQueenCapture(currentRowIndex, currentColIndex, colorEaten1, colorEaten2))
432                     doubleFound = true;
433                 enemy.setColor(enemyColor);
434             }
435         }
436     }
437 }
```

# קוד הפרויקט

```
433     }
434
435     tiles[currentRowIndex][currentColIndex].setColor(DamkaTile.TilePawn.RED.ordinal());
436 }
437
438 currentRowIndex += QUEEN_DIRS[i][0];
439 currentColIndex += QUEEN_DIRS[i][1];
440 if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
441     currentColIndex == -1 || currentColIndex == LENGTH)
442     break;
443 currentColor = tiles[currentRowIndex][currentColIndex].color;
444
445 // Check for enemy
446 if ((currentColor == colorEaten1 || currentColor == colorEaten2) && !enemyFound && isForced)
447 {
448     enemyFound = true;
449     enemy = tiles[currentRowIndex][currentColIndex];
450     enemyColor = enemy.color;
451     currentRowIndex += QUEEN_DIRS[i][0];
452     currentColIndex += QUEEN_DIRS[i][1];
453     if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
454         currentColIndex == -1 || currentColIndex == LENGTH)
455         break;
456     currentColor = tiles[currentRowIndex][currentColIndex].color;
457 }
458 }
459
460 // If double found, remove red squares that do not lead to double
461 if (doubleFound)
462 {
463     enemy.setColor(DamkaTile.TilePawn.BLACK.ordinal()); // will be returned
464     if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
465         currentColIndex == -1 || currentColIndex == LENGTH ||
466         tiles[currentRowIndex][currentColIndex].color != DamkaTile.TilePawn.RED.ordinal())
467     {
468         currentRowIndex -= QUEEN_DIRS[i][0];
469         currentColIndex -= QUEEN_DIRS[i][1];
470     }
471     currentColor = tiles[currentRowIndex][currentColIndex].color;
472
473     while (currentColor == DamkaTile.TilePawn.RED.ordinal())
474     {
475         if (!canQueenCapture(currentRowIndex, currentColIndex, colorEaten1, colorEaten2))
476             tiles[currentRowIndex][currentColIndex].setColor(DamkaTile.TilePawn.BLACK.ordinal());
477
478         currentRowIndex -= QUEEN_DIRS[i][0];
479         currentColIndex -= QUEEN_DIRS[i][1];
480         currentColor = tiles[currentRowIndex][currentColIndex].color;
481     }
482
483     enemy.setColor(enemyColor);
484 }
485 }
486 }
```

# קוד הפרויקט

```
487
488 /**
489  * Turn all red squares to black.
490  */
491 public void turnRedSquaresOff()
492 {
493     for (int i = 0; i < LENGTH; i++) {
494         for (int j = 1 - i % 2; j < LENGTH; j += 2) {
495             if (tiles[i][j].color == DamkaTile.TilePawn.RED.ordinal()) {
496                 tiles[i][j].setColor(DamkaTile.TilePawn.BLACK.ordinal());
497             }
498         }
499     }
500 }
501
502 /**
503  * Move the chosen pawn to a new tile.
504  * Promote the pawn to a queen if needed.
505  * @param tile: tile to move to
506  */
507 public void movePawn(DamkaTile tile)
508 {
509     movesWithoutProgress = 0;
510     if (turn)
511     {
512         if (tile.row != LENGTH - 1)
513             tile.setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
514
515         else
516         {
517             tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
518             if (isComputerPlaying && !Computer.comp.movesStack.isEmpty()) //changedd
519                 Computer.comp.movesStack.peek().wasPromotion = true;
520             blackQueens++;
521         }
522     }
523     else
524     {
525         if (tile.row != 0)
526             tile.setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
527         else
528         {
529             tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
530             if (isComputerPlaying && !Computer.comp.movesStack.isEmpty())
531                 Computer.comp.movesStack.peek().wasPromotion = true;
532             whiteQueens++;
533         }
534     }
535 }
536
537 tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.BLACK.ordinal());
538 isPawnChosen = false;
539 turnRedSquaresOff();
540 changeTurn();
```

# קוד הפרויקט

```
541 }
542
543 /**
544  * Move the queen from one tile to another.
545  * Announce a draw if needed.
546  * @param tile: tile to move the queen to.
547  */
548 public void moveQueen(DamkaTile tile)
549 {
550     movesWithoutProgress++;
551     if (movesWithoutProgress == MOVES_FOR_DRAW && !isComputerPlaying)
552         endGame("Draw!!!");
553     if (turn)
554         tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
555     else
556         tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
557
558     tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.BLACK.ordinal());
559     isPawnChosen = false;
560     turnRedSquaresOff();
561     changeTurn();
562 }
563
564 /**
565  * 1) Change the turn
566  * 2) Check if the position is forced
567  * 3) Make the computer play if it should
568  * 4) Check if the game is over.
569  * 5) reevaluate the position if needed
570  */
571 public void changeTurn()
572 {
573     //<editor-fold defaultstate="collapsed" desc="flip board">
574     /*gamePanel.removeAll();
575     gamePanel.repaint();
576     gamePanel.
577
578     if (turn)
579     {
580         for (int i = 0; i < LENGTH; i++)
581         {
582             for (int j = 0; j < LENGTH; j++)
583             {
584                 gamePanel.add(tiles[i][j]);
585             }
586         }
587     }
588     else
589     {
590         for (int i = LENGTH - 1; i >= 0; i--)
591         {
592             for (int j = LENGTH - 1; j >= 0; j--)
593             {
594                 gamePanel.add(tiles[i][j]);
```

# קוד הפרויקט

```
595     }
596     }
597     }
598
599     gamePanel.repaint();
600     gamePanel.*/
601 //</editor-fold>
602
603     showEval.setEnabled(true);
604     turn = !turn;
605     isForced = DoesCaptureExist();
606
607     if (isComputer && !isComputerPlaying && turn)
608     {
609         //gamePanel.update(gamePanel.getGraphics());
610         Computer.comp.play();
611         isForced = DoesCaptureExist();
612     }
613     else
614     {
615         if (isComputerPlaying)
616             return;
617         if (turn && !isForced && blackPawnsLeft <= MAX_PAWNS_IN_STALEMATE &&
618             !canPlay(DamkaTile.TilePawn.BLACK_PAWN.ordinal(),
619                     DamkaTile.TilePawn.BLACK_QUEEN.ordinal()))
620             endGame("White Wins!!!");
621         else if (!turn && !isForced && whitePawnsLeft <= MAX_PAWNS_IN_STALEMATE &&
622             !canPlay(DamkaTile.TilePawn.WHITE_PAWN.ordinal(),
623                     DamkaTile.TilePawn.WHITE_QUEEN.ordinal()))
624             endGame("Black Wins!!!");
625         else if (showEval.isSelected())
626             evaluate();
627     }
628
629 }
630
631 private boolean DoesCaptureExist()
632 {
633     int colorEating1;
634     int colorEating2;
635     int colorEaten1;
636     int colorEaten2;
637
638     if (turn)
639     {
640         colorEating1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
641         colorEating2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
642         colorEaten1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
643         colorEaten2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
644     }
645     else
646     {
647         colorEating1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
648         colorEating2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
```

# קוד הפרויקט

```
649     colorEaten1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
650     colorEaten2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
651 }
652
653 for (int i = 0; i < LENGTH; i++)
654 {
655     for (int j = 1 - i%2; j < LENGTH; j+= 2)
656     {
657         if (tiles[i][j].color == colorEating1)
658         {
659             if (canPawnCapture(i, j, colorEaten1, colorEaten2))
660                 return true;
661         }
662         else if (tiles[i][j].color == colorEating2)
663             if (canQueenCapture(i, j, colorEaten1, colorEaten2))
664                 return true;
665     }
666 }
667
668 return false;
669 }
670
671 private boolean canPawnCapture(int row, int col, int colorEaten1, int colorEaten2)
672 {
673     if (col >= 2 && row >= 2)
674         if (tiles[row - 2][col - 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
675             (tiles[row - 1][col - 1].color == colorEaten1 ||
676              tiles[row - 1][col - 1].color == colorEaten2))
677             return true;
678     if (col >= 2 && row <= LENGTH - 3)
679         if (tiles[row + 2][col - 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
680             (tiles[row + 1][col - 1].color == colorEaten1 ||
681              tiles[row + 1][col - 1].color == colorEaten2))
682             return true;
683     if (col <= LENGTH - 3 && row >= 2)
684         if (tiles[row - 2][col + 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
685             (tiles[row - 1][col + 1].color == colorEaten1 ||
686              tiles[row - 1][col + 1].color == colorEaten2))
687             return true;
688     if (col <= LENGTH - 3 && row <= LENGTH - 3)
689         if (tiles[row + 2][col + 2].color == DamkaTile.TilePawn.BLACK.ordinal() &&
690             (tiles[row + 1][col + 1].color == colorEaten1 ||
691              tiles[row + 1][col + 1].color == colorEaten2))
692             return true;
693     return false;
694 }
695
696 private boolean canQueenCapture(int row, int col, int colorEaten1, int colorEaten2)
697 {
698     int currentColor;
699     int currentRowIndex;
700     int currentColIndex;
701     boolean edgeReached;
702     for (int i = 0; i < 4; i++)
```

# קוד הפרויקט

```
703 {
704     edgeReached = false;
705     currentRowIndex = row + QUEEN_DIRS[i][0];
706     currentColIndex = col + QUEEN_DIRS[i][1];
707     if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
708         currentColIndex == -1 || currentColIndex == LENGTH)
709         continue;
710     currentColor = tiles[currentRowIndex][currentColIndex].color;
711
712     while (currentColor == DamkaTile.TilePawn.BLACK.ordinal() ||
713           currentColor == DamkaTile.TilePawn.RED.ordinal())
714     {
715         currentRowIndex += QUEEN_DIRS[i][0];
716         currentColIndex += QUEEN_DIRS[i][1];
717         if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
718             currentColIndex == -1 || currentColIndex == LENGTH)
719         {
720             edgeReached = true;
721             break;
722         }
723         currentColor = tiles[currentRowIndex][currentColIndex].color;
724     }
725
726     if (!edgeReached)
727     {
728         if (currentColor != colorEaten1 && currentColor != colorEaten2)
729             continue;
730
731         currentRowIndex += QUEEN_DIRS[i][0];
732         currentColIndex += QUEEN_DIRS[i][1];
733         if (currentRowIndex == -1 || currentRowIndex == LENGTH ||
734             currentColIndex == -1 || currentColIndex == LENGTH)
735             continue;
736         currentColor = tiles[currentRowIndex][currentColIndex].color;
737         if (currentColor == DamkaTile.TilePawn.BLACK.ordinal())
738             return true;
739     }
740
741     return false;
742 }
743
744 /**
745  * Perform a capture.
746  * Call the correct capture function.
747  * @param tile: tile to move to.
748  */
749 public void Capture(DamkaTile tile)
750 {
751     showEval.setEnabled(false);
752     movesWithoutProgress = 0;
753     if (turn)
754         whitePawnsLeft--;
755     else
756         blackPawnsLeft--;
```



# קוד הפרויקט

```
757
758     int eatingTileColor = tiles[chosenPawnRow][chosenPawnCol].color;
759     tiles[chosenPawnRow][chosenPawnCol].setColor(DamkaTile.TilePawn.BLACK.ordinal());
760     turnRedSquaresOff();
761     if (eatingTileColor == DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal() ||
762         eatingTileColor == DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal())
763         CaptureWithPawn(tile);
764     else
765         CaptureWithQueen(tile);
766
767 }
768
769 private void CaptureWithPawn(DamkaTile tile)
770 {
771     DamkaTile current = tiles[chosenPawnRow][chosenPawnCol];
772     DamkaTile deadTile;
773     int colorToEat1, colorToEat2;
774     int deadTileRow = (tile.row + current.row)/2;
775     int deadTileCol = (tile.col + current.col)/2;
776     if (isComputerPlaying && !Computer.comp.movesStack.isEmpty())
777     {
778         Computer.comp.movesStack.peek().capturedCol = deadTileCol;
779         Computer.comp.movesStack.peek().capturedRow = deadTileRow;
780         Computer.comp.movesStack.peek().capturedColor = tiles[deadTileRow][deadTileCol].color;
781     }
782     boolean wasPromoted = false;
783
784     deadTile = tiles[deadTileRow][deadTileCol];
785     if (turn)
786     {
787         if (deadTile.color == DamkaTile.TilePawn.WHITE_QUEEN.ordinal())
788             whiteQueens--;
789         colorToEat1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
790         colorToEat2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
791         deadTile.setColor(DamkaTile.TilePawn.DEAD_WHITE.ordinal());
792         if (tile.row == LENGTH - 1)
793         {
794             tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
795             wasPromoted = true;
796             blackQueens++;
797         }
798         else
799             tile.setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
800     }
801     else
802     {
803         if (deadTile.color == DamkaTile.TilePawn.BLACK_QUEEN.ordinal())
804             blackQueens--;
805         colorToEat1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
806         colorToEat2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
807         deadTile.setColor(DamkaTile.TilePawn.DEAD_BLACK.ordinal());
808         if (tile.row == 0)
809         {
810             tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
```

# קוד הפרויקט

```
811         wasPromoted = true;
812         whiteQueens++;
813     }
814     else
815         tile.setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
816 }
817
818 if (isComputerPlaying && !Computer.comp.movesStack.isEmpty())
819     Computer.comp.movesStack.peek().wasPromotion = wasPromoted;
820
821 // Check if Position is on streak
822 if (!wasPromoted && canPawnCapture(tile.row, tile.col, colorToEat1, colorToEat2))
823 {
824     isOnStreak = true;
825     if (turn)
826         tile.setColor(DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal());
827     else
828         tile.setColor(DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal());
829
830     isPawnChosen = true;
831     chosenPawnRow = tile.row;
832     chosenPawnCol = tile.col;
833     turnRedPawnSquaresOn();
834
835     if (isComputer && !isComputerPlaying && turn)
836     {
837
838         try {
839             gamePanel.update(gamePanel.getGraphics());
840             Thread.sleep(321);
841         } catch (InterruptedException ex) {
842             Logger.getLogger(Damka.class.getName()).log(Level.SEVERE, null, ex);
843         }
844         Computer.comp.play();
845     }
846
847
848 }
849 else if (wasPromoted && canQueenCapture(tile.row, tile.col, colorToEat1, colorToEat2))
850 {
851     isOnStreak = true;
852     if (turn)
853         tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal());
854     else
855         tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal());
856
857     isPawnChosen = true;
858     chosenPawnRow = tile.row;
859     chosenPawnCol = tile.col;
860     turnRedQueenSquaresOn();
861     if (isComputer && !isComputerPlaying && turn)
862     {
863
864         try {
```

# קוד הפרויקט

```
865         gamePanel.update(gamePanel.getGraphics());
866         Thread.sleep(321);
867     } catch (InterruptedException ex) {
868         Logger.getLogger(Damka.class.getName()).log(Level.SEVERE, null, ex);
869     }
870     Computer.comp.play();
871 }
872
873 }
874 else
875 {
876     removeDeadTiles();
877     isOnStreak = false;
878     isForced = false;
879     isPawnChosen = false;
880     changeTurn();
881 }
882
883 }
884
885 private void CaptureWithQueen(DamkaTile tile)
886 {
887     DamkaTile current = tiles[chosenPawnRow][chosenPawnCol];
888     DamkaTile deadTile;
889     int colorToEat1, colorToEat2;
890     int rowDir = (int)Math.signum(tile.row - current.row);
891     int colDir = (int)Math.signum(tile.col - current.col);
892
893     int deadTileRow = current.row + rowDir;
894     int deadTileCol = current.col + colDir;
895
896     if (turn)
897     {
898         tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
899         colorToEat1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
900         colorToEat2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
901     }
902     else
903     {
904         tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
905         colorToEat1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
906         colorToEat2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
907     }
908
909     deadTile = tiles[deadTileRow][deadTileCol];
910     while (deadTile.color != colorToEat1 && deadTile.color != colorToEat2)
911     {
912         deadTileRow += rowDir;
913         deadTileCol += colDir;
914         deadTile = tiles[deadTileRow][deadTileCol];
915     }
916
917     if (isComputerPlaying && !Computer.comp.movesStack.isEmpty())
918     {
919         Computer.comp.movesStack.peek().capturedCol = deadTileCol;
```

# קוד הפרויקט

```
919 Computer.comp.movesStack.peek().capturedRow = deadTileRow;
920 Computer.comp.movesStack.peek().capturedColor = tiles[deadTileRow][deadTileCol].color;
921 }
922
923 if (turn)
924 {
925     if (deadTile.color == DamkaTile.TilePawn.WHITE_QUEEN.ordinal())
926         whiteQueens--;
927     deadTile.setColor(DamkaTile.TilePawn.DEAD_WHITE.ordinal());
928 }
929 else
930 {
931     if (deadTile.color == DamkaTile.TilePawn.BLACK_QUEEN.ordinal())
932         blackQueens--;
933
934     deadTile.setColor(DamkaTile.TilePawn.DEAD_BLACK.ordinal());
935 }
936
937 // Check for strak
938 if (canQueenCapture(tile.row, tile.col, colorToEat1, colorToEat2))
939 {
940     turnRedSquaresOff();
941     isPawnChosen = true;
942     isOnStreak = true;
943     chosenPawnRow = tile.row;
944     chosenPawnCol = tile.col;
945     if (turn)
946         tile.setColor(DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal());
947     else
948         tile.setColor(DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal());
949     turnRedQueenSquaresOn();
950
951     if (isComputer && !isComputerPlaying && turn)
952     {
953         //gamePanel.update(gamePanel.getGraphics()); // has it's downside...
954         Computer.comp.play();
955     }
956
957 }
958 else
959 {
960     isPawnChosen = false;
961     removeDeadTiles();
962     isForced = false;
963     isPawnChosen = false;
964     isOnStreak = false;
965     changeTurn();
966 }
967 }
968
969 private void removeDeadTiles()
970 {
971     for (int i = 0; i < LENGTH; i++) {
972         for (int j = 1 - i % 2; j < LENGTH; j += 2) {
```

# קוד הפרויקט

```
973         if (tiles[i][j].color == DamkaTile.TilePawn.DEAD_WHITE.ordinal() ||
974             tiles[i][j].color == DamkaTile.TilePawn.DEAD_BLACK.ordinal()) {
975             tiles[i][j].setColor(DamkaTile.TilePawn.BLACK.ordinal());
976         }
977     }
978 }
979 }
980
981 /**
982  * End the game with custom form.
983  * @param message: message to display
984  */
985 public void endGame(String message)
986 {
987     setEnabled(false);
988
989     JFrame gameOverF = new JFrame();
990     JLabel gameOverL = new JLabel(message);
991     JButton gameOverB = new JButton("OK!");
992
993     gameOverL.setBounds(60, 95, 100, 20);
994     gameOverB.setBounds(35, 20, 120, 40);
995
996     gameOverB.addActionListener(new ActionListener()
997     {
998         @Override
999         public void actionPerformed(ActionEvent ae) {
1000             gameOverF.dispose();
1001             dispose();
1002         }
1003     });
1004
1005     gameOverF.add(gameOverL);
1006     gameOverF.add(gameOverB);
1007
1008     gameOverF.setSize(200, 150);
1009     gameOverF.setLayout(null);
1010     gameOverF.setDefaultCloseOperation(EXIT_ON_CLOSE);
1011     gameOverF.setResizable(false);
1012     gameOverF.setLocationRelativeTo(null); // Passing null centers the form!
1013     gameOverF.setVisible(true);
1014
1015 }
1016
1017 /**
1018  * Checks if a move is possible in the current position
1019  * @param movingColor1: Color of moving pawn
1020  * @param movingColor2: Color of moving queen
1021  * @return
1022  */
1023 public boolean canPlay(int movingColor1, int movingColor2)
1024 {
1025     int direction = (turn) ? 1 : -1;
1026 }
```

# קוד הפרויקט

```
1027     for (int i = 0; i < LENGTH; i++) {
1028         for (int j = 1 - i % 2; j < LENGTH; j += 2) {
1029             if (tiles[i][j].color == movingColor1) // Check if pawn can move
1030             {
1031                 if (j > 0 && tiles[i + direction][j-1].color == DamkaTile.TilePawn.BLACK.ordinal())
1032                     return true;
1033                 if (j < LENGTH - 1 && tiles[i + direction][j+1].color == DamkaTile.TilePawn.BLACK.ordinal())
1034                     return true;
1035             }
1036             else if (tiles[i][j].color == movingColor2) // Check if queen can move
1037             {
1038                 for (int queen_i = 0; queen_i < 4; queen_i++)
1039                 {
1040                     int rowIndex = i + QUEEN_DIRS[queen_i][0];
1041                     int colIndex = j + QUEEN_DIRS[queen_i][1];
1042                     if (rowIndex >= 0 && rowIndex <= LENGTH - 1 &&
1043                         colIndex >= 0 && colIndex <= LENGTH - 1 &&
1044                         tiles[rowIndex][colIndex].color == DamkaTile.TilePawn.BLACK.ordinal())
1045                         return true;
1046                 }
1047             }
1048         }
1049     }
1050
1051     return false;
1052 }
1053
1054 // Evaluate the position using the In-Place Minimax DFS with Alpha Beta pruning algorithm
1055 // in the Computer class.
1056 // Update the evaluation panel according to the evaluation
1057 private void evaluate()
1058 {
1059     if (!isOnStreak && isPawnChosen)
1060         turnPawnOff();
1061     isComputerPlaying = true;
1062     int temp = movesWithoutProgress, msgIndex;
1063     float posValForWhite =
1064         -Computer.comp.miniMaxAlphaBeta(0, turn, Computer.MIN_POS_VAL, Computer.MAX_POS_VAL);
1065     evaluation.setText("" + posValForWhite);
1066     if (posValForWhite > 1.5)
1067     {
1068         msgIndex = 3;
1069         winningColor = Color.YELLOW;
1070     }
1071     else if (posValForWhite > 0.6)
1072     {
1073         msgIndex = 1;
1074         winningColor = Color.ORANGE;
1075     }
1076     else if (posValForWhite > -0.6)
1077     {
1078         msgIndex = 0;
1079         winningColor = Color.WHITE;
1080     }
}
```

# קוד הפרויקט

```
1081     else if (posValForWhite > -1.5)
1082     {
1083         winningColor = Color.CYAN;
1084         msgIndex = 2;
1085     }
1086     else
1087     {
1088         winningColor = PURPLE;
1089         msgIndex = 4;
1090     }
1091     stateMsg.setText(STATES[msgIndex]);
1092     evalPanel.setBackground(winningColor);
1093     evaluation.setBackground(winningColor);
1094     evalMsg.setBackground(winningColor);
1095
1096     isComputerPlaying = false;
1097     movesWithoutProgress = temp;
1098 }
1099 }
```

# קוד הפרויקט

## Computer

```
1 package myproject;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5 import java.util.Stack;
6 import static myproject.Damka.MAX_PAWNS_IN_STALEMATE;
7 import static myproject.DamkaTile.TilePawn.BLACK;
8
9 /**
10  * AI for playing Shashki
11  * @author Daniel Kanevsky
12  */
13 public class Computer {
14
15     private Computer() {}
16
17     class Move
18     {
19         public int fromRow; // Row index of origin tile
20         public int fromCol; // Col index of origin tile
21         public int toRow; // Row index of destination tile
22         public int toCol; // Col index of destination tile
23         public int movesWithoutProgress; // Moves without progress before this move was played
24         public boolean isCapture; // Is this move a capture?
25         public int capturedColor; // Color of captured piece
26         public int capturedRow; // Row of captured piece
27         public int capturedCol; // Col of captured piece
28
29         public boolean wasOnStreak; // Was this move made as the part of streak
30         public boolean wasPromotion; // Was this a promotion to a queen
31         public boolean turn; // Whose turn is it? false - white, true - black
32         ArrayList<Integer> deadTiles; // Dead tiles on board before making this move
33
34         public Move(int fromRow, int fromCol, int toRow, int toCol, int movesWithoutProgress,
35                     boolean isCapture, boolean wasOnStreak, boolean turn)
36         {
37             this.fromRow = fromRow;
38             this.fromCol = fromCol;
39             this.toRow = toRow;
40             this.toCol = toCol;
41             this.movesWithoutProgress = movesWithoutProgress;
42             this.isCapture = isCapture;
43             this.wasOnStreak = wasOnStreak;
44             this.turn = turn;
45             wasPromotion = false;
46         }
47
48         private static final int DEPTH_MAX = 8;
49         public static final float MIN_POS_VAL = -10000;
50         public static final float MAX_POS_VAL = 10000;
51
52         // Singleton instance
53         public static Computer comp = new Computer();
54     }
```



# קוד הפרויקט

```
55 public Damka board;
56 public Move moveToPlay; // The move which will be played
57 private static Random rand = new Random();
58
59 Stack<Move> movesStack = new Stack<>();
60
61 // Generate all possible moves in the current position
62 private ArrayList<Move> generateMoves ()
63 {
64     ArrayList<Move> moves = new ArrayList<>();
65     int playing1, playing2;
66     boolean isCapture = board.isForced;
67
68     // If the board is on strak, generate the moves whit the pawn/queen on streak.
69     if (board.isOnStreak)
70     {
71         for (int i = 0; i < board.LENGTH; i++)
72         {
73             for (int j = 1 - i%2; j < board.LENGTH; j+= 2)
74             {
75                 if (board.tiles[i][j].color == DamkaTile.TilePawn.RED.ordinal())
76                 {
77                     moves.add(new Move(board.chosenPawnRow, board.chosenPawnCol,i,j,0,true, true, board.turn));
78                 }
79             }
80         }
81
82         return moves;
83     }
84
85     if (board.turn)
86     {
87         playing1 = DamkaTile.TilePawn.BLACK_PAWN.ordinal();
88         playing2 = DamkaTile.TilePawn.BLACK_QUEEN.ordinal();
89     }
90     else
91     {
92         playing1 = DamkaTile.TilePawn.WHITE_PAWN.ordinal();
93         playing2 = DamkaTile.TilePawn.WHITE_QUEEN.ordinal();
94     }
95
96
97
98     for (int i = 0; i < board.LENGTH; i++) {
99         for (int j = 1 - i % 2; j < board.LENGTH; j += 2) {
100             if (board.tiles[i][j].color == playing1 ||
101                 board.tiles[i][j].color == playing2)
102             {
103                 board.chosenPawnRow = i;
104                 board.chosenPawnCol = j;
105
106                 // Turn the red squares for the current tile
107                 if (board.tiles[i][j].color == playing1)
108                 {
```

# קוד הפרויקט

```
109     if (board.turn)
110         board.tiles[i][j].setColor(DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal());
111     else
112         board.tiles[i][j].setColor(DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal());
113     board.turnRedPawnSquaresOn();
114     if (board.turn)
115         board.tiles[i][j].setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
116     else
117         board.tiles[i][j].setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
118 }
119 else
120 {
121     if (board.turn)
122         board.tiles[i][j].setColor(DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal());
123     else
124         board.tiles[i][j].setColor(DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal());
125     board.turnRedQueenSquaresOn();
126     if (board.turn)
127         board.tiles[i][j].setColor(DamkaTile.TilePawn.BLACK_QUEEN.ordinal());
128     else
129         board.tiles[i][j].setColor(DamkaTile.TilePawn.WHITE_QUEEN.ordinal());
130 }
131
132 // Add the moves for the current tile
133 for (int k = 0; k < board.LENGTH; k++)
134 {
135     for (int l = 1 - k%2; l < board.LENGTH; l += 2)
136     {
137         if (board.tiles[k][l].color == DamkaTile.TilePawn.RED.ordinal())
138         {
139             board.tiles[k][l].setColor(BLACK.ordinal());
140             moves.add(new Move(i, j, k, l, board.movesWithoutProgress, isCapture, false, board.turn));
141         }
142     }
143 }
144 }
145 }
146 }
147
148 return moves;
149 }
150
151 private void findBestMove()
152 {
153     ArrayList<Move> possibilities = generateMoves();
154     if (possibilities.isEmpty())
155         return;
156
157     // the Move to play is the first possible move as a default option
158     moveToPlay = possibilities.get(0);
159
160     // If there is only one move to make -> do it!
161     if (possibilities.size() == 1)
162         return;
```

# קוד הפרויקט

```
163 float moveValue, bestValue = MIN_POS_VAL;
164 boolean isOver = false; // No need to look for more moves if a win found
165
166
167 for (Move possibility : possibilities)
168 {
169     movesStack.push(possibility);
170     makeMove(possibility);
171     // evaluate the position with the move made, the best value so far is the alpha value
172     moveValue = miniMaxAlphaBeta(1, board.isOnStreak, bestValue, MAX_POS_VAL);
173
174     if (moveValue > bestValue)
175     {
176         bestValue = moveValue;
177         moveToPlay = possibility;
178         if (bestValue == MAX_POS_VAL)
179             isOver = true;
180     }
181     undoMove(movesStack.pop()); // return the board to it's previous state
182     if (isOver)
183         break;
184 }
185
186 }
187
188 /**
189  * Search for the best move using MiniMax DFS search with
190  * Alpha-Beta pruning optimization.
191  * @param currentDepth : the Depth reached by the recursive calls from the root node
192  * @param Max : true -> Max is playing ~~~ false -> Min is playing
193  * @param alpha : alpha value -> best position value guaranteed for Max in the current node
194  * @param beta : beta value -> best position value guaranteed for Min in the current node
195  * @return : position Value
196  */
197 public float miniMaxAlphaBeta(int currentDepth, boolean Max, float alpha, float beta)
198 {
199     if (board.movesWithoutProgress == board.MOVES_FOR_DRAW)
200         return 0;
201     if (currentDepth == DEPTH_MAX)
202         return evaluatePosition();
203
204     float positionValue; // value of position = best moveValue so far
205     float moveValue; // used as a 'temp' variable' to determine value of each move
206     ArrayList<Move> possibilities = generateMoves();
207     int sign;
208
209     if (Max)
210     {
211         positionValue = MIN_POS_VAL;
212         sign = 1;
213     }
214     else
215     {
216         positionValue = MAX_POS_VAL;
```

# קוד הפרויקט

```
217         sign = -1;
218     }
219
220     // Choose the best move from the possible moves recursively
221     for (Move possibility : possibilities)
222     {
223         movesStack.push(possibility);
224         makeMove(possibility);
225         /*board.gamePanel.update(board.gamePanel.getGraphics());
226         try {
227             Thread.sleep(50);
228         } catch (InterruptedException ex) {
229             Logger.getLogger(Computer.class.getName()).log(Level.SEVERE, null, ex);
230         }*/
231         if (board.isOnStreak)
232             moveValue = miniMaxAlphaBeta(currentDepth, Max, alpha, beta);
233         else
234             moveValue = miniMaxAlphaBeta(currentDepth+1, !Max, alpha, beta);
235         if (Math.signum(moveValue - positionValue) == sign)
236             positionValue = moveValue;
237
238         if (Max && positionValue > alpha)
239             alpha = positionValue;
240         else if (!Max && positionValue < beta)
241             beta = positionValue;
242
243         undoMove(movesStack.pop());
244         /*board.gamePanel.update(board.gamePanel.getGraphics());
245         try {
246             Thread.sleep(50);
247         } catch (InterruptedException ex) {
248             Logger.getLogger(Computer.class.getName()).log(Level.SEVERE, null, ex);
249         }*/
250         // Alpha-Beta Purning!!!
251         if (alpha >= beta)
252             break;
253     }
254
255     return positionValue;
256 }
257
258 // Search for the best move via minimax without Alpha-Beta pruning
259 private float miniMaxBestMove(int currentDepth, int sign)
260 {
261     if (board.movesWithoutProgress == Damka.MOVES_FOR_DRAW)
262         return 0;
263     if (currentDepth == DEPTH_MAX)
264         return evaluatePosition();
265
266     ArrayList<Move> possibilities = generateMoves();
267     float positionValue = MIN_POS_VAL;
268     float moveValue;
269     boolean bestMoveFound = false;
270 }
```

# קוד הפרויקט

```
271 for (Move possibility : possibilities)
272 {
273     movesStack.push(possibility);
274     makeMove(possibility);
275     if (board.isOnStreak)
276         moveValue = miniMaxBestMove(currentDepth + 1, sign)*sign;
277     else
278         moveValue = miniMaxBestMove(currentDepth + 1, -sign)*sign;
279
280     if (moveValue > positionValue)
281     {
282         positionValue = moveValue;
283         if (currentDepth == 0)
284             moveToPlay = possibility;
285         if (positionValue == MAX_POS_VAL)
286             bestMoveFound = true;
287     }
288
289     undoMove(movesStack.pop());
290     if (bestMoveFound)
291         break;
292 }
293
294 return positionValue*sign;
295 }
296
297
298 private void makeMove(Move move)
299 {
300     board.chosenPawnRow = move.fromRow;
301     board.chosenPawnCol = move.fromCol;
302     DamkaTile originTile = board.tiles[move.fromRow][move.fromCol];
303     DamkaTile destinationTile = board.tiles[move.toRow][move.toCol];
304
305     // set the pawn/queen as chosen if isn't already
306     if (originTile.color == DamkaTile.TilePawn.WHITE_PAWN.ordinal())
307         originTile.setColor(DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal());
308     else if (originTile.color == DamkaTile.TilePawn.BLACK_PAWN.ordinal())
309         originTile.setColor(DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal());
310     else if (originTile.color == DamkaTile.TilePawn.WHITE_QUEEN.ordinal())
311         originTile.setColor(DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal());
312     else if (originTile.color == DamkaTile.TilePawn.BLACK_QUEEN.ordinal())
313         originTile.setColor(DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal());
314
315     if (!move.isCapture)
316     {
317         if (originTile.color == DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal() ||
318             originTile.color == DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal())
319             board.movePawn(destinationTile);
320         else
321             board.moveQueen(destinationTile);
322     }
323     else // Forced to play = capture
324         board.Capture(destinationTile);
```

# קוד הפרויקט

```
325 }
326
327 }
328
329 private void undoMove(Move move)
330 {
331     board.movesWithoutProgress = move.movesWithoutProgress;
332     int deadTileColor;
333     board.turnRedSquaresOff();
334     if (!board.isOnStreak && move.wasOnStreak)
335     {
336
337         if (move.turn)
338             deadTileColor = DamkaTile.TilePawn.DEAD_WHITE.ordinal();
339         else
340             deadTileColor = DamkaTile.TilePawn.DEAD_BLACK.ordinal();
341
342         Stack<Move> temp = new Stack<>();
343         while (!movesStack.isEmpty() && movesStack.peek().isCapture && movesStack.peek().turn)
344         {
345             Move prevMove = movesStack.pop();
346             temp.push(prevMove);
347             board.tiles[prevMove.capturedRow][prevMove.capturedCol].setColor(deadTileColor);
348         }
349
350         while (!temp.isEmpty())
351             movesStack.push(temp.pop());
352     }
353     DamkaTile originTile = board.tiles[move.fromRow][move.fromCol];
354     DamkaTile destinationTile = board.tiles[move.toRow][move.toCol];
355     int destColor = destinationTile.color;
356
357     if (!board.isOnStreak && move.wasOnStreak)
358         destinationTile.setColor(DamkaTile.TilePawn.RED.ordinal());
359     else
360         destinationTile.setColor(DamkaTile.TilePawn.BLACK.ordinal());
361
362
363     board.isOnStreak = move.wasOnStreak;
364     board.isForced = move.isCapture;
365     board.turn = move.turn;
366
367     if (!move.wasOnStreak &&
368         (destColor == DamkaTile.TilePawn.WHITE_PAWN_CHOSEN.ordinal() ||
369          destColor == DamkaTile.TilePawn.BLACK_PAWN_CHOSEN.ordinal() ||
370          destColor == DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal() ||
371          destColor == DamkaTile.TilePawn.BLACK_QUEEN_CHOSEN.ordinal()))
372         destColor -= 2;
373     if (move.wasOnStreak &&
374         (destColor == DamkaTile.TilePawn.WHITE_PAWN.ordinal() ||
375          destColor == DamkaTile.TilePawn.BLACK_PAWN.ordinal() ||
376          destColor == DamkaTile.TilePawn.WHITE_QUEEN.ordinal() ||
377          destColor == DamkaTile.TilePawn.BLACK_QUEEN.ordinal()))
378         destColor += 2;
```

# קוד הפרויקט

```
379 originTile.setColor(destColor);
380
381 if (move.wasPromotion)
382 {
383     if (originTile.color == DamkaTile.TilePawn.WHITE_QUEEN.ordinal() ||
384         originTile.color == DamkaTile.TilePawn.WHITE_QUEEN_CHOSEN.ordinal())
385     {
386         board.whiteQueens--;
387         originTile.setColor(DamkaTile.TilePawn.WHITE_PAWN.ordinal());
388     }
389     else
390     {
391         board.blackQueens--;
392         originTile.setColor(DamkaTile.TilePawn.BLACK_PAWN.ordinal());
393     }
394 }
395
396 if (move.isCapture)
397 {
398     board.tiles[move.capturedRow][move.capturedCol].setColor(move.capturedColor);
399     if (move.turn)
400     {
401         board.whitePawnsLeft++;
402         if (move.capturedColor == DamkaTile.TilePawn.WHITE_QUEEN.ordinal())
403             board.whiteQueens++;
404     }
405
406     else
407     {
408         board.blackPawnsLeft++;
409         if (move.capturedColor == DamkaTile.TilePawn.BLACK_QUEEN.ordinal())
410             board.blackQueens++;
411     }
412 }
413
414
415 if (move.wasOnStreak)
416 {
417     board.isPawnChosen = true;
418     board.chosenPawnRow = move.fromRow;
419     board.chosenPawnCol = move.fromCol;
420
421     if (originTile.color == DamkaTile.TilePawn.BLACK_PAWN.ordinal() ||
422         originTile.color == DamkaTile.TilePawn.WHITE_PAWN.ordinal())
423         board.turnRedPawnSquaresOn();
424     else
425         board.turnRedQueenSquaresOn();
426 }
427 else
428     board.isPawnChosen = false;
429 }
430
431 // Evaluate position in static manner.
432 private float evaluatePosition()
```

# קוד הפרויקט

```
433 {
434     boolean isStartGame = board.whitePawnsLeft + board.blackPawnsLeft > board.PAWN_ROWS*board.LENGTH/3;
435
436     if (board.whitePawnsLeft == 0)
437         return MAX_POS_VAL;
438     if (board.blackPawnsLeft == 0)
439         return MIN_POS_VAL;
440
441
442     float posVal = board.blackPawnsLeft - board.whitePawnsLeft;
443     posVal += 1.1*board.blackQueens;
444     posVal -= 1.1*board.whiteQueens;
445
446     // A queen has a higher value before the endgame
447     if (isStartGame)
448     {
449         posVal += 0.5*board.blackQueens;
450         posVal -= 0.5*board.whiteQueens;
451     }
452
453     if (board.turn)
454     {
455         posVal += 0.25;
456
457         // if the position is forced than a capture exists, usually good
458         if (board.isForced)
459             posVal+= 0.35;
460
461         // check for a loss
462         else
463         {
464             if (board.whitePawnsLeft <= MAX_PAWNS_IN_STALEMATE &&
465                 !board.canPlay(DamkaTile.TilePawn.BLACK_PAWN.ordinal(),
466                     DamkaTile.TilePawn.BLACK_QUEEN.ordinal()))
467                 return MIN_POS_VAL;
468         }
469     }
470     else
471     {
472         posVal -= 0.25;
473         // if the position is forced than a capture exists, usually good (for white)
474         if (board.isForced)
475             posVal -= 0.35;
476         // check for a loss (for white)
477         else
478         {
479             if (board.whitePawnsLeft <= MAX_PAWNS_IN_STALEMATE &&
480                 !board.canPlay(DamkaTile.TilePawn.WHITE_PAWN.ordinal(),
481                     DamkaTile.TilePawn.WHITE_QUEEN.ordinal()))
482                 return MAX_POS_VAL;
483         }
484     }
485
486     // check for black pawns in white territory
487     for (int i = board.LENGTH - 3; i < board.LENGTH - 1; i++) {
```



# קוד הפרויקט

```
487     for (int j = 1 - i % 2; j < board.LENGTH; j += 2) {
488         if (board.tiles[i][j].color == DamkaTile.TilePawn.BLACK_PAWN.ordinal())
489         {
490             // bad in start of the game
491             if (isStartGame)
492                 posVal -= 0.15;
493             // good in the end of the game
494             else
495                 posVal += 0.25;
496         }
497     }
498 }
499
500 // check for white pawns in black territory
501 for (int i = 1; i < 3; i++) {
502     for (int j = 1 - (i % 2); j < board.LENGTH; j += 2) {
503         if (board.tiles[i][j].color == DamkaTile.TilePawn.WHITE_PAWN.ordinal())
504         {
505             // bad in start of the game (for white)
506             if (isStartGame)
507                 posVal += 0.15;
508             // good in the end of the game (for white)
509             else
510                 posVal -= 0.25;
511         }
512     }
513 }
514
515 // check for pawns in back-most and front-most rows at start-game
516 if (isStartGame)
517 {
518     for (int i = 1; i < board.LENGTH; i+= 2)
519         if (board.tiles[0][i].color == DamkaTile.TilePawn.BLACK_PAWN.ordinal())
520             posVal += 0.2;
521
522     for (int i = board.LENGTH % 2; i < board.LENGTH; i+= 2)
523         if (board.tiles[board.LENGTH - 1][i].color == DamkaTile.TilePawn.WHITE_PAWN.ordinal())
524             posVal -= 0.2;
525 }
526
527 return posVal;
528 }
529
530 /**
531  * Find the best move in the position for the computer (black)
532  * And make it
533  */
534 public void play()
535 {
536     moveToPlay = null;
537     board.isComputerPlaying = true;
538     //int movesWithoutProgress = board.movesWithoutProgress;
539     findBestMove();
540     board.isComputerPlaying = false;
541     //board.movesWithoutProgress = movesWithoutProgress;
542     if (moveToPlay != null)
543         makeMove(moveToPlay);
544     else
545         board.endGame("User Wins!!!");
546 }
547 }
548 }
```

## רפלקציה

כשהתחלתי את הפרויקט לא היה לי מושג למה אני נכנס, לא היה לי מושג כמה מאמץ, סקרנות, זמן, ריכוז, יצירתיות, עבודה קשה ומסירות יהיה עליי לתת מעצמי כדי ליצור את פרויקט החלומות שלי. אני מאוד מרוצה מהתוצאה ובאותה מידה אני מרוצה מהדרך. אני שמח שיהיה לפרויקט שלי שימוש מעבר לבחינה. לא הייתי מרגיש שלם כפי שאני מרגיש עכשיו אילולא הייתי נהנה ממה שיצרתי. גם אם לא יהיה לי עם מי לשחק, אני יכול לשחק אם המחשב (:). הפרויקט דרש ממני להרחיב את אופקיי. בתחילת הפרויקט לא היה לי מושג איך לממש דמקה, לא לפי החוקים הרשמיים, ובטח שלא עם אינטליגנציה מלאכותית ממוטבות מהרבה בחינות. לא למדתי הכל בבת אחת אלא בהדרגתיות. היו לי רגעי משבר שבהם רציתי פשוט לוותר ולחשוב על פרויקט אחר, פשוט יותר. אך לא ויתרתי ולא נשברתי, שאלתי שאלות וחיפשתי אחר תשובות, חשבת, גיליתי ויזמתי המון פרטים שביחד הופכים את הפרויקט שלי לשלם. הצלחתי להפתיע את עצמי למה שאני מסוגל, וכעת אני מעריך הרבה יותר את תהליך הלמידה העצמית שאני מסוגל אליו, ואת יכולות התכנות שלי. כעת אני אהיה יותר נועז ונכון לקחת עליי אתגרים למרות ספקות וחששות, הפרויקט הקנה לי ביטחון עצמי רב.

אני מודה מכל הלב למי שקרא את הספר (או חלקו), ואני מאוד מקווה שתרמתי לידע שלך. אני מאפשר להשתמש בקוד לשימוש אישי בלבד. יש לינק לקוד בעמוד 29. אתם יותר ממוזמנים לנסות להביס את ה A.I.

בכל שאלה ניתן לפנות אליי ואשמח לנסות לעזור!



# ביבליוגרפיה

<https://www.wikipedia.org>

<https://stackoverflow.com>

<https://gamedev.stackexchange.com>

<http://tim.hibal.org/blog>

<http://science.sciencemag.org/content/317/5844/1518>

<https://www.chess.com>

<https://www.youtube.com/watch?v=QYNRvMoIN20&t=463s>

[https://www.youtube.com/watch?v=I41\\_sdkuzOY&list=PLY-H7NI6qqgntwySdPesG8eLJRpdjin19](https://www.youtube.com/watch?v=I41_sdkuzOY&list=PLY-H7NI6qqgntwySdPesG8eLJRpdjin19)

<https://www.youtube.com/watch?v=xBXHtz4Gbdo&t=289s>

<https://www.playok.com/en/russiancheckers>

<https://www.quora.com>

<http://vergil.chemistry.gatech.edu/resources/programming/pdf/TIJ2.pdf>

[https://fmjd64.org/wp-content/uploads/2012/02/Rules\\_of\\_Official\\_FMJD-Section-64\\_competitions\\_rus-2014.pdf](https://fmjd64.org/wp-content/uploads/2012/02/Rules_of_Official_FMJD-Section-64_competitions_rus-2014.pdf)