**עבודת גמר במסגרת הנדסת תוכנהChess -**

שם המגיש: אלון יחזקאל  
ת.ז: 213107766  
שם בית הספר: תיכון רבין כפר סבא   
שם המנחה: עופר צשלר  
תאריך הגשה:





Contents

[**פרק 1 - מבוא** 3](#_Toc68708174)

[חוקי המשחק 4](#_Toc68708175)

[מהלכי החיילים 4](#_Toc68708176)

[**פרק 2 - מבנה הפרויקט ובסיסי נתונים** 9](#_Toc68708177)

[מחלקות הפרויקט 10](#_Toc68708178)

[שימוש במבני נתונים 12](#_Toc68708179)

[זרימת המידע בפרויקט בדרכים השונות 13](#_Toc68708180)

[אלגוריתם AI: שימוש ב - Minimax, AlphaBeta וStack 15](#_Toc68708181)

[שיפור Minimax ל Negamax - 24](#_Toc68708182)

[אלגוריתם ניקוד - BFS (Breadth-first search) 25](#_Toc68708183)

[**פרק 3 - מדריך למשתמש** 26](#_Toc68708184)

[הרצת המשחק 26](#_Toc68708185)

[מסך הפתיחה 26](#_Toc68708186)

[מסך המשחק 30](#_Toc68708187)

[**פרק 4 - סיכום אישי** 32](#_Toc68708188)

[**פרק 5 - ביבליוגרפיה** 33](#_Toc68708189)

[**פרק 6 - נספחים** 34](#_Toc68708190)

[זרימת תורות שחקנים (Flow Diagram) 35](#_Toc68708191)

[תרשים היררכית קבצים / תיקיות 36](#_Toc68708192)

[הרצת הפרויקט - 36](#_Toc68708193)

[חבילות להתקנה – 36](#_Toc68708194)

[תרשים UML 37](#_Toc68708195)

[תרשים Use Case 38](#_Toc68708196)

[צילומי קוד חשוב 39](#_Toc68708197)

# פרק 1 - מבוא

ספר פרויקט זה מציג את המימוש של המשחק "**Chess**", כפי שנכתב על ידי.   
בספר הפרויקט ניתן לקרוא אודות אלגוריתמים שונים בפרויקט, בעיות שעלו בתהליך הפיתוח, ממשק המשתמש ועל תהליך כתיבת הפרויקט ממבט אישי.   
המשחק "Chess", המכונה גם שח-מט בעברית, הוא משחק לשני שחקנים, כאשר במימוש זה אחד מן השחקנים משוחק על ידי המחשב כברירת מחדל (ניתן לשנות שיהיה מחשב מול מחשב, חלק או אדם נגד אדם).

המשחק פותח בשפת Python בשימוש בסביבת העבודה Pycharm. הסיבות לכך שהחלטתי להשתמש בשפה זו היא בשל הידע הרב שלי בשפה ובספריות רבות המוצעות בה עבור הפרויקט. ההחלטה עבור סביבת העבודה היית בשל נוחות הסביבה אשר מאפשרת להתמקד בהרבה דברים בו זמנית.  
ההכנות לפרויקט התמקדו בשני רבדים שונים: הרובד התכנותי והרובד המשחקי. במסגרת הרובד התכנותי, שיפרתי את רמת התכנות שלי ע"י אתרים כמו GitHub וערוצים שונים בYoutube, למדתי כללי כתיבת OOP והתנסיתי בספריית tkinter. כדוגמה, את לימודיי OOP שלי רכשתי במסגרת הבית ספר ומדריכים באינטרנט כאשר עבדתי על פרויקטים כגון אחד אשר פיתחתי בסביבת העבודה Unity. באתרGitHub יצא לי להשתמש רבות לצורך העשרה של ידע, הבנה ברמת הקוד של תחומים שונים והעלאת פרויקטים שפיתחתי שכן מטרת האתר היא לשתף קוד.

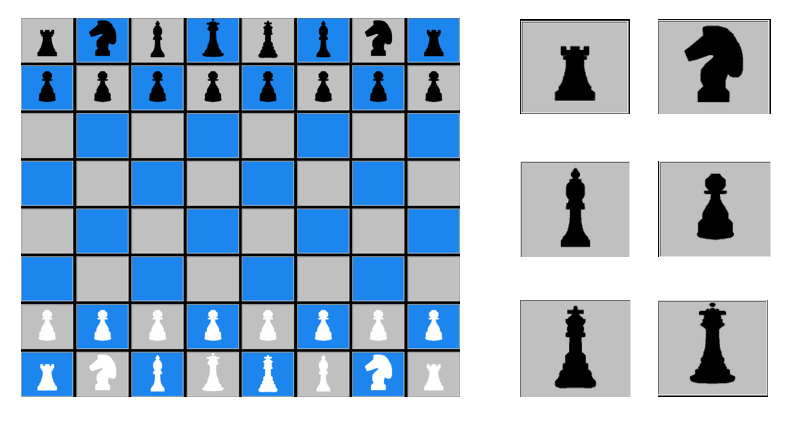
במסגרת הרובד המשחקי, שיחקתי במשחק פעמים רבות כנגד אנשים שונים באינטרנט יחד עם כך ששיחקתי נגד מחשב. מכך ניתן לגבש הבנה טובה יותר של המשחק. כך, התאפשר לי לכתוב מחשב מדויק וחכם יותר ולהבין את הלוגיקה שמובילה לאסטרטגיות ניצחון.

במהלך הפרויקט התמודדתי עם מספר בעיות, כאשר רובן היו בלתי צפויות. המרכזיות עסקו בתיאום בין הקלאסים השונים, ניהול התורות, פונקציות הערכה למחשב, בחינת כמות מהלכים ענקית בצורה יעילה (מהירה) והתמודדות עם מצבי קיצון במשחק (למשל: מצב שאי אפשר להגיע לניצחון כאשר יש 5 חיילים על הלוח במקרים רבים או לולאה אינסופית של מהלכים חוזרים הקוראת במשחקי Minimax רבים).

בעיות אלו נפתרו בעזרת שימוש בDebug, ,כתיבת פונקציות הערכה לפי הסברים של מומחים בתחום והרבה חשיבה מחוץ לקופסה.

## חוקי המשחק

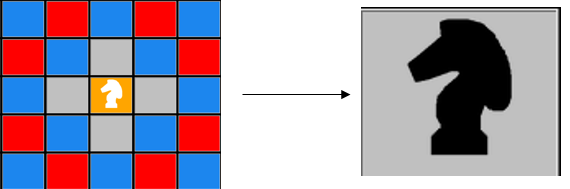
מהלך המשחק ומבנהו הוא כדלהלן:



לכל שחקן יש שישה-עשר חיילים, משישה סוגים שונים: רגלי, פרש, רץ, צריח, מלכה, מלך. כל שחקן בטורו יכול להזיז חייל אחד מכל החיילים שברשותו לפי המהלכים החוקיים עבור כל סוג חייל במצבו הנוכחי. המלך איננו יכול לזוז למקום בו הוא מאוים, כלומר יכול להיאכל ע"י האויב. אותו דבר קורא כאשר היריב הזיז חייל ומאיים על המלך, מצב זה נקרא שח ויש להסיר את האיום מהמלך. אם לא ניתן למנוע שח אז המצב נקרא שח-מט. מטרת המשחק היא להביא את המלך של היריב למצב של שח-מט, כלומר היריב לא יכול לעשות דבר כדי למנוע מהמלך להיות מאוים. (חשוב לציין שכמובן היא אפשר לעשות מהלכים שיוצאים מגבולות הלוח)

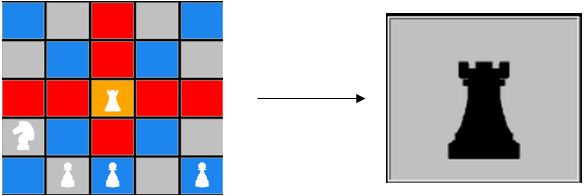
## מהלכי החיילים

### **סוס -**



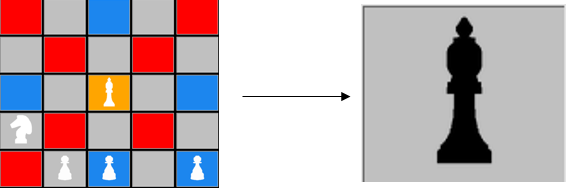
כמו שניתן לראות הסוס יכול לזוז שני צעדים קדימה ואחד הצידה או ההפך בכל הכיוונים, כלומר, כמו שניתן לראות, במקרה הטוב יש לו 8 מהלכים אפשריים.

### **צריח -**



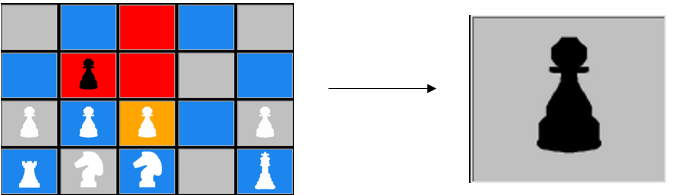
צריח יכול לזוז רק בטורים או בשורות ולא באלכסונים. כמו שניתן לראות בדוגמה, הצריח יכול לזוז רק בטור או שורה ולא יכול לעקוף חיילים אם הם חוסמים את הטור.

### **רץ -**



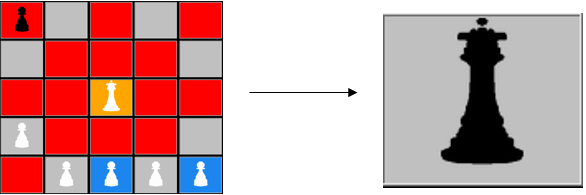
רץ יכול לזוז רק באלכסונים. כמו שניתן לראות בדוגמה, הצריח יכול לזוז רק בשני אלכסונים ולא יכול לעקוף חיילים אם הם חוסמים את המשך האלכסון.

### **רגלי –**



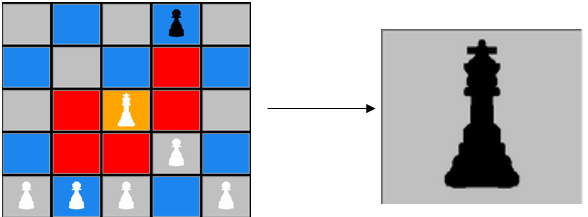
רגלי יכול לזוז שני מהלכים קדימה רק ממיקומו ההתחלתי. במצב הרגיל רגלי זז מהלך אחד קדימה ויכול לאכול חיילים של האויב באלכסון של מהלך אחד קדימה כמו שניתן לראות בדוגמה.

### **מלכה –**



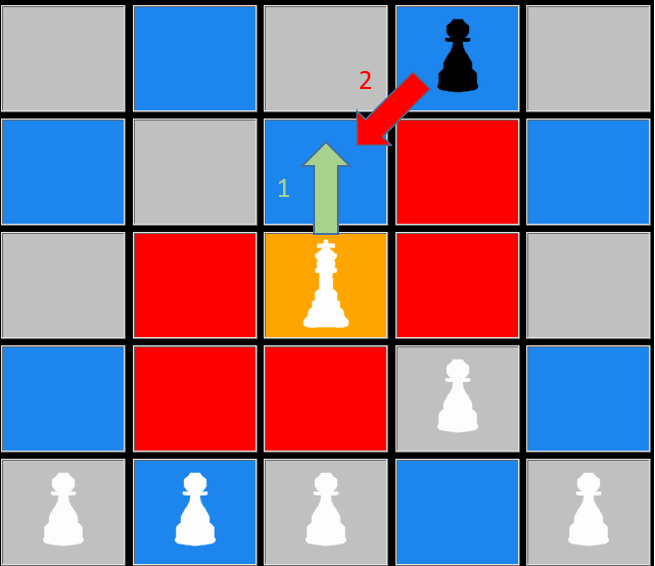
מלכה משלבת את המהלכים האפשריים של הרץ והצריח ביחד, כלומר, תזוזה גם באלכסונים, בטורים ושורות.

### **מלך –**



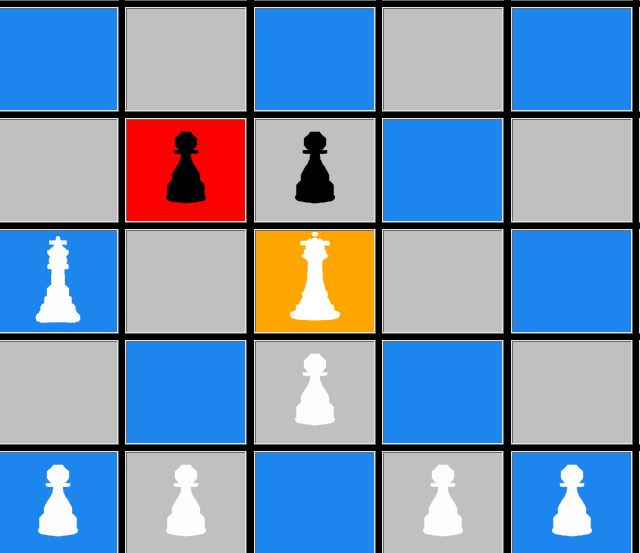
מלך יכול ללכת מהלך אחד קדימה בכל אחד מהמשבצות הנמצאות לידו. למרות זאת, אם המקום חסום בחייל מאותו צבע או המלך הולך להיות מאוים אז הוא לא יכול לזוז לשם. ניתן לראות זאת בדוגמה הנ"ל ובדוגמה הבאה אתן הסבר מה האיום. חשוב לציין כי המלך לא יכול לזוז למשבצת שמאלה למעלה כי הוא מאוים באותה נקודה והתמונה הנ"ל היא רק חלק קטן מכל הלוח.

### **איום על המלך –**



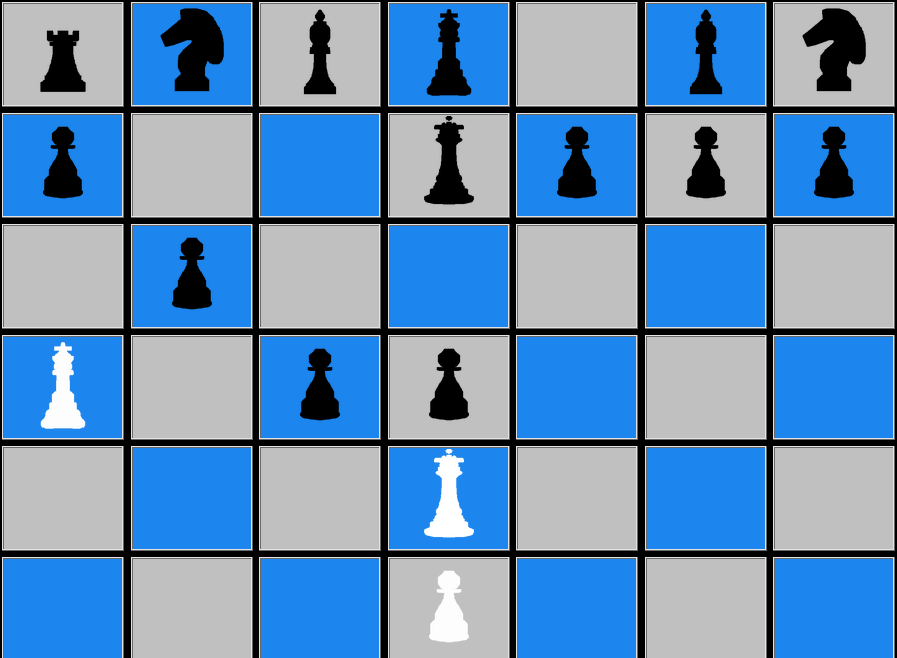
בדוגמה הזו ניתן לראות שנניח ונתן למלך להתקדם עם החץ הירוק אז אחריו החייל השחור יוכל לאכול אותו וזה לא יתכן, כלומר המלך יהיה מאוים אם יזוז לאותה משבצת. מקרה זה נקרא איום או שח עתידי.

### **שח –**



במצב הנתון אנחנו לוחצים על המלכה הלבנה ורוצים לראות את כל המהלכים האפשריים. כביכול אמורים להיות עוד מהלכים באלכסונים, בשורות ובטורים אך אנחנו רואים רק מהלך בודד אחד של לאכול רגלי שחור. אנחנו נמצאים במצב זה מכיוון שהמלך הלבן נמצא בשח, כלומר מאוים. כדי לצאת מהאיום כאשר משחקים עם המלכה הלבנה חייבים לאכול את החייל.

### **שח מט –**



במצב הנתון המלך הלבן גם מאוים בשח וגם לא יכול לזוז לשום מקום שישחרר אותו מהאיום בנוסף לכך ששום חייל לא יכול להסיר את האיום. מצב זה נקרה שח-מט.

# פרק 2 - מבנה הפרויקט ובסיסי נתונים

הפרויקט נכתב לפי כללי OOP נוקשים, כפי שניתן לראות בתרשים ה - UML שבהמשך.

דגש מרכזי בפרויקט היה על מבנה MVC או בצורה היותר מפורטת (Model, View, Controller). מבנה זה מגיע בעיקרו כדי לסדר את חלוקת המחלקות השונות למטרות השונות. בגדול יש שני חלקים מרכזיים, החלק הראשון הוא החלק הגרפי, החלק המוצג למשתמש במשחק. החלק הגרפי מתבטא בחלק הView. החלק השני הוא החלק הלוגי (Model) והוא עוסק בכל החישובים שנעשים מאחורי הקלעים, לדוגמה, המהלכים האפשריים במצב מסוים. כדי ששני החלקים יוכלו לתקשר ביניהם, בשביל הנוחות והסדר נוצר החלק המגשר תחת הכותרת (Controller) ומטרתו היא לנתב בין החלקים.

למימוש שחקן המחשב השתמשתי באלגוריתם Minimax ובשיטת האופטימיזציה Alpha Beta.   
באלגוריתם זה נפרס עץ וירטואלי המתאר מצבים שונים במשחק, אשר כל מצב מנוקד לפי היתכנותו לניצחון או הפסד. כדי לשמור את המצבים ולבחור את המצב הטוב ביותר, בשורש העץ מתקבלים כל המהלכים החוקיים אשר נשמרים ברשימה. במהלך חזרת הערך מהבדיקה בתוך העץ אל השורש בודקים מה המהלך הכי טוב. המהלך הכי טוב נשמר ומוחזר בסוף הבדיקה. חשוב לציין שבעץ Minimax ניתן להיתקע בלולאה אינסופית של מהלכים וזה מכיוון שעבור כל פעם יתכן והמחשב ימצא את המהלך הכי טוב כמשהו שקרה ממקודם. כדי להימנע מכך, פיתחתי שני מנגנונים הנמצאים בשורש העץ העוזרים ליצור משחק יותר ורסטילי.

על מנת להקל על העבודה, הפרויקט חולק לשלוש תיקיות אב שונות: תיקיית גרפיקה, תיקיית לוגיקה ותיקיית עזר. בכל תיקייה מאוחסנים קבצים האחראים על נושא התיקייה. למשל, בתיקיית הגרפיקה מאוחסנים קבצים העוסקים בייצוג הגרפי של המשחק, וכוללים אובייקטים גרפיים.  
מבנה מפורט יותר של הפרויקט, הכולל תרשים UML מקיף, תרשים זרימה של תורות המשחק, ו – use case, ניתן למצוא בסוף ספר הפרויקט.

## מחלקות הפרויקט

* ConstantValues**:** מחלקה המכילה בתוכה את כל הקבועים של המשחק, לרבות קבועים לוגיים וגרפיים, האחראית על בניית המשחק.
* SettingValues**:** מחלקה המכילה בתוכה את כל הערכים המשתנים בהתאם לשינויי המשתמש בתחילת הגדרת השינויים למשחק.
* Stack**:** מימוש שלי למחלקה של מחסנית בפייתון לשימוש בעץ הMinimax (לשמירת מהלכים כי עובדים על הלוח המקורי ורוצים לא לשנות אותו).
* Tools**:** מחלקה מרכזית המכילה בתוכה פונקציות רבות שחלקן גנריות המשומשות במחלקות רבות לאורך המשחק, לדוגמה פונקציית בדיקת המהלכים המתאימים לכל סוגי החיילים השונים. מחלקה זו היא סטטית.
* ImageCollection**:** מחלקה המכילה בתוכה את כל התמונות בהם משתמשים במשחק. מחלקה זו ממירה את התמונות לצורה בה ניתן לעבוד עם התמונות בפייתון. התמונות נשמרות במערך והמחלקה נכללת במחלקות גרפיקה רבות.
* Start**:** מחלקה הנקראית בעמוד הראשי (Main) אשר מכילה בתוכה את המחלקה האחראית על פתיחת העמוד.
* GameMenu**:** מחלקה המכילה את מה שרואים לפני תחילת המשחק. העמוד הראשי עם כל האפשרויות לשינוי הגדרות, קריאת החוקים, התחלת המשחק ויציאה.
* ViewPage**:** מחלקה הכוללת בתוכה את כל הגרפיקה הנראית על המסך מיד לאחד לחיצה על תחילת המשחק.
* ViewGamePlay**:** מחלקה הכוללת בתוכה את כל הגרפיקה התלויה בשינויים עבור המשחק עצמו, כמו כפתורים, לוח המשחק והצבעה על טור השחקן יחד עם הודעה של שח-מט.
* ViewBoard**:** מחלקה הכוללת בתוכה את כל גרפיקת הלוח, השינויים בה, קריאה למחלקה Controller המגשרת בין החלק הגרפי לחלק הלוגי ואת התחלת פונקציות הEvent כאשר לוחצים על משבצת כלשהי על הלוח. הלוח נשמר במטריצה של Labels.
* ViewTurn**:** מחלקה האחראית על הצגת טור השחקן (שחור או לבן) ועל הודעה האם יש שח מט.
* Controller**:** מחלקה המגשרת בין המחלקות הגרפיות והלוגיות. מחלקה זו קוראת לפונקציות רבות בחלק הלוגי ומחזירה לחלק הגרפי. מטרת המחלקה היא להקל על סידור המעברים בין המחלקות ולהבנה.
* Model**:** מחלקה עיקרית בחלק הלוגי השומרת בתוכה את הצורה הלוגית של המשחק במטריצה. מחלקה זו מבצעת שינויים כאשר התבצע Event בפונקציה onaction() ומחזירה ערכים בהתאם. פונקציית השינוי נקראת מ Controller. כמו כן, במחלקה זו קיים מילון של פונקציות המהלכים החוקיים, לפי סוג חייל, והמפתח הוא הערך המתקבל שלמעשה הוא הערך הנמצא במטריצה לפי שורה וטור. דוגמה למימוש זה במחלקה: (פנייה לפי ערך חייל)

function\_type = self.soldier\_dict.get(value)

* Computer**:** מחלקה המכילה בתוכה את כל המחלקות האחראיות על שחקני המחשב. במקרה הזה יש שתי מחלקות, האחת חכמה (AI) ואחרת בודקת מהלכים לפי "הצורה" בה אני רוצה (TestPlayer), למשל בצורה חכמה (AI), טיפשה (Bad AI), רנדומלית (Random Move) וכו'.
* AI**:** המחלקה הכוללת את שחקן המחשב עם "מוח ה"Minimax ( play\_ai()) וקוראת לעץ הבדיקה ( nega\_max\_root()). כמו כן, מחלקה זו כוללת בתוכה שתי שיטות להערכת מצב הלוח לשחקן, האחת לפי סוג חייל והשנייה לפי מיקום על הלוח. שתי הדרכים נשמרות במילון ונקראות כמפתח של ערך חייל המוחזר ממחלקת הModel הנכללת במחלקה זו. מבחינת קוד זה נראה כך: (value מתייחס לערך החייל על הלוח)

self.piece\_nikud\_dict.get(abs(value))+ self.location\_nikud\_dict.get(value)[y][x]

* TestPlayer**:** מחלקה הכוללת את שחקן הבדיקה לפי תנאים שבחרתי לבדוק. במקרה הזה של הפרויקט, המוח של שחקן זה הוא רנדומלי.
* Bishop, Rook, Knight, Pawn, Bishop**:** מחלקות סטטיות הכוללות בתוכן את פונקציות המהלכים החוקיים עבור כל סוג חייל.
* King**:** מחלקה סטטית הכוללת בתוכה את פונקציית המהלכים החוקיים עבור סוג חייל זה ואת הבדיקה האם המלך מאוים.

## שימוש במבני נתונים

בפרויקט נדרשתי לשמור הרבה נתונים בצורות שונות ועל כן נדרשתי לעבוד עם טיפוסי נתונים מגוונים. לדוגמה, עבור שמירת המהלכים החוקיים נדרשתי להצגה במטריצה ואילו עבור שמירת הפונקציות להערכה והמהלכים החוקיים נדרשתי לשיטה שתחסוך משפטי if else רבים. כמו כן, שמירת הלוחות הגרפי והלוגי בצורה נוחה לסידור הנתונים. אפילו כדי לחקות את הצורה של העץ Minimax ולבדוק על הלוח המקורי ללא שינויו הייתי זקוק למבנה המסוגל להסתנכרן עם הבדיקה בכל שלב. על כן בפרויקט נעשה שימוש רב במבני נתונים, בעיקר ב - list, stack וdictionary.

דוגמא רחבה לשימוש בכל אחד ממבני נתונים אלה בפרויקט.

**שימוש ב - list:** מטריצה (ליסט בתוך ליסט) שימשה לשמירת התאים השונים המרכיבים את לוח המשחק, כך שיש חשיבות לסדר התאים וניתן לפנות לכל אובייקט לפי שורה וטור.  
**שימוש ב - dictionary:** שמירת פונקציית הערכה ממוספרים לפי מספרם (ערך): מאפשר גישה קלה לפונקציות הערכה ומהלכים חוקיים. יעילות זו מתבצעת ביעילות השואפת לO(1).

**שימוש ב - stack:** שמירת מהלכים אחרונים שנעשו בעץ הבדיקה של הMinimax. בגלל המבנה החזרתי של הבדיקות, מחסנית מתאימה בצורה מצוינת לבדיקות על הלוח המקורי.

## זרימת המידע בפרויקט בדרכים השונות

### **הסבר כללי**

אחת הדברים הראשונים בהם נתקלתי בפרויקט היה תקשורת בין הקלאסים השונים (חיבור בין לוגיקה וגרפיקה) עבור Use Case שונים כמו: בנית הלוח, לחיצה על המסך, בדיקת שחקן המחשב, שינויי התצוגה למשתמש בהתאם לביצועים וכו'.

### **תקשורת בין האובייקטים: מימוש תבנית ה – Controller**

ניתן לראות כי להרבה קלאסים בסיסים יש קשר בין החלק הגרפי לחלק הלוגי. למעשה, כדוגמה: הלוח המוצג למשתמש אינו הלוח בו המחשב משתמש לביצוע הבדיקות. מה שקורה בפועל זה שהם מסונכרנים בשינויים ע"י מחלקת העזר Controller המגשרת על הפערים בין הלוגיקה והגרפיקה.

### **ניהול המשחק**

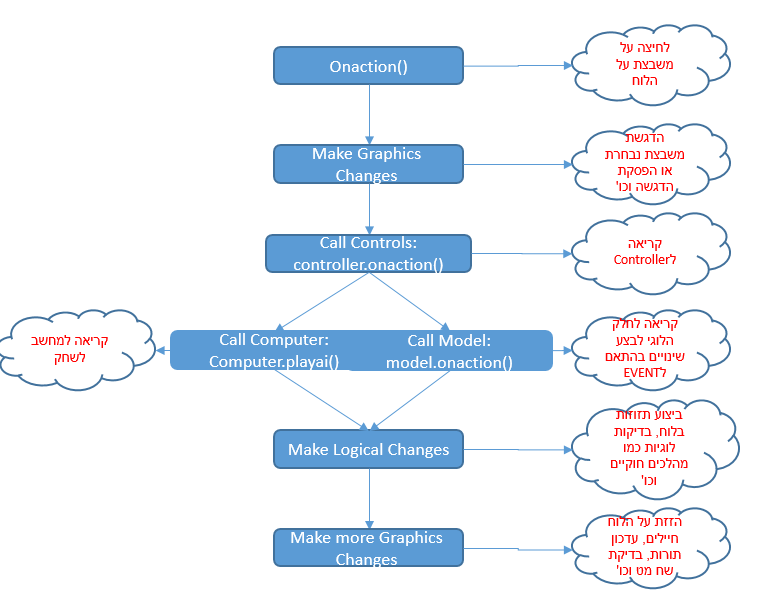
כאשר ה - tkinter.Label של האובייקט הגרפי (הלוח) נלחץ, נקראת המתודה Onaction() השייכת לאובייקט הגרפי. מכאן עולה בעיה ברורה: מכיוון שכל אובייקט גרפי מאחסן מידע הנוגע אך ורק אליו, לא יהיה ניתן לדעת רק באמצעותו האם התנועה שהשחקן מבקש לבצע חוקית. למעשה, האובייקט היחיד אשר יכול לשנות שינויים ללוח שהמחשב עושה עליו את השינויים הלוגיים הוא המודל, המחשב ובניו (Model, Computer, etc).

לפי כללי כתיבת OOP נכונה, כל קלאס עוסק רק במשימות עליו הוא אחראי ובאובייקטים הכפופים לו. ניתן לראות, אפוא, כי קלאסי הגרפיקה לא אחראים על החלק הלוגי ונהפכו. על כן, מימשתי את מחלקת העזר Controller, אשר מגשרת על הפער וקוראת לפונקציות עבור כל מחלקה המתאימה בין שני הצדדים.

### **הרחבה – Design pattern**

במילים אחרות מה שיצרתי הוא: design pattern ספציפי, בו משתמשים רבות בתכנות מונחה אירועים (Event oriented programming) ככלל ובממשק משתמש (GUI) בפרט: ה - Observer pattern.  
בתבנית זו, אובייקטים רבים מחוברים לאובייקט אחד, אשר מיידע אותם אילו פונקציות לבצע לפי אירועים שונים. אובייקט זה מכונה Observableובפרויקט Controller, בזמן שהאובייקטים המקושרים אליו מכונים Observers ובפרויקט מחלקות הלוגיקה.

נסביר את אופן פעולת ה - design pattern באמצעות המשחק.  
כאשר השחקן לוחץ על ה - tkinter.Label של הלוח, הפונקציה Onaction() המובנית בתוך ViewBoard מופעלת. פונקציה זו קוראת לכל הפונקציות האחראיות על השינויים הגרפיים וקוראת בין היתר לController שיקרא למחלקות שיעשו את השינויים הלוגיים בהתאם ולאחר מכן ממשיכה עם השינויים הגרפיים בהתאם. להלן תרשים זרימה סכמטי של לחיצה על הלוח ולסידור ניהול המשחק בהתאם בין המחלקות השונות:



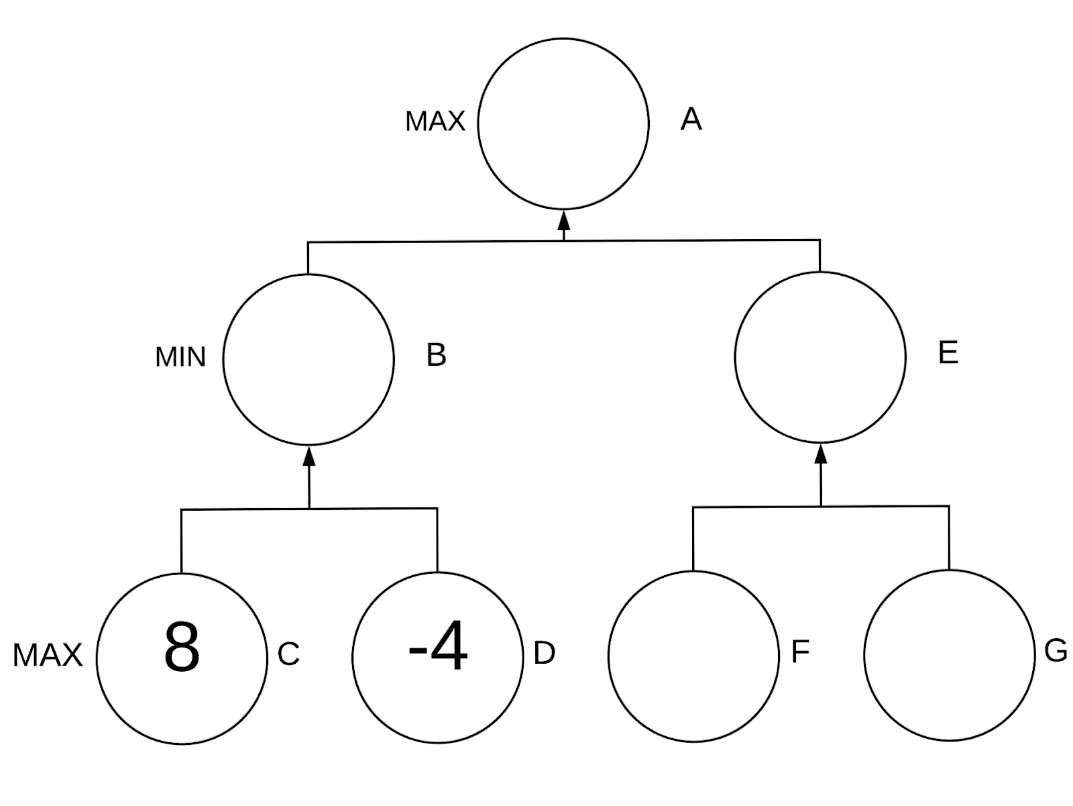
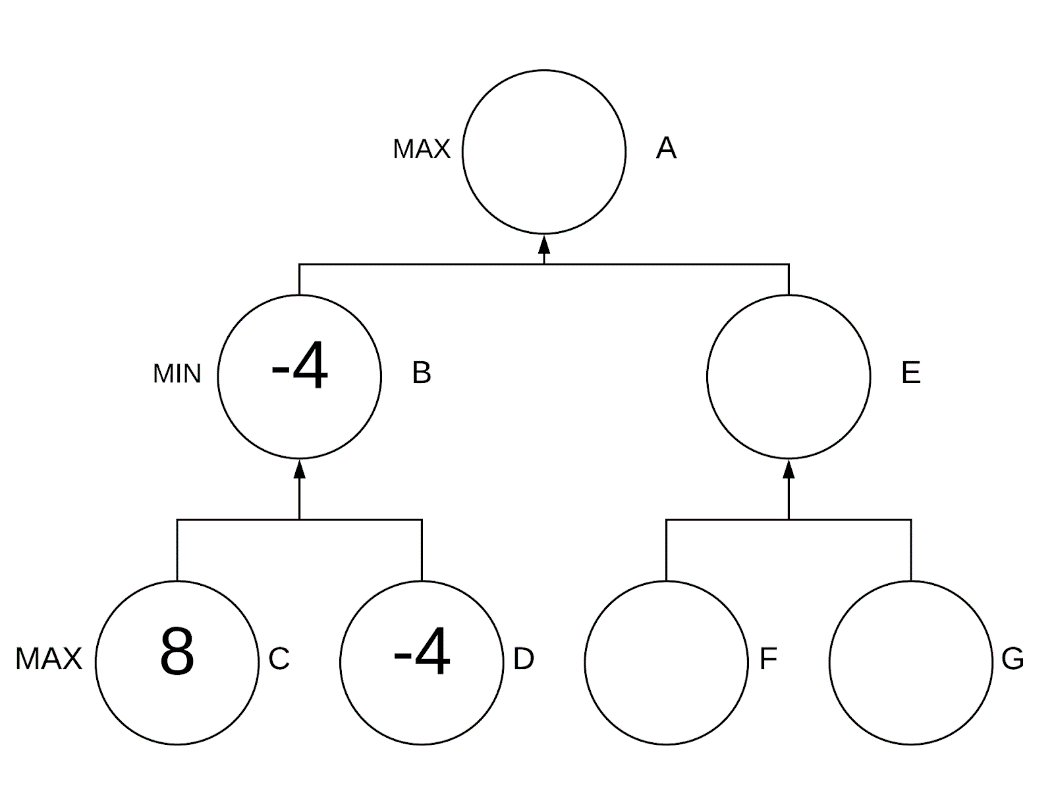
ניתן לראות בנספחים את שאר ההסברים של שאר הUse Case.

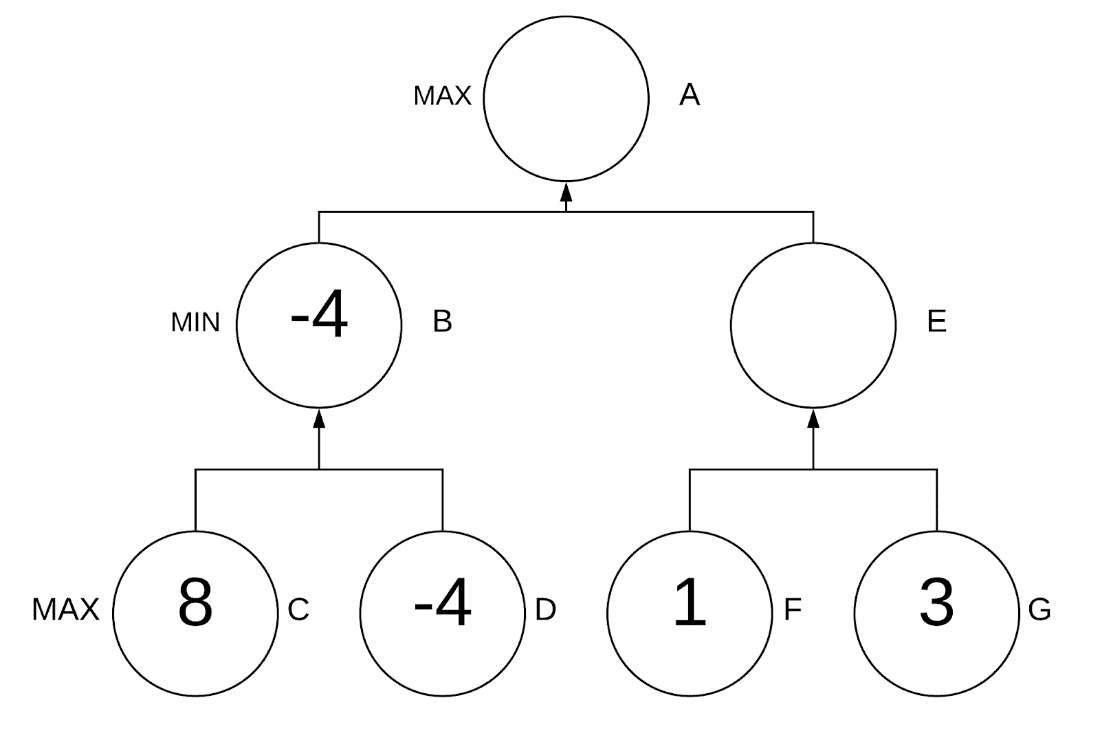
## אלגוריתם AI: שימוש ב - Minimax, AlphaBeta וStack

אחת הבעיות המרכזיות במשחק היא יצירת אלגוריתם ממוחשב המאתגר שחקן אמיתי ה"חושב כמוהו". על כן, בחרתי להשתמש באלגוריתם ה - Minimax, יחד עם שיטת האופטימיזציה AlphaBeta. אלגוריתם זה הוא אלגוריתם הפורס עץ של הלוחות האפשריים הבאים עד עומק ספציפי, כאשר בכל רמה של העץ מיקומו של חייל אחד משתנה. האלגוריתם עובר על העץ באמצעות DFS, וברמה האחרונה מבצע evaluation function: פונקציה המנקדת את הלוח הנוכחי, בטווח ציונים של פלוס מינוס אינסוף.

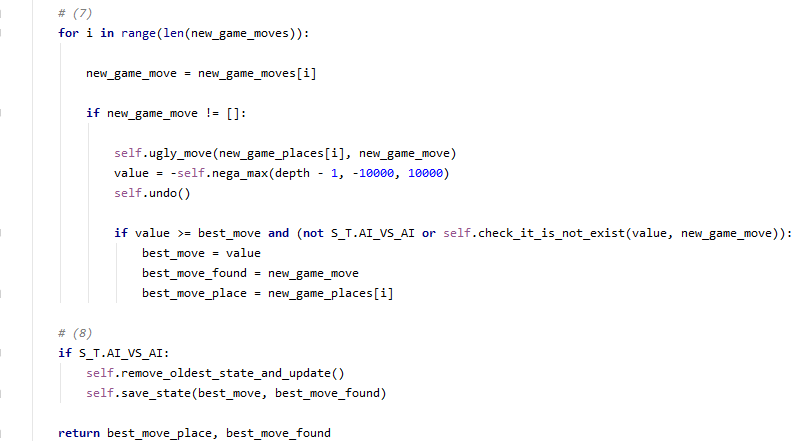
האלגוריתם מחולק כך שבכל רמה, זהות השחקן מתחלפת: ברמה הראשונה המחשב משחק, ברמה הבאה האדם משחק וחוזר חלילה. במימוש בפרויקט, המחשב הוא ה - Max, כלומר לוחות שציונם מקסימלי עדיפים למחשב, ולכן הצעדים הדרושים להגעה אליהם הם אלה שייבחרו על ידו. לכן, האדם הוא ה - Min, כלומר לוחות אשר ציונם מינימלי מועדפים עליו, והם ייבחרו על ידו.

על מנת לתאר את אופן פעולת האלגוריתם, נציג דוגמא לעץ Minimax, ונראה את השתלשלות האירועים בו.

1. האלגוריתם מתחיל עם משתנה depth = 2, המציין כי על האלגוריתם לרדת 2 רמות בעץ.
2. בתחילת החיפוש בעץ, האלגוריתם נמצא בצומת A. האלגוריתם מגלה שבצומת A, לשחקן ה - MAX יש 2 תזוזות אפשריות: צומת B או צומת C. (כל צומת מייצג מצב אחר של הלוח, אליו הגיעו על ידי תזוזות של החיילים). לכן האלגוריתם יורד לרמה הבאה, לצומת B. כעת depth = 2.
3. בצומת B, לשחקן ה - MIN יש 2 תזוזות אפשריות: צומת D או צומת C. לכן האלגוריתם יורד לרמה הבאה, לצומת C. במצב זה, depth = 1.
4. האלגוריתם בצומת C,  ו - depth = 0. במצב זה, האלגוריתם לא ממשיך לרדת יותר בעץ, אלא חוקר את הלוח הנוכחי, ומקבל שציונו הוא 8.
5. האלגוריתם חוזר לצומת B. מכיוון שצומת זה משוחק על ידי שחקן ה - MIN, האלגוריתם שומר את המספר 8 כציון המינימלי שהשיג עד כה, ואת צומת C כצומת הטוב ביותר עד כה.
6. האלגוריתם ממשיך לבדוק את צומת D, מכיוון שהוא עשוי להיות עדיף על צומת C, ולכן יורד לצומת D.
7. האלגוריתם בצומת D. במצב זה, depth = 0, ועל כן הלוח הנוכחי נבדק וזוכה לציון (4-).
8. האלגוריתם חוזר לצומת B. מכיוון שצומת זה משוחק על ידי שחקן ה - MIN, האלגוריתם שומר את המספר (4-) כציון המינימלי שהושג עד כה, ואת צומת D כצומת הטוב ביותר שהושג עד כה. ניתן לראות כי המספר 8 הוחלף במספר (4-), וכי הצומת C הוחלף בצומת D. על כן, הציון של צומת B יהיה (4-), והמהלך הטוב ביותר יהיה המהלך המוביל לצומת D.

1. כעת, האלגוריתם בצומת A. השחקן המשחק כעת הוא שחקן ה - MAX, ולכן האלגוריתם ישמור את המספר (4-) כמספר המקסימלי, ואת המהלך שהוביל לצומת B כמהלך האופטימלי. אך, צומת E יכול להביא תוצאה טובה יותר, ולכן האלגוריתם יורד לצומת E.
2. כעת, האלגוריתם בצומת E, ו - depth = 1. במצב זה, האלגוריתם מזהה 2 תזוזות אפשריות: לצומת F ולצומת G. האלגוריתם יורד לצומת F.
3. כעת, האלגוריתם בצומת F. המשתנה depth  = 0, ולכן האלגוריתם ינקד את הלוח הנוכחי, אשר מקבל את הציון 1. ציון זה מוחזר לצומת E.
4. האלגוריתם כעת נמצא בצומת E. מכיוון שהשחקן המשחק כרגע הוא שחקן ה - MIN, האלגוריתם ישמור את הציון 1 כציון המינימלי, ואת המהלך שהוביל לצומת F כמהלך האופטימלי. אך, יכול להיות מצב בו הצומת G תהיה טובה יותר לשחקן ה - MIN מאשר צומת F, ולכן האלגוריתם יורד לצומת G.
5. האלגוריתם כעת נמצא בצומת G. המשתנה depth = 0, ולכן האלגוריתם ינקד את הלוח הנוכחי, אשר מקבל את הציון 3. ציון זה מוחזר לצומת G.
6. כעת, האלגוריתם בצומת E. מכיוון שזהו תורו של שחקן ה - MIN, האלגוריתם ישמור את הציון 1 כציון המינימלי, ואת המהלך שהוביל לצומת F כמהלך האופטימלי. ציון זה מוחזר לצומת A.
7. כעת האלגוריתם נמצא בצומת A, וזה תורו של שחקן ה - MAX לשחק. מכיוון שהציון 1 עדיף על הציון   (4-), האלגוריתם יבחר את הציון 1 כציון המקסימלי, ואת המהלך שהוביל לצומת E כמהלך האופטימלי.
8. כעת, המעבר על עץ ה - Minimax נשלם, והמהלך האופטימלי עבור המחשב הוא המהלך אשר הוביל לצומת E.

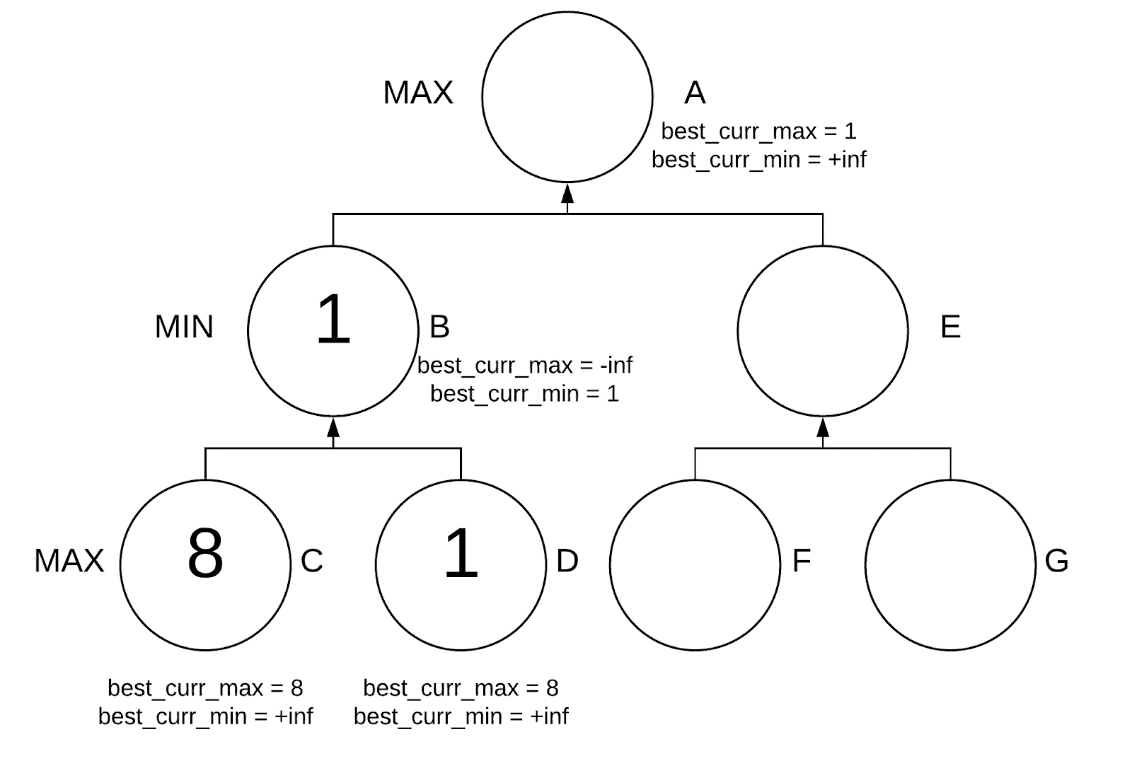
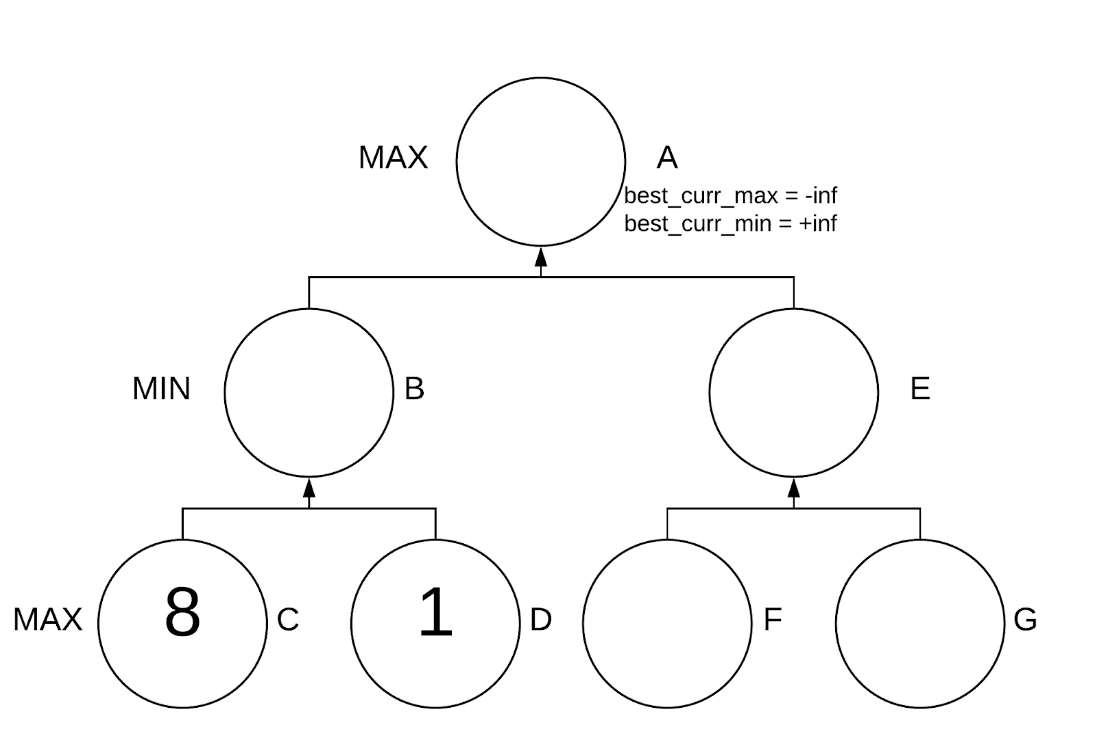
דוגמה לחלק מקוד פונקצית ראש עץ הAlphaBeta:



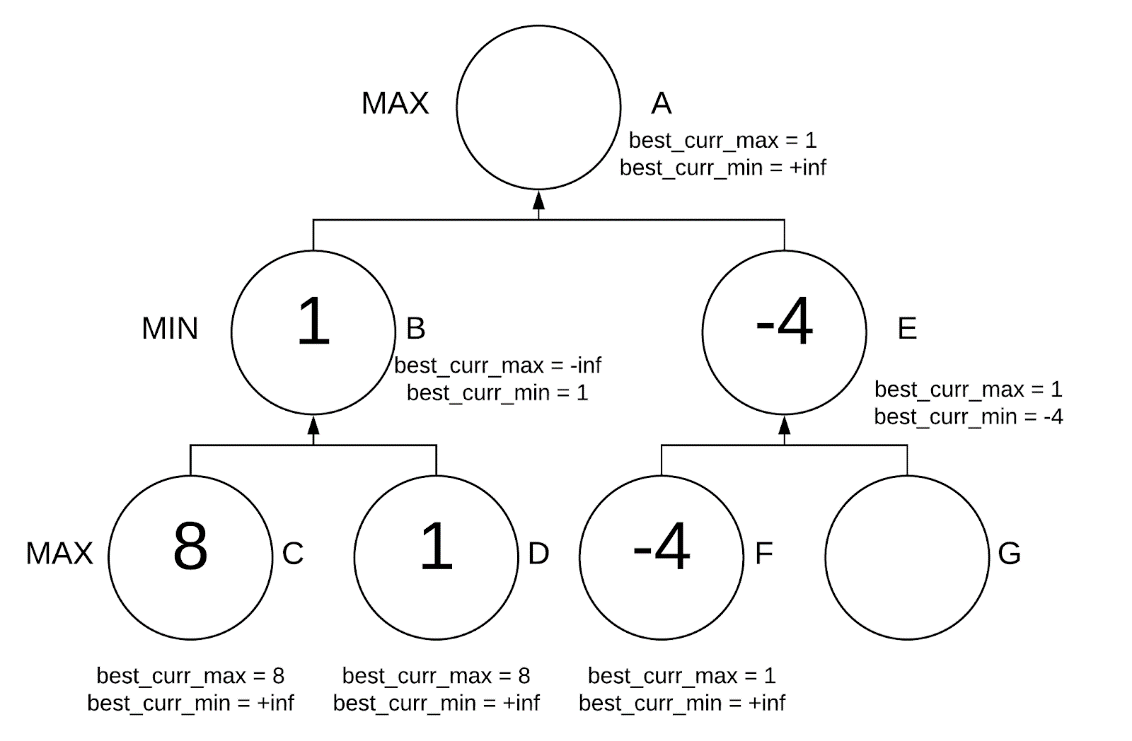
סקירה קצרה זו של האלגוריתם מאפשרת לראות כי יעילותו היא ,o(b^n) כאשר b מהווה את כמות המהלכים הממוצעת בכל רמה בעץ, ו - n מהווה את כמות הרמות שיורדים בעץ. האלגוריתם ללא שיפור הAlphaBeta אינו יעיל. על כן, פותחו שיטות אופטמיזציה לאלגוריתם, המסתמכות על ההנחה כי הן שחקן ה - Min והן שחקן ה - Max יבחרו תמיד באופציה הטובה ביותר עבורם. אחת משיטות אופטמיזציה אלו היא ה - AlphaBeta, אותו מימשתי בפרויקט.

בשיטה זו, עם הקריאה הראשונית לפונקציית ה – Minimax, מאותחלים 2 משתנים: alpha ו - beta, השומרים בהתאמה את הציון המקסימלי שהושג על ידי שחקן ה - Max עד כה, ואת הציון המינימלי שהושג על ידי שחקן ה - Min עד כה. על מנת לבאר את אופן פעולת האלגוריתם, נעזר בדוגמא:

הקריאה הראשונית לאלגוריתם מאותחלת עם alpha= -10000, ו - beta = 10000 (בציורים ייוצג תחילה כאינסוף ומינוס אינסוף).   
מהלך האלגוריתם כדלהלן:

1. האלגוריתם מגלה 2 תזוזות אפשריות עבור שחקן ה - Max בצומת A. על כן, הוא יורד לצומת B.
2. האלגוריתם נמצא בצומת B, כאשר depth = 1. במצב זה, alpha = -10000 ו -   
   beta = 10000. האלגוריתם יורד לצומת C.
3. בצומת C, המשתנה depth = 0. על כן, האלגוריתם מנקד את צומת C ומעניק ללוח זה ציון 8. הציון מוחזר לצומת B.
4. בצומת B, המשתנה ערך המינימום הטוב ביותר (מיוצג כbest\_curr\_min בציור) מתעדכן ל - 8, בזמן שהמשתנה ערך המקסימום הטוב ביותר (מיוצג כbest\_curr\_max בציור) נשאר 10000-. המספר הטוב ביותר שיש לשחקן ה - MIN בצומת B הוא 8, והמהלך האופטימלי הוא זה שהוביל לצומת C. במצב זה, ייתכן כי צומת D ינוקד באופן טוב יותר מאשר 8, ולכן האלגוריתם יחקור את הצומת D.
5. בצומת D, המשתנה depth = 0. האלגוריתם מנקד את הצומת ומעניק ללוח ציון 1. הציון מוחזר לצומת B.
6. בצומת B, המשתנה best\_curr\_min מתעדכן ל - 1, והציון 1 מוחזר לצומת A. כעת העץ נראה כך:
7. בצומת A, המספר 1 נשמר כמספר הטוב ביותר, המהלך שהוביל לצומת B כצומת הטוב ביותר והמשתנה best\_curr\_max = 1. ייתכן כי מעבר על תת העץ של צומת E יניב מהלך טוב יותר לשחקן ה - MAX, ולכן האלגוריתם יורד בעץ.
8. האלגוריתם כעת בצומת E, במצב בו best\_curr\_max = 1, best\_curr\_min = 10000.
9. האלגוריתם ירד לצומת F, ומקבל את הציון (4-). (מכיוון שהמשתנה depth התאפס). הציון המוחזר לצומת E הוא (4-).
10. בשלב זה, בצומת E, אלגוריתם האלפא ביתא נכנס לפעולה: כעת, best\_curr\_min מעודכן ל - (4-), ו - best\_curr\_max מעודכן ל - 1. חשיבה מדוקדקת מראה לנו כי לא משנה מה ערכו של הצומת G כלל. מכיוון שבצומת E שחקן ה - MIN משחק, ייבחר ציון הקטן או שווה ל - (4-). במקרה בו הציון של G גדול מ - (4-), למשל 6, שחקן ה - MIN יבחר את (4-). במקרה בו הציון של G קטן מ - (4-), למשל (7-), שחקן ה - MIN יבחר את (7-). ניתן לראות כי הציון המקסימלי האפשרי להשגה מצומת E הוא אפוא (4-).   
    אך, אני יודעים כי לשחקן A, שחקן ה - MAX, יש אופציה טובה יותר בצד השמאלי של העץ: הצומת B המנוקד 1. לכן, אין כל סיבה להמשיך לחפש בעץ, מכיוון שכך או כך ישנו מהלך טוב יותר לשחקן ה - MAX, והוא לא יבחר בצומת E.

ניתן לראות כי הבחנה זו התקבלה על סמך אי השוויון best\_curr\_min <= best\_curr\_max. סקירה של המצב ההופכי, בו ה - MIN גילה אופציה טובה יותר בענף אחר בעץ, תתבצע על ידי אותו אי שוויון בדיוק.

על כן, ניתן לעצור את החיפוש בעץ כאשר best\_curr\_min <= best\_curr\_max, או בשפה המקצועית, כאשר beta <=alpha .

1. בצומת A, האופציות האפשריות הן  (4-) ו - 1, ולכן האופציה הנבחרת היא 1.

חשוב לציין שלמרות שיפור הAlphaBeta אשר מקצר משמעותית את בדיקת לוחות המהלכים עדיין לאחר כ5-6 מהלכים מגיעים למספר אסטרונומי של בדיקות כך שאפילו המחשב מתקשה (בעיית הבדיקה הגדולה אליה התייחסתי במבוא).

כמו כן, לעיתים מתרחש מצב בו לאחר כמה מהלכים הלוח חוזר למצבו הקודם. מכיוון שהמחשב מניח כי השחקן האנושי בוחר את הבחירות האופטימליות לו, הוא ינהג בדיוק באותה דרך בה נהג ממקודם. בהינתן שהשחקן האנושי יישחק כפי ששיחק מקודם, ולא ישחק אחרת, המשחק ייתקע בלולאה אינסופית (בעיית לולאת המהלכים האינסופית הקוראת במשחקים מבוססי Minimax אליה התייחסתי במבוא). לכן יצרתי שני מנגנונים כדי למנוע מצב אינסופי במצב של מחשב נגד מחשב:

כדי למנוע את המצב דלעיל, החלטתי לשמור כל לוח שהיה בעבר ( עד 13 מהלכים אחורה). לאחר שאלגוריתם ה - Minimax מוצא מהלך אופטימלי, מתבצעת בדיקה אם הלוח לאחר המהלך שוחק בעבר. אם כן, נבחר מהלך רנדומלי וכן הלאה עד אשר מושג מהלך שלא שוחק בעבר. בחרתי את המספר 13 כי לאחר ניסיונות רבים נראה שמספר זה מתאים לשיבוש רוב הלולאות האינסופיות.

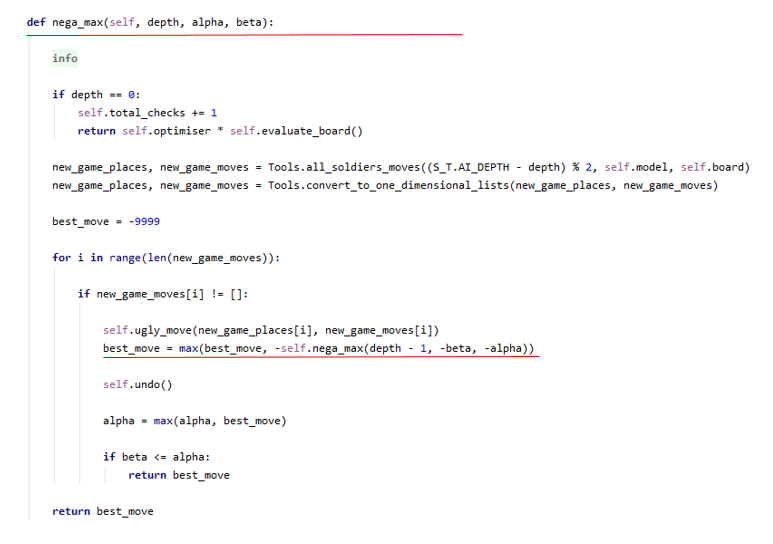
### **התלבטויות, פתרונות ושיפורים מתקדמים –**

מבחינת זיכרון שמירת המהלכים נשמרת ברשימה שסיבוכיותה היא o(n)וניתן לשפר זאת ע"י hash אך בשל האורך הקטן ההבדל כו זניח שהחלטתי להישאר עם רשימה. חשוב לציין כי בשח-מט אחרי מספר מסוים של חיילים על הלוח כמעט ולא אפשרי להגיע לשח-מט ועל כן במנגנון בדיקת חזרתיות המהלכים, אם כמות החיילים על הלוח קטנה מ-5 ושום מהלך רנדומלי שלא התבצע מקודם לא קיים אז יוכרע **שוויון**.

בנוסף, כדי למנוע דטרמיניסטיות יצרתי סיכוי של מספר / 1 שהחלטתי לסיכוי של מהלך רנדומלי ולאחר ניסויים רבים נראה שמגיעים למשחקים הרבה יותר מעניינים וחדשים.

## שיפור Minimax ל Negamax -

אלגוריתם הMinimax מתבסס על כך שמצד אחד הוא משחק עבורו, כלומר מחפש את הניקוד הכי גבוה של הלוח ומצד שני את הניקוד הכי נמוך עבור היריב שלו. כתוצאה מכך באלגוריתם הMinimax המקורי היה צורך בהעברת פרמטר נוסף שבודק אם צריך לשחק כמקסימום או כמינימום. אלגוריתם הNegamax בה לפתור את הבעיה הנ"ל בצורה מתמטית. למעשה מבחינה מתמטית בעץ, מקסימום שווה למינוס מינימום וההפך. ע"פ הנחה זו שהוכחה מתמטית כנכונה אפשר ייעל את האלגוריתם ולהחסיר פרמטר אחד בהעברה. הפתרון הוא בצורה הבאה:



(השורות הצבועות באדום הן אלו ששיפרנו)

השיפור הוא שבכל קריאה נוספת אין צורך בהעברת פרמטר נוסף לבדיקת מינימום או מקסימום כמו שניתן לראות בשורה הצבועה הראשונה. מבחינה מתמטית ניתן לראות את היישום הזה בשורה הצבועה השנייה. תמיד מחפשים את המקסימום של הניקוד הכי גבוה בין הניקוד השמור כגבוה ביותר לבין מינוס הבדיקה הנוספת. נוסף מינוס כי הבדיקה הנוספת היא עבור המינימום וכדי שמבחינה לוגית זה לא ישתנה מחליפים בין המיקום של הAlpha וBeta. בצורה זו חסכתי העברת פרמטר, עודף קוד ובדיקה של מקסימום או מינימום.

## אלגוריתם ניקוד - BFS (Breadth-first search)

שיטת ניקוד BFS מתייחסת לכך שבודקים כל פעם חלק אחד בבנים של צומת מסוימת בעץ הMinimax כמו שמוצג בתרשים הדוגמה למעלה ופוסלים אם צריך לבדוק עוד חלק או שאפשר להחזיר ערך. למעשה שיטה זו מייעלת עוד יותר את זמן הריצה ו"גוזמת" ענפים מהעץ. הגעתי להבנה שגישה זו היא הטובה ביותר מלמידה עצמית דרך ערוצים שונים באינטרנט של מתכנתים בנוסף לבית הספר.

אלגוריתם ניקוד הלוחות הכללי בו אני משתמש משתמש ב2 טכניקות ניקוד שונות:

1. ניקוד לפי סוג חייל
2. ניקוד לפי מיקום על הלוח

האלגוריתם עובר על כל המיקומים במטריצה ובודק את הערך עבור כל מיקום. הערך המוחזר הוא חיבור כל הנקודות מכל המטריצה. מה שמוחזר זה ערך הלוח במצבו הנוכחי. גם כאן הגעתי למסקנה ששיטה זו עובדת מאוד טוב מצפייה בערוצים שונים באינטרנט ומקריאה בפורומים כיצד לנקד כל חייל ואזור על הלוח.

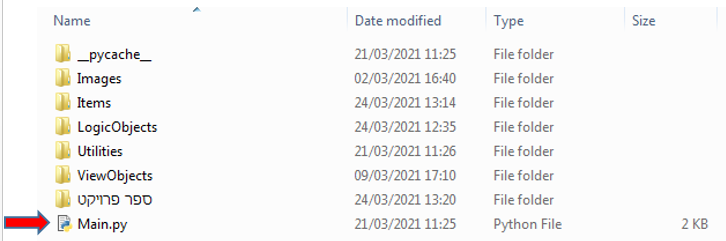
שתי טכניקות הניקוד שמורות במילונים שמפתחותיהם הם ערך החייל שמגיע ממיקום במטריצת המשחק. חיבור שתי אופציות הניקוד מחזירות את הניקוד בהתאם לערך החייל. ע"י שימוש במילונים נמנעתי מהשיטה ה"מכוערת" של מדרגות if else שרבים נוהגים לעבוד בה וגם ייעלתי את זמן הבדיקה וריצת התכנית. (ניתן לראות את קוד פונקצית הניקוד בנספחים)

# פרק 3 - מדריך למשתמש

במשחק ישנם 2 מסכים עיקריים: מסך הפתיחה ומסך המשחק.  מסך הפתיחה מהווה, כשמו, את הפתיחה למשחק, וממנו ניתן להגיע אל המשחק עצמו, או אל האופציות האחרות השונות. הן מסך הפתיחה והן מסך המשחק תוכנתו בעזרת ספריית tkinter ו – PIL  של פייתון.

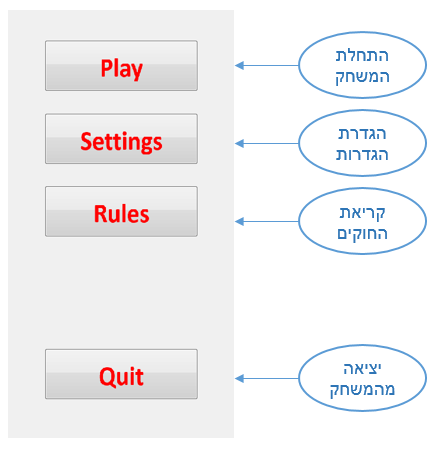
## הרצת המשחק

לאחר הורדת המשחק יש להיכנס לתיקיית המשחק (Chess) וללחוץ על הקובץ הנ"ל כדי להיכנס למסך הפתיחה:

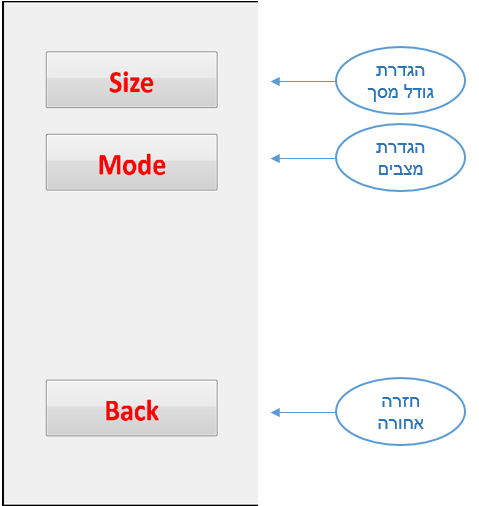


## מסך הפתיחה

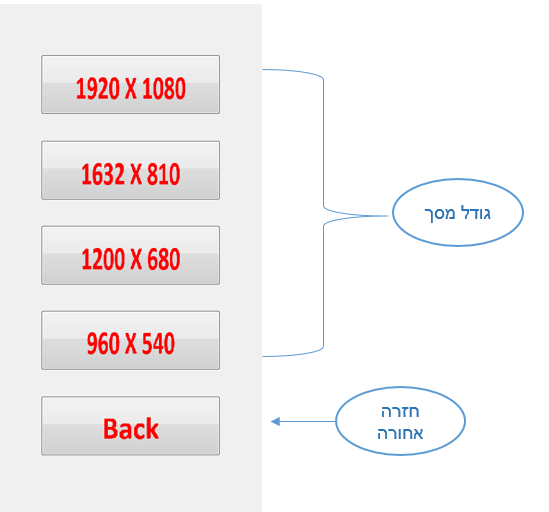
מסך ראשוני זה מופיע כאשר פותחים את קובץ המשחק. במסך זה ישנם מספר כפתורים שכל אחד מוביל למשהו אחר: (שאר העמודים הם דוגמאות לכל האופציות הקיימות)



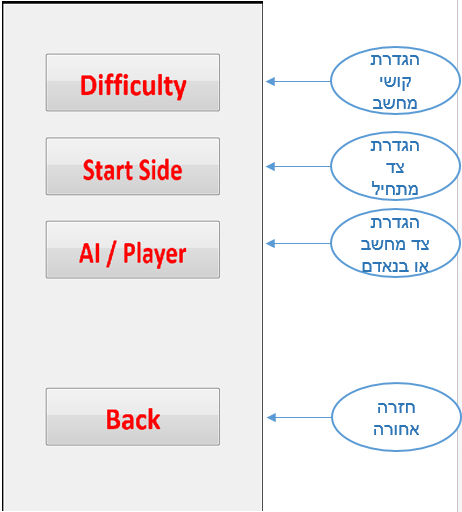
### **מעבר להגדרת הגדרות**



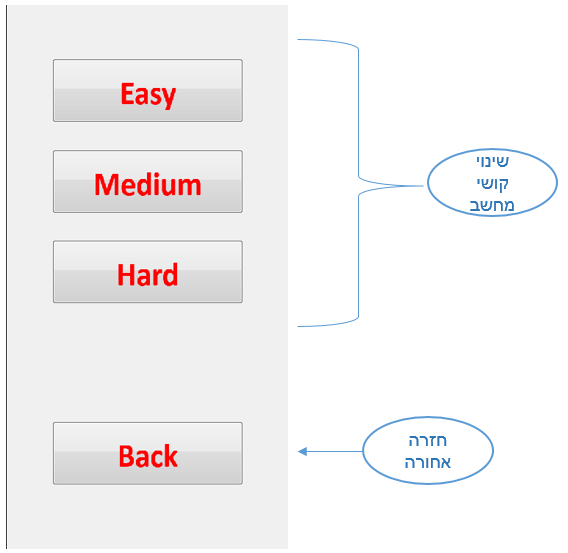
### **מעבר להגדרת גודל מסך**



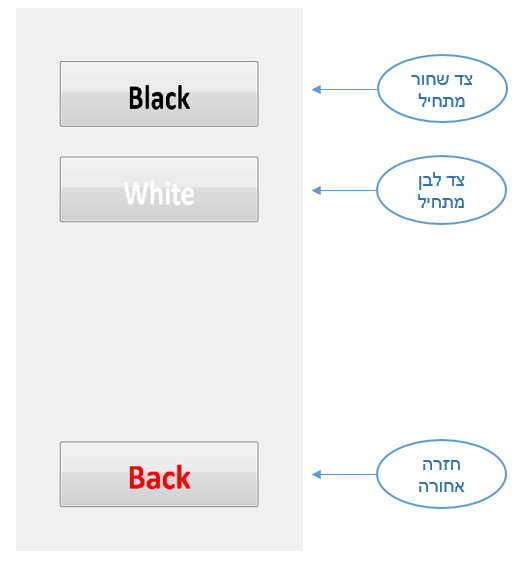
### **מעבר להגדרת מצבים**



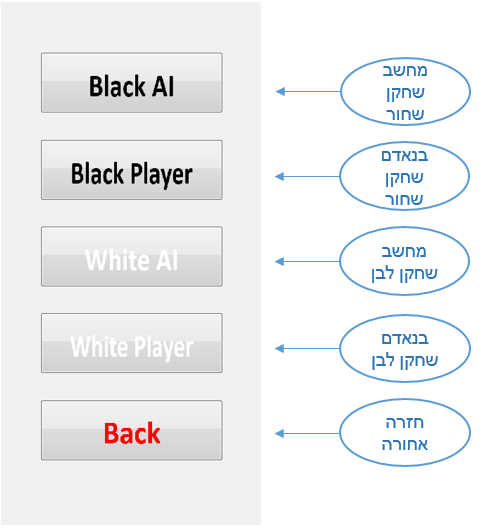
### **מעבר להגדרת קושי מחשב**



### **מעבר להגדרת צד מתחיל**

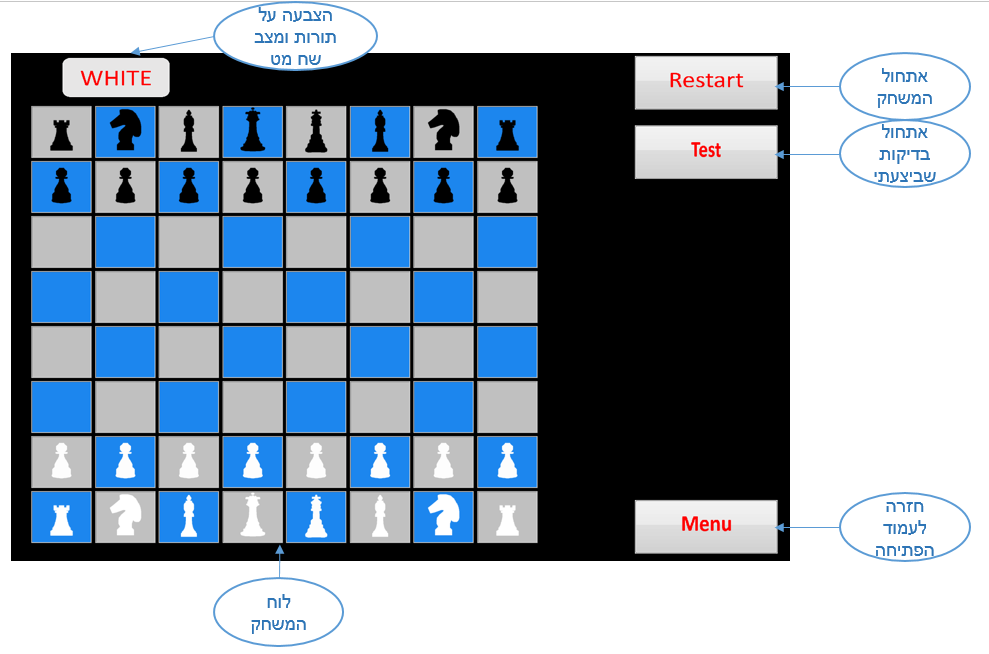


### **מעבר להגדרת צד מחשב או בנאדם**



## מסך המשחק

מסך זה מופיע בפתיחת המשחק, הלוח מוצג כדלהלן, כאשר בממשק המשתמש מוצגים כפתורים:



### **על הלוח**

בלוח מסודרים כלי השח מט כפי שנהוג בעולם. לכל שחקן יש שישה-עשר חיילים משישה סוגים שחורים. כברירת מחדל השחקן הלבן מתחיל והוא בנאדם והשחקן השחור הוא מחשב. ניתן לשנות את הגדרות אלו בחזרה אל אופציית ההגדרות.

### **על המשחק**

ראשית כאשר השחקן לוחץ על חייל המתאים עבורו נצבעות כל המשבצות עבורן הוא יכול להעביר את החייל. אם הוא לא רוצה להזיז את החייל הנ"ל הוא יכול ללחוץ עליו פעם נוספת ובכך הוא יכול לבחור חייל אחר. אם הוא לחץ על אחת מהמשבצות המסומנות, החייל זז לאותה נקודה ומתבצעים השינויים בהתאם. למשל: הצבעה על התור של השחקן השני, בדיקה אם יש שח-מט וכו'.

כאשר אחד משני הצדדים הצליח להגיע למצב של שח-מט מודפסת הודעת "checkmate" ומודפס בקונסול מי ניצח. לאחר מצב ניצחון או הפסד, כדי להתחיל משחק חדש יש ללחוץ על כפתור האתחול (Restart).

### **על האפשרויות השונות המוצאות בממשק המשתמש סביב הלוח**

Restart – אתחול המשחק למצב ההתחלתי כמו שקיבלנו אותו כאשר פתחנו את עמוד המשחק לראשונה.

Test – כפתור המאפשר לראות בדיקות של שחקנים שונים של מחשב מול מחשב. במקרה הזה בחרתי שהשחור יהיה שחקן חכם, כלומר עם מוח (Minimax) והשני עם מהלכים רנדומליים. במצב זה אחרי מספר משחקים מוגדר יסתיים המשחק בין שני הצדדים ויודפס כמה ניצחונות עבור כל צד וכמה שוויונות.

Menu – חזרה לעמוד הפתיחה

כל המאפיינים הללו מאפשרים משחקיות מהירה, נוחה לתפעול ופשוטה, המקלים את תפעול המשחק ומשפרים את חווית המשתמש.

# פרק 4 - סיכום אישי

כתיבת הפרויקט הייתה חוויה משמעותית ומלמדת במיוחד עבורי. בתחילת הכתיבה התמודדתי עם קשיים רבים, אשר נבעו בעיקר בגלל אי ההחלטה על מודל המחלקות ועל הצורה בה "אתקוף" את הפרויקט. בניגוד לפרויקטים רבים בהם כמות המחלקות בהן עבדתי היית קטנה יותר, פרויקט זה כלל מספר רב של מחלקות ועל כן היית צריך לסדר את מבנה המחלקות. כמו כן, בפרויקטים רבים בהם עבדתי בסביבות עבודה מובנות כמו Unity, בפרויקט הנ"ל היית צריך לבנות את היררכית הסידור כמעט מאפס עבור המשחק.  
בפרויקט עצמו למדתי ומימשתי מונחים רבים, כדוגמת ה - Observer Design Pattern, שימוש ב פונקציות הערכה יחד עם מימוש של טכניקות שונות, כדוגמת AlphaBeta, Negamax, עבודה על מחסניות, בדיקת ערכים לפי ערך במילון ועוד.

באמצעות כל אלה הצלחתי לכתוב קוד יעיל יותר, מתוחכם יותר ובעיקר מהנה יותר. כל אלה ועוד רבים אחרים שלמדתי בפרויקט אקח עימי להמשך. על כן, אין ספק כי הפרויקט היווה הזדמנות לימודית חשובה, בעיקר עקב העובדה כי פרויקט זה היה אחד מהגדולים ביותר אותו כתבתי אי פעם, מעל אלפיים שורות קוד ולקח יותר ממאתיים שעות.

ניהול הקוד, חלוקתו למודלים שונים וארגונו היוו אתגר בתחילה, אך גם לימדו אותי רבות על ארגון קוד, סידורו והביאו לתוצר אסתטי שהוא אבן דרך להמשך חיי.

כמילה אחרונה, ארצה להודות לעופר צשלר שתמך בי לאורך כל הדרך ונתן לי הערות בונות שעזרו להתקדם.

# פרק 5 - ביבליוגרפיה

* Code Bullet. (2018). I created an AI to Play Chess. <https://www.youtube.com/watch?v=DZfv0YgLJ2Q>
  + סרטון שמסביר את הרעיון של משחק השחמט.
* GitHub, Inc. (2021). Github. <https://github.com>
  + אתר בו מתכנתים משתפים קודים ורעיונות עבור פרויקטים מגוונים (השתמשתי עבור בדיקת רעיונות וגישות לנושאים כמו כתיבה נכונה של קוד בצורה המקובלת בתעשייה).
* Wikimdeia Foundation, Inc. (2021). Minimax. <https://en.wikipedia.org/wiki/Minimax>
  + אתר המסביר בכלליות על רעיון אלגוריתם המינימקס הגנרי (עזר לי לפתח את הרעיון של המוח למשחק שלי).
* Stack Exchange Inc. (2021). stackoverflow. <https://stackoverflow.com>
  + אתר בו מתייעצים בנושאים של באגים ובעיות שהיו למתכנתים אחרים (למשל: מהירות שמירה הכי מהירה של נתונים למצבים מסוימים).

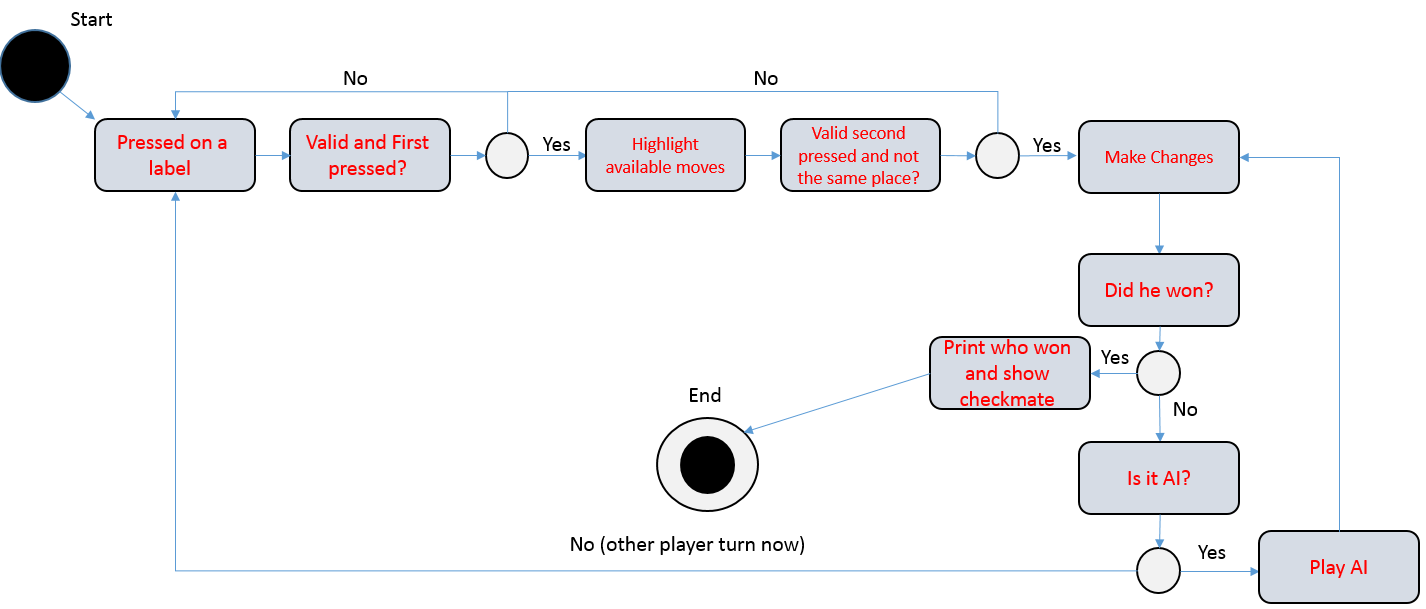
# פרק 6 - נספחים

ישנם 4 נספחים לספר פרויקט זה:

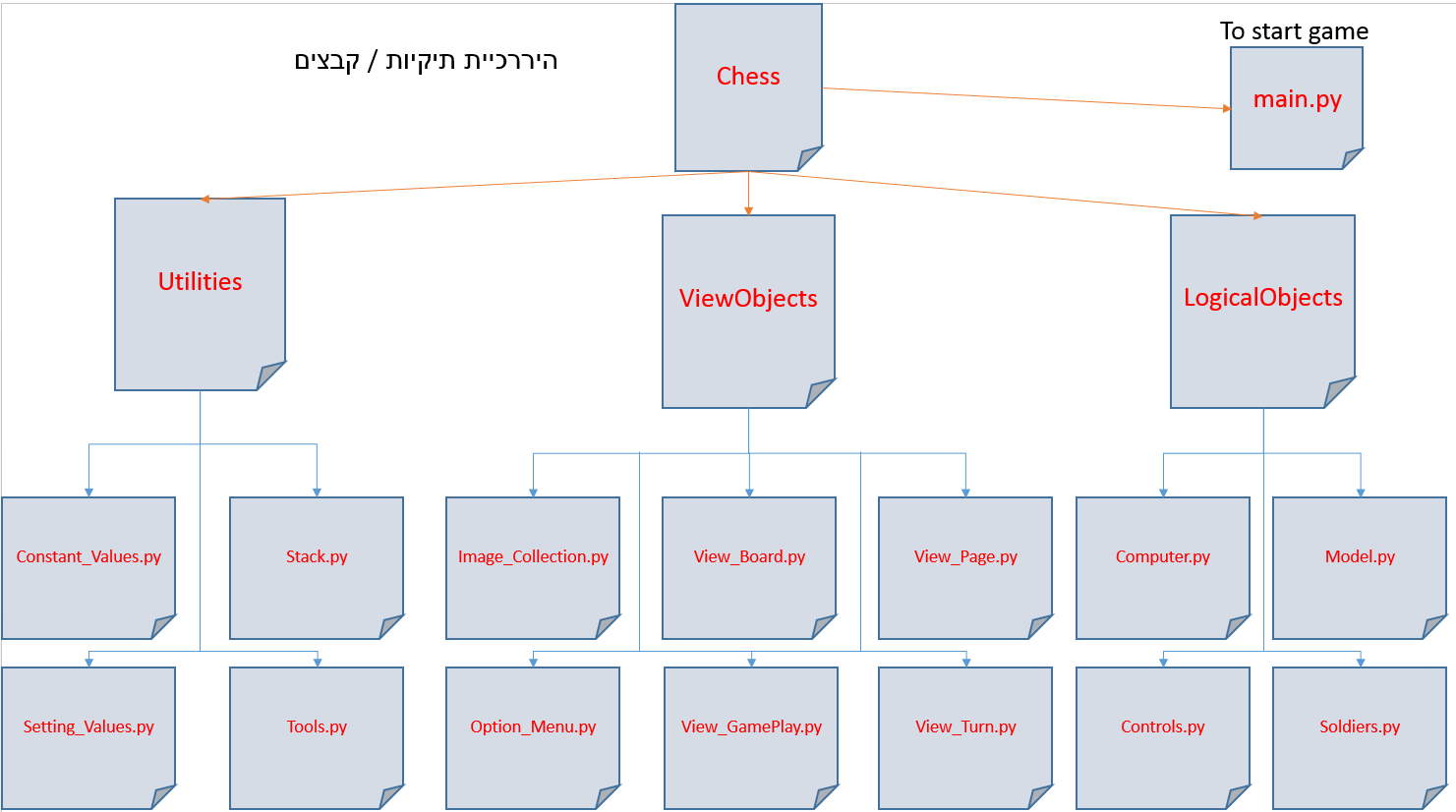
* תרשים זרימה כללי של התורות בפרויקט: Flow Diagram.
* תרשים סכמטי של היררכיית התיקיות בפרויקט יחד עם הרצת הפרויקט ותיקיות שיש להתקין.
* תרשים UML של כלל המחלקות בפרויקט.
* תרשים use case של  מצבים מרכזיים בפרויקט.
* קוד חשוב

נספחים אלו יוצגו, לפי סדר הופעתם, בעמודים הבאים.

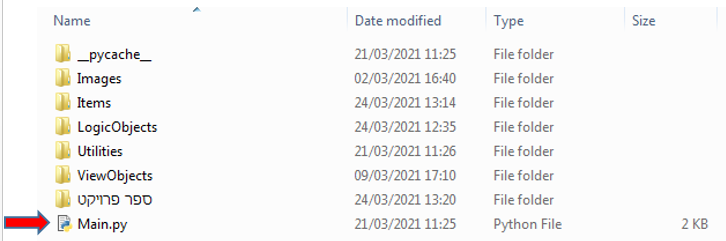
## זרימת תורות שחקנים (Flow Diagram)



## תרשים היררכית קבצים / תיקיות



## הרצת הפרויקט -

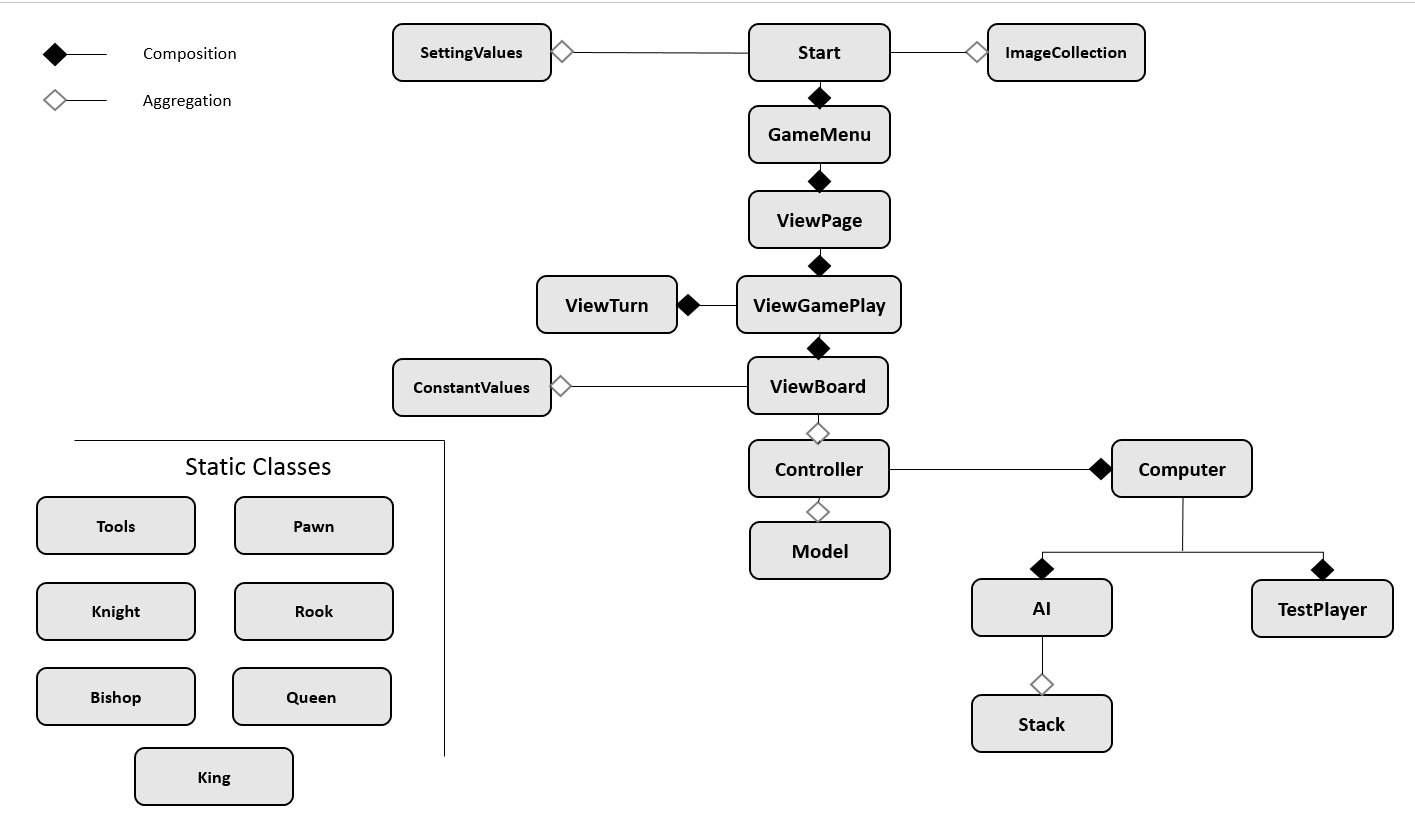


יש ללחוץ בתיקיית המשחק (Chess) על קובץ הmain.py ומשם עמוד הפתיחה ייווצר.

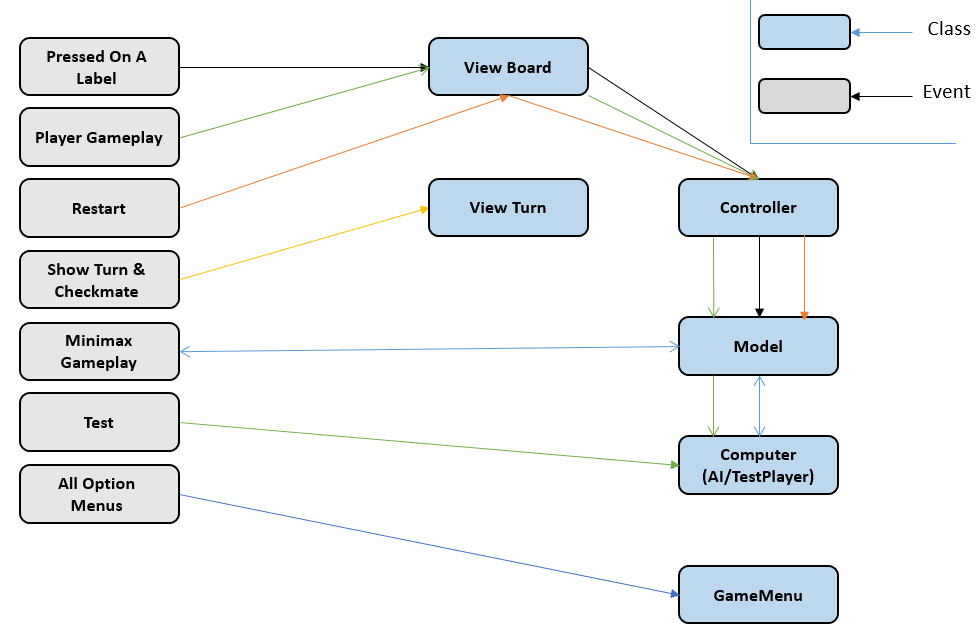
## חבילות להתקנה –

Random, tkinter, PIL, time

## תרשים UML

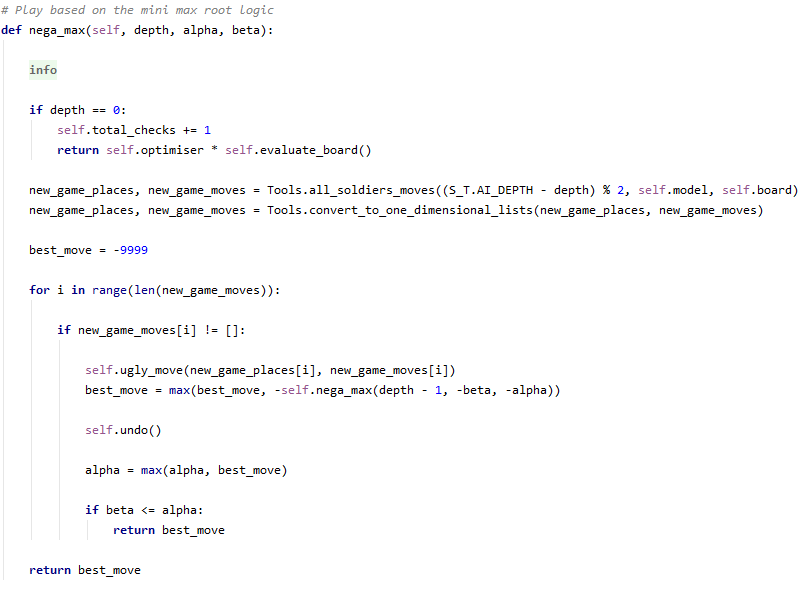


## תרשים Use Case

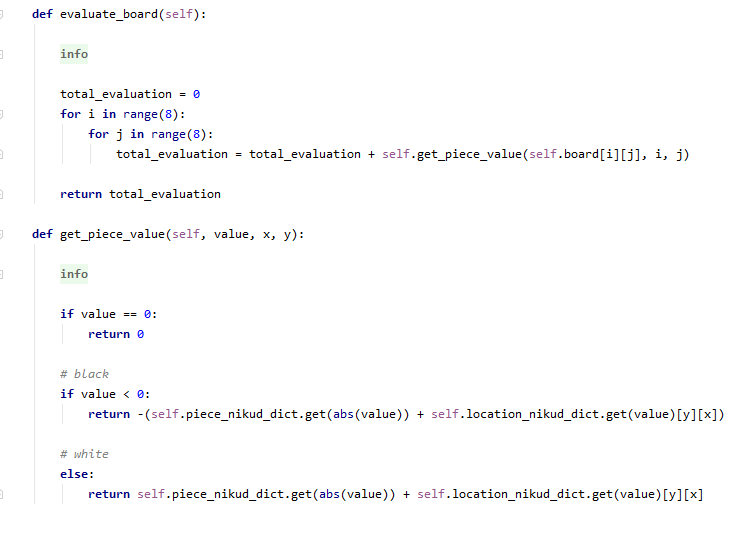


## צילומי קוד חשוב

**קוד Negamax**



**פונקציות ניקוד**



**ביצוע מהלך על הלוח המקורי ומחיקתו**

