**עבודות גמר בjava**

**נושא הפרויקט: "Queah"**

**מגיש: נתנאל חכמון**

**ת.ז:** 324199504

**מנחה: מריו סולאי**

**תאריך הגשה: 1.5.2022**

**תוכן**

[תקציר: 3](#_Toc101810876)

[תיאור הנושא: 3](#_Toc101810877)

[רקע תאורטי: 4](#_Toc101810878)

[תיאור הבעיה האלגוריתמית: 5](#_Toc101810879)

[סקירת אלגוריתמים בתחום הבעיה: 5](#_Toc101810880)

[אסטרטגיה 6](#_Toc101810881)

[מבנה נתונים 8](#_Toc101810882)

[תרשים מחלקותUML: 9](#_Toc101810883)

[ארכיטקטורה של הפתרון בפורמט Top down level design 10](#_Toc101810884)

[תיאור סביבת העבודה ושפת התכנות 11](#_Toc101810885)

[אלגוריתם ראשי 11](#_Toc101810886)

[תיאור ממשקים מחלקות ופונקציות ראשיות בפרויקט 14](#_Toc101810887)

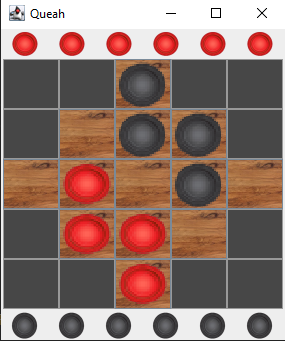
[התוכנית הראשית 18](#_Toc101810888)

[מדריך למשתמש 19](#_Toc101810889)

# תקציר:

הפרויקט שבחרתי לעשות הוא queah. זהו משחק בו ניתן לשחק מול המחשב או מול שחקן אחר. בפרויקט זה שילבתי את הנושא תכנות מונחה עצמים ומימשתי אלגוריתמים שונים. בנוסף, המשחק בנוי על אסטרטגיה שיצרתי ומאפשר לשחק בלוחות בגדלים שונים.

# תיאור הנושא:

**לוח המשחק**: המשחק משוחק על לוח מרובע משופע או אלכסוני עם 13 רווחים בלבד.

**כללי משחק הבסיסיים:**

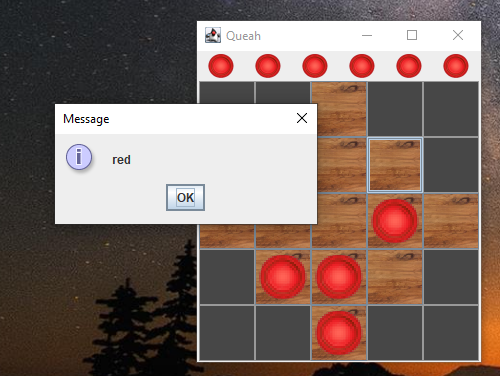
משחק ל – 2 משתתפים, 10 אבני משחק לכל משתתף 4 אבנים על הלוח.

מטרה: "לאכול" את כול האבנים של המשתתף השני.

תנועה על-גבי הלוח: כול שחקן בתור שלו יכול להזיז אבן למקום הצמוד הפנוי אל אותו אבן. שחקן יכול לאכול חתיכת אויב בקפיצה קצרה בגודל של שתיים. הכלי של השחקן חייב להיות צמוד לכלי האויב, ולנחות על שטח פנוי בצד השני. הלכידה חייבת להיעשות בכיוון אורתוגונלי בהתאם לעיצוב המלוכסן או האלכסוני של הלוח. ניתן ללכוד רק חתיכת אויב אחת בכל תור. חתיכה שנתפסה מוסרת מהלוח.

מהלך המשחק:

שחקנים מחליטים באילו צבעים לשחק, ומי מתחיל ראשון. אם כלי של שחקן נתפס, אז השחקן בתחילת התור הבא שלו חייב לקחת חתיכה אחת מהרזרבה שלו, ולשחרר אותו על כל מקום פנוי. יש להחזיר תמיד את מספר הכלים של שחקן על הלוח לארבעה, אלא אם כן השחקן מיצה את הרזרבה שלו. ששחקן יכול להוריד רק חתיכה מהרזרבה שלו, אם אחד הכלים שלהם נכבש בתורו האחרון של היריב.

**סיום המשחק:**

שחקן אדום ניצח

בגלל שלשחור לא

נישאר חיילים-

 שחקן שחור ניצח

בגלל שלאדום לא

נישאר חיילים-

# רקע תאורטי:

Liberian Queah הוא משחק אסטרטגיה מופשט לשני שחקנים מליבריה. השם הרשמי של המשחק הזה אינו ידוע, מכיוון שהוא לא צוין כאשר המשחק נכתב לראשונה. המשחק הוקלט לראשונה בשנת 1882, שחקניו היו חברים בשבט הקואה בליבריה. הלוח המסורתי עשוי מסורג של זרדים, ומקלות יוצרים את החלקים. את החלק העליון של המקלות חותכים בצורה מלוכסנת מצד אחד, ונקראים "גברים", ואילו המקלות של הצד השני נחתכים ישר ונקראים "נשים".

# תיאור הבעיה האלגוריתמית:

במהלך העבודה על הפרויקט נתקלתי במספר בעיות אלגוריתמיות אותן הייתי צריך לפתור כדי שהמערכת תעבוד כמו שצריך:

בדיקת תקינות של המהלך - כאשר המשתמש מבצעה מהלך המחשב בודק אם הוא תקין.

בניית אלגוריתם לשחקן ממוחשב – ליצור שחקן אשר פועל לפי המהלכים של היריב. יש למחשב אלגוריתם אשר סורק את הלוח שומר את כול המהלכים האפשריים לכול חייל וגם נותן משקל לכול אחד מהחלקים שלו.

מציאת כל המהלכים האפשריים לחייל מסוים- מציאת כל האפשרויות למהלכים עבור אחד החיילים ממצב לוח מסוים.

# סקירת אלגוריתמים בתחום הבעיה:

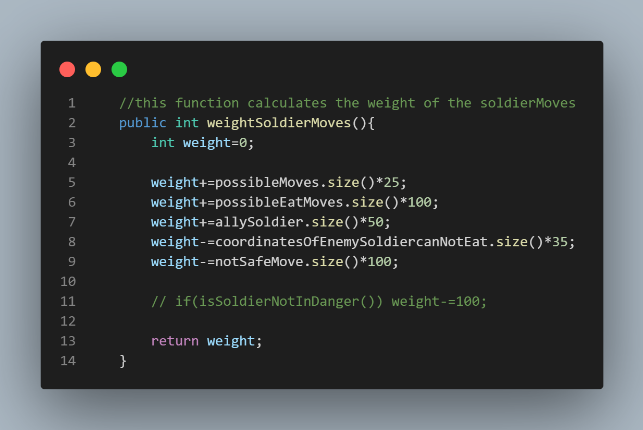
בדיקת תקינות של המהלך - הבדיקות תקינות שהיו הם: אם השחקן מזיז את החייל שלו או שם חייל חדש. אם הוא מזיז אז אם הוא ביצעה אכילה או הזזה רגילה. אם זה אכילה(כאשר יש עד 4 אפשרויות של אכילה. אכילה מתבצעת בקפיצה קצרה בגודל של שתיים ובאמצע יש את היריב) אז האם האכילה הייתה חוקית ואם זה הזזה(כאשר יש עד 4 אפשרויות של הזזה. הזזה מתבצעת בכך שצריך להזיז אבן למקום הצמוד הפנוי אל אותו אבן) אז האם הזזה הייתה חוקית. השחקן שם חייל חדש(בכול מקום רק בלוח) רק אם חייל שלך נאכל בתור הקודם וגם אם נישאר לך חיילים שאתה יכול לשים.

בניית אלגוריתם לשחקן ממוחשב – למחשב יש 3 אפשריות רמה שונות בכול האפשרויות האופציה הראשונה תהייה תמיד לאכול אחר כך אם זה רמה קלה אז הוא יבדוק אם יש חיילים שעומדים להיאכל ואם כן אז הוא יזיז את החייל לאחד מהאפשרויות הזזה הרנדומליות שיש לחייל אם לא אז הוא יבחר חייל בצורה רנדומלית ומזיז אותו לאחד מהאפשרויות הרנדומליות. ברמה בינונית הוא יבדוק אם יש חיילים שעומדים להיאכל ואם כן אז הוא יזיז את החייל לאפשרות הזזה אם המשקל הגבוה ביותר אם אין חייל שעומד להיאכל אז הוא מחשב לכול אחד מהחיילים את המשקל שלו ולוקח את החייל אם המשקל הגבוה ביותר ומזיז אותו לאפשרות אם המשקל הגבוה ביותר. המשקל מתחשב במה שקורה מסביב לחייל ב 2 בלוקים לכול כיוון ( למעלה למטה ימינה ושמאלה ). כאשר זה הרמה הקשה החייל משתמש באלגוריתם Negamax ובוחר את ההזזה לפי אלגוריתם זה.

מציאת כל המהלכים האפשריים לחייל מסוים - מציאת כל האפשרויות למהלכים עבור אחד החיילים ממצב לוח מסוים בכך שמקבליים את קואורדינטות של חייל וזה והאלגוריתם מסורק 2 בלוקים לכול כיוון ( למעלה למטה ימינה ושמאלה ) ובודק אם יש אכילה , הזזה או אין כלום. הוא שומר את כול ההזזות האפשריים ברשימה וכך גם האכילות האפשריים. וזה בודק גם אם החייל יכול להיאכל על ידי היריב.

# אסטרטגיה

האסטרטגיה מתחלקת לכמה חלקים שהם:

1. דירוג החייל של השחקן – דירוג זה מתבצע על מתן נקודות לפי פרמטרים וחישובם כך. כמות אפשרויות הזזה של החייל כפול 25 + כמות חיילים של היריב אשר צמודים לחייל ואפשר לאכול אותם כפול 100 + כמות החיילי בירית הצמודים לחייל כפול 50 – כמות חיילים של היריב אשר צמודים לחייל ואי אפשר לאכול אותם כפול 35 – כמות המהלכים של החייל שיגרמו לו להיאכל על ידי היריב.
2. דירוג הלוח של שחקן – הדירוג מתבצע על ידי חישוב וחיבור הדירוג של כול אחד מהחיילים של השחקן.



1. דירוג הלוח עצמו - הדירוג מתבצע על ידי מתן נקודות לשחקן וליריב שלו על ידי פרמטרים זהים ובסוף להחזיר את ההפרש. פרמטרים הם הדירוג של הלוח השחקן ,כמות הכלים של השחקן , הדירוג של הלוח של היריב ו כמות הכלים של היריב.



1. הוספת חייל חדש – כול אחד מהמקומות בלוח מדורג אם ניקוד קבועה וכאשר מוסיפים חייל חדש עדיף לשים את החייל במיקום הפנוי אם הניקוד הגבוה ביותר.



# מבנה נתונים

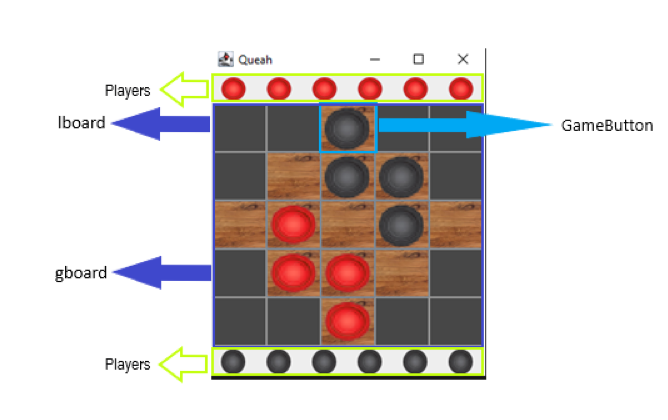
בפרויקט יש כמה מבני נתונים שונים ולכול אחד יש מטרה שונה:

1. יש מטריצה דו ממדית דינמית(שלוש אופציות: קטן ,בינוני וגדול) של int בשם lboard והיא שומרת בכול אחד מהמשתנים במערך את האופציות האלה(1- אם זה לא פעיל/לא קשור ללוח , 0 משבצת ריקה , 1 משבצת אם חייל אדום ו 2 משבצת אם חייל שחור)
2. יש מטריצה דו ממדית דינמית(שלוש אופציות: קטן ,בינוני וגדול) של GameButton(מחזיק תמונה וכפתור) בשם gboard שהיא אחראית לשמירה והצגה גרפית של הלוח.

1. מערך חד ממדי בגודל 8 של int בשם test שימושו היא בשליחת מידע ועדכון מידה כאשר המחשב מבצעה מהלך. כול תא אומר כך.

[newRow][ newColumn][ previsRow][ previsColumn][ eatRow][ eatColum][ isEat][ isSoldierLeft]

1. מחלקה בשם Coordinate אשר שומרת שלוש משנים של int בשם row, column, value. והמטרה שלה היא לשמור את השורה, עמודה ומידע.
2. יש מחלקה בשם SoldierMoves אשר ובתוכו יש מבנה נתונים הללו
   * 1. רשימה של Coordinate בשם possibleMoves ששומרת את כול ההזזות החוקיות של החייל.
     2. רשימה של Coordinate בשם possibleEatMoves ששומרת את כול האכילות החוקיות של החייל.
     3. רשימה של Coordinate בשם notSafeMove ששומרת את כול ההזזות החוקיות אך יגרמו לחייל להיאכל בתור הבא.
     4. רשימה של Coordinate בשם allySoldier ששומרת את כול חיילי ברית אשר צמודים לחייל.
     5. רשימה של Coordinate בשם coordinatesOfEnemySoldiercanNotEat ששומרת את כול החיילים של היריב הצמודים לחייל ואי אפשר לאכול אותם.

כך מתבטאים מבני הנתונים השייכים לגרפיקה.

# תמונה שמכילה טקסט, צילום מסך, מקורה התיאור נוצר באופן אוטומטיתרשים מחלקותUML:

# ארכיטקטורה של הפתרון בפורמט Top down level design

* התוכנית הראשית Game מכילה 3 פאנלים: Bottom, Center, Top .
* Top ו Bottom מכילים את הפאנל Players/ Computer.
  + המחלקה Computer יורשת ממחלקה Players ומוסיפה פונקציות אשר מאפשרות למחשב לשחק ולחשוב. המחלקה משתמשת במחלקות הללו.
    - soldierMovesStack - זהו מחסנית של SoldierMovesשימוש המחלקה היא לשמור ולסרוק את המהלכים של חייל היא משתמשת במשתנים והפונקציות ההלו.
      * SoldierMoves משתמש במחלקה Coordinate אשר מוזכרת בסעיף "מבנה נתונים" מס' 4.
      * המשנים מוסברים בהרחבה בסעיף "מבנה נתונים" מס' 5.
      * הפונקציות מוסברות בהרחבה בסעיף "הפונקציות/ המחלקות הראשיות בפרויקט".
    - Computer משתמש גם במחלקה Negamax. הפונקציות והמשתנים מוסברים בהרחבה בסעיף "הפונקציות/ המחלקות הראשיות בפרויקט".
    - שאר הפונקציות והמשתנים מוסברים בהרחבה בסעיף "הפונקציות/ המחלקות הראשיות בפרויקט".
  + המחלקה והפאנל Players(תפקידה של המחלקה היא לשמור את כמות החיילים של השחקן על הלוח וגם את כמות החיילים שנמצאים ברזרבה) מכילה את המשתנים והפונקציות הללו.
    - soldier\_left – מספר החיילים שנשארו.
    - soldier\_on\_board – מספר החיילים בלוח.
    - player\_color – צבע השחקן.
    - Copy – פונקציה שמעתיקה שחקן.
    - removeSoldierFromStack – פונקציה שמורידה חייל מהרזרבה גם בצורה גרפית
    - removeSoldierFromBoard – פונקציה שמורידה מכמות החיילים בלוח.
    - addSoldierToBoard – הפונקציה מוספיה שחקן לכמות השחקנים שבלוח.
* Center מכיל את הפאנל QueahBoard שמטרתו היא להיות לוח המשחק. המחלקה מכילה מספר פונקציות ומשתנים.
  + lBoard – מערך דו ממדי של מספרים שלמים ומטרתו לשמור בכול אחד מהמשתנים במערך את האופציות האלה (1- אם זה לא פעיל/לא קשור ללוח , 0 משבצת ריקה , 1 משבצת אם חייל אדום ו 2 משבצת אם חייל שחור).
  + gboard - מערך דו ממדית דינמית של GameButton שמטרתו להיות אחראית לשמירה והצגה גרפית של הלוח.
    - GameButton זה מחלקה אשר משמשת ככפתור. המחלקה משתמשת במשתנים הללו.
      * img – זהו תמונה של הכפתור.
      * weight – זהו המשקל של הכפתור (בשביל המחשב).
      * soldier – זהו מחלקה בשם Soldier שמטרתה לייצג חייל. היא משתמשת במשתנים הללו.
        + img – זה התמונה של החייל.
        + color – זה מספר אשר מיצג את הצבע של החייל.
  + שאר הפונקציות והמשתנים מוסברות בהרחבה בסעיף "הפונקציות/ המחלקות הראשיות בפרויקט".

# תיאור סביבת העבודה ושפת התכנות

שפת התכנות היא: .java

שימוש מינימלי בשפת תגיות: .html

ספריות: util, awt, swing ,file.

סביבת העבודה היא: .visual studio code

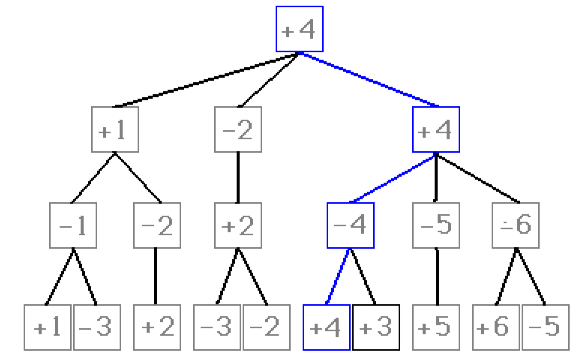
מפרט תכני:

* מעבד: .Intel(R) Core(TM) i5-5300U CPU 2.30GHz
* זיכרון: 8 GB RAM.
* סביבת עבודה: windows 10.

# אלגוריתם ראשי

האלגוריתם הראשי הוא האלגוריתם למציאת המהלך הטוב ביותר, בעזרת האלגוריתם negamax. האלגוריתם מוצא את התור הבא הטוב ביותר בעזרת מציאת כל התורות הבאים האפשריים ומוצא את התוצאה הטובה ביותר. בעזרת האלגוריתם אפשר גם לבדוק את כמה הצעדים הבאים על ידי מציאת את כל הצעדים האפשריים עבור הלוחות שמתקבלים לאחר חישוב כל הצעדים האפשריים בלוח הקודם.

האלגוריתם Negamax מיוצג בעזרת עץ משחק. האלגוריתם סורק את כל המצבים האפשריים, עד לעומק מסוים אותו נבחר. שורש העץ מייצג את הלוח הקיים, לפני ביצוע מהלך כלשהו. ברמה הבאה יופיעו כל מצבי הלוח, עליו בוצע מהלך אחד קדימה, וכך הלאה. כל עלי העץ יקבלו ציון, לפי הפעולה, evaluate שנותנת ציון ללוח. ברמה שבודקת מהלכים של השחקן הממוחשב, נבחר באופציה בעל הציון הגבוה ביותר, המהלך הטוב ביותר עבור המחשב. ורמה שבודקת מהלכים עבור השחקן האנושי, נבחר את האופציה בעל הדירוג הנמוך ביותר, משמע האופציה הכי פחות טובה לשחקן הממוחשב.

כדי להשתמש באלגוריתם זה, נכפול במינוס אחד (1-), את התוצאה שחוזרת ברקורסיה מה- negamax .תמיד נחזיר את הציון בעל הערך הגבוה ביותר, וכך בתור השחקן האנושי נחזיר את התוצאה בעל הערך המוחלט הקטן ביותר.

דוגמא לעץ negamax:

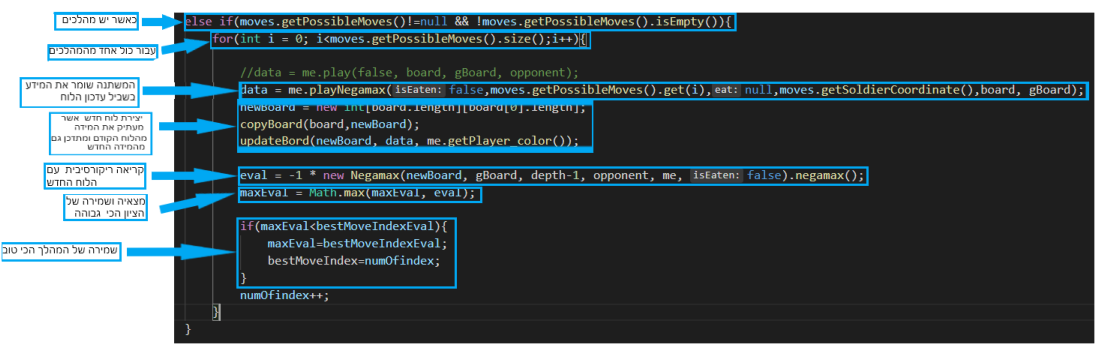
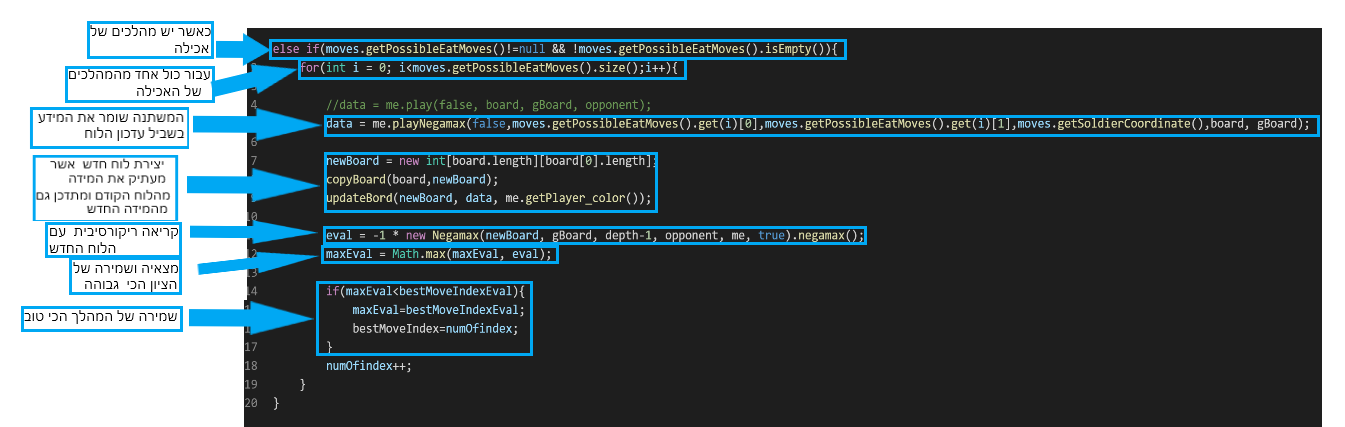
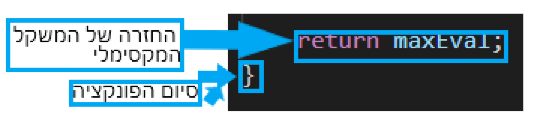
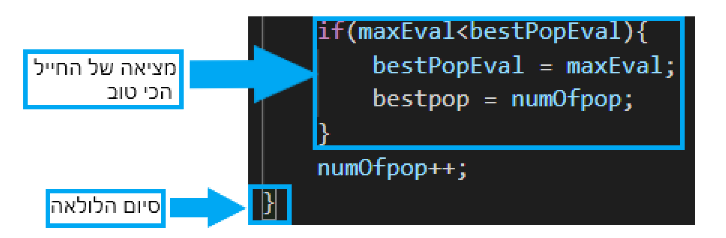
הפונקציה negamax בפרויקט:



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט

התיאור נוצר באופן אוטומטיתמונה שמכילה טקסט, צילום מסך, צג, מסך

התיאור נוצר באופן אוטומטי 

# תיאור ממשקים מחלקות ופונקציות ראשיות בפרויקט

|  |  |  |  |
| --- | --- | --- | --- |
| מחלקה | משתנה/פעולה | הסבר | יעילות |
| Game | int turn | שומר את הטור(1 אדום 2 שחור) | - |
| int gameMode | שומר את סוג המשחק (0 שחקן נגד שחקן 1 שחקן נגד מחשב 2 מחשב נגד מחשב) | - |
| int gameStart | שומר איזה שחקן מתחיל | - |
| int difficultyComputer1  int difficultyComputer2 | שומר את הרמת קושי של המחשבים | - |
| Players playerRed  Players playerBlack | שומר את השחקן האדום והשחור | - |
| Computer computerRed  Computer computerBlack | שומר את המחשב האדום והשחור | - |
| QueahBoard board | הלוח משחק | - |
| String map | שומרת את סוג המפה של המשחק | - |
| public Game() | פעולה בונה שמתחלה את המשחק ומשתמשת בפונקציות אחרות כדי לאתחל ממשתנים | O(1) |
| private static String getFileInfo() | פונקציה שקוראת את הקובץ של החוקים ומחזירה אותו. | O(1) |
| private void menuMap() | פונקציה שמציגה את האופציות של המפות ומאתחלת את | O(1) |
| private void startMap() | פונקציה שמציגה את האופציות של איזה מהשחקנים מתחיל | O(1) |
| private void gameModeMenu() | פונקציה שמציגה את האופציות של איזה סוג משחק. | O(1) |
| private void difficultyComputer1()  private void difficultyComputer2() | פונקציות שמציגות את האופציות של איזה קושי כול מחשב יכול להיות | O(1) |
| private void mapSolid(QueahBoard board) | פונקציה שמקבלת את הלוח ומעדכנת לפי המפה את הלוח. | O(1) |
| QueahBoard | int [][]lBoard | מערך דו מימדי של מספרים שלמים אשר מיצג את הלוח הלוגי | - |
| GameButton [][]gBoard | מערך דו מימדי של כפתורים אשר מיצגת את הלוח הגרפי של המשחק | - |
| public QueahBoard(Game game) | פונקציה בונה שמקבלת את המחלקה game ומאתחלת את הנתונים | O(n)  n=row\*column |
| public void initBoard() | פונקציה מאפסת את הלוחות(הגרפי והלוגי)  ומוסיפה במקום המתאים את החיילים בצורה לוגית וגרפית  וגם מוסיפה משקל. | O(n)  n=row\*column |
| private void addWeight() | הפונקציה מוסיפה את המשקלים ללוח. | O(n)  n=row\*column |
| public void victory(int player) | הפונקציה מקבלת את השחקן ומציגה הודעה שאומרת שהשחקן ניצח | O(1) |
| public void moveSoldier() | הפונקציה מזיזה חייל | O(1) |
| public void removeSoldier(int row, int column) | הפונקציה מקבלת את השורה והעמודה ומסירה חייל מהלוח | O(1) |
| public void addSoldierToBoard() | הפונקציה מוספיה חייל ללוח | O(1) |
| public void actionPerformed(ActionEvent e) | הפונקציה מקבלת ActionEvent ומחכה ללחיצת כפתור כאשר נלחץ הכפתור היא בודרת מי לחץ ובאיזה סוג משחק אנכנו אם אנכנו בשחקן נגד שחקן היא מפעילה את הפונקציה של השחקן אם זה שחקן נגד מחשב היא בודקת טור מי זה אם זה טור השחקן היא מפעילה את הפונקציה של השחקן אם זה טור של המחשב אז היא מזיזה אותו | O(1) |
| private void HumanMove() | הפונקציה אחראית לזה לבדיקת הפעולה של השחקן אם היא חוקית היא משנה אותו בלוח הלוגי והגרפי | O(1) |
| private void ComputerMove(boolean isEaten) | הפונקציה אחראית לזה שהמחשב יזוז גרפית ולוגית והיא מקבלת משתנה בוליאני שאומר אם נאכל חייל של המחשב | O(1) |
| Computer | public Computer(int player\_color, String map,int difficulty) | הפונקציה הבונה מאתחלת מישתנים.  היא מקבלת את הצבע של המחשב את סוג המפה ואת קושי של המחשב | O(1) |
| public int[] play(boolean isEaten,int [][]lBoard,GameButton [][]gBoard, Computer enamy) | הפונקציה מחשבת ומחזירה מידה שאומר אך למחשב רוצה להזיז את החייל.  היא מקבל משתנה בוליאני שאומר אם האחל למחשב חייל הוא מקבל גם את הלוח הלוגי והגרפי וגם הוא מקבל את היריב שלו. | O(n)  n= row\*column |
| private int[] addNewSolid() | הפונקציה מחשבת ומחזירה למחשב את המקום הטוב ביותר לשים שחקן חדש | O(n)  n= row\*column |
| private int[] move() | הפונקציה מחשבת לפי הרמת קושי של המחשב את המהלך שהוא צריך לעשות ומחזיר את המהלך הזה. | O(n)  n= all item in negamax tree |
| private SoldierMoves findBestMove(Stack<SoldierMoves> soldierMovesStack)  private SoldierMoves findBestEat(Stack<SoldierMoves> eatSoldierMovesStack) | הפונקציות מקבלות את המחסנית של האכילה/הזזה ומחזירות את החייל אם הזזות הטובות ביות | O(n)  n= all item in the stack |
| private int indexOfBestMove(List<Coordinate> possibleMoves)  private int indexOfBestEat(List<Coordinate[]> possibleEatMoves) | הפונקציה מחזירה את המיקום ברשימה של האכילה/הזזה של החייל שבוא המשקל הוא הטוב ביותר. | O(n)  n= all item in the list |
| private void findAllPossibleSoldier() | הפונקציה מחפשת את כול החיילים של השחקן שנמצאים על הלוח. | O(n)  n= row\*column |
| private int[] findMostWeightBlock() | הפונקציה מחזירה את המיקום אם המשקל הכי גבוה בשביל לשים חייל חדש. | O(n)  n= row\*column |
| SoldierMoves | public SoldierMoves(int [][]lBoard,GameButton [][]gBoard,Coordinate soldierCoordinate) | הפונקציה בונה ומאפסת את כול המשתנים. | O(1) |
| public void scannMap() | הפונקציה סורקת את הלוח ומכניסה למישתנים את הנתונים שלהם. (המישתנים מתוארים בסעיף "מיבנה נתונים" מס' 5 ) | O(1) |
| public void findPossibleMoves()  public void findPossibleEatMoves()  public void findCoordinatesOfEnemySoldiercanNotEat()  public void findAllySoldier()  public void findMoveNotSafe() | הפונקציות מחפשות לפי המישתנים של הסריקה של המפה את(לפי שם בפונקציה)  ומעכן את הרשימה המתאימה) | O(1) |
| public boolean isSoldierStuck() | בודק אם חייל תקועה ומחזיר את התשובה. | O(1) |
| public boolean isSoldierNotInDanger() | בודק אם חייל לא נימצה בסכנה. ומחזיר את התשובה | O(1) |
| public int weightSoldierMoves() | הפונקציה מחשבת ומחזירה את המישקל של החייל. | O(1) |
| Negamax | public Negamax(int[][] board,GameButton [][]gBoard, int depth,Computer me, Computer opponent, boolean isEaten) | הפונקציה בנוה מקבלת ממשתנים ומעדכנת אותם במחלקה. | O(n)  n=row\*column |
| public boolean gameOver() | הפונקציה מחזירה משתנה בוליאני אם נגמר המשחק. | O(1) |
| private void updateBord(int[][] board,int[] data, int player\_color) | הפונקציה מקבלת את הלוח,מידע,את צבע השחקן ומעדכנת את הלוח שקיבלה. | O(1) |
| private int evaluateBordByPlayer(Computer computer) | הפונקציה מחזירה את המשקל של המחשב שהיא קיבלה. | O(n)  n=the number of item in the stack |
| private int evaluate() | הפונקציה מחזירה את המשקל של הלוח. | O(n)  n=the number of item in the stack |
| public int negamax() | פונקציה רקורסיבית שעובדת לפי(ההסברת בסעיף האלגוריתם הראשי) ומחזירה את המשקל. | O(n)  n=מספר האיברים בעץ |

# התוכנית הראשית

התוכנית הראשית מייצרת משחק חדש. ואחראית לכול האופציות לפני המשחק עצמו.

# מדריך למשתמש