**עבודת גמר**

**התמחות: מערכות הפעלה**

שם הפרויקט: דמקה

**המגיש : דוד הלוי**

**המנחה : איציק גולדברגר**

**שנה''ל תשע"ט**

**אישורים**

**חתימת התלמיד :**

**שם חתימה תאריך**

**מנחה :**

**שם חתימה תאריך**

**הערות מרכז המגמה**

**שם חתימה תאריך**

**הערות הבוחן**

**חתימת הבוחן:**

**שם חתימה תאריך**

**תוכן עניינים**

**הצעת פרויקט**

1. **מבוא**
   1. **מטרה ----------------------------------------------------------------------------------- 4**
   2. **תיאור המערכת ------------------------------------------------------------------------ 4**
   3. **מבנה המסמך -------------------------------------------------------------------------- 4**
   4. **ביבליוגרפיה ---------------------------------------------------------------------------- 4**
2. **מפרטי תוכנה**
   1. **תיאור כללי ----------------------------------------------------------------------------- 5**
   2. **ממשק אדם מכונה MMI ----------------------------------------------------------- 6**
   3. **יכולות פונקציונליות ------------------------------------------------------------------ 11**
   4. **תיאור אלגוריתמים--------------------------------------------------------------------- 12**
   5. **רשימת נתונים -------------------------------------------------------------------------15**
3. **תכנון**
   1. **חלוקה למודולים ----------------------------------------------------------------------- 18**
   2. **פירוט המודולים ----------------------------------------------------------------------- 20**
   3. **מבנה נתונים --------------------------------------------------------------------------- 25**
4. **מדריך למשתמש**
   1. **הוראות התקנה (Setup) ----------------------------------------------------------- 26**
   2. **הפעלה --------------------------------------------------------------------------------- 26**

**נספח 1 Listing ------------------------------------------------------------------------- 27**

**1. מבוא**

* 1. **מטרה**

המשחק שנבחר לפרויקט הוא דמקה. מטרת המשחק הינה חשיבה אסטרטגית לטווח ארוך של המשתמש בנוסף ליכולת ליהנות הן מהאינטראקציה שבמשחק בין שחקן לשחקן והנאה ממשחק נעים ונחמד שכיף להעביר איתו את הזמן.

* 1. **תיאור המערכת**

המשחק דמקה משוחק על לוח של 64 משבצות, ומשוחק על ידי שני שחקנים כאשר לכל שחקן יש אבן בצבע אחר (לבן – שחקן ראשון , שחור– שחקן שני). כל משחק דמקה מתחיל עם אבנים בכל צד מהלוח מסודרים בצורה של אלכסונים על המקומות הריקים הירוקים וכך למשך שלוש שורות מכל צד של הלוח – כל צד – צבע אחר. השחקן שברשותו האבנים הלבנות מתחיל תמיד לשחק ראשון. אתה מוריד מהלוח את אבני היריב על ידי קפיצה באלכסון מעל אבן שלו.

המשחק משוחק רק בצורה אלכסונית בלי קדימה אחורה או לצדדים.

המהלכים החוקיים היחידים האפשריים כוללים קפיצה באלכסון אחד, או שתיים, אם אתה מדלג מעל אבן של יריב, יש גם אפשרות לקפוץ עוד פעם שתיים, אם אחרי קפיצה אחת – יש לך אפשרות לעוד אחד.

אם אין לך שום מהלך חוקי אתה הפסדת במשחק, והשחקן השני ניצח. המשחק נגמר כאשר לאחד השחקנים נגמרים האבנים או כאשר אין לשחקן מסויים תור לעשות, ואז השחקן שברשותו נשאר הכדורים ולאחר אין – הוא המנצח או האחד שיש לו מהלכים, כאשר לשני אין.

* 1. **מבנה המסמך**

הספר כולל ארבעה פרקים:

הפרק הראשון - נותן מבוא למשחק עצמו : מטרת המשחק, תיאור כללי של המשחק, מבנה מסמך ורשימת מקורות שנעזרתי לצורך הכנת הפרויקט.

הפרק השני - מפרט את התוכנה. הוא נותן תיאור מפורט יותר של המשחק (כולל מצבי משחק שונים ותיאור נקודות יתרון ותורפה של שחקנים) מממשק האדם מכונה, יכולות של המערכת, תיאור האלגוריתם ורשימת הנתונים.

הפרק השלישי - מפרט את התכנון של המערכת. הוא מתאר את המודולים השונים ואת מבנה נתונים.

הפרק הרביעי - הוא הוראות ההפעלה. הוא מפרט את הוראות ההתקנה ואת הוראות ההפעלה.

* 1. **רשימת מראה – מקום**

1. C++ סדנת לימוד בעריכת זהר עמיהוד. הוצאת הוד-עמי, 2001.
2. התוכנהMICROSOFT VISUAL STODIO, קובץ העזרה

**2. מפרטי תוכנה**

**2.1. תאור כללי**

יישום זה בא לדמות את המשחק המרתק דמקה תוך שימוש בסביבת עבודה C++ visual studio שבאמצעותה פותח המשחק.

המשחק משוחק על לוח סטנדרטי 8X8 ,כאשר במצב ההתחלתי של המשחק מוצבות אבנים בצורה אלכסונית למשך שלוש שורות מתחלת הקצה העליון והתחתון של הלוח, כל צד צבע אחר, והן מונחות על המשבצות השחורות. (כל המשחק מנוהל רק על המשבצות השחורות– המשחק כולו באלכסון.)

**2.1.1 סוגי מהלכים**

המשחק משוחק רק באלכסון, ולכן אפשר או לקפוץ ריבוע אחד באלכסון רק לכיוון הקדמי. (יש אפשרות להיות מלך, שהיא אבן שהגיע כבר לקצה השני של הלוח – ואז היא יכולה לזוז גם אחורה וקדימה), , ואז – גם הורדתי את האבן שקפצת מעליה של היריב, וגם אם יש לך עוד פעם לקפוץ מעל אבן של יריב – יש לך עוד תור.

**2.1.1.1 הצבת אבן:**

כדי להציב אבן חדשה על הלוח, יש ללחוץ על אחד מהאבנים מהצבע שלך, וכך תוכל לראות את המהלכים האפשריים שיש לך אם אותה אבן, יש אפשרות לשנות את הבחירה, על ידי לחיצה על אבן אחרת מהצבע שלך או על קובייה ריקה, אחרי שהחלטת איזה אבן אתה רוצה להזיז, אתה לוחץ על אחד מהריבועים שבצבע זהב– שמסמל – לאן אפשר לזוז עם אותה האבן.

להלן דוגמה של מצב זה :



בדוגמה זו השחקן השחור יכול להציב אבן רק בנקודה המסומנת בעיגול (- אם יש אפשרות לשחקן לקפוץ מעל היריב – הוא חייב לעשות זאת.)

**2.1.2. חוקים חשובים:**

1. ניתן להציב אבן רק במשבצות שבצבע זהב המסומנים על גבי הלוח.

2. כשאין לך שום מהלך חוקי – הפסדת במשחק.

**2.2 ממשק אדם מכונה (MMI)**

**2.2.1 לוח כניסה**



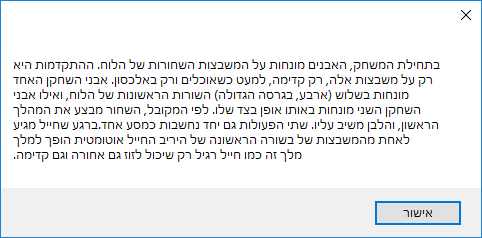
זהו לוח הכניסה למשחק. ישנם שלושה אפשרויות בלוח הכניסה:

1.משחק חדש - בכדי לעשות זאת יש ללחוץ על הלחצן  ***משחק חדש***

2.המשך משחק- בכדי להמשיך את המשחק האחרון ששוחק בלי שינויים יש ללחוץ ***המשך***

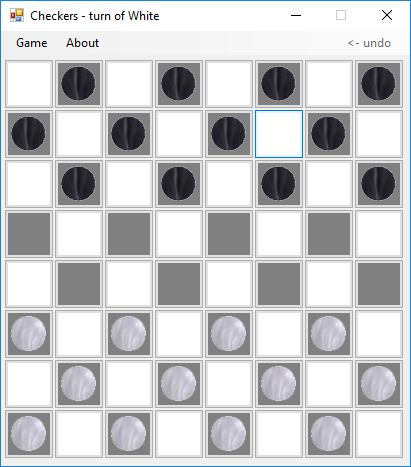
3.חוקים של המשחק- בכדי לעשות זאת יש ללחוץ על הלחצן ***חוקים של המשחק***

**2.2.2 מסך עזרה למשתמש**



מסך זה הוא מסך עזרה לשחקנים מתחילים בו הם יכולים ללמוד את החוקים של המשחק ואיך משחקים בו

**2.2.3 לוח משחק**

****

אפשרות למהלך חוזר

משבצות שעליהם השחקנים זזים

תורו של מי:

אודות המשחק:

אפשרויות:

משחק חדש

יציאה

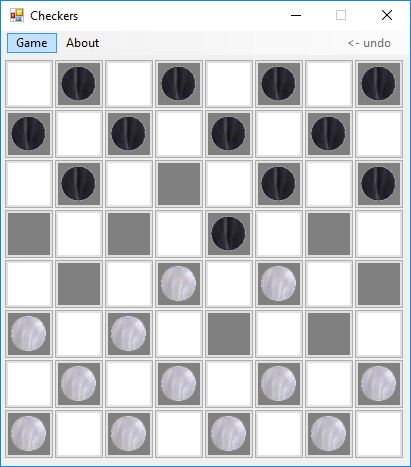
תמונה מתארת לוח 8X8 של משחק דמקה במצב התחלתי. מהלך ראשון הוא של משתמש המשחק בצבע לבן. במצב זה אבנים בכל צד מהלוח מסודרים בצורה של אלכסונים על המקומות הריקים השחורים וכך למשך שלוש שורות מכל צד של הלוח – כל צד – צבע אחר., בנוסף ישנו כפתור שנותן לך את האפשרות לבטל מהלך אחרון שלך ולשנות אותו

**2.2.4 תפריט משחק**

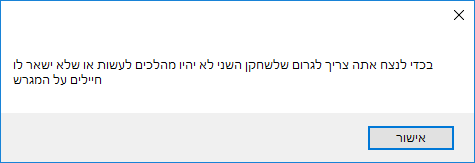
2.2.4.1 בחירת משחק חדש ויציאה ממשחק

\* אפשרות יציאה מהמשחק

\*אפשרות ליצירת משחק חדש



2.2.4.2 קבלת עזרה והסברים

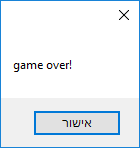
כדי לקבל הסבר על המשחק, מהי מטרתו, ואיך מנצחים. -e

**ABOUT>HOW TO PLAY**

**2.2.5 הודעות מערכת**

\* כאשר שחקן ינסה להציב אבן במשבצת שהיא הינה מסומנת בעיגול קטן, או לבחור אבן שאינה שלו יתבטל המשבצות הזהובות.

**\*** כשנגמר המשחק - כשהלוח התמלא או כשלאחד השחקנים לא נותרו מהלכים חוקים לאף אחד מהאבנים שלו, תופיע הודעת סוף המשחק:



* 1. **יכולות פונקציונליות**

1. המערכת מספקת אפשרות של יציאה מהמשחק בכל שלב במשחק.

2. המערכת מספקת עזרה למשתמש בנוגע לכללי ולדרכי המשחק.

3. המערכת מספקת אפשרות של התחלת משחק חדש .

4. בסוף כל משחק המערכת מכריזה על סוף משחק.

5. המערכת שומרת את המצב של המשחק אחרי יציאה ממנו.

6. המערכת נותנת אפשרות לחזור למצב האחרון שנעשה לפני היציאה מהמשחק .

7. המערכת מאפשרת לחזור על מהלך מסוים אם הוא אינו חוקי. 8. המערכת מספקת הודעות על מהלכים מסוימים שאינם חוקיים.

9. המערכת מודיעה של מי התור בכל שלב ושלב במשחק.

10. המערכת מאפשרת לחזור ממהלך מסוים גם אם הוא חוקי.

**2.4 תיאור אלגוריתם**

**\*** ישנו אלגוריתם עיקרי במשחק שמטפל במציאת כל המהלכים האפשריים עבור השחקן.

**2.4.1 אלגוריתם מציאת כל המהלכים**

**מטרת האלגוריתם:**

מטרתו של אלגוריתם זה היא למצוא עבור כל אבן, את המהלכים האפשריים לה.

**תיאור האלגוריתם:**

האלגוריתם מחפש באלכסונים המתאימים לאותו צבע אבן, (אם זה מלך– המצב שונה – ויש לו אפשרות לכל כיוון באלכסון.) ומשנה את התמונה ואת הסוג של המקום שם לתמונה וסוג של מהלך, שמסומן על ידי ריבוע זהב.

**2.4.2 אלגוריתם עדכן לוח**

**מטרת האלגוריתם:**

מטרתו של האלגוריתם היא לעדכן את מצב הלוח לפי המיקום שנלחץ על ידי השחקן, והאבן איתה הוא זז – את המיקום שלה שמרנו במשתנים של המחלקה.

**תיאור האלגוריתם:**

האלגוריתם קודם כל מוחק את הריבועים של מהלך, (ריבוע זהב) אחר כך הוא משנה את המיקום והתמונה של האבן שעברה מיקום, ואם הוא קפץ מעל אבן של היריב – היא מוחקת את האבן שנאכלה מהלוח.

**2.4.3 אלגוריתם הצב צעד**

**מטרת האלגוריתם:**

מטרת האלגוריתם היא להציב על הלוח המשחק תמונות של חלקי המשחק לפי הנתונים שהתקבלו.

**תיאור האלגוריתם:**

האלגוריתם מקבל מיקום של משבצת וכן את הצבע הרצוי ופועל כך:

|  |  |
| --- | --- |
| צבע משבצת - color | תיאור |
| move | האלגוריתם מזיז את האבן מהקום שנמצא למקום שנבחר מהשבצות בזהב |
| Empty | האלגוריתם מציב במשבצת הרצויה תמונה של משבצת ריקה |
| ellClickedWithrightMouse | האלגוריתם יסמן את המשבצות שאליהם יכולה האבן שבחרתה לזוז בצבע זהב ולחיצה על אחת מהמקומות שסומנו תגרום לאבן לזוז לשם |
| ellClickedWithLeftMouse | על ידי לחיצה במקש השמאלי בעכבר האלגוריתם יחפש אכילה נוספת ואם יש אז הוא יסמן אותו בצבע זהב . |
| White | האלגוריתם מציב במשבצת הרצויה תמונה של אבן משחק לבנה בלי כלום בפנים מסמל אבן משחק רגילה. |
| black | האלגוריתם מציב במשבצת הרצויה תמונה של אבן משחק שחורה בלי כלום בפנים מסמל אבן משחק רגילה. |
| 'BLACK' | האלגוריתם מציב במשבצת הרצויה תמונה של אבן משחק שחורה עם כתר אדום– זה מסמל אבן שבמצב מלך. וגם היא מוקפת בפס תכלת– מסמל בחירה. |
| ' WHITE ' | האלגוריתם מציב במשבצת הרצויה תמונה של אבן משחק לבנה עם כתר אדום בפנים – זה מסמל אבן שבמצב מלך. וגם היא מוקפת בפס כחול– מסמל בחירה. |

**\*** לאחר שהשחקן בחר את האבן איתה הוא רוצה לשחק, מתרחשת בדיקה של המהלכים החוקיים של האבן הזאת על פני הלוח, והוא יכול לבחור את המהלך, או לשנות את הבחירה של האבן לאבן אחרת.

**\*** לאחר כל צעד במשחק, מתבצעת בדיקה עם יש ניצחון על ידי האלגוריתם בדיקת ניצחון.

**2.4.5 אלגוריתם בדיקת ניצחון**

**מטרת האלגוריתם:**

לבדוק האם כתוצאה מהמהלך האחרון אחד השחקנים ניצח.

**תיאור האלגוריתם:**

האלגוריתם בודק שני מצבים:

1. אם לאחד השחקנים נגמרו האבנים – במצב זה אם לשחקן הראשון נגמרו האבנים המנצח הוא השחקן השני. ולהיפך, אם לשחקן השני נגמרו האבנים המנצח הוא השחקן הראשון.

2. אם לאחד השחקנים אין מהלכים חוקיים – במצב זה אם לשחקן הראשון אין מהלכים חוקיים כלל המנצח הוא השחקן השני, ולהיפך, אם לשחקן השני אין מהלכיים חוקיים המנצח הוא השחקן הראשון.

בכל אחד מהמצבים תופיע על המרקע הודעת הפסד של סוף המשחק.

**2.4.8 ביטול פעולה:**

**מטרת האלגוריתם:**

לבטל את הפעולה האחרונה שנעשתה .

**תיאור האלגוריתם:**

האלגוריתם שומר את הפעולה האחרונה בפורם, שנעשתה לפני הפעולה הנוכחית וברגע שהשחקן לוחץ על כפתור undo הפעולה מחזירה את הלוח למצב של הפעולה האחרונה .

**2.5 רשימת נתונים**

**2.5.1 רשימת מניע Ball\_Type**

|  |  |
| --- | --- |
| סוג | ערך |
| EMPTY | 0 |
| BLACK | 1 |
| WHITE | 2 |
| MOVE | 3 |
| WMOVE | 4 |
| RMOVE | 5 |
| WQ | 6 |
| RQ | 7 |
| ROQ | 8 |
| WOQ | 9 |

סוגים שונים לאכלוס משבצות בלוח.

**2.5.2 מבנה logic**

1. getsolider– משתנה המכיל את מיקום התא שבו הוצבה אבן.

2. iswhiteturn– זוכר של מי התור

3. changeTurnWithoutMovingAnything – משנה את התור בלי להזיז כלום.

4. currentPlayerLost – האם השחקן שעכשיו תורו הפסיד.

5.Inbord-בודקת האם הוא נמצא בגבולות הלוח

Eat - רשימה שנותנת את האפשרות אכילה

nIspiecewhite -משתנה בוליאני שבודק האם השחקן הוא לבן אם לא הוא שחור.

Move- רשימה שמכילה את המהלכים של התזוזה

**2.5.3 מחלקת mainform**

1. howToPlayToolStripMenuItem – משתנה שמציג על המסך את חוקי המסך.

2. setPicturesOnButtons – מתאים את התמונות לצ’ארים של הלוגיקה.

3 setButtonsToBlackAndWhite – משתנה שדואג לרקע של הכפתורים שיהיה שחור לבן בלי שום זהב.

ContinueFromSavedGameButton .4 שומר את המשחק גם אחרי יציאה ממנו.

OnButtons .5 setPictures – משתנה שדואג שכל החיילים נמצאים במקום שלהם.

**2.5.4 מחלקת cell.cs**

1. cell – משתנה שמוסיף את הכפתור לרשימת הדברים שמצויירים על החלון.

2. resetColor – משתנה שהופף את צבע התא לשחור או לבן בהתאם למיקום שלו.

3. setimage – משתנה שמדביק את התמונות של החיילים במקומות שלהם

**2.5.5 מחלקת files**

1. SaveGame– שומר את המשחק

**2.5.11 רשימת קבועים**

|  |  |  |
| --- | --- | --- |
| שם קבוע | ערך קבוע | מטרה |
| ROWS | 8 | מציין את מספר השורות בלוח המשחק. |
| COLS | 8 | מציין את מספר הטורים בלוח המשחק. |
| BALLSIZE | 47 | מציין את גודל התמונה של הכדור. |

**3. תכנון**

**3.1 חלוקה למודלים**

|  |  |  |
| --- | --- | --- |
|  | Logic | מממש את החלון הראשי. אחראי על קבלת הודעות ממערכת ההפעלה כגון:   * כניסה ללוח המשחק. * כניסה ללוח העזרה. * יציאה מהמשחק.   מממש את החלון הראשי. אחראי על קבלת הודעות ממערכת ההפעלה כגון:   * כניסה ללוח המשחק. * כניסה ללוח העזרה. * יציאה מהמשחק. |
|  | Cell | אחראי על כל ההודעות של שהמשחק מוציא על פעולות  מממש את החלון העזרה. מטרתו לספק מידע על המשחק למשתמש וחוקים. |
|  | Maineform | אחראי על פעולת המשחק של המחשב, מחלקת משחק אחראי על פעולות כגון:   * בדיקה של ניצחון בכל שלב במשחק. * הפעלת פונקציה מהלך של שחקן אנושי. * בדיקת חוקיות של צעדים. |

**3.1.2 עץ ראשי של מודולים**

**3.2 פירוט המודולים**

**3.2.1 מודול logic**

|  |  |  |
| --- | --- | --- |
|  | **שם פונקציה** | **תפקיד כללי** |
| 1. | CGame | בנאי ברירית מחדל - מאתחל חברי המחלקה ומקצה זיכרון עבור לוח המשחק. |
| 2. | ~CGame | מפרק - משחרר זיכרון מלוח המשחק. |
| 3. | CheckWin | פונקציה שבודקת אם יש ניצחון במשחק. |
| 4. | Press | פונקציה שמופעלת כתוצאה מלחיצת השחקן על לוח המשחק. פונקציה זו נותנת הפנייה לשני הסוגים השונים של המשחקים: שחקן\שחקן ושחקן\שחקן ברשת. |
| 5. | TwoPlayer | פונקציה שמתפלת במשחק של שחקן נגד שחקן על אותו מחשב. |
| 6. | NetTwoPlayer | פונקציה שמטפלת במשחק של שחקן נגד שחקן ברשת. |
| 7. | CheckStep | פונקציה המחזירה אמת אם המהלך תקין – אחרת שקר. |
| 8. | InitBoard | פונקציה שמאתחלת את המסך למצב ההתחלתי. |
| 9. | SetBall | פונקציה שמציבה על כדור במטריצה ועל הלוח לפי המיקום והצבע שהיא מקבלת. |
| 10. | FindAllMoves2 | פונקציה שמוצאת את כל המהלכים האפשריים לשחקן לקפיצה מעל אבן של יריב. |
| 11. | CopyBoard | פונקציה שמעתיקה את מצב הלוח הנוכחי למטריצת עזר. |
| 12. | 2CopyBoard | פונקציהשמעתיקה את המטריצת עזר חזרה למצב הלוח הנוכחי. |
| 13.. | UpdateBoard | פונקציה שמוחקת מלוח המשחק מהלכים אפשריים שלא נבחרו על ידי השחקן בתור הנוכחי, ועדכנת את מיקום האבן אחרי קפיצה, ומחיקה לפי הצורך של אבנים שקפצו מעליהם. |
| 14. | FindAllMoves | פונקציה שמוצאת את כל המהלכים האפשריים לשחקן. |
| 15. | EraseMoves | פונקציה שמוחקת את המהלכים האפשריים שלא השתמשו בהם, או שנוצרו כתוצאה מחישוב. |

**3.2.2 מודול logic**

|  |  |  |
| --- | --- | --- |
|  | **שם פונקציה** | **תפקיד כללי** |
| 1. | Bord | מערך צ'ארים שעליו מבצעים את החישובים. |
| 2. | isWhiteturn | זוכר של מי התור. |
| 3. | currentPlayerLost | אנחנו שואלים אם השחקן שתורו הפסיד כי אם בתור הקודם השחקן השני ביצע מהלך ניצחון, אז התור הוחלף וכעת תורנו, אך הפסדנו. |
| 4. | Move | פעולה שמזיזה חייל ממקום למקום. |
| 5. | GetMoves | כל המקומות שאפשר להגיע אליהם ממשבצת כלשהי - כולל אכילות וכולל הליכות. |
| 6. | manuallySwitchTurns | החלפת תורות באופן ידני, נועד לאכילה הכפולה |
| 7. | regularJumps | קפיצות רגילות ללא אכילה. |
| 8. | Inbord | בודק שזה בגבולות הלוח-המספרים לא חורגים מאפס לשבע. |

**Logic:**

מטרת השגרה: לבצע את הפעולה שנשלחה על ידי הלקוח

ארגומנטיים:

\*Sender – אובייקט

\*Socket – חיבור

תיאור השגרה:

1. קבל מידע מחיבור

2. פעל לפי סוג מידע שליחה

2.1 מקרה 0

2.1.1 התנתקות()

2.1.2 שקר ->פעילות->חיבורשרת

2.1.3 צא

2.2 מקרה 1

2.2.1 מחרוזת שליחה -> שם שחקן יריב

2.2.2 תוויות\_תור -> תוויות1 -> כתובית

2.2.3 " תור" +-> תוויות\_תור

2.2.4 תוויות\_תור ->כתובית -> תוויות3

2.2.5 צא

2.3 מקרה 2

2.3.1 קבל מידע מחיבור אודות תור

2.3.2 תור ? "שחור" : "לבן" -> כתובית -> תוויות3

2.3.3 צא

2.4 מקרה 3

2.4.1 הקצאת זיכרון לטיפוס נתונים חדש מסוג CBall

2.4.2 מידע מהחיבור על מיקום הטור של האבן -> כדור.שורה

2.4.3 מידע מחיבור על מיקום השורה של האבן -> כדור.טור

2.4.4 מידע מחיבור על סוג האבן -> כדור.סוג

2.4.5 משחק של שני שחקנים ברשת (כדור.שורה,כדור.טור) -> המשחק שלי

2.4.6 צא

**Maineform:**

מטרת השגרה: הפונקציה אחראית לטיפול בלחיצה של העכבר על איזה שהיא משבצת בלוח המשחק, מציאת התא שהאבן נמצאת בו, הפנייה לסוג המשחק שנבחר, שליחת נתוני הצבת האבן לשחקן היריב ברשת.

ארגומנטיים:

Sender\* - אובייקט

Button – משתנה המייצג איזה לחצן של העכבר נלחץ שהפעיל את הפעולה.

Shift – משתנה שמייצג את המצב של העכבר כאשר התבצע הפעולה.

X – היקום האופקי של מקום הלחיצה.

Y – המיקום האנכי של מקום הלחיצה.

תיאור השגרה:

1. אם (המשחק\_שלי) אזי

1.1 אם (לחיצה(כדור\_זמני)->משחק\_שלי && נראה ->רשת\_אייקון)

1.1.1 קבלת מיקום השורה של המשבצת שנלחצה מהחיבור -> כדור\_זמני.שורה

1.1.2 קבלת מיקום הטור של המשבצת שנלחצה מהחיבור -> כדור\_זמני.טור

1.1.3 קבלת סוג המשבצת שנלחצה מהחיבור -> כדור\_זמני.סוג\_כדור

1.1.4 3 -> הגדרת סוג המידע שנשלח לעמית ברשת

1.1.5 אם (לוח\_משחק -> האם\_שרת) אזי

1.1.5.1 שליחת סוג המידע שנשמר ללקוח

1.1.6 אחרת

1.1.6.1 שליחת סוג המידע שנשמר לשרת

**3.2.3 מודול cell**

|  |  |  |
| --- | --- | --- |
| 1. | Cell | תא גרפי שיכול להחזיק תמונה ולדעת כשהוא נלחץ. |
| 2. | setImage | התמונה המתאימה לפי הצ'אר. |
| . |  |  |

**3.2.4 מודול maineform**

|  |  |  |
| --- | --- | --- |
| 1. | InitializeComponent | מצייר את החלון עצמו - מה שיש בדיזיינר - זה קוד שכבר היה כתוב פה. |
| 2. | draw8x8Buttons | יוצר שישים וארבעה כפתורים. |
| 3. | setButtonsToBlackAndWhite | מסדר את הרקע שלהם לשחור לבן שחור לבן. |
| 4. | setPicturesOnButtons | הפעולה שמה את התמונות של החיילים. |
| 5. | draw8x8Buttons | בונה את הלוח בתחילתו של המשחק. |
| 6. | setButtonsToBlackAndWhite | מוחק את הסימוני זהב מהקובייה. |
| 7. | setPicturesOnButtons | מתאים את התמונות שעל הכפתורים לצ'ארים שיש בלוח של הלוגיקה - וגם מעדכן את כותרת החלון שנדע תור מי. |
| 8. | logicToUndoTo | מאפס את הכל לפני תחילת משחק. |
| 9. | logicBeforeDoubleEatingHasBegu | בודק שאין חייל באמצע אכילה כפולה. |

**3.3 מבנה נתונים**

**3.3.1 רשימת לאיכלוס מהלכים אפשריים**

כל איבר ברשימה הוא מבנה שבו ישנם 4 משתנים : Row, Col, Grade, Next. בחרתי ברשימה כי מספר המהלכים האפשריים בכל תור אינו קבוע.

**3.3.2 מטריצה מסוג \*CBall**

מטריצה במידות הלוח, שמחזיקה בכל תא מצביע לאובייקט CBall שהוא תמונה של כדור, מיקום וסוג (Ball\_Type).

**3.3.3 מטריצה מסוג Ball\_Type**

מטריצה בגודל הלוח, שמחזיקה בכל תא Ball\_Type שמייצג את התוכן האמיתי של המשבצת (שחור, לבן, צעד, ריק) על לוח המשחק המופיע על המרקע.

4. **מדריך למשתמש**

**4.1 הוראות התקנה**

* צור ספרייה חדשה בדיסק קשיח בשם דמקה .
* הכנס את דיסקט לכונן התקליטונים.
* פתח את קובץ שבדיסק והעתק את הקבצים שבדיסק לספרייה דמקה שיצרת בדיסק קשיח.

**4.2 הוראות הפעלה**

דרישות במערכת להפעלת המשחק הם:

* מעבד מסוג 133 MHz Pentium ומעלה.
* זיכרון פנימי 16 MB RAM ומעלה.
* כרטיס מסך MB 4 ומעלה.
* מסך SVGA.
* מערכת הפעלהXP \Windows 98\2000\ME

לצורך הפעלת המשחק הכנס לספרייה דמקה שיצרת, והפעל את הקובץ.

**Logic:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace damka\_game

{

[Serializable]

class logic

{

private char[,] bord; // מערך צ'ארים שעליו מבצעים את החישובים

private bool isWhiteturn; // זוכר של מי התור

public char getSoldier(int i, int j)

{

return bord[i, j];

}

public bool IsWhiteTurn()

{

return isWhiteturn;

}

public void changeTurnWithoutMovingAnything()

{

isWhiteturn = !isWhiteturn;

}

/// <summary>

/// האם השחקן שתורו - הפסיד

/// </summary>

public bool currentPlayerLost()

{

// אנחנו שואלים אם השחקן שתורו הפסיד כי

// אם בתור הקודם השחקן השני ביצע מהלך ניצחון, אז התור הוחלף וכעת תורנו, אך הפסדנו

if (isWhiteturn)

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

if (bord[i, j] == 'w' || bord[i, j] == 'W')

return false;

}

else

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

if (bord[i, j] == 'b' || bord[i, j] == 'B')

return false;

}

return true;

}

public logic clone()

{

logic result = new logic();

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

result.bord[i, j] = this.bord[i, j];

result.isWhiteturn = this.isWhiteturn;

return result;

}

public logic()

{

this.bord = new char[8, 8];

this.isWhiteturn = true;

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

bord[i, j] = ' ';

}

}

// עכשיו נשים מיקומים התחלתיים

for (int i = 0; i < 3; i++)

{

for (int j = 1 - i % 2; j < 8; j += 2)

{

bord[i, j] = 'b';

}

}

for (int i = 5; i < 8; i++)

{

for (int j = 1 - i % 2; j < 8; j += 2)

{

bord[i, j] = 'w';

}

}

}

public void move(Tuple<int, int> from, Tuple<int, int> to)

{

bord[to.Item1, to.Item2] = bord[from.Item1, from.Item2];

bord[from.Item1, from.Item2] = ' '; // שתי השורות האלה מזיזות

if (from.Item1 - to.Item1 == 2 || from.Item1 - to.Item1 == -2) // אם הוא הלך צעד כפול

{

int i = (from.Item1 + to.Item1) / 2; // הממוצע בין הפרום לטו

int j = (from.Item2 + to.Item2) / 2;

bord[i, j] = ' '; // מעלימים את החייל האכול

}

if (isWhiteturn && to.Item1 == 0) // אם הוא הגיע לשורה האחרונה והוא לבן

bord[to.Item1, to.Item2] = 'W'; // להפוך למלך

else if (!isWhiteturn && to.Item1 == 7)

bord[to.Item1, to.Item2] = 'B';

isWhiteturn = !isWhiteturn; // להעביר תור

}

/// <summary>

/// כל המקומות שאפשר להגיע אליהם ממשבצת כלשהי - כולל אכילות וכולל הליכות

/// </summary>

public List<Tuple<int, int>> GetMoves(int i, int j)

{

List<Tuple<int, int>> moves = new List<Tuple<int, int>>();

// char.ToLower(אות) ==

// ממיר לאות קטנה - רווח נשאר רווח

if (isWhiteturn && char.ToLower(bord[i, j]) != 'w' || !isWhiteturn && char.ToLower(bord[i, j]) != 'b')

return moves; // אם לא תורו, נחזיר את הרשימה כשהיא עדיין ריקה, כדי לציין שאין לו לאן ללכת

foreach (Tuple<int, int> shortJump in eats(i, j)) // אכילות

moves.Add(shortJump);

foreach (Tuple<int, int> shortJump in regularJumps(i, j)) // צעדים רגילים

moves.Add(shortJump);

return moves;

}

/// <summary>

/// האם הם בצבע הנגדי

/// </summary>

private bool IsOppositeColor(char piece1, char piece2)

{

piece1 = char.ToLower(piece1); // הפכנו אותה לאות קטנה

piece2 = char.ToLower(piece2);

return piece1 != piece2 && piece1 != ' ' && piece2 != ' ';

}

/// <summary>

/// להחליף תורות באופן ידני, נועד לאכילה הכפולה

/// </summary>

public void manuallySwitchTurns()

{

isWhiteturn = !isWhiteturn;

}

private bool isPieceWhite(char piece)

{

return piece == 'w' || piece == 'W';

}

/// <summary>

/// אכילות

/// </summary>

public List<Tuple<int, int>> eats(int i, int j) // אכילות

{

List<Tuple<int, int>> list = new List<Tuple<int, int>>(); // רשימה של זוגות של מספרים

if (bord[i, j] == 'B' || bord[i, j] == 'W' || bord[i, j] == 'w')

{

if (inbord(i - 2, j + 2) && IsOppositeColor(bord[i, j], bord[i - 1, j + 1]) && bord[i - 2, j + 2] == ' ')

list.Add(new Tuple<int, int>(i - 2, j + 2));

// אם אין חריגה מהלוח, ויש כלי נגדי צמוד אליך, ויש משבצת ריקה מעבר

if (inbord(i - 2, j - 2) && IsOppositeColor(bord[i, j], bord[i - 1, j - 1]) && bord[i - 2, j - 2] == ' ')

list.Add(new Tuple<int, int>(i - 2, j - 2));

}

if (bord[i, j] == 'B' || bord[i, j] == 'W' || bord[i, j] == 'b')

{

if (inbord(i + 2, j + 2) && IsOppositeColor(bord[i, j], bord[i + 1, j + 1]) && bord[i + 2, j + 2] == ' ')

list.Add(new Tuple<int, int>(i + 2, j + 2));

// אם אין חריגה מהלוח, ויש כלי נגדי צמוד אליך, ויש משבצת ריקה מעבר

if (inbord(i + 2, j - 2) && IsOppositeColor(bord[i, j], bord[i + 1, j - 1]) && bord[i + 2, j - 2] == ' ')

list.Add(new Tuple<int, int>(i + 2, j - 2));

}

return list;

}

/// <summary>

/// קפיצות שהם לא אכילה

/// </summary>

private List<Tuple<int, int>> regularJumps(int i, int j)

{

var rezults = new List<Tuple<int, int>>();

if (bord[i, j] == 'W' || char.ToLower(bord[i, j]) == 'b')

{

if (inbord(i + 1, j + 1) && bord[i + 1, j + 1] == ' ') rezults.Add(new Tuple<int, int>(i + 1, j + 1));

if (inbord(i + 1, j - 1) && bord[i + 1, j - 1] == ' ') rezults.Add(new Tuple<int, int>(i + 1, j - 1));

}

if (bord[i, j] == 'B' || char.ToLower(bord[i, j]) == 'w')

{

if (inbord(i - 1, j + 1) && bord[i + -1, j + 1] == ' ') rezults.Add(new Tuple<int, int>(i + -1, j + 1));

if (inbord(i - 1, j - 1) && bord[i + -1, j - 1] == ' ') rezults.Add(new Tuple<int, int>(i + -1, j - 1));

}

return rezults;

}

/// <summary>

/// נמצא בגבולות הלוח - המספרים לא חורגים מאפס ושבע

/// </summary>

private bool inbord(int i, int j)

{

return (i <= 7 && j <= 7 && i >= 0 && j >= 0);

}

}

**Class cell.cs:**

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace damka\_game

{

/// <summary>

/// תא גרפי שיכול להחזיק תמונה ולדעת כשהוא נלחץ

/// </summary>

class Cell : Button // לקבל את כל התכונות של באטן

{

readonly int i, j; // לקריאה בלבד - תכונה שלא יכולה לשנות ערך מהרגע שהיא הוגדרה בפעולה הבונה

Mainform mainform;

public Cell(int i, int j ,Mainform mainform)

{

this.i = i; this.j = j; this.mainform = mainform;

mainform.Controls.Add(this); // להוסיף את הכפתור לרשימת הדברים שמצויירים על החלון

this.Size = new Size(50, 50);

this.Location = new Point(3 + 50 \* j, 28 + 50 \* i);

MouseDown += new MouseEventHandler(whenClicked);

}

private void whenClicked(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

mainform.cellClickedWithLeftMouse(i, j);

else

mainform.cellClickedWithRightMouse(i, j);

}

/// <summary>

/// התמונה המתאימה לפי הצ'אר

/// </summary>

public void setImage(char c)

{

if (c == 'w')

Image = Properties.Resources.w\_regular;

else if (c == 'B')

Image = Properties.Resources.b\_king;

else if (c == 'W')

Image = Properties.Resources.w\_king;

else if (c == 'b')

Image = Properties.Resources.b\_regular;

else

Image = null; // שלא תהיה עלינו תמונה

}

/// <summary>

/// הופגת את צבע התא לשחור או לבן - בהתאם למיקום שלו

/// </summary>

public void resetColor()

{

if ((i + j) % 2 == 0)

this.BackColor = Color.White;

else

this.BackColor = Color.Gray;

}

}

}

**Mainform:**

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace damka\_game

{

public partial class Mainform : Form

{

private Cell[,] cells = new Cell[8, 8];

logic logic;

logic logicToUndoTo; // בשביל אנדו

logic logicBeforeDoubleEatingHasBegun = null;

public Mainform()

{

InitializeComponent(); // מצייר את החלון עצמו - מה שיש בדיזיינר - זה קוד שכבר היה כתוב פה

draw8x8Buttons(); // יוצר 64 כפתורים

setButtonsToBlackAndWhite(); // מסדר את הרקע שלהם לשחור לבן שחור לבן

try

{

logic = Files.LoadGame();

button1.Enabled = true;

setPicturesOnButtons(); // כבר שמים את התמונות של החיילים

}

catch

{

button1.Enabled = false;

}

this.Text = "Checkers"; // כותרת של החלון

}

protected override void OnFormClosing(FormClosingEventArgs e)

{

if (logicBeforeDoubleEatingHasBegun != null)

{

MessageBox.Show("סיים את התור בבקשה");

e.Cancel = true;

}

try

{

Files.SaveGame(logic);

}

catch

{

MessageBox.Show("your game could not be saved", "oops");

}

}

/// <summary>

/// בונה את הלוח בתחילת המשחק

/// </summary>

private void draw8x8Buttons()

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

cells[i, j] = new Cell(i, j, this);

// this == אנחנו בתור חלון. הכפתור רוצה לדעת על מי הוא יושב? אז עלינו הוא יושב

this.Size = new Size(0, 0); // מתאים את הגודל של

this.AutoSize = true; // של החלון לגודל הלוח

}

private Tuple<int, int> from = null; // המקום שהוא לחץ עליו כדי להזיז ממנו חייל

public void cellClickedWithLeftMouse(int i, int j)

{

if (logicBeforeDoubleEatingHasBegun == null)

{

if (from == null) // שואל אם הוא כבר החליט מאיפה ללכת

{

userChoseFrom(i, j); // נסמן זהובים וכו

}

else // אם אנחנו נכנסים לאלס זה אומר שהוא כבר בחר קודם מאיפה ועכשיו הוא לחץ לאן

{

userChoseTo(i, j); // נצבע הזזה או דברים בסגנון

}

}

else // אם אתה באמצע אכילה כפולה

{

// נסיים את האכילה הכפולה

logicToUndoTo = logicBeforeDoubleEatingHasBegun;

undoToolStripMenuItem.Enabled = true;

logicBeforeDoubleEatingHasBegun = null;

logic.changeTurnWithoutMovingAnything();

setButtonsToBlackAndWhite(); // מוחק את סימוני הזהב

setPicturesOnButtons(); // כי הפעולה הזו מעדכנת בכותרת החלון תור מי עכשיו

}

}

/// <summary>

/// השחקן בחר מאיפה הוא רוצה להזיז - נסמן זהובים וכו

/// </summary>

private void userChoseFrom(int i, int j)

{

List<Tuple<int, int>> locations = logic.GetMoves(i, j);

for (int l = 0; l < locations.Count; l++)

{

cells[locations[l].Item1, locations[l].Item2].BackColor = Color.Gold;

}

from = new Tuple<int, int>(i, j); // נזכור מאיפה הוא רוצה להזיז

}

/// <summary>

/// השחקן בחר לאיפה הוא רוצה להזיז

/// </summary>

private void userChoseTo(int x, int y)

{

setButtonsToBlackAndWhite(); // למחוק את הסימונים הכחולים אם יש כאלה

if (logic.GetMoves(from.Item1, from.Item2).Contains(new Tuple<int, int>(x, y))) // אם הצעד חוקי

{

logicToUndoTo = logic.clone();

logic.move(from, new Tuple<int, int>(x, y)); // להזיז

setPicturesOnButtons(); // לדאוג שכל הכלים מצויירים במקומות הנכונים

undoToolStripMenuItem.Enabled = true;

from = null; // עכשיו צריך לקלוט מחדש מאיפה הוא רוצה להזיז

if (logic.currentPlayerLost())

MessageBox.Show("game over!");

}

else

{

from = new Tuple<int, int>(x, y);

userChoseFrom(x, y);

}

}

/// <summary>

/// כפתור נלחץ עם עכבר ימני

/// </summary>

public void cellClickedWithRightMouse(int x, int y)

{

if (logicBeforeDoubleEatingHasBegun == null) // בעצם אם הוא רק מתחיל את רצף האכילות הכפולות

{

if (from == null) // זה אם הוא סתם לחץ על עכבר ימני בלי לסמן קודם מאיפה הוא רוצה לזוז

return; // צא מהפעולה בלי לבצע את מה שקורה בהמשכה

}

var clickLocation = new Tuple<int, int>(x, y); // מיקום התא שנלחץ, לתוך טאפל סתם כי זה נוח לשורה הבאה

if (logic.eats(from.Item1, from.Item2).Contains(clickLocation)) // אם אפשר לאכול מאיפה שנבחר קודם לאיפה שנלחץ עכשיו

{

if (logicBeforeDoubleEatingHasBegun == null) // רק עכשיו אנחנו מתחילים אכילה כפולה

{

logicBeforeDoubleEatingHasBegun = logic.clone(); // כדי שאח"כ אנדו יחזור לתחילת האכילה הכפולה

}

logic.move(from, clickLocation); // לאכול

setButtonsToBlackAndWhite(); // מוחק את סימוני הזהב

var newEats = logic.eats(x, y); // עכשיו אחרי שביצענו את הצעד, נראה מה הוא יכול לאכול מכאן

if (newEats.Count > 0) // אם יש אכילות חדשות מאיפה שהוא הגיע

{

foreach (var newEat in newEats)

cells[newEat.Item1, newEat.Item2].BackColor = Color.Gold; // צובע אכילות חדשות בזהב

logic.manuallySwitchTurns(); // מה שקרה זה ש"מוב" החליפה תורות. אז אם יש עוד אכילה, נחזיר את התור

from = clickLocation;

}

else // כבר אין אכילות חדשות - נסיים את רצף האכילות הכפולות

{

logicToUndoTo = logicBeforeDoubleEatingHasBegun;

undoToolStripMenuItem.Enabled = true;

logicBeforeDoubleEatingHasBegun = null;

from = null;

}

setPicturesOnButtons(); // לדאוג שכל הכלים מצויירים במקומות הנכונים - וגם לעדכן בכותרת החלון תור מי

}

else

{

// נתעלם

}

}

/// <summary>

/// דואג שהרקע של כל הכפתורים יהיה שחור ולבן - לא משנה מה היה הצבע הקודם שלהם

/// </summary>

private void setButtonsToBlackAndWhite()

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

cells[i, j].resetColor();

}

/// <summary>

/// מתאים את התמונות שעל הכפתורים לצ'ארים שיש בלוח של הלוגיקה - וגם מעדכן את כותרת החלון שנדע תור מי

/// </summary>

private void setPicturesOnButtons()

{

for (int i = 0; i < 8; i++)

for (int j = 0; j < 8; j++)

cells[i, j].setImage(logic.getSoldier(i, j));

this.Text = "Checkers - turn of " + (logic.IsWhiteTurn() ? "White" : "Black");

}

private void howToPlayToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("בכדי לנצח אתה צריך לגרום שלשחקן השני לא יהיו מהלכים לעשות או שלא ישאר לו חיילים על המגרש");

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

/// <summary>

/// אם לוחצים על משחק חדש

/// </summary>

private void newToolStripMenuItem\_Click(object sender, EventArgs e)

{

logic = new logic();

setPicturesOnButtons(); // להתאים את התמונות לצ'ארים בלוגיקה

setButtonsToBlackAndWhite(); // לדאוג לרקע של הכפתורים שיהיה שחור לבן בלי שום זהב

from = null; // לא בחרנו חייל שרוצה לזוז

logicBeforeDoubleEatingHasBegun = null; // אין חייל שבאמצע אכילה כפולה

logicToUndoTo = null; // מאפסים הכל לפני תחילת משחק חדש...

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

private void newGameButtonClicked(object sender, EventArgs e)

{

logic = new logic();

setPicturesOnButtons();

panel1.Visible = false;

logicBeforeDoubleEatingHasBegun = null; // אין חייל שבאמצע אכילה כפולה

logicToUndoTo = null; // מאפסים הכל לפני תחילת משחק חדש...

}

private void ContinueFromSavedGameButtonClicked(object sender, EventArgs e)

{

// אם הפעולה נקראה סימן שהכפתור לחיץ

// וזה אומר שכבר שחזרנו את המשחק בפעולה הבונה

// אז כל שנותר זה

panel1.Visible = false;

logicBeforeDoubleEatingHasBegun = null; // אין חייל שבאמצע אכילה כפולה

logicToUndoTo = null;

}

private void undoToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (logicBeforeDoubleEatingHasBegun == null) // אם אתה לא באמצע אכילה כפולה

{

this.logic = logicToUndoTo;

logicToUndoTo = null;

setPicturesOnButtons();

setButtonsToBlackAndWhite();

undoToolStripMenuItem.Enabled = false;

}

else

{

MessageBox.Show("!סיים את תורך ואז ואז תוכל לעשות אנדו תודה");

}

}

private void label2\_Click(object sender, EventArgs e)

{

MessageBox.Show("בתחילת המשחק, האבנים מונחות על המשבצות השחורות של הלוח. ההתקדמות היא רק על משבצות אלה, רק קדימה, למעט כשאוכלים ורק באלכסון. אבני השחקן האחד מונחות בשלוש (ארבע, בגרסה הגדולה) השורות הראשונות של הלוח, ואילו אבני השחקן השני מונחות באותו אופן בצד שלו. לפי המקובל, השחור מבצע את המהלך הראשון, והלבן משיב עליו. שתי הפעולות גם יחד נחשבות כמסע אחד." +

"ברגע שחייל מגיע לאחת מהמשבצות של בשורה הראשונה של היריב החייל אוטומטית הופך למלך" +

" ." +

"מלך זה כמו חייל רגיל רק שיכול לזוז גם אחורה וגם קדימה");

}

private void Mainform\_Load(object sender, EventArgs e)

{

}

}

}

**files:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace damka\_game

{

class Files

{

static string path = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "//damka";

public static void SaveGame(logic logic)

{

using (Stream stream = File.Open(path, FileMode.OpenOrCreate))

{

var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

binaryFormatter.Serialize(stream, logic);

}

}

public static logic LoadGame()

{

using (Stream stream = File.Open(path, FileMode.Open))

{

var binaryFormatter = new System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();

return (logic)binaryFormatter.Deserialize(stream);

}

}

}

}