דוח סיכום פרויקט

**בנושא:**

משחק דמקה נגד שחקן אנושי/מחשב

מבוסס אלגוריתם Min-Max

[GITHUB](https://github.com/israel-sahar/final-project-checkers-2020)

[YOUTUBE VIDEO](https://www.youtube.com/watch?v=N7D5DRXu0ak)

**מבצעים:**

* סהר ישראל – ת.ז. 205654536
* אביתר חן – ת.ז. 307976928

**מנחה:** ד"ר תמר צמח

**תאריך:** יוני 2021

**דף תודות**

נרצה להודות לתמר, על כך שהסכימה ללוות אותנו בחיוך ושמחה והנחתה אותנו לאורך הדרך. תמר הייתה קשובה ונתנה לנו לבטא את עצמנו במהלך הפרויקט.

תודה ענקית מגיעה לדני על קורס C# שזכינו לקחת בו חלק, הקורס עבר ברמה מקצועית ומעניינת, דבר אשר גרם לנו להמשיך לעסוק בתחום גם מחוץ לקורס. למדנו כלים רבים בקורס זה אשר עזרו לנו רבות בפרויקט. הפרויקט מבוסס ברובו על חלונות WPF בשפת C#.

תודה אחרונה מיועדת לרשת האינטרנט. קהילה כמו StackOverflow אשר מפיצה את הידע הנרחב שלהם ובעזרתם פתרנו כל בעיה שנתקלענו בה.

**תוכן עניינים**

**תקציר4**

**חוקי המשחקי5**

**אלגוריתם Min-Max 6**

**גיזום אלפא-ביתא7-8**

**כלים לביצוע9**

**מבנה הפרויקט10**

**מסד הנתונים11-12**

**צד שרת13-14**

**צד לקוח15-19**

**הוראות הפעלת התוכנית20**

**ביבליוגרפיה 21**

**תקציר**

פרויקט זה נבחר אחרי חשיבה והרבה התלבטויות. היה לנו קושי רב וחוסר החלטיות לגבי בחירת הפרויקט. רצינו לבחור פרויקט אשר לא קשור למסלול הלימודים אותו בחרנו. לאחר עיון ברשימת הפרויקטים הרחבה במודל בחרנו לעשות את משחק הדמקה בתור משימתנו. בחרנו בפרויקט זה בשביל לטעום טיפה מעולם ה-AI , ורכישת כישורים נוספים בשפת C#. שמחים שיצא לנו לעבוד על פרויקט מהסוג הזה. בניית פרוייקט מאפס אשר הקמנו לו מסד נתונים, שרת, וצד לקוח.

המוצר מספק מגוון רחב של אפשרויות:

* משחק מול המחשב.
* משחק מול שחקן אנושי.
* צפייה חוזרת של משחקים שהסתיימו.
* צפייה במשחקים שקורים בזמן אמת.

נכנס לפירוט מעמיק בהמשך הדו"ח.

**חוקי המשחק**

דמקה הוא משחק לוח אשר מכיל 64 משבצות (8 על 8) או 100 משבצות (10 על 10).

מטרת המשחק היא להוריד מהלוח ("לאכול") את כל אבני המשחק של היריב, או לחסום אותן, כלומר לא לאפשר ליריב לקיים מסע (צעד/מהלך). שחקן שנשאר ללא אבני משחק או ללא מסע אפשרי מוכרז כמפסיד.

בתחילת המשחק, כל אבני המשחק שעל הלוח יקראו **חיילים**. כאשר השחקן מגיע עם אבן משחק לקצה הלוח, אבן המשחק תקרא **מלך**.

אלו חוקי המשחק שהמשחק מתבסס עליהם:

* מבחינת תנועה: **חייל** רגיל רשאי לנוע רק בכיוון המתאים, משבצת אחת (בתנאי שאין אכילה). **מלך** רשאי לנוע לכל הכיוונים באלכסונים וללא מגבלה.
* אכילה: ברגע שניתן לבצע שרשרת אכילות (אכילה רציפה) אין צורך לבצע את כל האכילות, אך חובה לאכול לפחות אחד. בנוסף, אין חובה לבחור מסלול אכילה שארוך ממסלול אחר.

אם קיים מסלול אכילה והשחקן בוחר לא לבצע את הפעולה, אזי אבן המשחק שבחר להזיז יוצאת מהמשחק.

**מלך** יכול לבצע אכילה לכל כיוון כל עוד קיים חייל של הקבוצה השניה הצמוד לו. **חייל** אינו יכול לבצע אכילה נגד כיוון התנועה. בנוסף, בשביל להפוך למלך צריך שמסלול האכילה יסתיים בסוף הלוח.

המשחק יכול להסתיים בניצחון או בתיקו. ניצחון מושג אם מתקיים אחד מהבאים:

1. לשחקן היריב לא נותרו כלל אבני משחק על הלוח (חיילים או מלכים).
2. לשחקן היריב אין אפשרות לבצע מהלך מאחר שכל אבני המשחק שלו חסומים.

תיקו מתקיים אם במשך 15 מהלכים רצופים נעו מלכים בלבד על גבי הלוח (אף חייל לא התקדם צעד), ולא השתנה מספר אבני המשחק על גבי הלוח (לא התבצעו אכילות).

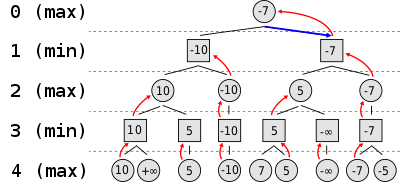
לפני המשחק על המשתמש לבחור:

1. **האם לשרוף** שחקנים שיכולים לאכול.
2. **גודל** השולחן הרצוי (8\*8 או 10\*10) .
3. **רמת הקושי** של שחקן המחשב (אם המשחק מתבצע נגד המחשב).

**אלגוריתם Min-Max**

אלגוריתם Min-Max סייע לחשב את המהלכים של המחשב בפרויקט.

כמו משחק הדמקה, משחקים רבים בנויים על עקרון של סט חוקים קבוע ומוגדר היטב ומשחק לסירוגין. האלגוריתם משתמש בעץ מינימקס בתור מבנה נתונים . ראש העץ מייצג את המצב הנוכחי בו נמצא הלוח, בשלב השני (ממוספר בתור **1** בתמונה) כל צומת תייצג מהלך ששחקן המחשב יכול לבצע. בשכבה השלישית (ממוספר בתור **2** בתמונה), כל צומת תייצג מהלך כתגובה למהלך שבוצע קודם. העלים בעץ מייצגים את המהלכים האחרונים שניתן לבצע(בחירת עומק העץ). עומק העץ תלוי בכוח החישוב של המחשב שלנו, גם מבחינת זכרון וגם מבחינת חישוב). לכל עלה בלוח מחושב 'ציון' שישקף עד כמה המצב טוב לאותו שחקן. בפונקציה המחשבת את הציון ניתן לתת דגש למלכים או שניתן פשוט לחשב את הפרש אבני המשחק. אלגוריתם זה נקרא Min-Max כי בכל שלב תבחר הצומת שהציון שלה הכי מקסימלי או הכי מינימלי בהתאם לתור. כל שחקן שואף למקסם את התשלום המינימלי שהוא יכול לקבל מהמשחק, או למזער את ההפסד המקסימלי.

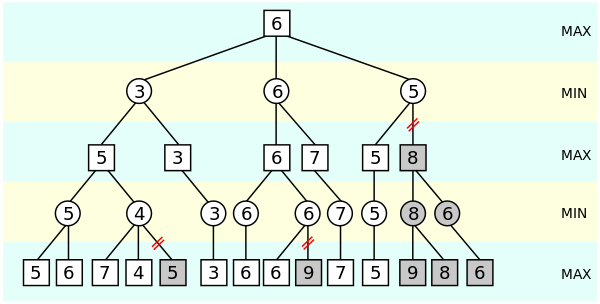


תיאור של עץ Min-Max

בעיה קשה בבניית עץ מינימקס היא הזיכרון הרב שהוא צורך. מספר הקודקודים שיש לפתח עולה בטור הנדסי ככל שנעמיק את החיפוש. קיימת שיטת "גיזום" המבטלת בנייה של ענפים שברור לנו עוד בשלב מוקדם כי הם לא מועילים לחיפוש שלנו. ניתן כך להקטין את מספר הצמתים בעץ לשורש המספר שהיה מתקבל ללא הגיזום (בממוצע). שיטה זו נקראת גיזום אלפא-ביתא.

**גיזום אלפא-ביתא**

גיזום אלפא-ביתא היא שיטת אופטימיזציה עבור עצי חיפוש מסוג Min-Max. השיטה נועדה לצמצם את מספר תתי העצים עליהם יש להלך בעת הערכת מהלך אפשרי בעץ. הגיזום אינו משנה את התוצאה שהיה מחזיר האלגוריתם המקורי, אלא רק מחזיר את התוצאה בזמן קצר יותר.במקרה הגרוע, זמן הריצה יהיה שווה לזמן ריצתו של האלגוריתם. במהלך החיפוש לעומק ניתן לזנוח פתרונות חלקיים ברגע שברור שהם גרועים מפתרונות שכבר ראינו.



**גיזום אלפא (Alpha cut)** –

אם הקודקוד הוא קודקוד של שחקן ה- Max זה אומר שתורו של ה-Max לשחק , השחקן לא יבחר תתי-עצים בעלי תוצאה נמוכה יותר מזו שהושגה בתת-עץ הקודם.  
הרציונל לכך הוא ש- Max תמיד בוחר למקסם , כלומר , אם Max כבר יודע על מהלך בעל ניקוד מסוים , אם באחד התתי עצים שלו שחקן ה-Min הצליח להשיג ניקוד נמוך יותר על ידי אחד תתי העצים של עצמו, אז שחקן ה-Min לא צריך לבדוק את שאר תתי העצים משום שלא יבחר ניקוד גבוה יותר ממה שהשיג כעת .

**גיזום ביתא (Beta cut)** –

אם הקודקוד הוא קודקוד של שחקן ה-Min - נבצע פעולה הפוכה.  
לאלגוריתם יש שני משתני עזר , אלפא וביתא , המייצגים את התוצאה המינימלית המובטחת לשחקן ה- Max ואת התוצאה המקסימלית ל-Min.   
בהתחלה נקבעים משתנים אלו להיות בעלי ערך קיצון (מינוס אינסוף ואינסוף) שהם מייצגים את המצב הגרוע ביותר מבחינתו של כל שחקן.  
עם התקדמות החיפוש , הולך ומצטמצם המרחק בין אלפא לביתא , כאשר ביתא מקבלת ערך נמוך מאלפא זה מצביע על כך שהעמדה הנוכחית לא תוביל לתוצאה הטובה ביותר האפשרית לשחקן הנוכחי .

**כלים לביצוע**

אז כמו שנאמר קודם. הפרויקט כולו התבצע בשפת C#. הפרויקט נכתב בסביבת עבודה Microsoft Visual Studio.

צד הלקוח מומש על ידי כלי הפיתוח של Microsoft הנקרא WPF, אשר שם דגש על עיצוב גרפי מתקדם של ממשק המשתמש. ממשק המשתמש מעוצב באמצעות שפת תגיות מבוססת XML הקרויה XAML.

צד השרת מומש על ידי WCF, אשר מהווה מודל מערכת וממשקי API ב-.Net Framework ליצירת תקשורת בין יישומים ובניית ישומים מבוזרים, מוכווני שירותים.

המסד נתונים נוצר בעזרת Entity Framework, ספריה של מיקרוסופט, כלי שמשמש בתיכנות לגישה קלה לנתונים, לעבודה קלה יותר עם דאטאבייסים.

בנוסף, לצורך צפיית משחקים בלייב היה שימוש בספריה אשר מקימה ומנהלת תקשורת TCP בצורה קלה ונוחה, ללא התעסקות רבה מצד המתכנת, שם הספריה הוא SimpleTCP.

**מבנה הפרויקט**

הפרויקט התחלק לשלושה חלקים מרכזיים:

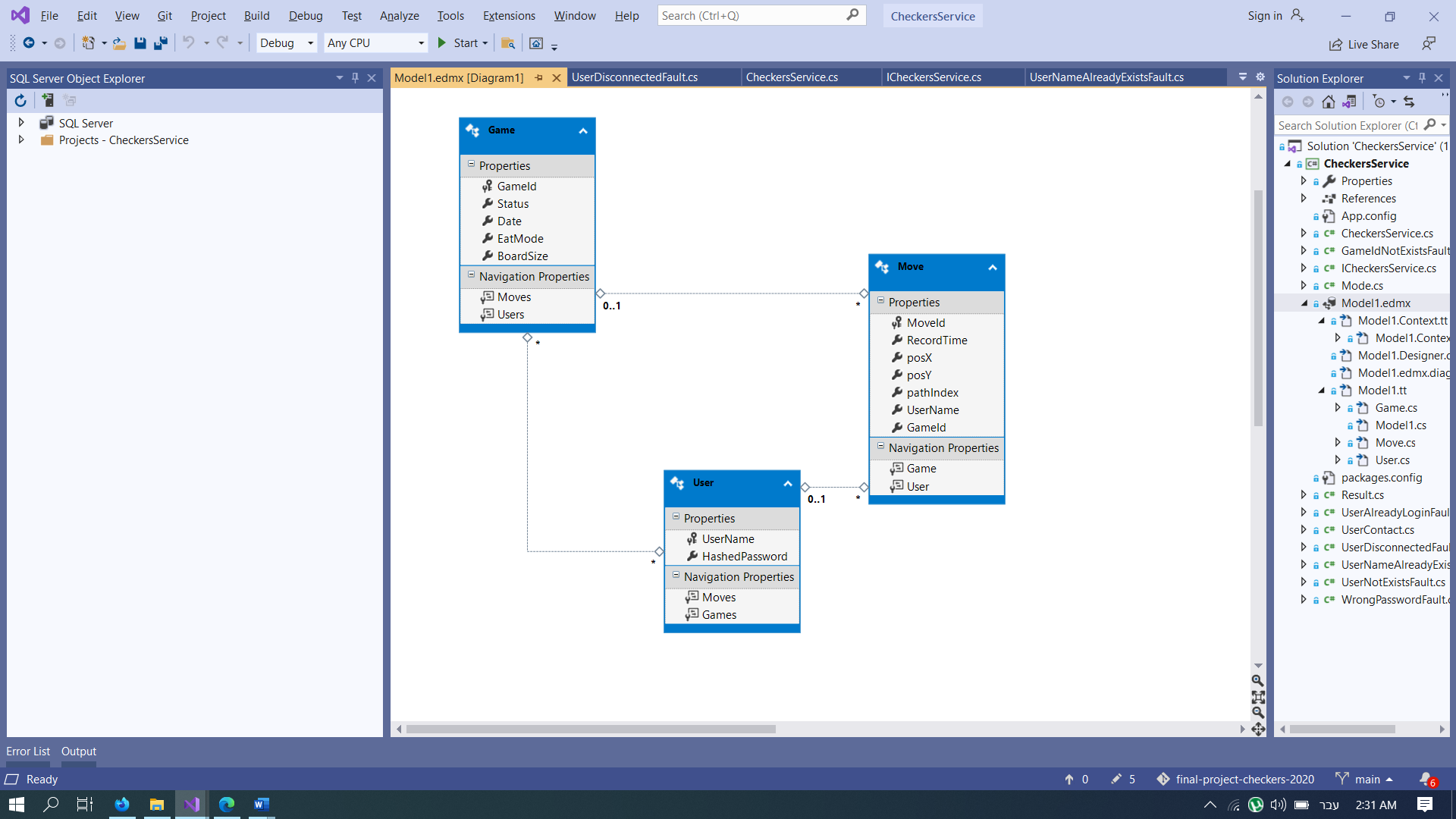
* **מסד הנתונים –** מחזיק במידע על המשתמשים הקיימים במערכת והיסטוריית המשחקים לצורך צפייה חוזרת.
* **צד שרת –** מהווה כתקשורת ללקוחות. במעקב אחרי משתמשים מחוברים והמשחקים הפעילים. דואג לחבר בין שחקנים אשר מעוניינים לשחק. דואג לחבר בין צופה לייב למארח המשחק.
* **צד לקוח –** מהווה את כל החלונות שהלקוח רואה. אם זה מסך ההתחברות, מסך ההרשמה, מסך המשחק, צפייה חוזרת, צפייה ישירה וכו'.בנוסף, מנהל את כל המשחק מאחורי הקלעים

1. מסד הנתונים

היה ניסיון לשמור על מסד הנתונים כמה שיותר פשוט. כמובן שתמיד יש איפה להרחיב, כמו לדאוג להרשמה בעזרת אימייל, איפוס סיסמא בעזרת טלפון, שמירת כמות הנצחונות וכו'.

החלטנו שמעדיפים להשתמש בכוח החישוב של מכונת הלקוח מאשר תפיסת מקום נוסף במסד הנתונים. עשינו זאת בעזרת שמירת אינדקס המהלך שהמשתמש ביצע, במקום לשמור כל צעד וצעד שביצע במהלך.

דיאגרמת UML אשר מייצגת את מסד הנתונים:



המסד כולל רק 3 מחלקות:Game, User, Move.

* יישות Game – בנויה מאינדקס אשר נבחר ע"י המערכת, סטטוס המשחק (נצחון, תיקו, עדיין מתקיים), גודל הלוח, תאריך, ומצב אכילה(ערך בוליאני)
* יישות User – מכילה רק שם המשתמש ומחרוזת Hash של הסיסמא של הלקוח. בתור הרחבה לפרויקט ניתן להוסיף סטטיסטיקות ומידע נוסף.
* יישות Move- מייצגת מהלך אחד שהתבצע במשחק. מציינת את מיקום השחקן הנוכחי, שם השחקן, שם המשחק,ואינדקס המהלך שנבחר

קיימת טבלה רביעית שנסתרת על ידי המערכת. טבלה אשר מקשרת בין משחק לשחקן.

1. צד שרת

אומנם הוא לא אחראי למהלכים של המחשב, אך תקשורת הלקוחות תלויה במהירות השרת. ניסינו להקל על השרת כמה שיותר. היינו יכולים להקל גם אם היינו מקימים תקשורת TCP בין 2 שחקנים וככה השרת לא יהיה מעורב עד תום המשחק.

נסביר בגדול איך השרת עובד וקצת על פונקציות עיקריות אצלו.

טיפוסי enum:

* Mode : מציין מה מצב השחקן במצב הנוכחי. עוזר בשלב ההתנתקות ( האם בלובי? האם משחק? האם צופה? ).
* Result : מציין תוצאת משחק לשחקן שביצע מהלך. הפסד, ניצחון, תיקון,המשך משחק.
* Status : מיועד יותר לקביעת סטטוס למשחק.שחקן ראשון/שני ניצח, תיקו, המשחק לא נגמר.

השרת מחזיק בשלושה מילונים:

* 1. מילון אשר מייצג את המשתמשים המחוברים ברגע נתון.
  2. מילון אשר מייצג 4 רשימות המתנה למשחק. קיימים 4 חדרי המתנה בגלל שיש לבחור גודל שולחן, ומצב אכילה.
  3. מילון אשר מייצג את המשחקים המתרחשים ברגע נתון.

ברגע ששחקן ראשון שולח בקשה להצטרפות למשחק. השרת מבצע את הפעולות הבאות:

1. בדוק האם קיים שחקן שממתין.
2. אם כן,
   1. צור משחק חדש [השחקן שהצטרף כעת מתחיל את המשחק].
3. אחרת, הוסף את השחקן לרשימת ההמתנה, עד להצטרפות שחקן חדש.

במידה ושחקן רוצה לצפות במשחק לייב, השרת מבקש מהמארח(השחקן שהתחיל את המשחק) IP ופורט. שולח בחזרה למבקש, וכאן מסתיים תפקידו של השרת.

רשימת פונקציות מרכזיות שמתבצעות ע"י השרת ותיאורן:

1. Connect(string usrName, string hashedPassword)

Register(string userName, string hashedPassword)

**קלט:** שם משתמש,סיסמא מגובבת.

**תיאור:** בהתאם לפונקציה שנקראה. השרת יבדוק האם קיים משתמש במערכת באותו שם, או שמחובר למערכת. במידה והמשתמש יכול להתחבר תתבצע בדיוק אם הסיסמא תואמת. לאחר שתי הקריאות המשתמש יחשב כמחובר למערכת.

1. (int, string, bool) JoinGame(string user, bool isVsCPU, int boardSize, bool EatMode)

**קלט:** שם משתמש,האם משחק מול המחשב או לא, גודל השולחן המבוקש, האם בחר באפשרות של חובת אכילה.

**תיאור:** במידה והמשחק הינו מול המחשב. השרת יוצר את המשחק ומעדכן את רשימותיו. אחרת, השחקן ימתין לשחקן נוסף שבחר את אותו סוג משחק או שיווצר משחק כי יש שחקן שממתין.

**פלט:**Tuple אשר מייצג מספר משחק, שם היריב, האם התור הראשון של השחקן הנוכחי או לא.

1. (string,int) StartWatchGame(int gameId)

**קלט:** המשחק שהשחקן הפונה ירצה לראות

**פלט:** כתובת IP ופורט של מארח המשחק. השחקן שתורו ראשון.

1. MakeMove(string UserName, int GameId, Point correntPos, int indexPath, Result result)

**קלט:** שם משתמש שביצע מהלך, מספר משחק, הנקודה הנוכחית בה נמצאת אבן המשחק,מספר המהלך,תוצאת המהלך

**תיאור:** הפונקציה מוסיפה את מהלך המשתמש למסד הנתונים. מעדכנת את היריב במהלך ובתוצאת המהלך. נזכור כי היריב מקבל את מספר המהלך והמיקום ומחשב לבד את המהלך שהשחקן עשה. מעדכן את רשימותיו במידת הצורך.

3. צד לקוח

צד הלקוח הוא החלק המרכזי בפרויקט. מהווה נתח גדול מכלל הפרויקט. הוא מחולק ל-2 אגפים: ניהול המשחק מאחורי הקלעים (ייקרא Backend), והחלונות המוצגים למשתמש(ייקרא UI).

ה-Backend מכיל כמה מחלקות אשר מנהלות את המשחק. מהקמת לוח המשחק עם כל אבני המשחק, ועד חישוב כל המסלולים האפשריים לאבן משחק שנבחרה בהתאם לתפקידה ( חייל / מלך ).

להלן המחלקות שה-Backend מכיל ותיאורן:

* מחלקת Board – המחלקה מנהלת את המשחק. יוצרת את הלוח עם אבני המשחק של שתי הקבוצות. מבצעת מהלכים ובודקת את תוצאת המהלך. חלק מהחוקים שצויינו מעלה ממומשים במחלקה זאת.
* מחלקת Checker – יורשת ממחלקה אבסטרקית Piece. מחלקה זאת מייצגת אבן משחק כחייל. במחלקה יש שדה אשר מצביע על כיוון התנועה של החייל. המתודה GetPossibleMoves מחשבת את כל המסלולים האפשריים שחייל יכול לבצע.
* מחלקת King – מחלקה זאת גם יורשת ממחלקה אבסטרקית Piece. מחלקה זאת מייצגת אבן משחק כמלך. מלך רשאי לנוע לכל כיוון. גם במחלקה זאת יש מתודה הנקראת GetPossibleMoves בהתאם לחוקי תנועתו של המלך.
* מחלקת Path – מתארת מסלול של מהלך יחיד. מחזיק בשתי רשימות: קורדינטות של אבני המשחק הנאכלים במהלך זה, הקורדינטות שאליהם אבן המשחק צריכה לנוע.
* מחלקת ComputerMove – מייצגת את שחקן המחשב. במחלקה מחושב המהלך הבא של המחשב, בהתאם לרמת הקושי שנבחרה.

ניתן לבחור עד שלוש רמות קושי: קל, בינוני וקשה.

**קל –** במידה וקיים מהלך בו יש מסלול אכילה ארוך לעומת שאר המהלכים, מהלך זה יבחר.אחרת, יבחר מהלך באופן אקראי.

**בינוני –** שימוש באלגוריתם Min-Max שעומק העץ הוא שניים. המחשב חושב שני צעדים קדימה.

**קשה –** שימוש באלגוריתם Min-Max שעומק העץ הוא ארבע. המחשב חושב ארבעה צעדים קדימה.

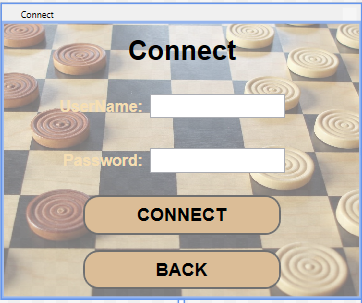
החלונות המוצגים למשתמש הם להלן:

* WelcomeWindow – מאפשר להתקדם להתחברות/הרשמה. קיימת אפשרות להתחיל משחק מול המחשב במצב Offline. המצב מתבצע מול המחשב והמהלכים לא עוברים לשרת.

חשוב: בכל לחיצה על כפתור PLAY, המשתמש ישאל: איזה גודל שולחן ירצה, מה רמת קושי המשחק, והאם מעוניין לשרוף חיילים בעת הצורך(לא אוכלים שיש להם אופציה).



* LoginWindow/RegisterWindow – חלונות שמיועדים להרשמה והתחברות. לאחר הרשמה, משתמש מחובר אוטומטית ועובר לתפריט. הסיסמא עוברת בפונקציית Hash לפני שנשלחת לשרת.

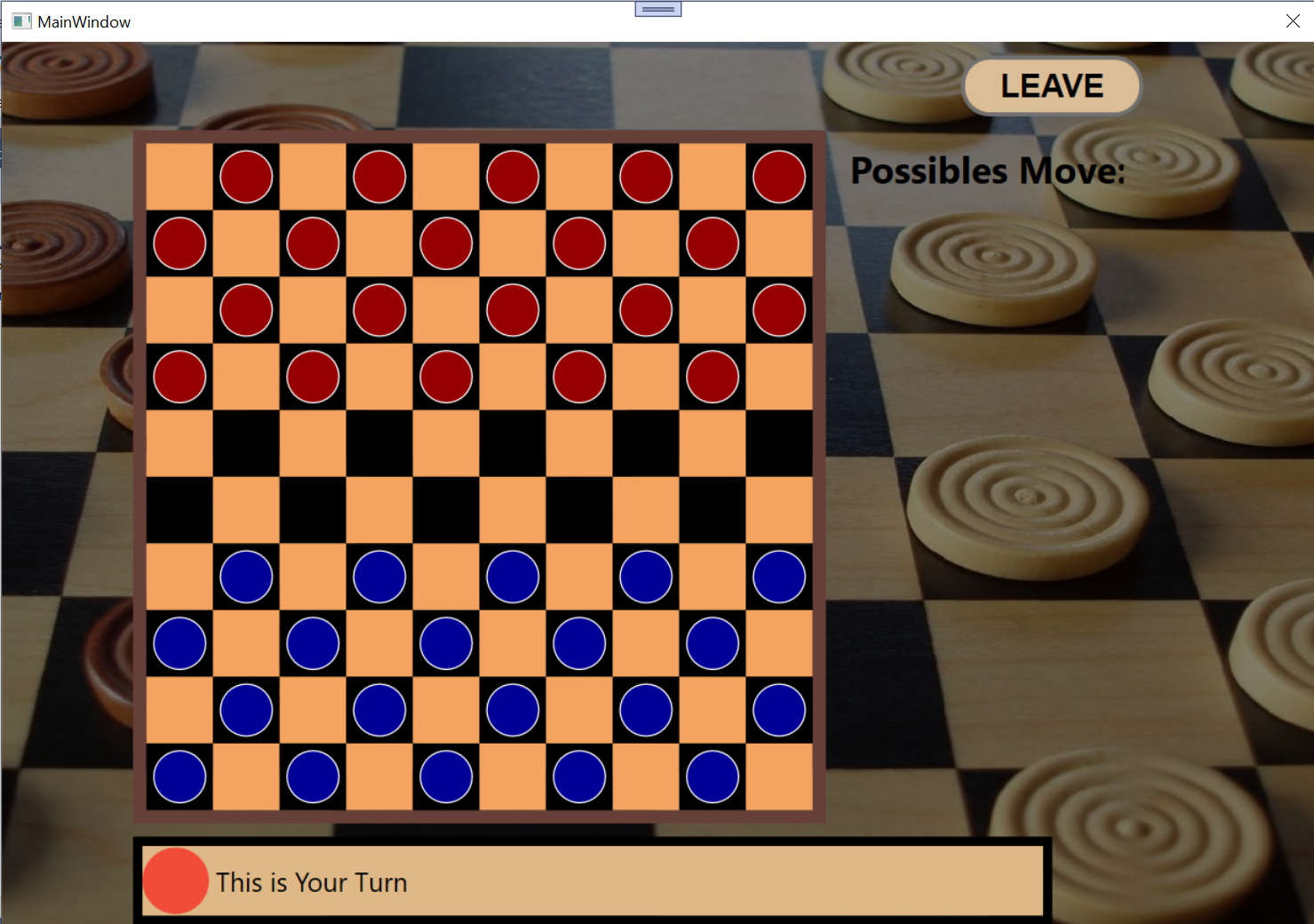


* MenuWindow – חלון המופיע לאחר התחברות. ארבע אפשרויות ללקוח:
  1. בחירת התחלת משחק מול המחשב.
  2. בחירת התחלת משחק מול שחקן אחר.
  3. צפייה במשחקים קודמים.
  4. צפייה במשחקים שמתקיימים כעת.

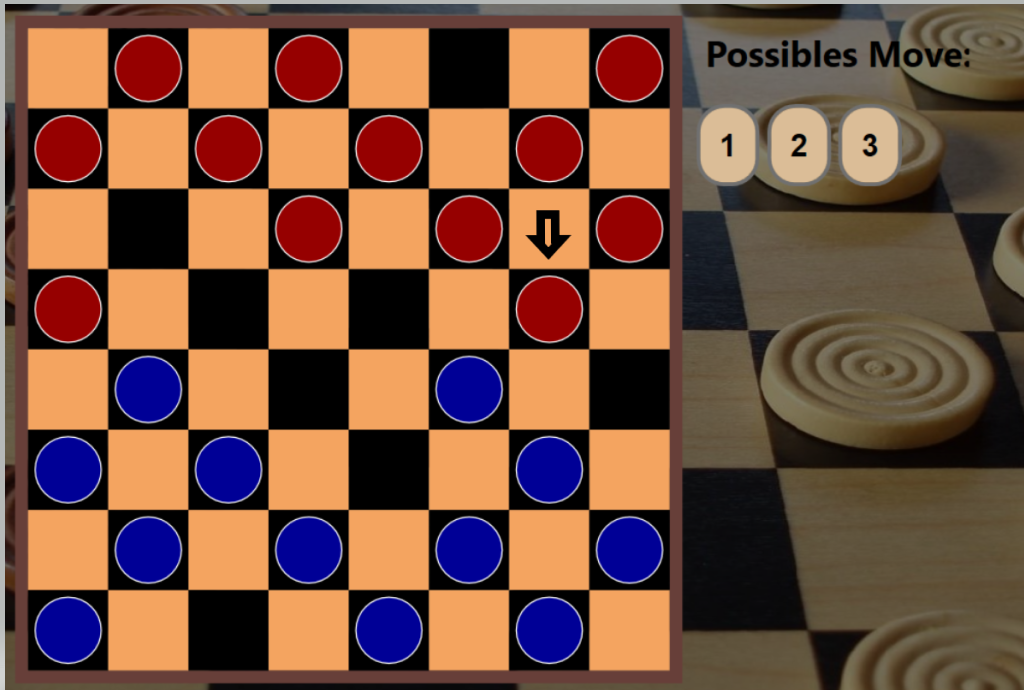
באופציה b יש לבחור את גודל השולחן הרצוי והאם לבצע שריפת חיילים או לא.



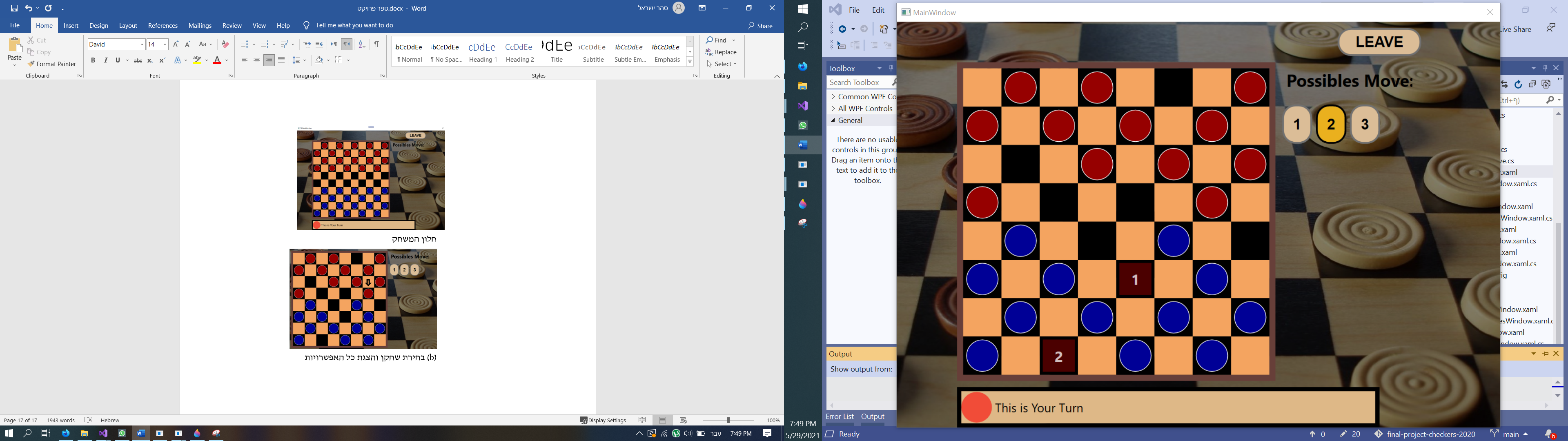
* WaitingWindow – חלון קטן בו שחקן ממתין ליריב שלו. המשתמש רשאי לחזור לחלון הראשי, ותשלח בקשה לשרת שהמשתמש אינו ממתין עוד.
* GameWindow - החלון בו כל המשחק מתרחש. המשתמש מבצע מהלך באופן הבא:
  1. לחיצה על אבן המשחק אותו השחקן רוצה להזיז.
  2. בצד ימין תופיע רשימת המהלכים האפשריים.
  3. מעבר עם העכבר על מהלך כלשהו יראה על לוח המשחק את המסלול שאבן המשחק יבצע במהלך הנבחר.
  4. לאחר בחירת המהלך, תבוצע האנימציה והמהלך יישלח לשרת.



חלון המשחק.

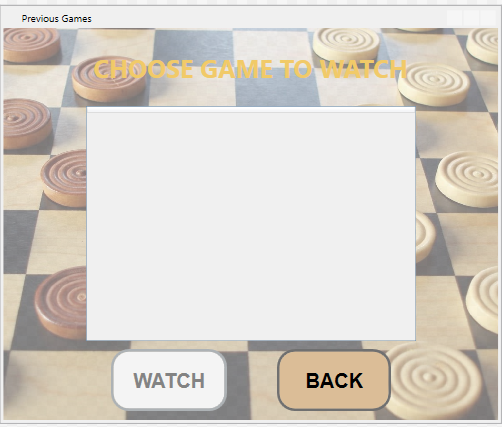


(b) בחירת אבו משחק והצגת כל האפשרויות.



(c) הצגת המהלך על לוח המשחק.

* PrevGamesWindow/LiveGamesWindow – מציג את המשחקים שמתרחשים כרגע/ שהתבצעו בעבר. יש לבחור את אחד המשחקים, וללחוץ על WATCH.



* WatchingGameWindow – בחלון זה מתבצעת הצפייה גם במשחקים שבוצעו וגם במשחקים שמתבצעים בזה הרגע. בבחירת משחק עבר, יופיע כפתור START לתחילת הצפייה וכפתור להרצת המשחק מחדש. בבחירת משחק לייב, המהלכים שהתבצעו יוצגו למשתמש עד הגעה למהלך הנוכחי.



**הוראות הפעלת התוכנית**

השימוש במוצר הוא כרגע בצורה לוקאלית.

[ לוודא שמותקן על המחשב Microsoft Visual Studio ]

הדרכת וידאו: <https://youtu.be/q2CpgdWc0j0>

1. יש להוריד את קבצי הפרויקט מהגיט.Code->Download Zip

לינק: <https://github.com/israel-sahar/final-project-checkers-2020>

1. חלצו את קבצי הפרויקט.
2. יש ליצור את התיקייה הבאה: C:\Checkers
3. גררו את הקבצים הבאים לתיקיית Checkers: CheckersDB.mdf , CheckersDB\_log.ldf. הקבצים יושבים בתיקיה הראשית של הפרויקט.
4. יש להכנס לפרויקט הנקרא CheckersService דרך הקובץ הבא:

final-project-checkers-2020-main\CheckersService\ CheckersService.sln

לאחר שנכנסתם לפרויקט יש לבצע לו Build.

1. העתיקו את הקובץ

final-project-checkers-2020-main\CheckersService\bin\Debug\ CheckersService.dll

לתיקיה Checkers שיצרנו בסעיף 3.

1. יש להכנס לפרויקט הנקרא GameHost. הרצת קובץ ה-sln בתיקיה GameHost. יש לטעון את קובץ ה-dll באופן הבא:
   1. ב-Solution Explorer תופיע רשימה הנקראת References.
   2. לחצו לחצן ימני על הכותרת References ולחץ על Add Reference.
   3. יופיע חלון חדש. יש ללחוץ על browse ולבחור את הקובץ CheckersService.dll שהעברנו לתיקייה Checkers.
2. כעת אפשר להפעיל את פרויקט ה-Host. אמור להופיע חלון עם הכיתוב RUNNING.זה הסימן שהשירות דלוק
3. יש להכנס לפרויקט הנקרא Client. ניתן להריץ את התוכנית כמה פעמים במקביל בעזרת CTRL+F5.
4. במסד הנתונים יש שלושה משתמשים שהבודק יוכל להשתמש:user1, user2, user3. הסיסמא לשלושת המשתמשים היא 123456.

**ביבלוגרפיה**

# [1] Windows Communication Foundation

<https://he.wikipedia.org/wiki/Windows_Communication_Foundation>

<https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf>

# [2] Windows Presentation Foundation

<https://en.wikipedia.org/wiki/Windows_Presentation_Foundation>

[3] Information about Checkers game and the rules of the game

Hebrew - [https://he.wikipedia.org/wiki/דמקה](https://he.wikipedia.org/wiki/%D7%93%D7%9E%D7%A7%D7%94)

English - <https://en.wikipedia.org/wiki/Draughts>

[4] Min-Max

<https://en.wikipedia.org/wiki/Minimax>

[https://he.wikipedia.org/wiki/משפט](https://he.wikipedia.org/wiki/%D7%9E%D7%A9%D7%A4%D7%98_%D7%94%D7%9E%D7%99%D7%A0%D7%99%D7%9E%D7%A7%D7%A1)

# [5] Alpha-Beta pruning

[https://he.wikipedia.org/wiki/alpha-beta](https://he.wikipedia.org/wiki/%D7%92%D7%99%D7%96%D7%95%D7%9D_%D7%90%D7%9C%D7%A4%D7%90-%D7%91%D7%99%D7%AA%D7%90)

[6] Open Source – SimpleTCP

<https://github.com/jchristn/SimpleTcp>