**עבודת גמר**

**לקבלת תואר**

**טכנאי תוכנה**

**הנושא: פיתוח משחק "דמקה סינית"**

**המגיש : בן גדעון פלנטר**

**ת.ז. המגיש: 32563871**

**שם המנחה : מיכאל צ'רנובילסקי**

אפריל 2023 תשפ"ג

תוכן

[תקציר 3](#_Toc129342585)

[מושגים 4](#_Toc129342586)

[תיאור המשחק 5](#_Toc129342587)

[רקע 5](#_Toc129342588)

[חוקי המשחק 6](#_Toc129342589)

[רקע תאורתי בתחום הנושא 7](#_Toc129342590)

[הבעיה האלגוריתמית 9](#_Toc129342591)

[האלגוריתם הנבחר 10](#_Toc129342592)

[מבוא 10](#_Toc129342593)

[הסבר 10](#_Toc129342594)

[השלב הראשון - הערכת מצב המשחק 10](#_Toc129342595)

[השלב השני – מציאת מהלכים אפשריים 11](#_Toc129342596)

[השלב השלישי – בחירת המהלך הטוב ביותר 11](#_Toc129342597)

[השלב הרביעי – החלפת תור ובדיקת ניצחון 14](#_Toc129342598)

[ארכיטקטורת הפתרון 15](#_Toc129342599)

[ארכיטקטורת MVC (Model View Controller) 15](#_Toc129342600)

[Top Down Level Design 15](#_Toc129342601)

[MODEL 15](#_Toc129342602)

[VIEW 16](#_Toc129342603)

[CONTROLLER 16](#_Toc129342604)

[Use Cases Diagram 17](#_Toc129342605)

[מבני נתונים 18](#_Toc129342606)

[שפת התכנות וסביבת העבודה 20](#_Toc129342607)

# תקציר

משחק הדמקה הסיני הוא משחק לוח פופולרי כבר למעלה ממאה שנה, לוח המשחק בנוי ממשושה עם 10 חורים בכל אחד מששת הצדדים. מטרת המשחק היא להעביר את כל הכלים של אחד השחקנים ממשולש ההתחלה שלו לצד הנגדי של הלוח לפני שהיריב יעשה את אותו הדבר. בפרויקט זה פיתחתי משחק דמקה סינית עם שני מצבים; 2 שחקנים ושחקן אחד נגד המחשב.

הבעיה האלגוריתמית העיקרית שכוונתי לפתור בפרויקט הזה הייתה יצירת יריב ממוחשב שיכול לשחק נגד השחקן. זה הצריך פיתוח אלגוריתם מתאים שיוכל לבצע מהלכים חכמים על בסיס הכרעת מצבי הלוח הנוכחי.

בתיק פרויקט זה אציג ואת תהליך פיתוח המשחק "דמקה סינית", אסקור תאוריות ומושגים בתחום נושא המשחק, אציג את האלגוריתם שבחרתי לממש כדי לפתור את בעיית הכרעת המצבים, אציג את מבני הנתונים השונים שבחרתי כדי לממש את האלגוריתם, האלגוריתם עצמו לשחקן ממוחשב ומימושו בקוד, המחלקות השונות שיצרתי והקשרים ביניהן, ואת התוצר הסופי שהגעתי אליו בסוף תהליך זה. בנוסף, ארחיב על תהליך המחשבה שהוביל אותי למצב הסופי של הפרויקט, על קשיים שבהם נתקלת י, ועל אלגוריתמים אלטרנטיביים שבהם ניסיתי להשתמש לאורך הדרך. יימצאו בתיק גם תרשימים שונים המתארים את מבנה הפרויקט.

# מושגים

* בינה מלאכותית - בינה מלאכותית (intelligence Artificial או בקיצור AI )הוא ענף בתחום מדעי המחשב העוסק ביכולת לתוכנת מחשב לפעול באופן המציג יכולות שאפיינו עד כה את האינטליגנציה האנושית בלבד. המבחן המקובל ביותר לבינה מלאכותית הוטבע בשנת 1950 על ידי אלן טיורינג, וידוע בשם "מבחן טיורינג": מכונה תחשב לתבונית, אם יינתן לאדם, "הבוחן" היושב בחדר סגור, לנהל שיחה באמצעות ממשק מחשב (Console) עם שתי ישויות שנמצאות בחדר השני, כאשר אחת מהן תהיה מכונה והשנייה אנושית, והמשוחח לא יוכל לזהות מי משתי הישויות היא מכונה או אדם מכונה.
* הכרעת מצבים - בהקשר של תכנות שחקן ממוחשב, "הכרעת מצבים" מתייחסת לנקודות הקריטיות במשחק בהן האלגוריתם צריך לקבל החלטה לגבי הצעד הבא. לדוגמה, במשחק שח, מצב מכריע יכול להיווצר כאשר הבינה המלאכותית צריכה להחליט איזו מהלך כדאי לה לבצע. זיהוי וטיפול יעיל במצבי הכרעה אלו הם קריטיים לשחקן הבינה המלאכותית לשחק את המשחק בצורה מיטבית ולקבל החלטות טובות שיובילו לניצחון. זה דורש מהשחקן הממוחשב להעריך את המצב הנוכחי של המשחק, לשקול מהלכים אפשריים שונים ואת ההשלכות הפוטנציאליות שלהם, ולבחור את המהלך שסביר להניח שיוביל לתוצאה חיובית.
* היוריסטיקה – גישה לפתרון בעיות שמפעילה שיטה פרקטית לפתרון שאינו בהכרח אופטימלי, לרוב נשתמש בהיוריסטיקה על מנת ליעל את האלגוריתמים הקיימים שלנו ולהאיץ אותם.
* פסאודו קוד – הוא תיאור מצומצם ולא רשמי לאלגוריתם של תוכנית מחשב. פסאודו קוד משתמש בקונבנציות של שפת תכנות, אך מיועד לקריאה של בני אדם לא לקריאה על ידי מחשב.

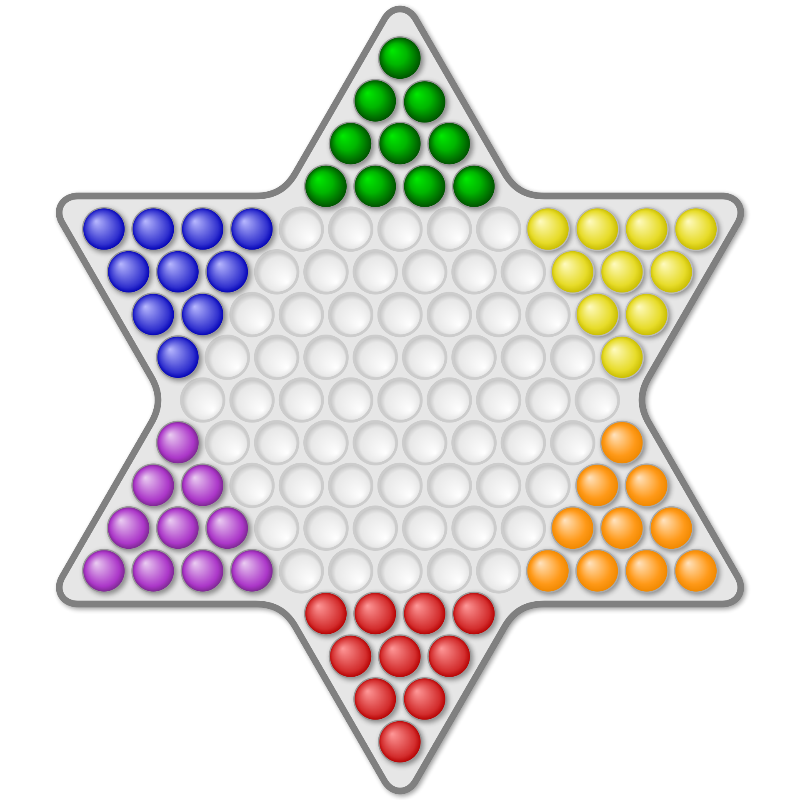
# תיאור המשחק

## https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/ChineseCheckersboard.jpeg/250px-ChineseCheckersboard.jpegרקע

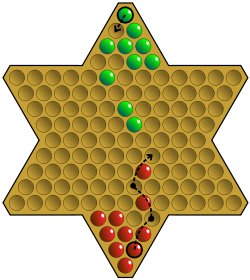
דמקה סינית או "הלמה" כוכב הוא משחק לוח המיועד לשניים עד שישה שחקנים. המשחק הוא גרסה של המשחק הלמה שפותח בשלהי המאה ה-19, והמטרה בו היא להעביר את כלי המשחק, על פי רוב גולות או יתדות, לצד המנוגד של לוח המשחק.

למרות שמו, המשחק דמקה סינית לא הומצא בסין או באסיה כלל. הוא אף אינו גרסה של המשחק דמקה. המשחק הומצא בגרמניה בשנת 1892 תחת השם "שטרן-הלמה" (הלמה כוכב), כגרסה למשחק הלמה האמריקאי. שמו הנוכחי ניתן לו בארצות הברית ב-1928, כאמצעי שיווקי של ביל וג'ק פרסמן, אך בטרם זאת נקרא דמקה הופ צ'ינג.

לוח המשחק עשוי 121 שקערוריות (גומות) המסודרות בצורת מגן דוד, כאשר המשולשים הקטנים המרכיבים את פינותיו מכילים עשר שקערוריות כל אחד. בתחילת המשחק מציב כל שחקן עשרה כלי משחק מאותו צבע באחד מקדקדיו של הלוח. כך נראה לוח המשחק המקורי:



## חוקי המשחק

* השחקן בצבע שחור תמיד מתחיל ראשון.
* בתחילת המשחק מוצבים עשרה כלי משחק מאותו צבע באחד מקדקדיו של הלוח.
* כל שחקן בתורו מזיז כלי משחק אחד, או לשקערורית סמוכה פנויה או תוך כדי דילוג מעל כלי משחק אחד אחר, אין זה משנה של איזה שחקן, אל השקערורית הפנויה הבאה אחריה.
* אסור לדלג מעל שני כלי משחק צמודים, אך ניתן לבצע באותו מהלך מספר דילוגים, ובתנאי שבכל דילוג בשרשרת יונח כלי המשחק בשקערורית פנויה.
* הדילוג יכול להתבצע באלכסונים, כמו התנועה, וכלִי המשחק שמדלגים מעליו אינו מושפע מכך.
* מטרת המשחק היא להעביר את כל עשרת כלי המשחק של השחקן מפינה משולשת אחת אל הפינה המשולשת הנגדית בטרם יעשה זאת שחקן אחר. המנצח הוא הראשון שמצליח להעביר את כל כליו לפינה הנגדית.

# רקע תאורתי בתחום הנושא

לאורך השנים, המשחק דמקה סינית היווה השראה לתיאוריות ואלגוריתמים רבים. התיאוריות והאלגוריתמים הללו נועדו לעזור לשחקנים לקבל החלטות אסטרטגיות ולשפר את סיכויי הניצחון שלהם.

אחת מהתאוריות החשובות בנושא היא תיאוריית "הנתיב הקצר ביותר" (Shortest Path). לא ברור מי פיתוח את התאוריה הזאת מכיוון שהיא אסטרטגיה נפוצה כבר שנים. לפי תיאוריה זו, על השחקנים לשאוף להזיז את הכלים שלהם לאורך הנתיב הקצר ביותר לבסיס היריב שלהם על לוח, כלומר ליצור מסלול שמאפשר קפיצות רחוקות ביותר במינימום הזזות. הטענה לפי תאוריה זו היא שאם שחקן יכול לשים חייל שלו בבסיס של האויב, הוא יכול להתחיל לקפוץ מעל חייליו של האויב, בתקווה שהוא יגיע למטרתו בזמן יותר קצר.

כדי ליישם תיאוריה זו, על השחקנים להשתמש בשילוב של קפיצה כמה שיותר רחוקה, ומיקום אסטרטגי של החייל. על ידי מיקום הכלים שלהם בצורה המאפשרת סדרה של קפיצות, שחקנים יכולים להעביר במהירות את הכלים שלהם על פני הלוח. חשוב גם לשחקנים לחסום את הכלים של היריב ולהגביל את אפשרויות התנועה שלהם כדי למנוע מהם להשתמש באותה אסטרטגיה.

תיאוריה נוספת בדמקה סינית היא תיאוריית ה"תנועה הכפויה" (Forced Move). תיאוריה זו מציעה חיפוש מהלכים המאלצים את היריב לנוע בכיוון מסוים או לבצע פעולה מסוימת. ניתן להשיג זאת על ידי מיקום חיילים באופן שמגביל את האפשרויות של היריב, או על ידי יצירת מצבים שבהם היריב נאלץ לבצע מהלך ספציפי.

אלגוריתם נוסף בשימוש בדמקה סינית הוא אלגוריתם Greedy. אלגוריתם זה פועל על ידי הערכת כל מהלך אפשרי ובחירת המהלך שנותן מיד את התוצאה הטובה ביותר, על בסיס המצב הנוכחי של הלוח.

אלגוריתם זה יכול להיות שימושי במצבים מסוימים, כגון כאשר שחקן צריך להזיז במהירות את הכלים שלו לכיוון הצד הנגדי של הלוח כדי להשיג יתרון על היריב שלו, אך הוא לא תמיד הוא יוביל לתוצאה הטובה ביותר לטווח ארוך.

הסיבה לכך היא שהאלגוריתם שוקל רק את ההשלכות המיידיות של מהלך ואינו לוקח בחשבון את ההשלכות העתידיות הפוטנציאליות. למשל, מהלך שנראה מועיל בטווח הקצר עלול למעשה להשאיר את השחקן פגיע למהלכים עתידיים של היריב. אלגוריתם Greedy הוא אלגוריתם טוב למצב בו אין חשיבות למצבים עתידיים ואין יכולת להשתמש בעצי החלטה.

אלגוריתמים נוספים שחשוב לציין הם Minimax. Alpha – Beta Pruning ו- Monte Carlo:

* Monte Carlo Tree Search הוא אלגוריתם לקבלת החלטות המדמה מספר רב של תוצאות משחק ובוחר את המהלך שיש לו את ההסתברות הגבוהה ביותר להוביל לזכייה. .האלגוריתם מבצע חיפוש עץ בסיסי , בוחר את הנתיב הטוב ביותר ומנסה לחשב דרכו את הדרך הטובה ביותר לנצח את המשחק, עד שנגמר לו הזמן. בסיום הזמן האלגוריתם מחזיר את התוצאה הכי טובה של הניחוש שבוצע. חסרון גדול באלגוריתם זה הוא שהניחוש אינו יהיה דומה בכל ריצה של המחשב, משמע הוא לא יהיה עקבי בהחלטותיו. האלגוריתם בוחר בכל פעם את הצומת הטובה ביותר באמצעות נוסחה ומבצע סימולצית מהלכים אקראית עד למצב של סיום המשחק. כאשר מגיע האלגוריתם אל סיום משחק הוא בודק האם השחקן שהפעיל אותו הוא המנצח או לא, את התוצאה יחזיר אחורה לכל אבותיו של הצומת שנחקרה ויתחיל את האלגוריתם מחדש.
* .minimax – אלגוריתם המאפשר לנו למצוא את המהלך הטוב ביותר עבורנו. האלגוריתם דואג למצוא בכל שלב במשחק את המהלך שמקבל את הציון המקסימלי (בתור שלנו) ואת הציון המינימלי (בתור היריב), כך ניתן למצוא את המהלך עם הציון הטוב ביותר ביחס למהלכים הכי טובים שהיריב יכול לבצע. בכל תור שנבצע (כעומק העץ שהגדרנו מראש) נדאג לבחור כל פעם ציון אחר ביחס לשחקן הנוכחי.
* pruning beta Alpha – אופטימיזציה של אלגוריתם minimax ,המטרה באלגוריתם זה הוא "לגזום" תתי עצים בעץ החיפוש שלנו. האלגוריתם מתייחס לשני משתנים נוספים בעת החיפוש בעץ alpha – משתנה המתאר את החסם העליון של המהלך הטוב ביותר שהשחקן יכול לבצע וbeta – משתנה המתאר את החסם העליון של המהלך הטוב ביותר שהיריב יכול לבצע. ברגע שalpha עובר את beta אנו יודעים שתת העץ שאנו בודקים לא יכול להביא תוצאה שתשנה את beta או alpha ולכן ניתן "לדלג" על חיפוש לאורך העץ הנתון. אלגוריתם זה במקרה הרע יפעל באותה סיבוכיות כמו minimax אך במקרה הכללי יקצר את הזמן משמעותית לחיפוש המהלך הטוב ביותר.

# הבעיה האלגוריתמית

הבעיה האלגוריתמית בפרויקט זה מתחלקת לשתי תת בעיות המתייחסות למקרים שונים: שחקן מול ושחקן או שחקן מול ממחשב.

1. הבעיה במימוש האלגוריתם לשני שחקנים היא שיהיה צורך לבדוק עם כל מהלך הוא מהלך חוקי (עומד בחוקי המשחק, לבדוק שהמקום שאליו אנחנו רוצים לזוז אינו תפוס. בנוסף על כך יש לבדוק כל תור אם אף אחד מהשחקנים לא העביר את כל הגולות שלו למשולש הנגדי ,כלומר ניצח במשחק.

כדי לפתור בעיה זו יש לממש את לוח המשחק בעזרת מבני נתונים שונים, לממש עצמים כמו חייל או מהלך כדי לשמור את נתוני המשחק, ולבסוף לבנות מערכת אשר תממש את מהלכי המשחק בצורה לוגית, בהתאם לחוקים ובצורה יעילה. בנוס

1. הבעיה האלגוריתמית במימוש האלגוריתם לשחקן ממוחשב מבטאת את בעיית הכרעת המצבים בלב הפרויקט. יש לבנות אלגוריתם לשחקן הממוחשב שימצא בכל תור את המהלך הטוב ביותר בעבורו על מנת להגיע לניצחון המהיר ביותר. על האלגוריתם לבחור את המהלך שבו הוא מניח את הגולה במקום האסטרטגי והמשתלם ביותר. על מנת להכריע איזה מהלך הוא הטוב ביותר יש לקחת לידי שקלול מספר רב של גורמים, לתת לכל אחד מהם משקל מסוים שישפיע על הבחירה הסופית ולבסוף לבצע את המצב אידיאלי. כאן מתבטאת בעיית הכרעת המצבים: יש לדרג ולהחליט איזה מהלך יהיה הכי אפקטיבי ואסטרטגי מבין המהלכים האפשריים.

# האלגוריתם הנבחר

## מבוא

האלגוריתם הסופי שאני בחרתי לממש בפרויקט הינו שילוב של אלגוריתם "Greedy" ותיאוריית "הנתיב הקצר ביותר" (Shortest Path).

האלגוריתם Greedy, כפי שהוזכר קודם לכן, פועל על ידי הערכת כל מהלך אפשרי ובחירת המהלך שנותן את התוצאה הטובה ביותר, על בסיס המצב המידי. בהתייחסות למשחק "דמקה סינית", ניתן להשתמש באלגוריתם זה כדי למצוא את המהלך שגורם לכך שרוב הכלים יוזזו קדימה לעבר בסיס המטרה בצורה המיטבית. עם זאת, כאמור, האלגוריתם Greedy אינו מתחשב בהשלכות העתידיות הפוטנציאליות של מהלך, ועובדה זו לעתים יכולה להוביל לתוצאות פחות אופטימליות.

כדי להתמודד עם מגבלה זו, שילבתי את תיאוריית הנתיב הקצר ביותר באלגוריתם שלי. תיאוריית הנתיב הקצר ביותר מציעה ששחקנים צריכים לשאוף להזיז את הכלים שלהם בדרך הקצרה ביותר האפשרית לצד של היריב שלהם בלוח, דבר שניתן להשיג על ידי שימוש בשילוב של קפיצה ומיקום אסטרטגי של חיילים. על ידי שילוב התיאוריה הזו באלגוריתם, ניתן להבטיח כי גם אם המהלך שנבחר ע"י האלגוריתם Greedy שאינו מתחשב בתוצאות העתידיות, עדיין קיימת אסטרטגיה כללית והתחשבות במצבים עתידיים.

השילוב של שני האלגוריתמים יתבצע בצורה הבאה: האלגוריתם Greedy יבצע את מה שהוא עושה כרגיל, יעבור על המהלכים האפשריים וייתן להם דירוג על בסיס המצב הקיים, ומהלכים אשר מממשים את תאוריית Shortest Path יקבלו דירוג גבוה בצורה משמעותית, כך תהיה אסטרטגיה אשר הן מתחשבת במצב הקיים לפי Greedy, והן שואפת למצב עתידי לפי תיאוריית הנתיב הקצר ביותר.

לסיכום, שילוב האלגוריתמים הפך לגישה פופולרית יותר ויותר לפתרון בעיות בתחומי מחקר רבים. בפרויקט שלי, בחרתי לשלב את האלגוריתם Greedy ותיאוריית הנתיב הקצר ביותר כדי ליצור אלגוריתם חדש לשחקן הממוחשב במשחק, אשר שלוקח בחשבון את ההשלכות המידיות והעתידיות של מהלך. על ידי שילוב שתי הגישות הללו, אני מאמין שהאלגוריתם שלי יכול לספק פתרון מדויק ויעיל יותר עבור שחקני המשחק.

## הסבר

### השלב הראשון - הערכת מצב המשחק

על בסיס סריקה של מיקומם של החיילים ניתן לקבוע באיזה מצב נמצא המשחק:

התחלה כאשר תרם כל החיילים עזבו את המשולש בסיס ההתחלה, אמצע כאשר מתבצעת תנועה של החיילים לעבר משולש בסיס המטרה (הבסיס הנגדי), וסוף כאשר רוב החיילים עברו את קו החצי של הלוח ומתבצעת הכנסה וסידור של החיילים בתוך בסיס המטרה. כאמור, המשחק נגמר כאשר כל החיילים נמצאים במשולש המטרה.

הערכת מצב המשחק חשובה מכיוון שהיא מאפשרת לשחקן לצבור מידע על הלוח, בהתאם להבנה שלנו היום על כיצד משחקים. בכך שהשחקן סורק את הלוח הוא רוכש מידע חשוב שעוזר לו בהמשך להכריע איזה מהלך לבחור.

כדי לממש סריקה יעילה של לוח המשחק בחרתי לשמור את נתוני החיילים של כל שחקן במבני נתונים של מילון: המפתח בנוי מביטוי מתמטי המורכב ממיקום החייל על הלוח והערך הוא אובייקט מסוג חייל אשר מכיל את הפרטים המפורטים של אותו חייל. כך אפשר להימנע מסריקה של הלוח כולו ולגשת לחיילים בצורה יעילה.

### השלב השני – מציאת מהלכים אפשריים

על מנת לבחור מהלך צריך ראשית לדעת איזה מהלכים ניתן לבצע. עבור כל חייל יש לחשב את המהלכים האפשריים שלו בהתאם לחוקי המשחק ולמצב הנוכחי של הלוח. יש לדאוג כי חישוב המהלכים יהיה עדכני בגלל שהלוח משתנה כל תור.

את מציאת המהלכים בחרתי לממש בצורה של רשימת מהלכים אפשריים, כלומר לכל שחקן תהיה פעולה שיוצרת רשימה של אובייקטים מסוג מהלך אשר כוללים את החייל שזז ולאיזה מיקום הוא זז.

חישוב המהלכים האפשריים יתבצע לפי רקורסיה ובעזרת מטריצת עזר שמייצגת את הלוח:

* עבור חייל מסוים נבדוק את כל המקומות סביבו ובמידה והם ריקים/חוקיים נוסיף אותם לרשימת המהלכים.
* נסמן את המיקום הנוכחי של החייל במטריצת העזר ב-1.
* עבור חייל מסוים נבדוק לכל כיוון אם קיים חייל שם ואם אחריו יש מקום ריק, במידה וכן נוסיף את המיקום לרשימת המהלכים ונקרא ברקורסיה לפעולה.
* במידה והגענו למקום שהיינו כבר/מקום לא חוקי יש לחזור צעד אחורה ברקורסיה.

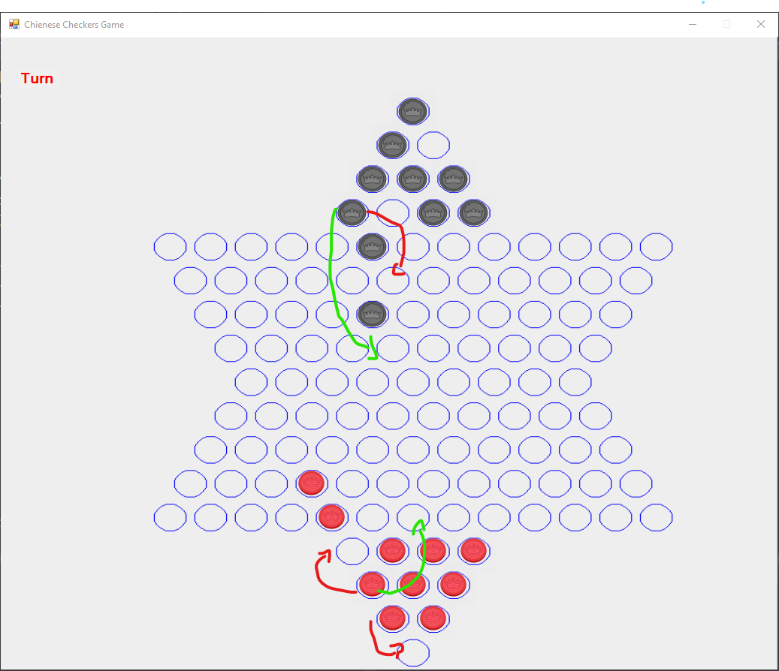
התוצר הסופי הוא רשימת כל המהלכים לחייל מסוים, את הרשימה הזאת נוסיף לרשימה שתייצג את רשימת

המהלכים לכל החיילים יחדיו, כך ניתן למצוא את כל המהלכים עבור חייל מסוים או כל החיילים של שחקן מסוים.

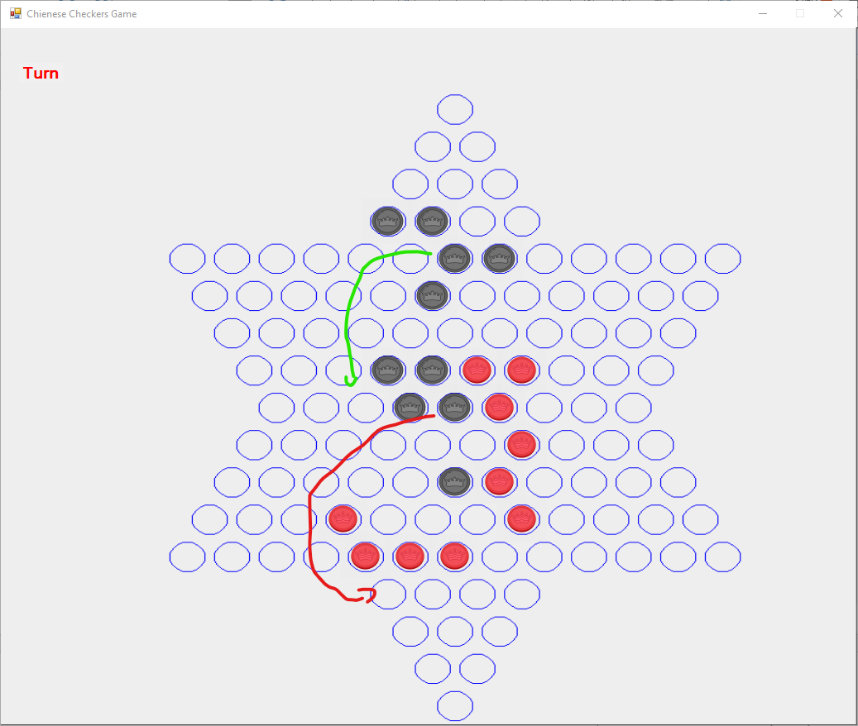
### השלב השלישי – בחירת המהלך הטוב ביותר

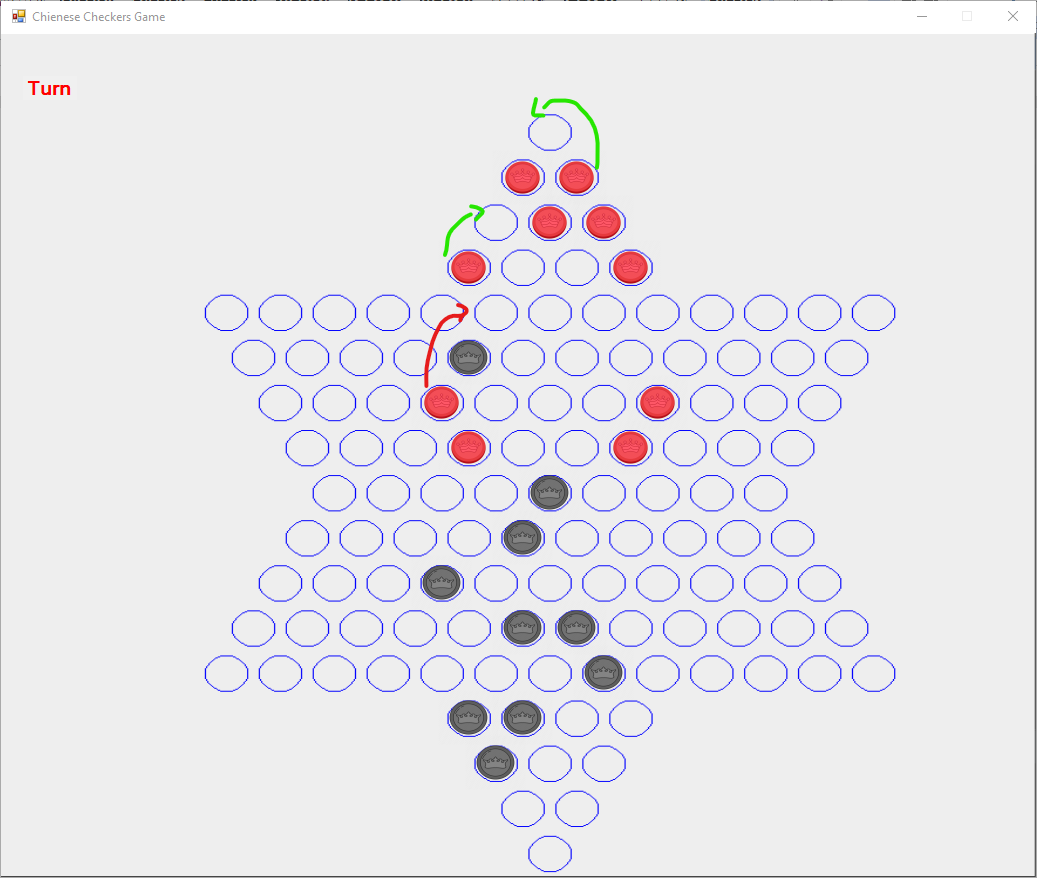
לאחר שיצרנו רשימת מהלכים אפשריים יש לבצע הכרעת מצבים ולקבוע את המהלך הטוב ביותר, וכפי שביססתי קודם לכן, האלגוריתם/אסטרטגיה שבחרתי לממש הוא שילוב של אלגוריתם "Greedy" ותיאוריית "הנתיב הקצר ביותר" (Shortest Path). אסטרטגיה זו תתממש בכך שבכל תור השחקן הממוחשב יעבור על כל המהלכים האפשריים שלו, ועל בסיס המצב הנוכחי של המשחק (greedy) , ייתן ציון/משקל לכל מהלך. הציון יקבע ע"פ התאמת המהלך לתאוריות Shortest Path וקריטריונים נוספים.

איזה קריטריונים מגדירים מהלך כמהלך טוב?

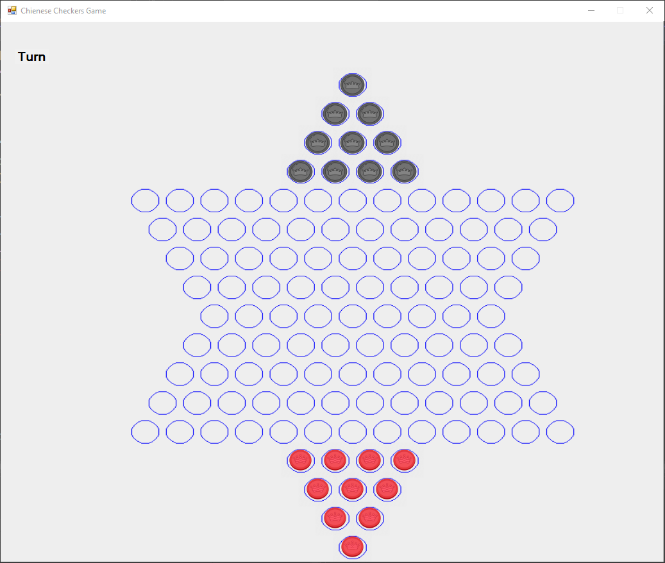
ראשית, מהלכים שמקדמים את החייל על עבר בסיס המטרה שלו, או לחלופין, לא מרחיקות את החייל ממטרתו. גם מהלכים שמוציאים את החייל מבסיס ההתחלה שלו נכנסים לקטגוריה זו, וככול שההוצאה היא יותר רחוקה כך לשחקן יש יותר יתרון. 

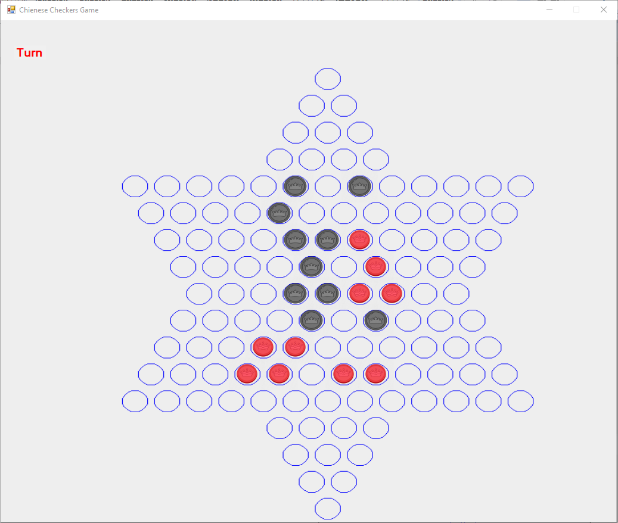
שנית, קפיצות רחוקות עבר בסיס המטרה ביחס לסטייה משער החיילים. על פניו ככל שהקפיצה רחוקה יותר המהלך טוב יותר, אומנם זה לא מדויק. אלא אם הקפיצה ממקמת את החייל בבסיס המטרה, יש להתחשב במיקום החיילים האחרים בעת קפיצה רחוקה וזאת כדי שלא ייווצר מצב בו חיילים נשארים מאחור ולא מתקדמים. קיימת עדיפות להוציא את כל החיילים מבסיס ההתחלה שלהם תרם הכנסה לבסיס המטרה ויש דגש על הזזה של חיילים שנשארים מאחור.

עוד על כן, ישנו דגש חשוב מאוד על הכנסה של חיילים על בסיס המטרה, בעזרת הכנסה כמה שיותר עמוקה. זאת אומרת שאם חייל יכול להכניס לשורה הראשונה של בסיס המטרה ולשורה השנייה, יש להעדיף את השורה השנייה. עוד על כן, בשלב האחרון של המשחק ישנו דגש על דחיסה של חיילים כמה שיותר עמוק בבסיס המטרה. כל זאת במטרה להקל על ההכנסה של שאר החיילים לבסיס המטרה ולמנוע חסימה.

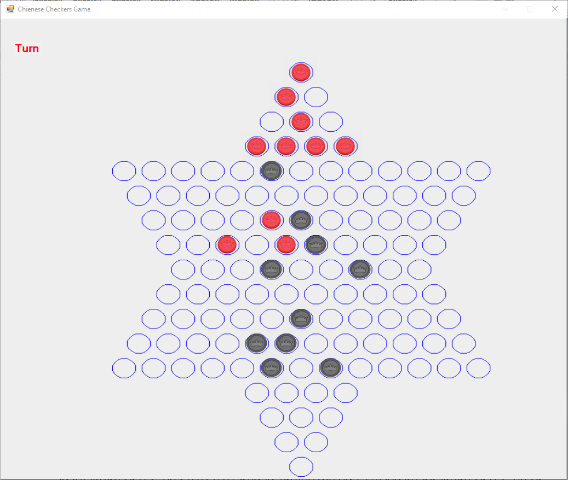


לפי הכרעת מצב המשחק, האלגוריתם בוחר אחד משלוש אסטרטגיות שונות המותאמות לכל אחד ממצבי המשחק.

Start - תחילת המשחק, הדגש הוא להוציא את החיילים מבסיס ההתחלה במינימום מהלכים ובמיקום אסטרטגי טוב. שלב זה רק חיילים שנמצאים בתוך בסיס ההתחלה יזוזו.

Middle - לאחר שכל החיילים עוזבים את בסיס התחלה מתחילה ההתקדמות עבר בסיס המטרה. כאן מתבטאים הדגשים שציינתי בנוגע מהלכים עדיפים ותזוזה כיחידה.

End – לאחר שכל החיילים עברו את קו החצי של הלוח על עבר בסיס המטרה מתחילים להתחשב במיקומם של החיילים בתוך בסיס המטרה, מתבצעת דחיסה, ויש דגש גדול על מהלכים שמובילים חיילים לבסיס המטרה.

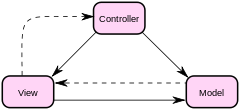


### השלב הרביעי – החלפת תור ובדיקת ניצחון

לאחר השחקן הממוחשב בוחר מהלך, הוא מבצע אותו. כעת צריך לבדוק אם אחד השחקנים ניצח. מתבצעת סריקה של החיילים של השחקן ששיחק אחרון, בין אם הוא ממוחשב או אנושי, ובמידה וכל החיילים של השחקן נמצאו בבסיס אויבו, הוא מוכרז כמנצח והמשחק נגמר. במידה ולא, מתבצעת החלפת תורים והאלגוריתם חוזר חלילה.

# ארכיטקטורת הפתרון

## ארכיטקטורת MVC (Model View Controller)

ארכיטקטורת Model-View-Controller (בקיצור MVC) היא ארכיטקטורת עיצוב בהנדסת תוכנה המשמשת להפשטת יישום כלשהו. התבנית מתארת טכניקה לחלוקת היישום לשלושה חלקים: "מודל", "תצוגה" ו"בקר", המחוברים ביניהם בצימוד רפוי מונחה אירועים. בדרך זו, התלות ההדדית בין ממשק המשתמש לשאר חלקי התוכנה פוחתת, ואת החלקים השונים ניתן לפתח באופן בלתי-תלוי.

## Top Down Level Design

### MODEL

מערכת הנתונים והלוגיקה הראשית של הפרויקט. יחידה זו מקבלת נתונים מהCONTROLLER ומבצעת עליהם פקודות בהתאם. ברוב המקרים היא גם מחזירה נתונים חזרה על הCONTROLLER.

* 1. STORE GAME DATA – מכילה את כל הנתונים, חיילים, מהלכים, הזזות, שחקנים והלוח. כלומר, מכילה את כל המידע על המשחק ואת המימוש שלו מאחורי הממשק.
  2. CHECK RULES – בודקת שכל ההזזות והמהלכים שהשחקנים מנסים לבצע הם מהלכים חוקיים בהתאם לחוקי המשחק. היא מאתחלת את המשחק במצבו ההתחלתי לפי החוקים, ומאפשרת שינויים בהתאם (או לא מאפשרת בהתאם).
  3. CHANGE DATA - המודל מקבל מידע על שינויים שהמשתמש רוצה לבצע. במידה והשינוי חוקי, המודל מבצע את השינוי מעדכן את כל מערכות המידע בהתאם.
  4. AI PLAYER - במקרה של שחקן אחד, המודל מדמה את השחקן השני באמצעות מחלקת שחקן מחשב. היא מממשת אלגוריתם שמחשב את המהלכים האפשריים באותו רגע ובוחר את המהלך הטוב ביותר.

### VIEW

מכיל את היחידה הגרפית האחראית על הצגת המידע של המודל בצורה וויזואלית (תפריט, לוח משחק, הזזות בזמן אמת). יחידה זו מקבלת INPUT מהמשתמש ומעבירה אותו לCONTROLLER. גם הCONTROLLER מעביר פקודות הצגת מידע לVIEW, כאשר הVIEW מבצע אותם, זהו הפלט למשתמש.

במערכת זו מוכל לוח המשחק, שעליו מצייר הController וגם התפריט הראשי שממנו המשתמש בוחר את מצב המשחק.

### CONTROLLER

היחידה המחברת בין הממשק למודל, אשר שולטת במהלך המשחק ומנהלת את מאזן "הקלט – פלט".

* 1. TURN – שולטת במעברת התורות בין השחקנים
  2. Send Input To Model, And Data To View

CONTROLLER מקבל קלט דרך הממשק ומעביר אותו אל המודל. המודל מבצע את החישוב/החיפוש ומחזיר נתונים חזרה על CONTROLLER .

CONTROLLER מציג את הנתונים שהוא מקבל מהמודל דרך הממשק. לדוגמה: השחקן רוצה להזיז חייל, CONTROLLER מקבל את הבקשה, מבקש מהמודל לבדוק אם זה אפשרי, אם כן אז הוא אומר לממשק לבצע את ההזזה על הלוח.

* 1. כפי שנאמר, הCONTROLLER מצייר לוח המשחק שממוקם בVIEW את השינויים שנעשו בMODEL.

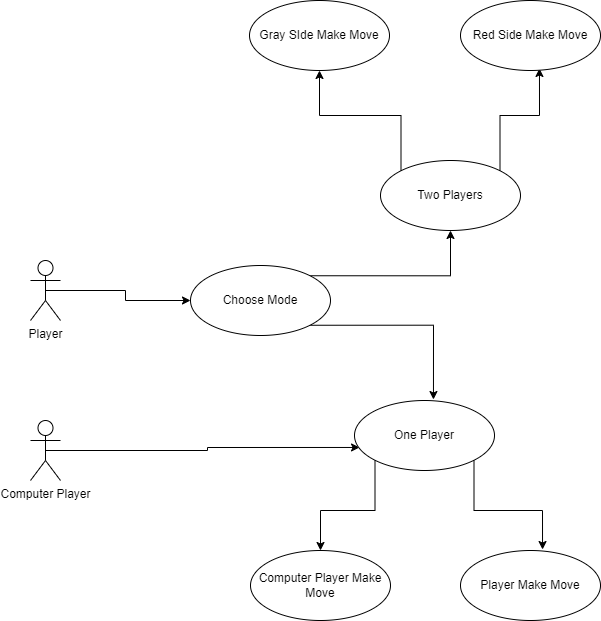
# Use Cases Diagram

תרשים Case Use מתאר את הקשר שקיים בין שחקנים שפועלים במערכת ל-Cases Use השונים.

כל Use Case מתאר סדרה של פעולות אשר מייצרות ערך עבור הלקוח שמשתמש במערכת.

בדיאגרמה הנ"ל תיארתי גם את פעולותיו של השחקן הא נושי וגם את פעו לותיו של השחקן הממוחשב

כמשתמש במערכת הכוללת



# מבני נתונים

1. מילון – לכל שחקן ישנו מילון שמאפשר גישה מהירה לכלי המשחק שלו. ישנם שני מילונים, אחת לכל שחקן, אשר מאחסנים בתוכם את הכלים, כאשר המפתח הוא נוסחה שכוללת את מיקומם הנוכחי על הלוח.

בנוסף על כך, לשחקן הממוחשב ישנם עוד 2 מילונים שמייצגים את הכלים בבסיס/משולש ההתחלה ומקומות בבסיס/משולש המטרה שאליו הוא שואף להגיע. להלן דוגמה למילון כמתואר:

Dictionary<int, Piece> pieces = new Dictionary<int, Piece>();

Dictionary<int, Piece> origins = new Dictionary<int, Piece>();

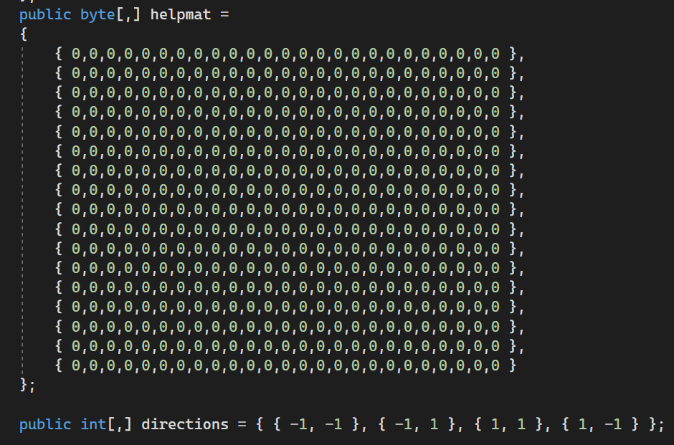
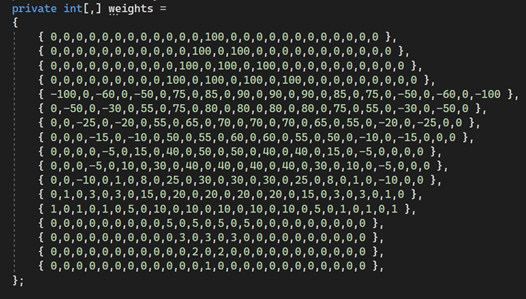
Dictionary<int, Piece> destinations = new Dictionary<int, Piece>();

1. רשימת מהלכים – לשחקן יש מספר פעולות יוצרות יחד רשימת מהלכים אפשריים עבור חייל מסוים שנבחר או כל החיילים. הרשימה מכילה אובייקטים מסוג "מהלך". בגלל שכל תור המהלכים האפשריים משתנים, הרשימה אינה תכונה של מחלקת שחקן, אלא תוצר של פעולות המחלקה.

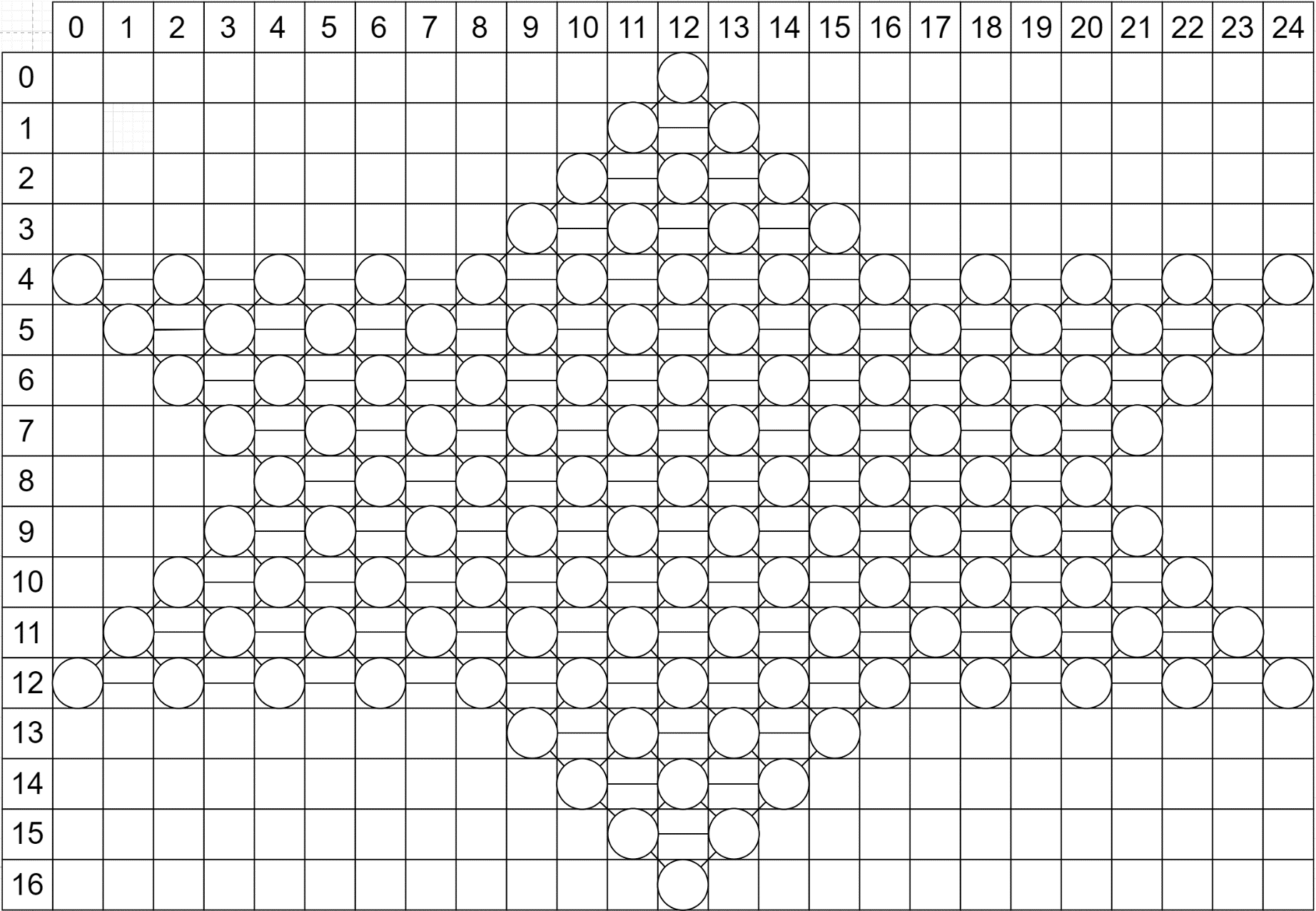
List<Move> moves = new List<Move>();



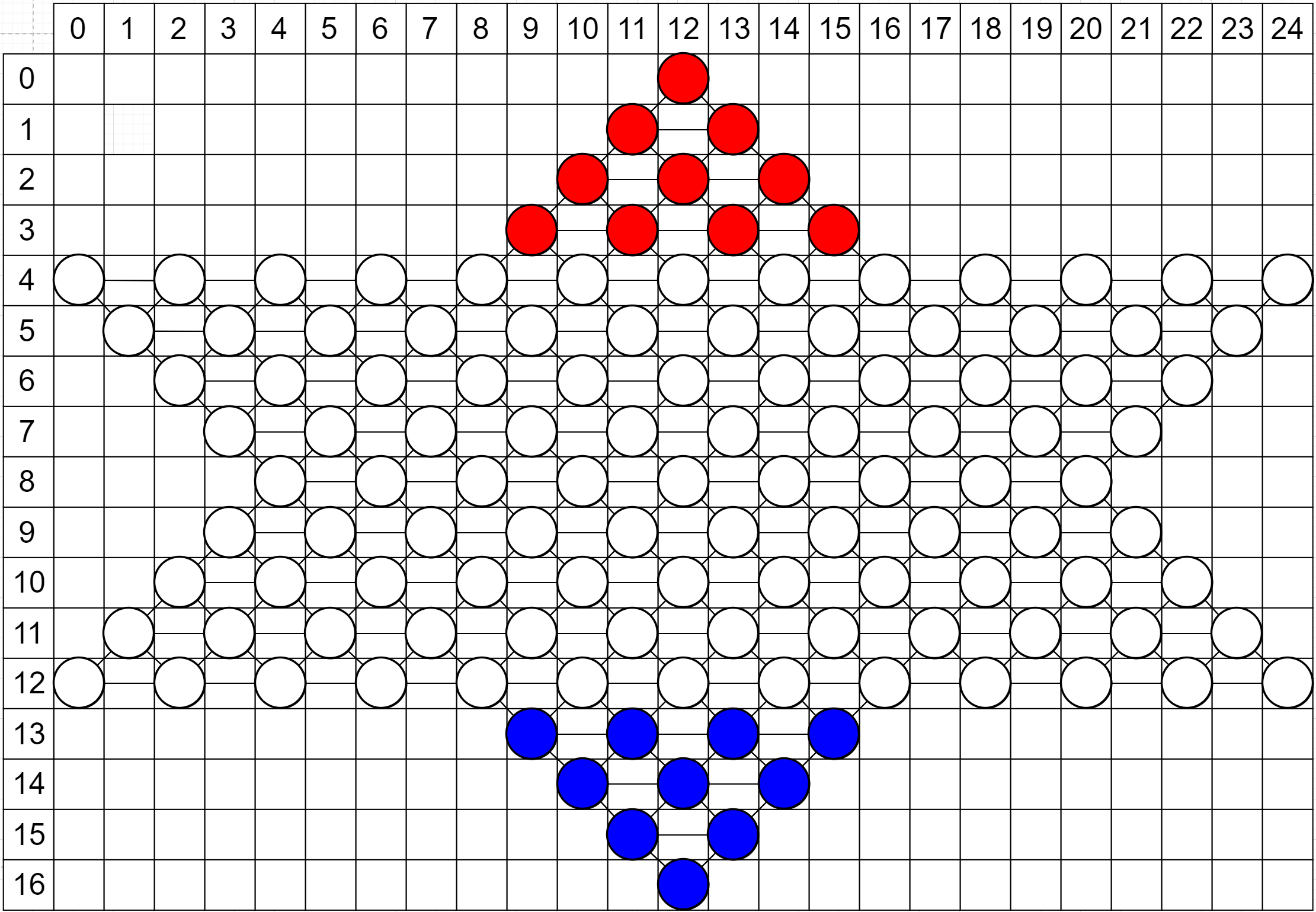
1. מטריצה- קיימות מטריצות שמטרתן לעזור לממש את לוח המשחק וחיפוש מהלכים אפשריים בו.

* מטריצת בתים: לאור העובדה שלוח משחק הוא בצורת כוכב, והשורות עצמן נבדלות באור ורווח, המטריצה הראשונה עוזרות לממש את הלוח המורכב בדרך שפשוט לעבוד איתו.
* מטריצת מעקב: תפקיד המטריצה השנייה היא לסמן מיקום מסוים על הלוח המורכב כאשר מתבצע חיפוש מהלכים אפשריים עבור חייל או חיילים. המטריצה מכילה כולה "0", מיקום שהחיפוש נעשה בו מסומן ב- "1". מאפסים אותה כל חיפוש.
* מטריצת כיוונים: מטריצה נוספת היא מטריצת עזר אשר קובעת את כיוון החיפוש של מהלכים אפשריים על הלוח. 
* מטריצת משקלים: המטריצה האחרונה בפרויקט היא מטריצה שמשקפת את מטרצת הלוח, רק שכל איבר בה בעל ערך שמסמל את המשקל של מהלך למקום זה. הערך הוא קבוע ומשמש נקודת מוצא למציאת המהלך עם מהשקל הכי טוב.

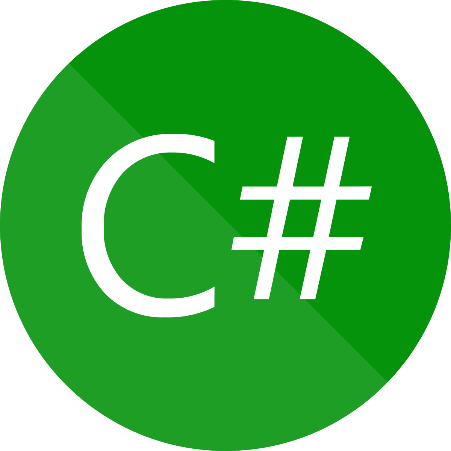
המטריצות יוצרת את הלוח הבא:



לאחר תיאום עם המילונים של מחלקו השחקנים, הלוח יראה כך:



# שפת התכנות וסביבת העבודה



C# היא שפת תכנות מונחת עצמים מודרנית שפותחה על ידי מיקרוסופט בתחילת שנות ה-2000. היא נמצא בשימוש נרחב בפיתוח משחקים, כאשר מנועי משחקים פופולריים רבים כמו Unity משתמשים ב-C# כשפת התכנות העיקרית שלהם. Microsoft Visual Studio היא סביבת פיתוח משולבת (IDE) פופולרית לבניית יישומים עם C#.

בחרתי לממש את הפרויקט שלי עם C# ו-Microsoft Visual Studio מכמה סיבות. ראשית, C# היא שפת תכנות חזקה ורב-תכליתית שקל ללמוד, עם תחביר נקי ופשוט. יכולת זיהוי השגיאות שלו והתמיכה בתכונות מתקדמות הופכות אותו לבחירה מצוינת לבניית משחקים מורכבים.

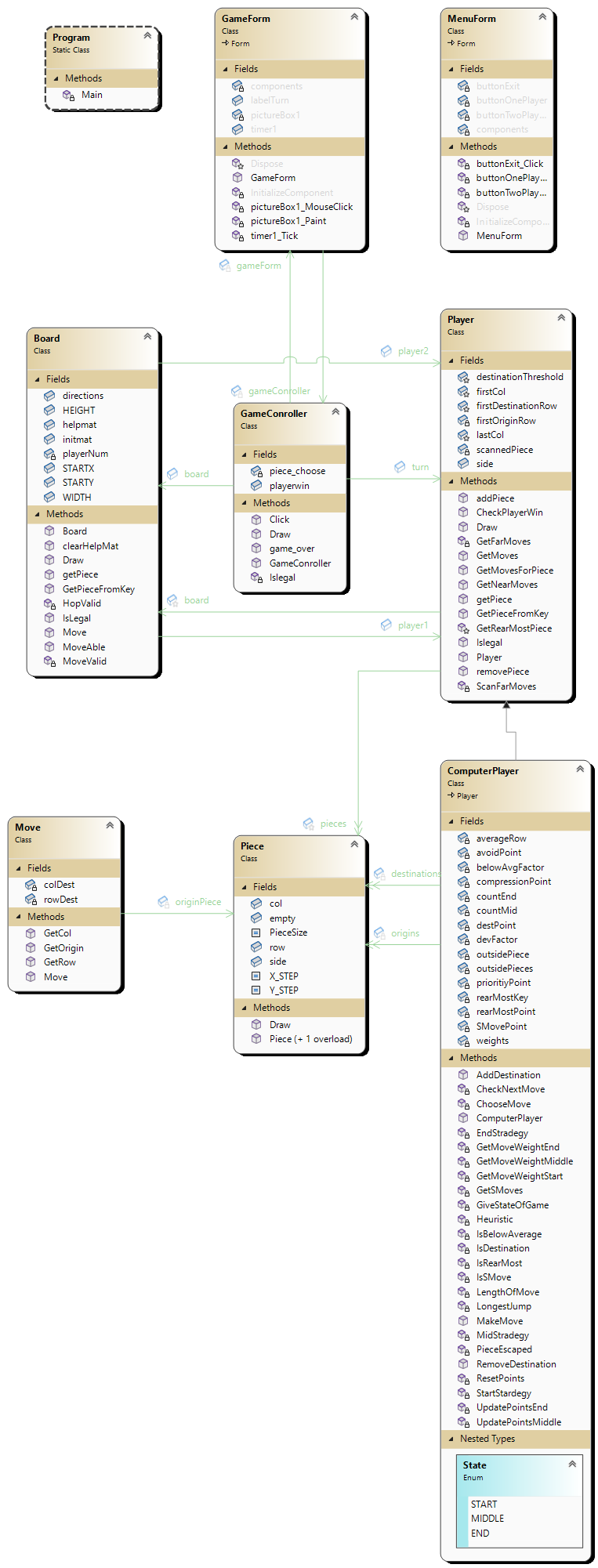
Microsoft Visual Studio הוא גם כלי נהדר לפיתוח משחקים, עם תכונות רבות שמקלות על בנייה וניפוי באגים בקוד. הכלים המשולבים באגים, עורך הקוד וכלי המעצב החזותי עוזרים להאיץ את תהליך הפיתוח, והשילוב שלו עם כלים אחרים של Microsoft כמו Azure ו-GitHub מקלים על ניהול התהליך.

לבסוף, בחרתי ב-C# וב-Microsoft Visual Studio מכיוון שהם נמצאים בשימוש נרחב בקהילת פיתוח המשחקים, עם משאבים ותמיכה רבים שזמינים באינטרנט. עובדה זו מקלה עלי למצוא עזרה כשאני נתקל בבעיות או צריך ללמוד משהו חדש, וגם עוזרת לי להבטיח שהמשחק שלי יהיה תואם לכלים וטכנולוגיות אחרות המשמשות בתעשייה.



# אלגוריתם ראשי

# תרשים מחלקות



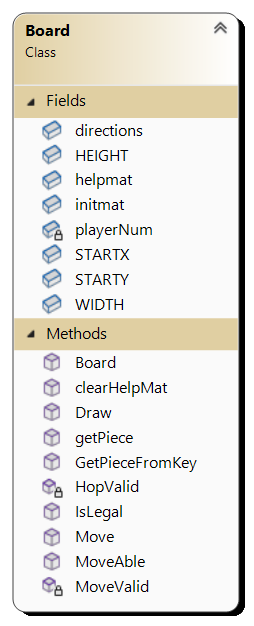
# תיאור מחלקות

## Game Controller

מחלקה ראשית אשר מאתחלת ושולטת במשחק עד סיומו. דואגת לטיפול בקלט משתמש מהממשק, שליחת נתונים למחלקות אחרות כמו Input, עדכון הממשק ובקשות ציור, שולטת במעבר התורות של המשחק וסיומו.

|  |  |
| --- | --- |
| פונקציות | תיאור פונקציות |
| public GameConroller(GameForm gameForm, int num) | פונקצית בנאי שמקבלת אובייקט מסוג GameForm ומספר שחקנים. הפונקציה מאתחלת את לוח המשחק עבור מספר השחקנים שנקלט, מאתחלת את ממשק המשחק ל **GameForm**שנקלט, ומאתחלת את התור הנוכחי של המשחק לתור של השחקן הראשון. |
| public void Click(int row, int col) | הפונקציה מקבלת מיקום של כלי מסוים, אם לא נבחר עוד חייל אז הפונקציה מסמנת את החייל שנקלט כהחייל שנבחר.  אם כן נבחר חייל, הפונקציה מבקשת מהמודל לבצע את המהלך. הפונקציה גם מעבירה את התור לשחקן הבא וסוגרת את המשחק במידה ואחד השחקנים ניצח. |
| public void Draw(Graphics graphics) | הפונקציה מקבלת אובייקט גרפיקה ומעבירה אותו למודל כדי לצייר את כל האובייקטים של המשחק.  בנוסף, הפונקציה מקיפה בעיגול ירוק את החייל שנבחר, ובעיגולים צהובים את כל המהלכים האפשריים של אותו חייל. |
| public void game\_over() | הפונקציה נקראת כאשר אחד השחקנים מנצח, היא מציגה הודעת ניצחון בהתאם למנצח וסוגרת את הממשק. |

## Board

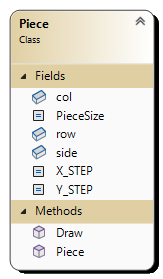
המחלקה הראשית של רכיב הMODEL , מכילה את רוב המידע והלוגיקה מאחורי המשחק, אחראית על פעולות הזזה של נתונים, שומרת על חוקיות המשחק בעזרת פעולות, ומאפשרת גישה למידע לשאר המחלקות והרכיבים.

|  |  |
| --- | --- |
| פונקציות | תיאור פונקציות |
| public Board(int totalNumberOfPlayers) | פונקציית בנאי שמקבלת מספר שחקנים ומאתחלת את השחקנים בהתאם. |
| public void Draw(Graphics graphics) | פונקציה שמקבלת אובייקט גראפי ומציירת את השחקנים שאיתחלה. |
| public Piece getPiece(int row, int col) | פונקציה שמקבלת מיקום של חייל, ובמידה והוא נמצא שם, היא מחזירה את החייל במיקום. |
| public Piece GetPieceFromKey(int key) | פונקציה שמקבלת מפתח של חייל, ובמידה והוא קיים אצל אחד השחקנים היא מחזירה אותו, אחרת Null. |
| public Player Move(Piece piece, int rowDest, int colDest) | פונקציה שמקבלת אובייקט מסוג חייל ומיקום מסוים על לוח המשחק. הפעולה בודקת אם אפשר לבצע הזזה של החייל למיקום, ובמידה וכן היא מבצעת את ההזזה. בנוסף על כך, במידה ואחד השחקנים ניצח, היא מחזירה את השחקן, אחרת Null. |
| private bool MoveValid(Piece piece, int rowDest, int colDest) | פונקציה שמקבלת אובייקט מסוג חייל ומיקום מסוים על לוח המשחק. הפעולה בודקת אם המיקום הוא מיקום פנוי וצמוד לחייל. מחזירה ערך בוליאני בהתאם. |
| public bool MoveAble(Piece piece, int rowDest, int colDest) | פונקציה שמקבלת אובייקט מסוג חייל ומיקום מסוים על לוח המשחק. הפעולה מחזירה האם ניתן לבצע הזזה של החייל למיקום. זאת פעולת הבדיקה הראשית של חוקיות הזזות. |
| public void clearHelpMat() | פונקציה שמאפסת את מטריצת העזר. |
| private bool IsLegal(Piece piece) | פונקציה שמקבלת אובייקט של חייל ובודקת האם המיקום שלו בלוח חוקי. |
| private bool HopValid(Piece piece, int rowDest, int colDest) | פונקציה רקורסיבית שמקבלת חייל ומיקום ובודקת האם ניתן לבצע קפיצות על הלוח ולהגיע למיקום היעד. |

## Piece

מחלקה שמייצגת את האובייקט Piece, חייל שנמצא על הלוח. התפקיד המרכזי של מחלקה זו הוא בעיקר לאכסן מידע על החייל בצורה שפשוט לגשת עליו.

|  |  |
| --- | --- |
| פונקציות | תיאור הפונקציות |
| public Piece(int row, int col, bool side) | פונקציה שמקבלת מיקום וצד, היא מאתחלת חייל עם הנתונים האלו. |
| public void Draw(Graphics graphics) | מציירת חייל במיקום שלו. |

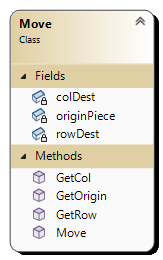


## Move

מחלקה שמייצגת את האובייקט Move, מהלך של חייל מסוים למיקום מסוים על הלוח.

המחלקה מכילה את האובייקט Piece שהוא החייל המקורי, ומיקום על הלוח שרוצים להגיע אליו.

פעולות המחלקה הן מאוד פשוטות, פעולות Get שמחזירות את הנתונים ששמורים (החייל המקורי והמיקום הרצוי).



|  |  |
| --- | --- |
| פונקציה | תיאור |
| public Move(Piece origin, int rowDest, int colDest) | פונקציית בנאי שמקבל אובייקט חייל מקור ומיקום (row ו- col) שאליו המהלך. כלומר הפונקציה מקבלת מיקום מקום ומיקום יעד. |
| public Piece GetOrigin() | הפונקציה מחזירה את piece המקור של המהלך. |
| public int GetRow() | הפונקציה מחזירה את שורת היעד של המהלך. |
| public int GetCol() | הפונקציה מחזירה את עמודת היעד של המהלך. |

## Player

מחלקה שמממשת שחקן במשחק, כוללת את מילון החיילים שלו, את הצד שלו ומופע של לוח המשחק על מנת לבצע בדיקות ופעולות.

למחלקה שלושה תפקידים מרכזיים: לשמור את המידע העדכני על מיקום החיילים של כל שחקן, לבצע חיפוש מהלכים אפשריים עבור החיילים של השחקן ולהוות תבנית שאותה תירוש מחלקת ComputerPlayer.

* המחלקה מבצעת את שמירת ועדכון החיילים בעזרת הפעולות addPiece ו – removePiece, כלומר מוסיפה למילון במיקום החדש ומוחקן במיקום הישן.
* את חיפוש המהלכים המחלקה מבצעת בעזרת רצף פקודות:

|  |  |
| --- | --- |
| פונקציות | תיאור פונקציות |
| public Player(bool side, Board board) | פונקצית בנאי שמקבלת צד ומופע של מחלקת לוח. בהתאם לצד, הפונקציה יוצרת ומאתחלת את החיילים של השחקן, את הצד שלו, ואת הלוח. |
| public void Draw(Graphics graphics) | הפונקציה מקבלת אובייקט גראפי וקוראת לפונקציית הציור בPiece עבור כל חייל במילון. |
| public Piece getPiece(int row, int col) | הפוקנציה מקבלת מיקום של חייל על הלוח, ובמידה וקיים חייל של השחקן על הלוח באותו מיקום, היא מחזירה אותו, אחרת null. |
| public void removePiece(Piece piece) | הפונקציה מקבלת אובייקט Piece ומוחקת אותו מהמילון |
| public void addPiece(int rowDest, int colDest, bool side) | הפונקציה מקבלת את כל הנתונים של חייל מסוים ומוסיפה אותו למילון. |
| public bool CheckPlayerWin() | הפונקציה עוברת על המקומות שעל השחקן למלא על מנת לנצח, ובמידה והוא ממלא אותם ועומד בתנאי הניצחון, היא מחזירה ערך בולאני בהתאם. |
| public List<Move> GetMoves() | פעולה ראשית שמחזירה רשימה מסוג Move של מהלכים קרובים ורחוקים של כל החיילים במילון השחקן. |
| private List<Move> GetNearMoves(Piece piece) | פעולה שיוצרת ומחזירה רשימה מסוג Move של מהלכים קרובים לחייל מסוים שהיא מקבלת. |
| private List<Move> GetFarMoves(Piece piece) | פעולה שיוצרת ומחזירה רשימה מסוג Move של מהלכים הרחוקים לחייל מסוים שהיא מקבלת. |
| private void ScanFarMoves(Piece Piece, List<Move> moves) | פעולה רקורסיבית שמקבלת רשימה ריקה של Move וחייל, הפונקציה מחפשת ברקורסיה את כל המהלכים הרחוקים ומוסיפה אותם לרשימה – פעולת עזר לפעולה הקודמת. |
| private bool Islegal(int row, int col) | פעולה שמחזירה האם המיקום שנקלט הוא חוקי בלוח. |
| public List<Move> GetMovesForPiece(Piece piece) | פעולה ראשית שמחזירה רשימה מסוג Move של מהלכים קרובים ורחוקים של חייל מסוים. |

## Computer Player

מחלקת השחקן הממוחשב שיורשת ממחלקת Player. התפקיד הראשי של המחלקה הוא לפתור את בעיית הכרעת המצבים עליה מבוסס הפרויקט, בעצם לדמות שחקן ממוחשב ולבצע מהלכים מחושבים ומותאמים למצב המשחק בעת התור.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |