quoridor ai



**מנחים:** אלון חיימוביץ'

**תאריך:** מאי 2022, תשפ"ב

**הדר לייבושור**

**ת.ז.** 209782002

**תוכן עניינים**

[תקציר 4](#_Toc101836599)

[תיאור הנושא 5](#_Toc101836600)

[חוקי המשחק 5](#_Toc101836601)

[תחילת המשחק 5](#_Toc101836602)

[תזוזת חיילים 5](#_Toc101836603)

[מיקום מחסומים 5](#_Toc101836604)

[פנים מול פנים 6](#_Toc101836605)

[מטרת המשחק 7](#_Toc101836606)

[רקע תאורטי 8](#_Toc101836607)

[מהו משחק אסטרטגי? 8](#_Toc101836608)

[מהו שחקן ממוחשב? 8](#_Toc101836609)

[אלגוריתם 8](#_Toc101836610)

[בינה מלאכותית 8](#_Toc101836611)

[אפליקציה 8](#_Toc101836612)

[תיאור הבעיה האלגוריתמית 9](#_Toc101836613)

[השרת 9](#_Toc101836614)

[פיתוח המשחק עצמו 9](#_Toc101836615)

[דוגמאות לבדיקות תקינות מהלכים 9](#_Toc101836616)

[ייצוג לוח המשחק 9](#_Toc101836617)

[יצירת מהלכים אפשריים 9](#_Toc101836618)

[קביעת המהלך הטוב ביותר 9](#_Toc101836619)

[סקירת אלגוריתמים בתחום הבעיה 10](#_Toc101836620)

[DFS 10](#_Toc101836621)

[MiniMax 10](#_Toc101836622)

[NegaMax 10](#_Toc101836623)

[Alpha Beta Pruning 10](#_Toc101836624)

[Transposition Table 11](#_Toc101836625)

[Iterative Deepening 11](#_Toc101836626)

[Killer Heuristic 11](#_Toc101836627)

[מושגים 12](#_Toc101836628)

[עץ משחק 12](#_Toc101836629)

[שחקן ממוחשב 12](#_Toc101836630)

[Heuristic Function 12](#_Toc101836631)

[אסטרטגיה 13](#_Toc101836632)

[מבנה נתונים 14](#_Toc101836633)

[תרשים מחלקות 15](#_Toc101836634)

[Use cases 15](#_Toc101836635)

[UML 15](#_Toc101836636)

[Top-Down Level Design 16](#_Toc101836637)

[תיאור סביבת העבודה ושפת התכנות 17](#_Toc101836638)

[שפת התכנות 17](#_Toc101836639)

[סביבת העבודה 17](#_Toc101836640)

[Operating System 17](#_Toc101836641)

[Code Editor 17](#_Toc101836642)

[JDK 17](#_Toc101836643)

[תיאור ממשקים 18](#_Toc101836644)

[org.glassfish 18](#_Toc101836645)

[javax.websocket 18](#_Toc101836646)

[com.google.gson 18](#_Toc101836647)

[Javax.Utils 18](#_Toc101836648)

[אלגוריתם ראשי 19](#_Toc101836649)

[פונקציות ראשיות 20](#_Toc101836650)

[מדריך למשתמש 21](#_Toc101836651)

[קוד הפרויקט 22](#_Toc101836652)

[GitHub 22](#_Toc101836653)

[קוד 22](#_Toc101836654)

[סיכום אישי 23](#_Toc101836655)

[ביבליוגרפיה 24](#_Toc101836656)

[נספחים 25](#_Toc101836657)

# **תקציר**

# **תיאור הנושא**

בכל תור השחקן יכול לבחור האם לזוז או האם לחסום.

## חוקי המשחק

כעת אסביר מהם חוקי המשחק, אילו מהלכים אפשריים ואילו לא אפשריים, וכיצד המשחק מתקדם.

### תחילת המשחק

A close-up of a keyboard

Description automatically generated with low confidenceהמשחק מתחיל כאשר שני שחקנים עומדים בקצוות שונים של הלוח באמצע השורה:

### תזוזת חיילים

A picture containing text, kitchen appliance, stove

Description automatically generatedשחקן יכול לזוז בתורו צעד אחד קדימה, אחורה, ימינה או שמאלה במידה והכיוון לא חסום לו:

### מיקום מחסומים

Diagram

Description automatically generatedכאשר שחקן מניח מחסום, על המחסום להפריד בין שני סטים של שני ריבועים:

Diagram, engineering drawing

Description automatically generatedאסור להניח מחסומים בדרך שלא יאפשר פתח יציאה של שחקן (לנעול שחקן):

### פנים מול פנים

Diagram

Description automatically generatedבמפגש "פנים מול פנים" בין שני השחקנים, כאשר אין מחסום המפריד ביניהם, שחקן שתורו יכול לקפוץ מעל אויבו בצורה הבאה:

A picture containing text, kitchen appliance

Description automatically generatedבמצב "פנים מול פנים", אם יש מחסום מאחורי האויב, שחקן יכול לקפוץ לצד האויב (כל עוד מחסום לא חוסם תנועה כזאת):

## מטרת המשחק

Diagram

Description automatically generatedהשחקן הראשון שמגיע לשורה שבו האויב התחיל ממנה מנצח, וכך גם נגמר המשחק:

# **רקע תאורטי**

## מהו משחק אסטרטגי?

משחק אסטרטגיה מושפט, הוא משחק המתאפיין בתכונות הבאות:

1. אין אלמנט של מזל, כמו למשל זריקת קובייה או בחירת קלף רנדומלי.

2. קיימת ידיעה מלאה של מצב המשחק בזמן אמת, כלומר בכל רגע כל השחקנים יודעים הכול על מצב המשחק.

3. המהלכים מתבצעים בזה אחר זה, לא כמו משחק אבן נייר ומספריים בו שני השחקנים צריכים להראות את ידיהם בו זמנית.

בהתאם לכך, גורל המשחק נקבע רק על פי ההחלטות של השחקנים ולפי החשיבה שלהם.

## מהו שחקן ממוחשב?

שחקן ממוחשב (בוט) הוא בעצם תוכנה שמבצעת פעולות אוטומטיות במשחק מחשב בהתאם לאדם המשחק מולו ובהתאם לאלגוריתם שנכתב לו. השחקן הממוחשב ישחק לפי חוקי המשחק המקוריים וינסה לחשוב על כל דרכי הפעולה המתאימות בשביל לנצח בזמן הקצר ביותר ובצורה היעילה ביותר.

## אלגוריתם

אלגוריתם הוא דרך שיטתית וחד-משמעית לביצוע של משימה מסוימת, במספר סופי של צעדים. אלגוריתם נועד לפתור בעיה או לבצע משימה על ידי מעקב אחר הוראות, אלגוריתם משמש לדוגמה עבור חישובים, קבלת החלטות, עיבוד מידע ועוד.

## בינה מלאכותית

מצב שבו מדמים את יכולת החשיבה של בן אדם על ידי אמצעים טכנולוגיים, בינה מלאכותית מבצעת פעולות שסיכויהם להשיג את מטרתם גבוהה במיוחד. מערכת מוגדרת כבינה מלאכותית אם היא מצליחה לזהות את מרכיבי סביבתה ומבצעת פעולות המשפרות את סיכויי השגת מטרתה באופן מקסימלי. לדוגמה: רכבים אוטונומים, המלצות לפי העדפות אישיות בתחומים שונים.

## אפליקציה

היא סוג של תוכנת מחשב אשר מנצלת את יכולות המחשב ישירות לביצוע משימות אותן המשתמש מבקש לבצע.

# **תיאור הבעיה האלגוריתמית**

הבעיה האלגוריתמית העיקרית בביצוע הפרויקט היא יצירת אלגוריתם נכון ויעיל שמסוגל לבצע החלטות חכמות בזמן אמת. חשוב שההחלטות יבוצעו בזמן קצר כדי שהמשתמש יהנה מהמשחק ולא ישב ויחכה למחשב שיבצע את מהלכו.

במהלך בניית הפרויקט נתקלתי במספר בעיות אלגוריתמיות נוספות לפני שהגעתי לבעיה העיקרית:

## פיתוח המשחק עצמו

למרות שהנושא המרכזי של הפרויקט הוא פיתוח שחקן המחשב, אך במהלך הפרויקט היה עליי לפתח את המשחק עצמו, משחק בו שחקנים אנושיים יכלו לשחק, ולאחר מכן גם שחקן המחשב. בחלק זה, היה עליי לפתח את משחק הקורידור באופן היעיל ביותר, בכדי שכאשר ששחקן המחשב יהיה מוכן, פרק הזמן שייקח לבצע פעולות הנוגעות למשחק עצמו. היה עלי להתחשב בבדיקות של תקינות המהלכים.

### דוגמאות לבדיקות תקינות מהלכים

* האם החומה מונח על חומה קיים
* האם הלוח

## ייצוג לוח המשחק

בעבודה על הפרויקט נדרשתי לבחור מבנה נתונים בכדי לייצג את הלוח, על המבנה להיות היעיל ביותר, ולאפשר לבצע את כל הפעולות הנדרשות הנוגעות בלוח המשחק בפרק זמן מינימלי, ובאופן היעיל ביותר. בחירת מבנה הנתונים של הלוח היא החלטה חשובה מאוד ותשפיע על כל המשך הפרויקט, מכיוון שכמעט כל הפעולות המרכזיות והחשובות ביותר בפרויקט מתבססות ומשתמשות בלוח.

## יצירת מהלכים אפשריים

במהלך העבודה על הפרויקט נדרשתי לחשב בכל תור את המהלכים האפשריים של כל שחקן, דבר זה היה אתגר מכיוון שבכל תור, שחקן יכול או להזיז את השחקן שלו, או לשים חומה (בכל מקום בלוח). נדרשתי לחשב את המהלכים האפשריים באופן יעיל, זאת מכיוון שזאת אחת הפעולות שבהן השחקן הממוחשב ישתמש באופן נפוץ.

## קביעת המהלך הטוב ביותר

# **סקירת אלגוריתמים בתחום הבעיה**

## DFS

אלגוריתם המאפשר לנו לעבור על קודקודי הגרף (במקרה שלנו עץ המשחק) לעומק, עד שמגיעים לעלה (קודקוד בעומק הרצוי שהגדרנו). כאשר נגיע לעלה, ננקד את הלוח באמצעות ה – heuristic function. נוכל לשפר את ביצועיו של אלגוריתם זה באמצעות beta alpha, minimax pruning ו table transposition שעליהם אסביר בהמשך.

## MiniMax

אלגוריתם המאפשר למצוא את המהלך הטוב ביותר עבורינו.

האלגוריתם בעצם מעין של "מדמה" את המשחק, כך שבכל שלב במשחק (בכל תור) מחזיר את הציון המקסימלי של המהלך אשר אנו יכולים לבצע, ואת הציון המינימלי עבור מהלכים שהיריב יכול לבצע, וכך בעצם מאפשר למצוא את המהלך הטוב ביותר, מספר צעדים קדימה, בהתייחסות למהלכים שיכול היריב לבצע.

האלגוריתם בעצם יוצר עץ משחק בגובה שהוגדר (העומק אותו אנו מגדירים בקריאה לפונקציה של האלגוריתם), עובר עליו, ומחזיר את המהלך ואת ניקוד המהלך הטוב ביותר אשר אנו יכולים לבצע בהסתכלות של מספר מהלכים קדימה (מספר זה הוא העומק שהגדרנו). אלגוריתם זה הוא כמובן רקורסיבי.

## NegaMax

אלגוריתם שהוא בעצם סוג של MiniMax.

אלגוריתם זה מחזיר את אותו הדבר כמו אלגוריתם ה - MiniMax אך עושה זאת עם קוד קצר יותר ומובן יותר לקריאה. זאת מכיוון שבאלגוריתם זה, במקום שתהיה בדיקה של מינימום ומקסימום, בכל פעם בוחרים את המהלך בעל הניקוד המקסימלי, ובקריאה הרקורסיבית קוראים למינוס אחד כפול הערך שיחזיר ה - NegaMax בעצם ניתן לקצר את הקוד כמעט פי 2 .זאת מכיוון שכאשר אני קורא לאלגוריתם עם מינוס לפני, ובוחר את המקסימום, אני בעצם בוחר את המינימום (מה שצריך לעשות כאשר התור הוא תור היריב, זאת מכיוון שכאשר היריב מבצע את המהלך המקסימלי, כלומר הטוב ביותר עבורו, זה בעצם המהלך הגרוע ביותר שיכל לבצע עבורי - משחק סכום אפס ולכן מה שטוב ליריב בטוח רע בשבילי). אלגוריתם זה בעצם זהה ברעיון לאלגוריתם ה -MiniMax ובאמצעותו ניתן למצוא את המהלך הטוב ביותר שניתן לבצע בהסתכלות לעץ המשחק בעומק מסוים, על ידי הסתכלות על המהלכים האפשריים אשר אני יכול לבצע בכל תור והמהלכים האפשריים אשר היריב יכול לבצע בכל תור.

## Alpha Beta Pruning

אופטימיזציה ושיפור של אלגוריתם ה – MiniMax / NegaMax על ידי "גזימת" תתי עצים.

בעצם ישנם מצבים בהם ניתן לדעת מראש שהמהלכים בתת עץ מסוים אינם רלוונטים ושהמהלך הטוב ביותר אשר אנו יכולים לבצע אינו נמצא בתתי עצים אלו. זה נעשה על ידי התייחסות לעוד שני פרמטרים, alpha ו – beta. כאשר alpha מתאר את החסם העליון (ניקוד המהלך הטוב ביותר של השחקן) של מהלך שהשחקן יכול לבצע, וbeta אשר מתאר את החסם העליון ביותר של מהלך שהיריב יכול לבצע. כאשר alpha גדול מ – beta ניתן לדעת כי תת העץ אשר אנו בודקים אינו יכול להביא תוצאה אשר תשנה את alpha או beta ולכן ניתן "לגזום" אותו, כלומר להפסיק לבדוק אותו.

אלגוריתם זה כמעט תמיד משפר משמעותית את הביצועים של ה - MiniMax או ה – NegaMax, ובמקרה הגרוע ביותר פועל באותה סיבוכיות כמוהם.

## Transposition Table

מבנה נתונים המאפשר לנו לשמור מעין cache של מצבים בלוח שכבר בדקנו, וכך בעצם לשפר את יעילות החיפוש של המהלך הטוב ביותר.

זאת מכיוון שכאשר אנו שומרים מצב של הלוח אותו אנחנו בודקים במעבר על עץ המשחק, ואנו מגיעים אליו פעם נוספת, כבר נדע מה המהלך הטוב ביותר לבצע במצב זה של הלוח ואת הscore של הלוח באותו מצב, ולא נצטרך לבצע את החישובים שוב, דבר אשר ישפר את היעילות באופן משמעותי.

באמצעות שמירת המהלכים נוכל גם באופן מידי לעדכן את הציון שניתן למצב הלוח ואת ערכי ה -alpha ו -beta הנוכחיים. עבור כל מצב שנגיע אליו בלוח והוא לא נמצא כבר ב – Transposition Table, נכניס אותו אליה, בכדי שפעם הבאה שניגע למצב כזה בלוח, לא נצטרך לחשב דברים מחדש.

מימוש ה – Tansposition Table נעשה באמצעות מילון, כאשר הערך הוא המהלך הטוב ביותר, הציון שלו, העומק בו עברתי על מצב לוח זה, ודגל המייצג את היחס של ציון המהלך ל – alpha ו – beta. המפתח של כל entry במילון הוא מפתח הנוצר כפונקציה של מצב לוח הנוכחי, ונקרא Zobrist Key. מפתח זה מיוצר על ידי נתינת ערך מספרי בגודל 64 ביט לכל משבצת על הלוח, לכל חייל על הלוח, ולכל אחד מהשחקנים, ולאחר מכאן ביצוע פעולת xor בין כל ערכי המשבצות (האקראיים) אשר עליהן יש חתיכות, עם ערכי החתיכות (האקראיים) ועם ערכי השחקן (הצבע של החתיכה, האקראיי). ביצוע פעולות xor אלו בין כל המקומות על הלוח בהן יש חתיכות משחק, מחזיר מספר, שהוא בעצם המפתח של מצב הלוח.

## Iterative Deepening

שיטה המאפשרת לנו לרוץ לעומק רב יותר, ומאפשרת לנו יותר גמישות בריצה לעומק במהלך המשחק.

השיטה עובדת כך, קובעים פרק זמן בו רוצים ששחקן המחשב יבצע את המהלך, וכל עוד פרק זמן זה עוד לא עבר, מריצים את פונקציית מציאת המהלך הטוב ביותר ובכל פעם מגדילים את עומק החיפוש עד שפרק הזמן עובר, כך בעצם ניתן להגיע לעומק מקסימלי, וניתן להיות גמישים, במצבים בהם ישנם מהלכים אפשריים רבים, נרוץ לעומק נמוך יותר בהשוואה לכאלה שבהם מספר המהלכים האפשריים רב יותר וניתן לרוץ לעומק רב יותר.

## Killer Heuristic

הרעיון מאחורי אלגוריתם זה, הוא שאם ישנו מהלך אשר גזם ענף של עץ ברמה מסוימת, ישנו סיכוי גבוה שיכול לגזום גם ענפים אחרים באותה הרמה, ובכך להקטין בעצם את כמות הענפים שאותם ממשיכים לבדוק, ובכך להקטין את מספר המהלכים אותם בודקים, דבר אשר יכול לשפר באופן משמעותי את זמן הריצה של אלגוריתם ה – Mini / Nega Max.

הרעיון הוא שאם מהלך מסוים במצב לוח מסוים גזם ענף, אותו מהלך יוכל לגזום ענפים נוספים באותו הרמה כאשר מצב הלוח דומה למצב בו המהלך גזם.

# **מושגים**

## עץ משחק

מושג בתיאוריית המשחקים, המשמש לתיאור גרף בו קיימים כל מצבי הלוח האפשריים במשחק, כאשר כל קודקוד הוא בעצם מצב לוח אפשרי, כאשר המסלול לכל קודקוד מהקודקוד הנוכחי, מייצג את המהלכים שיש לבצע בכדי להגיע למצב זה של הלוח.

ניתן להשתמש בעץ המשחק בכדי לחשב סיבוכיות של משחק, מכיוון שככל שישנם יותר מצבים אפשריים, המשחק יותר מסובך.

משתמשים בעץ המשחק בכדי למצוא את המהלך הטוב ביותר מנקודה מסוימת. האלגוריתמים Mini / Nega Max שעליהם הסברתי בסעיף הקודם, עוברים על עץ המשחק (לפעמים רק עד עומק מסוים, מכיוון שכמות המצבים עצומה והזמן שייקח לעבור על כולם רב מדי בשביל שהמחשב יוכל לחשב באופן ריאלי מהלך). כמות העלים בעץ משחק, היא כמות המצבים הסופיים האפשריים לסיום המשחק מהמצב הנוכחי. מספר הקודקודים בעץ המשחק הוא עצום, במשחק פשוט מאוד כמו איקס עיגול, כמות העלים היא 168,255 ולכן במשחקים כמו שחמט או שחמט סיני לא ניתן לעבור על כל המהלכים ועוברים על העץ רק עד לעומק מסוים.

## שחקן ממוחשב

שחקן AI אשר מחשב בכל תור את המהלך הטוב ביותר שיכול לבצע, ומבצע אותו. השחקן הממוחשב הוא Responsive. כלומר, בכל רגע מתייחס למה שקורה בלוח ומגיב בהתאם. השחקן הממוחשב חכם, ומסתכל קדימה מספר צעדים בכל תור בכדי לקרב אותו כמו שיותר לניצחון. שחקן ממוחשב יכול להיות בכל מני רמות. כיום במשחק השחמט, השחקנים הממוחשבים הטובים ביותר, טובים משחקני השחמט האנושיים הטובים ביותר.

## Heuristic Function

פונקציה המשומשת במשחקי מחשב בכי לתת ניקוד למצב המשחק בכל רגע נתון, ובאמצעות פרמטרים שונים שאליה מתייחסת, להעריך את מצב המשחק.

לרוב הפונקציה מחזירה מספר, שככל שהוא גבוה יותר, מצב הלוח של השחקן שעבורו מנקדת את הלוח, טוב יותר. לרוב משתמשים בפונקציה זו על עלים בעץ המשחק, כמו שעושים באלגורתמים כמו ה – Mini / Nega Max שעליהם הסברתי קודם.

ככל שלפונקציית ה - heuristic יותר פרמטרים והיא מתייחסת ליותר דברים על הלוח, היא מחזירה ציון מדויק יותר של הלוח.

# **אסטרטגיה**

# **מבנה נתונים**

## מבנים פשוטים

### List

רשימה היא [מבנה נתונים מופשט](https://he.wikipedia.org/wiki/%D7%9E%D7%91%D7%A0%D7%94_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D_%D7%9E%D7%95%D7%A4%D7%A9%D7%98) שתוכנו בעל [סדר חלקי](https://he.wikipedia.org/wiki/%D7%A1%D7%93%D7%A8_%D7%97%D7%9C%D7%A7%D7%99). מימוש של רשימה הוא למעשה ייצוג ממוחשב של [סדרה מתמטית](https://he.wikipedia.org/wiki/%D7%A1%D7%93%D7%A8%D7%94_(%D7%9E%D7%AA%D7%9E%D7%98%D7%99%D7%A7%D7%94)) סופית.

### HashMap

טבלת ערבול היא [מבנה נתונים](https://he.wikipedia.org/wiki/%D7%9E%D7%91%D7%A0%D7%94_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D) [מילוני](https://he.wikipedia.org/wiki/%D7%9E%D7%99%D7%9C%D7%95%D7%9F_(%D7%9E%D7%91%D7%A0%D7%94_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D)), אשר נותן גישה ל[רשומה](https://he.wikipedia.org/wiki/%D7%A8%D7%A9%D7%95%D7%9E%D7%94_(%D7%90%D7%97%D7%A1%D7%95%D7%9F_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D)) באמצעות [המפתח](https://he.wikipedia.org/w/index.php?title=%D7%9E%D7%A4%D7%AA%D7%97_(%D7%9E%D7%93%D7%A2%D7%99_%D7%94%D7%9E%D7%97%D7%A9%D7%91)&action=edit&redlink=1) המתאים לה. השתמשתי במבנה זה כדי לשמור אילו חומות יש במשחק. במידע זה משתמשים מספר פעמים בפונקציה הבודקת האם מהלך של חומה היא חוקית. הפונקציה הזאת נקראת מספר רב של פעמים במהלך חישוב המהלכים האפשריים. שימוש בטבלה תייעל את התהליכים האלו מפני שהגישה לערכים בעזרת המפתחות היא ב o(1).

### Queue

תור הוא מבנה נתונים מופשט. הוא פועל לפי עיקרון FIFO – הראשון שנכנס הוא הראשון שיוצא.

## מבנים מורכבים

### Graph

גרף היא מבנה נתונים לא ליניארי המורכב מצמתים וקשתות המקשרות ביניהם. בחרתי במבנה זה כמבנה עיקרי לפרויקט שלי, זאת מפני שיותר פשוט לייצג פעולות הקשורות לחומות. אם מתייחסים למיקומים בלוח כאל צמתים עם קשרים דו כיווניים למיקומים הסמוכים, פעולה של לשים חומה היא בעצם כמו ניתוק קשתות בין הצמתים.

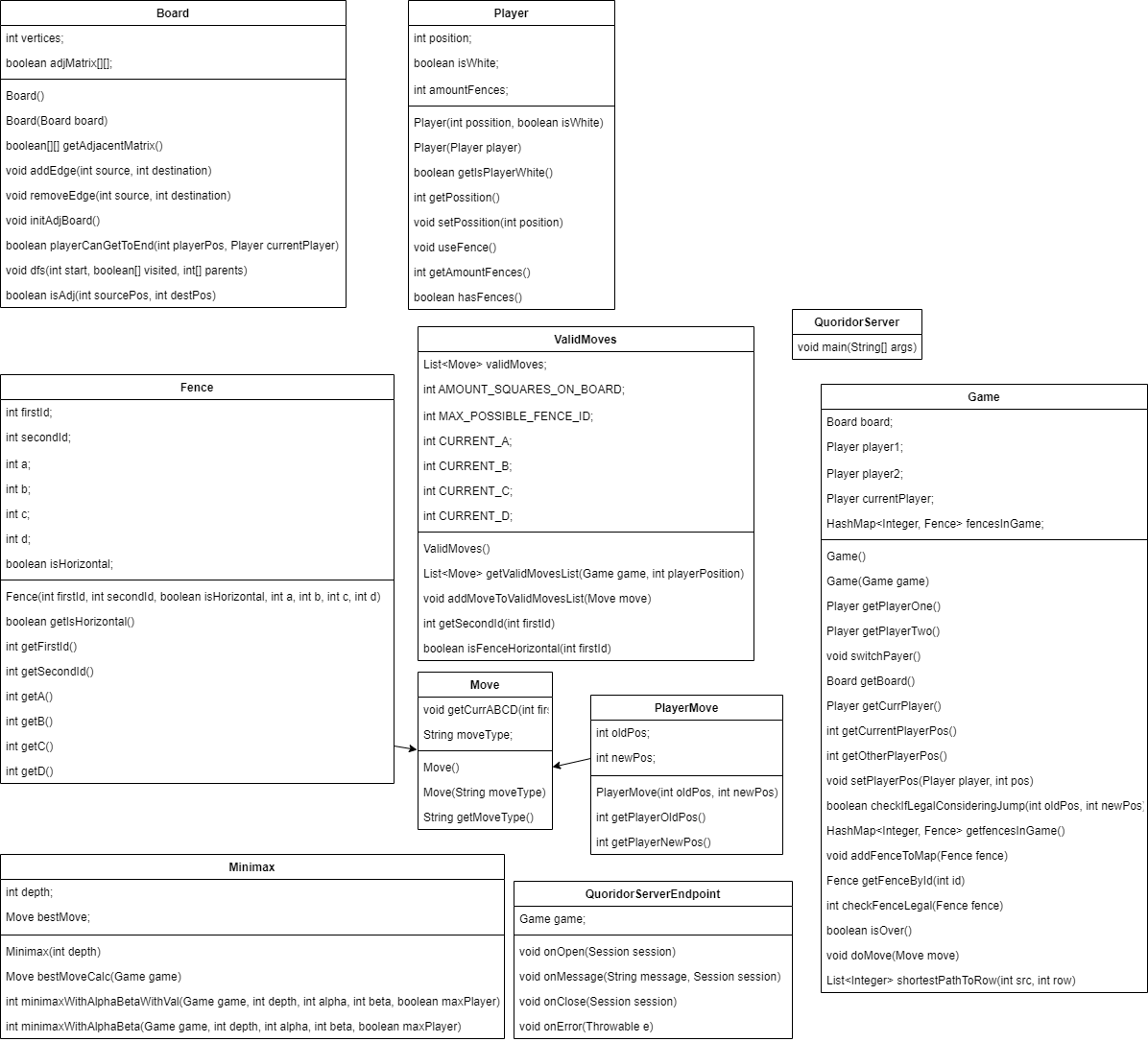
#### Adjacency matrix

השיטה שבחרתי לייצוג הגרף היא מטריצת סמיכויות. מטריצת סמיכויות  היא שיטה לייצוג [גרף מכוון](https://he.wikipedia.org/wiki/%D7%92%D7%A8%D7%A3_%D7%9E%D7%9B%D7%95%D7%95%D7%9F) בעל N צמתים בעזרת מטריצה ריבועית בגודל N על N שערכי תאיה הם 1 או 0. תא (i,j) בגרף מתאר את קיומה (או העדרה) של הקשת המכוונת מקודקוד i לקודקוד j בגרף. אם אין קשת כזו, הערך בתא במטריצה יהיה 0. אם יש קודקוד כזה, אזי הערך יהיה 1.

השתמשתי בשיטה זו כדי לייצג לאן שחקן יכול לקפוץ. הוספת חומה תשנה את הערכים המייצגים לקשר בין שני צמתים ל 0.

# **תרשים מחלקות**

## UML



# **Top-Down Level Design**

Show board state to player

Show errors if input is ilegal

Send input to server

GUI

Handle user input

Checks if game is over

Handles adjacency matric

Calculates possible moves

Checks that the move is legal

Logic

Finds best move for computer

AI

Server

Client

Quoridor

# **תיאור סביבת העבודה ושפת התכנות**

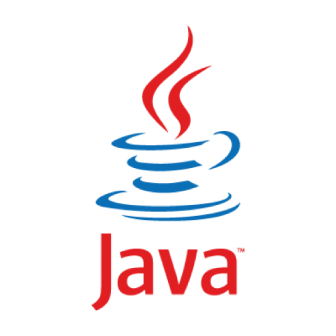
Logo, icon

Description automatically generated

## How to view JavaScript for a web pageשפת התכנות

בפרויקט השתמשתי במספר שפות תכנות.

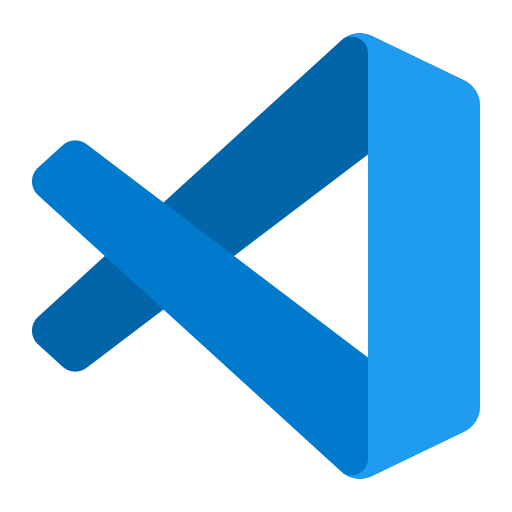
עבור הלקוח, ה – GUI, השתמשתי ב – HTML, CSS ו – JavaScripts.

עבור השרת והלוגיקה של המשחק השתמשתי ב – Java.

## A picture containing text, display Description automatically generatedסביבת העבודה

### Operating System

* Microsoft Windows 10



### Code Editor

* Visual Studio Code
  + version 1.66.2 (user setup)

### JDK

* jdk 17

# **תיאור ממשקים**

## org.glassfish

ממשק המאפשר יצירת שרת של Web sockets ב – Java.

## javax.websocket

ממשק המאפשר תקשורת באמצעות Web sockets ב – Java.

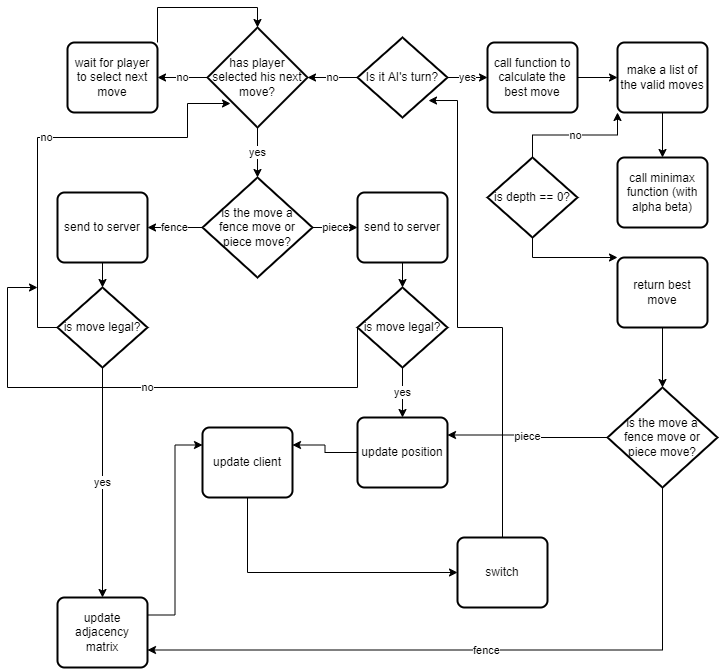
## com.google.gson

ממשק המאפשר ניהול ושימוש נוח של שליחה וקבלה של Json בצד השרת.

## Javax.Utils

ממשק מובנה ב java המספק גישה ל event model, date and time, collections ועוד. ממשק זה סיפק לי גישה למספר מבני נתונים פשוטים (רשימות, האשים ועוד).

# **אלגוריתם ראשי**



# **פונקציות ראשיות**

# **מדריך למשתמש**

כעת אסביר מהם השלבים הנדרשים על מנת שתוכלו להריץ את הפרויקט על מחשבכם:

1. וודאו שיש לכם JDK, בפיתוח הפרויקט השתמשתי בגרסא 17.

2. העתיקו את קוד הפרויקט למחשבכם מ – GitHub: <https://github.com/hadarLeib/Quoridor.git>

3. הריצו את הסקריפט server.bat ב – cmd על מנת להריץ את הסרבר.

4. פתחו את הקובץ index.html בדפדפן האהוב עליכם על מנת להתחיל לשחק!

וזהו 😀

# **קוד הפרויקט**

## GitHub

להלן קישור לקוד הפרויקט ב – GitHub: <https://github.com/hadarLeib/Quoridor.git>

## קוד

# **סיכום אישי**

## השרת

אמנם לשרת אין אלגוריתם מרכזי שהוא מממש, אך במהלך בנייתו נתקלתי במספר בעיות אלגוריתמיות שונות שהייתי צריכה לפתור, ולכן היה לי חשוב לציין כך.

אחת מן הבעיות הייתה,

# **ביבליוגרפיה**

<https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

# **נספחים**