quoridor ai



**מנחים:** אלון חיימוביץ'

**תאריך:** מאי 2022, תשפ"ב

**הדר לייבושור**

**ת.ז.** 209782002

**תוכן עניינים**

[תקציר 4](#_Toc102335847)

[תיאור הנושא 5](#_Toc102335848)

[חוקי המשחק 5](#_Toc102335849)

[תחילת המשחק 5](#_Toc102335850)

[תזוזת חיילים 5](#_Toc102335851)

[מיקום מחסומים 6](#_Toc102335852)

[פנים מול פנים 6](#_Toc102335853)

[מטרת המשחק 7](#_Toc102335854)

[רקע תאורטי 8](#_Toc102335855)

[מהו משחק אסטרטגי? 8](#_Toc102335856)

[מהו שחקן ממוחשב? 8](#_Toc102335857)

[אלגוריתם 8](#_Toc102335858)

[בינה מלאכותית 8](#_Toc102335859)

[אפליקציה 8](#_Toc102335860)

[מודל שרת לקוח 8](#_Toc102335861)

[תיאור הבעיה האלגוריתמית 9](#_Toc102335862)

[פיתוח המשחק עצמו 9](#_Toc102335863)

[דוגמאות לבדיקות תקינות מהלכים 9](#_Toc102335864)

[ייצוג לוח המשחק 9](#_Toc102335865)

[יצירת מהלכים אפשריים 9](#_Toc102335866)

[קביעת המהלך הטוב ביותר 9](#_Toc102335867)

[סקירת אלגוריתמים בתחום הבעיה 10](#_Toc102335868)

[DFS 10](#_Toc102335869)

[MiniMax 10](#_Toc102335870)

[Alpha Beta Pruning 10](#_Toc102335871)

[Transposition Table 10](#_Toc102335872)

[Killer Heuristic 11](#_Toc102335873)

[מושגים 12](#_Toc102335874)

[עץ משחק 12](#_Toc102335875)

[שחקן ממוחשב 12](#_Toc102335876)

[Heuristic Function 12](#_Toc102335877)

[אסטרטגיה 13](#_Toc102335878)

[תיאור ה Minimax 13](#_Toc102335879)

[תיאור ה Alpha Beta Pruning 13](#_Toc102335880)

[תיאור האסטרטגיה 13](#_Toc102335881)

[מבנה נתונים 14](#_Toc102335882)

[מבנים מורכבים 14](#_Toc102335883)

[Graph 14](#_Toc102335884)

[Adjacency matrix 14](#_Toc102335885)

[מבנים פשוטים 14](#_Toc102335886)

[List 14](#_Toc102335887)

[HashMap 14](#_Toc102335888)

[Queue 14](#_Toc102335889)

[תרשים מחלקות 15](#_Toc102335890)

[UML 15](#_Toc102335891)

[Use Case 16](#_Toc102335892)

[Top-Down Level Design 17](#_Toc102335893)

[דיאגרמה 17](#_Toc102335894)

[תיאור מילולי- צד שרת 18](#_Toc102335895)

[תיאור מילולי- צד לקוח 19](#_Toc102335896)

[תיאור סביבת העבודה ושפת התכנות 20](#_Toc102335897)

[שפת התכנות 20](#_Toc102335898)

[סביבת העבודה 20](#_Toc102335899)

[Operating System 20](#_Toc102335900)

[Code Editor 20](#_Toc102335901)

[JDK 20](#_Toc102335902)

[תיאור ממשקים 21](#_Toc102335903)

[org.glassfish 21](#_Toc102335904)

[javax.websocket 21](#_Toc102335905)

[com.google.gson 21](#_Toc102335906)

[Javax.Utils 21](#_Toc102335907)

[אלגוריתם ראשי 22](#_Toc102335908)

[פונקציות ראשיות 23](#_Toc102335909)

[מדריך למשתמש 24](#_Toc102335910)

[תכנית ראשית 26](#_Toc102335911)

[סיכום אישי ורפלקציה 27](#_Toc102335912)

[קוד הפרויקט 28](#_Toc102335913)

[GitHub 28](#_Toc102335914)

[קוד 28](#_Toc102335915)

[ביבליוגרפיה 30](#_Toc102335916)

[נספחים 31](#_Toc102335917)

# **תקציר**

הפרויקט עוסק במשחק הקופסה המוכר קורידור, ובנייתו כך ששחקן המשתמש יתמודד מול שחקן ממחושב- בינה מלאכותית.

כששמעתי שאנחנו צריכים לכתוב פרויקט ישר עלה לי המשחק קורידור. משחק זה הוא משחק ילדות שלי, שלמדתי לשחק בגיל צעיר ושיחקתי שנים רבות, ואף התחריתי בו, ולכן כשהתבקשנו ליצור פרויקט, ידעתי בוודאות שזה הפרויקט המתאים עבורי.

מטרות הפרויקט היא לאתגר את עצמי לבנות משחק, דבר שלא התעסקתי בו בעבר, ולא סתם משחק, משחק בעל סיבוכיות משחק גבוהה שיאתגר אותי, וידרוש ממני רמה גבוהה בהתאם. בנוסף לכך, הפרויקט נועד לתת לי טעימה מנושא הבינה המלאכותית, נותן לי הזדמנות להכיר ולחקור את האלגוריתמים השונים בנושא זה, שיודעים לדמות חשיבה של שחקן ממחושב לחשיבה של שחקן אמיתי, ולממשם.

יתר על כן, פרויקט זה נועד ליישם את החומרים שלמדתי בקורסים השונים במהלך השנה האחרונה: תכנות מונחי עצמים, מבני נתונים ויעילות אלגוריתמית.

התוצר הסופי של הפרויקט הוא בניית ארכיטקטורת שרת לקוח בעל ממשק נוח למשתמש שיאפשר למשתמש לשחק בקורידור מול בינה מלאכותית. בתיק פרויקט זה ניתן לקרוא על תוצר זה, ללמוד על משחק קורידור וחוקיו, ללמוד על ארכיטקטורת שרת לקוח, לראות הרחבה על האלגוריתמים השונים של שחקן ממוחשב ולחקור את נושא הבינה המלאכותית ככלל.

בנוסף על כל הנאמר, בתיק זה ניתן לקרוא גם על התהליך אותו עברתי ביצירת הפרויקט, הקשיים שליוו אותי, ההצלחות הקטנות שריגשו אותי, והחוויות שחוויתי במהלך כתיבת הפרויקט.

# **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.

## מודל שרת לקוח

[ארכיטקטורת תוכנה](https://he.wikipedia.org/wiki/%D7%90%D7%A8%D7%9B%D7%99%D7%98%D7%A7%D7%98%D7%95%D7%A8%D7%AA_%D7%AA%D7%95%D7%9B%D7%A0%D7%94) ל[חישוב מבוזר](https://he.wikipedia.org/wiki/%D7%97%D7%99%D7%A9%D7%95%D7%91_%D7%9E%D7%91%D7%95%D7%96%D7%A8), אשר מגדירה את היחס בין [תוכנות](https://he.wikipedia.org/wiki/%D7%AA%D7%95%D7%9B%D7%A0%D7%94) משתפות פעולה. המודל מחלק את המשימות או עומס העבודה בין ספק השירות או המשאבים (השרת), בין מבקש השירות (הלקוח). מודל שרת לקוח היא אחת מתצורות ההתקשרות הנפוצות ב[רשתות מחשבים](https://he.wikipedia.org/wiki/%D7%AA%D7%A7%D7%A9%D7%95%D7%A8%D7%AA_%D7%A0%D7%AA%D7%95%D7%A0%D7%99%D7%9D). השרת הוא תוכנה פסיבית, המאזינה לרשת ומחכה לקבל בקשות. הלקוח לעומתו בדרך כלל מהווה את [ממשק המשתמש](https://he.wikipedia.org/wiki/%D7%9E%D7%9E%D7%A9%D7%A7_%D7%9E%D7%A9%D7%AA%D7%9E%D7%A9) – הוא מופעל על ידי המשתמש ופונה לשרת כאשר הוא זקוק למידע או שירותים ממנו.

# **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 שכבר קיים
* האם החומה תמנע משחקן גישה מוחלטת לסוף
* קפיצה של שחקן מעל שחקן אחר

## 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

על מנת שהשחקן הממוחשב יוכל לשחק באופן הטוב ביותר, עליו לבחור בכל תור את המהלך הטוב ביותר שהוא יכול לבצע, זה שיוביל אותו לניצחון בסופו של דבר. כדי שזה יתאפשר, על המחשב להסתכל כמה צעדים קדימה, ולשקול את כל המהלכים האפשריים של השחקן הממוחשב וגם של היריב, ובכל פעם להניח שהיריב ישחק את המהלך הטוב ביותר שיכול לבצע. הבעיה והאתגר בכתיבת פונקציה זו היא שככל שמתקדמים לעומק, זאת אומרת, שכמות התורות שהמחשב יסתכל קדימה גדול יותר, מספר המהלכים שיש לנתח גדל באופן מעריכי. ניתן לראות כי כמות המהלכים שבודקת הפונקציה עצומה, ולכן בכדי שהבדיקה וקביעת המהלך הטוב ביותר תהיה מהירה ויעילה, על פונקציית ה"ניקוד" (heuristic function) להיות כמה שיותר יעילה.

# **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.

## Alpha Beta Pruning

אופטימיזציה ושיפור של אלגוריתם ה – MiniMax 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, 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..

## 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 – Minimax שעליהם הסברתי קודם.

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

# **אסטרטגיה**

## תיאור ה Minimax

אלגוריתם המאפשר לנו למצוא את המהלך הטוב ביותר עבורנו. האלגוריתם דואג למצוא בכל שלב במשחק את המהלך שמקבל את הציון המקסימלי (בתור של המחשב) ואת הציון המינימלי (בתור של השחקן האנושי).

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.

## תיאור ה Alpha Beta Pruning

g Beta Alpha היא אחת האופטימיזציות שהוספתי לאלגוריתם ה Minimax. אופטימיזציה זו משפרת את הביצועים ואת היעילות של ה Minimax בכך שבעצם "גוזמת" תתי עצים בעץ המשחק, מכיוון שיודעת שבתת עץ זה, מצב הלוח לא טוב יותר מהמצב הטוב ביותר שמצא האלגוריתם עד עכשיו. שיפור זה מתבצע באמצעות שליחת שני פרמטרים נוספים לפונקציית ה Minimax שהם alpha ו beta אשר alpha הוא החסם העליון של מהלך שיכול השחקן הממוחשב לבצע, בעוד ש beta הוא החסם העליון של מהלך שיכול היריב 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 ובמקרה הגרוע ביותר הביצועי נשארים זהים.

## תיאור האסטרטגיה

בתהליך חישוב המהלך הבא הטוב ביותר של המחשב, לא אוכל לעשות חישוב שמתחשבים בכלל המהלכים של המשחק עד לסיומו. הסיבוכיות עולה באופן מעריכי בכל התקדמות בעומק וחישוב כל תור בראייה עד סוף המשחק הוא גם לא יעיל וגם לא אפשרי במחשבים שלנו. לכן בחרתי לרדת רק לעומק שלוש על כל מהלך אפשרי ולהחזיר תשובה של ניקוד מכל עלה. המהלך שיוביל לתוצאת הניקוד הטוב ביותר, יהיה המהלך שנבחר לבצע.

החלטתי שהניקוד עצמו יקבע על ידי השוואת המרחק בין כל שחקן לסוף לשורת הניצחון שלו.

# **מבנה נתונים**

## 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.

## 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. השתמשתי במבנה זה כדי לייצג את רשימת המהלכים החוקיים וגם כדי לייצג את המסלול הקצר ביותר מנקודה מסוימת.

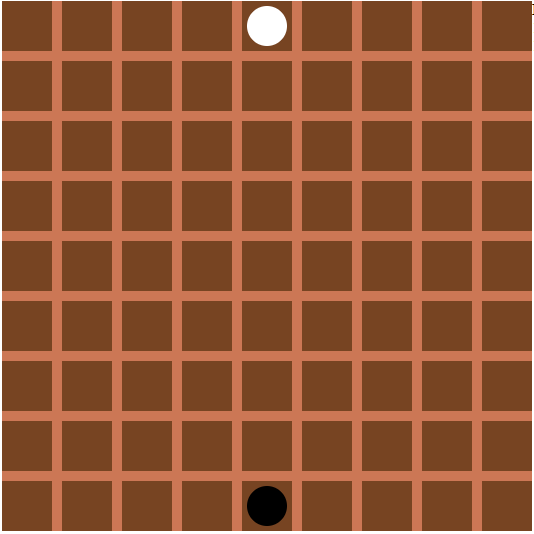
### HashMap

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. השתמשתי במבנה זה בפונקציה לחישוב המסלול הקצר ביותר. זהו מפני שהשתמשתי באלגוריתם BFS.

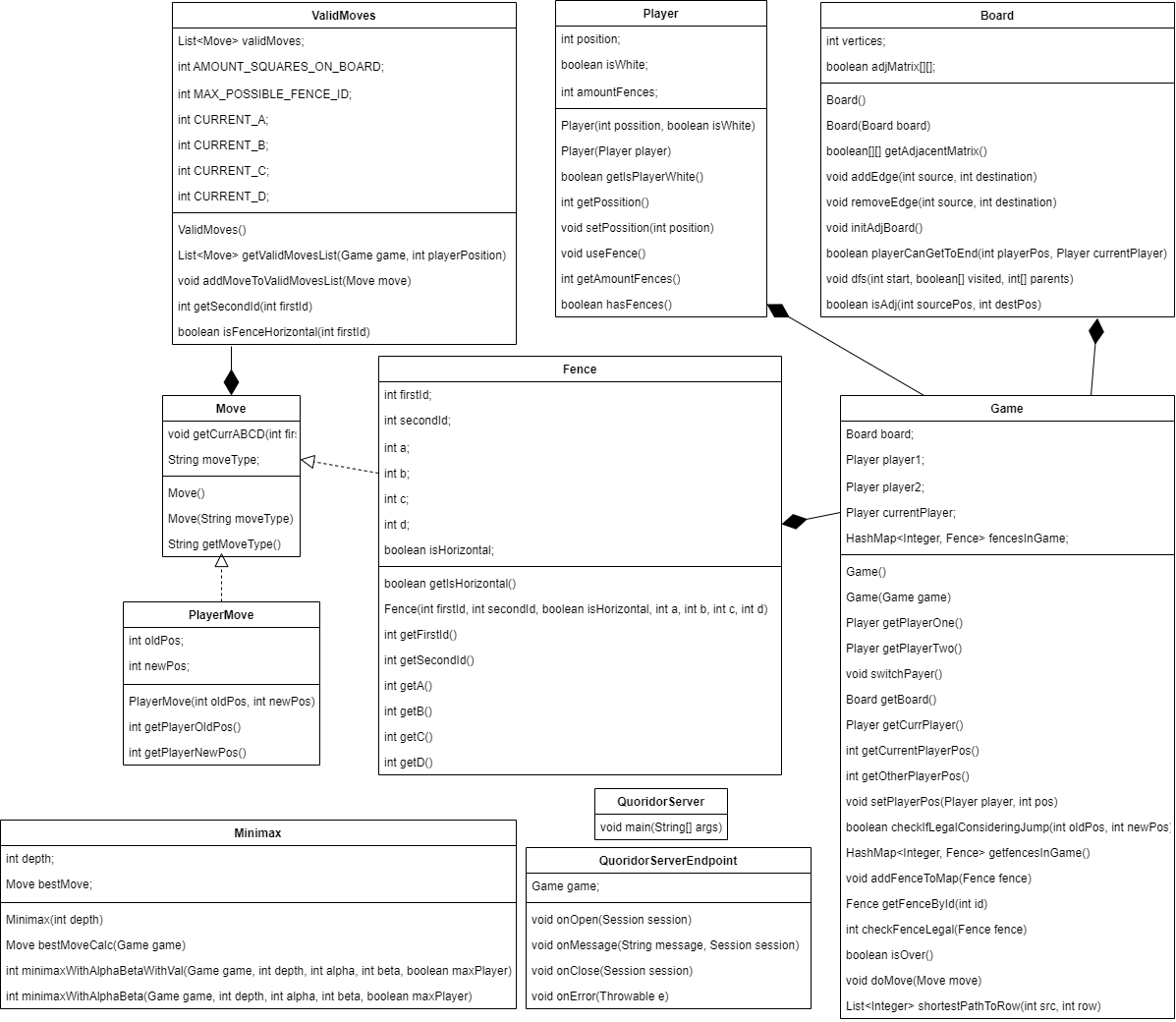
חשוב לציין כי את ה id של המיקומים בלוח את ה id של מיקומי החומות בלוח מספרתי בצורה הבאה:



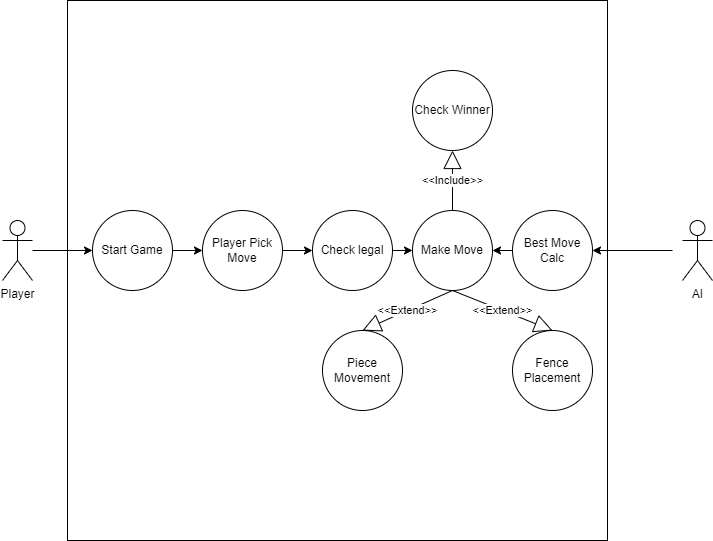
# 

# **תרשים מחלקות**

## UML



## Use Case



# **Top-Down Level Design**

## דיאגרמה

Get button clicks as moves

Show errors if input is illegal

Show board state to player

GUI

Handle user input

Sends a signal that it's the AI's turn to server

Sends AI move to client

Receives signal for AI to make it's move from client

Sends the result if user move is legal or not to client

Receives information about user's move from client

AI

Calculates possible moves

Handles adjacency matric

Checks if game is over

Logic

Checks that the move is legal

Finds best move for computer

Receives from server an answer is user's move legal

Sends user's move to server

Receives from server the AI's move

Client side socket

Server side socket

Server

Client

Quoridor

## תיאור מילולי- צד שרת

Quoridor

* Sever
* Server side socket
  + Receives information about user's move from client

מקבל מהצד לקוח את הפרטים לגבי איזה מהלך השחקן רוצה לעשות (כדי לבדוק בלוגיקה).

* + Receives signal for AI to make it's move from client

מקבל מהצד לקוח סימן שעכשיו הוא תור של המחשב לשחק.

* + Sends the result if user's move is legal or not to client

שולח מהצד לקוח האם המהלך שהשחקן האנושי היה חוקי או לא.

* + Sends AI move to client

שולח את המהלך של המחשב לצד לקוח.

* AI
* Finds best move for computer

תפקיד ה AI הוא לחשב מה המהלך הכי טוב למחשב לעשות.

* Logic
* Checks that the move is legal

בדיקה האם מהלך הוא חוקי או לא. משתמשים בלוגיקה זו כאשר בודקים האם מהלך שהמשתמש מנסה לעשות הוא חוקי וגם כאשר בונים את רשימת המהלכים החוקיים כחלק מהחישובים של ה AI.

* Calculates possible moves

חישוב רשימה של המהלכים האפשריים בכל מצב במשחק – לשימוש ה AI.

* Handles adjacency matrix

מטפל בעדכונים של מטריצת הסמיכויות של הגרף המייצג את הלוח. הוספת קשת/ מחיקת קשת.

* Checks if game is over

בדיקה של האם המשחק נגמר.

## תיאור מילולי- צד לקוח

Quoridor

* Client
* Client side socket
  + Receives from server an answer is user's move legal

מקבל מהצד שרת תשובה לגבי האם המהלך שהמשתמש ביצע הוא חוקי הוא לא.

* + Receives from server the AI's move

מקבל מהצד שרת את המהלך שהמחשב החליט לעשות.

* + Sends user's move to server

שולח לצד לקוח את המהלך שהמשתמש רוצה לבצע.

* + Sends a signal that it's the AI's turn to server

שולח לצד שרת שתור המחשב לשחק.

* Handle user input
* Get button clicks as moves

השחקן האנושי לוחץ על המהלך שרוצה לבצע באמצעות העכבר, זה מתורגם למהלך.

* GUI
  + Show board state to player

מראה לשחקן את מצב הלוך בזמן אמת.

* + Shows errors if user's input is illegal

אם שחקן עושה מהלך לא חוקי, יוצג הודעה למסך.

# **תיאור סביבת העבודה ושפת התכנות**

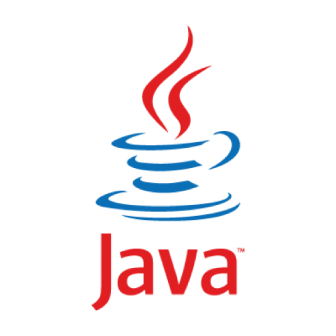
Logo, icon

Description automatically generated

## How to view JavaScript for a web pageשפת התכנות

בפרויקט השתמשתי במספר שפות תכנות.

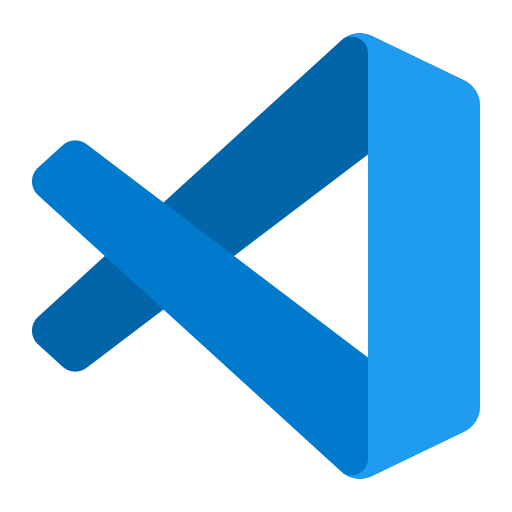
עבור הלקוח, ה – GUI, השתמשתי ב – HTML, CSS ו – JavaScripts.

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.

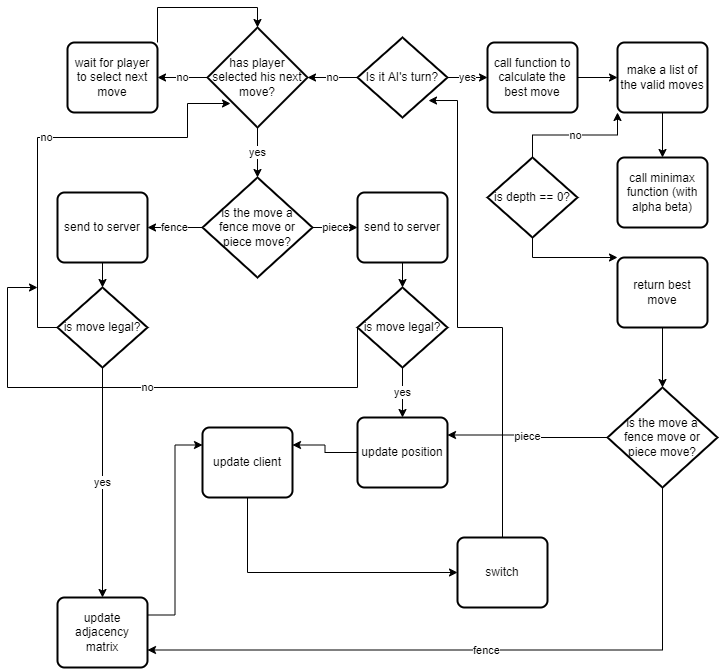
## com.google.gson

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 ומחלקות ראשיות**

## מחלקת Board

אחראית על עדכונים של המטריצת סמיכויות.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| כותרת הפונקציה | הפרמטרים של הפונקציה | איזה טיפוס הפונקציה מחזירה | מה הפונקציה מבצעת | יעילות הפונקציה |
| addEdge | Int source  Int destination | void | מוסיפה קשר במטריצה בין מיקום אחד לאחר (משני הצדדים) | O(1) |
| removeEdge | Int source  Int destination | void | מוחקת קשר במטריצה בין מיקום אחד לאחר (משני הצדדים) | O(1) |
| initAdjBoard |  | void | מאתחלת את המטריצה למצב תקין של תחילת המשחק | O(n)  N – מספר הצמתים בגרף |
| playerCanGetToEnd | Int playerPos  Player currentPlayer | boolean | מאתחל את המערך הורים ב (-1), מבצע dfs הממלא את מערך ההורים, בודק אם נמצא מסלול בין מיקום של שחקן לסוף | O(n)  N – מספר הצמתים בגרף |
| dfs | Int start  Boolean[] visited  Int[] parents | void | באמצעות האלגוריתם המוכר dfs, הפונקציה ממלאה את מערך ההורים. | O(n)  N – מספר הצמתים בגרף |

פסיאודו קוד dfs

1. התחל
2. Visited[start] <- true
3. עבור כל i שקטן מכמות הצמתים בגרף

3.1. אם (adjMatrix[start][i] == true) && (visited[i] == false)

3.1.1. start -> parents[i]

3.1.2. dfs(I, visited, parents)

פסיאודו קוד playerCanGetToEnd

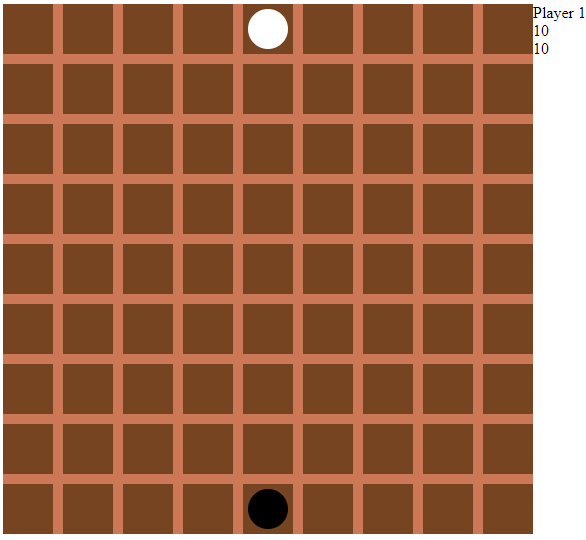
# **מדריך למשתמש**

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!

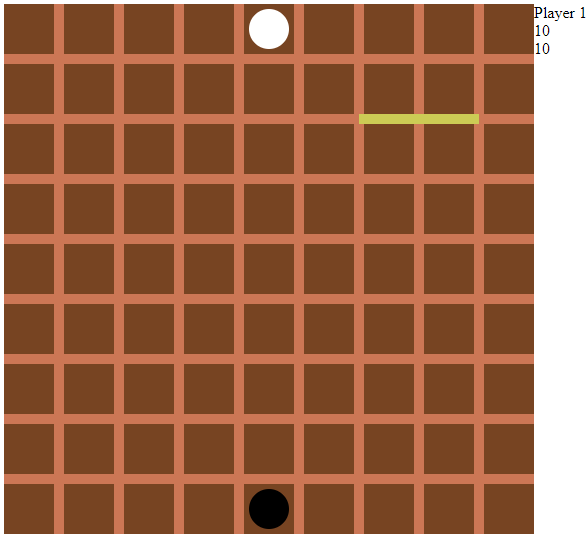
המשחק:

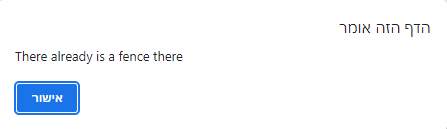
1. השחקן האנושי הוא השחקן הלבן (Player 1) והממוחשב הוא השחקן השחור (Player 2).
2. בצד ימין של הלוח יוצג תור איזה שחקן עכשיו וכמה חומות נשארו לכל שחקן.
3. כדי להתחיל את המשחק, לחצו על כפתור open מה שיתחיל את החיבור בין השרת ללקוח.
4. כדי לבצע מהלך של תזוזה של שחקן, על השחקן האנושי ללחוץ על הריבוע אליו ירצה לקפוץ בלוח:



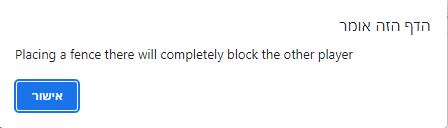
1. אם השחקן ינסה לקפוץ למקום אסור בלוח או מעל חומה, המסך הבא יעלה:



1. כדי לבצע מהלך של חומה, לחצו על איזור בין שני מיקומים בלוח:
2. הודעות שגיאה הקשורות לחומות:







# **C:\Users\משתמש\Downloads\main algorithm quoridor.drawio.pngתכנית ראשית**

# **סיכום אישי ורפלקציה**

כפי שתיארתי בתחילת תיק זה, קורידור הוא משחק ילדות שלי, ולכן פרויקט זה היה מין סגירת מעגל מיוחדת בה נהניתי לבנות את המשחק האהוב, ולפתח לו בינה מלאכותית.

בנוסף לכך, הפרויקט היה משמעותי עבורי מאחר ונחשפתי לעולם הבינה המלאכותית ולאלגוריתמים השונים שבו. למדתי לעומק על ארכיטקטורת שרת לקוח, המימושים השונים של מבני נתונים במשחק ובאלגוריתמים השונים ועוד.

כפי שכתבתי בראשית תיק זה, פרויקט זה הוא הפרויקט הראשון שבו עסקתי בתחום המשחקים והבינה המלאכותית, ולכן קשיים רבים היו סביב תחום זה. לא ידעתי איך להתחיל במימוש, או אפילו באיזה סביבת עבודה כדאי ליצור משחק שידמה את קורידור בצורה הממוחשבת, הכי מציאותית שיש. כך תהליך יצירת ה- GUI התארך, וכך גם תהליך בניית חוקי המשחק בלבד.

יתר על כן, היו לי התלבטויות רבות בבחירת האלגוריתם המתאים ביותר לשחקן הממוחשב, שכן זה עתה נחשפתי לתחום. היה לי קשה לקרוא את כמויות החומר שנדרשו, ולדעת מה רלוונטי לפרויקט שלי. מלבד זאת, נתקלתי בבעיות רבות בארכיטקטורת השרת לקוח, דבר שלרוב עצר את רצף העבודה עד אשר נפתרו התקלות.

עם זאת, כל החקר שביצעתי, בין עם בנושא השרת לקוח, השחקן הממוחשב, או האסטרטגיות השונות שניתן לנקוט בפרויקט זה, היה מרתק, נהניתי לקרוא וללמוד את הנושאים החדשים, ושמחתי על כל הצלחה. שמחתי על כל תיקון בעיה, מהקטנה שבקטנות כמו הצלחת התקשורת, או הגדולה שבגדולות, יצירת אלגוריתם מספק לפעולת השחקן הממוחשב. כל הצלחה גרמה לי לסיפוק אדיר ולתחושת גאווה שלשמה בסופו של יום, בחרתי במשחק כה מורכב.

קושי שהינו פחות רלוונטי לפרויקט עצמו, הוא כתיבת תיק זה. מאחר שאני עולה חדשה, כתיבת התיק בעברית הייתה עבורי קשה. קשה לי להתנסח ולבטא את החשיבה שלי בעברית וזה הקשה מאוד על תהליך הכתיבה ומציאת הניסוחים המתאימים שיסבירו בצורה הטובה ביותר את הפרויקט, אופן פעולתו ויצירתו.

חשוב לי לציין, כי אם הייתי מתחילה את הפרויקט היום, הייתי מוותרת על ארכיטקטורת שרת הלקוח, שכן היא מנעה ממני במשך זמן ממושך התקדמות בפרויקט, ואף לפעמים הורידה לי את המוטיבציה לעבוד עליו. עם זאת, אני שמחה כי המשכתי בארכיטקטורה זו, היא לימדה המון, גם על שרת לקוח וגם על עצמי, שגם כשקשה ומורכב, אסור לוותר וצריך להמשיך לעבוד.

# **ביבליוגרפיה**

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

# **קוד הפרויקט**

## GitHub

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

## קוד