quoridor ai



**מנחים:** אלון חיימוביץ'

**תאריך:** מאי 2022, תשפ"ב

**הדר לייבושור**

**ת.ז.** 209782002

**תוכן עניינים**

[תקציר 5](#_Toc102412256)

[תיאור הנושא 6](#_Toc102412257)

[חוקי המשחק 6](#_Toc102412258)

[תחילת המשחק 6](#_Toc102412259)

[תזוזת חיילים 6](#_Toc102412260)

[מיקום מחסומים 7](#_Toc102412261)

[פנים מול פנים 7](#_Toc102412262)

[מטרת המשחק 8](#_Toc102412263)

[רקע תאורטי 9](#_Toc102412264)

[מהו משחק אסטרטגי? 9](#_Toc102412265)

[מהו שחקן ממוחשב? 9](#_Toc102412266)

[אלגוריתם 9](#_Toc102412267)

[בינה מלאכותית 9](#_Toc102412268)

[אפליקציה 9](#_Toc102412269)

[מודל שרת לקוח 9](#_Toc102412270)

[תיאור הבעיה האלגוריתמית 10](#_Toc102412271)

[פיתוח המשחק עצמו 10](#_Toc102412272)

[דוגמאות לבדיקות תקינות מהלכים 10](#_Toc102412273)

[ייצוג לוח המשחק 10](#_Toc102412274)

[יצירת מהלכים אפשריים 10](#_Toc102412275)

[קביעת המהלך הטוב ביותר 10](#_Toc102412276)

[סקירת אלגוריתמים בתחום הבעיה 11](#_Toc102412277)

[DFS 11](#_Toc102412278)

[MiniMax 11](#_Toc102412279)

[Alpha Beta Pruning 11](#_Toc102412280)

[Transposition Table 11](#_Toc102412281)

[Killer Heuristic 12](#_Toc102412282)

[מושגים 13](#_Toc102412283)

[עץ משחק 13](#_Toc102412284)

[שחקן ממוחשב 13](#_Toc102412285)

[Heuristic Function 13](#_Toc102412286)

[אסטרטגיה 14](#_Toc102412287)

[תיאור ה Minimax 14](#_Toc102412288)

[תיאור ה Alpha Beta Pruning 14](#_Toc102412289)

[תיאור האסטרטגיה 14](#_Toc102412290)

[מבנה נתונים 15](#_Toc102412291)

[מבנים מורכבים 15](#_Toc102412292)

[Graph 15](#_Toc102412293)

[Adjacency matrix 15](#_Toc102412294)

[מבנים פשוטים 15](#_Toc102412295)

[List 15](#_Toc102412296)

[HashMap 15](#_Toc102412297)

[Queue 15](#_Toc102412298)

[16](#_Toc102412299)

[16](#_Toc102412300)

[תרשים מחלקות 17](#_Toc102412301)

[UML 17](#_Toc102412302)

[Use Case 18](#_Toc102412303)

[Top-Down Level Design 19](#_Toc102412304)

[דיאגרמה 19](#_Toc102412305)

[תיאור מילולי- צד שרת 20](#_Toc102412306)

[תיאור מילולי- צד לקוח 21](#_Toc102412307)

[תיאור סביבת העבודה ושפת התכנות 22](#_Toc102412308)

[שפת התכנות 22](#_Toc102412309)

[סביבת העבודה 22](#_Toc102412310)

[Operating System 22](#_Toc102412311)

[Code Editor 22](#_Toc102412312)

[JDK 22](#_Toc102412313)

[תיאור ממשקים 23](#_Toc102412314)

[org.glassfish 23](#_Toc102412315)

[javax.websocket 23](#_Toc102412316)

[com.google.gson 23](#_Toc102412317)

[Javax.Utils 23](#_Toc102412318)

[אלגוריתם ראשי 24](#_Toc102412319)

[פונקציות ומחלקות ראשיות 25](#_Toc102412320)

[מחלקת Board 25](#_Toc102412321)

[מחלקת Game 27](#_Toc102412322)

[מחלקת Minimax 31](#_Toc102412323)

[מחלקת validMoves 34](#_Toc102412324)

[מדריך למשתמש 35](#_Toc102412325)

[תכנית ראשית 37](#_Toc102412326)

[סיכום אישי ורפלקציה 38](#_Toc102412327)

[ביבליוגרפיה 39](#_Toc102412328)

[קוד הפרויקט 40](#_Toc102412329)

[GitHub 40](#_Toc102412330)

[קוד 40](#_Toc102412331)

[מחלקת Board 41](#_Toc102412332)

[מחלקת Fence 44](#_Toc102412333)

[מחלקת Game 46](#_Toc102412334)

[מחלקת Minimax 52](#_Toc102412335)

[מחלקת Move 56](#_Toc102412336)

[מחלקת PlayerMove 56](#_Toc102412337)

[מחלקת Player 57](#_Toc102412338)

[מחלקת ValidMoves 58](#_Toc102412339)

[מחלקת QuoridorServerEndpoint 61](#_Toc102412340)

[מחלקת QuoridorServer 65](#_Toc102412341)

[Sockets.js 66](#_Toc102412342)

[Ui.js 69](#_Toc102412343)

[Index.html 76](#_Toc102412344)

[Style.css 77](#_Toc102412345)

# **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.

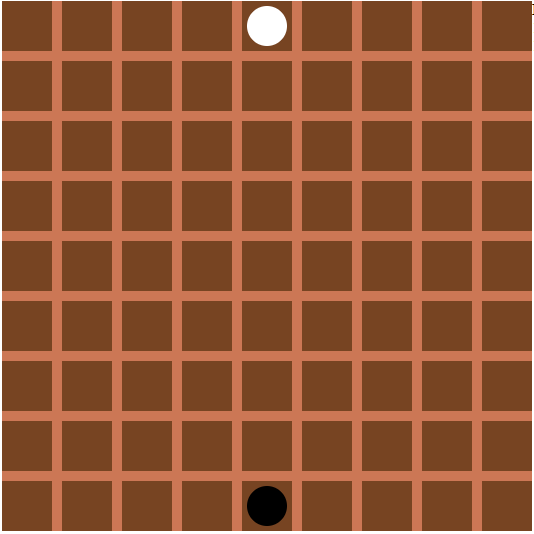
### HashMap

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:



0 1 2 3 4 5 6 7

8

7

6

5

3

0

2

1

# 

17 18 19 20 21 22 23 24

8 9 10 11 12 13 14 15 16

9 10 11 12 13 14 15 16 17

# 

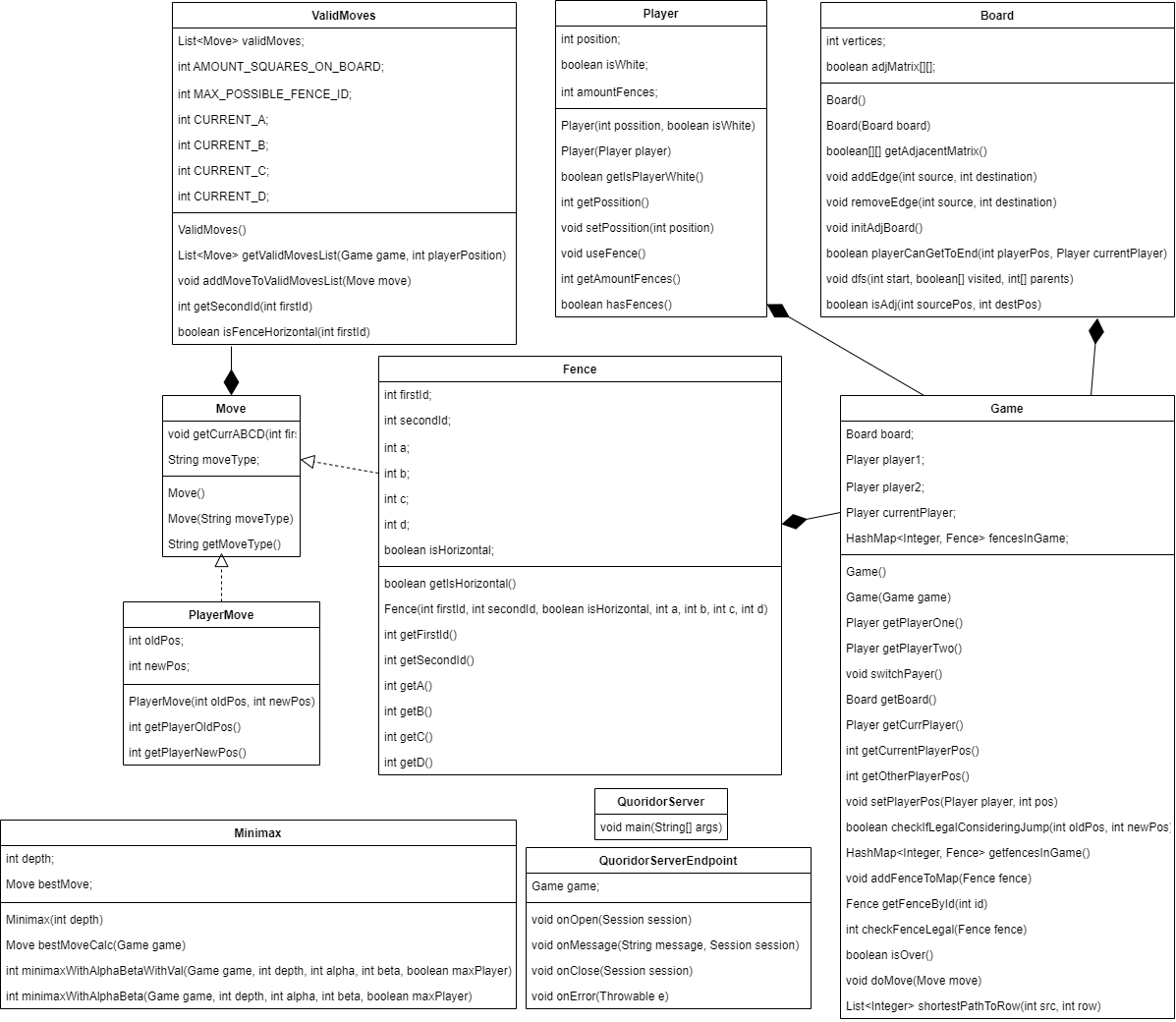
18 19 20 21 22 23 24 25 26

27 28 29 30 31 32 33 34 35

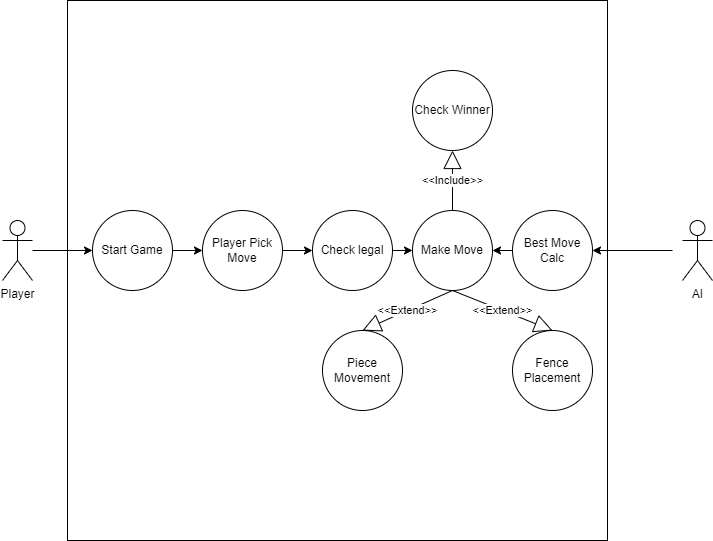
36 37 38 39 40 41 42 43 44

# **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**

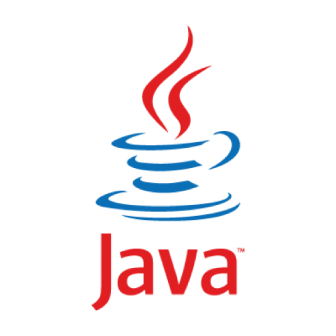
Logo, icon

Description automatically generated

## How to view JavaScript for a web pageשפת התכנות

בפרויקט השתמשתי במספר שפות תכנות.

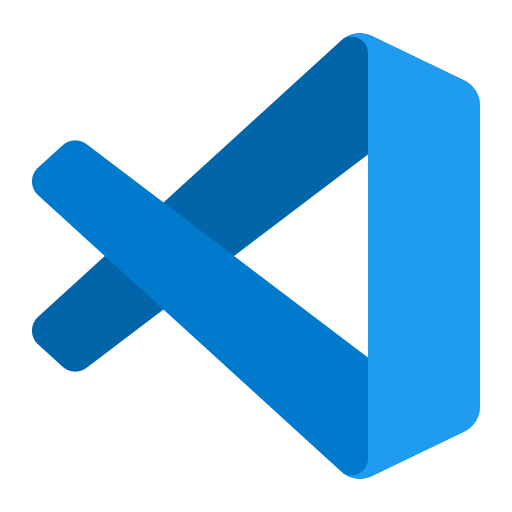
עבור הלקוח, ה – GUI, השתמשתי ב – HTML, CSS ו – JavaScripts.

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.

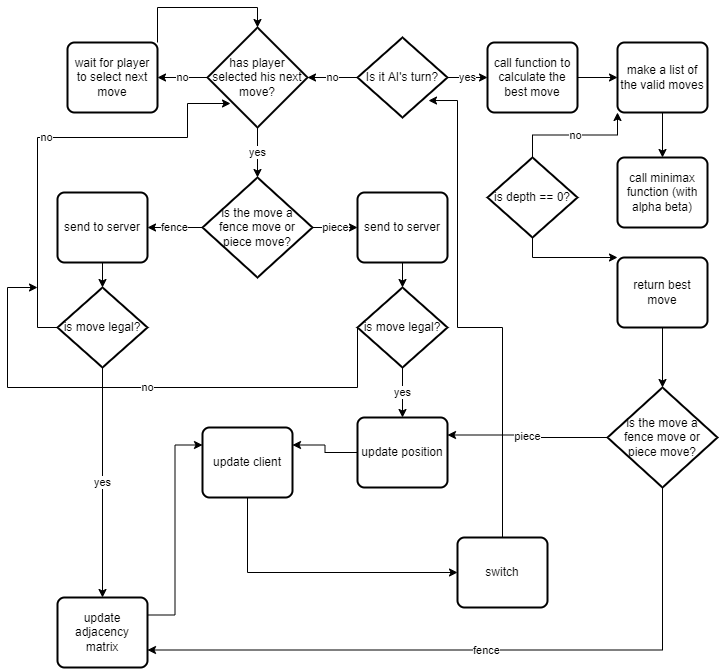
## com.google.gson

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
   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

1. התחל
2. אתחול:

boolean[] visited = new boolean[amount of vertices]

int[] parents = new int[amount of vertices]

1. עבור כל i שקטן מכמות הצמתים בגרף
   1. (-1) -> parents[i]
2. dfs(playerPos, visited, parents)
3. אם השחקן הנוכחי לבן
   1. 72 -> endPlayerPos
4. אחרת
   1. 0 -> endPlayerPos
5. עבור כל i שקטן מ 9:
   1. EndPlayerPos + i -> curPlayerPos
   2. כל עוד parents[curPlayerPos] != (-1)
      1. אם parents[curPlayerPos] == playerPos
         1. Return true
      2. Parents[curPlayePos] -> curPlayerPos
6. Return false

## מחלקת Game

אחראית על פונקציות הקשורות למהלך המשחק.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| כותרת הפונקציה | הפרמטרים של הפונקציה | איזה טיפוס הפונקציה מחזירה | מה הפונקציה מבצעת | יעילות הפונקציה |
| switchPayer |  | void | מחליפה את השחקן הנוכחי בשחקן האחר מבחינת מצב המשחק | O(1) |
| checkIfLegalConsideringJump | Int oldPos  Int newPos | boolean | בדיקה האם הזזה של השחקן היא חוקית | O(1) |
| checkFenceLegal | Fence fence | int | בדיקה האם הנחת לוח במיקום ספציפי היא חוקית. אם לא חוקי, תחזיר את סוג השגיאה | O(n)  N – כמות הצמתים בגרף |
| isOver |  | boolean | בודק האם המשחק נגמר לפי מיקומי השחקנים | O(1) |
| shortestPathToRow | Int src  Int row | List<Integer> | מחזירה רשימה של המסלול הקצר ביותר ממיקום כלשהו עד row | O(n)  N – כמות הצמתים בגרף |

Diagram

Description automatically generatedפסיאודו קוד checkIfLegalConsideringJump

הפונקציה בודקת מה ההפרש בין המיקום של השחקן הנוכחי לבין השחקן השני. לפי כך ניתן לזהות איפה השחקן הנוכחי ביחס לשחקן השני בלוח. אם נמצא כי השחקן הנוכחי צמוד לשני, ניצור קשת בין השחקן הנוכחי לבין המיקום בלוח שאחרי השחקן השני וננתק את הקשר בינו לבין המיקום בלוח שעליו השני עומד. לבסוף נבדוק אם המיקום הנוכחי של השחקן והמיקום אליו ירצה לקפוץ הם סמוכים בגרף.

1. התחל
2. אתחול:

Board boardCopy = new Board(this.getBoard())

int playerPosDif = getOtherPlayerPos() - getCurrentPlayerPos()

1. אם במקרה (playerPosDif == 1) //השחקן הנוכחי צמוד מצד שמאל
   1. boardCopy.removeEdge(oldPos, getOtherPlyerPos)
   2. אם (boardCopy.isAdj(getOtherPlayerPos(), (getOtherPlayerPos() + 1))) //אין חומה
      1. boardCopy.addEdge(oldPos, (getOtherPlayerPos() + 1))
2. אם במקרה (playerPosDif == -1) //השחקן הנוכחי צמוד מצד ימין
   1. boardCopy.removeEdge(oldPos, getOtherPlyerPos)
   2. אם (boardCopy.isAdj(getOtherPlayerPos(), (getOtherPlayerPos() - 1))) //אין חומה
      1. boardCopy.addEdge(oldPos, (getOtherPlayerPos() - 1))
3. אם במקרה (playerPosDif == 9) //השחקן הנוכחי צמוד מלמטה
   1. boardCopy.removeEdge(oldPos, getOtherPlyerPos)
   2. אם (boardCopy.isAdj(getOtherPlayerPos(), (getOtherPlayerPos() + 9))) //אין חומה
      1. boardCopy.addEdge(oldPos, (getOtherPlayerPos() + 9))
4. אם במקרה (playerPosDif == -9) //השחקן הנוכחי צמוד מעל
   1. boardCopy.removeEdge(oldPos, getOtherPlyerPos)
   2. אם (boardCopy.isAdj(getOtherPlayerPos(), (getOtherPlayerPos() - 9))) //אין חומה
      1. boardCopy.addEdge(oldPos, (getOtherPlayerPos() - 9))
5. Return boardCopy.isAdj(oldPos, newPos)

פסיאודו קוד checkFenceLegal

מחזירה אם החומה היא חוקית או לא ומחזירה גם את סוג השגיאה אם לא

תחזיר 1 אם חוקי, 2 אם הנחת חומה על חומה קיימת, 3 אם חומה ממוקמת מחוץ לגבולות הלוח, 4 אם החומה תחסום את השחקן לגמרי מלהגיע לסוף.

חשוב לציין כי fence מורכב מ firstId שהוא המיקום הראשון ו secondId שהוא המיקום השני.

1. התחל
2. אתחול:

int errorType = 1

1. אם: (this.fencesInGame.containsKey(fence.getFirstId()) או ((this.fencesInGame.containsKey(fence.getSecondId())
   1. errorType = 2
2. אם (fence.getIsHorizontal())
   1. אם this.fencesInGame.containsKey(fence.getFirstId() – 8)
      1. אם

(this.fencesInGame.get(fence.getFirstId() - 8).getFirstId() == (fence.getFirstId() - 8))

* + - 1. errorType = 2

1. אחרת
   1. אם this.fencesInGame.containsKey(fence.getFirstId() + 8)
      1. אם

(this.fencesInGame.get(fence.getFirstId() + 8).getFirstId() == (fence.getFirstId() + 8))

* + - 1. errorType = 2

1. אם (fence.getFirstId() % 17 == 16) || (fence.getFirstId() > 135)
   1. errorType = 3
2. this.getBoard().addEdge(fence.getA(), fence.getB())
3. this.getBoard().addEdge(fence.getC(), fence.getD())
4. אם

(!this.getBoard().playerCanGetToEnd(this.getCurrentPlayerPos(), this.getCurrPlayer()))

* 1. errorType = 4

1. this.getBoard().removeEdge(fence.getA(), fence.getB())
2. this.getBoard().removeEdge(fence.getC(), fence.getD())
3. return errorType

פסיאודו קוד shortestPathToRow

1. התחל
2. אתחול:

List<Integer> path = new LinkedList<>();

Queue<Integer> queue = new LinkedList<>();

HashMap<Integer, Integer> parentNode = new HashMap<>();

boolean[][] gameBoard = this.getBoard().getAdjacentMatrix();

1. Queue.add(src)
2. ParentNode.put(src, null)
3. כל עוד queue לא ריק
   1. Int t = queue.poll()
   2. Int t\_row = t / 9
   3. If(t\_row == row)
      1. כל עוד (t != src)
         1. Path.add(t)
         2. t = parentNode.get(t)
      2. reverse(path)
      3. return path
   4. עבור I קטן מ כמות הצמתים בגרף
      1. אם gameBoard[t][i]
         1. אם אין parentNode.containsKey(i)
            1. ParentNode.put(I, t)
            2. Queue.add(i)
4. Return path

## מחלקת Minimax

אחראית על החישובים הקשורים למציאת המהלך הטוב ביותר למחשב.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| כותרת הפונקציה | הפרמטרים של הפונקציה | איזה טיפוס הפונקציה מחזירה | מה הפונקציה מבצעת | יעילות הפונקציה |
| bestMoveCalc | Game game | Move | מחזירה את המהלך הטוב ביותר לשחקן הממוחשב | O(b^(d/2))  B – כמות המהלכים האפשריים  D - עומק |
| minimaxWithAlphaBetaWithVal | Game game  int depth  int alpha  int beta  boolean maxPlayer | void | עוברת על כל המהלכים האפשריים של המחשב ועל כל אחד קוראת לפונקציה minimaxWithAlphaBeta המהלך שקיבל את הציון הכי גבוה ישמר ויוחזר בהמשך | O(b^(d/2))  B – כמות המהלכים האפשריים  D - עומק |
| minimaxWithAlphaBeta | Game game  int depth  int alpha  int beta  boolean maxPlayer | int | מחשבת ציון למהלכים בחשיבה קדימה על כל מהלך כפי שתיארתי מספר פעמים בספר זה | O(b^(d/2))  B – כמות המהלכים האפשריים  D - עומק |
| heuristic | Game game | int | תחזיר את ההפרש בין המרחק בין השחקן הממוחשב לסוף לבין המרחק בין השחקן האנושי לסוף | O(1) |

פסיאודו קוד bestMoveCalc

1. התחל
2. אם נשארו לשחקן הנוכחי חומות
   1. Game gameCopy = new Game(game)
   2. MinimaxWithAlphaBetaWithVal(gameCopy, depth, Integer.MIN\_VALUE, Integer.MAX\_VALUE, true)
3. אחרת
   1. בצע:

this.bestMove = new PlayerMove(game.getCurrentPlayerPos(), game.shortestPathToRow(game.getCurrentPlayerPos(), 0).get(0))

1. Return this.bestMove

פסיאודו קוד minimaxWithAlphaBetaWithVal

1. התחל
2. אתחול:

int val;

ValidMoves validMoves = new ValidMoves();

1. עבור כל move ברשימת המהלכים האפשריים לשחקן הנוכחי
   1. Game child = new Game(game)
   2. Child.doMove(move)
   3. val = minimaxWithAlphaBeta(child, depth, alpha, beta, false)
   4. אם val > alpha
      1. Alpha = val
      2. this.bestMove = move

פסיאודו קוד heuristic

1. התחל
2. Return (game.shortestPathToRow(game.getPlayerTwo().getPosition(), 0).size() – game.shertestPathToRow(game.getPlayerOne().getPosition(), 0).size())

פסיאודו קוד minimaxWithAlphaBeta

1. התחל
2. אם depth == 0 או המשחק נגמר
   1. Return heuristic(game)
3. אם maxPlayer == true
   1. ValidMoves validMoves = new ValidMoves()
   2. עבור כל move ברשימת המהלכים האפשריים לשחקן המקסימלי
      1. Game child = new Game(game)
      2. Child.doMove(move)
      3. alpha =

Math.max(alpha, minimaxWithAlphaBeta(child, depth – 1, alpha, beta, false))

* + 1. אם beta <= alpha
       1. Break
  1. Return alpha

1. אחרת
   1. ValidMoves validMoves = new ValidMoves()
   2. עבור כל move ברשימת המהלכים האפשריים לשחקן המינימלי
      1. Game child = new Game(game)
      2. Child.doMove(move)
      3. beta =

Math.min(beta, minimaxWithAlphaBeta(child, depth – 1, alpha, beta, false))

* + 1. אם beta <= alpha
       1. Break
  1. Return beta

## מחלקת validMoves

אחראית על בניית רשימה של המהלכים האפשריים.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| כותרת הפונקציה | הפרמטרים של הפונקציה | איזה טיפוס הפונקציה מחזירה | מה הפונקציה מבצעת | יעילות הפונקציה |
| getValidMovesList | Game game  Int playerPosittion | List<Move> | מחזירה של כל המהלכים האפשריים | O(v + e)  V – כמות הצמתים בגרף  E – כמות מיקומים אפשריים שיהיה חומה |

פסיאודו קוד getMovesList

1. התחל
2. This.validMoves.clear()
3. עבור כל i שקטן מכמות הצמתים בלוח
   1. אם קפיצה ממיקום השחקן למיקום i בלוח חוקי
      1. PlayerMove playerMove = new PlayerMove(playerPosition, i)
      2. This.addMoveToValidMovesList(playerMove)
4. עבור כל i שקטן מכמות מיקומים אפשריים לחומה בלוח
   1. This.getCurrABCD(i)
   2. Fence fence =

new Fence(i, getSecondId(i), isFenceHorizontal(i),

this.CURRENT\_A, this.CURRENT\_B,

this.CURRENT\_C, this.CURRENT\_D);

* 1. אם ה fence חוקי
     1. AddMoveToValidMovesList(fence)

1. Return this.validMovesList

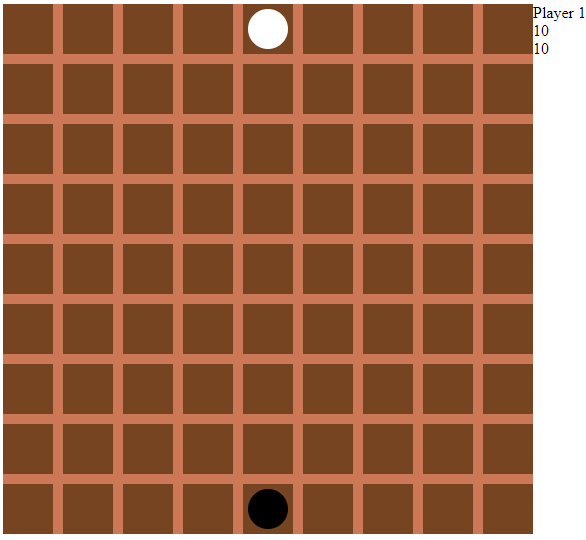
# **מדריך למשתמש**

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!

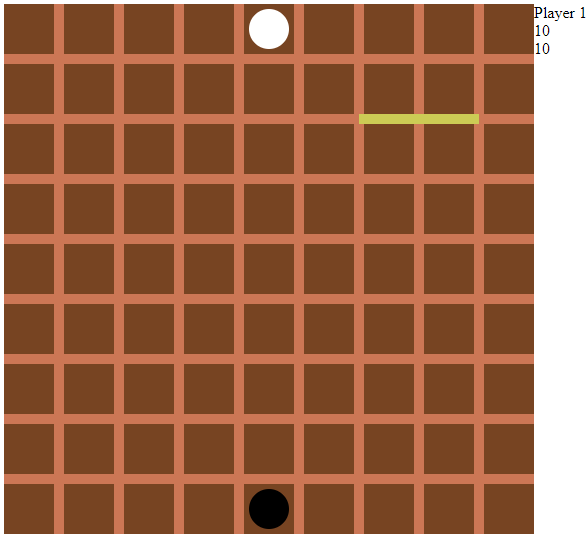
המשחק:

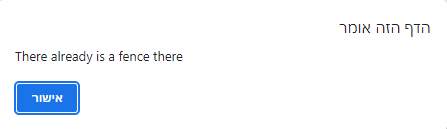
1. השחקן האנושי הוא השחקן הלבן (Player 1) והממוחשב הוא השחקן השחור (Player 2).
2. בצד ימין של הלוח יוצג תור איזה שחקן עכשיו וכמה חומות נשארו לכל שחקן.
3. כדי להתחיל את המשחק, לחצו על כפתור open מה שיתחיל את החיבור בין השרת ללקוח.
4. כדי לבצע מהלך של תזוזה של שחקן, על השחקן האנושי ללחוץ על הריבוע אליו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



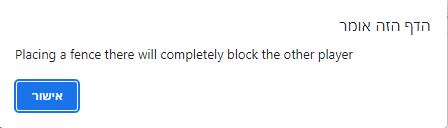
1. אם השחקן ינסה לקפוץ למקום אסור בלוח או מעל חומה, המסך הבא יעלה:



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

ראה בדף הבר

### מחלקת Board

אחראית על עדכונים של המטריצת סמיכויות.

public class Board {

    private int vertices;

    private boolean adjMatrix[][];

    // Default constructor

    public Board() {

        this.vertices = 81;

        this.adjMatrix = new boolean[vertices][vertices];

    }

    // Copy constructor

    public Board(Board board) {

        this.vertices = board.vertices;

        this.adjMatrix = new boolean[vertices][vertices];

        for (int i = 0; i < this.vertices; i++) {

            for (int j = 0; j < this.vertices; j++) {

                this.adjMatrix[i][j] = board.adjMatrix[i][j];

            }

        }

    }

    //returns adjMatrix

    public boolean[][] getAdjacentMatrix(){

        return this.adjMatrix;

    }

    //receives two points and adds an edge between them

    //both directions

    public void addEdge(int source, int destination) {

        this.adjMatrix[source][destination] = true;

        this.adjMatrix[destination][source] = true;

    }

    //receives two points and removes the edge between them

    //both directions

    public void removeEdge(int source, int destination) {

        this.adjMatrix[source][destination] = false;

        this.adjMatrix[destination][source] = false;

    }

    //is called in the begining of the game

    //adds edges where there needs to be in initialization

    public void initAdjBoard() {

        int i = 0, j = 0;

        for (i = 0; i < this.vertices; i++) {

            for (j = 0; j < this.vertices; j++) {

                this.adjMatrix[i][j] = false;

            }

        }

        for (i = 0; i < this.vertices - 1; i++) {

            this.adjMatrix[i][i + 1] = true;

            this.adjMatrix[i + 1][i] = true;

            if (i < 72) {

                this.adjMatrix[i][i + 9] = true;

                this.adjMatrix[i + 9][i] = true;

            }

        }

    }

    //receives a position on the board ((int) playerPos)

    //receives the Player ((Player) currentPlayer)

    //checks if it is possible for the player to get to the end (it's winning row)

    //returns boolean: true if this is possible, false if not

    public boolean playerCanGetToEnd(int playerPos, Player currentPlayer) {

        boolean[] visited = new boolean[this.vertices];

        int[] parents = new int[this.vertices];

        int curPlayerPos;

        // Initialize with -1 to know when to stop if we haven't reached playerPos

        for (int i = 0; i < this.vertices; i++) {

            parents[i] = -1;

        }

        dfs(playerPos, visited, parents);

        // 72 is starting point for range 72 -> 80. possible end points for

        // player1(playerNo = true)

        // 0 for range 0 -> 8. possible end points for player2(playerNo = false)

        int endPlayerPos = (currentPlayer.getIsPlayerWhite() == true) ? 72 : 0;

        // 9 possible endpoints.

        for (int i = 0; i < 9; i++) {

            curPlayerPos = endPlayerPos + i;

            while (parents[curPlayerPos] != -1) {

                if (parents[curPlayerPos] == playerPos) {

                    return true;

                }

                curPlayerPos = parents[curPlayerPos];

            }

        }

        return false;

    }

    //receives starting position ((int) start)

    //receives (boolean) array of visited

    //receives (int) array of parents

    //function will fill parent array with the help of the visited array

    public void dfs(int start, boolean[] visited, int[] parents) {

        // Set current node as visited

        visited[start] = true;

        // For every node of the graph

        for (int i = 0; i < this.adjMatrix[start].length; i++) {

            // If some node is adjacent to the current node

            // and it has not already been visited

            if (this.adjMatrix[start][i] && (!visited[i])) {

                parents[i] = start;

                dfs(i, visited, parents);

            }

        }

    }

    //prints graph - helper to check graph state, not relevant for game

    public void printAdjGraph() {

        for (int i = 0; i < this.vertices; i++) {

            for (int j = 0; j < this.vertices; j++) {

                if (this.adjMatrix[i][j]) {

                    System.out.print("1 ");

                } else {

                    System.out.print("0 ");

                }

            }

            System.out.println();

        }

    }

    //receives to (int) points in graph

    //returns boolean: true if the two points are adjacent and false if not

    public boolean isAdj(int sourcePos, int destPos) {

        return (this.adjMatrix[sourcePos][destPos] || this.adjMatrix[destPos][sourcePos]);

    }

}

### מחלקת Fence

public class Fence extends Move{

    private int firstId;

    private int secondId;

    private int a;

    private int b;

    private int c;

    private int d;

    private boolean isHorizontal;

    //constructor

    //a, b, c, d are the nodes in the graph that will be affected by the fence

    //in placing a fence, a will be disconnected froam b and c from d

    public Fence(int firstId, int secondId, boolean isHorizontal, int a, int b, int c, int d){

        super("f");

        this.firstId = firstId;

        this.secondId = secondId;

        this.a = a;

        this.b = b;

        this.c = c;

        this.d = d;

        this.isHorizontal = isHorizontal;

    }

    //returns boolean: true if fence is horizontal, false if not

    public boolean getIsHorizontal(){

        return this.isHorizontal;

    }

    //returns int: the first id of the fence

    public int getFirstId(){

        return this.firstId;

    }

    //returns int: the second id of the fence

    public int getSecondId(){

        return this.secondId;

    }

    //returns int: A node

    public int getA(){

        return this.a;

    }

    //returns int: B node

    public int getB(){

        return this.b;

    }

    //returns int: C node

    public int getC(){

        return this.c;

    }

    //returns int: D node

    public int getD(){

        return this.d;

    }

}

### מחלקת Game

אחראית על פונקציות הקשורות למהלך המשחק.

import java.util.HashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Collections;

import java.util.Queue;

public class Game {

    private Board board;

    private Player player1; // white - starts from low to high

    private Player player2; // black - starts from high to low

    private Player currentPlayer;

    /\*

    the integer, fence hashmap:

    fences in game are entered twice into the hashmap-

    once with the key being the first id

    second time key is second id

    this makes the checking of thee fences later far less complex and \*/

    private HashMap<Integer, Fence> fencesInGame;

    //default constructor

    public Game() {

        this.board = new Board();

        this.board.initAdjBoard();

        this.player1 = new Player(4, true); // white - starts from low to high

        this.player2 = new Player(76, false); // black - starts from high to low

        this.currentPlayer = player1;

        this.fencesInGame = new HashMap<Integer, Fence>();

    }

    // copy constructor

    public Game(Game game) {

        this.board = new Board(game.getBoard());

        this.player1 = new Player(game.getPlayerOne());

        this.player2 = new Player(game.getPlayerTwo());

        this.currentPlayer = game.getCurrPlayer();

        this.fencesInGame = new HashMap<Integer, Fence>();

        this.fencesInGame.putAll(game.getfencesInGame());

    }

    //returns Player: player1

    public Player getPlayerOne() {

        return this.player1;

    }

    //returns Player: player2

    public Player getPlayerTwo() {

        return this.player2;

    }

    //switches the current player in this game

    public void switchPayer() {

        if (this.currentPlayer == player1)

            this.currentPlayer = player2;

        else

            this.currentPlayer = player1;

    }

    //returns Board: board of this game

    public Board getBoard() {

        return this.board;

    }

    //returns Player: this game's current player

    public Player getCurrPlayer() {

        return this.currentPlayer;

    }

    //returns int: this game's current player's position

    public int getCurrentPlayerPos() {

        return this.currentPlayer.getPossition();

    }

    //returns int: this game's other(not current) player's position

    public int getOtherPlayerPos() {

        if (this.currentPlayer == this.player1)

            return this.player2.getPossition();

        return this.player1.getPossition();

    }

    //recieves ((Player) player) and ((int) pos)

    //sets player's position to pos

    public void setPlayerPos(Player player, int pos) {

        player.setPossition(pos);

    }

    //receives ((int) oldPos) and ((int) newPos) - two positions on the board

    //checks if the jump from oldPos to newPos is legal

    //considers special cases of player jumping over other player

    //returns boolean: true if the jump is legal and false if not

    public boolean checkIfLegalConsideringJump(int oldPos, int newPos) {

        Board boardCopy = new Board(this.getBoard());

        int playerPosDif = getOtherPlayerPos() - getCurrentPlayerPos();

        switch (playerPosDif) {

            case (1): // curr player is on the left side of other player

                boardCopy.removeEdge(oldPos, getOtherPlayerPos());

                //checks that there is no fence on right side of other player

                if (boardCopy.isAdj((getOtherPlayerPos()), (getOtherPlayerPos() + 1)))

                    boardCopy.addEdge(oldPos, (getOtherPlayerPos() + 1));

                break;

            case (-1): // curr player is on the right side of other player

                boardCopy.removeEdge(oldPos, getOtherPlayerPos());

                //checks that there is no fence on left side of other player

                if (boardCopy.isAdj((getOtherPlayerPos()), (getOtherPlayerPos() - 1)))

                    boardCopy.addEdge(oldPos, (getOtherPlayerPos() - 1));

                break;

            case (9): // curr player is below the other player

                boardCopy.removeEdge(oldPos, getOtherPlayerPos());

                //checks that there is no fence above other player

                if (boardCopy.isAdj((getOtherPlayerPos()), (getOtherPlayerPos() + 9)))

                    boardCopy.addEdge(oldPos, (getOtherPlayerPos() + 9));

                break;

            case (-9): // curr player is above the other player

                boardCopy.removeEdge(oldPos, getOtherPlayerPos());

                //checks that there is no fence below other player

                if (boardCopy.isAdj((getOtherPlayerPos()), (getOtherPlayerPos() - 9)))

                    boardCopy.addEdge(oldPos, (getOtherPlayerPos() - 9));

                break;

            default:

                // regular move

        }

        return boardCopy.isAdj(oldPos, newPos);

    }

    //retruns map of all fences in the game

    public HashMap<Integer, Fence> getfencesInGame() {

        return this.fencesInGame;

    }

    //adds a fence to the map

    public void addFenceToMap(Fence fence) {

        this.fencesInGame.put(fence.getFirstId(), fence);

        this.fencesInGame.put(fence.getSecondId(), fence);

    }

    //receves (int(id))

    //returns Fence: the fence from the map of fences in the game with that id(key)

    public Fence getFenceById(int id) {

        return this.fencesInGame.get(id);

    }

    //receives ((Fence)fence)

    //checks if the fence is legal on the board

    //return int: error type

    public int checkFenceLegal(Fence fence) {

        /\*

         \* returns 1 if move is legal

         \* returns 2 if a fence is placed on another fence (either parallel or cross)

         \* returns 3 if fence is placed out of bounds

         \* returns 4 if placing the fence will create a situation in which a player is

         \* completely blocked

         \*/

        int errorType = 1;

        // no fence on fence - parallel

        if ((this.fencesInGame.containsKey(fence.getFirstId()))

                || (this.fencesInGame.containsKey(fence.getSecondId()))) {

            errorType = 2;

        }

        // no fence on fence - cross

        if (fence.getIsHorizontal()) { // horizontal fence

            if (this.fencesInGame.containsKey(fence.getFirstId() - 8)) {

                if (this.fencesInGame.get(fence.getFirstId() - 8).getFirstId() == (fence.getFirstId() - 8)) {

                    errorType = 2;

                }

            }

        }

        else { // vertical fence

            if (this.fencesInGame.containsKey(fence.getFirstId() + 8)) {

                if (this.fencesInGame.get(fence.getFirstId() + 8).getFirstId() == (fence.getFirstId() + 8)) {

                    errorType = 2;

                }

            }

        }

        // no out of bounds

        //%17 == 16 for horizontal out of bounds

        //>135 for vertical out of bounds

        if ((fence.getFirstId() % 17 == 16) || (fence.getFirstId() > 135)) {

            errorType = 3;

        }

        // setup for dfs check

        this.getBoard().addEdge(fence.getA(), fence.getB());

        this.getBoard().addEdge(fence.getC(), fence.getD());

        // player cant get across

        if (!this.getBoard().playerCanGetToEnd(this.getCurrentPlayerPos(), this.getCurrPlayer())) {

            errorType = 4;

        }

        // edges are no longer relavent

        this.getBoard().removeEdge(fence.getA(), fence.getB());

        this.getBoard().removeEdge(fence.getC(), fence.getD());

        return errorType;

    }

    //returns boolean: true if game is over (player has reached anending point)

    //false if not

    public boolean isOver() {

        return (this.player1.getPossition() > 71 || this.player2.getPossition() < 9);

    }

    //receives ((Move) move)

    //function will do the move

    public void doMove(Move move) {

        if (move.getMoveType().equals("m")) {

            //the move is a piece movement

            setPlayerPos(getCurrPlayer(), ((PlayerMove) move).getPlayerNewPos());

            switchPayer();

        }

        else if (move.getMoveType().equals("f")) {

            //the move is a fence placement

            addFenceToMap(((Fence) move));

            getCurrPlayer().useFence();

            getBoard().removeEdge(((Fence) move).getA(), ((Fence) move).getB());

            getBoard().removeEdge(((Fence) move).getC(), ((Fence) move).getD());

            switchPayer();

        }

    }

    //receives ((int) src) position on the board

    //receives ((int)row) a row on the board (0 - 8)

    //function will find the shortest path from src position to anywhere on the row of the board

    //return List<Integer>: list of integers that represent the IDs of the positioons in the shortest path

    public List<Integer> shortestPathToRow(int src, int row) {

        List<Integer> path = new LinkedList<>();

        Queue<Integer> queue = new LinkedList<>();

        HashMap<Integer, Integer> parentNode = new HashMap<>();

        // enqueue start configuration onto queue

        queue.add(src);

        // mark start configuration

        parentNode.put(src, null);

        // Get game board of the current game

        boolean[][] gameBoard = this.getBoard().getAdjacentMatrix();

        while (!queue.isEmpty()) {

            int t = queue.poll();

            int t\_row = t / 9; // Get row by position

            if (t\_row == row) {

                while (t != src) {

                    path.add(t);

                    t = parentNode.get(t);

                }

                Collections.reverse(path);

                return path;

            }

            for (int i = 0; i < gameBoard[0].length; i++) {

                // Get current pos on board by row and col

                if(gameBoard[t][i]){

                    if (!parentNode.containsKey(i)) {

                        parentNode.put(i, t);

                        queue.add(i);

                    }

                }

            }

        }

        return path;

    }

}

### מחלקת Minimax

אחראית על החישובים הקשורים למציאת המהלך הטוב ביותר למחשב.

public class Minimax {

    int depth;

    Move bestMove;

    public Minimax(int depth) {

        this.depth = depth;

    }

    //receives Game game and returns the best move for AI player to make (Move)

    public Move bestMoveCalc(Game game) {

        if (game.getCurrPlayer().hasFences()) {

            //if has fences left - check best move option using minimax alpha beta

            Game gameCopy = new Game(game);

            minimaxWithAlphaBetaWithVal(gameCopy, depth, Integer.MIN\_VALUE, Integer.MAX\_VALUE, true);

        }

        else {

            //no fences left - best option is to head for the path shortest to the finish line

            this.bestMove = new PlayerMove(game.getCurrentPlayerPos(),

                    game.shortestPathToRow(game.getCurrentPlayerPos(), 0).get(0));

        }

        return this.bestMove;

    }

    //receives the game (Game game)

    //receives the depth for AI to search (int depth)

    //receives int alpha, int beta foor alpha beta pruning

    //recieves (boolean maxPlayer): true if we are searching for max resalt, false if searching for min result

    //goes over all possible moves of AI (at current game state)

    //function will give a score to each move, saves the best

    public void minimaxWithAlphaBetaWithVal(Game game, int depth, int alpha, int beta, boolean maxPlayer) {

        int val;

        ValidMoves validMoves = new ValidMoves();

        //goes ove list of valid moves

        for (Move move : validMoves.getValidMovesList(game, game.getCurrentPlayerPos())) {

            Game child = new Game(game);

            child.doMove(move);//tries move

            //gets score (val)for move

            val = minimaxWithAlphaBeta(child, depth, alpha, beta, false);

            if (val > alpha) {

                //found bbest move yet

                alpha = val;

                //sets as best move

                if(move.getMoveType().equals("m")){

                    this.bestMove = new PlayerMove(((PlayerMove)move).getPlayerOldPos(), ((PlayerMove)move).getPlayerNewPos());

                }

                else{

                    this.bestMove = new Fence(((Fence)move).getFirstId(), ((Fence)move).getSecondId(),

                    ((Fence)move).getIsHorizontal(), ((Fence)move).getA(), ((Fence)move).getB(),

                    ((Fence)move).getC(), ((Fence)move).getD());

                }

            }

            //pruning

            if (beta <= alpha) {

                break;

            }

        }

    }

    //receives the game (Game game)

    //receives the depth for AI to search (int depth)

    //receives int alpha, int beta foor alpha beta pruning

    //recieves (boolean maxPlayer): true if we are searching for max resalt, false if searching for min result

    //calcuates score for each possible move of the computer

    //uses minimax algorithm with the extension of alpha beta pruning

    //returns int: the score

    public int minimaxWithAlphaBeta(Game game, int depth, int alpha, int beta, boolean maxPlayer) {

        if (depth == 0 || game.isOver()) {

            return heuristic(game);

        }

        if (maxPlayer) {

            ValidMoves validMoves = new ValidMoves();

            //goes ove list of valid moves

            for (Move move : validMoves.getValidMovesList(game, game.getCurrentPlayerPos())) {

                Game child = new Game(game);

                child.doMove(move); //tries move

                //alpha has to be maximum score

                alpha = Math.max(alpha, minimaxWithAlphaBeta(child, depth - 1, alpha, beta, false));

                //pruning

                if (beta <= alpha) {

                    break;

                }

            }

            return alpha;

        }

        else {

            ValidMoves validMoves = new ValidMoves();

            //goes ove list of valid moves

            for (Move move : validMoves.getValidMovesList(game, game.getCurrentPlayerPos())) {

                Game child = new Game(game);

                child.doMove(move);//tries move

                //beta has to be minimum score

                beta = Math.min(beta, minimaxWithAlphaBeta(child, depth - 1, alpha, beta, true));

                //pruning

                if (beta <= alpha) {

                    break;

                }

            }

            return beta;

        }

    }

    //receives Game game

    //returns int: the difference between the shortest path from AI player to it's finish

    //and between the shortest path from human player to it's finish line

    public int heuristic(Game game) {// shortest path difference

        return game.shortestPathToRow(game.getPlayerTwo().getPossition(), 0).size()

                - game.shortestPathToRow(game.getPlayerOne().getPossition(), 8).size();

    }

}

### מחלקת Move

האב של מחלקת Fence ומחלקת PlayerMove

public class Move {

    //moveType- "f" for fence placement, "m" for piece movement

    private String moveType;

    //abstract parent of Fence and PlayerMove

    //default constructor

    public Move(){}

    //constructor

    public Move(String moveType){

        this.moveType = moveType;

    }

    //returns string: the movement type ("f" or "m")

    public String getMoveType(){

        return this.moveType;

    }

}

### מחלקת PlayerMove

public class PlayerMove extends Move{

    private int oldPos;

    private int newPos;

    //constructor

    public PlayerMove(int oldPos, int newPos){

        super("m");

        this.oldPos = oldPos;

        this.newPos = newPos;

    }

    //returns int: the old(current) position of a player on the board(id)

    public int getPlayerOldPos(){

        return this.oldPos;

    }

    //returns int: the position on the board a player will jump to in this move(id)

    public int getPlayerNewPos(){

        return this.newPos;

    }

}

### מחלקת Player

public class Player {

    private int position;

    private boolean isWhite;

    private int amountFences;

    // constructor

    public Player(int possition, boolean isWhite) {

        this.isWhite = isWhite;

        this.position = possition;

        this.amountFences = 10; //starts game with ten fences, goes down with use

    }

    // copy constructor

    public Player(Player player){

        this.isWhite = player.isWhite;

        this.position = player.position;

        this.amountFences = player.amountFences;

    }

    //returns boolean: true if player is white, false if not

    public boolean getIsPlayerWhite(){

        return this.isWhite;

    }

    //returns int: position of the player on the board (id)

    public int getPossition() {

        return this.position;

    }

    //sets player position on board (id)

    public void setPossition(int position) {

        this.position = position;

    }

    public void useFence(){

        this.amountFences--;

    }

    //returns int: the amount of fences the player has remaining

    public int getAmountFences(){

        return this.amountFences;

    }

    //returns boolean: true if player has fences left, false if not

    public boolean hasFences(){

        return (this.amountFences > 0);

    }

}

### מחלקת ValidMoves

אחראית על בניית רשימה של המהלכים האפשריים.

import java.util.List;

import java.util.ArrayList;

public class ValidMoves {

    private List<Move> validMoves;

    private int AMOUNT\_SQUARES\_ON\_BOARD;

    private int MAX\_POSSIBLE\_FENCE\_ID;

    private int CURRENT\_A;

    private int CURRENT\_B;

    private int CURRENT\_C;

    private int CURRENT\_D;

    public ValidMoves(){

        validMoves = new ArrayList<Move>();

        // constants

        AMOUNT\_SQUARES\_ON\_BOARD = 81;

        MAX\_POSSIBLE\_FENCE\_ID = 134;

    }

    //receives the game(Game) and (int)playerPositions

    //function will run on all the options of moves in the game and add the legal options to th list

    //returns List<Move>: the list of valid moves created

    public List<Move> getValidMovesList(Game game, int playerPosition) {

        this.validMoves.clear();

        // adding player moves

        for (int i = 0; i < AMOUNT\_SQUARES\_ON\_BOARD; i++) {

            if (game.checkIfLegalConsideringJump(playerPosition, i)) {

                //move is legal

                PlayerMove playerMove = new PlayerMove(playerPosition, i);//create PlayerMove

                this.addMoveToValidMovesList(playerMove);//add to list

            }

        }

        // adding fence moves

        for (int i = 0; i <= MAX\_POSSIBLE\_FENCE\_ID; i++) {

            this.getCurrABCD(i);

            Fence fence = new Fence(i, getSecondId(i), isFenceHorizontal(i),

                    this.CURRENT\_A, this.CURRENT\_B,

                    this.CURRENT\_C, this.CURRENT\_D);//create Fence to try

            if (game.checkFenceLegal(fence) == 1) {

                //fence placement is legal

                addMoveToValidMovesList(fence);//add to list

            }

        }

        return this.validMoves;

    }

    //receives Move move

    //adds to the list of valid moves

    public void addMoveToValidMovesList(Move move) {

        this.validMoves.add(move);

    }

    //receives int firstId (of a fence)

    //returns int: the second id of the fence

    public int getSecondId(int firstId) {

        if (isFenceHorizontal(firstId)){

            //if fence is horizontal, the next id will be one on the right

            return firstId + 1;

        }

        //if fence is vertical, the next fence id will be one bellow (+17)

        return firstId + 17;

    }

    //receives the first id of the fence

    //calculates by the id if the fence is horizontal or not

    //returns boolean: true if it is horizontal, false if not

    public boolean isFenceHorizontal(int firstId) {

        return (firstId % 17 > 7);

    }

    //receives int firstId (the first id of a fence)

    //calculates the id s of the four nodes (a b c d) affected by the fence

    //a and b will have a fence between them

    //c and d will have a fence between them

    public void getCurrABCD(int firstId) {

        int row = firstId / 17; //row fence is in

        int col = (firstId - 9 \* row) % 8; //columb fence is in

        if (isFenceHorizontal(firstId)) {

            //if fence is horizontal, b is below a, c is right of a, d is below c

            this.CURRENT\_A = col + row \* 9;

            this.CURRENT\_B = this.CURRENT\_A + 9;

            this.CURRENT\_C = this.CURRENT\_A + 1;

            this.CURRENT\_D = this.CURRENT\_B + 1;

        }

        else {

            //if fence is vertical, b right of a, c below a, d is right of c

            this.CURRENT\_A = col + row \* 9;

            this.CURRENT\_B = this.CURRENT\_A + 1;

            this.CURRENT\_C = this.CURRENT\_A + 9;

            this.CURRENT\_D = this.CURRENT\_C + 1;

        }

    }

}

### מחלקת QuoridorServerEndpoint

פונקציות הקשורות לקשר השרת לקוח: קבלת הודעות מהלקוח ושליחה ללקוח.

import javax.websocket.OnClose;

import javax.websocket.OnError;

import javax.websocket.OnMessage;

import javax.websocket.OnOpen;

import javax.websocket.Session;

import javax.websocket.server.ServerEndpoint;

import com.google.gson.Gson;

import com.google.gson.JsonObject;

import com.google.gson.JsonParser;

import java.io.IOException;

@ServerEndpoint(value = "/echo")

public class QuoridorServerEndpoint {

    Game game;

    //opens session server - client

    @OnOpen

    public void onOpen(Session session) {

        System.out.println("[Connection] SessionID = " + session.getId());

        try {

            session.getBasicRemote().sendText("Connection Established");

        } catch (IOException ex) {

            ex.printStackTrace();

        }

        this.game = new Game(); //creates game

    }

    //receivesthe session (Session) and message (String) from client

    @OnMessage

    public void onMessage(String message, Session session) {

        System.out.println("[Message] " + message);

        int a, b, c, d, oldPos, newPos, fenceLegalErrorType, firstFenceId, secondFenceId;

        String fenceType = "";

        JsonObject innerObject = new JsonObject();

        Gson gson = new Gson();

        boolean isAdj;

        String str = "";

        JsonParser parser = new JsonParser();

        JsonObject obj = parser.parse(message).getAsJsonObject();

        String messageType = obj.get("type").getAsString(); //getting the message type from the client

        //three types of messages:

        //contains type = "f" : player would like to place a fence

        //contains type = "m" : player would like to move his piece

        //contains type = "i" : it is the AI's turn to move

        // `send` message

        if (messageType.equals("s")) {

            return;

        }

        // Fence move

        else if (messageType.equals("f")) {

            //receiveing information from client's message about the fence

            a = obj.get("a").getAsInt();

            b = obj.get("b").getAsInt();

            c = obj.get("c").getAsInt();

            d = obj.get("d").getAsInt();

            firstFenceId = obj.get("firstId").getAsInt();

            secondFenceId = obj.get("secondId").getAsInt();

            fenceType = obj.get("fType").getAsString(); //h - horizontal, v - vertical

            //creating the fence via information received from client's message

            Fence fence = new Fence(firstFenceId, secondFenceId, (fenceType.equals("h")), a, b, c, d);

            fenceLegalErrorType = this.game.checkFenceLegal(fence);

            if (fenceLegalErrorType == 1) { // no errors - legal fence move

                //place fence

                this.game.addFenceToMap(fence);

                this.game.getCurrPlayer().useFence();

                this.game.switchPayer();

                this.game.getBoard().removeEdge(a, b);

                this.game.getBoard().removeEdge(c, d);

            }

            innerObject.addProperty("isLegal", (fenceLegalErrorType == 1)); // if returns one, no errors

            innerObject.addProperty("errorType", fenceLegalErrorType);

        }

        // Player move

        else if (messageType.equals("m")) {

            //receiveing information from client's message about the movement

            oldPos = obj.get("oldPos").getAsInt();

            newPos = obj.get("newPos").getAsInt();

            //legal or not

            isAdj = this.game.checkIfLegalConsideringJump(oldPos, newPos);

            if (isAdj) {

                //is legal - move piece

                this.game.setPlayerPos(this.game.getCurrPlayer(), newPos);

                this.game.switchPayer();

            }

            innerObject.addProperty("isLegal", isAdj);

        }

        // AI's turn

        else if (messageType.equals("i")) {

            Minimax minimaxAlphaBeta = new Minimax(2); ////////// depth change ///////////

            Move move = new Move();

            move = minimaxAlphaBeta.bestMoveCalc(this.game);

            //move will be the best move for the computer to make

            if (move.getMoveType().equals("f")) {

                //the move is a fence placement

                //place fence

                this.game.addFenceToMap(((Fence) move));

                this.game.getCurrPlayer().useFence();

                this.game.switchPayer();

                this.game.getBoard().removeEdge(((Fence) move).getA(), ((Fence) move).getB());

                this.game.getBoard().removeEdge(((Fence) move).getC(), ((Fence) move).getD());

                //preparing message for client (from server)

                innerObject.addProperty("aiMoveType", "f");

                innerObject.addProperty("fenceID", ((Fence)move).getFirstId());

            }

            else if (move.getMoveType().equals("m")) {

                //the moveis a piece movement

                //move piece

                this.game.setPlayerPos(this.game.getCurrPlayer(), ((PlayerMove) move).getPlayerNewPos());

                this.game.switchPayer();

                //preparing message for client (from server)

                innerObject.addProperty("aiMoveType", "m");

                innerObject.addProperty("oldPos", ((PlayerMove)move).getPlayerOldPos());

                innerObject.addProperty("newPos", ((PlayerMove)move).getPlayerNewPos());

            }

            innerObject.addProperty("isLegal", true);

        }

        // type of message the client should deal with

        //("m" - piece movement, "f" - fence placement, "i" - ai turn)

        innerObject.addProperty("messageType", messageType);

        str = gson.toJson(innerObject);

        try {

            session.getBasicRemote().sendText(str);

        } catch (IOException ex) {

            ex.printStackTrace();

        }

    }

    //close session

    @OnClose

    public void onClose(Session session) {

        System.out.println("[Disconnection] SessionID = " + session.getId());

    }

    //error catch

    @OnError

    public void onError(Throwable e) {

        System.out.println("[Error] " + e);

    }

}

### מחלקת QuoridorServer

import org.glassfish.tyrus.server.Server;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class QuoridorServer {

    public static void main(String[] args) {

        Server server = new Server("localhost", 8080, "/Quoridor", QuoridorServerEndpoint.class);

        try {

            server.start();

            System.out.println("\nServer is up...");

            System.out.println("Press any key to stop the server...\n");

            BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));

            bufferRead.readLine();

        } catch (Exception e) {

            e.printStackTrace();

        } finally {

            server.stop();

        }

    }

}

### Sockets.js

הצד לקוח- שולח הודעות ומקבל הודעות מהשרת

var webSocket;

var messages;

function sockets() {

    messages = document.getElementById("messages");

}

function openSocket() {

    // Ensures only one connection is open at a time

    if (webSocket !== undefined && webSocket.readyState !== WebSocket.CLOSED) {

        writeResponse("WebSocket is already opened.");

        return;

    }

    // Create a new instance of the websocket

    webSocket = new WebSocket("ws://localhost:8080/Quoridor/echo");

    /\*\*

     \* Binds functions to the listeners for the websocket.

     \*/

    webSocket.onopen = function (event) {

        // For reasons I can't determine, onopen gets called twice

        // and the first time event.data is undefined.

        if (event.data === undefined)

            return;

        writeResponse(event.data);

    };

    webSocket.onmessage = function (event) {

        var obj = JSON.parse(event.data);

        if (typeof obj == "object") {

            //Json

            if(obj.messageType == "f"){

                if(obj.isLegal == true){

                    placeFenceForReal();

                }

                // fence on fence

                else if (obj.errorType == 2){

                    alert("There already is a fence there");

                }

                // fence out of bounds

                else if (obj.errorType == 3){

                    //alert("Fence is out of bounds");

                    alert("Placing a fence there will completely block the other player");

                }

                // fence will completely block player

                else if (obj.errorType == 4){

                    alert("Placing a fence there will completely block the other player");

                }

            }

            else if(obj.messageType == "m"){

                if(obj.isLegal == true)

                    movePlayerForReal();

                else

                    alert("Can't jump to there")

            }

            else if(obj.messageType == "i"){

                if(obj.aiMoveType == "f"){

                    //placeFenceForReal();

                    aiPlaceFence(obj.fenceID);

                }

                else{

                    aiMovePlayer(obj.newPos);

                    //movePlayerForReal();

                }

            }

            writeResponse("" + obj.isLegal);

        }

        else {

            //Not Json

            writeResponse(event.data);

        }

    };

    webSocket.onclose = function (event) {

        writeResponse("Connection closed.");

    };

}

/\*\*

 \* Sends the value of the text input to the server

 \*/

function send() {

    var text = document.getElementById("messageinput").value;

    // to read "type" for regular `send` messages.

    var obj = { "type": "s", "message": text, "name": "hadar" };

    webSocket.send(JSON.stringify(obj));

}

function sendAi(){

    var obj = { type: "i"};

    webSocket.send(JSON.stringify(obj));

}

function sendFence(fenceId, nextFenceId, a, b, c, d, fType) {

    var obj = { type: "f", firstId: fenceId, secondId: nextFenceId, a: a, b: b, c: c, d: d, fType: fType };

    webSocket.send(JSON.stringify(obj));

}

function sendMove(oldPos, newPos) {

    var obj = { type: "m", oldPos: oldPos, newPos: newPos };

    webSocket.send(JSON.stringify(obj));

}

function closeSocket() {

    webSocket.close();

}

function writeResponse(text) {

    messages.innerHTML += "<br/>" + text;

}

### Ui.js

קוד צד לקוח – התעסקות עם ה gui בעיקר

function ui() {

    $(function () {

        Quoridor.init('#board');

    });

}

var Quoridor = new function () {

    this.boardDimension = 9;

    this.startingFences = 10;

    this.player1 = null;

    this.player2 = null;

    this.moves = null;

    this.currentTurn = null;

    this.init = function (board) {

        var board = $(board);

        for (var i = 0; i < this.boardDimension; i++) {

            if (i != 0) {

                for (var j = 0; j < this.boardDimension; j++) {

                    var horizontalFence = $('<div class="fence horizontal" />')

                    if (j == 0) {

                        horizontalFence.addClass('left');

                    }

                    if (j == 8) {

                        horizontalFence.addClass('right');

                    }

                    board.append(horizontalFence);

                }

            }

            for (var j = 0; j < this.boardDimension; j++) {

                var squareNumber = (i \* this.boardDimension) + j;

                var square = $('<div id="square\_' + squareNumber + '" class="square" />')

                if (j != 0) {

                    var fence = $('<div class="fence" />')

                    board.append(fence);

                }

                board.append(square);

            }

        }

        initPlayers();

        bindSquareEventHandlers();

        bindFenceEventHandlers();

        updateInformation();

        board.after(Information.getPanel());

    }

    function bindFenceEventHandlers() {

        $('.fence').each(function (i, v) {

            $(v).attr('id', 'fence\_' + i);

        });

        $('.fence').hover(function () {

            $(this).addClass('selected');

            getAdjacentFence(this).addClass('selected');

        }, function () {

            $(this).removeClass('selected');

            getAdjacentFence(this).removeClass('selected');

        });

        $('.fence').click(function () {

            placeFence(this);

        });

    }

    function bindSquareEventHandlers() {

        $('.square').click(function () {

            var newPosition = parseInt($(this).attr('id').split('\_')[1]);

            movePlayer(newPosition);

        });

    }

    function initPlayers() {

        this.player1 = new Player("Player 1", 4, "player\_1", Quoridor.startingFences);

        this.player2 = new Player("Player 2", 76, "player\_2", Quoridor.startingFences);

        currentTurn = this.player1;

        updatePlayerPosition(this.currentTurn.pos);

        switchPlayerFirst();

        updatePlayerPosition(this.currentTurn.pos);

        switchPlayerFirst();

    }

    var firstObjectSent;

    var secondObjectSent;

    function movePlayer(newPosition) {

        if (isPossibleMove(newPosition)) {

            firstObjectSent = this.currentTurn.pos;

            secondObjectSent = newPosition

            sendMove(firstObjectSent, secondObjectSent)

        }

    }

    movePlayerForReal = function () {

        updatePlayerPosition(secondObjectSent);

        switchPlayer();

        checkWinner();

    }

    function isPossibleMove(newPosition) {

        var isLegal = true;

        return isLegal;

    }

    function updatePlayerPosition(position) {

        this.currentTurn.pos = position;

        var playerDiv = $('<div id="' + this.currentTurn.id + '" />');

        $('#' + this.currentTurn.id).remove();

        $('#square\_' + position).append(playerDiv);

    }

    function getAdjacentFence(fence) {

        if ($(fence).hasClass('horizontal')) {

            return $(fence).next();

        }

        else {

            adjacentIndex = parseInt($(fence).attr('id').split('\_')[1]) + 17;

            return $($('.fence')[adjacentIndex]);

        }

    }

    function isFencePlaceable(fence) {

        var isLegal = true;

        var nextFence = getAdjacentFence(fence);

        var fenceId = parseInt($(fence).attr('id').split('\_')[1]);

        var nextFenceId;

        if ($(fence).hasClass("placed")) {

            alert("There is a fence already there");

            $(isLegal) = false;

        }

        if ($(nextFence).hasClass("placed")) {

            alert("There is a fence already there");

            $(isLegal) = false;

        }

        // find vertical cross

        if (($(fence).hasClass("horizontal"))) {

            nextFenceId = parseInt($(nextFence).attr('id').split('\_')[1]);

            topVertical = fenceId - 8;

            bottomVertical = fenceId + 9;

            if ($($('.fence')[topVertical]).hasClass('placed') && $($('.fence')[bottomVertical]).hasClass('placed')) {

                alert("There is a fence already there");

                $(isLegal) = false;

            }

            if ((nextFenceId) != (fenceId + 1)) {

                alert("out of bounds");

                $(isLegal) = false;

            }

        }

        else {

            leftHorizontal = fenceId + 8;

            rightHorizontal = fenceId + 9;;

            if ($($('.fence')[leftHorizontal]).hasClass('placed') && $($('.fence')[rightHorizontal]).hasClass('placed')) {

                alert("There is a fence already there");

                $(isLegal) = false;

            }

            if (fenceId > 135) {

                alert("out of bounds");

                $(isLegal) = false;

            }

        }

        if (!hasFencesRemaining()) {

            alert("No fences left");

            $(isLegal) = false;

        }

        return (isLegal);

    }

    function hasFencesRemaining() {

        return this.currentTurn.fencesRemaining > 0;

    }

    function placeFence(fence) {

        firstObjectSent = fence;

        setUpForSenendFence(fence);

    }

    aiPlaceFence = function(id){

        var fence = $("#" + id);

        $(fence).addClass('placed');

        getAdjacentFence(fence).addClass('placed');

        this.currentTurn.fencesRemaining--;

        switchPlayer();

    }

    aiMovePlayer = function(position){

        updatePlayerPosition(position);

        switchPlayer();

        checkWinner();

    }

    placeFenceForReal = function () {

        if (firstObjectSent != null) {

            var fence = firstObjectSent;

            $(fence).addClass('placed');

            getAdjacentFence(fence).addClass('placed');

            this.currentTurn.fencesRemaining--;

            switchPlayer();

            firstObjectSent = null;

        }

    }

    function setUpForSenendFence(fence) {

        //vertical (a/b - c/d)

        // a | b

        // c | d

        //horizontal (a/b - c/d)

        // a c

        // \_ \_

        // b d

        var a;

        var b;

        var c;

        var d;

        var fType;

        var fenceId = parseInt($(fence).attr('id').split('\_')[1]);

        var nextFence = getAdjacentFence(fence);

        var nextFenceId = parseInt($(nextFence).attr('id').split('\_')[1]);

        if (($(fence).hasClass("horizontal"))) {

            //alert("horizontal");

            a = (fenceId - (8 \* Math.floor(fenceId / 17)) - 8);

            b = a + 9;

            c = a + 1;

            d = b + 1;

            fType = "h";

        }

        else {

            //alert("vertical");

            a = fenceId - (8 \* Math.floor(fenceId / 17));

            b = a + 1;

            c = nextFenceId - (8 \* Math.floor(nextFenceId / 17));

            d = c + 1;

            fType = "v";

        }

        //alert("a:" + a + "b:" + b + "c:" + c + "d:" + d);

        //send json

        sendFence(fenceId, nextFenceId, a, b, c, d, fType);

    }

    function updateInformation() {

        Information.currentTurn.text(this.currentTurn.name);

        Information.player1FencesRemaining.text(this.player1.fencesRemaining);

        Information.player2FencesRemaining.text(this.player2.fencesRemaining);

    }

    function checkWinner() {

        if (this.player1.pos > 71) {

            alert("white won");

        }

        else if (this.player2.pos < 9) {

            alert("black won");

        }

    }

    function switchPlayerFirst() {

        if (this.currentTurn == this.player1) {

            this.currentTurn = this.player2;

        }

        else {

            this.currentTurn = this.player1;

        }

        updateInformation();

    }

    function switchPlayer() {

        if (this.currentTurn == this.player1) {

            this.currentTurn = this.player2;

            // ai turn:

            updateInformation();

            sendAi();

        }

        else {

            this.currentTurn = this.player1;

        }

        updateInformation();

    }

};

var Information = new function () {

    this.panel = $('<div id="info" />');

    this.currentTurn = $('<div id="currentTurn" />');

    this.player1FencesRemaining = $('<div id="player1FencesRemaining" />');

    this.player2FencesRemaining = $('<div id="player2FencesRemaining" />');

    this.getPanel = function () {

        this.panel.append(this.currentTurn);

        this.panel.append(this.player1FencesRemaining);

        this.panel.append(this.player2FencesRemaining);

        return this.panel;

    }

}

function Player(name, position, id, fences) {

    this.name = name;

    this.pos = position;

    this.id = id;

    this.fencesRemaining = fences;

}

### Index.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta name="author" content="Hadar Leibushor">

    <title>Quoridor</title>

    <!-- Import js -->

    <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.min.js"></script>

    <script src="js/ui.js"></script>

    <script src="js/sockets.js"></script>

    <!-- Import css -->

    <link rel="stylesheet" href="css/styles.css">

    <script> ui() </script>

</head>

<body>

    <div id="board"></div>

    <div>

        <input type="text" id="messageinput" />

    </div>

    <div>

        <button type="button" onclick="openSocket();">Open</button>

        <button type="button" onclick="send();">Send</button>

        <button type="button" onclick="closeSocket();">Close</button>

    </div>

    <!-- Server responses get written here -->

    <div id="messages"></div>

    <script> sockets() </script>

</body>

</html>

### Style.css

#board {

    width: 530px;

    height: 530px;

    float: left;

}

.square,

.fence {

    float: left;

}

.square {

    width: 50px;

    height: 50px;

    background-color: #742;

}

.fence {

    width: 10px;

    height: 50px;

    background-color: #c75;

}

.fence.selected,

.fence.placed {

    background-color: #cc5;

}

.fence.horizontal {

    height: 10px;

    width: 50px;

    padding: 0 5px;

}

.fence.horizontal.left {

    padding-left: 0;

}

.fence.horizontal.right {

    padding-left: 0;

}

#player\_1,

#player\_2 {

    height: 80%;

    width: 80%;

    margin: 10%;

    border-radius: 2em;

}

#player\_1 {

    color: black;

    background-color: #fff;

}

#player\_2 {

    color: white;

    background-color: #000;

}

#info {

    float: left;

}