

רכיבים ופרוטוקלים

# AI – Artificial Intelligence

AI, Artificial Intelligence, או, במילים אחרות, בינה מלאכותית, היא ענף של מדעי המחשב העוסק ביכולת לתכנת מחשבים באופן המציג יכולות המאפיינות את הבינה האנושית. הגדרה רחבה יותר לתחום זה: "לגרום למכונה להתנהג בדרך שהייתה נחשבת לאינטליגנטית לו אדם התנהג כך" – מרווין מינסקי.

בינה מלאכותית נחקרת בשתי רמות:

* בינה מלאכותית דמוית אנוש, שבה המחשב חושב ומסיק מסקנות באופן דומה לזה של המוח האנושי.
* בינה מלאכותית שאינה דמויות אנוש, שבה המחשב יגיע ליכולות עצמאיות לחשיבה, ללא קשר לדרך שבה המוח האנושי עושה זאת.

והיא מסווגת לפי ביצועים:

* בינה מלאכותית חזקה: שבה התוכנה אמורה לבצע חיקוי של פעילות השכל והחשיבה, באופן כללי: פתרון בעיות כללי, וכדומה.
* בינה מלאכותית חלשה: שבה התוכנה מיועדת לבצע פעילות נבונה בתחום יישום מוגדר: משחק שחמט, גילוי הוכחות בגיאומטריה, מומחיות ספציפית, וכדומה.

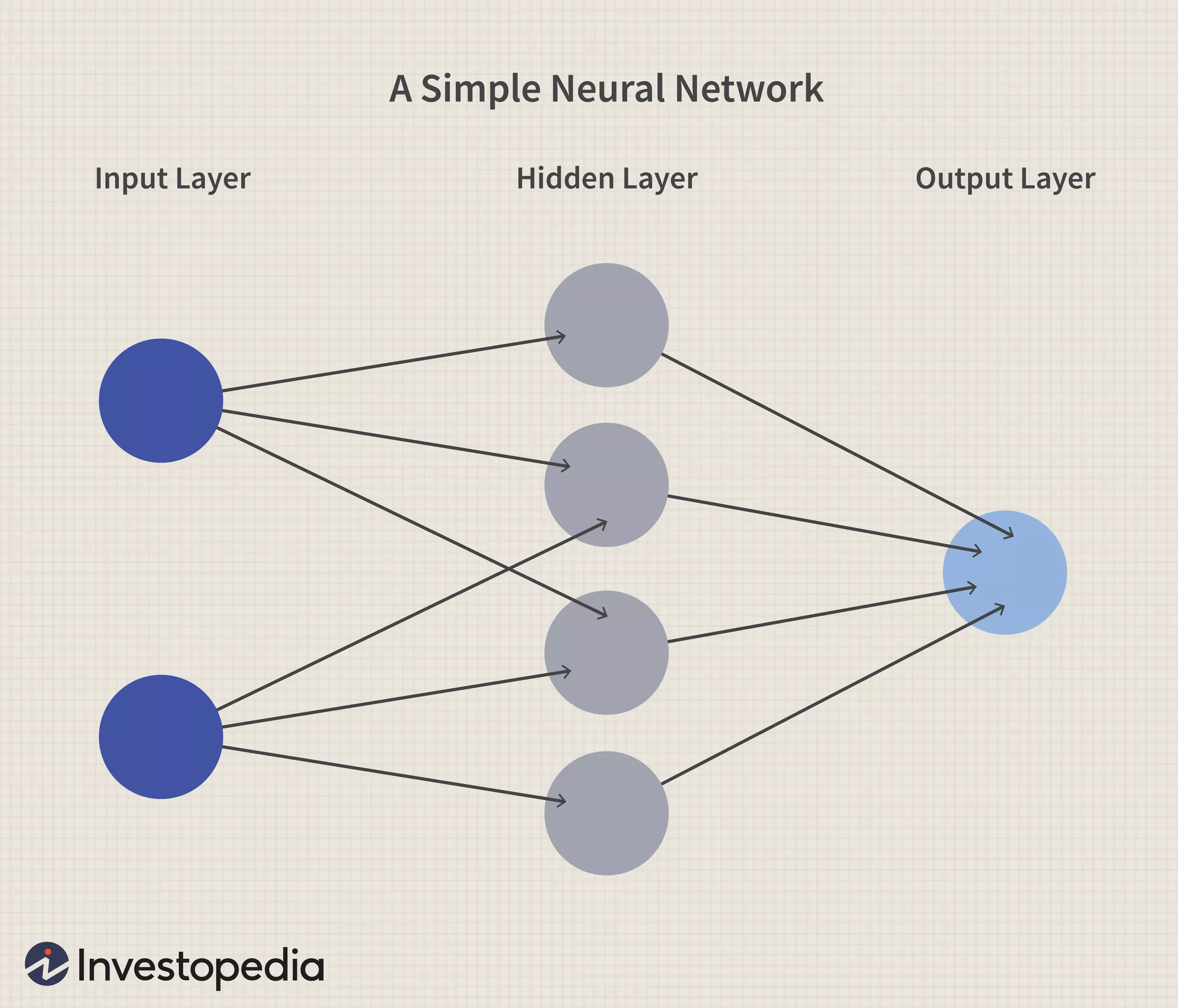
מכיוון שהניסיונות ליצור בינה מלאכותית חזקה נתקלו בקשיים רבים, עיקר ההתפתחות בחקר הבינה המלאכותית, כבר מאז שנות ה-80, הוא בתחום הבינה המלאכותית החדשה.

**בפרויקט שלנו נבנה "בינה מלאכותית חלשה" באחת משתי הדרכים הבאות:**

*דרך 1: Neural Networks*

*Neural Networks* זה סדרה של אלגוריתמים שמנסים למצוא יחסים בסיסיים בסט של מידע דרך תהליך שמדמה את הדרך שבה מוח אנושי עובד. בדרך זו, *Neural Networks* מתחבר בעצם למערכות של נוירונים, מלאכותיים או טבעיים. יש ביכולת של *Neural Networks* לאמץ קלט משתנה, ובכך ה-*network* מייצרת את הפלט הטוב ביותר ללא צורך בשינוי הקריטריונים של הפלט.

דוגמא:



*כיום, Neural Networks משומשים לרוב בכלכלה, יזמות, אנליזה של עסקים, ומניות.*

דרך 2: *Minimax Algorithm with Alpha-Beta Pruning*

קודם כל, מהו Minimax Algorithm?

*ה-Minimax Algorithm הינו אלגוריתם המתאים למשחקים אשר בנויים על העיקרון של סט חוקים קבוע ומוגדר היטב, לדוגמא: שחמט ודמקה. בתורת המשחקים, Minimax Tree הינו עץ הפורס את האפשרויות למשחק של שחקן א', את התגובות של שחקן ב' לכל פעולה של שחקן א', את תגובותיו של שחקן א' לתגובותיו של ב' וכך הלאה. מעשית, המחשב מגביל את עומקו של העץ שכן ישנה מכסת זיכרון וזמן.*

*העלים שנוצרים בעץ הם מצבים סטטיים אליהם נגיע לאחר רצף של מהלכים. לכל מצב סטטי כזה ניתן ציון, שמשקף כמה המצב טוב מבחינת המתכנת.*

*כמובן שאם נבחר במהלך (ענף) המוביל לעלה עם הציון הגבוה ביותר, לא מובטח לנו שאכן נגיע לאותו עלה. העלה הטוב ביותר הוא התסריט האופטימלי מבחינתנו, וניתן להניח שיריב לא יוביל אותנו בהמשך הנתיב אלא יסיט אותו לנתיב הטוב ביותר מבחינתו.*

כיצד עובד ה-Algorithm?

*לאחר שאנחנו יצרנו לנו את העץ, אנחנו נסרוק אותו החל בעלים, דרך הקודקודים (קודקוד הוא צומת פנימי בעץ) שמעליהם עד השורש (מלמטה למעלה). הציון שניתן לכל קודקוד הוא הערך הגבוה ביותר של העלים \ הקודקודים שתחתיו, אם אותו הקודקוד מסמל מהלך שלי, והערך הנמוך ביותר של העלים \ קודקודים שתחתיו אם אותו קודקוד מסמל מהלך של היריב.*

*כאשר נגיע לשורש העץ, נבחר בקודקוד שמתחתיו בעל הציון הגבוה והטוב ביותר.*

בעיה קשה בבניית עץ Minimax היא הזיכרון הרב שהוא צורך. מספר הקודקודים שיש לפתח עולה בטור הנדסי ככל שנעמיק את החיפוש. קיימת שיטת "גיזום" המבטלת בנייה של ענפים שברור לנו עוד בשלב מוקדם כי הם לא מועילים לחיפוש שלנו. ניתן כך להקטין את מספר הצמתים בעץ לשורש המספר שהיה מתקבל ללא הגיזום (במקרה הממוצע), ושיטה זו היא ה-*Alpha-Beta Pruning*

מה זה Alpha-Beta Pruning?

*בהתאם לנאמר למעלה, שיטה זו משפרת רבות את ה-Minimax Algorithm בכך שמבטלת את הצורך לחפש חלקים רבים מהעץ באמצעות יישום ה-Branch-and-bound technique.*

*בצורה מדהימה, שיטה זו עושה זאת ללא האופציה שתפספס מהלך טוב יותר!*

*ב-Algorithm ישנם שני משתנים אשר נשמרים, alpha ו-beta. הם מבטאים את הציון המינימלי שיש ל-maximizing player, ואת הציון המקסימלי שיש ל-minimizing player מקבל.*

דוגמא לקוד:

int alphaBetaMax( int alpha, int beta, int depthleft ) {

if ( depthleft == 0 ) return evaluate();

for ( all moves) {

score = alphaBetaMin( alpha, beta, depthleft - 1 );

if( score >= beta )

return beta; // fail hard beta-cutoff

if( score > alpha )

alpha = score; // alpha acts like max in MiniMax

}

return alpha;

}

int alphaBetaMin( int alpha, int beta, int depthleft ) {

if ( depthleft == 0 ) return -evaluate();

for ( all moves) {

score = alphaBetaMax( alpha, beta, depthleft - 1 );

if( score <= alpha )

return alpha; // fail hard alpha-cutoff

if( score < beta )

beta = score; // beta acts like min in MiniMax

}

return beta;

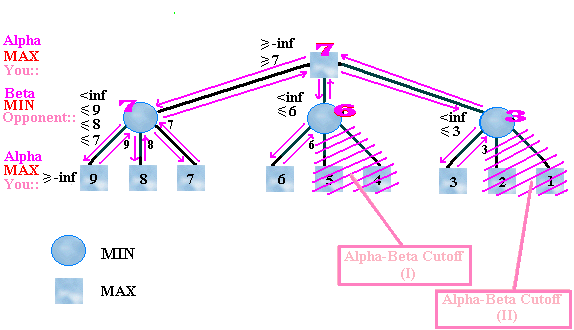
}

// This is the call from the Root:

score = alphaBetaMax(-infinity, +infinity, depth);

// you define depth, -infinity and +infinity are also known

// as integer.MAX\_VALUE and integer.MIN\_VALUE in Java.

תמונה המדגימה את האלגוריתם:

# FPGA



FPGA, או Field Programmable Gate Arrays הוא סוג של מעגל משולב, אשר ניתן להגדיר ולשנות את תפקודו לאחר הייצור, בתהליך תכנות הדומה להתקנת תוכנה. הגדרת תפקוד רכיב זה מתבצעת על ידי הזנה של תרשים לוגי או על ידי שפות לתיאור מעגלים (במקרה שלנו, VHDL).

### עיקרון הפעולה של ה-FPGA:

ה-*FPGA* מורכב מיחידות לוגיות אשר ניתנות לתכנות ומרשת של אמצעים לחיבור ולניתוק בין היחידות השונות. בטעינת ה-*FPGA* מקבלות היחידות הגדרה לאופי פעולתן, והיחידות מתחברות ליחידות אחרות, בצורה שמממשת את המעגל הלוגי הכולל. רכיבי *FPGA* יכולים לשמש למגוון פונקציות, החל מלוגיקות פשוטות כגון שערים לוגיים עד ליכולות מורכבות כגון דלגלגים ומרחבי זיכרון.

### שימושיו של ה-FPGA:

* *מערכות על שבב (מעגל משולב המכיל בתוכו את רוב תפקידיו של מחשב שלם).*
* *מעגלים שנועדו לבצע משימות בעלות אותו אופי, אך שונות אחת מהשנייה בלוגיה הפנימית ולא בממשקים החיצוניים ומיוצרים בכמויות קטנות.*
* *תחומים הדורשים עיבוד מקבילי ופיתוח הצפנות.*
* *פתח אפשרות לסוג חדש של עיבוד.*

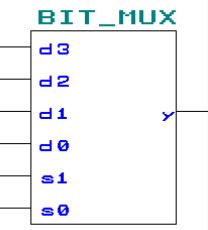
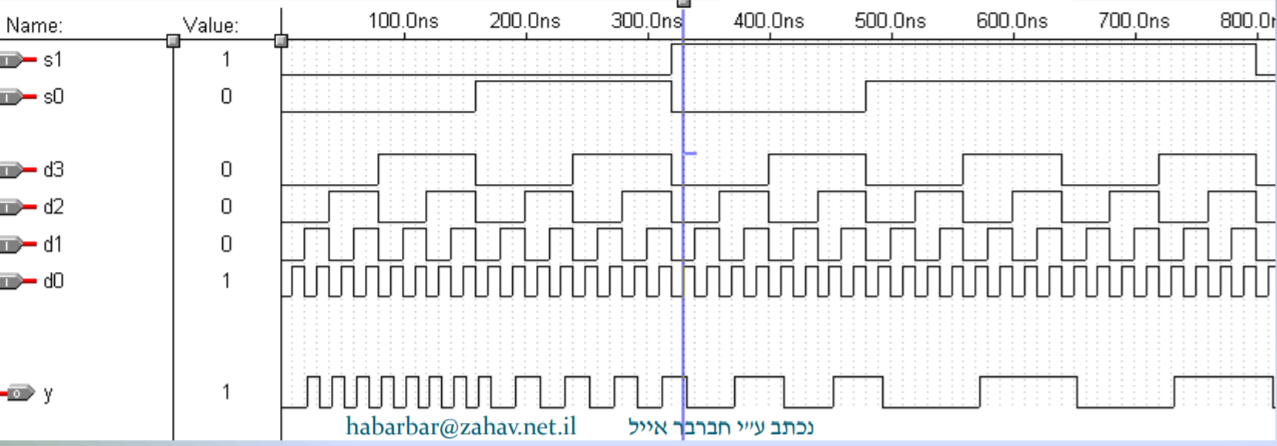
### היתרונות בשימוש ב*-FPGA*:

* *ניתן לעדכן את התוכנה של רכיבים אלו גם לאחר שלב הייצור ואפילו לאחר שהמוצר מגיע אל הלקוח.*
* *שלב התכנון של מעגל אלקטרוני נהפך לקל יותר ופשוט יותר, והוא כולל בעיקר תכן של שילוב רכיב ה-FPGA במעגל מודפס.*
* *ניתן להתחיל לייצר את המעגלים המודפסים לפני שהלוגיקה נכתבה או הושלמה.*
* *ניתן לחלק את העבודה בין צוותים שונים, האחד עובד על תכנון המעגלים הפיזיים, והשני על הלוגיים שיטענו לתוך ה-FPGA. לחילופין, ניתן אף לחלק את זמן הפיתוח של צוות בודד, שראשית מתכנן את המעגל המודפס, ובזמן שזה נמצא בתהליכי ייצור, כותב את הלוגיקה עבור ה-FPGA.*
* *תהליך הפיתוח מהיר ביותר שכן אין צורך בהליך ייצור כלשהו ולכן מתאים מאוד לפיתוח של אב-טיפוס.*

### תכנות ה-:*FPGA*

כדי להחליט על התנהגות ה-*FPGA,* יש על המתכנת, או המשתמש, לכתוב תוכנה בשפת תיאור חומרה או בתור תרשים סכמתי. שפות תיאור החומרה הנפוצות ביותר הן VHDL ו-Verilog. דוגמאות לקטעי קוד בשפות תיאור החומרה האלו:

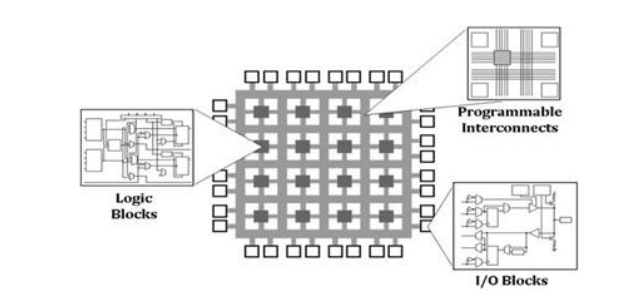
מימוש MUX בעזרת VHDL:



### דוגמא לבלוק הלוגי אשר נמצא בתוך ה-FPGA:



### דוגמא ל-*FPGA*:

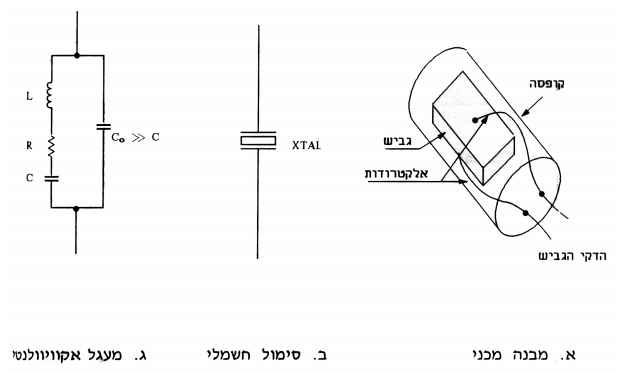


# מתנד גבישי – Crystal Oscillator



*מתנד גבישי הוא מתנד חשמלי, המתבסס על גביש פיאזואלקטרי\* ומגבר (לרוב מגבר שרת), ליצירת תנודות חשמליות בדיוק רב בגורם איכות\* (Q) גבוה מאוד. תדר זה משמש לייצוב תדר במערכות קליטה ושידור בתדרי רדיו, שעונים, מחשבים ומערכות משובצות מחשב, ועוד. הגביש הנפוץ ביותר הוא גביש "קווארץ".*

*למתנד הגבישי צורת תיבה, והוא נתון בין שתי אלקטרודות משני צדדיו. כל המעגל נמצא בתוך אריזה. ניתן לראות זאת באיור הבא:*



### עקרון הפעולה של הגביש

*כאשר הגביש מעורר על ידי מתח חיצוני בין האלקטרודות, הוא רוטט (מתנדנד) מבחינה מכנית בדיוק כמו מטוטלת\*. תוך כדי הריטוט, הגביש מהווה עכבה חשמלית\* המחוברת בין האלקטרודות, מעגל שווה ערך שלה נראה באיור למעלה (ג').*

### היתרון המרכזי של מתנד גביש*י*

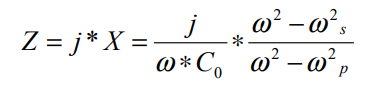
*היתרון הגדול ביותר של מתנד גביש על פני מתנדים אחרים הוא יציבות התדירות שלו.*

*יציבות זו מוסברת כך:*

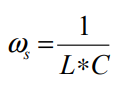
* *במתנדי LC (סליל וקבל) נקבעת התדירות על ידי ההשראות של הסליל* נקבעת התדירות על ידי ההשראות של הסליל L והקיבול של הקבל C . גדלים אלה משתנים בסדר גדול של כמה עשרות ppm עם שינוי טמפרטורה במעלה אחת. שינויים אלה גורמים לחוסר יציבות בתדירות המתנד. לעומת זאת, במתנד גבישי, תדירות התנודות שווה בקירוב רב לתדירות הריטוט של הגביש עצמו. תדירות זו יציבה ביותר. על כן, במתנד גבישי טיפוסי, יציבות התדירות טובה פי 100 ויותר מזו של מתנד LC רגיל .
* במתנדי LC מלבד הסלילים והקבלים היוצרים מעגל תהודה (מצב של מערכת פיזיקלית הנתונה להשפעת כוח חיצוני, אשר מתקבל כאשר הכוח פועל בתדירות קרובה לאחת מתדירויות התנודה הטבעיות של המערכת) שקול, שאר העכבות במעגל הן : התנגדויות אוהמיות טהורות. מבחינה מעשית, אין זה כך בגלל הקיבולים הפנימיים של הטרנזיסטור וכן הקיבולים וההשראויות בין רכיבי המעגל ובין עצמם , או בינם ובין הלוח שעליו הם בנויים (סוג זה של קיבולים נקרא " קיבולים תועים "capacitances stray - גורמים לכך שבתדירויות גבוהות, מתנד ה LC לא יתנדנד בדיוק בתדירות התהודה של מעגל התהודה השקול אלא יסטה ממנה במקצת. אפשר להוכיח שסטייה זו קטנה ככל שה-Q של המעגל גדול יותר .(

ניתוח פעולת הגביש בצורה מפורטת

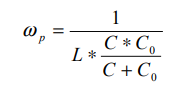
*נשתמש באיור החשמלי שלו (האיור לדלעיל), נחשב את עכבת הגביש (נזניח את R):*



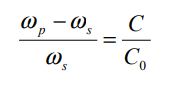
*כאשר Ws מייצג את תדירות התהודה הטורית של הגביש:*



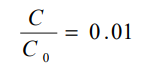
*ו Wpמייצג את תדירות התהודה המקבילית של הגביש:*



*לפי הביטויים הנ"ל אפשר להראות שמתקיים:*

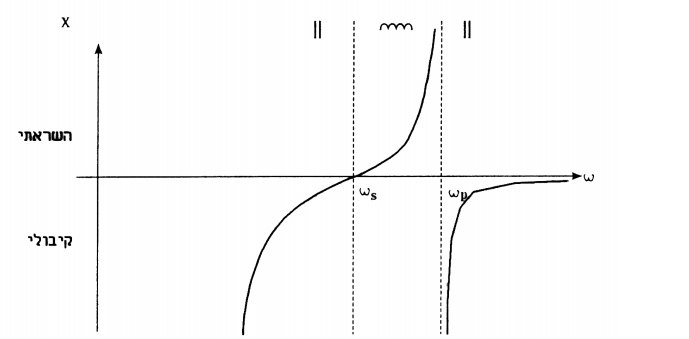


*כאשר:*



*הוא ערך טיפוסי בגבישים.*

### עקום ההיגב של הגביש בקירוב:



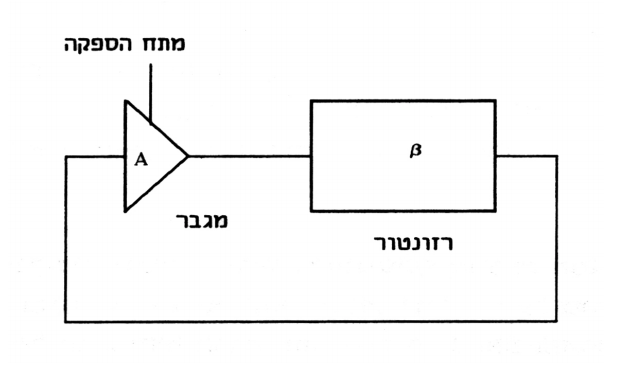
### תנאים לתנודות

*על מנת להבין מתי ובאיזה תנאים יכולים להיווצר תנודות יכולים להיווצר תנודות יש להבין קודם כל את המבנה העקרוני של מתנד. נוכל להגדיר מתנד אלקטרוני כמעגל ללא כניסה אשר מכיל מגבר ורשת פסיבית. עיקרון הפעולה הוא שאות הנוצר במקום מסוים במעגל יעבור הנחתה סלקטיבית ומגבר המפצה על ההפסדים, כך שבסופו של דבר האות ישחזר את עצמו הן מבחינת הגודל והן מבחינת המופע.*

*כל מתנד הוא מערכת סגורה המורכבת מ-2 חלקים:*

1. *מגבר*
2. *רזונטור (רשת פסיבית)*

*כפי שמתואר באיור הבא:*



*במבט ראשון נראה המעגל כ"יש מאין" מכוון ללא כניסה, נוכל להסביר תופעות היווצרות התנודות בצורה הבאה:*

*בזמן הפעלת מתח \ מתחי הספקה למעגל נוצרות תופעות מעבר בכל התדרים או פשוט רעש לבן מפולג בצורה אחידה בכל התדרים. רעש זה אשר נוצא עובר דרך מעגל הרזונטור המנחית את האותות בצורה סלקטיבית. ההנחתה של הרזונטור היא פונקציה של תדר אות הכניסה. בנוסף לכך, הרזונטור משנה את המופע בהתאם לתדר האות.*

*המגבר A יכול להיות לא הופך מופע, כלומר יוצר זווית של 0 מעלות בין מתח הכניסה ומתח היציאה, ואז נדרוש שגם הזזת המופע ברזונטור תהיה גם 0 מעלות בתדר התנודות המסומן כ- f0.*

*המגבר יכול להפוך מופע בזווית 180 מעלות, ואז הרזונטור יוצר הפרש מופע של 180 מעלות בתדר תנודות f0.*

*האות היוצא מן הרזונטור מונחת ומוזז בהתאם לתדר שלו. האות נכנס למגבר A המחזיר את ההגבר ש"אבד" לאות בתדר f0. בצורה זאת אות בתדר f0 שחזר את עצמו לאחר מעבר דרך המעגל. אות בתדר שונה נשאר מונחת ומוזז פאזה, ולכן לאחר מספר מעברים דרך המעגל – האות ייעלם. במעגל נשאר רק האות בתדר f0.*

*מתוך תיאור המנגנון של מתנד סינוסי ניתן לקבוע 2 קריטריונים ראשיים להיווצרות התנודות:*

1. *הגבר החוג הוא 1 עבור אותות בתדר התנודות f0.*
2. *הזזת מופע של המתנד היא 0.*

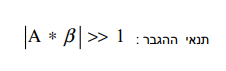
*שני הקריטריונים הנ"ל ידועים כתנאי Barkhausen לתודות. אם נחזור לאיור הקודם נוכל לסכם את התנאים לתנודות באמפליטודה קבועה:*

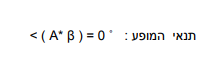




*כאשר רוצים לבנות מתנד לגל ריבוי, עלינו לדאוג שהיציאות של המגברים ינועו בין רוויה חיובית ורוויה שלילית כלומר נדרוש שהאותות יכנסו לרוויה. על מנת ליצור גלים רוויים עלינו לדרוש שההגבר של החוג יהיה הרבה יותר גדול מ-1.*

*אם כך, התנאים להיווצרות גל ריבועי הם:*

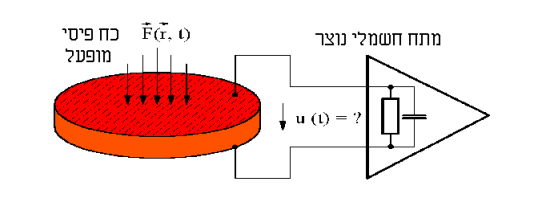




### מושגים

פיאזואלקטריות

*תכונה של גבישים וחומרים קרמיים מסוימים לייצור מתח חשמלי כתגובה ללחץ מכני. התכונה הפיאזואלקטרית היא דו-כיוונית, וגבישים יכולים לשנות קלות את צורתם כתוצאה מהפעלת מתח חשמלי גבוה עליהם (השינוי זעיר).*



גורם איכות \ מקדם איכות

*מקדם האיכות, או מקדם Q, הוא פרמטר חסר ממדים המגדיר את מידת הריסון של מתנד הרמוני מרוסן. ככל שמקדם זה גדול יותר, המתנד מרוסן פחות ומתנודד במשך מחזורי תנודה רבים יותר.*

מטוטלת

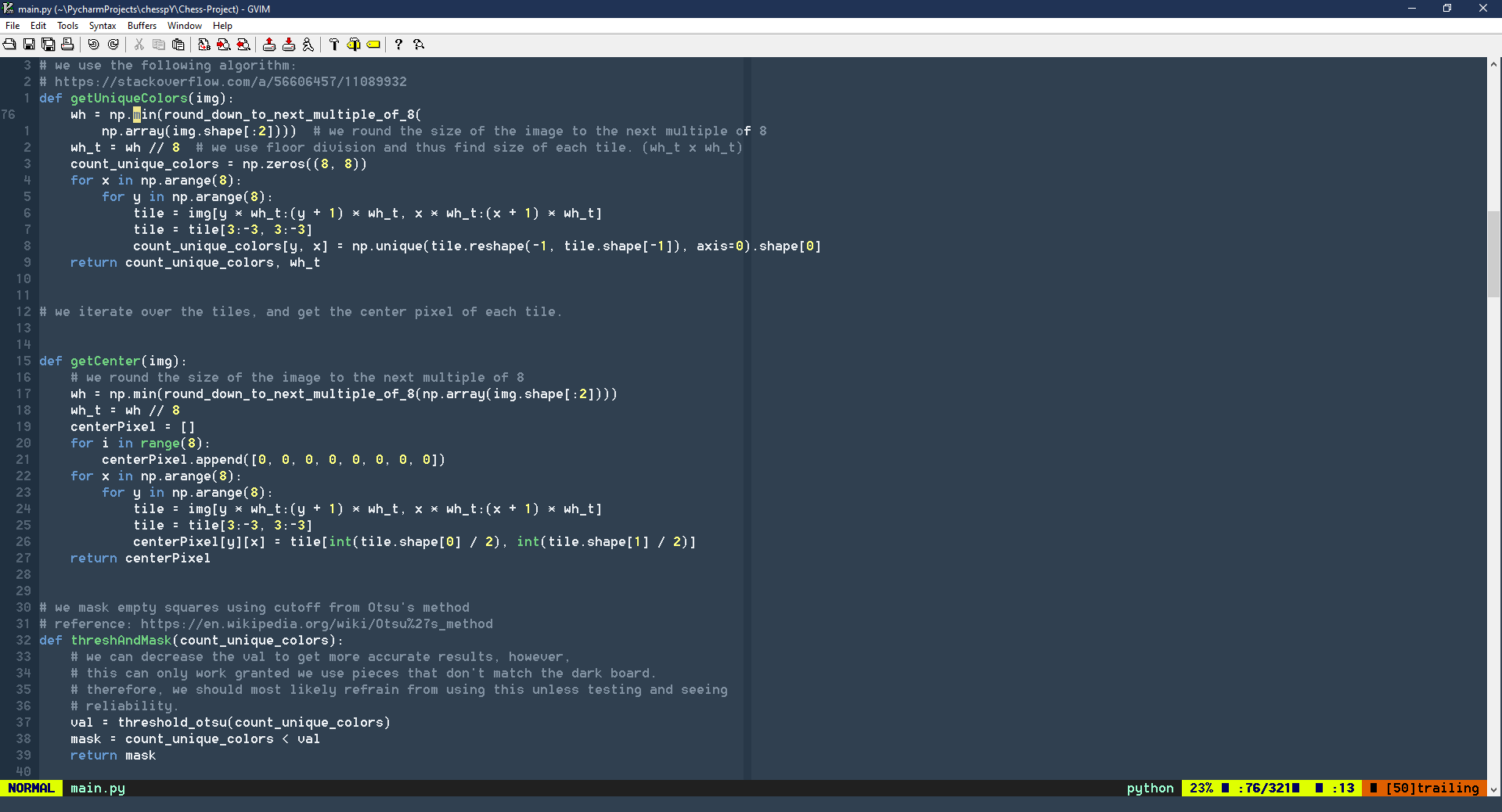
*מטוטלת היא משקולת אשר תלויה מציר כך שהיא יכולה לנוע באופן חופשי. כאשר מטוטלת ממוקמת לצד נקודת המנוחה שלה (אשר מכונה שיווי משקל מכני), הכבידה מפעילה עליה כוח מחזיר אשר יאיץ אותה בחזרה אל עבר נקודת שיווי המשקל ויגרום לה להתנדנד סביבה.*

עכבת המעגל

*עכבה זו היא ההתנהגות הכוללת במעגל. המושג נפוץ במיוחד במשוואות מתמטיות של מעגל חשמלי שבו זורם זרם חילופין. שני הגדלים (עכבה והתנגדות) נמדדים באותן יחידות, אוהם - Ω.*

# סביבות העבודה

## Vim – Vi iMproved



Vim, היא תוכנת עריכה טקסט חופשית, אשר מהווה הרחבה של העורך vi של Unix. Vim יצאה לאור במקור על ידי המפתח הראשי שלה (בעבר ובהווה) בראם מולינאר בשנת 1991.

Vim היא אחת משתי תוכנות עריכת הטקסט הפופולריות ביותר בקרב מתכנתים במערכות יוניקס ולינוקס, לצד Emacs. קיימות גרסאות של Vim לרוב מערכות ההפעלה המודרניות. בניגוד לעורכים אחרים, ב-Vim (וב-Vi) מצבים שונים של עריכה, פקודות לחיצה, ושורת פקודה. אוסף הפעולות ניתן להרחבה באמצעות כתיבת 'תסריטים' ו'קובצי צביעה' לשפות תכנות שונות.

כדוגמא לכוחה של תוכנת Vim, אביא דוגמאות בהן אני משתמש לרוב כאשר אני עורך בתוכנה זו:

\* לצורך העניין, ב-configuration שלי ב-vim, אני עובר מ-Normal Mode ל- Insert Mode ולהפך על ידי לחיצה של הצירוף: *jk \**

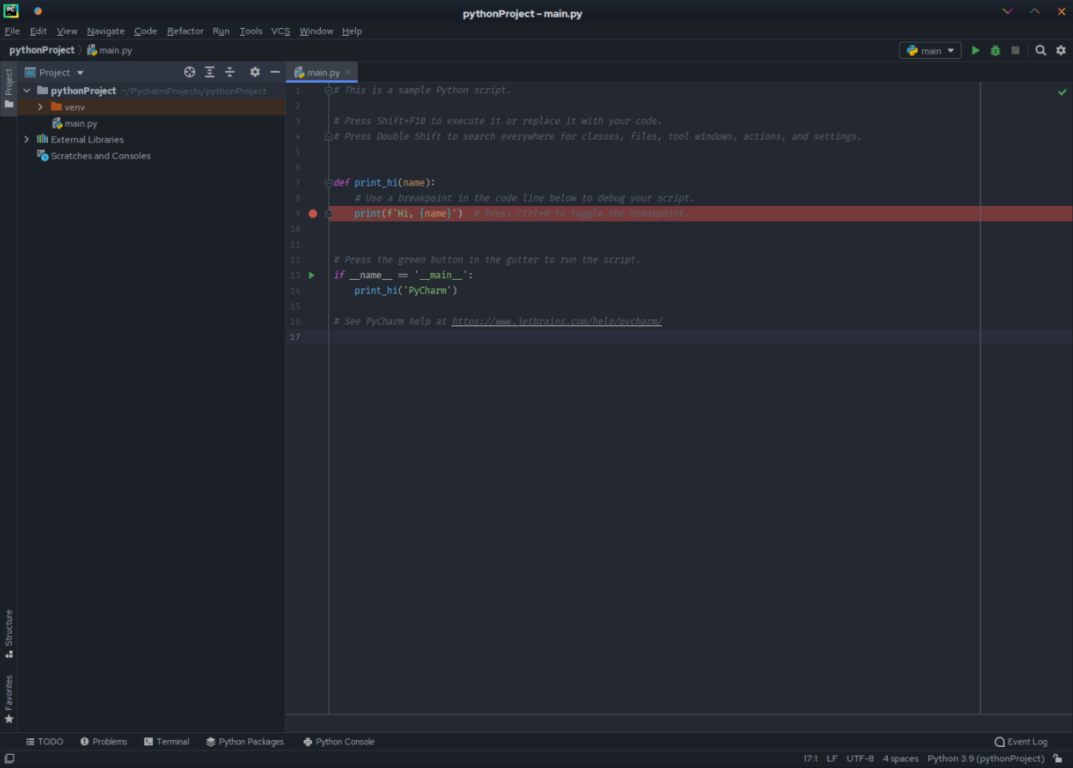
* כאשר אני עומד על מילה ומעוניין להחליפה, ברוב ה-Editors יש עליך ללחוץ פעמיים על המילה למחוק אותה ולאחר מכן לכתוב, זו בניגוד ל-Vim, בה העריכה פשוטה יותר. ב-Vim, יש עליי לכתוב את הצירוף הבא: *jkciw,* כלומר: *jk –* יציאה מ-Insert Mode, *c* – שינוי, *iw* – המילה הפנימית, או inner word.
* כאשר אני עומד על שורה מסוימת וברצוני למחוקה, אוכל לכתוב את הצירוף: *jkdd,* כלומר: *jk –* יציאה מ-Insert Mode, *dd* – מחיקת השורה בה אני נמצא. (שימושים נוספים יכולים להיות *diw – Delete Inner Word*, *d$ - Delete Until the End of the Line, dw – Delete Word ועוד הרבה).* נוסף על כך, בשביל למחוק *n* שורות, לדוגמא, יש עליי לכתוב: *jkdnd.*
* כאשר אני מעוניין להתחיל ולכתוב משהו בסוף השורה בה אני עומד, כל שעליי לעשות זה לכתוב את הצירוף הבא: *jkA,* כלומר: *jk –* יציאה מ- Insert Mode, A – Append At End of Line.
* ועוד הרבה.

כלומר, ישנם קיצורים רבים ב-Vim בעזרתם ניתן לערוך ולכתוב קוד בצורה מהירה ויעילה, זו לאחר שאתה לומד את Vim בצורה טובה, וזה לוקח זמן. **יש כבערך 2000 פקודות וקיצורי דרך ייחודיים ב-Vim בעזרת זיכרון של כ-30 key mappings!**

*מדוע שאשתמש בסביבה זו?*

הסיבות פשוטות למדי. סביבה זו מקנה לך מהירות ויעילות שלא ניתן להשיג בשום מקום אחר, וזו בעוד שאתה כותב את הקוד של עצמך ללא שימוש בהשלמות ובעזרה חיצונית שיש ב – IDE. נוסף על כך, סביבה זו מקנה לך את האפשרות להפוך אותה לשלך. ניתן לעשות configure אישי לכל דבר שם ובכך ליצור לעצמך את הסביבה הייחודית שלך, בה כל דבר נוח לך ומותאם לך. ולאחרונה, ישנם Plugins כמעט לכל צורך שיש לך ב-Vim, דבר אשר גם עוזר בדברים רבים.

## PyCharm by JetBrains – תשומש לכתיבה ועריכת קוד עיבוד התמונה והבינה המלאכותית

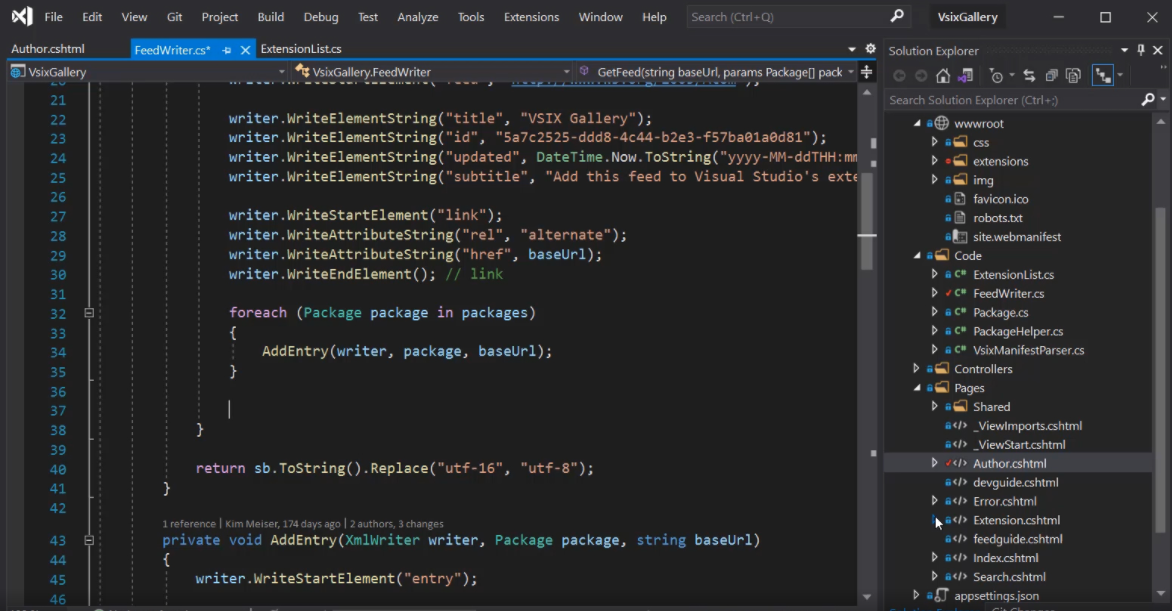


PyCharm היא סביבת פיתוח משולבת לפיתוח תוכנות בעיקר בשפת *Python*, אשר פותחה על ידי תאגיד התוכנה הצ'כי JetBrains. תוכנה זו כתובה ב-*Java ו-Python,* והגרסה היציבה הראשונה שלה יצאה בפברואר 2010. הסביבה מספת שירותים כגון ניתוח קוד, Debugging, תומכת בסביבת אינטרנט באמצעות *Django* וכן ב-*Data Science* באמצעות *Anaconda.*

*מדוע שנשתמש בסביבה זו?*

אנו נשתמש בסביבה זו לרוב להרצה ול-Debugging של הקוד שלנו, ואביתר ישתמש בה גם בכתיבת הקוד. סביבה זו מאוד נוחה במציאת בעיות ובמיוחד בהרצה ובניתוח התוצאות. נוסף על כך, סביבה זו גם מאוד נוחה לשימוש עבור כתיבת הקוד שכן מספקת כלים כגון *Autocomplete, Reformatting* ועוד אשר הופכים את כתיבת הקוד לקלה ונעימה יותר.

## Visual Studio – תשומש לכתיבה ועריכה של משחק השחמט



Visual Studio היא סביבת פיתוח מבית החברה Microsoft, אשר מאפשרת למתכנתים לפתח תוכניות מחשב ואתרי אינטרנט אינטראקטיביים. הגרסה הרשמית האחרונה הינה של שנת 2019, אשר מיועדת לפתח יישומים עבור סביבות Win32 ואפליקציות של Windows 10 וחנות האפליקציות של Windows 8, עבור .NET Framework וכן עבור NET Core. .

ב-Visual Studio ניתן לתכנת בשפות הבאות:

* C++
* C#
* VB.NET
* J#
* F#
* TypeScript

תוכנה זו מהווה סביבת פיתוח משולבת, שכן מוטמעים בה כלים שונים לפיתוח מהיר וקל של יישומים. הרכיב העיקרי של הסביבה הוא עורך הטקסט שצובע את המילים לפי ההקשר התכנותי שלהן. לעורך IntelliSense – תכונת השלמה אוטומטית אשר מציגה חלונית עם שמות העצמים הקשורים לכתיבה ומאפשרת כתיבה מהירה ומניעת טעויות ושגיאות.

בין היתר, מוטמעים כלי פיתוח חזותיים שונים שמחוללים קוד בצורה אוטומטית, כמו מעצבים חזותיים למשקי משתמש Winform ו-WPF, וכלים שמאפשרים נגישות והתמצאות מרחבי הפרוייקט כמו סייר הפרוייקט לעיון בקבצי הפרוייקט. גם כלי Debugging מוטמעים בסביבה זו.

בסביבה זו ניתן גם לשלב בפתרון אחד (Solution) שפות תכנות שונות, וכן להעביר רכיבי תוכנה משפה אחת לשנייה, וכן למחזר קוד משפה אחת לאחרת.

*מדוע שנשתמש בסביבה זו?*

אנו נשתמש בסביבה זו בשביל כתיבה, עריכה והרצה של קוד השחמט שלנו אשר נכתב ב-C++. כדומה לסיבות שאנו משתמשים ב-PyCharm, אנו משתמשים בסביבה זו שכן היא נוחה ביותר בעקבות הכלים אשר מוטמעים בה כגון כלי ה-Debugging ו-Intellisense, וכי היא מותאמת עבור שפת C++. נוסף על כך, סביבה זו יכולה גם לשלב ב -Solution אחד שפות תכנות שונות.

## Arduino IDE – תשומש לתכנות ה-ESP בשביל תקשורת ותזוזת האלקטרומגנט

ה-*Arduino Integrated Development Environment* היא תוכנה מרובת פלטפורמות (מתאימה ל-*Windows, Linux, Mac..*) שכתובה בעזרת פונקציות מ-*C* ו-*C++*. תוכנה זו משומשת עבור כתיבה והעלאה של תוכניות לכרטיסים (או, *Boards*) שמותאמים ל-*Arduino* וגם, בעזרת עזרה של אנשים אחרים ותמיכה של הקהילה, כרטיסים של יוצרים שונים.

ה-*IDE* מספק מגוון פונקציות שונות מהשפה *C++*,ואף פונקציות ומשתנים שונים ונוחים יותר מהשפה (לדוגמא, String). הסביבה ממירה את התוכנות שכותב המשתמש מקוד שניתן להרצה לקובץ טקסט בקידוד הקסאדצימאלי אותו טוענים לתוך הלוח באמצעות *Loader Program* בתוך ה-*firmware* של הלוח.

*מדוע שנשתמש בסביבה זו?*

הסיבות מדברות בפני עצמן. תוכנה זו נוחה ביותר לשימוש עבור תכנות לוחות *Arduino*, או במקרה שלנו, *ESP*. הסיבה לכך היא השפה הנוחה בעזרתה כותבים בתוכנה זו. שפה זו דומה מאוד ל-C++ (אנו אפילו מחשיבים אותה ל-C++), אך יש בה פונקציות ומשתנים נוספים כדי להקל על המשתמש. דוגמאות לכך מונות את String, digitalWrite ועוד.

# שפות התכנות

## Python – תשומש עבור Image Recognition and Processing ועבור ה-AI



*Python*, או פייתון, היא שפת תכנות דינמית (בעלת פעולת הידור אשר קורית לרובה במהלך הריצה) מהנפוצות ביותר. פייתון תוכננה תוך כדי שימת דגש על קריאות הקוד, וכוללת מבנים המיועדים לאפשר ביטוי של תוכניות מורכבות בדרך קצרה וברורה. אחד המאפיינים הבולטים בתחביר השפה הוא השימוש בהזחה להגדרת בלוקים של קוד (ללא שימוש בסוגריים או במילים שמורות כרוב שפות התכנות הנפוצות).

פייתון הינה שפה מרובת-פרידגמות (סט מוסכמות לכתיבת תוכנה) אשר מאפשרת תכנות מונחה-עצמים (פרדיגמת התכנות המשתמשת ב"עצמים", כלומר, אובייקטים, לשם תכנות תוכניות מחשב), תכנות פרוצדורלי (פרדיגמת תכנות המגבירה את מודולריות קוד המקור ומקלה על התחזוקה) ובמידה מסוימת גם תכנות פונקציונלי (פרדיגמת תכנות השמה דגש על חישוב ביטוי תוך שימוש בפונקציות ההפשטה העיקריים). לשפה ספרייה סטנדרטית גדולה וענפה, והיא תומכת באופן מובנה בהרחבה שלה אל שפות אחרות.

לשפה שתי גרסאות ראשיות: *Python 2 ו- Python 3.* הגרסה השנייה (Python 2) איננה בפיתוח יותר (למעט תיקון באגים) והתמיכה בה הסתיימה בשנת 2020.

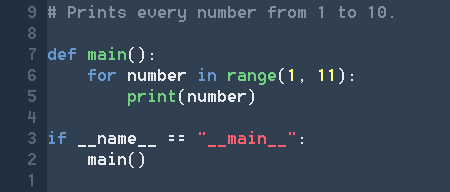
שם השפה *Python* נבחר כמחווה לקבוצה הקומית "מונטי פייתון".

אדגים קטעי קוד פשוטים בשפה:

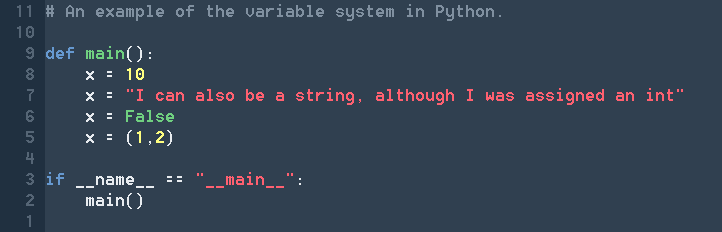
*תוכנית בסיסית המדפיסה Hello, World!:*



*לולאת for בסיסית אשר מדפיסה את כל המספרים מ-1 עד 10:*



*הצבת ערכים במשתנים בPython:*



## C++ – תשומש עבור משחק השחמט

++C היא שפת תכנות מרובת פרדיגמות (הוסבר למעלה) המבוססת על שפת התכנות C, שפותחה בשנות ה-80. שפה זו מיישמת עקרונות של תכנות פרוצדורלי (הוסבר למעלה), תכנות מונחה-עצמים (הוסבר למעלה) ותכנות גנרי (סגנון פיתוח תוכנה בו אלגוריתמים נכתבים במונחים של טיפוסים שיוגדרו המשך, וסוג הטיפוס יוגדר על פי הצורך ואז יתבצע שימוש באלגוריתם עבור הטיפוס המסוים שנקבע). שפה זו היא מהשפות הפופולריות בקרב מתכנתים עד היום, ושפות פופולריות אחרות כגון Java ו – C# הושפעו ממנה במידה רבה מאוד.

כשפה מונחית עצמים, שפה זו תומכת בעקרונות כימוס, ירושה ופולימורפיזם. מטרת כלים אלו היא לפשט את מבנה התוכנה, לאפשר שימוש חוזר בחלקי תוכנה קיימים ולהקל על תהליך הפיתוח. שימוש נכון בהם מאפשר לזהות שגיאות כבר בשלב ההידור (הקומפילציה) ולחסוך את הצורך בגילוין ותיקונן בשלבים מאוחרים יותר של תהליך הפיתוח. נוסף על כך, בניגוד לשפות מודרניות כגון C# או Java, ++C תומכת בירושה מרובה.

++C תוכננה כך שהיא שומרת על תאימות לאחור עם השפה C במידה רבה. ולכן, ברוב המקרים קוד אשר כתוב בשפת C עלול לרוץ גם בעזרת קומפיילר של שפת ++C, בעזרת שינויים מינוריים או ללא שינויים כלל. שפה זו, כמו C, מהודרת לרוב ישירות לשפת מכונה, ומקפידה לאפשר גישה ישירה אל זיכרון המחשב. עיקרון חשוב בשפה הוא שליטה מלאה של המתכנת על הפעולות שמתבצעות בזמן ריצה. עיקרון זה שולל הפעלה של מנגנון איסוף זבל (למרות שישנה אפשרות כזו בה נעשה שימוש לעיתים רחוקות). ולכן, ניהול משאבי מערכת מתבצע לרוב בטכניקה שנקראת RAII, בה ניהול של משאב מתנהל דרך אובייקט המוקדש לכך: "מצביעים חכמים" או אוספים כגון Vector.

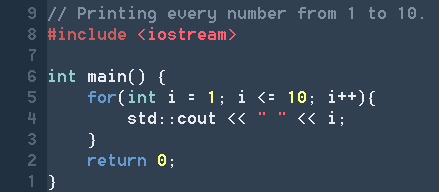
שפה זו כוללת גם מערכת טיפוסים סטטית מורכבת, שתפקידה לגלות שגיאות בזמן הידור התוכנית ולאפשר יצירת תוכניות יעילות יותר.

אדגים קטעי קוד פשוטים בשפה:

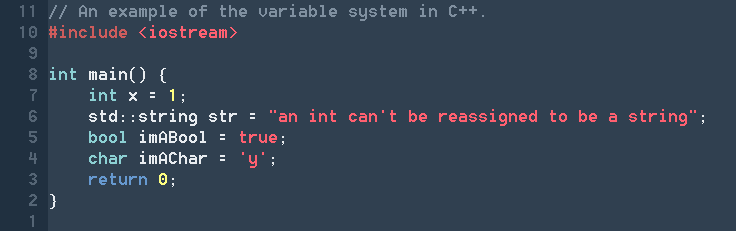
*תוכנית בסיסית המדפיסה Hello, World!:*



*לולאת for אשר מדפיסה את כל המספרים מ-1 עד 10:*



*הצבת ערכים במשתנים ב++C:*



# Wi-Fi Protocol

Wi-Fi היא טכנולוגיה אשר מאפשרת למכשירים אלקטרוניים לתקשר אחד עם השני בצורה אלחוטית באמצעות גלי מיקרו שהם חלק מהספקטרום הקרינה האלקטרומגנטית.

### רשת Wi-Fi פועלת באחד משני מצבים:

* רשת תשתית (Infrastructure) – מצב בו הרשת מנוהלת על ידי נקודת גישה אחת או יותר. נקודת הגישה בדרך כלל משולבת בנתב\* ואחראית על התקשורת במרחב. כל התחנות משדרות לנקודת הגישה ורק אליה והיא מנתבת את החבילות.
* רשת אד-הוק – מצב המאתר רשת בלתי מנוהלת שבה כל הצרכנים מתקשרים בינם לבין עצמם ללא תשתית\* מאורגנת. כל אחת מהתחנות עשויה לשדר באותו זמן, אך היות שכולן ממשות את הפרוטוקול, על פי רוב נמנעות הפרעות הדדיות.

### יתרונות:

* אין צורך בפריסת כבלים והגדרות פרוטוקול מורכבות במחשב.
* זול.
* מחשבים ניידים רבים נמכרים כשהם כוללים כרטיסי Wi-Fi, כך שניתן לחבר אותם בקלות לרשתות אלחוטיות.

### יישומים:

* רשת אלחוטית ביתית
* גלישה באינטרנט באמצעות נתב אלחוטי
* גלישה באינטרנט באמצעות חיבור מחשב נייד או טלפון לנקודת גישה
* תקשורת בין שני מחשבים או מכשירים שונים, לצורכי צבא לדוגמא.
* ועוד הרבה.

## מושגים

*נתב*

נתב או ראוטר (Router), הוא רכיב תקשורת מחשבים שנועד לקביעת נתיבן והפצתן של חבילות נתונים ברשתות תקשורת נתונים.

*תשתית*

תשתית הוא מונח המציין יסודות מבניים אשר מספקים את המסגרת לתמיכה במערכת כולה. קיימות משמעויות שונות למונח תשתית בתחומים שונים, אך בתחום המחשוב והתקשורת הכוונה לערוצי תקשורת רשמיים ולא רשמיים, לכלי פיתוח תוכנה ולתקשורת נתונים.

תכנות עיבוד התמונה שלנו

הפרוייקט שלנו דורש מאיתנו למצוא מספר דברים שקורים ב"שטח" בזמן המשחק. ביניהם, יש עלינו למצוא את מספר הצעדים שיש לשלוח למנוע שיזוז, לקבל קלט מן המשתמש כשהוא מסיים את תורו, וכמובן, לקלוט את תזוזות השחקן.

עיבוד התמונה אשר כתבנו, מוצאת את תזוזת השחקן בתום תורו, ושולחת את המידע למשחק השחמט שלנו דרך *socket*. אם התזוזה תקינה, התוכנית שולחת את המידע לבינה המלאכותית שלנו (אשר גם היא נכתבת ב-*Python*).

# קוד עיבוד התמונה

## חלק 1 – הגדרת משתנים ואתחול התוכנית

בחלק זה, אנו עושים *import* לספריות ההכרחיות (*cv2, numpy, skimage, sockets etc.)* ויוצרים מחלקת Chess בה אנו עושים *initiate* (אתחול) ל"מסיכת הלוח" (*board mask –* שבה נשתמש בהמשך), את לוח המשחק (מאותחל למצב ההתחלתי), ולוח זמני אותו נעדכן בהתאם למסיכת הלוח (ה-*board mask*) בזמנים המתאימים. חוץ מדברים אלו, יצרנו *socket* בשם sock אשר תשמש לתקשורת בין הקוד של משחק השחמט לבין הקוד של ה-*Python*, יצרנו *vid* אשר ישמש לקבלת ה-*stream* מן המצלמה שלנו (באמצעות *frame*)ויצרנו את *refPt,* משתנים אשר יופיעו לעומק בנספח, אך בהסבר שטוח אלו הן נקודות הייחוס שלנו כאשר אנו חותכים את ה-*frame*.



## חלק 2 – פונקציות להכנת התמונה

בפונקציות אלו אנו הכנו את התמונה למציאת המקומות הריקים על הלוח ולמציאת תזוזת השחקן. לכל פונקציה יש תיעוד באנגלית מלמעלה, אך נתרגם אותו לעברית.

round\_down\_to\_next\_multiple\_of\_8(a)

פונקציה זו מעגלת מספר למטה למספר הבא אשר מהווה מכפלה של המספר 8. הפונקציה משתמשת באלגוריתם הבא:

<https://www.geeksforgeeks.org/round-to-next-smaller-multiple-of-8/>

resizeImage(img)

מקבל תמונה, וחותך אותה לצורה הריבועית בעלת האורך והרוחב של

המכפלה הבאה הקטנה של 8.

הסבר מילולי: *img.shape[:2]* מחזיק את האורך והרוחב של התמונה. אנחנו יוצרים *array* מהאורך והרוחב, מעגלים כל אחד מהם למטה למספר הבא אשר מהווה מכפלה של 8, ומשתמשים ב-*np.min* כדי לקחת את הערך המינימלי מהמערך. לאחר מכן, אנו משתמשים בפונקציה *cv2.resize* כדי להקטין את התמונה לגודל שרצינו.

blurImg(img)

הפונקציה מקבלת תמונה ומטשטשת אותה בעזרת הפונקציה *cv2.blur*.

getUniqueColors(img)

פונקציה זו מקבלת תמונה, ועוברת על כל "*tile"* של התמונה תוך כדי ספירה של ה"צבעים הייחודיים" שבתוך כל אחד מהם.

הפונקציה משתמשת באלגוריתם הבא: [*https://stackoverflow.com/a/56606457/11089932*](https://stackoverflow.com/a/56606457/11089932)

threshAndMask(count\_unique\_colors)

בפונקציה זו אחנו מוצאים ריבועים ריקים תוך כדי שימוש של

*cutoff* מ- *Otsu's Method.* (<https://en.wikipedia.org/wiki/Otsu%27s_method>)

****

## חלק 3 – פונקציות למציאת תזוזת השחקן

בפונקציות הללו אנו השתמשנו בממצאים של הפונקציות הקודמות כדי למצוא את תזוזת השחקן. לכל פונקציה יש תיעוד באנגלית מלמעלה, אך נתרגם אותו לעברית.

convert(x, y)

בפונקציה זו נקבל שני מספרים, *x* ו-*y*, ונמיר אותם לכתב מקובל של

."Chess Position"

mark(out, mask, wh\_t)

בפונקציה זו אנו מסמנים כל ריבוע ריק בעזרת ה-*mask*.

updateBoardAndMark(out, mask, wh\_t, chess)

בפונקציה זו אנו גם מסמנים כל ריבוע ריק, וגם מעדכנים את הלוח *updatedChess* בהתאם ל-*mask*.

get\_move(chess)

בפונקציה זו אנו מוצאים את התזוזה אותה ביצע השחקן, מדפיסים אותה אל המשחק ואז מחזירים אותה כדי שתשלח לקוד של משחק השחמט.

stabilizeMask(prevMasks)

פונקציה זו היא פונקציה שמייצבת את ה-*mask* שלנו בשימוש עם המסכות הקודמות, ולאחר מכן היא מחזירה את המסכה המיוצבת. חשוב לדעת: פונקציה זו נכתבה הן בעזרת ידע והבנה והן בעזרת ניסויים בעזרתם הצלחנו להבין מהי הטעות הנפוצה ביותר.





תהליך העבודה עם המנועים וה-ESP

החלטנו תחילה ליצור קשר בין ה-*ESP* לבין קוד ה-*Python* שלנו, והחלטנו לעשות זאת באמצעות *Sockets* ו-*AP*. כלומר, הפכנו את ה-*ESP* שלנו *לAccess* *Point*, אליה התחברנו עם המחשב ובאמצעותה תקשרנו עם *sockets*.

## הקוד הבסיסי ב-C++



## הקוד הבסיסי ב-Python



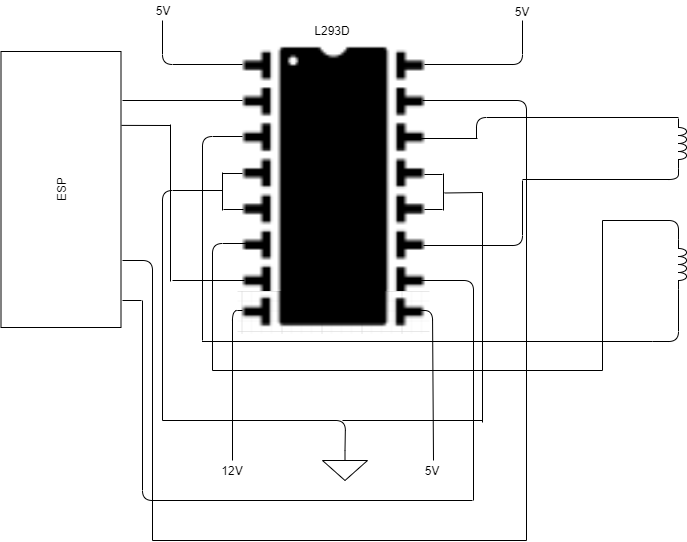
## **הסבר כללי**

תחילה, הקוד שרץ על ה-*ESP32* פותח *Access Point*, עם שם וסיסמא לבחירתנו, אליו נתחבר עם המחשב שמריץ את קוד הלקוח. לאחר מכן נריץ את הלקוח שיתחבר אל ה-*ESP32* דרך ,*socket* ודרכו הלקוח יעביר ל-*ESP32* את המידע בשביל המנועים.

## **חיבור הדרייברים והמנועים אל ה-ESP**

לצורך חיבור הדרייברים והמנועים אל ה-ESP השתמשנו בתרשים אותו הביא לנו אייל, מנחה הפרוייקט שלנו.

## התרשים



## **המשך העבודה עם ה-ESP וה-Python**

## קוד ה-Python

כעת, לאחר שהצלחנו לחבר בין ה-*ESP* לבין קוד ה-*Python* שלנו, הוספנו את המנועים, ושיפרנו את הקוד. החלטנו שרירותית איזה מנוע אחראי על כל כיוון תזוזה (למעלה, למטה, ימינה ושמאלה), והחלטנו שרירותית על פרוטוקול התקשורת בין ה-*ESP* ל-*Python*.

קוד ה-*Python* מקבל כקלט את מספר הצעדים שרוצים לזוז ואת כיוון התזוזה ושולח אותם בפרוטוקול הבא:

חמש האותיות הראשונות בהודעה מסמלות את מספר הצעדים הרצוי: *xxxxx*

האות האחרונה בהודעה מסמלת את הכיוון הרצוי: *u/d/r/l*

להלן קוד ה-*Python*:



## קוד ה-ESP

קוד ה-*ESP* יוצר *AP* אליו אנו מתחברים עם המחשב שמריץ את קוד ה-*Python* ודרכו אנו מתקשרים בעזרת *Sockets*. הקוד מחכה למידע ב-*Socket*, וכשהוא מקבל אותה הוא קורא לפונקציה אשר מטפלת במידע שנכנס כמו שצריך. לאחר מכן, אם התזוזה הצליחה ה-*ESP* יחזיר *OK* דרך ה-*Socket*.





בניית משחק השחמט

### לוגיקת קוד השחמט

תוכנת הלוגיקה של השחמט מתקשרת עם תוכנת עיבוד התמונה דרך sockets. הלוגיקה בנויה מכמה מחלקות מרכזיות:

- *Board*אחראית על שמירת המצב הנוכחי של הלוח והצגתו.

*HandleGame* - אחראית על הרצת המשחק וקבלת הפקודות מה-*frontend*.

*Piece* – מחלקה אבסטרקטית שממנה יוצאות כל מחלקות החלקים (לדוגמא *Queen*, *Pawn* ועוד). כל מחלקה של חלק אחראית על התזוזה של חלק זה, אך שאר הפונקציות זהות לשל האחרות.

לאחר שה-*frontend* שולח בקשה של תזוזה ל-*backend*, ה-*backend* יחזיר לו את אחד מהקודים הבאים:

* 0 – מהלך תקין.
* 1 – מהלך תקין, התבצעה תזוזה שגרמה שח על היריב.
* 2 – מהלך לא תקין, במשבצת המקור אין כלי של השחקן הנוכחי.
* 3 – מהלך לא תקין, במשבצת היעד קיים כלי של השחקן הנוכחי.
* 4 – מהלך לא תקין, בעקבות התזוזה יגרם שח על השחקן הנוכחי.
* 5 – מהלך לא תקין, אינדקסים של המשבצות אינם חוקיים.
* 6 – מהלך לא תקין, תזוזה לא חוקית של כלי.
* 7 – מהלך לא תקין, משבצת המקור ומשבצת היעד זהות.
* 8 – מהלך תקין, התבצע שחמט!
* 9 – הצרחה.

### פונקציות מרכזיות

**isInCheck**

הפונקציה עוברת על כל הלוח ובודקת האם יש חלק שבאפשרותו לאכול את המלך של השחקן הנגדי, אם קיים חלק כזה: הוא נמצא ב-"שח".

**isInMate**

הפונקציה בודקת האם המלך נמצא ב-"שח" והאם אין לו מהלכים חוקיים לבצע. אם שני תנאים אלו מתקיימים, המלך נמצא ב-"שח-מט" והמשחק נגמר.

**isValidMove**

פונקציה זו מקבלת יעד מסוים שברצון השחקן לזוז (או, במקרים מסויימים בו הקוד מחליט על היעד, המחשב), בודקת האם התזוזה חוקית (בהתאם לחלק הזז ולמצב הלוח) ומחזירה תשובה בהתאם.

### קוד השחמט

### **Piece**

#### **Piece.h**

1. #ifndef \_\_PIECE\_H\_\_
2. #define \_\_PIECE\_H\_\_
4. #include <iostream>
5. #include "HandleGame.h"
7. #define BLACK 1
8. #define WHITE 0
10. class Board;
12. class Piece
13. {
14. protected:
15. char \_type;
16. bool \_hasMoved;
18. /// <summary>
19. /// This function handles the checking of codes SELF\_CHECK and CHECK.
20. /// </summary>
21. /// <param name="board"> The game board </param>
22. /// <param name="src"> The position of the piece that the player wants to move </param>
23. /// <param name="dest"> Where the player wants to move the piece to </param>
24. /// <returns></returns>
25. static int HandleCheckCodes(Board& board, const int\* src, const int\* dest);
26. public:
27. /// <summary>
28. /// This function is the constructor of the class Piece.
29. /// </summary>
30. /// <param name="type"> The type of the piece </param>
31. Piece(const char type);
33. /// <summary>
34. /// This function sets a new type to \_type.
35. /// </summary>
36. /// <param name="type"> The new type </param>
37. void setType(char type);
39. /// <summary>
40. /// This function gets the type of the piece.
41. /// </summary>
42. /// <returns> The piece type </returns>
43. char getType() const;
45. /// <summary>
46. /// This function checks if a move that the player wants to play is valid.
47. /// </summary>
48. /// <param name="src"> The position of the piece that the player wants to move </param>
49. /// <param name="dest"> Where the player wants to move the piece to </param>
50. /// <param name="board"> The game board </param>
51. /// <returns> A number code between 0 - 8 that indicates a certain outcome of the checking </returns>
52. static int isValidMove(const int\* src, const int\* dest, Board& board);
54. /// <summary>
55. /// This function checks if a move that is performed on a piece matches the way that the piece moves.
56. /// </summary>
57. /// <param name="src"> The position of the piece that the player wants to move </param>
58. /// <param name="dest"> Where the player wants to move the piece to </param>
59. /// <param name="board"> The game board </param>
60. /// <returns> If the move is valid or not </returns>
61. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const = 0;
63. bool getHasMoved() const;
64. void setHasMoved(const bool hasMoved);
65. };
67. #endif // !\_\_PIECE\_H\_\_

#### 

#### **Piece.cpp**

1. #include "Piece.h"
3. Piece::Piece(const char type)
4. {
5. this->\_type = type;
6. this->\_hasMoved = false;
7. }
9. void Piece::setType(char type)
10. {
11. this->\_type = type;
12. }
14. char Piece::getType() const
15. {
16. return this->\_type;
17. }
19. int Piece::isValidMove(const int\* src, const int\* dest, Board& board)
20. {
21. Piece\* srcPiece = board.getBoard()[src[0]][src[1]];
22. Piece\* destPiece = board.getBoard()[dest[0]][dest[1]];
23. bool srcColor = (isupper(srcPiece->getType())) ? WHITE : BLACK;
24. bool destColor = (isupper(destPiece->getType())) ? WHITE : BLACK;
26. if (srcColor != HandleGame::getCurrentPlayer() || srcPiece->getType() == EMPTY\_SQUARE)
27. {
28. return INVALID\_SRC;
29. }
31. if (destPiece->getType() != EMPTY\_SQUARE && destColor == HandleGame::getCurrentPlayer())
32. {
33. return INVALID\_DEST;
34. }
36. if (!srcPiece->isValidPieceMove(src, dest, board))
37. {
38. return INVALID\_MOVE;
39. }
41. return Piece::HandleCheckCodes(board, src, dest);
42. }
44. int Piece::HandleCheckCodes(Board& board, const int\* src, const int\* dest)
45. {
46. Piece\* destPiece = board.getBoard()[dest[0]][dest[1]];
47. int code = VALID;
48. int checkValue = NO\_CHECK;
49. int currentPlayer = HandleGame::getCurrentPlayer();
51. board.movePiece(src, dest, false);
52. checkValue = board.isInCheck();
53. if (currentPlayer != checkValue &&
54. checkValue != NO\_CHECK)
55. {
56. code = CHECK;
57. }
58. if (currentPlayer == checkValue && checkValue != NO\_CHECK)
59. {
60. code = SELF\_CHECK;
61. }
62. if (checkValue == 2)
63. {
64. code = SELF\_CHECK;
65. }
67. board.movePiece(dest, src, true);
68. board.getBoard()[dest[0]][dest[1]] = destPiece;
70. return code;
71. }
73. bool Piece::getHasMoved() const
74. {
75. return this->\_hasMoved;
76. }
78. void Piece::setHasMoved(const bool hasMoved)
79. {
80. this->\_hasMoved = hasMoved;
81. }

### **HandleGame**

#### **HandleGame.h**

1. #pragma once
2. #include <iostream>
3. #include "Board.h"
4. #include <WinSock2.h>
5. #include <string>
6. #include <vector>
7. class Board;
8. class Piece;
10. #define BLACK 1
11. #define WHITE 0
12. #define INIT\_STR "rnbqkbnrpppppppp################################PPPPPPPPRNBQKBNR0"
14. enum returnCodes {VALID, CHECK, INVALID\_SRC, INVALID\_DEST, SELF\_CHECK, INVALID\_INDEX, INVALID\_MOVE, SAME\_SRC\_AND\_DEST, CHECKMATE, CASTLE};
16. class HandleGame
17. {
18. private:
19. static bool \_currentPlayer; // black - true, white - false
21. public:
22. /// <summary>
23. /// This function returns the current player.
24. /// </summary>
25. /// <returns> The current player (black or white) </returns>
26. static bool getCurrentPlayer();
28. /// <summary>
29. /// This function sets the current player.
30. /// </summary>
31. /// <param name="color"> The current player (black or white) </param>
32. static void setCurrentPlayer(int color);
34. /// <summary>
35. /// This function runs the main loop of the game.
36. /// </summary>
37. /// <param name="board"> The game board </param>
38. /// <param name="pipe"> A connection between the backend and the frontend </param>
39. static void startGame(Board\* board, SOCKET\* pipe);
41. /// <summary>
42. /// This function handles a player's turn:
43. /// Gets his move, checks if it's valid and returns the fitting code.
44. /// </summary>
45. /// <param name="instruction"> The instruction from the graphics </param>
46. /// <param name="board"></param>
47. /// <returns> The code to send to the graphics </returns>
48. static int handleTurn(std::string instruction, Board& board);
50. /// <summary>
51. /// This function changes the current player according to the number of rounds.
52. /// </summary>
53. /// <param name="code"> A code that was sent to the graphics at the end of a turn </param>
54. /// <param name="roundCounter"> A pointer to the amount of rounds that have been played </param>
55. static void changeCurrentPlayer(const int code, int\* roundCounter);
57. static std::string receiveMsg(SOCKET sock, int len, int offset);
58. static void sendMsg(SOCKET sock, const char\*\* buffer, int len);
59. };

#### **HandleGame.cpp**

1. #include "HandleGame.h"
2. #include "Rook.h"
3. #include "Pawn.h"
4. #include "Queen.h"
5. #include "King.h"
6. #include "Bishop.h"
7. #include "Knight.h"
8. #include "Piece.h"
10. const std::string castleCodes[4] = { "e1g1", "e1c1", "e8g8", "e8c8" };
11. bool HandleGame::\_currentPlayer;
13. bool HandleGame::getCurrentPlayer()
14. {
15. return \_currentPlayer;
16. }
18. void HandleGame::setCurrentPlayer(int color)
19. {
20. HandleGame::\_currentPlayer = color;
21. }
23. void HandleGame::startGame(Board\* board, SOCKET\* pipe)
24. {
25. int roundCounter = 0;
26. int code = 0;
27. char strCode[2] = { 0 };
28. std::string msgToGraphics = INIT\_STR;
29. const char\* pStr = msgToGraphics.c\_str();
30. std::string msgFromGraphics = "";
32. HandleGame::setCurrentPlayer(WHITE);
33. board->printBoard();
35. while (msgFromGraphics != "quit")
36. {
37. if (msgFromGraphics == "reset")
38. {
39. roundCounter = 0;
40. delete board;
41. board = new Board(INIT\_STR);
42. HandleGame::setCurrentPlayer(WHITE);
43. board->printBoard();
44. msgToGraphics = INIT\_STR;
45. pStr = msgToGraphics.c\_str();
46. sendMsg(\*pipe, &pStr, msgToGraphics.length());
47. msgFromGraphics = receiveMsg(\*pipe, 4, 0);
48. continue;
49. }
50. msgFromGraphics = receiveMsg(\*pipe, 4, 0);
51. code = HandleGame::handleTurn(msgFromGraphics, \*board);
53. strCode[0] = code + '0';
54. strCode[1] = '\0';
56. msgToGraphics = strCode;
57. pStr = msgToGraphics.c\_str();
58. sendMsg(\*pipe, &pStr, msgToGraphics.length());
60. HandleGame::changeCurrentPlayer(code, &roundCounter);
61. }
62. }
64. int HandleGame::handleTurn(std::string instruction, Board& board)
65. {
66. int\* src = board.convertIndex(instruction.substr(0, 2));
67. int\* dest = board.convertIndex(instruction.substr(2, 2));
68. char srcChar = board.getBoard()[src[0]][src[1]]->getType();
69. char destChar = board.getBoard()[dest[0]][dest[1]]->getType();
70. const std::string\* str = std::find(std::begin(castleCodes), std::end(castleCodes), instruction);
71. int code = 0;
73. if (tolower(srcChar) == B\_KING && str != std::end(castleCodes))
74. {
75. code = board.castle(src, dest);
76. }
77. else
78. {
79. code = board.getBoard()[src[0]][src[1]]
80. ->isValidMove(src, dest, board);
81. }
83. if (code == VALID || code == CHECK || code == CASTLE)
84. {
85. board.getBoard()[src[0]][src[1]]->setHasMoved(true);
86. board.movePiece(src, dest, true);
88. if (srcChar == B\_PAWN || srcChar == W\_PAWN)
89. {
90. if (dest[0] == 7 || dest[0] == 0)
91. {
92. delete board.getBoard()[dest[0]][dest[1]];
93. board.getBoard()[dest[0]][dest[1]] = (dest[0] == 7)
94. ? new Queen(B\_QUEEN) : new Queen(W\_QUEEN);
95. }
96. }
98. if (code == CHECK)
99. {
100. code = board.isInMate(!HandleGame::\_currentPlayer);
101. }
102. }
103. board.printBoard();
105. delete[] src;
106. delete[] dest;
107. return code;
108. }
110. void HandleGame::changeCurrentPlayer(const int code, int\* roundCounter)
111. {
112. if (code == VALID || code == CHECK || code == CASTLE)
113. {
114. (\*roundCounter)++;
115. }
116. if ((\*roundCounter) % 2 == 0 && (code == VALID || code == CHECK || code == CASTLE))
117. {
118. HandleGame::setCurrentPlayer(WHITE);
119. }
120. else if (code == VALID || code == CHECK || code == CASTLE)
121. {
122. HandleGame::setCurrentPlayer(BLACK);
123. }
124. }
126. std::string HandleGame::receiveMsg(SOCKET sock, int len, int offset)
127. {
128. if (len == 0)
129. {
130. return (char\*)"";
131. }
133. char\* data = new char[len + 1];
134. int res = recv(sock, data, len, 0);
136. if (res == INVALID\_SOCKET)
137. {
138. std::string s = "Error while recieving from socket: ";
139. s += std::to\_string(sock);
140. throw std::exception(s.c\_str());
141. }
143. data[len] = 0;
144. return data;
145. }
147. void HandleGame::sendMsg(SOCKET sock, const char\*\* buffer, int len)
148. {
149. if (send(sock, \*buffer, len, 0) == INVALID\_SOCKET)
150. {
151. throw std::exception("Error while sending message to client");
152. }
153. }

### **EmptySquare**

#### **EmptySquare.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
4. #include "Board.h"
5. class Board;
7. class EmptySquare : public Piece
8. {
9. public:
10. /// <summary>
11. /// This function is the constructor of the class EmptySquare.
12. /// </summary>
13. /// <param name="type"> An empty square char - '#' </param>
14. EmptySquare(char type);
16. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
17. };

#### **EmptySquare.cpp**

1. #include "EmptySquare.h"
3. EmptySquare::EmptySquare(char type) : Piece(type)
4. {
5. }
7. bool EmptySquare::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
8. {
9. return false;
10. }

### **Board**

#### **Board.h**

1. #ifndef \_\_BOARD\_H\_\_
2. #define \_\_BOARD\_H\_\_
4. #include <iostream>
6. #define BOARD\_SIDE 8
7. #define BOARD\_INDEX 2
8. #define FIRST\_LETTER 'a'
9. #define EMPTY\_SQUARE '#'
10. #define WHITE\_IN\_CHECK 0
11. #define BLACK\_IN\_CHECK 1
12. #define NO\_CHECK -1
14. #define B\_KING 'k'
15. #define W\_KING 'K'
16. #define B\_ROOK 'r'
17. #define W\_ROOK 'R'
18. #define B\_BISHOP 'b'
19. #define W\_BISHOP 'B'
20. #define B\_QUEEN 'q'
21. #define W\_QUEEN 'Q'
22. #define B\_PAWN 'p'
23. #define W\_PAWN 'P'
24. #define B\_KNIGHT 'n'
25. #define W\_KNIGHT 'N'
27. class Piece;
28. class King;
30. class Board
31. {
32. private:
33. Piece\*\* \_board[BOARD\_SIDE];
34. King\* \_blackKing;
35. King\* \_whiteKing;
37. /// <summary>
38. /// This function checks if a piece can move to anywhere on the board.
39. /// </summary>
40. /// <param name="piece"> The piece to check </param>
41. /// <param name="srcX"> it's row </param>
42. /// <param name="srcY"> in's col </param>
43. /// <returns> If the piece can move </returns>
44. bool canMove(Piece& piece, int srcX, int srcY);
46. public:
47. /// <summary>
48. /// This function is the constructor of the class Board.
49. /// </summary>
50. /// <param name="strBoard"> A string that contains the board content </param>
51. Board(std::string strBoard);
53. /// <summary>
54. /// This function is the destructor of the class Board.
55. /// </summary>
56. ~Board();
58. /// <summary>
59. /// This function prints the board.
60. /// </summary>
61. void printBoard() const;
63. /// <summary>
64. /// This function returns the board.
65. /// </summary>
66. /// <returns> The board </returns>
67. Piece\*\*\* getBoard() const;
69. /// <summary>
70. /// This function gets the black king.
71. /// </summary>
72. /// <returns> The black king </returns>
73. King\* getBlackKing() const;
75. /// <summary>
76. /// This function gets the white king.
77. /// </summary>
78. /// <returns> The white king </returns>
79. King\* getWhiteKing() const;
81. /// <summary>
82. /// This function moves a piece of the board.
83. /// </summary>
84. /// <param name="src"> The position of the piece that the player wants to move </param>
85. /// <param name="dest"> Where the player wants to move the piece to </param>
86. /// <param name="toDelete"> If to delete the dest piece </param>
87. void movePiece(const int\* src, const int\* dest, bool toDelete);
89. /// <summary>
90. /// This function converts a position on the graphics board to match the board in the code.
91. /// </summary>
92. /// <param name="strIndex"> A position on the graphics board </param>
93. /// <returns> A position on the code board </returns>
94. int\* convertIndex(std::string strIndex) const;
96. /// <summary>
97. /// This function checks if one of the players is in check.
98. /// </summary>
99. /// <returns> If a player is in check and which player it is </returns>
100. int isInCheck() const;
102. /// <summary>
103. /// This function checks if one of the players is in mate.
104. /// </summary>
105. /// <param name="playerInCheck"> The player that is in check </param>
106. /// <returns> If the player is in mate </returns>
107. int isInMate(bool playerInCheck);
109. /// <summary>
110. /// This function checks if a castling move is valid.
111. /// </summary>
112. /// <param name="src"> The position of the piece that the player wants to move </param>
113. /// <param name="dest"> Where the player wants to move the piece to </param>
114. /// <returns> 9 - if castle is valid, 6 - if not valid move and 2 if the dest is not free </returns>
115. int castle(const int\* src, const int\* dest);
117. /// <summary>
118. /// These function all check if the squares between the king and the rook are empty and not under attack.
119. /// </summary>
120. /// <returns> All function return if the result of the checking was true or false </returns>
121. bool checkBottomLeft();
122. bool checkBottomRight();
123. bool checkTopLeft();
124. bool checkTopRight();
126. /// <summary>
127. /// This function checks if a square on the board is being attacked.
128. /// </summary>
129. /// <param name="pos"> A position on the poard </param>
130. /// <param name="isWhite"> If white is the attacker or not </param>
131. /// <returns> If the position is under attack </returns>
132. bool isUnderAttack(const int\* pos, bool isWhite);
133. };
135. #endif // !\_\_BOARD\_H\_\_

#### **Board.cpp**

1. #include "Board.h"
2. #include "Rook.h"
3. #include "Pawn.h"
4. #include "Queen.h"
5. #include "King.h"
6. #include "Bishop.h"
7. #include "Knight.h"
8. #include "Piece.h"
9. #include "EmptySquare.h"
10. #include <string>
12. Board::Board(std::string strBoard)
13. {
14. int counter = 0;
15. char piece = 0;
16. this->\_blackKing = 0;
17. this->\_whiteKing = 0;
19. for (int i = 0; i < BOARD\_SIDE; i++)
20. {
21. this->\_board[i] = new Piece\*[BOARD\_SIDE];
22. }
24. for (int i = 0; i < BOARD\_SIDE; i++)
25. {
26. for (int j = 0; j < BOARD\_SIDE; j++)
27. {
28. piece = strBoard[counter];
29. switch (piece)
30. {
31. case B\_KING:
32. case W\_KING:
33. {
34. King\* king = new King(piece, i, j);
35. if (piece == B\_KING)
36. {
37. this->\_blackKing = king;
38. }
39. else
40. {
41. this->\_whiteKing = king;
42. }
44. this->\_board[i][j] = king;
45. break;
46. }
48. case B\_ROOK:
49. case W\_ROOK:
50. this->\_board[i][j] = new Rook(piece);
51. break;
53. case B\_BISHOP:
55. case W\_BISHOP:
56. this->\_board[i][j] = new Bishop(piece);
57. break;
59. case B\_QUEEN:
60. case W\_QUEEN:
61. this->\_board[i][j] = new Queen(piece);
62. break;
64. case B\_PAWN:
65. case W\_PAWN:
66. this->\_board[i][j] = new Pawn(piece);
67. break;
69. case B\_KNIGHT:
70. case W\_KNIGHT:
71. this->\_board[i][j] = new Knight(piece);
72. break;
74. default:
75. this->\_board[i][j] = new EmptySquare(EMPTY\_SQUARE);
76. break;
77. }
79. counter++;
80. }
81. }
82. }
84. Board::~Board()
85. {
86. for (int i = 0; i < BOARD\_SIDE; i++)
87. {
88. for (int j = 0; j < BOARD\_SIDE; j++)
89. {
90. delete this->\_board[i][j];
91. }
92. delete[] this->\_board[i];
93. }
94. }
96. void Board::printBoard() const
97. {
98. std::cout << "The board:" << std::endl;
99. for (int i = 0; i < BOARD\_SIDE; i++)
100. {
101. for (int j = 0; j < BOARD\_SIDE; j++)
102. {
103. std::cout << this->\_board[i][j]->getType() << " ";
104. }
105. std::cout << "\n";
106. }
107. }
109. Piece\*\*\* Board::getBoard() const
110. {
111. return (Piece\*\*\*)this->\_board;
112. }
114. King\* Board::getBlackKing() const
115. {
116. return this->\_blackKing;
117. }
119. King\* Board::getWhiteKing() const
120. {
121. return this->\_whiteKing;
122. }
124. void Board::movePiece(const int\* src, const int\* dest,
125. bool toDelete)
126. {
127. char srcChar = this->\_board[src[0]][src[1]]->getType();
129. if (toDelete)
130. {
131. delete this->\_board[dest[0]][dest[1]];
132. }
133. this->\_board[dest[0]][dest[1]] = this->\_board[src[0]][src[1]];
134. this->\_board[src[0]][src[1]] = new EmptySquare(EMPTY\_SQUARE);
136. if (srcChar == B\_KING)
137. {
138. this->\_blackKing->setPosition(dest);
139. }
140. else if (srcChar == W\_KING)
141. {
142. this->\_whiteKing->setPosition(dest);
143. }
144. }
146. int\* Board::convertIndex(std::string strIndex) const
147. {
148. int\* index = new int[BOARD\_INDEX];
150. index[0] = (int)strIndex[1] - '0';
151. index[0] = index[0] + (BOARD\_SIDE - 2 \* index[0]);
152. index[1] = (int)strIndex[0] - FIRST\_LETTER;
154. return index;
155. }
157. int Board::isInCheck() const
158. {
159. int i = 0, j = 0;
160. bool blackInCheck = false, whiteInCheck = false;
161. int currPos[BOARD\_INDEX] = { 0 };
162. int returnValue = 0;
163. int\* kingToCheck = nullptr;
164. bool checkWhiteKing = false;
165. Piece\* piece;
167. for (i = 0; i < BOARD\_SIDE; i++)
168. {
169. for (j = 0; j < BOARD\_SIDE; j++)
170. {
171. piece = this->\_board[i][j];
172. if (piece->getType() != EMPTY\_SQUARE)
173. {
174. currPos[0] = i;
175. currPos[1] = j;
176. if (islower(piece->getType()))
177. {
178. checkWhiteKing = true;
179. kingToCheck = this->\_whiteKing->getPosition();
180. }
181. else
182. {
183. checkWhiteKing = false;
184. kingToCheck = this->\_blackKing->getPosition();
185. }
187. if (piece->isValidPieceMove(currPos, kingToCheck, \*this))
188. {
189. if (checkWhiteKing)
190. {
191. whiteInCheck = true;
192. }
193. else
194. {
195. blackInCheck = true;
196. }
197. }
198. }
199. }
200. }
202. if (blackInCheck && whiteInCheck)
203. {
204. return 2;
205. }
206. else if (whiteInCheck)
207. {
208. return WHITE\_IN\_CHECK;
209. }
210. else if (blackInCheck)
211. {
212. return BLACK\_IN\_CHECK;
213. }
214. return -1;
215. }
217. int Board::isInMate(bool playerInCheck)
218. {
219. int i = 0, j = 0;
220. HandleGame::setCurrentPlayer(!HandleGame::getCurrentPlayer());
222. for (i = 0; i < BOARD\_SIDE; i++)
223. {
224. for (j = 0; j < BOARD\_SIDE; j++)
225. {
226. if (playerInCheck == WHITE && isupper(this->\_board[i][j]->getType()))
227. {
228. if (canMove(\*this->\_board[i][j], i, j))
229. {
230. HandleGame::setCurrentPlayer(!HandleGame::getCurrentPlayer());
231. return CHECK;
232. }
233. }
235. if (playerInCheck == BLACK && islower(this->\_board[i][j]->getType()))
236. {
237. if (canMove(\*this->\_board[i][j], i, j))
238. {
239. HandleGame::setCurrentPlayer(!HandleGame::getCurrentPlayer());
240. return CHECK;
241. }
242. }
243. }
244. }
245. return CHECKMATE;
246. }
248. int Board::castle(const int\* src, const int\* dest)
249. {
250. Piece\* rook = this->\_board[0][0];
251. int rookSrc[BOARD\_INDEX] = { 0 };
252. int rookDest[BOARD\_INDEX] = { 0 };
253. bool srcColor = (isupper(this->
254. \_board[src[0]][src[1]]->getType())) ? WHITE : BLACK;
256. if (HandleGame::getCurrentPlayer() != srcColor ||
257. this->\_board[src[0]][src[1]]->getType() == EMPTY\_SQUARE)
258. {
259. return INVALID\_SRC;
260. }
262. if (dest[0] == 7 && dest[1] == 2) // white queen side
263. {
264. if (this->\_board[7][0]->getType() == EMPTY\_SQUARE
265. || !checkBottomLeft())
266. {
267. return INVALID\_MOVE;
268. }
270. rook = this->\_board[7][0];
271. rookSrc[0] = 7;
272. rookSrc[1] = 0;
273. rookDest[0] = 7;
274. rookDest[1] = 3;
275. }
276. else if (dest[0] == 7 && dest[1] == 6) // white king side
277. {
278. if (this->\_board[7][7]->getType() == EMPTY\_SQUARE
279. || !checkBottomRight())
280. {
281. return INVALID\_MOVE;
282. }
284. rook = this->\_board[7][7];
285. rookSrc[0] = 7;
286. rookSrc[1] = 7;
287. rookDest[0] = 7;
288. rookDest[1] = 5;
289. }
290. else if (dest[0] == 0 && dest[1] == 2) // black queen side
291. {
292. if (this->\_board[0][0]->getType() == EMPTY\_SQUARE
293. ||!checkTopLeft())
294. {
295. return INVALID\_MOVE;
296. }
298. rook = this->\_board[0][0];
299. rookSrc[0] = 0;
300. rookSrc[1] = 0;
301. rookDest[0] = 0;
302. rookDest[1] = 3;
303. }
304. else if (dest[0] == 0 && dest[1] == 6) // black king side
305. {
306. if (this->\_board[0][7]->getType() == EMPTY\_SQUARE
307. || !checkTopRight())
308. {
309. return INVALID\_MOVE;
310. }
312. rook = this->\_board[0][7];
313. rookSrc[0] = 0;
314. rookSrc[1] = 7;
315. rookDest[0] = 0;
316. rookDest[1] = 5;
317. }
319. if (this->\_board[src[0]][src[1]]->getHasMoved() ||
320. rook->getHasMoved() || this->isInCheck() != -1)
321. {
322. return INVALID\_MOVE;
323. }
325. movePiece(rookSrc, rookDest, true);
326. return CASTLE;
327. }
329. bool Board::checkBottomLeft()
330. {
331. int square1[BOARD\_INDEX] = { 7, 1 };
332. int square2[BOARD\_INDEX] = { 7, 2 };
333. int square3[BOARD\_INDEX] = { 7, 3 };
335. if (this->\_board[7][1]->getType() != EMPTY\_SQUARE
336. || this->\_board[7][2]->getType() != EMPTY\_SQUARE
337. || this->\_board[7][3]->getType() != EMPTY\_SQUARE)
338. {
339. return false;
340. }
341. if (isUnderAttack(square1, false)
342. || isUnderAttack(square2, false)
343. || isUnderAttack(square3, false))
344. {
345. return false;
346. }
347. return true;
348. }
350. bool Board::checkBottomRight()
351. {
352. int square1[BOARD\_INDEX] = { 7, 5 };
353. int square2[BOARD\_INDEX] = { 7, 6 };
355. if (this->\_board[7][5]->getType() != EMPTY\_SQUARE
356. || this->\_board[7][6]->getType() != EMPTY\_SQUARE)
357. {
358. return false;
359. }
360. if (isUnderAttack(square1, false)
361. || isUnderAttack(square2, false))
362. {
363. return false;
364. }
365. return true;
366. }
368. bool Board::checkTopLeft()
369. {
370. int square1[BOARD\_INDEX] = { 0, 1 };
371. int square2[BOARD\_INDEX] = { 0, 2 };
372. int square3[BOARD\_INDEX] = { 0, 3 };

375. if (this->\_board[0][1]->getType() != EMPTY\_SQUARE
376. || this->\_board[0][2]->getType() != EMPTY\_SQUARE
377. || this->\_board[0][3]->getType() != EMPTY\_SQUARE)
378. {
379. return false;
380. }
381. if (isUnderAttack(square1, true)
382. || isUnderAttack(square2, true)
383. || isUnderAttack(square3, true))
384. {
385. return false;
386. }
387. return true;
388. }
390. bool Board::checkTopRight()
391. {
392. int square1[BOARD\_INDEX] = { 0, 5 };
393. int square2[BOARD\_INDEX] = { 0, 6 };
395. if (this->\_board[0][5]->getType() != EMPTY\_SQUARE
396. || this->\_board[0][6]->getType() != EMPTY\_SQUARE)
397. {
398. return false;
399. }
400. if (isUnderAttack(square1, true) || isUnderAttack(square2, true))
401. {
402. return false;
403. }
404. return true;
405. }
407. bool Board::isUnderAttack(const int\* pos, bool isWhite)
408. {
409. int i = 0, j = 0;
410. int src[BOARD\_INDEX] = { 0 };
412. for (i = 0; i < BOARD\_SIDE; i++)
413. {
414. for (j = 0; j < BOARD\_SIDE; j++)
415. {
416. src[0] = i;
417. src[1] = j;
419. if (isWhite && isupper(this->getBoard()[i][j]->getType()))
420. {
421. if (this->\_board[i][j]->isValidPieceMove(src, pos, \*this))
422. {
423. return true;
424. }
425. }
426. else if (!isWhite &&
427. islower(this->getBoard()[i][j]->getType()))
428. {
429. if (this->\_board[i][j]->isValidPieceMove(src, pos, \*this))
430. {
431. return true;
432. }
433. }
434. }
435. }
436. return false;
437. }
439. bool Board::canMove(Piece& piece, int srcX, int srcY)
440. {
441. int i = 0, j = 0;
442. int src[BOARD\_INDEX] = { srcX, srcY };
443. int dest[BOARD\_INDEX] = { 0 };
445. for (i = 0; i < BOARD\_SIDE; i++)
446. {
447. for (j = 0; j < BOARD\_SIDE; j++)
448. {
449. dest[0] = i;
450. dest[1] = j;
452. if (piece.isValidMove(src, dest, \*this) == VALID
453. || piece.isValidMove(src, dest, \*this) == CHECK)
454. {
455. return true;
456. }
457. }
458. }
460. return false;
461. }

### **Bishop**

#### **Bishop.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
5. class Bishop : public Piece
6. {
7. public:
8. /// <summary>
9. /// This function is the constructor of the class bishop.
10. /// </summary>
11. /// <param name="type"> If the bishop is black or white </param>
12. Bishop(char type);
14. /// <summary>
15. /// This function checks if a move that is proformed on a bishop matches the way that a bishop moves.
16. /// </summary>
17. /// <param name="src"> The position of the bishop that the player wants to move </param>
18. /// <param name="dest"> Where the player wants to move the bishop to </param>
19. /// <param name="board"> The game board </param>
20. /// <returns> If the move is valid or not </returns>
21. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
22. };

#### **Bishop.cpp**

1. #include "Bishop.h"
3. Bishop::Bishop(char type) : Piece(type)
4. {
5. }
6. bool Bishop::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
7. {
8. int subX = src[0] - dest[0];
9. int subY = src[1] - dest[1];
11. if (src[0] == dest[0] || src[1] == dest[1]
12. || abs(subX) != abs(subY)) { {
13. {
14. // If the bishop is not going diagonally
15. return false;
16. }
17. for (int i = 1; i < abs(subX); i++)
18. {
19. if (subX > 0 && subY > 0)
20. {
21. if (board.getBoard()[src[0] - i][src[1] - i]
22. ->getType() != EMPTY\_SQUARE) // If path is not clear
23. {
24. return false;
25. }
26. }
27. else if (subX > 0 && subY < 0)
28. {
29. if (board.getBoard()[src[0] - i][src[1] + i]
30. ->getType() != EMPTY\_SQUARE) // If path is not clear
31. {
32. return false;
33. }
34. }
35. else if (subX < 0 && subY > 0)
36. {
37. if (board.getBoard()[src[0] + i][src[1] - i]
38. ->getType() != EMPTY\_SQUARE) // If path is not clear
39. {
40. return false;
41. }
42. }
43. else if (subX < 0 && subY < 0)
44. {
45. if (board.getBoard()[src[0] + i][src[1] + i]
46. ->getType() != EMPTY\_SQUARE) // If path is not clear
47. {
48. return false;
49. }
50. }
51. }
52. return true;
53. }

### **King**

#### **King.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
5. class Board;
7. class King : public Piece
8. {
9. private:
10. int \_position[BOARD\_INDEX];
12. public:
13. /// <summary>
14. /// This function is the constructor of the class king.
15. /// </summary>
16. /// <param name="type"> If the king is black or white </param>
17. King(char type, int pos1, int pos2);
19. /// <summary>
20. /// This function returns the position of the king.
21. /// </summary>
22. /// <returns> The position of the king. </returns>
23. int\* getPosition() const;
25. /// <summary>
26. /// This function sets the position of the king.
27. /// </summary>
28. /// <param name="position"> The new position for the king </param>
29. void setPosition(const int\* position);
31. /// <summary>
32. /// This function checks if a move that is proformed on a king matches the way that a king moves.
33. /// </summary>
34. /// <param name="src"> The position of the king that the player wants to move </param>
35. /// <param name="dest"> Where the player wants to move the king to </param>
36. /// <param name="board"> The game board </param>
37. /// <returns> If the move is valid or not </returns>
38. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
39. };

#### **King.cpp**

1. #include "King.h"
2. #include "HandleGame.h"
3. class Piece;
5. King::King(char type, int pos1, int pos2) : Piece(type)
6. {
7. this->\_position[0] = pos1;
8. this->\_position[1] = pos2;
9. }
11. int\* King::getPosition() const
12. {
13. return (int\*)this->\_position;
14. }
16. void King::setPosition(const int\* position)
17. {
18. this->\_position[0] = position[0];
19. this->\_position[1] = position[1];
20. }
22. bool King::isValidPieceMove(const int\* src, const int\* dest,
23. const Board& board) const
24. {
25. int subX = src[0] - dest[0];
26. int subY = src[1] - dest[1];
28. if (abs(subX) > 1 || abs(subY) > 1)
29. {
30. return false;
31. }
33. return true;
34. }

### **Knight**

#### **Knight.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
4. #include "Board.h"
5. class Board;
7. class Knight : public Piece
8. {
9. public:
10. /// <summary>
11. /// This function is the constructor of the class knight.
12. /// </summary>
13. /// <param name="type"> If the knight is black or white </param>
14. Knight(char type);
16. /// <summary>
17. /// This function checks if a move that is proformed on a knight matches the way that a knight moves.
18. /// </summary>
19. /// <param name="src"> The position of the knight that the player wants to move </param>
20. /// <param name="dest"> Where the player wants to move the knight to </param>
21. /// <param name="board"> The game board </param>
22. /// <returns> If the move is valid or not </returns>
23. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
24. };

#### **Knight.cpp**

1. #include "Knight.h"
3. Knight::Knight(char type) : Piece(type)
4. {
5. }
7. bool Knight::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
8. {
9. int subX = abs(src[0] - dest[0]);
10. int subY = abs(src[1] - dest[1]);
12. if ((subX == 2 && subY == 1) || (subX == 1 && subY == 2))
13. {
14. return true;
15. }
17. return false;
18. }

### **Pawn**

#### **Pawn.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
4. #include "Board.h"
5. class Board;
7. #define BLACK\_PAWN\_ROW 1
8. #define WHITE\_PAWN\_ROW 6
10. class Pawn : public Piece
11. {
12. public:
13. /// <summary>
14. /// This function is the constructor of the class Pawn.
15. /// </summary>
16. /// <param name="type"> If the pawn is black or white </param>
17. Pawn(char type);
19. /// <summary>
20. /// This function checks if a move that is proformed on a pawn matches the way that a pawn moves.
21. /// </summary>
22. /// <param name="src"> The position of the pawn that the player wants to move </param>
23. /// <param name="dest"> Where the player wants to move the pawn to </param>
24. /// <param name="board"> The game board </param>
25. /// <returns> If the move is valid or not </returns>
26. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
28. private:
29. /// <summary>
30. /// This function gets information about the pawn:
31. /// It gets the char after the pawn, gets the initial row of the pawn
32. /// and checks if the pawn is going backwards.
33. /// </summary>
34. /// <param name="board"> The game board </param>
35. /// <param name="initialRow"> A pointer to the initial row of the pawns </param>
36. /// <param name="afterSrc"> A pointer to the char after the src char </param>
37. /// <param name="src"> The position of the pawn to be moved </param>
38. /// <param name="dest"> Where to move the pawn to </param>
39. /// <returns> If the pawn tried to move backwards </returns>
40. bool getPawnInfo(const Board& board, int\* initialRow, Piece\*\* afterSrc, const int\* src, const int\* dest) const;
41. };

#### **Pawn.cpp**

1. #include "Pawn.h"
3. Pawn::Pawn(char type) : Piece(type)
4. {
5. }
7. bool Pawn::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
8. {
9. int subX = src[0] - dest[0];
10. int subY = src[1] - dest[1];
11. int pawnRow = src[0];
12. int initialRow = 0;
13. Piece\* destChar = board.getBoard()[dest[0]][dest[1]];
14. Piece\* afterSrc = 0;
16. if (!this->getPawnInfo(board, &initialRow, &afterSrc, src, dest))
17. {
18. // If the pawn is going backwards
19. return false;
20. }
22. if (destChar->getType() != EMPTY\_SQUARE) // If the pawn is eating
23. {
24. if (subY == 0) // If the pawn is going forward
25. {
26. return false;
27. }
28. else if (!(abs(subX) == 1 && abs(subY) == 1))
29. {
30. // If the pawn went more than one step
31. return false;
32. }
33. }
34. else
35. {
36. if (subY != 0) // If the pawn is not going forward
37. {
38. return false;
39. }
40. else if (abs(subX) > 1 && pawnRow != initialRow)
41. {
42. // If the pawn went more than one step
43. // and moved already
44. return false;
45. }
46. else if (abs(subX) > 2)
47. {
48. // If the pawn went more than two steps
49. return false;
50. }
52. else if (afterSrc)
53. {
54. if (afterSrc->getType() != EMPTY\_SQUARE)
55. {
56. // If the pawn jumped over a piece
57. return false;
58. }
59. }
60. }
62. return true;
63. }
65. bool Pawn::getPawnInfo(const Board& board, int\* initialRow,
66. Piece\*\* afterSrc, const int\* src,
67. const int\* dest) const
68. {
69. int subX = src[0] - dest[0];
71. if (isupper(this->\_type))
72. {
73. \*initialRow = WHITE\_PAWN\_ROW;
74. if (src[0] - 1 > 0 && src[0] - 1 < BOARD\_SIDE)
75. {
76. \*afterSrc = board.getBoard()[src[0] - 1][src[1]];
77. }
79. if (subX < 0)
80. {
81. return false;
82. }
83. return true;
84. }
85. else
86. {
87. \*initialRow = BLACK\_PAWN\_ROW;
88. if (src[0] + 1 > 0 && src[0] + 1 < BOARD\_SIDE)
89. {
90. \*afterSrc = board.getBoard()[src[0] + 1][src[1]];
91. }
93. if (subX > 0)
94. {
95. return false;
96. }
97. return true;
98. }
99. }

### **Queen**

#### **Queen.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
4. #include "Board.h"
5. class Board;
7. class Queen : public Piece
8. {
9. public:
10. /// <summary>
11. /// This function is the constructor of the class queen.
12. /// </summary>
13. /// <param name="type"> If the queen is black or white </param>
14. Queen(char type);
16. /// <summary>
17. /// This function checks if a move that is proformed on a queen matches the way that a queen moves.
18. /// </summary>
19. /// <param name="src"> The position of the queen that the player wants to move </param>
20. /// <param name="dest"> Where the player wants to move the queen to </param>
21. /// <param name="board"> The game board </param>
22. /// <returns> If the move is valid or not </returns>
23. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
24. };

#### **Queen.cpp**

1. #include "Queen.h"
2. #include "Bishop.h"
3. #include "Rook.h"
5. Queen::Queen(char type) : Piece(type)
6. {
7. }
9. bool Queen::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
10. {
11. Rook rook('r');
12. Bishop bishop('b');
13. return rook.isValidPieceMove(src, dest, board) || bishop.isValidPieceMove(src, dest, board);
14. }

### **Rook**

#### **Rook.h**

1. #pragma once
2. #include <iostream>
3. #include "Piece.h"
4. #include "Board.h"
5. class Piece;
6. class Board;
8. class Rook : public Piece
9. {
10. public:
11. /// <summary>
12. /// This function is the constructor of the class rook.
13. /// </summary>
14. /// <param name="type"> If the rook is black or white </param>
15. Rook(char type);
17. /// <summary>
18. /// This function checks if a move that is proformed on a rook matches the way that a rook moves.
19. /// </summary>
20. /// <param name="src"> The position of the rook that the player wants to move </param>
21. /// <param name="dest"> Where the player wants to move the rook to </param>
22. /// <param name="board"> The game board </param>
23. /// <returns> If the move is valid or not </returns>
24. virtual bool isValidPieceMove(const int\* src, const int\* dest, const Board& board) const;
25. };

#### **Rook.cpp**

1. #include "Rook.h"
2. #include "HandleGame.h"
3. class HandleGame;
5. Rook::Rook(char type) : Piece(type)
6. {
7. }
9. bool Rook::isValidPieceMove(const int\* src, const int\* dest, const Board& board) const
10. {
11. int tempSrc[BOARD\_INDEX] = { src[0], src[1] };
12. int tempDest[BOARD\_INDEX] = { dest[0], dest[1] };
14. if (tempSrc[0] != tempDest[0] && tempSrc[1] != tempDest[1]) // If the rook is not going straight
15. {
16. return false;
17. }
19. if (tempSrc[0] == tempDest[0]) // If the rook is going right or left
20. {
21. if (tempDest[1] < tempSrc[1])
22. {
23. std::swap(tempSrc[1], tempDest[1]);
24. }
26. for (int i = tempSrc[1] + 1; i < tempDest[1]; i++)
27. {
28. if (board.getBoard()[tempSrc[0]][i]->getType() != EMPTY\_SQUARE) // If the rook is trying to pass a piece
29. {
30. return false;
31. }
32. }
33. }
34. else // If the rook is going up or down
35. {
36. if (tempDest[0] < tempSrc[0])
37. {
38. std::swap(tempSrc[0], tempDest[0]);
39. }
41. for (int i = tempSrc[0] + 1; i < tempDest[0]; i++)
42. {
43. if (board.getBoard()[i][tempSrc[1]]->getType() != EMPTY\_SQUARE) // If the rook is trying to pass a piece
44. {
45. return false;
46. }
47. }
48. }
50. return true;
51. }

### **Main**

#### **Main.cpp**

1. #include <iostream>
2. #include <thread>
3. #include "Board.h"
4. #include "WSAInitializer.h"
5. #include <WinSock2.h>
6. #include "HandleGame.h"
7. #include <string>
9. #define PORT\_ADDRESS 3000
11. void bindAndListen(SOCKET& m\_socket)
12. {
13. struct sockaddr\_in sa { 0 };
14. sa.sin\_port = htons(PORT\_ADDRESS);
15. sa.sin\_family = AF\_INET;
16. sa.sin\_addr.s\_addr = INADDR\_ANY;
18. if (bind(m\_socket, (struct sockaddr\*)&sa, sizeof(sa)) == SOCKET\_ERROR)
19. {
20. throw std::exception(\_\_FUNCTION\_\_ " - bind");
21. }
23. if (listen(m\_socket, SOMAXCONN) == SOCKET\_ERROR)
24. {
25. throw std::exception(\_\_FUNCTION\_\_ " - listen");
26. }
27. std::cout << "Listening on port " << PORT\_ADDRESS << std::endl;
28. }
29. int main()
30. {
31. WSAInitializer wsaInit;
32. srand(time\_t(NULL));
33. SOCKET m\_socket;
34. Board\* board = new Board(INIT\_STR);
35. m\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);
37. if (m\_socket == INVALID\_SOCKET)
38. throw std::exception(\_\_FUNCTION\_\_ " - socket");
39. try
40. {
41. bindAndListen(m\_socket);
42. }
43. catch (const std::exception& e)
44. {
45. std::cerr << "Something bad happened here... " << e.what() << std::endl;
46. }
47. SOCKET client\_socket = accept(m\_socket, NULL, NULL);
49. if (client\_socket == INVALID\_SOCKET)
50. throw std::exception(\_\_FUNCTION\_\_);
51. HandleGame::startGame(board, &client\_socket);
52. closesocket(m\_socket);
53. return 0;
54. }

### **WSAInitializer**

#### **WSAInitializer.h**

1. #pragma once
3. #include <WinSock2.h>
4. #include <Windows.h>
5. #pragma comment(lib,"WS2\_32")

8. class WSAInitializer
9. {
10. public:
11. WSAInitializer();
12. ~WSAInitializer();
13. };

#### **WSAInitializer.cpp**

1. #include "WSAInitializer.h"
2. #include <exception>

5. WSAInitializer::WSAInitializer()
6. {
7. WSADATA wsa\_data = { };
8. if (WSAStartup(MAKEWORD(2, 2), &wsa\_data) != 0)
9. throw std::exception("WSAStartup Failed");
10. }
12. WSAInitializer::~WSAInitializer()
13. {
14. // Q: why is this try necessary ?
15. // A: to avoid throwing exceptions in d-tors !
16. // if we do throw think what will happened in regular exception, our object
17. // will be destroyed because an exception occurred and then the d-tor will be
18. // called and if we throw now there is no one to handle the exception because
19. // we are already in the flow of exception handling !! (inception...)
20. // please read more about exception handling and why it's forbidden to throw
21. // exception from the destructor.
22. try
23. {
24. WSACleanup();
25. }
26. catch (...) {}
27. }