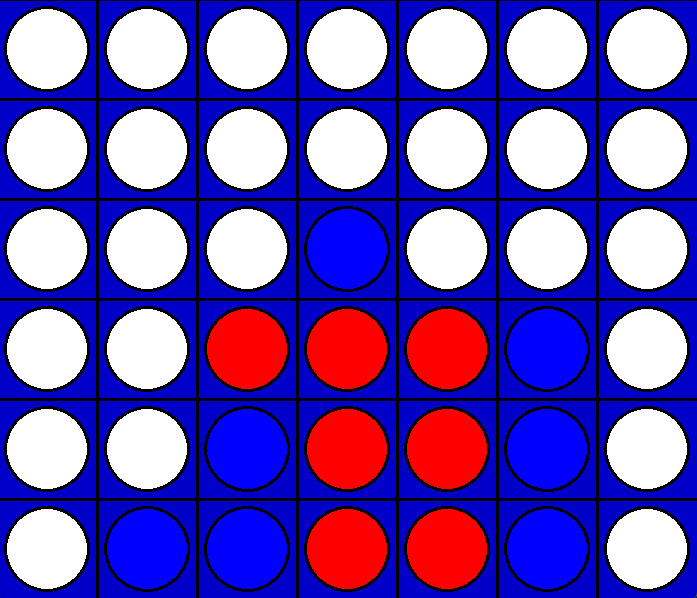
פרויקט בהנדסת תוכנה

בינה מלאכותית

סוכן AI למשחק ארבע בשורה



שם התלמיד: אופק אביבי

ת.ז.: 215712746

שם המנחה: גלעד מרקמן

שם החלופה: בינה מלאכותית ולמידת מכונה

תאריך הגשה: 5/5/24

# תוכן העניינים

[תוכן העניינים 2](#_Toc168314905)

[מבוא 4](#_Toc168314906)

[תיאור המשחק 5](#_Toc168314907)

[כללי 5](#_Toc168314908)

[כללי המשחק 5](#_Toc168314909)

[נתונים נוספים 5](#_Toc168314910)

[מדריך למשתמש 6](#_Toc168314911)

[מודל סביבה – סוכן 12](#_Toc168314912)

[תיאור המודל באופן כללי 12](#_Toc168314913)

[מימוש המודל בפרויקט 12](#_Toc168314914)

[המחלקה State 12](#_Toc168314915)

[המחלקה Graphics 13](#_Toc168314916)

[המחלקה Connect4 – Environment 14](#_Toc168314917)

[המחלקה Human\_Agent 15](#_Toc168314918)

[המחלקה Random\_Agent 15](#_Toc168314919)

[הקובץ Game 15](#_Toc168314920)

[סוכן AI באמצעות אלגוריתם חיפוש 16](#_Toc168314921)

[סוכן minMax 16](#_Toc168314922)

[האלגוריתם minMax 16](#_Toc168314923)

[מימוש האלגוריתם בפרויקט – מחלקת minMax\_Agent 18](#_Toc168314924)

[מסקנות למימוש האלגוריתם 18](#_Toc168314925)

[סוכן alpha-beta 20](#_Toc168314926)

[האלגוריתם alpha\_beta 20](#_Toc168314927)

[מימוש האלגוריתם בפרויקט – מחלקת AlphaBeta\_Agent 22](#_Toc168314928)

[מסקנות למימוש האלגוריתם 22](#_Toc168314929)

[סוכן Reinforcement - DQN 23](#_Toc168314930)

[למידת חיזוק 23](#_Toc168314931)

[מודל MDP – כללי 23](#_Toc168314932)

[מהי למידת חיזוק ומשוואות בלמן 23](#_Toc168314933)

[האלגוריתם Q-Learning 24](#_Toc168314934)

[למידת חיזוק עמוקה DQN 25](#_Toc168314935)

[האלגוריתם DQN 25](#_Toc168314936)

[Replay buffer 26](#_Toc168314937)

[נוסחת בלמן בלמידת חיזוק עמוקה 26](#_Toc168314938)

[מימוש DQN 27](#_Toc168314939)

[כללי 27](#_Toc168314940)

[המחלקה DQN – רשת הנוירונים שנבחרה. 27](#_Toc168314941)

[המחלקה DQN\_Agent 28](#_Toc168314942)

[המחלקה Replay\_buffer 28](#_Toc168314943)

[המחלקה Trainer\_Red 29](#_Toc168314944)

[המחלקה Trainer\_Blue 30](#_Toc168314945)

[המחלקה Trainer\_RB 31](#_Toc168314946)

[תוצאות ומסקנות המחקר 32](#_Toc168314947)

[רפלקציה 33](#_Toc168314948)

[מדוע בחרת בפרויקט זה 33](#_Toc168314949)

[תיאור תהליך התכנון 33](#_Toc168314950)

[התמודדות עם אתגרים שבדרך 33](#_Toc168314951)

[דברים שלמדתי על עצמי בעקבות הפרויקט. 33](#_Toc168314952)

[תודות 34](#_Toc168314953)

[ביבליוגרפיה 34](#_Toc168314954)

[נספחים 35](#_Toc168314955)

[המחלקה Graphics 35](#_Toc168314956)

[המחלקה State 37](#_Toc168314957)

[המחלקה Connect4 – הסביבה 38](#_Toc168314958)

[סוכן AI – minMax 39](#_Toc168314959)

[סוכן AI– alpha\_beta 41](#_Toc168314960)

# מבוא

**תמצית מנהלים של הפרויקט:** פיתוח פרויקט משחק ארבע בשורה בפייתון ומימוש הממשק באמצעות PyGame, פיתוח סוכני חיפוש ואימון סוכני בינה מלאכותית בשיטת DQN באמצעות PyTorch . הצלחה באימון סוכן DQN וסוכני חיפוש עד כדי ניצחון של שניהם מול סוכן רנדומלי וניצחון של DQN נגד סוכני החיפוש.

**מסקנות עיקריות:** המשחק יחסית קל ללמד ולנצח. באופן כללי סוכן האלפא-בטא מצליח להגיע לרמה גבוהה מול סוכן רנדומלי, ואפשר דרך סוכני החיפוש להגיע להצלחה גבוהה במשחק. סוכן הDQN גם מגיע להישגים גבוהים ובמקרה הכי טוב גם הצליח לנצח את סוכן האלפא-בטא. בכללי, ניתן לסוכן להשתפר במשחק די בקלות משום שהוא משחק יחסית פשוט עם פעולה חד-ממדית ולכן גם רשת הנוירונים הכי טובה שנמצאה היא יחסית רדודה.

**קישור לGithub:** <https://github.com/Vexrz2/Connect-4-AI>

# תיאור המשחק

## כללי

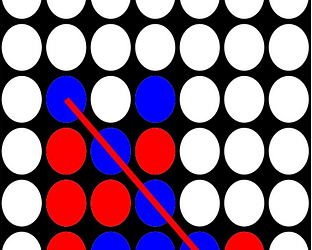
ארבע בשורה הוא משחק שבו השחקנים בוחרים צבע ואז בתורות מפילים אסימונים צבעוניים לרשת תלויה אנכית בת שש שורות ושבעה טורים. החלקים נופלים ישר למטה, תופסים את התא הזמין הנמוך ביותר בתוך הטור.

המשחק נוצר על ידי הווארד וקסלר, ונמכר לראשונה תחת הסימן המסחרי Connect Fourעל ידי מילטון בראדלי בפברואר 1974.

קישורים מעניינים: <https://connect4.gamesolver.org/>

## כללי המשחק

מטרת המשחק היא להיות הראשון ליצור קו אופקי, אנכי או אלכסוני של ארבעה אסימונים משלו. דוגמא לניצחון:



## נתונים נוספים

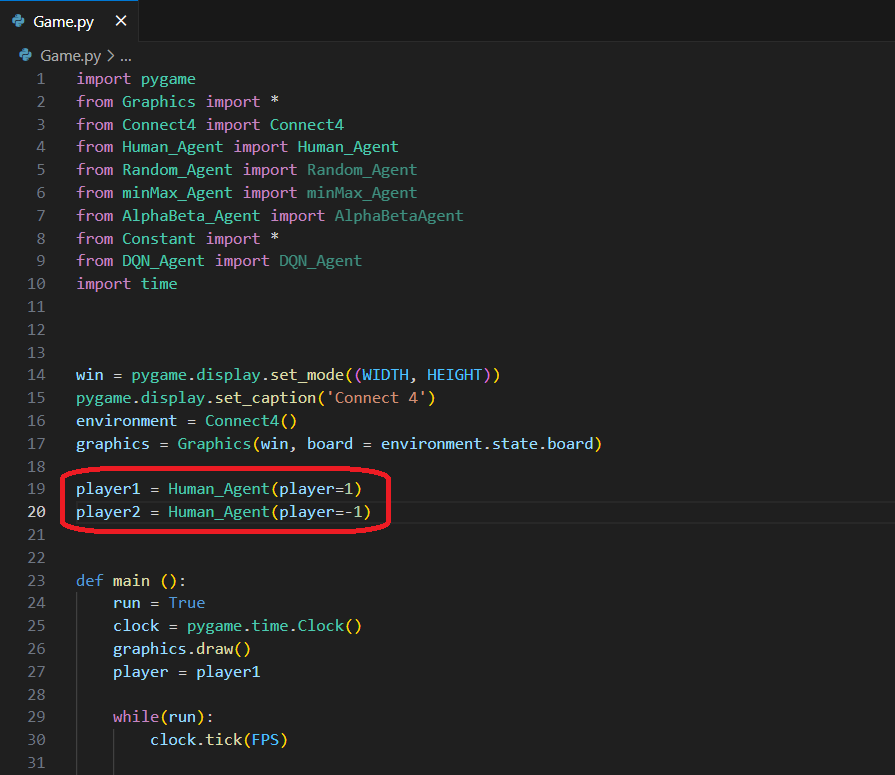
ארבע בשורה הוא משחק פתור. השחקן הראשון תמיד יכול לנצח על ידי הפעלת המהלכים הנכונים.

* אם השחקן הפותח ישים אסימון בעמודה האמצעית – הוא יכול להבטיח ניצחון על ידי משחק מושלם.
* אם יפתח על ידי הנחת אסימונים בשתי העמודות הצמודות לאמצעית – משחק מושלם של השחקן השני יאפשר לו להשיג תוצאת תיקו (אפילו אם השחקן הפותח ימשיך במשחק מושלם משלו).
* על ידי פתיחה בהנחת אסימונים באחת מארבע העמודות החיצוניות – משחק מושלם של השחקן השני יאפשר לו לנצח במשחק.
* למשחק יש 4,531,985,219,092 (4.5 טריליון) מצבים אפשריים (מקור: (<https://oeis.org/A212693>

# מדריך למשתמש

**ספריות פייתון:** במידת הצורך, יש להוריד את הספריות הבאות של פייתון (באמצעות pip):

Pygame, torch, time

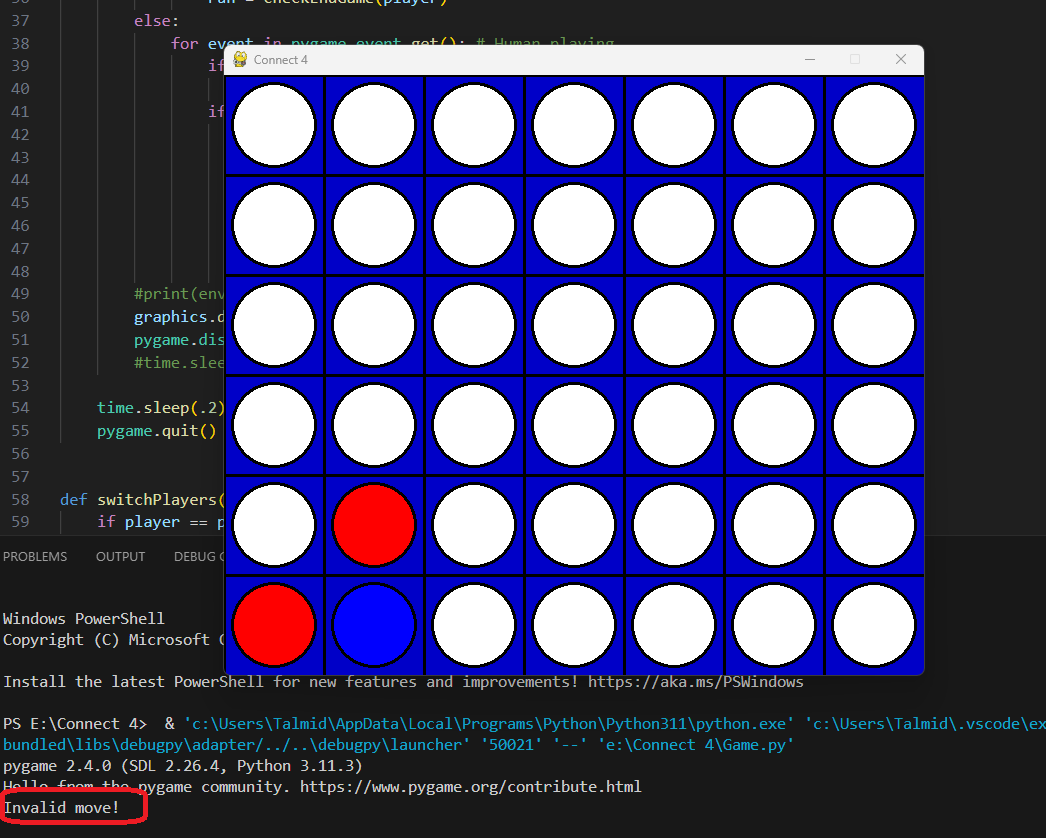
**התחלת משחק:** הפעלת משחק רגיל בתוך קובץ 'Game'. להלן צילום המסך המציג כיצד לבחור את הסוכנים לשני השחקנים:

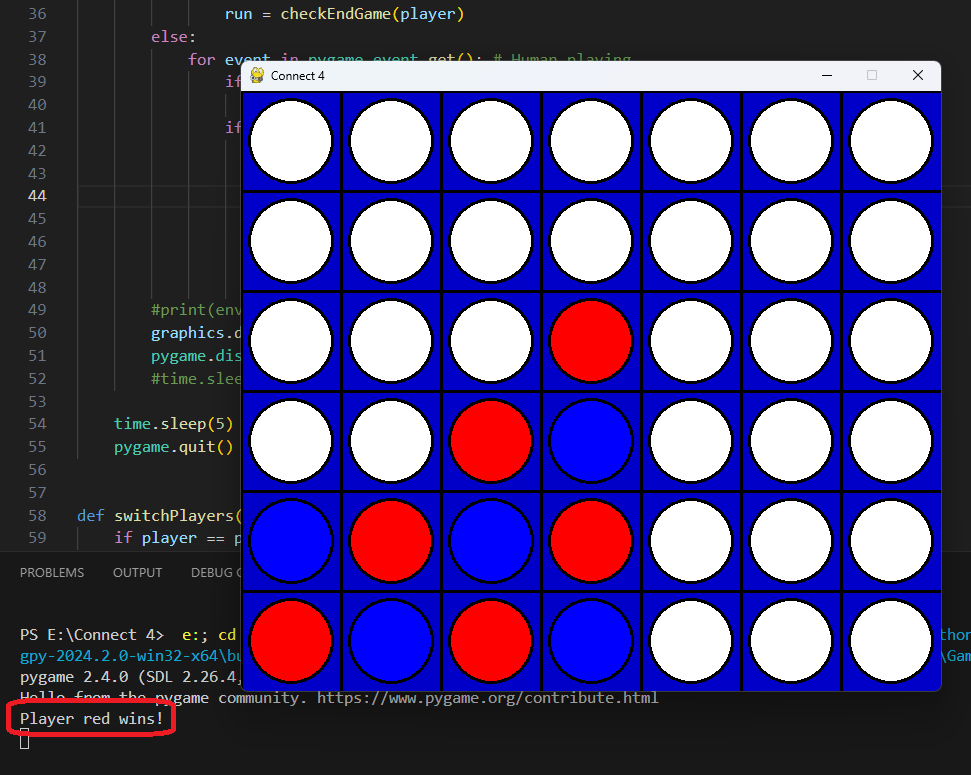
כמובן שיש לדאוג לפרמטרים שהסוכנים מקבלים וכי השחקנים תואמים. (1 עבור שחקן 1, -1 עבור שחקן 2)

**סוכנים אפשריים לבחירה:**

Human\_Agent, Random\_Agent, minMax\_Agent, AlphaBeta\_Agent, DQN\_Agent

כעת יש להריץ את הקובץ באמצעות Ctrl + F5 ויתחיל המשחק.

**מהלך המשחק:** כדי לשחק יש להקיש על אחד המקשים 1-7 כאשר כל מספר תואם למספר הטור בלוח כאשר 1 הטור הכי שמאלי. יש לציין כי הקשת מקש אחר תדפיס שגיאה והמשחק ימשיך כרגיל ויחכה לפעולת השחקן.

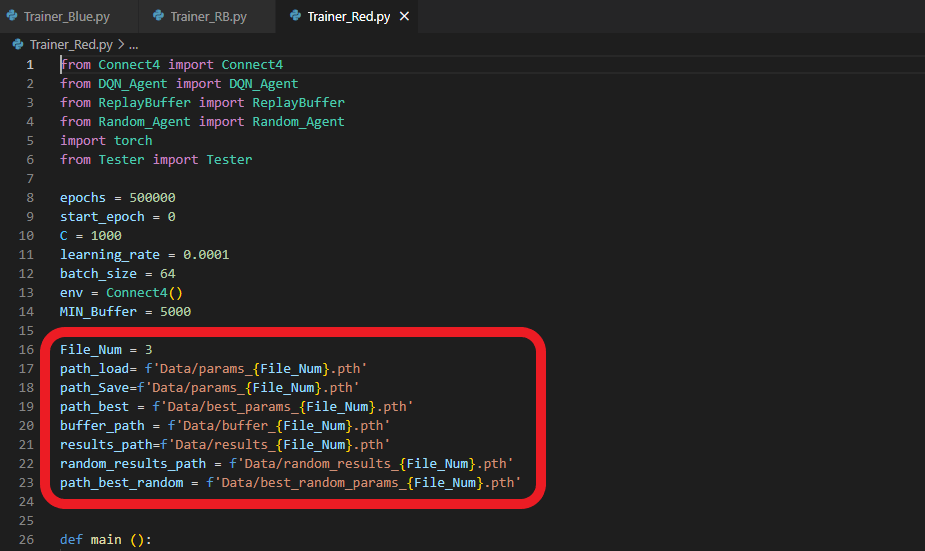
**סיום המשחק:** כאשר אחד השחקנים מחבר ארבעה מכליו בשורה – המשחק נגמר והשחקן מנצח, וחלון המשחק נסגר אוטומטית. בנוסף מודפסת הודעה על המסך של השחקן שניצח. 

**אימון שחקן:** קודם כל, הגדרת הפרמטרים נמצאת בקבצים הבאים:

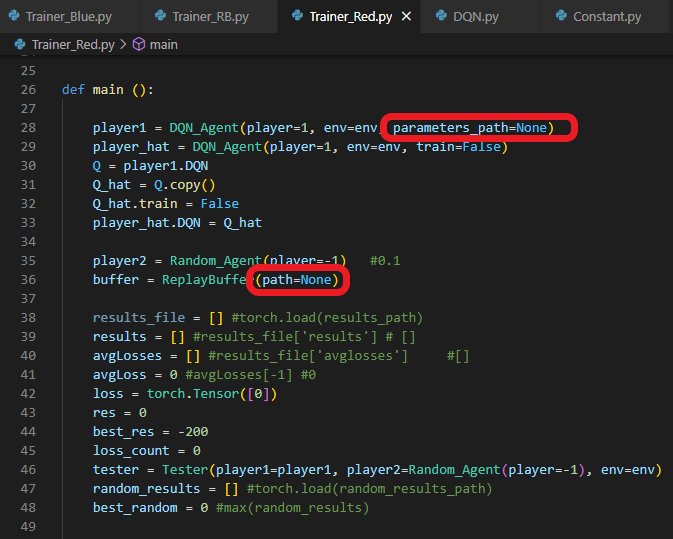
“Constant”, “ReplayBuffer”, “Trainer\_\*”, “DQN”

כאשר Trainer כולל את רובם, DQN כולל את מבנה רשת הנוירונים המלאכותית (ANN), ReplayBuffer כולל את גודל הbuffer, וConstant כולל את decay. בנוסף, ניתן להגדיר scheduler בתוך קובץ האימון ולהגדיר כל כמה זמן מתבצע עדכון קצב הלמידה (LR).

הקבצים להריץ כדי לאמן סוכן הם אחד מהקבצים מהצורה “Trainer\_\*” כאשר ההבדל ביניהם הוא סוג השחקן שמאמנים. נציג את האימון בקובץ “Trainer\_Red”. לאחר הגדרת הפרמטרים, יש ליצור\לבחור קבצים לשמור את הפרמטרים של רשת הנוירונים והמודל הנוכחי.

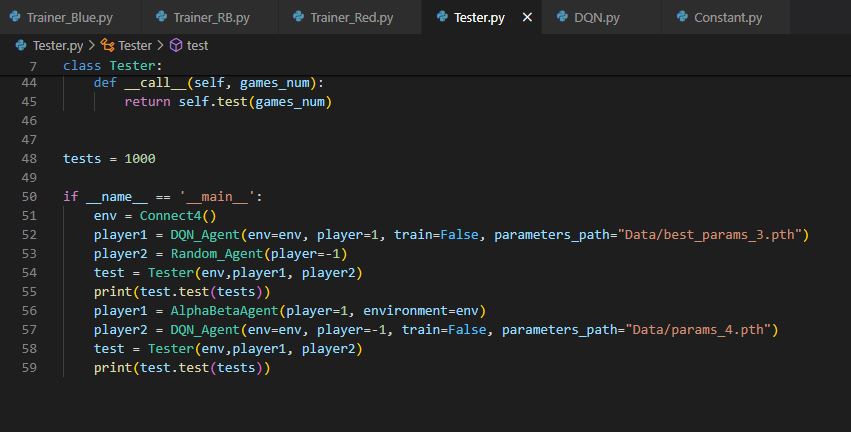


אנו נשתמש ב2 הקבצים הראשונים לטעינה ושמירת הפרמטרים. בקובץ ה3 נשמרים הפרמטרים הכי טובים שהתקבלו עד כה באימון (עם אחוז הניצחון הכי גבוה). בקובץ ה4 נשמרים נתוני הbuffer, בקבצים 5 ו6 נשמרים נתונים להצגה בגרף (בהמשך). בקובץ האחרון נשמרים הפרמטרים הכי טובים נגד שחקן רנדומלי.

**טעינת פרמטרים:** 

באזורים המוקפים באדום, אם רוצים לטעון מודלים קיימים יש להגדיר את הpath לפי הקבצים שהוגדרו בתחילת הקובץ (כפי שפורט דלעיל). בנוסף, כדי לטעון גם את הפרמטרים של הגרף, יש להחליף את כל המערכים הריקים במה שמסומן אחר כך כהערה (כדי לטעון את הנתונים הקיימים).

**בדיקת סוכנים וטעינת מודלים למשחקים:**



כעת נציג את המחלקה Tester, כאשר מטרתה לבחון כמות מסוימת של משחקים של שני סוכנים ולהחזיר את כמות הניצחונות לכל סוכן. אם מריצים את הקובץ לבד, יש להגדיר את מס' המשחקים בפרמטר tests, ואם משתמשים בקובץ חיצונית (למשל כדי לבדוק איכות אימון סוכן) יש לקרוא לאובייקט Tester עם מס' המשחקים כפרמטר. בנוסף, מתוארת כאן טעינת הפרמטרים אל תוך סוכן הDQN, שכן ניתן להשתמש בו גם במחלקת Game כדי לשחק מולו, ויש להגדיר אותו כפי שמתואר בתמונה, כאשר הקובץ לטעינה תלוי באיזה פרמטרים רוצים להשתמש.

# מודל סביבה – סוכן

## תיאור המודל באופן כללי

נציג מודל לבניית מערכת לפתרון בעיות באמצעות בינה מלאכותית. המודל מחלק את המשחק לחלקים לוגיים כאשר כל אחד מבצע תפקיד שונה (SOLID). המודל מאפשר לנו לתאר את העולם שסביבנו, ואת היחסים בין מרכיבי העולם לסביבה. המודל מתאים גם לבניית משחקים בהם המחשב מהווה את אחד המשתתפים במשחק. מודל זה מהווה בסיס למודל מתמטי לפתרון בעיות הנקרא MDP – Markov Decision Process שבאמצעותו מתבצעת למידת חיזוק (קרא בהמשך).

**רכיבי המודל:**

* סביבה – (Environment) מתאר את העולם בו אנו נמצאים - חוקי העולם / חוקי המשחק
* סוכן – (Agent) הרובוט אותו אנחנו מתכנתים. תוכנת המחשב הפועלת בעולם זה. הסוכן קולט את הסביבה ומבצע פעולות המשנות את הסביבה.
* מצב – (State) תיאור של מצב רגעי של העולם. המצב מתאר "הקפאה" של הסביבה.
* פעולה – (Action) פעולות אותן מבצע הסוכן המשנות את הסביבה, צעדי המשחק.

**מטרת המודל:** לחלק את המשחק לחלקים לוגיים על מנת לממש סוכנים, ביניהם גם סוכנים חכמים. בנוסף זוהי דרך יעילה וקלה לממש ממשק עבור משחק כמו ארבע בשורה.

## מימוש המודל בפרויקט

### המחלקה State

ייצוג מצב הלוח הקיים באמצעות מערך דו-ממדי. אסימון של שחקן 1 מיוצג ע"י הספרה 1, שחקן 2 מיוצג ע"י הספרה -1, וכאשר התא ריק הוא מיוצג ע"י הספרה 0.

**מאפיינים:**

* לוח – board – מערך דו-ממדי של מצב הלוח נוכחי
* שחקן – player – תור השחקן
* פעולה – action – פעולת השחקן, 1-7 מייצגים את מס' הטור של האסימון (משמאל לימין)
* פעולה אחרונה – last\_action – שמירת פעולה של תור קודם (בשביל ייעול בדיקת ניצחון)

**פעולות עיקריות:**

* toTensor (self, device = torch.device('cpu')) – ממירה את הלוח הנוכחי לtensor להכנסה אל תוך רשת הנוירונים, מקבלת רק את סוג המכשיר (לשימוש אחר כך ברשת).
* tensorToState (state\_tensor, player) – פעולה הפוכה מtoTensor, ממירה את הtensor שהיא מקבלת ללוח ואת תור השחקן הנוכחי לאובייקט state חדש שהיא מחזירה. זוהי פעולה סטטית לכן לא מקבלת self.
* reverse (self) – הופכת את כל הלוח (כפל ב-1) וגם את תור השחקן.

**נספח נמצא בסוף תיק הפרויקט תחת "מחלקת State"**

### המחלקה Graphics

ייצוג הממשק הגרפי של המשחק

**מאפיינים:**

* לוח – board – מערך דו-ממדי של מצב הלוח נוכחי
* חלון – win – חלון המשחק
* שורות – rows – כמות השורות בלוח
* טורים – cols – כמות הטורים בלוח

**פעולות עיקריות:**

* draw\_Lines\_Circles(self) – מציירת את קווי הלוח ואת התאים על חלון המשחק
* draw\_all\_pieces(self) – מציירת את כל האסימונים בלוח
* draw\_piece(self, row\_col, player) – מציירת אסימון יחיד – לפי מס' השחקן קובעת את צבע האסימון, ולפי row\_col מציירת את האסימון במיקום המדויק בלוח.
* draw(self) – הפעולה העיקרית. מציירת את כל רכיבי הלוח ואת כל האסימונים בחלון המשחק.

**נספח נמצא בסוף תיק הפרויקט תחת "מחלקת Graphics"**

### המחלקה Connect4 – Environment

ייצוג סביבת המשחק. מחלקה זו מטפלת בלוגיקה של המשחק – בדיקות ניצחון ותיקו, מה קורה כאשר מטילים אסימון (פעולה).

**מאפיינים:**

* מצב – state – מצב הלוח הנוכחי

**פעולות עיקריות:**

* get\_actions(self, state: State = None) – מחזירה את כל הפעולות האפשריות עבור תור השחקן הנוכחי
* move(self, col, state: State) – מבצעת את פעולת השחקן. התור עובר לשחקן השני, ולוח המשחק מתעדכן. מחזירה אם הפעולה חוקית (אם לא, הפעולה לא עושה כלום).
* next\_state(self, col, state: State) – מחזירה את המצב הבא עבור המצב הנוכחי והפעולה שננקטה
* checkGameWin(self, state : State) – בודקת מצב ניצחון במשחק
* checkNInARow(self, state: State, n) – סוכמת את כל הרצפים של תור השחקן הנוכחים באורך n, ומחזירה את הסכום (משמש להערכה היוריסטית).
* get\_all\_next\_states (self, state: State) – מחזירה את כל המצבים הבאים האפשריים

והפעולות החוקיות (לפי כל הפעולות החוקיות של המצב הנוכחי ותור השחקן הנוכחי)

* toTensor (self, list\_states, device = torch.device('cpu')) – ממירה רשימה של מצבים ופעולות חוקיות לtensorים של הפעולות והמצבים.
* reward (self, state : State, action = None) – פעולת התגמול. מחזירה tupleשל התגמול וערך בוליאני התלוי אם המשחק הגיע לסופו או לא. התגמול הוא 1 עבור ניצחון, -1 עבור הפסד, ו-0 עבור תיקו.

**נספח נמצא בסוף תיק הפרויקט תחת "מחלקת Connect4"**

### המחלקה Human\_Agent

סוכן אנושי

**מאפיינים:**

* שחקן – player – השחקן המשחק (1 או -1)

**פעולות עיקריות:**

* get\_Action(self, event= None, state = None) – מחזירה את פעולת השחקן לפי קלט המקלדת (1-7)

### המחלקה Random\_Agent

סוכן רנדומלי

**מאפיינים:**

* שחקן – player – השחקן המשחק (1 או -1)

**פעולות עיקריות:**

* get\_Action(self, event= None, state = None) – מחזירה את פעולת הסוכן באקראי (1-7)

### הקובץ Game

הקובץ העיקרי להרצת משחק רגיל בין שני שחקנים (סוכנים).

**פעולות עיקריות:**

* main() – הפעולה העיקרית המריצה את המשחק בממשק pygame
* checkEndGame(player) – בודקת סיום משחק. מדפיסה הודעה חגיגית

# סוכן AI באמצעות אלגוריתם חיפוש

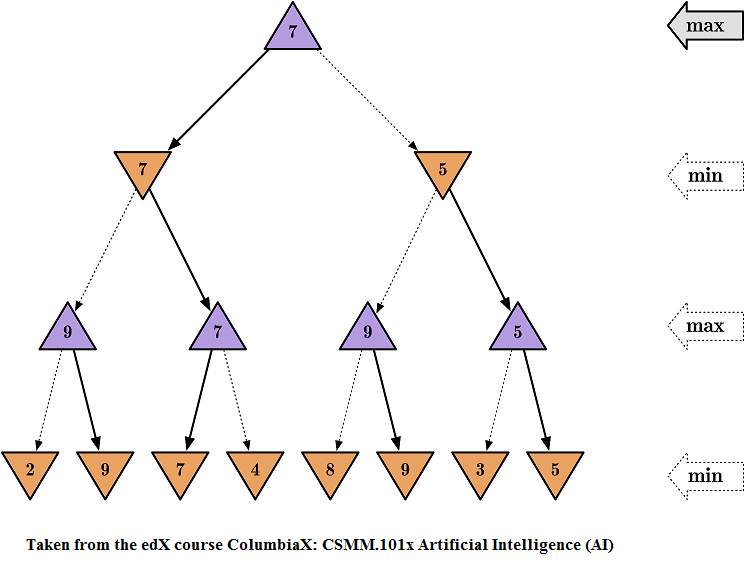
## סוכן minMax

### האלגוריתם minMax

אלגוריתם ה minMaxהוא אלגוריתם קבלת החלטות נפוץ בתורת המשחקים, במיוחד במשחקים של שני שחקנים עם מידע מושלם, כגון שחמט, דמקה או ארבע בשורה. המטרה העיקרית שלו היא לקבוע את המהלך הטוב ביותר עבור שחקן, בהנחה שגם היריב מבצע מהלכים אופטימליים.

כך עובד אלגוריתם minMax:

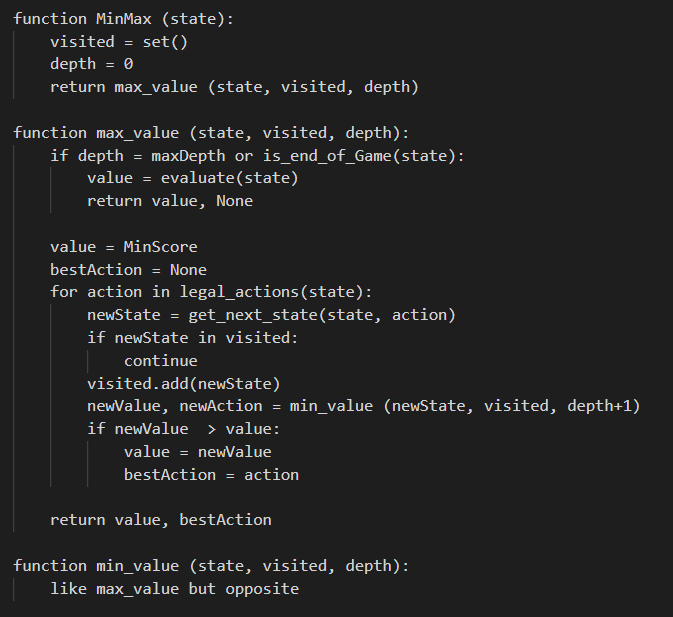
1. **ייצוג עץ:** האלגוריתם בונה עץ משחק המייצג את כל המהלכים ומהלכי הנגד האפשריים החל ממצב המשחק הנוכחי. כל רמה בעץ מתחלפת בין מהלכי השחקן למהלכי היריב.



1. **פונקציית הערכה (היוריסטית):** בצמתי העלים של העץ (כלומר, מצבי קצה או מצבים שבהם המשחק מסתיים), מופעלת פונקציית הערכה כדי להעריך את הערך של תוצאה זו עבור השחקן.
2. **חזרה לאחור:** לאחר מכן האלגוריתם מעריך באופן רקורסיבי את הצמתים של העץ, ועושה את דרכו חזרה מעלה מצמתי העלים. בכל רמה, הוא מחליף בין מיקסום למזעור פונקציית ההערכה: כאשר מגיע תורו של השחקן לזוז, הוא בוחר את המהלך שממקסם את פונקציית ההערכה (ומכאן ה"מקסימום" בminMax). כשזה תורו של היריב לזוז, הוא בוחר את המהלך שממזער את פונקציית ההערכה (ומכאן ה"מינימום" בminMax).
3. **מגבלת עומק:** כדי להפוך את האלגוריתם ליעיל, הוא משתמש לעתים קרובות במגבלת עומק, כלומר הוא חוקר רק מספר מסוים של רמות בעץ המשחק.

על ידי חקירה שיטתית של עץ המשחק וקבלת החלטות המבוססות על מזעור ההפסד הפוטנציאלי (לתרחיש הגרוע ביותר) ומקסום הרווח הפוטנציאלי (לתרחיש הטוב ביותר), אלגוריתם ה minMaxעוזר לקבוע את המהלך האופטימלי עבור שחקן ב-משחק סכום אפס מבוסס תורות (כמו ארבע בשורה).

**פסאודו קוד: (מתוך מצגת באתר giladmarkman)**



### מימוש האלגוריתם בפרויקט – מחלקת minMax\_Agent

**פעולות עיקריות:**

* evaluate(self, gameState : State) – פונקציית ההערכה היוריסטית. הפונקציה עושה שימוש בפונקציה ממחלקת Connect4 – CheckNInARow. ההערכה סוכמת את כמות רצפים של 3 של השחקן הנוכחי (עם ספירות כפולות) פחות הרצפים של 3 של היריב, וכמות הרצפים של 4 פי 10 (משקל פי 10) פחות הרצפים של 4 של היריב פי 10. ההערכה מסתמכת על כמה כל שחקן קרוב לניצחון (3 או 4 אסימונים ברצף כבר במצב הבא), ולכן זוהי הערכה טובה.
* get\_Action(self, state: State, train = False) – מחזירה את הפעולה בעלת הערך הכי גבוה לפי minMax
* minMax(self, state:State) – מימוש האלגוריתם minMax במחלקה. מחזירה את הערך הכי גבוה ואת הפעולה הכי טובה.
* max\_value(self, state:State, depth) – מקבלת את העומק המקסימלי לבדוק ומוצאת את הפעולה עם הערך המקסימלי עבור המצב שקיבלה.
* min\_value(self, state:State, depth) – מקבלת את העומק המקסימלי לבדוק ומוצאת את הפעולה עם הערך המינימלי עבור המצב שקיבלה.

### מסקנות למימוש האלגוריתם

* האלגוריתם יעיל ביותר עבור depth = 2. מעבר לכך הוא נהיה מסורבל (בשל זמן הריצה האיטי של בדיקת רצפים בכל הלוח – O(n^2))
* הסוכן משחק מצוין נגד הסוכן הרנדומלי, באחוז הצלחה של 90%. הTester מדפיס מס' ניצחונות של כל שחקן:



* משחק כנגד סוכן אדם – לא רע, אך לפעמים לא מזהה מצבי ניצחון. אפשר להפסיד לו.

**נספח נמצא בסוף תיק הפרויקט תחת "מחלקת minMax\_Agent"**

## סוכן alpha-beta

### האלגוריתם alpha\_beta

גיזום אלפא-בטא הוא טכניקת אופטימיזציה המשתמשת באלגוריתם minMax כדי להפחית את מספר הצמתים שיש להעריך בעץ המשחק. האלגוריתם עובד על ידי "גיזום" ענפים של העץ שידוע שערכיהם גרועים יותר מענפים שנבדקו בעבר, ובכך מזרז את תהליך החיפוש.

כך עובד גיזום אלפא בטא:

1. **ערכי אלפא וביטא**: האלגוריתם שומר על שני ערכים, אלפא וביטא, המייצגים את הגבולות של הציונים האפשריים של השחקן הממקסם (אלפא) והשחקן הממזער (בטא) בהתאמה. בתחילה, אלפא מוגדר לאינסוף שלילי ובטא מוגדר לאינסוף חיובי.
2. **בצמתי מקסימום**: בעת הערכת צומת מקסימום, אם האלגוריתם מוצא מהלך עם ערך גדול או שווה לבטא (כלומר, מהלך שהוא לפחות טוב כמו המהלך הטוב ביותר של היריב), אין צורך לחקור אותו. בנוסף, זה עוצר את החיפוש ומחזיר את המהלך הטוב ביותר הנוכחי שנמצא עד כה.
3. **בצמתי מינימום**: באופן דומה, בעת הערכה של צומת מינימום, אם האלגוריתם מוצא מהלך עם ערך קטן או שווה לאלפא (כלומר, מהלך טוב לפחות כמו המהלך הטוב ביותר של השחקן), הוא מפסיק לחקור את הענף הזה ומחזיר את המהלך הטוב ביותר הנוכחי שנמצא עד כה.
4. **עדכון אלפא וביטא**: ככל שהאלגוריתם מתקדם, הוא מעדכן את ערכי האלפא והבטא בהתאם: בצמתים מקסימליים: אם נמצא המהלך הטוב ביותר עד כה (כלומר, מהלך עם ערך גבוה יותר), אלפא מתעדכן להיות המקסימום של הערך הנוכחי שלו והערך של המהלך הטוב ביותר החדש. בצמתי מינימום: אם נמצא המהלך הטוב ביותר עד כה (כלומר, מהלך עם ערך נמוך יותר), בטא מתעדכן להיות המינימום של הערך הנוכחי שלו ושל הערך של המהלך הטוב ביותר החדש.

על ידי גיזום יעיל של ענפים של עץ המשחק שאינם יכולים להשפיע על הפעולה הסופית, גיזום אלפא-ביתא מפחית באופן משמעותי את המורכבות החישובית של אלגוריתם minMax, משום שחוסכים בדיקות מיותרות.

**פסאודו קוד: (לקוח ממצגת באתר giladmarkman)**

A computer screen shot of a program

Description automatically generated

### מימוש האלגוריתם בפרויקט – מחלקת AlphaBeta\_Agent

**פעולות עיקריות:**

* evaluate(self, gameState : State) – פונקציית ההערכה היוריסטית. הפונקציה עושה שימוש בפונקציה ממחלקת Connect4 – CheckNInARow. ההערכה סוכמת את כמות רצפים של 3 של השחקן הנוכחי (עם ספירות כפולות) פחות הרצפים של 3 של היריב, וכמות הרצפים של 4 פי 10 (משקל פי 10) פחות הרצפים של 4 של היריב פי 10. ההערכה מסתמכת על כמה כל שחקן קרוב לניצחון (3 או 4 אסימונים ברצף כבר במצב הבא), ולכן זוהי הערכה טובה.
* get\_Action(self, state: State, train = False) – מחזירה את הפעולה בעלת הערך הכי גבוה לפי minMax
* minMax(self, state:State) – מימוש האלגוריתם minMax במחלקה. מחזירה את הערך הכי גבוה ואת הפעולה הכי טובה. בנוסף מגדירה את אלפא ובטא
* max\_value(self, state:State, depth, alpha, beta) – מקבלת את העומק המקסימלי לבדוק ומוצאת את הפעולה עם הערך המקסימלי עבור המצב שקיבלה.
* min\_value(self, state:State, depth, alpha, beta) – מקבלת את העומק המקסימלי לבדוק ומוצאת את הפעולה עם הערך המינימלי עבור המצב שקיבלה.

### מסקנות למימוש האלגוריתם

* האלגוריתם גם יעיל ביותר עבור depth = 2, אך אפשר גם להשתמש ב3 ללא בעיה משום שהאלגוריתם יותר יעיל מminMax בעזרת הגיזום.
* לסוכן אותם אחוזי ניצחון מול סוכן רנדומלי (90%) – הרי מעשית האלגוריתם זהה לminMax, האלגוריתם פשוט יותר יעיל.

**נספח נמצא בסוף תיק הפרויקט תחת "מחלקת AlphaBeta\_Agent"**

# סוכן Reinforcement - DQN

## למידת חיזוק

### מודל MDP – כללי

מודל MDP – Markov Decision Process הוא מודל המבוסס על מודל סביבה-סוכן (כפי שתואר קודם לכן) אשר באמצעותו ניתן לתאר משחקים כשרשרת מצבים ופעולות (כמו מודל סביבה-סוכן) אך לכל פעולה יש תגמול (reward) שהוא בעצם ערך הפעולה שאותו מגדירים בד"כ במצבים סופיים (1 לניצחון, -1 להפסד ו-0 לתיקו). ערך מצב (לאו דווקא סופי) מחושב ע"י משוואות בלמן. מודל זה משומש הרבה בלמידת חיזוק.

**רכיבי המודל:**

* סביבה – (Environment) מתאר את העולם בו אנו נמצאים - חוקי העולם / חוקי המשחק
* סוכן – (Agent) הרובוט אותו אנחנו מתכנתים. תוכנת המחשב הפועלת בעולם זה. הסוכן קולט את הסביבה ומבצע פעולות המשנות את הסביבה.
* מצב – (State) תיאור של מצב רגעי של העולם. המצב מתאר "הקפאה" של הסביבה.
* פעולה – (Action) פעולות אותן מבצע הסוכן המשנות את הסביבה, צעדי המשחק.
* תגמול (Reward) – תגמול מספרי שהסוכן מקבל בעת ביצוע פעולה. התגמול יכול להיות חיובי או שלילי או אפס.

**מטרת המודל:** לאפשר כימוס של מחלקות המשחק ולאפשר לכל סוכן "להשתפר" – כלומר למקסם את התגמול הכולל עבורו (תגמול כולל – סך כל התגמולים על כל הפעולות במשחק עד לסוף המשחק (תיקו, הפסד או ניצחון)) ולמצוא את המדיניות (Policy) הכי טובה.

### מהי למידת חיזוק ומשוואות בלמן

למידת חיזוק (RL) היא סוג של למידת מכונה שבה סוכן לומד לקבל החלטות על ידי אינטראקציה עם סביבה כדי למקסם את התגמולים המצטברים. בעצם, מטרת הסוכן היא למצוא מדיניות (Policy) אופטימלית, כאשר בכל צעד הסוכן בוחר את הצעד הכי טוב עבורו (בעל ה"תגמול" הכי גבוה). כיצד מחשבים את התגמול הזה עבור כל מצב? הרי לא אחרי כל פעולה מתקבל ניצחון או הפסד. חישוב זה מחושב ע"י משוואות בלמן, המתארות את התגמול עבור פעולה מסוימת כאשר היא לא בהכרח מובילה לסוף המשחק. התגמול מחושב ע"י המשוואה הבאה (משוואת בלמן):

כאמור, הערכה זו היא לא מדויקת לגמרי, אך בלמן הוכיח כי אם נלמד ונפעיל את הנוסחה הזאת מספיק פעמים הערכים יתכנסו לערכים אופטימליים. גאמא הוא מקדם של הערך העתידי, כלומר הערך של מצב מסוים שווה לתגמול של הפעולה ועוד גאמא כפול הערך של המצב הבא. משתמשים במקדם זה כדי להעריך מצבים טובים לעומת פחות טובים (למשל אם נגיע לניצחון תוך 5 צעדים זה יותר טוב מאשר אם נגיע אליו ב-20 צעדים).

### האלגוריתם Q-Learning

Q-learning הוא אלגוריתם למידת חיזוק ללא מודל (Model free) המשמש למציאת מדיניות בחירת פעולה אופטימלית עבור תהליך החלטה סופי של מרקוב (MDP) נתון. הוא לומד פונקציה ערכית Q(s,a), המייצגת את התגמול המצטבר הצפוי של נקיטת פעולה a במצב s ולאחר מכן ביצוע מדיניות אופטימלית. בעצם, האלגוריתם מחזיק בטבלה ענקית בשם Q-Table אשר שומרת את הערכים של כל הזוגות של מצבים ופעולות (s,a) כאשר לאט לאט הטבלה מתמלאת. הטבלה משמשת אותנו כדי לזכור ערכים של מצבים ובאמצעותה נבחר פעולה אופטימלית.

**מהלך האלגוריתם:**

1. **אתחול:** אתחול ערכי ה-Q לאפסים. (ערכי Q – ערכי הQ-Table)
2. **בחירת פעולה:** הסוכן בוחר פעולה המבוססת על אסטרטגיית חקירה-ניצול

(Exploitation vs Exploration) כמו epsilon-greedy. האלגוריתם בעצם בוחר תחילה בפעולה רנדומלית (Exploration) בסבירות גבוהה כאשר לאט לאט הסבירות יורדת (בעצם ערך האפסילון יורד (decay) – ערך האפסילון מציג את הסבירות שהסוכן יבחר בפעולה רנדומלית) והסבירות לבחירת פעולה מטבלת הQ-Table עולה (Exploitation).

1. **למידה:** לאחר ביצוע פעולה, הסוכן מחשב את התגמול r והמצב הבא s’. לאחר מכן הוא מעדכן את ערך ה-Q עבור צמד המצב-פעולה הקודם באמצעות כלל העדכון של

Q-learning:

כלל העדכון מבוסס על משוואת בלמן כאשר בעצם נעדכן את הערך להיות הערך ועוד ההפרש הטמפורלי (TD – Temporal Difference) בין הערך האופטימלי בקירוב (לפי משוואת בלמן) לבין הערך הנוכחי, ואת כל זה כמובן נכפיל בקצב הלמידה.

## למידת חיזוק עמוקה DQN

### האלגוריתם DQN

DQN (Deep Q-Learning) הוא אלגוריתם רב עוצמה בתחום למידת החיזוק שפותח על ידי צוות חוקרים בGoogle Deepmind ופורסם בשני מאמרים בשנת 2013 ו-2015:

[Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).](https://www.nature.com/articles/nature14236)

[Playing Atari with Deep Reinforcement Learning, V. Minh (NIPS-DLW 2013)](https://arxiv.org/abs/1312.5602)

האלגוריתם משלב את העקרונות של רשתות נוירונים עמוקות עם Q-learning, ומאפשר לסוכנים ללמוד מדיניות אופטימלית בסביבות מורכבות, כאשר לא ניתן להחזיק Q-Table (עבור משחק כמו ארבע בשורה, יש להחזיק 4.5 טריליון מצבים שונים, ולכל אחד את הפעולות האפשריות שלו, וכמובן שזה לא ניתן ליישום אופטימלי). בעצם, במקום להשתמש בטבלה, נשתמש ברשת נוירונים אשר מתעדכנת בדומה לכלל העדכון של Q-Learning: נסמן פונקציית רשת הנוירונים המחליפה את Q:

כאשר w הוא ערך המשקלים של רשת הנוירונים. רשת הנוירונים בעצם מקבלת tensor של המצב והפעולה (s,a) ומחזירה פלט (Q-Value) שהוא בעצם ערך הפעולה במצב הנוכחי. בעיה נוספת נעוצה בעבודה שבאימון רשת נוירונים ערכי המטרה (target) הם תמיד קבועים ולא משתנים. ערך המטרה שלנו משתנה בכל פעם שאנחנו משנים את הפרמטרים של הרשת. נעשה שימוש בשתי רשתות נוירונים בעלות אותו מבנה אך עם פרמטרים שונים: רשת אחת משמשת לבחירת הצעדים ואותה אנחנו מעדכנים בכל איטרציה (איטרציה – לולאה אחת, לקיחת batch אחד והכנסתו לרשת – ראה בהמשך). רשת שניה משמשת לבחירת המטרות והיא קבועה למשך C איטרציות, כאשר לאחר C איטרציות אנחנו מעדכנים את משקלי רשת המטרות למשקלי הרשת הראשונה. ככה בעצם מתנהל אימון רשת הנוירונים. כאשר אנו רוצים להשתמש ברשת, נבחר פעולה לפי epsilon-greedy, אם לא בוחרים פעולה רנדומלית נבחר את הפעולה שהרשת מניבה ממנה את הערך הכי גבוה – כלומר נבדוק את כל זוגות המצב-פעולה האפשריים עבור כל הפעולות החוקיות ממצב מסוים ונחזיר את הפעולה אשר ערך המצב של הזוג הזה הוא הכי גבוה.

### Replay buffer

כפי שהוזכר קודם, בכל איטרציה נלקח batch – אוסף של שרשראות s,a,r,s’, (מצב, פעולה, תגמול והמצב הבא) כאשר לכל batch יש גודל קבוע. אך מאיפה לוקחים את השרשראות האלה? כדי להבין זאת נלך טיפה אחורה. אחת הבעיות המרכזיות ביישום רשת נוירונים באלגוריתם Q-Learning המשתמש ב-bootstrapping הוא העובדה שהדגימות שלנו מהסביבה תלויות אחת בשניה - בעת האימון של הסוכן אנחנו יוצרים שרשרת של מצבים ופעולות אשר אחד נובע מהשני. בנוסף, הדגימות רלוונטיות למשחק ולמדיניות מסויימת ואינה אקראית. על מנת שלמידת מכונה תצליח על הקלט שלנו למלא אחר התנאי: i.i.d - Independent and Identically Distributed

לכן, נעשה שימוש במבנה נתונים הנקרא Replay Buffer השומר דגימות אשר ניקח מהסביבה. ולסיכום, עדכון רשת הנוירונים יעשה באמצעות batches אקראיים אשר יילקחו ממבנה הנתונים הזה לאחר שנמלא אותו מעט. בנוסף, מבנה הנתונים יהיה מוגבל בגודלו כך שיישמרו בו תמיד הנתונים האחרונים שנדגמו בהתאם למדיניות המעודכנת שלנו.

### נוסחת בלמן בלמידת חיזוק עמוקה

בלמידת החיזוק שלנו אנו משתמשים בכלל העדכון כמו בQ-Learning, ולכן גם משוואת בלמן תהיה דומה רק שבמקום לעדכן את ערכי הQ-Table נעדכן את ערכי פונקציית הרשת נוירונים, בהתאם למשוואה שתיארנו קודם:

ומכאן נקבל את פונקציית הטעות שבעזרתה נעדכן ונאמן את רשת הנוירונים:

פונקציה זו היא בעצם MSELoss הממומשת על משוואת בלמן על רשת הנוירונים שלנו.

## מימוש DQN

### כללי

אלגוריתם הDQN מומש באמצעות PyTorch, על מנת ליצור את רשת הנוירונים ולתחזק אותה. ישנן 3 מחלקות שונות לאימון הסוכנים – Trainer\_Red, Trainer\_Blue, Trainer\_RB.

כל המחלקות הללו משתמשות ברשת הנוירונים ממחלקת DQN אשר נגדיר לה פרמטרים בתוך המחלקה (גודל ועומק הרשת, וערך גאמא עבור חישוב ערך עתידי בפונקציית הטעות) , וגם את הפרמטרים של האימון (קצב למידה, גודל batch, וכו') ניתן להגדיר בקבצי המחלקות הנ"ל. בהרצת כל אחד מהקבצים הללו מתקיימת למידה כאשר תחילה נמלא את ה-replay buffer הממומש כתור (Queue) לגודל מינימום (הסוכן ישחק לפי המדיניות שלו) ואז נלמד את הרשת, כאשר כל פעם נדגום batch מ-replay buffer ונעביר אותו ברשת הנוירונים, ונעדכן את משקלי הרשת (באמצעות optim). כל כמות מסוימת של איטרציות (epochים) אנו נשמור את הפרמטרים של הרשת, או נבחן את רמת הסוכן מול סוכן רנדומלי כדי לעקוב אחר התקדמותו. מידע האימון נשמר בקבצים שהוגדרו מראש לפרמטרים של הרשת ועוד נתונים (ראה מדריך למשתמש – אימון הרשת). בנוסף, כל מחלקה משמשת לאימון של שחקן ספציפי או שניהם (Trainer\_RB) ולכן יש שימוש במחלקות שונות לאימון כל שחקן בנפרד.

### המחלקה DQN – רשת הנוירונים שנבחרה.

**משתנים גלובליים (פרמטרים של הרשת):**

* גודל קלט – input\_size – גודל הקלט שהרשת מקבלת. עבור המשחק גודל הקלט הוא 43 (גודל הלוח: 6\*7 ועוד גודל הפעולה – 1)
* גודל שכבה נסתרת – layer1 – גודל שכבה נסתרת מס' 1. נבחרה להיות 128.
* גודל פלט – output\_size – גודל הפלט נבחר להיות 1 – מס' הפעולה מ-1 עד 7.
* גאמא – gamma – פקטור גאמא המשומש בפונקציית הטעות לחישוב הערך העתידי לפי נוסחת בלמן. נבחר להיות 0.99

**פעולות עיקריות:**

* forward (self, x) – פעולת הforward של רשת הנוירונים. משתמשת בשכבה נסתרת אחת ובפונקציית אקטיבציה מסוג LeakyReLU.
* loss (self, Q\_value, rewards, Q\_next\_Values, Dones ) – פונקציית הטעות של הרשת. מחשבת את הטעות לפי משוואת בלמן וMSELoss.

### המחלקה DQN\_Agent

**מאפיינים:**

* רשת – DQN – רשת הנוירונים של הסוכן
* שחקן – player – השחקן שהסוכן משחק (1 או -1)
* מצב אימון – train – משתנה בוליאני המחזיק את מצב הסוכן. אם הסוכן לא במצב אימון, הוא מחזיר פעולה רק לפי הרשת ולא עם epsilon-greedy.
* סביבה – env – סביבת הסוכן

**פעולות עיקריות:**

* get\_Action (self, state, epoch, events, train, blue\_state) – מחזירה את פעולת הסוכן, לפי epsilon-greedy (מרשת הנוירונים או פעולה רנדומלית, תלוי בערך אפסילון).
* epsilon\_greedy(self, epoch, start, final, decay=epsilon\_decay) – פונקציית epsilon-greedy המחשבת את ערך אפסילון (סבירות לבחור פעולה מהרשת) לפי מספר האיטרציה (epoch) וקצב הדעיכה (epsilon-decay) אשר מוגדר במחלקה Constant.

### המחלקה Replay\_buffer

**משתנים גלובליים (פרמטרים של מבנה הנתונים):**

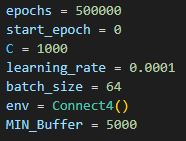
* גודל replay buffer – capacity – גודל מבנה הנתונים (תא אחד הוא שרשרת אחת של s,a,r,s’ (מצב, פעולה, תגמול, מצב הבא), וגם משתנה בוליאני done אם זה מהלך סופי.
* קדימות סוף – end\_priority – כמות הפעמים הנוספות שמוסיפים שרשרת של פעולה סופית ל-replay buffer – פעולות סופיות בעצם מקבלות יותר כובד ב-replay buffer. משתנה זה בעצם משנה את כובד הפעולות האלה ביחס לשאר.

**פעולות עיקריות:**

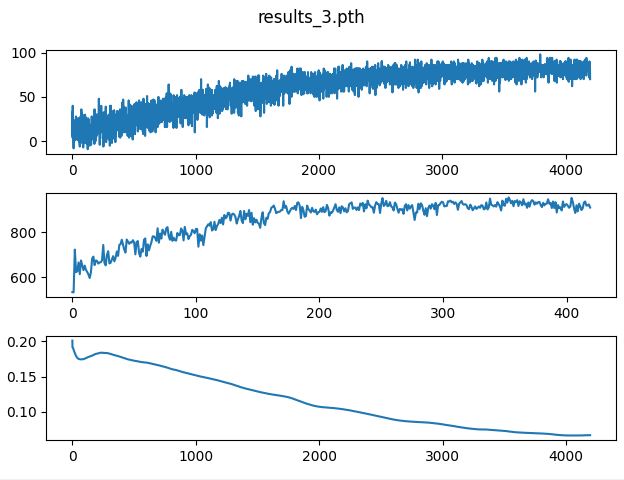
* push (self, state : State, action, reward, next\_state: State, done) – מוסיפה שרשרת לתוך ה-replay buffer
* sample (self, batch\_size) – דוגמת כמות מסוימת של שרשראות (batch\_size) ומחזירה אותן כtensors (שיהיה אפשר להכניס אותן לתוך הרשת).

### המחלקה Trainer\_Red

אימון השחקן הראשון (האדום). האימון נעשה נגד סוכן פשוט וממומש כנגד סוכן רנדומלי. באימון הזה ניסיתי עומקי רשתות שונים, ושיחקתי עם כל הפרמטרים, בין אם זה גודל ועומק הרשת, לבין הדעיכה של epsilon-greedy, וכמובן כל הפרמטרים של האימון. הפרמטרים הכי טובים לאימון שמצאתי הם להלן:



כאשר גודל ועומק רשת הנוירונים פורט בפרק "מחלקת DQN". קצב הדעיכה של אפסילון נבחר להיות 300,000. אימון שחקן זה היה מוצלח ביותר, והסוכן למד לשחק מאוד טוב נגד סוכן רנדומלי כאשר הסוכן מתחיל (שחקן ראשון), והגיע לאחוזי ניצחון של 96% מולו.



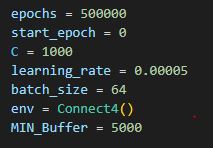
גרף 1 - אחוזי הניצחון מול סוכן רנדומלי (ציר אופקי - כמות האיטרציות (במאות).

גרף 2 - גם אחוזי ניצחון מול סוכן רנדומלי, אך בממוצע (ציר אופקי – כמות האיטרציות (באלפים).

גרף 3 – הטעות (loss) (ציר אופקי – כמות האיטרציות (במאות).

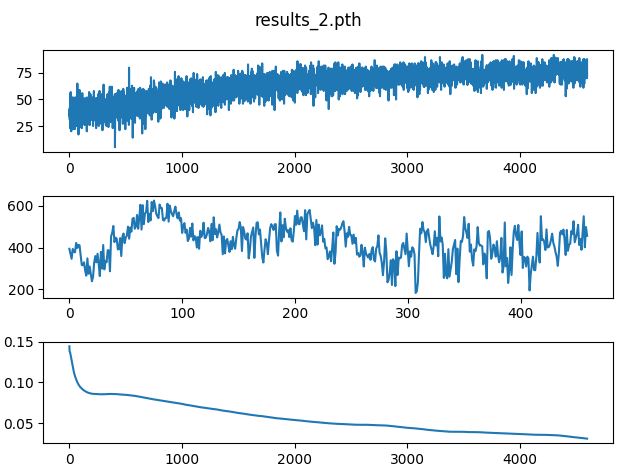
### המחלקה Trainer\_Blue

אימון השחקן השני (הכחול). האימון נעשה נגד סוכן פשוט וממומש כנגד סוכן רנדומלי כמו הסוכן הקודם. באימון הזה גם ניסיתי את כל הפרמטרים השונים ואלו הפרמטרים האידיאלים לאימון שמצאתי:



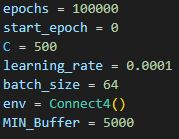
כאשר גודל ועומק רשת הנוירונים פורט בפרק "מחלקת DQN".

אימון שחקן זה היה פחות מוצלח, והסוכן לא הגיע לרמת הצלחה גבוהה מול הסוכן הרנדומלי. אני סבור שהסיבה היא שלשחקן השני יש חסרון כי הוא לא מתחיל את המשחק. בגלל שהשחקן הכחול הוא השחקן השני ורשת הנוירונים של הסוכן היא כביכול של השחקן הראשון (האדום), בכל איטרציה היה צורך לבצע היפוך של הלוח כאילו השחקן הראשון משחק, ולהכניס דגימות הפוכות אלו לרשת הנוירונים.



### המחלקה Trainer\_RB

אימון שני השחקנים נגד השני בעזרת רשת נוירונים משותפת. השחקן האדום התאמן כרגיל והשחקן הכחול התאמן כמו מקודם, כאשר מתבצע היפוך עבור כל המצבים שלו, כדי שכביכול יתאמן על אותה הרשת (המקבלת מצבים של שחקן אדום). כל שחקן היה סוכן DQN שונה. מטרת רשת הנוירונים המשותפת הייתה לאמן את שני השחקנים בו זמנית כדי שהסוכן יידע לשחק את שני השחקנים ברמה גבוהה. אימון זה הפיק את הסוכן הכי טוב, כאשר נבחנו הרבה פרמטרים (ניסיתי גם 100,000 איטרציות, ו-500,000, בסוף נמצא כי 100,000 הפיק תוצאות טובות יותר, כי לאחר הרבה איטרציות הלמידה כבר התחילה להיות לא יציבה – היו פרקים שהסוכן נהיה יותר גרוע) , ואלו הפרמטרים הכי טובים שמצאתי:



כאשר גודל ועומק רשת הנוירונים פורט בפרק "מחלקת DQN" (גודל ועומק הרשת היה אופטימלי עבור כל האימונים, לכן השתמשתי באותה הרשת לכל האימונים). אימון הסוכן הזה היה הכי מוצלח מכולם, משום שסוכן זה השתפר ושיחק מול עצמו, מה שאיפשר לו למצוא אסטרטגיות שונות והוא הגיע לרמה שהוא הצליח לנצח את סוכן אלפא-בטא כאשר הוא משחק ראשון (כאשר הוא משחק שני – לפעמים מנצח, תלוי בפרמטרים הנוכחיים). מול הסוכן הרנדומלי הוא גם הצליח: 88% כשחקן ראשון, 80% כשחקן שני.

A graph of a graph of a graph

Description automatically generated with medium confidence

יש לשים לב כי בגרף זה אחוזי הניצחון של הסוכן מול עצמו (גרף 1) כל הזמן משתנים (הרי כל שחקן משתפר ולא קבוע)

# תוצאות ומסקנות המחקר

**גודל ועומק רשת אופטימלית:**

נמצא כי גודל ועומק הרשת האופטימלית הוא גם אופטימלי עבור כל האימונים – לא רק אימון ספציפי אחד. משום שארבע בשורה משחק פשוט, לא נדרשה רשת עמוקה, ומצאתי כי רק שכבה נסתרת אחת בגודל 128 היא אופטימלית. ככל שעומק הרשת גדל, גם זמן האימון והזמן שלוקח לסוכן להשתפר (כי יש יותר משקלים).

**פרמטרים של האימון:**

נמצא כי batch\_size של 64 הוא אופטימלי לאימון. ככל שbatch\_size קטן, האימון מתבצע מהר יותר (כי כל איטרציה משתמשת בפחות שרשראות מreplay buffer). אפילו ניסיתי אימון עם 20, אשר הוא מאוד מהיר (מריץ בערך 10 איטרציות בשנייה), אך לא מאמן לרמה טובה. בנוסף, בטווח של 20-100 אלף איטרציות נמצא כי האימון היה הכי אפקטיבי, והסוכנים השתפרו הכי הרבה. לאחר מכן, או שהאימון לא השפיע (הסוכן הפסיק להשתפר) או שהסוכן התחיל גם להידרדר, במיוחד במקרה של אימון סוכן מול עצמו (כי הוא ניסה אסטרטגיות שונות לשבור את השיוויון בינו לבין עצמו ולמעשה הרע). היה צריך למצוא את הsweet spot כך שאורך האימון לא יהיה ארוך מדי אך שגם יהיה ברמה גבוהה בסוף. בנוסף, חקרתי כי קצב הלמידה האופטימלי גם צריך להיות קטן אך לא קטן מדי, כדי שהסוכן ישאר יציב ולא פתאום ידרדר ממש, אך גם שישתפר. גם כן, קצב עדכון הרשתות אחת לשנייה (C) נמצא להיות אופטימלי כאשר מעדכנים פעם ב-1000 איטרציות. היה חשוב שהקצב הזה יתעדכן באופן תדיר כדי שהסוכן ישתפר, אך לא באופן תדיר מדי כדי שהוא לא ישתפר כלל (כי הוא ישחק מול עצמו לפני כמות קטנה של איטרציות – שהם כמעט אותם הסוכנים).

**סוגי אימון:**

בחנתי שני סוגי אימון. הראשון – אימון מול סוכן רנדומלי, כאשר אימנתי שחקן אחד בבת אחת מול סוכן רנדומלי. הסוכן של השחקן הראשון למד לשחק מאוד טוב נגד הסוכן הספציפי הזה אך לא למד לשחק טוב מול אלפא-בטא, כי הרי הסוכן השתפר רק מול הסוכן הספציפי הזה ופיתח אסטרטגיות בייחוד אליו (overfitting). סוג האימון השני היה אימון של סוכן מול עצמו. אימון זה הוכיח את עצמו כסוג האימון האופטימלי, כאשר הסוכן משחק מול עצמו הוא לומד דינמית, ולא ספציפית מול סוכן שאינו משתפר ומשנה את עצמו. ככה למעשה הסוכן הזה הצליח אפילו לנצח את סוכן החכם אלפא-בטא. בכללי, סוכן האלפא-בטא היה יותר טוב מסוכני הDQN משום שיש לו אסטרטגיית ניצחון טובה בהרבה יותר - הוא מעריך ערכי מצבים בדיוק רב יותר מהרשת ברוב המקרים (באמצעות היוריסטיקה), אך באימון הסוכן מול עצמו הוא התעלה על הערכות ההיוריסטיקה של סוכן האלפא-בטא, שזה מאוד מרשים בעיניי בהתחשב בכך שפונקציית ההיוריסטיקה נכתבה ע"י בן אדם.

# רפלקציה

## מדוע בחרת בפרויקט זה

בחרתי בפרויקט זה (ובמגמה בכללי) משום שבינה מלאכותית זה תחום שמאוד עניין אותי ורציתי להתעמק בו. בחרתי במשחק ארבע בשורה כי זה משחק ששיחקתי בו הרבה בילדותי ואני מכיר אותו היטב. בנוסף, חשבתי שזה יהיה מעניין לראות איך בינה מלאכותית תלמד לשחק משחק פשוט יחסית שקשה לחשוב על אסטרטגיה מוצקת חוץ מלחסום את היריב ולנסות ליצור רביעייה.

## תיאור תהליך התכנון

תיכננתי את המשחק בשלבים כך שקודם אני אבנה את המשחק כמשחק בסיסי עם סוכנים אנושיים ולבנות על זה עם סוכנים חכמים ובהמשך גם DQN. בהתחלה פיתחתי את ממשק המשחק באמצעות pygame, ואת מודל הסביבה-סוכן הבסיסי עבור המשחק שלי כדי שאני יוכל לפתח אותו אחר כך לסוכני DQN. תחילה פיתחתי סוכנים פשוטים כמו לאחר מכן פיתחתי סוכנים חכמים כגון אלפא-בטא. לבסוף, פיתחתי את מחלקות האימונים ואת סוכני הDQN, שישבו טוב על מודל הסביבה-סוכן שפותח בהתחלה.

## התמודדות עם אתגרים שבדרך

האתגרים הקשים היו ללמוד את החומר התיאורטי בהתחלה ולהבין תחום שהוא מאוד חדש ומסובך. בהתחלה זה היה ללמוד את ממשק pygame ואיך ליצור חלון משחק עם frames וגרפיקה, וללמוד איך עובד מודל סביבה-סוכן כדי שאני יוכל לממש אותו במשחק. לאחר מכן, זה היה ללמוד את האלגוריתמים של הסוכנים החכמים ולפתח אותם (היו מלא באגים).

אני חושב אבל שבפרט החלק הכי קשה היה אימון הסוכנים, וללמוד את החומר התיאורטי כיצד לפתח רשת נוירונים בעצם על המשחק הפשוט שבניתי. אימון הסוכנים היה תהליך ארוך וכל פעם שסוכן לא עובד טוב בכל מקרה צריך לחכות כמה שעות כדי שאני אראה שהוא באמת לא מצליח להתאמן, וזה תהליך שפשוט לקח מלא זמן.

הקפדתי לאורך כל הדרך להתרכז ולהתחבר לאינטואיציה של החומר התיאורטי, ולנסות להבין איך גם לשפר את האימון בסוף ולהגיע למצב אופטימלי.

## דברים שלמדתי על עצמי בעקבות הפרויקט.

אני למדתי שאני מאוד מתחבר לתחום של בינה מלאכותית, והוא מאוד מעניין אותי גם כי זה תחום העתיד שמלא דברים יתבססו עליו כנראה, ולכן זה גם מאוד חשוב לי ללמוד עליו כי הוא יהיה בקדמת הבמה לשנים הקרובות.

## תודות

אני רוצה להוקיר תודה לגלעד מרקמן המורה המלך, שעזר לי להתחבר לחומר ולהבין אותו וכל פעם ניגש לעזור בפרטני לכל מי שהיה צריך ועזר למצוא תיקונים שבלעדיו היו לוקחים שנים למצוא.

# ביבליוגרפיה

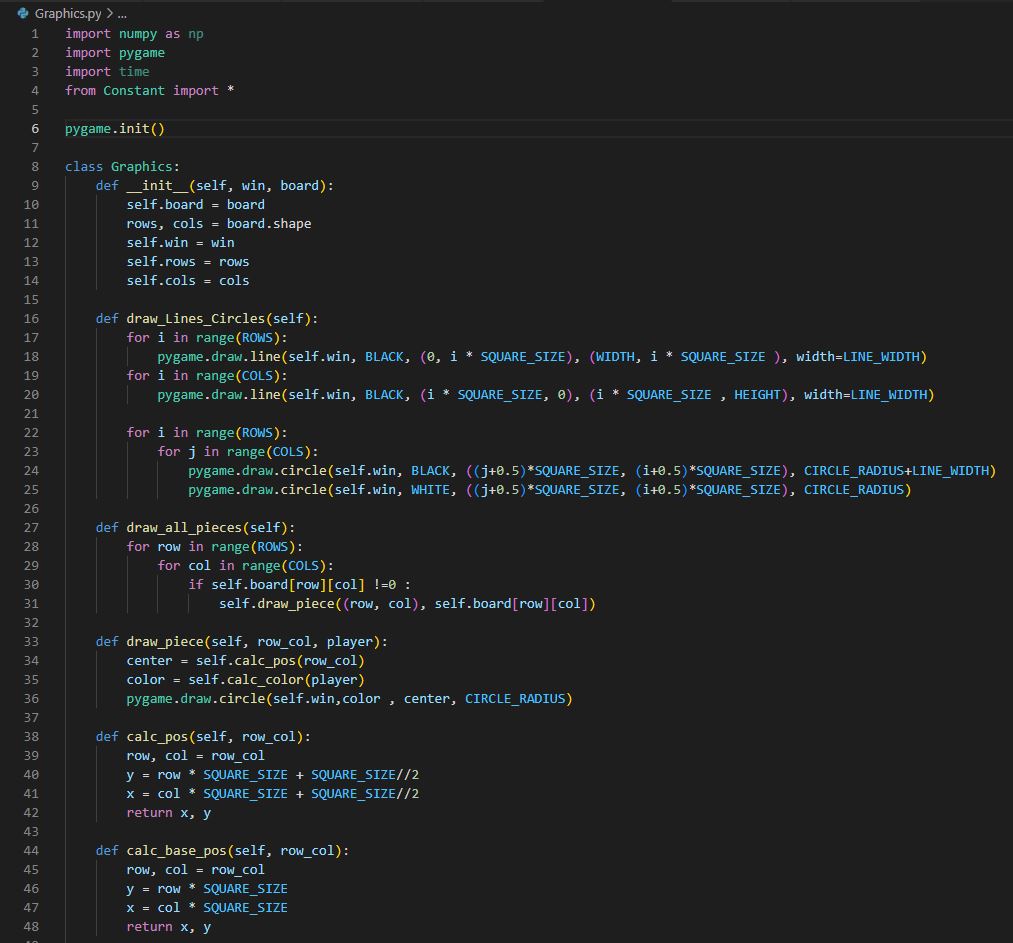
אתר גלעד מרקמן: <https://webprogramming.azurewebsites.net/Pages/Main/Home.aspx>

מאמר על DQN:

<https://medium.com/@shruti.dhumne/deep-q-network-dqn-90e1a8799871>

# נספחים

## המחלקה Graphics



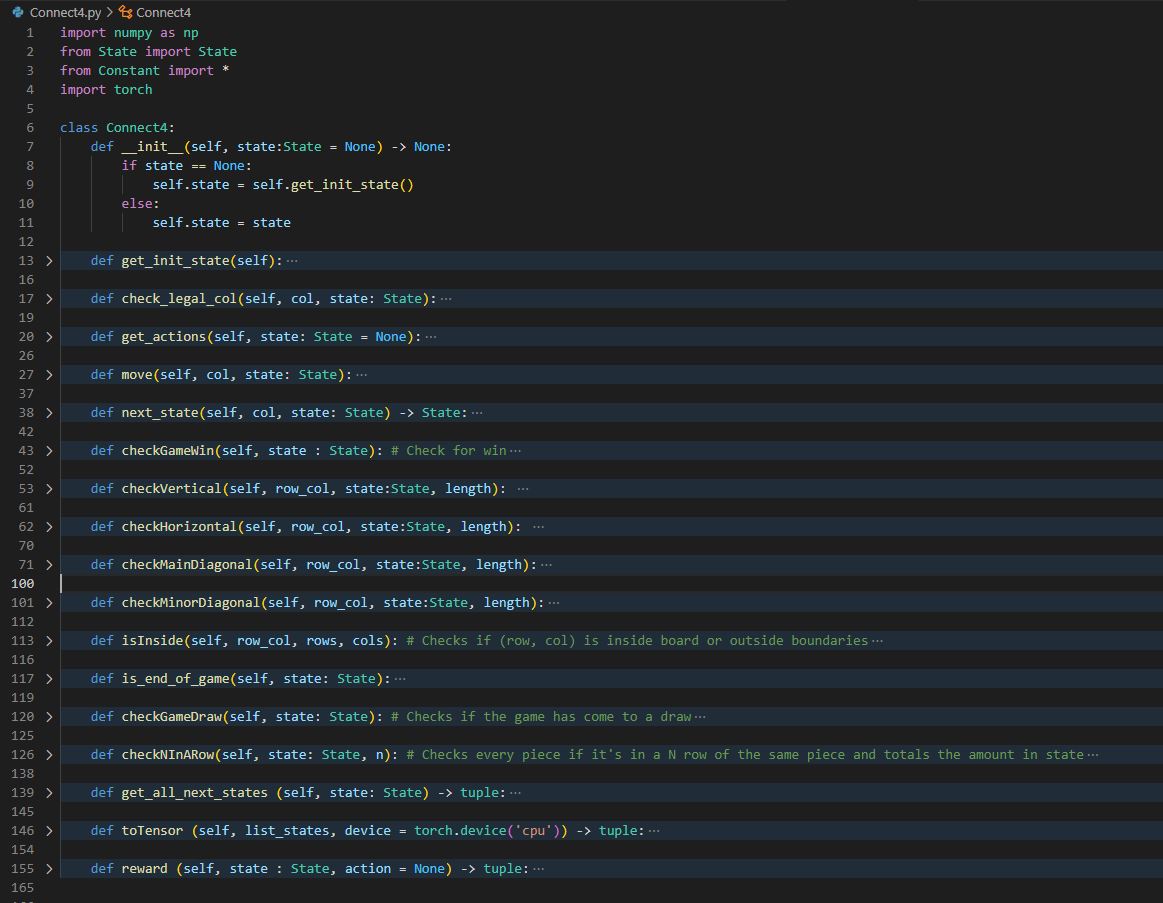
A screen shot of a computer code

Description automatically generated

## המחלקה State



## המחלקה Connect4 – הסביבה



## A computer screen with text and images Description automatically generatedסוכן AI – minMax

A screen shot of a computer program

Description automatically generated

## A screen shot of a computer program Description automatically generatedסוכן AI– alpha\_beta

A screen shot of a computer program

Description automatically generated