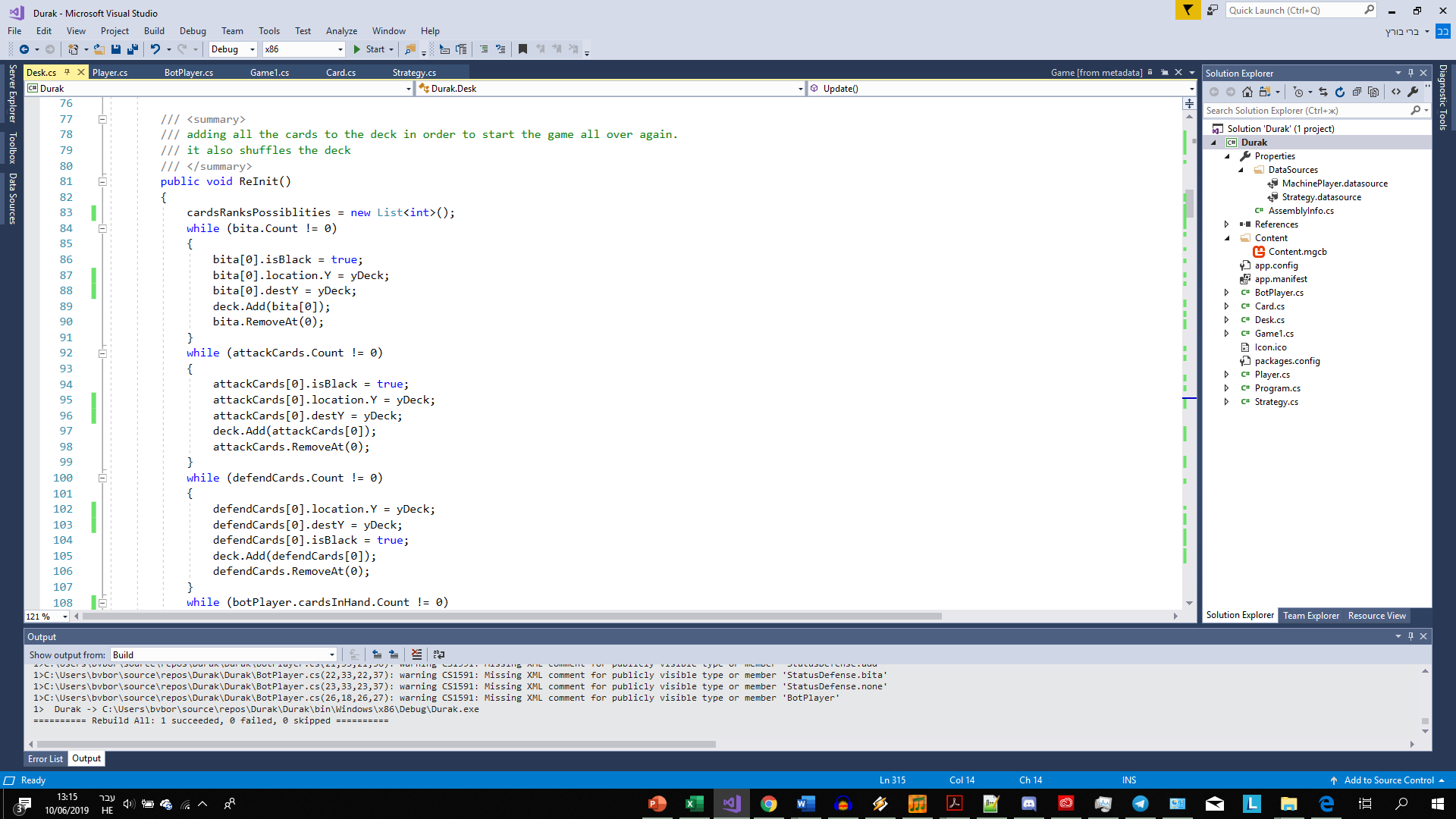
**מבוא**

שמי ברי בורץ ובמסגרת פרויקט של חמש יחידות מדעי המחשב בחרתי לממש משחק קלפים רוסי פופולרי עם בינה מלאכותית ב-C# בעזרת חבילת התוכנה Monogame.

מאז שהייתי ילד אני משחק במשחק הזה עם המשפחה. בנוסף, יש תלמידים רבים בכיתה שלנו שאוהבים את המשחק ולכן אנחנו משחקים בו בכל הפסקה. תמיד הייתי טוב במשחק ותמיד עניינה אותי השאלה: מהי הנוסחה להצליח במשחק? באיזו טקטיקה כדאי להשתמש? אני ניסיתי טקטיקות שונות ועם הזמן הבנתי מהי הטקטיקה המתאימה ביותר. כבר שנים רבות חשבתי על הרעיון שאני ייצור משחק "דוראק" בעצמי. ידעתי שתהליך בניית המשחק יהיה קשה ומורכב אך אף פעם לא חששתי מעבודה קשה. הסביבה שעבדתי הייתה מאוד נוחה להבנה והיא הקלה את העבודה שלי.



**חלק תיאורטי:**

בחלק התיאורטי אסביר בקיצור את המושגים הקשורים לעבודה שלי. כאמור לעיל, במהלך העבודה השתמשתי בבינה מלאכותית בסביבת עבודה Monogame. יש מספר אלגוריתמים שונים בתחום של בינה מלאכותית ותהליך הבחירה היה לא פשוט. ML.NET זה תוסף ל- C# שיכול לממש אלגוריתמים של למידת מכונה ואחרי שבחנתי את השימושים שהוא מציע הגעתי למסקנה שהוא לא מתאים למשחק שאני מממש. בנוסף החלטתי לא להשתמש באלגוריתם הידוע minimax, שיוצר עץ שכולל את כל המהלכים האפשריים ומזה מסיק מהו המהלך האופטימלי. מכיוון שבמשחק שלי יש יותר מדי אפשרויות עדיף להשתמש בגישה היוריסטית. המשכתי לחפש ובסוף בחרתי באלגוריתם Q-learning שמשתמש בלמידת חיזוק (Reinforcement learning).

**Monogame ו-****C#**

C# היא שפת תכנות עילית מונחת עצמים טהורה שפותחה על ידי מייקרוסופט במסגרת פרויקט "דוט נט". דוט נט היא שכבת תוכנה שמספקת כלים ושירותים לתוכנות שונות. השפה המרכזית שנתמכת על ידי פלטפורמה זאת היא C#.

Monogame זו חבילת תוכנה שעוזרת לפתח משחקי מחשב ב- .C#היא מיישמת את ממשק תכנות היישומים של מייקרוסופט "XNA 4".

**בינה מלאכותית**

בינה מלאכותית היא תחום רחב במדעי המחשב שעוסק ביכולת לתכנת מחשבים לפעול באופן שמאפיין בני אדם. בינה מלאכותית באה לידי ביטוי בחיינו בתחומים רבים כמו לדוגמה: זיהוי קול, זיהוי פנים, רובוטיקה, זיהוי כתב יד וגם משחק נגד המחשב.

**למידת מכונה**

במשחק שלי החלטתי להתמקד בלמידת מכונה - תת תחום בבינה מלאכותית שמשיק גם לתחומי האופטימיזציה והסטטיסטיקה. המטרה היא לגרום למחשב, על סמך צבירת ניסיון, לקבל החלטות יותר נכונות במשחק.

**Reinforcement learning**

בגישה זאת הסוכן של הבינה המלאכותית ( הידוע גם כ"בוט") לומד באמצעות ניסוי וטעיה על פי המשוב (הפידבק או reward ) אחרי כל שלב. הסוכן לומד להבין את הסביבה ולהתקדם בה על פי ה- reward שהוא מקבל. הוא תמיד ירצה להתאים התנהגות כדי לקבל reward מקסימלי.

**Q-learning**

מטרת האלגוריתם היא ללמוד מדיניות שמבוססת על צבירת ניסיון, שתקבע מה הפעולה המועדפת במצב נתון. האות Q מגיעה מהמילה " quality"- האיכות של כל פעולה במצב נתון. כדי לממש את האלגוריתם צריך טבלה שנקראת " Q-Table" שמכילה ערכים מספריים (Q values) שמציינים עד כמה משתלם לבצע פעולה מסוימת במצב הנתון. מספרים אלו מעודכנים בהיוודע התוצאות של כל פעולה. כדי ליצור את הטבלה בשלב הראשוני ("exploration") יש שימוש במספרים רנדומליים.

**היוריסטיקה**

היוריסטיקה (מהמילה אאוריקה- מצאתי ביוונית) היא כלל חשיבה המבוסס על אינטואיציה או הגיון פשוט והיא מציעה דרך קלה ומהירה לקבלת החלטות על חשבון הדיוק. החוקרים המרכזיים של הטכניקות ההיוריסטיות הם הפסיכולוגים עמוס טברסקי וזוכה הנובל דניאל כהנמן. הם היו הראשונים להדגים כיצד המוח האנושי משתמש בטכניקות היוריסטיות במהלך קבלת החלטות במצבים של חוסר וודאות. טכניקות אלה הם אמנם אפקטיביות אך יכולות להוביל גם לשגיאות.

**יישום האלגוריתם במשחק שלי**

המימוש שלי מתבסס על Q-learning אך יש בו מספר הבדלים בהשוואה לאלגוריתם המקורי:

* התוכנה לא מתחילה עם מספרים רנדומליים לגמרי אלא ממספרים שמבוססים על ניסיון קודם (priors).
* הפידבק על הפעולה מתקבל בסוף המשחק ולא אחרי כל מהלך כיוון שבמשחק הדוראק קשה לראות מיד האם המהלך היה טוב או לא.
* מכיוון שלא מצאתי פונקציות מוכנות של NET. שמעדכנות Q values בניתי פונקציה היוריסטית שמבצעת חישובים אלו.

**הסבר על הפונקציות והמחלקות**

**המחלקה Card**

Card הוא אובייקט שמייצג קלף.

השדות של Card:

* suit - מה סוג הקלף- האם הוא עלה, לב, תלתן או יהלום.
* rank - הערך של הקלף
* lengthOfSeries- נועד רק למקרים בהם המחשב מגן או תוקף ומציין את גודל הסדרה- כמה פעמים יש למחשב קלף מאותו ערך.
* Rotate- ערך הסיבוב של הקלף
* Location- המיקום של הקלף על המסך
* destX ו- destYשיעורי ה-X וה-Y של היעד של הקלף- המקום אליו הוא זז.
* isBlack- האם פני הקלף הם כלפי מטה
* isMoving- האם הקלפים עדיין זזים
* trumpFlag- האם הקלף הוא קוזיר
* בנוסף, יש שדות של אובייקטים שנועדו לציור וטעינה של התמונות של הקלפים.

הפונקציות של Card:

* Load- טוענת את התמונות של הקלפים.
* DrawCard- מציירת את הקלף על פי הצד שנקבע.
* Move- מזיזה את הקלף לכיוון היעד שנקבע לו. בנוסף היא מעדכנת אם הקלף הנתון הוא קוזיר.

**המחלקה** **Desk**

Desk הוא שולחן המשחק ועליו מתרחשים כל האירועים במשחק. נמצאות בו הפונקציות שממשות את התרחישים של המשחק- שחקן שלא יכול להגן ולוקח, שחקן שלוקח מהקופה ועוד.

השדות שנמצאים במחלקה:

* deck – רשימה של הקלפים שנמצאים בקופה.
* cardsRanksPossiblities– רשימה של הערכים של הקלפים שנמצאים בהתקפה ובהגנה (ערכי הקלפים שהשחקן או המחשב יכול להוסיף בהתקפה).
* attackPos- רשימה סטטית של המיקומים של הקלפים שיהיו בהתקפה. הקלף הראשון יהיה במיקום הראשון שנמצא ברשימה זאת, השני במיקום השני וכך הלאה. הקלפים יהיו בהתקפה ובהגנה במיקום קבוע שמאותחל מראש.
* Bita- רשימה של הקלפים שנמצאים בביטה.
* attackCards – רשימה של הקלפים שנמצאים בהתקפה.
* defendCards - רשימה של הקלפים שנמצאים בהגנה.
* Cozir – משתנה סטטי שקובע מה הקוזיר במשחק
* gameStarted- משתנה בוליאני שקובע אם המשחק התחיל
* hasFinishedMoving- משתנה בוליאני שקובע אם כל הקלפים סיימו לנוע על המסך
* botPlayer- אובייקט מסוג BotPlayer שמייצג את המחשב, אני אפרט על מחלקת BotPlayer בהמשך
* humanPlayer- אובייקט מסוג Player שמייצג את השחקן, אני אפרט על מחלקת Player בהמשך

הפונקציות במחלקה:

* ShuffleList- פונקציה נועדה לערבב את חבילת הקלפים באופן רנדומלי.[[1]](#footnote-2)
* initAttackPos- מגדירה את הרשימה attackPos
* ReInit- מאפסת את המשחק לאחר שהשחקן לחץ על כפתור try again. היא מעבירה את כל הקלפים לקופה ומגדירה את המיקום והיעד שלהם לפי המיקום בו תהיה הקופה. בנוסף היא קוראת לפונקציה DealCards שתחלק את הקלפים לשני השחקנים.
* InitDeck- יוצרת את כל הקלפים שקיימים במשחק ושמה אותם בקופה. קוראת לפונקציה ShuffleList כדי לערבב את הקלפים וקובעת את המיקומים של הקלפים.
* DealCards- קובעת מה יהיה הקוזיר ומחלקת שישה קלפים לכל שחקן. היא מעבירה קלפים מהקופה לרשימה המתאימה של השחקן ומגדירה את יעדם החדש של הקלפים.
* Init- קוראת לפונקציות InitDeck, initAttackPos ו- .chooseTactics
* DoesHumanPlayerBegin- בודקת איזה שחקן מתחיל. היא מוצאת את הקוזיר המינימלי של השחקן ושל המחשב ומשווה ביניהם. אם לשני השחקנים אין קוזיר הפונקציה פועלת לפי חוקי המשחק ועוברת על ערכי הקלפים האפשריים החל משש ועד אס, אם לשני השחקנים יש או לשניהם אין קלף עם אותו הערך היא עוברת לערך הבא ואם רק לשחקן אחד יש אותו אז הוא מתחיל.
* IsCardFound- מקבלת ערך של קלף ושחקן. הפונקציה מחפשת אם הערך שהיא קיבלה קיים בקלפים של השחקן המבוקש.
* Load- קוראת לפונקציה Load של כל הקלפים ולאחר מכן קוראת ל- DealCards.
* Draw- אם המשחק כבר התחיל היא קוראת לפונקציה DrawCard של כל הקלפים שקיימים.
* CheckForEmptySpaces- ברגע שקלף נשלח להתקפה או להגנה יש צורך להזיז את שאר הקלפים ימינה כדי שאם תהיה ביטה יהיה קל לקחת מהקופה, כי אז הקלפים היווספו לסוף הרשימה ולא יהיו מקומות ריקים. הפונקציה עוברת על הקלפים של המחשב ואם המרחק בין שני קלפים סמוכים גדול מהמרחק שנקבע היא מזיזה את השמאלי לעבר הימני. לאחר מכן, אם הקלף הראשון של המחשב לא נמצא במקום שנקבע לו, הפונקציה מזיזה אותו ואת שאר הקלפים של המחשב ימינה למיקום שנקבע. לאחר מכן היא מבצעת דבר זהה עם הקלפים של השחקן.
* CheckFinishedMoving- פונקציה שמחזירה ערך בוליאני שמצביע על האם כל הקלפים סיימו לנוע.
* TakeFromDeck- מקבלת שחקן ומאפיינים ייחודיים שלו כמו מיקום קלפיו בציר Y והאם הקלפים צריכים להיות עם הפנים כלפי מטה או כלפי מעלה. היא מעבירה לשחקן קלפים מהקופה עד שיהיו לו 6 קלפים.
* DoBita- הפונקציה אחראית על הביטה. היא מעבירה את הקלפים שנמצאים בהגנה ובהתקפה אל רשימת הביטה ומזיזה אותם ליעד המתאים בקצה הDesk. הפונקציה מקבלת משתנה בוליאני שקובע מי הגן ומי תקף או במילים אחרות מי לוקח מהקופה ראשון ומי לוקח שני. הפונקציה קוראת ל- TakeFromDeck פעמיים, אך מעבירה לפונקציה פרמטריים שונים כך שסדר הלקיחה מהקופה יתאים לערך הפרמטר הבוליאני שהיא קיבלה. בסוף היא מוחקת את המספרים שהיו ב- cardsRanksPossiblities.
* DoTake - הפונקציה מקבלת שחקן ומעבירה לו את הקלפים שנמצאים בהגנה ובהתקפה וכל קלף היא שמה בסוף רשימת הקלפים שלו. לאחר מכן היא מאפסת את cardsRanksPossiblities.
* Update- קוראים לפונקציה זו 60 פעמים בשנייה והיא בודקת בכל פעם מה השתנה ומזיזה את הקלפים. לאחר מכן, היא קוראת לפונקציה checkForEmptySpaces ומעדכנת את hasFinishedMoving בעזרת checkFinishedMoving. לאחר מכן, הפונקציה מזיזה את כל הקלפים (אם הם צריכים לזוז).

**מחלקת BotPlayer**

המחלקה יורשת מ- Playerולכן היא יכולה לקרוא לפונקציות של Player וגם לה יש שדה cardsInHand.

פונקציות במחלקה:

* MakeDefenseMove - אחראית על בחירת המהלך של המחשב בהגנה. אם נגמרו הקלפים בקופה והשינוי הטקטיקה טרם נעשה היא קוראת לפונקציה ChangeTactic שעושה זאת. בדוראק, בסוף המשחק כבר ממש לא הגיוני לשמור קלפים לאחר כך ולכן יחס העדיפויות בין הטקטיקות חייב להשתנות. היא קוראת לפונקציה DefendPosibilities שמוצאת את כל הקלפים איתם ההמחשב יכול להגן ואחר מכן ל- narrowDefendPosibilities שמצמצמת את האפשרויות על פי הטקטיקה. היא מחזירה enum מתאים במטרה להודיע לשחקן שהמחשב לא יכול להגן. ואם יש אפשרויות היא קוראת לפונקציה sortBySeries של מחלקת Strategy שעליה אפרט בהמשך. היא קוראת לפונקציה evaluateCards של המחלקה Strategy שגם עליה אפרט בהמשך. הקלף שנמצא באינדקס של המספר המקסימלי ברשימה שהוחזרה הוא הקלף שהכי כדאי לשים ואכן הפונקציה שמה בהגנה בעזרת הפונקציה MoveCardToDefense . הפונקציה מחזירה enum כדי שהפונקציה שקראה לה תדע איך לשנות את מצב המשחק.
* MakeAttackMove - אחראית על בחירת המהלך של במחשב בהתקפה. אם נגמרו הקלפים בקופה והשינוי הטקטיקה טרם נעשה היא קוראת לפונקציה ChangeTactic. לאחר מכן היא שומרת את האפשריות המתאימות של המחשב להתקפה בעזרת קריאה לפונקציה narrowAttackPossibilities. אם הדגל addAll דלוק אז היא מוסיפה את כל הקלפים שנמצאים ברשימת האפשרויות להתקפה בעזרת קריאה לפונקציה MoveCardToAttack. במקרים האחרים היא קוראת לפונקציה sortBySeries של מחלקת Strategy שעליה אפרט בהמשך. לאחר מכן אם מדובר בהוספה אז היא מוסיפה קלף להתקפה בין האפשרויות באמצעות MoveCardToAttack. אם לא מדובר בהוספה הוא משתמש ב evaluateCards ובוחר את הקלף שנמצא באינדקס עם היעילות המקסימלית ותוקף איתו. . הפונקציה מחזירה enum כדי שהפונקציה שקראה לה תדע איך לשנות את מצב המשחק.

**מחלקת Player**

המחלקה מכילה את רשימת הקלפים של השחקן- cardsInHand ועוד שתי פונקציות:

* MoveCardToAttack- מקבלת את האינדקס של הקלף ואובייקט מסוג Desk כדי שאפשר יהיה לגשת למידע כמו ערכי הקלפים שאפשר להוסיף. אם המשחק לא נגמר והאינדקס הוא לא 99- מספר שמסמן שלא נבחר אף קלף הוא יבדוק אם אפשר להוסיף את הקלף עם אינדקס הנתון. אם זאת התקפה ראשונה ולא הוספה (cardsRanksPossiblities יהיה ריק) הוא מוסיף את הערך של הקלף ל- cardsRanksPossiblities. לאחר מכן הוא בודק אם ערך הקלף נמצא ב- cardsRanksPossiblities ואם כן הוא מוסיף אותו להתקפה.
* MoveCardToDefense- מקבלת את האינדקס של הקלף שנבחר ואובייקט מסוג Desk. אם המשחק לא נגמר והאינדקס הוא לא 99 היא מפעילה את הפונקציה DefendPosibilities כדי לבדוק אם השחקן באמת יכול להגן עם הקלף שהוא בחר. רק אם הקלף שהשחקן בחר מופיע ברשימת הקלפים שהחזירה DefendPosibilities אז הפונקציה מעבירה את הקלף להגנה.

**מחלקת Strategy**

זוהי מחלקה סטטית שקובעת את הטקטיקה של המחשב במשחק ואת המשקל של כל קלף, כשהמחשב צריך לבחור מה לעשות. היא משתמשת ב- Q table שנמצאת בקובץ CSV ומממשת את האלגוריתם לבינה מלאכותית. כל שורה מייצגת מצב אחר במשחק וכל עמודה מייצגת את המשקל של פרמטרים שונים כמו פעולות אפשריות שאפשר לבצע במהלך המשחק. בטבלה במשחק זה יש שבע עמודות שמייצגות את שבע הטקטיקות שקיימות, עמודה שמייצגת את חשיבות המשקל של הקלף (שנקבע יחסית לקלפים שנמצאים עוד במשחק), עמודה שמייצגת את חשיבות העובדה שהקלף הוא חלק מסדרה ועמודה נוספת מכילה את כמות הפעמים שהקובץ עודכן על ידי התוכנה. כשהמחשב לא מספיק מנוסה הוא משתמש גם במספרים רנדומליים כדי לפצות על חוסר הניסיון ולהמשיך לחקור האם הפעולות יעילות או לא.

ה-enums במחלקה:

* AttackTactics- מכיל את שלוש הטקטיקות האפשריות בהתקפה.
* DefenseTactics- מכיל את ארבעת הטקטיקות האפשריות בהגנה.

המשתנים במחלקה:

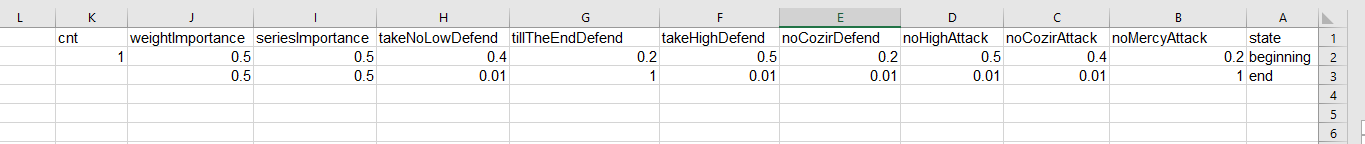
* lengthTableLine - מייצגת את כמות העמודות בטבלה.
* seriesImportance- מספר עשרוני שמייצג את חשיבות העובדה שהקלף הוא חלק מסדרה. הכוונה בסדרה כשלמחשב יש כמה קלפים עם אותו הערך.
* weightImportance- מספר עשרוני שמייצג את חשיבות המשקל של הקלף.
* attackTacticBeginning- משתנה מסוג AttackTactics שמייצג את הטקטיקה להתקפה שנבחרה בהתחלה.
* defenseTacticBeginning- משתנה מסוג DefenseTactics שמייצג את הטקטיקה להגנה שנבחרה בהתחלה.
* attackTacticEnd- משתנה מסוג AttackTactics שמייצג את הטקטיקה להתקפה שנבחרה בסוף המשחק כשהטקטיקה השתנתה.
* defenseTacticEnd- משתנה מסוג DefenseTactics שמייצג את הטקטיקה להגנה שנבחרה בסוף המשחק כשהטקטיקה השתנתה.
* numOfTimesWritten- שומר את כמות הפעמים שעודכן הקובץ. הכמות רשומה בטבלה ומשתנה זה נקרא מטבלה בתחילת המשחק ונרשם אליה בסוף משחק.
* attackTacticsBeginningQvalues- רשימה של מספרים עשרוניים כל איבר מייצג את המשקל של טקטיקת התקפה בתחילת המשחק כפי שרשום בטבלה. אם המחשב לא מיומן מספיק (numOfTimesWritten קטן) כל איבר בסדרה מוכפל במספר רנדומלי. הטקטיקה הנבחרת היא זאת שמיוצגת על ידי האיבר המקסימלי ברשימה זאת.
* attackTacticsEndQvalues- רשימה של מספרים עשרוניים כל איבר מייצג את המשקל של טקטיקת התקפה בסוף המשחק כפי שרשום בטבלה. אם המחשב לא מיומן מספיק כל איבר בסדרה מוכפל במספר רנדומלי.
* defenseTacticsBeginningQvalues- רשימה של מספרים עשרוניים כל איבר מייצג את המשקל של טקטיקת הגנה בתחילת המשחק כפי שרשום בטבלה. אם המחשב לא מיומן מספיק כל איבר בסדרה מוכפל במספר רנדומלי.
* defenseTacticsEndQvalues- רשימה של מספרים עשרוניים כל איבר מייצג את המשקל של טקטיקת הגנה בסוף המשחק כפי שרשום בטבלה. אם המחשב לא מיומן מספיק כל איבר בסדרה מוכפל במספר רנדומלי.
* Random- נועד כדי להגריל מספרים רנדומליים.

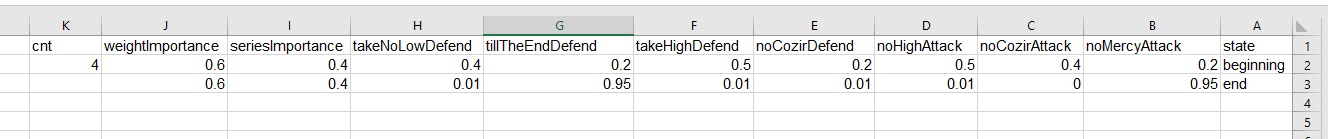
הטקטיקות האפשריות:

* noMercyAttack- תקיפה ללא רחמים- מתחיל בקלף החלש ביותר אך כשצריך להוסיף הוא מוסיף את כל הקלפים שהוא יכול ולא שומר קלף לאחר כך גם אם זה קוזיר.
* noCozirAttack- תקיפה בלי קוזירים- תוקף עם קוזיר רק אם אין ברירה כמו לדוגמה כשכל הקלפים שלו קוזיר והוא חייב להתחיל את התקפה (בניגוד להוספה).
* noHighAttack- תקיפה בלי קלפים גבוהים- תוקף עם קלפים גבוהים מנסיך רק אם כל הקלפים שלו גבוהים והוא חייב להתחיל את התקפה (בניגוד להוספה).
* noCozirDefend- הגנה בלי קוזירים- אם אין לו אפשרות להגן כך אז הוא לוקח.
* takeHighDefend- אם השחקן מביא למחשב קלף גבוה מנסיך המחשב לוקח אוטומטית
* tillTheEndDefend- המחשב מגן עם כל מה שיש לו ורק אם נגמרות לו כל האפשרויות הוא לוקח.
* takeNoLowDefend- אם יש בהתקפה קלפים נמוכים מתשע הוא יגן עליהם עם כל מה שיש לו ורק אם נגמרות לו כל האפשרויות הוא ייקח. אם יש גבוהים (בלי נמוכים) אז הוא יקח. טקטיקה זאת היא בעצם שילוב של tillTheEndDefend ו- takeHighDefend. הוא בוחר באיזו מהם להשתמש על פי מצב המשחק.

הפונקציות שנמצאות במחלקה:

* MinCard- מוצאת את הקלף הנמוך ביותר אצל המחשב. היא שומרת את הקלף הראשון ברשימת הקלפים ורצה על שאר הקלפים. אם הקלף הנוכחי הוא לא קוזיר והקלף שהפונקציה שמרה הוא קוזיר היא שומרת אותו במקום. אם אחד מהקלפים קוזיר היא שומרת את זה שלא קוזיר, ואם שניהם לא קוזירים היא שומרת את הקלף בעל הערך הנמוך ביותר. היא מחזירה את הקלף ששמור אצלה בסוף.
* chooseTactics- קוראים לפונקציה בתחילת המשחק והיא בוחרת את הטקטיקה. הפונקציה קוראת את מה שרשום בקובץ ושומרת את כל המילים שמופיעות בשורה השנייה. מכיוון שבקובץ CSV בין כל איבר בשורה יש פסיק ניתן להשתמש בפונקציה Split כדי לשמור את כל הערכים בשורה במערך אחד. לאחר מכן, היא עוברת על הערכים של השורה של מצב ההתחלה ומכניסה בהתאם ערכים ל- attackTacticBeginning ו- defenseTacticBeginning. לאחר מכן הפונקציה מעדכנת את numOfTimesWritten בהתאם לנתון בטבלה. אם הוא קטן מ-100 המחשב לא מספיק מנוסה וצריך לעבור על attackTacticBeginning ו- defenseTacticBeginning ולהכפיל את כל הערכים במספר רנדומלי. הפונקציה בודקת מהו האינדקס עם המספר המקסימלי בשני המערכים וקובעת את טקטיקת המשחק לטקטיקה המתאימה לאינדקס הזה. לאחר מכן הפונקציה מעדכנת את ערכי seriesImportance ו- weightImportance על פי הרשום בטבלה ומכפילה אותם במספר רנדומלי אם המחשב לא מספיק מנוסה.
* changeTactic- הפונקציה משנה את הטקטיקה לקראת סוף המשחק. לקראת סוף המשחק המצב שונה מאוד בהשוואה להתחלה וכבר אין טעם לשמור קלפים לאחר כך. אם אתחלתי את הסטייט של ההתחלה עם מספרים שרירותיים אך מבוססים על ידע קודם שלי, בסטייט של סוף המשחק יהיה אחרת. הטקטיקות ההגיוניות ביותר לסוף משחק הן noMercyAttack ו- tillTheEndDefend ולכן בשורה של סוף המשחק בטבלה אתחלתי טקטיקות אלו כ-1 ואת השאר אתחלתי כ-0.01. הפונקציה עצמה פועלת באופן זהה ל- chooseTactics.
* CalcRewards- פונקציה היוריסטית שמחזירה מחרוזת של הטקסט שצריך להיכתב בטבלה ((Q-table שבקובץ על מנת לעדכנו בהתאם לתוצאת המשחק שהיא מקבלת. הפונקציה צריכה להגדיל או להקטין את הערכים (Q-VALUES) שמייצגים את הפעולות בהן השתמשנו אך אסור שערכים אלו יגיעו ל0 או יהפכו לשליליים. הפונקציה נעזרת במספר קבוע וכשהיא רוצה לשנות ערך בטבלה היא מכפילה את הקבוע בערך הקיים ורק אז מוסיפה/מורידה אותו מהערך הקיים. וכך הערך ישתנה אך לא יגיע לעולם לאפס. תחילה הפונקציה מגדירה שני קבועים deltaState1Propotion ו- deltaState2Propotion. אם המחשב הפסיד אז הם יהיו שליליים ואם המחשב ניצח- חיוביים. הערך המוחלט של הקבוע השני יהיה תמיד קטן מהראשון כי תמיד נרצה שהערכים בסטייט של סוף המשחק השתנו פחות מכיוון שבסוף יש עדיפות ברורה של טקטיקה אחת על פני השנייה. הפונקציה מוצאת את ה- Q-Valueשל הטקטיקה בה היא השתמשה, מגדירה את השינוי כמכפלת הפרופורציה המתאימה ב- Q-Value ומוסיפה את השינוי ל- Q-Value. עם seriesImportance ו- weightImportance הפונקציה פועלת באופן דומה אך כשהיא מקטינה אחד מהם היא מעלה את האחר ולהפך. כדי לעדכן את הטבלה הפונקציה יוצרת מערך שמכיל את כל הערכים של הסטייט ההתחלתי ומערך נוסף שמכיל את כל הערכים של הסטייט של סוף המשחק. כשהיא משנה את הערכים היא משנה את הערכים במערך ולאחר מכן היא עוברת על כל מערך מוסיפה כל איבר למחרוזת strToWrite ומוסיפה אחריו פסיק, בין שורה לשורה היא מוסיפה "n\". בסיום היא מחזירה את המחרוזת.

דוגמה למצב הטבלה ההתחלתי (שאתחלתי ידנית):

דוגמה למצב הטבלה אחרי כמה ניסיונות (המצב ההתחלתי לא היה כמו בדוגמה):

* sortBySeries- מעדכנת לכל קלף של המחשב את כמות הפעמים שקלף עם אותו ערך נמצא אצל השחקן הוירטואלי- המחשב. היא מקבלת משתנה בוליאני שמצביע על האם מדובר בהתקפה או הגנה ואובייקט מסוג Desk כדי לגשת למידע שהיא צריכה. היא יוצרת רשימה דו ממדית של קלפים כך שבכל שורה יהיו קלפים אם אותו ערך. קודם כל הפונקציה יוצרת רשימת קלפים זמנית חדשה שאליה היא מוסיפה את הקלפים של המחשב ואם המחשב מגן אז היא מוסיפה גם את הקלפים שבהגנה. כל עוד רשימת הקלפים לא התרוקנה היא מוסיפה שורה חדשה לרשימה הדו ממדית, מכניסה אליה את האיבר בראש רשימת הקלפים, מוחקת אותו מהרשימת הזמנית ומחפשת עוד קלפים ברשימה הזמנית שיש להם אותו ערך וכשהיא מוצאת היא מוסיפה אותם לאותה השורה של הרשימה הדו ממדית ומוחקת אותם מהרשימה הזמנית. כשהיא סיימה את הלולאה היא עוברת על כל הרשימה הדו ממדית ומציבה ב-lengthOfSeries של כל קלף את אורך השורה בה הוא נמצא.
* isFirstStronger- מקבלת שני קלפים ומחזירה ערך בוליאני שמראה אם הקלף הראשון יכול לנצח את הקלף השני. הפונקציה בודקת אם הקלף הראשון הוא קוזיר והשני לא ואם כן היא מחזירה true. אחר כך היא בודקת אם שני הקלפים הם מאותו סוג וערך הקלף הראשון גדול מערך הקלף השני, אם כן היא מחזירה true. לאחר מכן היא בודקת אם הראשון קוזיר ואם השני נמוך מהראשון (אם היא הגיעה לתנאי הזה והוא התקיים ברור שגם הקלף השני הוא קוזיר), אם כן היא מחזירה true. לכל מקרה אחר היא מחזירה false.
* calcWeight- מקבלת קלף ואובייקט של Desk ומחזירה את המשקל של הקלף- עד כמה הוא חזק ביחס לקלפים האחרים במשחק. היא עוברת על הקלפים שעדיין במשחק (הקופה והקלפים של היריב) והיא קוראת לפונקציה isFirstStronger ובכל פעם שהיא מוצאת קלף שחלש מהקלף שהיא קיבלה היא מגדילה את המונה באחד. לבסוף היא מחזירה את המונה.
* EvaluateCards- מקבלת רשימת קלפים ואובייקט של Desk ומחזירה רשימה של מספרים עשרוניים שכל איבר בה מצביע על היעילות של הקלף שנמצא באותו האינדקס ברשימה שהתקבלה. מכיוון שככל שהקלף יותר חלש כך עדיף להיפטר ממנו יותר מהר יש יחס הפוך בין יעילות המהלך למשקל הקלף. הפונקציה עוברת על הרשימה שהיא קיבלה ומוסיפה בכל פעם לרשימה החדשה שהיא יצרה מספר על פי נוסחה קבועה: המשתנה weightImportance חלקי המשקל של הקלף שמחושב באמצעות הפונקציה calcWeight ועוד seriesImportance כפול ה- lengthOfSeries של הקלף.
* narrowDefendPosibilities- מקבלת את רשימת הקלפים האפשריים ואובייקט של Desk ומצמצמת את אפשרויות ההגנה על פי הטקטיקה שנקבעה. אם noCozirDefend במצב פעיל אז הפונקציה מוחקת את כל הקוזירים ברשימה שהתקבלה. לאחר מכן הוא עובר על הרשימה ובודק אם יש בה קלפים נמוכים או קלפים גבוהים. אם takeNoLowDefend פעיל ויש לפחות קלף נמוך אחד בהתקפה, אז הפונקציה הופכת את tillTheEndDefend לאמת, ואם הטקטיקה היא takeNoLowDefend ואין קלפים נמוכים בהתקפה, היא הופכת את takeHighDefend לאמת. כאמור, takeNoLowDefend היא שילוב של שתי הטקטיקות. לאחר מכן הפונקציה בודקת אם takeHighDefend הוא אמת, ואם כן ואם יש גם קלפים גבוהים שמשתתפים בהתקפה, הפונקציה מחזירה רשימה ריקה- משמע אין אפשרויות וצריך לקחת.
* narrowAttackPossibilities- הפונקציה מקבלת אובייקט מסוג Desk ורשימת אשרויות ומחזירה רשימת אפשרויות מעודכנת להתקפה שמתאימות לטקטיקה הפעילה. אם הטקטיקה היא או noCozirAttack או noHighAttack, היא עוברת על הרשימה ומוחקת את הקוזירים. אם הטקטיקה היא noHighAttack , הפונקציה עוברת על כל הרשימה ומוחקת את כל הקלפים שגבוהים מנסיך. אם לא נשארו אפשרויות וחייבים לתקוף (במקרים ש-cardsRanksPossiblities ריק אז לא מדובר בהוספה אלא במהלך הראשון בהתקפה), קוראים לפונקציה MinCard כדי להכניס לרשימת האפשרויות את הקלף הכי נמוך. אם מדובר בהוספה אז מוחקים מרשימת האפשרויות את כל הקלפים שהערך שלהם לא נמצא בהתקפה או בהגנה. לבסוף מחזירים את רשימת האפשריות שנותרה.

**מחלקת Game1 - המחלקה הראשית**

המחלקה מגיעה יחד עם MONOGAME וכוללת פונקציות שהממשק של MONOGAME מפעיל אוטומטית.

המחלקה משתמשים בכמה enums:

* ActivePlayerState - קובע של מי התור להתקיף/להגן
* GameState- קובע מה מצב המשחק ומכיל את האפשריות הבאות:
* addAllYouCan - במצב בו השחקן הכריז שהוא לוקח, הדגל הזה נדלק והשחקן הוירטואלי (המחשב) יודע שהוא יכול להוסיף את כל הקלפים שיש לו ומתאימים לטקטיקה בה הוא משתמש.
* waitingForBita- האם מחכים לכך שהשחקן ילחץ על הכפתור bita.
* waitingForTake- האם מחכים לכך שהשחקן ילחץ על take.
* takeBot - האם המחשב צריך לקחת.
* waitingForAdd- אחרי שהמחשב לוקח, השחקן יכול להוסיף עוד קלפים. המשתנה קובע אם מחכים לכך שהשחקן יכריז שאין לו יותר מה להוסיף.
* botHasNothingToAdd - בהמשך ל- addAllYouCan כשהמחשב מוסיף את כל הקלפים המתאימים מבחינתו, הוא מדליק את הדגל הזה שמסמן שאין לו יותר מה להוסיף.
* Endgame- האם המשחק נגמר. כשהמשחק נגמר אז יהיה תופיע תמונה של ניצחון או הפסד ויופיעו כפתורים של try again ו-exit.
* Welcome- האם צריך להציג את מסך הפתיחה
* gameRules- האם צריך להציג את המסך של חוקי המשחק
* howToPlay- האם צריך להציג את המסך של איך משחקים
* intermediate - אף מצב מהמצבים הנ"ל אינו מתקיים
* Winner- קובע מי ניצח

המשתנים במחלקה:

* המיקומים של כל הכפתור, ההודעות והתמונות שמוצגות על המסך.
* התבניות (Texture) של התמונות שמוצגות על המסך.
* mousePos- המיקום של העכבר
* Desk- אובייקט בסוג Desk.
* gameStarted- האם המשחק התחיל- אם לא תציג מסך פתיחה.
* hasFileUpdated- האם כבר עידכנו את קובץ ה-csv. צריך לעדכן רק פעם אחת במשחק.
* hasTacticChanged- האם הטקטיקה השתנתה. צריך לשנות אותה רק פעם אחת- ברגע שנגמרים הקלפים.
* needToNullify- האם צריך לאפס את המשחק. הופך לחיובי אחרי לחיצה על הכפתור try again.
* gameOn- האם המשחק נמצא בעיצומו או במילים אחרות האם יצאנו ממסך הפתיחה או ממסך הסיום.
* switchTurnToComputer - במצבים בהם התור עובר למחשב וגם המחשב צריך לקבל קלפים מהקופה, דגל זה נדלק וגורם לכך שהמחשב מחכה עד שכל הקלפים יגיעו ליעדם ורק אז מתחיל את התור.
* switchTurnToOpponent – משתמשים רק כשמצב אימון דלוק. במצבים בהם התור עובר לבוט שמייצג את השחקן כשהוא עוד צריך לקחת מהקופה דגל זה נדלק והמחשב מחכה.

הפונקציות במחלקה:

* Initialize- מופעלת ראשונה ומאחלת את המשחק. היא מעדכנת את אורך ורוחב המסך, מאתחלת את המשתנים וקוראת לפונקציה Init של Desk.
* LoadContent- מופעלת אחרי Initialize, טוענת את כל התמונות לזיכרון וקוראת לפונקציה Load של Desk.
* FindIdxClicked- מוצאת על איזה קלף של השחקן נמצא העכבר ומחזירה את האינדקס של קלף זה. היא עוברת על הקלפים של השחקן ומוצאת אם העכבר נמצא בתחום של אחד הקלפים.
* UpdateResults- מקבלת enum שמציין מי ניצח ומעדכנת את תוצאת המשחק במשנים המתאים ומעדכנת את הטבלה בעזרת הפעולה CalcRewards.
* DefendPosibilities- מקבלת קלף ושחקן ומחזירה רשימה עם כל הקלפים שאיתם השחקן יכול להגן על הקלף. אם הקלף הוא קוזיר אז הפונקציה מחפשת בקלפים של המחשב קוזיר שהוא יותר גבוה. אם לא היא מחפשת קלף שהוא או קוזיר או קלף מאותה צורה אך יותר גבוה. אם היא לא מוצאת כלום היא מחזירה רשימה ריקה.
* trainingPlay – מבצעת את המהלכים של שני השחקנים ונקראת רק כשמצב אימון דלוק. היא בודקת של מי התור וקוראת בהתאם לפונקציה המתאימה של BotPlayer. בהתאם ל-enum שהפונקציה החזירה היא פועלת בהתאם ומשנה את מצב המשחק.
* checkClick – מעדכנת את idxClickedCard בעזרת FindIdxClicked ובודקת אם העכבר נמצא על אחד הכפתורים:
* אם מוצג המסך welcome ונלחץ הכפתור start שנה את המשתנים המתאימים. אם נלחץ כפתור gameRules או howToPlay תציג את המסך המתאים.
* אם מוצגים כעת המסכים gameRules או howToPlay ונלחץ כפתור Close תחזור למסך welcome.
* אם המשחק נגמר והקלפים סיימו לנוע: אם נלחץ כפתור exit, תצא מהמשחק לגמרי, ואם נלחץ כפתור try again, אז תדליק את .needToNullify
* אם מחכים לכך שהמשתמש ילחץ על ביטה (על פי הדגל waitingForBita), הקלפים סיימו לנוע, המשחק עוד לא נגמר ונלחץ כפתור ביטה תקרא לפונקציה DoBita ותשנה בהתאם את המצב המשחק.
* אם מחכים לכך שהמשתמש ילחץ על הכפתור שאומר שאין לו מה להוסיף (על פי הדגל waitingForAdd), הקלפים סיימו לנוע, המשחק עוד לא נגמר והכפתור נלחץ, תשנה בהתאם את מצב המשחק.
* אם מחכים לכך שהמשתמש ילחץ על הכפתור take (על פי הדגל waitingForTake), הקלפים סיימו לנוע, המשחק עוד לא נגמר והכפתור נלחץ, תשנה בהתאם את מצב המשחק– תעביר את התור למחשב כדי שיבדוק אם יש לו מה להוסיף.

חשוב לציין שאם מצב אימון דלוק (והמשחק התחיל) היא מתעלמת מהלחיצות על הכפתורים.

* regularPlayBot – מבצעת את המהלכים של הבוט ונקראת רק כשמצב אימון כבוי. היא בודקת של מי התור וקוראת בהתאם לפונקציה המתאימה של BotPlayer. בהתאם ל-enum שהפונקציה החזירה היא פועלת בהתאם ומשנה את מצב המשחק.
* Update- מופעלת 60 פעמים בשנייה ואחראית על עדכון המשחק. תחילה היא מעדכנת את מצב העכבר ואם אחד משחקנים ניצח היא קוראת ל- UpdateResults. אם כפתור שמאלי של העכבר נלחץ, היא קוראת ל- checkClick.

בנוסף היא בודקת אם המחשב או השחקן צריכים לקחת (והיריב כבר הספיק להוסיף את כל הקלפים שרצה) ואם כן היא קוראת לפונקציה שמבצעת זאת ולאחר מכן משנה את מצב המשחק.

לאחר מכן, הפונקציה בודקת אם צריך לאפס את המשחק ואחר כך היא קוראת לפונקציה Update של מחלקת Desk. בהתאם למשתנה הבוליאני training היא קוראת ל- regularPlayBot או ל- trainingPlay. אם צריך להחליף תור והקלפים סיימו לנוע הפונקציה מחליפה תור. במקרה שהקלפים סיימו לנוע אך המשחק עצמו טרם התחיל היא בודקת מי אמור להתחיל – על פי תוצאות המשחק הקודם ואם אין אז על פי DoesHumanPlayerBeginומעדכנת את activePlayerState בהתאם.

* Draw- מציירת על המסך בהתאם לערכים של המשתנים הבוליאניים המתאימים. קודם כל היא קוראת לפונקציה Draw של המחלקה Desk. אם המשחק טרם התחיל, היא מציגה את מסך הפתיחה. אם המשחק נגמר והקלפים סיימו לנוע, היא מציגה את מסך הסיום. אם זה תורו של השחקן לתקוף, המשחקן לא נגמר והקלפים סיימו לנוע, היא מציגה את ההודעה "עכשיו תורך". את הכפתור היא מציירת כשמתקיימים אותם התנאים שמופיעים בפונקציה Update. אם השחקן ניצח או הפסיד, היא מציגה תמונה בהתאם. היא גם מציירת את העכבר בהתאם למיקום הנוכחי שלו כי אחרת לא יראו אותו.

**ביבליוגרפיה**

visual C# kicks

<http://www.vcskicks.com/randomize_array.php>

Reinforcement Learning Tutorial by Juha Kiili from "Towards Data Science" site 2019

<https://towardsdatascience.com/reinforcement-learning-tutorial-part-3-basic-deep-q-learning-186164c3bf4>

How to teach AI to play Games: Deep Reinforcement Learning by Mauro Comi from "Towards Data Science" site 2018

<https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a>

ויקיפדיה

המקל והגזר של למידת מכונה באמצעות חיזוקים מאת ד"ר יעקב רימר אתר דה מארקר.

<https://www.themarker.com/techblogs/misterdata/BLOG-1.6588600>

techopedia site

<https://www.techopedia.com/definition/32055/reinforcement-learning>

1. הפונקציה לקוחה מהאתר visual C# kicks <http://www.vcskicks.com/randomize_array.php> [↑](#footnote-ref-2)