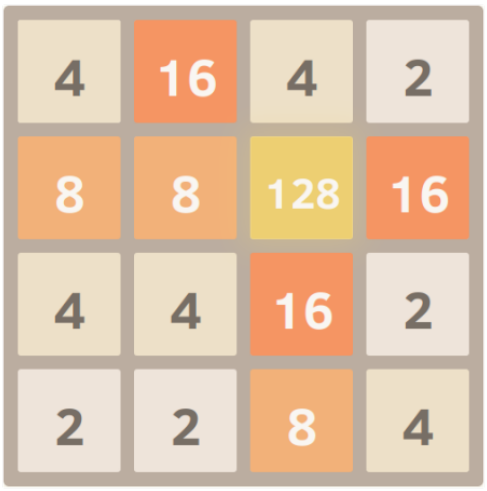
# הבסת המשחק 2048 באמצעות סוכן DQN


בר וייסמן 215528365

שם המנחה: אריאל בר יצחק

מרכז חינוך ליאו באק

# תוכן העניינים


I מבוא 2

1 המשחק 2048 2

2 הפרויקט 2

II רקע תיאורטי 3

1 Reinforcement Learning 3

1.1 למידה מחיזוקים . . . . . 3

1.2 $\varepsilon$-greedy . . . . . 3

2 $Q$-Learning 4

3 Deep $Q$-Learning 4

III מימוש 5

1 סביבה 5

2 מודל 6

3 התקנה ותפעול 9

IV מחקר וניתוח תוצאות 10

1 שיטת אימון #1 10

2 שיטת אימון #2 11

3 שיטת אימון #3 13

V סיכום אישי 15

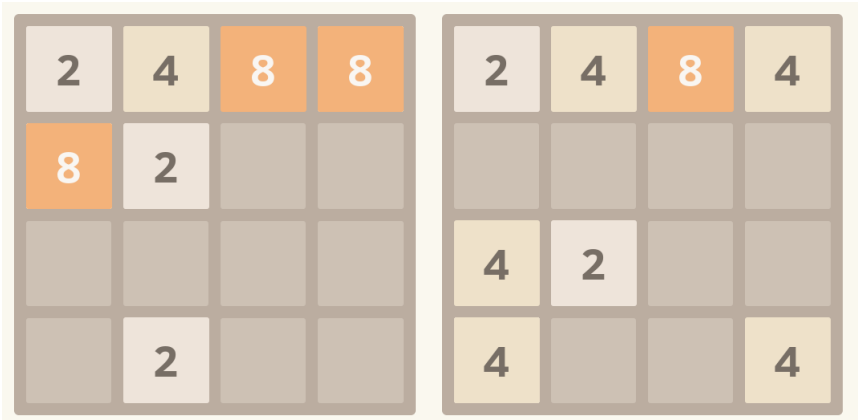
VI ביבליוגרפיה 16

# חלק I
# מבוא

## 1 המשחק 2048

המשחק 2048 (קישור למשחק המקורי בגיטהאב) הוא משחק לשחקן יחיד. המשחק מורכב מלוח בגודל $4 \times 4$, וכל משבצת בלוח מכילה חזקה כלשהי של $2$, או משבצת ריקה. בכל מהלך, השחקן בוחר כיוון (שמאלה, מעלה, ימינה או מטה), ומשבצות הלוח זזות בהתאם. כאשר שתי משבצות בעלות אותו ערך מספרי זזות לאותו הכיוון ללא שום משבצת מלאה ביניהן, נוצרת משבצת חדשה - מיזוג של שתי הקודמות, שמכילה את את סכום ערכיהן. בנוסף, לאחר כל מהלך נוסף נבחרת משבצת ריקה באופן אקראי, ומושם בה ערך בסיסי ($2$ בהסתברות $0.9$, ו-$4$ בהסתברות $0.1$).

המשחק נגמר כאשר לשחקן לא נותרו עוד פעולות אפשריות - כלומר אף פעולה לא תגרום ליצירת משבצת ריקה חדשה, שאליה תיכנס המשבצת הנוספת בתום המהלך. בכל פעם ששתי משבצות $x$ מתמזגות למשבצת $2x$, הניקוד הכולל של הסיבוב גדל ב-$x$. המטרה העיקרית של המספר היא להגיע למשבצת בעל ערך מקסימלי, ובו בזמן למקסם את הניקוד הכולל.




איור 1: דוגמא למצב המשחק. מימין המצב ההתחלתי, ומשמאל המצב הסופי - לאחר תזוזה מעלה. חלק מהמשבצות שערכיהן $4$ (ואין אף משבצת מלאה ביניהן) התמזגו למשבצת חדשה עם ערך $8$. בנוסף, בשורה התחתונה, בעמודה השנייה משמאל, נוספה משבצת עם ערך $2$.

## 2 הפרויקט

מטרת פרויקט זה היא מימוש ואימון סוכן $DQN$ למשחק 2048, כך שיצליח להגיע לתוצאות ולהישגים מוצלחים. קיימים סוכני $DQN$ למשחק, אך התוצאות הקיימות לא מספיקות. על כן, פרויקט זה מתייצב מול הקושי של ה-$DQN$ מול המשחק 2048. נוסו גישות שונות למימוש הסביבה והסוכן: הן בסוג וסיבוכיות המודל עליו מתבססת קבלת ההחלטות של הסוכן, והן שינוי מערכת הפרסים. קבצי הפרויקט, לרבות מחברת jupyter שמכילה את הקוד, נמצאים ב-github[1].

[1] קישור: https://github.com/bowass/2048-DQN-Agent.

# חלק II
# רקע תיאורטי

## 1 Reinforcement Learning

### 1.1 למידה מחיזוקים

למידה מחיזוקים (Reinforcement Learning - RL) הוא תחום בלמידת מכונה שעוסק בנקיטת פעולות שיגררו תגמול כולל מירבי. כלומר - בהינתן מצב מסוים, נרצה שהסוכן (המודל אותו אנו מאמנים) יבצע את הפעולה שתוביל לתגמול כולל גדול ככל האפשר. במשחק $2048$, למשל, נרצה שהסוכן יבחר בכל פעם בפעולה (יבחר מין שמאלה, מעלה, ימינה או מטה) שתמקסם את הניקוד המצטבר של המשחק, תוך כדי התייחסות לטווח הן לטווח הקצר והן לארוך.

ב-reinforcement learning אין מאגר נתונים מוכן מראש: הסוכן אוסף את הנתונים תוך כדי זמן האימון על ידי ביצוע פעולות שונות וזכירת המצבים השונים בהם היה, יחד עם הפעולות שנקט והתגמול שקיבל בהתאם.

### 1.2 $\varepsilon$-greedy

מאחר ולסוכן אין ידע מוקדם, תוך כדי תהליך האימון מתבצעות פעולות משני סוגים - exploration ו-exploitation. כאשר הסוכן לא צבר ידע רב על סביבת המשחק (לרוב בתחילת שלב האימון), נרצה שהוא יחקור ויגלה את האפשרויות השונות באופן אקראי יחסית. לכן, לעיתים הסוכן ינקוט בפעולת explore: ביצוע פעולה אקראית לשם העשרת מאגרי הידע וההבנה על סביבת המשחק. לעומת זאת, נרצה שהסוכן גם יבצע את הפעולות שהוא מחשיב לטובות ביותר, ויעדכן את הערכותיו בהתאם.

לכן, כאשר הסוכן לא יבצע explore, הוא יבחר את הפעולה שנראית לו כטובה ביותר (נכון לאותו הרגע, כמובן), ויבצע אותה - exploit. בכל שלב באימון הסוכן יבחר באופן אקראי האם לבצע explore או exploit. ככל שהאימון נמצא בשלב מוקדם יותר, לסוכן אין ידע רב ולכן יש חשיבות לפעולת ה-explore. לעומת זאת, ככל ששלב האימון מתקדם יש פחות ופחות צורך ב-explore. לכן, ככל שמתקדם תהליך האימון ההסתברות שהסוכן יבחר ב-explore קטנה.

שיטת פעולה זו נקראת $\varepsilon$-חמדן. לאורך תהליך האימון קיים סף $\varepsilon$, וההסתברות לבחור ב-exploration תהיה $\varepsilon$. כאמור, ככל שתהליך האימון ההסתברות לבחור ב-explore תקטן, ולכן $\varepsilon$ יקטן תוך כדי תהליך האימון. עם זאת, נרצה שגם בשלבים מתקדמים של האימון תהיה לסוכן את האפשרות לנסות כיוונים שונים (פן יתקבע על שיטה לא רווחית). לכן, קיים $\varepsilon$ מינימלי (המוגדר מראש ש)הוא הסף המינימלי לביצוע פעולת explore - כך הסוכן יוכל לבצע exploration גם בשלבים מאוחרים של האימון.

## 2 $Q$-Learning

אלגוריתם $Q$-Learning הוא אלגוריתם שמשערך עבור כל מצב ופעולה את התגמול שיתקבל - כלומר, כמה רווחים תספק פעולה זו במצב זה לתוצאות המשחק (לטווח הקצר והארוך כאחד). הפונקציה איתה מבצעים את ההערכה היא $Q$. שיטת הפעולה של האלגוריתם:

1. השם ערך $Q$ שרירותי לכל זוג (מצב, פעולה).
2. בצע פעולות אקראיות, לאחר הפעולה ה-$t$, $a_t$, במצב ה-$t$, $s_t$, עדכן את $Q(s_t, a_t)$:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{הערך הישן}} + \underbrace{\alpha}_{\text{קצב הלמידה}} \cdot \underbrace{\left( \underbrace{\underbrace{r_t}_{\text{תגמול}} + \underbrace{\gamma}_{\text{מקדם ההנחה}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{הערכת התגמול הטוב ביותר}}}_{\text{הערך החדש}} - \underbrace{Q(s_t,a_t)}_{\text{הערך הישן}} \right)}$$

- $0 \le \alpha \le 1$, קצב הלמידה, קובע את המידה בה הידע הקודם שלנו על הערך של נקיטת הפעולה $a_t$ במצב $s_t$, משפיע על הערך החדש.
- $0 \le \gamma \le 1$ - ה-discount rate, קובע את ההשפעה של התגמול לטווח הארוך של פעולה זו על הערך החדש.

אלגוריתם זה מבוסס על תהליך החלטה מרקובי - MDP. MDP הוא מודל מתמטי, המכיל מרחב מצבים $S$, מרחב פעולות $A$, פונקציית תגמול $R$ ומקדם הנחה $\gamma$. בנוסף, פונקציית הסתברות $P : S \times S \times A \to [0,1]$, שמחזירה בעבור כל זוג $(s_1, s_2, a)$ את ההסתברות לעבור ממצב $s_1$ ל-$s_2$ ע"י נקיטת הפעולה $a$. עבור מודל שכזה, נרצה למצוא שיטה (מדיניות פעולה - policy), שתמקסם את התגמול הכולל במידה וננקוט בה. על כן, תהליך הלמידה $Q$-Learning יפתור את הבעיה עבור ה-MDP המתאים לו.

## 3 Deep $Q$-Learning

אלגוריתם ה-$Q$-Learning הבסיסי אינו קשור ללמידת מכונה, אך ניתן לשלב בין השניים. ב-$Q$-Learning נשמר ערך ה-$Q$ לכל זוג (מצב, פעולה), אך כאשר יש מספר גדול של מצבים (בדומה למשחק $2048$) ופעולות, תחזוקה של טבלת ערכי ה-$Q$ יקרה ולא יעילה.

במקום זאת, נשתמש ברשת נוירונים להערכת ערך ה-$Q$: מקבלת כקלט מצב, ומחזירה את ערכי ה-$Q$ לכל פעולה שתינקט על המצב. עדכון ערכי ה-$Q$ יתבצע ע"י אופטימיזציה של משקלי הרשת. ב- Deep $Q$-Learning נפעל באופן דומה ל-$Q$-Learning הרגיל, אך עם מספר שינויים ותוספים: ערכי ה-$Q$ יוערכו ע"י רשת נוירונים, ולא ע"י טבלה.

בכל אופטימיזציה יוגרלו חוויות מהזיכרון וישמשו ככקלט. הערך הרצוי ($y_{true}$) הוא $Q^{new}(s_t, a_t)$, ו-$\hat{y} = Q(s_t, a_T)$ הנוכחי. כך תחושב השגיאה ותתבצע האופטימיזציה לערך ה-$Q$ הנכון ביותר. בנוסף, כאשר הסוכן יבצע explore הפעולה תיבחר באופן אקראי בדומה לתהליך הרגיל, אך ב-exploit תיבחר הפעולה הטובה ביותר (בעלת ערך ה-$Q$ הגדול ביותר שמשוערך ע"י הרשת).

# חלק III
# מימוש

פרויקט זה מומש בעזרת ספריית PyTorch, המימוש המלא נמצא במחברת[2]. לפרויקט מספר חלקים מרכזיים.

## 1 סביבה

הפרויקט מכיל מימוש של שתי סביבות משחק: האחת, $Env2048$, היא מהירה ובסיסית יחסית, ושימשה לאימון הסוכן. הסביבה השנייה, $WebEnv2048$, היא איטית יותר ועובדת עם סביבת משחק אינטרנטית, ועל כן שימשה לבדיקה והמחשה של הישגי הסוכן המאומן באופן יותר נוח. שתי הסביבות יורשות מהמחלקה האבסטרקטית $Env$: מחלקה שמכילה את הפעולות המשותפות לשתי הסביבות ($is\_done, max\_tile, \_can\_perform, possible\_actions$ ו-$empty\_tiles$). בנוסף, לכל סביבה יש את הפעולות היחודיות שלה:

- בסביבה $Env2048$ ממומשים כללי המשחק. התכונות הן $grid$ (לוח המשחק), מערך $numpy$ בגודל $4 \times 4$ ו-$score\_$, הניקוד הכולל של המשחק.
  - הפעולה $reset$ מאתחלת את הסביבה למצב ההתחלתי על ידי קריאה לבנאי: לוח ריק וניקוד $0$.
  - הפעולה $add\_tile$ מוסיפה ערך חדש (לפי חוקי המשחק) במשבצת ריקה, ומניחה שקיימת משבצת כזאת.
  - הפעולה $step$ מסובבת את הלוח מספר פעמים, כך שהפעולה תהיה שקולה לתזוזה שמאלה.
    * הפעולות הן $0 : left, 1 : up, 2 : right, 3 : down$ (לעיתים אשתמש באות הראשונה של הפעולה בלבד - $L$ עבור שמאלה וכו').
    * תזוזה מעלה שקולה לסיבוב אחד, ימינה לשני סיבובים ומטה ל-$3$ סיבובים.
    * הפונקציה מחזירה את התגמול שהתקבל מביצוע הפעולה.
  - הפעולות $score$ ו-$state$ מחזירות את הניקוד והמצב הנוכחי בהתאמה, והפעולה $render$ מדפיסה את הלוח באופן בסיסי.
- הסביבה $WebEnv2048$ עובדת עם סביבת המשחק האינטרנטית, ולכן מעבדת את המצב הנוכחי והניקוד מהסביבה המוכנה. נעזרתי ב ... + קישור
  - הפעולה הבונה מתחברת לאתר.
  - שאר הפעולות דולות את הנתונים המתאימים מהאתר ופועלות בהתאם - מחזירות ערך / מבצעות פעולה.
  - בכל רגע נתון ניתן לראות בזמן אמת את המצב באתר.
- בשתי הסביבות הלוח מיוצג ע"י חזקות של $2$: בתא ה-$i, j$ יהיה ערך $k$ אם באותה המשבצת יש את הערך $2^k$, או $0$ אם המשבצת ריקה.
- התגמול המוחזר מהפעולה $step$ הוא מספר המשבצות הריקות, מפורט בהמשך המחברת.

[2] קישור: https://github.com/bowass/2048-DQN-Agent/blob/main/RL_2048_DQN_Agent.ipynb.

## 2 מודל

המודל הכולל מורכב ממספר חלקים: זיכרון, רשתות policy ו-target, ופונקציות אימון. בבניית מודל זה נעזרתי במדריך של PyTorch[3] ל-$DQN$.

- המחלקה ReplayMemory, המתחזקת Experience Memory לשימוש הסוכן.
- המחלקה DQN, שמגדירה את הצורה בה בנויות רשתות ה-policy וה-target.
- פונקציות עזר.
  - הפונקציה $one\_hot\_encode\,(state)$ מקודדת את המידע - בהינתן מערך $numpy$ בגודל $4 \times 4$ (הלוח), מוחזר מערך תלת מימדי, שמכיל $16$ מערכי $4 \times 4$. במערך ה-$i$ יש ערך $1$ בכל המשבצות בהן יש את הערך $2^i$ (במערך הראשון - המשבצות הריקות), ובכל שאר המקומות אפסים. מאחר ויש $16$ מערכים, המודל מסוגל לעבוד עם משבצות בגודל $2^{15} = 32768$ לכל היותר (ערך מקסימלי סביר, ואין טעם להקצות יותר). כך קודדנו את הלוח המקורי למערך תלת מימדי, ויהיה יותר נוח לעבוד איתו: באופן אינטואיטיבי, מאחר ומיזוג יכול לקרות רק בין שתי משבצות שוות ערך, יהיה יותר נוח להפריד בצורה זו ולהסתכל ממבט-על על כל האפשרויות למיזוג.
  - הפונקציה $select\_action\,(state)$, שבוחרת את הפעולה הנכונה לסוכן ע"פ ה-policy או בחירה אקראית (כתלות ב-$\varepsilon$): $\varepsilon$-חמדן.

הערה. בפונקציה $select\_action$ לסוכן לא מתאפשר לבצע פעולות לא חוקיות (כאלה בהן לא יהיו מיזוגים ומשבצות לא יזוזו למקומות ריקים).

כאשר אפשרתי לסוכן לבצע פעולות לא חוקיות, לעיתים קרובות (בעיקר בתחילת תהליך האימון) הסוכן "נתקע" על אותן פעולות - שלא משנות את מצב הלוח.

הדבר גרר זמני אימון ארוכים ולא משמעותיים (הסוכן "נתקע" גם בשלבים מאוחרים של האימון), מאחר והסוכן בחר בפעולה אחרת רק כאשר הוגרלה פעולה אקראית ב-exploration.

על כן, כאשר מתבצע explore מוגרלת פעולה מבין הפעולות החוקיות לשלב זה, וב-exploit נבחרת הפעולה החוקית הטובה ביותר.

- פונקציות אימון.
  - הפונקציה $optimize\_model\,()$, שמעדכנת את הפרמטרים של ה-policy בהתאם ל-batch של חוויות שנבחר מהזיכרון.
  - הפונקציה $train\,(env, episodes, save\_rate, print\_rate)$, שמאמנת את הסוכן על סביבה $env$ לאורך $episodes$ פרקים, שומרת את פרמטרי הרשת כל $save\_rate$ פרקים ומדפיסה את המצב הנוכחי כל $print\_rate$ פרקים.

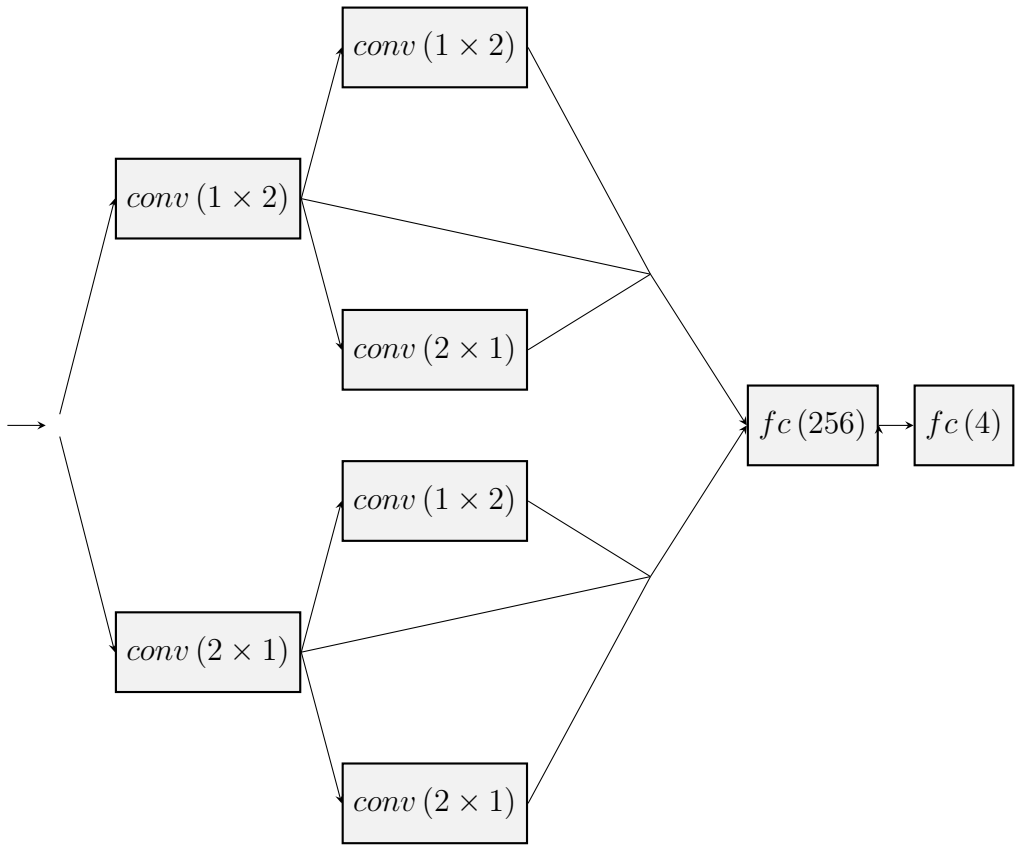
בנוסף, למודל זה מספר היפר-פרמטרים.

[3] קישור: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.

| שם ההיפר-פרמטר | משמעות | ערך |
|---|---|---|
| $GAMMA$ | מקדם ההנחה $\gamma$ | 0.99 |
| $EPS\_START$ | ה-$\varepsilon$ ההתחלתי | 0.1 |
| $EPS\_END$ | ה-$\varepsilon$ הסופי (המינימלי) | 0.0001 |
| $DECAY$ | מספר הצעדים מה-$\varepsilon$ ההתחלתי לסופי | 10000 |
| $BUFFER\_SIZE$ | הגודל המקסימלי של הזיכרון | 50000 |
| $BATCH\_SIZE$ | גודל ה-sample מהזיכרון בכל אופטימיזציה | 128 |
| $UPDATE\_RATE$ | תדירות עדכון הפרמטרים מה-policy ל-target | 10 |
| $LEARNING\_RATE$ | קצב הלמידה $\alpha$ | 0.0001 |
| $episodes$ | מספר פרקי האימון | 3000 |

טבלה 1: היפר-פרמטרים בהם המודל משתמש.

. המודל מכיל שכבות קונבולוציה ו-Fully Connected, ועוצב בהשראת המאמר Developing Value Networks for Game 2048 with RL[4].




איור 2: דיאגרמה שמתארת את מבנה המודל. הקלט בגודל $4 \times 4 \times 16$ נכנס לשכבות הקונבולוציה הראשונות כתמונה עם 16 ערוצים, ולבסוף מתקבל הפלט - ערכי ה-$Q$ לכל אחת מהפעולות. גודל הפילטר בשכבות $conv\,(1 \times 2)$ הוא $1 \times 2$, ובשכבות $conv\,(2 \times 1)$ הוא $2 \times 1$.

[4] קישור: https://www.jstage.jst.go.jp/article/ipsjjip/29/0/29_336/_pdf/-char/ja.

מדוע הרשת בנויה כך?

- תחילה, הקלט שנכנס לרשת מפריד בין הערכים השונים, וכך כל מערך $4 \times 4$ מייצג את המקומות בהם נמצאים ערכים מסוג ספציפי.
- הפילטרים בכל שכבות הקונבולוציה הם מהצורה $1 \times 2$ או $2 \times 1$. נסתכל על שתי שכבות הקונבולוציה הראשונות:
  - השכבה עם פילטר בגודל $1 \times 2$ יעזור לזהות מיזוג אנכי בין שתי משבצות צמודות: שני 1ים צמודים במערך כלשהו, אחד מעל השני.
  - באופן דומה, פילטר בגודל $2 \times 1$ יעזור לזהות מיזוג אופקי בין שתי משבצות צמודות.
- לאחר מכן, $4$ שכבות הקונבולוציה לאחר מכן יוכלו לזהות מיזוגים אנכיים/אופקיים עבור משבצות שאינן צמודות, ואפילו מיזוגים שיצריכו פעולה אופקית ואחריה אנכית.
- מאחר ואנחנו מתעלמים במידה מסוימת משאר המשבצות כאשר אנחנו מתייחסים לערך ספציפי, ייתכן שנזהה מיזוגים פוטנציאליים כאשר אין כאלה (יש משבצת מערך אחר בין שתי המשבצות, ולא ראינו זאת כי הסתכלנו על ערך אחד בלבד). עם זאת, מיזוגים בין משבצות צמודות לא יכולים להיפגע מכך.
- על כן, לאחר זיהוי של מיזוגים פוטנציאליים לכל כיוון - נשטח את הנתונים ונעביר לשכבות Fully Connected, ולבסוף $4$ נוירוני פלט - אחד לכל פעולה.

הערה. אציין כי המיזוגים שנוצרים מתזוזה ימינה יהיו זהים לתזוזה שמאלה, ובאופן דומה עבור מעלה/מטה. לכן, הפרדה למיזוגים אנכיים ואופקיים לא פוגעת באבחנה בין הפעולות השונות. בחלק ניתוח התוצאות יורחב על ההשלכות האפשריות של מבנה המודל והאסטרטגיה שפותחה במהלך האימון. בנוסף, למניעת בלבול - מחברת הפרויקט נכתבה עם PyTorch. בשכבות הקונבולוציה ב-PyTorch מספר הפילטרים הוא השדה הראשון, ולכן קידוד התמונות הוא לצורה $(1, 16, 4, 4)$: ה-$1$ לצורך נוחות כאשר עורמים מספר דוגמאות ל-batch, מספר הפילטרים הוא $16$ והגודל ההתחלתי הוא $(4, 4)$.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 128, 4, 3]           4,224
            Conv2d-2            [-1, 128, 3, 4]           4,224
            Conv2d-3            [-1, 128, 4, 2]          32,896
            Conv2d-4            [-1, 128, 3, 3]          32,896
            Conv2d-5            [-1, 128, 3, 3]          32,896
            Conv2d-6            [-1, 128, 2, 4]          32,896
            Linear-7                  [-1, 256]       1,900,800
              ReLU-8                  [-1, 256]               0
            Linear-9                    [-1, 4]           1,028
================================================================
Total params: 2,041,860
Trainable params: 2,041,860
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 7.79
Estimated Total Size (MB): 7.85
----------------------------------------------------------------
```

איור 3: מבנה המודל שלפיו בנויות רשתות ה-policy וה-target. שתי שכבות הקונבולוציה הראשונות מהוות קלט ל-4 האחרונות. לאחר מכן שרשור ושיטוח פלט כל 6 השכבות לתוך שכבות לינאריות, ולבסוף החזרת ערכים ל-4 הפעולות.

# 3 התקנה ותפעול

1. פתיחת המחברת[5] ויצירת תיקייה וקבצים עבור הרשתות (mount ל-drive דרך colab או תיקייה מקומית במידה ומריצים לוקאלית).

2. אימון הסוכן - שתי אפשרויות.

(א) הרצת כל פרקי החלק Train, כולל אלו המתאימים לאימון מחדש של הסוכן ושמירת תוצאות האימון החדש בקבצים.

(ב) העתקת הקבצים המאומנים והרצה של כל הפרקים מלבד Train Agent.

3. בדיקת הסוכן - הרצת הבלוק Test. תחילה, טעינת המשקלים של הסוכן המאומן ל-policy_net, לאחר מכן הרצה בסביבה המתאימה וויזואליזציה של התוצאות.

(א) הרצה על סביבת $Env2048$, ללא ממשק נוח אך מהירה יותר ומומלצת עבור מספר פרקים גדול.

(ב) הרצה על סביבת $WebEnv2048$, המתממשקת עם סביבה אינטרנטית של המשחק[6] (עובדת רק כאשר מריצים לוקאלית, google colab יעבוד רק עם סביבת $Env2048$).

[5] קישור: https://github.com/bowass/2048-DQN-Agent/blob/main/RL_2048_DQN_Agent.ipynb.
[6] קישור: https://play2048.co/.

## חלק IV

# מחקר וניתוח תוצאות

תחילה, למודל זה יש מספר רב של היפר-פרמטרים, וכוונון של כל אחד מהם יכול להשפיע על צורת הלמידה ויכולות הסוכן המאומן. לכן, אשווה בין 3 דרכים בהן הסוכן התאמן:

| | שיטת התגמול | | שכבות המודל | |
|---|---|---|---|---|
| | ניקוד כולל | משבצות ריקות | Fully Connected | קונבולוציה |
| #1 | ✓ | ✓ | ✓ | |
| #2 | ✓ | ✓ | ✓ | ✓ |
| #3 | | ✓ | ✓ | ✓ |

טבלה 2: שלוש הדרכים בהן אומן הסוכן ותכונותיהן.

לאחר שהתעמקתי בנושא וחקרתי פרויקטים שנעשו, ראיתי כי סוכן ה-$DQN$ הבסיסי לא מצליח בסביבת משחק זו: לאחר קריאת חומרים, פרויקטים ומאמרים שעסקו בנושא[7], נראה ש-$DQN$ אינו מצליח להגיע לניקוד גבוה מ-512, וברוב המוחלט של המשחקים נעצר ב-256. לכן, אלגוריתם חיפוש בסיסי (או אפילו מהלכים אקראיים!) יתגברו על תוצאות הסוכן. בנוסף, לכל אורך האימון הסוכן יכל לבצע מהלכים חוקיים בלבד, וכך לא "נתקע" על מהלכים חוקיים. הדבר שיפר את זמני האימון והקל על התהליך, מה גם שבכל מקרה מהלכים לא חוקיים אינם אפשריים במשחק, ולא מקלים על הסוכן יתר על המידה עם פיצ'ר זה.

בנוסף, בשלוש דרכי האימון השתמשתי ב-GPU מקומי - מה שהאיץ את תהליך האימון רבות. ניתן גם לאמן את המודל באמצעות CPU אך התהליך יהיה משמעותית יותר איטי. (ניתן גם להשתמש ב-GPU של google colab, אך ייתכן שלא יספיק עבור המודלים המורכבים).

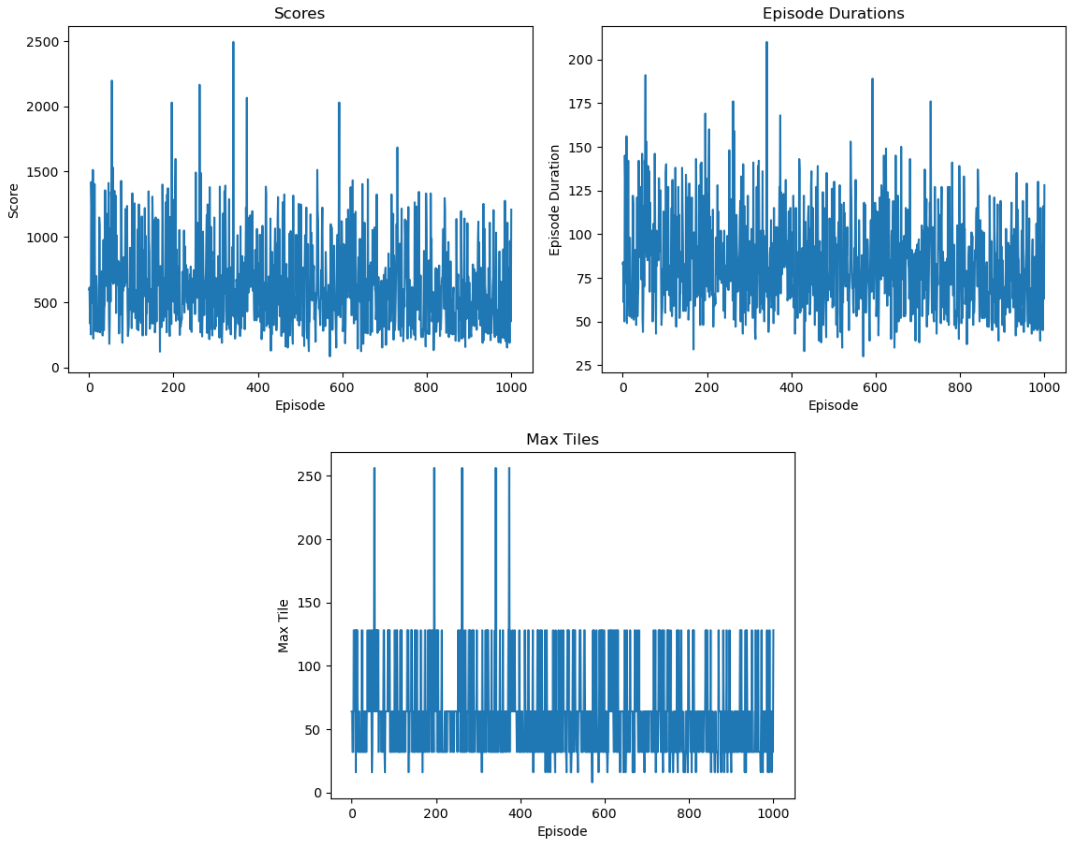
## 1 שיטת אימון #1

בתחילת הפרויקט עבדתי בדרך #1, האימון היה קצר (כ-10 דקות בלבד) מאחר והמודל היה פשוט מאוד ביחס לשאר השיטות.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 512]         131,584
            Linear-2                  [-1, 256]         131,328
            Linear-3                  [-1, 128]          32,896
            Linear-4                    [-1, 4]             516
================================================================
Total params: 296,324
Trainable params: 296,324
Non-trainable params: 0
```

איור 4: מודל ה-$DQN$ מדרך #1.

[7] קישורים: https://arxiv.org/pdf/2110.10374v1.pdf, https://github.com/dsgiitr/rl_2048.

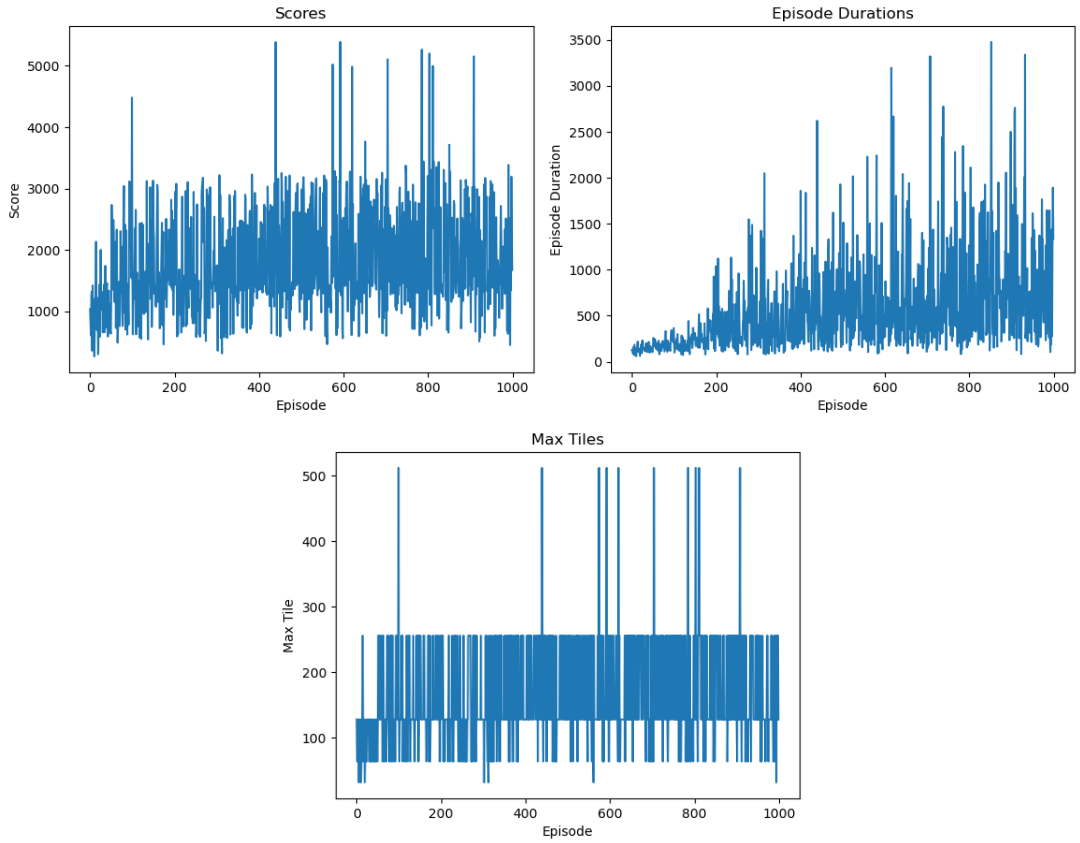



איור 5: תוצאות אימון הסוכן בדרך #1. מימין לשמאל: אורכי הפרקים (מספר המהלכים), ניקוד כולל וערך מקסימלי שהושג במשחק.

בדומה למאמרים ולפרויקטים אחרים שראיתי, גם כאן תוצאות הסוכן אינן מרשימות במיוחד: הניקוד המקסימלי שמושג הוא 128 ברוב המקרים, ולעיתים גם ל- 256.

## 2 שיטת אימון #2

לאחר מכן ניסיתי לסבך את המודל - דרכים #2 ו-#3 משתמשות באותו המודל, שתואר קודם לכת בחלק המימוש. עם זאת, בדרך #2 יש חשיבות לניקוד הכולל, ובדרך #3 מערכת התגמול תלויה במספר המשבצות הריקות בלבד. תחילה אימנתי בדרך #2. האימון היה ארוך יחסית, כ-5 שעות ורבע, מאחר והמודל מסובך הרבה יותר מאשר המודל מדרך #1.




איור 6: תוצאות אימון הסוכן בדרך #2. מימין לשמאל: אורכי הפרקים (מספר המהלכים), ניקוד כולל וערך מקסימלי שהושג במשחק.

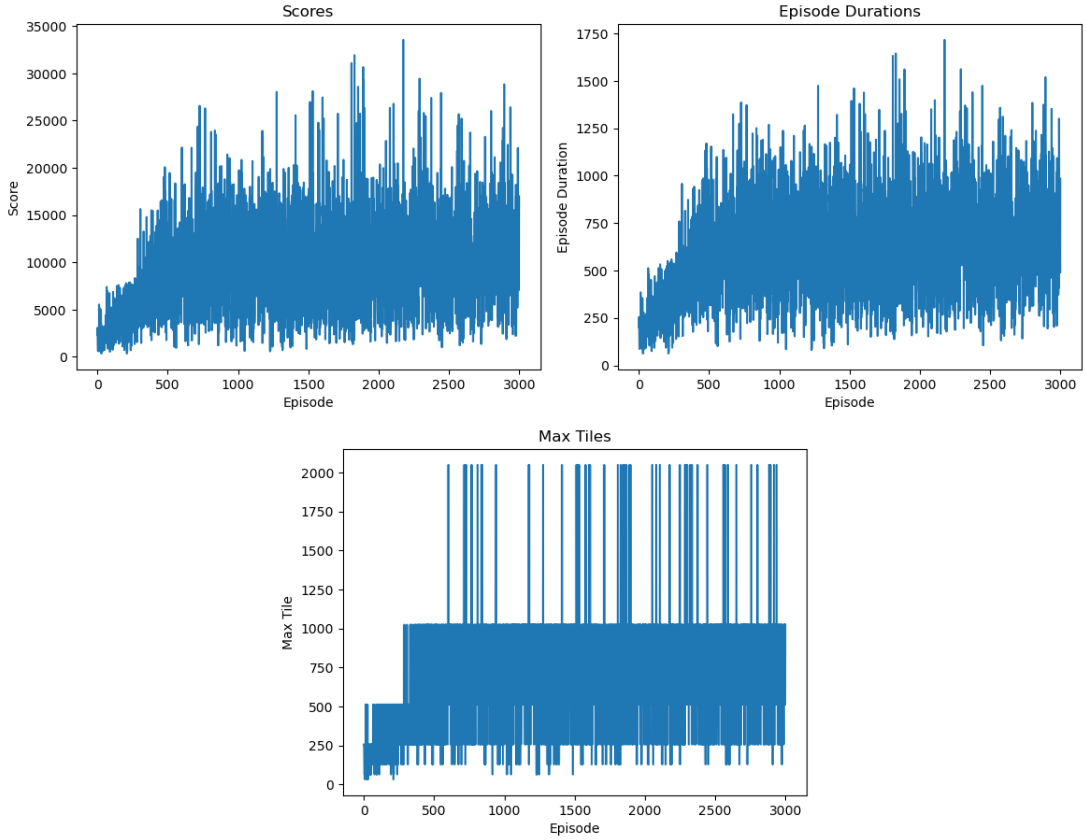
יש שיפור בתוצאות - הסוכן מצליח להגיע פעמים רבות ל-$256$, ולעיתים גם ל-$512$. כלומר, ככל הנראה שהמודל עליו התבססה דרך #1 לא היה מסובך מספיק, ומודל זה מספק תוצאות טובות יותר. עם זאת - התוצאות עדיין לא מספיקות, וקיימים אלגוריתמי חיפוש (בין אם מבוססי למידת מכונה ובין אם לא) שמגיעים לתוצאות יותר טובות פי כמה וכמה[8]. ייתכן שסיבוך המודל עוד ועוד ישפר את תוצאות הסוכן, אך תהליך האימון יהיה ארוך ומפרך מדי כתוצאה מהמודל היקר והכבד. על כן, ניתן להישאר עם אותו המודל ולשנות את הגישה: שינוי שיטת התגמול.

[8] קישור: https://github.com/nneonneo/2048-ai, ניתן לקרוא את הדיון ב-Stack Overflow

# 3 שיטת אימון #3

עד כה, שיטת התגמול בו השתמש הסוכן הייתה שילוב בין מספר המיזוגים והשיפור בניקוד בין המצבים, ומספר המשבצות הריקות (מתוך הנחה ושיעור שהשילוב ביניהן יאזן את הצורה בה פועל הסוכן ויושגו תוצאות גבוהות). עם זאת, נשים לב כי כאשר מתמקדים במספר המשבצות הריקות בלבד, הניקוד חייב לעלות והגעה למשבצות עם ערך גבוה נגררת באופן מיידי: מספר המשבצות הריקות תלוי במספר המיזוגים - ככל שיהיו יותר מיזוגים כך מספר המשבצות הריקות יגדל. כך, אם נייחס חשיבות למספר המשבצות הריקות בלבד יקרו הרבה מיזוגים, והניקוד והמשבצת המקסימלית יהיו מוכרחים לגדול.

על כן, מערכת התגמול בדרך #3 מתבססת על מספר המשבצות הריקות בלבד. זמן האימון היה ארוך גם כן - כ-5 שעות ו-45 דקות. זמן האימון תלוי באורכי הפרקים והצלחת הסוכן: ככל שהסוכן מצליח יותר, לכל פרק לוקח יותר זמן להיגמר וכך זמן האימון היה ארוך יותר גם מדרך #2.




איור 7: תוצאות אימון הסוכן בדרך #3. מימין לשמאל: אורכי הפרקים (מספר המהלכים), ניקוד כולל וערך מקסימלי שהושג במשחק.

ניתן לראות שיפור משמעותי מאשר שתי הדרכים הקודמות - לאחר כמה מאות פרקים בלבד הסוכן עקף באופן משמעותי את 2 הדרכים הקודמות. ברוב המוחלט של המקרים המשבצת המקסימלית היא 1024, ולעיתים אף ל-2048.

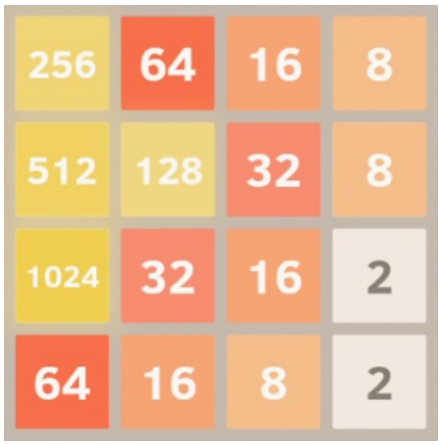
בסך הכל, קיבלנו כי האימון בדרך #3 מניב תוצאות טובות הרבה יותר מאשר שאר הדרכים. מאחר ובחלק מגרסאות המשחק המטרה היא הגעה לערך 2048 בלבד, ניתן לראות

את פרויקט זה כהצלחה - ואימון הסוכן בדרך #3 גורר את ניצחון המשחק במקרים רבים. עם זאת, ייתכן כי סיבוך המודל, אימון לאורך זמן רב יותר, שינוי מערכת התגמול או שילוב עם אלגוריתמי חיפוש שונים ישפרו את תוצאות הפרויקט ויניבו תוצאות גבוהות הרבה יותר, והנ"ל מהווים כיוון להמשך מחקר ועבודה בנושא.

בסרטון[9] ניתן לראות משחק בודד של הסוכן והגעה לניקוד 2048. בנוסף, ניתן ללמוד מהסרטון על האסטרטגיה שפותחה ע"י הסוכן בזמן האימון:

- הסוכן נוטה לרכז את הערכים הגדולים בקצה הלוח, ולהשאיר את המקומות המרכזיים בלוח לערכים קטנים. בנוסף, הסוכן כמעט ולא פונה ימינה - וכך מרכז את הערכים הגדולים בקצה השמאלי של הלוח.

- הסוכן מצליח לערום ערכים ולאגדם כך שבהינתן משבצת נוספת עם ערך קטן, נוצרת שרשרת של מיזוגים. המחשה:




איור 8: מתוך הסרטון (זמן 1:43). בהינתן סדר הפעולות הבא (מימין לשמאל), שרשרת המיזוגים תגיע ל-2048: $U,U,L,U,L,U,L,U$. הסוכן פועל באופן זה ומגיע כך לניקוד גבוה.

- כאמור, פונקציית המטרה של הסוכן היא מספר המשבצות הריקות. כאשר הסוכן מצליח לצמצם את מספר המשבצות, הוא חייב למזג ערכים ולהגיע לערכים גדולים יותר.

  - כך הסוכן מגיע להישגים גבוהים בהרבה מאשר עבודה עם פונקציית מטרה שמתייחסת לניקוד הכולל.

[9] קישור: https://github.com/bowass/2048-DQN-Agent/blob/main/2048%20DQN%20Agent.mp4.

## חלק V

# סיכום אישי

בפרויקט זה צברתי המון ידע חדש - עסקתי ב-RL, נושא שלא כלול בתכנית הלימודים הרגילה. יחד עם תמיכתו של אריאל למדתי באופן עצמאי את הנושאים השונים, ואני שמח על הישגיי בפרויקט זה. העבודה לא הייתה פשוטה - קראתי מחקרים ומאמרים שנכתבו בנושא, בתקווה שאצליח לגרום לסוכן להצליח.

כפי שציינתי קודם לכן, אלגוריתם ה-DQN הבסיסי עם מודל לא מפותח לא מצליח להגיע להישגים גבוהים במשחק. כאשר הסוכן הצליח להגיע להישגים גבוהים בשיטות האימון המאוחרות סופקתי ושמחתי מאוד. עם זאת, יש מקום רב לשיפור הביצועים של המודל - מספר כיוונים לשיפורים והמשך מחקר:

1. שינוי ההיפר-פרמטרים, לרבות מערכת התגמול ומספר פרקי האימון.
2. מודל DQN מסובך יותר, או כזה שעובד ע"פ עקרונות אחרים.
3. קידוד הלוח באופן חכם יותר, כך שהמודלים הקיימים יעבדו באופן יותר טוב ויגיעו להישגים גבוהים יותר.
4. עבודה עם אלגוריתמי RL אחרים, או שילוב של DQN עם היוריסטיקות / אלגוריתמי חיפוש.

# חלק VI
# ביבליוגרפיה

Paszke, A. (2022). Reinforcement Learning (DQN) Tutorial.
PyTorch. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.

Li, S & Peng, V (2020). Playing 2048 With Reinforcement Learning. Stanford University.
https://arxiv.org/pdf/2110.10374v1.pdf.

Matsuzaki, K (2021). Developing Value Networks for Game 2048 with Reinforcement
Learning. Journal of Information Processing, Val.29 336-346.
https://www.jstage.jst.go.jp/article/ipsjjip/29/0/29_336/_pdf/-char/ja.

nneonneo. 2048-ai. github. https://github.com/nneonneo/2048-ai.

- סביבת המשחק האינטרנטית, שאיתה מתממשקת הסביבה $WebEnv2048$: https://play2048.co/.
- דיון ב-Stack Overflow על אסטרטגיות שונות לאימון מודל למשחק:
https://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048/22498940#22498940.
- תיקיית הפרויקט ב-github - https://github.com/bowass/2048-DQN-Agent, שמכילה:
  - מחברת הפרויקט.
  - משקלי המודלים המאומנים.
  - סרטון להמחשה של הסביבה $WebEnv2048$, בו הסוכן מגיע לניקוד 2048.
  - תיק פרויקט זה.