

**Geometric Image Processing Laboratory**

**TermiNet -** Learning a First Person Shooter Game Agent Using Deep Learning

סמסטר אביב תשע"ו

**מגיש**: איתי כספי

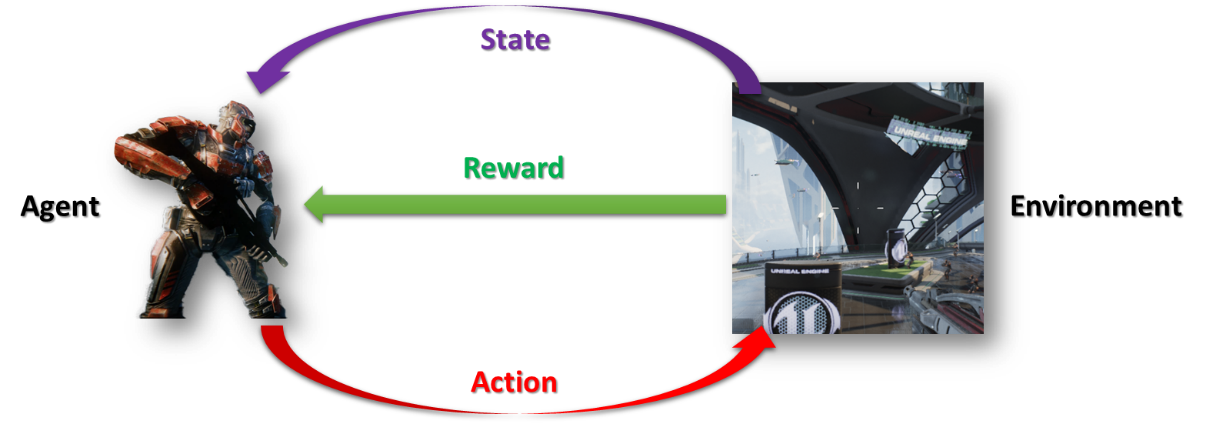
**מנחה**: אהרון וצלר

1. **מבוא**

מטרת הפרויקט הינה שימוש בפיתוחים אחרונים בתחום של Deep Reinforcement Learning לטובת למידת סוכן המשחק במשחק First Person Shooter. משחקי First Person Shooter מכילים סביבה עשירה ביחס למשחקים עליהם נוסה התחום עד כה, מציגים עולם תלת מימדי מנקודת מבטו של השחקן, ודורשים שילוב של כישורים שונים על מנת לנצח במשחק. במהלך הפרויקט השתמשנו בשתי גישות שונות לפתרון הבעיה המפרקות את התנהגות הסוכן לסט כישורים אשר נלמדים בנפרד.

2. **רקע**

באלגוריתמי למידה מחיזוקים אנו ממדלים את הבעיה כסוכן הפועל מול סביבה. במודל זה, הסוכן מקבל מהסביבה תצפיות וחיזוקים (חיוביים או שליליים) והוא יכול לבצע פעולות מסויימות על הסביבה. תחת מודל זה, מטרת הסוכן היא להגדיל את החיזוקים אותם הוא מקבל. מבחינה מתמטית הצורה המקובלת למידול הבעיה היא כתהליך החלטות מרקובי.



**תמונה 1.1: מודל ה-Reinforcement Learning**

**Markov Process**

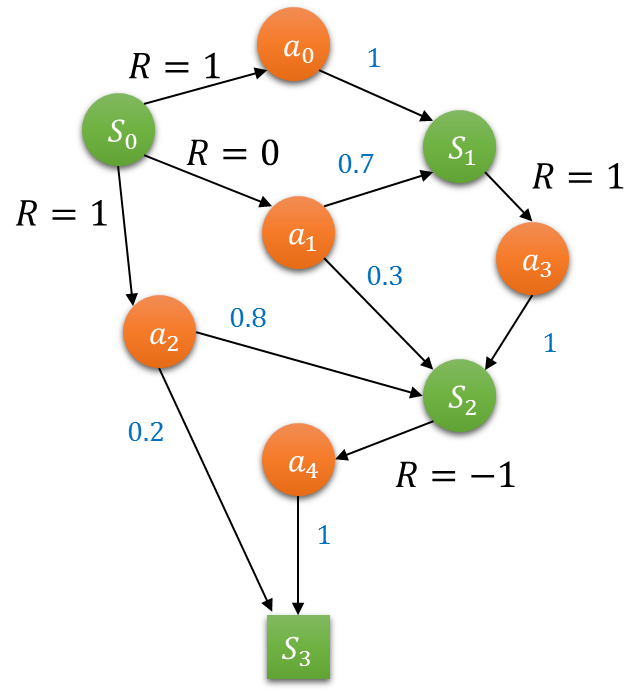
תהליך מרקובי הוא קבוצה של מצבים והסתברויות המעבר ביניהם כאשר מצב הוא מרקובי אם ורק אם מתקיים –

כלומר, בהנתן המצב הנוכחי, המצב הבא תלוי רק בו ובלתי תלוי סטטיסטית בכל המצבים הקודמים לו. בנוסף, מסמנים את הסתברות המעבר בין שני מצבים ב- כאשר היא מטריצת הסתברויות המעברים -

**Markov Decision Process (MDP)**

תהליך החלטות מרקובי הוא סוג של תהליך מרקוב שבו מכל מצב ניתן לקבל החלטות שונות שיובילו למצבים הבאים בהסתברויות שונות. מבחינה מתמטית, התהליך מוגדר כשישייה מהצורה - כאשר -

* – קבוצה סופית של מצבים מרקוביים
* – קבוצה סופית של פעולות
* – התפלגות מעבר בין מצבים
* - פונקציית reward
* - התפלגות מצבים התחלתיים
* – פרמטר discount

**

**תמונה 1.2: דוגמא ל-Markov Decision Process**

*מעבר או* ***transition*** *הוא רביעייה אשר מייצגת את מצב, את הפעולה שנלקחה ממצב זה, ה-reward שהתקבל והמצב אליו עברנו.*

*בנוסף מגדירים מדיניות או* ***policy*** *- להיות התפלגות מעל סט הפעולות בהינתן המצבים, כך ש -*

*במידול של תהליכים שונים כתהליכי החלטות מרקוביים, אנחנו נתעניין בתוחלת ה-reward-ים העתידיים או ה-****Return*** *אותם נקבל מכל מצב בהינתן ה-policy לפיו אנו פועלים.*

*בהמשך נתעניין בורייאנט ספציפי של ביטוי זה –* ***Discounted Return****, שמשתמש בפרמטר ה-discount כדי לייחס חשיבות גדולה יותר ל-reward בעתיד הקרוב על פני reward רחוקים יותר. נסמן זאת על ידי –*

*כאשר -*

**Reinforcement Learning**

בלמידה מחיזוקים, אנחנו ממדלים סוכן שפועל במסגרת תהליך החלטות מרקובי ומטרתו למקסם את ה-reward אותו הוא מקבל מהסביבה. הסוכן פועל לפי policy מסויים והמטרה בלמידה היא לבצע אופטימיזציה ל-policy זה. בפרויקט זה עסקנו בעיקר באלגוריתמי אופטימיזציה מסוג Policy Iteration אותם נתאר בהמשך.

למידה מחיזוקים מתחלקת לשני סוגים עיקריים – למידה מבוססת מודל, ולמידה לא מבוססת מודל. אנחנו נתעסק בעיקר בלמידה לא מבוססת מודל, שבה ה-reward לא ידוע לסוכן מראש ועליו לחזות אותו. לשם כך, נגדיר שתי פונקציות להערכת ה-expected discounted reward העתידי:

* **state value function** – פונקציה המשערכת את ה-reward העתידי בהינתן מצב ו-policy מסויים
* **state-action value function** – פונקציה המשערכת את ה-reward העתידי בהינתן מצב והפעולה שנלקחה ממצב זה לפי ה-policy המסויים

ברור מההגדרה שקיים קשר חזק בין הפונקציות הנ"ל. קשר זה מבוטא באמצעות ה-advantage function –

בהינתן ה-state-action value function - מכל מצב ופעולה, בחירת ה-policy האופטימלי על ידי הסוכן תתבצע בצורה הבאה –

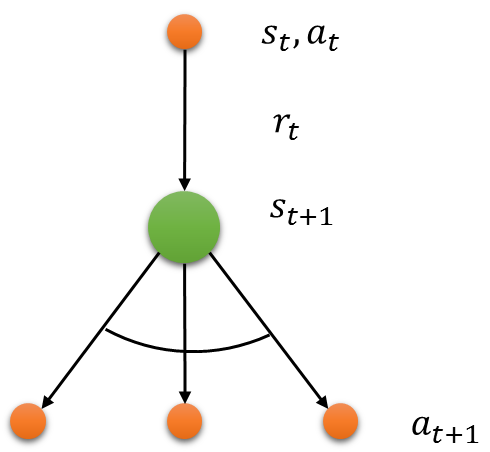
1. בהינתן מצב , מצא את עבור כל פעולה אפשרית
2. בחר את הפעולה שמניבה את ערך ה- המקסימלי

לכן, נרצה ללמוד את הפונקציה . אלגוריתמי האופטימזציה האיטרטיביים השונים מבצעים שערוך של value functions אלו כדי ללמוד את ה-policy האופטימלי על ידי שימוש במשוואת בלמן. משוואת בלמן פורסת את ה-reward העתידי בצורה רקורסיבית. כלומר –

*באופן שקול ניתן לפרוס את ה-value functions –*

באלגוריתם **Q-learning**, נתחיל מפונקציית רנדומלית, ונעדכן את הפונקציה בהתאם ל-transitions אותם חווה הסוכן. הסוכן יפעל ביחס ל- function זה באופן חמדני לפי האלגוריתם שתואר. כלומר בכל שלב, בהנתן מצב, הסוכן יבחר את הפעולה שממקסמת את ה-, יבצע אותה ויקבל reward. בשלב זה הסוכן יהיה מיודע יותר לערך הנכון של מתוך משוואת בלמן –

כאשר היא הפונקציית המיודעת יותר. שימוש בשערוך קודם לטובת שערוך חדש נקרא **bootstraping** וישמש אותנו באלגוריתם זה.



**תמונה 1.3: אלגוריתם Q-learning. בהינתן transition, האלגוריתם מבצע Bootstrap לפעולה הממקסמת את ערך המצב הבא**

נגדיר את ה-**Temporal Difference Error** להיות –

כאשר באופן כללי משמעותו פריסה של משוואת בלמן על פני צעדים.

נשים לב ש הוא נגזרת השגיאה הריבועית הממוצעת (MSE) בין ה-target Q function לבין ה-actual Q function הנוכחי - . לכן נוכל לבצע צעדי אופטימזציה על ידי אלגוריתם Gradient Descent –

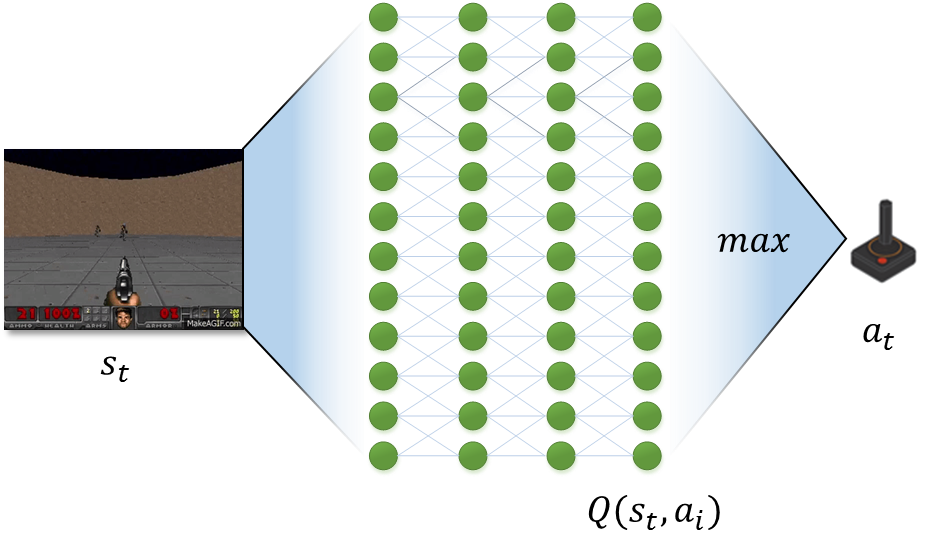
כאשר הוא גודל צעד העדכון.

לסיכום, האלגוריתם הבסיסי יהיה מהצורה הבאה –

1. לכל זמן :
   1. קבל מצב מהסביבה
   2. הערך את לכל אפשרי
   3. בצע צעד חמדני לפי וקבל חיזוק ואת המצב הבא
   4. עדכן את לפי ה- כפי שתואר לעיל.

**Deep Q Network (DQN)**

מודל זה משלב את האלגורתמיקה של Q-learning עם שערוך פונקציה על ידי רשתות נוירונים.



**תמונה 1.4: שילוב של רשת נוירונים כמשערך של פונקציית ה-Q**

בכניסה מוזנת הרשת במצב . הרשת בנויה ממספר שכבות שבכל אחת מהן מתבצעת קומבינציה לינארית של הכניסות לפי משקולות נלמדות, ולאחר מכן פעולה לא לינארית. כלומר בהינתן קלט לשכבה ה-, משקולות ואיברי bias - , כל נוירון בשכבה ה- מבצע את הפעולה הבאה –

כאשר היא פעולה לא לינארית כדוגמת –

המוצא של הרשת הוא ערך עבור כל פעולה אפשרית ממצב . לאחר הרצת המצב ברשת וקבלת ערכי עבור כל הפעולות האפשריות, בוחרים את הפעולה המניבה את ערך ה- המקסימלי.

המשקולות של הרשת נלמדות על ידי אלגוריתם ה-backpropagation. כפי שראינו, בכל שלב מקבלים את ערך ה- המוערך הנוכחי, מבצעים פעולה ומחשבים ערך מיודע יותר. בשלב זה מחשבים את ה-TD error שמניב את הגרדיינטים הנדרשים לעדכון, ומבצעים עדכון Gradient Descent על הפלט של הרשת. כדי לעדכן את המשקולות, מחלחלים את הגרדיינטים לאורך הרשת על ידי שימוש בכלל השרשרת לנגזרות.

במקרה שלנו, כיוון שמצב העולם הוא תמונה, משתמשים ברשתות נוירונים עם שכבות קונבולוציה שמנצלות את הלוקליות במקום של מאפייני התמונה. שכבות אלו שקולות לשכבות הלינאריות שתיארנו לעיל אך במקום שכל נוירון מוצא יהיה פונקציה של כל הקלטים, הוא פונקציה של אזור לוקלי קטן בקלטים.

שילוב של רשתות נוירונים כמשערך של פונקציית ה- אינו טריוויאלי. עדכוני ה-Gradient Descent מתבצעים בצורה סטוכסטית על פני batch של transitions, ולכן יש חשיבות גדולה לכך שלא תהיה קורלציה גבוהה בין הדוגמאות בתוך ה-batch. הסיבה לכך היא בעיית ה-catastrophic forgetting, שגורמת לכך שאם רשת מאומנת על דוגמאות דומות מאוד בכל שלב, אך הדוגמאות אינן דומות בין השלבים השונים, הרשת "שוכחת" את מה שלמדה בשלבים הקודמים. לכן, לא ניתן לאמן את הרשת על ידי אינטרקציה של הסוכן עם העולם ולימוד online, אלא יש צורך לשמור transitions שונים שבהם חזה הסוכן בתוך buffer ולדגום ממנו transitions בצורה רנדומלית מאזורים שונים של הסביבה. Buffer זה נקרא ***Experience Replay*** ובד"כ מכיל מאות אלפי transitions שמהם נדגמים ה-batch-ים.

בשלב זה חשוב לציין שאלגוריתם ה-Q-learning אינו האלגוריתם האולטימטיבי לפתרון בעיות של reinforcement learning, אך השימוש ב-Experience Replay מחייב עבודה איתו. הסיבה לכך היא שלאורך האימון, אנחנו מעדכנים את ה-policy של הסוכן אך ממשיכים לצבור דוגמאות לתוך ה-Experience Replay. מתוך כך נובע שב-Experience Replay קיימות דוגמאות רבות שהגיעו מ-policies שונים לגמרי. לכן, כאשר מאמנים את פונקציית ה-Q (רשת הנוירונים), ומתוך כך את ה-policy, אנחנו בעצם מאמנים אותו מתוך הנסיון של policy שונים ולא מנסיון של ה-policy הנוכחי. צורת למידה זו נקראת **off-policy learning** ואלגוריתם Q-learning מאפשר לבצע אותה בצורה מיטבית.

טריק נוסף שנדרש לטובת אימון הרשתות הוא ביצוע של low pass filtering על העדכונים על ידי שימוש בשתי רשתות נפרדות:

* **Online network** - משמשת לביצוע פעולות בפועל ומתעדכנת בכל צעד. את הפרמטרים של רשת זו נסמן ב-.
* **Target network** – ה-policy האמיתי שנלמד על ידי הסוכן. הרשת מתעדכנת מה-online network כל צעדים. את הפרמטרים של רשת זו נסמן ב-

לכן, אלגוריתם העדכון של ה-online network יהיה מעתה לפי השגיאה –

והאלגוריתם לעדכון של ה-target network בכל צעדים יהיה –

**בעיית ה-eploration vs eploitation** – לפי האלגוריתם שתיארנו, הסוכן לומד את ה-policy בצורה חמדנית ולכן יהיה מוטה מאוד לטובת פעולות שביצע בעבר והניבו reward חיובי. מצד אחד, זהו דבר חיובי כיוון שזו הייתה מטרתנו מלכתחילה, אך עם זאת, באופן זה הסוכן יתקבע מהר מאוד על policy מסויים ויחזה רק בתת קבוצה קטנה של המצבים האפשריים בסביבה. באופן זה, יתכן כי הסוכן יפספס פעולות רבות שיכלו להניב reward גבוה יותר.

לכן, היינו רוצים לאפשר לסוכן לחקור את הסביבה בצורה טובה יותר ולקחת פעולות גם אם אין אופטימליות באופק הקרוב לטובת מיקסום של ה-reward בעתיד הרחוק. כלומר, יתכן שפעולה מניבה ערך נמוך יותר מפעולה מכיוון שהסוכן לא ניסה לבצע אותה בעבר ולכן לא קיבל reward עבורה. כדי לפתור את הבעיה הנ"ל משתמשים לרוב באחת משתי השיטות הסטוכסטיות הבאות –

* **-greedy policy** – מגדירים ערך בין 0 ל-1. בכל שלב בוחרים את הפעולה שממקסמת את ה- function בהסתברות או פעולה רנדומלית בהסתברות . בנוסף, נהוג להתחיל מ- גבוה יחסית ולהוריד אותו לאורך האימון.
* **Softmax policy** – בכל שלב, משערכים את ערכי עבור כל הפעולות האפשריות. דוגמים את הפעולה הבאה מתוך התפלגות מולטינומית על פני הערכים –

כאשר הוא פרמטר שנקרא **טמפרטורה** ומגדיר את ההטייה לטובת exploration לעומת exploitation.

במקרים רבים, כמו במשחקי FPS, התהליך הוא לא מרקובי מלא כיוון שלא כל המידע על ה-state הנוכחי קיים ב-observation של הסוכן על הסביבה. מידע זה יכול להיות כיוון תנועה של עצמים, או עצמים שלא ניתן לראות מנקודת המבט של הסוכן. במקרה זה התהליך הואPartially Observable Markov Decision Process. כדי להתמודד עם בעיה זו, ה-state שבו נשתמש יהיה ערימה של מספר תצפיות אחרונות (לדוגמא, 4 תמונות מסך אחרונות).

**Double DQN (DDQN)**

לאורך השנים התגלה בצורה אמפירית כי פונקציית ה- באלגוריתם ה-DQN מבצע שערוך מוטה של ה-expected discounted reward האמיתי. ההנחה באלגוריתם ה-DDQN היא שהסיבה לשערוך המוטה היא שבשימוש באופרטור ה-max בעדכון ערך ה-, משתמשים באותם ערכי גם לבחירת הפעולה הבאה וגם לשערוך של הערך שלה. כדי להתמודד עם הבעיה, מבצעים באלגוריתם ה-DDQN הפרדה בין שני השלבים הנ"ל על ידי עדכון שגיאת ה-TD לערך הבא –

השינוי הנ"ל מביא לשיפור משמעותי בביצועי הסוכן ולמזעור של ה-overestimation של ערך ה-.

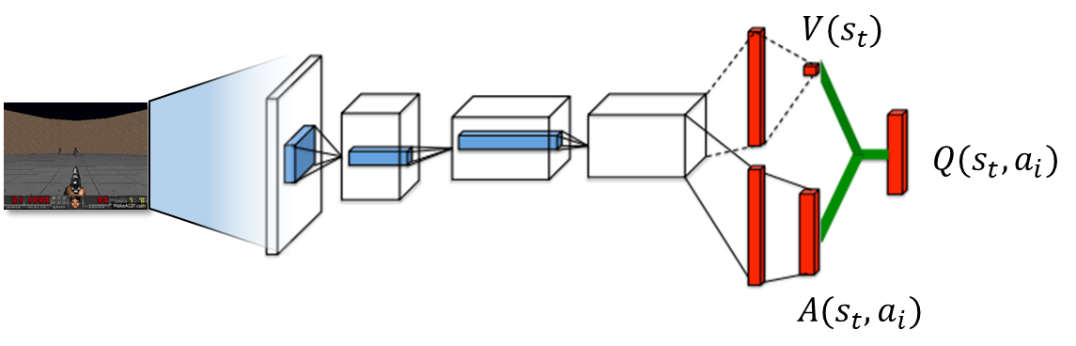
**Dueling Network Architecture**

הבחנה חשובה היא שקיימים מצבים רבים בהם אין יתרון משמעותי של פעולה אחת לעומת פעולה אחרת ולכן אין חשיבות גבוהה לשערוך מדויק של ערך ה- לכל פעולה. עם זאת, באלגוריתמים המבוססים על bootstrap כפי שראינו, יש חשיבות גדולה מאוד לדיוק של ערך המצב עצמו . לכן, נגדיר ארכיטקטורה חדשה שמבצעת הפרדה בין ערכי הפעולות לערכי המצבים. לשם כך נשתמש בפיתוח שהצגנו מוקדם יותר עבור ה-Advantage Function –

ובמקום לשערך את ערך ה- ישירות, נבצע שערוך ביניים של ו-, ולאחר מכן נחשב את לפי הנוסחה –

חשוב לשים לב שכיוון שאנחנו עדיין מבצעים עדכונים לכיוון ערכי ה-, אין הבטחה ששערוך ערכי ה- יהיה שערוך נכון. לכן נדרוש שעבור הפעולה הנבחרת ה-Advantage יהיה 0. כלומר –

מבנה הרשת החדשה יהיה מהצורה הבאה -



**תמונה 1.5: ארכיטקטורת ה-Dueling Network**

**Prioritized Experience Replay**

כפי שראינו ה-Experience Replay משמש בכדי לדגום transitions לאימון כך שלא תהיה ביניהם קורלציה. עם זאת, יש חשיבות גדולה לעדיפות שנותנים לדגימת כל אחד מה-transitions. באופן אידיאלי, היינו רוצים לדגום מה-Experience Replay את ה-transitions שהם הסוכן יוכל ללמוד הכי הרבה. כדי להעריך את המדד הזה עבור כל transition, נוכל להשתמש ב-TD error שראינו מוקדם יותר, שמבטא כמה הרשת הייתה רחוקה מלהעריך את המצב נכון. לכן נגדיר את ההסתברות לדגום transition מסויים באינדקס i על ידי –

כאשר מבטא כמה אנחנו מתעדפים לפי הקריטריון, כך שכאשר נקבל הסתברות אחידה לדגימת כל אחד מה-transitions. הוספת תעדוף לאימון גורמת ל-bias, בגלל שהוא משנה את הסתברות הדגימה בצורה לא מבוקרת. ניתן לתקן את ההטייה על ידי שימוש ב-Importance Sampling באימון המשקולות. כלומר נמשקל את ה- של כל transition לפי –

כאשר הוא גודל ה-Experience Replay.

**Skills**

עבור סביבות מורכבות יותר, פעולה לפי policy יחיד אינה מספיקה. לעומת זאת, קבלת החלטות אנושית מערבת בחירה תדירה בין רצפי פעולות מתוך יכולות שונות. מידול יכולות אלה באלגוריתמים דורש אבסטרקציה היררכית של פעולות על פני ציר הזמן. לשם כך מגדירים skill (נקרא גם option) הוא מבנה בקרה שמוגדר כשלישייה כך ש-

* – קבוצת מצבים שבהם ה-skill יכול להתחיל
* – הוא ה-policy הפנימי ל-skill
* – הוא תנאי עצירה על פני המצבים שבהם יסתיים ה-skill

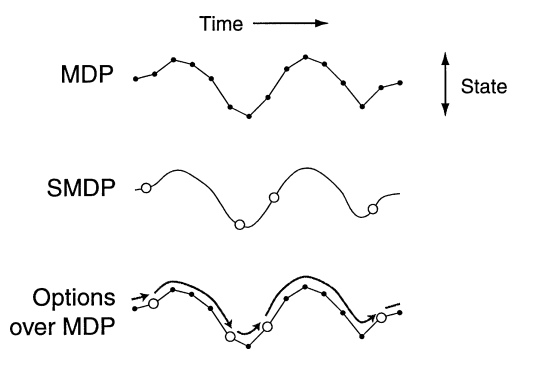
במידה ו-skill נבחר החל ממצב מסויים , הפעולות נבחרות בהתאם ל- עד שה-skill מסתיים לפי .

**Semi-Markov Decision Process (SMDP)**

כדי למדל את השימוש ב-skills לתוך ה-Markov Decision Process, נשתמש בהכללה שלו שנקראת Semi-Markov Decision Process, המחליפה פעולות ב-skills. SMDP מוגדר כחמישייה כך ש-

* – קבוצת מצבים
* – קבוצת ה-skills האפשריים
* – הסתברויות המעבר בין מצבים
* - פונקציית reward
* – ערך ה-discount

הפתרון ל-SMDP הוא **skill policy** – כך ש- הן הסתברויות על פני ה-skills.

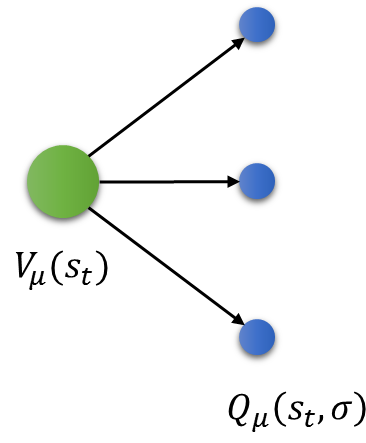


**תמונה 1.6: MDP, SMDP ושילוב של Skills (Options) על גבי התהליך**

**Option Interrupt**

אחת האופציות האפשריות לעצירה של skill (option) היא על ידי בחינת התועלת של להמשיך בביצוע ה-policy של ה-skill בכל מצב על פני החלפת ה-skill. תחת מודל ה-SMDP, פונקציות ה-value מתחלפות בפונקציות value על פני ה-skills - ו-. במידה שאנחנו במהלך ביצוע skill , ומתקיים –

המשמעות היא שקיים skill המביא את ערך המצב לערך גבוה יותר מ-. לכן במקרה זה, עדיף לעצור את יישום ה-policy של ולבחור skill אחר על פניו.



**תמונה 1.7: Heatmap הפלט של רשת זיהוי האויבים. אזורים מושחרים הינם אזורים שהרשת זיהתה הסתברות נמוכה להמצאות אויבים בהם.**

3. **הפתרון**

משחקי מחשב הינם סביבת סימולציה מעולה לאימון ובחינת סוכנים באופן מבוקר. עם זאת, עד כה מרבית הסוכנים שפותחו ניסו לפתור משחקים פשוטים יחסית, כדוגמת משחקי Atari שדורשים סט כלים ויכולות מצומצמים מהסוכן על מנת לפתור אותם. בפרויקט זה, המטרה המרכזית היתה להעלות את רמת האתגר לעבודה במשחקים מורכבים יותר, המדמים סביבה תלת מימדית ומציאותית יותר, ודורשים שילוב של מספר יכולות על מנת לפתור אותם.

**סביבת הפיתוח**

כסביבת סימולציה בחרנו לעבוד תחילה עם פרויקט Unreal Engine ומשחק דמו First Person Shooter. לאחרונה שולבה סביבת Torch אל תוך המנוע, מה שאפשר מימוש של אלגוריתמי למידה עמוקה אל תוך המשחק. במשחק זה, השחקן רואה את העולם מנקודת מבט של גוף ראשון. המטרה במשחק היא השמדה של כמה שיותר רובוטים. המשחק דורש 3 יכולות מרכזיות מהשחקן –

1. יכולת ניווט במבוך, מציאת רובוטים ואיסוף קופסאות חימוש או חיים
2. כיוון וירי אל עבר רובוטים
3. התחמקות מרובוטים שיורים אל עבר השחקן



**תמונה 3.1: סביבת Unreal Engine ומשחק Shooter Demo בו השתמשנו**

בסביבה זו העבודה הינה בשפת Lua תוך שימוש בספריית Torch המאפשרת אימון מודלים על GPU. הבעיות העיקריות בסביבה הן:

1. המנוע דורש משאבים רבים ולכן קצב האימון נמוך מאוד.
2. כלי הדיבוג של Torch הם דלים מאוד.
3. הסביבה לא פותחה לטובת אימון סוכנים ולכן לא מספקת כלים המתאימים לכך באופן פשוט כמו ה-reward, מצב השחקן ושלבים המתאימים לאימון הדרגתי.

בהמשך, עברנו לעבוד בסביבה יותר יציבה ומבוססת של משחק Doom. היתרון בסביבה זו היא שהיא אינה דורשת משאבים רבים מהמחשב, מה שמאפשר סביבה יציבה יותר ואימון מהיר יותר. בנוסף, הסביבה פותחה לשם אימון אלגוריתמי למידה ולכן כל המאפיינים הנדרשים עבור למידה, כמו ה-reward, סט הפעולות האפשריות ותמונת המסך מסופקים ישירות על ידי הסביבה. העבודה בסביבה זו מבוססת על שפת Python ומאפשרת שימוש בספריות כמו TensorFlow ו-Keras, שמקלות על הכתיבה, האימון והדיבוג של רשתות נוירונים.



**תמונה 3.2: סביבת ViZDoom**

**מודל ראשון**

בשלב הראשון, עבדנו על פיתוח של סוכן על שמכיל מספר תת סוכנים, בהם כל אחד עובד לפי skill מאוד ספציפי אך כולם פועלים לטובת מטרה משותפת. תחת מודל זה, חילקנו את הסוכן לשלושה תת-סוכנים –

1. סוכן השולט על המבט של השחקן
2. סוכן השולט על הירי
3. סוכן השולט על ניווט במשחק

כל סוכן מאומן בנפרד בשלב הדורש את סט הכישורים הספציפי לו. לדוגמא, הסוכן ששולט על הירי אומן בשלב המדמה מטווח, בו הוא צריך לכוון אל עבר האויבים ולירות.

בסביבה הראשונה (Unreal Engine) הוספנו שני מודלים כדי להקל על הלמידה:

* מודל לביצוע Parsing של הניקוד מתמונת המסך
* מודל לזיהוי של אויבים

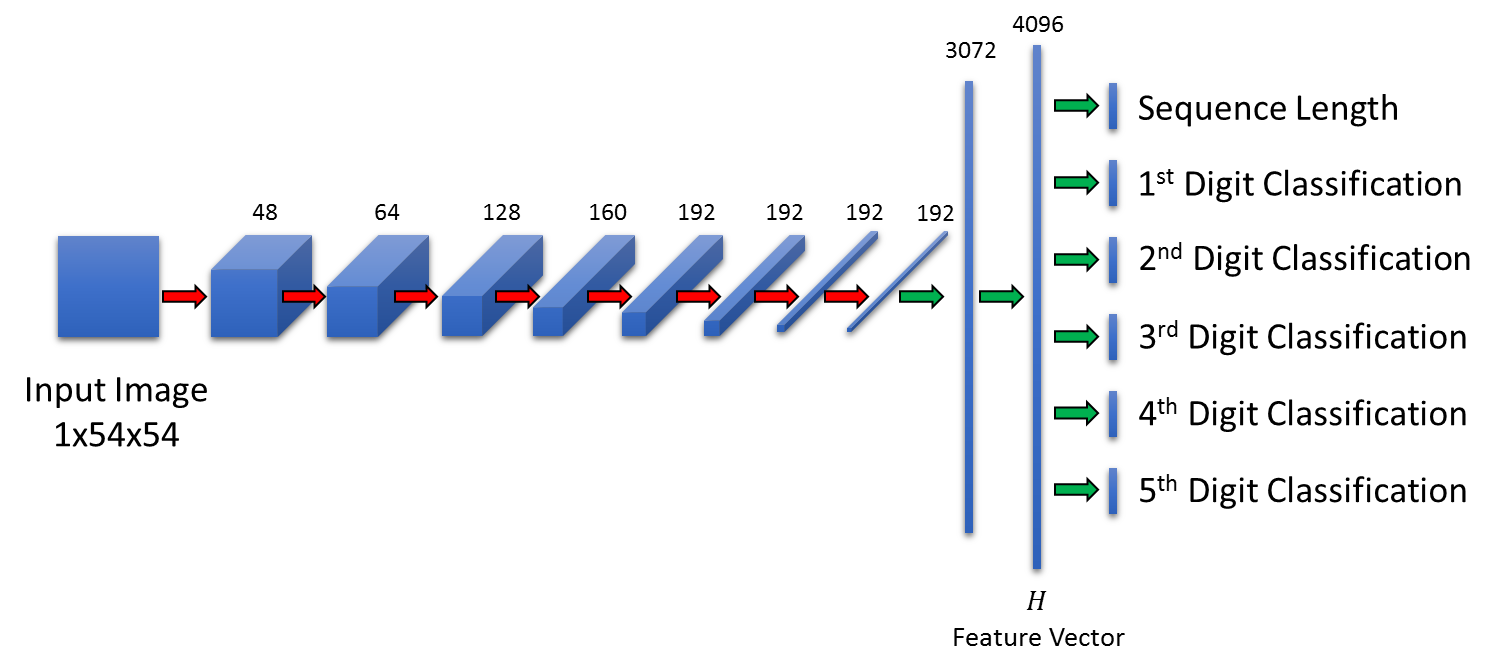
**מודל לביצוע Parsing של הניקוד**

כדי לבצע Parsing של הניקוד ביצענו אימון של מודל מוכר עבור ה-dataset - Street View House Numbers (SVHN). Dataset זה מכיל כ-50 אלף תצלומים של מספרי בתים רבים מרחבי העולם, שנלקחו מתוך תצלומי Street View של גוגל. המידע הנ"ל מתאים מאוד לביצוע Parsing של הניקוד כיוון שהוא מכיל מספרים של עד 5 ספרות על גבי רקעים שונים ומגוונים.



**תמונה 3.3: דוגמאות למספרים ממאגר SVHN ומספרים המייצגים ניקוד או חימוש במשחק של Unreal Engine**

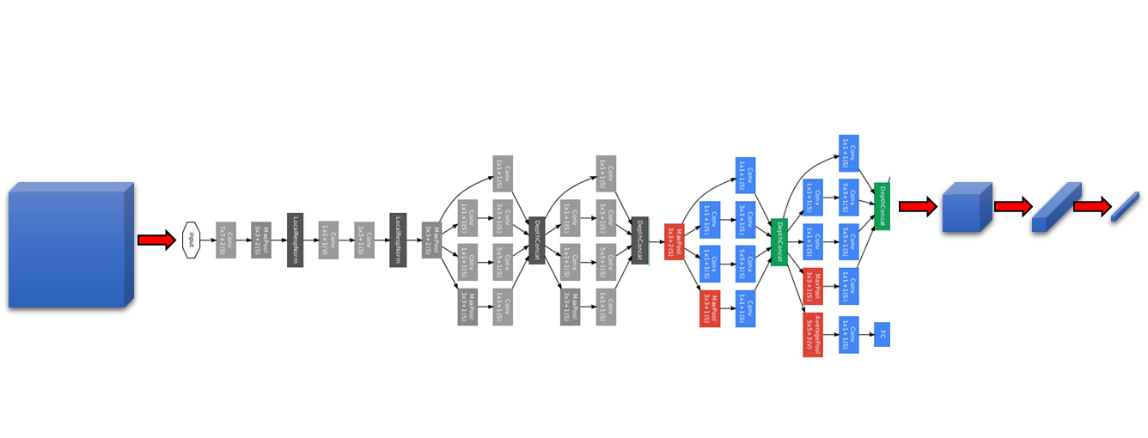
המודל שבו השתמשנו משתמש בתמונת המספר כקלט, וכפלט מוציא הסתברויות על פני 5 ספרות שונות והסתברות על אורך סדרת הספרות. המודל מגיע ל95.4% דיוק על ה-Dataset של SVHN ולכ-97% דיוק על מספרים מהמשחק.



**תמונה 3.4: מבנה רשת ה-Parsing של המספרים.**

**מודל לזיהוי של אויבים**

המטרה של מודל זה היא לספק ידע קודם עבור הרשתות של הסוכנים השונים. הקלט של המודל היא תמונת מסך של המשחק, והפלט הוא Saliency Map שמציג אזורים בהם ההסתברות להמצאות אויבים היא גבוהה. המודל מבוסס על רשת GoogLeNet אשר אומנה לפני כן על Dataset בשם ImageNet אשר מכיל מאות אלפי תמונות מכ-1000 מחלקות שונות של עצמים. כדי להתאים את הרשת לזיהוי אויבים, השתמשנו ב-Transfer Learning על ידי הסרת השכבות האחרונות של הרשת, החלפתם בשכבות קונבולוציה הניתנות לאימון והקפאת שאר המשקולות של הרשת. שיטה זו אפשרה ללמוד את זיהוי האויבים מתוך כ-100 דוגמאות חיוביות ו-100 דוגמאות שליליות בלבד מתוך המשחק.



**תמונה 3.5: מבנה רשת זיהוי האויבים. הרשת מורכבת מרשת GoogLeNet עם תוספת שכבות נלמדות.**

האימון בוצע באמצעות חלונות של 128x128 מתוך המשחק. החלפת השכבות האחרונות של הרשת בשכבות קונבולוציה מאפשרת הזנה של הרשת בחלון גדול יותר (תמונת מסך מלאה) וביצוע הרצה של חלון בגודל 128x128 על פני תמונת המסך המלאה בצורה יעילה על מנת לקבל הסתברות עבור כל אזור במסך.



**תמונה 3.6: Heatmap הפלט של רשת זיהוי האויבים. אזורים מושחרים הינם אזורים שהרשת זיהתה הסתברות נמוכה להמצאות אויבים בהם.**

**אימון הסוכנים**

בתחילה ניסינו לבצע את אימון הסוכנים בתוך סביבת Unreal Engine, אך גילינו כי הסביבה כבדה מידי ולא מאפשרת ביצוע אימון יעיל של המודלים. על מנת להגיע לתוצאות ראשוניות נדרשו שבועות של אימון, מה שלא אפשר התקדמות מספקת. לאחר מכן, עברנו לבצע אימון בסביבת Doom, מה שאפשר לאמן את כל אחד מהסוכנים במספר מועט של שעות. נתאר עתה את אימון הסוכנים בסביבת Doom.

**סוכן הירי** – את סוכן זה אימנו בשני שלבים המדמים מטווח בהם השליטה של השחקן היא בתנועה ימינה שמאלה וירי בלבד. בשלב הראשון מופיעה מול השחקן דמות של אויב שאינו מזיק ועליו לחסל אותו במספר הנמוך ביותר של כדורים. בשלב השני מופיעות מול השחקן דמויות רבות של אויבים שהינן מזיקות ועליו לחסל כמות גדולה ככל האפשר של אויבים לפני שנגמרים לו הכדורים או שהוא מת. המטרה בשימוש בשלב זה היא אימון הסוכן לזיהוי אויבים וכיוון הנשק עליהם.



**תמונה 3.7: שני השלבים אשר שימשו לאימון סוכן הירי. בצד ימין - שלב DefendTheCenter. בצד שמאל - שלב Basic**

**סוכן הניווט** – את סוכן זה אימנו בשלב המכיל מבוך ובו מפוזרות קופסאות של חיים. הסביבה של המבוך הינה רעילה ולכן החיים של השחקן דועכים ככל שעובר הזמן. המטרה של השחקן היא לנווט במבוך ולאסוף קופסאות כך שיצבור זמן משחק גדול ככל הניתן. המטרה בשימוש בשלב זה היא אימון הסוכן לניווט ואיסוף קופסאות.



**תמונה 3.8: השלב ששימש לאימון סוכן הניווט - HealthGathering**

**הסוכן המשולב**

הסוכנים המאומנים שולבו יחד לסוכן יחיד שבו כל אחד מתת הסוכנים שולט על פעולות אחרות שלו. הסוכן נבחן על שלב מורכב יותר בו הוא נמצא במבוך עם הרבה אויבים מזיקים והרבה קופסאות של חימוש וחיים. המטרה של השחקן היא לשרוד כמה שיותר זמן במשחק ולחסל כמה שיותר אויבים.

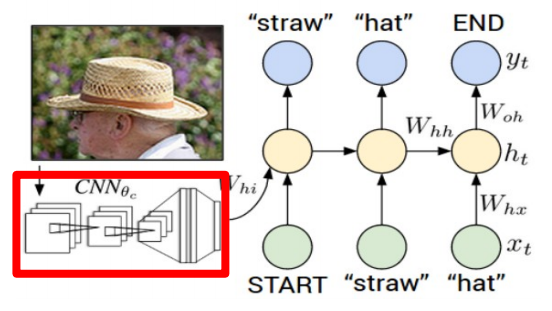


**תמונה 3.9: השלב ששימש לבחינת הסוכן המשולב - DeathMatch**

**מודל שני**

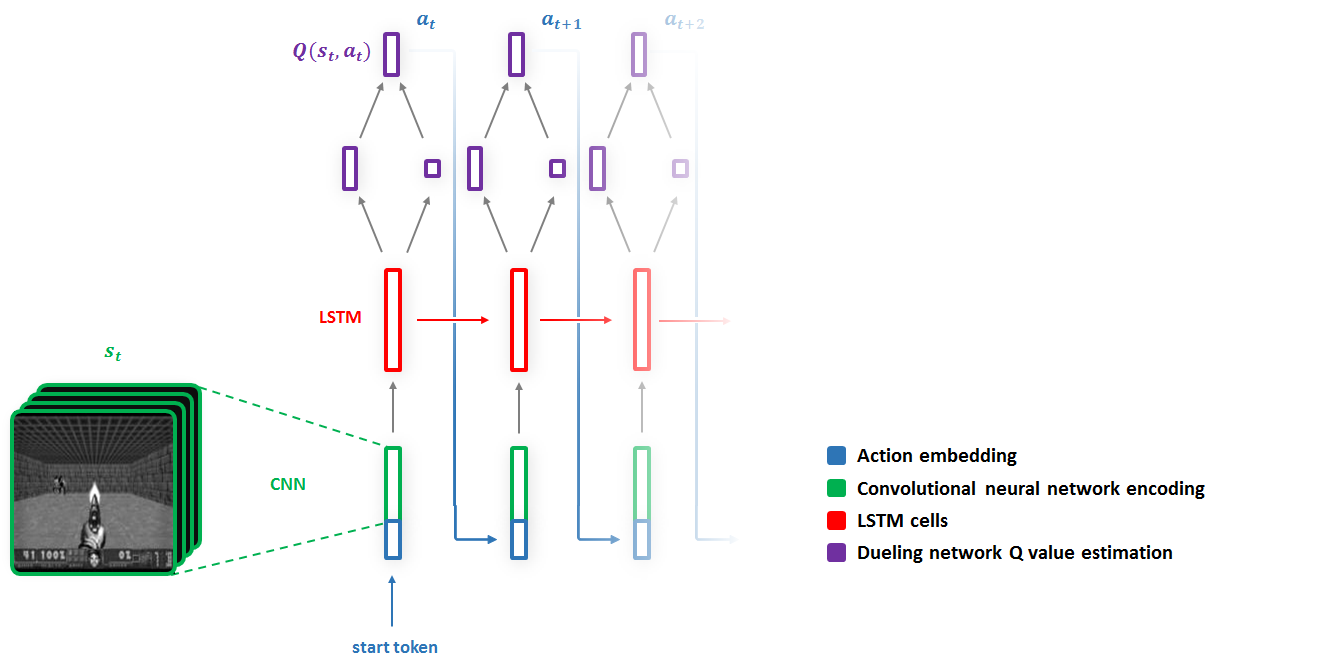
המודל השני אותו ניסינו הינו מודל ללמידה אוטומטית של skills. במקום לדרוש אימון מוקדם של כל אחד מהסוכנים בסביבה ייעודית עבור הכישור של סוכן זה, הסוכן מאומן ישירות על הסביבה המאתגרת אך לומד את כל אחד מהכישורים מתוך ניסוי וטעייה. כדי לממש אלגוריתם מסוג זה, ניסינו להשתמש ב-Recurrent Neural Network שילמד לחזות מספר פעולות קדימה במקום פעולה אחת. בכל שלב, הסוכן יראה את המצב הנוכחי ויצטרך לחזות רצף של פעולות באורך משתנה אותו יבצע השחקן. ההנחה היא שרצפי הפעולות האופטימליים ייצגו policy של כישורים נדרשים עבור הסוכן כמו כיוון וירי או ניווט אל עבר מטרה.

הרשת מבוססת על השוואה של למידת policy במשחק ללמידת משפטים בעיבוד שפה טבעית. במקרה זה ניתן להסתכל על כל אחת מהפעולות שנחזות כמילה במשפט. מתוך כך ניתן להשתמש במודלים של image captioning שהוצגו לאחרונה. רשתות אלו מקבלות תמונה כקלט וחוזות מילה אחרי מילה משפט המתאר את התמונה. המודלים משתמשים בשכבת LSTM (Long-Short Term Memory) המחזיקה מצב פנימי שמתעדכן בכל הרצה של השכבה, אך שומר ייצוג אבסטרקטי של פעולת השכבה עד כה.



**תמונה 3.10: מבנה רשת ה-image captioning**

במקרה של למידת policy, הרשת מקבלת את המצב הנוכחי שהינו תמונת מסך מהמשחק, וחוזה בכל שלב פעולה בודדת עד לחיזוי של end token שהינה פעולה מיוחדת שמסיימת את רצף הפעולות. כדי ללמוד את תנאי העצירה השתמשנו בכלל ה-Option Interrupt שהוצג בפרק ברקע. בנוסף, עבודה עם episodes שלמים של משחקים אפשרה החלפה של אלגוריתם העדכון המשתמש בעדכון צעד אחד קדימה (1-step Q learning) לעבודה עם אלגוריתם עדכון של צעדים קדימה (n-step Q learning), שאמור להיות מיודע יותר ולכן מדויק יותר.



**תמונה 3.11: מבנה הרשת לחיזוי רצפי פעולות**

**4. תוצאות**

תוצאות ויזואליות של כל אחד מהמודלים ניתן לראות בקישורים הבאים:

* מודל ל-Parsing של מספרים –

<https://www.youtube.com/watch?v=dvWzVPVcTjw&t=46s>

* מודל לזיהוי אויבים -

<https://www.youtube.com/watch?v=wpotGS8CiLI>

* סוכן הירי –

<https://www.youtube.com/watch?v=CFUoUhWKee0>

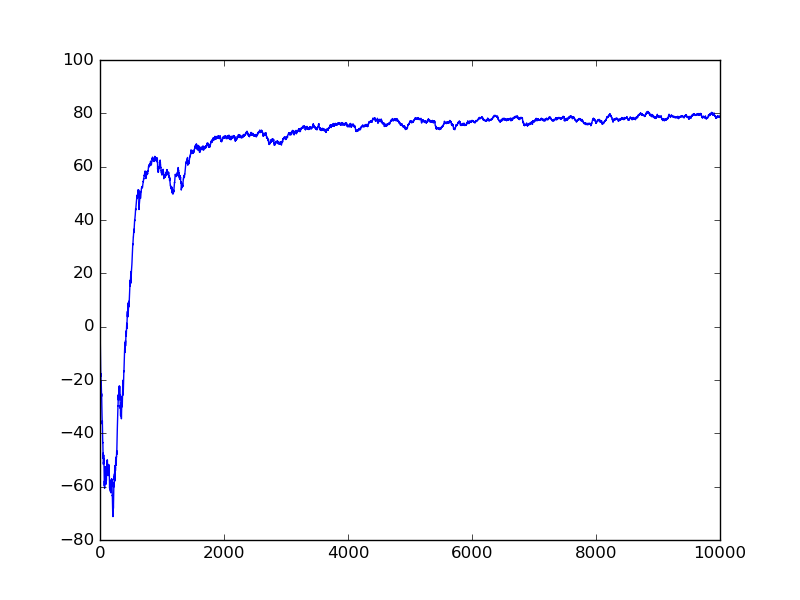
<https://www.youtube.com/watch?v=vnFHonjWoHE>

* סוכן הניווט –

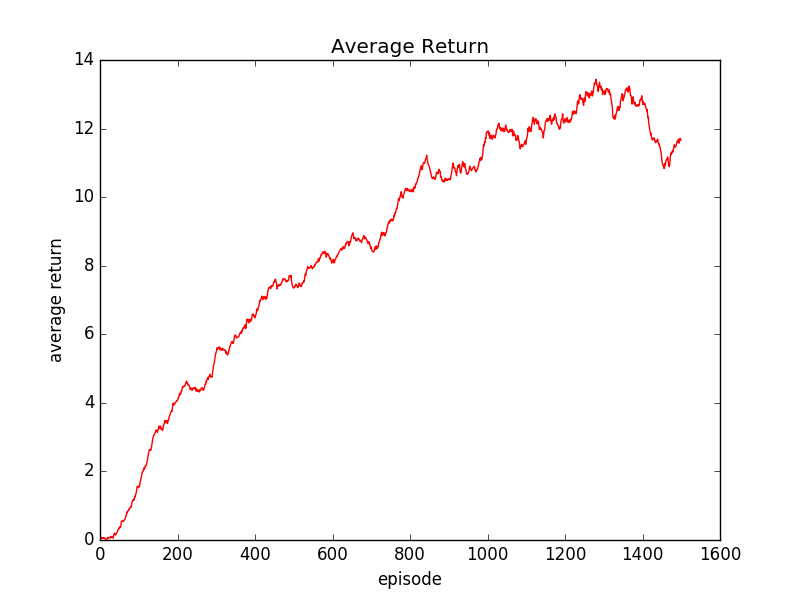
<https://www.youtube.com/watch?v=Bvery9W-WQI>

עבור תת הסוכנים, הסוכנים הגיעו קרוב לניקוד המקסימלי האפשרי בשלבים אלו. בכל הגרפים הבאים, ציר x מייצג את מספר איטרציות האימון וציר מייצג את הניקוד של השחקן

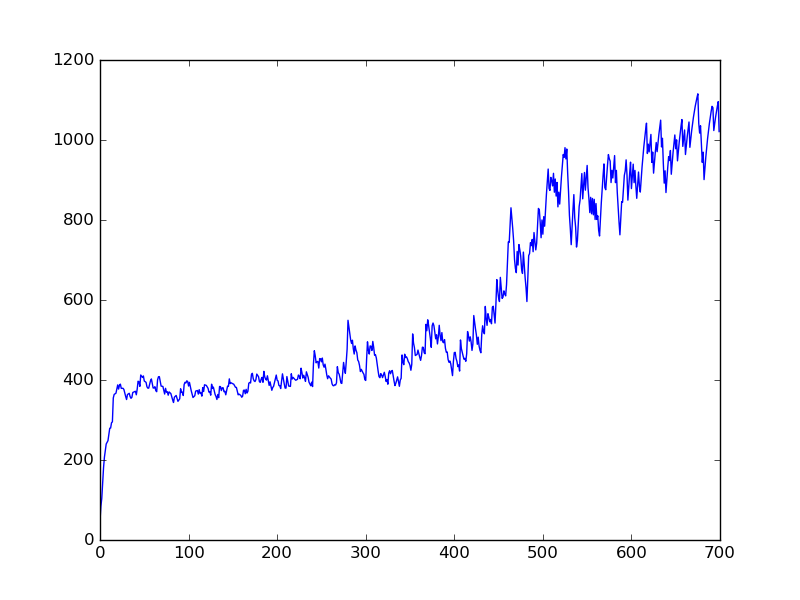
**שלב המטווח (Basic)** – ניקוד מקסימלי אפשרי – 100



**שלב המטווח (DefendTheCenter)** – ניקוד מקסימלי אפשרי – 20



**שלב הניווט (HealthGathering)** – ניקוד מקסימלי אפשרי 1200



**5. סיכום ומסקנות**

בפרויקט זה הצגנו דרך לשילוב של מספר סוכנים להשגת מטרה משותפת כסוכן-על יחיד. השילוב שביצענו היה ידני יחסית ודרש תכנון מוקדם של חלוקת העבודה בין הסוכנים ואימון נפרד שלהם. על ידי אימון מוקדם של כל אחד מהסוכנים הצלחנו להגיע לתוצאות טובות מעט לעומת אימון סוכן יחיד על המשחק המורכב.

באופן אידאלי, היינו רוצים שחלוקת העבודה בין תת הסוכנים תילמד בצורה אוטומטית. לכן, פיתחנו את המודל השני אשר ביצע חיזוי של מספר פעולות קדימה. עם זאת, המודל השני לא הניב תוצאות טובות יותר מאשר אימון של סוכן DQN רגיל.

מסקנה חשובה מפרויקט זה היא חשיבות בחירת הסביבה הנכונה לעבודה. במהלך הפרויקט עבדנו תחילה בסביבת Unreal Engine אשר דרשה התעסקות יתרה במנגנונים של הסביבה וגזלה מזמן פיתוח אלגוריתמי הלמידה עצמם. בנוסף, הלמידה עצמה הייתה איטית ודרשה זמן רב להגעה לתוצאות מספקות. במעבר לסביבת ViZDoom בעיות אלו נפתרו, מה שאפשר ניסוי של הרבה מאוד קונפיגורציות שונות בזמנים קצרים והתמקדות באלגוריתמיקה עצמה.

**6. ביבליוגרפיה**

Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. **Vizdoom: A doom-based AI research platform for visual reinforcement learning**. CoRR, abs/1605.02097, 2016. URL http://arxiv.org/abs/1605.02097.

Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. **Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation**. CoRR, abs/1604.06057, 2016a. URL <http://arxiv.org/abs/1604.06057>.

Tejas D. Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J. Gershman. **Deep successor reinforcement learning**. CoRR, abs/1606.02396, 2016b. URL http://arxiv.org/abs/ 1606.02396.

Guillaume Lample and Devendra Singh Chaplot. **Playing FPS games with deep reinforcement learning**. CoRR, abs/1609.05521, 2016. URL <http://arxiv.org/abs/1609.05521>.

Adam Lerer, Sam Gross, and Rob Fergus. **Learning physical intuition of block towers by example**. CoRR, abs/1603.01312, 2016. URL <http://arxiv.org/abs/1603.01312>.

Amy McGovern and Andrew G. Barto. **Automatic discovery of subgoals in reinforcement learning using diverse density**. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001, pp. 361–368, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. **Playing atari with deep reinforcement learning**. CoRR, abs/1312.5602, 2013. URL <http://arxiv.org/abs/1312.5602>.

Volodymyr Mnih, Adria Puigdom ` enech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim ` Harley, David Silver, and Koray Kavukcuoglu. **Asynchronous methods for deep reinforcement learning**. CoRR, abs/1602.01783, 2016. URL <http://arxiv.org/abs/1602.01783>.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. **Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning**. Artif. Intell., 112(1-2):181– 211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL http://dx.doi.org/10.1016/ S0004-3702(99)00052-1.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. **A deep hierarchical approach to lifelong learning in minecraft**. CoRR, abs/1604.07255, 2016. URL <http://arxiv.org/abs/1604.07255>.

Hado van Hasselt, Arthur Guez, and David Silver. **Deep reinforcement learning with double qlearning**. CoRR, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461. 5 Under review as a conference paper at ICLR 2017

Alexander Vezhnevets, Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, and Koray Kavukcuoglu. **Strategic attentive writer for learning macro-actions**. CoRR, abs/1606.04695, 2016. URL <http://arxiv.org/abs/1606.04695>.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. **Show and tell: A neural image caption generator**. CoRR, abs/1411.4555, 2014. URL http://arxiv.org/abs/1411. 4555.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. **Dueling network architectures for deep reinforcement learning**. In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19- 24, 2016, pp. 1995–2003, 2016. URL http://jmlr.org/proceedings/papers/v48/ wangf16.html