

פרויקט מסכם לתואר בוגר במדעים (B.Sc)
במתמטיקה שימושית

מערכת בינה מלאכותית לפתרון המשחק 2048 בעזרת
אלגוריתם אקספקטימקס ואופטימיזציה הנחיל

יפתח בן זקן
ת.ז. 021546957

2048 AI Solver using Expectimax Algorithm &
Particle Swarm Optimization

Iftah Ben Zaken

**פרויקט מסכם לתואר בוגר במדעים (B.Sc)
במתמטיקה שימושית**

**מערכת בינה מלאכותית לפתרון המשחק 2048 בעזרת
אלגוריתם אקספקטימקס ואופטימיזציה הנחיל**

יפתח בן זקן
ת.ז. 021546957

**2048 AI Solver using Expectimax Algorithm &
Particle Swarm Optimization**

Iftah Ben Zaken

Advisor:

Dr. Gabriel Ben-Simon

מנחה:

ד"ר גבריאל בן סימון

Karmiel

כרמיאל

2015

תקציר

פרוייקט זה עוסק מחד במשחקים ואיך לפתור אותם, ומאידך בניתוח מתמטי של המשחק 2048 וכתיבת אלגוריתם בינה מלאכותית שיפתור אותו.

תחילה נסקור את נושא המשחקים והבינה המלאכותית. נדבר על ההיסטוריה של פתרון ממוחשב של משחקים ועל סוגי משחקים שונים. בנוסף, נדבר על עצי משחק ועל אלגוריתם חשוב מתורת המשחקים בשם מינימקס (Minimax). נגדיר גם מהי פונקציית הערכה היוריסטית.

בשלב השני נתאר ונמדל את המשחק 2048. נסביר את החוקים ואת האסטרטגיות השונות לנצח בו. ננסח היוריסטיקות שיעזרו לנו להעריך לוח משחק ונדבר על האלמנט הסטוכסטי במשחק, שבגללו עלינו להשתמש באלגוריתם אקספקטימקס (Expectimax) במקום במינימקס. לבסוף ננסח את האלגוריתם לפתרון המשחק בצורת נוסחה מתמטית רקורסיבית.

בשלב השלישי של הפרוייקט נציג שיטות אופטימיזציה מטה-היוריסטיות בצורה כללית ואת אופטימיזציית הנחיל (Particle Swarm Optimization) בפרט. נעטוף את האלגוריתם שניסחנו בשלב הראשון בתהליך אופטימיזציה אשר יאפשר לו להשיג תוצאות גבוהות יותר. לסיום, נדון בתוצאות ונסכם.

1.	הקדמה	3
2.	חלק א' – רקע תאורטי	4
2.1	משחקים ובינה מלאכותית	4
2.2	עץ משחק	5
2.3	אלגוריתם מינימקס	6
2.4	פונקציית הערכה היוריסטית	9
3.	חלק ב' – בניית המודל המתמטי	10
3.1	המשחק 2048	10
3.2	האלמנט הסטוכסטי	12
3.3	אלגוריתם אקספקטימקס	13
3.4	היוריסטיקות למשחק 2048	15
3.5	ניסוח מתמטי של האלגוריתם לפתרון המשחק 2048	18
4.	חלק ג' – תהליך האופטימיזציה	19
4.1	אופטימיזציה מטה-היוריסטית (Metaheuristic Optimization)	19
4.2	אופטימיזציית הנחיל (Particle Swarm Optimization)	20
4.3	מהלך הניסוי	22
4.4	תוצאות	24
5.	סיכום	25
6.	ביבליוגרפיה	26
7.	נספח – הוראות התקנה והפעלה	26

1. הקדמה

כשהיה עלי לבחור נושא לעבודת הגמר, ניסיתי למצוא רעיון לפרוייקט קצת שונה, משהו מסקרן ומלהיב. היה לי גם חשוב למצוא רעיון שישלב בין שני התחומים האהובים עלי, מתמטיקה ומדעי המחשב. באותו זמן ישב בסלון אחיין שלי והוא שיחק במשחק 2048 בסמארטפון, נאבק ללא הצלחה להגיע לאריח 2,048. בזמן שניסיתי ללמד אותו שיטות יעילות לשחק את המשחק עלה במוחי הרעיון – אכתוב תוכנה שתפתור את המשחק.

תחום המשחקים תמיד סיקרן אותי והיה לי מוכר ברמה התאורטית מקורסים שלמדתי כגון תורת המשחקים, בינה מלאכותית, תורת הסתברות ותהליכים אקראיים. לעומת זאת, מעולם לא הזדמן לי לבחון את הידע בצורה מעשית.

בשבילי, מטרת הפרוייקט היא להשתמש בידע ובכלים שרכשתי במהלך התואר, לנתח את המשחק בצורה מתמטית ולתכנן אלגוריתם שיפתור את המשחק בצורה מיטבית.

היעד שהצבתי בתחילת העבודה על הפרוייקט הוא להגיע לאריחים בסדר גודל של 4,096-8,192 שזה קצת יותר טוב משחקן אנושי ממוצע.

2. חלק א' – רקע תאורטי

2.1 משחקים ובינה מלאכותית

תורת המשחקים היא ענף במתמטיקה וכלכלה העוסק באסטרטגיות וקבלת החלטות. ליתר דיוק, בתורת המשחקים לומדים ומפתחים מודלים מתמטיים של שיתוף פעולה ועימות בין משתתפים רציונליים ואינטלגנטיים אשר כל אחד מהם מעוניין לנצח/להרוויח. לתורת המשחקים שימושים רבים בכלכלה, מדע המדינה, פסיכולוגיה, לוגיקה, ביולוגיה ומדעי המחשב. בעבודה זו, נדבר על תורת המשחקים בהקשר של בינה מלאכותית – פתרון משחקים בעזרת המחשב.

כבר בימיו הראשונים של המחשב, חוקרים ראו את הפוטנציאל הגלום בו לפתרון בעיות מתמטיות ומשחקים ולשם כך השתמשו באלגוריתמים מתורת המשחקים. בשנת 1951 נכתבה התוכנה הראשונה שמשחקת שחמט ע"י לא אחר מאשר אלן טיורינג, ממציא מכונת טיורינג, אשר נחשב בעיני רבים לאבי מדעי המחשב והבינה המלאכותית. טיורינג התבסס על מאמר שהתפרסם שנה קודם ע"י קלוד שאנון, אבי תורת האינפורמציה. במשך כמעט יובל לא הצליחו תוכנות המחשב לנצח יריב אנושי משמעותי, דבר המראה את גודל האתגר בבניית תוכנה לפתרון המשחק.

חקירה של משחק מורכב מתאפשרת על ידי הפשטתו לאחד מכמה מודלים כלליים, הניתנים לניתוח מתמטי. המטרה היא "לפתור" את המשחק, כלומר, לזהות בו את דרכי הפעולה הצפויות של השחקנים השונים וכך ליעץ על דרכי פעולה מומלצות.

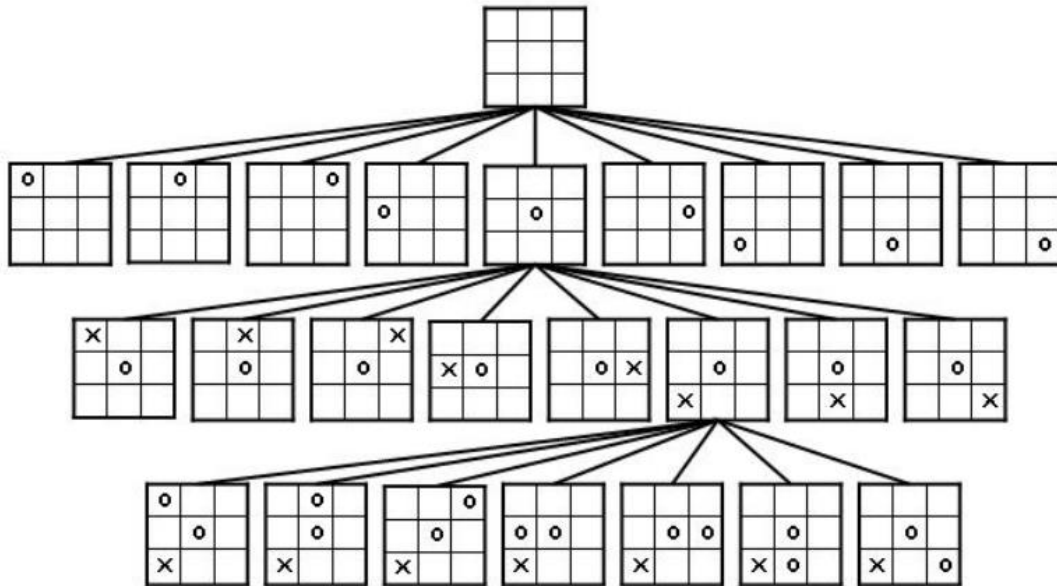
מאז ועד היום חלה התפתחות משמעותית בכוח המחשוב, בשיטות המחקר ובאלגוריתמים. בשנת 1997 ניצח מחשב המכונה "כחול עמוק" את אלוף העולם דאז גארי קספרוב. באותו משחק מפורסם, "כחול עמוק" סרק כ-200 מיליון מצבים בשניה והשתמש בטכניקות מתקדמות של בינה מלאכותית שאיפשרו לו (במקרים מסוימים) לצפות 40 מהלכים קדימה. תוכנות שחמט מודרניות מתפקדות אפילו טוב יותר.

למשחקים שונים ישנה רמת מורכבות שונה. כך למשל ישנם משחקים פחות מורכבים משחמט, שבהם מספר מהלכים קטן יחסית כמו איקס-עיגול ודמקה. משחקים אלו נחשבים "פתורים" במובן שכל המהלכים האפשריים במשחק יכולים להיות מחושבים מראש בזמן סביר. כאשר משחק הוא פתור, קיים אלגוריתם שמבטיח ניצחון או תיקו לשחקן מול כל מהלך אפשרי של היריב. בעזרת האלגוריתם ניתן לבנות תכנת מחשב שלעולם לא תפסיד במשחק.

היכולת "לפתור" משחק תלויה במידה רבה בכמות המהלכים האפשריים בכל תור. בשחמט למשל, ישנם בממוצע 35 מהלכים אפשריים בכל תור ואורך משחק ממוצע של כ-80 תורים. מספר המצבים האפשריים במשחק מוערך בכ- 10^{123} מצבים שונים של הלוח. לשם השוואה, כיום מעריכים שמספר האטומים ביקום הנראה הוא בתחום $4 \times 10^{81} - 4 \times 10^{79}$. כמובן שלא קיים כרגע מחשב המסוגל להתמודד עם כמות כזו של מידע. בהמשך אציג שיטות שונות להתמודדות עם בעיה זו.

2.2 עץ משחק

בתורת המשחקים, עץ משחק הוא גרף מכוון שצמתיו מייצגים מצבי משחק והקשתות הינם המהלכים המובילים למצבי המשחק הנ"ל. עץ משחק שלם הוא עץ שהשורש שלו מייצג את המצב ההתחלתי וממנו יוצאים כל המהלכים האפשריים בצורה של עץ. מספר העלים בעץ משחק הוא כמספר הדרכים השונות שהמשחק יכול להיות משוחק. לדוגמה, בעץ המשחק המלא של איקס-עיגול ישנם 255,168 עלים.



איור 1 – עץ משחק חלקי של המשחק איקס-עיגול

לעצי משחק יש ייצוג נכבד בתחום האלגוריתמיקה והבינה המלאכותית. זוהי דרך נוחה למדל תהליך קבלת החלטות במשחק כך שבכל תור מבוצע חיפוש על עץ המשחק ונבחר המהלך הטוב ביותר. שיטה אחת כזו בה מטיילים בעץ ולוקחים החלטות בהתאם למצב המשחק נקראת אלגוריתם מינימקס (Minimax).

2.3 אלגוריתם מינימקס

זהו אלגוריתם רקורסיבי שמטרתו להחליט מהו המהלך הבא שעל השחקן לבצע בהינתן מצב משחק מסוים. התאוריה שמאחורי האלגוריתם פורסמה בשנת 1928 ע"י ג'ון פון נוימן – אבי תורת המשחקים. הוא צוטט כמי שאמר:

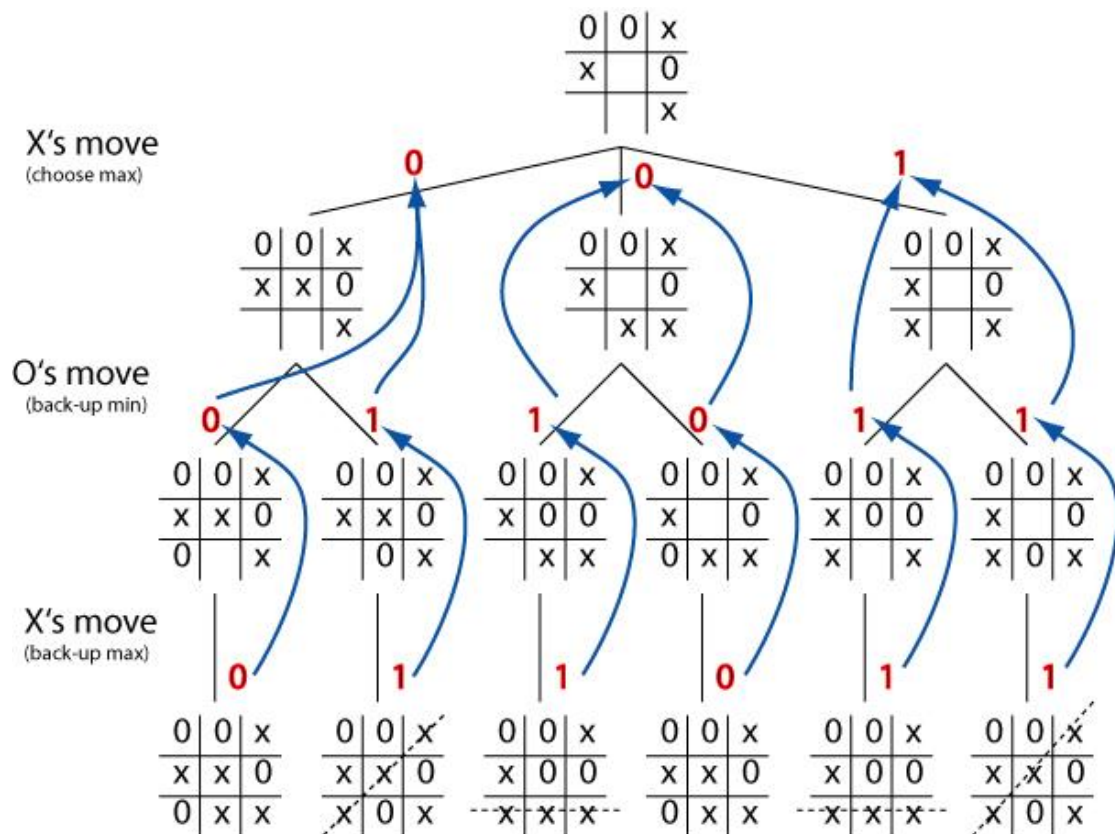
"As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the Minimax Theorem was proved".

מינימקס הינו אלגוריתם חזק מאוד ופשוט יחסית להבנה. הוא מתבסס על בניית עץ משחק הפורס את כל אפשרויות המשחק של שחקן א', את התגובות של שחקן ב', את תגובותיו של שחקן א' לשחקן ב' וכך הלאה עד להגעה למצב סופי (נצחון, הפסד, תיקו) או לעומק מוגדר מראש.

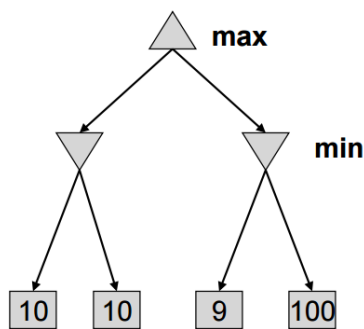
העלים של העץ הם המצבים הסופיים במשחק (terminal states) והם מכילים ערך מספרי אשר מייצג לנו כמה מצב זה הוא "טוב". מתן ציון למצב של לוח נעשה ע"י פונקציית הערכה (evaluation function). במשחק איקס-עיגול למשל, קל לתת ציון למצבים הסופיים. מצב סופי שבו שחקן א' מנצח יכול את הערך 1 כאשר מצבים של תיקו והפסד יכולו ערכים של 0 ו-1 בהתאמה. האלגוריתם יעזור לשחקן א' לבצע את מהלכיו ע"י בחירת המקסימום בכל תור של השחקן.

מינימקס מוצא את המהלך הכי טוב ע"י מעבר אחורה מתחתית העץ עד הנקודה בה אמורה להתבצע ההחלטה (backtracking). בכל שלב, מניחים ששחקן א' מנסה למקסם (maximize) את הסיכוי שינצח בעוד שבתור הבא שחקן ב' ינסה למזער (minimize) את הסיכוי ששחקן א' ינצח. מכאן גם מגיע שמו של האלגוריתם.

נסרוק את העץ החל בעלים, דרך הצמתים הפנימיים שמעליהם ועד השורש. הציון שניתן לכל צומת הוא הערך הגבוה ביותר של העלים/צמתים שתחתיו, אם אותו צומת מסמל מהלך של שחקן א'. מנגד, נבחר את הערך הנמוך ביותר של העלים/צמתים שתחתיו אם אותו צומת מסמל מהלך של היריב – שחקן ב'. כשנגיע לשורש, נבחר בצומת שמתחת לשורש עם הציון הגבוה ביותר כי השורש מסמל מהלך של שחקן א'.



איור 2 – המחשת פעולת אלגוריתם המינימקס במשחק איקס-עיגול: שחקן א' (X) מנסה למקסם את הציון בעוד שחקן ב' (O) מנסה למזער אותו. זהו מצב לוח המבטיח ניצחון לשחקן א' (X) מכיוון שמבין שלושת המהלכים האפשריים, הוא יבחר את המקסימום – הענף הימני של עץ המשחק.



איור 3 – עץ מינימקס הממחיש שמרנות

נשים לב שתהליך זה שונה ממצאת מצב עם ציון מקסימלי בלוח. מובן שאם נבחר במהלך (ענף) המוביל לעלה עם הציון הגבוה ביותר, לא מובטח לנו שאכן נגיע לאותו עלה. העלה הטוב ביותר הוא התסריט האופטימלי מבחינתנו, וניתן להניח שהיריב לא יוביל אותנו בהמשך הנתיב אלא יסיט אותנו לנתיב הטוב ביותר מבחינתו. לדוגמא, באיור 3 קיים מצב אופטימלי עם ציון 100 אך היריב ממזער את הציון ולכן יבחר את 9 דווקא – מה שיכריח אותנו לפנות לענף השמאלי בו היריב בחר 10. זוהי דוגמא שממחישה שמינימקס הוא אלגוריתם שמרן שאינו נוטה לסיכונים.

בעיה קשה בבניית עץ מינימקס היא הזיכרון הרב שהוא צורך. מספר הצמתים שיש ליצור עולה בטור הנדסי ככל שנעמיק את החיפוש. במשחקים בהם גורם ההסתעפות (branching factor) גבוה במיוחד נהוג להשתמש בגרסה משודרגת של האלגוריתם הנקראת "גיזום אלפא-ביתא" (Alpha-Beta Pruning). בגרסה זו לא בונים את כל העץ אלא מבטלים ענפים שברור לנו עוד בשלב מוקדם כי הם לא מועילים לחיפוש שלנו. כלומר, אנו יכולים להיתקל בענף שגרוע מפתרונות שכבר ראינו, ו"לגזום אותו". גיזום אלפא-ביתא מקטין את מספר הצמתים בעץ ממוצע לשורש מספר הצמתים המקורי.

שיטה נוספת להתמודד עם עומק עץ החיפוש היא הגבלת עומק החיפוש. ניתן להחליט שבונים את העץ רק עד עומק מסויים ואז עוצרים. בשיטה זו אמנם לא מגיעים עד המצבים הסופיים אך לפעמים מספיק להסתכל רק כמה מהלכים קדימה כדי לבצע החלטה טובה. דרך אחת היא להחליט על עומק קבוע. דרך נוספת היא לבנות עץ דינמי אשר ענפים מסויימים שנראים מבטיחים ייבנו עד עומק גבוה ולעומת זאת, הבנייה של ענפים מאכזבים תיפסק מוקדם בעומק נמוך. מקובל גם להגביל את זמן בניית העץ וריצת האלגוריתם לזמן סביר המקובל במשחק אנושי. במקרה זה, האלגוריתם יבנה את העץ לא עד עומק מסוים אלא עד שייגמר לו הזמן.

אלגוריתם מינימקס:

```
1. function minimax(node, depth, maximizingPlayer)
2.   if depth = 0 or node is a terminal node
3.     return the heuristic value of node
4.   if maximizingPlayer
5.     bestValue :=  $-\infty$ 
6.     for each child of node
7.       val := minimax(child, depth - 1, FALSE)
8.       bestValue := max(bestValue, val)
9.     return bestValue
10.  else
11.    bestValue :=  $+\infty$ 
12.    for each child of node
13.      val := minimax(child, depth - 1, TRUE)
14.      bestValue := min(bestValue, val)
15.  return bestValue
```

2.4 פונקציית הערכה היוריסטית

כאשר עץ המינימקס נבנה, כל מצב סופי מקבל ציון הערכה לגבי כמה הוא טוב. לשם כך צריך לתת למחשב דרך להסתכל על לוח ולהבין מה טוב בו ומה פחות. את הלוגיקה הזו אנו ממשים בעזרת היוריסטיקות.

היוריסטיקה (heuristic) היא כלל חשיבה פשוט המבוסס על הגיון פשוט או אינטואיציה ומציע דרך קלה ומהירה לקבלת החלטות. בהקשר של משחקים, היוריסטיקה היא פונקציה המנתחת מצב של לוח לפי הגיון כלשהו ומחזירה ציון למצב הלוח. דוגמאות להיוריסטיקות פשוטות בשחמט למשל הם כמות המלכות של השחקן פחות כמות המלכות של היריב או מספר המהלכים האפשריים.

$$h_1 = \text{no. of white queens} - \text{no. of black queens}$$

$$h_2 = \text{no. of possible moves}$$

כאשר נבחן מצבי לוח שחמט לפי ההיוריסטיקה הראשונה, האלגוריתם ייתן ציונים גבוהים יותר למצבים בהם היריב איבד את המלכה שלו ולכן ינסה ללכת לכיוון מצבים אלו בעץ המשחק. ההיוריסטיקה השנייה למשל יכולה לעזור בכך שהאלגוריתם ישאף למצבים בהם יש הרבה מהלכים, בניגוד ללהיות תקוע כשהיריב מכריח אותך לבצע מהלכים רעים. כל חוקר מנסה למצוא את ההיוריסטיקות שהכי מתאימות לאופי של המשחק אותו הוא חוקר.

לשם כך מגדירים פונקציה שמעריכה לוח משחק ובנוייה מכמה תת-פונקציות (היוריסטיקות) אשר לכל אחת משקל שונה בציון.

$$f(B) = \sum_{k=1}^N w_k \cdot h_k(B) = w_1 \cdot h_1(B) + \dots + w_k \cdot h_k(B)$$

לכל משחק צריך גם למצוא את האיזון העדין בין ההיוריסטיקות. פונקציית הערכה תכלול בדרך כלל יותר מהיוריסטיקה אחת וחלקן חשובות יותר מאחרות. בשחמט למשל, ברור שההיוריסטיקה "לא לאבד מלכה" חשובה יותר מההיוריסטיקה "לא לאבד צריח". מצד שני קיימים מצבים במשחק שדווקא כן שווה להקריב את המלכה שלך. את המשקלים ניתן לקבוע ידנית לפי אינטואיציה או ע"י תהליך אופטימיזציה שעליו נרחיב בהמשך.

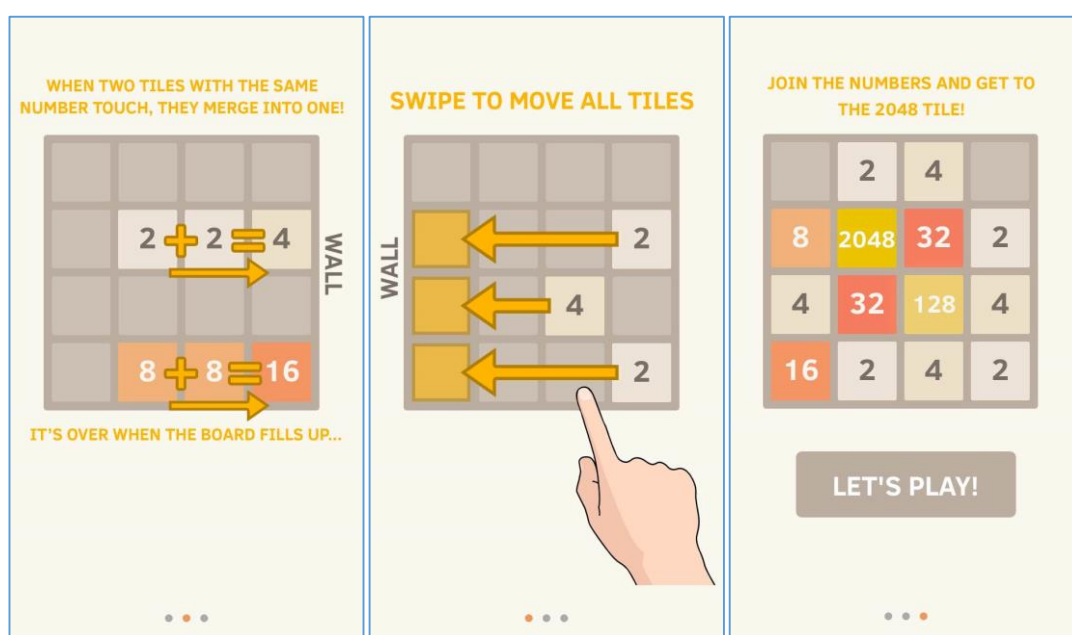
פונקציית הערכה שבנוייה מההיוריסטיקות עוזרת לנו בכך שהיא מאפשרת לנו להעריך מצב לוח שנמצא בשלב ביניים במשחק ולא רק במצבים סופיים של נצחון/תיקו/הפסד. כאשר מגבילים את עומק החיפוש יש צורך בהיוריסטיקות חזקות שיודעות לזהות לוח טוב או רע גם אם הוא רחוק ממצב סיום.

3. חלק ב' – בניית המודל המתמטי

3.1 המשחק 2048

2048 הוא משחק שחקן יחיד למחשב/סמארטפון שנוצר ע"י איטלקי בן 19 בשם גבריאלה צ'ירולי (Gabriele Cirulli) בהשראת משחקים דומים כגון Threes! ו-1024.

מטרת המשחק הינה להסיט אריחים (משבצות בעלות ערך מספרי) על לוח המשחק ולאחד צמידים של אריחים זהים כדי ליצור אריחים עם ערך כפול. כך למשל, איחוד של שני אריחים בעלי ערך 4 יתן אריח אחד עם ערך 8. ניצחון במשחק מושג כאשר מצליחים לאחד שני אריחים בעלי ערך 1024 לאריח אחד בעל הערך 2048. לאחר הנצחון אפשר להמשיך לשחק ולנסות להגיע לאריחים בעלי ערכים גבוהים יותר ויותר.



איור 4 – הוראות המשחק 2048 כפי שמצורפות למשחק. בצד ימין מוצג לוח המשחק, באמצע דוגמה להסטה הלוח ובצד שמאל דוגמה לאיחוד אריחים.

לוח המשחק (איור 4 ימין) הינו בגודל 4x4 ועליו כאמור מבצעים את הפעולות על המשבצות (הסטה של הלוח ואיחוד אריחים). בכל תור של המחשב, נבחרת משבצת ריקה בצורה אקראית ומקבלת ערך 2 בהסתברות 0.9 או ערך 4 בהסתברות 0.1.

השחקן בתורו בוחר לאן להסיט את הלוח: ימינה, שמאלה, למעלה או למטה. כאשר הלוח מוסט ימינה או שמאלה, כל האריחים זזים ימינה או שמאלה בכל שורה כאילו כוח המשיכה פועל כעת בצד שאליו הוסט הלוח (איור 4 אמצע). כאשר ישנם שני אריחים זהים באותה שורה וביניהם אין אריח שונה, הסטה של המסך לכיוון מסויים תחבר בין שני האריחים הזהים (איור 4 שמאל). לפי אותו היגיון פועלת גם הסטה של המסך לכיוון מעלה או מטה, כאשר התזוזה של האריחים מתבצעת בעמודות ולא בשורות.

המשחק מסתיים כאשר הלוח מלא ולשחקן אין יותר אפשרויות לאחד אריחים.

בזמן יציאת המשחק, הוא סחף אחריו מיליוני מעריצים בכל הגילאים שהתמכרו לפשטות, לכיף ובעיקר לאתגר השכלי שהוא מספק. שחקן אנושי ממוצע יצליח לאחר מאמץ מסויים להגיע לאריח 2048 אך יהיה לו קשה מאוד להגיע לאריחים גבוהים יותר. ישנם אנשים מסויימים בעלי יכולות גבוהות ומשמעת טובה אשר יכולים להגיע לאריחים בגובה 8,192 ואף 16,384. החסם העליון לאריח הגבוה ביותר הוא 131,072 אך הוא תאורטי בלבד והסיכוי להגיע לתוצאה כזו שואף לאפס. נכון לתחילת 2015, התוצאה הגבוהה ביותר שהושגה (ללא רמאות ופריצת המשחק) היא הגעה לאריח בגובה 32,768 והיא הושגה בעזרת תוכנת אינטליגנציה מלאכותית.

בחודשים שלאחר יציאת המשחק, כאשר קצת נרגעה המולת המעריצים, צצו ברחבי הרשת אינספור דיונים מתמטיים/הנדסיים הקשורים במידול הלוח, ייצוג מתמטי של הבעיה, חסמים תאורטיים של התוצאה ואסטרטגיות שונות לנצחון במשחק.

3.2 האלמנט הסטוכסטי

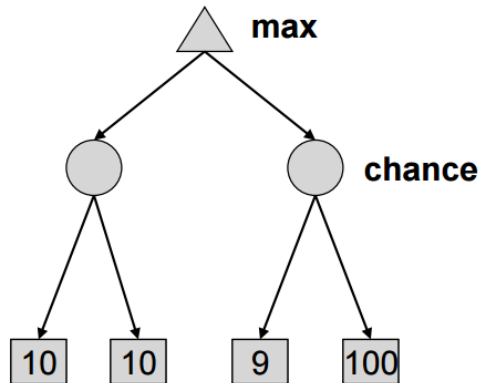
2048 הוא משחק בעל אלמנט סטוכסטי. המחשב מגריל את המהלך הבא שלו ואין לוגיקה מאחורי החלטתו. בבינה מלאכותית קיימות שתי גישות שונות המתמודדות עם בעיה זו. גישה ראשונה קובעת שאי אפשר להתייחס אל החלטות המחשב כאל החלטות יריב לוגי. נתייחס למשחק זה כאילו קיים שחקן יחיד שמבצע החלטות ותורו של המחשב הוא כמו הטלת קוביה – אין לוגיקה מאחורי החלטותיו והוא לא מנסה למזער את הציון (זה יכול לקרות בגלל האקראיות אבל המחשב גם יכול למקסם את הציון באותה מידה). מסיבה זו נטען, **לא רצוי** לנסות לפתור את המשחק בעזרת אלגוריתם מינימקס שתואר לעיל.

יש הטוענים דווקא את הטענה ההפוכה – למרות שהמחשב לא מתכנן את מהלכיו בצורה לוגית ניתן להחמיר וכן להתייחס אליו ככזה. בצורה זו, המשחק יתנהל בצורה יותר זהירה והאלגוריתם לא ינסה להגיע למצבים שמהם יהיה לו קשה לצאת. הוא יתייחס להגרלות המחשב כהחלטות של יריב לוגי שרוצה ברעתנו וינסה למזער את הנזקים ממהלכים הרסניים (גם אם אקראיים). לפי גישה זו, ניתן להשתמש באלגוריתם מינימקס גם לפתרון משחקים עם אקראיות.

החלטתי ללכת עם הגישה הראשונה שהיא קצת יותר אופטימית. המטרה של הפרוייקט היא להגיע לציון כמה שיותר גבוה במשחק והאקראיות תאפשר לנו להגיע למצבים מעולים שיריב לוגי לא היה מאפשר, מצבים שיוכלו להוביל לתוצאות גבוהות יותר. קיימים משחקים רבים בעלי מאפיינים סטוכסטיים ולכן קיימת גם גרסה מיוחדת של מינימקס אשר מיועדת למשחקים אלו. גרסה זו נקראת אלגוריתם אקספקטימקס (Expectimax) והיא מתמודדת עם הבעיה שתוארה לעיל.

3.3 אלגוריתם אקספקטימקס

זוהי גרסה משודרגת של אלגוריתם מינימקס בה מתמודדים עם האלמנט האקראי של המשחק. בשש-בש למשל, לפני כל תור שמבצע שחקן עליו להטיל זוג קוביות ותוצאת ההגרלה משפיעה ישירות על המהלכים. במשחק 2048 זה מתבטא בתורו של המחשב שמציב אריח במיקום אקראי ריק בלוח.



איור 5 - עץ אקספקטימקס הממחיש אופטימיות

בנוסף לצמתים שקיימים באלגוריתם מינימקס, שתפקידה למקסם (maximizer node) או למזער (minimizer node) את הציון לפי תור השחקן, כאן קיים גם צומת מזל (chance node) שתפקידו לדמות את האלמנט האקראי במשחק ומחזיר את התוחלת של בניו. בצורה זו, כאשר מסתכלים על ערך הצומת, משקללים אותו עם הסיכוי שנגיע לצומת זה.

בדוגמא שבאיור ניתן לראות שאלגוריתם אקספקטימקס יבחר ללכת בכיוון הצומת עם הערך המקסימלי 100 מכיוון שאין יריב שיסיט אותו

מהצומת אלא רק מזל. נניח לרגע שהסיכוי של כל צומת להיווצר שווים, אזי התוחלת של הענף הימני תהיה 54.5 בעוד שבענף השמאלי היא תהיה שווה רק ל-10. ברור שעדיף לעשות את המהלך שיוביל אותנו לענף הימני כי שם הסיכוי לרווח גדול מאוד וההפסד הפוטנציאלי מאוד קטן (100 מול 10 לעומת 9 מול 10). אלגוריתם זה מצטיין במובן שהוא לא מניח הנחות (מה יעשה היריב במצב כזה או אחר) אלא פשוט מבצע חיפוש סטטיסטי חכם על כל האפשרויות והולך בענפים שבהם הסיכוי הכי טוב להגיע למשבצות טובות.

חשוב לציין שלא אלגוריתם זה לא ניתן לממש גיזום אלפא-ביתא בגלל האלמנט האקראי. בגיזום אלפא-ביתא, מבטלים ענפים שלמים תוך כדי בניית העץ מבלי לחשב את הציון של כל הילדים. באלגוריתם אקספקטימקס עושים חישוב סטטיסטי על פני כל מרחב המצבים ולכן ביטול של ענף יפגע בנכונות החישוב.

שיטה אחרת שכן אפשר להשתמש בה כדי להתמודד עם גודל העץ היא Memoization (לא קיים תרגום לעברית). בשיטה זו מכינים מבנה נתונים נוסף לצד העץ (hash table) שיזכור את הציונים של מצבי לוח שכבר ראינו. ניתן לשים לב שבמהלך משחק, בונים את עץ המשחק שוב ושוב עבור לוחות שונים, והמון מצבים של הלוח חוזרים על עצמם. זה קורה מכיוון שלמצב משחק מסויים אפשר להגיע בכמה אופנים שונים. לכן, אם לאחר חישוב האלגוריתם עבור מצב משחק כלשהו נאחסן את התוצאה, כשניתקל במצב זה שנית לא נצטרך לבנות את כל עץ המשחק (או הענף) מחדש אלא פשוט נפנה למבנה הנתונים ונבקש את התוצאה – פעולה מהירה וחסכונית משמעותית.

```

1. function expectimax(node, depth)
2.   if node is a terminal node or depth = 0
3.     return the heuristic value of node
4.   if the adversary is to play at node
5.     let  $\alpha := +\infty$ 
6.     foreach child of node
7.        $\alpha := \min(\alpha, \text{expectimax}(\text{child}, \text{depth}-1))$ 
8.   else if we are to play at node
9.     let  $\alpha := -\infty$ 
10.    foreach child of node
11.       $\alpha := \max(\alpha, \text{expectimax}(\text{child}, \text{depth}-1))$ 
12.   else if random event at node
13.     let  $\alpha := 0$ 
14.     foreach child of node
15.        $\alpha := \alpha + (\text{Probability}[\text{child}] * \text{expectimax}(\text{child}, \text{depth}-1))$ 
16.   return  $\alpha$ 

```


3.4 היוריסטיקות למשחק 2048

על מנת לנסח היוריסטיקות טובות צריך לדעת לשחק את המשחק ולהבין מהי אסטרטגיה טובה. חשוב לאמר שציון של היוריסטיקה יכול להיות גם שלילי. אפשר לנסח כלל מסוים כך שהוא יהווה עונש (Penalty) ויגרע מהציון הכללי של כל ההיוריסטיקות.

די ברור כבר בהתחלה – בשלב הלימוד של המשחק - ששווה לסדר את הלוח בצורת משולש כך שהערך הגבוה ביותר יהיה באחת הפינות והערכים הגבוהים הבאים צמודים אליו. שמירה על צורה זו עוזרת ללוח להישאר מסודר ושאינך גבוהים נשארים קרובים אחד לשני (עוזר במיזוג ערכים גבוהים).

The image displays three 4x4 grids illustrating the progression of a 16-point game. Each grid shows a player's current score and a goal score, with the difference representing the remaining points needed to win.

- Grid 1:** A player has 2 points, and the goal is 4 points. The difference is 2 points.
- Grid 2:** A player has 32 points, and the goal is 4 points. The difference is 28 points.
- Grid 3:** A player has 8 points, and the goal is 16 points. The difference is 8 points.

The grids use a color scale to represent the point values, ranging from light yellow (low) to dark red (high). The goal score is always 4 or 16 points, and the player's score is always a multiple of 4.

א'ור 6 - דוגמאות ללוח בצורת משולש

לכל פינה נגדיר מטריצת משקל. כאשר מכפילים את ערכי הלוח בערכי מטריצת משקל כלשהי, תתקבל תוצאה גבוהה יותר במידה והלוח מסודר בצורה דומה לאותה קונפיגורציית משקלים. נגדיר את קבוצת מטריצות המשקלים $W_{corners}$ אשר תכיל את המטריצות הבאות (אחת לכל פינה):

$$W_{tr} = \begin{bmatrix} 4 & 5 & 6 & 7 \\ 3 & 4 & 5 & 6 \\ 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 \end{bmatrix}, W_{tl} = \begin{bmatrix} 7 & 6 & 5 & 4 \\ 6 & 5 & 4 & 3 \\ 5 & 4 & 3 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}, W_{br} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{bmatrix}, W_{bl} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 5 & 4 & 3 & 2 \\ 6 & 5 & 4 & 3 \\ 7 & 6 & 5 & 4 \end{bmatrix}$$

יהי B מצב לוח של המשחק 2048 – מטריצה 4×4 כאשר B_{ij} מייצג אריח בלוח בשורה i ועמודה j . ההיוריסטיקה הראשונה הינה סכום המכפלות בין כל איברי הלוח לאיברי מטריצת המשקל המתאימים:

$$h_1(B) = \max_{w \in W_{\text{corners}}} \sum_{i=1}^4 \sum_{j=1}^4 B_{i,j} \cdot w_{i,j}$$

אסטרטגיה נוספת שרצוי שנממש בצורת היוריסטיקה היא מיזוג אריחים. די ברור שכדי להתקדם במשחק עלינו למקם אריחים זהים אחד ליד השני ולמזג אותם לאחר מכן. נדאג לכך בהגדרת ההיוריסטיקה השנייה ע"י כך שנספור את כל זוגות האריחים הצמודים והזהים בכל שורה ועמודה.

$$\text{equal}(x, y) = \begin{cases} 1, & x = y \\ 0, & \text{otherwise} \end{cases}$$

$$h_2(B) = \sum_{i=1}^4 \sum_{j=2}^4 \text{equal}(B_{i,j}, B_{i,j-1}) + \sum_{j=1}^4 \sum_{i=2}^4 \text{equal}(B_{i,j}, B_{i,j-1})$$

היוריסטיקה שלישית תוגדר להיות כמות האריחים הריקים בלוח. היוריסטיקה זו עוזרת לנו למזג אריחים צמודים מכיוון שכמות אריחים נמוכה יותר אומרת שבוצעו מיזוגים. לכן נשאף לחפש מצבי לוח כמה שיותר ריקים.

$$\text{empty}(x) = \begin{cases} 1, & x = 0 \\ 0, & \text{otherwise} \end{cases}$$

$$h_3(B) = \sum_{i=1}^4 \sum_{j=1}^4 \text{empty}(B_{ij})$$

2048	1024	512	256
32	32	64	128
2	8	4	2
			2

איור 7 - דוגמה לצורת נחש

לבסוף נשים לב לאסטרטגיה חשובה שבלעדיה קשה להגיע לתוצאה גבוהה. לפי אסטרטגיה זו, כדאי לנו לסדר את האריחים הגבוהים בצורה מונוטונית דמויית נחש כך שיהיה קל למזג אריחים. באיור 7 ניתן לראות את הכוונה. מצב זה הוא אופטימלי להגעה לאריח 4096. מכאן ניתן ברצף למזג את 32+32 ומיד אח"כ להמשיך עם 64+64, 128+128 וכן הלאה, עד שמגיעים ל- 2048+2048.

במהלך המשחק, כאשר נכפה עלינו לצאת ממצב מונוטוני נוח, המבנה שבנינו יכול לקרוס בקלות ולהוביל לתסבוכת גדולה ולהפסד. הדבר משמעותי מאוד כאשר הערכים בלוח גבוהים במיוחד.

לכן, היוריסטיקה זו תמומש בצורת עונש על שורות לא מונוטוניות, נעבור על כל השורות והעמודות ונבדוק מונוטוניות מכל הכיוונים. ניתן משקל גבוה יותר לסדרות לא מונוטוניות של אריחים עם ערך גבוה. בדיקת מונוטוניות של עמודות תתבצע ע"י שחלוף הלוח ובדיקתו בצורת מונוטוניות של שורות.

נאמר ששורה היא מונוטונית ימנית אם הערכים של האריחים בה גדלים משמאל לימין כמו השורה השנייה באיור 7. לפי אותו הגיון, השורה הראשונה באיור 7 היא מונוטונית שמאלית.

$$\text{bigger}(x, y) = \begin{cases} 1, & x \geq y \\ 0, & \text{otherwise} \end{cases}$$

$$\text{right}_{\text{monotonicity}}(B) = \sum_{i=1}^4 \sum_{j=2}^4 \text{bigger}(B_{i,j}, B_{i,j-1}) * (B_{i,j} - B_{i,j-1})$$

$$\text{left}_{\text{monotonicity}}(B) = \sum_{i=1}^4 \sum_{j=2}^4 \text{bigger}(B_{i,j-1}, B_{i,j}) * (B_{i,j-1} - B_{i,j})$$

כעת ניקח את המינימום מבין המונוטוניות השמאלית והימנית כי זוהי היוריסטיקה שלילית ואנו מחפשים פגיעה במונוטוניות.

$$monotonicity(B) = \min(right_{monotonicity}(B), left_{monotonicity}(B))$$

$$h_4(B) = -monotonicity(B) - monotonicity(\bar{B})$$

3.5 ניסוח מתמטי של האלגוריתם לפתרון המשחק 2048

לשם בניית בינה מלאכותית שתפתור את המשחק 2048, נשתמש באלגוריתם אקספקטימקס תוך שימוש בהיוריסטיקות שתוארו לעיל. נגדיר מושגים, משתנים ופונקציות שבעזרתם נוכל להגדיר את פונקציית ההערכה.

תהי $M_B = \{up, down, left, right\}$ קבוצת המהלכים האפשריים במצב לוח B .

תהי $move(B, m)$ פונקציה אשר מקבלת מצב לוח B ומהלך $m \in M$. הפונקציה מבצעת מהלך בכיוון הנבחר ומחזירה מצב לוח חדש.

תהי R_B קבוצת כל מצבי הלוח האפשריים המתקבלים מהצבת אריח אקראי בלוח B .

תהי $P(B, B') \in [0, 1]$ ההסתברות לקבל את מצב הלוח $B' \in R_B$ מהלוח B ע"י הצבת אריח אקראי.

יהי $d \in \mathbb{N}$ משתנה המציין את עומק עץ הרקורסיה הנוכחי.

תהי $h_k(B)$ פונקצית היוריסטיקה אשר מקבלת מצב לוח B ומחזירה ציון. בנוסף, יהי w_k משקל ההיוריסטיקה בפונקציית ההערכה. $k \in \{1..N\}$ כאשר N הוא מספר ההיוריסטיקות.

תהי $evaluate(B)$ פונקציית ההערכה למצב לוח B . $eval$ מקבלת מצב לוחי סופי (terminal node) ומחזירה ציון לפי הנוסחה:

$$evaluate(B) = \sum_{k=1}^N w_k h_k(B)$$

האלגוריתם הינו רקורסיבי בשני שלבים המדמים צמתי החלטה של מקסום ומזל. תהי $score$ פונקציה רקורסיבית אשר מקבלת מצב לוח. במידה ומצב הלוח הוא סופי, הפונקציה תקרא לפונקציית ההערכה $evaluate$ שתחזיר ציון. במידה ומצב הלוח לא סופי היא קוראת לעצמה שוב ושוב כמספר המהלכים האפשריים. הציון שיוחזר הוא המקסימלי מכל המהלכים האפשריים שנוסו (צומת החלטה). בכל חזרה מקריאה רקורסיבית תחושב תוחלת הציונים (צומת מזל).

ניתן לנסח אפוא את הנוסחה הרקורסיבית שלפיה האלגוריתם יעבוד:

$$score(B, d) = \sum_{B' \in R_B} P(B, B') \cdot \begin{cases} eval(B'), & \text{if } d = 0 \text{ or state is terminal} \\ \max_{m \in M_B} score(move(B', m), d - 1), & \text{otherwise} \end{cases}$$

לכן, בהינתן מצב לוח מסויים בשלב שבו על האלגוריתם להמליץ על מהלך, יבחר המהלך בעל הציון המקסימלי מבין המהלכים האפשריים:

$$decision(B) = \operatorname{argmax}_{m \in M_B} score(move(B, m))$$

4. חלק ג' – תהליך האופטימיזציה

4.1 אופטימיזציה מטה-היוריסטית (Metaheuristic Optimization)

אופטימיזציה מטה-היוריסטית הינו תחום חדש יחסית שהתפתח בשל הצורך לעשות אופטימיזציה לפונקציות מורכבות שאינן בהכרח מתמטיות אלא יותר דומות לבינה המלאכותית שתוארה בסעיפים הקודמים. פונקציות אלה הן בדרך כלל בעלות פרמטרים רבים ואין להן בהכרח פתרון אופטימלי. למרות זאת, עדיין קיים הצורך לעשות לאלגוריתמים אלה אופטימיזציה כדי להשיג תוצאה טובה יותר.

שיטת אופטימיזציה מטה-היוריסטית לא נסמכת על תכונה כלשהי של הבעיה למציאת פתרון, אלא מספקת מגנון כללי למציאת אופטימום לוקאלי של הבעיה. נתייחס לאלגוריתם שלנו כקופסה שחורה (black-box) אשר מקבלת משקלי היוריסטיקות כפרמטרים, מריצה משחק שלם ומחזירה לבסוף את תוצאת המשחק. לאלגוריתם האופטימיזציה אין ידע נוסף לגבי איך פועל האלגוריתם שלנו.

תהי $g: \mathbb{R}^n \rightarrow \mathbb{R}$ "פונקציית מעטפת" אשר תקבל כפרמטר את המשקלים ותריץ משחק שלם של 2048 בעזרת האלגוריתם שלנו והמשקלים שהתקבלו.

פונקציית המעטפת:

```
1. function g(w1, w2, ..., wn)
2.     start game with input weights
3.     while game is not over do:
4.         ask AI for next move
5.         execute suggested move
6.         execute random move
7.     return game final score
```

בתהליך האופטימיזציה נפעיל את פונקציית g שוב ושוב בצורה איטרטיבית ונחפש פתרון אופטימלי. הפתרון האופטימלי $\vec{z} = (w_1, w_2, \dots, w_n) \in \mathbb{R}^n$ הינו בעל הציון הגבוה ביותר ולכן מקיים את המשוואה:

$$\forall \vec{y} \in \mathbb{R}^n: g(\vec{z}) \geq g(\vec{y})$$

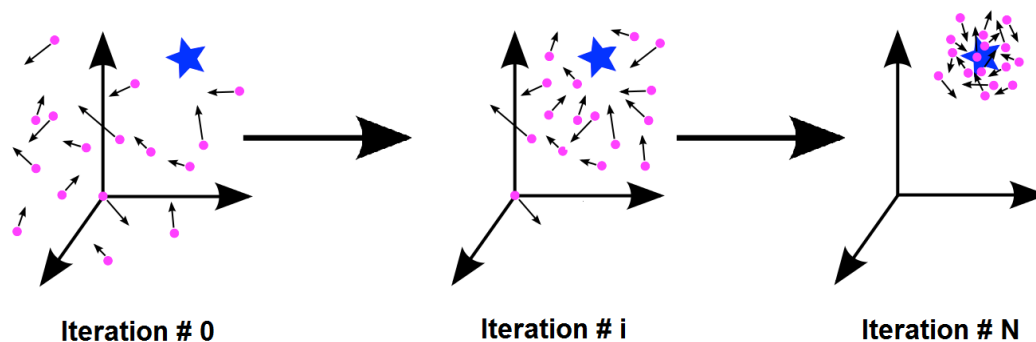
הדרך המסורתית לפתור בעיה שכזו היא בעזרת מעקב אחרי הגרדיאנט אך כנראה שהפונקציה שלנו איננה רציפה והגרדיאנט אינו ידוע ולכן בחרתי בשיטה שונה.

4.2 אופטימיזצית הנחיל (Particle Swarm Optimization)

אופטימיזצית הנחיל (PSO) היא שיטת אופטימיזציה מטה-היוריסטית מבוססת אוכלוסייה (Population Based Metaheuristic Optimization) אשר פותחה ב-1995 ע"י ד"ר קנדי וד"ר אוורהארט. השיטה נוצרה בהשראת ההתנהגות החברתית של להקת ציפורים או דגים. כדי להבין כיצד פועלת השיטה, אפשר לדמיין להקת ציפורים בשעת חיפוש אחרי אוכל. נניח שמרחב החיפוש שלהן מוגבל, כמו שדה גדול למשל, ושבמקום כלשהו בשדה מונחת כיכר לחם. תחילה, הציפורים אינן יודעות את מיקום הלחם אך ככל שחולף הזמן חלק מהציפורים קרובות יותר ללחם ויכולות להריח או לראות אותו. בסופו של דבר ציפור מוצאת את הלחם. מה יעשו שאר הציפורים? הן יעקבו אחרי הציפור שמצאה את האוכל וכולן באופן טבעי יבוא לחלוק איתה את הארוחה.

PSO משתמש באותו הגיון כדי לחפש פתרון אופטימלי במרחב החיפוש. תחילה מגדירים חלקיקים (particles) במיקומים אקראיים ובעלי מהירות אקראית בתחום מרחב החיפוש. כל מיקום של חלקיק מייצג פתרון. ה"ציון" של החלקיק מתקבל ע"י קריאה ל- g עם המיקום שלו כפרמטר (כל קואורדינטה מייצגת משקל אחד). המהירות אחראית לכוון את החלקיק ליעד נוסף באיטרציה הבאה, שם כל חלקיק יקרא ל- g פעם נוספת עם מיקומו החדש.

בכל איטרציה, החלקיק מכוון את מהירותו כדי לעקוב אחרי שני פתרונות מיטביים. הפתרון המיטבי הראשון הוא החלק הקוגניטיבי, שבו החלקיק עוקב אחרי המיקום הטוב ביותר שנמצא עד כה ע"י אותו חלקיק בעצמו. הפתרון השני שלפיו מכוונת המהירות הוא החלק הסוציאלי, שבו החלקיק עוקב אחרי המיקום הטוב ביותר שנמצא עד כה ע"י כל חברי הנחיל (החלקיקים).



איור 8 – המחשת פעולת PSO על פונקציה עם אופטימום יחיד. תחילה כל החלקיקים בעלי מהירות ומיקום אקראיים. לאחר i איטרציות החלקיקים מתחילים להתקבץ בכיוון הפתרון, ולאחר N איטרציות כל החלקיקים קרובים מאוד לפתרון האופטימלי.

תהי $rand(a, b)$ פונקציה המחזירה מספר אקראי בתחום $[a, b]$.

בכל איטרציה $i \in \mathbb{N}$, כל חלקיק מכוון את מיקומו ($\vec{x} \in \mathbb{R}^n$) ומהירותו ($\vec{v} \in \mathbb{R}^n$) ע"י המשוואות הבאות:

$$\vec{v}_{i+1} = \vec{v}_i + c_1 \cdot rand(0,1) \cdot (\vec{p}_{best} - \vec{x}_i) + c_2 \cdot rand(0,1) \cdot (\vec{s}_{best} - \vec{x}_i)$$

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_{i+1}$$

כאשר $\vec{p}_{best}$ הוא המקסימום של החלקיק עצמו (particle best) ו- $\vec{s}_{best}$ הוא המקסימום של הנחיל כולו (swarm best). c_1 ו- c_2 מייצגים את פקטור המדד החברתי באוכלוסייה ונותנים משקלים לחלק הקוגניטיבי ולחלק הסוציאלי.

אלגוריתם PSO :

```

1. function PSO()
2.   for each particle
3.     Initialize particle with random position and velocity
4.   for N iterations
5.     for each particle
6.       score = g( $\vec{x}$ )
7.       If score  $\geq \vec{p}_{best}$ 
8.          $\vec{p}_{best} :=$  score
9.       Choose particle with highest  $\vec{p}_{best}$  and set  $\vec{s}_{best}$  as well.
10.      Save  $\vec{s}_{best}$ 's  $\vec{x}$  value as optimum
11.      Calculate new position and velocity
12. return optimum

```

לאחר N איטרציות, נתבונן ב- $\vec{s}_{best}$. מיקומו של $\vec{s}_{best}$ הינו בעל הציון המקסימלי מבין כל המשחקים ששוחקו. לכן בוחרים את המשקלים שמיוצגים ע"י מיקומו.

יש לציין של-PSO מאפיינים משותפים עם אלגוריתמים גנטיים והוא קצת מזכיר את פעולתם. בשניהם המערכת מאותחלת עם אוכלוסייה התחלתית אקראית ומבצעת חיפוש אופטימום ע"י עדכון דורות (generations) של פתרונות. לעומת זאת, בניגוד לאלגוריתמים גנטיים, ל-PSO אין אופרטורים אבולוציוניים כגון זיווג (crossover) ומוטציה (mutation). ב-PSO, הפתרונות הפוטנציאליים (החלקיקים) "עפים" ברחבי מרחב החיפוש ע"י עקיבה אחרי האופטימום הידוע ולא נוצרים לפי לוגיקה אבולוציונית.

4.3 מהלך הניסוי

כאמור, PSO מהווה פונקציית מעטפת לאלגוריתם הבינה המלאכותית המכוונת אותו לפי הפרמטרים – משקלי ההיוריסטיקות. פונקציית המעטפת תריץ את האלגוריתם (את המשחק) שוב ושוב מספר קבוע של פעמים ובסופו של דבר תבחר את המשקלים שהניבו את הציון הגבוה ביותר.

הרצה של האלגוריתם פעם אחת לוקחת זמן ניכר. זמן זה נע בין עשר שניות לכעשר דקות. כאשר משקלי ההיוריסטיקות כוונו בצורה ידנית לפי האינטואיציה, האלגוריתם לא הצליח להגיע לציונים גבוהים מ-2,048. משחק כזה נכשל לאחר כ- 40-20 שניות. לעומת זאת, לאחר שהאלגוריתם כוון ע"י תהליך האופטימיזציה, הוא הצליח להגיע לתוצאות טובות בהרבה אך גם זמן המשחק גדל בצורה משמעותית. הניסוי בוצע על מחשב נישא פשוט שנקנה ב-2012, בעל מעבד Intel Core i5 וזכרון 4GB.

יש להתחשב בזמני ביצוע של האלגוריתם כאשר מתכננים את הסימולציה מכיוון שהחישוב צריך להתבצע בזמן סביר בהתאם למערכת עליה הוא מתבצע. לכן, בחרתי בכמות של 50 חלקיקים שיבצעו 100 איטרציות של PSO. כלומר, בסימולציה אחת משוחקים בסה"כ 500 משחקים שלמים של 2048.

כבר בהרצה הראשונה של הסימולציה (שארכה כ-5 שעות), נמצאו משקלים שבזכותם האלגוריתם הצליח להגיע לאריח מירבי בעל ערך 8,192. זהו בערך הגבול שבו זמן ביצוע האלגוריתם (עבור משחק אחד) הוא סביר ולוקח כ-2 דקות. כמובן שכדי להגיע לאריח בגובה 16,384 יש צורך בזמן כפול לפחות. כדי להגיע ל-16,384 צריך להגיע פעמיים ל-8,192 אך בנוסף, נוצר מחסור באריחים ריקים לתמרון ולכן זמן החשיבה גדל וזמן המשחק הכולל גדל משמעותית.

מהתבוננות בסדרות המשקלים שנבחרו בכל פעם, שמתי לב שהאלגוריתם נותן משקלים הקרובים לאפס להיוריסטיקה הראשונה. כאשר ניסיתי לבטל אותה לגמרי לא הייתה השפעה ממשית על התוצאות. לבסוף ניסיתי להפוך את ההיוריסטיקה לשלילית (עונש) והתוצאות היו מעולות. אני משער שהיוריסטיקה זו התנגשה עם היוריסטיקת המונוטוניות מכיוון שהן משלימות אחת את השנייה. כנראה שלמונוטוניות יש יותר השפעה מאשר סידור הלוח בצורת משולש ושינוי המשקל לשלילי עוזר יותר מהסרתו. זה מה שיפה בתהליך אופטימיזציה, לפעמים מוצאים פתרונות שמנוגדים לאינטואיציה, פתרונות שלא היינו מגיעים אליהם ללא ניסוי וטעייה.

את הסימולציה השנייה ביצעתי כך שהחלקיקים פוזרו מסביב למיקום שאותו מצאנו בסימולציה הראשונה, זאת כדי להמשיך את המגמה ולהתחיל ממקום שכבר ידוע שמביא תוצאות יפות. ביצעתי גם שינויים בפרמטרים של PSO שמציינים כיצד להגריל מיקום חדש או לשנות מהירות, בתקווה למצוא פתרונות יותר טובים. סימולציה זו ארכה זמן רב מדי וגרמה למחשב לקרוס ממחסור בזכרון.

ההשערה שלי הייתה שנמצא משקל שבעזרתו התקבלו תוצאות גבוהות מאוד. לכן, מאותו רגע המשחקים לקחו הרבה יותר זמן והאריכו את זמן הסימולציה מעבר ליכולתו של המחשב. המחשב קרס והתוצאות לא נשמרו ולכן חזרתי על הניסוי.

בסימולציה השלישית ביצעתי שינויים נוספים והפחתתי את כמות החלקיקים (כמות הזכרון בכל איטרציה) ואת מספר האיטרציות כדי לנסות בכל זאת לשבור את המחסום בזמן חישוב סביר. בנוסף הרצתי הפעם את הסימולציה על מחשב שולחני חזק יותר.

הסימולציה השלישית והאחרונה הוגדרה כהצלחה ובסופה קיבלתי משקלים שהביאו את האלגוריתם לאריח שיא בגובה 16,384. אלו המשקלים שנמצאו ע"י PSO והוגדרו להיות משקלי ההיוריסטיקות באלגוריתם:

$$h_1 = 1.6321, \quad h_2 = 2.7553, \quad h_3 = 3.0645, \quad h_4 = 6.8515$$

4.4 תוצאות

לאחר מציאת המשקלים, הרצתי את האלגוריתם 100 פעמים ובדקתי את ביצועיו בצורה סטטיסטית. להלן התוצאות:

האריח המקסימלי 32,768 הושג ב-3 משחקים.

האריח המקסימלי 16,384 הושג ב-27 משחקים.

האריח המקסימלי 8,192 הושג ב-46 משחקים.

האריח המקסימלי 4,096 הושג ב-24 משחקים.

בדיקה נוספת שביצעתי היא האם הגדלת עומק החיפוש משפיע על התוצאה. עומק החיפוש של האלגוריתם הוגדל ל-16 רמות בעץ הרקורסיה (במקום 9-3 רמות). ביצעתי את הניסוי במשך 3 לילות רצופים ובשלושת הניסיונות הושג אריח מקסימלי של 32,768. אין להסיק מכך כי אחוזי ההצלחה הם 100% להגעה לאריח 32,768 אך כן ניתן להסיק שאחוז ההצלחה עלה בצורה משמעותית. קיימת בעיה לבדוק את העניין בצורה סטטיסטית בגלל זמן הריצה הארוך במיוחד (המשחק האחרון ארך 11 שעות).

5. סיכום

בתחילת העבודה, עוד בשלב התכנון, הצבתי לעצמי יעד. רציתי שהאלגוריתם יוכל לפחות להשתוות לשחקן אנושי מבחינת האריח המקסימלי שאליו הוא יגיע, משהו בסביבות 8,192-4,096.

אני שמח להגיד שהיעד הושג בהצלחה ואף הרבה מעבר לכך. 27% הצלחה בהגעה לאריח מקסימלי של 16,384 ו-3% הצלחה בהגעה לאריח מקסימלי של 32,768, מראים שהאלגוריתם שתכננתי פועל היטב ובעל ביצועים מעולים.

תוצאה זו לא הייתה מושגת במידה ולא הייתי מבצע את תהליך האופטימיזציה המיוחד. ע"י כיוון ידני של משקלי ההיוריסטיקות הצלחתי להגיע לאריח מקסימלי של 2,048 ורק לאחר הכיוון הממוחשב הושגו התוצאות הטובות באמת.

נחמד להסתכל על הרצה של משחק מלא ולשים לב לכל אותם מצבים מנוגדי היגיון, שבהם שחקן אנושי לבטח היה עושה טעות ומפסיד במשחק. ע"י שילוב נכון של היוריסטיקות ועומק חיפוש דינמי, האלגוריתם מבצע פעולות חכמות מאוד. בדרך כלל פעולות אלו קשורות בכך שנסרקו המון מהלכים קדימה וזוהה מסלול בעץ ששחקן אנושי בדרך כלל לא היה חושב עליו.

כמובן שעדיין יש מקום לשיפור ואפשר להשיג תוצאות טובות אף יותר. מעבר לכוח מחשוב חזק יותר יכול להיות צעד בכיוון הנכון. אפשר גם לשפר את מנגנון ההחלטה לגבי עומק החיפוש ולהוסיף היוריסטיקות חדשות. ישנו גם מקום לשיפור ביעילות האלגוריתם והזכרון שהוא צורך. כל שיפור שכזה יכול לדחוף את האלגוריתם עוד צעד קדימה.

אסכם ואומר שמאוד נהנתי במהלך ביצוע הפרויקט - זה היה אתגר מאוד גדול ועמדתי בו בהצלחה. החכמתי במגוון נושאים וצברתי נסיון מעשי במימוש אלגוריתמים של תורת המשחקים, בינה מלאכותית ואופטימיזציה.

ברצוני גם להודות מקרב לב למנחה הפרויקט, ד"ר גבריאל בן סימון על תמיכתו המלאה לכל אורך הפרויקט, על סבלנותו הרבה ועל כל השיחות המעניינות שניהלנו בתקופה זו.

6. ביבליוגרפיה

1. Claude E. Shannon (1950) , Programming a Computer Playing Chess, Philosophical Magazine, Ser. 7, Vol 41, No. 312.
2. Russell, Stuart J.; Norvig, Peter (2003). Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 163–171.
3. Osborne, Martin J., and Ariel Rubinstein (1994). A Course in Game Theory. Cambridge, MA: MIT Print.
4. D. Michie (1966). Game-playing and game-learning automata. In L. Fox (ed.), Advances in Programming and Non-Numerical Computation, pp. 183-200.
5. Kennedy, J.; Eberhart, R. (1995). Particle Swarm Optimization, Proceedings of IEEE International Conference on Neural Networks IV. pp. 1942–1948.
6. <https://www.youtube.com/watch?v=jaFRyzp7yWw> – CSS188 lecture, Stanford University.
7. <http://www.swarmintelligence.org/tutorials.php> - Information about PSO.

7. נספח – הוראות התקנה והפעלה

לעבודה זו מצורף קובץ ZIP. המכיל את קוד המקור של המערכת וכמו כן קובץ הרצה EXE.

רשימת הקבצים:

2048.h – Header file with definitions.

2048.cpp – Where the game 2048 and the Expectimax algorithm are implemented.

PSO.cpp – Where PSO algorithm is implemented.

2048.py – Script to run the AI with graphics directly in browser.

Gamectrl.py – Remote control to access the game JavaScript controls via keyboard.

Buid.bat – Batch file to run in Windows Command window and compile the system.

Bin\2048.exe – Executable file to run a full game using the AI or run a PSO simulation.

תהליך התקנה:

1. חובה להתקין קומפיילר של C++. במקרה שלנו עדיף VS2012 ומעלה.
2. יש להוסיף את הקומפיילר של VS בשם "CL" שכרגע הותקן ל-PATH של ווינדוס או לחלופין לבצע את הפעולות הבאות מתוך ה-command window שבתוך VS.
3. לפתוח את קובץ ה-ZIP ולהגיע לתיקיה המתאימה ב-command window
4. להריץ את הקובץ build.bat

שלב זה בהתקנה נחוץ להפעלת הגרסה הגרפית:

5. להוריד Add-On של Firefox בשם Remote Control 1.0

הרצה של משחק בודד:

1. להכנס לתיקיה bin ולהריץ את 2048.exe

הרצה של גרסה גרפית:

1. להיכנס לאתר הרשמי של 2048 ולהדליק את התוסף
2. יש להיכנס ל-2048.cpp ולהוסיף הערה לשורה: #define NO_GUI
3. להריץ את 2048.py

הרצה של PSO:

1. יש להיכנס ל-2048.cpp ולהוריד מהערה את פונקציית ה-main של PSO ולשים בהערה את פונקציית ה-main הרגילה.
2. לקמפל מחדש בעזרת build.bat ולהריץ את 2048.exe.