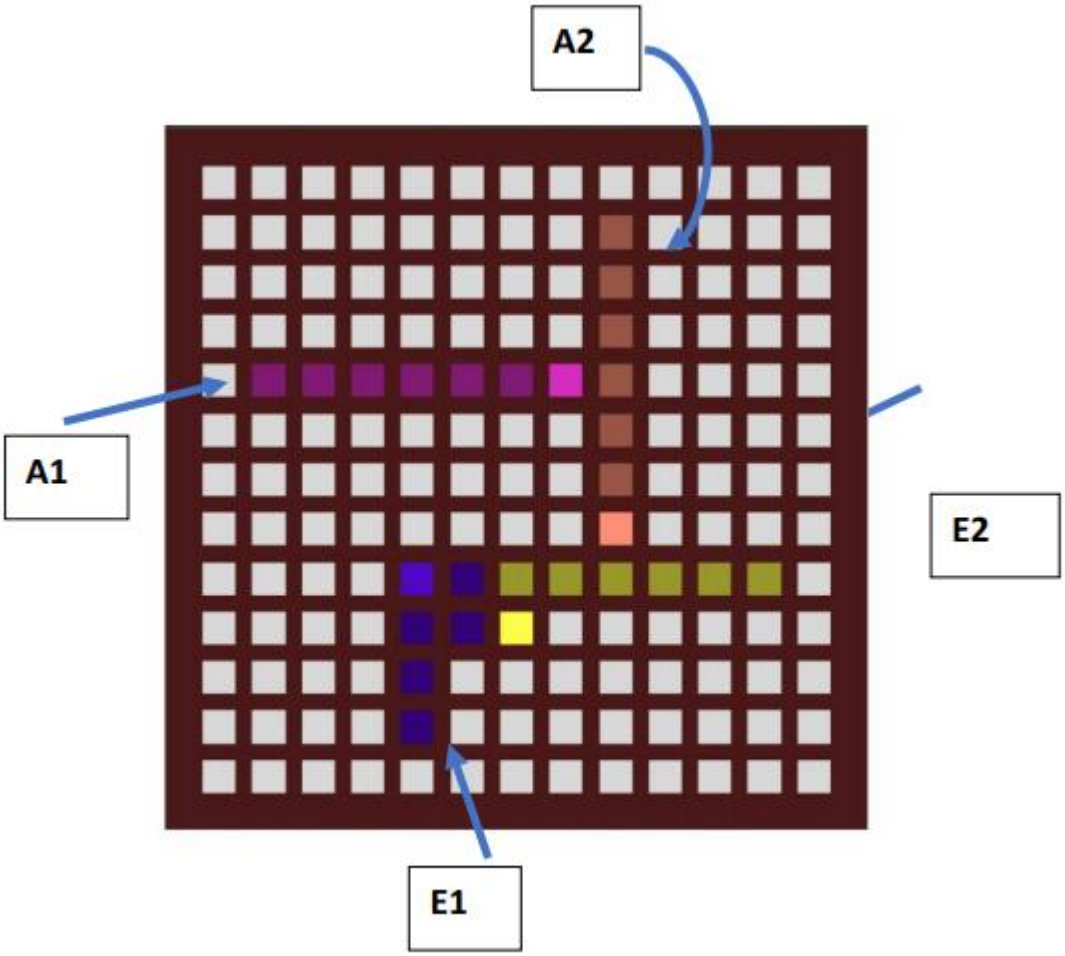
פרויקט בינה מלאכותית



1 הבעיה

1.1 הגדרת הבעיה

במסגרת קורס בינה מלאכותית קיבלנו לממש משחק מסוג סנייק, המבוסס על עיקרון Zero-Sum Games, עם סביבת מרובת סוכנים בעלי התנהגויות מנוגדות. עלינו לבחור אילו אלגוריתמים של בינה מלאכותית נוכל להתמיע בסוכנים על מנת לפתור את הבעיה.

1.2 חוקי המשחק

לוח בגודל 13X13 כאשר קיימים בו שתי קבוצות של שני סוכנים . בעלי הצבעים $Color = \{Blue, Cyan, Red\ , Orange\}$.

<u>קבוצה א'</u>: "**Agents**" סוכנים שמטרתם לחיות כמה שיותר זמן על הלוח.

צבעי הסוכנים בקבוצה הם, $\{Blue, Cyan\}$.

<u>קבוצה ב'</u>: "**Enemies**" סוכנים שמטרתם היא להכשיל את הסוכנים של קבוצה א'

צבעי הסוכנים בקבוצה הם,$\{Red, Orange\}$.

צבירת נקודות: על כל פריים שסוכן מסוג Agent בחיים הוא צובר נקודה אחת. ברגע שסוכן מסוג Enemy אוכל סוכן מסוג Agent הוא מפסיק לצבור נקודות.

התחלת משחק: הסוכנים ימוקמו בצורה אקראית בקצוות הלוח כל סוכן בעל שלושה חוליות

סיום המשחק: מתרחש כאשר שני הEnemies אוכלים את הסוכנים של קבוצה א'

כל הסוכנים יכולים לנוע לפי ה-MoveSet:

$$MoveSet = \{Up, Down, Left, Right\}$$

איסורים:

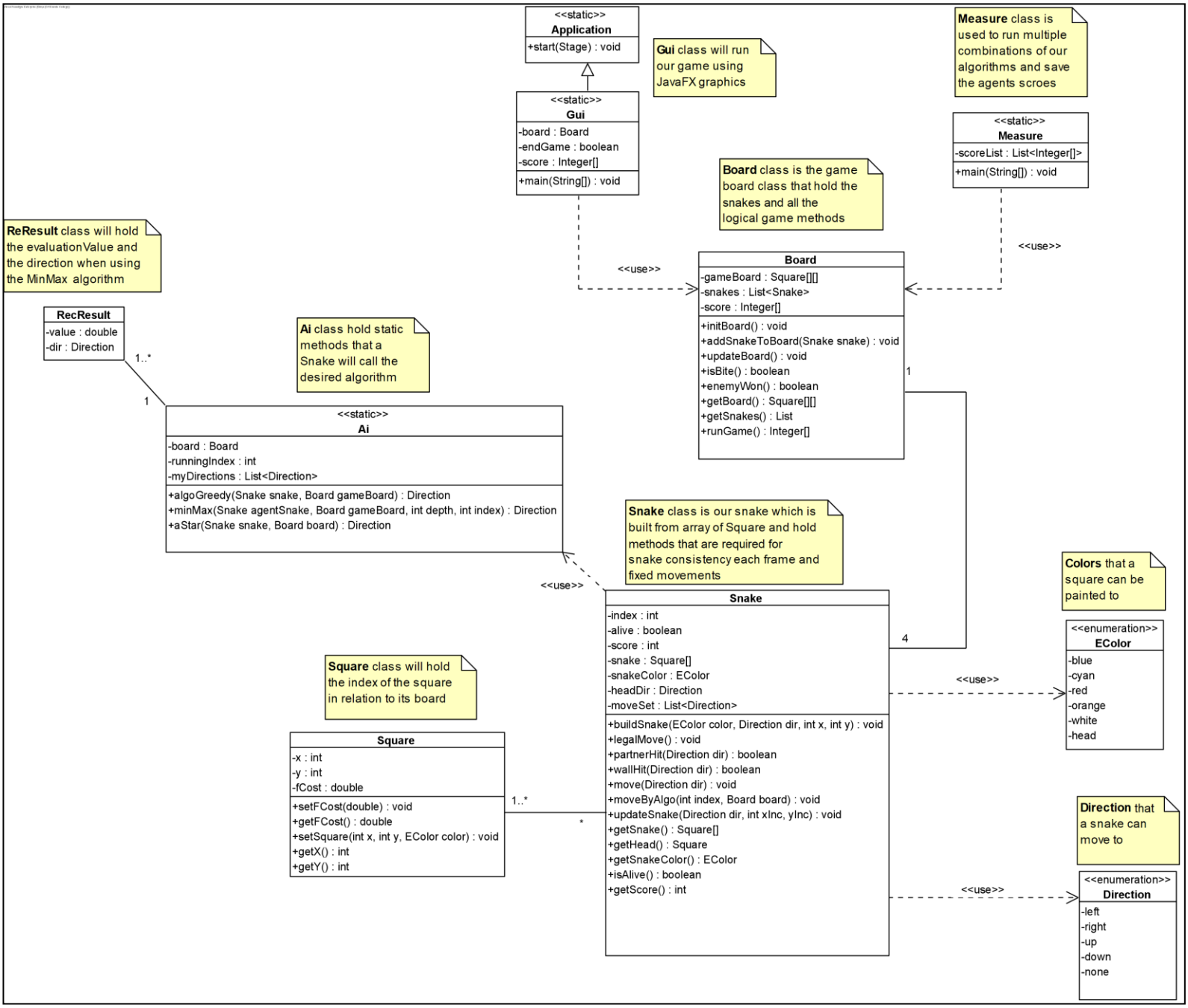
- אסור לצאת מגבולות הלוח
- נחש לא יכול לעמוד על אותה משבצת שהוא כבר נמצא בה
- סוכנים מאותה קבוצה לא יכולים להיות על אותה המשבצת

2 דרך הפתרון

2.1 סביבת המשחק

את סביבת המשחק מימשנו בסביבת Java בעזרת שימוש באובייקטים הבאים:

- Square מציין קורדינטות על הלוח וצבע
- Snake אוסף של Square בעל מטודות שונות לעקביות ונכונות המשחק
- Board אוסף של Square המייצג לוח NxN ומחזיק בתוכו אוסף של נחשים
- Ai אוסף של אלגוריתמים של בינה מלאכותית המשומש על ידי Snake



2.2 מימוש הבינה המלאכותית

על מנת שהבינה מלאכותית תבצע את תהליך קבלת ההחלטות שלה בחרנו באלגוריתמים שאותם נפרט בהמשך אשר מבוססים על חישובי מרחקים אשר למדנו בקורס.

בעזרת המרחקים הללו אנו יכולים לתת הערכה עד כמה הנחשים קרובים אחד לשני ובכך נוכל לתת לבינה הערכה כיצד יש להתנהג, כלומר עבור Agents מטרתם תהיה להגדיל את המרחק כמה שניתן מה- Enemies על מנת שיחיו יותר זמן, ועבור ה-Enemies ההפך.

החלטנו להשתמש בשני סוגי מרחקים:

1. **מרחק אוקלידי**

מרחק אוקלידי הינו מרחק בין שני וקטורים המחושב בנוסחה הבאה:

$$v_1, v_2 \in R^N$$

$$Distacne_{v_1,v_2} = \sqrt{\sum_{i=1}^{N} \left(v_{1_i} - v_{2_i}\right)^2}$$

2. **מרחק מנהטן**

מרחק הינו מרחק בין שני וקטורים המחושב בנוסחה הבאה:

$$v_1, v_2 \in R^N$$

$$Distacne_{v_1,v_2} = \sum_{i=1}^{N} |v_{1_i} - v_{2_i}|$$

במשחק הסנייק שלנו אנו מחשבים את המרחק האוקלידי/מנהטן בעזרת שימוש באובייקט Square אשר הינו וקטור בעל מימד 2.

2.2.1 אלגוריתם Greedy

אסטרטגיה

במשחק שלנו החלטנו להשתמש באלגוריתם חמדן על מנת להקנות לסוכנים Enemies את היכולת למצוא איזה פעולה יש לבחור על מנת להגיע אל הסוכן Agent בצורה יעילה.

אלגו' פועל בצורה כזו שעבור כל Square אשר ניתן להגיע אליו אחרי צעד אחד והוא חוקי (כפי שהוגדר באיסורים) מחשבים עליה את המרחק שלה בינה לבין ראש נחשי ה- Agents ובוחרים את המינימלי, לאחר חישוב כל ה Squares אנו בוחרים את ה Square עם הערך המינימלי.

בעזרת חישובים אלו אנו מעניקים לנחש את היכולת לבחור את הצעד שהכי ייקרב אותו ל Agent שהכי קרוב אליו ובכך לממש את מטרתו.

פסאודו - קוד

Function algoGreedy(Snake snake, Board gameBoard):

```
for each direction in directions do:
    if move in direction is valid then:
        move snake by direction
        if agent1 is alive AND agent2 is alive then:
            find minimum distance between agent1 and agent2
        if agent1 is dead AND agent2 is alive then:
            calculate distance to agent2
        if agent1 is alive AND agent2is dead then:
            calculate distance to agent1
        if agent1 is dead AND agent2 is dead then:
            return Direction.none
        update minimum distance and save the direction
return the direction that had the minimum distance
```

2.2.2 אלגוריתם A*

אסטרטגיה

במשחק שלנו החלטנו להשתמש באלגוריתם A* אשר הינו אלגוריתם חיפוש על מנת להקנות לסוכנים Enemies את היכולת למצוא מסלול קצר על מנת שיוכלו להגיע אל הסוכנים Agents.

באלגוריתם זה יש שימוש במשקלים H, G, F אשר הם:

- H = הינו המרחק בין הנקודה הנוכחית לבין ראש ה Agent הקרוב ביותר.
- G = הינו המרחק בין נקודה הנוכחית לבין הנקודה הקודמת.
- F = סכום של ערכי G ו- H.

בעזרתם שימוש בערכי F הסוכן יכול להרכיב מסלול מינימלי משום שהוא בוחר בכל פעם את ערכי F המינימלים בכדי להקטין את המרחק בינו לבין ה Agent.

לאחר מציאת המסלול, הסוכן מבצע את הפעולה הראשונה על מנת להתקדם אל המטרה, ולאחר כל צעד אנו מבצעים חישוב מסלול מחדש משום שנקודת המטרה השתנתה.

פסאודו - קוד

Function algoAStar(Square enemySnake, Square agentSnake, Board board):

```
reset F-Costs of all nodes in board
create openNodes
create closedNodes
create neighborNodes
create path
add enemySnake to openNodes
while openNodes is not empty do:
        currentNode = node in opendNodes with lowest F-Cost
        add currentNode to path
        add currentNode to closedNodes
        remove currentNode from openNodes
        if currentNode position  is the agentSnake  then:
            return path
        update neighborNodes as the valid neighbors of currentNode
        for each node in neighborNodes do:
            If openNode not contains node OR node F-Cost is larger then the new F-Cost then:
                update node F-Cost
            If openNode not contains node AND closedNodes not contains node
                add node to openNodes
return null
```

2.2.3 אלגוריתם MinMax

אסטרטגיה

בעזרת אלגוריתם MinMax אנו מקנים לסוכנים Agents את היכולת לנבא ביצד ה- Enemies יזוזו וינסו להכשיל אותם, ובעזרת ניבוי זה ה-Agent יוכל לקבל הערכה על איזה צעד לעשות כך שימקסם את יכולת הישרדותו.

אופן פעולת הניבוי מתבצע בצורה הבאה:

ה- Agent תמיד בוחר את הערכים המקסימלים בכדי לבחור את הפעולה אשר תרחיק אותו מה Enemies כמה שיותר בנוסף ה- Enemies תמיד יבחרו את הערכים המינימלים על מנת להכשיל את ה Agent.

את הערכים ניתן לקבל בעזרת evaluation function.

בסוף האלגורתים ה Agent מבצע את הצעד אשר העניק לו את הערך הכי גדול.

פסאודו - קוד

Function algoMinMax(Square agentHead, int depth, boolen isMax, List<Snake> snakes, int index):

```
1.  if depth == 0 OR agentSnakeDied then:
2.     return new RecResult(evaluationValue, Direction.none)
3.  create List evalArr
4.  if isMax then:
5.      finalResult.value = -infinity
6.      for each direction in directions do:
7.          if move in direction is valid then:
8.              Move snake by direction
9.              tempScore = minMax(agentHead, depth – 1, false, snakes, RED)
10.             add tempScore to evalArr
11.             for each recResult in evalArr do:
12.                 finalResult = max{ finalResult, recResult }
13. else if index == RED then:
14.     finalResult.value = infinity
15.     for each direction in directions do:
16.         if move in direction is valid then:
17.             Move snake by direction
18.             tempScore = minMax(agentHead, depth – 1, false, snakes, ORANGE)
19.             add tempScore to evalArr
20.             for each recResult in evalArr do:
21.                 finalResult = min{ finalResult, recResult }
22. if index == ORANGE then:
23.     finalResult.value = infinity
```

```
    for each direction in directions do:
        if move in direction is valid then:
            Move snake by direction
            tempScore = minMax(agentHead, depth – 1, true, snakes, agentIndex)
            add tempScore to evalArr
            for each recResult in evalArr do:
                finalResult = min{ finalResult, recResult }
return finalResult
```

Evaluation function

אסטרטגיה

את פונקצית הערכה עבור אלגו' MinMax מימשנו בצורה הבאה:

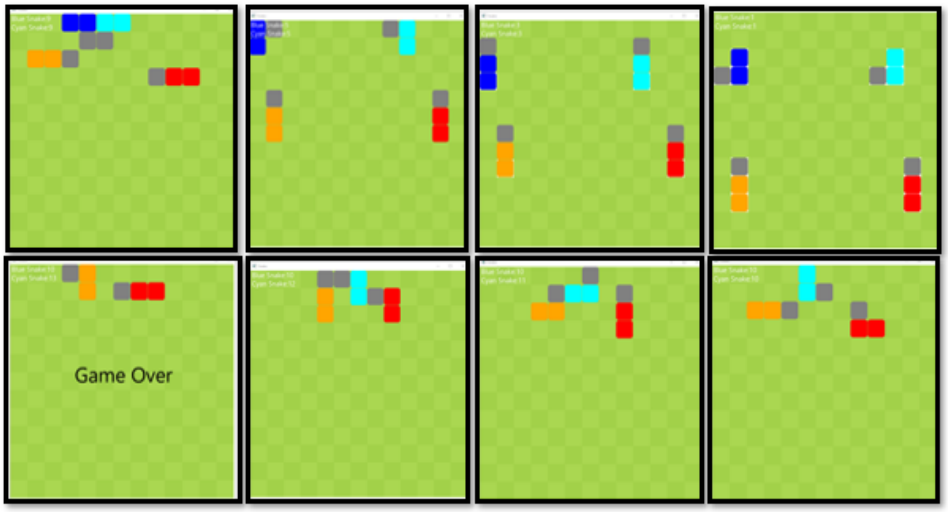
1. חישוב מרחק לפי גופים של ה- Enemies, כלומר אנו מחשבים את המרחק הממוצע של כל חוליות ה Enemy לבין ראש ה Agnet ובכך נותנים הערכה גסה על מיקום גוף ה-Enemy.
2. במידה ושני ה Enemies קרובים אחד לשני במרחק אשר קבענו, אנו מחשבים את המרחק הממוצע ביניהם על מנת לתת הערכה על מיקום גופם בלוח.
3. במידה ושני ה-Enemies רחוקים אחד לשני, אנו נבחר את המרחק המינמלי מבין שניהם על מנת להתרחק מה- Enemy הקרוב יותר.

קוד

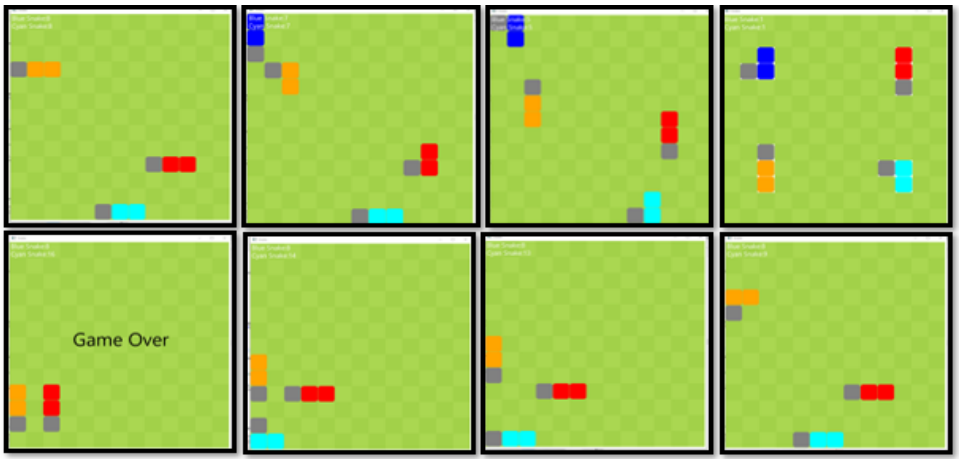
```
Function evaluationFunction(Square agentPosition, List<Snake> snakes):
    dist1 = calculateDistance(agentPosition, enemy1)
    dist2 = calculateDistance(agentPosition, enemy2)
    distEnemies = calculateDistance(enemy1, enemy2)
    if distEnemies < minimum_enemies_distance then:
        return (dist1 + dist2) / 2
    return min{dist1, dist2}
```

3 סימולציות

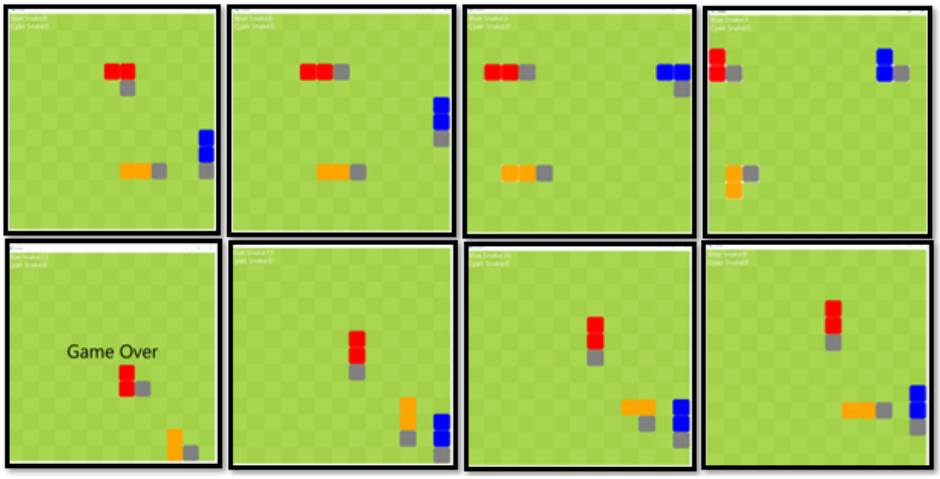
3.1 משחק מס' 1



3.2 משחק מס' 2



3.3 משחק מס' 3



4 תוצאות

4.1 אוסף של נתונים

את המשחק הרצנו בכמה קונפיגורציות שונות:

RRRR, RRAG, MRAG, MMAG, MMAA, MMGG.

כאשר כל קונפיגורציה רצה למשך 50 משחקים, ועבור כל קונפיגורציה אנו אוספים את ניקוד הסוכנים הממוצע ובנוסף בודקים אותם עם depth שונים וגם עם חישובי מרחק שונים.

ניתן לראות את התוצאות בטבלאות הבאות:

1. **שימוש בנוסחת מרחק אוקלידי**

	Agents		Enemy		
Depth	**Blue**	**Cyan**	**Red**	**Orange**	**AVG Score**
-	Random	Random	Random	Random	112
-	Random	Random	A-Star	Greedy	15
-	MinMax	Random	A-Star	Greedy	22
3	MinMax	MinMax	A-Star	Greedy	29
3	MinMax	MinMax	A-Star	A-Star	32
3	MinMax	MinMax	Greedy	Greedy	30
6	MinMax	MinMax	A-Star	Greedy	30
6	MinMax	MinMax	A-Star	A-Star	31
6	MinMax	MinMax	Greedy	Greedy	32
12	MinMax	MinMax	A-Star	Greedy	24
12	MinMax	MinMax	A-Star	A-Star	27
12	MinMax	MinMax	Greedy	Greedy	25

2. שימוש בנוסחת מרחק מנהטן

	Agents		Enemy		
Depth	**Blue**	**Cyan**	**Red**	**Orange**	**AVG Score**
-	Random	Random	Random	Random	120
-	Random	Random	A-Star	Greedy	21
-	MinMax	Random	A-Star	Greedy	36
3	MinMax	MinMax	A-Star	Greedy	72
3	MinMax	MinMax	A-Star	A-Star	63
3	MinMax	MinMax	Greedy	Greedy	74
6	MinMax	MinMax	A-Star	Greedy	71
6	MinMax	MinMax	A-Star	A-Star	84
6	MinMax	MinMax	Greedy	Greedy	80
12	MinMax	MinMax	A-Star	Greedy	74
12	MinMax	MinMax	A-Star	A-Star	68
12	MinMax	MinMax	Greedy	Greedy	64

4.2 מסקנות

- לאחר הוספת בינה ל-Enemies ניתן להבחין כי ניקוד ה- Agents ירד משמעותית לבין תזוזות אקריות של ה- Enemies, לפיכך נוכל לראות את האפקטיביות של האלגוריתמים אשר ממשים את מטרות ה- Enemies שהוגדרו.
- ניתן לראות כי אין הרבה הבדל בין האלגו' A* ו – Greedy מבחינת התוצאות, ובכך אנו יכולים להסיק כי עדיף יהיה להתשמש באלגו' Greedy משום שהוא צורך פחות כוח חישוב ומביא לאותן תוצאות.
- לאחר בדיקות של ערכי Depth שונים באלגו' MinMax אנו מסיקים כי אין הבדלים משמעותיים ביחסי התוצאות לכן יהיה אפשר להריץ את האלגו' בערך depth = 3 ובכך להגיע לאותן תוצאות בפחות כוח חישוב.
- ניתן לשפר את כוח החישוב של אלגו' MinMax בעזרת שימוש בטכניקת alpha beta pruning מפני שהריצה ב depth גבוהה נמשכת זמן רב.
- בהרצה השנייה עם שימוש בנוסחת מרחק מנהטן ניתן לראות שהניקוד הממוצע עלה פי 2, ניתן להסיק כי השימוש בנוסחת מנהטן הוא עדיף לפי התוצאות שהתקבלו. זאת מפני שמרחק מנהטן מחשב את המרחק הוא לפי צעדים על הלוח Left, Right ,Up, Down לעומת מרחק אוקלידי שהוא מחשב מרחק אוירי בלבד.