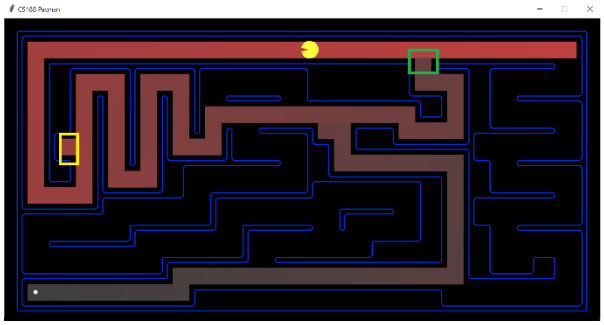
דו"ח מסכם לתרגיל 1 – בינה מלאכותית

איתמר לרדו – 311547087

שאלה 1 – מימוש DFS:

בשאלה זו מימשתי את אלגוריתם החיפוש DFS שסוקר לעומק. השתמשתי ברשימה ששומרת את הקודקודים שכבר פאקמן ביקר בהם על מנת להימנע ממעגלים ולולאות אינסופיות. כמבנה הנתונים השתמשתי במחסנית רגילה שניתנה בקובץ util.



נשים לב שסדר החיפוש הוא לא כפי שחשבתי.

בצומת שמסומן בריבוע ירוק, היה כדאי לפאקמן

לפנות כי ניתן לראות שזה מקצר את דרכו ליעד.

בנוסף, בצומת שמסומן בצהוב פאקמן לא באמת נכנס

אלה המשיך בדרכו צפונה – על אף שצומת זה נחקר.

בהרצת האלגוריתם זה, הפתרון שקיבלתי היה באורך 130. כמובן שזו לא העלות הנמוכה ביותר כפי שציינתי למעלה כי פאקמן האריך את דרכו לאחר שפסח על הפניה דרומה בצומת הירוק.

שאלה 2 – מימוש BFS:

Breadth-First-Search (Graph-search)

```
def BFS(problem):
    frontier = deque([Node(problem.s_start)]) # FIFO queue
    closed_list = set()
    while frontier:
        node = frontier.popleft()
        if problem.is_goal(node.state):
            return node.solution()
        closed_list.add(node.state)
        for child in node.expand(problem):
            if child.state not in closed_list and child
                not in frontier:
                frontier.append(child)
    return None
```

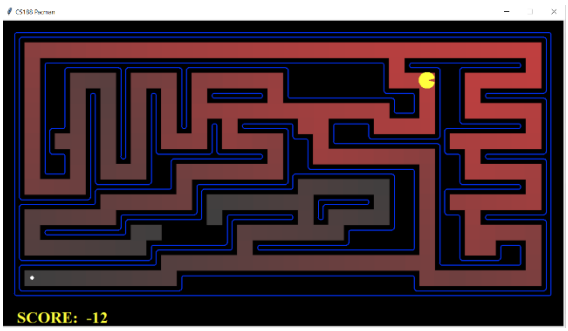
בהתבסס על הפסאדו קוד שלמדנו לאלגוריתם

חיפוש לעומק בגרפים (BFS עם רשימה שמונה את

הקודקודים שכבר ביקרנו בהם ומונעת לולאות),

כתבתי את האלגוריתם החיפוש BFS.

כעת, פאקמן פונה בצומת הראשון דרומה מה שניתן לראות מקצר את דרכו אל המטרה וכמובן מפחית את העלות הכוללת של הפתרון.



בשונה מ-DFS בו פאקמן מצא פתרון בעלות 130,

באלגוריתם זה פאקמן מגיע לפתרון בעלות של 68.

ניתן לשים לב להבדל נוסף, באלגוריתם BFS פאקמן מפתח

יותר צמתים מאשר באלגוריתם DFS (269 לעומת 146),

ניתן לשים לב לזה בכמות המסלולים המסומנים באדום

בעולם של פאקמן. דבר נוסף הוא שכיוון שהאלגוריתם שכתבתי גנרי, הקוד שלי עבד גם לבעיית החיפוש "the eight puzzle".

שאלה 3 – מימוש UCS:

כעת האלגוריתם UCS יתחזק תור עדיפויות. נעבור על הצמתים החדשים ונבדוק אם ביקרנו בהם אז לא נכניס אותם לתור ואם הם כבר נמצאים בתור והעלות של הבן החדש טובה יותר נחליף בניהם. כך נקבל

עלויות טובות יותר לבעיית החיפוש. לכל איבר ב-frontier שמרתי כזוג את הקורדינטה של המיקום של פאקמן והמסלול, וכן גם את העלות. ((coordinate, path), cost).

כאשר הרצתי את הסוכן StayEastSearchAgent קיבלתי עלות של 1 למציאת המזון ואילו ב-StayWestSearchAgent העלות הכוללת של החיפוש הייתה 68719479864.

שאלה 4 – מימוש A*:

מימוש האלגוריתם Astar משתמש בתור עדיפויות במבנה הנתונים.בנוסף, Astar מקבלת פונקציה היוריסטית ומחשב את ההיוריסטיקה שזוהי הערכה לעלות המטרה.

כאשר הרצתי את Astar ו UCS על בעיית החיפוש bigMaze קיבלתי את התוצאות הבאות:

```
[redacted] python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 549
```

Astar

```
[redacted] python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=ucs,heuristic=manhattanHeuristic
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 620
```

UCS

כעת, לאחר מימוש ארבעת האלגוריתם ניתן לבחון את יעילותם על אותו עולם של פאקמן.

האלגוריתם	המבוך	כמות צמתים שפיתח	עלות
BFS	openMaze	682	54
DFS	openMaze	576	298
UCS	openMaze	682	54
Astar	openMaze	535	54

וניתן לראות ש-Astar הוא המובחר מבניהם.

שאלה 5 – מציאת כל הפינות

במימוש הבעיה של מציאת כל הפינות הגדרתי את ה- state באופן הבא: $state = (startingPosition, goals\_list)$. את ה-goals_list הגדרתי להיות ארבעת הפינות. כאשר פאקמן יבקר בפינה, אותה פינה תמחק מרשימת המטרות. כאשר הרשימה תהיה ריקה אדע שפאקמן ביקר בארבעת הפינות וכך הגיע למטרה הסופית וניצח.

כאשר הרצתי את האלגוריתם BFS על בעיית הפינות פותחו 1966 צמתים והעלות הכוללת הייתה 106. בהרצת Astar על אותה בעיה, נפתחו 692 צמתים והעלות הכוללת הייתה גם 106.

שאלה 6 – היוריסטיקה לבעיית הפינות

כיוון שהגדרתי את המצב להיות הזוג $(startingPosition, goals\_list)$ וה-goals_list שלי מחזיקה את הפינות שעדיין פאקמן לא ביקר בהן, בניתי את ההיוריסטיקה באופן הבא:

הכנסתי לרשימה את כל הפינות שעדיין פאקמן לא ביקר בהן. כל עוד הרשימה לא ריקה, נבחר את הפינה שהמרחק אליה הוא הנמוך ביותר. את חישוב המרחק נבצע באמצעות הפונקציה util.manhattanDistance שעל נכונותה דיברנו בשיעור. נוסיף מחרק זה להיריסטיקה ולבסוף נחזיר

את האומדן הזה. הנכונות, נובעת מנכונות מרחק מנהטן. לכל צעד המרחק אי שלילי, במטרה מן ה-goals_list המרחק הוא 0, וההיוריסטיקה מקיימת את אי שוויון המשולש – המונוטוניות.

כאשר הרצתי את $mediumCorner - p\ AStarCornersAgent$ קיבלתי 692 צמתים שפותחו בעלות כוללת של 106.

<u>שאלה 7 – אכילת כל הנקודות</u>

בשאלה זו מימשתי את הפונקציה ההיוריסטית שמעריכה את המרחק לנקודות.

בתור התחלה קיבלתי את האוכל בגריד והמרתי את זה לרשימה של נקודות שעל פאקמן לאסוף. כעת נרצה למיין אותם לפי עלות. לכן, לכל נקודה מן הנקודות שעל פאקמן לאכול, נבדוק מהו מרחקה מן המיקום הנוכחי של פאקמן. את הבדיקה נעשה באמצעות mazeDistance שמוגדרת בתרגיל. נבחר את הנקודות בעלות המרחק הגבוה ביותר וכך ניצור מונוטוניות יורדת כלפי 0 – היעד.

כאשר הרצתי את python pacman.py -l trickySearch -p AStarFoodSearchAgent קיבלתי תוצאה טובה של 4137 צמתים שפותחו בעלות כוללת של 60.