בס"ד

תאריך הגשה: 8.11.21

תרגיל 1 – בעיות חיפוש

pacman

הקדמה

בפרויקט זה, סוכן הפקמן שלך ימצא דרכים בעולם המבוך שלו, הן כדי להגיע למיקום מסוים והן לאסוף מזון ביעילות. תוכל לבנות אלגוריתמי חיפוש כלליים וליישם אותם על תרחישי פאקמן.

הקוד לפרויקט זה מורכב מכמה קבצי Python, חלקם תצטרכו לקרוא ולהבין על מנת להשלים את התרגיל, ומחלקם תוכלו להתעלם. כל קבצי הקוד נמצאים בתיקית zip ב classroom

קבצים לעריכה:

search.py – כאן תערוך את אלגוריתמי החיפוש שלך.

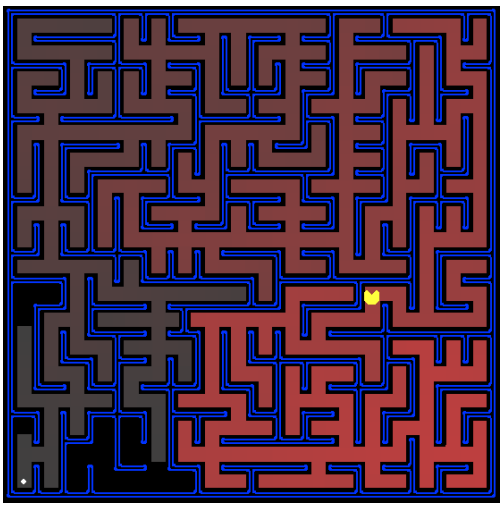
searchAgents.py – כאן תערוך את הסוכנים המבוססים על חיפוש.

קבצים שכדאי להסתכל עליהם:

pacman.py - הקובץ הראשי שמריץ משחקי פאקמן. קובץ זה מתאר את ה Pacman GameState, שבו אתה משתמש בפרויקט זה.

game.py – לוגיקת המשחק של עולם הפקמן. קובץ זה מתאר מספר טיפוסים תומכים כמו AgentState, Agent, Direction ו- Grid.

util.py - מבני נתונים שימושיים ליישום אלגוריתמי חיפוש.



קבצי תמיכה שניתן להתעלם מהם:

graphicsDisplay.py - גרפיקה

graphicsUtils.py - תמיכה בגרפיקת Pacman

textDisplay.py - גרפיקת ASCII עבור פאקמן

ghostAgents.py - סוכנים לשליטה ברוחות רפאים

keyboardAgents.py - ממשקי מקלדת לשליטה בפקמן

layout.py - קוד לקריאת קבצי פריסה ואחסון תוכנם

הגשה

במהלך התרגיל תערוך חלקים של **search.py** ו- **searchAgents.py**. עליך לשלוח קבצים אלה עם הקוד והערות שלך. נא לא לשנות את הקבצים האחרים או לשלוח אף אחד מהקבצים המקוריים מלבד קבצים אלה.
בראש כל אחד מקבצים אלו נא לכתוב את שם הסטודנט ות.ז.

בנוסף לקבצים אלו עליך להגיש דו"ח העונה על שאלות בתרגיל בקובץ בשם **report_<id>.pdf**, כאשר id יהיה ת.ז. של הסטודנט וכן קובץ פרטים אישיים בשם detail.txt והוא יכיל את שם הסטודנט בשורה הראשונה ות.ז. בשורה השניה.

ההגשה דרך ה Moodle בלבד!! לא יתקבלו הגשות ב classroom משום סיבה.

ברוך הבא ל pacman!

לאחר הורדת קבצי התרגיל, תוכל להתחיל לשחק על ידי הקלדת השורות הבאות:

```
python pacman.py
```

פאקמן חי בעולם כחול נוצץ של מסדרונות מתפתלים ופינוקים עגולים וטעימים. ניווט בעולם הזה ביעילות יהיה הצעד הראשון של פאקמן בשליטה על עולמו.

הסוכן הפשוט ביותר ב- searchAgents.py נקרא GoWestAgent, שתמיד עובר מערבה (סוכן טריוויאלי). סוכן זה יכול מדי פעם לנצח:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

אבל הדברים הופכים בעייתיים כשנדרשות פניות..

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

כאשר pacman נתקע, ניתן לצאת מהמשחק על ידי לחיצה על CTRL-c ב terminal או לחיצה על ה X בצד העליון של המשחק.

שימו לב ש pacman.py תומך במספר אופציות שכל אחת מהן יכולה להיות מיוצגת בצורה ארוכה (למשל: `--layout`) או קצרה (למשל: l-) ניתן לראות את הרשימה המלאה על ידי הקשת הפקודה:

```
python pacman.py -h
```

הפקודות מופיעות גם בקובץ commands.txt.

חיפוש

ב- searchAgents.py תמצא SearchAgent המיושם במלואו, המתכנן מסלול דרך העולם של פאקמן ולאחר מכן מבצע את הנתיב הזה צעד אחר צעד. אלגוריתמי החיפוש ליצירת מסלול אינם מיושמים - זה התפקיד שלך.

ראשית, בדוק ש- SearchAgent פועל כראוי על ידי הפעלה:

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
```

הפקודה למעלה אומרת ל- SearchAgent להשתמש ב- tinyMazeSearch כאלגוריתם החיפוש שלו, המיושם ב- search.py. בעזרותו פאקמן מנווט במבוך בהצלחה.
עכשיו הגיע הזמן לכתוב פונקציות חיפוש גנריות מן המניין שיעזרו לפקמן לתכנן מסלולים! זכור שצומת חיפוש חייב להכיל לא רק מצב אלא גם את המידע הדרוש לשחזור הנתיב (המסלול) שמגיע לאותו מצב.

<u>הערה חשובה:</u> כל פונקציות החיפוש שלך צריכות להחזיר רשימה של פעולות שיובילו את הסוכן מההתחלה ליעד. כל הפעולות האלה צריכות להיות מהלכים חוקיים (כיוונים תקפים, אין מעבר דרך קירות).

<u>הערה חשובה:</u> הקפד להשתמש במבני הנתונים Stack, Queue ו- PriorityQueue המסופקים לך ב- util.py!

שאלה 1 – מימוש DFS (15 נקודות)

ממש את אלגוריתם החיפוש depth-first-graph-search (DFS) בפונקציית deepFirstSearch בקובץ search.py. כדי להשלים את האלגוריתם שלך, כתוב את גרסת החיפוש בגרף של DFS, המונעת הרחבת מצבים שכבר ביקרת בהם.

הקוד שלך אמור למצוא במהירות פתרון ל:

```
python pacman.py -l tinyMaze -p SearchAgent

python pacman.py -l mediumMaze -p SearchAgent

python pacman.py -l bigMaze -z .5 -p SearchAgent
```

לוח Pacman יציג "שכבה" של המצבים שנחקרו, והסדר בו נחקרו (אדום בהיר יותר פירושו חקר קודם).

1.1) האם סדר החיפושים הוא מה שהיית מצפה? האם פאקמן באמת הולך לכל הצמתים שנחקרו בדרך לנקודת הסיום? (ציין תשובותך בדו"ח)

רמז: אם אתה משתמש ב- Stack כמבנה הנתונים שלך, הפתרון שנמצא על ידי אלגוריתם DFS שלך עבור mediumMaze צריך להיות באורך של 130 (בתנאי שתכניס את הבנים בסדר בו הם התקבלו ב getSuccessors; ייתכן שתקבל 246 אם תכניס אותם בסדר ההפוך).

1.2) האם זהו פתרון בעלות נמוכה ביותר? אם לא, חשוב מה החיפוש הראשון DFS עושה לא בסדר. (ציין תשובתך בדו"ח)

שאלה 2 - מימוש BFS (15 נקודות)

ממש את אלגוריתם breadth-first-graph-search (BFS) בפונקציה breedthFirstSearch בקובץ search.py. שוב, כתוב אלגוריתם לחיפוש גרפים המונע הרחבת מצבים שכבר ביקרת בהם. בדוק את הקוד שלך כפי שעשית עבור חיפוש ראשון לעומק.

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

האם BFS מוצא פתרון בעלות נמוכה יותר? אם לא, בדוק את היישום שלך.

רמז: אם pacman נע לאט מדי עבורך, נסה את האפשרות

```
--frameTime 0
```

הערה: אם כתבת את קוד החיפוש שלך באופן כללי, הקוד שלך אמור לפעול באותה מידה עבור בעיית החיפוש "the eight puzzle" ללא שינויים. תוכל לבדוק זאת על ידי הרצת הפקודה:

```
python eightpuzzle.py
```

שאלה 3 – מימוש UCS (15 נקודות)

בעוד BFS ימצא את המסלול הקטן ביותר למטרה, ייתכן שתרצה למצוא מסלולים שהם "הטובים ביותר" במובנים אחרים.

נסתכל על mediumDottedMaze ו- mediumScaryMaze.

על ידי שינוי פונקציית העלות, אנו יכולים לעודד את פאקמן למצוא נתיבים שונים. לדוגמה, אנו יכולים לגבות פחות על צעדים באזורים "עתירי מזון", וסוכן פאקמן רציונלי צריך להתאים את התנהגותו בתגובה.

ממש את אלגוריתם uniform-cost graph search בפונקציית uniformCostSearch בקובץ search.py.
אנו ממליצים לך לחפש בקובץ util.py מבני נתונים שעשויים להיות שימושיים ביישום שלך.

כעת עליך לצפות בהתנהגות מוצלחת בכל שלושת הלוחות הבאים, כאשר הסוכנים להלן כולם סוכני UCS, השונים רק בפונקציית העלות שבה הם משתמשים (הסוכנים ופונקציות העלות נכתבו עבורך):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

הערה: אתה צריך לקבל עלויות מאוד נמוכות וגבוהות מאוד עבור StayEastSearchAgent ו- StayWestSearchAgent בהתאמה, בשל פונקציות העלות האקספוננציאליות שלהן (הסתכל בקובץ searchAgents.py לפרטים).

שאלה 4 – מימוש A* (15 נקודות)

ממש חיפוש A* בפונקציה הריקה aStarSearch בקובץ search.py.
A* מקבלת פונקציה היוריסטית כפרמטר. ההוריסטיקה מקבלת שני ארגומנטים: המצב בבעיית החיפוש (הארגומנט העיקרי) והבעיה עצמה (למידע). הפונקציה היוריסטית nullHeuristic בקובץ search.py היא דוגמה טריוויאלית.

אתה יכול לבדוק את יישום A* שלך על הבעיה המקורית של מציאת נתיב דרך מבוך למיקום קבוע באמצעות היוריסטיקת מרחק מנהטן (מיושמת כבר כ manhattanHeuristic ב- searchAgents.py).

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
```

רמז: A* מוצא את הפיתרון האופטימלי מעט מהר יותר מאשר UCS. במידה והפיתרון שלך נכון, A* יפתח 549 צמתים לעומת 620 ש UCS יפתח.
4.1) מה קורה ב openMaze באסטרטגיות החיפוש השונות?

שאלה 5 – מציאת כל הפינות (10 נקודות)

הכח האמיתי של A* יתגלה רק עם בעיית חיפוש מאתגרת יותר. לשם כך, ננסח בעיה חדשה ונעצב עבורה היוריסטיקה.

במבוכי pacman יש ארבע נקודות, אחת בכל פינה. בעיית החיפוש החדשה שלנו היא למצוא את הדרך הקצרה ביותר במבוך שנוגעת בכל ארבע הפינות (בין אם יש שם אוכל ובין אם לאו).

שים לב שאצל כמה מבוכים כמו tinyCorners, הדרך הקצרה ביותר לא תמיד עוברת קודם דרך האוכל הקרוב ביותר! (רמז: הדרך הקצרה ביותר דרך tinyCorners לוקחת 28 צעדים.)

הערה: הקפד להשלים את שאלה 2 לפני עבודה על שאלה 5, מכיוון ששאלה 5 מבוססת על התשובה שלך לשאלה 2.

ממש את בעיית החיפוש CornersProblem בקובץ searchAgents.py. יהיה עליך לבחור ייצוג של המצב המקודד את כל המידע הדרוש כדי לזהות אם הושגו כל ארבע הפינות. כעת, אלגוריתם החיפוש שלך צריך לפתור:

```
python pacman.py -l tinyCorners -p SearchAgent -a
fn=bfs,prob=CornersProblem

python pacman.py -l mediumCorners -p SearchAgent -a
fn=bfs,prob=CornersProblem
```

כדי לקבל ניקוד מלא, עליך להגדיר ייצוג של מצב שאינה מקודדת מידע לא רלוונטי (כמו המיקום של רוחות רפאים במידה ויש, היכן נמצא מזון נוסף וכו '). בפרט, אין להשתמש ב- Pacman GameState כמצב חיפוש. הקוד שלך יהיה מאוד מאוד איטי אם תעשה זאת (וגם שגוי).

רמז: החלקים היחידים של מצב המשחק שאתה צריך להתייחס אליהם ביישום הם עמדת פקמן ההתחלתית ומיקום ארבע הפינות.

המימוש של BFS מרחיבה פחות מ -2000 צמתים לחיפוש ב- mediumCorners. עם זאת, היוריסטיקה (בשימוש בחיפוש A*) יכולה להפחית את כמות החיפושים הנדרשת.

5.1) תאר בדו"ח את המצב שבחרת לייצג.

<u>שאלה 6 – בעיית הפינות – היוריסטיקה (15 נקודות)</u>

הערה: הקפד להשלים את שאלה 4 לפני עבודה על שאלה 6, מכיוון ששאלה 6 נבנית על התשובה שלך לשאלה 4.

ממש היוריסטיקה לא טריוויאלית ועקבית (consistent) לבעית הפינות ב - cornersHeuristic.

```
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
```

הערה AStarCornersAgent: הוא קיצור דרך עבור:

```
-p SearchAgent -a
fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic.
```

קבילות (admissible) מול עקביות (consistent): זכור, היוריסטיקה היא רק פונקציות שלוקחות מצבי חיפוש ומחזירים מספרים המעריכים את העלות ליעד הקרוב ביותר. היוריסטיקה יעילה יותר תחזיר ערכים קרוב יותר לעלות המטרה בפועל. כדי להיות קבילה, הערכים ההוריסטיים חייבים להיות בעלי ערך של הגבול הנמוך למסלול הקצר ביותר בפועל אל היעד הקרוב ביותר (ולא שלילי). כדי להיות עקבי, הוא חייב גם לקבוע שאם פעולה עלתה c, אמצעי הפעולה יכול לגרום רק לירידה בהיוריסטיקה של c לכל היותר.

זכור, קבילות (admissible) אינה מספיקה להבטחת נכונות בחיפוש הגרפים - אתה זקוק למצב עקבי (consistent) יותר חזק. עם זאת, היוריסטיקה קבילה היא בדרך כלל גם עקבית. ברגע שיש לך היוריסטיק קבילה שעובדת טוב, אתה יכול לבדוק אם היא אכן עקבית. הדרך היחידה להבטיח עקביות היא באמצעות הוכחה.

ציון: ההוריסטיות שלך חייבת להיות היוריסטיקה עקבית (consistent) לא שלילית כדי לקבל נקודות.
וודא כי ההוריסטיות שלך מחזירה 0 בכל מצב יעד ולעולם לא מחזירה ערך שלילי. הציון יתקבל בהתאם לכמות צמתים שהיוריסטיקה שלך מרחיבה (באופן יחסי לניקוד השאלה):

Number of nodes expanded	Grade
more than 2000	0/3
at most 2000	1/3
at most 1600	2/3
at most 1200	3/3

זכור: אם ההוריסטיות שלך אינה עקבית(consistent), לא תקבל ניקוד!

6.1) תאר הפו' היוריסטית שבחרת, והסבר את נכונותה.

שאלה 7 – אכילת כל הנקודות (15 נקודות)

עכשיו נפתור בעיית חיפוש קשה: אכילת כל הנקודות האוכל של פאקמן בכמות צעדים מינימלית.
לשם כך נזדקק להגדרת בעיית חיפוש חדשה המסדירה את בעיית ניקוי המזון: FoodSearchProblem ב- searchAgents.py (מיושמת עבורך).
הפתרון מוגדר להיות מסלול שאוסף את כל המזון בעולם הפקמן. בתרגיל זה, הפתרונות תלויים רק במיקום הקירות, מזון רגיל ופקמן.
אם כתבת נכון את שיטות החיפוש הכלליות שלך, A* עם היוריסטיקה שווה אפס (שווה ערך לחיפוש בעלות אחידה) מציאת פתרון אופטימלי ל testSearch תהיה מהירה מאוד ללא שינוי קוד מצידך (עלות כוללת של 7).

```
python pacman.py -l testSearch -p AStarFoodSearchAgent
```

הערה: AStarFoodSearchAgent הוא קיצור דרך ל-

```
-p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic
```

תגלה כי UCS מתחיל להאט אפילו ב tinySearch הפשוט לכאורה. לשם היחס, היישום שלנו לוקח 2.5 שניות למצוא נתיב באורך 27 לאחר הרחבת 5057 צמתי חיפוש.

הערה: הקפד להשלים את שאלה 4 לפני עבודה על שאלה 7, מכיוון ששאלה 7 מבוססת על התשובה שלך לשאלה 4.

ממש את foodHeuristic ב searchAgents.py עם פו' היוריסטית קונסיסטנטית (consistent) עבור FoodSearchProblem. נסה את הסוכן שלך בלוח החיפוש המסובך:

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
```

וודא כי ההוריסטיות שלך מחזירה 0 בכל מצב יעד ולעולם לא מחזירה ערך שלילי. הציון יתקבל בהתאם לכמות צמתים שהיוריסטיקה שלך מרחיבה (באופן יחסי לניקוד השאלה):

Number of nodes expanded	Grade
more than 15000	1/4
at most 15000	2/4
at most 12000	3/4
at most 9000	4/4 (full credit; medium)
at most 7000	5 points (optional extra credit; hard)

זכור: אם ההוריסטיות שלך אינה עקבית (consistent), לא תקבל ניקוד!

7.1) תאר הפו' היוריסטית שבחרת, והסבר את נכונותה.

בהצלחה רבה!