שם: עמית סידס. תעודת זהות: 206768780. קורס: בינה מלכותית 20551

השתמשתי ב- Python 2.7.16.

שאלה 1

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l bigMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300

C:\Repos\AI\mmn11\search>python2 pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
```

א. האם סדר הסריקה תואם לסדר שהייתם מצפים ?

כן. ניתן לראות שבכל פיצול במבוך, השחקן בוחר לנוע כל פעם בכיוון מסוים ע"פ ההעדפה קבועה (שמאלה > ימינה > למטה > למעלה). כלומר, לדוגמא אם השחקן מגיע לפיצול לימין ולשמאל (והוא הגיע מלמעלה או למטה) אז הוא יבחר קודם ללכת שמאלה, ורק אחרי שהוא בדק את כל אפשרויות המסלולים בכיוון הזה, הוא "ינסה" ללכת ימינה. בדיוק כפי שהיינו מצפים מ-DFS.

ב. האם זה פתרון זול ביותר? נמקו.

לרוב לא. הפתרון הראשון שנמצא הוא זה שיבחר, למרות שיכול להיות שבהסתעפות כלשהיא בהתחלה, במידה והייתה נבחרת פנייה אחרת היינו יכולים לקבל פתרון יותר טוב. דוגמא טובה לכך היא המסלול ב-mediumSearch שבו השחקן בוחר במסלול שהולך לכיוון שמאל (כי הוא מתעדף שמאל על פני כל כיוון אחר) והפתרון שהוא מוצא הוא ממש לא אופטימאלי.

שאלה 2

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l bigMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300

C:\Repos\AI\mmn11\search>python2 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
```

ג. האם אלגוריתם חיפוש לרוחב מוצא פתרון זול ביותר? נמקו .

כן. BFS מבטיח לנו את מציאת המסלול הכי אופטימלי, משום שהוא סורק באופן מקבילי את כל המסלולים האפשריים (סריקה לרוחב העץ) ולכל המסלולים באותה רמה יש אותו cost. לכן, המסלול הראשון שימצא יהיה המסלול ברמה הנמוכה ביותר שמגיע ליעד. מכיוון שה-cost רק עולה ככל שהרמה עולה, המסלול הזה יניב את הפתרון האופטימלי.

שאלה 3

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 268
Pacman emerges victorious! Score: 442
```

שם: עמית סידס. תעודת זהות: 206768780. קורס: בינה מלכותית 20551

שאלה 4

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l bigMaze -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 0.0 seconds
Search nodes expanded: 538
Pacman emerges victorious! Score: 300
```

ד. מה קורה ב-openMaze לאסטרטגיות החיפוש השונות?

1. DFS

כיוון ש-DFS מתעדף פניות לצדדים על פני פניות מטה, הוא מוצא מסלול ממש גרוע עבור openMaze שבו כמעט ואין קירות. המסלול שלו גורם לו ללכת מקצה אחד לקצה השני של המבוך ורק אז להחליט לרדת שורה לכיוון היעד.

2. BFS

כמובן שהמסלול שנמצא הוא המסלול האופטימלי, אך מכיוון שהיעד נמצא בנק' הכי רחוקה מהמיקום ההתחלתי של השחקן, אז כל מיקום במבוך נסרק במהלך החיפוש (ניתן לראות שכל משבצת נצבעת במפה). מה שיוצר הרבה מאוד צמתים.

3. UCS

באופן לא מפתיע, התוצאות המוחזרות הן זהות לאלו של BFS, וזאת משום שה-cost לכל צומת בגרף (צעד במסלול) הוא 1. ועל כן גם המסלול זהה לזה של BFS, וכך גם החיפוש שסורק את כל המבוך.

4. A*

כאשר משתמשים ב-a* עם פונקציית היוריסטיקה nullHeuristic, החיפוש זהה לזה של BFS, כיוון שאין העדפה לשום צומת. אבל, כאשר משתמש בפונקצייה manhattanHeuristic, ניתן לראות שהמסלול הנמצא הוא אופטימלי (זהה לזה של BFS) אך בכשליש מהצמתים שנדרשו לחיפוש ב-BFS. ברגע ש-a* הצליח למצוא את המסלול שמתגבר על אחד המכשולים בדרך ליעד, הוא ישר חיפש האם הוא יכול להגיע ליעד ממנו, מבלי לסרוק עוד מסלולים מיותרים.

שאלה 5

בחרתי לייצג מצב בתור נקודה שמייצגת את המיקום הנוכחי ורשימה של הפינות שעדיין לא ביקרנו בהן. בכל פעם שנבקר בפינה מסוימת, נסיר אותה מהרשימה. אם הרשימה ריקה, הרי שלא נותרו עוד פינות והגענו ליעד.

```python
def getStartState(self):
  "Returns the start state (in your state space, not the full Pacman state space)"
  return self.startingPosition, list(self.corners)

def isGoalState(self, state):
  "Returns whether this search state is a goal state of the problem"
  return len(state[1]) == 0  # If number of corners left is 0, we reached the goal!
```

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
```

שם: עמית סידס. תעודת זהות: 206768780. קורס: בינה מלכותית 20551

שאלה 6

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l mediumCorners -p AStarCornersAgent
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 741
Pacman emerges victorious! Score: 434
```

האלגוריתם כאן הוא די פשוט. מחשבים את מרחק מנהטן המינימלי על מנת להגיע לכל ארבעת הפינות מהמיקום הנוכחי. כיוון שמרחק מנהטן לא מתייחס לקירות, הרי שהמרחק המינימלי האמיתי לא יכול להיות קטן מזה.

ה. האם היוריסטיקה קבילה? כן.

הפונקציה *smallest_distance_of_4_points* מקבלת את המיקום הנוכחי ו-iterable המכיל 4 (או פחות) נקודות. היא מחזירה את אורך מסלול מנהטן הטוב ביותר (המסלול הקצר ביותר ללא מכשולים) אשר עובר בכל ארבעת הנקודות ומתחיל במיקום הנוכחי. הפונקציה הזו היא הדבר הכי אופטימי שקיים, היא מניחה שהיא תצליח ללכת לכל מקום אשר תבחר מבלי להתייחס לקירות. לפיכך, האורך שהיא תחזיר יהיה בהכרח קטן/שווה לאורך המסלול האופטימלי (שעלול להתארך במידה ויש קירות) ולכן הפונקציה הזו היא קבילה.

שאלה 7

עבור שאלה זו מצאתי 2 אלגוריתמים כמעט זהים (שונים רק בפרמטר יחיד) שלכל אחד מהם יתרון וחסרון. האלגוריתם הכללי כולל פרמטר בשם amount, שמייצג מספר של נקודות אוכל (בין 1 ל-4). הפונקציה היוריסטית מחפשת קבוצת נקודות אוכל בגודל amount שהכי קשה להגיע אליהן (כנראה). הכוונה בכך זה שהיא בודקת את כל קבוצות הנקודות האפשריות בגודל amount ומחשבת עבור כל קבוצה את מרחק מנהטן הקטן ביותר שמגיע לכל הנקודות בקבוצה. לאחר מכן, מכל המרחקים הללו האלגוריתם בוחר את המרחק הגדול ביותר שחושב, שמייצג את קבוצת הנקודות שהכי קשה להגיע אליה (כנראה, בהנתן שקירות המבוך לא יפריעו).

שני האלגוריתמים משתמשים בערך שונה עבור הפרמטר amount:

1. האלגוריתם הראשון, שאיתו הגשתי את המטלה (ע"פ המלצה שלך), משתמש בערך קבוע עבור amount והוא 3. הפרמטר הזה מוצא מסלול באורך 60 בכ-6 שניות בערך, בפיתוח של כ-7040~ צמתים.
2. האלגוריתם השני, שנמצא בקוד ב-comment (בדיוק במקום שבו מוגדר amount=3), משתמש בערך דינאמי עבור amount ומשתנה בין 3 ל-4. הפרמטר הזה מוצא מסלול באורך 60 ב-12 שניות בערך, בפיתוח של כ-6980~ צמתים.

```
"""
# I discovered that after some point, it's faster to check larger combinations,
# So, when our food list gets smaller than 9, we check combinations of 4 instead of 3.
amount = 3 if len(food_list) > 9 else 4
"""
amount = 3  # Using fixed amount (3) gives an optimal solution in about 6 seconds with ~7040 nodes.
            # Using dynamic amount (like in the previous commented line) gives an optimal solution in about 12 seconds with ~6980 nodes.
```

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 5.8 seconds
Search nodes expanded: 7038
Pacman emerges victorious! Score: 570

C:\Repos\AI\mmn11\search>
```

```
amount = 3
```

```
C:\Repos\AI\mmn11\search>python2 pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 12.0 seconds
Search nodes expanded: 6979
Pacman emerges victorious! Score: 570

C:\Repos\AI\mmn11\search>
```

```
amount = 3 if len(food_list) > 9 else 4
```

המשך שאלה 7

ו. האם היוריסטיקה קבילה? כן.
הפונקציה *hardest_points_to_reach* מקבלת את המיקום הנוכחי, מערך נקודות ומספר 'amount' שמייצגת כמות מספרית של נקודות בין 0 ל-4. הפונקציה מחפשת את קומבינציית הנקודות (כלומר, אם amount הוא 4, היא מחפשת קומבינצייה של 4 נקודות) שהכי קשה לעבור בכולן ומחזירה את מסלול מנהטן האופטימלי שעובר בכולן מהמיקום הנוכחי (ללא הפרעה של קירות). בהנתן שיש יותר מ-amount נקודות במערך הנקודות, בהכרח האורך הזה הוא שווה/קטן מהמסלול האופטימלי, כיוון שהמסלול האופטימלי יצטרך גם הוא לעבור באותן נקודות ואפילו בעוד נקודות, ולכן ככל הנראה המסלול האופטימלי יהיה ארוך יותר (ישנם מקרים שבהם הוא יהיה זהה, זה קורה כאשר כל הנקודות הן על אותו הקו לדוגמא. אבל תמיד הפונקציה תחזיר מספר שהוא קטן/שווה לאורך המסלול האופטימלי).

ז. האם היוריסטיקה עקבית? כן.
נתחיל בכך שה-cost של כל פעולה יחידה (כלומר, צעד במבוך) הוא 1 תמיד. לכן, צריך בעצם להראות כי פונקציית היוריסטיקה אינה קטנה ביותר מ-1 במעבר בין משבצות סמוכות.
אתייחס קודם למקרה שבו רשימת הנקודות לא השתנתה בין מעבר בשתי משבצות סמוכות, כלומר בתזוזה של צעד אחד: כיוון שהיוריסטיקה מתבסס על חישוב מרחק מנהטן, אז בהנתן שזזנו לכיוון המסלול שהפונקציה חישבה בפעם הקודם, אז האורך שלו נהפך לקטן ב-1, ולכן גם ערך החזרה של פונקציית היוריסטיקה הנוכחית תקטן ב-1. אם זזנו לכיוון אחר מהמסלול, האורך יכול או לגדול ב-1 (אם זזנו לכיוון מנוגד) או להשאר זהה (אם יש מסלול סימטרי מהנקודה החדשה). לא ייתכן שלאחר תזוזה של צעד אחד אנחנו נמצא מסלול שיותר טוב ב-2 או יותר מהמסלול הקודם שחושב, כיוון שבמקרה הזה היינו מוצאים אותו בחישוב הקודם, ומכיוון שלא מצאנו אותו, הוא לא קיים.
אם רשימת הנקודות כן משתנית, אז זה יכול לקרות מ-2 סיבות:

1. במקרה עברנו באחת מנקודות האוכל שאינה קשורה לקומבינציה שאליה מחשבים את המסלול. במקרה הזה היוריסטיקה לא תתחשב בכך שאכלנו אותה והערך שלה לא יושפע ויהיה זהה להסבר מעלה.
2. הגענו לאחת מהנקודות שאליה מחשבים את המסלול, ולכן היוריסטיקה חיפשה נקודה חדשה שתתווסף לקומבינציית הנקודות החדשה. כעת היוריסטיקה תחפש קומבינצייה חדשה של amount נקודות שהכי קשה להגיע לכולן. הערך שיוחזר עבור הקומבינצייה החדשה בהכרח יהיה גדול/שווה לקומבינצייה הקודמת כיוון שבקומבינצייה הקודמת השחקן היה במרחק של 1 מאחת הנקודות, ולכן הנקודה החדשה שתתווסף לקומבינצייה כנראה רק תגדיל את המרחק להגעה לכולן, ובוודאות תשאיר אותו גדול/שווה למרחק הקודם (אם המרחק הזה היה קטן יותר אז לא היינו בוחרים את הקומבינצייה הקודמת מלכתחילה).

כדאי לציין שהמעבר מ-amount=3 ל-amount=4 גם כן לא פוגע בעקביות, כיוון שהוא גורם לפונקציה רק לגדול. הרי שלהגיע לשלוש הנקודות שהכי קשה להגיע אליהן זה יותר קל מאשר להגיע לארבעת הנקודות שהכי קשה להגיע אליהן.