Introduction to Artificial Intelligence (236501)

תרגיל 2

מגישים : אייל אמדור, בארי זיטלני

ת.ז : 318849270, 209351626

ImprovedGreedy

שאלה 1

נגדיר את המשחק כמו שנלמד בהרצאות ובתרגול:

- S – מרחב המצבים. מצב כולל את הרובוטים: מיקום, בטרייה, קרדיט ואיזה חבילה מחזיקים (או $None$ אם לא מחזיקים; של מי התור הנוכחי; החבילות: מיקום התחלתי ומיקום היעד; מיקום תחנות הטעינה; ומספר תורות (נסמן ב-N את מס' התורות המקסימלי) שנותר לשחק.
בצורה פורמלית:

$$R_i = \{(pos_i, battery_i, credit_i, package_i) | according\ to\ instuctions\}$$

$$Turn = \{r_0, r_1\}, P_i = \{(pos_i, dest_i) | according\ to\ instructions\}$$

$$C_i = \{(pos_i) | according\ to\ instructions\}, numOfSteps = [N]$$

$$S = R_0 \times R_1 \times Trun \times P_0 \times P_1 \times C_0 \times C_1 \times numOfSteps$$

- A – קבוצת הפעולות שכל רובוט יכול לבצע: להתקדם צפונה, להתקדם דרומה, להתקדם מזרחה, להתקדם מערבה, לאסוף חבילה, להוריד חבילה, להטעין.

$$A = \{north, south, east, west, drop\ off, pick\ up, charge\}$$

- f – פונקציית המעברים: מוגדת $f: S \times A \to S$ כלומר מקבלת מצב ופעולה ומחזירה את המצב המתקבל ע"י הפעולת הפעולה על המצב הנתון.
- c – פונקציית העלות, מחזירה את עלות השימוש בכל פעולה (חוקית). עבור משחק זה, השימוש הוא 1 יחידות-בטרייה לכל תנועה במרחב ו-0 לכל הפעולות הסטטיות.

$$c: S \times A \to \{0,1\}, \quad c\big((s,a)\big) = \begin{cases} 1, & a \in \{north, south, east, west\} \\ 0, & otherwise \end{cases}$$

- s_0 – המצב ההתחלתי של המשחק. נקבע באופן רנדומאלי (בקבלת הפרמטרים כמות בטרייה התחלתית וכמות הקרדיט ההתחלתי).
- R – פונקצית התועלת, מוגדרת $R: S \times A \to \mathbb{R}$ ומחזירה עבור מצב ופעולה את התועלת שתתקבל מהפעלת הפעולה על המצב. מוגדרת עבור כל סוכן לפי היורסטיקה והאסטרטגיה שלו.

שאלה 2

נגדיר את היוריסטיקה הבאה עבור $s \in S$: לטובת הסדר נשתמש בהגדרות הבאות –

החבילה הכי קרובה, או זאת שאצל הרובוט המשחק

$$P(s) = \begin{cases} R.package, & R.package \\ \arg\min\{MD(R.position, p_i) | i \in \{0,1\}, NOTE^*\}, & otherwise \end{cases}$$

תחנת הטעינה הפנויה הקרובה ביותר

$$C(s) = argmin\{MD(R.position, c_i) | c_i \in Chargers. NOTE^*\}$$

היעד של הרובוט המשחק : לאסוף חבילה אם אין ביד, או יעד החבילה אחרת

$$T(s) = \begin{cases} P(s).destination, & R.package \\ P(s).position, & otherwise \end{cases}$$

מרחק הרובוט המשחק מהיעד שלו

$$target\_dist(s) = MD(R.position, T(s))$$

מרחק הרובוט המשחק מתחנת העגינה שלו

$$charger\ dist(s) = MD(R.position, C(s)$$

הרווח המתקבל מהעברת חבילה ליעדה

$$reward(p) = 2 \cdot MD(p.position, p.destination)$$

משקול ערך החבילה, בהתחשב באם לרובוט המשחק יש חבילה ביד או לא

$$Pweight(s) = \begin{cases} 50, & R.package \\ 5, & otherwise \end{cases}$$

עלות המסלול של הרובוט המשחק ליעדו, כולל לתחנת העגינה לאחר מכן

$$Bcost = \begin{cases} target\ dist + MD(C(s), position, T(s)), & R.package \\ target\ dist + MD(P(s).position, P(s).destination) + MD(P(s).destination, C(s).position), & otherwise \end{cases}$$

כעת נעריך את טיב המצב הנתון ולבסוף נסכום לערך יוריסטי :

הפרש הקרדיטים במשקל גבוה ומצב הסוללה הנתון, נסכם עבור כל מצב

$$h_0(s) = (R.credit - R_{enemy}.credit) \cdot 10{,}000 + R.battery \cdot 220$$

אם לרובוט המשחק אין קרדיט ואינו יכול לטעון מקבל משקל בהתאם לטיב החבילה, מרחקה ממנו והקרדיט שלו

$$h_1(s) = \begin{cases} reward(P(s)) \cdot Pweight + robot.credit \cdot 100 - target\ dist(s) \cdot 30, & R.credit \le 0 \\ 0, & otherwise \end{cases}$$

אם המרחק מהמטען הוא בדיוק הסוללה ויש קרדיט כדי להטעין, תן משקל גבוה לסוללה והתייחס למרחק מהמטען

$$h_2(s) = \begin{cases} robot.battery \cdot 110 - chrger\ dist(s) * 30, & R.battery = charger\ dist \wedge R.credit > 0 \\ 0, & otherwise \end{cases}$$

אם יש מספיק סוללה להשלים את המשימה ואז להטעין, תוסיף משקל לפי ערך החבילה, הקרדיט ועלות השימוש בסוללה

$$h_3(s) = \begin{cases} reward\big(P(s)\big) \cdot Pweight + robot.credit \cdot 100 - Bcost \cdot 30, & Bcost \leq R.battery \\ 0, & otherwise \end{cases}$$

אחרת: אם יש קרדיט ואפשר להטעין, תן משקל גבוה לסוללה והתייחס למרחק מהמטען

$$h_4(s) = \begin{cases} R.battery \cdot \frac{110}{3} - charger\ dist \cdot 30, & Bcost > R.battery\ \wedge R.credit > 0 \\ 0, & otherwise \end{cases}$$

לבסוף, עבור מצב s נחזיר את סכום הערכים שהערכנו.

$$h(s) = h_0(s) + h_1(s) + h_2(s) + h_3(s) + h_4(s)$$

<u>שאלה 3</u>

מימוש בקוד.

RB-Minimax

שאלה 1

בהינתן מינמקס מוגבל במשאבים, שימוש ביורסטיקה קלה לחישוב ייתן עץ פעולות עמוק יותר ולכן ייצג בחירה על מרחב אפשרויות רחב יותר – כלומר ניבוי שחוזה בצורה מדויקת יותר את "עתיד" המשחק.
לעומת זאת, היורסטיקה הינה קלה ולכן מתארת פחות נאמנה את המציאות, כלומר מביאה לידי ביטוי פחות פרמטרים או מורכבויות שצעדים מסוימים עלול ליצור.
מנגד, שימוש ביורסטיקה כבדה לחישוב ייתן עץ פעולות עמוק פחות ולכן ייצג בחירה על מרחב אפשרויות מוגבל יותר כלומר ינבא פחות מהשלכות הצעד במשחק ("רואה פחות רחוק"). לעומת זאת, היורסטיקה הכבדה מתארת בצורה מהימנה יותר את המציאות ולכן השערוך היורסטי טובה יותר.

שאלה 2

ייתכן ולדנה אין באג באלגוריתם. כתלות במימושה לאלגוריתם מינימקס, אין התחייבות של האלגוריתם לבחירת הניצחון המהיר ביותר, לכן אם יש לסוכן של דנה אפשרות לנצח בתור הנוכחי והיא לא נבחרת ייתכן וקיימת אפשרות לניצחון בהמשך המשחק והאלגוריתם בוחר בה.

שאלה 3

אלגוריתמי any-time הם אלגוריתמים שיכולים לשפר את ביצועיהם בהינתן משך ריצה ארוך יותר.
במקרה הנוכחי, הגבלת זמן במקום הגבלת עומק תתבטא בהרצת מינימקס עד לעומק L=1 ובסיום כל איטרציה של האלגוריתם נגדיל את L ב-1 ונריץ מחדש את RB-Minimax עם L גדול 1.
אלגוריתם any-time נוסף שנלמד בקורס הינו ID-DFS שמריץ DFS לעומק מוגבל לזמן ריצה מוגבל.

שאלה 4

מימוש בקוד.

שאלה 5

a. במשחק מרובה משתתפים יש יותר שחקנים שרוצים לנצח (למקסם את ערך הניצחון שלהם ללא התייחסות להרעת שחקנים אחרים). כלומר בחירה בצומת עם הערך המקסימלי עבורי תמיד. כתלות במספר השחקנים (NUM_OF_PLAYERS) ולכן נעשה התאמה לפסודו-קוד מהתרגול:

```
Function RB-Minimax-Multiple-Players (State, Turn, depth):
    If G(State) OR depth=0 then return h(State, Turn)
    Children←Succ(State)
    Turn ← Turn(state)
    CurMax← −∞
    Loop for c in Children:
        v← RB-Minimax-Multiple-Players(c, Turn, depth-1)
        CurMax←Max(v, CurMax)
    Return CurMax
```

b. במידה וכל שאר השחקנים רוצים להרע לאחרים, כל סוכן יבחר עבורו את הערך המקסימלי ועבור שאר הסוכנים יבחר את הערך המינימלי. לכן נעשה התאמה לפסודו-קוד מהתרגול:

```
Function RB-Minimax-Multiple-Players (State,Agent, Turn, depth):

    If G(State) OR depth=0 then return h(State, Turn)

    Children←Succ(State)

    Turn ← Turn(state)

    If turn.getAgent == Agent:    #My Turn

        CurMax← −∞

        Loop for c in Children:

            v← RB-Minimax-Multiple-Players(c, Agent, Turn, depth-1)
            CurMax←Max(v, CurMax)
        Return CurMax
    Else:
        CurMin← ∞

        Loop for c in Children:

            v← RB-Minimax-Multiple-Players(c, Agent, Turn, depth-1)
            CurMin←Min(v, CurMin)
        Return CurMin
```

c. כעת כל שחקן רוצה להטיב את השחקן שבא אחריו ולכן נחשב את היוריסטיקה עבור השחקן הבא ונבחר את הפעולה המקסימלית עבורו:

```
Function RB-Minimax-Multiple-Players (State, Turn, depth):

    nextTurn ← Turn(state)
    If G(State) OR depth=0 then return h(State, nextTurn)
    Children←Succ(State)
    CurMax← −∞
    Loop for c in Children:
        v← RB-Minimax-Multiple-Players(c, nextTurn, depth-1)
        CurMax←Max(v, CurMax)
    Return CurMax
```

Alpha-Beta

<u>שאלה 1</u>

מימוש בקוד.

<u>שאלה 2</u>

מבחינת זמן ריצה, גם הסוכן הזה וגם הסוכן הקודם מוגבלים ע״י אותו חסם ולכן הם עתידים להתנהג בצורה דומה. נציין כי במקרה שבו סוף המשחק (מצב שבו לשני השחקנים אין בטרייה) קרוב, סוכן אלפא-בתא יכול להגיע לפיתוח המצבים האלה מהר יותר ולכן גם לסיים את הריצה מהר יותר.

מבחינת בחירת המהלכים, סוכן אלפא-בטא מצליח באותו זמן לפתח עץ עמוק יותר ולכן בחירת המהלכים שלו הינה מושכלת יותר. כלומר, בהחלט ייתכן כי יבחר לבצע מהלכים שונים מאשר הסוכן הקודם.

Expectimax

שאלה 1

כאשר שחקן מקסימום ימצא בצאצאיו צומת בעל ערך 1, הוא יכול לוותר על פיתוח שאר הבנים שלו. זאת מכיוון שמובטח לו שכל ערך אחר בשאר הבנים שלו חסום על ידי $\forall s : -1 \leq h(s) \leq 1$ כלומר לא יימצא בהם ערך הגבוה מ- 1 (ולכן גם חישוב התוחלת יחזיר ערך בין 1 ל-1-) ולכן נעדיף לגזום אותם. באופן דומה, בצומת מינימום, אם שחקן מינימום מוצא צומת בעל ערך של 1-, גם הוא יכול להימנע מפיתוח שאר הבנים שלו. שוב מאותה סיבה, אין אפשרות למצוא ערך נמוך יותר בהמשך הפיתוח ולכן נעדיף לגזום את יתר הבנים.

שאלה 2

מימוש בקוד.

משחק עם פקטור סיעוף גדול

שאלה 1

a) פקטור הסיעוף לא ישתנה. הגדלת לוח המשחק לא משנה את מספר הפעולות האפשרי בכל צעד (צפון, דרום, מזרח, מערב, הורדת חבילה, איסוף חבילה וטעינה). בנוסף, הוספת מחסומים לא משנה גם היא את פקטור הסיעוף שהוא מספר הפעולות המקסימלי (מחסום עלול לשנות את מספר הפעולות עבור צעד ספציפי בדומה לעמידה בצמוד לקיר שלא מאפשרת בתקדמות בכיוון הקיר אך לא משנה את פקטור הסיעוף).

b) פקטור הסיעוף יגדל ב-23. במקרה המקסימלי בו שני הרובוטים מחזיקים את החבילות ועומדים על תחנות העגינה שבמקרה זה גם תחנות הורדת החבילה, נוספו 23 פעולות אפשריות: הנחת בלוק בכל משבצת פנויה (25 משבות סה״כ מתוכן רק 2 לא ריקות). לכן פקטור הסיעוף גדל ב23.

שאלה 2

a) כפי שנלמד בהרצאות, אלגוריצם GREEDY הוא לינארי ולכן גם עם השינוי הנ״ל זמן הריצה שלו לא ישתנה בצורה משמעותית. לעומת זאת, אלגוריתמי ה-minimax לסוגיהם משתמשים בעצים ולכן זמן הריצה שלהם הוא אקספוננציאלי למקדם הסיעוף, כלומר עם השינוי הנ״ל הריצה שלהם תתארך בצורה משמעותית.

b) MCTS – כפי שלמדנו בהרצאה אלגוריתם MCTS ממקסם את היחס בין ה-exploration ל-exploitation ולכן ימנע חיפושים מיותרים בתוך העץ. סה״כ האלגוריתם מאפשר חיפוש יעיל בתוך עץ ההחלטות ולכן ירוץ בזמן סביר.

שאלה פתוחה – MCTS

שאלה 1

א. נחשב:

$$UCB(a_1) = \frac{U(a_1)}{N(a_1)} + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_1)}} = \frac{3.5}{5} + 1 \cdot \sqrt{\frac{\ln(7)}{5}} = 1.3238$$

$$UCB(a_2) = \frac{U(a_2)}{N(a_2)} + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_2)}} = \frac{1.4}{2} + 1 \cdot \sqrt{\frac{\ln(7)}{2}} = 1.6863$$

ב. לפי הגדרת האלגוריתם הוא ייבחר את הזרוע שממקסמת את UCB ולכן ייבחר בזרוע $a2$

שאלה 2

כעת נחשב:

$$UCB(a_1) < UCB(a_2) \Rightarrow \frac{U(a_1)}{N(a_1)} + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_1)}} < \frac{U(a_2)}{N(a_2)} + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_2)}}$$

$$1 + C \cdot \sqrt{\frac{\ln(10)}{8}} < \frac{1}{2} + C \cdot \sqrt{\frac{\ln(10)}{2}} \Rightarrow \frac{1}{2} < \frac{1}{2} \cdot C \cdot \sqrt{\frac{\ln(10)}{2}} \Rightarrow C > \sqrt{\frac{2}{\ln(10)}} = 0.931$$

שאלה 3

התשובה הנכונה היא: *2*. כן, עבור כל ערך $C > 0$ מתקיים כי מתישהו a_2 ייבחר, אולם לא עבור $C = 0$.
זאת מכיוון שכדי ש- a_2 ייבחר נדרוש שערך ה-UCB שלו יהיה גדול יותר משל a_1 ולכן אם נגדיל את מספר הפעמים שנבחר את a_1 מספיק נקבל שערך ה-exploration שלו יקטן מספיק בעוד שהערך של a_2 ייגדל מספיק ולכן נבחר בו לכל $C > 0$. אם $C = 0$ אזי שערך ה-exploration לא משפיע כלל ולכן תמיד נעדיף את ערך ה-exploitation שהוא תמיד יעדיף את a_1 (בעל ערך 1 לעומת 0 של a_2). מתמטית נדרוש:

$$1 + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_1)}} < 0 + C \cdot \sqrt{\frac{\ln(N_{all})}{N(a_2)}} \Rightarrow \frac{1}{C} < \sqrt{\frac{\ln(N_{all})}{N(a_2)}} - \sqrt{\frac{\ln(N_{all})}{N(a_1)}}$$

כאשר מגדילים רק את $N(a_1)$ אזי האגף הימני מתנהג כך: $\frac{1}{C} < \sqrt{\frac{\ln(N_{all})}{1}} - \sqrt{\frac{\ln(N_{all})}{N_{all}-1}}$

לכן, כפי שאפשר לראות לכל $C < 0$ מתקיים שלאחר מספר גדול מספיק של משיכות המשוואה תתקיים ובפרט $UCB(a_1) < UCB(a_2)$ כלומר הזרוע של a_2 תבחר.

שאלה פתוחה – אלגוריתם מונטה קרלו למשחקי אינפורמציה חלקית

שאלה 1

הבעיה שהלאגוריתם מנסה לפתור היא חוסר מודעות למצב המשחק המלא, בדרך כלל מצב המשחק של היריב לא ידוע ולכן כדי לפתור זאת האלגוריתם ינסה לנחש את k מצבים אפשריים של היריב ובהתאם להם יחליט על דרך פעולה. (למה k ? זאת מטעמי הגבלת משאבים שכן פיתוח כל המצבים האפשריים יהיה יקר וארוך מאוד).

שאלה 2

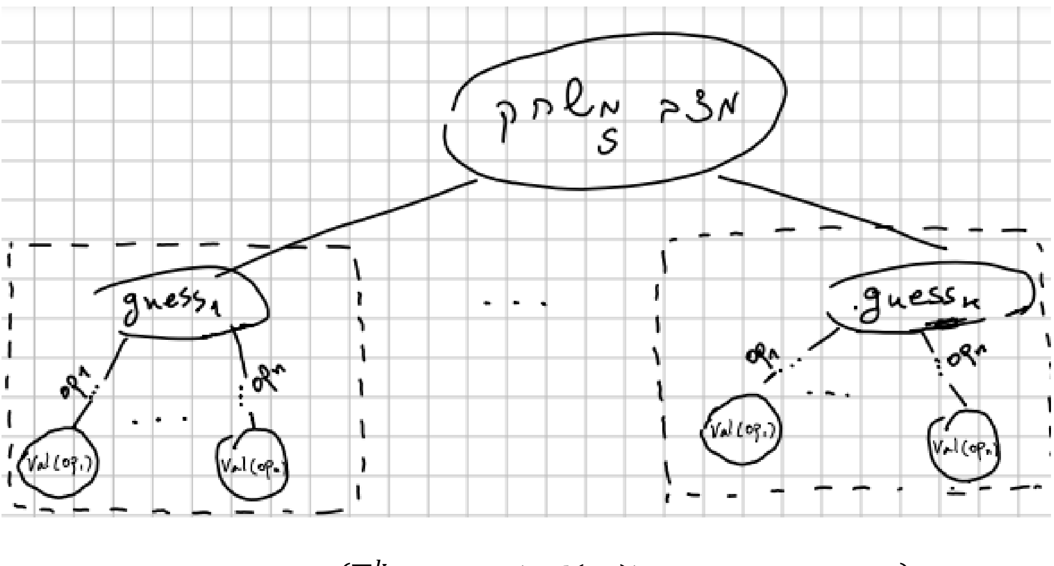
בחירת ערך k גדול מתארת יותר מצבים אפשריים ולכן הסיכוי שלה לנחש נכון את המצב הנוכחי גדול יותר. עם זאת בחירת k גדול עולה יותר משאבי זכרון ותקח יותר זמן להרצת האלגוריתם.

עבור k קטן האלגוריתם מנחש פחות מצבים ולכן הסיכוי שלה לנחש נכון את המצב הנוכחי נמוך יותר, אך היא דורשת פחות משאבי זכרון וזמן ריצתה יהיה מהיר יותר.

שאלה 3

באלגוריתם Expectimax יש מרחב אי וודאות שמבוטא בצמתים עם הסתברות P. נבנה את עץ ההחלטה כך שתחת שורש העץ (המצב) יהיו K צמתים עם ניחוש למצב משחק כאשר ההסתברות שלהם יוניפורמית: $\frac{1}{k}$. תחת כל ניחוש כזה, המשחק הוא דטרמיניסטי ואין אינפורמציה חלקית לכן נוכל לשחק עם אלגוריתם minimax (אנחנו מניחים שהיריב רוצה למקסם על ניצחון). בסיום ריצות כל עצי המינימקס (סיום המשחק או עד עומק מוגבל) נוכל לסכום את הערך שניתן ע״י הפעולה (בכל הניחושים) ולחלק בכמות הניחושים (k). נדגיש כי אם פעולה לא אפשרית בניחוש מסוים הערך שהיא תקבל יהיה 0.

כלומר, נחשב את התוחלת של כל פעולה ונבחר את הפעולה עם התוחלת הגבוהה ביותר – כמו בהרצת Expectimax רגילה. בצורה זו, שבה כל צומת Expectimax מכיל K עצי minimax עבור K ניחושים נקבל למעשה ריצת שקולה לריצת MCTS על משחק אינפורמציה חלקית.



$$op = maxarg\left\{\frac{\sum_{i=1}^{k} guess_i(val(op))}{k} \mid op \in operations\right\}$$

שאלה 4

רעיון: נריץ אלגוריתם MCTS עם ערך k=1 וכל עוד יש לנו זמן נריץ שוב את האלגוריתם עם k גדול ב1. כשיגמר הזמן נחזיר את הפתרון עבור ריצת האלגוריתם ה-k האחרון שסיים את ריצתו.

פסודו קוד:

```
best_op = Null
best_val = -inf
k = 1
while current_time < TIME_LIMITATION:
    val, op = MCTS(partial_state, agent, depth, k, TIME_ LIMITATION)
    if (val > best_val):
        best_val = val
        best_op = op
    k += 1
return best_op
```

- העברנו את ה-TIME_ LIMITATION לאלגוריתם-MCTS כדי שנוכל לקטוע אותו באמצע הריצה במידה והזמן חרג.