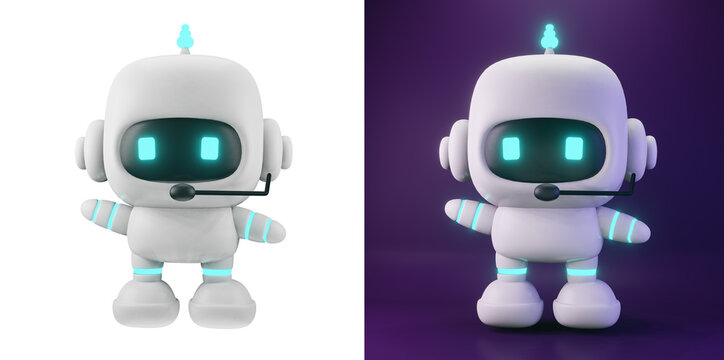
**תרגיל בית 2 מבוא לבינה מלאכותית**

**חיפוש רב סוכנים**

**בינה מחסנים**



**הקדמה ואדמיניסטרציה**

**הנחיות כלליות**

* תאריך הגשת התרגיל: **24.3.2024**
* את המטלה יש להגיש בזוגות בלבד – בקשות להגשה ביחידים באישור המתרגל האחראי בלבד (ספיר טובול).
* יש להגיש את המטלה מוקלדת בלבד – פתרון בכתב יד לא ייבדק.
* התשובות צריכות להיות כתובות בשפה העברית או באנגלית.
* אפשר לשלוח שאלות בנוגע לתרגיל דרך הפיאצה.
* המתרגלת האחראית על התרגיל: **אופק גוטליב**.
* בקשות דחיה מוצדקות יש לשלוח למתרגל האחראי בלבד.
* במהלך התרגיל ייתכן שיעלו עדכונים – הודעה תפורסם בהתאם במקרה זה.
* העדכונים מחייבים וזוהי אחריותכם להתעדכן לגביהם עד מועד הגשת התרגיל. עדכונים יופיעו בטופס בצבע צהוב.
* העתקות תטופלנה בחומרה.
* ציון המטלה כולל חלק יבש וחלק רטוב. בחלק היבש נבדוק שתשובתכם נכונה, מלאה, קריאה ומסודרת. בחלק הרטוב הקוד שלכם ייבדק הן על הגבלת הזמן שתפורט בהמשך ועל אחוזי ההצלחה של האלגוריתמים שלכם לעומת אלו שאנו נממש כבדיקה.
* מומלץ להסתכל בקוד בעצמכם. שאלות בסיסיות על פייתון שלא נוגעות לתרגיל כדאי לבדוק באינטרנט לפני שאתם שואלים בפיאצה. מומלץ לקרוא את הקוד הנתון על מנת להבין את אופן פעולתו – במקרה שישנם דברים לא מובנים . (לשם כך יש הערות רבות ואף הסבר מורחב על הסביבה!)
* מומלץ לא לדחות את התרגיל לרגע האחרון מאחר שהמימוש וכתיבת הדו"ח עלולים לקחת יותר זמן מהצפוי.
* התייחסות בלשון זכר, נקבה או רבים מתייחסים כלפי כלל המינים.
* אין צורך להשתמש בתהליכים ובמאפיינים מורכבים של מערכות הפעלה, התרגיל בית אמור לרוץ על כל מערכת הפעלה ולא למערכות הפעלה ספציפיות. אם ישנה בעייה פרטנית, שלחו מייל למתרגלת האחראית על התרגיל.

**הוראות הגשה**

בתוך קובץ זיפ עם השם : HW2\_AI\_id1\_id2.zip

את הדו"ח היבש בפורמט הבא : id1\_id2.pdf

ואת הקובץ: submission.py שבו אתם ממשים את האלגוריתמים

**הקדמה**

בינה מחסנים רוצים להפוך את המחסן לאוטונומי והם מתלבטים מיהו הרובוט אותו הם רוצים לשכור למשימה. על הרובוטים רובוט R1 ו-R2 להתחרות אחד בשני והמנצח יתקבל לעבודה.

בתרגיל זה תממשו ותחקרו אלגוריתמי משחק סכום אפס אדברסריאלים שלמדתם בהרצאות ובתרגולים.

**תיאור המשחק**

המשחק מתרחש בלוח משבצות בגודל 5X5 כאשר על הלוח: 2 רובוטים, 2 תחנות הטענה,   
2 חבילות ו-2 יעדים (אחד לכל חבילה).   
לכל רובוט יחידות הטענה (battery), וניקוד (credit).

מטרת כל רובוט לצבור יותר ניקוד מהרובוט האחר עד סוף המשחק, המשחק נגמר כאשר לאחד מהרובוטים נגמרת הסוללה או כאשר נגמר מספר הצעדים המקסימלי לכל רובוט (ערך מוגדר מראש, דוגמה בהמשך).

ניקוד נצבר כאשר רובוט מביאה חבילה אל היעד שלה. כאשר רובוט מביאה חבילה ליעד הוא מקבל N יחידות ניקוד כאשר N הוא מרחק מנהטן בין מיקום החבילה המקורי ויעדה של החבילה כפול 2. לאחר שחבילה מגיעה אל היעד שלה היא והיעד שלה נעלמים ובמקומם מופיעים יעד וחבילה חדשים. כלומר, בכל עת על הלוח יש בדיוק שתי חבילות ושני יעדים, אחד לכל חבילה.

רובוטים מתחילים עם טעינה וללא חבילה. כל פעולת תנועה של הרובוט עולה לו יחידת סוללה אחת. רובוט יכול לנוע למעלה, למטה, ימינה, שמאלה. בנוסף, רובוט יכול להטעין את הסוללה בתחנת הטענה כאשר הוא נמצא במשבצת של תחנת הטענה, הוא עושה זאת ע"י המרה של כל יחידות הניקוד שלו ליחידות טעינה. רובוט יכול לאסוף ולהוריד חבילה. רובוט אוסף חבילה כאשר הוא עומד באותה משבצת כמוהה ומבצע פעולת אסיפה. רובוט מוריד חבילה כאשר הוא נמצא ביעד ומבצע פעולת הורדה. (אי אפשר להוריד חבילה שנאספה במשבצת שאינה היעד)

סה"כ לרובוט 7 פעולות אפשריות בכל עת:

move north, move south, move east, move west, pick up, drop off, charge

לא יתאפשר צעד למשבצת לא חוקית (משבצת לא חוקית היא משבצת מחוץ לגבולות הלוח או כזו שנמצא בה הרובוט השני). לא יתאפשר הטענה, אסיפה, הורדה במשבצות לא חוקיות. כל סוכן יכול להטעין בכל אחת מהתחנות הטענה.

**הסבר על הסביבה ו הרצת / דיבוג המשחק**

בשונה מתרגיל בית 1 הפעם תעבדו עם קבצי py ולא מחברות pynb. מוזמנים לעבוד על התרגיל בכל IDE מתאים שנוח לכם לעבודה בפייתון, אנו ממליצים על pycharm של חברת jetBrains אליה יש לכם מנוי מטעם הטכניון.

הסביבה שאיתה תעבדו ממומשת בקובץ WarehouseEnv.py מוזמנים לעיין בה.

בקובץ Agent.py ממומשים סוכנים מהם הסוכנים שאתם תממשו יורשים, מומלץ להסתכל על הסוכנים הממומשים בה בכדי להבין כיצד הם עובדים עם הסביבה.

מומלץ להסתכל על הפונקציות get\_legal\_operator ו - apply\_operator בכדי להבין את אופן ההתממשקות שלכם עם הסביבה.

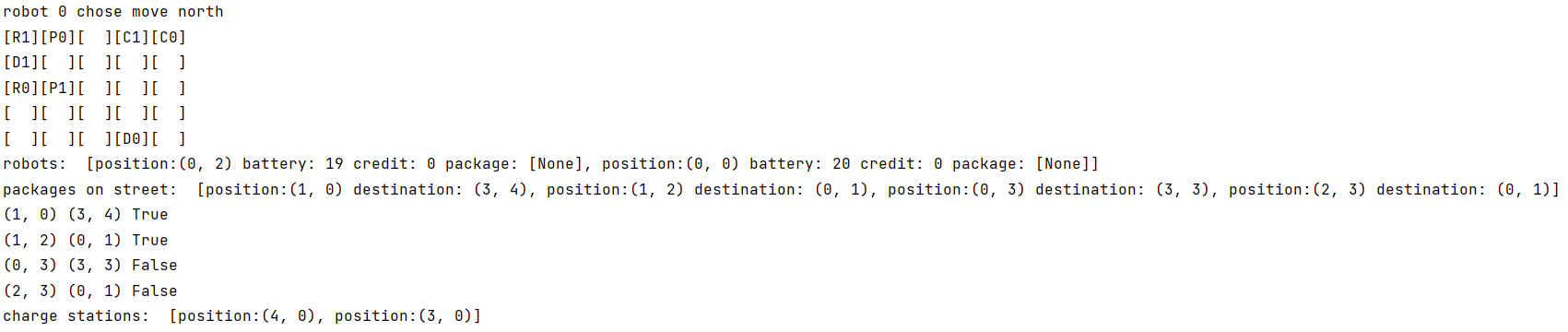
שימו לב שבקובץ submission יש סוכן שנקרא hardcoded, מטרתו לעזור לכם להבין את הסביבה. לפני שאתם שואלים לגבי כיצד הסוכן יתנהג אם יבצע פעולה בדקו בעצמכם בעזרת סוכן זה.

**הרצה**

הרצת משחק נעשית ע"י שורת הפקודה הבאה שמריצים מהטרמינל:

python main.py greedy random -t 0.5 -s 1234 -c 200 --console\_print --screen\_print

* הארגומנט הראשון (במקרה הזה greedy) מציין את האלגוריתם שלפיו ישחק agent0
* הארגומנט השני (במקרה הזה random) מציין את האלגוריתם שלפיו ישחק agent1
* הגבלת זמן לצעד t- מקבלת ערך שמייצג את מספר השניות המקסימילי לצעד
* גרעין לקבלת ערך רנדומלי s- מקבל ערך שעוזר לחולל סביבה באופן רנדומלי, כאשר מועבר אותו ערך seed תחולל אותה סביבה
* מספר הצעדים המקסימלי עבור סוכן c-
* ערך console\_print-- דגל אופציונלי, כאשר מועבר מתבצעת הדפסה לקונסולה של המשחק שנראת כך :

מודפס מספר הסוכן יכול להיות 0 או 1 והאופרטור בו בחר. לאחר מכן מודפס הלוח לאחר שהסוכן הפעיל את האופרטור. הלוח כולל 25 משבצות כאשר בכל אחת יכולים להיות רובוט, חבילה המחכה לרובוט או תחנת הטענה. 

האותיות אשר מופיעות בלוח מסמלות:

R – Robot

C – Charge station

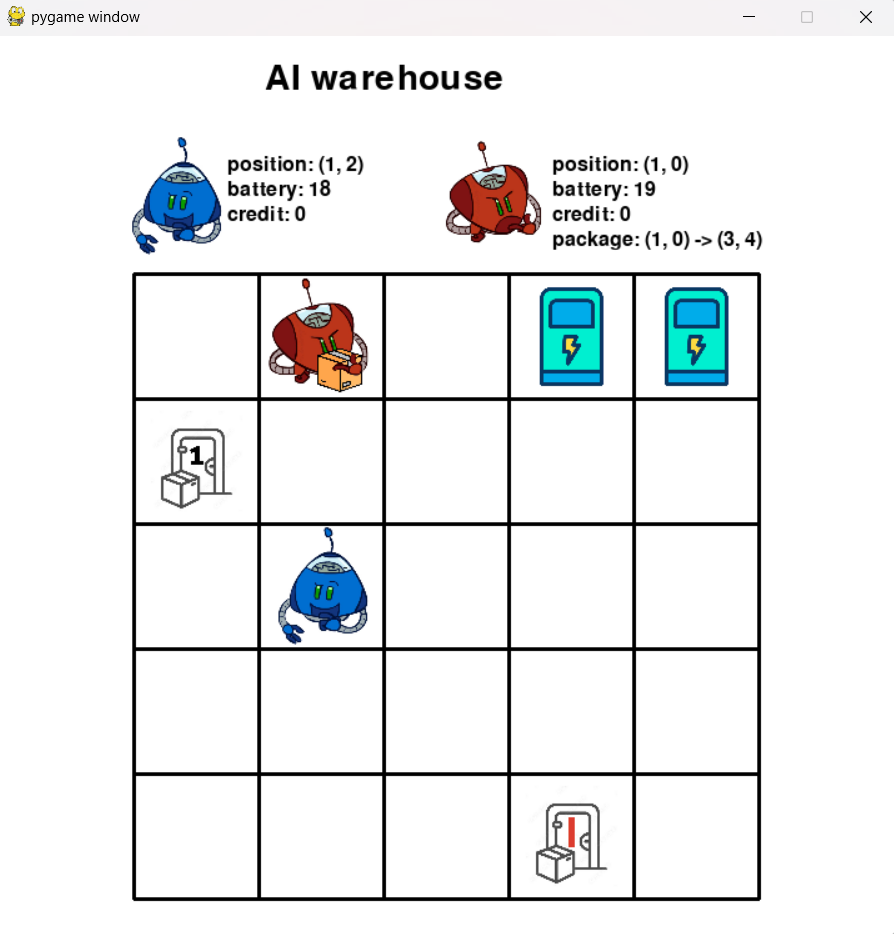
P – Package on street

D – Destination of Package on street

X – Destination of Package taken by Robot

המספר המופיע ליד כל אחת מהאותיות הוא המזהה של האובייקט אליו הוא משתייך – לדוגמא: R1 עבור הרובוט הראשון. עבור חבילה שנאספה ע"י רובוט, המספר ליד X הוא המזהה של הרובוט שאסף אותה. עבור חבילה במרחב, המספר ליד D הוא המזהה של החבילה במרחב. לעיתים מספר אובייקטים מופיעים באותו המיקום בתמונה ואז מודפס רק אחד מהם. במקרה זה, אפשר להשתמש ברשימות המפורשות שמופיעות אחרי הלוח. בנוסף שימו לב! מופיע לכם ברשימות את המקור והיעד של שתי החבילות הבאות שיופיעו על הלוח, מוזמנים ומומלץ להשתמש במידע זה בהמשך התרגיל.

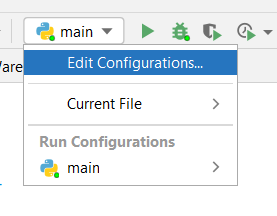
* ערך screen\_print-- דגל אופציונלי, כאשר מועבר יודפס לכם הנפשה של המשחק בחלון pygame שנראה כך:

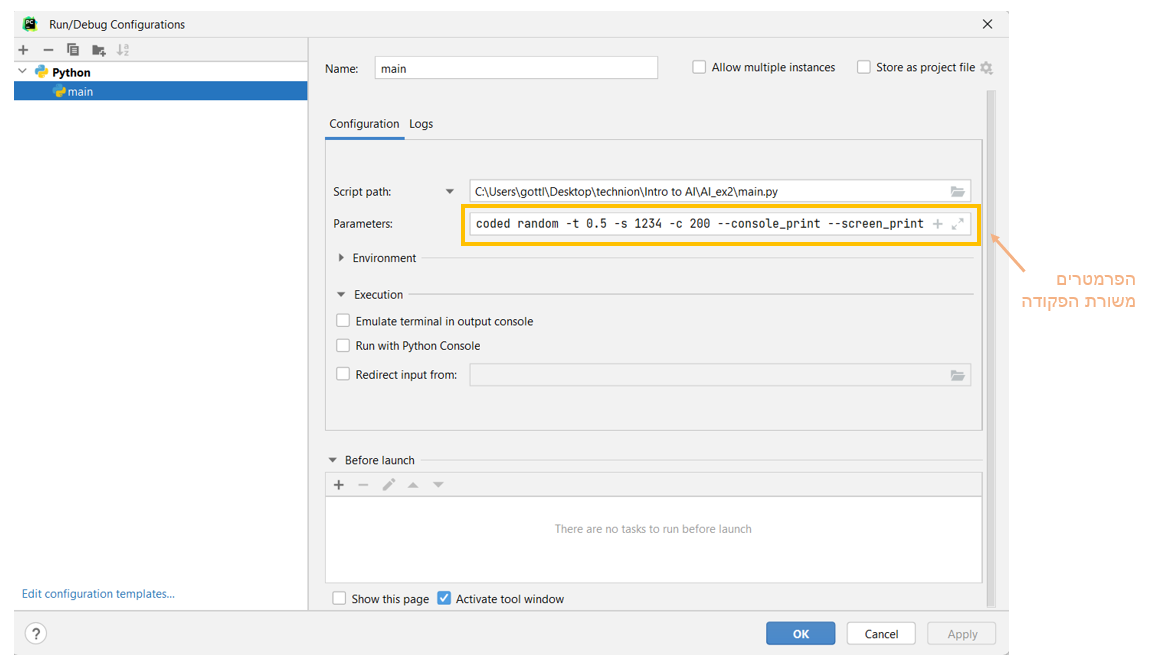


בנוסף בכדי לבדוק את הסוכנים שלכם על מספר רב של משחקים, אתם יכולים להוסיף את הדגל tournament-- וכך תקבלו הרצה רצופה של n משחקים. מוזמנים לשחק עם המשתנה של מספר המשחקים ולראות האם מקבלים התנהגויות רצויות עבור האלגוריתמים.

**דיבוג**

בכדי לדבג את המשחק עליכם לעקוב אחר השלבים הבאים:

2.



**מתחילים לכתוב !**

**חלק א - ImprovedGreedy**

1. (יבש: 4 נק') כפי שלמדתם, אנו מגדירים בעייה במרחב בתור רבעייה (S, O, I, G). הגדירו פורמלית (הסבירו כיצד נראים ערכי הרבעייה) את המשחק המתואר לכם ע"פ הנתונים שאתם מקבלים מהסביבה.

Sמרחב המצבים יכלול את המידע הבא:

O קבוצת האופרטורים:

move north, move south, move east, move west, pick up, drop off, charge

I קבוצת המצבים ההתחלתיים:

הוא כל קבוצת המצבים שמקיימים את התנאים

G קבוצת המצבים הסופיים:

המשחק נגמר כאשר לשני הרובוטים נגמרת הסוללה או כאשר נגמר מספר הצעדים המקסימלי לכל רובוט, לכן קבוצת המצבים הסופיים הוא כל קבוצת המצבים המקיים

1. (יבש: 4 נק') הגדירו היוריסטיקה משלכם להערכת מצבי המשחק. עליכם לתעד אותה בנוסחה מפורשת ועלייה לכלול לפחות שלושה מאפיינים של הסביבה. בחרו שמות ברורים בנוסחה שלכם.

def smart\_heuristic(env: WarehouseEnv, robot\_id: int):

    robot = env.get\_robot(robot\_id)

    other\_robot = env.get\_robot(1 - robot\_id)

    # Weights for different components of the heuristic

    weight\_distance\_to\_package = -1  # We want to minimize distance

    weight\_distance\_to\_destination = -1  # Minimize distance to destination

    weight\_battery\_level = 7  # Higher weight because battery is critical

    weight\_score\_difference = 10  # Prioritize maintaining/increasing score lead

    nearest\_package\_distance = min([manhattan\_distance(robot.position, package.position)

  for package in env.packages if (package.on\_board and (other\_robot.package is not None or manhattan\_distance(other\_robot.position, package.position) > 1))], default=0)

    if robot.package:

        distance\_to\_destination = manhattan\_distance(robot.position, robot.package.destination)

        nearest\_package\_distance=0

    else:

        distance\_to\_destination = 0  # No package, no distance to calculate

    nearest\_charging\_station\_distance = min([manhattan\_distance(robot.position, charging\_station.position)

                     for charging\_station in env.charge\_stations])

    if (robot.battery <nearest\_package\_distance or robot.battery < distance\_to\_destination ) and nearest\_charging\_station\_distance >= robot.battery and robot.credit < other\_robot.credit :

        nearest\_package\_distance = 0

        distance\_to\_destination = 0

    battery\_level = robot.battery

    score\_difference = robot.credit - other\_robot.credit

    heuristic\_value = (weight\_distance\_to\_package \* nearest\_package\_distance +

                  weight\_distance\_to\_destination\*distance\_to\_destination

                       +weight\_battery\_level \* battery\_level +

                       weight\_score\_difference \* score\_difference )

    return heuristic\_value

נסביר את היוריסטיקה.

בתחילה נתנו משקולת לכמה מרכיבים שונים

משקולות שליליות למרחק עבור החבילה הקרובה ביותר והיעד שלה שכן אנחנו רוצים למנמם מרחק שכזה.

משקולת עבור טעינה שכן אנחנו רוצים לחיות כמה שיותר

ומשקלות כבדה ביותר עבור הפרש הקרדיט – הדבר שבסופו של יום קובע מי המנצח של המשחק.

לאחר מכן חישבנו את המרחק הקטן ביותר לחבילות חוקיות על הלוח , ואם לרובוט יש חבילה כבר אז את המרחק ליעד שלה.

את המרחק לתחנת הטעינה הקרובה ביותר , שבא לידי ביטוי רק במקרים שאנו לא בדרך לאיסוף חבילות או נקודות עם סוללה מספקת.

וכן את הפרש הנקודות ממנו ומהיריב.

את הדבר כולו חישבנו בתור ציון משוקלל עם המשקולות בהתחלה וזהו הערך היוריסטי שניתן למצב כלשהו במרחב המצבים.

1. (רטוב: 10 נק') ממשו בקובץ submission.py את הפונקציה **smart\_heuristic**

שבה משתמש הסוכן AgentGreedyImproved

1. (יבש: 2 נק') מהו החיסרון העיקרי של האלגוריתם? (לעומת minimax)

החיסרון של האלגוריתם שהוא לא מתחשב בערך צעדי היריב שככל הנראה ינסה להביא למינימום את התועלת של הסוכן הראשון, ובנוסף לכך אלגוריתם זה מנסה למקסם רק את הצעד הנוכחי ולא מסתכל על השפעת הצעד לטווח רחוק יותר, לעומת מינימקס אשר מנסה למקסם את ערך הצעד התועלת העתידית בצעדי המשחק הבאים כתלות בעומק החיפוש.

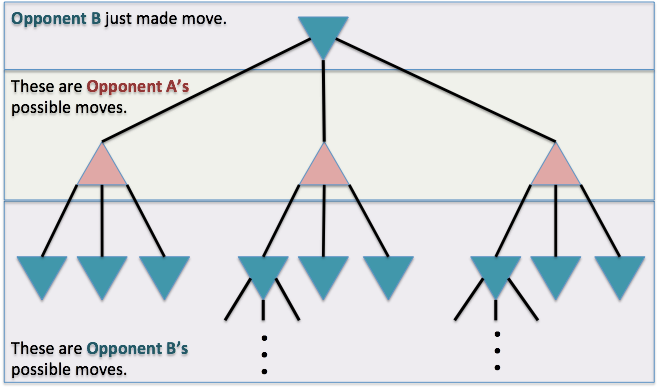
**חלק ב - RB-Minimax**

1. (יבש 3 נק') מה היתרונות והחסרונות של שימוש בהיוריסטיקה קלה לחישוב לעומת היוריסטיקה קשה לחישוב בהינתן שהיוריסטיקה הקשה לחישוב יותר מיודעת מהקלה לחישוב ? בהינתן שאנו בmin-max מוגבל משאבים.

היוריסטיקה קלה לחישוב תוכל לחשב לעומק עמוק יותר בעץ הצעדים וכך "לצפות יותר צעדים קדימה", אך היוריסטיקה קלה לחישוב עלולה להיות פשוטה מידי ולא לייצג את ערך התועלת במידה גבוהה.

1. (יבש: 4 נק') חברתכם לקורס דנה מימשה סוכן minimax, היא שמה לב כי לעיתים הסוכן יכול לנצח בצעד אחד אך הוא בוחר בצעד אחר. האם יש לה באג באלגוריתם? אם אין באג הסבירו מה באלגוריתם גורם להתנהגות שכזו. אם יש באג מה הוא יכול להיות?

לדנה אין באג באלגוריתם. כפי שלמדנו minimax הוא אלגוריתם שמבצע חיפוש לעומק . לכן סוכן minimax מוגבל במשאבים שלא מבצע חיפוש על פי השיפורים שלמדנו (לדוגמא שיפור anytime) יכול להעמיק את החיפוש ממצב לא מנצח בעוד שקיים מצב מנצח בעומק 1 אך משאב הזמן נגמר לסוכן. ננסה לתת תחושה ויזואלית להסבר



בעוד שהפתרון בכלל כאן

יכול להיות

שהסוכן מעמיק כאן

1. (רטוב: 10 נק') עליכם לממש את המחלקה AgentMinimax בקובץ submission.py. שימו לב! הסוכן מוגבל משאבים, כאשר המשתנה time\_limit מגביל את מספר השניות שהסוכן יכול לרוץ לפני שיחזיר תשובה. (הגבלת הזמן עלייה אתם נבדקים הינה שנייה כלומר 1 t-).
2. (יבש: 3 נק') נניח שבסביבה היו K שחקנים במקום 2. אילו שינויים יהיה צריך לעשות במימוש סוכן Minimax? **כתבו פסאודו קוד בדומה לזה שראינו בתרגול.**
   1. בהינתן שכל סוכן רוצה לנצח ולא אכפת לו רק ממכם.

השינוי שיהיה צריך לשנות הוא שבמקום לבצע מינימום בתור של היריב על התועלת שלנו נרצה לבצע מקסימום בתור של היריב על התועלת שלו.

הפסאודו קוד יראה כך (השינויים מסומנים באפור):

***Function*** *Minimax(State,Agent):*

***If*** *G(State) then* ***return*** *U(State,Agent)*

*Turn Turn(State)*

*Children Succ(State)*

***if*** *Turn = Agent then:*

*CurMax*

***Loop*** *for c in Children:*

*V Minimax(c,agent)*

*CurMax max(v,CurMax)*

***Return*** *(CurMax)*

***else****:* (Turm ≠ Agent)

*CurMax*

***Loop*** *for c in Children:*

*V Minimax(c,Turn)*

*CurMax max(v,CurMax)*

***Return*** *(CurMax)*

* 1. בהנחה והדבר היחיד שכל סוכן רוצה הוא שלא תנצחו.

במקרה הזה כל שאר הסוכנים יתפקדו כסוכני מינימום שרוצים שנקבל את המצב עם הערך היוריסטי הנמוך ביותר. אנחנו עדיין נתפקד כסוכן מקסימום שרוצה לבחור במצב עם הערך היוריסטי הגבוה ביותר . נשים לב שבתורם של השחקנים האחרים הם עדיין לא ינסו למקסם את המצב שלהם עצמם, אלא רק להביא למינימום את המצב שלנו.  
 בפסאודו קוד מהתרגול הקוד ישאר אותו הדבר, כל כניסה לפונקציה עם תור של סוכן שהוא לא אנחנו , מנסה למנמם ולא למקסם.

***Function*** *Minimax(State,Agent):*

***If*** *G(State) then* ***return*** *U(State,Agent)*

*Turn Turn(State)*

*Children Succ(State)*

***if*** *Turn = Agent then:*

*CurMax*

***Loop*** *for c in Children:*

*V Minimax(c,agent)*

*CurMax max(v,CurMax)*

***Return*** *(CurMax)*

***else****:* (Turm ≠ Agent)

*CurMax*

***Loop*** *for c in Children:*

*V Minimax(c,Agnet)*

*CurMax min(v,CurMax)*

***Return*** *(CurMax)*

* 1. בהנחה שכל סוכן רוצה שהסוכן שאחריו בתור ינצח.

במקרה זה כל סוכן ירצה למקסם את התועלת של הסוכן שבתור אחריו.

הפסאודו קוד יראה כך (השינויים מסומנים באפור):

***Function*** *Minimax(State,Agent):*

***If*** *G(State) then* ***return*** *U(State,Agent)*

*Children Succ(State)*

*CurMax*

***Loop*** *for c in Children:*

*V Minimax(c, Turn(c))*

*CurMax max(v,CurMax)*

***Return*** *(CurMax)*

**חלק ג - Alpha-Beta**

1. (רטוב: 10 נק') ממשו שחקן אלפא - בטא מוגבל משאבים במחלקה AgentAlphaBeta בקובץ submission.py, כך שיתבצע גיזום כפי שנלמד בהרצאות ובתרגולים.
2. (יבש: 3 נק') האם הסוכן שמימשתם בחלק זה יתנהג שונה מהסוכן שמימשתם בחלק ב מבחינת זמן ריצה ובחירת מהלכים ? הסבירו.

מבחינת זמן ריצה, סוכן זה הינו יותר יעיל מאחר ומבצע גיזום של ענפים לא רלוונטיים ואינו מפתח בנים מיותרים, לכן בזמן קצוב ובמימוש anytime, כפי שעשינו, סוכן זה יצליח להגיע לעומק יותר גדול (עד פי 2) מאשר סוכן minimax, ולכן יוכל לבחור מהלכים אחרים אשר יהיו יותר טובים בראייה למהלכים יותר מתקדמים במשחק, אך אילו לא הייתה הגבלת זמן הסוכנים anytime וminimax יבחרו כמעט באותם מהלכים, למעט מהלכים בעלי ערך יוריסטי זהה.

**חלק ד - Expectimax**

1. (יבש: 3 נק') בהנחה ואתם משתמשים באלגרותים Expectimax נגד סוכן שמשחק באופן רנדומלי לחלוטין באיזה הסתברות תשתמשו? ומדעו?

במקרה זה נרצה להשתמש בהסתברות יוניפורמית על כל המהלכים האפשריים מאחר שבמצב זה ההסתברות של הסוכן לבצע מהלך מסוים מבין המהלכים האפשריים הינה שווה.

1. (יבש: 4 נק') עבור משחקים הסתברותיים כמו שש בש, בהם יש מגבלת משאבים, משתמשים באלגוריתם .RB-Expectimax הניחו כי ידוע שהפונקציה היוריסטית ℎ באלגוריתם Expectimax-RB מקיימת

איך ניתן לבצע גיזום לאלגוריתם זה? תארו בצורה מפורטת את תנאי הגזימה, והסבירו את הרעיון מאחוריו.

נוכל לבצע גיזום לאלגוריתם זה מאחר וקיים חסם עליון על ערך היוריסטיקה והוא : 1.

במידה ובמהלך הריצה שחקן max מוצא באחד מהצאצאים שלו את הערך המקסימלי ניתן לגזום את שאר הבנים, שכן הם לא יספקו תוצאה טובה מזו.

1. (רטוב: 10 נק') הסוכן של minimax ו-alpha-beta מניח שהסוכן היריב יבחר באופרטור שיוביל לתוצאה האופטימלית בתורו, אולם זה לא תמיד מתרחש.

לדוגמה, כאשר אנו מתחרים עם סוכן חמדן, סביר להניח שהוא לא יבחר בפעולה האופטימלית בכל צעד. אפשר להתחשב באפשרות שהיריב יבחר בפעולה שאינה אופטימלית בתורו באמצעות סוכן Expectimax.

גילינו מידע סודי על הרובוט המתחרה, הוא בוחר בין כל הפעולות בצורה יוניפורמית (בצורה אחידה) אבל לתזוזה ימינה ולאסיפת חבילה (כאשר פעולות אלו אפשריות) יש הסתברות גדולה פי 2 מלשאר הפעולות.

ממשו אלגוריתם Expectimax המשתמש במידע הסודי שקיבלתם.

**חלק ה - משחק עם פקטור סיעוף גדול**

1. (יבש: 6 נק') להלן שינויים אפשריים ששוקלים בינה מחסנים לעשות במשחק בכדי לבחון יכולות נוספות של הרובוטים. עבור כל שינוי ציינו מה ההשפעה שלו על מקדם הסיעוף וחשבו את מקדם הסיעוף החדש המתקבל.
   1. הגדלת לוח המשחק להיות 8X8 והוספת מחסומים בסביבה. (מחסומים משמע משבצות שהסוכן לא יכול לעבור בהן)

מקדם הסיעוף מושפע ממספר הפעולות האפשריות בכל מצב, לכן כאשר נגדיל את לוח המשחק מקדם הסיעוף לא יושפע מאחר ובכל מצב עדיין יהיו מקסימום 7 פעולות אפשריות בהנחה כי ניתן לבצע את כל הפעולות, או מקסימום 5 בהנחה כי בהכרח ניתן לבצע מקסימום אחת מהפעולות {pick up, drop off, charge} . כאשר יתווספו מחסומים בסביבה, יהיו יותר מצבים שבהם מספר הפעולות האפשריות יהיה קטן יותר, ולכן מקדם הסיעוף הממוצע יקטן, אך עדיין מקדם הסיעוף המקסימלי יישאר זהה.

* 1. הוספת היכולת של רובוט בכל תור לבחור משבצת על הלוח ולהניח עלייה בלוק, משמע בכל תור יכול הרובוט לנוע למעלה, למטה,ימינה, שמאלה, לאסוף חבילה, להוריד חבילה, להטעין, ולהניח בלוק על הלוח, בלוק יכול להיות מונח על כל משבצת ריקה.

במצב זה מקדם הסיעוף יגדל בכמות המשבצות הריקות בלוח מאחר ונוספת פעולה אפשרית נוספת של להניח בלוק על משבצת ריקה בכל מצב.

נשים לב כי ככל והמשחק יתקדם ויהיו יותר בלוקים על הלוח כך מקדם הסיעוף הממוצע יקטן מאחר ומספר המשבצות הריקות יקטן ולכן יהיו פחות אופציות בהן ניתן להניח בלוק, וכן יהיו יותר מצבים, הסמוכים למשבצת בהן יש בלוק, שבהם מספר הפעולות האפשריות יהיה קטן יותר, אך עדיין מקדם הסיעוף המקסימלי יישאר זהה.

נחשב את מקדם הסיעוף המקסימלי במצב זה:

מקדם הסיעוף המקסימלי יהיה כעת מהצורה:

לכן מקדם הסיעוף המקסימלי במצב זה יתקבל כאשר מספר המשבצות הריקות יהיה מקסימלי. על הלוח ישנם תמיד 4 משבצות סטאטיות שתמיד יתפסו משבצת – 2 תחנות טעינה ו2 יעדי חבילה. לכן, מספר המשבצות הריקות המקסימלי יהיה כאשר הרובוטים והחבילות יהיו מונחים במשבצות אלו ולא יתפסו משבצות נוספות, זאת אומר כאשר כל רובוט יאחז בחבילה ויעמוד על אחד מהמשבצות הסטאטיות הנ"ל. לכן מקדם הסיעוף המקסימלי יהיה:

לא בטוח בחישוב, למה 7 ולא 6 ? אנחנו לא יכולים באותו state לקחת בחשבון גם pickup וגם drop-off

1. (יבש: 6 נק') בהנחה ומימשו את השינוי השני עבור הסביבה (סעיף 1b)
   1. האם יש אלגוריתם מהסעיפים הקודמים שנוכל להשתמש בו שזמן הריצה שלו סביר? (סביר משמע לא גדול מהותית מהזמן שלוקח לו להחזיר צעד עבור המשחק בלי השינוי).

נוכל להשתמש באלגוריתם greedy, אשר בכל צעד מסתכל רק על הבנים שלו בעומק הנוכחי ולכן סיבוכיות הזמן שלו הינה ולכן עליה של מקדם הסיעוף מ7 ל28 תהיה יחסית זניחה.

* 1. הציעו אלגוריתם שונה מאלו שממשתם בסעיפים הקודמים שנלמד בקורס שירוץ בזמן סביר. הסבירו מדוע בחרתם בו ולמה הוא טוב להתמודדות עם האתגר שנוצר משינוי הסביבה.

נוכל להשתמש באלגוריתם MCTS מאחר ואלגוריתם זה מתאים מאוד למשחקים בעלי מקדם סיעוף גדול, מאחר ואלגוריתם זה למעשה מבצע סימולציות על הבנים ועובר על הבנים בצורה סלקטיבית, כך שבנים יותר "מעניינים", אשר יותר רלוונטיים לסוכן, יסומלצו יותר פעמים לעומת בנים פחות "מעניינים" אשר יגזמו ולא נעבור עליהם כלל. הזמן שלוקח לאלגוריתם MCTS לבצע rollout, זאת אומרת סימולציה מלאה של הבן, הינו לינארי בעומק בלבד, ולכן ניתן להגיע לעומקים הרבה יותר גבוהים מאשר באלגוריתם MiniMax או AlphaBeta וכך לשפר את אבליואציית העלים, בזמן הנתון. כך למעשה כל עלה יקבל, ככל הנראה, ערך יותר מדויק מהיוריסטיקה שהצענו.

**חלק ו - יבש - שאלה פתוחה - MCTS**

שחקן אדום ושחקן כחול שיחקו משחק. להלן עץ המשחק שמתאר את העץ שנוצר בשלב ביניים בהרצת MCTS עם פיתוח צמתים לפי UCB1 על משחק סכום אפס בין שניהם, נתון .

a

b

c

43/100

1/9

6/21

3/4

5/5

0/3

10/20

6/7

דגש על האלגוריתם: הערך בצומת הכחול מייצג את כמות הניצחונות של השחקן האדום מתוך כמות המשחקים שבוצעו עם הצעד הזה (ולהיפך). למשל צומת b מייצג את כמות הניצחונות של השחקן האדום אם בחר בפעולה שגרמה לו להגיע למצב b מתוך סך כל המשחקים ששוחקו עם הצעד.

1. (5 נק') חלק מהערכים בצמתים נמחקו, השלימו את החלקים החסרים, אין צורך לנמק.
2. (5 נק') הצומת הבא שייבחר בשלב ה - selection יהיה (הוסיפו חישובים לנמק את בחירתכם):
   1. צאצא של a
   2. צאצא של b
   3. צאצא של c

נחשב עבור שלושת הבנים מי הצומת הבא שיבחר

לכן הבן שיבחר הוא בהכרח צאצא של b

1. (5 נק')בהנחה שכל סימולציה מכאן והלאה מסתיימת בניצחון של השחקן הכחול מה מספר הניצחונות המינימלי שנדרש כדי שצאצא אחר של השורש ייבחר בשלב ה-selection (הוסיפו חישובים לנמק את תשובתכם)?

נראה כי במידה ובבן האמצעי ישנו עוד משחק עם ניצחון של השחקן הכחול תהמספר המינימלי של ניצחונות שנדרש כדי לבחור בצומת השמאלי הוא 1

1. (3 נק') כעת רוצים לבצע שינוי כך שנעדיף exploration יותר מ - exploitation. הגישה לנוסחה שמחשבת את ה - UCB1 חסומה לכם, אך הנוסחה משתמשת ב - אשר אליו יש לכם גישה ואתם יכולים לשנותו. כיצד תשנו אותו בכדי שהנוסחה החדשה שתיווצר תעדיף יותר exploration מ - exploitation לעומת הנוסחה הקודמת.

ניזכר בנוסחה של UCB1

כאשר הגורם הראשון הוא רכיב הexploitation והשני הוא רכיב הexploration . נשים לב שעבור כל ערך גדול מ1 של N(s) מתקיים ש

לכן המסקנה המתבקשת עבור לתעדף את הexploration על פני הexplotation הוא להגדיל את ערך ה . נוכל לעשות זאת על ידי כל מניפולציה שמגדילה מספר הניסויים כמו למשל להעלות אותו בריבוע כך נקבל כי והנוסחה החדשה תעדיף exploration .