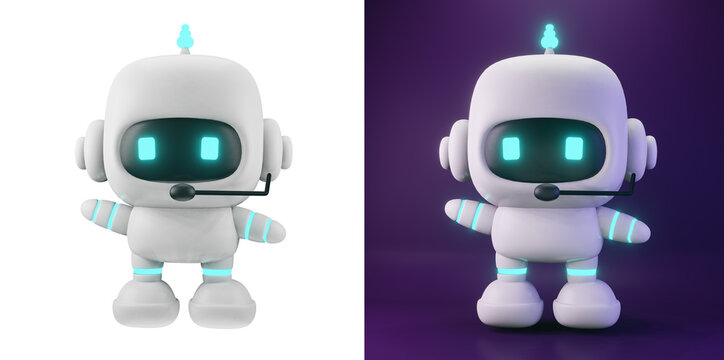
**תרגיל בית 2 מבוא לבינה מלאכותית**

**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.

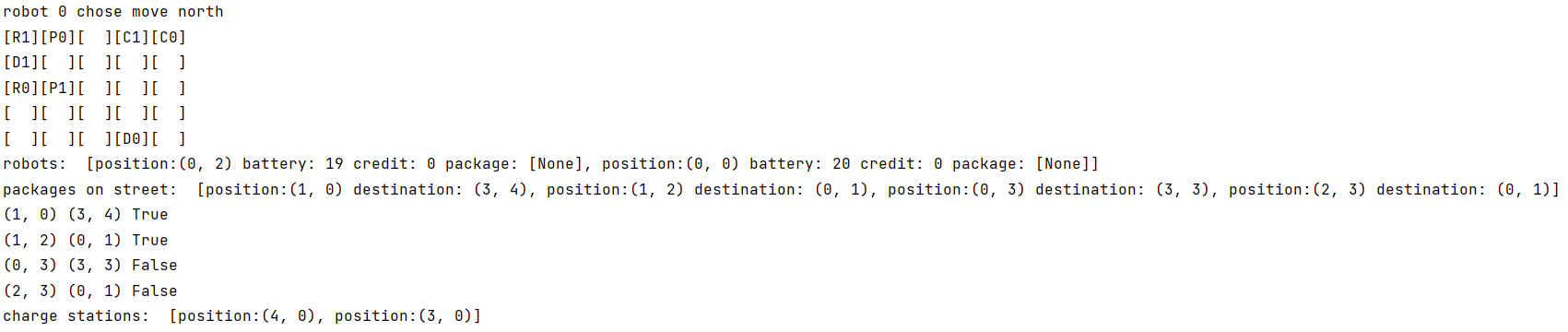
שימו לב שבקובץ submission יש סוכן שנקרא hardcoded, מטרתו לעזור לכם להבין את הסביבה. לפני שאתם שואלים לגבי כיצד הסוכן יתנהג אם יבצע פעולה בדקו בעצמכם בעזרת סוכן זה.

**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

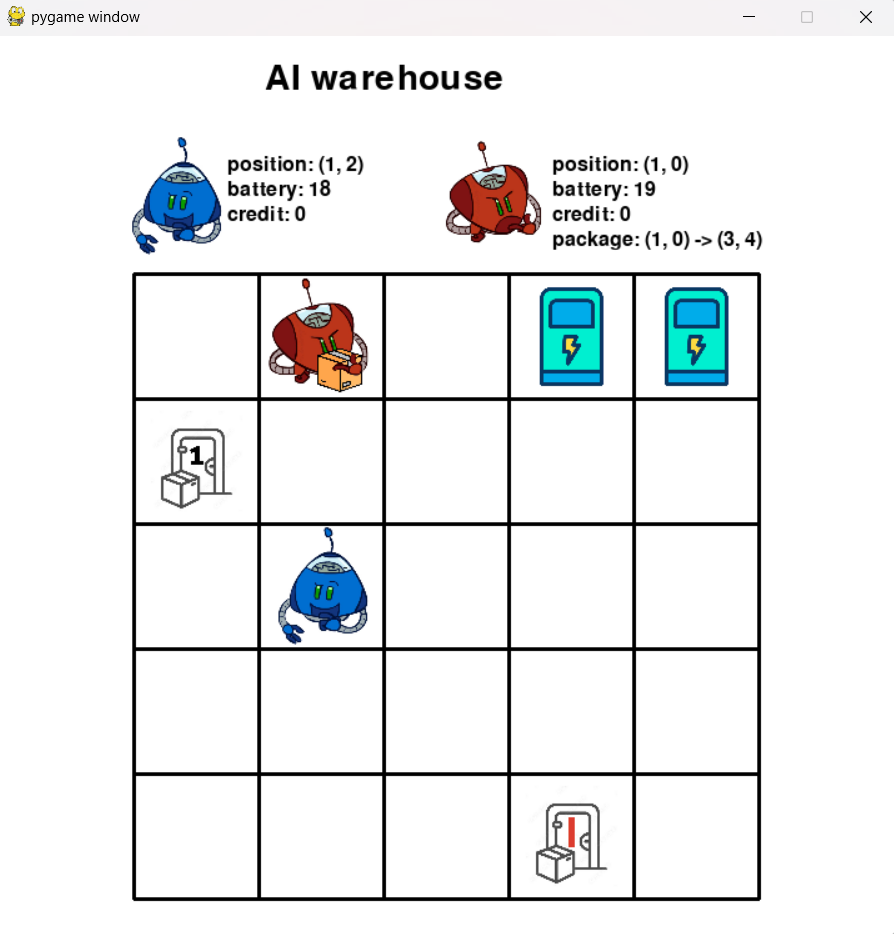
P – Package on street

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.

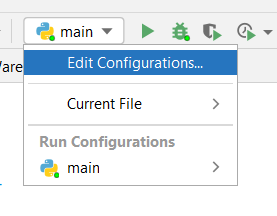
* ערך screen\_print-- דגל אופציונלי, כאשר מועבר יודפס לכם הנפשה של המשחק בחלון pygame שנראה כך:

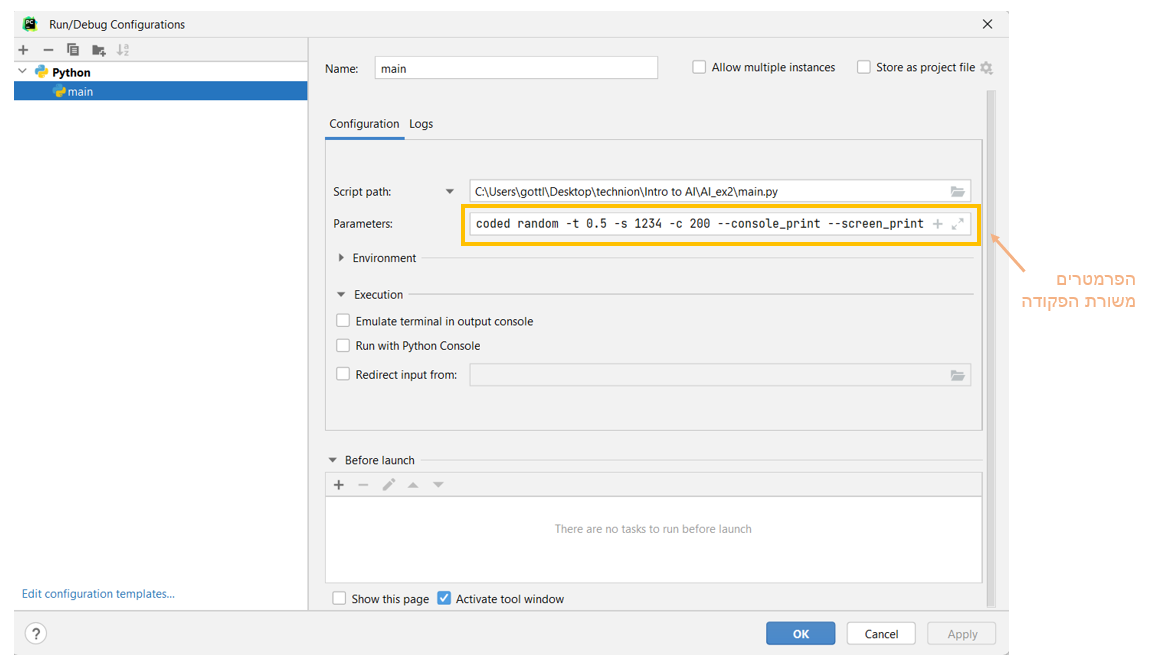


בנוסף בכדי לבדוק את הסוכנים שלכם על מספר רב של משחקים, אתם יכולים להוסיף את הדגל tournament-- וכך תקבלו הרצה רצופה של n משחקים. מוזמנים לשחק עם המשתנה של מספר המשחקים ולראות האם מקבלים התנהגויות רצויות עבור האלגוריתמים.

**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.

State space - S = it consists of all possible combinations of the robots with their position, credit scores, battery and remaining moves, packages, package destinations and charging stations on a board. We have a board of 5x5:

-R0, R1 with values [0,24]

-R0-Battery, R1-Battery with values N U {0}

-R0-Credits, R1-Credits with values N U {0}

-R0-Package, R1-Package if each robot has a package on him with values {P0, P1, None}

-P0, P1 with values [0,24]

-D0, D1 destination with respect to packages with values [0,24]

-Remaining Moves with values N U {0}

Operators - O = {move North, move South, move East, move West, Pick up, Drop off, Charge} – available moves/actions for every agent to do the tasks at hand.

Initial state - I = the initial state which represents a configuration of the game, by default the robots start without credits and with a battery in a random location on the board as well as the packages, destinations and charging stations.

Goal state - G = the goal states in this game is when the remaining moves are zero or one of the robots runs out of battery with the goal of reaching higher credit score than the opponent.

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.

Manhattan Distance ~ MD

We define the following values:

tValue = max{ battery – MD(robot,package1) – MD(package1,destination1)

, battery – MD(robot,package2) – MD(package1,destination2)}

Meaning that tvalue indicates the maximum battery level remained after delivering either package.

If tvalue is negative, this means that there is no sufficient battery for any package delivering then we assign fvalue to be the minimal distance to a charging station.

If tvalue is non-negative, then we assign fvalue = tvalue.

Calculating the final heuristic depends on whether the robot has a package in hand or not.

The coefficients (weights) are adjusted and set after trial and error.

If the robot picked a package, the returned heuristic is:

10\*battery + 40 – 0.5\*MD( robot , package destination )

If the robot doesn’t have a package, the returned heuristic is:

10\*battery + fvalue + 10\*credits

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)

Using our heuristic, ImprovedGreedy doesn’t take into consideration the other agents available moves, current credit, battery, location…

In addition, the Greedy algorithm, compared to Minimax, does not consider the long-term consequences of its actions and focuses on immediate gain, which might in some cases lead to a disadvantageous position late in the game or suboptimal results.

Minimax covers that by exploring the decision tree and finding the possible best results outcome.

**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.

**Advantages:**

**-Fast to calculate:** Easy-to-calculate heuristics are computed quickly, which makes for faster decision making, helps mainly in real time scenarios, as in situations where time is a critical factor.

**-Low resource requirements:** fewer computational resources such as memory and processing time, helps mainly in resource limited scenarios where minimizing usage is an important factor.

**Disadvantages:**

**-Lack of precision:** Easy-to-calculate heuristics have limited information meaning they usually consider only a part of the features in the problems domain, which often leads less accurate estimation of the value of the state ultimately leading to suboptimal decisions or missed opportunities, this can be problematic in scenarios that have high complexity, accuracy and strategic requirements.

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?

There isn’t necessarily a bug, minimax works with the assumption of a perfect opponent whose only goal is to make us lose, all while trying to maximize the score available against him, meaning it doesn’t guarantee optimal moves in all situations.

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.
   2. בהנחה והדבר היחיד שכל סוכן רוצה הוא שלא תנצחו.
   3. בהנחה שכל סוכן רוצה שהסוכן שאחריו בתור ינצח.
3. In every turn each agent chooses the maximum score/utility for said agent, meaning that every agent wants to win and doesn’t care about us, so at every node in the minimax tree the choice will be max.
4. The other agents want us to lose, so in every node/turn that is not ours the agents will choose the minimum utility for us (min), and at our turn we will choose the node that maximizes our utility (max).
5. Similar to **a** but with slight changes, at each node we still choose the max out of all the children but not the max for the same agents turn, we choose the max for the agent next in turn because each agent needs to maximize the next in line agents utility.

**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.

Yes, the agent will behave differently given enough resources, for example time, mainly because he has more time to explore the minimax tree because of the pruning i.e. not exploring child nodes that have less potential for being an optimal move, also we can expect alpha beta pruning minimax to be faster.

**חלק ד - Expectimax**

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

If we want to use expectimax against an agent with random steps, we should use equal probability for each possible action.

The opponent moves randomly, meaning that each possible action has the same probability. in contrast to minimax, we cant assume a certain behavior of the opponent, thus we need to explore the tree equally and explore all possible moves, and then use the average game outcome to evaluate our moves.

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.

We can apply the same principles of alpha-beta pruning in the probabilistic nodes, this is because we know that the heuristic value is 1 at most, thus after visiting some children, we can calculate the upper limit of the and apply pruning accordingly.

let “ current\_expectencies ” be the expected values of children already seen, and “ sum\_probabilities “ be the sum of the probabilities of these children.

Lets prove this for the general situation where there are n children and we iterated over k children of them.

The formula for the total expectancy is:

Total\_expectancy = current\_expectancy +

And since the upper limit is reached when all expectations are 1, we can substitute expectimax(i) = 1 to find the limit.

Total\_expectancy

We can use the complimentary probability principle to substitute the sum of pi for 1-sum\_probabilities, since we know that they sum to 1, and get:

Total\_expectancy

We got this upper bound, we can use it in the algorithm as follows:

if while iterating we found that the upper bound is less than alpha this means that remaining children are irrelevant to parent max node, so we prune the remaining children.

We can describe the algorithm as follows:

* + - 1. *Initialize : current\_expectancy = 0 , sum\_probabilities = 0 , i=0*
      2. *For i = 1,2,3,…,n:*

*2.1 current\_expectancy += pi \* expectimax(i)*

*2.2 sum\_probabilities += pi*

*2.3 if alpha >= current\_expectancy + 1 – sum\_probabilities:*

*2.3.1 break*

*3. return current\_expectancy*

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)

**The branching factor will not change**, it will not be bigger because we did not add a new action to the action list or other features that give the robot at a specific state new possible state to advance to. But it’s the opposite, we constricted the robot by adding barriers that the robot can detect but never move to, this means that there will probably be child nodes In the decision tree that correspond to the barriers that the robot chooses to not explore, thus not changing the branching factor **(**we might even say that the branching factor gets smaller because there barrier nodes are seen but never explored**)**.

* 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.

**The branching factor will be bigger**. Since we added a new action to the action list, this means that at each state of the robot there will be more potential states to advance to.

Before the change the robot had 7 possible moves (up, down, right, left, pickup package, drop package, charge) making the branching factor equal to 7.

After the change, we still have the same 7 possible moves, but we add to them the action to put a barrier, so that’s a new action for every empty slot, on the board,

Let N be the number of empty slots on the board, **the new branching factor equals to 7+N.**

**N** is equal to the number of **free slots** on the board, which is 5\*5=25 total slots minus the occupied slots by robots, charging stations, packages, and package destinations, this can vary depending on the state of the board, it can be 8 if all objects are separated, or 4 if the two robots have picked packages, and are standing on charging stations, and any number between 4-8 for different board states, which means **N is in the range [17,21].**

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).

Yes, greedy search algorithm, this is because when considering only the time complexity of one step, greedy complexity grows **linearly** with the branching factor, because it only compares immediate actions based on rewards and doesn’t consider future possibilities.

In contrast to the other algorithms that will need to look further into the decision tree in order to calculate one step, the complexity of returning one step will grow **exponentially** with the branching factor.

* 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.

We can use MCTS (Monte Carlo tree search), because as we saw in class, it can work on decision trees with bigger branching factors more efficiently than previous algorithms.

**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 .

9/43

11/27

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

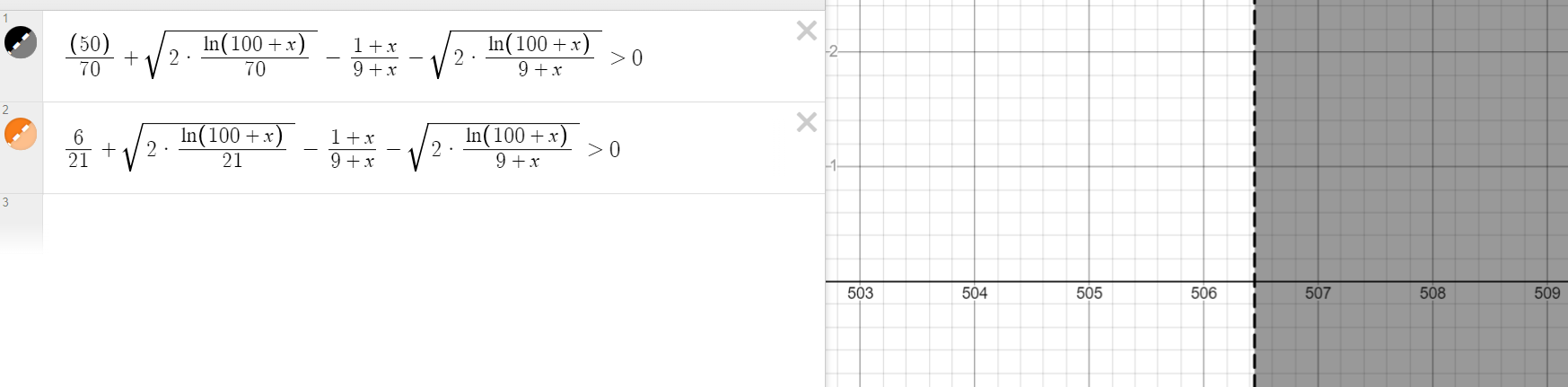


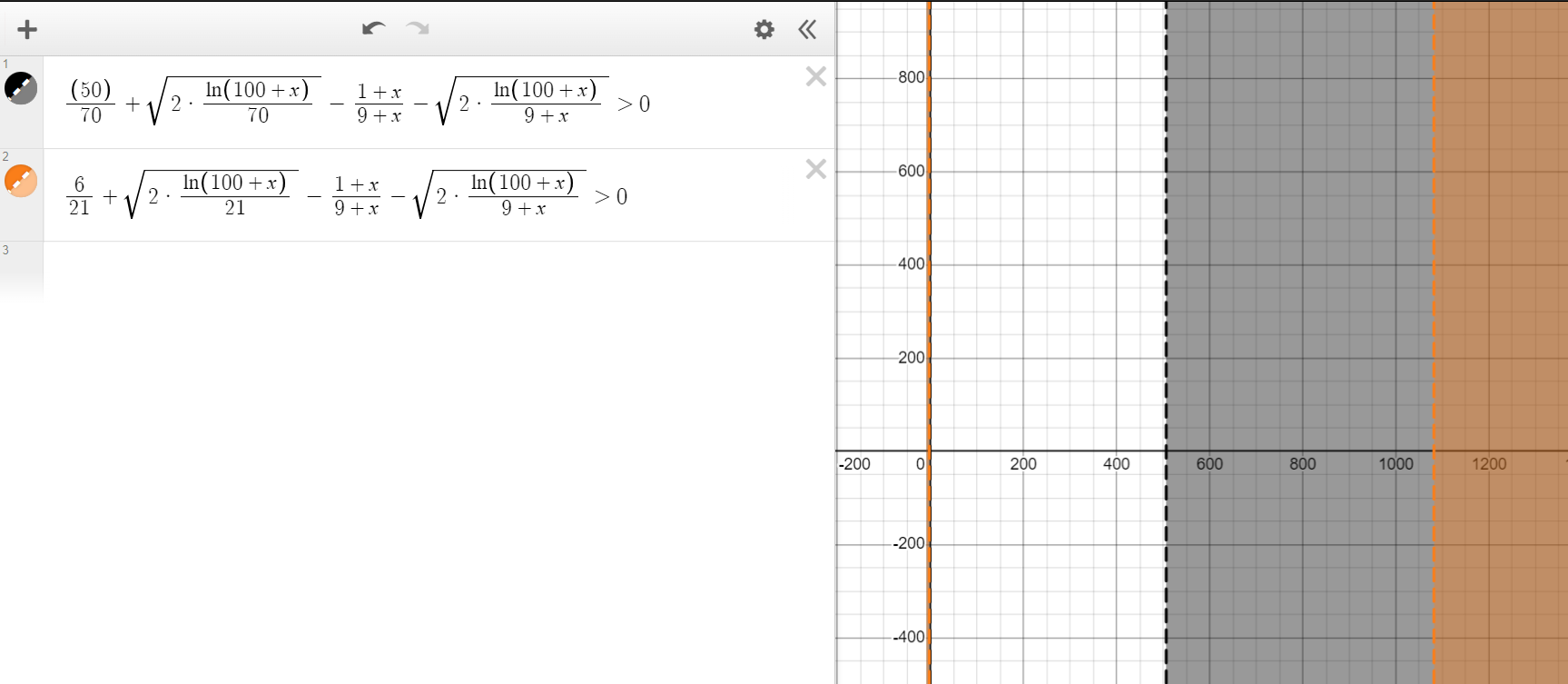
That means the next node in the selection process is b.

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)?

Now our functions look like this depending on the selection process, we use Desmos to help with the calculations and find the minimal N that changes the selection from **b** to another node:

-Assuming that the selection process is the same as 6.2 meaning we will start by choosing **b**, the following graph shows that after **N = 507** we will start to choose **a**.





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.

Since in UCB1, exploitation is divided by N(s) and exploration is divided by N(s)^0.5, simply increasing N(s) will favor exploration.

For example, if N\*(s) = 4\*N(s). then exploitation got smaller by a factor of 4, while exploration got smaller by a factor of 2.

The Math: