תרגיל הגשה

שם הקורס: מבוא לבינה מלאכותית

מספר התרגיל: 2

מגישים: עופר ניסים – ת.ז 312367576

רועי קרניץ – ת.ז 205907777

תאריך: 5.1.23

תרגיל בית 2 מבוא לבינה מלאכותית

Ex2 Introduction to AI

Gobblet Gobblers

(enhanced TicTacToe)

הקדמה ואדמיניסטרציה

הנחיות כלליות

- תאריך הגשת התרגיל: **5.1.2023**
- 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 בצורה מקיפה באמצעות בדיקות אוטומטיות – אשר ישוו את המימוש שלכם למימוש שלנו. חשוב לעקוב בזהירות אחר הוראות התרגיל מאחר שסטיות מהמימוש המבוקש עלולות להוביל לכשל בטסט האוטומטי (גם אם המימוש "נכון ברובו"). במהלך הבדיקה האוטומטית יינתן זמן סביר לכל הרצה – כך שכל עוד תעקבו אחר ההוראות, המימוש שלכם יעמוד בהגבלות הזמנים ויחזיר תוצאות טובות מספיק בשביל הטסט.
- 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.
- ציון התרגיל המקסימלי הינו **110**כיוון שיש בונוס - פירוט על הבונוס בסוף החלק הרטוב

הוראות הגשה

בתוך קובץ זיפ עם השם : HW2_AI_id1_id2.zip

את הדו"ח היבש בפורמט הבא : id1_id2.pdf

ואת הקובץ: subbmision.py שבו אתם ממשים את האלגוריתמים

היכרות עם המשחק

Introduction with the game

לוח וכלי משחק:

לוח משחק של 3X3

2 שחקנים

לכל שחקן: 6 כלים

2 בגודל קטן

2 בגודל בינוני

2 בגודל גדול

המשחק בכללי:

גרסה משופרת של איקס עיגול, השחקן המנצח הוא זה שהצליח למלא שורה, טור או אלכסון בסימנים שלו (הגודל לא משנה).

שחקנים יכולים לזלול שחקנים אחרים ובכך לשנות את הצבע של משבצת קיימת.

להלן סרטון אשר עוזר להבין בכלליות את מהות המשחק:

How to Play Gobblet Gobblers

המשחק בספציפי:

- כל שחקן בתורו מניח גובלין על הלוח, או על משבצת ריקה או על גובלין קטן יותר
- במקום להניח גובלין חדש על הלוח יכול שחקן בתורו לבחור גובלין שלו שנמצא על הלוח ולהזיז אותו למשבצת חוקית
- גובלין יכול לזלול כל גובלין שקטן ממנו (גם אם הם בצבעים זהים)
- ניתן להניח 3 גובלינים אחד על השני (גדול על בינוני כאשר הבינוני על קטן)
- דגש חשוב: במשחק המקורי אין הגבלת צעדים ובגרסה שלנו אנו מגבילים את מספר התורות במשחק. (אם אין נצחון ונעבור את הגבלת הצעדים נחשיב זאת בתור תיקו)
- דגש חשוב: במשחק המקורי חשוב לזכור מה שיש מתחת לכל גובלין שעל הלוח, אצלנו יודעים תמיד (למחשב יש זכרון טוב 🙂)

להלן ספר החוקים הרישמי של המשחק: Gobblet gobblers rules

מה קיבלתם?

קיבלתם 4 קבצים מרכזיים:

1. Gobblet_Gobblers_Env.py סביבת המשחק אותו ישחקו הסוכנים שלכם - הסבר מפורט בהמשך
2. submission.py קובץ שבו תממשו את הסוכנים שלכם וזהו גם הקובץ אותו אתם מגישים
3. game.py מכיל שתי פונקציות שמטרתן להריץ את המשחק ויעזרו לכם לבדיקה עצמית - הסבר מפורט עליהן בהמשך
4. main.py משמש גם כן לבדיקה עצמית

היכרות עם הסביבה

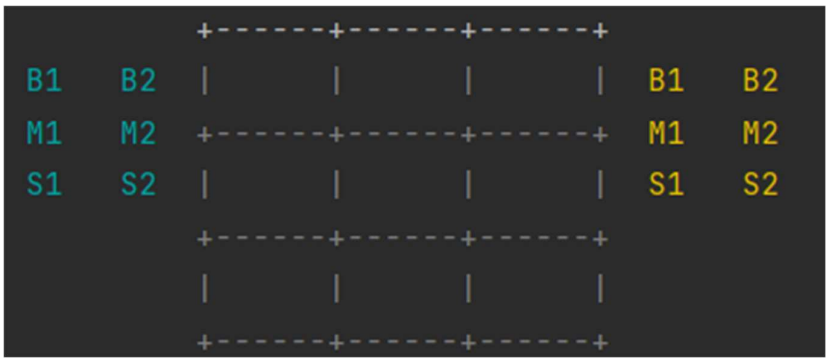
הסביבה איתה תעבדו ממומשת בקובץ Gobblet_Gobblers_Env.py היא מבוססת על סביבה של gym כמו הסביבה איתה עבדתם בתרגיל בית 1.

היא מכילה את המתודות הבאות שקשורות לתפקוד הסביבה:

- **init()** - הקונסטרקטור של הסביבה (המחלקה). ניתן להשתמש בו באופן הבא:

```
env = GridWorldEnv()
```

- **reset()** - מאפסת את לוח המשחק. שמה את כל הכלים בצדדים ולוח המשחק ריק. משתמשים בה באופן הבא `state = env.reset()` פונקציה מחזירה מצב (כדי להבין מה המשמעות של להחזיר מצב תקראו את הפירוט על הפונקציה get_state()), להלן הלוח במצב מאותחל:



*האותיות מייצגות את גודל השחקן (B - גדול, M - בינוני, S - קטן)

- **step()** - פונקציה שמקבלת action ומבצעת את הפעולה. הפעולה צריכה להיות tuple בפורמט (pawn,location) כאשר pawn מייצג את השחקן שתרצו להזיז מבין האופציות {2B1,B2,M1,M2,S1,S} ו-location את המקום על הלוח בו תרצו להניח את הכלי. location הוא מספר בין 0 ל - 8 שמייצג את המשבצת על הלוח באופן הבא:

דוגמא לשימוש בפונקציה

```
env.step(("B1", 7))
```

המשמעות: נזיז את גובלין 1B לאריח מספר 7.

- **render()** - מדפיסה את לוח המשחק דוגמא לשימוש בפונקציה

```
env.render()
```

  - שימו לב, כלים שנמצאים על כלים אחרים יסתירו את הכלים שמתחתיהם, כדי לדעת בדיוק איפה כל כלי ראו get_state().
  - שימו לב שיפתח לכם חלון pygame עם גרפיקה נוחה של המשחק ובו תוכלו לראות את המצב של הלוח לאחר שתבצעו (ליתר דיוק הסוכן שלכם יבצע 🙂) צעדים, בנוסף יש הדפסות לקונסולה שתוכלו להיעזר בהן בדיבוג.

* ישנן פונקציות פנימיות רבות נוספות, מוזמנים לקרוא את התיאור שלהם חלקן אפילו יכולות לעזור לכם בכתיבת היוריסטיקה בהמשך, אך אין צורך להתעמק בהן.

בנוסף הסביבה מכילה את הפונקציות הבאות שקשורות לאינטגרציה שלה עם הסוכנים שאתם הולכים לממש בתרגיל זה:

get_state() - פונקציה שמחזירה את המצב הנוכחי של הסביבה, משתמשים בה באופן הבא:

```
env.get_state()
```

והצורה שבה המצב של הסביבה מוחזר הוא במבנה של State (שמוגדר בקובץ Gobblet_Gobblers_Env.py).

get_neighbors() - פונקציה שמקבלת טיפוס מסוג state ומחזירה רשימה שמכילה את כל השכנים שלו (כל המצבים שיווצרו מכל הפעולות החוקיות שאפשר להפעיל על אותו מצב) ברשימה בעצם כל איבר הוא tuple של (action, state) כך תוכלו לעבור על המצבים ולהחזיר בקלות את הפעולה עבור המצב שתבחרו מרשימת השכנים.

is_final() - מבלבל אז חשוב לשים לב! פונקציה שמקבלת מצב מסוים ומחזירה :

None– עבור מצב לא סופי.

0 - אם יש תיקו, אם שני השחקנים ניצחו באותו צעד (מצב תקין) או אם עברו 100 תורות (50 לכל שחקן) ואין הכרעה.

1 - ניצחון של השחקן הראשון. <u>שימו לב!</u> במהלך המשחק, התור של השחקן הראשון מצוין בתור 0 ולא 1!

2 - ניצחון של השחקן השני.

- **play_game()** - פונקציה שמריצה משחק בודד. הפונקציה הנ"ל מקבלת מחרוזות שמתארות כל סוכן כפי שמתואר בתצלום של המילון בפונקציה הבאה. בנוסף יש דוגמא בהמשך. הפונקציה מחזירה מיהו המנצח ומדפיסה זאת למסך. מחזירה תוצאות כמו שמחזירה is_final().

```
"human": submission.human_agent,
"random": submission.random_agent,
"greedy": submission.greedy,
"greedy_improved": submission.greedy_improved,
"minimax": submission.rb_heuristic_min_max,
"alpha_beta": submission.alpha_beta,
"expectimax": submission.expectimax
```

- **play_tournament()** - פונקציה המריצה מספר play_games לפי ערך num_games שתתנו לה (אנו נבדוק את הקוד שלכם עם num_games = 50) ובכך מחזירה באחוזים כמה ניצחונות יש לכל שחקן וכמה תיקו היו במשחק. הפונקציה הנ"ל מקבל מחרוזות שמתארות כל סוכן כפי שמתואר בתצלום של המילון הנ"ל. דוגמא בהמשך.

num_games * הוא לא מספר המשחקים שרצים בפועל, בפועל רצים פי 2 משחקים. הוא מייצג כמה משחקים יש בהם כל שחקן הינו השחקן הראשון. (תורו לשחק ראשון)

מתחילים לכתוב!

חלק א - היכרות עם הסביבה (4 נק')

1. (יבש: 1 נק') כנסו לקובץ main והריצו את השורה שמסומנת בהערה ומעליה כתוב 1#PART השורה מריצה את המשחק עם שני סוכנים אנושיים, שחקו אחד נגד השני עד לניצחון וצרפו את ההדפסה של המצב הסופי.

```
        +------+------+------+
B2      |  S1  |      |  B1  |
M2      +------+------+------+   M1    M2
        |      |  B1  |      |
        +------+------+------+
        |  M1  |      |  B2  |
        +------+------+------+
```

2. (יבש: 2 נק') האם ניתן לגרום בתור של סוכן מסוים לביצוע פעולה שתגרום לסוכן האחר לנצח? אם כן הסבירו מדוע מצב שכזה יכול לקרות וצרפו את שני הצעדים האחרונים במשחק שכזה, אם לא, הסבירו מדוע לא יתכן מצב כזה?

ניתן ליצור תרחיש כזה, במצב בו קיימים שלושה גובלינים מאותו צבע בשורה/עמודה/אלכסון, אך אחד מהם מכוסה על ידי גובלין גדול יותר מהצבע השני וזהו תורו של השחקן מהצבע השני. אם שחקן זה יבחר להסיר את הגובלין המכסה ולהעבירו למשבצת אחרת, כך שהמצב שנוצר אינו מקנה לו ניצחון, אזי הרצף של השחקן האחר נחשף ופעולת הסוכן האחרון ששיחק גרמה לאחר לנצח.
שני צעדים אחרונים במשחק שכזה: (השחקן הצהוב הזיז את M1 וחשף את S2 של הכחול ובכך הפסיד)

```
        +------+------+------+
B1  B2  |      |      |  M1  |  B1  B2
    M2  +------+------+------+      M2
        |  S1  |  S1  |      |      S2
        +------+------+------+
        |  M1  |      |      |
        +------+------+------+
```

```
        +------+------+------+
B1  B2  |  M1  |      |  S2  |  B1  B2
    M2  +------+------+------+      M2
        |  S1  |  S1  |      |      S2
        +------+------+------+
        |  M1  |      |      |
        +------+------+------+
```

3. (יבש: 1 נק') מהו המספר המקסימלי של אפשרויות לפעולות שונות (בהנחה שכל פעולה חוקית) שניתן לעשות בתור?

המספר המקסימלי של אפשרויות לפעולות שונות בתור ניתן במצב בו ניתן למקם כל אחד מ-6 הגובלינים (אנו מתייחסים לשני גובלינים מאותו גודל כשונים) בכל אחת מ-9 המשבצות בלוח, כלומר 54 אפשרויות.

חלק ב - **Improved Greedy**

(15 נק')

בקובץ submission.py שקיבלתם ממומש עבורכם סוכן greedy שמשתמש בהיוריסטיקה לא חכמה שנקראת `dumb_heuristic2` היא מחשבת את כמות השחקנים הגלויים (לא נמצאים מתחת לשחקן אחר) שיש לסוכן שלה על הלוח.

*שימו לב! ממומשת ב-submission היוריסטיקה שנקראת `dumb_heuristic1` איננו משתמשים בה בשום מקום אך מה שהיא מבצעת הוא : מחזירה 0 - אם המצב אינו מצב סופי. מחזירה 1 אם ניצחנו ו-1- אם הפסדנו או שהיה תיקו. אתם מוזמנים להסתכל עלייה בכדי להבין כיצד להשתמש ב - is_final().

1. (יבש: 6 נק') הגדירו היוריסטיקה משלכם להערכת מצבי המשחק, כתבו נוסחה מפורשת עבור היוריסטיקה. מוזמנים להוסיף תרשים או פירוט מפורט של מה היוריסטיקה עושה בהינתן מצב הלוח והשחקן שמשחק. בחרו בהסבר שמות ברורים ומשמעותיים.

$$h_{smart}(s,k) = \begin{cases} 1000 & P_G(s) \wedge player\ k\ wins \\ -1000 & P_G(s) \wedge player\ k\ loses \\ -50 & P_G(s) \wedge tie \\ VPD(s,k) + TPSD(s,k) & otherwise \end{cases}$$

$P_G(s)$ – Boolean function that indicates if the state s is final (1 if final and 0 else).

$VPD(s,k)$ – Function that finds the Visible Pawns Difference between the two players on the board (for example if player k has 3 visible pawns on the board, while the other has 1 – difference is 2).

$TPSD(s,k)$ – Function that finds the Two Pawns Sequence Difference (in a row, column or diagonal) between the two players on the board (for example if player k has goblins in squares 0, 2, 6 – he has 3 sequences of 2 paws, and the other player has goblins in squares 5 and 7 – he has 0 sequences, so overall the 2 pawns sequence difference is 3).

2. (רטוב: 5 נק') ממשו בקובץ submission.py את `greedy_improved` וממשו את היוריסטיקה החכמה שלכם תחת הפונקציה `smart_heuristic.` (אל תשנו חתימות של פונקציות זה יכשיל את הטסטים!)

greedy_improved הינה פונקציה המקבלת:

- curr_state - מצב נוכחי.
- agent_id - השחקן שמשחק כעת - אם זהו השחקן הראשון יועבר 0 אם זהו השחקן השני יועבר 1.
- time_limit - הגבלת הזמן (בשלב זה אתם יכולים להתעלם ממנה, היא תיהיה רלוונטית באלגוריתמים הבאים).

הפונקציה מחזירה את הפעולה שהסוכן צריך בוחר לבצע כזוג סדור (pawn, location) כפי שפורט בהסבר על הסביבה (זהה לקלט של מתודת step()).

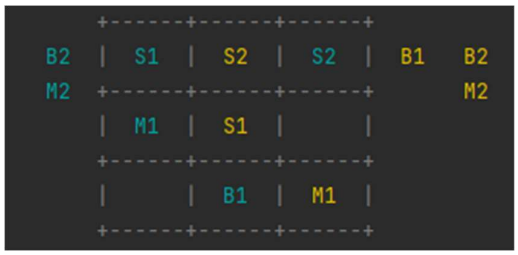
3. (יבש : 2 נק') הסבירו את המוטיבציה לשינויים שביצעתם בהיוריסטיקה האם לפי דעתכם סוכן חמדן המבוסס על היוריסטקיה שלכן (greedy_improved) ינצח את הסוכן החמדן (greedy)? אם כן, פרטו מדוע.

השינויים שביצענו בהיוריסטיקה מסתמכים על ניסיונות משחק מרובים, שאותם בחנו בקפידה, ובאמצעותם הסקנו כי מצב הלוח שאליו נשאף להגיע כשחקנים הוא מצב בו מספר השורות/עמודות/אלכסונים המכילים 2 גובלינים מהצבע שלנו, גדול ככל האפשר. בנוסף, נעדיף גם לזלול גובלין של היריב, אם ניתן, כדי "לבסס שליטה" על הלוח בעזרת הגדלת כמות הגובלינים שלנו ו"למזער שליטה עוינת" של היריב ע"י הקטנת מספר הגובלינים שלו.

לדעתנו, הסוכן החמדן המבוסס על היוריסטיקה שלנו (h_{smart}) אכן ינצח את הסוכן החמדן.
הסוכן החמדן רק מנסה למקסם את כמות הגובלינים הכוללת על הלוח, אך זוהי אסטרטגיה בעייתית שאף עשויה לגרום לו לבחור מהלכים שאינם אופטימליים ובכך להפסיד, כפי שניתן לראות בדוגמה שבהמשך. לעומת זאת, הסוכן שלנו לוקח בחשבון מספר גורמים שונים, שאכן מובילים אותו ליתרון מול היריב (כמו הפחתת מספר הגובלינים של היריב והגדלת שלו, ושאיפה למצבים המובילים למצב מנצח – כאלה שבהם יש מספר גדול של זוגות גובלינים שלו בשורה/עמודה/אלכסון).

<u>דוגמה:</u> במצב שלמטה זהו תור השחקן הצהוב. לפי היוריסטיקה dumb2 שבה עושה שימוש הסוכן החמדן (greedy), הצעד הבא שיבצע הצהוב יהיה למקם גובלין במשבצת כלשהי, בה יש גובלין כחול או ריקה, כדי למקסם את כמות הגובלינים הצהובים על הלוח. אם כן, ייתכן שיבחר למקם במשבצת ריקה (5 או 6) ולא במשבצת בה יש גובלין כחול (0 או 7). צעד זה אינו אופטימלי, שכן אם יבחר למקם גובלין כלשהו, מבין אלה שעדיין לא בלוח, במשבצת 0 – הדבר יוביל לניצחון השחקן הצהוב. כפי שהוגדרה היוריסטיקה h_{smart}, אם

השחקן הצהוב ישחק לפיה – הוא יבחר בצעד האחרון שתואר, כך שיגיע לניצחון, וזאת היות שהיוריסטיקה נותנת משקל גדול למצב בו יש לשחקן הנוכחי רצף של 3 גובלינים בשורה/עמודה/אלכסון על גבי הלוח.



4. (רטוב: 2 נק') הריצו את השורות שנמצאות בהערה ומעליהן 2#PART וצרפו את התוצאות שיודפסו למסך כתוצאה מכך. אתם בעצם תריצו:
   - חמדן נגד אקראי (random)
   - חמדן משופר נגד אקראי (random)
   - חמדן נגד חמדן משופר

תוצאות:

- חמדן נגד אקראי (random):

ties: 0.0 % greedy player1 wins: 90.0 % random player2 wins: 10.0

- חמדן משופר נגד אקראי (random):

ties: 0.0 % greedy_improved player1 wins: 100.0 % random player2 wins: 0.0

- חמדן נגד חמדן משופר:

ties: 0.0 % greedy player1 wins: 0.0 % greedy_improved player2 wins: 100.0

חלק ג - **RB heuristic MiniMax**

(20 נק')

1. (יבש: 2 נק') מה היתרונות והחסרונות של שימוש בהיוריסטיקה קלה לחישוב לעומת היוריסטיקה קשה לחישוב בהינתן שהיוריסטיקה הקשה לחישוב יותר מיודעת מהקלה לחישוב (נותנת אינפורמציה טובה יותר לגבי השאלה מהו מצב טוב)? בהינתן שאנו ב-min max מוגבל משאבים.

כאשר אנו משתמשים באלגוריתם min-max מוגבל משאבים, יתרון עיקרי של שימוש בהיוריסטיקה קלה לחישוב הוא היכולת להתקדם לעומק גדול יותר בעץ ההחלטות ובכך לאפשר בחירה מיודעת יותר ביחס לעולם. זאת לעומת היוריסטיקה קשה לחישוב, שלמרות שאמורה להעניק מידע רב יותר על העולם ולאפשר בחירה מושכלת יותר, היא דורשת משאבים רבים אשר פוגעים באפשרותה להתקדם לעומק כמו היוריסטיקה קלה לחישוב. משמעות עומק קטן בעץ ההחלטות היא "תמונת מצב" מדויקת פחות ויכולת חיזוי של מעט צעדים קדימה.
בהתאם לתכונות שתוארו לעיל, החיסרון של שימוש בהיוריסטיקה קלה לחישוב הוא קבלת החלטות מיודעת פחות, כלומר מדויקת פחות, בניגוד להיוריסטיקה קשה לחישוב שמייצגת באופן מהימן יותר את המצבים בעולם. ניתן לראות כי קיימת במקרה זה פשרה בין שתי היוריסטיקות, המתגלמת בעומק עץ ההחלטות וברמת המיודעות בכל שלב.

2. (יבש: 3 נק') דני מימש את האלגוריתם RB-heuristic-MiniMax והריץ אותו ממצב s בו קיימת פעולה בודדת שמביאה לניצחון. דני נדהם לגלות שהאלגוריתם לא בחר בפעולה זו. האם בהכרח יש טעות באלגוריתם שדני כתב? אם כן, הסבר מה הטעות, אחרת, הסבר מדוע האלגוריתם פעל באופן זה.

האלגוריתם יכול לפעול באופן זה מבלי שיש בו טעות, למשל במצב בו קיים ערך היוריסטי זהה למספר פתרונות (מצבי ניצחון), כאשר אחד נמצא ברמה הנוכחית בעץ ואחר נמצא ברמה עמוקה יותר (המגיעה מצומת הבן). אנו יודעים כי האלגוריתם מסייר בעץ באופן Post order, כך שמגיע ראשית לצמתי הבנים ורק לאחר מכן לצומת האב, מה שעשוי להביא ראשית לבדיקת מצב ניצחון בעומק, עדכון הערך ההיוריסטי באופן חד פעמי (לא יתבצע בהמשך עדכון של ערך זהה בצומת האב) ובחירה במסלול שיביא לניצחון בעומק על פני ניצחון מידי, אשר נמצא בעומק 1.

3. (יבש: 3 נק') למדתם בהרצאות ובתרגולים גישה שנקראת anytime search. כיצד היא מתמודדת עם הגבלת הזמן ? איזה בעיה נפוצה יש באיטרציה האחרונה ואיך פותרים אותה?

גישת anytime search מתמודדת עם הגבלת הזמן באמצעות העמקה הדרגתית עד תום הזמן הנתון, עד עומק שנסמנו k, והחזרת פתרון של האיטרציה האחרונה שהסתיימה במלואה (בעומק k-1). נשים לב כי בניגוד לאלגוריתמים אחרים, שלפעמים לא מחזירים פתרון, האלגוריתם הנ"ל תמיד נותן פתרון (גם אם לא אופטימלי). הבעיה הנפוצה באיטרציה האחרונה היא שהאלגוריתם לא מספיק לחשב את הערכים של כל הצמתים, מחזיר פתרון מהאיטרציה ה--k) (1 ואינו לוקח בחשבון את האיטרציה האחרונה, כך שמתבצעים חישובים מיותרים בעומק k. ניתן לפתור זאת על ידי אחסון ערכי המינימקס בכל איטרציה, כך שבתום האיטרציה האחרונה נבחן את ערכי המינימקס שחושבו באיטרציה k לצד ערכי המינימקס שטרם חושבו באיטרציה זו, עבורם ניקח את הערכים השמורים מהאיטרציה ה-(k-1), ונחזיר פתרון אופטימלי בהתאם – שכעת מתחשב גם בערכים מהאיטרציה האחרונה, שלא הסתיימה.

4. (יבש: 4 נק') נניח שבסביבה היו K שחקנים במקום 2 (תחשבו על משחק כללי לאו דווקא המשחק שלנו, אך עדיין משחק סכום אפס). אילו שינויים יהיה צריך לעשות במימוש סוכן Minimax?
   a. בהינתן שכל סוכן רוצה לנצח ולא אכפת לו רק מכם
   b. בהנחה שכל סוכן שונא אתכם והדבר היחיד שאכפת לו זה שלא תנצחו

a. במקרה זה נשנה את אופן פעולתו של כל סוכן להיות שחקן מקסימום, כך שישאף לקדם את עצמו בעזרת ערכים טובים יותר שיוחזרו ע"י האלגוריתם.

b. במקרה זה נדאג לשנות את אופן פעולתו של כל סוכן להיות שחקן מינימום, ובכך כל הסוכנים היריבים ינסו לפגוע ב"ניקוד" שלנו וינסו למנוע מאיתנו לנצח.

5. (רטוב: 8 נק') עליכם לממש את הפונקציה `rb_heuristic_min_max` בקובץ submission.py. שימו לב כי הסוכן מוגבל משאבים, כאשר המשתנה time_limit מגביל את מספר השניות שהסוכן יכול לרוץ לפני שיחזיר תשובה.(בסעיף זה אסור להשתמש בגיזום כדי לפתור את הבעיה - אל תדאגו בחלק הבא יהיה לכם גיזום).

* מומלץ להריץ את הסוכנים שלכם אחד נגד השני כמו בחלק א ולראות כי אתם מקבלים תוצאות הגיוניות

* שימו לב שהגבלת הזמן השתנתה מ-1 שנייה ל - 80 שניות

*השינויים הם בקובץ game.py

חלק ד - **Alpha_Beta**

(20 נק')

1. (רטוב: 10 נק') כעת אתם ממשו סוכן אלפא בטא, בקובץ submission.py ממשו את הפונקציה `alpha_beta` כך שתבצע גיזום כפי שנלמד בהרצאות ובתרגולים.
2. (יבש: 3 נק') האם הסוכן שמימשתם בחלק ד' יתנהג שונה מהסוכן שמימשתם בחלק ג' מבחינת זמן ריצה ובחירת מהלכים?

הסוכן שמימשנו כעת יבצע גיזום לתתי עצים שאינם רלוונטיים לחישוב, דבר שיחסוך זמן ויאפשר לו להגיע לעומק גדול יותר. אופן קבלת ההחלטות של סוכן זה לא שונה מזה של הסוכן מחלק ג', אבל השימוש בגיזום עשוי להביא לזמן ריצה קצר יותר ולבחירת מהלכים שעשויה להיות טובה יותר, דבר שנובע מבדיקה של צמתים נוספים בזמן שנחסך. בהינתן הגעה לעומק זהה תחת מגבלת הזמן, שני הסוכנים צפויים לפעול באופן דומה, אך אם נותר לסוכן מחלק ד' זמן לסיים מעבר על פתרונות באיטרציה הבאה – כנראה שאופן פעולתו יהיה עדיף.

3. (יבש: 2 נק') למדתם מספר שיטות לשיפורים של alpha_beta אחת מהשיטות הללו נקראת ספריות פתיחה / סיום. מה שיטה זו מבצעת ומדוע היא עוזרת לגזום נתח מהעץ?

שיטה זו למעשה שומרת רצפי משחק למצבי פתיחה וסיום נפוצים ומקובלים – שמספרן לרוב מצומצם, ומאפשרת שליפה מהירה של מצבים אלה בעזרת הספרייה, ללא צורך בחישוב מלא של ערכם לפי min-max. היא עוזרת לגזום נתח מהעץ בזכות העובדה שבהגעה למצב המהווה פתיחה/סיום, ניתן לחסוך את החישובים הנדרשים ל-alpha_beta ולבחור מיידית במצב הפתיחה/סיום האופטימלי, כך שבמקרים רבים אין צורך לחשב ערכים בתת-העץ ומתבצע גיזום.

4. (יבש: 3 נק') למדתם על גישה נוספת שנקראת טבלאות מצבים. מה השיטה מבצעת, איך היא שומרת על נכונות, ומדוע היא עוזרת "לגזום" מהעץ?

בשיטת טבלאות המצבים אנו שומרים ערכי מינימקס של המצבים בטבלה ומשתמשים בערכים השמורים של מצב אם הגענו לאותו מצב שוב. בנוסף נשמור עומק חיפוש המשוקף ע"י הערך, שמסייע בהחלטה על שימוש בערכים מהטבלה. שמירה על נכונות, ע"י קיום משפט ההבטחה של מינימקס, מתקיימת עם שימוש בערכים מהטבלה רק אם עומק החיפוש שהערך משקף שווה בדיוק לעומק החיפוש שנותר. בנוסף, אם עומק החיפוש שהערך משקף גדול או שווה מהעומק שנותר בקריאה הנוכחית למינימקס - יתבצע גיזום של תת העץ, שכן כבר ביקרנו במצב זה והמשחק שלנו דטרמיניסטי. עבור מצבים שלא קיימים בטבלה, נתעד ונשמור לשימוש עתידי.

5. (יבש: 2 נק') בהרצה $\alpha\beta - Minimax$ המתכנת מחק בטעות את הצעד המומלץ ונשאר רק עם ערך המינימקס של העץ. כיצד ניתן לשפר את יעילות ההרצה החוזרת באמצעות שינויים פשוטים באלגוריתם? תארו את התנהגות האלגוריתם המתוקן. הסבירו כיצד יתנהג במקרה הטוב ביותר, במקרה הרע ביותר, ובמקרה הכללי.

ניתן לשפר את יעילות ההרצה החוזרת ע"י עצירת ההרצה, כאשר ערך המינימקס של בן כלשהו של שורש העץ זהה לערך המינימקס של העץ מההרצה הקודמת, ונחזיר את הצעד המומלץ, שהוא הצעד בין השורש לבן המתאים. שיפור זה יביא לביצוע מהלך מוצלח, משום שאם היה קיים מהלך טוב יותר שיוצא מאחד הבנים של השורש – בהרצה הראשונה היה מתקבל ערך מינימקס גבוה יותר. בנוסף, אם במהלך ריצת האלגוריתם נמצא בן של מצב מינימום עם ערך נמוך מערך המינימקס הידוע של העץ, נוכל לבצע גיזום של תתי העצים של הבנים האחרים (משום ששחקן המינימום יבחר ערך קטן או שווה לזה של הבן שמצאנו עם הערך הקטן, כך שבכל תת העץ שלו לא נמצא את הצעד המומלץ, המחזיק בערך המינימקס הגבוה). באופן דומה נוכל לבצע גיזום עבור בן של מצב מקסימום המחזיק ערך הגבוה מערך המינימקס של העץ. במקרה הטוב ביותר – הצעד הנ"ל יוחזר מהרצת מינימקס על הבן הראשון (השמאלי), במקרה הרע ביותר – הצעד יוחזר מהרצת מינימקס על הבן האחרון (הימני) לאחר ריצה על כל הבנים האחרים של השורש ובמקרה הכללי – הצעד המומלץ יוחזר מהרצת מינימקס העוברת על כמחצית מצמתי העץ.

חלק ה - **Expectimax**

(20 נק')

1. (יבש: 2 נק') סוכן Minimax (עם או בלי שיפורים) מניח כי היריב בוחר בכל צעד בפעולה האופטימלית עבורו, מה בעייתי בגישה הזו ואיך אלגוריתם Expectimax מתגבר על הבעייתיות שתיארתם?

הבעייתיות בגישה שלעיל היא שייתכן שבפועל היריב יבחר בצעד כלשהו בפעולה שאינה האופטימלית עבורו, אך סוכן המינימקס עדיין מניח כי מדובר בפעולה אופטימלית ועל כן יבחר בצעד "פחדני" (שאינו מיטבי) בשל הנחה זו.
אלגוריתם Expectimax מתגבר על הבעייתיות בעזרת שימוש בהתפלגות הסתברותית בהגעה לכל מצב (המותאמת לעולם המאפיין את הבעיה) וחישוב תוחלת על ערכי המינימקס של הבנים לפי התפלגות זו, כך שהסוכן מניח התנהגות "ממוצעת" של היריב במקום "אופטימלית" – הנחה שמתאימה יותר לתיאור העולם.

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

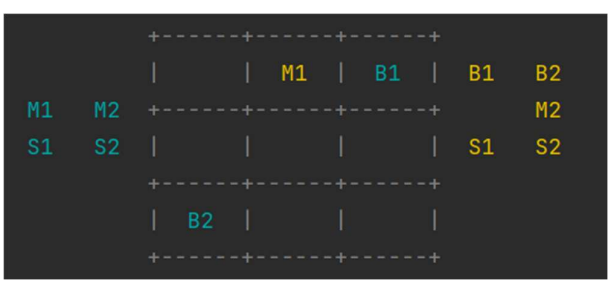
במקרה זה נניח התפלגות אחידה על המהלכים השונים ונשתמש בהסתברות יוניפורמית לחישוב התוחלת (הסתברות השווה לאחד חלקי מספר הבנים). נבחר בהסתברות זו בשל העובדה שמדיניות משחק רנדומלית עשויה לבצע כל מהלך באותו סיכוי, כלומר בהסתברות זהה לכל המהלכים.

3. (יבש: 5 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 $\forall s: -6 \leq h(s) \leq 6$
   a. איך ניתן לבצע גיזום לאלגוריתם זה? תארו בצורה מפורטת את תנאי הגזימה, והסבירו את הרעיון מאחוריו.
   b. הציגו דוגמה להיוריסטיקה כזאת עבור המשחק בתרגיל שלנו וצרפו דוגמא ללוח עבור כל אחד מהמצבים הבאים:
      - מצב המקיים $h(s) = 6$
      - מצב המקיים $h(s) = -6$

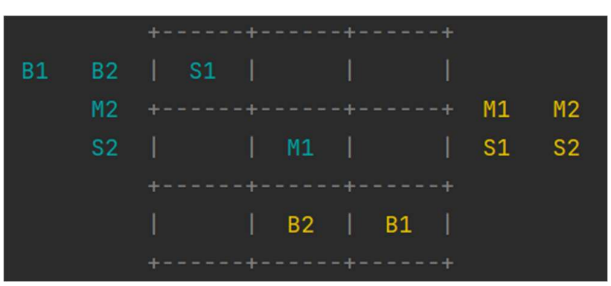
a. נוכל לבצע גיזום לאלגוריתם המתואר באמצעות שימוש בערך ההיוריסטי המקסימלי/מינימלי שהוגדר: 6/-6. למשל אם עבור שחקן max נמצא שערך התוחלת שחושב עבור אחד הבנים שלו הוא 6, נגזום את תתי העצים השייכים לבנים האחרים, שכן בחישוב התוחלת עבורם לא נוכל להגיע לערך הגבוה מהמקסימלי (ערכי התוחלת נעים בין 6 למינוס 6, בהתאם לערכי הפונקציה ההיוריסטית). באופן דומה נוכל לבצע גיזום עבור שחקן מינימום אם מצאנו בן עם תוחלת שערכה 6-.

b. נבחר בהיוריסטיקה הנותנת 3+ עבור כל גובלין גדול שלנו שנמצא על הלוח ונותנת 3- עבור כל גובלין גדול של היריב שנמצא על הלוח.

- דוגמה למצב המקיים $h(s) = 6$:
(השחקן שלנו הוא הכחול)



- דוגמה למצב המקיים $h(s) = -6$:
(השחקן שלנו הוא הכחול)



4. (רטוב: 10 נק') כעת תממשו סוכן expectimax ע"י כך שתממשו בקובץ submission.py את הפונקציה `expectimax,` תחת ההנחה כי היריב שלנו אוהב אקשן ומחליט שאת כל הפעולות מבצע בהסתברות שווה למעט שתי סוגי פעולות:
   a. לפעולות שבהן יכול לאכול חייל אחר יש הסתברות גבוהה פי 2 משאר הפעולות
   b. הוא מעדיף חיילים קטנים (בגודל S) ולכן לפעולה שמערבת שחקן S יש גם כן הסתברות גבוהה פי 2 משאר הפעולות

בונוס בונוס בונוס בונוס בונוס

כפי שאתם יודעים האלגוריתמים שמימשתם הינם אדברסיאלים, משמע מתחרים אחד בשני, ולכן אנו מזמינים אתכם לכתוב סוכן שיתחרה בסוכנים של שאר הסטודנטים בקורס עליכם לממש אותו תחת הפונקציה supre_agent יש לכם יד חופשית בבחירת האופן בה תממשו את הסוכן (יכולים לבחור אפילו אחד מהאלו שמימשתם בתרגיל). שני הזוגות שינצח מול הכי הרבה קבוצות אחרות יקבלו 10 נק' בונוס לציון הסופי של תרגיל הבית הנ"ל. ארבעת הזוגות שאחריהם יקבלו 5 נק' בונוס.

שאלה פתוחה - **Monte Carlo Tree Search**

(21 נק')

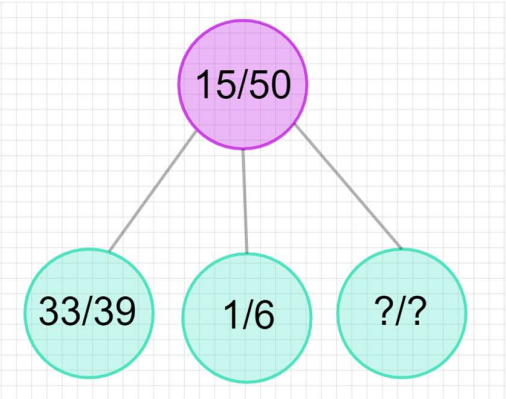
1. (2 נק') הסבירו את המונחים הבאים:
   a. exploitation
   b. Exploration

a. Exploitation הוא מונח המתאר אופן פעולה המתמקד בביצוע מהלכים מוכרים, אשר העניקו לנו תועלת בעבר. אופן פעולה זה מניח כי נקבל שוב תועלת דומה ולכן הוא עדיף על חוסר וודאות.

b. Exploration הוא מונח המתאר אופן פעולה המתמקד בביצוע מהלכים שקיים חוסר ודאות בנוגע לתועלת שנקבל מהם. גישה זו שואפת לגלות מהלכים חדשים אשר יעניקו תועלת טובה יותר משל מהלכים מוכרים ובנוסף לקבל ידע טוב יותר אודות העולם.

אנדריי ואופק החליטו לשחק קאטאן (נניח שזה משחק לשני אנשים), הם החליטו שהמשחק פשוט מידי והחליטו להוסיף את ההרחבות: "ערים ואבירים" ו"יורדי הים". אחרי שהבינו שהגזימו וכעת אינם בטוחים מה הצעד החכם ביותר ובגלל שה-branching factor גדול מאוד החליטו להשתמש ב- MCTS (Monte Carlo Tree Search) (דגש - זהו משחק סכום אפס שנגמר בניצחון של אחד הצדדים)

להלן עץ שמתאר שני שלבים במשחק. כחול זה תור של אופק וורוד זה תור של אנדריי (פחות רלוונטי לשני הסעיפים הראשונים)



2. (2 נק') עבור העץ הנ"ל שמתאר שני שלבים במשחק עבור העלה עם ה- (?) השלימו מהו הערכים שאמורים להיות במקום הסימני שאלה.

הערכים עבור הבן הימני הם: 1/5. זאת היות שבכל רמה המספר הימני (מכנה) מייצג את מספר הסימולציות שבוצעו והמספר השמאלי (מונה) מייצג את מספר הניצחונות של השחקן שזהו תורו ברמה זו. ברמה העליונה אנו רואים שבוצעו 50 סימולציות, מתוכן 15 הסתיימו בניצחון של השחקן הוורוד – לכן נסיק כי 35 משחקים ברמה התחתונה הסתיימו בניצחון של השחקן השני ויחד עם שני הבנים הנוספים, נדרש משחק יחיד ($35 - 33 = 1$) מתוך 5 ($50 - 39 - 6 = 5$) כדי להשלים את העץ.

3. (7 נק') חשבו את $UCB_1(s)$ עבור הצמתים הכחולים עם ערך $C = \sqrt{2}$, השוו בין הערכים שיצאו לכם בחישוב ה - $UCB_1(s)$, הסבירו מה המשמעות שלהם. האם יש צמתים (רק מהכחולים) שהערך שלהם שונה מהותית אחד מהשני אך ה - $UCB_1(s)$ דומה? אם כן הסבירו מדוע מצב כזה קורה ע"פ העקרונות של UCB_1

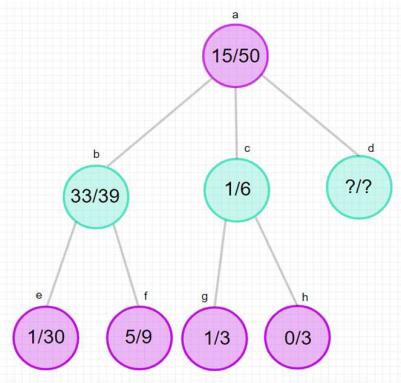
נחשב בעזרת הנוסחה שראינו בכיתה:

$$UCB_1(left\ node) = \frac{33}{39} + \sqrt{2} \times \sqrt{\frac{\ln(50)}{39}} = 1.294$$

$$UCB_1(middle\ node) = \frac{1}{6} + \sqrt{2} \times \sqrt{\frac{\ln(50)}{6}} = 1.308$$

$$UCB_1(right\ node) = \frac{1}{5} + \sqrt{2} \times \sqrt{\frac{\ln(50)}{5}} = 1.451$$

ערך ה-UCB_1 תלוי בערכי ה-exploitation וה-exploration המרכיבים אותו. במצב שבו נראה ערך exploitation גבוה קיימים ניצחונות רבים ביחס למספר המשחקים ובמצב שבו נראה ערך exploration גבוה נבחין כי שוחק מספר קטן יחסית של משחקים. אנו רואים כי ערכי הצומת השמאלי והאמצעי שונים מאוד ($\frac{33}{39}$ לעומת $\frac{1}{6}$) אך ערך ה-UCB_1 שלהם דומה (1.294 לעומת 1.308). ניתן להסביר זאת בעזרת עקרונות ה-UCB_1 , שכן לצומת השמאלי יש ערך exploitation גבוה בשל ריבוי ניצחונות, אך ערך exploration נמוך בשל מספר גדול של משחקים ששוחקו (כלומר מעט חוסר ודאות). לעומת זאת, בצומת האמצעי יש ערך exploitation נמוך בשל מעט ניצחונות, אבל ערך exploration גבוה מפאת מספר המשחקים הנמוך ששוחק (המייצג חוסר ודאות גדול יחסית). מדד זה יוצר איזון בין שני המצבים – מצד אחד נותן משקל לתועלת הנובעת מהיכרות עם העולם (ריבוי ניצחונות בהרצת הסימולציות), ומצד שני גם מעניק משמעות לחוסר הוודאות הטמון במצבים לא מוכרים, אשר עשויים להביא לתועלת גדולה יותר מזו של המצבים הידועים. לכן בשני המצבים הנ"ל, המגלמים ערכי exploitation ו-exploration הפוכים בגודלם היחסי, קיים ערך UCB_1 דומה.



כעת חושפים לנו המשך של שני השלבים שראינו והעץ כעת נראה כך:

4. (4 נק') מצאו וציינו מיהו הצומת הבא שיפותח

הצומת הבא שיפותח הוא הצומת d, אשר מחזיק בערך ה-UCB_1 הגבוה ביותר. הדבר נובע מכך שבוצע בו מספר נמוך של סימולציות ביחס לצמתים האחרים, מה שמגדיל מאוד את ערך ה-exploration שלו.

5. (6 נק') בהנחה ולו יש בן יחיד שלאחר סימולציה ממנו מתקבל הפסד של אופק. ציירו את העץ החדש הנוצר (שנו את ערכי שאר הצמתים בהתאם) לאחר שנפעפע את המידע על הסימולציה שהתרחשה.

<u>להלן העץ החדש:</u>