מבוא לבינה מלאכותית תרגיל 2 חלק יבש

איל רביד, ראובן טימסיט

חלק א

1. נגדיר את $< S, A, f, c, s_0, R >$ כך:

כל מצב בקבוצה S (קבוצת כל המצבים) יהיה וקטור שיכיל את מיקום הרובוט 0, מיקום רובוט 1, מיקום של חבילה מספר 1, מיקום של חבילה מספר 2, נתונים אצל מי נמצאת חבילה מספר 1 וחבילה מספר 2 (יכול להיות שאצל אף רובוט), המיקום המקורי של החבילות, מיקום של תחנות הטעינה, הניקוד הנוכחי של כל אחד מהרובוטים וכמות הסוללה של כל אחד מהרובוטים ובנוסף את הרובוט שתורו לזוז עכשיו ואת מספר הצעדים שנותר לכל רובוט לעשות.

הקבוצה A תכיל את A_0, A_1 כאשר A_i יהיה כל הפעולות האפשריות שרובוט יכול לעשות. כלומר הליכה למעלה, למטה, ימינה, שמאלה, איסוף חבילה, הורדת חבילה וטעינה.

הפונקציה f תוגדרת כך: $f(a, s) = s'$ (a זו פעולה ו s זה מצב) כאשר s' זה יהיה המצב העוקב שנגיע אליו אם הרובוט שתורו לפעול במצב s, יבחר לבצע את פעולה a. לדוגמא אם הוא יבחר לזוז ימינה (כלומר פעולה זו חוקית) אז נגיע למצב זהה למצב הקודם פרט לכך שהרובוט זז צעד ימינה, ירדה לו טעינה אחת והתור עבר לרובוט השני (במידה ולרובוט השני נשארה סוללה).

הפוקנציה c תהיה שווה ל 0 כי אין מחיר לפעולות פרט לירידת סוללה (או לירידת ניקוד במידה ומטעינים) אבל דבר זה מגולם כבר במצבים.

s_0 זה המצב ההתחלתי כלומר טעינה מלאה של שני הרובטוים וניקוד התחלתי 0. מיקומם ומיקום האובייקטים האחרים ישתנה מהרצה להרצה.

הקבוצה R זו קבוצה של פוקנציות פרס לכל שחקן כאשר שתקיים ש $R_i(a, s)$ יהיה שווה לרווח שהשחקן ה i ירוויח אם השחקן ה i בוחר להוריד חבילה (כלומר a תהיה פעולת הורדת חבילה) כאשר הוא במצב s. אחרת, הרווח יהיה 0.

2. נסמן ב R_0 את הרובוט שלנו (שמנסה למקסם את היוריסטיקה) וב R_1 את היריב. נגדיר פונקציה $g_i(s)$ שבעזרתה נגדיר את היורסטיקה. $g_i(s)$ תהיה

תלויה בשלושה מקרים. מקרה ראשון, אם במצב s לרובוט ה i יש פחות נקודות מלרובוט השני ובנוסף, לרובוט ה i יש 6 יחידות סוללה או פחות ובנוסף לרובוט ה i יש יותר מ 6 נקודות (כדי שיהיה מתשלם ללכת להטעין) אז $g_i(s)$ יהיה:

$$g_i(s) = -20 - dist(R_i, charger)$$

כאשר $dist(R_i, charger)$ זה המרחק (מנהטן) המינימלי בין הרובוט ה i צריך למטען כלשהו. כלומר במקרה זה נעדיף לטעון את הסוללה בעדיפות הכי גבוהה. ברגע שהרובוט יגיע לנקודת טעינה, הוא יבחר בפעולת טעינה כי זה יוביל אותו ליוריסטיקה הכי גבוה.

אחרת, אם הרובוט ה i לא מחזיק בחבילה אז נגדיר:

$$g_i(s) = points(R_i) - \frac{dist(R_i, package)}{10}$$

כאשר $dist(R_i, package)$ זה מרחק מנהטן בין הרובוט ה-i לחבילה הקרובה ביותר אליו. כלומר במצב זה נתעדף לקחת חבילה. אחרת (לרובוט ה-i יש חבילה אז נגדיר:

$$g_i(s) = points(R_i) - \frac{dest(R_i, package)}{10} + \frac{R(package)}{2}$$

כאשר $dest(R_i, package)$ זה מרחק מנהטן ליעד החבילה שאצלו ו $\frac{R(package)}{2}$ זה הנקודות שיקבל הרובוט אם יביא את החבילה ליעדה. כלומר במצב זה נרצה לשים את החבילה ביעד שלה.

הסיבה שבחרנו להוסיף ליוריטיקה מחוברים כמו $\frac{dist(R_i, package)}{10}$ היא בגלל שמצד אחד אנחנו רוצים שאם אין לרובוט חבילה אז הוא ינסה להתקרב אליה כדי לאסוף אחת. אבל מצד שני לא נרצה שהמרחק בין הרובוט לחבילה יוריד יותר מדי מהיוריסטיקה כי אז אולי הרובוט יעדיף לא להוריד חבילה שהוא מחזיק ביעד שלה. כלומר, אם במקרה שבו לרובוט אין חבילה אז הנוסחא היתה נראית כך

$$g_i(s) = points(R_i) - dist(R_i, package)$$

אז במידה והרובוט מחזיק בחבילה הוא עלול לא לרצות להוריד אותה כי אז זה יגרום ליוריסטיקה שלו לרדת. לבסוף, מכיוון ש R_0 ינסה למקסם את היוריסטיקה ו R_1 ינסה להביא אותה למינימום אז נגדיר את היוריסטיקה h להיות (במידה והמצב הוא אינו סופי):

$$h(s) = g_0(s) - g_1(s)$$

ובמידה והמצב הוא מצב סופי אז נגדיר:

$$h(s) = (R_0.credit - R_1.credit) * 10$$

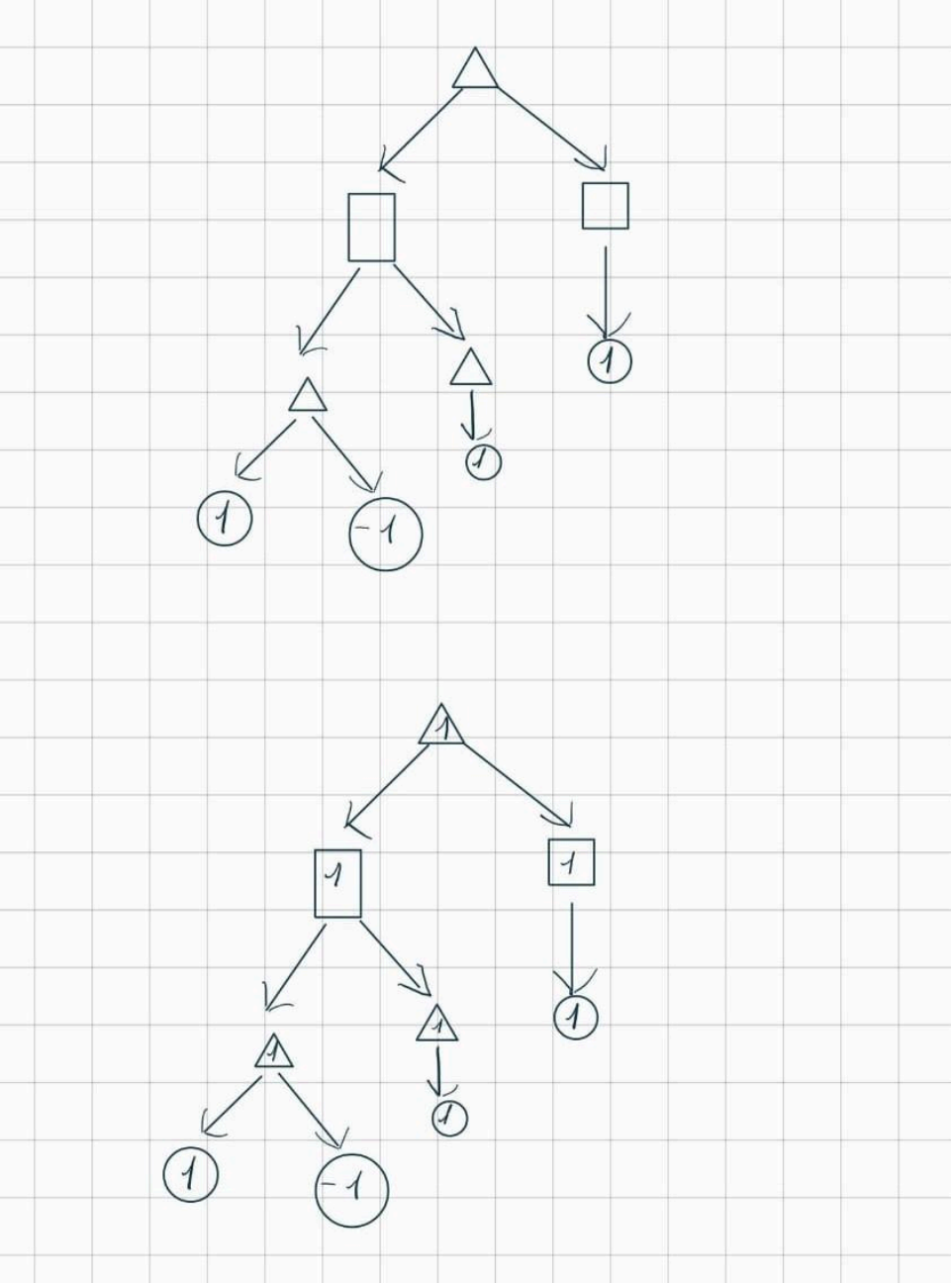
כאשר את ההכפלה ב 10 אנו עושים כדי להבליט מצבים סופיים על פני מצבים לא סופיים.

3. בקוד.

חלק ב

1. באלוגריתם mini-max מוגבל משאבים, היתרון של יוריסטיקה קלה לחישוב (אך פחות מיודעת) לעומת יוריסטיקה קשה לחישוב (אך יותר מיודעת) זה שיקח פחות זמן לחשב את היוריסטיקה הקלה לחישוב. במידה והגבלת המשאבים שלנו היא עומק החישוב, אז אלגוריתם המשתמש ביוריסטיקה קלה לחישוב ירוץ מהר יותר ובמידה והגבלת המשאבים שלנו היא זמן (כלומר נריץ mini-max עד שנגמר הזמן) אז אלגוריתם המשתמש ביוריסטיקה קלה לחישוב יוכל להגיע לצמתים עמוקים יותר בעץ לעומת אלגוריתם שמשתמש ביוריסטיקה קשה לחישוב.
החסרונות של יוריסטיקה קלה לחישוב ופחות מיודעת זה שהיא יכולה להטעות את השחקן בכך שהיא תגרום לו לבצע צעד פחות טוב מאשר שיורסטיקה מיודעת יותר היתה גורמת לו לעשות.

2. דנה טועה מכיוון שאלגוריתם מינימקס לא מתחשב באורך המסלול. דוגמא נגדית לכך תהיה הבאה:



כאשר הגרף העליון זה גרף המצבים בתחילת משחק כלשהו. צמתי משולש הם צמתי מקסימום וצמתי מרובע הם צמתי מינימום וצמתים בצורת עיגול הם מצבים סופיים שבתוכם כתוב את הערך שלהם. הגרף התחתון מתאר את הניקוד של הצמתים אחרי הרצת אלגוריתם mini-max. כפי שהניתן לראות. מהמצב הראשון (השורש) שחקן המקס יכול לבחור ללכת לכל אחד מבניו וזה יבטיח לו את אותו ערך אבל אם הוא יבחר ללכת לבן השמאלי אז כל מסלול למצב סופי בתת עץ זה ארוך יותר מכל מסלול למצב סופי בתת עץ הימני. לכן, במקרה שבו השחקן

יזוז לבן השמאלי (בחירה שאלגוריתם mini-max יכול לגרום לה) אז אומנם יובטח ערך מקסימלי אך בחירה זו בהרכח תגרום לו להגיע לנצחון במספר **לא** מינימלי של צעדים.

3. הפתרון להגבלה במשאב הזמן הוא העמקה איטרטיבית. כלומר הרצה איטרטיבית של RB-mini-max (שמבצע mini-max עד שעומק נתון) כשבכל איטרציה מגדילים את העומק שאליו יגיע RB-mini-max ובסוף מקבלים החלטה לפי הערכים שהחזיר ה RB-mini-max האחרון שסיים לרוץ. כלומר נריץ RB-mini-max לעומק 1 ואז לעומק 2 וכך הלאה עד שיגמר הזמן. אלגוריתמים מסוג זה נקראים אלגרויתמים Anytime search . דוגמא לאלגוריתם מסוג זה שלמדנו בקורס הוא $Anytime\ A^*$. באלגוריתם זה נריץ את wA^* בערכי w הולכים וקטנים מהסיבה שעבור ערכי w גדולים, נצליח למצוא מסלול מהר מאוד (אך לא אופטימלי) וככל שמקטינים את w אז המסלול שנמצא יהיה יותר ויותר טוב אך יקח לנו יותר זמן למצוא אותו.

4. בקוד.

5. במקרה זה בכל מצב סופי יהיה וקטור תועלות X בגודל K כך ש X_i יהיה התועלת שמקבל השחק ה-i אם המשחק יסתיים במצב זה. את הפסודו קוד נכתוב בעברית.

סעיף א.

פונקציית מינימקס (מקבלת מצב s וסוכן agent):

1. אם s מצב סופי אז החזר את וקטור התועלות של מצב זה
2. נסמן ב turn את השחקן שתורו לשחק במצב s.
3. לכל מצב s' שעוקב למצב s חשב מינימקס (s',agent)
4. החזר את וקטור התעלות X' מהוקטורים שמצאת בשלב 3 שעבורו X_{turn} מקסימלי (כלומר הוקטור שממקסם את הרווח של השחקן שתורו לשחק).

סעיף ב.

פונקציית מינימקס (מקבלת מצב s וסוכן agent):

1. אם s מצב סופי אז החזר את וקטור התועלות של מצב זה
2. סמן ב turn את השחקן שתורו לשחק במצב s.
3. לכל מצב s' שעוקב למצב s חשב מינימקס (s',agent)
4. אם $turn != agent$ אז החזר את וקטור התעלות X' מהוקטורים שמצאת בשלב 3 שעבורו X_{agent} מינימלי (כלומר הוקטור שעבורו התועלת עבור השחקן שלנו היא המוכנה ביותר).
5. אחרת $turn == agent$ אז החזר את וקטור התעלות X' מהוקטורים שמצאת בשלב 3 שעבורו X_{turn} מקסימלי (כלומר הוקטור שממקסם את הרווח של השחקן שתורו לשחק).

סעיף ג.

פונקציית מינימקס (מקבלת מצב s וסוכן agent):

1. אם s מצב סופי אז החזר את וקטור התועלות של מצב זה
2. סמן ב turn את השחקן שתורו לשחק במצב s. ונסמן ב turn+1 להיות השחקן שתורו לשחק בתור הבא
3. לכל מצב s' שעוקב למצב s חשב מינימקס (s',agent)
4. אם $turn != agent$ אז החזר את וקטור התעלות X' מהוקטורים שמצאת בשלב 3 שעבורו $X_{turn+1t}$ מקסימלי (כלומר הוקטור שעבורו התועלת עבור השחקן הבא היא הגבוהה ביותר).
5. אחרת $turn == agent$ אז החזר את וקטור התעלות X' מהוקטורים שמצאת בשלב 3 שעבורו X_{turn} מקסימלי (כלומר הוקטור שממקסם את הרווח של השחקן שתורו לשחק).

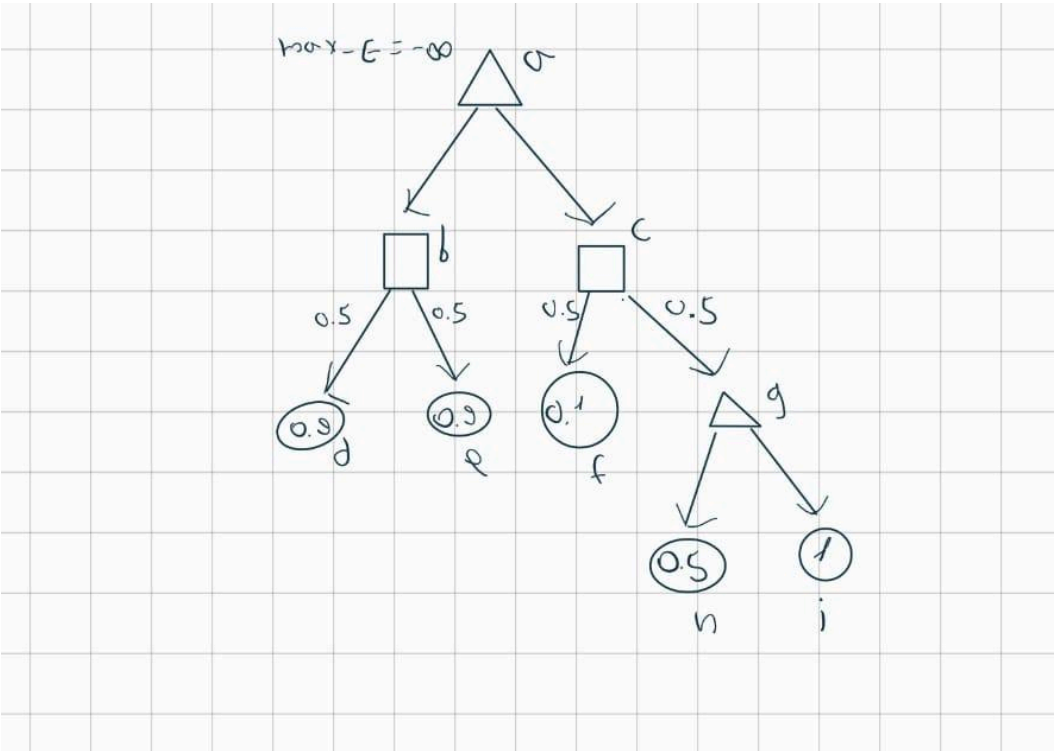
חלק ג

1. בקוד.
2. הסוכן יתנהג שונה מבחינת בחירת המהלכים אך לא יתנהג שונה מבחינת זמן הריצה. הוא לא יתנהג שונה מבחינת זמן הריצה כי שני האלגוריתמים הם מסוג anytime. כלומר שני האלגוריתמים ירוצו עד שיגמר להם הזמן ולכן הם ירוצו אותו זמן (בהנחה שנותנים להם אותו זמן לרוץ). מכיוון שאלגוריתם alpha-beta מבצע גיזום ענפים אז כל איטרציה עבור עומק נתון d של האלגוריתם שמבצע גיזום צפוייה (במקרה הממוצע) לקחת פחות זמן מאיטרציה של האלגוריתם שלא מבצע גיזום. לכן, האלגוריתם alpha-beta יצליח להשלים איטרציה בעלת עומק נמוך יותר מהאלגוריתם RB-mini-max שלא מבצע גיזום ולכן הוא עלול להוביל לבחירת מהלכים שונה, כי האלגוריתם שמצבע גיזום יתחשב בצמתים (כלומר במצבים) עמוקים יותר.

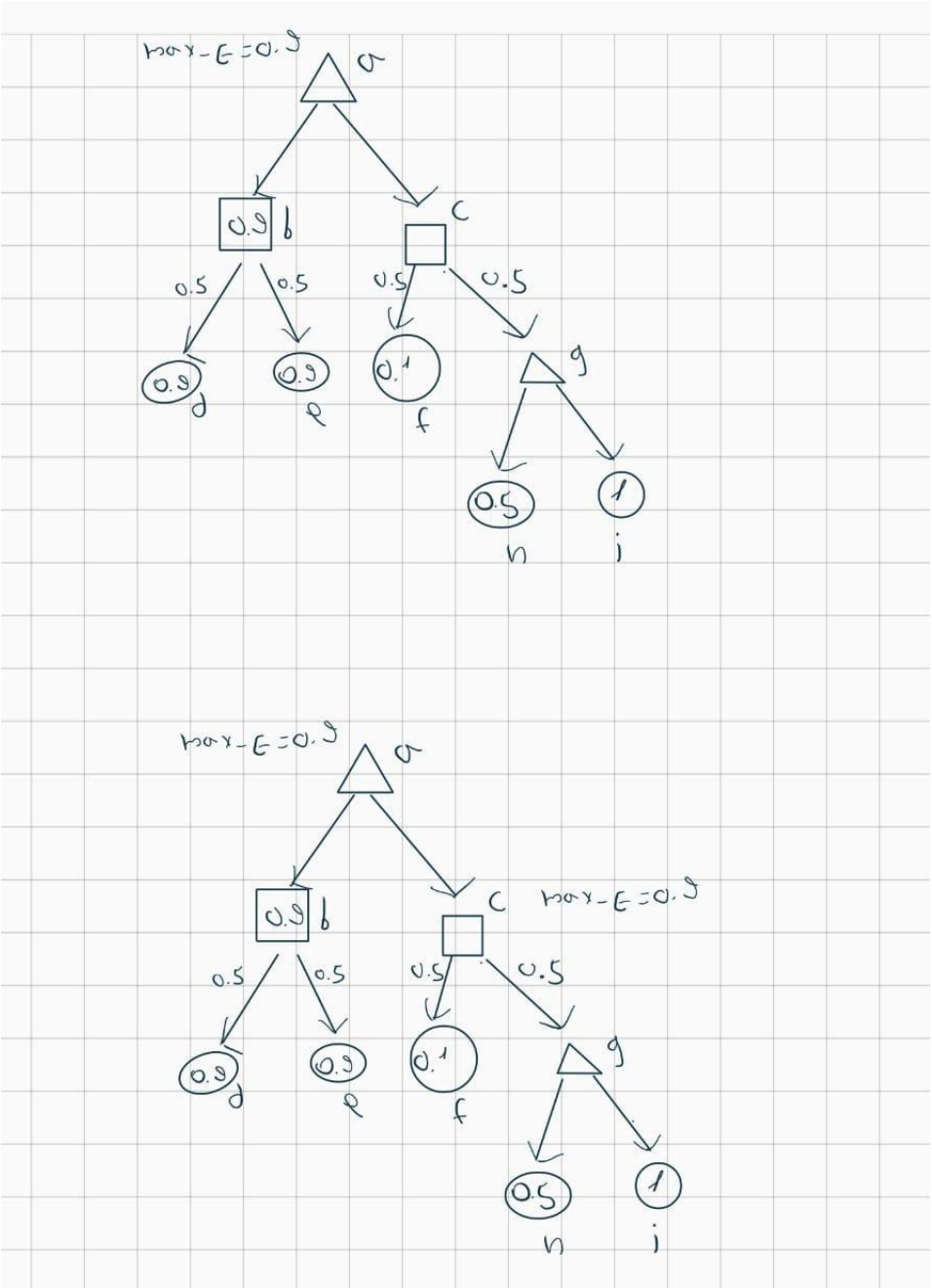
   אם הכוונה שנווה בין mini-max בלי הגבלת משאבים ו alpha-beta בלי הגבלת משאבים אז התשובה תהיה הפוכה. כלומר הסוכנים לא יתנהגו שונה מבחינת בחירת מהלכים אך כן יתנהגו שונה מבחינת הזמן. זה בגלל שכפי שלמדנו בקורס, אז גיזום אינו משפיע על בחירת המהלך האופטימלי אבל כן צפוי לקצר את זמן הריצה.

חלק ד

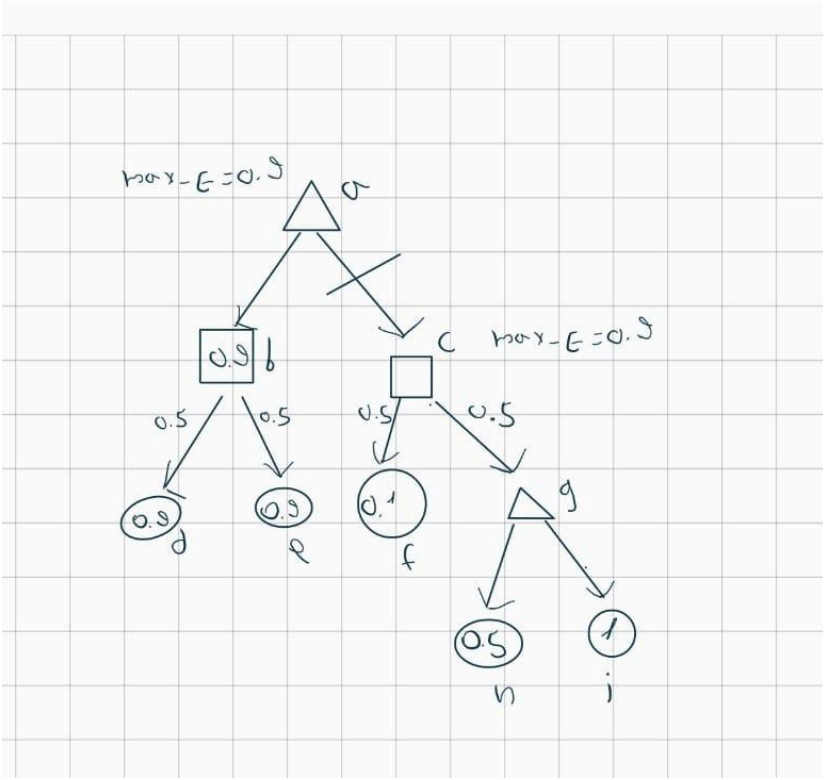
1. במקרה זה אז, ניתן הסתברות שווה לכל אחד מהמצבים העוקבים של מצב שבו תורו של היריב לשחק. כלומר עם במצב s, תור היריב לשחק ויש n מצבים עוקבים אפשרים, אז לכל מצב ניתן הסתברות של $\frac{1}{n}$.
2. נסביר עבור משחק של סוכן מקסימום שמשחק נגד סוכן הסתברותי. המעבר למקרים נוספים הוא פשוט. במקרה זה, נוכל לבצע גיזום במצבים הסתברותיים, אם אנחנו נדע שהם לא יכולים לספק שיפור בתוחלת. כלומר נפעפע למטה את התוחלת שמובטחת לסוכן מקסימום, ואם נגיע למצב שחקרנו אותו חלקית ולא משנה מה יהיו ערכי היוריסטיקה בהמשך החקירה, התוחלת המקסימלית שמצב זה יכול לקבל היא נמוכה (או שווה) לתוחלת המקסימלית שכבר מצאנו אז נוכל לגזום את תת עץ זה. נדגים:



נניח שעץ המצבים שלנו הוא כבשרטוט שלמעלה. כאשר צומת משולש זה צומת מקסימום, צומת מרובע זה צומת הסתברותי, צומת עיגול זה מצב סופי והמספרים על הקשתות זה ההסתברויות של צמתים התסברותיים ולצדי הצמתי כתבנו את שמות הצמתים (אותיות באנגלית). בתחילת ה Expctimax אז נאתחל את התוחתלת המקסימלית שמובטח לשורש a הוא מינוס אינסוף (כי זה צומת מקסימום) (נסמן ערך זה ב max_E) ונחלחל ערך זה למטה כאשר יורדים בעומק העץ.



לאחר שנחקור את הבן השמאלי (צומת b) אז העץ יראה כפי שנראה בעץ הראשון בשרטוט שלמעלה. ערך max_E אצל צומת a יתעדכן ל 0.9 כי מצאנו צומת שתוחלת הערך בה היא 0.9. ולאחר מכן נעבור לצומת c ונחלחל את הערך max_E למטה (כפי שרואים בעץ השני בשרטוט מעל). מצומת c נרד לצומת f ונראה שבהסתברות 0.5 אנחנו מקבלים ערך של 0.1 ולכן במקרה זה נדע שלא משנה איזה ערך נקבל מצומת g, (מכיוון שערך היוריסטי או הסופי המקסימלי הוא 1) אז בצומת c לא נוכל לקבל ערך גבוה יותר מ max_E ולכן, אין צורך לחקור את צומת g ונוכל לגזום את c כך:



3. בקוד.

חלק ה

1. **סעיף a.** במקרה זה, מקדם הסיעוף ישאר כפי שהיה לפני. כלומר מקדם הסיעוף יהיה 5. הסיבה לכך שמקדם הסיעוף הוא 5 ולא 7 (למרות שניתן לבצע 7 פעולות) היא שבכל מצב נתון, ניתן לבצע 5 פעולות לכל היותר כי בכל מצב ניתן לבצע לכל היותר 1 מהפעולות "הרם חבילה", "הורד חבילה" ו "טען סוללה" כי אנו מניחים שלא יכולה להיות חבילה או יעד של חבילה בנקודות טעינה וגם אם לרובוט יש חבילה אז הוא לא יכול להרים חבילה נוספת ואם אין לו חבילה אז הוא לא יכול לבצע הורדת חבילה. ולכן בכל מצב ניתן לבצע 4 פעולות הליכה ועוד לכל היותר פעולה אחת מתוך "הרם חבילה", "הורד חבילה" ו "טען סוללה" ולכן מקדם הסיעוף הוא 5 במשחק המקורי והוא לא ישתנה במשחק המעודכן כי לא הוספנו פעולות חדשות וגם לא הורדנו את מספר הפעולות המקסימלי שסוכן יכול לעשות בכל מצב. כנדרש.

   סעיף b. במקרה זה הגדלנו את מקדם הסיעוף ב 21. וזה מכיוון שהלוח הוא בגודל של 25 משבצות (5 על 5) ומכיוון שניתן להוסיף קוביה רק על משבצות ריקות, ויש לכל הפחות 4 משבצות תפוסות בכל מצב (כי תמיד יש על הלוח 2 יעדים ושתי עמדות טעינה ואולי 0 חבילות כי הרובוטים הרימו אותם) כלומר יש לכל היותר 21 משבצות פנויות ולכן הגדלנו את מספר הפעולות האפשריות בכל מצב בלכל היותר 21 ולכן כעת, מקדם הסיעוף יהיה 25 (21+5). כנדרש.

2. **סעיף a.** הבהרה: בשעת קבלה הובהר שהכוונה להתייחס לזמן הריצה במידה והאלגוריתמים לא היו anytime.
   אין אלגוריתם שמימשנו בסעיפים הקודמים שעבור שני הגרסאות יקח לו זמן דומה להחזיר צעד. דבר זה נובע מכיוון שסיבוכיות הזמן של כל האלגוריתמים שמימשנו בסעיפים הקודמים היא:

$$O(b^m)$$

כאשר b זה מקדם הסיעוף ו m זה העומק אליו אנחנו מגיעים ולכן עבור הגרסא החדשה של המשחק מכיוון שהגדלנו את מקדם הסיעוף אז נקבל:

$$O(25^m) > O(5^m)$$

כלומר הגדלנו משמעותית את סיבוכיות הזמן. נציין שגם עבור אלגוריתם alpha-beta, אז במקרה האופטימלי נקבל שסיבוכיות זמן הריצה הופכת להיות

$$O(b^{\frac{m}{2}})$$

ואז גם במקרה זה נקבל:

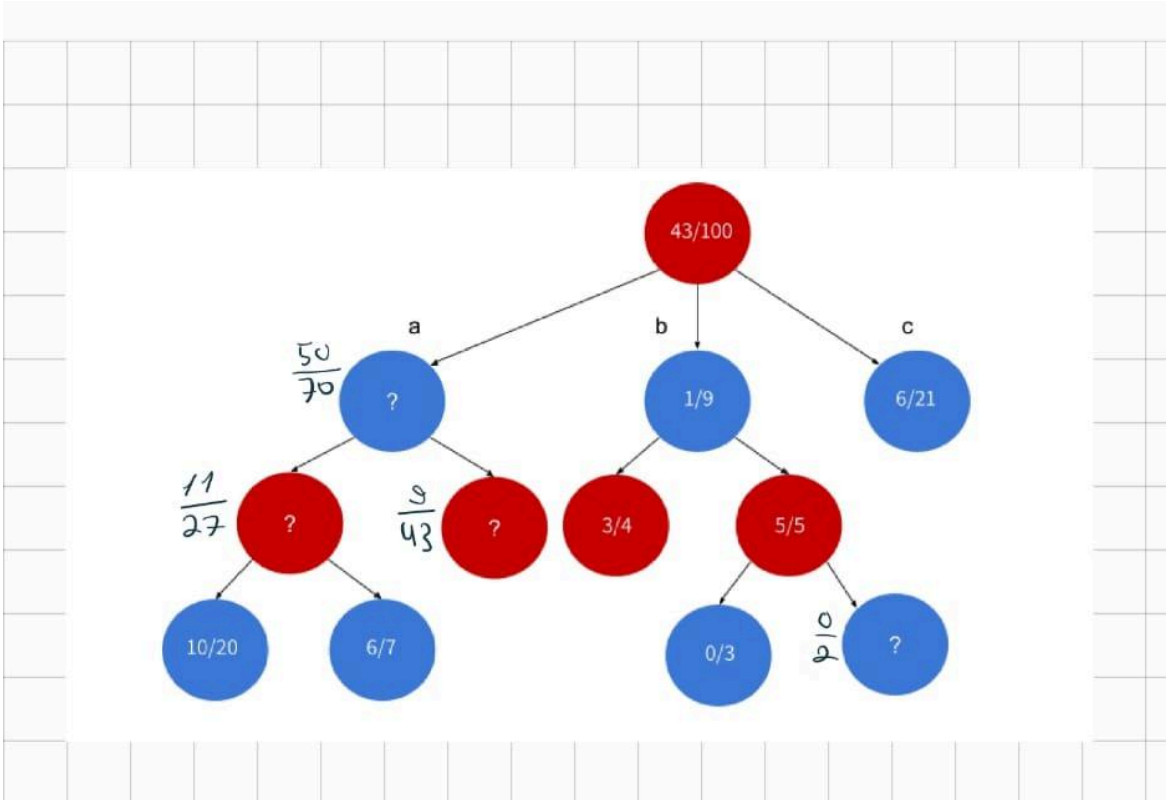
$$O(25^{\frac{m}{2}}) > O(5^{\frac{m}{2}})$$

כנדרש.

סעיף b. נוכל להשתמש באלגוריתם monte-carlo שלמדנו בהרצאות. הסיבה שהוא טוב להתמודדות עם השינוי בסביבה היא שאלגוריתם זה לא מנסה לחקור את כל העץ (או את כל הצמתים עד לעומק מסויים) ולכן הוא פחות רגיש למקדם הסיעוף. באלגוריתם זה, כדי להחליט מה יהיה הצעד הבא של השחקן שלנו, נבצע מספר סימולציות של המשחק החל מהמצב הנוכחי עד למצב סופי, ולבסוף נבחר בפעולה שהחל ממנה הצלחנו לנצח הכי הרבה פעמים. כלומר, סבוכיות הזמן של האלגוריתם היא לא אקספוננציאלית בעומק הפתרון כי היא לא מפתחת את כל העץ אלא רק מספר סימולציות של המשחק. ולכן אלגוריתם זה מתמדד היטב עם השינוי לסביבה ולכן נבחר בו.

חלק ו

1. הערכים הדרושים מופיעים לצדי הצמתים:



2. נחשב את $UCB1(s)$ לכל אחד מהבנים של השורש ונבחר בבן עם הערך המקסימלי כפי שלמדנו בהרצאה לפי ההגדרה מתקיים

$$UCB1(s) = \frac{U(s)}{N(s)} + C \times \sqrt{\frac{ln(N(s.parent))}{N(s)}}$$

נציב את הצמתים a,b,c בנוסחא ונקבל:

$$UCB1(a) = \frac{50}{70} + \sqrt{2} \times \sqrt{\frac{ln100}{70}} = 1.077$$

$$UCB1(b) = \frac{1}{9} + \sqrt{2} \times \sqrt{\frac{ln100}{9}} = 1.122$$

$$UCB1(c) = \frac{6}{21} + \sqrt{2} \times \sqrt{\frac{ln100}{21}} = 0.947$$

לכן הצומת הבא שיבחר זה צאצא של צומת b. כנדרש.

3. ההנחה בסעיף זה גוררת שערכי $UCB1$ לאחר n סימולציות נוספות יראו כך (כי נתון שרק הכחול מנצח):

$$UCB1(a) = \frac{50}{70} + \sqrt{2} \times \sqrt{\frac{ln(100+n)}{70}}$$
$$UCB1(b) = \frac{1}{9+n} + \sqrt{2} \times \sqrt{\frac{ln(100+n)}{9+n}}$$
$$UCB1(c) = \frac{6}{21} + \sqrt{2} \times \sqrt{\frac{ln(100+n)}{21}}$$

ולכן נחפש את ה-n המינימלי שעבורו יתקיים $UCB1(b) < UCB1(a)$ או $UCB1(b) < UCB1(c)$. עבור אי השוויון הראשון:

$$\frac{50}{70} + \sqrt{2} \times \sqrt{\frac{ln(100+n)}{70}} > \frac{1}{9+n} + \sqrt{2} \times \sqrt{\frac{ln(100+n)}{9+n}}$$

ולאחר הצבה של $n = 1$ נקבל שמתקיים אי השוויון כי:

$$UCB1(a) = \frac{50}{70} + \sqrt{2} \times \sqrt{\frac{ln(100+1)}{70}} 1 = 1.07$$
$$UCB1(b) = \frac{1}{9+1} + \sqrt{2} \times \sqrt{\frac{ln(100+1)}{9+1}} = 1.06$$

כלומר התשובה היא שאחרי סימולציה 1 של ניצחון כחול אז יבחר צאצא אחר של השורש. כנדרש.

שאלה פתוחה

1. הבעיה שאלגוריתם מונטה קרלו מהתרגולים מנסה לפתור היא שבבעיות מסוימות, הסוכן לא יודע את המצב הנוכחי באופן מלא אלא יודע רק מידע חלקי על המצב. דוגמא לכך יכולה להיות משחק קלפים שבו הסוכן יודע רק את הקלפים שיש לו ולא יודע את הקלפים שיש ליריב. האלגוריתם פותר את הבעיה בכך שהוא משלים את המצב (החלקי) הנוכחי ל k מצבים מלאים אפשריים (כלומר שעקביים עם הידע שיש לסוכן) בצורה רנדומלית ועבור כל פעולה אפשרית שיכול לבצע הסוכן, אז הוא מפעיל את הפעולה על כל אחד מ k המצבים האשפרים ומריץ rb-minimax מהמצב שהתקבל. לבסוף, האלגוריתם מחזיר את הפעולה שעבורה הערך הממוצע על פני k המצבים המלאים (הערך התקבל מ rb-minimax) הגבוה ביותר. כלומר האלגוריתם פותר את הבעיה בכך שהוא "מנחש" k מצבים מלאים שעקביים עם הידע שיש לו ומנסה להחזיר את הפעולה שממוצע על k המצבים תהיה הכי טובה.
2. הקלט K זה מספר המצבים המלאים שהאלגוריתם ישלים אליהם את המצב החלקי שנתון לו (ועליהם יריץ rb-minimax). אם נבחר K קטן מדי, אז האלגוריתם ירוץ מהר אבל הממוצע של הערך שיחזיר rb-minimax על פני K המצבים עבור פעולה a לא ייצג את הטיב האמיתי של הפעולה a כי יכול להיות שבמקרה השלמנו ל K מצבים לא מייצגים. אם נבחר K גדול מדי אז לאלגוריתם יקח הרבה מאוד זמן לבחור פעולה כי הוא יצטרך להריץ rb-minimax כמספר הפעולות האפשריים כפול K אבל אמינות הפעולה שהוא יחזיר תהיה גבוהה. ולכן השיקולים הם זמן ריצה גדול והחזרת תוצאה אמינה לעומת זמן ריצה יותר קצר אבל החזרת פעולה פחות בטוחה.
3. עבור ערך K ששווה ל $|S\_complete|$ אז נמדל את האלגוריתם מונטה קרלו בעזרת אלגוריתם expctimax בכך נשתמש באלגוריתם expectimax על המשחק שיהיה יהיה שקול למשחק המקורי פרט לשינוי הבא: כאשר תורו של הסוכן שלנו והוא נמצא במצב ידע חלקי, אז פעולה חוקית מ s תוביל אותנו למצב הסתברותי s' (כלומר לתור של סוכן הסתברותי) ובמצב זה, הפעולות האפשריות הן השלמת המצב החלקי למצב עם מידע מלא (שעקבי עם המצב החלקי) והפעלת הפעולה a על מצב זה. על המשחק החלופי שהגדרנו, ניתן להריץ את אלגוריתם expectimax והרצה שכזו תהיה שקולה להרצה של אלגוריתם מונטה קרלו על המשחק המקורי. הסיבה לכך היא שבאלגוריתם מונטה קרלו המקורי עם הערך K שנתון בשאלה אז, האלגוריתם משלים את מצב הידע החלקי לכל אחד מהמצבים העקביים ולאחר מכן הפעלת כל הפעולות על כל המצבים המלאים והחזרת הפעולה שעבורה הממוצע rb-minimax על כל המצבים היא הגבוהה ביותר. זה בדיוק מה שאנחנו עושים גם במשחק החלופי שהצענו שעליו נריץ את אלגוריתם expctimax. במשחק

החלופי שהצענו, כאשר הסוכן שלנו יגיע לצומת עם מידע חלקי אז הוא יבחר פעולה כלשהי a ומשם יגיע למצב הסתברותי s' ששם נבדוק את כל ההשלמות האפשריות של המצב והפעלת הפעולה a עליהם ו s' יחזיר את הערך הממוצע שאלגוריתם expectimax נותן על כל ההשלמות האפשריות. נחזור על התהליך לכל הפעולות האפשריות על מצב s (כי אלגוריתם expectimax בודק את כל הילדים של המצב כאשר מתעלים מגיזומים) ולבסוף מהמצב s נבחר בפעולה שתיתן לנו את הממוצע הטוב על פני כל ההשלמות האפשריות. כלומר עשינו בדיוק את אותו הדבר שעושה אלגוריתם מונטה קרלו ובכך הראינו את השקילות הדרושה (האלגוריתמים יבצעו את אותם החישובים יחזירו את אותו פלט).

4. גרסת anytime על אלגוריתם מונטה קרלו יכולה להיות גרסה שמריצה את אלגוריתם מונטה קרלו על ערכים הולכים וגדלים של K עד שנגמר הזמן. כלומר הפוסדו קוד יראה כך:

```
func monte_carlo_anytime(partial_state, agent, d, time):
    k = 1
    best_action = random.succ(partial_state)
    while (time > 0):
        best_action = MonteCarloPartialInformation(partial_state,
            agent, d, k)
        k+=1
    return best_action
```

כלומר, האלגוריתם יקבל מצב מידע חלקי, סוכן, d (העומק שיעבור ל rb-minimax) וכמות מסויימת של זמן לרוץ. כל עוד נשאר זמן, אז האלגוריתם יקרא ל מונטה קרלו עם ערכים הולכים ועולים של k וכשיגמר הזמן אז הוא יחזיר את הפעולה האחרונה שמונטה קרלו החזיר. הפתרון מתיישב עם עקרון אלגוריתמי ה- anytime מכיוון שכפי שהסברנו בסעיף 2, אז עבור k נמוך, אז נמצא פתרון מאוד מהר אבל איכות הפתרון מוטלת בספק, ועבור וככל שנגדיל את k אז יקח לנו יותר זמן למצוא פתרון אבל איכות הפתרון תלך ותגדל, וזה בדיוק העקרון שעומד מאחורי אלגוריתמי anytime. שככל שהאלגוריתם ירוץ יותר זמן כך הוא ישפר את הפתרון שהוא מוצא. כנדרש.