מבוא לבינה מלאכותית
תרגיל בית 2 – חלק יבש

<u>מגישים:</u>
רוני נודלמן – 209120112
רן ברשינסקי – 208387324

1. היתרונות של היוריסטיקה:
   א. היוריסטיקה מבטאת את הפוטנציאל של השחקן לעומת היריב: ככל שמספר הטחנות הכמעט-שלמות של השחקן גבוה יותר לעומת מספר הטחנות הכמעט-שלמות של היריב, כך גדל הפוטנציאל של השחקן להשלים את הטחנות האלו ובכך להסיר חיילים של היריב מהלוח.
   ב. היורסטיקה עוזרת לשחקן להימנע ממצבים שבהם החיילים שלו לא יוכלו לזוז. כל עוד יש טחנות כמעט-שלמות, מובטח כי החיילים של השחקן יוכלו לזוז למקום הריק באותה טחנה כמעט-שלמה.

   החסרונות של היוריסטיקה:
   א. היוריסטיקה "מעודדת" את השחקן להמשיך וליצור טחנות כמעט-שלמות ודווקא מונעת ממנו להשלים טחנות אלו, כיוון שאז הערך היוריסטי יקטן.
   ב. היוריסטיקה לא מביאה לידי ביטוי את מספר הטחנות הכמעט-שלמות שאכן ניתן להשלימן לטחנות שלמות. לדוגמה, כאשר לשחקן כלשהו יש טחנה כמעט-שלמה בשלב השני של המשחק, אך אין באפשרותו להשלימה כיוון שלא ניתן להזיז חיילים לכיוון התא הריק בטחנה זו.

2. נגדיר את הפונקציות הבאות:

$$h_1(s) = 4 \cdot \ number\ of\ player_1\ possible\ moves - 5 \cdot \ number\ of\ player_2\ possible\ moves$$
$$h_2(s) = 5 \cdot number\ of\ player_1\ potential\ mills - 4 \cdot number\ of\ player_2\ potential\ mills$$
$$h_3(s) = 4 \cdot \ number\ of\ player_1\ soldiers - 5 \cdot \ number\ of\ player_2\ soldiers$$
$$h_4(s) = 5 \cdot number\ of\ player_1\ mills - 4 \cdot number\ of\ player_2\ mills$$
$$h_5(s) = 5 \cdot number\ of\ player_1\ almost\ mills - 4 \cdot number\ of\ player_2\ almost\ mills$$
$$h_6(s) = 5 \cdot player_1\ killed - 4 \cdot player_2\ killed$$
$$h_7(s) = 5 \cdot player_1\ num\ of\ kills - 4 \cdot player_2\ num\ of\ kills$$

נגדיר את היוריסטיקה:
$$h(s) = 2 \cdot h_1(s) + 30 \cdot h_2(s) + 50 \cdot h_3(s) + 30 \cdot h_4(s) + 15 \cdot h_5(s) + 30 \cdot h_6(s) + 30 \cdot h_7(s)$$

*נשים לב כי אם השחקן שלנו הוא השחקן השני, אז היוריסטיקה תקבע להיות $-h(s)$.
נסביר את המוטיבציה לבחירת פונקציות אלו:
$h_1(s)$ – מטרה נוספת של המשחק הינה לצמצם את כמות המהלכים האפשריים של היריב עד לכדי 0, ומצד שני לשמור על מספר גבוה של צעדים אפשריים של השחקן. לכן נרצה למקסם גם את הפרש הערכים האלה.
$h_2(s)$ – שלשה רצופה תיקרא *טחנה פוטנציאלית* אם היא טחנה כמעט-שלמה וגם קיימת משבצת שסמוכה למשבצת הריקה בטחנה זו ובה קיים חייל של השחקן. *טחנה פוטנציאלית* מבטאת פוטנציאל גבוה ליצירת טחנה של השחקן בצעד הבא ולכן תרצה למקסם ערך זה, ומצד שני להוריד את הערך הזה עבור היריב.
$h_3(s)$ – אחת ממטרות המשחק הינה לצמצם את כמות חיילי היריב ומצד שני השחקן שואף להגדיל את מספר החיילים שברשותו. לכן נרצה למקסם הפרש ערכים אלו.

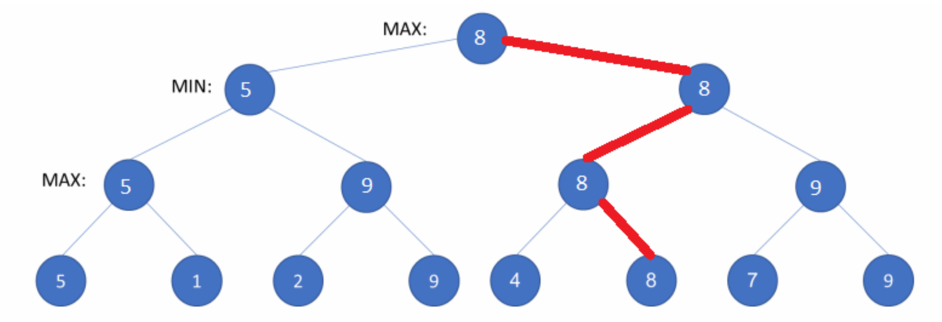
$h_4(s)$ – אחת ממטרות הביניים של המשחק היא ליצור טחנות וזאת בכדי להסיר חיילים של היריב, וכך לצמצם את מספר חיילי היריב שעל הלוח עד לכדי ניצחון. לכן נרצה להגדיל את מספר הטחנות של המשחקן ולצמצם את מספר הטחנות של היריב, וזאת על ידי מיקסום ההפרש של ערכים אלו.
$h_5(s)$ – היתרונות עבור פונקציה זו צוינו לעיל בסעיף 1.
$h_6(s)$ – אלה הם דגלים המציינים האם השחקנים הרגו חייל בתור האחרון שלהם. כמובן שאחת מהמטרות היא להרוג חיילים של היריב, ולכן נרצה למקסם הפרש זה.
$h_7(s)$ – זהו הפרש החיילים שכל אחד מהשחקנים הצליח להרוג. בדומה לפונקציות האחרות ולמטרת המשחק, נרצה להגדיל את ההפרש הזה ובכך להגדיל את הפרש החיילים של שני השחקנים לטובת השחקן שלנו.
נסביר את המוטיבציה לבחירת המשקלים:
במשחקים רבים שבהם משחקים 2 שחקנים, יש נטייה לשחקן הראשון להיות שחקן יותר התקפי ואילו השחקן השני נוטה להיות יותר הגנתי. כלומר השחקן הראשון מנסה להיות יותר אגריסיבי, ועל כן מעדיף להביס חיילים של היריב מאשר לשמור על החיילים שלו, ודווקא השחקן השני מנסה להעדיף לשמור על החיילים שלו במקום לתקוף את חיילי היריב. כמובן שנטיות אלו אינן אבסולוטיות, והתפקידים עלולים להתהפך במהלך המשחק, אך בכל זאת החלטנו לקחת נקודה זו בחשבון. על כן, הגדרנו כי ערכם של החיילים של השחקן הראשון נמוכים יותר מערכם של החיילים של השחקן השני (ע"י המשקלים 4,5), ובכך נקבל התנהגת שבה השחקן הראשון מעדיף להביס חיילים של השחקן השני, ואילו השחקן השני מעדיף לשמור על החיילים שלו בחיים. באופן זה, לכל הפונקציות h_i הוספנו את המקדמים 4,5 שמשקפים התנהגות זאת.
בנוסף, שאר המשקלים ... ,2,30,50,30,15 נבחרו על מנת לשקף את החשיבות ואת הכמות של כל אחד מהערכים היוריסטיים. לדוגמה, בשלב ממוצע במשחק, קיימים הרבה צעדים אופציונליים עבור כל אחד מהחיילים, אך מספר הטחנות עלול להיות נמוך בהשוואה לערך זה. לכן, על ידי מתן משקלים אלו, איזנו בין הערכים הללו. בנוסף, ערכן של טחנות שלמות גבוה יותר מערכן של טחנות כמעט-שלמות ועל כן נתנו משקל יותר גבוה לטחנות השלמות.

.3

א. היתרון בהוספת גיזום אלפא-בטא לאלגוריתם מינימקס הוא צמצום הזמן שלוקח לחשב את ערך שורש העץ, בהשוואה להרצת אלגוריתם מינימקס ללא גיזום, וזאת ע"י גיזום ענפים שחישוב ערכי המינימקס עבורם לא מועילים לחישוב ערכי המינימקס של האבות שלהם.
יתרון זה מושג באופן הבא:
עבור צומת max כלשהו, נניח כי ערכו של הבן הראשון שלו חושב, והינו x. כעת, כאשר נחשב את ערכי הבנים של הבן השני של הצומת הנ"ל, אם נקבל ערך קטן או שווה ל-x אז נוכל לדלג על החישוב של הבנים הבאים של הבן השני. זאת כיוון שהבן השני הינו צומת min ולכן ערכו יהיה לכל היותר x, ולכן אין טעם להמשיך לפתח את שאר הבנים שלו, שהרי צומת max בוחר את הבן עם הערך המקסימלי. כלומר, כאשר הצומת max יבחר את הבן עם הערך המקסימלי, לא ייתכן שהוא יבחר את הצומת השני על פני הצומת הראשון. באופן זהה, נוכל לבצע את תהליך זה עבור שאר הבנים של הצומת max.
באופן אנלוגי, נוכל לעשות את אותו התהליך עבור צומת min.

ב. כפי שראינו בהרצאה, על מנת לקבל גיזום מקסימלי נמיין את הצמתים באופן הבא:
עבור צמתי max, הערך המקסימלי מבין ערכי המינימקס של הבנים חייב להיות בבן השמאלי.
עבור צמתי min, הערך המינימלי מבין ערכי המינימקס של הבנים חייב להיות בבן השמאלי.

ג. אף עלה לא יגזם ע"י אלגוריתם אלפא-בטא בעץ הנתון.
המסלול האופטימלי להמשיך המשחק מסומן במסלול האדום בציור.



4. לא בהכרח יצרנו סוכן שמשחק עם אסטרטגיה אופטימלית. דוגמא לכך היא היורסיטקה של שאלה 1, שהיא יורסיטקה מטעה. מכיוון שלפי היורסטיקה הזאת, הסוכן יעדיף ליצור כמה שיותר טחנות כמעט שלמות, ולא ליצור טחנות שלמות, אך בשביל לנצח, צריך ליצור טחנות, כדי להסיר את החיילים של היריב. כלומר היורסטיקה לא בהכרח מבטיחה אסטרטגיה אופטימלית.

5.

1. נגדיר את פונקציית התועלת באופן הבא כאשר k מציין את אינדקס השחקן:

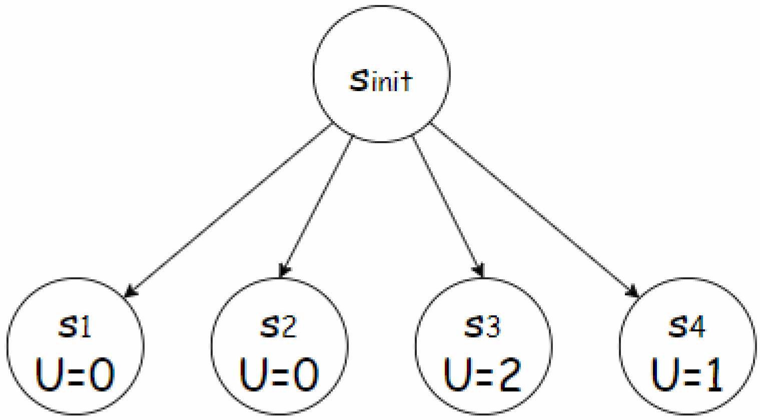
$$U(s_1,k)=\begin{cases}5 & k=1\\ -1 & k=2\end{cases}$$
$$U(s_2,k)=\begin{cases}-1 & k=1\\ 5 & k=2\end{cases}$$
$$U(s_3,k)=\begin{cases}5 & k=1\\ -1 & k=2\end{cases}$$
$$U(s_4,k)=\begin{cases}-1 & k=1\\ 5 & k=2\end{cases}$$

נניח כי המצבים s_1, s_3 אלו מצבים שבהם השחקן הראשון מנצח, ואילו המצבים s_2, s_4 אלו מצבים שבהם השחקן השני מנצח.
אכן, פונקציית תועלת זאת לא מגדירה משחק סכום אפס. למרות זאת, עדיין מתקיים כי כל אחד מהשחקנים ינסה למקסם את התועלת שלו ובכך להגיע למצב שבו הוא מנצח.

2. נגדיר פונקציית תועלת באופן הבא כאשר k מציין את אינדקס השחקן:

$$U(s_1,k) = \begin{cases} 0 & k=1 \\ 0 & k=2 \end{cases}$$
$$U(s_2,k) = \begin{cases} 0 & k=1 \\ 0 & k=2 \end{cases}$$
$$U(s_3,k) = \begin{cases} 2 & k=1 \\ 5 & k=2 \end{cases}$$
$$U(s_4,k) = \begin{cases} 1 & k=1 \\ 0 & k=2 \end{cases}$$

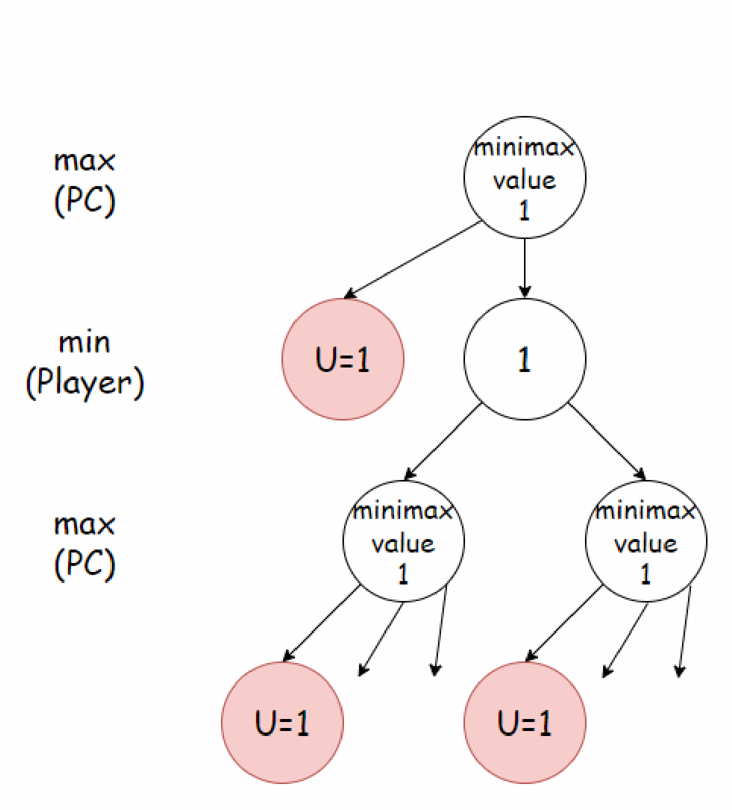
נראה דוגמה עבור עץ חיפוש עם אלגוריתם $minimax$, שבו השחקן הראשון מתחיל לשחק:



במקרה זה, ערך המינימקס של השורש יהיה 2 ועל כן השחקן הראשון יבחר לבצע את הפעולה שתגרום לו להגיע למצב s_3. אולם, מבחינת שקלול הניקוד עבור השחקנים, השחקן הראשון היה יכול לבחור במצב s_4 וכך להשיג יתרון בניקוד על פני היריב.

.6

א. ייתכן מצב במשחק שבו המחשב בעל יתרון גדול על פני השחקן, כך שהניצחון של המחשב מובטח כמעט בכל צעד שיעשה. ובמצב זה, המחשב יכול לבצע צעד אחד ולנצח מיד, או שלבצע צעד אחר שלא יגרום לו לנצח מיד אך עדיין יבטיח ניצחון בשלב מאוחר יותר במשחק. לדוגמה, ניתן לראות זאת ע"י עץ המינימקס הבא:



*המצבים האדומים בעץ הם מצבים סופיים שבהם המחשב מנצח.
בדוגמה זאת, המחשב יוכל לבחור אם לנצח תוך צעד אחד (לבחור בענף השמאלי), או לבצע צעד אחר (לבחור בענף הימני) ולנצח בכל מקרה בצעד הבא.

ב. בסיום הריצה של האלגוריתם, במקום לבחור בן רנדומלי עם ערך $minimax$ שזהה לערך שיצא לשורש, נבחר בן שמייצג מצב סופי עם הערך הזה, אם קיים. וכך נוודא שאם ניתן לסיים את המשחק בצעד אחד, השחקן בוודאות יבחר בצעד זה.

.7

א. הערכים של הצמתים הם:

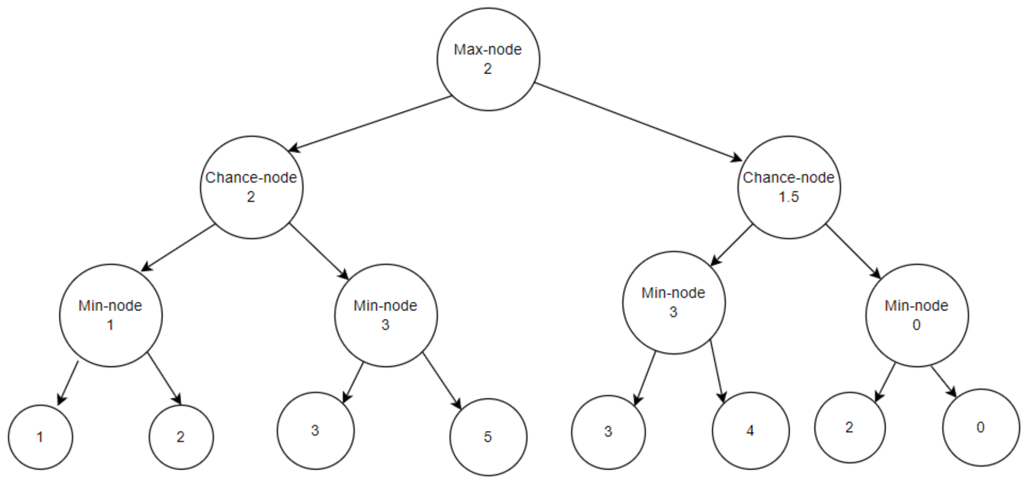
$$Expectimax(B) = 5 \cdot 0.3 + 1 \cdot 0.7 = 2.2$$
$$Expectimax(C) = 2 \cdot 0.4 + 3 \cdot 0.2 + 9 \cdot 0.4 = 5$$
$$Expectimax(D) = 4 \cdot 0.1 + 0.9 \cdot 7 = 6.7$$

ב. הפעולה שתיבחר היא $a3$. זאת מכיוון שצומת max בוחר את הבן עם הערך $Expectimax$ המקסימלי.

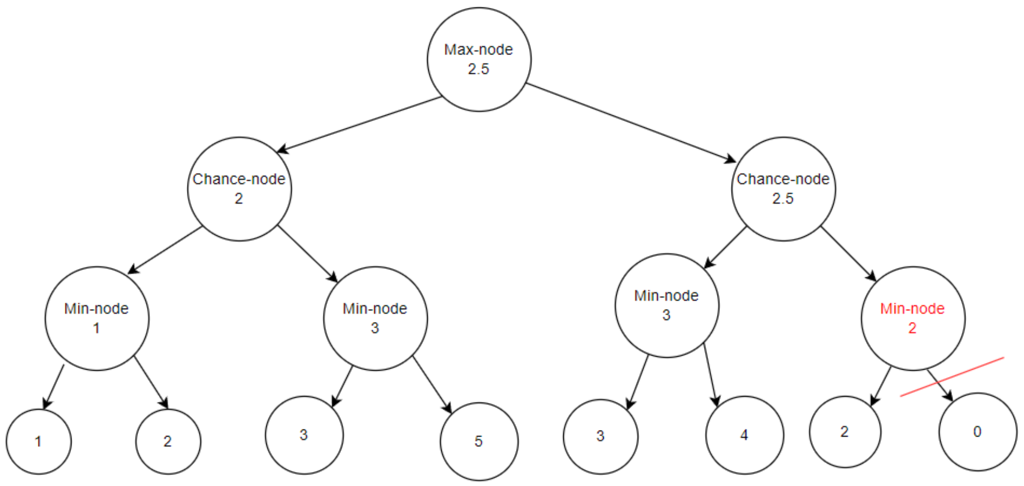
ג. לא נוכל לגזום ב- $expectimax$.
נראה דוגמאות לעצים, שאכן מראים כי גזימה זו בלתי אפשרית.

לשם הדוגמאות, נניח, כי בצמתי $chance$ בשני העצים שנראה, האחוזים עבור כל בן הינם 50 אחוז.

העץ בלי גיזום:



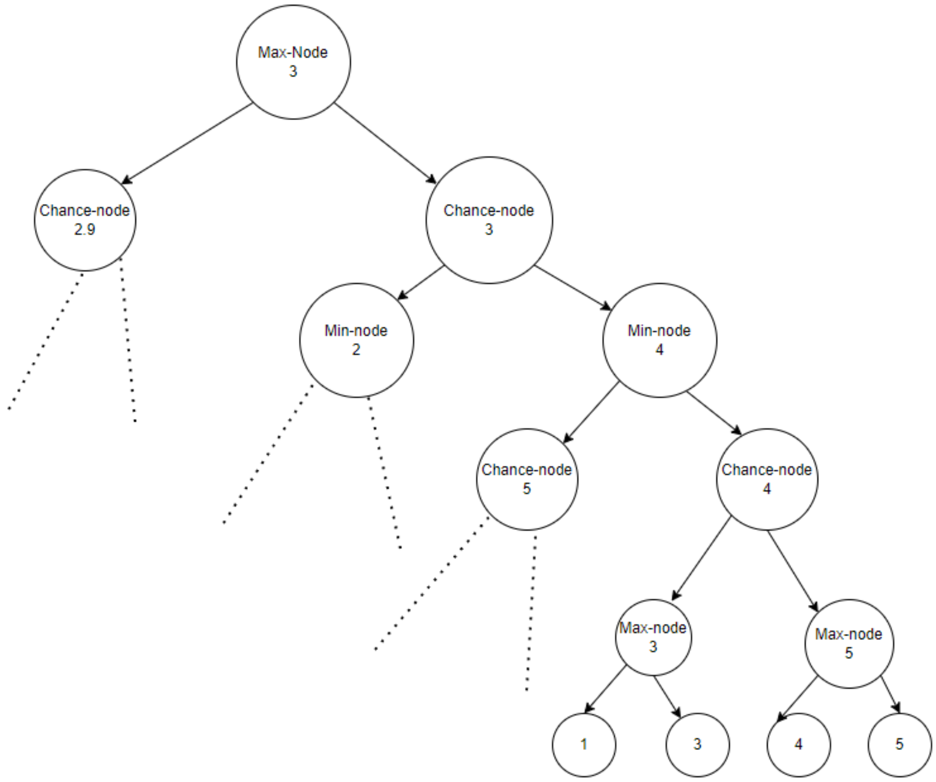
העץ עם גיזום:



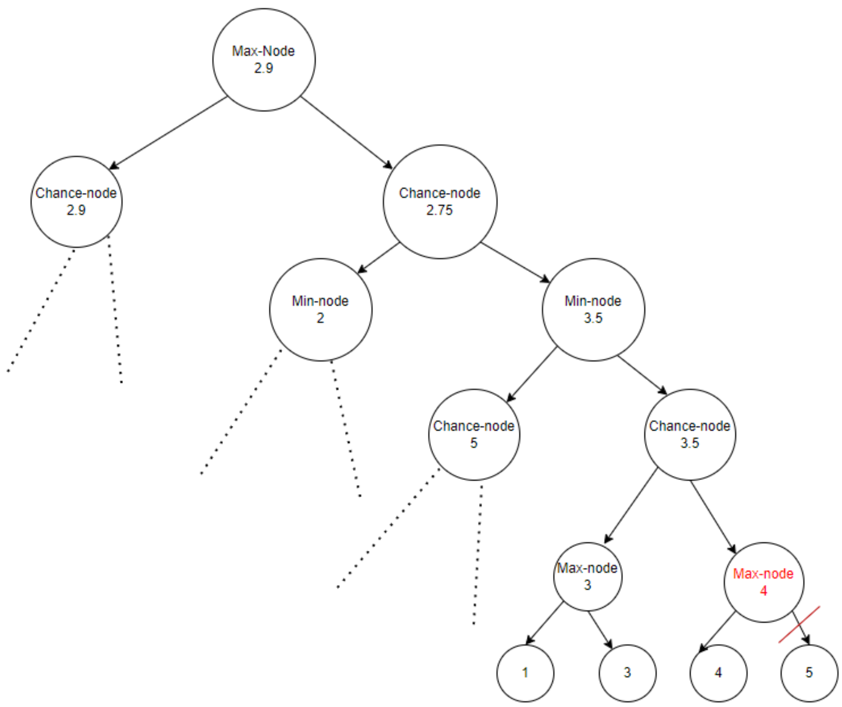
נשים לב שאם נרצה להשתמש בגיזום של $Minimax$, אז כשנגיע לצומת מינימום שצבענו אותו באדום, אז יתקיים $\alpha = 3$, וזאת מכיוון שהאח שלו קיבל את הערך 3. לפי האלגוריתם, בודקים אם הערך שהצומת האדום קיבל קטן או שווה ל-α. אם כן, אז גוזמים את שאר הבנים שלו. אחרי שנחקור את הבן השמאלי, נקבל את הערך 2, שקטן מ-3, ועל כן נגזום את הבן הימני. לכן ערך הצומת האדום הינו 2, דבר הישפיע על ערך האבות הקדמונים שלו. לכן, נקבל כי לבסוף הערך של השורש ישתנה ויהיה 2.5 במקום 2. באופן זה, ערך השורש השתנה וגם הבן האופטימלי השתנה.
לכן לא ניתן לגזום בנים של צמתי מינימום.

כעת נראה דוגמא לכך שאי אפשר לגזום צמתי מקסימום.

עץ בלי גיזום:



עץ עם גיזום:



נשים לב שאם נרצה להשתמש בגיזום של $Minimax$, אז כשנגיע לצומת מקסימום שצבענו באדום, אז יתקיים $\beta = 3$, מכיוון שהאח שלו קיבל את הערך 3. לפי האלגוריתם, בודקים אם הערך שהצומת האדום קיבל גדול או שווה ל-β. אם כן, גוזמים את שאר הבנים שלו. אחרי שנחקור את הבן השמאלי, נקבל את הערך 4, שגדול מ-3, ועל כן נגזום את הבן הימני. לכן ערך הצומת האדום הינו 4, דבר הישפיע על ערך האבות הקדמונים שלו. לכן, נקבל כי לבסוף הערך של השורש ישתנה ויהיה 2.9 במקום 3. באופן זה, ערך השורש השתנה וגם הבן האופטימלי השתנה.
לכן לא ניתן לגזום בנים של צמתי מקסימום.

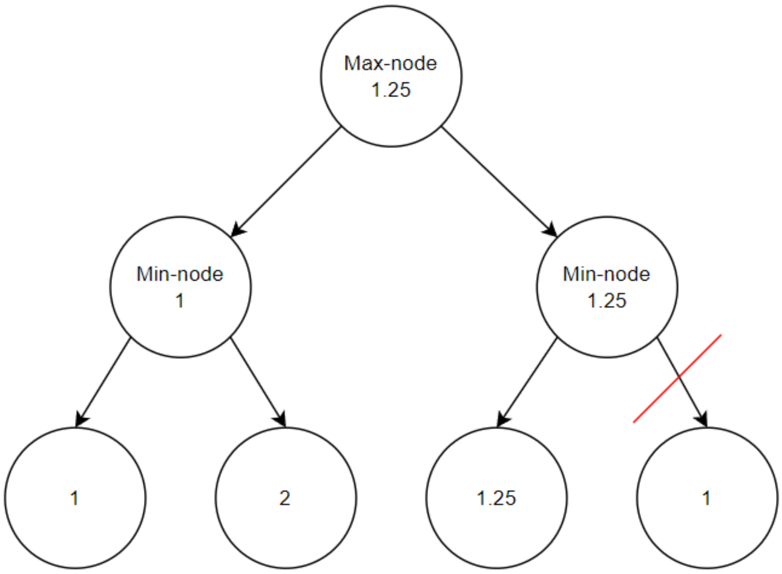
לכן לא ניתן לגזום בעצי $expectimax$ באותו אופן שבו אנחנו רוצים לגזום באלגוריתם אלפא בטא בלי לפגוע בנכונות האלגוריתם, שהרי שורש העץ לא נשאר בהכרח זהה ועל בחירת הבנים האופטימליים עלולה להיות שגויה.

.8

א. פסודו קוד של אלגוריתם אלפא-בטא עם השינוי:

```
alphaBeta(state, epsilon):
    return maxValue(state, -INFINITY, INFINITY, 0, epsilon)

maxValue(state, alpha, beta, depth, epsilon):
    if cutoffTestTest(state, depth)
        return utility(state)

    value = -INFINITY
    for successor in state.getSuccessors():
        value = max(value, minValue(successor, alpha, beta, depth+1, epsilon))
        if value >= beta - epsilon / (2 ^ (depth + 1)):
            return value
        alpha = max(alpha,value)

    return value

minValue(state, alpha, beta, depth, epsilon):
if cutoffTestTest(state, depth)
    return utility(state)

value = -INFINITY
for successor in state.getSuccessors():
    value = min(value, maxValue(successor, alpha, beta, depth + 1, epsilon))
    if value <= alpha + epislon / (2 ^ (depth + 1)):
        return value
    alpha = min(beta, value)

return value
```

ב. להלן דוגמה:



נגדיר $\varepsilon = 1$ במקרה זה.

נשים לב כי כאשר מגיעים לצומת הבן הימני של שורש העץ, יתקיים $\alpha = 1$.
לפי האלגוריתם החדש, אנחנו בודקים אם הבן השמאלי שלו קטן או שווה מ-

$$\alpha + \frac{epsilon}{2^2} = 1 + \frac{1}{4} = 1.25$$

מכיוון שזה המצב, אז גוזמים את העלה הימני שלו, ועל כן ערך הצומת יקבע להיות 1.25.
ולכן יוצא שערך השורש הוא 1.25.
נשים לב שאם היינו משתמשים באלגוריתם המקורי של אלפא בטא, היינו מקבלים שערך השורש הוא 1.
כלומר הראנו דוגמה שבה האלגוריתם החדש מחזיר ערך שונה עבור השורש בהשוואה לאלגוריתם המקורי.

.9

א. נראה כי אם נשתמש בפרוצדורה $rival\_move$ נוכל להגיע בזמן T לעומק $d2 = 2 \cdot d1$.
נשים לב שמספר הצמתים בעץ מינמקס רגיל הוא $O(b^{d1})$.
לעומת זאת, כעת, בכל שכבת min, כלומר שכבת היריב, לכל $node$ קיים רק בן אחד, שהוא הבן שאותו היריב יקח לפי פלט הפונקציה $rival\_move$.
אבל לכל צומת max קיימים כל b (בממוצע) הבנים שקיימים גם בעץ מינימקס רגיל.
לכן בכל שכבת max, כמות הצמתים שווה לכמות הצמתים בשכבת ה-min שמעליה, אבל לכל שכבת min, כמות הצמתים שבה גדולה פי b (בממוצע) מכמות הצמתים בשכבת ה-max שמעליה.
מכיוון שזמן הריצה הנתון לנו בעץ מינימקס רגיל (ללא שימוש בפונקציה $rival\_move$) זהה לזמן הריצה בעץ המינימקס שבו קיים השימוש בפונקציה $rival\_move$, אז נקבל כי עבור שני העצים, נספיק להגיע למספר שווה של צמתים, והוא:

$$2 \cdot 1 + 2 \cdot b + 2 \cdot b^2 + \cdots + 2 \cdot b^{\frac{d2}{2}} = O\left(b^{\frac{d2}{2}}\right) = O(b^{d1})$$

לכן מתקיים:

$$d1 = \frac{d2}{2}$$

כלומר מתקיים:

$$d2 = 2 \cdot d1$$

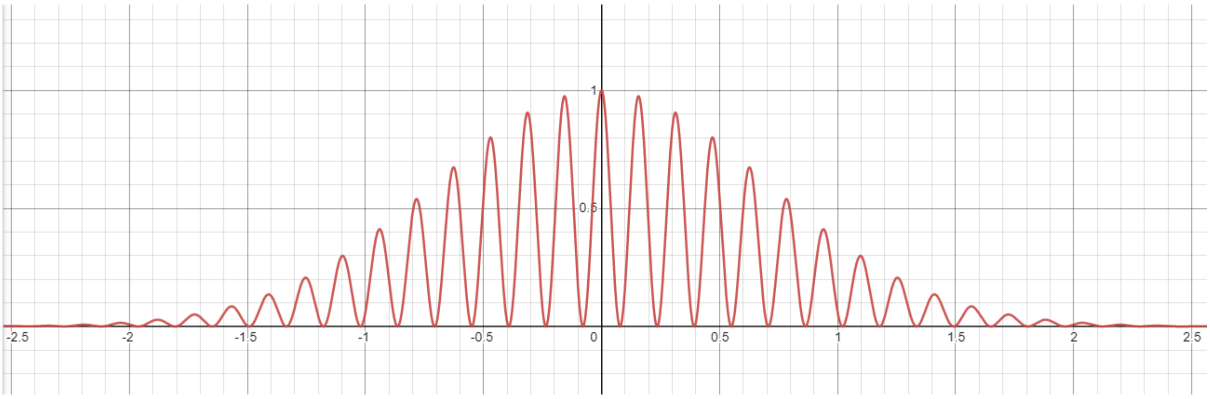
ב. הערך של מינימקס עם שימוש בפרוצדורה יהיה **גדול או שווה** לערך של מינימקס בלי שימוש בפרוצדורה כאשר שניהם מוגבלים לעומק d.
הסיבה לכך היא, שבמקרה של מינימקס בלי שימוש בפרוצדורה, אנחנו לוקחים בצמתי min תמיד את המצב הכי גרוע לנו, כלומר את הערך המינימלי. אבל, במקרה של מינימקס עם שימוש בפרוצדורה, יכול להיות שמצד אחד, יבחר המצב הכי גרוע עבורנו, ומצד שני, יבחר המצב הכי טוב עבורנו. לכן בסופו של דבר, ערך כל צומת min יהיה גדול או שווה לערך אותו הצומת ללא השימוש בפרוצדורה. לכן, כיוון ששאר הצמתים הם צמתי max אז ממילא נלקח עבורם הערך המקסימלי מבין הבנים שלהם. לכן נקבל כי כאשר נשתמש בפרוצדורה, ערך השורש יכול רק לגדול בהשוואה לערכו ללא השימוש בפרוצדורה.

10.

א. לא, האופטימום הגלובלי לא בהכרח שווה ל-5.8.
מכיוון שמרחב החיפוש עצום (בגודל 10^{12}), והסטודנט הפעיל את האלגוריתם רק ב-1000 נקודות שונות, ייתכן כי קיים אופטימום גבוה יותר באיזור כלשהו שתחומו קטן ביחס לגודל מרחב החיפוש, ועל כן דרושות הרבה הרצות של האלגוריתם על מנת להגדיל את הסיכוי ולקבל את האופטימום. כלומר ייתכן כי קיימת קבוצה קטנה של נקודות במרחב החיפוש שאם נבחר דווקא להתחיל בהן אז נקבל אופטימום גדול יותר מ-5.8.

ב. נעדיף להשתמש באלגוריתם $Simulated\ Annealing$ אשר בו נוכל גם להגיע למצבים גרועים יותר, אבל כך אולי בהמשך להגיע למצבים טובים יותר, ולקוות שעד שהטמפרטורה תרד לאפס, נוכל להגיע לאופטימום האמיתי של הבעיה. כך נוכל לוודא אם האופטימום האמיתי הינו 5.8 או אף גבוה מכך.

ג. להלן מרחב החיפוש:



ע"י הפעלת האלגוריתם $SAHC$, קיימת הסתברות גבוהה כי הנקודה שתיבחר לא תוביל לאופטימום של הבעיה, אלא דווקא יתקבל מקסימום מקומי בלבד.

בחיפוש $Simulated\ Annealing$, גם אם נקודת ההתחלה לא נמצאת סביב נקודת האפס (לפי השרטוט), עדיין קיים סיכוי גבוה שדווקא נקודות שכן נמצאות סביב נקודת האפס יבחרו במהלך ריצת האלגוריתם וכך לבסוף נגיע למקסימום הגלובלי עד שהטמפרטורה תתאפס.

חלק רטוב – סעיפים ה' ו-ו'

1. היוריסטיקה שקבענו עבר שחקן $minimax$ זהה ליוריסטיקה שהגדרנו בשאלה 2. בנוסף, בסעיף זה גם מוסברת המוטיבציה לבחירת יוריסטיקה זו.

$$h_1(s) = 4 \cdot\ number\ of\ player_1\ possible\ moves - 5 \cdot\ number\ of\ player_2\ possible\ moves$$
$$h_2(s) = 5 \cdot number\ of\ player_1\ potential\ mills - 4 \cdot number\ of\ player_2\ potential\ mills$$
$$h_3(s) = 4 \cdot\ number\ of\ player_1\ soldiers - 5 \cdot\ number\ of\ player_2\ soldiers$$
$$h_4(s) = 5 \cdot number\ of\ player_1\ mills - 4 \cdot number\ of\ player_2\ mills$$
$$h_5(s) = 5 \cdot number\ of\ player_1\ almost\ mills - 4 \cdot number\ of\ player_2\ almost\ mills$$
$$h_6(s) = 5 \cdot player_1\ killed - 4 \cdot player_2\ killed$$
$$h_7(s) = 5 \cdot player_1\ num\ of\ kills - 4 \cdot player_2\ num\ of\ kills$$

$$h(s) = 2 \cdot h_1(s) + 30 \cdot h_2(s) + 50 \cdot h_3(s) + 30 \cdot h_4(s) + 15 \cdot h_5(s) + 30 \cdot h_6(s) + 30 \cdot h_7(s)$$

2. שחקן התחרות שלנו הוא בעצם השחקן $alphabeta$ שלנו. אנחנו משתמשים באותה יוריסטיקה שהגדרנו בסעיף הקודם. השחקן שלנו בכל תור, מפעיל את האלגוריתם $minimax - alphabeta$ באיטרציות לפי עומק המקסימלי של העץ שנבנה. ניקח את הערך שקיבלנו בשורש של העץ עם העומק הכי גדול שהאלגוריתם הספיק לסיים, ונעשה את הצעד (זה גם צעד, ואם צריך להרוג את שחקן של היריב אז גם איזה שחקן) של הבן הרלוונטי של השורש, לפי הערך של השורש. זה בעצם הצעד הכי טוב שאנחנו יכולים לעשות במקרה שהיריב הוא יריב אופטימלי ועושה את הצעד הכי טוב שהוא יכול לעשות בשביל עצמו.

3. במקרה של הגבלת זמן לכל תור:

בתחילת הפונקציה $make\_move$ חישבנו את הזמן שצריך לסיים עד אז את הפונקציה, נסמנו end_time.

כל פעם שהיינו נכנסים לפונקציה של $minimax$ או $alphabeta$ (גם רקורסיבית), אנחנו בודקים אם לא הגענו לזמן $end\_time - 0.01$. אם הגענו לזמן הזה, אז נחזיר $None$, מהפונקציה של $minimax$ או $alphabeta$, מכיוון שזה אומר שלא הספקנו לסיים את האלגוריתם עד הסוף בעומק המקסימלי הנוכחי שניסינו. אחרי זה, פשוט החזרנו את הערך האחרון של השורש שקיבלנו, שהוא לא $None$.

במקרה של הגבלת זמן גלובלי:

חילקנו את הזמן בין התורות בצורה הבאה:

ב-28 התורות הראשונים של השחקן נתנו לכל תור את הזמן הבא:
*נשים לב שמספר התור $turn\_number$, הוא מספר התור שלנו, כשלא מחשיבים גם את התורות של היריב במספור.

$$-\left(\frac{(turn\_number + 1) * (turn\_number - 30)}{5000}\right) * 0.8 * game\_time$$

זו בעצם פרבולה עם נקודת מקסימום. נותנים הכי הרבה זמן לתור ה-14, ועד התור ה-14, מגדילים את כמות הזמן שנותנים לתור, ומהתור ה-14 מקטינים את כמות הזמן שנותנים לתור.

הסיבה שהחלטנו לעשות זאת, כי שמנו לב, שמספר התורים הרבה פעמים למשחק לשחקן מסויים, הוא בממוצע 28 ואף פחות, ולכן הקדשנו 80 אחוז מהזמן הכולל ל-28 התורות הראשונים של השחקן. בנוסף, בדרך כלל במשחקי חשיבה בסגנון המשחק הנתון, חושבים הכי הרבה דווקא באמצע המשחק, ולא בסוף או בהתחלה, ולכן רצינו לתת דווקא לאמצע המשחק יותר זמן, מאשר ההתחלה והסוף.

מהתור ה-29 ועד התור ה-40 נתנו לכל תור את הזמן הבא:

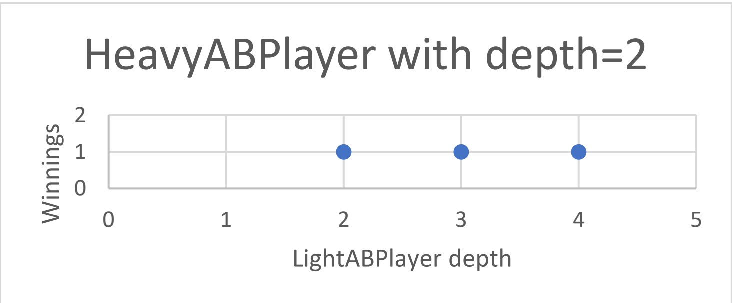
$$\max\left(0.1, -game\_turn * \left(\frac{turn\_number}{1210} - \frac{5}{121}\right)\right)$$

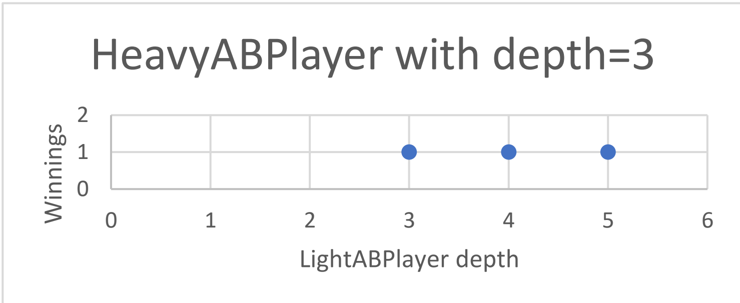
זה קו ישר ששיפועו שלילי, לכן לתור 29 אנחנו נותנים $\frac{5 * game\_time}{121}$ או 0.1 אם הוא גדול יותר ולאחר מכן, אנחנו מקטינים את כמות הזמן לכל תור עד שמגיעים ל 0.1 שניות לתור.

נשים לב שסכום הזמנים של 40 התורות הראשונים הוא קרוב ל- $game\_time$, אך לא בדיוק, אלא מעט פחות. אמנם לרוב המשחקים לוקח פחות מ-40 תורות לשחקן על מנת להגיע להכרעה, אך הרבה פעמים, השחקנים מגיעים בסוף המשחק ל"התשה". זהו מצב שבו השחקנים חייבים לשחק כמה שיותר מהר וזאת על מנת שהיריב יפסיד בגלל מגבלת הזמן שלו.
בנוסף, לקראת הסוף הרבה פעמים יהיו עצים הרבה פחות עמוקים (שהרי המשחק מתקרב לסיומו), ולכן גם רצינו לשים זמנים קצרים יותר בסוף.

4. כאשר הרצנו את המשחקים עם ערכי מגבלת זמן נמוכים (5-10 שניות), לפעמים $Minimax$ ניצח ולפעמים $Alphabeta$ ניצח. תוצאה זו סבירה כיוון שכאשר מגבלת הזמן לתור נמוכה, האלגוריתם $Alphabeta$ לא מצליח לפתח עץ עם עומק משמעותי הרבה יותר ביחס לאלגוריתם $Minimax$ ולכן ביצועי שני השחקנים דומים מאוד.
לעומת זאת, כאשר הרצנו את המשחקים עם ערכי מגבלת זמן גבוהים יותר, קיבלנו כי $Alphabeta$ ניצח ברוב המשחקים, וזאת כפי שמצופה. כאשר לשחקנים היה נתון זמן גדול יותר לביצוע תור, $Alphabeta$ הצליח לפתח עץ עם עומק גדול יותר בהשוואה לאלגוריתם $Minimax$ ולכן הצליח לבצע מהלכים יותר טובים ובכך לנצח.

5. להלן תוצאות הניסויים:





כפי שניתן לראות, בשני הניסויים השחקן $HeavyABPlayer$ תמיד ניצח. הסיבה לכך היא שהיוריסטיקה של $LightABPlayer$ אינה מספיק טובה על מנת לנצח, וזאת למרות שהשחקן $LightABPlayer$ מפתח עץ עמוק יותר.