בינה מלאכותית ש"ב 2

חלק ב':

2. השחקן הבסיסי עובד כך שעבור כל צעד אפשרי שלו הוא בודק את כל המצבים האפשריים כתלות בצעדים של היריבים שלו ומחשב בורם ערך בעזרת הפונק' היוריסטית שלו ובוחר את הצעד שיביא אותו לערך המקסימלי.
היוריסטיקה היא פונק' של מס' המהלכים שנשארו ומס' הפירות שהוא עוד לא אכל (כאלה שעל הרצפה וכאלה שנחשים חיים אכלו) ומחשב סכום של סדרה הנדסית עם q=discount_factor=0.5.
היוריסטיקה עוזרת לו להימנע ממוות מכיוון שאם הוא מת אז הערך המוחזר הוא האורך הנוכחי שלו ואם הוא חיי אז הערך יהיה האורך הנוכחי פלוס סכום הסדרה ההנדסית שהוזכר ולכן ההערך של מצב בו הסוכן חיי תמיד יהיה גדול יותר ממצב בו הוא מת פרט לאם הוא השחקן האחרון בחיים ולא נשארו יותר פירות לאכול, כלומר אין יותר טעם לשחק.

חלק ג':

1.
$$h(s) = \begin{cases} 0 & \{isDead \\ snakeLength & \{noFruitsLeft \\ \frac{1}{1+(dlosestFruitDist)} + snakeLength & \{noOpponentsLeft \\ \frac{1}{width+height+closestFruitDist} + snakeLength & \{noAttainableFruits \\ snakeLength & \{snakeLength > 0.5 \bullet numFruits \wedge isWinner \\ \frac{1}{1+closestAttainableFruit} + snakeLength & \{else \end{cases}$$

כאשר:
Dist - מרחק מנהטן בין ראש נחש לפרי
Attainable - הפירות אשר הסוכן קרוב אליהם יותר מכל סוכן אחר (בוודאות יכול להשיג אותם)

2. היא תשפר מכיוון שהסוכן הבסיסי בעצם זז באופן רנדומלי תוך כדי הימנעות ממוות אבל הסוכן המשופר עוקב אחרי המצב של המשחק ושואף להגדיל את האורך שלו כמה שאפשר

חלק ד':

2. האלגוריתם יעבוד יותר טוב בעיקר בתרחישים מסוכנים בהם הנחש עלול למות. המקרים הנ"ל הם מקרים בהם הנחש נמצא במקום "צפוף" שמוגבל ע"י גבולות הלוח, נחשים יריבים או אפילו הגוף של עצמו. במקרים כאלה מאוד חשוב להיות מסוגל להסתכל קדימה ולראות כיצד צריך לזוז על מנת שהנחש לא יכנס למצב ללא מוצא בו הוא בהכרח ימות. לכן ככל שעומק החיפוש של minmax יותר גדול כך הנחש יוכל להסתכל רחוק יותר ולהימנע מהפסד בטוח.
3. מס' הבנים יהיה כמס' הפרמוטציות של הוקטור, כלומר 3^k ולכן זה לא באמת פרקטי בפועל מכיוון שעבור מצב בו יש המון נחשים, לדוגמה עבור 20 יהיו $3.5 \bullet 10^9$ לצומת מינימום 1 וסכ"ה עבור עומק חיפוש של k ו-3 מהלכים חוקיים בכל טור יהיו $3^{20 \bullet k} \bullet 3^k = 3^{21 \bullet k}$ עלים בעץ שזו כמות אסטרונומית של מצבים.
4. ההנחה היא שהיריבים עובדים ביחד ולא לחוד, כלומר הם עובדים ביחד על מנת לפגוע כמה שיותר במצב של הסוכן שלנו ולא על מנת למקסם את הנק' של עצמם. הנחה הזו שגויה מכיוון שבמשחק אין קבוצות וכל שחקן מנסה לשפר את המצב של עצמו בלבד ולכן לדוגמה יריב א' הורג יריב ב' כאשר יריב ב' היה גדול יותר מיריב א' ומהסוכן שלנו, כתוצא מכך המצב של יריב א' והסוכן שלנו השתפר אבל המצב של היריבים כקבוצה הפך ליותר גרוע (גודל הקב' קטן, פחות אורך כולל וכו').
5. a)

כל שכבה בעץ תיהיה שייכת לשחקן מסויים כאשר השורש הוא הסוכן שלנו. לכל צומת, מתחזקים רשימת מהלכים של השחקנים כך שלכל צומת s כך שהצומת נמצא בשכבה i נפתח בן לכל מהלך חוקי שהשחקן ה-i%n יכול לבצע,כאשר n זה מס' השחקנים. כאשר נגיע לעלה נחשב לכל שחקן את פונק היוריסטיקה

ונשמור רשימה שאותה נפעפע במעלה העץ. כאשר כל צומת תבחר את היוריסטיקה המקסימלית שמתאימה לשחקן של השכבה בה הצומת נמצאת. כלומר, אם צומת נמצאת בשכבה ה-i אז היא תחזיר את הרשימה שבה הערך במקום i%n הוא הכי טוב ותחזיר אותה לאב. בסופו של דבר הסוכן שלנו יקבל רשימה ויחזיר את הערך שנמצא במקום ה-0.

היתרונות והחסרונות של שתי השיטות הן בדיוק הפכים, יתרונות של אחד הם החסרונות של השני ולהיפך ולכן נשווה:

- באלגוריתם השני כל סוכן בוחר את הערך הכי טוב לו ובכך מבטל את המשחק הקבוצתי שנוצר במימוש הראשון וכתוצא מכך אינו מחזיר את ערכים יוריסטים שהסיכוי שיקרו אפסי
- באלגוריתם השני עומק העץ גדול הרבה יותר מעומק העץ באלגוריתם הראשון ולכן מס' הקריאות הרקורסיביות גם כן גדול יותר וכתוצאה מכך המחיר בזיכרון עולה וכמובן עלול לגרום ל-stack overflow מוקדם יותר מאשר באלגוריתם הראשון.

b)

האלגוריתם השני מניח שהשחקנים משחקים בטורות וכתוצאה מכך האפשרויות ששחקן יכול לבחור מהם את הערך היוריסטי שהכי טוב לו מסונן ע"י השחקן שבא לפניו (שכבה אחריו בעץ חיפוש) ולכן לא בהכרח מייצג את המהלך הכי טוב שהיריב יעשה במציאות.

<u>חלק ה'</u>:

2. a)

מבחנית זמן ריצה האלגוריתם שמשתמש באלפא ובטא יהיה במקרה הגרוע שקול לאלגוריתם בחלק הקודם ובממוצע יותר טוב בגלל הגיזומים שחוסכים בפיתוח תתי עצים מיותרים.

b)

מבחינת מהלכים, לפי המימוש שלנו הוא יהיה זהה מכיוון שאנו לא בוחרים רנדומלית מבין המהלכים בעלי ערך יוריסטי הכי טוב ולכן הוא דטרמיניסטי, כתוצאה מכך למרות שהגיזום לא מפתח צמתים מסוימים שעשויים להיות בעלי ערך יוריסטי זהה (בהכרח לכל היותר שקולים לערך היוריסטי עד כה מכיוון שהגיזום מוריד רק מצבים בהם הערך היוריסטי יהיה קטן שווה לערך היוריסטי הכי טוב) הם לא יבחרו במימוש שלנו. אבל אם היה רנדומליות בבחירת המצבים אז היה שוני בחלק מהמקרים מכיוון שהמבחר מצבים ממנו האלגוריתם היה בוחר היה קטן יותר (חלק מהמצבים הכי טובים לא היו מפותחים).

<u>חלק ו'</u>:

1. חייבים להניח כי מהלך המשחק זהה מכיוון שאחרת הטופוגרפיה של התועלת תשתנה כל פעם ולא יהיה ניתן למצוא את סט המהלכים הטוב ביותר. כלומר, תלוי במצב ההתחלתי ובמהלכי היריבים המקסימומים המקומיים והגלובלים ישתנו וכתוצאה מכך המהלכים שצריך לעשות. לדוגמה אם הסוכן מתחיל משמאל לפרי אז על מנת לעלות למקסימום הקרוב ביותר הוא יצטרך לפנות ימינה אבל אם הוא מתחיל מימן לפרי אז הוא יצטרך לפנות שמאלה.
2. תלוי בפונק' התועלת ומס' משתמשים, אם נניח שפונק' התועלת אומרת אם ניצחת/הפסדת/תיקו (בדומה לתרגול) ויש רק שני שחקנים אז המשחק יהיה סכום אפס (אחד ניצח והשני הפסיד או תיקו), אחרת נניח עבור אותה תועלת ומספר שחקנים שונה אז הסכום לא יהיה 0 (תמיד יותר מפסידים מאשר מנצחים ולכן סכום שלילי) או עבור נניח פונק' תועלת שמחזירה את אורך הנחש אז התארכות של נחש אחד לא בא על חשבון התארכות של נחש אחר (כן על המקסימום שאליו הוא יכול להגיע אבל לא על אורך נוכחי) ולכן גם כן לא סכום 0.
3. מרחב המצבים הוא כל הפרמוטציות באורך לכל היותר 50 (כולל) של המהלכים האפשריים של הסוכן (ימין,שמאל,ישר)
4. האופרטורים שלנו הם המהלכים האפשריים שהסוכן יכול לעשות באותו הטור, מס' האופרטורים בכל איטרציה הוא כמס' המהלכים החוקיים שניתן לעשות באותו מצב והוא 0 אם מת ו-3 אם חיי
5. נצטרך לבצע לכל היותר 50 איטרציות (אם התחלנו ממצב בו סט המהלכים ריק)

6. הפונק' המתאימה היא כזו שתחשב את אורך הנחש אחרי שיבוצעו סט המהלכים הנתון (מקבל כפרמטר), הפונק' תעבוד בכך שהיא תריץ משחק שמצב הלוח ההתחלתי שלו הוא הלוח שעליו סוכם (לפי סעיף 1) ותריץ משחק שמס' התורות שלו שווה למס' המהלכים עד כה ובסוף המשחק תחזיר את אורך הנחש פלוס קבוע (נניח 1) עבור זה שהנחש עדיין בחיים. כתוצאה מכך הטופוגרפיה שנוצרת לנו היא שמשבצת עם פרי תביא לעליה, משבצת ריקה תשאיר אותנו באותו הגובה ומשבצת עם נחש יריב שיביא למוות יביא לירידה וכתוצאה מכך אם הנחש ליד פרי הוא בהכרח יקח אותו, אחרת הוא יזוז רנדומלית (מקסימום מקומי) תוך כדי שהוא מתחמק ממוות.
7. החלטנו להתחיל ממצב דטרמיניסטי בו רשימת הצעדים ריקה, זאת מכיוון שב-local_search החלטנו לממש סוג של beam-stochastic-hc ולכן המצב ההתחלתי הנ"ל הוא נק' השווה טובה לכך מכיוון שהסיכויים שלו להצליח לצאת מנק' מקסימום מקומית נמוכים ויאפשר להשוות עם התוצאות של האלגוריתם beam-SHAC ולבדוק האם המימוש טיפל בבעיית המקסימום המקומי (שזה רוב הטופוגרפיה של במשחק).
8. השינוי הנ"ל לא ישפיע על מצבי המשחק (הצעדים) שבאו לפניו מכיוון שהצעדים הנ"ל נעשו ללא תלות במהלכים עתידיים שהסוכן יעשה, אבל כל המצבים מהמצב ה-i והלאה יהיו שונים מהמצבים לפני השינוי מכיוון ששינינו את הפרמוטציה של הצעדים ולכן התוצאה הסופית יכולה ליהיות כל דבר. כלומר, הסוכן עלול למות במהלך המסלול או לגדול לאורך שונה או שהאורך שלו בסוף יהיה זהה, זאת מכיוון שהחרנו מסלול שונה במרחב המצבים. שינוי כזה ישפיע על לכל היותר $50 - i + 1$ (מהמצב ה-i עד וכולל המצב ה-50 במסלול).
10. הנחש ניצח והגיע לאורך 5, וקטור הפעולות הטובות ביותר הוא:

[<GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.LEFT: 0>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>]

11. בחרנו באלגוריתם beam-stochastic-hc.

לדעתנו האלגוריתם יעבוד טוב מכיוון שבזכות העובדה שהוא מפתח מספר מסלולים שונים במקביל, יהיה לו יותר סיכוי להיחלץ ממקסימומים לוקלים שהם בעצם רוב המצבים במשחק מכיוון שהפעמים היחידות שהסוכן יכול לעלות הם כאשר הוא נמצא ליד פרי ובכל שאר המצבים הוא פשוט בוחר מהלך רנדומלית מכיוון שהוא לא ליד פרי (עליה) או ליד יריב (ירידה). בנוסף האלגוריתם הוא סטוכסטי ולכן מתעדף לפתח צמתים בעלי תועלת גבוה יותר ובכך להביא לאורך טוב יותר של הסוכן. השינויים העיקריים שעשינו הם לוודא כי התוצאה של המהלכים תיהיה באורך 50. ממשנו כך שכאשר אין בכלל מצבים משפרים אז

האלגוריתם יבחר את k המצבים הטובים ביותר מבין אלה שלא משפרים אבל גם לא פוגעים וימשיך את הריצה בפיתוח שלהם ואם כל המהלכים מובילים למוות אז שפשוט יתקדם קדימה. את האלגוריתם אנחנו מריצים עם k=15 (מס' מצבים לפיתוח במקביל) כאשר כל המצבים ההתחלתיים מגרילים את 10 הצעדים הראשונים.

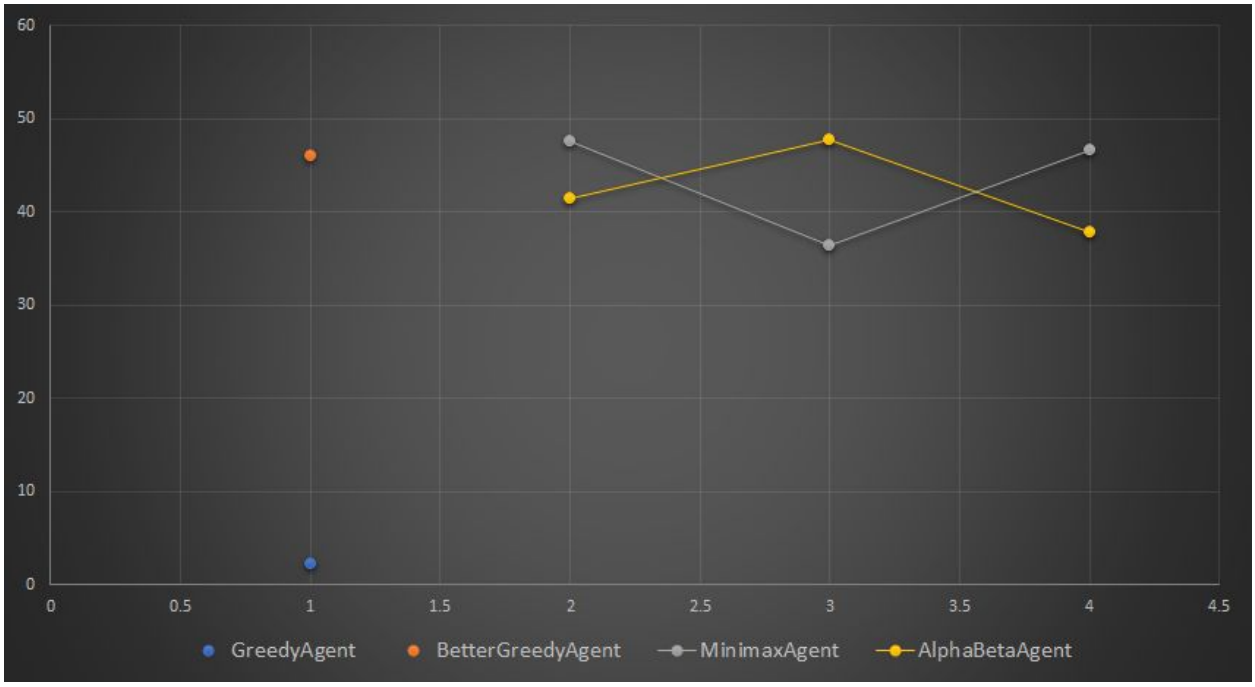
12. המצבים והאופרטורים שלנו הם זהה לאלגוריתם הקודם, כל מצב הוא פרמוטציה של לכל היותר 50 מהלכים, כאשר האופרטורים הם המהלכים האפשריים של הסוכן באותו מצב.

14. הסוכן ניצח, הגיע לאורך 8, מס' האיטרציות הוא 489 ווקטור המהלכים הכי טוב הוא:

[<GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.RIGHT: 2>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.STRAIGHT: 1>, <GameAction.LEFT: 0>, <GameAction.RIGHT: 2>, <GameAction.LEFT: 0>, <GameAction.LEFT: 0>]

חלק ז':

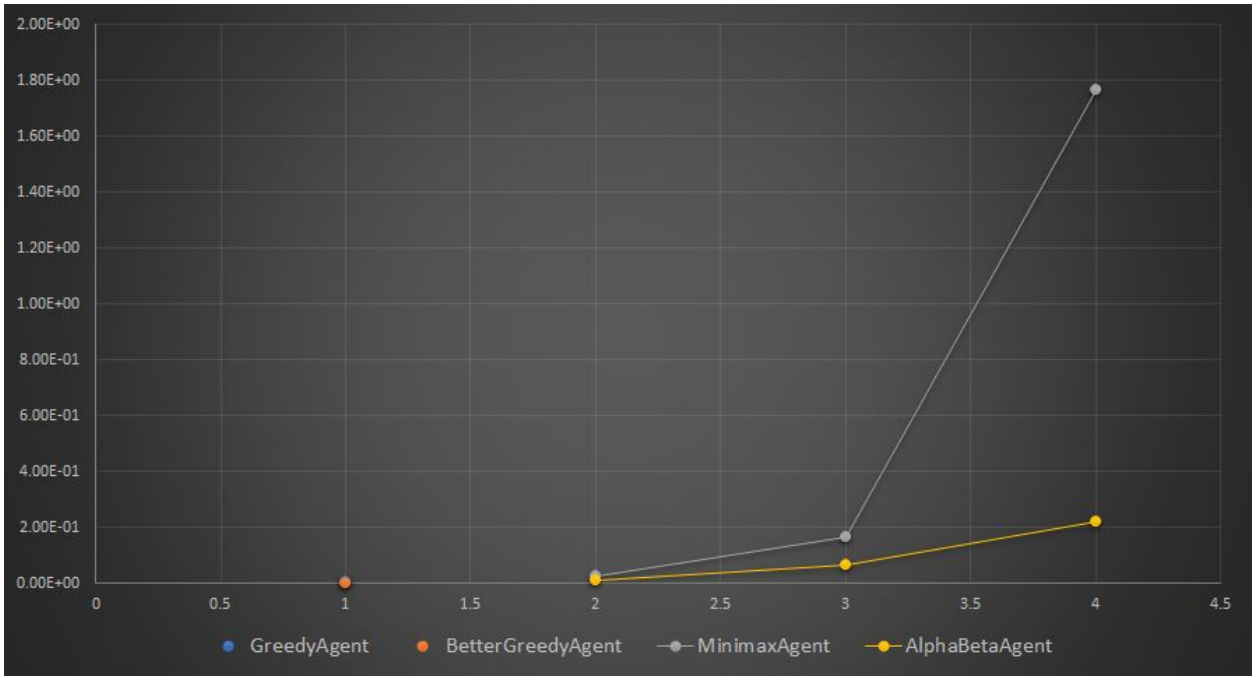
.2



d=4	d=3	d=2	d=1	
			2.2	**GreedyAgent**
			46	**BetterGreedyAgent**
46.6	36.4	47.6		**MinimaxAgent**
37.9	47.8	41.4		**AlphaBetaAgent**

3. מהגרף ומהטבלה ניתן להסיק כי BetterGreedyAgent מספק תוצאות דומות אם לא טובות יותר מאשר MinimaxAgent ו-AlphaBetaAgent לכל עומק ובנוסף העומק לא בהכרח משפיע על הניקוד של הסוכנים. זה נוגד את הציפיות שלנו מכיוון שהיינו מצפים כי הניקוד של minimax ו-alphabeta יהיו הרבה יותר דומים זה לזה בכל עומק מכיוון שעבור כל מצב הם מחזירים את אותו המהלך (לפי המימוש שלנו), לדעתנו זה נובע מהיוריסטיקה שלנו שגורמת לסוכן ללכוד את עצמו ולמות ולכן גורמת ל-variance גבוה בתוצאות. בנוסף היינו מצפים כי minimax ו-alphabeta ישיגו ניקוד יותר טוב מאשר BetterGreedyAgent (לפי הטבלה זה קרה אבל עם הפרש נמוך). זאת מכיוון שלמרות שהם פסימים מדי ביחס ליריב (שמסתובב במקום), יש מספיק תורות בשביל שהסוכן יוכל להגיע לכל הפירות ולא לפספס כמה בגלל היוריסטיקה המחמירה. אבל התוצאות לא משקפות זאת בגלל ה-variance הגבוה.

.4



d=4	d=3	d=2	d=1	
			7.20E-03	**GreedyAgent**
			2.98E-03	**BetterGreedyAgent**
1.77E+00	1.67E-01	2.75E-02		**MinimaxAgent**
2.22E-01	6.37E-02	1.05E-02		**AlphaBetaAgent**

5. המסקנות הן שחישוב היוריסטיקה שלנו יותר מהיר מאשר החישוב של היוריסטיקה ש-GreedyAgent משתמש ובנוסף כי בכל עומק הסוכן של alphabeta יותר מהיר מאשר הסוכן של minimax ככל שהעומק גדול יותר אז היחס בשיפור גדול יותר.
זה תואם את הציפיות שלנו מכיוון שהיוריסטיקה של GreedyAgent מפתחת כל מצב אפשרי עבור הטור הנוכחי בניגוד להיוריסטיקה שלנו שמחזירה ערך כפונק' של המצב הנוכחי (בלי לפתח עוד מצבים), בנוסף ציפינו כי alphabeta יהיה יותר מהיר מ-minimax מכיוון שהוא גוזם תתי עצים ושהיחס בשיפור המהירות יהיה ביחס ישר לעומק החיפוש מכיוון שאז alphabeta יוכל לגזום תתי עצים גדולים יותר ולחסוך יותר זמן.

6. מהניסוי ניתן לראות כי היוריסטיקה שלנו מתגברת תמיד על היוריסטיקה הפשוטה, בנוסף ניתן לראות כי שינוי העומק לא משפיע, מכיוון שהיוריסטיקה שלנו לא מונעת לכידה עצמית של הסוכן וכתוצא מזה מוות. סביר להניח שהשינויים בעומק אינם משפיעים מכיוון שהם קטנים מדי, אם היינו מריצים עם עומק של 10 (שזה לא פרקטי) אז היינו רואים שיפור משמעותי ויתרון גדול יותר ל-Minimax ו-AlphaBeta על פני ה-BetterGreedy.
מבחינת איכות ביצועים ביחס לזמן ממוצע למהלך (בהינתן עומקים קטנים), הסוכן הכי טוב הוא BetterGreedy מכיוון שהוא נותן ביצועים דומים אם לא יותר טובים ביחס לשאר הסוכנים ובשבריר הזמן שלוקח לסוכנים האחרים לבצע מהלך.
אם הייתה מגבלת זמן במקום עומק אז התוצאות יהיו שונות כתלות בכמות הזמן המסופק למהלך. בה"כ נניח מקבעים את עומק החיפוש לפני תחילת הריצה ל-4 וכשנגמר הזמן אז הסוכנים מחזירים את המהלך הכי טוב שהם הספיקו לפתח, אז אם כמות הזמן שמספקים למהלך מספיק גדולה בשביל שהמינימקס יפתח את כל העץ שלו אז התוצאות יהיו זהות לטבלה עבור עומק 4. אבל אם נניח הוא קטן יותר אז עלול לקרות מצב בו המינימקס והאלפא בטא לא יספיקו לפתח את כל העץ ולא יבדקו מסלולים מסויימים ולכן יבצעו מהלך גרוע. מעבר לזה במימוש שלנו אם אין מספיק זמן בשביל לפתח יותר ממהלך אחד אז הסוכנים יסתובבו במקום מכיוון שאנחנו לא מגרילים מהלכים וכתוצאה מכך יקבלו ניקוד דומה ל-GreedyAgent.

<u>חלק ח'</u>:
סוכן התחרות שלנו ממש alphaBetaMiniMax לעומק 2 ומשתמש בהיוריסטיקה שמימשנו בסעיפים קודמים, כאשר לפני פיתוח של כל צומת הוא בודק האם נשאר לו זמן, אם כן אז הוא ממשיך כרגיל אחרת הוא יפעפע את התשובות במעלה הרקורסיה ויחזיר את המהלך הטוב ביותר שהוא מצא עד כה. בחרנו באלגוריתם הנ"ל מכיוון שהוא יוכל לגזום צמתים ובכך לחסוך בזמן ריצה ובנוסף האלגוריתם מסתכל רק 2 צעדים קדימה מכיוון שזהו עומק סביר על מנת שיוכל לבדוק את רוב המסלולים האפשריים תחת מגבלות הזמן הנתונות, אחרת עלול להתרחש מצב שהוא יספיק לבדוק רק חלק קטן מהמסלולים או אפילו רק אחד ואז לבחור מהלך גרוע בהרבה ביחס שאר המסלולים שהוא לא הספיק לפתח.