מבוא לבינה מלאכותית – תרגיל 2

מגישים:

אביעד טל 307854463

דקל מירום 204690366

חלק ב'

שאלה 2

השחקן בוחן את כל האפשרויות לתנועה ועבור כל אפשרות בודק מה מצב הלוח בהינתן הצעדים האפשריים של היריב (בוחן רק יריב אחד אפילו אם יש יותר). עבור כל מצב אפשרי הוא מריץ את פונקציית היוריסטיקה ובוחר את האפשרות שמביאה למקסימום. במידה ויש כמה כאלה, הוא יגריל אחת מהן.

היוריסטיקה המצויה בקובץ מבוססת על העיקרון שבכל תור יש סיכוי לאכול פרי (בדוגמה – חצי), וככל שמסתכלים יותר תורות קדימה הסיכוי קטן. היוריסטיקה בוחנת האם הנחש חי ואם כן מחזירה את אורך הנחש בתוספת מספר שמשקף את הניקוד של הנחש בהנחה שיאכל את שארית הפירות בהסתברות שחושבה בשארית הצעדים שנותרו למשחק (סכום טור גיאומטרי).

היוריסטיקה מנסה לשמר את חיי הנחש על ידי כך שבעת צעד שיוביל למוות הניקוד יהיה אורכו של הנחש, בעוד שכל צעד אחר יוסיף ניקוד חיובי שמייצג את הניקוד הפוטנציאלי שהנחש ירוויח בשארית המשחק. מכאן שתמיד יהיה רווחי לבחור בצעד שלא הורג את הנחש.

חלק ג'

שאלה 1

$$h(s) = \begin{cases} (w_1 * len) + \left(w_2 * \frac{1}{nearestFruitDist}\right) + \left(w_3 * \frac{1}{avgFruitdDist}\right) & if\ alive \\ w_1 * len & if\ dead \end{cases}$$

כאשר:

$$Len\ =\ the\ length\ of\ the\ snake\ in\ state$$

$$nearestFruitDist\ =\ Manhattan\ distance\ between\ the\ snake's\ head\ and\ the\ nearest\ fruit$$

$$avgFruitDist\ =\ \frac{Manhattan\ distance\ between\ the\ snake's\ head\ and\ all\ the\ fruits}{number\ of\ fruits}$$

$$w_1 = 3$$

$$w_2 = 2$$

$$w_3 = 1$$

שאלה 2

ביוריסטיקה שלנו אנו מכוונים שהנחש יאכל כמה שיותר פירות במספר מינימלי של צעדים. הפרמטרים שבחרנו משרתים את מטרה זו בכמה דרכים. האורך של הנחש משקף את הצלחתו באכילת פירות, ולכן נעדיף מצב בו גודלו ארוך יותר, זה הפרמטר החשוב ביותר (ולכן משקלו בהתאם). מכיוון שאנחנו רוצים לאכול כמה שיותר פירות

במספר מינימלי של צעדים המרחק לפרי הקרוב ביותר הוא פרמטר חשוב שמקרב אותנו למטרה זו; נעדיף צעדים שמקרבים אותנו לפרי. בנוסף, מתוך הבנה שהפירות מפוזרים ברחבי המגרש נעדיף צעדים שמקרבים אותנו לפירות רבים ולכן צעדים שמקרבים אותנו בממוצע לכל הפירות עדיפים.

השחקן שבנינו מעדיף צעדים שמקרבים אותו לפירות, בעוד שהשחקן המקורי כמעט לא מתחשב בזה. מבחינת השחקן המקורי אם יש פרי במרחק 2 ומעלה הוא לא ינסה להתקרב אליו, אלא רק יבחר בצעד הזה רק אם יש פרי קרוב אליו – אחרת צעדיו באקראי (עד כדי צעדים שיהרגו אותו).

חלק ד'

שאלה 2

בהינתן היוריסטיקה "האנוכית" שלנו (שאינה מתייחסת לפעולות היריב), לא יהיה הבדל משמעותי בפעולות שנבחר. עם זאת במצבים מסוימים נראה השפעה, למשל:

- כשהנחש סוגר את עצמו בלולאה שתהרוג אותו, הוא יוכל לצפות זאת מראש ולהימנע מהצעד.
- כשנחשים אחרים אוכלים פירות הם לא יהיו במשוואה של פרי קרוב ביותר או מרחק ממוצע לכל הפירות.

שאלה 3

לכל צומת מינימום יהיו 3^k בנים. זאת מכיוון שיש 3 פעולות שכל נחש יכול לבחור ואנו מתחשבים בכל צירוף פעולות של k היריבים. במספר רב של שחקנים האלגוריתם לא פרקטי כי בכל צעד נצטרך לבדוק מיליארדי אפשרויות (וזה נהיה אף יותר קיצוני בעומקים גדולים).

שאלה 4

ישנה הנחה סמויה שכל הנחשים יבחרו בצעד שיעשה לנו את הנזק המקסימלי, בעוד שבפועל כל נחש רוצה להשיג כמה שיותר נקודות לעצמו על חשבון כל יריביו. מכאן נסיק שהנחה זו אינה נכונה והאלגוריתם פראנואידי.

שאלה 5

חלק א':

בשלב הבחירה של הפעולה הבאה, במקום לחלק לתור שלנו ותור היריבים (AGENT_TURN, OPPONENT_TURN) אז נפריד את היריבים לפי מספרם. בכל תור של יריב נבחר את האופציה שמביאה יוריסטיקה עם ערך מינימלי ונעבור ליריב הבא, בתור של הנחש שלנו נבחר במקסימום.

בניגוד לשיטה הקודמת שבה תור של המשחק מיוצג על ידי 2 שכבות של העץ, בשיטה הזו כל תור יהווה n+1 שכבות כאשר n הוא מספר היריבים. החיסרון של השיטה היא שהיא דורשת יותר זיכרון (עומק רקורסיבי גדול יותר) ויותר זמן. מספר האפשרויות (ולכן העלים) בתת העץ של התור הזה לא משתנה בין השיטות, לכן אין הבדל בין בחירה של מינימום בין כל העלים בשיטה א' לבין פעפוע של מינימום על פני השכבות בתת העץ בכל אחד מתורות היריבים בשיטה ב'.

חלק ב:

ההנחה בשיטה א' היא שכולם מקבלים את ההחלטה בו-זמנית, ובשיטה ב' שהם מקבלים את ההחלטה זה אחר זה (כאשר כל שחקן יודע על ההחלטות של השחקנים שבחרו לפניו). ההנחה בשיטה א' יותר נכונה וקרובה למבנה הסטנדרטי של המשחק, לשחקנים אין ידע על הבחירה של השחקנים האחרים כשהם בוחרים איך לפעול. ניתן לחשוב על מימוש שמתחשב בפעולות היריב באמצעות תקשורת בין הסוכנים, אבל לא ראינו דוגמה לשיתוף פעולה כזה.

חלק ה'

שאלה 2

חלק א':

האלגוריתם אמור לשפר את זמן הריצה כי לא נפתח מצבים מיותרים שבהכרח לא נבחר (כי הם מעל המינימום הנוכחי בצומת מינימום או מתחת למקסימום הנוכחי בצומת מקסימום).

חלק ב':

לא יהיה שינוי בבחירת המהלכים כי גם ככה לא היינו בוחרים במצבים האלה אפילו אם היינו מפתחים אותם.

חלק ו'

שאלה 1

אם המשחקים לא היו זהים לא נצליח לזהות מסלול אופטימלי. עבור כל סדר מהלכים שתיתן ערך גבוה, ייתכן שבריצה נוספת נקבל ערך נמוך. בפרט – סדרת מהלכים שהייתה אופטימלית בריצה אחת עלולה להיות גרועה בריצה אחרת.

שאלה 2

לא. בתור התחלה, אין ניקוד שלילי ואורך הנחשים חיובי, מכאן שסכום הנקודות תמיד גדול מאפס. מעבר לכך - סך הניקוד משתנה ממשחק למשחק, הוא תלוי בכמות הפירות שנאכלו וגם במספר הנחשים שנשארו חיים בסופו.

שאלה 3

המצבים במרחב החיפוש הם וקטור באורך N עם מספרים שמייצגים פעולה בכל תא (ישר, שמאל, ימינה). סך הכל יש 3^N מצבים.

שאלה 4

האופרטורים הם:

- החלפת הפעולה הנוכחית בישר.
- החלפת הפעולה הנוכחית בשמאלה.
- החלפת הפעולה הנוכחית בימינה.

בפועל בכל איטרציה נפעיל 2 אופרטורים כי בכל מצב יש כבר ברירת מחדל מהמצב ההתחלתי.

שאלה 5

נצטרך לבצע N איטרציות.

שאלה 6

לכל וקטור צעדים פונקציית היוריסטיקה תריץ את המשחק ותבחר פעולות לסוכן לפי ההוראות של הוקטור ותחזיר את הניקוד. במילים אחרות – ניקוד גבוה יותר זה מצב טוב יותר.

שאלה 7

המצב ההתחלתי יהיה "ישר" בכל תאי הוקטור. האינטואיציה היא ששינוי מוקדם מייצר השפעה גדולה יותר להמשך המסלול. ניתן לחשוב על זה כקו ישר שצריך להחליט היכן שוברים אותו.

שאלה 8

כל שינוי שנבצע בצעד ה-i ישפיע על כל המצבים שיגיעו מנקודה זו במשחק והלאה. קיבוע הפעולה בתא ה-i תיכנס לכל המשחקים העתידיים ולא תשנה את הצעדים שהגיעו לפני כן. כל מצבי המשחק לפני מצב זה ישארו זהים, ולכן שינוי זה ישפיע רק על 3^{N-i} המצבים האפשריים ש"ממשיכים" את הוקטור שקיבענו עד כה.

<u>שאלה 10</u>

וקטור הצעדים שקיבלנו הוא:

[GameAction.LEFT,GameAction.LEFT,GameAction.LEFT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.LEFT,GameAction.STRAIGHT,GameAction.LEFT,GameAction.LEFT,GameAction.STRAIGHT,GameAction.RIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.RIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.LEFT,GameAction.LEFT,GameAction.RIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.STRAIGHT,GameAction.LEFT,GameAction.LEFT,GameAction.STRAIGHT,GameAction.RIGHT,GameAction.RIGHT,GameAction.STRAIGHT]

האורך שהגענו אליו בריצה הוא 13, והניקוד הסופי הוא 14 כי הוא נשאר בחיים.

<u>שאלה 11</u>

נשתמש באלגוריתם Random-restart hill climbing with sidesteps כדי לנסות לשפר את הניקוד. בנוסף, נשנה את האופרטורים כך שישפיעו על 2 צעדים קדימה, כלומר יהיו לנו 9 אופרטורים לנסות בכל אחת מהאיטרציות (שמתוכן נפעיל בפועל 8). שינוי זה נועד לאפשר הסתכלות של צעד נוסף קדימה עבור מקרים בהם השיפור אינו מיידי.

האפשרות של restart נועדה לבדוק אלטרנטיבות לבחירה של המצב ההתחלתי, שאולי לא היה אופטימלי עבור המשחק הקיים.

<u>שאלה 12</u>

המצבים לא השתנו, הם וקטורים באורך N כאשר בכל תא יש ערך מספרי המתאר את הפעולה שצריך לבצע בתור המתאים.

האופרטורים הם זוגות של כיוונים (ישר, שמאלה וימינה) שלוקחים מצב נתון וערך תא i ומחליפים את הערך הנוכחי בתא ה-i ובתא ה-i+1 בערכים של האופרטור.

<u>שאלה 14</u>

הוקטור שיצא לנו הוא:

[GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.LEFT, GameAction.RIGHT, GameAction.STRAIGHT, GameAction.RIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.RIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.LEFT, GameAction.LEFT, GameAction.STRAIGHT, GameAction.RIGHT, GameAction.LEFT, GameAction.LEFT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.RIGHT, GameAction.STRAIGHT,

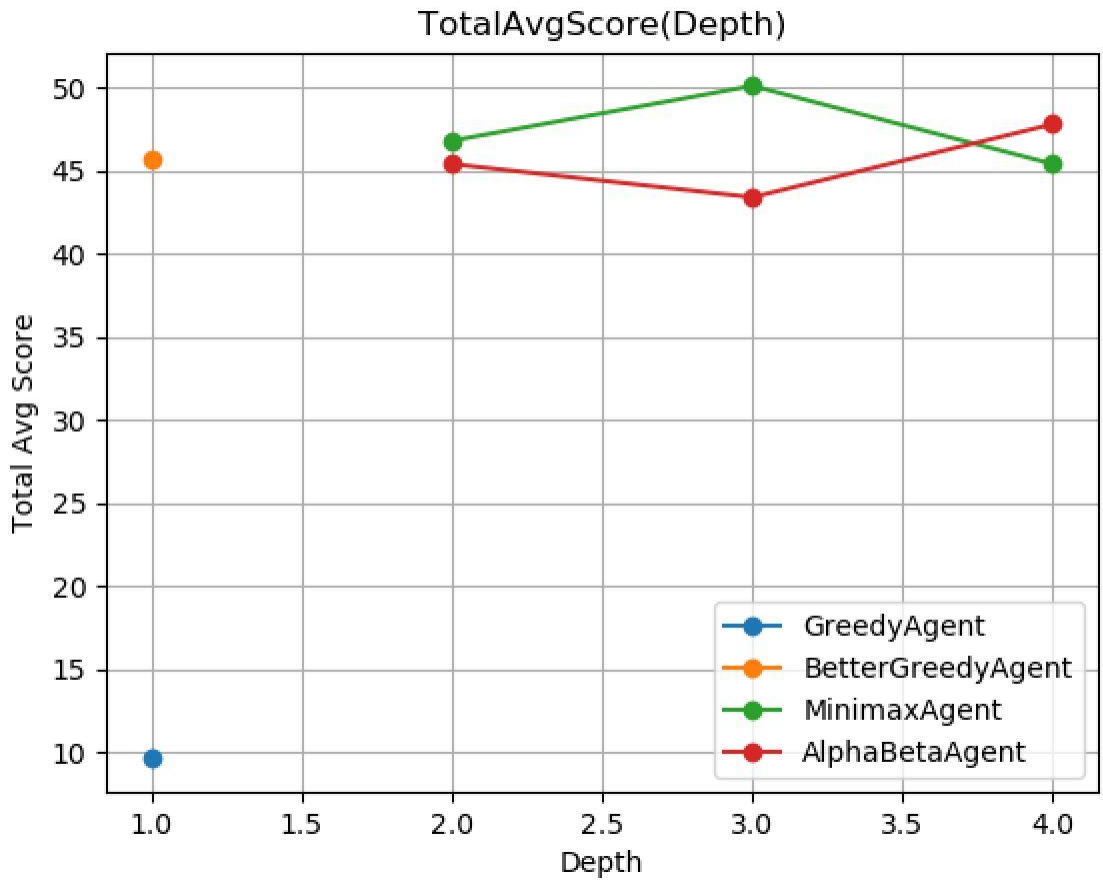
GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.RIGHT, GameAction.LEFT, GameAction.LEFT, GameAction.RIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.STRAIGHT, GameAction.RIGHT]

האורך שהגענו אליו בריצה הוא 9 והוא נשאר בחיים (כלומר הניקוד הוא 10). הוא ניצח בבחירה שלו אחרי 150 איטרציות, 50 עבור הריצה מהמצב ההתחלתי ועוד 100 עבור 2 התחלות מחדש עם וקטור אקראי.

האלגוריתם לא שיפר, זאת מכיוון שיש כאן השפעה רנדומית.

חלק ז'

שאלה 2



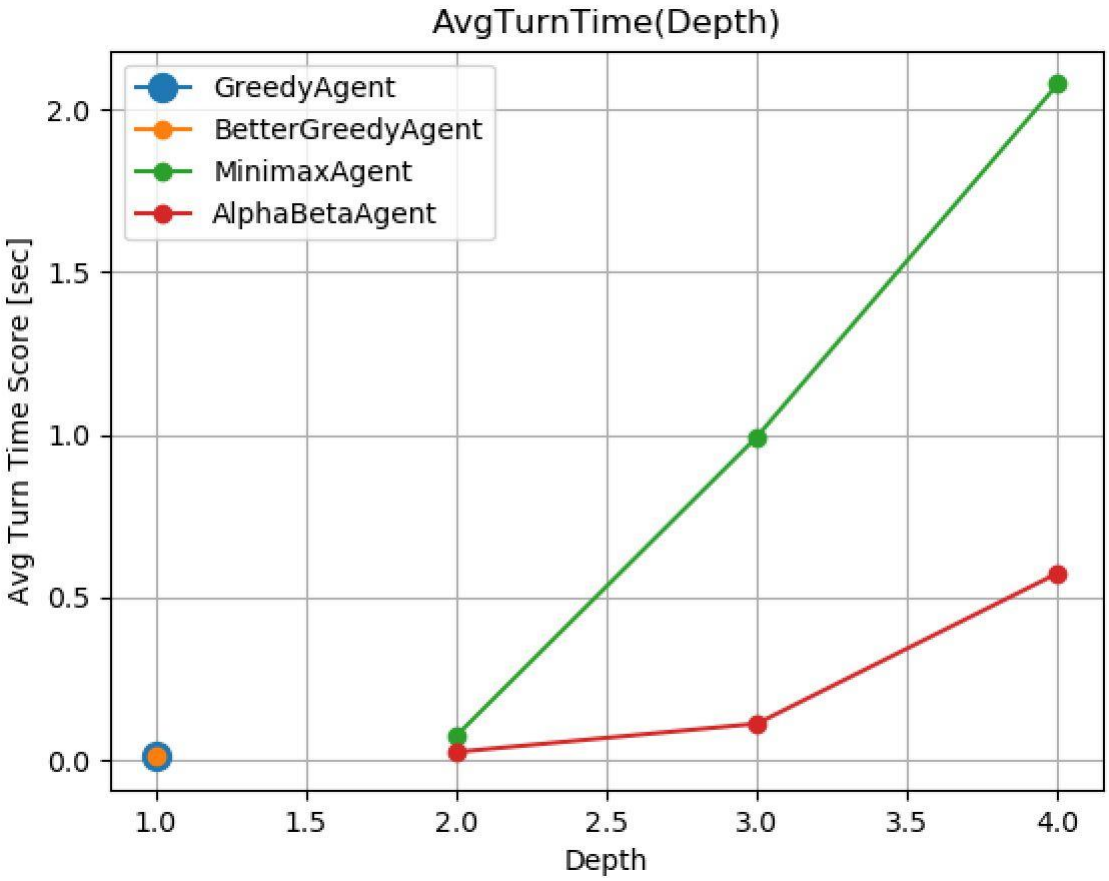
	D=1	D=2	D=3	D=4
GreedyAgent	10.5			
BetterGreedyAgent	45.7			
MinimaxAgent		46.8	50.1	45.4
AlphaBetaAgent		45.4	43.4	47.8

שאלה 3

הניקוד של ה-GreedyAgent נמוך (הסברנו בסעיפים קודמים מדוע הוא לא ממקסם ניקוד). האלגוריתם שלנו מגיע לתוצאות טובות גם בלי הסתכלות לעומק. מעבר לעומק 2 לא חל שיפור באלגוריתם Minimax או AlphaBeta. נשים לב גם שמבחינת ניקוד אין הבדל משמעותי בין Minimax ל-AlphaBetaMinimax, זאת מכיוון שהוא לא משנה את האלגוריתם בחירה אלא רק את זמן הריצה (ההבדל הקיים נובע מרנדומיות של המשחקים).

ציפינו לתוצאות הללו. מכיוון והיוריסטיקה שלנו לא מסתמכת על פעולות היריב, הסתכלות לעומק משפרת רק מקרים בהם הנחש נכנס ללולאה עצמית שתהרוג אותו ב-d צעדים הבאים. מכיוון ומקרים אלה נדירים השיפור אינו משמעותי סטטיסטית.

שאלה 4



	D=1	D=2	D=3	D=4
GreedyAgent	0.011521			
BetterGreedyAgent	0.014583			
MinimaxAgent		0.074059	0.993472	2.077844
AlphaBetaAgent		0.02487	0.111983	0.574608

שאלה 5

ניתן לראות שללא הסתכלות על צעדים קדימה האלגוריתמים בעומק 1 בוחרים צעדים במהירות רבה. כאשר מתחילים להסתכל בעומק העץ קל לראות שהזמן הממוצע עולה משמעותית, אך פחות באלגוריתם AlphaBeta שמקצץ ענפים לא רלוונטיים. במקרים הללו זמן הריצה גדל אקספוננציאלית בעומק העץ, כפי שחשבנו שיקרה.

<u>שאלה 6 – סיכום כללי</u>

השוואה יוריסטית בין GreedyAgent ל-BetterGreedyAgent מראה שבהפרש זניח בזמן החישוב לכל צעד ניתן להשיג שינוי משמעותי בניקוד. הסוכן שכתבנו מצליח להשיג את רוב הפירות על המגרש כמעט בכל המשחקים ומצליח למקסם את הניקוד שלו. הפרשי הזמנים זניחים ולכן נסיק שאפילו עם הגבלת זמן על חישוב, פעולות הסוכן שכתבנו ינצח את GreedyAgent.

כאשר נכנסים לניתוח מקיף על אלגוריתמי Minimax, נראה כי אין שיפור משמעותי מול הרצת היוריסטיקה ללא הסתכלות קדימה, זאת מכיוון שהיוריסטיקה לא מתחשבת בפעולות היריבים. המקרים היחידים שצפינו הבדל הם במקרים נדירים בהם הנחש נכנס ללולאה שהרגה אותו d צעדים קדימה ומנע זאת. קשה למדוד בפועל כמה פעמים הלולאה הזו נמנעה במהלך הריצה, אך מכיוון שהעומק אינו גבוה, אנו משערים שזה לא היה מדד משמעותי וסטיות הניקוד בין כל האלגוריתמים מקורה בעיקר בחלקים האקראיים של המשחק.
עם זאת, זמן החישוב לכל פעולה גדל משמעותית באלגוריתמים של Minimax ו-AlphaBeta, מכאן נסיק שהם **לא משתלמים** עם היוריסטיקה שכתבנו. גם אם נוסיף הגבלת זמן לאלגוריתמים האלה לא יהיה שינוי בביצועים שלהם.

ניתן להגיד בצורה חד משמעית שהקיצוץ שמבצע אלגוריתם AlphaBeta משמעותי לזמן הריצה בכל עומק, ולא פוגע בביצועים. בתוצאות שיצאו לנו יש שינויים קטנים בניקוד אך אלה נובעים מהאקראיות ואם נריץ את האלגוריתם על פני מאות משחקים נגיע לשוויון בממוצע הנקודות.

סוכן התחרות

הסוכן שהגשנו לתחרות מבוסס על BetterGreedyAgent, להלן סיכום של אופן בחירת הפעולה שלו וכן פונקציית היוריסטיקה החדשה שלו.

בחירת הפעולה

<u>מנגנון בריחה ממלכודת</u>

שמנו לב שבמהלך הריצה של BetterGreedyAgent שהוא מכניס את עצמו למצבי מלכוד – כלומר מתנגש בעצמו עקב חוסר ברירה בעקבות חוסר בהסתכלות קדימה. במקום להכניס הסתכלות בצעדים קדימה, בחרנו גישה אחרת לפתרון הבעיה. בתחילת כל קריאה של פונקציה getAction, אנחנו בודקים האם אנחנו במצב ממולכד. הגדרנו מצב זה כמצב בו המשבצת שנמצא מול הנחש (אחת קדימה בהמשך הכיוון שהוא כרגע פונה אליה) מכילה את עצמו, כלומר – מצב בו התקדמות קדימה תוביל להפסד. במצב כזה אנחנו בוחרים פעולה עבורו תוך התעלמות מהיוריסטיקה הרגילה.

אם הנחש עומד להתנגש בזנב של עצמו, נסתכל על זנב הנחש והמשבצות הסמוכות לו (ומאונכות לכיוון שאנחנו מתקדמים אליו). מכיוון שזה הזנב מובטח לנו שבאחת מהן לא יהיה נחש – נבחר לפנות לכיוון הזה.

אם הנחש עומד להתנגש בעצמו אך לא בזנב, אנו נבדוק את מיקום הזנב ביחס לראש. שמנו לב כי במצבים רבים המיקום היחסי של הזנב ביחס לראש נמצא בקורלציה עם הכיוון שצריך לפנות אליו כדי להיחלץ מלולאה עצמית. זה נובע מכך שהנחש באורך ממוצע נמוך יחסית במשחק עם 40 פירות לכל היותר, ולא סביר שיכנס ללולאה כפולה סביב עצמו, לכן ברוב המקרים הלולאה תהיה יחידה. במקרה של לולאה יחידה, נרצה תמיד לפנות לכיוון הזנב כדי להימנע מלסגור את עצמנו במלכודת.

במקרים הנדירים בהם הזנב נמצא בדיוק מאחורינו, אנחנו בוחרים להחזיר כיוון באקראי. החלטנו שבמשחק בו יש כמות מעטה יחסית של פירות המקרים אלה נדירים מספיק כדי שלא נכסה אותם.

<u>שאר מנגנון בחירת הפעולה</u>

מלבד השינוי בהוספת מנגנון הבחירה ממלכודת, הפונקציה זהה לזו של BetterGreedyAgent, מה ששונה זה היוריסטיקה שאנחנו משתמשים בה לחישוב הניקוד. בחרנו שלא להסתכל לעומק העץ כי זה לא הביא לתוצאות טובות מספיק והוסיף כמות גדולה של זמן לחישוב.

יוריסטיקה

$$h(s) = \begin{cases} (w_1 * len) + \left(w_2 * \frac{1}{nearestFruitDist}\right) + \left(w_3 * \frac{1}{avgFruitdDist}\right) + \left(w_4 * \frac{1}{goodFruitDistance}\right) & if\ alive \\ w_1 * len & if\ dead \end{cases}$$

כאשר:

$$Len = the\ length\ of\ the\ snake\ in\ state$$

$$nearestFruitDist = Manhattan\ distance\ between\ the\ snake's\ head\ and\ the\ nearest\ fruit$$

$$avgFruitDist = \frac{Manhattan\ distance\ between\ the\ snake's\ head\ and\ all\ the\ fruits}{number\ of\ fruits}$$

$$goodFruitDist = Manhattan\ distance\ between\ the\ snake's\ head\ and\ the\ nearest\ \text{good fruit}.$$

$$w_1 = 5.1$$

$$w_2 = 3$$

$$w_3 = 2$$

$$w_4 = 5$$

To determine "good fruits", we have calculated a score for every fruit on the grid. For each fruit, the score is calculated by the number of nearby fruits to the current one. Let x be our fruit and y be the fruit we are comparing nearby.

$$nearbyFruits = \{y | dist(x, y) \leq 4\} \text{ (This includes x)}$$

$$Score(x) = \sum_{y \in nearbyFruits} 5 - dist(x, y)$$

A good fruit is a fruit with a score of 10 or higher. The minimum score of any fruit is 5 (because it has distance zero from itself). Fruits with a score higher than that indicate a cluster of fruits, which are something we want to get near to.