תרגיל בית 2 : חיפוש רב סוכני
Multi-Player Snake

מטרות התרגיל

- התנסות באלגוריתמים לחיפוש רב סוכני.
- מימוש אלגוריתמים Minimax, Local-Search + שיפורים והרחבות.
- התנסות בפיתוח יוריסטיקה למשחק Multi-Player Snake.
- ביצוע מחקר המשווה ביצועים של שחקנים.

הערות

- תאריך הגשה : 1.1.20, 23: 59.
- את המטלה יש להגיש **בזוגות בלבד!**
- שאלות בנוגע לתרגיל יש לשלוח אך ורק **למייל הקורסי** : ai.technion@gmail.com
- המתרגל האחראי על התרגיל הוא **ליאור כהן.**
- בקשות לדחייה יש לחפש ראשית תשובה בסילבוס הקורס.
  במידה ולא קיבלתם מענה יש לשלוח מייל ל**גיא**.

- קִראו היטב את ההסברים וההוראות במסמך זה, מטרתם לסייע לכם בהבנת הדרישות של התרגיל. שימו לב להוראות ההגשה המצורפות בסוף התרגיל.
- הִתְעַדְכְּנוּ ברשימת ה-FAQ באתר הקורס בתדירות גבוהה, לפני פנייה בשאלות דרך המייל ולפני הגשת התרגיל. ההערות שתתפרסמנה באתר הקורס מחייבות את כלל הסטודנטים בקורס!
- עקבו בתשומת לב רבה אחר הוראות ההגשה המצורפות במהלך התרגיל ובסופו לפני הגשתו.
- בתרגיל זה, כמו גם בתרגילים הבאים בקורס, הרצת הניסויים עשויה לקחת זמן ולכן מומלץ מאוד להימנע מדחיית העבודה על התרגיל לרגע האחרון. **לא תינתנה דחיות על רקע זה.**

מבוא והנחיות

1. במטלה זו תתכננו ותממשו שחקנים למשחק Multi-Player Snake. הגדרת המשחק והחוקים יופיעו בחלק א' של מסמך זה. קראו אותם בקפידה.
2. המטלה מכילה חלק יבש וחלק רטוב.
3. **<u>העתקות</u> (גם מהאינטרנט) יטופלו בחומרה בדין משמעתי.**
4. מומלץ לחזור על שקפי ההרצאות והתרגולים בנושא חיפוש רב סוכני וכן על ההרצאות בנושא חיפוש לוקאלי בדגש על אלגוריתמים גנטיים לפני תחילת העבודה על התרגיל.
5. הקוד שאתם מקבלים מכיל את הקבצים הבאים:
   a. Submission.py- הקובץ היחיד שעליכם לשנות ולכתוב בו קוד, פה תממשו את השחקנים והאלגוריתמים שלכם.
   b. environment.py- הקובץ המרכזי המגדיר את המשחק. מכיל את הטיפוס GameState בו תעשו שימוש רב.
   c. agents.py- מכיל הגדרות של שחקנים.
   d. main.py- הקובץ שמגדיר את הרצת המשחק דרך שורת הפקודה (אין לכם צורך להכיר את הקוד בקובץ זה).
   e. utils.py – מכיל פונ' עזר להרצת אלגוריתמי חיפוש מקומי.
6. **שימו לב שעליכם לכתוב קוד ולשנות אך ורק את הקובץ Submission.py**
7. הקוד שסופק לכם עושה שימוש בספריות הבאות:
   a. gym
   b. numpy
   c. scipy
   d. pillow

על מנת שתוכלו להריץ את הקוד עליכם להתקין את הספריות הנ״ל. יש מספר דרכים להתקנת ספריות בפיתון, הפשוטה שבהן היא הרצת הפקודה

pip install packageName

על מנת להתקין חבילה ששמה packageName. את הפקודה הזו יש להריץ מחלון הפקודה שבעזרתו אתם מריצים קבצי פייתון.

אם אתם משתמשים בסביבת PyCharm (מומלץ), בסרגל הכלים בתחתית המסך ישנה לשונית בשם terminal. לחצו על לשונית זו, והקלידו את הפקודה בחלון שנפתח. אם ביצעתם את הפקודה בדרך זו תוכלו לדלג על קריאת המשך ההסבר.

אם אינכם עובדים עם PyCharm, פתחו חלון פקודה שמכיר את ה-python interpreter המותקן אצלכם. כדי לוודא שחלון הפקודה שלכם תומך בפיתון תוכלו להריץ את הפקודה

python --version

כדי לראות את גרסאת הפיתון שלכם. במידה וחלון הפקודה לא מכיר את פיתון תיזרק שגיאה. אם התקנתם את פיתון דרך סביבת אנקונדה, ישנו חלון פקודה מיוחד בד״כ שהותקן יחד עם אנקונדה שדרכו תצטרכו להריץ את הפקודות הנ״ל. שימו לב, סביבת PyCharm מזהה את אנקונדה וחוסכת לכם את העבודה מול החלון הנ״ל ישירות.

8. בסיום העבודה תגישו גם חלק יבש הכולל מענה לשאלות ודיווח תוצאות, עליכם לכתוב תשובה בדו״ח היבש בכל סעיף שלצדו תראו את הסימן 

חלק א' – הגדרת המשחק

במשחק ה- Multi-Player Snake, ישנם 2 או יותר שחקנים, אשר כל אחד מהם מיוצג ע"י נחש על לוח המשחק. כל שחקן שולט על הנחש שלו באמצעות 3 פעולות אפשריות: פנייה שמאלה, פנייה ימינה והתקדמות קדימה. בכל תור, כל השחקנים בוחרים את הפעולה הבאה לביצוע **במקביל**, ולאחר שבחרו, כל הנחשים מתקדמים צעד אחד בכיוון שבחרו **בו-זמנית**.

לוח המשחק הוא מלבן (מטריצה) בגודל w*h, כלומר ישנן w עמודות (שקובעות את רוחב המלבן) ו- h שורות (שקובעות את גובה המלבן). על גבי לוח המשחק, מפוזרים פירות אותם רוצים הנחשים לאכול.

כאשר נחש בוחר להתקדם למשבצת המכילה פרי, גופו מתארך ביחידה אחת, כאשר ההתארכות הינה מיידית. כלומר, גופו של הנחש נשאר באותן המשבצות שבהן היה בתור הקודם לאכילת הפרי, והמשבצת בה היה הפרי תעודכן להכיל את ראש הנחש.

כאשר נחש בוחר

1. להתקדם למשבצת המכילה גוף של נחש (אחר או של עצמו),
2. להתקדם למשבצת מחוץ לגבולות הלוח

הנחש מת, וכל משבצת השייכת לגופו הופכת לפרי שיכולים הנחשים האחרים לאכול. לאחר שנחש מת, השחקן ששלט בו ממתין לסיום המשחק.

אורך משחק הינו N תורות. כלומר, N סיבובים שבכל אחד בוחרים השחקנים (שאינם מתים) פעולה לביצוע. כעת נגדיר "הנחש הארוך בתור ה-n" הוא הנחש הארוך ביותר מבין הנחשים החיים בתור ה-n. במידה ויש יותר מנחש אחד ארוך ביותר בתור ה-n, שובר השוויון הוא הנחש שראשו קרוב ביותר לפרי (קרבה נמדדת במרחק מנהטן). במידה ויש יותר מנחש אחד כזה, ייבחר הנחש שראשו קרוב ביותר לפרי השני בקרבתו מבין נחשים אלו. בצורה דומה, ייפתרו שוויונות עד שיימצא הנחש שראשו קרוב ביותר לפרי ה-i בקרבתו מבין הנחשים השווים.
במידה וכל השחקנים מתו לפני התור ה-N, המשחק מסתיים בסוף התור האחרון בו היו שחקנים חיים.

מטרת כל שחקן היא להיות הנחש הארוך ביותר במשחק. בצורה מדויקת, הנחש הארוך ביותר במשחק הוא הנחש הארוך ביותר מבין קבוצת הנחשים שעונים להגדרה "הנחש הארוך בתור ה-n" לכל $n \leq N$, כאשר במקרה זה שובר השוויון הוא שמבין הנחשים באורך המקסימלי המנצח יהיה בעל ערך ה-n הגדול ביותר, כלומר זה ששרד הכי הרבה זמן בחיים.

חלק ב' – היכרות עם הקוד והמשחק

1. ראשית, הריצו את המשחק בעזרת שורת הקוד הבאה: Python main.py

שימו לב שזוהי הרצה אינטראקטיבית (כלומר, נותנת אפשרות למשתמש להחליט מהו המהלך הבא דרך המקלדת). אתם יכולים להשתמש בה על מנת להבין את המשחק טוב יותר (או סתם לשחק להנאתכם) אך בהמשך התרגיל לא נשתמש בהרצה זו אלה ניתן לסוכן, כלומר לקוד שתכתבו, להיות זה שמחליט על המהלכים הבאים. תוכלו לשלוט בנחש בעזרת המקלדת ע"י לחיצה על המקשים a כדי לפנות שמאלה ו- d כדי לפנות ימינה. במידה ואתם משתמשים במערכת הפעלה שאינה תומכת בקריאה אוטומטית של המקשים (למשל Mac OS), העבירו את הדגל kb_listener_off-- בקריאה לפיתון (כלומר הריצו את הפקודה python main.py --kb_listener_off).

- כל אחד מהיריבים שלכם במצב זה יהיה instance של השחקן הבסיסי שסופק לכם בתרגיל זה במחלקה GreedyAgent (הממומש בקובץ agents.py).

- דגלים נוספים שכדאי להכיר לקראת המשך התרגיל:

- הדגל n_agents=4-- מגדיר את מספר השחקנים במשחק (כולל אתם), בדוגמה זו, הוגדרו 4 שחקנים, אך תוכלו לשנות ערך זה. שימו לב, אם תכניסו יותר שחקנים ממה שהמשחק מאפשר (תלוי בגודל הלוח) הקריאה תיכשל. ברירת המחדל היא 2 שחקנים.
- הדגלים game_duration=50--, N 50– מגדירים את מספר התורות במשחק. בדוגמה זו משך משחק הוא 50 תורות.
- הדגל W 50– מגדיר שרוחב לוח המשחק הוא 50 משבצות.
- הדגל H 50– מגדיר שגובה לוח המשחק הוא 50 משבצות.
- הדגל n_fruits=50-- מגדיר שמספר הפירות המפוזרים על לוח המשחק הינו 50.
- הדגל kb_listener_off-- מכבה את קריאת התווים האוטומטית במקלדת ובהרצה אינטראקטיבית תתבקשו להכניס כל מהלך בנפרד בעזרת המקלדת.
- דגל part_c-- מריץ את חלק ג' של התרגיל.
- דגל part_d-- מריץ את חלק ד' של התרגיל.
- דגל part_e-- מריץ את חלק ה' של התרגיל.
- דגל fast_run--, f– מריץ במהירות מקסימלית. ברירת המחדל היא שלא להריץ במהירות מקסימלית על מנת שתוכלו לראות יותר בבירור את מהלך המשחק.
- דגל g-, graphics_off-- יכבה את הגרפיקה (דגל זה לא יעבוד במצב האינטראקטיבי).
- הדגל custom_game-- יאפשר לכם להגדיר משחק מותאם שבו תוכלו לבחור שני שחקנים שישחקו זה נגד זה. דגל זה יעזור לכם במיוחד בהכנת הסוכן שתגישו לתחרות הקורס. כאשר אתם מעבירים דגל זה, העבירו גם את שני הדגלים הבאים:
  - דגל p1-- ולאחריו את שם המחלקה הממשת את השחקן שתרצו שישחק בתור השחקן הראשון (הכחול). למשל p1 GreedyAgent--
  - דגל p2-- ולאחריו את שם המחלקה הממשת את השחקן שתרצו שישחק בתור השחקן השני (האדום). למשל p2 MinimaxAgent--

כדי שתוכלו לשחק נגד שחקן אחר לבחירתכם בעזרת המקלדת, העבירו עבור השחקן הראשון את הדגל p1 KeyboardPlayer--.

2. הסבירו בפירוט כיצד השחקן הבסיסי GreedyAgent (הממומש בקובץ agents.py) עובד ומהי היוריסטיקה בה הוא משתמש. כיצד יוריסטיקה זו עוזרת לשחקן להימנע ממוות?

חלק ג' – בניית סוכן משופר

אתם נדרשים לממש יוריסטיקה יותר מתוחכמת מזו הפשוטה של השחקן GreedyAgent . על היוריסטיקה להיות ממומשת בפונקציה heuristic בקובץ submission.py. פונקציה זו תגדיר את היוריסטיקה של השחקן BetterGreedyAgent המוגדר בקובץ agents.py. שימו לב, אתם יכולים להוסיף פונ' עזר נוספות במידת הצורך (בקובץ submission.py), אך אין לדרוס את הפונקציות הקיימות או לשנות את חתימות המתודות הקיימות. לידיעתכם, היוריסטיקה שתגדירו ותממשו בחלק זה תשמש אתכם גם בהמשך התרגיל.

1. הגדירו באופן מפורש יוריסטיקה משלכם להערכת מצבי המשחק. על היוריסטיקה להכיל **לפחות** 3 פרמטרים שונים (אפשר ורצוי למצוא יותר). שימו לב כי היוריסטיקה צריכה לעבוד לכל מספר של יריבים ולכל גודל של לוח. הציגו בצורה פורמלית את ההגדרה במסמך שלכם.

2. הסבירו את המוטיבציה להגדרה מהסעיף הקודם (ולכל פרמטר המופיע בחישובים שלכם). למה אתם צופים שהיא תשפר את ביצועי השחקן BetterGreedyAgent ביחס ליוריסטיקה בה השתמש השחקן הפשוט GreedyAgent?

3. ממשו את היוריסטיקה שהגדרתם בפונקציה heuristic בקובץ submission.py. על מנת להריץ את המשחק בחלק זה השתמשו בדגל part_c--. הפקודה שלכם צריכה להתחיל כך:
python main.py --part_c <other options here…>

טיפים לפני מימוש: עברו על המחלקה GameState בקובץ environment.py כדי להבין באילו פונקציות אתם יכולים להשתמש כדי לקבל מידע על מצב המשחק. שימו לב- אין לשנות את הקוד בקובץ environment.py, אך ניתן (ואף מומלץ) להשתמש בפונקציות של המחלקות GameState, SnakeAgent ו-Player הקיימות בו. דרכן תוכלו להשיג את כל המידע שתרצו על מצב הלוח הנוכחי באופן פשוט יחסית. כמו כן שימו לב שבקובץ זה מוגדרת פונקציה בשם get_next_state בה תוכלו להשתמש (פונקציה זו כבר יובאה לקובץ submission.py עבורכם).
שימו לב, בפייתון (ובשפות אחרות) כשעובדים עם מחלקות (classes) נהוג להגדיר מתודות עם גישה public עם שם שמתחיל באות, מתודות עם גישה protected עם שם שמתחיל בקו תחתון יחיד, ולבסוף מתודות עם גישת private עם שם שמתחיל בשני קווים תחתונים. אינם מחויבים לקונבנציות אלו, אך הן יעזרו לכם ולאחרים שמשתמשים בקוד שלכם. כאשר אתם מגדירים פונקציות עזר שאינן מתוכננות להיות חלק מהממשק הייעודי של המחלקה, העדיפו להגדיר אותן בגישה חלקית protected או private.

חלק ד' – בניית סוכן Minimax

נרצה לממש סוכן RB Minimax ב class בשם MinimaxAgent אשר בקובץ submission.py.
שימו לב כי בניגוד למשחקים שדיברנו עליהם בכיתה, שם היה לכל שחקן יריב אחד, במשחק זה יכולים להיות מספר יריבים (נחשים אדומים).

לכן, נרצה להרחיב את אלגוריתם המינימקס שלמדנו בכיתה (שהיה לו רק שלב מינימלי אחד עבור יריב יחיד) למקרה הכללי יותר של יריבים מרובים. נגדיר שבמקרה הכללי שבו ישנם k יריבים, בהגעה לצומת מינימום נפתח צומת בן עבור כל וקטור $(a_1, a_2, \dots, a_k)$ המתאר בחירה ספציפית של פעולת כל אחד מהיריבים.
כלומר, עבור מקרה בו היריב הראשון בוחר בפעולה a_1, היריב השני בוחר בפעולה a_2 וכן הלאה, יפותח צומת בן שיתאר את מצב המשחק לאחר בחירת פעולות אלו (ובחירת הפעולה של הסוכן שלנו שגרמה להגעה לצומת מינימום זה).
שימו לב שעל מנת להשתמש באלגוריתם מינימקס כשפי שנלמד, המשחק צריך להתנהל בתורות, כאשר לאחר שסוכן בוחר פעולה היא מבוצעת ומצב המשחק משתנה ומכיל מידע על הסוכן הבא בתורו לשחק. מכיוון שבמשחק שלנו כל השחקנים בוחרים פעולה הבאה לביצוע במקביל, עלינו לבנות מודל של מצב המשחק המתאים לשימוש באלגוריתם המינימקס. לצורך כך, נרצה בתחילת חישוב עץ האסטרטגיה להגדיר מצב פנימי מסוג חדש שישמור את הפעולה שהסוכן שלנו בחר באותו מצב בנוסף למצב עצמו שקיבלנו בקריאה get_action (שמתאר את מצב המשחק). הגדרת מצב פנימי זה נתונה לכם בקוד המחלקה MinimaxAgent

ואנו ממליצים לכם להשתמש בה במימוש שלכם. לפי מודל זה, חיפוש בעץ האסטרטגיה תמיד יסתיים בבנים של צמתי מינימום. כלומר, כל העמקה נוספת תכלול פיתוח של צומת מקסימום וצומתי המינימום העוקבים לו. עומק החיפוש יוגדר כמספר צמתי המינימום במסלול מהשורש לעלה.

שימו לב שהסבר זה מתייחס גם לשאלות הקשורות לאלגוריתם -Alpha Beta Minimax בהמשך התרגיל.

1. ממשו את האלגוריתם בגרסה מוגבלת המשאבים כפי שהוצג ביחד עם היוריסטיקה שיצרתם בחלק ג'. וודאו שהשחקן החדש שיצרתם רץ ללא תקלות. על מנת להריץ את הקוד של חלק זה השתמשו בדגל part_d--, כלומר הריצו את הפקודה
python main.py --part_d <other options here…>

בעת המימוש שימו לב לפרטים הבאים (רלוונטי גם לסעיפים הבאים):
- ודאו כי המימוש שלכם עובד עבור מספר נחשים משתנה.
- אתם רשאים לממש כל פונקציית עזר שתרצו בתוך הclass של הסוכן, אך אתם חייבים לממש את הפונקציה get_action ואין לשנות את חתימתה.

2. האם אתם צופים שאלגוריתם Minimax שכתבתם יעבוד טוב? אם כן, באיזה תרחישים? מדוע? פרטו בהרחבה.

3. בהינתן שמספר הנחשים היריבים הוא k, כמה בנים יהיו לכל צומת מינימום (כפי שהגדרנו למעלה) בהנחה שכל היריבים בחיים? האם ניתן בפועל להשתמש באלגוריתם זה עבור מספר רב (20+) של שחקנים? מדוע?

4. באלגוריתם Minimax כאשר יש יותר מיריב אחד, איזו הנחה אנו מניחים על פעולות שאר היריבים כאשר אנחנו בוחרים בפעולה הבאה שאנו מעוניינים לבצע? האם במשחק שלנו הנחה זו נכונה? הסבירו.

5. נרצה לחשוב על דרך נוספת לממש את אלגוריתם Minimax, כך שניצור שכבה נפרדת לכל אחד מהנחשים היריבים במקום שכבה אחת לכולם יחד כפי שתואר למעלה.

   a. הסבירו (במילים, אין צורך לממש) איך הייתם עושים זאת. מה החסרונות והיתרונות של כל אחת מהשיטות?

   b. מהי ההנחה שלנו (אחת או יותר) בהגדרת המצב בעץ האסטרטגיה כמתואר מעלה בהשוואה לדרך הנוספת שהוצגה, בנוגע לסדר קבלת ההחלטות של הסוכנים במשחק? האם היא בהכרח נכונה? הרחיבו.

חלק ה' – בניית סוכן Alpha-Beta

1. ממשו את אלגוריתם אלפא בטא (מוגבל משאבים) בclass המתאים (AlphaBetaAgent) בקובץ submission.py. כאמור, אתם נדרשים לממש את המתודה get_action, שאת חתימתה אין לשנות, אך תוכלו להוסיף פונקציות עזר כרצונכם. על מנת להריץ את הקוד של חלק זה השתמשו בדגל
part_e--, כלומר הריצו את הפקודה
python main.py --part_e <other options here…>

2. האם הסוכן אלפא בטא שמימשתם בסעיף זה יתנהג שונה מסוכן הMinimax שממשתם בסעיף הקודם:

a. מבחינת זמן ריצה? הסבירו מדוע.
b. מבחינת בחירת מהלכים? הסבירו מדוע.

חלק ו' – חיפוש מדיניות בעזרת אלגוריתמי חיפוש מקומי

בחלק זה נרצה לבנות סוכן המשתמש במדיניות קבועה מראש ל N הצעדים במשחק. סוכן זה ישמור מערך בגודל N של פעולות, ובתור ה- i יבצע את הפעולה השמורה במערך במקום ה- i. כמו כן, נרצה להשתמש באלגוריתם חיפוש מקומי על מנת למצוא את סדרת N הפעולות הטובות ביותר עבור סוכן זה. בחלק זה דאגנו לכך שבכל הרצה של המשחק נקודות ההתחלה של השחקנים ופעולות שאר השחקנים במהלך המשחק יהיו דטרמיניסטיים וזהים. בנוסף, מספר היריבים בחלק זה יהיה גדול ויכלול 20 יריבים. שימו לב, בחלק זה היריבים לא יהיו מהמחלקה GreedyAgent אלא יבחרו באקראי רצף פעולות ויבצעו את אותו הרצף בכל הרצה של המשחק. נגדיר שפונקציית התועלת במשחק שלנו עבור כל שחקן ומצב סופי תחזיר לשחקן את אורכו + נקודה נוספת אם הוא סיים את המשחק בחיים.
שימו לב, חלק זה בעל אופי מחקרי יותר, ולכן אנו מעודדים אתכם לנסות רעיונות חדשים, והניקוד יינתן על פי איכות הצגת התוצאות, הגדרת האלגוריתם, ביצוע הניסויים וכד' ולא ירד ניקוד אם תוצאות הניסויים שלכם אינן מוצלחות כמו שקיוויתם (כלומר אם ניקוד הסוכן במשחק נמוך).

1. הסבירו מדוע היינו צריכים להניח שמהלך כל משחק יהיה זהה (מבחינת נק' ההתחלה של השחקנים והמהלכים של היריבים) על מנת לאפשר חיפוש מדיניות בעזרת חיפוש מקומי? באיזו בעיה הינו נתקלים אם הינו מנסים להשתמש באלג' אלו ללא הנחה זו?

2. האם משחק זה הוא משחק סכום אפס? מדוע?

בתור התחלה, נשתמש באלגוריתם Steepest Ascent Hill Climbing with Sideways Steps. כזכור, באלגוריתם זה אנו מחזיקים מצב יחיד ובכל איטרציה מתקדמים למצב עם הערך הטוב ביותר שמשפר, ומתירים להתקדם למצבים עם ערך זהה (לא משפר אך גם לא גרוע יותר).
נגדיר שבאיטרציה ה-i בחיפוש ננסה לבדוק את כל האופציות עבור הפעולה ה-i במערך הצעדים של הסוכן שלנו.

3. מהם המצבים במרחב החיפוש שלנו? הגדירו במדויק.

4. מהם האופרטורים במרחב החיפוש שלנו? בכמה אופרטורים אנו משתמשים בכל איטרציה?

5. כמה איטרציות נצטרך לבצע בחיפוש זה?

6. כפי שלמדתם בהרצאה, בחיפוש לוקאלי אנו משתמשים בפונ' יוריסטית להערכת טיב המצב בו אנו נמצאים. איזו פונ' לדעתם תתאים להערכת טיב מצב? כיצד היא תעבוד? הסבירו.

7. בחרו מצב התחלתי באיזו דרך שתרצו. תוכלו לקבוע מצב בצורה דטרמניסטית (חישבו על מצבים מעניינים שתוכלו לבחור בדרך זו), או שתוכלו להגריל מצב באקראי (תוכלו להשתמש בהתפלגות החביבה עליכם). אם תבחרו להגריל באקראי, תוכלו להשתמש בפונקציות שבספרייה numpy (ספריה זו יובאה עבורכם בקובץ submission.py). למשל np.random.choice (עוד מידע יש בלינק הזה).
מאיזה מצב בחרתם להתחיל? מדוע? הסבירו את הבחירה שלכם.

8. כאשר אנחנו בוחרים לשנות את הפעולה ה-i בסדרת הפעולות של הסוכן, על מה שינוי זה משפיע? הסבירו בפירוט. התייחסו בפרט למצבים שבאו לפני הפעולה ולמצבים שיגיעו לאחר הפעולה. על כמה מצבים לכל היותר שינוי זה ישפיע?

9. ממשו את החיפוש המקומי כפי שמוגדר למעלה בפונקציה SAHC_sideways שבקובץ submission.py עבור N=50. לשימושכם, יובאה לקובץ זה הפונקציה get_fitness שמוגדרת בקובץ utils.py. פונקציה זו מקבלת רצף מהלכים עבור הסוכן ומחזירה את ערך פונקציית התועלת עבור השחקן בסוף המשחק שבו השתמש השחקן ברצף המהלכים הנתון.
שימו לב, בפונקציה SAHC_sideways עליכם לכתוב את כל מהלך החיפוש, כולל בחירת המצב ההתחלתי, והרצת אלגוריתם החיפוש כך שבסוף הקריאה לפונקציה יודפס רצף המהלכים הטוב ביותר שמצאתם.

10. הריצו את החיפוש ורשמו את ווקטור הפעולות הטוב ביותר שקיבלתם. לאיזה אורך הגיע הנחש שלכם? האם הוא ניצח?

11. כעת, בחרו באלגוריתם חיפוש מקומי אחר מהתרגולים וההרצאות שלדעתם יעבוד טוב על בעיה זו. ציינו באיזה אלגוריתם בחרתם והסבירו מדוע. מדוע אתם חושבים שאלגוריתם זה יעבוד עבור הבעיה שלנו?
תוכלו לבצע התאמות ושינויים באלגוריתם כרצונכם (זהו החלק היצירתי), אך אם ביצעתם שינויים כאלה, הסבירו בפירוט מה הם.

12. הגדירו במדויק מהם המצבים והאופרטורים במרחב החיפוש שלכם.

13. ממשו את האלגוריתם שתיארתם בסעיף 11 בפונקציה local_search שבקובץ submission.py.

14. רשמו את ווקטור הפעולות שקיבלתם. לאיזה אורך הגיע הסוכן שלכם? האם הוא ניצח? כמה איטרציות הייתם צריכים לבצע על מנת להגיע לפתרון זה?

חלק ז' – ניסויים, תוצאות ומסקנות.

נרצה להשוות בין השחקנים השונים שראינו עד כה ביחד עם היוריסטיקות השונות, תחת מגבלות זמן שונות לוחות משחק ומספר נחשים משתנים. בפועל, ההשוואה תיעשה בין 4 שחקנים:

(1) GreedyAgent המסופק
(2) BetterGreedyAgent
(3) MinimaxAgent
(4) AlphaBetaAgent

נסמן $D = \{2,4,6\}$. עבור כל שחקן (סה"כ 4) הריצו סדרה של 10 משחקים עבור כל מגבלת עומק $d \in D$. בכל מגבלה של d, התוצאה הסופית היא ממוצע ה score של 10 המשחקים שהרצתם. שימו לב שעבור השחקנים 1 ו2 אין משמעות לפרמטר העומק כי השחקנים אינם מסתכלים לעומק הגדול מ1, ולכן עבורם הריצו רק עבור d=1.

אנחנו נתעניין במצב שבו ישנם 2 נחשים בלבד (אויב יחיד) ועליכם לקבע את הארגומנט עבור כל הריצות.

לסיכום, כל תת ניסוי מוגדר ע"י:

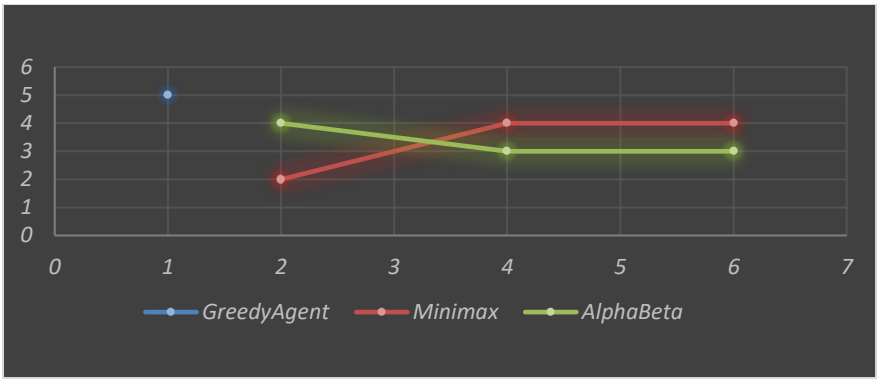
בחירת שחקן, בחירת מגבלת עומק ולאחר מכן חישוב ממוצע ה score של 10 הרצות.

סה"כ עליהם להריץ: $2_{players} \cdot 10_{\text{games}} + 3_{players} \cdot 10_{\text{games}} \cdot 3_{d \in D}$ משחקים.

בנוסף לנתונים אלה, נרצה להוסיף את הזמן הממוצע שלוקח לשחקן לשחק תור, לשם כך שמרו את ההפרש בין הזמן בתחילת הפונקציה get_action ובסופה בכל תור, והדפיסו את הממוצע בסוף המשחק.

1. יש להגיש קובץ בשם $experiments.csv$ שיכיל את תוצאות המשחקים שהרצתם. כל שורה בקובץ תכיל 4 עמודות: שם שחקן, מגבלת העומק, ניקוד השחקן, ממוצע הזמן לחישוב הפעולה get_action.
לנוחיותכם מסופקת שורה לדוגמא:
GreedyAgent, 2, 6, 1.23
השם GreedyAgent מתאר את סוג השחקן, המספר 2 מתאר את מגבלת העומק, המספר 6 מתאר את הניקוד של השחקן (ממוצע), והמספר 1.23 מתאר את ממוצע הזמן לקריאה ל-get_action.
אין לכלול כותרות בקובץ, אלא רק את תוצאות הניסויים.

2. הצגת התוצאות:
   I. הציגו גרף המתאר עבור כל אחת ממגבלות העומקים את <u>הניקוד הכולל של כל שחקן</u>.
הקפידו להשתמש בצבעים או בסוגי קו שונים עבור שחקנים שונים. לנוחיותכם מצורף גרף לדוגמא (*).



* שימו לב כי הגרף חלקי- לא מכיל את כל הסוכנים

** שימו לב כי עבור GreedyAgent הגרף הוא נקודה מכיוון שרק ערך D אחד רלוונטי עבורו.

II. צרפו את טבלת הנתונים שיצרו אותו. למשל(*):

	$d=6$	$d=4$	$d=2$	$d=1$
$GreedyAgent$				1
$BetterGreedyAgent$				4
$MinimaxAgent$	4	3	3	
$AlphaBetaAgent$	1	2.5	4	

(*) שימו לב כי הגרפים והטבלאות שהצגנו כאן נוצרו לצורך **המחשה בלבד**.

3. מהן המסקנות שניתן להסיק מהגרף והטבלה מסעיף 2? האם התוצאות שקיבלתם תואמות את ציפיותיכם?

4. צרפו את אותו גרף ואותה טבלה כמו בסעיף 2, רק שהפעם הטבלה היא של הזמן הממוצע שלקח כל תור כפונקציה של d. כלומר, עבור כל שחקן וכל אחת ממגבלות העומקים הציגו את <u>הזמן הממוצע שלקח תור</u>.

5. מהן המסקנות שניתן להסיק מהגרף והטבלה מסעיף 4? האם התוצאות שקיבלתם תואמות את ציפיותיכם?

6. השוו את ביצועי השחקנים ביחס למגבלת העומק, לזמן הממוצע לתור, ושאר הפרמטרים שראיתם בחלק הניסויים.
תוכלו לבצע ניתוח של מגמות השיפור והדעיכה, לבחון את הניקוד הכולל של כל אחד מהשחקנים.
שאלות שתוכלו לבחון:
באילו מקרים היוריסטיקה שלכם מתגברת על היוריסטיקה הפשוטה אם בכלל?
מי השחקן הטוב ביותר בכל לוח? איך הלוחות משפיעים על השחקנים השונים?
האם מגבלות העומק השפיעו על התוצאות? מה היה קורה אילו היו מגבלות זמן במקום מגבלות עומק?
הסבירו <u>בהרחבה</u> את כל המאפיינים שראיתם בתרגיל ובצעו ניתוח מעמיק בין זוגות השחקנים.
שימו לב, זהו סעיף סיכום התרגיל ומשקלו בציון משמעותי.

חלק ח' – תחרות בקורס

לסיום, עליכם להגיש שחקן Multi-Player Snake לתחרות הקורסית. עליכם לממש סוכן זה במחלקה TournamentAgent שבקובץ submission.py. ספציפית, ממשו את הפוקציה get_action מבלי לשנות את חתימתה. תוכלו להוסיף פונקציות עזר כרצונכם. זכרו, על מנת לראות את ביצועי השחקן שלכם תוכלו

להשתמש בהרצה המיוחדת עם הדגל custom_game-- ולהעביר את הדגל TournamentAgent p1--. (או עם הדגל p2--, במצב זה השחקן שלכם יהיה הנחש האדום).

שימו לב, סעיף זה ייתן נקודות בונוס לציון ה**סופי של הקורס** באופן הבא: הזוג שיזכה במקום הראשון בתחרות יקבל 5 נקודות בונוס לציון הסופי, הזוג שיזכה במקום השני בתחרות יקבל 3 נקודות בונוס לציון הסופי והזוג שיזכה במקום השלישי בתחרות יקבל 2 נקודות בונוס לציון הסופי.

ההשתתפות בתחרות היא **חובה**. לא יורדו נקודות על שחקן שביצועיו לא טובים אך יורדו נקודות אם השחקן שתגישו לא ירוץ/ לא תגישו את ההסבר המילולי הנדרש.

הגישו הסבר קצר בדו"ח שאתם מגישים המתאר את שחקן התחרות שלכם. כתבו איך הוא עובד ומדוע בחרתם בשיטת פעולה זו.

התחרות תערך כך:

אנו נבנה טורניר, שבו יתחרו הנחשים זה בזה. בכל משחק בטורניר ישחקו שני נחשים מתחרים זה נגד זה. שימוש לב, הנחש שתגישו ישחק כנגד נחשים שסטודנטים אחרים הגישו ולא נגד הנחש GreedyAgent נגדו שיחקתם במהלך התרגיל! הנחשים שיגיעו לשלושת המקומות הראשונים בטורניר יקבלו את הבונוס שהוגדר למעלה. כל משחק יורץ בלוח שגודלו 25 על 25, מספר הפירות שיפוזרו על הלוח יהיה 40, והמשחק ירוץ למשך 500 תורות (כלומר N=500). לכל נחש תהיה מגבלת זמן של דקה לכל ההחלטות שיקבל <u>במהלך כל המשחק</u>. שימו לב, תצטרכו במימוש השחקן לקבוע מדיניות שתעזור לשחקן להשתמש בזמן שעומד לרשותו בצורה חכמה! מכיוון שזמן הריצה יכול להשתנות בין הרצה על המחשב שלכם לבין הרצה על מחשב אחר, תצטרכו לקבוע מדיניות שתמדוד את הזמן שנוצל ותקבל החלטות בהתאם, ללא תלות במחשב עליו רץ הסוכן.

אתם יכולים להשתמש בכל אלגוריתם בו השתמשתם במהלך התרגיל ו/או לפתח אלגוריתמים חדשים כרצונכם, כל עוד הם לא ניגשים לאינטרנט וירוצו בסביבה מבודדת עליה אנחנו נבדוק.

הוראות הגשה

- הגשת התרגיל תתבצע **אלקטרונית בלבד** דרך אתר הקורס.
- אתם מתבקשים ליצור קובץ zip בשם $AI2\_\langle id1\rangle\_\langle id2\rangle$ (ללא הסוגריים המשולשים), שבתוכו הקבצים הבאים:
  - קובץ הקוד submission.py שימו לב, זה קובץ הקוד היחיד שאתם מגישים ולכן מומלץ לוודא שהוא רץ עם הקבצים המקוריים שקיבלתם מאתנו, כלומר ללא שינויים שאולי עשיתם בטעות בקבצים האחרים.
  - קובץ בשם $AI\_HW2.pdf$, המכיל את התשובות לחלק היבש
  - קובץ בשם $experiments.csv$ כפי שמתואר בסעיף ח חלק 1.

- **בעיות קומפילציה יגררו לירידה של עד 5 נקודות בציון התרגיל.**

- אין להעתיק את הקבצים המסופקים לכם אל תוך תיקיית ההגשה. הניחו כי קבצים אלו יהיו זמינים בעת בדיקת התרגיל.

- שימו לב שכל הפנייה למיקום קובץ/תיקייה כלשהם בקוד תהיה רלטיבית ($relative\ path$) ולא אבסולוטית, כך שהקוד יעבוד כפי שהוא על כל מחשב בכל מיקום שנבחר לתיקיית הפרויקט. הקפידו לבדוק זאת לפני ההגשה!
- הקפידו על קוד **ברור, קריא ומתועד!** עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם.
- "המצאת" נתונים לצורך בניית הגרפים **אסורה** ותוביל לדיון בבית הדין המשמעתי של הטכניון.

בהצלחה!