

תרגיל בית 2 - חיפוש רב סוכני

Keep On Moving

מטרת התרגיל

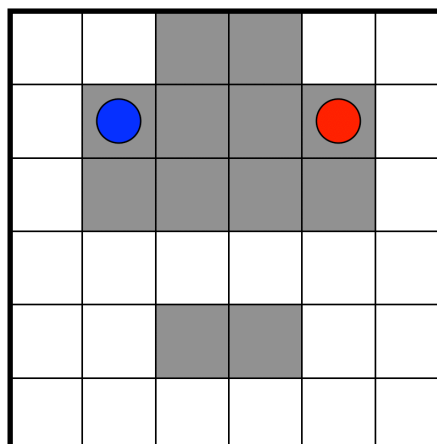
בתרגיל זה נממש סוכני חיפוש בסביבה מרובת סוכנים. נממש את הסוכן על פי אלגוריתם *Minimax* ושיפוריו, בווריאצית Anytime Contract, עבור המשחק Keep On Moving. נתנסה במימוש היוריסטיקה עבור המשחק, ביצוע ניסויים וכתובת דו"ח.

הערות

- תאריך הגשה: יום שני, 08.06.2020, עד השעה 23:59.
- יש להגיש את המטלה בזוגות בלבד.
- יש להגיש דוחות מוקלדים בלבד.
- המתרגל אחראי על התרגיל: טל סויסה.
- ניתן לשלוח שאלות בנוגע לתרגיל לתיבת המייל בכתובת: talswisa@cs.technion.ac.il, לא לפני שבדקתם האם קיימת תשובה לשאלה ב- FAQ באתר הקורס.
- שימו לב שזו לא כתובת המייל הקורסית כמו בתרגיל הראשון.
- ניתן לשאול שאלות על התרגיל בשעת הקבלה של טל.
- בקשות דחיה מוצדקות יש לשלוח למתרגל האחראי של הקורס - גיא קושלביץ.
- ייתכן שנעלה עדכונים והבהרות לדף FAQ, העדכונים מחייבים.
- העתקות תטופלנה בחומרה.

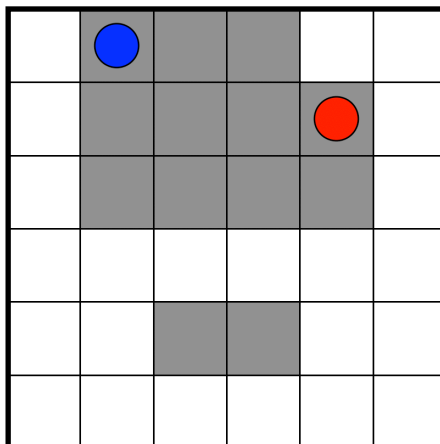
תיאור המשחק

Keep On Moving הוא משחק לוח לשני שחקנים. המשחק מתחיל כאשר לכל שחקן מיקום התחלתי על לוח מלבני המורכב ממשבצות אפורות ולבנות. משבצות אפורות הן משבצות שאסור לנוע אליהן. כל שחקן בתורו עובר למשבצת לבנה לפי הצעדים החוקיים במשחק, וזו הופכת לאפורה, כלומר אף שחקן לא יכול לדרוך עליה שוב. אם בתורו שחקן לא יכול לנוע לאף משבצת, הוא הפסיד, אלא אם כן גם השחקן השני לא יכול לנוע, ואז המשחק נגמר בתיקו. מטרת המשחק היא לא להתקע, כלומר לא להיות השחקן הראשון שלא יכול לנוע. המהלכים החוקיים בגרסת המשחק שלנו הם למעלה, למטה, שמאלה וימינה. לדוגמה, יתכן שהלוח ההתחלתי יהיה הלוח הבא:

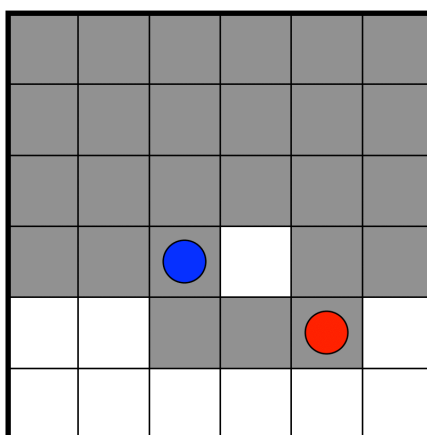


הנקודות הכחולה והאדומה מייצגות את מיקום השחקנים על הלוח. השחקן הכחול הוא השחקן שמשחק ראשון, והוא יכול לנוע שמאלה או למעלה.

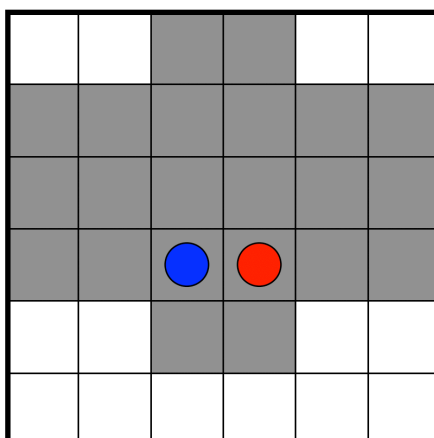
לאחר שהוא ינוע למעלה הלוח יראה כך:



המשבצת שהוא נע אליה הפכה לאפורה, ואי אפשר יהיה לדרוך עליה שוב. בהמשך המשחק יתכן והמשחק יגיע למצב הבא, כאשר תורו של השחקן הכחול לנוע.



השחקן הכחול חייב לנוע ימינה, ולאחר מכן השחקן האדום יוכל לנוע למטה או ימינה. לאחר שהשחקן האדום יבצע את תורו, השחקן הכחול יפסיד מכיוון שהוא לא יוכל לנוע. כאמור במשחק יתכן גם מצב תיקו: כאשר שני השחקנים לא יכולים לנוע, למשל במצב הבא:



הרצת משחקים

לפני תחילת מימוש הסוכנים, מומלץ להתנסות במשחק. הריצו את השורות הבאות ב- terminal בתקייה בה נמצא הקוד המצורף לתרגיל:

כדי לשחק בתור שני הסוכנים, הריצו את השורה:

```
python LiveAnimatedGame.py LivePlayer LivePlayer
```

* אם זה לא עובד או שיש תקלה במהלך הריצה, תפעלו לפי ההנחיות בקובץ "animation running guide".

כדי לשחק נגד שחקן SimplePlayer (שחקן שהמימוש שלו נמצא בקבצי הקוד שקיבלתם) הריצו:

```
python LiveAnimatedGame.py LivePlayer SimplePlayer
```

לאחר הרצת השורות תפתח אנימציה המציגה את מצב המשחק. כדי לשלוט ב-LivePlayer, נשתמש במקשים w (למעלה), s (למטה), a (שמאלה), ו-d (ימינה). יש להקליד את התו ב-terminal, ולאחר מכן להקיש Enter. במידה והצעד שבחרתם לא חוקי (מחוץ ללוח, אל משבצת אפורה, או לא אחד מהתווים החוקיים: w, s, a, d), תוכלו לבחור שוב צעד.

ניתן להריץ את המשחקים עם הדפסות בטרמינל במקום אנימציה על ידי הרצת השורה:

```
python NotAnimatedGame.py LivePlayer SimplePlayer
```

ולהריץ בלי הדפסות של המשחק (תודפס רק הכרזת מנצח בסוף המשחק) על ידי הרצת השורה האחרונה בתוספת הדגל -dont_print_game. אם חסר לכם הקובץ NotAnimatedGame.py, הורידו שוב את קבצי הקוד של התרגיל.

בהמשך נממש את השחקנים AlphaBetaPlayer, MinimaxPlayer ו-ContestPlayer ועוד. כדי להריץ משחק בין שני שחקנים כלשהם הריצו שורה דומה עם השמות של השחקנים המתאימים. בנוסף אתם יכולים לבחור לוח מבין חמישה לוחות על ידי הוספת הדגל -board i עבור i בין 0 ל-4, ולבחור את מגבלת הזמן עבור הפונקציה make_move (תוגדר בהמשך) עם הדגל -move_time t כשאר t הוא מספר השניות. בנוסף ניתן לשנות את מהירות המשחק על ידי שינוי הפרמטר interval בשורה 65 ב-LiveAnimation.py. משמעות הפרמטר היא משך זמן הופעת כל פריים באנימציה.



חלק א' - חלק יבש (40 נק')

בחלק זה עליכם לענות על שאלות, ולצרף את התשובות לדו"ח.

שאלות כלליות על המשחק:

נגדיר מצב במשחק על ידי מטריצה $s \in \{-1, 0, 1, 2\}^{n \times m}$:
הערך 0 מסמל משבצת ריקה.
הערך -1 מסמל משבצת אפורה.
הערך 1 מסמל את מיקומו הנוכחי של השחקן שלכם.
הערך 2 מסמל את מיקומו הנוכחי של השחקן היריב.

1. (5 נק') הגדירו פורמלית את פונקציית העוקב $Succ : S \times \{1, 2\} \rightarrow 2^S$.
2. (5 נק') הגדירו פורמלית תנאים הכרחיים ומספיקים על מצב $s \in S$, עבור המקרים:
 1. s הוא מצב נצחון עבור שחקן i .
 2. s הוא מצב תיקו.
3. (2 נק') מהו מקדם הסיעוף במשחק?

שאלות על SimplePlayer:

בקובץ `SimplePlayer.py` שמצורף לתרגיל נמצא מימוש של השחקן. השאלות הבאות נוגעות אליו, לכן יש להבין את אופן פעולתו (על ידי קריאת הקוד, והרצה של משחקים איתו) לפני שניתן לענות על השאלות:

4. (3 נק') מהי האסטרטגיה של השחקן?
מנו מספר יתרונות וחסרונות של אסטרטגיה זו.
5. (3 נק') הציגו לוח שבו השחקן `SimplePlayer` יפעל בהכרח בצורה אופטימלית, והסבירו.

שאלות על הגדרת היוריסטיקה:

6. (3 נק') נסתכל על ההיוריסטיקה הבאה עבור שחקן `Minimax`:
 $h(s) = |Succ(s, i)|$, כאשר i הוא מספר סוכן המינימקס עבורו מחושב ערך ההיוריסטיקה. כלומר הערך היוריסטי של מצב הוא מספר המשבצות הלבנות השכנות למקומו של הסוכן.
מה החסרון בהיוריסטיקה זו?
7. (3 נק') הגדירו היוריסטיקה המורכבת לפחות משלושה מרכיבים שונים (אותו מרכיב ביחס ליריב אינו נחשב למרכיב נוסף). נמקו מדוע בחרתם במרכיבים אלו.

שאלות על Minimax בווריאצית Anytime Contract:

8. (3 נק') הסבירו מהי ווריאציות Anytime Contract של אלגוריתם `Minimax`, מהי העמקה הדרגתית בהקשר זה, ומה הבעיה הנוגעת להעמקה הדרגתית המוצגת בהרצאה.
9. (3 נק') הסבירו את הפתרון המוצע בהרצאה לבעיה זו.
10. (6 נק') הניחו משחק שבו לכל מהלך של שחקן נתונה מסגרת זמן (`time_limit`), שחקן שלא מבצע את המהלך עד תום מסגרת הזמן מפסיד במשחק, ושחקן שמחליט על הצעד שהוא מבצע לפני סיום הזמן הנתון מקבל תוספת זמן לתור הבא.
במשחק כזה יתכן ונרצה לבצע בכל תור את המהלך הכי מידע שאנחנו יכולים (כלומר המהלך שנבחר על ידי `Minimax` לעומק מירבי שנצליח להגיע אליו במסגרת הזמן).
לשם כך, לאחר ביצוע `Minimax` לעומק d , נרצה לדעת האם נספיק להפעיל `Minimax` לעומק $d + 1$: אם כן, נפעיל את ה-`Minimax` לעומק $d + 1$ כדי לבצע צעד מידע יותר, ואם לא, נרצה להחזיר תשובה כעת, ולא להתחיל איטרציית `Minimax` לעומק $d + 1$, כדי לקבל תוספת זמן לתור הבא.
לשם כך נרצה לחסום מלמעלה את זמן הריצה של ה-`Minimax` לעומק $d + 1$.
האלגוריתם הבא מתאר את ההתנהגות הרצויה כפי שתיארנו:

```
1 ID_start_time = time()
2 d = 1
3 move, l = Minimax(s, d)
4 last_iteration_time = time() - ID_start_time
5 next_iteration_max_time = f(l, last_iteration_time)
6 time_until_now = time() - ID_start_time
7 While time_until_now + next_iteration_max_time < time_limit:
8     d += 1
9     iteration_start_time = time()
10    move, l = Minimax(s, d)
11    last_iteration_time = time() - iteration_start_time
12    next_iteration_max_time = f(l, last_iteration_time)
13    time_until_now = time() - ID_start_time
14 return move
```

המשתנה `time_limit` הוא מספר השניות המוקצב עבור החזרת הצעד של הסוכן, הפונקציה `time` מחזירה את הזמן הנוכחי, הערך l המוחזר מהפונקציה `Minimax` הוא מספר העלים בעץ המינימקס שפותח, ו- s הוא המצב הנוכחי.

הגדירו את הפונקציה f כך שזמן הריצה לא יחרוג מהזמן המוקצב, כלומר מצאו חסם עליון לזמן הריצה של האיטרציה הבאה כפונקציה של מספר העלים שפותחו בקריאה הקודמת ל-*Minimax*, ושל זמן הריצה של האיטרציה הקודמת.
על החסם להיות הדוק ככל הניתן. הסבירו.

השוואה בין סוכן ה-*Minimax* לסוכן ה-*Alpha – Beta*

11. (2 נק') מי מבין הסוכנים צפוי להיות יותר טוב בווריאצית Anytime Contract?
12. (2 נק') האם התשובה הייתה זהה, אם במקום מגבלת זמן, ההגבלה הייתה על עומק החיפוש?

חלק ב' - מימוש סוכן *Minimax* (15 נק')

בחלק זה נממש סוכן *Minimax* מוגבל משאבים עבור המשחק.
את השחקן עליכם לממש בקובץ ששמו *MinimaxPlayer.py*, במחלקה (class) ששמה *MinimaxPlayer*.
השחקן יממש את אלגוריתם *Minimax* בווריאצית Anytime Contract.
על השחקן להיבנות על ידי השורה:

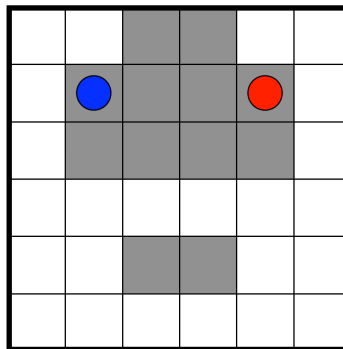
```
minimax_player = MinimaxPlayer()
```

לשם כך פונקציית `__init__` צריכה להיות חסרת פרמטרים, או בעלת ערכי ברירת מחדל לכל הפרמטרים.
על השחקן שתבנו לממש את הפונקציות `set_game_params`, `make_move` ו-`set_rival_move`:

`set_game_params(self, board)` - פונקציה זו תיקרא (לא על ידכם, אלא על ידי הקוד שמנהל את המשחק) פעם אחת עבור כל שחקן, בתחילת המשחק, ומטרתה להעביר לשחקנים את הלוח שבו יתנהל המשחק.

פרמטרים:

`board` - הלוח מיוצג על ידי רשימה של רשימות. כל רשימה מייצגת שורה בלוח, כלומר לכל $0 \leq i \leq \text{len}(\text{board})$, `board[i]` היא שורה, ו-`board[i][j]` היא משבצת.
בתמונה הבאה מהאנימציה, המשבצת השמאלית התחתונה היא `board[0][0]`.
המשבצת שבה נמצא השחקן הכחול בלוח הבא היא המשבצת `board[4][1]`.



כל כניסה בלוח מכילה את אחד מהערכים 0, -1, 1, 2.
הערך 0 מסמל משבצת ריקה.
הערך -1 מסמל משבצת אפורה.
הערך 1 מסמל את מיקומו ההתחלתי של השחקן שלכם.
הערך 2 מסמל את מיקומו ההתחלתי של השחקן היריב.
* ייתכן שהשחקן שלכם לא יהיה השחקן שישחק ראשון, ובמקרה זה, הפונקציה `set_rival_move` תיקרא לפני הפונקציה `make_move`.

לפונקציה זו לא צריך להיות ערך החזרה.

`make_move(self, time)` - פונקציה זו תיקרא (לא על ידכם, אלא על ידי הקוד שמנהל את המשחק) בכל תור של השחקן שלכם, ומטרתה היא לקבל מהשחקן שלכם את הצעד שהוא מבצע.

פרמטרים:

`time` - מספר השניות הנתונות כדי לסיים את ריצת הפונקציה. במידה והשחקן שתממשו יחרוג ממסגרת זמן זו יורד לכם ציון על חלק זה בתרגיל.

ערך החזרה:

על הפונקציה להחזיר את האופרטור שהשחקן שלכם בחר להפעיל. אופרטורים מיוצגים על ידי `tuple` בגודל 2. ערכי ההחזרה האפשריים הם $(1, 0)$, $(0, 1)$, $(-1, 0)$, $(0, -1)$. האופרטור $(1, 0)$ מייצג צעד למעלה, $(-1, 0)$ מייצג צעד למטה, $(0, 1)$ מייצג צעד ימינה ו- $(0, -1)$ מייצג צעד שמאלה. אם $loc = (i, j)$ הוא מיקום השחקן טרם הקריאה לפונקציה `make_move` ו- $op = (v, u)$ הוא ערך ההחזרה, אז מיקומו החדש הוא $(i + v, j + u)$. כדי שצעד זה יהיה חוקי, צריכים להתקיים התנאים הבאים:

- השחקן נע באחד הכיוונים האפשריים - $(v, u) \in \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$.
- המיקום החדש הוא בלוח: $0 \leq i + v < len(board)$ וגם $0 \leq j + u < len(board[0])$.
- במיקום החדש יש משבצת לבנה: $board[i + v][j + u] = 0$.

במידה ואין מהלכים אפשריים, הפונקציה לא תקרא, ולכן אפשר להניח שבכל קריאה לפונקציה קיים לפחות אופרטור אחד שאפשר להפעיל.

`set_rival_move(self, loc)` - פונקציה זו תיקרא (לא על ידכם, אלא על ידי הקוד שמנהל את המשחק) לאחר כל תור של השחקן היריב, ומטרתה לעדכן את השחקן שלכם בצעד שהשחקן היריב ביצע, כדי שתוכלו לעדכן את הלוח.

פרמטרים:

`loc` - המיקום החדש של השחקן היריב.

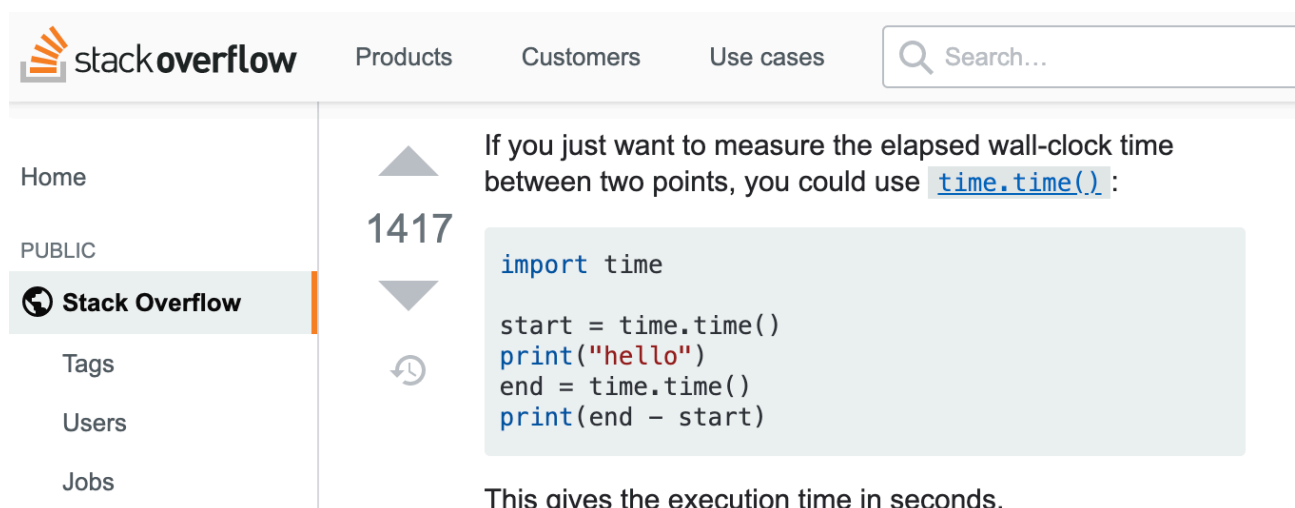
לפונקציה זו לא צריך להיות ערך החזרה.

היוריסטיקה

כפי שלמדנו בהרצאות, סוכן מינימקס מוגבל משאבים משתמש בהיוריסטיקה כדי להעריך את טיב העלים בעץ שאותו פיתח, ולקבוע את אסטרטגיית המשחק על פי ערכים אלו. עליכם להגדיר ולממש היוריסטיקה עבור שחקן ה-`minimax`, על היוריסטיקה להיות מורכבת מלפחות 3 מרכיבים, כלומר עליכם להגדיר לפחות 3 ערכים שהם פונקציה של המצב אותו אנחנו רוצים להעריך, כך שההיוריסטיקה תהיה מורכבת מקומבינציה כלשהי שלהם.

טיפ

כדי להתמודד עם מגבלת הזמן, ניתן להשתמש בספרייה `time`:



The screenshot shows a Stack Overflow page with the title "If you just want to measure the elapsed wall-clock time between two points, you could use `time.time()`:". The post has 1417 votes. The code snippet is as follows:

```
import time

start = time.time()
print("hello")
end = time.time()
print(end - start)
```

Below the code, it says "This gives the execution time in seconds."

חלק ג' - מימוש סוכן $\alpha - \beta$ (15 נק')

בחלק זה נשפר את סוכן ה- $\alpha - \beta$ $minimax$ שבנינו לסוכן $\alpha - \beta$. את השחקן עליכם לממש בקובץ ששמו `AlphaBetaPlayer.py`, במחלקה (class) ששמה `AlphaBetaPlayer`.

השחקן יכול להיות זהה לשחקן ה- $\alpha - \beta$ $minimax$, מלבד לכך שיממש את אלגוריתם $\alpha - \beta$. על השחקן להיבנות על ידי השורה:

```
alpha_beta_player = AlphaBetaPlayer()
```

לשם כך פונקציית `__init__` צריכה להיות חסרת פרמטרים, או בעלת ערכי ברירת מחדל. על השחקן לממש את הפונקציות `set_game_params`, `make_move` ו-`set_rival_move` כפי שתוארו בחלק ב'.

חלק ד' - מימוש סוכן $\alpha - \beta$ עם סידור ילדים (10 נק')

בחלק זה נשדרג את סוכן ה- $\alpha - \beta$ שלנו על ידי הוספת סידור ילדים. את השחקן עליכם לממש בקובץ ששמו `OrderedAlphaBetaPlayer.py`, במחלקה (class) ששמה `OrderedAlphaBetaPlayer`. ראינו בהרצאה שסדר הילדים בעץ המינימקס יכול להשפיע בצורה משמעותית על מידת הגיזום של אלגוריתם $\alpha - \beta$, ולכן על זמן הריצה. בחלק זה נשפר את סוכן ה- $\alpha - \beta$ על ידי סידור ילדים.

עליכם לממש סוכן $\alpha - \beta$ עם סידור ילדים דינאמי: הסוכן יבצע העמקה הדרגתית, בכל איטרציה נמיון את הילדים לפי ערכי המינימקס שלהם מהאיטרציה הקודמת, ונפתח את הילדים לפי סדר זה (שימו לב שיש חשיבות לסדר המיון (עולה או יורד), איזה בסדר הוא הנכון?). **משיקולי זכרון נבצע סידור רק לילדים של שורש עץ המינימקס.** על השחקן להיבנות על ידי השורה:

```
ordered_alpha_beta_player = OrderedAlphaBetaPlayer()
```

לשם כך פונקציית `__init__` צריכה להיות חסרת פרמטרים, או בעלת ערכי ברירת מחדל. על השחקן לממש את הפונקציות `set_game_params`, `make_move` ו-`set_rival_move` כפי שתוארו בחלק א'.



חלק ה' - תחרות

כל זוג נדרש לממש שחקן יחיד שייצג אותו בתחרות, בה ישתתפו כל הזוגות. הסטודנטים ששחקניהם יצטיינו בתחרות יזכו בבונוס לציון הסופי של הקורס: 5 נקודות לזוג המנצח, 3 נקודות למקום שני, ו-2 נקודות למקום שלישי. את שחקן התחרות שלכם יש לממש בקובץ `ContestPlayer.py`, במחלקה בשם `ContestPlayer`. על המחלקה לממש את הפונקציות `set_game_params`, `make_move` ו-`set_rival_move` כפי שתוארו בחלק א'.

אתם יכולים להגיש את אחד השחקנים שמימשתם בחלקי התרגיל הקודמים, או לממש שחקן חדש. על השחקן להיבנות על ידי השורה:

```
contest_player = ContestPlayer()
```

במידה והשחקן שתגישו לא ירוץ, יורדו לכם נקודות בתרגיל. אין להגיש את השחקן הנמצא בקובץ `SimplePlayer.py`. שחקן שלא יעמוד במגבלת הזמן באיזשהו משחק במהלך התחרות, יפסיד את המשחק ויודח מהתחרות.

מגבלת הזמן עבור הפונקציה `make_move` תהיה לכל הפחות שניה. ההשתתפות בתחרות היא חובה. יורדו נקודות לזוגות ששחקן התחרות שלהם לא ירוץ.

חוקים:

- אסור לשחקן להשתמש ברשת האינטרנט או ברשת מקומית כלשהי.
- אסור לשחקן להשתמש בכל סוג של קוד מקבילי.
- כל שחקן שינסה לרמות או לשבש את קוד היריב או המערכת ייפסל, ויורדו למגשישים נקודות בתרגיל.
- חל איסור להשתמש במידע שעובד מראש ונשמר בקובץ.
- אסור להשתמש בחבילות/סיפריות שאינן מובנות בפייתון.
- כל **חישוב** לשם קביעת הצעד הבא של הסוכן צריך להתבצע אך ורק בקריאה לפונקציה `make_move`.

חלק ו' - כתיבת דו"ח (6 נק')

ענו על הסעיפים בהמשך לשאלות בחלק היבש.

13. (2 נק') תארו את ההיוריסטיקה שקבעתם עבור שחקן ה-*Minimax*, והסבירו.
14. (2 נק') הסבירו את אופן פעילותו של שחקן התחרות שלכם. אם הגדרתם פונקציה היוריסטית חדשה, הסבירו עליה.
15. (2 נק') תארו כיצד ניהלתם את זמן ריצת הפונקציה `make_move`.

חלק ז' - ביצוע ניסויים (14 נק')

הוסיפו את התשובותיכם לסעיפים הבאים לדו"ח.

16. (3 נק') הריצו משחקים בין סוכן ה-*Minimax* לסוכן ה-*alpha – beta* שמימשתם על כל הלוחות הנתונים. מה התוצאות שקיבלתם? האם התוצאות מתאימות לצפיותיכם?

השוואת עומק החיפוש בין הסוכנים השונים

17. (4 נק') בניסוי זה נבחן את כיצד זמן הריצה הנתון משפיע על עומק החיפוש של הסוכנים השונים. לכל סוכן מבין הסוכנים *Minimax*, *Alpha – beta*, *Ordered Alpha – Beta*, צרו גרף בו ציר ה-*x* הוא הזמן הנתון עבור הרצת הפונקציה `make_move`, וציר ה-*y* הוא העומק המקסימלי שאליה הגיע הסוכן. יש להריץ את הניסוי על הלוח `ai_board` שיש ליבא מהקובץ `MapsGenerator`. הסבירו את הגרפים, ואת ההבדלים בניהם אם יש כאלו. ניתן ליצור את הגרף על ידי שינוי זמני של הפונקציה `make_move` כך שתחזיר את העומק המקסימלי בחיפוש במקום את הצעד שהסוכן בחר לעשות, ועל ידי הרצת הקוד הבא:

```
from MapsGenerator import ai_board
import numpy as np
from PlayerModule import Player
import matplotlib.pyplot as plt

times = []
depths = []

for t in np.linspace(0.1, 3, 50):
    player = Player()
    player.set_game_params(ai_board.copy())
    d = player.make_move(t)
    times.append(t)
    depths.append(d)

plt.scatter(times, depths)
plt.show()
```

שימו לב שאת מה שכתוב בירוק צריך לשנות לפי השחקן עבורו אתם יוצרים את הגרף.

השוואה בין היורסטיקה פשוטה למסובכת

18. (7 נק') בניסוי זה נרצה לבחון האם עדיף להשתמש בהיורסטיקה חכמה שלוקחת זמן רב לחישוב, או בהיורסטיקה פשוטה ומהירה לחישוב, כאשר הרעיון הוא שהיורסטיקה פשוטה תאפשר הגעה לעומק גדול יותר מאשר היורסטיקה מורכבת.

הגדירו שתי היורסטיקות: h_{lite} ו- h_{heavy} . צריכה להיות היורסטיקה פשוטה ומהירה לחישוב, ו- h_{heavy} צריכה להיות מורכבת יותר ובעלת זמן חישוב רב יותר.

צרו שני שחקני $Alpha - Beta$, אחד מהם עם h_{lite} והשני עם h_{heavy} . יש לממש את השחקנים בקבצים `LiteAlphaBetaPlayer.py` ו- `HeavyAlphaBetaPlayer.py` במחלקות `LiteAlphaBetaPlayer` ו- `HeavyAlphaBetaPlayer` בהתאמה, שצריכות להיבנות על ידי השורות:

```
lite_alpha_beta_player = LiteAlphaBetaPlayer()
```

-1

```
heavy_alpha_beta_player = HeavyAlphaBetaPlayer()
```

בהתאמה. כדי להשוות בין עומק החיפוש אליו מגיעים הסוכנים, עבור כל אחד מהם צרו גרף כמו הגרף בסעיף הקודם. בנוסף, הריצו מספר משחקים בין שני הסוכנים על כל אחד מהלוחות הנתונים. כרגיל, כדי להריץ את משחק בין הסוכנים הריצו את השורה:

```
python LiveAnimatedGame.py HeavyAlphaBetaPlayer LiteAlphaBetaPlayer
```

התשובה בסעיף זה צריכה להיות מורכבת מהסבר על ההיורסטיקות, גרף עבור כל שחקן ופירוט תוצאות המשחקים.

הוראות הגשה

יש להגיש את כל הקבצים (קוד ודו"ח) בקובץ zip יחיד ששמו `AI2_<id1>_<id2>` כאשר במקום `<id1>` ו- `<id2>` יש לרשום את מספרי תעודת הזהות של המגישים. מלבד לדו"ח, קובץ ה-`zip` צריך להכיל את כל קבצי הקוד שאתם יצרתם לשם מימוש השחקנים והרצת הניסויים. אין להגיש אף קובץ שהיה מצורף לתרגיל.

בהצלחה!

