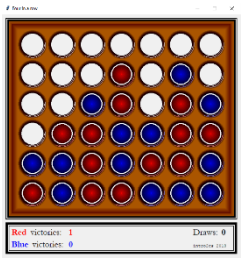
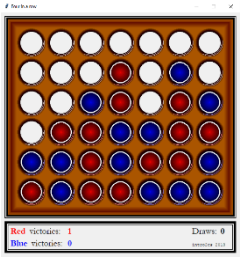
בית הספר להנדסה ולמדעי המחשב על שם רחל וסלים בנין



מבוא למדעי המחשב – 67101

תרגיל 12 – "ארבע בשורה"

(להגשה עד תאריך 16/01/2019 בשעה 22:00)



בתרגיל זה עליכם לממש את המשחק "ארבע בשורה" (Four in a Row).

המשחק מורכב מלוח בעל 6 שורות ו-7 עמודות ומיועד עבור שני שחקנים (לדוגמא, אדום וכחול). בכל תור, על השחקן לבחור עמודה אליה הוא מכניס דיסקית – הדיסקית נוחתת תמיד בתחתית, בראש ערימת הדיסקיות שכבר הוכנסו בעמודה זו. המשחק נגמר כאשר אחד מהשחקנים מייצר רצף של ארבע דיסקיות בצבע שלו, בכל אחד מהכיוונים האפשריים (מאונך, מאוזן או אלכסון). ניתן לשחק במשחק "ארבע בשורה" לדוגמא בקישור: https://www.coolmath-games.com/0-4-in-a-row.

מימוש המשחק

בתרגיל זה מוגדרות לכם חתימות פומביות לשתי מחלקות – Game ו-AI (ראו להלן). שתי המחלקות ריקות ממימוש, ויש לכם חופש פעולה מוחלט בתכנון הקוד (אך אינכם יכולים להוריד או לשנות את המתודות הפומביות). כלומר, מותר לכם להוסיף מתודות ומשתנים פרטיים ופומביים, לכתוב מחלקות חדשות, לייבא ולהשתמש בהן מכל מקום ובכל צורה שתחפצו, להשתמש בתמונות, ועוד. זו ההזדמנות לתכנן מהמסד ועד הטפחות תוכנה שלמה על פי ראות עיניכם ולהיות יצירתיים: שימו לב לעקרונות של קוד נקי, אי חזרה על קוד, מודולריות, ועוד. Show us what you've got! ☺

הערות:

1. משחק לא "יקרוס" מפעולות בלתי חוקיות, וגם לא יבצע אותן (ישנן דרכים רבות להשיג זאת).
2. שימו לב למבנה התיקיות הרצוי (והחריג), תחת סעיף **"מבנה הפרויקט והוראות הגשה"**.
3. מותר לכם לעשות שימוש בכל כלי בשפת פייתון, בין אם נלמד בקורס ובין אם לאו, מלבד שימוש בכלים המתקדמים הבאים (שלא נלמדו בקורס): תקשורת (networking מכל סוג שהוא), ריבוי תהליכים לסוגיו (בין אם מבוסס process-ים ובין אם מבוסס thread-ים), או בסיגנלים.

גרפיקת ממשק משתמש (GUI)

המשחק ימומש בצורה גרפית בלבד, על ידי שימוש במודול **tkinter** שראיתם בשיעור – **במהלך ריצת המשחק, אין להדפיס דבר ל-console באמצעות פונקציית print או כל פונקציה אחרת** (כמובן, במהלך הכתיבה ניתן ואף מומלץ לבצע בדיקות שונות על ידי הדפסה ל-console). בידיכם בחירה מלאה לגביי עיצוב הלוח: צבעים, נראות, גודל ואופן ביצוע המהלכים. חשבו מה תהיה חוויית משתמש נוחה ולא מסורבלת: למשל, עליכם לדאוג כי לא ניתן יהיה לבצע מהלכים שאינם חוקיים באמצעות הממשק הגרפי. בפרט, על המימוש לקיים את כל התנאים הבאים:

1. ניתן להבין בקלות (בעיניים) את מצב הלוח.
2. ניתן להבין בקלות (בעיניים) כיצד לבצע מהלך.
3. המימוש לוקח בחשבון את סוגיית התורות, כלומר: שחקן א' לא יכול לשנות את מצב הלוח בזמן ששחקן ב' אמור לשחק, ולהיפך.
4. עם סיום המשחק, יש לתת חיווי ברור על המנצח (או תיקו במקרה שאין עוד מהלכים אפשריים), כולל כל הדיסקיות בלוח מהן התקבל הרצף המנצח. שוב, חיווי זה **לא** יתבצע על ידי הדפסה ל-console. עם סיום המשחק, ניתן לבחור האם להמשיך למשחק נוסף (הלוח מתאפס ומוכן לסיבוב נוסף), או לצאת מהמשחק (התכנה נסגרת).

נציין כי גם מימושים בסיסיים ביותר (ובתנאי שעונים על הקריטריונים לעיל) יקבלו ציון מלא (גם אם העיצוב אינו מרשים ויזואלית). עם זאת, על מימושים אלגנטיים במיוחד, תינתנה עד **5 נקודות** בונוס לציון התרגיל (לשיקול דעתו של הבודק).

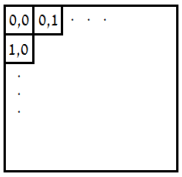
ביצוע מהלכים והמחלקה Game

ניתן לכם שלד המחלקה Game בתוך הקובץ game.py (יודגש כי הרצת הקובץ game.py לבדו אינה אמורה לעשות דבר). מחלקה זו תייצג את "הלוגיקה הפנימית" של מצב המשחק (ללא הגרפיקה), ותאפשר לבצע מהלכים על הלוח. עליכם לתכנן את אופן הפעולה הפנימי של מחלקה זו – מותר לכם להוסיף משתנים ומתודות (פומביים ופרטיים) כראות עיניכם, אך עליכם לשמור על המתודות הפומביות והקבועים המוגדרים במחלקה בדיוק כפי שהם, ולדאוג לכך שהם יבצעו את הפעולה המבוקשת. בפרט:

1. **חתימת בנאי (Consructor)** המחלקה הינה **__init__(*self*)**.
2. **המחלקה תכיל את המתודה make_move(*self, column*)** – ביצוע מהלך במשחק יתבצע דרך קריאה למתודה זו, המקבלת את העמודה בה רוצים להציב דיסקית (שימו לב כי המתודה צריכה להסיק מי השחקן הנוכחי, בהתאם להתקדמות המשחק). אם לא ניתן להציב דיסקית

(כיוון שהעמודה הנבחרת מלאה, מחוץ לתחום, או שהמשחק הסתיים), עליכם להעלות Exception עם ההודעה: "Illegal move.".

3. **המחלקה תכיל את המתודה (get_current_player(*self*** – מחזירה את השחקן הנוכחי (הערך השלם 1 עבור שחקן א' והערך השלם 2 עבור שחקן ב').



4. **המחלקה תכיל את המתודה (get_player_at(*self, row, col*** – מחזירה דיסקית של איזה שחקן נמצאת ב-[*row, col*] (1 עבור שחקן א', 2 עבור שחקן ב', או None – אם אף שחקן לא הציב דיסקית במיקום זה). אם המיקום אינו תקין, המתודה תעלה את השגיאה "Illegal location.". <u>שימו לב!</u> הפינה השמאלית העליונה מוגדרת עם הקואורדינטות row=0, col=0.

5. **המחלקה תכיל את המתודה (get_winner(*self*** – קריאה למתודה בכל שלב (גם לא בסוף) תחזיר את "מצב הניצחון": 1 – ניצחון שחקן א', 2 – ניצחון של שחקן ב', או 0 – תיקו (כלומר, כאשר הלוח מלא לגמרי ואין מנצח) אם טרם הסתיים המשחק, הפונקציה תחזיר None (שימו לב לבדוק שורות, עמודות ואלכסונים).

אופן מימוש והשימוש באובייקט Game צריך להיות כזה, שאתחול אובייקט Game יחיד, ואז קריאות חוזרות ונשנות לביצוע מהלך (על ידי make_move( )), תקדמנה את לוגיקת המשחק. בסופו של דבר, נגיע למצב בו יש מנצח, או תיקו (לוח מלא). עליכם לזהות סיום משחק, ולא לאפשר ביצוע מהלכים נוספים באמצעות הממשק הגרפי. קריאה נוספת ל-make_move, כשהמשחק הסתיים, תעלה את ה-Exception: "Illegal move.". לדוגמא, קטע הקוד הבא מתאר מצב בו שחקן א' ניצח את המשחק:

```
g = Game()
g.make_move(0)   # 1 "red" disc on col-0.
g.make_move(1)   # 1 "red" disc on col-0, 1 "blue" disc on col-1.
g.make_move(0)   # 2 "red" discs on col-0, 1 "blue" disc on col-1.
g.make_move(1)   # 2 "red" discs on col-0, 2 "blue" discs on col-1.
g.make_move(0)   # 3 "red" discs on col-0, 2 "blue" discs on col-1.
g.make_move(1)   # 3 "red" discs on col-0, 3 "blue" discs on col-1.
g.make_move(0)   # player one has won. Cannot make another move.
g.get_winner()   # returns 1 (victory for player 1).
g.make_move(0)   # raises an exception.
```

המחלקה Game **תייצג את לוגיקת המשחק בלבד**, ללא שום מעורבות מצד האלמנט הגרפי (כמובן, שמותר לה להשתמש באובייקטים נוספים שתגדירו, כל עוד הם אינם מערבים מודול גרפי כלשהו).

בינה מלאכותית

עליכן לממש את המחלקה AI בתוך הקובץ ai.py (הרצת קובץ זה לבדו אינה אמורה לעשות דבר). המחלקה אמונה על ביצוע מהלכים על ידי שחקן ממוחשב. אנו מצפים מכן לבצע מימוש בסיסי ביותר למחלקה, אך מימושים מתוחכמים יותר, עשויים לזכות אתכן בבונוס (ראו לעיל).

תוכן המחלקה ופעולתה נתון לשיקולכן, אך אתן חייבות להשאיר לפחות את ה-API הבא:

- **למחלקה יהיה בנאי בחתימה (__init__(*self, game, player*** שמקבל אובייקט Game המנהל את לוגיקת המשחק, ואת מספר השחקן (1 או 2) עבורו האובייקט הנ"ל "משחק" (כלומר, אם שני השחקנים ממוחשבים, נצטרך לייצר שני אובייקטים בעת ניהול המשחק, אחד לכל שחקן). <u>שמנה לב להוראות הבאות!</u>
  - **אי אפשר להניח שאובייקט ה-Game הוא ה-Game במימוש שלכן: מותר להשתמש אך ורק במתודות הפומביות שהוגדרו עבור Game בהוראות התרגיל** (במקומות אחרים בהם יש שימוש ב-Game, כן מותר להשתמש בכל המתודות שתגדרנה).
  - חוקי לתת למחלקה AI אובייקט Game המייצג משחק באמצעו (לוח שאינו ריק). ניתן להניח שהאובייקט מייצג משחק במצב תקין (למשל, אין דיסקיות בשורה גבוהה מבלי שתחתיהן יש דיסקיות). לא ניתן להניח שאין כבר מנצח במשחק.
  - **אסור לקרוא למתודה make_move מתוך אף אחת ממתודות AI (כולל הבנאי ו-find_legal_move המוגדרת בהמשך)**, דהיינו, אסור לשנות את מצב הלוח כחלק מהפעולה של AI. אם, כחלק מפעולת מנגנון ה-AI שלכן, אתן רוצות "לעבוד" עם אובייקט Game שניתן לשנות אותו, תוכלנה לייצר עותק (אובייקט) נוסף של Game, אשר יכול להיות אובייקט Game בתכנון ובמימוש שלכן, המשקף את מצב המשחק השמור באובייקט Game של המחלקה. תוכלנה לעדכן את אובייקט העותק שלכן, מבלי לעדכן את אובייקט Game השמור במחלקה (בדקו שאכן שינוי ב-Game שלכן אינו גורר שינוי ב-Game השמור במחלקה). **יודגש כי אין כל הכרח לעשות זאת מלכתחילה אם אין לכן כוונה לשנות את מצב האובייקט Game השמור במחלקה**; ניתן לבצע מימוש פשוט של מתודות המחלקה AI גם עם אובייקט Game השמור במחלקה ומבלי לבצע שינויים במצב הלוח (כלומר – מבלי כלל להתייחס לכל ההבהרה הרשומה בסעיף זה).
- **למחלקה תהיה מתודה בשם (find_legal_move(*self, timeout=None*** – מחשבת ומחזירה מהלך חוקי עבור השחקן שניתן לאובייקט בבנאי. "חוקי", הכוונה למספר המייצג עמודה לא מלאה (בין 0 ל-6), כפי שמבוטא על ידי מצב האובייקט Game שניתן למחלקה. אם לא נמצא

מהלך אפשרי, או שהמשחק הסתיים טרם הקריאה למתודה, היא תעלה Exception עם הכיתוב "No possible AI moves.".

<u>שימו לב</u>:

1. **אין צורך להתייחס לפרמטר *timeout*, אלא אם תממשנה את הבונוס** (לעיל). אם אינכן ממששות את הבונוס, ניתן גם למחוק את פרמטר זה מהגדרת המתודה.
2. מבחינת ה-AI, המצבים בהם אין מהלך תקין הם או א) לוח מלא או ב) יש מנצח במשחק.

איך מוצאים מהלך? לשיקולכן! עם זאת **<u>המינימום הנדרש</u>**, הינו למצוא מהלך רנדומאלי תקין.

- **למחלקה תהיה מתודה בשם get_last_found_move( )** – אם אינכן מממשות את הבונוס, אין צורך לממש את המתודה, וניתן להשאיר אותה עם פקודת pass בלבד.

בעת הרצת המשחק, ניתנת אפשרות לבחור האם כל אחד מהשחקנים יהיה אנושי, או ממוחשב (ראו להלן). בזמן תורו של שחקן ממוחשב, **אין לאפשר לשחקן אנושי לבצע אף מהלך, אך כן יש להציג את הלוח וביצוע המהלכים באופן גרפי**. כמו כן, אם שני השחקנים הינם ממוחשבים, יש לבצע השהייה בין שני מהלכים עוקבים (למשל – אך לא בהכרח – ע"י שימוש ב-time.sleep() של המודול time ב-Python), וזאת על מנת שניתן יהיה לעקוב אחר המהלכים בצורה ויזואלית.

<u>כמו כן</u>:

1. המחלקה AI **תייצג את לוגיקת האינטליגנציה המלאכותית בלבד**, ללא שום מעורבות מצד הצד הגרפי (אך כמובן תוך שימוש ב-Game שמסופק לה, ולשיקולכן, אולי גם אובייקטים נוספים).
2. מהלכי האינטליגנציה המלאכותית צריכים לקרות בזמן סביר (לא יותר מכמה שניות).

למען הסר ספק, <u>שימו לב</u>! למחלקות Game ו-AI <u>אסור</u> לייבא (import) או להשתמש באף מודול גרפיקה של שפת Python (כולל tkinter). מותר להשתמש בתוך Game ו-AI במחלקות אחרות שתגדירו, רק בתנאי שגם הן <u>לא משתמשות בספריות גרפיקה, או משתמשות בספריות שמשתמשות בספריות גרפיקה, וכו'</u>.

!

טיפים

1. שימו לב שהמשחק מאוד מודולארי:
   - מודול אחד, אחראי על הגרפיקה (GUI).
   - מודול אחר, אחראי על לוגיקת המשחק.
   - מודול נוסף, אחראי על בינה מלאכותית.
   - מודולים נוספים, אחראים על דברים נוספים (?)

כפועל יוצא, שווה מאוד להבין שאפשר לכתוב חלק ניכר מכל מודול, ולבדוק שהוא עובד, מבלי לערב בכלל את המודולים האחרים (אחד מהיתרונות הגדולים של תכנות מונחה עצמים: מודולריות ואנקפסולציה). למשל, ניתן לכתוב את מודול הגרפיקה, מבלי שיתממשק (בינתיים) ללוגיקת המשחק – רק לבדוק שניתן ללחוץ ולהציב דיסקיות, גם אם ההשמה אינה חוקית.

אנו ממליצים (אך כמובן, אין זו חובה) להתחיל ממימוש של Game ולוגיקת המשחק: נסו לכתוב תכנה פשוטה המסמלצת משחק (תור-תור), תוך שימוש ב-Game ובאובייקטים נוספים שאולי תיצרו. ניתן לעשות זאת, למשל, על ידי בקשה מהמשתמש להקליד מספר עמודה שבה הוא מעוניין להציב, לשנות את מצב הלוח, להדפיס את הלוח כפלט ל-console, ולבקש מהלך נוסף. לאחר שתראו שהמשחק עובד, נסו לממש את מתודת הבינה המלאכותית (הטריוויאלית), ולראות איך ניתן לשלב אותה לעשיית מהלך, במידה והתכנה הורצה תוך בחירת מחשב כשחקן. לאחר מכן, נסו לראות אם אתם מצליחים לשלב את ה-GUI שכתבתם.

2. בשלב של מימוש ה-GUI, עדיין ניתן להדפיס ל-console על מנת לבצע בדיקות תקינות: "למה מספר העמודה לא תקין?", "למה הדיסקית לא הופיעה?" וכו'. השתמשו בכלי זה, כיוון שניפוי שגיאות בעזרת ה-debugger מסובך יותר בשלב הגרפי (בשל לולאת ה-mainloop ב-tkinter).

הרצת המשחק

הטיפול ב-main של המשחק צריך להיעשות מתוך קובץ בשם four_in_a_row.py. עם הרצת המשחק, יש לאפשר למשתמש לבחור <u>בצורה גרפית</u> (לבחירתכם) מה יהיה סוג השחקן עבור כל אחד מהשחקנים: אנושי, או מחשב (כלומר, קיימת אפשרות לשחק במצבים אנושי-אנושי, אנושי-מחשב, מחשב-אנושי

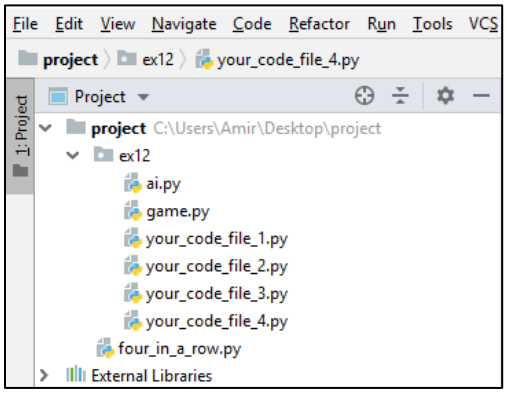
ומחשב-מחשב). אתם יכולים לבחור האם, בעת סיום משחק, ניתנת אפשרות נוספת לבחירת סוג השחקנים, או שממשיכים עם אותו סוג שנבחר קודם לכן.

שימו לב כי גם אם שני השחקנים מוגדרים כממוחשבים, עדיין בסיום המשחק ישנה שאילתא למשתמש, השואלת האם אנו מעוניינים במשחק נוסף. במקרה של שני שחקנים ממוחשבים, אנו מצפים לראות "פינג-פונג" של מהלכים אוטומטיים (עם השהייה קטנה בין מהלך למהלך על מנת שניתן יהיה לעקוב ויזואלית), בסופם ישנו מנצח או תיקו (עם חיווי ברור), ואז אפשרות מצד המשתמש להריץ עוד משחק.

מבנה הפרויקט והוראות הגשה

<u>חלוקה לספריות</u>:

שימו לב: מבנה הפרויקט שונה מהתרגילים שהגשתם עד כה.



עליכם ליצור תיקייה בשם **ex12**, תחתיה יהיו כל קבצי הקוד שלכם, <u>מלבד four_in_a_row.py</u> (קובץ זה צריך להיות מחוץ לתיקייה). כלומר, תיקייה זו תכיל את הקבצים game.py, ai.py, וכן כל קובץ קוד אחר שתבחרו לייצר. ראו התרשים משמאל, המתאר את מבנה הפרויקט הנכון (כפי שיוצר בסביבת PyCharm), בו ייצרנו קבצי עזר בשם "your_code_file_1.py", "your_code_file_2.py" וכו'.

אם אתם עובדים ב-PyCharm, ניתן לייצר את הסביבה הנכונה בצורה פשוטה: ראשית, מייצרים פרוייקט חדש ע"י File → New Project (בתרשים קראנו לפרויקט בשם המקורי "project"), ואז קליק ימני על הפרויקט ובחירה ב-New → Directory, שלה נקרא "ex12" (כאמור, חובה לבחור בשם זה). שימו לב כי אם במקום New Directory תייצרו New Python Package (אפשרות נוספת בתפריט), יווצר לכם קובץ לוואי בשם __init__.py אותו עליכם למחוק. כעת, אם נרצה לייצר קבצי Python בתוך התיקייה, נבצע לחיצה ימנית על התיקייה ונייצר קובץ; אם נרצה לייצר קובץ מחוץ לתיקייה (כאמור, אך ורק four_in_a_row.py), נעשה זאת ע"י קליק ימני על הפרויקט עצמו.

על מנת לעשות ייבוא (import) של קבצים, חשוב מאוד לעקוב אחר ההוראות להלן:

עד כה, כאשר רציתם לייבא קובץ, מחלקה, פונקציות או משתנים, השתמשתם בפקודת import פשוטה בסגנון של "import game", "from game import Game", "from game import *" וכיוצ"ב. כעת,

עליכם לעשות ייבוא שמתחשב במיקום הקובץ. כלומר, אם אתם בקובץ four_in_a_row.py, אשר נמצא מחוץ לתיקייה ex12, או אם אתם בקובץ כלשהו שנמצא בתוך התיקייה, תצטרכו לשנות את פקודות ה-import באופן הבא:

בתוך four_in_a_row.py:

הפקודה שאינה מתחשבת במיקום יחסי:	**הפקודה בה עליכם להשתמש:**
import game	import ex12.game
from game import *	from ex12.game import *
from game import Game	from ex12.game import Game

אך אם ברצונכם לייבא קבצים מתוך התיקייה, עבור קוד הנמצא גם הוא בתוך התיקייה (למשל: לייבא את המחלקה Game מתוך הקובץ game.py, בתוך הקובץ ai.py), תצטרכו לרשום נתיב יחסי באמצעות שימוש בתו הנקודה (".."), באופן הבא:

הפקודה שאינה מתחשבת במיקום יחסי:	**הפקודה בה עליכם להשתמש:**
import game	from . import game
from game import *	from .game import *
from game import Game	from .game import Game

שימו לב **לא** לעשות import עם שם התיקייה המפורש (ex12), עבור קבצים הנמצאים בתוך התיקייה ומייבאים קבצים הנמצאים גם הם בתוך התיקייה, זאת בניגוד לקובץ four_in_a_row.py הנמצא מחוץ לתיקייה ex12 וחייב להשתמש בשמה. כלומר, אם נרצה לייבא את Game מתוך game עבור הקובץ ai.py, לא נרשום from ex12.game import Game אלא from .game import Game.

<u>הגשת התרגיל הינה בזוגות</u>:

נהלי ההגשה בזוגות זהים לאלו של תרגיל 6 ו-10 – לפני ההגשה ותחת הלינק המיועד להגשה, עליכם לפתוח קבוצה ב-moodle. אחד השותפים ייצור את הקבוצה על ידי הזנת שם השותף השני (שימו לב להוסיף את השם בדיוק כפי שהוא מופיע ב-moodle, כולל אותיות גדולות וקטנות במקומות הנכונים). השותף (שלא יצר את הקבוצה) יוכל לראות שרשמו אותו על ידי כניסה לקישור ההגשה וצפייה בשם בן

הזוג. כדאי לרשום את בן הזוג בשלב מוקדם (אין צורך להגיש את התרגיל בפועל על מנת להירשם כזוג). כמו כן, וודאו כי הקבוצה אינה מכילה יותר משני שותפים.

יש להגיש את כל הקבצים בהם עשיתם שימוש, כולל תמונות (אם השתמשתם בתמונות). בדומה לתרגיל 6, יש להגיש קובץ authors בשם **AUTHORS** (ללא כל סיומת). קובץ זה יכיל שורה אחת ובו ה-login (במחשבי האוניברסיטה) של שני השותפים, מופרדים על ידי פסיק וללא רווח, כמו בדוגמא הבאה:

rey,kyloren

בנוסף, כבכל תרגיל, עליכם להגיש README סטנדרטי (פרטים מזהים, מספר תרגיל, וכמה שורות תיאור) – למעט במקרה בו החלטתם לפתור את הבונוס, ואז על ה-README להכיל גם את תיאור אלגוריתם הבינה המלאכותית שלכם (ראו לעיל). את הקבצים עליכם להגיש בתוך קובץ zip בשם **ex12.zip**, במבנה כפי שמתואר בתרשים; כלומר, ישירות תחת קובץ ה-zip צריכים להיות הקבצים four_in_a_row.py, README ו-AUTHORS, ותיקייה בשם ex12. תיקייה זו תכיל את הקבצים game.py, ai.py וכן כל קובץ אחר בו עשיתם שימוש, כולל תמונות.

```
ex12.zip
---------
↳README
↳AUTHORS
↳four_in_a_row.py
↳ex12
  ↳game.py
  ↳ai.py
  ↳additional files...
```

ההגשה הינה עד לתאריך **16/01/2019** (יום רביעי) **בשעה 22:00**.

הערה א': כדאי לבדוק, לאחר שעשיתם zip, שאתם מצליחים לעשות unzip (כלומר, extract לתיקייה אחרת) ולהריץ את הקוד על ידי קריאה ל-four_in_a_row.py.

הערה ב': אם אתם בוחרים לייצר קובץ zip באמצעות הטרמינל של לינוקס, זכרו להיזהר: אם לא תציינו שם קובץ לקובץ ה-zip, הקובץ הראשון ברשימה יידרס ויאבד! (ראו מצגת תרגול 1).

הערה ג': בתרגיל זה, קוד ה-presubmit, מעבר למספר בדיקות כלליות (הימצאותם של כל הקבצים הרלוונטיים, למשל) יבדוק גם שימוש בסיסי מאוד במחלקה Game (הרצת משחק אחד עם עמודות שאנו בוחרים, תוך שימוש באובייקט שלכם). עם זאת, הקוד אינו בודק את תקינות הקוד מעבר לכך: בפרט, הוא אינו בודק שהקוד רץ, שהגרפיקה תקינה, וכל מקרי הבסיס מכוסים. עליכם לבדוק זאת בעצמכם, ואנו ממליצים לכתוב בדיקות אוטומטיות משלכם טרם הגשת התרגיל. אתם רשאים לחלוק ביניכם בדיקות אוטומטיות שכאלו (אך כמובן, לא את קוד התרגיל עצמו).

שוב, זכרו:

1. **לבדוק שהמחלקות Game ו-AI שכתבתם עומדות בתנאים שפורטו עבורן**, כלומר: אינן תלויות או משתמשות (או מייבאת) אף מודול גרפיקה, ואינן משתמשת באף אובייקט או מחלקה אשר תלויים או משתמשים במודול גרפיקה כלשהו (למשל, tkinter).
2. **לבדוק שמבנה הקבצים בקוד שלכם תקין** (כפי שפורט במבנה הפרויקט), וכן שכל הקבצים הנחוצים על מנת להריץ את המשחק שלכם כלולים בהגשה.

בהצלחה!

בונוס (אינטליגנציה מלאכותית)

תוכלנה לנסות ולממש בינה מלאכותית חכמה בכל דרך שתבחרנה, תוך שימוש באותה מתודה find_legal_move() אשר הוגדרה לכן לעיל, ואנחנו נבדוק עד כמה היא טובה (כאן, יש הרבה מקום למחשבה אלגוריתמית וניסיון למצוא מהלכים שאינם פשוט עמודה תקינה אקראית). מותר לכן להיעזר ברעיונות מהאינטרנט, אך כמובן שלא להעתיק קוד קיים. אם יש ספק, יש לשאול בפורום התרגיל; מכל מקום, אנו נצפה מכן לדעת להסביר את הפיתרון שלכן בצורה משכנעת.

<u>כיצד זה יעבוד, ואיך צריך להתאים את המתודה?</u>

אנחנו נבנה סביבת הרצה נפרדת, שאינה מתחשבת בגרפיקה או בכל קוד אחר שכתבתן, מלבד ה-AI וה-Game שלכן (וכל קוד נוסף, שאינו גרפי, ודרוש על מנת להפעיל את מנגנון ה-AI). בסביבה זו, אנו נבדוק את איכות המהלכים ש-find_legal_move תייצר.

מה הכוונה "לייצר" מהלכים?

המתודה find_legal_move פועלת תחת שימוש באובייקט Game אשר ניתן למחלקה (ומייצג את "מצב המשחק" הנוכחי), מתחשבת בזהות השחקן ובפרמטרים נוספים ומנסה למצוא מהלך מיטבי. היא עשויה למצוא מספר מהלכים תקינים, ואולי לשפר את המהלך הטוב ביותר שהיא הגיעה אליו באופן

דינמי בזמן פעולתה. נוסף על להחזיר את המהלך **הסופי** שאליו הגיעה, המתודה גם תשמור – במחלקת AI – את המהלך (מספר עמודה) העדכני הטוב ביותר אליו הצליחה להגיע **עד כה**, בכל רגע נתון (גם לפני שסיימה). לאחר שה-timeout שניתן למתודה חלף (זוהי מגבלה שאנחנו אוכפים, ואין לכם צורך לממש אותה), אנחנו נפסיק מיידית את find_legal_move שלכן, **גם אם היא עדיין לא סיימה**. מבחינתנו, המהלך הכי טוב שהמתודה הצליחה למצוא, הוא המהלך האחרון שהיא שמרה במחלקה.

כדי להגיע למהלך שנשמר במחלקה, אנו נקרא למתודה get_last_found_move (אשר תחזיר את אותו ערך אחרון ש-find_legal_move שמרה במחלקת ה-AI, לפני שפעולתה הופסקה): במימושים מתקדמים בהם המתודה find_legal_move עשויה לרוץ זמן ארוך ולמצוא מספר מהלכים, אם היא הופסקה באופן מלאכותי טרם סיום פעולתה ייתכן כי הערך שיוחזר ב-get_last_found_move יהיה שונה מזה שהמתודה find_legal_move הייתה מחזירה עם "ריצה מלאה". שימו לב כי אם ה-timeout עבר, והמתודה find_legal_move לא סיימה לפעול, אנחנו ניקח את המהלך שמוחזר מ-get_last_found_move בכל מקרה. אם המהלך אינו תקין, אנחנו נבחר מהלך רנדומי עבורכן.

הפרמטר timeout מסופק לכן על מנת שתוכלנה לקבל החלטות אלגוריתמיות בהתבסס על הזמן שנכפה עליה לרוץ (במילישניות). למשל, אם יש לכן אלגוריתם א' שפועל מאוד מהר, אבל מוצא מהלך בינוני, ואלגוריתם ב' שפועל לאט, ומוצא מהלך טוב, אולי יש מקום להריץ דווקא את א' כשפרק הזמן שניתן לרוץ הוא קטן. **אנו נבדוק את איכות התוצאות שהמתודה לכן מנפקת עם כמה timeout-ים, חלקם קטנים מאוד (כמה מילישניות) וחלקם עד כמה שניות – נסו לתכנן בהתאם.**

<u>לדוגמא</u>:

אתחלנו שני אובייקטי AI בשם ai_1 ו-ai_2 המייצגים את החלטות הבינה המלאכותית לשחקנים א' ו-ב', בהתאמה, עם אובייקט Game המייצג לוח ריק (שוב, חשוב לשים לב כי מדובר באובייקט Game גנרי, העוקב אחר כללי ה-API, ולא ניתן להניח כי הוא מכיל מתודות מיוחדות שלכן). ai_1 ו-ai_2 מקבלים, כמובן, את אותו אובייקט Game, כיוון שמדובר באותו משחק עבור שניהם.

- מבצעים, ב-ai_1, קריאה ל-find_legal_move(timeout=1000) (שנייה אחת לרוץ). המתודה צריכה למצוא מהלך מיטבי לשחקן א'. המתודה מוצאת מהלך בודד – העמודה הרביעית (column=3), ומחזירה אותו. קריאה ל-get_last_found_move תחזיר את הערך 3.
- מבצעים, ב-ai_2, קריאה ל-find_legal_move(timeout=100) (עשירית שנייה לרוץ) – כעת, מצב המשחק הוא שיש כבר דיסקית בעמודה הרביעית. המתודה צריכה למצוא מהלך מיטבי

לשחקן ב׳. היא מחליטה שהצעד החכם ביותר הוא להציב דיסקית בעמודה הרביעית ולחסום את שחקן א׳, וזה המהלך שהיא מחזירה. קריאה ל-get_last_found_move תחזיר 3.

(... מהלכים נוספים במשחק... כעת הלוח מכיל דיסקיות רבות)

- מבצעים, ב-ai_2, קריאה ל-find_legal_move(timeout=100) – המתודה מנסה למקסם מהלך עבור שחקן ב׳. לאחר בחינה מיידית של הלוח, היא רואה שכדאי לשחקן ב׳ לבחור בעמודה הראשונה (column=0), ולכן מעדכנת את ערך המהלך האחרון העדכני במחלקה. עברו בינתיים 60 מילישניות. לאחר 98 מילישניות וחישובים נוספים, המתודה מחליטה שמהלך טוב יותר יהיה עמודה 2, ולכן מעדכנת את ערך המהלך האחרון העדכני במחלקה. לו המתודה הייתה מקבלת זמן רב יותר, המהלך שהיא הייתה מחזירה בסופו של דבר הינו עמודה 6 (ולא 2), אבל לאחר 100 מילישניות הפסקנו את פעולתה. כעת, קריאה למתודה get_last_found_move תחזיר את הערך עבור עמודה 2 (column=1).
- מבצעים, ב-ai_1, קריאה ל-find_legal_move(timeout=50) – תורו של שחקן א׳, אך המתודה לא מצאה ועדכנה אף מהלך בזמן ריצתה, לפני שהופסקה מלאכותית לאחר 50 מילישניות. הערך האחרון השמור במחלקה הוא עדיין עמודה 2, ולכן זה הערך שיוחזר ב-get_last_found_move.
- מבצעים, ב-ai_2, קריאה ל-find_legal_move(timeout=5) – תורו של שחקן ב׳: המתודה רואה שהזמן לקבלת החלטות קטן מאוד, ולכן פשוט שומרת מהלך רנדומלי במחלקה, ומחזירה אותו.

<u>ומה הבונוס?</u>

1. <u>הסבר בעת ההגשה</u>: אם בחרתן לממש מתודה של בינה מלאכותית בדרך שאינה טריוויאלית, עליכן לכתוב תיאור **קצר** (עד 200 מילים) ב-README, עבור הבודק/ת. מימושים יפים יזכו ל**עד 3 נקודות** בונוס בציון התרגיל.
2. <u>תחרות מעשית</u>: אנו נריץ את אלגוריתמי הבינה המלאכותית שתכתובנה אחד נגד השני: כלומר, נסמלץ משחק של בינה מלאכותית אחת כנגד השנייה, ובכל תור ניקח את המהלך הטוב ביותר שסופק על ידי המתודה ונבצע אותו. התחרות תכלול שימוש בקבועי זמן שונים, ותוך שימוש במצבי לוח שונים. **20 התוכנות בעלות התוצאות הטובות בתחרות (בעלות מספר הנצחונות הגבוה ביותר), תזכינה את כותבותיהן ב-2 נקודות בונוס נוספות בתרגיל**, ובמקום של כבוד בלוח המנצחים שיפורסם עם הבדיקה (כלומר, תרגיל מושלם בקוד וב-README, מושקע גרפית ושהצליח להתברג בין 20 התכנות החכמות ביותר, יכול לקבל ציון של עד **110 נקודות**).