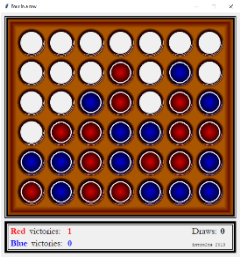
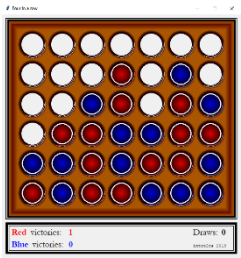
בית הספר להנדסה ולמדעי המחשב על שם רחל וסלים בנין



מבוא למדעי המחשב – 67101

תרגיל 12 – "ארבע בשורה"

(להגשה עד תאריך 24/01/2018 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 עבור העמודה. המשחק נגמר כאשר אחד מהשחקנים מייצר רצף של ארבע דיסקיות בצבע שלו, בכל אחד מהכיוונים האפשריים (מאונך, מאוזן או אלכסון). ניתן לשחק במשחק לדוגמא בקישור: https://www.coolmath-games.com/0-4-in-a-row.

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. בפרט, על המימוש לקיים את כל התנאים הבאים:

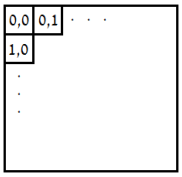
1. ניתן להבין בקלות (בעיניים) את מצב הלוח.
2. ניתן להבין בקלות (בעיניים) כיצד לבצע מהלך.
3. המימוש לוקח בחשבון את סוגיית התורות, כלומר: שחקן א' לא יכול לשנות את מצב הלוח בזמן ששחקן ב' אמור לשחק, ולהיפך.
4. עם סיום המשחק, יש לתת חיווי ברור על המנצח (או תיקו במקרה שאין עוד מהלכים אפשריים), כולל כל הדיסקיות בלוח מהן התקבל הרצף המנצח. לאחר הסיום, אף אחד מהשחקנים אינו רשאי לבצע מהלך נוסף והדבר האפשרי היחיד הינו לסגור את החלון.

נציין כי גם מימושים בסיסיים ביותר (העונים על הקריטריונים לעיל) יתקבלו, מבלי לגרוע בציון בשל נראות או עיצוב. עם זאת, על מימושים אלגנטיים במיוחד, ינתנו עד **5 נקודות** בונוס לציון התרגיל (לשיקול דעתו של הבודק).

ביצוע מהלכים והמחלקה Game

ניתנת לכם המחלקה Game בתוך הקובץ game.py (יודגש כי הרצת הקובץ game.py לבדו אינו אמור לעשות דבר). מחלקה זו תייצג את "הלוגיקה הפנימית" של מצב הלוח, ותאפשר לבצע מהלכים על הלוח. אתם רשאים לעצב ולשנות מחלקה זו (התנהגות פנימית, משתנים פנימיים, מתודות וכו'), אולם עליכם לשמור על מספר אלמנטים במחלקה בדיוק כפי שמופיעים כאן:

1. המחלקה תשאיר את הקבועים (Game.PLAYER_ONE, Game.PLAYER_TWO ו-Game.DRAW) כמות שהם. תוכלו להוסיף קבועים נוספים במידת הצורך.
2. **חתימת בנאי (Consructor)** המחלקה הינה **__init__(***self***)**. ניתן להוסיף פקודות לבנאי, אך אין לשנות את חתימתו.
3. **המחלקה תכיל את המתודה make_move(***self, column***)** – ביצוע מהלך, בין אם על ידי ממשק המשתמש, או אם התקבל מהשחקן היריב, **חייב** להתבצע דרך קריאה למתודה זו, המקבלת את העמודה בה רוצים להציב דיסקית (שימו לב כי המתודה צריכה להסיק מי השחקן הנוכחי, בהתאם להתקדמות המשחק). במידה ולא ניתן להציב דיסקית (כיוון שהעמודה הנבחרת מלאה, מחוץ לתחום, או כיוון שהמשחק הסתיים), עליכם להעלות exception עם ההודעה: "Illegal move.".
4. **המחלקה תכיל את המתודה get_current_player(***self***)** – מחזירה את השחקן הנוכחי.
5. **המחלקה תכיל את המתודה get_player_at(***self, row, col***)** – מחזירה איזו דיסקית נמצאת ב-[*row, col*] (Game.PLAYER_ONE, Game.PLAYER_TWO או None). שימו לב כי הפינה השמאלית העליונה בלוח, מוגדרת להיות כבעלת הקואורדינטות row=0, col=0.



- **המחלקה תכיל את המתודה get_winner(***self***)** – קריאה למתודה זו בכל שלב במשחק (גם לא בסופו) תחזיר את "מצב הניצחון": Game.PLAYER_ONE – ניצחון של שחקן א', Game.PLAYER_TWO – ניצחון של שחקן ב', Game.DRAW – תיקו (כלומר,

כאשר הלוח מלא לגמרי ואין מנצח), ואם 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 (כל עוד הן חוקיות). בסופו של דבר, נגיע למצב בו יש מנצח, או תיקו, כיוון שהלוח מלא לחלוטין. בזמן זה, עליכם לזהות כי הסתיים המשחק (למשל, על ידי שימוש ב-get_winner) ולא לאפשר ביצוע מהלכים נוספים באמצעות הממשק הגרפי. ביצוע של מהלך נוסף, כשהמשחק הסתיים, יעלה את ה-exception: "Illegal move.".

תקשורת בין שני שחקנים

שחקן א' ושחקן ב' אינם משתמשים באותה הרצה (instance) של התכנית, אלא עליהם לייצר שתי הרצות ולהתחבר אחד לשני באמצעות תקשורת מבוססת sockets, כפי שראיתם בתרגול (כך, שחקנים הנמצאים על מחשבים שונים באותה רשת פנימית, למשל במחשבי האוניברסיטה או על אותו חיבור wifi, מסוגלים להתחרות זה בזה). כל הרצה בודדת משרתת שחקן אחד, והמשחק מתנהל על ידי "פינג-פונג" הדדי של הודעות בין התכנה המשרתת את שחקן א', לזו המשרתת את שחקן ב', כך שמצב הלוח, הגרפיקה וסיום המשחק מתעדכנים אצל שני השחקנים. יודגש כי ההרצות יכולות להיות על מחשבים שונים, או על אותו מחשב: מנגנון התקשורת זהה.

עבור התקשורת, מסופקת לכם המחלקה Communicator בקובץ communicator.py. המחלקה ממומשת עבורכם, והוסברה בתרגול: היא מכילה מנגנון תקשורת לא חוסם (non-blocking), המיועד להפעלה בתוך ה-mainloop של אובייקט Tk( ). אתם תצטרכו להבין כיצד להשתמש באובייקט באופן שמותאם לתכנה שלכם, אך שימו לב כי המתודות דלעיל מכילות מידע מפורט על אופן השימוש, והדוגמא שראינו בתרגול ("YO") דומה לאופן השימוש הנדרש בתרגיל זה:

- בנאי המחלקה מוגדר כ – **__init__(***self*, *root*, *ip*, *port*, *server***)** , כאשר *root* הינו אובייקט ה-Tk( ) ממנו אתם קוראים ל-mainloop, *ip* הינה כתובת ה-IP של המחשב (אשר ניתנת לגילוי במגוון דרכים, למשל בפייתון: socket.gethostbyname(socket.gethostname( )),

פונקציה המוגדרת במודול socket) ו-*port* הינו מספר בין 1000 ל-65535 (אנו ממליצים לבחור במספרים באזור ה-8000).

- לאחר יצירת ה-Communicator, יש לקרוא למתודה connect(), אשר ,תנסה (באופן לא חוסם) להתחבר לאובייקט Communicator נוסף אשר הוגדר באמצעות אותם ip ו-port. אם ה-Communicator יוצר בתור server, הוא יוכל להתחבר ל-Communicator אחר המוגדר כ-client (לאחר שהשני גם קרא ל-connect( )), ואותו כנ"ל במהופך עבור Communicator שיוצר בתור client (יוכל להתחבר ל-server). סדר ההתחברות (איזה אובייקט יוצר וקרא ל-connect() קודם) אינו משנה.

שימו לב! כדי לתקשר, הכרחי שאחד מה-Communicator-ים יוגדר בתור "שרת" (על ידי אי אספקת כתובת IP) והשני יוגדר כ"לקוח" (על ידי אספקת כתובת ה-IP של השרת).

- שתי פעולות התקשורת הבסיסיות להעברת מידע הינן שליחת הודעה, וקבלת הודעה:
  - <u>שליחת הודעה</u> – מתבצעת על ידי קריאה ל-**send_message(***self, message***)** ובה הודעה נשלחת ל-Communicator מקושר, לאחר שהחיבור הראשוני התבצע. עם קבלת ההודעה, ה-Communicator המקושר יטפל בה (ראו הסעיף הבא).
  - <u>קבלת הודעה</u> – נעשית על ידי המתודה **bind_action_to_message(***self, func***)**. למתודה זו קוראים **פעם אחת** לאחר ייצור ה-Communicator, והיא "קושרת" פעולה שאתם מגדירים (*func*), כפונקציה מסדר שני, לאירוע של קבלת הודעה (בדומה לדרך בה event handlers נקשרים לאירועים כגון "לחיצה" או "כניסה" ב-tkinter). כלומר, בכל פעם שתתקבל הודעה, ה-Communicator ידאג להפעיל את *func* עם פרמטר בודד, שהוא תוכן ההודעה (כמחרוזת). למשל, ניתן לקשור פונקציה של "ביצוע מהלך בלוח" למתודה זו – ואז, כשתתקבל הודעה, שתוכנה הוא מהלך מסויים שביצע היריב, המהלך יתבצע גם בלוח הנוכחי.
    כמובן, שאם רוצים להחליף את הפעולה שתתבצע עם קבלת הודעה, יש לקשור פונקציה אחרת, על ידי קריאה נוספת ל-bind_action_to_message().

שימו לב! ניתן לבצע פעולות אלו רק לאחר היצירה וההתחברות!

דגש א': בכל התחלת משחק, **השחקן המתחיל הוא תמיד זה שבתכנת השרת**. שחקן זה יהיה השחקן שיזוהה עם Game.PLAYER_ONE באובייקט Game.

דגש ב׳: אין צורך לטפל בשום שגיאות שעשויות לנבוע מבעיות תקשורת גרידא, בהן סגירה של צד אחד באופן מפתיע (כולל באמצע משחק) וכל התנהגות שתבחרו במקרה זה תתקבל (לרבות קריסת התכנה). עם זאת, כן עליכם לטפל במקרה בו מתקבלות הודעות שלא על פי הפרוטוקול, כפי שמוסבר בסעיף הבא.

פרוטוקול התקשורת במשחק

על מנת שהמשחק יוכל להתנהל בצורה תקינה בין שני שחקנים בשתי תוכנות שונות, תצטרכו להחליט על *פרוטוקול תקשורת* – זהו הפורמט של ההודעות – **כמחרוזות** – שנשלחות בין תכנה אחת לשנייה, ומודיעות על שינוי. נסו לממש את הפרוטוקול באופן הפשוט ביותר. העבירו רק מידע הנצרך להעברה בין השחקנים!

הפרוטוקול אמור לתת מענה לשתי שאלות:

1. איזו הודעה לשלוח (הפורמט והתוכן).
2. מתי לשלוח.

לדוגמא, חלק ממהלך משחק (לאחר ההתחברות) יכול להיראות כך:

- <שחקן א׳ שם דיסקית בעמודה 2>
- <מצב הלוח והמשחק מתעדכן אצל שחקן א׳>
- <שחקן א׳ שולח את ההודעה: ״שמתי דיסקית בעמודה 2״>
- <שחקן ב׳ מקבל את ההודעה: ״שמתי דיסקית בעמודה 2״, ויודע שלפי הפורמט, העמודה בה שחקן א׳ בחר לשים דיסקית נמצאת בתו האחרון בהודעה: 2>
- <מצב הלוח והמשחק מתעדכן אצל שחקן ב׳>
- <שחקן ב׳ שם דיסקית בעמודה 0>
- <מצב הלוח והמשחק מתעדכן אצל שחקן ב׳>
- <שחקן ב׳ שולח את ההודעה: ״שמתי דיסקית בעמודה 0״>
- ...

שימו לב א׳ – אם יש מצב שבו אחת התוכנות שולחת שתי הודעות אחת לאחר השנייה (לדוגמא: הודעה x ולאחר מכן הודעה y), ייתכן כי ההודעות תתקבלנה ביחד: xy (כלומר, x ומיד לאחריה y, באותה מחרוזת). עליהם לדאוג לטפל במצב זה בעצמכם.

שימו לב ב' – אנו מספקים לכם רק את האובייקט Communicator, אולם עליכם להחליט לבד מה פורמט ההודעות, ולממש את לוגיקת המשחק בצורה כזו שתשתמש ב-Communicator ובהודעות בצורה שמתאימה לחוקי המשחק: למשל, לא יכול להיות מצב בו שחקן מקבל שתי הודעות "מהלך" ברצף, מבלי שביניהן הוא ביצע מהלך ושלח אותו, לא יכול להיות מצב בו שחקן מקבל או שולח הודעה על מהלך שאינו תקין (למשל, עמודה לא קיימת או עמודה מלאה). האחריות לממש מהלך משחק תקין, על פי החוקים שתוארו לעיל, היא שלכם.

בינה מלאכותית

עליכם לממש את המחלקה AI בתוך הקובץ ai.py (הרצת קובץ זה לבדו אינה אמורה לעשות דבר). המחלקה אמונה על ביצוע מהלכים על ידי המחשב. אנו מצפים מכם לבצע מימוש בסיסי ביותר לחלק זה, אך מימושים מתוחכמים יותר, עשויים לזכות אתכם בבונוס (ראו לעיל).

תוכן המחלקה ופעולתה נתון לשיקולכם, אך אתם חייבים לממש מתודה אחת בצורה הבאה:

find_legal_move(*self, game, func, timeout=None***)** –מקבלת את אובייקט Game המתאר את לוגיקת המשחק ומחשבת מהלך חוקי (מהלך, פירושו להחזיר אינדקס בין 0 ל-6, אשר מייצג עמודה שאיננה מלאה). אזי, המתודה מפעילה את הפונקציה *func*, כפונקציה מסדר שני, על אותו מהלך שהיא מצאה. במידה ולא נמצא מהלך אפשרי (כל העמודות מלאות), על המתודה להעלות exception עם הכיתוב "No possible AI moves.". **אין לכם צורך להתייחס לפרמטר *timeout*, אלא אם כן תבחרו לממש את הבונוס (ראו לעיל).**

כיצד ניתן למצוא מהלך? לשיקולכם. עם זאת, המימוש הנאיבי **והמינימום הנדרש**, הינו למצוא מהלך רנדומאלי מבין העמודות התקניות הנוכחיות. שימו לב שהעובדה שהמתודה מקבלת פונקציה מהווה בחירה תכנונית שעליכם לעבוד איתה – למשל, ניתן להעביר למתודה את הפונקציה שמבצעת מהלך ובכך לבצע את המהלך מיד עם מציאתו.

<u>נבהיר שוב את דרך השימוש</u>: יצירת אובייקט Game ושימוש בבינה המלאכותית (ללא תצוגה גרפית בדוגמה זו – תצטרכו להוסיף זאת בעצמכם), אמורה לבצע מהלך תקין בלוגיקת המשחק:

```
g = Game()
ai = AI()
# makes a single move, for player one, based on the AI.
ai.find_legal_move(g, g.make_move)
```

בעת הרצת המשחק (ראו לעיל), ניתנת אפשרות לבחור האם השחקן יהיה אנושי, או ממוחשב (ניתנת אפשרות בחירה גם כשמריצים את תכנת השרת וגם את תכנת הלקוח, כלומר קיימת אפשרות לשחק במצבים: אנושי-אנושי, מחשב-אנושי, אנושי-מחשב ומחשב-מחשב). במידה ונבחר שחקן ממוחשב, **אין לאפשר לשחקן האנושי לבצע אף מהלך ידנית, אך כן יש להציג את הלוח וביצוע המהלכים בצורה גרפית**: בכל פעם שיידרש להתבצע מהלך, נוכל לקבל אותו על ידי קריאה למתודה find_legal_move() של אובייקט AI.

הרצת המשחק

כאמור, על מנת לשחק יש לייצר שתי הרצות (instances) של התכנה במקביל, אחת כ-server והשנייה כ-client. ניתן להריץ גם בשני מחשבים שונים על אותה רשת פנימית (למשל, במעבדות). הטיפול ב-main של התכנה צריך להתבצע בקובץ four_in_a_row.py, וההרצה הינה כדלקמן:

four_in_a_row.py is_human port <ip>

כאשר:

is_human = *human* if the player is human, *ai* if the player is AI.

port = The port number used for communication (e.g., 8000)

הפרמטר האחרון, ip, ניתן רק בתכנת הלקוח, ומתאר את כתובת המחשב אליו מתחברים (בתכנת השרת, כתובת זו מזוהה אוטומטית ככתובת של המחשב ברשת, ולכן אינה נדרשת).

כלומר, הרצה פוטנציאלית יכולה להיות:

python3 four_in_a_row.py ai 8000

ובמקרה זה אנו מרימים תכנה שבה ה-Communicator ישמש כשרת, התקשורת תנוהל ב-port 8000, ובה המשחק ישוחק על ידי המחשב. כדי להתחיל במשחק, נצטרך עתה לייצר תכנת לקוח (באותו מחשב, או מחשב אחר), למשל על ידי:

python3 four_in_a_row.py human 8000 10.0.0.7

ובמקרה זה, צד הלקוח ישוחק על ידי שחקן אנושי, והחיבור יתבצע ל-ip וה-port שתוארו.

שימו לב כי ניתן לדעת האם התכנה היא תכנת לקוח או שרת, בהתבסס על מספר הארגומנטים שניתנו לה בשורת ההפעלה. במידה וניתנים ארגומנטים שאינם תקינים, על התכנית לא לקרוא לממשק הגרפי, ולהדפיס ל-console: "Illegal program arguments." (יש צורך לבדוק שניתנו מספר תקין של ארגומנטים ושמספר ה-port הוא עד 65535, אך אין צורך לבדוק את צורת כתובת ה-IP, במידה וניתנת).

טיפים

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

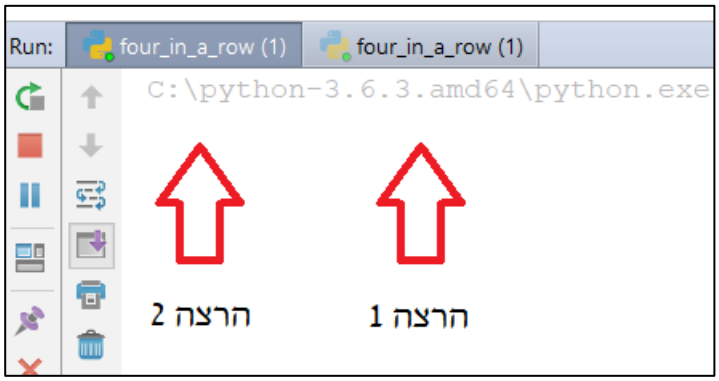
כפועל יוצא, שווה מאוד להבין שאפשר לכתוב חלק ניכר מכל מודול, ולבדוק שהוא עובד, מבלי לערב בכלל את המודולים האחרים (אחד מהיתרונות הגדולים של תכנות מונחה עצמים: מודולריות ואנקפסולציה). למשל, ניתן לכתוב את מודול הגרפיקה, מבלי שיתממשק (בינתיים) ללוגיקת המשחק – רק לבדוק שניתן ללחוץ ולהציב דיסקיות, גם אם ההשמה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

אנו ממליצים (אך כמובן, אין זו חובה) להתחיל ממימוש של Game ולוגיקת המשחק: נסו לכתוב תכנה פשוטה, ללא תקשורת server-client, שרצה על מחשב אחד ומסמלצת 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, ללא

תקשורת server-client (באותו מחשב) ללוגיקת המשחק, ולבסוף שלבו את התקשורת וסגרו פינות אחרונות ומקרי קצה (לא לאפשר מהלכים שלא בזמן, וכו').

2. בשלב של מימוש ה-GUI, עדיין ניתן להדפיס ל-console על מנת לבצע בדיקת תקינות וניפוי שגיאות: "האם קיבלתי הודעה?", "איזו הודעה קיבלתי?", "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).

3. מאוד פשוט להריץ פעמיים את התכנה באותו מחשב: לדוגמא, אם אתם משתמשים בסביבת PyCharm (עליה המלצנו בתחילת הקורס), ניתן להריץ את four_in_a_row.py פעמיים, ולכל הרצה יווצר console משלה, בו אתם יכולים להדפיס הודעות בדיקה (אך זכרו להוריד אותן בסוף). ניתן להחליף בין ה-console-ים השונים בקלות, כיוון שהם בטאבים נפרדים בחלק התחתון של התכנה, כמתואר בתמונה משמאל.



אם אתם משתמשים בסביבות פיתוח אחרות, אפשרויות דומות קיימות. בלינוקס, ניתן למשל פשוט להריץ את התכנה פעמיים, משני terminal-ים שונים.

4. אם תרצו לבדוק את ביצועי הבינה המלאכותית של התכנה שלכם מול זוגות אחרים, זה קל מאוד: פשוט הריצו את התכנה שלכם, כשחקן ממוחשב בתכנת שרת, ובקשו מהזוג הנוסף להריץ תכנת לקוח (עם ה-IP במחשב בו אתם מריצים), גם כשחקן ממוחשב. המשחק אמור לרוץ אוטומטית ובסופו תראו מי ניצח.

May the odds be ever in your favor.

נהלי הגשה

<u>הגשת התרגיל הינה בזוגות</u>.

יש להגיש את הקבצים **four_in_a_row.py**, **game.py**, **ai.py**, את **communicator.py** (שאנחנו מימשנו עבורכם), וכן כל קובץ אחר בו עשיתם שימוש במהלך הקוד. כמו כן, 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** עד לתאריך **24/01/2018** (יום רביעי) בשעה 22:00.

ניתן להריץ קוד pre-submit על מנת לבדוק את תקינות הקבצים בקישור הבא:

```
python3 ~intro2cs/bin/students_presubmit/ex12.py ex12.zip
```

כאשר ex12.zip הוא קובץ ה-zip של התרגיל שלכם.

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 עם הפרמטרים הנדרשים (תצטרכו להריץ פעמיים, פעם עבור שרת ופעם עבור לקוח, פעם בלי IP, ופעם עם IP).

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': זכרו – קוד ה-presubmit בודק רק את הימצאותם של כל הקבצים הרלוונטיים, עם כמה בדיקות מינוריות) נוספות, אולם הוא אינו בודק את תקינות הקוד: בפרט, הוא אינו בודק שהקוד רץ, ואינו בודק שלוגיקת המשחק עובדת כהלכה. עליכם לבדוק זאת בעצמכם, ואנו ממליצים לכתוב בדיקות אוטומטיות (unit tests) משלכם טרם הגשת התרגיל. אתם רשאים לחלוק ביניכם בדיקות אוטומטיות שכאלו (אך כמובן, לא את קוד התרגיל עצמו). השיתוף הנ"ל לא ייחשב כעבירה על חוקי הקורס.

בהצלחה!

בונוס

תוכלנה לנסות ולממש בינה מלאכותית חכמה בכל דרך שתבחרנה, תוך שימוש באותה מתודה find_legal_move( ) אשר הוגדרה לכן לעיל. כשנבדוק את המימושים שלכן, נשתמש במתודה זו בלבד, בצירוף אובייקט Game שייצרתן, ונסמלץ משחקים תוך שימוש בפרמטר timeout משתנה: המשמעות היא, שאנחנו נאכוף את ה-timeout במובן הזה שניתן לאלגוריתם שבתוך find_legal_move( ) לרוץ עד שיעבור אותו קבוע זמן (אינכן *חייבות* להשתמש בקבוע הזה במהלך הקוד כיוון שהוא ייאכף חיצונית, אך ייתכן כי תרצו להתחשב בו).

<u>מה קורה אם עובר ה-timeout והמתודה לא "סיימה"? האם לא מוחזר מהלך?</u>

כאן, נכנסת לפעולה הפונקציה func – שאותה אנחנו נספק. אתן כותבות את מתודת הבינה המלאכותית איך שתבחרנה. היא צריכה "לייצר" מספר מהלכים פוטנציאליים במהלך הריצה, ואולי לשפר אותם ככל שתתקדם: למשל, בהתחלה, המהלך הטוב ביותר הוא עמודה 2, אבל לאחר מספר בדיקות נוספות, המתודה החליטה שהמהלך בעמודה 5 הוא טוב יותר. בכל פעם שהגעתן למהלך (תקין!) פוטנציאלי **טוב יותר**, עליכן לקרוא ל-func עם המהלך הזה כפרמטר: בדוגמא דלעיל, נקרא לפונקציה פעמיים: func(2) ו-func(5). המימוש שלנו את func יגרום לה "לזכור" את המהלך האחרון שהיא הופעלה איתו. לאחר שיעבור ה-timeout, נשתמש במהלך האחרון שקראתן ל-func איתו: אם לא מצאתן מהלך, נבחר עבורכן מהלך אקראי.

<u>ומה הבונוס?</u>

1. <u>הסבר בעת ההגשה</u>: אם בחרתן לממש מתודה של בינה מלאכותית בדרך שאינה טריוויאלית ובעלת היגיון, עליכן לכתוב תיאור **קצר** (עד 200 מילים) ב-README, עבור הבודקת. 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, תוך שימוש בקבועי זמן שנחליט (ייתכן כי נבחר מספר קבועי זמן). **20 התוכנות בעלות הבינה המלאכותית הטובה ביותר (בעלות מספר הנצחונות הגבוה ביותר), תזכינה את כותבותיהן ב-7 נקודות בונוס נוספות בתרגיל וחמש הטובות ביותר, ב-12 נקודות בונוס** (כלומר, תרגיל מושלם בקוד וב-README, מושקע גרפית ושהצליח להתברג בין 5 התכנות החכמות ביותר, יכול לקבל ציון של עד **120 נקודות** ☺).