תכנות מתקדם 1 – תרגיל מס' 3

בתרגיל זה נוסיף אפשרות לשחק כנגד המחשב (שחקן AI).
מעתה אתם עובדים בזוגות. תחילה עליכם להחליט על איזה Design של המערכת לבסס את המשך הפיתוח של הפרויקט. דונו בשני העיצובים שכל אחד מכם בחר בתרגיל 2 – מהם ההבדלים בין העיצובים? מהם היתרונות/חסרונות של כל אחד מהם? ייתכן ותרצו לשלב חלקים שונים משני הפרויקטים שכתבתם וליצור פרויקט חדש משופר. צרפו קובץ המתעד את ההחלטות שקיבלתם.
בנוסף מעתה עליכם לעבוד עם כלים החיוניים לעבודת פיתוח בצוות – עבודה עם מנהל גירסאות ובדיקות תוכנה (מוסבר בהמשך).

תיאור שחקן ה- AI

בתחילת התוכנית יוצג למשתמש תפריט שיאפשר לו לבחור האם לשחק מול שחקן רגיל (Human player) או כנגד המחשב (AI player).
לשם הפשטות נניח בשלב הזה שהמחשב תמיד משחק את השחקן הלבן (שחקן ה- O).
נממש אסטרטגיית משחק פשוטה עבור שחקן ה- AI:
בכל תור, שחקן ה- AI יעבור על כל המהלכים האפשריים שלו על הלוח ויתן ציון עבור כל מהלך. הציון של כל מהלך יקבע לפי הניקוד המקסימלי שהיריב יכול להשיג במצב החדש של הלוח. שחקן ה-AI יבחר את המהלך שקיבל את הציון הנמוך ביותר (כלומר המהלך שימזער את הניקוד שהיריב יוכל להשיג בצעד הבא שלו).

פסיאודו-קוד של האלגוריתם:

1. בהינתן מצב הלוח הנוכחי, מצא את כל המהלכים האפשריים של שחקן ה- AI
2. עבור כל מהלך אפשרי *m*:
   a. בצע ב"זיכרון" את המהלך ועדכן את מצב הלוח
   b. עבור כל המהלכים האפשריים של היריב במצב הלוח החדש:
      i. חשב את הניקוד של היריב במצב החדש (כלומר מספר הדיסקיות שלו פחות מספר הדיסקיות של המחשב)
   c. קבע את הציון של *m* כניקוד המקסימלי שהיריב יכול לקבל במצב החדש
3. בחר את המהלך *m* עם הציון הנמוך ביותר.

אסטרטגיה זו היא הבסיס לאלגוריתם ידוע בתחום המשחקים הנקרא minimax, שבו בכל תור שחקן ה-max בוחר בצעד הטוב ביותר בהינתן שבמהלך הבא היריב (שחקן ה- min) יבחר את הצעד הגרוע ביותר עבורו.

לדוגמא, נניח שנתון המצב הבא על הלוח (המתקבל לאחר שהשחקן השחור ביצע את המהלך (3,4) בעמדת הפתיחה):

```
viki@c3po: ~/adv_prog/ReversiClient/Debug
Current board:

 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
1|   |   |   |   |   |   |   |   |
---------------------------------
2|   |   |   |   |   |   |   |   |
---------------------------------
3|   |   |   | X |   |   |   |   |
---------------------------------
4|   |   |   | X | X |   |   |   |
---------------------------------
5|   |   |   | X | O |   |   |   |
---------------------------------
6|   |   |   |   |   |   |   |   |
---------------------------------
7|   |   |   |   |   |   |   |   |
---------------------------------
8|   |   |   |   |   |   |   |   |
---------------------------------
```

עתה תור המחשב לשחק. המהלכים האפשריים שלו הם: (5,3) ,(3,5) ,(3,3).

המהלך (3,3) יקבל את הציון 3, כיוון שהמהלך הטוב ביותר שהשחקן השחור יכול לבצע במהלך הבא שלו הוא לשים דיסקית ב- (3,2), מה שיוביל למצב הלוח הבא:

```
viki@c3po: ~/adv_prog/ReversiClient/Debug
currMove: (3,3)
Current board:

 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
1|   |   |   |   |   |   |   |   |
---------------------------------
2|   |   |   |   |   |   |   |   |
---------------------------------
3|   | X | X | X |   |   |   |   |
---------------------------------
4|   |   |   | O | X |   |   |   |
---------------------------------
5|   |   |   | X | O |   |   |   |
---------------------------------
6|   |   |   |   |   |   |   |   |
---------------------------------
7|   |   |   |   |   |   |   |   |
---------------------------------
8|   |   |   |   |   |   |   |   |
---------------------------------
score: 3
```

במצב הזה לשחקן השחור יש 5 דיסקיות, ואילו לשחקן הלבן יש 2 דיסקיות, ולכן הניקוד של הלוח הוא 3.

לעומת זאת, המהלך (3,5) הוא מהלך גרוע יותר מבחינת המחשב, כי לאחריו אם השחקן השחור יבצע את המהלך הטוב ביותר שלו, הוא יוכל להגיע לניקוד 5 על הלוח:

```
viki@c3po: ~/adv_prog/ReversiClient/Debug
Current board:

 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
1|   |   |   |   |   |   |   |   |
---------------------------------
2|   |   |   |   |   |   |   |   |
---------------------------------
3|   |   |   | X | X | X |   |   |
---------------------------------
4|   |   |   | X | X |   |   |   |
---------------------------------
5|   |   |   | X | O |   |   |   |
---------------------------------
6|   |   |   |   |   |   |   |   |
---------------------------------
7|   |   |   |   |   |   |   |   |
---------------------------------
8|   |   |   |   |   |   |   |   |
---------------------------------
score: 5
```

חישוב דומה יראה שהמהלך (5,3) מוביל לניקוד 3 של השחקן השחור.

לכן כדאי למחשב לבחור את אחד המהלכים (3,3) או (5,3) שמובילים לניקוד נמוך יותר של השחקן השחור מאשר המהלך (3,5). במידה ויש יותר ממהלך אחד עם אותו ציון מינימלי, ניתן לבחור באופן שרירותי את אחד המהלכים.

כמובן שאין צורך להציג למשתמש את החישובים שעשיתם עבור המהלך של המחשב, אלא רק מהו המהלך הסופי שנבחר:

```
viki@c3po: ~/adv_prog/ReversiClient/Debug
 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
---------------------------------
1|   |   |   |   |   |   |   |   |
---------------------------------
2|   |   |   |   |   |   |   |   |
---------------------------------
3|   |   | O | X |   |   |   |   |
---------------------------------
4|   |   |   | O | X |   |   |   |
---------------------------------
5|   |   |   | X | O |   |   |   |
---------------------------------
6|   |   |   |   |   |   |   |   |
---------------------------------
7|   |   |   |   |   |   |   |   |
---------------------------------
8|   |   |   |   |   |   |   |   |
---------------------------------
O played (3,3)

X: It's your move.
Your possible moves: (3,2),(4,3),(5,6),(6,5)

Please enter your move row,col:
```

עבודה עם מנהל גירסאות

אנו מדמים את המציאות בה אנו מתכנתים כחלק מצוות תכנות. אחד האתגרים הוא ניהול העבודה, ובפרט ניהול הקוד. לא נרצה שתדרסו את הקוד של אחד ע"י השני, או שתלכו לאיבוד בין אינסוף גרסאות ששלחתם במייל... כמו-כן, נרצה לשמור גרסאות קודמות כדי שנוכל לחזור לגירסא עובדת במקרה של תקלות או כדי לתמוך במשתמשים בעלי גרסאות קודמות של המוצר שלנו.

לשם כך צוותי פיתוח משתמשים במנהל גרסאות.

הרעיון הוא פשוט: בתחילת יום עבודה מורידים ממאגר הקוד שלנו (היושב על שרת כלשהו) את הגירסא האחרונה. עושים את השינויים שלנו על עותק מקומי. לאחר שאנו בטוחים שהשינוי עובד כראוי אנו מעלים אותו חזרה לשרת ומעדכנים את כולם בקוד שלנו.

עליכם להתחיל לעבוד כמו המקצועניים. כלומר להגדיר מאגר של קוד ב- GIT (https://github.com), ולהיעזר בפקודות ה- command line של git לניהול השינויים בקוד (git push, git pull, git commit וכו'). בד"כ בלינוקס ה- git כבר מותקן כחלק מההתקנה של מערכת ההפעלה.

להלן לינק לסרטון המדגים את הנושא:

https://www.youtube.com/watch?v=HVsySz-h9r4

כעת צרו פרויקט hello world פשוט ותתנהלו מולו עם שינויים בקוד עד שתבינו כיצד לנהוג כשותפים לאותו מאגר קוד. לאחר שהבנתם כיצד לעבוד יחד תוכלו להתחיל את העבודה על המטלה בפרויקט חדש.

בדיקות תוכנה (Unit Tests)

תפקידו של מפתח הוא גם לבדוק את המחלקות שהינו אחראי להן. הוא מתחייב שכל מחלקה שהוא מעלה ל repository היא בדוקה ונמצאה אמינה. מאוחר יותר ה- QA בודק האם החלקים השונים של הפרויקט מדברים זה עם זה כמו שצריך ואין בעיות שנוצרות ביניהם.

בקורס זה אנחנו נעבוד עם ספריית Google Test לביצוע בדיקות התוכנה (https://github.com/google/googletest).

הרעיון הוא שלכל מחלקה חשובה שתכתבו תהיה לה גם מחלקת gtest שבודקת אותה. כך, לאחר שביצענו שינויים בקוד, נריץ תחילה את ריצת הבדיקה, ואם כל הבדיקות "עברו" אז נוכל להריץ את הפרויקט ולהעלות אותו ל repository. במידה ולא עברו, נוכל לפי הבדיקה שכשלה לבודד את התקלה שגרמנו בעקבות השינוי. כך נחסך זמן פיתוח רב.

ישנם הגורסים שאת קוד הבדיקות יש לכתוב עוד לפני שכותבים את המחלקה הנבדקת עצמה. כך, הבדיקה תיעשה ללא ההשפעה של הלך המחשבה שהוביל לכתיבת המחלקה, ועלול להיות מוטעה (גישה זו נקראת Test Driven Development).

הוראות הגשה

1. הגישו קובץ zip אחד שיכיל את כל קבצי המקור וקובץ makefile שיאפשר לבנות את הפרויקט.
2. בנוסף עליכם להגיש קובץ טקסט המתעד את היסטוריית העדכונים שלכם ב- git (ניתן להפיק אותו באמצעות הפקודה git log).
3. תרגיל זה הוא להגשה **בזוגות**.
4. את התרגיל יש להגיש עד ה- 27/11. לא ניתן להגיש באיחור בכלל, אלא בנסיבות המוכרות ע"י המחלקה (מילואים, מחלה וכו'). כל הגשה באיחור דורשת אישור מראש של המרצה.
5. לתרגיל ישנו פורום ייעודי בפיאצה. את כל השאלות יש לשאול אך ורק דרך הפורום במודל. חובה להתעדכן בפורום. כל הנאמר בו מחייב את כולם.
6. הקפידו לכתוב את הקוד ע"פ ה Naming Convention המקובלים של שפת C++. פירוט תוכלו למצוא בקישור הבא:

https://google.github.io/styleguide/cppguide.html

בהצלחה

רועי ונועה