

תיכון עירוני ד' ע"ש פרופ' אהרון קציר

עבודת גמר מדעית בהיקף 5 יח"ל

סמל שאלון 899589

עבודת גמר במדעי המחשב:

**פיתוח תוכנה מלאה סביב משחק הדמקה עם בינה מלאכותית, מערכת
משתמשים, צבירת סטטיסטיקות ועוד**

כותב העבודה : נדב סרגוסי, יב'7

ת.ז. : *****

מנחה אקדמי : עידו גודיס

שנה"ל תשפ"ב

ינואר 2022

תוכן עניינים

| | | |
|------|--|----|
| 1. | הקדמה אישית ותודות..... | 3 |
| 2. | מבוא..... | 4 |
| 3. | סקירת הספרות..... | 5 |
| 3.1. | משחק הדמקה..... | 5 |
| 3.2. | מבוא לתכנות מונחה עצמים בשפת פייתון..... | 7 |
| 3.3. | פיתוח משחקים וממשק משתמש ב-Pygame..... | 12 |
| 3.4. | אלגוריתם ה-minimax..... | 14 |
| 3.5. | מבוא לשפת SQL..... | 22 |
| 3.6. | מבוא לשימוש במסד הנתונים MySQL..... | 25 |
| 4. | פיתוח התוכנה..... | 28 |
| 4.1. | לוח המשחק..... | 28 |
| 4.2. | חיילי המשחק..... | 29 |
| 4.3. | מימוש יכולת המשחק לשני שחקנים..... | 33 |
| 4.4. | פיתוח שחקן AI..... | 34 |
| 4.5. | פיתוח מסך בית עם תפריט ראשי..... | 37 |
| 4.6. | פיתוח מסך הרשמה וכניסה למערכת..... | 39 |
| 4.7. | פיתוח מסך סטטיסטיקות..... | 41 |
| 5. | הצגת הפרויקט הסופי..... | 43 |
| 6. | סיכום..... | 47 |
| 7. | ביבליוגרפיה..... | 48 |
| 8. | נספחים..... | 51 |
| 8.1. | מחלקות נבחרות מן הקוד..... | 51 |

הקדמה אישית ותודות

נחשפתי לתכנות לראשונה לפני כשלוש שנים. מיד התחברתי לרעיונות, למופשטות, למגוון הכלים והאפשרויות ולעוצמה הטמונה ברכישת שפת תכנות. מאז אני עוסק בתכנות (פיתוח תוכנה ובניית אתרים בעיקר) כתחביב רציני ומשמעותי.

במרוצת השנים הלא-רבות הללו, מצאתי כי הדרך הטובה ביותר ללמוד ולהתנסות בתחום היא, איך לא, לפתח דברים בכוחות עצמי. כאשר ישנה מטרה וחזון ברור לתוצר, קל להישאב לתוך התהליך, או במילים אחרות להשקיע ימים רבים של מחשבה, מחקר, למידה ופיתוח.

תחום פיתוח המשחקים הוא בהחלט דרך לממש ולשפר את יכולות המחקר והפיתוח, ולכן בחרתי לבצע עבודת גמר במקצוע מדעי המחשב (5 יחידות לימוד). למעשה, הייתה לי הזכות לעסוק בתחום עניין מושבע שלי באופן רציני, מחויב ואקדמי, ולקבל הכרה בכך מטעם המנחה, בית הספר ומשרד החינוך. אני סבור כי עבודות גמר הן דרך יוצאת דופן לשפר ולהעשיר את החוויה בתיכון – העובדה כי תלמיד יכול לבחור נושא ספציפי הקרוב לליבו, ולבצע עליו מחקר מעמיק, בתוך מסגרת סדורה ותומכת, היא לא פחות מאשר מיוחדת במינה.

נהניתי מאוד לכתוב את העבודה, על כל חלקיה: החל מסקירת הספרות המעמיקה וקריאה וצפייה בחומרים הרלוונטיים ועד הבהייה הממושכת במסך עורך הקוד (IDE).

קריאה מהנה,

נדב

תודות

ברצוני להודות לעידו גודיס, על הנחייה אקדמית, מחקרית ומעשית מקיפה, תמיכה מלאה לאורך כל הדרך וזמינות תמידית. עידו, אתה הוא דמות המורה (והמנחה) האידיאלי עבורי. מסירותך, שקידתך וגישתך למקצוע מדעי המחשב, והוראה ככלל, ראויות לשבח, ומשפרות את חוויית הלימודים בביה"ס עד לכדי אין שיעור. מי יתן וכמה שיותר תלמידים יוכלו להנות מכך. על כן, שמחתי מאוד להמשיך איתך גם בשנה"ל זו. עוד אודה ללימור שיאון, רכזת עבודות הגמר בביה"ס, על ליווי מעמיק ומסור וניהול מקצועי. לימור יקרה, בזכותך מספר כותבי עבודות הגמר בעירוני ד' הולך וגדל, באין מפתיע. הפכת את ענף עבודות הגמר לנגיש, פתוח ומרתק וחשוב להכיר תודה בכך, וכן באהדתך הגלויה לתחום והעבודה הרבה שאת מבצעת מאחורי הקלעים.

מבוא

את עולמנו המודרני כיום, עוטפות אינספור אפליקציות ותוכנות בהן אנו עושים שימוש תדיר וממושך. הטכנולוגיות הנ"ל נוגעות הן בחיינו האישיים – האפליקציות שבטלפון הנייד, התוכנות שבמחשב האישי, במכוניות ובמכשירי החשמל הביתיים (ועוד...) – והן מחוצה להם – מערכות התוכנה של הצבא ומוסדות המדינה, פיתוחים רפואיים מתקדמים בבתי החולים, חלליות ולוויינים וכן שלל דוגמאות נוספות. כולן, ללא ספק, שיפרו וקידמו את האנושות (וכפועל יוצא גם את חיינו האישיים כמובן) בצורה חסרת תקדים. יתרה מכך, בעשור האחרון החלה פרצה פנומנלית בתחום הבינה המלאכותית (Artificial Intelligence). היטיב לתאר זאת אנדרו אנג (Andrew Ng), מהמדענים הבולטים כיום בתחום הבינה המלאכותית, במילים "AI is the new electricity", או בתרגום חופשי: "הבינה המלאכותית הינה החשמל החדש".

ובכן, נשאלת השאלה, כיצד הכל נעשה? כיצד האנושות הגיעה לפיתוחים וטכנולוגיות שכאלה? שאלה מרתקת זו מורכבת למעשה ממגוון שאלות נוספות כגון: כיצד מפתחים אפליקציה? כיצד מפתחים תוכנה? מהי בינה מלאכותית? כיצד עוסקים בבינה מלאכותית? כיצד משלבים בינה מלאכותית בתוכנות ואפליקציות? כיצד שומרים מידע? אילו כוחות טמונים במידע ובינה מלאכותית? אילו כלים טכנולוגיים עומדים לרשותנו בעת הזו?

בעבודתי בחרתי להתייחס לחלק לא מבוטל מהשאלות הנ"ל. כבר כמה זמן שהן שהו במוחי, ובמסגרת העבודה ניסיתי למצוא להן תשובה ולהגיע למסקנות. על כן, בחרתי לפתח תוכנת מחשב בנושא משחק הדמקה, בה יהיה ניתן לשחק בשני שחקנים או נגד המחשב, ובנוסף תהיה מערכת של משתמשים שיוכלו להתחבר ו/או להירשם למערכת ולצבור סטטיסטיקות אישיות במשחקים.

לצורך כך, צללתי לתוך מקורות מידע רבים ומגוונים (בעיקר במרשתת) וביליתי ימים רבים במחקר ובפיתוח של התוכנה. אם ברצונכם לראות קודם את התוצר הסופי (מומלץ), ולאחר מכן להתחיל לקרוא את העבודה, גשו לעמוד 43.

סקירת הספרות (החלק העיוני)

משחק הדמקה

דמקה הינו משחק לוח אסטרטגי, אשר משוחק על לוח בן 64 משבצות ומיועד לשני שחקנים אשר מתחרים אחד נגד השני. מטרת המשחק היא להוריד מהלוח (באמצעות אכילה) את כל אבני השחקן היריב או לחסום אותו - שחקן שנשאר ללא אבנים או מסע אפשרי מוכרז כמפסיד.

משחק הדמקה הוא מהנפוצים בעולם, ועל אף מופשטותו היחסית הוא דורש חשיבה רבה, תחכום ותכנון טקטי. דמקה הוא משחק אסטרטגיה מופשט, כלומר משחק ללא אלמנט מזל, אשר קיימת בו ידיעה מלאה מצד כל השחקנים על מצב המשחק בכל רגע נתון. בהתאם לזאת, תוצאת המשחק נקבעת אך ורק לפי החלטות השחקנים. שחמט ואיקס עיגול הן דוגמאות נוספות למשחקי אסטרטגיה מופשטים.

מקורו של המשחק נמצא במצרים העתיקה, ואף לוחות המזכירים לוחות דמקה משנת 3000 לפנה"ס נתגלו בעיראק. המשחק לא הומצא בגרסתו הנוכחית, אלא התפתח מהמשחק הערבי העתיק "אל-קורקוה".

חוקים ואופן המשחק

על הלוח, מניחים 12 אבנים לכל שחקן (צבע שונה לכל אחד) על המשבצות הכהות שבשלוש השורות הראשונות מכל צד, כך שנשארות במרכז הלוח שתי שורות ריקות. התקדמות האבנים במשחק היא רק על המשבצות הכהות בצורה אלכסונית, קדימה (ביחס למיקום ההתחלתי של כל שחקן). המשחק מתנהל בתורות, כאשר על פי המוסכם מתחיל השחקן הלבן, אחריו השחור וכך לסירוגין עד תום המשחק.

אבני המשחק

תנועה - כל שחקן בתורו מניע אבן-משחק בצורה אלכסונית רק בכיוון היריב ("מעלה"). על המשבצות להיות פנויה מכלים. כלומר, לכל אבן יש שתי אפשרויות תנועה, ימינה ושמאלה, למעט אם האבן נמצאת בשולי הלוח, ועל כן יש לה אפשרות אחת בלבד.

אכילה (דילוג) - מתאפשרת כאשר מימין או משמאל (באלכסון) לאבן ישנה אבן יריב, ומעבר לאותה אבן יריב, באלכסון שבאותו הכיוון, קיימת משבצת ריקה. האכילה מתבצעת על ידי הזזת האבן אל המשבצת הפנויה שמאחורי אבן היריב, והוצאת אבן היריב מהלוח. אם בתום האכילה נוצרת אפשרות לאכילה נוספת, יכול השחקן לבצע אותה ברציפות באותו התור (לא בהכרח באלכסון שבאותו הכיוון). בדומה לתנועת האבנים, לא ניתן לבצע אכילה "אחורה" (כלומר לא בכיוון היריב). בחוגים מסוימים של המשחק, קיימת חובת אכילה, כלומר אם שחקן לא אכל אבן יריב, אותה אבן שהיה באפשרותה לאכול נחשבת פסולה והיא מוצאת מן הלוח.

מלכות (נקראות "מלכים" בגרסאות הלועזיות) - כאשר אבן מגיעה לשורה האחרונה בלוח (ביחס למיקומה ההתחלתי), היא הופכת להיות "מלכה", ומסומנת לרוב על ידי שתי אבני משחק אשר מונחות אחת על השנייה. המלכה, בניגוד לאבנים הרגילות, יכולה לנוע (אלכסונית) ולבצע אכילות לכל כיוון. המלכה יכולה לזוז משבצת אחת בלבד, אם כי בגרסאות שונות של המשחק ביכולתה לזוז מספר בלתי מוגבל של משבצות לאורך אלכסון נתון.

סיום המשחק

המשחק יכול להסתיים בניצחון של אחד השחקנים או בתיקו. שחקן נחשב מנצח במידה ואחד מן המקרים הבאים מתקיים :

- לשחקן היריב לא נותרו כלל אבנים על הלוח.
- לשחקן היריב אין אפשרות לבצע מהלך (מאחר ואבניו חסומות).
- השחקן היריב נכנע / פורש מהמשחק.

תיקו, לעומת זאת, יכול להיות מושג באחת מן הדרכים הבאות :

- כאשר הכלים שעל הלוח אינם מאפשרים ניצחון - למשל שתי מלכות נגד מלכה אחת.
- אם במשך 15 מסעים רצופים (מסע משמעותו שני מהלכים, אחד של כל שחקן) נעו מלכות בלבד, ולא התבצעו אכילות כלל.
- בכל שלב במשחק רשאי כל אחד מן השחקנים להציע תיקו, אשר אל ההצעה יכול השחקן השני להסכים או לסרב.

מבוא לתכנות מונחה עצמים בשפת פייתון

תכנות מונחה עצמים (Object-Oriented Programming, או בקיצור OOP), הוא פרדיגמה (סט מוסכמות) לכתיבת תוכנה, אשר משתמש בעצמים (למעשה אובייקטים) לשם פיתוח תוכניות מחשב. הפרדיגמה מאופיינת במודולריות, מדרגיות (scalability) ואבסטרקציה (הפשטה), ומרחב התוכנה שבה כולל אובייקטים בעלי יחסים היררכיים ביניהם. כל ישות תכנותית במערכת היא אובייקט מסוג כלשהו / מחלקה בעלת מאפיינים ופעולות משלה, הקיימת כיחידה סגורה ועצמאית. פרדיגמת ה-OOP מספקת למתכנת מספר דרכים לארגן, לפשט ולייעל את הכתיבה והפיתוח, בין השאר מכיוון שקיימת הפרדה בין המימוש הפנימי לבין הממשק החיצוני של כל מחלקה. הפרדיגמה היוותה מהפכה בכתיבת תוכנה והחלה לשמש בפיתוח תוכנה החל מראשית שנות ה-80 של המאה ה-20, אך השימוש בשלמותה החל כעשור לאחר מכן. כיום, מרבית שפות התכנות המודרניות תומכות בתכנות מונחה עצמים, והוא נחשב לאחת מפרדיגמות הפיתוח המובילות והנפוצות ביותר בעולם הפיתוח.

מחלקה

מחלקה (class) היא המסגרת הבסיסית ביותר של תכנות מונחה עצמים. מחלקה היא למעשה אוסף של משתנים, תכונות, ופונקציות (הנקראות מתודות / שיטות), אשר מאוגדים למבנה לוגי אחד ופועלים יחדיו. מבחינה מהותית, מחלקה מתארת ישות כללית מופשטת, אשר ניתנת למימוש בפועל באמצעות יצירת עצם (אובייקט) כמופע (instance) של אותה מחלקה. מופע המחלקה מכיל מידע אודות אותו העצם, וכולל מתודות (פונקציות) המאפשרות לבצע פעולות על העצם.

```
class Example:
    pass
```

איור א' – הגדרת מחלקה

באיור א' נראה כיצד מגדירים מחלקה בפייתון: נכתוב את המילה השמורה "class" ומיד אחריה את שם המחלקה שנבחר.

```
ex1 = Example()
```

איור ב' – יצירת מופע

לשם יצירת מופע של המחלקה, ניצור עצם חדש (כאמור מחוץ לגוף המחלקה) על ידי שימוש בשם המחלקה עם סוגריים מיד אחריו, כפי שמתואר באיור ב'.

כרגע, המחלקה ריקה לחלוטין ואינה מכילה דבר. באפשרותנו להוסיף משתנים קבועים למחלקה, שיהיו זהים בעבור כל מופע שלה. לשם כך, נכתוב בגוף המחלקה את שם המשתנה וערכו. אומנם, לרוב נרצה במסגרת המחלקה להגדיר משתניים כללים, שערכם יכול להיות מוגדר באופן ספציפי עבור כל מופע. למשל, נוכל ליצור משתנה כללי בשם "age", ללא כל ערך בגוף המחלקה, אלא שבעת יצירת כל מופע נוכל להעביר למשתנה ערך כלשהו. משתנים אלו נקראים תכונות. לשם כך, נצטרך פעולה בונה (constructor). הפעולה הבונה היא למעשה אותם סוגריים שמצורפים לשם המחלקה בעת יצירה של מופע חדש - דרכה אנו יכולים להעביר את אותם ערכים ייחודיים עבור המופע שיצרנו. הפעולה הבונה מאתחלת את העצם (מופע המחלקה) שיצרנו בזיכרון עם אפשרות לערכים מהמשתמש, ומכינה אותו לשימוש עתידי. בפייתון, נכתוב את הפעולה הבונה באמצעות המילה השמורה `__init__`, ובה נכתוב את הפרמטרים שברצוננו שתקבל עבור העצם.

עבור כל תכונה או מתודה, נשתמש במילת המפתח `self`, אשר למעשה הופכת את המשתנה לתכונה אשר ייחודית לכל מופע של המחלקה. `self` מאפשרת לנו לייצג, לקרוא ולהפנות למופע ספציפי של המחלקה.

```
class GoogleEmployee:
    company_name = "Google LLC."

    def __init__(self, name, job_title):
        self.name = name
        self.job_title = job_title
```

איור ג' – מחלקה לדוגמא עם פעולה בונה

```
emp1 = GoogleEmployee("Yossi", "QA")
```

איור ד' – יצירת מופע עם ערכי תכונות

במחלקה לדוגמא "GoogleEmployee" (איור ג'), ניתן לראות שמוגדר משתנה כללי אשר יהיה קבוע בכל מופעי המחלקה. לעומת זאת, בתוך הפעולה הבונה, ישנן שתי תכונות אשר מוגדרות באמצעות `self`. הפעולה הבונה מקבלת שני פרמטרים, אחד עבור כל תכונה, ומעבירה אותם כערכים לאותן תכונות.

עתה נראה כיצד ניתן ליצור מופע מחלקה עם ערכי התכונות הרצויים: שם המופע, שם המחלקה בצירוף סוגריים (למעשה הפעולה הבונה), ובתוכם את ערכי התכונות הרצויים שבחרנו. כעת (איור ד'), למעשה הגדרנו מופע של המחלקה `GoogleEmployee` בשם `emp1`, אשר תכונותיו (הייחודיות) הן `name = Yossi` ו-`job_title = QA`. כאמור, יש למופע גם משתנה קבוע בשם `company_name`.

```
def __init__(self, name, job_title, level=1):
    self.name = name
    self.job_title = job_title
    self.level = level
```

איור ה' – הוספת תכונה אופציונלית

כעת נוסיף למחלקה תכונה נוספת בשם "level", שתציג את דרגת העובד בחברה. ידוע כי הדרגה הראשונית היא 1, וממנה מתחילים עובדים צעירים בהגיעם אל החברה. אולם, ישנם מצבים בהם מגיעים אל החברה עובדים מנוסים יותר, שדרגתם ההתחלתית גבוהה מ-1. אם כן, כיצד נוכל לממש זאת בפעולה הבונה?

ובכן בפייתון, באפשרותנו להוסיף לפעולה הבונה פרמטרים אופציונליים עם ערך ברירת מחדל, כלומר כאלה שלא תינתן שגיאה במידה ולא העברנו אותם לפעולה בעת יצירת המופע (מימוש – באיור ה'). אם למשל ביצירת `emp1` היינו מעבירים רק ארגומנט אחד, היינו מקבלים הודעת שגיאה.

```
emp2 = GoogleEmployee("Ronit", "R&D", 3)
```

איור ו' – יצירת מופע עם העברת ערך לתכונה level

לאחר ההוספה, ערך התכונה "level" של `emp1` וכל מופע חדש שניצור ולא נעביר לו ערך לתכונה הוא 1, ברירת המחדל. באיור ו' ניתן לראות מצב בו כן נעביר ערך לתכונה.

```
def get_name(self):
    return self.name

def set_name(self, name):
    self.name = name
```

איור ז' – הוספת שתי מתודות

לרוב, בתכנות מונחה עצמים נרצה למנוע גישה ישירה לתכונות אובייקט כלשהו מהמחלקה הראשית, אלא רק באמצעות מתודות ייחודיות למחלקה אליה שייך האובייקט. לכן, עבור כל תכונה במחלקה, ניצור פעולות החזרה ועדכון (Getters & Setters). פעולות ההחזרה כשמן כן הן מחזירות את ערך התכונה הספציפית, ופעולות העדכון מעדכנות את ערך התכונה. לדוגמא, עבור התכונה "name", נוסיף למחלקה את שתי המתודות הבאות (איור ז').


```
print(emp1.get_name())
emp1.set_name("Avi")
```

איור ח' – קריאה למתודות

עתה נראה (איור ח') קריאה למתודות מהמחלקה הראשית - נדפיס את התכונה "name" של המופע emp1 (יודפס "Yossi"), ולאחר מכן נשנה את תכונה זו. אם נחזור על פעולת ההדפסה, יודפס "Avi", הרי זה עתה שינינו את ערך התכונה. מעבר לפעולות ההחזרה ועדכון, נוכל להוסיף מתודות נוספות למחלקה.

ובכן, ראינו כי קיימים שני סוגים של מתודות - כאלה מובנות, כמו הפעולה הבונה __init__, וכאלה שאנו יוצרים, דוגמת get ו-set.

```
print(emp1)
```

איור ט' – הדפסת מופע

פעולות מובנות אלו נקראות מתודות קסם (Magic / Dunder Methods). דוגמא נוספת למתודה כזאת היא המתודה __repr__. מה יקרה אם ננסה להדפיס את מופע המחלקה (איור ט')? נקבל:

```
<__main__.GoogleEmployee object at 0x7f93d19c0970>
```

```
def __repr__(self):
    return f"Name: {self.name}, Job: {self.job_title}"
```

איור י' – הוספת המתודה __repr__

למעשה, מכיוון ש-emp1 הוא אובייקט, קיבלנו הפנייה למקום האחסון של אותו אובייקט בזיכרון המחשב. אולם, באמצעות מימוש המתודה __repr__

במחלקה (איור י'), שמחזירה למעשה ייצוג מחרוזתי מותאם אישית של המופע, נקבל את ההודעה המותאמת אישית, יחד עם ערכי התכונות של המופע emp1:

```
Name: Yossi, Job: QA
```

```
def increment_level(self):
    self.level += 1
```

איור כ' – הגדרת מתודה נוספת

כעת נראה דוגמא אחרונה למתודה לא מובנית אותה נגדיר בעצמנו (איור כ'). כפי שניתן לראות, המתודה "מעלה דרגה" את העובד, בכך שהיא למעשה מוסיפה 1 לתכונה ה-"level".

ככלל, ראינו כי מתודות הקסם המובנות מאופיינות בקו תחתון כפול לפני ואחרי שמן, בעוד מתודות מותאמות אישית מוגדרות כפונקציות רגילות.

```
from GEmpClass import GoogleEmployee
```

איור ל' – ייבוא המחלקה מקובץ אחר

בעבודה אשר מתפרשת על יותר מקובץ אחד, ניתן לייצא ולייבא בקלות מחלקות מקובץ לקובץ, ולהשתמש בהן באופן חופשי. לדוגמא, אם לקובץ בו נמצאת המחלקה קוראים GEmpClass, ואנו נמצאים בקובץ אחר ורוצים לייבא את המחלקה אליו, ניעזר בפקודות השמורות בפייתון, כפי שמוצג באיור ל'. כעת נוכל להשתמש באופן חופשי במחלקה בקובץ הנוכחי.

ירושה

ירושה (inheritance) היא יכולת של מחלקה לרשת את כל התכונות והמתודות של מחלקה אחרת. המחלקה אשר ממנה יורשים נקראת מחלקת העל (Parent / Base / Super Class), והמחלקה היורשת נקראת תת-מחלקה (Child / Derived / Sub Class). לדוגמא, נרצה ליצור מחלקה חדשה בשם GoogleManager עבור מנהל בחברת גוגל. ברצוננו שהמחלקה תכלול את כל תוכן המחלקה GoogleEmployee, בתוספת תכונות נוספות. כאן בא לידי ביטוי מנגנון ההורשה. במקום להעתיק את כל הקוד מן המחלקה המקורית, נכתוב את שמה בסוגריים בעת יצירת המחלקה החדשה, באופן הבא (איור מ'): :

```
class GoogleManager(GoogleEmployee):  
    pass
```

איור מ' – יצירת מחלקה יורשת

```
m1 = GoogleManager("Avi", "Sales")  
print(m1)
```

איור נ' – יצירת מופע חדש והדפסתו

ובכן, המחלקה GoogleManager היא תת-מחלקה של מחלקת העל GoogleEmployee, וביכולתה לרשת את כל תכונותיה ומתודותיה של מחלקת העל. עתה, מבלי להוסיף דבר במחלקה החדשה, ניצור מופע חדש של אותה מחלקה, וננסה להדפיסו (איור נ'). הפלט הינו :

Name: Avi, Job: Sales

למעשה, פייתון אכן מאפשר לנו לעשות זאת; מבלי שהוספנו כלום לתת-המחלקה, כל התכונות והמתודות הורשו אליה, דבר שאיפשר לנו ליצור ולהדפיס מופע חדש.

אומנם, לא לחינם יצרנו את המחלקה החדשה. ברצוננו להוסיף לה תכונות ומתודות חדשות. לצורך כך, נפעל בדיוק באופן ליצירת מחלקה רגילה, באמצעות פעולה בונה (מתודת האתחול __init__). לפעולה הבונה נוסיף את כל הפרמטרים שהפעולה הבונה במחלקת העל GoogleEmployee מקבלת, בתוספת התכונות החדשות הייחודיות לתת המחלקה. בזכות ההורשה, אנו יכולים לקצר את תהליך האתחול, באמצעות שימוש בפונקציה super(). פונקציה זו מדמה את מתודת האתחול של מחלקת העל, כלומר נעביר אליה את הפרמטרים ההכרחיים. לאחר מכן, נוסיף בכתוב הדרוש את התכונות החדשות (איור ס').

```
class GoogleManager(GoogleEmployee):  
  
    def __init__(self, name, job_title, employees, level=5):  
        super().__init__(name, job_title, level)  
        self.employees = employees
```

איור ס' – פעולה בונה בתת-המחלקה

באיור ס', ניתן לראות את מתודת האתחול. הוספנו תכונה חדשה בשם `employees`, הייחודית רק למנהלים. לאחר מכן, עשינו שימוש בפונקציה `super` לצורך אתחול שאר התכונות. בנוסף, התייחסנו לתכונה `level`, והפעם נתנו לה ערך ברירת מחדל של 5, במקום 1. כאמור, כל המתודות גם עוברות בירושה, כך שנצטרך להוסיף פעולות `Get` ו-`Set` רק לתכונה החדשה שיצרנו. גם התכונה הקבועה `company_name` עוברת בירושה. מנגד, התכונה `employees` לא עוברת למחלקת העל והיא אינה קשורה אליה.

פולימורפיזם

פולימורפיזם (`polymorphism`, בעברית - רב צורתיות) הוא עיקרון תכנותי האומר כי יש לקרוא למתודות אשר מבצעות את אותו תפקיד בשם זהה. לדוגמא, הפונקציה המובנית למציאת אורך אובייקטים `len()`, מושתתת על עיקרון זה. לא משנה איזה טיפוס נעביר לה (מחרוזת / רשימה / מילון וכו'), היא עושה את התאמותיה הנדרשות ומחזירה את האורך.

```
def __repr__(self):
    return "Manager."

איור ע' – __repr__ בתת-המחלקה
```

נחזור למחלקה החדשה שלנו, `GoogleManager`. אחת התוספות שנרצה היא, שבעת הדפסת המופע, נקבל בנוסף לשם ולתפקיד הודעה שהעובד שלנו הוא גם מנהל. אזי, נוסיף את מתודת הקסם `repr` במחלקה (איור ע'), וננסה להדפיס את המופע. פלט:

```
Manager.
```

אולם - קיבלנו רק את ההודעה "Manager" במקום לקבל את ההודעה המלאה. זאת מכיוון שלמעשה ביצענו מה שמכונה "דריסה" (`override`) של המתודה - העלמנו את המימוש המקורי של המתודה `repr` ממחלקת העל.

```
def __repr__(self):
    print(super().__repr__())
    return "Manager."

איור פ' – עדכון __repr__
```

כיצד נוכל לפתור זאת? ובכן, אנו תמיד יכולים להעתיק את השורות החסרות, אך מנגנון ההורשה מספק לנו חלופה אפקטיבית בהרבה. בגוף הפונקציה, נבצע קריאה לפונקציה המקבילה לה במחלקת העל באמצעות הפונקציה `super()`, ולאחר מכן את התוספת שלנו (איור פ'). ועתה - אכן הודפסו הנתונים כפי שרצינו:

```
Name: Avi, Job: Sales
Manager.
```

פיתוח משחקים וממשק משתמש ב-Pygame

פייגייס (Pygame) הינה ספריית פיתוח חוצה-פלטפורמות (Cross-Platform), אשר פותחה לצורך כתיבת משחקי מחשב. היא כוללת בתוכה גרפיקה ממוחשבת דו-מימדית וספריות שמע, אשר פותחו במיוחד לשימוש בשפת פייתון.

הספרייה פותחה לראשונה בשנת 2000 ע"י פיט שינרז (Pete Shinnars), אך זמן קצר לאחר מכן הפכה לפרויקט קהילתי. גרסתה הראשונה של הספרייה יצאה ב-28 באוקטובר 2000. עד היום, פייגייס נותרה פרויקט קהילתי חינוכי בעל קוד פתוח לחלוטין, בכתובת <https://github.com/pygame/pygame>. פייגייס נכתבה בעיקר בשפת פייתון, אולם גם בעזרת שפות C, Cython ואסמבלי. נכון לכתיבת שורות אלו, הגרסה היציבה האחרונה, Pygame 2.0, יצאה ב-28 באוקטובר 2020, במיוחד לכבוד יום הולדתה ה-20 של הספרייה.

לצורך ייבוא הספרייה לקוד, ראשית יש להתקין אותה על המחשב באמצעות הפקודה `pip install pygame`. לאחר ההתקנה, נייבא את הספרייה לקובץ הנוכחי באמצעות ההצהרה `import pygame`. מיד לאחר מכן, לצורך אתחול הספרייה, נכתוב `pygame.init()`. כעת, כל הספרייה עומדת לרשותנו.

באיור א' (בעמוד הבא) נראה דוגמא לתוכנית `pygame` בסיסית. בשתי השורות הראשונות ביצענו את הייבוא והאתחול.

בשורה 4, יצרנו את אובייקט החלונות. כשמריצים תוכנית `pygame`, נפתחת חלונות חדשה. כפי שניתן לראות, אנו שומרים אותה בשם "window". כפרמטר לפונקציה זו, מעבירים טאפל (tuple) עם הגובה והרוחב (בפיקסלים) של החלונות.

לאחר מכן, יצרנו דגל בשם `running` ללולאת ה-`while` של התוכנית. כל עוד הלולאה רצה, כך גם חלונות ה-`pygame`. בשורה 8, העברנו לפונקציה `set_caption` את השם שיופיע בעבור כותרת החלונות. בהמשך גוף הלולאה, שזהו חלקה המרכזי של כל תוכנית `pygame`, נטפל בכל מה שקשור לאינטראקציה של המשתמש. בשורה 10, באמצעות הלולאה, ניגש לכל האירועים (events) שיכולים לקרות מבחינת המשתמש. לדוגמא, בבלוק התנאי הראשון, אנו בודקים האם סוג האירוע הוא `QUIT`, כלומר המשתמש לחץ על כפתור האיקס שבחלונות. במידה והתנאי מתקיים, נפסיק את ריצת התוכנית. דוגמא נוספת היא בלוק התנאי השני, אשר בודק אם המשתמש לחץ על העכבר. במידה וכן, יודפס (בפייתון) את קואורדינטות העכבר בעת הלחיצה. לתכונה זו של העכבר הגענו כפי שניתן לראות בעזרת הפונקציה `pygame.mouse.get_pos` על האובייקט `pygame.mouse`.

מחוץ ללולאת ה-`for`, מילאנו את החלונות בצבע שחור. הפונקציה `fill`, שהופעלה על החלונות, מקבלת כפרמטר טאפל בעל ערכי RGB של הצבע המבוקש. בשורה שלאחר מכן, יצרנו מלבן, בעזרת הפונקציה `draw.rect`. פונקציה זו מקבלת כפרמטר את המשטח עליו היא תיצור (החלונות), את צבע המלבן (כרגיל, בטאפל), וטאפל שני נוסף. בטאפל זה, שני האיברים הראשונים הם הגובה והרוחב של המלבן, והשניים הנוספים הם קואורדינטות ה-x וה-y בחלונות, מהן יתחיל המלבן.

בסיום בלוק לולאת ה-while, עלינו לדאוג שחלונית ה-pygame תמשיך להתעדכן בכל רגע, בהתאם למה שקורה. לצורך כך, יש להשתמש בפונקציה pygame.display.update, שכשמה כן היא מעדכנת את המוצג בחלונית בכל רגע. לבסוף, מחוץ ללולאה, סגרנו את התוכנית באמצעות הפקודה pygame.quit.

```
1 import pygame
2 pygame.init()
3
4 window = pygame.display.set_mode((500, 500))
5 running = True
6 while running:
7
8     pygame.display.set_caption("MyGame")
9
10    for event in pygame.event.get():
11        if event.type == pygame.QUIT:
12            running = False
13        if event.type == pygame.MOUSEBUTTONDOWN:
14            print(pygame.mouse.get_pos())
15
16    window.fill((0, 0, 0))
17    pygame.draw.rect(window, (255, 255, 255), (200, 100, 50, 100))
18
19    pygame.display.update()
20
21 pygame.quit()
```

איור א' – דוגמה לתוכנית pygame

לסיכום, בספריית pygame קיימות אינספור פונקציות, פקודות וכלים המאפשרים למתכנת שלל אפשרויות מגוונות בפיתוח ובניית המשחק. ספרייה זו היא מבין הפופולריות ביותר בפיתוח, ולא לחינם. למרות מורכבותה היחסית והרבגוניות שבה, היא קלה מאוד ללמידה והסתגלות תוך כדי פיתוח. ישנם מדריכים רבים ברחבי המרשתת, בתוספת כמובן אתר התיעוד (documentation) הרשמי של pygame, אשר בהחלט מקלים על תהליך הפיתוח.

אלגוריתם ה-minimax

אלגוריתם המינימקס פותח ונהגה לראשונה בתחילת המאה ה-20, ומשפט האלגוריתם ('משפט המינימקס') הוכח רשמית על ידי המתמטיקאי האמריקאי ג'ון ון נוימן בשנת 1928. בבסיסו, אומר המשפט כי כל שחקן שואף למקסם את הרווח המינימלי שלו, או במילים אחרות למזער את ההפסד המקסימלי.

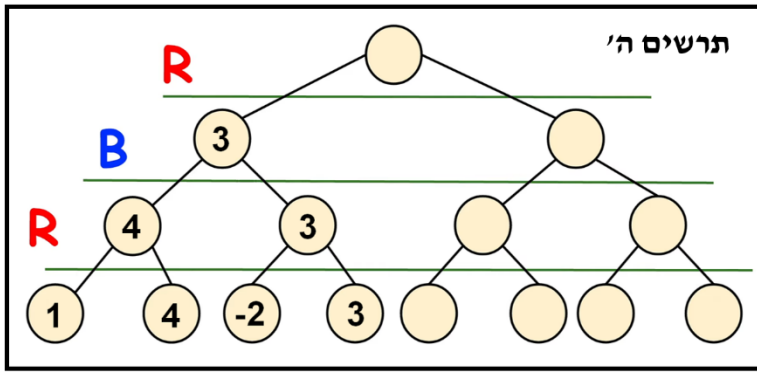
האלגוריתם משמש לרוב לאסטרטגיות במשחקים סכום-אפס לשני שחקנים; משחק בו הרווח של צד אחד מאוזן על ידי הפסדו של הצד האחר, כלומר סכום הרווח וההפסד של שני הצדדים הוא אפס. בכלליות, האלגוריתם פורס את אפשרויות המהלכים של שחקן א', ולכל מהלך נ"ל - את תגובתו של שחקן ב', וכך לסירוגין עד עומק מסוים. כך, נוצר עץ המינימקס, תרשים דמוי עץ הפורס את כלל המהלכים האפשריים בכל שלב של המשחק החל מהעת הנוכחית, ועד לעומק מסוים. עומק זה מוגבל מפאת החישובים הרבים שהוא מבצע, כלומר נדרשים כוחות מחשוביים רבים וזמן רב ככל שהעץ נעשה עמוק וסבוך יותר.

שמו של האלגוריתם נגזר משני השחקנים אשר בהם הוא מתחשב - השחקן המקסימלי (maximizer), והשחקן הממזער (minimizer). מטרתו של השחקן המקסימלי היא, כשמו כן הוא, למקסם את הרווח שלו במשחק. בניגוד ישיר לכך, נמצא השחקן הממזער, שמטרתו היא למזער את הרווח של השחקן המקסימלי.

כעת, נשאלת השאלה כיצד מודדים רווח, הרי זהו העיקרון הבסיסי ביותר עליו האלגוריתם מסתמך. למעשה, במשחקי לוח, לצורך מימוש האלגוריתם, מבצעים הערכה מספרית למצב הלוח הנוכחי, כך שהשחקן המקסימלי שואף שההערכה תהיה פלוס אינסוף, בעוד השחקן הממזער שואף למינוס אינסוף. לדוגמה, ניתן להעריך בצורה בסיסית לוח דמקה ע"פ מספר החיילים של השחקן המקסימלי פחות מספר החיילים של השחקן הממזער: למשל, במידה לשחקן המקסימלי יש יותר חיילים, תהיה ללוח הערכה חיובית, בדיוק כפי ששואף המקסימלי.

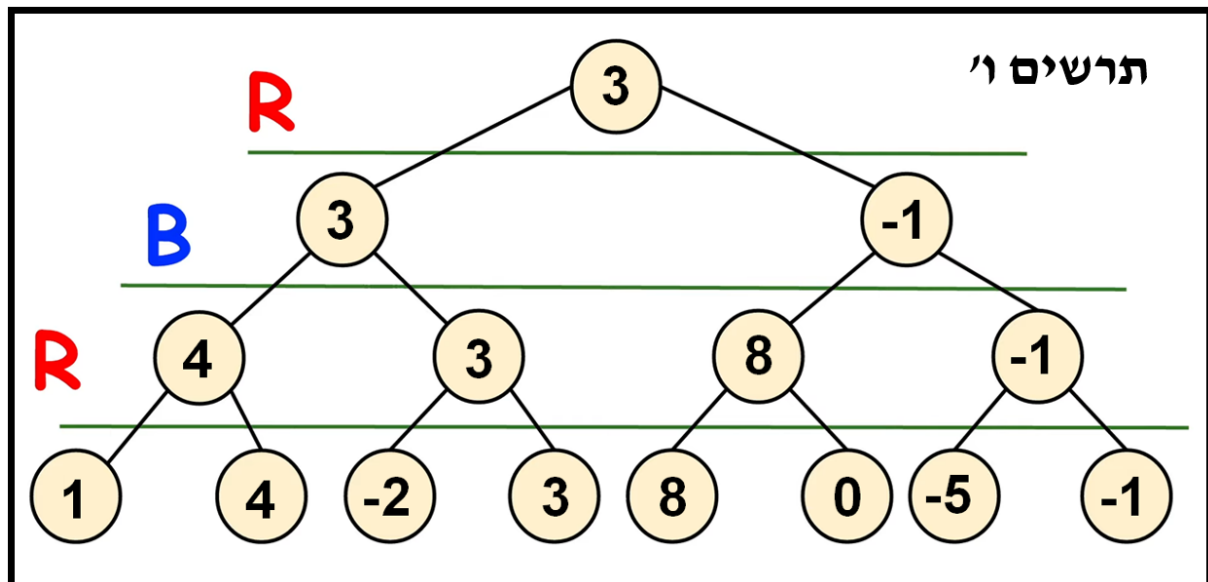
אלגוריתם המינימקס פועל בצורה רקורסיבית לצורך גילוי כל המהלכים האפשריים, ועוצר בשכבה האחרונה בעץ (שכאמור נקבעת ע"פ העומק), וממנה הוא מתחיל לחשב ולהעריך מספרית כל מצב של הלוח.

כך, בוחר האלגוריתם עבור כל שחקן את המהלך הטוב ביותר עבורו (שחקן מקסימלי - לוח בעל הערכה מספרית גבוהה ככל האפשר, ושחקן ממזער - הערכה נמוכה ככל האפשר). כל שכבה בעץ היא של שחקן אחר, ובחירתה מתבססת ע"פ השכבות התחתונות לה, כפי שמיד נראה.



בתרשים ה', כבר סיימנו להעריך את כל הענף השמאלי, שערכו 3. נניח והשחקן האדום יבחר ללכת בענף זה - מגיע תורו של השחקן הכחול. עתה ניצבים מולו 3 ו-4. הוא כמובן בוחר 3, והתור עובר לשחקן האדום, שמולו ניצבים 3 ו-2. הוא בוחר 3 - וכך למעשה הגענו למסקנה שערך כל הענף השמאלי 3.

לצורך סיום ההמחשה, נראה דוגמא למצבו הסופי של העץ (תרשים ו').



האלגוריתם סיים לפרוס את כל המהלכים האפשריים, להעריך את מצב הלוח בכל אחד מהם ולבחור בצורה אופטימלית בעבור כל שחקן. ערכו של העץ, עבור השחקן האדום הממקסם, הוא 3. מניתוח הענף הימני עולה כי ערכו הוא -1, ועל כן השחקן האדום יבחר בענף השמאלי, שערכו גבוה יותר. כמובן שבכמעט כל המשחקים ישנם מספר רב יותר של מהלכים אפשריים בכל רגע נתון, ועץ המינימקס סבוך בהרבה. בנוסף, חשוב לציין כי בעת מימוש האלגוריתם, אין למעשה שימוש במבנה הנתונים עץ, כפי שנראה מיד. עץ המינימקס משמש להסברת והמחשת אופן פעולת האלגוריתם.

להלן הפסאודו-קוד הכללי של אלגוריתם המינימקס.

סימונים לצורך הכתיבה:

1. לכל איבר בעץ נקרא בשם "מצב".
2. "מצב סופי" - משתנה בין משחק למשחק, אך תמיד חשוב לבדוק אותו. מדובר למשל על מצב בו הושג ניצחון לאחד הצדדים, ואין טעם להמשיך לסרוק את המהלכים האפשריים.
3. נשתמש בכתיב: $=$ לצורך השמת ערך במשתנה.

פונקציה מינימקס(מצב, עומק, שחקן_ממקס) {

- **אם** עומק $= 0$ **או** המצב הוא סופי:
 - **החזר** את ההערכה של מצב

- **אם** שחקן_ממקס:

- הערכה $= -\infty$
- **עבור כל** מצב_אפשרי **מתוך** כל המצבים האפשריים הנגזרים ממצב:
 - הערכה $=$ **המקסימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, שקר))
- **החזר** את הערכה

- **אחרת** (כלומר השחקן הממזער):

- הערכה $= +\infty$
- **עבור כל** מצב_אפשרי **מתוך** כל המצבים האפשריים הנגזרים ממצב:
 - הערכה $=$ **המינימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, אמת))
- **החזר** את הערכה

}

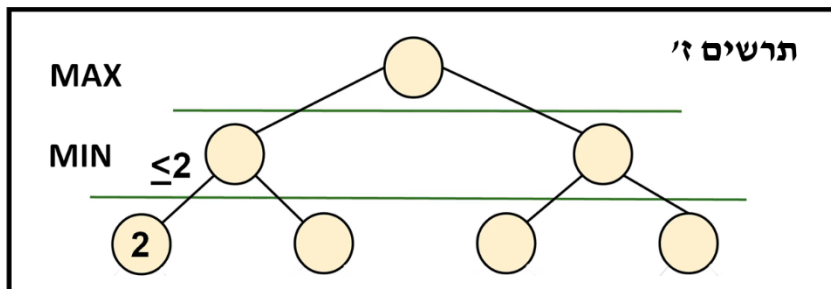
וכך נקרא לפונקציה מתוך המחלקה הראשית:
מינימקס(מצב_נוכחי, עומק, אמת)

עם זאת, אלגוריתם המינימקס מאופיין בסיבוכיות זמן גבוהה מאוד. על כן, הוא לא ריאלי במשחקים בהם ישנם מספר רב של מהלכים אפשריים, בשילוב עומק מינימלי דרוש גבוה, דוגמת שחמט. במידה ו- b מייצג את מקדם ההסתעפות, כלומר מספר המהלכים האפשריים לכל שחקן בתורו (מספר קבוע / ממוצע), ו- d מייצג את עומק העץ, נגיע לסיבוכיות זמן של $O(b^d)$.

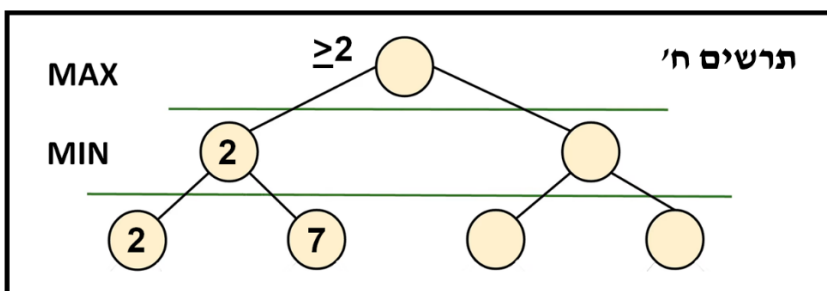
אולם, קיים אלגוריתם אופטימיזציה עבור עצי-חיפוש מסוג מינימקס. אלגוריתם זה נקרא **גזיון אלפא-ביתא** (Alpha-Beta Pruning). מטרתו של האלגוריתם היא לצמצם באופן דרסטי את מספר הענפים בהם יש לחפש ולסרוק. כלומר, במהלך החיפוש יזניח האלגוריתם פתרונות חלקיים אשר כבר ברור כי הם גרועים מפתרונות שכבר נמצאו.

ואכן, לאחר מימוש, סיבוכיות הזמן יורדת בצורה חדה לזמן ממוצע של $O(b^{d/2})$. לשם השוואה, אם במשחק דמקה יש בממוצע 7 מהלכים אפשריים בכל תור, ונרצה לרדת לעומק מהלכים של 5, נקבל 7^5 , כלומר 16,807 לוחות להעריך. אולם אם נשתמש באופטימיזציה הגזיון, נקבל בממוצע $7^{2.5}$, כלומר כ-130 בלבד.

עתה נראה דוגמא לאופן ביצוע הגזיון. הפעם, נשתמש בסימון "MAX" ו-"MIN" בכל ענפי העץ, לייצוג תורם של השחקן הממקסם והממזער, בהתאמה.



נביט בתרשים ז'. נראה כי ערכו של המצב השמאלי ביותר בעץ הוא 2. על כן, השחקן הממזער יודע בוודאות כי בעת הגעה לענף זה, הוא יקבל ערך של "לכל היותר – 2". זאת מפני שאם במצב השני בענף יהיה מספר גדול מ-2, השחקן הממזער יבחר 2. אולם אם יהיה מספר קטן מ-2, הוא יבחר בו במקום.

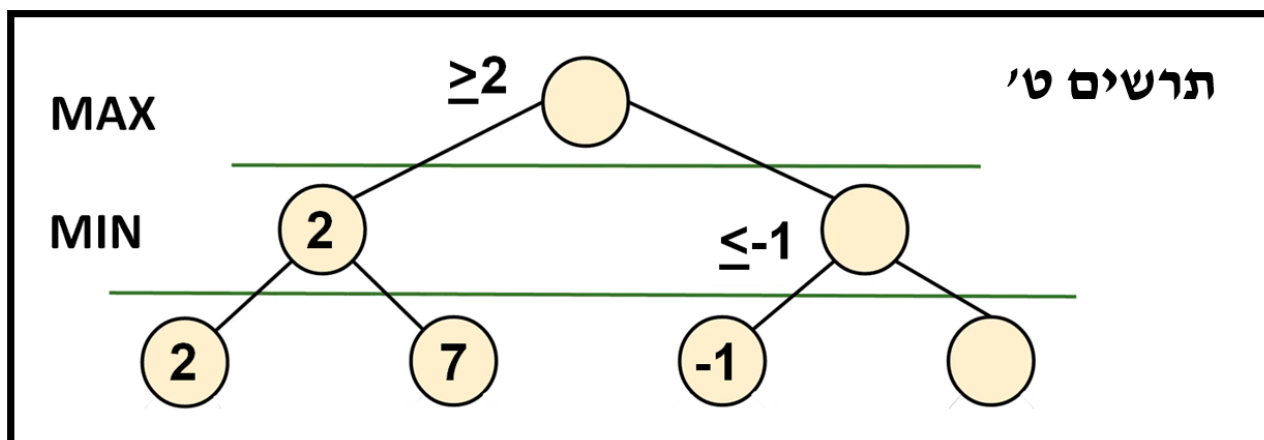


כעת (תרשים ח'), נראה כי ערך המצב השני בענף הוא 7, אז השחקן הממזער יבחר 2, ולכן זו גם הכותרת שניתן לענף זה. ובכן, מכיוון שערך הענף השמאלי בעץ הוא 2, השחקן הממקסם יודע כי לא משנה מה, באפשרותו להשיג ערך של "לכל הפחות – 2".

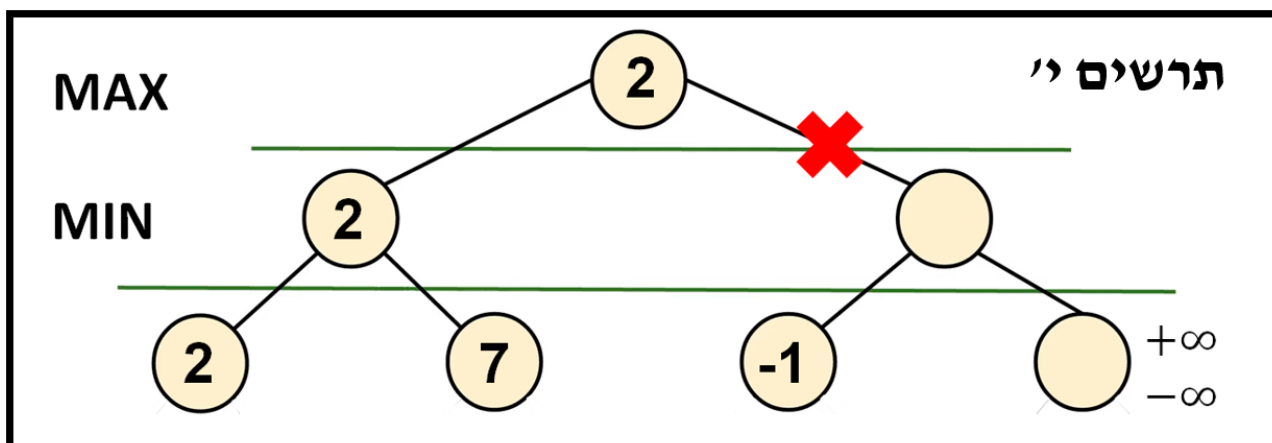
אם בהמשך החיפוש יתברר כי ערך הענף הימני קטן מ-2, השחקן הממקסם יבחר בענף השמאלי, בעוד אם הערך יהיה גבוה מ-2, יבחר בימני.

נמשיך בסריקת העץ ונעבור לתרשים ט'. בתרשים זה, ניתן לראות כי ערך המצב הנוכחי הוא 1. באופן זהה לתרשים א', נוכל להסיק כי מבחינת השחקן הממזער, באפשרותו להשיג בענף ערך שקטן או שווה ל-1. אולם כעת, האם ענף זה בכלל רלוונטי?

התשובה טמונה בעובדה שראינו בתרשים ז' כי השחקן הממקסם יכול להשיג ערך ≥ 2 . כלומר, האם השחקן הממקסם בכלל יגיע למצב כזה, בו הערך יהיה ≤ -1 ? - התשובה היא כמובן שלא, ולכן באותו הרגע הענף הימני הופך ללא רלוונטי לחלוטין.



מתרשים י' נבין בדיוק למה ערך העץ הוא 2, ומדוע ניתן להסיק כי הענף הימני למעשה "לא קיים". נניח שבמצב הימני ביותר בעץ, הלא הוא זה שפסלנו, היה פלוס אינסוף - השחקן הממזער היה בוחר ב-1-, כלומר ערך הענף הימני היה -1. כיוצא בזה, השחקן הממקסם יעדיף 2 על פני -1. באותו אופן, במידה והערך היה מינוס אינסוף - השחקן הממזער היה בוחר בו, כלומר ערך הענף היה מינוס אינסוף, וכמובן, השחקן הממקסם יעדיף את הערך 2 על פני מינוס אינסוף.



לסיכום התרשימים, ראינו דוגמא בסיסית למדי לאופן בו מתבצע הגיזום. בפועל, כאשר העץ סבוך בהרבה, לעתים קרובות מגיעים למצבים בו ענפים שלמים נפסלים בזה אחר זה. לפסילת הענפים יש יתרון כפול - מצד אחד, לא צריך להעריך כל מצב ומצב בענף ולבצע את החישובים הנדרשים. מצד שני, אפילו לא צריך למצוא ולהגיע לאותם מצבים.

מימוש האלגוריתם גיזום אלפא-ביתא כולל כמה תוספות קטנות לקוד של אלגוריתם המינימקס. לצורך כך, נוסיף לפונקציה שני פרמטרים נוספים - אלפא (α) וביתא (β) (מהם נגזר שמו של האלגוריתם). את α נאתחל במינוס אינסוף, והיא תייצג את הערך המינימלי שהשחקן הממקסם יכול להשיג. באופן זה, את β נתאחל בפלוס אינסוף, והיא תייצג את הערך המקסימלי שהשחקן הממזער יכול להשיג. כלומר, שני השחקנים מתחילים בתוצאה הגרועה ביותר האפשרית עבורם.

אם במהלך החיפוש והסריקה יימצא ערך גדול מ- α עבור השחקן הממקסם, אותו הערך יושם ב- α . באותו אופן, אם יימצא ערך קטן מ- β עבור השחקן הממזער, אותו הערך יושם ב- β .

במהלך ריצת האלגוריתם, אם מתקיים $\alpha \geq \beta$, באותו הרגע ניתן להחשיב את הענף הנוכחי כלא רלוונטי. במילים אחרות, ברגע שהשחקן הממזער יכול להשיג ערך שקטן מהערך המינימלי הנוכחי שהשחקן הממקסם יכול להשיג, זהו מצב שהשחקן הממקסם לעולם לא יגיע אליו.

תרשים ד' מהעמוד הקודם ממחיש זאת בצורה הטובה ביותר. כשהגענו למצב שערכו היה -1, אלפא הייתה 2, וזה עתה עדכנו את ביתא ל-1. ברגע שהתקיים $-1 \geq 2$, או בכלליות $\alpha \geq \beta$, פסלנו את הענף.

(בעמוד הבא נמצא הפסאודו-קוד של אלגוריתם המינימקס בשילוב גיזום אלפא-ביתא)

להלן הפסאודו-קוד של אלגוריתם המינימקס בתוספת אלגוריתם האופטימיזציה גיזום אלפא-ביתא :

פונקציה אלפא_ביתא(מצב, עומק, α , β , שחקן_ממקס) {

• **אם** עומק == 0 **או** המצב הוא סופי :

○ **החזר** את ההערכה של מצב

• **אם** שחקן_ממקס :

○ הערכה = $-\infty$

○ **עבור כל** מצב_אפשרי **מתוך** כל המצבים האפשריים הנגזרים ממצב :

▪ הערכה = **המקסימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, שקר))

▪ α = **המקסימום מבין** (α , הערכה)

▪ **אם** $\alpha \geq \beta$:

▪ **הפסק**

○ **החזר** את הערכה

• **אחרת** (כלומר השחקן הממוזער) :

○ הערכה = $+\infty$

○ **עבור כל** מצב_אפשרי **מתוך** כל המצבים האפשריים הנגזרים ממצב :

▪ הערכה = **המינימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, אמת))

▪ β = **המינימום מבין** (β , הערכה)

▪ **אם** $\alpha \geq \beta$:

▪ **הפסק**

○ **החזר** את הערכה

{

וכך נקרא לפונקציה מתוך המחלקה הראשית :

אלפא_ביתא(מצב_נוכחי, עומק, $-\infty$, $+\infty$, אמת)

מבוא לשפת SQL

רקע כללי

שפת SQL, או בשמה המלא "שפת שאילתות מובנית" (Structured Query Language), הינה שפת תכנות הצהרתית* אשר משמשת לטיפול ועיבוד מידע במסדי נתונים יחסיים (Relational Databases). השפה פותחה ע"י חברת IBM בהובלת דונלד צ'מברליין וריימונד בויס בתחילת שנות ה-70, במקביל לפיתוח מסד הנתונים היחסי הראשון באוניברסיטת MIT. בתחילה, השפה נקראה SEQUEL, אך מאוחר יותר שמה הוחלף ל-SQL מטעמים מסחריים.

כאמור, תפקיד השפה (בעת פיתוחה) היה תשאול ומניפולציה על נתונים במסגרת מסדי נתונים יחסיים. מהר מאוד חברות ותאגידים שונים זיהו את הפוטנציאל הגלום בשפה וברעיונותיה, וב-1986 אומצה SQL כסטנדרט ע"י מכון התקנים האמריקני (ANSI) ושנה לאחר מכן ע"י ארגון התקינה הבינלאומי (ISO).

*שפת תכנות הצהרתית הינה שפה המבטאת את הלוגיקה החישובית שבה ללא בקרת זרימה, אלא דרך אוסף של פקודות והוראות בזו אחר זו. כלומר, השפות הללו למעשה מתארות אך ורק **מה** צריך לבצע, ולא כיצד לבצע זאת. מסי' דוגמאות לשפות כאלו הן HTML, CSS, XAML וכן SQL.

SQL מבוססת על טבלאות, ועיקר עבודתה מתבצעת עם נתונים במודל טבלאי רלציוני (בעל קשרי גומלין). על אף שמה, השפה איננה רק שפת שאילתות, אלא למעשה שפת הנתונים המקיפה ביותר, המאפשרת קשת רחבה של עבודה מול מסדי נתונים: החל מיצירת טבלאות סטטיות, אינדקסים, יצירת טבלאות דינמיות מדומות, הגדרת אילוצים עסקיים ועוד. SQL עושה זאת דרך קריאת נתונים בדרכים מגוונות, הכוללות פעולות בין טבלאות (כגון צירוף נתוני טבלאות ופעולות איחוד וחיתוך בין נתוני טבלאות), וכלה במניפולציה על נתונים כמו הוספת רשומות, עדכון ומחיקתן, ואף אבטחת נתונים וניהול תנועות.

כזכור, SQL איננה שפת תכנות מלאה, כיוון שחסרים בה פקודות לוגיות ומשפטי בקרה, ולכן יש בה קשיחות שלרוב צריכה להיפתר על ידי אירוחה בשפת תכנות עילית. בשפה קיימות מילים שמורות (שאין להשתמש בהן בשמות טבלאות / שדות / אינדקסים), טיפוסים קבועים, משתנים, אופרטורים ופונקציות מובנות.

על אף שהשפה איננה רגישה לגודל אות (Case Insensitive), נהוג לרשום את הפקודות (שנקראות Clauses) באותיות גדולות, ולפצל פסוקיות לשורות נפרדות. אולם, אין זאת חובה מבחינת תחביר השפה, אשר לה משנה רק סדר הפסוקיות. בנוסף, כל משפט צריך להסתיים בנקודה-פסיק ;.

בניגוד לשפות תכנות עיליות, סדר הביצוע של משפט SQL איננו מתבצע לפי סדר הכתיבה. בדרך כלל פסוקית SELECT תתבצע לקראת הסוף, כאשר הפסוקית הראשונה שתתבצע תהיה פסוקית FROM. כל שלב בעיבוד מייצר טבלה וירטואלית זמנית אשר מהווה את הקלט לשלב הבא. בשאילתת SELECT מורכבת במיוחד (שיש בה צירוף בין טבלאות, תנאי סינון, הקבצה, מניעת כפילויות וצמצום מספר הרשומות למספר מסוים) מספר הטבלאות הווירטואליות עשוי להגיע אף לעשר.

תחביר ומאפיינים

הפקודות בשפה מחולקות למספר תחומים עיקריים:

- אחזור נתונים
 - SELECT - אחזור נתונים מתוך טבלה / כמה טבלאות.
- מניפולציית נתונים
 - INSERT - הוספת שורות לטבלה.
 - UPDATE - עדכון נתונים קיימים.
 - DELETE - מחיקת שורות נתונים.
 - MERGE - מיזוג נתונים של מספר טבלאות.
- הגדרת נתונים
 - CREATE - פריט מבנה חדש (למשל טבלה).
 - ALTER - שינוי תכונותיו של פריט קיים.
 - DROP - מחיקת פריט קיים.
- בקרת נתונים
 - GRANT - הענקת גישה.
 - REVOKE - מניעת גישה.

*יש לציין כי הפקודות (clauses) הנ"ל אינן כלל הפקודות הקיימות בשפה, אלא רק הבולטות והשימושיות ביותר שבהן.

בשפה קיימים 22 טיפוסים נתונים שונים, כאשר הבולטים ביותר הם CHARACTER (תו מחרוזת), BOOLEAN (בוליאני), INTEGER (מספר שלם), FLOAT (מספר ממשי), NULL (מידע חסר או לא ידוע), TEXT (מחרוזת), ARRAY (אוסף אלמנטים), DATE (תאריך) ו-TIME (שעה).

כמו כן, קיימים גם עשרה אופרטורים: = (שווה ל-), != (שווה מ-), < / > (גדול מ- / קטן מ-), <= / >= (גדול שווה מ- / קטן שווה מ-), BETWEEN (בין טווח מסוים), LIKE (מתאים לתבנית תווים), IN (שווה לערך אחד מתוך מס' ערכים אפשריים) ו-IS NOT / IS (השוואה ל-NULL).

דוגמאות

בחירת כל המידע מטבלה מסוימת :

```
table FROM * SELECT;
```

יצירת טבלה חדשה :

```
CREATE TABLE books (  
    id INTEGER PRIMARY KEY AUTO_INCREMENT,  
    name TEXT NOT NULL,  
    author TEXT,  
    year INTEGER  
);
```

הוספת שורה חדשה לטבלה :

```
INSERT INTO books (name, author, year)  
VALUES ('Ulysses', 'James Joyce', 1922);
```

הוספת עמודה חדשה לטבלה :

```
ALTER TABLE books  
ADD COLUMN price INTEGER;
```

מחיקת שורות בטבלה :

```
DELETE FROM books  
WHERE author IS NULL;
```

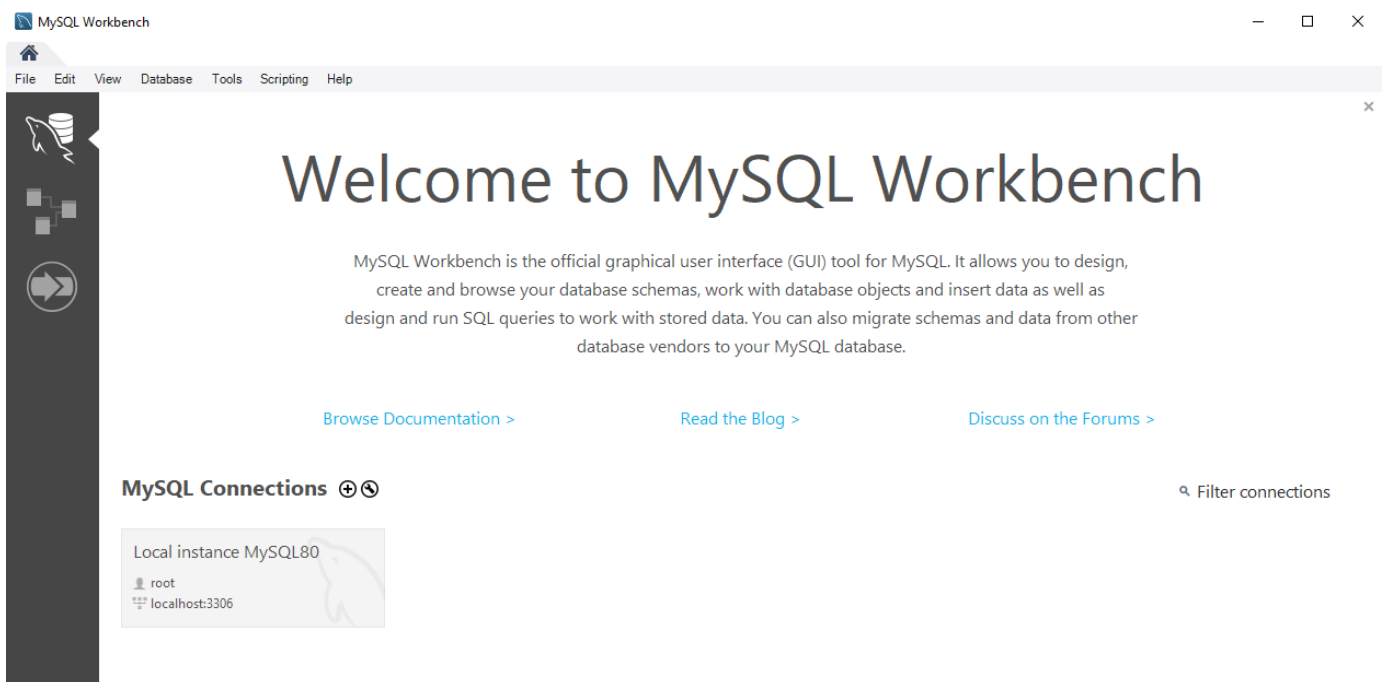
בחירת מידע באופן ספציפי יותר וסידורו :

```
SELECT name, year FROM books  
WHERE price > 50  
ORDER BY year;
```

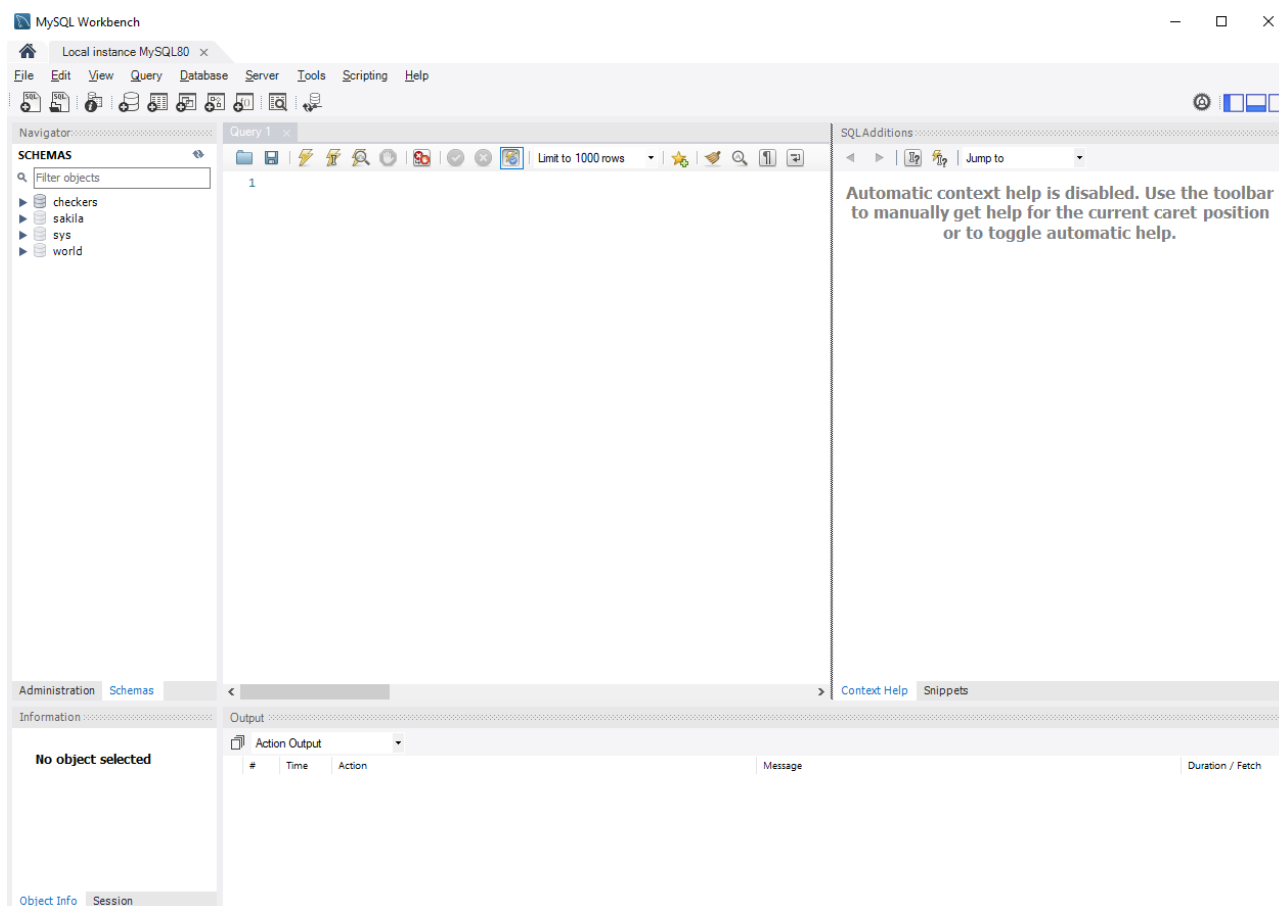

מבוא לשימוש במסד נתונים MySQL

MySQL הינו מסד נתונים יחסי (טבלאי) ורוב משתמשים המבוסס על שפת SQL. התוכנה פותחה במקור ע"י החברה השוודית MySQL AB, וכיום היא בבעלות ענקית התוכנה האמריקאית אורקל (Oracle). בנוסף, המסד זמין בגרסה חינמית (MySQL Community Server) ופתוחה (open-source), או בגרסה מקצועית לתאגידים (Enterprise Server). MySQL נפוץ בתוכנות ובאתרי אינטרנט רבים הדורשים מסד נתונים, דוגמת ויקיפדיה, טוויטר, יוטיוב ופייסבוק. הוא נחשב כמסד קל וידידותי ללימוד ולשימוש באופן יחסי למסדי נתונים אחרים. המסד כתוב בשפות C++ ו-C, וזמין על שלל מערכות הפעלה, כגון Windows, Mac, Linux ועוד.

דרך טובה להשתמש ב-MYSQL, היא להתקין את התוכנה MySQL Workbench על המחשב האישי. MySQL Workbench הינה תוכנה ויזואלית לעיצוב וניהול מסדי נתונים יחסיים מסוג SQL, אשר משלבת פיתוח בשפת SQL, אדמינסטרציה (ניהול), עיצוב, יצירה ותחזוק של אותם מסדי נתונים. תוכנה זו היא אחת מהנפוצות והשימושיות ביותר בתחום התוכנה, ומגיעה למיליוני הורדות מדי שנה. לאחר ההורדה, הרשמה והתקנה מלאה, יש לפתוח את התוכנה, שנראית כך:



ניתן לראות שישנו חיבור ברירת מחדל, אשר משתמש לצורך תקשורת עם שרת ה-MYSQL, ובאופן ספציפי יותר - עם מסד הנתונים הנוכחי של המשתמש. להתחלת עבודה, יש ללחוץ על החיבור המבוקש, וכדי להתחבר יש להקיש את הסיסמה שהוגדרה מראש ע"י המשתמש. עתה, מסך התוכנה נראה כך:



מצד שמאל למעלה נמצא סרגל הניווט. בסרגל הניווט ניתן לנווט בין מסדי הנתונים והטבלאות השונות, וכן לנווט בין פעולות האדמיניסטרציה, התחזוקה והביצועים. במרכז, ממוקם עורך השאילתות, אשר בו ניתן לכתוב, לערוך ולהריץ שאילתות. הפלט לשאילתות הללו נמצא מתחת לעורך בכותרת "Output". בצד ימין, נמצא מסך התוספות וכלי העזר ל-SQL. לבסוף, בחלק העליון ביותר של המסך, ממוקם סרגל הכלים של התוכנה, ובו שלל פעולות ואפשרויות.

ליצירת מסד נתונים חדש, יש ליצור שאילתה חדשה בעורך השאילתות, ולכתוב בה את הפקודה:
 "CREATE DATABASE mydb;". לאחר הרצה של השאילתה ע"י לחיצה על כפתור הברק, נוצר מסד נתונים חדש בשם mydb.

על מנת לקשר את mydb לקוד (לשם הדוגמא - בשפת פייתון, בעזרת הספרייה mysql.connector, ראו פרק בנושא), יש לכתוב את השורות הבאות:

```
mydb = mysql.connector.connect(
    host='localhost',
    user='root',
    passwd='{סיסמה}',
    database='mydb'
)
```

כעת mydb הינו אובייקט מטיפוס MySQLConnection, ודרכו ניתן לתקשר עם מסד הנתונים.

על מנת לבצע פעולות הקשורות במסד, כגון עדכון, מחיקה ושליפה, יש ליצור אובייקט חדש (דרך mydb) מטיפוס MySQLCursor בשם mycursor:

```
mycursor = mydb.cursor()
```

דרך שלל המתודות של אובייקט זה, דוגמת fetch, execute ועוד, מבצעים את אותן פעולות הקשורות במסד. לאחר כל עדכון, יש לקרוא למתודה mydb.commit(), על מנת שהשינויים אכן ישמרו.

פיתוח התוכנה (החלק המחקרי)

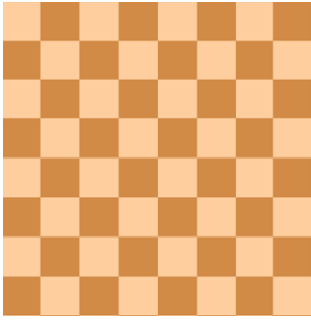
לוח המשחק

מטרה : פיתוח ויזואלי של לוח משחק בן 64 משבצות, אשר יסודר בשני צבעים לסירוגין.

לצורך הפיתוח, ניצור מחלקה Board אשר תייצג את הלוח.

נוסיף למחלקה פעולה בונה ריקה (ככל שפיתוח המשחק יתקדם, כך גם המחלקה Board תשתכלל, אך כעת היא פשוטה יחסית), ובנוסף מתודה בשם draw_squares אשר תיצור ותציג בחלונית ה-pygame (שמימדיה אותחלו בצורה ריבועית) את הלוח, כיאה למטרה (איור א').

בתוך draw_squares, נתחיל בשימוש בפונקציה fill (של pygame), אשר תצבע את כל מסך החלונית בצבע חאקי. לאחר מכן, נרוץ בלולאה מקוננת לרוחב המסך (כלומר כל פעם "שורה"), וכל מספר קבוע של פיקסלים ניצור, בעזרת הפונקציה draw.rect (של pygame), ריבוע בצבע חום בגודל קבוע. כך למעשה, מקבל הלוח את צורתו.



איור א' – לוח המשחק

חיילי המשחק

כלי החייל

מטרה: פיתוח ויזואלי של 24 חיילי משחק, כאשר מחציתם בצבע אחד בתחתית הלוח, והשאר בצבע שונה בחלקו העליון של הלוח.

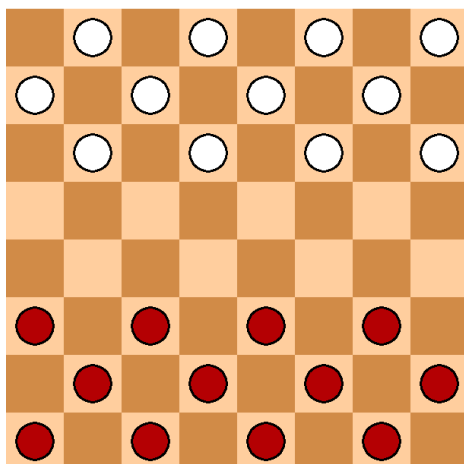
לצורך הפיתוח, ניצור מחלקה Piece אשר תייצג כל אחת מ-24 חיילי המשחק (כל חייל כמופע מחלקה נפרד).

למחלקה נוסף שני משתנים קבועים, PADDING ו-OUTLINE (לצרכים ויזואליים טכניים), ופעולה בונה בעלת חמש תכונות; מס' שורה, מס' עמודה, צבע, ערך x של החייל במסך וערך y תואם. ערכי ה-x ו-y מאותחלים לפי השורה והעמודה וכן לפי רוחב הריבוע בלוח (מספר קבוע). בנוסף, נכתוב מתודה בשם draw, אשר תפקידה יהיה להציג על המסך את החייל, בעזרת הפונקציה draw.circle (של pygame). המתודה תציג את החייל בשורה ובעמודה המתאימים לו (לפי התכונות) ובצבע השייך לו.

במחלקה Board, נוסף תכונה board - רשימה דו-ממדית בגודל x88 שתכיל בתוכה את כל משבצות הלוח (בין אם הן ריקות או מכילות אובייקטי חיילים), ובנוסף לכך גם מתודה create_pieces, אשר תיצור מופעי חייל חדשים ותציב אותם על גבי הלוח במקומות המתאימים.

במסגרת המתודה, נעבור בלולאה מקוננת על כל המשבצות, ובמידה והן עונות לקריטריון ההשמה ההתחלתי (שלוש השורות העליונות / תחתונות ביותר), ניצור מופע חייל חדש עם מס' שורה, עמודה וצבע מתאימים, ונוסיף אותו לרשימה הדו-ממדית board. משבצות ריקות נאתחל ב-board כ-0.

לבסוף (עדיין במחלקה Board), ניצור מתודה draw אשר "תצייר" (תציג בצורה ויזואלית בחלונית) הן את הלוח והן את כל החיילים. היא תתחיל מלקרוא למתודה draw_squares, ולאחר מכן תעבור בלולאה מקוננת על התכונה board - במידה והתא הנוכחי לא מכיל 0, נקרא למתודה draw של מופע החייל המאוחסן בתא זה. כך, כל החיילים יהיו מוצגים במקומות המתאימים, ונקבל חלונית כמו באיור ב'. (בפעולה הראשית; board.draw(WINDOW))



איור ב' – לוח המשחק במלואו

מסעים אפשריים

מטרה : חישוב והצגת המהלכים האפשריים לחייל מסוים בכל רגע נתון.

לצורך הפיתוח, ניצור במחלקה Board שתי מתודות רקורסיביות, `search_left` ו-`search_right`, שכל אחת מהן תבדוק את המהלכים האפשריים בהתאם לכיוון האלכסון (אלכסון ימינה / שמאלה). שתיהן מקבלות שישה פרמטרים ; שורת התחלה, שורת סיום, כיוון ("במורד" או "במעלה" הלוח), צבע, עמודת התחלה ורשימה בשם `skipped_pieces` שתכיל בתוכה אובייקטי חיילים עליהם עברנו במהלך הסריקה. כל אחת מהמתודות הללו מחזירה מילון בשם `moves`, שמכיל את כל המהלכים האפשריים. במילון זה, המפתחות הם טאפלים (tuple) הכוללים את מס' השורה והעמודה (או במילים אחרות - משבצת הלוח), והערכים הם רשימות הכוללות את הערך/ים הנמצא/ים במשבצת/ות עבור אותו מהלך - 0 במידה ואין שם חייל, ואובייקט חייל במידה ויש. השימוש בלשון רבים הוא מפאת האפשרות לאכילה ואף כמה אכילות במהלך בודד. המתודות קוראות לעצמן בצורה רקורסיבית ומוסיפות מהלכים חדשים למילון עד להגעה לקצה הלוח.

לבסוף, ניצור מתודה `get_valid_moves` שתקבל אובייקט מטיפוס חייל כפרמטר, ותחזיר מילון מהלכים אפשריים עבור אותו החייל. המתודה כמובן נעזרת ב-`search_left` ו-`search_right`. כעת נותר להציג את המהלכים האפשריים לחייל מסוים בעת לחיצה עליו. לצורך כך, ולצרכי פיתוח עתידיים נוספים, ניצור מחלקה `Game` שתייצג את המשחק. היא תכלול שתי תכונות - חלונית ה-`pygame` (תכונה הכרחית עבור כל פעולה ב-`pygame`) והלוח, כאמור מטיפוס `Board`. בתוך `Game`, נוסיף מתודה בשם `draw_valid_moves`, שתקבל את המילון `moves` כפרמטר. במידה והמשחק לא נגמר (תכונה במחלקה `Board`), נעבור על כל המהלכים שבמילון, ונשתמש שוב בפונקציה `draw.circle` (של `pygame`) על מנת לצייר עיגול כחול קטן במרכז משבצת המהלכים.

פעולות ויכולות החייל

מטרה : אפשרות תזוזה ואכילה לכלל חיילי המשחק.

נתמקד כעת במחלקה Piece, כשהמטרה שלנו היא להעביר אובייקט החייל כלשהו ממקום אחד בלוח למקום אחר, כלומר לשנות את מיקומו בהתאם ברשימה הדו-ממדית board (במחלקה Board). ראשית ניצור מתודה בשם calculate_pos, שתעדכן את תכונות ערכי ה-x וה-y של החייל, ע"פ מיקום המשבצת שבו הוא נמצא. בנוסף, ניצור מתודה בשם move, שתקבל כפרמטרים מס' שורה ועמודה חדשים. המתודה תעדכן את הערכים הנוכחיים של החייל לאלה שהתקבלו, ולאחר מכן תקרא למתודה calculate_pos.

בחזרה למחלקה Board, ניצור מתודה move_piece שתקבל כפרמטרים אובייקט מטיפוס חייל (שברצוננו להזיז) ומס' שורה ועמודה (למעשה מיקומי המשבצת החדשה). המתודה תעביר את אובייקט החייל למיקום המתאים ברשימה הדו-ממדית board, וכך בפעם הבאה שהמתודה draw מופעלת (דבר הקורה בכל אלפיות השנייה בעזרת pygame.display.update), החייל יוצג במשבצת החדשה. בחירת החייל עצמה באמצעות לחיצת עכבר, מתבצעת על ידי מתודה נוספת בשם select_piece, שעוקבת אחר לחיצות המשתמש ומחזירה אמת אם הוא אכן לחץ על חייל בצבע התואם לתור הנוכחי.

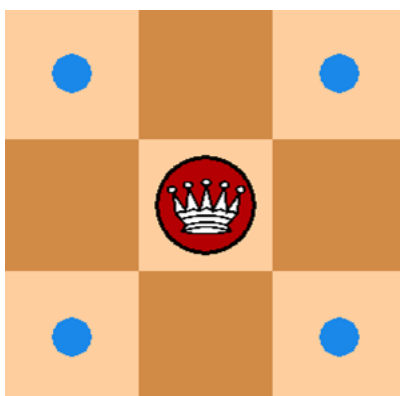
בפועל, תזוזות החיילים מתבצעות מהמחלקה ה"מרכזית", Game. שם, ניצור מתודה make_move, שתקבל כפרמטרים מס' שורה ועמודה אליהם החייל יזוז. המתודה מבצעת את השינוי, בעזרת שימוש בכל המתודות שהגדרנו קודם לכן.

כלי המלכה

מטרה : הפיכת חייל רגיל לכלי מלכה, בעת שהוא מגיע לסוף הלוח ביחס למיקומו ההתחלתי. באפשרות המלכה לנוע לכל כיוון, והיא תתאפיין על ידי סימון כתר מיוחד.

לשם כך, נוסיף תכונה בוליאנית בשם queen שתאותחל ל-False במחלקה Piece. במתודה move_piece (במחלקה Board), נבצע בדיקה לראות האם המיקום החדש משמעותו מלכה, ובמידה וכן נעדכן את ערך התכונה של אובייקט החייל. ערך תכונה זו יבוא לידי ביטוי במתודות המהלכים האפשריים - בה, במידה והחייל אכן מלכה, מתודות החיפוש search_left ו-search_right יפעלו פעמיים כל אחת, במקום פעם אחת בלבד. זאת, כאמור, מכיוון שביכולתה של המלכה לנוע לכל כיוון, ולא רק קדימה (ביחס למיקומה ההתחלתי). על כן, מהלכים בכל אחד מארבעת האלכסונים נבדקים ונאספים למילון moves.

לבסוף, במתודה draw שב-Piece, בעת ציור כל החיילים, נבדוק האם החייל הנוכחי הוא מלכה, ובמידה וכן נוסיף לו כתר מלכה. באיור ג' ניתן לראות מלכה בצבע אדום, ומסביבה את המהלכים האפשריים עברה, מסומנים בעיגולים כחולים כחולים קטנים.



איור ג' – מלכה, לאחר לחיצה עליה

מימוש יכולת המשחק לשני שחקנים

מטרה: פיתוח פונקציונליות למשחק דמקה (לוקאלי) עבור שני שחקנים.

כאמור, כבר פיתחנו את יכולות התזוזה והאכילה של החיילים, וכן של המלכות. כעת, במחלקה `Game`, ניעזר בתכונה `turn`, ששווה למעשה אדום או לבן. את התכונה נאתחל לאדום באופן שולי, כלומר השחקן האדום יבצע את המהלך הראשון. כעת, לאחר כל מהלך, נקרא למתודה `switch_turns`, שכשמה כן היא תחליף את התור. כזכור, לתכונה `turn` יש חשיבות רבה בעת ביצוע המהלך, שכן היא קובעת אילו חיילים (מאיזה צבע) יהיה ניתן להזיז.

לבסוף, ניצור במחלקה `Board` מתודה בשם `check_win` אשר תבדוק האם הסתיים המשחק (ע"פ חוקי משחק הדמקה). במידה ואכן הושג ניצחון של אחד הצדדים, התכונה `is_game_over` תהיה אמת, ובכל חוסמת את המשך ביצוע המהלכים, כלומר למעשה נועלת את הלוח. בנוסף, תוצג הודעה סמלית למשתמש, אשר מציעה לו לצאת מהמשחק או להתחיל משחק חדש. במידה ויבחר להתחיל משחק חדש, אובייקט המשחק (`Game`) יקרא לאחת מהמתודות שבו בשם `reset_game`, שתיצור לוח חדש ותאפס את כל שאר התכונות המשתנות.

במחלקה הראשית `main`, ניצור אובייקט חדש מטיפוס `Game` שינהל את המשחק, ובלולאת `while` אינסופית, הכוללת את שלל פרוצדורות `pygame` (טכני למדי), חלונית המשחק עצמה רצה בקצב רענון של 60 פריימים לשנייה. בתוך הלולאה, מעבר לייצוג הויזואלי, מתבצעת גם התקשורת עם המשתמש, באמצעות כל "אירועי" (`events`) המקלדת והעכבר, אשר נגישים דרך `pygame`.

פיתוח שחקן AI

מטרה: פיתוח שחקן ממוחשב (AI) שיוכל לשחק נגד המשתמש, במצב של שחקן יחיד (singleplayer).

ראשית, בתפריט הראשי של המשחק, נוסיף שני כפתורים - מצב 2 שחקנים או מצב שחקן יחיד, על מנת שהמשתמש יוכל לבחור במצב המשחק הרצוי. בנוסף, במחלקה הראשית ניצור משתנה בוליאני `with_ai` אשר מאותחל לשקר, שישתנה לאמת במידה והמשתמש בחר את מצב השחקן היחיד. כך בעת הרצת המשחק יהיה הבדל בין שני המצבים. כמו כן, נחליט שהשחקן האדום הוא המשתמש והשחקן הלבן הוא המחשב.

ובכן, כפי שראינו באלגוריתם המינימקס (minimax), טובת המהלכים נקבעת בראש ובראשונה ע"פ הערכה (למעשה ציון) מספרית שניתנת לכל מצב של הלוח. כאמור החלטנו כי השחקן הלבן הוא ה-AI, ונחליט שגם יהיה השחקן הממקסם, כלומר הוא שואף להערכת לוח של פלוס אינסוף ($+\infty$). על כן, ניצור במחלקה Board מתודה `evaluate`, שתחזיר את אותה הערכה מספרית, שתחושב באופן הבא:

```
self.white_left - self.red_left + (self.white_queens * 1.5 - self.red_queens * 1.5)
```

כלומר, ההערכה מורכבת מחיבור שני הפרשים; הראשון - ההפרש בין מספר החיילים הלבנים למספר החיילים האדומים, והשני, שהינו ההפרש בין מספר המלכות הלבנות כפול 1.5, למספר המלכות האדומות כפול 1.5.

חישוב זה נעשה משני שיקולים עיקריים. ראשית, השחקן הלבן הוא הממקסם, כלומר הערכה חיובית ככל האפשר של הלוח טובה לו, ועל כן מספר הכלים (חיילים / מלכות) הלבנים הוא המחוסר ולא המחסר. השיקול השני הינו שערך המלכות במשחק גדול מערך החיילים, ועל כן יש לתת יותר משקל למלכות. השיקול הנ"ל בא לידי ביטוי בעצם ההכפלה ב-1.5 את מספרי המלכות.

כעת, ניצור מחלקה `SearchTree` (איור ד'), שתכיל ותרכז בתוכה את כל הקשור במציאת המהלך האופטימלי בעזרת אלגוריתם המינימקס וגיזום אלפא-ביתא. הפעולה הבונה כוללת שתי תכונות: `depth`, כלומר עומק עץ חיפוש המהלכים, ו-`player_color`, כלומר צבע השחקן הנוכחי.

לאחר מכן, ניצור מתודה `find_optimal_move`, שמקבלת אובייקט מטיפוס לוח כפרמטר. היא קוראת למתודה המרכזית, `minimax_alpha_beta`, אשר מקבלת כפרמטר עומק, לוח, האם ממקסם, אלפא וביתא. מתודה זו, אשר נכתבה בדיוק ע"פ הפסאודו-קוד, מחזירה הערכה נוכחית ללוח ואת הלוח עצמו. הפלט הסופי הוא כאמור ההערכה הטובה ביותר לשחקן ואת הלוח שנגרם כתוצאה מהמהלך. בפועל, כל המהלכים הללו הם למעשה **לוחות** שנוצרו כתוצאה מאותו מהלך - כאשר המחשב "מבצע מהלך", הוא לא לוחץ / בוחר חייל כלשהו ו"מזיז" אותו, אלא מחליט לבחור **לוח** בעל ההערכה המספרית התואמת ביותר עבורו.

נשים לב כי ההערכה המספרית של הלוח אינה רלוונטית למשתמש, אלא רק הלוח עצמו. עם זאת, ההערכה משמשת לשמירה והשוואת כל מהלך שנבדק, ובלעדיה החיפוש אינו יכול להתבצע.

מסביב ל-minimax_alpha_beta, ישנן עוד שתי מתודות עיקריות אשר דואגות לביצוע מהלכי המחשב. הראשונה, get_all_moves, מקבלת כפרמטר אובייקט מטיפוס Board, ומחזירה רשימה הכוללת את כל המהלכים האפשריים עבור אותו צבע שחקן (לפי התכונה player_color) בהתאם ללוח הנוכחי. את סימולציית המהלכים הללו, אשר יוצרת אינספור לוחות חדשים, get_all_moves עושה בעזרת המתודה השנייה, make_temp_move. פונקציה זו מקבלת לוח (למעשה עותק של הלוח המקורי, ולא אותו לוח עצמו), חייל, מהלך וחיילים ש"נקפצו מעל" (כלומר החייל אכל אותם, במידה ויש כמובן), ומחזירה לוח חדש לאחר השינויים.

כך, פורשת alpha_beta את שלל ענפיה בחיפוש אחר המהלך האופטימלי.

```
class SearchTree:

    def __init__(self, depth, player_color):...

    def find_optimal_move(self, board):...

    def _minimax_alpha_beta(self, current_depth, board, is_max_turn, alpha, beta):...

def get_all_moves(board, color):...

def make_temp_move(piece, move, board, skipped):...
```

איור ד' – מבט על המחלקה SearchTree והפונקציות הנוספות

(מימוש האלגוריתם בפייתון מבוסס בדיוק על הפסאודו-קוד).

הפונקציה האחרונה המשחקת תפקיד בשחקן ה-AI הינה ai_move, אשר נמצאת במחלקה Game, והיא מקבלת את אותו לוח שנבחר כתוצאה מהמהלך האופטימלי ש-alpha_beta מצאה. היא מעדכנת את הלוח הנוכחי (כאמור תכונה במחלקה Game) ללוח שהתקבל ומעבירה את התור למשתמש.

הדבר האחרון שנותר הוא בחירת דרגת הקושי.
לאחר שהמשתמש לחץ בתפריט הראשי על מצב "שחקן יחיד", נפתח לו עוד מסך בעל שלושה כפתורים, אחד לכל דרגת קושי (קל, בינוני, קשה). הפונקציה שעושה זאת נקראת `select_difficulty`, והיא למעשה מחזירה את **עומק** האלגוריתם (בהתאם ללחיצת המשתמש) - ככל שהעומק רב יותר, כך מחושבים מהלכים רבים יותר וניתן למצוא מהלך אופטימלי בצורה מעמיקה יותר - כלומר בפועל דרגת הקושי קשה יותר.
במשחק, מצאתי לנכון (לאחר ניסיונות ובדיקות) להעניק לדרגת קושי "קל" עומק של 2, לדרגה "בינוני" עומק של 3 ולדרגה האחרונה, "קשה", עומק של 4.

בתוכנית הראשית, אשר מנהלת את המשחק, בכל פעם שמגיע תורו של המחשב, ניצור אובייקט חדש מטיפוס `SearchTree`, ונעביר לו כפרמטר את העומק וצבע לבן, ונקרא לפונקציה `ai_move` עם הלוח האופטימלי החדש שחושב בעקבות קריאה ל-`find_optimal_move` על לוח המשחק הנוכחי.

פיתוח מסך בית עם תפריט ראשי

מטרה: פיתוח מסך בית ("Main Menu") הכולל תפריט ראשי* ובו מספר אפשרויות, כגון התחלת משחק במצב שחקן יחיד, מצב 2 שחקנים, כניסה למערכת, סטטיסטיקות ועוד.

*עד כה נעשה שימוש במונח "תפריט ראשי". כעת נעשה סדר בדברים ונציג את פיתוחו בצורה מסודרת.

ראשית, כאשר מריצים את קובץ הפיתוח, קוראים למעשה לפונקציה הראשית main. ובכן, main מייצגת את התפריט הראשי של המשחק. בעזרת הפונקציה window.blit (של pygame), נגדיר את רקע המסך לתמונה מעוצבת בהתאמה אישית למשחק, אשר כוללת מיקומים לכפתורים השונים. ב-pygame, אין למעשה "כפתור", אלא יש אירועי עכבר / מקלדת (pygame events) - לדוגמה לחיצה שמאלית על העכבר. לאחר לחיצה כזו, שמיד מזוהה ע"י pygame, ניתן לקבל את קואורדינטות סמן העכבר בעת הלחיצה. כך, ניתן לבדוק אם הסמן היה על אחד הכפתורים (לפי קואורדינטות הכפתור שגם ידועות), ובהתאם לכך לקרוא / להפעיל את הפונקציות הרצויות.

כיוצא בזאת, נוסיף בתפריט מספר כפתורים:

- מצב שחקן יחיד - Single Player: בעת לחיצה עליו, מתבצעת קריאה לפונקציה אשר מנהלת את המשחק עצמו, start_game, עם המשתנה with_ai = True. כמו כן, מתבצעת קודם קריאה לפונקציה select_difficulty, אשר כאמור נותנת למשתמש לבחור את רמת הקושי.
- מצב שני שחקנים - Two Players: בעת לחיצה עליו, מתבצעת קריאה ל-start_game, ללא כל תוספות (ברירת המחדל של המשתנה with_ai היא False).
- כפתור כניסה למערכת - אייקון בנוסח Login: בעת לחיצה נותן למשתמש אפשרות להיכנס למערכת עם שם משתמש וסיסמא (Login) או להירשם למערכת (Sign Up).
- כפתור פרופיל - אייקון בדמות אדם: בעת לחיצה מוביל למסך פרופיל אישי, אשר כולל בתוכו מספר פרטים בסיסיים על המשתמש.
- כפתור סטטיסטיקות - אייקון גרף עמודות: בעת לחיצה מוביל למסך הסטטיסטיקות, במידה והתבצעה כניסה של משתמש רשום.
- כפתור עזרה - אייקון סימן שאלה: בעת לחיצה מוביל למסך עזרה, הכולל הדרכה קצרה על שימוש בתוכנה ובנוסף את חוקי משחק הדמקה.
- כפתור Leaderboard - אייקון פודיום: בעת לחיצה מוביל למסך ה-Leaderboard, אשר מציג טבלה של עשרת המשתמשים בעלי מספר הניצחונות הרב ביותר.

- כפתור פאזלים - אייקון פאזל: בעת לחיצה מוביל למסך הפאזלים, אשר בו יכול המשתמש לבצע אותם. פאזלים של דמקה הם למעשה מצבים מתוך משחקים מן העבר, בהם יש מהלך אשר מוביל לניצחון. על המשתמש לשחק את המהלך (ואף סדרת המהלכים) אשר מוביל/ים (בסופו של דבר) לניצחון. ישנם שישה פאזלים סך הכל המסודרים בדרגת קושי עולה, וברגע שהמשתמש מצליח את הפאזל, נפתחת לו האפשרות לבצע את הפאזל הבא.

פיתוח מסך הרשמה וכניסה למערכת

מטרה: פיתוח מסך ובו ניתן להיכנס / להירשם למערכת המשחק, באמצעות שם משתמש וסיסמא. על הנתונים להישמר, ולאפשר כניסה ויציאה מחודשת למשתמשים בכל עת.

ראשית, ניצור מסד נתונים חדש בשם checkers בתוכנה MySQL Workbench. בתוך המסד ניצור טבלה חדשה בשם users, שתכיל את כל המידע הנחוץ על המשתמשים:

- עמודה ID - מפתח ראשי (מטיפוס מספר שלם), ייחודי לכל משתמש. auto increment מופעל.
- עמודה username - שם משתמש (מחרוזת).
- עמודה password - סיסמה (מטיפוס מחרוזת).

בקוד, נוסיף לכל הפונקציות (main_menu, start_game וכו') פרמטר user_id=None. המטרה היא לדעת האם יש משתמש מחובר (ואם כן, איזה), וכתוצאה מכך להתאים את התוכנה למשתמש הנוכחי אשר מחובר.

בעזרת pygame, ניצור מסך חדש בשם login_page עבור הכניסה למערכת (מעוצב בהתאם). נוסיף שני שדות להזנת טקסט, אחד עבור שם המשתמש והשני עבור הסיסמה. לאחר שהמשתמש לוחץ על הכפתור "Login" או על מקש ה-Enter, נבדוק האם הרשומות קיימות בטבלה users שבמסד הנתונים.

את הקישור של מסד הנתונים לקוד נבצע בעזרת הספרייה mysql.connector. בעזרת הפקודה mysql.connector.connect נוכל להתחבר למסד הנתונים, ולבצע עליו / ממנו פעולות.

לאחר שהקישור למסד הנתונים התבצע כראוי, נבדוק את הנתונים שהוזנו במסך ה-Login (איור א'):

```
cursor.execute("SELECT * FROM users;")
data = cursor.fetchall()
for user in data:
    if user[1] == entered_username and user[2] == entered_password:
        return user[0]
return -1
```

איור א' – בדיקת נתוני ההתחברות

אנו למעשה מבצעים שאילתה אשר מחזירה את כל הנתונים מטבלת המשתמשים, ובודקת האם הנתונים שהוזנו תואמים את אחת מהרשומות בטבלה. במידה ולא, מוצגת הודעת שגיאה. במידה כן, הפונקציה מחזירה את ה-ID של המשתמש, אחרת היא מחזירה -1. כעת, אותה ID (במידה והוא לא שווה -1) מועבר בכל קריאה לכל אחת מהפונקציות שמנהלות את התוכנה.

עתה יש לטפל במשתמשים חדשים, אשר אין להם חשבון קיים וברצונם להירשם למערכת. על כן, ניצור מסך חדש בשם `signup_page` עבור הרשמה למערכת (מעוצב בהתאם). גם שם, ניצור שני שדות להזנת טקסט, עבור שם המשתמש והסיסמה. לאחר שהמשתמש לוחץ על כפתור "Sign Up" או על מקש ה-Enter, נבדוק קודם האם שם המשתמש הוא באורך של לפחות שלושה תווים והסיסמה באורך של לפחות שישה. במידה והתנאי מושג, נמשיך בבדיקה האם שם המשתמש כבר קיים במערכת. אם אחד מהתנאים הללו מופרים, תוצג למשתמש הודעת שגיאה מתאימה. כאשר כל התנאים הושגו, נוסיף את המידע שהוכנס כרשומה חדשה בטבלה `users` (איור ב'):

```
clause = "INSERT INTO users (username, password) VALUES (%s, %s);"
user_info = (entered_username, entered_password)
cursor.execute(clause, user_info)
db.commit()
```

איור ב' – הוספת המידע לטבלה

* (אין צורך בהשמת ערך בשדה ה-ID, כאמור - מדובר בשדה אוטומטי שגדל ב-1 בכל הזנת רשומה חדשה). כעת, פרטי המשתמש החדש מופיעים במסד הנתונים, ובאפשרותו לבצע כניסה למערכת.

פיתוח מסך סטטיסטיקות

מטרה: פיתוח מסך אשר מציג את סטטיסטיקות המשחקים עבור כל המשתמש (לפי המשתמש שמחובר כעת). על המסך להציג סטטיסטיקות כגון מס' ניצחונות, הפסדים, מספר מהלכים ממוצע וכיו"ב.

ראשית, עלינו ליצור טבלה חדשה אשר בה נוכל לאחסן את הנתונים על המשחקים. על כן, ניצור טבלה חדשה בשם games שתכיל את כל המידע על כלל המשחקים שמשוחקים:

- עמודה difficulty - רמת הקושי שמולה בחר השחקן לשחק* (מטיפוס מספר שלם).
- עמודה gtime - זמן המשחק מתחילתו ועד סופו (מטיפוס TIME בפורמט hh:mm:ss).
- עמודה total_moves - מספר המהלכים הכולל שהתבצעו במשחק (מספר שלם).
- עמודה userID - מפתח זר (foreign key) לעמודה ID בטבלה users. תפקידה לשמור את ה-ID של המשתמש אשר שיחק את המשחק (מספר שלם).
- עמודה won - האם המשתמש ניצח או הפסיד. מטיפוס TINYINT, כלומר 0 (הפסד) או 1 (ניצחון).

*סטטיסטיקות נצברות רק במשחקי מצב שחקן יחיד.

בקוד, ניצור מחלקה חדשה בשם GameStats שתרכז את כל צבירת הסטטיסטיקות על משחק. להלן ממשק המחלקה:

| מתודה | תיאור |
|--|---|
| <code>def __init__(self, ai_diff, user_id):</code> | פעולה בונה. בנוסף, מאתחלת את התכונות won, start_time ו- total_moves ל-0, ואת total_time למחרוזת ריקה. זמן קריאה: בעת הפעלה של משחק חדש. |
| <code>def start_timer(self):</code> | מעדכנת את התכונה start_time לזמן הנוכחי (בעזרת הספרייה time). זמן קריאה: לאחר ביצוע המהלך הראשון במשחק. |
| <code>def end_timer(self):</code> | מעדכנת את התכונה total_time למחרוזת בפורמט "hh:mm:ss", אשר מייצגת את הזמן הנוכחי בעת הקריאה פחות הזמן בתכונה start_time (או במילים אחרות - הזמן הכולל של המשחק). זמן קריאה: מיד לאחר סיום המשחק. |

| | |
|--|---|
| <pre>def assign_winner(self, winner):</pre> | <p>אם winner שווה "שחקן אדום" (כלומר השחקן, ולא המחשב), המתודה מעדכנת את התכונה won ל-1.</p> <p>זמן קריאה: מיד לאחר סיום המשחק.</p> |
| <pre>def insert_stats_to_database(self):</pre> | <p>מכניסה את כלל נתוני המשחק (למעשה תכונות המחלקה) לטבלה games שבמסד הנתונים.</p> <p>זמן קריאה: בעמיד לאחר סיום המשחק.</p> |

במחלקה Game, במידה והמשחק הוא מצב שחקן יחיד, נוסף תכונה בשם stats שהיא למעשה אובייקט מטיפוס GameStats. כך, כל הסטטיסטיקות מנוהלות בצורה אוטומטית בתוך המחלקה Game מתחילת המשחק (יצירת אובייקט סטטיסטיקות חדש) ועד סופו (הכנסת נתוני המשחק למסד הנתונים).

כעת, ניצור מסך חדש בשם stats_page, עבור הצגת הסטטיסטיקות של המשתמש המחובר כעת (כניסה למסך הסטטיסטיקות מחייבת חיבור של משתמש פעיל, אחרת מוצגת הודעה המציינת זאת). שליפת כל הנתונים מתבצעת בעזרת פונקציית עזר בשם get_all_stats, אשר שולפת את כל המשחקים מהטבלה games בהם השדה userID תואם את ה-ID של המשתמש המחובר כעת (איור א'):

```
cursor.execute("SELECT * FROM games WHERE userID = " + str(user_id))
data = cursor.fetchall()
```

איור א' – שליפת כל המשחקים בעבור משתמש ספציפי

הפונקציה מסדרת את המידע בצורה נוחה ומתאימה, והאחרון הוא שמתקבל ב-stats_page. לאחר מכן, בעזרת pygame נציג את כל הסטטיסטיקות:

- אחוז ניצחון (Win %) - מספר הניצחונות חלקי מספר המשחקים הכולל.
- מספר ניצחונות והפסדים נגד כל רמת קושי.
- מספר ניצחונות והפסדים כוללים וכן מספר המשחקים הכולל.
- המשחק הקצר ביותר והארוך ביותר (מבחינת זמן).
- מספר מהלכים ממוצע למשחק.

בנוסף, מוצג מעין גרף עוגה, שחלקו (לפי יחס הניצחונות וההפסדים) צבוע בירוק והשאר באדום. (במובנים מתמטיים - הגרף, מהזווית 0 ועד הזווית "מספר הניצחונות חלקי מספר המשחקים הכולל, כפול 360" צבוע בירוק, והשאר (מהזווית הנ"ל ועד 360) צבוע באדום).

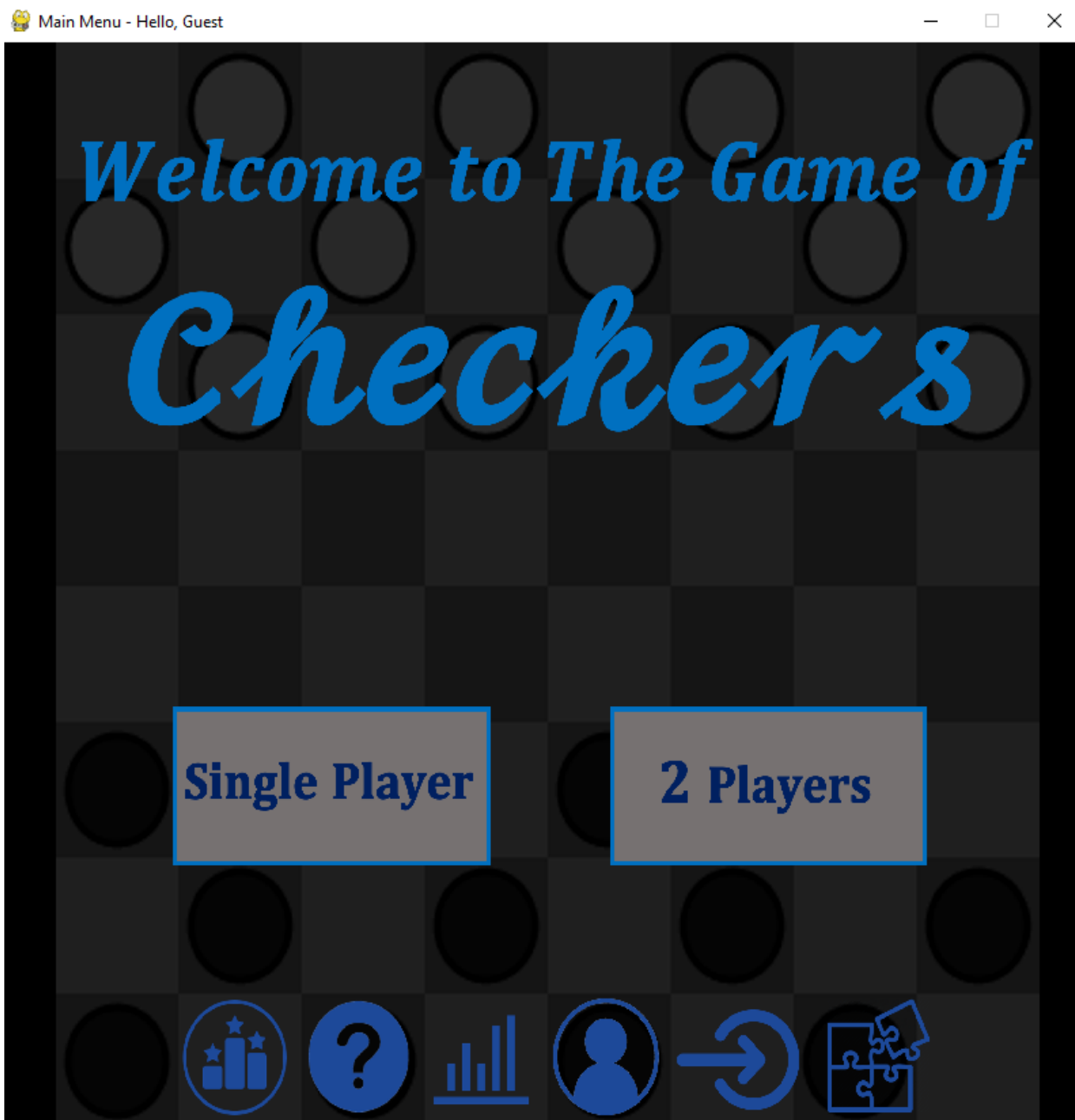
הצגת הפרויקט הסופי

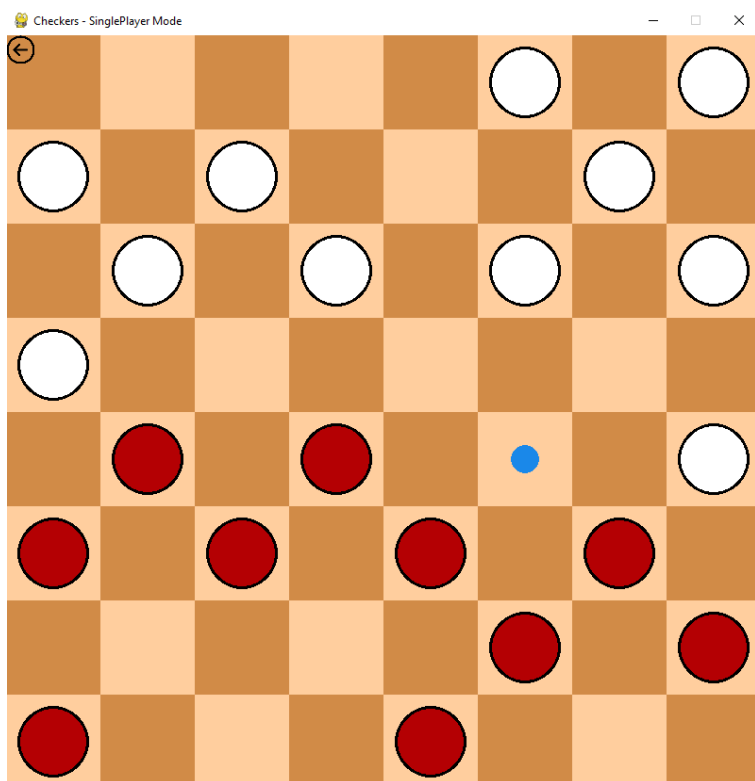
צילומי מסך נבחרים המציגים את התוכנה.

מסך הבית של התוכנה.

ניתן לראות את הכותרת מצד שמאל למעלה ("Main Menu"), ואת העובדה כי מדובר באורח ("Guest"), מכיוון שטרם בוצעה כניסה / הרשמה למערכת.

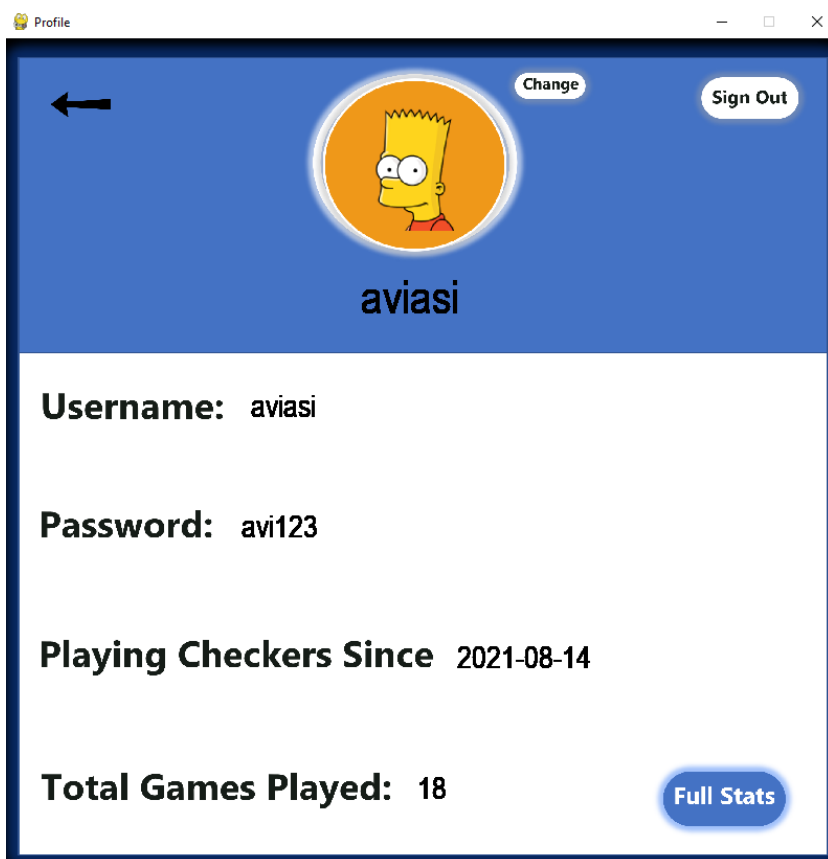
בנוסף, ניתן לראות את שלל כפתורי הניווט – במרכז: מצב שחקן יחיד ומצב שני שחקנים, מטה (משמאל): טבלת הישגי המשתמשים (Leaderboard), מסך עזרה, סטטיסטיקות, פרופיל, כניסה למערכת ופאזלים.





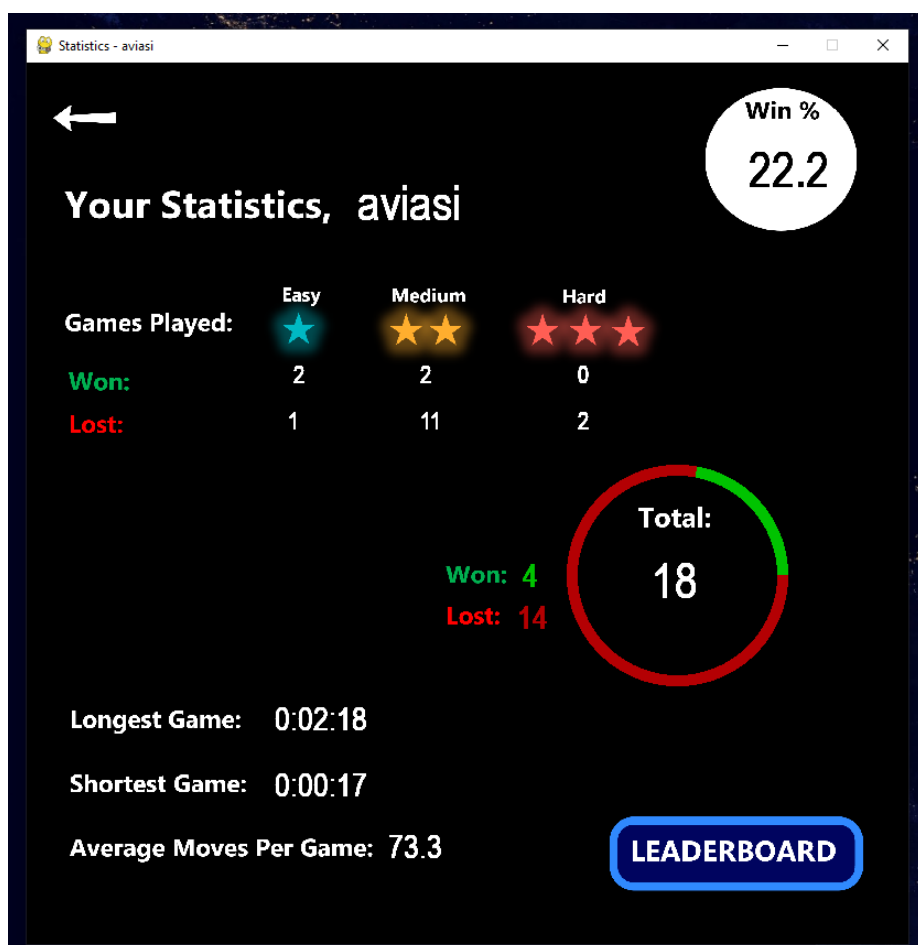
בצילום משמאל ניתן לראות קטע מתוך משחק דמקה במצב שחקן יחיד.

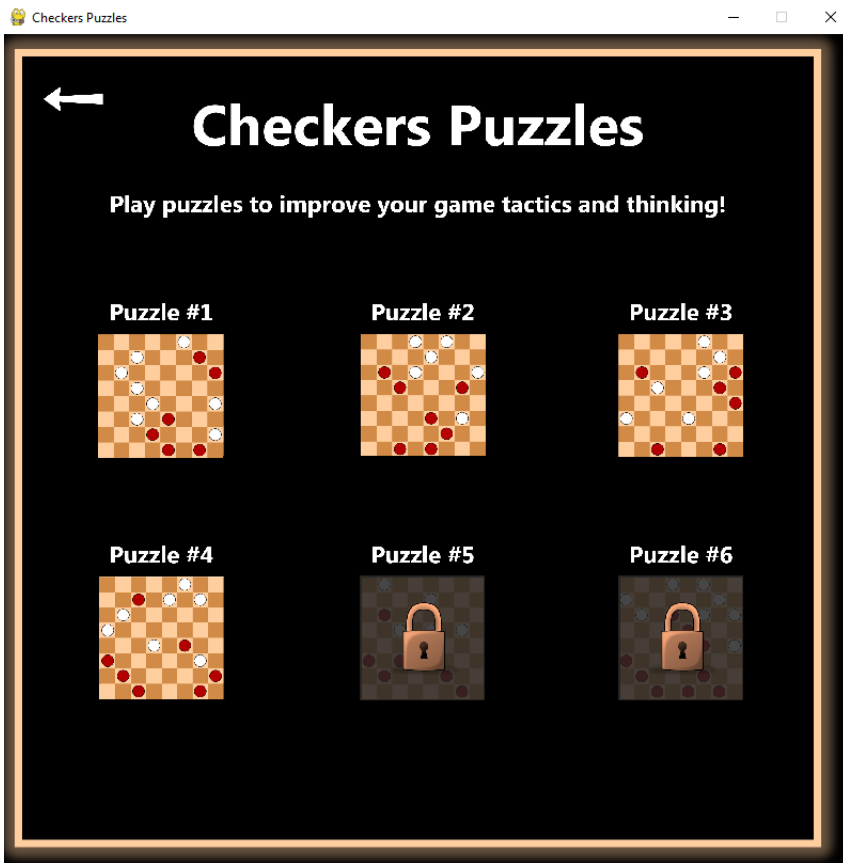
בצילום מימין ניתן לראות ביצוע של כניסה למערכת (Login), ע"י הזנת שם משתמש וסיסמה. כעת המשתמש נמצא בעת הקשת הסיסמה, ולכן שדה הסיסמה מוקף במלבן סגול.



בצילום משמאל ניתן לראות את מסך הפרופיל האישי, לאחר ביצוע ההתחברות למערכת. במסך הנ"ל מוצגים שם המשתמש, הסיסמה, תאריך ההרשמה למערכת, מספר המשחקים הכולל ותמונת הפרופיל ניתנת לשינוי.

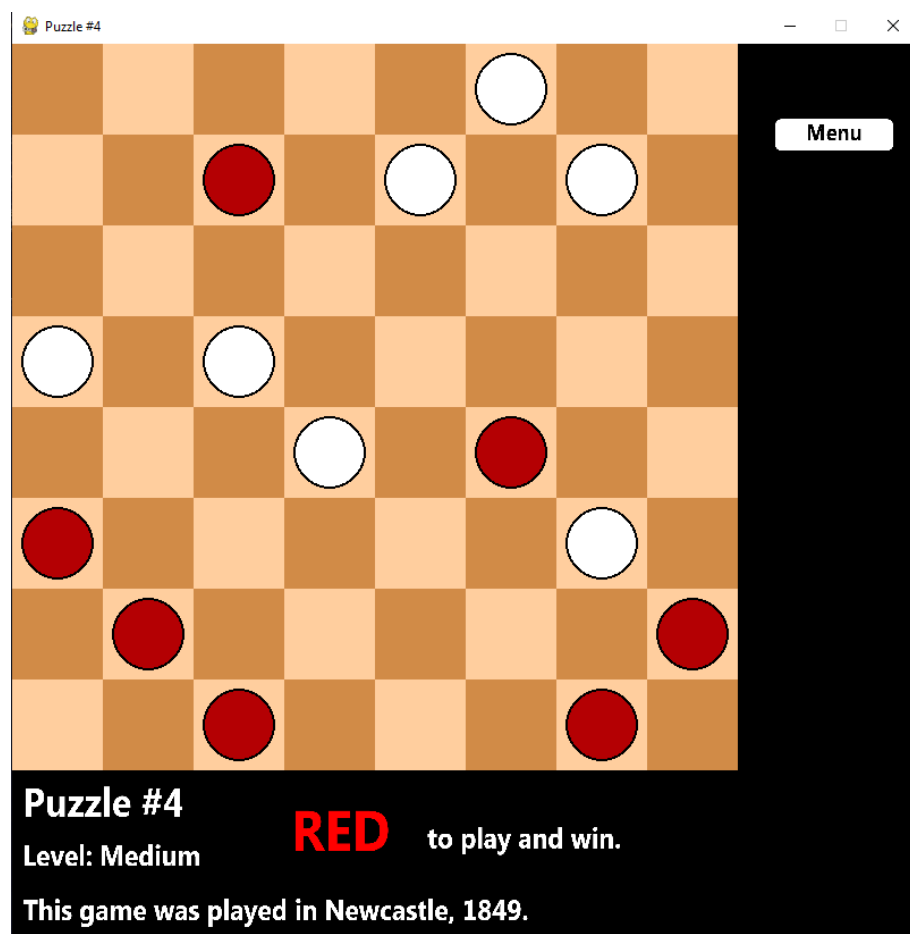
בצילום מימין ניתן לראות את מסך הסטטיסטיקות המלאות עבור המשתמש המחובר כעת. מוצג אחוז הניצחון, סך המשחקים, הפסדים וניצחונות, גם בעבור כל רמת קושי בנפרד, המשחק הארוך והקצר ביותר ומספר המהלכים הממוצע.





בצילום משמאל ניתן לראות את מסך
הפאזלים, עבור המשתמש המחובר כעת;
כל משתמש נמצא בשלב אחר בפתרון
הפאזלים – מתחילים מהפאזל הראשון,
ולאחר השלמתו נפתח הפאזל השני וכך
הלאה.

בצילום מימין ניתן לראות את מסך
פאזל #4, בעת שהמשתמש מנסה
לפתור אותו.



סיכום

ראשית, ניכר כי מבחינה פונקציונלית כלל המערכת פועלת ומתנהגת כראוי, ללא שגיאות או בעיות : המשחק עובד חלק הן במצב שני שחקנים והן במצב שחקן יחיד, ניתן להיכנס ו/או להירשם למערכת, לאחר משחקים ניתן לראות את הסטטיסטיקות וכמו כן שאר הדברים פועלים כנדרש.

בראייה ספציפית יותר, למשל על פיתוח השחקן הממוחשב (AI), הצלחתי לפתח ולממש אותו בשלוש דרגות קושי שונות (קל, בינוני, קשה) בעזרת אלגוריתם המינימקס (minimax), בשילוב אלגוריתם האופטימיזציה אלפא-ביתא (Alpha-Beta Pruning), לצורך שיפור משמעותי של סיבוכיות הזמן. למידת אופן פעולת האלגוריתם והשימוש בו היו מרתקים עבורי. בין השאר, מפאת השוני מלימודי הנדסת תוכנה, שם אנו עוסקים בלמידה עמוקה (Deep Learning), כלומר בטכניקות שונות לחלוטין בכדי "לאמן" את המחשב (לדוגמא, לאפשר לו לשחק מספר רב של פעמים בצורה אוטומטית, ובעקבות כך "ללמוד ולהשתפר" בעצמו).

דוגמא נוספת היא מימוש מסד הנתונים וקישורו באופן ישיר ואוטומטי לתוכנה. בעזרת התוכנה MySQL Workbench, אשר מאפשרת שימוש במסדי נתונים מסוג SQL, והספרייה mysql.connector (בפייתון), יכולתי לבצע את השליפות, העדכונים ושאר הפעולות על מסד הנתונים בהקשר לתוכנה : הרשמה ו/או כניסה למערכת (גם לאחר סגירה של התוכנה), שמירת הסטטיסטיקות עבור כל שחקן בצורה ייחודית וכן שליפתן לאחר מכן, שמירת המידע האישי על כל משתמש (למשל תמונת הפרופיל שבחר) וכיו"ב. שימוש במסד נתונים לראשונה עורר אצלי השתאות ופליאה רבה. עד כה, הייתי רגיל שבתוכניות קוד כל המידע שנצבר במהלך ההרצה נמחק מיד בסיומה. אולם הפעם, למדתי כיצד משתמשים בכלי כה עצמתי, המאפשר שמירה של מידע באופן תמידי, גם לאחר סיום ההרצה. לדוגמא, כאשר כותבים ומריצים שאילתת עדכון כלשהי (למשל הוספת טבלה), ניתן למחוק לחלוטין את שורות הקוד הנ"ל לאחר ההרצה – הרי הפקודה כבר בוצעה ונשמרה במסד.

בראייה לאחר, ללא ספק נהניתי מאוד לאורך כל התהליך. חקרתי ולמדתי רבות (הן ידע תיאורטי והן ידע מעשי), פיתחתי תוכנה במשך שעות אינסופיות, כתבתי עבודת גמר אקדמית – והכל, מתוך בחירה, בנושאים המרתקים אותי וקרובים לליבי.

ביבליוגרפיה

- בן-הרוש, י' (2020). "פייתון מונחה עצמים 1: מחלקות, אובייקטים, תכונות ומתודות". מתוך אתר "רשתטק".
<https://reshetech.co.il/python-tutorials/object-oriented-python-classes-objects-methods-and-variables>
- בן-הרוש, י' (2020). "פייתון מונחה עצמים 4: פולימורפיזם ושמות של מתודות". מתוך אתר "רשתטק".
<https://reshetech.co.il/python-tutorials/polymorphism-and-method-naming>
- "גיזום אלפא-ביתא" (2022), ויקיפדיה.
https://he.wikipedia.org/wiki/%D7%92%D7%99%D7%96%D7%95%D7%9D_%D7%90%D7%9C%D7%A4%D7%90-%D7%91%D7%99%D7%AA%D7%90
- "דמקה" (2022), ויקיפדיה.
<https://he.wikipedia.org/wiki/%D7%93%D7%9E%D7%A7%D7%94>
- "מחלקה (תכנות)" (2022), ויקיפדיה.
https://he.wikipedia.org/wiki/%D7%9E%D7%97%D7%9C%D7%A7%D7%94_%D7%AA%D7%9B%D7%A0%D7%95%D7%AA
- "עץ מינימקס" (2022), ויקיפדיה.
https://he.wikipedia.org/wiki/%D7%A2%D7%A5_%D7%9E%D7%99%D7%A0%D7%99%D7%9E%D7%A7%D7%A1
- "תכנות הצהרתי" (2022), ויקיפדיה.
https://he.wikipedia.org/wiki/%D7%AA%D7%9B%D7%A0%D7%95%D7%AA_%D7%94%D7%A6%D7%94%D7%A8%D7%AA%D7%99
- "תכנות מונחה-עצמים" (2022), ויקיפדיה.
https://he.wikipedia.org/wiki/%D7%AA%D7%9B%D7%A0%D7%95%D7%AA_%D7%9E%D7%95%D7%A0%D7%97%D7%94-%D7%A2%D7%A6%D7%9E%D7%99%D7%9D
- "MySQL" (2022), ויקיפדיה.
<https://he.wikipedia.org/wiki/MySQL>

- *SQL* (2022), ויקיפדיה.
<https://he.wikipedia.org/wiki/SQL>
- “*Alpha-Beta Pruning*”, Wikipedia.
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning
- *Checkers Cruncher*
<https://www.checkercruncher.com/problems>
- Mandziuk, J, Kusiak, M, and Waledzik, K. (2007). “*Evolutionary-Based Heuristic Generators for Checkers and Give-Away Checkers*”. [Online Version].
https://www.mini.pw.edu.pl/~mandziuk/PRACE/es_init.pdf
- Jewell, C. (2019). “*Artificial Intelligence: the new electricity*”. WIPO Online Magazine.
https://www.wipo.int/wipo_magazine/en/2019/03/article_0001.html
- “*Minimax*”, Wikipedia.
<https://en.wikipedia.org/wiki/Minimax>
- “*Minimax Algorithm in Game Theory*”, GeeksforGeeks Website.
<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>
- “*MySQL*”, Wikipedia.
<https://en.wikipedia.org/wiki/MySQL>
- “*MySQL Workbench*”, Wikipedia.
https://en.wikipedia.org/wiki/MySQL_Workbench
- *Official MySQL Documentation*
<https://dev.mysql.com/>
- *Official Pygame Documentation*, Pygame Website.
<https://www.pygame.org/docs/>

- Payne, T. “*Let’s Learn Python #21 - Min Max Algorithm*”. “Trevor Payne” YouTube Channel.
<https://www.youtube.com/watch?v=fInYh90YMJU>
- “*Polymorphism in Python*”, Programiz Website.
<https://www.programiz.com/python-programming/polymorphism>
- “*Python Inheritance*”, w3schools.com Website.
https://www.w3schools.com/python/python_inheritance.asp
- “*Pygame*”, Wikipedia.
<https://en.wikipedia.org/wiki/Pygame>
- Ruscica, T. “*Pygame Tutorial*”. “Tech with Tim” YouTube Channel. A series of YouTube videos.
<https://www.youtube.com/watch?v=i6xMBig-pP4&t=370s>
- Sen, G. “*What is the Minimax Algorithm? - Artificial Intelligence*”. “Guarav Sen” YouTube Channel.
<https://www.youtube.com/watch?v=KU9Ch59-4vw>
- *Stack Overflow*, Official Website.
<https://stackoverflow.com/>
- Winston, P. “*6. Search: Games, Minimax, and Alpha-Beta*” lecture. “MIT OpenCourseWare” YouTube Channel.
[https://www.youtube.com/watch?v=STjW3eH0Cik&ab_channel=MITOpenCourseW
are](https://www.youtube.com/watch?v=STjW3eH0Cik&ab_channel=MITOpenCourseWare)

נספחים

מחלקות נבחרות מן הקוד

*חשוב לציין כי כל הקבצים נמצאים וזמינים לצפייה במרשתת בכתובת:

<https://github.com/MrFred17/AvodatGmar>

המחלקה אשר מנהלת את המשחק, Game :

```
class Game:

    def __init__(self, window, ai_diff=None,
user_id=None, correct_board=None):
        self.window = window
        self.new_board()
        self.started = False
        self.stats = None
        if user_id:
            self.stats = GameStats(ai_diff, user_id)

    def update_game(self, arrows=None):
        self.board.draw(self.window, arrows)
        self.draw_valid_moves(self.valid_moves)
        pygame.display.update()

    def select_piece(self, row, col):
        if not self.board.is_game_over and
self.started_puzzle and not
self.board.is_draw_offer_visible and not
self.board.is_draw:
            if self.selected:
                result = self.make_move(row, col)
                if result is False:
                    self.selected = None
                    self.select_piece(row, col)

            if row > 7 or col > 7:
                return False
            piece = self.board.get_piece_at(row, col)

            if piece != 0 and piece.color == self.turn:
                self.selected = piece
                self.valid_moves =
```

```

self.board.get_valid_moves(piece)
    return True
    return False

def switch_turns(self):
    self.valid_moves = {}
    if self.turn == WHITE:
        self.turn = RED
    else:
        self.turn = WHITE
    if self.stats:
        self.stats.total_moves += 1

def make_move(self, row, col):
    if not self.board.is_game_over:
        selected_piece =
self.board.get_piece_at(row, col)
        if (row, col) in self.valid_moves and
self.selected and selected_piece == 0:
            self.board.move_piece(self.selected,
row, col)

            if self.stats:
                if not self.started:
                    self.stats.start_timer()
                    self.started = True

            skipped_piece = self.valid_moves[(row,
col)]

            if skipped_piece:
self.board.remove_piece(skipped_piece)
                self.switch_turns()

            return True

        return False

def new_board(self):
    self.board = Board()
    self.turn = RED
    self.selected = None
    self.valid_moves = {}

def draw_valid_moves(self, moves):
    if not self.board.is_win_msg_visible and not

```

```

self.board.is_game_over:
    for move in moves:
        row, col = move
        pygame.draw.circle(self.window, BLUE,
                            (col * SQUARE_SIZE +
                             SQUARE_MIDDLE, row * SQUARE_SIZE + SQUARE_MIDDLE), 15)

    def reset_game(self, ai_depth, user_id):
        self.new_board()
        if user_id:
            self.stats = GameStats(ai_depth, user_id)
            self.started = False

    def ai_move(self, board):
        self.board = board
        self.switch_turns()

```

המחלקה אשר מצייגת את הלוח (Board):

```

class Board:

    def __init__(self):
        self.board = []
        self.red_left, self.white_left = 12, 12
        self.red_queens, self.white_queens = 0, 0
        self.create_pieces()
        self.is_game_over = False
        self.is_draw = False

    def create_pieces(self):
        for row in range(ROWS):
            self.board.append([])
            for col in range(COLS):
                if col % 2 == ((row + 1) % 2):
                    if row < 3:
                        self.board[row].append(Piece(row, col,
WHITE))
                    elif row > 4:
                        self.board[row].append(Piece(row, col,
RED))
                    else:
                        self.board[row].append(0)
                else:
                    self.board[row].append(0)

    def get_piece_at(self, row, col):
        return self.board[row][col]

```

```

    def move_piece(self, piece, row, col):
        self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col], self.board[piece.row][piece.col]
        piece.move(row, col)

        if row == ROWS - 1 or row == 0:
            if not piece.queen:
                piece.make_queen()
                if piece.color == RED:
                    self.red_queens += 1
                else:
                    self.white_queens += 1

    def remove_piece(self, pieces):
        for piece in pieces:
            self.board[piece.row][piece.col] = 0
            if type(piece) != int:
                if piece.color == WHITE:
                    self.white_left -= 1
                else:
                    self.red_left -= 1

    def draw_squares(self, window):
        window.fill(HAKI)
        for row in range(ROWS):
            for col in range(row % 2, COLS, 2):
                pygame.draw.rect(window, BROWN, (row * SQUARE_SIZE, col * SQUARE_SIZE, SQUARE_SIZE, SQUARE_SIZE))

    def get_valid_moves(self, piece):
        moves = {}
        left = piece.col - 1
        right = piece.col + 1
        row = piece.row

        if piece.color == RED or piece.queen:
            moves.update(self.search_left(row - 1, max(row - 3, -1), -1, piece.color, left))
            moves.update(self.search_right(row - 1, max(row - 3, -1), -1, piece.color, right))
        if piece.color == WHITE or piece.queen:
            moves.update(self.search_left(row + 1, min(row + 3, ROWS), 1, piece.color, left))
            moves.update(self.search_right(row + 1, min(row + 3, ROWS), 1, piece.color, right))
        return moves

    def search_left(self, self, start, stop, step, color, left,
skipped_pieces=[]):
        moves = {}

```

```

        last_seen = []
        for r in range(start, stop, step):
            if left < 0:
                break

            current_square = self.board[r][left]
            if current_square == 0:
                if skipped_pieces and not last_seen:
                    break
                elif skipped_pieces:
                    moves[(r, left)] = last_seen +
skipped_pieces
            else:
                moves[(r, left)] = last_seen

            if last_seen:
                if step == -1:
                    row = max(r - 3, -1)
                else:
                    row = min(r + 3, ROWS)

                moves.update(self.search_left(r + step,
row, step, color, left - 1, skipped_pieces=last_seen))
                moves.update(self.search_right(r + step,
row, step, color, left + 1, skipped_pieces=last_seen))
                break

            elif current_square.color == color:
                break
            else:
                last_seen = [current_square]

        left -= 1

    return moves

    def search_right(self, start, stop, step, color, right,
skipped_pieces=[]):
        moves = {}
        last_seen = []
        for r in range(start, stop, step):
            if right >= COLS:
                break

            current_square = self.board[r][right]

            if current_square == 0:
                if skipped_pieces and not last_seen:
                    break
                elif skipped_pieces:
                    moves[(r, right)] = last_seen +

```

```

skipped_pieces
    else:
        moves[(r, right)] = last_seen

    if last_seen:
        if step == -1:
            row = max(r - 3, -1)
        else:
            row = min(r + 3, ROWS)

        moves.update(self.search_left(r + step,
row, step, color, right - 1, skipped_pieces=last_seen))
        moves.update(self.search_right(r + step,
row, step, color, right + 1, skipped_pieces=last_seen))
        break

    elif current_square.color == color:
        break
    else:
        last_seen = [current_square]

    right += 1

    return moves

def get_all_pieces(self, color):
    pieces = []
    for row in self.board:
        for piece in row:
            if piece != 0 and piece.color == color:
                pieces.append(piece)
    return pieces

def evaluate(self):
    return self.white_left - self.red_left +
(self.white_queens * 1.5 - self.red_queens * 1.5)

def draw(self, window, arrows_info=None):
    self.draw_squares(window)
    for row in range(ROWS):
        for col in range(COLS):
            piece = self.board[row][col]
            if piece != 0:
                piece.draw(window)

    window.blit(quit_button, (0, 0))
    if self.is_error_msg_visible:
        self.show_quit_msg()

    if self.is_win_msg_visible:
        window.blit(game_over_img, (0, 218))

```



```

        if self.is_draw_offer_visible:
            self.show_draw_offer()

        if not self.is_win_msg_visible and (self.check_win()
or self.is_draw):
            window.blit(back_show_img, (5, 225))

    def check_win(self):
        if self.white_left == 0:
            return RED
        if self.red_left == 0:
            return WHITE

        white_has_moves = False
        for white_piece in self.get_all_pieces(WHITE):
            moves = self.get_valid_moves(white_piece)
            if len(moves) > 0:
                white_has_moves = True
                break

        red_has_moves = False
        for red_piece in self.get_all_pieces(RED):
            moves = self.get_valid_moves(red_piece)
            if len(moves) > 0:
                red_has_moves = True
                break

        if not white_has_moves:
            return RED
        if not red_has_moves:
            return WHITE

        if self.red_queens == self.white_queens == 1 and
self.red_left == self.white_left == 0:
            return WHITE

```

המחלקה אשר מציגת כלי (Piece):

```

class Piece:
    PADDING = 15
    OUTLINE = 3

    def __init__(self, row, col, color):
        self.row = row
        self.col = col
        self.color = color
        self.queen = False

```

```

        self.x = 0
        self.y = 0
        self.calculate_pos()

    def __repr__(self):
        return str(self.color)

    def calculate_pos(self):
        self.x = SQUARE_SIZE * self.col + SQUARE_SIZE // 2
        self.y = SQUARE_SIZE * self.row + SQUARE_SIZE // 2

    def move(self, row, col):
        self.row = row
        self.col = col
        self.calculate_pos()

    def make_queen(self):
        self.queen = True

    def draw(self, window):
        radius = SQUARE_MIDDLE - self.PADDING
        pygame.draw.circle(window, BLACK, (self.x, self.y),
radius + self.OUTLINE)
        pygame.draw.circle(window, self.color, (self.x,
self.y), radius)
        if self.queen:
            if self.color == WHITE:
                window.blit(BLACK_QUEEN, (self.x - QUEEN_WIDTH
// 2, self.y - QUEEN_HEIGHT // 2))
            else:
                window.blit(WHITE_QUEEN, (self.x - QUEEN_WIDTH
// 2, self.y - QUEEN_HEIGHT // 2))

    def compare_piece(self, other):
        a = self.row == other.row and self.col == other.col
        b = self.color == other.color and self.queen is
other.queen
        return a and b

```

דוגמא למסך ב-pygame, בדוגמא הנ"ל – מסך ההרשמה למערכת (Signup):

```

def signup_page():
    pygame.display.set_caption("Sign Up")
    clock = pygame.time.Clock()
    screen = pygame.display.set_mode((800, 800))
    screen.fill((10, 110, 225))
    screen.blit(signup_page_img, (12, 12))

```

```

username_typing = password_typing = False
username_input = password_input = ""
username_color = password_color = WHITE

invalid_msg = ""

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if is_mouse_on_button(516, 743, 71, 15) or
is_mouse_on_button(61, 83, 53, 20):
                login_page()
            if is_mouse_on_button(546, 61, 160, 170):
                main_menu()
            if is_mouse_on_button(555, 539, 182, 72):
                if len(username_input) < 3:
                    invalid_msg = "Username length must be
3 or longer."
                elif len(password_input) < 6:
                    invalid_msg = "Password length must be
6 or longer."
                else:
                    user_info = (username_input,
password_input)
                    if check_if_exists(user_info, True) !=
-1:
                        invalid_msg = "username already
exists"
                    else:
                        clause = "INSERT INTO users
(username, password, since) VALUES (%s, %s, %s);"
                        today =
str(datetime.date.today()).replace('-', ':')
                        user_info = (user_info[0],
user_info[1], today)
                        SQL.mycursor.execute(clause,
user_info)
                        SQL.mydb.commit()
                        login_page()

            if is_mouse_on_button(59, 359, 583, 14):
                username_typing = True
                password_typing = False
                username_color = PURPLE
                password_color = WHITE

            elif is_mouse_on_button(62, 444, 583, 14):
                username_typing = False

```

```

        password_typing = True
        username_color = WHITE
        password_color = PURPLE

    else:
        username_typing = False
        password_typing = False
        username_color = password_color = WHITE

    if event.type == pygame.KEYDOWN:

        if username_typing:
            if event.key == pygame.K_BACKSPACE:
                username_input = username_input[:-1]
            elif len(username_input) < 20 and
event.key != pygame.K_RETURN:
                username_input += event.unicode

        elif password_typing:
            if event.key == pygame.K_BACKSPACE:
                password_input = password_input[:-1]
            elif len(username_input) < 20 and
event.key != pygame.K_RETURN:
                password_input += event.unicode

        if event.key == pygame.K_RETURN:
            if len(username_input) < 3:
                invalid_msg = "Username length must be
3 or longer."

            elif len(password_input) < 6:
                invalid_msg = "Password length must be
6 or longer."

            else:
                user_info = (username_input,
password_input)

                if check_if_exists(user_info, True) !=
-1:
                    invalid_msg = "username already
exists"

                else:
                    clause = "INSERT INTO users
(username, password, since) VALUES (%s, %s, %s);"
                    today =
str(datetime.date.today()).replace('-', ':')
                    user_info = (user_info[0],
user_info[1], today)

                    SQL.mycursor.execute(clause,
user_info)

                    SQL.mydb.commit()
                    print("inserted to db")
                    login_page()

```

```

pygame.draw.rect(screen, WHITE, (175, 322, 460, 60))
username_surface = font.render(username_input, True,
BLACK)
screen.blit(username_surface, (193, 345))
pygame.draw.rect(screen, username_color, (178, 343,
351, 40), 3)

pygame.draw.rect(screen, WHITE, (175, 402, 460, 60))
password_surface =
font.render("*" * len(password_input), True, BLACK)
screen.blit(password_surface, (193, 430))
pygame.draw.rect(screen, password_color, (176, 422,
351, 40), 3)

pygame.draw.rect(screen, WHITE, (130, 480, 370, 60))
invalid_msg_surface = small_font.render(invalid_msg,
True, RED)
screen.blit(invalid_msg_surface, (140, 490))

pygame.display.update()
clock.tick(60)

```