

תיכון עירוני ד' ע"ש פרופ' אהרון קציר

עבודת גמר מדעית בהיקף 5 יחליל

סמל שאלון 899589

עבודת גמר במדעי המחשב:

**פיתוח תוכנה מלאה סביב משחק הדמיה עם בינה מלאכותית, מערכת
משתמשים, צבירת סטטיסטיות 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.	סקירה הספרות.....	5.
5.....	משחק הדמeka.....	3.1
7.....	מבוא לתוכנות מונחה עצמים בשפת פיתון.....	3.2
12.....	פיתוח משחקים וממשק משתמש ב-Pygame.....	3.3
14.....	אלגוריתם ה-minimax.....	3.4
22.....	מבוא לשפת SQL.....	3.5
25.....	מבוא לשימוש במסד הנתונים MySQL.....	3.6
28.....	פיתוח התוכנה.....	4.
28.....	לוח המשחק.....	4.1
29.....	חילוי המשחק.....	4.2
33.....	IMPLEMENTATION.....	4.3
34.....	פיתוח שחקן AI.....	4.4
37.....	פיתוח מסך בית עם תפריט ראשי.....	4.5
39.....	פיתוח מסך הרשמה וכניסה למערכת.....	4.6
41.....	פיתוח מסך סטטיסטיות.....	4.7
43.....	הצגת הפרויקט הסופי.....	5.
47.....	סיכום.....	6.
48.....	ביבליוגרפיה.....	7.
51.....	נספחים.....	8.
51.....	מחולות נבחרות מן הקוד.....	8.1

הקדמה אישית ותודות

נחשפט לתוכנות הראשונה לפני כשלוש שנים. מיד התחברתי לרעיונות, למופשיות, למגוון הכלים והאפשרויות ולעוצמה הטמונה ברכישת שפת תוכנות. מאז אני עוסק בתוכנות (פיתוח תוכנה ובנית אתרים בעיקר) כתהביב רציני ומשמעותי.

במשך השנים הללו, מצאתי כי הדרך הטובה ביותר ללמידה ולהתנסות בתחום היא, איך לא, לפתח דברים בכוחות עצמי. כאשר ישנה מטרה וחוזן ברור לתוצר, קל להישאב לתוכה התהילה, או במילים אחרות להשקיע ימים רבים של מחשבה, מחקר, למידה ופיתוח.

תחומי פיתוח המשחקים הוא בהחלט דרך למש ולשפר את יכולות המחקר והפיתוח, וכן בחרתני לביצוע עבודות גמר במקצוע מדעי המחשב (5 ייחדות לימוד). למעשה, הייתה לי הזכות לעסוק בתחום עניין מושבע שלי באופן רציני, מחויב ואקדמי, ולקבל הכרה בכך מטעם המנהה, בית הספר ומשרד החינוך. אני סבור כי עבודות גמר הם דרך יוצאת דופן לשפר ולהעшир את החוויה בתיכון – העובדה כי תלמיד יכול לבחר נושא ספציפי הקרוב אליו, ולבצע עליו מחקר מעמיק, בתוך מסגרת סדורה ותומכת, היא לא פחות מאשר מיוחדת במיוחד.

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

קריאה מהנה,

נדב

תודות

ברצוני להודות **לעידו גודיס**, על הנחיה אקדמית, מחקרית ומעשית מקיפה, תמיכה מלאה לאורך כל הדרך וזמינותו תמידית. עידו, אתה הוא דמות המורה (והמנה) האידיאלי עבורי. מסירותך, שיקידתך וגישתך למקצוע מדעי המחשב, והוראה כלל, ראיות לשบท, ומשמעות את חווית הלימודים בבייה"ס עד לכדי אין שיעור. מי יתן וכמה שיותר תלמידים יוכל להנות מכך. על כן, שמחתי מאוד להמשיך איתך גם בשנה"ל זו.

עוד אודה **לليمור שיואן**, רכזת עבודות הגמר בבייה"ס, על ליווי מעמיק ומסור וניהול מקצועי. לימור יקרה, בזכותו מספר כותבי עבודות הגמר בעירוני ד' הולך וגדל, באין מפתיע. הפכת את ענף עבודות הגמר לנגיש, פתוח ומרתק וחשוב להכיר תודה בכך, וכן באחדתך הגלולה בתחום והעבודה הרבה שהיא מבצעת מאחוריו הקלאסים.

מבוא

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

ובכן, נשאלת השאלה, כיצד הכל נעשה? כיצד האנושות הגיעו לפיתוחים וטכנולוגיות שכאה? שאלה מרתקת זו מורכבת למעשה מגוון 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 טמוניים במידע ובבינה מלאכותית? אילו כלים טכנולוגיים עומדים לרשותנו בעת הזו?

בעבודתי בחרטוי להתייחס חלק לא מבוטל מהשאלות הנ"ל. כבר כמה זמן שהו במוחי, ובמסגרת העבודה ניסיתי למצוא להן תשובה ולהגיע למסקנות. על כן, בחרטוי לפתח תוכנת מחשב בנושא משחק הדמeka, בה יהיה ניתן לשחק בשני שחנים או נגד המחשב, ובנוסף תהיה מערכת של משתמשים שיוכלו להתחבר ואו להירשם למערכת ולצבור סטטיסטיות אישיות במהלך.

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

סקירת הספרות (החלק העיוני)

משחק הדמeka

דמeka הינו משחק לוח אסטרטגי, אשר משוחק על לוח בן 64 משבצות ומועד 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 אבני או מסע אפשרי מוכרז כפסיד.

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

מקורות של המשחק נמצאים במצרים העתיקה, ולאחר מכן המזיכרים לוחות דמeka משנת 3000 לפני הס' נתקלו בעיראק. המשחק לא נמצא בגרסתו הנוכחית, אלא התפתח מהמשחק העתיק "אל-קורקווה".

חוקים ואופן המשחק

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

אבני המשחק

תנוועה - כל שחקן בתورو מניע אבן-משחק בצורה אלכסונית רק בכיוון היריב ("מעלה"). על המשבצת להיות פנויה מכלים. ככלומר, לכל אבן יש שתי אפשרויות תנוועה, ימינה ושמאליה, למעט אם האבן נמצאת בשולי הלוח, ועל כן יש לה אפשרות אחת בלבד.

אכילה (דילוג) - מתאפשרת כאשר מימין או משמאלי (באלאנסו) לאבן ישנה אבן ייריב, ומעבר אותה אבן ייריב, באלאנסו שבאותו הכיוון, קיימת משבצת ריקה. האכילה מתבצעת על ידי הזזה האבן אל המשבצת הפנויה שמאחורי אבן היריב, והזאת אבן היריב מהלוח. אם בתום האכילה נוצרת אפשרות לאכילה נוספת, יכול השחקן לבצע אותה ברציפות באותו התור (לא בהכרח באלאנסו שבאותו הכיוון). בדומה לתנועת האבניים, לא ניתן לבצע אכילה "אחריה" (כלומר לא בכיוון היריב). בחוגים מסוימים של המשחק קיימת חובת אכילה, ככלומר אם שחקן לא אכל אבן ייריב, אותה אבן יהיה באפשרותו לאכול נחשת פסולה והיא מוצאת מן הלוח.

מלכות (נקראות "מלכים" בגרסאות הלועזיות) - כאשר אבן מגיעה לשורה האחורונה בלוח (ביחס למיקומה ההתחלתי), היא הופכת להיות "מלך", ומסומנת לרוב על ידי שני אבני משחק אשר מונחות אחת על השניה. המלכה, בניגוד לאבני הרגילות, יכולה לנوع (אלכסונית) ולבצע אכילות בכל כיוון. המלכה יכולה לזרז משבצת אחת בלבד, אם כי בגרסאות שונות של המשחק ביכולתה לזרז 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 אבני על הלוח.
- לשחקן היריב אין אפשרות לבצע מהלך (מאחר ואבניו חסומות).
- השחקן היריב נכנע / פורש מהמשחק.

תיקו, לעומת זאת, יכול להיותמושג באחת מן הדרכים הבאות:

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

מבוא לתוכנות מונחה עצמים בשפת פיתון

תוכנות מונחה עצמים (Object-Oriented Programming, או בקיצור OOP), הוא פרדיגמה (סט מוסכמת) לכתיבת תוכנה, אשר משתמש בעצים (למענה אובייקטים) לשם פיתוח תוכניות מחשב. הפרדיגמה מאופיינת במודולריות, מדרגות (scalability) ובאבסטרקציה (הפשטה), ומרחיב התוכנה שבה כולל אובייקטים בעלי יחסים היררכיים ביניהם. כל ישות תוכניתה במערכת היא אובייקט מסווג כלשהו / מחלוקת בעלת מאפיינים ופעולות משלה, הקיימת כיחידה סגורה ועצמאית.

פרדיגמת OOP מספקת לתוכנת מספר דרכי לארגן, פשוט וליעיל את הכתיבה והפיתוח, בין השאר מכיוון שקיימת הפרדה בין המימוש הפנימי לבין המשק החיצוני של כל מחלוקת.

הפרדיגמה הייתה מהפכה בכתיבה תוכנה והחלה לשמש בפיתוח תוכנה החל מראשית שנות ה-80 של המאה ה-20, אך השימוש בשלמותה החל כעשור לאחר מכן.

כיום, מרבית שפות התוכנות המודרניות תומכות בתוכנות מונחה עצמים, והוא נחשב לאחת מפרדיגמות הפיתוח המובילות והנפוצות ביותר בעולם הפיתוח.

מחלקה

מחלקה (class) היא המבגרת הבסיסית ביותר של תוכנות מונחה עצמים. מחלוקת היא למעשה אוסף של משתנים, תוכנות, ופונקציות (הנקראות מותודות / שיטות), אשר מוגדים לבנייה לוגי אחד ופועלים יחדיו. מבחינה מהותית, מחלוקת מתארת ישות כללית מופשטת, אשר ניתנת למימוש בפועל באמצעות יצרת עצם (אובייקט) כמופיע (instance) של אותה מחלוקת. מופיע המחלוקת מכיל מידע אודות אותו העצם, וכל מותודות (פונקציות) המאפשרות לבצע פעולות על העצם.

```
class Example:  
    pass  
  
# איור א' – הגדרת מחלוקת
```

באיור א' נראה כיצד מגדירים מחלוקת בפייתון : נכתב את המילה השמורה "class" ומיד אחריה את שם המחלוקת שנבחר.

```
ex1 = Example()  
# איור ב' – יצרת מופיע
```

לשם יוצרת מופיע של המחלוקת, נוצר עצם חדש (כאמור מחוץ לגוף המחלוקת) על ידי שימוש בשם המחלוקת עם סוגרים מיד אחריו, כפי שמתואר באיור ב'.

כרגע, המחלוקת ריקה לחלוון ואנייה מכילה דבר. באפשרותנו להוסיף משתנים קבועים למחלוקת, יהיה זהים בעבר כל מופיע שלה. לשם כך, נכתבו בגוף המחלוקת את שם המשתנה וערךו. אומנם, לרוב נרצה במסגרת המחלוקת להגדיר משתנים כללים, שערכם יכול להיות מוגדר באופן ספציפי עבור כל מופיע. למשל, יוכל ליצור משתנה כללי בשם "age", ללא כל ערך בגוף המחלוקת, אלא שבעת יצרת כל מופיע יוכל להעביר למשתנה ערך כלשהו. משתנים אלו נקראים תוכנות.

לשם כך, נctruct פועלה בונה (constructor). הפעולה הבונה היא למעשה אותם סוגרים שמצוירים לשם המחלוקת בעת יצרה של מופיע חדש - דרך אנו יכולים להעביר את אותם ערכים ייחודיים עבור המופיע שיצרנו. הפעולה הבונה מתחילה את העצם (מופיע המחלוקת) שיצרנו בזיכרונו עם אפשרות לערכים מהמשתמש, ומכוונה אותו לשימוש עתידי.

בפייתון, נכתבת הפעולה הבונה באמצעות המילה השמורה `__init__`, ובها נכתב את הפרמטרים שברצוננו שתתקבל עבור העצם.

עבור כל תכונה או מתודה, נשימוש במילת המפתח `self`, אשר למעשה הופכת את המשטנה לתכונה אשר ייחודית לכל מופע של המחלקה. `self` מאפשרת לנו ליציג, 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' – יצרת מופע עם ערכי תכונות
```

במחלקה לדוגמא "GoogleEmployee" (איור ג'), ניתן לראות שМОדרם 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 אותן כערך לאותן תכונות.

עתה נראה כיצד ניתן ליצור מופע מחלקה עם ערכי התכונות הרצויות: שם המופע, שם המחלקה בצויר סוגרים (למעשה הפעולה הבונה), ובתוכם את ערכי התכונות הרצויות שהכרנו. בukt (איור ד'), למעשה הגדרנו מופע של המחלקה `GoogleEmployee` בשם `emp1`, אשר תכונתו (היחודית) זה `"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' – הוספה תכונה אופציונלית
```

בukt נוסף למחלקה תכונה נוספת בשם `"level"`, שתציג את דרגת העובד בחברה. ידוע כי הדרגה הראשונית היא 1, וממנה מתחלים עובדים צעירים בהגעים אל החברה. אולם, ישנים מצבים בהם מגיעים אל החברה עובדים מנוסים יותר, שדרוגתם ההתחלטת גבוהה מ-1. אם כן, כיצד נוכל למשוך זאת בפעולת הבונה?

ובכן בפייתון, אפשרותנו להוסיף לפעולת הבונה פרמטרים אופציונליים עם ערך ברירת מחדל, ככלומר ככל שהיא לא תינן שגיאה במידה ולא העברנו אותם לפעולת בukt יצרת המופע (מיימוש – באירור ה'). אם למשל ביצירת `emp1` הינו מעבירים רק ארגומנט אחד, הינו מקבלים הודעה שגיאה.

```
emp2 = GoogleEmployee("Ronit", "R&D", 3)
# איור ו' – יצרת מופע עם העברת ערך לתכונה level
```

לאחר ההוספה, ערך התכונה `"level"` של `emp1` וכל מופע חדש שנוצר ולא נעביר לו ערך לתכונה הוא 1, ברירת המחדל. באירור 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' – הוספה שתי מתודות
```

לרוב, בתכונות 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, נוצר פעולות החזורה ועדכון (`& Getters`). פעולות החזורה כשםן כן מחזירות את ערך התכונה הספציפית, ופעולות העדכון מעדכנים את ערך התכונה. לדוגמה, עבור התכונה `"name"`, נוסיף למחלקה את שתי המתודות הבאות (אייר ז').

עתה נראה (איור ח') קריאה למתודות מהמחלקה הראשית - נדפיס את התוכנה "name" של המופיע emp1 (יודפס "Yossi"), ולאחר מכן נשנה את תוכנה זו. אם נחרור על פועלות ההדפסה, יודפס "Avi", הרי זה עתה שינינו את ערך התוכנה. מעבר לפועלות החזרה ועדרכו, יוכל להוסיף מתודות נוספות למחלקה.

ובכן, ראיינו כי קיימים שני סוגי של מתודות - אלה מובנות, כמו הפעולה הבונה `__init__`, וכלה שאנו יוצרים, דוגמת `get`-`set`.

```
print(emp1)
# Aiur T' - הדפסת מופיע
```

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

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

```
def __repr__(self):
    return f"Name: {self.name}, Job: {self.job_title}"
# Aiur Y' - הוספה המתודה __repr__
```

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

במחלקה (איור י'), שמחזירה למעשה ייצוג מחרוזתי מותאם אישית של המופיע, קיבל את הודעה

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

המודאמת אישית, יחד עם ערכי התוכנות של המופיע :emp1

```
def increment_level(self):
    self.level += 1
# Aiur C' - הגדלת מתודה נוספת
```

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

כלל, ראיינו כי מתודות הקסם המובנות מאופיינות בכך תחתון כפול לפני ואחרי שמו, בעוד מתודות מותאמות אישית מוגדרות כפונקציות רגילות.

```
from GEmpClass import GoogleEmployee
# Aiur L' - יבוא המחלקה מקובץ אחר
```

בעובדה אשר מתפרשת על יותר מקובץ אחד, ניתן לייצא וליבא בקבילות מחלקות מקובץ לקובץ, ולהשתמש בהן באופן חופשי. לדוגמה, אם לקובץ בו נמצאת המחלקה קוראים GEmpClass, וanno נמצאים בקובץ אחר ורוצים ליבא את המחלקה אליו, ניעזר בפקודות השמרות בפייתון, כפי שמוצג באיר ל'.
כעת יוכל להשתמש באופן חופשי במחלקה בקובץ הנוכחי.

ירושה (inheritance) היא יכולה לרשת את כל התכונות והmethod'ות של מחלקה אחרת. המחלקה אשר ממנה יורשים נקראת מחלקה 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

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

איור מי – ייצרת מחלקה יורשת

ההורשה. במקום להעתיק את כל הקוד מן המחלקה המקורית, נכתב את שמה בסוגרים בעת יצירת המחלקה החדשה, באופן הבא (איור מי) :

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

איור נ' – ייצרת מופע חדש והדפסתו

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

הפלט הינו :

Name: Avi, Job: Sales

למעשה, פיתון אכן מאפשר לנו לעשות זאת ; מבלי שהוספנו כלום לתת-מחלקה, כל התכונות והmethod'ות הורשו אליה, דבר שאיפשר לנו ליצור ולהדפיס מופע חדש.

אומנם, לא לחינם יצרנו את המחלקה החדשה. ברצוננו להוסיף לה תכונות וmethod'ות חדשות. לצורך כך, נפעל בבדיקה באופן לייצרת מחלקה רגילה, באמצעות פעולה בונה (מתודת האתחול `__init__`). פעולה הבונה נוסיף את כל הפרמטרים שהפעולה הבונה במחלקה העל GoogleEmployee מקבלת, בתוספת התכונות החדשניות הייחודיות לתת המחלקה.

בזכות ההורשה, אנו יכולים לזכיר את תהליך האתחול, באמצעות שימוש בפונקציה `super()`. פונקציה זו מדמה את מתודת האתחול של מחלקה העל, כלומר נעביר אליה את הפרמטרים החכרחים. לאחר מכן, נוסיף בכתיב הדרוש את התכונות החדשניות (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-מחלקה

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 עברות בירושה, כך שנctrיך להוסיף פעולות `Get` ו-`Set` רק לתכונה החדשה `company_name`. גם התכונה הקבועה `company_name` עברת בירושה. מנגד, התכונה `employees` לא עברת למחלקה העיל והיא אינה קשורה אליה.

פולימורפיזם

פולימורפיזם (polymorphism, בעברית - רב צורתיות) הוא עיקרונו תכונתי האומר כי יש לקרוא למетодות אשר מבצעות את אותו תפקיד בשם זהה. לדוגמה, הפונקציה המובנית למציאת אורך אובייקטים (`len`), מושתתת על עיקרונו זה. לא משנהאיזה טיפוס נעביר לה (מחרוזת / רשימה / מילון וכו'), היא עושה את התאימותיה הנדרשות ומהזירה את האורך.

```
def __repr__(self):
    return "Manager."
איור ע' – __repr__ בתת-מחלקה
```

נחזיר למחלקה החדש שלנו, `GoogleManager`. אחת התוספות שנרצה היא, שבעת הדפסת המופיע, קיבל בנוסף לשם ולתפקיד הودעה שהעובד שלנו הוא גם מנהל. אז, נוסיף את מתודת הקיטם `repr` במחלקה (איור ע'), וננסה להדפיס את המופיע. פלט:

```
Manager.
```

אולם - קיבלנו רק את ההודעה "Manager" במקום לקבל את ההודעה המלאה. זאת מכיוון שלמעשה ביצענו מה שמכונה "דרישה" (override) של המתודה - העלמנו את השימוש המקורי של המתודה `repr` למחלקת העיל.

```
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. בוגר הפונקציה, נבצע קריאה לפונקציה `super()`, המתקבלת לה במחלקת העיל 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

פייגיים (Pygame) הינה ספרייה פיתון חזקה-פלטפורמות (Cross-Platform), אשר פותחה לצורך כתיבת משחקי מחשב. היא כוללת בתוכה גרפיקה ממוחשבת דו-מימדית וספריות שמע, אשר פותחו במילויו לשימוש בשפת פיתון.

הספרייה פותחה לראשונה בשנת 2000 ע"י פיט שינר (Pete Shinners), אך זמן קצר לאחר מכן הופקה לפרויקט קהילתי. גרסה הראשונה של הספרייה יצאה ב-28 באוקטובר 2000. עד היום, פייגיים נותרה פרויקט קהילתי חינמי בעל קוד פתוח לחלוּtin, כתובות <https://github.com/pygame/pygame> פיתון, בעזרת שפות C, Cython ואסמבלי. נכון לכתיבת שורות אלו, הגרסה היציבה الأخيرة, Pygame 2.0, יצאה ב-28 באוקטובר 2020, במיוחד לכבוד יום הולדתה ה-20 של הספרייה.

לצורך יבוא הספרייה לקוד, ראשית יש להתקין אותה על המחשב באמצעות הפקודה `pip install pygame`. לאחר ההתקנה, נייבה את הספרייה לקובץ הנוכחי באמצעות החצירה `import pygame`. מיד לאחר מכן, לצורך אתחול הספרייה, נכתב `pygame.init()`. כתע, כל הספרייה עומדת לרשותנו.

באיור אי' (בעמוד הבא) נראה דוגמא לתוכנית pygame בסיסית. בשתי השורות הראשונות ביצעוו את הייבוא והאתחול.

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

לאחר מכן, יצרנו דגל בשם `running` לולאת `while` של התוכנית. כל עוד הלולאה רצה, כך גם חלונית pygame. בשורה 8, העברנו לפונקציה `set_caption` את השם שיפוייע בעבר כותרת החלונית. בהמשך גוף הלולאה, שזהו חלקה המרכזי של כל תוכנית pygame, נטפל בכל מה הקשור לאיןטראקטיה של המשתמש. בשורה 10, באמצעות הלולאה, ניתן לכל האירועים (events) שיכולים לקרות מבחינות המשמש. לדוגמה, בבלוק התנאי הראשון, אנו בודקים האם סוג האירוע הוא QUIT, כלומר המשתמש לחץ על כפתור האיקס שבחלונית. במידה והתנאי מתקיים, נפסיק את ריצת התוכנית. דוגמא נוספת היא בлок התנאי השני, אשר בודק אם המשתמש לחץ על העכבר. במידה וכן, יודפס (בפייתון) את קואורדינטות העכבר בעת הלחיצה. לתמונה זו של העכבר הגענו כפי שניתן לראות בעזרת הפונקציה `get_pos` על האובייקט `pygame.mouse`.

מחוץ לולאת `for`, מילאנו את החלונית בצבע שחור. הפונקציה `fill`, שהופעלה על החלונית, מקבלת כפרמטר טאפל בעל ערכי RGB של הצבע המבוקש. בשורה שלאחר מכן, יצרנו מלבן, בעזרת הפונקציה `rect.draw`. פונקציה זו מקבלת כפרמטר את המשטח עליו היא תיצור (החלונית), 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 בחלונית, מהן יתחליל המלבן.

בסיום בлок לולאת ה-while, علينا לדאוג שחלונית pygame תמשיך להתעדכן בכל רגע, בהתאם למה שקרה. לצורך כך, יש להשתמש בפונקציה pygame.display.update, שכשמה כן היא מעודכנת את המוצג בחלוןית בכל רגע. לבסוף, מחוץ לlolalla, 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' – דוגמא לתוכנית pygame

לסיום, בספרייה pygame קיימות אינספור פונקציות, פקודות ו כלים המאפשרים לתוכנת שלל אפשרויות מגוונות בפיתוח ובנית המשחק. ספרייה זו היא מבין הפופולריות ביותר בפייתון, ולא לחינם. לмерות מרכיבותה היחסית והרבגונית שבها, היא קלה מאוד ללמידה והסתגלות תוך כדי פיתוח. ישנו מדריכים רבים ברוחבי המרשתת, בתוספת כמובן אתר התיעוד (documentation) הרשמי של pygame אשר בהחלט מקלים על תהליך הפיתוח.

אלגוריתם המינימקס minimax

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

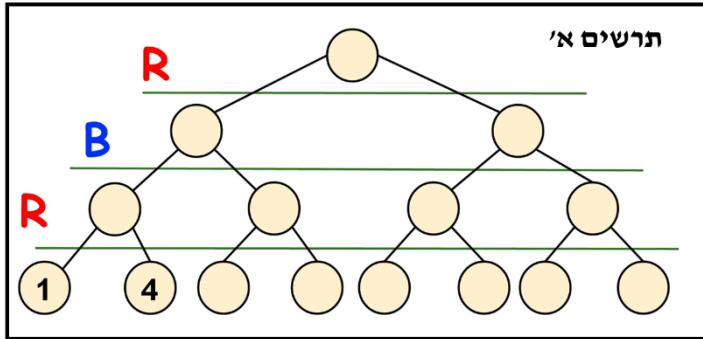
האלגוריתם משמש לרוב לאסטרטגיות משחקים סכום-אפס לשני שחקנים; משחק בו הרוח של צד אחד מאוזן על ידי הפסדו של הצד الآخر, כלומר סכום הרוח וההפסד של שני הצדדים הוא אפס. בכללות, האלגוריתם פורס את אפשרויות המהלך של שחקן אי, ולכל מהלך ניל - את תגובתו של שחקן ב', וכך לシリוגין עד עומק מסוים. כך, נוצר עץ המינימקס, תרשימים דמיוניים של המשחקים האפשריים בכל שלב המשחק החל מהתהoctה, ועד לעומק מסוים. עומק זה מוגבל מפאת החישובים הרבים שהוא מבצע, כמו נדרשים כוחות מחשביים רבים וזמן רב ככל שהעץ נעשה עמוק יותר.

שםו של האלגוריתם נגזר משני השחקנים אשר בהם הוא מתחשב - השחקן הממקסם (maximizer), והשחקן המזער (minimizer). מטרתו של השחקן הממקסם היא, כמובןו כוון הוא, למקסם את הרוחה שלו במהלך. בנגדוד ישיר לכך, נמצא השחקן המזער, שמטרתו היא למזער את הרוחה של השחקן הממקסם.

כעת, נשאלת השאלה כיצד מודדים רוח, הרי זה העיקרונו הבסיסי ביותר עליו האלגוריתם מסתמך. למעשה, במשחקי לוח, לצורךימוש האלגוריתם, מוצעים הערכות מספרית למצב הלוח הנוכחי, כך שהשחקן הממקסם שואף שהערכתה תהיה פלוט אינסוף, בעוד השחקן המזער שואף למינוס אינסוף. לדוגמה, ניתן להעריך בצורה בסיסית לוח דמיה ע"פ מספר החיילים של השחקן הממקסם פחות מספר החיילים של השחקן המזער: למשל, במידה לשחקן הממקסם יש יותר חיילים, תהיה לוח הערכה חיובית, בדוק כפי שואף הממקסם.

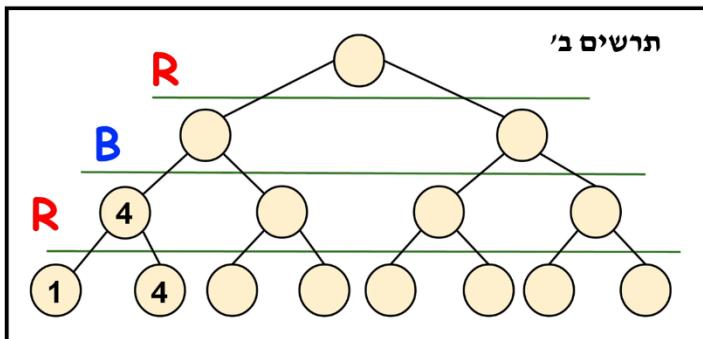
אלגוריתם המינימקס פועל בצורה רקורסיבית לצורך גילוי כל המהלך האפשריים, ועוצר בשכבה الأخيرة בעז (שכאמור נקבעת ע"פ העומק), וממנה הוא מתחילה לחשב ולהעריך מספורית כל מצב של הלוח.

כך, לאחר האלגוריתם עבור כל שחקן את המהלך הטוב ביותר עבורו (שחקן ממקסם - לוח בעל הערכה מספורית גבוהה ככל האפשר, ושחקן מזער - הערכה נמוכה ככל האפשר). כל שכבה בעז היא של שחקן אחר, ובבחירה מתבססת ע"פ השכבות התתונות לה, כפי שמייד נראה.

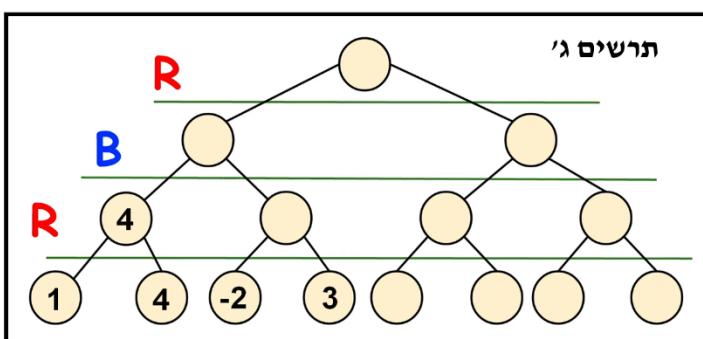


נראה דוגמא לעצם מהלכים בסיסי (תרשים א'). האיבר הראשון (העלון ביותר) בעצם מייצג את מצב הלווח הנוכחי. כפי שניתן לראות, זהו תור השחקן האדום, אשר שואף למקסם את התוצאה. כאמור, האלגוריתם מתחילה להעריך את מצב הלווחות מתחתייה העצם.

בענף השמאלי ביותר של העץ, נראה כי מצב לווח אחד הוערך כ-1, והשני כ-4. מכיוון שהוורו של השחקן האדום בעבר ענף זה, והוא השחקן הממקסם, הוא יבחר ב-4. אז, נוכל לתת לענף זה "cotract" של 4, מכיוון שהוורו שווה.



עתה (תרשים ב'), אם נסתכל על השורה השלישייה שהיא תורו של השחקן הכחול (המזען), הוא למעשה יודע שאם ילק לענף השמאלי, הלווח יהיה במצב של 4, מצב רע יחסית עבורו.

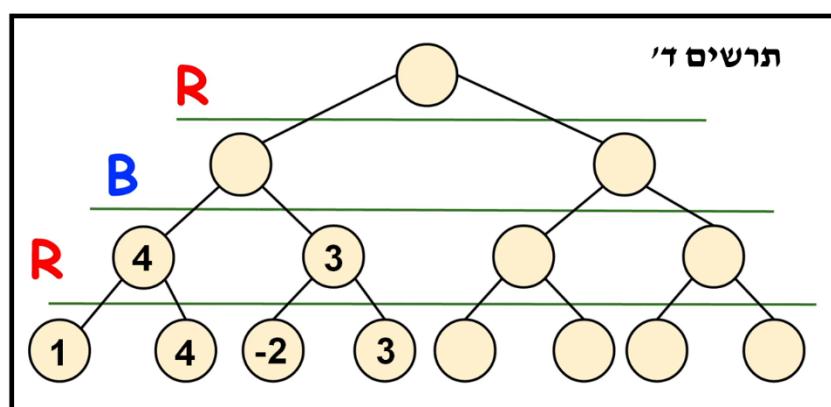


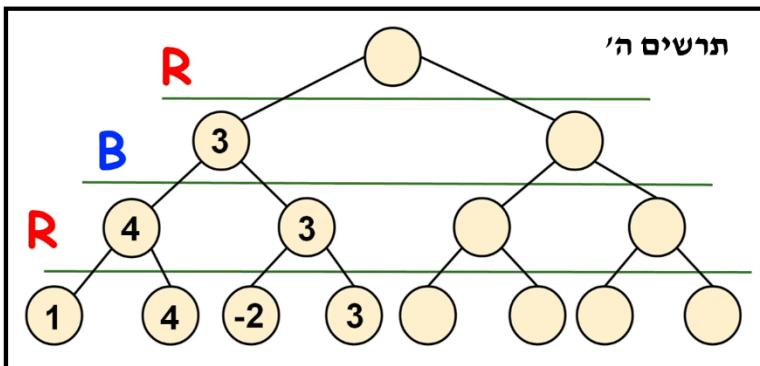
ນמשיך בສריקת המהלךים.

נביט בתרשים ג'. ובכן, האלגוריתם חישב ענף נוסף בעבר השחקן האדום, ונראה כי האפשרויות העומדות בפניו הן 3 ו-2. במה יבחר? כמובן שב-3, הרוי הוא שואף למקסם. ושוב, ניתן את ה"cotract" 3 לענף זה.

עתה (תרשים ד') נסתכל על הענף השמאלי שבשורה השנייה של העץ. זו כאמור נקודת מבטו של השחקן הכחול, השואף למזען. מפאת זאת, באיזו אפשרויות יבחר, 3 או 4? מן הסתם ש-3, הרוי הוא שואף למזען התוצאה.

כיווץ בזאת, נוכל לעלות שלב בעז ולתתcotract של 3 לכל הענף השמאלי. שוב, כל "cotract" זוatta מתבססת על כך שככל שהחקן בוחר את האפשרות הטובה ביותר ביותר עבורי.

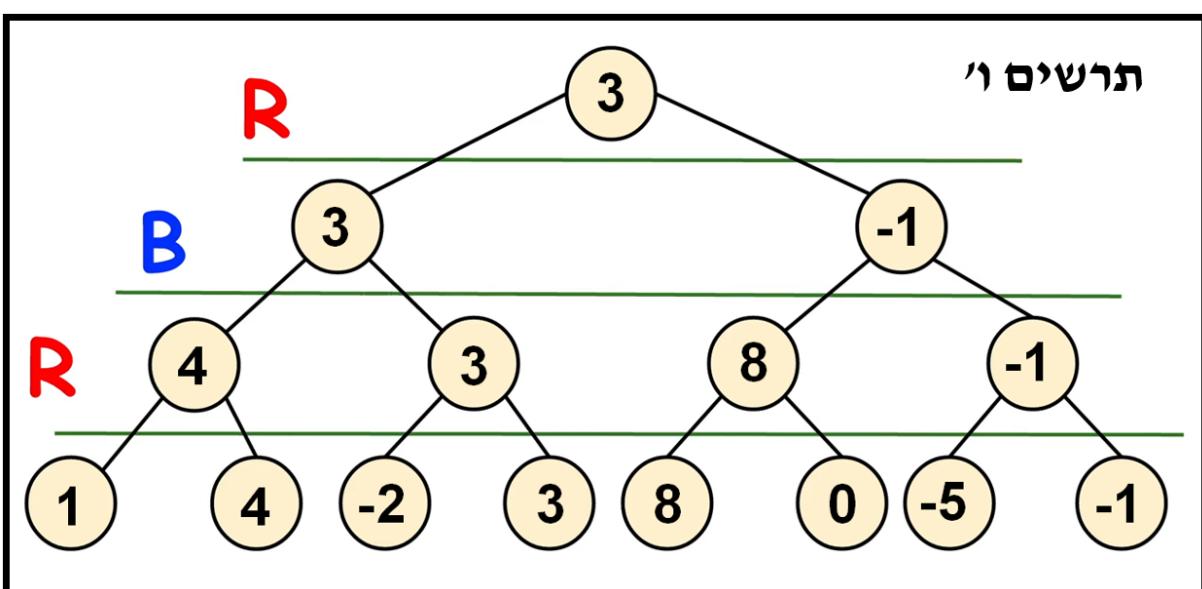




בתרשים ה', כבר סיימנו להעריך את כל הענף השמאלי, שערךו 3.

נניח והשחקן האדום יבחר ללקט בענף זה -
מגיע תורו של השחקן הכחול. עתה ניצבים מולו
3-4. הוא כמובן בוחר 3, והטור עובר לשחקן
האדום, שמולו ניצבים 3 ו-2. הוא בוחר -3
וכך למעשה הגיעו למסקנה שערך כל הענף
השמאלי 3.

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



האלגוריתם סיימם לפרסס את כל המהלך האפשריים, להעריך את מצב הלוח בכל אחד מהם ולבחר בצורה אופטימלית בעבר כל שחקן.

ערךו של העץ, עבור השחקן האדום הממקסם, הוא 3. מניתו הענף הימני עולה כי ערכו הוא -1, ועל כן השחקן האדום יבחר בענף השמאלי, שערך גובה יותר.

כਮובן שככל מעתה כל המשחקים ישנים מספר רב יותר של מהלכים אפשריים בכל רגע נתון, וע"ז המינימקס סבוך בהרבה. בנוסף, חשוב לציין כי בעת מימוש האלגוריתם, אין למעשה שימוש במבנה הנתונים עצ, כפי שנראה מיד. ע"ז המינימקס משמש להסברת והמחשת אופן פועלות 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 במקרה של מצב בו הושג ניצחון לאחד הצדדים, ואין טעם להמשיך לסרוק את המהלכים האפשריים.
3. נשתמש בכתיב: = לצורך השמת ערך במשתנה.

פונקציה מינימקס(מצב, עומק, שחקו_ממקסם) {

- אם עומק == 0 או המצב הוא סופי:
 - החזר את הערכתה של מצב
 - אם שחקו_ממקסם :
 - הערכה = $-\infty$
 - עברו כל מצב_אפשרי מותוך כל המצבים האפשריים הנגזרים במצב :
 - הערכתה : = **המקסימום מבין** (הערכתה, מינימקס(מצב_אפשרי, עומק פחות 1,
שקר))
 - החזר את הערכתה
 - אחרת (כלומר השחקן המזעיר) :
 - הערכה = $+\infty$
 - עברו כל מצב_אפשרי מותוך כל המצבים האפשריים הנגזרים במצב :
 - הערכתה : = **המינימום מבין** (הערכתה, מינימקס(מצב_אפשרי, עומק פחות 1,
אמת))
 - החזר את הערכתה
- {

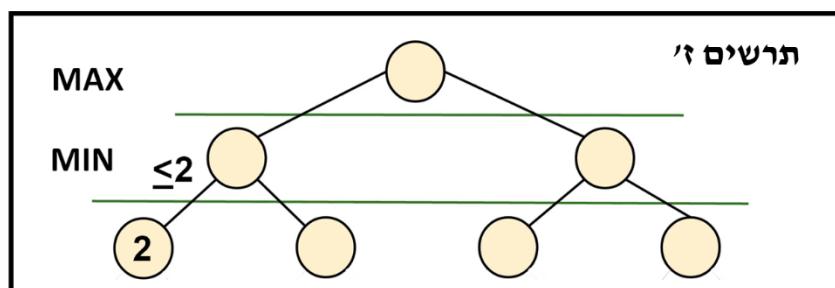
וכך נראה לפונקציה מותוך המחלקה הראשית:
מינימקס(מצב_nocchi, עומק, אמת)

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

אולם, קיים אלגוריתם אופטימיזציה עבור עצי-חישוף מסווג מינימקס. אלגוריתם זה נקרא **אייזום אלפא-ביתא** (Alpha-Beta Pruning). מטרתו של האלגוריתם היא לצמצם באופן דרמטי את מספר הענפים בהם יש לחפש ולסרוק. ככלומר, במהלך החישוף יזניח האלגוריתם פתרונות חלקיים אשר כבר ברור כי הם גורועים מפתרונות שכבר נמצאו.

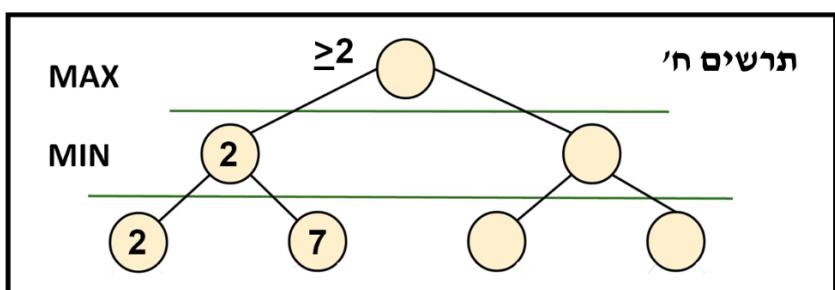
ואכן, לאחר שימושו, סיבוכיות הזמן יורדת בצורה חדה לעומת ממוצע של $O(b^d)$. לשם השוואה, אם במשחק דמקה יש בממוצע 7 מהלכים אפשריים בכל תור, ונרצה לרדת לעומק מהלכים של 5, קיבל $5^7 = 7,812$ לוחות העריך. אולם אם השתמש באופטימיזציה הגייזום, קיבל בממוצע $7^2 = 49$, ככלומר כ-130 בלבד.

עתה נראה דוגמא לאופן ביצוע הגייזום. הפעם, נשתמש בסימון "MAX" ו-"MIN" בכל ענפי העץ, לייצוג תורם של השחקן הממקסם והמזער, בהתאם.



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

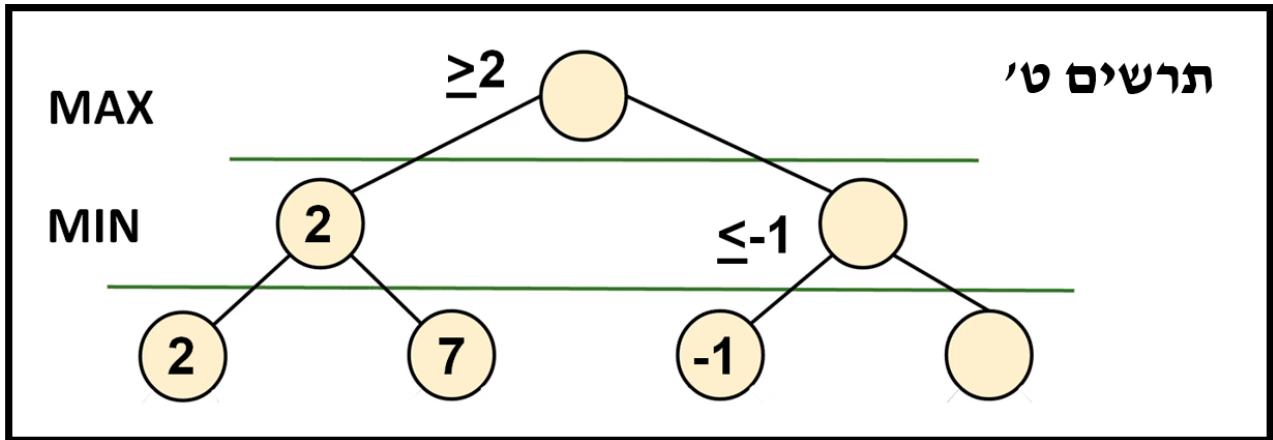
אולם אם יהיה מספר קטן מ-2, הוא יבחר בו במקום.



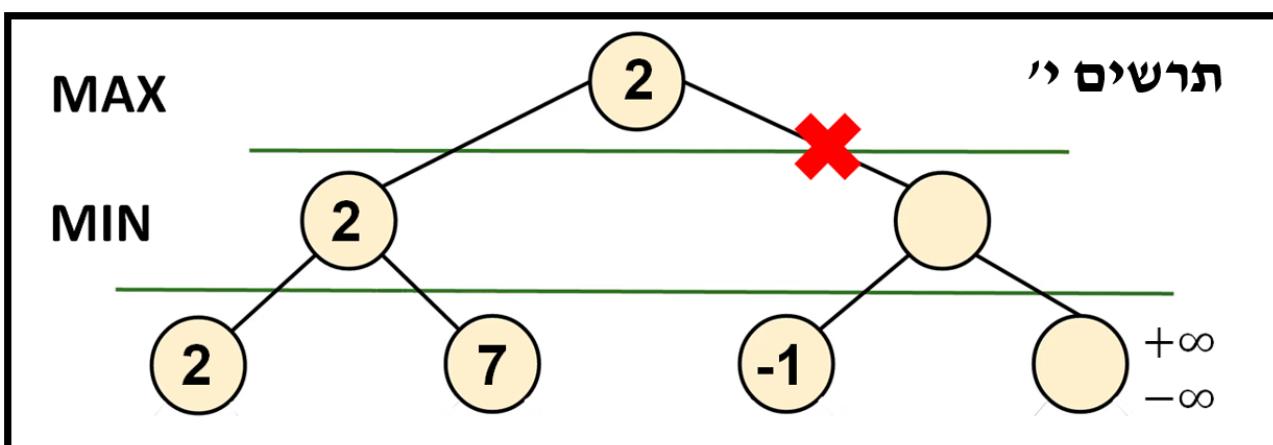
כעת (תרשים ח'), נראה כי ערך המצב השני בעץ הוא 7, אז השחקן המזער יבחר 2, וכן זו גם הכוורתה שניתן לענף זה. ובכן, מכיוון שערך הענף השמאלי בעץ הוא 2, השחקן הממקסם יודע כי לא משנה מה, באפשרותו להשיג ערך של "לכל הפחות – 2". אם בהמשך החישוף יתברר כי ערך הענף ימני קטן מ-2, השחקן הממקסם יבחר בענף השמאלי, בעוד אם הערך יהיה גבוה מ-2, יבחר ימני.

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

התשובה טמונה בעובדה שראינו בתרשים ז' כי השחקן הממקסם יכול להשיג ערך ≥ 2 . לעומת זאת, אם השחקן הממקסם בכלל יגיע במצב כזה, בו הערך יהיה $1 \leq$? - התשובה היא כמובן שלא, ולכן באותו הרגע הענף הימני הופך ללא רלוונטי לחלוטין.



תרשים י' נבון לבדוק למה ערך העץ הוא 2, ומדובר בכך להסיק כי הענף הימני למעשה "לא קיים".
נניח שבמצב הימני ביותר בעץ, הלא הוא זה שפסלנו, היה פלוט אינסוף - השחקן הממזער היה בוחר ב-1,-. לעומת זאת הענף הימני היה 1-. כיוצאה בזו, השחקן הממקסם יעדיף 2 על פני 1-.
באותו אופן, במידה והערך היה מינוס אינסוף - השחקן הממזער היה בוחר בו, לעומת ערך הענף היה מינוס אינסוף, וכמוון, השחקן הממקסם יעדיף את הערך 2 על פני מינוס אינסוף.



לסיכום התרשיים, ראיינו דוגמא בסיסית למדיד לאופן בו מתבצע הגיזום. בפועל, כאשר העץ סבוך בהרבה, לעיתים קרובות מגיעים במצבים בו ענפים שלמים נפסלים בזיה אחר זה. לפסילת הענפים יש יתרון כפול - מצד אחד, לא צריך להעיריך 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 נגור שמו של האלגוריתם). את α נאותחל במינוס אינסוף, והיא תיעציג את הערך המינימלי שהשחקן הממקסם יכול להשיג. באופן זהה, את β נאותחל בפלוס אינסוף, והיא תיעציג את הערך המקסימלי שהשחקן המזער יכול להשיג. כלומר, שני השחקנים מתחילה בתוצאה הגדולה ביותר האפשרית עבורם.

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

במהלך ריצת האלגוריתם, אם מתקיים $\beta \geq \alpha$, באותו רגע ניתן להחשב את הענף הנוכחי ללא רלוונטי. במקרה אחרות, ברגע שהשחקן המזער 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 بصورة הטובה ביותר. כשהגענו למצב שערכו היה 1-, אלפא הייתה 2, וזה עתה עדכנו את ביתא ל-1-. ברגע שהתקיימים $1- \geq 2$, או בכללiot $\beta \geq \alpha$, פסלו את הענף.

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

להלן הפסאודו-קוד של אלגוריתם המינימקס בתוספת אלגוריתם האופטימיזציה ניזום אלף-ביתא :

פונקציה אלף_ביתא(מצב, עומק, α , β , שחkon_מינימקס) {

- אם עומק == 0 או המצב הוא סופי :
 - החזר את ההערכה של מצב
- אם שחkon_מינימקס :
 - הערכה = $-\infty$
 - עברו כל מצב_אפשרי מתוך כל המצבים האפשריים הנגזרים ממצב :
 - הערכה = **המקסימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, שקר))
 - $\alpha = \text{המקסימום מבין}(\alpha, \text{הערכה})$
 - אם $\alpha \geq \beta$:
 - **הפסק**
 - החזר את הערכה
- אחרת (כלומר השחקן המזעיר) :
 - הערכה = $+\infty$
 - עברו כל מצב_אפשרי מתוך כל המצבים האפשריים הנגזרים ממצב :
 - הערכה = **המינימום מבין** (הערכה, מינימקס(מצב_אפשרי, עומק פחות 1, אמת))
 - $\beta = \text{המינימום מבין}(\beta, \text{הערכה})$
 - אם $\alpha \geq \beta$:
 - **הפסק**
 - החזר את הערכה

וכך נראה לפונקציה מתוך המחלקה הראשית :

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

מבוא לשפת SQL

רקע כללי

שפה SQL, או בשמה המלא "שפה שאלות מובנית" (Structured Query Language), הינה שפה תכנות הצהרתית* אשר משמשת לטיפול ועיבוד מידע במסדי נתונים יחסיים (Relational Databases). השפה פותחה ע"י חברת IBM בהובלת דונלד צימברליין וריימונד בויס בתחילת שנות ה-70, במקביל לפיתוח מסד הנתונים היחסי הראשון באוניברסיטת MIT. בתחילת, השפה נקראה SEQUEL, אך מאוחר יותר שמה הוחלף ל-SQL מטעמים מסחריים. כאמור, תפקיד השפה (בעת פיתוחה) היה תשאל ומניפולציה על נתונים במסגרת מסדי נתונים יחסיים. מהר מאוד חברות ותאגידים שונים זיהו את הפוטנציאל הגלום בשפה ובריעוניותה, וב-1986 אומצה SQL כסטנדרט ע"י מכון התקנים האמריקני (ANSI) ושנה לאחר מכן ע"י ארגון התקינה הבינלאומי (ISO).

*שפה תכנות הצהרתית הינה שפה המבatta את הלוגיקה החישובית שבה ללא בקרת זרימה, אלא דרך אוסף של פקודות וזראות בזו אחר זו. לעומת זאת, השפות הללו למעשה מותאמות לכך ורек מה צריך לבצע, ולא כיצד לבצע זאת. מALLERY דוגמאות לשפות כאלה הן XAML, CSS, HTML ו-SQL.

SQL מבוססת על טבלאות, ועיקר עבודתה מתבצעת עם נתונים במודל טבלי רלציוני (בעל קשרי גומלין). על אף שמה, השפה אינה רק שפת שאלות, אלא למעשה שפת הנתונים המקיפה ביותר, המאפשרת קשת רחבה של עבודה מול מסדי נתונים: החל מיצירת טבלאות סטטיות, אינדקסים, יצרת טבלאות דינמיות מודומות, הגדרת אילוצים עסקיים ועוד. SQL עשויה זאת דרך קריית נתונים בדרכים מגוונות, הכוללות פעולות בין טבלאות (כגון צירוף נתונים טבליות ופעולות איחוד וחיתוך בין נתונים טבליות), וכלה במניפולציה על נתונים כמו הוספה רשומות, עדכון ומחיקתן, ואף אבטחת נתונים וניהול תנועות.

זכור, SQL אינה שפת תכנות מלאה, כיון שהcharsים בה פקודות לוגיות ומשפטים בקרה, ולכן יש בה קשיות שלרוב צריכה להיפטר על ידי אירוחה בשפת תכנות עילית. בשפה קיימות מיילים שמורות (שאין להשתמש בהן בשמות טבליות / שדות / אינדקסים), טיפוסי קבועים, משתנים, אופרטורים ופונקציות מובנות.

על אף שהשפה אינה רגילה 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 אינו מתבצע לפי סדר הכתיבה. בדרך כלל פסוקית SELECT תבוצע לקראת הסוף, כאשר הפסוקית הראשונה שתבוצע תהיה פסוקית FROM. כל שלב בעיבוד מייצר טבלה וירטואלית זמנית אשר מהווה את הקלט לשלב הבא. בשאלות SELECT מרכיבת במיוחד (שיש בה צירוף בין טבליות, תנאי סינון, הקבוצה, מניעת כפילויות וצמצום מספר הרשומות במספר מסוים) מספר הטבליות הווירטואליות עשוי להגיע אף לעשר.

תחביר ומאפיינים

הפקודות בשפה מחולקות למספר תחומים עיקריים :

- **אחזור נתונים**
 - SELECT - אחזור נתונים מתוך טבלה / כמה טבלאות.
- **מניפולציה 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) הניל אין כלל הפקודות הקיימות בשפה, אלא רק הבולטות והשימושיות ביותר שבחן.

בשפה קיימים 22 טיפוסי נתונים, כאשר הבולטים ביותר הם CHARACTER (טו 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-), != (שונה 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 (השוואה ל-NULL).

דוגמאות

בחירה 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
);
```

הוספה שורה חדשה לטבלה :

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

הוספה עמודה חדשה לטבלה :

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

מחיקת שורות בטבלה :

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

בחירה מידע באופן ספציפי יותר וסידורו :

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

מבוא לשימוש בMySQL

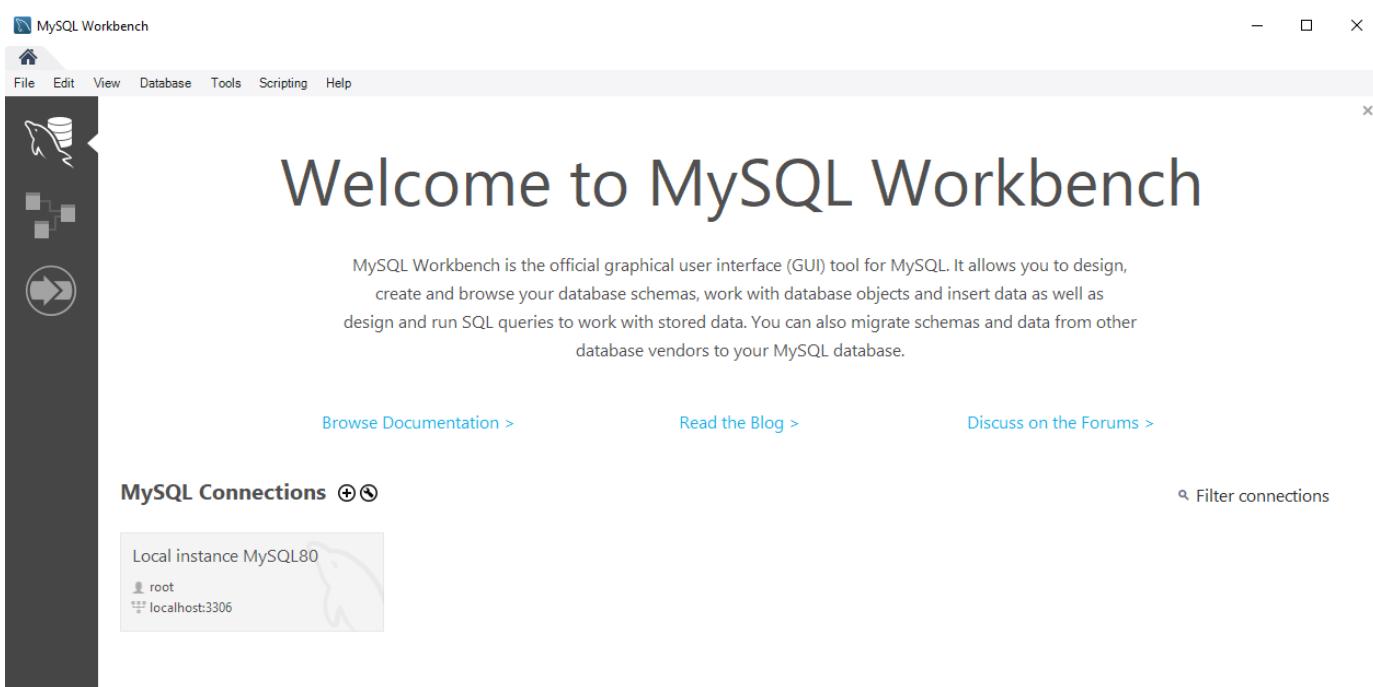
MySQL הינו מסד נתונים יחסית (טבלי) ורב משתמשים המבוסס על שפת SQL. התוכנה פותחה במקור ע"י החברה השוודית MySQL AB, וכיום היא בבעלות ענקית התוכנה האמריקאית אורקל (Oracle). בנוסף, המסד זמין בגרסה חינמית (MySQL Community Server) ופתוחה (open-source), או בגרסה מקצועית לtagידים (Enterprise Server).

MySQL נפוץ בתוכנות ובאתרי אינטרנט רבים הדורשים מסד נתונים, דוגמת ויקיפדיה, טוויטר, יוטיוב ופייסבוק. הוא נחשב כמסד קל וידידותי ללימוד ולשימוש באופן יחסית למסדי נתונים אחרים.

המסד כתוב בשפות C ו-C++, זמין על שלל מערכות הפעלה, כגון Windows, Mac, Linux ועוד.

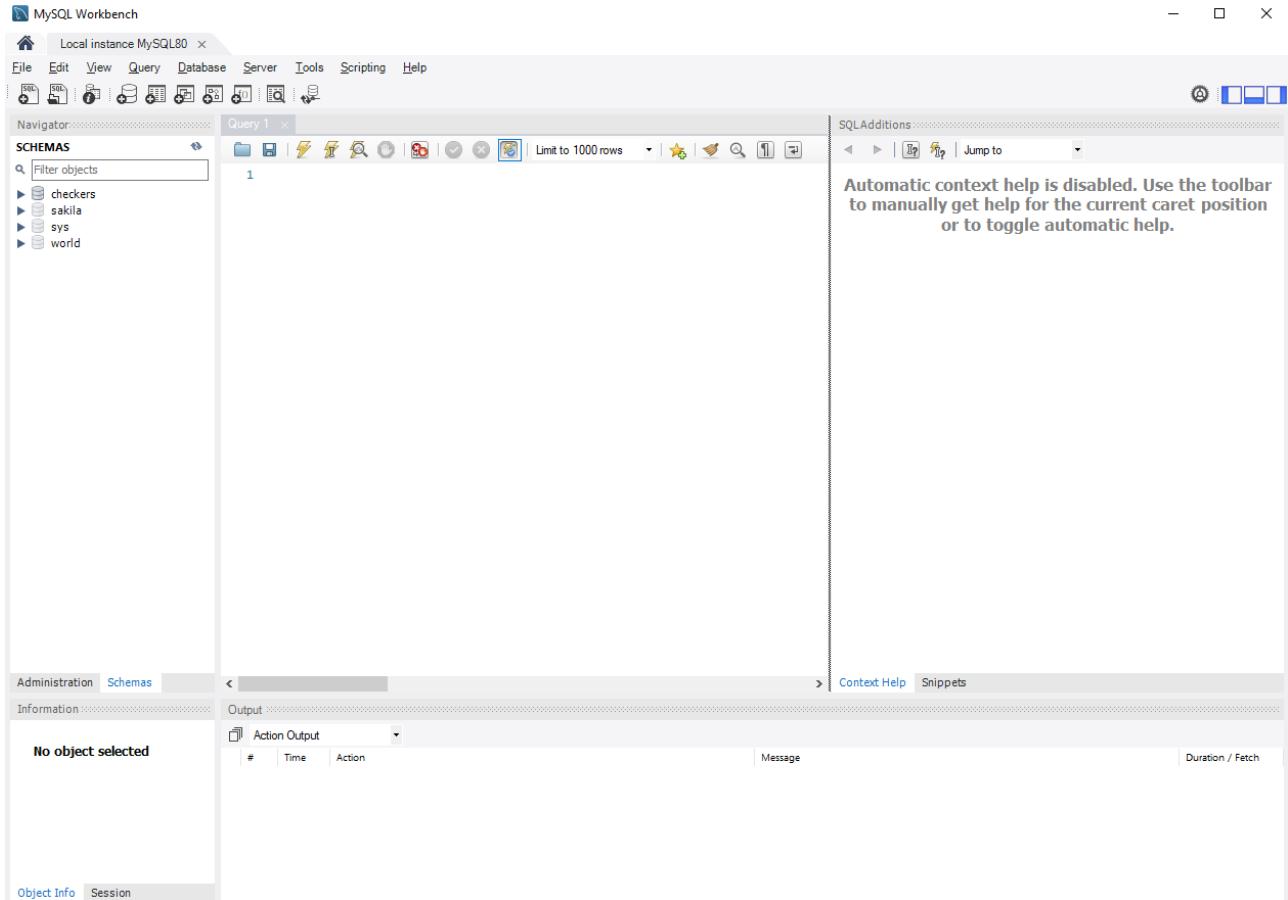
דרך טובה להשתמש ב-MySQL, היא להתקין את התוכנה MySQL Workbench על המחשב האישי. MySQL Workbench הינה תוכנה ויזואלית לעיצוב וניהול מסדי נתונים יחסיים מסוג SQL, אשר משלבת פיתוח בשפת SQL, אדמיניסטרציה (ניהול), עיצוב, יצירה ותחזוק של אוטם מסדי נתונים. תוכנה זו היא אחת מהנפוצות והשימושיות ביותר בתחום התוכנה, ומגיעה למיליאוני הורדות מדי שנה.

לאחר הורדה, הרשמה והתקנה מלאה, יש לפתח את התוכנה, שנראית כך:



ניתן לראות שישנו חיבור בירית מחדל, אשר משתמש לצורך תקשורת עם שרת ה-MySQL, ובאופן ספציפי יותר - עם מסד הנתונים הנוכחי של המשתמש.

להתחלה העבודה, יש ללחוץ על החיבור המבוקש, וכך להתחבר יש להקיש את הסיסמה שהוגדרה מראש ע"י המשתמש. עתה, מסך התוכנה נראה כך:



מצד שמאל למעלה נמצא סרגל הניווט. בסרגל הניווט ניתן לנוחות בין מסדי הנתונים והטבלאות השונות, וכן לנוחות בין פעולות האדמיניסטרציה, התחזוקה והביצועים. במרכז, ממוקם עורך השאלות, אשר בו ניתן לכתוב, לעורך ולהריץ שאלות. הפלט לשאלות הללו נמצא מתחת לעורך בכתובת "Output". בצד ימין, נמצא מסך התוספות וכלים העזר ל-SQL. לבסוף, בחלק העליון ביותר של המסך, ממוקם סרגל הכלים של התוכנה, שבו שולל פעולות ואפשרויות.

ליצירת מסד נתונים חדש, יש ליצור שאלתה חדשה בעורך השאלות, ולכטוב בה את הפקודה: "CREATE DATABASE mydb;". לאחר הריצה של השאלתה ע"י לחיצה על כפתור הברך, נוצר מסד נתונים חדש בשם .mydb.

על מנת לחבר את mydb לקוד (לשם הדוגמא - בשפת פיתון, בעזרת הספרייה mysql.connector, ראו

פרק בנושא), יש לכתוב את השורות הבאות:

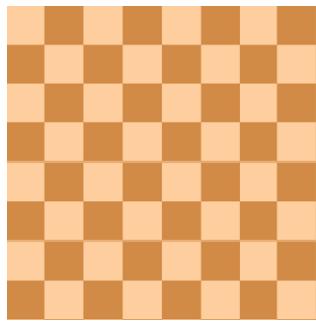
```
mydb = mysql.connector.connect(
    host='localhost',
    user='root',
    passwd='סיסמה',
    database='mydb'
)
```

כעת mydb הינו אובייקט מטיפוס MySQLConnection, ודרך ניתן לתחבר עם מסד הנתונים.

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

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

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



איור א' – לוח המשחק

פיתוח התוכנה (החלק המחקרי)

לוח המשחק

מטרה: פיתוח ויוזאלי של לוח משחק בן 64 משבצות, אשר יסודר בשני צבעים לシリוגין.

לצורך הפיתוח, ניצור מחלקה Board אשר תציג את הלוח.

נוסף למחלקה פועלה בונה ריקה (ככל שפיתוח המשחק יתקדם, כך גם המחלקה Board תשתכלל, אך בעת היא פשוטה יחסית), ובנוסף מתודה בשם draw_squares אשר תיצור ותציג בחלוןית ה-pygame (שמיידיה אותחלו בצורה ריבועית) את הלוח, כיהה למטרה (איור א').

בתוך draw_squares, נתחל 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, בעזרתו הפונקציה draw.rect (של pygame), ריבועצבע חום בגודל קבוע. כך למעשה, מקבל הלוח את צורתו.

חילוי המשחק

כלי החיליל

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

לצורך הפיתוח, ניצור מחלקה Piece אשר תציג כל אחת מ-24 חילוי המשחק (כל חיליל כמורפע מחלקה נפרד).

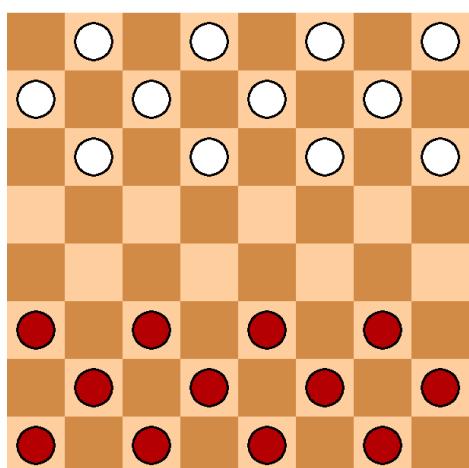
لمחלקה נוסיף שני משתנים קבועים, PADDING ו-OUTLINE (לצרכים ויוזואליים טכניים), ופעולה בונה בעלת חמישה תכונות; מס' שורה, מס' עמודה, צבע, ערך x של החיליל במסך וערך y תואם. ערכי ה-x ו-y מאוחלים לפי השורה והעמודה וכן לפי רוחב הריבוע בלוח (מספר קבוע).

בנוספ', נכתב מתודה בשם draw, אשר תפקידה יהיה להציג על המסך את החיליל, באמצעות הפונקציה draw.circle (של pygame). המתודה תציג את החיליל בשורה ועמודה המתאימים לו (לפי התוכנות) ובצבע השיך לו.

במחלקה Board, נוסיף תוכנה board - רשימה דו-ממדית בגודל 8x8 שתכיל בתוכה את כל משבצות הלוח (בין אם הן ריקות או מכילות אובייקטי חילילים), ובנוסף לכך גם מתודה create_pieces, אשר תיצור מופעי חיל חדש ותציב אותו על גבי הלוח במקומות המתאים.

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

לבסוף (עדין במחלקה Board), ניצור מתודה draw אשר "תצייר" (תציג בצורה ויוזאלית בחלוןית) הן את הלוח והן את כל החילילים. היא תתחליל מלפנים למתודה draw_squares, ולאחר מכן תעבור בלולאה מקוונת על התוכנה board - במידה והתא הנוכחי לא מכיל 0, נקרא למתודה draw של מופע החיליל המאוחסן בתא זה. כך, כל החילילים יהיו מוצגים במקומות המתאים, ונקבל חלונית כמו באירור ב'.
(בפעלה הראשית ; (board.draw(WINDOW))



איור ב' – לוח המשחק במלואו

מטרה : חישוב והציגת המהלךים האפשריים לחיל מוסום בכל רגע נתון.

לצורך הפיתוח, ניצור במחלקה Board שתי متודות רקורסיביות, search_left ו-search_right, שכל אחת מהן תבדוק את המהלךים האפשריים בהתאם לכיוון האלכסון (אלכסון ימינה / שמאלה). שתיהן מקבלות שיטה פרמטרים ; שורת התחלת, שורת סיום, כיוון ("במורד" או "במעלה" הלוח), צבע, עמודת התחלת ורשימה בשם skipped_pieces שתכיל בתוכה אובייקטי חיילים עליהם עברנו במהלך הסריקה. כל אחת מהمتודות הללו מ חוזירה מילון בשם moves, שמכיל את כל המהלךים האפשריים. במילון זה, המפתחות הם טאפלים (tuple) הכוללים את מס' השורה והעמודה (או במילויים אחרות - משכצת בלוח), והערכים הם רשימות הכוולות את הערך/ים הנמצאים במשכצת/ות עברו אותו מהלך - 0 במידה ואין שם חייל, ואובייקט חייל במידה יש. השימוש בלשון רבים הוא מפאת האפשרות לאכילה ואף כמה אכילות במהלך בזdeck. המתודות קוראות לעצמן בצורה רקורסיבית ומוסיפות מהלך חדשים למילון עד להגעה לקופה הלוח.

לבסוף, ניצור מתודה get_valid_moves שתקבל אובייקט מטיפוס חייל כפרמטר, ותחזיר מילון מהליכים אפשריים עברו אותו החיל. המתודה מבון נזורת ב-search_left ו-search_right.

כעת נותר להציג את המהלךים האפשריים לחיל מוסום בעת לחיצה עליו. לצורך כך, ולצרכי פיתוח עתידיים נוספים, ניצור מחלקה Game שתציג את המשחק. היא תכלול שתי תוכנות - חלונית ה-pygame (תמונה הכרחית עברו כל פעולה ב-pygame) והלוח, כאמור מטיפוס Board.

בתוך Game, נווסף מתודה בשם draw_valid_moves, שתקבל את המילון moves כפרמטר. במידה והמשחק לא נגמר (תמונה במחלקה Board), נעביר על כל המהלךים שבמילון, ונשתמש שוב בפונקציה pygame (של pygame) על מנת לצייר עיגול כחול קטן במרכז משכצת המהלך.

פעולות ויכולות החיל

מטרה : אפשרות תזוזה ואכילה לכל חילי המשחק.

נתמך כעת במחלקה Piece, כשהמטרה שלנו היא להעביר אובייקט החיל כלשהו ממוקם אחד בלוח למקום אחר, ככלומר לשנות את מיקומו בהתאם ברשימה הדו-מדית board (במחלקה Board).
ראשית ניצור מתודה בשם calculate_pos, שתעדכן את תכונות ערכי ה-x וה-y של החיל, ע"פ מיקום המשבצת שבו הוא נמצא. בנוסף, ניצור מתודה בשם move, שתקבל כפרמטרים מס' שורה ועומדה חדשים. המתודה תעדכן את הערכים הנוכחיים של החיל ללא שהתקבלו, ולאחר מכן תקרה למתודה calculate_pos

בחזרה למחלקה Board, ניצור מתודה move_piece שתקבל כפרמטרים אובייקט מטיפוס חיל (שברצוננו להזיז) ומס' שורה ועומדה (למעשה מיקומי המשבצת החדשה). המתודה תעביר את אובייקט החיל למקום המתאים ברשימה הדו-מדית board, וכך בפעם הבאה שהמתודה draw מופעלת (דבר הקורה בכל אלףות השניה בעזרת pygame.display.update), החיל יוצג בשבצת החדשה. בחירת החיל עצמה באמצעות לחיצת עכבר, מתבצעת על ידי מתודה נוספת בשם select_piece, שעוקבת אחר לחיצות המשתמש ומחזירה אמת אם הוא אכן לחץ על חיל בצבא התואם לתור הנוכחי.

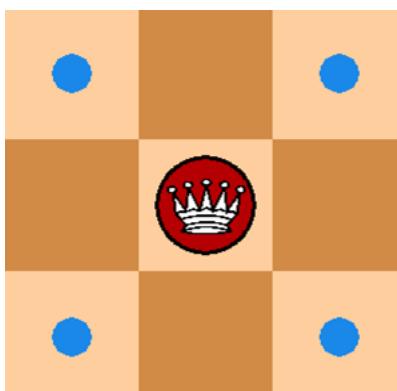
בפועל, תזוזות החילים מתבצעות מהמחלקה ה"מרכזית", Game. שם, ניצור מתודה make_move, שתקבל כפרמטרים מס' שורה ועומדה אליהם החיל יוזז. המתודה מבצעת את השינוי, בעזרת שימוש בכל המתודות שהגדנו קודם לכן.

כלי המלכה

מטרה : הפיקת חיל רגיל לכלי מלכה, בעט שהוא מגיע לסוף הלוח ביחס למיקומו ההתחלתי. באפשרות המלכה לנوع לכל כיוון, והיא תתאפשר על ידי סימון כתר מיוחד.

שם כך, נוסיף תכונה בוליאנית בשם `move_piece` שתאותחל ל-`False` במחלקה `Piece`. במתודת `search_right` במחלקה `Board`, נבצע בדיקה לראות האם המיקום החדש משמעותו מלכה, ובמידה וכן נעדרן את ערך התכונה של אובייקט החיל. ערך תכונה זו יבוא לידי ביטוי במתודות המהלך האפשריים - בה, במידה וחיליל אכן מלכה, מתודות החיפוש `search_left`-`search_right` יפעלו פעמיים כל אחת, במקומות פעם אחת בלבד. זאת, כאמור, מכיוון שביכולתה של המלכה לנوع לכל כיוון, ולא רק קידמה (ביחס למיקומה ההתחלתי). על כן, מהלכים בכל אחד מארבעת האלכסונים נבדקים ונאספים למלון `moves`.

לבסוף, במתודת `draw` שב-`Piece`, בעט ציור כל החילילים, נבדוק האם החיל הנוכחי הוא מלכה, ובמידה וכן נוסיף לו כתר מלכה. באյור ג' ניתן לראות מלכה בצבע אדום, ומסביבה את המהלך האפשריים עבורה, מסומנים בעיגולים כחולים קטנים.



איור ג' – מלכה, לאחר לחיצה עלייה

מיומש יכולת המשחק לשני שחקנים

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

כאמור, כבר פיתחנו את יכולות התזוזה והאכילה של החילים, וכן של המלכות. כעת, במחלקה Game, ניעזר בתוכונה `next_turn`, ששויה למשהו אדום או לבן. את התוכנה נачזור לאדם באופן שולי, ככלומר השחקן האדום יבצע את המהלך הראשון. כעת, לאחר כל מהלך, נקרה למתחודה `switch_turns`, שכשמה כן היא תחליף את התור. כזכור, לתוכונה `next_turn` יש חשיבות רבה בעקבות ביצוע המהלך, שכן היא קובעת אילו חילים (מאיזה צבע) יהיה ניתן להזיז.

לבסוף, ניצור במחלקה Board מתחודה בשם `check_win` אשר תבדוק האם הסתיים המשחק (על'פ חוקי משחק הדמקה). במידה ואכן הושג ניצחון של אחד הצדדים, התוכנה `is_game_over` תהיה אמת, ובכל חוסמת את המשחק ביצוע המהלים, ככלומר למשהו Noulett את הלוח. בנוסף, תוצג הודעה סמלית למשתמש, אשר מציעה לו לצאת מהמשחק או להתחיל משחק חדש. במידה ויבחר להתחיל משחק חדש, אובייקט המשחק (Game) יקרה לאחת מהמетодות שבו בשם `reset_game`, שתיצור לוח חדש ותאפשר את כל שאר התוכנות המשותפות.

במחלקה הראשית `main`, ניצור אובייקט חדש מטיפוס Game שיניהל את המשחק, ובוללת את `while` אינסופית, הכוללת את שלל פרוטודות `pygame` (טכני למד), חלוגיות המשחק עצמה רצתה בקצב רענון של 60 פרמיימרים לשנייה. בתוך הולאה, מעבר לייצוג הויזואלי, מתבצעת גם התקשרות עם המשתמש באמצעות כל "אירוע" (events) המקלט והעכבר, אשר נגישים דרך `.pygame`.

פיתוח שחקן AI

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

ראשית, בתפריט הראשי של המשחק, נוסף שני כפתורים - מצב 2 שחקנים או מצב שחקן יחיד, על מנת שהמשתמש יוכל לבחור במצב המשחק הרצוי. בנוסף, במחלקה הראשית נוצר 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 המהלים נקבעת בראש ובראשונה ע"פ הערכה (למעשה ציון) מספירת שניתנת לכל מצב של הלוח. כאמור החلطנו כי השחקן הלבן הוא ה-AI, ונחליט שגם יהיה השחקן הממקסם, כלומר הוא שואף להערכת לוח של פלוס אינסוף ($+\infty$). על כן, נוצר במחלקה `Board` מתודה `evaluate`, שתחזיר את אותה הערכה מספירתית, שתחושב באופן הבא :

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

כלומר, ההערכתה מורכבת מחיבור שני הפרשים ; הראשון - ההפרש בין מספר החילילים הלבנים למספר החילילים האדומים, והשני, שהוא ההפרש בין מספר המלכות הלבנות כפול 1.5, למספר המלכות האדומות כפול 1.5.

чисוב זה נעשה משני שיקולים עיקריים. ראשית, השחקן הלבן הוא הממקסם, כלומר הערכת חיובית לכל האפשר של הלוח טובה לו, ועל כן מספר הכלים (חילילים / מלכות) הלבנים הוא מהויר ולא המחר. השיקול השני הינו שערך המלכות במשחק גדול מערך החילילים, ועל כן יש לתת יותר משקל למלכות. השיקול הנ"ל בא לידי ביטוי בעצם הכפלת ב-5.1 את מספרי המלכות.

כעת, נוצר מחלוקת `SearchTree` (איור ד'), שתוביל ותרכו בתוכה את כל הקשור במציאות המהלך האופטימלי בעזרת אלגוריתם המינימקס וגיוזם אלף-ביתא. הפעולה הבונה כוללת שתי תכונות : `depth`, כולם עומק עץ חיפוש המהלים, ו-`player_color`, כלומר צבע השחקן הנוכחי.

לאחר מכן, נוצר מתודה `find_optimal_move`, שמקבלת אובייקט מטיפוס לוח כפרמטר. היא קוראת למתחודה המרכזית, `minimax_alpha_beta`, אשר מקבלת כפרמטר עומק, לוח, האם_ממקסם, אלף-ביתא. מתודה זו, אשר נכתבה כדי לבדוק ע"פ הפסאדו-קוד, מחזירה הערכת נוכחית ללוח ואת הלוח עצמו. הפלט הסופי הוא כאמור הערכת הטובה ביותר לשחקן ואת הלוח שנגרם כתוצאה מההלך. בפועל, כל המהלים הללו הם למעשה **ЛОחות** שנוצרו כתוצאה מההלך - כאשר המחשב "מבצע מהליך", הוא לא לוחץ / בוחר חיליל כלשהו ו"מזיז" אותו, אלא מחליט לבחור בלוח בעל הערכת המספרית התואמת ביותר עבורו.

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

מסביב ל-alpha_beta, ישנו עוד שתי מתודות עיקריות אשר דואגות לביצוע מהלכי המחשב. הראשונה, get_all_moves, מקבלת כפרמטר אובייקט מטיפוס Board, ומחזירה רשימה הכוללת את כל המהלך האפשריים עבורו אותו צבע שחקן (לפי התוכנה player_color) בהתאם ללוח הנוכחי. את סימולציית המהלך היללו, אשר יוצרת אינספור לוחות חדשים, get_all_moves עשויה להשתמש המתודה השניה, make_temp_move. פונקציה זו מקבלת לוח (למעשה עותק של הלוח המקורי), ולא אותו לוח עצמו, חיל, מהלך וחילים ש"נקפכו מעל" (כלומר החיל אכל אותם, במידה ויש כmobן), ומחזירה לוח חדש לאחר השינויים.

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

```
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 4' – מבט על המחלקה SearchTree והפונקציות הנוספות

(ימוש האלגוריתם בפייטון מבוסס בדיק על הפסאודו-קוד).

הפונקציה האחראית המשקפת תפקיד בשחקן AI-הינה `ai_move`, אשר נמצאת במחלקה Game, והיא מקבלת את אותו לוח שנבחר כתוצאה מההתקן האופטימי `alpha_beta`. היא מעודכנת את הלוח הנוכחי (כאמור תוכנה במחלקה Game) ללוח שהתקבל ומעבירה את התור למשתמש.

הדבר האחרון שנותר הוא בחירת דרגת הקושי.

לאחר שהמשתמש לחץ בתפריט הראשי על מצב "שחקן ייחיד", נפתח לו עוד מסך בעל שלושה כפתורים, אחד לכל דרגת קושי (קל, בינוני, קשה). הפענציה שעשויה זאת נקראת `select_difficulty`, והיא למעשה מחזירה את **עומק האלגוריתם** (בהתאם ללחיצת המשטמש) - ככל שהעומק רב יותר, כך מוחשבים 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 الأخيرة, "קשה", עומק של 4.

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

פיתוח מסך בית תפיריט ראשי

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

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

ראשית, כאשר מרייצים אתקובץ הפיתון, קוראים למשזה לפונקציה הראשית main. ובכן, main מייצגת את התפיריט הראשי של המשחק. בעזרת הפונקציה window.blit (של pygame), נגידר את רקע המסך לתמונה מעוצבת בהתאם אישית למשחק, אשר כוללת מיקומים לכפתורים השונים. ב-game, אין למעשה "כפתור", אלא יש אירועי עכבר / מקלדת (pygame events) - לדוגמה לחיצה שמאלית על העכבר. לאחר לחיצה כזו, מיד מזוהה ע"י pygame, ניתן לקבל את קואורדינטות סמן העכבר בעת הלחיצה. כך, ניתן לבדוק אם הסמן היה על אחד הכפתורים (לפי קואורדינטות הceptor שגמ ידועה), ובהתאם לכך להפעיל את הפונקציות הרצויות.

כיווץ בזאת, נוסיף בתפיריט 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-game,start_game, ללא כל תוספות (ברירת המחדל של המשתנה with_ai הוא False).
- כפטור כניסה למערכת - איקון בנוסח Login : בעת לחיצה נותן למשתמש אפשרות להיכנס למערכת עם שם משתמש וסיסמה (Login) או להירשם למערכת (Sign Up).
- כפטור פרופיל - איקון בדמות אדם : בעת לחיצה מוביל למסך פרופיל אישי, אשר כולל בתוכו מספר פרטים בסיסיים על המשתמש.
- כפטור סטטיסטיקות - איקון גרפ עמודות : בעת לחיצה מוביל למסך הסטטיסטיקות, במידה והתבצעה כניסה של משתמש רשום.
- כפטור עזרה - איקון סימן שאלה : בעת לחיצה מוביל למסך עזרה, הכולל הדרכה קצרה על שימוש בתוכנה ובנוסף את חוקי משחק הדמקה.
- כפטור Leaderboard - איקון פודיום : בעת לחיצה מוביל למסך ה-leaderboard, אשר מציג טבלה של עשרת המשתמשים בעלי מספר הניצחונות הרב ביותר.

- **כפתר פאולים - איקון פאול:** בעת לחיצה מוביל למסך הפאולים, אשר בו יכול המשתמש לבצע אותם. פאולים של דמeka הם למעשה מצלבים מותוק משחקים מן העבר, בהם יש מהלך אשר מוביל לניצחון. על המשתמש לשחק את המהלך (ואף סדרת המהלךים) אשר מוביל/ים (בסיומו של דבר) לניצחון. ישנו שישה פאולים סך הכל المسؤولים בדרגת קושי גבוהה, וברגע שהמשתמש מצליח את הפאול, נפתחת לו האפשרות לבצע את הפאול הבא.

פיתוח מסך הרשמה וכניסה למערכת

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

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

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

בקוד, נוסיף לכל הפונקציות (main_menu, start_game, user_id=None) פרמטר main_menu וכוי) פרמטר user_id=None. המטרה היא לדעת האם יש משתמש מחובר (ואם כן, איזה), וכתוכאה מכך בהתאם התוכנה למשתמש הנוכחי אשר מחובר.

בעזרת pygame, נוצר מסך חדש בשם login_page עבור הכניסה למערכת (מעוצב בהתאם). נוסיף שני שדות להזנת טקסט, אחד עבור שם המשתמש והשני עבור הסיסמה. לאחר שהמשתמש לחץ 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. בעזרתו נוכל להתחבר למסד הנתונים, ולבצע עליו / מיניו פעולות.

לאחר שה קישור למסד הנתונים הטענו כראוי, נבדוק את הנתונים שהזנו 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 נתונים ההתחברות

אנו למעשה מבצעים שאילתת מהזירה את כל הנתונים מטבלת המשתמשים, ובודקת האם הנתונים שהזנו תואמים אחת מהרשומות בטבלה. במידה ולא, מוצגת הודעה שגיאה. במידה כן, הפונקציה מחזירה את ה-ID של המשתמש, אחריה היא מחזירה -1. במקרה זה והוא לא שווה 1-) מועבר בכל קריאה לכל אחת מהפונקציות שמנהל את התוכנה.

עתה יש לטפל במשתמשים חדשים, אשר אין להם חשבון קיימים 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 תוויים והסיסמה באורך של לפחות שישה. במידה והתנאי מושג, נמשיך בבדיקה האם שם המשתמש כבר קיים במערכת. אם אחד מהתנאים הללו מופרדים, תוצג למשתמש הודעה 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' – הוספה המיידית בטבלה

*(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 סטטיסטיות

מטרה : פיתוח מסך אשר מציג את סטטיסטיות המשחקים עבור כל משתמש (לפי המשתמש שמחובר בעתה). על המסך להציג סטטיסטיות כגון מס' ניצחונות, הפסדים, מספר מהלכים מוצע 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 שמושכים:

- עמודה difficulty - רמת הקושי שモלה בחר השחקן לשחק* (מтиפוס מספר שלם).
 - עמודה gtime - זמן המשחק מתחילת ועד סופו (מтиפוס TIME בפורמט hh:mm:ss).
 - עמודה total_moves - מספר המהלךים הכלל שהתבצעו במהלך המשחק (מספר שלם).
 - עמודה userID - מפתח זר (foreign key) לעמודה ID בטבלה users. תקידה לשמור את ה-ID של המשתמש אשר שיחק את המשחק (מספר שלם).
 - עמודה won - האם המשתמש ניצח או הפסיד. מטיפוס TINYINT, קלומר 0 (הפסד) או 1 (נצח).

*סטטיסטיקות נצברות רק במשחקי מצב שחקנו יחד.

בוקוד, ניצור מחלקה חדשה בשם GameStats שתרכז את כל צבירת הסטטיסטיות על משחק.
להלן ממוקם המחלקה:

מетодה	תיאור
<pre data-bbox="328 1284 776 1493"><code>def __init__(self, ai_diff, user_id):</code></pre>	<p>פעולה בונה.</p> <p>בנוספַּ, מתחילה את התוכנות <code>won</code>, <code>start_time</code> ו-<code>total_moves</code> ל-0, ואת <code>time</code> ל-0, וזמן קרייה : בעת הפעלה של משחק חדש.</p>
<pre data-bbox="328 1493 776 1641"><code>def start_timer(self):</code></pre>	<p>מעדכנת את התוכנה <code>start_time</code> לזמן הנוכחי (בעזרת הספרייה <code>(time)</code>).</p> <p>זמן קרייה : לאחר ביצוע המהלך הראשון במשחק.</p>
<pre data-bbox="328 1641 776 1909"><code>def end_timer(self):</code></pre>	<p>מעדכנת את התוכנה <code>total_time</code> למשך הזמן הנוכחי בפורמט "ss:mm:ss", אשר מייצגת את הזמן הנוכחי בעת הקריאה פחותו הזמן בתוכנה (או ב的日子里 אחורות - הזמן הכלול של <code>start_time</code> (מיד לאחר סיום המשחק).</p>

<pre>def assign_winner(self, winner):</pre>	אם <code>winner</code> שווה ל"שחקן אדום" (כלומר השחקן, ולא המחשב), המתוודה מעדכנת את התוכנה <code>now</code> ל-1. זמן קריאה: מיד לאחר סיום המשחק.
<pre>def insert_stats_to_database(self):</pre>	מכניסה את כל נתונים המשחק (למענה תוכנות המחלקה) לטבלה <code>games</code> שבבסיס הנתונים. זמן קריאה: ברגע לאחר סיום המשחק.

במחלקה `Game`, במידה והמשחק הוא מצב שחקן יחיד, נוסיף תוכונה בשם `stats` שהיא למעשה אובייקט מטיבוס `GameStats`. כך, כל הסטטיסטיות מנוהלות בצורה אוטומטית בתוך המחלקה `Game` מתחילה המשחק (יצירת אובייקט סטטיסטיות חדש) ועד סוף (הכנסת נתונים המשחק למסד הנתונים).

כעת, נוצר מסך חדש בשם `stats_page`, עבור הצגת הסטטיסטיות של המשתמש המחבר CUT (כניסה למסך הסטטיסטיות מחיבור של משתמש פעיל, אחרית מוצגת הודעה המצינית 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 המחבר CUT (אյור א').

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

איור א' – שליפת כל המשחקים בעבור משתמש ספציפי

הפונקציה מסדרת את המידע בצורה נוחה ו邏輯ית, והאחרון הוא שמתפרק ב-`stats_page`. לאחר מכן, בעזרת `pygame` נציג את כל הסטטיסטיות:

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

בנוסף, מוצג מעין גרפ' עוגה, שחלקו (לפי יחס הניצחונות והפסדים) צבוע בירוק והשאר באדום. (ב모בנים מתמטיים - הגרף, מהזווית 0 ועד הזווית 360 צבוע בירוק, ומהזווית הניל ועד 360 צבוע באדום).

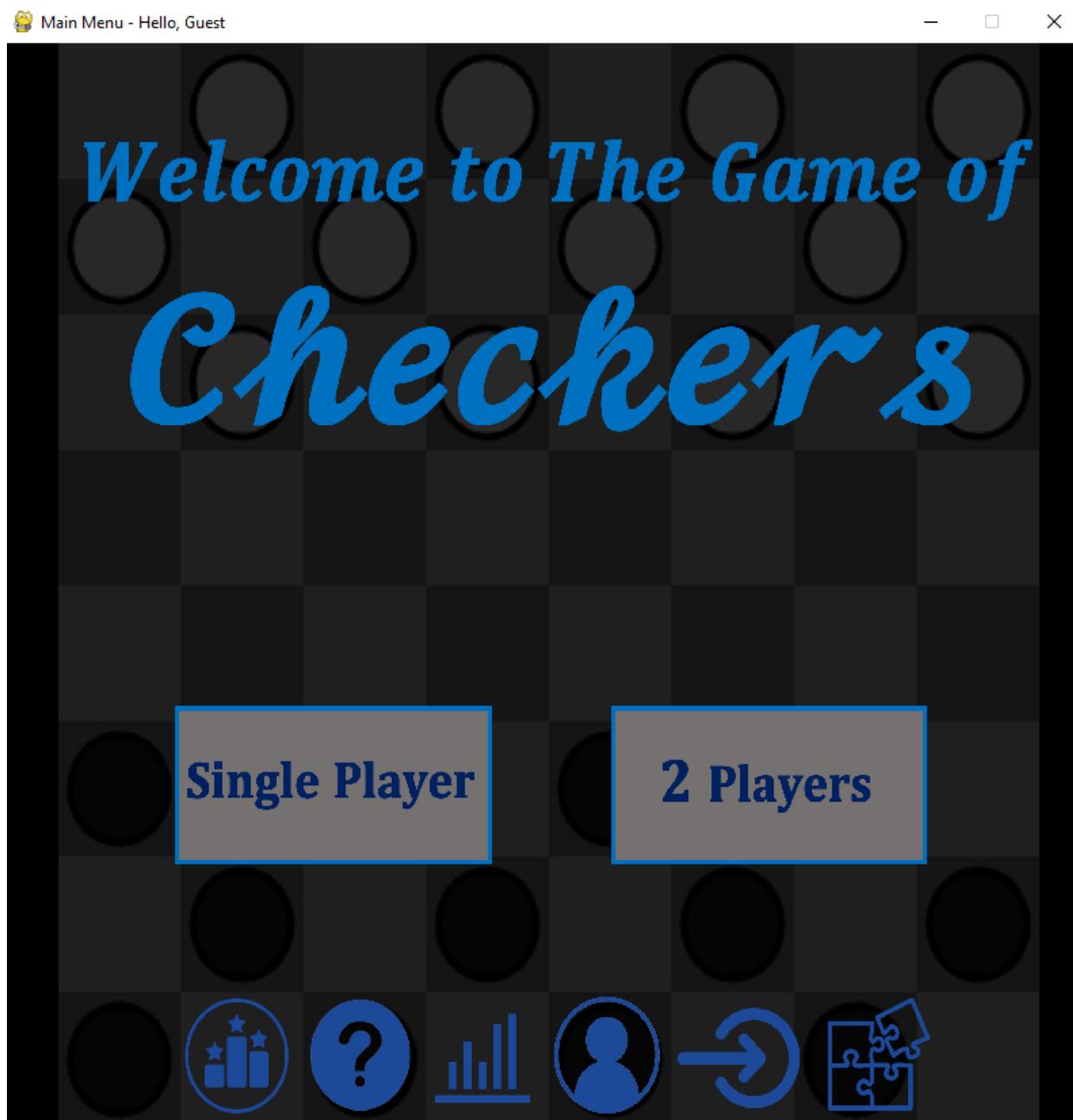
הציגת הפרויקט הסופי

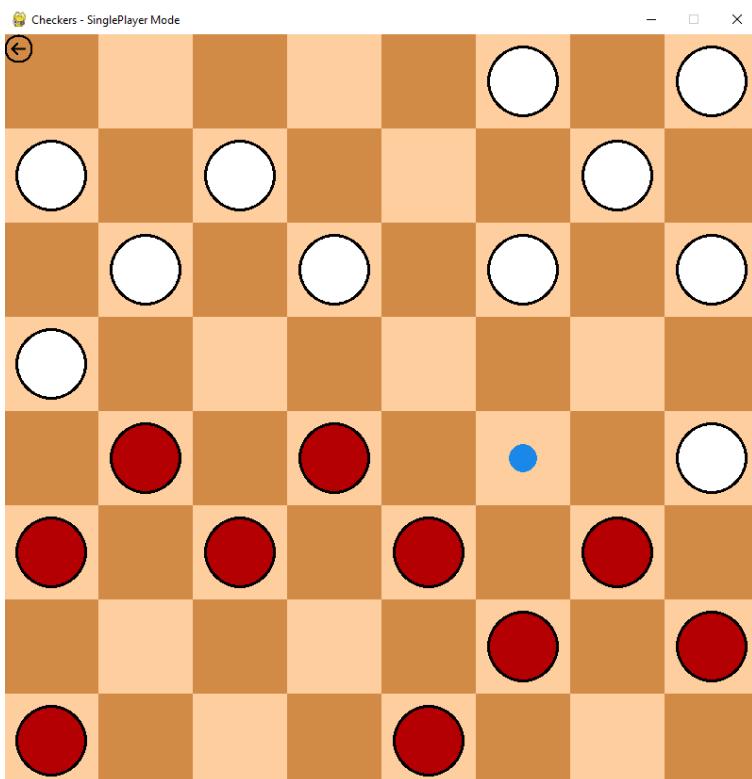
צילומי מסך נבחרים המציגים את התוכנה.

מסך הבית של התוכנה.

ניתן לראות את הគוורת 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 שחקו יחיד ומצב שני שחקנים, מטה (משמאלי) : טבלת הישגי המשתמשים (Leaderboard), מסך עזרה, סטטיסטיות, פרופיל, כניסה למערכת ופאלרים.





בצלום משמאל ניתן לראות קטע מותוֹך
משחק דמקה במצב שחקן יחיד.

בצלום מימין ניתן לראות ביצועו
של כניסה למערכת (Login, ע"י
הזנת שם משתמש וסיסמה.
כעת המשתמש נמצא בעט הקשת
הסיסמה, וכך שדה הסיסמה מוקף
במלבן סגול.

Login

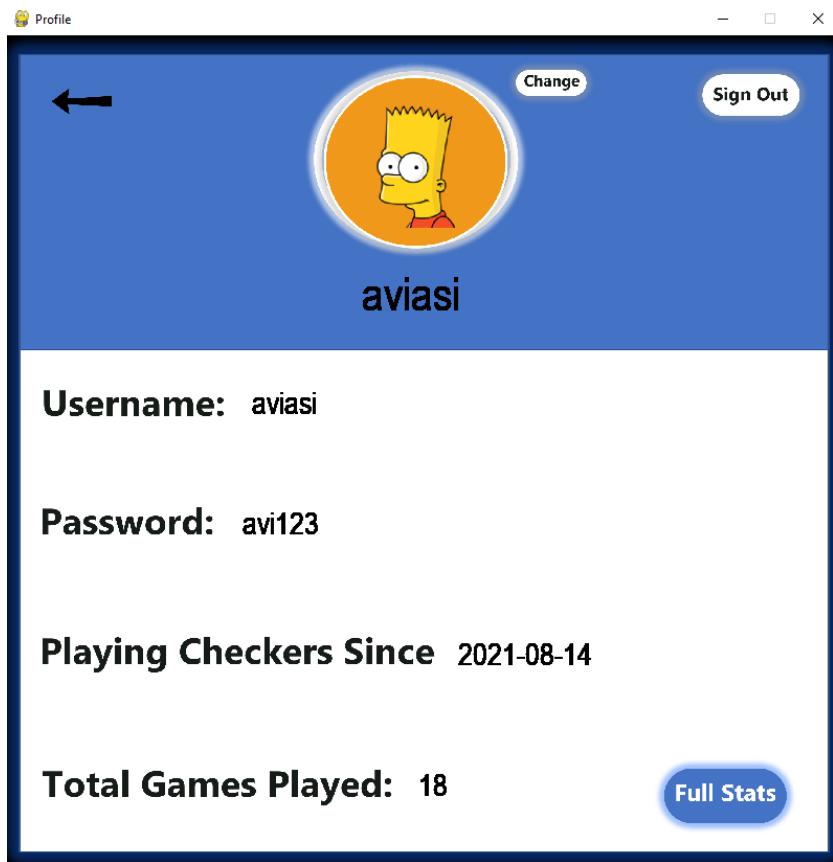
Please sign in to continue.

USERNAME: aviasi

PASSWORD: *****

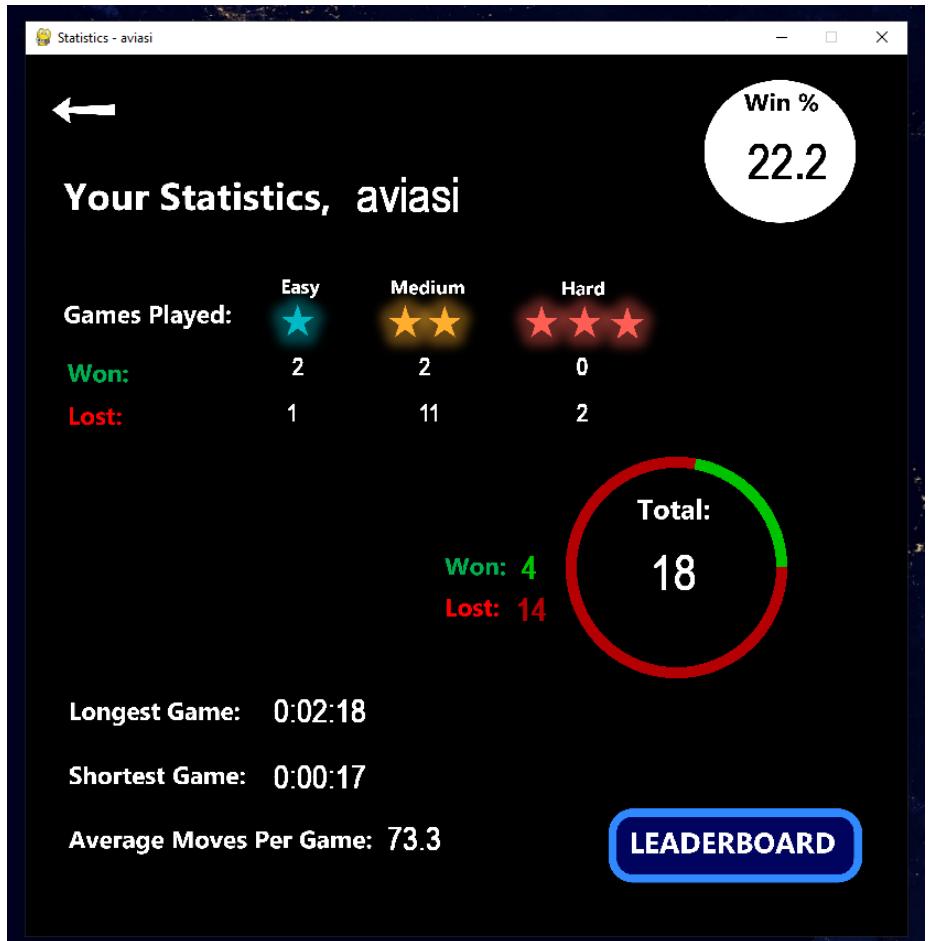
LOGIN →

Don't have an account? [Sign Up](#)

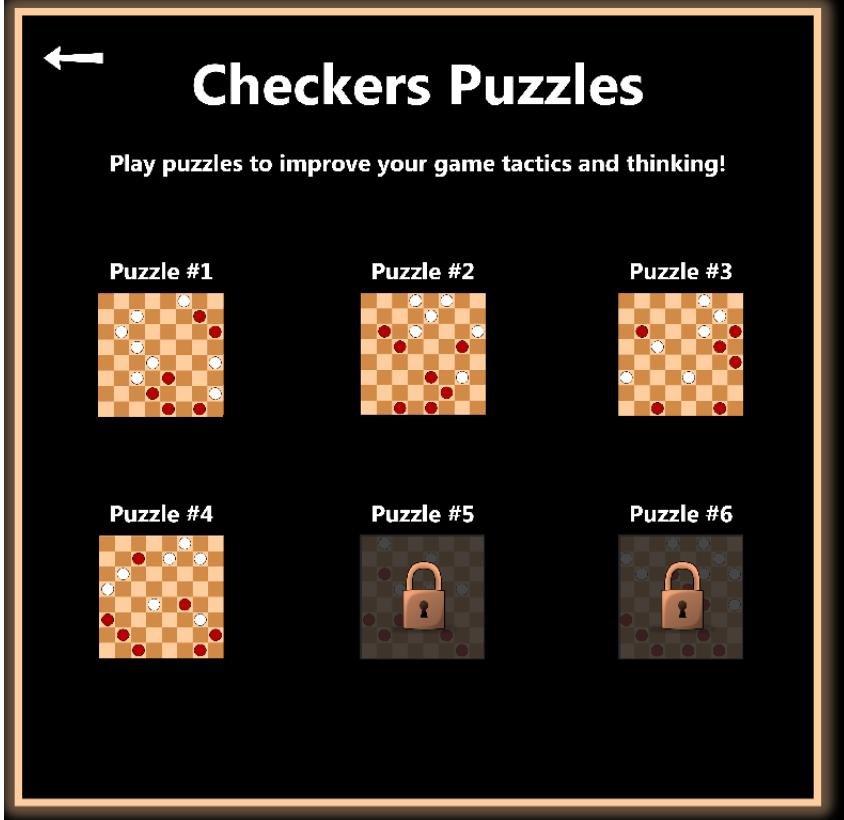


בצילום משמאל ניתן לראות את מסך ה פרופיל האישי, לאחר ביצוע ההתחברות למערכת.

במסך הנ"ל מוצגים שם המשתמש, הסיסמה, תאריך ההרשמה למערכת, מספר המשחקים הכלול ותמונה הפרופיל ניתנת לשינוי.



בצילום מימין ניתן לראות את מסך הסטטיסטיות המלאות עבור המשתמש המחבר כעט. מוצג אחוז הניצחון, סך המשחקים, הפסדים וניצחונות, גם בעבר כל רמת קושי בנפרד, המשחק הארוך והקצר ביותר ומספר המהלךים הממוצע.



בצילום משמאל ניתן לראות את מסך הפאזלים, עבור המשמש המחבר כעת; כל משתמש נמצא בשלב אחר בפתרון הפאזלים – מתחילה מהפАЗל הראשון, ולאחר השלמתו נפתח הפАЗל השני וכך הלאה.

בצילום מימין ניתן לראות את מסך פАЗל #4, בעת שהמשמש מנסה לפתור אותו.

Puzzle #4

Menu

Puzzle #4

Level: Medium

RED to play and win.

This game was played in Newcastle, 1849.

סיכום

ראשית, ניכר כי מבחינה פונקציונלית כל המערכת פועלת ומתחנגת כראוי, ללא שגיאות או בעיות: המשחק עובד חלק חן במצב שני שחקנים והוא במצב שחקן יחיד, ניתן להיכנס ו/או להירשם למערכת, לאחר משחקים ניתן לראות את הסטטיסטיות וכמו כן שאור הדברים פועלים כנדרש.

בראייה ספציפית יותר, למשל על פיתוח השחקן הממוחשב (AI), הצלחתי לפתח ולמשמש אותו בשלוש דרגות קושי שונות (קל, בינוני, קשה) בעזרת אלגוריתם המינימקס (minimax), בשילוב אלגוריתם האופטימיזציה אלפא-ביטה (Alpha-Beta Pruning), לצורך שיפור משמעותו של סיבוכיות הזמן. מידת אופן פועלת האלגוריתם והשימוש בו היו מרטקים עבורי. בין השאר, מפתה השוני מלימודי הנדסת תוכנה, שם אנו עוסקים בלמידה عمוקה (Deep Learning), ככלمر בטכניקות שונות לחלוטין בצד "לאמן" את הממוחשב (לדוגמא, לאפשר לו לשחק מספר רב של פעמים בצורה אוטומטית, ובעקבותכך "ללמוד ולהשתפר" בעצמו).

דוגמה נוספת היא שימוש מסד הנתונים ו קישורו באופן ישיר ואוטומטי לתוכנה. בעזרת התוכנה MySQL, אשר מאפשר שימוש במסדי נתונים מסוג SQL, והספרייה mysql.connector (בפייתון), Workbench יכולתי לבצע את השילוף, העדכוןים ושאר הפעולות על מסד הנתונים בהקשר לתוכנה: הרשמה ו/או כניסה למערכת (גם לאחר סגירה של התוכנה), שמירת הסטטיסטיות עברו כל שחקן בצורה ייחודית וכן שליפתן לאחר מכן, שמירת המידע האישי על כל משתמש (למשל תמונה הפרופיל שלו) וכיובי. שימוש במסד נתונים לראשונה עורר אצלם השתאות ופלה רבה. עד כה, הייתה רגילה שבתוכניות קוד כל המידע שנוצר במהלך ההרצאה נמחק מיד בסיוםה. אולם הפעם, למדתי כיצד משתמשים בכליה עצמאית, המאפשר שמירה של מידע באופן תמידי, גם לאחר סיום ההרצאה. לדוגמא, כאשר כותבים ומריצים שאלות עדכון כלשהי (למשל הוספה טבלה), ניתן למחוק לחלוטין את שורות הקוד הניל לאחר ההרצאה – הרי הפקודה כבר בוצעה ונשמרה במסד.

בראייה לאחר מכן, לא ספק נהנתי מאוד לאורך כל התהליך. חקרתי ולמדתי הרבה (הן ידע תיאורטי והן ידע מעשי), פיתחת תוכנה במשך שעوت אינסופיוות, כתבת עבודת גמר אקדמית – והכל, מתוך בחירה, בנושאים המרטקים אותי וקרובים ליibi.

ביבליוגרפיה

- בן-הרוש, י" (2020). "פיזיון מונחה עצמים 1: מחלקות, אובייקטים, תכונות וمتודות". מתוך אתר "רשתטק".
<https://reshetech.co.il/python-tutorials/object-oriented-python-classes-objects-methods-and-variables>
- בן-הרוש, י" (2020). "פיזיון מונחה עצמים 4: פולימורפיזם ושמות של متודות". 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_\(%D7%AA%D7%9B%D7%A0%D7%95%D7%AA\)](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%99%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>

- “**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=STjW3eHoCik&ab_channel=MITOpenCourseWare

נספחים

מחלקות נבחרות מן הקוד

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

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

: Game , המחלקה אשר מנהלת את המשחק

```
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 מציינת 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, 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 מציינת 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)
```