**The Grandmeizer**

A picture containing screenshot, pc game, video game software, digital compositing

Description automatically generated

מגישים:

יועז סבג – 205826001

דור בר שלום – 305786527

מנחה: דר' תמר צמח.

המכללה האקדמית תל חי.

תוכן עניינים:

תקציר ............................................................עמוד 3

מבוא...............................................................עמוד 4

* חוקים .............................................עמוד 4
* שיטת עבודה ...................................עמוד 5-6
* סביבת עבודה וחבילות ......................עמוד 6-8

מבנה האפליקציה.............................................עמוד8-15

אלגוריתמיקה...................................................עמוד 16-19

ארכיטקטורת הפרויקט.....................................עמוד 20

סיכום.............................................................עמוד 21

תודות.............................................................עמוד 22

מקורות...........................................................עמוד 22

תקציר:

עולם משחקי המחשב עבר המון שינויים בעשורים האחרונים וז'אנר "משחקי יריות מגוף ראשון" בפרט. כיום, כל כותר חדש שיוצא לשוק מסוג FPS

(First Person Shooter) בדרך כלל כולל עומדים מאחוריו עשרות\מאות מתכנתים ומעצבים שעומלים רבות על לייצר עולם תלת-ממדי עם סיפור ומשחקיות שיכבשו את הקהל.

The Grandmeizer הינו משחק FPS המצריך יכולות מצד השחקן הן טקטיות והן יכולות דיוק ומפתח יכולת קשר עין-יד. מטרת המשחק היא להישאר אחרון במפה מבין כל האויבים שמופיעים בה במיקומים רנדומליים. לאויבים במשחק נכתב אלגוריתם מיוחד אשר מחפש את השחק בצורה יעילה ברגע שנוצר בינו לבין האויב "קשר עין". ניתן לבחור בין שתי מפות שונות ובין שתי רמות קושי.

המשחק נכתב בשפת Python ונעשה שימוש רחב בתיקיית pygame על מנת לאפשר לממש את הרעיונות שרצינו.

המשחק פותח באמצעות אחת השיטות הראשונות בה השתמשו כדי להפוך מפה שתוכננה בדו-ממד אל מפה שניתן לנוע בה בצורה שתרגיש תלת-ממדית. שיטה זאת נקראת Ray Casting בה נדון בהמשך הספר, המאפשרת לנו להפוך תזוזה במפה דו-ממדית לחוויה שתרגיש למשתמש כתזוזה בעולם תלת-ממדי.

בזמן פיתוח המשחק חקרנו ולמדנו המון אודות שיטות לפיתוח משחקי FPS ואיך להסתכל על זה מהצד של המפתח ולא מהצד של המשתמש (שחקן).

**מבוא**

**חוקים**

המשחק מיועד לשחקן יחיד. בתחילת המשחק המשתמש יצטרך לבחור בין שתי מפות :

* מפה 1
* מפה 2

ובין שתי רמות קושי:

* רמה 1 (10 אויבים)
* רמה 2 (20 אויבים)

לאחר מכן, השחקן יופיע בתחילתה של המפה ביחד עם עוד 10\20 אויבים שיופיעו במקומות רנדומליים.

קיימים שלושה סוגי אויבים:

* אויב עם רובה, מבצע ירי מטווח רחוק, נהרג תוך 2 פגיעות.
* אויב בטווח קרוב, מבצע נזק רק בטווח נגיעה, נהרג תוך 3 פגיעות.
* אויב מסוג בוס, מבצע נזק מטווח רחוק בקצב מהיר, נהרג תוך 10 פגיעות.

על מנת לצלוח את המשחק בהצלחה, על השחקן להבין את חוזקותיה וחולשותיה של כל דמות, לדעת לנהל קרב יריות וחישוב סיכונים נכון ומהיר. התפרצות לא מחושבת לתוך חלל חדש ולא מוכר לרוב תוביל להיפסלות מיידית מנזק שייעשה על ידי האויבים באותו חדר.

בנוסף, על מנת לתת לשחק יתרון נוסף, בר החיים יעלה לאט ויאפשר לשחק להחלים בין קרב לקרב.

**המטרה** – לנקות את המפה מאויבים ולהישאר "האחרון שעומד".

**שיטת עבודה – XP**

Extreme Programming (או בקיצור XP) היא מתודולוגיית פיתוח תוכנה שנוצרה על ידי קנט בק. מערכות המפותחות לפיה הן גמישות מאוד לשינויים, וניתן להרחיבן בקלות ובאופן בטוח. שיטה זו דורשת הבנה טובה של עקרונות תכנות מוכוון עצמים ומשמעת עצמית גבוהה.

ארבעת ערכי היסוד של XP הם:

תקשורת - יצירת תקשורת שוטפת בין המתכנתים, הלקוח (מנחה) ומשתמש הקצה.

פשטות - יצירת קוד פשוט וכן תהליך פיתוח פשוט.

משוב - קבלת ביקורת על העבודה באופן קבוע מצד הלקוח (מנחה).

אומץ - בשל הדינמיות הרבה בה מאופיינת השיטה, יש צורך באומץ כדי לא לפחד לשנות ואף למחוק עבודה שנעשתה כבר או קוד שכבר נכתב.

לשיטת XP ישנם מנהגים / עקרונות , המסודרים בשלושה מעגלי פעולה ואנחנו עבדנו על פי חלק גדול מאוד מהם:

המעגל הפנימי:

- תכנות בזוגות - מול מחשב אחד יושבים זוג מתכנתים. אחד מחזיק במקלדת ובעכבר ומכונה 'הטייס'. הוא מבצע בפועל את עבודת התכנות. השני, המכונה 'הנווט', עוקב אחר הטייס, מייעץ לו לגבי דרך העבודה, ובעיקר אחראי על איתור תקלות בקוד (תחביריות, לוגיות וכו'). את הזוגות מחליפים מדי פעם, בשביל שכל המתכנתים יכירו כמה שיותר קוד, ואילו במבט על קוד חדש יהיה מי שיחנוך את התוכניתן שאינו מכיר את הקוד.

- עיצוב פשוט - ההנחה היא שהתוכנה הטובה ביותר היא זו שעומדת בכל הדרישות, אין בה קוד כפול, הקוד שלה מובן למתכנתים ויש בה המינימום הנדרש של מחלקות ושיטות, ולא זאת המתוחכמת ביותר, האבסטרקטית ביותר וזו המתוכננת לשנים רבות.

המעגל האמצעי:

- אינטגרציה מתמשכת - יש לבצע אינטגרציה של המערכת, הכוללת הידור וחיבור (link) לפחות פעם ביום. כך ניתן לאתר בעיות כמה שיותר מהר. אם נוצרה תקלה, המתכנת אחראי לפתור אותה בהקדם האפשרי.

- קצב הגיוני - יש לשמור על קצב פיתוח הגיוני. יש להגביל מראש את סך שעות העבודה השבועי .

- סטנדרטים בקידוד - על הצוות להסכים מראש על הסטנדרטים בקידוד, ולדאוג לאכוף אותם באופן שוטף, רציף וקבוע .

- בעלות משותפת - כל המתכנתים בצוות אחראים לכל הקוד. כל מתכנת הרואה מקום לשינוי או שיפור של הקוד, רשאי ומחוייב לעשות כן ולעדכן את השאר.

המעגל החיצוני:

- התנהלות ציוותית - הצוות כולו מאוחד במטרה ומסונכרן כל הזמן. בתחילת כל יום מתבצעת 'פגישת עמידה' באורך מוגדר מראש, בה כל שני המתכנתים נוכחים, ומספרים על התקדמותם ביום הקודם ותוכניותיהם ליום הנוכחי.

- "משחק התכנון" - המתכנתים עובדים בשיתוף פעולה מלא עם הלקוח בתכנון המערכת. הלקוח כותב סיפורים והמתכנתים מנתחים אותם ומסדרים אותם על-פי סדר עדיפויות. לאחר מכן, הסיפורים הופכים למשימות פיתוח.

- גרסאות קטנות - שחרור של גרסאות קטנות ללקוח. XP תומכת בשחרור גרסאות כל זמן מוגדר מראש, הסיבות הן: קבלת משוב מהיר, תחושת הישגיות של הצוות, הפחתת סיכונים, הגברת ביטחון הלקוח בצוות הפיתוח והתאמת התוכנה לדרישות.

**סביבת עבודה וחבילות**

Python

פייתון היא שפת תכנות עילית דינמית מהנפוצות ביותר. מציעה תחביר קריא וברור שהופך את הקוד לקריא ובר-הבנה, וכוללת מבנים המיועדים לאפשר ביטוי של תוכניות מורכבות בדרך קצרה וברורה.

הסיבות שבחרנו להשתמש בפייתון:

* תכנות מונחה עצמים - מאפשרת למפתחים לעבוד באמצעות אובייקטים, תכונות ושיטות. היכולות הללו מקלות על יצירת תוכנות מורכבות ופרויקטים גדולים יותר, ומסייעות בתחזוקה ופיתוח עתידי
* תמיכה בספריות מובנות וספריות חיצוניות- פייתון מגיעה עם מגוון רחב של ספריות מובנות שמספקות פונקציות וכלים מועילים לפיתוח תוכנה. בנוסף, ישנן ספריות חיצוניות רבות המאוחסנות בחבילות כמו NumPy, pandas ו-Django,Pygame , המספקות פונקציות מתקדמות לניתוח נתונים, פיתוח אתרים, מבני נתונים ועוד.
* נסיון מקורסים קודמים.
* רלוונטיות בתעשייה- נחשבת לשפת תכנות רחבת שימושים, משמשת למגוון תחומים בניהם פיתוח משחקים בגלל יתרונותיה.

Pygame

חבילת Pygame היא ספריית פייתון שנוצרה כדי לספק כלי לפיתוח משחקים תלת-ממדיים ודינמיים בפייתון. היא מאפשרת למפתחים ליצור משחקים, אפליקציות וסביבות וירטואליות בצורה יעילה וידידותית.  
Pygame מתבססת על ספריית SDL (Simple DirectMedia Layer), שמספקת ממשק בין התוכנה לקוד המכונה של המחשב, ומאפשרת שליטה בגרפיקה, הקלדה, עכבר וכלי ניהול זמן במשחקים.  
עם עזרת Pygame, ניתן ליצור חלונות, לטעון תמונות וקבצי סאונד, לנהל אירועים כמו לחיצת מקשים ותזוזת העכבר, וליצור דמויות, רקעים ואובייקטים תלת-ממדיים בצורה פשוטה.  
החבילה כוללת מספר מודולים ופונקציות שימושיים לפיתוח משחקים כמו פיזיקה, קול, תמונות, טקסטורות, תנועה וכו'.

לסיכום, Pygame היא חבילת פייתון חזקה וגמישה שמאפשרת לך ליצור משחקים באמצעות פייתון בצורה יעילה וידידותית.

Math

תיקיית "math" בפייתון היא תיקייה המכילה מודולים ופונקציות מתמטיות מובנות. המודול math מספק גישה לפעולות מתמטיות מתקדמות ופונקציות חישוביות בפייתון.

בתוך תיקיית math תוכל למצוא פונקציות שימושיות כמו:

- פונקציות חשבון בסיסיות כמו סכום, חיסור, כפל וחילוק.

- פונקציות מתמטיות מתקדמות כמו שורש ריבועי, חזקה, לוגריתמים וטריגונומטריה.

- פונקציות עיגוליות כמו סינוס, קוסינוס וטנגנס.

התיקייה מספקת גם קבועים מתמטיים כמו π (פי) ו-e (בסיס הלוגריתמי הנפוץ).

בקיצור, תיקיית math מספקת גישה לפונקציות וקבועים מתמטיים בפייתון, המאפשרים ביצוע פעולות מתמטיות מתקדמות.

השימוש שלנו בתיקייה זו בא לידי ביטוי בחישובים שונים על מנת לייצג תנועה במרחב המפה, חישובי וקטורים (קו יריה, הצגת האובייקטים הנמצאים בטווח הראייה של השחקן וכו').

Collections

תיקיית "collections" בפייתון מכילה מבנים נתונים מתקדמים נוספים שאינם כלולים בתוך המבנים הנתונים הבסיסיים של השפה. המבנים שנמצאים בתיקיית "collections" מספקים כלים נוספים לניהול ועיבוד נתונים בצורה יעילה וידידותית.

מבנה הנתונים שהשתמשנו בפרוייקט שלנו:

deque: מבנה נתונים שמזין תכונות של תור כפול (double-ended queue). הוא מאפשר הוספה והסרה מהשני צדדיו של התור בצורה מהירה ויעילה.

Random

המודול "random" בפייתון הוא מודול שמספק כלים ליצירת מספרים אקראיים וביצוע פעולות קשורות לאקראיות. המודול מאפשר לך ליצור מספרים אקראיים, לבצע תערובת, ולבצע פעולות כמו בחירת פריטים אקראיים מרשימות או שוברי שורות, ועוד.

השתמשנו בכלים מתיקייה זו על מנת לממש אקראיות במקרים שונים (לדוגמא: הופעת האויבים במקומות לא צפויים וידועים מראש.

Tkinter

Tkinter הוא ממשק גרפי לפייתון (GUI) המשמש ליצירת יישומי Desktop פשוטים. הוא מבוסס על ספריית Tk. מאפשר לך ליצור חלונות, כפתורים, תיבות טקסט, תמונות ועוד באמצעות קוד פייתון.

Tkinter משמש כיישום רב בפיתוח תוכנה מבוססת גרפיקה פשוטה וכמו כן נפוץ בעיצוב ממשקי משתמש (UI) לצורכי יישומים מגוונים כמו תוכנות ניהול, משחקים פשוטים ויישומים תעשייתיים.

השתמשנו בממשק זה על מנת לממש את דפי בחירת המפות ורמות הקושי לפני תחילת המשחק.

**מבנה האפליקציה**

main.py

קובץ המכיל מחלקה בשם Game אשר מאתחלת את כל המרכיבים החשובים לתחילת הרצת משחק

הפונקציות במחלקה:

- `\_\_init\_\_(self)`: פונקציה בונה המאתחלת את התנאים ההתחלתיים של המשחק ומכינה את המסך.

- ` mapSelection (self)`: פונקציה שמציגה במסך את מסך בחירת המפה ומאפשרת לשחקן לבחור מפה.

- ` difficultySelection (self)`: פונקציה שמאפשרת לשחקן לבחור דרגת קושי.

- `new\_game(self, map\_choice, difficulty\_choice)`: פונקציה שמתחילה משחק חדש עם המפה והדרגת קושי שנבחרו.

- `update(self)`: פונקציה שמעדכנת את המצב הנוכחי של המשחק וטופלת בלוגיקת המשחק.

- `draw(self)`: פונקציה שמציירת את אלמנטי המשחק על המסך.

- ` checkEvents (self)`: פונקציה שבודקת לחיצות מקלדת ואירועים במשחק כמו יציאה מהמשחק.

- `run(self)`: פונקציה שמריצה את לולאת המשחק, בה המשחק מתעדכן ונמצא בתהליך של ריצה.

מכיל את הקבצים הבאים:

- settings.py: מכיל פרמטרים כמו גודל המסך, מהירות המשחק, וכו'.

- map.py: מכיל קוד ליצירת המפה בתוך המשחק.

- player.py: מכיל את המחלקה של השחקן הראשי במשחק.

- raycasting.py: מכיל את המחלקה שמטפלת בתצוגת הריות.

- renderingEngine.py: מכיל את המחלקה שמטפלת בתצוגת אובייקטים (כמו קירות וחפצים) במשחק.

- spriteEntity.py: מכיל את המחלקה של אובייקט ספרייט.

- entityManager.py: מכיל את המחלקה שמטפלת בטיפול ובהצגת האובייקטים במשחק.

- weapon.py: מכיל את המחלקה של הנשק במשחק.

- sound.py: מכיל את המחלקה שמטפלת בקולות במשחק.

- graphNavigator.py: מכיל את המחלקה שמטפלת באלגוריתם חיפוש דרך.

map.py

המחלקה Map מייצגת את המפה במשחק שבו השחקן ינווט.

בנאי המחלקה מקבל את מופע המשחק (game) ואת מספר המפה (map\_num). היא משתמשת במספר המפה כדי לבחור את המפה המתאימה מבין mini\_map1 ו-mini\_map2.

הפונקציות במחלקה:

-`get\_map()` מתקבלת על מפה ומייצרת את המפה המלאה באמצעות העתקת הערכים מהמיני מפה למפה העולמית. הפונקציה עוברת על כל מיקום במיני מפה ואם הערך שלו אינו ריק (שווה ל- False), היא מוסיפה את המיקום והערך שלו למפה העולמית.

-`draw()` מציירת את המפה על המסך. היא עוברת על כל מיקום במפה העולמית ומציירת מלבן בגודל 100x100 פיקסלים בצבע חום בהיר ועם מסגרת. המיקום של המלבן מתבצע בהתאם למיקום של התא המתאים במפה.

npc.py

המחלקה `NPC` היא תת-מחלקה של `AnimatedSprite` ומייצגת דמות המשוחקת על ידי המחשב (Non-Player Character, NPC) במשחק. היא מכילה פונקציות שמתאימות להתנהלות והתנהגות הדמות במשחק.

הפונקציות במחלקה:

- `\_\_init\_\_(…)`: פונקציה בנאי שמאתחלת את המחלקה. היא מקבלת מספר פרמטרים, כולל הפרמטרים הנדרשים על ידי הפונקציה הבנאית של `AnimatedSprite`. היא מגדירה מאפיינים כגון מרחק התקיפה, מהירות, גודל, בריאות, נפילת נזק, דיוק ועוד.

- `update(self)`: פונקציה שמעדכנת את מצב הדמות. היא בודקת את זמן האנימציה, מקבלת את הדמות הנוכחית ומפעילה את הלוגיקה המתאימה.

- ` checkWall (self, x, y)`: בודקת אם מיקום נתון (x, y) הוא קיר בהתאם למפת העולם של המשחק.

- ` checkWallCollision (self, dx, dy)`: בודקת התנגדות בקירות ומתאימה את המיקום של הדמות בהתאם.

- `movement(self)`: מחשבת את המיקום הבא של הדמות באמצעות אלגוריתם למציאת נתיב ומזיזה את הדמות לעבר המיקום הזה.

- `attack(self)`: מפעילה פעולת התקפה עבור הדמות. משמיעה תפקיד קולי ומחשבת האם התקפה פוגעת בשחקן על פי הדיוק של הדמות. אם התקפה מוצלחת, גורמת נזק לשחקן.

- ` animateDeath (self)`: מפעילה את סדרת האנימציה של הדמות במצב מוות.

- `animatePain(self)`: מפעילה את סדרת האנימציה של הדמות במצב כאב.

- `check\_hit\_in\_npc(self)`: בודקת האם יריית השחקן מתקשרת עם הדמות של ה- NPC. משמיעה תפקיד קולי ומעדכנת את בריאות הדמות ומצב הכאב שלה.

- `checkHealth(self)`: בודקת האם בריאות הדמות הגיעה לאפס. מגדירה את הדמות כ"לא חיה" ומשמיעה תפקיד קולי של מוות.

- `runLogic(self)`: מבצעת את הלוגיקה העיקרית להתנהגות הדמות. בודקת אינטראקציה שחקן-דמות, כמו זיהוי השחקן באמצעות שיגור קרן. מפעילה אנימציה בהתאם למצב הדמות ובודקת את הפעולות המתאימות.

- `ray\_cast\_player\_npc(self)`: מבצעת חישה על ידי קרנות כדי לבדוק אם יש קו ראייה בין ה- NPC לשחקן. מגדירה את המרחקים לשחקן ולקירות בשני הכיוונים (אופקית ואנכית).

- `draw\_ray\_cast(self)`: מציירת סמלים חזותיים של תהליך קרנות החישה. מציירת מעגל במיקום ה- NPC וקו למיקום השחקן אם יש קו ראייה.

בקובץ, יש שלוש מחלקות נוספות בסוף הקוד:

1. ` RangeNPC `: מחלקה זו מייצגת דמות חייל ירייה במשחק. היא יורשת מהמחלקה `NPC` ומתאימה לדמויות שיכולות לירות מרחוק. היא מגדירה ערכים מותאמים אישית עבור מאפייני ההתנהגות של החייל, כמו טווח התקפה, נפילת בריאות.

2. ` MeleeNPC `: מחלקה זו מייצגת דמות קאקו דמון במשחק. היא יורשת מהמחלקה `NPC` ומייצגת דמות שתוקפת מקרוב בלבד. היא מגדירה ערכים מותאמים אישית עבור מאפייני ההתנהגות של הקאקו דמון, כמו טווח התקפה ונפילת בריאות.

3. ` BossNPC `: מחלקה זו מייצגת דמות סייבר דמון במשחק. היא יורשת מהמחלקה `NPC` ומתאימה לדמויות בוס מיוחדות. היא מגדירה ערכים מותאמים אישית עבור מאפייני ההתנהגות של הסייבר דמון, כמו טווח התקפה ונפילת בריאות.

כל אחת מהמחלקות הנוספות מתאימה לסוג ספציפי של דמות במשחק ומגדירה את התכונות הייחודיות שלה בהתאם. זאת כוללת את האנימציות המתאימות, המרחק והמהירות שבהן הדמות תתקרב או תתקוף את השחקן, נפילת הבריאות בקרבה לדמות ודיוק התקפה. המחלקות משמשות ליצירת פונקציות ותכונות מותאמות אישית עבור סוגי הדמויות השונים במשחק, מה שמאפשר לכל דמות להתנהג בדרך ייחודית ולהיות חלק מתהליך המשחק.

entityManager.py

המחלקה ` entityManager ` מנהלת את היצירה, העדכון והאינטראקציה של ספרייטים ו-NPCs (דמויות שהן לא השחקן) במשחק.

הפונקציות במחלקה:

1. `\_\_init\_\_(self, game, difficulty\_choice)`: פונקציה זו מאתחלת את האובייקט עם המשחק ורמת הקושי שנבחרו. היא מגדירה את נתיבי הספרייטים השונים, מאתחלת רשימות ספרייטים ו-NPCs, ומפעילה את פונקציית `spawn\_npc()` על פי הרמת הקושי הנבחרת כדי ליצור NPCים במשחק.

2. `spawn\_npc(self)`: פונקציה זו מפעילה את הייצור של NPCים בעולם המשחק על פי רמת הקושי הנבחרת. היא בוחרת סוגי NPCים באופן אקראי עם הסתברויות משוקללות ומבצעת השמה למיקומים אקראיים במפת המשחק, תוך תחושת אזורים מוגבלים.

3. `check\_win(self)`: פונקציה זו בודקת אם נגמרו כל ה-NPCים במשחק ומכריזה על מצב ניצחון. במקרה של ניצחון, היא מפעילה אנימציה ואז מפעילה מחדש את המשחק עם הגדרות המפה ורמת הקושי הנבחרות.

4. `update(self)`: פונקציה זו מעדכנת את מיקומי ה--NPCים, האנימציות שלהם ובודקת תנאי ניצחון. היא מקצה מחדש את המיקומים של ה-NPC-ים והספרייטים, מעדכנת את האנימציות שלהם ומבצעת בדיקה לגבי מצב הניצחון.

5. `add\_npc(self, npc)`: פונקציה זו מוסיפה NPC לרשימת ה-NPCים במשחק.

6. `add\_sprite(self, sprite)`: פונקציה זו מוסיפה ספרייט לרשימת הספרייטים במשחק.

מה שקורה בעיקרון הוא שבתהליך האתחול (`\_\_init\_\_`), מוגדרים הנתיבים לתמונות הספרייטים השונים ומבוצע יצירה של NPC-ים בהתאם לרמת הקושי שנבחרה. לאחר מכן, בפונקציה `update`, המיקומים והאנימציות של ה-NPC-ים והספרייטים מתעדכנים בכל צירוף. יש גם בדיקה לגבי הגעת המשחק למצב ניצחון.

renderingEngine.py

המחלקה ` RenderingEngine ` אחראית להצגת האובייקטים במשחק על המסך ולניהול תצוגת האלמנטים השונים כמו רקעים, בריאות השחקן, מסכי ניצחון או הפסד, וטקסטורות.

הפונקציות במחלקה:

1. `\_\_init\_\_(self, game)`: פונקציה זו מאתחלת את האובייקט עם המשחק. היא מגדירה את המשתנים הדרושים כמו מסך המשחק (`screen`), טקסטורות הקירות, טקסטורת השמים, טקסטורת הדם, תמונות המספרים שמייצגות בריאות שחקן, ותמונות המסכים לניצחון ולהפסד.

2. `draw(self)`: פונקציה זו מציירת את האובייקטים במשחק על המסך. היא משתמשת בפונקציות נוספות כמו `draw\_background()` לציור הרקע, `render\_game\_objects()` לציור האובייקטים במשחק, ו-`draw\_player\_health()` לציור בריאות השחקן.

3. `win(self)`: פונקציה זו מציגה מסך ניצחון על המסך.

4. `game\_over(self)`: פונקציה זו מציגה מסך הפסד על המסך.

5. `draw\_player\_health(self)`: פונקציה זו מציירת את בריאות השחקן על המסך באמצעות תמונות של ספרות.

6. `player\_damage(self)`: פונקציה זו מציגה מסך דם על המסך כאשר השחקן סופג נזק.

7. `draw\_background(self)`: פונקציה זו מציירת את רקע המשחק כולל השמים והרצפה.

8. `render\_game\_objects(self)`: פונקציה זו מציירת את האובייקטים במשחק על המסך בהתאם לעומקם.

9. `get\_texture(self, path, res)`: פונקציה זו מטעינה תמונת טקסטורה מהנתיב שסופק ומשנה את גודלה לגודל שנספק כארגומנט

10. `load\_wall\_textures(self)`: פונקציה זו טוענת את טקסטורות הקירות מהמסלולים המתאימים.

graphNavigator.py

המחלקה `GraphNavigator` מטפלת בפעולות חיפוש נתיב במשחק. היא משתמשת באלגוריתם חיפוש רחב ראשוני (BFS) כדי למצוא את הנתיב הקצר ביותר ממיקום התחלה למיקום יעד על מפה.

הפונקציות במחלקה:

- `\_\_init\_\_(self, game)`: מאתחל את אובייקט ה-`GraphNavigator` על ידי קבלת מופע המשחק כפרמטר. הפונקציה משיגה את המיני-מפה מהמפה של המשחק ומגדירה את כיווני התנועה האפשריים (דרכים) כתלות בגדודים בכיווני ה-X וה-Y.

- `get\_path(self, start, goal)`: מחשבת ומחזירה את הנתיב הקצר ביותר ממיקום התחלה למיקום יעד באמצעות אלגוריתם החיפוש הרחב הראשוני. הפונקציה משתמשת במתודת `bfs()` לביצוע החיפוש עצמו ומבנה את הנתיב על ידי הליכה אחורה מהמיקום היעד למיקום התחלה באמצעות המילון `visited`.

- `bfs(self, start, goal, graph)`: מבצעת את אלגוריתם החיפוש הרחב הראשוני כדי למצוא את הנתיב הקצר ביותר מהתחלה ליעד. הפונקציה משתמשת בתור תור עבור עיבוד הצמתים בצורה רחבה ומוחקת צמתים שכבר נבדקו מהמילון `visited`. היא משתמשת גם בתנאי לעצירת החיפוש כאשר מגיעים לצומת היעד ומחליפה את המצב בשדה `visited` כאשר מצאה צומת חדש שניתן להוסיף לתור.

- `get\_next\_nodes(self, x, y)`: מחשבת ומחזירה רשימת צמתים שסמוכים (צמתים המתקרבים ביותר) למיקום נתון (x, y). הפונקציה מתייחסת לכיווני התנועה האפשריים (דרכים) ובודקת אם הצמתים הסמוכים שמתקבלים אינם מכשולים במפה.

- `get\_graph(self)`: בונה את התיאור הגרפי של המפה על ידי מעבר על כל מיקום (x, y) במיני-מפה. עבור כל מיקום ריק, הפונקציה מוסיפה את הצמתים הסמוכים שלו (צמתים המתקרבים ביותר) לגרף באמצעות מתודת `get\_next\_nodes`.

raycasting.py

המחלקה `RayCasting` מתמודדת עם אלגוריתם ייצוג התמונה במשחק באמצעות הטיה של קרניים (ray casting). המחלקה אחראית לאחסון תוצאות הטיה של הקרניים, כולל עומק, גובה מרובעת, טקסטורה והיסטות.

הפונקציות במחלקה:

- `\_\_init\_\_(self, game)`: מאתחל את אובייקט ה-`RayCasting` על ידי קבלת אובייקט המשחק כפרמטר. הפונקציה מאתחלת רשימה ריקה לתוצאות הטיה של הקרנים ומקובצת את העצמים שיש להם צורך להיות עבור ההטיה.

- `get\_objects\_to\_render(self)`: מכינה את העצמים להצגה על פי תוצאות הטיה של הקרנים. הפונקציה יוצרת רשימה ריקה שתשמש לאיחסון העצמים. לכל קרן, היא מקבלת את העומק, הגובה המרובעת, הטקסטורה וההיסט שקובעים תוצאות הטיה של הקרן, ויוצרת את העצם המתאים שישמש בהצגתו על המסך. העצם כולל את העומק, הטקסטורה ומיקומו של העצם במסגרת המסך.

- `ray\_cast(self)`: מבצעת את הטיה האמיתית על ידי חישוב נקודות החיתוך עם הקירות וקביעת הטקסטורות והעומקים המתאימים. הפונקציה יוצרת רשימה ריקה לתוצאות הטיה ולכל קרן מחשבת את הנקודות של החיתוך, הטקסטורות המתאימות, העומקים וההיסטים. תוצאות הטיה מתווספות לרשימה.

- `update(self)`: מפעילה את הטיה האמיתית ומכין את העצמים להצגה על פי התוצאות.

spriteEntity.py

המחלקה `SpriteObject` מייצגת אובייקט ספרייט במשחק. היא מטפלת בהקרנת הספרייט ובחישוב ההיסט והגובה המתאימים עבור ההצגה על המסך.

הפונקציות במחלקה:

- `\_\_init\_\_(…)`: פונקציה בונה שמאתחלת את אובייקט ה-`SpriteObject` ומגדירה את התכונות הראשיות שלו כמו משחק, שחקן, מיקום, גודל, ותמונה.

- `get\_sprite\_projection(self)`: פונקציה המחשבת את ההיסט והגובה המתאימים עבור ההצגה של הספרייט על המסך. היא מתאימה את גודל התמונה לפי ההיסט והגובה, ומגדירה את מיקומה על המסך.

- `getSprite(self)`: פונקציה המבצעת את החישובים הדרושים עבור ההיסט והגובה של הספרייט, כולל חישוב הזווית והמרחק מהשחקן, ומגדירה את מיקומה על המסך.

- `update(self)`: פונקציה המפעילה את הפונקציות הנדרשות בכל מעגל עדכון, כולל חישוב ההיסט והגובה והצגת הספרייט על המסך.

המחלקה `spriteAnimator` הנמצאת גם היא באותו קובץ, היא מחלקה יורשת מהמחלקה `SpriteObject` ומתאימה לספרייטים אנימציות. היא מכילה תכונות ופונקציות נוספות שמתייחסות לאנימציה של הספרייט.

המחלקה מכילה את הפונקציות הבאות:

- `\_\_init\_\_(…)`: פונקציה בונה שמאתחלת את אובייקט ה-`spriteAnimator` ומגדירה את התכונות הראשיות שלו, כולל נתיב התמונות של האנימציה, מיקום הספרייט, גודל, וזמן האנימציה.

- `update(self)`: פונקציה המפעילה את הפונקציות הנדרשות בכל מעגל עידכון, כולל חישוב ההיסט והגובה, בדיקת הפעלת האנימציה, וביצוע האנימציה על ידי חילוף תמונות הספרייט.

weapon.py

תת-מחלקה של "spriteAnimator" ומייצגת אובייקט אמצעי נשק במשחק.

הפונקציות במחלקה:

\_\_init\_\_(): פונקציה בונה המאתחלת את אובייקט Weapon. היא מקבלת את המשחק (game) כפרמטר ופרמטרים אופציונליים נוספים כמו נתיב התמונה של הנשק, גודל התמונה, זמן האנימציה ועוד. היא מבצעת גם את פונקציית הבנאי של המחלקה האב (AnimatedSprite).

animate\_shot(self): פונקציה שמטפלת באנימציה של הנשק כאשר השחקן יורה (לחיצה על העכבר השמאלי). היא בודקת אם הנשק נמצא בתהליך טעינה ואם כן, היא מבצעת את האנימציה המתאימה על ידי סיבוב התמונות של הנשק בתוך הdeque של התמונות. גם הספירה של המסגרות (frame\_counter) מתעדכנת ואם היא מגיעה למספר התמונות, הדגל של הטעינה מוחזר לערך הקודם והספירה מתאפסת.

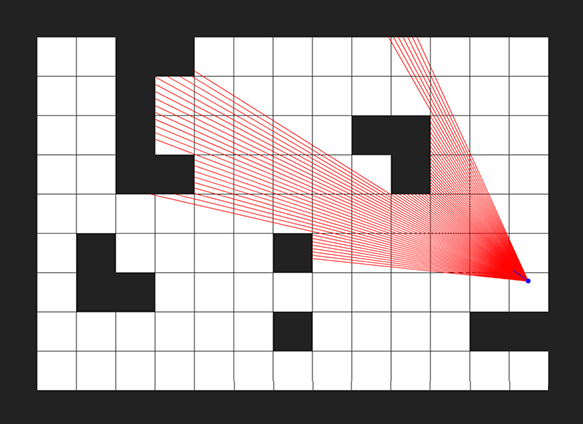
draw(self): פונקציה שמציירת את התמונה הנוכחית של הנשק על מסך המשחק במיקום המתאים.

update(self): פונקציה שמתעדכנת את מצב האנימציה של הנשק. היא בודקת את זמן האנימציה וסופרת אותו. גם פה נבדק אם הנשק נמצא בתהליך טעינה על ידי קריאה לפונקציה animate\_shot().

**אלגוריתמיקה**

**בעיה:** יצירת חוויה תלת ממדית עבור המשתמש

**פתרון:** Raycasting – אחת מהשיטות הראשונות שבאמצעותן פותחו משחקי FPS בעבר. הופכת מפה דו-ממדית למשחק עם חווית משחק תלת ממדית.

**הסבר השיטה:** שיגור של קרניים (וקטורים) מתוך נקודת המבט של השחקן אל כל רוחב שדה ראייתו (רוחב המסך).   
במשחק שלנו, אנו משגרים 800 קרניים   
כאשר רוחב המסך שלנו הינו 1600,   
וכל קרן "פוגעת" בסופו של דבר באובייקט  
מסוים (קיר או אויב) אשר יוצג במרחק   
ובזווית המתאימה.

כל קרן מתנהגת כפונקציית קו ישר במחרב.  
על מנת להציג את זווית, עומק   
ומרחק האובייקטים בצורה הנכונה,  
המחלקה מבצעת את המתודה הבאה:

A screen shot of a computer program

Description automatically generated with medium confidenceA screenshot of a computer program

Description automatically generated with medium confidence

**בעיה:** איך לגרום לאויבים לחפש אחרי השחקן בצורה הגיונית שתרגיש טבעית ואנושית כמה שאפשר. תנועת האויבים תתחיל מהרגע שנוצר "קשר עין" בינם לבין השחקן (מנגנון ה- raycasting שהוזכר לעיל משמש גם את האויבים על מנת לקבוע מתי השחקן נראה לעין עבורם).

**פתרון:** נתייחס למערך הדו-ממדי שמייצג את המפה כאל גרף ונפעיל חיפוש BFS ממיקום האויב אל מיקום השחקן.

**הסבר השיטה:** אלגוריתם BFS (חיפוש לרוחב בגרף) סורק את הגרף בצורה שמבטיחה שכל צומת שנמצא באותו רכיב קשירות של הצומת ההתחלתי ייבדק, וסריקה זו נעשית בזמן אופטימלי, הליניארי למספר הקשתות והצמתים בגרף.  
במשחק, הוגדר משתנה דגל שהופך ל-true אך ורק כאשר האויב "מזהה" את השחקן על ידי שיטת raycasting, רק אז, מופעל אלגוריתם BFS עבור אותו אויב שמתחיל תנועה לכיוון השחקן.  
גם כאשר השחקן מנסה לברוח ולהתחבא, האויב מחפש אחריו בדרך הקצרה ביותר בצורה בה סביר להניח אויב במציאות היה מחפש אחר אויבו מהרגע שנגלה אליו.

מצורפת תמונה מהקוד שמראה את המתודות החשובות במחלקה GraphNavigator שאחראית למימוש אלגוריתם החיפוש בגרף (מפה):

A screenshot of a computer program

Description automatically generated with medium confidence

**בעיה:** איך לנהל את לוגיקת האויבים על מנת שתוצג לנו האנימציה (סדרת תמונות – sprite sheet) המתאימה והנכונה בהתאם למה שקורה במשחק.

**פתרון:** נכתבה מתודת runLogic במחלקת Npc המבצעת סדרת התניות ופועלת בהתאם לסיטואציה.

A screenshot of a computer program

Description automatically generated with medium confidence

הסבר השיטה: **המתודה שואלת מספר שאילתות על דגלים בוליאנים לדוגמא:**

**אם isAlive = true – כלומר, האויב חי ויש להיכנס לסקופ ולהמשיך בדיקות כמתוכנן, אחרת, תוצג לנו גופתו של אותו אויב.**

**- אם isPain = true – כלומר, האויב נפגע מירייה של השחקן ויש להציג את האנימציה שבה אותו אויב חווה כאב פיזי, אחרת, נדלג הלאה ואנימציית הכאב לא תוצג.**

**בנוסף, מתבצעת גם השאילתה הבאה: האם מרחק האויב מן השחקן קטן מטווח התקיפה שלו? אם כן, תוצג לנו אנימציית ההתקפה וכן השחקן יושפע בהתאם.  
אחרת, האויב ממשיך כרגיל בשלו מבלי לתקוף.  
יש לשים לב שהתקפה לא יכולה להתבצע אם יש קיר מפריד מכיוון שאין לאויב "קשר עין" עם השחקן ולכן אינו יודע את מרחקו.**

**בעיה:** איך לאפשר למשתמש לבחור בין מפות שונות ובין רמות קושי שונות.

**פתרון:** נכתבו מתודות ייעודיות במחלקת Game האחראיות להצגת דף הפתיחה בו תחילה נבחר את המפה ולאחר מכן את רמת הקושי.

**הסבר השיטה:** בשלב איתחול המשחק, לפני שיוצאים לדרך, מחלקת Game מפעילה את המתודות הבאות בנוסף לסדר האתחול הרגיל:  
- mapSelection – בעזרת רכיבי GUI , המתודה מציגה לנו כפתורים אשר ניתן לנווט בניהם עם החצים ולבחור עם לחיצה על ENTER.  
בסופה, מחכה לנו שדה במחלקת Game שמעיד על בחירת המשתמש.

- difficultySelection – פועלת בדומה למתודה הקודמת לה, בסופה יש לנו שדה במחלקת Game שמעיד על בחירת רמת הקושי. לבסוף, קוראת למתודת new\_game עם הפרמטרים שנבחרו.

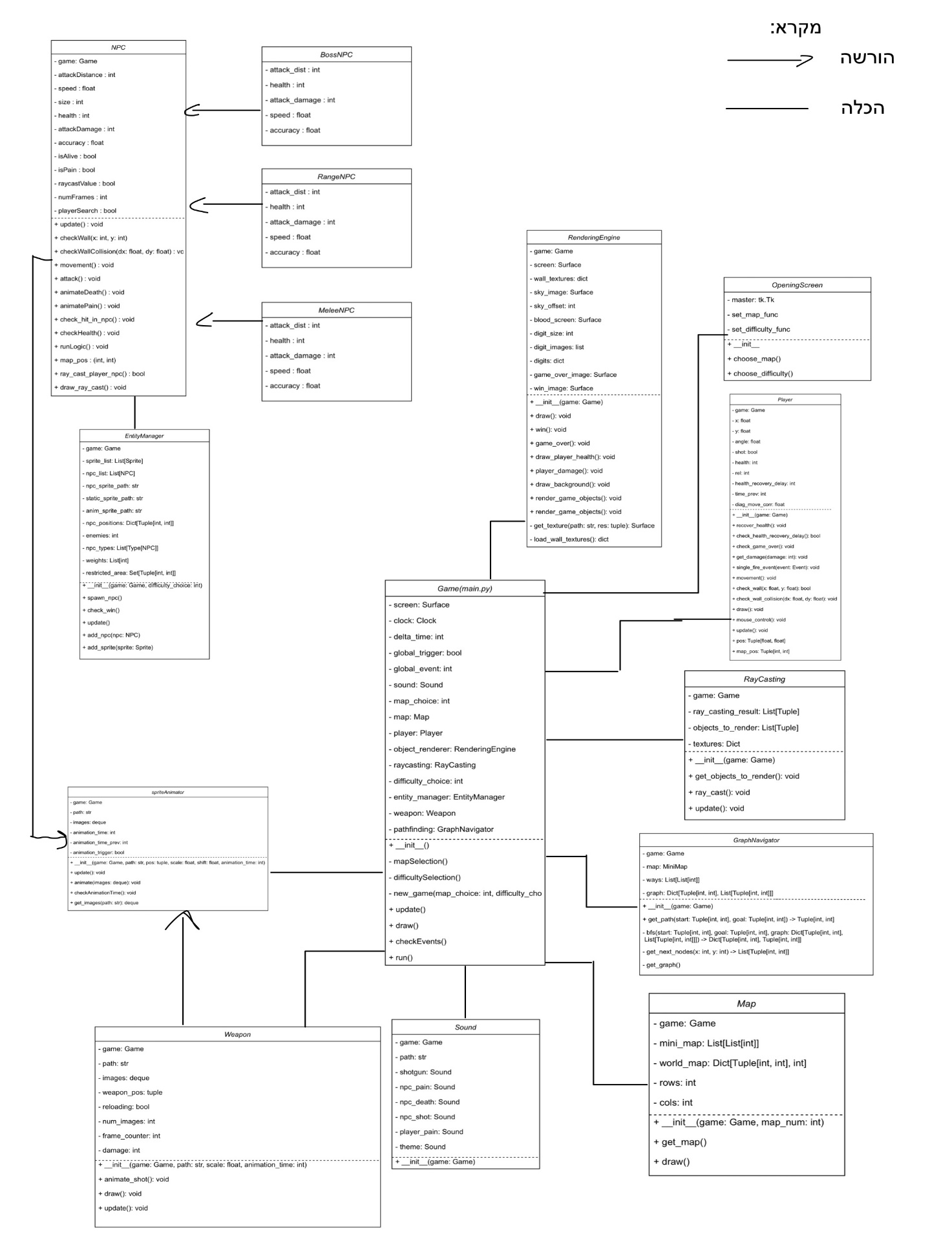
**A screenshot of a video game

Description automatically generated with low confidence**

**A screenshot of a computer screen

Description automatically generated with medium confidence**

**ארכיטקטורת הפרויקט**

**UML המחלקות**

**סיכום**

**התרשמות מתהליך העבודה**

לאחר שהוחלט על נושא הפרויקט, השפה python נבחרה על מנת שצוות המתכנתים יתעסק פחות בבעיות syntax ויותר באלגוריתמיקה הדרושה מכיוון ששפה זו מוכרת ויש לצוות ניסיון בה. כנזכר לעיל, אופי העבודה על הפרויקט היה במפגשים משותפים ותכנות בשיטת "טייס-נווט" (כמוזכרת בשיטת XP).  
בחירת נושא הפרויקט נבעה מתוך תשוקה וסקרנות של הצוות בפיתוח של משחקים ויצירת מעין "עולם וירטואלי" בו השחקן יכול לנוע בחופשיות ולהיכנס לנעלי הדמות. תהליך פיתוח הפרויקט היה מאתגר וכלל לא מעט מכשולים שהצריכו מחקר מעמיק ולמידה של הנושאים הרלוונטיים על מנת ליישם אותם בפרויקט.  
מעבר לכתיבת הקוד, כתיבת ספר הפרויקט והכנת הפרזנטציה הביאו איתם אתגרים אחרים שלימדו ופיתחו את יכולות הצוות להצגת פרויקטים בעתיד.  
לסיכום, כל התהליך היה מלמד והקנה לצוות הפיתוח כלים שיעזרו מאוד באתגרים עתידיים במקצוע.

**ניתן לשיפור**

הפרויקט פותח במסגרת שנת לימודים ומגבלות זמן מסויימות.  
צוות הפיתוח העלה רעיונות אפשריים שבהינתן הזמן והאפשרות היו ניתנים למימוש:

- הוספת אפשרות לשחק ביחד עם עוד שחקן מרוחק בשרת המאפשר משחק רב-משתתפים.

- הוספת מפות .

-הוספת עוד סוגי NPC's.

- הוספת נשקים.

- הוספת רמת קושי נוספת.

-הוספת שלבים ובניית עלילה רחבה יותר.

- הוספת תאימות לבקרי שליטה שונים (כגון שלט, מסך מגע וכ'ו...)

לסיכום, דבריהם של צוות הפיתוח :"למרות שתיכננו להוסיף עוד המון פיצ'רים שונים למשחק, אנחנו מרוצים מהתוצר הסופי והצלחנו לכלול לא מעט דברים שתיכננו מראשית הפרויקט".

**תודות**

דר' תמר צמח – תודה על הנחיה ופידבק שעזרו לנו לפתח פרויקט זה כראוי.

**רשימת מקורות**

<https://lodev.org/cgtutor/raycasting.html>

<https://github.com/vinibiavatti1/RayCastingTutorial>

<https://pythonprogramming.altervista.org/raycasting-with-pygame/?doing_wp_cron=1686397271.2182509899139404296875>

<https://killerrobotics.me/2021/08/13/raycasting-game-in-python-and-pygame-part-1/comment-page-1/>  
קורס אלגוריתמים מודל תל חי