

תיק פרויקט - Liad Koren



תיקון הרצוג

ליעד קורן

215671066

מורה אופיר שביט

הגנת סייבר

תאריך – 7/11/23

תוכן עניינים

<u>3</u>	<u>מבוא</u>	•
<u>6</u>	<u>תיאור יכולות</u>	•
<u>10</u>	<u>לוח זמנים</u>	•
<u>11</u>	<u>תיאור תחום ידע</u>	•
<u>14</u>	<u>מבנה ופירוט תקשורת</u>	•
<u>19</u>	<u>סביבת פיתוח</u>	•
<u>19</u>	<u>פרוטוקול תקשורת לעומק</u>	•
<u>24</u>	<u>תיאור שכבת האפליקציה</u>	•
<u>25</u>	<u>מדריך למשתמש</u>	•
<u>29</u>	<u>ימוש הפרויקט</u>	•
<u>38</u>	<u>רפלקציה</u>	•
<u>39</u>	<u>ביבליוגרפיה</u>	•
<u>39</u>	<u>נספחים</u>	•

מבוא

יזום - תיאור המערכת:

פרויקט "Raycaster תלת-מימדי" הוא פרויקט מרסים שמתמקד בעולם התכונות והגרפיקה התלת-מימדית. הפרויקט מיועד להריץ על מחשב, ויציג תצוגה מריהיבה ומרשימה בעולם התלת-מימד. בנוסף, אפשר התנסות במשחק אונליין בלבד עם חברים מחשבים אחרים. המשחק המשמש יהיה יידוטי ואפשר לשחק בקלות, באמצעות המקלדת והעכבר על מחשב. בסיכום, הפרויקט מתמקד בנושאים מתקדמים כמו משחק משתמש גרפי שנכתב בשפת C++ באמצעות ספריית sfml, תקשורת בראשת עברו משחק אונליין, וחקר עצמי בתחוםים שונים כגון תקשורת, C++, ואלגוריתמים מיוחדים שמאפשרים את הגרפיקה המרשימה.

堃ירת פתרונות קיימים:

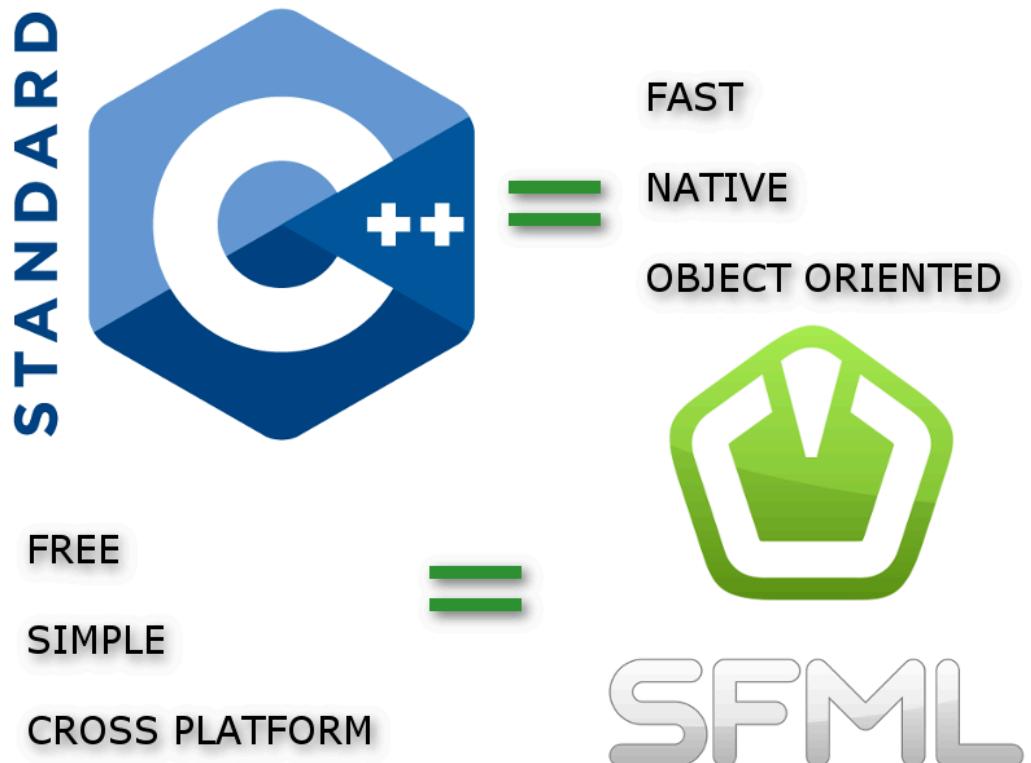
הפרויקט אינו בא לענות על שום בעיה קיימת אלא ליצור משחק מהנה, ישנו משחק דומה בשםWolfenstein 3D או Doom או wolfenstein 3d אשר שוכן באותו שם אונליין.

המשחק משנות ה-90 בשם Wolfenstein 3d



תחומי הפרויקט:

הפרויקט עוסק במגוון תחומיים שונים בינהם:
 גרפייקה – התוכנה משתמש בספרית sfml על מנת להציג את הגרפיקה.
 רשות – התוכנה תכיל צד שרת וצד לקוח שיתקשרו זה עם זה על מנת שהפרויקט יפעל.
 אבטחה והצפנה – כל ההודעות המועברות בין צד שרת לצד לקוח. המידע על המשחק עצמו מועבר באמצעות פרוטוקול מיוחד שהמצאת, ומוצפן באמצעות AES 16 byte.



```

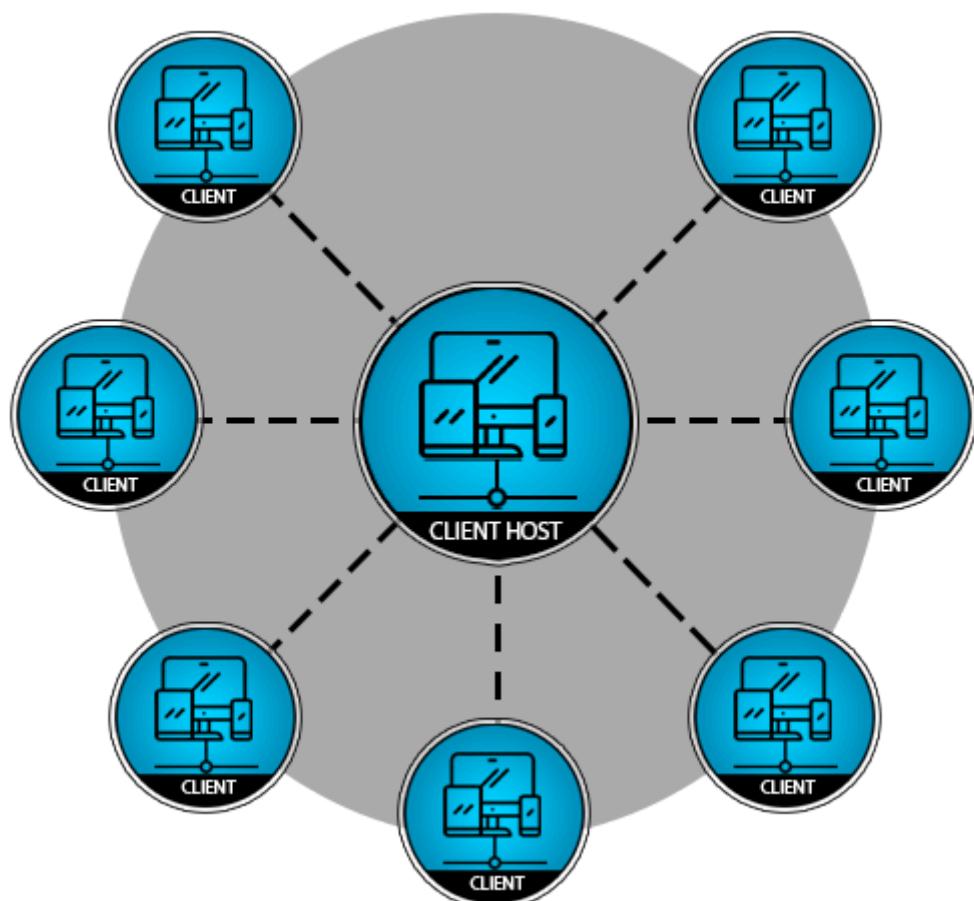
3
4
5     #include <SFML/Graphics.hpp>
6     #include <SFML/Network.hpp>
7
8
  
```

תיאור מפורט יותר של המערכת:

עד לקו - המשתמש יפעיל תוכנת לקוח דרכה יוכל את האפשרות להירשם או להתחבר עם משתמש קיימ. במידה וירצה להירשם יצרר לרשום שם משתמש וסיסמה ולענות על שאלת אישית שתעזר לשחרר את הסיסמה במקרה של שכחה. לאחר ההתחברות למשתמש יופיע דף המציג את

כל החדרים ותינן למשתמש בחירה בין יצירת משחק ל怛טרופות למשחק קיימ. במידה ויחלטו ליצור משחק יוכל לבחור את מספר השיבובים ומספר השניות לכל סיבוב. במהלך המשחק במידה וטורך המשתמש יציר צייר (הלקוק) ישלח לשרת את הנקודות אותן ציר על לוח המשחק (, במידה ולא תורך לציר יש צ'אט של משתמשים האחרים שמנסים לנחש את המילה המסתתרת (הלקוק ישלח לשרת את הנחושים והשרת יבודק האם הנחושים נכונים). בסוף כל תור המשתמש יקבל ניקוד על פי זמן הניחס ובתום כל השיבובים השחקן עם מספר הנקודות הרב ביותר יוכתר כמנצח.

צד שרת – לכל לקוח המתחבר לשרת יופעל תהליך שיטפל בתקשורת בין השירות על מנת ליצור מעין מקבילות. תחילת השירות ייכה לקלטת שם משתמש וסיסמה על מנת להתחבר, לאחר שיקבל קלט מהלקוק השירות יאמת אותו מול מאגר המידע על מנת לבדוק שאכן הוא קיימ. לאחר מכן השירות יענה על בקשות של יצירה או חיבור למשחק קיימ, כאשר המשחק מתחילה יורם תהליך



שינהל את החדר הספציפי והתהליכיונים המתחברים עם הلكוחות יתקשרו עם התהליין על מנת להעביר הודעה.

הגדרת לקוחות:

הלקוח שאלוי פונה המשחק יכול להיות כל אחד שמחפש חוות יחודית ומרהייבת, כמו גם אלו המעניינים להתחבר ולשחק בזוג או יותר חברים מחשבים שונים באמצעות הרשת. המערכת פותחה ונועדה למגוון רחב של משתמשים. המערכת מאוזנת לכל רמות ההתקדמות ומיעדת לכך אדם המעניין חוות מודרנית ומתקדמת של משחקי תלת-מימד כמו Doom או 3D Wolfenstein, ובפרט לקהל שורצה לשחק ביחד עם חברים דרך בקרה קלה ומהירה.

פירוט יכולות:

המערכת "Raycaster תלת-מימדי" תציע מספר יכולות מרכזיות:

1. תצוגה תלת-מימדית מרהייבת:

המערכת תספק תצוגה תלת-מימדית מדויקת ומרהייבת, המשדרת את חוות הגרפיט בצורה מדוודה.

2. משחק אונליין:

אפשר להתחבר ולשחק במשחק יחד עם חברים מחשבים אחרים, באמצעות התחברות לרשת ראשי באמצעות פרוטוקול UDP. מצופה חוות משחק אונליין מהירה ומרהייבת.

3. משחק משתמש יידוטי:

המערכת מתוכננת עם משחק המשתמש יידוטי המספק אמינות ונוחות בשימוש, כולל פעולות באמצעות המקלדת והעכבר.

4. בדיקות תקינות:

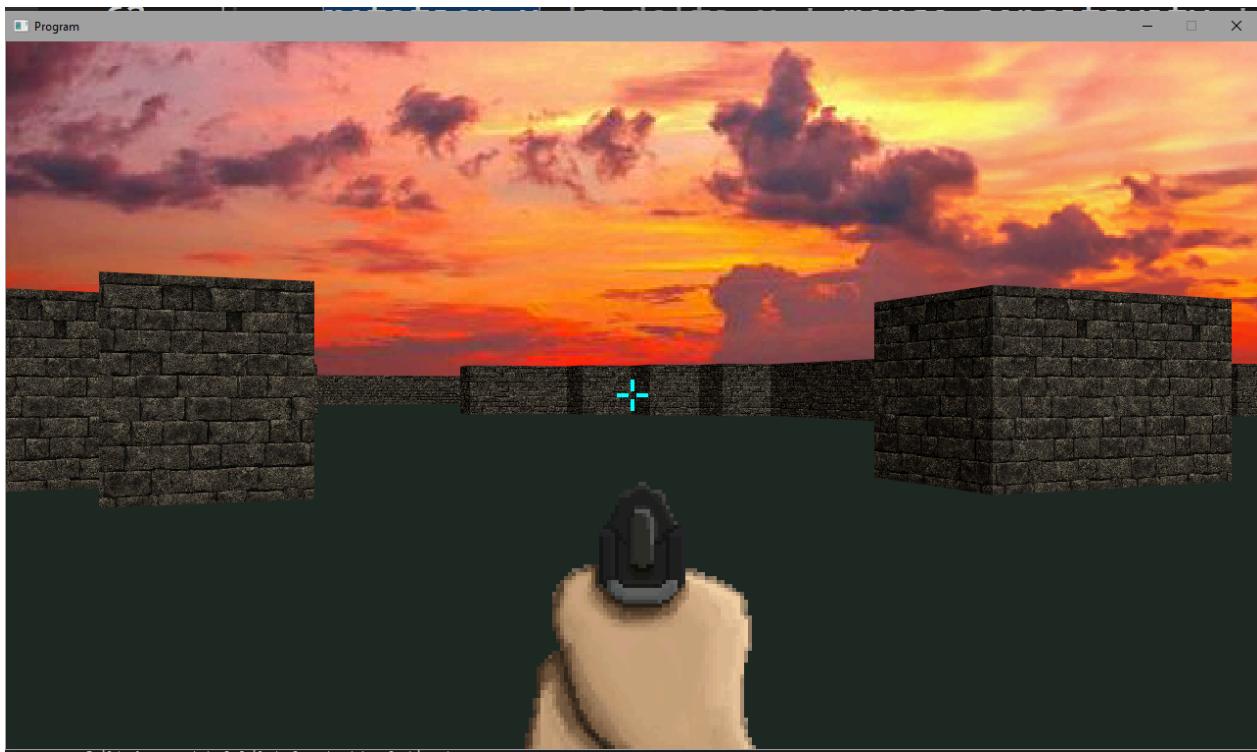
יתבצעו בדיקות תקינות משתמשות וקפדיות, כולל בדיקת קלט משתמש, להבטחת תקינות, יציבות ותפקה גבוהה.

5. פריצות דרך בגרפיקה:

הפרויקט יכול לאפשר לחזור ולהתעמק בוIFI של העולם התלת-מימדי שבוני, ולהתפעיל מהאלגוריתמים הקלואיסיים שמאפשרים את חוות המרייבת של המשחק, אותם כתבתי בשפה נומקה כל כר כמו ++.

לייעד קורן - Raycaster

תמונה של המצב הנוכחי של המשחק

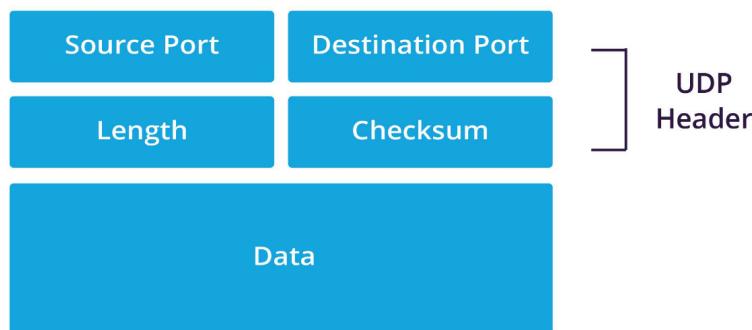


הסיכונים בפרויקט והדריכים להתמודדות איתם:

הפרויקט "Raycaster תלת-מימדי" נתקל בסיכונים ואתגרים רבים. כאשר אנו נתקלים בבעיות גבוהות של אובייקטים בסביבה תלת-מימדית, יהיה علينا למצוא דרכים ייחודיות להתמודד עם האתגרים הטכניים והגרפיים بصورة יעליה. אתגרים אלו יעלו כאשר נרצה להציג מה להציג במסcisיהם של כל אחד מהשחקנים, יותר מזאת שנרצה לבדוק אם הירייה של השחקן האחד אכן פגעה באיברים או שמא היא פגעה בקיר.

בנוסף, כאשר משתמשים ב프וטוקול UDP עבור משחק רשת בזמן אמיתי, יהיה علينا לפתח מנגנון סינכרון אמין ויעיל, היכול לספק חווית משחק חלקה וחווית משתמש נעימה.

בדיקות מעמימות ישפרו את יכולת האיתור ותיקונים של בעיות ובאגים, למרות שהפיתוח עלול להתקדם بصورة יותר ארוכה קלות. כמו כן, נוחות המשמש נמצאת בעדיפות עליונה ועלינו להימנע מבעיות הצד המשתמש שיגרמו לשחקנים לחושם גורעים במשחק או שהמשחק גורע.



sekirat choloshot vohaimim ul hatakorot (abtachah):

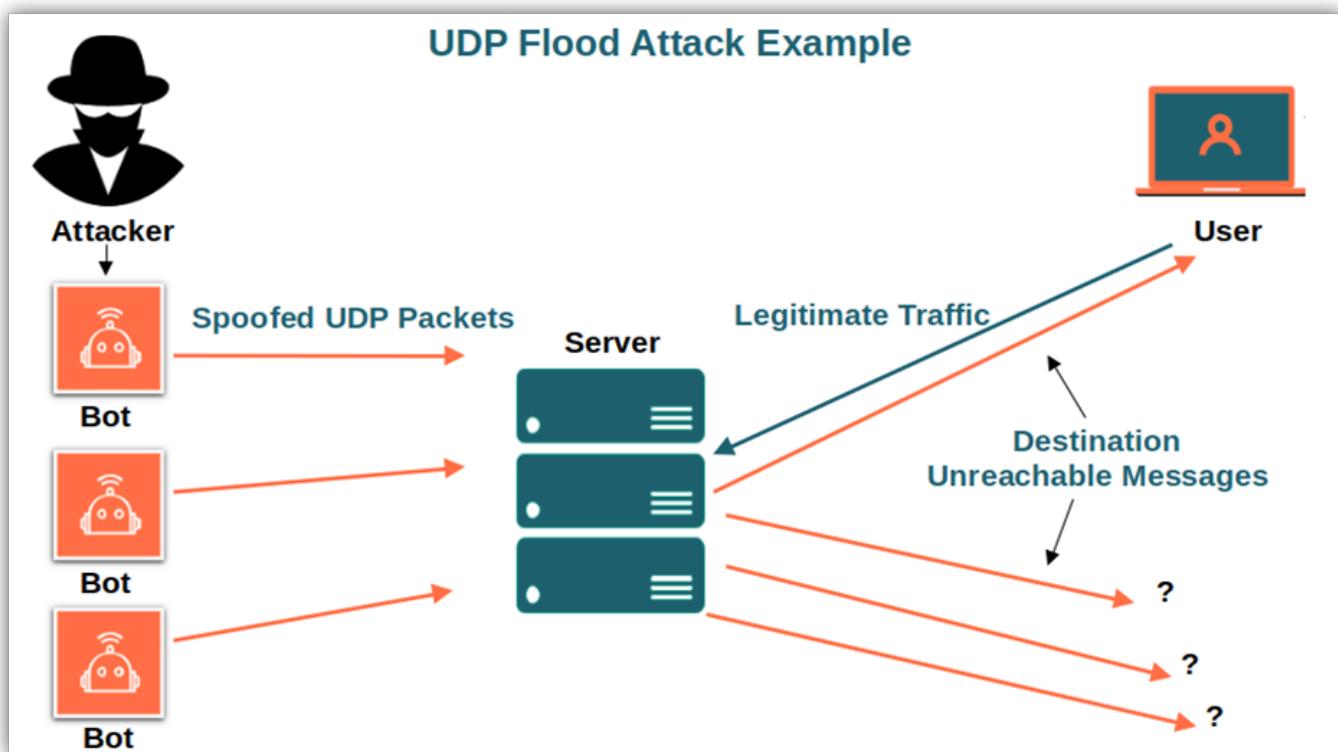
בפרויקט "Raycaster תלת-ממדי", נזהה איום וחולשות בתחום התקשרות, בעיקר בהקשר של אבטחת המידע:

אחד מחולשות התקשרות הפטנציאליות בפרויקט היא הגישה לנוטונים של המשתמשים במקרה של משחק רשות. השימוש ב프וטוקול UDP, גורם לנו לאבד את האמינות והבטחה שניתנו להציג באמצעות פרוטוקולים יותר מורכבים כמו TCP. זאת מושם שהמידע שנשלח באמצעות UDP יכול להיאבק באובדן או בדילול בכתובת, וכך נדרש תכנון מיוחד על מנת להבטיח נתיב תקשורת אמין.

כמו כן, חשיבות גבוהה נדרשת למניעות התקפות סייבר. מכיוון שמדובר במסpiel רשות, ישנה חשיבות גבוהה להגן על הנתונים שנשלחים בין השחקנים מפני גישה לא רצiosa של גורמים עוניים ושוניים במידע שנשלח ומתקבל. בעיקר בגלל הקצב הגבוה שבו הנתונים מעוברים במסpielים בזמן אמיתי, נדרש פיקוח ותיקול מראש כדי למנוע את הפרצות התקפות סייבר.

בנוסף, במקרה של עמוד הראשי להתחברות למשחק דרך רשות, יש להתמודד עם אתגרים של ביצועיות ויכולת להתמודד עם כמות רבה של בקשות משתמשים רבים בו זמן. נדרש ייעול וטיפול תקין וחכם מצד השירות על מנת למנוע הפסקות פעולה או איטיות במהלך המשחק.

על מנת להתמודד עם חולשות והאומים הללו, נדרש פיתוח ייציב, מחקר בטכנולוגיות אבטחה, והטמעת מנגנונים שישפקו מגן אל מול האומים האפשריים שהוצעו לעיל.

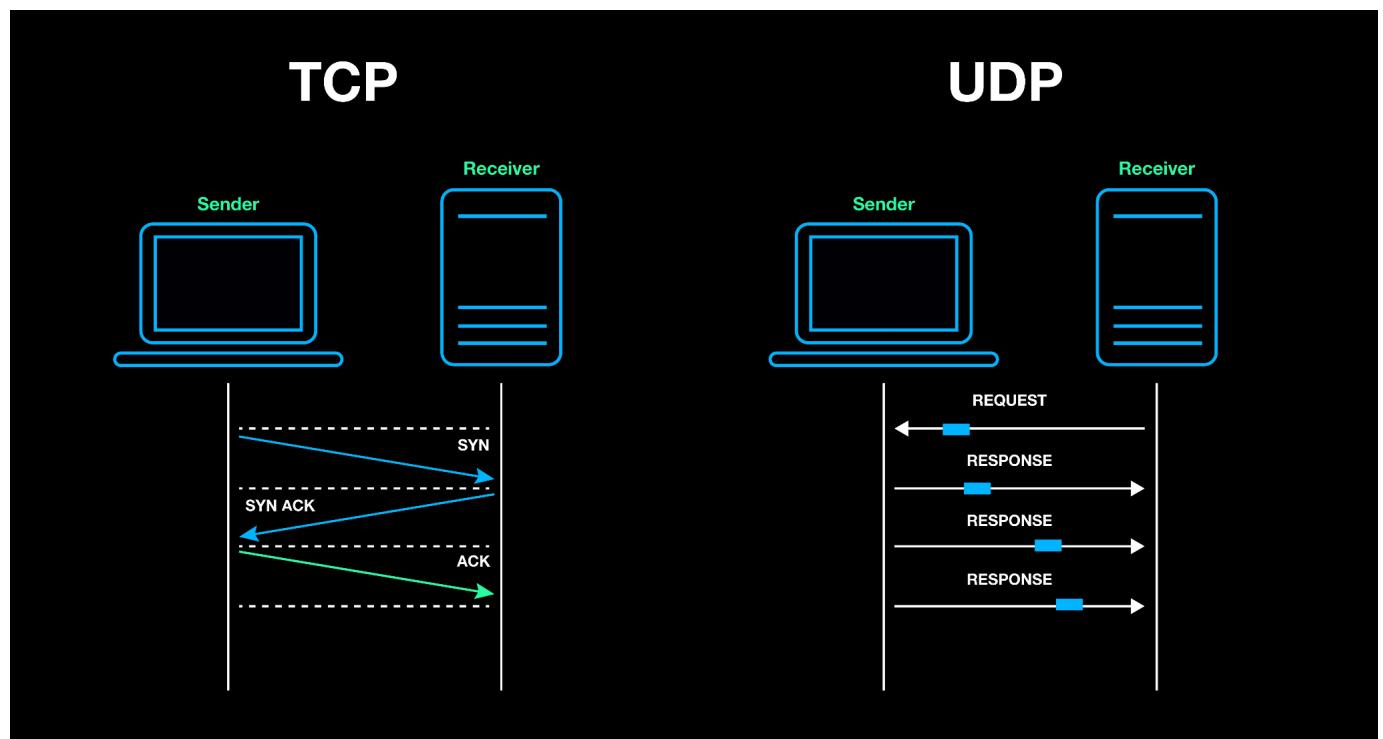


פירוט בדיקות:

במהלך הפרויקט, יתבצעו מגוון בדיקות כדי להבטיח יציבות וביצועים טובים של המערכת. כל חלק בפרויקט ישם באופן ייחודי ובודק בקפידה על-ידי בדיקות אוטומטיות.

לדוגמה, בדיקת קלט משתמש תבצע חלק מתהליכי האינטראקציה שבין הלוקו לשרת, בו יודגשו תקינות, תגובה מהירה וabetחה בכניסה המשמש למערכת והפעולות כחלק منها.

בדיקות אלו מוספות למערכת במקביל ובמהלך הפיתוח כדי להבטיח תקינות יציבות תוך כתיבת המערכת.



תכנון וניהול לו"ז לפיתוח המערכת:

25/10/23	נושא עד תאריך מנוע גרפי מאפשר לתנועה בעולם תלת מימדי שניינט לעצוב
2/11/23	הוספת איקדח ואנימציות בעת ירידיה
6\12\2023	מתן אפשרות להירושם لتוכנה, והכנת מערכת התקשרות הבסיסית.
2023\12\20	מתן אפשרות להתחבר לשחקן אחר ולשחק אליו
7\3\2023	הוספת אויבים או מטרות
2023\3\17	שלב משחק שלם מוגדר עם אויבים או מטרות

תיאור תחום הידע

• פירוט מעמיק של יכולות שהוצעו בשלב הקודם ומעבר:

פירוט יכולות בצד ל Koh:

1. שם יכולת: הרשמה למערכת.

מהות יכולת: הרשמה למערכת על מנת להתחילה בשלב ההכנה לתחילת שימוש. אוסף יכולות דרישות:

- ממשק גרפי.
- קליטת נתונים.
- בדיקת נתונים.
- הצפנה.
- שליחה לשרת מרכז.
- המתנה לתשובה מן השרת.
- פענוח תשובה.
- הציגה למשתמש.

אובייקטים נוחוצים: ממשק משתמש, הצפנה ופענוח, תקשורת.

2. שם יכולת: יצירת חדר

מהות יכולת: מתן יכולת משחק עם משתתפים נוספים
אוסף יכולות דרישות:

- קליטת נתונים.
- בקשה משרת.
- הצפנה.
- פענוח.
- ממשק גרפי.
- ממסד נתונים.

אובייקטים נוחוצים: ממשק משתמש, ממסד נתונים, הצפנה ופענוח, תקשורת, אובייקט שחקן.

3. שם יכולת: שינוי הגדרות משחק

מהות יכולת: מתן שליטה ליוצר החדר לשלוט על ההגדרות
אוסף יכולות דרישות:

- קליטת נתונים.

- הצפנה.
- ממשק גרפי.
- שליחה לשרת מרכז.
- פענוח.

אובייקטים נוחוצים: ממסד נתונים, ממשק משתמש, תקשורת, הצפנה ופיענוח.

4. שם היכולת: התחלת משחק
מהות יכולת: מתן יכולת ליצור החדר להתחיל את המשחק כשהוא רוצה אוסף יכולות דרישות:

- ממשק גרפי.
- בקשה שירות מרכז.
- פענוח.
- הצפנה.

אובייקטים נוחוצים: ממשק משתמש, תקשורת, פענוח והצפנה.

5. שם היכולת: כתיבה ביצ'אט המשחק
מהות יכולת: שימוש בגיבויים שעל השירות המרכז.
אוסף יכולות דרישות:

- ממשק גרפי.
- בקשה שירות מרכז.
- פענוח.
- הצפנה.

אובייקטים נוחוצים: ממשק משתמש, תקשורת, פענוח והצפנה.

פירוט היכולות בצד שירות

1. שם היכולת: הזרחות משתמש.

מהות יכולת: בדיקת תקינות מידע משתמש בהתאם למאגר המידע. אוסף יכולות דרישות:
• בדיקת נתונים.

- הצפנה.
- שליחה ללקוח בית.
- פענוח תשובה.
- הצגה למשתמש.
- גישה למאגר מידע
- עבודה עם LOCK.

אובייקטים נוחוצים: עבודה עם מאגר מידע, הצפנה ופענוח, תקשורת.

2. שם היכולת: עדכון משתמש בנוגע למשחק
מהות יכולת: כל שחקן מקבל מידע על כל שאר השחקנים כדי שיוכל לציר אותם על המסך שלו.

אוסף יכולות דרישות:

- קלייטת נתוניים.
- הצפנה.
- שליחה ללקוח .
- פענוח.

אובייקטים נחוצים: הצפנה ופענוח, תקשורת.

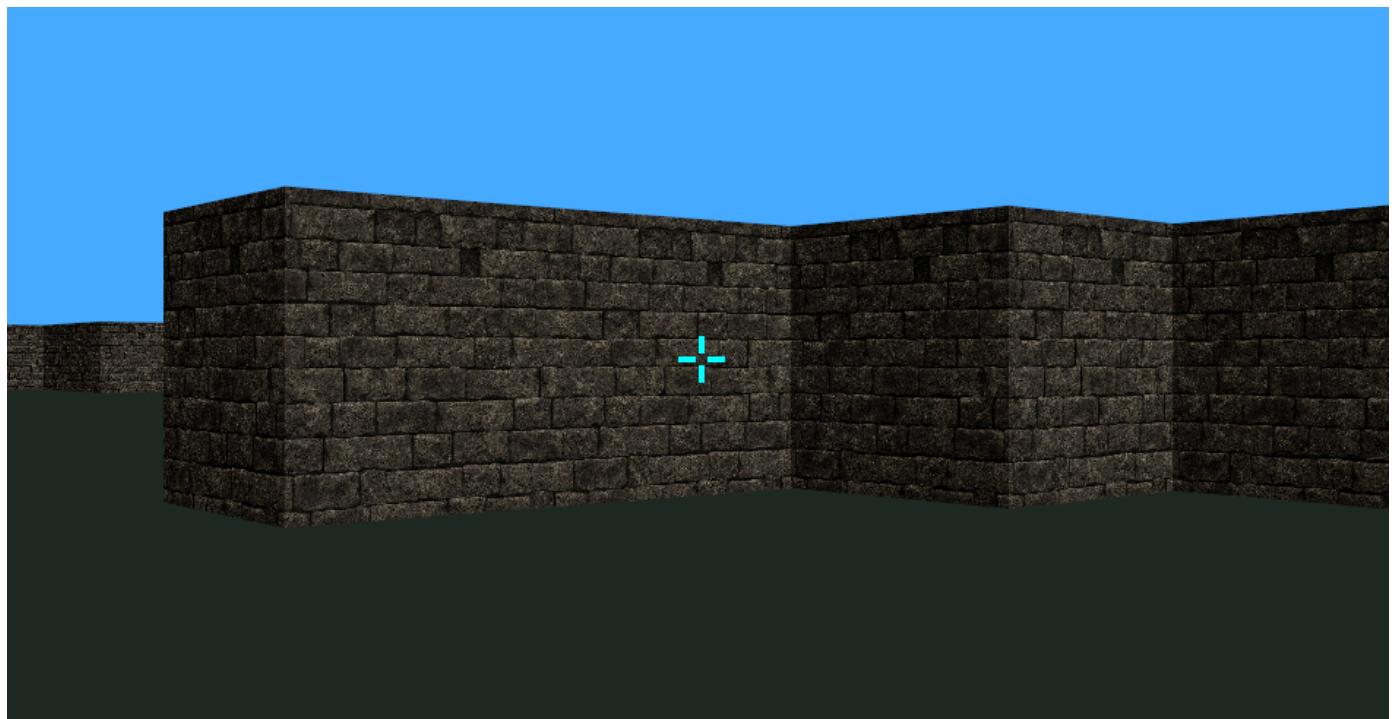
3 שם היכולת: ניהול משחק
מהות יכולת: תהליכי האחראי על ניהול כל חדר משחק.

אוסף יכולות דרישות:

- קלייטת נתונים.
- הצפנה.
- העברת נתונים.
- תהליכיונים.

אובייקטים נחוצים: הצפנה ופענוח, תהליכיונים.

תמונה התקדמות מההתחלת של הכנת המנוע הגרפי:

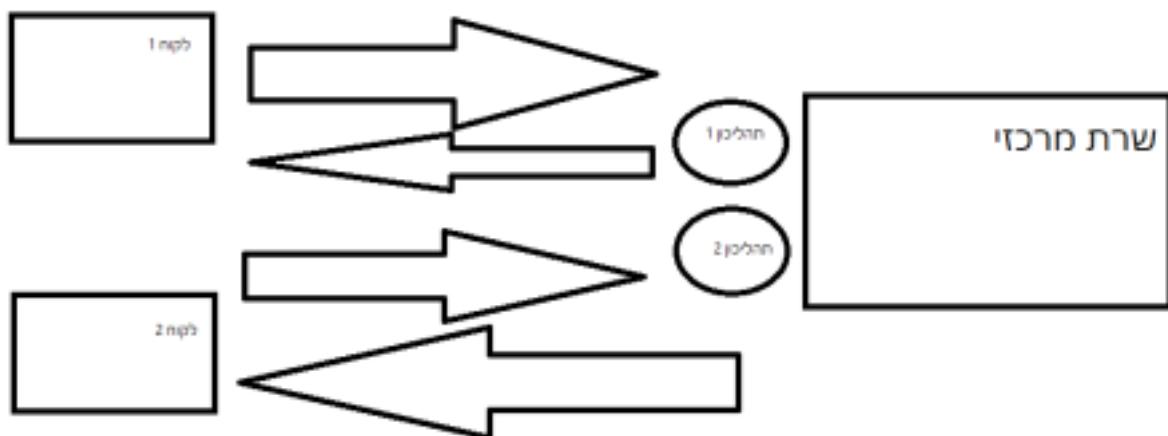


מבנה

• **פירוט החלטות שנלקחו לימוש המערכת (עיצוב):**

תיאור הארכיטקטורה של המערכת המוצעת:

- תיאור החומרה – רכיבים שונים והקשרים ביניהם: המערכת תוכל צריכה להיות במחשב היפעל את שרת המשחק ובמחשבים היפעלו את תוכנת הלוקוח. תוכנת השרת והלוקוח יתקשרו ביניהם.
- צירוף שרטוט המציג קשרים אלו -



תיאור הטכנולוגיות של המערכת:

הטכנולוגיות הכלולות בפרויקט יהו: שפת התכנות C++, תקשורת באמצעות סוקט, הצפנה באמצעות השיטה המייחודת שלו.

- **C++** פלאס פלאס (באנגלית: C++) היא שפת תכנות לא עילית, סטטיטית למטרות ספציפיות מהנדירות ביתר. CPP תוכננה תוך שימוש דגש על אי קריאות הקוד, וככללית מספר אפסי של מבנים המיועדים לאפשר ביטוי של תוכניות מורכבות בדרך קצרה וברורה, בעיקר סיבוכים.

השפה משלבת בין תוכנות של שפות תכנות מקוריות וABBYYCAT'יות, מה שהופך אותה לשפת תוכנות מתקדמת וგמישה. C++ נוצרה כהרחבה של השפה C, והוא מוסיף אלמנטים מתקדמים כמו ירושה, פולימורפיזם, וניהול זיכרון אוטומטי.

שפה זו נפוצה בפיתוח תוכנה בסגמנטים שונים, כולל פיתוח מערכות מורכבות, משחקים, "ישומי מחשב", תוכניות מוטוריות, ועוד. ה יתרונות המרכזיים של C++ הם יכולת כתיבת קוד מהיר ויעיל, יכולת לנוהל משאבים באופן יעיל, והאפשרות ליצור תוכנות קروس-פלטפורמה.

שפת התכנות C++ מצrica הינה טובה של מספר מושגים בסיסיים כמו משתנים, פונקציות, מחלקות, וירושה.

יש לה למבנה קוד ברור וארגון, והיא מספקת כל' לניהול זיכרון באופן ידני וגם באופן אוטומטי. בנוסף, השפה מספקת ספריות רבות לפתרון מגוון רחב של שימושות תכניות, כולל גרפיקה, תקשורת, ועוד. C++ היא כל' חשובה לתוכננים שרצו לפתח תוכנות מתקדמות ויעילות במגוון תחומים.

ס' פלאס פלאס מדורגת באופן עקבי כאחת משפטות התכנות הפופולריות ביותר.

פרוטוקול התקשרות בכלליות:

במהלך המשחק כאשר השחקנים מחוברים והמשחק רץ, קיימת תקשורת פעילה בין הלוקוחות והשרת. ההודעות שנשלחות בין הלוקוחות לשרת משמשות להעברת מידע שוטף ועדכוניים רלוונטיים למצב המשחק. להלן תיאור כללי של מהלך ההודעות במשחק:

החברות לשרת:

כאשר שחקן חדש מתחבר למשחק, הוא ישלח הודעה לשרת כדי להצטרף. השרת קיבל את ההודעה ויבצע את התchapות השחקן לשון המשחק. התהילה ישולם כאשר השחקן מצטרף בהצלחה ומקבל אישור מהשרת.

עדכן מערכת:

בכל שלב במשחק, הלוקוח והשרת יתקשרו כדי לעדכן אחד השני על מצב המשחק. הלוקוח ישלח הודעה עם המיקום הנוכחי שלו, פעולות שלו, ומידע נוסף כדי שהשרת יוכל לעדכן את המשחק. השרת ישלח הודעה דומה לлокוחות אחרים כדי לעדכן אותם על השינויים.

פעולות משחק:

השחקן בлокוח יכול לבצע פעולה במשחק כמו יריות, תנואה, ופעולות נוספות. כל פעולה צו תודיע לשרת באמצעות הודעה מתאימה והשרת יעדכן את שאר הלוקוחות.

התנטקות:

כאשר שחקן מתנתק מהמשחק, הלוקוח שלו ישלח הודעה לשרת על ההתנטקות. השרת יכול לסגור את החיבור ולעדכן את שאר השחקנים על ההתנטקות.

טיפול בקייסת משתמש:

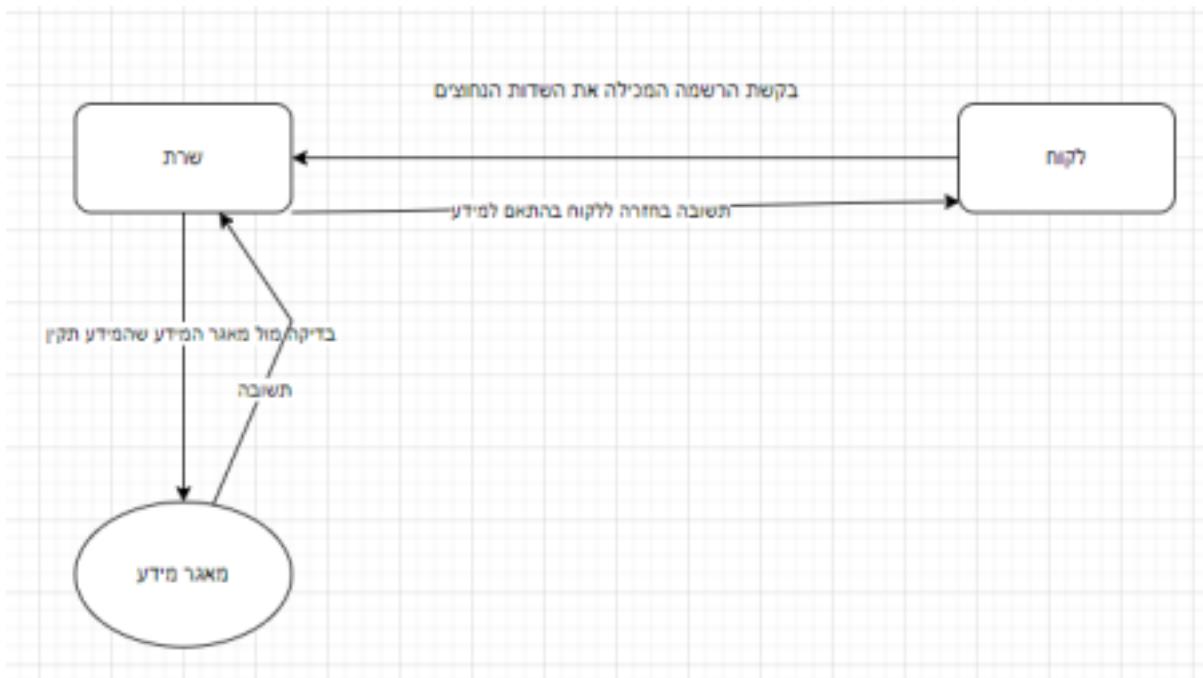
כאשר צד הלוקוח קורט, שום דבר לא נהרס מצד השרת. זאת מפני שהתקשרות בין הצדדים במהלך המשחק קורית בעזרת UDP וUDP אינו פרוטוקול מבוסס קשר. התוצאה היחידה של ההתנטקות היא שהשרת מפסיק לקבל הודעות מהлокוח.

ישנו חוט (thread) שרצ' בשרת שבודק כל כמה שניות אם ישנו לוקוח שלאשלח הודעה כבר הרבה זמן (3 שניות), ובמקרה זה מנתק את הלוקוח ומוחק את המשתנה Player שלו מרישימת השחקנים החיים. כמובן שהשרת גם מעדכן את כל שאר השחקנים על עציבתו של אותו שחקן.

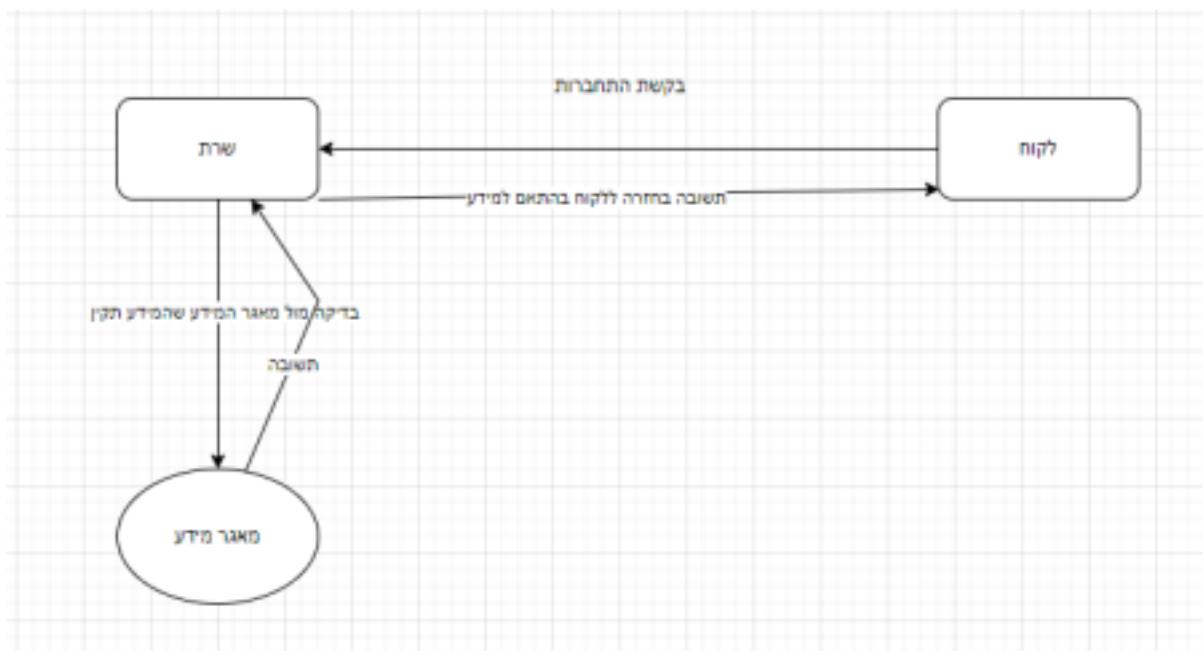
לייד קורן - Raycaster

תיאור זרימת המידע במערכת:

שלב ההרשמה:

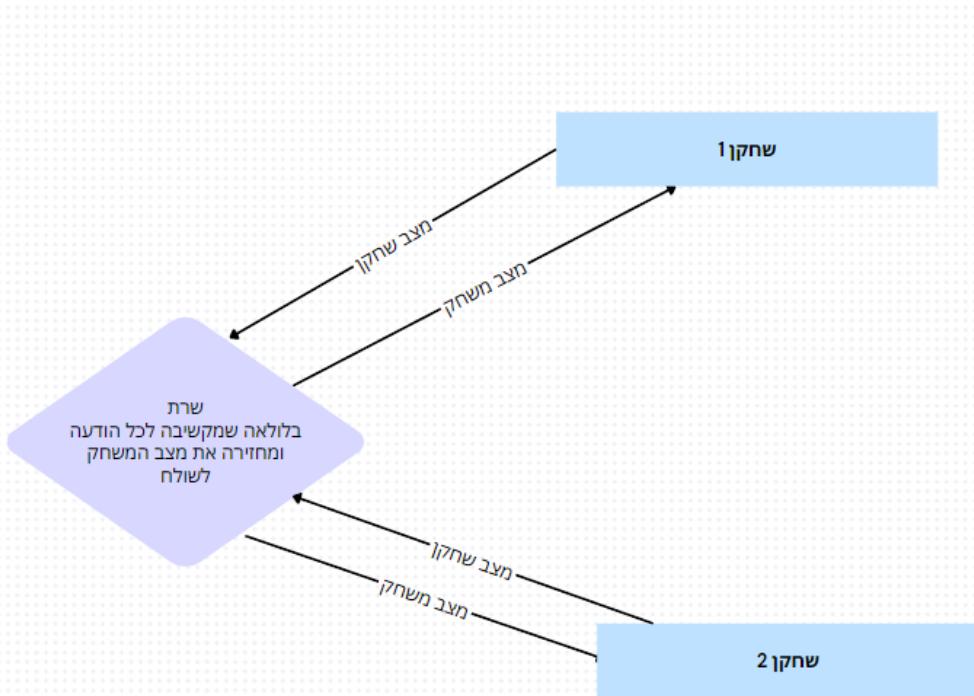


שלב הלוגין:



לייעד קורן - Raycaster

שלב המשחק:



תיאור אלגוריתם Raycasting שמאפשר ייצוג של עולם תלת מימדי במרחב דו מימדי:

האלגוריתם פותח והוציא לאור לפני כ-30 שנה, המציאו אותו כמה חברות מהחברה id software. הם אלו שכ כתבו את DOOM ו-Wolfenstein 3D.

המפה מיוצגת בעזרת מטריצה דו מימדית שמחזיקה מידע על אם אריך ספציפי הוא ריק או קיר. הרינדור עובד בעזרת "שליחת" קרניים מתמטיות כדי לדעת את המרחק בין השחקן לבין הקירות הקרובים (מרחק בין 2 נקודות). אם השחקן קרוב לקיר, הקיר יוצג באופן גדול ומרכזי, ואם השחקן רחוק מהקיר, הקיר יוצג כמעט קטן למרחוק.

כך נוצרת האשליה של תלת מימד אפלו שהמשחק עצמו מיוצג במרחב דו מימדי בלבד.

מקורות למידה:

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

https://www.youtube.com/watch?v=gYRrGTC7GtA&ab_channel=3DSage

```

225
226 void Player::shootRays()
227 {
228     sf::Sprite sprite;
229     sprite.setTexture(wall_txs[0]);
230
231     float angle_offset = -fov_x / 2;
232     float step = fov_x / WIDTH;
233
234
235     map.drawGround();
236
237     // For each column of pixels
238     for (int x = 0; x < WIDTH; x++)
239     {
240         HitInfo hit_info = shootRay(angle_offset);
241
242         float dist = hit_info.distance * cos(angle_offset);
243
244         float len = 1000 / dist;
245
246         if (hit_info.on_x_axis)
247             sprite.setTexture(wall_txs[1]);
248         else
249             sprite.setTexture(wall_txs[0]);
250
251
252         // drawing
253         sprite.setTextureRect(sf::IntRect(
254             v2i(hit_info.texture_x * wall_txs[0].getSize().x, 0), v2i(1, wall_txs[0].getSize().y)
255         ));
256         sprite.setScale(1, len / wall_txs[0].getSize().y);
257         sprite.setPosition(x, map.floor_level - len / 2);
258         window.draw(sprite);
259
260
261
262         angle_offset += step;
263     }
264 }
265

```

תיאור סביבת הפיתוח:

השתמשתי ב 2 סביבות פיתוח מרכזיות, אחת לצד הלקוח ב C++, והשנייה לצד השרת ב Python.

הסיבה שבה עשית שימוש בכתיבת הלקוח ב C++ הייתה סביבה של Microsoft Visual Studio. זו הינה סביבה של Microsoft Visual Studio לעריכת קוד ופיתוח תוכנה הפעלת על מערכות הפעלה Windows, Linux ו- OS. העורר תומך בניפוי שגיאות, בקרת גרסאות של GIT, המחשה סינטקטית של קטעי קוד, השלמת קוד חכמה automatic code completion, קטעי קוד אוטומטיים snippets ושינויי קוד רוחביים refactoring code .
Boost::Multiprecision, SFML, OpenSSL

הסיבה שבה תכננתי את צד השרת ב Python היא Visual Studio Code, מאוד נוח, בדומה ל VSCode, אם כי VSCode נחשב יותר "קל" ונוח, יותר טוב לעבודה עם פיתון לטעמי, איפה שדרושים פחות משתי סביבה ושליטה על הקומpileציה.

תיאור פרוטוקול התקשרות:

- העברת המידע הראשוני, בעת ההתחברות, מבוצע בעיקר באמצעות פרוטוקול התקשרות Protocol Control Transmission (TCP/IP) (בראשי תיבות: TCP) שהוא פרוטוקול בתקשורת נתונים הפעל בשכבות התעבורה של מודול ה-OSI ומודול ה- TCP/IP, ומבטיח העברה אמינה של נתונים בין שתי תחנות בראשת, באמצעות ייצור חיבור מקוישר. כמובן שבעת ההתחברות נעשית מאחורי הקלעים לחיצת יד משולשת, בה השרת והלקוח מבוססים את החיבור בעזרת שליחת SYN-ACK ו-ACK.

TCP מעביר את הנתונים שהועברו באמצעות IP, מודיא את נוכנותם, ומאשר את קבלת הנתונים במלואם או מבקש שליחה מחדש של נתונים שלא הגיעו בצורה תקינה.

- במהלך המשחק עצמו, פרוטוקול התקשרות משומש כדי להעביר את המידע על השחקנים והמשחק הוא UDP, שם מלא User Datagram Protocol. ההבדל בין TCP לUDP במקורה זהה הוא שUDP לא מאשר את קבלת כל ההודעות. החיסרון של זה הוא שהאמינות של ההודעות נפגעת, אבל היתרון הוא שהמעבר של ההודעות יכול לKEROT מהר יותר.

מהירות קבלת ההודעות בזמן המשחק היא קריטית, היה ומדובר במשחק בזמן אמיתי, וכל שחקן צריך לדעת איפה נמצא שאר השחקנים בדיק באותו רגע, כדי שיווכל לראות את תנועתם באופן חלק.

תיאור התקשרות שבין הלקוח לשרת:שלב 1: שלב ההתחברות לשרת

תיאור ההודעה	שולח	נמען	מבנה ההודעה
--------------	------	------	-------------

<u>שלב 1: שלב ההתחברות לשחקן</u>				
X1 Diffie Hellman	לקוח	שרת	X157623699307983268930944835066795715439X	
X2 Diffie Hellman	שרת	לקוח	X19759893347675296600793514949522752105X	
כעת השרת והלקוח משתמשים בX1 וX2 כדי להגיע למפתח AES משותף. כל ההודעות מעכשי יוצפנו בעזרתו.				
<u>שלב 2: שלב ההתחברות לשחקן מוצלח</u>				
ההרשמה הצלילה	לקוח	שרת	נוסין הרשמה <pre>string creds = "LOGIN~" + username + "~" + password + "~"; sendEncryptedTCP(creds, error);</pre>	
ללקוח מקבל את המספר משתמש שלו, מספר דינامي שמייצג אותו וرك אותו			<pre>if(users_db.user_exists(parts[1], parts[2])): response = "SUCCESS~" + str(current_player_id) + "~" key_bytes[current_player_id] = current_key_bytes</pre>	
			שרת שומר את המפתח במילון לצד מספר המשתמש של השחקן, בשביל תקשורת מוצפנת בזמן המשחק עם UDP	

<u>שלב 1 שלב</u> <u>ההתחברות למשחן</u>				
X1 Diffie Hellman	לקוּחַ	שרת		X157623699307983268930944835066795715439X
X2 Diffie Hellman		שרת	לקוּחַ	X19759893347675296600793514949522752105X
כעת השרת והלקוּח משתמשים בX1 ו X2 כדי להגיע למפתח AES משותף. כל ההודעות מעכשי יוצפנו בעזרתו.				
<u>שלב 2 שלב</u> <u>ההתחברות למשחן</u> <u>כשל</u>				
ניסיונו הרשמה נסילה	לקוּחַ	שרת		<pre>string creds = "LOGIN~" + username + "~" + password + "~"; sendEncryptedTCP(creds, error);</pre>
	ההרשמה נסילה	לקוּחַ	שרת	<pre>response = "FAIL~Username Or Password Incorrect~"</pre>
				לקוּח מקבל את ההודעה ומציג אותה על המסך, התחברות נסילה זואת בגלל שם המשתמש או הסיסמה שגויים.

<u>שלב 1 שלב</u> <u>ההתחברות למשחק</u>					
X1 Diffie Hellman	לקוֹחַ	שרָׁתַּה	X157623699307983268930944835066795715439X		
X2 Diffie Hellman	שרָׁתַּה	לקוֹחַ	X19759893347675296600793514949522752105X		
		כעת השרת והלקוֹח משתמשים בX1 ו-X2 כדי להגיע למפתח AES משותף. כל ההודעות מעכשי יוצפנו בעזרתו.			
		<u>שלב 2 שלב יצרת</u> <u>משתמש</u>			
שם משתמש וסיסמה	לקוֹחַ	שרָׁתַּה	<pre>string creds = "SIGNUP~" + username + "~" + password + "~"; sendEncryptedTCP(creds, error);</pre>		
הצלחה	שרָׁתַּה	לקוֹחַ	<pre>response = "FAIL~username already exists~" elif(users_db.insert_new_user(parts[1], parts[2])): response = "SUCCESS~"</pre>		
			לקוֹח כותב על המסר שההתחברות הצליחה ועובר למסך התחברות משתמש קיים		
מצב המשחק	לקוֹחַ	שרָׁתַּה			

תשובה, מצב משחק	שרת	לקוח	

פרוטוקול התקשרות של מהלך המשחק עצמו:

```
struct PlayerInfo
{
    enum Flag
    {
        moving = 1,
        forward = 2,
        gun_shot = 4,
        quit = 8,
        got_shot = 16,
        dead = 32
    };
    int player_id;
    float dist2wall;
    float pos_x, pos_y, rot_x, rot_y;
    int flags;
};
```

השחקן מחזיק את כל המידע הנוכחי על עצמו במבנה נתוניים, מחלוקת ציבורית שקוראים לה `PlayerInfo`.

את המידע זהה הוא מצפין באlgorithם AES הסכימו עליו בתהילר הולמן Diffie Hellman שנעשה בעת ההתחברות.

למיעוד המוצפן הוא מצמיד מקדים, באופן לא מוצפן, את ה`id` `player` שלו.

unlessו השרת מקבל הודעה UDP, והוא לא יודע באיזה מפתח לשימוש כדי לפתח את הצפנה שלה.

השרת משתמש ב`playerid`, הבית הראשון בהודעה כדי לדעת מאייזה לקוב ההודעה הגיעה, ומשתמש במפתח הקשור לו לאותו לקוב כדי להצפין את שאר ההודעה, המידע עצמו על השחקן.

את המידע הזה הוא מעבד בפייטון, ומצירף אותו למשתנה בינארי שנקרא `player_binaries`.

כתגובה להודעת UDP שהשחקן שלח, (בה הוא מעדכן את השרת בנוגע למיקום הנוכחי שלו והמצב הכללי שלו), השרת שולח בחזרה את המשתנה `player_binaries` כולם, ועוד מידע שקרהתי לו `events`.

ב`player_binaries` שמורות המיקומים היכי עדכנים של כל שאר השחקנים, בנוסף למיקום של השחקן עצמו. כשהלך מקבל את זה הוא יכול לצייר את כל שאר השחקנים במיקום המתאים שלהם.

ב חלק שנקרא `events` השרת מספור ללקוח על דברים שלא שמורות ב`player_binaries`, דברים כמו יריות, פגיעות, ושחקנים שנרגו.

תיאור שכבת האפליקציה, בפרט העבודה מול ממסד הנתונים:

לפני כל התחברות בין הלקוח לשרת, בין אם במטרה להתחבר למשחק או רק ליצור משתמש, קוריות לחיצת יד משולשת כדי לבסס את קשר TCP, ואז פרוטוקול העברת מפתחות בסיסי בשם Diffie Hellman, שם הלקוח והשרת משתמשים במספרים רנדומליים עצומים בגודלים (עד 128 ביטים) כדי להגיע לאותו מפתח בגודל 16 בתים בהם ישמשו כשם מצפינים ומפתחים כל הودעה בעזרת אלגוריתם AES128.

בעת יצירת משתמש, שם המשתמש והסיסמה עוברים כר, מוצפנים, אז השרת בודק אם שם המשתמש כבר קיים במערכת. במקרה שלא, הוא יוצר מהירות רנדומלית בת 5 תווים, מה שנקרא `salt`, מכרף אותה לסוף הסיסמה, מגבב אותה בעזרת SHA256, ושומר את הגיבוב לצד שם המשתמש וה`salt`.

כעת, כשהמשתמש רוצה להתחבר לשרת, הלקוח שולח באופן מוצפן את שם המשתמש והסיסמה (AES). השרת מאמת שאכן קיים שם משתמש זהה, ועוד מוסיף את ה`salt` הציבורי ששמור במאסד לצד שם המשתמש לסיסמה שכעת קיבל מהמשתמש. אם לאחר גיבוב המחרוזת החדשה התקבלת יוצאה זהה זו ששמורה לצד שם המשתמש, השרת שולח ללקוח שהכניסה שלו התאפשרה ויוצר לו משתמש מסווג Player בצד הפיתון. כעת התקשרות של UDP בין השרת ללקוח תחל והsocket של TCP נסגר.

```
self.cursor.execute(  
    "SELECT username FROM Users WHERE username=? AND password=? ;",  
    (username, hash_pass))
```

אין סכנה של injection sql, (הכנסתה של נתונים זדוניים בצד הלקוח כדי לשלוט על ממסד הנתונים). עשייתי כמה פעולות כדי למנוע התקפה זו:

- הנתונים של הלקוח עוברים למאסד בדרך נקייה, כך sqlite יודע שהשם משתמש הינו בסך הכל פרמטר של השאלה, ולא חלק מהשאילתת עצמה.
- שם המשתמש והסיסמה חיבבים להיות בעלי אורך של מעל 4 אותיות, ופחות מ16, כך שם משתמש ריק אפיו לא נשלח לשרת, והוא נדחה בצד הלקוח.
- אסור להכניס סימנים מיוחדים כמו ~ או רווח '' לשם המשתמש או לסיסמה, וזה עוזר עם מניעה של שליטה בפרמטרים הנשלחים לשרת.
- שחררי כדי להבין את תוכן ההודעות, השרת מפצל אותן לפי ~ או רווח.

מדריך למשתמש

תיאור מערכת קבצים:

ישנה תיוקית פרויקט גדולה שבה נמצאים קבצי הקוד והגרפיקה של המשחק, ובה אני משתמש כשיי מתכונת את המשחק.

משתמש:

יש תיוקיה קטנה יותר, תיוקית ההרצה, שם נמצא exe הסופי וקבצי הגרפיקה של המשחק. הספריות נמצאות בתיקיות שנקרוות include וlib, המשתמש לא צריך לדאוג מהדברים האלה, היה זה נמצאות במקומן המתאים בתיקית ההרצה. המשתמש מקבל רק את תיוקית ההרצה וכל שעליו לעשות זה להריץ את executable בשם Raycaster.exe

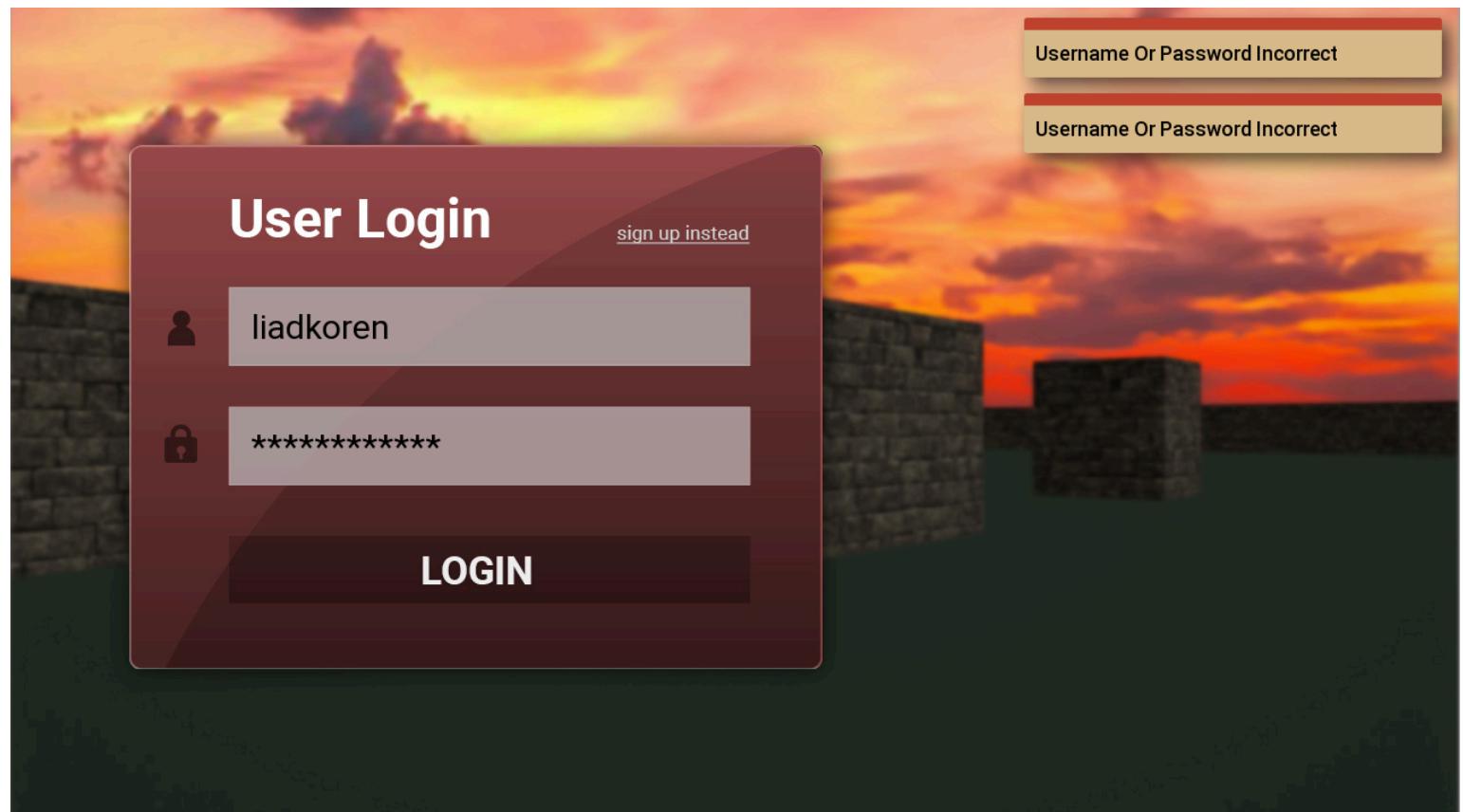
מנהל מערכת:

תיוקית השרת גם קריטית להרצה המשחק, והיא נמצאת בתיקיה שקוראים לה server.サーバー. בתיקיה זו נמצא קובץ פיתון בשם server.py וצריך להריץ אותו עם פיתון 3.10 ומעלה.

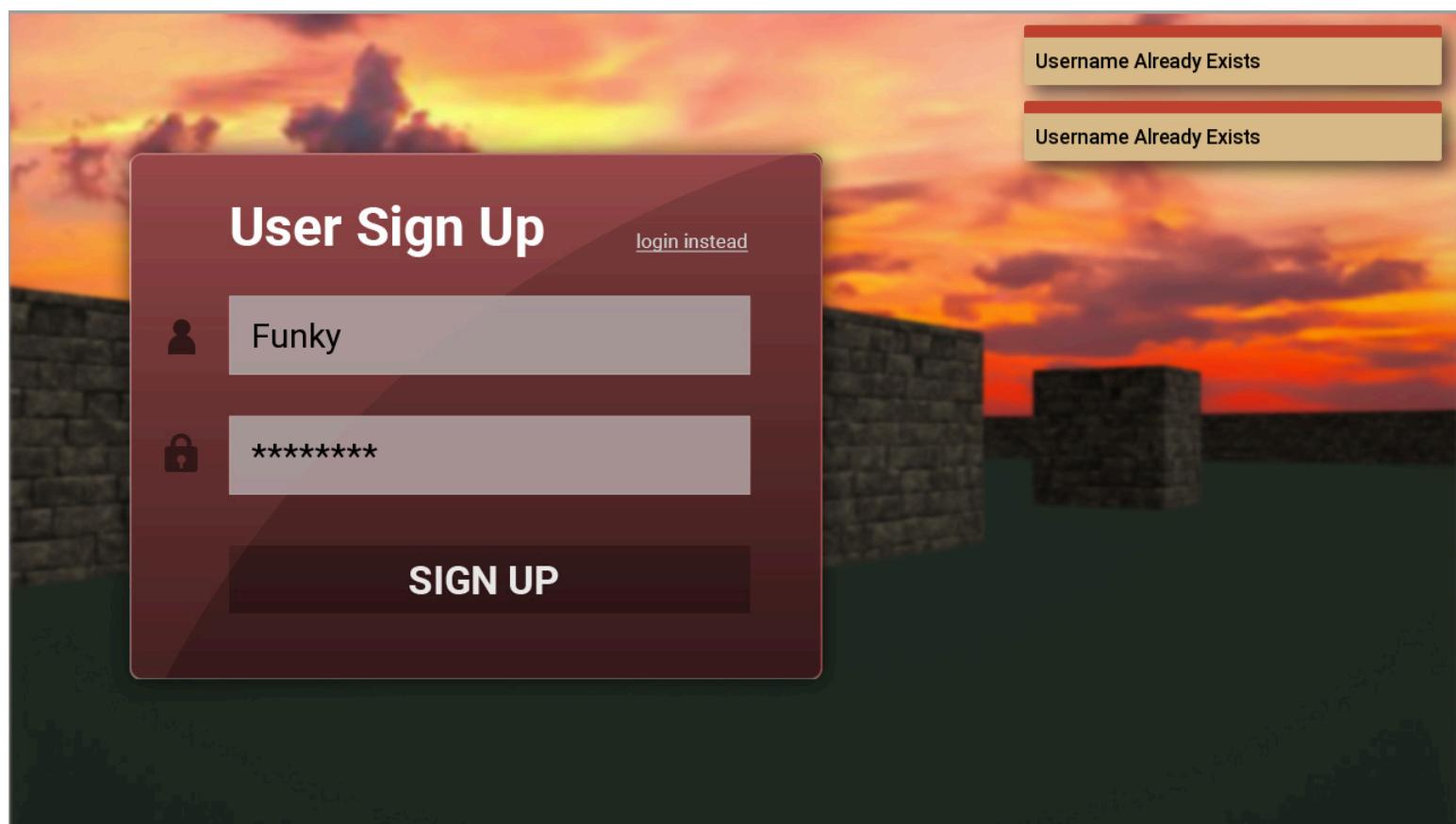
יש בתיקית השרת קובץ בשם server_address.text, שם הלקוח מסתכל כדי לדעת את כתובתו של השרת, הפקיד על השרת שדרכו נעשית תקשורת TCP והפקיד של UDP.

תיאור מסכי המערכת:

- שם / תפקיד - מסך כניסה משתמש קיימ.
- תיאור - מסך ובו תבוצע הזרהוות משתמש לאחר ההרשמה בתוכנה.
- תמונה מסך –



- שם / תפקיד – מסך הרשמה לתוכנה, ייצור משתמש.
תיאור - מסך בו תבצעו הרשמה של המשתמש לתוכנה. המשתמש כותב את השם המשתמש הרצוי ואת הסיסמה הרצויה, ובעת לחיצה המידע מוצפן, נשלח לשרת, והשרת מחזיר תגובה.
אם ההתחברות הצליחה - המשתמש עובר למסך ההירשות.
אם ההתחברות נכשלה - המשתמש רואה הודעה קופצת שמספרת על סיבת הכישלון, או שהשרת לא פעיל, או שהשם משתמש כבר קיים.
- תמונה מסך -

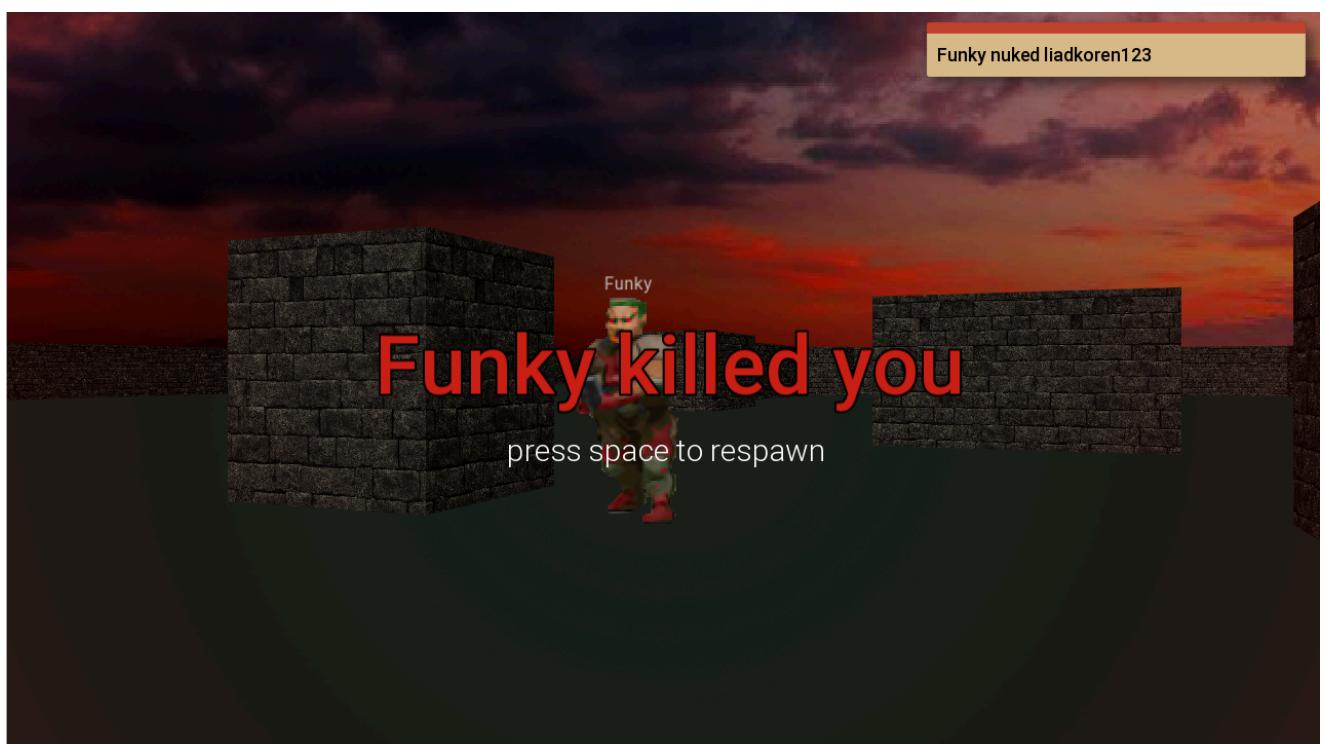


לייעד קורן - Raycaster

- שם / תפקיד – מסך משחק ראש.
- תיאור – מסך שמוצג בו המשחק הגרפי שהכנתתי.
- תמונה מסך -



- שם / תפקיד – מסך מות.
- תיאור – לאחר שחקן אחר יורה בר מספר פעמים, ונגמרים לירח חיים, קופץ המסך הזה שמספר לירח, אתה לא יכול לירות או לזרז, אתה יכול לחזור לחיים במקום אחר במפה.
- תמונה מסך -



מיומש הפרויקט

תיאור מודולים מרכזיים, מיבאים:

צד לkipot (ס' פלאו פלאו):

- ספרית גרפיקה SFML. בה השתמשתי כדי לצייר את הטקסטורות במקומות אח הנכון על המסר, ולצייר את הטקסט: יש טקסט במסך התחברות, מסך יצירת המשתמש, ובמשחק הראשי בהודעות למשתמש (ימין למעלה) וטבלת הנוקודות של המשתמשים (שמאלי למעלה).

השתמשתי בSFML גם כדי להציג רעשים כשהשחקן יורה את הרובה שלו, וכדי ליצור את התקשרות בין הלוקו לשרת. יש מחלקות TcpSocketi UdpSocketi שאפשרו זאת.

ספרית מספרים גדולים int_int.cpp::Multiprecision::Boost. השתמשתי בספריה זו כדי לחשב את החזקות הגודלות והמודולו הדרוש בעית יצירת המפתח הסודי של הצפנה בתהילר Diffie Hellman. המפתח שומש בהמשך כדי להצפין הודעות עם AES128.

ספרית הצפנה OpenSSL. השתמשתי בספריה זו כדי להצפין את ההודעות שעוברות בין הלוקו לשרת ובוחרה, עם אלגוריתם AES128.

מודול thread - אפשרות הריצה של פעולות ההකשה לשרת במקביל לצורך המשחק על המסר בזמן אמת.

מודול mutex שאפשר לי לנעול את המשתמשים של האובייקטים של השחקנים האחרים, בזמן שהthead שמחשייב לשרת משנה את המיקום שלהם.

צד שרת (פיתון):

מודול socket - תקשורת עם סוקטים, UDP וTCP

מודול threading - פיצול עבודה השירות לחוטים שיכולים לרוץ ביחד, כמעט במקביל.

מודול cryptography - הצפנה AES

מודול secrets - יצירה מספרים רנדומליים גדולים בusers.db

מודול sqlite3 - עבודה מול מבסד נתונים db

מודול hashlib - גיבוב הסיסמה בעזרת hashing algorithm נפוץ ובתוון בשם SHA256

מודול struct - עיבוד המידע שהлокו שלוח לשרת בזמן המשחק, המידע מועבר מ-+++-במבנה שנקרא struct, המודול זהה מאפשר ליפענחו את הבטים ולהמיר אותם בשורה אחת לכל המשתמשים שמרכיבים את המבנה נתונים UserInfo

מודול math - דרוש בחישוב כיווני היריה של השחקנים, כדי לאמת אם אחת מהיריות פגעה באחד מהשחקנים האחרים. ובמי.

תיאור מודולים מרכזיים, שאני הcenטי:

צד ל Koh (ס' פלאס פלאס):

- מחלקת Player - נוצר אחד ממנו בכל ל Koh, ושמור שם כל המידע על השחקן: מיקום, מהירות, סיבוב ראש, תזמון אנימציות, מצב חיים, טקסטורות, וכמה נקודות שיש לו. יש למחלקה זו פעולה כדי ליצור את הרובה ולדאוג לאנימציה שלו, פעולה כדי להקשיב לשרת ודבר עם השרת ועוד.
- מחלקת Object - בשבייל כל שחקן אחר שנמצא במשחק, נוצר Object. במחלקה זו שמור המצב הכלול של השחקן האחר, האם הוא זז, איפה הוא נמצא האם הוא יורה. כל המידע זהה משמש כSKUורים לפעולה drawObject(const Object& object, float dt).
- מחלקת Client - שם שמור המידע על הsockטים TCP UDP, ושם מוחזקות הפעולות הדרישות כדי לדבר עם השרת ולהציג ולפענה את ההודעות.
- מחלקת map - שמור שם המידע על המפה שבה השחקנים מhalכים, לשם ניגשת הפעולה drawWorld כשהיא מצירפת את המפה.
- מחלקת Toaster - ניכר לפי השם שאני זה שבחר איך לקרוא למחלקות האלה, ושלא תשגו, זהו שם ממשמעתי ביותר; ישן הודעות שkopatz בימין לעללה כדי לעדכן את השחקן ברגע לצב המשחק, לדוגמה אם התחרבות לשרת נכשלה, או שימושו נהרג במהלך המשחק. להודעות האלה קוראים toasts, لكن המחלקה שמנהל אותם ואליה קוראים כדי ליצור toast בפעולת Toaster.toast ()).

צד שרת (פייתן):

- מחלקת Player - באך השרת דרושה הרבה פעות עבורה עם מחלקות, אך זהה המחלקה היחידה שקיים. בשבייל כל Koh שמתחבר, נוצר עצם מסוג Player בשרת והוא נכון לרשימת השחקנים. כשהשרת מקבל הודעה מאחד השחקנים, כתוב שם שהרובה שלו נורה, השרת משתמש בפעולת handle_gun_shot ופעולת handle_gun_shun_at כדי לבדוק אם השחקן מסתכל על שחקן אחר תוך כדי שהוא יורה את הרובה שלו. אם יש שחקן מול נשלחת הודעה לכולם שהייתה פגיעה, כולם מגיבים את הנפגע עם דם על החזה, והשרת מוריד את health(player) בכמות מוגדרת. כמובן שגם החיים יורדים מעבר ל-0, השרת שולח לכולם שהשחקן מת, ועליו ללחוץ על Spacebar כדי לקום מחדש לחיים.

אנימציה של הרובה והיד במשחקך:

```

328
329
330 // draws the hand holding the gun at the bottom of the screen
331 // dt - deltaTime, the time in seconds since the beginning of the last frame
332 void Player::drawGun(float dt)
333 {
334     gun_animation_timer += dt;
335     gun_movement_stopwatch += dt * 7;
336
337     if (moving)
338         hand_move_range = lerp(hand_move_range, max_hand_range, 0.007f);
339     else
340         hand_move_range = lerp(hand_move_range, 0, 0.001f);
341
342     float gun_offset_x = sin(gun_movement_stopwatch) * hand_move_range;
343     float gun_offset_y = 0.3f * cos(gun_movement_stopwatch) * cos(gun_movement_stopwatch) * hand_move_range;
344     gun_offset = { gun_offset_x, gun_offset_y };
345
346     gun_sprite.setPosition(gun_position + gun_offset);
347
348 //cout << "\n";
349 window.draw(gun_sprite);
350
351 // if frame bigger than zero, we animating
352 if (gun_animation_frame && gun_animation_timer >= gun_animation_duration[gun_animation_frame])
353 {
354     gun_animation_frame = (gun_animation_frame + 1) % 5;
355     gun_animation_timer = 0;
356     gun_sprite.setTexture(gun_text[gun_animation_frame]);
357 }
358
359 }
360
361

```

בקוד שמציג, ישנן שתי רמות של אנימציה - התזוזה של יד השחקן והחלפה בין תמונות של הרובה.

התזוזה של יד השחקן:

התזוזה מתבצעת באמצעות שינוי דינמי במיקום היד בהתאם לתנועת השחקן. המיקום משתנה באופן רך ומתמיד בעזרת פונקציית הלרפרפציה (lerp).
עם תנועה והפסקת תנועה, ישנה התאמה של הטווח שבו תבוצע התזוזה, מטור מחשבה על רמת הטבעיות והמראה הכללי של המשחק.

החלפת תמונות של הרובה, בעת ירייה:

בקטע הקוד, יש שימוש במבנה של תמונות (Textures) המשמשות כשלב אינטגרלי באנימציה. עם כל מעבר של מסגרת זמן מסוימת (gun_animation_duration), התמונה מתחלפת, ובכך יוצרת אפקט חלק ומתרמשך של הרובה. זה קורה בעקבות הצורך לשחקן לחווית ראייה שוטפת שמשתנה באופן דינמי, ששומרת על רמה גבוהה של זרימה במהלך המשחק.

בחירה בכמויות מועטה של תמונות:

בחירת התמונות נעשתה על-פי יכולתם ליצור אפקט חזותי יוצא דופן ומעניין. השימוש בפונקציית הסינון והקואינון בחישוב המיקום מזכיר את תנועת היד בצורה יומיומית, מפני שהם מחווריות וגם תנועת היד הטבעית שלנו, קדימה אחורה קדימה אחרת, הינה מהזרירות. הקוד מאפשר עבדה נוחה ומאורגנת עם התמודדות עם פרמטרים שונים כמו זמן וטוויח תנועה. התוצאה היא אנימציה חלקה ומרשימה שמתאימה לצורה טبيعית לזרימת המשחק.



מפת המשחקן:

המפה במשחק יוצגה באמצעות תמונה המורכבת מפיקסלים, כאשר כל פיקסל מייצג ארכח קרקע במשחק. התמונה משתמשת כ-map Tilemap בה מגוון של ערכים ביןaries - שחור (0) ולבן (1), המייצגים שטח נגיעה וקיר בהתאם.

טעינת המפה:

טעינת המפה נעשית בעזרת קובץ תמונה בפורמט שתומך במידע על פיקסלים (למשל PNG). בקוד, אנו טוענים את התמונה מקובץ ומשתמשים בה כדי ליצור את המפה.

קביעת גודל המפה:

הקוד קובע את גודל המפה על פי גודל התמונה באמצעות פעולות קביעת רוחב וגובה.

יצירת מערך המייצג את המפה:

ונוצר מערך דינامي בגודל המפה, שבו כל תא מייצג פיקסל בתמונה. הערכים במערך יכולים להיות 0 או 1 בהתאם לצבע של הפיקסל בתמונה (שחור או לבן).

כך המפה מיוצגת על ידי מפה שניית בזיכרון להחליף לכל מפה אחרת, בעזרת החלפת התמונה. השחקן יכול לזרז תוך התיקות לתאים במערך המייצג את המפה. המפה באמצעות קידוד זה יכולה להכיל מגוון רחב של סוגי תחזוקות ומכשורים במשחק, ותאפשר ליצור חוויות משחק שונה בכל פעם שהמשתמש משחק.

```

7     sf::Image map_texture;
8     map_texture.loadFromFile("map.png");
9
10    width = map_texture.getSize().x;
11    height = map_texture.getSize().y;
12
13
14    ▶ data = new int[width * height];
15
16    for(int i = 0; i < width; i++)
17        □    for (int j = 0; j < height; j++)
18        {
19            ▶        if (map_texture.getPixel(i, j).r == 255)
20                data[i + j * width] = 1;
21            else
22                data[i + j * width] = 0;
23        }
24
25

```

```

    player.handleKeys(dt);

    // Graphics
    window.clear(sf::Color::Red);

    map.drawSky(); // Sky

    player.shootRays(); // World

    player.drawGun(dt); // Gun

    player.drawCrosshair(); // Crosshair

    window.display(); // Render to screen

    frame_count++;
}

```

לולאה ראשית:

הlolala הראשית במשחק היא הלולאה המרכזית שבה תרוץ כל הלוגיקה של המשחק, היא תקבע את הקלט מהמשתמש, ותציג את כל התצוגה על המסך. בקטע הקוד שסופק, ישנו מספר פעולות חשובות שמתרוצצות בכל לולאה. הלולאה נקראת פעמי אחת בכל פריטים.

הfonקציה handleKeys של אובייקט השחקן נקראת. פונקציה זו אחראית לטיפול بكلטיים מהמשתמש, קלטים אלו כוללים את התנועה של השחקן, כיוון הרזיה, ופעולות אחרות כמו ירייה ברובה.

עכשו לאחר קבלת הקלט מתחילה הציור למסך. בקריאה ל-clear, הfonקציה מצירמת את כל המסך בצבע אחד, במקרה זה, צבע אדום. המחיקה זו מתחילה את המסך ועכשו ניתן לצייר את הפריטים החדש.

עם הסקירה הנומוכה שלהם, רקע השמיים הוא חשוב כיוון שהוא מציגים את תמונה תלת-מימדית. drawSky היא פונקציה שמטרתה לצייר את הרקע של השמיים. במשחקי רובה (Raycasting),

shooting היא פונקציה שמטרתה לצייר את התמונה התלת-מימדית שנראית בעין השחקן באמצעות קרייה ל-"קראיים" מהמשחקן לכל כיוונו, בדיק כמו יש במשחקי רובה הקלאסיים.

drawGun נקראות הfonקציות שהפורטוגוניסט השחקן משתמש בהן כדי לצייר את רובו, כולל את התמונה הדו-מימדית של הרובה שלו.

drawCrosshair מטרתה לצייר את האיקון של הכוונת (Crosshair), הפלוס שמצין לשחקן את מקום המטרה הנוכחי.

לולאה זו נקרת שוב ושוב עד שהמשחק מגיע לסיוםו או שעוברים מסך הבית.

אלגוריתם התזוזה ב-`map::move` לפי זווית השחקן

הfonקציה `Player::move` מקבלת שני פרמטרים: `angle_offset` שמיועד לשינוי בזווית התנועה, ו-`dt` המיצג את הזמן שעבר מאז סיום ציור הפריים האחרון, הזמן שבין שלבי העడון עצמו.

תנוועה:

הfonקציה מתחילה על ידי חישוב קצב התזוזה הנוכחי, משתמשת בהתאם למהירות רגילה או מהירה בהתאם לתנאי הריצה (`running`). התנוועה נעשית באמצעות הzzת השחקן לפי הזווית שלו בהתאם לתזוזה הנוכחיות ולזמן.

שקלות האзор הקרוב לשחקן:

לאחר התזוזה הפטנציאלית, הfonקציה מחשבת את האזור שסביב השחקן ובודקת אם קרו שם התנגשויות עם הקירות שבמפה. היא משתמשת בסטראקט `oi` (המיצגת נקודה במרחב עם ערכיהם שלמים) ובfonקציות `floor` ו-`min` כדי למצוא את התאים המקומיים שסביב השחקן.

פור על האזור:

הלוואה הבאה עוברת על כל התאים באזורי המחשב ובודקת אם השחקן יתקע בתא מסוים (תא עם ערך של 1, המיצג קיר). אם כן, היא מחשבת את הנקודה היכי קרובה בתא למקום שבו השחקן יגיע לו, ובודקת את ההתנגשות ומתקן את המיקום אם התנגשות נמצאת מתרחשת.

בסוף כל התהליכיים, מעדכנת הfonקציה את המיקום החדש של השחקן על פי התנוועה וההתנגשויות שנגרמו על ידי הקירות.

```

83 void Player::move(float angle_offset, float dt)
84 {
85     :
86
87     float current_speed = speed;
88     if (running)
89         current_speed *= run_multiplier;
90
91     v2f potential_position;
92     potential_position.x = position.x + current_speed * run_multiplier * dt * cos(rotation_x - angle_offset);
93     potential_position.y = position.y + current_speed * run_multiplier * dt * sin(rotation_x - angle_offset);
94
95     v2i ones(1, 1);
96     v2i predicted_cell = v2i(floor(potential_position.x), floor(potential_position.y));
97
98     v2i area_tl = min((v2i)position, predicted_cell);
99     area_tl = max({0, 0}, area_tl - ones); // clamp
100
101    v2i area_br = max((v2i)position, predicted_cell);
102    area_br = min({map.width, map.height}, area_br + ones); // clamp
103
104
105    for (int cell_x = area_tl.x; cell_x <= area_br.x; cell_x++)
106    {
107        for (int cell_y = area_tl.y; cell_y <= area_br.y; cell_y++)
108        {
109
110            // is wall
111            if (map.getCell(cell_x, cell_y) == 1)
112            {
113                v2f nearest_point;
114                nearest_point.x = std::max(float(cell_x), std::min(potential_position.x, float(cell_x + 1)));
115                nearest_point.y = std::max(float(cell_y), std::min(potential_position.y, float(cell_y + 1)));
116
117                v2f ray_to_nearest = nearest_point - potential_position;
118
119                float overlap = body_radius - mag(ray_to_nearest);
120
121                if (std::isnan(overlap)) overlap = 0;
122
123                if (overlap > 0)
124                {
125                    potential_position -= norm(ray_to_nearest) * overlap;
126                }
127            }
128        }
129    }
130
131    position = potential_position;
132
133 }
134 }
```

לייעד קורן - Raycaster

```
52     //gun shot
53     bool gun_shot;
54     sf::Texture damage_overlay_tex;
55     sf::Sprite damage_overlay_sprite;
56     float current_damage_opacity;
57     float max_damage_opacity = 130;
58     string killer_name;
59
60
61     // movement
62     v2f position;
63     float speed = 2.0f;
64     bool running = false, crouching = false;
65     bool moving = false, moving_forward;
66     float run_multiplier = 1.75f, crouch_multiplier = 0.5f;
67
68     // orientation
69     float rotation_x = -3.169f;
70     float rotation_y = -0.56f;
71     float mouse_sensitivity = 0.05f;
72     float fov_y = 0.7f;
73     float fov_x = 1.22173f; // 70 degrees
74
75     // map
76     float body_radius = 0.4f;
77
78     //sound
79     sf::SoundBuffer gunshot_buffer, gunclick_buffer;
80     sf::Sound gun_sound, click_sound;
81
82     //font
83     sf::Font nametag_font, bold_font, deathscreen_font;
84
85     //debug
86     float debug_float = 0;
87
88     int received_events_size = 0;
89     char received_events[128];
90
91     int score = 0;
92
93     //server
94
95     void updateServer();
96     void listenToServer();
97     void handleEvents(char* events, int event_count);
98     Client::PlayerInfo getPlayerInfo();
99
100    Object* getObject(int id);
101    Object* getAnyObject();
102    void handle_shooting_victim(int victim_id, int shooter_id);
103    void handle_killing(int killer_id, int victim_id);
104    void getKilled(const string& killer_name);
105    void respawn();
106
107    void addToLeaderboard(int player_id, int score, const string& username);
108    void updateLeaderboard(int player_id);
109
110    string getUsername(int id);
```

תיאור מחלקת השחקן:

מחלקה Player במשחק מהווה את הליבה של דמות השחקן ומספקת את הfonקציות והמשתנים הדרושים לניהול והפעלת השחקן במהלך התלת-ממד. להלן סקירה קצרה על התוכן של המחלקה.

משתנים פרטיים:

המחלקה מכילה מספר משתנים פרטיים המשמשים לניהול פרטיים מקומיים של השחקן, כולל מיקום, זווית רוטציה, מידע על התנועה והtekסטורות של הקירות והרובה.

fonקציות עיבוד קלט:

ישנן Fonקציות המקובלות וublisher את הקלט המשמש, כגון handleKeys שמתיחס לקלט מקלדת וfonkציה rotateHead שמנהל את זווית הראש בהתאם לתנועה העבר.

fonקציות ניהול תנועה:

ישנן Fonקציות שמתיקנות את תנועת השחקן, כמו move שמתיחס לפועלות תנועה וfonkציה shootRays שמבצעת את הדרך שבה השחקן מתקשר עם הסביבה.

fonקציות עיבוד גרפיות:

ישנן Fonקציות העוסקות בעיבוד והציגת גרפיקה, כמו drawGun שמצוירת את הנשק של השחקן ו-air drawCrosshair שמצוירת תמונה חצייה על המסך.

fonקציות פנימיות:

ישנן Fonkציות פנימיות המשמשות לצורך פעולות פנימיות במחלקה, כמו shootGun שמטפלת ביריות השחקן.

fonקציות אחרות:

ישנן Fonkציות אחרות המתיחסות לפחות נסיפות, כמו loadTextures שמעבינה את הטקסטורות של השחקן. מהלך המחלקה ישתמש כתפריט מרכזי לקובץ העיקרי שלך, יכול לשרת כבסיס להוספה יכולות וfonקציות נוספות בעת פיתוח המשחק.

```
Player(int x, int y, sf::RenderWindow& window, Toaster& toaster);
void setFocus(bool focus);
void handleKeys(float dt);
void rotateHead(int delta_x, int delta_y, float dt);
void move(float angle_offset, float dt);

void shootRays(HitInfo*& hits);
void drawWorld(HitInfo*& hits, float dt);
void drawColumn(int x, const Player::HitInfo& hit_info);
Player::HitInfo shootRay(float angle_offset);

void drawObject(Object& object, float dt);
void drawGun(float dt);
void drawCrosshair(float dt);
void drawDeathScreen(float dt);

void shootGun(bool left_click);
void getShot(int shooter_id);

void loadSFX();
void loadTextures();

void quitGame();

//server
void updateServer();
void listenToServer();
void handleEvents(char* events, int event_count);
Client::PlayerInfo getPlayerInfo();

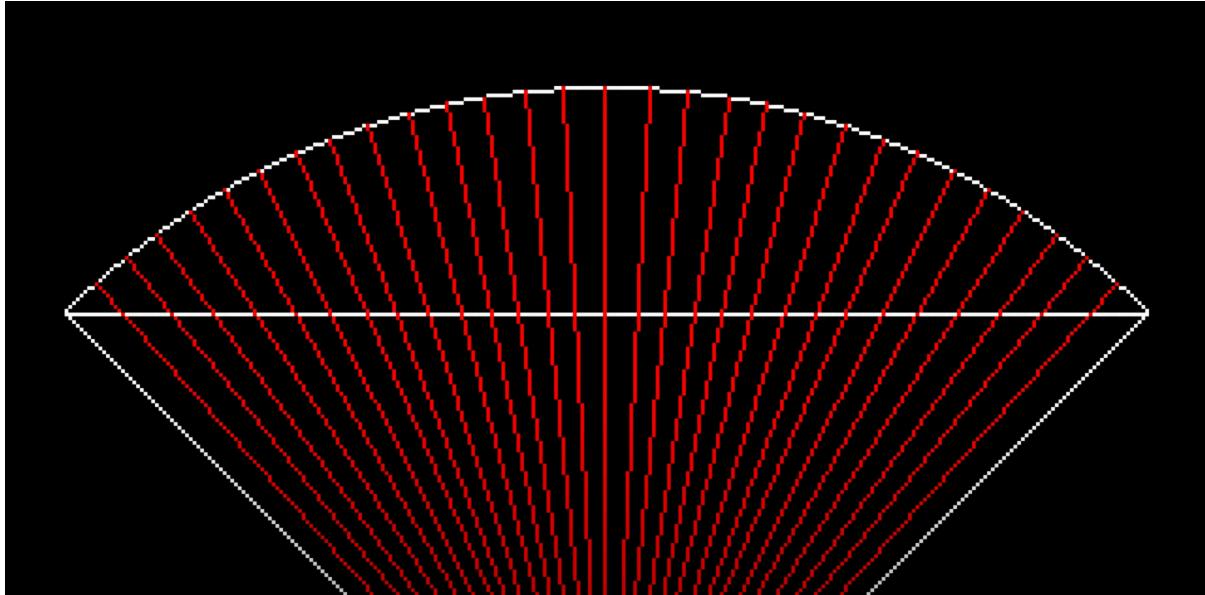
Object* getObject(int id);
Object* getAnyObject();
void handle_shooting_victim(int victim_id, int shooter_id);
void handle_killing(int killer_id, int victim_id);
void getKilled(const string& killer_name);
void respawn();

void addToLeaderboard(int player_id, int score, const string& username);
void updateLeaderboard(int player_id);

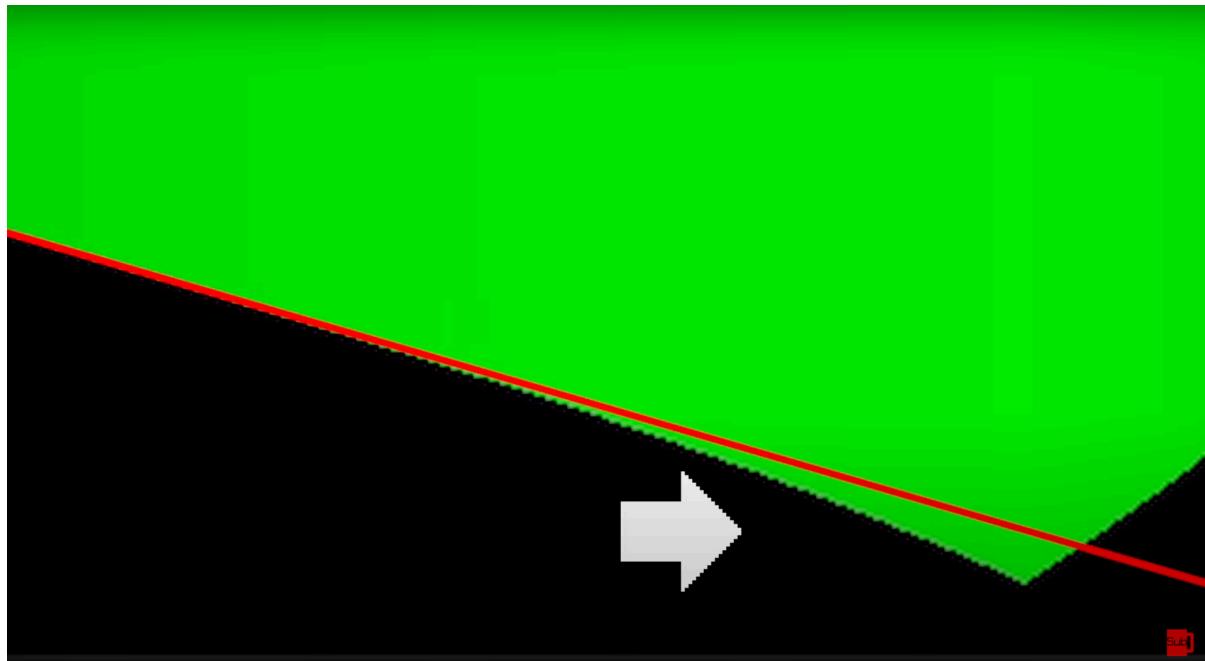
string getUsername(int id);
```

בעיה שנתקلت בה במהלך התוכנות:

עלוי למצא את הזווית המדויקת לשЛОח את הקרנינים החוצה מעיני המשתמש. במחשבה ראשונה אפשר להתחיל מ민וס $2/\text{fov}$ ולעלות בקצב קבוע אחדות עד $2/\text{fov}$, כלומר עם הבדל זהה בין כל זווית לזוית. התמונה מציגה את המצב זהה:

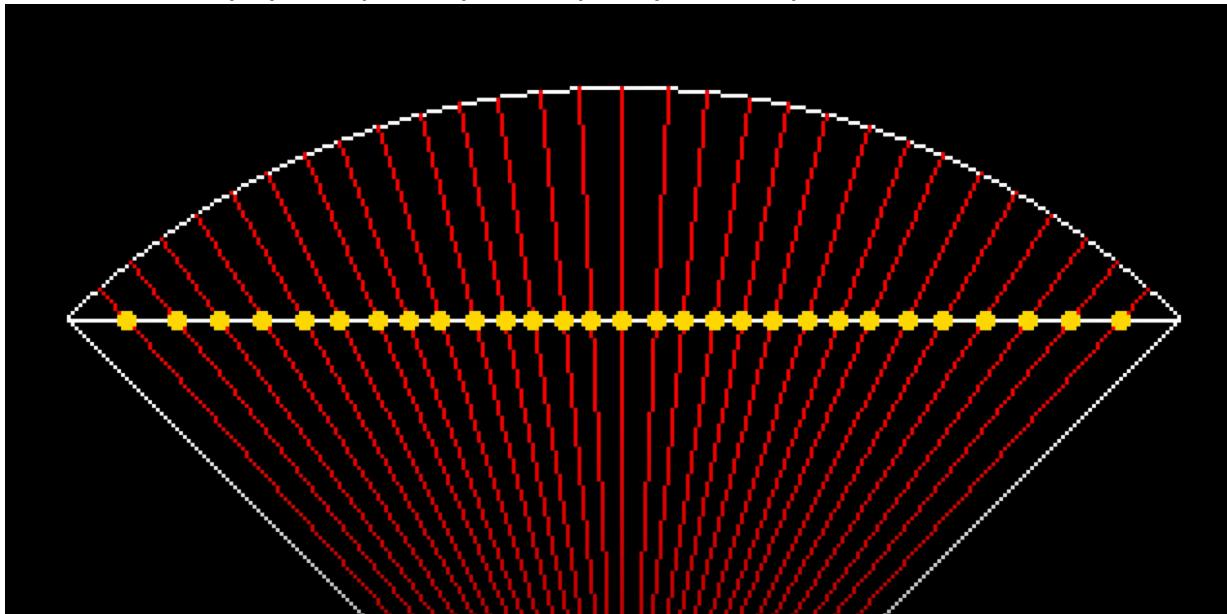


בתמונה, נקודת חיתוך של כל קיר עם הלבן מייצגת נקודת הקיר שהשחקן רואה. הבעיה היא שהמרחק בין כל נקודת חיתוך משתנה ככל שהקרנינים מתקרבים לקצוות. דבר זה גורם לעביה ויזואלית שנראית כך:



פתרון הבעיה:

כדי לפתור את הבעיה, علينا למצוא זווית מתאימה לפני המיקום האופקי של הפיקסלים שאנו חישב. זווית מתאימה יתנו מרחוקים זהים בין כל נקודות החיתוך עם הקיר הלבן, כר:



הבחן במדרייך שלח קרניים עם קפיצות זווית שוויה, ולפי פונקציית המרת מצא את המיקום על המסך שלו לצייר את הקרן. כתוצאה לכך הוא הגיע למצב שיש חורים על המסך במקומות שבמקרה לא הגיעו אליוים קרן, והוא היה צריך להשלים את החורים עם הרבה קוד לא יעיל. אני חשבתי על פתרון יותר טוב.

למה לשולח קרניים שגויות, ולשים אותם במיקום הנכון על המסך? לא נגיע לכל המיקומות במסך. אם יש לנו פונקציית המרת בין זווית למיקום נכון על המסך, אנחנו יכולים להפוך את הפונקציה כך שהיא תעתן זווית מדויקת לפי מיקום על המסך. כך נוכל לעבור על כל הטוירם במסך, ובסביל כל טור נדע לאן לשולח את הקרן כך שהמרחק בין נקודות החיתוך של הקרניים עם מישור הקיר יהיה זהה.

פונקציה מהמדרייך:

$$\text{Screen_Pos} = 0.5 * \tan(\alpha) / \tan(\text{fov}/2)$$

$$\tan^{-1}(2 * \text{Screen_Pos} * \tan(\text{fov}/2)) = \alpha$$

הfonktsiya she ani meshutash ba

בעזרת אלגברה פשוטה של כיתה י', לקחנו את הפונקציה הלא שימושית מהמדרייך והגענו לביטוי עיל שմביא לנו את הזווית לפני המיקום על המסך.

רפלקציה

לקח המון זמן להכין את הפרויקט זהה, ואני מרגיש את ההשפעה שהייתה לו על תחומי הידע שאני מומחה בהם. הפרויקט שיפר לי את הבנה בפרוטוקולים של החלפת מפתחות, החשיבות שלהם, הדרך שבה הם פועלים וגם הדרך שבה ממשיכים אותם. למדתי על המתמטיקה החדשאה כדי לציר עולם תלת מימדי שלהם ואמין, בנוסף לתלת מימדיות בעלות אণימציה שmagיבת בזמן אמיתי, והכל בעזרת מבנים וטקסטורות בלבד. למדתי איך לישם תקשורת TCP וUDP במקום מתאימים לכל אחד.

חשוב לציין שמעל חצי מהפרויקט (כל צד הלוקו) נעשה בשפת C++, שפה מאוד "נמנוה" שמחייבת הבנה عمוקה של כל פעולה שנתקראת. בפייטון אפשר לעשות `import` לספריה ולתת לה לדאוג לפועלות הלא מעניינות, אבל בC++ חייב להבין ולישם את הדברים היכי קטנים, כמו העבודה עם המספרים הראשוניים בHellman Diffie, אלוקציית הזיכרון בשבייל תובורת המידע UDP וTCP, והמיקום על המסך שבו צריך לציר את הקיר או את השחקנים האחרים.

תמיד התעניינתי בתכונות ברמה נמנוה, מאז פרויקט האסמבלי שעשינו בכיתה י' בмагמה, שגם שם עשיתי פרויקט שהיה מרשים במיוחד מנקודת ה视圖 שלו.

תמיד כשמדברים על C++ מזהירים מפני דיליפות זיכרון, וכשניסיתי לגרום לאחת לקרות, לא הצליחתי. הcompiler תמיד תיקן את הדיליפה ולא הצלחתי לראות דיליפה של עצמי. אחרי כמה ניסיונות, יתרתי. במהלך תכונות הפרויקט הזה, אחרי הוספת התעבורה בראשת, שמתי לב שהמשחק קורס בערך אחרי 15 שניות שהוא רץ. מוזר, חבשתי, ופתרתי את Task Manager כדי לראות אם יש בעיה שkopatzת לעין. שראיתי שהפרויקט צריך מעל 100,000 מגהבייטס של זיכרון כל שנייה. חיכתי לאור העבודה שנתקלה. בתופעה המפורמת שניסיתי בעבר לישם, ותיקנתי את הבעיה עם הוספת שורה אחת הקוד.

הגראפיקה של המשחק היא החלק היכי מרשים ומשמעותי בו. עשיתי המון מחקר באינטראקט בוגע בדרך היכי קלה והיכי יפה לישם את האפקט Raycasting, והגעתי למסקנה Raycasting היא דרך טובה ומשמעותית לישם את המראה התלת מימדי שחייבת. עקבתי אחר מדריכים וחישבתי הרבה נוסחאות עצמי, ולאחר שימוש של הכל בC++, התוצאה היא משחק שנראה לעין כתלת מימדי, עם יכולת לנوع מרחב בקלות עם המקלדת, ולסובב את הראש עם העכבר, ממש כמו במקרים מפורטים היום, כמו Fortnite או Doom.

כרגע שתפלו יכולות C++ שלי, ניהול הזיכרון, ודרך בניית הפרויקט של התקשרות של המשחק הוכחתו לעצמי שאני יכול להתמודד עם אתגרים קשים וגם בדרך עיליה מספיק כדי לתמוך במסpiel חלק.

בכיתבת השרת, שרצ בפייטון, למדתי על הדרך היכי טובה לעבוד עם מחלקות והאיזון המושלם שבין פעולות פנימיות לבין פעולות חיצונית. דבר שעזר לי בכתיבת הפייטון הוא פירוט של סוג המשתנה שאני מבקש בפעולות. דבר זה הפך את הקוד שלי לクリיא יותר והוא לי הרבה יותר קל וכיף לעבוד איתו, כדי לבדוק איזה סוג משתנה כל משתנה מייצג, כמו בשפות שאני אוהב, C++, C#.

ביבליוגרפיה:

Vandevenne, L. (2019). *Raycasting*. Lode's Computer Graphics Tutorial.
<https://lodev.org/cgtutor/raycasting.html#Performance>

3DSage. (2020). *Make Your Own Raycaster Part 1*. Youtube.
<https://www.youtube.com/watch?v=qYRrGTC7GtA>

javidx9. (2021). *Super Fast Ray Casting in Tiled Worlds using DDA*. Youtube.
<https://www.youtube.com/watch?v=NbSee-XM7WA>

Kofybrek. (2021). *Making my First RAYCASTING Game in C++ - SFML Gamedev - Devlog 1*. Youtube. <https://www.youtube.com/watch?v=LUYxLjic0Bc>

Kofybrek. (2023). *I Used RAYCASTING to Make a HORROR Game in C++ - SFML Gamedev - Devlog 2*. Youtube. <https://www.youtube.com/watch?v=OpIS1zoz6fU>

Shirley, P. Black, T. Hollasch, S. (2024). *Ray Tracing in One Weekend*.
<https://raytracing.github.io/books/RayTracingInOneWeekend.html>

נופחים:

בעמודים הבאים ניתן לראות את כל הקוד שלuproject שלי. תחילה את קוד הפיתון של השרת
ואז הקוד בC++ של הלקוח.