

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:

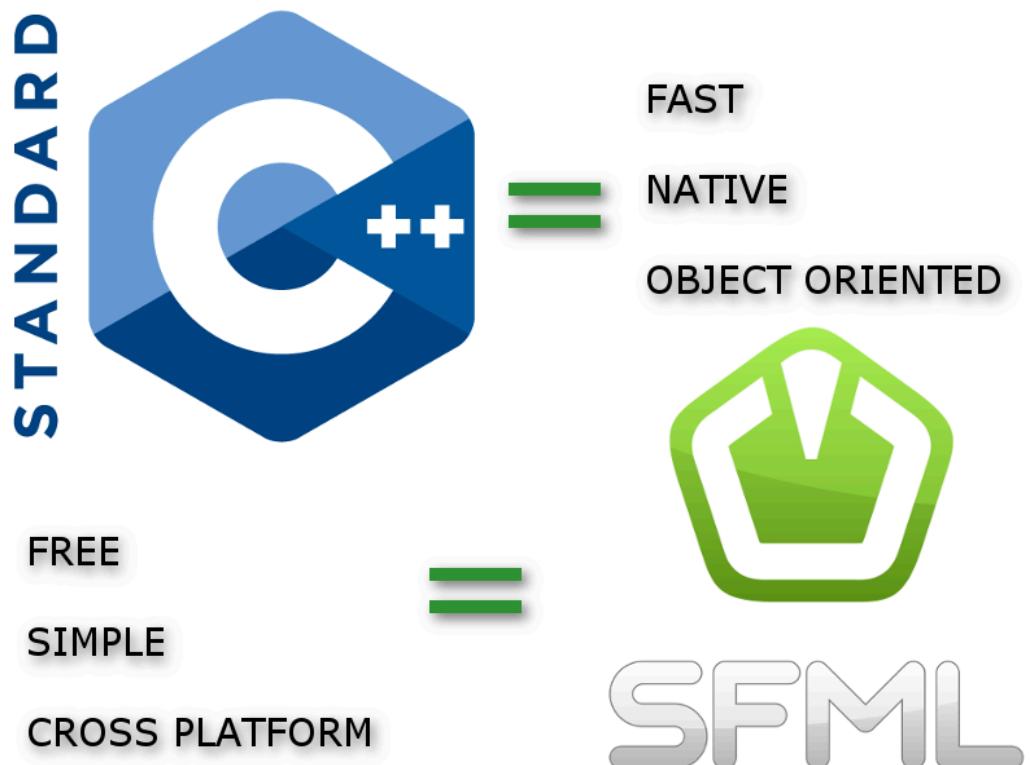
הפרויקט אינו בא לענות על שום בעיה קיימת אלא ליצור משחק מהנה, ישנו משחק דומה בשםWolfenstein 3D או Doom או wolfenstein 3D אשר לא ידוע לי על משחק ספציפי שכזה שכלל גם אונליין.

המשחק משנות ה-90 בשם 3d Wolfenstein



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.



```

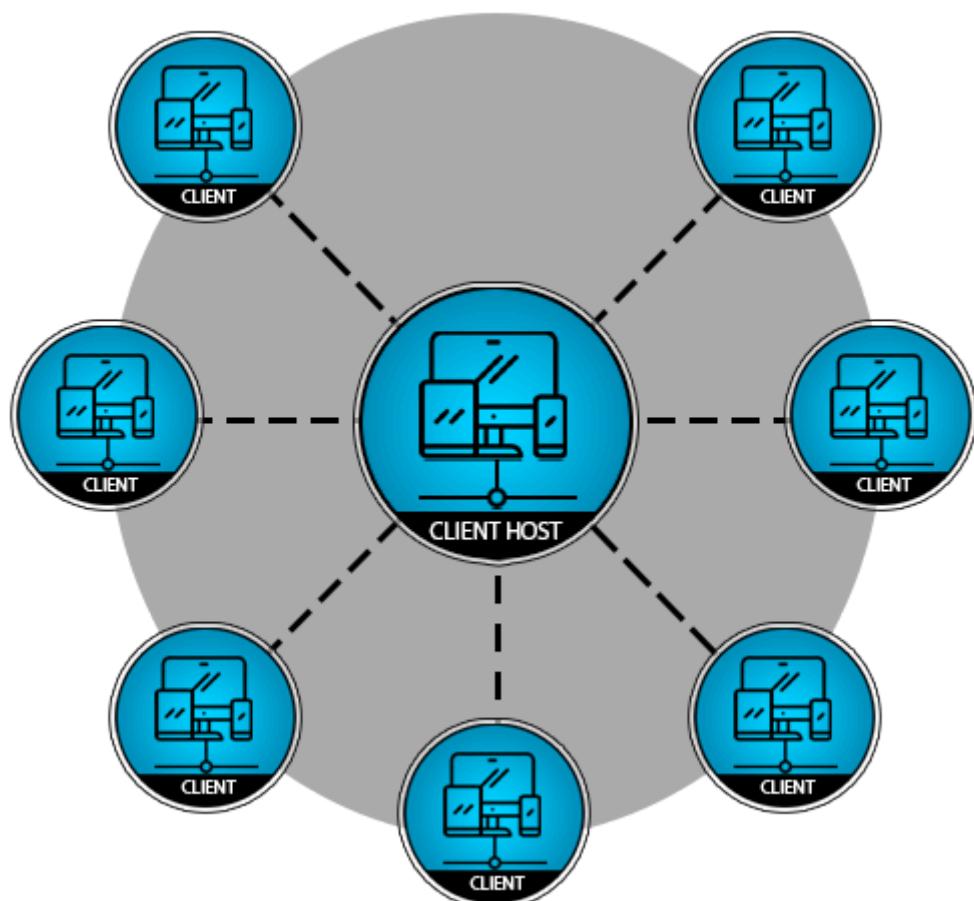
3
4
5     #include <SFML/Graphics.hpp>
6     #include <SFML/Network.hpp>
7
8
  
```

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 שתudging 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 הנחישים והשרת יבודק האם הנחישים נכונים). בסוף כל תור המשתמש יקבל ניקוד על פי זמן הנחיש 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" תציג מספר יכולות מרכזיות:

1. תצוגה תלת-מימדית מרהייבת:

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

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:

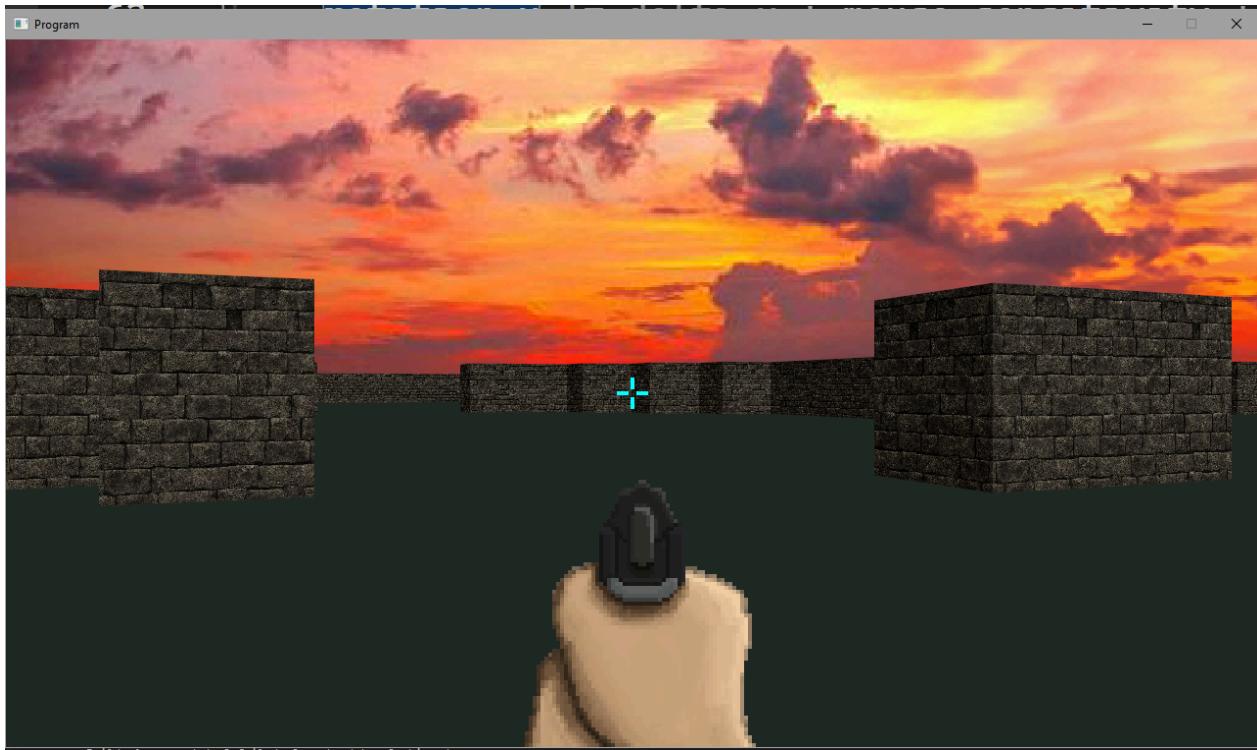
יתבצעו בדיקות תקינות משתמשות וקפדיות, כולל בדיקת קלט משתמש, להבטחת תקינות, יציבות ותפקה גבוהה.

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 נומrica כל כר כמו C++.

לייעד קורן - Raycaster

תמונה של המצב הנוכחי של המשחק

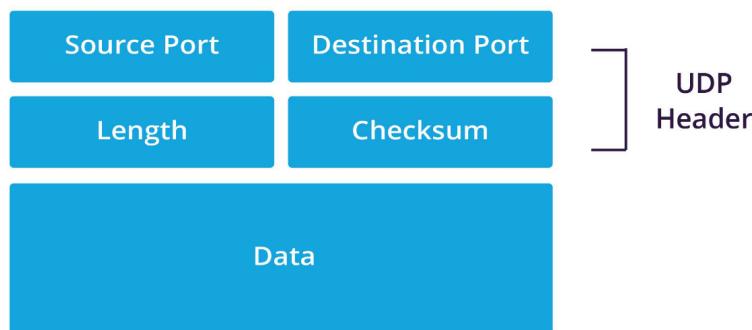


הסיכונים בפרויקט והדריכים להתמודדות איתם:

הפרויקט "Raycaster תלת-מימדי" נתקל בסיכוןים 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 היריה 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 מעמיקות ישפרו את יכולת האיתור ותיקונים של בעיות ובאגים, למרות שהפיתוח עלול להתקדם בצורה 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):

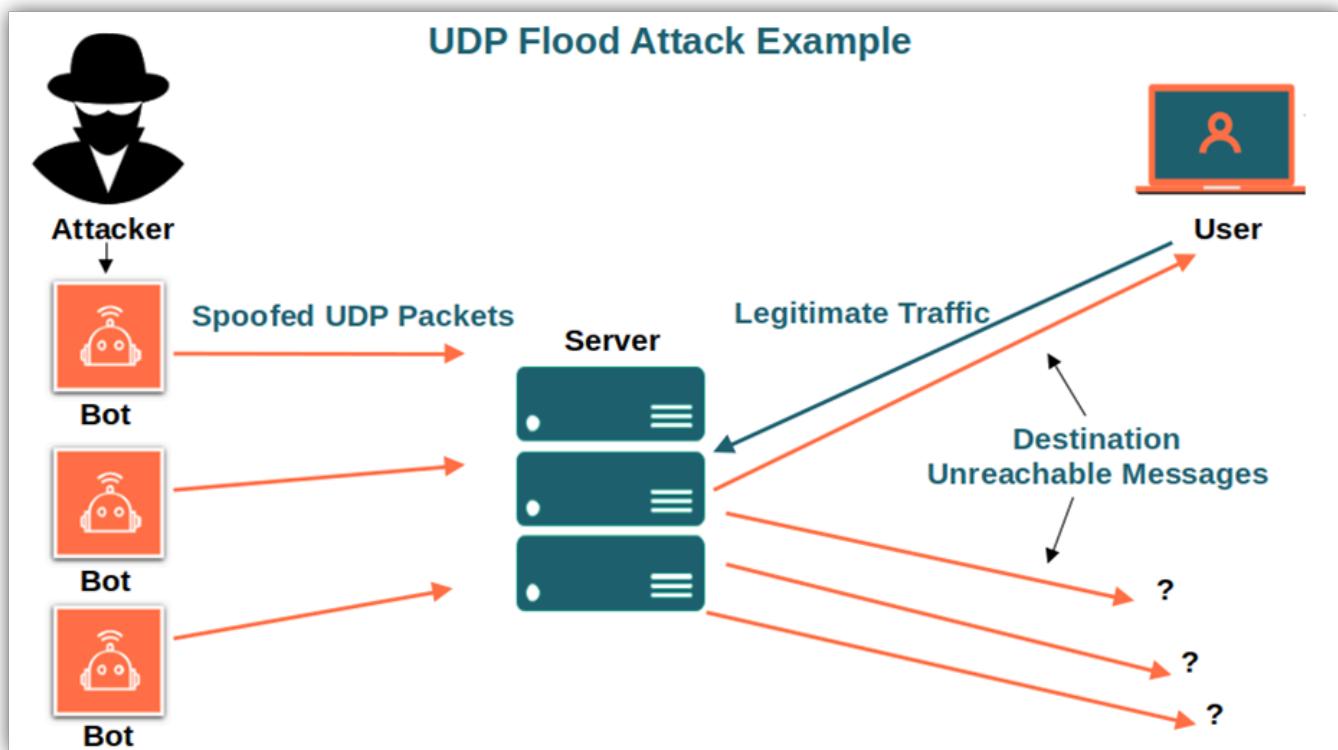
בפרויקט "תלת-מיידי", נזהה איוםים וחולשות בתחום התקשרות, בעיקר בהקשר של אבטחת המידע:

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 שהמיידע שנשלח באמצעות UDP יכול להיבדק באובדן או בדילול בכתובת, וכך נדרש תכנון מיוחד לบทיח 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 שרת, 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 במסpiel.

על מנת להתמודד עם חולשות והאיומים הללו, נדרש פיתוח ייציב, מחקר בטכנולוגיות אבטחה, והטמעת מנגנוןים שישפקו מגן אל מול האיומים האפשריים שהוצעו לעיל.

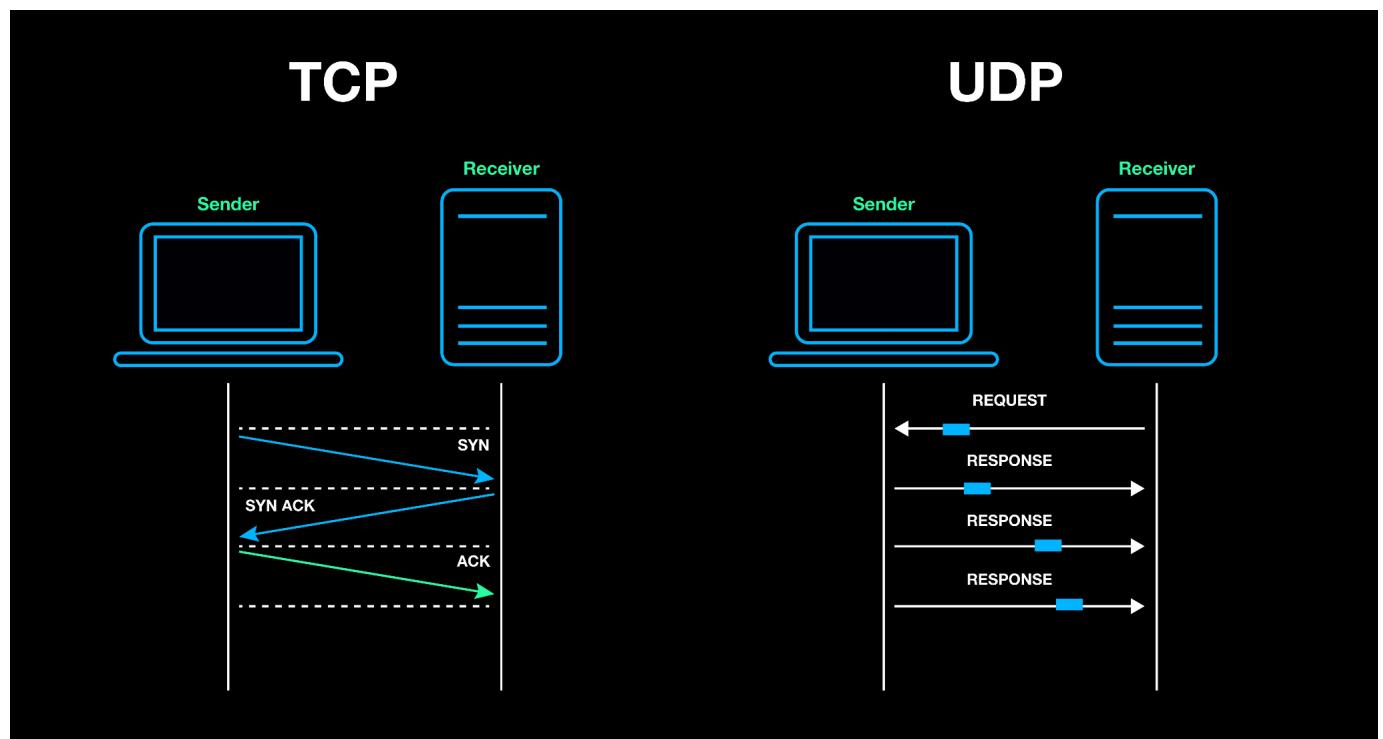


פירוט בדיקות:

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 שניית לעצמך
2/11/23	הוספת אקדח ואנימציות בעת ירייה
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 בקשר לשחקן
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.

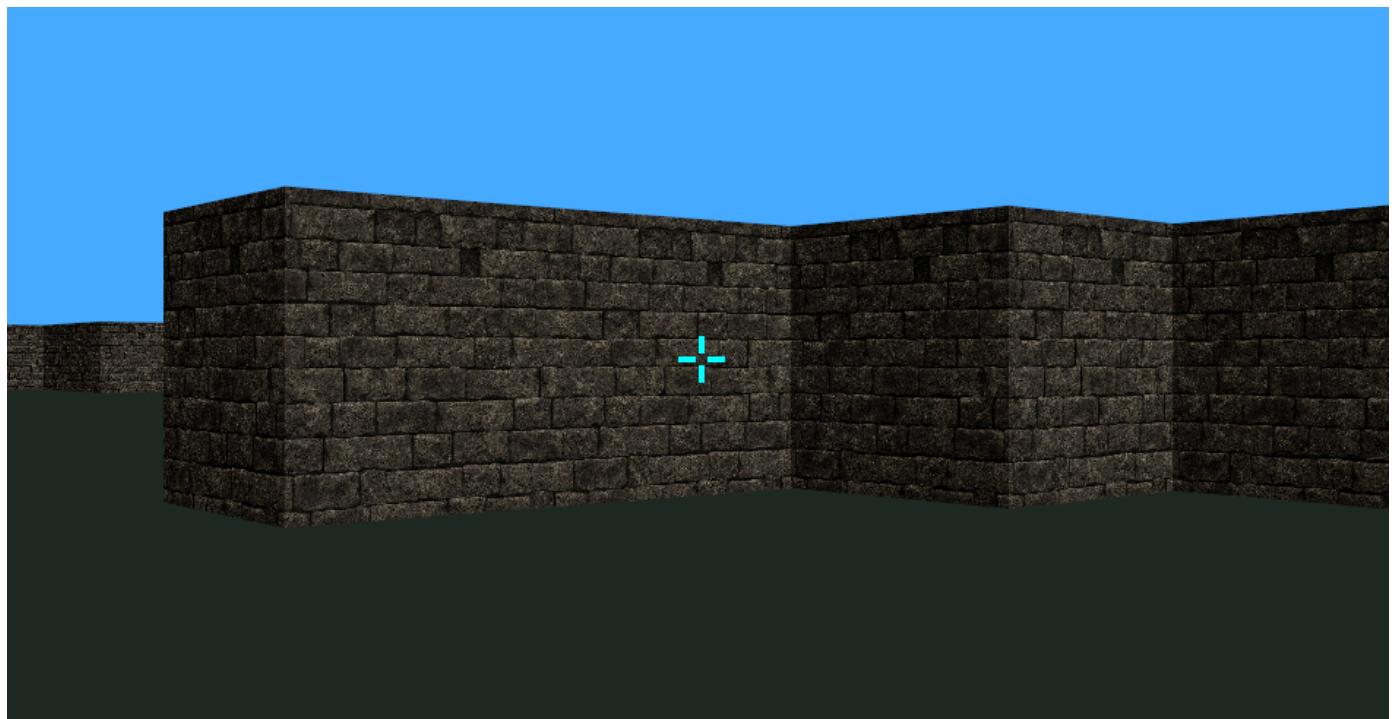
3 שם היכולת: ניהול משחק
מהות יכולת: תהליכי האחראי על ניהול כל חדר משחק.

אוסף יכולות דרישות:

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

אובייקטים נחוצים: הצפנה ופענוח, תהליכיונים.

תמונה התקדמות מההתחלת של הכנת המנוע הגרפי:

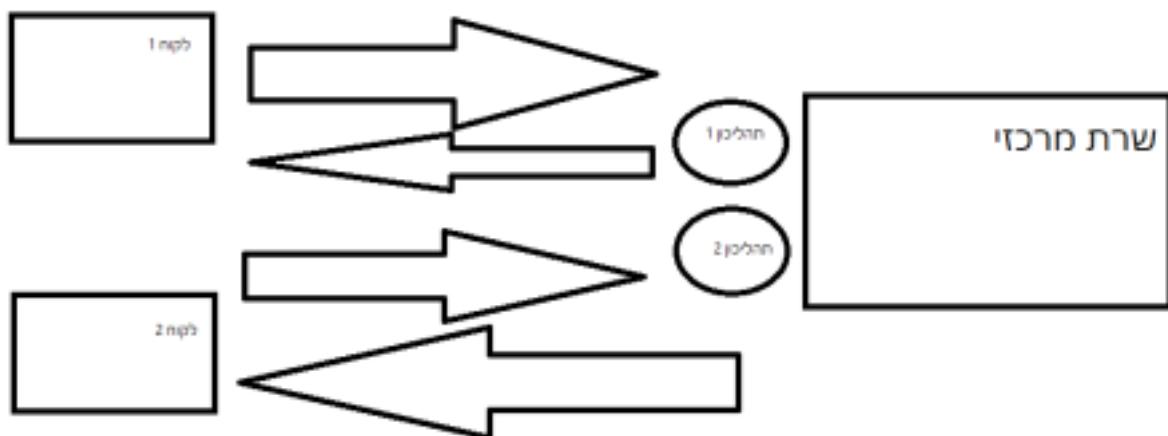


מבנה

• 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.

- ס. 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 וABBYYTEXT, 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. התהיליך 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 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:

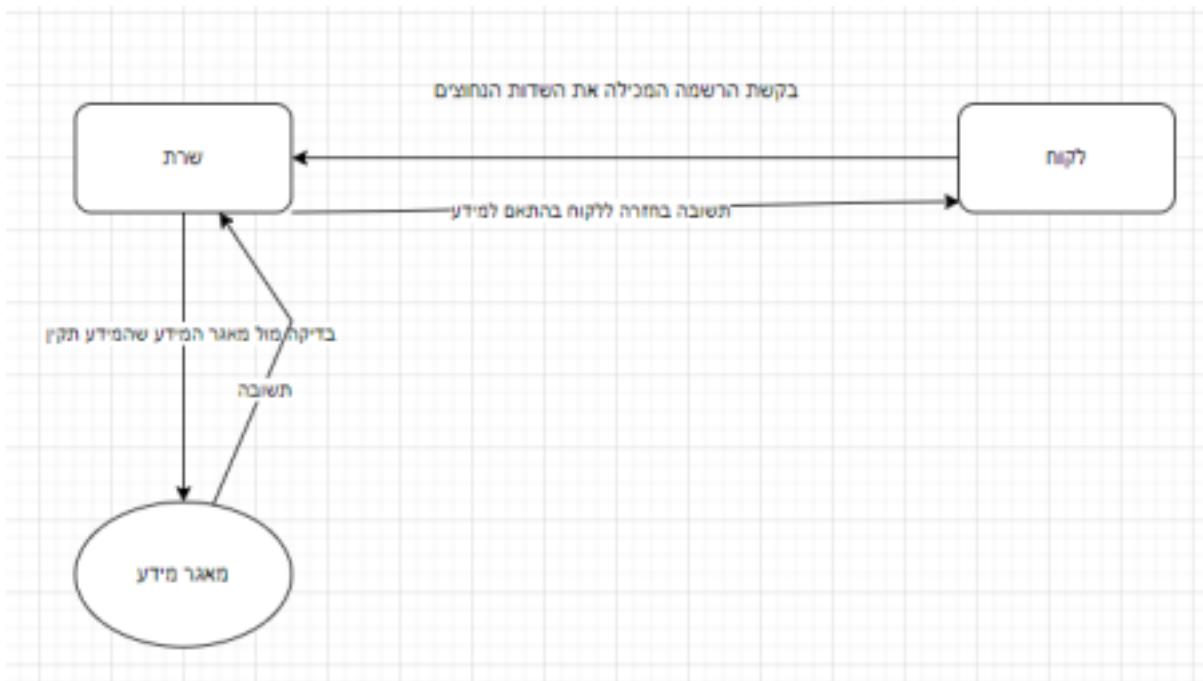
כאשר צד הלוקוח קורט, שום דבר לא נהרס מצד השרת. זאת מפני שהתקשרות בין הצדדים במשחק קורית בעזרת UDP וUDP אינו פרוטוקול מבוסס קשור. התוצאה היחידה של ההתנטקות היא שהשרת מפסיק לקבל הודעות מהлокוח.

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.

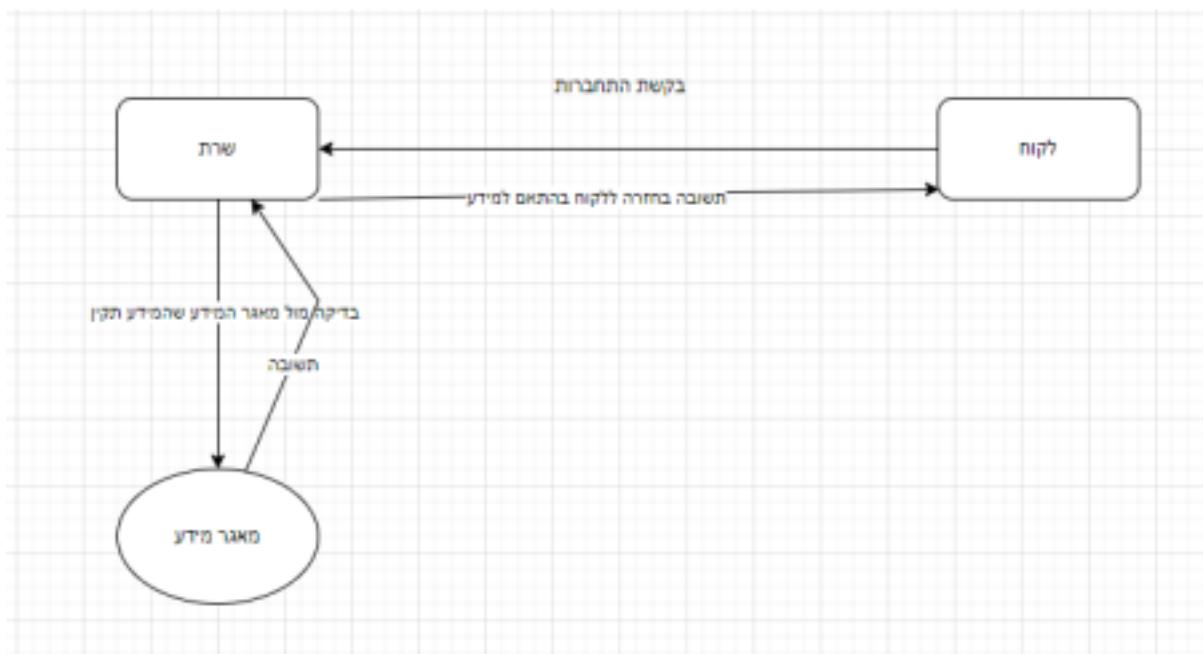
לייד קורן - Raycaster

תיאור זרימת המידע במערכת:

שלב ההרשמה:

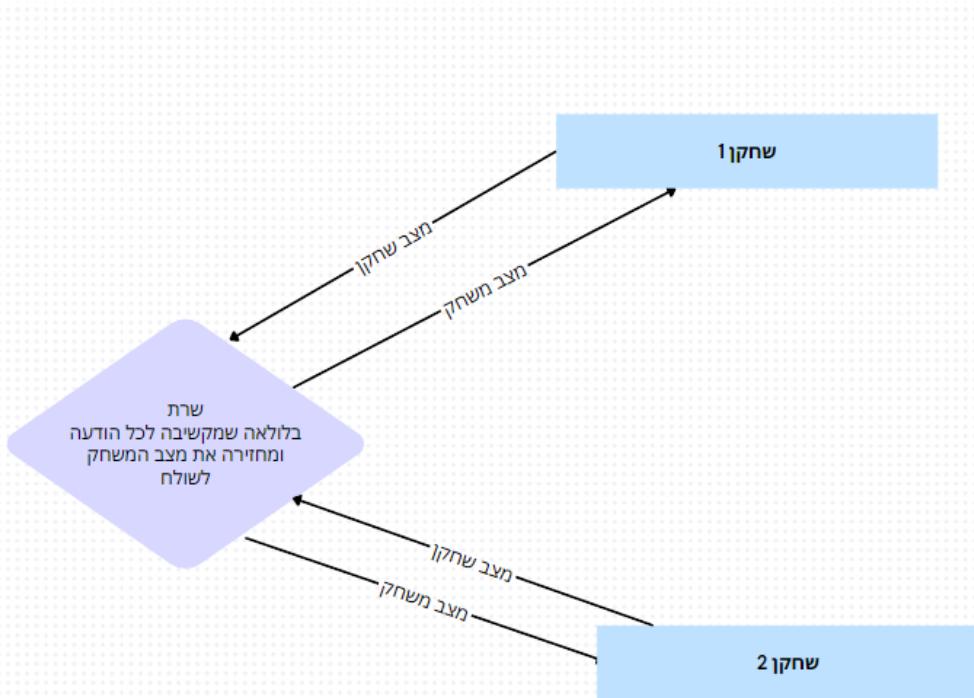


שלב הלוגין:



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 ככלבן 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
253     // drawing
254     sprite.setTextureRect(sf::IntRect(
255         v2i(hit_info.texture_x * wall_txs[0].getSize().x, 0), v2i(1, wall_txs[0].getSize().y)
256     ));
257     sprite.setScale(1, len / wall_txs[0].getSize().y);
258     sprite.setPosition(x, map.floor_level - len / 2);
259     window.draw(sprite);
260
261
262     angle_offset += step;
263 }
264 }
```

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, בקורס 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 הפעיל בשכבות התעבורה של מודל OSI/IP 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 לקратות מהר יותר.

מהירות קבלת ההודעות בזמן המשחק היא קריטית, היות 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 ורק 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 וمضחאים כל הודיעו בעזרתו אלגוריתם AES128.

בעת יצירת משתמש, שם המשתמש והסיסמה עוברים קר, 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 מצד הלקוח כדי לשלוט על ממסד הנתונים). עשייתי כמה פעולות כדי למנוע התקפה זו:

- הנתונים של הלקוח עוברים לממסד 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 מצד 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 אותם 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

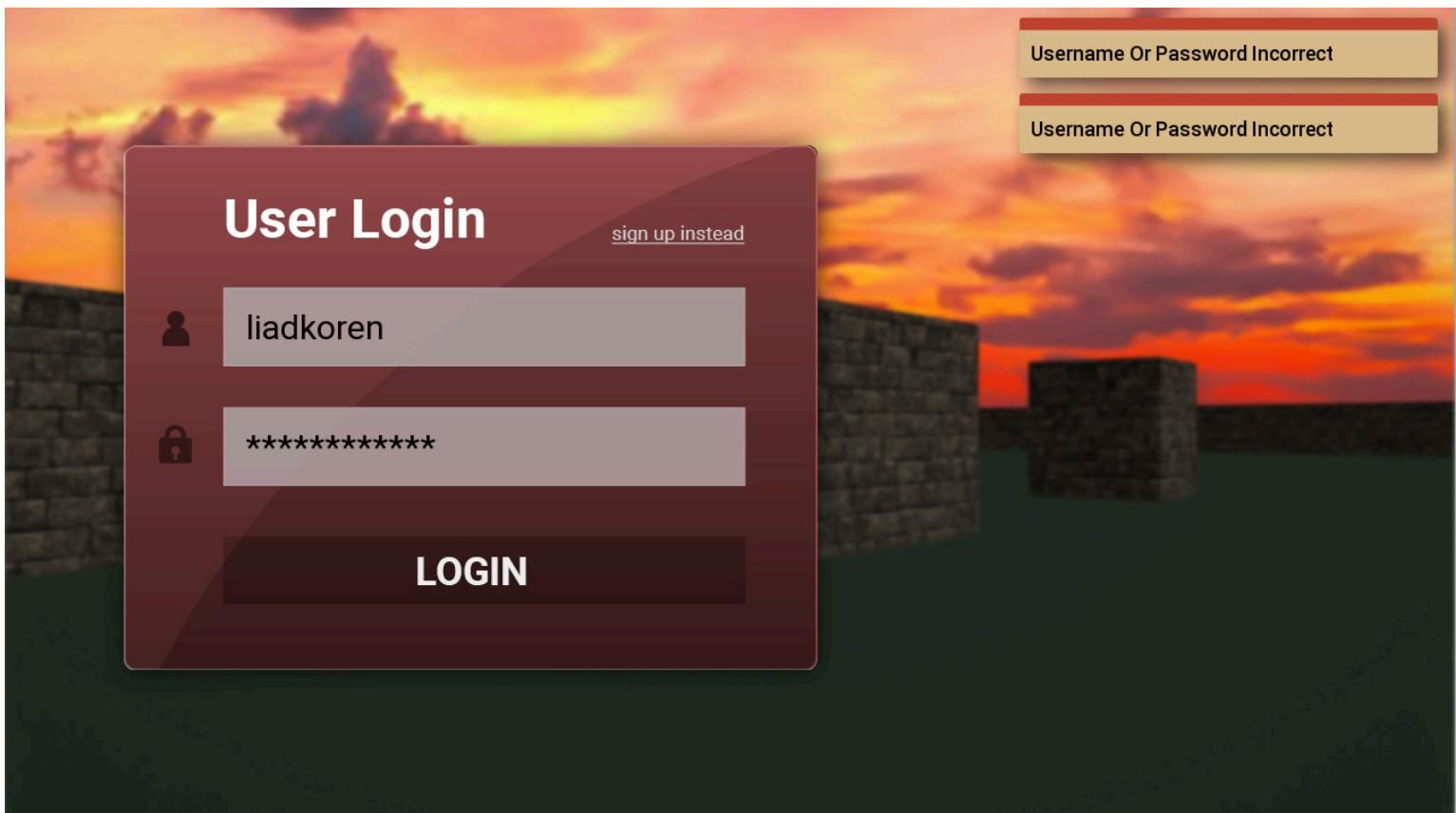
מנהל מערכת:

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.

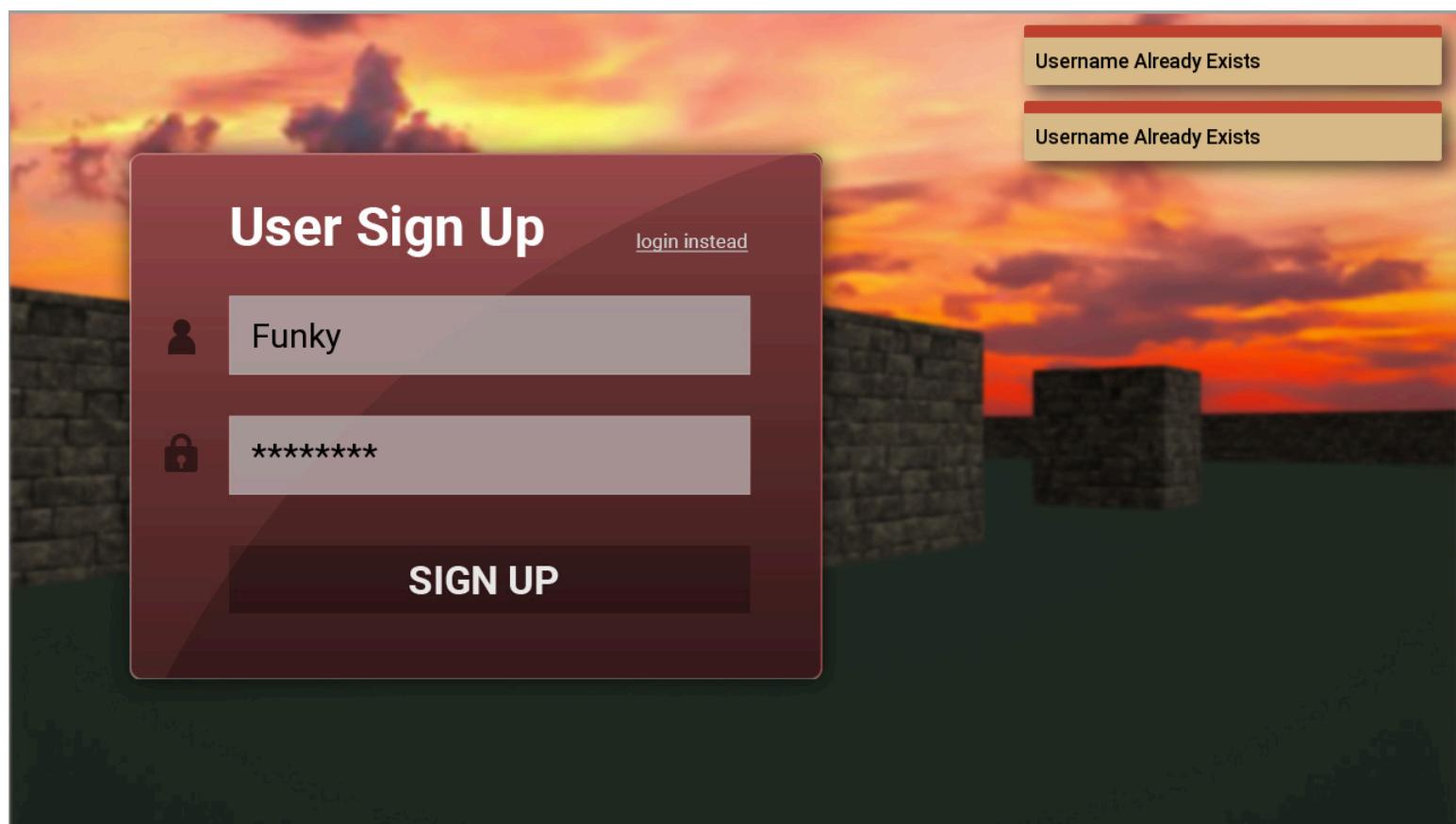
יש בתיקית השרת קובץ בשם server_address.text, שם הלקוח מסתכל כדי לדעת את כתובתו של השרת, הפקיד על השרת שדרכו נעשית תקשורת TCP והפקיד של UDP.

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 -

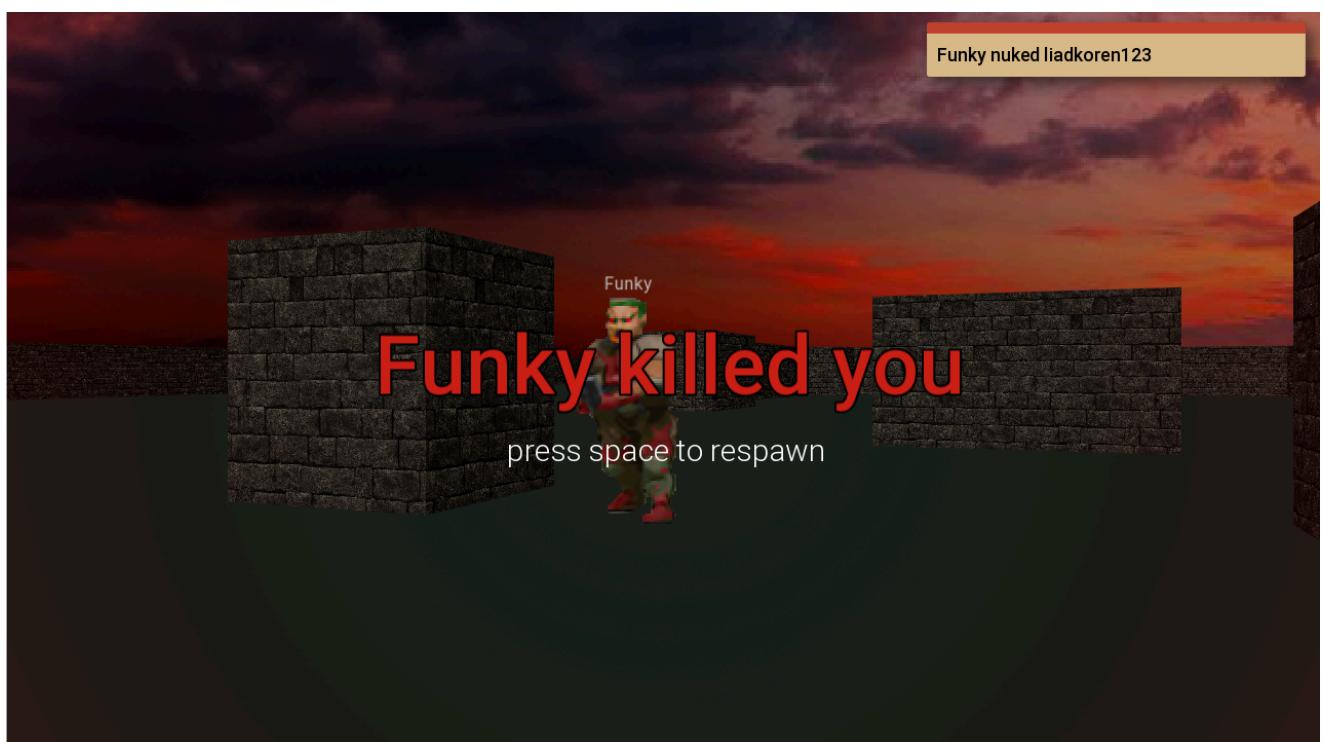


לייעד קורן - Raycaster

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



- שם / תפקיד – מסך מות.
- תיאור – לאחר ששחקן אחר יורה בר מספר פעמים, ונגמרים לירח חיים, קופץ המסך הזה שמספר לירח, אתה לא יכול לירוח או לזרז, ואתה 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 לczyיר את הטקסטורות במייקום א"ח הנכון על המסך, ולczyיר את הטקסט: יש טקסט במסך ההתחרבות, מסך יצירת המשתמש, ובמשחק הראשי בהודעות למשתמש (ימין למעלה) וטבלת הנקודות של המשתמשים (שמאלי למעלה).

השתמשתי בSFML גם כדי להשמיע רעשים כשהשחקן יורה את הרובה שלו, וכך ליצור את התקשרות בין הלוקוח לשרת. יש מחלקות TcpSocket וUdpSocket שאפשרו זאת.

ספרית מספרים גדולים `int` (`cpp_int`) `Multiprecision::Boost::`. השתמשתי בספריה זו כדי לחשב את החזקות הגדולות והמדוילו הדרוש בעת יצירת המפתח הסודי של הצפנה בתהילך Diffie Hellman. המפתח שומש בהמשך כדי להצפין הודעות עם AES128.

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

מודול `thread` - מאפשר הריצה של פעולות הנקשנה לשרת במקביל לצור המשחק על המסך בזמן אמת.

מודול `mutex` שיאפשר לי לנעול את המשתנים של האובייקטים של השחקנים האחרים, בזמן `thead` שמקשייב לשרת משנה את המיקום שלהם.

צד שרת (פייטון):

מודול `socket` - תקשורת עם סוקטים, TCP וUDP.

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

מודול `cryptography` - הצפנה AES

מודול `secrets` - יצירה מספרים רנדומליים גדולים ב-`users.db`

מודול `sqlite3` - עבודה מול מסד נתונים בשם SHA256

מודול `hashlib` - גיבוב הסיסמה באמצעות `hashing algorithm` נפוץ ובתווח שמו `++`

מודול `struct` - עיבוד המידע שהлокוח שולח לשרת בזמן המשחק, מידיע מעבר מ-`C++` לבניה שנקרה `struct`, המודול זהה מאפשר ליפענחו את הבטים ולהמיר אותם בשורה אחת לכל המשתנים שמרכיבים את המבנה נתונים `PlayerInfo`

מודול `math` - דריש בחישוב כיווני היריה של השחקנים, כדי לאמת אם אחת מהיריות פגעה באחד מהשחקנים האחרים. ובמ"ן.

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 כskins ופועלה 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 פחות עבודה עם מחלקות, אז 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 at_is_pointing_to 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 מוחזרות 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 ומכשולים 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.

באמצעות קראיה ל-"קראיים" מהמשחק לכל כיוונו, בדוק כמו יש במשחקי רובה הקלסטיים.

ב-drawGun נקראות הfonקציות שהפורטוגוניסט השחקן משתמש בהן כדי לצייר את רבו, כולל את תמונה הדו-מימדית של הרובה שלו.

ב-shootRays היא פונקציה שמטרתה לקבע את התמונה התלת-מימדית שנראית בעין השחקן ממקום המטרה הנוכחי.

ולולאה זו נקראת שוב ושוב עד שהמשחק מגיע לסיומו 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 ערכים 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 הcy קרוביה 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
102     v2i area_br = max((v2i)position, predicted_cell);
103     area_br = min({map.width, map.height}, area_br + ones); // clamp
104
105
106     for (int cell_x = area_tl.x; cell_x <= area_br.x; cell_x++)
107     {
108         for (int cell_y = area_tl.y; cell_y <= area_br.y; cell_y++)
109         {
110
111             // is wall
112             if (map.getCell(cell_x, cell_y) == 1)
113             {
114                 v2f nearest_point;
115                 nearest_point.x = std::max(float(cell_x), std::min(potential_position.x, float(cell_x + 1)));
116                 nearest_point.y = std::max(float(cell_y), std::min(potential_position.y, float(cell_y + 1)));
117
118                 v2f ray_to_nearest = nearest_point - potential_position;
119
120                 float overlap = body_radius - mag(ray_to_nearest);
121
122                 if (std::isnan(overlap)) overlap = 0;
123
124                 if (overlap > 0)
125                 {
126                     potential_position -= norm(ray_to_nearest) * overlap;
127                 }
128             }
129
130         }
131     }
132
133     position = potential_position;
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 פונקציות המקובלות וublisher את הקלט המשמש, כגון handleKeys שמתיחסת לקלט מקלדת וfonkציה rotateHead שמנהל את זווית הראש בהתאם לתנועה העבר.

fonקציות ניהול תנועה:

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

fonקציות עיבוד גרפיות:

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

fonקציות פנימיות:

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

fonקציות אחרות:

ישנן פונקציות אחרות המתיחסות לפחות נסיפות, כמו loadTextures שמטעינה את הטקסטורות של השחקן. מהלך המחלקה ישתמש כתפריט מרכזי לקבץ 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();

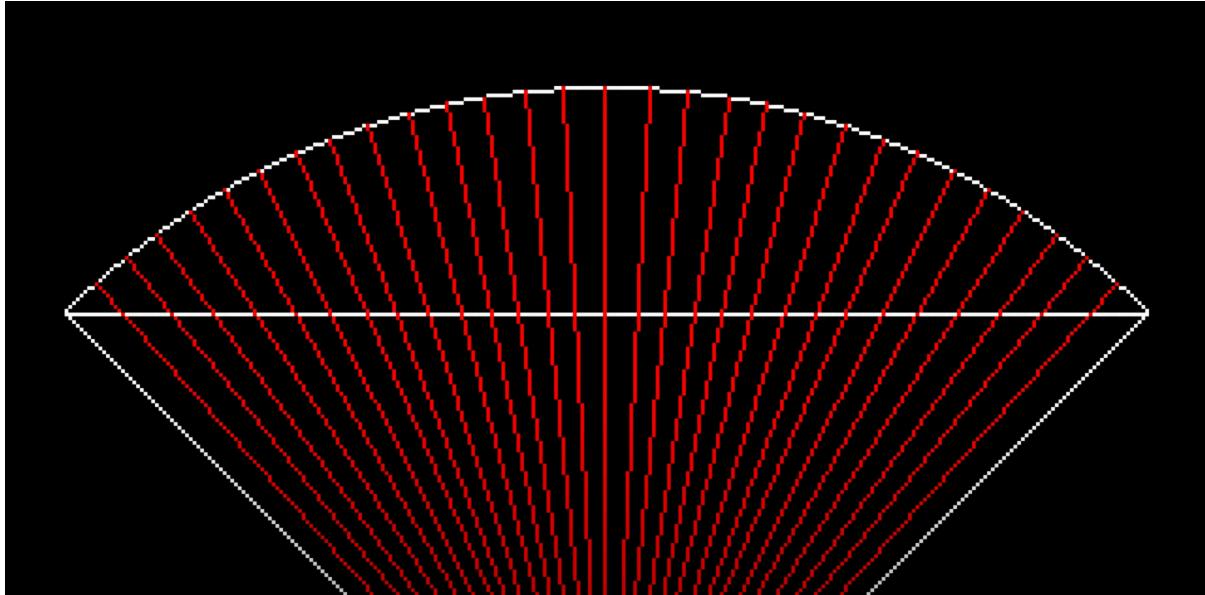
Object* getObject(int id);
Object* getAnyObject();
void handle_shooting_victim(int victim_id, int shooter_id);
void handle_killing(int killer_id, int victim_id);
void getKilled(const string& killer_name);
void respawn();

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

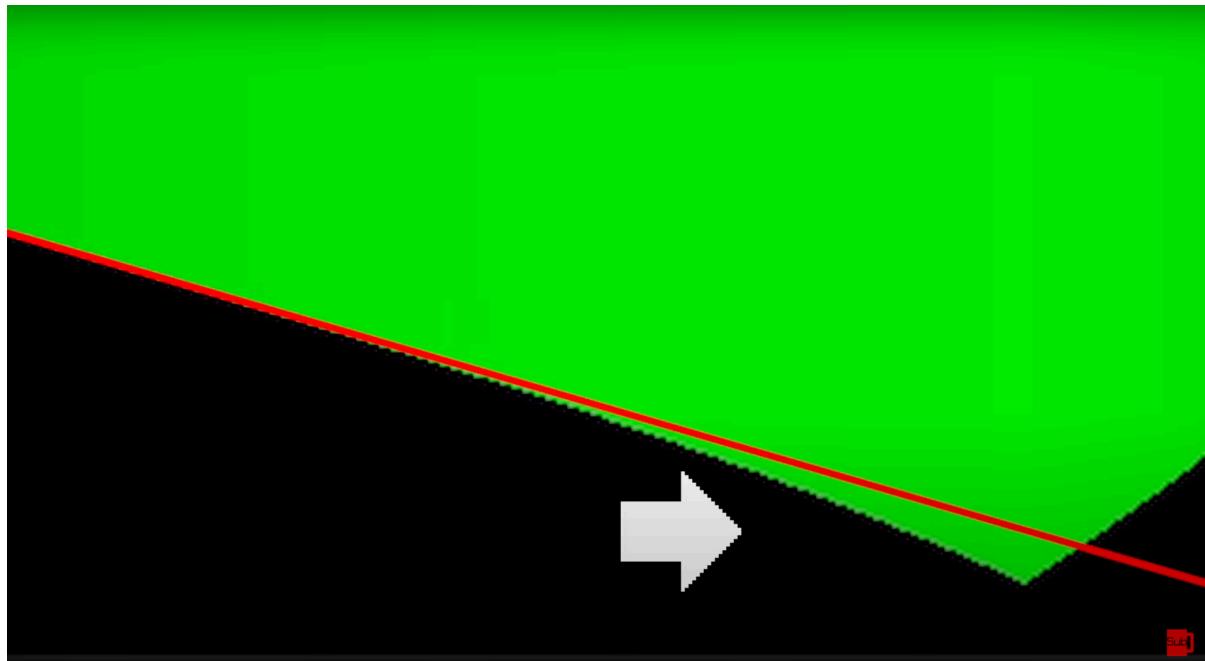
string getUsername(int id);
```

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:

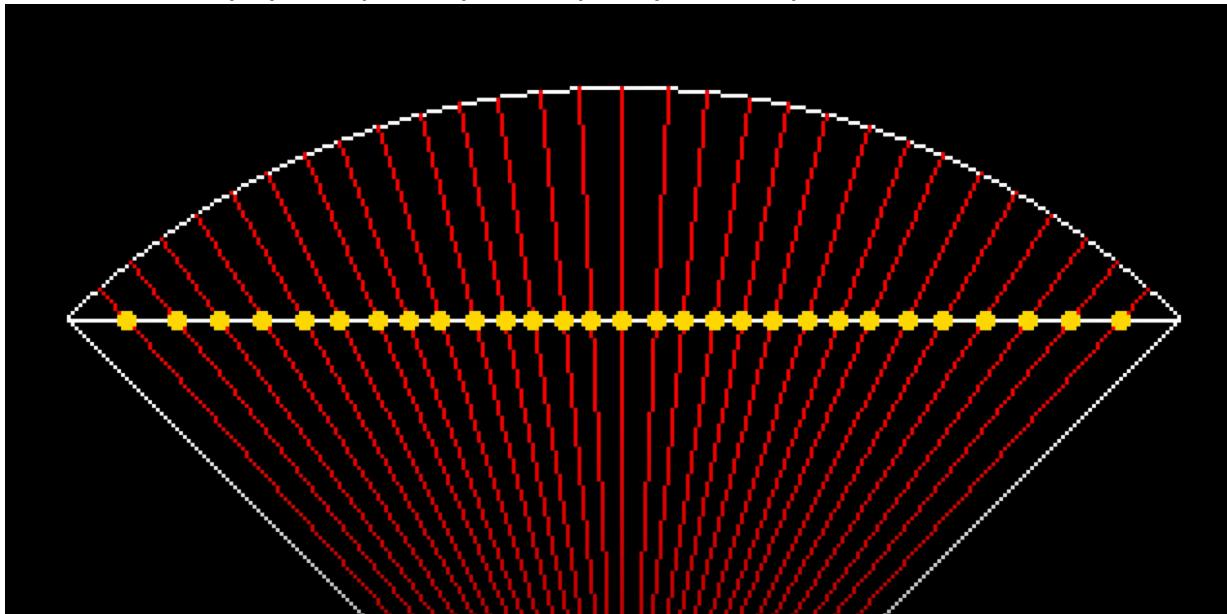


בתמונה, נקודת חיתוך של כל קיר עם הלבן מייצגת נקודת הקיר שהשחקן רואה. הבעיה היא שהמפרק בין כל נקודת חיתוך משתנה ככל שהקרנינים מתקרבים לקצוות. דבר זה גורם לעביה ויזואלית שנראית כך:



פתרון הבעיה:

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



הבחן במדרי' 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 אליהם 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 הטויריים במסך, ובשביל כל טור נדע לאן לשולח את הקרן כך שהמרחק בין נקודות החיתוך של הקרניים עם מישור הקיר יהיה זהה.

פונקציה מהמדרי':

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

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

הfonktsiya she ani meshutash ba

בעזרת אלגברה פשוטה של כיתה י', לקחנו את הפונקציה הלא שימושית מהמדרי' 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. בפייתון 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 מאוד מובן הגрафי 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 רצ. מוזר, חבשתי, ופתחתי 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 שיתפלו 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 בפייתון, למדתי על הדרך היכי טובה לעבוד עם מחלקות והאיזון המושלם שבין פעולות פנימיות לבין פעולות חיצונית. דבר שעזר לי בכתיבת הפייתון 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 אליו, מאשר בבדיקהizia סוג משתנה כל משתנה מייצג, כמו בשפות שאני אוהב, C++ ו-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.

client.cpp

```
#include "headers.hpp"
#include "client.hpp"
#include "tools.hpp"
#include <iomanip>
#include <openssl/evp.h>
#include <openssl/rand.h>
#include <fstream>

Client::Client()
{
    p = bigint("170141183460469231731687303715884105757");
    g = bigint("340282366920938463463374607431768211507");
    unsigned char secret_buffer[16];
    RAND_bytes(secret_buffer, 16);

    secret = 0;
    for (int i = 0; i < 16; i++)
    {
        secret <= 8;
        secret |= secret_buffer[i];
    }
}

bool Client::connectToServer(string& error)
{
    // parse server address file

    int server_tcp_port = -1, server_udp_port = -1;
    try
    {
        std::ifstream file("server/server_address.txt");
        // comment lines

        string line;
        while (std::getline(file, line))
        {
            if (line[0] != '#')
                break;
        }
        // ip address
        ip = line;

        // tcp & udp ports
        std::getline(file, line);
        server_tcp_port = std::stoi(line);
        std::getline(file, line);
        server_udp_port = std::stoi(line);

        cout << "ports: " << server_tcp_port << ":" << server_udp_port
        << "\n";
    }
}
```

client.cpp

```
catch (const std::exception&)
{
    error = "Invalid File at server/server_address.txt";
    return false;
}

this->ip = ip;
this->udp_sender_port = server_udp_port;
this->udp_listener_port = server_udp_port;

sf::Socket::Status connection_status = tcp_socket.connect(ip,
server_tcp_port, sf::milliseconds(50));
if (connection_status != sf::Socket::Done)
{
    error = "Failed To Connect To Server";
    return false;
}

cout << "address: " << tcp_socket.getRemoteAddress() << "\n";

// Diffie Hellman
// X1
bigint x1 = powm(g, secret, p);

string message = "X" + x1.str() + "X";

if (!sendTCP(tcp_socket, message, error))
    return false;

// X2
void* buffer;
int buffer_size;
if (!recvTCP(tcp_socket, buffer, buffer_size))
{
    error = "Failed To Receive Message From Server";
    return false;
}

string x2_str((char*)buffer, buffer_size);
free(buffer);

if (x2_str == "ERROR" || x2_str[0] != 'X' || x2_str[x2_str.size() - 1] != 'X')
{
    error = "Invalid X2 Received From Server: " + x2_str;
    return false;
}

bigint x2(x2_str.substr(1, x2_str.size() - 2));

// Key
```

client.cpp

```
bigint key = powm(x2, secret, p);
bigintToBytes(key, key_bytes);

if (udp_socket.bind(sf::Socket::AnyPort) != sf::Socket::Done)
{
    error = "Error binding UDP socket";
    return false;
}

udp_sender_address = udp_listener_address =
tcp_socket.getRemoteAddress();
cout << "UDP Socket Local Port: " << udp_socket.getLocalPort() <<
"\n";

udp_socket.setBlocking(false);

connected = true;
cout << "Connected To Server At (" << ip << ", " <<
server_tcp_port << ")\n";

return true;
}

bool Client::tryLogIn(const string& username, const string& password,
string& error)
{

if (username.size() <= 4)
{
    error = "Username Is Too Short";
    return false;
}
if (password.size() <= 4)
{
    error = "Password Is Too Short";
    return false;
}

if (username.find('~') != usernamenpos)
{
    error = "Username Cannot Have Tilde ~";
    return false;
}

// Connect to server
if (!connectToServer(error))
```

client.cpp

```
return false;

string creds = "LOGIN~" + username + "~" + password + "~";

sendEncryptedTCP(creds, error);

void* buffer;
int buffer_size;
if (!recvEncryptedTCP(buffer, buffer_size, error))
{
    error = "Failed To Receive Message From Server";
    return false;
}

string response((char*)buffer, buffer_size);
free(buffer);

vector<string> parts = split(response);

if (parts[0] == "SUCCESS")
{
    this->username = username;
    player_id = std::stoi(parts[1]);
    cout << "Player ID: " << player_id << "\n";
    return true;
}

error = parts[1];

return false;

}

bool Client::trySignUp(const string& username, const string& password,
string& error)
{

if (username.size() <= 4)
{
    error = "Username Is Too Short";
    return false;
}
if (password.size() <= 4)
{
    error = "Password Is Too Short";
    return false;
}
```

client.cpp

```
if (username.find('~') != usernamenpos)
{
    error = "Username Cannot Have Tilde ~";
    return false;
}

// Connect to server
if (!connectToServer(error))
    return false;

string creds = "SIGNUP~" + username + "~" + password + "~";

sendEncryptedTCP(creds, error);

void* buffer;
int buffer_size;
if (!recvEncryptedTCP(buffer, buffer_size, error))
{
    error = "Failed To Receive Message From Server";
    return false;
}

string response((char*)buffer, buffer_size);
free(buffer);

vector<string> parts = split(response);

if (parts[0] == "SUCCESS")
{
    return true;
}

error = parts[1];

return false;

}

//UDP
//encrypts message, sends the (char)id + the encrypted message.
bool Client::sendEncryptedUDP(void* buffer, int size, string& error)
{
```

client.cpp

```
string encrypted = encryptAES(string((char*)buffer, size),
key_bytes, error);
if (encrypted == "") {
    error += "encryption empty";
    return false;
}

string msg = (char)player_id + encrypted;

if (!sendUDP(msg, error))
    return false;
}

bool Client::recvEncryptedUDP(void*& buffer, int& buffer_size, string&
error)
{
if (!recvUDP(buffer, buffer_size))
{
    error = "Error Trying to receive encrypted udp";
    return false;
}

string ciphertext((char*)buffer, buffer_size);

string decrypted = decryptAES(ciphertext, key_bytes, error);
if (decrypted == "") {
    cout << "ERROR when decrypting the shit from the cunt: " <<
    error << "\n";
    return false;
}

buffer_size = decrypted.size();

memcpy(buffer, decrypted.c_str(), decrypted.size());
}

bool Client::sendUDP(const string& message, string& error)
{

int buffer_size = 128;
void* buffer = malloc(buffer_size);
if (buffer == 0)
{
    error = "couldn't allocate space for message buffer";
    return false;
}

if (message.size() > buffer_size)
{
```

client.cpp

```
error = "message size bigger than 128 bytes, " +
std::to_string(message.size()) + " bytes without null";
free(buffer);
return false;
}

// message string
memcpy(buffer, message.c_str(), message.size());

if (udp_socket.send(buffer, message.size(), udp_sender_address,
udp_sender_port) != sf::Socket::Done)
{
    error = "Failure When Sending The Message: " + message;
    free(buffer);
    return false;
}

free(buffer);
}

// returns true on success
bool Client::recvUDP(void*& buffer, int& buffer_size)
{
    //buffer
    int msg_length = 256;
    buffer = malloc(msg_length);
    if (buffer == 0)
    {
        cout << "couldn't allocate space for payload\n";
        return false;
    }

    size_t amount_received;
    sf::Socket::Status status = udp_socket.receive(buffer, msg_length,
amount_received, udp_listener_address, udp_listener_port);

    if (status == sf::Socket::Done)
    {
        buffer_size = amount_received;
        return true;
    }

    if (status == sf::Socket::Error)
        cout << "Error Receiving UDP\n";

    free(buffer);

    return false;
}

// TCP
bool Client::sendEncryptedTCP(const string& msg, string& error)
```

client.cpp

```
{  
    string encrypted = encryptAES(msg, key_bytes, error);  
    if (encrypted == "")  
        return false;  
  
    sendTCP(tcp_socket, encrypted, error);  
}  
  
bool Client::recvEncryptedTCP(void*& buffer, int& buffer_size, string&  
error)  
{  
    if (!recvTCP(tcp_socket, buffer, buffer_size))  
    {  
        error = "Error Trying to receive encrypted tcp";  
        return false;  
    }  
  
    string ciphertext((char*)buffer, buffer_size);  
  
    string decrypted = decryptAES(ciphertext, key_bytes, error);  
    if (decrypted == "")  
    {  
        cout << "ERROR when decrypting the shit from the cunt: " <<  
        error << "\n";  
        return false;  
    }  
  
    memcpy(buffer, decrypted.c_str(), decrypted.size());  
}  
  
// returns true on success  
bool Client::recvTCP(sf::TcpSocket& socket, void*& buffer, int&  
buffer_size)  
{  
    socket.setBlocking(false);  
  
    sf::SocketSelector selector;  
    selector.add(socket);  
  
    if (!selector.wait(sf::milliseconds(1000)))  
    {  
        cout << "Nothing Received\n";  
        socket.setBlocking(true);  
        return false;  
    }  
  
    socket.setBlocking(true);  
  
    //length
```

client.cpp

```
short msg_len = 0;
size_t amount_received;
if (socket.receive(&msg_len, 2, amount_received) != sf::Socket::Done || amount_received != 2)
{
    cout << "Error when receiving msg length\n";
    return false;
}

buffer = malloc(msg_len);
if (buffer == 0)
{
    cout << "couldn't allocate space for payload\n";
    return false;
}

// message string
socket.receive(buffer, msg_len, amount_received);

if (amount_received != msg_len)
{
    cout << "Error when receiving message with length: " << msg_len
    << "\n";
    return false;
}

buffer_size = amount_received;
return true;
}

bool Client::sendTCP(sf::TcpSocket& socket, const string& message,
string& error)
{
//cout << "Sending: " << message << "\n";

int buffer_size = 2 + message.size();
void* buffer = malloc(buffer_size);
if (buffer == 0)
{
    error = "couldn't allocate space for message buffer";
    return false;
}

// length
short msg_len = message.size();
memcpy(buffer, &msg_len, 2);

// message string
memcpy((char*)buffer + 2, message.c_str(), msg_len);
```

client.cpp

```
size_t amount_sent;
if (socket.send(buffer, buffer_size, amount_sent) != sf::Socket::Done)
{
    error = "Failure When Sending The Message: " + message;
    free(buffer);
    return false;
}

free(buffer);

if (amount_sent != buffer_size)
{
    error = "Error sending message, not whole message sent: " +
message;
    return false;
}

void printBytes(const unsigned char* pBytes, const uint32_t nBytes) // should more properly be std::size_t
{
    for (uint32_t i = 0; i < nBytes; i++)
    {
        std::cout <<
            std::hex <<                      // output in hex
            std::setw(2) <<                    // each byte prints as two characters
            std::setfill('0') <<               // fill with 0 if not enough
            characters
            (int)pBytes[i] << " ";
    }
    cout << std::dec << "\n";
}

void printBytes(void* pBytes, const uint32_t nBytes)
{
    printBytes((unsigned char*)pBytes, nBytes);
}

string encryptAES(const void*& buffer, int size, unsigned char* key,
string& error) {
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    if (!ctx) {
        error = "Error creating EVP_CIPHER_CTX.";
        return "";
    }

    if (EVP_EncryptInit_ex(ctx, EVP_aes_128_ecb(), NULL, key, NULL) != 1) {
        cout << "Error initializing AES encryption.";
        EVP_CIPHER_CTX_free(ctx);
    }
}
```

client.cpp

```
    return "";
}

string ciphertext(size + EVP_CIPHER_block_size(EVP_aes_128_ecb()), '\0');
int outLen;
if (EVP_EncryptUpdate(ctx, reinterpret_cast<unsigned char*>(&ciphertext[0]), &outLen, reinterpret_cast<const unsigned char*>(buffer),
    error = "Error encrypting data.");
    EVP_CIPHER_CTX_free(ctx);
    return "";
}

int finalLen;
if (EVP_EncryptFinal_ex(ctx, reinterpret_cast<unsigned char*>(&ciphertext[outLen]), &finalLen) != 1) {
    error = "Error finalizing encryption.";
    EVP_CIPHER_CTX_free(ctx);
    return "";
}

EVP_CIPHER_CTX_free(ctx);
ciphertext.resize(outLen + finalLen);
return ciphertext;
}

string encryptAES(const string& plaintext, unsigned char* key, string& error) {
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    if (!ctx) {
        error = "Error creating EVP_CIPHER_CTX.";
        return "";
    }

    if (EVP_EncryptInit_ex(ctx, EVP_aes_128_ecb(), NULL, key, NULL) != 1) {
        cout << "Error initializing AES encryption.";
        EVP_CIPHER_CTX_free(ctx);
        return "";
    }

    string ciphertext(plaintext.size() +
EVP_CIPHER_block_size(EVP_aes_128_ecb()), '\0');
    int outLen;
    if (EVP_EncryptUpdate(ctx, reinterpret_cast<unsigned char*>(&ciphertext[0]), &outLen, reinterpret_cast<const unsigned char*>(plain,
        error = "Error encrypting data.");
        EVP_CIPHER_CTX_free(ctx);
        return "";
    }

    int finalLen;
    if (EVP_EncryptFinal_ex(ctx, reinterpret_cast<unsigned char*>(&ciphertext[outLen]), &finalLen) != 1) {
        error = "Error finalizing encryption.";
        EVP_CIPHER_CTX_free(ctx);
        return "";
    }

    ciphertext.resize(outLen + finalLen);
    return ciphertext;
}
```

client.cpp

```
char*>(&ciphertext[outLen]), &finalLen) != 1) {
    error = "Error finalizing encryption.";
    EVP_CIPHER_CTX_free(ctx);
    return "";
}

EVP_CIPHER_CTX_free(ctx);
ciphertext.resize(outLen + finalLen);
return ciphertext;
}

string decryptAES(const string& ciphertext, unsigned char* key, string&
error) {
    EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
    if (!ctx) {
        error = "Error creating EVP_CIPHER_CTX.";
        return "";
    }

    if (EVP_DecryptInit_ex(ctx, EVP_aes_128_ecb(), NULL, key, NULL) !=
1) {
        error = "Error initializing AES decryption.";
        EVP_CIPHER_CTX_free(ctx);
        return "";
    }

    string plaintext(ciphertext.size(), '\0');
    int outLen;
    if (EVP_DecryptUpdate(ctx, reinterpret_cast<unsigned
char*>(&plaintext[0]), &outLen, reinterpret_cast<const unsigned char*>(cipher
error = "Error decrypting data.");
    EVP_CIPHER_CTX_free(ctx);
    return "";
}

int finalLen;
if (EVP_DecryptFinal_ex(ctx, reinterpret_cast<unsigned
char*>(&plaintext[outLen]), &finalLen) != 1) {
    error = "Error finalizing decryption.";
    EVP_CIPHER_CTX_free(ctx);
    return "";
}

EVP_CIPHER_CTX_free(ctx);
plaintext.resize(outLen + finalLen);
return plaintext;
}

void bigintToBytes(bigint key, unsigned char* buffer)
{
    for (int i = 15; i >= 0; i--)
    {
        buffer[i] = (unsigned char)(key & 255);
    }
}
```

client.cpp

```
key >= 8;  
}  
}
```

client.hpp

```
#pragma once

#include <SFML/Network.hpp>
#include <boost/multiprecision/cpp_int.hpp>
#include <thread>
#include <mutex>

typedef boost::multiprecision::cpp_int bigint;

using boost::multiprecision::powm;

//bool tryLogIn(const string& username, const string& password, string& error);

bool sendUDP(sf::UdpSocket& socket, const string& message, string& error);
bool recvUDP(sf::UdpSocket& socket, void*& buffer, int& buffer_size);

void printBytes(const unsigned char* pBytes, const uint32_t nBytes);
void printBytes(void* pBytes, const uint32_t nBytes);

//Encryption
string encryptAES(const std::string& plaintext, unsigned char* key,
string& error);
string decryptAES(const string& ciphertext, unsigned char* key, string& error);

//string bigintToHexString(const bigint& number);
void bigintToBytes(bigint key, unsigned char* buffer);

class Client
{
private:
    sf::TcpSocket tcp_socket;
    sf::UdpSocket udp_socket;
    sf::IpAddress udp_sender_address, udp_listener_address;
    string ip;
    unsigned short udp_sender_port, udp_listener_port;

    bigint p, g, secret;
    unsigned char key_bytes[16];

public:
    int player_id;
    string username;

    struct PlayerInfo
    {
        enum Flag
        {
            moving = 1,
```

client.hpp

```
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;
int score;
char username[16];
};

Client();

bool connectToServer(string& error);

bool tryLogIn(const string& username, const string& password,
string& error);
bool trySignUp(const string& username, const string& password,
string& error);

bool sendEncryptedTCP(const string& msg, string& error);
bool recvEncryptedTCP(void*& buffer, int& bufferSize, string&
error);
static bool recvTCP(sf::TcpSocket& socket, void*& buffer, int&
buffer_size);
static bool sendTCP(sf::TcpSocket& socket, const string& message,
string& error);

bool sendEncryptedUDP(void* buffer, int size, string& error);
bool recvEncryptedUDP(void*& buffer, int& bufferSize, string&
error);
bool sendUDP(const string& message, string& error);
bool recvUDP(void*& buffer, int& buffer_size);
bool connected = false;
};
```

headers.hpp

```
#ifndef HEADERS_HPP
#define HEADERS_HPP

#include <SFML/Graphics.hpp>
#include <SFML/Network.hpp>

#include <vector>
#include <iostream>
#include <cmath>

typedef sf::Vector2f v2f;
typedef sf::Vector2i v2i;
using std::cout;
using std::vector;
using std::string;

const double PI = 3.14159265358979323846;
const double HALF_PI = 3.14159265358979323846/2;
const float TO_DEGREES = 57.29577955f;

const int WIDTH = 1280, HEIGHT = 720;

#endif // !HEADERS_HPP
```

main.cpp

```
#include "headers.hpp"
#include "tools.hpp"
#include "player.hpp"
#include "map.hpp"
#include "object.hpp"
#include "client.hpp"
#include "toaster.hpp"

#include <chrono>

void loginPage(sf::RenderWindow& window, Player& player, Toaster&
toaster);
void mainLoop(sf::RenderWindow& window, Player& player, Toaster&
toaster);

int main()
{
    //Window
    sf::RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "Program",
    sf::Style::Close, sf::ContextSettings(24, 8, 8));

    window.setFramerateLimit(60);

    Toaster toaster;

    Player player(40, 21, window, toaster);

    loginPage(window, player, toaster);

    // Game loop
    mainLoop(window, player, toaster);

    return 0;
}

void loginPage(sf::RenderWindow& window, Player& player, Toaster&
toaster)
{
    // background image
    sf::Texture login_tex, signup_tex;
    login_tex.loadFromFile("sprites/loginpage.jpg");
    signup_tex.loadFromFile("sprites/signuppage.jpg");

    sf::Sprite bg_sprite(login_tex);

    // font
    sf::Font input_font;
    if (!input_font.loadFromFile("Fonts/Roboto-Regular.ttf"))
```

main.cpp

```
{  
    std::cerr << "Error Loading File.\n";  
    return;  
}  
  
bool logging_in = true;  
bool enter_pressed = false;  
  
// text box and text  
  
TextBox username(v2f(194, 249), v2f(461, 70), "", input_font);  
TextBox password(v2f(194, 355), v2f(461, 70), "", input_font);  
  
password.hidden = true;  
  
TextBox* text_boxes[3] = { nullptr, &username, &password };  
  
int box_focused = 1;  
  
sf::Clock clock;  
  
v2f enter_position(193, 469), enter_size(462, 61);  
v2f switch_position(530, 185), switch_size(130, 30);  
  
while (window.isOpen())  
{  
    float dt = clock.restart().asSeconds();  
  
    string typed_text = "";  
    int backspace_counter = 0;  
  
    sf::Event event;  
    while (window.pollEvent(event))  
        if (event.type == sf::Event::Closed)  
            window.close();  
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))  
            window.close();  
        else if (event.type == sf::Event::TextEntered) {  
            // actual typing  
            if (event.text.unicode > 32 && event.text.unicode < 127) {  
                typed_text += event.text.unicode;  
            }  
  
            // backspaces  
            if (event.text.unicode == '\b')  
                backspace_counter++;  
            if (event.text.unicode == 127) // ctrl backspace  
                backspace_counter = -100;  
  
            // tab
```

main.cpp

```
if (event.text.unicode == '\t' && box_focused)
{
    box_focused = (box_focused + 1) % 3;
    if (box_focused == 0)
        box_focused = 1;

    text_boxes[box_focused]->turnOnCursor();
}

//enter
if (event.text.unicode == '\r')
    enter_pressed = true;

}

else if (event.type == sf::Event::MouseButtonPressed) {
    // Check if mouse click is within the text box
    v2i mousePos = sf::Mouse::getPosition(window);

    box_focused = 0;
    for (int i = 1; i < 3; i++)
    {
        if (text_boxes[i]->inBox(mousePos))
        {
            box_focused = i;
            text_boxes[i]->turnOnCursor();
        }
    }

    }

if (inBounds(enter_position, enter_size, mousePos))
    enter_pressed = true;

if (inBounds(switch_position, switch_size, mousePos))
{
    logging_in ^= true;
    if (logging_in) bg_sprite.setTexture(login_tex);
    else bg_sprite.setTexture(signup_tex);
    text_boxes[1]->clearText();
    text_boxes[2]->clearText();
    box_focused = 1;
}
}

if (enter_pressed)
{
    string error;
    if (logging_in)
    {
        if (player.client.tryLogIn(username.getString(),
password.getString(), error))
        {
```

main.cpp

```
        toaster.toast( "Connection Successful! " );
        return;
    }
    toaster.toast(error);

}

else // signing up
{
    if (player.client.trySignUp(username.getString(),
password.getString(), error))
    {
        toaster.toast("Signup Successful!");
        logging_in = true;
        bg_sprite.setTexture(login_tex);
        text_boxes[1]->clearText();
        text_boxes[2]->clearText();
        box_focused = 1;
    }
    else
        toaster.toast(error);
}
}

enter_pressed = false;

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

window.draw(bg_sprite);

//box highlight
if (box_focused)
{
    if (typed_text.size())
        text_boxes[box_focused]->addText(typed_text);
    if (backspace_counter)
        text_boxes[box_focused]->backspace(backspace_counter);
}

for (int i = 1; i < 3; i++)
    text_boxes[i]->draw(window, i == box_focused);

toaster.drawToasts(window, dt);

window.display();

}

void mainLoop(sf::RenderWindow& window, Player& player, Toaster&
```

main.cpp

```
toaster)
{
    v2i screen_center(WIDTH / 2, HEIGHT / 2);

    int frame_count = 0;
    sf::Clock clock;

    Player::HitInfo* hits = new Player::HitInfo[WIDTH];

    std::thread udpThread(&Player::listenToServer, &player);

    player.setFocus(true);

    while (window.isOpen())
    {
        float dt = clock.restart().asSeconds();

        sf::Event event;
        while (window.pollEvent(event))
            if (event.type == sf::Event::Closed)
                player.quitGame();
            else if (player.window_focused && event.type ==
sf::Event::MouseButtonPressed)
                player.shootGun(event.mouseButton.button ==
sf::Mouse::Left);
            else if (event.type == sf::Event::LostFocus)
                player.setFocus(false);
            else if (event.type == sf::Event::GainedFocus)
                player.setFocus(true);
            else if (player.window_focused && event.type ==
sf::Event::MouseMoved)
            {
                v2i current_pos = sf::Mouse::getPosition(window);

                player.rotateHead(current_pos.x - screen_center.x,
                    current_pos.y - screen_center.y, dt);

                sf::Mouse::setPosition(screen_center, window);
            }
            else if (event.type == sf::Event::KeyReleased)
            {
                if (event.key.code == sf::Keyboard::Space)
                    player.respawn();
            }

//if (frame_count % 100 == 0)
//    cout << (1 / dt) << "\n";
```

main.cpp

```
player.updateServer( );

player.handleKeys(dt);

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

player.map.drawSky(); // Sky

player.map.drawGround();

player.shootRays(hits); // populate hits[]

// World

{
    std::lock_guard<std::mutex> lock(player_mtx);

    player.drawWorld(hits, dt);

    //std::cout << "Elapsed time: " << elapsed.count() * 1000
    //                << " ms" << std::endl;
}

if (player.debug_mode)
{
    player.rotateHead(1, 0, 0.3);
}

player.drawGun(dt); // Gun

player.drawCrosshair(dt); // Crosshair

player.drawDeathScreen(dt);

toaster.drawToasts(window, dt);

toaster.drawLeaderboard(window, player.leaderboard, dt);

window.display(); // Render to screen

frame_count++;

}
```

main.cpp

```
delete[] hits;  
}
```

map.cpp

```
#include "headers.hpp"
#include "map.hpp"

Map::Map(int dist_from_side, sf::RenderWindow& window) : window(window)
{
    sf::Image map_texture;
    map_texture.loadFromFile("sprites/map.png");

    width = map_texture.getSize().x;
    height = map_texture.getSize().y;

    data = new int[width * height];
    for(int i = 0; i < width; i++)
        for (int j = 0; j < height; j++)
    {
        if (map_texture.getPixel(i, j).r == 255)
            data[i + j * width] = 1;
        else
            data[i + j * width] = 0;
    }

    map_texture ~Image();

    position.x = dist_from_side;
    position.y = HEIGHT - dist_from_side - cell_size * height;

    sky_color = sf::Color(70, 170, 255);
    ground_color = sf::Color(33, 43, 35);

    sky_tex.loadFromFile("sprites/sky.png");
    sky_sprite.setTexture(sky_tex);
    //sky_sprite.setTextureRect(sf::IntRect(0, 30, 1833, 460));
    sky_sprite.setScale(sky_scale, sky_scale);
}

int Map::getCell(int x, int y)
{
    return data[x + width * y];
}

void Map::drawMap()
{
    sf::RectangleShape cell(v2f(cell_size, cell_size));
    //cell.setOutlineColor(sf::Color(70, 70, 80));
}
```

map.cpp

```
■// cell.setOutlineThickness(1);

■sf::Color colors[] {
■■sf::Color::Black,
■■sf::Color::White
■};

■for (int i = 0; i < width; i++)
■{
■■for (int j = 0; j < height; j++)
■■{
■■■int color_index = getCell(i, j);
■■■cell.setFillColor(colors[color_index]);
■■■cell.setPosition(position.x + cell_size * i, position.y + cell_size
* j);
■■■window.draw(cell);
■■}
■}
}

void Map::drawPoint(float x, float y)
{
■sf::CircleShape dot(5);

■dot.setFillColor(sf::Color::Red);

■dot.setPosition(position.x + cell_size * x - 2.5f, position.y +
cell_size * y - 2.5f);
■
■window.draw(dot);
■■
}

void Map::drawSky()
{
■float sky_y = floor_level - 950;
■sky_sprite.setPosition(sky_offset - sky_width * sky_scale, sky_y);
■window.draw(sky_sprite);

■sky_sprite.setPosition(sky_offset, sky_y);
■window.draw(sky_sprite);

■//cout << "Floor Level: " << floor_level << "\n";
}

void Map::shiftSky(float offset)
{
■sky_offset += offset * sky_sensitivity;

■if(sky_offset > sky_width * sky_scale)
■■sky_offset -= sky_width * sky_scale;
```

map.cpp

```
■if (sky_offset < WIDTH - sky_width * sky_scale)
■■sky_offset += sky_width * sky_scale;
}

void Map::drawGround()
{
■//cout << width << " " << height << "\n";
■float ground_height = HEIGHT - floor_level;

■sf::RectangleShape ground(v2f(WIDTH, ground_height));

■ground.setFillColor(ground_color);
■ground.setPosition(v2f(0, floor_level));
■window.draw(ground);
}
```

map.hpp

```
#pragma once

class Map
{
private:
    int* data;
    sf::RenderWindow& window;

    sf::Texture sky_tex;
    sf::Sprite sky_sprite;

public:
    int width, height;
    v2i position;

    int cell_size = 4;

    float floor_level = 354;

    float sky_offset = 0;
    float sky_scale = 2.5f;
    float sky_width = 1833;
    float sky_sensitivity = -1000;

    sf::Color sky_color, ground_color;

    Map(int dist_from_side, sf::RenderWindow& window);

    int getCell(int x, int y);
    void drawMap();
    void drawPoint(float x, float y);

    void drawGround();
    void drawSky();
    void shiftSky(float offset);
    void darkenScreen();
};

}
```

object.cpp

```
#include "headers.hpp"
#include "object.hpp"
#include "client.hpp" // for struct PlayerInfo

Object::Object(float x, float y, const sf::Texture& tex)
    : player_id(-1), position(x, y), direction_index(0)
{
    tex_size = (v2f)tex.getSize();
    sprite.setTexture(tex, true);
    sprite.setTextureRect(getTextureRect(direction_index, 0));
    shrink_by = 1.9f;
    scale_by = 0.0039f;
}

float Object::distFrom(const v2f& pos)
{
    return sqrtf(pow(pos.x - position.x, 2) + pow(pos.y - position.y,
    2));
}

void Object::animate(float dt)
{
    if (dying_timer >= 0)
    {
        dying_timer += dt;
        if (dying_timer > 0.5f)
        {
            dead = true;
            sprite.setTextureRect(getTextureRect(4, 8));
            dying_timer = -1;
        }
        else
        {
            int animation_index = dying_timer * 10;
            sprite.setTextureRect(getTextureRect(animation_index, 8));
        }
        return;
    }

    if (dead)
        return;

    if (gun_timer >= 0)
    {
        gun_timer += dt;

        if (gun_timer > 0.2f)
            gun_timer = -1;
    }
}
```

object.cpp

```
else if (gun_timer < 0.03f)
    sprite.setTextureRect(getTextureRect(direction_index, 5));
else if (gun_timer < 0.12f)
    sprite.setTextureRect(getTextureRect(direction_index, 6));
else
    sprite.setTextureRect(getTextureRect(direction_index, 5));

    return;
}

if (getting_shot_timer >= 0)
{
    getting_shot_timer += dt;
    if (getting_shot_timer > 0.22f)
        getting_shot_timer = -1;

    sprite.setTextureRect(getTextureRect(direction_index, 7));
    return;
}

if (!moving)
{
    sprite.setTextureRect(getTextureRect(direction_index, 4)); // 4
    - standing frame
    return;
}

float interval = 0.2f;
animation_timer += dt;
if (started_moving || animation_timer > interval)
{
    animation_timer -= interval;

    // there are 4 animation frames
    if (forward)
        animation_index = (animation_index + 1) % 4;
    else
        animation_index = (animation_index - 1 + 4) % 4;

    sprite.setTextureRect(getTextureRect(direction_index,
    animation_index));

    started_moving = false;
}

void Object::shootGun()
{
    gun_timer = 0;
}
```

object.cpp

```
void Object::gotShot()
{
    getting_shot_timer = 0;
}

void Object::gotKilled()
{
    dying_timer = 0;
}

void Object::loadPlayerInfo(Client::PlayerInfo player_info)
{
    player_id = player_info.player_id;

    username = player_info.username;

    position.x = player_info.pos_x;
    position.y = player_info.pos_y;

    rotation_x = player_info.rot_x;

    bool moving_flag = player_info.flags &
Client::PlayerInfo::Flag::moving;
    bool forward_flag = player_info.flags &
Client::PlayerInfo::Flag::forward;
    bool shot_gun_flag = player_info.flags &
Client::PlayerInfo::Flag::gun_shot;
    bool got_shot_flag = player_info.flags &
Client::PlayerInfo::Flag::got_shot;
    bool dead_flag = player_info.flags &
Client::PlayerInfo::Flag::dead;

    if (moving_flag != moving)
    {
        //player started or stopped moving
        started_moving = true;
    }

    dead = dead_flag;

    if (dead)
        sprite.setTextureRect(getTextureRect(4, 8));

    moving = moving_flag;
    forward = forward_flag;

    if (shot_gun_flag)
        shootGun();
```

object.cpp

```
}
```

```
sf::IntRect Object::getTextureRect(float rotation, float frame)
{
    return sf::IntRect(280 * rotation, 280 * frame, 280, 280);
}
```

```
Object::Object()
{
    player_id = -1;
}
```

object.hpp

```
#pragma once
#include "headers.hpp"
#include "client.hpp"

class Object
{
public:
    sf::Sprite sprite;

    v2f tex_size;
    float scale_by, shrink_by;

    v2f position;
    float rotation_x = 0;
    int direction_index; // 0, 1, 2, 3, 4, 5, 6, 7

    bool moving = false, started_moving = false, forward;
    int animation_index = 0;
    float animation_timer = 0;

    float gun_timer = -1;

    float getting_shot_timer = -1;

    bool dead = false;
    float dying_timer = -1;

    int player_id = -1;
    string username = "MISSING USERNAME";

    Object();
    Object(float x, float y, const sf::Texture& tex);
    float distFrom(const v2f& pos);

    void loadPlayerInfo(Client::PlayerInfo player_info);

    void animate(float dt);
    void shootGun();
    void gotShot();
    void gotKilled();

    static sf::IntRect getTextureRect(float rotation, float frame);
};

}
```

player.cpp

```
#include "headers.hpp"
#include "player.hpp"
#include "tools.hpp"
#include "client.hpp"
#include <time.h>

Player::Player(int x, int y, sf::RenderWindow& window, Toaster&
toaster)
    : toaster(toaster), map(20, window), position(x, y),
      window(window)
{
    loadTextures();
    loadSFX();
    srand(time(NULL));
}

void Player::setFocus(bool focus)
{
    window.setMouseCursorVisible(!focus);
    window_focused = focus;
    if (focus)
        sf::Mouse::setPosition({ WIDTH / 2, HEIGHT / 2 }, window);
}

#define KEY_PRESSED(key)
sf::Keyboard::isKeyPressed(sf::Keyboard::Key::key)
void Player::handleKeys(float dt)
{
    if (!window_focused)
        return;

    if (KEY_PRESSED(Escape))
        quitGame();

    // move
    v2f movement(0, 0);
    if (KEY_PRESSED(W))
        movement.y += +1;
    if (KEY_PRESSED(A))
        movement.x += +1;
    if (KEY_PRESSED(S))
        movement.y += -1;
    if (KEY_PRESSED(D))
        movement.x += -1;

    if (dead)
```

player.cpp

```
movement = v2f(0, 0);

moving = (movement.x != 0 || movement.y != 0);

if (moving)
    move(atan2(movement.x, movement.y), dt);

moving_forward = movement.y > 0;

// run
if (KEY_PRESSED(LShift))
    running = true;
else
    running = false;

if (KEY_PRESSED(LControl))
    crouching = true;
else
    crouching = false;

}

void Player::quitGame()
{
    has_quit = true;

    updateServer();

    window.close();
}

void Player::rotateHead(int delta_x, int delta_y, float dt)
{
    // ignore enormous rotation requests
    float move_size = mag(v2f(delta_x, delta_y)) * mouse_sensitivity *
        dt;
    if (dt != -1 && move_size > 0.6f)
    {
        cout << "size " << move_size << " movement suppressed" << "\n";
        return;
    }

    // horizontal
    rotation_x += delta_x * mouse_sensitivity * dt;
    map.shiftSky(delta_x * mouse_sensitivity * dt); // shift sky
```

player.cpp

```
// vertical
rotation_y += delta_y * -mouse_sensitivity * dt;

// cap y rotation
if (rotation_y > -0.15f)
    rotation_y = -0.15f;

if (rotation_y < -1.05f)
    rotation_y = -1.05f;

map.floor_level = HEIGHT / 2 * (1 + tan(rotation_y)) / tan(fov_y /
2);

}

void Player::move(float angle_offset, float dt)
{

    float current_speed = speed;
    if (running)
        current_speed *= run_multiplier;
    else if (crouching)
        current_speed *= crouch_multiplier;

    v2f potential_position;
    potential_position.x = position.x + current_speed * run_multiplier
    * dt * cos(rotation_x - angle_offset);
    potential_position.y = position.y + current_speed * run_multiplier
    * dt * sin(rotation_x - angle_offset);

    v2i ones(1, 1);
    v2i predicted_cell = v2i(floor(potential_position.x),
    floor(potential_position.y));

    v2i area_tl = min((v2i)position, predicted_cell);
    area_tl = max({ 0, 0 }, area_tl - ones); // clamp

    v2i area_br = max((v2i)position, predicted_cell);
    area_br = min({ map.width, map.height }, area_br + ones); // clamp

    for (int cell_x = area_tl.x; cell_x <= area_br.x; cell_x++)
    {
        for (int cell_y = area_tl.y; cell_y <= area_br.y; cell_y++)
        {

            // is wall
            if (map.getCell(cell_x, cell_y) == 1)
```

player.cpp

```
{  
    v2f nearest_point;  
    nearest_point.x = std::max(float(cell_x),  
        std::min(potential_position.x, float(cell_x + 1)));  
    nearest_point.y = std::max(float(cell_y),  
        std::min(potential_position.y, float(cell_y + 1)));  
  
    v2f ray_to_nearest = nearest_point -  
        potential_position;  
  
    float overlap = body_radius - mag(ray_to_nearest);  
  
    if (std::isnan(overlap)) overlap = 0;  
  
    if (overlap > 0)  
    {  
        potential_position -= norm(ray_to_nearest) *  
            overlap;  
    }  
}  
  
}  
  
position = potential_position;  
}  
  
// returns distance in that direction  
Player::HitInfo Player::shootRay(float angle_offset)  
{  
    //angle_offset = angle_offset * (tan(angle_offset));  
  
    v2f ray_dir(cos(rotation_x + angle_offset), sin(rotation_x +  
        angle_offset));  
  
    v2f ray_unit_step_size;  
    ray_unit_step_size.x = sqrt(1 + (ray_dir.y / ray_dir.x) *  
        (ray_dir.y / ray_dir.x));  
    ray_unit_step_size.y = sqrt(1 + (ray_dir.x / ray_dir.y) *  
        (ray_dir.x / ray_dir.y));  
  
    v2i current_cell(position);  
  
    v2f ray_length;  
    v2i cell_step;  
  
    if (ray_dir.x < 0)  
    {  
        cell_step.x = -1;  
        ray_length.x = (position.x - (float)current_cell.x) *  
            ray_unit_step_size.x;  
    }  
    else
```

player.cpp

```
{  
    cell_step.x = 1;  
    ray_length.x = (float(current_cell.x + 1) - position.x) *  
        ray_unit_step_size.x;  
}  
  
if (ray_dir.y < 0)  
{  
    cell_step.y = -1;  
    ray_length.y = (position.y - (float)current_cell.y) *  
        ray_unit_step_size.y;  
}  
else  
{  
    cell_step.y = 1;  
    ray_length.y = (float(current_cell.y + 1) - position.y) *  
        ray_unit_step_size.y;  
}  
  
float distance = 0;  
bool tile_found = false;  
bool latest_hit_on_x;  
while (!tile_found)  
{  
    if (ray_length.x < ray_length.y)  
    {  
        current_cell.x += cell_step.x;  
        distance = ray_length.x;  
        latest_hit_on_x = false;  
        ray_length.x += ray_unit_step_size.x;  
    }  
    else  
    {  
        current_cell.y += cell_step.y;  
        distance = ray_length.y;  
        latest_hit_on_x = true;  
        ray_length.y += ray_unit_step_size.y;  
    }  
}  
  
if (current_cell.x < 0 || current_cell.y < 0 || current_cell.x  
    >= map.width || current_cell.y >= map.height)  
    return { -1 };  
  
//hit wall  
if (map.getCell(current_cell.x, current_cell.y) == 1)  
{  
    v2f hit_position = position + ray_dir * distance;  
    //getting the x offset on the texture  
    float texture_x;  
  
    if (latest_hit_on_x)  
    {  
        ...  
    }  
}
```

player.cpp

```
    texture_x = hit_position.x - floor(hit_position.x);
    //if (sin(rotation_x) > 0) texture_x = 1 - texture_x;
}
else
{
    texture_x = hit_position.y - floor(hit_position.y);
    //if (cos(rotation_x) < 0) texture_x = 1 - texture_x;
}

    return { distance, latest_hit_on_x, texture_x };
}

}

void Player::drawObject(Object& object, float dt)
{
    float projection_distance = 0.5f / tan(fov_y / 2);

    // object's direction relative to ours.
    float angle_to_object = atan2(position.y - object.position.y,
        object.position.x - position.x) + rotation_x;

    float projection_position = 0.5f * tan(angle_to_object) / tan(fov_x
        / 2);

    float screen_x = WIDTH * (0.5f - projection_position);

    float object_distance = sqrt(pow(position.x - object.position.x, 2)
        + pow(position.y - object.position.y, 2));
    float screen_size = HEIGHT * projection_distance / (object_distance
        * cos(angle_to_object));

    screen_size *= object.scale_by;

    if (screen_size < 0) // behind us
        return;

    v2f player2object = position - object.position;
    v2f object_direction = object.position +
    v2f(cos(object.rotation_x), sin(object.rotation_x));
    //float direction_offset = angleBetweenVectors(object_direction,
    player2object) * TO_DEGREES;// + 22.5f;
    float direction_offset = 360 + 90 + TO_DEGREES * (object.rotation_x
        + atan2(object.position.x - position.x, object.position.y - position.y));

    object.direction_index = ((int)round(direction_offset / 45) % 8 +
        8) % 8;
```

player.cpp

```
object.animate(dt);

object.sprite.setScale(screen_size * object.shrink_by, screen_size
* object.shrink_by);
object.sprite.setPosition(
    screen_x - 0.5f * object.sprite.getTextureRect().height *
    object.sprite.getScale().x,
    map.floor_level - 0.5f * object.sprite.getTextureRect().height
    * (object.sprite.getScale().y - screen_size * (1 - object.shrink_by)));

window.draw(object.sprite);

if (object.dead) // no nametag on dead guys
    return;

// nametag
sf::Text nametag(object.username, nametag_font, 16);
nametag.setFillColor(sf::Color::White);
nametag.setPosition(screen_x - 0.25f *
nametag.getLocalBounds().width * object.sprite.getScale().x,
    map.floor_level - 0.17f * object.sprite.getTextureRect().height
    * object.sprite.getScale().y);
nametag.setScale(screen_size, screen_size);
window.draw(nametag);

}

void Player::shootRays(Player::HitInfo*& hits)
{
    // For each column of pixels
    for (int x = 0; x < WIDTH; x++)
    {
        float y = x / float(WIDTH - 1) - 0.5f;
        float angle = atan(2 * y * tan(fov_x / 2));
        HitInfo hit_info = shootRay(angle);

        hits[x] = hit_info;
        hits[x].perceived_distance = hit_info.distance * cos(angle);
    }
}

void Player::drawWorld(HitInfo*& hits, float dt)
{
```

player.cpp

```
//Sort objects and get the distances
bool column_drawn[WIDTH];
for (int x = 0; x < WIDTH; x++)
    column_drawn[x] = false;

// sort from farthest to closest (descending distance)
int i;
for (i = 0; i < sorted_objects.size(); i++)
{
    bool perfect = true;

    for (int j = 0; j < sorted_objects.size() - 1; j++)
    {
        if (sorted_objects[j]->distFrom(position) <
            sorted_objects[j + 1]->distFrom(position))
        {
            perfect = false;

            Object* temp = sorted_objects[j];
            sorted_objects[j] = sorted_objects[j + 1];
            sorted_objects[j + 1] = temp;
        }
    }

    if (perfect)
        break;
}

/*cout << "Objects: ";
for (int i = 0; i < objects.size(); i++)
{
    cout << sorted_objects[i]->player_id << " ";
}
cout << "enough of objects\n";*/

for (Object* object : sorted_objects)
{
    float object_distance = object->distFrom(position);
    for (int x = 0; x < WIDTH; x++)
    {
        if (!column_drawn[x] && hits[x].distance > object_distance)
        {
            drawColumn(x, hits[x]);
            column_drawn[x] = true;
        }
    }

    drawObject(*object, dt);
}

for (int x = 0; x < WIDTH; x++)
```

player.cpp

```
if (!column_drawn[x])
    drawColumn(x, hits[x]);

}

void Player::drawColumn(int x, const Player::HitInfo& hit_info)
{
    float len = 1000 / hit_info.perceived_distance;

    if (hit_info.on_x_axis)
        wall_sprite.setTexture(wall_txs[1]);
    else
        wall_sprite.setTexture(wall_txs[0]);

    // drawing
    wall_sprite.setTextureRect(sf::IntRect(
        v2i(hit_info.texture_x * wall_txs[0].getSize().x, 0), v2i(1,
        wall_txs[0].getSize().y)
    ));
    wall_sprite.setScale(1, len / wall_txs[0].getSize().y);
    wall_sprite.setPosition(x, map.floor_level - len / 2);
    window.draw(wall_sprite);
}

void Player::drawCrosshair(float dt)
{
    if (!dead)
    {
        sf::RectangleShape rect(v2f(4, 12));
        rect.setFillColor(sf::Color::Cyan);

        rect.setPosition(v2f(WIDTH / 2 - 2, HEIGHT / 2 - 16));
        window.draw(rect);

        rect.setPosition(v2f(WIDTH / 2 - 2, HEIGHT / 2 + 4));
        window.draw(rect);

        rect.setSize(v2f(12, 4));
        rect.setPosition(v2f(WIDTH / 2 - 16, HEIGHT / 2 - 2));
        window.draw(rect);

        rect.setPosition(v2f(WIDTH / 2 + 4, HEIGHT / 2 - 2));
        window.draw(rect);
    }

    // damage direction indicator

    for (int i = 0; i < hit_direction_timers.size(); i++)
    {
        hit_direction_timers[i] += dt;
```

player.cpp

```
if (hit_direction_timers[i] > 0.4f) // timer done
{
    hit_direction_timers.erase(hit_direction_timers.begin() + i);
    hit_direction_angles.erase(hit_direction_angles.begin() + i);

    i--;
}
else
{
    hit_indicator_sprite.setRotation(180 + (+ hit_direction_angles[i] - rotation_x) * TO_DEGREES);
    window.draw(hit_indicator_sprite);
}

// reticle
if (reticle_timer >= 0)
{
    reticle_timer += dt;

    if (reticle_timer > 0.06f)
        reticle_timer = -1;
    else
        window.draw(reticle_sprite);
}

if (current_damage_opacity > 0)
{
    damage_overlay_sprite.setColor(sf::Color(255, 255, 255,
(int)current_damage_opacity));
    window.draw(damage_overlay_sprite, sf::BlendAlpha);

    if (!dead)
        current_damage_opacity -= dt * 100;
}

void Player::loadTextures()
{
    // walls
    wall_txs = new sf::Texture[2];

    sf::Texture wall;
    wall.loadFromFile("sprites/1L.png");
    wall_txs[0] = wall;
    wall.loadFromFile("sprites/1D.png");
    wall_txs[1] = wall;
```

player.cpp

```
// gun
gun_texs = new sf::Texture[5]();

for (int i = 0; i < 5; i++)
{
    string path = "sprites/gun_animation/gun_X.png";
    path[path.find('X')] = i + '0';
    gun_texs[i].loadFromFile(path);
}

gun_sprite.setTexture(gun_texs[0]);
gun_sprite.setScale(0.8, 0.8);
gun_position = { WIDTH / 2 - gun_texs[0].getSize().x *
    gun_sprite.getScale().x / 2 + 25,
    HEIGHT - gun_texs[0].getSize().y * gun_sprite.getScale().x + 30 };
gun_sprite.setPosition(gun_position);

gun_animation_timer = 0;
gun_movement_stopwatch = 0;

gun_animation_duration = new float[5];
gun_animation_duration[1] = 0.08f;
gun_animation_duration[2] = 0.12f;
gun_animation_duration[3] = gun_animation_duration[4] = 0.2f;

enemy_tex.loadFromFile("sprites/spritesheet2.png");

indicator_texture.loadFromFile("sprites/indicator.png");
hit_indicator_sprite = sf::Sprite(indicator_texture);

hit_indicator_sprite.setScale(0.7f, 0.7f);

hit_indicator_sprite.setPosition(
    WIDTH / 2,
    HEIGHT / 2);

hit_indicator_sprite.setOrigin(
    hit_indicator_sprite.getLocalBounds().width / 2,
    hit_indicator_sprite.getLocalBounds().height / 2);

reticle_texture.loadFromFile("sprites/hit_marker2.png");
reticle_sprite = sf::Sprite(reticle_texture);
reticle_sprite.setPosition(
    WIDTH / 2,
    HEIGHT / 2);

reticle_sprite.setOrigin(
    reticle_sprite.getLocalBounds().width / 2,
    reticle_sprite.getLocalBounds().height / 2);

reticle_sprite.setScale(0.6f, 0.6f);
```

player.cpp

```
damage_overlay_tex.loadFromFile("sprites/getting-hit-overlay.png");
damage_overlay_sprite.setTexture(damage_overlay_tex);
damage_overlay_sprite.setScale(1.3, 1.3);
damage_overlay_sprite.setOrigin(WIDTH / 2, HEIGHT / 2);
damage_overlay_sprite.setPosition(WIDTH / 2, HEIGHT / 2);

//font
if (!nametag_font.loadFromFile("Fonts/Roboto-Regular.ttf"))
{
    cout << "Couldn't Find Nametag Font\n";
}
if (!deathscreen_font.loadFromFile("Fonts/Roboto-Light.ttf"))
{
    cout << "Couldn't Find deathscreen Font\n";
}
if (!bold_font.loadFromFile("Fonts/Roboto-Medium.ttf"))
{
    cout << "Couldn't Find bold Font\n";
}

void Player::loadSFX()
{
    gunshot_buffer.loadFromFile("sfx/9mm-pistol.wav");
    gun_sound.setBuffer(gunshot_buffer);
    gun_sound.setVolume(10);

    gunclick_buffer.loadFromFile("sfx/handgun-release.wav");
    click_sound.setBuffer(gunclick_buffer);
    click_sound.setVolume(25);
}

// draws the hand holding the gun at the bottom of the screen
// dt - deltaTime, the time in seconds since the beginning of the last
frame
void Player::drawGun(float dt)
{
    gun_animation_timer += dt;
    gun_movement_stopwatch += dt * 8;

    if (moving)
        hand_move_range = lerp(hand_move_range, max_hand_range, 0.14f);
    else
        hand_move_range = lerp(hand_move_range, 0, 0.06f);

    float hand_x = sin(gun_movement_stopwatch) * hand_move_range;
    float hand_y = 0.2f * cos(gun_movement_stopwatch) *
cos(gun_movement_stopwatch) * hand_move_range;

    if (dead)
```

player.cpp

```
{  
    gun_offset_y = lerp(gun_offset_y, 300, 0.2f);  
}  
else  
    gun_offset_y = lerp(gun_offset_y, hand_y, 0.1f);  
  
gun_offset = { hand_x , gun_offset_y };  
  
gun_sprite.setPosition(gun_position + gun_offset);  
  
//cout << "\n";  
window.draw(gun_sprite);  
  
// if frame bigger than zero, we animating  
if (gun_animation_frame && gun_animation_timer >=  
gun_animation_duration[gun_animation_frame])  
{  
    gun_animation_frame = (gun_animation_frame + 1) % 5;  
    gun_animation_timer = 0;  
    gun_sprite.setTexture(gun_texs[gun_animation_frame]);  
}  
  
}  
  
void Player::drawDeathScreen(float dt)  
{  
    if (!dead) return;  
  
    string main_string = killer_name + " killed you";  
    sf::Text main(main_string, bold_font, 80);  
    main.setOrigin(main.getLocalBounds().width / 2,  
    main.getLocalBounds().height / 2);  
    main.setPosition(WIDTH / 2, HEIGHT / 2 - 30);  
    main.setOutlineColor(sf::Color::Black);  
    main.setOutlineThickness(2);  
    main.setFillColor(sf::Color(200, 30, 20));  
    window.draw(main);  
  
    string sub_string = "press space to respawn";  
    sf::Text sub(sub_string, deathscreen_font, 30);  
    sub.setOrigin(sub.getLocalBounds().width / 2,  
    sub.getLocalBounds().height / 2);  
    sub.setPosition(WIDTH / 2, HEIGHT / 2 + 60 );  
    window.draw(sub);  
}  
  
void Player::shootGun(bool left_click)  
{  
    if (dead)  
        return;
```

player.cpp

```
//if right click or gun is animating, don't shoot
if (!left_click || gun_animation_frame)
{
    click_sound.play();
    return;
}

gun_shot = true;

gun_sound.play();

gun_animation_frame = 1;
gun_sprite.setTexture(gun_texs[gun_animation_frame]);
gun_animation_timer = 0;

}

void Player::updateServer()
{

std::lock_guard<std::mutex> lock(mtx);

Client::PlayerInfo player_info = getPlayerInfo();

void* buffer = malloc(sizeof(player_info) + received_events_size);
int buffer_size = sizeof(player_info) + received_events_size;
memcpy(buffer, &player_info, sizeof(player_info));

if (received_events_size)
{
    memcpy((char*)buffer + sizeof(player_info), received_events,
    received_events_size);

    received_events_size = 0;
}

string error;
if (!client.sendEncryptedUDP(buffer, buffer_size, error))
{
    cout << "couldn't send udp because: " << error << '\n';
    free(buffer);
    return;
}

free(buffer);
}
```

player.cpp

```
void Player::listenToServer()
{
    while (!has_quit)
    {
        string error;
        void* buffer;
        int buffer_size;
        if (!client.recvEncryptedUDP(buffer, buffer_size, error))
        {
            continue;
        }

        //cout << "Got This UDP (with size " << buffer_size << "): ";
        //printBytes(buffer, buffer_size);

        std::lock_guard<std::mutex> lock(mtx);

        char player_count = *(char*)buffer;

        int other_players_count = player_count - 1;
        if (other_players_count != objects.size())
        {

            objects.resize(other_players_count);
            sorted_objects.resize(other_players_count);

            for (int i = 0; i < objects.size(); i++)
            {
                objects[i] = Object(-10, -10, enemy_tex);
                sorted_objects[i] = &objects[i];
            }
        }

        // 1 byte of player_count and then PlayerInfo structs one after
        // the other

        int events_byte_count = buffer_size - 1 - player_count *
        sizeof(Client::PlayerInfo);

        // update all others
        int object_index = 0;
        Client::PlayerInfo* current_info_buffer =
        (Client::PlayerInfo*)((char*)buffer + 1);
        for (int i = 0; i < player_count; i++, current_info_buffer++)
        //point to next buffer
    }
}
```

player.cpp

```
int current_player_id = current_info_buffer->player_id;

addToLeaderboard(current_player_id,
current_info_buffer->score, current_info_buffer->username);

if (current_player_id == client.player_id)
    continue;

objects[object_index].loadPlayerInfo(*current_info_buffer);

object_index++;
}

updateLeaderboard(-1);

if (events_byte_count > 0)
    handleEvents((char*)buffer + 1 + player_count *
sizeof(Client::PlayerInfo), events_byte_count);

free(buffer);
}

}

void Player::handleEvents(char* events, int events_size)
{
    bool all_events_intelligible = true;

    int event_size = 0;
    for (int index = 0; index < events_size; index += event_size)
    {
        event_size = *(events + index);

        //cout << "Event: ";
        //printBytes(events + index, event_size);

        int event_type = *(events + index + 1);

        if (event_type == 1) // shooting happened
        {
            int shooter_id = *(char*)(events + index + 2);
            int victim_id = *(char*)(events + index + 3);
            handle_shooting_victim(victim_id, shooter_id);

            if (shooter_id == client.player_id) // i shot
            {
                reticle_timer = 0; // start reticle
            }
        }

        else if (event_type == 2) // death happened
        {
```

player.cpp

```
int killer_id = *(char*)(events + index + 2);
int victim_id = *(char*)(events + index + 3);
handle_killing(killer_id, victim_id);

}

else if (event_type == 3) // new person joined
{
    continue;

    int new_guy_id = *(char*)(events + index + 2);
    char* new_guy_username = events + index + 3;

    addToLeaderboard(new_guy_id, 0, new_guy_username);

    if (new_guy_id == client.player_id) continue;

    Object* object = getObject(new_guy_id);
    if (object == nullptr)
    {
        cout << "player with id " << new_guy_id << " not
        found\n";
        all_events_intelligible = false;
        continue;
    }

    object->username = new_guy_username;

    toaster.toast(object->username + " has joined the lobby");

}

else if (event_type == 4) // username of someone
{
    continue;

    int player_id = *(char*)(events + index + 2);
    int score = *(char*)(events + index + 3);
    char* username = events + index + 4;

    addToLeaderboard(player_id, score, username);

    if (player_id == client.player_id) continue;

    cout << username << " is already here.\n";

    Object* object = getObject(player_id);
    if (object == nullptr)
    {
```

player.cpp

```
cout << "player with id " << player_id << " not
found\n";
all_events_intelligible = false;
continue;
}

//set the object to the new player
//object->player_id = player_id;
object->username = username;
}
else if (event_type == 5) // someone left
{
    continue;
    int player_id = *(events + index + 2);
    cout << "event 5\n";
    //remove from leaderboard
    for(int i = 0; i < leaderboard.size(); i++)
        if (leaderboard[i].player_id == player_id)
    {
        cout << "erasing\n";
        leaderboard.erase(leaderboard.begin() + i);
        break;
    }

if (player_id == client.player_id) continue;

Object* object = getObject(player_id);
if (object == nullptr)
{
    cout << "leaving player not found\n";
    all_events_intelligible = false;
    continue;
}

//toaster.toast(object->username + " has left the lobby");
}
else
{
    cout << "Unrecognized Event: ";
    printBytes(events + index, event_size);
    all_events_intelligible = false;
}

}

if (all_events_intelligible)
{
    received_events_size = events_size;
    memcpy(received_events, events, events_size);
}
```

player.cpp

```
}

void Player::getKilled(const string& killer_name)
{
    cout << "You got Killed\n";
    dead = true;
    this->killer_name = killer_name;
    current_damage_opacity = max_damage_opacity;
}

void Player::handle_killing(int killer_id, int victim_id)
{
    int verb_index = rand() % 9;
    string killer_name = getUsername(killer_id);
    toaster.toast(killer_name + " " + verbs[verb_index] + " " +
    getUsername(victim_id));

    if (client.player_id == killer_id)
        score++;

    updateLeaderboard(killer_id);

    if (victim_id == client.player_id)
    {
        getKilled(killer_name);
        return;
    }

    //someone else died
    Object* victim = getObject(victim_id);
    if (victim == nullptr)
    {
        cout << "victim not found\n";
        return;
    }

    victim->gotKilled();

}

void Player::handle_shooting_victim(int victim_id, int shooter_id)
{
    // you got shot
    if (victim_id == client.player_id)
    {
        getShot(shooter_id);
        return;
    }

    //someone else got shot
}
```

player.cpp

```
Object* victim = getObject(victim_id);
if (victim == nullptr)
{
    cout << "victim not found\n";
    return;
}

victim->gotShot();
}

void Player::getShot(int shooter_id)
{
    Object* shooter = getObject(shooter_id);
    if (shooter == nullptr)
    {
        cout << "shooter not found\n";
        return;
    }

    // turn on hit direction
    float relative_angle = atan2(position.y - shooter->position.y,
    position.x - shooter->position.x);
    hit_direction_timers.push_back(0);
    hit_direction_angles.push_back(relative_angle);

    // turn on damage overlay
    current_damage_opacity = max_damage_opacity;
}

Object* Player::getObject(int id)
{
    for (int i = 0; i < objects.size(); i++)
    {
        if (objects[i].player_id == id)
            return &objects[i];
    }
    return nullptr;
}

Object* Player::getAnyObject()
{
    for (int i = 0; i < objects.size(); i++)
    {
        if (objects[i].player_id == -1)
            return &objects[i];
    }
    return nullptr;
}

Client::PlayerInfo Player::getPlayerInfo()
{
    Client::PlayerInfo info = {
        client.player_id,
```

player.cpp

```
shootRay(0).distance,
position.x, position.y, rotation_x, rotation_y,
0,
score
};

info.flags = 0;
info.flags |= moving * Client::PlayerInfo::moving;
info.flags |= moving_forward * Client::PlayerInfo::forward;
info.flags |= gun_shot * Client::PlayerInfo::gun_shot;
info.flags |= has_quit * Client::PlayerInfo::quit;
info.flags |= dead * Client::PlayerInfo::dead;

if (client.username.size() > 15)
    cout << "username too long oh nooooooo\n";

strcpy_s(info.username, client.username.c_str());

// reset gun shot flag
gun_shot = false;

return info;

}

void Player::respawn()
{
    if (!dead)
        return;

    dead = false;

    float x, y;
    while (true)
    {
        x = (float)rand() / RAND_MAX * map.width;
        y = (float)rand() / RAND_MAX * map.height;
        if (map.getCell(x, y) == 0)
            break;
    }
    rotation_x = (float)rand() / RAND_MAX * 2 * PI;
    position = { x, y };
}

string Player::getUsername(int id)
{
    if (id == client.player_id)
    {
        return client.username;
    }
}
```

player.cpp

```
Object* victim = getObject(id);
if (victim == nullptr)
{
    cout << "Username not found\n";
    return "MISSING USERNAME 2";
}

return victim->username;
}

void Player::addToLeaderboard(int player_id, int score, const string& username)
{
    for (const auto& board : leaderboard)
        if (board.player_id == player_id)
            return;

    cout << "adding player " << player_id << " to leaderboard.\n";
    leaderboard.emplace_back(player_id, score, username);
}

void Player::updateLeaderboard(int killer_id)
{
    for (int i = 0; i < leaderboard.size(); i++)
    {
        if (leaderboard[i].player_id == killer_id)
        {
            leaderboard[i].score++;
            break;
        }
    }

    for (int i = leaderboard.size() - 1 - 1; i >= 0; i--)
    {
        if (leaderboard[i].score < leaderboard[i + 1].score)
        {
            Toaster::LeaderboardEntry temp = leaderboard[i];
            leaderboard[i] = leaderboard[i + 1];
            leaderboard[i + 1] = temp;
        }
    }
}

void Player::debug()
{
    cout << position.x << ": X\n"
        << position.y << ": Y\n"
        << map.floor_level << ": map floor level\n\n";

    debug_mode ^= 1;
}
```

player.hpp

```
#pragma once

#include "headers.hpp"
#include "map.hpp"
#include "object.hpp"
#include "client.hpp"
#include "toaster.hpp"
#include <SFML/Audio.hpp>

class Player
{
private:
    // game logic
    sf::RenderWindow& window;
    Toaster& toaster;
    bool has_quit;
    bool dead = false;

    sf::Texture* wall_texs;
    sf::Sprite wall_sprite;

    sf::Texture enemy_tex;
    vector<Object> objects;
    vector<Object*> sorted_objects;

    // gun animation
    int gun_animation_frame;
    float gun_animation_timer, gun_movement_stopwatch;
    float* gun_animation_duration;
    sf::Sprite gun_sprite;
    sf::Texture* gun_texs;
    v2f gun_position;
    float gun_offset_y = 0;
    v2f gun_offset;
    float max_hand_range = 40;
    float hand_move_range;

    string verbs[9] = { "zoinked", "zooked", "styled on",
        "slaughtered", "kassified on", "nuked",
        "eradicated", "decimated", "pulverized" };

    // hit direction and reticle indicator
    sf::Texture indicator_texture, reticle_texture;
    sf::Sprite hit_indicator_sprite, reticle_sprite;
    vector<float> hit_direction_timers;
    vector<float> hit_direction_angles;
    float reticle_timer = -1;

    //gun shot
```

player.hpp

```
bool gun_shot;
sf::Texture damage_overlay_tex;
sf::Sprite damage_overlay_sprite;
float current_damage_opacity;
float max_damage_opacity = 130;
string killer_name;

// movement
v2f position;
float speed = 2.0f;
bool running = false, crouching = false;
bool moving = false, moving_forward;
float run_multiplier = 1.75f, crouch_multiplier = 0.5f;

// orientation
float rotation_x = -3.169f;
float rotation_y = -0.56f;
float mouse_sensitivity = 0.05f;
float fov_y = 0.7f;
float fov_x = 1.22173f; // 70 degrees

// map
float body_radius = 0.4f;

//sound
sf::SoundBuffer gunshot_buffer, gunclick_buffer;
sf::Sound gun_sound, click_sound;

//font
sf::Font nametag_font, bold_font, deathscreen_font;

//debug
float debug_float = 0;

int received_events_size = 0;
char received_events[128];

int score = 0;

public:
Client client;
std::mutex mtx;
Map map;
bool window_focused;

// leaderboard
vector<Toaster::LeaderboardEntry> leaderboard;

bool debug_mode = false;

struct HitInfo {
    float distance;
    bool on_x_axis;
```

player.hpp

```
float texture_x;

    float perceived_distance;
};

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);

//debug
```

player.hpp

```
void debug( );
```

```
};
```

server_address.txt

```
# line 1: this. line 2: server ip address. line 3: tcp port. line 4:  
udp port  
87.71.155.68  
21567  
21568
```

server.py

```
import socket
import threading
import time

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
from cryptography.hazmat.backends import default_backend
import secrets # for randbits, Diffie Hellman

from sql_orm import Users_db

import struct
import math

# Player data structure
class Player:
    def __init__(self, username, id):
        self.username = username
        self.player_id = id
        self.dist2wall = -1
        self.position_x = -1
        self.position_y = -1
        self.rotation_x = -1
        self.rotation_y = -1

        self.health = 100
        self.dead = False
        self.score = 0

        self.last_message_time = None

        self.events = []

    def update_events(self, received_events: bytes):
        index = 0
        while index < len(received_events):
            event_size = received_events[index]
            event = received_events[index:index + event_size]
            if(event in self.events):
                self.events.remove(event)
            index += event_size

    def add_event(self, event: bytes):
        self.events.append(event)

# Define server address and port
TCP_PORT = 3000 # Separate port for TCP communication
UDP_PORT = 3001
players:list[Player] = []
struct_format = 'ifffffii16s'
struct_size = struct.calcsize(struct_format)
player_binaries = b'\x00'
```

server.py

```
current_player_id = 0
key_bytes : dict = {}

# gets socket and string to send
def send(client, msg: str):
    send_bytes(client, msg.encode())

def send_bytes(client, msg: bytes):
    msg_length = len(msg) # get length int
    msg_length = msg_length.to_bytes(2, byteorder='little') # int to byte
    full_message = msg_length + msg # append msg length to msg
    client.send(full_message)

# returns msg bytes
def recvfrom(client) -> bytes:

    try:
        msg_length_bytes = client.recv(2)

        if len(msg_length_bytes) != 2:
            raise ConnectionError("Wrong message length received")

        msg_length = int.from_bytes(msg_length_bytes,
                                    byteorder='little')

        # Receive the actual message data
        message = client.recv(msg_length)

        if len(message) != msg_length:
            raise ConnectionError("Incomplete message received")

    return message

    except socket.error as e:
        raise ConnectionError(f"Socket error while receiving data: {e}") from e

def send_UDP(socket:socket.socket, address, msg: bytes):
    socket.sendto(msg, address)

# returns msg bytes
def recvUDP(udp_server : socket.socket) -> tuple[bytes, any]:

    try:
        msg_bytes, address = udp_server.recvfrom(256)

        # Receive the actual message data
        #if len(msg_bytes) != 33:
        #    raise ConnectionError("Incomplete message received (len:"


```

server.py

```
{0})", len(msg_bytes))

return msg_bytes, address

except socket.error as e:
    raise ConnectionError(f"Socket error while receiving data:
{e}") from e

# # Function to broadcast game state to all players
# def broadcast_game_state(players):
#     # Prepare game state data (replace with your game state
# representation)
#     game_state = [player.to_dict() for player in players]
#     data = pickle.dumps(game_state)

#     for player in players.values():
#         player.udp_socket.sendto(data, (SERVER_ADDRESS, address[1]))

def key_to_bytes(key):
    s = b''
    for i in range(16):
        s = int.to_bytes(key & 255, 1) + s
        key >>= 8

    return s

def pad_bytes(message_bytes: bytes, block_size):
    padding_size = block_size - (len(message_bytes) % block_size)
    padding = bytes([padding_size]) * padding_size
    return message_bytes + padding

def clean_bytes(dirty: bytes):
    return ' '.join([format(byte, '02X') for byte in dirty])

def unpad_bytes(padded_bytes: bytes):

    padding_size = padded_bytes[-1] # Get the last byte, which
    represents the padding size
    if padding_size == 0 or padding_size > len(padded_bytes):
        raise ValueError(f"Invalid padding, looking at final char")

    # Verify that the padding bytes are all the same
    expected_padding = bytes([padding_size]) * padding_size
    if not padded_bytes.endswith(expected_padding):
        raise ValueError(f"Invalid padding, looking at the last
{padding_size} bytes")

    # Remove the padding bytes
    unpadded_bytes = padded_bytes[:-padding_size]
    return unpadded_bytes

def encrypt_AES(plaintext: bytes, key):
```

server.py

```
blocks = pad_bytes(plaintext, 16)
backend = default_backend()
cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
encryptor = cipher.encryptor()
ciphertext = encryptor.update(blocks) + encryptor.finalize()
return ciphertext

def decrypt_AES(cipherbytes: bytes, key):
    backend = default_backend()
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(cipherbytes) + decryptor.finalize()
    return unpad_bytes(plaintext)

def add_player(username):
    global players, player_binaries, current_player_id

    new_guy = Player(username, current_player_id)
    players.append(new_guy)

    player_binaries += int.to_bytes(current_player_id, 1) * struct_size
    current_player_id += 1

    player_binaries = int.to_bytes(len(players), 1) +
    player_binaries[1:]

    someone_joined(new_guy) # notify all players that this guy joined

    # tell all players about all players
    for player in players:
        give_usernames(player)

def remove_player(player_id):
    global player_binaries, players

    # Iterate through the players list
    for player in players:
        # Check if the current player's username matches the target
        # username
        if player.player_id == player_id:
            # Remove the player from the list
            players.remove(player)
            break
    else:
        # If the player with the specified username is not found
        print(f"Player with id #{player_id} not found.")
        return

    # get index of this players buffer
```

server.py

```
buffer_index = 1
while True:
    if buffer_index >= len(player_binaries):
        print("Something fucked when looking for this player in the
binaries", buffer_index)
        return None

    if player_binaries[buffer_index] == player_id:
        break

    buffer_index += struct_size

# remove this players buffer from the binaries
player_binaries = player_binaries[:buffer_index] +
player_binaries[buffer_index+struct_size:]

player_binaries = int.to_bytes(len(players), 1) +
player_binaries[1:]

event = int.to_bytes(5, 1) # someone left
event += int.to_bytes(player_id, 1) # user id
send_event(event)

print(f"Player with id #{player_id} removed successfully.")

def handle_client(client_socket, address, users_db:Users_db, lock:
threading.Lock):
    global players, key_bytes, current_player_id

    # Diffie Hellman
    (p, g) = 170141183460469231731687303715884105757,
    340282366920938463463374607431768211507
    secret = secrets.randbits(128)

    # X2
    x2 = pow(g, secret, p)
    send(client_socket, 'X' + str(x2) + 'X')

    # X1
    x1 = recvfrom(client_socket).decode()
    if(not (x1[0] == 'X' and x1[-1] == 'X')):
        print("Incorrect X1 received. disconnecting client")
        send(client_socket, "ERROR")

    x1 = int(x1[1:-1])

    # Encryption
```

server.py

```
# Key
key = pow(x1, secret, p)

current_key_bytes = key_to_bytes(key)

#print("key_bytes: " + str(key_bytes))

cipherbytes = recvfrom(client_socket)
#print("cipherbytes: " + ciphertext.hex())

message = decrypt_AES(cipherbytes, current_key_bytes)

#print("got this: " + str(message))

parts = message.decode().split('~')
print(parts)

# users_db.insert_new_user(parts[1], parts[2])
# users_db.remove_user(parts[1])

lock.acquire()

response = "ERROR~Message Unrecognized"
if(parts[0] == "LOGIN"):
    if(users_db.user_exists(parts[1], parts[2])):
        response = "SUCCESS~" + str(current_player_id) + "~"
        key_bytes[current_player_id] = current_key_bytes

        add_player(parts[1])

    else:
        response = "FAIL~Username Or Password Incorrect~"

elif(parts[0] == "SIGNUP"):
    if(users_db.username_exists(parts[1])):
        response = "FAIL~Username Already Exists~"
    elif(users_db.insert_new_user(parts[1], parts[2])):
        response = "SUCCESS~"
    else:
        response = "FAIL~Can't Add User For Some Reason~"

lock.release()
#print("sending this: " + ''.join([format(byte, '02X') for byte
in cipherbytes]))
print(f"{{response=}}")
send_bytes(client_socket, encrypt_AES(response.encode(),
current_key_bytes))

client_socket.close()
```

server.py

```
def dot_product(x1, y1, x2, y2):
    return x1*x2 + y1*y2

def send_event(event: bytes):
    event = int.to_bytes(len(event)+1 , 1) + event # size
    for player in players:
        player.add_event(event)

# give the new player the usernames of the rest
def give_usernames(new_player : Player):
    for player in players:
        if player == new_player: continue

        print(f"telling {new_player.player_id} about
{player.player_id}'s username")

        event = int.to_bytes(4, 1) # username giving
        event += int.to_bytes(player.player_id, 1) # user id
        event += int.to_bytes(player.score, 1) #score
        event += player.username.encode() + b'\0' # username
        event = int.to_bytes(len(event)+1 , 1) + event # size

        print("sending ", clean_bytes(event))

        new_player.add_event(event)

# events: size byte, type byte, data bytes
def someone_joined(player: Player):
    event = int.to_bytes(3, 1) # new user event
    event += int.to_bytes(player.player_id, 1) # user id
    event += player.username.encode() + b'\0' # username

    send_event(event)

def someone_died(killer: Player, victim: Player):
    killer.score += 1

    event = int.to_bytes(2, 1) # killing
    event += int.to_bytes(killer.player_id, 1) # killer
    event += int.to_bytes(victim.player_id, 1) # killee
    send_event(event)

def someone_got_shot(shooter: Player, shootee : Player):
    if shootee.dead:
        return # can't shoot a dead person

    print(f"player {shooter.player_id} shot player
{shootee.player_id}")

    # event details what happened.
```

server.py

```
event = int.to_bytes(1, 1) # shooting
event += int.to_bytes(shooter.player_id, 1) # shooter
event += int.to_bytes(shootee.player_id, 1) # shootee
send_event(event)

# decrease health
shootee.health -= 40
if shootee.health <= 0:
    someone_died(shooter, shootee)

def player_distance(one:Player, two:Player):
    return math.sqrt((one.position_x - two.position_x)**2 +
    (one.position_y - two.position_y)**2)

def is_pointing_at(pointer: Player, pointee: Player):

    dir_x, dir_y = math.cos(pointer.rotation_x),
    math.sin(pointer.rotation_x)
    a = dot_product(dir_x, dir_y, dir_x, dir_y)

    qc_x, qc_y = pointee.position_x - pointer.position_x,
    pointee.position_y - pointer.position_y
    b = -2 * dot_product(dir_x, dir_y, qc_x, qc_y)

    r = 0.4
    c = dot_product(qc_x, qc_y, qc_x, qc_y) - r*r

    discy = b*b -4*a*c
    return discy > 0

def handle_gun_shot(player: Player):
    for i in range(len(players)):
        if players[i] == player: continue

        dist2other = player_distance(player, players[i])

        if dist2other < player.dist2wall and is_pointing_at(player,
players[i]):
            someone_got_shot(player, players[i])

def update_player(player_info: bytes):
    global players, player_binaries

    if(len(player_info) != struct_size):
        print("Invalid Player Info Received")
        return None

    player_id, dist2wall, pos_x, pos_y, rot_x, rot_y, flags, score,
    username_bytes = struct.unpack(struct_format, player_info)
```

server.py

```
for player in players:
    if player.player_id == player_id:
        break
else:
    print("Got message from nonexistent player, id=", player_id)
    return None

player.position_x = pos_x
player.position_y = pos_y
player.rotation_x = rot_x
player.rotation_y = rot_y
player.dist2wall = dist2wall

has_quit = flags & 8
if(has_quit):
    remove_player(player_id)
    return None

gun_shot = flags & 4
if(gun_shot):
    handle_gun_shot(player)

dead_flag = flags & 32
if(not dead_flag and player.dead): # player came back to life
    player.health = 100

player.dead = dead_flag

# get index of this players buffer
buffer_index = 1
while True:
    if buffer_index >= len(player_binaries):
        print("Something fucked when looking for this player in the
binaries", buffer_index)
        return None

    if player_binaries[buffer_index] == player_id:
        break

    buffer_index += struct_size

# update this players buffer in the binaries
player_binaries = player_binaries[:buffer_index] + player_info +
player_binaries[buffer_index+struct_size:]

player.last_message_time = time.time()

return player

def handle_game():
    global key_bytes, player_binaries
```

server.py

```
# UDP
udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
udp_socket.bind(('0.0.0.0', UDP_PORT))

while(True):
    msg, address = recvUDP(udp_socket)

    player_id, encrypted = msg[0], msg[1:]

    correct_key_bytes = key_bytes[player_id]

    try:
        decrypted = decrypt_AES(encrypted, correct_key_bytes)
        player_info = decrypted[:struct_size]
        received_events = decrypted[struct_size:]

    except Exception as e:
        print(f"invalid message from {address}, error: {e}")
        print(f"encrypted:", clean_bytes(encrypted))
        print()
        continue

    player = update_player(player_info)

    if(player == None):
        continue

    # print(f"Sending Player #{player_id} These Binaries:",
    # player_binaries[0])
    # for i in range(len(players)):
    #     print(f"player {i}.",
    # clean_bytes(player_binaries[1+struct_size*i:1+struct_size*i+struct_size]))
    # print()

    #delete the events that the player already received
    player.update_events(received_events)

    response = player_binaries + b''.join(player.events)
    #player.events = b'' # reset events.

    udp_socket.sendto(encrypt_AES(response , correct_key_bytes),
address)

#print(end - start)
#print("boutta send UDP: " + ' '.join([format(byte, '02X') for
```

server.py

```
byte in others))
```

```
def remove_idles():
    while(True):
        now = time.time()
        for player in players:
            if(player.last_message_time and now -
               player.last_message_time > 2):
                print(f"player {player.player_id} is idle.
disconnected.")
                remove_player(player.player_id)
        time.sleep(2)
```

```
# Main server function
def main():
    global players
```

```
# TCP
SERVER_ADDRESS = ('0.0.0.0', TCP_PORT)
tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcp_socket.bind(SERVER_ADDRESS)
tcp_socket.listen(5)
```

```
users_db = Users_db()
lock = threading.Lock()
```

```
threading.Thread(target=handle_game).start()
```

```
threading.Thread(target=remove_idles).start()
# Threading for concurrent client handling
```

```
print("Server listening on", SERVER_ADDRESS)
```

```
while True:
    client_socket, address = tcp_socket.accept()
    threading.Thread(target=handle_client, args=(client_socket,
                                                 address, users_db, lock)).start()
```

```
if __name__ == "__main__":
    main()
```

sql.orm.py

```
import sqlite3
import hashlib
import secrets
import string

class Users_db:
    def __init__(self):
        self.conn = None
        self.cursor = None

    def open_DB(self):
        self.conn = sqlite3.connect('users.db')
        self.cursor = self.conn.cursor()

    def close_DB(self):
        self.conn.close()

    def commit(self):
        self.conn.commit()

    def create_table(self):
        self.open_DB()
        self.cursor.execute('''DROP TABLE Users''')
        self.cursor.execute('''CREATE TABLE Users (username VARCHAR(20)
UNIQUE NOT NULL, password VARCHAR(64), salt VARCHAR(5));'''')
        self.commit()
        self.close_DB()

    def insert_new_user(self, username: str, password: str) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT username FROM Users WHERE username
= ?", (username,))
        res = self.cursor.fetchone()
        if res: # Exists
            self.close_DB()
            return False

        salt = ''.join(secrets.choice(string.ascii_letters) for _ in
range(5))
        hash_pass =
        hashlib.sha256((password+salt).encode()).hexdigest()
        self.cursor.execute("INSERT INTO Users (username, password,
salt) VALUES (?, ?, ?);",
                           (username, hash_pass, salt))
        self.commit()
        self.close_DB()
        return True

    def remove_user(self, username) -> bool:
        self.open_DB()
        self.cursor.execute("SELECT username FROM Users WHERE username
= ?", (username,))
        res = self.cursor.fetchone()
```

sql.orm.py

```
if res == None:
    print("User Doesn't Exist, can't remove")
    self.close_DB()
    return False

self.cursor.execute("DELETE FROM Users WHERE username=?;",
                   (username,))
self.commit()
self.close_DB()
return True

def user_exists(self, username, password) -> bool:
    self.open_DB()
    self.cursor.execute("SELECT salt FROM Users WHERE username=?;",
                       (username,))
    salt = self.cursor.fetchone()
    if salt == None:
        self.close_DB()
        return False

    salt = salt[0] # db returns tuple of return values, we need the
    str itself

    hash_pass = hashlib.sha256((password +
salt).encode()).hexdigest()

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

    res = self.cursor.fetchone()
    self.close_DB()

    if res:
        return True
    return False

def username_exists(self, username) -> bool:
    self.open_DB()
    self.cursor.execute("SELECT username FROM Users WHERE
username=?;", (username,))
    res = self.cursor.fetchone()
    self.close_DB()
    if res:
        return True
    return False

def upgrade_to_pro(self, username):
    self.open_DB()
    self.cursor.execute("UPDATE Users SET is_pro = ? WHERE username
= ?;", (True, username))
    self.commit()
```

sql orm.py

```
self.close_DB( )

def get_email_from_username(self, username):
    self.open_DB()
    self.cursor.execute("SELECT email FROM Users WHERE username=?",
    (username, ))
    res = self.cursor.fetchone()
    self.close_DB()
    if not res:
        return ''
    return res[0]

def set_new_password(self, username, password):
    self.open_DB()
    hash_pass = hashlib.sha256(password.encode()).hexdigest()
    self.cursor.execute("UPDATE Users SET password = ? WHERE
    username = ?;", (hash_pass, username))
    self.commit()
    self.close_DB()
```

toaster.cpp

```
#include "toaster.hpp"
#include "tools.hpp"
#include "object.hpp"

Toaster::Toaster() : first_toast_position(870, -10), goal_y(-10)
{
    toast_tex.loadFromFile("sprites/toast.png");
    toast_sprite.setTexture(toast_tex);

    notch_tex.loadFromFile("sprites/LB_notch.png");
    notch_sprite.setTexture(notch_tex);
    notch_sprite.setPosition(leaderboard_position);
    notch_sprite.setScale(leaderboard_scale, leaderboard_scale);
    notch_sprite.setColor(sf::Color(255, 255, 255, 190));

    board_tex.loadFromFile("sprites/LB_player.png");
    board_sprite.setTexture(board_tex);
    board_sprite.setScale(leaderboard_scale, leaderboard_scale);
    board_sprite.setColor(sf::Color(255, 255, 255, 190));

    font.loadFromFile("Fonts/Roboto-Medium.ttf");

    text = sf::Text("Missing Text", font, 18);
    text.setFillColor(sf::Color(35, 35, 35));

    leaderboard_text = sf::Text("Missing Text", font, 18);
    leaderboard_text.setFillColor(sf::Color(35, 35, 35));
}

Toaster::LeaderboardEntry::LeaderboardEntry(int player_id, int score,
const string& username) :
    player_id(player_id), username(username), score(score),
    position_y(-100)
{
}

void Toaster::drawLeaderboard(sf::RenderWindow& window,
vector<LeaderboardEntry>& leaderboard, float dt)
{
    // empty leaderboard
    if (leaderboard.size() < 2) return;

    window.draw(notch_sprite);

    float initial_y = leaderboard_position.y + 14;
    for (int i = 0; i < leaderboard.size(); i++)
    {
        float desired_pos_y = initial_y + 34 * i;
```

toaster.cpp

```
leaderboard[i].position_y = lerp(leaderboard[i].position_y,
desired_pos_y, 0.1f);

}

for (int i = 0; i < leaderboard.size(); i++)
{
    board_sprite.setPosition(leaderboard_position.x,
    leaderboard[i].position_y);
    window.draw(board_sprite);

    //name
    leaderboard_text.setString(leaderboard[i].username);
    leaderboard_text.setPosition(board_sprite.getPosition() +
    leaderboard_name_offset);
    window.draw(leaderboard_text);

    //score

    leaderboard_text.setString(std::to_string(leaderboard[i].score));
    leaderboard_text.setPosition(board_sprite.getPosition() +
    leaderboard_score_offset);
    window.draw(leaderboard_text);
}

}

void Toaster::drawToasts(sf::RenderWindow& window, float dt)
{

    for (int i = 0; i < toast_timers.size(); i++)
    {
        toast_timers[i] -= dt;
        if (toast_timers[i] < 0)
        {
            goal_y -= 67;
            toast_timers.erase(toast_timers.begin() + i);
            i--;
        }
    }
}

first_toast_position.y = lerp(first_toast_position.y, goal_y,
0.05);

int amount = 0;

v2f toast_position = first_toast_position;
for(int i = 0; i < toasts.size(); i++)
{
    toast_slides[i] = lerp(toast_slides[i], 0, 0.3);
    v2f slide_offset(toast_slides[i], 0);
```

toaster.cpp

```
toast_sprite.setPosition(toast_position + slide_offset);

text.setString(toasts[i]);
text.setPosition(toast_position + text_position +
slide_offset);

if (toast_position.y + toast_sprite.getLocalBounds().height -
24 > 0)
{
    window.draw(toast_sprite);
    window.draw(text);

    amount++;
}

toast_position.y += 67;
}

void Toaster::toast(const string& text)
{
    cout << "Toasting: " << text << "\n";
    toasts.push_back(text);
    toast_slides.push_back(500);
    toast_timers.push_back(lifetime);
}
```

toaster.hpp

```
#pragma once
#include "headers.hpp"
#include "tools.hpp"

class Toaster
{
private:
    vector<string> toasts;
    vector<float> toast_slides;
    vector<float> toast_timers;

    v2f first_toast_position;
    float goal_y = 0;
    v2f toast_size = { 250, 80 };

    sf::Texture toast_tex;
    sf::Sprite toast_sprite;

    sf::Font font;
    sf::Text text;

    v2f text_position = { 36, 40 };

    float lifetime = 6; // in seconds

    // leaderboard

    sf::Texture notch_tex, board_tex;
    sf::Sprite notch_sprite, board_sprite;

    v2f leaderboard_position = { 20, 10 };
    float leaderboard_scale = 0.8f;

    sf::Text leaderboard_text;
    v2f leaderboard_name_offset = { 9, 6 };
    v2f leaderboard_score_offset = { 185, 6 };

public:
    struct LeaderboardEntry
    {
        int player_id;
        string username;
        int score;

        float position_y;

        LeaderboardEntry(int player_id, int score, const string& username);
    };

    Toaster();
    void drawToasts(sf::RenderWindow& window, float dt);
```

toaster.hpp

```
void drawLeaderboard(sf::RenderWindow& window,  
vector<LeaderboardEntry>& leaderboard, float dt);  
void toast(const string& text);  
};
```

tools.cpp

```
#include "headers.hpp"
#include "tools.hpp"

v2i min(const v2i& first, const v2i& second)
{
    return v2i(std::min(first.x, second.x), std::min(first.y,
    second.y));
}

v2i max(const v2i& first, const v2i& second)
{
    return v2i(std::max(first.x, second.x), std::max(first.y,
    second.y));
}

float mag(const v2f& vec)
{
    return sqrtf(vec.x * vec.x + vec.y * vec.y);
}

v2f norm(const v2f& vec)
{
    return vec / mag(vec);
}

float lerp(float a, float b, float t)
{
    return a * (1.0 - t) + (b * t);
}

sf::Color lerp(sf::Color c1, sf::Color c2, float t)
{
    return sf::Color(
        t * c1.r + (1 - t) * c2.r,
        t * c1.g + (1 - t) * c2.g,
        t * c1.b + (1 - t) * c2.b
    );
}

float angleBetweenVectors(const v2f& v1, const v2f& v2)
{
    float nigga = (v1.x * v2.x + v1.y * v2.y) / (mag(v1) * mag(v2));

    cout << "nig: " << (nigga) << " angle: ";
    if (nigga > 0)
        return acos(nigga);

    return acos(nigga);
}

bool inBounds(const v2f& box_pos, const v2f& box_size, const v2i& pos)
```

tools.cpp

```
{  
    if (pos.x < box_pos.x || pos.x > box_pos.x + box_size.x) {  
        return false;  
    }  
  
    // Check if the point's y coordinate is within the box's vertical  
    // bounds.  
    if (pos.y < box_pos.y || pos.y > box_pos.y + box_size.y) {  
        return false;  
    }  
  
    // If both checks pass, the point is within the box.  
    return true;  
}  
  
vector<string> split(const string& str)  
{  
  
    std::istringstream iss(str);  
    vector<string> tokens;  
    string token;  
  
    // Split the string by spaces and store each token in a vector  
    while (std::getline(iss, token, ' ')) {  
        tokens.push_back(token);  
    }  
  
    return tokens;  
}  
  
TextBox::TextBox(const v2f& pos, const v2f& size, const string& str,  
const sf::Font& font)  
: position(pos), size(size), text_string(str), text(str, font, 30),  
shadow(size), cursor(v2f(1.5f, 30))  
{  
    text.setFillColor(sf::Color::Black);  
    text.setPosition(position + text_offset);  
  
    shadow.setPosition(position);  
    shadow.setFillColor(sf::Color(0, 0, 0, 70));  
  
    cursor.setFillColor(sf::Color::Black);  
    cursor.setSize(v2f(1.5, 30));  
}  
  
string TextBox::getString()  
{  
    return text_string;  
}  
  
void TextBox::draw(sf::RenderWindow& window, bool is_focused)
```

tools.cpp

```
{  
    text.setString(text_string);  
  
    if (hidden)  
    {  
        string hashed(text_string.size(), '*');  
        text.setString(hashed);  
    }  
  
    window.draw(text);  
  
    if (!is_focused)  
    {  
        window.draw(shadow);  
        return;  
    }  
  
    if (cursor_visible)  
    {  
        cursor.setPosition(text.getPosition() +  
                           v2f(text.getGlobalBounds().getSize().x + 6, 4));  
        window.draw(cursor);  
  
    }  
  
    if (cursor_timer.getElapsedTime().asMilliseconds() > 500)  
    {  
        cursor_visible ^= true;  
        cursor_timer.restart();  
    }  
  
}  
  
void TextBox::turnOnCursor()  
{  
    cursor_visible = true;  
    cursor_timer.restart();  
}  
  
void TextBox::addText(const string& added_text)  
{  
    turnOnCursor();  
  
    text_string = text_string + added_text;  
    if (text_string.size() > 15)  
    {  
        text_string = text_string.substr(0, 15);  
    }  
}
```

tools.cpp

```
}
```

```
void TextBox::backspace(int backspace_counter)
{
    turnOnCursor();

    if (backspace_counter < 0) // ctrl backspace
    {
        text_string = "";
        return;
    }

    text_string = (text_string.substr(0, text_string.size() -
    backspace_counter));
}
```

```
void TextBox::clearText()
{
    text_string = "";
}
```

```
bool TextBox::inBox(const v2i& pos)
{
    return inBounds(position, size, pos);
}
```

tools.hpp

```
#pragma once
#include "headers.hpp"
#include <sstream>

v2i min(const v2i& first, const v2i& second);

v2i max(const v2i& first, const v2i& second);

float mag(const v2f& vec);

v2f norm(const v2f& vec);

float lerp(float a, float b, float t);

sf::Color lerp(sf::Color c1, sf::Color c2, float t);

bool inBounds(const v2f& box_pos, const v2f& box_size, const v2i& pos);

vector<string> split(const string& str);

float angleBetweenVectors(const v2f& v1, const v2f& v2);

struct TextBox
{
    v2f position, size;

    string text_string;
    sf::Text text;
    v2f text_offset = { 20, 16 };

    sf::RectangleShape shadow;

    sf::Clock cursor_timer;

    sf::RectangleShape cursor;
    bool cursor_visible = true;

    bool hidden = false;

    TextBox(const v2f& pos, const v2f& size, const string& str, const
sf::Font& font);

    void addText(const string& added_text);
    void backspace(int backspace_counter);
    void draw(sf::RenderWindow& window, bool is_focused);
    void clearText();

    string getString();
    bool inBox(const v2i& pos);

    void turnOnCursor();
}
```

tools.hpp

};