ספר פרויקט

מכונה לומדת משחק רשת PONG



מגיש: אדיר צורי כהן

ת.ז: 215322645

המנחה: שלמה וקנין

תאריך הגשה: קיץ תשפ"ג

ביה"ס: ישיבת אמי"ת עמיחי

תוכן

[מבוא: 3](#_Toc136881221)

[תיאור תכולת הספר 3](#_Toc136881222)

[הגדרת רשת ניורונים: 4](#_Toc136881223)

[הגדרת הלקוח: 6](#_Toc136881224)

[הגדרת יעדים: 6](#_Toc136881225)

[תיחום הפרויקט: 6](#_Toc136881226)

[אתגרים שאני צופה להם בפרויקט: 6](#_Toc136881227)

[הרקע לפרויקט: 6](#_Toc136881228)

[תהליך המחקר: 9](#_Toc136881229)

[שירותים שהשתמשתי בהם: 13](#_Toc136881230)

[ניהול זמן: 14](#_Toc136881231)

[אתגרים מרכזיים: 15](#_Toc136881232)

[האתגר הראשון: 15](#_Toc136881233)

[אתגר שני: 16](#_Toc136881234)

[אתגר שלישי: 18](#_Toc136881235)

[תיאור תחום הידע: 19](#_Toc136881236)

[פירוט יכולות: 20](#_Toc136881237)

[ארכיטקטורה: 22](#_Toc136881238)

[דיאגרמה: 24](#_Toc136881239)

[סביבת פיתוח: 26](#_Toc136881240)

[תיאור איומים: 28](#_Toc136881241)

[מדריך למשתמש + הסברת מסכים: 30](#_Toc136881242)

[מסך הפתיחה: 30](#_Toc136881243)

[מסך מנהל הלובי: 31](#_Toc136881244)

[מסך המשחק: 32](#_Toc136881245)

[מסך בחירת הלובים:. 33](#_Toc136881246)

[מסך הלקוח בתוך השרת: 34](#_Toc136881247)

[מימוש: 35](#_Toc136881248)

[מחלקה neural network: 35](#_Toc136881249)

[מחלקה onlineBall: 36](#_Toc136881250)

[תיאור המחלקה: 36](#_Toc136881251)

[תכונות: 36](#_Toc136881252)

[פעולות: 38](#_Toc136881253)

[המחלקה TestLobby: 40](#_Toc136881254)

[תיאור המחלקה: 40](#_Toc136881255)

[תכונות: 40](#_Toc136881256)

[פעולות: 43](#_Toc136881257)

[המחלקה TestRelay: 45](#_Toc136881258)

[תיאור המחלקה: 45](#_Toc136881259)

[תכונות: 46](#_Toc136881260)

[המחלקה setScorePlayerOne: 46](#_Toc136881261)

[תיאור המחלקה: 46](#_Toc136881262)

[תכונות: 47](#_Toc136881263)

[פעולות: 47](#_Toc136881264)

[המחלקה setScorePlayerTwo: 47](#_Toc136881265)

[תיאור המחלקה: 48](#_Toc136881266)

[תכונות: 48](#_Toc136881267)

[פעולות: 48](#_Toc136881268)

[המחלקה PlayerController: 48](#_Toc136881269)

[תיאור המחלקה: 48](#_Toc136881270)

[תכונות: 49](#_Toc136881271)

[פעולות: 52](#_Toc136881272)

[רפלקציה: 61](#_Toc136881273)

[ביבליוגרפיה: 62](#_Toc136881274)

[הדפס הקוד: 64](#_Toc136881275)

# מבוא:

## תיאור תכולת הספר

פרק המבוא יכלול את תיאור הרקע לפרויקט- מאיפה הרעיון נבע ועל איזה צורך נועד לתת מענה, תהליך המחקר המקדים- סקירת המצב הקיים בשוק, יתרונות וחסרונות וחידושים בפרויקט אשר נועדו לפתור את החסרונות לפתרונות הקיימים בשוק, אתגרים מרכזיים- קשיים ובעיות אשר נתקלתי במהלך כתיבת התכנית והיוו מכשול להתקדמות הפרויקט , והסיבות לבחירת נושא העבודה- מדוע בחרתי בנושא ומה משך אותי במיוחד ברעיון לפרויקט.

לאחר מכן, אעבור להסביר על מבנה וארכיטקטורת הפרויקט.

פרק זה יכלול את הצגת הפתרונות והסיבות לבחירתם, יחידות הפרויקט, תפקידה של כל יחידה, קלטי ופלטי היחידה, מבני הנתונים אשר נעשה בהם שימוש, זרימת המידע בין היחידות השונות, תיאור פרוטוקולי התקשורת, שרת- לקוח, תיאור האלגוריתם הראשי, הקשרים בין היחידות השונות, והצגת מקרה עבור הפונקציות העיקריות בפרויקט.

משם אציג את המדריך למשתמש אשר יכיל את אופן השימוש הנכון בתוכנה על מנת שהוא יוכל להפיק את המיטב ממנה.

חלק זה יכיל את הוראות ההתקנה ואילו קבצים או תיקיות נדרשים במחשב המשתמש. נוסף על כך חלק זה יכיל גם תרשים מסכים המתאר את היררכיית המסכים והמעברים בניהם, מה תפקידו של כל מסך בצירוף צילום מסך של החלון הרלוונטי, תיאור מסך הפתיחה- מה הוא מכיל ואת נקודות הניווט השונות, כל מסכי האפליקציה בליווי הסברים, והסבר על כל אלמנט תצוגה (כפתור, תיבת טקסט וכדו') את תפקידו, והודעות למשתמש.

בשלב הבא של הספר אתאר את המדריך למפתח אשר משמש הרחבה של המדריך למשתמש.

המדריך למפתח יציג עבור כל קובץ שקיים בפרויקט את: שם הקובץ, מחלקה ותפקידו, תפקיד המחלקה, מיקום הקובץ (בספרית תיקיות הפרויקט), תוכנו, הסבר על כל משתנה בקובץ, והסבר עבור הפונקציות בקובץ.

לאחר מכן, אכתוב סיכום אישי (רפלקציה) על החוויה האישית שלי בפרויקט.

בחלק זה אכתוב על חווית העבודה על הפרויקט, מה קיבלתי מהפרויקט, אילו כלים לקחתי איתי לעתיד מהפרויקט, אילו קשיים ואתגרים עמדו בפניי, מה המסקנות שלי מהתהליך שעברתי, מה המסקנות שלי, מה הייתי עושה אחרת לו הייתי מתחיל לעבוד על הפרויקט היום, מה היה יכול לשפר את יעילות העבודה, ועוד שאלות חקר עצמי.

חוץ מזה, אכתוב את הביבליוגרפיה.

בחלק זה של העבודה אציין את מקורות המידע (כדוגמת מאמרים, קישורים לאתרי מידע, ספרים וכתבי עת) אשר עשיתי בהם שימוש במהלך המחקר המקדים לפרויקט ובמהלך המחקר תוך כדי כתיבת הפרויקט.

בסיום הספר אכתוב את הנספחים בהם אוסיף הסברים על הטכנולוגיות אשר עשיתי בהם שימוש.

## הגדרת רשת ניורונים:

רשת נוירונים (Neural Network) היא דגם חישובי מתמקד באמירה שהתפקיד המרכזי שלו הוא ללמוד מהנתונים ולהבין מורכבות מבניתם. הרשת נוירונים מבוססת על מודל החשיבה ביונים נוירונים במערכת העצבים האנושית. הרעיון המרכזי מאחורי רשת נוירונים הוא לבנות מודל מתמיד של רשת נוירונים סינטטית שיכולה להיות מאומנת לזהות וללמוד דפוסים ומבנים בנתונים.

הרשת הנוירונית מורכבת מיחידות חישוב, הנקראות "נוירונים", שמשמשות כבניינים המחזיקים משתנים (משקלים) ומבצעים חישובים על הקלטים. כל נוירון מקבל קלטים מנוירונים אחרים, מחשב ערך יחיד ומעביר אותו כתוצאה (output) לנוירונים אחרים ברשת. המקומות שבהם מתקבלים הקלטים מנוירונים אחרים נקראים "שדות ראוי" (Receptive Fields). ברשת נוירונים, השדות הראויים של כל נוירון מתקשרים עם שדות הראויים של נוירונים אחרים ומורידים את המידע ברשת.

המשקלים בין הנוירונים ברשת מייצגים את התורה של המידע. כל קלט משוקלל באופן מסוים לפי המשקל המתא

ים, ומתקבלת התוצאה באמצעות פונקציה פעולה. על ידי כמה וכמה מידות קלט ומשקלים נקבל פונקציות ריבועיות חד-משתנות. לכן, רשת נוירונים יכולה לכלול פונקציות רב-משתנות מורכבות יותר.

המטרה העיקרית של רשת נוירונים היא ללמוד את הקישורים המופעלים בין הקלטים לבין התוצאות. המידע מאומן באמצעות תהליכי אימון שבהם הרשת מתאימה את המשקלים שלה כך שהתוצאות שהיא מחשבת יתאימו בצורה מיטבית לתוצאות הרצויות. התהליך של למידה זה מתבצע באמצעות אלגוריתמים כמו אחוז השגיאה הממוצעת (Mean Squared Error) והאחוז המבוגר (Backpropagation) שמתרגמים את השגיאות בין התוצאות המחושבות של הרשת לבין התוצאות הרצויות לתהליכי שינוי במשקלים.

רשתות נוירונים משמשות במגוון רחב של תחומים ויישומים, כולל תחומים כמו ראיית מחשב, תיאום ניידות, מילולון טבעי, מזהה פנים, מזהה קול, מינהל מערכות מותאמות אישית, ועוד. היכולת של הרשתות הנוירוניות ללמוד ולתקן את עצמן על ידי אימון נתונים ה

ופכת אותן לכלי חזק ביותר לניתוח ולעיבוד מידע מורכב.

בסיכום, רשתות נוירונים מהוות מודל מתמקד בחישוב המדמה חלקים ממערך העצבים האנושי. הן מתבססות על נוירונים סינטטיים שיכולים ללמוד מהנתונים ולהבין מורכבות בניהם. בעזרת תהליכי אימון, הרשת מתאימה את המשקלים שלה כדי להתאים בצורה מיטבית לתוצאות הרצויות. הרשתות הנוירוניות נמצאות בשימוש נרחב במגוון תחומים ויישומים ככלי חזק לניתוח ולעיבוד מידע מורכב.

## הגדרת הלקוח:

המשחק מיועד לכל משתמש שרוצה לשחק pong עם חבריו ללא צורך לפתוח שרת מיוחד.

## הגדרת יעדים:

שהמשחק יעבוד בזמן אמת.

שהלקוח לא אצטרך להסתבך עם כניסה לשרתים על ידי הקלדת כתובת ip.

קובץ התקנה אחד ללא צורך לפתוח פורטים בראוטר.

# תיחום הפרויקט:

הפרויקט עוסק בעיקר בhigh level networking, כלומר במה כורך משחק רשת אם התחשבות לזמן אמת, בעיות נפוצות, פתיחת פורטים ועוד.

## אתגרים שאני צופה להם בפרויקט:

לעבוד עם netocde פעם ראשונה, פיתחתי כבר משחקים ביוניטי אבל עכשיו אני מוסיף לזה עוד מימד, מימד של חישובים בזמן אמת וידע חדש שאני צריך ללמוד ולחקור כדי להצליח בפרויקט.

## הרקע לפרויקט:

את הרצון לעשות את הפרויקט הזה גיליתי כשאר UNITY הכריזה על השירותי רשת החדשים שלה, **Unity Gaming Services** באוקטובר 2021. משunity חידשה בעצם זה דרך מהירה ואיכותית ליצור משחק רשת שהוא גם cross platform, כלומר יוצרים את המשחק פעם אחת והוא יעבוד לכל הפלטפורמות האפשריות לשוק כגון: PlayStation 4, PlayStation 5; Xbox One, Xbox Series S/X, Android phones, IOS phone\ipads, macOS, Linux, windows and HTML5.

תמיד ידעתי שאני אעשה משחק וזה היה אופציה מדהימה להתחיל, אני משחק המון במשחקי מחשב מרובי משתתפים ותמיד סיקרן אותי איך עובד ד

השירותים שUnity מציעה הם:

1. Unity Multiplayer - זהו מנוע של Unity לפיתוח משחקי מרובי משתתפים באינטרנט, המאפשר יצירת שרתים וטכנולוגיות לניהול החיבורים בזמן אמת.

2. Unity Lobby - הוא כלי נוסף של Unity לפיתוח משחקי מרובי משתתפים ברשת. הוא מאפשר למפתחים ליצור לובי עבור המשחק שלהם, שבו השחקנים יכולים להתחבר ולצפות בפעילויות שונות קודם למשחק עצמו.

הלובי מאפשר לשחקנים לתקשר עם יחדיו, להתארגן למשחק ולהתחיל לשחק ביחד. בנוסף, הלובי יכול לכלול את האפשרות לשוחח בין השחקנים, לשתף קבצים כמו תמונות וסרטוני וידאו, ולהציג מידע על המשחק ועל השחקנים.

בכל מקרה, המטרה העיקרית של הלובי היא לאפשר לשחקנים להתחבר למשחק ולהתארגן לפני שהם מתחילים לשחק בצורה מאורגנת. זה מאפשר לשחקנים להיות יותר מעורבים ומקצועיים, ולשפר את החוויה הכללית של המשחק ברשת.

3. Unity Relay Servers - מספק שירותי רשת בזמן אמת עבור משחקי מרובי משתתפים באמצעות שרתי חיבור Relay ענן של Unity.

4. Unity Transport - זהו חבילת כלי המאפשרת למפתחים ליצור חיבורים עמיתיים עבור משחקי מרובי משתתפים, כולל תמיכה בטכנולוגיות רשת מסיביות ואלחוטיות.

5. Unity Analytics - כלי ניתוח נתונים המאפשר למפתחים לעקוב אחר ההתנהגויות של משתמשים במשחק, ולקבל מידע חשוב על הביצועים והשימושיות של המשחק במספר פלטפורמות שונות.

6. Unity Ads - מספק פלטפורמת פרסומות חכמה למפתחי משחקים, כולל מודלי רווח על פי תצוגות, לחיצות, או הורדות.

שימו לב כי זהו רק חלק מהשירותים המוצעים על ידי Unity בתחום המרובי משתתפים.

אני החלטתי להשתמש במגוון מן השירותים של unity על מנת לדמות עבודה על פרויקט אמיתי שאכן אין צורך להמציא את הגלגל מחדש אם חברה עם שם גדול בתחום משחררת ממשק נוח ואיכותי.

אני החלטתי שאני לא אצור משחק מסובך אלא אנסה להשתמש בהרבה מן השירותים של unity שאכן זה פלטפורמה שיש לי עבר איתה.

## תהליך המחקר:

איך החלטתי בסוף על unity gaming services?

ישנם פתרונות אחרות וטובות:

כאן הם שלושת מערכות הרשת הפופולריות ביותר ב-Unity:

1. Mirror Networking: מערכת זו היא פתרון קל ופשוט לצרכי רשת של משחקי Unity. זה מציע פלטפורמה קלה לשימוש, כוללת תמיכה לשרתים ולקונטרולרים. זה גם מציע ממשק גרפי לניהול ומיפוי רשתות.

2. Photon Networking: מערכת זו מספקת פלטפורמה עוצמתית למשחקי רשת של Unity. זה כולל תמיכה בשרתים וקונטרולרים עם ממשק API ברור. המערכת כוללת כלי מובנה לניתוח ומיפוי רשתות.

3. - FishNet היא עוד מערכת רשת פופולרית ביותר עבור משחקי Unity. זהו פתרון רשת עם כלי פיתוח נוח לשימוש ומציע פלטפורמה עוצמתית למשחקי רשת בעזרת טכנולוגיית משתמש-אפס (zero-configuration).

המערכת כוללת תמיכה מלאה במכשירי טלפון נייד ובדפדפנים, מה שמאפשר לשחקנים להתחבר ולהשתתף במשחקים רשת עם קלות רבה. המערכת כוללת כלי ניתוח רשת מובנה ומתאים לפיתוח משחקים למספר גדול של שחקנים במקביל.

בסיכום, FishNet היא מערכת רשת עוצמתית עם כלי פיתוח קל לשימוש ותמיכה מלאה במכשירי נייד ובדפדפנים. זה מאפשר למפתחים לפתח משחקי רשת במהירות ובקלות רבה.

בתיאור הזה קל ללכת לאיבוד ולכן עשיתי רשימה של יתרונות וחסרונות.

כן, הנה כמה יתרונות וחסרונות לכל אחד מחמשת פתרונות הרשת ל-Unity:

1. Photon Unity Networking (PUN)

יתרונות:

- קל לשימוש עם API ישיר.

- תומך במספר פלטפורמות, כולל התקני נייד.

- תיעוד טוב וקהילה מועילה.

- מציע תוכניות חינם ובתשלום.

חסרונות:

- יכול להיות יקר לפרוייקטים בקנה מידה גדול.

- שליטה מוגבלת על ארכיטקטורת השרת והאירוח.

- קצב ההתאמה וכמות השחקנים המקבילים מוגבלים.

קישור: [https://www.photonengine.com/#](https://www.photonengine.com/)

2. Mirror

יתרונות:

- קוד פתוח וחינמי לשימוש.

- ביצועים והתאמה טובים.

- מציע טווח של דגמי רשת להתאמה לסוגי משחקים שונים.

- תיעוד טוב וקהילה פעילה.

חסרונות:

- דורש הבנה טובה של מושגי רשת.

- לא כל כך ידידותי למתחילים כמספר מפתחים אחרים.

- עשוי לדרוש יותר הגדרה והגדרה מאשר פתרונות אחרים.

קישור: <https://mirror-networking.com/>

3. FishNet:

יתרונות:

- קל לשימוש עם API פשוט.

- ביצועים טובים וסלביליות.

- תומך במספר פלטפורמות, כולל התקני נייד.

- יש לו קהילה פעילה ותמיכה מפתחים רגישים.

חסרונות:

- אפשרויות מדיניות השרת ואירוח מוגבלות.

- עשוי לא להתאים לפרוייקטים בגודל גדול יותר.

- התיעוד עשוי להיות מוגבל במקומות.

ולבסוף:

קישור: <https://fish-networking.gitbook.io/docs/>

Unity Gaming Services הוא פלטפורמה משולבת ומתקדמת יותר לניהול משחקים מרובים משמעותיים בצד השרת, המספקת חוויית מפתח משוכללת ומובנית עם Unity Editor. הנה כמה מהיתרונות של Unity Gaming Services:

- ניהול משחקים מרובים מאובטח יותר בצד השרת.

- פלטפורמה משולבת עם מנהל הפרויקט של Unity וכן עם סטודיו הפיתוח של Unity.

- יכולת לנהל כמויות גדולות של משתמשים ונתונים בצורה יעילה יותר.

- מציע גם פתרונות ניהול משחק קצרי טווח, כמו תוצאות משחק וכלי ניתוח.

- מספק גם תכונות כמו זיהוי משתמשים, תקשורת בזמן אמת, ניהול חיווי משחק ועוד.

מכיוון ש-Unity Gaming Services משלב כלי פיתוח משחק וניהול משחקים בצד השרת תחת קונסולת אחת, יתרונו המשמעותי ביותר הוא הפשטות שהוא מביא לפיתוח ולניהול משחקים מרובים מסובכים. בנוסף, Unity Gaming Services מאפשר למפתחי משחקים להתמקד בפיתוח המשחק עצמו, במקום להתמקד בפיתוח והקמת שרתים ושילוב של מערכות ניהול משחק.

קישור: <https://unity.com/solutions/gaming-services>

החיסרון היחיד שהיה שיקול לוותר על UGS הוא שהפלטפורמה יחסית חדשה ואין המון מדריכים, אבל unity השקיעה המון כסף במדריכים איכותיים ביוטיוב בחינם אצל ערוצי יוטיוב ברמה גבוהה מאוד לדוגמה, קודמאנקי יוניטי הוא ערוץ יוטיוב שמתמקד בפיתוח משחקים באמצעות יצירת וידיאו-שיעורים לגבי המגוון של התכניות והכלים של יוניטי. הערוץ מציע תכנים מתקדמים עבור מפתחי משחקים, אך גם מספק מדריכים למתחילים. התכנים מתמקדים בנושאים כמו עיבוד תמונה, פיזיקה, AI, ניהול רשתות ועוד. הערוץ מתאים לכל מפתחי המשחקים ולכל מי שרוצה להתחיל בעולם פיתוח המשחקים באמצעות יוניטי. המדריכים של קודמואנקי הם ברמה גבוהה ואחרי שהוא הוציא קורס וכמה מדריכים של מעל 5 שעות על איך להשתמש בשירותים החלטתי שאני אלך על השירותים של unity.

קישור: <https://www.youtube.com/@CodeMonkeyUnity>

## שירותים שהשתמשתי בהם:

1. Unity Lobby - lobby מאפשר למשתמשים לפתוח שרת ולהכניס אותו לתוך pool, כל שרת lobby יכול להכיל מידע פרטי על משתמשים וגם מידע על השרת, השרת מתעדכן פעם בשנייה ולכן אי אשפר להשתמש בלובי כשרת של המשחק אלא לחיבור בין חברים ובשרת להעביר את המידע על השרת של המשחק. אפשר לשתף קוד של השרת lobby עם חבר וככה להתחבר לשרתי lobby פרטיים עם חברים.
2. Unity Netcode for GameObjects - הוא פלטפורמה שפותחה על ידי חברת Unity על מנת לאפשר למפתחי משחקים ליצור משחקי מרובי משתתפים בצורה יעילה וקלה יותר. הכלי הזה מאפשר לפיזור מחשבים מרוחקים באופן אוטומטי, בצורה שתפחית את העומס על המחשב המארח, ויאפשר למשתמשים לשחק בזמן אמת באופן מסונכרן. בנוסף, הפלטפורמה מאפשרת למפתחים לנהל את התקשורת בין המחשבים וליישם אסטרטגיות חכמות לניהול השכבת הרשת. באמצעות Unity Netcode for GameObjects משתמשים יכולים לפתח משחקים עשירי תכונות עם חוויות משתמש מקוונות חלקות ומהירות ללא פשרות בפונקציונליות או בביצועי המשחק.

## ניהול זמן:

|  |  |  |
| --- | --- | --- |
| משימה: | מה אלמד: | חומר: |
| חקירת פתרון networking | על הפתרונות הקיימים והפופלרים שיש לunity | [סרטון 1](https://www.youtube.com/watch?v=7w19z0c29Uo&t=133s&pp=ygUhYmVzdCBuZXR3b3JrIHNvbHV0aW9ucyAgZm9yIHVuaXR5)  [סרטון 2](https://www.youtube.com/watch?v=fb5PzAewDjU&pp=ygUhYmVzdCBuZXR3b3JrIHNvbHV0aW9ucyAgZm9yIHVuaXR5) |
| חקירת על הפתרון שמצאתי | מה אני יכול לעשות עם הפתרון שמצאתי | [סרטון 1](https://youtu.be/ecyK0vHmbpQ) |
| איך להשתמש בnetcode |  | [סרטון](https://youtu.be/3yuBOB3VrCk) |
| ליצור שרת בסיסי | להשתמש בידע שצברתי במשימה הקודמת |  |
| איך להשתמש בlobby |  | [סרטון](https://youtu.be/-KDlEBfCBiU) |
| ליצור שרת בסיסי |  |  |
| איך להשתמש בRELAY |  | [סרטון](https://youtu.be/msPNJ2cxWfw) |
| ליצור שרת בסיסי | להשתמש בידע שצברתי במשימה הקודמת |  |
| לחבר אותם ביחד | לממשק שלושה שירותים ביחד | ניסוי ותהייה עצמית |
| ליצור UI נחמד | עבודה עם canvas וUNITYUI | [סרטון](https://www.youtube.com/watch?v=IuuKUaZQiSU) + למידה עצמית |
| חקירת על יעול הקוד וזמן הריצה | Client prediction, lag. | [סרטון](https://youtu.be/TFLD9HWOc2k) |
| הוספה לקוד שלי את היעול |  |  |
| Polish |  |  |
| יצירת המשחק עצמו | הפיזיקה של הכדור והשערים |  |
| Debugging |  |  |

# אתגרים מרכזיים:

במהלך העבודה התמודדתי עם אתגרים רבים ומגוונים אשר הקשו עלי את המשך העבודה על הפרויקט אך נטעו ונתנו לי את הכלים והאמצעים בהם אוכל להשתמש בעתיד לפרויקטים אחרים.

האתגר הראשון: היה בהתקנת כל השירותים, ניסיתי להוריד פשוט אבל זה לא כל כך פשוט. עברתי אחרי התיעוד שהיה נוח וטוב, גילתי את ה Packet Manager הוא כלי ב-Unity שמאפשר למפתחים לנהל חבילות קוד שנמצאות בפרויקט. הוא מאפשר למפתחים להתקין, לעדכן ולנהל חבילות בצורה יעילה ומאורגנת. זהו כלי חשוב לפיתוח משחקים מרובי משתתפים ב-Unity, כי הוא מאפשר למפתחים להתמקד במפתח המשחק ולא בניהול החבילות שלו.

Packet Manager מאפשר למפתחים להתקין חבילות קוד מרחוק ממקורות שונים, לדוגמה GitHub או NuGet. הוא גם מאפשר למפתחים לעדכן חבילות קוד, להסיר חבילות קוד, ולנהל את התלות של החבילות הללו.

Packet Manager מאפשר למפתחים לפתח משחקים מרובי משתתפים עם קוד נקי ומאורגן. הוא מסייע למפתחים לנהל את הפרויקט שלהם ולהתמקד במפתח המשחק במקום בניהול החבילות שלו.

הוספתי את כל החבילות שהייתי צריך, אחרי זה צריך ללכת לdashboard ביונטי ולהפעיל אותם.

וככה התמודדתי אם האתגר הראשון, שאכן היה פשוט אבל הכין אותי להמשך.

## אתגר שני:

יצירת משחק מרובה משתתפים היינה תהליך מורכב הדורש פיתוח תשתיות ופתרון של בעיות מורכבות. במהלך הפיתוח, נדרשת החלטה מכרעת בין שתי שיטות אימות: server auth ו-client auth.

במאמץ למנוע הונאות ופעילות מסוכנת בתוך המשחק, נוצרה שיטת ה-server auth בה כל הפעולות נובעות מהשרת המרכזי ללא אימות נוסף. זהו גישה פחות מורכבת שכן לא נדרש כלי אבטחה מתקדם או אימות פרטי. לעומת זאת, client auth נדרש להפעיל מכשירים עם אישור מתאים ויצירת יחס אמין עם המשתמשים. זאת מתבססת גם על כלי אבטחה מתקדם נוסף כגון כרטיסי אבטחה.

עם זאת, הפיתוח של תשתיות כאלה נועד לא לכפות על המשתמש תהליך מיומן קושי ולמצוא את הפתרון הטוב יותר למפתחי המשחק\צוות הפיתוח. המודל השונה יתאים לשימוש נכון ולדרישות המשחק כך שהמימוש של האימות יהיה יעיל ויוביל למשחק חלק ובטוח לכל המשתמשים.

Server auth הוא בחירה אפקטיבית יותר בשביל הפרויקט שלי כיוון ששימוש בו יעשה רבים מן הנהלים המשחקים הרגילים פחות תלויים בתקשורת האינטרנט, ישנו מאגר שרת מרכזי שמנתח ומאמת את הפעולות ויודע לזהות בקלות מחלות ואי התאמות בפעולות המשתמשים. ולכן שכר העלות הוא מינימלי יותר מאשר ב-client auth.

כמו כן, ה-server auth מפחית את הסיכונים הבטחוניים, כך שבמקום זה שכל פעולה תצטרך לעבור אימות ואישור בצד הלקוח (במקרה של-client auth), הצד השרת עוסק בדבריים ומונע ניסיונות הונאה.

כחלק מבחירת האימות המתאים, חשוב לקחת בחשבון את הדרישות המסוימות של המשחק או המערכת, כמו גם לבדוק את הסיכונים בנוגע לבטחון ואפשרות הונאה, תוך דרישת הנדסת פתרון המאפשר משחק בטוח לכל המשתמשים.

Client prediction הוא עיבוד נתונים המתבצע על ידי מחשב הלקוח במשחק מקוון ומאפשר למשתמש לקבל תגובות מהירות ודווקאות יותר במשחק. כאשר משתמש משחק מקוון, המשחק חייב להיות מסונכרן עם השרת כדי להסיר התנגשויות בינו לבין השחקנים האחרים. אך מכיוון שתקשורת רשת הוא עיבוד בעצמו, הקשר ברשת יכול להיות איטי או חסר תגובה לפעמים, שזה עשוי לגרום למשחק לאט מה שיגרום לצורך במחשב המקומי לרשת עוצמה נימוסית וקבלת החלטות מהירות. בשביל לפתור את הבעיה הזו, חלק נתונים מסוים מאוחסנים במחשב הלקוח במשחק ואם הם נדרשים, הם ייקחו מהמקומיות כדי להציע חוויית משחק חלק ומזמינה יותר.

סיכום זה ש-client prediction משפר את חוויית המשחק במשחק מקוון על ידי מיזוג נתונים סיסמאות לוקוליות עם כמה מבחני אימות קטנים לפני שהם מתבצעים בפועל כדי ליצור תגובות מהירות ואופטימליות.

האתגר היה המימוש של הclient prediction, צריך היה לבסס את המשחק על טיקים שהן אותו דבר גם בשרת וגם בלקוח. מצאתי [סרטון טוב](https://youtu.be/TFLD9HWOc2k) שעוזר אבל בסופו של דבר עשיתי לבד הרוב.

## אתגר שלישי:

Unity Relay היא פתרון רשת ענן המיועד לטפל באתגרים במשחקי משתמש מקוון, כולל בעיות עם חומת אש, רשתות כתובות רשת (NAT) וחיבורים peer-to-peer. האתגרים הללו עשויים לכלול קשיים עם פתיחת יציאות, שבדרך כלל דרושה כדי להקים חיבורים בין הלקוחות של המשחק, או בעיות עם מעבר NAT, אשר עשויות להיווצר עקב מכשירי ראוטר ברשת המנסים ליצור שכבות נוספות של אבטחה למכשירים המחוברים.

בעת ניסיון להתחבר עם נגנים אחרים דרך האינטרנט, יש למהדרי משחקים לעבור דרך חומות אש ומכשירי NAT כדי להקים חיבורי רשת בין כל המכשירים השותפים. מכיוון שחומות האש מיועדות להגביל תעבורת נכנסת לרשת, ו-NAT עשויות ליצור בעיות עם חיבור peer-to-peer, יתכן שהתוצאה תהיה אין אפשרות להתחבר לנגנים אחרים, וזה עשוי להשפיע על המשחק ועל המעורבות של השחקן.

Unity Relay נועד לעזור להתגבר על האתגרים הללו על ידי מתן פתרון ענן תקמפתי היכול לעבור דרך כל חומת אש או מכשיר NAT. במקום להסתמך על חיבורי peer-to-peer, Unity Relay מקים עבור החלקה מבוססת ענן באמצעותה נעביר תעבורת מספרי המשחקים. גישה זו עוזרת לשפר את החיבור, להפחית את הירידה בביצועים ולהגביר את הביצועים הכלליים, ומבטיחה חוויית משחק חלקה גם בפני האתגרים המתייחסים לתקשורת רשת.

# תיאור תחום הידע:

Unity Netcode for GameObjects היא טכנולוגיה של Unity שמאפשרת למפתחי משחקים ליצור משחקים מרובי משתמשים באינטרנט. הטכנולוגיה משתמשת בקוד אחד גם לשרת וגם ללקוח, כדי להבטיח שהתנהגות המשחק תהיה זהה לכל המשתמשים.

בעזרת Unity Netcode for GameObjects, המפתחים יכולים ליצור אובייקטים במשחק ולהעביר אותם בין השרת והלקוח. המערכת מאפשרת למשתמשים להתחבר לשרת, לייצר אובייקטים ולבצע פעולות במשחק.

הטכנולוגיה מאפשרת למשתמשים להתחבר ולשחק במשחק באופן חלק ובלי עיכובים. הכלים המובנים של Unity Netcode for GameObjects מסייעים למפתחים ליצור משחקים מרתקים ומהירים באינטרנט, ולכן אני אפרט את היכולות ביחד.

## פירוט יכולות:

1: Unity AUTHENTICATION הוא מערכת אימות משתמשים במשחקים שנבנו עם הפלטפורמה Unity. המערכת מאמתת את זהות המשתמש ומאפשרת לו להתחבר למשחק.

אוסף היכולות/פעולות הנדרשות למימוש היכולת כולל:

- קבלת הזדהות ואימות זהות מהשרת

- יצירת אובייקט שחקן בעולם המשחק למשתמש מאומת

- בדיקת הרשאות והתאמה של המשתמש לפעולות מסוימות במשחק

- ניהול חיבורי המשתמשים לשרת

אובייקטים נחוצים כוללים: ממשק משתמש, שרת אימות, תקשורת.

2. Unity lobby היא יכולת בתוך Unity שמאפשרת לשחקני ליצור לובי מרכזי למשחק שלהם. הלובי מאפשר למשתמשים להתחבר, ליצור חדרים ולהתחיל משחקים. זה מאפשר למשתמשים למצוא ולהתחבר למשחקים בקלות ובמהירות.

אוסף היכולות והפעולות הנדרשות למימוש Unity lobby כוללים:

- יצירת לובי ראשי וחדרי משחק

- ניהול התחברויות למשתמשים וניהול חדרי משחק

- אפשרות להתחבר למשחקים קיימים וליצור משחקים חדשים

- שליטה על פעולות המשתמשים בלובי ובחדרי המשחקים

אובייקטים שנחוצים למימוש Unity lobby כוללים:

- לובי ראשי וחדרי משחק

- משתמשים והתחברויות

- חדרי משחק ומשחקים עצמם.

3. Unity Relay היא יכולת בתוך Unity שמאפשרת למשחקים להתחבר ולתקשר עם שרתים רחוקים בצורה מאובטחת ובעזרת טכנולוגיות רשת מתקדמות. זה מאפשר למשחקים להיות חלק מרשת משותפת ולהתחבר לשחקנים מרחבי העולם.

אוסף היכולות והפעולות הנדרשות למימוש Unity Relay כוללים:

- התחברות לשרת רחוק

- שליחת וקבלת נתונים בצורה מאובטחת

- ניהול חיבורים והתנתקויות

אובייקטים שנחוצים למימוש Unity Relay כוללים:

- שרתים רחוקים

- מחשבי לקוח/משחקים

- חיבורי רשת מאובטחים.

4. Unity Netcode for GameObjects הוא יכולת בתוך Unity שמאפשרת למשחקים ליצור חיבורים רשת מהירים ומאובטחים בין אובייקטים במשחק. זה מאפשר למשחקים להיות מאונפים ולתקשר עם שחקנים אחרים בזמן אמת.

אוסף היכולות והפעולות הנדרשות למימוש Unity Netcode for GameObjects כוללים:

- יצירת חיבורי רשת מהירים ומאובטחים

- שליחת וקבלת נתונים בצורה מהירה ומאובטחת

- ניהול חיבורי רשת והתנתקויות

אובייקטים שנחוצים למימוש Unity Netcode for GameObjects כוללים:

- אובייקטי משחק שמשתתפים בחיבורי הרשת

- חיבורי רשת מהירים ומאובטחים.

# ארכיטקטורה:

הארכיטקטורה של משחק מרובה משתתפים ב-Unity בסיסי כוללת מספר רכיבים מרכזיים:

1. לקוח: הלקוח הוא היישום שרץ על מכשיר השחקן ומתאים למשתמש קלט, עיבוד גרפיקה ושליחה / קבלת נתונים מהשרת.

2. שרת: השרת אחראי לניהול מצב המשחק, לטיפול בחיבורי השחקנים ולשדר עדכונים לכל הלקוחות המחוברים.

3. ספריית רשת: Unity מספקת מספר ספריות רשת שמאפשרות למפתחים ליצור משחקי מרובה משתתפים. ספריות אלו מתעסקות בתקשורת רשת רמה נמוכה בין לקוחות ושרתים.

4. הלוגיקה של המשחק: הלוגיקה של המשחק אחראית לניהול כללי המשחק, לטיפול בפעולות השחקנים ולעדכון מצב המשחק.

הזרימה הבסיסית של משחק מרובה משתתפים ב-Unity בסיסי כוללת את השלבים הבאים:

1. השחקן משיק את לקוח המשחק ומתחבר לשרת.

2. השרת מאמת את פרטי הזיהוי של השחקן ויוצר אובייקט שחקן חדש בעולם המשחק.

3. השחקן משתמש בעולם המשחק על ידי שליחת פקודות קלט לשרת.

4. השרת מקבל את הפקודות הנכנסות מכל השחקנים המחוברים ומעדכן את מצב המשחק להתאם.

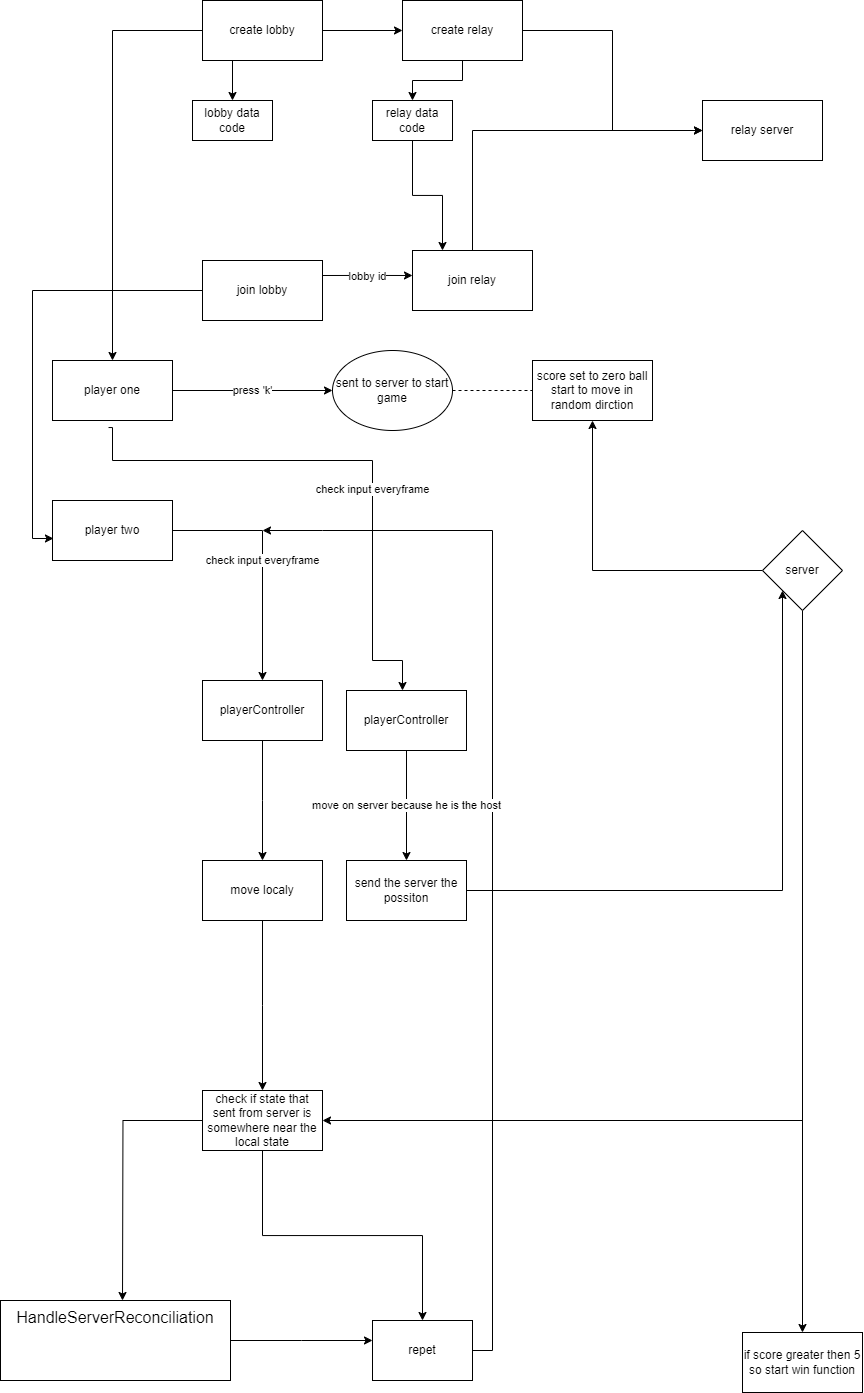
5. השרת משדר את מצב המשחק המעודכן לכל הלקוחות המחוברים.

6. הלקוח מקבל את מצב המשחק המעודכן ומעדכן אותו על מסך השחקן.

7. המחזור חוזר עד שהמשחק מסתיים או השחקן מנתק.

בסך הכל, הארכיטקטורה של משחק מרובה משתתפים ב-Unity בסיסי כוללת מודל לקוח-שרת עם ספריית רשת, הלוגיקה של המשחק ומסד נתונים. הארכיטקטורה זו מאפשרת למספר שחקנים להתקשר זה עם זה בזמן אמת וליהנות מחוויית משחק משותפת.

## דיאגרמה:



## סביבת פיתוח:

Visual Studio הוא סביבת פיתוח משולבת (IDE) שמיועדת לפיתוח תוכנה בשפות שונות כמו C#, C++, ו-Visual Basic. סביבת הפיתוח חזקה ומאובטחת ומספקת כלים מתקדמים לכתיבת קוד, ניהול פרויקטים, איתור באגים ודיבאג, ועוד.

Unity היא פלטפורמה לפיתוח משחקים ויישומים תלת-ממדיים. היא מאפשרת למפתחים ליצור משחקים בצורה מהירה ויעילה באמצעות כלים ומערכת פיתוח מתקדמת.

השילוב בין Visual Studio ו-Unity מאפשר למפתחים לפתח משחקים באמצעות Unity בצורה יעילה ומתקדמת. הניכון הראשון של Unity כולל תמיכה מובנית ל-Visual Studio והתממשות של Unity עשויה במידה רבה בעזרת C# כשפת התכנות הראשית. עם זאת, ניתן לשלב גם שפות תכנות אחרות כמו JavaScript ו-Boo ביחד עם Unity, אך רוב הפיתוח בוצע באמצעות C#.

הניכון של Visual Studio עם Unity מספק מערכת פיתוח מתקדמת עם כלים חזקים לעריכת קוד, איתור באגים, ניהול פרויקטים ודיבאג. תמיכה ב-C# מאפשרת קישור מהיר ויעיל בין קוד המקור של המשחק ב-Unity וקוד התכנית ב-Visual Studio.

Visual Studio מאפשרת יכולות נוספ

ות כמו צבירת קוד, אזורי עבודה מותאמים אישית, איתור באגים באמצעות מנוע דיבאג מתקדם, תמיכה בפרויקטים מרובים וצוותים, ועוד. התממשות ה-IntelliSense של Visual Studio גם מספקת יכולת משופרת של השלמת קוד והצגת הוראות אוטומטיות, מה שיכול להוביל לפיתוח יעיל ומהיר יותר.

בסה"כ, השימוש של Visual Studio עם Unity מאפשר למפתחים לפתח משחקים באמצעות Unity בצורה נוחה ויעילה יותר, תוך כדי הנצלת כלים מתקדמים לפיתוח תוכנה ותכונות של ה-IDE המובנות ב-Visual Studio.

## תיאור איומים:

1. איומים באבטחת המידע: כאשר משחקים משתמשים ברשת כדי לתקשר עם שרתים או עם יישומים אחרים, אבטחת המידע היא גורם מרכזי. זה כולל מניעת גישה לא מורשית לנתונים הרגישים של המשתמשים, כמו פרטי חשבונות או מידע אישי. Unity Netcode, Unity Relay, ו-Unity Lobby מספקים מנגנונים אבטחתיים כגון הצפנה (encryption) ואימות (authentication) כדי להבטיח שהנתונים שמתעברים ברשת נשמרים בסודיות ואינם נגישים לצדדים לא מורשים.

2. איומים בגידור וניהול חיבור: בגלל טבע הרשת והשימוש בגידור (networking) במשחקים, ייתכן ויתרחשו הפרעות בחיבור ובתקשורת. זה עשוי לגרום לבעיות כמו היעלמות חברים מהמשחק, הפסקות בצירוף ובהתאמה של מידע בין השחקנים, או התנתקות מתישהו בעת המשחק. הכלים של Unity מספקים טכנולוגיות ותוספים כמו Unity Relay כדי להתמודד עם הפרעות כאלה ולספק חוויית משחק חלקה ומתמשכת.

3. איומים בניהול הקהילה וחדרי המתנה (lobbies): Unity Lobby מספק פתרונות לניהול חדרי המתנה במשחקים מרובי משתתפים (multiplayer) ולסינכרון בין השחקנים. אתגרים נפוצים כוללים התמודדות עם רווחי חדרי המתנה וניהול משחקים במקביל בצורה יעילה. על מנת להתמודד עם זה, Unity Lobby משתמש בטכנולוגיות כמו ניתוב משתמשים (user routing) וניתוב רשת (network routing) כדי למקד את התקשורת ולהבטיח שחדרי המתנה מנוהלים בצורה יעילה וחלקה.

4. אבטחת המידע והתמודדות עם התקפות נפוצות כמו DDoS מהווה אתגר בכל מערכות הרשת ואינה מוגבלת ל Unity בלבד. כדי להתמודד עם התקפות DDoS, נדרשים פתרונות נפרדים שפועלים ברמת האינפרה והתשתיות הרשת של המשחקים. תשתית חזקה, שירותי צינור (pipe) רחבים, תצורות עומס ומידע מוטעה מגוון הם דוגמאות לפתרונות שמטרתם להגן על משאבי השרת ולמנוע או להקטין את השפעת התקפות DDoS.

לגבי ההצפנה, Unity Netcode, Unity Relay ו-Unity Lobby מאפשרים הצפנה וקישור בין השרתים והלקוחות (clients) באמצעות פרוטוקולים רשת מבוססי תקן, כמו SSL/TLS (Secure Sockets Layer/Transport Layer Security). השימוש בהצפנה מבטיח שהתקשורת בין המשתמשים והשרתים מוגנת מפני חדירה למידע והערעורים לא מורשים..

# מדריך למשתמש + הסברת מסכים:

## מסך הפתיחה:



מסך הפתיחה מכיל 4 כפתורים, ולוג.

הלוג הוא בשביל המפתח בלבד. אפשר לשים לב שכתוב שם את הid של השחקן.

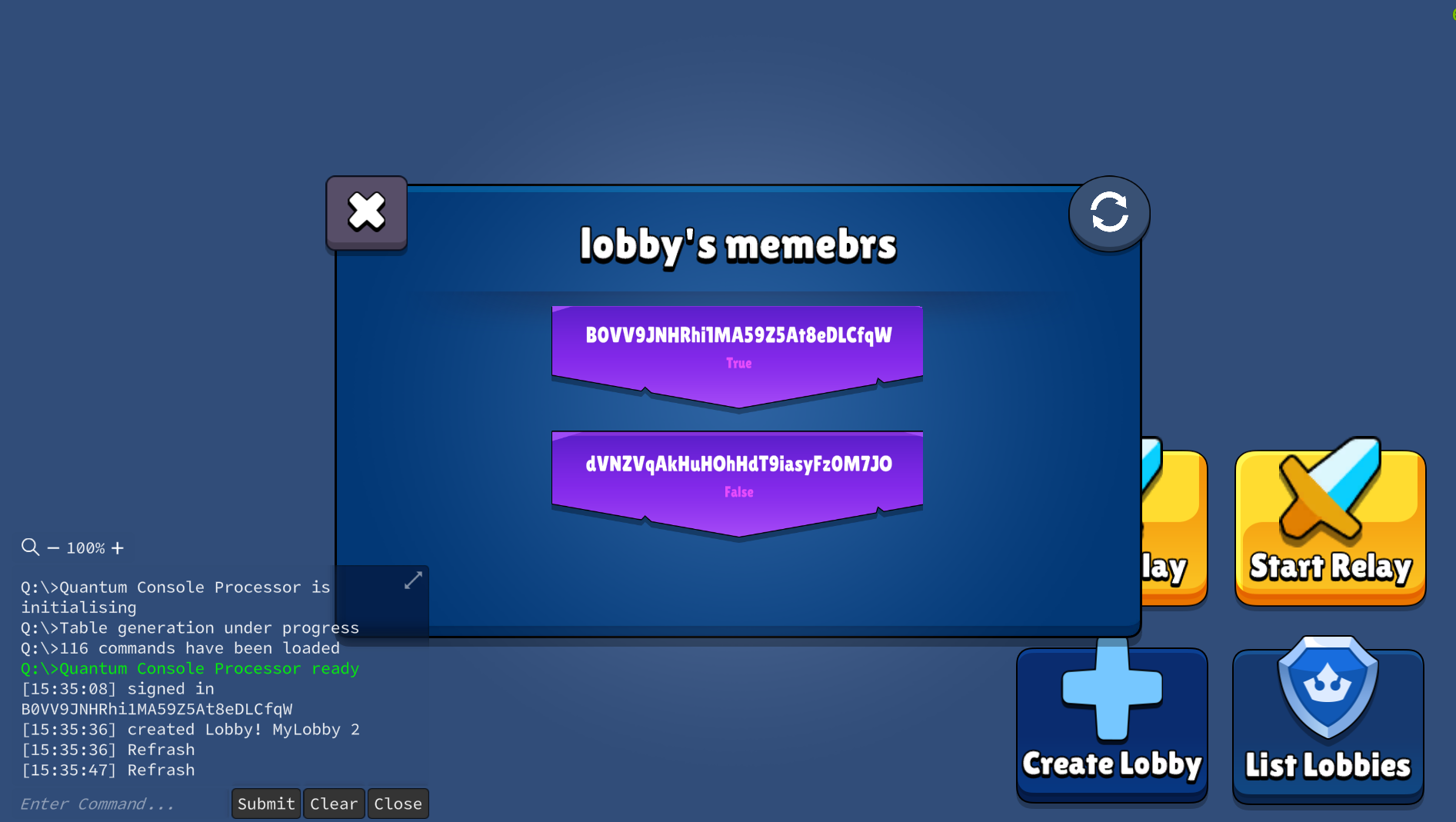
כפתור Create Lobby: הוא יוצר לובי פתוח אשר מאפשר כניסה לכולם.

כפתור List Lobbies: הוא מראה את השרתים הפתוחים שאפשר להיכנס אליהם.

כפתור Start relay: הוא מאפשר רק למנהל הלובי לפתוח את שרת הrelay.

כפתור Join relay: הוא מאפשר רק למנהל הלובי לפתוח את שרת הrelay.

## מסך מנהל הלובי:



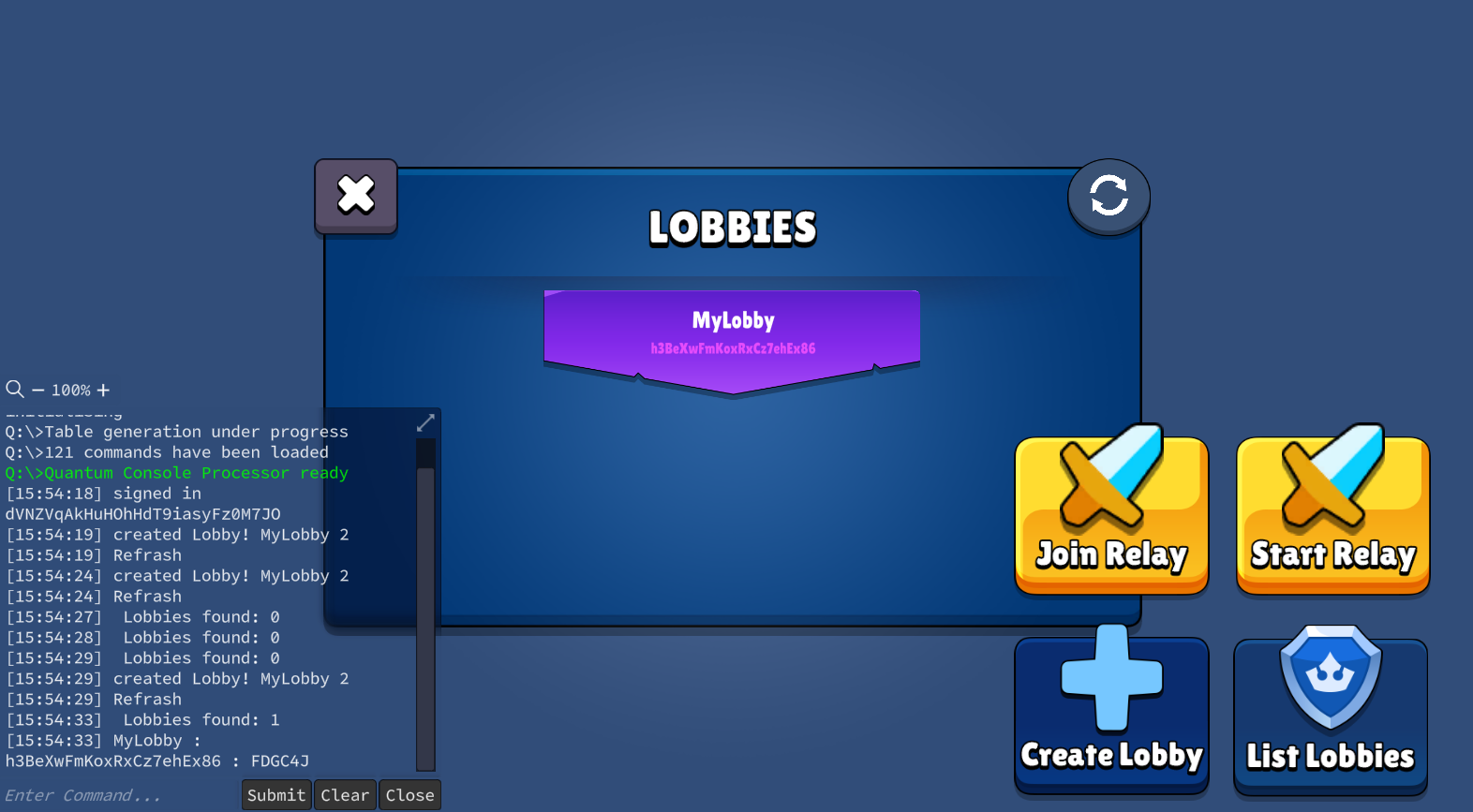
אחרי שלחצנו על כפתור הcreate lobby, קופץ רשימה של השחקנים שמחוברים ללובי, מוצגים כid שלהם כמו שאפשר לראות בלוג, מתחת לזה יש משתנה bool שהוא אמת אם המשתמש הוא האדמין של הלובי. אם אתה האדמין יש אפשרות ללחוץ על השחקנים כדי להעיף אותם מהלובי.

אחרי זה יש אפשרות להחיל את שרת הrelay על ידי לחיצה על הכפתור.

## תמונה שמכילה צילום מסך, עיגול, גרפיקה, גופן התיאור נוצר באופן אוטומטימסך המשחק:

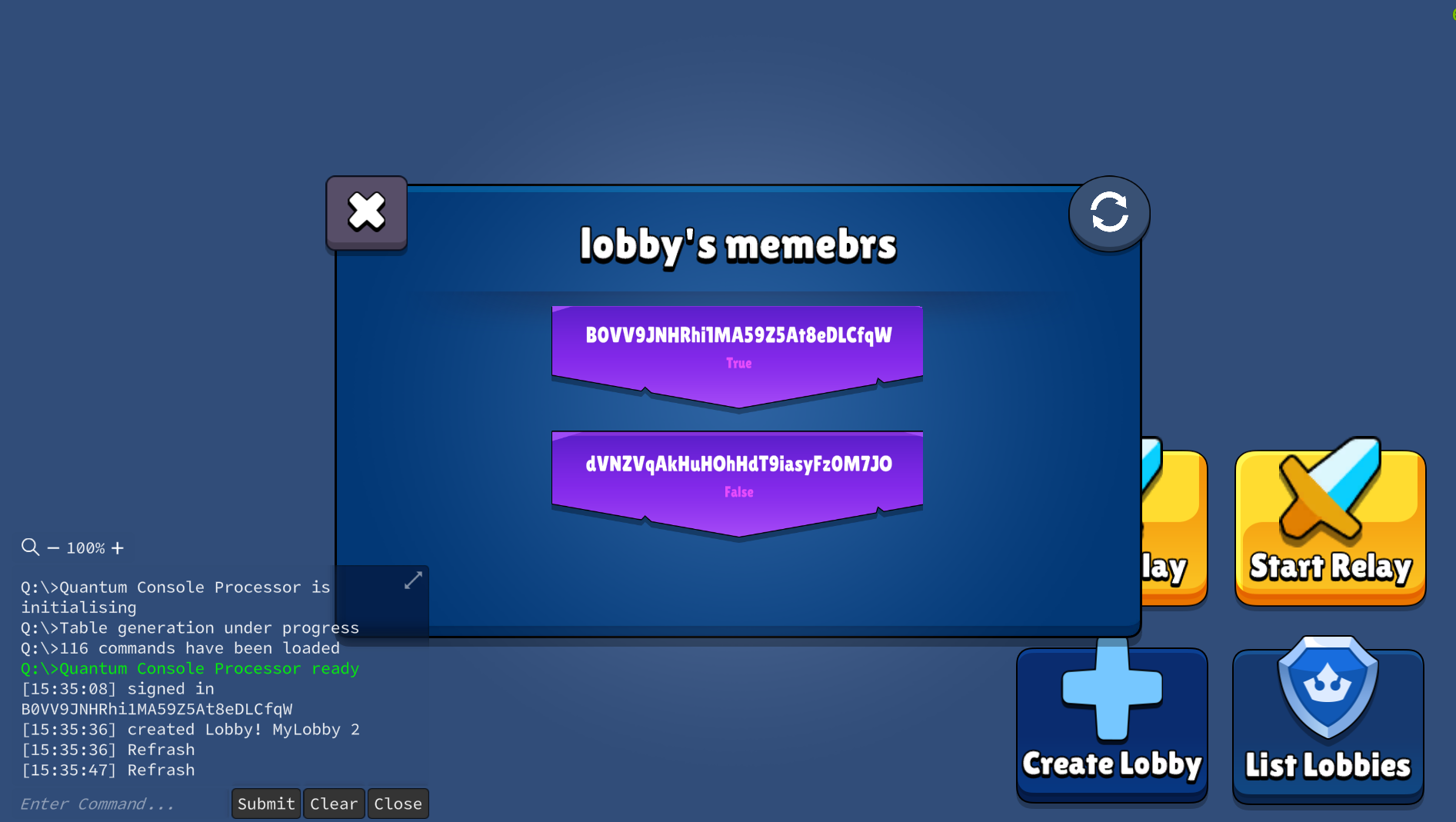
משחק בוא שני השחקנים מחוברים ועוד לא התחיל. פה שחקן מספר 1 צריך ללחוץ על הכפתור k במקלדת כדי להתחיל את המשחק

## מסך בחירת הלובים:.



המסך מציג את השרתי הלובי שאפשר להיכנס אליהם, יחד עם שם הלובי ומתחתיו הid של הלובי שמאפשר כניסה אל תוך הלובי.

## מסך הלקוח בתוך השרת:



המסך כמו המסך של האדמין של השרת רק שפה צריך ללחוץ על כתפור הjoin relay.

# מימוש:

## מחלקה neural network:

הקוד מגדיר מחלקה בשם "Network" שמייצגת מודל רשת נוירונים לסוכן למידת משוגע (Reinforcement Learning). הנה פרשנות למה שהקוד עושה:

1. הקוד מתחיל בייבוא המודולים הנדרשים: os.path לפעולות ניהול נתיבי קבצים, numpy לפעולות מספריות, ו- tensorflow לבניית ואימון רשתות נוירונים.

2. OBSERVATIONS\_SIZE הוא קבוע שמגדיר את גודל התצפיות הקלט.

3. מוגדרת מחלקת "Network". היא כוללת את השיטות הבאות:

- "\_\_init\_\_()": זו שיטת הבנאי שמאתחלת את הרשת. היא מקבלת שלושה פרמטרים: hidden\_layer\_size (מספר היחידות בשכבה החבויה), learning\_rate (שיעור הלמידה של המותאם), ו- checkpoints\_dir (תיקייה שבה יישמרו הבדיקות).

- בתוך הבנאי, מייצרים הפעלה אינטראקטיבית של TensorFlow (self.sess).

- מוגדרים placeholders עבור תצפיות הקלט (self.observations), פעולות שנדגשו (self.sampled\_actions), וערכי יתרון (self.advantage).

- מוגדרות שכבות הרשת הנוירונים באמצעות הפונקציה tf.layers.dense(). תצפיות הקלט עוברות דרך שכבה חבויה עם פעולת הפעלה ReLU, ופלט השכבה החבויה עוברת דרך שכבה נוספת עם פונקציית הפעלה sigmoid. פלט השכבה הזו מייצג את ההסתברות לביצ

וע פעולת "למעלה" (up).

- הפון מאבד הוא פונקציית ההפסד שמוגדרת כהפסד הלוגריתם בין הפעולות שנדגשו להסתברויות המנובאות. ההפסד משוקלל בערכי היתרון, שמייצגים את הפרסים עבור כל פעולה בכל סבב.

- מתקבל מזין Adam עם שיעור הלמידה המתאים, והשיטה minimize() נקראת עליו עם ההפסד כמטרה למינימיזציה. זה מגדיר את התהליך האימון (self.train\_op).

- מתבצעת איתחול של משתני העולם הגלובליים, ונוצרת אובייקט "Saver" כדי לשמור ולשחזר נקודות בדיקה של המודל.

- "load\_checkpoint()": שיטה זו מטעה את נקודת הבדיקה השמורה מהנתיב המצוין באמצעות אובייקט ה-"Saver".

- "save\_checkpoint()": שיטה זו שומרת את נקודת הבדיקה הנוכחית לנתיב המצויין באמצעות אובייקט ה-"Saver".

- "forward\_pass()": שיטה זו מבצעת פינוי קדימה דרך הרשת לפי סט תצפיות נתונות. היא מחזירה את ההסתברות המיוחדת לביצוע הפעולה "למעלה".

- "train()": שיטה זו מבצעת אימון של הרשת בעזרת קבוצת הזוגות "(מצב, פעולה, פרסה)" הספציפית. היא מקבלת רשימה של זוגות כקלט ו

מפרקת אותם לרשימות נפרדות של מצבים, פעולות ופרסות. הרשימות מומרות למערכים של NumPy ומוזנות לרשת בעזרת מילוי של Feed Dictionary. המשימה train\_op מופעלת כדי לבצע שלב אימון אחד.

בסך הכל, הקוד מבצע הגדרת מודל רשת נוירונים עם ארכיטקטורה ספציפית ומגדיר שיטות לטעינה, שמירה, פינוי קדימה ואימון של המודל.

## מחלקה onlineBall:

### תיאור המחלקה:

המחלקה אחראית על הקוד של הכדור והסנכרון שלו עם כל הלקוחות. המחלקה יורשת ממחלקת NetworkBehaviour שאחראית על מתן שלל פקודות עזר לאובייקטים במרחב.

### תכונות:

private NetworkVariable<Vector2> serverPostion

* משתנה מסוג networkVariable זה משתנה המאפשר סנכרון של המשתנה מול כל הלקוחות בקלות, המשתנה מסוג vector2 הכוונה מכיל x ו Y אשר מבטאים את המיקום של הכדור השרת

private const float maxHight

-משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המקסימום Y של הכדור לפני שמתנגש בתקרה ונאלץ לשנות כיוון.

private const float minHight

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המינימום Y של הכדור לפני שמתנגש בתקרה ונאלץ לשנות כיוון.

private const float maxWidth

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המקסימום X של הכדור לפני שמתנגש בשחקן ונאלץ לשנות כיוון.

private const float minWidth

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המינימום X של הכדור לפני שמתנגש בשחקן ונאלץ לשנות כיוון.

private const float playerMax

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המרחק של קצה המקסימלי של השחקן מן המיקום שלו, מאפשר לדעת אם הכדור מתנגש בשחקן רק לפי מיקום השחקן

private const float playerMin

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את המרחק של קצה המינמלי של השחקן מן המיקום שלו, מאפשר לדעת אם הכדור מתנגש בשחקן רק לפי מיקום השחקן

private const float playerWidth

- משתנה const float פרטי מכיוון שלא צריך להשתנות, מבטא את העובי של השחקן, מאפשר לדעת אם הכדור מתנגש בשחקן רק לפי מיקום השחקן

private Rigidbody2D rb

-משתנה סטטי מסוג rigidbody2d שזה הcomponnent של unity שמבטא את הכוחות הפועלים על אוביקט, סטטי מכיוון שצריך להשתמש בו מחוץ למחלקה ולא צריך לעשות רפרנס למחלקה כשאני משתמש בrigidbody.

[SerializeField] static private float speed

-משתננה סטטי פרטי מסוג float אחראי על המהירות של הכדור, יש סירלזציה באינספקטור כדי לעשות חווית game design נוחה יותר. נותן לשינוי דרך האינספטור ביונטי בזמן הפיתוח.

### פעולות:

private void Start()

* פונקציה הנקראת לפני הפריים הראשון, מאתחלת את המשתנה rigidbody2d מאתחל בשרת את המיקום של הכדור.

public override void OnNetworkSpawn()

* פונקציה הנקראת כשאר האוביקט נוצר ברשת. בשרת מאתחלת את המיקום של הכדור לשאר הלקוחות, בלקוח מקבל את המיקום של הכדור. הפונקציה היא override כי היא לקוחה מה NetworkBehaviourואני משרבט אותה.

public static void startPoint()

* פונקציה הנקראת כאשר השחקן שהוא הhost לוחץ על המקש k והוא מאתחל את המיקום של המיקום ושולח את הכדור בכיוון רנדומלי.

private void Update()

* פונקציה הנקראת כל פריים, בשרת היא שולחת את המיקום של הכדור ללקוחות אצל הלקוח הוא משנה את המיקום הלוקאלי של הכדור, בשרת הוא מגביל את המהירות של הכדור, ושם נמצא גם החישובים של מתי פוגש את התקרה\קיר או השחקן.

void addScorePlayerOne()

* פונקציה הנקראת כאשר שחקן מספר 1 מבקיע גול.

void addScorePlayerTwo()

* פונקציה הנקראת כאשר שחקן מספר 2 מבקיע גול.

[ClientRpc]

void addScorePlayerOneClientRpc()

* פונקציה הנקראת כאשר שחקן מספר 1 מבקיע גול. הפונקציה היא clientRPC כלומר קוראים לה מן השרת והוא שולח את הפונקציה לכל אחד מהלקוחות ומבצע את זה אצלם.

[ClientRpc]

void addScorePlayerTwoClientRpc()

* פונקציה הנקראת כאשר שחקן מספר 2 מבקיע גול. הפונקציה היא clientRPC כלומר קוראים לה מן השרת והוא שולח את הפונקציה לכל אחד מהלקוחות ומבצע את זה אצלם.

## המחלקה TestLobby:

### תיאור המחלקה:

המחלקה אחראית על הקישור לunity lobby ועל שמירת תכונות סטטיות.

### תכונות:

public static TextMeshProUGUI playerOneScore;

* מתשנה מסוג TMPGUI המכיל את הטקסט של הניקוד של שחקן מספר אחד, המשתנה סטטי כדי שיהיה אפשר לגשת ממחלקות אחרות.

public static TextMeshProUGUI playerTwoScore;

* משתנה מסוג TMPGUI המכיל את הטקסט של הניקוד של שחקן מספר שתיים, המשתנה סטטי כדי שיהיה אפשר לגשת ממחלקות אחרות.

[SerializeField] private GameObject scoreHUD;

* משתנה מסוג אוביקט המכיל את הטקסט של הניקוד של שני השחקנים השימוש היחידי שלו הוא היכולת להסתיר את הטקסט כאשר אין צורך לראותו, לדוגמה כאשר השחקן בבחירת ההשרתים. עם סרליזציה באינספקטור.

[SerializeField] private GameObject LobbyTemplate;

* משתנה מסוג אובייקט אשר מכיל את הטימפלייט לרשימת שרתי לובי, ומשם אפשר ליצור כפתורים שמצרפים לשרתי לובי. עם סרליזציה באינספקטור.

[SerializeField] private GameObject LobbiesMenu;

* משתנה מסוג אוביקט המכיל את הרשימת שרתים השימוש היחידי שלו הוא היכולת להסתיר את הטקסט כאשר אין צורך לראותו, לדוגמה כאשר השחקן בזמן המשחק. עם סרליזציה באינספקטור.

[SerializeField] private GameObject MembersMenu;

* משתנה מסוג אוביקט המכיל את הרשימת השחקנים בשרת לובי השימוש היחידי שלו הוא היכולת להסתיר את האובייקט כאשר אין צורך לראותו, לדוגמה כאשר השחקן בזמן המשחק. עם סרליזציה באינספקטור.

[SerializeField] private Transform LobbiesContent;

* משתנה מסוג טרנפרום המכיל את המיקום שבוא אמור להופיע רשמית השרתים. עם סרליזציה באינספקטור.

[SerializeField] private Transform MembersContent;

* משתנה מסוג טרנפרום המכיל את המיקום שבוא אמור להופיע רשמית השחקנים. עם סרליזציה באינספקטור.

[SerializeField] private GameObject HUD;

* משתנה מסוג אוביקט המכיל את הui השימוש היחידי שלו הוא היכולת להסתיר את האוביקטים כאשר אין צורך לראותו, לדוגמה כאשר השחקן בזמן המשחק. עם סרליזציה באינספקטור.

static public GameObject playerOne;

* משתנה מסוג אוביקט המכיל את השחקן מספר 1, המשתנה סטטי כדי שיהיה גישה מסקריפטים אחרים.

static public GameObject playerTwo;

* משתנה מסוג אוביקט המכיל את השחקן מספר 2, המשתנה סטטי כדי שיהיה גישה מסקריפטים אחרים.

static public GameObject ball;

* משתנה מסוג אוביקט המכיל את הכדור, המשתנה סטטי כדי שיהיה גישה מסקריפטים אחרים.

const string RELAY\_JOIN\_CODE = "RELAY\_JOIN\_CODE";

* כונסט מסוג סטרינג המכיל את המיקום שבוא נשמר הקוד relay המאפשר להתחבר לשרת relay.

private static Lobby hostLobby;

* משתנה מסוג lobby מהכיל את השרת המרכזי, אפשר לקחת משם מידע כגון מי המשתתפים ומה הקוד של הlobby . המשתנה הוא סטטי כדי שיהיה אפשר לגשת מסקריפטים אחרים בנוח.

private float timer;

* מתשנה מסוג float, מטרתו זה לשמש כטיימר כדי לשלוח כל 15 שניות heartbeat ללובי כדי שלא יכבה.

### פעולות:

private void Update()

* פעולה הנקראת כל פריים, מטרתה היא לקרוא לפונקציה handleLobbyTimer.

private async void HandLobbyTimer()

* פעולה אסינכרונית אשר שולחת כל 15 שניות heartbeat לשרת lobby על מנת שלא יכבה כי יש לו טיימ אוט כל 20 שניות.

private async void Start()

* פעולה אסינכרונית הנקראת לפני הפריים הראשון, בפעולה הלקוח מתחבר לשרותים של יונטי, והפעולה גם מוסיפה לאיבנט signedIn שידפיס את הplayerID . וגם מכבה את ה hud

public async void CreateLobby()

* פעולה אסינכרונית אשר יוצרת לובי לשני משתתפים ושומרת את הקוד בתוך הdata של הlobby. ומציגה את השחקנים המחוברים

public async void ListLobbies()

* פעולה אסינכרונית אשר מחפשת בכל השרתים הפתוחים ומציגה אותם, לחיצה על אותם שרתים מאפשרת כניסה אליהם.

public async void JoinLobby(string id)

* פעולה אסינכרונית אשר מקבל id של לשרת לובי ופעולה מכניסה את הלקוח לשרת.

public async void ListMembers()

* פעולה אסינכרונית אשר מציגה את כל השחקנים המחוברים לשרת בזמן אמת.

private async void KickPlayer(string Id)

* פעולה אסינכרונית אשר עובדת רק למנהל השרת לובי ומאפשרת להעיף שחקנים מהלובי על ידי הID שלהם.

private void DestroyChilds(Transform parent)

* פעולה אשר מוחקת את כל הילדים של אובייקט.

public async void StartRelay()

* פעולה אסינכרונית אשר עובדת רק למנהל שרת הlobby ומתחילה שרת relay. מוסיפה את הקוד שלו לדאטה של הלובי.

public void JoinRelay()

* פעולה אסינכרונית אשר עובדת רק ללקוחות השרת המאפשרת כניסה של הלקוחות לשרת הrelay.

## המחלקה TestRelay:

### תיאור המחלקה:

המחלקה אחראית על הקשר של הלקוח לשרת הrelay.

### תכונות:

אין תכונות

פעולות:

public async static Task<string> CreateRelay(int MaxPlayers)

* הפעולה הינה פעולה אסינכרונית סטטית מסוג Task אשר פותחת שרת relay ומחזירה את הקוד שלו במידה והצליח לפתוח קוד.

public async static void JoinRelay(string joinCode)

* הפעולה הינה פעולה אסינכרונית אשר מצטרפת לשרת relay על ידי קוד.

## המחלקה setScorePlayerOne:

### תיאור המחלקה:

המחלקה מאתחלת את המשתנה את הניקוד של שחקן מספר 1

### תכונות:

[SerializeField] private TextMeshProUGUI playerOneScore;

* משתנה פרטי שמתסכנרן באינספקטור ששומר את הרפרנס לטקסט של הניקוד של שחקן מספר 1

### פעולות:

void Start()

* פעולה שנקראת לפני הפריים הראשון שמאתחלת את המשתנה של הניקוד של שחקן מספר אחד.

## המחלקה setScorePlayerTwo:

### תיאור המחלקה:

המחלקה מאתחלת את המשתנה את הניקוד של שחקן מספר 2

### תכונות:

[SerializeField] private TextMeshProUGUI playeTwoScore;

* משתנה פרטי שמתסכנרן באינספקטור ששומר את הרפרנס לטקסט של הניקוד של שחקן מספר 2

### פעולות:

void Start()

* פעולה שנקראת לפני הפריים הראשון שמאתחלת את המשתנה של הניקוד של שחקן מספר שתיים

## המחלקה PlayerController:

### תיאור המחלקה:

המחלקה יורשת מnetworkBehaviour וגם מהintrface IPlayerController באיינטרפייס בסך כל יש frameInput. המחלקה אחראית על השחקן על התזוזה שלו, על סינכרון שלו לשרת ולשאר הלקוחות.

### תכונות:

public FrameInput Input { get; private set; }

* משתנה מסוג FrameInput מטרתו זה לשמור את האינפוט לפריים הנוכחי מכיוון שאפשר להשתמש בקילטת input רק בפעולת הupdate

[SerializeField] private Rigidbody2D rb;

* משתנה פרטי משסונכרן באיספקטור מסוג rigidbody2d סוג זה מכיל את סכומי הכוחות הפעולים על השחקן.

private NetworkVariable<Vector2> serverPostion = new NetworkVariable<Vector2>();

* משתנה מסוג networkVariable זה משתנה המאפשר סנכרון של המשתנה מול כל הלקוחות בקלות, המשתנה מסוג vector2 הכוונה מכיל x ו Y אשר מבטאים את המיקום של השחקן בשרת.

private NetworkVariable<int> currentTickServer = new NetworkVariable<int>();

* משתנה מסוג networkVariable זה משתנה המאפשר סנכרון של המשתנה מול כל הלקוחות בקלות, המשתנה מסוג int ומכיל את מספר הtik הנוכחי.

private const float minHight =-4.3f;

* Const מסוג float המכיל את המיקום מינימום של השחקן משתמש להגבלת תזוזת השחקן

private const float maxHight = 4.3f;

* Const מסוג float המכיל את המיקום המקסימום של השחקן, משתמש להגבלת תזוזת השחקן

// Shared

private float timer;

* משתנה מסוג float הוא אחראי על הטיקים, הוא עוזר יחד עם הפעולה fixedUpdate להגיע למספר אחיד של טיקים בשנייה ולבצע פעולה בכל טיק.

private int currentTick;

* משתנה מסוג int אחראי על לשמור את מספר הטיק הנוכחי

private float minTimeBetweenTicks;

* משתנה מסוג float שווה לזמן בין הטיקים.

private const float SERVER\_TICK\_RATE = 30f;

* Const מסוג float המבטא את מספר הטיקים בשרת

private const int BUFFER\_SIZE = 1024;

* const מסוג int הוא אחראי על גודל הבאפרים.

private float Speed = 15;

* משתנה מסוג float הוא מבטא את מהירות השחקן.

private int state =0;

* משתנה מסוג int המבטא את מצב התזוזה של השחקן, 0 זה אומר שהוא לא זז, 1 זה למעלה מינוס 1 זה למטה.

// Client specific

private StatePayload[] stateBuffer;

* מערך של משתנים מסוג הstruct state payload הstruct מכיל tick ומיקום של השחקן. מאותחל בגודל של 1024.

private StatePayload latestServerState;

* משתנה מסוג הstruct state payload הstruct מכיל tick ומיקום של השחקן. שומר את המיקום האחרון.

// server specific

private StatePayload[] stateBufferServer;

משתנה מסוג הstruct state payload הstruct מכיל tick ומיקום של השחקן. שומר את המיקום האחרון של השרת, ולכן מתי שצריך להשוואות בינהם אז שמים את מספר הטיק ואז אפשר לראות איפה הייתה בעיה בתקשורת

### פעולות:

public override void OnNetworkSpawn()

המתודה הזו מחליפה את המימוש המקורי של מתודת OnNetworkSpawn() במחלקה הבסיסית, ומציינת שהיא תופעל כאשר אובייקט הרשת מתווסף.

הנה פירוט מה עושה המתודה:

1. אם הלקוח המקומי הוא השרת:

א. מיקום השרת יוגדר למיקום הנוכחי של האובייקט.

ב. ערך ה-tick הנוכחי של השרת יוגדר ל-0.

ג. אם הלקוח המקומי הוא בעל האובייקט:

- המשתנה "TestLobby.playerOne" יוקצה עם הפניה לאובייקט המשחק הנוכחי.

- המיקום של האובייקט יוגדר ל־(-8, 0) על מספרי המימדים.

ד. אם הלקוח המקומי אינו בעל האובייקט:

- המיקום של האובייקט יוגדר ל־(8, 0) על מספרי המימדים.

2. אם הלקוח המקומי הוא בעל האובייקט אך לא השרת, מתבצעת קריאה למתודה setAsPlayerTwoServerRpc().

בסיכום, המתודה הזו מאתחלת את מיקום ובעלות האובייקט ברשת. אם הלקוח המקומי הוא השרת, היא מגדירה את מיקום השרת ומשמיעה את הבעלות ללקוח. אם הלקוח המקומי הוא בעל האובייקט אך לא השרת, היא מבצעת לוגיקה נוספת הקשורה להגדרת השחקן השני.

[ServerRpc] void setAsPlayerTwoServerRpc()

* הערה: יש לשים לב שהפונקציה שניתנה היא [ServerRpc] המציינת שזו פונקציה שמבצעת קריאה לשרת בלבד ולא ללקוחות אחרים ברשת. הפונקציה setAsPlayerTwoServerRpc() מצויה בתוך הקוד ומסומנת עם [ServerRpc], ובכך מציינת שהיא יכולה להתבצע רק על ידי השרת. הפונקציה מבצעת השמה של האובייקט הנוכחי (this.gameObject) למשתנה "TestLobby.playerTwo". בהקשר הזה, ניתן להניח שהמשתנה "TestLobby.playerTwo" הוא משתנה סטטי או עקומט גלובלי (מוגדר מחוץ לפונקציה) שמשמש לאחסון המשתנה המסומן כשחקן מספר שני במשחק. בסיכום, פונקציה זו מבצעת השמה של האובייקט הנוכחי כשחקן מספר שני במשחק, באמצעות הקריאה לשרת.

void Start()

הפונקציה Start() מופעלת בתחילת ריצת התוכנית ומבצעת פעולות ראשוניות להגדרת האובייקט.

הנה הסבר על מה שהפונקציה עושה:

1. השמה של ערך למשתנה minTimeBetweenTicks:

- המשתנה minTimeBetweenTicks מציין את הזמן המינימלי בין טיקים בשרת.

- הערך של minTimeBetweenTicks מחושב על פי הזמן בין טיקים המתאים למהירות הטיקים של השרת (SERVER\_TICK\_RATE).

2. הכנת מערך stateBuffer:

- נוצר מערך בגודל BUFFER\_SIZE של אובייקטים מסוג StatePayload.

- מערך זה משמש לאחסון מצב האובייקט לאורך זמן.

3. בדיקה האם הלקוח המקומי הוא השרת:

- אם הלקוח המקומי הוא השרת:

- נוצר מערך stateBufferServer בגודל BUFFER\_SIZE של אובייקטים מסוג StatePayload.

- מערך זה משמש לאחסון מצב האובייקט בצד השרת לאורך זמן.

בסיכום, הפונקציה Start() מבצעת השמות והכנות ראשוניות של משתנים ומערכים עבור האובייקט, ובודקת האם האובייקט הוא השרת כדי ליצור מערך נוסף עבור השרת.

private void Update()

הפונקציה Update() מופעלת בכל מעגל עידכון של המשחק ומבצעת פעולות תוך כדי התבצעות המשחק.

הנה הסבר על מה שהפונקציה עושה:

1. עידכון הטיימר:

- הטיימר timer מתעדכן בערך הזמן שחלף מאז עידכון המצב הקודם (Time.deltaTime).

- הטיימר משמש לניתור הזמן בין טיקים.

2. איסוף קלט (Input) במידת הצורך:

- אם הלקוח המקומי הוא בעל האובייקט, נקרא לפונקציה GatherInput() המאגדת את הקלט מהמשתמש.

3. ביצוע התבצעות הטיקים (Tick):

- בזמן שהטיימר timer משווה או עולה מערך הזמן בין הטיקים המינימלי (minTimeBetweenTicks), מתבצעות הפעולות הבאות:

3.1. אם הלקוח המקומי הוא בעל האובייקט:

- HandleTickClient(): טיפול בטיק הלקוח - כלול מניעת כדור הלקוח ותיקון אזור המעקב.

- יצירת latestServerState שהוא מופע של StatePayload, ומכיל את נתוני הטיק הנוכחי של השרת והמיקום הנוכחי של השרת.

- HandleServerReconciliation(): טיפול באיחוד הסרבור והלקוח - כלול תיקון הזזות ותיקון דריפה.

- serverPostion.Value מתעדכן עם מיקום האובייקט.

3.2. אם הלקוח המקומי אינו בעל האובייקט ו

אינו השרת:

- המיקום של האובייקט מתעדכן למיקום הנוכחי של השרת.

3.3. אם הלקוח המקומי הוא השרת ואינו בעל האובייקט:

- HandleTickServer(): טיפול בטיק השרת - כלול בעיקר הזזת השחקן השני.

3.4. הגדרת הטיק הנוכחי (currentTick) על ידי עלייתו ב-1.

4. בדיקה וטיפול נוסף רק אם הלקוח המקומי הוא השרת ובעל האובייקט:

- אם לוחץ על מקש בשם KeyCode.K, ואם ישנו שחקן מספר שני מחובר (TestLobby.playerTwo):

- הפונקציה startPoint() באובייקט onlineBall מתבצעת.

- אם אין שחקן מספר שני מחובר, נכתבת הודעת שגיאה בלוג שמציינת שהשחקן מספר שני לא מחובר.

בסיכום, הפונקציה Update() מבצעת פעולות עדכון רציפות על המשחק בהתאם לתזוזות, טיקים וקלט מהמשתמש. היא גם מתייחסת לתקשורת בין הלקוחות השונים ומתבצעת טיפולים נוספים בתנאי מסוים.

private void HandleTickServer()

-הפעולה ריקה בנתיים אם ארצה להוסיך משהו שרץ על השרת יש פונקציה מוכנה.

private void HandleTickClient()

* הפונקציה HandleTickClient() מתבצעת כחלק מתהליך טיק (Tick) בצד הלקוח.

הנה הסבר על מה שהפונקציה עושה:

1. יצירת אובייקט מסוג InputPayload:

- נוצר אובייקט מסוג InputPayload עם התכונות הבאות:

- tick: הטיק הנוכחי.

- y: ערך הקלט מהציר ה-Y (מיקום אנכי) של המשתמש.

2. שמירת המצב המתועדף במערך stateBuffer:

- מצב המשחק שמתועדף בטיק הנוכחי מעובר למערך stateBuffer במיקום המתאים בעזרת המודולו של הטיק ב-BUFFER\_SIZE.

- המצב מתועדף באמצעות קריאה לפונקציה ProcessMovement() עם הפרמטר inputPayload, שמחזירה את המצב המעודכן לפי הקלט המתקבל.

3. בדיקה האם הלקוח הוא המארח (השרת):

- אם הלקוח הוא המארח, הפונקציה מסתיימת ואין צורך בתהליך נוסף.

4. קריאה לשיטת moveOnServerRpc():

- המתוארת כ-[ServerRpc], ומבצעת העברת הפרמטרים inputPayload.tick ו-inputPayload.y לשרת באמצעות תהליך תקשורת ברשת.

בסיכום, הפונקציה HandleTickClient() מטפלת בתהליך טיק (Tick) על צד הלקוח. היא מציירת את הקלט מהמשתמש, מעבירה אותו למצב המשחק ומבצעת תהליך תקשורת עם השרת.

[ServerRpc]

void moveOnServerRpc(int tick, float Y)

הפונקציה moveOnServerRpc() היא שיטת [ServerRpc] שמופעלת על השרת.

הנה הסבר על מה שהפונקציה עושה:

1. יצירת אובייקט מסוג InputPayload:

- נוצר אובייקט מסוג InputPayload עם התכונות הבאות:

- tick: הטיק שקיבל מהפרמטר tick.

- y: ערך ה-Y שקיבל מהפרמטר Y.

2. שמירת המצב המתועדף במערך stateBufferServer:

- מצב המשחק שמתועדף בטיק הנוכחי מעובר למערך stateBufferServer במיקום המתאים בהתאם לערך הפרמטר tick.

- המצב מתועדף באמצעות קריאה לפונקציה ProcessMovement() עם הפרמטר input, שמחזירה את המצב המעודכן לפי הקלט המתקבל.

3. עדכון המיקום הנוכחי של השרת:

- serverPostion.Value מתעדכן עם מיקום האובייקט הנוכחי.

4. עדכון הטיק הנוכחי של השרת:

- currentTickServer.Value מתעדכן עם הערך של הפרמטר tick.

בסיכום, הפונקציה moveOnServerRpc() מבצעת העברת הנתונים מהלקוח לשרת ועדכון המצב המתועדף בצד השרת בהתאם לקלט המתקבל.

private void GatherMovement(InputPayload input)

-הפונקציה GatherMovement() מקבלת את הקלט המתקבל מהמשתמש ומבצעת תהליך איסוף התנועה בהתאם לקלט הזה.

הנה הסבר על מה שהפונקציה עושה:

1. בדיקת הקלט:

- אם ערך ה-Y של הקלט הוא -1, המשתנה state מקבל את הערך -1.

- אם ערך ה-Y של הקלט הוא 1, המשתנה state מקבל את הערך 1.

2. בדיקות ותהליכי תנועה:

- אם מיקום האובייקט על ציר ה-Y גדול מהגובה המינימלי וערך המשתנה state הוא -1:

- האובייקט מבצע תהליך תנועה באמצעות שימוש בפונקצית rb.MovePosition().

- התנועה מתבצעת באמצעות לינארית בין המיקום הנוכחי למיקום המבוקש, בתחושת חלקות של 90% מהמרחק הנדרש לקרות.

- אם מיקום האובייקט על ציר ה-Y קטן מהגובה המרבי וערך המשתנה state הוא 1:

- האובייקט מבצע תהליך תנועה באמצעות שימוש בפונקצית rb.MovePosition().

- התנועה מתבצעת באמצעות לינארית בין המיקום הנוכחי למיקום המבוקש, בתחושת חלקות של 90% מהמרחק הנדרש לקרות.

בסיכום, הפונקציה GatherMovement() מבצעת איסוף התנועה לפי הקלט המתקבל מהמשתמש. היא משנה את המצב הנוכחי של האובייקט ומבצעת תהליכי תנועה בהתאם למצב ולגבולות הגובה המוגדרים.

StatePayload ProcessMovement(InputPayload input)

void HandleServerReconciliation()

- הפונקציה HandleServerReconciliation() מתבצעת על ידי הלקוח ומבצעת את תהליך התיקון וההתאמה בין מצב השרת לבין מצב הלקוח.

הנה הסבר על מה שהפונקציה עושה:

1. חישוב טעות המיקום:

- מחשבים את טעות המיקום בין המצב האחרון בשרת (latestServerState) לבין המצב ב-buffer המקומי של הלקוח (stateBuffer) בעזרת פונקצית Vector3.Distance().

- הטעות מתקבלת כמרחק בין שני נקודות המיקום.

2. בדיקת טעות המיקום:

- אם טעות המיקום גדולה מ-1:

- מודפסת הודעת פתיחה ("We have to reconcile bro") בטור הנכון.

- מתבצעת התאמה של מיקום האובייקט של הלקוח למיקום האובייקט של השרת.

- מעדכנים את מיקום המצב האחרון בשרת (latestServerState) למיקום החדש של האובייקט של השרת.

- מעדכנים את מיקום המצב ב-buffer המקומי של הלקוח (stateBuffer) למיקום החדש של האובייקט של השרת.

בסיכום, הפונקציה HandleServerReconciliation() בודקת את טעות המיקום בין הלקוח לשרת ובמקרה של טעות גדולה, מבצעת תיקון והתאמה של מיקום הלקוח למיקום השרת. הפונקציה משמשת לטובת הסנכרון והתיקון בין הצד הלקוח לשרת בתהליך המשחק.

private void GatherInput()

- הפונקציה GatherInput() מבצעת איסוף קלט מהמשתמש בכדי להשתמש בו בתהליך המשחק.

הנה הסבר על מה שהפונקציה עושה:

1. יצירת אובייקט מסוג FrameInput:

- נוצר אובייקט מסוג FrameInput עם התכונות הבאות:

- Y: הערך שמתקבל מקלט המשתמש בציר ה-Y, באמצעות קריאה לפונקצית UnityEngine.Input.GetAxisRaw("Vertical").

פונקצית זו מחזירה ערך רציף שמתקבל מקלט המשתמש בציר ה-Y (לעיתים ערך של -1, 0 או 1 בהתאם ללחיצה על מקשי החצים למעלה או למטה).

2. שמירת הקלט:

- הקלט שנאסף מהמשתמש מתווסף למשתנה Input, שהוא משתנה מסוג FrameInput.

בסיכום, הפונקציה GatherInput() מבצעת איסוף קלט מהמשתמש ושומרת את הערכים שנאספו במשתנה Input. ערכי הקלט ישמשו בתהליך המשחק לצורך טיפול בתנועה של האובייקט.

# רפלקציה:

רפלקציה

במהלך עבודה על פרויקט זה חוויתי הרבה הצלחות וגם כמה כשלונות, אך ללא ספק, אני

שמח ומתגאה בתוצאה הסופית. למדתי לקרוא דוקמנטציה, לעבוד עם api חיצוני, עסקתי בהמון טכנולוגיות חדשות והכי חשוב למדתי ללמוד בעצמי, וזה נותן לי אמונה שאני יכול לפתח מה שאני רוצה עם רק אשים בזה את השקעה.

לוקח איתי לדרך את תכנון הזמן שהיה לי ואת הנחישות לסיים את הפרויקט, גם הדרך

שבה התמודדתי עם כשלונות ועם דברים שלא הלכו כמצופה ללא ספק תמשיך איתי להמשך

ואני גאה בה.

שמח להגיד שיצא לי לעזור למספר תלמידים מהכיתה וללא ספק גם להיעזר בהם, חושב

שמשוב העמיתים שנתנו אחד לשני עזר לשני הצדדים ובנימה אישית עזר לי לחדד דברים

בקוד שלי שלא חשבתי עליהם קודם.

בראייה לאחור, ללא ספק הייתי מתכנן את מחלקות הפרויקט בצורה טובה יותר וברורה

יותר, היו כמה מקרים לאורך כתיבת הפרויקט שפתאום נפלה בי ההבנה כי שכחתי שדה

מהותי, דבר שגרר "עבודה שחורה" שיכלה להימנע.

במידה והיו בידיי משאבים נוספים, הייתי מריץ בדיקות נוספות, אך הבדיקות שבצעתי מכסות לדעתי בצורה מספקת .

בהחלט, והפרויקט רץ בצורה חלקה.

# ביבליוגרפיה:

Here is the APA-style bibliography for the provided YouTube links:

1. Best Multiplayer Networking Solution for Unity 2021 | Photon PUN 2 vs Mirror vs PlayFab vs MLAPI [Video file]. (2021, May 17). Retrieved from <https://www.youtube.com/watch?v=7w19z0c29Uo&t=133s&pp=ygUhYmVzdCBuZXR3b3JrIHNvbHV0aW9ucyAgZm9yIHVuaXR5>

2. What Unity Networking Solution Should I Use? [Video file]. (2022, December 22). Retrieved from <https://www.youtube.com/watch?v=fb5PzAewDjU&pp=ygUhYmVzdCBuZXR3b3JrIHNvbHV0aW9ucyAgZm9yIHVuaXR5>

3. Unity OFFICIAL Multiplayer is FINALLY HERE! [Video file]. (2022, September 20). Retrieved from <https://youtu.be/ecyK0vHmbpQ>

4. COMPLETE Unity Multiplayer Tutorial (Netcode for Game Objects) [Video file]. (2022, September 26). Retrieved from <https://youtu.be/3yuBOB3VrCk>

5. Making a MULTIPLAYER Game? Join your Players with LOBBY! [Video file]. (2022, November 29). Retrieved from <https://youtu.be/-KDlEBfCBiU>

6. How to use Unity Relay, Multiplayer through FIREWALL! (Unity Gaming Services) [Video file]. (2022, December 2). Retrieved from <https://youtu.be/msPNJ2cxWfw>

7. Unity UI Tutorial | An introduction [Video file]. (2021, August 10). Retrieved from <https://www.youtube.com/watch?v=IuuKUaZQiSU>

8. How To Do Client Prediction [Unity Tutorial/C#] [Video file]. (2022, January 17). Retrieved from <https://youtu.be/TFLD9HWOc2k>

# הדפס הקוד:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Netcode;

using TMPro;

public class onlineBall : NetworkBehaviour

{

private NetworkVariable<Vector2> serverPostion = new NetworkVariable<Vector2>();

private const float maxHight = 5;

private const float minHight = -5;

private const float maxWidth = 7.8f;

private const float minWidth = -7.8f;

private const float playerMax = 1;

private const float playerMin = -1.25f;

private const float playerWidth = 0.2f;

static Rigidbody2D rb;

[SerializeField] static private float speed = 5.0f;

private void Start()

{

rb = GetComponent<Rigidbody2D>();

TestLobby.ball = this.gameObject;

}

public override void OnNetworkSpawn()

{

base.OnNetworkSpawn();

if (IsServer)

{

serverPostion.Value = transform.position;

}

else

{

transform.position = serverPostion.Value;

}

}

public static void startPoint()

{

rb.position = new Vector2(0, 0);

rb.velocity = new Vector2(Random.Range(-1f, 1f), Random.Range(-1f, 1f)).normalized \* speed;

}

private void Update()

{

if (IsServer)

{

serverPostion.Value = transform.position;

}

else

{

transform.position = serverPostion.Value;

}

if (!IsServer) return;

// Limit the speed of the ball

if (rb.velocity.magnitude > 20)

{

rb.velocity = rb.velocity.normalized \* 20;

}

if(transform.position.y >= maxHight)

{

if (rb.velocity.y > 0)

{

rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y \* -1);

}

}

if (transform.position.y <= minHight)

{

if (rb.velocity.y < 0)

{

rb.velocity = new Vector2(rb.velocity.x, rb.velocity.y \* -1);

}

}

if (transform.position.x >= maxWidth)

{

if (transform.position.x >= maxWidth + 1.2)

{

startPoint();

addScorePlayerOneClientRpc();

}

if (transform.position.x <= maxWidth + playerWidth)

{

if (TestLobby.playerTwo.transform.position.y > transform.position.y + playerMin && TestLobby.playerTwo.transform.position.y < transform.position.y + playerMax)

{

if (rb.velocity.x > 0)

{

rb.velocity = new Vector2(rb.velocity.x \* -1, rb.velocity.y);

}

}

}

}

if (transform.position.x <= minWidth)

{

if (transform.position.x <= minWidth - 1.2)

{

startPoint();

addScorePlayerTwoClientRpc();

}

if (transform.position.x >= minWidth - playerWidth)

{

if (TestLobby.playerOne.transform.position.y > transform.position.y + playerMin && TestLobby.playerOne.transform.position.y < transform.position.y + playerMax)

{

if (rb.velocity.x < 0)

{

rb.velocity = new Vector2(rb.velocity.x \* -1, rb.velocity.y);

}

}

}

}

}

void addScorePlayerOne()

{

TestLobby.playerOneScore.text = (int.Parse(TestLobby.playerOneScore.text) + 1).ToString();

if(int.Parse(TestLobby.playerOneScore.text) == 5)

{

TestLobby.playerOneScore.text = "$";

TestLobby.playerTwoScore.text = ":/";

}

}

void addScorePlayerTwo()

{

TestLobby.playerTwoScore.text = (int.Parse(TestLobby.playerTwoScore.text) + 1).ToString();

if (int.Parse(TestLobby.playerTwoScore.text) == 5)

{

TestLobby.playerOneScore.text =":/";

TestLobby.playerTwoScore.text = "$";

}

}

[ClientRpc]

void addScorePlayerOneClientRpc()

{

addScorePlayerOne();

}

[ClientRpc]

void addScorePlayerTwoClientRpc()

{

addScorePlayerTwo();

}

}

using UnityEngine;

using TMPro;

public class setScorePlayerOne : MonoBehaviour

{

[SerializeField] private TextMeshProUGUI playerOneScore;

// Start is called before the first frame update

void Start()

{

TestLobby.playerOneScore = playerOneScore;

}

// Update is called once per frame

}

using UnityEngine;

using TMPro;

public class setScorePlayerTwo : MonoBehaviour

{

[SerializeField] private TextMeshProUGUI playerTwoScore;

// Start is called before the first frame update

void awake()

{

TestLobby.playerTwoScore = playerTwoScore;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class setText : MonoBehaviour

{

public TextMeshProUGUI MainText;

public TextMeshProUGUI SubText;

public void SetText(string mainText,string subText)

{

MainText.text = mainText;

SubText.text = subText;

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Services.Authentication;

using Unity.Services.Core;

using Unity.Services.Lobbies;

using Unity.Services.Lobbies.Models;

using Unity.Services.Relay;

using Unity.Services.Relay.Http;

using Unity.Services.Relay.Models;

using Unity.Netcode;

using Unity.Netcode.Transports.UTP;

using Unity.Networking.Transport;

using Unity.Networking.Transport.Relay;

using NetworkEvent = Unity.Networking.Transport.NetworkEvent;

using QFSW.QC;

using TMPro;

using UnityEngine.UI;

using System.Threading.Tasks;

public class TestLobby : MonoBehaviour

{

//text score

public static TextMeshProUGUI playerOneScore;

public static TextMeshProUGUI playerTwoScore;

[SerializeField] private GameObject scoreHUD;

// Start is called before the first frame update

[SerializeField] private GameObject LobbyTemplate;

[SerializeField] private GameObject LobbiesMenu;

[SerializeField] private GameObject MembersMenu;

[SerializeField] private Transform LobbiesContent;

[SerializeField] private Transform MembersContent;

[SerializeField] private GameObject HUD;

static public GameObject playerOne;

static public GameObject playerTwo;

static public GameObject ball;

const string RELAY\_JOIN\_CODE = "RELAY\_JOIN\_CODE";

private static Lobby hostLobby;

private float timer;

private void Update()

{

HandLobbyTimer();

}

private async void HandLobbyTimer()

{

if (hostLobby != null && hostLobby.HostId == AuthenticationService.Instance.PlayerId)

{

timer -= Time.deltaTime;

if (timer < 0f)

{

timer = 15f;// 4 times a minute

await LobbyService.Instance.SendHeartbeatPingAsync(hostLobby.Id);

}

}

}

private async void Start()

{

await UnityServices.InitializeAsync();

AuthenticationService.Instance.SignedIn += () =>

{

Debug.Log("signed in " + AuthenticationService.Instance.PlayerId);

};

try

{

await UnityServices.InitializeAsync();

await AuthenticationService.Instance.SignInAnonymouslyAsync();

var playerID = AuthenticationService.Instance.PlayerId;

scoreHUD.SetActive(false);

}

catch (LobbyServiceException e)

{

Debug.Log(e);

}

}

[Command]

public async void CreateLobby()

{

try

{

string lobbyName = "MyLobby";

int maxPlayers = 2;

CreateLobbyOptions options = new CreateLobbyOptions();

options.IsPrivate = false;

options.Data = new Dictionary<string, DataObject>

{

{RELAY\_JOIN\_CODE, new DataObject(DataObject.VisibilityOptions.Member,"0") },

};

Lobby lobby = await LobbyService.Instance.CreateLobbyAsync(lobbyName, maxPlayers, options);

Debug.Log("created Lobby! " + lobby.Name + " " + lobby.MaxPlayers);

hostLobby = lobby;

ListMembers();

}

catch (LobbyServiceException e)

{

Debug.Log(e);

}

}

[Command]

public async void ListLobbies()

{

try

{

QueryResponse queryResponse = await Lobbies.Instance.QueryLobbiesAsync();

Debug.Log(" Lobbies found: " + queryResponse.Results.Count);

DestroyChilds(LobbiesContent);

foreach (Lobby lobby in queryResponse.Results)

{

Debug.Log(lobby.Name + " : " + lobby.Id + " : " + lobby.LobbyCode);

var t = Instantiate(LobbyTemplate, LobbiesContent);

t.GetComponent<setText>().SetText(lobby.Name, lobby.Id);

t.GetComponent<Button>().onClick.AddListener(delegate { JoinLobby(lobby.Id); });

}

}

catch (LobbyServiceException e)

{

Debug.Log(e);

}

}

[Command]

public async void JoinLobby(string id)

{

try

{

hostLobby = await LobbyService.Instance.JoinLobbyByIdAsync(id);

ListMembers();

}

catch (LobbyServiceException e)

{

Debug.Log(e);

}

}

public async void ListMembers()

{

LobbiesMenu.SetActive(false);

MembersMenu.SetActive(true);

DestroyChilds(MembersContent);

Debug.Log("Refrash");

hostLobby = await Lobbies.Instance.GetLobbyAsync(hostLobby.Id);

var members = hostLobby.Players;

foreach (Player item in members)

{

string Name = item.Id;

bool IsHost = (item.Id == hostLobby.HostId);

var t = Instantiate(LobbyTemplate, MembersContent);

t.GetComponent<setText>().SetText(Name, IsHost.ToString());

t.GetComponent<Button>().onClick.AddListener(delegate { KickPlayer(Name); });

}

}

private async void KickPlayer(string Id)

{

if (AuthenticationService.Instance.PlayerId == hostLobby.HostId || AuthenticationService.Instance.PlayerId == Id)

{

await LobbyService.Instance.RemovePlayerAsync(hostLobby.Id, Id);

}

}

private void DestroyChilds(Transform parent)

{

while (parent.childCount > 0)

{

DestroyImmediate(parent.GetChild(0).gameObject);

}

}

[Command]

public async void StartRelay()

{

try

{

string joinCode = await TestRelay.CreateRelay(4);

await Lobbies.Instance.UpdateLobbyAsync(hostLobby.Id, new UpdateLobbyOptions

{

Data = new Dictionary<string, DataObject>

{

{RELAY\_JOIN\_CODE, new DataObject(DataObject.VisibilityOptions.Member, joinCode) }

}

}

);

HUD.SetActive(false);

scoreHUD.SetActive(true);

}

catch (LobbyServiceException e)

{

Debug.LogError(e);

}

}

public void JoinRelay()

{

try

{

string code = hostLobby.Data[RELAY\_JOIN\_CODE].Value;

TestRelay.JoinRelay(code);

HUD.SetActive(false);

scoreHUD.SetActive(true);

}

catch (RelayServiceException e)

{

Debug.LogError(e);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Unity.Services.Authentication;

using Unity.Services.Core;

using Unity.Services.Lobbies;

using Unity.Services.Lobbies.Models;

using Unity.Services.Relay;

using Unity.Services.Relay.Http;

using Unity.Services.Relay.Models;

using Unity.Netcode;

using Unity.Netcode.Transports.UTP;

using Unity.Networking.Transport;

using Unity.Networking.Transport.Relay;

using NetworkEvent = Unity.Networking.Transport.NetworkEvent;

using QFSW.QC;

using TMPro;

using UnityEngine.UI;

using System.Threading.Tasks;

public class TestRelay : MonoBehaviour

{

[Command]

public async static Task<string> CreateRelay(int MaxPlayers)

{

try {

Allocation allocation = await RelayService.Instance.CreateAllocationAsync(MaxPlayers);

string joinCode = await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);

Debug.Log(joinCode);

RelayServerData relayServerData = new RelayServerData(allocation, "dtls");

NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

NetworkManager.Singleton.StartHost();

return joinCode;

}

catch (RelayServiceException e)

{

Debug.Log(e);

return null;

}

}

[Command]

public async static void JoinRelay(string joinCode)

{

try

{

Debug.Log("join Relay with " + joinCode);

JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(joinCode);

RelayServerData relayServerData = new RelayServerData(joinAllocation, "dtls");

NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

NetworkManager.Singleton.StartClient();

}

catch (RelayServiceException e)

{

Debug.Log(e);

}

}

}

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

using Unity.Netcode;

public struct InputPayload

{

public int tick;

public float y;

}

public struct StatePayload

{

public int tick;

public Vector3 position;

}

namespace adirGames {

public class PlayerController : NetworkBehaviour, IPlayerController {

#region vars

// Public for external hooks

public FrameInput Input { get; private set; }

[SerializeField] private Rigidbody2D rb;

#endregion

private NetworkVariable<Vector2> serverPostion = new NetworkVariable<Vector2>();

private NetworkVariable<int> currentTickServer = new NetworkVariable<int>();

private const float minHight =-4.3f;

private const float maxHight = 4.3f;

// Shared

private float timer;

private int currentTick;

private float minTimeBetweenTicks;

private const float SERVER\_TICK\_RATE = 30f;

private const int BUFFER\_SIZE = 1024;

private float Speed = 15;

private int state =0;

// Client specific

private StatePayload[] stateBuffer;

private StatePayload latestServerState;

// server specific

private StatePayload[] stateBufferServer;

public override void OnNetworkSpawn()

{

if (IsServer)

{

serverPostion.Value = transform.position;

currentTickServer.Value = 0;

if(IsOwner)

{

TestLobby.playerOne = this.gameObject;

transform.position = new Vector2(-8, 0);

}

else

{

transform.position = new Vector2(8, 0);

}

}else if(IsOwner)

{

setAsPlayerTwoServerRpc();

}

}

[ServerRpc]

void setAsPlayerTwoServerRpc()

{

TestLobby.playerTwo = this.gameObject;

}

void Start()

{

minTimeBetweenTicks = 1f / SERVER\_TICK\_RATE;

stateBuffer = new StatePayload[BUFFER\_SIZE];

if(IsServer)

{

stateBufferServer = new StatePayload[BUFFER\_SIZE];

}

}

private void Update()

{

timer += Time.deltaTime;

if(IsOwner) GatherInput();

while (timer >= minTimeBetweenTicks)

{

timer -= minTimeBetweenTicks;

if (IsOwner)

{

HandleTickClient();

latestServerState = new StatePayload { tick = currentTickServer.Value, position = new Vector3(serverPostion.Value.x, serverPostion.Value.y) };

if (!IsHost) HandleServerReconciliation();

if (IsHost) serverPostion.Value = transform.position;

}

if (!IsOwner && !IsServer)

{

transform.position = serverPostion.Value;

}

if (IsServer && !IsOwner)

{

HandleTickServer();

}

currentTick++;

}

if(IsServer && IsOwner)

{

if (UnityEngine.Input.GetKeyDown(KeyCode.K))

{

if (TestLobby.playerTwo)

{

onlineBall.startPoint();

}

else

{

Debug.LogError("player 2 not connected");

}

}

}

}

private void HandleTickServer()

{

}

private void HandleTickClient()

{

InputPayload inputPayload = new InputPayload

{

tick = currentTick,

y = Input.Y,

};

stateBuffer[currentTick % BUFFER\_SIZE] = ProcessMovement(inputPayload) ;

if (IsHost) return;

moveOnServerRpc(inputPayload.tick, inputPayload.y);

}

[ServerRpc]

void moveOnServerRpc(int tick, float Y)

{

InputPayload input = new InputPayload

{

tick = tick,

y = Y,

};

StatePayload statePayload = ProcessMovement(input);

stateBufferServer[tick] = statePayload;

serverPostion.Value = transform.position;

currentTickServer.Value = tick;

}

private void GatherMovement(InputPayload input)

{

if (input.y == -1 )

{

state = -1;

}

if(input.y == 1)

{

state = 1;

}

if(transform.position.y > minHight && state == -1)

{

rb.MovePosition(Vector3.Lerp(rb.position, rb.position + minTimeBetweenTicks \* Speed \* new Vector2(0, state), 0.9f));

}

if(transform.position.y < maxHight && state == 1)

{

rb.MovePosition(Vector3.Lerp(rb.position, rb.position + minTimeBetweenTicks \* Speed \* new Vector2(0, state), 0.9f));

}

}

StatePayload ProcessMovement(InputPayload input)

{

;

// Should always be in sync with same function on Server

GatherMovement(input);

return new StatePayload()

{

tick = input.tick,

position = transform.position,

};

}

void HandleServerReconciliation()

{

int serverStateBufferIndex = latestServerState.tick % BUFFER\_SIZE;

float positionError = Vector3.Distance(latestServerState.position, stateBuffer[serverStateBufferIndex].position);

if (positionError > 1f)

{

Debug.Log("We have to reconcile bro");

transform.position =serverPostion.Value;

latestServerState.position = serverPostion.Value;

stateBuffer[serverStateBufferIndex].position = latestServerState.position;

}

}

#region Gather Input

private void GatherInput()

{

Input = new FrameInput

{

Y = UnityEngine.Input.GetAxisRaw("Vertical")

};

}

#endregion

}

}