

הכפר הירוק

חלופת סייבר : מנחה – יהודה אור

Card game with real time chat

רוני לוטן ת.ז 329604151

Joker Game

מילון מונחים

שקע – Socket

תהליכון – Thread

גיבוב – Hashing

מבוא

ייזום

תיאור ראשוני של הפרויקט

הפרויקט הוא פלטפורמה שבה ניתן לשחק את המשחק “Joker Game”, המיועדת לאפשר ללקוחות לשחק ולתקשר אחד עם השני בזמן אמת. כל לקוח שרוצה לשחק צריך להירשם ולאחר מיכן ליצור לובי או להצטרף אל לובי. בתוך הלובי המשתמשים יכולים לתקשר זה עם זה בצורה מאובטחת.

המחשק עצמו הוא משחק קלפים שנועד למינימום שלושה שחקנים שבו המטרה היא לסיים את המחשק כאשר בידך אין את קלף הג'וקר. השחקן שמסיים את המשחק עם הג'וקר מפסיד ושאר השחקנים מנצחים.

בחרתי במשחק הזה מכיוון שיש בו הרבה שחקנים ורציתי להתנסות בשרת מרובה לקוחות. בחרתי גם כי התקשורת בין השחקנים חשובה מאוד ובאמצעותה המשחק יכול להתקדם לכן חשוב מאוד ליצור צ'אט שיאפשר זאת.

אתגרים שאני צופה לי בפרויקט הם יצירת קשר מאובטח בין השרת ללקוח, הימנעות מקריסות במקרי קצה לדוגמא כאשר שחקן מתנתק בזמן המשחק או כשהשרת סוגר את המשחק, והרצה באותו הזמן של כמה זירות ללא בעיות ביניהם.

הגדרת הלקוח

הפרויקט מיועד לכולם כאשר הדגש הוא על ילדים שרוצים לשחק עם חבריהם משחקים על המחשב. המחשק הוא משחק חשיבה שדורש מהשחקן לפתח אסטרטגיות ולתקשר בצורה חכמה עם שאר השחקנים לכן מי שישחק במחשק הזה לרוב יהיו ילדים שרוצים אתגר מחשבתי.

הגדרת יעדים ומטרות

מטרת הפרויקט היא לאפשר לאנשים לשחק מחשק קלפים עם הרבה משתתפים על המחשב בצורה יעילה ומאובטחת. המטרה שלי היא שהתקשורת בין השחקן לשרת תהיה מהירה ובטוחה ושהחשקנים יוכלו ליצור זירות עם חבריהם ולשחק כמה שהם רוצים.

בעיות תועלת וחסכונות

הבעיה העיקרית היא שאת המשחק הזה אפשר לשחק רק כאשר יש חפיסת קלפים בהישג יד וגם קל מאוד לרמות בו. בעזרת הפרויקט שלי יהיה ניתן לשחק את המחשק בכל מצב בצורה מסודרת וללא תקלות לא נעימות כמו הצצה בקלפים של שחקן אחר או איבוד של קלף. בנוסף לכך המשחק עצמו הוא משחק מעניין שמאפשר לחברים לעשות משהוא מהנה בזמנם הפנוי. למרות שהמשחק יכול להיות קשה להבנה הפרויקט שלי מאפשר לשחקנים ללמוד אותו בצורה נוחה וברורה ווכתוצאה מכך לפתח חשיבה אסטרטגית.

סקירת פתרונות קיימים

המשחק שבחרתי הוא לא הכי פופולרי אבל קיימים מאות של משחקי קלפים שונים באינטרנט שדומים אליו. ישנם אתרים של משחק הקלפים רבעיות שבהם גם אפשר להתחבר ללובי ולשחק עם אנשים אחרים, לעשות צ'אט עם אנשים אחרים ושישמר מידע על הלקוחות. יש הרבה משחקיי קלפים באינטרנט שמהם קיבלתי השראה ליישם את משחק הקלפים שאני ממש אוהב על המחשב. הנה כמה דוגמאות למשחקיי קלפים שיש באינטרנט: <https://cardgames.io/>, <https://solitaired.com/> ועוד...

סקירת טכנולוגית הפרויקט

הפרויקט שלי משתמש בטכנולוגיות הבאות:

1. פיתוח אפליקצית שרת-לקוח באמצעות שפת #C. #C היא [שפת תכנות עילית](https://he.wikipedia.org/wiki/%D7%A9%D7%A4%D7%AA_%D7%AA%D7%9B%D7%A0%D7%95%D7%AA_%D7%A2%D7%99%D7%9C%D7%99%D7%AA) מרובת-[פרדיגמות](https://he.wikipedia.org/wiki/%D7%A4%D7%A8%D7%93%D7%99%D7%92%D7%9E%D7%AA_%D7%AA%D7%9B%D7%A0%D7%95%D7%AA), [מונחית עצמים](https://he.wikipedia.org/wiki/%D7%AA%D7%9B%D7%A0%D7%95%D7%AA_%D7%9E%D7%95%D7%A0%D7%97%D7%94-%D7%A2%D7%A6%D7%9E%D7%99%D7%9D) בעיקרה, המשלבת רעיונות. היתרונות שלה הם:
   1. היא תומכת בתיכנות מונחה עצמים, פונקציונלי ואסיכרוני. זה מאוד עוזר לכתוב אפליקציות בדסקטופ, משחקים, אפליקציות ברשת ועוד.
   2. היא שפה חזקה ובטוחה. היא משתמשת ב-strong typing שזה עוזר למצוא טעויות. תכונות כמו סוגי משתנים שיכולים להיות null וניהול זיכרון בטוח עוזרים לצמצם את כמות הבאגים.
   3. Visual Studios שתומכת בפיתוח ב-#C מספקת כלים רבים למען דיבאגינג, profiling ו- refactoring.
2. פיתוח ממשק משתמש גרפי באמצעות תשתית winfroms.
3. תקשורת בין לקוחות ושרת באמצעות פרוטוקול TCP.
4. תקשורת מאובטחת באמצעות פרוטוקול הצפנה SSL.
5. שמירה מאובטחת של סיסמאות באמצעות hash ו-salt ו- pepper
6. הפרויקט שלי נכתב בשפת התיכנות #C שמאוד שימושית בעיקר בלעצב את ה-UI.את העיצוב עשיתי באמצעות Winforms. Winforms זה תשתית ממשק משתמש לבניית יישומי שולחן עבודה של Windows ב-NET. באמצעות C# או VB.NET. זה מספק מעצב גרירה ושחרור ב-Visual Studio, יחד עם סט כלים (לחצנים, תיבות טקסט, תוויות וכו') ליצירת ממשקי משתמש גרפיים (GUIs).

בנוסף על כך השתמשתי בTcpClient וTcpListener- שהם מחלקות ברמה גבוהה שעוטפות פונקציונליות שקע ברמה נמוכה יותר הניתנות על ידי NET Framework. גם TcpClient וגם TcpListenerמשתמשים באופן פנימי במחלקה `System.Net.Sockets.Socket`, המספקת את פונקציונליות הליבה עבור תקשורת TCP ו-UDP ברשת. הם בעצם עטיפות נוחות סביב 'Socket' כדי לפשט דפוסי TCP נפוצים.

בשביל להצפין השתמשתי בשילוב של RSA בשביל הצפנה אסימטרית וב-AES בשביל הצפנה סימטרית. הסיבה ששילבתי היא שהצפנה אסימטרית היא איטית ולא יעילה כאשר מדובר בכמיות גדולות של מידע ולכן עדיף להשתמש בהצפנה סימטרית שהיא יותר מהירה ויעילה. באמצעות הRSA אני מעביר את מפתח ה-AES בצורה מאובטחת ועכשיו אני יכול להשתמש רק בהצפנה סימטרית.

תהליכונים (Threads) הם

תיחום הפרויקט

הפרויקט שלי עוסק בתחומים הבאים:

1. תקשורת ברשתות בעזרת sockets ופרוטוקול TCP .
2. הצפנה והגנת סייבר.
   1. התקשורת מאובטחת באמצעות שילוב של הצפנה אסימטרית והצפנה סימטרית.
   2. שמירה מאובטחת של סיסמאות באמצעות פונקציית hash המחוזקת על ידי salt ו-pepper.
3. ממשק משתמש גרפי הממומש באמצעות תשתית winforms.
4. תיכנות בשפת #C ופיתוח בסביבת Visual Studio של Microsoft.
5. תיכנות multi-threading. תהליכונים נפרדים ל-UI ולתקשורת וניהול המשחק בצד הלקוח, ותהליכונים נפרדים לתקשורת עם כל לקוח ולכל זירה בצד השרת.

פירוט תיאור המערכות (אפיון)

תיאור מפורט של המערכת

המערכת שלי היא משחק רב משתתפים. בהתחלה כאשר פותחים את המשחק רואים מסך עם שני כפתורים – התחברות והרשמה. כל כפתור מוביל למסך שממנו השחקן יכול להירשם/ להתחבר. במסך ההרשמה יש כפתור שמעביר למסך ההתחברות ולהפך. לאחר שהלקוח מתחבר/ נרשם הוא מועבר למסך הבית. במסך הבית יש לו כפתור שיוצר לו זירה ומכניס אותו לזירה ותיבת טקסט שבה הוא מכניס קוד של זירה קיימת. לאחר שהוא נכנס לזירה הוא מחכה שעוד שחקנים יצטרפו, יש לו את האפשרות לצאת מהזירה, ועכשיו הוא מחכה שהמשחק יתחיל. מי שיכול להתחיל את המשחק הוא המנהל של הזירה שהוא זה שיצר את הזירה. כאשר המשחק מתחיל כל השחקנים מקבלים ארבעה קלפים חוץ משחקן אחד שמקבל חמישה. השחקן רואה את הקלפים שמולו ויכול לבחור מבינהים איזה קלף להציע לשחקן הבא אחריו באמצעות לחיצה כפולה על העכבר, יש לו גם כפתור שבו הוא מסכים לקבל קלף או יכול לדחות את הקלף שהוצע לו. בחירת הקלפים מנוטרלת כאשר לא תורך. לאחר שהמשחק מסתיים כאשר לשחקן יש רק ארבעה קלפים מאותו הסוג קופצת הודעה על המסך שהמשחק נגמר ומי היה השחקן שהפסיד. לאחר מכן אתה חוזר לזירה שבה יש לך את האפשרות להישאר או לעזוב. על כל שחקן נשמרים הנתונים של כמה משחקים הוא ניצח וכמה משחקים הוא הפסיד.

פירוט היכולות ברמת כותרת

יכולות שרת:

* השרת מאפשר למספר רב של לקוחות להתחבר אליו בו זמנית
* השרת מאפשר תקשורת מוצפנת ומאובטחת עם כל אחד מהלקוחות שהתחברו
* השרת מאפשר לכל לקוח ליצור משתמש ושומר אצלו את פרטי ההתקשרות של הלקוח ופרטים נוספים
* השרת מאפשר ללקוחות לבצע login ומוודא את נכונות פרטי ההתקשרות שלהם (שם משתמש וסיסמא)
* השרת מאפשר ללקוח ליצור זירת משחק חדשה ושומר את פרטי הזירה אצלו
* השרת מאפשר ללקוח להצטרף לזירה קיימת ויודע לאמת את פרטי ההצטרפות (שם זירה וקוד כניסה)
* השרת מאפשר את הקיום של כמה זירות משחק כך שניתן לקיים מספר משחקים במקביל
* השרת מאפשר למנהל הזירה להתחיל משחק חדש
* השרת מאפשר לכל הלקוחות שנמצאים באותה זירה לתקשר באמצעות צ'אט
* השרת מנהל את מהלך המשחק
  + מחלק את הקלפים בתחילת המשחק
  + מודיע לכל שחקן מתי מגיע תורו
  + מתווך בתקשורת בין שני השחקנים שצריכים להעביר ביניהם קלף
  + מעדכן את שאר השחקנים על התקדמות המשחק
  + מזהה כאשר המשחק מסתיים ומכריז על המפסיד
* השרת יכול לטפל במקרים שבהם שחקן עוזב את המשחק או הזירה בפתאומיות או באמצעות הודעה מסודרת.

יכולות לקוח

* התחברות לשרת
* הלקוח יכול להירשם בתור משתמש חדש
* הלקוח יכול לבצע login
* המידע שהלקוח מעביר מוגן ברשת
* הלקוח יכול ליצור זירה חדשה
* הלקוח יכול להתחבר לזירה קיימת בהינתן שיש לו את שם הזירה וקוד הכניסה
* הלקוח יכול להתנתק מהמשחק בכל שלב
* הלקוח יכול להתחיל את המשחק כאשר יש לו מספיק שחקנים בזירה שלו
* לאחר שהצטרף לזירה הלקוח יכול לתקשר באמצעות צ'אט עם כל הלקוחות האחרים באותה זירה
* בזמן המשחק הלקוח יכול לבצע את הפעולות הנדרשות ממנו:
  + בתורו הוא יכול להציע קלף לשחקן הבא בתור
  + הוא יכול לאשר או לסרב להצעת קלף של השחקן שלפניו

פירוט הבדיקות

בדיקת המערכת תיעשה במספר שלבים כדי לוודא פעולה תקינה כאשר בכל פעם נעלה את רמת המורכבות. האמצעי העיקרי כדי לוודא שהמערכת פועלת כראוי הוא קובץ התיעוד המשותף אשר יכיל את כל ההודעות שנרשמות על ידי השרת והלקוחות הפעילים לפי סדר הרישום שלהם. כך נוכל לבדוק שכל מה שאמור היה לקרות אכן קרה ובסדר בו אנו מצפים שיקרה. בנוסף כאשר נרצה להתבונן בערכם של משתנים ספציפיים או כאשר מסתבר מקריאת הקובץ שלא בוצעו קטעי קוד מסויימים או שקטעים אחרים בוצעו למרות שלא היו אמורים לרוץ, נשתמש בכלי ניפוי הבאגים של סביבת הפיתוח כדי להתחקות אחר שורות הקוד שרצו וערכי המשתנים בנקודות חשובות בקוד.

הבדיקות שיבוצעו הן:

**שלב א**

בשלב זה גם השרת וגם הלקוחות רצים על אותו המחשב ומשתמשים בכתובת ה-IP של ה-LocalHost, כלומר 127.0.0.1, כדי לפשט את התקשורת ביניהם. כמו כן ההודעות הנשלחות אינן מוצפנות.

1. בדיקות ראשוניות:
   1. הרצה של השרת ובדיקה שהוא מתחיל לרוץ, פותח תהליכון ונכנס ללואה המחכה לקבלת בקשות חיבור מלקוחות.
   2. הרצה של הלקוח ובדיקה שהוא מתחיל לרוץ, פותח תהליכון ומגיע לשורה בה נשלחת בקשת חיבור לשרת.
   3. הרצה של השרת ולאחר מכן הרצה של לקוח ובדיקה שמתבצע חיבור בין הלקוח והשרת כך שבשני הצדדים יש אובייקט TcpClient מחובר ומוכן להחלפת הודעות.
   4. בדיקה שבצד השרת נוצר אובייקט ה-ClientHandler עבור הלקוח ונפתח תהליכון שמנהל את התקשורת עם הלקוח באמצעות לולאה הדוגמת את שקע ה-TCP.
2. בדיקת תהליך האיתחול של פרוטוקול ההצפנה:
   1. בדיקה שהלקוח והשרת מייצרים זוג מפתחות לצורך הצפנת RSA.
   2. בדיקה שהלקוח שולח לשרת את המפתח הציבורי שלו ושהמפתח מתקבל בצד השרת.
   3. בדיקה שהשרת מייצר מפתח AES יחודי עבור הלקוח
   4. בדיקה שהשרת מצפין את מפתח ה-AES באמצעות מפתח ה-RSA הציבורי שקיבל מהלקוח ושולח אותו ללקוח, ושהלקוח מקבל את המפתח המוצפן, מפענח את ההצפנה ושומר את המפתח.
3. בדיקת רישום משתמש:
   1. בדיקה שלחיצה על כפתור "Connect" בחלון הלקוח פותחת את חלון ההתחברות.
   2. בדיקה שלחיצה על כפתור ה-"Register" בחלון ההתחברות סוגר את חלון ההתחברות ופותח את חלון רשום המשתמש.
   3. בדיקה שהפרטים הממולאים בחלון רשום המשתמש נשמרים כאשר החלון נסגר לאחר לחיצה על כפתור ה-"Register".
   4. בדיקה שכל הפרטים עוברים הליך תיקוף ושנפתחת תיבת שיח (DialogBox) עם הודאת שגיאה מתאימה אם אחד הפרטים לא עומד בדרישות.
   5. בדיקה שנוצרת הודעה מסוג RegisterRequest הנשלחת לשרת.
   6. בדיקה שהודעת ה- RegisterRequestמתקבלת בצד השרת ושפרטי הרישום מתקבלים בצורה נכונה.
   7. בדיקה שהסיסמא עוברת הליך גיבוב בתוספת Salt ו-Pepper.
   8. בדיקה שנוצר קובץ Json חדש המכיל את פרטי הרישום.
   9. בדיקה שנוצרת הודעת Register בצד השרת המאשרת את הצלחת פעולת הרישום ונשלחת ללקוח.
   10. בדיקה שהודעת ה-Register התקבלה אצל הלקוח והוא מציג תיבת שיח המאשרת ללקוח שהרישום הצליח.
   11. בדיקה שכאשר מנסים לרשום שם משתמש שכבר נרשם בעבר, פעולת הרישום נכשלת והודעת ה-Register מכילה את סיבת הכישלון המוצגת ללקוח בתיבת שיח.
4. בדיקת חיבור משתמש:
   1. בדיקה שהפרטים הממולאים בחלון ההתחברות נשמרים כאשר החלון נסגר לאחר לחיצה על כפתור ה-"Connect".
   2. בדיקה שכל הפרטים עוברים הליך תיקוף ושנפתחת תיבת שיח (DialogBox) עם הודאת שגיאה מתאימה אם אחד הפרטים לא עומד בדרישות.
   3. בדיקה שנוצרת הודעה מסוג LoginRequest הנשלחת לשרת.
   4. בדיקה שהודעת ה- LoginRequestמתקבלת בצד השרת ושפרטי ההתחברות מתקבלים בצורה נכונה.
   5. בדיקה שנשלף קובץ ה-Json המכיל את פרטי המשתמש
   6. בדיקה שהסיסמא מאומתת מול הסיסמא השמורה בקובץ ושיש התאמה ביניהן לאחר שהסיסמא שנשלחה עוברת גיבוב בתוספת מחרוזת ה-Salt השמורה בקובץ ומחרוזת ה-Pepper השמורה בתוך הקוד.
   7. בדיקה שנוצרת הודעת Login בצד השרת המאשרת את הצלחת פעולת ההתחברות ונשלחת ללקוח.
   8. בדיקה שהודעת ה- Loginהתקבלה אצל הלקוח והוא מציג תיבת שיח המאשרת ללקוח שההתחברות הצליחה.
   9. בדיקה שכאשר שם המשתמש אינו רשום אצל השרת, מתקבלת הודעת שגיאה אצל הלקוח עם הסיבה המתאימה.
   10. בדיקה שכאשר הסיסמא אינה תואמת את הסיסמא הרשומה אצל השרת מתקבלת הודאת שגיאה אצל הלקוח.
5. בדיקת יצירת זירת משחק:
   1. בדיקה שאחרי שהסתיים תהליך ההתחברות מופיע הכפתור "Join Lobby" בחלון הלקוח.
   2. בדיקה שלחיצה על כפתור ה-"Join Lobby" פותחת את חלון ההצטרפות לזירת משחק.
   3. בדיקה שלחיצה על כפתור ה-"Create New" סוגרת את חלון ההצורפות לזירת משחק ופותחת את חלון יצירת הזירה החדשה.
   4. בדיקה שהפרטים הממולאים בחלון יצירת הזירה החדשה נשמרים כאשר החלון נסגר לאחר לחיצה על כפתור ה-"Create".
   5. בדיקה שכל הפרטים עוברים הליך תיקוף ושנפתחת תיבת שיח עם הודאת שגיאה מתאימה אם אחד הפרטים לא עומד בדרישות.
   6. בדיקה שנוצרת הודעה מסוג CreateLobbyRequest הנשלחת לשרת.
   7. בדיקה שהודעת ה- CreateLobbyRequestמתקבלת בצד השרת ושפרטי הזירה מתקבלים בצורה נכונה.
   8. בדיקה שקוד הכניסה עובר הליך גיבוב בתוספת Salt ו-Pepper.
   9. בדיקה שנוצר קובץ Json חדש המכיל את פרטי הזירה החדשה.
   10. בדיקה שנוצרת הודעת CreateLobby בצד השרת המאשרת את הצלחת פעולת יצירת הזירה ונשלחת ללקוח.
   11. בדיקה שהודעת ה- CreateLobbyהתקבלה אצל הלקוח והוא מציג תיבת שיח המאשרת ללקוח שיצירת הזירה הצליחה.
   12. בדיקה שכאשר מנסים ליצור זירה עם שם שכבר קיים, פעולת היצירה נכשלת והודעת ה- CreateLobbyמכילה את סיבת הכישלון המוצגת ללקוח בתיבת שיח.
6. בדיקת הצטרפות לזירת משחק:
   1. בדיקה שהפרטים הממולאים בחלון ההצטרפות נשמרים כאשר החלון נסגר לאחר לחיצה על כפתור ה-"Enter".
   2. בדיקה שכל הפרטים עוברים הליך תיקוף ושנפתחת תיבת שיח עם הודאת שגיאה מתאימה אם אחד הפרטים לא עומד בדרישות.
   3. בדיקה שנוצרת הודעה מסוג JoinLobbyRequest הנשלחת לשרת.
   4. בדיקה שהודעת ה- JoinLobbyRequestמתקבלת בצד השרת ושפרטי הצטרפות מתקבלים בצורה נכונה.
   5. בדיקה שנשלף קובץ ה-Json המכיל את פרטי הזירה
   6. בדיקה שקוד הכניסה מאומת מול קוד הכניסה השמור בקובץ ושיש התאמה ביניהם לאחר שקוד הכניסה שנשלח עובר גיבוב בתוספת מחרוזת ה-Salt השמורה בקובץ ומחרוזת ה-Pepper השמורה בתוך הקוד.
   7. בדיקה שנוצרת הודעת JoinLobby בצד השרת המאשרת את הצלחת פעולת ההצטרפות ונשלחת ללקוח.
   8. בדיקה שהודעת ה- JoinLobbyהתקבלה אצל הלקוח והוא מציג תיבת שיח המאשרת ללקוח שההצטרפות הצליחה.
   9. בדיקה שכאשר שם הזירה אינו רשום אצל השרת, מתקבלת הודעת שגיאה אצל הלקוח עם הסיבה המתאימה.
   10. בדיקה שכאשר הקוד הכניסה אינו תואם את קוד הכניסה הרשום אצל השרת מתקבלת הודאת שגיאה אצל הלקוח.
7. בדיקת התחלת המשחק:
   1. בדיקה שכאשר מנהל הזירה נכנס לזירה מופיע כפתור "Start Game".
   2. בדיקה שאם אין מספיק שחקנים בזירה הכפתור במצב לא פעיל.
   3. בדיקה שכאשר נכנסים עוד שחקנים לזירה הכפתור עובר למצב פעיל אם יש מספיק שחקנים להתחיל משחק.
   4. בדיקה שאם יוצאים שחקנים מהזירה כך שאין מספיק שחקנים הכפתור עובר למצב לא פעיל.
   5. בדיקה שכאשר נלחץ הכפתור קורים הדברים הבאים:
      1. הלקוח שולח הודעת StartGame לשרת
      2. השרת מקבל את ההודעה ויוצר אובייקט של מחלקת "Game".
      3. פעולת DealCards של האובייקט נקראת וכל אחד מהשחקנים מקבל 4 קלפים אקראיים, חוץ מהשחקן הראשון שמקבל 5.
      4. השרת שולח הודעת DealCards לכל שחקן הכוללת את הקלפים שחולקו לו.
      5. הלקוח מקבל את הודעת ה-DealCards ושומר אצלו את הקלפים שנשלחו
      6. הקלפים מוצגים בחלון הלקוח.
      7. השרת שולח הודעת MakeOffer לשחקן הראשון (המחזיק 5 קלפים) והודעת ReceiveOffer לשחקן השני.
      8. הלקוח שהוא השחקן הראשון מקבל את הודעת ה-MakeOffer. בחלון ההודעות שלו מופיעה השורה: "XXX will offer cards to YYY" ומתחת לקלפים תופיע השורה: "Make and Offer".
      9. הלקוח שהוא השחקן השני מקבל את הודעת ה-ReceiveOffer. בחלון ההודעות שלו תופיע השורה: "XXX will offer cards to YYY" ומתחת לקלפים תופיע השורה: "Waiting to receive #1 offer from XXX". בנוסף יופיעו שני כפתורים "Accept" ו-"Reject" מתחת לתמונות, שניהם במצב לא פעיל.
      10. אצל כל השחקנים מופיע כפתור "End Game" במצב פעיל.
8. בדיקת מהלך המשחק:
   1. בדיקה שכשהשחקן המציע מבצע הקלקה כפולה על אחד הקלפים שלו קורים הדברים הבאים:
      1. מסביב לקלף מופיעה מסגרת אדומה.
      2. הלקוח שולח הודעת OfferCard לשרת הכוללת את הקלף אותו הוא מציע.
      3. השרת מקבל את הודעת ה- OfferCardושולח לכל השחקנים הודעת GameLog המכילה את הטקסט: "XXX made #1 offer to YYY". בנוסף הוא שולח לשחקן המקבל הודעת AcceptCard הכוללת את הקלף שהוצע.
      4. כל הלקוחות בזירה מקבלים את הודעת ה- GameLogומציגים אותה בחלון ההודעות.
      5. הודעת AcceptCard מגיעה לשחקן המקבל ומתחת לתמונות בחלון שלו מופיעה ההודעה "Do you accept #1 card from XXX". בנוסף שני הכפתורים "Accept" ו-"Reject" עוברים למצב פעיל.
   2. בדיקה שכשהשחקן המקבל לוחץ על כפתור "Accept" קורים הדברים הבאים:
      1. הלקוח המקבל שולח הודעת ResponseToOffer לשרת הכוללת את הערך "Accept". בנוסף הוא מציג את הקלף שקיבל בחלון שלו ומסתיר את שני הכפתורים "Accept" ו-"Reject".
      2. השרת מקבל את הודעת ה-ResponseToOffer ומזהה שהתשובה היא "Accept". השרת שולח את ההודעה שהתקבלה לשחקן המציע. השרת גם שולח הודעת GameLog המכילה את הטקסט: "YYY accepted card from XXX" לכל השחקנים בזירה. בנוסף השרת קורא לפעולת AdvanceGameState של אובייקט ה-Game כדי לעדכן את מצב המשחק אצלו, ושולח הודעת MakeOffer לשחקן שהיה קודם המקבל, והודעת ReceiveOffer לשחקן הבא אחריו בתור.
      3. השחקן המציע מקבל את הודעת ה- ResponseToOfferומזהה שהתשובה היא "Accept". הוא מוחק את הקלף שהציע מרשימת הקלפים שלו ומהתצוגה.
      4. אצל השחקן שהיה קודם המקבל מתקבלת הודעת ה-MakeOffer. בחלון ההודעות שלו מופיעה השורה: "YYY will offer cards to ZZZ" ומתחת לקלפים תופיע השורה: "Make an Offer".
      5. אצל השחקן הבא בתור תתקבל ההודעה ReceiveOffer. בחלון ההודעות שלו תופיע השורה: "YYY will offer cards to ZZZ" ומתחת לקלפים תופיע השורה: "Waiting to receive #1 offer from YYY". בנוסף יופיעו שני כפתורים "Accept" ו-"Reject" מתחת לתמונות, שניהם במצב לא פעיל.
   3. בדיקה שכשהשחקן המקבל לוחץ על כפתור "Reject" קורים הדברים הבאים:
      1. הלקוח המקבל שולח הודעת ResponseToOffer לשרת הכוללת את הערך "Reject". בנוסף הוא מעביר את שני הכפתורים "Accept" ו-"Reject" למצב לא פעיל. צתחת לקלפים מופיע הטקסט: "Waiting to Receive #2 offer from YYY".
      2. השרת מקבל את הודעת ה-ResponseToOffer ומזהה שהתשובה היא "Reject". השרת שולח את ההודעה שהתקבלה לשחקן המציע. השרת גם שולח הודעת GameLog המכילה את הטקסט: "YYY rejected card from XXX" לכל השחקנים בזירה.
      3. כל הלקוחות בזירה מקבלים את הודעת ה- GameLogומציגים אותה בחלון ההודעות.
      4. השחקן המציע מקבל את הודעת ה- ResponseToOfferומזהה שהתשובה היא "Reject". הוא משנה את תצוגת הקלף שנדחה כך שיופיע בשחור-לבן ומציג שוב את הודעת "Make an Offer" מתחת לקלפים.
   4. בדיקת שכאשר שחקן מסיים את תורו להציע ונשאר עם ארבעה קלפים זהים קורים הדברים הבאים:
      1. השרת מזהה, כחלק מטיפול בהודעת ResponseToOffer שקיבל אחרי התור האחרון, שיש מנצח ושולח הודעת AnnounceWinner לכל השחקנים הכוללת את שם המשתמש של המנצח והמפסיד. בנוסף השרת מאפס את אוביקט ה-Game.
      2. הודעת ה-AnnounceWinner מתקבלת אצל כל הלקוחות. בחלון ההודעות של כל לקוח מודפסות ההודעות: "Game over!!!", "The winner is XXX" ו-"The loser is YYY".
      3. תצוגת הקלפים אצל כל הלקוחות נעלמת. בנוסף לא רואים את כפתורי "Accept" ו-"Reject".
      4. כפתור ה-"Exit Lobby" מופיע פעיל אצל כל הלקוחות.
      5. כפתור ה-"Start Game" מופיע פעיל אצל מנהל הזירה.
9. בדיקת הפסקת משחק על ידי אחד השחקנים:
   1. בדיקה שכאשר שחקן לוחץ על כפתור "End Game" נשלחת הודעת InterruptGame לשרת. ההודעה כוללת את שם המשתמש של השחקן.
   2. בדיקה שההודעה מתקבלת אצל השרת והוא שולח אותה לכל השחקנים האחרים בזירה. בנוסף השרת מאפס את אובייקט ה-Game.
   3. בדיקה שכל הלקוחות מקבלים הודעת InterruptGame.
   4. בדיקה שכל הלקוחות מציגים בחלון ההודעות שלהם את הטקסט: "Game interrupted by XXX".
   5. בדיקה שכל הלקוחות מאפסים את הקלפים שלהם ואת תצוגת הקלפים. בנוסף לא רואים את כפתורי "Accept" ו-"Reject".
   6. בדיקה שכפתור ה-"Exit Lobby" מופיע פעיל אצל כל הלקוחות.
   7. בדיקה שכפתור ה-"Start Game" מופיע פעיל אצל מנהל הזירה.
10. בדיקת יציאה מהזירה של המנהל:
    1. בדיקה שלחיצה על כפתור "Exit Lobby" אצל מנהל הזירה גורמת לשיחת הודעת ExitLobbyRequest לשרת
    2. בדיקה שהשרת מקבל הודעת ExitLobbyRequest ומזהה שמדובר במנהל הזירה.
    3. בדיקה שהשרת שולח הודעת GameLog לכל שאר האורחים בלובי הכוללת את הטקסט "The host XXX left. Lobby YYY is closing!"
    4. בדיקה שהשרת שולח הודעת LobbyClosing לכל האורחים בזירת המשחק.
    5. בדיקה שהשרת מוציא את כל האורחים מהזירה ומוחק אותה מרשימת כל הזירות הפעילות.
    6. בדיקה שאובייקט ה-ClientHandler של כל אחד מהאורחים מופעל ושמאותחל התהליכון המטפל בקבלת הודעות מהלקוח.
    7. בדיקה שהלקוח של כל אחד מהאורחים בזירה מקבל את הודעת ה- GameLogוכותב את הטקס לחלון ההודעות שלו.
    8. בדיקה שהלקוח של כל אורח מקבל את הודעת ה- LobbyClosingומעדכן את מצב הלקוח בהתאם. בנוסף תצוגת חלון הלקוח משתנה כך שמופיעים הכפתורים "Join Lobby" ו-"Disconnect" במצב פעיל.

**שלב ב**

בשלב זה נפעיל את ההצפנה, כך שכל ההודעות הנשלחות גם על ידי השרת וגם על ידי הלקוחות מוצפנות. נבצע את הבדיקות הבאות.

1. בדיקה שהודעת LoginRequest מהלקוח לשרת מוצפנת לפני שהיא נשלחת.
2. בדיקה שהודעת LoginRequest מתקבלת בצד השרת כשהיא מוצפנת והפענוח שלה מצליח
3. בדיקה שהשרת שולח ללקוח הודעת Login מצורה מוצפנת.
4. בדיקה שהלקוח מקבל את ההודעה מוצפנת ומצליח לפענח אותה.

כעת נחזור על כל הבדיקות של שלב א כדי לוודא שהכל עובד גם בשההודעות מוצפנות.

**שלב ג**

בשלב זה נריץ את הלקוחות במחשבים שונים ברשת כאשר השרת רץ במחשב משלו. נבצע את הבדיקות הבאות.

1. בדיקה שהלקוח מזהה את קיומו שח השרת ברשת
2. בדיקה שהלקוח מזהה את כתובת ה-IP של השרת
3. בדיקה שנוצר חיבור בין הלקוח והשרת ונוצרים אובייקטי ה-TcpClient הן בשרת והן בלקוח.

כעת נחזור על כל הבדיקות של שלב א כדי לוודא שהכל עובד גם כהשלקוחות והשרת רצים במחשבים נפרדים.

לוח זמנים

ראשוני - בזמן החגים בספטמבר (סוכות) ללמוד #C, בפועל – ראש השנה ואחרי סוכות: למדתי #C.

ראשוני – בזמן חנוכה, בפועל – בזמן חנוכה: למדתי איך להשתמש בsockets, threads ואיך להשתמש בwinforms בישביל ליצור UI.

ראשוני – להתחיל בפסח, בפועל – בפסח: התחלתי את הפרויקט בהתמקדות על יצירת המשחק, כלומר המחלקות של המשחק ופעולות שניתן לעשות במשחק וגם התחלתי ליצור את UI.

ראשוני – אחרי פסח להמשיך את הפרויקט ולסיים עד יוני, בפועל – המשכתי את הפרויקט וסיימתי אותו בתחילת יוני: לכתוב את שאר הפרויקט:

1. יצירת המחלקות של התמונות של הקלפים שבהם אשתמש בפרויקט.

ניהול סיכונים ודרכי התמודדות איתם

תיאור תחום הידע

פירוט מעמיק של היכולות

פירוט היכולות בצד השרת

* **שם:** השרת יכול להתחבר ולנהל מספר לקוחות

**מהות:** השרת מאזין לשקע קבוע אליו שולחים כל הלקוחות בקשות חיבור.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין לשקע בעזרת אובייקט TcpListener. האובייקט מזהה לקוחות המנסים להתחבר לכתובת ה-IP ולשקע של השרת באמצעות פרוטוקול TCP.
* כאשר מזוהה לקוח הרוצה להתחבר נוצר אובייקט מסוג TcpClient אשר ינהל מעתה והלאה את התקשורת עם לקוח זה.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* TcpListener
* TcpClient

*אובייקטים של הפרויקט:*

* Client
* Server
* **שם:** הגנה על המידע שמועבר ברשת

**מהות:** המידע שמועבר בין השרת ללקוח ולהפך מוצפן ומאובטח בעזרת שילוב של הצפנה אסימטרית של RSA והצפנה סימטרית של AES.  
**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* יצירת מפתח פרטי ומפתח ציבורי להצפנת RSA
* החלפת מפתחות ציבוריים בין השרת ללקוח
* העברת מפתח הצפנת AES מהשרת לכל הלקוחות (לכל לקוח מפתח שונה) בצורה מוצפנת

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* RSACryptoServiceProvider
* AES

*אובייקטים של הפרויקט:*

* Encryption
* **שם:** יצירת משתמשים חדשים ושמירת הפרטים שלהם בצורה מאובטחת.

**מהות:** השרת מאזין להודעות של יצירת משתמש המגיעות מלקוחות ומאפשר לכל לקוח ליצור משתמש ושומר אצלו את פרטי ההתקשרות של הלקוח ופרטים נוספים. הפרטים הרגישים (הסיסמא) עוברים קידוד המונע את אפשרות גניבתם.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* קבלת הודעה מוצפנת מכל לקוח הרוצה להתחבר הכוללת את פרטי ההתחברות שלו.
* השרת מוודא ששם המשתמש לא בשימוש כבר.
* פיענוח הפרטים ויצירת קובץ json לשמירתם.
* קידוד הסיסמא לפני שמירתה באמצעות פונקצית Hash עם תוספת אבטחה בטכניקת salt ו-pepper.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* TcpClient
* JsonSerializer
* SHA256
* StreamReader
* StreamWriter

*אובייקטים של הפרויקט:*

* Server
* Jsn
* ClientHandler
* **שם:** כניסת לקוחות למערכת באמצעות שם משתמש וסיסמא

**מהות:** השרת מאפשר ללקוחות לבצע login ומוודא את נכונות פרטי ההתקשרות שלהם (שם משתמש וסיסמא).

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין לבקשות של כניסת משתמש המגיעות מהלקוחות המחוברים.
* השרת שולף את פרטי המשתמש הרשומים אצלו.
* השרת מאמת את נכונות הסיסמא ומאפשר כניסה למערכת.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* SHA256
* TcpClient
* JsonSerializer
* StreamReader
* StreamWriter

*אובייקטים של הפרויקט:*

* Jsn
* Server
* ClientHandler
* **שם:** יצירת זירת משחק חדשה ושמירת פרטיה.

**מהות:** השרת מאפשר ללקוחות ליצור זירות משחק חדשות ושומר את הפרטים שלהם.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין לבקשות של יצירת זירת משחק המגיעות מהלקוחות המחוברים.
* השרת מוודא שזירת המשחק לא קיימת כבר.
* השרת שומר את הפרטים בקובץ json כאשר קוד הכניסה נשמר בצורה מאובטחת.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* SHA256
* TcpClient
* StreamReader
* StreamWriter
* JsonSerializer

*אובייקטים של הפרויקט:*

* Jsn
* Server
* ClientHandler
* **שם:** הצטרפות לקוחות לזירה קיימת.

**מהות:** השרת מאפשר ללקוחות להצטרף לזירת משחק קיימת באמצעות שליחת שם הזירה וקוד הכניסה.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין לבקשות של הצטרפות לזירה המגיעות מהלקוחות המחוברים.
* השרת שולף את פרטי הזירה הרשומים אצלו.
* השרת מאמת את נכונות קוד הכניסה ומוודה שהזירה פעילה (מנהל הזירה פתח אותה) ואז מאפשר כניסה.
* אם מנהל הזירה שלח את הבקשה אז הזירה מופעלת

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* SHA256
* TcpClient
* JsonSerializer
* StreamReader
* StreamWriter

*אובייקטים של הפרויקט:*

* Jsn
* Server
* ClientHandler
* Lobby
* **שם:** קיום של מספר זירות משחק במקביל.

**מהות:** השרת מאפשר התקיימות של מספר זירות במקביל כך שכמה קבוצות משתמשים יכולות לנהל משחקים בנפרד זו מזו.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מחזיק מספר זירות פעילות במקביל.
* לכל זירה יש תהליכון משלה לניהול התקשורת שלה עם כל הלקוחות הנמצאים בזירה.
* בכל זירה יכול להתקיים משחק בלי תלות בזירות האחרות.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* TcpClient
* Thread

*אובייקטים של הפרויקט:*

* Lobby
* **שם:** התחלת משחק חדש.

**מהות:** השרת מאפשר למנהל הזירה להתחיל משחק חדש בהשתתפות כל מי שנמצא בזירה.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין לבקשה של מנהל הזירה להתחיל משחק חדש.
* מאותחל משחק חדש והשרת מחלק קלפים לכל השחקנים.
* השרת מודיע לשחקן שאמור לשחק ראשון.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient

*אובייקטים של הפרויקט:*

* Server
* Lobby
* Game
* Player
* Card
* MyReader
* MyWriter
* **שם:** תקשורת של שחקנים באותה זירה באמצעות צ'אט.

**מהות:** השחקנים בזירה יכולים לתקשר זה עם זה באמצעות הצ'אט הזירתי.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת מאזין להודעות צ'אט הנשלחות מהשחקנים הנמצאים בזירה.
* כל הודעה נשלחת לכל שאר השחקנים בזירה.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient

*אובייקטים של הפרויקט:*

* Server
* Lobby
* MyReader
* MyWriter
* **שם:** השרת מנהל את מהלך המשחק.

**מהות:** השרת מקדם את המשחק באמצעות העברת הודעות בין השחקנים, העברת קלפים, ושמירה על החוקים.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* מחלק את הקלפים בתחילת המשחק
* מודיע לכל שחקן מתי מגיע תורו
* מתווך בתקשורת בין שני השחקנים שצריכים להעביר ביניהם קלף
* מעדכן את שאר השחקנים על התקדמות המשחק
* מזהה כאשר המשחק מסתיים ומכריז על המפסיד
* כל הודעה נשלחת לכל שאר השחקנים בזירה.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient

*אובייקטים של הפרויקט:*

* Server
* Lobby
* MyReader
* MyWriter
* Game
* Player
* **שם:** השרת מטפל בצורה מסודרת במקרים של עזיבת שחקן בזמן המשחק.

**מהות:** השרת יודע לזהות התנתקות או עזיבה של אחד השחקנים ועוצר את המשחק בצורה מסודרת.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* השרת יודע לזהות התנתקות של אחד הלקוחות המחוברים.
* השרת מקשיב לבקשות עזיבה של השחקנים.
* השרת מפסיק את המשחק ומחזיר את כל השחקנים הנותרים ללובי.
* במקרה שמנהל הלובי עוזב הלובי נסגר.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient

*אובייקטים של הפרויקט:*

* Server
* Lobby
* MyReader
* MyWriter

פירוט היכולות בצד הלקוח

* **שם:** התחברות לשרת.

**מהות:** הלקוח יודע לזהות את כתובת השרת וליזום התחברות.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* הלקוח מחפש ברשת שרת המאזין לשקע של המערכת.
* כאשר נמצא שרת הלקוח יוזם התחברות.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* TcpListener
* TcpClient

*אובייקטים של הפרויקט:*

* Client
* Server
* **שם:** יצירת משתמש חדש אצל השרת.

**מהות:** קליטת הפרטים מהמשתמש ושליחתם לשרת.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* פתיחת חלון לקליטת הפרטים הנדרשים ליצירת משתמש חדש.
* שליחת הפרטים לשרת.
* קבלת אישור על רישום או דחייה במקרה שהמשתמש כבר קיים

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* RegisterForm
* **שם:** הלקוח מנהל תקשורת מאובטחת עם השרת.

**מהות:** הצפנה של כל ההודעות שנשלחות לשרת באמצעות מפתח AES.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* יצירת מפתח פרטי ומפתח ציבורי להצפנת RSA
* החלפת מפתחות ציבוריים בין השרת ללקוח.
* קבלת מפתח הצפנת AES מהשרת בצורה מוצפנת.
* שימוש במפתח AES להצפנת כל ההודעות הבאות.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* RSACryptoServiceProvider
* AES
* TcpClient

*אובייקטים של הפרויקט:*

* Client
* Encryption
* **שם:** יצירת זירת משחק חדשה אצל השרת.

**מהות:** קליטת הפרטים מהמשתמש ושליחתם לשרת.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* פתיחת חלון לקליטת הפרטים הנדרשים ליצירת זירה חדשה.
* שליחת הפרטים לשרת.
* קבלת אישור על רישום או דחייה במקרה שהזירה קיימת.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* CreateNewLobbyForm
* MyReader
* MyWriter
* **שם:** התחברות לזירת משחק קיימת.

**מהות:** המשתמש יכול להקליד את שם זירת המשחק אליה הוא רוצה להתחבר וקוד הכניסה והודעה תשלח לשרת לבצע את הפעולה.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* פתיחת חלון לקליטת פרטי ההתחברות לזירה.
* שליחת הפרטים לשרת.
* קבלת אישור או דחייה במקרה שהזירה לא קיימת, לא פעילה או שפרטי ההתחברות אינם נכונים.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* JoinLobbyForm
* MyReader
* MyWriter
* **שם:** התנתקות מהזירה בכל שלב.

**מהות:** המשתמש יכול לעזוב את המשחק ולהתנתק מהזירה בכל רגע שירצה.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* בחלון של הלקוח יש כפתור שמאפשר לבקש לעזוב את הזירה.
* שליחת הודעה על עזיבת הזירה לשרת.
* אפשרות להתחבר לזירה אחרת.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* MyWriter
* **שם:** מנהל זירה יכול להתחיל משחק חדש.

**מהות:** המשתמש שהוא מנהל הזירה יכול לשלוח הוראה לשרת להתחיל משחק חדש.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* כפתור במסך הלקוח לשליחת ההודעה לשרת.
* ניתן ללחוץ על הכפתור רק כאשר יש מספיק שחקנים בלובי.
* כל השחקנים מקבלים הודעה מהשרת שהמשחק התחיל.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* MyWriter
* **שם:** כל המשתמשים באותה הזירה יכולים לתקשר אחד עם השני באמצעות צ'אט.

**מהות:** המשתמש יכול לשלוח הודעות אשר יתקבלו על ידי כל שאר המשתמשים בזירה ולראות את כל ההודעות ששלחו המשתמשים האחרים.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* תיבת טקסט בה יכול הלקוח לכתוב את ההודעות שלו.
* תיבת טקסט לקריאה בלבד בה יופיעו כל ההודעות שנשלחו בזירה.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

* Client
* MyReader
* MyWriter
* **שם:** ביצוע פעולות המשחק.

**מהות:** בזמן המשחק המשתמש יכול לבצע את כל הפעולות הנדרשות בזמן התור שלו.

**אוסף היכולות/הפעולות הנדרשות למימוש היכולת**:

* בתורו המשתמש יכול לבחור קלף ולהציע אותו לבא בתור.
* המשתמש יכול לקבל או לסרב לקלף שהוצע לו.

**אובייקטים נחוצים:**

*אובייקטים של השפה:*

* StreamReader
* StreamWriter
* TcpClient
* Form

*אובייקטים של הפרויקט:*

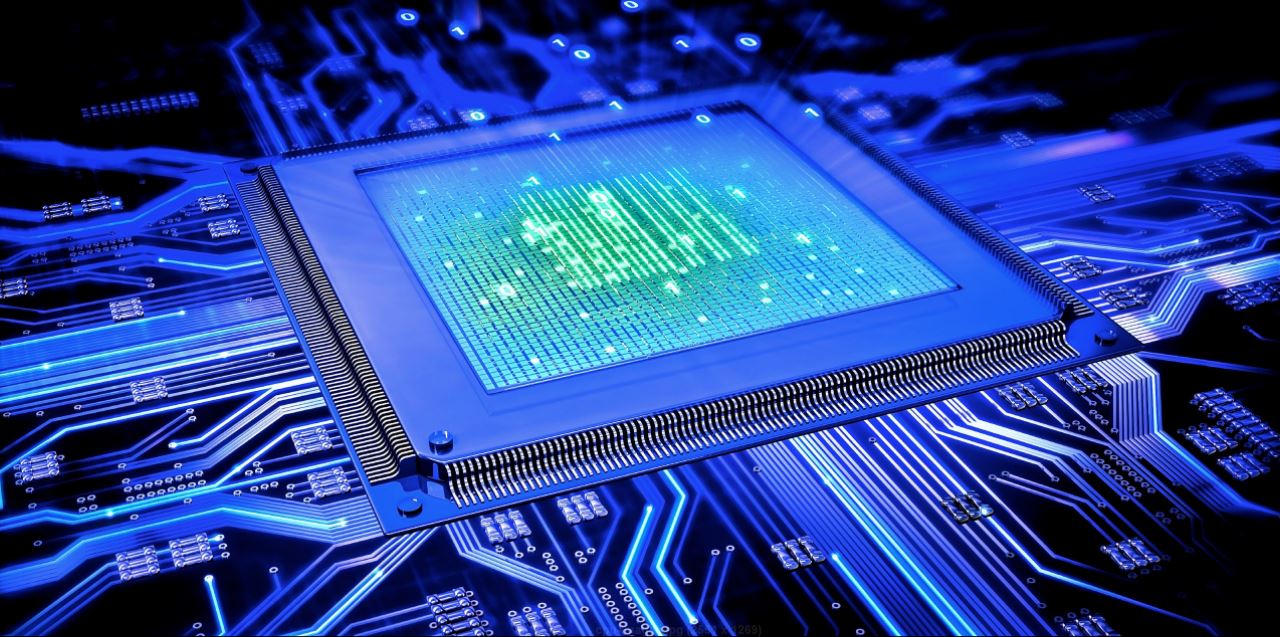
* Client
* MyWriter
* MyReader

**מבנה הפרויקט**

תיאור החומרה

ראשית נתחיל עם אבני הבניין של המחשב שלי:

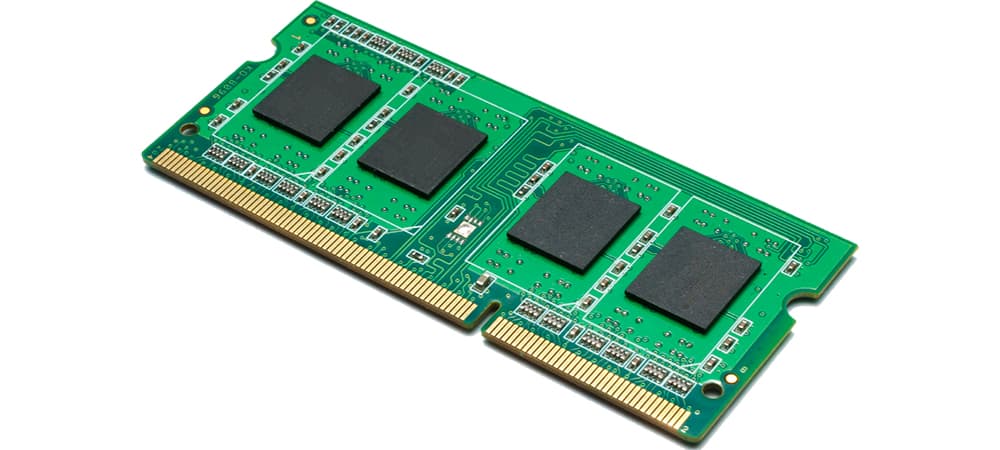
1. הרכיב הראשון הוא מעבד המחשב (יחידת עיבוד מרכזית) ,CPU (Central Processing Unit), הנקרא גם "מוח המחשב". כמו המוח שמתקשר ואחראי על האיברים השונים בגוף, המעבד מתקשר עם רכיבים שונים במחשב כמו הזיכרון והאחסון והוא אחראי על עיבוד הנתונים והתהליכים שקורים במחשב. הוא מבצע את הפקודות השמורות בזיכרון של המחשב, בלעדיו המחשב קופא אם הוא לא תקין. כל פקודה שהוא עושה בנויה מרצף קצר של ביטים, בשם אחר לביטים- שפת מכונה. סוג המעבד במחשב שלי מופיע בתמונה של פרטי המחשב שלי.



1. הרכיב השני הוא זיכרון ה-RAM (Random Access Memory)

זיכרון ה-RAM הוא הזיכרון לטווח הקצר של המחשב בעוד שהדיסק הקשיח (HARD DISK) של המחשב הוא הזיכרון לטווח ארוך. משתמשים ב-RAM כדי לזרז תהליכים במחשב מכיוון שהגישה לדיסק הקשיח לוקחת הרבה זמן יחסית לעומת הגישה ל-RAM שהיא מהירה. ה-RAM מאחסן את כל מה שפועל במחשב ברגע זה כדי שהשליפה של מה שצריך תהיה יותר מהירה .

ככל שהזיכרון של הRAM- יהיה גדול יותר, ככה הפעלת המחשב, הנתונים והאפליקציות שפועלות בו ירוצו מהר יותר. בגלל שה-RAM הוא זיכרון זמני כאשר מכבים את המחשב כל מה שנשמר בו נמחק.



1. הרכיב האחרון הוא המעבד הגרפי,GPU (Graphics Processing Unit), הנקרא גם כרטיס המסך. רכיב זה אחראי על העיבוד הגרפי של המחשב ועל הצגת תמונות על המסך. ככל שהכרטיס הגרפי של המחשב טוב יותר, כך ניתן לייצר תמונה חלקה יותר .



הנושא המרכזי של הפרויקט שלי הוא תקשורת בין מחשבים, בחלק הזה אסביר בפירוט כיצד תקשורת בין מחשבים עובדת ועם מה היא משתמשת. בהמשך אתאר את הרכיבים המאפשרים תקשורת.

תקשורת בין מחשבים מאפשרת העברת מידע דיגיטלי בין שני מכשירים או יותר. תהליך זה, המהווה את הבסיס לאינטרנט, לרשתות מקומיות (LAN) ולרשתות אזוריות רחבות (WAN), נשען על שילוב של רכיבים פיזיים ולוגיים הפועלים יחדיו .

ברמת האפליקציה, תקשורת בין מחשבים מתבצעת באמצעות תוכנות המשתמשות בפרוטוקולים סטנדרטיים, כגון HTTP, FTP ו-TCP/IP. פרוטוקולים אלו מגדירים כללי תקשורת סטנדרטיים המאפשרים לתוכנות "לדבר" זו עם זו באמצעות פורמטים מוגדרים מראש, תוך הבטחת חילוף מידע יעיל ומוקפד .

בתהליך התקשורת משתתפים מספר רכיבים:

* **פורט (Port)** לכל תוכנה המבקשת לתקשר עם מחשב אחר מוקצה מספר פורט ייחודי. פורט זה משמש כמזהה ייחודי עבור התוכנה ברשת, ומאפשר ניתוב מדויק של המידע אל היעד המיועד. ניתן לדמות פורטים כשערים ייחודיים במחשב, המאפשרים לכל תוכנה "להאזין" לפניות המגיעות דרך פורט ספציפי .
* **כרטיס רשת** (**NIC**): ממשק חומרה המאפשר למחשב להתחבר לרשת. כרטיס הרשת אחראי על המרת הנתונים מפורמט דיגיטלי לפורמט פיזי (כגון אותות חשמליים או אור) שניתן לשדר דרך כבלי רשת או באופן אלחוטי.



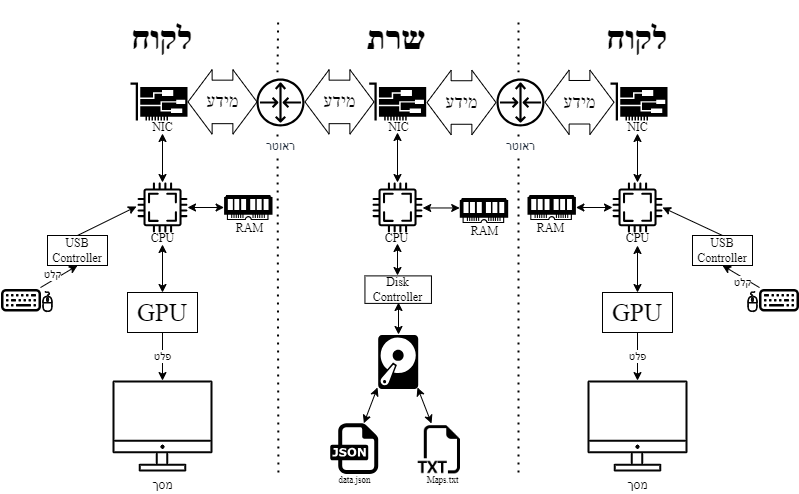
כרטיסי רשת זמינים במגוון סוגים, ביניהם:

* כרטיסי רשת **Ethernet**: מאפשרים חיבור באמצעות כבלי רשת ומספקים פס רחב גבוה ויציבות רבה, במהירויות של עד 100 Gbps.
* כרטיסי רשת **Wi-Fi**: מאפשרים חיבור אלחוטי באמצעות גלי רדיו, ומציעים נוחות וניידות, אך טווח הקליטה שלהם מוגבל ויכול להיות מושפע ממכשולים פיזיים בתווך בין המכשירים המתקשרים.
* כרטיסי רשת אופטיים: פתרון מתקדם המאפשר העברת נתונים במהירות עצומה, עד 100 Gbps ואף יותר, באמצעות סיבים אופטיים .
* **כתובת MAC**: מזהה ייחודי המוקצה לכל כרטיס רשת ברשת. כתובת MAC מאפשרת זיהוי חד- משמעי של כל כרטיס רשת ברשת, וחיונית להעברת נתונים נכונה בין מחשבים.
* **מתגים** (**switches**) **ונתבים** **(routers)**: רכיבים אקטיביים ברשת המשמשים לניתוב נתונים בין מחשבים שונים. מתגים מחברים מספר מחשבים יחד באותה רשת מקומית (LAN), ומאפשרים להם לשתף ביניהם משאבים (קבצים, מדפסות, מידע). נתבים מחברים רשתות מקומיות שונות זו לזו, ומאפשרים תקשורת בין מחשבים הנמצאים ברשתות נפרדות.

**תהליך התקשורת:**

* **שליחת נתונים:** תוכנת האפליקציה שולחת את המידע המיועד להעברה דרך פורט ייעודי .
* **כרטיס רשת:** כרטיס הרשת מקבל את המידע מהתוכנה ומתרגם אותו לפורמט פיזי מתאים לשידור .
* **שידור נתונים:** כרטיס הרשת משדר את המידע דרך כבלי רשת או אלחוטית למחשב היעד .
* **מתגים (switches) ונתבים (router**s**):** נתונים אלו עוברים דרך מתגים ונתבים ברשת, בהתאם ליעדם הספציפי. המתגים מבצעים החלפת נתונים פיזית בין מחשבים באותה רשת מקומית, בעוד שהנתבים מכוונים את התעבורה בין רשתות מקומיות שונות .
* **קבלת נתונים:** כרטיס הרשת של המחשב המקבל קולט את המידע הפיזי ומתרגם אותו בחזרה לפורמט דיגיטלי .
* **פענוח נתונים:** תוכנת האפליקציה במחשב היעד מקבלת את המידע הדיגיטלי ומפרשת אותו בהתאם לייעודו .

להלן תרשים המתאר כיצד כל הרכיבים שתוארו לעיל מתקשרים ביניהם:



הנושא האחרון שארצה לתאר קשור ליכולות החומרה שמאפשרות את מימוש הפרויקט שלי .כדי שהמחשב יוכל לתקשר הוא צריך לבצע מספר משימות בו זמנית. כדי לעשות זאת הוא משתמש בטכנולוגיה , ריבוי תהליכונים (Multi-Threading). במקום לבצע משימות אחת אחרי השנייה, תהליכונים מאפשרים חלוקת משאבי המחשב )מעבד, זיכרון( בין מספר משימות המתבצעות במקביל, מה שמגביר את יעילותו ומהירות התגובה שלו.

בשביל לנהל תהליכונים יש מספר רכיבים חיוניים:

מעבד שהוא כמו ה"מוח" המחשב, אחראי על ביצוע חישובים והפעלת תוכנות. מעבדים מודרניים רבים כוללים מספר ליבות (Cores), המאפשרות לכל ליבה לבצע תהליכון נפרד במקביל .

A blue and white square with white squares

AI-generated content may be incorrect.

זיכרון מאחסן את נתונים והוראות עבור התוכנות. כל ה תהליכונים משתפים זיכרון משותף, אך לכל תהליכון יש גם מרחב זיכרון פרטי משלו .

מנהל תהליכונים היא תוכנת מערכת שמנהלת את התהליכונים השונים, וקובעת באיזו תדירות כל תהליכון יקבל זמן שימוש במעבד .

**תהליך ניהול** תהליכונים:

**יצירת התהליכון**: תוכנה יכולה ליצור תהליכונים חדשים באמצעות פונקציות מיוחדות במערכת ההפעלה .

הקצאת משאבים: מנהל התהליכונים מקצה לכל תהליכון משאבי מעבד וזיכרון בהתאם לצרכיו .

**החלפה בין תהליכונים**: מנהל התהליכונים מחליף בין תהליכונים שונים בצורה יעילה, תוך התחשבות בגורמים כמו עומס המעבד, זמינות הזיכרון וצרכים עדיפים .

**סנכרון תהליכונים**: המשתפים נתונים משותפים זקוקים למנגנוני סנכרון כדי למנוע התנגשויות וגישה לא תקינה לנתונים.

יתרונות השימוש בתהליכונים:

**ביצועים משופרים:** תהליכונים מאפשרים ניצול יעיל יותר של משאבי המחשב, וכתוצאה מכך ביצועים מהירים יותר במיוחד ביישומים מרובי משימות .

**סינכרוניזציה:** תהליכונים מאפשרים ניהול יעיל של משימות המתבצעות בו-זמנית תוך מניעת התנגשויות .

**ממשקיות משתמש רספונסיבית:** תהליכונים מאפשרים ליישומים להישאר רספונסיביים גם כאשר הם מבצעים משימות מורכבות, כיוון שתמיד יש תהליכון זמין לטיפול באירועים ממשק המשתמש.

תיאור הטכנולוגיה

שפת התיכנות שהשתמשתי בה היא #C. #C היא שפת תכנות מודרנית ורבת-עצמה שפותחה על ידי מיקרוסופט בתחילת שנות ה-2000. השפה עוצבה על ידי אנדרס היילסברג והושקה לראשונה ב-2000 כחלק מפלטפורמת .NET. C#/ היא שפה מונחית עצמים עם הקלדה חזקה. השפה משלבת את הפשטות של Visual Basic עם העוצמה של C++, ומתעדכנת באופן קבוע עם תכונות חדשות ומתקדמות. השפה תומכת בניהול זיכרון אוטומטי באמצעות Garbage Collection, מה שמקל על המתכנתים ומפחית שגיאות קשורות לניהול זיכרון. היא כוללת מערכת ניהול חריגים (Exceptions) מפותחת וכלים לבדיקת קוד בזמן קומפילציה.

בישביל ה-UI (User Interface) התשמשתי בWinForms. WinForms היא תשתית לבניית ממשק משתמש ב-C#/.NET לבניית יישומי שולחן עבודה הפועלים על Windows.

מערכת ההפעלה שהשתמשתי בה היא Windows 10. Windows 10 היא מערכת הפעלה שפיתחה מיקרוסופט והושקה ביולי 2015 כחלק מסדרתWindows NT . היא הגיעה כיורשת של Windows 8.1 ונועדה לאחד את החוויה בין מחשבים אישיים, טאבלטים וסמארטפונים. מיקרוסופט תפסיק לתמוך בה ב-15 באוקטובר 2025 לאחר תאריך זה מי שלא שדרג ל11 windows עדיין יוכל להשתמש בה אך המחשב יהיה יותר חשוף לסיכוני אבטחה ולווירוסים.

התקשורת בין הלקוח לשרת ממושת באמצעות שקעים בפרוטוקול TCP שבשכבת התעבורה.שקע הוא נקודת הקצה של החיבור בין שני הרכיבים המתקשרים, ממנו שולחים מידע לשקע אחר באותו מחשב או במחשב אחר ברשת ומקבלים מידע משקע אחר. פרוטוקול TCP הוא פרוטוקול תקשורת בשכבה 4 של מודל 7 השכבות (OSI). הפרוטוקול מאפשר תקשורת בין שקעים, כאשר לכל יש כתובת IP ומספר פורט. משמעות פרוטוקול TCP הוא שהנתונים בתקשורת נשלחים בצורה אמינה והוא מוודא שכל המידע נשלח בסדר הנכון, אחרת המידע ישלח שוב, בניגוד לפרוטוקול UDP בו המידע נשלח רק פעם אחת אפילו אם היו שגיעות. פרוטוקול TCP הוא איטי יותר אך מדויק יותר. משתמשים ב-TCP לשימושים כגון צ'אטים, בהם צריך שכל המידע יגיע ובסדר הנכון, ומשתמשים ב-UDP לשימושים כמו שיחות טלפון ווידיאו, בהן לא חייבים שכל המידע יגיע ואפשר שיהיו קיטועים במידע. בנוסף, עד שהמידע שלא הגיע בפרוטוקול UDP ישלח שוב יעבור זמן ולא ניתן לשלב אותו בשיחה או בקטע הוידיאו שהתקדמו הלאה.

טכניקת ההצפנה שהשתמשתי בה היא הצפנה היברידית המשלבת שיטת הצפנה אסימטרית לצורך החלפת מפתחות ולאחריה שימוש בהצפנה סימטרית לצורך החלפת הודעות מאובטחת. ההצפנה האסימטרית נעשית באמצעות אלגוריתם הצפנה RSA. בהצפנה אסימטרית מיוצרים שני מפתחות: מפתח פרטי המשמש את בעליו כדי לפענח הודעות מוצפנות המתקבלות אצלו ומפתח ציבורי הנשלח לצד השני ומשמש להצפנת ההודעות הנשלוח אליו. בגלל איטיותה של ההצפנה האסימטרית היא אינה מתאימה למשלוח כמויות גדולות של מידע ולכן נעדיף להשתמש בהצפנה סימטרית לאבטחת התקשורת, אך עדיין נצטרך להשתמש בההצפנה האסימטרית כדי לבצע את השליחה הראשונית של מפתח ההצפנה הסימטרי. נשתמש באלגוריתם ההצפנה הסימטרי AES שהוא מהיר ויעיל בהצפנה ופיענוח של קטעי מידע גדולים.

סיסמאות המשתמשים נשמרות בצורה מאובטחת על ידי שימוש בטכניקת גיבוב הממירה את הסיסמא למחרוזת באורך קבוע אשר ממנה אי אפשר לשחזר את הסיסמא המקורית (פונקצית הגיבוב אינה הפיכה). כדי ליצור גיבוב חסין יותר נשתמש בטכניקה הנקראת Salt. עבור כל משתמש מוגרלת מחרוזת קצרה אשר מצורפת לסיסמא שלו לפני שמופעלת עליה פונקצית הגיבוב. מחרוזת זו נשמרת יחד עם הסיסמא ותשמש בהמשך כדי לוודא את נכונות הסיסמא של המשתמש המנסה להתחבר. טכניקה זו מונעת התנגשות בין שני משתמשים אשר בוחרים באותה סיסמא ומגנה על משתמשים שבוחרים סיסמא קלה לניחוש. בנוסף נשתמש גם בטכניקת Pepper לפיה מוסיפים לסיסמא בנוסף ל-Salt מחרוזת נוספת, זהה לכל המשתמשים, הנשמרת בקוד האפליקציה, בניגוד למחרוזות ה-Salt הנשמרות בקובץ פרטי המשתמשים. ה-Pepper מספק הגנה נוספת למקרה בו נגנב הקובץ עם פרטי המשתמשים.

תיאור זרימת המידע במערכת

A screenshot of a cell phone

AI-generated content may be incorrect.





תיאור האלגוריתמים המרכזיים בפרויקט

אלגוריתם של השרת לטיפול בהתנתקות של לקוחות

* **בעיה:**

כיצד למנוע קריסה של השרת לאפשר המשך פעילות תקינה של זירת המשחק כאשר אחד מהמשתמשים מתנתק באופן בלתי צפוי?

* **תיאור אלגוריתמים קיימים לפתרון הבעיה:**
* **הפנייה למקורות רלוונטיים:**

הפתרון שבחרתי להשתמש בו לא נלקח משום מקור ומותאם ספציפית לפרויקט הזה.

* **סקירת הפיתרון הנבחר:**

אלגוריתם של השרת לחלוקה הוגנת של הקלפים בתחילת המשחק

* **בעיה:**

כיצד לחלק את קלפי המשחק באופן אקראי בין השחקנים בתחילת המשחק?

* **תיאור אלגוריתמים קיימים לפתרון הבעיה:**

בהתחלה ניסיתי לצרף את קבוצות הקלפים לרשימה בזו אחר זו ולאחר מכן לחלק את הקלפים בין השחקנים לפי הסדר שלהם כאשר כל שחקן מקבל כל פעם קלף אחד עד שמחולקים כל הקלפים. השחקן הראשון תמיד יקבל את קלף הג'וקר. ניתן לשכלל את הפיתרון הזה על ידי סידור אקראי של השחקנים לפני התחלת החלוקה.

פיתרון אחר שניתן לעשות הוא דבר ראשון לסדר את הקלפים בסדר אקראי באמצעות יצירת סידרה אקראית של מספרים באורך של מספר הקלפים ואז סידור הקלפים באותו סדר אשר בו סידרת המספרים האקריים תיהיה ממוינת מהקטן לגדול. זו היא דרך ליצר פרמוטציה אקראית של הקלפים.

* **הפנייה למקורות רלוונטיים:**

הפתרון שבחרתי להשתמש בו לא נלקח משום מקור ומותאם ספציפית לפרויקט הזה.

* **סקירת הפיתרון הנבחר:**

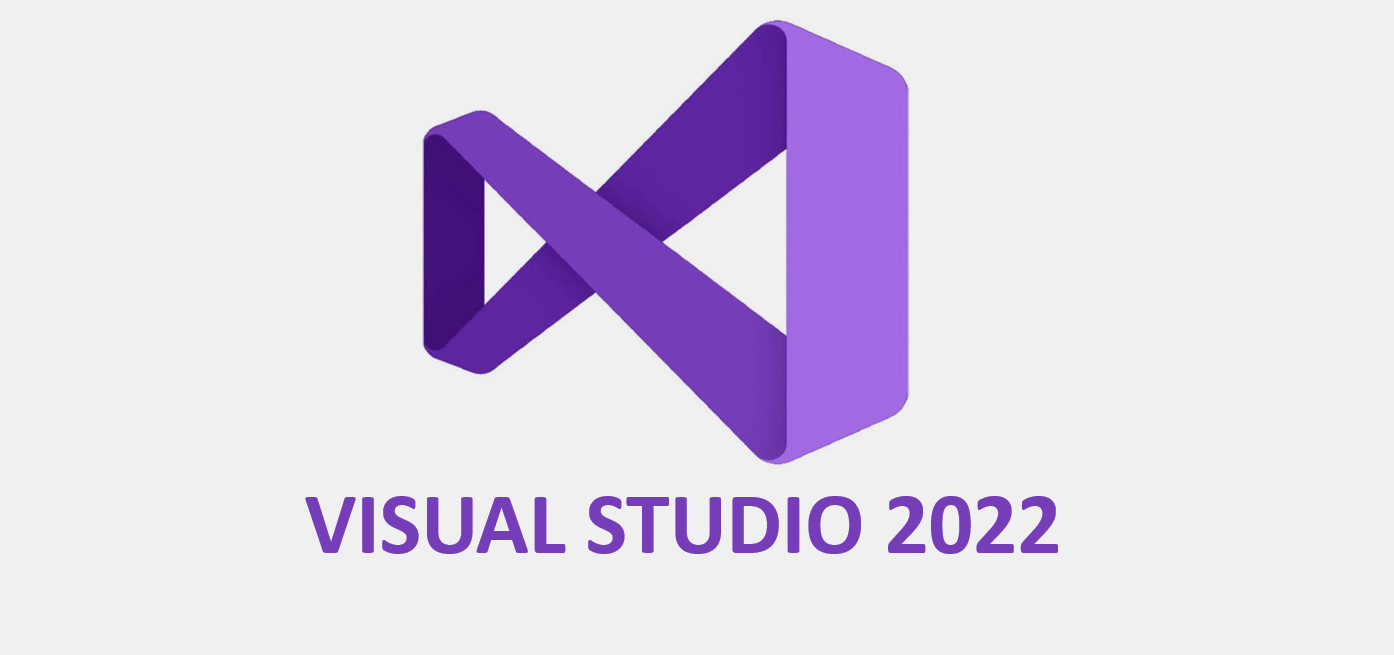
הפיתרון הראשון נפסל מכיוון שבכל משחק השחקנים יקבלו בדיוק את אותם הקלפים ודבר זה פוגע בהתנהלות המשחק ובהוגנות שלו.

הפיתרון השני יתן קלפים שונים לשחקנים בכל משחק אך קבוצות הקלפים עצמן לא ישתנו בין המשחקים.

הפיתרון השלישי הוא זה שבחרתי כי הוא מספק אקראיות מלאה של חלוקת הקלפים, כלומר בכל משחק כל שחקן יקבל קבוצת קלפים שונה מזו שקיבל במשחק הקודם ואף שחקן לא יוכל לקבל את אותה קבוצת קלפים שהייתה אצל שחקן אחר במשחק הקודם.

לצורך יצירת הסידור האקראי של הקלפים אני יוצר רשימה של כל הקלפים של המשחק ולאחר מכן משתמש בפונקציית OrderBy אשר משתמשת ביוצר של מספרים אקראיים כדי לקבוע את סדר המיון של קלפי המשחק שברשימה.

תיאור סביבת הפיתוח

כלי הפיתוח העיקרי שלי הוא Visual Studio 2022 (VS) של מיקרוסופט. VS היא סביבת פיתוח משולבת (IDE) המאפשרת פיתוח של מגוון רחב של תוכנות כגון אפליקציות דסקטופ, אפלקציות ווב, אפליקציות למכשירים ניידים, סיפריות, API ועוד. VS מאפשר פיתוח בשפות ++C, #C, Visual Basic, Python, JavaScript ועוד. המאפיינים העיקריים של VS ששימשו אותי ביצירת הפרויקט הם:

* כלי לעריכת קוד – עורך חכם שמכיל השלמה אוטומתית של שמות משתנים ופקודות והדגשת מילים בצורה שונה בהתאם לתפקידן בתחביר השפה.
* כלי לניפוי באגים (Debugger) – אפשרות להריץ את הקוד פקודה אחר פקודה, לשים נקודות עצירה בקוד ולראות את ערכי המשתנים בכל שלב בהרצה.
* מעצב ממשק משתמש ויזואלי – מאפשר לעצב את רכיבי ממשק המשתמש השונים באמצעות גרירה ושיחרור (drag and drop).
* כלי בנייה והידור (compilation) – VS כולל מהדרים (compilers) לכל השפות בהן הוא תומך ומאפשר לבנות אפליקציות הניתנות להרצה (executables) וספריות תוכנה (dynamic link library).
* ניהול פרויקטים מורכבים – VS מאפשר פיתוח של מספר פרויקטים ביחד על ידי יצירת Solution המכיל את כולם.

פיתוח הפרויקט התבצע בשפת #C גירסה 8.0 וממשק המשתמש מבוסס על טכנולוגית WinForms.

לצורך בדיקה וניפוי באגים השתמשתי בכלים הבאים:

* כלי ניפוי הבאגים של VS באמצעותו יכולתי להיכנס לחלקים בקוד בהם זיהיתי התנהגות השונה ממה שהתכוונתי, לקדם את הקוד פקודה אחרי פקודה, לעקוב אחרי ערכי המשתנים ולזהות את הבאג הגורם להתנהגות הלא רצויה. הפעלתי את כלי ניפוי הבאגים גם על קוד השרת וגם על קוד הלקוח (אך לא באותו הזמן).
* שילבתי בתוך הקוד, הן של השרת והן של הלקוח, רישום של הודעות לקובץ משותף. בכל שלב בביצוע הקוד התוכנה שומרת הודעות בקובץ הרישום המתעדות את מצבה באותו הרגע ואת הפעולות האחרונות שביצעה. בשלב הבדיקות, גם השרת וגם הלקוחות רושמים הודעות באותו הקובץ וכך ניתן לראות את התקדמות הריצה שלהם כרצף אירועים על אותו ציר זמן וכך ניתן לזהות בעיות הקשורות לסינכרון ובעיות תקשורת. הכתיבה המשותפת לקובץ אחד ממספר אפליקציות שונות הרצות במקביל מנוהלת על ידי מנגנון Mutex גלובלי הנגיש לכל האפליקציות הרצות על המחשב. ניתן לכבות את מנגנון הרישום כאשר מסיימים את הבדיקות כך שלא יאט את ריצת התוכנה אחרי שיחרורה ומסירתה ללקוחות.

תיאור פרוטוקול התקשורת

התקשורת מתבצעת על ידי שליחה של מחרוזות מוצפנות בין הלקוחות והשרת. כל הודעה בנויה משני חלקים המופרדים ביניהם על ידי התו “|” :

* סוג ההודעה – מחרוזת קצרה המקודדת את סוג ההודעה הנשלחת.
* גוף ההודעה – כולל מספר מחרוזות בהתאם לסוג ההודעה המופרדות על ידי “;” או “:” . גוף ההודעה יכול להיות גם המחרוזת הריקה.

מחרוזת ההודעה מוצפנת לפני שליחתה באמצעות מפתחה הצפנה הסימטרי מסוג AES הנמצא גם בידי השרת וגם בידי הלקוח. המחרוזת המוצפנת מצורפת להודעה מסוג 'הודעה מוצפנת' ומהווה את הגוף של אותה ההודעה. כך יודע הצד המקבל שעליו לפענח את גוף ההודעה שהתקבלה לפני שהוא מנסה לנתח אותה.

הודעות מהשרת ללקוח

* **הרשמה**

מטרת ההודעה: השרת מידע את הלקוח אם בקשת ההרשמה שלו הצליחה או נכשלה.

סוג ההודעה: Register

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + המחרוזת “Success” המציינת שבקשת ההרשמה הצליחה או המחרוזת “Failure” המציינת שבקשת ההרשמה נכשלה.
  + במקרה של הצלחה המחרוזת השנייה היא ריקה ובמקרה של כישלון המחרזות השנייה מכילה את סיבת הכישלון.
* **התחברות**

מטרת ההודעה: השרת מידע את הלקוח אם בקשת ההתחברות שלו הצליחה או נכשלה.

סוג ההודעה: Login

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + המחרוזת “Success” המציינת שבקשת ההתחברות הצליחה או המחרוזת “Failure” המציינת שבקשת ההתחברות נכשלה.
  + במקרה של הצלחה המחרוזת השנייה היא ריקה ובמקרה של כישלון המחרזות השנייה מכילה את סיבת הכישלון.
* **יצירת זירת משחק חדשה**

מטרת ההודעה: השרת מידע את הלקוח אם בקשת יצירת זירת משחק חדשה שלו הצליחה או נכשלה.

סוג ההודעה: CreateLobby

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + המחרוזת “Success” המציינת שבקשת יצירת זירת משחק חדשה הצליחה או המחרוזת “Failure” המציינת שבקשת יצירת זירת משחק חדשה נכשלה.
  + במקרה של הצלחה המחרוזת השנייה היא ריקה ובמקרה של כישלון המחרזות השנייה מכילה את סיבת הכישלון.
* **הצטרפות לזירת משחק**

מטרת ההודעה: השרת מידע את הלקוח אם בקשת ההצטרפות שלו לזירת משחק הצליחה או נכשלה.

סוג ההודעה: JoinLobby

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + המחרוזת “Success” המציינת שבקשת ההצטרפות הצליחה או המחרוזת “Failure” המציינת שבקשת ההצטרפות נכשלה.
  + במקרה של הצלחה המחרוזת השנייה היא ריקה ובמקרה של כישלון המחרזות השנייה מכילה את סיבת הכישלון.
* **המפתח הסימטרי**

מטרת ההודעה: השרת שולח ללקוח מפתח הצפנה סימטרי מסוג AES.

סוג ההודעה: AesKey

גוף ההודעה: מחרוזות אחת

* + המחרוזת מיכלה את מפתח ההצפנה הסימטרי מסוג AES.
* **אפשר להתחיל משחק**

מטרת ההודעה: השרת מידע את הלקוח שאפשר להתחיל את המשחק.

סוג ההודעה: CanStartGame

גוף ההודעה: ההודעה ריקה

* **חלוקת הקלפים**

מטרת ההודעה: השרת שולח לכל לקוח במשחק את הקלפים שלו.

סוג ההודעה: DealCards

גוף ההודעה: מספר מחרוזות (ארבע או חמש) המופרדות על יד התו “;”.

* + כל מחרוזת היא שם של קלף.
* **הצע קלף**

מטרת ההודעה: השרת מידע את הלקוח שעליו להציע קלף לשחקן שאחריו בתור.

סוג ההודעה: MakeOffer

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “;”.

* + מספר ההצעה (מספרים שלמים בין אחד לשלוש)
  + שם השחקן אליו מופנת ההצעה.
* **הוצע לך קלף**

מטרת ההודעה: השרת מידע את הלקוח שהציעו לו קלף שהוא לא חייב לקחת.

סוג ההודעה: AcceptCard

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + שם הקלף (הלקוח לא רואה את הקלף).
  + מי הציע לו את הקלף.
* **עומדים להציע לך קלף**

מטרת ההודעה: השרת מידע את הלקוח שעומדים להציע לו קלף ומי זה שעושה זאת.

סוג ההודעה: Receive Offer

גוף ההודעה: מחרוזת אחת.

* + שם השחקן שעומד להציע לו קלף.
* **קח את הקלף**

מטרת ההודעה: השרת מידע את הלקוח שהוא חייב לקחת את הקלף הבא.

סוג ההודעה: TakeCard

גוף ההודעה: מחרוזת אחת.

* + שם הקלף שהוא חייב לקחת.
* **לא תורך**

מטרת ההודעה: השרת מידע את הלקוח שלא תורו ושהוא אינו יכול לבצע אף פעולה.

סוג ההודעה: NotYourTurn

גוף ההודעה: ההודעה ריקה.

* **קיבלת צ'אט**

מטרת ההודעה: השרת מידע את הלקוח שנשלחה אליו הודעה דרך הצ'אט.

סוג ההודעה: ReceiveChat

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “;”.

* + שם השחקן ששלח את ההודעה.
  + תוכן ההודעה.
* **פרוטוקול המשחק**

מטרת ההודעה: השרת מידע את כל השחקנים בפעולות המתרחשות במשחק.

סוג ההודעה: GameLog

גוף ההודעה: מחרוזת אחת.

* + תוכן ההודעה.

הודעות מהלקוח לשרת

* **בקשת התחברות**

מטרת ההודעה: הלקוח מבקש להתחבר לשרת.

סוג ההודעה: LoginRequest

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + שם משתמש.
  + סיסמא.
* **בקשת הרשמה**

מטרת ההודעה: הלקוח מבקש להרשם לשרת.

סוג ההודעה: RegisterRequest

גוף ההודעה: שלוש מחרוזות המופרדות על יד התו “:”.

* + שם משתמש.
  + סיסמא.
  + אימייל
* **בקשה ליצור זירת משחק**

מטרת ההודעה: הלקוח מבקש מהשרת ליצור זירת משחק.

סוג ההודעה: CreateLobbyRequest

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + שם זירת המשחק.
  + סיסמאת זירת המשחק.
* **בקשת הצטרפות לזירת משחק**

מטרת ההודעה: הלקוח מבקש להתחבר לזירת משחק מסויימת.

סוג ההודעה: JoinLobbyRequest

גוף ההודעה: שתי מחרוזות המופרדות על יד התו “:”.

* + שם זירת המשחק.
  + סיסמאת זירת המשחק.
* **התחל משחק**

מטרת ההודעה: הלקוח מבקש מהשרת להתחיל את המשחק.

סוג ההודעה: StartGame

גוף ההודעה: ההודעה ריקה.

* **העברת קלף**

מטרת ההודעה: להודיע לשרת איזה קלף המשתמש בחר לתת.

סוג ההודעה: OfferCard

גוף ההודעה: מחרוזת אחת.

* + שם הקלף.
* **העברת הודעה בצ'אט**

מטרת ההודעה: להעביר הודעה לשרת ושהוא יעביר אותה לשאר השחקנים.

סוג ההודעה: BroadCastChat

גוף ההודעה: מחרוזת אחת.

* + תוכן ההודעה.

הודעות העוברות בשני הכיוונים

* **תשובה להצעת קלף**

מטרת ההודעה: להעביר הודעה שהצעת הקלף התקבלה. הודעה זו עוברת ממקבל ההצעה לשרת ומהשרת לנותן ההצעה.

סוג ההודעה: ResponseToOffer

גוף ההודעה: מחרוזת אחת.

* + Accept או Reject.

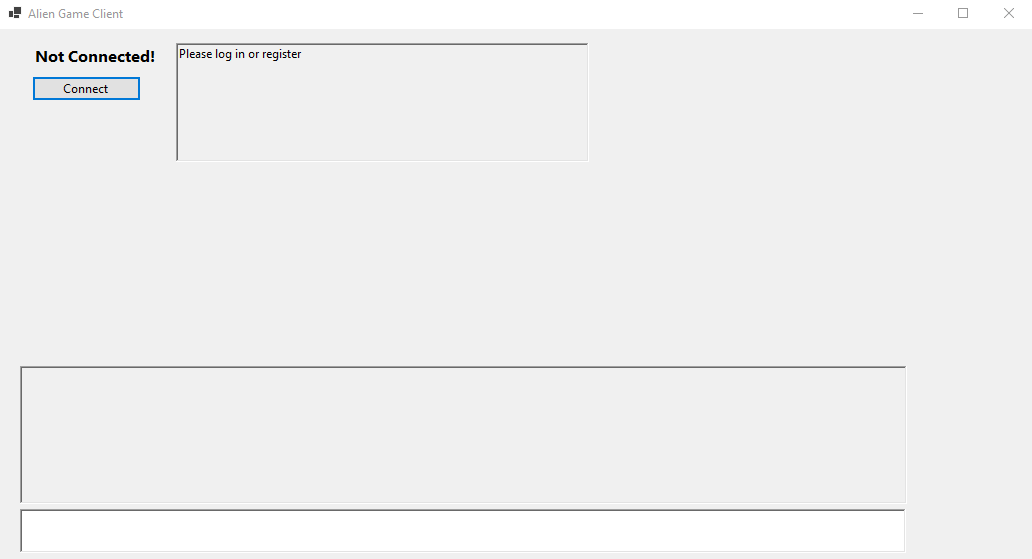
תיאור מסכי המערכת

**מסכים בצד הלקוח:**

מסכי טרום משחק:

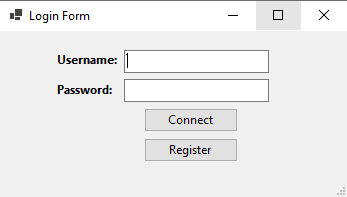
**שם**: מסך טרום התחברות

**תפקיד**: החלון הראשון שפוגש הלקוח כאשר הוא מריץ את האפליקציה. בחלון יש כפתור "Connect" באמצעותו יכול הלקוח להתחיל את תהליך ההתחברות.



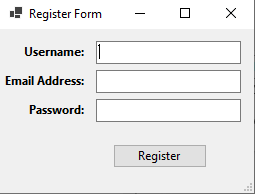
**שם**: מסך פרטי התחברות

**תפקיד**: הלקוח מכניס את שם המשתמש שלו והסיסמא ויכול להתחבר על ידי לחיצה על כפתור "Connect". בנוסף הוא יכול לעבור למסך רישום משתמש חדש על יד לחיצה על כפתור "Register".



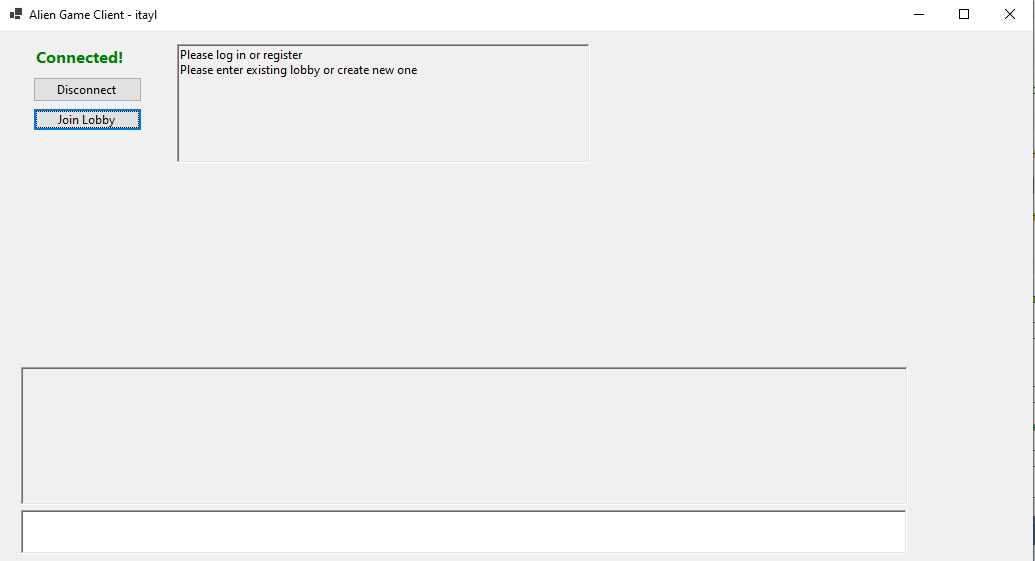
**שם**: מסך רישום משתמש חדש

**תפקיד**: המסך משמש להכנסת פרטים של משתמש חדש. לסיום ניתן ללחוץ על כפתור "Register" כדי לשלוח את פרטי הרישום לשרת.



**שם**: מסך הצטרפות לזירת משחק

**תפקיד**: מאפשר למשתמש להצטרף לזירת משחק על ידי לחיצה על כפתור "Join Lobby". המשתמש יכול גם להתנתק על ידי לחיצה על כפתור "Disconnect".



**שם**: מסך פרטי זירת משחק

**תפקיד**: הלקוח מכניס את שם הזירה ואת קוד הכניסה ולוחץ על כפתור "Enter" כדי לשלוח את בקשת ההצטרפות לשרת. הלקוח גם יכול לבחור ללחוץ על כפתור "Create New" כדי ליצור זירת משחק חדשה.

A screenshot of a computer

AI-generated content may be incorrect.

**שם**: מסך יצירת זירת משחק חדשה

**תפקיד**: המסך משמש להכנסת פרטים של זירת משחק חדשה. לסיום ניתן ללחוץ על כפתור "Create" כדי לשלוח את פרטי הרישום לשרת.

A screenshot of a computer

AI-generated content may be incorrect.

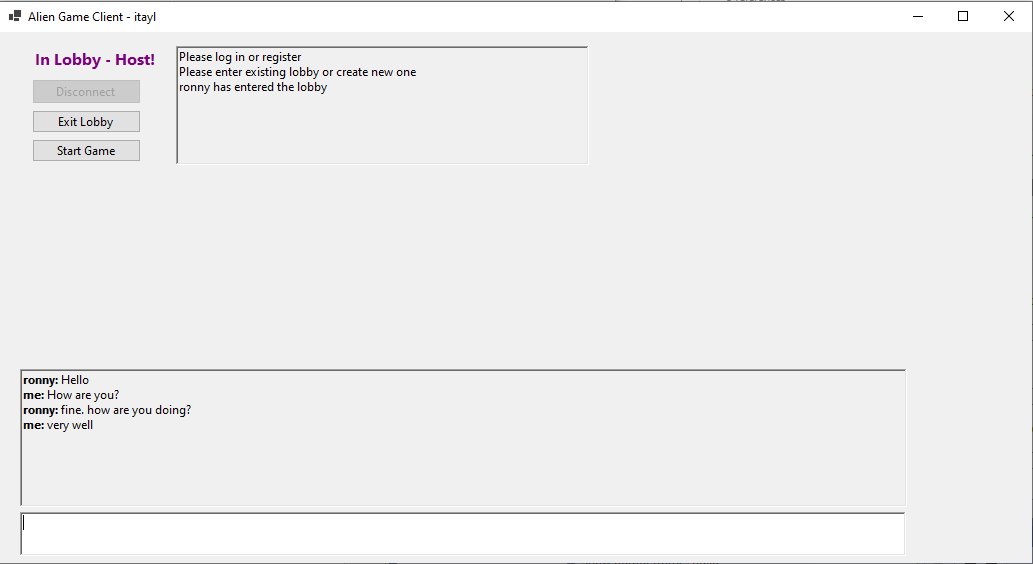
**שם**: מסך המתנה למשחק

**תפקיד**: השחקן נמצא בתוך הזירה וממתין להצטרפות שחקנים נוספים ותחילת המשחק. אם מדובר במנהל הזירה יופיע אצלו גם כפתור "Start Game". הכפתור יהיה לא פעיל כל עוד לא הצטרפו מספיק שיקנים לזירה. בנוסף יכול המשתמש לעזוב את הזירה על ידי לחיצה על כפתור “Exit Lobby".

A screenshot of a computer

AI-generated content may be incorrect.

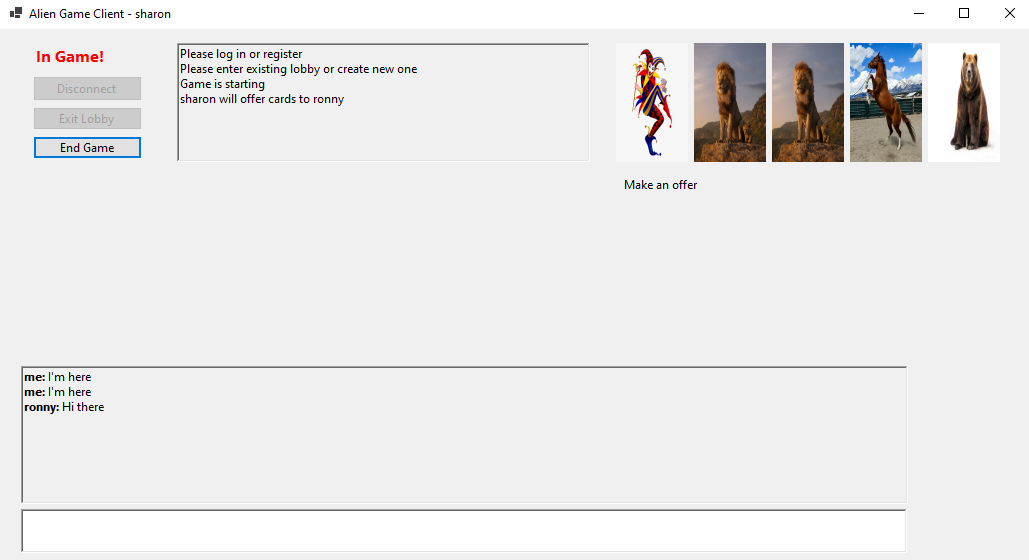
כך נראה המסך אחרי שעוד שחקנים הצטרפו וניתן להתחיל את המשחק. כפתור ה-"Start Game" פעיל ובחלון העידכונים מופיעות הודעות על שחקנים נוספים שהצטרפו. בנוסף, בשלב זה הצ'אט כבר פעיל וכל השחקנים בזירה יכולים להתכתב זה עם זה.



מסכי מהלך המשחק:

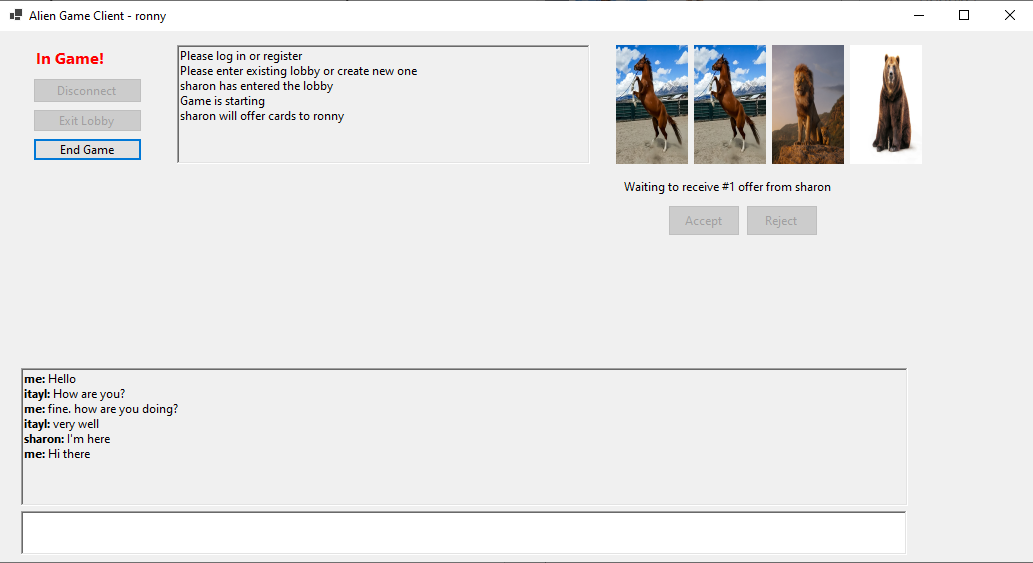
**שם**: מסך השחקן המציע

**תפקיד**: זהו המסך המופיע אצל השחקן שתורו להציע קלף לשחקן הבא בתור. במסך מוצגים הקלפים של השחקן והוא מוזמן לבחור קלף אותו יציע לשחקן הבא בתור. שמו של השחקן אליו תשלח ההצעה מופיע בחלון ההודעות. בנוסף בכל שלב במשחק השחקן יכול ללחוץ על כפתור ה-"End Game" ולגרום להפסקת המשחק.



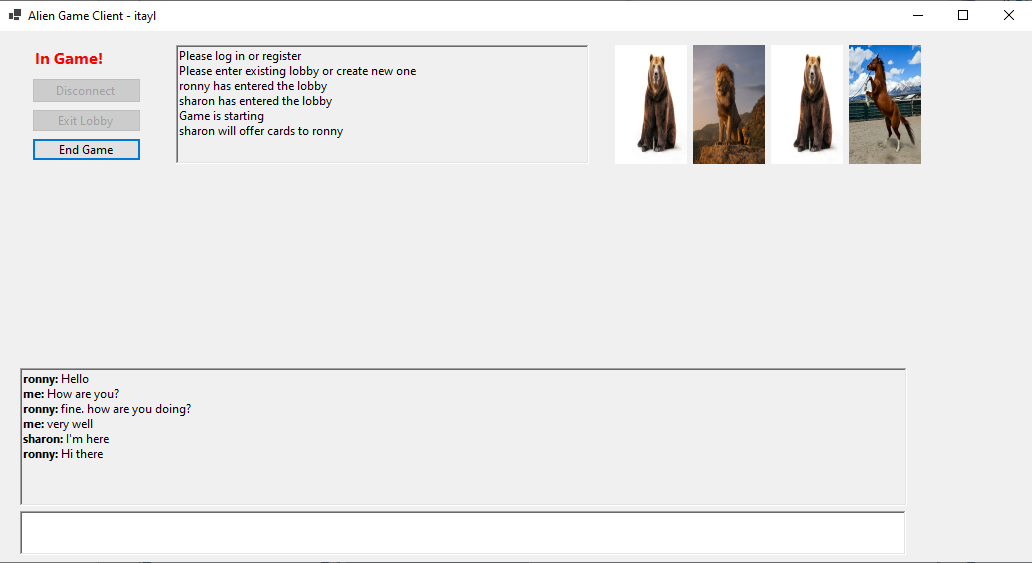
**שם**: מסך השחקן המקבל

**תפקיד**: זהו המסך המופיע אצל השחקן שתורו לקבל קלף מהשחקן שלפניו. במסך מוצגים הקלפים של השחקן. שמו של השחקן שיתן את ההצעה מופיע בחלון ההודעות. בשלב ההמתנה להצעה כפתורי ה-"Accept" ו-"Reject" מופיעים על המסך אך הם אינם פעילים.



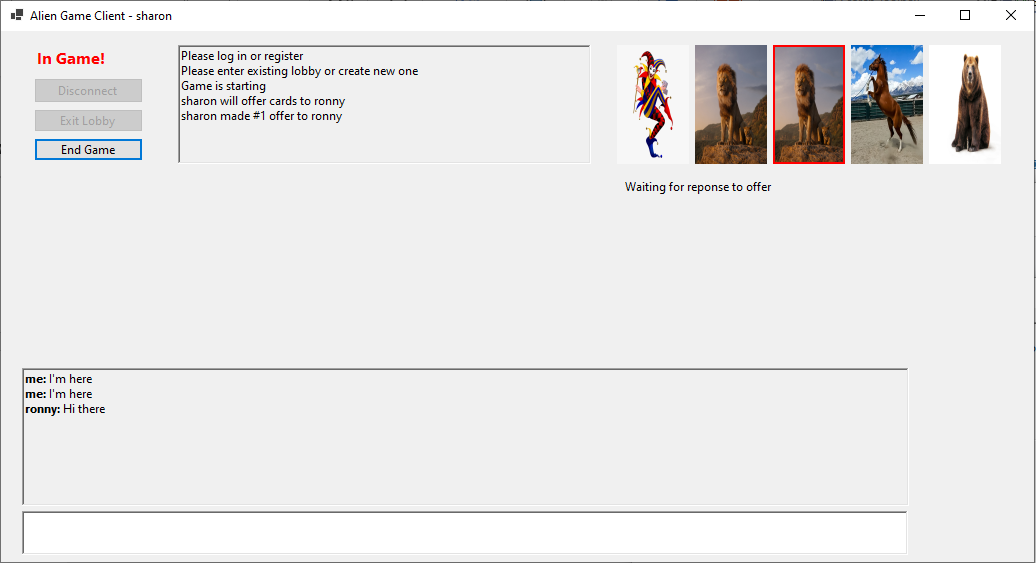
**שם**: מסך השחקנים הממתינים

**תפקיד**: זהו המסך המופיע אצל שאר השחקנים, אלו שלא תורם לתת או לקבל קלף. במסך מוצגים הקלפים של השחקן. שמו של השחקן שתורו לתת ההצעה ושמו של השחקן המקבל מופיעים בחלון ההודעות.



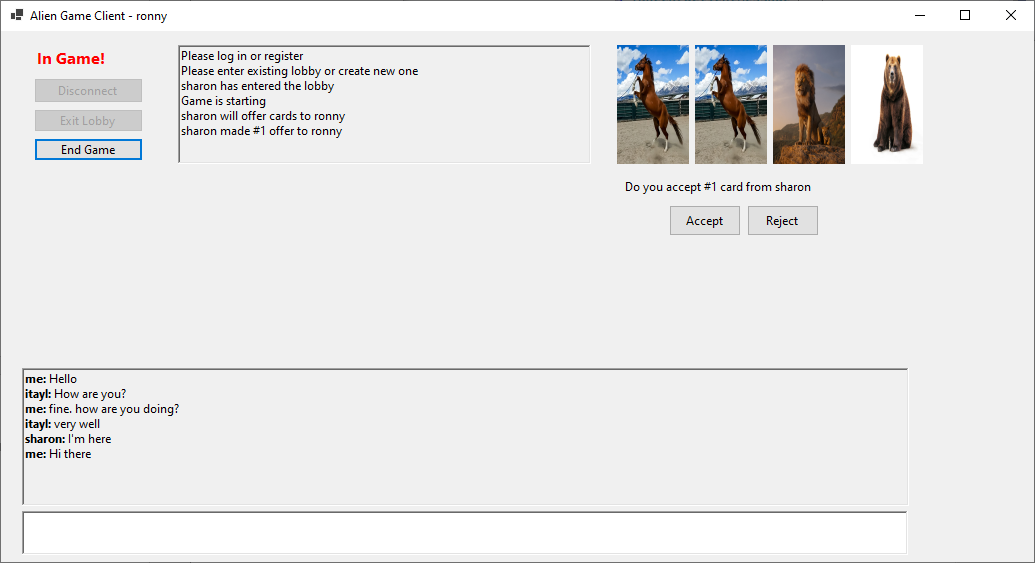
**שם**: מסך המתנה לתשובה

**תפקיד**: המסך של השחקן שתורו לתת הצעה לאחר שבחר את הקלף שברצונו להציע. הקלף הנבחר מודגש במסגרת אדומה ומופיעה הודעה מתחת לקלפים שעל השחקן להמתין לתשובה.



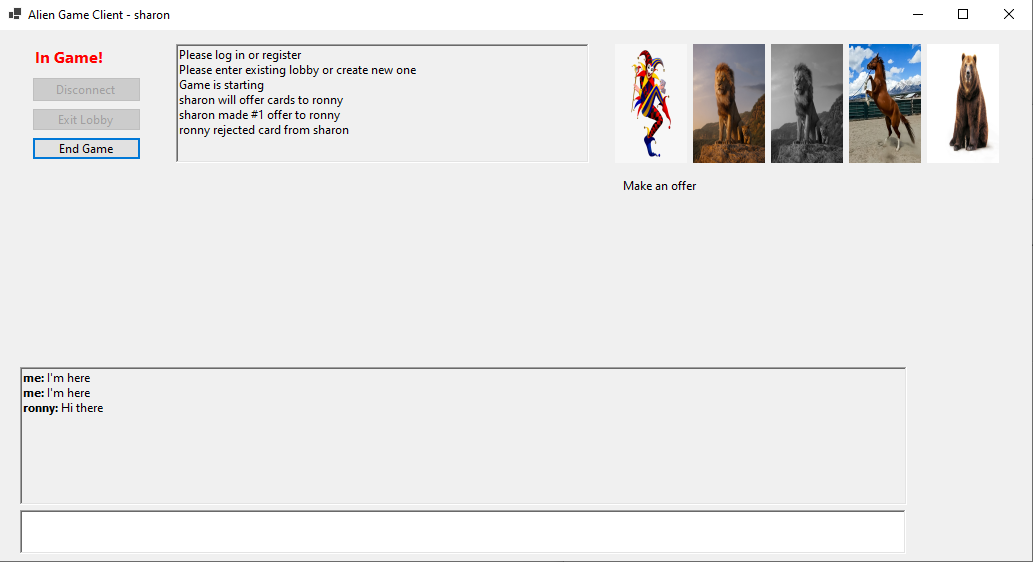
**שם**: מסך קבלת ההצעה

**תפקיד**: המסך של השחקן שתורו לקבל או לדחות את ההצעה לאחר שהשחקן המציע בחר קלף. הקלף המוצע אינו מוצג בשלב וזה ומופיעה הודעה מתחת לקלפים שעל השחקן למסור את תשובתו. השחקן יכול להשתמש בכפתור "Accept" כדי לקבל את הקלף שהוצע אן בכפתור "Reject" כדי לדחות את ההצעה.



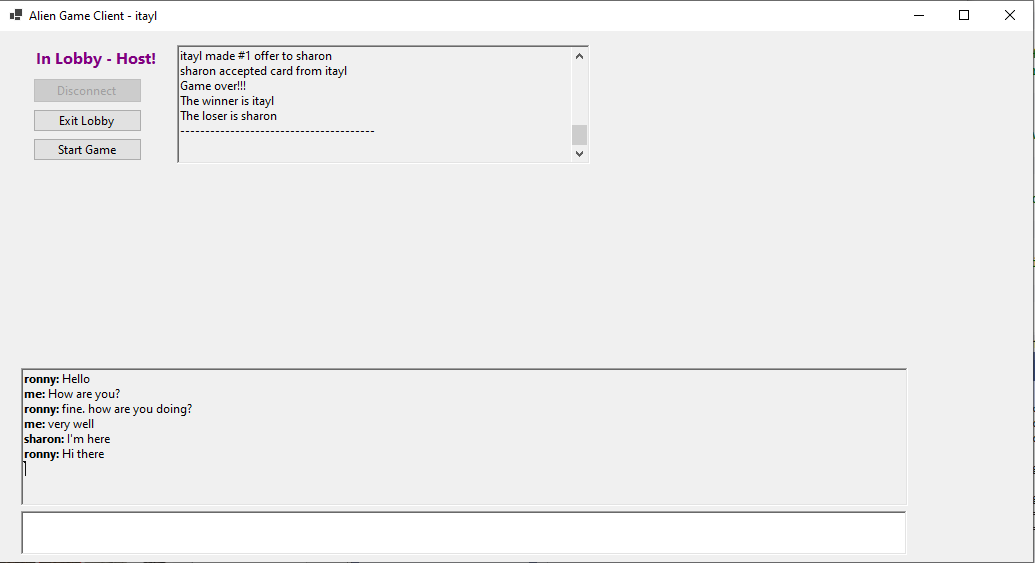
**שם**: מסך השחקן המציע (לאחר סירוב)

**תפקיד**: זהו המסך המופיע אצל השחקן שתורו להציע קלף לשחקן הבא בתור. הקלף שסורב מופיע באפור ואי אפשר לבחור בו שוב.



**שם**: מסך סיום המשחק

**תפקיד**: המסך מודיע בחלון ההודעות שהמשחק הסתיים ובנוסף חושף את זהות השחקן המנצח והשחקן המפסיד. בשלב זה מנהל הזירה יכול להתחיל משחק חדש וכל רחד מהשחקנים יכול לצאת מהזירה.



תיאור מבני הנתונים

בצד השרת נשמרות שתי תיקיות המכילות את פרטי המשתמשים הרשומים ופרטי זירות המשחק.

פרטי המשתמשים

השרת שומר את פרטי כל המשתמשים שנרשמו בתיקיה הנקראת users. עבור כל משתמש נוצר קובץ בפורמט Json ששמו הוא שם המשתמש. בתוך כל קובץ רשומים הפרטים הבאים:

* שם המשתמש
* מחרוזת ה-Salt של המשתמש הזה המשמשת לאבטחת הסיסמא
* הסיסמא המגובבת (בתוספת Salt ו-Pepper)
* כתובת המייל של המשתמש

כאשר מתבצעת פעולת הרשמה של משתמש חדש השרת בודק האם קיים כבר קובץ לאותו שם משתמש ואם כן דוחה את בקשת ההרשמה. אם לא, נוצר קובץ חדש עם כל פרטי ההרשמה בתוכו. כאשר מתבצע ניסיון התחברות (Login) השרת בודק שקיים קובץ עבור שם המשתמש ואז שולף ממנו את מחרוזת ה-Salt והסיסמא המגובבת. השרת מגבב את הסיסמא שנשלחה בבקשת ההתחברות בתוספת מחרוזות ה-Salt וה-Pepper ומשווה לסיסמא המגובבת השמורה בקובץ.

פרטי זירות המשחק

השרת שומר את פרטי כל זירות המשחק שנוצרו בתיקיה הנקראת lobbies. עבור כל זירה נוצר קובץ בפורמט Json ששמו הוא שם הזירה. בתוך כל קובץ רשומים הפרטים הבאים:

* שם זירת המשחק
* מחרוזת ה-Salt של המשתמש הזה המשמשת לאבטחת קוד הכניסה
* קוד הכניסה המגובב (בתוספת Salt ו-Pepper)
* שם המשתש של מנהל הזירה

כאשר מתבצעת פעולת יצירה של זירת משחק חדשה השרת בודק האם קיים כבר קובץ לאותו שם זירה ואם כן דוחה את בקשת היצירה. אם לא, נוצר קובץ חדש עם כל פרטי הזירה בתוכו. כאשר מתבצע ניסיון כניסה לזירה השרת בודק שקיים קובץ עבור שם הזירה ואז שולף ממנו את מחרוזת ה-Salt וקוד הכניסה המגובב. השרת מגבב את קוד הכניסה שנשלח בבקשת היצירה בתוספת מחרוזות ה-Salt וה-Pepper ומשווה לקוד הסיסמא המגובב השמור בקובץ.

סקירת חולשות ואיומים

בספרות ניתן למצוא תאור של מספר חולשות נפוצות באפליקציות שרת-לקוח מהסוג שבניתי והאיום שהן מהוות לבטיחות המידע של המשתמשים ופעולה התקינה של המערכת. בחלק זה אדון בהן ובדרכים בהן הגנתי על המערכת שלי מפניהן.

**חולשות בשמירת הקבצים בצד השרת**

השרת שומר את שמות פרטי המשתמשים ואת פרטי זירות המשחק בקבצים. כל משתמש וכל זירה נשמרים בקובץ משלהם.

**חולשה**: לקוח עויין יכול לשלוח שמות קבצים הכוללים את התווים '/' ו-'.' וכך לגרום לכתיבה של קבצים בכל מקום במערכת הקבצים של המערכת ולדרוך על קבצים חשובים שיפגעו במערכת ההפעלה.

**הגנה:** הלקוח אינו מאפשר יצירה של שמות משתמש הכוללים תווים שאינם אותיות. כך נמנעת שליחה של שמות משתמש המכילים את התווים המסוכנים. בנוסף מתבצעת גם בדיקה של שם המשתמש בצד השרת למקרה שתוכנה המתחזה ללקוח של המערכת תנסה לשלוח שמות משתמש הכוללים את התווים המסוכנים.

**חולשות בתהליך ההתחברות למערכת והגנה על סיסמאות משתמשים**

כל משתמש הרוצה להתחבר למערכת חייב לספק שם משתמש וסיסמא לשרת. השרת מצידו בודק שמתמש בשם זה אכן רשום אצלו, ושהסיסמא שסופקה תואמת את הסיסמא השמורה עבור המשתמש אצל השרת. לתהליך זה מספר חולשות הדורשות הגנה:

**חולשה**: פורץ עלול לגנוב את פרטי ההתחברות של משתמש וכך להתחזות אליו ולהיכנס למערכת.

**הגנה**: הסיסמא של כל משתמש אינה נשמרת כלשונה אלא עוברת תהליך הנקרא גיבוב (Hashing) ונשמרת בתצורתה המגובבת. תהליך הגיבוב ממפה את הסיסמא למחרוזת באורך קבוע. חוזקה של פונקציית הגיבוב הוא בכך שאינה הפיכה כך שבהינתן התצורה המגובבת של הסיסמא, לא ניתן לחשב מה היתתה הסיסמא המקורית. בנוסף פוקצית הגיבוב, בהסתברות גבוהה מאד, תמפה סיסמאות שונות למחרוזות שונות. במערכת שבניתי בחרתי להשתמש בפונקציית SHA256 הנפוצה, הממפה למחרוזת באורך 256 בתים. גם אם יפרץ קובץ פרטי המשתמש, לא ניתן יהיה לעשות שימוש בשם המשתמש כי לא ניתן יהיה לגלות מה הסיסמא (שהרי היא מגובבת). כאשר המשתמש עצמו רוצה להתחבר, השרת לוקח את הסיסמא שהתקבלה, מגבב אותה ומשווה את התוצאה לערך המגובב השמור אצלו. אם הערכים המגובבים זהים, הרי בהכרח נשלחה הסיסמא הנכונה וניתן בבטחה לאפשר למשתמש להתחבר למערכת.

**חולשה**: משתמשים עלולים להשתמש בסיסמאות פשוטות וסטנדרטיות (מילים בשפה, תאריכים מוכרים, וכד.). ברחבי הרשת ניתן למצוא מילונים המכילים כמות גדולה של סיסמאות נפוצות ואת הערך המגובב שלהן, כך שפורץ השם ידו על סיסמא מגובבת יכול למצוא את הסיסמא המקורית על ידי חיפוש במילון שכזה.

**הגנה**: נירצה לגונן על משתמשים שבחרו סיסמא חלשה על ידי שימוש בטכניקה הנקראת "Salt". עבור כל משתמש שנרשם במערכת, השרת יגריל מחרוזת באורך 10 תוים וישמור אותה יחד עם פרטי המשתמש. במקום לשמור את הערך המגובב של הסיסמא שסיפק המשתמש, נצרף את המחוזת האקראית לסוף הסיסמא ונגבב את המחורזת שהתקבלה. אין סיכוי שהערך המגובב של הסיסמא בתוספת המחרוזת האקראית יופיע במילון כלשהו ברשת (יש פשוט יותר מדי אפשרויות). כאשר המשתמש עצמו ירצה להתחבר, השרת יקח את הסיסמא שהתקבלה, יצרף אליה את מחרוזת ה-Salt שנשמרה אצלו, יגבב את המחרוזת שהתקבלה וישווה את התוצאה לערך המגובב השמור אצלו. אם הערכים המגובבים זהים, הרי בהכרח נשלחה הסיסמא הנכונה וניתן בבטחה לאפשר למשתמש להתחבר למערכת.

**חולשה**: בגלל שמחרוזת ה-Salt נמצאת בקובץ פרטי המשתמש ביחד עם הסיסמא המגובבת, קיים עדיין חשש שפורץ המשיג את הקובץ ויודע באיזו פונקציית גיבוב השתמשתי, יוכל לגלות את הסיסמא.

**הגנה**: כדי להוסיף עוד רמה של הגנה אשתמש בטכניקה הנקראת "Pepper". מחרוזת אקראית נוספת תשמר כחלק מאפליקציית השרת עצמה ולכן לא תהיה נגישה למי שיגנוב את הקובץ עם פרטי המשתמש. מחרוזת זו תצורף גם היא לסיסמא (בנוסף למחרוזת ה-Salt) לפני הגיבוב וכך יקשה עוד יותר על פורץ לגנוב את פרטי ההתחברות של משתמש ולהצליח להתחבר למערכת בדרך זו.

הגנות נוספות שלא מימשתי הן:

* דרישה שהמשתמש יבחר סיסמא קשה לניחוש, כלומר כזו הכוללת מספר רב של תווים, אך גם מגוון רחב של תווים כך שתכלול אותיות גדולות וקטנות, מספרים וסימנים אחרים. סיסמאות שאינן עומדות בקריטריונים הללו לא יאושרו, ורישום המשתמש יתבטל.
* מניעה ממשתמש לנסות מספר רב של סיסמאות בזו אחר זו כדי לנחש את הסיסמא הנכונה. הדרך לעשות זאת היא למנוע ממשתמש שכשל מספר פעמים רצופות בניחוש הסיסמא להתחבר במשך זמן מסויים לאחר מכן.

**חולשות בתקשורת בין שרת ללקוחות**

התקשורת בין השרת ללקוחות מתרחשת באמצעות שליחה של מגוון הודעות אשר מסתמכות על פרוטוקול המגדיר מה יהיה המבנה של כל סוג הודעה.

**חולשה**: גורם המתחזה ללקוח שולח הודעה בעלת מבנה שגוי בכוונה. מתקפה זו שייכת לסוג של תקיפות הנקרא Denial of service או DOS בקיצור. המטרה של תקיפות מסוג זה היא לגרום לשגיאות אצל השרת ואף לגרום לו לקרוס וכך למנוע ממנו לשרת את הלקוחות שלו. הצד המקבל את ההודעה מניח שמבנה ההודעה יהיה בהתאם להגדרות הפרוטוקול אך בגלל שאין זה כך תיגרם שגיאה ואולי אפילו קריסה של האפליקציה.

**הגנה**: הצד המקבל את ההודעה אינו מניח שמבנה ההודעה הוא בהכרח תקין, אלא מוודא זאת כחלק מניתוח ההודעה שהתקבלה. כך ניתן לזהות הודעות לא תקינות ולטפל בהן בצורה בטוחה.

כדי לייצר תקשורת מוצפנת בין השרת ללקוחות כך שגורם עוין המאזין לתקשורת לא יוכל להבין מה קורה או להתערב ולשלוח הודעות חוקיות אשר יבלבלו את פעולת המערכת, יש צורך שלשרת וכל לקוח יהיה מפתח הצפנה מוסכם הידוע רק להם. השיטה הראשונה שמימשתי הייתה שבתחילת התקשורת ישלח הלקוח לשרת את מפתח ההצפנה הציבורי שלו והשרת בתמורה ייצר מפתח הצפנה סימטרי, יצפין אותו בעזרת המפתח הציבורי של הלקוח שבידיו, וישלח חזרה לללקוח. מאותו הרגע יכולים השרת והלקוח לתקשר בבטחה באמצעות המפתח הסיצטרי הידוע רק להם.

**חולשה**: שיטה זו פגיעה לתקיפה הנקראת Man in the middle או MITM בקיצור. תקיפה כזו כוללת גורם עויין המאזין לתקשורת הין השרת ללקוח ואף יכול להתערה ולשנות את תוכן ההודעות או לשלוח הודעות בשם השרת או בשם הלקוח. גורם כזה יכול להקשיב להודעה בה משתף הלקוח את השרת במפתח הציבורי שלו (הודעה שאינה מוצפנת), לחסום אותה, ולשלוח במקומה לשרת את המפתח הציבורי שלו. השרת ישלח חזרה את מפתח ההצפנה הסימטרי באופן מוצפן, אך לגורם העויין יש עכשיו את המפתח הנדרש לפיענוח ההצפנה. הגורם העוין מצפין את המפתח הסימטרי מחדש בעזרת מפתח ההצפנה הציבורי של הלקוח השמור אצלו, ושולח אותו ללקוח, שאינו מודע כלל למה שקרה. מאותו הרגע יכול הגורם העויין לפענח את כל ההודעות המוצפנות בין השרת ללקוח ואף להתחזות ללקוח ולשלוח הודעות בשמו.

**הגנה**: השיטה שממשתי כדי להתגבר על החולשה הזו היא לשמור, בזמן כתיבת אפליקציות השרת והלקוח, זוג מפתחות, את הפרטי אצל השרת ואת הציבורי אצל כל אחד מהלקוחות. כך נחסך הצורך להעביר את המפתח הציבורי בתקשורת לא מוצפנת החשופה למאזינים ברשת. בשיטה החדשה, הלקוח הוא זה המייצר את מפתח ההצפנה הסימטרי, מצפין אותו בעזרת המפתח הציבורי הנמצא בידיו ושולח אותו לשרת. השרת מחזיק בידיו את המפתח הפרטי הדרוש לפיענוח ההודעה. מאותו הרגע יש בידי שני הצדדים מפתח הצפנה סימטרית מוסכם הידוע רק להם והם יכולים ךתקשר בבטחה.

**מימוש הפרויקט**

סקירת המודולים והמחלקות

מודולים ומחלקות מיובאות

מודול **System.Net.Sockets**

מספק יישום מנוהל של ממשק Windows Sockets (Winsock) עבור מפתחים שצריכים לשלוט בגישה לרשת. בתוך מודול זה השתמשתי במחלקות הבאות:

* **TcpListener** - מספקת מתודות פשוטות שמאזינות ומקבלות בקשות חיבור נכנסות במצב סינכרוני חוסם. ניתן להשתמש ב-TcpClient כדי להתחבר עם TcpListener. ניתן ליצור אותו באמצעות כתובת IP מקומית ומספר פורט, או רק מספר פורט.
* **TcpClient** - המחלקה מספקת שיטות פשוטות לחיבור, שליחה וקבלה של נתוני זרם (Data Stream) דרך רשת במצב חסימה סינכרוני. כדי ש-TcpClient יוכל להתחבר ולהחליף נתונים, TcpListener חייב להאזין לבקשות חיבור נכנסות.
* **NetworkStream** – המחלקה מספקת שיטות לשליחה וקבלה של נתונים דרך sockets במצב חסימה. ניתן להשתמש במחלקה הן להעברת נתונים סינכרונית והן לתקשורת אסינכרונית.

מודול **System.IO**

מכיל טיפוסים המאפשרים קריאה וכתיבה לקבצים ולזרמי נתונים, וטיפוסים המספקים תמיכה בסיסית בקבצים ובספריות. בתוך מודול זה השתמשתי במחלקות הבאות:

* **StreamReader** - מיועד לקלט תווים בקידוד מסוים. ניתן להשתמש בו לקריאת שורות מידע מזרם נתונים של NetworkStream.
* **StreamWriter** - מיועד לפלט תווים בקידוד מסוים. ניתן להשתמש בו לכתיבת מידע לזרם נתונים של NetworkStream.

מודול **System.Threading**

מספק מחלקות וממשקים המאפשרים תכנות מרובה הליכי משנה. בתוך מודול זה השתמשתי במחלקות הבאות:

* **Thread** - יוצר ושולט בתהליכון, מגדיר את העדיפות שלו ומקבל את הסטטוס שלו.
* **Lock** - ניתן להשתמש במחלקה כדי להגדיר אזורי קוד הדורשים גישה בלעדית בין תהליכי משנה של תהליך, המכונים בדרך כלל מקטעים קריטיים, כדי למנוע גישות בו-זמניות למשאב.

מודול **System.Security.Cryptography**

מספק שירותי קריפטוגרפיה, כולל קידוד ופענוח מאובטחים של נתונים, כמו גם פעולות רבות אחרות, כגון גיבוב, יצירת מספרים אקראיים ואימות הודעות. בתוך מודול זה השתמשתי במחלקות הבאות:

* **RSACryptoServiceProvider** - מבצע הצפנה ופענוח אסימטריים באמצעות יישום אלגוריתם ה-RSA.
* **Aes** – מחלקה המספקת שיטות להצפנה ופענוח סימטריים באמצעות אלגוריתם AES (תקן הצפנה מתקדם).
* **SHA256** – מחלקה המייצרת גיבוב המשמש כערך ייחודי בגודל קבוע המייצג כמות גדולה של נתונים. גיבובי נתונים של שתי קבוצות נתונים צריכים להתאים אם ורק אם הנתונים המתאימים תואמים גם הם. שינויים קטנים בנתונים גורמים לשינויים גדולים ובלתי צפויים בגיבוב.

מודול **System.Windows.Forms**

מכיל מחלקות ליצירת יישומים מבוססי Windows המנצלים באופן מלא את תכונות ממשק המשתמש העשירות הזמינות במערכת ההפעלה Microsoft Windows. בתוך מודול זה השתמשתי במחלקות הבאות:

* **Form** – ייצוג של כל חלון המוצג ביישום. ניתן להשתמש במחלקה Form כדי ליצור חלונות סטנדרטיים, חלונות כלים, חלונות ללא גבולות וחלונות צפים. ניתן גם להשתמש במחלקה כדי ליצור חלונות מודאליים כגון תיבת דו-שיח.
* **Button** – מייצג כפתור ב-Windows. ניתן ללחוץ על הכפתור באמצעות העכבר, מקש ENTER או מקש הרווח אם הכפתור נמצא בפוקוס.
* **RichTextBox** – בעזרתו המשתמש יכול להזין ולערוך טקסט. הוא מספק גם תכונות עיצוב מתקדמות. ניתן להקליד טקסט ישירות, או לטעון אותו מקובץ טקסט עשיר (RTF) או מקובץ טקסט רגיל.
* **PictureBox** – משמש להצגת גרפיקה מקובץ bitmap, מטא-קובץ, אייקון, JPEG, GIF או PNG.
* **MessageBox** – מציג חלון הודעות, המכונה גם תיבת דו-שיח, המציג הודעה למשתמש. זהו חלון מודאלי, החוסם פעולות אחרות ביישום עד שהמשתמש סוגר אותו. החלון יכול להכיל טקסט, כפתורים וסמלים המספקים מידע והנחיות למשתמש.

מודול **System.Collections.Generic**

מכיל מגוון של מחלקות אוספים גנריות. מחלקות אלו מספקות מבני נתונים בטוחים לסוגים שיכולים לאחסן אוספים של אובייקטים. בתוך מודול זה השתמשתי במחלקות הבאות:

* **List** – מייצגת רשימה מוגדרת היטב של אובייקטים שניתן לגשת אליהם באמצעות אינדקס. מספקת שיטות לחיפוש, מיון ותפעול רשימות.
* **Dictionary** - מחלקה גנרית זו מספקת מיפוי מקבוצת מפתחות לקבוצת ערכים. כל תוספת למילון מורכבת מערך ומפתח המשויך לו. אחזור ערך באמצעות המפתח שלו הוא מהיר מאוד, קרוב ל-O(1), מכיוון שהמחלקה ממומשת כטבלת גיבוב.

מודולים ומחלקות שפיתחתי

התוכנה שפיתחתי מחולקת לשלושה מודולים (פרויקטים בתוך ה-Solution של VisualStudio).

מודול **Server**

התוכנה של אפליקצית השרת. התוצר שלה הוא קובץ הניתן להרצה (executable) המהווה את השרת של המערכת שפיתחתי. בתוך מודול זה ממומשות המחלקות הבאות:

**שם**: Server

**קובץ**: Server.cs

**תפקיד**: המחלקה המרכזית של המודול. יורשת ממחלקת System.Windows.Forms.Form של השפה. אחראית על ניהול ההתחברות הראשונית של כל הלקוחות ומחזיקה את רשימת כל הלקוחות המחוברים. בנוסף אחראית על פתיחת זירות והצטרפות משתמשים אליהם ומחזיקה את רשימת כל הזירות הפעילות.

**תכונות המחלקה:**

* listener\_ – אובייקט TcpListener האחראי על יצירת הקשר הראשוני עם כל הלקוחות המבקשים להתחבר. זו היא תכונה פרטית של המחלקה. מאותחלת בפעולת הבנאי של המחלקה.
* Clients – אובייקט מסוג List<ClientHandler> המחזיק רשימת אובייקטים המייצגים את כל הלקוחות הפעילים כרגע. תכונה זו ניתנת לשינוי רק מתוך המחלקה (private) אך ניתנת לקריאה גם מבחוץ (public). רשימה זו מאותחלת ריקה בפעולת הבנאי ובכל פעם שמצטרף לקוח חדש האובייקט המייצג אותו מתווסף לרשימה.
* Lobbies – אובייקט מסוג Dictionary<string, Lobby> המחזיק מיפוי משמות הזירות הפעילות לאובייקטים המייצגים אותן. תכונה זו ניתנת לשינוי רק מתוך המחלקה (private) אך ניתנת לקריאה גם מבחוץ (public). מילון זה מאותחל ריק בפעולת הבנאי ובכל פעם שנפתחת זירת משחק חדשה האובייקט המייצג אותה מתווסף למילון.
* PrivateKey – אובייקט מסוג מחרוזת המחזיק את המפתח הפרטי של השרת עבור הצפנת RSA. התכונה הזאת היא ציבורית. התכונה מאותחלת בפעולת הבנאי של המחלקה.
* PublicKey – אובייקט מסוג מחרוזת המחזיק את המפתח הציבורי של השרת עבור הצפנת RSA. התכונה הזאת היא ציבורית. התכונה מאותחלת בפעולת הבנאי של המחלקה.
* logger\_ - אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה. התכונה היא פרטית ומאותחלת בפעולת הבנאי של המחלקה.

**פעולות המחלקה:**

* Server()

**טענת כניסה:** אין

**טענת יציאה:** פעולת הבנאי של המחלקה. מאתחלת את התכונות של המחלקה ואת אובייקטי ממשק המשתמש. אינה מחזירה דבר.

* DisconnectUser(ClientHandler) public void

**טענת כניסה:** אובייקט מסוג ClientHandler המייצג את הלקוח שמתנתק.

**טענת יציאה:** הפעולה מסירה את הלקוח מרשימת הלקוחות הפעילים ומנתקת אותו מהשרת. אינה מחזירה דבר.

* public bool JoinLobby(string name, ClientHandler guest, out string? reason)

**טענת כניסה:** מחרוזת של שם הזירה, אובייקט מסוג ClientHandler המייצג את הלקוח המבקש להצטרף לזירה.

**טענת יציאה:** אם הזירה קיימת (נמצאת ברשימת הזירות הפעילות) ולא מתקיים בה משחק פעיל אז קוראים לפעולה של הזירה שמצרפת את המשתמש. כמו כן נקראת הפעולה באובייקט הלקוח המעדכנת שהלקוח נכנס לזירה. אם הזירה לא קיימת או מתקיים בה משחק פעיל מוחזרת מחרוזת המכילה את סיבת כישלון ההצטרפות. הפעולה מחזירה גם ערך בוליאני המציין האם הפעולה הצליחה או נכשלה.

* public void OpenLobby(string name, ClientHandler guest)

**טענת הכניסה:** מחרוזת המציינת את שם הזירה המופעלת ואובייקט מסוג ClientHandler המייצג את הלקוח המבקש לפתוח את הזירה.

**טענת יציאה:** נוצר אובייקט המייצג את הזירה ונקראת הפעולה של הזירה שמצרפת את המשתמש. כמו כן נקראת הפעולה באובייקט הלקוח המעדכנת שהלקוח נכנס לזירה. מופעל התהליכון שינהל את פעילות הזירה. אינה מחזירה דבר.

* public void AddClient(ClientHandler client)

**טענת כניסה:** אובייקט מסוג ClientHandler המייצג את הלקוח שהצטרף לשרת.

**טענת יציאה:** מצרף את הלקוח לרשימת הלקוחות הפעילים. אינה מחזירה דבר.

* public bool IsClientLoggedIn(string username)

**טענת כניסה:** מחרוזת המייצגת את שם הלקוח.

**טענת יציאה:** בודקת האם הלקוח כבר ביצע פעולת התחברות (Login) ומחזירה ערך בולאני המייצג את תוצאת הבדיקה.

* private void LoadAsync(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** מופעלת תכונת listener\_ כך שתתחיל להאזין לנסיונות התחברות של לקוחות ומופעל התהליכון שיטפל בניסיונות ההתחברות. אינה מחזירה דבר.

* void ListenerAction(CancellationToken token)

**טענת כניסה:** אובייקט המאפשר להודיע לתהליכון שעליו להסתיים.

**טענת יציאה:** לולאה אינסופית המאזינה לניסיונות התחברות לשרת. אם מזוהה ניסיון התחברות נשלחת הודעת אישור ללקוח ונוצר עבורו אובייקט מסוג ClientHandler אשר ינהל את המשך ההתחברות איתו ומופעל התהליכון אשר יעשה זאת. אינה מחזירה דבר.

* internal void PurgeLobby(Lobby lobby)

**טענת כניסה:** אובייקט המייצג את הזירה הנסגרת.

**טענת יציאה:** הזירה הנסגרת נמחקת מרשימת הזירות הפעילות. אינה מחזירה דבר.

**שם**: ClientHandler

**קובץ**: ClientHandler.cs

**תפקיד**: מחלקה זו מנהלת את התקשורת עם הלקוח טרם הצטרף לזירת משחק, כלומר מחלקה זו מטפלת בתהליך איתחול פרוטוקול ההצפנה (החלפת מפתחות), בתהליך התחברות הלקוח (Login) או יצירת משתמש חדש ובתהליך הכניסה לזירת משחק או יצירת זירה.

**תכונות המחלקה:**

* counter\_ - מספר סידורי ייחודי עבור כל אחד מהלקוחות המיוחדים. זו היא תכונה סטטית של המחלקה. בכל פעם שמצטרף לקוח חדש הוא מקבל את הערך הנוכחי של משתנה זה וערך המשתנה מקודם.
* server\_ - הפנייה לאובייקט ה-Server של האפליקציה.
* logger\_ - אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה. התכונה היא פרטית ומאותחלת בפעולת הבנאי של המחלקה.
* UserData – אובייקט המכיל מידע על המשתמש. נוצר בעקבות פעולת התחברות מוצלחת של הלקוח.
* Id – הפנייה למספר הזיהוי של הלקוח. חושף את מספר הלקוח השמור באובייקט ה-UserData.

**פעולות המחלקה:**

* public ClientHandler(TcpClient client, Server server)

**טענת כניסה:** אובייקט מסוג TcpClient המנהל את פרוטוקול התקשורת עם הלקוח והפנייה לאובייקט ה-Server של האפליקציה.

**טענת יציאה:** פעולת הבנאי של המחלקה. מאתחלת את אובייקט ה-UserData עם מספר זיהוי חדש ומאתחלת את אובייקט ה-logger\_.

* public void Start()

**טענת כניסה:** אין

**טענת יציאה:** מפעיל את התהליכון המנהל את התקשורת עם הלקוח. אינה מחזירה דבר.

* void ClientLoop()

**טענת כניסה:** אין

**טענת יציאה:** מצרפים את הלקוח לרשימת הלקוחות הפעילים המנוהלת על ידי אובייקט ה-Server. שולחים ללקוח את מספר הזיהוי שלו. אז מתחילה לולאה אינסופית המטפלת בכל ההודעות המגיעות מהלקוח. הלולאה מסתיימת כאשר הלקוח מצטרף לזירת משחק. אינה מחזירה דבר.

* void Disconnect()

**טענת כניסה:** אין

**טענת יציאה:** מודיע לאובייקט השרת שהלקוח מתנתק. אינה מחזירה דבר.

* private void HandleParsedMessage(CommMessage msg)

**טענת כניסה:** אובייקט המייצג את ההודעה שהתקבלה מהלקוח.

**טענת יציאה:** טיפול פרטני בכל אחת מההודעות שיכולות להתקבל מהלקוח:

* + PublicKey: הלקוח שלח לשרת את ה-PublicKey שלו. השרת מגיב בהצפנת מפתח ה-AES ושליחתו חזרה ללקוח.
  + LoginRequest: השרת מוודא שאותחלה התקשורת המאובטחת ושהלקוח אינו מחובר כבר. השרת שולף ומאמת את פרטי ההתקשרות של הלקוח ומידע את הלקוח בתוצאות הפעולה (והסיבה לכישלון אם היה כזה).
  + RegisterRequest: השרת מנסה ליצור משתמש חדש על סמך הפרטים שהתקבלו בהודעה מהלקוח ומעדכן אותו בתוצאות הפעולה ובסיבה לכישלון אם היה כזה.
  + JoinLobbyRequest: השרת מנסה להכניס את המשתמש לזירה על ידי אימות פרטי הכניסה שנשלחו על ידו. אם המשתמש הוא מנהל הזירה תיפתח הזירה ותרשם כזירה פעילה אחרת יצורף המשתמש לזירה אם היא כבר פתוחה. הודעה תשלח ללקוח לדיווח על תוצאות הכניסה לזירה.
  + CreateLobbyRequest: השרת מנסה לרשום זירה חדשה על פי הפרטים שנשלחו על ידי הלקוח. הודעה נשלחת חזרה ללקוח לדווח על תוצאות ההרשמה.

אינה מחזירה דבר.

* internal void EnterLobby(Lobby lobby)

**טענת כניסה:** אובייקט המייצג זירת משחק.

**טענת יציאה:** רושם באובייקט ה-UserData את הזירה אליה הצטרף המשתמש. אינה מחזירה דבר.

**שם:** Lobby

**שם הקובץ:** Lobby.cs

**תפקיד:** ניהול זירת המשחק. אחראית על איתחול המשחק, ניהול מהלך המשחק והתקשורת בין השחקנים, סיום המשחק וטיפול בתקלות ובקשות יציאה של שחקנים.

**תכונות המחלקה:**

* Name – מחרוזת המחזיקה את שם הזירה.
* Host – מחרוזת המחזיקה את שם מנהל הזירה.
* Game – אובייקט המייצג את מצב המשחק בכל רגע ורגע.
* GameInProgress – משתנה בוליאני המציין האם כרגע מתנהל משחק בזירה.
* guestInfo\_ - מילון המחזיק מיפוי ממספר הזיהוי של הלקוח לאובייקט המייצג אותו. משמש גם כרשימת כל הלקוחות המתארחים כרגע בזירה.
* hostInfo\_ - האובייקט המייצג את הלקוח שהוא מנהל הזירה.
* server\_ - הפנייה לאובייקט הסרבר של האפליקצייה.
* socketToClientDict\_ - מילון הממפה בין שקעים לאובייקטים המייצגים את הלקוחות. מאפשר לזהות כאשר מתקבלת הודעה, מאיזה לקוח היא הגיעה.
* logger\_ - אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה. התכונה היא פרטית ומאותחלת בפעולת הבנאי של המחלקה.
* UpdateLock – מנעול סינכרון המונע שינוי בו זמני של תכונות המחלקה.

**פעולות המחלקה:**

* public override string ToString()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה מחרוזת המייצגת את אובייקט הזירה.

* public ClientHandler GetClientHandler(int id)

**טענת כניסה:** מספר הזיהוי של הלקוח.

**טענת יציאה:** שולף את אובייקט ה-ClientHandker מתוך מילון ה-guestInfo\_ ומחזיר אותו.

* public Lobby(string name, ClientHandler host, Server server)

**טענת כניסה:** מחרוזת המייצג את שם הזירה, אובייקט ה-ClientHandler של מנהל הזירה והפנייה לאובייקט ה-Server של האפליקציה.

**טענת יציאה:** פעולת הבנאי של המחלקה. אתחול של תכונות המחלקה ורישום המנהל כאורח בזירה.

* private void log(string text)

**טענת כניסה:** מחרוזת המייצגת את ההודעה שתרשם לקובץ התיעוד.

**טענת יציאה:** רושמת את ההודעה לקובץ התיעוד. אינה מחזירה דבר.

* public List<UserData> GetGuestUsers()

**טענת כניסה:** אין.

**טענת יציאה:** שולף את רשימת האורחים בזירה ומחזיר אותה.

* private void CloseLobby (string msg)

**טענת כניסה:** מחרוזת המכילה את ההודעה שתשלח לכל האורחים בזירה.

**טענת יציאה:** הודעה לכל האורחים שהזירה נסגרת, סגירת הלולאה המטפלת בהודעות מהאורחים והפעלת ללואות התקשורת של כל אחד מהאורחים מול השרת כדי שיוכלו להצטרף לזירה אחרת. אינה מחזירה דבר.

* private void RemoveGuest(string msg, string leaver)

**טענת כניסה:** מחרוזת המכילה את ההודעה שתשלח לכל האורחים בזירה ומחרוזת המכילה את שם המשתמש העוזב את הזירה.

**טענת יציאה:** הוצאת המשתמש שעוזב ממבני הנתנוים של המחלקה והודעה לכל האורחים האחרים שהמשתמש עזב. אינה מחזירה דבר.

* public void ExitLobbyRequest(string name)

**טענת כניסה:** שם המשתמש המבקש לעזוב את הזירה.

**טענת יציאה:** אם המשתמש המבקש לעזוב הוא מנהל הזירה נשלחת הודעה לכל האורחים שהזירה נסגרת ולאחר מכן הזירה נסגרת. אם אחד האורחים מבקש לעזוב אז נשלחת הודעה לכל האורחים האחרים בזירה והמשתמש מוצא ממנה. אינה מחזירה דבר.

* internal void BroadcastGameLogMessage(string msg)

**טענת כניסה:** מחרוזת המייצגת אתההודעה שתשלח.

**טענת יציאה:** ההודעה נשלחת לכל האורחים בזירה ותופיעה בחלון ההעידכונים שלהם. אינה מחזירה דבר.

* public void StartGame()

**טענת כניסה:** אין.

**טענת יציאה:** מאתחלת את אובייקט מצב המשחק ומחלק קלפים לכל השחקנים. שולח הודעות לשחקנים שתורם לשחק. אינה מחזירה דבר.

* public bool CheckGameOver()

**טענת כניסה:** אין.

**טענת יציאה:** בודקת את מצב המשחק ומזהה אם אחד השחקנים מחזיק יד מנצחת. אם כן נשלחת הודעה לכל השחקנים כדי לידע אותם מי השחקן המנצח ומי המפסיד ולאחר מכן סוגרת את המשחק. מחזירה משתנה בוליאני המציין אם המשחק הסתיים או לא.

* public void NotifyClientsOfInterrupt(InterruptGameMessage msg)

**טענת כניסה:** הודעה בדבר שחקן שעזב את המשחק באמצע.

**טענת יציאה:** ההודעה נשלחת לכל השחקנים. אינה מחזירה דבר.

* public void NotifyClientsOfNewTurn()

**טענת כניסה:** אין.

**טענת יציאה:** מודיעה לשחקן המציע קלף ולשחקן אליו מופנית ההצעה על תפקידם בתור הקרוב במשחק. אינה מחזירה דבר.

* public void EndGame()

**טענת כניסה:** אין.

**טענת יציאה:** מסיימת את המשחק על ידי איפוס אובייקט מצב המשחק. אינה מחזירה דבר.

* public void AddGuest(ClientHandler guest)

**טענת כניסה:** אובייקט המייצג את הלקוח המצטרף כאורח.

**טענת יציאה:** הוספת הלקוח למבני הנתונים של המחלקה, הודעה לשאר האורחים שאורח חדש הצטרף לזירה והודעה למנהל הזירה אם יש מספיק אורחים כדי להתחיל משחק. אינה מחזירה דבר.

* public void Start()

**טענת כניסה:** אין.

**טענת יציאה:** הפעלת הלולאה המטפלת בהודעות המתקבלות מכל האורחים. אינה מחזירה דבר.

* public void WriteUser(int id, CommMessage message)

**טענת כניסה:** מספר הזיהוי שך הלקוח אליו תשלח ההודעה ואובייקט המייצג את ההודעה הנשלחת.

**טענת יציאה:** כתיבת ההודע לזרם הכתיבה של הלקוח. אינה מחזירה דבר.

* public void HandleUserLeft(ClientHandler client, bool isError)

**טענת כניסה:** אובייקט המייצג את הלקוח שעוזב ומשתנה בוליאני המציין האם המשתמש עזב בגלל שגיאה.

**טענת יציאה:** אם הלקוח שעוזב הוא מנהל הזירה אז צריך לסגור את הזירה, אחרת מבטלים את המשחק הנוכחי ומוציאים את הלקוח העוזב מן הזירה. אינה מחזירה דבר.

* private void MessageLoop(CancellationToken token)

**טענת כניסה:** אובייקט המאפשר להפסיק את ריצת הלולאה מבחוץ.

**טענת יציאה:** זו הלולאה המרכזית המנהלת את התקשורת של כל האורחים עם הזירה לפני ןבמהלך המשחק אינה מחזירה דבר.

**שם:** Game

**שם הקובץ:** Game.cs

**תפקיד:** מעקב אחר מהלך המשחק והקלפים של כל השחקנים. המחלקה מחלקת את הקלפים בצורה הוגנת ומיודעת לזהות מתי מסתיים המשחק

**תכונות המחלקה:**

* Players – רשימה המחזיקה את כל האובייקטים המייצגים את השחקנים במשחק
* Receiver – אובייקט המייצג את השחקן שתורו כרגע לקבל הצעת קלף
* Giver – אובייקט המייצג את השחקן שתורו כרגע להציע קלף
* OfferedCard – אובייקט המייצג את הקלף המוצע כרגע (יכול שלא יהיה כזה קלף)
* NumRejections – כמה הצעות כבר סורבו על ידי השחקן שתורו לקבל הצעות
* State– משתנה היכול לקבל אחד מהערכים הבאים המייצגים את המצב הנוכחי שח המשחק: WaitForOffer או WaitForResponse

**פעולות המחלקה:**

* public Game(List<Player> players)

**טענת כניסה:** אין.

**טענת יציאה:** פעולת הבנאי של המחלקה. מאתחלת את מצב המשחק ומחלקת את הקלפים לשחקנים

* private List<T> RandomizeOrder<T>(IEnumerable<T> source)

**טענת כניסה:** רשימה של אובייקטים מכל סוג.

**טענת יציאה:** מחזירה רשימה של האובייקטים בסדר אקראי.

* public override string ToString()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה מחרוזת המייצגת את האובייקט.

* public void advanceGameState()

**טענת כניסה:** אין.

**טענת יציאה:** מקדמת את מצב המשחק בסיום תור. מעדכנת בהתאם את כל התכונות של המחלקה. אינה מחזירה דבר.

* private void DealCards()

**טענת כניסה:** אין.

**טענת יציאה:** מערבבת את קלפי המשחק ומחלקת אותם לשחקנים בצורה הוגנת. אינה מחזירה דבר.

* public Player? DoWeHaveAWinner()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה אובייקט המייצג את השחקן המנצח אם יש כזה. אחרת מחזירה null.

* public Player FindPlayerWithJoker()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה אובייקט המייצג את השחקן המנצח המחזיק בקלף הג'וקר.

**שם:** Player

**שם הקובץ:** Player.cs

**תפקיד:** מייצגת את הקלפים של שחקן במשחק. מאפשרת עידכון של מצב הקלפים בהתאם להתפתחויות במשחק

**תכונות המחלקה:**

* Id – מספר שלם המהווה את המזהה הייחודי של השחקן.
* Name – שם המשתמש של השחקן.
* Cards – רשימת הקלפים שמחזיק השחקן.

**פעולות המחלקה:**

* public Player(int id, String name)

**טענת כניסה:** מספר שלם המההוה את המזהה של השחקן ומלרוזת המייצגת את שם המשתמש של השחקן.

**טענת יציאה:** זוהי הפעולת הבנאי של המחלקה. מאתחלת את התכונות שלו.

* public void RemoveCard(Card card)

**טענת כניסה:** אובייקט המייצג את הקלף הנמסר לשחקן אחר.

**טענת יציאה:** מסירה את הקלף מרשימת הקלפים. אינה מחזירה דבר.

* public void AddCard(Card card)

**טענת כניסה:** אובייקט המייצג את הקלף שהתקבל משחקן אחר.

**טענת יציאה:** מוסיפה את הקלף לרשימת הקלפים. אינה מחזירה דבר.

* public override string ToString()

**טענת כניסה:** אין

**טענת יציאה:** מחזירה מחרוזת המייצגת את אובייקט ה-Player.

* public void Initialize(List<Card> cards)

**טענת כניסה:** רשימת הקלפים ההתחלתית של השחקן.

**טענת יציאה:** מאתחלת את רשימת הקלפים של השחקן.

**שם:** UserData

**שם הקובץ:** UserData.cs

**תפקיד: מ**אגדת אינפורמצייה על כל משתמש פעיל המוחזקת בצד של השרת כגון האוביקטים שמנהלים את התקשורת עם המשתמש, שם המשתמש ומספר הזיהוי שלו ומצב החיבור של המשתמש.

**תכונות המחלקה:**

* Id – המספר המזהה של המשתמש.
* Client – אובייקט מסוג TcpClient המייצג את החיבור של הלקוח עם השרת
* nws\_ – אובייקט מסוג NetworkStream לניהול הכתיבה והקריאה מול הלקוח
* Reader – אובייקט האחראי על קריאת מחרוזות מזרם הנתונים של הלקוח
* Writer – אובייקט האחראי על כתיבת מחרוזות לזרם הנתונים של הלקוח
* Name – שם המשתמש של הלקוח (אם ידוע)
* LoggedIn – משתנה בוליאני המציין האם הלקוח ביצע Login
* Lobby – הפנייה לאובייקט הזירה אליה שייך המשתמש (אם יש כזו)
* InLobby – משתנה בוליאני המציין האם הלקוח שייך לזירת משחק
* PublicKey – מפתח ההצפנה הציבורי של הלקוח

**פעולות המחלקה:**

* public UserData(TcpClient client, int id)

**טענת כניסה:** אובייקט מסוג TcpClient המייצג את החיבור של הלקוח עם השרת ומספר הזיהוי של הלקוח.

**טענת יציאה:** זוהי פעולת הבנאי של המחלקה.

* public void ResetLobby()

**טענת כניסה:** אין.

**טענת יציאה:** מאפסת את אוביקט הזירה אליה שייך המשתמש. אינה מחזירה דבר.

* public void ResetName()

**טענת כניסה:** אין.

**טענת יציאה:** מאפסת את שם המשתמש של הלקוח.

* public override string ToString()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה מחרוזת המייצגת את אובייקט הזירה.

* public override string ToString()

**טענת כניסה:** אין.

**טענת יציאה:** מחזירה מחרוזת המייצגת את האובייקט.

* public void Close()

**טענת כניסה:** אין.

**טענת יציאה:** סוגרת את זרמי הנתונים של התקשורת עם הלקוח ואת חיבור ה-TCP עם הלקוח.

מודול **Client**

התוכנה של אפליקצית הלקוח. התוצר שלה הוא קובץ הניתן להרצה (executable) המהווה את הלקוחות של המערכת שפיתחתי. בתוך מודול זה ממומשות המחלקות הבאות:

**שם**: Client

**קובץ**: Client.cs

**תפקיד**: המחלקה המרכזית של המודול. יורשת ממחלקת System.Windows.Forms.Form של השפה. אחראית על ניהול ההתחברות הראשונית של הלקוח עם השרת ומנהלת את התקשורת עם השרת מצד הלקוח. אחראית גם על חלון ממשק המשתמש של הלקוח המאפשר לבצע את שלבי ההתחברות ולאחר מכן להשתתף במשחק.

**תכונות המחלקה:**

* User – אובייקט המחזיק את כל המידע על הלקוח.
* Game – אובייקט המחזיק את המצב הנוכחי של המשחק.
* cardPictures\_ – רשימה של אובייקטים מסוג PictureBox המנהלים את התצודה של קלפי המשחק.
* logger\_ – אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה.

**פעולות המחלקה:**

public Client()

**טענת כניסה:** אין.

**טענת יציאה:** פעולת הבנאי של המחלקה. מאתחלת את תצוגת ממשק המשתמש ואת ה-logger\_.

* public void log(string text)

**טענת כניסה:** מחרוזת המייצגת את ההודעה שתכתב לקובץ.

**טענת יציאה:** רושמת את ההודעה לקובץ. אנה מחזירה דבר.

* private void LoadAsync(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** הפעלה של התהליכון המנהל את התקשורת של הלקוח עם השרת. אינה מחזירה דבר.

* public void StartGame(List<Card> cardList)

**טענת כניסה:** רשימת הקלפים שמקבל המשצמש בתחילת המשחק.

**טענת יציאה:** יצירת האובייקט המייצג את מצב המשחק ואתחול ממשק המשתמש לקראת התחלת המשחק. אינה מחזירה דבר.

* public void EndGame(string winner, string loser)

**טענת כניסה:** שם המשתמש של המנצח והמפסיד במשחק.

**טענת יציאה:** מודיעה בחלון העידכונים על סיום המשחק וזהות המנצח והמפסיד ומאתחלת את ממשק המשתמש לקראת משחק נוסף.

* public void InterruptGame(string user)

**טענת כניסה:** שם המשתמש שהפסיק את המשחק.

**טענת יציאה:** מודיע בחלון העידכונים על הפסקת המשחק ומאתחלת את ממשק המשתמש לקראת משחק נוסף.

* public void ExitLobby()

**טענת כניסה:** אין.

**טענת יציאה:** עידכון ממשק המשתמש ומצב התכונות של המחלקה בעקבות יציאת הלקוח מזירת המשחק. אינה מחזירה דבר.

* private void ResetForNewGame()

**טענת כניסה:** אין.

**טענת יציאה:** איתחול ממשק המשתמש לקראת התחלה של משחק חדש. אינה מחזירה דבר.

* public void ActivateNotYourTurnMode()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון מצב ממשק המשתמש כאשר זה אינו תורו של המשתמש לשחק. אינה מחזירה דבר.

* public void ActivateMakeOfferMode(List<int> rejectedCardIndices)

**טענת כניסה:** רשימת האינדקסים של הקלפים שכבר הוצעו ונידחו במהלך התור הנוכחי.

**טענת יציאה:** עידכון מצב ממשק המשתמש לכך שהמשתמש עומד לבחור קלף אותו יציע לשחקן הבא בתור. אינה מחזירה דבר.

* public void ActivateAwaitResponseMode()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון ממשק המשתמש כדי ליידע את המשתמש שעליו להמתין לתגובה על ההצעה שנתן. אינה מחזירה דבר.

* public void ActivateAwaitOfferMode()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון ממשק המשתמש כדי ליידע את המשתמש שעליו להמתין להצעה של קלף. אינה מחזירה דבר.

* public void ActivateNeedToReplyMode()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון ממשק המשתמש כדי ליידע את המשתמש שעליו להשיב להצעה שקיבל. אינה מחזירה דבר.

* private void ConnectButton\_Click(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** נקראת בעקבות לחיצת המשתמש על כפתור ההתחברות/התנתקות. במצב התחברות נפתח חלון ההתחברות ובעקבותיו חלון רישום משתמש חדש אם צריך. במצב התנתקות נשלחת הודעה לשרת שהמשתמש מבקש להתנתק ומצב ממשק המשתמש מעודכן כך שניתן לבצע התחברות חדשה. אינה מחזירה דבר.

* private void JoinLobbyButton\_Click(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** נקראת בעקבות לחיצת המשתמש על כפתור ההצטרפות/יציאה מזירת משחק. במצב ההצטרפות נפתח חלון ההצטרפות ובעקבותיו חלון יצירת שירת משחק חדשה אם צריך. במצב יציאה נשלחת הודעה לשרת שהמשתמש מבקש לצאת מהזירה ומצב ממשק המשתמש מעודכן כך שניתן לבצע הצטרפות חדשה. אינה מחזירה דבר..

* private bool ClientLoop(CancellationToken token)

**טענת כניסה:** אובייקט המאפשר להפסיק את ריצת הלולאה מבחוץ.

**טענת יציאה:** מייצרת חיבור מאובטח בין הלקוח והשרת ולאחר מכן מאזינה להודעות מהשרת ומעדכנת את מצב ממשק המשתמש והמשחק בהתאם. מחזיר משתנה בוליאני המציין האם הייתה תקלה בזמן הריצה. אינה מחזירה דבר.

* public void LoginUser(bool success, string? reason)

**טענת כניסה:** משתנה בוליאני המציין האם פעולת ההתחברות הצליחה ומחרוזת המציינת את הסיבה לכישלון.

**טענת יציאה:** מעדכנת את ממשק המשתמש בעקבות תוצאות פעולת ההתחברות. אינה מחזירה דבר.

* private void PrepareForLogin()

**טענת כניסה:** אין.

**טענת יציאה:** מעדכנת את ממשק המשתמש לקראת ביצוע של התחברות על ידי המשתמש. אינה מחזירה דבר.

* private void PrepareForJoinLobby()

**טענת כניסה:** אין.

**טענת יציאה:** מעדכנת את ממשק המשתמש לקראת ביצוע של הצטרפות לזירת משחק על ידי המשתמש. אינה מחזירה דבר.

* public void UserJoinLobby(bool success, bool isHost, string? reason)

**טענת כניסה:** משתנה בוליאני המציין האם פעולת ההצטרפות לזירה הצליחה ומחרוזת המציינת את הסיבה לכישלון.

**טענת יציאה:** מעדכנת את ממשק המשתמש בעקבות תוצאות פעולת ההתחברות. אינה מחזירה דבר.

* private bool EstablishConnection(TcpClient client, CancellationToken token)

**טענת כניסה:** אובייקט מסוג TcpClient המייצג את החיבור של הלקוח עם השרת ואובייקט המאפשר להפסיק ריצת תהליכון מבחוץ.

**טענת יציאה:** ניהול התקשורת עם השרת כדי לקבל את מפתח ההצפנה ומספר הזיהוי של הלקוח. מחזירה משתנה בוליאני המציין אם הפעולה הצליחה.

* internal void EnableStartGame(bool canStart)

**טענת כניסה:** משתנה בוליאני המציין אם אפשר להתחיל משחק חדש.

**טענת יציאה:** מעדכן את ממשק המשתמש כדי לאפשר/למנוע ממנהל הזירה להתחיל משחק חדש. אינה מחזירה דבר.

* public void DisplayCards(List<Card> cards)

**טענת כניסה:** רשימת הקלפים שכרגע מחזיק הלקוח.

**טענת יציאה:** מציגה את הקלפים בחלון המשחק. אינה מחזירה דבר.

* public void ClearCardDisplay()

**טענת כניסה:** אין.

**טענת יציאה:** מנקה את תצוגת הקלפים כך שלא מוצג דבר. אינה מחזירה דבר.

* public void RemoveOfferedCard()

**טענת כניסה:** אין.

**טענת יציאה:** מעדכנת את מצב המשחק והתצוגה בעקבות מסירת קלף. אינה מחזירה דבר.

* public void AddAcceptedCard(Card card)

**טענת כניסה:** הקלף אותו קיבל המשתמש.

**טענת יציאה:** מעדכנת את מצב המשחק והתצוגה בעקבות קבלת הקלף. אינה מחזירה דבר.

* public void DisableRejectedCard(int index)

**טענת כניסה:** מספרו הסידורי של הקלף שנדחה.

**טענת יציאה:** מעדכנת את התצוגה של הקלף שנדחה ומונעת בחירתו שוב באותו התור. אינה מחזירה דבר.

* void UpdateConnectionStatusConnected()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון מצב החיבור המוצג בחלון למצב מחובר. אינה מחזירה דבר.

* void UpdateConnectionStatusInLobby()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון מצב החיבור המוצג בחלון למצב בתוך זירה. אינה מחזירה דבר.

* void UpdateConnectionStatusNotConnected()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון מצב החיבור המוצג בחלון למצב לא מחובר. אינה מחזירה דבר.

* void UpdateConnectionStatusInGame()

**טענת כניסה:** אין.

**טענת יציאה:** עדכון מצב החיבור המוצג בחלון למצב בתוך משחק. אינה מחזירה דבר.

* UpdateConnectionStatus

**טענת כניסה:** מחרוזת המייצגת את הטקסט שיוצג ושם הצבע בו יוצג הטקסט..

**טענת יציאה:** עדכון מצב החיבור המוצג בחלון. אינה מחזירה דבר.

* public void AppendToChat(string text, bool bold, bool endLine)

**טענת כניסה:** המחרוזת שתוצג בחלון הצ'אט, משתנה בוליאני הקובע האם תוצג בגופן מודגש ומשתנה בוליאני הקובע האם לרדת שורה בסוף הטקסט.

**טענת יציאה:** מציגה את הטקסט בחלון הצ'אט של הלקוח. אינה מחזירה דבר.

* public void AppendToGameLog(string text)

**טענת כניסה:** מחרוזת המייצגת את הטקסט שיוצג בחלון העידכונים.

**טענת יציאה:** מציגה את המחרוזת בחלון העידכונים. אינה מחזירה דבר.

* public void UpdateStatus(string text)

**טענת כניסה:** מחרוזת המייצגת את הטקסט שיוצג בחלון הסטאטוס.

**טענת יציאה:** מציגה את המחרוזת בחלון הסטאטוס. אינה מחזירה דבר.

* private void WriteToServer(CommMessage message)

**טענת כניסה:** אובייקט המייצג את ההודעה שתשלח לשרת.

**טענת יציאה:** שליחת ההודעה לשרת. אינה מחזירה דבר.

* private void ChatInputBox\_KeyUp(object sender, KeyEventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** מגיבה ללחיצה על מקש Enter בחלון כתיבת הצ'אט. שולחת את הטקסט שהוקלד לחלונות הצ'אט של כל האורחים בזירה. אינה מחזירה דבר.

* private void OfferAcceptButton\_Click(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** שליחת הודעה לשרת שהשחקן מקבל את הקלף שהוצע לו. אינה מחזירה דבר.

* private void OfferRejectButton\_Click(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** שליחת הודעה לשרת שהשחקן דוחה את הקלף שהוצע לו. אינה מחזירה דבר.

* private void CardPicture\_DoubleClick(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** בחירת הקלף שיוצע לשחקן הבא בתור על ידי הקלקה כפולה. ההצעה נשלחת בהודעה לשרת ומצב המשחק מתעדכן. הקלף הנבחר מסומן במסגרת אדומה. אינה מחזירה דבר.

* private void CardPicture\_Paint(object sender, PaintEventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** מציירת מסגרת אדומה מסביב לקלף שנבחר. אינה מחזירה דבר.

* private void StartGameButton\_Click(object sender, EventArgs e)

**טענת כניסה:** האובייקט שקרא לפעולה, פרמטרים אופציונלים נוספים.

**טענת יציאה:** מגיבה ללחיצה על כפתור ההתחל/הפסק משחק. נשלחת הודעה מתאימה לשרת. אינה מחזירה דבר.

* public void ShutDown()

**טענת כניסה:** אין.

**טענת יציאה:** סגירת החיבור עם השרת והפסקת הלולאה הממתינה להודעות מהשרת. אינה מחזירה דבר.

**שם**: MessageHandler

**קובץ**: MessageHandler.cs

**תפקיד**: טיפול בכל ההודעות שהשרת שולח ללקוח.

**תכונות המחלקה:**

* client\_ – הפנייה לאובייקט ה-Client.
* user\_ – הפנייה לאובייקט המחזיק מידע של הלקוח

**פעולות המחלקה:**

* public MessageHandler(Client client, UserData user)

**טענת כניסה:** הפנייה לאובייקט ה-Client והפנייה לאובייקט המחזיק מידע של הלקוח.

**טענת יציאה:** פעולת הבנאי של המחלקה. מאתחלת את התכונות של האובייקט.

* private void log(string text)

**טענת כניסה:** מחרוזת המייצגת את הטקסט הנכתב לקובץ.

**טענת יציאה:** כותבת את הטקסט לקובץ התיעוד המשותף. אינה מחזירה דבר.

* public void Handle(CommMessage msg)

**טענת כניסה:** אובייקט המייצג את ההודעה שהתקבלה מהשרת.

**טענת יציאה:** בדיקת סוג ההודעה וטיפול פרטני בכל אחד מסוגי ההודעות שהלקוח יכול לקבל.

**שם**: GameState

**קובץ**: GameState.cs

**תפקיד**: מייצגת את מצב המשחק בכל רגע מנקודת מבטו של השחקן. מצב הקלפים שבידו, כמה הצעות סורבו וכד.

**תכונות המחלקה:**

* client\_ – הפנייה לאובייקט ה-Client.
* Cards – רשימה של הקלפים אותם מחזיק השחקן כרגע
* logger\_ – אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה
* OfferedCardIndex – האינדקס של הקלף אותו הציע המשתמש. יכול להיות גם null.
* OfferedCard – הקלף אותו הציע המשתמש
* Cards – רשימה
* RejectedCardIndices – רשימה המכילה את האינקסים של הקלפים שהוצעו ונדחו
* ReceiverName – מחרוזת המחזיקה את שם המשתמש של השחקן המקבל ממני הצעות, אם יש כזה
* GiverName – מחרוזת המחזיקה את שם המשתמש של השחקן הנותן לי הצעות, אם יש כזה
* ReceivedCard – הקלף שהוצע לי בהצעה האחרונה.
* PlayerMode – מצב המשחק הנוכחי. יכול לקבל אחד מחמישה ערכים: NotMyTurn, MakeOffer, WaitForReponse, AwaitOffer או NeedToReply.

**פעולות המחלקה:**

* private void OnUpdatePlayerMode()

**טענת כניסה:** אין.

**טענת יציאה:** מבצעת עידכון של תכונות המחלקה ומצב ממשק המשתמש בהתאם לשינוי במצב המשחק. אינה מחזירה דבר.

* internal GameState(Client client, Logger logger, List<Card> cardList)

**טענת כניסה:** הפנייה לאובייקט ה-Client, הפנייה לאובייקט ה-Logger ורשימת הקלפים ההתחלתית של השחקן.

**טענת יציאה:** מאתחלת את תכונות המחלקה. אינה מחזירה דבר.

* internal void removeCard()

**טענת כניסה:** אין.

**טענת יציאה:** מוחקת את הקלף שנמסר לשחקן השני מרשימת הקלפים שלי. אינה מחזירה דבר.

* internal void AppendCard(Card card)

**טענת כניסה:** אין.

**טענת יציאה:** מוסיפה טת הקלף שניתן לי לשרשימת הקלפים שלי. אינה מחזירה דבר.

**שם**: UserData

**קובץ**: UserData.cs

**תפקיד**: מייצגת את מצב המשחק בכל רגע מנקודת מבטו של השחקן. מצב הקלפים שבידו, כמה הצעות סורבו וכד.

**תכונות המחלקה:**

* Id – מספר המייצג את הזיהוי הייחודי של הלקוח.
* Client – הפנייה לאובייקט ה-TcpClient של הלקוח
* nws\_ – הפנייה לאובייקט ה- NetworkStreamשל הלקוח
* Reader – האובייקט המאפשר קריאה של הודעות מהשרת
* Writer – האובייקט המאפשר כתיבה של הודעות לשרת
* Logger – אובייקט מסוג Logger המשמש לרישום בזמן הבדיקות בקובץ משותף של כל הפעולות המתבצעות בתוך המחלקה
* Name – מחרוזת המייצגת את שם המשתמש של הלקוח.
* IsHost – משתנה בוליאני המציין אם הלקוח הוא המנהל של זירת המשחק
* Connected – משתנה בוליאני המציין אם הלקוח הוא התחבר לשרת
* InLobby – משתנה בוליאני המציין אם הלקוח נמצא בזירת משחק

**פעולות המחלקה:**

* UserData(int id, TcpClient socket, NetworkStream nws, string aesKey, Logger logger)

**טענת כניסה:** מספר הזיהוי של הלקוח, הפנייה לאובייקט ה-TcpClient של הלקוח, הפנייה לאובייקט ה-NetworkStream של הלקוח, מחרוזת המייצגת את קוד ההתפנה הסימטרי מסוג AES ואובייקט Logger המשמש לתיעוד בקובץ משותף.

**טענת יציאה:** זוהי פעולת הבנאי של המחלקה. מאתחלת את התכונות של המחלקה.

* public override string ToString()

**טענת כניסה:** אין.

**טענת יציאה:** מייצרת ייצוג במחרוזת של האובייקט. מחזירה את המחרוזתץ

* public void ResetName()

**טענת כניסה:** אין.

**טענת יציאה:** מאפס את שם המשתמש של הלקוח.

* public void ShutDown()

**טענת כניסה:** אין.

**טענת יציאה:** סוגרת את החיבור לשרת. אינה מחזירה דבר.

מודול **Shared**

פונקציונאליות המשותפת גם לאפליקציית הלקוח וגם לאפליקציית השרת. ארוז בצורת dll (Dynamically Linked Library) כך שתוכל לרוץ גם בצד השרת וגם בצד הלקוח. כוללת הגדרות של ההודעות המשמשות לתקשורת בין השרת ללקוח, פעולות ההצפנה והגיבוב בהן משתמשות האפליקציות, המחלקה שמאפשרת שמירה של תיעוד הרצה לצורך ניפוי באגים, מחלקות לכתיבה וקריאה לשקעים, ועוד. בתוך מודול זה ממומשות המחלקות הבאות:

**שם**: CommMessage

**קובץ**: CommMessage.cs

**תפקיד**: זוהי מחלקת הבסיס ממנה יורשות כל הודעות התקשורת הנשלחות בין השרת והלקוח. ניתן להמיר כל הודעה למחרוזת אותה ניתן לשלוח ברשת באופן גלוי או מוצפן, וניתן להמיר כל מחרוזת המגיעה ברשת חזרה להודעה ממנה נוצרה, כולל פענוח ההצפנה אם יש צורך.

**תכונות המחלקה:**

* Type – תכונה מופשטת אשר תמומש על ידי כל המחלקות היורשות. מייצגת את סוג ההודעה. מוגדרים סוגי ההודעות הבאים:
  + הודעות הנשלחות מהשרת ללקוח: DealCards, TakeCard, AcceptCard, MakeOffer, ReceiveOffer, NotYourTurn, ReceiveChat, Login, JoinLobby, GameLog, AesKey, Register, CreateLobby, CanStartGame, AnnounceWinner, LobbyClosing.
  + הודעות הנשלחות מהלקוח לשרת: BroadcastChat, OfferCard, LoginRequest, RegisterRequest, CreateLobbyRequest, JoinLobbyRequest, ExitLobbyRequest, StartGame, Logout.
  + הודעות הנשלחות בשני הכיוונים: ResponseToOffer, InterruptGame, PublicKey, Encrypted.
  + הודעות שגיאה (אינן נשלחות אך משמשות לייצוג בעיות בתקשורת ובתהליך היצירה או הניתוח של הודעות): ParseError, UnrecognizedMessageTypeError, MessageBodyError, EncryptionError, CommunicationError.
* Text – תכונה וירטואלית הממומשת באופן חלקי במחלקת הבסיס וכל מחלקה יטרשת אחראית להשלים את המימוש בהתאם לצרכיה. מיצגת את צורת המחרוזת של ההודעה.

**פעולות המחלקה:**

* public bool IsError()

**טענת כניסה:** אין.

**טענת יציאה:** בודקת אם ההודעה היא מסוג הודעת שגיאה ומחזירה ערך בוליאני בהתאם.

* public string EncryptedText(string aesKey)

**טענת כניסה:** מחרוזת המייצגת את מפתח ההצפנה מסוג AES.

**טענת יציאה:** לוקחת את הייצוג של ההודעה כמחרוזת ומצפינה אותה בעזר המפתח שקיבלה. המחרוזת נארזת באובייקט של הודעה מוצפנת המומר למחרוזת אותה ניתן לשלוח לצד דרך הרשת. מחזירה את המחרוזת המייצגת את ההודעה המוצפנת.

* public static CommMessage FromText(string rawMessage, string? aesKey = null)

**טענת כניסה:** מחרוזת המייצגת את ההודעה, ומחרוזת המייצגת מפתח הצפנה אופציונלי מסוג AES.

**טענת יציאה:** זוהי פעולה סטטית של המחלקה היודעת לקחת הודעה שהתקבלה כמחרוזת, לפענח את ההצפנה אם צריך, לזהות את סוג ההודעה ולנתח אותה בהתאם. מחזירה אובייקט של המחלקה המתאימה לסוג ההודעה או אובייקט מסוג הודעת השגיאה המתאימה.

המחלקות היורשות מ-CommMessage ומייצגות הודעות ספציפיות הנשלחות במערכת הינן בעלות מבנה דומה שיפורט כאן:

**תכונות המחלקה:**

* type\_ – תכונה סטאטית המייצגת את סוג ההודעה. יכולה להיות אחד מדוגי ההודעות שפורטו לעיל בתיאור של מחלקת האב.
* Text – דריסה (override) של התכונה במחלקת האב, המרחיבה את הייצוג הבסיסי של ההודעה בתצורת מחרוזת שנוצרה על ידי מחלקת האב כך שתתאים לצרכים של המחלקה היורשת.
* תכונות נוספות הייחודיות למחלקה היורשת הספציפית (יפורטו בהמשך)

**פעולות המחלקה:**

* public static CommMessage FromText(string msgBody)

**טענת כניסה:** מחרוזת המייצגת את גוף ההודעה שהתקבלה.

**טענת יציאה:** מנתחת את גוף ההודעה בהתאם למבנה הצפוי של הודעות מסוג זה. מחזירה אובייקט מטיפוס מחלקת האב המייצג את אובייקט ההודעה שנוצר או אובייקט של הודעת שגיאה אם פעולת הניתוח נכשלה.

* static public XXXMessage Create(TypeA a, TypeB b ...)

**טענת כניסה:** ערכים ספציפיים לטיפוס ההודעה.

**טענת יציאה:** מייצרת אוביקט מהטיפוס של המחלקה המחזיק את הערכים שסופקו ומחזירה אותו

להלן פירוט מחלקות ההודעות היורשות ממחלקת CommMessage בהן משתמשת המערכת:

הודעות מהשרת ללקוחות

**שם**: DealCardsClientMessage

**תפקיד**: שליחת הקלפים שכל שחקן מקבל בתחילת המשחק.

**תכונות המחלקה:**

CardList – רשימה של הקלפים אותם מקבל השחקן

**שם**: CardMessage

**תפקיד**: מחלקה מופשטת המשמשת כבסיס לשתי הודעות דומות העוסקות בהצעה של קלף לאחד השחקנים.

**תכונות המחלקה:**

Card – הקלף המוצע.

**שם**: TakeCardClientMessage

**תפקיד**: יורשת ממחלקת CardMessage. נשלחת במצב בו השחקן המקבל חייב לקחת את הקלף שהוצע לו.

**תכונות המחלקה:** אין

**שם**: AcceptCardClientMessage

**תפקיד**: יורשת ממחלקת CardMessage. נשלחת במצב בו השחקן המקבל רשאי לסרב לקחת את הקלף שהוצע לו.

**תכונות המחלקה:** אין

**שם**: MakeOfferClientMessage

**תפקיד**: ליידע את השחקן שהגיע תורו להציע קלף לשחקן הבא אחריו.

**תכונות המחלקה:**

Num – מספר ההצעה (ראשונהת שנייה או שלישית)

Receiver – שם המשתמש של השחקן המקבל את ההצעה.

**שם**: ReceiveOfferClientMessage

**תפקיד**: ליידע שחקן שהגיעתורו לקבל הצעה מהשחקן שלפניו.

**תכונות המחלקה:**

Giver – שם המשתמש של השחקן שנותן את ההצעה.

**שם**: NotYourTurnClientMessage

**תפקיד**: ליידע שחקן שהפעולה שביצע אינה חוקית כי לא הגיע התור שלו.

**תכונות המחלקה:** אין

**שם**: ReceiveChatClientMessage

**תפקיד**: לשלוח ללקוח הודעת צ'אט שנשלחה על ידי אחד השחקנים האחרים.

**תכונות המחלקה:**

Sender – שם המשתמש של שולח ההודעה

Msg – מחרוזת המחילה את תוכן הודעת הצ'אט.

**שם**: ResponseMessage

**תפקיד**: מחלקה מופשטת המשמשת כבסיס לארבע הודעות דומות העוסקות בתשובת השרת לפעולות שביצע הלקוח.

**תכונות המחלקה:**

Success – משתנה בוליאני המייצג את תוצאת הפעולה: הצלחה או כישלון

Reason – מחרוזת אופציונלית המכילה את הסיבה לכישלון (אם הפעולה נכשלה)

**שם**: LoginResponseMessage

**תפקיד**: תשובת השרת לבקשת החיבור של הלקוח. יורשת ממחלקת ResponseMessage.

**תכונות המחלקה:** אין

**שם**: RegisterResponseMessage

**תפקיד**: תשובת השרת לבקשת רישום משתמש חדש של הלקוח. יורשת ממחלקת ResponseMessage.

**תכונות המחלקה:** אין

**שם**: CreateLobbyResponseMessage

**תפקיד**: תשובת השרת לבקשת יצירת זירת משחק חדשה של הלקוח. יורשת ממחלקת ResponseMessage.

**תכונות המחלקה:** אין

**שם**: JoinLobbyResponseMessage

**תפקיד**: תשובת השרת לבקשת הצטרפות לזירת משחק של הלקוח. יורשת ממחלקת ResponseMessage.

**תכונות המחלקה:** אין

**שם**: CanStartGameClientMessage

**תפקיד**: להודיע למנהל הזירה אם יש או אין מספיק שחקנים כדי להתחיל משחק חדש.

**תכונות המחלקה:**

CanStart – משתנה בוליאני המציין אם יש מספיק שחקנים כדי להתחיל משחק

**שם**: AnnounceWinnerClientMessage

**תפקיד**: הודעה לכל השחקנים כי המשחק הסתיים. ההודעה כוללת את שם המשתמש של המנצח ושל המפסיד

**תכונות המחלקה:**

Winner – מחרוזת המייצגת את שם המשתמש של המנצח

Loser – מחרוזת המייצגת את שם המשתמש של המפסיד

**שם**: LobbyClosingClientMessage

**תפקיד**: ליידע את האורחים בזירת המשחק כי היא נסגרת.

**תכונות המחלקה:** אין

**שם**: GameLogClientMessage

**תפקיד**: שליחת הודעה לכל השחקנים אשר תוצג בחלות העידכונים שלהם.

**תכונות המחלקה:**

Msg – מחרוזת המייצגת את תוכן ההודעה שתוצג

**שם**: AesKeyMessage

**תפקיד**: שליחת מפתח ההצפנה הסימטרי מסוג AES ללקוח.

**תכונות המחלקה:**

Key – מחרוזת המייצגת את המפתח שנשלח

הודעות מהלקוחות לשרת

**שם**: BroadcastChatServerMessage

**תפקיד**: שליחת תוכן הודעת הצ'אט לשרת כדי שיפיץ לכל האורחים בזירה.

**תכונות המחלקה:**

Msg – מחרוזת המייצגת את תוכן ההודעה הנשלחת.

**שם**: OfferCardServerMessage

**תפקיד**: ליידע שחקן שתורו להציע קלף לשחקן הבא אחריו בתור.

**תכונות המחלקה:** אין.

**שם**: LoginRequestServerMessage

**תפקיד**: שליחת בקשת התחברות לשרת.

**תכונות המחלקה:**

UserName – מחרוזת המייצגת את שם המשתמש.

Password – צחרוזת המייצגת את הסיסמא.

**שם**: RegisterRequestServerMessage

**תפקיד**: שליחת בקשה לרישום משתמש חדש.

**תכונות המחלקה:**

UserName – מחרוזת המייצגת את שם המשתמש.

Password – צחרוזת המייצגת את הסיסמא

Email – מחרוזת המייצגת את כתובת הדואל של המשתמש

**שם**: LobbyRequestServerMessage

**תפקיד**: מחלקה מופשטת המשמשת כבסיס לשתי הודעות דומות העוסקות ביצירה והצטרפות לזירת משחק.

**תכונות המחלקה:**

Name – מחרוזת המייצגת את שם הזירה

EntryCode – מחרוזת המייצגת את קוד הכניסה לזירה

**שם**: CreateLobbyRequestServerMessage

**תפקיד**: בקשה מהשרת ליצור זירת משחק חדשה. יורשת ממחלקת LobbyRequestServerMessage.

**תכונות המחלקה:** אין.

**שם**: JoinLobbyRequestServerMessage

**תפקיד**: בקשה מהשרת להצטרף לזירת משחק. יורשת ממחלקת LobbyRequestServerMessage.

**תכונות המחלקה:** אין

**שם**: ExitLobbyRequestServerMessage

**תפקיד**: בקשה של הלקוח מהשרת לצאת מזירת המשחק בה הוא נמצא.

**תכונות המחלקה:**

UserName – מחרוזת המייצגת את שם המשתמש של הלקוח

**שם**: StartGameServerMessage

**תפקיד**: בקשה של מנהל הזירה מהשרת להתחיל משחק חדש.

**תכונות המחלקה:** אין.

**שם**: LogoutServerMessage

**תפקיד**: בקשה של הלקוח מהשרת להתנתק.

**תכונות המחלקה:** אין

הודעות הנשלחות בשני הכיוונים

**שם**: ResponseToOfferMessage

**תפקיד**: הודעה על קבלה או דחייה של הקלף שהוצע. נשלחת מהשחקן שקיבל את ההצעה לשרת, המעביר אותה לשחקן שנתן את ההצעה

**תכונות המחלקה:**

Accept – משתנה בוליאני המייצג את התשובה להצעה: קבלה או דחייה.

**שם**: InterruptGameMessage

**תפקיד**: הודעה של לקוח המבקש להפסיק את המשחק. נשלח מהלקוח לשרת, אשר מיידע את שאר השחקנים.

**תכונות המחלקה:**

UserName – מחרוזת המייצגת את שם המשתמש המבקש לסיים את המשחק.

**שם**: PublicKeyMessage

**תפקיד**: הודעה הנשלחת על ידי השרת או הלקוח כדי ליידע את הצד השני בדבר מפתח ההצפנה הציבורי שלו.

**תכונות המחלקה:**

Key – מחרוזת המייצגת את מפתח ההצפנה הציבורי.

Id – מספר הזיהוי של הלקוח

**שם**: EncryptedMessage

**תפקיד**: שליחה של הודעה מוצפנת. הודעה זו מכילה אחת משאר ההודעות אשר עברה הצפנה. בשלב ניתוח ההודעה המחרוזת עוברת פיענוח כדי לקבל חזרה את ההודעה המקורית.

**תכונות המחלקה:**

Content – תוכן ההודעה המקורית

הודעות שגיאה

**שם**: ParseErrorMessage

**תפקיד**: שגיאה בניתוח הודעה שהתקבלה.

**תכונות המחלקה:**

RawMessage – מחרוזת המייצגת את ההודעה המקורית אשר ניתוחה נכשל

**שם**: UnrecognizedMessageTypeErrorMessage

**תפקיד**: שגיאה בסוג ההודעה. הסוג שנשלח אינו מוכר.

**תכונות המחלקה:**

UnknownType – מחרוזת המייצגת את סוג ההודעה שהתקבלה

MsgBody – מחרוזת המייצגת את גוף ההודעה שהתקבלה

**שם**: MessageBodyErrorMessage

**תפקיד**: שגיאה בניתוח גוף ההודעה.

**תכונות המחלקה:**

MsgType – מחרוזת המייצגת את סוג ההודעה שהתקבלה

MsgBody – מחרוזת המייצגת את גוף ההודעה שהתקבלה

**שם**: EncryptionErrorMessage

**תפקיד**: שגיאה בפיענוח הודעה מוצפנת שהתקבלה.

**תכונות המחלקה:**

Reason – מחרוזת המייצגת את הסיבה לשגיאה

**שם**: CommunicationErrorMessage

**תפקיד**: שגיאה במהלך השליחה או הקבלה של ההודעה.

**תכונות המחלקה:**

Error – מחרוזת המייצגת את הודעת השגיאה.

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

* ToString

**טענת כניסה:** .

**טענת יציאה:** .

**רפלקציה**

התחלתי ללמוד #C בחודשים הראשונים של השפה, עד אז לא היה לי הרבה ניסיון עם שפת תכנות זו אבל כן ידעתי ג'אווה שהיא שפת תכנות דומה ל-#C. החלטתי להשתמש בסביבית התכנות של Visual Studio שהייתה הפלאטפורמה שהשתמשתי בה כבר בעבר. באמצעות WinForms למדתי איך ליצור את ה-UI ואיך לעשצב אותו. כאשר הייתי לא בטוח מה לעשות או איך להתקדם נעזרתי בסרטונים ביוטיוב, באתרים באינטרנט ובדוגמאות של אחרים, כל המקורות הללו איפשרו לי למצוא פיתרון כל בעיה שנתקלתי בה. אתגר שהייתי צריך להתמודד איתו היה להצפין את המידע של המשתמשים. חיפשתי באינטרנט מה היא דרך טובה לאבטח מידע והדרך שקבילתי הייתה הצפנה משולבת שהתמשה ב-RSA ו-AES בשביל להעביר מידע, בגיבוב בשביל לשמור עליו ולאחר מכן גם למדתי איך להוסיף Salt ו-Pepper.

הלמידה בעיקר כללה אותי יושב מול המחשב ומחפש דרכים לגרום לפרויקט שלי לעבוד ולאחר מכן לשפר אותו. דוגמא למקרה כזה היא שבתהליך של התחברות או הרשמות של משתמש קיים תהליכון שמטפל בזה וקולט את הבקשות של המשתמש. קיים תהליכון דומה לזה שמשמש לכניסה לזירות משחק וקיים עוד תהלכיון שמטפל במשחק העצמו. הבעיה פה היא שאחרי שחקן עובר מתהליכון לתהליכון הוא לא יכול לחזור אחורה וכשאר הוספתי את האפשרות לצאת מזירה הגעתי למצב שבו המשתמש יכול לבקש להצטרף לזירת משחק אחרת אבל השרת עדיין חושב שהוא בתוך מחשק. כתוצאה מכך הייתי צריך לשים את כל התהליכונים ביחד ולאפשר וזה מאפשר למשתמש לזוז קדימה ואחורה בין תחנות.

נהינתי מאוד מכל התהליך של יצירת הפרויקט. הלמידה, התכנות והעיצוב של הפריוקט, אלה היו אתגרים שהתגברתי עליהם בדרך ליצרית הפרויקט. אני גאה בתוצר הסופי ומקווה שאני אוכל ליצור עוד משחקים כמוהו. את הכל הכלים שלמדתי בפרויקט הזה אקח איתי ואנסה שהפרויקטים הבאים שלי יהיו טובים יותר.

בראייה לאחור הייתי מיישם שונה את זירות המשחק. כרגע הדרך היחידה להצטרף לזירה היא רק באמצעות ידיע של השם שלה וקוד הכניסה שלה ובגלל לא ניתן לדעת על הקיום של הזירה ללא דרכי תיקשורת חיצוניים. הייתי מוסיף מסך שבו מתמשים יכולים לראות אילו זירות פתוחות ומה מצבן, כלומר האם הן באמצע משחק או לא. ככה תוכל ללחוץ עליהם בישביל להצטרף במקום להקליד את הפרטים שלהם.

זה לא הדבר היחיד שהייתי מוסיף. אם היה לי עוד זמן הייתי גם מוסיף:

* שהמשתמש יוכל לבחור אילו תמונות יהיו על הקלפים שלהם וככה להפוך את חווית המחשק ליותר מיוחדת ואישית.
* צ'אט גם מחוץ לזירות שיאפשר לשחקנים לתאם באיזו זירה הם רוצים לשחק או מי זה שינהל את הזירה.
* תמונות של המשתמשים שהם יכולים להלביש על עצמם וככה בזמן המשחק כשאר תקבל מהם הודעה בצ'אט אתה גם תראה את תמונת הפרופיל שלהם.

יש עוד הרבה דברים שהייתי מוסיף אבל אילו הדברים שאני חושב שיוכלים לשפר מאוד את המשחק שלי.

אני רוצה להודות למורה שלי יהודה אור, שעזר לי עם בעיות בפרויקט ובעיות הבנה.