

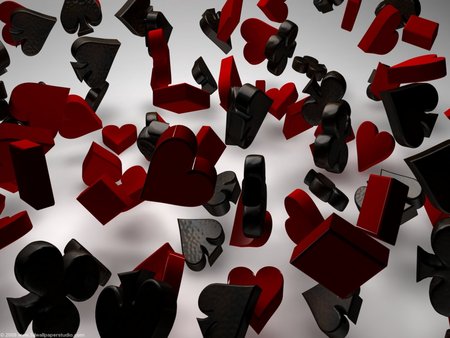
**מגישה:** קטיה בוצמן

**מספר זהות:** 317573657

**מדריך:** אנטולי פיימר

**שנת לימודים:** 2011, תשע"א

**בית ספר:** מקיף ג' אשדוד



תוכן עניינים

[הקדמה: 4](#_Toc292561681)

[תקציר הפרויקט 4](#_Toc292561682)

[מטרת המשחק 4](#_Toc292561683)

[קהל היעד של המשחק 4](#_Toc292561684)

[דברי המפתחת- מטרות ויעדים 5](#_Toc292561685)

[מדריך למפתח 6](#_Toc292561686)

[הכלים לכתיבת הממשק 6](#_Toc292561687)

[דרישות חומרה: 6](#_Toc292561688)

[תיאור המערכת הקיימת 7](#_Toc292561689)

[תיאור משתמשי המערכת 7](#_Toc292561690)

[מבנה השרת 8](#_Toc292561691)

[מחלקות הקשורות לקלפים: 8](#_Toc292561692)

[מחלקות הקשורות לתקשורת: 11](#_Toc292561693)

[מבנה הלקוח 21](#_Toc292561694)

[מחלקות הקשורות לקלפים: 21](#_Toc292561695)

[מחלקות הקשורות למשחק: 23](#_Toc292561696)

[מחלקות הקשורות לתקשורת ולחלונות המשחק 28](#_Toc292561697)

[הגראפיקה במשחק: 39](#_Toc292561698)

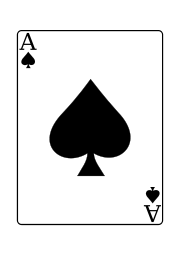
[בעיות לא פתורות והצעות לשיפור 40](#_Toc292561699)

[רפלקציה אישית ותודות 41](#_Toc292561700)

[הוראות לפתיחת הממשק 42](#_Toc292561701)

[ביבליוגרפיה 42](#_Toc292561702)

[נספחים: 43](#_Toc292561703)



הקדמה:

### תקציר הפרויקט

הפרויקט שהכנתי הינו משחק המבוסס על משחק קלפים ישראלי בשם יניב, הנפוץ בעיקר בקרב מטיילים, הן בישראל והן בעולם. המערכת אותה בניתי מאפשרת לשני שחקנים לשחק במשחק אחד נגד השני דרך שרת.

### מטרת המשחק

המטרה הגדולה של המשחק כולו היא לא להגיע לסכום נקודות גבוהה מ-200. המשחק מורכב מסבבים אשר בסוף כל סבב השחקנים מקבלים נקודות, בהתאם למהלך הסבב.

בתחילת המשחק כל שחקן מקבל 5 קלפים מהקופה. בתורו, על השחקן לבחור קלף או כמה קלפים (בתנאי שיהיו מאותו סוג ובסדר עולה, או מאותו הערך) ו"לזרוק" אותם, תוך לקיחת קלף אחר- מן הקופה הראשית או מן הקלפים שזרק יריבו קודם לכן. המטרה היא להגיע לסכום קלפים נמוך מ-7.

כאשר סכום קלפי השחקן נמוך מ-7, מופיע במסך המשחק כפתור YANIV, אשר באמצעותו על השחקן להכריז שבעצם יש לו סכום קלפים נמוך מ-7. השחקן לא חייב ללחוץ עליו, אם הוא לחץ, מתבצעת בדיקה האם סכום קלפי היריב נמוך מסכום קלפי המכריז, אם לא, השחקן ניצח בסבב זה, הוא מקבל 0 נקודות ואילו יריבו מקבל נקודות לפי סכום כלפיו. בשל העובדה כי השחקן לא חייב ללחוץ על הכפתור, יכול להיווצר מצב בו לשני השחקנים יש יניב, אחד לא מכריז והשני מכריז, במצב זה יכול להיות כי סכום קלפי השחקן שלא הכריז נמוך יותר מסכום קלפי המכריז, וכעת השחקן שהכריז בעצם מפסיד את הסבב, ומקבל ASAF, מצב בו למכריז סכום קלפים גבוה משל היריב- היריב מנצח ומקבל 0 נקודות, ואילו המכריז מקבל 30 נקודות.

כאשר אחד מהשחקנים מגיע לסכום נקודות שעולה על 200, הוא מפסיק והמשחק נגמר.

במצב בו השחקן מגיע לסכום נקודות השווה בדיוק 200, לפי כללי המשחק סכום נקודותיו יורד ל-150 והמשחק ממשיך.

### קהל היעד של המשחק

המערכת מיועדת לכל אדם היודע איך להשתמש במחשב. היעד העיקרי הינו תרמילאים וטיילים אחרים הנוהגים לשחק את המשחק במסעותיהם, וכעת נמצאים בלימודים או בעבודה ויכולים להעביר את זמנם בנעימים במשחק. כמו כן ילדים ובני נוער ימצאו במשחק כלי להעברת הזמן בימי הקיץ החמים ומבוגרים יוכלו לשחק במשחק בכדי להשתחרר מהלחץ היומיומי.

דברי המפתחת- מטרות ויעדים

בתחילת העבודה על הפרויקט התלבטתי קשות במה לבחור. מצד אחד רציתי ליצור חידוש למשחק ישן שמאד אהבתי ומצד שני רציתי ליצור משהו שלא קיים עדיין. בסופו של דבר, בחרתי ליצור את המשחק ,יניב, אותו הכרתי בטיול עם חבריי, המשחק נפוץ מאד בין תרמילאים וכל הזמן רציתי לשחק בו עם חברי, אך מגבלות המרחק מנעו ממני.

מטרותיי המרכזיות הן:

יצירת תקשורת בין מחשבים מרוחקים.

יצירת ממשק ידידותי למשתמש בעל גרפיקה איכותית.

לבדוק את יכולתי בתכנות ממשק מתחילתו ועד סופו, תוך שילוב טכנולוגיות שונות.

בסופה של הדרך, אני יכולה להסתכל בגאווה על הפרויקט שיצרתי, מהמקום בו עמדתי בחילת הדרך- חוסר אמון עצמי שיש ביכולתי ליצור פרויקט עד למקום בו אני עומדת היום ומסתכלת אחורה בגאווה על הממשק אותו יצרתי לבד בעבודה קשה והתמדה.

כל מטרותיי קיבלו מענה-

ישנה תקשורת דרך שרת, כרגע בין 2 מחשבים, רציתי לשלב טכנולוגיות נוספות בזמן שהיה לי ולא להתמקד רק בתחום התקשורת, לכן יצרתי אפשרות רק לשני שחקנים בזמן נתון, על שאיפותיי לעתיד לגבי התחום הזה אפרט תחת המלצות לשיפור.

הממשק אותו יצרתי ידידותי למשתמש וקל להבינו, וכמו כן שילבתי טכנולוגיה חדשנית-WPF, אותה למדתי באופן עצמאי, לשם יצירת גרפיקה איכותית.

כמו שכבר אמרתי, בסופה של הדרך התוודעתי לכך שאם יש רצון ניתן לעשות הכל, ממקום של חוסר רצון ופחד הגעתי למצב בו אני רק רוצה להוסיף עוד ועוד ולהוציא לפועל לבד את כל רעיונותיי ושאיפותיי.



מדריך למפתח

## הכלים לכתיבת הממשק

שפת התכנות שבה השתמשתי בכדי לכתוב את הפרויקט היא C#.

בשל השימוש בטכנולוגית ה-WPF, ישנם קטעי קוד בXAML, אך לא כתבתי אותם ידנית.

סביבת העבודה:

פיתוח הממשק:

תחילה הן השרת והן הלקוח נכתבו ב-

Microsoft Visual C# 2008 Express

אך בשל המעבר לטכנולוגית WPF, הלקוח הועבר ל-

Microsoft Visual C# 2010 Express ונכתב בה עד סופו.

Microsoft .NET Framework – Version 4.0

גראפיקה:

Microsoft Expression Blend 4

Photoshop CS4

## דרישות חומרה:

מערכת הפעלה WINDOWS XP

עכבר ומקלדת

חיבור לאינטרנט

DVD-ROM

PORT מספר 5000 צריך להיות פנוי

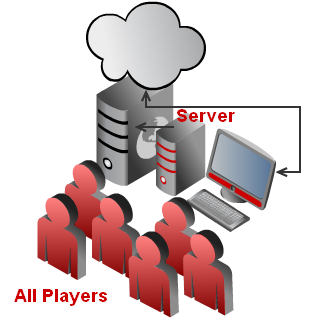
Micrisoft Word 2003 ומעלה

## תיאור המערכת הקיימת

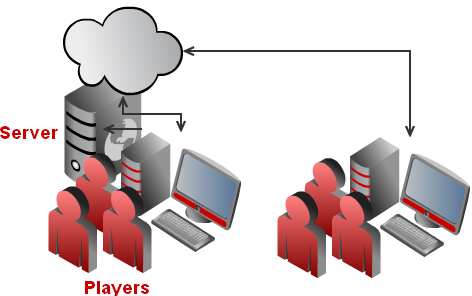
### תיאור משתמשי המערכת

בתרשימים שלהלן ניתן לראות פירוט תפקידי המשתמשים במערכת שלי, ואיך הם יכולים להשתנות.

ראשית, בכל האפשרויות מחשב מסויים מוקצה להיות השרת. תפקיד השרת הוא העברת נתונים בין שני הלקוחות, ביצוע חישובים ובכללית ניהול המשחק עצמו.

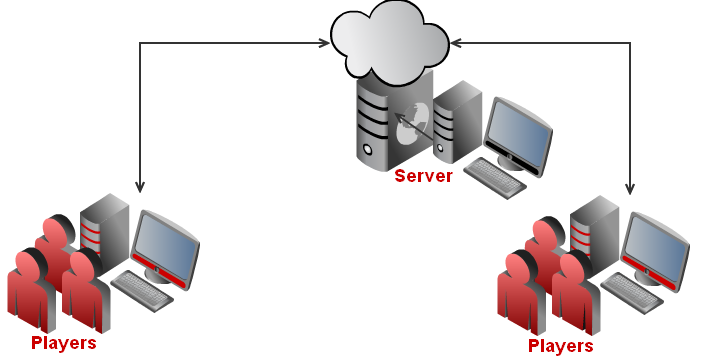
לגבי הלקוחות ישנן כמה אפשרויות.

אם המשתמשים מוגבלים במחשב אחד, ניתן לפתוח בו הן את השרת והן את שני המשתמשים. באופציה הזו אובד רעיות ההפתעה-השחקנים רואים את המהלכים והקלפים של היריב.



אפשרות שניה היא לפתוח שרת ולקוח

אחד באותו מחשב, ולקוח נוסף במחשב אחר.



כמו כן ניתן כמובן לפתוח את שלושתם במחשבים נפרדים.

הלקוחות מתחברים לשרת באמצעות כתיבת IP המחשב בו נמצא השרת (אותו גם מציג השרת עצמו).

אין צורך שאדם מסויים יתפעל את השרת, כל המידע המופיע בו לא רלוונטי לשחקנים או בכלל.

### מבנה השרת



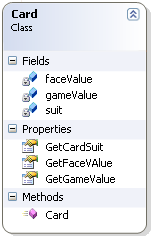
הממשק שלי מורכב בעצם משני פרויקטים נפרדים-אחד לשרת ואחד ללקוח. בעמודים הבאים אסביר על המחלקות מהן בנוי השרת וכן על הקשר ביניהן.

### מחלקות הקשורות לקלפים:

#### class Card

המחלקה הבסיסית ביותר והעיקרית אשר מטפלת בקלפים במשחק הינה המחלקה Card.

♥ המחלקה זהה הן בשרת והן בלקוח.

בשל העובדה כי בחרתי להציג ויזואלית הקלפים במשחק נעשת מתוך קובץ ה-card.DLL, התאמתי את ייצוג הקלף במערכת במשתנים בהם ישתמשו גם פונקציות ההצגה.

לכל קלף במשחק 3 משתנים, כל שלושת המשתנים הם מסוג ENAM.

private cardRank faceValue;

dllמשתנה עבור הערך של הקלף בקובץ ה-

private cardGameValue gameValue;

משתנה עבור הערך של הקלף במשחק עצמו

private CardSuit suit;

dllמשתנה עבור הערך של 'חליפת' הקלף בקובץ ה-

כמו שניתן לראות, לכל משתנה יש פ' properties, ע"י שימוש בהן מקבלים ערכי המשתנים של כל קלף.

public cardGameValue GetGameValue

public cardRank GetFaceVAlue

public CardSuit GetCardSuit

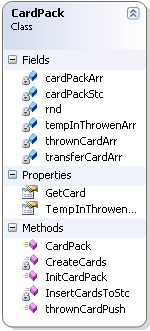
וכן ישנו הבנאי.

public Card(cardGameValue g\_Value, cardRank Value, CardSuit Suit)

#### class CardPack

זוהי המחלקה המטפלת בחפיסת הקלפים. ישנה חפיסת קלפים אחת לשני השחקנים.

המשתנים:

private Card[] cardPackArr;

מערך התחלתי של כל הקלפים לפי הסדר

private Stack<Card> cardPackStc;

מחסנית הקלפים הסופית-חפיסת הקלפים המעורבבת

private Card[] thrownCardArr;

מערך אליו מתווספים בכל סבב הקלפים הזרוקים, וכאשר בחפיסה הראשית נגמרים הקלפים, הקלפים הנמצאים כאן מתערבבים מחדש ונכנסים לחפיסה

private int tempInThrowenArr = 0;

משתנה עזר-למיקום אחרון של קלף שהוכנס למערך הקלפים הזרוקים

private static Random rnd = new Random();

משתנה רנדום, לשם ערבוב הקלפים

private Card[] transferCardArr;

מערך מעבר מהמערך המסודר למחסנית הקלפים

פונקציות:

public CardPack()

בנאי המחלקה. בשל העובדה כי חפיסת הקלפים משנה בכל סבב, בכדי לא ליצור כל פעם אוביקט חדש, ישנה פונקציית עזר שבה בעצם נוצרת החפיסה:

public void InitCardPack()

הפונקציה יוצרת חפיסת קלפים-מאתחלת משתנים, בונה את כל הקלפים במערך הראשוני ע"י שימוש בפ' אחרת private void CreateCards(), ומערבבת את חפיסת הקלפים ע"י שימוש בפ' הבאה:

private void InsertCardsToStc(Card[] cardArrToInsertFrom)

הפ' אשר מכניה רנדומאלית את הקלפים למערך. תחילה מוגדרים מערך ההעברה ומערך נוסף, לפיו נבין איזה מקום כבר תפוס. לאחר ההגדרה, ישנה פ' אשר מבצעת רנדומיזציה למספרים, בודקת אם המספר זה עוד לא תפוס, אם לא נכנס לשם קלף וכך מוגדר מערך ההעברה. לאחר מכן כל המערך נדחס לפי הדר המעורבב למחסנית.

public Card GetCard

פ' המוציאה את הקלף העליון בחפיסה. במקרה בו החפיסה נגמרה, הפ' קוראת לערבוב קלפים חדש, מתך מערך הזרוקים: InsertCardsToStc(thrownCardArr);

ומחזירה את הקלף העליון החדש.

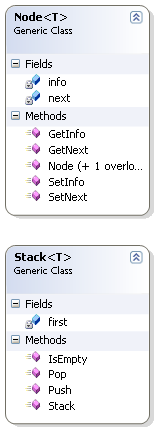
public void thrownCardPush(Card thrownCard)

פ' המכניסה את הקלפים הזרוקים למערך הזרוקים.

#### class Stack<T> ו- class Stack<T>

מחלקות עזר, לשם יצירת חפיסת הקלפים, הבנויה ממחסנית (Stack) יש צורך בשתי הפ' הללו, אשר נלמדו במקצוע עיצוב תוכנה. המחסנית בנויה מחוליות, ולכן יש צורך במחלקה Node.

ראשית יש לציין כי שתי המחלקות גנריות, כלומר ניתן לבחור את סוג המשתנים שיוכנסו ולא כל פעם לבנות מחלקה חדשה עבור כל סוג משתנה. בנוסף,פ' ה- properties בנויות בצורה שונה מפ' properties המקובלות ב-C#, זאת משום שהמחלקות נלקחו מן הספר עיצוב תוכנה מבוסס עצמים, בו המחלקות הותאמו גם לשפת ג'אווה,שם לא ניתן להשתמש בפ' המקובלות בC#.



המחלקה Node:

משתנים:

private T info;

במשתנה זה נשמר המידע של החוליה, במקרה שלנו, הקלף.

private Node<T> next;

המצביע על החוליה הבאה ו- info

פונקציות:

public Node(T x)

public Node(T x, Node<T> next)

בנאיי המחלקה

public Node<T> GetNext()

public void SetNext(Node<T> next)

public T GetInfo()

public void SetInfo(T x)

פונקציות ה propertiesשל המחלקה.

המחלקה Stack:

משתנים:

private Node<T> first;

המחסנית בנויה מחוליות

פונקציות:

public Stack()

בנאי המחלקה

public bool IsEmpty()

פ' הבודקת האם המחסנית ריקה

public void Push(T x)

פ' המכניסה אוביקט חדש למחסנית-מכניסה כעליון

public T Pop()

פ' המוציאה את האוביקט העליון מהמחסנית

### מחלקות הקשורות לתקשורת:

#### הקדמה:

התקשורת אותה בניתי בפרויקט שלי משתמשת במחלקות מתוך:

1. System.Net.Sockets;

2. System.Threading;

3. System.IO;

System.Net \*

השימוש העיקרי הוא ב-(1), אשר נובע מ(\*)

מרחב השמות (\*) מכיל מחלקות המספקות ממשקי תכנות למספר פרוטוקולי רשת, כגון יבוא משאבים מן הרשת, שליחת אימיילים, גישה למידע אינטרנטי וכ"ו.

לגבי תקשורת, קיימות מספר אפשרויות. ניתן לתכנת את הממשק בעזרת פיר-טו-פיר, שם התקשורת מתבצעת בין 2 משתמשים, כל אחד מהם גם שרת וגם לקוח, או לחלופין לבצע תקשורת דרך שרת, בו ניתן ליצור תקשורת בין מעל ל-2 משתמשים. האפשרות שאני בחרתי היא תקשורת דרך שרת, וזאת בשל העובדה כי השארתי לעצמי דלת פתוחה להוספת 'חדרים' והרחבת המשחק למעל ל-2 משתמשים.

בכדי לבצע את התקשורת שלי, דרך השרת, השתמשתי במרחב השמות (2) שהוא בעצם 'ילד' של (\*). מרחב השמות הזה מפשר לי לאפשר גישה לזרמי רשת, וע"י כך ליצור תקשורת בין מחשבים.

בשל העובדה כי התקשורת היא בעצם שימוש בתהליכים, השתמשתי גם במרחב השמות (2). מרחב השמות הזה מאפשר תכנות מרובה תהליכים. בממשק שלי קיימים תהליכים רבים, בשרת אלו תהליכי התקשורת, (על התהליכים אצל הלקוח אסביר בחלק על הלקוח) ולשם תמרון ביניהם משתמשים בפונקציות ו'כלים' מתוך המרחב הנ"ל.

מרחב השמות (3) מכיל טיפוסים המאפשרים קריאה וכתיבה לקבצים וזרמי מידע.

בכדי להשתמש בזרמים עליהם מבוססת התקשורת, ישנו שימוש ב'קורא' ו'כותב'

אשר לקוחים ממרחב השמות הנ"ל.

הכתיבה (בכל הפרויקט) מתבצעת באמצעות אותו 'כותב' עליו דיברתי קודם- באופן הבא:

Frame frame = new Frame(המשתנים לבניית המערך, כל המידע שיועבר);

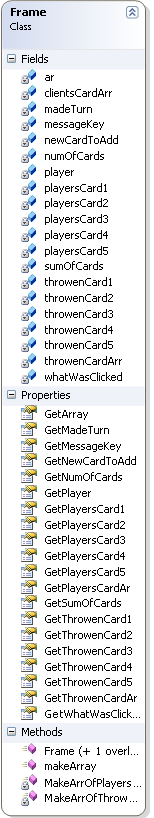
writer.Write(frame.GetArray);

הקריאה (בכל הפרויקט) מבוצעת בופן הבא- הקורא 'תופס' מערך בגודל במערך הדרוש, לאחר מכן המערך מפורק.

arForReceive = reader.ReadBytes(39);

Frame frame = new Frame(arForReceive);

#### class Frame

התקשורת נעשת, כמו שאמרתי באמצעות זרמים. בכדי ליצור את הזרמים ישנו שימוש ב'קורא' ו'כותב' אשר "מתרגמים" טיפוסים לזרמים. הטיפוס בתקשורת שאני יצרתי הוא מערך של ביטים. המחלקה frame היא המחלקה אשר אחראית על המערך הזה.

♥ חשוב לציין, כי הן השרת והן הלקוח שניהם משתמשים בדיוק באותה מחלקה frame ולכן לא אפרט עליה פעם נוספת בחלק על הלקוח.

משתנים:

private byte player;

הסימן של השחקן. כאשר כל שחקן מתחבר הוא מקבל סימן-0 לראשון 1 לשני, בכדי לזהות ממי מגיע המידע וכ"ו

private byte numOfCards;

מספר הקלפים של השחקן

private byte sumOfCards;

סכום הנקודות של השחקן (לפי ערכי הקלפים שבידיו)

private byte messageKey;

משתנה להעברת הודעות (ההודעות יפורטו במחלקה enumAll)

private byte madeTurn;

משתנה המודיע אם השחקן סיים את תורו או לא

private Card playersCard1;… playersCard5;

משתנים המייצגים את קלפי השחקן. כל קלף מורכב ממשתני enum:byte ובעת הכנסת הקלפים למערך כל קלף "מפורק" לשלושת משתניו וכשהמידע מתקבל מורכב מחדש.

private Card newCardToAdd;

משתנה להעברת הקלף החדש מהחפיסה

private Card throwenCard1;… throwenCard5;

משתנםי להעברת הקלפי םהזרוקים. גם הם מפורקים למרכיביהם.

private byte whatWasClicked;

משתנה המציין מה נלחץ (סוגי הלחיצות גם הן יפורטו במחלקה enumAll)

private Card[] clientsCardArr = new Card[5];

מערך לקלפי השחקן- בעת "ההרכבה" וה"פירוק הקלפים הם בתוך מערך וממנו\אליו הם מועברים בשליחה\קבלה של המידע

private Card[] throwenCardArr = new Card[5];

מערך לקלפים הזרוקים

private byte[] ar;

המערך בו מועברים כל הנתונים.

פונקציות:

public byte[] GetArray

public Card[] GetPlayersCardAr

public Card[] GetThrowenCardAr

public byte GetPlayer

public byte GetNumOfCards

public byte GetSumOfCards

public byte GetMessageKey

public byte GetMadeTurn

public Card GetPlayersCard1

public Card GetPlayersCard2

public Card GetPlayersCard3

public Card GetPlayersCard4

public Card GetPlayersCard5

public Card GetNewCardToAdd

public Card GetThrowenCard1

public Card GetThrowenCard2

public Card GetThrowenCard3

public Card GetThrowenCard4

public Card GetThrowenCard5

public byte GetWhatWasClicked

כל הנ"ל הן פונקציות properties, הן משמשות אך ורק להוצאת הנתונים מן המערך שהתקבל.

הפ' הבונות את המערכים:

public void makeArray()

הפ' מכניסה את הנתונים שיש להעביר לתוך המערך שיישלח, בסדר קבוע

private void MakeArrOfThrowenCards()

private void MakeArrOfPlayersCards()

כאשר הנתונים מתקבלים המחלקה משתמשת בפונקציות הנ"ל כדי ליצור מן הקלפים שהתקבלו מערכים, אותם מעבירים ללקוחות.

הבנאים:

בשל העובדה כי המערך יכול להבנות גם ממערך שהתקבל וגם בתור מערך שנבנה מהמשתנים לשליחה ישנם שני בנאים.

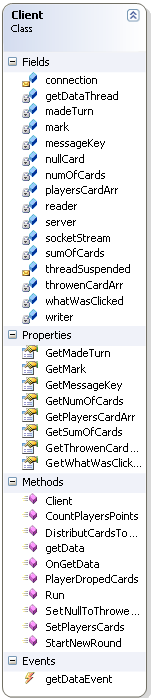
public Frame(byte player, MadeTurnMK madeTurn, connectionMK messageKey, byte numOfcards, byte sumOfCards, Card pCard1, Card pCard2, Card pCard3, Card pCard4, Card pCard5, Card newCard, Card tCard1, Card tCard2, Card tCard3, Card tCard4, Card tCard5, whatClickedMK whatWasClicked)

הבנאי מקבל את כל המשתנםי לשלחה, מעביר אותם למשתני האוביקט ולאחר מכן בונה את המערך, ומזמן את הפ' המכניסה את כל המשתנים למערך בתור משתנםי מסוג בייט

public Frame(byte[] ar)

הבנאי הנ"ל הוא הבנאי שמקבל את המערך שנשלח מן השרת ועליו לפרק אותו, לכן הוא מעביר כל משתנה ממקומו במערך למשתנה האובייקט. לגבי הקלפים- הוא מרכיב כל קלף ממשתניו שהתקבלו כבייטים. לאחר מכן הוא מזמן את פ' בניית מערכי הקלפים של השחקן.

#### class Client

המחלקה client היא המחלקה העיקרית המטפלת בתקשורת.

המחלקה מגדירה את הלקוח, נתוניו, מחלקת לו את הקלפים ודואגת להעביר את המידע בין השרת ללקוח המעשי. ישנם בעצם במצב כרגע 2 לקוחות, והמחלקה הזו של כל לקוח מאפשרת להבחין עם מי התקשורת, למי שולחים, מה מקבלים וכ"ו.

המשתנים:

internal Socket connection;

מגדירה את נק' היעד של התקשורת הדו-צדדית בין 2 תוכניות הרצות ברשת. ה Socket הוא בעצם קשר למספר פורט כך ששכבת ה-TCP תוכל לזהות את האפליקציה אליה המידע מיועד להשלח.

private NetworkStream socketStream;

כמו שאמרתי, התקשורת מתבצעת באמצעות זרם בין הלקוח לשרת, לכן על מנת לתקשר נחוץ הזרם.

private Server server;

מגדיר את השרת

private BinaryWriter writer;

private BinaryReader reader;

כמו שאמרתי, בכדי לתקשר בזרמים יש צורך ב'קורא' ו'כותב' אשר 'מתרגמים' את המידע הנשלח לזרם וממנו.

private Thread getDataThread;

התקשורת בעצם מתבצעת בתהליכים, לכן פה מוגדר תהליך קבלת המידע.

public event GetData getDataEvent;

מגדיר אירוע לקבלת המידע

internal bool threadSuspended = true;

משתנה הנועד להשעות את תחילת תהליך ההתקשרות בעוד הלקוח השני לא מחובר.

private byte mark;

private Card[] playersCardArr;

private Card[] throwenCardArr;

private byte numOfCards;

private byte sumOfCards;

private byte messageKey;

private byte madeTurn;

private byte whatWasClicked;

האינפורמציה מהלקוח המעשי אותו מקבל אוביקט הלקוח בשרת תבכדי לטפל במשחק וכ"ו

private Card nullCard = new Card(cardGameValue.Null, cardRank.Null, CardSuit.Null);

משתנה עזר, מערך ההעברה הוא קבוע וכאשר אין קלף פיסי להעביר (כאשר מהות התקשרות אינה לקיחת וזריקת קלף בעצם) אשתמש בקלף ריק.

הפונקציות:

הבנאי:

public Client(Socket socket, Server server, byte newNumber, CardPack cardPack)

הבנאי מגדיר את נתוני הלקוח-ראשית מספרו, מחלק לו קלפים וסופר נקודותיו, ושנית את כל הקשור לתקשורת-נק' היעד של התקשורת, השרת, את הזרם, הכותב והקורא, וכמו כן מגדיר את תהליך קבלת המידע עבור הלקוח הזה.

public void DistributCardsToPlayer(CardPack pack)

פ' עזר המשמשת לחלוקת הקלפים, הן התחלתית והן בתחילת כל סבב חדש.

public byte GetMark

public Card[] GetPlayersCardArr

public Card[] GetThrowenCardArr

public byte GetNumOfCards

public byte GetSumOfCards

public byte GetMessageKey

public byte GetMadeTurn

public byte GetWhatWasClicked

פונקציות ה- propertis של הלקוח, בהן משתמש האוביקט Clients לשם מהלך המשחק וכ"ו.

public void CountPlayersPoints(Card newCard)

הפ' סופרת את סכום הנקודות החדש של השחקן, יחד עם הקלף ששלף

public void SetPlayersCards(Card newcard)

מוסיפה את הקלף שהשחקן שלף זה עתה למערך קלפיו (המערך הזה ישלח אליו בחזרה)

פונקציות הקשורות לתקשורת:

public void PlayerDropedCards(byte player, byte numOfCards, byte newSumOfCards, MadeTurnMK TurnDone, Card newCard, connectionMK message, Card[] otherPlayersThrowen)

הפ' הזו משמשת את השרת לשם שליחת המידע ללקוח המיועד, בעת הסבב.

public void StartNewRound(byte player, connectionMK messageKey)

גם הפונקציה הזו משמשת לשם שליחת מידע, ההבדל הוא שכעת השליחה היא לשם תחילת סבב חדש, כלומר מועבר מיידע באאופן מעט שונהף לכן הפונקציות נפרדות.

public void getData()

הפ' הזו משמשת את האוביקט לקבל את המידע אשר הלקוח המעשי שלח. מוגדר מערך קבלה, ואז מופעלת לולאה אינסופית שמטרתה "לתפוס" את המידע שנשלח. ברגע שה'קורא' קולט מידע שהתבל בזרם שבין הלקוח המעשי לשרת, הוא מפרק את המערך שהתקבל ומעדכן את משתני האוביקט, אם נעשה תור האירוע getDataEvent של קבלת המידע מופעל. האירוע עצמו מוגדר בעזרת דליגייט לפ' המתבצעת במחלקה clients, וזאת משום שבסוף האירוע מידע צריך להשלח הן ללקוח הזה והן לשני, כאשר שני האוביקטים הללו הוגדרו במחלקה clients.

public void OnGetData()

הפ' שבעזרתה הפ' getData מפעילה את האירוע getDataEvent.

התקשורת מתבצעת באופן הבא:

לכל לקוח מוגדרת פ' RUN- הפ' רצה כאשר השרת מזהה התחברות של לקוח

public void Run()

{

bool done = false;

server.Display("Player " + mark + " connected");

הלקוח החדש זה עתה התחבר, כעת יישלחו אליו כל הנתוניו שאתחל השרת

Frame frame = new Frame((byte)this.mark, (byte)0, (byte)0,

(byte)5, (byte)this.sumOfCards, this.playersCardArr[0], this.playersCardArr[1], this.playersCardArr[2], this.playersCardArr[3], this.playersCardArr[4], this.nullCard, this.nullCard, this.nullCard, this.nullCard, this.nullCard, this.nullCard, whatClickedMK.nothing);

writer.Write(frame.GetArray);

השורה הבאה מגדירה את תחילת תהליך קבלת המידע

getDataThread.Start();

קטע הקוד הבא נועד להשעות את התהליך כל עוד השחקן השני מנותק

if (mark == 0)

{

lock (this)

{

while (threadSuspended)

Monitor.Wait(this);

}

}

לקטע הקוד הזה מגיעים כאשר שני השחקנים התחברו, כעת מופעלת לולאה אינסופית לשם הפעלת התקשורת

while (!done)

{

while (connection.Available == 0)

{

if (server.disconnected)

return;

}

server.Display(string.Format("{0} - {1}", mark));

ברגע שהמשחק הסתיים, יש לסגור את הליכי התקשורת, לכן המשתנה done הופך לאמת, וכעת הלולאה האינסופית של ההתקשרות נסגרת

if (server.GameOver())

done = true;

}

משתני ההתקשרות נכבים.

writer.Close();

reader.Close();

socketStream.Close();

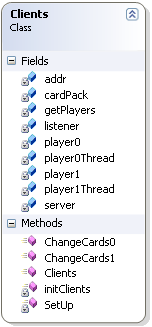
connection.Close();

}

#### class Clients

המחלקה אחראית על ניהול המשחק. כלומר, זו המחלקה המנהלת את החפיסה ואוביקטי הלקוחות, אחראית על אלגוריתם המשחק, כיוון שליחת המידע הרלוונטי ללקוח הרלוונטי הגדרת התהליכים של כל לקוח וכ"ו.

משתנים:

private Client player0;

private Client player1;

מגדיר את אובקטי השחקנים.

private Thread getPlayers;

תהליך המשחק עצמו

private TcpListener listener;

זהו משתנה המאזין לתקשורת בין לקוחות ברשת TCP

TCP (Transmission Control Protocol)-פרוטוקול העברת מידע באינטרנט

private Thread player0Thread

private Thread player1Thread;

תהליך המשחק של השחקן- מה שיקרה בתורו

IPAddress[] addr;

זהו בעצם מערך של מספרי ה-IP של המחשב הנתון

private Server server;

מגדיר שרת

private CardPack cardPack;

מגדיר את אוביקט חפיסת הקלפים למשחק

פונקציות:

בנאי:

public Clients(Server server)

פונקציה הבונה את האובייקט. מגדירה חפיסה ושרת ומשתמשת בפ' העזר להלן לשם הגדרת המשחק.

לפ' הבאות יש חשיבות רבה בהקשר לתקשורת לכן אסביר את הקוד:

private void initClients()

{

בכדי שהשרת ייזהה את הIP של המחשב המארח יש צורך הפעולות הבאות: ראשית, נלקח השם של המארח, לאחר מכן הוא לוקח מאגר כתובות של המחשב המארח ומכניס אותן למערך. במקום 0 במערך זה נמצא הIP הדרוש.

String strHostName = Dns.GetHostName();

IPHostEntry ipEntry = Dns.GetHostEntry(strHostName);

addr = ipEntry.AddressList;

server.Display("IP: "+addr[0].ToString());

לאחר מכן הפ' מגדירה את המשחק-מגדירה תהליך המגדיר את המשחק ומפעילה אותו.

getPlayers = new Thread(new ThreadStart(SetUp));

getPlayers.IsBackground = true;

getPlayers.Start();

}

בפונקציה הבאה מתבצעת ההגדרה של המשחק:

private void SetUp()

{

מגדיר את נק' היעד של התקשורת- הIP עליו ירוץ השרת ואליו וממנו ישלח המידע.

IPEndPoint ipEP = new

IPEndPoint(IPAddress.Parse(addr[0].ToString()), 5000); listener = new TcpListener(ipEP);

listener.Start();

כעת מוגדר השחקן הראשון שיתחבר-

player0 = new Client(listener.AcceptSocket(), server, 0,

this.cardPack);

בנקודה זו הקוד ממתין להתחברות של שחקן, ורק ברגע שהוא מתחבר הקוד ממשיך

player0.getDataEvent += new GetData(ChangeCards0);

מגדיר את האירוע אותו מפעיל השחקן כשמשחק.

player0Thread = new Thread(new ThreadStart(player0.Run));

מגדיר את תהליך התקשורת של השחקן הראשון

player0Thread.IsBackground = true;

player0Thread.Start();

מפעיל את התהליך.

כנ"ל לגבי הקוד ל השחקן השני

player1 = new Client(listener.AcceptSocket(), server, 1,

this.cardPack);

player1.getDataEvent += new GetData(ChangeCards1);

player1Thread = new Thread(new ThreadStart(player1.Run));

player1Thread.IsBackground = true;

player1Thread.Start();

הקטע הזה נועל את הקוד כדי לבצע קטע קוד כקריטי, לאחר הביצוע הנעילה מתבטלת. בעצם כעת הנעילה שבוצעה כשהתחבר השחקן הראשון מתבטלת, כעת ידוע כי שני שחקנים מחוברים.

lock (player0)

{

player0.threadSuspended = false;

Monitor.Pulse(player0);

}

}

הפונקציות לביצוע המשחק מוגדרות כלהלן:

public void ChangeCards0(Object sender)

public void ChangeCards1(Object sender)

הפונקציה בודקת בכל פעם מהיא ההודעה שהתקבלה, ולפי ההודעה מבצעת את הקוד המתאים. הקוד שלהן זהה, רק יש החלפה בין ההכוונה ללקוח.

האפשרויות הן:

הודעה מספר 0: השחקן זרק קלף ולקח קלף אחר, הפ' מכניסה תא הקלפים הזרוקים למערך הזרוקים ולחר מכן בודקת על מה לחץ השחקן- האם על החפיסה הראשית- אם כן ההוראה היא לשלוף את הקלף העליון מהחפיסה, או לחלופין אם לחץ על אחד הקלפים הזרוקים של השחקן הקודם- ואם כן זהו הקלף המוחזר. לאחר מכן מעדכנים משתנים, ומזמנים את הפ' PlayerDropedCards מהמחלקה Client, האחראית, כנאמר על שליחת המידע המעודכן ללקוח הרלוונטי.

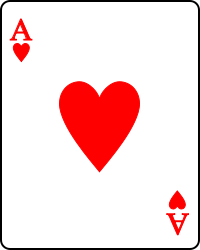
הודעה מספר 2: הלקוח לחץ על יניב, כעת הפ' בודקת את מצב הנקודות של הלקוח ושל היריב, ושולחת הודעות בהתאם בעזרת הפ' PlayerDropedCards.

הודעה מספר 6: הסבב נגמר ואחד השחקנים לחץ על כפתור Ready, לכל שחקן נשלחת הודעה בעזרת הפ' StartNewRound, גם היא מהמחלקה Client, המודיעה לו על הסטטוס שלו- ללוחץ על כך שלחץ וממתין לשני וליריב הודעה בדבר הלחיצה של השחקן הראשון.

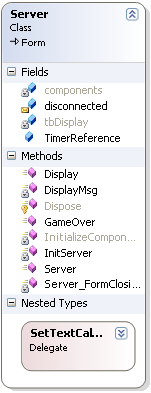
הדעה מספר 9: שני השחקנים מוכנים להתחיל לשחק-כלומר שניהם מחוברים ולחצו Ready. כעת מאותחלים נתוני המשחק- החפיסה וכ"ו ולשחקנםי נשל חכל המידע החדש בעזרת StartNewRound, והמשחק אצל הלקוחות מתחיל.

הודעה מספר 11: הלקוח התנתק, כלומר יש לסגור את התקשורת.

לסיכום, המחלקה מגדירה משתנים ותהליכים לפיהם יתנהל המשחק והתקשורת. יש תהליך המאתחל את המשחק, ובו מאתחלים את תהליכי התקשורת של כל לקוח עם השרת וכמו כן מגדירים את הפ' לביצוע של כל שחקן כאשר מתקבל מידע (האירוע).



#### class Server : Form

זהו השרת הכולל. המעטפת הכללית של כל הפ' והפורם בו מוצג המידע של השרת.

משתנים:

internal bool disconnected = false;

משתנה בוליאני בו אשתמש לשם התנתקות וסגירת התקשורת כאשר השרת נסגר.

public System.Threading.Timer TimerReference;

משתנה המספק מנגנון לביצוע פונקציה מסוימת במרווחי זמן שצוינו.

פונקציות:

בנאי:

public Server()

בונה את הפורם וקורא לפ' עזר לאתחול השרת

private void InitServer()

פ' עזר המאתחלת את האוביקט clients

public void Display(string message)

פ' המזמנת את הפ' להצגת הודעה כלשהי

private void DisplayMsg(string msg)

הפ' מציגה את ההודעה על השרת. במקרה ורץ תהליך אחר, הפ' משתמשת בinvoke בכדי לגרום לקוד להתבצע.

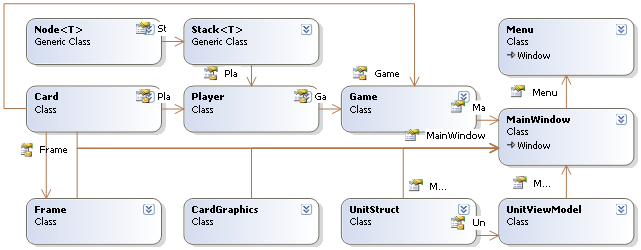
(אדבר בהרחבה על invoke בקטע על הלקוח)

private void Server\_FormClosing(object sender,FormClosingEventArgs e)

פ' המבצעת את הפעולות הדרושות בעת סגירת השרת.

♥ אציין, כי לפרויקט השרת יש חלק נוסף- EnumAll הכולל בתוכו את הייצוג של כל משתני הENUM בהם השתמשתי בשרת. את הפירוט אביא בחלק על הלקוח, וזאת משום שגם שם קיימים אותם משתני ENUM בנוסף לכמה אחרים.

### מבנה הלקוח



בעמודים הבאים אסביר על המחלקות מהן בנוי הלקוח.

### מחלקות הקשורות לקלפים:

#### class Card, \*class Frame, class Stack<T>, class Node<T>

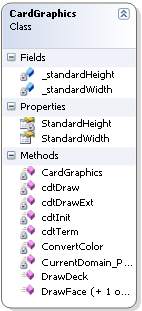
מבנה המחלקות הנ"ל זהה לזה שבלקוח, לכן לא אפרט עליהן פעם נוספת.

(\*)-לא קשורה לקלפים, אלא, כנאמר, לתקשורת.

#### class CardGraphics

זוהי המחלקה האחראית לכל מה שנוגע בגרפיקת הקלפים. במקום להוסיף תמונה לכל קלף, בחרתי להשתמש בקובץ card.dll המשמש את משחקי הקלפים של windows.

משתנים:

private static int \_standardWidth;

private static int \_standardHeight;

משתני גובה ורוחב סטנדרטי, אלו הפרמטרים בעזרתם אני בודקת האם הספרייה cards.dll אותחלה.

פונקציות:

public static int StandardHeight

public static int StandardWidth

פ' ה- Propertiesשל משתני המחלקה.

static CardGraphics()

בנאי המחלקה-בודק האם הספרייה אותחלה, אם לא שולח הודעת שגיאה, ומגדיר אירוע לניקוי המשאבים כשהתהליך נסגר.

private static void CurrentDomain\_ProcessExit(object sender, EventArgs e)

פ' המזמנת את ניקוי המשאבים.

public static void DrawFace(Graphics g, int x, int y, int width, int height, int faceValue, CardSuit suit, Color invertedColor)

פ' הציור הראשית של פני הקלף, בודקת אם כל המשתנים תקניים ומזמנת את הפ' אותה מייצאת ספריית ה-DLL לציור.

public static void DrawFace(Graphics g, int x, int y, int width, int height, int faceValue, CardSuit suit)

הפ' מזמנת את הפ' DrawFace הראשונה, זו הפ' בה אני משתמשת, בה אין אזכור לצבע.

public static void DrawDeck(Graphics g, int x, int y, int width, int height, CardDeck deck)

פ' האחראית לציור החפיסה. גם הציור של החפיסה מגיע מתוך קובץ ה-DLL וניתן לבחור בין כמה איורים מובנים.

הקובץ card.dll מייצא 5 פונקציות שבעזרתן ניתן לצייר את הקלפים. בפרויקט שלי השתמשתי ב-3 מהן.

[DllImport("cards.dll")]

private static extern bool cdtInit(ref int width, ref int height);

הפ' הזו היא פונקצת האתחול של הספרייה cards.dll - הפ' לוקחת 2 ארגומנטים- הגובה והאורך של הקלף, היא מחזירה אמת אם cards.dll אותחל ושקר אם קרתה תקלה. הזו אחראית על המשאבים בהם השתמש הקלף.

השימוש בפ' הנ"ל במערכת שלי הוא בתור משתנה בוליאני שבעזרתו אני יודעת האם הקלף אותחל או לא, אם לא כנראה שקרתה טעות.

[DllImport("cards.dll")]

private static extern void cdtTerm();

הפ' הזו היא פ' העושה את הפעולה ההפוכה לאתחול, היא בעצם משחררת את המשאבים של הקלף. השימוש בה הוא חד פעמי, רק רגע לפני שהאפליקציה נסגרת ברצוני לשחרר את המשאבים בהם השתמש הקלף.

[DllImport("cards.dll")]

private static extern bool cdtDrawExt(IntPtr hdc, int x, int y,int dx, int dy, int card, int type, int color);

הפ' הזו אחראית על הציור הפיסי של הקלף. היא מקבלת פרמטרים ולפיהם מציירת את הקלף.

### מחלקות הקשורות למשחק:

#### טבלת הניקוד:

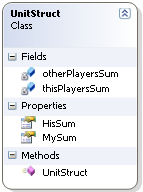
בשל העובדה כי העברתי את הלקוח מwindows form לWPF, נאלצתי למצוא פתרונות לכמה קטעי קוד אשר לא פועלים\קיימים ב-WPF. אחד מאותם חלקים הוא הטיפול בטבלת התוצאות. כשהשתמשתי ב windows form הוספתי data base ולו היו פעלות מובנות בהן השתמשתי בכדי להציג את תוצאות הסבבים. במעבר ל-WPF נתקלתי בבעיה-לא היה קיים קונטרול data base, אלא רק dataGrid, בו השימוש הוא ייבוא נתונים מקבצי דטה קיימים. לכן בניתי 2 מחלקות, אותן אני מייבאת ל dataGrid.

#### class UnitStruct

המחלקה מייצגת שורה אותה אני אייבא בעזרת מחלקה נוספת ל dataGrid.

המחלקה בעצם מייצגת את המידע שיוכנס לתוך כל שורה. המידע אות וברצוני להציג הינו הניקוד של השחקנים. בסוף כל סבב יש שימוש כפול באובייקט הזה- האחד לסכום הנקודות בסבב הנוכחי והשני סכום הנקודות הכולל בכל הסבבים.

משתנים:

private byte thisPlayersSum;

private byte otherPlayersSum;

כל משתנה מייצג את סכום הנקודות של השחקנים.

פונקציות:

בנאי המחלקה:

public UnitStruct(byte mySum, byte hisSum )

פונקציות properties:

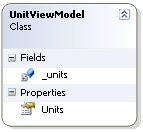
public byte MySum

public byte HisSum

#### class UnitViewModel

תפקיד המחלקה הזו הוא לקשר בין אובייקטי ה- UnitStructלבין ה- dataGrid, בה הם יוצגו.

משתנים:

private ObservableCollection<UnitStruct> \_units = new ObservableCollection<UnitStruct>();

המשתנה הזה בעצם מייצג את היחידות-כלומר השורות אותן אוסיף ל dataGrid.

ObservableCollection) מייצגת אוסף מידע דינאמי, ומספקת התראות על הוספה הסרה של המידע כ"ו. )

פונקציות:

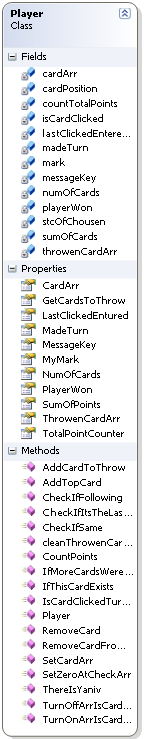
public ObservableCollection<UnitStruct> Units

פ' properties דרכה אני מעבירה את הנתונים שביוניטס ל dataGrid.

#### class Player

זוהי המחלקה המייצגת את הלקוח-השחקן במשחק. ישנו ייצוג לכל מאפייניו וכל הפ' הקשורות לתפעולו ומהלכיו מוכללות כאן.

משתנים:

private byte mark;

משתנה זה מכיל את הסימן של השחקן- הראשון או השני. נחוץ לשם תיאום עם השרת.

private Card[] cardArr;

מערך לקלפי השחקן בכל תור

private Card[] throwenCardArr;

מערך לקלפים אותם זורק השחקן בכל תור

private byte numOfCards;

מספר הקלפים בידי השחקן בכל תור

private byte sumOfCards;

סכום הנקודות של השחקן בכל תור (לפי ערכי הקלפים בידיו)

private byte madeTurn;

משתנה המסמן האם השחקן סיים תורו או שלא.

private byte messageKey;

משתנה לשם העברת הודעות בין השחקן לשרת

private int[] isCardClicked;

משתנה הקשור ללוגיקת המשחק-בשל העובדה כי ניתן ללחוץ על וריאציות שונות של שילובי קלפים, בניתי משתנה עזר בו אני מסמנת את מקומות הקלפים שנלחצו ובעזרתו בודקת האם ישנם עוד קלפים לחוצים וכ"ו.

private Card lastClickedEnteredCard;

לשם הבדיקות לאיזו וריאציה שייך הקלף-סדרה או רצף, יש לדעת בכל פעם מהו הקלף האחרון שנלחץ.

private Stack<Card> stcOfChousen;

הקלפים שבוחר השחקן לזרוק בלכ תור נכנסים למחסנית, ממנה הם נזרקים

private int cardPosition;

בכדי להכניס את הקלף החדש למקום של הקלף האחרון שנזרק יש לדעת את מיקומו.

private byte countTotalPoints;

משתנה זה סופר בתוכו את מספר הנקודות הכולל בכל המשחק של השחקן.

private bool playerWon;

משתנה בוליאני בערתו המשחק יודע מי השחקן שמתחיל.

פונקציות:

public byte TotalPointCounter

public bool PlayerWon

public byte MyMark

public Card[] ThrowenCardArr

public byte SumOfPoints

public byte NumOfCards

public Card[] CardArr

public byte MadeTurn

public byte MessageKey

public Card LastClickedEntured

public Stack<Card> GetCardsToThrow

פונקציות ה- propertisשל המחלקה.

public void SetZeroAtCheckArr()

לאחר כל תור יש לאפס את המערך בו סימנתי את הקלפים הלחוצים.

public void TurnOnArrIsCardClicked(int whereTo)

מטרת הפ' לסמן שקלף מסויים נלחץ

public void TurnOffArrIsCardClicked(int whereTo)

מטרת הפ' להוריד את הסימון של קלף מסויים שנחץ

public bool IsCardClickedTurndOn(int whrer)

הפ' בודקת אם הקלף הנתון נלחץ

public bool IfMoreCardsWereClicked()

הפ' בודקת אם נלחצו קלפים נוספים, אם כן מתבצעות בדיקות של וריאציות לחיצה בעזרת פ' אחרות.

public bool CheckIfFollowing(Card nextClickedCard)

הפ' בודקת אם הקלף שנלחץ זה הרגע הוא הקלף העוקב של הקלף שנלחץ לפניו.

public bool CheckIfSame(Card nextClickedCard)

הפ' בודקת אם הקלף שנחץ זה הרגע והקלף שנלחץ לפני כן הם מאותו ערך.

public bool CheckIfItsTheLastCardWhichWasClicked(Card nextClickedCard)

בכדי לבטל לחיצה, יש לבדוק שזהו אכן הקלף שנלחץ אחרון. לא ניתן לבטל לחיצה על קלף שנלחץ לפני הקלף האחרון, וזאת משום שאם בוחרים למשל סדרה, אי אפשר להוריד את הלחיצה מהקלף האמצעי, כי אז יבחרו קלפים ללא קשר.

public void AddCardToThrow(Card cardToThrow)

הפ' הזו מוסיפה את הקלף למערך הזריקה, לאחר כל הבדיקות.

public void RemoveCardFromArrThrow()

הפ' מוציאה את הקלף לזריקה מן מחסנית הנבחרים.

public void RemoveCard(int cardPo)

הפ' מוחקת קלף ממערך קלפי השחקן.

public void AddTopCard(Card pack)

הפ' מוסיפה את הקלף החדש למערך של השחקן.

public bool IfThisCardExists(int where)

הפ' בודקת אם קיים קלף במיקום הנבחר.

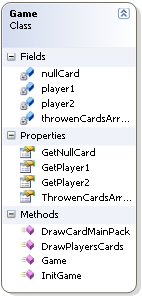
public bool ThereIsYaniv()

הפ' בודקת אם לשחקן יש יניב, אם כן מחזירה אמת.

public void CountPoints()

הפ' סופרת את סכום הנקודות של השחקן (לפי ערכי הקלפים שבידיו).

#### class Game

המחלקה הזו היא ה"מעטפת" של המשחק, ממנה יש גישה לשחקנים וכמו כן פ' הזימון לציור קלפים.

משתנים:

private Player player1;

private Player player2;

המשתנםי מגדירים את אובייקטי השחקנים במשחק. שחקן 1 הוא השחקן המעשי, אילו שחקן 2 הוא היריב.

private Card nullCard;

משתנה עזר, מערך ההעברה הוא קבוע וכאשר אין קלף פיסי להעביר (כאשר מהות התקשרות אינה לקיחת וזריקת קלף בעצם) אשתמש בקלף ריק.

private Card[] throwenCardsArrToDraw;

מערך בו נכנסים הקלפים הזרוקים שיש לצייר, לשימוש פ' הציור שבמחלקה זו.

♥ היני מודעת לעובדה כי ערבבתי ביזנס לוג'יק וגרפיקה באותה המחלקה, אך לא מצאתי מקום מתאים יותר לפ'.

פונקציות:

בנאי:

public Game()

הבנאי משתמש בפ' עזר לשם הגדרת המשחק, וזאת משום שהשארתי חלון אפשרויות לתחילת משחק נוסף ללא סגירת משטח המשחק, כדי שלא אצטרך לבנות את האוביקט חדש.

public void InitGame()

פ' העזר להגדרת המשחק.

public Player GetPlayer1

public Player GetPlayer2

public Card GetNullCard

public Card[] ThrowenCardsArrToDraw

פונקציות ה- propertisשל המחלקה.



פונקציות ציור הקלפים:

public void DrawPlayersCards(System.Windows.Controls.Image[] imgArr, Card[] cardArr)

לציור קלפי השחקן

public void DrawCardMainPack(System.Windows.Controls.Image[] cardPacksImgArr)

לציור החפיסה

בשל העובדה כי עברתי לWPF, נאלצתי לשנות את פונקציות הציור. ב-WPF לא קיים קונטרול של imageBox המאפשר לצייר עליו ישירות. קיים הקונטרול IMAGE שעבורו יש צורך ביצירת ביטמפ במיקום וירטואלי, משום שהו מאפשר רק הצגת תמונה ממקור כלשהו. הרעיון העיקרי של הקוד בפונקציות זהה, לכן אסביר רק קטע קוד אחד.

הקוד מוצג כלהלן: (הקוד הוא לאחר בדיקות הקלפים, זהו הקוד המעשי של הציור)

ראשית, יש ליצור ביטמפ ולקחת ממנו את הגרפיקס בו נצייר את הקלף.

Bitmap bitmap = new Bitmap(60, 90, System.Drawing.Imaging.PixelFormat.Format16bppRgb555);

System.Drawing.Graphics grfx = Graphics.FromImage(bitmap);

אלו הם נתוני הקלף שיש לצייר

int f = (int)cardArr[n].GetFaceVAlue;

CardSuit s = cardArr[n].GetCardSuit;

נקרא לפ' הציור אותה הגדרנו במחלקה Card

CardGraphics.DrawFace(grfx, 0, 0,60 , 90,f, s);

כעת יש ליצור מקטע זכרון זמני עליו נשמור את הציור

MemoryStream ms = new MemoryStream();

bitmap.Save(ms, ImageFormat.Png);

ms.Position = 0; מיקום הציור בזכרון

הקוד הבא יוצא מקור המאפשר טעינת תמונות תוך שימוש בשפת XAML, כלומר בשביל להעביר את הציור לקונטרול image של WPF יש קודם כל להעביר את מה ששמרנו במקטע הזכרון במקור הזה עליו נשמור את הציור ששמרנו לפני כן

BitmapImage bi = new BitmapImage();

bi.BeginInit();

bi.StreamSource = ms;

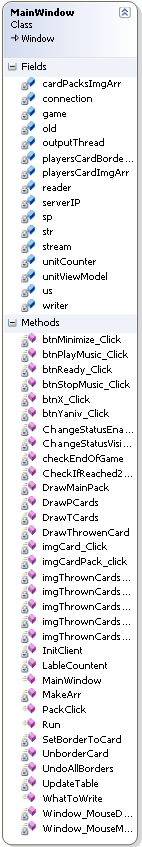
bi.EndInit();

כעת נעביר את המקור עם הציור לimage

imgArr[n].Source = bi;

### מחלקות הקשורות לתקשורת ולחלונות המשחק

#### class MainWindow : Window

המחלקה הזו היא בעצם חלון המשחק ממנו מתופעל המשחק וכמו כן התקשורת.

במחלקה הזו 2 דברים עיקריים- התקשורת והעבודה עם תהליכים.

בעמודים הבאים אתאר כל קטגוריה של פונקציות בנפרד, בנוסף להסברים היכן שדרוש.

משתנים:

private TcpClient connection;

מגדיר לקוח לתקשורת ברשת TCP

private Thread outputThread;

מגדיר תהליך להרצת התקשורת

private NetworkStream stream;

private BinaryWriter writer;

private BinaryReader reader;

מוגדרים זרם, 'קורא' ו'כותב', בדיוק כמו בשרת, לשם העברת המידע

private string serverIP;

משתנה לIP אליו יש להתחבר

private Game game;

מגדיר את אוביקט המשחק (בו מוגדרים השחקנים)

private System.Windows.Controls.Image[] playersCardImgArr

private System.Windows.Controls.Image[] cardPacksImgArr

private System.Windows.Controls.Border[] playersCardBorderArr

מערכים לשם תפעול נוח של ציורי הקלפים

private System.Windows.Point old;

בשל העובדה כי בעיצוב החלון בחרתי להוריד את הבורדר של החלון, בו הופיעו כפתורי המזעור והסגירה, והוא זה שאפשר לשנות את מיקום החלון, נאלצתי ליצור פונקציות משלי לכל הנ"ל. הנקודה הזו מסמנת את הנקודה בה היה החלון כשנלחץ להזזה.

private string str;

המשתנה משמש להעברת ההודעה לפ' שמציגה הודעות בtextBox, המופעלת בעזרת תהליך.

private UnitStruct us;

private UnitViewModel unitViewModel;

private int unitCounter = 0;

המשתנים הנ"ל הם השורות, הקישירית ומשתנה עזר להכנסת שורות הניקוד לטבלה.

SoundPlayer sp;

בעזרת המשתנה הזה מופעלת המוסיקה במשחק.

פונקציות:

#### פונקציות הקשורות לתפעול החלון:

private void Window\_MouseDown(object sender, MouseButtonEventArgs e)

הפ' מופעלת כאשר העכבר לוחץ החלון ושומר את המיקום של הנק' שנלחצה במשתנה OLD. לאחר מכן משתמשים במידע הזה בפ' המזיזה החלון.

private void Window\_MouseMove(object sender, System.Windows.Input.MouseEventArgs e)

הפ' מזיזה את החלון (אחרי בדיקת נכונות הלחיצה וכ"ו)

private void btnStopMusic\_Click(object sender, RoutedEventArgs e)

הפ' עוצרת את המוסיקה

private void btnPlayMusic\_Click(object sender, RoutedEventArgs e)

הפ' מפעילה את המוסיקה

private void btnYaniv\_Click(object sender, RoutedEventArgs e)

הפ' מופעלת כאשר הלרוח לוחץ על כפתור היניב, כלומר מכריז על ניצחון. תפקידה העיקרי של הפ' הוא לשלוח לשרת הודעה על לחיצה.

private void btnReady\_Click(object sender, RoutedEventArgs e)

הפ' מופעלת כאשר השחקן לוחץ על הכפתור ready ז"א מודיע לשרת שהוא מוכן לשחק. הפ' בעיקרה מעבירה לשרת הודעה כי נלחץ הכפתור.

private void btnX\_Click(object sender, RoutedEventArgs e)

תפקיד הפ' לסגור את החלון ולהודיע לשרת שהחלון נסגר, כלומר יש לנתק א תהתקשורת.

private void btnMinimize\_Click(object sender, RoutedEventArgs e)

הפ' ממזערת את החלון

private void UpdateTable()

הפ' מעדכנת את הטבלה לאחר כל סבב בסכום הנק' שכל שחקן קיבל, וכמו כן את סכום הנק' הטוטאלי.

#### פונקציות הקשורות למשחק עצמו:

private void imgCard\_Click(object sender, MouseButtonEventArgs e)

זוהי הפ' אשר מופעלת בלחיצה על אחד מקלפי השחקן (על קונטרול ה-IMAGE ). לכל קונטרול מחובר טג מספר, לפיו הקוד מזהה איזה קלף נלחץ. ראשית בודקים מהו הקלף שנלחץ, לאחר מכן האם זוהי הלחיצה הראשונה או השניה עליו, אם זו השניה זה בעצם UNCLICK, והקלף יוצא מהקלפים הנבחרים. אם זוהי הפעם הראשונה שהוא נלחץ, נבדק האם עוד קלפים נלחצו, ואם כן האם זהו קלף שהוא חלק מוריאציות הלחיצה האפשריות.

public void PackClick(whatClickedMK whatWasClicked)

הפ' הזו מופעלת כאשר יש לחיצה על הקופה או על אחד הקלפים הזרוקים. ראשית בודקים אם יש קלפים לזריקה, אם כן מעבירה את כל הקלפים ממחסנית הזריקה למערך זריקה, את המקום הריק ממלאת בקלפים ריקים (משום שלא ניתן לשלוח משהו לא מוגדר) לאחר מכן מצויירים הקלפים הזרוקים, הם מוצאים ממערך הקלפים של השחקן ומתבצע עדכון מספר קלפיו ונקודותיו. לאחר מכן נשלח לשרת כל המידע הדרוש על הזריקה, לטיפולו.

private void DrawThrowenCard(Card[] throwenCardsArr)

הפ' המציירת את הקלפים הזרוקים. מיקומה כאן ולא במחלקה GAME משום שהיא מציירת מן המערך, המוגדר במחלקה זו. הקוד שלה דומה לזה שהסברתי קודם לכן, בהקשר למחלקה GAME.

private void checkEndOfGame()

הפ' בודקת אם המשחק הסתיים, לאחר כל תור לפי סכום הנקודות. כרשר סכום נקודותיו לש שחקן מסויים עולה על 200 המשחק נגמר. הפ' מודיעה לשרת על סוף המשחק והוא מעביר את ההודעה גם ליריב.

private bool CheckIfReached200

פ' בוליאנית לבדיקה האם עלה סכום הנקודות על 200.

private void SetBorderToCard(int where)

הפ' בעצם מדגישה את הקלף הנבחר

private void UnborderCard(int where)

הפ' מורידה את הדגשת הקלף

private void UndoAllBorders()

הפ' מורידה את הדגשת כל הקלפים שהיו מודגשים

#### הסבר על תהליכים:

כמו שכבר הסברתי, במשחק רק תהליך אין סופי של התקשורת בכל זמן נתון. ידוע, כי בכל פעם יכול לרוץ רק תהליך אחד על המערכת. בעת התכנות של המשחק נתקלתי בבעיה- הייתי צריכה לצע פעולות אחרות הקשורות לחלון ולא יכלתי, משום שלא ניתן לעבור את התהליך להתקשורת. לכן השתמשתי פעמים רבות מאד ב-Invoke

הרעיון המרכזי הוא כזה-

הגדרתי כמה דליגייטים (מצביעים על פ'), באמצעותם אני מקשרת בין פעולת ה-Invoke לבין הפונקציה. הדליגיטים הם כלהלן:

delegate void oneStringDelegate(string str);

מופעל בכדי להציג הודעה ב-textbox שעל מסך המשחק

delegate void DoFunction();

מופעל תמיד כשיש לבצע פונקציה כלשהי שלא דורשת קבלת משתנים כלשהם

delegate void SetLable(WhichLable which, string emount);

הדליגיט משמש להפעלת פ' של עדכון התוויות של סכום וכמות קלפי השחקנים.

delegate void SetEnableStatus(WhichControl which, bool tf);

הדליגיט משמש להפעלת פונקציה אשר משנה את סטטוס הפעולה (פעיל\לא פעיל) של קונטרול כלשהו

delegate void SetVisibilityStatus(WhichControl which,

Visibility vis);

הדליגיט הזה משמש להפעלת פ' אשר משנה את סטטוס ההופעה (מופיע\מוסתר) של קונטרול כלשהו

לאחר מכן מופיע בקוד רבות השורות כלהלן (מבנה כללי):

this.Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal, new הדליגייט(הפונקציה),המשתנים הנדרשים);

זוהי הוראה לדיספצ'ר (האחראי על ניהול התהליכים במערכת) לתת עדיפות להפעלת הדליגיט (ובאמצעותו את הפונקציה בעצם) זה 'מעורר' את המערכת, עד שהפונקציה מתבצעת.

#### הסבר הפונקציות המופעלות באמצעות הדליגייט:

private void LableCountent(WhichLable which,string emount)

הפ' מקבלת תווית מסויימת ואת המידע שיש לשנות בה ומשנה זאת.

private void ChangeStatusEnable(WhichControl which,bool tf)

הפ' משנה את סטטוס הפעולה של קונטרול כלשהו.

private void ChangeStatusVisibility(WhichControl which, Visibility vis)

הפ' משנה את סטטוס ההופעה של קונטרול כלשהו.

private void DrawPCards()

private void DrawTCards()

private void DrawMainPack()

מזמנים פונקציות ציור

public void WhatToWrite(string str)

פ' לכתיבת הודעה בתיבת הטקסט

#### פונקציות הקשורות לתקשורת:

ראשית, הבנאי:

public MainWindow(string IPmessage)

בנוסף לפעולות הגדרת המשחק, הבנאי מקבל את ה-IP שנכתב בשורת הטקסט בתפריט הפתיחה, ועם ה-IP הזה הוא מזמן את הפ' הבאה:

private void InitClient(string IPmessage)

{

בשורות הבאות מוגדרת נקודת היעד של תקשורת ברשת ה-TCP, כלומר הIP של השרת והפורט עליו תרוץ התקשורת. לאחר מכן מוגדר זרם התקשורת בתוך רשת ה-TCP וכמו כן מוגדרים קורא וכותב מאותו הזרם.

connection = new TcpClient(IPmessage, 5000);

stream = connection.GetStream();

writer = new BinaryWriter(stream);

reader = new BinaryReader(stream);

לאחר מכן מוגדר ומופעל התהליך של התקשורת- באמצעות הפונקציה האינסופית RUN

outputThread = new Thread(new ThreadStart(Run));

outputThread.Start();

}

#### public void Run()

הפונקציה run הינה הפונקציה העיקרית והחשובה ביותר של התקשורת בממשק שלי, וזת משום ששם מוגדרות כל הפעולות והמצבים האפשריים. זוהי הפונקציה האחראית על קבלת המסרים מן השרת.

אחלק את הסבר הפונקציה לשני חלקים.

ראשית קטע הקוד המופעל פעם אחת ולאחר מכן הלולאה האינסופית האחראית על מצבי המשחק ופרוש ההודעות.

תחילה, מוגדר מערך לקבלת המידע. לאחר מכן מופעלות שורות ההאזנה (כמו שפורטו בהקדמה על התקשורת בחלק על השרת). לאחר מכן מתקבל סימן השחקן ולפי זה מחליטים האם השחקן מתחיל או לא.

החלק השני:

while (true)

{

הקטע מאזין למסרים מזרם התקשורת, ברגע שמקבל משהו הוא מפרק אותו ומבצע בדיקות.

arForReceive = reader.ReadBytes(39);

frame = new Frame(arForReceive);

if (frame.GetPlayer == this.game.GetPlayer1.MyMark)

כשהמסר הוא כביכול מהשחקן לעצמו- כלומר בין השרת לשחקן בנוגע למשחקו, ולא מידע המועבר מן השחקן השני.

{

if (frame.GetMadeTurn == (byte)MadeTurnMK.turnEnded)

לכאן נכנסים כאשר מהות המסר הוא סיום תור

{

if (frame.GetMessageKey == (byte)connectionMK.nothing)

{

ההודעה היא כלום, כלומר רק נזרקו קלפים ונלקח קלף חדש. במצב כזה הקוד מעדכן את משתני השחקן והיריב, לאחר מכן בודק אם קיים מצב של יניב ומעדכן בהתאם. הקוד בנוי כך שאם בתור מסויים היה לשחקן מצב יניב ובתור שלאחריו אין לו, הכפתור ייעלם בכדי לא לגרום הכרזת שווא.

}

else

הסבב נגמר, כעת נבדוק האם השחקן ניצח או לא.

{

if (frame.GetMessageKey == (byte)connectionMK.playerWonRound)

{

השחקן הכריז יניב וניצח. כל משתניו מתעדכנים, הטבלה מתעדכנת, ומתבצעת בדיקה הא המשחק נגמר או לא. אם לא מופיע הכפתור ready.

}

if (frame.GetMessageKey == (byte)connectionMK.playerGotAsaf)

{

השחקן הכריז יניב והפסיד. המשתנים מתעדכנים, הטבלה מתעדכנת ושוב מתבצעת בדיקה האם המשחק נגמר ואם לא מופיע כפתור ready.

}

}

}

if (frame.GetMadeTurn == (byte)MadeTurnMK.newRound)

הבדיקה הזו אומרת שהתחיל סבב חדש.

{

if (frame.GetMessageKey == (byte)connectionMK.ready\_waitsToOther)

{

השחקן לחץ על מוכנות, השרת קיבל אתהודעתו ושלח לו הודעה כי הוא מוכן וממתין לשחקן השני.

}

if (frame.GetMessageKey == (byte)connectionMK.only1PlayerIsCon)

{

השחקן לחץ על מוכנות, אך השרת מודיע לו כי רק שחקן אחד מחובר ושינסה מאוחר יותר.

}

if (frame.GetMessageKey == (byte)connectionMK.startNewGame )

{

שני השחקנים הודיעו על מוכנות, הקוד מאתחל את כל המשתנםי ומתחיל משחקחדש. כמו כן מתבצעת בדיקה האם השחקן מתחיל או לא (מנצח-מתחיל)

}

}

}

if (frame.GetPlayer == this.game.GetPlayer2.MyMark)

{

כשהמסר הוא מהשחקן השני- כלומר מצב שקרה לשחקן השני והשרת מודיע על כך לשחקן הנוכחי.

if (frame.GetMadeTurn == (byte)MadeTurnMK.turnEnded)

{

כשהמסר הוא מהשחקן השני- כלומר מצב שקרה לשחקן השני והשרת מודיע על כך לשחקן הנוכחי.

if (frame.GetMessageKey == (byte)connectionMK.nothing)

{

כשהמסר הוא מהשחקן השני- כלומר מצב שקרה לשחקן השני והשרת מודיע על כך לשחקן הנוכחי.

}

else

{

כשמהות המסר היא ניצחון או הפסד

if (frame.GetMessageKey == (byte)connectionMK.playerWonRound)

{

השחקן השני לחץ על יניב, אך סכום נקודות השחקן הנוכחי קטן משלו, לכן השחקן הנוכחי ניצח.

}

if (frame.GetMessageKey == (byte)connectionMK.playerLostround)

{

השחקן השני לחץ על יניב וניצח

}

}

}

if (frame.GetMadeTurn == (byte)MadeTurnMK.newRound)

{

התחלת סבב חדש

if (frame.GetMessageKey == (byte)connectionMK.ready\_OtherIsReady)

{

מודיע כי השחקן השני לחץ על מוכנות והוא ממתין לשחקן הנוכחי.

}

}

}

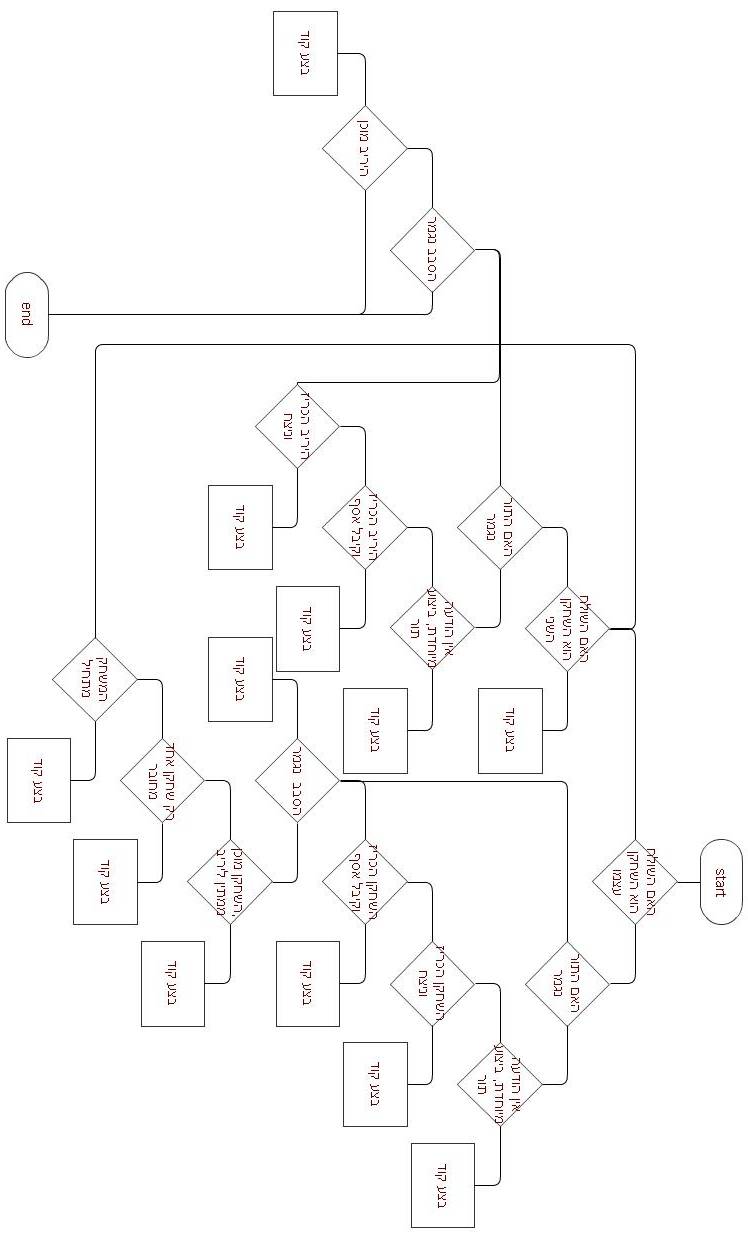
}

}

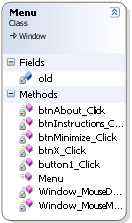
בעמוד הבא מצורף תרשים זרימה של הפונקציה.

יש לציין, כי לאחר ביצוע הקוד השאיפה היא להגיע לסוף הפונקציה.





#### class Menu : Window

זוהי המחלקה של תפריט המשחק.

משתנים:

private System.Windows.Point old;

בשל העובדה כי בעיצוב החלון בחרתי להוריד את הבורדר של החלון, בו הופיעו כפתורי המזעור והסגירה, והוא זה שאפשר לשנות את מיקום החלון, נאלצתי ליצור פונקציות משלי לכל הנ"ל. הנקודה הזו מסמנת את הנקודה בה היה החלון כשנלחץ להזזה.

פונקציות:

private void Window\_MouseDown(object sender, MouseButtonEventArgs e)

הפ' מופעלת כאשר העכבר לוחץ החלון ושומר את המיקום של הנק' שנלחצה במשתנה OLD. לאחר מכן משתמשים במידע הזה בפ' המזיזה החלון.

private void Window\_MouseMove(object sender, System.Windows.Input.MouseEventArgs e)

הפ' מזיזה את החלון (אחרי בדיקת נכונות הלחיצה וכ"ו)

private void btnX\_Click(object sender, RoutedEventArgs e)

תפקיד הפ' לסגור את החלון ולהודיע לשרת שהחלון נסגר, כלומר יש לנתק את התקשורת.

private void btnMinimize\_Click(object sender, RoutedEventArgs e)

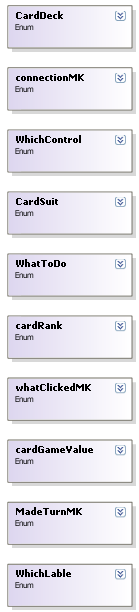
הפ' ממזערת את החלון

private void btnInstructions\_Click(object sender, RoutedEventArgs e)

private void btnAbout\_Click(object sender, RoutedEventArgs e)

שתי הפ' הנ"ל נותנות הוראה לפתוח קובץ בתוכנה Microsoft word.

#### enum type:

הן הלקוח והן השרת משתמשים בייצוג הנ"ל.

אסביר מה תפקידו של כל ייצוג:

ששת הייצוגים הבאים משותפים הן ללקוח והן לשרת:

שלושת הבאים קשורים לקלפים- בקובץ ה-DLL הקלפים מיוצגים ע"י מספרים וממוקמים לפיהם, לכן אלו ה המאפיינים מהם בנוי כל קלף

public enum CardSuit : byte

הסימן של הקלף

public enum cardRank : byte

הערך של הקלף בקובץ ה-DLL

public enum cardGameValue : byte

הערך של הקלף במשחק

בכדי שהקוד יהיה קריא יותר ייצגתי את סוגי ההודעות המועברים בתקשורת בעזרת ייצוג ENUM

public enum connectionMK : byte

public enum MadeTurnMK : byte

public enum whatClickedMK : byte

הייצוגים הבאים שייכים ללקוח בלבד:

public enum CardDeck : int

הערך של האיור (לחפיסת הקלפים) בקובץ ה-DLL

שוב, למען קוד קריא יותר ייצגתי את האפשרויות בהן משתמשות פ' לתפעול מסך המשחק

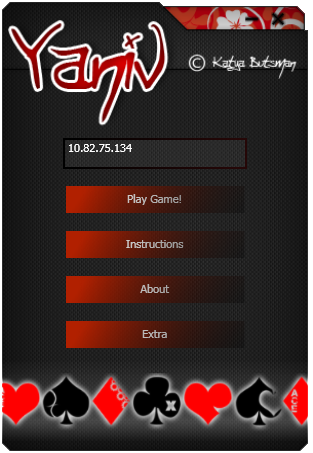
public enum WhichLable : byte

public enum WhichControl : byte

public enum WhatToDo : byte

משמעות הייצוג כמשמעות שמו.



הגראפיקה במשחק:

בתחילת הדרך התחלתי לתכנת את הלקוח ב-windows form. במהלך השנה נחשפתי לטכנולוגיה חדשנית ומדהימה בשם WPF-Windows Presentation Foundation , והחלטתי להעביר את הממשק שלי לשימוש בטכנולוגיה הזו.

מפתחי הטכנולוגיה הזו הצליחו להפריד לחלוטין בין החלק העיצובי בממשק לבין הלוגיקה של הממשק. זוהי טכנולוגית הדור הבא בכל הקשור לפיתוח חלונות לאפליקציות.

הטכנולוגיה מספקת למתכנת מגוון כלים עשיר בעזרתם הוא יכול ליצור חלונות חכמים, ואף לשלב בהם קבצי מדיה ומסמכים.

הטכנולוגיה מרחיבה את הליבה של תחום עיצוב הממשקים עם מערך מקיף של תכונות, פיתוח יישומים הכוללים שפת XAML, גרפיקת דו מימד ותלת מימד, אנימציה, סגנונות, תבניות, מסמכים, מדיה ועוד.

פיסית, הקוד הקשור לעיצוב החלון נרשם בשפת XAML.

בשימוש ב-WPF ניתן ליצור אנימציות ועוד תכונות נוספות ע"י קוד. בנוסף לטכנולוגיה הזו פותח כלי נוסף-micrisoft expression blend 4, המאפשר ליצור אנימציות בקלות ובמהירות, ומגוון האפשרויות בכלי הזה הוא אינסופי.

בשל הגבלת הזמן, לא יצא לי לחקור את הטכנולוגיה לעומק, אך השתמשתי מעט בידע שכן הצלחתי להכיר לשם יצירת חלונות הלקוח:



בעיות לא פתורות והצעות לשיפור

בעיות ללא פתרון בממשק שלי:

1. בשל הרצון לשלב ידע מתחומים שונים והמוגבלות בזמן, ישנה מגבלה משמעותית בתחום התקשורת. יש להפעיל את השרת כל פעם מחדש להתחברות 2 משתמשים, כלומר עבור כל משחק מלא, יש לפתוח מחדש את השרת. לא ניתן לסגור את חלונות הלקוח ולאחר מכן להתחבר שוב.
2. בדיקת ה-IP. לא הצלחתי לחשוב על אלגוריתם שיוכל לבדוק את תקינות מספר ה-IP, וזאת משום שבמחשבים שונים לא קיימת חוקיות לגבי מספר הספרות וכ"ו.
3. סגירת התקשורת- לא הצלחתי לטפל בכל מצבי סגירת התקשורת-למשל כאשר השרת מתנתק- התקשורת נסגרת אך הלקוחות לא יודעים על כך ולכן סביר להניח שהממשק ייתקע\יקרוס. כמו כן לא טיפלתי במצב של סגירת לקוח פתאומית- שלא מכפתור ה-X. המסקנה היא שלסגירה נכונה יש לסגור קודם את הלקוחות, מכפתורי ה-X של חלון המשחק ולאחר מכן את השרת.
4. בתפריט הפתיחה של המשחק, כאשר לוחצים עם הסמן על תיבת הטקסט, הממשק זז לנקודה הנלחצת. לדעתי הבעיה היא בעובדה כי בשל העובדה שהורדתי את שורות הכותרת לשם העיצוב שלי, והוספתי פונקציות משלי לתזוזה משהו שם לא פועל טוב, לכן מתרחשת התזוזה. לא הצלחתי למצוא לכך פתרון.

המלצות לשיפור:

1. על בסיס הבעיה הראשונה, באה גם ההצעה לשיפור- אם היה לי זמן נוסף הייתי מרחיבה את האפשרויות- יוצרת שרת אחד בו ניצן ליצור כמה משחקים בו-זמנית ואף ליצור חלוקה לחדרי משחק.
2. תכננתי להוסיף אפשרות בה הלקוח יוכל להרשם באמצעות האי-מייל, ראשית כך שניצחונותיו ופרטיו ישמרו במסד נתונים, ושנית לקבל באי-מייל ההרשמה את IP השרת אליו צריך להתחבר-חשבתי אף על דרך למימוש אך בשל מגבלות הזמן לא הספקתי.
3. בהקשר למשחק-הוספת כלים לנוחות המשתמש, כגון צ'אט, ואף הרחבת אלגוריתם וחוקי המשחק.

רפלקציה אישית ותודות

תהליך כתיבת הפרויקט היווה חוויה מאד משמעותית עבורי. המקום בו עמדתי בשנה שעברה היה מקום של בלבול, חוסר מוטיבציה ואף רצון לעזיבה. היום, בעקבות החוויה הזו, מבטי מופנה רק קדימה. למדתי רבות על עצמי בתהליך המפרך הזה- למדתי אודות התמדה, מחויבות, רכשתי כלים רבים ללמידה עצמית, למדתי כי עבודה קשה וממושכת- השעות הרבות שהעברתי בעבודה על הפרויקט מתחילת השנה ועד סופה- מביאה לתוצאה השווה את כל המאמץ. היום אני יכולה לעמוד ולהסתכל אחורה ולומר בגאווה שזהו הפרויקט שלי, אני יכולה לשים עליו את שמי ולדעת שזהו הפרויקט שלי, השקעתי בו מחשבה רבה ומאמץ אדיר. בתחילת השנה לא חשבתי שאצליח ליצור פרויקט ברמה כה גבוהה, לא ידעתי שיש ברשותי ידע כה רחב, ובעת כתיבת הפרויקט הבנתי איך לממש את הידע שלי, ואם הידע לא קיים בידי, למדתי כי תמיד ישנה למאגרי מידע עצומים ברחבי הרשת וכי שתמיד או להרחיב את הידע שלי באופן עצמאי וללמוד איך לעשות דברים שלא ידעתי קודם.

הדבר העיקרי ביותר שלמדתי הוא המשפט הבא-

ברצוני להודות למורה להנדסת תוכנה- אנטולי פיימר, שבלעדיו לא הייתי מצליחה ליצור פרויקט כה מושקע, ואם בכלל.

הוראות לפתיחת הממשק

בכדי להפעיל את הממשק, ראשית יש להפעיל את השרת- זהו הקובץ YanivServer, הנמצא בתיקיה EXE.

בכדי להפעיל את הלקוח, יש לפתוח את הקובץ YanivGameWindow, גם הוא נמצא בתיקיה EXE, הקובץ יפתח תפריט, בו ניתן לקרוא את הוראות המשחק, לשם כניסה לחלון המשחק יש להקליד את כתובת ה-IP של שרת היעד (הכתובת המדויקת, אחרת התכנית לא תפעל) וללחוץ על Play Game!.

כנאמר, ניתן לפתוח את הממשק על אותו מחשב או על מחשבים נפרדים, כרשום בפרק 'תיאור משתמשי המערכת'.

על הליך המשחק ניתן לקרוא בקובץ הInstructions, אליו ניתן להגיע מהתפריט הראשי.

(ההוראות מצורפות גם בנספחים.)

ביבליוגרפיה

אתר בו השתמשתי רבות-

http://msdn.microsoft.com

http://www.csharpfriends.com/Articles

http://www.codeproject.com

http://www.wpftutorial.net/wpfdevtools.html

http://www.kirupa.com/blend\_wpf/index.htm

http://www.microsoft.com/design/toolbox/school

http://www.inforall.net

הספר עיצוב תוכנה מבוסס עצמים- המרכז להוראת המדעים. האוניברסיטה העברית בירושלים

נספחים:



### game instructions:

**First of all, make sure there is an open server for your game.**

**Game Start**:

Enter the servers IP into the text box above the button 'Play Game!'.

After entering the IP, click 'Play Game!'.

Now, the window of the game will be opened,as you enter to the game window, click 'Ready'.

If the other player is not connected yet, wait a few seconds and try again.

If you clicked first, wait for the other player to connect. And then the game will begin. You play first.

If the other player clicked first, click 'Ready' and the game will begin. He plays first.

**Game Objective**  -  Accumulating the lowest amount of points

**Card Values**  -  Each card is worth a certain amount of points defined by the following table:

|  |  |
| --- | --- |
| King, Queen, Jack | 10 |
| 1 - 10 | the card's number |

**Game Play**  -  On your turn you must throw one or more cards and take exactly one card

|  |  |  |
| --- | --- | --- |
| **What can be thrown?** | **Examples** | |
| Single card | Khttp://www.yanivgame.com/img/suit_s.gif 10http://www.yanivgame.com/img/suit_d.gif | |
| Series | 5http://www.yanivgame.com/img/suit_h.gif5http://www.yanivgame.com/img/suit_s.gif5http://www.yanivgame.com/img/suit_c.gif      Qhttp://www.yanivgame.com/img/suit_d.gifQhttp://www.yanivgame.com/img/suit_c.gif      2http://www.yanivgame.com/img/suit_c.gif2http://www.yanivgame.com/img/suit_s.gif2http://www.yanivgame.com/img/suit_d.gif2http://www.yanivgame.com/img/suit_h.gif | |
| Straight | 7http://www.yanivgame.com/img/suit_s.gif8http://www.yanivgame.com/img/suit_s.gif9http://www.yanivgame.com/img/suit_s.gif      10http://www.yanivgame.com/img/suit_d.gifJhttp://www.yanivgame.com/img/suit_d.gifQhttp://www.yanivgame.com/img/suit_d.gifKhttp://www.yanivgame.com/img/suit_d.gif | |
| **What can be taken?** | | |
| If a single card was thrown | you can take it | you can always take a card from the deck |
| If a series/ straight was thrown | you can take any card |

If you wish to throw a series, click the **lowest** card first:

You can **unclick** cards, but only the last you clicked- if it’s a series, you can't unclick the previous card**.**

**Winning the round**  -  When the sum of the cards in your hand is 7 or lower a button will appear allowing you to call YANIV.

The meaning of this call is that you believe you have the lowest hand of all players.  
If you have the lowest sum of points, you win, and the other player loses.

If the other player has the same or lower sum, the caller receives a fine and is referred to as ASAF.

**Scoring**  -  At the end of every round players receive points according to the following table:

|  |  |
| --- | --- |
| The winner | 0 |
| ASAF | 30 point fine |
| Others | Sum of the hand |

**Winning the game**  -  When a player reaches more than 200 points they lose.  
The winner is the player with the lowest score.

Special rule: If a player reaches exactly 200 points, they are automatically reduced back to 150.