הצעת נושא לפרויקטים לטכנאים במגמת הנדסת תוכנה

א. הנושא: "יניב" משחק קלפים

ב. שם המנחה : ינאי סנד

ג. שם התלמיד :תומר מורסקי ת.ז. : 322622382

ד. עבודה ביחיד

ה. אופי עבודת הפרויקט : תכנון ובניית משחק

ו. מקום ביצוע הפרויקט **:** מכללת אורט חולון

ז. תיאור נושא המשחק :

המשחק מתנהל במבט מלמעלה. במרכז הלוח תמוקם הקופה והקלף הנוכחי של המשחק – ערימה/ קופה גלויה. מטרת המשחק היא להחזיק בסכום הקלפים הנמוך ביותר מבין המשתתפים במשחק. כל סיבוב מסתיים כאשר אחד השחקנים מכריז כי ברשותו קלפים בערך כולל של 8 ומטה. בסוף כל סבב מתווספות לשחקן הנקודות שצבר בסיבוב הנוכחי (מחושב על פי ערך הקלפים כמפורט בהמשך). הנקודות שצבר מאז שהתחיל את סבב המשחקים מופיעות לצד הקלפים שלו.

חלוקת הקלפים :

כל שחקן מקבל תחילה חמישה קלפים ושאר הקלפים נותרים כקופה. ישנו קלף פתיחה אשר נותר פתוח לעיני כולם בתחילת המשחק – ערימה גלויה. ערך כל קלף הוא כערך המספר שלו (קלף 7 – 7 נקודות..) קלפי המלוכה – 10 ג'וקר – 0 .

מהלך המשחק :

כל שחקן בתורו חייב להשליך לפחות קלף אחד מידו, ולקחת קלף מהקופה או את הקלף האחרון מערימת הזריקות הגלויה. תחילה ישליך את הקלף שברצונו להיפטר ממנו, ורק לאחר מכן ייקח קלף נוסף. השלכת הקלף תתבצע ע"י לחיצה על הקלף\ קלפים הרלוונטיים לזריקה. לאחר בחירת כל הקלפים לזריקה ילחץ השחקן על המקור ממנו מעונין לקחת קלף (קופה או ערימה גלויה) לאחר הלחיצה על המקום (הקופה/ קופה גלויה) יוסרו הקלפים שנבחרו מידו של השחקן ויושלכו אל הערימה הגלויה. בכדי לזרוק רצף יש לבחור את הקלפים ע"פ סדר עולה ברצף.

השחקן יוכל להשליך יותר מקלף אחד מהמקרים הבאים:

- סדרה עולה של לפחות 3 שלושה קלפים כשהם מאותה צורה. (לדוגמה : 6,7,8 כשהצורה שלהם משותפת).
- זוג או יותר מאותו קלף לדוגמה 9,9 או 8,8,8 .

לאחר כל זריקה (ללא קשר למספר הקלפים שנזרקו , על השחקן לקחת קלף מהקופה או מהחבילה הגלויה (כפי שהוסבר מעלה) כך שלאחר זריקה מרובה מספר הקלפים של השחקן פוחת, ואיתו עולה הסיכוי להוריד את הסכום הכולל של הקלפים ולנצח במשחק.

סבב ניצחון :

בכל סיבוב ,במידה וסכום הקלפים של שחקן קטן מ – 7 יש ברשותו האופציה להכריז "יניב". כאשר שחקן מכריז זאת , כל שאר השחקנים חושפים את הקלפים בידיהם והשחקן שמחזיק בסכום הנמוך ביותר הוא המנצח. אם הסכום הנמוך ביותר (או סכום שווה לו) נמצא בידי שחקן אחר (שלא הכריז "יניב") אז שחקן זה הוא המנצח, מצב זה מכונה "אסף" .

מסך ניצחונות:

במסך זה יופיע השחקן אשר מאזן הנקודות שלו היה הטוב ביותר מבין כל המשחקים ששוחקו אי פעם. מאזן נקודות מחושב באופן הבא : מספר הנקודות שצבר שחקן ברצף משחקים מסוים חלקי מספר המשחקים בסבב, ז"א, אם בסבב מסוים צבר שחקן 12 נקודות ובסבב זה שוחקו שני משחקים, משמע מאזן הנקודות שלו יהיה 6. אם מאזן הנקודות שלו הוא הנמוך ביותר שהושג, הוא יירשם בטבלת השיאים של המשחק.

פירוט הדרישות:

Must have-

- מסך פתיחה
- ניקוד
- מסך המשחק הראשי (המסך עליו מתנהל המשחק)
- כמה קלפים מחזיק כל משתתף ביד בכל רגע נתון ז"א לא את סכום הקלפים אלא כמות קלפים 1-5 (פתרון באמצעות אנימציה – הצגה של הקלפים).
- כפתור להכרזת יניב (יתאפשר רק אם סכום הקלפים מאפשר זאת)
- סכום הקלפים של כל שחקן ברגע חשיפת הקלפים
- המערכת תנהל משחק מול משתתף / משתתפים נוספים ותבצע את קבלת ההחלטות הדרושה בכדי לשחק בצורה הטובה ביותר בהתאם למצב הנוכחי.

Important-

- דף הוראות משחק
- מסך הפסד
- מסך ניצחון
- תזוזה ברגע סימון קלף מסיים (מעיין אנימציה לצורך אינדיקציה של בחירה)
- תזוזה ברגע משיכת קלף מהקופה
- סידור קלפי השחקן בתצוגה מהקטן לגדול

Nice to have-

- רשימת השחקנים המובילים
- צלילים ברגע ניצחון, משיכת קלף, הפסד, זריקת קלף
- אפשרות להשהות משחק
- אפשרות לשנות את המינימום של סכום הקלפים להכרזת "יניב"

<u>מחלקות :</u>

Card

Joker

Scoring

Pile – קופה

Open pile –ערימת קלפים גלויה

Open screen

Game

Game screen

Player

Instructions

Drawable

gameGraphic

gamePanel

playerActions

– **Card**

<u>תכונות:</u>

- מספר (1-10)
- צורה – לב, תלתן, עלה ,מעוין. (מספרים בין 1-4 ייצגו את הצורה)
- צבע – אדום שחור
- **Serial** – מספר סידורי 1-54
- **Image** – תמונה של הקלף. ישמש בעתיד לציור הקלף

המחלקה **card** תייצג כרטיס במשחק. המחלקה תכיל את כל הנתונים על כרטיס.

Joker – יורש מ **card**

- לא מכיל תכונות נוספות חוץ מ **card** . המספר שלו יהיה 0 , צבע – אדום או שחור והצורה היא **null** .

ג'וקר הוא קלף מיוחד, הוא בעל מספר 0 ויכול "להצטרף" לכל זריקה מרובה ז"א בעת זריקת רצף של מספרים, יכול הג'וקר להשלים את הרצף ולאפשר זריקה שלו (דוגמא : יש ברשות השחקן ג'וקר, 2 – לב 4 – לב. בכדי להיפטר מרצף של 3 יוכל השחקן להשתמש ב ג'וקר)

Scoring?? – שם השחקן, ניקוד

– **Openscreen**

תכונות:

- כפתור – **start**
- כפתור – **instructions**
- ??כפתור – שיאים ??

מחלקה זו תאפשר "לצייר" את מסך הפתיחה. היא תכיל את כל האובייקטים שיש במסך בית ומתקם אותם על **panel**

כפתור התחלת משחק, כפתור הוראות, כפתור שיאים

Drawable- ממשק

פעולה **draw**

GamescreenPanel –

- קופה- במרכז
- קלף נוכחי/ ערימה גלויה - מרכז מתחת לקופה
- השחקן – דרום
- שחקנים נוספים – מזרח מערב
- כפתור יניב – צד ימין תחתון

במחלקה זו אצייר את כל האובייקטים על הפאנל אבצע באמצעות פעולת **paint component** . ארוץ על מערך של טיפוסים **draeable** ואצייר את כולם. המחלקה תמקם את האובייקטים על הפאנל.

playerActions – מחלקה אשר תגיב בהתאם לכל פעולה שיבצע המשתמש (תאזין ללחיצות)

gameGraphic – אבנה פאנל מסוג **gameScreenPanel** ואוסיף אותו ל פריים. למעשה אגרום לציור של כל האובייקטים במשחק אשר נמצאים על לוח המשחק. במחלקה זו אבנה את ה**gameLoop** .

Player–

תכונות:

- רשימה של **card**
- שם
- סכום הקלפים

Pile– קופה

תכונות:

- תור של קלפים

רשימה של קלפים, פעולות: שליפת קלף.

HomePile – מערך שמסודר לפי המספר הסידורי של קלף. מטרת מחלקה זו היא לאפשר גישה לקלפים כשהם מסודרים לפי אינדקס = מספר סידורי.

- מערך של קלפים

OpenPile – ערימה פתוחה

<u>תכונות:</u>

- מחסנית של קלפים

קופה גלויה היא קופה. רשימת קלפים (שנזרקו). פעולות : שליפת קלף (אם השחקן בחר לקחת את הקלף האחרון), הכנסת קלף (זריקה של קלף ע"י שחקן).

Instructions –

<u>תכונות:</u>

- הוראות **string**

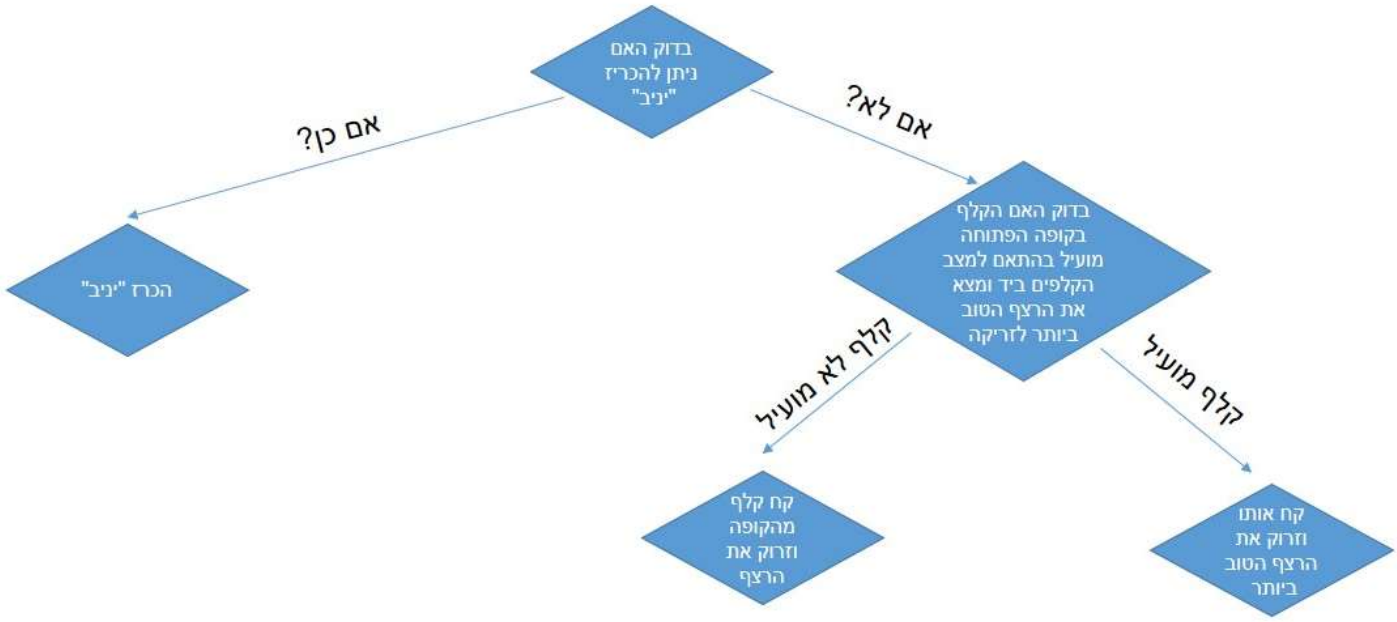
תיאור אלגוריתם המהווה את לב הפרוייקט:

האלגוריתם שמהווה האלגוריתם העיקרי והמסובך בפרוייקט הוא אלגוריתם קבלת ההחלטות.
במשחק "יניב" בו משחקים מול מחשב (ולא מול שחקן נוסף) יש צורך באלגוריתם שידע לקבל החלטות מתאימות בהתאם למצב הקלפים בלוח ובהתאם למצב הקלפים ביד.

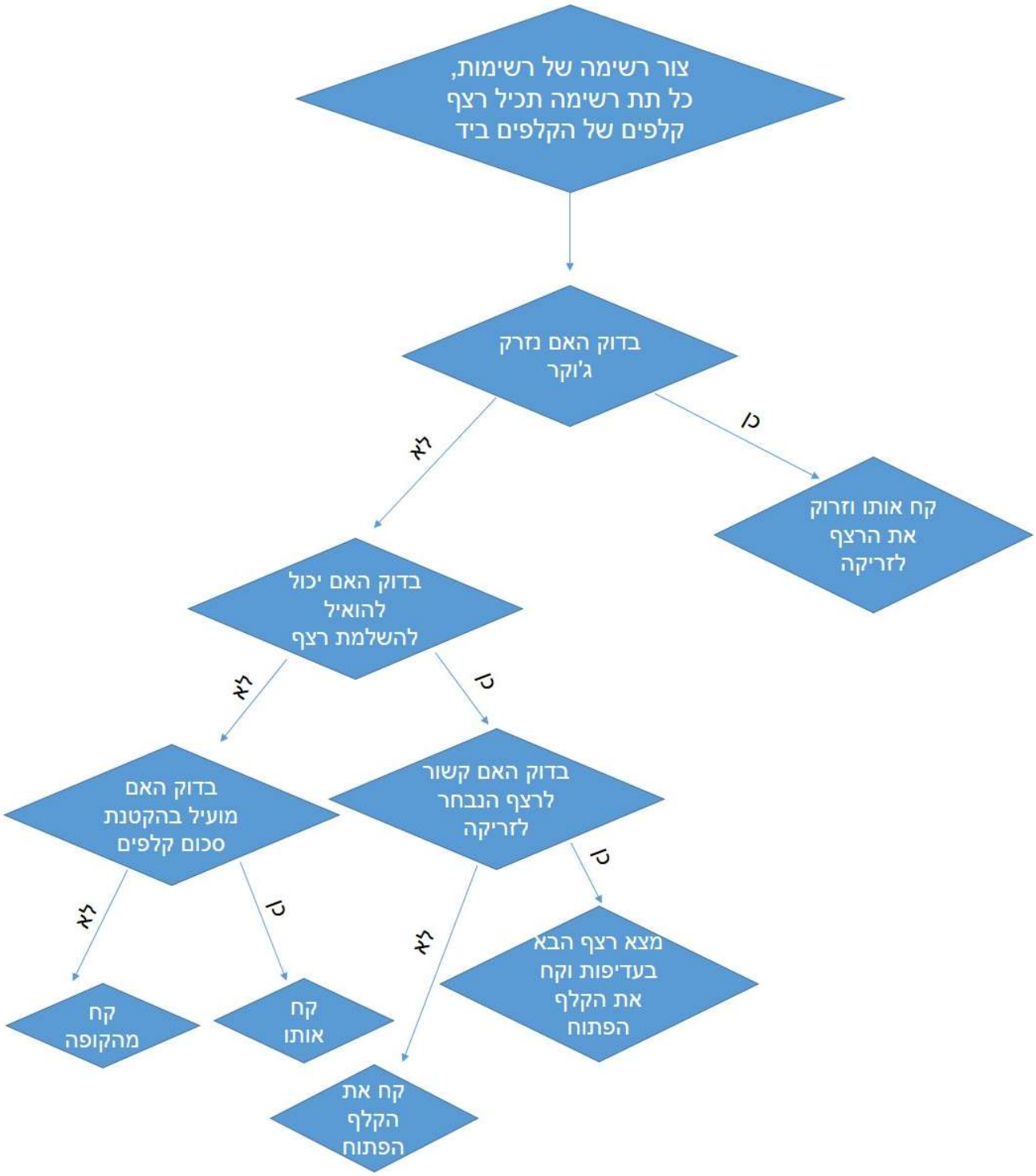
הבעיה שעליה עונה האלגוריתם :

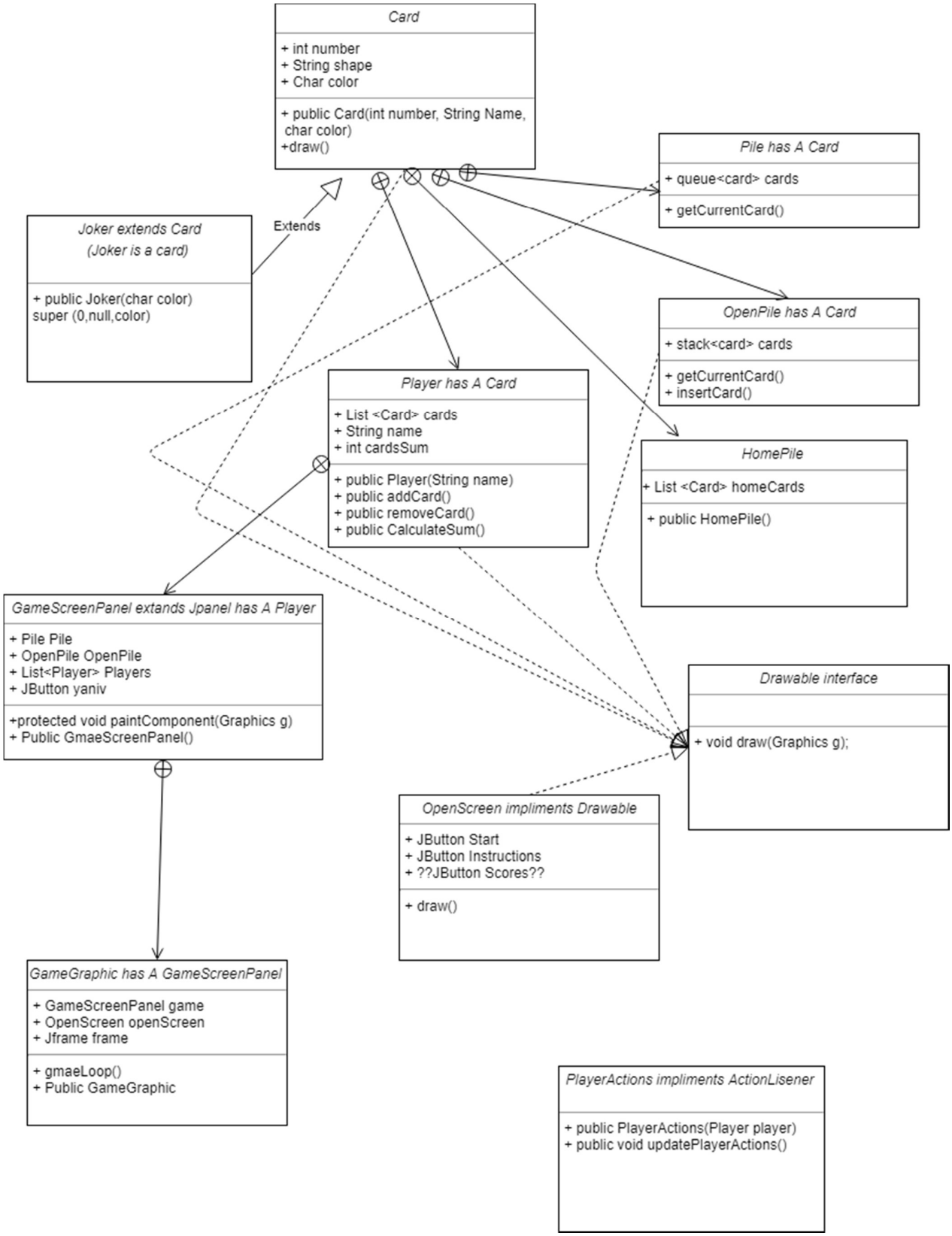
בחירת קלף/ קלפים שעל המחשב לזרוק ו/או לקחת כך שיבצע את המהלך הטוב ביותר עבורו. האלגוריתם ידע, ע"פ עקרונות המשחק, איזה קלף הכי כדאי שיזרוק וייקח במידת האפשר. לכאורה הדבר אולי יראה כפשוט – המטרה היא להשיג את סכום הקלפים הנמוך ביותר כך שעדיף תמיד לזרוק את הקלף הגדול ביותר, אך הדבר אינו נכון. ישנם מהלכים אשר יגרמו להורדה משמעותית יותר של סכום הקלפים כמו למשל זריקה של רצף קלפים (כפי שפורט מעלה במהלך המשחק). לדוגמה : אם ברשותו של שחקן הקלפים 3,6,7,7,10 , האלגוריתם יעדיף לזרוק דווקא את רצף הקלפים 7,7 על פני המספר הגדול ביותר. דוגמה נוספת היא בעבור הקלפים 5,5,8,8,3 , בעבור דוגמה זו האלגוריתם יחפש מעבר לרצף הקליפים הארוך ביותר הוא יחפש גם את הגבוה מבניהם (כמו שפורט קודם המטרה היא להיפטר מסכום הגדל ככל שניתן) לכן במקרה זה נעדיף לזרוק את הרצף 8,8 . (<u>האלגוריתם יודע להתייחס כמובן גם לרצפים עולים מאותה צורה למשל – **6,7,8** תלתן, ולזהות אותו כרצף אפשרי לזריקה בנוסף את השחקן מחזיק ביד את הקלפים 6,7 תלתן ובקופה הפתוחה ייחשף הקלף 5 או 8 תלתן- האלגוריתם יעדיף לקחת את קלף זה בכדי להשלים את הרצף ולזרוק ערך גדול יותר בתור הבא</u>).
בנוסף האלגוריתם ידע לקבל החלטות לגבי כדאיות לקיחת הקלף מהקופה או הקופה הפתוחה, ז"א מה ההסתברות שייקח קלף טוב יותר, לדוגמא אם הקלף בקופה הפתוחה הוא 2 והקלף הכי נמוך שהשחקן מחזיק ביד הוא גבוה מ – 2 , הסיכוי שסכום הקלפים ביד ירד גבוה משמעותית אם ייקח את הקלף בקופה הפתוחה (2) מאשר ייקח את הקלף בקופה הסגורה (בה יוכל לקבל אמנם גם קלפים נמוכים יותר מ-2 כמו ג'וקר ואס אך קיים סיכוי גבוה יותר שיקבל קלף גבוה מ – 2).

תרשים זרימה של התרחשויות :


בדוק האם ניתן להכריז "ניב"
אם כן?
אם לא?
הכרז "ניב"
בדוק האם הקלף בקופה הפתוחה מועיל בהתאם למצב הקלפים ביד ומצא את הרצף הטוב ביותר לזריקה
קלף לא מועיל
קלף מועיל
קח קלף מהקופה וזרוק את הרצף
קח אותו וזרוק את הרצף הטוב ביותר

פירוט האלגוריתם:




Card
+ int number
+ String shape
+ Char color
+ public Card(int number, String Name, char color)
+draw()
Pile has A Card
+ queue<card> cards
+ getCurrentCard()
Joker extends Card
(Joker is a card)
+ public Joker(char color)
super (0,null,color)
Extends
OpenPile has A Card
+ stack<card> cards
+ getCurrentCard()
+ insertCard()
Player has A Card
+ List <Card> cards
+ String name
+ int cardsSum
+ public Player(String name)
+ public addCard()
+ public removeCard()
+ public CalculateSum()
HomePile
+ List <Card> homeCards
+ public HomePile()
GameScreenPanel extands Jpanel has A Player
+ Pile Pile
+ OpenPile OpenPile
+ List<Player> Players
+ JButton yaniv
+protected void paintComponent(Graphics g)
+ Public GmaeScreenPanel()
Drawable interface
+ void draw(Graphics g);
OpenScreen impliments Drawable
+ JButton Start
+ JButton Instructions
+ ??JButton Scores??
+ draw()
GameGraphic has A GameScreenPanel
+ GameScreenPanel game
+ OpenScreen openScreen
+ Jframe frame
+ gmaeLoop()
+ Public GameGraphic
PlayerActions impliments ActionLisener
+ public PlayerActions(Player player)
+ public void updatePlayerActions()

אבני דרך:

– 15.11.18

בניית המחלקות קופה וקופה פתוחה + קלף וג'וקר

– 25.11.18

טיפול בקלפי המשחק – סידור תמונות של קלפי המשחק (לכל מספר יש 4 קלפים = תמונות פרט לג'וקר) יש צורך לדאוג לעריכת תמונות הקלפים כך שיתאימו לשימוש במשחק.

– 30.12.18

גרפיקה בסיסית של המשחק, תצוגה בסיסית של מסך המשחק ומסך הפתיחה

– 5.1.19

מסך ניצחון והפסד בסיסיים

– 10.1.19

בחירת שיטת ניקוד ויישומה במשחק + שמירת הנתונים לקובץ לשם הצגת שיאני נקודות ושמם

-15.2.19

בניית האלגוריתם החכם, חשיבה לעומק על מצבים אפשריים לניצחון והדרך הטובה ביותר לנצח.

– 30.2.19

שיפור גרפיקת המשחק, הוספת אלמנטים ואנימציות. שיפור נראות המשחק

תמחור זמנים :

בניית המחלקות הבסיסיות כגון קופה, קופה פתוחה, קלף, ג'וקר ושחקן – **5 שעות**.

טיפול בתמונות הדרושות לייצוג הקלפים במשחק – **10 שעות**.

בניית האלגוריתם החכם- **7 שעות** .

יצירת האזנות ותגובות בהתאם ללחיצה – **5-6 שעות.**

יצירת מסך הבית/ פתיחה – **4 שעות**.

יצירת מסך המשחק הסטטי (ללא אנימציות וכשהוא מכיל אלמנטים בסיסיים) – **4 שעות**.

שיפור נראות המשחק (למשל תמונה של כל מצב נתון של קלפים של שחקן – אם ברשותו 4 קלפים יופיעו 4 קלפים הפוכים..) – **4-5 שעות**.

הוספת אנימציה של קלף נגרר לקופה או להיפך – **5-6 שעות**.

הוספת מסך הפסד ניצחון – **4 שעות** .

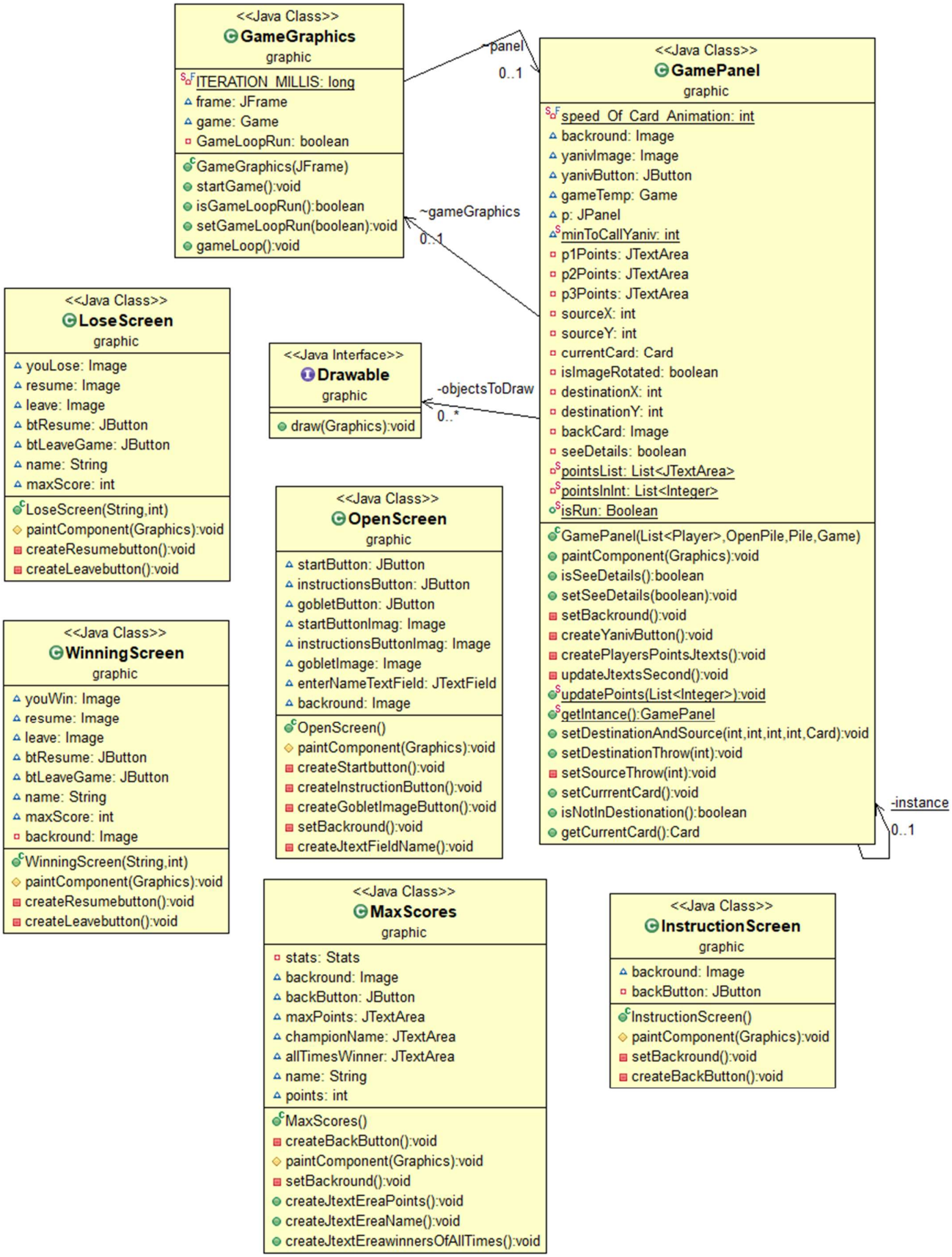
סאונד – **6 שעות ?**

החלטה על שיטת ניקוד ויישומה במשחק – **4 שעות.**

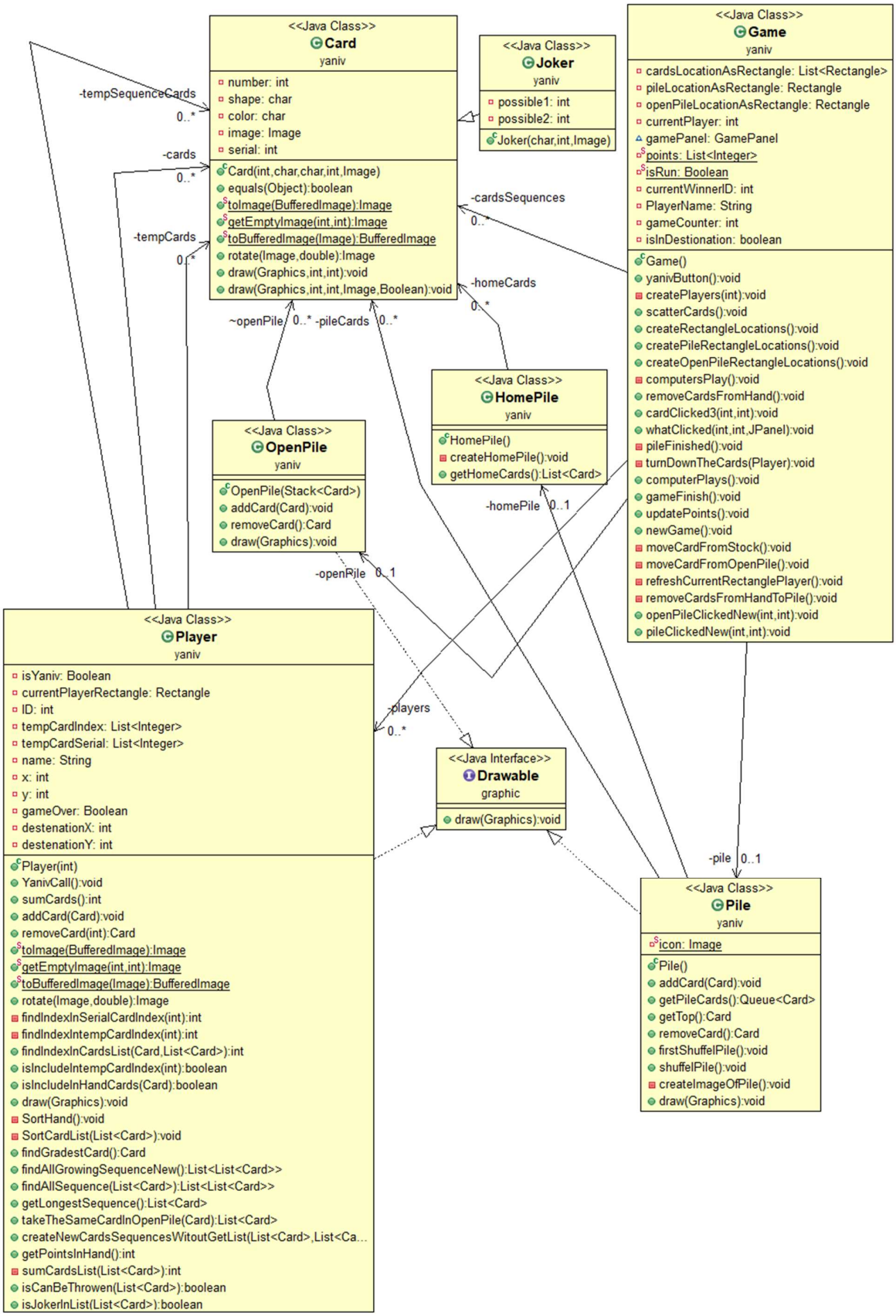
אפשרות לשינוי הגדרות משחק – מינימום של הכרזת "יניב" , זמן תור ... – **10 שעות .**

טיפול בקלף ג'וקר **– 3 שעות ד**

תרשי מחלקות גרפיות:



תרשים מחלקות משחק:



מבני נתונים בפרוייקט :

1. המחלקה homePile - המחלקה אחראית על אתחול ראשוני של קלפי המשחק. מופע של מחלקה זו נוצר פעם אחד בלבד במהלך הרצת המשחק מטרתה היא לאחסן את חפיסת הקלפים. מבנה הנתונים בו בחרתי לאחסן את הקלפים הוא **רשימה**. בפעולה הבונה שלה היא למעשה יוצרת את חבילת הקלפים – יוצרת 54 קלפים ומאתחלת אותם בערכים המתאימים : מספר, צורה ותמונה אשר תייצג את הקלף. ההיגיון שעומד מאחורי מחלקה זו הוא לחסוך בזמן ריצה, טעינת התמונות מהזיכרון מתבצעת פעם אחד וללא חזרות מיותרות.

```
public HomePile() {
        this.homeCards = new LinkedList<Card>();
        createHomePile();
    }
    /**
     * the method will create the home pile. will adds 54 cards to the list.This list
changes never!
     * every card in the list has his image so it can be draw.
     */
    private void createHomePile(){
        char shape = 's';
        char color = 'b' ;
        int number = 1;
        for(int i = 1; i < 53; i++){
            try{
                Image icon = ImageIO.read(new File("pictures/"+i+".png"));
                Card c = new Card(number, shape, color, i, icon);
                homeCards.add(c);
                }catch (IOException e){
                    System.out.println("file not found");
                }
            if(shape == 's'){
                shape = 'h';
                color = 'r';
            }
            else if(shape == 'h'){
                shape = 'd';
                color = 'r';
            }
            else if (shape == 'd'){
                shape = 'c';
                color = 'b';
            }else{
                    shape = 's';
                    color = 'b';
            }
            /**
             * every 4 cards the number of the card changes
             */


            if(i % 4 == 0)
                number ++;
            //Card c = new Card(number, shape, color, i,icon);
            }
        /**
         * create the jokers- special cards, and add them to the list
         */
        try{
        Image icon = ImageIO.read(new File("pictures/"+53+".png"));
        Joker j = new Joker('r', 53, icon);
        homeCards.add(j);
        icon  = ImageIO.read(new File("pictures/"+54+".png"));
        j = new Joker('b', 54, icon);
        homeCards.add(j);
        }catch(IOException e){
            System.out.println("file not found");
        }

    }
```

2. המחלקה pile – מחלקה המייצגת את קופת המשחק. במחלקה זו בחרתי להשתמש ב"**תור**" בכדי לאחסן את הקלפים, זאת בשל הדימיון הרב בין אופן פעולתו של התור לבין קופה במציאות. השימוש במבנה זה הוא בעיקר שליפה אך יש צורך גם בפעולת הכנסה לתור ראשית בשביל התיחול הראשוני של הקופה באמצאות ה homePile ושנית משום שלאחר שהקופה נגמרת יש צורך לערבב את הקופה הפתוחה ולהכניס לקופה את הקלפים מחדש (באמצאות הכנסה לתור) . בתחילתו של כל משחק הקופה מתערבבת ומוכנסת לתוך התור (הכוונה לרשימה של homePile), לאחר מכן הקלפים מחולקים למשתתפי המשחק ומרגע זה יישלף בכל פעם קלף יחיד מהחלק העליון של הקופה לשחקן המתאים.

```
public class Pile implements Drawable {
	private static final int X_start = 470;
	private static final int Y_start = 100;
	private static  Image icon;
	private Queue<Card> pileCards;
	private HomePile homePile;

	public Pile() {
		this.pileCards = new LinkedList<Card>();
		homePile = new HomePile();
		firstShuffelPile();
		createImageOfPile();
	}

public void addCard(Card card){
		this.pileCards.add(card);
	}

	public Card getTop() {
		return this.pileCards.peek();
	}
	/**
	 *
	 * @return the top of the queue and remove it from the queue
	 */
	public Card removeCard(){
		return this.pileCards.remove();
	}

/**
	 * does not destroy the list, copy to another list and shuffle the copied list,
then copy the shuffled list to the pileCards
	 * now the pileCards include a queue of shuffled cards
	 * Shuffle all the cards and put them in the pileCards
	 * @param pileCards include the  cards
	 */
	public void firstShuffelPile(){
		List<Card> tempCards = new LinkedList<Card>();
		for(int i = 0; i < this.homePile.getHomeCards().size(); i++){
			tempCards.add(this.homePile.getHomeCards().get(i));
		}
		Collections.shuffle(tempCards);
		for(int i = 0; i < tempCards.size(); i++){
			this.pileCards.add(tempCards.get(i));
		}

	}

	public void shuffelPile() {
		List<Card> tempCards = new LinkedList<Card>();
		for(int i = 0; i < this.pileCards.size(); i++){
			tempCards.add(this.pileCards.remove());
		}
		Collections.shuffle(tempCards);
		for(int i = 0; i < tempCards.size(); i++){
```

```
                this.pileCards.add(tempCards.get(i));
            }
        }
```

3. המחלקה openPile – מחלקה במייצגת את הקופה הפתוחה במשחק. במחלקה זו בחרתי להשתמש ב"**מחסנית**" בכדי לאחסן את הקלפים וגם זאת בשל הדימיון הרב בין אופן פעולתו של תור לבין קופה פתוחה : הקלף האחרון שנזרק – הוא הזמין לשליפה (או ל"הצצה" משום שבמציאות הוא גלוי לעיני כל המשתתפים).

```
public class OpenPile implements Drawable {
        Stack<Card> openPile;
        private static final int xStart = 470;
        private static final int yStart = 250;

        public OpenPile() {
            this.openPile = new Stack<Card>();
        }

/**
         * add a card to the stack
         * @param card
         */
        public void addCard(Card card) {
            this.openPile.push(card);
        }
        /**
         *
         * @return the top of the stack and remove it from the stack
         */
        public Card removeCard() {
            return this.openPile.pop();
        }
```

4. המחלקה player – מחלקה המייצגת שחקן. במחלקה זו בחרתי להשתמש **ברשימה** על מנת לאחסן את הקלפים של שחקן ברגע נתון (הקלפים שמחזיק ביד). בנוסף, במחלקה זו ישנה פעולה האחראית על חלק מאלגוריתם ה AI , מטרתה היא למצוא את כל הרצפים בעלי אותו מהספר אשר נמצאים "ביד" של אותו שחקן. אופן האחסון של הרצפים בעבור שחקן ספציפי הוא באמצעות **רשימה של רשימות**. כל תת רשימה ברשימה הגדולה תכיל רצף אשר מוחזק ביד (בין איבר אחד לחמישה איברים), ז"א בעבור הקליפם 2,2,3,4,5 הרשימה הגדולה תכיל ארבע תתי רשימות בעבור כל קלף יחיד תיהיה רשימה וכן בעבור הרצף.

```
public class Player implements Drawable{
        private List<Card> cards;
.
.
.
.
.
.
.
}

/**
         * only for same numbers
         * @param HandCards
         * @return a list of all the sequences in hand (only of same number). every list
in the list of lists will include a sequence of cards
         */
        public List<List<Card>> findAllSequence (List<Card> HandCards) {
            int checkNumber;
            List<List<Card>> cardsSequences = new LinkedList<List<Card>>();
            List<Card> cardsLowSequence = new LinkedList<Card>();
            cardsLowSequence.add(HandCards.get(0));
```

```
            for (int i = 0; i < HandCards.size() - 1 ; i++){
                  checkNumber = HandCards.get(i).getNumber();
                  if (HandCards.get(i + 1).getNumber() == checkNumber) {
                        cardsLowSequence.add(HandCards.get(i + 1));
                  } else {
                        cardsSequences.add(cardsLowSequence);
                        cardsLowSequence =  new LinkedList<Card>();
                        cardsLowSequence.add(HandCards.get(i + 1));
                  }
            }
            cardsSequences.add(cardsLowSequence);// needs to add the last one because
not added
            //System.out.println(cardsSequences);
            return cardsSequences;

      }
```

5. במחלקה הגרפית gamePanel - בחרתי להשתמש **ברשימה** של פרטים "הניתנים לציור" (יורשים מ drawable) . כל פריט אשר צריך להצטייר על הפאנל, יתווסף לרשימה זו. לאחר מכן בפעולת ה paintComponent רצים על רשימה זו ומפעילים את פעולת ה draw בעבור כל אחד מהאיברים ברשימה.

```
public class GamePanel extends JPanel {
      private static final int speed_Of_Card_Animation = 1; // recommended 1. anyway
not above 3!!
      private List<Drawable> objectsToDraw;
.
.
.
.
.
.
}

@Override
      public void paintComponent(Graphics g) {

.
.
.
.
            for(Drawable drawable : objectsToDraw){
                  drawable.draw(g);
            }
.
.
.
.
.

      }
```

ניתוח פונקציות עיקריות בפרויקט :

```
public List<List<Card>> findAllSequence (List<Card> HandCards) {
            int checkNumber;
            List<List<Card>> cardsSequences = new LinkedList<List<Card>>();
            List<Card> cardsLowSequence = new LinkedList<Card>();
            cardsLowSequence.add(HandCards.get(0));
            for (int i = 0; i < HandCards.size() - 1 ; i++){
                  checkNumber = HandCards.get(i).getNumber();
                  if (HandCards.get(i + 1).getNumber() == checkNumber) {
                        cardsLowSequence.add(HandCards.get(i + 1));
                  } else {
                        cardsSequences.add(cardsLowSequence);
                        cardsLowSequence =  new LinkedList<Card>();
                        cardsLowSequence.add(HandCards.get(i + 1));
                  }
            }
            cardsSequences.add(cardsLowSequence);// needs to add the last one because not
added
            //System.out.println(cardsSequences);
            return cardsSequences;

      }
```

ניתוח :

מטרת הפונקצייה :

מציאת כל הרצפים אשר מורכבים מאותו מספר ברשימה מסויימת של קלפים אשר נשלחת לפונקציה.

הפונקציה תרוץ בלולאה כמספר האיברים ברשימה. דבר זה מתאפשר משום שהקלפים מסודרים ברשימה בסדר עולה, אילולא הקלפים ברשימה היו מסודרים בסדר עולה סדר הגדול של יעילות הפונקציה היה גדול יותר. למעשה הפונקציה רצה על הקלפים ומכניסה אותם לרשימה – כל עוד מספר הקלף זהה לקודם שהוכנס. ברגע שנתקלים בקלף בעל מספר שונה, מוסיפים את הרשימה הקודמת לרשימה של רשימות ויוצרים רשימה חדשה עם הקלף בעל המספר השונה, כך הלאה עד לסיום הלולאה. בסוף הפונקציה למעשה נקבל רשימה שבה כל איבר יהיה רשימה של הקלפים מאותו מספר. פונקצייה זו משרתת בין היתר את האלגוריתם החכם AI .

היעילות של הפונקציה היא O(n) .

```
/**
         * Responsible to create the tempCardIndex and put in the array the cards that needs to
draw "up"
         * in addiction adds from the tempCardSerial.
         * @param card
         * @param id
         * @param index
         */

        // in temp cards index will be the index of the card in hand
        public void chooseUP(Card card, int id, int index) {
                     tempSequenceCards.add(card);
                     tempCards.add(card);
                     tempCardIndex.add(index);
                     tempCardSerial.add(cards.get(index).getSerial());// adds the serial
instead of the index
       }
        public void chooseDOWN(Card card, int id, int index) {
                     if(findIndexIntempCardIndex(index) != -1 &&
findIndexInSerialCardIndex(card.getSerial()) != -1){
                           tempCardIndex.remove(findIndexIntempCardIndex(index));
                           tempCardSerial.remove(findIndexInSerialCardIndex(card.getSerial()));
                           tempCards.remove(findIndexInCardsList(card,tempCards));
```

```
tempSequenceCards.remove(findIndexInCardsList(card,tempSequenceCards));
                }else{
                        System.out.println("not found");
                }
        }
```

ניתוח :

מטרת הפונקציות:

מטרתם של שתי הפונקציות היא למעשה מנוגדת זו לזו. מטרתה של chooseUP היא למעשה לגרום לקלף הנבחר להצטייר מעל שאר הקלפים – מעין אינדיקציה לכך שזהו הקלף הנבחר ע"י השחקן. לעומתה , הפונקציה chooseDOWN מאפשרת לשחקן להתחרט על הבחירה שביצע, ולבחור קלפים שונים לזריקה. בפועל הפונקציה גורמת לקלפים שנבחרו " לחזור בחזרה ליד " כלומר לאחר שתופעל עליהם הפונקציה chooseDOWN הם לא יהיו מיועדים יותר לזריקה.

בפונקציה chooseUP כל הפונקציות הן ביעילות של o(1) . בנוסף בפעולה עצמה לא מבוצעות לולאות או פקודות שחוזרות על עצמן ולכן היעילות של פונקציה זו היא o(1) . הפונקציה למעשה מוסיפה למערך קלפים אשר מייצג את הקלפים שנבחרו ע"י השחקן, את הקלף שנבחר (במידה וחוקי כל קלף חוקי לבחור, ז"א להרים מהיד אך בעת הזריקה יכול להיות שהפקודה תסורב משום שהקלפים שהורמו לא משלימים רצף). ההתייחסות למערך זה מתבצעת בחלק הגרפי – יש טיפול בקלפים אשר נמצאים במערך זה כך שיצוירו באופן המתאים.

הפונקציה chooseDOWN קוראת גם כן לפונקציות נוספות אך בשונה מ chooseUP היא משתמשת בפונקציה נוספת אשר יעילותה היא O(n) , למעשה בכדי להסיר קלף מסויים על הפונקציה לדעת באיזה אינדקס הקלף נמצא במערך של chooseUP. בכדי לגלות את אינדקס זה יש לעבור על המערך של chooseUP לגלות את האינדס של הקלף הרצוי למחיקה ולאחר מכן לבצע מחיקה שלו ממערך זה בכדי שלא יצויר כקלף שנבחר.

לאחר שהשחקן יסיים את בחירת הקלפים החוקיים לזריקה הקלפים ייזרקו מידו והמערך האחראי על זכירת הקלפים שנבחרו – יתרוקן.

```java
/**
	 * check if regular sequence can be throw
	 * @param list
	 * @return
	 */
	public boolean isCanBeThrowen(List<Card> list) {
		if(!isJokerInList(list)){
			SortCardList(list);
			if(list.size() <= 1){
				return true;
			} else if (list.get(0).getNumber() == list.get(1).getNumber()) {
				for(int i = 1; i < list.size(); i++) {
					if(list.get(i).getNumber() != list.get(i - 1).getNumber()) {
						return false;
					}
				}
				return true;
			} else if (list.get(1).getNumber() == (list.get(0).getNumber() + 1) &&
(list.get(1).getShape() == list.get(0).getShape()) &&
					list.size() > 2) {
				for (int i = 1; i < list.size(); i ++) {
					if (list.get(i).getNumber() != list.get(i - 1).getNumber() + 1
|| list.get(i).getShape() != list.get(i - 1).getShape()) {
						return false;
					}
				}
				return true;
			} else {
				return false;
			}
		} else {
			if (isCanBeThrowen(getListWithoutJoker(list))) {
				return true;
				// else will be onle for sequence that need to be complete by joker
(2 4 5 | 2 3)
			} else {
				SortCardList(list);
				int countJoker = countJoker(list);
				List<Card> tmpList = getListWithoutJoker(list);
				if (tmpList.size() >= 2) {
					for (int i = 0; i < tmpList.size() - 1; i++) {
						if(tmpList.get(i).getNumber() == tmpList.get(i +
1).getNumber() - 1
								&& tmpList.get(i).getShape() ==
tmpList.get(i + 1).getShape()){
							continue;
						} else if (tmpList.get(i).getNumber() == tmpList.get(i
+ 1).getNumber() - 2
								&& tmpList.get(i).getShape() ==
tmpList.get(i + 1).getShape() && countJoker >= 1){
							countJoker --;
							continue;
						}else if (tmpList.get(i).getNumber() == tmpList.get(i +
1).getNumber() - 2
								&& tmpList.get(i).getShape() ==
tmpList.get(i + 1).getShape() && countJoker >= 2) {
							countJoker --;
							continue;
						}else {
							return false;
						}
					}
					return true;
				}
```

```
                }
            }
            return false;
        }
```

ניתוח :

מטרת הפונקציה:

מטרתה של הפונקציה היא להחזיר ערך בוליאני האם הרצף שנבחר לזריקה הינו חוקי או לא. במידה וחוקי תחזיר true אחרת false . הפונקציה היא רקורסיבית, ישנו טיפול שונה בעבור ג'וקרים ולכן כל התנאים שמופיעים בתחילת הפונקציה מתאימים כשאין בקלפים ג'וקרים, לכן ברגע שיימצא ג'וקר בקלפים הפונקציה תיקרא בשנית אך יקוצץ מרשימת הקלפים – הג'וקרים, ז"א היא תיקרא שוב עם ערך שונה מהערך שנקראה תחילה. במקרה הראשון בו אין ג'וקרים בחבילה (בדיקת הג'וקרים היא ביעילות o(n)) יעילות הפונקציה תהיה o(n) וזאת שבנוסף להפעלת הפונקציה למניית הג'וקרים ישנה ריצה על כל הקלפים ביד (ז"א לולאה שתרוץ כמספר הקלפים ביד). במקרה בו ישנם גוקרים ביד ולאחר שהפונקציה נקראה בשנית ולא ענתה על התנאי הראשון, היא תיכנס לתנאי השני אשר מכסה מצב בו ג'וקר צריך להשלים חלק חסר בסרייה. בחלק זה יש מנייה של מספר הגוקרים ביד (O(n)) ושוב ריצה על הקלפים ביד O(n) . לכן סדר גודל היעילות של פונקצייה זו הוא O(n) .

```
/**
        * remove the card from the pile and add it to the player
        * @param x of click
        * @param y of click
        */
       public void pileClickedNew(int x , int y) {
             if(x > pileLocationAsRectangle.getX() &&  x < (pileLocationAsRectangle.getX() +
pileLocationAsRectangle.width)
                         && y > pileLocationAsRectangle.getY() && y <
(pileLocationAsRectangle.getY() + pileLocationAsRectangle.height)){
                   if(this.pile.getPileCards().isEmpty()){
                         pileFinished();// fill the pile again
                   }
                   createRectangleLocations();
                   if(players.get(1).isCanBeThrowen(players.get(1).getTempSequenceCards())){
                         moveCardFromStock();// first draw the animation and then remove from
pile. if first remove card
                         //from pile and only then draw the animation, the card is going to
be drawn is not the correct one
                         this.players.get(1).addCard(this.pile.removeCard());
                        removeCardsFromHandToPile();

                   }else{
                         turnDownTheCards(this.players.get(1));
                   }
                   if(this.pile.getPileCards().isEmpty()){
                         pileFinished();// fill the pile again
                   }

             }
       }
```

ניתוח :

מטרת הפונקציה:

טיפול בלחיצה על הקופה הסגורה. במידה והיא אכן נלחצה ישנו סיכוי שהיא ריקה ולכן יש צורך לבדוק זאת לפני הפעלת פונקציות על הקופה שכן לא תתקבל שגיאה. במידה ויהיה צורך להפעיל את pileFinished (שיעילותה O(n)) יש להתחשב בכך בחישוב היעילות של הפונקציה. בנוסף בכדי לבדוק האם בכלל ניתן לבצע את הזריקה, ז"א האם הקלפים שנבחרו עומדים בתנאים נפעיל את isCanBeThrowen שיעילותה היא O(n) . במידה והקלפים ניתנים לזריקה את תפעיל את הפעולה moveCardFromStock, יעילותה של פעולה זו היא O(n^2) וזאת משום שהיא רצה בלולאת while עד אשר המיקום של הקלף יגיע ליעד שהוגדר. יעילותה של turnDownTheCards היא O(1) .ולכן סדר גודל היעילות של הפונקציה הוא O(n^2) .

```
public void openPileClickedNew(int x, int y) {
            if(x > openPileLocationAsRectangle.getX() &&  x <
(openPileLocationAsRectangle.getX() + openPileLocationAsRectangle.width )
                        && y > openPileLocationAsRectangle.getY() && y <
(openPileLocationAsRectangle.getY() + openPileLocationAsRectangle.height )){
                  createRectangleLocations();
                  if(players.get(1).isCanBeThrowen(players.get(1).getTempSequenceCards())){
                        moveCardFromOpenPile();
                        this.players.get(1).addCard(this.openPile.removeCard());
                         removeCardsFromHandToPile();
                  }else{
                        turnDownTheCards(this.players.get(1));
                  }
            }
      }
```

ניתוח:
מטרת הפונקציה:
טיפול בלחיצה על הקופה הגלויה. בשונה מהפונקציה pileClickedNew, אין צורך לבדוק האם ניתן לקחת קלף מהקופה הגלויה (ז"א אין צורך לבדוק האם הקופה ריקה או לא משום שהיא תמיד מלאה בלפחות קלף אחד). שאר הפונקציה מזכיר את הפונקציה pileClickedNew . גם בפונקציה זו ישנה בדיקה האם הרצף שנבחר לזריקה תקין O(n) , בנוסף מופעלת הפונקציה moveCardFromOpenPile שאחראית על החלק הגרפי (תנועת הקלף על הלוח) – O(n^2) . לכן סדר הגודל הכללי של הפונקצייה הוא o(n^2) .

```
/**
       * Responsible to create the cardsLocationAsRectangle for player 1 (the user),
       *  uses for recognise which card has pressed
       */

      public void createRectangleLocations() {
            /**
             * run according to the cards size and create the rectangle location list
             */
            List<Rectangle> cardsLocationAsRectangleTemp = new LinkedList<Rectangle>();
            int x = players.get(1).getX();
            for(int i = 0; i < players.get(1).getCards().size() - 1; i++, x += 45){
                  Rectangle location = new Rectangle(x, players.get(1).getY() -
final_Value_fix_Rectangle_Location, 45, 120);
                  cardsLocationAsRectangleTemp.add(location);
            }
            Rectangle location = new Rectangle(x, players.get(1).getY() -
final_Value_fix_Rectangle_Location, 70, 120);
            cardsLocationAsRectangleTemp.add(location);
            this.cardsLocationAsRectangle = cardsLocationAsRectangleTemp;
      }
```

ניתוח :
מטרת הפונקציה:
ליצור רשימה של מלבנים אשר מייצגים את המיקומים של הקלפים בלוח. הפונקציה רצה כמספר הקלפים ביד ומעדכנת בכל ריצה את המערך בעבור כל שחקן. יעילותה של פונקציה זו היא O(n) .

```java
/**
       * Responsible for the animation of card from stock (pile) to hand
       * call to isNotInDestionation()
       * call to repaint()
       */
      private void moveCardFromStock() {
            gamePanel.setSeeDetails(false);
        gamePanel.setDestinationAndSource(pile.getxStart(), pile.getyStart(),
players.get(currentPlayer).getDestenationX(),
players.get(currentPlayer).getDestenationY(), pile.getTop());
        //System.out.println("to draw before" + gamePanel.isToDraw());
        isInDestionation = gamePanel.isNotInDestionation();
        while (isInDestionation) {
            isInDestionation = gamePanel.isNotInDestionation();
            gamePanel.repaint();
           try {
                Thread.sleep(speed_Of_card_Take_animatoin);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
       // System.out.println(gamePanel.isToDraw() + " draw me 4");
        gamePanel.setCurrrentCard();
    }
      /**
       * Responsible for the animation of card from stock (pile) to hand
       * call to isNotInDestionation()
       * call to repaint()
       */
      private void moveCardFromOpenPile() {
            gamePanel.setSeeDetails(true);
        gamePanel.setDestinationAndSource(openPile.getxStart(), openPile.getyStart(),
                  players.get(currentPlayer).getDestenationX(),
players.get(currentPlayer).getDestenationY(), openPile.getOpenPile().peek());
        //System.out.println("to draw before" + gamePanel.isToDraw());
        isInDestionation = gamePanel.isNotInDestionation();
        while (isInDestionation) {
            isInDestionation = gamePanel.isNotInDestionation();
            gamePanel.repaint();
           try {
                Thread.sleep(speed_Of_card_Take_animatoin);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
       // System.out.println(gamePanel.isToDraw() + " draw me 4");
        gamePanel.setCurrrentCard();
    }
```

ניתוח :

מטרת הפונקציות :

לגרום לאנימציה של תנועת הקלפים : בין הקופה הפתוחה ליד או בין הקופה הרגילה ליד (יד = יד השחקן הנוכחי).

שתי הפונקציות פועלות בצורה דומה : נותנות ערכים בעבור תחילת התנועה וסיום התנועה. בתוכן לולאה פנימית שתדאג לקדם את הציור הבא ממיקום המקור למיקום היעד כך שבכל פעם שהקלף יצטייר יראה כאילו הוא נגרר לכיוון היעד. כל ציור מחדש של קלף קורא ל repaint שקוראת ל paintComponent שיעילותה היא O(n) ולכן היעילות הכוללת של הפונקציות היא O(n^2) .

```java
private void computersPlay() {
            Thread thread = new Thread(new Runnable() {
                  @Override
                  public void run() {
                        for (int i = 0; i < 2; i++){
                              if (currentPlayer == 0 || currentPlayer == 2){
                                    try {
                                          Thread.sleep(2000);
                                    } catch (InterruptedException e) {
                                          e.printStackTrace();
                                    }
                                    if(players.get(0).getGameOver()) {
                                          continue;//change
                                    }
                                    computerPlays();
                                    gamePanel.repaint();
                                    currentPlayer++;
                                    currentPlayer = currentPlayer % 3;
                              }
                        }
                  }
            });
            thread.start();
      }
```

ניתוח:

מטרת הפוקנציה:

ניהול התורות מול המחשב, והפעלת האלגוריתם החכם. הפעולות יחזרו על עצמן בלולאה כמספר השחקנים פחות 1 (להפעיל את הפעולות רק בעבור שחקני המחשב). בכל לולאה יש קריאה ל `computerPlays` O(n^2) וגם ל `repaint` O(n) ולכן סדר הגודל של יעילות פעולה זו היא O(n^2 * m). כש n מבטא את מספר הקלפים ביד בעוד m מבטא את מספר השחקנים בלוח (שחקני המחשב) .

```java
public void computerPlays(){
            /**
             * first check if can call yaniv. If does - call yanivButton and end the round
             * if not -
             */
            if(this.players.get(currentPlayer).sumCards() <= amount_TO_Say_YANIV) {
                  yanivButton();
                  return; // do not keep the function After call to yanivButton.
            }
            this.cardsSequences = players.get(currentPlayer).getLongestSequence();
                  /**
                   * if the card in the open pile appear in the cards in hand - take it
                   * need to check if the there is a sequence with the same number - if in
hand there is 2,4,4,5,5
                   * the computer will prefer to throw the 5,5 but if he take the card from
the open he will have
                   * 5,5,5 which is better.will call a special method that create the
cardSequence again but that
                   * time the sequence to throw will be the 4,4.
                   */

      if(players.get(currentPlayer).isCompleteSequence(openPile.getOpenPile().peek())) {
                        this.cardsSequences =
players.get(currentPlayer).takeTheSameCardInOpenPile(openPile.getOpenPile().peek());
                        moveCardFromOpenPile();
                        Card cardToAdd = openPile.getOpenPile().pop();
                        for (int i = 0; i < this.cardsSequences.size(); i++) {
                              openPile.addCard(this.cardsSequences.get(i));

      players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
                        }
                        players.get(currentPlayer).addCard(cardToAdd);
                  } else if (openPile.getOpenPile().peek().getNumber() <= 5 &&
openPile.getOpenPile().peek().getNumber() <
                              players.get(currentPlayer).getCards().get(0).getNumber() ||
                              players.get(currentPlayer).findGradestCard().getNumber() >=
openPile.getOpenPile().peek().getNumber()
                              && openPile.getOpenPile().peek().getNumber() <= 5){
                              moveCardFromOpenPile();
                              Card cardToAdd = openPile.getOpenPile().pop();
                              for (int i = 0; i < this.cardsSequences.size(); i++) {
                                    openPile.addCard(this.cardsSequences.get(i));

      players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
                        }
                        players.get(currentPlayer).addCard(cardToAdd);
                  }else{
                        for (int i = 0; i < this.cardsSequences.size(); i++) {
                              openPile.addCard(this.cardsSequences.get(i));

      players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
                        }
                        moveCardFromStock();
                        players.get(currentPlayer).addCard(pile.removeCard());
                  }


            // the sound will be after all the checks
            try{
                  File  url = new File("sounds/draw.wav");
                  Clip clip = AudioSystem.getClip();
                  AudioInputStream ais = AudioSystem.getAudioInputStream(url);
                  clip.open(ais);
```

```
                clip.start();
            }catch(LineUnavailableException | UnsupportedAudioFileException | IOException
murle){
                System.out.println(murle);
            }
            if(this.pile.getPileCards().isEmpty()){
                pileFinished();// fill the pile again
            }
        }
```

ניתוח:

מטרת הפונקציה:

ניהול האלגוריתם החכם אשר מקבל את ההחלטות בעבור ביצוע מהלכי המחשב. ראשית נבדק האם ניתן לקרוא יניב אם כן נקראת הפעולה yanivButton שיעילותה היא O(n log n) . לאחר מכן מזומנת הפעולה למציאת הקלפים הטובים ביותר לזריקה getLongestSequence O(n^2). לאחר מכן מזומנת הפונקציה isCompleteSequence שיעילותה היא O(n^2) גם כן. לאחר מכן ישנה קריאה ל moveCardFromOpenPile שיעילותה היא O(n^2) ולאחר מכן לולאה שרצה במספר הקלפים ברשימה לזריקה בכדי להכניס ת האיברים בה לקופה הפתוחה. לכן סדר הגודל היעילות של פעולה זו הוא O(n^2).

```
/**
     * when someone wins that method called. It check who is the winner and play the yaniv
sound of a winning
     */
    public void yanivButton() {
        Player winner = Collections.min(players,new Comparator());
        // check if the count works well. the player with the lowest sum is the winner
        this.currentWinnerID = winner.getID();
        try{
            File  url = new File("sounds/yaniv.wav");
            Clip clip = AudioSystem.getClip();
            AudioInputStream ais = AudioSystem.getAudioInputStream(url);
            clip.open(ais);
            clip.start();
        }catch(LineUnavailableException | UnsupportedAudioFileException | IOException
murle){
            System.out.println(murle);
            System.out.println("exeption");
        }
        gameFinish();
        return;


}
```

ניתוח:

מטרת הפונקציה:

למצוא את השחקן המנצח בסיבוב הנוכחי, ז"א השחקן שסכום הנקודות שלו הוא הנמוך ביותר. בשביל למצוא שחקן זה בחרתי להשתמש ב

Collections.min . יצרתי מחלקה בשם Comparator אשר משווה בין הסכומים של השחקנים ברשימה. במחלקה Comparator יש שימוש בפונקצייה sumCards() אשר עוברת על כל הקלפים ברשימה מסויימת וסוכמת אותם יעילותה היא o(n) ויעילות החיפוש כולו היא O(n log n) . הפעולה gameFinish היא ביעילות של o(n) ולכן סדר גודל היעילות של הפונקציה הוא O(n log n) .

```
public List<List<Card>> findAllGrowingSequenceNew () {
            List<Card> grow = new LinkedList<Card>();
            int tempIndex = -1;
            List<List<Card>> cardsSequences = new LinkedList<List<Card>>();
            grow.add(this.cards.get(0));
            for (int i = 0; i < this.cards.size() - 1; i++) {
                  if ((tempIndex = findNextCardSequence(this.cards, i + 1, cards.get(i))) !=
-1) {
                        grow.add(cards.get(tempIndex));
                  } else {
                        cardsSequences.add(grow);
                        grow = new LinkedList<Card>();
                        grow.add(cards.get(i + 1));
                  }
            }
            cardsSequences.add(grow);
            return cardsSequences;

             }
```

ניתוח:

מטרת הפונקציה:

הפונקציה מחזירה רשימה של רשימות, כאשר כל תת רשימה מהווה רצף או חלק ממנו. הפונקציה רצה פעם אחת על כל רשימת הקלפים, דבר זה מתאפשר בזכות שימוש בפעולת עזר findNextCardSequence אשר מוצאת את הקלף הבא ברצף במידה וקיים , יעילותה היא o(n) משום שהיא רצה על רשימת הקלפים ביד של השחקן. לכן סדר גודל היעילות של פונקציה זו הוא O(n^2) משום שהיא רצה n פעמים (n מהווה את מספר הקלפים) ומבצעת n פעמים פעולה שהסיבוכיות שלה היא o(n) .

```
public boolean isCompleteSequence(Card card) {
            // boolean isComplete = false;
             if (isIncludeInHandCards(card)) {
                   return true;
             }
             List<List<Card>> growingSequence = findAllGrowingSequenceNew();
             for (int i = 0; i < growingSequence.size(); i++) {
                   if (growingSequence.get(i).size() > 1) {
                         if (growingSequence.get(i).get(0).getNumber() == card.getNumber() +
1 &&
                                     growingSequence.get(i).get(0).getShape() ==
card.getShape() ||

growingSequence.get(i).get(growingSequence.get(i).size() - 1).getNumber() == card.getNumber() -
1
                                     &&
growingSequence.get(i).get(growingSequence.get(i).size() - 1).getShape() == card.getShape() ){
                                 return true;
                         }
                   } else {
                         return false;
                   }
             }
             return false;

 }
```

ניתוח:
מטרת הפונקציה:
הפונקציה מחזירה ערך true את הקלף שנשלח אליה (הקלף הגלוי בקופה הגלוייה) יכול להשלים רצף כלשהו בקלפים אשר מחזיק השחקן ביד, ז"א אם בידו הקלפים 2,2,5,8 והקלף הגלוי הוא 2 אז הפונקציה תחזיר true, דוגמא נוספת היא אם בידו של השחקן הקלפים 4,5,9,1 (הסימן של 4 ו 5 הוא לב לצורך הדוגמא), והקלף בקופה הגלויה הוא 3 או 6 לב, הפונקציה תחזר true גם כן. הפונקציה ראשית קוראת ל `isIncludeInHandCards` O(n) , תנאי זה עונה על השאלה הראשונה בתיאור הנ"ל – האם יש בקלפים ביד קלף שמספרו זהה לקלף הגלוי, אם כן תחזיר true . לאחר מכן נשתמש ב `findAllGrowingSequenceNew` שיעילותה היא O(n^2) . לאחר מכן נרוץ על רשימה זו ונבדוק האם הקלף הפתוח יכול להשלים את אחת מתת הרשימות. אם כן – תחזיר true . לכן סדר הגודל של פעולה זו היא O(n^2).

```
public List<Card> getLongestSequence() {
            List<List<Card>> cardsSequences = findAllSequence(this.cards);//if there is 2
sequences with the same length
            List<List<Card>> cardsGrowingSequences = findAllGrowingSequenceNew();
            //it will take the one with the bigger number
            List<Card> longestSequence = Collections.max(cardsSequences,new
CompareSequence());
            cardsGrowingSequences.add(longestSequence);// add to the list of list
            //the biggest sum list from the regular sequence
            longestSequence = Collections.max(cardsGrowingSequences,new
CompareSequence());//then call to collections.max
            //to get again the biggest sum now.
            if (longestSequence.size() < 2) {
                  longestSequence.remove(0);
                  //before choosing the biggest card, need to check if the card in the open
pile is better to take
                  longestSequence.add(findGradestCard());// if the biggest sequence is only 1
card
                  //so prefer to throw the biggest card
            }
            /*
             * if there is a sequence of 2 jokers they will not be thrown. Instead, send a new
cards in hand without the
             * jokers. if there is only one joker he won't be thrown!If there is only 2 joker
say yaniv!
             */

            if (findGradestCard().getNumber() > sumCardsList(longestSequence)) {
                  // if the biggest card amount is bigger than the sequence amount, prefer to
throw the biggest card
                  longestSequence.clear();
                  longestSequence.add(findGradestCard());
            }
            return longestSequence;

}
```

ניתוח:
מטרת הפונקציה:
הפונקציה מחזירה רשימה של הקלפים הכדאיים ביותר לזריקה. הפונקציה מפעילה שתי פונקציות עזר, האחת מוצאת את כל הרצפים בעלי אותו מספר בעוד השנייה מוצאת את הרצפים האפשריים שניתן לזרוק. getLongestSequence מטרת פונקציה זו היא למצוא מבין כל הרצפים (גם אותו מספר וגם עולים) את הרצף בעל הערך הגבוה ביותר (שכמובן אותו כדאי לזרוק).
הפעולה `findAllSequence` היא ביעולות O(n) והפעולה `findAllGrowingSequenceNew` היא ביעילות O(n^2) . לאחר מכן נפעיל את `Collections.max` פעמיים: ראשית למצוא את הרצף אשר מורכב מאותו מספר בעל הסכום הגדול ביותר. לאחר מכן נצרף אותו לרשימת הרשימות שמייצגת את הרצפים העולים ונפעיל שוב את `Collections.max` . לאחר הפעם השנייה שתופעל פונקציה זו נוכל לדעת בוודאות שהרשימה שבא אנו מחזיקים היא בעלת הערך הכי גבוה מבין הרצפים. עתה יש לבדוק האם כדאי בכלל לזרוק רשימה זו או שכדאי לזרוק בכלל את הקלף בעל הערך הגדול ביותר (לדוגמא יכול להיות

הרצף 2,2,2 בעוד יש לשחקן קלפים אשר גדולים מ 6. במצב זה יעדיף האלגוריתם להיפטר מהסכום הגדול יותר ולא מהרצף – יעילות בדיקה זו היא O(n) .) סדר הגודל של יעילות פעולה זו הוא O(n^2).

ביבליוגרפיה:

- אלגוריתם סיבוב תמונה – השתמשתי באלגוריתם סיבוב תמונה על מנת לאפשר את ציור הקלפים על הלוח באופן מסובב (90 מעלות) .
https://code.google.com/archive/p/game-engine-for-java/source

- תמונות הקליפים נלקחו מאתר :
- http://acbl.mybigcommerce.com/52-playing-cards/

- קול צעקת יניב – הוקלט ע"י חבר לכיתה (נאור חמישה)

- האתר stackoverflow , שימוש לצורך מציאת פתרונות הקיימים בשפה (דוגמה לכך היא שימוש במטודה shuffle אשר אחראית על ערבוב הקלפים בקופה) ובנוסף הבנה כיצד לממש את פעולות אלו בצורה מיטבית בקוד.

- stackoverflow.com

קוד מקור :

מחלקה main:

```
 package game;

import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import graphic.OpenScreen;
import yaniv.Game;

public class main {
public static JFrame frame;
public static Game game;
public static void main(String[] args) {

frame = new JFrame("open screen");
game = new Game();
OpenScreen panel = new OpenScreen();
//LoseScreen panel = new LoseScreen("fgdfg",45);
frame.add(panel);//chnow
//frame.add(maxScores);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400,400);
//frame.setSize(1280,720);
frame.setVisible(true);//
frame.setLocation(400,250);
frame.setResizable(false);
try {
frame.setIconImage(ImageIO.read(new File("pictures/yanivImage.jpg")));
} catch (IOException e) {
e.printStackTrace();
}

}

{
```

מחלקה card

```
package yaniv;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;

public class Card}
private int number; // 0 for joker 11 for Jack 12 for queen 13 for king
private char shape;
private char color;
private Image image;// image of the card
private int serial;// between 1-54 needed a chart that explain the serials numbers.
//all the mixed cards will be in cards list and the identification depend on serial

 //wanted to have the image from the homePile according to serial
public Card(int number, char shape, char color, int serial, Image image)}
this.number = number;
this.shape = shape;
this.color = color;
this.serial = serial;
this.image = image;// uses for draw the cards
{


public int getSerial} ()
return serial;
{
public void setSerial(int serial)}
this.serial = serial;
{
public int getNumber} ()
return number;
{
public void setNumber(int number)}
this.number = number;
{
public char getShape} ()
return shape;
{
public void setShape(char shape)}
this.shape = shape;
{
public char getColor} ()
return color;
{
public void setColor(char color)}
this.color = color;
{

public Image getImage} ()
```

```
return image;
{

public void setImage(Image image)}
this.image = image;
{
**/
 *equals method, uses mainly for removeAll in Player class
/*
@Override
public boolean equals(Object obj)}
if (obj instanceof Card)}
Card temp = (Card)obj;
if (temp.serial == serial)}
return true;
{
{
return false;
{
**/
 *Converts a given BufferedImage into an Image
*
@ *param bimage The BufferedImage to be converted
@ *return The converted Image
/*
public static Image toImage(BufferedImage bimage)}
 //Casting is enough to convert from BufferedImage to Image
Image img = (Image) bimage;
return img;
{
**/
 *Creates an empty image with transparency
*
@ *param width The width of required image
@ *param height The height of required image
@ *return The created image
/*
public static Image getEmptyImage(int width, int height)}
BufferedImage img = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
return toImage(img);
{
**/
 *Converts a given Image into a BufferedImage
*
@ *param img The Image to be converted
@ *return The converted BufferedImage
/*
public static BufferedImage toBufferedImage(Image img)}
if (img instanceof BufferedImage)}
return (BufferedImage) img;
{
 //Create a buffered image with transparency
```

```
BufferedImage bimage = new BufferedImage(img.getWidth(null), img.getHeight(null),
BufferedImage.TYPE_INT_ARGB);
 //Draw the image on to the buffered image
Graphics2D bGr = bimage.createGraphics;()
bGr.drawImage(img, 0, 0, null);
bGr.dispose;()
 //Return the buffered image
return bimage;
{
**/
 *Rotates an image. Actually rotates a new copy of the image.
*
@ *param img The image to be rotated
@ *param angle The angle in degrees
@ *return The rotated image
/*
public Image rotate(Image img, double angle)}
double sin = Math.abs(Math.sin(Math.toRadians(angle))), cos =
Math.abs(Math.cos(Math.toRadians(angle)));
int w = img.getWidth(null), h = img.getHeight(null);
int neww = (int) Math.floor(w * cos + h * sin), newh = (int) Math.floor(h
 *cos + w * sin;(
BufferedImage bimg = toBufferedImage(getEmptyImage(neww, newh));
Graphics2D g = bimg.createGraphics;()
g.translate((neww - w) / 2, (newh - h) / 2);
g.rotate(Math.toRadians(angle), w / 2, h / 2);
g.drawRenderedImage(toBufferedImage(img), null);
g.dispose;()
return toImage(bimg);
{
//overload - use for draw player card
public void draw(Graphics g, int x, int y)}
g.drawImage(this.image, x, y, 70, 120, null);
{

//overload - use for draw player card if isAvalible is true - draw the card regular (see details) else draw his
back.
public void draw(Graphics g, int x, int y, Image backCard, Boolean isAvalible)}
if (isAvalible)}
Image temp = rotate(image, 90);
g.drawImage(temp, x, y, 120, 70, null);
{else}
backCard = rotate(backCard, 90);
g.drawImage(backCard, x, y, 120, 70, null);
{

{

{
```

מחלקה Comparator

```
package yaniv;

public class Comparator implements java.util.Comparator<Player> {

	//compare between players actually between their sum cards in hand
	@Override
	public int compare(Player o1, Player o2) {
		if (o1.sumCards() < o2.sumCards())
			return -1;
		if(o1.sumCards() == o2.sumCards())
			return 0;
		return 1;
	}
}
```

מחלקה Comparator cards

```
package yaniv;

import java.util.Comparator;

public class ComparatorCards implements Comparator<Card> {
//compare between cards value
	@Override
	public int compare(Card o1, Card o2) {
		if (o1.getNumber() < o2.getNumber())
			return -1;
		if(o1.getNumber() == o2.getNumber())
			return 0;
		return 1;

	}

}
```

מחלקה CompareSequence

```
package yaniv;

import java.util.Comparator;
import java.util.List;

public class CompareSequence implements Comparator<List<Card>>{
//Compare between the summary of the lists
@Override
public int compare(List<Card> o1, List<Card> o2) {
return sumList(o1) - sumList(o2);

}
private int sumList(List<Card> list) {
int sum = 0;
for(int i = 0; i < list.size(); i++) {
sum += list.get(i).getNumber();
}
return sum;
```

```
}

}
```

מחלקה game

```
package yaniv;


//import java.awt.Point;
import java.awt.Rectangle;
import java.io.File;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Collections;
//import java.io.IOException;
//import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.swing.JPanel;
import game.main;
import graphic.GamePanel;
import graphic.LoseScreen;
import graphic.WinningScreen;

public class Game  {
private static final int speed_Of_card_Take_animatoin = 1;//Recommend between 1 to 5
private static final int Cards_In_Hand = 5;
private static final int division = 30;
private static final int lenghtOfCard = 120;
private static final int final_Value_fix_Rectangle_Location = 35;
private static final int Number_Of_Players = 3;
private static final int amount_TO_Say_YANIV = 8;
final int CardsInHand = 5;
private List<Player> players;
private OpenPile openPile;
private Pile pile;
private List<Rectangle> cardsLocationAsRectangle;
private Rectangle pileLocationAsRectangle;
private Rectangle openPileLocationAsRectangle;
private int currentPlayer = 1;// every change will change + 3 and after method %3
GamePanel gamePanel;//change from jpanel
private List<Card> cardsSequences = new LinkedList<Card>();
private static List<Integer> points = new ArrayList<Integer>(); ;
```

```
private static Boolean isRun = false;
private int currentWinnerID = -1;
private String PlayerName;
private int gameCounter = 1;//count how many games the player play before he exit the game use for
calculate his points for max points
private boolean isInDestionation;
public Game() {
this.players = new LinkedList<Player>();
this.openPile = new OpenPile();
this.cardsLocationAsRectangle = new LinkedList<Rectangle>();
this.pile = new Pile();
createPlayers(Number_Of_Players);// create the players and put them in the players array
scatterCards();//scatter the cards
openPile.addCard(pile.removeCard());// after scatter the cards pull one card from pile and put in open pile
like opening card
createRectangleLocations();
createPileRectangleLocations();
createOpenPileRectangleLocations();
if (!isRun) {
for(int i = 0; i < this.players.size(); i ++)
points.add(0);
isRun = true;
}


}


public String getPlayerName() {
return PlayerName;
}

public void setPlayerName(String playerName) {
PlayerName = playerName;
}

public int getCurrentPlayer() {
return currentPlayer;
}

public void setCurrentPlayer(int currentPlayer) {
this.currentPlayer = currentPlayer;
}

public List<Player> getPlayers() {
return players;
}

public void setPlayers(List<Player> players) {
this.players = players;
}

public OpenPile getOpenPile() {
```

```
return openPile;
}

public void setOpenPile(OpenPile openPile) {
this.openPile = openPile;
}

public Pile getPile() {
return pile;
}

public void setPile(Pile pile) {
this.pile = pile;
}

public int getGameCounter() {
return gameCounter;
}

public void setGameCounter(int gameCounter) {
this.gameCounter = gameCounter;
}

/**
* when someone wins that method called. It check who is the winner and play the yaniv sound of a winning
*/
public void yanivButton() {
Player winner = Collections.min(players,new Comparator());
// check if the count works well. the player with the lowest sum is the winner
this.currentWinnerID = winner.getID();
try{
File  url = new File("sounds/yaniv.wav");
Clip clip = AudioSystem.getClip();
AudioInputStream ais = AudioSystem.getAudioInputStream(url);
clip.open(ais);
clip.start();
}catch(LineUnavailableException | UnsupportedAudioFileException | IOException murle){
System.out.println(murle);
System.out.println("exeption");
}
gameFinish();
return;

}

/**
* get number of players and create players (according to numberOfPlayers)
* @param numberOfPlayers
*/
private void createPlayers(int numberOfPlayers){
for(int i = 0; i < numberOfPlayers; i ++) {
players.add(new Player(i));
}
```

```
players.get(0).setName("computer 0");
players.get(1).setName("you");
players.get(2).setName("computer 2");
}

/**
* scatter the cards to the players in players array. give every player 5 cards
*/
public void scatterCards(){
for(int i = 0; i < this.players.size() ; i++){
for(int j = 0; j < Cards_In_Hand; j++)
players.get(i).addCard(pile.removeCard());
}
}



/**
* Responsible to create the cardsLocationAsRectangle for player 1 (the user),
*  uses for recognise which card has pressed
*/

public void createRectangleLocations() {
/**
* run according to the cards size and create the rectangle location list
*/
List<Rectangle> cardsLocationAsRectangleTemp = new LinkedList<Rectangle>();
int x = players.get(1).getX();
for(int i = 0; i < players.get(1).getCards().size() - 1; i++, x += 45){
Rectangle location = new Rectangle(x, players.get(1).getY() - final_Value_fix_Rectangle_Location, 45,
120);
cardsLocationAsRectangleTemp.add(location);
}
Rectangle location = new Rectangle(x, players.get(1).getY() - final_Value_fix_Rectangle_Location, 70,
120);
cardsLocationAsRectangleTemp.add(location);
this.cardsLocationAsRectangle = cardsLocationAsRectangleTemp;
}

/**
* Responsible to create the PileLocationAsRectangle for the pile
* use to recognise if pile pressed
*/
public void createPileRectangleLocations() {
// 11 is the offset from the first card that draw to the last one. 70 is the    and 120 is the
Rectangle locationPile = new Rectangle(Pile.getxStart(), Pile.getyStart(), 70 + 11, 120 + 11);
this.pileLocationAsRectangle = locationPile;
}

/**
* Responsible to create the OpenPileLocationAsRectangle for the pile
* use to recognize if pile pressed
*/
```

```
public void createOpenPileRectangleLocations() {
Rectangle locationPile = new Rectangle(openPile.getxStart(), openPile.getyStart(), 70 + 11, 120);
this.openPileLocationAsRectangle = locationPile;
}

/**
* a part of the AI. this method is responsible on the turns of the computer
* the method computerPlays is the AI itself
*/
private void computersPlay() {
Thread thread = new Thread(new Runnable() {
@Override
public void run() {
for (int i = 0; i < 2; i++){
if (currentPlayer == 0 || currentPlayer == 2){
try {
Thread.sleep(2000);
} catch (InterruptedException e) {
e.printStackTrace();
}
if(players.get(0).getGameOver()) {
continue;//change
}
computerPlays();
gamePanel.repaint();
currentPlayer++;
currentPlayer = currentPlayer % 3;
}
}
}
});
thread.start();
}

/**
* responsible to clear all the lists that involved in graphic
*/
public void removeCardsFromHand(){
for(int i = 0; i < players.get(1).getTempCardSerial().size(); i++){
openPile.addCard(players.get(1).removeCard(players.get(1).getTempCardSerial().get(i)));
createRectangleLocations();
}
/**
* start sound of pull a card
*/
try{
File  url = new File("sounds/draw.wav");
Clip clip = AudioSystem.getClip();
AudioInputStream ais = AudioSystem.getAudioInputStream(url);
clip.open(ais);
clip.start();
}catch(LineUnavailableException | UnsupportedAudioFileException | IOException murle){
System.out.println(murle);
```

```
}

players.get(1).getTempCardSerial().clear();
players.get(1).getTempCardIndex().clear();
players.get(1).getTempCards().clear();
players.get(1).getTempSequenceCards().clear();

}

public void cardClicked3(int x, int y) {

int cardClickedNumber = -1;
for(int i = 0; i < cardsLocationAsRectangle.size() && cardClickedNumber < 0; i++){
if(x > cardsLocationAsRectangle.get(i).getX() &&  x < (cardsLocationAsRectangle.get(i).getX() +
cardsLocationAsRectangle.get(i).getWidth())
&& y > cardsLocationAsRectangle.get(i).getY() + division && y < cardsLocationAsRectangle.get(i).getY() +
lenghtOfCard + division ){
if(players.get(1).isIncludeIntempCardIndex(i)) {
players.get(1).chooseDOWN(players.get(1).getCards().get(i), 1,i);
cardClickedNumber = players.get(1).getCards().get(i).getSerial();
}else {
players.get(1).chooseUP(players.get(1).getCards().get(i), 1,i);
}
}
}
}

/**
* call the correct method according to y
* @param x
* @param y
* @param p
*/
public void whatClicked(int x, int y, JPanel p) {
players.get(1).findAllGrowingSequenceNew();
if(!players.get(0).getGameOver()) {
gamePanel = (GamePanel) p;
if(currentPlayer == 1){
if (y > 400){
cardClicked3(x, y);
}else if (y >= 250 && !players.get(1).getTempCardSerial().isEmpty()){
openPileClickedNew(x, y);
}else if(!players.get(1).getTempCardSerial().isEmpty()){
pileClickedNew(x, y);
}
}
gamePanel.repaint();
}


}

private void pileFinished() {
```

```
Card tempCard;
tempCard = this.openPile.getOpenPile().pop();
while(!this.openPile.getOpenPile().isEmpty()){
this.pile.addCard(this.openPile.removeCard());
}
this.openPile.getOpenPile().push(tempCard);
this.pile.shuffelPile();
}

/**
* clear all the lists that saved the selected cards, means that all the cards turn down.
* @param player
*/
private void turnDownTheCards(Player player) {
players.get(1).getTempCardSerial().clear();
players.get(1).getTempCardIndex().clear();
players.get(1).getTempCards().clear();
players.get(1).getTempSequenceCards().clear();
}
public void computerPlays(){
/**
* first check if can call yaniv. If does - call yanivButton and end the round
* if not -
*/
if(this.players.get(currentPlayer).sumCards() <= amount_TO_Say_YANIV) {
yanivButton();
return; // do not keep the function After call to yanivButton.
}
this.cardsSequences = players.get(currentPlayer).getLongestSequence();
/**
* if the card in the open pile appear in the cards in hand - take it
* need to check if the there is a sequence with the same number - if in hand there is 2,4,4,5,5
* the computer will prefer to throw the 5,5 but if he take the card from the open he will have
* 5,5,5 which is better.will call a special method that create the cardSequence again but that
* time the sequence to throw will be the 4,4.
*/
if(players.get(currentPlayer).isCompleteSequence(openPile.getOpenPile().peek())) {
this.cardsSequences =
players.get(currentPlayer).takeTheSameCardInOpenPile(openPile.getOpenPile().peek());
moveCardFromOpenPile();
Card cardToAdd = openPile.getOpenPile().pop();
for (int i = 0; i < this.cardsSequences.size(); i++) {
openPile.addCard(this.cardsSequences.get(i));
players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
}
players.get(currentPlayer).addCard(cardToAdd);
} else if (openPile.getOpenPile().peek().getNumber() <= 5 && openPile.getOpenPile().peek().getNumber() <
players.get(currentPlayer).getCards().get(0).getNumber() ||
players.get(currentPlayer).findGradestCard().getNumber() >=  openPile.getOpenPile().peek().getNumber()
&& openPile.getOpenPile().peek().getNumber() <= 5){
moveCardFromOpenPile();
Card cardToAdd = openPile.getOpenPile().pop();
for (int i = 0; i < this.cardsSequences.size(); i++) {
```

```
openPile.addCard(this.cardsSequences.get(i));
players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
}
players.get(currentPlayer).addCard(cardToAdd);
}else{
for (int i = 0; i < this.cardsSequences.size(); i++) {
openPile.addCard(this.cardsSequences.get(i));
players.get(currentPlayer).removeCard(this.cardsSequences.get(i).getSerial());
}
moveCardFromStock();
players.get(currentPlayer).addCard(pile.removeCard());
}


// the sound will be after all the checks
try{
File  url = new File("sounds/draw.wav");
Clip clip = AudioSystem.getClip();
AudioInputStream ais = AudioSystem.getAudioInputStream(url);
clip.open(ais);
clip.start();
}catch(LineUnavailableException | UnsupportedAudioFileException | IOException murle){
System.out.println(murle);
}
if(this.pile.getPileCards().isEmpty()){
pileFinished();// fill the pile again
}
}

// methods for computer:
/**
* afer yaniv button called - come here, update the points of each player for the current play
* show the cards of the players for 4 seconds and move the player to the correct screen.
*/
public void gameFinish() {
for(int i = 0; i < players.size(); i++) {
players.get(i).setGameOver(true);
}
updatePoints();
gamePanel.repaint();
//if someone wanted to throw a card, and leave it up but decided to call yaniv
for (int i = 0; i < players.size(); i++) {
turnDownTheCards(players.get(i));
}
Timer timer = new Timer();
timer.schedule(new TimerTask() {

@Override
public void run() {
if (currentWinnerID != 1) {
main.frame.setSize(1280,720);
main.frame.setContentPane(new
LoseScreen(players.get(currentWinnerID).getName(),points.get(currentWinnerID)));
```

```
main.frame.validate();
main.frame.repaint();
PlayerName = "computer";
} else{
main.frame.setSize(1280,720);
main.frame.setContentPane(new
WinningScreen(players.get(currentWinnerID).getName(),points.get(currentWinnerID)));
main.frame.validate();
main.frame.repaint();
}

}
}, 4000);

//update the new points
}

/**
* update the points. the winner does not get points.
*/
public void updatePoints() {
for(int i = 0; i < points.size(); i++) {
Integer temp = points.get(i);
temp += this.players.get(i).getPointsInHand();
points.set(i, temp);// set the new points
GamePanel.updatePoints(points);
}
}

public void newGame() {
for(int i = 0; i < this.players.size(); i++) {
while (!this.players.get(i).getCards().isEmpty()) {
this.players.get(i).removeCard(this.players.get(i).getCards().get(0).getSerial());
}
}
for( int i = 0; i < players.size(); i++) {
players.get(i).setGameOver(false);
}
this.currentWinnerID = -1;
this.openPile = new OpenPile();
this.pile = new Pile();
scatterCards();//scatter the cards
openPile.addCard(pile.removeCard());// after scatter the cards pull one card from pile and put in open pile
like opening card
createRectangleLocations();
createPileRectangleLocations();
createOpenPileRectangleLocations();
this.currentPlayer = 1;
this.gameCounter ++;
refreshCurrentRectanglePlayer();

}
private void moveCardFromStock() {
```

```
gamePanel.setSeeDetails(false);
gamePanel.setDestinationAndSource(Pile.getxStart(), Pile.getyStart(),
players.get(currentPlayer).getDestenationX(), players.get(currentPlayer).getDestenationY(), pile.getTop());
isInDestionation = gamePanel.isNotInDestionation();
while (isInDestionation) {
isInDestionation = gamePanel.isNotInDestionation();
gamePanel.repaint();
try {
Thread.sleep(speed_Of_card_Take_animatoin);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
gamePanel.setCurrrentCard();
}
private void moveCardFromOpenPile() {
gamePanel.setSeeDetails(true);
gamePanel.setDestinationAndSource(openPile.getxStart(), openPile.getyStart(),
players.get(currentPlayer).getDestenationX(), players.get(currentPlayer).getDestenationY(),
openPile.getOpenPile().peek());
isInDestionation = gamePanel.isNotInDestionation();
while (isInDestionation) {
isInDestionation = gamePanel.isNotInDestionation();
gamePanel.repaint();
try {
Thread.sleep(speed_Of_card_Take_animatoin);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
gamePanel.setCurrrentCard();
}

private void refreshCurrentRectanglePlayer() {
for (int i = 0; i < players.size(); i++) {
Rectangle tempPlayerRectangle = players.get(i).getCurrentPlayerRectangle();
if (i == 0) {;
tempPlayerRectangle.setBounds(tempPlayerRectangle.x, tempPlayerRectangle.y,
tempPlayerRectangle.width, 45 * players.get(i).getCards().size() + 40);
} else if (i == 1){
tempPlayerRectangle.setBounds(tempPlayerRectangle.x, tempPlayerRectangle.y, 45 *
players.get(i).getCards().size() + 35, tempPlayerRectangle.height);
} else { // i == 2
tempPlayerRectangle.setBounds(tempPlayerRectangle.x, 340 - 45 * players.get(i).getCards().size(),
tempPlayerRectangle.width, 45 * players.get(i).getCards().size() + 40);
}
players.get(i).setCurrentPlayerRectangle(tempPlayerRectangle);
}
}

private void removeCardsFromHandToPile() {
removeCardsFromHand();
gamePanel.repaint();
```

```
refreshCurrentRectanglePlayer();
currentPlayer ++;
currentPlayer = currentPlayer % 3;
computersPlay();

}
public void openPileClickedNew(int x, int y) {
if(x > openPileLocationAsRectangle.getX() &&  x < (openPileLocationAsRectangle.getX() +
openPileLocationAsRectangle.width )
&& y > openPileLocationAsRectangle.getY() && y < (openPileLocationAsRectangle.getY() +
openPileLocationAsRectangle.height )){
createRectangleLocations();
if(players.get(1).isCanBeThrowen(players.get(1).getTempSequenceCards())){
moveCardFromOpenPile();
this.players.get(1).addCard(this.openPile.removeCard());
removeCardsFromHandToPile();
}else{
turnDownTheCards(this.players.get(1));
}
}
}

/**
* remove the card from the pile and add it to the player
* @param x of click
* @param y of click
*/
public void pileClickedNew(int x , int y) {
if(x > pileLocationAsRectangle.getX() &&  x < (pileLocationAsRectangle.getX() +
pileLocationAsRectangle.width)
&& y > pileLocationAsRectangle.getY() && y < (pileLocationAsRectangle.getY() +
pileLocationAsRectangle.height)){
if(this.pile.getPileCards().isEmpty()){
pileFinished();// fill the pile again
}
createRectangleLocations();
if(players.get(1).isCanBeThrowen(players.get(1).getTempSequenceCards())){
moveCardFromStock();// first draw the animation and then remove from pile. if first remove card
//from pile and only then draw the animation, the card is going to be drawn is not the correct one
this.players.get(1).addCard(this.pile.removeCard());
removeCardsFromHandToPile();

}else{
turnDownTheCards(this.players.get(1));
}
if(this.pile.getPileCards().isEmpty()){
pileFinished();// fill the pile again
}

}
}
}
```

מחלקה HomePile

```
package yaniv;

import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

import javax.imageio.ImageIO;

/**
* this class used to save all the cards in a list. It responsible on the first boot of the cards.
* @author Tomer
*
*/
public class HomePile {
private List<Card> homeCards;

public HomePile() {
this.homeCards = new LinkedList<Card>();
createHomePile();
}
/**
* the method will create the home pile. will adds 54 cards to the list.This list changes never!
* every card in the list has his image so it can be draw.Every card in the list get a shape, color and picture
*/
private void createHomePile(){
char shape = 's';
char color = 'b' ;
int number = 1;
for(int i = 1; i < 53; i++){
try{
Image icon = ImageIO.read(new File("pictures/"+i+".png"));
Card c = new Card(number, shape, color, i, icon);
homeCards.add(c);
}catch (IOException e){
System.out.println("file not found");
}
if(shape == 's'){
shape = 'h';
color = 'r';
}
else if(shape == 'h'){
shape = 'd';
color = 'r';
}
else if (shape == 'd'){
shape = 'c';
color = 'b';
}else{
shape = 's';
```

```
color = 'b';
}
/**
* every 4 cards the number of the card changes
*/


if(i % 4 == 0)
number ++;
//Card c = new Card(number, shape, color, i,icon);
}
/**
* create the jokers- special cards, and add them to the list
*/
try{
Image icon = ImageIO.read(new File("pictures/"+53+".png"));
Joker j = new Joker('r', 53, icon);
homeCards.add(j);
icon  = ImageIO.read(new File("pictures/"+54+".png"));
j = new Joker('b', 54, icon);
homeCards.add(j);
}catch(IOException e){
System.out.println("file not found");
}

}
/**
* only get method, avoid changing this list by mistake
* @return the homeCards list
*/
public List<Card> getHomeCards() {
return homeCards;
}


}
```

מחלקה Joker

```
package yaniv;


import java.awt.Image;
/**
 * class for joker card- special card. in addiction to a regular card it has two more
attributes this is because joker can be every card
 * @author Tomer
 *
 */
public class Joker extends Card {
	private int possible1;
	private int possible2;
	public Joker(char color, int serial, Image image) {
		super(0,'j', color, serial, image);// j for joker
		possible1 = -1;
		possible2 = -1;
```

```
        }
        public int getPossible1() {
                return possible1;
        }
        public void setPossible1(int possible1) {
                this.possible1 = possible1;
        }
        public int getPossible2() {
                return possible2;
        }
        public void setPossible2(int possible2) {
                this.possible2 = possible2;
        }


}
```

מחלקה OpenPile

```
package yaniv;

import java.awt.Graphics;
import java.util.Stack;

import graphic.Drawable;

public class OpenPile implements Drawable}
Stack<Card> openPile;
**/
 *xStart - the x to start draw the open pile
 *yStart - the y to start draw the open pile
/*
private static final int xStart = 470;
private static final int yStart = 250;

public OpenPile(Stack<Card> openPile)}
this.openPile = openPile;
{



public Stack<Card> getOpenPile}()
return openPile;
{

public void setOpenPile(Stack<Card> openPile)}
this.openPile = openPile;
{

public OpenPile}()
this.openPile = new Stack<Card;()<
{
**/
 *add a card to the stack
@ *param card
/*
public void addCard(Card card)}
this.openPile.push(card);
```

```
{
**/
*
@ *return the top of the stack and remove it from the stack
/*
public Card removeCard} ()
return this.openPile.pop;()
{


public int getxStart} ()
return xStart;
{

public int getyStart} ()
return yStart;
{

**/
 *use for the drawable interface
/*
@Override
public void draw(Graphics g)}
g.drawImage(openPile.peek().getImage(), OpenPile.xStart, OpenPile.yStart, 70, 120, null);
{







{
```

מחלקה Pile

```
package yaniv;

import java.awt.Graphics;
import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import javax.imageio.ImageIO;

import graphic.Drawable;

public class Pile implements Drawable}
**/
 *xStart - the x to start draw the open pile
 *yStart - the y to start draw the open pile
 *icon - the picture to draw
```

```
 *pile cards - the pile itself
 *homePile - the packet of the cards
/*
private static final int X_start = 470;
private static final int Y_start = 100;
private static  Image icon;
private Queue<Card> pileCards;
private HomePile homePile;

**/
 *build he pile, shuffle the cards
/*
public Pile} ()
this.pileCards = new LinkedList<Card;()<
homePile = new HomePile;()
firstShuffelPile;()
createImageOfPile;()
{

public static int getxStart} ()
return X_start;
{

public static int getyStart} ()
return Y_start;
{

public void addCard(Card card)}
this.pileCards.add(card);
{
public Queue<Card> getPileCards} ()
return pileCards;
{

public Card getTop} ()
return this.pileCards.peek;()
{
**/
*
@ *return the top of the queue and remove it from the queue
/*
public Card removeCard}()
return this.pileCards.remove;()
{
**/
 *does not destroy the list, copy to another list and shuffle the copied list, then copy the
shuffled list to the pileCards
 *now the pileCards include a queue of shuffled cards
 *Shuffle all the cards and put them in the pileCards
@ *param pileCards include the  cards
/*
public void firstShuffelPile}()
List<Card> tempCards = new LinkedList<Card;()<
for(int i = 0; i < this.homePile.getHomeCards().size(); i++)}
tempCards.add(this.homePile.getHomeCards().get(i));
{
Collections.shuffle(tempCards);
```

```
for(int i = 0; i < tempCards.size(); i++)}
this.pileCards.add(tempCards.get(i));
{

{

**/
 *the same as firstShuffelPile but is shuffle the pile itself and not a copy of the pile
/*
public void shuffelPile} ()
List<Card> tempCards = new LinkedList<Card;()<
for(int i = 0; i < this.pileCards.size(); i++)}
tempCards.add(this.pileCards.remove());
{
Collections.shuffle(tempCards);
for(int i = 0; i < tempCards.size(); i++)}
this.pileCards.add(tempCards.get(i));
{
{

**/
 *load the icon to draw to the attribute
/*
private void createImageOfPile} ()
try}
Pile.icon = ImageIO.read(new File("pictures/gray_back.png"));
{catch(IOException e}(
System.out.println("file not found");
{
{

**/
 *use for the drawable interface
/*
@Override
public void draw(Graphics g)}

int i;
for(i = 1; i < 10; i += 2)}
g.drawImage(Pile.icon, X_start + i , Y_start + i , 70, 120, null);
{


{
{
```

```
package yaniv;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import javax.imageio.ImageIO;
import graphic.Drawable;

public class Player implements Drawable}
private List<Card> cards;
private Boolean isYaniv;
private Rectangle currentPlayerRectangle;
private int ID;
private List<Integer> tempCardIndex; // list of card that chosen - will draw differently
private List<Card> tempCards;
private List<Integer> tempCardSerial;// list of the serial of the cards that chosen
//private Card tempCard;
private String name;
private int x = 120,y;//place of the draw cards
private Boolean gameOver = false;
private int destenationX;
private int destenationY;
private List<Card> tempSequenceCards;

**/
 *create the player. give him 5 mixed cards and set the YANIV call to false.
/*
public Player(int id)}
this.ID = id;
this.cards = new LinkedList<Card;()<
this.isYaniv = false;
tempSequenceCards = new ArrayList<Card;()<
tempCardIndex = new ArrayList<Integer;()<
tempCardSerial = new ArrayList<Integer;()<
tempCards = new ArrayList<Card;()<
currentPlayerRectangle = new Rectangle;()
if(id == 0)}
x = 80;
y = 120;
destenationX = 80;
destenationY = 210;
```

```
currentPlayerRectangle.setBounds(75, 110, 130, 270);// static rectangle,
//appear when the current player playing
{
if(id == 1)}
x = 385;
y =400;
destenationX = this.x + (45 * 2);//to the middle of the cards
destenationY = 400;
currentPlayerRectangle.setBounds;(130 ,260 ,395 ,380)
{

if(id == 2)}
y = 300;
x = 800;
destenationX = 800;
destenationY = 210;
currentPlayerRectangle.setBounds;(270 ,130 ,110 ,795)
{


{


public List<Card> getTempSequenceCards} ()
return tempSequenceCards;
{


public void setTempSequenceCards(List<Card> tempSequenceCards)}
this.tempSequenceCards = tempSequenceCards;
{


public int getDestenationX} ()
return destenationX;
{


public void setDestenationX(int destenationX)}
this.destenationX = destenationX;
{


public int getDestenationY} ()
return destenationY;
{


public void setDestenationY(int destenationY)}
this.destenationY = destenationY;
{
```

```
public Boolean getGameOver} ()
return gameOver;
{

public void setGameOver(Boolean gameOver)}
this.gameOver = gameOver;
{


public Rectangle getCurrentPlayerRectangle} ()
return currentPlayerRectangle;
{

public void setCurrentPlayerRectangle(Rectangle currentPlayerRectangle)}
this.currentPlayerRectangle = currentPlayerRectangle;
{

public List<Card> getTempCards} ()
return tempCards;
{

public void setTempCards(List<Card> tempCards)}
this.tempCards = tempCards;
{

public List<Integer> getTempCardSerial} ()
return tempCardSerial;
{

public void setTempCardSerial(List<Integer> tempCardSerial)}
this.tempCardSerial = tempCardSerial;
{

public List<Integer> getTempCardIndex} ()
return tempCardIndex;
{

public void setTempCardIndex(List<Integer> tempCardIndex)}
this.tempCardIndex = tempCardIndex;
{

public List<Card> getCards} ()
return cards;
{

public void setCards(List<Card> cards)}
this.cards = cards;
{

public Boolean getIsYaniv} ()
return isYaniv;
{
```

```
public void setIsYaniv(Boolean isYaniv)}
this.isYaniv = isYaniv;
{

public int getID} ()
return ID;
{

public void setID(int iD)}
ID = iD;
{

public int getX} ()
return x;
{

public void setX(int x)}
this.x = x;
{

public int getY} ()
return y;
{

public void setY(int y)}
this.y = y;
{

public String getName} ()
return name;
{

public void setName(String name)}
this.name = name;
{



**/
 *an event will be on that property. it will listen when it changes to true,
 *then it will check other players cards and announce who is the winner
/*
public void YanivCall} ()
this.isYaniv = true;
{

**/
*
@ *return the sum of the cards of the player
/*
public int sumCards} ()
int sum = 0;
for(int i = 0; i < cards.size(); i++)}
```

```
if(this.cards.get(i).getNumber() > 10)}
sum += 10;
{else}
sum += this.cards.get(i).getNumber;()
{
{
return sum;
{

public void addCard(Card card)}
this.cards.add(card);
{
**/
 *remove a card from the card list according to the serial of the card
 *if serial not found do nothing
@ *param serialCardNumber
/*
public Card removeCard(int serialCardNumber)}
for(int i = 0; i < this.cards.size(); i++)}
if(this.cards.get(i).getSerial() == serialCardNumber)
return this.cards.remove(i);
{
return null;
{




**/
 *Converts a given BufferedImage into an Image
*
@ *param bimage The BufferedImage to be converted
@ *return The converted Image
/*
public static Image toImage(BufferedImage bimage)}
 //Casting is enough to convert from BufferedImage to Image
Image img = (Image) bimage;
return img;
{
**/
 *Creates an empty image with transparency
*
@ *param width The width of required image
@ *param height The height of required image
@ *return The created image
/*
public static Image getEmptyImage(int width, int height)}
BufferedImage img = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
return toImage(img);
{
**/
 *Converts a given Image into a BufferedImage
```

```
*
@ *param img The Image to be converted
@ *return The converted BufferedImage
/*
public static BufferedImage toBufferedImage(Image img)}
if (img instanceof BufferedImage)}
return (BufferedImage) img;
{
 //Create a buffered image with transparency
BufferedImage bimage = new BufferedImage(img.getWidth(null), img.getHeight(null),
BufferedImage.TYPE_INT_ARGB);
 //Draw the image on to the buffered image
Graphics2D bGr = bimage.createGraphics;()
bGr.drawImage(img, 0, 0, null);
bGr.dispose;()
 //Return the buffered image
return bimage;
{
**/
 *Rotates an image. Actually rotates a new copy of the image.
*
@ *param img The image to be rotated
@ *param angle The angle in degrees
@ *return The rotated image
/*
public Image rotate(Image img, double angle)}
double sin = Math.abs(Math.sin(Math.toRadians(angle))), cos =
Math.abs(Math.cos(Math.toRadians(angle)));
int w = img.getWidth(null), h = img.getHeight(null);
int neww = (int) Math.floor(w * cos + h * sin), newh = (int) Math.floor(h
 *cos + w * sin;(
BufferedImage bimg = toBufferedImage(getEmptyImage(neww, newh));
Graphics2D g = bimg.createGraphics;()
g.translate((neww - w) / 2, (newh - h) / 2);
g.rotate(Math.toRadians(angle), w / 2, h / 2);
g.drawRenderedImage(toBufferedImage(img), null);
g.dispose;()
return toImage(bimg);
{

private int findIndexInSerialCardIndex(int serialNumberCard)}
int indexOfCardsInList;
for(indexOfCardsInList = 0; indexOfCardsInList < tempCardSerial.size(); indexOfCardsInList++)}
if(tempCardSerial.get(indexOfCardsInList) == serialNumberCard)
return indexOfCardsInList;
{
return -1;
{
private int findIndexIntempCardIndex(int cardHandIndex)}
int indexOfCardsInList;
for(indexOfCardsInList = 0; indexOfCardsInList < tempCardIndex.size(); indexOfCardsInList++)}
if(tempCardIndex.get(indexOfCardsInList) == cardHandIndex)
return indexOfCardsInList;
```

```
{
return -1;
{

public int findIndexInCardsList(Card card, List<Card> list)}
int indexOfCardsInList;
for(indexOfCardsInList = 0; indexOfCardsInList < list.size(); indexOfCardsInList++)}
if(list.get(indexOfCardsInList).getSerial() == card.getSerial())
return indexOfCardsInList;
{
return -1;
{
**/
 *check if the index that given appear in the list if it does return true
@ *param index
@ *return
/*
public boolean isIncludeIntempCardIndex(int index)}
for(int i = 0; i < tempCardIndex.size(); i++)}
if(tempCardIndex.get(i) == index)
return true;
{
return false;
{
**/
 *check if the card that send is appear in hand
@ *param card
@ *return true is does or false if not
/*
public boolean isIncludeInHandCards(Card card)}
for(int i = 0; i < this.cards.size(); i++)}
if(cards.get(i).getNumber() == card.getNumber())
return true;
{
return false;
{


**/
 *draw all the card in cards list
/*
@Override
public void draw(Graphics g)}
int xStart = this.x;
int yStart = this.y;
Image temp, backCard = null;
try}
backCard = ImageIO.read(new File("pictures/gray_back.png"));
 {catch (IOException e} (
e.printStackTrace;()
{
**/
 *if != 1 so the players at the sides of the screen, else draw the cards of the player
```

```
/*
SortHand();//sort cards before drawing
if (this.ID == 2)}
for(int i = 0; i < this.cards.size(); i++, yStart -= 45)}
if (this.gameOver)}
temp = rotate(cards.get(i).getImage(), 90);//work - open cards
{else}
temp = rotate(backCard, 90);
{
g.drawImage(temp, xStart, yStart, 120, 70, null);

{

{else if(this.ID == 0}(
for(int i = 0; i < this.cards.size(); i++, yStart += 45)}
if (this.gameOver)}
temp = rotate(cards.get(i).getImage(), 90);//work - open cards
{else}
temp = rotate(backCard, 90);
{
g.drawImage(temp, xStart, yStart, 120, 70, null);
{
{
else}
for(int i = 0; i < this.cards.size(); i++, xStart += 45)}
**/
*
/*
if(isIncludeIntempCardIndex(i))}
g.drawImage(cards.get(i).getImage(), xStart, 380, 70, 120, null);
{else
g.drawImage(cards.get(i).getImage(), xStart, yStart, 70, 120, null);
{

{
{
**/
 *method that sort the cards in hand according to serial
/*
private void SortHand}()
Collections.sort(cards, new SortCards());
{

private void SortCardList(List<Card> tempList)}
Collections.sort(tempList, new SortCards());
{
//return the highest card in hand
public Card findGradestCard} ()
Card maxCard = Collections.max(this.cards,new ComparatorCards());
return maxCard;
{

//works good
```

```
public List<List<Card>> findAllGrowingSequenceNew} ()
List<Card> grow = new LinkedList<Card;()<
int tempIndex = -1;
List<List<Card>> cardsSequences = new LinkedList<List<Card;()<<
grow.add(this.cards.get(0));
for (int i = 0; i < this.cards.size() - 1; i++)}
if ((tempIndex = findNextCardSequence(this.cards, i + 1, cards.get(i))) != -1)}
grow.add(cards.get(tempIndex));
 {else}
cardsSequences.add(grow);
grow = new LinkedList<Card;()<
grow.add(cards.get(i + 1));
{
{
cardsSequences.add(grow);
return cardsSequences;
{

**/
 *only for same numbers
@ *param HandCards
@ *return a list of all the sequences in hand (only of same number). every list in the list of lists will include a
sequence of cards
/*
public List<List<Card>> findAllSequence (List<Card> HandCards)}
int checkNumber;
List<List<Card>> cardsSequences = new LinkedList<List<Card;()<<
List<Card> cardsLowSequence = new LinkedList<Card;()<
cardsLowSequence.add(HandCards.get(0));
for (int i = 0; i < HandCards.size() - 1 ; i++)}
checkNumber = HandCards.get(i).getNumber;()
if (HandCards.get(i + 1).getNumber() == checkNumber)}
cardsLowSequence.add(HandCards.get(i + 1));
 {else}
cardsSequences.add(cardsLowSequence);
cardsLowSequence = new LinkedList<Card;()<
cardsLowSequence.add(HandCards.get(i + 1));
{
{
cardsSequences.add(cardsLowSequence);// needs to add the last one because not added
return cardsSequences;

{
**/
*
@ *return a list of cards which is the longest sequence
/*
public List<Card> getLongestSequence} ()
List<List<Card>> cardsSequences = findAllSequence(this.cards);//if there is 2 sequences with the same
length
List<List<Card>> cardsGrowingSequences = findAllGrowingSequenceNew;()
//it will take the one with the bigger number
List<Card> longestSequence = Collections.max(cardsSequences,new CompareSequence());
```

```
cardsGrowingSequences.add(longestSequence);// add to the list of list
//the biggest sum list from the regular sequence
longestSequence = Collections.max(cardsGrowingSequences,new CompareSequence());//then call to
collections.max
//to get again the biggest sum now.
if (longestSequence.size() < 2)}
longestSequence.remove;(0)
//before choosing the biggest card, need to check if the card in the open pile is better to take
longestSequence.add(findGradestCard());// if the biggest sequence is only 1 card
//so prefer to throw the biggest card
{
*/
 *if there is a sequence of 2 jokers they will not be thrown. Instead, send a new cards in hand without the
 *jokers. if there is only one joker he won't be thrown!If there is only 2 joker say yaniv!
/*

if (findGradestCard().getNumber() > sumCardsList(longestSequence))}
 //if the biggest card amount is bigger than the sequence amount, prefer to throw the biggest card
longestSequence.clear;()
longestSequence.add(findGradestCard());
{
return longestSequence;
{

public List<Card> takeTheSameCardInOpenPile(Card card)}
List<Card> longestSequence = this.getLongestSequence;()
**/
 *if the sequence to throw is the same number like in the pile, prefer to save the sequence,
 *throw the second card prefers to throw and take the card from the open, at the following course will throw
 *the new sequence.
/*
if(longestSequence.get(0).getNumber() == card.getNumber() && this.cards.size() > 1)}
List<Card> tempCardsInHand = new LinkedList<Card;()<
for(int i = 0; i < this.cards.size(); i++)}
tempCardsInHand.add(this.cards.get(i));
{
return createNewCardsSequencesWitoutGetList(tempCardsInHand, longestSequence);
 **/work but not if there is joker in hand
for(int i = 0; i < this.cards.size() - longestSequence.size(); i ++)}
tempCardsInHand.add(this.cards.get(i));
/**{
{
return longestSequence;
{

public List<Card> createNewCardsSequencesWitoutGetList (List<Card> cardsSequences,List<Card>
CardsToRemoveFromSequence)}
Collections.sort(CardsToRemoveFromSequence, new SortCards());
List<Card> tempCardsInHand = new LinkedList<Card;()<
for (int i = 0; i < cardsSequences.size(); i++)}
tempCardsInHand.add(cardsSequences.get(i));
{
tempCardsInHand.removeAll(CardsToRemoveFromSequence);
```

```
tempCardsInHand = Collections.max(findAllSequence(tempCardsInHand), new CompareSequence());
return tempCardsInHand;
{

**/
 *calculate the points in hand in a current time.
@ *return the sum of the points in hand
/*
public int getPointsInHand} ()
int sum = 0;
for(int i = 0; i < this.cards.size(); i++)}
if (this.cards.get(i).getNumber() > 10)}
sum += 10;
 {else}
sum += this.cards.get(i).getNumber;()
{
{
return sum;
{

private int sumCardsList(List<Card> list)}
int sum = 0;
for(int i = 0; i < list.size(); i++)}
sum += list.get(i).getNumber;()
{
return sum;
{


**/
 *check if regular sequence can be throw
@ *param list
@ *return
/*
public boolean isCanBeThrowen(List<Card> list)}
if(!isJokerInList(list))}
SortCardList(list);
if(list.size() <= 1)}
return true;
 {else if (list.get(0).getNumber() == list.get(1).getNumber} (()
for(int i = 1; i < list.size(); i++)}
if(list.get(i).getNumber() != list.get(i - 1).getNumber())}
return false;
{
{
return true;
 {else if (list.get(1).getNumber() == (list.get(0).getNumber() + 1) && (list.get(1).getShape() ==
list.get(0).getShape&& (()
list.size() > 2} (
for (int i = 1; i < list.size(); i ++)}
if (list.get(i).getNumber() != list.get(i - 1).getNumber() + 1 || list.get(i).getShape() != list.get(i - 1).getShape())
}
return false;
```

```
{
{
return true;
 {else}
return false;
{
 {else}
if (isCanBeThrowen(getListWithoutJoker(list)))}
return true;
 //else will be onle for sequence that need to be complete by joker(3 2 | 5 4 2)
 {else}
SortCardList(list);
int countJoker = countJoker(list);
List<Card> tmpList = getListWithoutJoker(list);
if (tmpList.size() >= 2)}
for (int i = 0; i < tmpList.size() - 1; i++)}
if(tmpList.get(i).getNumber() == tmpList.get(i + 1).getNumber() - 1
 &&tmpList.get(i).getShape() == tmpList.get(i + 1).getShape}(()
continue;
 {else if (tmpList.get(i).getNumber() == tmpList.get(i + 1).getNumber() - 2
 &&tmpList.get(i).getShape() == tmpList.get(i + 1).getShape() && countJoker >= 1}(
countJoker;--
continue;
{else if (tmpList.get(i).getNumber() == tmpList.get(i + 1).getNumber() - 2
 &&tmpList.get(i).getShape() == tmpList.get(i + 1).getShape() && countJoker >= 2} (
countJoker;--
continue;
{else}
return false;
{
{
return true;
{

{
{
return false;
{

public boolean isJokerInList(List<Card> list)}
for(int i = 0; i < list.size();i++)}
if(list.get(i).getNumber() == 0)}
return true;
{
{
return false;
{

public void chooseUP(Card card, int id, int index)}
tempSequenceCards.add(card);
tempCards.add(card);
tempCardIndex.add(index);
tempCardSerial.add(cards.get(index).getSerial());// adds the serial instead of the index
```

```
{
public void chooseDOWN(Card card, int id, int index)}
if(findIndexIntempCardIndex(index) != -1 && findIndexInSerialCardIndex(card.getSerial()) != -1)}
tempCardIndex.remove(findIndexIntempCardIndex(index));
tempCardSerial.remove(findIndexInSerialCardIndex(card.getSerial()));
tempCards.remove(findIndexInCardsList(card,tempCards));
tempSequenceCards.remove(findIndexInCardsList(card,tempSequenceCards));
{else}
System.out.println("not found");
{
{
private List<Card> getListWithoutJoker(List<Card> list)}
List<Card> noJoker = new LinkedList<Card;()<
for (int i = 0; i < list.size(); i ++)}
if (list.get(i).getNumber() != 0)}
noJoker.add(list.get(i));
{
{
return noJoker;

{
**/
*
@ *return number of joker in list
/*
private int countJoker(List<Card> list)}
int jokerCount = 0;
for(int i = 0; i < list.size(); i++)}
if(list.get(i).getNumber() == 0)}
jokerCount;++
{
{
return jokerCount;
{
**/
*
@ *param list
@ *param index from that index the method will look for the next card in the sequence
@ *return the index of the next card in sequence. If there is not such a card, the method return -1.
/*
private int findNextCardSequence(List <Card> list, int index, Card temp)}
for (int i = index; i < list.size(); i++)}
if(list.get(i).getNumber() - 1 == temp.getNumber() && list.get(i).getShape() == temp.getShape())}
return i;
{
{
return -1;
{
public boolean isCompleteSequence(Card card)}
 //boolean isComplete = false;
if (isIncludeInHandCards(card))}
return true;
{
```

```
List<List<Card>> growingSequence = findAllGrowingSequenceNew;()
for (int i = 0; i < growingSequence.size(); i++)}
if (growingSequence.get(i).size() > 1)}
if (growingSequence.get(i).get(0).getNumber() == card.getNumber() + 1&&
growingSequence.get(i).get(0).getShape() == card.getShape|| ()
growingSequence.get(i).get(growingSequence.get(i).size() - 1).getNumber() == card.getNumber() - 1
 &&growingSequence.get(i).get(growingSequence.get(i).size() - 1).getShape() == card.getShape}( ()
return true;
{
 {else}
return false;
{
{
return false;
{

{
```

מחלקה SortCards

```
package yaniv;

import java.util.Comparator;

public class SortCards implements Comparator<Card> {
/**
 * used for sort the cards in hand so in the screen it appear in the correct order(from small
to big - left to right)
 * The method return the difference between the two cards.It used in collection sort in Player
class.
 */
	@Override
	public int compare(Card o1, Card o2) {
		return o1.getSerial() - o2.getSerial();
	}

}
```

מחלקה Stats

```
package yaniv;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import game.main;


public class Stats}
```

```
private String nameOfWinner; // 1 - line
private int maxScore; // 2 - line
private File f;

public Stats() throws IOException}
f = new File("Scores");
if(!f.exists())}
this.nameOfWinner = "no winner yet;"
this.maxScore = 100000000;//point must be smaller than that
f.createNewFile;()
f.setWritable(true);
writeToFile;()
 {else}
readFromFile;()
{
{

**/
 *read from the file the data
@ *throws IOException
/*
public void readFromFile() throws IOException}
BufferedReader br = new BufferedReader(new FileReader(f));
String st;
//lines = new ArrayList<Integer;()<
st = br.readLine;()
this.maxScore = Integer.parseInt(st);
st = br.readLine;()
this.nameOfWinner = st;
br.close;()
//updateLocalVariables;()
{

**/
 *write to file the stats
@ *throws IOException
/*
public void writeToFile() throws IOException}
FileWriter fw = new FileWriter(f.getAbsoluteFile());
BufferedWriter bw = new BufferedWriter(fw);
bw.write(Integer.toString(this.maxScore));
bw.newLine;()
bw.write(this.nameOfWinner);
bw.newLine;()
bw.close;()
{


public String getNameOfWinner} ()
return nameOfWinner;
{

public void setNameOfWinner(String nameOfWinner)}
this.nameOfWinner = nameOfWinner;
{

public int getMaxScore} ()
```

```
return maxScore;
{

public void setMaxScore(int maxScore)}
this.maxScore = maxScore;
{

public File getF} ()
return f;
{

public void setF(File f)}
this.f = f;
{
**/
 *public void updateLocalVariables} ()
numOfWon = lines.get;(0)
numOfLost = lines.get;(1)
goalsScored = lines.get;(2)
goalsConceeded = lines.get;(3)
games = lines.get;(4)
numOfTies = lines.get;(5)
{
/*


**/
 *update the stats of the game
@ *param goalsPlayer1
@ *param goalsPlayer2
@ *throws IOException
 *calculate average point for one game
/*
public void updateStats(String name, int points) throws IOException}
readFromFile;()
if (points / main.game.getGameCounter() < this.maxScore)}
this.maxScore = points / main.game.getGameCounter;()
this.nameOfWinner = name;
{

//                                                                                    {
writeToFile;()
{
{
```

מחלקה writer

```
package yaniv;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;
```

```
public class writer}
private String winnerName;
private int highScore;
public writer} ()

{

public void writing} ()
File statText = new File("E:/Java/Reference/bin/images/statsTest.txt");
FileOutputStream is;
try}
is = new FileOutputStream(statText);
OutputStreamWriter osw = new OutputStreamWriter(is);
Writer w = new BufferedWriter(osw);
w.write("POTATO!!!");
w.close;()
 {catch (IOException e} (
e.printStackTrace;()
{

{

{
```

ממשק Drawable

```
package graphic;

import java.awt.Graphics;

public interface Drawable {
	void draw(Graphics g);
}
```

מחלקה GamePanel

```
package graphic;

import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import game.main;
import yaniv.Card;
import yaniv.Game;
import yaniv.OpenPile;
import yaniv.Pile;
import yaniv.Player;


public class GamePanel extends JPanel}
private static final int speed_Of_Card_Animation = 1; // recommended 1. anyway not above 3!!
private List<Drawable> objectsToDraw;
Image backround;
Image yanivImage;
JButton yanivButton;
Game gameTemp;
JPanel p;
static int minToCallYaniv = 9; // the minimal sum to call yaniv
private JTextArea p1Points, p2Points, p3Points;
private int sourceX;
private int sourceY;
private Card currentCard;
private boolean isImageRotated;
private int destinationX = 100;
private int destinationY = 100;
private Image backCard = null;
private boolean seeDetails = false;
private static List<JTextArea> pointsList = new ArrayList<JTextArea;()<
private static List<Integer> pointsInInt = new ArrayList<Integer;()<
public static Boolean isRun = false;
private static GamePanel instance = null;
**/
 *get a list of players that their cards will draw
@ *param players
/*
public GamePanel(List<Player> players,OpenPile openPile,Pile pile, Game game)}
instance = this;
if (!isRun)}
createPlayersPointsJtexts;()
isRun = true;
 {else}
updateJtextsSecond;()
{
setBackround;()
p = this;
setLayout(null);
gameTemp = game;
objectsToDraw = new ArrayList<Drawable;()<
for(int i = 0; i < players.size(); i++)}
objectsToDraw.add(players.get(i));
```

```
{
objectsToDraw.add(openPile);
objectsToDraw.add(pile);
createYanivButton;()
add(this.yanivButton);
this.yanivButton.setEnabled(false);// at first the button can not be clicked - works v
addMouseListener(new MouseAdapter}()
@Override
public void mouseClicked(final MouseEvent e)}
Thread resolveClicked = new Thread(new Runnable}()
public void run}()
gameTemp.whatClicked(e.getX(), e.getY(), p);
{
;({
resolveClicked.start;()

{
;({
try}
backCard  = ImageIO.read(new File("pictures/gray_back.png"));
 {catch (IOException e}(
e.printStackTrace;()
{

{

@Override
public void paintComponent(Graphics g)}
g.drawImage(backround, 0, 0, null);//draw the background
Rectangle playerGreenRecangle =
gameTemp.getPlayers().get(gameTemp.getCurrentPlayer()).getCurrentPlayerRectangle;()
g.setColor(Color.GREEN);
((Graphics2D)g).setStroke(new BasicStroke(4));
if(!main.game.getPlayers().get(0).getGameOver())}
g.drawRect(playerGreenRecangle.x, playerGreenRecangle.y, playerGreenRecangle.width,
playerGreenRecangle.height);
{
for(Drawable drawable : objectsToDraw)}
drawable.draw(g);
{
if(gameTemp.getCurrentPlayer() != 1)}
this.yanivButton.setEnabled(false);
{else}
 //only if the sum is lower than 9 can click the yaniv
if(gameTemp.getPlayers().get(1).sumCards() < minToCallYaniv)}
this.yanivButton.setEnabled(true);
{

{
if (currentCard != null)}
if (isImageRotated)}
currentCard.draw(g, sourceX, sourceY, backCard, seeDetails);// if "see details" true so you can
see the card if not you will see his back
{else}
currentCard.draw(g, sourceX, sourceY);

{
```

```
{
{

public boolean isSeeDetails}()
return seeDetails;
{

public void setSeeDetails(boolean seeDetails)}
this.seeDetails = seeDetails;
{

**/
 *set the background image
/*
private void setBackround}()
try}
backround = ImageIO.read(new File("pictures/backroundblue.jpg"));
{catch(IOException e}(
System.out.println("file not found");
{
Dimension size = new Dimension(backround.getWidth(null), backround.getHeight(null));
setPreferredSize(size);
setMinimumSize(size);
setMaximumSize(size);
setSize(size);
setLayout(new BorderLayout());
{

private void createYanivButton}()
try}
this.yanivImage = ImageIO.read(new File("pictures/yanivENA.png"));
 {catch (IOException e}(
e.printStackTrace;()
{
this.yanivButton = new JButton;()
this.yanivButton.setIcon(new ImageIcon(this.yanivImage));
this.yanivButton.setBounds;(77 ,116 ,450 ,700)
yanivButton.setContentAreaFilled(false);
yanivButton.setBorderPainted(false);
this.yanivButton.addActionListener(new ActionListener}()
@Override
public void actionPerformed(ActionEvent e)}
gameTemp.yanivButton;()

{
;({

{
**/
 *get first values of the points
/*
private void createPlayersPointsJtexts}()
p1Points = new JTextArea;()
p1Points.setText;("0")
p1Points.setFont(new Font("Arial Black", Font.BOLD, 22));
p1Points.setBounds;(40 ,60 ,80 ,80)
p1Points.setEditable(false);
```

```
p1Points.setOpaque(false);

p2Points = new JTextArea;()
p2Points.setText;("0")
p2Points.setFont(new Font("Arial Black", Font.BOLD, 22));
p2Points.setBounds;(40 ,60 ,500 ,300)
p2Points.setEditable(false);
p2Points.setOpaque(false);


p3Points = new JTextArea;()
p3Points.setText;("0")
p3Points.setFont(new Font("Arial Black", Font.BOLD, 22));
p3Points.setBounds;(40 ,60 ,80 ,900)
p3Points.setEditable(false);
p3Points.setOpaque(false);


p1Points.setVisible(true);
p2Points.setVisible(true);
p3Points.setVisible(true);

pointsList.add(p1Points);
pointsList.add(p2Points);
pointsList.add(p3Points);

for (int i = 0; i <pointsList.size(); i++)}
pointsInInt.add;(0)
{

add(p1Points);
add(p2Points);
add(p3Points);

{
**/
 *There is another method for restore the previous values.
/*
private void updateJtextsSecond} ()
p1Points = new JTextArea;()
p1Points.setText(String.valueOf(pointsInInt.get(0)));
p1Points.setFont(new Font("Arial Black", Font.BOLD, 22));
p1Points.setBounds;(40 ,60 ,80 ,80)
p1Points.setEditable(false);
p1Points.setOpaque(false);

p2Points = new JTextArea;()
p2Points.setText(String.valueOf(pointsInInt.get(1)));
p2Points.setFont(new Font("Arial Black", Font.BOLD, 22));
p2Points.setBounds;(40 ,60 ,500 ,300)
p2Points.setEditable(false);
p2Points.setOpaque(false);

p3Points = new JTextArea;()
p3Points.setText(String.valueOf(pointsInInt.get(2)));
p3Points.setFont(new Font("Arial Black", Font.BOLD, 22));
```

```
p3Points.setBounds;(40 ,60 ,80 ,900)
p3Points.setEditable(false);
p3Points.setOpaque(false);

p1Points.setVisible(true);
p2Points.setVisible(true);
p3Points.setVisible(true);

pointsList.add(p1Points);
pointsList.add(p2Points);
pointsList.add(p3Points);

add(p1Points);
add(p2Points);
add(p3Points);
{

public static void updatePoints(List<Integer> points)}
for (int i = 0; i < points.size(); i++)}
pointsList.get(i).setText(String.valueOf(points.get(i)));
pointsInInt.set(i, points.get(i));
{
GamePanel.getIntance().repaint;()
{
public static GamePanel getIntance}()
return instance;
{
//of pile
public void setDestinationAndSource(int sorX, int sorY, int desX, int desY, Card card)}
//setDestination;()
System.out.println;()
sourceX = sorX;//place of the pile
sourceY = sorY;//place of the pile
destinationX = desX;
destinationY = desY;
currentCard = card;
if (main.game.getCurrentPlayer() == 2 || main.game.getCurrentPlayer() == 0)}
isImageRotated = true;
{
else
isImageRotated = false;
{

public void setDestinationThrow(int index)}
int currentPlayer = main.game.getCurrentPlayer;()
setSourceThrow(index);
destinationX = 470;
destinationY = 100;
currentCard = main.game.getPlayers().get(currentPlayer).getCards().get(index);
isImageRotated = false;
{

private void setSourceThrow(int index)}
if (main.game.getCurrentPlayer() == 0)}
sourceY = 120 + 45 * index;
sourceX = 80;
```

```
{
else if (main.game.getCurrentPlayer() == 1)}
sourceY = 400;
sourceX = 385 + 45 * index;
{
else if (main.game.getCurrentPlayer() == 2)}
sourceY = 300 - 45 * index;
sourceX = 800;
{
{


public void setCurrrentCard}()
currentCard = null;
{
public boolean isNotInDestionation}()
if (sourceX != destinationX || sourceY != destinationY)
}
if (sourceY < destinationY)
sourceY += speed_Of_Card_Animation;
else if (sourceY > destinationY)
sourceY -= speed_Of_Card_Animation;
if (sourceX < destinationX)
sourceX += speed_Of_Card_Animation;
else if (sourceX > destinationX)
sourceX -= speed_Of_Card_Animation;
{
return (sourceX != destinationX || sourceY != destinationY);// works always but speed must be
lower than 5
{

public Card getCurrentCard}()
return currentCard;
{

{
```

מחלקה InstructionScreen

```
package graphic;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;


import game.main;
```

```
public class InstructionScreen extends JPanel}
Image backround;
private JButton backButton;

public InstructionScreen} ()
super;()
setBackround;()
createBackButton;()
add(this.backButton);
this.setLayout(null);
{

@Override
protected void paintComponent(Graphics g)}
g.drawImage(backround, 0, 0, null);//draw the background
{

**/
 *set the background image
/*
private void setBackround} ()
try}
backround = ImageIO.read(new File("pictures/instructions.png"));
{catch(IOException e}(
System.out.println("file not found");
{
Dimension size = new Dimension(backround.getWidth(null), backround.getHeight(null));
setPreferredSize(size);
setMinimumSize(size);
setMaximumSize(size);
setSize(size);
setLayout(new BorderLayout());
{

private void createBackButton} ()
Image backImage = null;

try}
backImage = ImageIO.read(new File("pictures/backButton1.png"));
 {catch (IOException e} (
e.printStackTrace;()
{
backButton = new JButton;()
backButton.setIcon(new ImageIcon(backImage));
backButton.setBounds;(90 ,90 ,570 ,20)
backButton.setContentAreaFilled(false);
backButton.setBorderPainted(false);
final InstructionScreen thisPanel = this;
backButton.addActionListener(new ActionListener} ()
@Override
public void actionPerformed(ActionEvent e)}
System.out.println("back clicked");
//main.frame.setVisible(false);
main.frame.invalidate;()
main.frame.remove(thisPanel);
```

```
main.frame.setContentPane(new OpenScreen());
main.frame.validate;()
main.frame.setSize;(400,400)

{
;({

{
{
```

מחלקה LoseScreen

```
package graphic;


import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import game.main;
import yaniv.Game;
import yaniv.Stats;

public class LoseScreen extends JPanel}
Image youLose, resume, leave;
JButton btResume, btLeaveGame;
String name;
int maxScore;
public LoseScreen(String playerName, int maxPoints)}
name = playerName;
maxScore = maxPoints;
setLayout(null);
main.frame.setBackground(new Color(255, 255, 255));
try}
youLose = ImageIO.read(new File("pictures/youLose3.png"));
resume = ImageIO.read(new File("pictures/continueBt.png"));
leave = ImageIO.read(new File("pictures/leavegameBt.png"));
 {catch (IOException e}(
e.printStackTrace;()
{
createResumebutton;()
createLeavebutton;()
add(btResume);
add(btLeaveGame);
{

@Override
protected void paintComponent(Graphics g)}
g.drawImage(youLose, 180, 50, 900, 456, null);
```

```
{

private void createResumebutton}()
btResume = new JButton;()
btResume.setIcon(new ImageIcon(resume));
btResume.setBounds;(79 ,217 ,600 ,100)
btResume.setContentAreaFilled(false);
btResume.setBorderPainted(false);
btResume.addActionListener(new ActionListener}()

@Override
public void actionPerformed(ActionEvent e)}
//instead of create new game, make a method that resume to the first situation
main.game.newGame;()
main.frame.setSize;(600 ,1000)
main.frame.setContentPane(new GamePanel(main.game.getPlayers(), main.game.getOpenPile(),
main.game.getPile(), main.game));
main.frame.validate;()
{
;({
{
private void createLeavebutton}()
btLeaveGame = new JButton;()
btLeaveGame.setIcon(new ImageIcon(leave));
btLeaveGame.setBounds;(81 ,213 ,600 ,900)
btLeaveGame.setContentAreaFilled(false);
btLeaveGame.setBorderPainted(false);
btLeaveGame.addActionListener(new ActionListener}()

@Override
public void actionPerformed(ActionEvent e)}
try}
Stats stats = new Stats;()
stats.updateStats(main.game.getPlayerName(), maxScore);
System.out.println("write to file!!!!!");
 {catch (IOException e1}(
e1.printStackTrace;()
{
GamePanel.isRun = false;
main.game = new Game;()
main.frame.setBackground(null);
main.frame.setSize;(400,400)
main.frame.setContentPane(new OpenScreen());
main.frame.validate;()
{
;({
{

{
```

```
package graphic;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import game.main;
import yaniv.Stats;

public class MaxScores extends JPanel}
private Stats stats;
Image backround;
JButton backButton;
JTextArea maxPoints, championName, allTimesWinner;
String name;
int points;
public MaxScores} ()
try}
stats = new Stats;()
points = stats.getMaxScore;()
name = stats.getNameOfWinner;()
 {catch (IOException e1} (
 //TODO Auto-generated catch block
e1.printStackTrace;()
{
**/
 *try}
this.stats = new Stats;()
 {catch (IOException e} (
e.printStackTrace;()
System.out.println("error while trying to make new stst");
{
/*



setBackround;()
createBackButton;()
createJtextEreaName;()
createJtextEreawinnersOfAllTimes;()
add(backButton);
createJtextEreaPoints;()
```

```
this.setLayout(null);
{

private void createBackButton}()
Image backImage = null;

try}
backImage = ImageIO.read(new File("pictures/backButton1.png"));
 {catch (IOException e}(
e.printStackTrace;()
{
backButton = new JButton;()
backButton.setIcon(new ImageIcon(backImage));
backButton.setBounds;(90 ,90 ,460 ,20)
backButton.setContentAreaFilled(false);
backButton.setBorderPainted(false);
backButton.addActionListener(new ActionListener}()
@Override
public void actionPerformed(ActionEvent e)}
System.out.println("back clicked");
main.frame.setContentPane(new OpenScreen());
main.frame.validate;()
main.frame.setSize;(400,400)

{
;({
{

@Override
protected void paintComponent(Graphics g)}
g.drawImage(backround, 0, 0, null);//draw the background
{
**/
 *set the background image
/*
private void setBackround}()
try}
backround = ImageIO.read(new File("pictures/noteBook.jpg"));
{catch(IOException e}(
System.out.println("file not found");
{
Dimension size = new Dimension(backround.getWidth(null), backround.getHeight(null));
setPreferredSize(size);
setMinimumSize(size);
setMaximumSize(size);
setSize(size);
setLayout(new BorderLayout());
{

public void createJtextEreaPoints}()
maxPoints = new JTextArea;()
maxPoints.setText("Points: " + String.valueOf(points));
maxPoints.setFont(new Font("Arial Black", Font.BOLD, 22));
maxPoints.setBounds;(30 ,300 ,130 ,75)
maxPoints.setEditable(false);
maxPoints.setOpaque(false);
```

```
add(maxPoints);
{

public void createJtextEreaName}()
championName = new JTextArea;()
championName.setText("The champion is: " + name);
championName.setFont(new Font("Arial Black", Font.BOLD, 22));
championName.setBounds;(30 ,600 ,100 ,75)
championName.setEditable(false);
championName.setOpaque(false);
add(championName);
{

public void createJtextEreawinnersOfAllTimes}()
allTimesWinner = new JTextArea;()
allTimesWinner.setText("Winners Of All Times:");
allTimesWinner.setFont(new Font("Arial Black", Font.BOLD, 22));
allTimesWinner.setBounds;(30 ,300 ,70 ,75)
allTimesWinner.setEditable(false);
allTimesWinner.setOpaque(false);
add(allTimesWinner);
{
{
```

מחלקה OpenScreen

```
package graphic;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

import game.main;
import yaniv.Game;

public class OpenScreen extends JPanel}
JButton startButton, instructionsButton, gobletButton;
Image startButtonImag, instructionsButtonImag, gobletImage;
JTextField enterNameTextField;
Image backround;
```

```
public OpenScreen} ()
setLayout(null);
try}
startButtonImag = ImageIO.read(new File("pictures/btStart2.png"));
instructionsButtonImag = ImageIO.read(new File("pictures/btInstructions.png"));
gobletImage = ImageIO.read(new File("pictures/goblet.png"));
 {catch (IOException e} (
e.printStackTrace;()
{
createStartbutton;()
createInstructionButton;()
createGobletImageButton;()
createJtextFieldName;()
add(startButton);
add(instructionsButton);
add(gobletButton);
add(enterNameTextField);
setBackround;()
{

@Override
protected void paintComponent(Graphics g)}
super.paintComponent(g);
g.drawImage(backround, 0, 0, null);//draw the background
{


private void createStartbutton} ()
startButton = new JButton;()
startButton.setIcon(new ImageIcon(startButtonImag));
startButton.setBounds;(73 ,186 ,100 ,112)
startButton.setContentAreaFilled(false);
startButton.setBorderPainted(false);
startButton.addActionListener(new ActionListener} ()

@Override
public void actionPerformed(ActionEvent e)}
main.game.setPlayerName(enterNameTextField.getText());
main.frame.setContentPane(new GamePanel(main.game.getPlayers(), main.game.getOpenPile(),
main.game.getPile(), main.game));
main.frame.validate;()
main.frame.setSize;(600 ,1000)
main.frame.setLocationRelativeTo(null);
{
;({
{

private void createInstructionButton} ()
instructionsButton = new JButton;()
instructionsButton.setIcon(new ImageIcon(instructionsButtonImag));
instructionsButton.setBounds;(59 ,55 ,310 ,325)
instructionsButton.setContentAreaFilled(false);
instructionsButton.setBorderPainted(false);
instructionsButton.addActionListener(new ActionListener} ()

@Override
```

```
public void actionPerformed(ActionEvent e)}
System.out.println("Instruction pushed");
main.frame.setContentPane(new InstructionScreen());
main.frame.validate;()
main.frame.setSize;(1280,720)

{
;({
{
**/
 *create the goblet button, when pushed go to the Max Score screen
/*
private void createGobletImageButton} ()
gobletButton = new JButton;()
gobletButton.setIcon(new ImageIcon(gobletImage));
gobletButton.setBounds;(47 ,57 ,315 ,0)
gobletButton.setContentAreaFilled(false);
gobletButton.setBorderPainted(false);
gobletButton.addActionListener(new ActionListener} ()

@Override
public void actionPerformed(ActionEvent e)}
System.out.println("gobletButton pushed");
main.frame.setContentPane(new MaxScores());
main.frame.validate;()
main.frame.setSize;(500,600)
{
;({
{
**/
 *set the background image
/*
private void setBackround} ()
try}
backround = ImageIO.read(new File("pictures/option2.jpg"));
{catch(IOException e}(
System.out.println("file not found");
{
Dimension size = new Dimension(backround.getWidth(null), backround.getHeight(null));
setPreferredSize(size);
setMinimumSize(size);
setMaximumSize(size);
setSize(size);
setLayout(new BorderLayout());
{

private void createJtextFieldName} ()
enterNameTextField = new JTextField(" your name:");
enterNameTextField.setOpaque(false);
enterNameTextField.addFocusListener(new FocusListener} ()
@Override
public void focusLost(FocusEvent e)}

{
 //use like hint : disappear the previous text and put""
@Override
public void focusGained(FocusEvent e)}
```

```
enterNameTextField.setText;("")
{
;({
enterNameTextField.setBounds;(200,30 ,95,230)
{

{
```

מחלקה WinningScreen

```
package graphic;


import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JPanel;
import game.main;
import yaniv.Game;
import yaniv.Stats;

public class WinningScreen extends JPanel}
Image youWin, resume, leave;
JButton btResume, btLeaveGame;
String name;
int maxScore;
private Image backround;
public WinningScreen(String playerName, int maxPoints)}
name = playerName;
maxScore = maxPoints;
setLayout(null);
main.frame.setBackground(new Color(255, 255, 240));
try}
youWin = ImageIO.read(new File("pictures/youWin.jpg"));
resume = ImageIO.read(new File("pictures/continueBt.png"));
leave = ImageIO.read(new File("pictures/leavegameBt.png"));
 {catch (IOException e}(
e.printStackTrace;()
{
createResumebutton;()
createLeavebutton;()
add(btResume);
add(btLeaveGame);
{

@Override
protected void paintComponent(Graphics g)}
g.drawImage(backround, 0, 0, null);//draw the background
g.drawImage(youWin, 360, 50, 500, 499, null);
```

```
{

private void createResumebutton} ()
btResume = new JButton;()
btResume.setIcon(new ImageIcon(resume));
btResume.setBounds;(79 ,217 ,600 ,100)
btResume.setContentAreaFilled(false);
btResume.setBorderPainted(false);
btResume.addActionListener(new ActionListener} ()

@Override
public void actionPerformed(ActionEvent e)}
main.game.newGame;()
main.frame.setSize;(600 ,1000)
main.frame.setContentPane(new GamePanel(main.game.getPlayers(), main.game.getOpenPile(),
main.game.getPile(), main.game));
main.frame.validate;()
{
;({
{
private void createLeavebutton} ()
btLeaveGame = new JButton;()
btLeaveGame.setIcon(new ImageIcon(leave));
btLeaveGame.setBounds;(81 ,213 ,600 ,900)
btLeaveGame.setContentAreaFilled(false);
btLeaveGame.setBorderPainted(false);
btLeaveGame.addActionListener(new ActionListener} ()

@Override
public void actionPerformed(ActionEvent e)}
try}
Stats stats = new Stats;()
stats.updateStats(main.game.getPlayerName(), maxScore);
System.out.println("write to file!!!!!");
 {catch (IOException e1} (
e1.printStackTrace;()
{
main.game = new Game;()
GamePanel.isRun = false;
main.frame.setBackground(null);
main.frame.setSize;(400,400)
main.frame.setContentPane(new OpenScreen());
main.frame.validate;()
{
;({
{

{
```