תמונה שמכילה טקסט, אוסף תמונות

התיאור נוצר באופן אוטומטי

פרויקט גמר בJAVA

שם הפרויקט: "בול פגיעה"  
מגיש: נעם חייק  
ת.ז:325765873  
מנחה הפרויקט: אלי גוריאל

**תוכן עניינים:**

**מבוא:**מטרה.....2  
תיאור מערכת.....2

**מפרטי תוכנה:**

תיאור כללי.....3-4  
רקע תיאורטי.....4  
תיאור הבעיה האלגוריתמית.....4  
מושגים.....5  
אסטרטגיה.....6-7  
מבני נתונים.....8  
תרשים UML.....9  
סביבת עבודה ושפת תכנות.....9  
פונקציות ומחלקות ראשיות בפרויקט....9  
אלגוריתם ראשי AI.....10  
מדריך למשתמש.....11-14

**קוד פרויקט**:

getReasult.....15-16  
העתקת נתונים.....18  
AI.....19-39

**רפלקציה**....40

**ביביליוגרפיה**.....40

**מבוא**

**מטרה:**

לבחון את הידע והכישורים התכנותיים שלי בפרויקט בסדר גודל כזה. לפתח אלגוריתמים חכמים אשר מנהלים את המשחק ובנוסף ליצור "מכונה" אשר יודעת לקרוא נתונים מהלוח ולהסיק מסקנות לגביהם ואופציה בה המחשב עצמו יכול לשחק.

**תיאור המערכת:**

המערכת הינה ממשק ממוחשב למשחק הקופסא "בול פגיעה", המשחק כולל 2 שחקנים אשר משחקים אחד נגד השני. לכל שחקן יש תפקיד, שחקן 1-מצפין, שחקן 2-מפענך צופן. המצפין יבחר צופן - הצופן הינו רצף של 4 צבעים שונים זה מזה )מתוך שישה צבעים אפשריים( אשר מסודרים בסדר מסויים. מפענך הצופן ינחש את הצופן )רציפ של 4 צבעים( ויקבל פידבק בעזרת דגלים )דגל לבן – צבע נכון במקום לא נכון, דגל שחור – צבע נכון במקום נכון.

המשחק מכיר כמה אופציות:

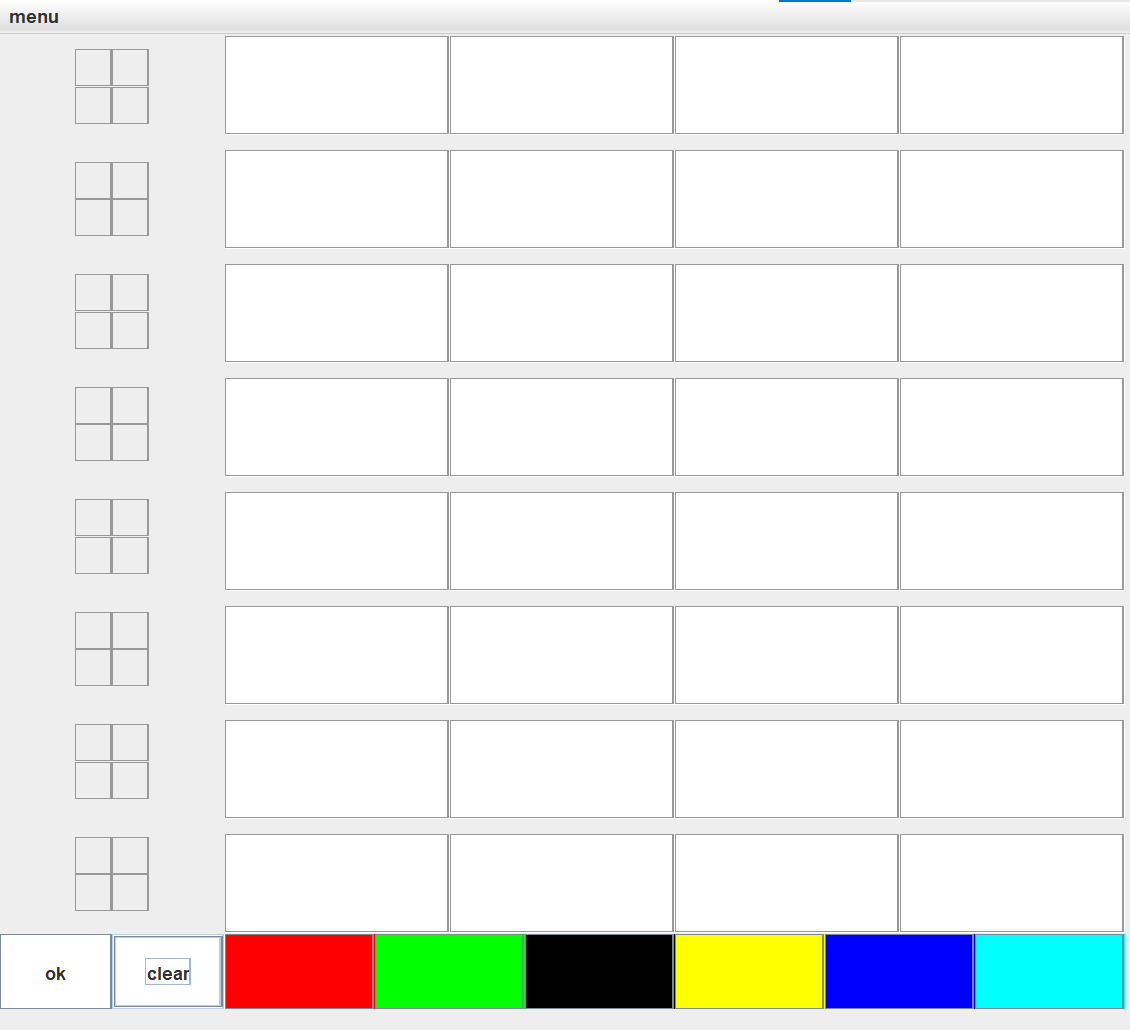
שחקן נגד שחקן  
שחקן נגד מחשב  
שחקן לבד

**מפרטי תוכנה:**

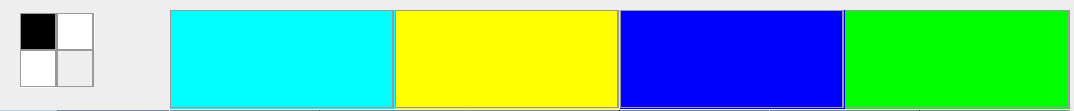
**תיאור כללי:**

המשחק בול פגיעה מכיל לוח אשר לשחקו יש שמונה נסיונות לגלות את הצופן הנסתר ולצד הלוח יש שמונה "תאים" של פידבקים לכל תור שהם מתמלאים אחרי לחיצת כפתור אוקיי ובדיקת תקינות הניחוש (ארבעה צבעים אשר שונים זה מזה)

מצב התחלתי של הלוח:

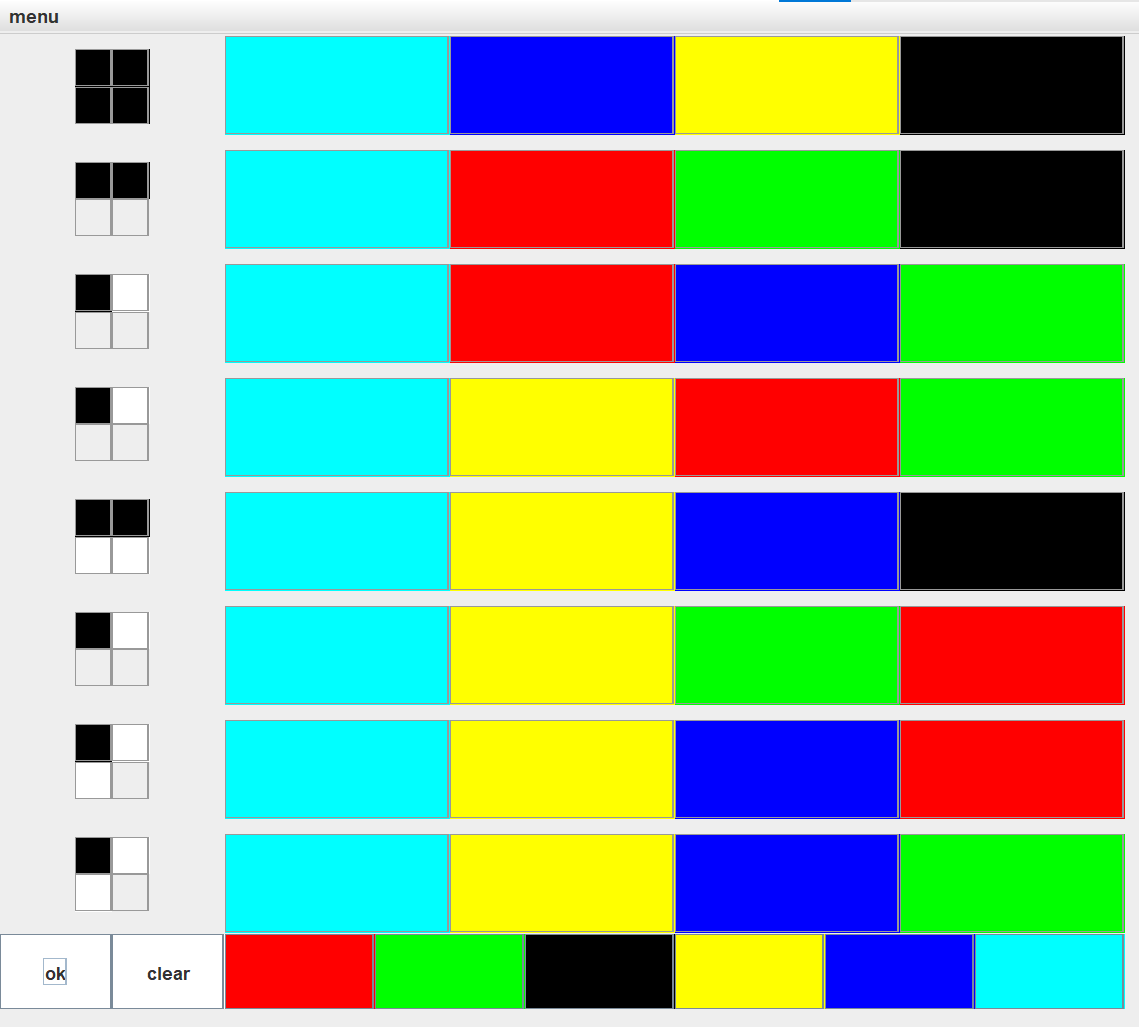


שורה בלוח לאחר הניחוש:



על פי שורה זו ניתן לראות שישנם שלוש צבעים נכונים בצופן אך רק אחד מהם נמצא במיקום הנכון.

תמונה של לוח המשחק בסופו:



ניתן לראות שבתור האחרון השחקן הצליח לגלות את הצופן (תכלת-כחול-צהוב-שחור) על ידי קבלת ארבע דגלים שחורים

**רקע תאורטי:**

בכדי להצליח לתכנת את המשחק שיחקתי במשחק המון פעמים (לבד,עם חברים,עם משפחה) ובנוסף קראתי את חוקי המשחק בצורה מעמיקה.

**תיאור הבעיה האלגוריתמית:**

במהלך העבודה על הפרויקט נתקלתי במספר בעיות אשר איתגרו אותי וגרמו לי לחדד את המחשבה שלי בכדי להצליח לגבור עליהן ולתכנת את המשחק לדוגמא:

בכדי ליצור את האופציה של שחקן נגד שחקן רציתי לתכנת את זה בצורה של תור תור ולכל שחקן יהיה לוח משלו, כוונתי הראשונה הייתה ליצור שתי לוחות משחק ובכל תור לשנות את הלוח על ידי שינוי הגדרת הנראות שלו. וכך עשיתי אך למרבה הצער רעיון זה לא עבד מכיוון שהקוד בכל זאת פועל על הלוח האחרון שנוצר ואחרי מחשבה רבה הגעתי למסקנה שאני צריך אחרי כל תור לשמור את הנתונים של הלוח ובכל תור שעובר למחוק את הלוח הקודם ולצייר לוח חדש עם הנתונים הקודמים ששמרתי.

בעיה נוספת שנתקלתי בה היא יצירת האפשרות שהמחשב ישחק בעצמו מכיוון שלמרות הרבה משחקים ששיחקתי לא הצלחתי לגלות דפוס פעולה לעבוד לגביו ובסוף אחרי חקירה מעמיקה באינטרנט והמון משחקים בסופו של דבר הצלחתי לגלות דפוס פעולה.

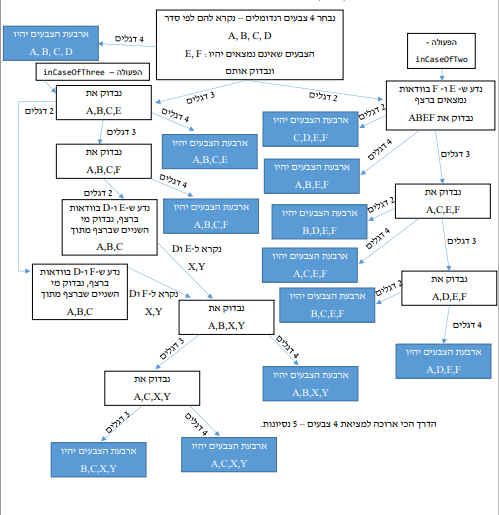
**מושגים:**

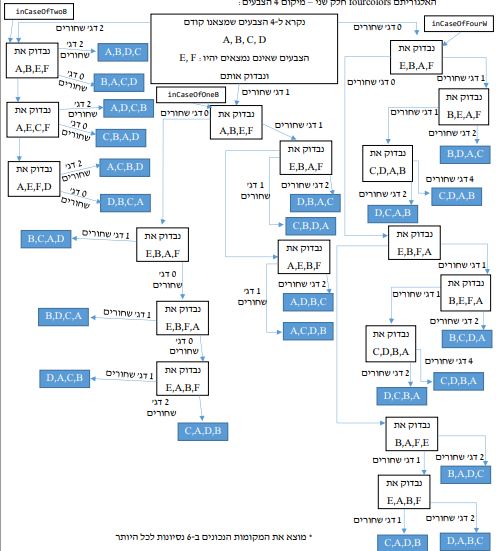
**אינטיליגנציה מלאכותית** – מחשב המתוכנת לפעול באופן המציג יכולות המאפיינות את הבינה האנושית. בהגדרה רחבה יותר, ליצור בינה מלאכותית זה "לגרום למכונה להתנהג בגרך שהיתה נחשבת אינטיליגנטית לו אדם התנהג כך" )נאמר ע"י מרווין מינסקי – מדען יהודי- אמריקאי מאבות הבינה המלאכותית(. בפרוייקט זה, הבינה המלאכותית היא היכולת של המחשב לפתור את הצופן שבחר לו המשתמש, בעזרת מערכת של החלטות ואסטרטגיה מובנת עליה עבדתי. את פירוט האסטרטגיה ניתן למצוא בעמודים 8 ו-9 תחת הכותרת "אסטרטגיה".

**קומבינטוריקה** - קומבינטוריקה היא ענף במתמטיקה בדידה שעוסק בספירה של כמויות סופיות של אובייקטים. קומבינטוריקה היא תחום רחב מאוד במתמטיקה, כך שאין לו גבולות ברורים. בפרוייקט ניתן לראות כמה תחומים של הקובינטוריקה: - צירופים )קומבינציה( - מספר האפשרויות לבחור k עצמים מתוך n עצמים שונים בלי חזרות, כאשר אין חשיבות לסדר הבחירה. בפרוייקט: כלל האפשרויות לבחירה של 4 צבעים ספציפים בסדר ספציפי מתוך 6 אפשריים, 4=k ו- 6=n - חלק ראשון של האסטרטגיה הממוחשבת. - תמורה )פרמוטציה( - סידור כלשהו של עצמים שונים בשורה. באופן פורמלי, תמורה היא פונקציה הפיכה מקבוצה סופית לעצמה. בפרוייקט: כלל האפשרויות לסידור 4 הצבעים אשר נמצאים ברצף מצפין הצופן – חלק שני של האסטרטגיה הממוחשבת

**אסטרטגיה:**

האלגוריתם חלק ראשון: מציאת ארבע צבעים נכונים:





**מבני נתונים:**

1. Check box array - **-** מבנה נתונים זה הוא מערך חד ממדי של מערכים חד ממדיים אשר שם יוצג לי הפידבק על הניסיון:

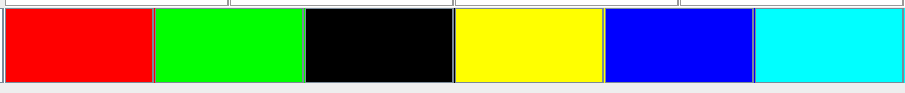


1. Board – מבנה נתונים זה הוא מעדך דו ממדי אשר מייצג לי את לוח המשחק:

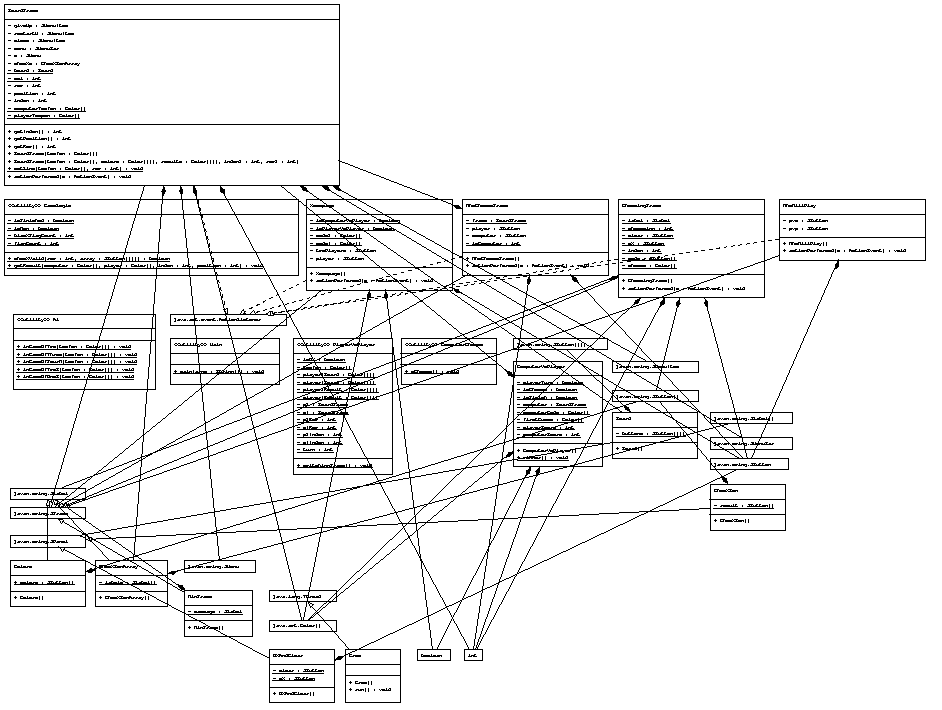
תמונה שמכילה שוג'י

התיאור נוצר באופן אוטומטי

1. Colors – מערך חד ממדי אשר מציג לי את ששת הצבעים הניתנים לבחירה:



**תרשים UML-**



**סביבת עבודה ושפת תכנות:**

את הפרויקט כתבתי IntelliJ בשפה JAVA

**תיאור ממשקים, מחלקות ופונקציות ראשיות:**

**Board** – מחלקה ששם נוצר לוח המשחק (מערך דו ממדי)

**Board frame** – מחלקה ששם נוצר המסך המשחק (לוח , פידבקים , כפתורי אוקיי ונקה , כפתורי הצבעים לבחירת הניחוש)

**HomePage** – מחלקה ששם השחקן בוחר את אופציית המשחק בה הוא רוצה לשחק ובנוסף משתנים גלובלים המורים לי על בחירת אופציית המשחק.

**Ai**- מחלקה אשר אחראית על "הבינה המלאכותית" כלומר אחראית על בחירות המחשב

**GameLogic** – מחלקה ששם נמצאות הפונקציות אשר אחראיות על משחק תקין (פונקציה שבודקת אם הצופן הנבחר תקין (ארבע צבעים אשר שונים זה מזה) , פונקציה אשר מזהה את הפידבק שהניחוש צריך לקבל)

**אלגוריתם ראשי-AI**

לאלגוריתם זה יש שני חלקים:

1. מציאת ארבעת הצבעים הנכונים
2. מציאת הסדר של ארבעת הצבעים הנכונים

האלגוריתם פכםומועל על ידי פונקציות אשר אחראיות על כל מצב:

1. קבלת פידבק של שני דגלים(חלק ראשון)
2. קבלת פידבק של שלושה דגלים(חלק ראשון)
3. קבלת פידבק של ארבע דגלים לבנים(חלק שני)
4. קבלת פידבק של דגל אחד שחור(חלק שני)
5. קבלת פידבק של שני דגלים שחורים(חלק שני)

שתי הפונקציות של החלק הראשון מקבלות את הצופן הראשוני (הניחוש הראשון)  
שלושת הפונקציות של החלק השני מקבלות את ארבעת הצבעים הנכונים

משתנים אשר נמצאים בפונקציות:

Found – array list המכיל את הצבעים שבטוח נמצאים

NotIn – array list המכיל את הצבעים שבטוח לא נמצאים

Temp – מערך של צבעים שאותו משנים כדי למצוא את הצופן הנכון

flagCount – כמות הדגלים שהתקבלו

**מדריך למשתמש:**

למשחק שיצרתי קיימים כמה אפשרויות משחק:

שחקו אחד:

א. שחקן אחד בוחר את הצופן והשחקן השני צריך לגלות את הצופן

ב. המחשב בוחר את הצופן והשחן צריך לגלות

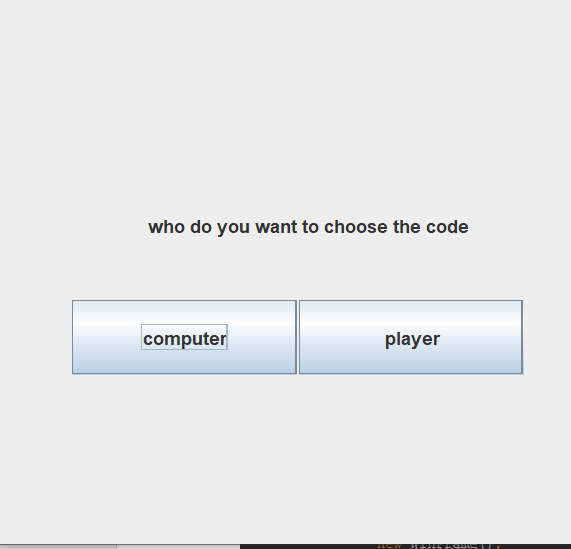
שני שחקנים:

1. שחקן מול שחקן: תחילה שני השחקנים יבחרו צופן שכל שחקן צריך לפצח ולאחר מכן ינסו כל אחד לגלות את צופן היריב בשיטת תור תור
2. שחקן מול מחשב: תחילה השחקו יבחר צופן למחשב ולאחר מכן המחשב ינסה לגלות את צופן ולאחר כל תורות המחשב השחקן יצטרך לגלות את הצופן שהמחשב בחר.

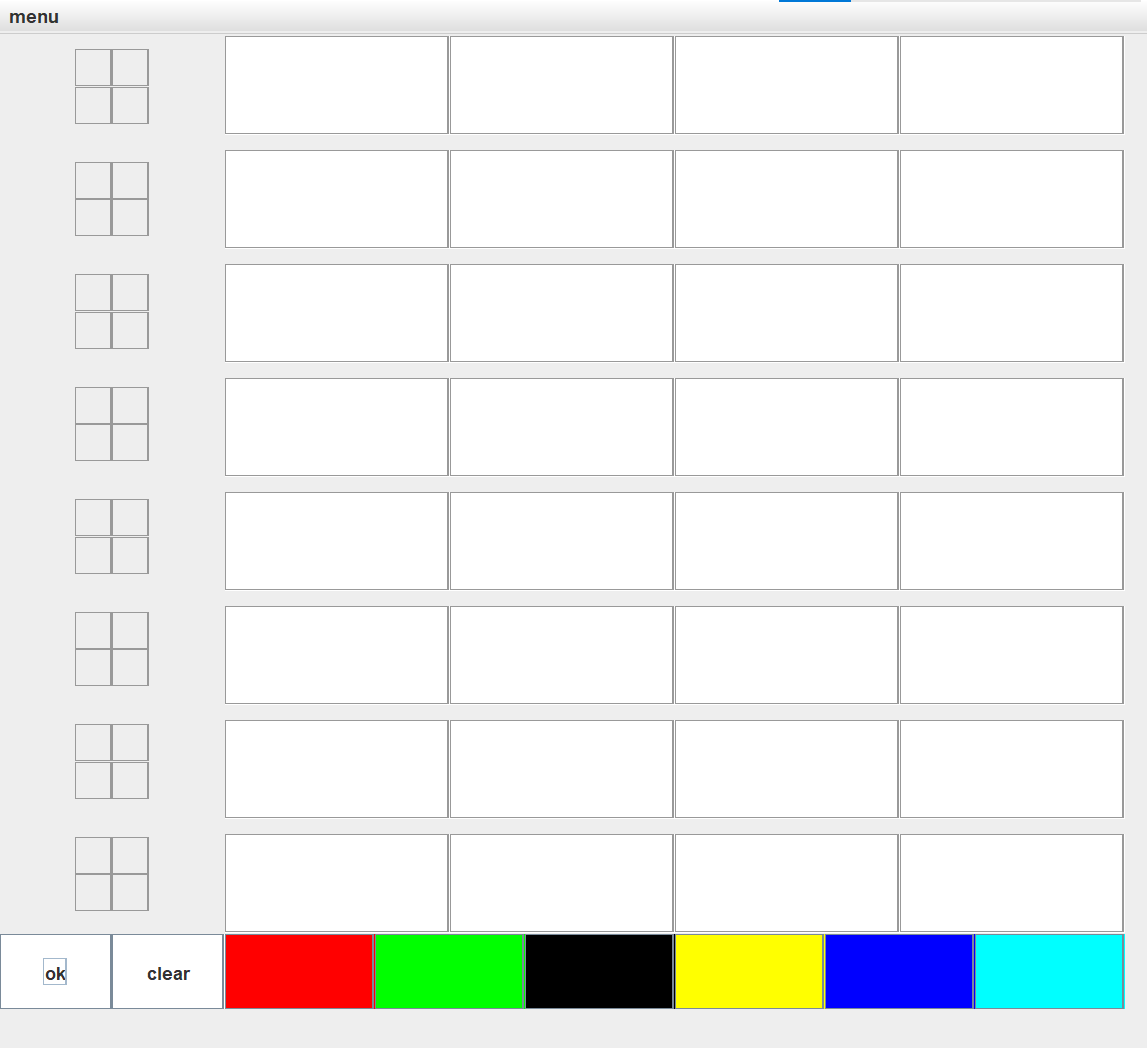
כדי לבחור את צורת המשחק הרצויה במסך הבית יהיו שתי אפשרויות שחקן אחד ושני שחקנים:



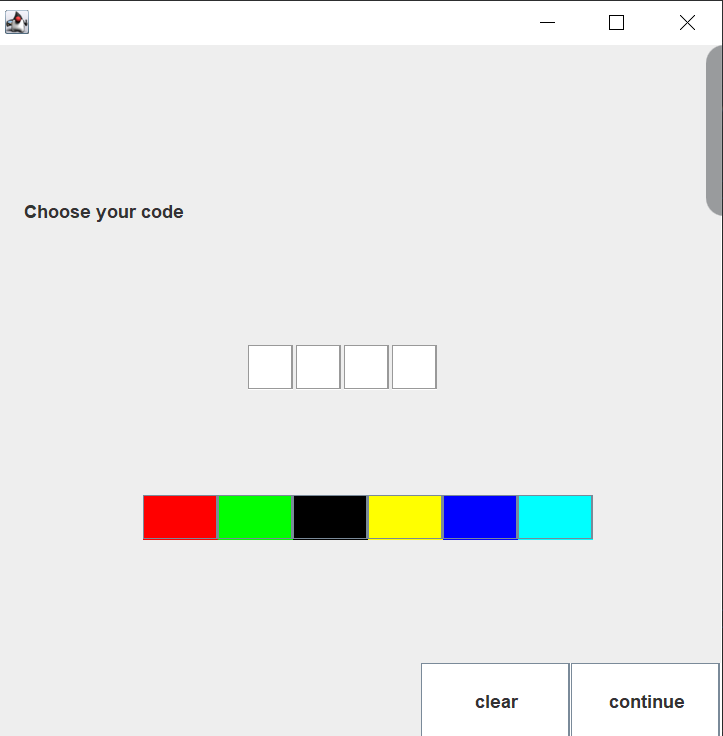
במידה ובחר שחקן אחד ישאל השחקן אם הוא רוצה לבחור את הצופן ידנית או שהמחשב יבחר את הצופן:



במידה ויבחר השחקן computer ישלח השחקן ללוח המשחק ויתחיל לשחק:

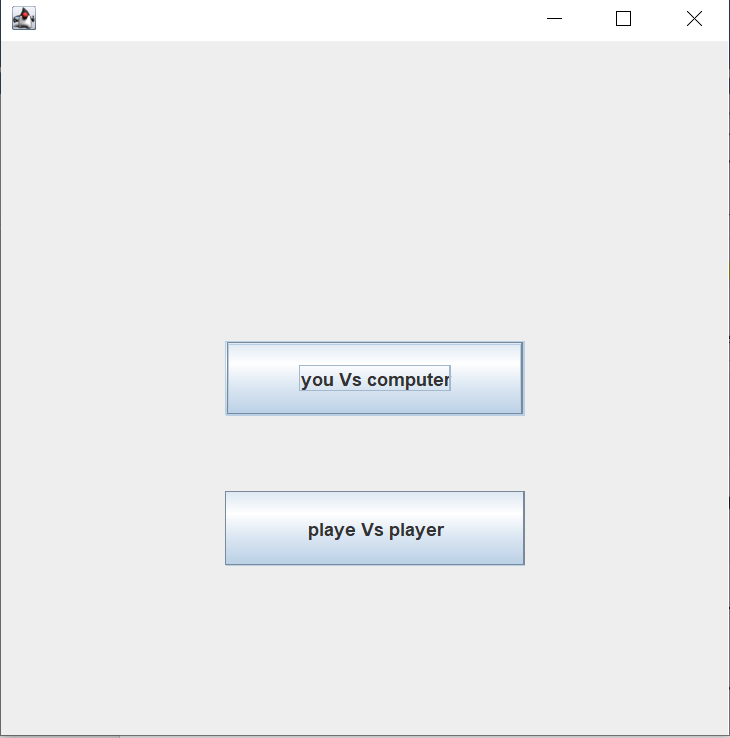


במידה ויבחר player ישלח השחקן למסך הבחירה ושם יצטרך לבחור את הצופן:

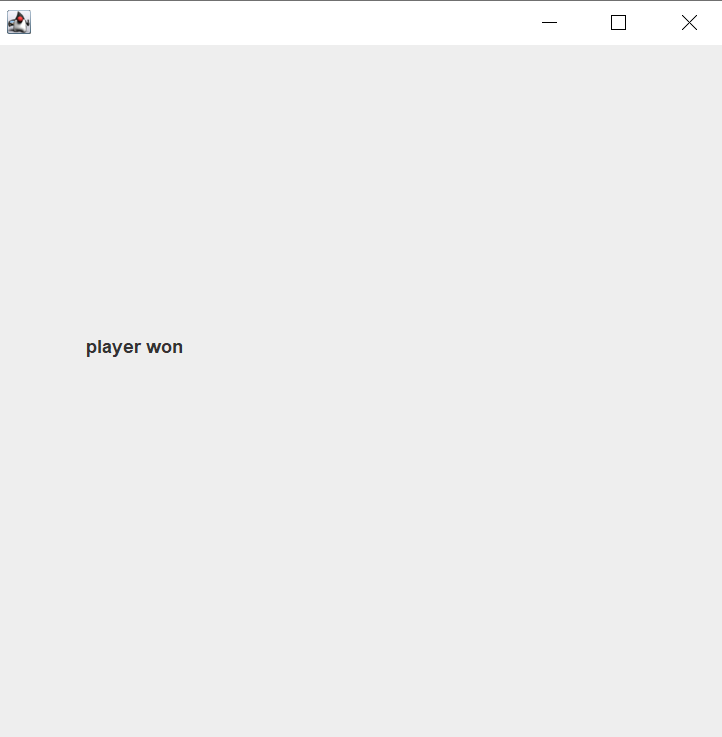


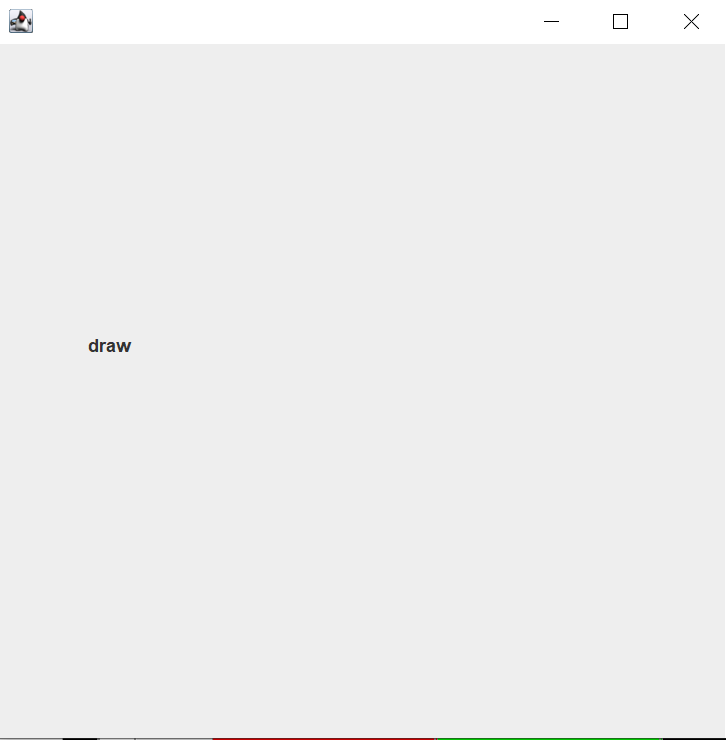
לאחר שיבחר את הצופן הרצוי וילחץ continue ישלח השחקן ללוח המשחק.

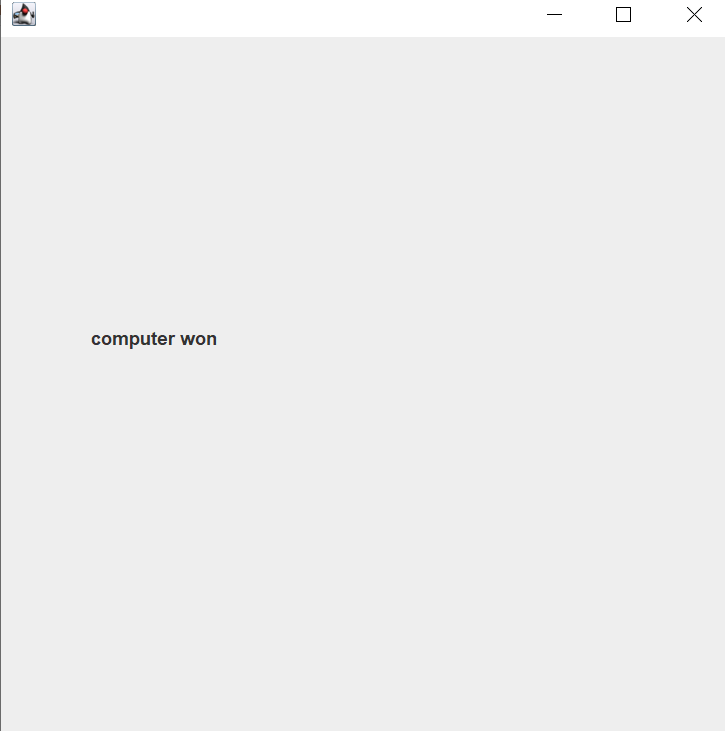
במידה והשחקן יבחר את האופציה של שני שחקנים יישלח השחקן לבחור אם הוא רוצה לשחק שחקן נגד שחק או מחשב נגד שחקן:



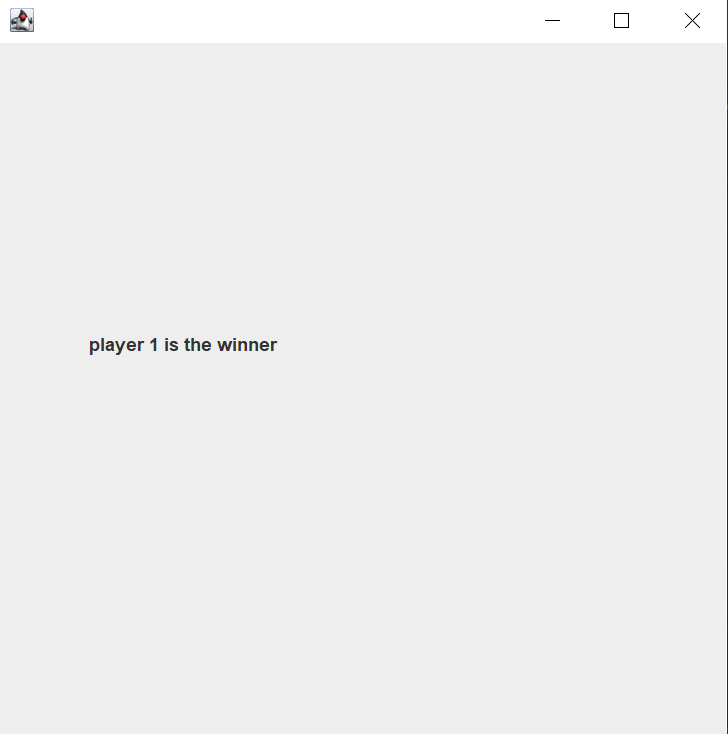
במידה ובוחר שחקן נגד שחקן ישלח השחקן ללוח הבחירה ולאחר מכן יוצר לו ניסיון המחשב לפיצוח הקוד בשלב זה על השחקן ללחוץ על כפתור הok ואז ישלח השחקן ללוח משחק ריק לעשות את ניסיונותיו ובסוף יוצג לשחקן את התוצאה שהתקבלה (שחקן ניצח/ מחשב ניצח/ תיקו)

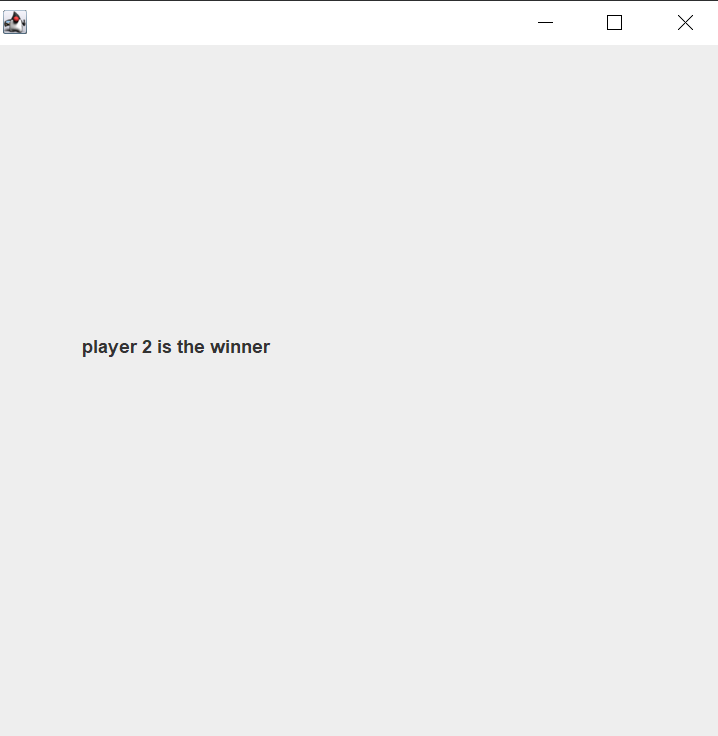
 שחקן ניצח

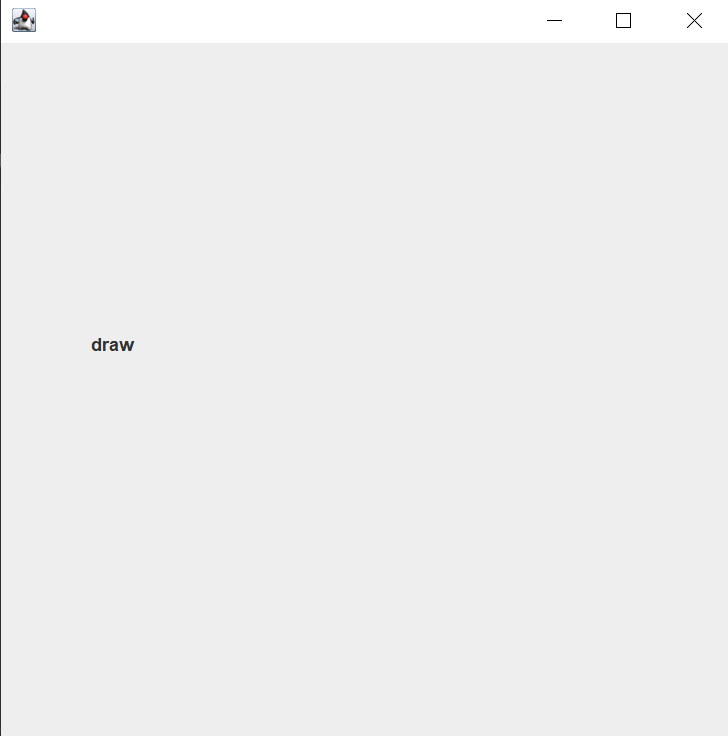
תיקו

 מחשב ניצח

במידה ויבחר השחקן אופציה של שחקן נגד שחקן יופיע לשחקן שתי מסכי בחירה אחד אחרי השני ואז ישלח ללוח המשחק ואחרי כל לחיצת כפתור אוקיי המסך יעבור למסך של השחקן השני ושם יעשה את התור שלו כך עד סוף והמשחק ופה גם כן בסוף המשחק יופיע לשחקן את תוצאת המשחק:  
שחקן 1 ניצח/שחקן 2 ניצח/ תיקו

 שחקן 1 ניצח

 שחקן 2 ניצח

 תיקו

**קוד הפרויקט:**

בעקבות המשך עבודה על הפרוייקט לא אצרף את כל הקוד אך כן אצרף את הקודים המעניינים:

**getResult – פונקציה אשר אחראית על קבלת הפידבק:**

public static void getResult (Color [] computer , Color [] player , int index , int position){  
 Color tsofen []= new Color[4];  
 Color code [] = new Color[4];  
 for(int i=0;i<4;i++){  
 tsofen[i]=computer[i];  
 code[i]=player[i];  
 }  
 int blacks=0;  
 *blackFlagCount*=0;  
 *flagCount*=0;  
 for(int i=0;i<4;i++){  
 if(computer!=null)  
 if(computer[i].equals(player[i])){  
 *flagCount*++;  
 *blackFlagCount*++;  
 ((JPanel) CheckBoxArray.*labels*[index].getComponent(0)).getComponent(position).setBackground(Color.*BLACK*);  
 computer[i]=null;  
 player[i]=null;  
 position++;  
 blacks++;  
 }  
 }  
 if(blacks==4 && !Homepage.*isComputerVsPlayer*){  
 if(Homepage.*isPlayerVsPlayer*){  
 if(PlayerVsPlayer.*turn*%2==0){  
 if(*isWon*)  
 WinFrame.*massage*.setText("draw");  
 else  
 WinFrame.*massage*.setText("player 2 is the winner");  
 PlayerVsPlayer.*p1*.dispose();  
 PlayerVsPlayer.*p2*.dispose();  
 *isFinished*=true;  
 new WinFrame();  
 }else{  
 *isWon*=true;  
 }  
 }else {  
 WinFrame.*massage*.setText("good job you manage to solve the code");  
 WhoChooseFrame.*frame*.dispose();  
 new WinFrame();  
 }  
 return;  
 }else if(*isWon*){  
 WinFrame.*massage*.setText("player 1 is the winner");  
 PlayerVsPlayer.*p1*.dispose();  
 PlayerVsPlayer.*p2*.dispose();  
 *isFinished*=true;  
 new WinFrame();  
 }  
 for(int i=0;i<4;i++){  
 if(player[i]!=null)  
 for(int j=0;j<4;j++){  
 if(player[i].equals(computer[j])) {  
 *flagCount*++;  
 computer[j]=null;  
 ((JPanel) CheckBoxArray.*labels*[index].getComponent(0)).getComponent(position).setBackground(Color.*white*);  
 position++;  
 break;  
 }  
 }  
 }  
 for(int i=0;i<4;i++){  
 computer[i]=tsofen[i];  
 player[i]=code[i];  
 }  
}

**העקת הנתונים מהלוח-**

if(PlayerVsPlayer.*turn*%2!=0){  
 for (int r = 0; r < 8; r++) {  
 for (int c = 0; c < 4; c++) {  
 PlayerVsPlayer.*player1Board*[r][c]=(*board*.*buttons*[r][c].getBackground());  
 }  
 Color result [] = new Color[4];  
 for(int i=0;i<4;i++){  
 result[i]=((JPanel)checks.*labels*[r].getComponent(0)).getComponent(i).getBackground();  
 }  
 PlayerVsPlayer.*player1Result*[r]=new Color[]{result[0],result[1],result[2],result[3]};  
 }  
 PlayerVsPlayer.*p1Index*=index;  
 PlayerVsPlayer.*p1Row*=row;  
}if(PlayerVsPlayer.*turn*%2==0){  
 for (int r = 0; r < 8; r++) {  
 for (int c = 0; c < 4; c++) {  
 PlayerVsPlayer.*player2Board*[r][c]=(*board*.*buttons*[r][c].getBackground());  
 }  
 Color result [] = new Color[4];  
 for(int i=0;i<4;i++){  
 result[i]=((JPanel)checks.*labels*[r].getComponent(0)).getComponent(i).getBackground();  
 }  
 PlayerVsPlayer.*player2Result*[r]=new Color[]{result[0],result[1],result[2],result[3]};  
 }  
 PlayerVsPlayer.*p2Index*=index;  
 PlayerVsPlayer.*p2Row*=row;  
}  
אלPlayerVsPlayer.*switchingFrame*();

**אלגוריתם הAI -** public static void inCaseOfTwo(Color tsofen []) {  
 int flagCount;  
 ArrayList<Color> found = new ArrayList<>();  
 Color temp [] = new Color[4];  
 found.add(Color.*RED*);  
 found.add(Color.*GREEN*);  
 found.add(Color.*YELLOW*);  
 found.add(Color.*BLUE*);  
 found.add(Color.*BLACK*);  
 found.add(Color.*CYAN*);  
 for (int j = 0; j < 4; j++) {  
 found.remove(tsofen[j]);  
 }  
 for (int i = 0; i < 4; i++) {  
 temp[i] = tsofen[i];  
 }  
 temp[0] = tsofen[0];  
 temp[1] = tsofen[1];  
 temp[2] = found.get(0);  
 temp[3] = found.get(1);  
// temp = (a,b,e,f)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
//found four colors(a, b, e, f)  
 found.add(tsofen[0]);  
 found.add(tsofen[1]);  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else if (flagCount == 2) { //(a,b,e,f)=2 flags  
 //(c, d)=2 flags  
 found.clear();  
 found.add(Color.*RED*);  
 found.add(Color.*GREEN*);  
 found.add(Color.*YELLOW*);  
 found.add(Color.*BLUE*);  
 found.add(Color.*BLACK*);  
 found.add(Color.*CYAN*);  
 for (int j = 0; j < 4; j++) {  
 found.remove(temp[j]);  
 }  
 temp[0]=found.get(0);  
 temp[1]=found.get(1);  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
 } else {  
 temp[1] = tsofen[2]; //temp=(a,c,e,f)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
//found four colors(a, c, e, f)  
 found.add(tsofen[0]);  
 found.add(tsofen[2]);  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else if (flagCount == 2) { //(a,c,e,f)=2 flags  
 //(b, d)=2 flags  
 temp[0]=tsofen[1];  
 temp[1]=tsofen[3];  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
 } else {  
 temp[1] = tsofen[3];  
//temp=(a,d,e,f)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if(flagCount==2){  
 *inCaseOfTwo*(temp);  
 }else if(flagCount==3){  
 *inCaseOfThree*(temp);  
 }else if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
 }  
 }  
  
 }  
 public static void inCaseOfThree( Color tsofen []) {  
 int flagCount;  
 ArrayList<Color> found = new ArrayList<>();  
 ArrayList<Color> notIn = new ArrayList<>();  
 Color temp [] = new Color[4];  
 notIn.add(Color.*red*);  
 notIn.add(Color.*green*);  
 notIn.add(Color.*yellow*);  
 notIn.add(Color.*blue*);  
 notIn.add(Color.*black*);  
 notIn.add(Color.*cyan*);  
  
 for (int j = 0; j < 4; j++) {  
 notIn.remove(tsofen[j]);  
//notIn =(e,f)  
 }  
 for (int i = 0; i < 4; i++) {  
 temp[i] = tsofen[i];  
 }  
 temp[3] = notIn.get(0); //temp=(a,b,c,e)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(tsofen[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else if (flagCount == 3) {  
 temp[3] = notIn.get(1); //temp=(a,b,c,f)  
  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else { //(a,b,c)=2 flags  
 // (e,d)=2 flags  
 for (int i = 0; i < 4; i++) {  
 temp[i] = tsofen[i];  
 }  
 found.add(tsofen[3]); // d  
 found.add(notIn.get(0)); // e  
 temp[2] = tsofen[3];  
 temp[3] = notIn.get(0); //temp=(a,b,d,e)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
 else if (flagCount==2){  
 *inCaseOfTwo*(temp);  
 }  
 else {  
 temp[1] = tsofen[2]; //temp=(a,c,d,e)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else { //temp=(b,c,d,e)  
 found.clear();  
 found.add(tsofen[1]);  
 found.add(tsofen[2]);  
 found.add(temp[2]);  
 found.add(temp[3]);  
 temp[0] = tsofen[1];  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
 }  
 }  
 }  
 } else { //(a,b,c)=2 flags  
 //(d,f)=2 flags  
 for (int i = 0; i < 4; i++) {  
 temp[i] = tsofen[i];  
 }  
 temp[2] = tsofen[3];  
 temp[3] = notIn.get(1); //temp=(a,b,d,f)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else {  
 temp[1] = tsofen[2]; //temp=(a,c,d,f)  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 } else { //temp=(b,c,d,f)  
 found.add(tsofen[1]);  
 found.add(tsofen[2]);  
 temp[0] = tsofen[1];  
 temp[1] = tsofen[2];  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*flagCount*;  
 if(flagCount==2){  
 *inCaseOfTwo*(temp);  
 }else if(flagCount==3){  
 *inCaseOfThree*(temp);  
 }else if (flagCount == 4) {  
 found.clear();  
 for (int j = 0; j < 4; j++) {  
 found.add(temp[j]);  
 }  
 if (GameLogic.*blackFlagCount* == 0) {  
 *inCaseOfFourW*(temp);  
 } else if (GameLogic.*blackFlagCount* == 1) {  
 *inCaseOfOneB*(temp);  
 } else if (GameLogic.*blackFlagCount* == 2) {  
 *inCaseOfTwoB*(temp);  
 }  
 }  
  
 }  
 }  
 }  
  
 }  
 public static void inCaseOfFourW(Color tsofen[]) {  
 int flagCount=0;  
 Color temp[] = new Color[4];  
 ArrayList<Color> notIn = new ArrayList<>();  
 notIn.add(Color.*red*);  
 notIn.add(Color.*green*);  
 notIn.add(Color.*yellow*);  
 notIn.add(Color.*blue*);  
 notIn.add(Color.*black*);  
 notIn.add(Color.*cyan*);  
 for (int j = 0; j < 4; j++) {  
 notIn.remove(tsofen[j]);  
//found = (e,f)  
 }  
 Color a = tsofen[0];  
 Color b = tsofen[1];  
 Color c = tsofen[2];  
 Color d = tsofen[3];  
 Color e = notIn.remove(0), f = notIn.remove(0);  
 temp[0]=e;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=f; //(e,b,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==1) { //found a - (, ,a, )  
 temp[0]=b;  
 temp[1]=e;  
 temp[2]=a;  
 temp[3]=f; //(b,e,a,f)  
 if(ComputerVsPlayer.*computer*.getRow()>-1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) { //found a,b -(b, ,a, )  
 temp[0]=b;  
 temp[1]=c;  
 temp[2]=a;  
 temp[3]=d; //(b,c,a,d)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount!=4) {  
 temp[0]=b;  
 temp[1]=d;  
 temp[2]=a;  
 temp[3]=c; //(b,d,a,c)  
 if(ComputerVsPlayer.*computer*.getRow()>-1 ){  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
//found a,b - ( , ,a,b)  
 else { //(e,f,a,b)  
 temp[0]=c;  
 temp[1]=d;  
 temp[2]=a;  
 temp[3]=b; //(c,d,a,b)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount!=4) {  
 temp[0]=d;  
 temp[1]=c;  
 temp[2]=a;  
 temp[3]=b; //(d,c,a,b)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 else {  
 temp[0]=e;  
 temp[1]=b;  
 temp[2]=f;  
 temp[3]=a; //(e,b,f,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==1) { //found a - (, , ,a)  
 temp[0]=b;  
 temp[1]=e;  
 temp[2]=f;  
 temp[3]=a; //(b,e,f,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
//found a,b - (b, , ,a)  
 temp[0]=b;  
 temp[1]=c;  
 temp[2]=d;  
 temp[3]=a; //(b,c,d,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount!=4) {  
 temp[0]=b;  
 temp[1]=d;  
 temp[2]=c;  
 temp[3]=a;  
//(b,d,c,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 else {  
//found a,b - ( , ,b,a)  
 temp[0]=c;  
 temp[1]=d;  
 temp[2]=b;  
 temp[3]=a; //(c,d,b,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount!=4) {  
 temp[0]=d;  
 temp[1]=c;  
 temp[2]=b;  
 temp[3]=a;  
//(c,d,b,a)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 else {  
//found a - ( ,a, , )  
 temp[0]=b;  
 temp[1]=a;  
 temp[2]=e;  
 temp[3]=f; //(b,a,e,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
//found a,b - (b,a, , )  
 temp[0]=b;  
 temp[1]=a;  
 temp[2]=d;  
 temp[3]=c; //(b,a,d,c)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=e;  
 temp[1]=a;  
 temp[2]=b;  
 temp[3]=f; //(e,a,b,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
//found a,b - ( ,a,b, )  
 temp[0]=d;  
 temp[1]=a;  
 temp[2]=b;  
 temp[3]=c;  
//(d,a,b,c)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 else {  
//found a,b - ( ,a, ,b)  
 temp[0]=c;  
 temp[1]=a;  
 temp[2]=d;  
 temp[3]=b;  
//(c,a,d,b)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
 public static void inCaseOfTwoB(Color tsofen[]) {  
 int flagCount=0;  
 Color temp[] = new Color[4];  
 ArrayList<Color> notIn = new ArrayList<>();  
 notIn.add(Color.*red*);  
 notIn.add(Color.*green*);  
 notIn.add(Color.*yellow*);  
 notIn.add(Color.*blue*);  
 notIn.add(Color.*black*);  
 notIn.add(Color.*cyan*);  
 for (int j = 0; j < 4; j++) {  
 notIn.remove(tsofen[j]);  
//found = (e,f)  
 }  
 Color a = tsofen[0];  
 Color b = tsofen[1];  
 Color c = tsofen[2];  
 Color d = tsofen[3];  
 Color e = notIn.remove(0), f = notIn.remove(0);  
 temp[0]=a;  
 temp[1]=b;  
 temp[2]=e;  
 temp[3]=f; //(a,b,e,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=a;  
 temp[1]=b;  
 temp[2]=d;  
 temp[3]=c;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
  
 else if(flagCount==0) {  
 temp[0]=b;  
 temp[1]=a;  
 temp[2]=c;  
 temp[3]=d; //(b,e,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=a;  
 temp[1]=e;  
 temp[2]=c;  
 temp[3]=f; //(a,e,c,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=a;  
 temp[1]=d;  
 temp[2]=c;  
 temp[3]=b; //(e,b,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else if(flagCount==0) {  
 temp[0]=c;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=d; //(e,b,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=a;  
 temp[1]=e;  
 temp[2]=f;  
 temp[3]=d; //(a,e,f,d)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=a;  
 temp[1]=c;  
 temp[2]=b;  
 temp[3]=d; //(a,c,b,d)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=d;  
 temp[1]=b;  
 temp[2]=c;  
 temp[3]=a; //(e,b,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 }  
 public static void inCaseOfOneB(Color tsofen[]) {  
 int flagCount=0;  
 Color temp[] = new Color[4];  
 ArrayList<Color> notIn = new ArrayList<>();  
 notIn.add(Color.*red*);  
 notIn.add(Color.*green*);  
 notIn.add(Color.*yellow*);  
 notIn.add(Color.*blue*);  
 notIn.add(Color.*black*);  
 notIn.add(Color.*cyan*);  
 for (int j = 0; j < 4; j++) {  
 notIn.remove(tsofen[j]);  
//found = (e,f)  
 }  
 Color a = tsofen[0];  
 Color b = tsofen[1];  
 Color c = tsofen[2];  
 Color d = tsofen[3];  
 Color e = notIn.remove(0), f = notIn.remove(0);  
 temp[0]=a;  
 temp[1]=b;  
 temp[2]=e;  
 temp[3]=f; //(e,b,a,f)  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==1) {  
 temp[0]=e;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=f;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=d;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=c;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else if(flagCount==1) {  
 temp[0]=c;  
 temp[1]=b;  
 temp[2]=d;  
 temp[3]=a;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=a;  
 temp[1]=e;  
 temp[2]=b;  
 temp[3]=f;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }if(flagCount==2) {  
 temp[0]=a;  
 temp[1]=d;  
 temp[2]=b;  
 temp[3]=c;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=a;  
 temp[1]=c;  
 temp[2]=d;  
 temp[3]=b;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 else {  
 temp[0]=e;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=f;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=c;  
 temp[1]=b;  
 temp[2]=a;  
 temp[3]=d;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else if(flagCount==1) {  
 temp[0]=b;  
 temp[1]=c;  
 temp[2]=a;  
 temp[3]=d;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=e;  
 temp[1]=b;  
 temp[2]=f;  
 temp[3]=a;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount==2) {  
 temp[0]=d;  
 temp[1]=b;  
 temp[2]=c;  
 temp[3]=a;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else if(flagCount==1) {  
 temp[0]=b;  
 temp[1]=d;  
 temp[2]=c;  
 temp[3]=a;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 else {  
 temp[0]=c;  
 temp[1]=a;  
 temp[2]=b;  
 temp[3]=d;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 if(flagCount!=4) {  
 temp[0]=d;  
 temp[1]=a;  
 temp[2]=c;  
 temp[3]=b;  
 if(ComputerVsPlayer.*computer*.getRow() > -1) {  
 BoardFrame.*setLine*(temp, ComputerVsPlayer.*computer*.getRow());  
 GameLogic.*getResult*(Homepage.*code1*, temp, ComputerVsPlayer.*computer*.getIndex(), ComputerVsPlayer.*computer*.getPosition());  
 ComputerVsPlayer.*computer*.*col* = 0;  
 ComputerVsPlayer.*computer*.position = 0;  
 ComputerVsPlayer.*computer*.index--;  
 ComputerVsPlayer.*computer*.row--;  
 flagCount = GameLogic.*blackFlagCount*;  
 }  
 }  
 }  
 }  
 }  
 }

**רפלקציה:**

העבודה על הפרוייקט הייתה מסע ארוך מעניין וכיף במיוחד עבורי. במהלך העבודה הייתי צריך לפתור המון בעיות, לבצע ניסיונות ובדיקות ופשוט לעבוד בצורה חזרתית ומעמיקה כדי להגשים את החזון שתכננתי. אני מרגיש שקיבלתי מהעבודה על הפרוייקט הזה המון כלים, רובם בתחום התכנות כמו הרחבת הידע שלי, כתיבה מסודרת תכנון ובניה ועוד... בנוסף, חלקם קשורים דווקא לתכונות שרכשתי לעצמי כמו היכולת לשבת הרבה זמן רצוף, לעבוד ולא להתייאש עד שאני מגיע למטרה שהצבתי לעצמי באותו יום, לקרוא טקסטים ארוכים ולהסיק מהם מה אני צריך לעשות, לסרטט על דף את מה שאני מדמיין ולסלול לשם את הדרך. אחד הדברים שהכי נהנתי ממנו היה תכנון המערכת של הפתרון הממוחשב. בעבודה על אלגוריתם זה שילבתי המון תחומים מתמטיים מעניינים כמו תורת הקבוצות, סדרות וסטטיסטיקה. אחד הקשיים העיקריים שעמדתי מולם היה הקושי בעיצוב הגרפי של הפרוייקט. אני מרגיש שלא קיבלתי מספיק כלים בכל מה שקשור לעיצוב הפרוייקט, ורוב המידע שמצאתי באינטנרט לא ממש עזר לי ובסופו של דבר השתמשתי בטכניקה פשוטה )grid )לעיצוב ומיקום כל הממשק הגרפי. לפי דעתי צריך לשים דגש גדול יותר על העיצוב כי בסופו של יום מי שנהנה מהתוכנה הוא המשתמש, ולכן התוכנה צריכה להיות כמה שיותר נגישה לו. אילו הייתי מתחיל את הפרוייקט היום, הייתי משקיע יותר זמן בעבודה על הממשק הגרפי בהתחלה ולא הייתי מתפשר על ממשק גרפי בינוני רק כדי להתחיל לעבוד על התוכנה עצמה )כפי שקרה הפעם(. בנוסף, אני חושב שכדי להגיע לתוצאה יותר טובה הייתי צריך גם להיות פחות ביישן, לשאול את המורים יותר שאלות, להיעזר יותר בתלמידים האחרים, חלק מהקושי שלי בפרוייקט היה חוסר בידע שהיה לי בכל הקשור לעיצוב וניסיתי להשלימו עם האינטרנט ואף הצלחתי חלקית אך התוצאה אינה מושלמת ויכול להיות שאם הייתי נעזר במורים ובחברים הייתי יכול להגיע לתוצאה יותר טובה.

**ביבליוגרפיה:**

stackoverflow.com  
[www.youtube.com](http://www.youtube.com)

www.javapoint.com  
www.geeksforgeeks.org