# 

# **עבודת גמר**

# **לקבלת תואר טכנאי**

# **הנדסת תוכנה**

**נושא הפרויקט:** Texas Holdem Poker

**שפת תכנות:** Java

**סביבת העבודה** ‎Eclipse

**שם המגיש:** נאור אנחייסי

**ת"ז:** 322292855

**מכללה:** אורטחולון

**כיתה:** י"ג 2, מגמת הנדסת תוכנה

**מנחה:** ינאי סנד

**מאי 2019 תשע"ט**

**תוכן עניינים**

**הצעת פרויקט.........................................................................................................** 4-3

**מבוא .............................................................................................................** 6

**מטרה .............................................................................................** 6

**תיאור המערכת.................................................................................** 6

**מבנה המסמך ..................................................................................** 6

**ביבליוגרפיה.....................................................................................** 6

**מפרטי תוכנה .............................................................................................** 11-7

**תיאור כללי ....................................................................................** 7-8

**ממשק אדם מכונה MMI ................................................................** 9

**יכולות פונקציונאליות.......................................................................** 9-10

**תיאור אלגוריתמים .........................................................................** 10-11

**תכנון.........................................................................................................** 13-23

**פירוט המודולים+רשימת נתונים.....................................................** 13-22

**מבנה נתונים .................................................................................** 23

**מדריך למשתמש.........................................................................................**24-25

**הוראות התקנה ( Setup ) ...........................................................** 24

**הוראות הפעלה ............................................................................** 24

**הודעות מערכת ..........................................................................** 24-25

**5. נספח - קוד מקור ...........................................................................................** 26-73

# **הצעת נושא לפרויקטים לטכנאים במגמת הנדסת תוכנה**

**א. הנושא:** Texas hold’ em Poker

**ב. שם המנחה :** ינאי סנד

**ג. שם התלמיד :** נאור אנחייסי **ת.ז. : 322292855**

**ד. עבודה ביחיד**

**ה. אופי עבודת הפרויקט :** תכנון ובניית אב טיפוס

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

**ז. תיאור נושא המשחק :**

**המשחק מתנהל בגוף ראשון, במבט מעל שולחן הפוקר. בשולחן הפוקר יושבים כ – 6 שחקנים, המשתמש ועוד כחמישה שחקני מחשב (בוטים). השחקן הוא היושב בשולחן שקלפיו מורמים כלפי מעלה, ומטרתו היא לנצח במשחק באמצעות חוקי המשחק של** Texas hold’ em**.  
המשחק מתנהל ב- 4 סיבובים שונים :**

Preflop Round – סבב הימורים בו השחקנים מקבלים את קלפם (קלפי כיס), וטרם רואים את קלפי הקהילה (הקלפים המופנים כלפי מעלה במרכז השולחן). בסוף, קלף אחד "נשרף".

Flop Round - סיבוב בו 3 קלפי קהילה נפתחים. בסוף, קלף אחד נשרף מהחפיסה.

Turn Round – סיבוב בו קלף קהילה נוסף נפתח בפני המשתמשים. אחר כך, שורפים קלף אחד מהחפיסה.

River Round – בסבב זה נפתח קלף קהילה חמישי ואחרון, וברגע שהסכום הושווה בין המשתמשים, כל הקלפים של המשתמשים נפתחים.

המשתמש בעל היד החזקה ביותר הוא המנצח. במידה ולשני שחקנים, או יותר, אותה יד, השחקן עם הקלף הבא החזק יותר (נקרא קיקר – Kicker) מנצח. אם אין קיקר, אז הקופה תחולק באופן שווה בין השחקנים.

כל סיבוב נגמר ברגע שבו הכסף שהגיע מכל משתמש שווה.

לכל שחקן יש 3 אפשרויות : השווה (Call) – השוואת הסכום שהשחקן האחרון שם (או העבר (Check) – במידה וסכום הכסף שהשחקן האחרון שווה לסכום הכסף שהשחקן שם כבר, אין העלאה של כסף והמשחק ממשיך) , להעלות (Raise) – לקבוע סכום חדש שיהיה על כל השחקנים להשוות או לפרוש, או לפרוש (Fold) – לצאת מהיד הנוכחית.

כשכל השחקנים פרשו מלבד אחד, השחקן שנותר מנצח. מקרה זה יכול לקרות בכל סיבוב הימורים, וכאשר הוא קורה, המנצח אינו חושף את הקלפים שהיו ברשותו.

סיום משחק:  
- ניצחון: כאשר השחקן הוא בעל היד החזקה ביותר בסבב האחרון.

- הפסד: כאשר השחקן פורש, או כאשר אין לו את היד החזקה ביותר בסבב האחרון.

סוגי הניצחונות האפשריים (לפי הסדר: 1- הכי חזק, 10 - הכי חלש) :



**ח. פרוט הדרישות מהמערכת :**

המערכת תנהל את המשחק של השחקן ושל היריבים הממוחשבים.

המערכת תטען את קלפי הכיס של כל שחקן וקלפי הקהילה בתחילת כל שלב.

המערכת תבחר עבור הבוטים את הצעד האופטימלי הבא בהתאם לתנאים (קלפי הכיס של כל שחקן בהתאמה לקלפי הקהילה, סיכויים לנצח).

המערכת תכריז על הודעות ניצחון ועל סכום הכסף שזכה השחקן המנצח.

המערכת תבצע ערבוב של חפיסת הקלפים בכל יד מחדש.

**מפרט טכני :**

מעבד: מעבד Intel® Core™ i7 6700 CPU @ 3.40 GHz

זיכרון: RAM 16 GB

סביבת עבודה: WINDOWS 10 Pro

תוכנה: Java in Eclipse.

י. לוח זמנים לביצוע הפרויקט לפי שלבים:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **11/2018** | **12/2018** | **01/2019** | **02/2019** | **03/2019** | **04/2019** | **05/2019** |
| **הכנת הצעת הפרויקט** | **X** |  |  |  |  |  |  |
| **הגשת ההצעה** | **X** |  |  |  |  |  |  |
| **תכנון** |  | **X** |  |  |  |  |  |
| **גרפיקה** |  | **X** | **X** |  |  |  |  |
| **כתיבה** |  |  | **X** | **X** | **X** |  |  |
| **בדיקות** |  |  |  |  | **X** | **X** |  |
| **הפעלה והגשה** |  |  |  |  |  | **X** | **X** |

**דוגמא לחלוקת דרישות עבור משחק: Texas hold’ em Poker:**

Must have

- 3 אפשרויות לכל שחקן : פרישה, העלאה, והשוואה (או העברה). (5 שעות)

- העברת התור בין כל אחד מן השחקנים (2 שעה)

- Small Blind - לפני חלוקת הקלפים, שם סכום כסף קבוע (4 שעה)

- Big Blind – לפני חלוקת הקלפים, שם סכום כסף קבוע, הכפול מזה של ה- Small Blind (3 שעה)

- All In (3 שעות)

- כמות כסף קבועה בתחילת המשחק לכל שחקן (1 שעה)

- אפשרויות לשחקן לבחור את העלאת הכסף שלו (2 שעה)

- נעילת אפשרויות לשחקן (כמו שלא יוכל להעלות שסכום הכסף שלו קטן מההימור הנוכחי) (4 שעות)

- מסך פתיחה (6 שעות)

- סכום כסף מורווח (1 שעה)

- שולחן פוקר ברקע (1 שעה)

- טיימר לכל שחקן – זמן מוגבל לכל תור (4 שעות)

- כאשר נגמר לשחקן הכסף הוא יוצא מהמשחק (1 שעה)

- חישוב לגבי סוגי הניצחון של כל שחקן (6 שעות)

- אלגוריתם לפעולות שחקני המחשב (9 שעות)

Important

- sound (2 שעות)

- אנימציה של קלפים זזים מהדילר ללוח, כשפותחים קלפי קהילה. (2 שעות)

- ערבוב של החפיסה שלושה פעמים, לפני חלוקת הקלפים בהתחלה. (1 שעה)

- לפני סיבובים Flop, Round, River, "שורפים" קלף מהחפיסה. (1 שעה)

Nice to have

- הדרכה (1 שעה)

- מד הסתברות לניצחון ביד (3 שעות)

- אנימציה של ג'וטונים, שבסוף כל סיבוב נאספים למרכז השולחן והופכים למספר. (3 שעות)

- טיפ לדילר (1 שעה)

**1. מבוא:**

**1.1 מטרה:**

מטרת המשחק היא לנצח את כל שחקני המחשב שבשולחן הפוקר, עפ"י הכללים הידועים במשחק "טקסס הולדם".

**1.2 תיאור המערכת:**

המערכת מורכבת ממצב אחד עיקרי – שהוא שולחן הפוקר, לצד חמישה שחקני מחשב. לבחירת השחקן האפשרויות : העלאה אל סכום מסויים של כסף ((Raise, להעביר ((Check, להשוות ((Call, לפרוש ((Fold או להעלאות הכל ((All In.

**1.3 מבנה המסמך:**

הספר כולל ארבעה פרקים:

הפרק הראשון- מבוא למשחק עצמו: מטרת המשחק, תיאור כללי של המשחק, מבנה המסמך כולו ורשימת מקורות עזר בכתיבת הפרויקט.

הפרק השני- פירוט התוכנה עצמה. תיאור מפורט של המשחק, ממשק אדם-מכונה, יכולות המערכת, תיאור האלגוריתם ורשימת נתונים.

הפרק השלישי- פירוט תכנון המערכת. תיאור המודולים השונים ומבנה הנתונים.

הפרק הרביעי- הוראות הפעלה. פירוט הוראות התקנה והפעלה.

והפרק האחרון – מכיל את כל קוד המקור של הפרויקט

**1.4 רשימה ביבליוגרפית:**

1) <https://stackoverflow.com/>

2) <http://cowboyprogramming.com/2007/01/04/programming-poker-ai/>

3) <https://www.tutorialspoint.com/>

4) [https://www.geeksforgeeks.org/system-exit-in-java](https://www.geeksforgeeks.org/system-exit-in-java/)

5) <https://github.com/>

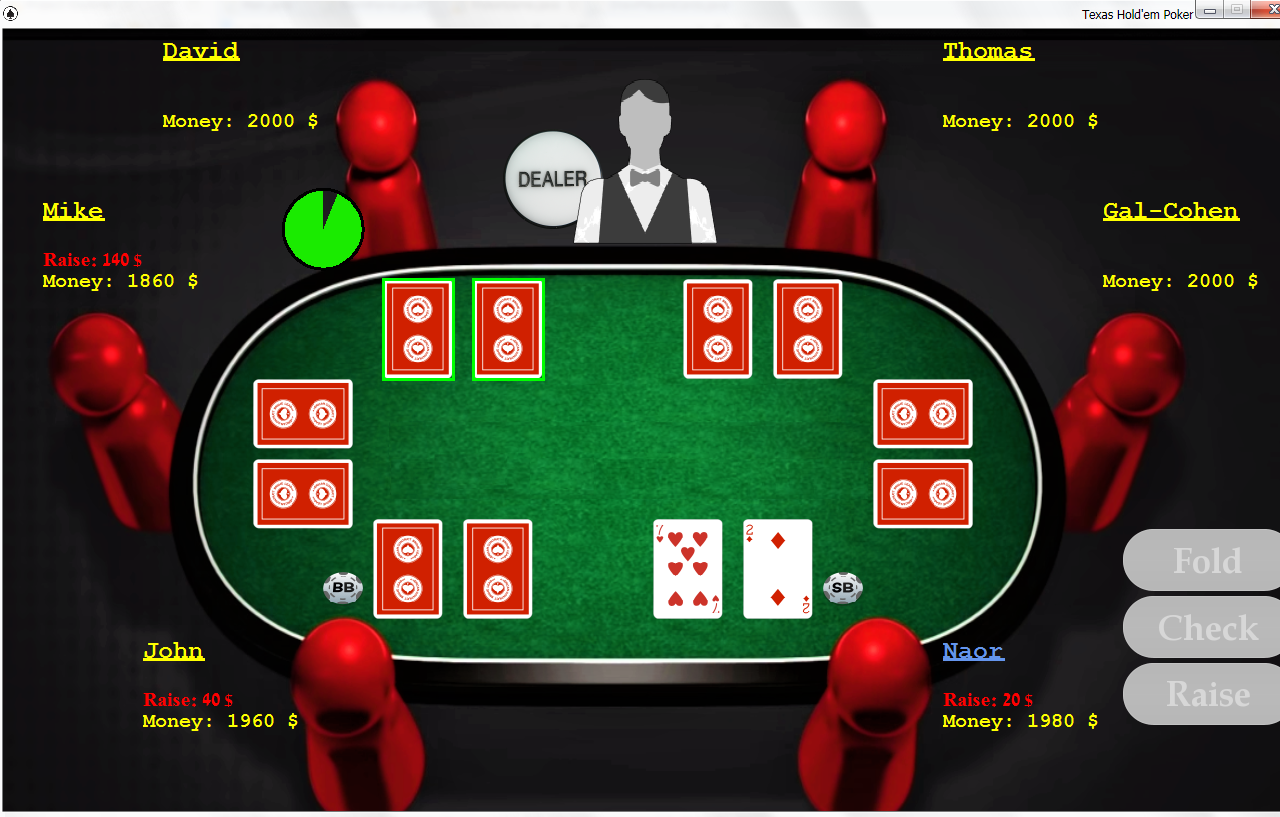
**2. מפרטי תוכנה:**

יישום זה פותח בסביבת העבודה :



**2.1 תיאור כללי:**

לאחר הכנסת שם ולחיצת כפתור ה- Play Now, יפתח חלון משחק הפוקר, בו יתחיל השחקן המסומל בתור הSmall Blind, יתחיל את המשחק בהעלאת סכום קבוע, לאחר השחקן המסומל בתור Big Blind יכניס סכום קבוע הכפול מקודמו, ואז המשחק יתחיל באופן רציף בין כל השחקנים, בכיוון השעון.



לשחקן האנושי (המשתמש) לא תהיה אפשרות לשחק עד שלא יגיע תורו.

לכל שחקן יש זמן מוגבל לתור, שנראה באמצעות "עוגת זמן" שמשנה את גזרתה כפונקציה של הזמן, וכן את צבעה. במקרה ושחקן לא ישחק בתורו בזמן, אם ההימור האחרון שלו שווה להימור הנוכחי, הוא יעביר (יעשה Check), ואחרת, יפרוש.

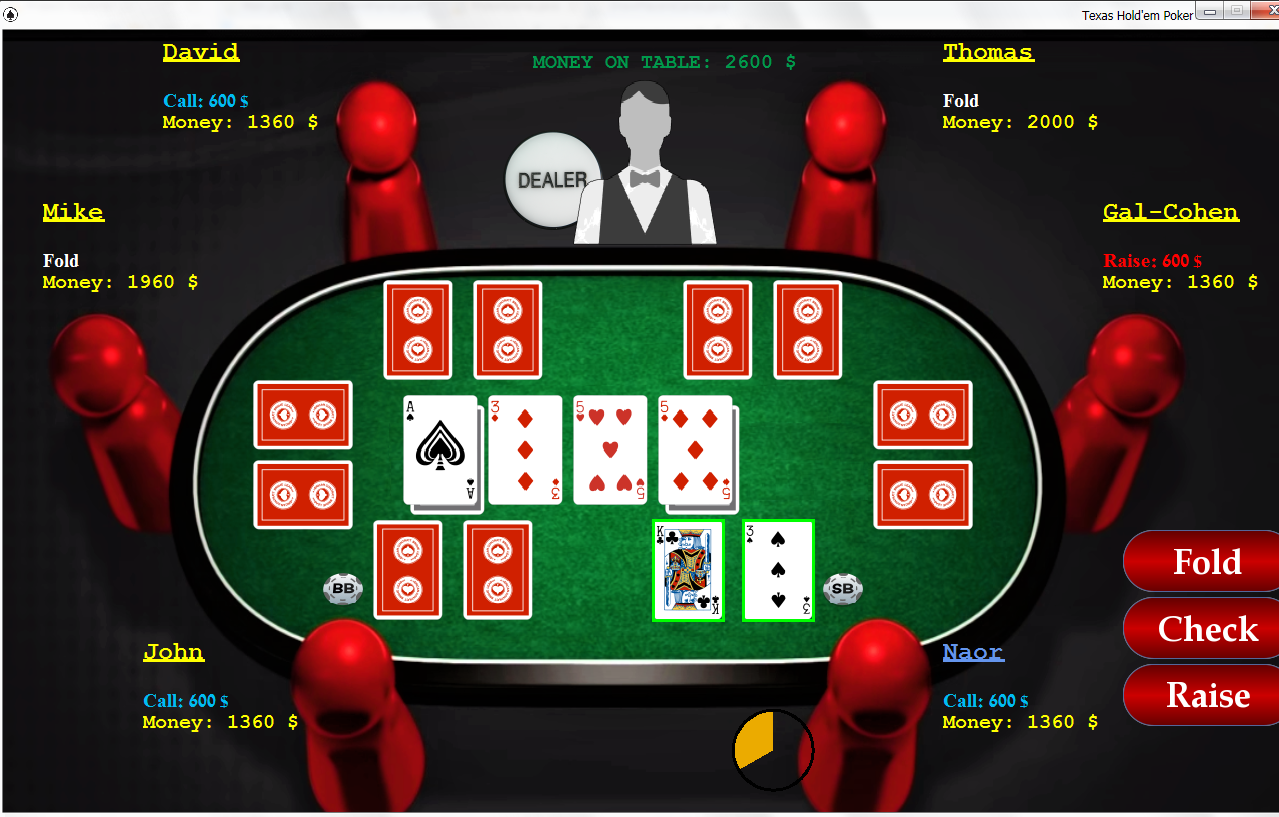
המשחק מתנהל לפי ידיים (Hands - מן סיבוב, שבו כל שחקן מקבל קלפים שונים והקלפי קהילה משתנים, וה- SB וBB - מתקדמים ב-1) ובסיבובים (Rounds) - כל יד מורכבת מ-3 סיבובים:

Flop

Turn

River

בכל סיבוב נפתחים קלפים נוספים בקלפי השולחן, ולפיה כל אחד מהשחקנים יכול להגדיר את סיכוייו לניצחון, וכך לבחור את האופציה העדיפה בשבילו.



**2.1.2 מצב של סיום משחק:**

המשחק יסתיים בהפסד כאשר המשתמש הפסיד את כל כספו.

המשחק יסתיים בניצחון כאשר המשתמש ינצח את כל השחקנים שנמצאים בשולחן (ירוויח את הכסף של כולם).

**2.1.3 אסטרטגית המשחק:**

אסטרטגיית המשחק היא לשער את חוזק היד ואת סיכוי השחקן לנצח, וגם תוך התחשבות במצבו הכספי וכמה הוא מוכן להשקיע. יש להתחשב בכך שמשום שזהו משחק הימורים שקשור באופן עיקרי למזל, שיתכן ושחקן יקבל במשך הרבה סיבובים קלפים חלשים יותר מיריבו. לכן, עליו לשחק בחוכמה וסבלנות, ולהשקיע בידיים (hand) שבהם הוא מסוגל לנצח.

סוגי הידיים קיימות בדף ההדרכה במשחק.

**2.2 ממשק אדם מכונה MMI:**



(1 Play Now - התחלת משחק חדש

2) How To Play? – פתיחת דף הוראות

3) סימן הסאונד – הדלקת וכיבוי הסאונד המשחק

4) Enter your nickname - על המשתמש להכניס את שמו במשחק בתיבת הטקסט בסמוך

**יכולות פונקציונאליות:**

המערכת מאפשרת משחק של שחקן אנושי מול שחקן ממוחשב.

המערכת לא מאפשרת לשחקן לעשות מהלכים שהוא לא יכול (כמו להעלות כאשר הימור הסבב גדול מסכום הכסף של השחקן).

המערכת מוציאה שחקנים שהפסידו מן המשחק.

המערכת שולטת בחילוק הקלפים בין השחקנים, ובחילוק הקלפים לערימה. (היא מערבבת את הקלפים בתחילת המשחק.)

המערכת מבצעת את חישוביי ההימור של כל שחקן, וכן מחשבת מיהו השחקן המנצח בסיום כל יד.

המערכת מאפסת או מעדכנת את השלבים בהתאם לפעולה כגון השחקן הפסיד או ניצח, או שהמשחק עוד נמשך.

המערכת מעדכנת את מצב השחקנים אחרי כל מהלך שלהם, ואת הכסף בקופה בסיום כל סיבוב.

**2.4 תיאור אלגוריתמים :**

במשחק זה קיים אלגוריתם שמחשב את סיכויי ההצלחה של שחקן, שמתחשב בכלל פרמטרים כגון: 2 הקלפים אותם הוא מחזיק, רשימת הקלפים הפתוחים ונראים לכולם על השולחן, מספר הסיבוב, מספר השחקנים ביד הנוכחית (הפוטנציאליים לניצחון), כמות ההימור הנוכחי, וסך הכסף שנמצא כרגע על השולחן.

האלגוריתם מחשב את חוזק היד בצורה הבאה:

**Create a pack of cards  
Set score = 0  
Remove the known cards (your hole cards, and any community cards)  
Repeat 1000 times (or more, depending on CPU resources and desired accuracy)  
Shuffle the remaining pack  
Deal your opponent's hole cards, and the remaining community cards  
Evaluate all hands, and see who has the best hands  
If you have the best hand then  
Add 1/(number of people with the same hand value) to your score (usually 1)  
End if  
end repeat  
Hand Strength = score/number of loops (10000 in this case).**

בעיקרון, מריצים כמות גדולה של "מיני משחקים" (10000 במקרה שלי) עבור כל שחקן שתורו הגיע. בכל מיני משחק, השחקן מקבל את הקלפים ששייכים לו, ובקלפים הפתוחים נמצאים הקלפים הידועים (אם ידוע), ומוציאים את כל הקלפים הידועים מהחפיסה. כל שאר הקלפים – הקלפים הסגורים וקלפי האנשים לא ידועים (רנדומליים).

המערכת מחשבת מיהו המנצח במשחק, ומוסיפה בעבור כל מיני משחק ניקוד לשחקן שזהו תורו. הניקוד המתווסף שווה ל:

אם השחקן הנוכחי נמצא ברשימת המנצחים (לעיתים יש מספר מנצחים), אזי מוסיפה ניקוד השווה ל – 1 חלקי גודל הרשימה.

אחרת, לא מוסיפה לניקוד.

לבסוף, מספר הניקוד של השחקן מחולק במספר המיני משחקים (10000), והתוצאה המתקבלת היא חוזק היד.

לאחר מכן, השחקן מחשב את הסיכוי רווח (התוחלת) שלו – כלומר, מה ירוויח מניצחון, בהתחשבות באם ינצח בכלל.

השחקן יחשוב אם בכלל כדאי לו להכנס בהתחשב בסיכוויו לזכיה, בהימור הסבב הנוכחי, ובכמות בכסף בקופה, ובכמות השחקנים שנותרו במשחק. דוגמה לחישוב תוחלת של כמה אפשרויות לשחקן לבצע:

**int** howMuchMoneyDoINeedToCall = (currentRoundBet - currentPlayer.getBet());

**int** howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls = sumMoneyOnTable + (40 \* numOfPlayerInHand);

**int** raise40 = 40 + currentRoundBet;

**int** raise100 = 100 + currentRoundBet;

**int** bigRaise = currentPlayer.getMoney() / 2 \* 20 / 20 + currentRoundBet;

**double** risk = 0.2;

// חישוב תוחלת של פרישה

**double** howGoodIsFold = 0;

// חישוב תוחלת של השוואה

**double** howGoodIsCall = -howMuchMoneyDoINeedToCall + (chances + risk) \* howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls;

// חישוב תוחלת של העלאה ב-40

**double** howGoodIsRaise40 = -howMuchMoneyDoINeedToCall - raise40 + (chances + risk) \* howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls;

// חישוב תוחלת של העלאה ב-100

**double** howGoodIsRaise100 = -howMuchMoneyDoINeedToCall - raise100 + (chances + risk) \* howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls;

// חישוב תוחלת של העלאה בחצי מהכסף של השחקן

**double** howGoodIsRaiseBig = -howMuchMoneyDoINeedToCall - bigRaise + (chances + risk) \* howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls;

//הגבוה מביניהם תהיה פעולתו של השחקן

Action actionToDo = doMax(howGoodIsFold, howGoodIsCall, howGoodIsRaise40, howGoodIsRaise100, howGoodIsRaiseBig);

בנוסף, ניתן לראות כי נעשה שימוש במשתנה נוסף בשם סיכון risk)). משתנה זה מהווה את רמת הסיכון ששחקן מרשה לעצמו לקחת (כי סך הכל בהימור צריך להסתכן במטרה להרוויח..)

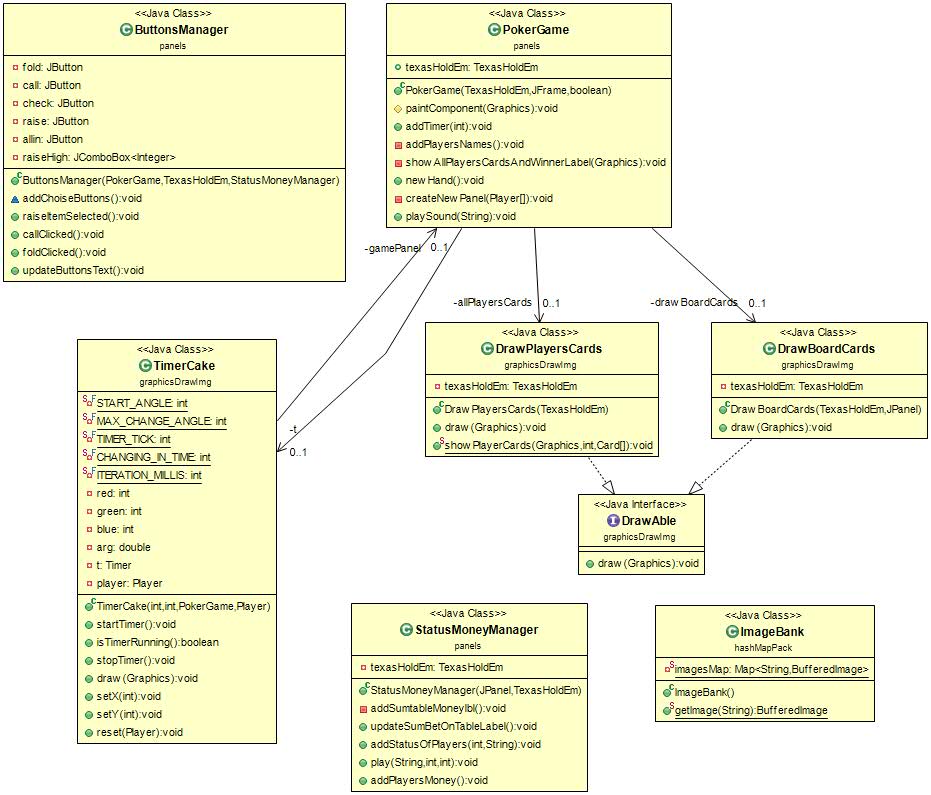
משתנה זה נקבע ע"י:

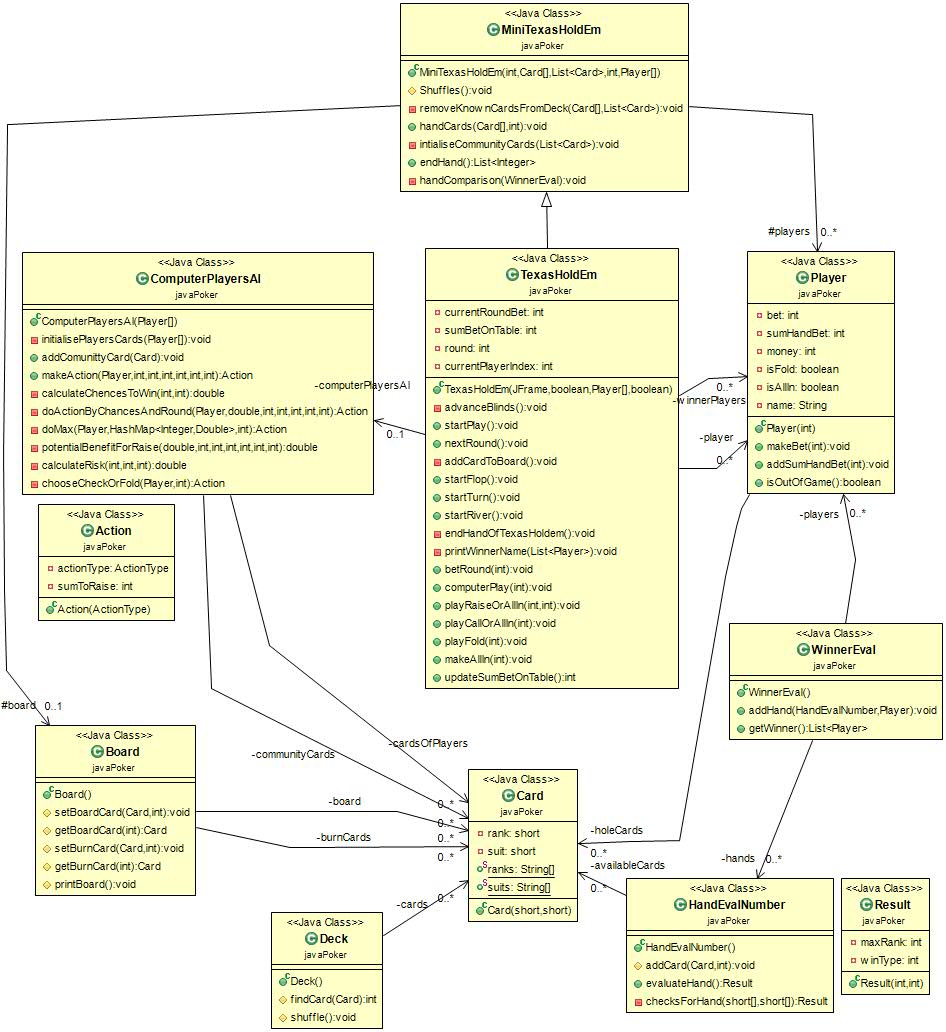
כמה כסף יש לי

כמה כסף יש בקופה

סכום ההעלאה.

תרשים UML של הפרוייקט:





תיאור מבני נתונים:

**Hash Map**

במחלקה ComputerPlayerAI ובעוד מספר מחלקות ב-Package:hashMapPack   
(המחלקות: BlindsHashMap, GreenRectHashMap, ImageBank, NameLblHashMap, TimerCakeHashMap) נעשה שימוש במבנה הנתונים <K,V>.HashMap

כידוע, Hash Map (טבלת גיבוב) הוא מבנה נתונים אשר מאחסן מידע (Value) לפי מפתח מסוים (Key). וכך כל שליפה והכנסה אליו מתקיים בסיבוכיות זמן ריצה של O(1).

במחלקה ComputerPlayerAI ישנו שימוש בטבלת גיבוב שמחזיקה את התוחלת של כל סכום העלאה מסוים. (תוחלת – כמה אני עשוי להרוויח ביחס להעלאה/ השוואה/ פרישה). לפי כל סכום מסוים, מהסכום כסף הנוכחי שהשחקן מחזיק עד לכל הכסף שלו, שהוא מהווה את המפתח, מוחזק מהי התוחלת עבור מפתח זה.   
הסיבה שהשתמשתי דווקא במבנה זה היא כדיי להקטין את תחום המפתחות ולשמרם במערך לצורך שליפה מהירה. כלומר, היה צורך בלשמור את הערכים, והדרך הנוחה והעילה ביותר היא לשמור אותם.

HashMap<Integer, Double> raiseToPotential = **new** HashMap<Integer, Double>();

**for** (**int** money = 0; money <= currentPlayer.getMoney(); money += 20) {

**double** potentialBenefitForRaise = potentialBenefitForRaise(chances, numOfPlayerInHand, howMuchMoneyDoINeedToCall, howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls, money, currentPlayer.getMoney(), currentRoundBet);

raiseToPotential.put(money, potentialBenefitForRaise);

}

action = doMax(currentPlayer, raiseToPotential, currentRoundBet);

במחלקה ImageBank מוחזק עבור כל מפתח שהוא path (משתנה מסוג String של מיקום התמונה בתיקיית הפרויקט) מהו הBufferedImage של התמונה.  
אם המיקום כבר קיים בטבלה, היא תשלוף את ה הBufferedImage, ואחרת היא תשים אותה בטבלה ותחזיר את ה הBufferedImage.

בשאר המחלקות המשתמשות במבנה נתונים זה, אנו מחזיקים עבור אינדקס מסוים של שחקן, מהו המיקום לתמונת הBig/Small Blinds , מהו מיקום המלבן המסמן את הקלפים של השחקן הנוכחי, מהו מיקום הפרטים של השחקן ומהו מיקום הטיימר שלו.

**Linked List**

כידוע, Linked list (רשימה מקושרת) היא אוסף איברים המפוזרים בזיכרון מחשב, בכל איבר מאוחסן מידע וכן מצביע לאיבר הבא ברשימה.

השתמשתי במבנה זה בשביל 2 מטרות עיקריות:

1. במחלקה ComputerPlayerAI, כל קלפי הקהילה הנמצאים כרגע שמורים בתוך מבנה רשימה מקושרת.

2. שמירת השחקנים המנצחים בכל סיבוב (במחלקות WinnerEval ו TexasHoldem מוגדרת רשימה מטיפוס שחקן, שבה הם שמורים, ובמחלקה MiniTexasHoldem הם שמורים ברשימה מטיפוס מספר ממשי (Integer) שמייצג את מספרו של השחקן המנצח).

List<Player> winners = **new** LinkedList<Player>();

הסיבה לשימושים במבנים אלו היא מפני שגמישות בהוספה ובמחיקה של איברים היא תכונה חשובה ברשימה מקושרת, ובמקרים אלו אכן יש לנו צורך להוסיף ולשנות נתונים, וכן לגשת אליהם בצורה יעילה, לכן נעשה כאן שימוש במבנה נתונים זה.

**Array**

כידוע, Array (מערך) הוא אחד ממבני הנתונים הפשוטים ביותר: מערך הוא אוסף פריטים שניתן לגשת אליהם בצורה ישירה באמצעות אינדקס, כך שפעולת הכנסה או שליפה תתבצע בסיבוכיות זמן ריצה של O(1).

בפרויקט היה שימוש נרחב במבנה נתונים זה:

מערכים סטטיים הכוללים את סוגי הקלפים הקיימים במחלקת Card (מערך של מספר הקלף Rank, ומערך של סוג הקלף Suit).

החזקת קלפי הלוח והקלפים השרופים בצורת מערך במחלקה Board.

החזקת ערימת הקלפים במחלקת Deck.

מערך שחקנים המכיל את פרטי כל השחקנים במחלקות TexasHoldem ו-MiniTexasHoldem.

החזקת התמונות המתחלפות במסך הפתיחה מוחזקות גם הן במערך מסוג BufferedImage.

**private** BufferedImage [] images = **new** BufferedImage[3];

הסיבה למימוש במבנה זה, היא כדיי להכניס ולשלוף מידע אשר מקומו ידוע במערך, ביעילות גבוהה.

ניתוח יעילות:

Deck() - Deck

O(n), כאשר :

n - מספר הקלפים בחפיסה.

הסבר: בפעולה זו יש לנו לולאה שעוברת עבור כל הצבעים שיש לקלף (כל הSuit -ים, ובתוכה לולאה שעוברת על כל המספרים שקיימים בצבע. לכן, אנו עוברים בעצם על כל הקלפים, ולכן היעילות היא O של מספר הקלפים בחפיסה.

Shuffles() – MiniTexasHoldem

O(n \* m), כאשר :

m - מספר הערבובים (3 במקרה זה).

n - המספר שהוגרל מ – 0 עד 51.

הסבר: אנו מבצעים כמספר הערבובים – פעולת ערבוב. בפעולת הערבוב אנו מגרילים מספר מ0 ל – 51, ומבצעים בתוך לולאה שרצה מ – 0 עד למספר הנ"ל פעולות בגודל O(1). אח"כ אנו מבצעים פעולת חיתוך של החפיסה המגרילה גם היא מספר מ- 0 ל – 52, ורצים פעמיים, בשתי לולאות **נפרדות** זו מזו, כך שכל לולאה היא בגודל של O(n), ולכן גם הפעולה CutDeck היא בגודל של O(n).

evaluateHand() – HandEval

O(n^2), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

בפעולה ()evaluateHand אנו רצים O(n) 3 פעמים בלולאות נפרדות זו מזו, ולאחר מכן ישנו מיון של 2 מערכים, שכל מיון הוא בגודל של O(n log n). לאחר מכן, עוברים לפעולה ()checksForHand שבה בודקים עבור יד מסוימת מהי חוזקה. אז עוברים החל מהיד החזקה ביותר (רויאל פלאש) עד ליד החלשה ביותר (קלף גבוה), ועבור כל אחד מהבדיקות הנפרדות, פרט לבדיקת רויאל פלאש, הסיבוכיות היא O(n), כאשר n הוא מספר הקלפים האפשריים. בבדיקת רויאל פלאש לעומת זאת, ישנן הקוד הבא:

**for** (**int** i = 0; i < 3; i++) {

**for** (**int** j = 1; j < 4 - i; j++) {

// O(1)

}

}

לפי סכימת סדרה חשבונית, ניתן לומר כי הסיבוכיות זמן ריצה של הקוד הבא הוא O(n^2). מכיוון שזוהי הסיבוכיות הגבוהה ביותר, אזי שזוהי היעלות של כל פונקציית ()evaluateHand.

getWinner() – WinnerEval

O(n^2 \* m), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

m – מספר השחקנים שנשארו ביד הנוכחית.

בפעולה זו אנו עוברים על כל השחקנים, ועבור כל אחד אנו מחשבים את ידו בסיבוכיות O(n^2) ושומרים את היד החזקה ביותר. לאחר מכן עוברים שוב על כל השחקנים, ואם ידם שווה ליד החזקה ביותר, מוסיפים אותם לרשימת המנצחים.

makeAction() – ComputerPlayerAI

O(n^2 \* m \* k), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

m – מספר השחקנים שנותרו בסיבוב.

k – מספר המשחקונים שמורצים ברקע.

הסבר: בפעולה זו אנו קוראים לפעולה calculateChencesToWin. פעולה זו, עבור I=0 עד K, בונה משחק חדש (new MiniTexasHoldEm), ועבור כל משחק מחשבת מיהו המנצח. פעולת בניית המשחק החדש מתבצעת בסיבוכיות זמן ריצה של הפעולה Shuffles, מפני שזו הפעולה שבמקרה הגרועה ביותר מתבצעת הכי הרבה פעמים. לאחר בניית המשחק, מתבצעת פעולת חישוב המנצח, שקוראת בתוכה לפונקציה getWinner, שמתבצעת בסיבוכיות של O(n^2 \* m), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

m – מספר השחקנים שנשארו ביד הנוכחית.

משום שאנו נמצאים בלולאה שקורת K פעמים, אזי שהסיבוכיות היא O(n^2 \* m \* k).

betRound(int bet) – TexasHoldem

O(n^2 \* m^2 \* k), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

m – מספר השחקנים שנותרו בסיבוב.

k – מספר המשחקונים שמורצים ברקע.

הסבר:

betRound קוראת לפעולה computerBetLoop, כאשר שאר הפעולות בה פועלות ב O(1). פעולה זו קוראת לפעולה loopComputerPlayersUntilMeOrLastRaise, שגם שאר הפעולות בה פועלות ביעלות O(1). בפעולה זו קיימת לולאה שרצה כל זמן שהשחקן הנוכחי אינו 0, כל זמן שהשחקן הנוכחי אינו השחקן האחרון שהעלה, וגם שהסיבוב הנוכחי אינו הסיבוב האחרון. לכן, זמן הריצה של הלולאה תלויה באחד מתנאים אלה שיסתיים כך שהיא תיפסק. בגוף הלולאה, אנו בודקים בעזרת הפעולה checkOnePlayerRemianing אם נשאר שחקן אחד במשחק. בדיקה זו עולה לנו במצב הגרוע ביותר O(r) כאשר r מייצג את מספר הסיבובים, אך אם כך יקרה אנו נצא מהלולאה לאחר r סיבובים. מצד שני, ישנו מצב גרוע ביותר אחר שבו הגורם המגביל הוא מספר השחקנים, ובו אנו קוראים לפעולה computerPlay בסיבוכיות של   
O(n^2 \* m \* k). במקרה כזה, הלולאה החיצונית תרוץ m פעמים, ולכן דרך זו מביאה לנו סיבוכיות זמן ריצה גבוהה יותר, המסתכמת   
ב- O(n^2 \* m^2 \* k).  
להלן קוד הפונקציה של loopComputerPlayersUntilMeOrLastRaise, אשר קובע על יעילות הפונקציה betRound.

**loopComputerPlayersUntilMeOrLastRaise() {**

**do** {  
 checkOnePlayerRemianing();  
 **if** (round != ***ENDING\_ROUND***)

computerPlay(currentPlayerIndex);

} **while** (currentPlayerIndex != 0 &&   
 currentPlayerIndex != lastRaisePlayer &&   
 round != ***ENDING\_ROUND***);

**}**

סיבוכיות של עוד פונקציות..

ComputerPlayersAI – initialisePlayersCards()

O(n), כאשר :

n – מספר השחקנים במשחק.

Deck – findCard , Deck - removeCard()

O(n), כאשר :

n – מספר הקלפים בחפיסת קלפים.

HandEval - sortBySuitThenRank()

O(n log n), כאשר :

n – מספר הקלפים המרכיבים "יד".

HandEval – evaluateRoyal()

O(n), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

HandEval – evaluateStraightFlush()

O(n), כאשר :

n – מספר הקלפים ביד פחות מספר הקלפים הנחוצים ליד בחוזק של Straight Flush.

HandEval – evaluateFourOfAKind()

O(n), כאשר :

n – כמות המספרים האפשריים לקלף.

HandEval – evaluateFullHouse()

O(n), כאשר :

n – מספר האפשרויות שמהערך יכול להסתדר בצורה כזו שיהיה רויאל פלאש.

HandEval – evaluateFlush()

O(n), כאשר :

n – מספר הקלפים ביד פחות מספר הקלפים הנחוצים ליד בחוזק של Flush.

HandEval – evaluateStraight()

O(n), כאשר :

n – מספר הקלפים ביד פחות מספר הקלפים הנחוצים ליד בחוזק של Straight.

HandEval – evaluateThreeOfAKind()

O(n), כאשר :

n – כמות המספרים האפשריים לקלף.

HandEval – evaluateTwoPair()

O(n), כאשר :

n – כמות המספרים האפשריים לקלף.

HandEval – evaluateOnePair()

O(n), כאשר :

n – כמות המספרים האפשריים לקלף.

HandEval – evaluateHighCard()

O(n), כאשר :

n – כמות המספרים האפשריים לקלף.

MiniTexasHoldem – handCards()

O(m \* n), כאשר :

m – מספר הקלפים שכל שחקן מחזיק.

n – מספר השחקנים במשחק.

MiniTexasHoldem – intialiseCommunityCards()

O(n), כאשר :

n – מספר הקלפים הסופי הנפתחים על השולחן.

TexasHoldem – getNextPlayerInGame()

O(n), כאשר :

n – מספר השחקנים בשולחן.

TexasHoldem – getNumOfPlayersAvailableToAct ()

O(n), כאשר :

n – מספר השחקנים בשולחן.

TexasHoldem – getNumOfPlayersInHand ()

O(n), כאשר :

n – מספר השחקנים בשולחן.

TexasHoldem – getMoneyInTable ()

O(n), כאשר :

n – מספר השחקנים בשולחן.

TexasHoldem – moveCard ()

O(n), כאשר :

n – המרחק בין המיקום ההתחלתי למיקום הסופי של הקלף.

***BackgroundImg***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import java.io.IOException;

import hashMapPack.ImageBank;

public class BackgroundImg implements DrawAble {

private BufferedImage image;

public BackgroundImg() throws IOException {

String path = "src/img/PokerTable.png";

image = ImageBank.getImage(path);

}

@Override

public void draw(Graphics g) {

g.drawImage(image, 0, 0, 1300, 810, null);

}

}

***BigSmallBlinds***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.Point;

import java.awt.image.BufferedImage;

import java.io.IOException;

import hashMapPack.BlindsHashMap;

import hashMapPack.ImageBank;

import javaPoker.TexasHoldEm;

public class BigSmallBlinds implements DrawAble {

private static final int BLIND\_IMAGE\_SIZE = 40;

private BufferedImage smallBlind;

private BufferedImage bigBlind;

private BlindsHashMap blindsHashMap;

private int smallBlindIndex = TexasHoldEm.smallBlind;

private int bigBlindIndex = TexasHoldEm.bigBlind;

public BigSmallBlinds() throws IOException {

String path = "src/img/SmallBlind.png";

smallBlind = ImageBank.getImage(path);

String path2 = "src/img/BigBlind.png";

bigBlind = ImageBank.getImage(path2);

blindsHashMap = new BlindsHashMap();

}

@Override

public void draw(Graphics g) {

Point small = blindsHashMap.getBlindImageLocation(smallBlindIndex);

Point big = blindsHashMap.getBlindImageLocation(bigBlindIndex);

g.drawImage(smallBlind, (int) small.getX() + 200, (int) small.getY(), BLIND\_IMAGE\_SIZE, BLIND\_IMAGE\_SIZE, null);

g.drawImage(bigBlind, (int) big.getX() + 200, (int) big.getY(), BLIND\_IMAGE\_SIZE, BLIND\_IMAGE\_SIZE, null);

}

}

***ConfettiKinds***

**package** graphicsDrawImg;

**import** java.util.Random;

**public** **enum** ConfettiKinds {

***SPADES***, ***HEART***, ***DIAMONDS***, ***CLUBS***;

**public** **static** ConfettiKinds getRandomKind() {

**return** ConfettiKinds.*values*()[**new** Random().nextInt(ConfettiKinds.*values*().length)];

}

}

***ConfettiManager***

package graphicsDrawImg;

import java.awt.Graphics;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import javax.swing.JPanel;

public class ConfettiManager implements DrawAble {

private static int FPS = 60;

private static int ITERATION\_MILLIS = 1100 / FPS;

private static final int MAX\_FLY\_IMAGES = 5;

private static final long TIME\_BETWEEN\_CONFETTIS\_IN\_MILIS = 1800;

private List<FlyingBackgroundConfetti> drawBackgroundConfettis;

private List<FlyingBackgroundConfetti> removebackgroundConfettis;

private long lastTimeConfettiEntered;

private int lastXEntered;

private JPanel mainPanel;

public ConfettiManager(JPanel mainPanel) {

this.mainPanel = mainPanel;

drawBackgroundConfettis = new LinkedList<FlyingBackgroundConfetti>();

removebackgroundConfettis = new LinkedList<FlyingBackgroundConfetti>();

lastTimeConfettiEntered = 0;

lastXEntered = 0;

}

public void timePassed() {

/\* Add new conffeti to the panel if allowed \*/

if (drawBackgroundConfettis.size() < MAX\_FLY\_IMAGES && System.currentTimeMillis() - lastTimeConfettiEntered >= TIME\_BETWEEN\_CONFETTIS\_IN\_MILIS) {

FlyingBackgroundConfetti newConfetti = new FlyingBackgroundConfetti(ConfettiKinds.getRandomKind(), lastXEntered);

lastXEntered = newConfetti.getX();

drawBackgroundConfettis.add(newConfetti);

lastTimeConfettiEntered = System.currentTimeMillis();

}

/\* Updating all the confetties \*/

for (FlyingBackgroundConfetti drawBackgroundConfetti : drawBackgroundConfettis) {

drawBackgroundConfetti.timePassedConfetti();

if (drawBackgroundConfetti.isOutOfBorders()) {

removebackgroundConfettis.add(drawBackgroundConfetti);

}

}

/\* Remove the out of the borders conffeties \*/

for (FlyingBackgroundConfetti removebackgroundConfetti : removebackgroundConfettis) {

drawBackgroundConfettis.remove(removebackgroundConfetti);

}

removebackgroundConfettis.clear();

mainPanel.repaint();

try {

Thread.sleep(ITERATION\_MILLIS);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public void draw(Graphics g) {

for (FlyingBackgroundConfetti drawBackgroundConfetti : new ArrayList<FlyingBackgroundConfetti>(drawBackgroundConfettis)) {

drawBackgroundConfetti.draw(g);

}

}

}

***DrawAble***

**package** graphicsDrawImg;

**import** java.awt.Graphics;

**public** **interface** DrawAble {

**void** draw(Graphics g);

}

***DrawBoardCards***

package graphicsDrawImg;

import java.awt.Graphics;

import hashMapPack.ImageBank;

import javaPoker.Card;

import javaPoker.Deck;

import javaPoker.TexasHoldEm;

import other.StaticFunctions;

public class DrawBoardCards implements DrawAble {

private TexasHoldEm texasHoldEm;

public static final int START\_X = 600;

public static final int START\_Y = 170;

public static final int BOARD\_CARD\_HEIGHT = 110;

public static final int BOARD\_CARD\_WIDTH = 75;

public static final int SPACE\_BETWEEN\_DECK\_CARDS = 85;

public DrawBoardCards(TexasHoldEm texasHoldEm) {

this.texasHoldEm = texasHoldEm;

}

public static void updateCardToDraw(Card cardToDraw) {

StaticFunctions.currentX = START\_X;

StaticFunctions.currentY = START\_Y;

if (cardToDraw.getRank() != -1) {

String s = Deck.getStringCard(cardToDraw);

String path = "src/cards/" + s + ".png";

StaticFunctions.cardToDraw = ImageBank.getImage(path);

} else

StaticFunctions.cardToDraw = DrawBurnCards.image;

}

@Override

public void draw(Graphics g) {

if (StaticFunctions.isNewRound) {

g.drawImage(StaticFunctions.cardToDraw, StaticFunctions.currentX, StaticFunctions.currentY, BOARD\_CARD\_WIDTH, BOARD\_CARD\_HEIGHT, null);

}

for (int i = 0; i < texasHoldEm.boardCounter; i++) {

String s = Deck.getStringCard(texasHoldEm.getBoardCards()[i]);

String path = "src/cards/" + s + ".png";

g.drawImage(ImageBank.getImage(path), 400 + SPACE\_BETWEEN\_DECK\_CARDS \* i, 365, BOARD\_CARD\_WIDTH, BOARD\_CARD\_HEIGHT, null);

}

}

}

***DrawBurnCards***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import hashMapPack.ImageBank;

import javaPoker.TexasHoldEm;

import panels.PokerGame;

public class DrawBurnCards implements DrawAble {

private static final int CARD\_HEIGHT = PokerGame.BOARD\_CARD\_HEIGHT;

private static final int CARD\_WIDTH = PokerGame.BOARD\_CARD\_WIDTH;

public static final int Y\_DECK\_CARDS = 375;

public static final int SPACE\_BETWEEN\_DECK\_CARDS = PokerGame.SPACE\_BETWEEN\_DECK\_CARDS;

public static final int X\_START\_DECK\_CARDS = 407;

public static BufferedImage image;

private TexasHoldEm texasHoldEm;

public DrawBurnCards(TexasHoldEm texasHoldEm) {

String path = "src/cards/gray\_back.png";

image = ImageBank.getImage(path);

this.texasHoldEm = texasHoldEm;

}

@Override

public void draw(Graphics g) {

int burnCardsNow = texasHoldEm.burnCounter;

if (burnCardsNow > 2) {

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 0, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 3, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 4, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

} else if (burnCardsNow > 1) {

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 0, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 3, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

} else if (burnCardsNow > 0) {

g.drawImage(image, X\_START\_DECK\_CARDS + SPACE\_BETWEEN\_DECK\_CARDS \* 0, Y\_DECK\_CARDS, CARD\_WIDTH, CARD\_HEIGHT, null);

}

}

}

***DrawChancesBar***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.image.BufferedImage;

import hashMapPack.ImageBank;

import javaPoker.TexasHoldEm;

import panels.PokerGame;

public class DrawChancesBar implements DrawAble {

private static final int FPS = 15;

private static final int MIN\_WIDTH\_FOR\_CHANCES\_BAR = 1;

private static final int BAR\_X = 940;

private static final int BAR\_Y = 730;

private TexasHoldEm texasHoldEm;

private PokerGame pokerGame;

private BufferedImage chancesBar, chancesBarContainer;

public static boolean isBarRemoved;

private double chances;

private int currentDrawingChances;

public DrawChancesBar(TexasHoldEm texasHoldEm, PokerGame pokerGame) {

chancesBar = ImageBank.getImage("src/img/ChancesBar.png");

chancesBarContainer = ImageBank.getImage("src/img/ChancesBarContainer.png");

this.texasHoldEm = texasHoldEm;

this.pokerGame = pokerGame;

isBarRemoved = false;

currentDrawingChances = MIN\_WIDTH\_FOR\_CHANCES\_BAR;

updateChancesBar();

}

@Override

public void draw(Graphics g) {

g.drawImage(chancesBarContainer, BAR\_X, BAR\_Y, null);

if (!isBarRemoved)

g.drawImage(getImageByWidth(chancesBar, currentDrawingChances), BAR\_X, BAR\_Y, null);

}

private void calculateMyChnacesOnce() {

chances = 100 \* texasHoldEm.getComputerPlayersAI().calculateChencesToWin(0, texasHoldEm.getNumOfPlayersInHand());

System.out.println("My player chances : " + (int) chances + " %");

}

public void updateChancesBar() {

calculateMyChnacesOnce();

moveChancesOnBar();

}

private void moveChancesOnBar() {

Thread t = new Thread(new Runnable() {

@Override

public void run() {

while (currentDrawingChances != (int) chances) {

if (currentDrawingChances > chances) {

currentDrawingChances--;

} else

currentDrawingChances++;

pokerGame.repaint();

try {

Thread.sleep(FPS);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

});

t.start();

}

public static void removeChancesBar() {

isBarRemoved = true;

}

private Image getImageByWidth(BufferedImage image, double width) {

double newWidth = image.getWidth() \* (width / 100);

if (newWidth < MIN\_WIDTH\_FOR\_CHANCES\_BAR)

newWidth = MIN\_WIDTH\_FOR\_CHANCES\_BAR;

return image.getSubimage(0, 0, (int) newWidth, image.getHeight());

}

}

***DrawGrayPlayers***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import hashMapPack.PlayersImagesLocations;

import hashMapPack.ImageBank;

import javaPoker.Player;

public class DrawGrayPlayers implements DrawAble {

private Player[] players;

private BufferedImage[] grayPlayers;

private PlayersImagesLocations graysHash;

public DrawGrayPlayers(Player[] players) {

this.players = players;

graysHash = new PlayersImagesLocations();

grayPlayers = new BufferedImage[players.length];

for (int i = 0; i < players.length; i++) {

grayPlayers[i] = ImageBank.getImage("src/img/gray" + i + ".png");

}

}

@Override

public void draw(Graphics g) {

for (int i = 0; i < players.length; i++) {

if (players[i].isOutOfGame())

g.drawImage(grayPlayers[i],

(int) graysHash.getGrayPlayersLocation(i).getX(), (int) graysHash.getGrayPlayersLocation(i).getY(),

graysHash.getGrayPlayersSize(i).width, graysHash.getGrayPlayersSize(i).height, null);

}

}

}

***DrawPlayersCards***

**package** graphicsDrawImg;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.geom.AffineTransform;

**import** java.awt.image.AffineTransformOp;

**import** java.awt.image.BufferedImage;

**import** hashMapPack.CardsLocationsHashMap;

**import** hashMapPack.ImageBank;

**import** javaPoker.Deck;

**import** javaPoker.TexasHoldEm;

**import** panels.PokerGame;

**public** **class** DrawPlayersCards **implements** DrawAble {

**private** **static** **final** **int** ***CARD\_WIDTH*** = PokerGame.***CARD\_WIDTH***;

**private** **static** **final** **int** ***CARD\_HEIGHT*** = PokerGame.***CARD\_HEIGHT***;

**private** BufferedImage redBack, redBackRotated, myPlayerimage, myPlayerimage2;

**private** TexasHoldEm texasHoldEm;

**private** **static** CardsLocationsHashMap *cardsLocationsHashMap*;

**public** DrawPlayersCards(TexasHoldEm texasHoldEm) {

redBack = ImageBank.*getImage*("src/cards/red\_back.png");

redBackRotated = ImageBank.*getImage*("src/cards/red\_back\_Rotated.png");

String s = Deck.*getStringCard*(texasHoldEm.getMyPlayerCards()[0]);

String path3 = "src/cards/" + s + ".png";

myPlayerimage = ImageBank.*getImage*(path3);

String s2 = Deck.*getStringCard*(texasHoldEm.getMyPlayerCards()[1]);

String path4 = "src/cards/" + s2 + ".png";

myPlayerimage2 = ImageBank.*getImage*(path4);

**this**.texasHoldEm = texasHoldEm;

*cardsLocationsHashMap* = **new** CardsLocationsHashMap();

}

@Override

**public** **void** draw(Graphics g) {

g.drawImage(myPlayerimage, (**int**) *cardsLocationsHashMap*.getCardsLocation(0, 0).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(0, 0).getY(), ***CARD\_WIDTH***, ***CARD\_HEIGHT***, **null**); // 0's Player (My Player)

g.drawImage(myPlayerimage2, (**int**) *cardsLocationsHashMap*.getCardsLocation(0, 1).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(0, 1).getY(), ***CARD\_WIDTH***, ***CARD\_HEIGHT***, **null**);

**for** (**int** i = 1; i < TexasHoldEm.***NUM\_PLAYERS***; i++) {

**if** (!texasHoldEm.getPlayer()[i].isOutOfGame()) {

**for** (**int** j = 0; j < 2; j++) {

**if** (i == 2 || i == 5) {

g.drawImage(redBackRotated, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, j).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, j).getY(), ***CARD\_HEIGHT***, ***CARD\_WIDTH***, **null**);

} **else** {

g.drawImage(redBack, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, j).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, j).getY(), ***CARD\_WIDTH***, ***CARD\_HEIGHT***, **null**);

}

}

}

}

}

**public** **void** showPlayerCards(Graphics g) {

**for** (**int** i = 0; i < TexasHoldEm.***NUM\_PLAYERS***; i++) {

**if** (!texasHoldEm.getPlayer()[i].isFold()) {

String s = Deck.*getStringCard*(texasHoldEm.getPlayer()[i].getHoleCards()[0]);

String path = "src/cards/" + s + ".png";

BufferedImage image = ImageBank.*getImage*(path);

String s2 = Deck.*getStringCard*(texasHoldEm.getPlayer()[i].getHoleCards()[1]);

String path2 = "src/cards/" + s2 + ".png";

BufferedImage image2 = ImageBank.*getImage*(path2);

**if** (i == 2 || i == 5) {

*drawCardRotated*(g, image, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 0).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 0).getY(), 90); // 2's Player

*drawCardRotated*(g, image2, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 1).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 1).getY(), 90); // 2's Player

} **else** {

g.drawImage(image, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 0).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 0).getY(), ***CARD\_WIDTH***, ***CARD\_HEIGHT***, **null**); // 1's Player

g.drawImage(image2, (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 1).getX(), (**int**) *cardsLocationsHashMap*.getCardsLocation(i, 0).getY(), ***CARD\_WIDTH***, ***CARD\_HEIGHT***, **null**);

}

}

}

}

**public** **static** **void** drawCardRotated(Graphics g, BufferedImage image, **int** x, **int** y, **double** angle) {

// The required drawing location

**int** drawLocationX = x - ***CARD\_WIDTH***;

**int** drawLocationY = y - 38;

// Rotation information

**double** rotationRequired = Math.*toRadians* (angle);

**double** locationX = image.getWidth();

**double** locationY = image.getHeight();

AffineTransform tx = AffineTransform.*getRotateInstance*(rotationRequired, locationX, locationY);

AffineTransformOp op = **new** AffineTransformOp(tx, AffineTransformOp.***TYPE\_BILINEAR***);

// Drawing the rotated image at the required drawing locations

Graphics2D g2d = (Graphics2D) g;

g2d.drawImage(op.filter(image, **null**), drawLocationX, drawLocationY, **null**);

}

}

***FlyingBackgroundConfetti***

package graphicsDrawImg;

import java.awt.Graphics;

import java.awt.Image;

import java.awt.image.BufferedImage;

import java.util.Random;

import hashMapPack.ImageBank;

import main.Main;

public class FlyingBackgroundConfetti implements DrawAble {

private static final int OFFSET\_BETWEEN\_CURRENT\_X\_AND\_LAST\_X = 50;

private double angleOfConfetti;

private int x, y;

private Random rand;

private BufferedImage ConfettiImage;

private Image ConfettiRotatedImage;

public FlyingBackgroundConfetti(ConfettiKinds kind, int lastXEntered) {

String path = null;

switch (kind) {

case SPADES:

path = "src/img/spadesfly.png";

break;

case DIAMONDS:

path = "src/img/diamondfly.png";

break;

case HEART:

path = "src/img/heartfly.png";

break;

case CLUBS:

path = "src/img/clubsfly.png";

break;

default:

break;

}

rand = new Random();

ConfettiImage = ImageBank.getImage(path);

angleOfConfetti = 0;

do {

x = rand.nextInt(Main.START\_PAGE\_WIDTH);

} while ((x + OFFSET\_BETWEEN\_CURRENT\_X\_AND\_LAST\_X > lastXEntered) &&

(x - OFFSET\_BETWEEN\_CURRENT\_X\_AND\_LAST\_X < lastXEntered));

y = -ConfettiImage.getHeight();

}

public void timePassedConfetti() {

y += 2;

angleOfConfetti++;

ConfettiRotatedImage = ImageTool.rotate(ConfettiImage, angleOfConfetti);

}

public boolean isOutOfBorders() {

return ((y > Main.START\_PAGE\_HEIGHT) || (x > Main.START\_PAGE\_WIDTH) || (x < 0));

}

@Override

public void draw(Graphics g) {

g.drawImage(ConfettiRotatedImage, x, y, null);

}

public int getX() {

return x;

}

}

***ImageTool***

**package** graphicsDrawImg;

**import** java.awt.Graphics2D;

**import** java.awt.Image;

**import** java.awt.image.BufferedImage;

**public** **abstract** **class** ImageTool {

**private** ImageTool() {

}

**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;

}

**public** **static** Image toImage(BufferedImage bimage){

// Casting is enough to convert from BufferedImage to Image

Image img = (Image) bimage;

**return** img;

}

**public** **static** Image getEmptyImage(**int** width, **int** height){

BufferedImage img = **new** BufferedImage(width, height, BufferedImage.***TYPE\_INT\_ARGB***);

**return** *toImage*(img);

}

**public** **static** 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);

}

}

***TimerCake***

package graphicsDrawImg;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import javax.swing.Timer;

import javaPoker.Player;

import panels.PokerGame;

public class TimerCake {

private static final int CHANGING\_IN\_TIME = 5;

private static final int START\_ANGLE = 90;

private static final int MAX\_CHANGE\_ANGLE = 360;

private static final double CHANGING\_IN\_ARC = 0.4;

private static final double CHANGING\_IN\_COLOR = 0.5;

private static final int TIMER\_TICK = 1000;

private static final int ITERATION\_MILLIS = TIMER\_TICK / (CHANGING\_IN\_TIME \* 10);

private static final int START\_RED = 0;

private static final int START\_GREEN = 235;

private static final int START\_BLUE = 0;

private static final int RADIUS = 80;

private double i;

private int red, green, blue;

private int x, y;

private double arg;

private PokerGame gamePanel;

private Timer t;

private Player player;

public static boolean suspended;

public TimerCake(int x, int y, PokerGame gamePanel, Player player) {

this.x = x;

this.y = y;

this.gamePanel = gamePanel;

reset(player);

}

public void startTimer() {

t = new Timer(ITERATION\_MILLIS, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (arg / CHANGING\_IN\_ARC > 1) {

arg = arg - CHANGING\_IN\_ARC;

i = i + CHANGING\_IN\_COLOR;

if (red < green) {

red = (int) i;

}

else {

green = START\_GREEN \* 2 - (int) i;

}

gamePanel.repaint();

}

else {

if (player.getBet() == gamePanel.texasHoldEm.getCurrentBet()) {

try {

gamePanel.getButtonsManager().callClicked();

} catch (IOException e1) {

System.out.println("Money / Check sound not found");

e1.printStackTrace();

}

}

else {

try {

gamePanel.getButtonsManager().foldClicked();

} catch (IOException e1) {

System.out.println("Fold sound not found");

e1.printStackTrace();

}

}

}

}

});

t.start();

}

public boolean isTimerRunning() {

if (t != null && t.isRunning()) {

return true;

}

return false;

}

public void stopTimer() {

t.stop();

}

public void draw(Graphics g) {

if (t.isRunning()) {

g.setColor(new Color(red, green, blue));

g.fillArc(x, y, RADIUS, RADIUS, START\_ANGLE, (int) (arg - arg / TIMER\_TICK));

g.setColor(Color.BLACK);

g.drawOval(x, y, RADIUS, RADIUS);

}

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

public void reset(Player player) {

i = 0;

arg = MAX\_CHANGE\_ANGLE;

red = START\_RED;

blue = START\_BLUE;

green = START\_GREEN;

this.player = player;

}

}

***BlindsHashMap***

package hashMapPack;

import java.awt.Point;

import java.util.HashMap;

import java.util.Map;

public class BlindsHashMap {

private Map<Integer, Point> blindsMap;

public BlindsHashMap() {

blindsMap = new HashMap<Integer, Point>();

blindsMap.put(0, new Point(620, 540));

blindsMap.put(1, new Point(120, 540));

blindsMap.put(2, new Point(65, 305));

blindsMap.put(3, new Point(350, 250));

blindsMap.put(4, new Point(650, 260));

blindsMap.put(5, new Point(780, 450));

}

public Point getBlindImageLocation(int playerIndex) {

return blindsMap.get(playerIndex);

}

}

***CardsLocationsHashMap***

package hashMapPack;

import java.awt.Point;

import java.util.HashMap;

import java.util.Map;

public class CardsLocationsHashMap {

private Map<Integer, Point> card1Map, card2Map;

public CardsLocationsHashMap() {

card1Map = new HashMap<Integer, Point>();

card1Map.put(0, new Point(650, 490));

card1Map.put(1, new Point(460, 490));

card1Map.put(2, new Point(250, 350));

card1Map.put(3, new Point(470, 250));

card1Map.put(4, new Point(770, 250));

card1Map.put(5, new Point(870, 350));

card2Map = new HashMap<Integer, Point>();

card2Map.put(0, new Point(740, 490));

card2Map.put(1, new Point(370, 490));

card2Map.put(2, new Point(250, 430));

card2Map.put(3, new Point(380, 250));

card2Map.put(4, new Point(680, 250));

card2Map.put(5, new Point(870, 430));

}

public Point getCardsLocation(int playerIndex, int numCard) {

if (numCard == 0)

return card1Map.get(playerIndex);

if (numCard == 1)

return card2Map.get(playerIndex);

throw new IllegalArgumentException("numCard must be 0 Or 1 Only!");

}

}

***ImageBank***

package hashMapPack;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import javax.imageio.ImageIO;

public abstract class ImageBank {

private static Map<String, BufferedImage> imagesMap = new HashMap<String, BufferedImage>();

public static BufferedImage getImage(String path) {

if (imagesMap.containsKey(path)) {

return imagesMap.get(path);

}

File file = new File(path);

BufferedImage image = null;

try {

image = ImageIO.read(file);

} catch (IOException e) {

e.printStackTrace();

}

imagesMap.put(path, image);

return image;

}

}

***NameLblHashMap***

package hashMapPack;

import java.awt.Point;

import java.util.HashMap;

import java.util.Map;

public class NameLblHashMap {

private static final int LEVEL\_0\_VERTICAL = 0;

private static final int LEVEL\_1\_VERTICAL = 160;

private static final int LEVEL\_2\_VERTICAL = 600;

private Map<Integer, Point> nameLblMap;

public NameLblHashMap() {

nameLblMap = new HashMap<Integer, Point>();

nameLblMap.put(0, new Point(940, LEVEL\_2\_VERTICAL));

nameLblMap.put(1, new Point(140, LEVEL\_2\_VERTICAL));

nameLblMap.put(2, new Point(40, LEVEL\_1\_VERTICAL));

nameLblMap.put(3, new Point(160, LEVEL\_0\_VERTICAL));

nameLblMap.put(4, new Point(940, LEVEL\_0\_VERTICAL));

nameLblMap.put(5, new Point(1100, LEVEL\_1\_VERTICAL));

}

public Point getNameLblLocation(int playerIndex) {

return nameLblMap.get(playerIndex);

}

}

***PlayersImagesLocations***

package hashMapPack;

import java.awt.Point;

import java.util.HashMap;

import java.util.Map;

import com.sun.glass.ui.Size;

public class PlayersImagesLocations {

private Map<Integer, Point> locationsMap;

private Map<Integer, Size> sizeMap;

public PlayersImagesLocations() {

locationsMap = new HashMap<Integer, Point>();

locationsMap.put(0, new Point(773, 572));

locationsMap.put(1, new Point(266, 564));

locationsMap.put(2, new Point(20, 256));

locationsMap.put(3, new Point(310, 40));

locationsMap.put(4, new Point(760, 24));

locationsMap.put(5, new Point(1022, 247));

sizeMap = new HashMap<Integer, Size>();

sizeMap.put(0, new Size(172, 236));

sizeMap.put(1, new Size(187, 243));

sizeMap.put(2, new Size(229, 280));

sizeMap.put(3, new Size(156, 209));

sizeMap.put(4, new Size(144, 228));

sizeMap.put(5, new Size(179, 296));

}

public Point getGrayPlayersLocation(int playerIndex) {

return locationsMap.get(playerIndex);

}

public Size getGrayPlayersSize(int playerIndex) {

return sizeMap.get(playerIndex);

}

}

***TimerCakeHashMap***

package hashMapPack;

import java.awt.Point;

import java.util.HashMap;

import java.util.Map;

public class TimerCakeHashMap {

private Map<Integer, Point> timerMap;

public TimerCakeHashMap() {

timerMap = new HashMap<Integer, Point>();

timerMap.put(0, new Point(730, 680));

timerMap.put(1, new Point(400, 680));

timerMap.put(2, new Point(130, 440));

timerMap.put(3, new Point(280, 160));

timerMap.put(4, new Point(860, 170));

timerMap.put(5, new Point(1040, 440));

}

public Point getTimerCakeLocation(int playerIndex) {

return timerMap.get(playerIndex);

}

}

***Action***

**package** javaPoker;

**public** **class** Action {

**private** ActionType actionType;

**private** **int** sumToRaise;

**public** Action(ActionType actionType) {

**this**.actionType = actionType;

**this**.sumToRaise = 0;

}

**public** Action(ActionType actionType, **int** sumToRaise) {

**this**.actionType = actionType;

**this**.sumToRaise = sumToRaise;

}

**public** ActionType getActionType() {

**return** actionType;

}

**public** **int** getSumToRaise() {

**return** sumToRaise;

}

}

***ActionType***

**package** javaPoker;

**public** **enum** ActionType {

***RAISE***, ***CALL***, ***CHECK***, ***FOLD***, ***ALL\_IN***;

}

***Board***

**package** javaPoker;

**public** **class** Board {

**private** **static** **final** **int** ***NUMBER\_OF\_BURN\_CARDS*** = 3;

**private** **static** **final** **int** ***NUMBER\_OF\_BOARD\_CARDS*** = 5;

**private** Card[] board = **new** Card[***NUMBER\_OF\_BOARD\_CARDS***];

**private** Card[] burnCards = **new** Card[***NUMBER\_OF\_BURN\_CARDS***];

**protected** **void** setBoardCard(Card card, **int** cardNum) {

**this**.board[cardNum] = card;

}

**protected** Card getBoardCard(**int** cardNum) {

**return** **this**.board[cardNum];

}

**protected** **void** setBurnCard(Card card, **int** cardNum) {

**this**.burnCards[cardNum] = card;

}

**protected** Card getBurnCard(**int** cardNum) {

**return** **this**.burnCards[cardNum];

}

**protected** **int** boardSize() {

**return** board.length;

}

**protected** **void** printBoard() {

System.***out***.println("The board contains the following cards:");

**for**(**int** i = 0; i < board.length; i++) {

System.***out***.println(i + 1 + ": " + getBoardCard(i).printCard());

}

System.***out***.println("\n");

}

**protected** **void** printBurnCards() {

System.***out***.println("The burn cards are:");

**for**(**int** i = 0; i < burnCards.length; i++) {

System.***out***.println(i + 1 + ": " + getBurnCard(i).printCard());

}

System.***out***.println("\n");

}

}

***Card***

**package** javaPoker;

**import** java.util.Comparator;

**public** **class** Card {

**private** **short** rank, suit;

// Strings of card's ranks : Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King

**public** **static** String[] *ranks* = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"};

// Strings of card's suits : Diamonds, Clubs, Hearts, Spades

**public** **static** String[] *suits* = {"Diamonds", "Clubs", "Hearts", "Spades"};

**public** Card(**short** rank, **short** suit) {

**this**.rank = rank;

**this**.suit = suit;

}

**public** **short** getSuit() {

**return** suit;

}

**public** **short** getRank() {

**return** rank;

}

**public** **static** String rankAsString(**int** \_rank) {

**return** *ranks*[\_rank];

}

**public** **static** String suitAsString(**int** \_suit) {

**return** *suits*[\_suit];

}

**protected** String printCard() {

**return** *ranks*[rank] + " of " + *suits*[suit];

}

**public** **static** **boolean** sameCard(Card card1, Card card2) {

**return** (card1.rank == card2.rank && card1.suit == card2.suit);

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + rank;

result = prime \* result + suit;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Card other = (Card) obj;

**if** (rank != other.rank)

**return** **false**;

**if** (suit != other.suit)

**return** **false**;

**return** **true**;

}

}

**class** rankComparator **implements** Comparator<Object> {

**public** **int** compare(Object card1, Object card2) **throws** ClassCastException {

// verify two Card objects are passed in

**if** (!((card1 **instanceof** Card) && (card2 **instanceof** Card))){

**throw** **new** ClassCastException("A Card object was expeected. "

+ "Parameter 1 class: " + card1.getClass()

+ " Parameter 2 class: " + card2.getClass());

}

**short** rank1 = ((Card)card1).getRank();

**short** rank2 = ((Card)card2).getRank();

**return** rank1 - rank2;

}

}

**class** suitComparator **implements** Comparator<Object> {

**public** **int** compare(Object card1, Object card2) **throws** ClassCastException {

// verify two Card objects are passed in

**if** (!((card1 **instanceof** Card) && (card2 **instanceof** Card))){

**throw** **new** ClassCastException("A Card object was expeected. "

+ "Parameter 1 class: " + card1.getClass()

+ " Parameter 2 class: " + card2.getClass());

}

**short** suit1 = ((Card)card1).getSuit();

**short** suit2 = ((Card)card2).getSuit();

**return** suit1 - suit2;

}

}

***ComputerPlayersAI***

package javaPoker;

import java.util.HashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Random;

public class ComputerPlayersAI {

private static final int NUMBER\_OF\_SIMULATIONS = 10000;

public static int nowNum;

private Card[][] cardsOfPlayers;

private List<Card> communityCards;

private Random rand = new Random();

public ComputerPlayersAI(Player[] players) {

communityCards = new LinkedList<Card>();

initialisePlayersCards(players);

}

private void initialisePlayersCards(Player[] players) {

cardsOfPlayers = new Card[TexasHoldEm.NUM\_PLAYERS][2];

for (int i = 0; i < cardsOfPlayers.length; i++) {

cardsOfPlayers[i][0] = players[i].getCard(0);

cardsOfPlayers[i][1] = players[i].getCard(1);

System.out.println("Player number " + (i + 1) + " holds: ");

System.out.print(Card.rankAsString(cardsOfPlayers[i][0].getRank()) + " of " + Card.suitAsString(cardsOfPlayers[i][0].getSuit()) + " , And ");

System.out.println(Card.rankAsString(cardsOfPlayers[i][1].getRank()) + " of " + Card.suitAsString(cardsOfPlayers[i][1].getSuit()));

}

}

public void addComunittyCard(Card card) {

communityCards.add(card);

}

public Action makeAction(Player currentPlayer, int currentPlayerIndex, int currentRoundBet, int sumMoneyOnTable, int round, int numPlayersInHand) {

System.out.println(" \n\n---------- player " + currentPlayerIndex + " start action --------------");

double chances = calculateChencesToWin(currentPlayerIndex, numPlayersInHand);

Action action = doActionByPotentialBenefit(currentPlayer, chances, currentRoundBet, round, sumMoneyOnTable, numPlayersInHand);

System.out.println("Current Player : " + (currentPlayerIndex) + ", chances " + chances + " , in round: " + round);

return action;

}

public double calculateChencesToWin(int currentPlayerIndex, int numOfPlayerInHand) {

double score = 0;

for (int i = 0; i < NUMBER\_OF\_SIMULATIONS; i++) {

// FORDEBUG

//nowNum = i;

MiniTexasHoldEm miniTexasHoldEm = new MiniTexasHoldEm(numOfPlayerInHand, cardsOfPlayers[currentPlayerIndex], communityCards, currentPlayerIndex, null);

List<Integer> winnersId = miniTexasHoldEm.endHand();

if (winnersId.contains(0))

score += 1.0 / winnersId.size();

}

return score / NUMBER\_OF\_SIMULATIONS;

}

private Action doActionByPotentialBenefit(Player currentPlayer, double chances, int currentRoundBet, int round, int sumMoneyOnTable, int numOfPlayerInHand) {

Action action = null;

int howMuchMoneyDoINeedToCall = (currentRoundBet - currentPlayer.getBet());

int howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls = sumMoneyOnTable + (currentRoundBet \* numOfPlayerInHand);

HashMap<Integer, Double> raiseToPotential = new HashMap<Integer, Double>();

for (int raise = 0; raise <= currentPlayer.getMoney(); raise += 20) {

double potentialBenefitForRaise = potentialBenefitForRaise(chances, numOfPlayerInHand, howMuchMoneyDoINeedToCall,

howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls, raise, currentPlayer.getMoney(), currentRoundBet);

raiseToPotential.put(raise, potentialBenefitForRaise);

// System.out.println(raise + " : " + potentialBenefitForRaise);

}

action = doMax(currentPlayer, raiseToPotential, currentRoundBet);

return action;

}

private Action doMax(Player currentPlayer, HashMap<Integer, Double> raiseToPotenional, int currentRoundBet) {

Action action = null;

double max = 0;

int maxRaise = -1;

for (Integer raise : raiseToPotenional.keySet()) {

if (raiseToPotenional.get(raise) > max) {

max = raiseToPotenional.get(raise);

maxRaise = raise;

}

}

if (maxRaise == -1) {

action = chooseCheckOrFold(currentPlayer, currentRoundBet);

return action;

}

if (maxRaise == 0) {

action = new Action(ActionType.CALL);

return action;

}

action = new Action(ActionType.RAISE, maxRaise + currentRoundBet);

return action;

}

private double potentialBenefitForRaise(double chances, int numOfPlayerInHand, int howMuchMoneyDoINeedToCall,

int howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls, int raise, int moneyOfPlayer, int currentRoundBet) {

double risk = calculateRisk(raise, moneyOfPlayer, howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls);

return -howMuchMoneyDoINeedToCall - raise + (chances \* risk) \* (howMuchMoneyWillThereBeInTheBoardIfEveryoneAfterMeCalls + (raise \* numOfPlayerInHand));

}

private double calculateRisk(int raise, int moneyOfPlayer, int moneyOnBoardAfterCall) {

if (raise == 0) {

return 1;

}

double randomValue = 0.75 + (0.95 - 0.75) \* rand.nextDouble();

// minimum = 0.75 , maximum = 0.95

return Math.max(randomValue - ((double)raise / moneyOfPlayer) - ((double)raise / Math.max(moneyOnBoardAfterCall, 20) / 100), 0.2);

}

private Action chooseCheckOrFold(Player currentPlayer, int currentRoundBet) {

Action action;

if (currentPlayer.getBet() == currentRoundBet)

action = new Action(ActionType.CALL);

else

action = new Action(ActionType.FOLD);

return action;

}

}

***Deck***

package javaPoker;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Random;

public class Deck {

private static final int NUMBER\_OF\_RANKS = 13;

private static final int NUMBER\_OF\_SUITS = 4;

private Card[] cards = new Card[NUMBER\_OF\_RANKS \* NUMBER\_OF\_SUITS];

public Deck() {

int i = 0;

for (short j = 0; j < NUMBER\_OF\_SUITS; j++) {

for (short k = 0; k < NUMBER\_OF\_RANKS; k++) {

cards[i] = new Card(k, j);

i++;

}

}

}

public Card[] getCards() {

return cards;

}

protected void printDeck() {

System.out.println("Length: " + cards.length);

for (int i = 0; i < cards.length; i++) {

System.out.println(i + 1 + ": " + cards[i].printCard());

}

System.out.println("\n");

}

protected int findCard(Card card){

for (int i = 0; i < cards.length; i++) {

if (Card.sameCard(cards[i], card)) {

return i;

}

}

return -1;

}

protected Card getCard(int cardNum) {

return cards[cardNum];

}

public void removeCard(Card card) {

List<Card> cardsList = new ArrayList<Card> (Arrays.asList(cards));

cardsList.remove(card);

cards = cardsList.toArray(new Card[cardsList.size()]);

// FORDEBUG

if (ComputerPlayersAI.nowNum == 5) {

System.out.println("\n" + Card.rankAsString(card.getRank()) + " of " + Card.suitAsString(card.getSuit()) + " Removed");

}

}

protected void shuffle() {

int length = cards.length;

Random random = new Random();

for (int i = 0; i < length; i++) {

int change = i + random.nextInt(length - i);

swapCards(i, change);

}

}

protected void cutDeck() {

Deck tempDeck = new Deck();

Random random = new Random();

int cutNum = random.nextInt(cards.length);

for (int i = 0; i < cutNum; i++) {

tempDeck.cards[i] = this.cards[cards.length - cutNum + i];

}

for (int j = 0; j < cards.length - cutNum; j++) {

tempDeck.cards[j + cutNum] = this.cards[j];

}

this.cards = tempDeck.cards;

}

public static String getStringCard(Card card) {

String suitStr = "";

String rankStr = "";

short rank = (short) (card.getRank() + 1);

short suit = card.getSuit();

switch (suit) {

case 0:

suitStr = "Diamonds";

break;

case 1:

suitStr = "Clubs";

break;

case 2:

suitStr = "Hearts";

break;

case 3:

suitStr = "Spades";

break;

}

switch (rank) {

case 1:

rankStr = "Ace";

break;

case 11:

rankStr = "Jack";

break;

case 12:

rankStr = "Queen";

break;

case 13:

rankStr = "King";

break;

default:

rankStr = String.valueOf(rank);

}

return (rankStr + suitStr);

}

private void swapCards(int a, int b) {

Card temp = cards[a];

cards[a] = cards[b];

cards[b] = temp;

}

}

***HandEval***

**package** javaPoker;

**import** java.util.Arrays;

**public** **class** HandEval {

**private** Card[] availableCards = **new** Card[7];

**private** **final** **static** **short** ***ONE*** = 1;

**private** **final** **static** **short** ***TWO*** = 2;

**private** **final** **static** **short** ***THREE*** = 3;

**private** **final** **static** **short** ***FOUR*** = 4;

**protected** **void** addCard(Card card, **int** i) {

availableCards[i] = card;

}

**protected** Card getCard(**int** i) {

**return** availableCards[i];

}

**protected** **int** numCards() {

**return** availableCards.length;

}

**protected** **void** sortByRank() {

Arrays.*sort*(availableCards, **new** rankComparator());

}

**protected** **void** sortBySuit() {

Arrays.*sort*(availableCards, **new** suitComparator());

}

**protected** **void** sortBySuitThenRank() {

Arrays.*sort*(availableCards, **new** suitComparator());

Arrays.*sort*(availableCards, **new** rankComparator());

}

**protected** **void** sortByRankThenSuit() {

Arrays.*sort*(availableCards, **new** rankComparator());

Arrays.*sort*(availableCards, **new** suitComparator());

}

**public** Result evaluateHand() {

Result handResult; // holds String of what hand does the Player have

**short**[] rankCounter = **new** **short**[13];

**short**[] suitCounter = **new** **short**[4];

// Initialisations of rankCounter array and suitCounter array

**for** (**int** i = 0; i < rankCounter.length; i++) {

rankCounter[i] = 0;

}

**for** (**int** i = 0; i < suitCounter.length; i++) {

suitCounter[i] = 0;

}

// Loop through sorted cards and total ranks

**for** (**int** i = 0; i < availableCards.length; i++) {

rankCounter[availableCards[i].getRank()]++;

suitCounter[availableCards[i].getSuit()]++;

}

//sort cards for evaluation

sortByRankThenSuit();

// hands are already sorted by rank and suit for royal and straight flush checks.

handResult = checksForHand(rankCounter, suitCounter);

**return** handResult;

}

**private** Result checksForHand(**short**[] rankCounter, **short**[] suitCounter) {

Result handResult = **null**;

//int handResult = 0;

// check for royal flush

handResult = evaluateRoyal(rankCounter, suitCounter);

// check for straight flush

**if** (handResult == **null**) {

handResult = evaluateStraightFlush(rankCounter, suitCounter);

}

// check for four of a kind

**if** (handResult == **null**) {

handResult = evaluateFourOfAKind(rankCounter);

}

// check for full house

**if** (handResult == **null**) {

handResult = evaluateFullHouse(rankCounter);

}

// check for flush

**if** (handResult == **null**) {

handResult = evaluateFlush(rankCounter, suitCounter);

}

// check for straight

**if** (handResult == **null**) {

// re-sort by rank, up to this point we had sorted by rank and suit

// but a straight is suit independent.

sortByRank();

handResult = evaluateStraight(rankCounter);

}

// check for three of a kind

**if** (handResult == **null**) {

handResult = evaluateThreeOfAKind(rankCounter);

}

// check for two pair

**if** (handResult == **null**) {

handResult = evaluateTwoPair(rankCounter);

}

// check for one pair

**if** (handResult == **null**) {

handResult = evaluateOnePair(rankCounter);

}

// check for highCard

**if** (handResult == **null**) {

handResult = evaluateHighCard(rankCounter);

}

**return** handResult;

}

**private** Result evaluateRoyal(**short**[] rankCounter, **short**[] suitCounter) {

Result r = **null**;

// Check if there are 5 of one suit, if not royal is impossible.

**if** ((rankCounter[9] >= 1 && /\* 10 \*/

rankCounter[10] >= 1 && /\* Jack \*/

rankCounter[11] >= 1 && /\* Queen \*/

rankCounter[12] >= 1 && /\* King \*/

rankCounter[0] >= 1) /\* Ace \*/

&& (suitCounter[0] > 4 || suitCounter[1] > 4 ||

suitCounter[2] > 4 || suitCounter[3] > 4)) {

royalSearch:

**for** (**int** i = 0; i < 3; i++) {

// Check if first card is the ace.

// Ace must be in position 0, 1 or 2

**if** (availableCards[i].getRank() == 0) {

// because the ace could be the first card in the array

// but the remaining 4 cards could start at position

// 1, 2 or 3 loop through checking each possibility.

**for** (**int** j = 1; j < 4 - i; j++) {

**if** ((availableCards[i + j].getRank() == 9 &&

availableCards[i + j + 1].getRank() == 10 &&

availableCards[i + j + 2].getRank() == 11 &&

availableCards[i + j + 3].getRank() == 12)

&&

(availableCards[i].getSuit() == availableCards[i + j].getSuit() &&

availableCards[i].getSuit() == availableCards[i + j + 1].getSuit() &&

availableCards[i].getSuit() == availableCards[i + j + 2].getSuit() &&

availableCards[i].getSuit() == availableCards[i + j + 3].getSuit())) {

// Found royal flush, break and return.

String handRes = "Royal Flush! Suit: " + Card.*suitAsString*(availableCards[i].getSuit());;

r = **new** Result(1, 10, handRes);

**break** royalSearch;

}

}

}

}

}

**return** r;

}

**private** Result evaluateStraightFlush(**short**[] rankCounter, **short**[] suitCounter) {

Result r = **null**;

**if** (suitCounter[0] > 4 || suitCounter[1] > 4 ||

suitCounter[2] > 4 || suitCounter[3] > 4) {

// min. requirements for a straight flush have been met.

// Loop through available cards looking for 5 consecutive cards of the same suit,

// start in reverse to get the highest value straight flush

**for** (**int** i = availableCards.length - 1; i > 3; i--) {

**if** ((availableCards[i].getRank() - ***ONE*** == availableCards[i - ***ONE***].getRank() &&

availableCards[i].getRank() - ***TWO*** == availableCards[i - ***TWO***].getRank() &&

availableCards[i].getRank() - ***THREE*** == availableCards[i - ***THREE***].getRank() &&

availableCards[i].getRank() - ***FOUR*** == availableCards[i - ***FOUR***].getRank())

&&

(availableCards[i].getSuit() == availableCards[i - ***ONE***].getSuit() &&

availableCards[i].getSuit() == availableCards[i - ***TWO***].getSuit() &&

availableCards[i].getSuit() == availableCards[i - ***THREE***].getSuit() &&

availableCards[i].getSuit() == availableCards[i - ***FOUR***].getSuit())) {

// Found royal flush, break and return.

String handRes = "Straight Flush: " + Card.*rankAsString*(availableCards[i].getRank()) + " high of " + Card.*suitAsString*(availableCards[i].getSuit());

r = **new** Result(availableCards[i].getRank(), 9, handRes);

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

}

**return** r;

}

**private** Result evaluateFourOfAKind(**short**[] rankCounter) {

Result r = **null**;

**for** (**int** i = 0; i < rankCounter.length; i++) {

**if** (rankCounter[i] == ***FOUR***) {

r = **new** Result(i, 8, "Four of a Kind: " + Card.*rankAsString*(i) +"'s");

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

**return** r;

}

**private** Result evaluateFullHouse(**short**[] rankCounter) {

Result r = **null**;

**short** threeOfKindRank = -1;

**short** twoOfKindRank = -1;

**for** (**int** i = rankCounter.length; i > 0; i--) {

**if** ((threeOfKindRank < (**short**) 0) || (twoOfKindRank < (**short**) 0)) {

**if** ((rankCounter[i - ***ONE***]) > 2) {

threeOfKindRank = (**short**) (i - ***ONE***);

} **else** **if** ((rankCounter[i - ***ONE***]) > 1) {

twoOfKindRank = (**short**) (i - ***ONE***);

}

} **else** {

**break**;

}

}

**if** ((threeOfKindRank >= (**short**) 0) && (twoOfKindRank >= (**short**) 0)) {

r = **new** Result(threeOfKindRank, 7, "Full House: " + Card.*rankAsString*(threeOfKindRank) + "'s full of " + Card.*rankAsString*(twoOfKindRank) + "'s");

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

}

**return** r;

}

**private** Result evaluateFlush(**short**[] rankCounter, **short**[] suitCounter){

Result r = **null**;

// verify at least 1 suit has 5 cards or more.

**if** (suitCounter[0] > 4 || suitCounter[1] > 4 ||

suitCounter[2] > 4 || suitCounter[3] > 4) {

**for** (**int** i = availableCards.length - 1; i > 3; i--) {

**if** (availableCards[i].getSuit() == availableCards[i-***ONE***].getSuit() &&

availableCards[i].getSuit() == availableCards[i-***TWO***].getSuit() &&

availableCards[i].getSuit() == availableCards[i-***THREE***].getSuit() &&

availableCards[i].getSuit() == availableCards[i-***FOUR***].getSuit()) {

// Found royal flush, break and return.

r = **new** Result(availableCards[i].getRank(), 6, "Flush: " + Card.*rankAsString*(availableCards[i].getRank()) + " high of " + Card.*suitAsString*(availableCards[i].getSuit()));

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

}

**return** r;

}

**private** Result evaluateStraight(**short**[] rankCounter) {

Result r = **null**;

// loop through rank array to check for 5 consecutive

// index with a value greater than zero

**for** (**int** i = rankCounter.length; i > 4; i--) {

**if** ((rankCounter[i - 1] > 0) &&

(rankCounter[i - 2] > 0) &&

(rankCounter[i - 3] > 0) &&

(rankCounter[i - 4] > 0) &&

(rankCounter[i - 5] > 0)) {

r = **new** Result(i - ***ONE***, 5, "Straight: " + Card.*rankAsString*(i-1) + " high");

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

**return** r;

}

**private** Result evaluateThreeOfAKind(**short**[] rankCounter) {

Result r = **null**;

// loop through rank array to check for 5 consecutive

// index with a value greater than zero

**for** (**int** i = rankCounter.length; i > 0; i--) {

**if** (rankCounter[i-1] > 2) {

r = **new** Result(i - ***ONE***, 4, "Three of a Kind: " + Card.*rankAsString*(i - 1) + "'s");

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

**return** r;

}

**private** Result evaluateTwoPair(**short**[] rankCounter) {

Result r = **null**;

**short** firstPairRank = -1;

**short** secondPairRank = -1;

**for** (**int** i = rankCounter.length; i > 0; i--){

**if** ((firstPairRank < (**short**)0) || (secondPairRank < (**short**) 0)) {

**if** (((rankCounter[i - ***ONE***]) > 1) && (firstPairRank < (**short**) 0)) {

firstPairRank = (**short**) (i-***ONE***);

}

**else** **if** ((rankCounter[i - ***ONE***]) > 1){

secondPairRank = (**short**)(i - ***ONE***);

}

}

**else**

{

// two pair found, break loop.

**break**;

}

}

// populate output

**if** ((firstPairRank >= (**short**) 0) && (secondPairRank >= (**short**)0)) {

**if** (secondPairRank == (**short**) 0) {

// Aces serve as top rank but are at the bottom of the rank array

// swap places so aces show first as highest pair

r = **new** Result(13, 3, "Two Pair: " + Card.*rankAsString*(secondPairRank) + "'s and " + Card.*rankAsString*(firstPairRank) + "'s");

}

**else**

{

r = **new** Result(firstPairRank, 3, "Two Pair: " + Card.*rankAsString*(firstPairRank) + "'s and " + Card.*rankAsString*(secondPairRank) + "'s");

}

}

**return** r;

}

**private** Result evaluateOnePair(**short**[] rankCounter) {

Result r = **null**;

**for** (**int** i = rankCounter.length; i > 0; i--) {

**if**((rankCounter[i - ***ONE***]) > 1) {

r = **new** Result(i - ***ONE***, 2, "One Pair: " + Card.*rankAsString*(i - ***ONE***) + "'s");

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

**return** r;

}

**private** Result evaluateHighCard(**short**[] rankCounter) {

Result r = **null**;

**for** (**int** i = rankCounter.length; i > 0; i--) {

**if**((rankCounter[i - ***ONE***]) > 0) {

r = **new** Result(i - ***ONE***, 1, "High Card: " + Card.*rankAsString*(i - ***ONE***));

**if** (r.getMaxRank() == 0)

r.setMaxRank(13);

**break**;

}

}

**return** r;

}

}

***MiniTexasHoldEm***

package javaPoker;

import java.util.LinkedList;

import java.util.List;

import other.Sound;

public class MiniTexasHoldEm {

private static final int NUMBER\_OF\_SHUFFELES = 3;

private int numOfPlayers;

private int currentPlayer;

protected Deck holdemDeck;

protected int cardCounter;

public int burnCounter;

public int boardCounter;

protected Board board;

protected Player[] players;

protected String winner;

protected String winnersHand;

protected Result[] handsValues;

private WinnerEval winnerEval;

public MiniTexasHoldEm (int numOfPlayers, Card[] cardsOfPlayer, List<Card> communityCards, int currentPlayer, Player[] playersArr) {

initialiseFields(numOfPlayers, currentPlayer);

Shuffles();

if (cardsOfPlayer != null)

removeKnownCardsFromDeck(cardsOfPlayer, communityCards);

if (playersArr == null) {

players = new Player[numOfPlayers];

for (int i = 0; i < numOfPlayers; i++) {

players[i] = new Player(i);

}

} else {

players = playersArr;

for (int i = 0; i < numOfPlayers; i++) {

players[i].setBet(0);

if (players[i].getMoney() == 0)

players[i].setFold(true);

else

players[i].setFold(false);

players[i].setAllIn(false);

players[i].setSumHandBet(0);

}

}

handCards(cardsOfPlayer, currentPlayer);

if (currentPlayer != -1)

intialiseCommunityCards(communityCards);

}

protected void initialiseFields(int numOfPlayers, int currentPlayer) {

this.currentPlayer = currentPlayer;

this.numOfPlayers = numOfPlayers;

board = new Board();

handsValues = new Result[numOfPlayers];

winnerEval = new WinnerEval();

holdemDeck = new Deck();

cardCounter = 0;

burnCounter = 0;

boardCounter = 0;

}

protected void Shuffles() {

for (int i = 0; i < NUMBER\_OF\_SHUFFELES; i++) {

holdemDeck.shuffle();

}

holdemDeck.cutDeck();

}

private void removeKnownCardsFromDeck(Card[] cardsOfPlayer, List<Card> communityCards) {

for (Card communityCard : communityCards) {

holdemDeck.removeCard(communityCard);

}

holdemDeck.removeCard(cardsOfPlayer[0]);

holdemDeck.removeCard(cardsOfPlayer[1]);

}

public void handCards(Card[] cardsOfPlayer, int currentPlayer) {

int hold;

if (currentPlayer == -1)

hold = 0;

else {

hold = 1;

players[0].setCard(cardsOfPlayer[0], 0);

players[0].setCard(cardsOfPlayer[1], 1);

}

for (int i = 0; i < 2; i++)

for (int j = hold; j < numOfPlayers; j++)

players[j].setCard(holdemDeck.getCard(cardCounter++), i);

}

private void intialiseCommunityCards(List<Card> communityCards) {

for (Card communityCard : communityCards) {

board.setBoardCard(communityCard, boardCounter++);

}

while (boardCounter < 5) {

board.setBoardCard(holdemDeck.getCard(cardCounter++), boardCounter++);

}

}

public List<Integer> endHand() {

handComparison();

List<Player> winnerPlayers = winnerEval.getWinner();

winnersHand = handsValues[winnerPlayers.get(0).getId()].getHandString();

if (currentPlayer == -1)

Sound.playSoundOfWinType(handsValues[winnerPlayers.get(0).getId()].getWinType());

List<Integer> winnersId = new LinkedList<Integer>();

for (Player player : winnerPlayers) {

winnersId.add(player.getId());

}

return winnersId;

}

private void handComparison() {

for (int i = 0; i < numOfPlayers; i++) {

if (!players[i].isFold()) {

HandEval handToEval = new HandEval();

// populate with player cards

for (int j = 0; j < players[i].holeCardsSize(); j++){

handToEval.addCard(players[i].getCard(j), j);

}

//populate with board cards

for (int j = players[i].holeCardsSize(); j < (players[i].holeCardsSize() + board.boardSize()); j++){

handToEval.addCard(board.getBoardCard(j - players[i].holeCardsSize()), j);

}

Result evaluator = handToEval.evaluateHand();

if (currentPlayer == -1) {

System.out.println("Player " + (i + 1) + " hand value: " + evaluator.getHandString());

System.out.println("Player " + (i + 1) + " hand win type value: " + evaluator.getWinType());

System.out.println("Player " + (i + 1) + " max rank: " + (evaluator.getMaxRank() + 1));

}

handsValues[i] = evaluator;

winnerEval.addHand(handToEval, players[i]);

}

}

}

public Player[] getPlayer() {

return players;

}

public String getWinner() {

return winner;

}

public String getWinnersHand() {

return winnersHand;

}

public Card[] getBoardCards() {

Card[] boardCards = {board.getBoardCard(0), board.getBoardCard(1), board.getBoardCard(2), board.getBoardCard(3), board.getBoardCard(4)};

return boardCards;

}

protected void initBets() {

for (Player player : players) {

player.setBet(0);

}

}

public Card[] getMyPlayerCards() {

Card[] firstPlayerCards = players[0].getHoleCards();

return firstPlayerCards;

}

}

***Player***

**package** javaPoker;

**public** **class** Player {

**private** **static** **final** **int** ***START\_MONEY*** = 10000;

**private** Card[] holeCards = **new** Card[2];

**private** **int** bet;

**private** **int** sumHandBet;

**private** **int** money;

**private** **boolean** isFold;

**private** **boolean** isAllIn;

**private** String name;

**private** **int** id;

**public** Player(**int** id) {

money = ***START\_MONEY***;

bet = 0;

sumHandBet = 0;

isFold = **false**;

isAllIn = **false**;

**this**.id = id;

}

**public** Card[] getHoleCards() {

**return** holeCards;

}

**public** **int** getBet() {

**return** bet;

}

**public** **void** setBet(**int** bet) {

**this**.bet = bet;

}

**public** **void** makeBet(**int** bet) {

money += **this**.bet;

**this**.bet = bet;

money -= bet;

}

**public** **void** setSumHandBet(**int** sumHandBet) {

**this**.sumHandBet = sumHandBet;

}

**public** **int** getSumHandBet() {

**return** sumHandBet;

}

**public** **void** addSumHandBet(**int** sumHandBet) {

**this**.sumHandBet += sumHandBet;

}

**public** **int** getMoney() {

**return** money;

}

**public** **void** setMoney(**int** money) {

**this**.money = money;

}

**public** **int** getId() {

**return** id;

}

**public** **boolean** isFold() {

**return** isFold;

}

**public** **void** setFold(**boolean** isFold) {

**this**.isFold = isFold;

}

**public** **boolean** isAllIn() {

**return** isAllIn;

}

**public** **void** setAllIn(**boolean** isAllIn) {

**this**.isAllIn = isAllIn;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**protected** **void** setCard(Card card, **int** cardNum){

holeCards[cardNum] = card;

}

**protected** Card getCard(**int** cardNum){

**return** holeCards[cardNum];

}

**protected** **int** holeCardsSize(){

**return** holeCards.length;

}

**protected** **void** printPlayerCards(**int** playerIndex) {

System.***out***.println("Player " + (playerIndex + 1) + " hole cards:");

**for** (**int** i = 0; i < 2; i++) {

System.***out***.println(holeCards[i].printCard());

}

System.***out***.println("\n");

}

**public** **boolean** isOutOfGame() {

**return** money == 0 && isFold == **true**;

}

}

***Result***

**package** javaPoker;

**public** **class** Result {

**private** **int** maxRank;

**private** **int** winType;

**private** String handString;

**public** Result(**int** maxRank, **int** winType, String handString) {

**this**.maxRank = maxRank;

**this**.winType = winType;

**this**.handString = handString;

}

**public** **int** getMaxRank() {

**return** maxRank;

}

**public** **void** setMaxRank(**int** maxRank) {

**this**.maxRank = maxRank;

}

**public** **int** getWinType() {

**return** winType;

}

**public** String getHandString() {

**return** handString;

}

}

***TexasHoldEm***

package javaPoker;

import java.io.IOException;

import java.util.LinkedList;

import java.util.List;

import java.util.Random;

import javax.swing.JFrame;

import graphicsDrawImg.DrawBoardCards;

import graphicsDrawImg.DrawBurnCards;

import other.Sound;

import other.StaticFunctions;

import panels.PokerGame;

import panels.StartPage;

public class TexasHoldEm extends MiniTexasHoldEm {

private static final int ENDING\_ROUND = 4;

public static final String ALL\_IN = "ALL IN";

public static final String RAISE = "Raise";

public static final String CALL = "Call";

public static final String CHECK = "Check";

public static final String FOLD = "Fold";

public static final String OUT = "Out";

public static final int NUM\_PLAYERS = 6;

private ComputerPlayersAI computerPlayersAI;

private PokerGame pokerGame;

private Player[] player;

private int currentRoundBet;

private int sumBetOnTable;

private int round;

private Random rand = new Random();

private int currentPlayerIndex;

public static int smallBlind, bigBlind;

private int lastRaisePlayer;

private List<Player> winnerPlayers;

public TexasHoldEm (JFrame frame, boolean isNewGame, Player[] players) throws Exception {

super(NUM\_PLAYERS, null, null, -1, players);

initialiseFields();

if (isNewGame) {

smallBlind = rand.nextInt(6);

bigBlind = (smallBlind + 1) % NUM\_PLAYERS;

inisialisePlayersNames();

} else {

advanceBlinds();

System.out.println("smallBlind " + smallBlind);

System.out.println("bigBlind " + bigBlind);

}

computerPlayersAI = new ComputerPlayersAI(player);

pokerGame = new PokerGame(this, frame);

for (int i = 0; i < NUM\_PLAYERS; i++) {

if (player[i].isFold())

pokerGame.getStatusMoneyManager().play(OUT, 0, i);

}

}

private void advanceBlinds() {

smallBlind = getNextPlayerInGame(smallBlind);

bigBlind = getNextPlayerInGame(smallBlind);

}

private int getNextPlayerInGame(int playerIndex) {

do {

playerIndex = (playerIndex + 1) % NUM\_PLAYERS;

} while (player[playerIndex].isFold());

return playerIndex;

}

private void initialiseFields() {

player = super.players;

currentRoundBet = 0;

sumBetOnTable = 0;

round = 0;

}

private void inisialisePlayersNames() {

player[0].setName(StartPage.name);

player[1].setName("John");

player[2].setName("Mike");

player[3].setName("David");

player[4].setName("Thomas");

player[5].setName("Gal-Cohen");

}

public void startPlay() {

playRaiseOrAllIn(smallBlind, 20);

playRaiseOrAllIn(bigBlind, 40);

pokerGame.repaint();

try {

Sound.playSound("StartBets");

Thread.sleep(2000);

} catch (IOException | InterruptedException e) {

System.out.println("Sound \"start bets\" not found or thread interrupted.");

}

pokerGame.getStatusMoneyManager().addMoneyLabelToPlayer(smallBlind);

pokerGame.getStatusMoneyManager().addMoneyLabelToPlayer(bigBlind);

currentPlayerIndex = (bigBlind + 1) % NUM\_PLAYERS;

computerBetLoop();

if (round != ENDING\_ROUND) {

pokerGame.getButtonsManager().updateButtonsText();

pokerGame.getButtonsManager().setButtonsEnabled(true);

pokerGame.addTimer(currentPlayerIndex);

} else {

pokerGame.newHand();

}

}

public void nextRound() {

try {

pokerGame.t.stopTimer();

} catch (NullPointerException e) {

System.out.println("Timer already stopped");

}

pokerGame.t = null;

currentPlayerIndex = smallBlind;

lastRaisePlayer = -1;

currentRoundBet = 0;

updateSumBetOnTable();

pokerGame.getStatusMoneyManager().updateSumBetOnTableLabel();

initBets();

try {

if (round == 3) {

Thread.sleep(1000);

Sound.playSound("LetsSeeEm");

Thread.sleep(1000);

}

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

switch (round) {

case 0:

startFlop();

round++;

break;

case 1:

startTurn();

round++;

break;

case 2:

startRiver();

round++;

break;

case 3:

endHandOfTexasHoldem();

round++;

break;

default:

break;

}

pokerGame.drawChancesBar.updateChancesBar();

pokerGame.getStatusMoneyManager().setAllChangingPlayersStatusLabelsVisible(false);

}

private void addCardToBoard() {

Card newCard = holdemDeck.getCard(cardCounter);

board.setBoardCard(newCard, boardCounter);

computerPlayersAI.addComunittyCard(holdemDeck.getCard(cardCounter++));

moveCardToBoard(newCard);

boardCounter++;

}

private void moveCardToBoard(Card newCard) {

StaticFunctions.isNewRound = true;

DrawBoardCards.updateCardToDraw(newCard);

if (newCard.getRank() != -1) {

try {

Sound.playSound("cardSlide8");

} catch (IOException e) {

e.printStackTrace();

}

StaticFunctions.moveCard(pokerGame, 400 + DrawBoardCards.SPACE\_BETWEEN\_DECK\_CARDS \* boardCounter, 365, true);

} else {

try {

Sound.playSound("cardShove2");

} catch (IOException e) {

e.printStackTrace();

}

StaticFunctions.moveCard(pokerGame, DrawBurnCards.X\_START\_DECK\_CARDS + DrawBurnCards.SPACE\_BETWEEN\_DECK\_CARDS \* boardCounter, DrawBurnCards.Y\_DECK\_CARDS, false);

}

StaticFunctions.isNewRound = false;

pokerGame.repaint();

}

public void startFlop() {

addBurnCardToBoard();

for (int i = 0; i < 3; i++){

addCardToBoard();

}

}

public void startTurn() {

addBurnCardToBoard();

addCardToBoard();

}

public void startRiver() {

addBurnCardToBoard();

addCardToBoard();

}

private void addBurnCardToBoard() {

moveCardToBoard(new Card((short)-1, (short)-1));

board.setBurnCard(holdemDeck.getCard(cardCounter++), burnCounter++);

}

private void endHandOfTexasHoldem() {

System.out.println("The hand is complete...\n");

System.out.println("Sum Money on Table: " + sumBetOnTable);

holdemDeck.printDeck();

board.printBoard();

System.out.println("The player cards are the following:\n");

for (int i = 0; i < NUM\_PLAYERS; i++) {

player[i].printPlayerCards(i);

}

board.printBurnCards();

List<Integer> winnersId = endHand();

winnerPlayers = new LinkedList<Player>();

for (Integer playerId : winnersId) {

winnerPlayers.add(player[playerId]);

}

printWinnerName(winnerPlayers);

int numWinners = winnerPlayers.size();

if (numWinners > 1) {

System.out.println("Some winners: ");

for (int i = 0; i < numWinners; i++) {

System.out.println(winnerPlayers.get(i).getName());

winnerPlayers.get(i).setMoney(winnerPlayers.get(i).getMoney() + (sumBetOnTable / numWinners));

}

}

else {

System.out.println("Winner is: " + winnerPlayers.get(0).getName());

winnerPlayers.get(0).setMoney(winnerPlayers.get(0).getMoney() + sumBetOnTable);

}

}

private void printWinnerName(List<Player> winnerPlayers) {

if (winnerPlayers.size() == 1) {

if (winnerPlayers.get(0) == player[0])

winner = "You Won The Hand!";

else

winner = "Hand's Winner is: " + winnerPlayers.get(0).getName();

}

else {

winner = "Winners Are: ";

winner += winnerPlayers.get(0).getName();

for (int i = 1; i < winnerPlayers.size(); i++) {

winner += ", " + winnerPlayers.get(i).getName();

}

}

}

public void betRound(int bet) {

makeMyPlayerBet(bet);

if (lastRaisePlayer == -1)

lastRaisePlayer = 0;

currentPlayerIndex = (currentPlayerIndex + 1) % NUM\_PLAYERS;

computerBetLoop();

}

public void makeMyPlayerBet(int bet) {

if (!player[0].isFold() && !player[0].isAllIn()) {

if (bet == player[0].getMoney() && !player[0].isFold())

makeAllIn(0);

else {

player[0].addSumHandBet(bet - player[0].getBet());

player[0].makeBet(bet);

}

if (bet > currentRoundBet) {

currentRoundBet = bet;

lastRaisePlayer = 0;

}

pokerGame.getStatusMoneyManager().addMoneyLabelToPlayer(0);

}

}

public void computerBetLoop() {

loopComputerPlayersUntilMeOrLastRaise();

if (currentPlayerIndex == lastRaisePlayer) {

nextRound();

pokerGame.repaint();

if (currentPlayerIndex != 0) {

loopComputerPlayersUntilMeOrLastRaise();

if (currentPlayerIndex == lastRaisePlayer) {

nextRound();

pokerGame.repaint();

}

}

}

checkOnePlayerRemianing();

if (round != ENDING\_ROUND && !player[0].isFold() && !player[0].isAllIn()) {

pokerGame.getButtonsManager().setButtonsEnabled(true);

pokerGame.getButtonsManager().updateButtonsText();

}

}

private void loopComputerPlayersUntilMeOrLastRaise() {

if (currentPlayerIndex != lastRaisePlayer && currentPlayerIndex != 0) {

do {

checkOnePlayerRemianing();

if (round != ENDING\_ROUND) {

try {

computerPlay(currentPlayerIndex);

} catch (Exception e) {

e.printStackTrace();

}

currentPlayerIndex = (currentPlayerIndex + 1) % NUM\_PLAYERS;

}

} while (currentPlayerIndex != 0 && currentPlayerIndex != lastRaisePlayer && round != ENDING\_ROUND);

}

}

private void checkOnePlayerRemianing() {

if (getNumOfPlayersAvailableToAct() == 0 || (getNumOfPlayersAvailableToAct() == 1 && player[currentPlayerIndex].getBet() == currentRoundBet)) {

while (round < ENDING\_ROUND) {

nextRound();

}

}

}

public void computerPlay(int playerIndex) throws Exception {

if (playerIndex > 5 || playerIndex < 1) {

throw new RuntimeException("Illegal player index received");

}

pokerGame.repaint();

if (!player[playerIndex].isFold() && player[playerIndex].getMoney() != 0) {

pokerGame.addTimer(playerIndex);

try {

//int x = rand.nextInt(5500) + 500;

int x = 1000;

Thread.sleep(x);

} catch (InterruptedException e) {

e.printStackTrace();

}

Action actionToMake = computerPlayersAI.makeAction(player[playerIndex], playerIndex, currentRoundBet, getMoneyInTable(), round, getNumOfPlayersInHand());

ActionType actionType = actionToMake.getActionType();

if (actionType != ActionType.RAISE)

System.out.println("Player number " + playerIndex + " choose to " + actionType.name());

else

System.out.println("Player number " + playerIndex + " choose to " + actionType.name() + " in " + actionToMake.getSumToRaise());

if (actionType == ActionType.RAISE)

playRaiseOrAllIn(playerIndex, actionToMake.getSumToRaise());

else if (actionType == ActionType.CALL)

playCallOrAllIn(playerIndex);

else if (actionType == ActionType.FOLD)

playFold(playerIndex);

}

pokerGame.getStatusMoneyManager().addMoneyLabelToPlayer(playerIndex);

}

public void playRaiseOrAllIn(int playerIndex, int raiseSum) {

if (player[playerIndex].getMoney() - raiseSum > 0) {

player[playerIndex].addSumHandBet(raiseSum - player[playerIndex].getBet());

player[playerIndex].makeBet(raiseSum);

currentRoundBet = raiseSum;

lastRaisePlayer = playerIndex;

pokerGame.getStatusMoneyManager().play(RAISE, currentRoundBet, playerIndex);

} else {

if (raiseSum > currentRoundBet)

lastRaisePlayer = playerIndex;

makeAllIn(playerIndex);

}

}

public void playCallOrAllIn(int playerIndex) {

if (player[playerIndex].getMoney() + player[playerIndex].getBet() - currentRoundBet > 0)

{

if (player[playerIndex].getBet() == currentRoundBet)

pokerGame.getStatusMoneyManager().play(CHECK, 0, playerIndex);

else {

player[playerIndex].addSumHandBet(currentRoundBet - player[playerIndex].getBet());

player[playerIndex].makeBet(currentRoundBet);

pokerGame.getStatusMoneyManager().play(CALL, currentRoundBet, playerIndex);

}

if (lastRaisePlayer == -1)

lastRaisePlayer = playerIndex;

} else

makeAllIn(playerIndex);

}

public void playFold(int playerIndex) {

player[playerIndex].setFold(true);

pokerGame.getStatusMoneyManager().play(FOLD, 0, playerIndex);

}

public void makeAllIn(int playerIndex) {

System.out.println(playerIndex + " make all in");

player[playerIndex].setAllIn(true);

int raiseSum = player[playerIndex].getMoney() + player[playerIndex].getBet();

player[playerIndex].makeBet(raiseSum);

if (currentRoundBet < raiseSum) {

currentRoundBet = raiseSum;

lastRaisePlayer = playerIndex;

}

pokerGame.getStatusMoneyManager().play(ALL\_IN, raiseSum, playerIndex);

}

private int getNumOfPlayersAvailableToAct() {

int count = 0;

for (int i = 0; i < NUM\_PLAYERS; i++)

if (player[i].isFold() || player[i].isAllIn())

count++;

return NUM\_PLAYERS - count;

}

public int getNumOfPlayersInHand() {

int count = 0;

for (int i = 0; i < NUM\_PLAYERS; i++)

if (player[i].isFold())

count++;

return NUM\_PLAYERS - count;

}

public int getCurrentBet() {

return currentRoundBet;

}

public int updateSumBetOnTable() {

int sum = 0;

for (int i = 0; i < NUM\_PLAYERS; i++)

sum += player[i].getBet();

sumBetOnTable += sum;

return sumBetOnTable;

}

private int getMoneyInTable() {

int sum = 0;

for (int i = 0; i < player.length; i++) {

sum += player[i].getSumHandBet();

}

return sum;

}

public int getSumBetOnTable() {

return sumBetOnTable;

}

public int getRound() {

return round;

}

public int getCurrentPlayerIndex() {

return currentPlayerIndex;

}

public PokerGame getPokerGame() {

return pokerGame;

}

public boolean isWholeGameOver() {

int downPlayersCount = 0;

for (int i = 0; i < NUM\_PLAYERS; i++) {

if (player[i].getMoney() == 0)

downPlayersCount++;

}

if (downPlayersCount == NUM\_PLAYERS - 1)

return true;

return false;

}

public boolean checkBetsEquals() {

for (int i = 1; i < NUM\_PLAYERS; i++) {

if (!player[i].isFold() && !player[i].isAllIn())

if (!(player[i].getBet() == currentRoundBet))

return false;

}

return true;

}

public List<Player> getWinnerPlayers() {

return winnerPlayers;

}

public ComputerPlayersAI getComputerPlayersAI() {

return computerPlayersAI;

}

}

***WinnerEval***

package javaPoker;

import java.util.ArrayList;

import java.util.List;

public class WinnerEval {

private List<HandEval> hands;

private List<Player> players;

public WinnerEval() {

hands = new ArrayList<HandEval>();

players = new ArrayList<Player>();

}

public void addHand(HandEval he, Player player) {

hands.add(he);

players.add(player);

}

public List<Player> getWinner() {

List<Player> winners = new ArrayList<Player>();

int maxWinningType = 0, maxValue = 0;

for (int i = 0; i < hands.size(); i++) {

if (hands.get(i).evaluateHand().getWinType() > maxWinningType) {

maxValue = hands.get(i).evaluateHand().getMaxRank();

maxWinningType = hands.get(i).evaluateHand().getWinType();

}

else if (hands.get(i).evaluateHand().getWinType() == maxWinningType) {

if (hands.get(i).evaluateHand().getMaxRank() > maxValue) {

maxValue = hands.get(i).evaluateHand().getMaxRank();

}

}

}

for (int i = 0; i < hands.size(); i++) {

if (hands.get(i).evaluateHand().getWinType() == maxWinningType && hands.get(i).evaluateHand().getMaxRank() == maxValue) {

winners.add(players.get(i));

}

}

return winners;

}

}

***ButtonsManager***

package panels;

import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JComboBox;

import graphicsDrawImg.DrawChancesBar;

import javaPoker.TexasHoldEm;

import other.Sound;

public class ButtonsManager {

private static final int DISTANCE\_BETWEEN\_BUTTONS = 67;

private static final int Y\_BUTTONS = 500;

private static final int X\_BUTTONS = 1120;

private StatusMoneyManager statusMoneyManager;

private TexasHoldEm texasHoldEm;

private PokerGame pokerGame;

private JButton fold, call, check, raise, allin;

private boolean isRaiseClicked;

private JComboBox<Integer> raiseHigh;

public ButtonsManager(PokerGame mainPanel, TexasHoldEm texasHoldEm) {

isRaiseClicked = false;

this.pokerGame = mainPanel;

this.texasHoldEm = texasHoldEm;

this.statusMoneyManager = mainPanel.getStatusMoneyManager();

}

public void addChoiseButtons() throws IOException {

fold = new JButton(new ImageIcon(makePNGButtonsPath("FoldButton1")));

changeButtons(fold, 0);

call = new JButton(new ImageIcon(makePNGButtonsPath("CallButton1")));

changeButtons(call, 1);

check = new JButton(new ImageIcon(makePNGButtonsPath("CheckButton1")));

changeButtons(check, 1);

raise = new JButton(new ImageIcon(makePNGButtonsPath("RaiseButton1")));

changeButtons(raise, 2);

allin = new JButton(new ImageIcon(makePNGButtonsPath("AllInButton1")));

changeButtons(allin, 2);

updateButtonsText();

fold.addActionListener(foldAction);

call.addActionListener(callCheckAction);

check.addActionListener(callCheckAction);

raise.addActionListener(raiseAction);

pokerGame.add(fold);

pokerGame.add(call);

pokerGame.add(check);

pokerGame.add(raise);

}

ActionListener foldAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

foldClicked();

} catch (IOException e1) {

System.out.println("Fold Sound not found");

e1.printStackTrace();

}

}

};

ActionListener callCheckAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

callClicked();

} catch (IOException e1) {

System.out.println("Money / Check Sound not found");

e1.printStackTrace();

}

}

};

ActionListener raiseAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (!isRaiseClicked) {

isRaiseClicked = true;

// Adds values to raiseHigh ComboBox

int minRaise = (texasHoldEm.getCurrentBet() + 21) / 20;

int maxRaise = (texasHoldEm.getPlayer()[0].getMoney() + texasHoldEm.getPlayer()[0].getBet()) / 20;

Integer[] raiseInts = new Integer[maxRaise - minRaise + 1];

for (int i = 0; i < raiseInts.length; i++) {

raiseInts[i] = (i + minRaise) \* 20;

}

raiseHigh = new JComboBox<Integer>(raiseInts);

raiseHigh.setSelectedItem(raiseInts[0]);

raiseHigh.setVisible(true);

// Value in raiseHigh clicked

raiseHigh.addActionListener(raiseHighAction);

raiseHigh.setVisible(true);

pokerGame.add(raiseHigh);

raiseHigh.setBounds(X\_BUTTONS, 710, 100, 30);

} else {

isRaiseClicked = false;

raiseHigh.setVisible(false);

}

}

};

ActionListener raiseHighAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

isRaiseClicked = false;

try {

raiseItemSelected();

} catch (IOException e1) {

System.out.println("Money Sound not found");

e1.printStackTrace();

}

}

};

public BufferedImage makePNGButtonsPath(String string) throws IOException {

String path = "src/img/" + string + ".png";

File file = new File(path);

return ImageIO.read(file);

}

private void changeButtons(JButton button, int i) {

button.setBorder(null);

button.setContentAreaFilled(false);

button.setOpaque(false);

button.setFocusPainted(false);

button.setBorderPainted(false);

button.setBackground(new Color(117, 62, 199));

button.setBounds(X\_BUTTONS, Y\_BUTTONS + DISTANCE\_BETWEEN\_BUTTONS \* i, 172, 62);

button.addMouseListener(createMouseOver());

}

private MouseAdapter createMouseOver() {

return new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

super.mouseEntered(e);

((JButton) e.getSource()).setOpaque(true);

((JButton) e.getSource()).setFocusPainted(true);

((JButton) e.getSource()).setBorderPainted(true);

}

@Override

public void mouseExited(MouseEvent e) {

super.mouseExited(e);

((JButton) e.getSource()).setOpaque(false);

((JButton) e.getSource()).setFocusPainted(false);

((JButton) e.getSource()).setBorderPainted(false);

}

};

}

public void raiseItemSelected() throws IOException {

final int x = (int) raiseHigh.getSelectedItem();

System.out.println("\n\nYou raised to " + x);

Sound.playSound("moneySound");

Sound.playSound("PlayerRaises");

statusMoneyManager.addStatusOfPlayers(0, "Raise");

statusMoneyManager.play("Raise", x, 0);

raiseHigh.setVisible(false);

setButtonsEnabled(false);

makeBetRoundAndHandleGameContinue(x);

}

public void callClicked() throws IOException {

statusMoneyManager.addStatusOfPlayers(0, "Call");

if (texasHoldEm.getPlayer()[0].getBet() == texasHoldEm.getCurrentBet()) {

Sound.playSound("PlayerCheck");

Sound.playSound("callSound");

System.out.println("\n\nYou checked");

statusMoneyManager.myStatuslbl.setText("Check");

} else {

Sound.playSound("moneySound");

Sound.playSound("PlayerCalls");

System.out.println("\n\nYou called");

statusMoneyManager.myStatuslbl.setText("Call: " + texasHoldEm.getCurrentBet() + " $");

}

try {

raiseHigh.setVisible(false);

} catch (Exception e) { }

setButtonsEnabled(false);

makeBetRoundAndHandleGameContinue(texasHoldEm.getCurrentBet());

}

private void makeBetRoundAndHandleGameContinue(final int bet) {

Runnable runnable = new Runnable() {

public void run() {

statusMoneyManager.updateSumBetOnTableLabel();

texasHoldEm.betRound(bet);

if (texasHoldEm.getRound() == 4) {

pokerGame.newHand();

setButtonsVisible(false);

} else {

updateButtonsText();

setButtonsEnabled(true);

pokerGame.addTimer(texasHoldEm.getCurrentPlayerIndex());

}

pokerGame.repaint();

}

};

Thread thread = new Thread(runnable);

thread.start();

}

public void foldClicked() throws IOException {

System.out.println("\n\nYou folded");

Sound.playSound("foldSound");

Sound.playSound("PlayerFolds");

statusMoneyManager.addStatusOfPlayers(0, "Fold");

statusMoneyManager.myStatuslbl.setText("Fold");

DrawChancesBar.removeChancesBar();

try {

raiseHigh.setVisible(false);

} catch (Exception e) { }

setButtonsVisible(false);

texasHoldEm.getPlayer()[0].setFold(true);

Runnable runnable = new Runnable() {

public void run() {

while (texasHoldEm.getRound() < 4)

{

texasHoldEm.betRound(texasHoldEm.getCurrentBet());

if (texasHoldEm.getRound() == 4) {

pokerGame.newHand();

}

}

pokerGame.repaint();

}

};

Thread thread = new Thread(runnable);

thread.start();

}

public void setButtonsEnabled(boolean enabled) {

raise.setEnabled(enabled);

fold.setEnabled(enabled);

call.setEnabled(enabled);

check.setEnabled(enabled);

allin.setEnabled(enabled);

pokerGame.repaint();

}

private void setButtonsVisible(boolean visibility) {

raise.setVisible(visibility);

fold.setVisible(visibility);

call.setVisible(visibility);

check.setVisible(visibility);

allin.setVisible(visibility);

pokerGame.repaint();

}

public void updateButtonsText() {

if (texasHoldEm.getPlayer()[0].getBet() == texasHoldEm.getCurrentBet()) {

call.setVisible(false);

check.setVisible(true);

} else {

call.setVisible(true);

check.setVisible(false);

}

if (texasHoldEm.getPlayer()[0].getMoney() + texasHoldEm.getPlayer()[0].getBet() < texasHoldEm.getCurrentBet() + 20) {

pokerGame.remove(call);

pokerGame.remove(check);

changeButtons(fold, 1);

changeRaiseToAllIn();

}

pokerGame.repaint();

}

private void changeRaiseToAllIn() {

pokerGame.remove(raise);

pokerGame.add(allin);

pokerGame.repaint();

if (allin.getActionListeners().length == 0) {

allin.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

allInClicked();

}

});

}

}

public void allInClicked() {

try {

Sound.playSound("moneySound");

} catch (IOException e1) {

System.out.println("Money Sound not found.");

}

texasHoldEm.makeMyPlayerBet(texasHoldEm.getPlayer()[0].getMoney());

setButtonsVisible(false);

Runnable runnable = new Runnable() {

public void run() {

while (texasHoldEm.getRound() < 4) {

if (texasHoldEm.checkBetsEquals())

texasHoldEm.nextRound();

else

texasHoldEm.betRound(0);

if (texasHoldEm.getRound() == 4)

pokerGame.newHand();

}

pokerGame.repaint();

}

};

Thread thread = new Thread(runnable);

thread.start();

}

}

***GuidePage***

package panels;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.BoxLayout;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextPane;

import javax.swing.SwingConstants;

import main.Main;

@SuppressWarnings("serial")

public class GuidePage extends JPanel {

private final JFrame frame;

private JPanel p;

private JButton previous;

JScrollPane scrollBar;

public GuidePage(JFrame frame) throws IOException {

this.frame = frame;

p = this;

setLayout(null);

addText();

}

private void addText() throws IOException {

JPanel txtPanel = new JPanel();

txtPanel.setLayout(new BoxLayout(txtPanel, BoxLayout.Y\_AXIS));

txtPanel.setBackground(Color.WHITE);

JLabel title = new JLabel("How To Play Poker ?");

title.setPreferredSize(new Dimension(0, 100));

title.setFont(new Font("Garamond", Font.BOLD, 70));

title.setForeground(Color.RED);

title.setHorizontalAlignment(SwingConstants.CENTER);

txtPanel.add(title);

JLabel title2 = new JLabel("<html><u>Texas Holde'm Rules<u></html>");

title2.setPreferredSize(new Dimension(0, 100));

title2.setFont(new Font("Garamond", Font.BOLD, 55));

title2.setForeground(Color.RED);

title2.setHorizontalAlignment(SwingConstants.CENTER);

txtPanel.add(title2);

JTextPane pane = new JTextPane();

pane.setContentType("text/html");

pane.setEditable(false);

pane.setPreferredSize(new Dimension(0, 950));

pane.setForeground(Color.YELLOW);

pane.setText("<html><p><h1><u>Betting Structures</u></h1></p>"

+ "<p><h2>One player acts as dealer. This position is called the button and it rotates clockwise after every hand.</h2>"

+ "</br><h2>The two players to the left of the dealer are called the small blind and the big blind, respectively.</h2>"

+ "</br><h2>These two positions require forced bets of a pre-determined amount and are the only players to put money in the pot before the cards are dealt (if no ante in place).</h2>"

+ "</br><h2>Every player then receives two cards face down. These are called “hole” cards.</h2>"

+ "</br><h2>Once all hole cards have been dealt, the first betting round begins with the player sitting immediately to the left of the big blind. This player can fold, call (match the amount of the big blind) or raise.</h2>"

+ "</br><h2>Betting then continues clockwise, with each player having the option to fold, call the amount of the highest bet before them, bet or raise.</h2>"

+ "</br><h2>When the first betting round is completed, three community cards are flipped face up on the table. This is called the flop.</h2>"

+ "</br><h2>The betting resumes, clockwise, with each player having the option to check (if no bet is in front of them), bet (or raise if a bet is before them), call or fold.</h2>"

+ "</br><h2>When the second round of betting is finished, a fourth community card is flipped face up on the table. This is called the turn.</h2>"

+ "</br><h2>The third round of betting commences with the first remaining player sitting to the left of the button.</h2>"

+ "</br><h2>When the third round of betting is over, a fifth community card is flipped face up on the table. This is called the river.</h2>"

+ "</br><h2>The fourth round of betting starts with the first remaining player seated to the left of the button. The betting continues to move clockwise.</h2>"

+ "</p></html>");

pane.setForeground(Color.RED);

txtPanel.add(pane);

JTextPane pane2 = new JTextPane();

pane2.setContentType("text/html");

pane2.setPreferredSize(new Dimension(0, 550));

pane2.setText("<html><p><h1><u>The Details</u></h1></p>"

+ "<p><h2>Now here are some more detailed looks at aspects of Texas Hold’em.</h2>"

+ "<h2> Dealer, Small Blind and Big Blind A standard hold ’em game showing the position of the blinds relative to the dealer button.Hold ’em is normally played using small and big blinds – forced bets by two players. Antes (forced contributions by all players) may be used in addition to blinds, particularly in later stages of tournaments. "

+ "<h2> A dealer “button” is used to represent the player in the dealer position; the dealer button rotates clockwise after each hand, changing the position of the dealer and blinds. The small blind is posted by the player to the left of the dealer and is usually equal to half of the big blind. The big blind, posted by the player to the left of the small blind, is equal to the minimum bet. "

+ "<h2> In tournament poker, the blind/ante structure periodically increases as the tournament progresses. After one round of betting is done, the next betting round will start by the person after the big blind and small blind.When only two players remain, special ‘head-to-head’ or ‘heads up’ rules are enforced and the blinds are posted differently. In this case, the person with the dealer button posts the small blind, "

+ "<h2> while his/her opponent places the big blind. The dealer acts first before the flop. After the flop, the dealer acts last and continues to do so for the remainder of the hand.</h2>"

+ "</p></html>");

//pane2.setText("<html><h2>Hello world</h2><h3>It's a beautiful day</h3></html>");

txtPanel.add(pane2);

JTextPane pane4 = new JTextPane();

pane4.setContentType("text/html");

pane4.setPreferredSize(new Dimension(0, 700));

pane4.setText("<html><p><h1><u>Play of the Hold’em hand</u></h1></p>"

+ "<p><h2>Each player is dealt two private cards in hold ’em, which are dealt first. Play begins with each player being dealt two cards face down, with the player in the small blind receiving the first card and the player in the button seat receiving the last card dealt. (As in most poker games, the deck is a standard 52-card deck containing no jokers.) These cards are the players’ hole or pocket cards. These are the only cards each player will receive individually, and they will only (possibly) be revealed at the showdown.</h2>"

+ "<h2>The poker hand begins with a “pre-flop” betting round, beginning with the player to the left of the big blind (or the player to the left of the dealer, if no blinds are used) and continuing clockwise. A round of betting continues until every player has folded, put in all of their chips, or matched the amount put in by all other active players. Note that the blinds in the pre-flop betting round are counted toward the amount that the blind player must contribute. If all players call around to the player in the big blind position, that player may either check or raise.</h2>"

+ "<h2>After the pre-flop betting round, assuming there remain at least two players taking part in the hand, the dealer deals a flop, three face-up community cards. The flop is followed by a second betting round. All betting rounds begin with the player to the button’s left and continue clockwise. After the flop betting round ends, a single community card (called the turn or fourth street) is dealt, followed by a third betting round. A final single community card (called the river or fifth street) is then dealt, followed by a fourth betting round and the showdown, if necessary</h2>"

+ "<h2>In all casinos, the dealer will “burn” a card before the flop, turn, and river. The burn occurs so players who are betting cannot see the back of the next community card to come. This is done for historical/traditional reasons, to avoid any possibility of a player knowing in advance the next card to be dealt.</h2></p></html>");

txtPanel.add(pane4);

JTextPane pane5 = new JTextPane();

pane5.setContentType("text/html");

pane5.setPreferredSize(new Dimension(0, 400));

pane5.setText("<html><p><h1><u>The Showdown</u></h1></p>"

+ "<p><h2>If a player bets and all other players fold, then the remaining player is awarded the pot and is not required to show his hole cards.</h2>"

+ "<h2>If two or more players remain after the final betting round, a showdown occurs. On the showdown, each player plays the best poker hand they can make from the seven cards comprising his two hole cards and the five community cards. A player may use both of his own two hole cards, only one, or none at all, to form his final five-card hand. If the five community cards form the player’s best hand, then the player is said to be playing the board and can only hope to split the pot, because each other player can also use the same five cards to construct the same hand.</h2></p></html>");

txtPanel.add(pane5);

JTextPane pane3 = new JTextPane();

pane3.setContentType("text/html");

pane3.setPreferredSize(new Dimension(0, 100));

pane3.setText("<html><p><h1><u>Hand's Types</u></h1></p></html>");

txtPanel.add(pane3);

BufferedImage myPicture = ImageIO.read(new File("src/img/hands.png"));

//Image resized = myPicture.getScaledInstance( 500, 500, Image.SCALE\_SMOOTH);

JLabel picLabel = new JLabel(new ImageIcon(myPicture));

txtPanel.add(picLabel);

JTextPane margin = new JTextPane();

margin.setContentType("text/html");

margin.setEditable(false);

margin.setPreferredSize(new Dimension(0, 100));

margin.setText("<html><p></p></html>");

txtPanel.add(margin);

previous = new JButton("Back to start page");

previous.setFont(new Font("Sherif", Font.ITALIC, 30));

previous.setForeground(Color.RED);

previous.setBorder(null);

//previous.setContentAreaFilled(false);

previous.setOpaque(false);

txtPanel.add(previous);

previous.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("guide");

frame.remove(p);

frame.repaint();

StartPage formPanel;

try {

frame.setSize(Main.START\_PAGE\_WIDTH, Main.START\_PAGE\_HEIGHT);

formPanel = new StartPage(frame);

frame.setTitle("Poker Start Page");

frame.add(formPanel);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

System.out.println("------ Start Page Loaded ------");

} catch (Exception e1) {

System.out.println("Error while loading start page.");

e1.printStackTrace();

}

}

});

add(txtPanel);

scrollBar = new JScrollPane(txtPanel);

scrollBar.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

scrollBar.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED);

scrollBar.getVerticalScrollBar().setUnitIncrement(16);

scrollBar.setBounds(0, 0, 790, 1000);

add(scrollBar);

javax.swing.SwingUtilities.invokeLater(new Runnable() {

public void run() {

scrollBar.getVerticalScrollBar().setValue(0);

}

});

}

}

***PokerGame***

package panels;

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Point;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.IOException;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.Timer;

import graphicsDrawImg.BackgroundImg;

import graphicsDrawImg.BigSmallBlinds;

import graphicsDrawImg.DealerGraphic;

import graphicsDrawImg.DrawBoardCards;

import graphicsDrawImg.DrawBurnCards;

import graphicsDrawImg.DrawChancesBar;

import graphicsDrawImg.DrawGrayPlayers;

import graphicsDrawImg.DrawPlayersCards;

import graphicsDrawImg.TimerCake;

import hashMapPack.CardsLocationsHashMap;

import hashMapPack.NameLblHashMap;

import hashMapPack.TimerCakeHashMap;

import javaPoker.Player;

import javaPoker.TexasHoldEm;

import other.Sound;

@SuppressWarnings("serial")

public class PokerGame extends JPanel {

public static final int BOARD\_CARD\_HEIGHT = 110;

public static final int BOARD\_CARD\_WIDTH = 75;

public static final int CARD\_WIDTH = 70;

public static final int CARD\_HEIGHT = 100;

public static final int SPACE\_BETWEEN\_DECK\_CARDS = 85;

private static final int NAME\_WIDTH = 200, NAME\_HEIGHT = 40;

private JFrame frame;

private JPanel pokerPanel;

public TexasHoldEm texasHoldEm;

public TimerCake t;

private DealerGraphic dealerIcons;

private DrawPlayersCards allPlayersCards;

private BigSmallBlinds bigSmallBlinds;

private BackgroundImg backgroundImg;

private DrawGrayPlayers drawGrayPlayers;

private DrawBurnCards drawBurnCards;

private DrawBoardCards drawBoardCards;

public DrawChancesBar drawChancesBar;

private StatusMoneyManager statusMoneyManager;

private ButtonsManager buttonsManager;

private TimerCakeHashMap timerCakeHashMap;

private NameLblHashMap nameLblHashMap;

private CardsLocationsHashMap greenRectHashMap;

private JLabel myNamelbl;

private JLabel namelbl1;

private JLabel namelbl2;

private JLabel namelbl3;

private JLabel namelbl4;

private JLabel namelbl5;

private JLabel winner;

private JLabel winnerHand;

public PokerGame(TexasHoldEm holdEm, JFrame frame) throws IOException {

setLayout(null);

this.frame = frame;

pokerPanel = this;

texasHoldEm = holdEm;

backgroundImg = new BackgroundImg();

timerCakeHashMap = new TimerCakeHashMap();

nameLblHashMap = new NameLblHashMap();

greenRectHashMap = new CardsLocationsHashMap();

drawGrayPlayers = new DrawGrayPlayers(texasHoldEm.getPlayer());

dealerIcons = new DealerGraphic();

allPlayersCards = new DrawPlayersCards(texasHoldEm);

bigSmallBlinds = new BigSmallBlinds();

drawBurnCards = new DrawBurnCards(texasHoldEm);

drawBoardCards = new DrawBoardCards(texasHoldEm);

drawChancesBar = new DrawChancesBar(texasHoldEm, this);

myNamelbl = new JLabel();

namelbl1 = new JLabel();

namelbl2 = new JLabel();

namelbl3 = new JLabel();

namelbl4 = new JLabel();

namelbl5 = new JLabel();

addPlayersNames();

statusMoneyManager = new StatusMoneyManager(this, texasHoldEm);

buttonsManager = new ButtonsManager(this, texasHoldEm);

buttonsManager.addChoiseButtons();

winner = new JLabel("", JLabel.CENTER);

winnerHand = new JLabel("", JLabel.CENTER);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Color backgroundColor = Color.WHITE;

setBackground(backgroundColor);

backgroundImg.draw(g);

drawGrayPlayers.draw(g);

dealerIcons.draw(g);

bigSmallBlinds.draw(g);

allPlayersCards.draw(g);

drawBurnCards.draw(g);

if (t != null && t.isTimerRunning()) {

addCurrentPlayerGreenMark(g, texasHoldEm.getCurrentPlayerIndex());

t.draw(g);

}

drawChancesBar.draw(g);

drawBoardCards.draw(g);

addPlayersCardsAndWinnerLabel(g);

}

private void addWinnerLbl() {

winner.setText(texasHoldEm.getWinner());

winner.setFont(new Font(Font.MONOSPACED, Font.BOLD, 30));

winner.setBounds(50, 650, frame.getWidth() - 100, 40);

changeWinnerAndWinnerHandLabels(winner);

winnerHand.setText("With: " + texasHoldEm.getWinnersHand());

winnerHand.setFont(new Font(Font.MONOSPACED, Font.BOLD, 20));

winnerHand.setBounds(50, 700, frame.getWidth() - 100, 40);

changeWinnerAndWinnerHandLabels(winnerHand);

}

private void changeWinnerAndWinnerHandLabels(JLabel label) {

Color foregroundColor = new Color(1, 147, 74);

Color backgroundColor = new Color(214, 237, 254);

label.setOpaque(true);

label.setForeground(foregroundColor);

label.setBackground(backgroundColor);

add(label);

}

public void addTimer(int playerIndex) {

int x = (int) timerCakeHashMap.getTimerCakeLocation(playerIndex).getX();

int y = (int) timerCakeHashMap.getTimerCakeLocation(playerIndex).getY();

if (t == null) {

t = new TimerCake(x, y, this, texasHoldEm.getPlayer()[playerIndex]);

t.startTimer();

} else {

t.setX(x);

t.setY(y);

t.reset(texasHoldEm.getPlayer()[playerIndex]);

}

}

private void addPlayersNames() {

for (int i = 0; i < TexasHoldEm.NUM\_PLAYERS; i++)

addNameOfPlayers(i);

}

private void addNameLabel(int x, int y, JLabel lbl, int i) {

lbl.setFont(new Font(Font.MONOSPACED, Font.BOLD, 25));

lbl.setForeground(Color.YELLOW);

lbl.setBounds(x, y, NAME\_WIDTH, NAME\_HEIGHT);

lbl.setText("<html><u>" + texasHoldEm.getPlayer()[i].getName() + "</html></u>");

add(lbl);

}

public void addNameOfPlayers(int playerIndex) {

int x = (int) nameLblHashMap.getNameLblLocation(playerIndex).getX();

int y = (int) nameLblHashMap.getNameLblLocation(playerIndex).getY();

switch (playerIndex) {

case 0:

addNameLabel(x, y, myNamelbl, playerIndex);

myNamelbl.setForeground(new Color(100, 149, 237));

break;

case 1:

addNameLabel(x, y, namelbl1, playerIndex);

break;

case 2:

addNameLabel(x, y, namelbl2, playerIndex);

break;

case 3:

addNameLabel(x, y, namelbl3, playerIndex);

break;

case 4:

addNameLabel(x, y, namelbl4, playerIndex);

break;

case 5:

addNameLabel(x, y, namelbl5, playerIndex);

break;

default:

break;

}

}

private void addPlayersCardsAndWinnerLabel(Graphics g) {

if (texasHoldEm.getRound() == 4) {

addWinnerLbl();

allPlayersCards.showPlayerCards(g);

}

}

private void addCurrentPlayerGreenMark(Graphics g, int i) {

Graphics2D g2 = (Graphics2D) g;

int thickness = 3;

g2.setStroke(new BasicStroke(thickness));

g2.setColor(Color.GREEN);

Point p1 = greenRectHashMap.getCardsLocation(i, 0);

Point p2 = greenRectHashMap.getCardsLocation(i, 1);

if (i == 2 || i == 5) {

g2.drawRect((int) p1.getX(), (int) p1.getY(), CARD\_HEIGHT, CARD\_WIDTH);

g2.drawRect((int) p2.getX(), (int) p2.getY(), CARD\_HEIGHT, CARD\_WIDTH);

} else {

g2.drawRect((int) p1.getX(), (int) p1.getY(), CARD\_WIDTH, CARD\_HEIGHT);

g2.drawRect((int) p2.getX(), (int) p2.getY(), CARD\_WIDTH, CARD\_HEIGHT);

}

}

public void newHand() {

JLabel newGame = new JLabel();

newGame.setFont(new Font(Font.MONOSPACED, Font.BOLD, 25));

newGame.setForeground(Color.WHITE);

newGame.setBounds(930, 705, 400, 100);

newGame.setVisible(true);

newGame.setBackground(Color.RED);

add(newGame);

if (texasHoldEm.getPlayer()[0].getMoney() > 0 && !texasHoldEm.isWholeGameOver()) {

newGame.setText("Starting new hand...");

System.out.println("Starting new hand...");

Timer t = new Timer(5000, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("New hand started.");

createNewPanel(texasHoldEm.getPlayer());

}

});

t.setRepeats(false);

t.start();

} else {

JButton newGameButton = null;

if (texasHoldEm.getWinnerPlayers().get(0).getId() == 0) {

newGame.setBounds(680, 705, 580, 100);

newGame.setText("Congratulations! You won the game!!");

try {

newGameButton = new JButton(new ImageIcon(buttonsManager.makePNGButtonsPath("button\_start-new-game")));

} catch (IOException e1) {

e1.printStackTrace();

}

newGameButton.setBounds(1050, 620, 200, 50);

} else {

try {

Thread.sleep(1200);

Sound.playSound("PlayerLoses");

} catch (IOException | InterruptedException e2) {

System.out.println("PlayerLoses sound not found.");

}

newGame.setText("You Lost !");

try {

newGameButton = new JButton(new ImageIcon(buttonsManager.makePNGButtonsPath("button\_new-game")));

} catch (IOException e1) {

e1.printStackTrace();

}

newGameButton.setBounds(1090, 725, 180, 50);

}

newGameButton.setVisible(true);

add(newGameButton);

newGameButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("New game started.");

createNewPanel(null);

}

});

}

}

private void createNewPanel(Player[] players) {

frame.remove(pokerPanel);

frame.repaint();

final TexasHoldEm texasHoldem;

try {

if (players != null)

texasHoldem = new TexasHoldEm(frame, false, players);

else

texasHoldem = new TexasHoldEm(frame, true, null);

frame.add(texasHoldem.getPokerGame());

frame.setVisible(true);

System.out.println("------ Game Loaded Again ------");

texasHoldem.getPokerGame().buttonsManager.setButtonsEnabled(false);

new Thread(new Runnable() {

@Override

public void run() {

try {

Thread.sleep(400);

} catch (InterruptedException e) {

e.printStackTrace();

}

texasHoldem.startPlay();

}

}).start();

} catch (Exception e1) {

System.out.println("Error while loading game page.");

e1.printStackTrace();

}

}

public StatusMoneyManager getStatusMoneyManager() {

return statusMoneyManager;

}

public ButtonsManager getButtonsManager() {

return buttonsManager;

}

}

package panels;

import java.awt.Color;

import java.awt.Component;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.image.BufferedImage;

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.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.Timer;

***StartPage***

import javax.swing.text.AttributeSet;

import javax.swing.text.BadLocationException;

import javax.swing.text.PlainDocument;

import graphicsDrawImg.ConfettiManager;

import hashMapPack.ImageBank;

import javaPoker.TexasHoldEm;

import other.Sound;

@SuppressWarnings("serial")

public class StartPage extends JPanel {

private static final int TIME\_BETWEEN\_SWITCH\_IMAGE = 7;

private static final int POKER\_GAME\_HEIGHT = 810;

private static final int POKER\_GAME\_WIDTH = 1300;

private static final int GUIDE\_PAGE\_WIDTH = 800;

private static final int NUMBER\_OF\_IMAGES = 3;

private static final int BUTTONS\_FONT = 18;

public static String name = "";

private final JFrame frame;

private JPanel p;

private JButton playButton, guide, soundOn, soundOff;

private JLabel addNameLbl, msgLbl;

private JTextField nickNameTxt;

private BufferedImage image;

private BufferedImage [] images = new BufferedImage[3];

private int numOfImg;

private ConfettiManager konfatiManager;

public StartPage(JFrame frame) throws IOException {

this.frame = frame;

p = this;

setLayout(null);

setBackground(Color.BLACK);

initaliseImages();

switchImages();

addAnimation();

createSoundButtons();

addNickName();

addButtons();

}

private void addAnimation() {

konfatiManager = new ConfettiManager(this);

Runnable runnable = new Runnable() {

@Override

public void run() {

while (true) {

konfatiManager.timePassed();

}

}

};

Thread animationThread = new Thread(runnable);

animationThread.start();

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

g.drawImage(image, 0, 0, frame.getWidth(), frame.getHeight() - 100, null);

g.drawImage(images[numOfImg], 0, 0, frame.getWidth(), 350, null);

konfatiManager.draw(g);

}

private void initaliseImages() throws IOException {

numOfImg = 0;

String path = "src/img/pokerStartPage.png";

image = ImageBank.getImage(path);

for (int i = 1; i <= NUMBER\_OF\_IMAGES; i++) {

path = "src/img/pic" + i + ".png";

images[i - 1] = ImageBank.getImage(path);

}

}

private void switchImages() {

Thread thread = new Thread(new Runnable() {

@Override

public void run() {

Timer timer = new Timer(TIME\_BETWEEN\_SWITCH\_IMAGE \* 100, new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

numOfImg = (numOfImg + 1) % NUMBER\_OF\_IMAGES;

repaint();

}

});

timer.setRepeats(true);

timer.start();

}

});

thread.start();

}

private void createSoundButtons() throws IOException {

soundOn = new JButton(new ImageIcon(ImageIO.read(new File("src/img/1.png"))));

soundOff = new JButton(new ImageIcon(ImageIO.read(new File("src/img/2.png"))));

changeSoundButtons(soundOn);

changeSoundButtons(soundOff);

soundOff.setVisible(false);

add(soundOn);

add(soundOff);

soundOn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

Sound.isSoundOn = false;

System.out.println("isSoundOn: " + Sound.isSoundOn);

soundOn.setVisible(false);

soundOff.setVisible(true);

}

});

soundOff.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

Sound.isSoundOn = true;

System.out.println("isSoundOn: " + Sound.isSoundOn);

soundOn.setVisible(true);

soundOff.setVisible(false);

}

});

}

private void changeSoundButtons(JButton sound) {

sound.setBorder(null);

sound.setContentAreaFilled(false);

sound.setOpaque(false);

sound.setBounds(40, 560, 50, 50);

}

private void addNickName() {

addNameLbl = new JLabel("<html><u>Enter your nickname:</html></u> ");

addNameLbl.setFont(new Font("Garamond", Font.BOLD, 25));

addNameLbl.setBounds(10, 500, 300, 30);

addNameLbl.setForeground(new Color(0, 200, 0));

add(addNameLbl);

nickNameTxt = new JTextField();

nickNameTxt.setFont(new Font("Garamond", Font.BOLD, 30));

nickNameTxt.setBounds(250, 495, 200, 40);

nickNameTxt.setForeground(new Color(230, 0, 0));

nickNameTxt.setOpaque(false);

nickNameTxt.setBorder(null);

nickNameTxt.setDocument(new JTextFieldLimit(12));

add(nickNameTxt);

msgLbl = new JLabel("(Enter 1 ~ 12 letters)");

msgLbl.setFont(new Font("Garamond", Font.BOLD, 15));

msgLbl.setBounds(50, 520, 300, 30);

msgLbl.setForeground(new Color(0, 200, 0));

add(msgLbl);

}

private void addButtons() {

playButton = new JButton("Play Now");

guide = new JButton("How to Play ?");

changeButtons(playButton);

changeButtons(guide);

playButton.setFont(new Font("verdana", Font.ITALIC, BUTTONS\_FONT));

playButton.setBounds(130, 560, 130, 50);

guide.setFont(new Font("verdana", Font.ITALIC, BUTTONS\_FONT));

guide.setBounds(270, 560, 180, 50);

add(playButton);

add(guide);

guide.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

System.out.println("guide");

frame.remove(p);

GuidePage guidePanel;

try {

guidePanel = new GuidePage(frame);

frame.add(guidePanel);

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

frame.setTitle("Guide - How To Play Texas Holde'm ?");

frame.setSize(GUIDE\_PAGE\_WIDTH, (int) screenSize.getHeight());

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

System.out.println("------ Guide Loaded ------");

} catch (Exception e1) {

System.out.println("Error while loading guide page.");

e1.printStackTrace();

}

}

});

playButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

name = nickNameTxt.getText();

if (!name.equals("")) {

frame.remove(p);

frame.repaint();

final TexasHoldEm texasHoldem;

try {

texasHoldem = new TexasHoldEm(frame, true, null);

frame.setTitle("Texas Hold'em Poker");

frame.add(texasHoldem.getPokerGame());

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(POKER\_GAME\_WIDTH, POKER\_GAME\_HEIGHT);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

System.out.println("------ Game Loaded ------");

texasHoldem.getPokerGame().getButtonsManager().setButtonsEnabled(false);

new Thread(new Runnable() {

@Override

public void run() {

try {

Thread.sleep(400);

} catch (InterruptedException e) {

e.printStackTrace();

}

texasHoldem.startPlay();

}

}).start();

} catch (Exception e1) {

System.out.println("Error while loading game page.");

e1.printStackTrace();

}

} else {

Runnable r = new Runnable() {

@Override

public void run() {

msgLbl.setForeground(Color.RED);

msgLbl.setFont(new Font("Garamond", Font.BOLD, 18));

Timer timer = new Timer(800, new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

msgLbl.setForeground(new Color(0, 200, 0));

msgLbl.setFont(new Font("Garamond", Font.BOLD, 15));

}

});

timer.setRepeats(false);

timer.start();

}

};

r.run();

}

}

});

}

private MouseAdapter createMouseOver() {

return new MouseAdapter() {

@Override

public void mouseEntered(MouseEvent e) {

((Component) e.getSource()).setFont(new Font("verdana", Font.BOLD, BUTTONS\_FONT));

super.mouseEntered(e);

}

@Override

public void mouseExited(MouseEvent e) {

((Component)e.getSource()).setFont(new Font("verdana", Font.ITALIC, BUTTONS\_FONT));

super.mouseExited(e);

}

};

}

private void changeButtons(JButton button) {

button.setForeground(Color.RED);

button.setFocusPainted(false);

button.setOpaque(false);

button.setContentAreaFilled(false);

button.setBorderPainted(false);

button.addMouseListener(createMouseOver());

}

}

/\*\*

\* Limits the JTextField's number of characters.

\* @author Naor

\*

\*/

@SuppressWarnings("serial")

class JTextFieldLimit extends PlainDocument {

private int limit;

JTextFieldLimit(int limit) {

super();

this.limit = limit;

}

public void insertString(int offset, String str, AttributeSet attr) throws BadLocationException {

if (str == null)

return;

if ((getLength() + str.length()) <= limit) {

super.insertString(offset, str, attr);

}

}

}

***StatusMoneyManager***

package panels;

import java.awt.Color;

import java.awt.Font;

import java.util.Arrays;

import java.util.List;

import javax.swing.JLabel;

import javax.swing.JPanel;

import hashMapPack.NameLblHashMap;

import javaPoker.TexasHoldEm;

import other.StaticFunctions;

public class StatusMoneyManager {

private static final int MONEY\_OFFSET\_FROM\_NAME\_LOCATION = 40;

private static final int STATUS\_OFFSET\_FROM\_NAME\_LOCATION = 50;

private static final Color CALL\_CHECK\_COLOR = new Color(0, 187, 243);

private static final Color ALL\_IN\_COLOR = new Color(0, 153, 0);

private static final int STATUS\_WIDTH = 160, STATUS\_HEIGHT = 40;

private static final int MONEY\_WIDTH = 400, MONEY\_HEIGHT = 100;

private NameLblHashMap nameLblHashMap;

private TexasHoldEm texasHoldEm;

private JPanel mainPanel;

public JLabel myStatuslbl;

private JLabel statuslbl1;

private JLabel statuslbl2;

private JLabel statuslbl3;

private JLabel statuslbl4;

private JLabel statuslbl5;

private List<JLabel> statusLabels;

private JLabel myMoneylbl;

private JLabel moneylbl1;

private JLabel moneylbl2;

private JLabel moneylbl3;

private JLabel moneylbl4;

private JLabel moneylbl5;

private List<JLabel> moneyLabels;

private JLabel sumTableMoney;

public StatusMoneyManager(JPanel mainPanel, TexasHoldEm texasHoldEm) {

this.mainPanel = mainPanel;

this.texasHoldEm = texasHoldEm;

nameLblHashMap = new NameLblHashMap();

addSumtableMoneylbl();

myMoneylbl = new JLabel();

moneylbl1 = new JLabel();

moneylbl2 = new JLabel();

moneylbl3 = new JLabel();

moneylbl4 = new JLabel();

moneylbl5 = new JLabel();

moneyLabels = Arrays.asList(myMoneylbl, moneylbl1, moneylbl2, moneylbl3, moneylbl4, moneylbl5);

addAllPlayersMoney();

myStatuslbl = new JLabel();

statuslbl1 = new JLabel();

statuslbl2 = new JLabel();

statuslbl3 = new JLabel();

statuslbl4 = new JLabel();

statuslbl5 = new JLabel();

statusLabels = Arrays.asList(myStatuslbl, statuslbl1, statuslbl2, statuslbl3, statuslbl4, statuslbl5);

}

private void addSumtableMoneylbl() {

sumTableMoney = new JLabel();

sumTableMoney.setFont(new Font(Font.MONOSPACED, Font.BOLD, 20));

sumTableMoney.setForeground(new Color(0, 153, 76));

sumTableMoney.setBounds(530, 0, 400, 60);

mainPanel.add(sumTableMoney);

}

public void updateSumBetOnTableLabel() {

sumTableMoney.setText("MONEY ON TABLE: " + texasHoldEm.getSumBetOnTable() + " $");

}

public void play(String choise, int bet, int playerIndex) {

addStatusOfPlayers(playerIndex, choise);

addMoneyLabelToPlayer(playerIndex);

updateStatusLabel(statusLabels.get(playerIndex), choise, bet);

}

public void addStatusOfPlayers(int i, String stringStat) {

int x = (int) nameLblHashMap.getNameLblLocation(i).getX();

int y = (int) nameLblHashMap.getNameLblLocation(i).getY() + STATUS\_OFFSET\_FROM\_NAME\_LOCATION;

addStatusOfPlayer(x, y, statusLabels.get(i), i, stringStat);

}

private void addStatusOfPlayer(int x, int y, JLabel lbl, int playerIndex, String stringStat) {

lbl.setFont(new Font(Font.SERIF, Font.BOLD, 20));

if (stringStat.equals("Raise"))

lbl.setForeground(Color.RED);

else if (stringStat.equals("ALL IN"))

lbl.setForeground(ALL\_IN\_COLOR);

else if (stringStat.equals("Call") || stringStat.equals("Check"))

lbl.setForeground(CALL\_CHECK\_COLOR);

else if (stringStat.equals("Fold"))

lbl.setForeground(Color.WHITE);

else

lbl.setForeground(Color.GRAY);

lbl.setBounds(x, y, STATUS\_WIDTH, STATUS\_HEIGHT);

lbl.setBackground(Color.WHITE);

lbl.setVisible(true);

mainPanel.add(lbl);

}

private void updateStatusLabel(JLabel lbl, String s, int x) {

if (!s.equals("Fold") && !s.equals("Out") && !s.equals("Check"))

lbl.setText(s + ": " + StaticFunctions.returnBigNumbersInShortcat(x) + " $");

else

lbl.setText(s);

}

public void addAllPlayersMoney() {

for (int i = 0; i < TexasHoldEm.NUM\_PLAYERS; i++) {

addMoneyLabelToPlayer(i);

}

}

public void setAllChangingPlayersStatusLabelsVisible(boolean visibility) {

for (JLabel statusLabel : statusLabels) {

if (statusLabel.getText().contains(TexasHoldEm.RAISE) || statusLabel.getText() == TexasHoldEm.CHECK

|| statusLabel.getText().contains(TexasHoldEm.CALL)) {

statusLabel.setVisible(visibility);

}

}

mainPanel.repaint();

}

public void addMoneyLabelToPlayer(int playerIndex) {

JLabel lbl = moneyLabels.get(playerIndex);

int x = (int) nameLblHashMap.getNameLblLocation(playerIndex).getX();

int y = (int) nameLblHashMap.getNameLblLocation(playerIndex).getY() + MONEY\_OFFSET\_FROM\_NAME\_LOCATION;

lbl.setFont(new Font(Font.MONOSPACED, Font.BOLD, 20));

lbl.setForeground(Color.YELLOW);

lbl.setBounds(x, y, MONEY\_WIDTH, MONEY\_HEIGHT);

lbl.setText("Money: " + StaticFunctions.returnBigNumbersInShortcat(texasHoldEm.getPlayer()[playerIndex].getMoney()) + " $");

mainPanel.add(lbl);

}

}

***Sound***

package other;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import sun.audio.AudioPlayer;

import sun.audio.AudioStream;

public abstract class Sound {

public static boolean isSoundOn = true;

public static void playSound(String soundName) throws IOException {

if (isSoundOn) {

String filepath = "src/sound/" + soundName + ".wav";

InputStream in = new FileInputStream(filepath);

AudioStream audioStream = new AudioStream(in);

AudioPlayer.player.start(audioStream);

}

}

public static void playSoundOfWinType(int winType) {

try {

switch (winType) {

case 1:

playSound("WinHighCard");

break;

case 2:

playSound("WinPair");

break;

case 3:

playSound("WinTwoPair");

break;

case 4:

playSound("WinThreeOfKind");

break;

case 5:

playSound("WinStraight");

break;

case 6:

playSound("WinFlush");

break;

case 7:

playSound("WinFullHouse");

break;

case 8:

playSound("WinFourOfKind");

break;

case 9:

playSound("WinStraightFlush");

break;

default:

break;

}

} catch (IOException e) {

System.out.println("Win sound of the winning type is not found.");

}

}

}

***StaticFunctions***

package other;

import java.awt.image.BufferedImage;

import panels.PokerGame;

public abstract class StaticFunctions {

private static final int BURN\_CARD\_ITERATION\_MILLIS = 2;

private static final int NORMAL\_CARD\_ITERATION\_MILLIS = 4;

public static boolean isNewRound = false;

public static BufferedImage cardToDraw = null;

public static int currentX, currentY;

public static String returnBigNumbersInShortcat(int num) {

if (num >= 10000 && num < 1000000) {

return String.valueOf(num / 1000) + "." + String.valueOf(num % 1000 / 100) + "K";

}

if (num >= 1000000 && num < 1000000000) {

return String.valueOf(num / 1000000) + "." + String.valueOf(num % 1000000 / 100000) + "M";

}

if (num >= 1000000000) {

return String.valueOf(num / 1000000000) + "." + String.valueOf(num % 1000000000 / 100000000) + "B";

}

return String.valueOf(num);

}

public static void moveCard(PokerGame pokerGame, int finalDestX, int finalDestY, boolean isBurn) {

if (isNewRound) {

while (currentX != (finalDestX) || currentY != finalDestY) {

if (currentX < finalDestX)

currentX++;

else if (currentX > finalDestX)

currentX --;

if (currentY < finalDestY)

currentY++;

else if (currentY > finalDestY)

currentY --;

try {

pokerGame.repaint();

if (isBurn)

Thread.sleep(BURN\_CARD\_ITERATION\_MILLIS);

else

Thread.sleep(NORMAL\_CARD\_ITERATION\_MILLIS);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

***Main***

package main;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

import panels.StartPage;

public class Main {

public static final int START\_PAGE\_HEIGHT = 650;

public static final int START\_PAGE\_WIDTH = 474;

public static void main(String[] args) throws IOException {

JFrame frame = new JFrame("Texas hold'em Start Page");

JPanel formPanel = new StartPage(frame);

frame.add(formPanel);

frame.setIconImage(ImageIO.read(new File("src/img/texas-holdem-logo.png")));

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(START\_PAGE\_WIDTH, START\_PAGE\_HEIGHT);

frame.setVisible(true);

frame.setResizable(false);

frame.setLocationRelativeTo(null);

System.out.println("Followed by the game's documentation:");

}

}