Card game 'Cambio'

Hodaya Tamano 313302978 Avital Israeli 315842773

Advisor: Dr. Dan Ophir

September 2019

1 introduction

In our project we will create an application of the card game "Cambio". We implemented the app in android studio based java.
We chose this topic because we are both interested in the gaming world. So we decided to look for an unknown card game and we decided Cambio game. When we implemented the game, we created an algorithm to computer vs. player.
This card game does not exist in the world of apps and computer games.
Our main goal is to prove that the computer wins the player in most cases.

2 Instructions

Basics:

'Cambio' is play with two players – In our case player vs. computer.
The goal is to have the least amount of points by the end of the game.
Each player is given one chance to check out the 2 cards closest to them and memorize them.
In order to peek your 2 close cards, the player need to press **short** click on the cards, and flip them back by pressing **short** click again.
The game starts by picking a card from the deck.
The player can replace the card he pulls with one of his own. This will reduce the number of points he has.
In order to swap between the cards, player need to press **long** click on the cards he wants to swap between them, then press on swap button.
If the player has completed his turn, he clicks the end turn button, and his card thrown to the garbage. now it's the computer turn and the player waits to his next turn.

Card Rules:

Certain cards have rules when you get them from the deck:
7 or 8 – Look at your own card - press **short** click on the card.
9 or 10 – Look at someone else's card - press **short** click on the cards.
Jack or Queen – Without looking at the card, you can blind switch one of your own with someone else's. press **long** click on the cards , then, press swap button.

Black King – You can look at anyone's card - press **short** click on the card - and switch anyone's card with any one of yours - press **long** click on the cards. If you want to swap between the cards ,press swap button.

Points:

*A*ce: 1 point
2-10: 2-10 points
*J*ack: 11 points
*Q*ueen: 12 points
*B*lack king: 13 points
*R*ed King: -1 point

Winning:

In order to win, one player has to call "Cambio" by pressing cambio, during their turn and cannot play a card if they call it.
Once the player calls Cambio, the rest of the players get one more turn to play before everyone flips their cards and adds up their points.
The person with the least amount of points wins. If there is a tie, the player who didn't call Cambio wins.

3 Classes

Card:

This class is building the cards.
Each card has the following values: value, type, and color.
Each of the cards also has a variable 'known' that initializes as false. As soon as the card is exposed to the computer it becomes true.
By toString() function we can access the images in the drawable folder.

```
public String toString(){
   //combine rank and suit together into a single string(ex: Ace of Diamonds)

   //using StringBuilder for modifiability later on
   StringBuilder displayCard = new StringBuilder();

   //personal choice to use switch
   switch(type){
      case 0:
         displayCard.append("spades");
         break;
      case 1:
         displayCard.append("hearts");
         break;
      case 2:
         displayCard.append("clubs");
         break;
      case 3:
         displayCard.append("diamonds");
         break;
      default: //anything else, do nothing
         break;
   }//end suit switch

   displayCard.append("of"); //setting the format of the output

   switch(value){
      //since rank is int type, now match int 11 to String jack...14 to Ace
      case 11:
         displayCard.append("jack");
         break;
      case 12:
         displayCard.append("queen");
         break;
      case 13:
         displayCard.append("king");
         break;
      case 1:
         displayCard.append("ace");
         break;
      default:
         displayCard.append(value); //number from 2 to 10 does not need to modify
         break;
   }//end rank switch

   //return the result of an entire combined string
   return displayCard.toString();
}
```

CardLocation:

This class saves for each card his position on the board. Each location has its own card, owner of the card and index(position).
We will use this class when we want to store in the computer memory the location of a card that is exposed on the board.

```
public class CardLocation {
   private Card card;
   private int index;
   private String owner; //which player the card belongs to.

   public CardLocation(){

   }

   public CardLocation(Card card, int index, String owner) {
      this.card = card;
      this.index = index;
      this.owner = owner;
   }

   @Override
   public String toString() {
      return "CardLocation [card=" + card + ", index=" + index + ", owner=" + owner +
"]";
   }
   public Card getCard() {
      return card;
   }
   public void setCard(Card card) {
      this.card = card;
   }
}
```

ConfigurationValue:

Definion of configuration value: For the computer and the player, we define another value called "configuration value". This value is the sum of the following parameters: x, y, z, w. Each parameter represents a numeric value of a card located in a specific location.
The numeric value of the parameters ranges is from 1 to 13 (Red King (-1)).
If the card is unknown the value of the card is 6 (the avarage).
We use this class when we want to check if it worth to the computer to swap between two cards.
In this class we have two functions:

1. before() – compute the configuration value of the computer or the player before swapping.

2. after() - compute the configuration value of the computer or the player after swapping.

If the configuration value after swapping is less then the configuration value before swap – the computer will do the swapping.

Computer:

In this class all the variables are related to the computer: computer cards array, computerMemory.
This class contains three functions:

1. computerTurn() - Executed all the algorithm behind the computer operations.

2. computerTurnRandom () - Executed an algorithm in which the computer does random actions in order to lower the difficulty of the game against the computer and to prove that the algorithm we have Implemented in computerTurn() is better.

3. My_Handler() – Helper function that handling the delay. We use it when we display the computer's operations to the player in our app.

The algorithmof the computer in computerTurn() function:

- If there is card in the garbage (during the game no in the beginning):
  - If it's worth swapping – the computer checks the configuration value before and after the swapping.
  - Else, if the computer didn't swap between the cards, he will take card from the deck.
    The computer will check if it's worth swapping with the card from the deck:

    - If it does – checks the configuration value before and after, and swap.
    - If it doesn't – use the power of the card (if exist).
  - Every step we enter the data into the computerMemory and cardLocation.

- If the computer received from the deck card with power:
  The computer won't swap and will use his power.
  - If the card is with swap power (Jack or Queen) and you know one of the player card, which is good card (less then 4):
    - Swap his card with your highest value and update the computerMemory and the cardLocation.
  - If the card is with peek power (7, 8, 9 or 10):
    - The computer use the flag known = true and add to the computerMemory.
  - If the card is ordinary less then 5:
    - The computer check all his known cards, if it worth swapping – swap and update computerMemory.
    - Else, if not worth swapping – throw to the garbage.
  - If the card is black King (extra power of swapping and peeking in one of the computer cards and one of the player cards):
    - If the card of the player if very good card (less then 3) – swap with unknown card (if all the cards are known – swap with the highest value).
    - Rest of the card values (more then 3) – check if it significant reducing with known card, if it doesn't swap with unknown card.

Player:

In this class we save array of the player cards.

Game:

This class linked everything in the game and contains 3 functions:

1. Start() – Executed the randomly dividing of the card into the cardDeck and 4 cards to the computer and to the player. After the division allows the computer peek his two cards (according to game instructions).

```
public static String start() {

   for(int x=0; x<4; x++){          //0-3 for type (4 types)
      for(int y=1; y<14; y++){     //2-14 for value (13 values)
         Card.cardDeck.add(new Card(x,y)); //create new card and add into the deck
      } //end value for
   }//end type for

   Collections.shuffle(Card.cardDeck, new Random()); //shuffle the deck randomly

   for (int i=0; i<4; i++) {//Dividing 4 cards for the players and remove from the
deck.

      Player.playerCards[i] = Card.cardDeck.get(i);
      Computer.computerCards[i] = Card.cardDeck.get(i+4);
   }

   // the computer peek two of his cards at the begining of the game.
   Computer.computerCards[0].setKnown(true);
   Computer.computerCards[1].setKnown(true);

   for (int i=0; i<8; i++) { // Removing the cards from the Deck.
      Card.cardDeck.remove(0);
   }
   // Define current card to be the first card of the deck,
   // set it as known card and remove it from the deck.
   currentCard = Card.cardDeck.get(0);

   // The computer adds his two cards which are close to him into the ArrayList.
   Computer.computerMemory.add(new
CardLocation(Computer.computerCards[0],0,"computer"));
   Computer.computerMemory.add(new
CardLocation(Computer.computerCards[1],1,"computer"));

   return winner;
}
```

2. theGame() – during the game, this function send us to other functions that implement the player or the computer turn. If the game is over we declared who is the winner or tie if no one won.
   Once we know who wins we update the appropriate field in Firebase. According to the new datum we are updating the computer's victories statistics from all games that have been up to now.

```
public static void theGame() {
    if (gameOn) {
        if (currentTurn.equals("computer")) {
            if (level == 1) {
                Computer.computerTurn();
            } else if (level == 0)//the easy level
                Computer.computerTurnRandom();
        } else
            MainActivity.playerTurn();
    } else { //game over
        Game.computer_sum = 0;
        Game.player_sum = 0;

        for (int i = 0; i < Computer.computerCards.length; i++) {
            if (Computer.computerCards[i].getValue() == 13 && Computer.computerCards[i].getColor().equals("black")) {
                computer_sum = computer_sum - 1;
            } else
                computer_sum = computer_sum + Computer.computerCards[i].getValue();
        }
        for (int i = 0; i < Player.playerCards.length; i++) {
            if (Player.playerCards[i].getValue() == 13 && Player.playerCards[i].getColor().equals("black")) {
                player_sum = player_sum - 1;
            } else
                player_sum = player_sum + Player.playerCards[i].getValue();
        }
        // Write a message to the database
        final FirebaseDatabase database = FirebaseDatabase.getInstance();
        if (computer_sum == player_sum) { //tie
            winner = "no one";
            if (level == 0) {
                Intent myIntent = new Intent(getContext(), EasyLevelStatistics.class);
                getContext().startActivity(myIntent);
            } else if (level == 1) {
                Intent myIntent = new Intent(getContext(), HardLevelStatistics.class);
                getContext().startActivity(myIntent);
            }
        } else if (computer_sum < player_sum) { //the computer win
            winner = "computer";
            if (level == 1) {
                final DatabaseReference myRef = database.getReference( s: "Hard level").child("Computer");
                myRef.addListenerForSingleValueEvent(new ValueEventListener() {...});
            } else if (level == 0) {
                final DatabaseReference myRef = database.getReference( s: "Easy level").child("Computer");
                myRef.addListenerForSingleValueEvent(new ValueEventListener() {...});
            }
        } else {
            winner = "player";
            if (level == 1) {
                final DatabaseReference myRef = database.getReference( s: "Hard level").child("Player");
                myRef.addListenerForSingleValueEvent(new ValueEventListener() {...});
            } else if (level == 0) {
                final DatabaseReference myRef = database.getReference( s: "Easy level").child("Player");
                myRef.addListenerForSingleValueEvent(new ValueEventListener() {...});
            }
        }
        showToastMethod(Game.getContext().getApplicationContext());
        Game.gameOn = false;
    }
}
```

3. showToastMethod() – helper function that display who is the winner.

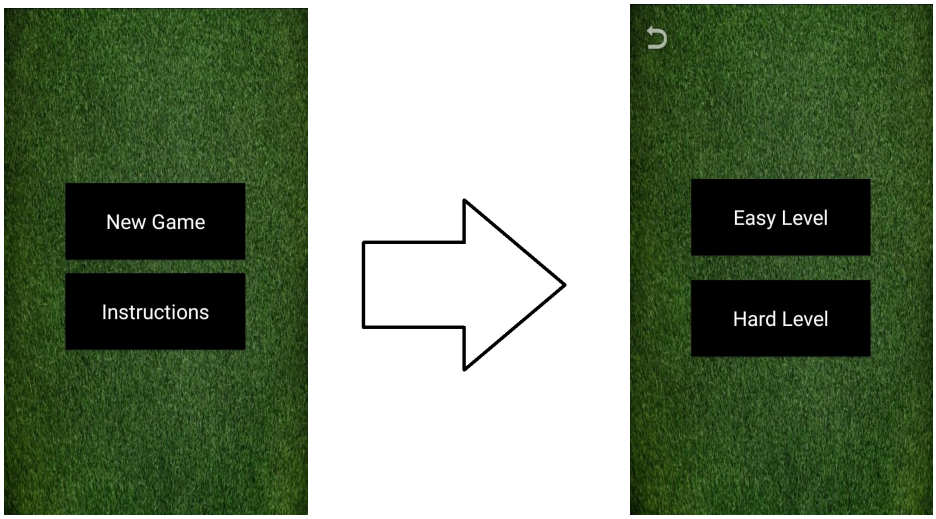
Menu:

This class responsible about linking the opening page in our app. In this function we listening to two buttons:

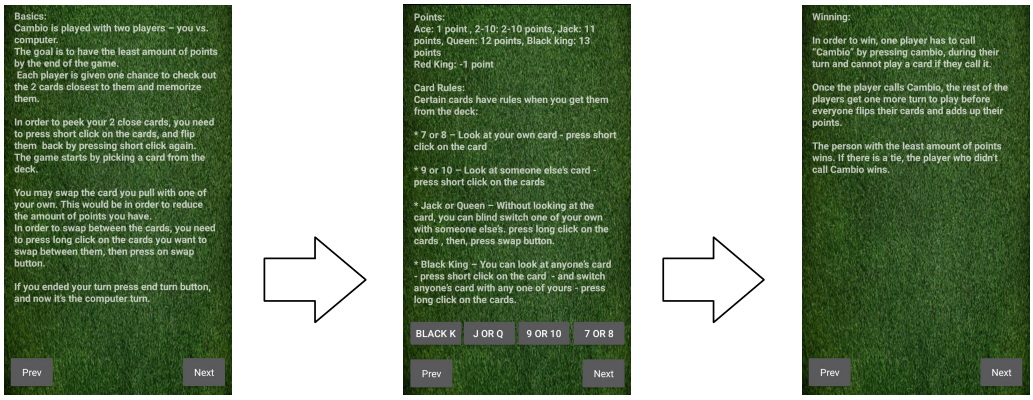
1. newGame – After pressing the button you will move to a new screen. The new screen has 2 buttons: 'Hard Level' and 'Easy Level'.

   - if you press 'Hard Level' you will play against the computer when the computer execute his operations according to the algorithm we implement in computerTurn() function.
   - if you press 'Easy Level' you will play against the computer when the computer execute his operations according to the algorithm we implement in computerTurnRandom() function.

   In the right corner there is a button that allows you to return to the main menu.



2. Instructions – after pressing you can read the instruction of the game in our app. For illustrative purposes, we have also added sample videos (on screen 2) that can be viewed before the game starts.



MainActivity:

In this class all the player operations are preformed.
Every card in the game is defined as image button that the player can click short press for peeking or long pressfor swapping the cards during the game when he allowed.
Other image buttons we have:

- cardDeck – by pressing it, the player can get new card which displays as the current.
- current – the current card.
- garbage – the card that off usage is thrown to the garbage.

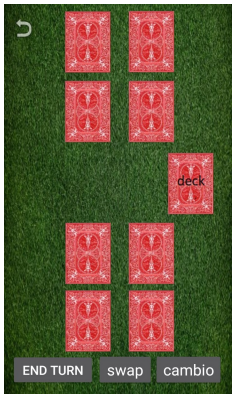
Buttons:

- swap – after choosing cards the player needs to press swap button in order to swap between the two cards.
- end turn – after the player finish his turn he needs to press end turn and now the computer is playing.
- menu – if the player wants to return to the menu he can press on to exit button on the left top of the screen.

- cambio – if the player thinks that he is the winner he will press the cambio button on his turn and after the computer do his turn, the real winner is declared.

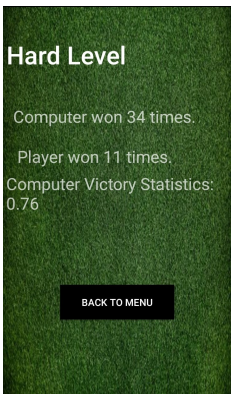
This class contains seven functions:

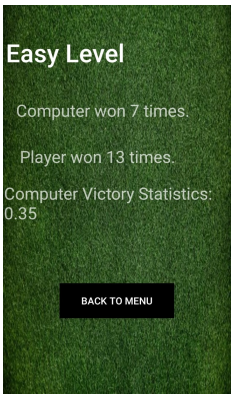
1. getContext() – helper function.
2. getGarbage() – helper function.
3. playerTurn() – according to the card that the player received from the deck we will let him to do the permissible operations (peek or swap) on the computer cards or on his cards.
4. flagIsFalse() – helper function that helps us to know If the player use the power of the card and he peeked or swapped. we use flags in order to do this.
5. swap() – this function handle the three situations:
   - Swapping between player card and computer card.
   - Swapping between player card and current card.
   - Swapping between player card and garbage card.
6. chooseYourCard() – this function will display to the player the cards he chose to swap between them.
7. peekYourCard() – this function will display to the player the card he peeked.



EasyLevelStatistics and HardLevelStatistics:

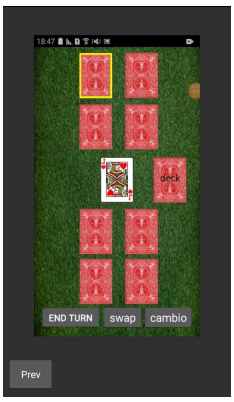
This class displays to the player the statistics of the game on the easy/ hard level, using the data on the Firebase. In addition, we have the option to return to the menu and start to play another game.





Videos:

In this class we display to the player videos samples of operations when the player received a card with power.
In the Instructions on the app the player has four buttons for each card power, by pressing the buttons the player can watch how to do this operation.



4 Firebase

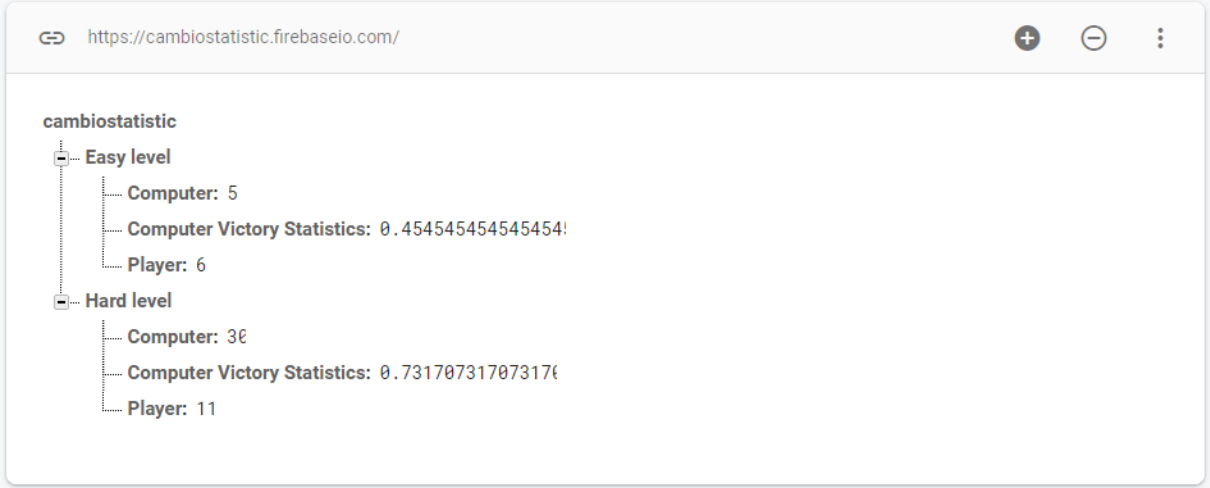
In our application we used Firebase DB.
in the Firebase project 'cambiostatistics' we have separated the data according to the difficulty level of the game we are playing.

- Easy level is the level which the player plays against the computer, while the computer does random operations with the cards on his turn in a non-strategic way. Using the function computerTurnRandom().
- Hard level is the level which the player plays against the computer, while the computer does his operations using the function computerTurn().

On the DB we have displayed the amount of computer victories, the amount of player victories and the statistics of computer victories against the player which will be calculated by the formula:

Computer Victory Statistics = computerWin / (computerWin + playerWin)

The goal: to help us to prove that when we use the algorithm we implemented in computerTurn() function in Hard level, the computer wins at least 50% of the cases. In Easy level the player have higher chances to win, and the computer wins in less the 50% of the cases.



The code to insert the data to the Firebase:

```
else if (computer_sum < player_sum) { //the computer win
  winner = "computer";
  if (level == 1) {
      final DatabaseReference myRef = database.getReference( s: "Hard level").child("Computer");
      myRef.addListenerForSingleValueEvent(new ValueEventListener() {
          @Override
          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
              if (dataSnapshot.exists()) {

                  computerWins = (long) dataSnapshot.getValue();
                  computerWins++;
                  myRef.setValue(computerWins);

                  database.getReference( s: "Hard level").child("Player").addListenerForSingleValueEvent(new ValueEventListener() {
                      @Override
                      public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                          if (dataSnapshot.exists()) {
                              playerWins = (long) dataSnapshot.getValue();
                              String statistic = String.format("%.2f", (double) computerWins / (computerWins + playerWins));
                              database.getReference( s: "Hard level").child("Computer Victory Statistics").setValue(statistic);
                              Intent myIntent = new Intent(getContext(), HardLevelStatistics.class);
                              getContext().startActivity(myIntent);
                          }
                      }
                  });
              }
          }
```

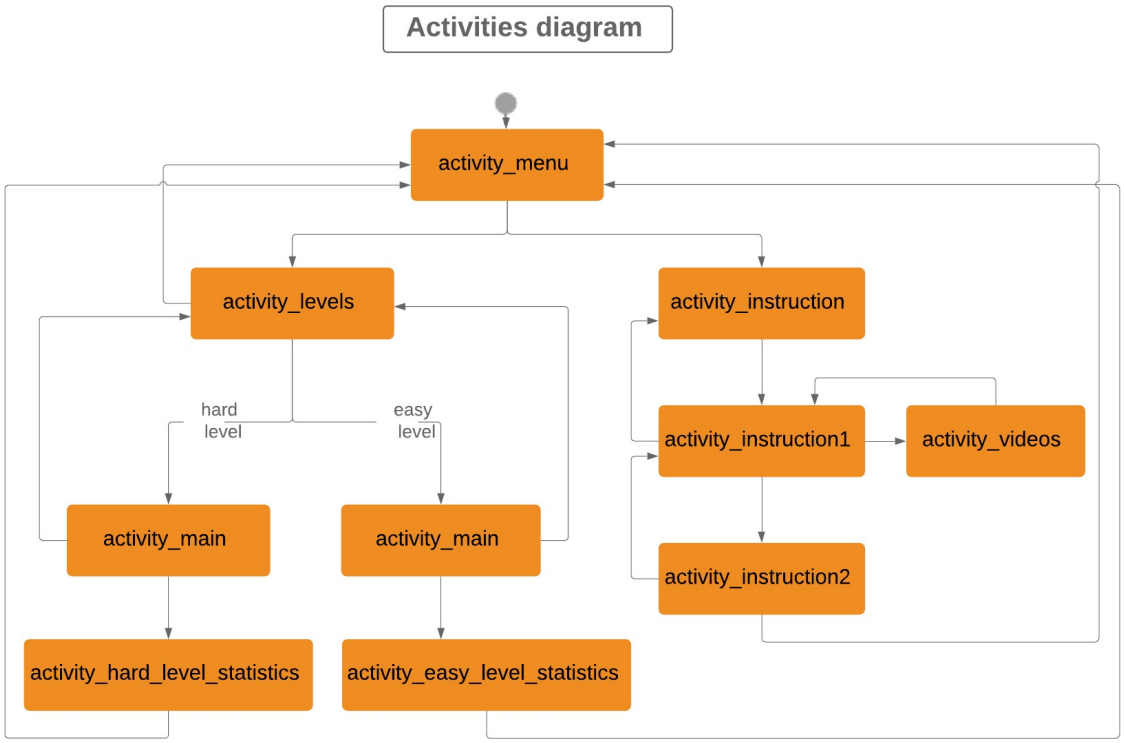
5 Conclusion

Our goal in the project is to achieve 50% of victories to the computer. As you can see in the last activity of the statistics, we displays the results and computer victories statistics.
There are situations where the player wins the computer due to the element of luck (the card deck is dividing randomly and the player gets the cards with the lowest sum), but we still achieved our goal.
Link to github:
https://github.com/HodayaTamano/CambioFinalProject

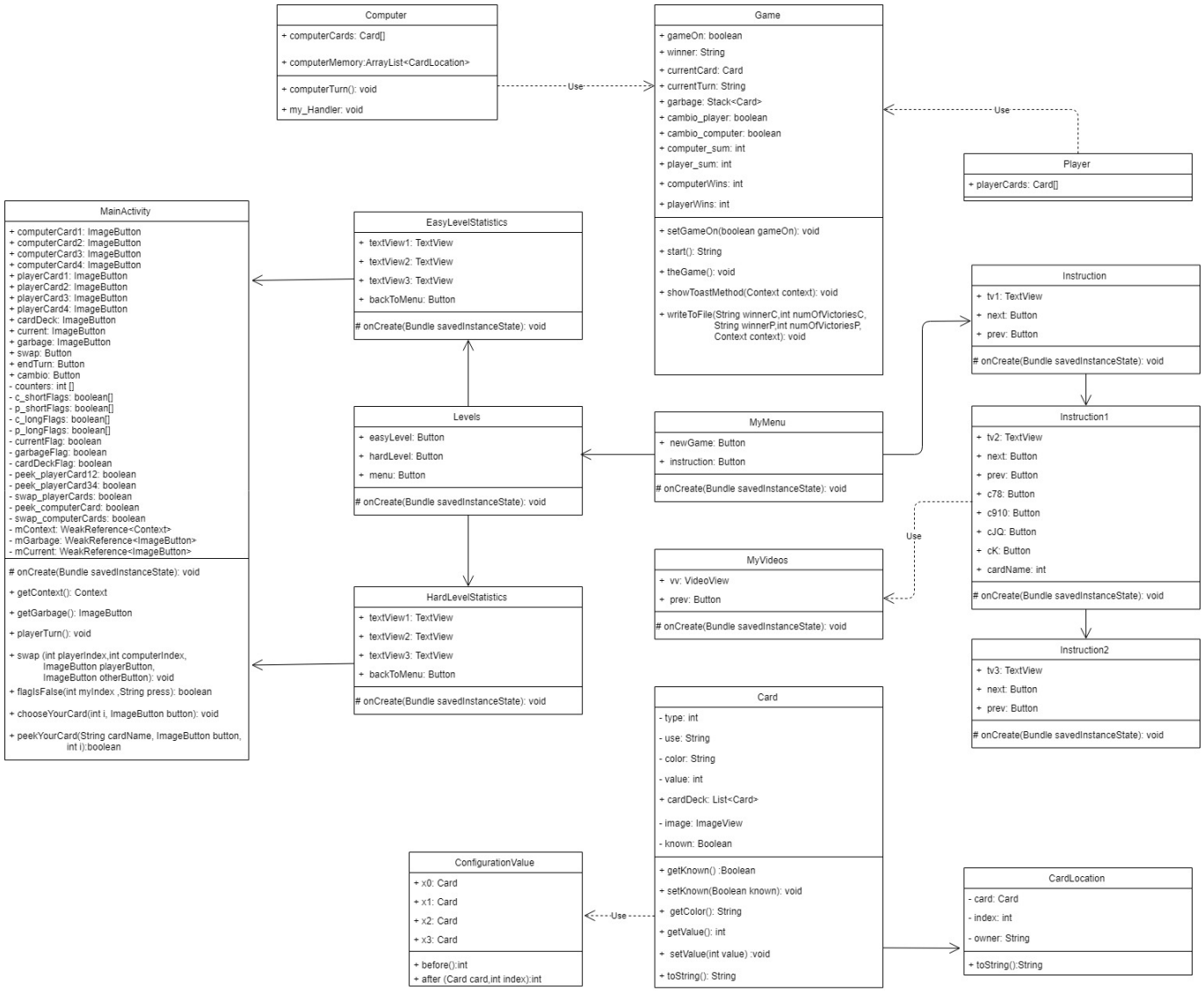
6 Diagrams

Activities diagram:

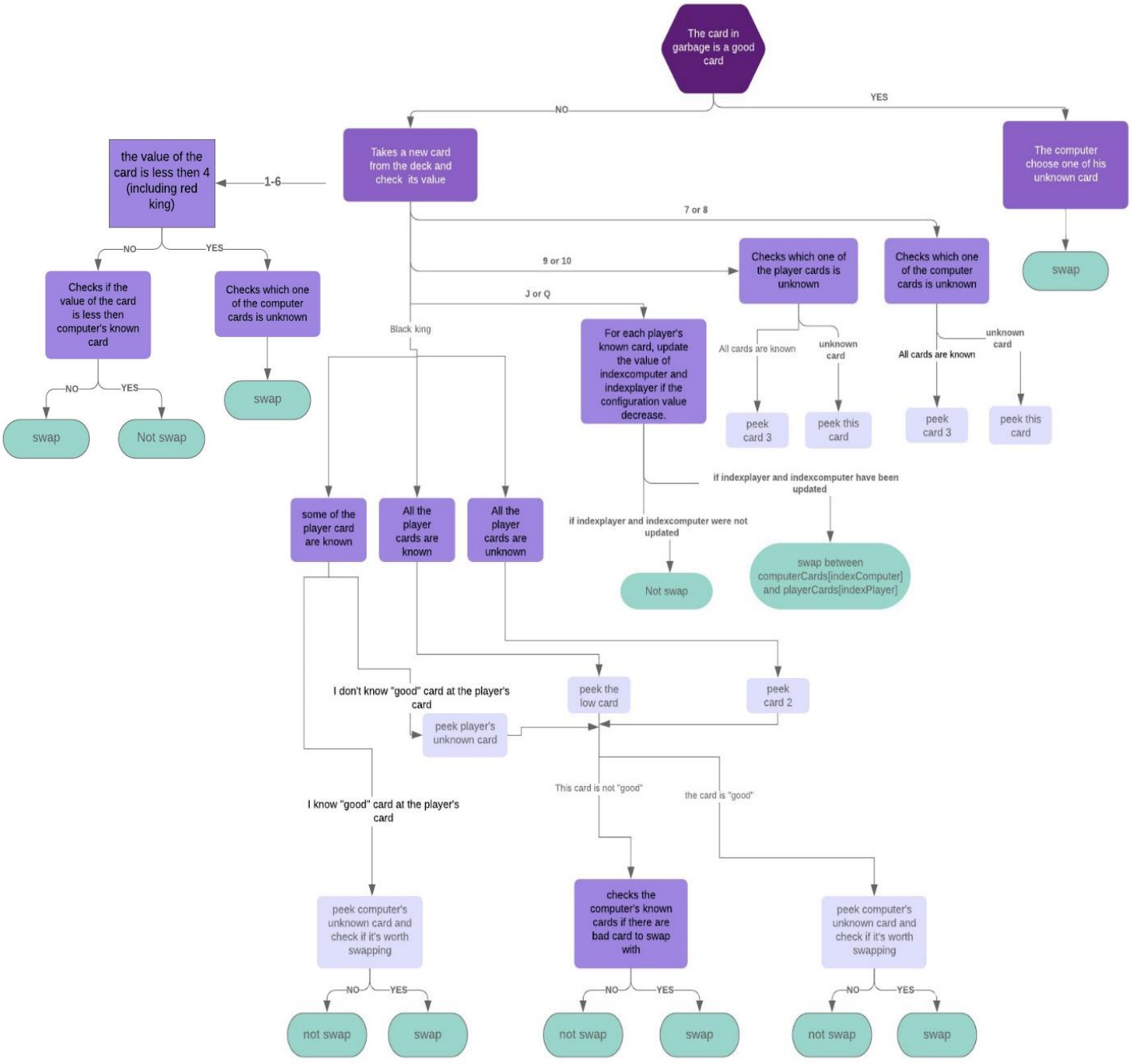
This diagram shows the transitions between the activities in the app.
At first, the player chooses whether to start the game or read the instructions.
If he chooses to read the instructions he goes through 3 screens including a screen with videos demonstrating the moves in the game.
If he chooses to start a new game, he moves to a screen where he has to choose a level of play.
After he chooses the level, he moves into the game.
Once there is a winner in the game (either the player or the computer) it automatically switches to a screen that shows the computer's winning stats by now.



UML diagram:


Computer
+ computerCards: Card[]
+ computerMemory:ArrayList<CardLocation>
+ computerTurn(): void
+ my_Handler: void
Use
Game
+ gameOn: boolean
+ winner: String
+ currentCard: Card
+ currentTurn: String
+ garbage: Stack<Card>
+ cambio_player: boolean
+ cambio_computer: boolean
+ computer_sum: int
+ player_sum: int
+ computerWins: int
+ playerWins: int
+ setGameOn(boolean gameOn): void
+ start(): String
+ theGame(): void
+ showToastMethod(Context context): void
+ writeToFile(String winnerC,int numOfVictoriesC, String winnerP,int numOfVictoriesP, Context context): void
Use
Player
+ playerCards: Card[]
MainActivity
+ computerCard1: ImageButton
+ computerCard2: ImageButton
+ computerCard3: ImageButton
+ computerCard4: ImageButton
+ playerCard1: ImageButton
+ playerCard2: ImageButton
+ playerCard3: ImageButton
+ playerCard4: ImageButton
+ cardDeck: ImageButton
+ current: ImageButton
+ garbage: ImageButton
+ swap: Button
+ endTurn: Button
+ cambio: Button
- counters: int []
- c_shortFlags: boolean[]
- p_shortFlags: boolean[]
- c_longFlags: boolean[]
- p_longFlags: boolean[]
- currentFlag: boolean
- garbageFlag: boolean
- cardDeckFlag: boolean
- peek_playerCard12: boolean
- peek_playerCard34: boolean
- swap_playerCards: boolean
- peek_computerCard: boolean
- swap_computerCards: boolean
- mContext: WeakReference<Context>
- mGarbage: WeakReference<ImageButton>
- mCurrent: WeakReference<ImageButton>
onCreate(Bundle savedInstanceState): void
+ getContext(): Context
+ getGarbage(): ImageButton
+ playerTurn(): void
+ swap (int playerIndex,int computerIndex, ImageButton playerButton, ImageButton otherButton): void
+ flagIsFalse(int myIndex ,String press): boolean
+ chooseYourCard(int i, ImageButton button): void
+ peekYourCard(String cardName, ImageButton button, int i):boolean
EasyLevelStatistics
+ textView1: TextView
+ textView2: TextView
+ textView3: TextView
+ backToMenu: Button
# onCreate(Bundle savedInstanceState): void
Levels
+ easyLevel: Button
+ hardLevel: Button
+ menu: Button
# onCreate(Bundle savedInstanceState): void
HardLevelStatistics
+ textView1: TextView
+ textView2: TextView
+ textView3: TextView
+ backToMenu: Button
# onCreate(Bundle savedInstanceState): void
MyMenu
+ newGame: Button
+ instruction: Button
# onCreate(Bundle savedInstanceState): void
Use
MyVideos
+ vv: VideoView
+ prev: Button
# onCreate(Bundle savedInstanceState): void
Instruction
+ tv1: TextView
+ next: Button
+ prev: Button
# onCreate(Bundle savedInstanceState): void
Instruction1
+ tv2: TextView
+ next: Button
+ prev: Button
+ c78: Button
+ c910: Button
+ cJQ: Button
+ cK: Button
+ cardName: int
# onCreate(Bundle savedInstanceState): void
Instruction2
+ tv3: TextView
+ next: Button
+ prev: Button
# onCreate(Bundle savedInstanceState): void
Card
- type: int
- use: String
- color: String
- value: int
+ cardDeck: List<Card>
- image: ImageView
- known: Boolean
+ getKnown() :Boolean
+ setKnown(Boolean known): void
+ getColor(): String
+ getValue(): int
+ setValue(int value) :void
+ toString(): String
ConfigurationValue
+ x0: Card
+ x1: Card
+ x2: Card
+ x3: Card
+ before():int
+ after (Card card,int index):int
Use
CardLocation
- card: Card
- index: int
- owner: String
+ toString():String

Decision tree- Computer turn:



7 Resources

1. https://joshaguirre.com
   This website has game instructions.

2. https://www.raywenderlich.com.
   At the beginning of the project we looked at this website and learned how to create a card game

3. https://firebase.google.com
   We learned from this website how to connect the Android to Firebase.

4. Stack Overflow.
   Every problem we had with Android Studio was helped by the stack overflow website.