MD. Asafuddaula Sobahani Shounak

JU CSE 45 batch  Roll : 2171

Teen PAtti

**Project Name:** **Teen Patti**

## **Description**

**Teen patti** (तीन पत्ती, meaning 'three cards' in English) is a gambling card game that originated in the Indian subcontinent and is popular throughout South Asia. It originated in the English game of three-card brag, with influences from poker. It is also called flush or flash in some areas.

## **Game starts**

The game starts with one of the players dealing the cards. The cards are usually dealt counter-clockwise.

Before starting the game, usually an agreed number of cards are picked or dealt to decide the dealer for the opening hand. Each player may be required to put up an ante into the pot before picking the cards. The winning player gets the pot. The relative rankings of the cards may also decide the seating arrangement for each player. This entire process is called *cut-for-seat*. After the opening hand, the winner of any hand will be the dealer for the next hand.

## **Betting**

There is usually an ante or boot amount put on the table (the pot). The betting then starts by the player next to the dealer.

### **Loose versus tight play**

Loose and tight in *teen patti* refer to a player's general tendency to play hands beyond the first round or to fold them quickly. There is no commonly accepted threshold in terms of a ratio or percentage of hands played, but a "tight" player will often choose to fold weaker hands, while a "loose" player will bet on more of these hands and thus play more hands to the show/showdown.

### **Entry fee**

There is usually an ante or boot amount put on the table (the pot). This ante may be in the form of an equal amount put by each player, or a single larger amount put by one player (usually on a rotation basis).

An ante is a forced bet in which each player places an equal amount of money or chips into the pot before the deal begins. Often this is either a single unit (a one-value or the smallest value in play) or some other small amount; a percentage such as half or one-quarter of the minimum bet is also common. An ante paid by each player ensures that a player who folds every round will lose money (though slowly), thus providing each player with an incentive, however small, to play the hand rather than toss it in when the opening bet reaches them.

Antes discourage extremely tight play. Without the ante, a player who has not paid a blind can toss in his hand at no cost to him; the ante ensures that doing so too often is a losing proposition. With antes, more players stay in the hand, which increases pot size and makes for more interesting play.

In games where the acting dealer changes each turn, it is not uncommon for the players to agree that the dealer (or some other position relative to the dealer or the button) provides the ante for each player. This simplifies betting, but causes minor inequities if other players come and go. During such times, the player can be given a special button indicating the need to pay an ante to the pot (known as "posting") upon their return.

### **Post**

A player who is temporarily away from his seat (i.e. for drinks or a restroom/bathroom break) and misses antes is also required to post to re-enter the game. They must pay the applicable ante to the pot for the next hand they will participate in. In this case, the amount to be posted is the amount of the ante at the time the player missed them.

Posting is usually not required if the player who would otherwise post happens to be in the ante. This is because the advantage that would otherwise be gained by missing the ante, that of playing several hands before having to pay ante, is not the case in this situation. It is therefore common for a new player to lock up a seat and then wait several hands before joining a table, or for a returning player to sit out several hands until the ante comes back around, so that he may enter in the ante and avoid paying the post. For this same reason, only one ante can be accumulated by the player; old missed antes are removed when the ante returns to that player's seat because the player was never in any position to gain from missing the antes.

### **Blind**

A blind or blind bet is usually a forced bet placed into the pot by one or more players before the deal begins, in a way that simulates bets made during play. The most common use of blinds as a betting structure calls for two blinds: the player after the dealer blinds about half of what would be a normal bet (small blind), and the next player blinds what would be a whole bet (big blind). This two-blind structure, sometimes with antes, is the dominating structure of play. Sometimes only one blind (half or whole bet) is used (often informally by the dealer as a "price of winning" the previous hand).

### **Call and raise (*chaal*)**

After the ante and the forced blind bet(s), the regular betting starts with the next player putting his bet amount to the pot.

The total bet can be divided into two components - the **call** and the **raise**, both being usually called as Chaal. Each player has to place a bet that is at least equal to the previous player's bet, with the option of raising the bet. This bet then becomes the (new) current level of bet (Chaal). Usually there is a limit imposed to the raise, such that the total bet amount (with the call and the raise) cannot exceed twice that of the previous player's bet. Also, the bet should be in even amounts (2,4,6,8..), especially when there is one player still in the game as Blind player. It is because the player playing blind plays half of the normal bet, and odd amounts can't be halved.

This is very important to understand that this betting structure is different than Poker, as every time the bet is new (disregarding of how much amount was previously bid.) Let's say, one player bet an amount of 2 and second player raises it up to 4, now the first person would have to put additional 4 into the pot to make a call or would have an option to raise the bet up to 8. (Which is different in case of Poker where the second player adds 2 more to bring his bet to the level 4.)

## **Limits**

Betting limits apply to the amount a player may **bet** or **raise**, and come in four common forms: *fixed limit*, *spread limit*, *pot limit*, and *no limit*.

**Note**: The limits may be applied to the **raise** or the**bet**.

All such games have a minimum bet as well as the stated maximums, and also commonly a **betting unit**, which is the smallest denomination in which bets can be made.

### **Fixed**

In a game played with a **fixed-limit** betting structure, the maximum raise amount generally equals the last bet amount (current level of bet). Thus, the maximum raise amount is equal to the call amount, and the maximum bet is double that of the last bet.

### **Spread**

A game played with a **spread-limit** betting structure allows a player to raise any amount within a specified maximum (subject to other betting rules).

For example, a game with a "$1000 spread-limit" allows each player to call and then raise up to a maximum of $1000.

A simpler approach is to bet up to a maximum amount equal to the spread-limit. So in the above example, the maximum bet (call + raise) is limited to $1000.

### **Pot**

A game played with a **pot-limit** betting structure allows any player to raise up to an amount equal to the size of the whole pot before the raise.

For example, let us assume that there is an ante of $50 and a single forced blind bet of $10 in the pot at the start of the regular betting round, and all subsequent players are seen/have seen their cards. The next player may bet the minimum amount of $20 (double that of the last blind bet of $10). This player may also raise up to $80 (as there is now $50 + $10 + $20 = $80 in the pot) to bring his total bet (also the new current level of bet) to $20 + $80 = $100. If he does in fact bets $100, the total amount of the pot is $160, and the next player may call (put the minimum bet of) $100, and raise up to $260 for a total bet of $360 (after calling the $100 bet, the total amount of the pot is $260, so he may raise up to $260). The next player would then be entitled to call at $360 and raise up to $880 (after calling $360, the pot would contain $880, thus he may raise $880).

A simpler approach is to bet up to a maximum amount in the pot. So in the above example, the first seen player may bet a minimum of $20 and a maximum of $60. If he does bet the maximum, the next player may bet a minimum of $60 to a maximum of $120. If he bets the maximum, the next player would then be entitled to bet a minimum of $120 and a maximum of $240. **Note**: In this example, for the later bets, the maximum bet is twice that of minimum bet because of the chosen ante and forced blind bet amounts. For other combinations, this relationship may not hold.

### **No-limit**

A game played with a **no-limit** betting structure allows each player to raise the bet by any amount.

## **Moves**

### **Blind play**

When somebody plays blind, he has not seen his cards and places his bet by guessing the strength of his card combination and of other players. He should place a higher bet if he has faith in his luck and his card combination will be stronger than other players. If somebody wants to play it safe and not take risks, he should place bet for a smaller amount.

### **Play (*chaal* phase)**

Anybody can choose to see their cards at any time and then place a bet (when it's their turn) depending on how strong he thinks his card combination is. He can pack if he thinks his card combination is not strong enough. Though he can pack out of turn, it's not considered courteous to do so. There is a variation to this rule in which players are only allowed to see and fold their cards when it's their turn.

### **Sideshow, backshow, and compromise**

If somebody have seen his cards, he can ask for a sideshow after placing his bet. He can ask for a sideshow with the previous player (who placed the last bet). Hence sideshows are also referred to as backshows. Sideshows are only permitted if no player is playing and if the player request for the third time no one can deny it and to whom it been requested the player have to show the cards

The player can either accept or decline others sideshow request. Player may want to decline sideshow if one is trying to bluff way through, in that case one is likely to fold due to poor hand when two hands are compared. Player may also want to decline sideshow request if one is holding a strong hand and/or would want pot money to increase to make game more interesting.

If the sideshow request is accepted, the two players involved privately compare their cards, and the player with the lower hand is forced to fold. If the hands are equal, the player who asked for the sideshow is forced to fold.

If the sideshow is refused, the player requesting the sideshow must bet to stay in the game or fold.[[3]](https://en.wikipedia.org/wiki/Teen_patti#cite_note-3)

### **Tie**

At any point during the game, if both players are not want to show/Back out his card then game will be consider as a cancelled. And total money which is invested in the game that will be return/share between both players. There have not any situation to win the game by any player.

### **Show**

The betting will go on like this until one of two things happens:

* All but one player pack. The sole remaining player wins the money in the pot, regardless of what cards he or she holds.
* All but two players pack. In this case, during his or her turn, one player pays for a "show". At this point, both players' cards are exposed and compared and the higher-ranking hand wins the pot.

The following are the rules for a "show":

* Seen player can ask for a show with blind player for four times the current bet.
* If both players are seen then either may ask for a show by paying two times the current bet.

Note: A blind player cannot ask for a show or sideshow.

In most modern games, especially where the stakes or bets are high, shows are paid for at the current bet level. Also, while a blind player cannot ask (nor can be asked) for a sideshow, the blind player can ask for a show. For a show, a player can choose not to show, if they give up the pot. Ultimately, all rules are discussed and agreed by all players before the game begins.

## **Ranking of hands**

Trio (trail)

[6 of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_6.svg)[6 of diamonds](https://en.wikipedia.org/wiki/File:Playing_card_diamond_6.svg)[6 of hearts](https://en.wikipedia.org/wiki/File:Playing_card_heart_6.svg)

Straight flush (pure sequence)

[Jack of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_J.svg)[10 of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_10.svg)[9 of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_9.svg)

Straight (sequence)

[9 of spades](https://en.wikipedia.org/wiki/File:Playing_card_spade_9.svg)[8 of diamonds](https://en.wikipedia.org/wiki/File:Playing_card_diamond_8.svg)[7 of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_7.svg)

Flush (colour)

[Queen of spades](https://en.wikipedia.org/wiki/File:Playing_card_spade_Q.svg)[10 of spades](https://en.wikipedia.org/wiki/File:Playing_card_spade_10.svg)[6 of spades](https://en.wikipedia.org/wiki/File:Playing_card_spade_6.svg)

Pair

[2 of diamonds](https://en.wikipedia.org/wiki/File:Playing_card_diamond_2.svg)[2 of hearts](https://en.wikipedia.org/wiki/File:Playing_card_heart_2.svg)[Queen of hearts](https://en.wikipedia.org/wiki/File:Playing_card_heart_Q.svg)

High card

[Ace of hearts](https://en.wikipedia.org/wiki/File:Playing_card_heart_A.svg)[Jack of clubs](https://en.wikipedia.org/wiki/File:Playing_card_club_J.svg)[10 of hearts](https://en.wikipedia.org/wiki/File:Playing_card_heart_10.svg)

Examples of poker hand categories in descending order

There are two ways *teen patti* is played. One is by using a standard 52-card deck and another is by adding the two Joker cards, which are used as wild cards. The object of the game is to have the best three-card hand and to maximize the pot before the showdown. Any hand of a higher category beats any hand of a lower category. If two players have the same combination then the pot is split between the two no matter which of the two players asked for showdown. In another variation, where the suits hold ranks, the player with the higher suit wins the pot.

The categories are ranked as follows:

**Three of a kind (trio)**

Three of the same cards. Three aces are the highest and three twos are the lowest trio.

**Straight flush (pure sequence)**

Three consecutive cards of the same suit.

The order of ranking from highest (defined by highest card in the sequence) to lowest is: A-K-Q, K-Q-J, Q-J-10, and so on down to 4-3-2. A wraparound (K-A-2) is not considered a straight flush, but is a valid flush.

**Straight (sequence)**

Three consecutive cards not all in the same suit. A straight is also referred as a round or sequence.

The highest to lowest ranking is (as with straight flushes): A-K-Q, A-2-3, K-Q-J, Q-J-10 and so on down to 4-3-2.[[4]](https://en.wikipedia.org/wiki/Teen_patti#cite_note-rrrn-4) As with straight flushes, K-A-2 is not a valid hand.

**Flush (colour)**

All the 3 cards are of same suit. If two players both have flushes, the player with the high card wins; if they match, then the next highest card is compared, then the third card if needed. If two players have the same card values, then the hands are ranked by suit, with spades first and clubs last.

**Pair (double)**

Two cards of the same rank. Between two pairs, the one with the higher value is the winner. If the pairs are of equal value, the value of the third card decides the winner. Therefore the lowest pair is 2-2-3 and the highest is A-A-K.

**No pair (high card)**

If two players share a common high card, then rest of the cards are compared based upon their values.

### Probabilities

The probabilities of the various ranking combinations are described below. All these probabilities are described for 52-card *teen patti*, without the two Joker cards. In Joker versions, the probabilities change widely, most importantly for pairs.

There are 52 cards in the deck. The sequence in which the cards are dealt does not matter, it is the combination of the three cards that matters. The total number of combinations of any elements, taken r at a time, from a set of n elements is given by the combination formula nCvar>r (COMBIN(n,r) in Microsoft Excel and compatible spreadsheets). Thus, the total number of three-card hands, from a deck of 52 cards, is calculated by the formula 52C3 = 22,100.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hand** | **Frequency** | **Probability** | **Cumulative Probability** | **Odds** |
| Three of a kind/trio | 52 | 0.24% | 0.24% | 424.00:1 |
| Straight flush/pure sequence | 48 | 0.22% | 0.45% | 459.42:1 |
| Straight/sequence | 720 | 3.26% | 3.71% | 29.69:1 |
| Flush/colour | 1096 | 4.96% | 8.67% | 19.16:1 |
| Pair | 3744 | 16.94% | 25.61% | 4.90:1 |
| No pair/high card | 16440 | 74.39% | 100.00% | 0.34:1 |
| **Total** | **22,100** | **100.00%** | **100.00%** | **0.00:1** |

Although the probability of being dealt a pure sequence is slightly less than that of a trio, trios are considered higher hands in most versions of the game. Because of this variance from strict rarity, a popular house rule is to treat 2-3-5 of the same suit as a straight flush, thereby increasing the number of possible straight flushes to 52, the same as a trio, bringing the probabilities even.

### **Draw**

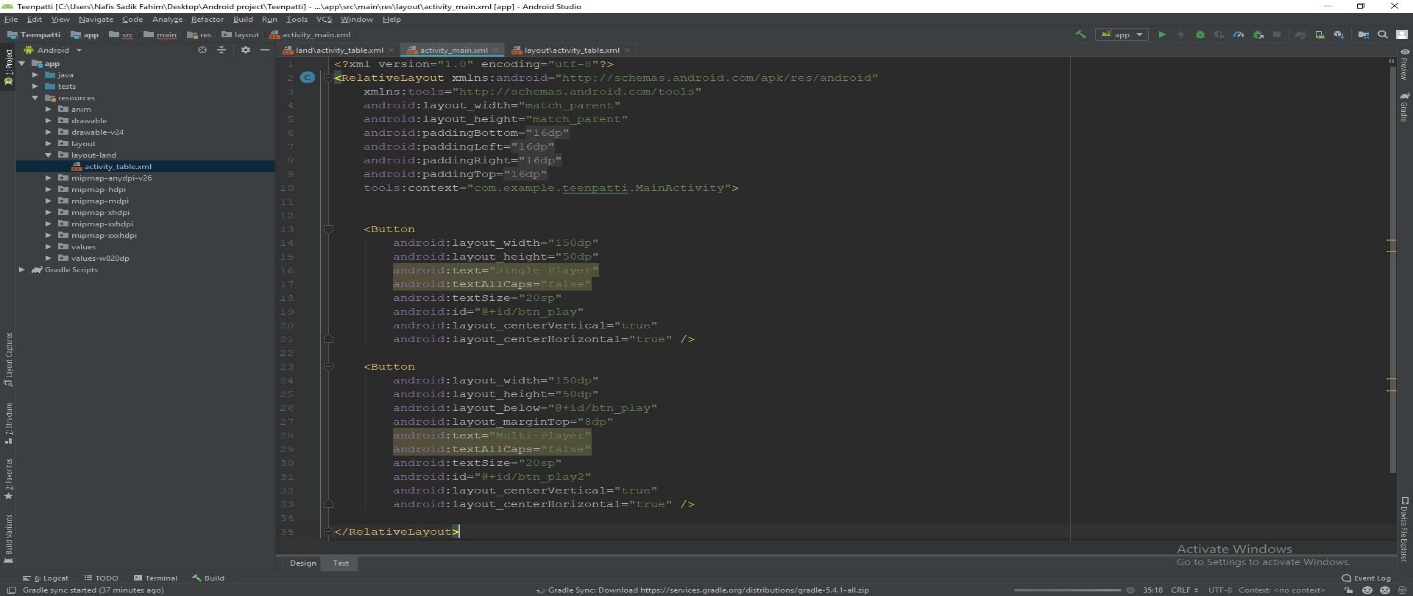
A complete hand is dealt to each player and, usually before (or after, but not both) betting, players are allowed to change their hand by discarding unwanted cards and being dealt new ones. Players may have to "purchase"/"buy" the new cards by putting a prearranged amount into the pot for each new card. Another version of draw allows players up to three chances to buy and change their hand, one card at a time, in the first three rounds of betting.

## **Implementation**

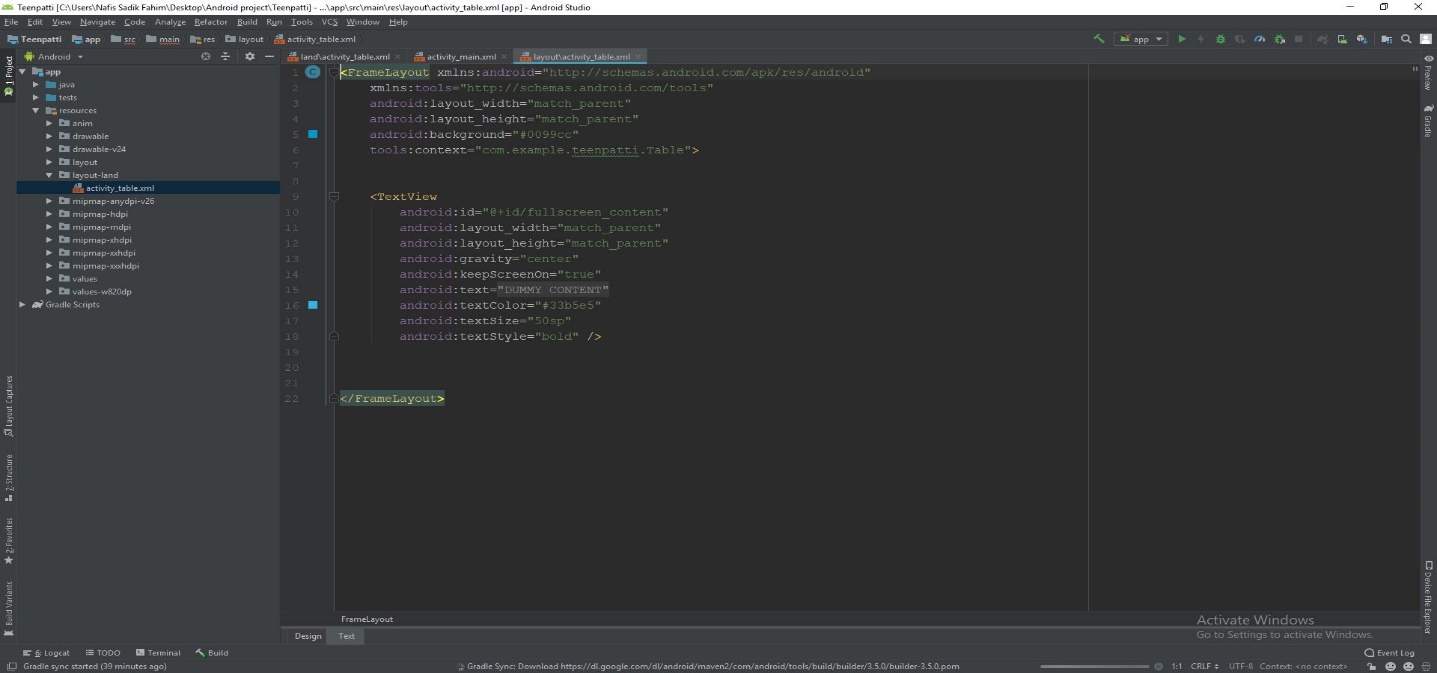
All code details are in appendix.

### **XML**

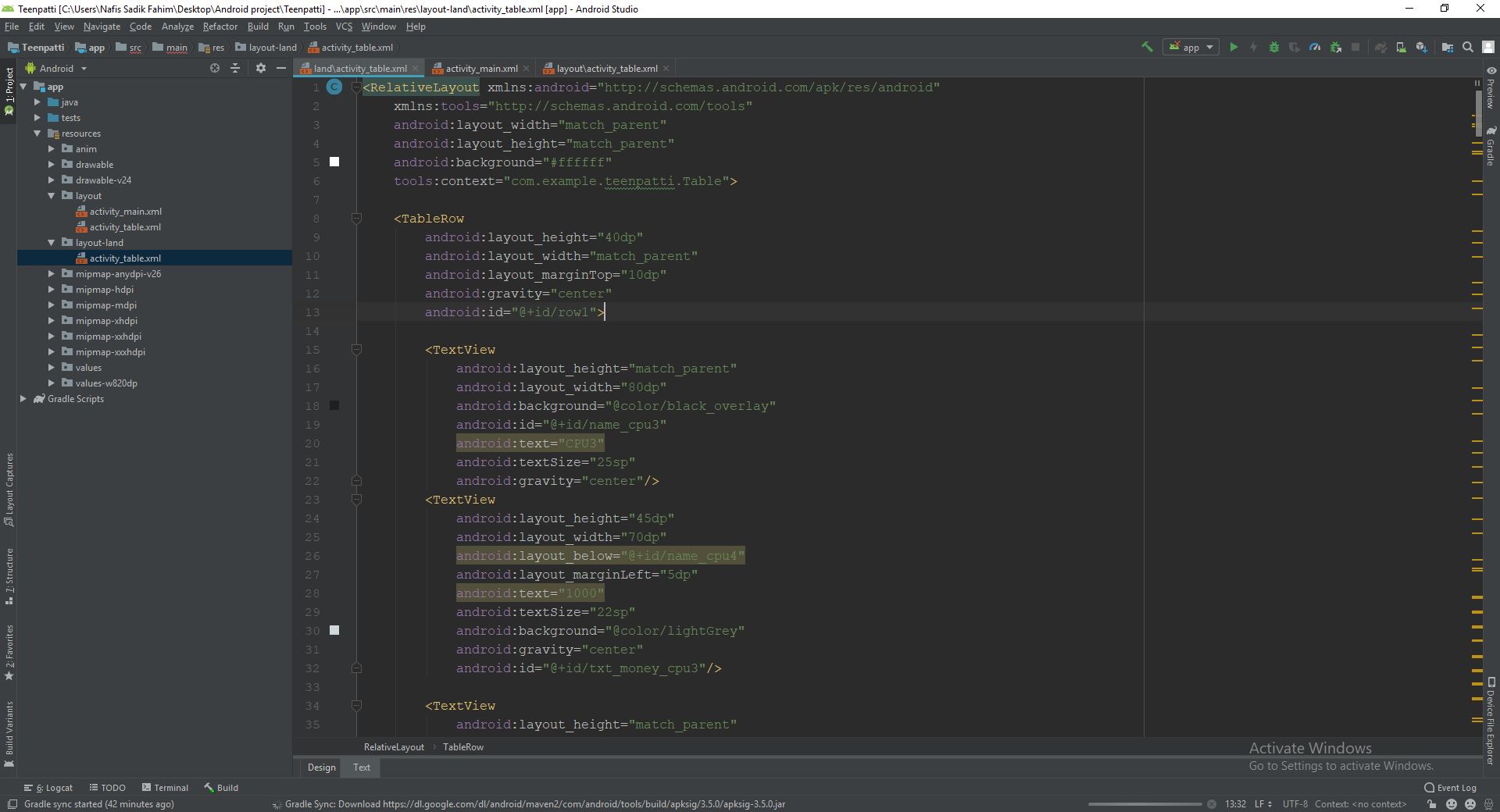
activity\_main.xml



layout\activity\_table.xml

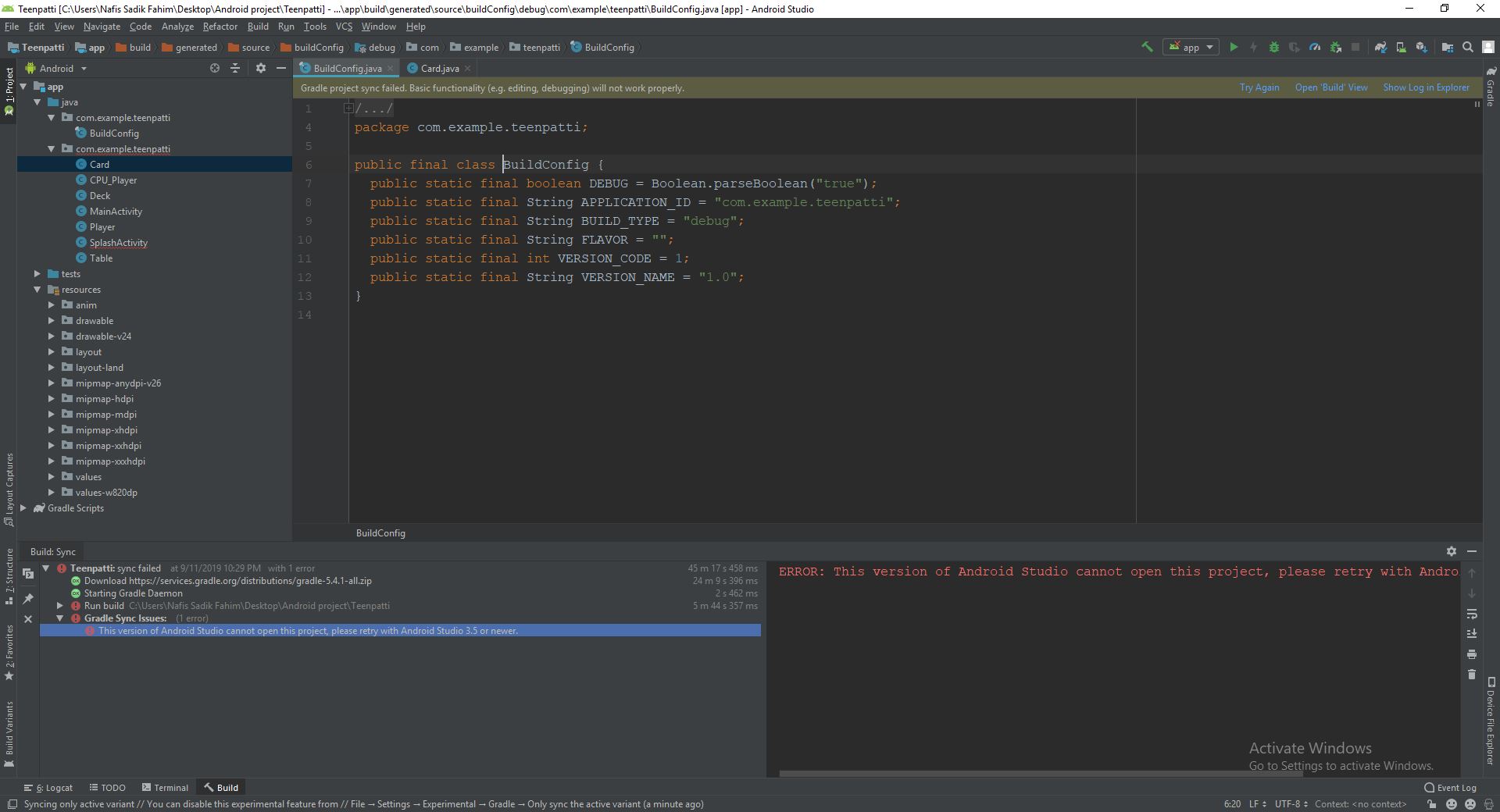


land\activity\_table.xml

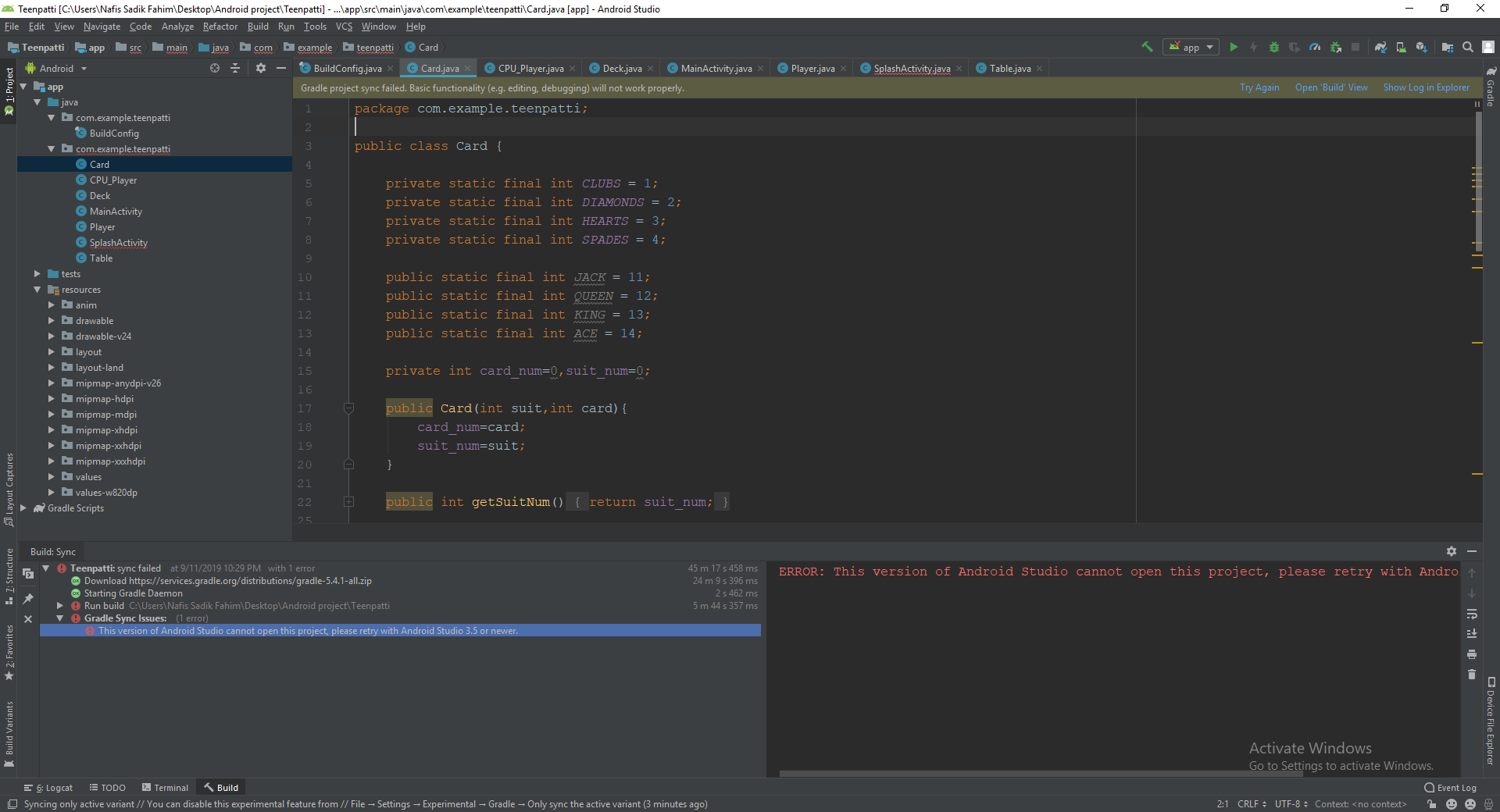


### **Java**

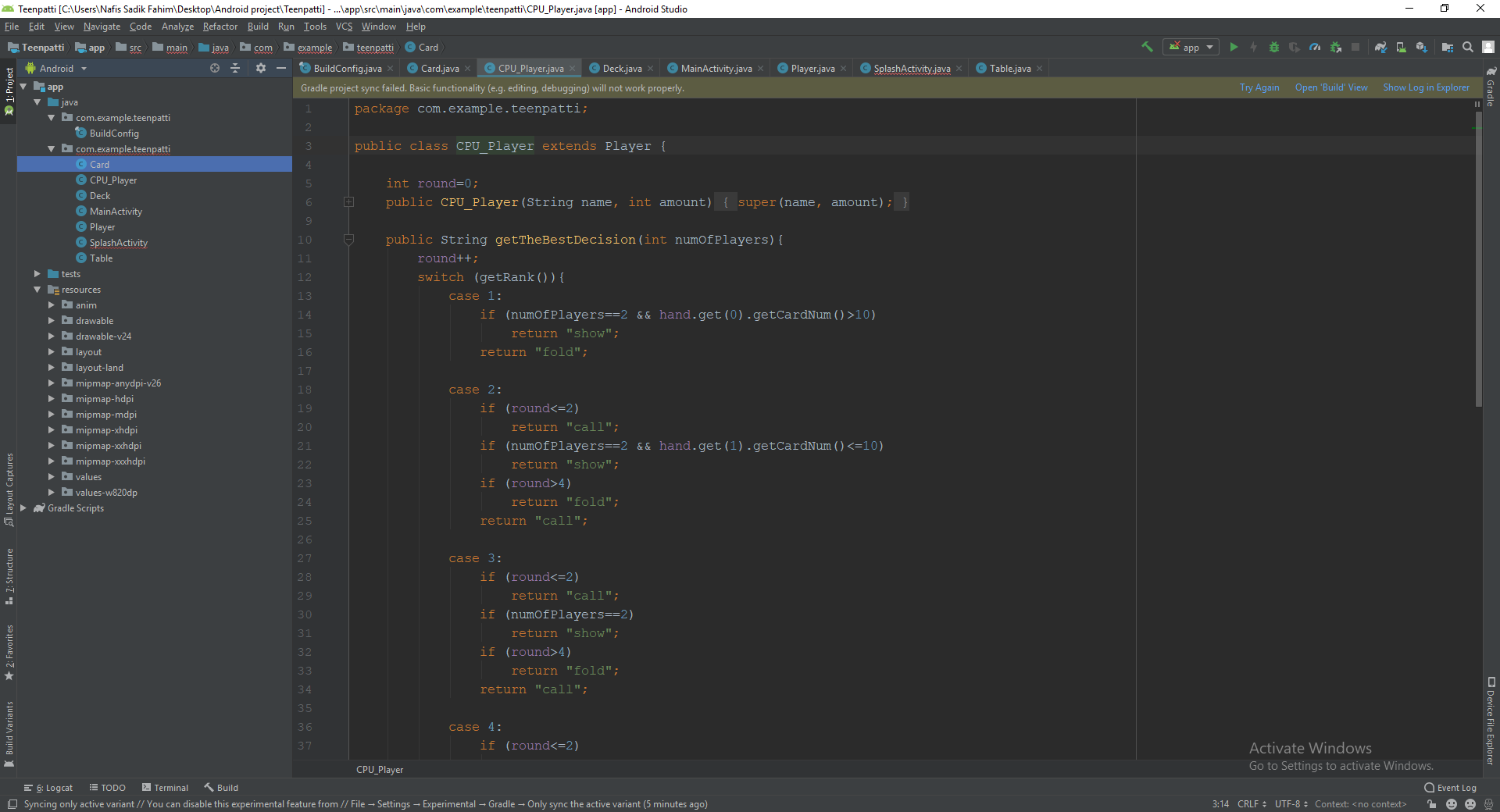
buildconfig.java



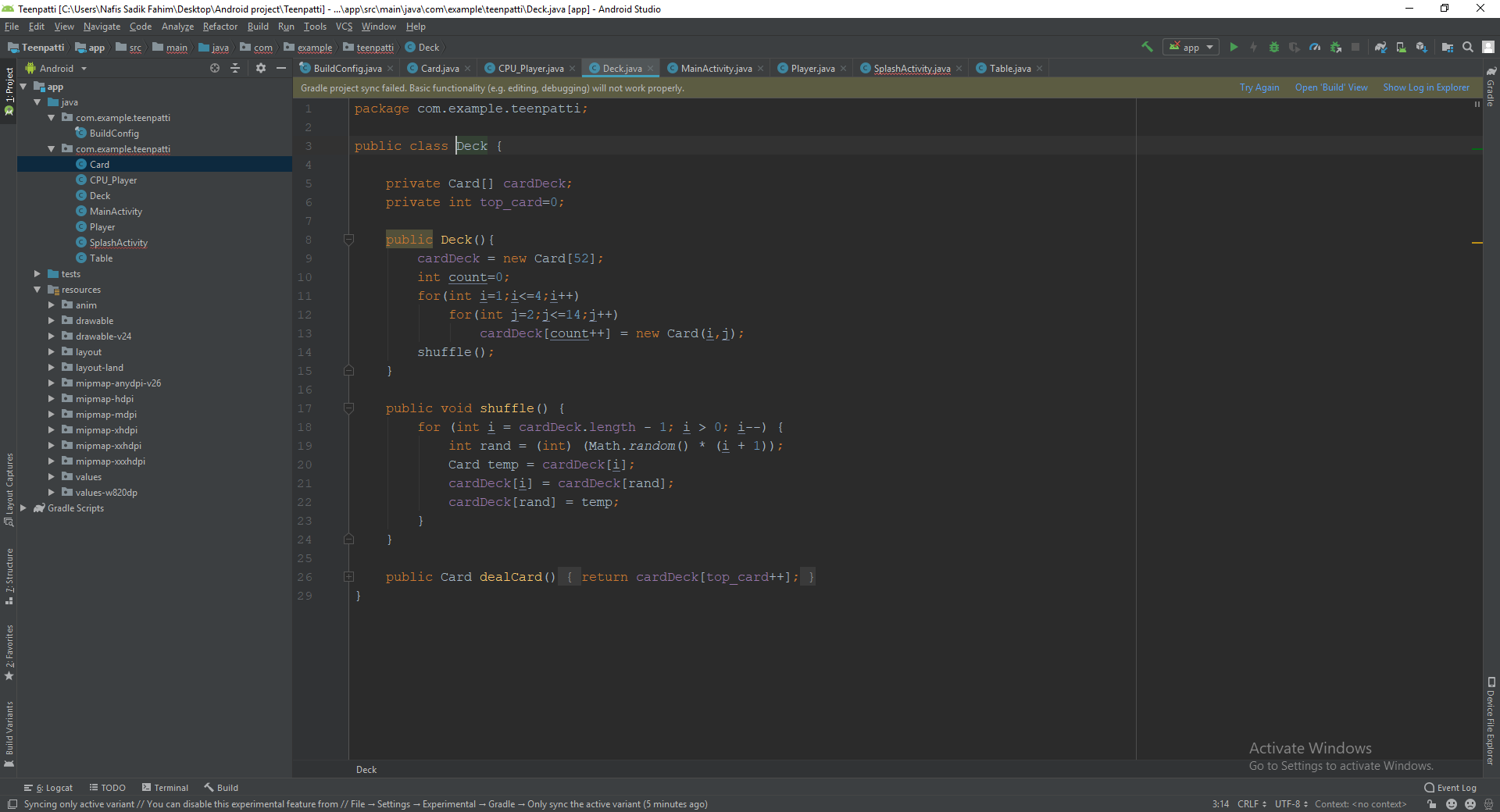
card.java



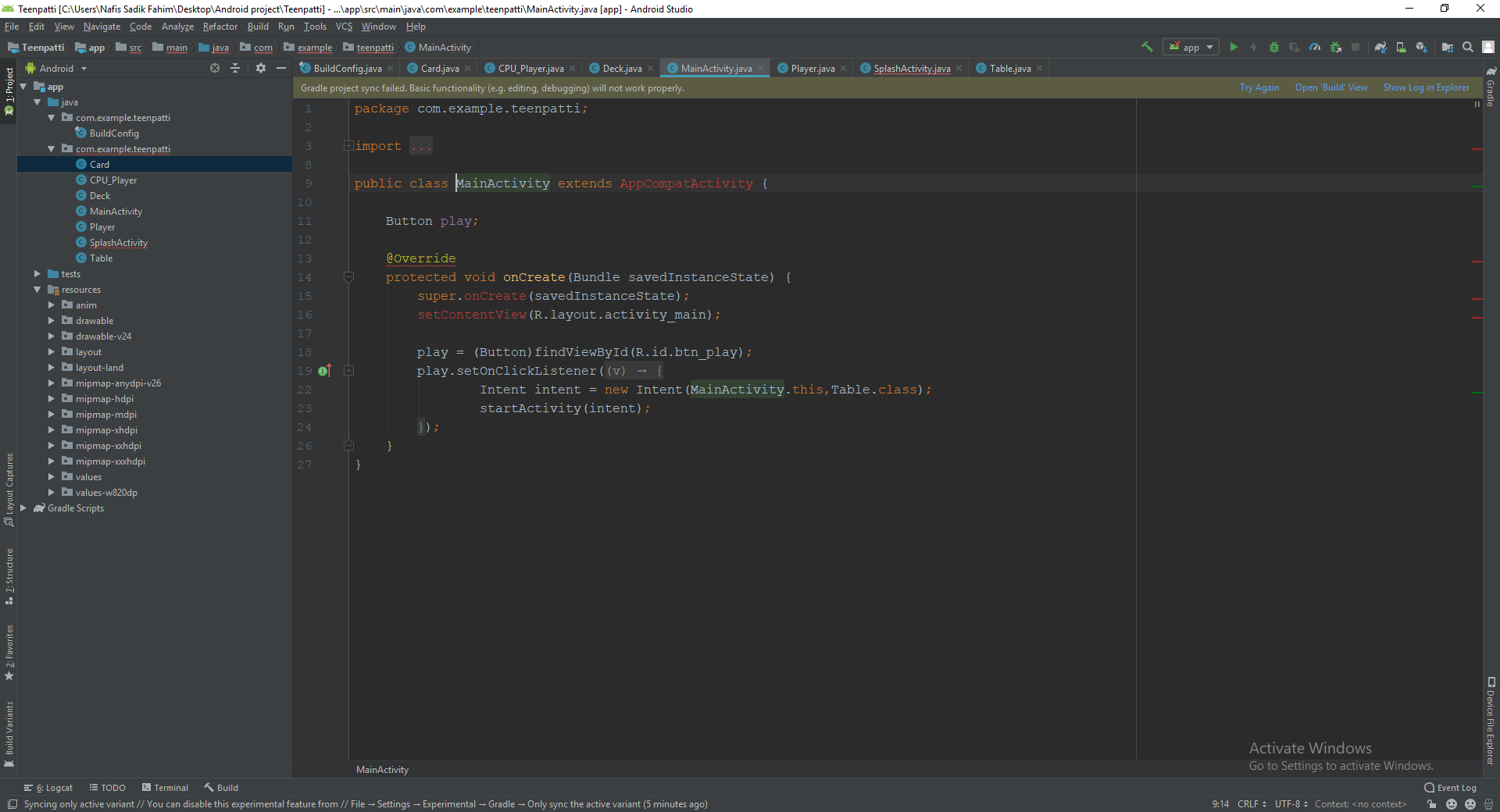
CPU\_player.java



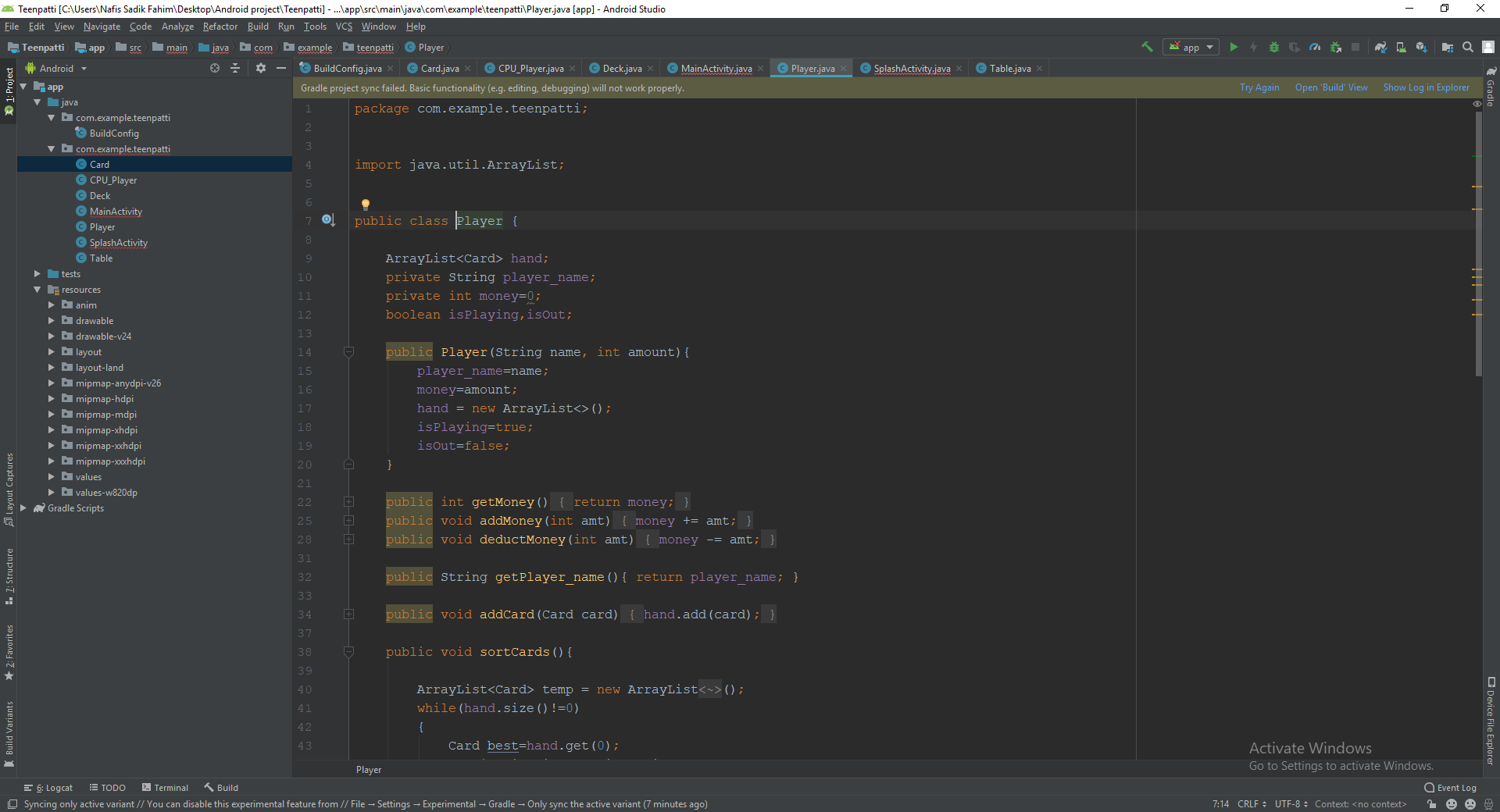
Deck.java



MainActivity.java



Player.java



SplashActivity.java

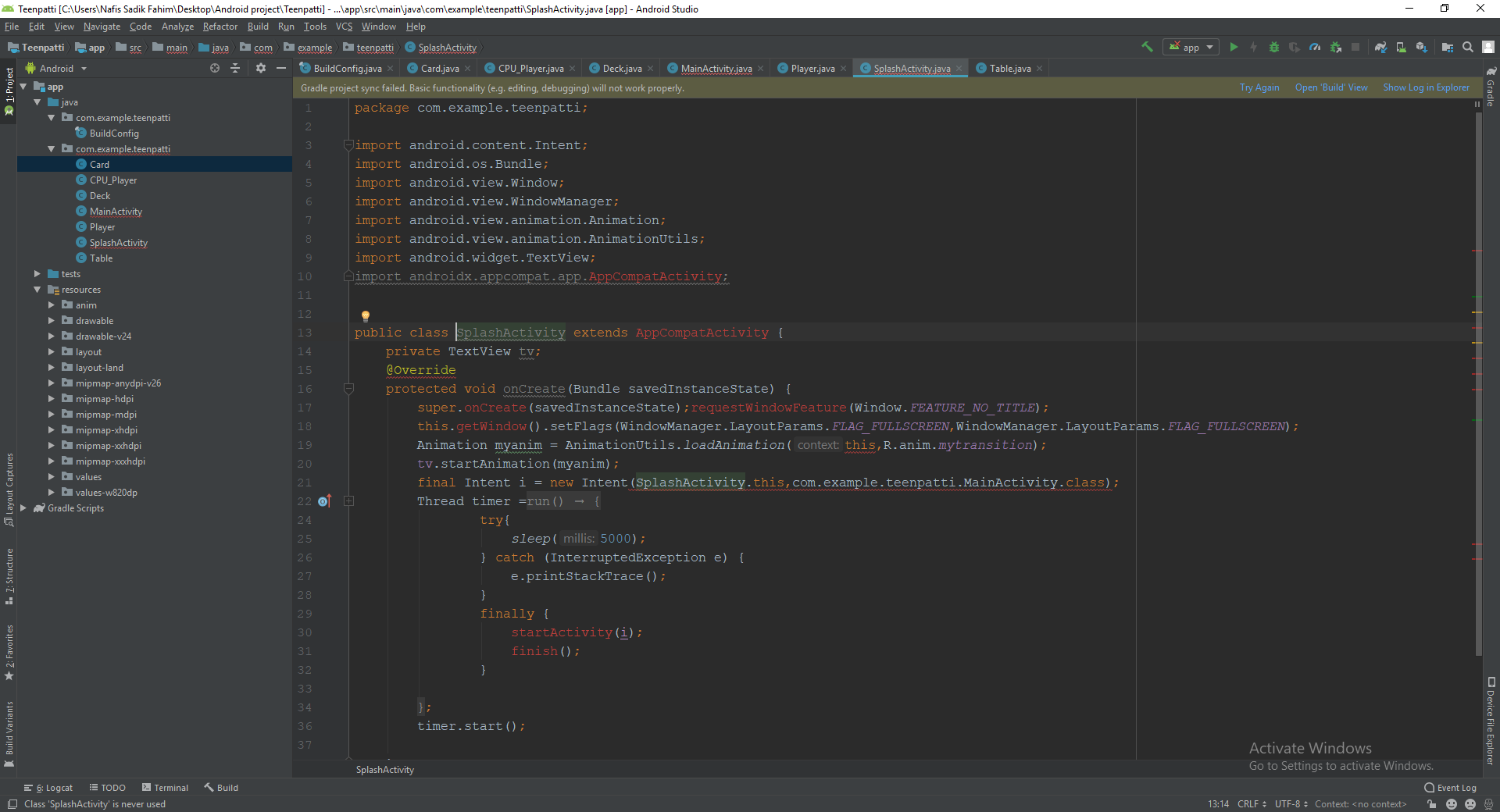
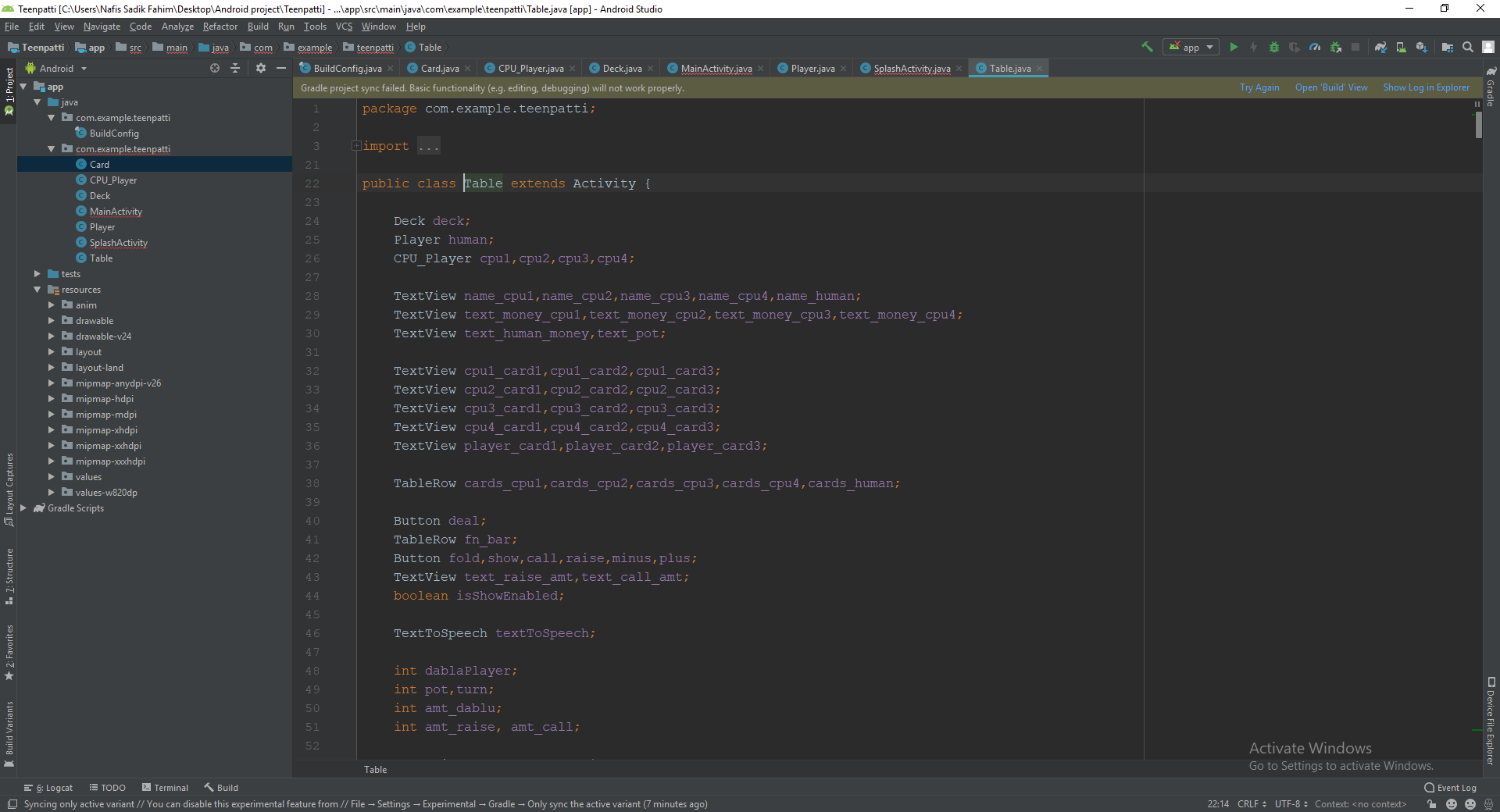
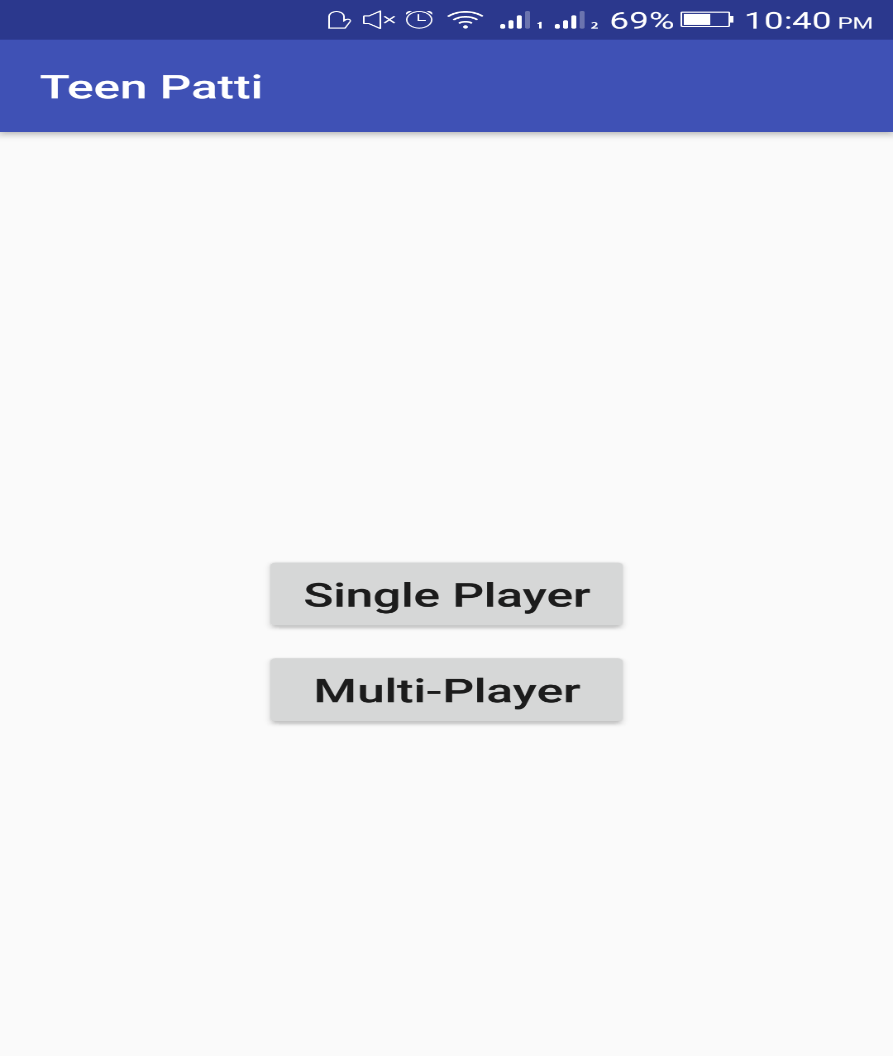


Table.java

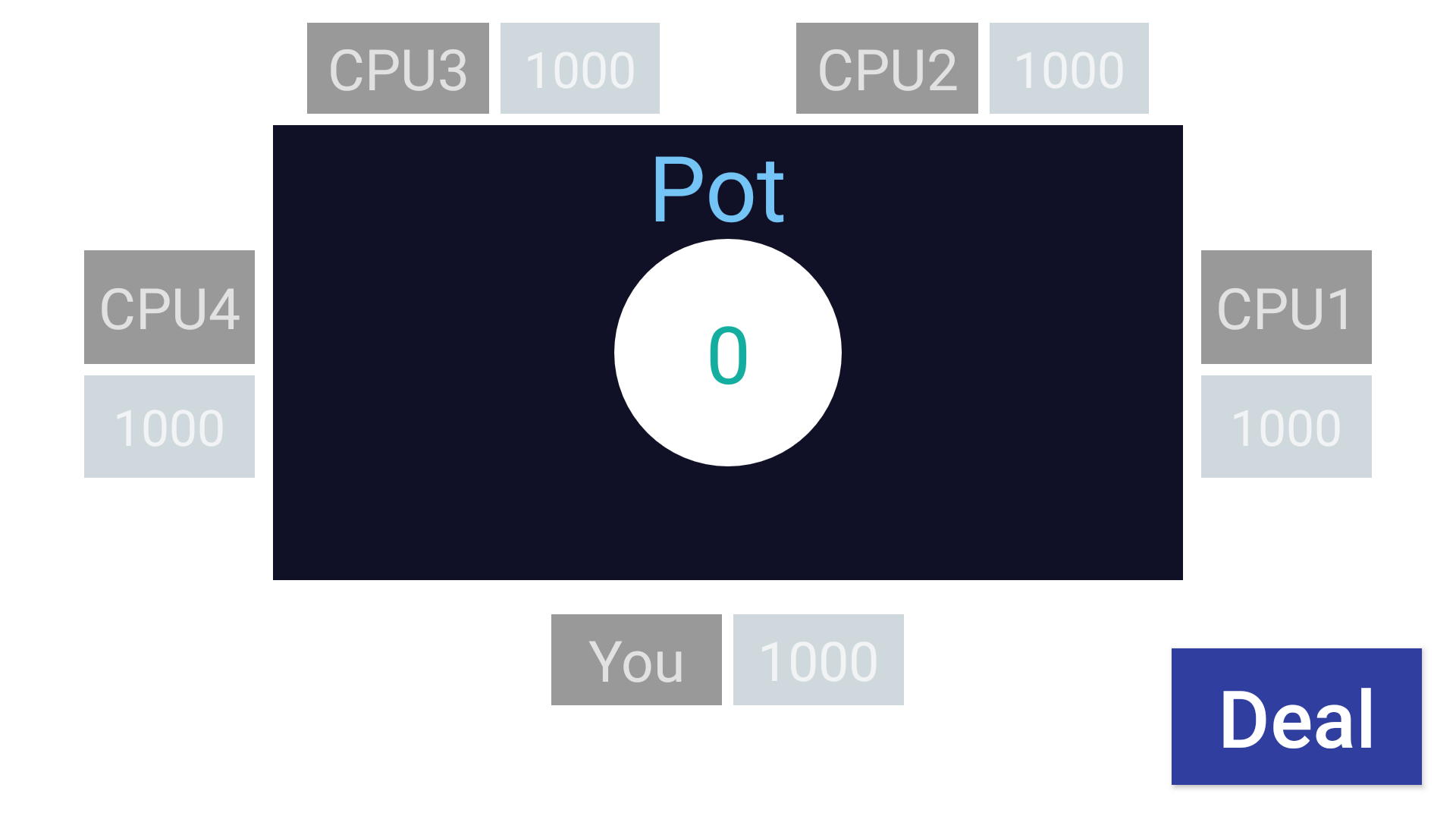


## **Interface(Simulation)**

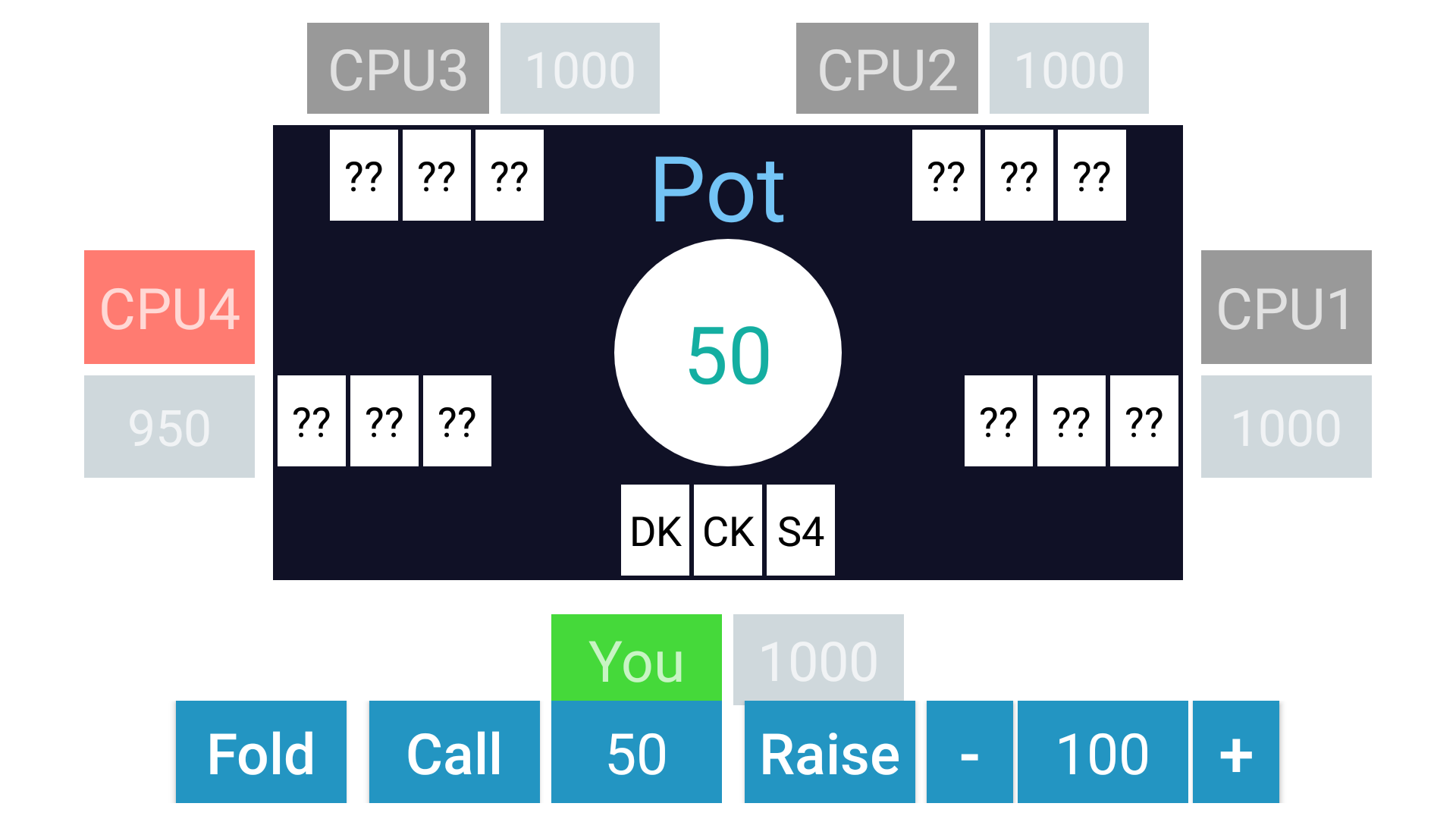
**Menu**



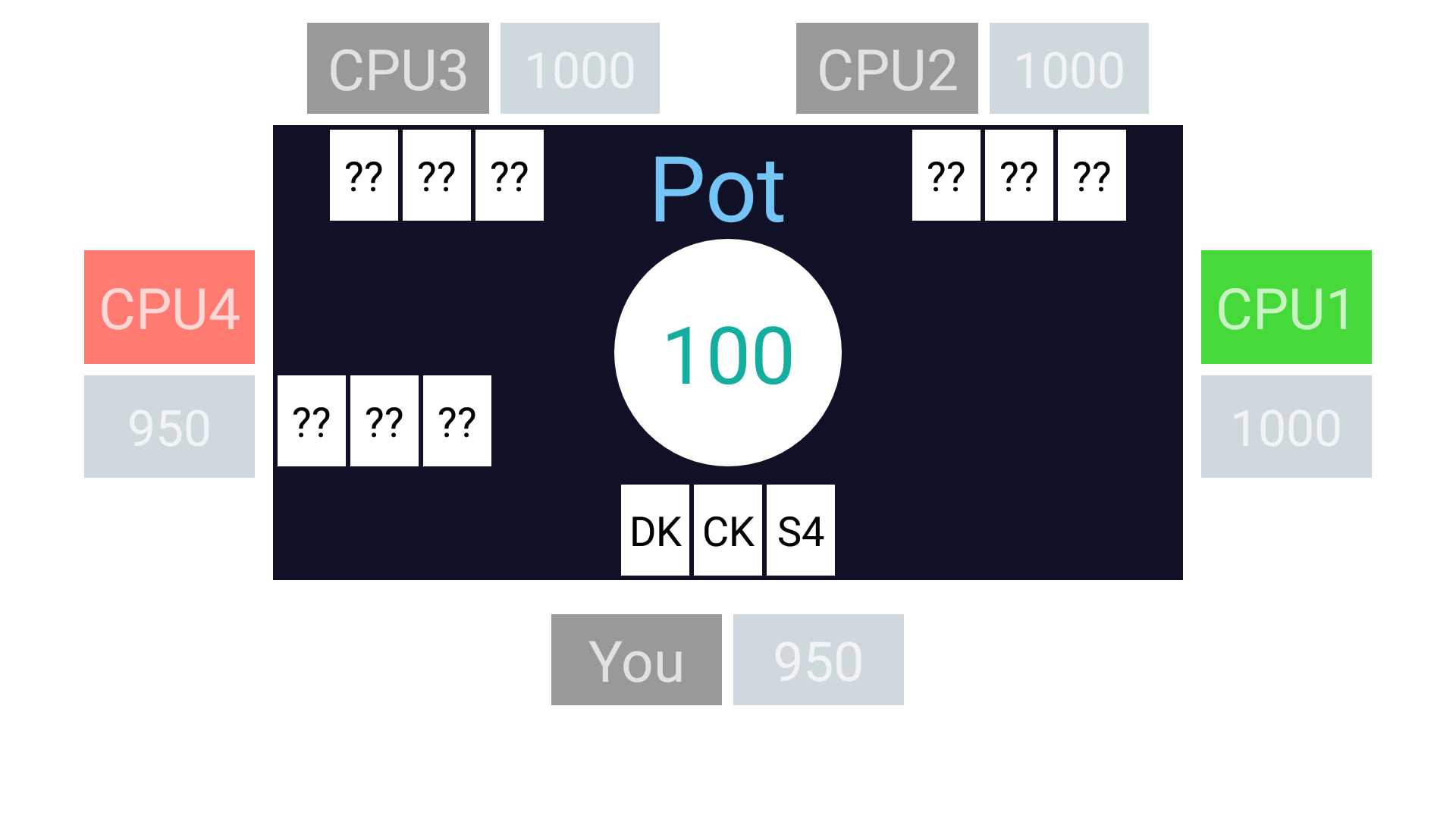
**Single player**



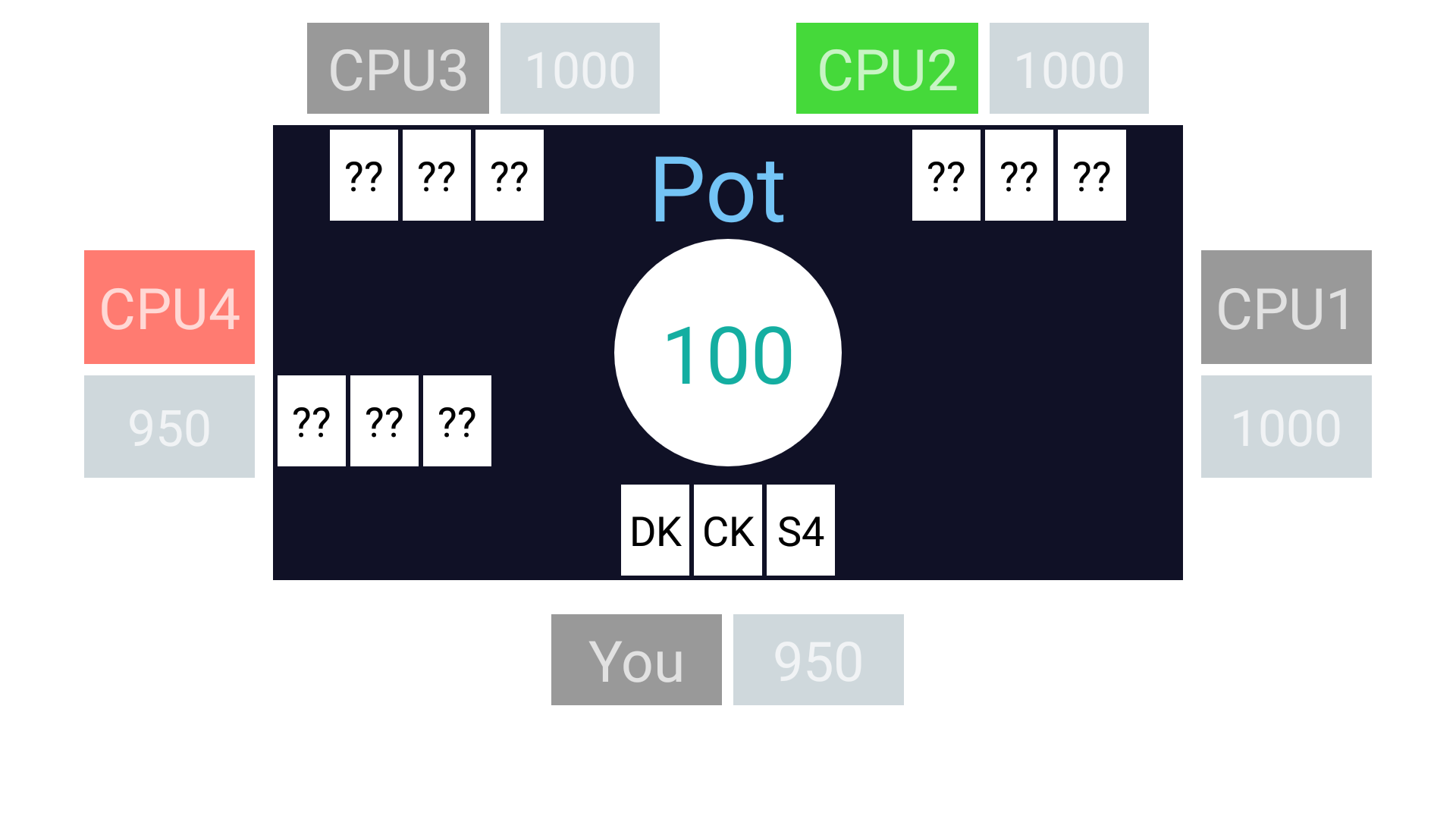
**After card are Dealt**



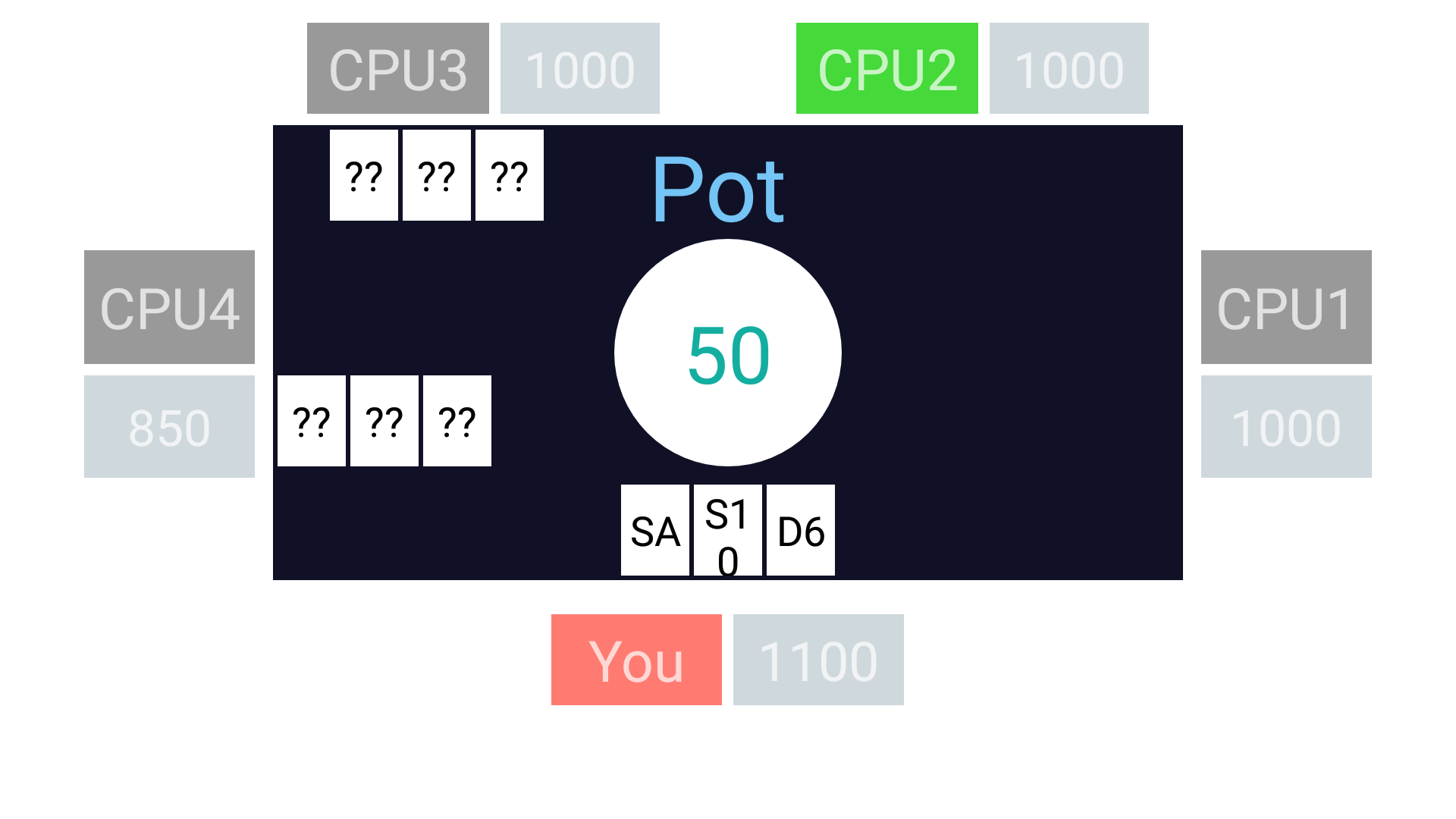
**After calling 50$**

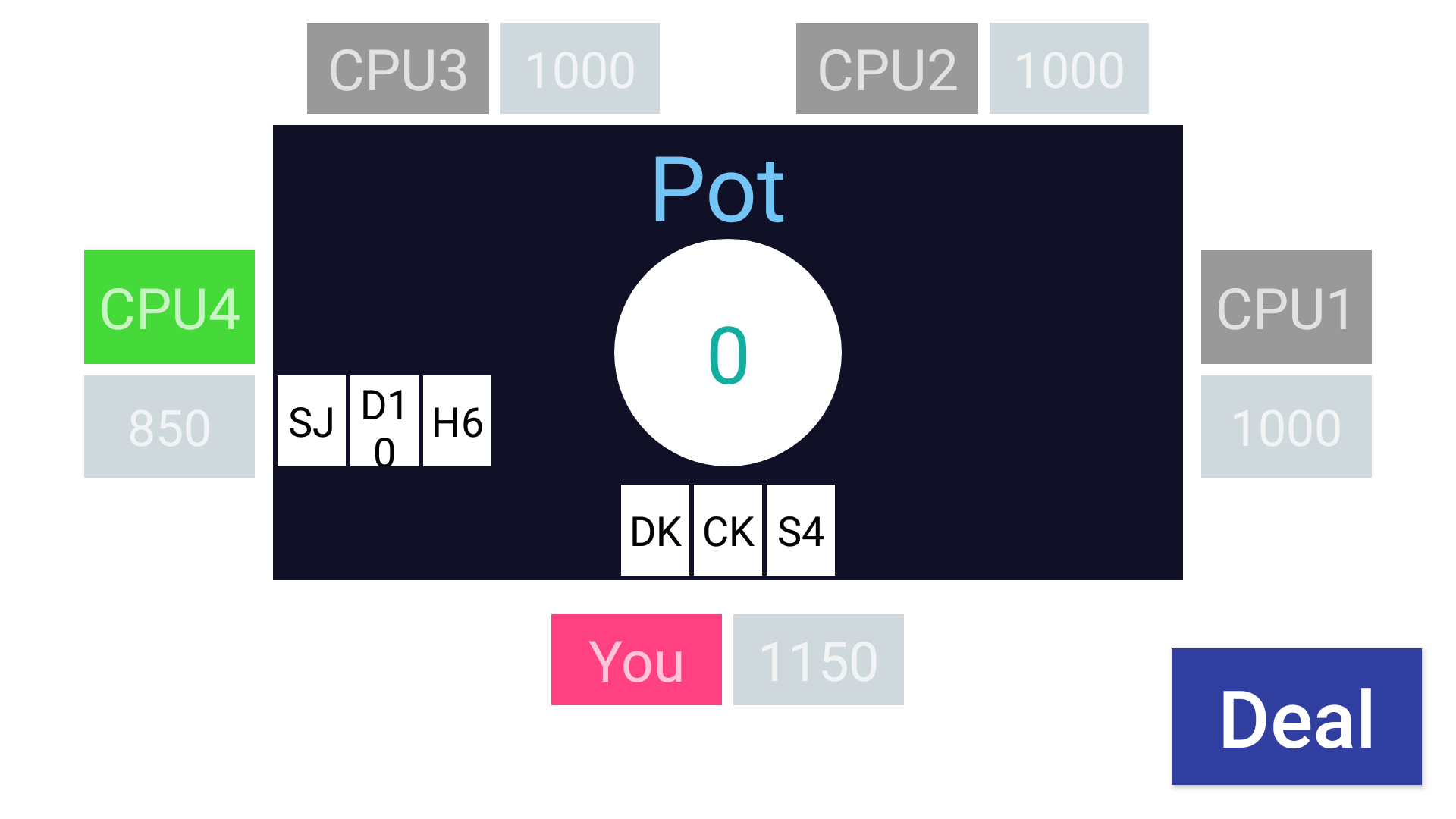


**CPU 1 folds**

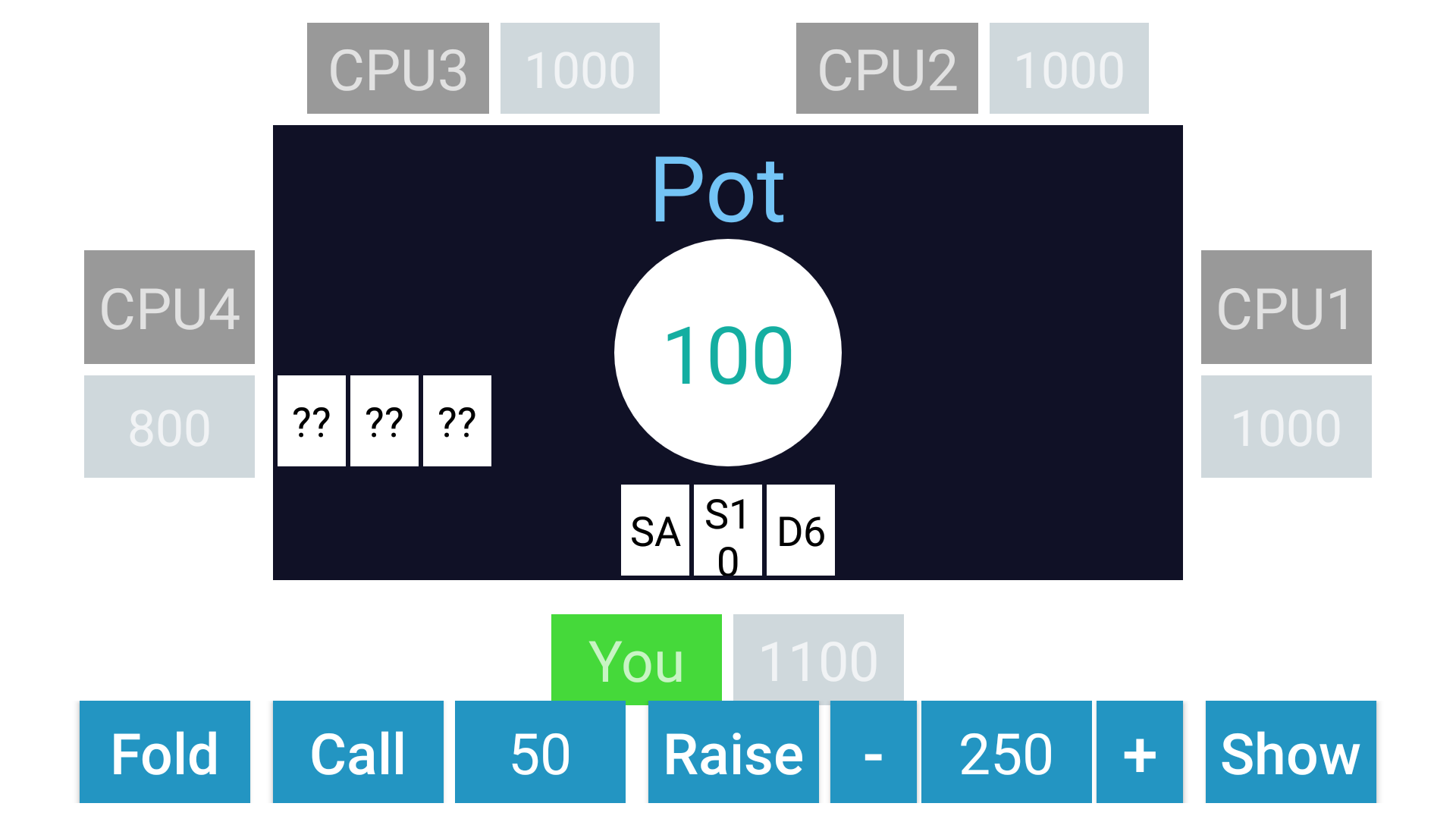


**CPU 2 is processing**

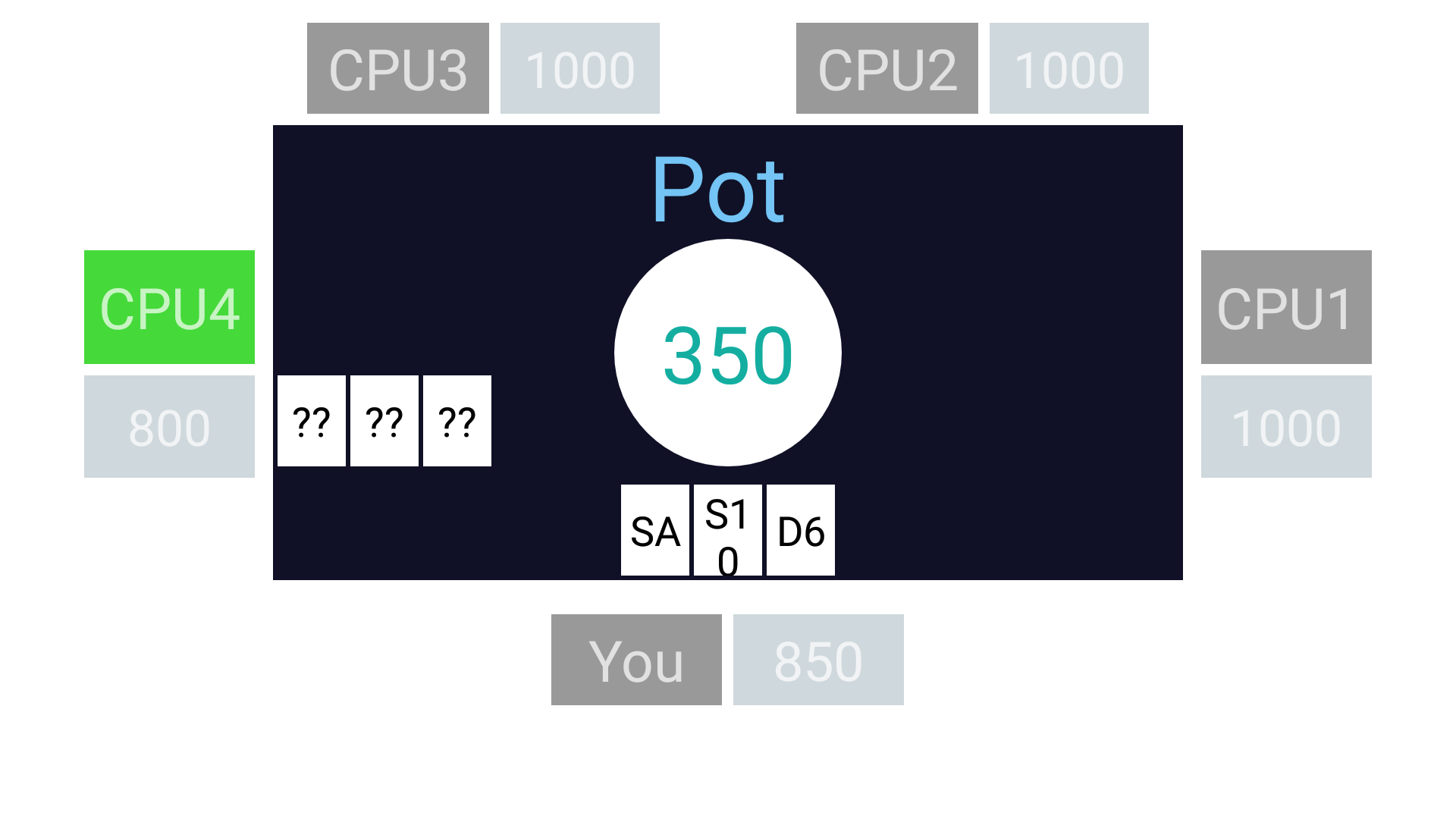


**CPU 4 called and user wins**

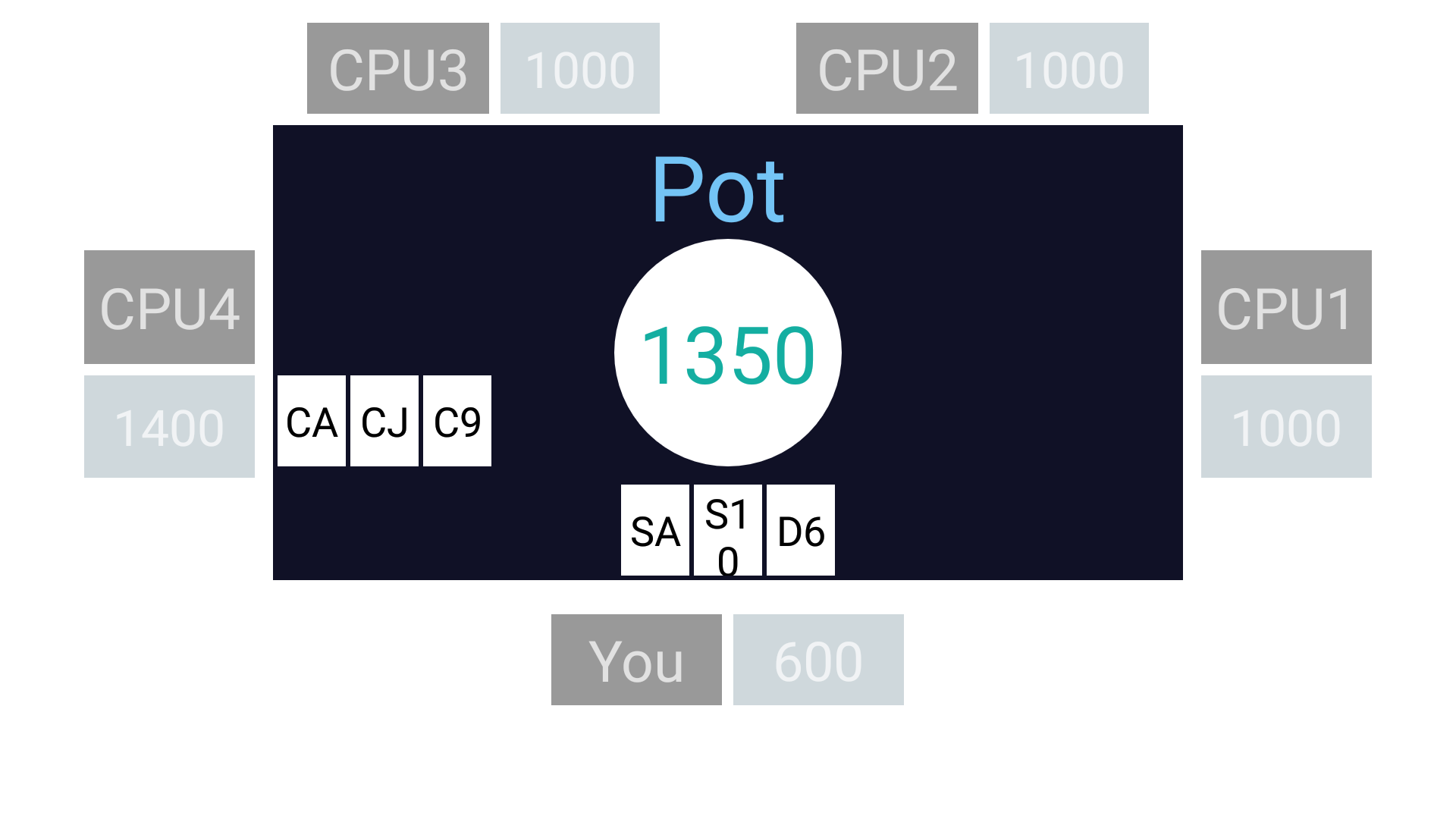
**User Raising stakes**



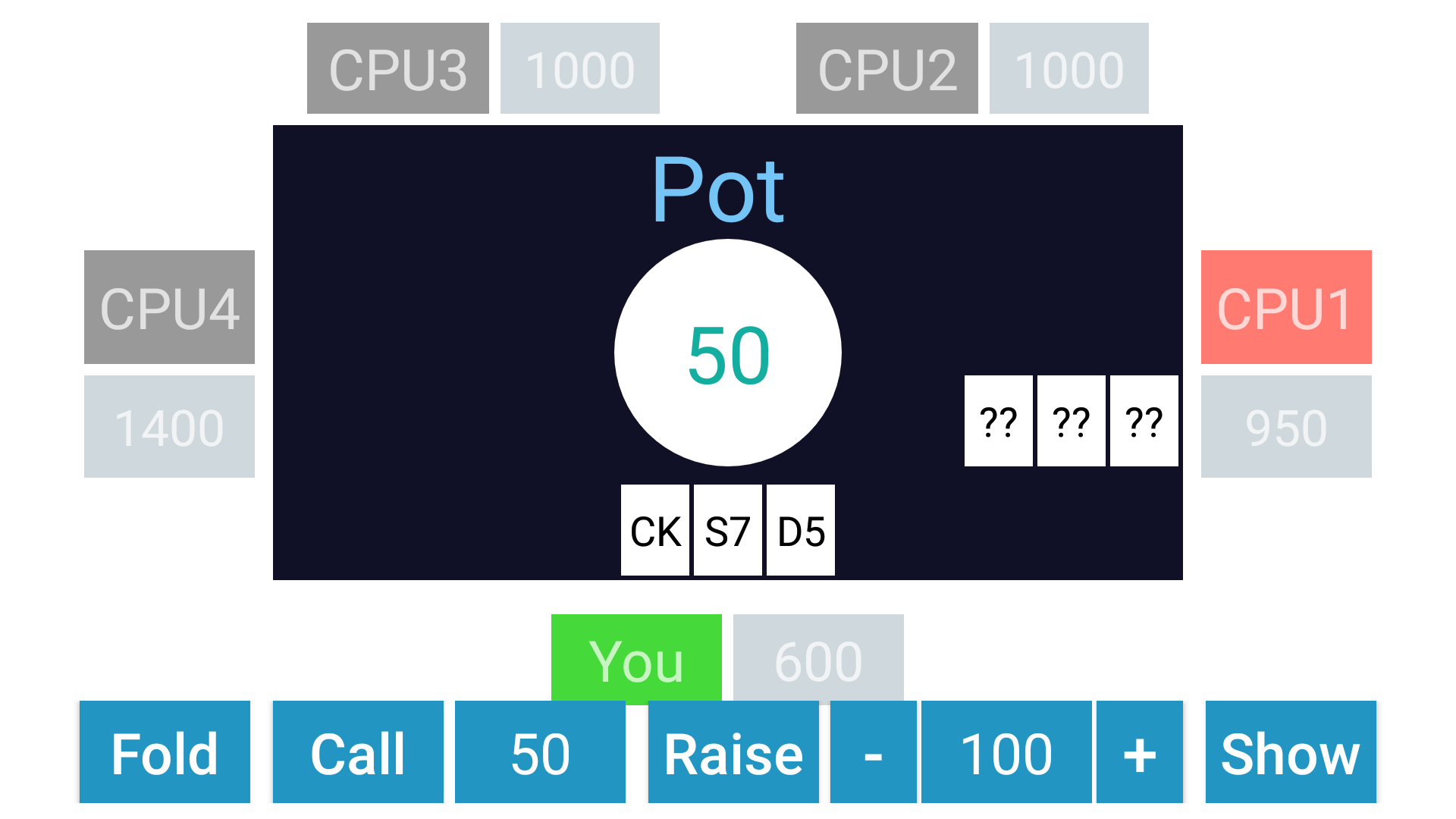
**CPU 1,2,3 folds. And CPU 4 is calling.**



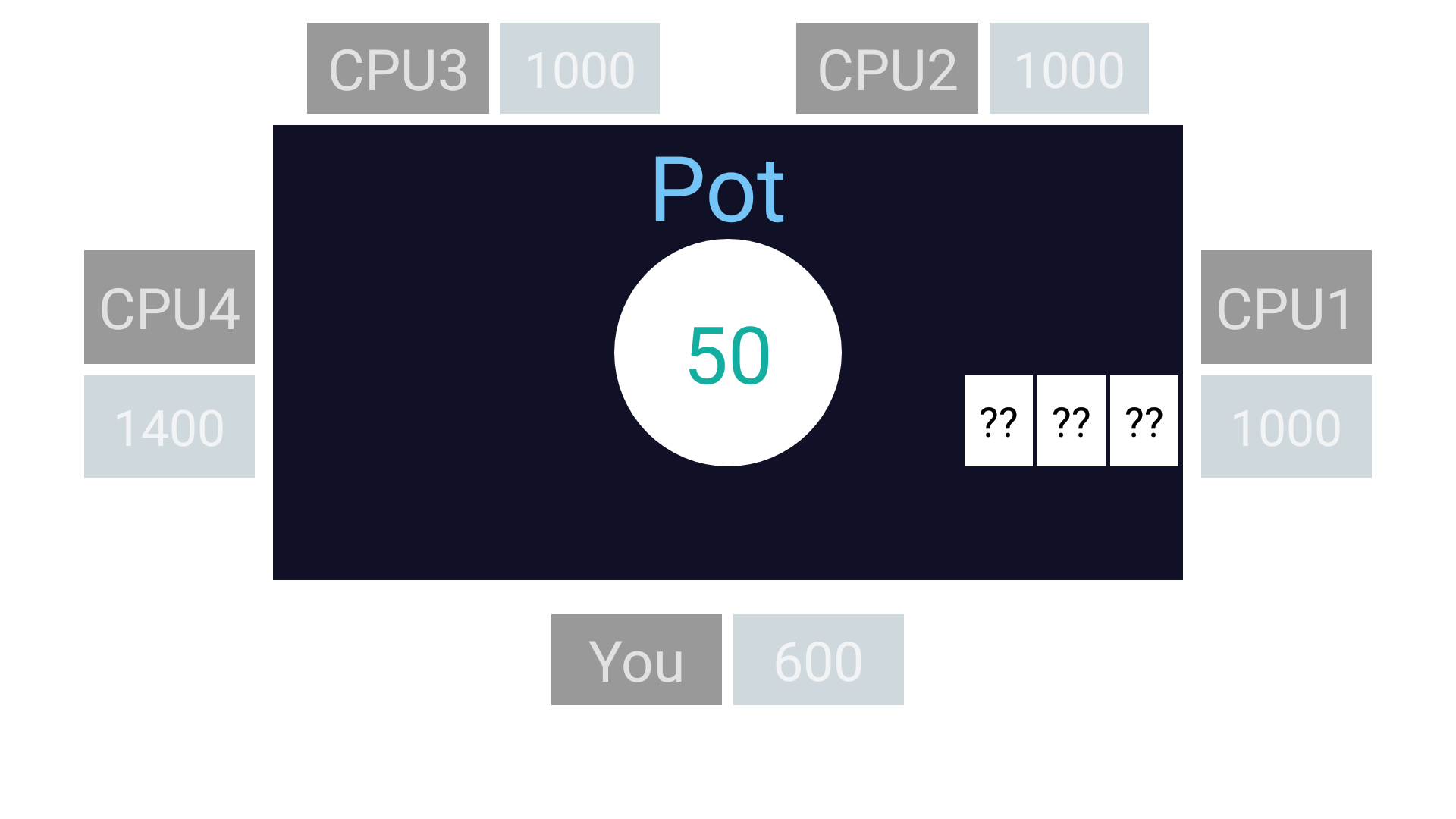
**CPU 4 wins**



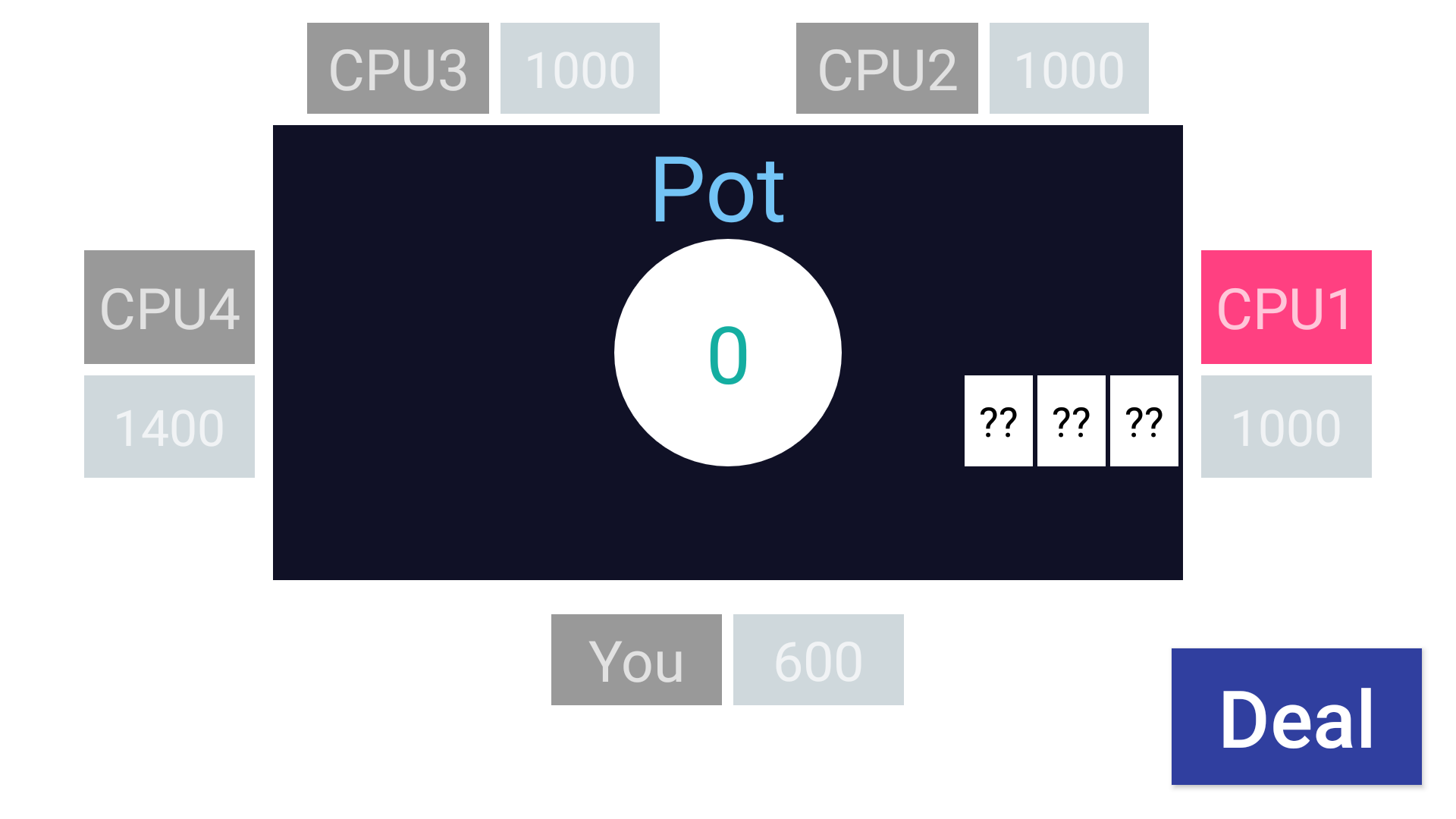
**CPU 1 processing and user is folding.**



**AI is carrying out the game**



**CPU 1 wins.**



## **Appendix**

MainActivity.java

|  |
| --- |
| package com.example.teenpatti;  import android.content.Intent; import android.os.Bundle; import android.view.View; import android.widget.Button; import androidx.appcompat.app.AppCompatActivity;  public class MainActivity extends AppCompatActivity {   Button play;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);   play = (Button)findViewById(R.id.*btn\_play*);  play.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(MainActivity.this,Table.class);  startActivity(intent);  }  });  } } |

SplashActivity.java

|  |
| --- |
| package com.example.teenpatti;  import android.content.Intent; import android.os.Bundle; import android.view.Window; import android.view.WindowManager; import android.view.animation.Animation; import android.view.animation.AnimationUtils; import android.widget.TextView; import androidx.appcompat.app.AppCompatActivity;   public class SplashActivity extends AppCompatActivity {  private TextView tv;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);requestWindowFeature(Window.*FEATURE\_NO\_TITLE*);  this.getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);  Animation myanim = AnimationUtils.*loadAnimation*(this,R.anim.*mytransition*);  tv.startAnimation(myanim);  final Intent i = new Intent(SplashActivity.this,com.example.teenpatti.MainActivity.class);  Thread timer =new Thread(){  public void run () {  try{  *sleep*(5000);  } catch (InterruptedException e) {  e.printStackTrace();  }  finally {  startActivity(i);  finish();  }  }  };  timer.start();   } } |

Card.java

|  |
| --- |
| package com.example.teenpatti;  public class Card {   private static final int *CLUBS* = 1;  private static final int *DIAMONDS* = 2;  private static final int *HEARTS* = 3;  private static final int *SPADES* = 4;   public static final int *JACK* = 11;  public static final int *QUEEN* = 12;  public static final int *KING* = 13;  public static final int *ACE* = 14;   private int card\_num=0,suit\_num=0;   public Card(int suit,int card){  card\_num=card;  suit\_num=suit;  }   public int getSuitNum() {  return suit\_num;  }   public int getCardNum() {  return card\_num;  }   public String getSuitAsString()  {  switch(suit\_num)  {  case *SPADES*: return "S";  case *CLUBS*: return "C";  case *DIAMONDS*: return "D";  case *HEARTS*: return "H";  default: return null;  }  }   public String getNumAsString()  {  switch(card\_num)  {  case 14: return "A";  case 2: return "2";  case 3: return "3";  case 4: return "4";  case 5: return "5";  case 6: return "6";  case 7: return "7";  case 8: return "8";  case 9: return "9";  case 10: return "10";  case 11: return "J";  case 12: return "Q";  case 13: return "K";  default: return null;  }  }   public String toString()  {  return getSuitAsString() + getNumAsString();  } } |

CPU\_player.java

|  |
| --- |
| package com.example.teenpatti;  public class CPU\_Player extends Player {   int round=0;  public CPU\_Player(String name, int amount) {  super(name, amount);  }   public String getTheBestDecision(int numOfPlayers){  round++;  switch (getRank()){  case 1:  if (numOfPlayers==2 && hand.get(0).getCardNum()>10)  return "show";  return "fold";   case 2:  if (round<=2)  return "call";  if (numOfPlayers==2 && hand.get(1).getCardNum()<=10)  return "show";  if (round>4)  return "fold";  return "call";   case 3:  if (round<=2)  return "call";  if (numOfPlayers==2)  return "show";  if (round>4)  return "fold";  return "call";   case 4:  if (round<=2)  return "call";  if (numOfPlayers==2)  return "show";  return "call";   case 5:  if (round<=3)  return "call";  if (round==4 && hand.get(0).getCardNum()<=8)  return "call";  if (round==4 && hand.get(0).getCardNum()>8)  return "raise,50";  if (numOfPlayers==2)  return "show";  return "call";   case 6:  if (round<=2)  return "call";  if (round==3)  return "raise,50";  if (round==4)  return "call";  if (round==5)  return "raise,100";  return "call";   default: return null;  }  }   @Override  public void clearHand() {  hand.clear();  round=0;  } } |

Deck.java

|  |
| --- |
| package com.example.teenpatti;  public class Deck {   private Card[] cardDeck;  private int top\_card=0;   public Deck(){  cardDeck = new Card[52];  int count=0;  for(int i=1;i<=4;i++)  for(int j=2;j<=14;j++)  cardDeck[count++] = new Card(i,j);  shuffle();  }   public void shuffle() {  for (int i = cardDeck.length - 1; i > 0; i--) {  int rand = (int) (Math.*random*() \* (i + 1));  Card temp = cardDeck[i];  cardDeck[i] = cardDeck[rand];  cardDeck[rand] = temp;  }  }   public Card dealCard(){  return cardDeck[top\_card++];  } } |

Player.java

|  |
| --- |
| package com.example.teenpatti;   import java.util.ArrayList;   public class Player {   ArrayList<Card> hand;  private String player\_name;  private int money=0;  boolean isPlaying,isOut;   public Player(String name, int amount){  player\_name=name;  money=amount;  hand = new ArrayList<>();  isPlaying=true;  isOut=false;  }   public int getMoney() {  return money;  }  public void addMoney(int amt){  money += amt;  }  public void deductMoney(int amt){  money -= amt;  }   public String getPlayer\_name(){ return player\_name; }   public void addCard(Card card){  hand.add(card);  }   public void sortCards(){   ArrayList<Card> temp = new ArrayList<Card>();  while(hand.size()!=0)  {  Card best=hand.get(0);  for(int i=1;i<hand.size();i++)  if(hand.get(i).getCardNum()>best.getCardNum())  best=hand.get(i);   temp.add(best);  hand.remove(best);  }  hand = temp;  }   public int getRank(){   // 6. Three of a kind  if (hand.get(0).getCardNum()==hand.get(1).getCardNum()  && hand.get(0).getCardNum()==hand.get(2).getCardNum())  return 5;   // Straight  else if (hand.get(0).getCardNum()-hand.get(1).getCardNum()==1  && hand.get(1).getCardNum()-hand.get(2).getCardNum()==1){   if (hand.get(0).getSuitNum()==hand.get(1).getSuitNum()  && hand.get(0).getSuitNum()==hand.get(2).getSuitNum())  return 6; // 6. Straight Flush  else  return 4; // 4. Straight  }   // 3. Flush  else if (hand.get(0).getSuitNum()==hand.get(1).getSuitNum()  && hand.get(0).getSuitNum()==hand.get(2).getSuitNum())  return 3;   // 2. Pair  else if (hand.get(0).getCardNum()==hand.get(1).getCardNum()  || hand.get(1).getCardNum()==hand.get(2).getCardNum())  return 2;   // 1. High Card  else  return 1;  }   public void clearHand(){  hand.clear();  } } |

Table.java

|  |
| --- |
| package com.example.teenpatti;  import android.annotation.TargetApi; import android.app.Activity; import android.content.Intent; import android.os.Build; import android.os.Bundle; import android.os.Handler; import android.os.Message; import android.speech.tts.TextToSpeech; import android.util.Log; import android.view.View; import android.widget.Button; import android.widget.TableRow; import android.widget.TextView;  import java.util.ArrayList; import java.util.Collections; import java.util.Comparator; import java.util.Locale;  public class Table extends Activity {   Deck deck;  Player human;  CPU\_Player cpu1,cpu2,cpu3,cpu4;   TextView name\_cpu1,name\_cpu2,name\_cpu3,name\_cpu4,name\_human;  TextView text\_money\_cpu1,text\_money\_cpu2,text\_money\_cpu3,text\_money\_cpu4;  TextView text\_human\_money,text\_pot;   TextView cpu1\_card1,cpu1\_card2,cpu1\_card3;  TextView cpu2\_card1,cpu2\_card2,cpu2\_card3;  TextView cpu3\_card1,cpu3\_card2,cpu3\_card3;  TextView cpu4\_card1,cpu4\_card2,cpu4\_card3;  TextView player\_card1,player\_card2,player\_card3;   TableRow cards\_cpu1,cards\_cpu2,cards\_cpu3,cards\_cpu4,cards\_human;   Button deal;  TableRow fn\_bar;  Button fold,show,call,raise,minus,plus;  TextView text\_raise\_amt,text\_call\_amt;  boolean isShowEnabled;   TextToSpeech textToSpeech;   int dablaPlayer;  int pot,turn;  int amt\_dablu;  int amt\_raise, amt\_call;   ArrayList<Player> playerList;   @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  //setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);  setContentView(R.layout.*activity\_table*);   name\_cpu1 = (TextView)findViewById(R.id.*name\_cpu1*);  name\_cpu2 = (TextView)findViewById(R.id.*name\_cpu2*);  name\_cpu3 = (TextView)findViewById(R.id.*name\_cpu3*);  name\_cpu4 = (TextView)findViewById(R.id.*name\_cpu4*);  name\_human = (TextView)findViewById(R.id.*name\_human*);   cards\_cpu1 = (TableRow)findViewById(R.id.*cards\_cpu1*);  cards\_cpu2 = (TableRow)findViewById(R.id.*cards\_cpu2*);  cards\_cpu3 = (TableRow)findViewById(R.id.*cards\_cpu3*);  cards\_cpu4 = (TableRow)findViewById(R.id.*cards\_cpu4*);  cards\_human = (TableRow)findViewById(R.id.*cards\_human*);   player\_card1 = (TextView)findViewById(R.id.*human\_card1*);  player\_card2 = (TextView)findViewById(R.id.*human\_card2*);  player\_card3 = (TextView)findViewById(R.id.*human\_card3*);   cpu1\_card1 = (TextView)findViewById(R.id.*cpu1\_cardnum1*);  cpu1\_card2 = (TextView)findViewById(R.id.*cpu1\_card2*);  cpu1\_card3 = (TextView)findViewById(R.id.*cpu1\_card3*);   cpu2\_card1 = (TextView)findViewById(R.id.*cpu2\_cardnum1*);  cpu2\_card2 = (TextView)findViewById(R.id.*cpu2\_card2*);  cpu2\_card3 = (TextView)findViewById(R.id.*cpu2\_card3*);   cpu3\_card1 = (TextView)findViewById(R.id.*cpu3\_cardnum1*);  cpu3\_card2 = (TextView)findViewById(R.id.*cpu3\_card2*);  cpu3\_card3 = (TextView)findViewById(R.id.*cpu3\_card3*);   cpu4\_card1 = (TextView)findViewById(R.id.*cpu4\_cardnum1*);  cpu4\_card2 = (TextView)findViewById(R.id.*cpu4\_card2*);  cpu4\_card3 = (TextView)findViewById(R.id.*cpu4\_card3*);   fn\_bar = (TableRow)findViewById(R.id.*fn\_bar*);  fold = (Button)findViewById(R.id.*btn\_fold*);  show = (Button)findViewById(R.id.*btn\_show*);  call = (Button)findViewById(R.id.*btn\_call*);  raise = (Button)findViewById(R.id.*btn\_raise*);  minus = (Button)findViewById(R.id.*btn\_minus*);  plus = (Button)findViewById(R.id.*btn\_plus*);  text\_raise\_amt = (TextView)findViewById(R.id.*txt\_raise*);  text\_call\_amt = (TextView)findViewById(R.id.*txt\_call*);   text\_money\_cpu1 = (TextView)findViewById(R.id.*txt\_money\_cpu1*);  text\_money\_cpu2 = (TextView)findViewById(R.id.*txt\_money\_cpu2*);  text\_money\_cpu3 = (TextView)findViewById(R.id.*txt\_money\_cpu3*);  text\_money\_cpu4 = (TextView)findViewById(R.id.*txt\_money\_cpu4*);  text\_human\_money = (TextView)findViewById(R.id.*txt\_human\_money*);   text\_pot = (TextView)findViewById(R.id.*pot*);  deal = (Button)findViewById(R.id.*deal*);   textToSpeech = new TextToSpeech(this, new TextToSpeech.OnInitListener() {  @Override  public void onInit(int i) {  textToSpeech.setLanguage(Locale.*ENGLISH*);  }  });   playerList = new ArrayList<>();   //<---------- Intialization ------------------>  fn\_bar.setVisibility(View.*INVISIBLE*);  fn\_bar.removeView(show);   cards\_human.setVisibility(View.*INVISIBLE*);  cards\_cpu1.setVisibility(View.*INVISIBLE*);  cards\_cpu2.setVisibility(View.*INVISIBLE*);  cards\_cpu3.setVisibility(View.*INVISIBLE*);  cards\_cpu4.setVisibility(View.*INVISIBLE*);   text\_money\_cpu1.setText(String.*valueOf*(1000));  text\_money\_cpu2.setText(String.*valueOf*(1000));  text\_money\_cpu3.setText(String.*valueOf*(1000));  text\_money\_cpu4.setText(String.*valueOf*(1000));  text\_human\_money.setText(String.*valueOf*(1000));  text\_pot.setText("0");   cpu1 = new CPU\_Player("CPU1",1000);  cpu2 = new CPU\_Player("CPU2",1000);  cpu3 = new CPU\_Player("CPU3",1000);  cpu4 = new CPU\_Player("CPU4",1000);  human = new Player("You",1000);   pot=0;  dablaPlayer = 4;  amt\_call=0;  amt\_raise=0;  amt\_dablu=50;   deal.setOnClickListener(new View.OnClickListener() {  @TargetApi(Build.VERSION\_CODES.*M*)  @Override  public void onClick(View v) {   name\_human.setBackground(getDrawable(R.color.*black\_overlay*));  name\_cpu1.setBackground(getDrawable(R.color.*black\_overlay*));  name\_cpu2.setBackground(getDrawable(R.color.*black\_overlay*));  name\_cpu3.setBackground(getDrawable(R.color.*black\_overlay*));  name\_cpu4.setBackground(getDrawable(R.color.*black\_overlay*));   cpu1.clearHand();  cpu2.clearHand();  cpu3.clearHand();  cpu4.clearHand();  human.clearHand();   cpu1.isPlaying=true;  cpu2.isPlaying=true;  cpu3.isPlaying=true;  cpu4.isPlaying=true;  human.isPlaying=true;   put\_dablu();  turn = (dablaPlayer+1)%5;  playerList.clear();  for (int i=1;i<=5;i++){  int num = (dablaPlayer+i)%5;  switch (num){  case 0: playerList.add(human); break;  case 1: if (!cpu1.isOut) playerList.add(cpu1); break;  case 2: if (!cpu2.isOut) playerList.add(cpu2); break;  case 3: if (!cpu3.isOut) playerList.add(cpu3); break;  case 4: if (!cpu4.isOut) playerList.add(cpu4); break;  }  }   pot = amt\_dablu;  amt\_call = amt\_dablu;  amt\_raise = amt\_dablu+50;   deck = new Deck();   // dealing the cards...  for (int i = 1; i <= 15; i++) {  switch (i%5){  case 0: human.addCard(deck.dealCard()); break;  case 1: cpu1.addCard(deck.dealCard()); break;  case 2: cpu2.addCard(deck.dealCard()); break;  case 3: cpu3.addCard(deck.dealCard()); break;  case 4: cpu4.addCard(deck.dealCard()); break;  }  }   cpu1.sortCards();  cpu2.sortCards();  cpu3.sortCards();  cpu4.sortCards();  human.sortCards();   text\_pot.setText(String.*valueOf*(pot));  text\_money\_cpu1.setText(String.*valueOf*(cpu1.getMoney()));  text\_money\_cpu2.setText(String.*valueOf*(cpu2.getMoney()));  text\_money\_cpu3.setText(String.*valueOf*(cpu3.getMoney()));  text\_money\_cpu4.setText(String.*valueOf*(cpu4.getMoney()));  text\_human\_money.setText(String.*valueOf*(human.getMoney()));  text\_call\_amt.setText(String.*valueOf*(amt\_call));  text\_raise\_amt.setText(String.*valueOf*(amt\_raise));   if (!cpu1.isOut) {  cpu1\_card1.setText("??");  cpu1\_card2.setText("??");  cpu1\_card3.setText("??");  }  if (!cpu2.isOut) {  cpu2\_card1.setText("??");  cpu2\_card2.setText("??");  cpu2\_card3.setText("??");  }  if (!cpu3.isOut) {  cpu3\_card1.setText("??");  cpu3\_card2.setText("??");  cpu3\_card3.setText("??");  }  if (!cpu4.isOut) {  cpu4\_card1.setText("??");  cpu4\_card2.setText("??");  cpu4\_card3.setText("??");  }  player\_card1.setText(human.hand.get(0).toString());  player\_card2.setText(human.hand.get(1).toString());  player\_card3.setText(human.hand.get(2).toString());   deal.setVisibility(View.*INVISIBLE*);  cards\_human.setVisibility(View.*VISIBLE*);  if (!cpu1.isOut) cards\_cpu1.setVisibility(View.*VISIBLE*);  if (!cpu2.isOut) cards\_cpu2.setVisibility(View.*VISIBLE*);  if (!cpu3.isOut) cards\_cpu3.setVisibility(View.*VISIBLE*);  if (!cpu4.isOut) cards\_cpu4.setVisibility(View.*VISIBLE*);   if (playerList.get(0).equals(human)) {  fn\_bar.setVisibility(View.*VISIBLE*);  name\_human.setBackground(getDrawable(R.color.*green*));  }  else  playCPU(0);  }  });   */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* // <-------------- Now the Game Begins... ------------------>  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/* fold.setOnClickListener(new View.OnClickListener() {  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void onClick(View v) {  fn\_bar.setVisibility(View.*INVISIBLE*);  cards\_human.setVisibility(View.*INVISIBLE*);   human.isPlaying=false;  // textToSpeech.speak("fold",textToSpeech.QUEUE\_FLUSH,null,null);   name\_human.setBackground(getDrawable(R.color.*black\_overlay*));   if (playerList.size()==2){  TwoPlayerFold(playerList.get( (playerList.indexOf(human)+1)%2 ));  }  else {  int num;  if (playerList.indexOf(human) == playerList.size() - 1)  num = 0;  else  num = playerList.indexOf(human);   playerList.remove(human);  playCPU(num);  }  }  });  show.setOnClickListener(new View.OnClickListener() {  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void onClick(View v) {  pot += (amt\_call\*2); //pay twice the call amount to show  text\_pot.setText(String.*valueOf*(pot));  human.deductMoney(amt\_call \* 2);  text\_human\_money.setText(String.*valueOf*(human.getMoney()));   fn\_bar.setVisibility(View.*INVISIBLE*);  //textToSpeech.speak("show",textToSpeech.QUEUE\_FLUSH,null,null);   showCPUCards();   decideWinner();  }  });  call.setOnClickListener(new View.OnClickListener() {  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void onClick(View v) {  pot += amt\_call;  text\_pot.setText(String.*valueOf*(pot));  human.deductMoney(amt\_call);  text\_human\_money.setText(String.*valueOf*(human.getMoney()));   fn\_bar.setVisibility(View.*INVISIBLE*);  //textToSpeech.speak("call",textToSpeech.QUEUE\_FLUSH,null,null);   name\_human.setBackground(getDrawable(R.color.*black\_overlay*));  int num = (playerList.indexOf(human)+1)%playerList.size();  playCPU(num);  }  });  raise.setOnClickListener(new View.OnClickListener() {  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void onClick(View v) {  amt\_raise = Integer.*valueOf*(text\_raise\_amt.getText().toString());  amt\_call = amt\_raise;  pot += amt\_raise;  human.deductMoney(amt\_raise);  amt\_raise += 50;   text\_pot.setText(String.*valueOf*(pot));  text\_human\_money.setText(String.*valueOf*(human.getMoney()));  text\_call\_amt.setText(String.*valueOf*(amt\_call));  text\_raise\_amt.setText(String.*valueOf*(amt\_raise));   fn\_bar.setVisibility(View.*INVISIBLE*);  //textToSpeech.speak("raise",textToSpeech.QUEUE\_FLUSH,null,null);   name\_human.setBackground(getDrawable(R.color.*black\_overlay*));  int num = (playerList.indexOf(human)+1)%playerList.size();  playCPU(num);  }  });  minus.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  int amt2 = Integer.*valueOf*(text\_raise\_amt.getText().toString());  if (amt2-50>amt\_call)  amt2 -= 50;  text\_raise\_amt.setText(String.*valueOf*(amt2));  }  });  plus.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  int amt = Integer.*valueOf*(text\_raise\_amt.getText().toString());  if (amt+50<=human.getMoney())  amt += 50;  text\_raise\_amt.setText(String.*valueOf*(amt));  }  });  }   @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  void put\_dablu(){  switch (dablaPlayer){  case 0: human.deductMoney(amt\_dablu);  name\_human.setBackground(getDrawable(R.color.*crimsonRed*)); break;  case 1: cpu1.deductMoney(amt\_dablu);  name\_cpu1.setBackground(getDrawable(R.color.*crimsonRed*)); break;  case 2: cpu2.deductMoney(amt\_dablu);  name\_cpu2.setBackground(getDrawable(R.color.*crimsonRed*)); break;  case 3: cpu3.deductMoney(amt\_dablu);  name\_cpu3.setBackground(getDrawable(R.color.*crimsonRed*)); break;  case 4: cpu4.deductMoney(amt\_dablu);  name\_cpu4.setBackground(getDrawable(R.color.*crimsonRed*)); break;  }  }   boolean isShowAsked=false;  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  private void playCPU(final int start) {   Runnable runnable = new Runnable() {  @Override  public void run() {  int i = start;  while (!playerList.get(i).equals(human)){   CPU\_Player cpuPlayer = (CPU\_Player) playerList.get(i);  String playerName = cpuPlayer.getPlayer\_name();   Log.*i*("tag",playerName+"'s turn");   Message msg = new Message();  Bundle bundle = new Bundle();  bundle.putString("name",playerName);  msg.setData(bundle);   greenLightHandler.sendMessage(msg);   long futuretime = System.*currentTimeMillis*() + 2000;  while (System.*currentTimeMillis*() < futuretime){}   String decision = cpuPlayer.getTheBestDecision(playerList.size());  Log.*i*("tag",playerName+" decided to "+decision);   interpretDecision(playerName,decision);   futuretime = System.*currentTimeMillis*() + 500;  while (System.*currentTimeMillis*() < futuretime){}   if (decision.equals("show")) {  tempHandler.sendEmptyMessage(0);  return;  }   if (decision.equals("fold")) {  if (i==(playerList.size()+1)-1)  i=0;  else  i=i; //no change  }  else  i=(i+1)%playerList.size();   if (playerList.size()==1){  TwoPlayerFold(playerList.get(i));  return;  }   futuretime = System.*currentTimeMillis*() + 1000;  while (System.*currentTimeMillis*() < futuretime) {}   Log.*i*("tag","changing turn...");   Message msg2 = new Message();  Bundle bundle2 = new Bundle();  bundle2.putString("name",playerName);  msg2.setData(bundle2);   greyLightHandler.sendMessage(msg2);   futuretime = System.*currentTimeMillis*() + 200;  while (System.*currentTimeMillis*() < futuretime){}  }   humanTurnHandler.sendEmptyMessage(0);  }  };  Thread sidethread = new Thread(runnable);  sidethread.start();  }   Handler tempHandler = new Handler(){  @Override  public void handleMessage(Message msg) {  decideWinner();  }  };  Handler humanTurnHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  Log.*i*("tag","human turn");  fn\_bar.setVisibility(View.*VISIBLE*);  name\_human.setBackground(getDrawable(R.color.*green*));  if (playerList.size()==2 && !isShowEnabled) {  fn\_bar.addView(show);  isShowEnabled = true;  }  }  };   Handler greenLightHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  switch (name){  case "CPU1": name\_cpu1.setBackground(getDrawable(R.color.*green*)); break;  case "CPU2": name\_cpu2.setBackground(getDrawable(R.color.*green*)); break;  case "CPU3": name\_cpu3.setBackground(getDrawable(R.color.*green*)); break;  case "CPU4": name\_cpu4.setBackground(getDrawable(R.color.*green*)); break;  }  }  };   Handler greyLightHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  switch (name){  case "CPU1": name\_cpu1.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU2": name\_cpu2.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU3": name\_cpu3.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU4": name\_cpu4.setBackground(getDrawable(R.color.*black\_overlay*)); break;  }  }  };   Handler foldHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  Log.*i*("tag",name+" folds...");   switch (name){  case "CPU1": cards\_cpu1.setVisibility(View.*INVISIBLE*);  cpu1.isPlaying=false;  playerList.remove(cpu1); break;   case "CPU2": cards\_cpu2.setVisibility(View.*INVISIBLE*);  cpu2.isPlaying=false;  playerList.remove(cpu2); break;   case "CPU3": cards\_cpu3.setVisibility(View.*INVISIBLE*);  cpu3.isPlaying=false;  playerList.remove(cpu3); break;   case "CPU4": cards\_cpu4.setVisibility(View.*INVISIBLE*);  cpu4.isPlaying=false;  playerList.remove(cpu4); break;  }  //textToSpeech.speak("fold",textToSpeech.QUEUE\_FLUSH,null,null);  }  };  Handler showHandler = new Handler(){  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  Log.*i*("tag",name+" shows...");  switch (name){  case "CPU1": cpu1.deductMoney(amt\_call\*2);  text\_money\_cpu1.setText(String.*valueOf*(cpu1.getMoney())); break;  case "CPU2": cpu2.deductMoney(amt\_call\*2);  text\_money\_cpu2.setText(String.*valueOf*(cpu2.getMoney())); break;  case "CPU3": cpu3.deductMoney(amt\_call\*2);  text\_money\_cpu3.setText(String.*valueOf*(cpu3.getMoney())); break;  case "CPU4": cpu4.deductMoney(amt\_call\*2);  text\_money\_cpu4.setText(String.*valueOf*(cpu4.getMoney())); break;  }   pot += amt\_call\*2;  text\_pot.setText(String.*valueOf*(pot));   showCPUCards();  }  };  Handler callHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  Log.*i*("tag",name+" calls...");  switch (name){  case "CPU1": cpu1.deductMoney(amt\_call);  text\_money\_cpu1.setText(String.*valueOf*(cpu1.getMoney())); break;   case "CPU2": cpu2.deductMoney(amt\_call);  text\_money\_cpu2.setText(String.*valueOf*(cpu2.getMoney())); break;   case "CPU3": cpu3.deductMoney(amt\_call);  text\_money\_cpu3.setText(String.*valueOf*(cpu3.getMoney())); break;   case "CPU4": cpu4.deductMoney(amt\_call);  text\_money\_cpu4.setText(String.*valueOf*(cpu4.getMoney())); break;  }   pot += amt\_call;  text\_pot.setText(String.*valueOf*(pot));  }  };  Handler raiseHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  int raise\_amt = msg.getData().getInt("raise\_amt");  Log.*i*("tag",name+" raise...");  switch (name){  case "CPU1":  cpu1.deductMoney(amt\_call+raise\_amt);  text\_money\_cpu1.setText(String.*valueOf*(cpu1.getMoney())); break;   case "CPU2":  cpu2.deductMoney(amt\_call+raise\_amt);  text\_money\_cpu2.setText(String.*valueOf*(cpu2.getMoney())); break;   case "CPU3":  cpu3.deductMoney(amt\_call+raise\_amt);  text\_money\_cpu3.setText(String.*valueOf*(cpu3.getMoney())); break;   case "CPU4":  cpu4.deductMoney(amt\_call+raise\_amt);  text\_money\_cpu4.setText(String.*valueOf*(cpu4.getMoney())); break;  }   amt\_call = amt\_call+raise\_amt;  pot += amt\_call;   text\_pot.setText(String.*valueOf*(pot));  text\_raise\_amt.setText(String.*valueOf*(amt\_call+50));  text\_call\_amt.setText(String.*valueOf*(amt\_call));  }  };   public void interpretDecision(String name, String decision){  Message msg = new Message();  Bundle bundle = new Bundle();  bundle.putString("name",name);   switch (decision){  case "fold":  msg.setData(bundle);  foldHandler.sendMessage(msg);  break;   case "show":  msg.setData(bundle);  showHandler.sendMessage(msg);  break;   case "call":  msg.setData(bundle);  callHandler.sendMessage(msg);  break;   case "raise,50":  bundle.putInt("raise\_amt",50);  msg.setData(bundle);  raiseHandler.sendMessage(msg);  break;   case "raise,100":  bundle.putInt("raise\_amt",100);  msg.setData(bundle);  raiseHandler.sendMessage(msg);  break;  }  }   Handler OnOffHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  int signal = msg.getData().getInt("signal");   if (signal==1) {  switch (name){  case "You": name\_human.setBackground(getDrawable(R.color.*colorAccent*)); break;  case "CPU1": name\_cpu1.setBackground(getDrawable(R.color.*colorAccent*)); break;  case "CPU2": name\_cpu2.setBackground(getDrawable(R.color.*colorAccent*)); break;  case "CPU3": name\_cpu3.setBackground(getDrawable(R.color.*colorAccent*)); break;  case "CPU4": name\_cpu4.setBackground(getDrawable(R.color.*colorAccent*)); break;  }  }else {  switch (name){  case "You": name\_human.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU1": name\_cpu1.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU2": name\_cpu2.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU3": name\_cpu3.setBackground(getDrawable(R.color.*black\_overlay*)); break;  case "CPU4": name\_cpu4.setBackground(getDrawable(R.color.*black\_overlay*)); break;  }  }  }  };   public void TwoPlayerFold(final Player winner){  Runnable runnable = new Runnable() {  @Override  public void run() {  long futuretime = System.*currentTimeMillis*() + 500;  while (System.*currentTimeMillis*() < futuretime){}   winner.addMoney(pot);   Message message = new Message();  Bundle bundle = new Bundle();  bundle.putString("name",winner.getPlayer\_name());  message.setData(bundle);  winnerHandler.sendMessage(message);   Blink(winner.getPlayer\_name());   futuretime = System.*currentTimeMillis*() + 1500;  while (System.*currentTimeMillis*() < futuretime) {}   endings.sendEmptyMessage(0);  }  };  Thread endThread = new Thread(runnable);  endThread.start();  }   public void showCPUCards(){  if (cpu1.isPlaying) {  cpu1\_card1.setText(cpu1.hand.get(0).toString());  cpu1\_card2.setText(cpu1.hand.get(1).toString());  cpu1\_card3.setText(cpu1.hand.get(2).toString());  }  if (cpu2.isPlaying) {  cpu2\_card1.setText(cpu2.hand.get(0).toString());  cpu2\_card2.setText(cpu2.hand.get(1).toString());  cpu2\_card3.setText(cpu2.hand.get(2).toString());  }  if (cpu3.isPlaying) {  cpu3\_card1.setText(cpu3.hand.get(0).toString());  cpu3\_card2.setText(cpu3.hand.get(1).toString());  cpu3\_card3.setText(cpu3.hand.get(2).toString());  }  if (cpu4.isPlaying) {  cpu4\_card1.setText(cpu4.hand.get(0).toString());  cpu4\_card2.setText(cpu4.hand.get(1).toString());  cpu4\_card3.setText(cpu4.hand.get(2).toString());  }  }   public void decideWinner() {   Runnable runnable = new Runnable() {  @Override  public void run() {  Player winner = null;   int maxRank=-1;  for (int i=0;i<playerList.size();i++)  if (playerList.get(i).getRank() > maxRank)  maxRank = playerList.get(i).getRank();   ArrayList<Player> finalist = new ArrayList<>();  for (int i=0;i<playerList.size();i++)  if (playerList.get(i).getRank()==maxRank)  finalist.add(playerList.get(i));   if (finalist.size()>1){  switch (maxRank){  case 2:  Collections.*sort*(finalist,new sortByPairCard());  winner = finalist.get(0);  break;  default:  Collections.*sort*(finalist,new sortByHighCard());  winner = finalist.get(0);  }  }else {  winner = finalist.get(0);  }   long futuretime = System.*currentTimeMillis*() + 2000;  while (System.*currentTimeMillis*() < futuretime){}   winner.addMoney(pot);   Message message = new Message();  Bundle bundle = new Bundle();  bundle.putString("name",winner.getPlayer\_name());  message.setData(bundle);  winnerHandler.sendMessage(message);   Blink(winner.getPlayer\_name());   futuretime = System.*currentTimeMillis*() + 2000;  while (System.*currentTimeMillis*() < futuretime) {}   endings.sendEmptyMessage(0);  }  };   Thread endThread = new Thread(runnable);  endThread.start();  }   Handler winnerHandler = new Handler(){  @TargetApi(Build.VERSION\_CODES.*LOLLIPOP*)  @Override  public void handleMessage(Message msg) {  String name = msg.getData().getString("name");  switch (name){  case "You": text\_human\_money.setText(String.*valueOf*(human.getMoney())); break;  case "CPU1": text\_money\_cpu1.setText(String.*valueOf*(cpu1.getMoney())); break;  case "CPU2": text\_money\_cpu2.setText(String.*valueOf*(cpu2.getMoney())); break;  case "CPU3": text\_money\_cpu3.setText(String.*valueOf*(cpu3.getMoney())); break;  case "CPU4": text\_money\_cpu4.setText(String.*valueOf*(cpu4.getMoney())); break;  }   String message;  if (name.equals("You"))  message = "You won!";  else  message = name+" wins.";  textToSpeech.speak(message,TextToSpeech.*QUEUE\_FLUSH*,null,null);  }  };   private class sortByHighCard implements Comparator<Player>{   @Override  public int compare(Player A, Player B) {  if (B.hand.get(0).getCardNum() != A.hand.get(0).getCardNum())  return B.hand.get(0).getCardNum()-A.hand.get(0).getCardNum();   if (B.hand.get(1).getCardNum() != A.hand.get(1).getCardNum())  return B.hand.get(1).getCardNum()-A.hand.get(1).getCardNum();   if (B.hand.get(2).getCardNum() != A.hand.get(2).getCardNum())  return B.hand.get(2).getCardNum()-A.hand.get(2).getCardNum();   return 0;  }  }   private class sortByPairCard implements Comparator<Player>{   @Override  public int compare(Player A, Player B) {  if (B.hand.get(1).getCardNum() != A.hand.get(1).getCardNum())  return B.hand.get(1).getCardNum()-A.hand.get(1).getCardNum();   if (B.hand.get(0).getCardNum() != A.hand.get(0).getCardNum())  return B.hand.get(0).getCardNum()-A.hand.get(0).getCardNum();   if (B.hand.get(2).getCardNum() != A.hand.get(2).getCardNum())  return B.hand.get(2).getCardNum()-A.hand.get(2).getCardNum();   return 0;  }  }   public void Blink(final String name){  Runnable blinker = new Runnable() {  @Override  public void run() {  for (int i=1;i<=11;i++){  Message message = new Message();  Bundle bundle = new Bundle();  bundle.putString("name",name);  bundle.putInt("signal",i%2);  message.setData(bundle);  OnOffHandler.sendMessage(message);   long waitTime = System.*currentTimeMillis*() + 250;  while (System.*currentTimeMillis*()<waitTime) {}  }  }  };  Thread sideThread = new Thread(blinker);  sideThread.start();  }   Handler endings = new Handler() {  @Override  public void handleMessage(Message msg) {   pot = 0;  amt\_dablu = 50;  amt\_call = amt\_dablu;   text\_pot.setText(String.*valueOf*(pot));  text\_call\_amt.setText(String.*valueOf*(amt\_call));  text\_raise\_amt.setText(String.*valueOf*(amt\_call + 50));   fn\_bar.removeView(show);  isShowEnabled = false;   int noOfPlayer = 5;  if (human.getMoney() <= 0) {  Intent intent = new Intent(Table.this, MainActivity.class);  startActivity(intent);  }  if (cpu1.getMoney() <= 0) {  cpu1.isOut = true;  noOfPlayer--;  }  if (cpu2.getMoney() <= 0) {  cpu2.isOut = true;  noOfPlayer--;  }  if (cpu3.getMoney() <= 0) {  cpu3.isOut = true;  noOfPlayer--;  }  if (cpu4.getMoney() <= 0) {  cpu4.isOut = true;  noOfPlayer--;  }   dablaPlayer = (dablaPlayer + 1) % noOfPlayer;   deal.setVisibility(View.*VISIBLE*);  }  };   public void onPause() {  if (textToSpeech != null) {  textToSpeech.stop();  textToSpeech.shutdown();  }  super.onPause();  } } |

## **Conclusion**

This project is currently a prototype made on card game TEEN PATTI. Implementation was based on CPU AI and Single player mode. Soft copy can be provided if required by analyzer. Also this project involves **text to speech feature** which is implemented on when user or CPU wins.