

Projekat iz predmeta: Soft computing

PokerBot

Profesor:

doc. dr. Đorđe Obradović

Asistent:

Miroslav Kondić

Projektni tim:

Nikola Stokić RA1/2012

Branislav Vezilić RA34/2012

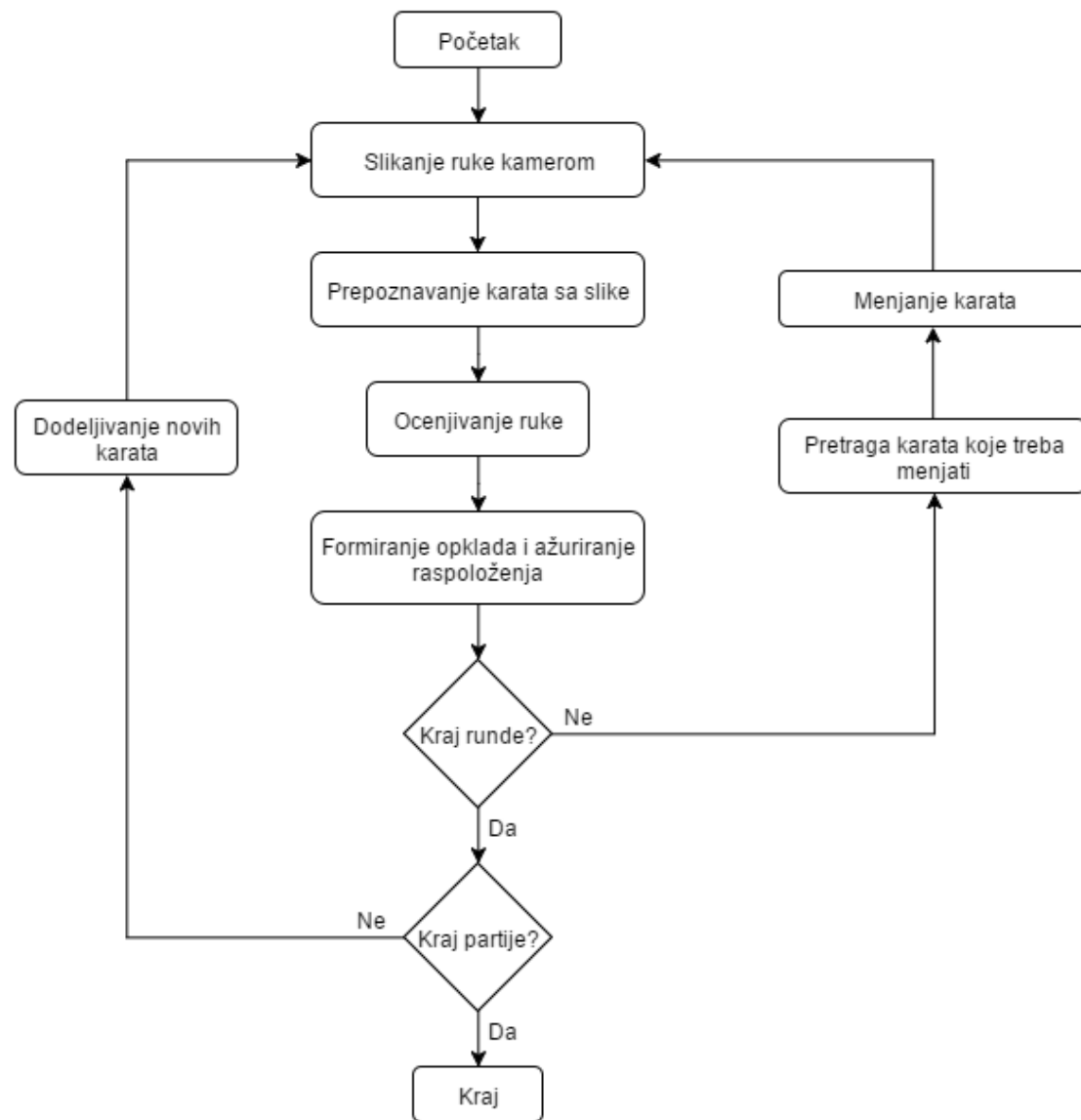
PokerBot i Motivacija

- ▶ PokerBot predstavlja softver koji treba da u što većoj meri igra poker kao što bi čovek to radio
- ▶ Poker je igra koja sadrži određene rizike i nesigurnosti kao i mogućnost lažnih informacija zbog kojih je teško predvideti najbolji potez
- ▶ Cilj:
 - Napraviti veštačku inteligenciju koja će biti u stanju da pobedi čoveka
 - Omogućiti početnicima da bolje nauče i razumeju igru poker
 - Moguća softverska podloga za robota

Zadatak

- ▶ Implementirati AI za bot-a koji će na osnovu OCR-a, neuronskih mreža, stabla odlučivanja, verovatnoća i algoritama pretrage igrati poker
- ▶ Prepoznavanje karata sa slike
- ▶ Ocenjivanje vrednosti ruke
- ▶ Pretraga karata koje treba izmeniti
- ▶ Formiranje opklada
- ▶ Prikazivanje raspoloženja
- ▶ Mogućnost blefiranja

Tok izvršavanja programa



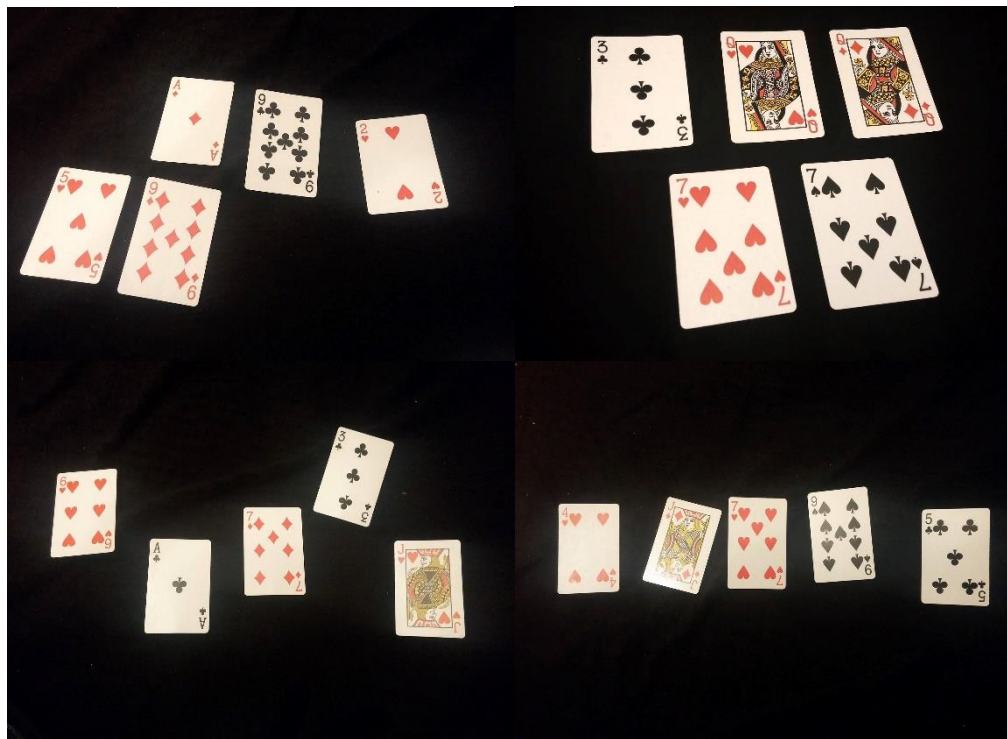
OCR - Prepoznavanje karata

► Cilj:

- Prepoznati 5 uspravno postavljenih karata na crnoj pozadini

► Uslovi:

- Karte ne smeju da se preklapaju
- Na slici mora biti prikazana cela karta
- Poželjna što veća rezolucija slike



OCR - Algoritam

- ▶ Učitavanje slike
- ▶ Primena Canny Edge detektora
- ▶ Pronalaženje karata sa slike
- ▶ Transformacija karte iz perspektive
- ▶ Određivanje vrednosti i tipa karte
- ▶ Obučavanje neuronske mreže
- ▶ Prosleđivanje podataka neuronskoj mreži

OCR - Canny Edge

► Canny Edge detektor predstavlja algoritam iz više faza:

1. Otklanjanje šumova korišćenjem Gausovog zamućenja tako što se za svaki piksel slike primenjuje Gausova funkcija u obliku matrice (kernel) 5x5 koja u centru ima najveću vrednost i opada sa udaljenošću od centra.

2. Određivanje gradijenta slike i ivica preko Sobelovog operatora. Za svaki piksel slike se primenjuju dva kernela G_x i G_y , a konačna vrednost piksela se izračunava kao

$$G = \sqrt{G_x^2 + G_y^2}.$$

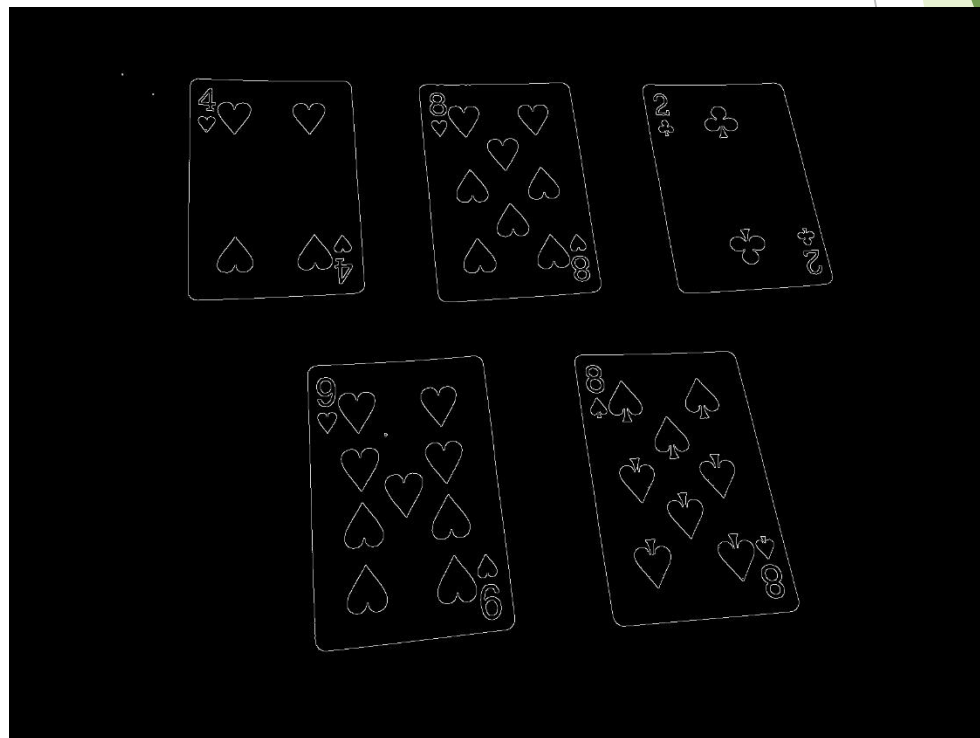
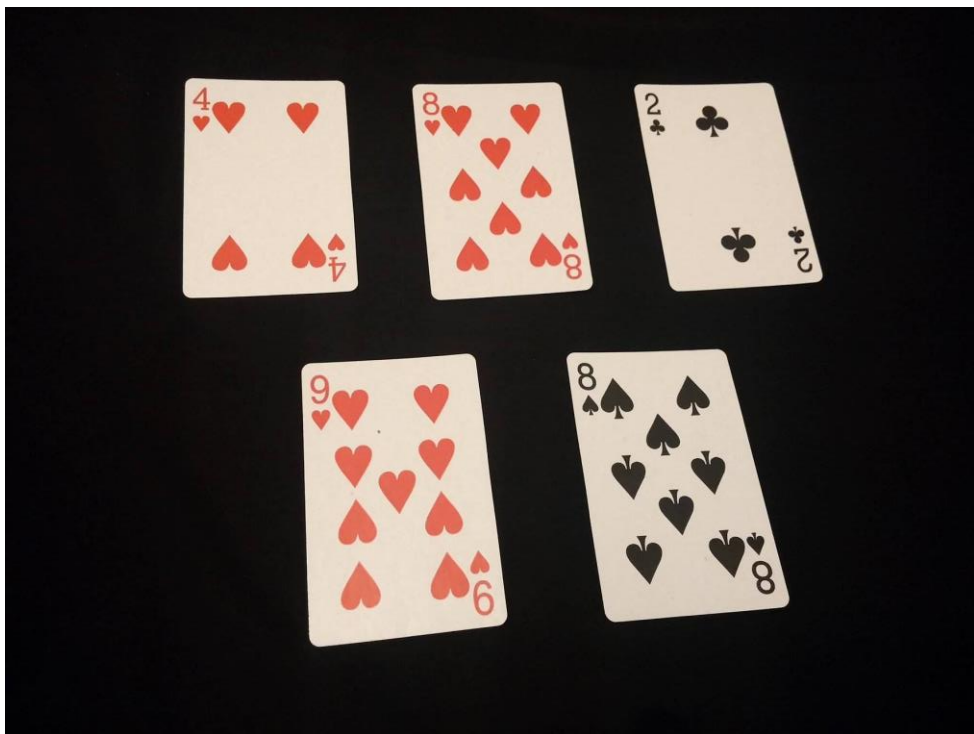
3. Non-maximum suppression je tehnika stanjivanja ivica, tako što se pretražuje slika u pravcu gradijenta i svi pikseli koji ne pripadaju maximumu se postavljaju na 0.

4. Odredjivanje dva thresholda (gornji i donji):

- Ako je gradient piksela iznad gornje granice, piksel se uzima za ivicu
- Ako je gradient piksela ispod donje granice, piksel se odbacuje
- Ako je između granica, onda se piksel uzima za ivicu ako je povezan sa nekim pikselom iznad gornje granice

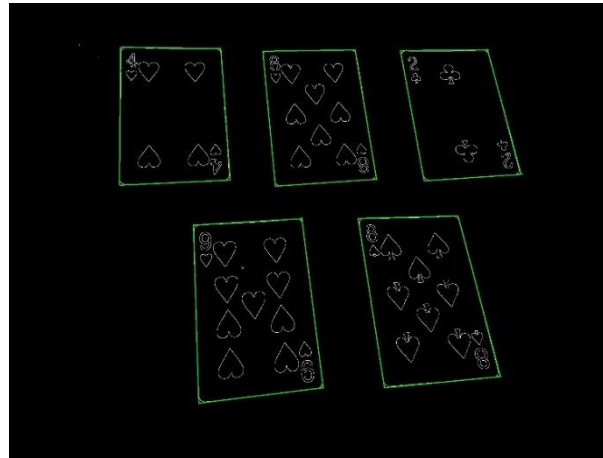
OCR - Canny Edge

- Primer slike na kojoj je primenjen Canny Edge



OCR - Pronalaženje karata sa slike

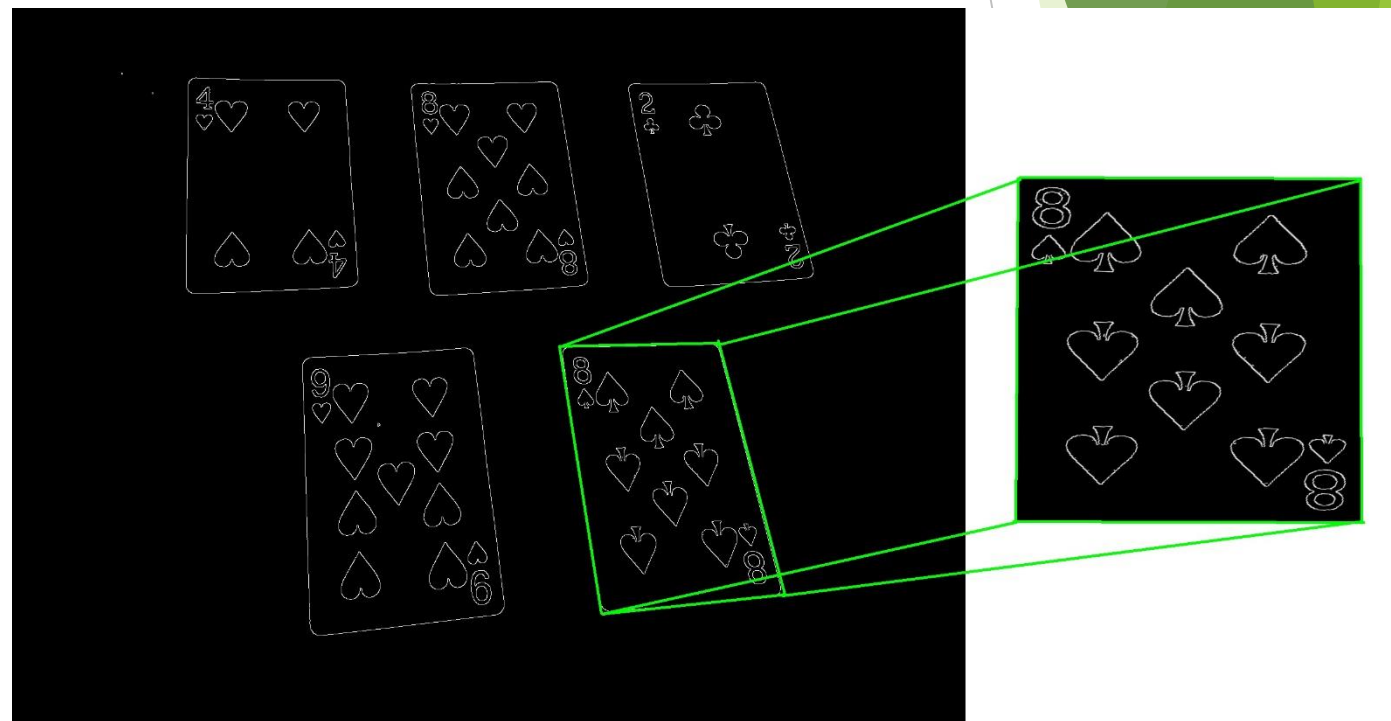
- ▶ Pronalažanje svih spoljašnjim kontura
- ▶ Sortiranje po veličini
- ▶ Sortiranje sa leva na desno po redovima



```
def select_roi(image_orig, img_bin, cardtype, numcards=5):  
    img, contour_borders, hierarchy = cv2.findContours(img_bin.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    contours = []  
    contour_borders = sorted(contour_borders, key=cv2.contourArea, reverse=True)[:numcards]  
    contour_borders = sorted(contour_borders, key=distance, reverse=False)  
    img = cv2.drawContours(image_orig, contour_borders, -1, (0,255,0), 2)
```

OCR - Transformacija

- ▶ Zbog načina na koji se vrši slikanje, karte su uvek poređanje pod nekim uglom
- ▶ Kako bi nastavili dalje prepoznavanje moramo ih pretvoriti u format koji je nama pogodniji za obradu



OCR - Transformacija

- ▶ Vrš se aproksimovanje dužinom poligona i pronalaze koordinate temena
- ▶ Kreiranje nove slike 640x640 i određivanje korespondentnih tačaka koja će predstavljati transformisanu kartu

```
for contour in contour_borders:
    epsilon = 0.1*cv2.arcLength(contour,True)
    approx = cv2.approxPolyDP(contour,epsilon,True)
    if len(approx) != 4: # ukoliko nije 4, znaci da nije karta u pitanju
        continue
    card = rectify(approx)
    size = 640
    new = np.array([ [0,0],[size,0],[size,size],[0,size] ],np.float32)
    transform = cv2.getPerspectiveTransform(card,new)
    warp = cv2.warpPerspective(img_bin,transform,(size,size))
    suit = select_suit(warp,size,cardtype)
    contours.append(suit)
```

OCR - Određivanje vrednosti i tipa karte

- ▶ Pretraživanje spoljašnjih kontura u gornjem levom uglu
- ▶ Traže se dve konture, jedna koja će predstavljati vrednost karte i druga koja će predstavljati njen tip
- ▶ Slikama se menja veličina u 28x28



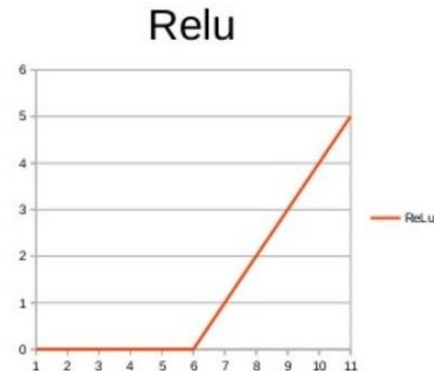
```
def select_suit(card, size, cardtype):  
    img, contour_borders, hierarchy = cv2.findContours(card.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
    contours = []  
    for contour in contour_borders:  
        x,y,w,h = cv2.boundingRect(contour) #koordinata i velicina granicnog pravougaonika  
        if size/6.4 > h > size/42.6 and size/6.4 > w > size/42.6 and x < size/3.2:  
            region = card[y:y+h+1,x:x+w+1]  
            contours.append(image_bin(resize_region(region)))  
  
    if len(contours) < 2:  
        #print 'NISAM USPEO DA NADJEM DVE KONTURE!'  
        return np.zeros((28,28))  
  
    return contours[cardtype]
```

OCR - Neuronska mreža

► Model:

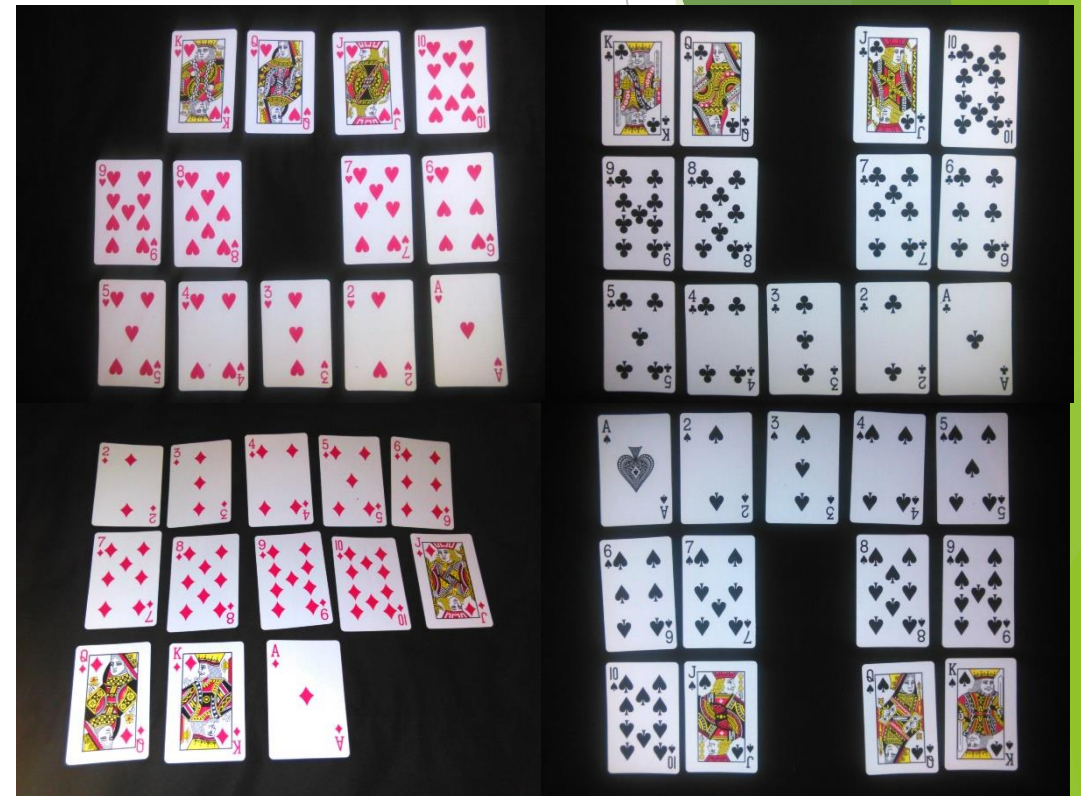
- Za prepoznavanje karata su korišćene dve neuronske mreže, jedna za prepoznavanje tipa karte (karo, tref,...), a druga za vrednost (2, 3, K,...)
- Ulaz u neuronsku mrežu predstavlja slika 28x28 tj. 784 neurona za svaki piksel.
- Jedan skriveni sloj sa 128 neurona.
- Izlaz za tip karte iznosi 4, dok za vrednost karte iznosi 13.
- Kako bi se sprečilo preobučavanje neuronske mreže, postoji 50% šansa da će težina biti postavljena na 0.

```
model = Sequential()  
model.add(Dense(128, input_dim=784, init='glorot_uniform'))  
model.add(Activation('relu'))  
model.add(Dropout(0.5))  
model.add(Dense(128, init='glorot_uniform'))  
model.add(Activation('relu'))  
model.add(Dropout(0.5))  
model.add(Dense(4, init='glorot_uniform'))  
model.add(Activation('relu'))
```



OCR - Neuronska mreža

- ▶ Za obučavanje neuronske mreže su korišćene 4 slike, uslikane mobilnom kamerom od 5MP.
- ▶ Svaki tip karte (karo, srce, pik, tref) sadrži po 13 primera, dok vrednost karte (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A) sadrži samo po 4 primera.

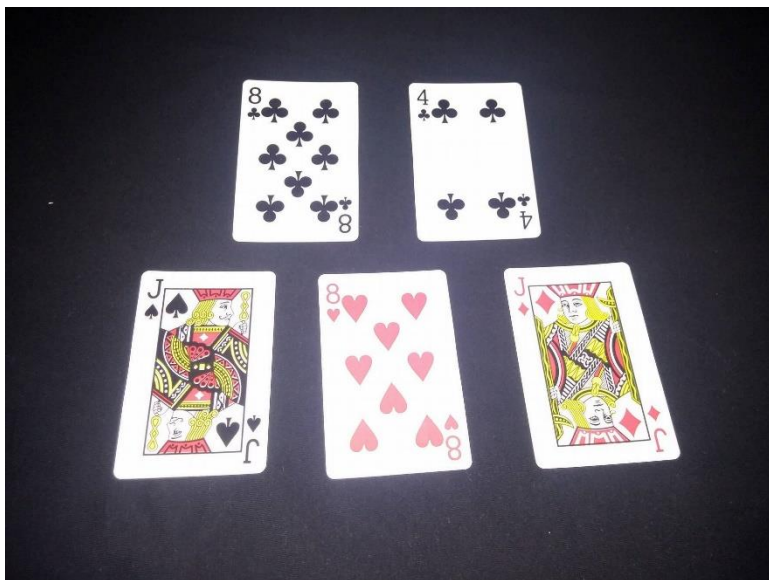


```
# definisanje parametra algoritma za obucavanje
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
ann.compile(loss='mean_squared_error', optimizer=sgd)

# obucavanje neuronske mreze
ann.fit(X_train, y_train, nb_epoch=50, batch_size=1, verbose = 1, shuffle=False, show_accuracy = True)
```

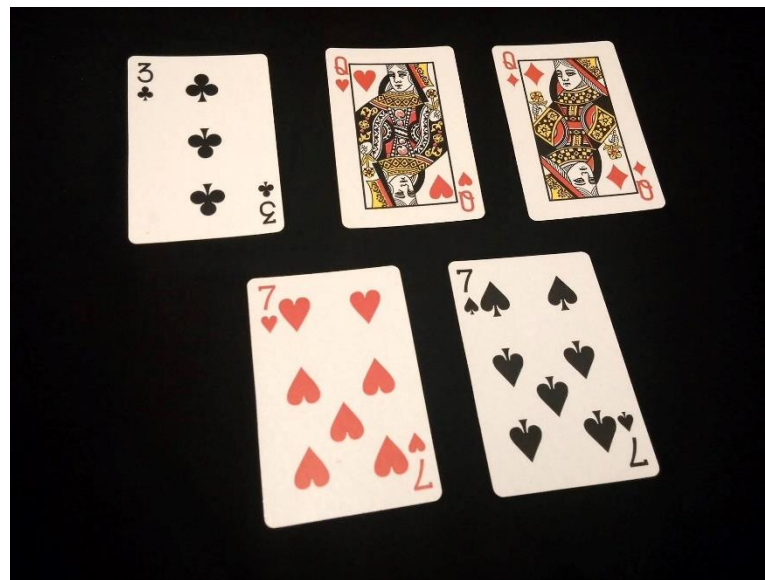
OCR - Rezultati

► Test 1



► Dobijeno: 8,C 4,C J,S 8,H J,D

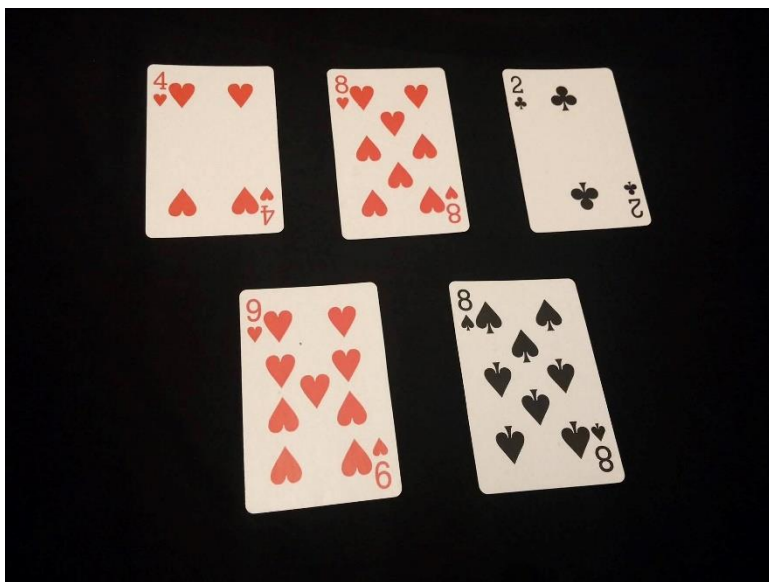
► Test 2



► Dobijeno: 3,C ?,? Q,D 7,H 7,S

OCR - Rezultati

► Test 3



► Dobijeno: 4,H 8,H 2,C 9,H 8,S

► Test 4



► Dobijeno: K,H K,D 5,H 8,C A,H

OCR - Rezultati

► Test 5



► Dobijeno: 10,S J,C 5,S 9,C 4,D

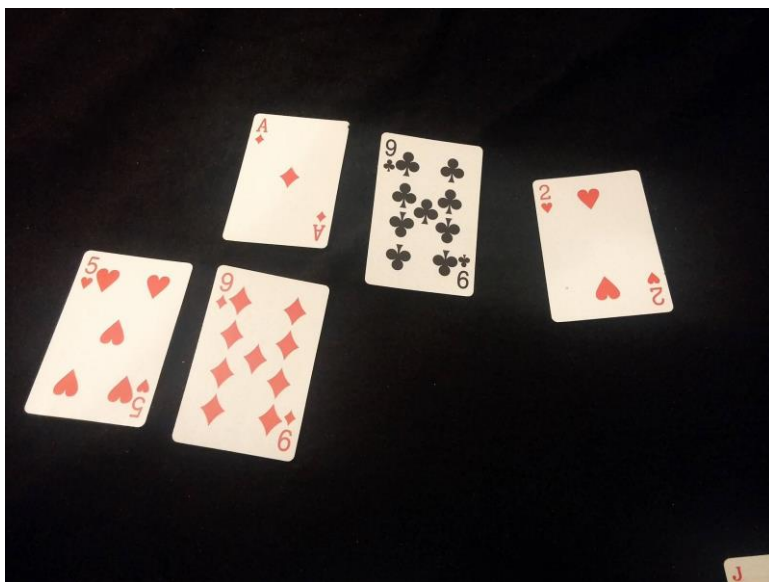
► Test 6



► Dobijeno: A,C 9,S 4,S 6,D ?,?

OCR - Rezultati

► Test 7



► Dobijeno: A,D 9,C 2,H 5,H 9,D

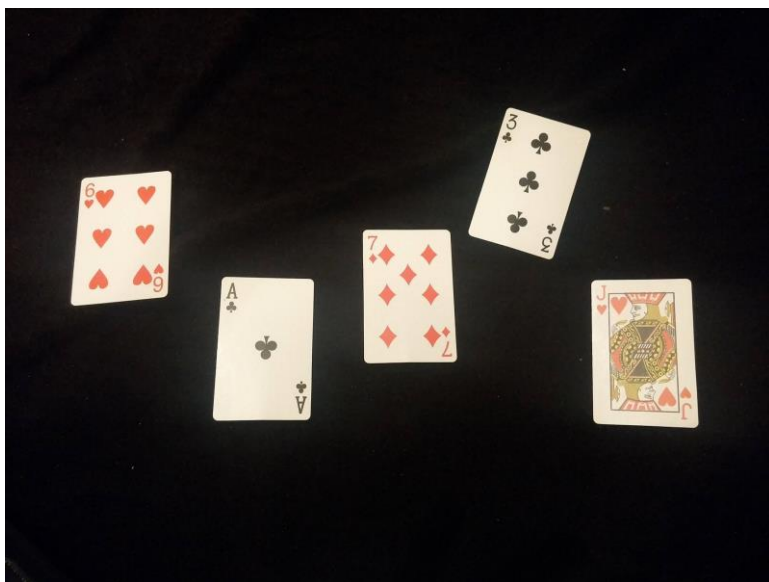
► Test 8



► Dobijeno: 4,H 7,D 9,S 7,H 5,C

OCR - Rezultati

► Test 9



► Dobijeno: 3,C 6,H 7,D A,C J,H

► Test 10



► Dobijeno: 7,S ?,? 8,S A,H 3,H

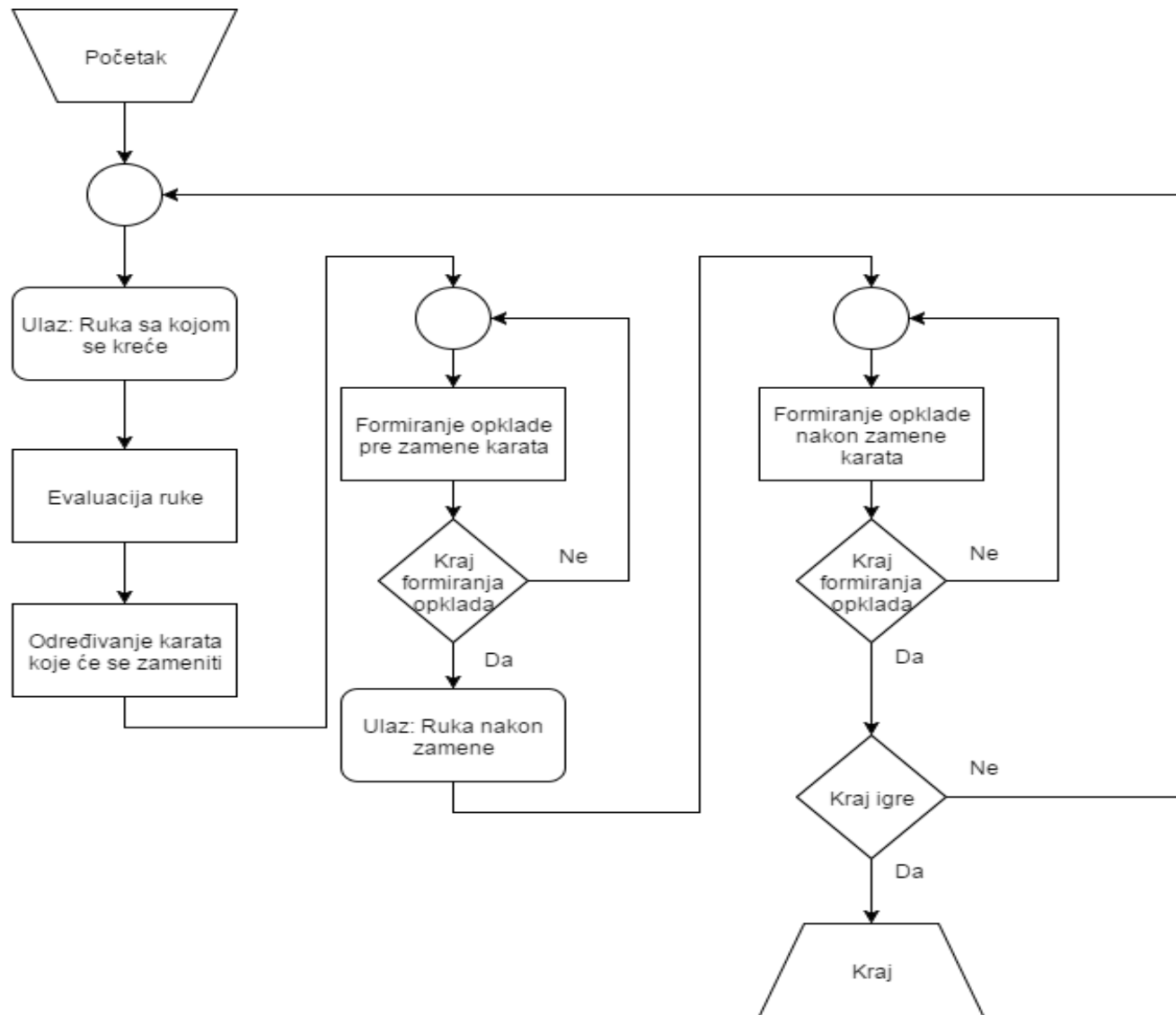
OCR - Zaključak

- ▶ U prethodnih 10 test primera (50 karata) dobijeno je:
 - Uspešnost pronalaska regiona $47/50$ karata = 0.94%
 - Greška pronalaska regiona $3/50$ = 0.06%
 - Uspešnost prepoznavanja neuronske mreže $46/47$ = 0.98%
 - Greška prepoznavanja neuronske mreže $1/47$ = 0.02%
- ▶ Poboljšanja:
 - Bolje prepoznavanje regiona na slikama manje rezolucije
 - Više primera za obučavanje neuronske mreže
 - Bolje otklanjanje šumova, poput senki i svetla
 - Eventualno ukljanjanje crne pozadine

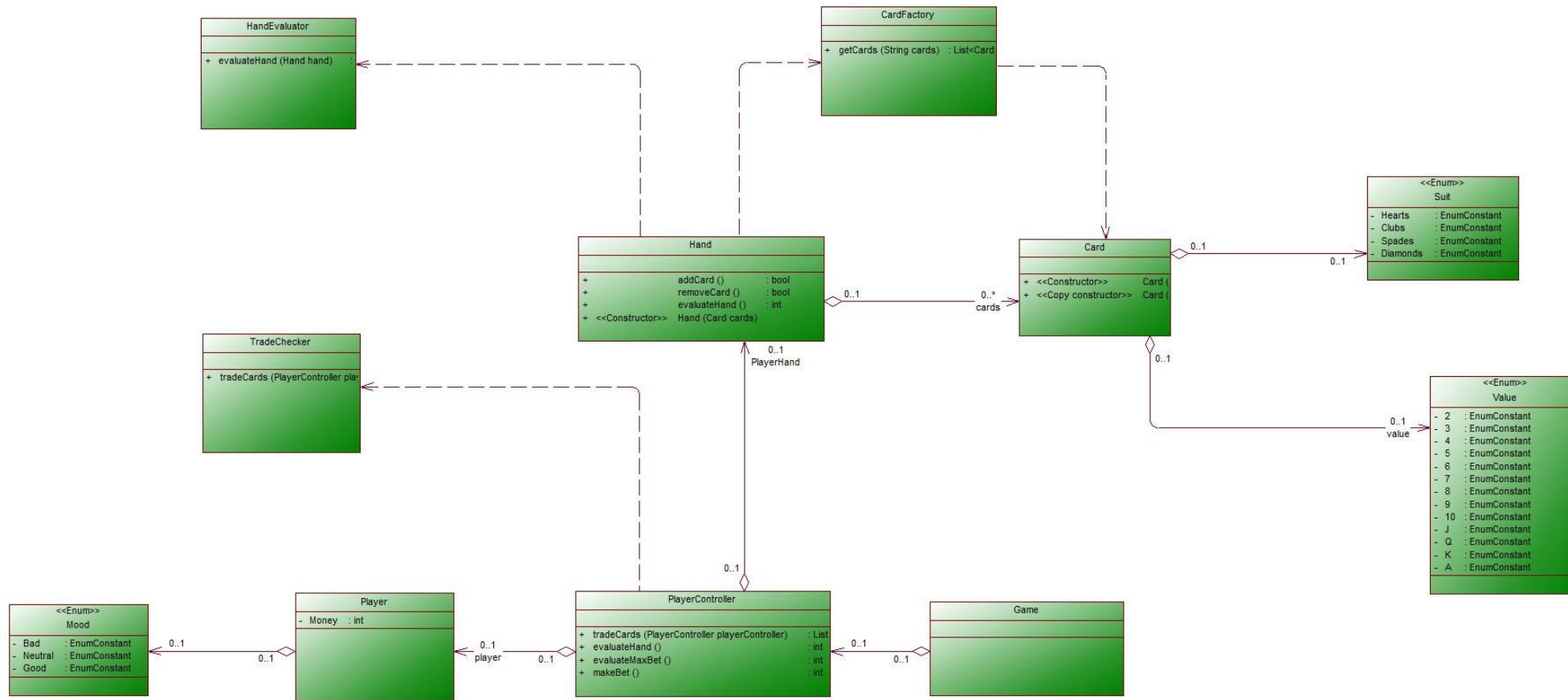
AI - Ideja

- ▶ Kreirati bota sposobnog da za datu ruku (karte):
 - Odrediti koje karte je neophodno promeniti
 - Formirati opklade

AI - Algoritam





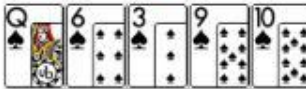







AI - Diagrama klasa



AI - Evaluacija karata

- ▶ Kako bi bot imao ikakvu svest o tome šta treba da igra i generalno koliko jaku ruku ima, mora se realizovati evaluacija iste.
- ▶ Vrednosti karata: 2, 3, 4, 5, 6, 7, 8, 9, 10, J11, Q12, K13, A14
- ▶ Vrednosti ruka: 0-9 respektivno
- ▶ Vrednost ruke je sedmocifren broj koji se formira na sledeći način.

Royal Flush	
Straight Flush	
Four of a Kind	
Full House	
Flush	
Straight	
Three of a Kind	
Two Pairs	
One Pair	
No Hand	

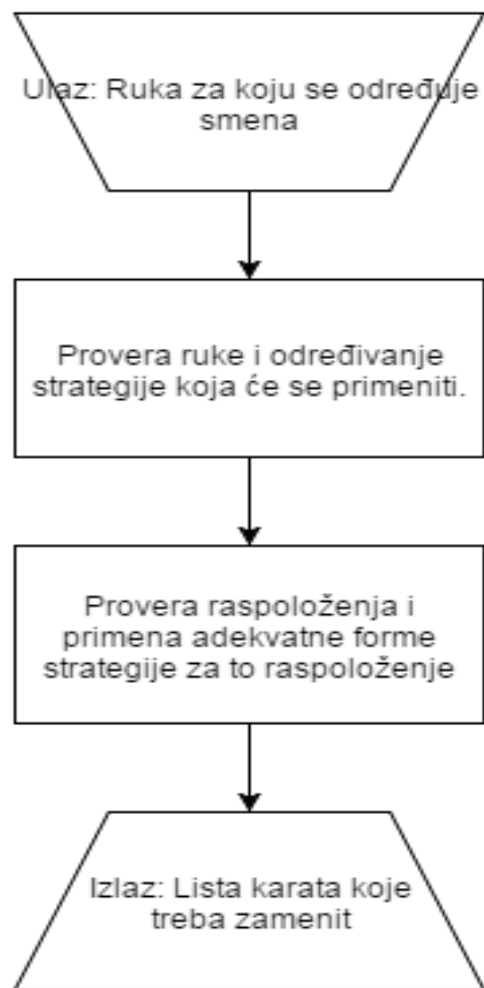
AI - Evaluacija karata

2 24 20 14

- ▶ Označava snagu ruke gde je 0 High Card, a 9 Royal Flush
 - ▶ Suma karata, veća (onih koje čine jednu grupu, veća od dve grupe ili suma u Straight-u)
 - ▶ Suma karata, manja (manja od dve grupe)
 - ▶ Vrednost High Card-a (za slučaj da je ima)
-
- ▶ Navedeni primer predstavlja dva para (Two Pairs = 2), dame (Q=12, $12*2=24$), desetke ($10*2=20$) i high card keca (A=14).

AI - Smena karata

- Agloritam za smenu karata se svodi na stablo odlučivanja. Gde se, u zavisnosti od ruke i raspoloženja, formira lista karata koje bot želi da zameni.



AI - Formiranje opklada

- ▶ Takođe algoritam zasnovan na stablu odlučivanja. Naime, formira se određeni modifikator (Bet Modifier) pomoću kog algoritam određuje koju će opkladu bot formirati (Fold, Call, Raise).
- ▶ Pre početka formiranja opklade (ili odgovora na protivnikovu) bot formira maksimalnu opkladu koja mu služi kao referenca do koliko je novaca spreman da da za određeno raspoloženje i koliko snagu ruke ima.
- ▶ U formiranju Bet modifikatora učestvuju:
 - Modifikator raspoloženja
 - Modifikator razlike novca
 - Modifikatori trenutne opklade (pre i posle smene karata)
 - Modifikator jačine ruke

AI - Moguća unapređenja

► Moguće je dodatno implementirati:

- Funkcionalnost da bot blefira. Ona bi se svodila na verovatnoću da on formira blef koja bi zavisila od toga koliko on novaca ima na raspolaganju, jačinu ruke, respoloženje i to šta je protivnik odigrao (praćenje ovoga je dosta teško implementirati).
- Funkcionalnost da bot pročita (odgovori) na protivnikov blef. Ovo bi opet uzimalo u obzir novac bota kao i snagu njegove ruke. Pored toga bi morao da sa dovoljnom preciznošću oceni iz protivnikove opklade kakvu ruku se on pravi da ima i da uspe da oceni kakvu ruku bi zapravo isti mogao da ima kako bi uspeo da izvede procenu. Takođe bi trebao i da prati koje je karte protivnik menjao kako bi dodatno (preciznije) mogao da oceni koju ruku je imao i koju ruku je pokušao da dobije kao i šansu da je dobije.

Slična rešenja u domenu problema

- ▶ **Computer Poker Research Group (University of Alberta, Canada)**
 - Cepheus - <http://poker.srv.ualberta.ca/>
- ▶ **The University of Auckland Game AI Group**
 - Sartre - <https://www.cs.auckland.ac.nz/poker/>
- ▶ **Neo Poker Laboratory**
 - Neo Poker Bot - <http://neopokerbot.com/>