SBNZ predlog projekta – Poker bot

Student: Danijel Radulović

Broj indeksa: SW-23/2016

Motivacija

Poker, odnosno igra kartama, je “igra nesavršenog znanja”, u kojoj učestvuje više takmičara, koji moraju razumeti procenu, predviđanje, upravljanje rizikom i čitanje protivničke igre, samim tim, predstavlja dobar primer sistema, u kome je potrebno doneti odluke u uslovima nesigurnosti i sistema, koji bi se mogao modelovati određenim pravilima.

Pregled problema

Problem, koji se rešava ovim projektom, predstavlja implementaciju bota, koji bi se mogao koristiti za igranje pokera. Njegovo ponašanje u različitim situacijama, u kojima može da se nađe tokom igre, bilo bi modelovano pravilima, zasnovanim na ekspertskom znanju. Tokom prethodnih dvadeset godina, pravljeni su poker botovi primenjujući razne metode, kao što su stabla igre, Monte Karlo simulacija, duboke neuronske mreže itd. Konkretno, performanse ovog bota bi se testirale pomoću PokerAcademy softvera, koji predstavlja platformu, na kojoj poker botovi mogu da igraju partije između sebe (<https://code.google.com/archive/p/opentestbed/source/default/source>), a kao literaturu bih koristio master rad sa Univerziteta u Alberti, na temu Dealing with Imperfect Information in Poker (<http://poker.cs.ualberta.ca/publications/papp.msc.pdf>).  
  
Metodologija rada

*Ulazi:* Vrednosti ulaza variraju na početku, a i u toku svake „ruke“ (partije) tačnije pri svakom menjanju faze jedne partije. U ulaze spadaju faza igre (*pre-flop, flop, turn, river*), broj igrača, pozicija bota za stolom, *hole cards* (karte koje je bot dobio), *board* (karte, koje su izašle), *pot* (suma novca ili čipova, koja se nalazi u igri), *bet/raise* (suma novca ili čipova, koje bi bot trebao da plati, ukoliko želi da nastavi igru), *blind* (suma, koju na početku partije moraju da plate dva igrača, zavisno od pozicije za stolom). Na osnovu ovih ulaza, bi se računale razne heuristike, koje bi se dalje koristile u pravilima.

*Izlazi:* Akcija, koju bi bot trebalo da sprovede

U prilogu je jednostavan primer, ukoliko je ukupno dva igrača i bot ima dobru „ruku“ (dobre početne karte).

Konkretan primer:

*Ulazi:* Faza *flop* (3 karte na terenu), ukupno dva igrača računajući i bota, prvi igrač je *check*-ovao, *hole cards* su *As Kh*, na *board*-u se nalaze *Ah 6s 9c*, *pot* 8, *blind* 4.

*Izlazi:* Akcija - *bet*, suma - 12 (3\**blind*)

Primer baze znanja: ulančana pravila

*Top pair* (*As Ah)* -> treba da igraš dalje, tj. da *call*-uješ/*bet-*uješ

Protivnik je *check*-ovao -> treba da *bet*-uješ

Faza *flop* -> *bet* 3\**blind*

Pravila implementirana u projektu

*Preflop rules* grupa pravila - Radna memorija sadrži *PlayerState* (trenutno stanje bota) i *Thresholds* koji sadrži razne *threshold* parametre, kao globalne promenljive prosleđeni su broj igrača koji bi još trebalo da odigraju tekuću ruku (*numOfPlayersToAct)*, *flag* koji označava tekuću fazu (*phase*) i podaci o igri (*GameInfo*). Pravila imaju dinamički *salience* u odnosu na fazu igre. Pravilo 1 služi za određivanje činjenice da li su karte iste boje, zatim pravila 8-16 služe za postavljanje određenih *threshold*-a, koji se računaju na osnovu različitih parametara igre. Pravila 2-7 određuju jačinu početnih karata, u odnosu na trenutno stanje igre i činjenicu vezanu za istu boju karata. Izvršavanje ovih pravila izaziva izvršavanje pravila 17-22, koja služe za određivanje strategije, koja zatim aktiviraju pravila 23-31, koja na osnovu strategije biraju akciju, koju bi bot trebalo da izvrši, što predstavlja *forward-chaining*.

1. Determine suits pre-flop

*When*

Postoji PlayerState, čije karte su iste boje

*Then*

Postavlja se *sameSuit* na true

1. Starting hand rank - number of players is 5+, cards have same suit

*When*

Postoji PlayerState, gde je očekivani broj igrača u igri veći od 5 i karte su iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 5+ igrača i uslova da su karte iste boje

1. Starting hand rank - number of players is 5+, cards do not have same suit pre-flop

*When*

Postoji PlayerState, gde je očekivani broj igrača u igri veći od 5 i karte nisu iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 5+ igrača i uslova da karte nisu iste boje

1. Starting hand rank - number of players is 3 or 4, cards have same suit pre-flop

*When*

Postoji PlayerState, gde je očekivani broj igrača u igri 3 ili 4 i karte su iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 3-4 igrača i uslova da su karte iste boje

1. Starting hand rank - number of players is 3 or 4, cards do not have same suit pre-flop

*When*

Postoji PlayerState, gde je očekivani broj igrača u igri 3 ili 4 i karte nisu iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 3-4 igrača i uslova da karte nisu iste boje

1. Starting hand rank - number of players is 1 or 2, cards have same suit pre-flop

*When*

Postoji PlayerState, gde je očekivani broj igrača 1 ili 2 i karte su iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 1-2 igrača i uslova da su karte iste boje

1. Starting hand rank - number of players is 1 or 2, cards do not have same suit pre-flop

*When*

Postoji PlayerState, gde je očekivani broj igrača 1 ili 2 i karte nisu iste boje

*Then*

Postavlja se odgovarajuća jačina početnih karata, na osnovu *IncomeRates* tabele za 1-2 igrača i uslova da karte nisu iste boje

1. Determine thresholds - tight and 1-2

*When*

Postoji Thresholdsi PlayerState, gde je očekivani broj igrača 1 ili 2 i stil igre je *TIGHT*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*ena osnovu formula, zasnovanih na ekspertskom znanju

1. Determine thresholds - tight and 3-4

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača 3 ili 4 i stil igre je *TIGHT*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Determine thresholds - tight and 5+

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača veći od 5 i stil igre je *TIGHT*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Determine thresholds - moderate and 1-2

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača 1 ili 2 i stil igre je *MODERATE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e

1. Determine thresholds - moderate and 3-4

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača 3 ili 4 i stil igre je *MODERATE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Determine thresholds - moderate and 5+

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača veći od 5 i stil igre je *MODERATE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Determine thresholds - loose and 1-2

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača 1 ili 2 i stil igre je *LOOSE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e

1. Determine thresholds - loose and 3-4

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača 3 ili 4 i stil igre je *LOOSE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Determine thresholds - loose and 5+

*When*

Postoji Thresholds i PlayerState, gde je očekivani broj igrača veći od 5 i stil igre je *LOOSE*

*Then*

Računaju se i postavljaju vrednosti za *threshold-*e, vodeći računa i o činjenici da li je trenutna pozicija bot-a *small blind*

1. Strategy MAKE4 pre-flop

*When*

Postoji Thresholds i PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i jačina početnih karata je veća ili jednaka od *threshold*-a za strategiju MAKE4

*Then*

Postavlja se strategija MAKE4

1. Strategy MAKE2 pre-flop

*When*

Postoji Thresholds i PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i jačina početnih karata je veća ili jednaka od *threshold*-a za strategiju MAKE2

*Then*

Postavlja se strategija MAKE2

1. Strategy CALL2 pre-flop

*When*

Postoji Thresholds i PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i jačina početnih karata je veća ili jednaka od *threshold*-a za strategiju CALL2

*Then*

Postavlja se strategija CALL2

1. Strategy MAKE1 pre-flop

*When*

Postoji Thresholds i PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i jačina početnih karata je veća ili jednaka od *threshold*-a za strategiju MAKE1

*Then*

Postavlja se strategija MAKE1

1. Strategy CALL1 pre-flop

*When*

Postoji Thresholds i PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i jačina početnih karata je veća ili jednaka od *threshold*-a za strategiju CALL1

*Then*

Postavlja se strategija CALL1

1. Strategy MAKE0 pre-flop

*When*

Postoji PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija

*Then*

Postavlja se strategija MAKE0

1. Determine action - strategy MAKE4 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE4

*Then*

Postavlja se akcija *raise za 3 \* trenutna veličina bet-a*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE2 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE2

*Then*

Ako je broj trenutnih *raise*-ova manji od 2, postavlja se akcija *raise za trenutnu veličinu bet-a*, u suprotnom akcija *call*, koju bi bot trebalo da izvrši

1. Determine action - strategy CALL2 pre-flop

*When*

Postoji PlayerState, čija je strategija CALL2

*Then*

Postavlja se akcija *call za trenutni iznos*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 1 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i iznos koji bi trebalo *call*-ovati je veći ili jednak dvostrukoj trenutnoj veličini *bet-*a i bot nije platio ništa u tekućoj fazi

*Then*

Postavlja se akcija *fold*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 2 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i iznos koji bi trebalo *call*-ovati je manji od dvostruke trenutne veličine *bet-*a

*Then*

Postavlja se akcija *call za trenutni iznos*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 3 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i broj igrača, koji su *bet/raise* ovu ruku je jednak nuli

*Then*

Postavlja se akcija *raise za trenutnu veličinu bet-*a, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy CALL1 - scenario 1 pre-flop

*When*

Postoji PlayerState, čija je strategija CALL1 i iznos koji bi trebalo *call*-ovati je veći ili jednak dvostrukoj trenutnoj veličini *bet-*a i bot nije platio ništa u tekućoj fazi

*Then*

Postavlja se akcija *fold*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy CALL1 - scenario 2 pre-flop

*When*

Postoji PlayerState, čija je strategija CALL1 i iznos koji bi trebalo *call*-ovati je manji od dvostruke trenutne veličine *bet-*a

*Then*

Postavlja se akcija *call za trenutni iznos*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE0 pre-flop

*When*

Postoji PlayerState, čija je strategija MAKE0 ili nije odabrana nijedna akcija

*Then*

Ako je trenutni iznos, koji bi trebalo *call*-ovati veći od 0, postavlja se akcija *fold*, u suprotnom, akcija *check*, kao akcija, koju bi bot trebalo da izvrši

*Postflop rules* grupa pravila - Radna memorija sadrži *PlayerState* (trenutno stanje bota) i *PostFlopParameters* (promenljive korišćene za određivanje akcije, poput *potOdds, potOdds2, showdownOdds,* zatim *semiBluffingFlag* za *semi-bluffing* taktiku), kao globalne promenljive prosleđeni su *threshold* parametri za strategiju *MAKE1* i *MAKE2*, *flag* koji označava tekuću fazu (*phase)* i podaci o igri (*GameInfo*). Pravila imaju dinamički *salience* u odnosu na fazu igre. Pravilo 1 služi za određivanje *HandRank, PositivePotential, NegativePotential* i *EffectiveHandStrength* karata, koje je bot dobio. Pravilo 2 služi za resetovanje *semiBluffingFlag*, ukoliko su ispunjeni uslovi za to. Pravila 3 i 4 služe za određivanje strategije. Njihovo izvršavanje aktivira pravila 5-9, koja služe za određivanje akcije. Ukoliko nije akcija, još uvek nije izabrana, aktiviraju se pravila 10-12 (*semi-bluffing* taktika), 13-15 (*calling with pot odds* taktika) i 16-20 (*calling with showdown odds taktika).* Aktiviranje određenih pravila iz ovih grupa može dovesti do aktiviranje *event processing*-a, koji je objašnjen u nastavku.

1. Calculate potentials and hand rank

*When*

Postoji PlayerState, čije vrednosti jačine karata nisu izračunate

*Then*

Računaju se i postavljaju vrednosti *HandRank*, *PositivePotential, NegativePotential* i *EffectiveHandStrength*

1. Reset semi-bluffing flag

*When*

Postoji PlayerState, faza igre je *Flop* ili je iznos koji bi trebalo *call*-ovati veći od trenutne veličine *bet-*a

*Then*

Postavlja se globalna promenljiva *semiBluffingFlag* na *false*, kao indikator da ne bi trebalo igrati *semi-bluffing* taktiku

1. Strategy MAKE2 post-flop

*When*

Postoji PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i *EffectiveHandStrength* karata je veći ili jednak *threshold*-u za strategiju MAKE2

*Then*

Postavlja se strategija MAKE2

1. Strategy MAKE1 post-flop

*When*

Postoji PlayerState, čija strategija nije nijedna od ekspertski definisanih strategija i *EffectiveHandStrength* karata je veći ili jednak *threshold*-u za strategiju MAKE1

*Then*

Postavlja se strategija MAKE1

1. Determine action - strategy MAKE2 - scenario 1 post-flop

*When*

Postoji PlayerState, čija je strategija MAKE2 i manje od dva *raise-*a je napravljeno u ovoj fazi

*Then*

Postavlja se akcija *raise*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE2 - scenario 2 post-flop

*When*

Postoji PlayerState, čija je strategija MAKE2 i iznos koji bi trebalo *call*-ovati je veći ili jednak od dvostruke trenutne veličine *bet-*a

*Then*

Postavlja se akcija *call*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 1 post-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i iznos koji bi trebalo *call*-ovati je veći ili jednak od dvostruke trenutne veličine *bet-*a i bot nije ništa platio u tekućoj fazi

*Then*

Postavlja se akcija *fold*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 2 post-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i iznos koji bi trebalo *call*-ovati je manji od dvostruke trenutne veličine *bet-*a

*Then*

Postavlja se akcija *call*, kao akcija, koju bi bot trebalo da izvrši

1. Determine action - strategy MAKE1 - scenario 3 post-flop

*When*

Postoji PlayerState, čija je strategija MAKE1 i nijedan *bet* nije napravljen u ovoj fazi

*Then*

Postavlja se akcija*bet*, kao akcija, koju bi bot trebalo da izvrši

1. Pot odds for semi-bluffing calculation

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije odabrana, broj *bet*-ova, koje bi trebalo platiti je 0

*Then*

Računa se i postavlja vrednost za parametar *potOdds2*, na osnovu stanja igre

1. Semi-Bluffing

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana, *semiBluffingFlag* je *true* ili faza igre nije *River* i *PositivePotential* karata je veći ili jednak od *potOdds2*

*Then*

Postavlja se akcija *bet*, ukoliko bot ima dovoljno novca, u suprotnom *checkOrFold*

1. Alternative for semi-bluffing try

*When*

Postoji PlayerState, nijedna akcija nije izabrana, broj *bet*-ova, koje bi trebalo platiti je 0

*Then*

Postavlja se akcija *check*, kao akcija, koju bi bot trebalo da izvrši

1. Pot odds calculation

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana

*Then*

Računa se i postavlja vrednost parametra *potOdds*, na osnovu stanja igre

1. Calling with pot odds - scenario 1

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana, *EffectiveHandStrength* karata je veći ili jednak *potOdds-*u i faza igre je *River*

*Then*

Postavlja se akcija *call*, kao akcija, koju bi bot trebalo da izvrši

1. Calling with pot odds - scenario 2

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana, *PositivePotential* karata je veći ili jednak *potOdds-*u i faza igre nije *River*

*Then*

Postavlja se akcija *call*, kao akcija, koju bi bot trebalo da izvrši

1. Fold river with showdown odds

*When*

Postoji PlayerState, nijedna akcija nije izabrana, faza igre je *River*

*Then*

Postavlja se akcija *fold*, kao akcija, koju bi bot trebalo da izvrši

1. Calculate showdown cost

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana

*Then*

Računa se i postavlja vrednost parametra *showdownCost*, na osnovu stanja igre

1. Calculate showdown odds

*When*

Postoji PostFlopParameters i PlayerState, nijedna akcija nije izabrana

*Then*

Računa se i postavlja vrednost parametra *showdownOdds*, na osnovu prethodno izračunate *showdownCost* i stanja igre

1. Calling with showdown odds

*When*

Postoji PlayerState, nijedna akcija nije izabrana i *EffectiveHandStrength* je veći ili jednak *showdownOdds-u*

*Then*

Postavlja se akcija *call*, kao akcija, koju bi bot trebalo da izvrši

1. Fold

*When*

Postoji PlayerState, nijedna akcija nije izabrana

*Then*

Postavlja se akcija *checkOrFold*, kao akcija, koju bi bot trebalo da izvrši

*Backwards chaining and querying* – Kako bih implementirao i ove koncepte, uveo sam dve specijalne taktike *Go All In* (*pre-flop*) i *Check flop nuts* (*post-flop*).

*Go All In* – cilj je ići *all-in* u *pre-flop* fazi, kako bi došli do cilja, potrebno je proveriti zadovoljenost tri okolnosti *preFlop* (trenutna faza je *pre-flop*), *lowBankroll* (stanje čipova bota je manje ili jednako od 0.5$) i *hasStrongHand* (jačina ruke koju imamo je iznad određene vrednosti što je čini jakom)

*Check flop nuts* – cilj je *check-ovati nuts* (pod ovim pojmom se označava najjača ruka u tom trenutku), kako bi došli do cilja, potrebno je proveriti zadovoljenost tri okolnosti *flop* (trenutna faza je *flop*), *playersInPot* (određeni mali broj igrača je ostao u igri) i *hasNuts* (*effectiveHandStrength* ruke je iznad određenog kriterijuma, što je čini kandidatom za *nuts* u tom trenutku)

*Event processing* - Pored svih strategija i akcija, potrebno je uvesti i nepredvidivost u igri bota, za čega sam iskoristio *event processing*. Odlučio sam se za tri grupe karata, na kojima ću to primeniti, karte sa vrednošću *EffectiveHandStrength* u intervalima [0.3, 0.4), [0.4, 0.5) i [0.5, 0.6). Karte iz prvog intervala, bot će *bet*-ovati 30% puta, iz drugog 50% i iz trećeg 70%.

Odluku o narednom *bet*-ovanju, bot donosi na osnovu događaja, vezanih za datu vrednost *EffectiveHandStrength*, koji su se desili u prethodnim rundama.

*EffectiveHandStrength in [0.5, 0.6)*

rule "Process 50-60 betting event scenario 1 post-flop"  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FIFTY\_TO\_SIXTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FIFTY\_TO\_SIXTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents >= 0.7, $ehs : results.getEHS(), strategy == Strategy.*MAKE1*, actionToTake != null, actionToTake.isBet(), results.getEHS() >= 0.5 && results.getEHS() < 0.6)  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 modify($ps) {  
 setActionToTake(Action.callAction(gameInfo))  
 }  
 insert(new BettingEvent(HandStrengthEnum.*FIFTY\_TO\_SIXTY*, false));  
end  
  
rule "Process 50-60 betting event scenario 2 post-flop"  
no-loop  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FIFTY\_TO\_SIXTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FIFTY\_TO\_SIXTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents < 0.7, $ehs: results.getEHS(), strategy == Strategy.*MAKE1*, actionToTake != null, actionToTake.isBet(), (results.getEHS() >= 0.5 && results.getEHS() < 0.6))  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 insert(new BettingEvent(HandStrengthEnum.*FIFTY\_TO\_SIXTY*, true));  
end

*EffectiveHandStrength in [0.4, 0.5)*

rule "Process 40-50 betting event scenario 1 post-flop"  
no-loop  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FORTY\_TO\_FIFTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FORTY\_TO\_FIFTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents >= 0.5, $ehs : results.getEHS(), actionToTake != null, actionToTake.isBet(), semiBluffingFlag == true, results.getEHS() >= 0.4 && results.getEHS() < 0.5)  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 modify($ps) {  
 setActionToTake(Action.callAction(gameInfo))  
 }  
 insert(new BettingEvent(HandStrengthEnum.*FORTY\_TO\_FIFTY*, false));  
end  
  
rule "Process 40-50 betting event scenario 2 post-flop"  
no-loop  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FORTY\_TO\_FIFTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*FORTY\_TO\_FIFTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents < 0.5, $ehs: results.getEHS(), actionToTake != null, actionToTake.isBet(), semiBluffingFlag == true, results.getEHS() >= 0.4 && results.getEHS() < 0.5)  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 insert(new BettingEvent(HandStrengthEnum.*FORTY\_TO\_FIFTY*, true));  
end

*EffectiveHandStrength in [0.3, 0.4)*

rule "Process 30-40 betting event scenario 1 post-flop"  
no-loop  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*THIRTY\_TO\_FORTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*THIRTY\_TO\_FORTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents >= 0.3, $ehs: results.getEHS(), actionToTake != null, actionToTake.isBet(), semiBluffingFlag == true, results.getEHS() >= 0.3 && results.getEHS() < 0.4)  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 modify($ps) {  
 setActionToTake(Action.callAction(gameInfo))  
 }  
 insert(new BettingEvent(HandStrengthEnum.*THIRTY\_TO\_FORTY*, false));  
end  
  
rule "Process 30-40 betting event scenario 2 post-flop"  
no-loop  
 when  
 $totalEvents: Number() from accumulate(  
 $be: BettingEvent(  
 handStrengthRange == HandStrengthEnum.*THIRTY\_TO\_FORTY* ),  
 count($be)  
 )  
 $numOfEvents: Number() from accumulate(  
 BettingEvent(  
 handStrengthRange == HandStrengthEnum.*THIRTY\_TO\_FORTY*,  
 betDone == true  
 ),  
 init(int numOfEvents = 0;),  
 action(numOfEvents += 1;),  
 result(numOfEvents)  
 )  
 $ps: PlayerState($numOfEvents / $totalEvents < 0.3, $ehs: results.getEHS(), actionToTake != null, actionToTake.isBet(), semiBluffingFlag == true, results.getEHS() >= 0.3 && results.getEHS() < 0.4)  
 then  
 System.*out*.println("Rule: " + drools.getRule().getName());  
 System.*out*.println("EHS: " + $ehs);  
 insert(new BettingEvent(HandStrengthEnum.*THIRTY\_TO\_FORTY*, true));  
end