**Working memory**

enum TStrikes {Punch, LowKick, HighKick, Unknown};

struct TWorkingMemory {

TStrikes strikeA; // previous, previous strike (data)

TStrikes strikeB; // previous strike (data)

TStrikes strikeC; // next, predicted, strike (assertion)

// note: can add additional elements here for things such as which counter to throw, etc....

};

TWorkingMemory WorkingMemory; // global working memory variable

**Ruleclass**

class TRule {

public:

TRule();

void SetRule(TStrikes A, TStrikes B, TStrikes C);

TStrikes antecedentA;

TStrikes antecedentB;

TStrikes consequentC;

bool matched;

int weight;

};

**SetRule method**

void TRule::SetRule(TStrikes A, TStrikes B, TStrikes C)

{

antecedentA = A;

antecedentB = B;

consequentC = C;

}

**Global variables**

TRule Rules[NUM\_RULES];

int PreviousRuleFired;

TStrikes Prediction;

TStrikes RandomPrediction;

int N;

int NSuccess;

int NRandomSuccess;

**Initialize function**

void TForm1::Initialize(void)

{

Rules[0].SetRule(Punch, Punch, Punch);

Rules[1].SetRule(Punch, Punch, LowKick);

Rules[2].SetRule(Punch, Punch, HighKick);

Rules[3].SetRule(Punch, LowKick, Punch);

Rules[4].SetRule(Punch, LowKick, LowKick);

Rules[5].SetRule(Punch, LowKick, HighKick);

Rules[6].SetRule(Punch, HighKick, Punch);

Rules[7].SetRule(Punch, HighKick, LowKick);

Rules[8].SetRule(Punch, HighKick, HighKick);

Rules[9].SetRule(LowKick, Punch, Punch);

Rules[10].SetRule(LowKick, Punch, LowKick);

Rules[11].SetRule(LowKick, Punch, HighKick);

Rules[12].SetRule(LowKick, LowKick, Punch);

Rules[13].SetRule(LowKick, LowKick, LowKick);

Rules[14].SetRule(LowKick, LowKick, HighKick);

Rules[15].SetRule(LowKick, HighKick, Punch);

Rules[16].SetRule(LowKick, HighKick, LowKick);

Rules[17].SetRule(LowKick, HighKick, HighKick);

Rules[18].SetRule(HighKick, Punch, Punch);

Rules[19].SetRule(HighKick, Punch, LowKick);

Rules[20].SetRule(HighKick, Punch, HighKick);

Rules[21].SetRule(HighKick, LowKick, Punch);

Rules[22].SetRule(HighKick, LowKick, LowKick);

Rules[23].SetRule(HighKick, LowKick, HighKick);

Rules[24].SetRule(HighKick, HighKick, Punch);

Rules[25].SetRule(HighKick, HighKick, LowKick);

Rules[26].SetRule(HighKick, HighKick, HighKick);

WorkingMemory.strikeA = sUnknown;

WorkingMemory.strikeB = sUnknown;

WorkingMemory.strikeC = sUnknown;

PreviousRuleFired = -1;

N = 0;

NSuccess = 0;

NRandomSuccess = 0;

UpdateForm();

}

**ProcessMove function**

TStrikes TForm1::ProcessMove(TStrikes move)

{

int i;

int RuleToFire = -1;

// Part 1:

if(WorkingMemory.strikeA == sUnknown)

{

WorkingMemory.strikeA = move;

return sUnknown;

}

if(WorkingMemory.strikeB == sUnknown)

{

WorkingMemory.strikeB = move;

return sUnknown;

}

// Part 2:

// Process previous prediction first

// Tally and adjust weights

N++;

if(move == Prediction)

{

NSuccess++;

if(PreviousRuleFired != -1)

Rules[PreviousRuleFired].weight++;

} else {

if(PreviousRuleFired != -1)

Rules[PreviousRuleFired].weight--;

// Backward chain to increment the rule that

// should have been fired:

for(i=0; i<NUM\_RULES; i++)

{

if(Rules[i].matched && (Rules[i].consequentC == move))

{

Rules[i].weight++;

break;

}

}

}

if(move == RandomPrediction)

NRandomSuccess++;

// Roll back

WorkingMemory.strikeA = WorkingMemory.strikeB;

WorkingMemory.strikeB = move;

// Part 3:

// Now make new prediction

for(i=0; i<NUM\_RULES; i++)

{

if(Rules[i].antecedentA == WorkingMemory.strikeA &&

Rules[i].antecedentB == WorkingMemory.strikeB)

Rules[i].matched = true;

else

Rules[i].matched = false;

}

// Pick the matched rule with the highest weight...

RuleToFire = -1;

for(i=0; i<NUM\_RULES; i++)

{

if(Rules[i].matched)

{

if(RuleToFire == -1)

RuleToFire = i;

else if(Rules[i].weight > Rules[RuleToFire].weight)

RuleToFire = i;

}

}

// Fire the rule

if(RuleToFire != -1) {

WorkingMemory.strikeC = Rules[RuleToFire].consequentC;

PreviousRuleFired = RuleToFire;

} else {

WorkingMemory.strikeC = sUnknown;

PreviousRuleFired = -1;

}

return WorkingMemory.strikeC;

}