Cart-pole Physics Code

Overview

The cart-pole game supposes an environment where you have a pole connected loosely to a cart with the help of a pin. The pole can rotate freely around the pin. The surface on which cart-pole travels is supposed frictionless. The goal is to keep the pole from rotating past a certain angle.

This code written in our Bemo language will help set up the environment for the cart-pole. Through random trial and error, the cart-pole is supposed to self-learn the variables it needs to keep balanced for the most possible number of frames. The code provides a realistic environment by taking into account the gravity, force of inertia, speed of the cart-pole, and angle of the pole.

Code Visualized

```
fvar gravity = 9.8 ;
fvar masscart = 1.0 ;
fvar masspole = 0.1 ;
fvar total_mass = masspole + masscart ;
fvar length = 0.5 ;
fvar polemass_length = masspole * length ;
fvar force_mag = 10.0 ;
fvar tau = 0.02 ; @ seconds between state updates
fvar pi = 355 / 113 ;
fvar theta_threshold_radians = 12.0 * pi ;
theta_threshold_radians = theta_threshold_radians / 180 ;
fvar x_threshold = 2.4 ; @ x is position
fvar highTheta = theta_threshold_radians * 2 ;
fvar highX =  x_threshold * 2 ;
bvar implicit = true ; @ Euler or emplicit euler
@-----------------------------------
```

```
ivar score = 0 ;
ivar counter1 = 0 ;
ivar seed = 0 ;
while ( 1w ) {
seed = seed + 1 ;
ivar counter2 = 0 ;
ivar reward = 0 ;
bvar done = false ;
ivar randomTemp = rand seed ;
randomTemp = randomTemp % 21000 ;
fvar randomLimit = randomTemp / 100000.0 ;
@----------------------------------
@randomizing some values
seed = seed + 1 ;
ivar state1 = rand seed ;
state1 = state1 % 10000 ;
state1 = state1 - 5000 ;
seed = seed + 1 ;
ivar state2 = rand seed ;
state2 = state2 % 10000 ;
state2 = state2 - 5000 ;
seed = seed + 1 ;
ivar state3 = rand seed ;
state3 = state3 % 10000 ;
state3 = state3 - 5000 ;
seed = seed + 1 ;
ivar state4 = rand seed ;
state4 = state4 % 10000 ;
state4 = state4 - 5000 ;
@----------------------------------
```

```
fvar x = state1 / 100000.0 ;
    fvar x_dot = state2 / 100000.0 ;
    @x_dot is speed(first derivative of x)
    fvar theta = state3 / 100000.0 ;
    fvar theta_dot = state4 / 100000.0 ;
    @----------------------------------
    @ Nested While:
    while( 2w ) {
    fvar costheta = cos theta ;
    fvar sintheta = sin theta ;
    fvar force ;
    ivar action ;
    @-----------------------------------
    bvar actionCond1 = theta > randomLimit ;
    bvar actionCond2 = theta == randomLimit ;
    bvar actionCond = actionCond1 or actionCond2 ;
    if( actionCond ) {
        action = 1 ;
    } else {
    action = 0 ;
    }
    @-----------------------------------
    @In which direction
    bvar forceCond = ( action == 1 ) ;
    if( forceCond ) {
        force = force_mag ;
    }
    else {
        force = 0.0 - force_mag ;
    }
    @-----------------------------------
```

```
@Calculating acceleration
    fvar temp1 = polemass_length * theta_dot ;
    fvar temp2 = theta_dot * sintheta ;
    fvar temp3 = temp1 * temp2 ;
    fvar temp4 = force + temp3 ;
    @ temp:
    fvar temp = temp4 / total_mass ;
    @-----------------------------------
    fvar thetaacc1 = gravity * sintheta ;
    fvar thetaacc2 = costheta * temp ;
    fvar thetaaccNum = thetaacc1 - thetaacc2 ;
    fvar thetaacc3 = masspole * costheta ;
    fvar thetaacc4 = costheta / total_mass ;
    fvar thetaacc5 = thetaacc3 * thetaacc4 ;
    fvar thetaacc6 = 4.0 / 3.0 - thetaacc5 ;
    fvar thetaaccDen = length * thetaacc6 ;
    @ thetaacc:
    fvar thetaacc = thetaaccNum / thetaaccDen;
    @-----------------------------------
    fvar xacc1 = polemass_length * thetaacc ;
    fvar xacc2 = xacc1 * costheta ;
    fvar xacc3 = xacc2 / total_mass ;
    @ xacc:
    fvar xacc  = temp - xacc3 ;
    @-----------------------------------
```

```
@calculating position & angle
    if( implicit ) {
        fvar tauAcc = tau * xacc ;
        x_dot = x_dot + tauAcc ;
        fvar tauXdot = tau * x_dot ;
        x  = x + tauXdot ;
        fvar tauThetaAcc = tau * thetaacc ;
        theta_dot = theta_dot + tauThetaAcc ;
        fvar tauThetaDot = tau * theta_dot ;
        theta = theta + tau * tauThetaDot ;
    }
    else{
        fvar xp = tau * x_dot ;
        fvar x_dotp = tau * xacc ;
        fvar thetap = tau * theta_dot ;
        fvar theta_dotp = tau * thetaacc ;
        x  = x + xp ;
        x_dot = x_dot + x_dotp ;
        theta = theta + thetap ;
        theta_dot = theta_dot + theta_dotp ;
        }
    @------------------------------------
    fvar negX_threshold = 0.0 - x_threshold ;
    bvar d1 = x < negX_threshold ;
    bvar d2 = x > x_threshold ;
    fvar negTheta_threshold = 0.0 - theta_threshold_radians ;
    bvar d3 = theta < negTheta_threshold ;
    bvar d4 = theta > theta_threshold_radians ;
    bvar d12 = d1 or d2 ;
    bvar d34 = d3 or d4 ;
    done = d12 or d34 ;
    @------------------------------------
```

```
counter2 = counter2 + 1 ;
reward = reward + 1 ;
@nested while conditions:
bvar 2w1 = counter2 < 700 ;
bvar 2w2 = not done ;
bvar 2w = 2w1 and 2w2 ;
@------------------------------------
}
score = reward ;
counter1 = counter1 + 1 ;
@first while conditions:
bvar 1w1 = score < 500 ;
bvar 1w2 = counter1 < 1000 ;
bvar 1w = 1w1 and 1w2 ;
@------------------------------------
}
```