```
////////////////////////////////////////////////////////////////////////////
//                          MAIN MC / TD(LAMBDA) UPDATE
////////////////////////////////////////////////////////////////////////////

// Decay the learning rate
double ALPHA = LEARNING_RATE * 100d / ((double)_nGamesFinished++);

// Return currently propagating back in time
double current_return = win ? 1d : 0d;

// TD(LAMBDA) CHANGE: Go through states from end to beginning...
for (int mv = _nMovesInGame - 1; mv >= 0; mv--) {

        // First calculate pre-update value of current state
        double cur_state_value = 0.0d;
        for (int i = 0; i < _aMovesMade[mv]._index ; i++) {
                Feature f = _aMovesMade[mv]._a[i];
                cur_state_value += f._dLocalWin;
        }
        double td_error = current_return - cur_state_value;

        // Now update states for this move
        for (int i = 0; i < _aMovesMade[mv]._index ; i++) {
                Feature f = _aMovesMade[mv]._a[i];
                double gradient_f = f._dLocalWin; // Grad of linear fn is just coef!

                // For non-function approximation, update is just ALPHA * td_error,
                // but for function approximation, also multiply in gradient of
                // function w.r.t. parameter... in this case, just boolean indicator
                // function that is 1 or 0... if it was 0, not included here, and
                // would not be updated anyways... so this is 1 case.
                f._dLocalWin += ALPHA * td_error; // * 1d
        }

        // TD(LAMBDA) CHANGE:
        // Would typically have to add in intermediate reward here, but no intermediate
        // reward in games, also no discounting!
        current_return = LAMBDA * current_return + (1 - LAMBDA) * cur_state_value;

}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```