MODULE *ProfileManager*

EXTENDS *Integers*, *Sequences*

– The TLA+ specification is based on the *Lua* module *sd_profile_manager.lua*.
– It models the player's profile, which adapts the game's difficulty.

– *Lua* State Variables Modeled in TLA+:
– *playerProfile* = {
–– *history* = {},
–– *historyLimit* = 15,
–– *currentClues* = 12,
–– *targetMistakes* = 3,
–– *Kp* = 0.5
–– }

CONSTANTS

HistoryLimit,	The maximum number of game results to store. Corresponds to *historyLimit*.
MinClues,	The minimum number of clues allowed.
MaxClues,	The maximum number of clues allowed.
Kp_num,	Numerator for the proportional gain *Kp*.
Kp_den	Denominator for the proportional gain *Kp*. (*e.g.*, *Kp* = 0.5 is *Kp_num* = 1, *Kp_den* = 2)

ASSUME

$\land\ HistoryLimit \in Nat$
$\land\ MinClues \in Int$
$\land\ MaxClues \in Int$
$\land\ MinClues \leq MaxClues$
$\land\ Kp\_den \neq 0$

VARIABLES

history,	A sequence of mistake counts from recent games.
currentClues,	The current number of clues for a puzzle.
targetMistakes	The ideal number of mistakes per puzzle.

$vars \triangleq \langle history,\ currentClues,\ targetMistakes \rangle$

– The *TypeOK* invariant defines the valid types and ranges for the state variables.
– history: A sequence of natural numbers, with its length not exceeding *HistoryLimit*.
– *currentClues*: An integer between *MinClues* and *MaxClues*.
– *targetMistakes*: A natural number.

$TypeOK \triangleq$
$\land\ history \in Seq(Nat)$
$\land\ Len(history) \leq HistoryLimit$
$\land\ currentClues \in MinClues\ ..\ MaxClues$
$\land\ targetMistakes \in Nat$

– The initial state of the system.

$Init \triangleq$
$\land\ history = \langle\rangle$

$\land\ currentClues = 12$
$\land\ targetMistakes = 3$

– Helper operator to clamp a value within a *min*/*max* range.

$Clamp(val,\ min,\ max) \triangleq$
IF $val < min$ THEN min ELSE IF $val > max$ THEN max ELSE val

– Action: A game is finished, and its mistake count is added to the history.
– If the history is full, the oldest entry is removed.

$UpdateHistory(mistakeCount) \triangleq$
$\land$ LET $newHistory \triangleq Append(history,\ mistakeCount)$
IN $history' =$ IF $Len(newHistory) > HistoryLimit$
THEN $SubSeq(newHistory,\ 2,\ Len(newHistory))$
ELSE $newHistory$
$\land$ UNCHANGED $\langle currentClues,\ targetMistakes\rangle$

– Action: Adjust the number of clues based on the player's performance.
– This models the *PID* controller logic from the *Lua* module.

$UpdateDifficulty(actualMistakes) \triangleq$
$\land$ LET
$error \triangleq targetMistakes - actualMistakes$
$adjustment \triangleq (error * Kp\_num) \div Kp\_den$
$newClues \triangleq currentClues - adjustment$
IN
$currentClues' = Clamp(newClues,\ MinClues,\ MaxClues)$
$\land$ UNCHANGED $\langle history,\ targetMistakes\rangle$

– Action: The target number of mistakes is updated, for example by a
– genetic algorithm or other external process.

$SetTargetMistakes(newTarget) \triangleq$
$\land\ newTarget \in Nat$
$\land\ targetMistakes' = newTarget$
$\land$ UNCHANGED $\langle history,\ currentClues\rangle$

– Action: A player profile is loaded, non-deterministically setting the state
– to any valid configuration. This models the effect of *loadProfile*().

$LoadProfile \triangleq$
$\land \exists\, h \in \{s \in Seq(Nat) : Len(s) \leq HistoryLimit\} : history' = h$
$\land \exists\, c \in MinClues\ ..\ MaxClues : currentClues' = c$
$\land \exists\, t \in Nat : targetMistakes' = t$

– The next-state relation: at any step, one of the actions can occur.
– The parameters to the actions are non-deterministically chosen from valid values.

$Next \triangleq$
$\lor \exists\, m \in Nat : UpdateHistory(m)$
$\lor \exists\, am \in Nat : UpdateDifficulty(am)$

$\lor \exists\, nt \in Nat : SetTargetMistakes(nt)$
$\lor LoadProfile$

– The complete specification for the system.

$Spec \triangleq Init \land \Box[Next]_{vars}$

– Theorem: The specification implies that the type invariant always holds.

THEOREM $Spec \Rightarrow \Box TypeOK$