MODULE *GAOptimizer*

EXTENDS *Integers*, *Sequences*, *Naturals*

– This specification models a genetic algorithm (*GA*) that optimizes the
– "*targetMistakes*" parameter based on a player's game history.
– It is based on the *Lua* module *sd_ga_optimizer.lua*.

CONSTANTS

$POPULATION\_SIZE$,	The number of individuals (chromosomes) in each generation.
$GENERATIONS$,	The number of generations the algorithm will evolve.
$MUTATION\_RATE\_PCT$,	The mutation rate as a percentage (*e.g.*, 10 for 10%).
$PRECISION$,	A factor to handle floating point numbers (*e.g.*, 100).
$TOP\_PERCENT\_PARENTS$	The percentage of the top performers to select as parents (*e.g.*, 25).

ASSUME

$\land\ POPULATION\_SIZE \in Nat$
$\land\ GENERATIONS \in Nat$
$\land\ MUTATION\_RATE\_PCT \in 0\,..\,100$
$\land\ PRECISION \in Nat \setminus \{0\}$
$\land\ TOP\_PERCENT\_PARENTS \in 1\,..\,100$

ChromosomeRange is now a defined operator, not a constant.
It is calculated automatically from *PRECISION*.

$ChromosomeRange \triangleq (1 * PRECISION)\,..\,(5 * PRECISION)$

VARIABLES

$population$,	A sequence of chromosomes, representing potential "*targetMistakes*" values.
$next\_generation$,	A temporary sequence to build the next generation.
$generation$,	The current generation number.
$history$,	The player's game history (sequence of mistake counts).
pc	Program counter to control the flow of the algorithm.

$vars \triangleq \langle population, next\_generation, generation, history, pc \rangle$

– The type invariant for the state variables.

$TypeOK \triangleq$
$\land\ population \in Seq(Int)$
$\land\ next\_generation \in Seq(Int)$
$\land\ generation \in 0\,..\,GENERATIONS$
$\land\ history \in Seq(Nat)$
$\land\ pc \in \{\text{"init"}, \text{"evolve"}, \text{"select\_parents"}, \text{"add\_child"}, \text{"yield"}, \text{"done"}\}$

– The initial state of the genetic algorithm. The history is provided as input.

$Init(hist) \triangleq$
$\land\ history = hist$
$\land\ pc = \text{"init"}$
$\land\ generation = 0$

$\land$ $population = \langle\rangle$
$\land$ $next\_generation = \langle\rangle$

Absolute value helper operator

$abs(n) \triangleq \text{IF } n < 0 \text{ THEN } -n \text{ ELSE } n$

– Helper operator to calculate fitness.

$$
\begin{aligned}
&CalculateFitness(target, playerHistory) \triangleq \\
&\quad \text{IF } playerHistory = \langle\rangle \\
&\quad \text{THEN } 0 \\
&\quad \text{ELSE LET } Sum[i \in 1 \,..\, (Len(playerHistory) + 1)] \triangleq \\
&\qquad\qquad \text{IF } i > Len(playerHistory) \\
&\qquad\qquad \text{THEN } 0 \\
&\qquad\qquad \text{ELSE } abs(target - playerHistory[i] * PRECISION) + Sum[i + 1] \\
&\qquad\quad \text{IN } Sum[1]
\end{aligned}
$$

– Action: Create the initial population with random values.

$$
\begin{aligned}
&CreateInitialPopulation \triangleq \\
&\quad \land\ pc = \text{“init”} \\
&\quad \land\ \exists\, pop \in [1 \,..\, POPULATION\_SIZE \to ChromosomeRange] : \\
&\qquad population' = pop \\
&\quad \land\ generation' = 1 \\
&\quad \land\ pc' = \text{“evolve”} \\
&\quad \land\ \text{UNCHANGED } \langle next\_generation, history\rangle
\end{aligned}
$$

– Action: The main loop condition. If generations are left, evolve. Otherwise, finish.

$$
\begin{aligned}
&EvolveLoopCondition \triangleq \\
&\quad \land\ pc = \text{“evolve”} \\
&\quad \land\ \text{IF } generation \leq GENERATIONS \\
&\qquad \text{THEN } pc' = \text{“select\_parents”} \\
&\qquad \text{ELSE } pc' = \text{“done”} \\
&\quad \land\ \text{UNCHANGED } \langle population, next\_generation, generation, history\rangle
\end{aligned}
$$

– Action: Non-deterministically choose a subset of the population to be parents.

$$
\begin{aligned}
&SelectParents \triangleq \\
&\quad \land\ pc = \text{“select\_parents”} \\
&\quad \land\ \text{LET } num\_parents \triangleq (POPULATION\_SIZE * TOP\_PERCENT\_PARENTS) \div 100 \\
&\qquad \text{IN } \exists\, parents \in \{s \in \{\text{SUBSET } population\} : Len(s) = num\_parents\} : \\
&\qquad\qquad next\_generation' = parents \\
&\quad \land\ pc' = \text{“add\_child”} \\
&\quad \land\ \text{UNCHANGED } \langle population, generation, history\rangle
\end{aligned}
$$

– Action: Add a new child to the next generation until it is full.

$$
\begin{aligned}
&AddChild \triangleq \\
&\quad \land\ pc = \text{“add\_child”}
\end{aligned}
$$

$$
\begin{array}{l}
\land\ \textsc{if}\ Len(next\_generation) < POPULATION\_SIZE \\
\quad \textsc{then}\ \land \exists\, p1\_idx,\ p2\_idx \in 1 \,..\, Len(next\_generation),\ r \in 0 \,..\, 99 : \\
\qquad\qquad \textsc{let}\ parents \triangleq next\_generation \\
\qquad\qquad\qquad p1 \triangleq parents[p1\_idx] \\
\qquad\qquad\qquad p2 \triangleq parents[p2\_idx] \\
\qquad\qquad\qquad child \triangleq (p1 + p2) \div 2 \\
\qquad\qquad\qquad mutated\_child \triangleq \textsc{if}\ r < MUTATION\_RATE\_PCT \\
\qquad\qquad\qquad\qquad\qquad \textsc{then}\ child + (((2 * r) - PRECISION) \div 2) \\
\qquad\qquad\qquad\qquad\qquad \textsc{else}\ child \\
\qquad\qquad \textsc{in}\ \ next\_generation' = Append(next\_generation,\ mutated\_child) \\
\qquad\quad \land\ pc' = \text{“add\_child”} \\
\quad \textsc{else}\ \land\ pc' = \text{“yield”} \\
\qquad\quad \land\ \textsc{unchanged}\ \langle next\_generation \rangle \\
\land\ \textsc{unchanged}\ \langle population,\ generation,\ history \rangle
\end{array}
$$

– Action: Yield control, commit the new generation, and loop back.

$$
\begin{array}{l}
YieldAndContinue \triangleq \\
\quad \land\ pc = \text{“yield”} \\
\quad \land\ population' = next\_generation \\
\quad \land\ generation' = generation + 1 \\
\quad \land\ pc' = \text{“evolve”} \\
\quad \land\ \textsc{unchanged}\ \langle next\_generation,\ history \rangle
\end{array}
$$

– The next-state relation for the algorithm’s execution.

$$
\begin{array}{l}
Next(hist) \triangleq \\
\quad \lor\ CreateInitialPopulation \\
\quad \lor\ EvolveLoopCondition \\
\quad \lor\ SelectParents \\
\quad \lor\ AddChild \\
\quad \lor\ YieldAndContinue
\end{array}
$$

– The specification defines a single run of the algorithm with a given history.

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

– Theorem: The algorithm eventually terminates.

$$\textsc{theorem}\ \forall\, hist \in Seq(Nat) : Spec(hist) \Rightarrow \Diamond(pc = \text{“done”})$$