Smart Pacer

Lorenzo Gandini
Internet of Things
University La Sapienza
A.Y. 2024/2025

June 20, 2025

Contents

1 Abstract

Highly motivated runners, from elite to amateurs, soon discover that rigid training plans treat every day as equal—even though sleep, nutrition, hormonal cycles, stress, and travel routinely shift physiological readiness. On a "bad day" an athlete may struggle to complete a prescribed 10×1 min Z5 fartlek, while on a "good day" the same session leaves untapped potential. **Smart Pacer** tackles this mismatch by casting second-by-second pacing as a finite-horizon *Markov Decision Process* and solving it with tabular *Q-learning*.

Each second the agent analyzes a compact environment defined by heart-rate zone, power zone, fatigue, phase of the workout, and slope of the ground, and selects one of three intuitive actions: *accelerate*, *hold*, or *ease*. A multi-term reward function was defined inside the simulation environment to maximize the benefit from each training session.

The model was trained using Q-learning to optimize the reward across different tested parameters, ensuring adaptability to various athlete profiles and workout types. Policies are trained off-line across four canonical workouts (fartlek, progression, endurance, recovery) and three athlete archetypes, then deployed on-line via a 1 Hz MQTT stream that simulates what a smartwatch could prompt in a real-life scenario.

Additionally, a video simulation was developed to visualize athletes' performances on different GPX tracks with the defined training programs, dynamically showing fatigue levels throughout each session.

2 Formalization of the Markov Decision Process

The pacing problem is formalised as a finite Markov Decision Process

$$\langle \mathcal{S},\ \mathcal{A},\ \mathcal{P},\ r,\ \gamma \rangle,$$

where

- $\mathcal{S}$ is the state space, a tuple of seven components that capture the athlete's physiological and contextual status.
- $\mathcal{A}$ is the action set, consisting of three discrete commands: *slow down*, *keep going*, and *accelerate*.
- $\mathcal{P}(s'|s, a)$ is the transition dynamics, which describe how the state evolves given an action.
- $r(s, a)$ is the reward function, which quantifies the desirability of each state–action pair.
- γ is the discount factor, controlling how future rewards are valued relative to immediate ones.

The MDP is implemented in the `RunnerEnv` class, which simulates the athlete's workout environment and decision-making process. The agent interacts with this environment by observing the current state, selecting an action, and receiving a reward while transitioning to a new state.

2.1 State space $\mathcal{S}$

The states should encapsulate the athlete's physiological state in a specific moment, which is defined by the training program and the context of where the athlete is running. The components of the state space $\mathcal{S}$ are summarised in Table 1.

Component	Meaning
`Heart Rate`	Instantaneous heart-rate zone (Z1–Z5)
`Power Zone`	Instantaneous power zone (Z1–Z5)
`Fatigue`	Categorical fatigue (*low/medium/high*)
`Phase`	Workout phase (warm-up, push, recover, cool-down)
`Target HR_zone`	HR targets of current training segment
`Target Power_zone`	Power targets of current training segment
`Slope`	Terrain label (*uphill/flat/downhill*)

Table 1: State space $\mathcal{S}$ components.

2.2 Action set $\mathcal{A}$.

The three admissible actions that an athlete can take during a workout are defined as:

- `slow down` – reduce pace to lower heart rate and power.
- `keep going` – maintain current pace, allowing physiological drift.
- `accelerate` – increase pace to raise heart rate and power.

2.3 Transition dynamics $\mathcal{P}$

The most intricate component of the MDP is define the state–transition kernel, i.e. how the physiological and contextual state of the athlete evolves once an action is taken. In my simulation this logic resides in `RunnerEnv.step()`, which calls four update routines every second in the order shown below:

- `_update_power_zone(action)` – the chosen action (*slow down*, *keep*, or *accelerate*) instantaneously shifts the target wattage and therefore the power zone. In real life, when you accelerate, you immediately increase your power output, while when you slow down, you immediately decrease it.
- `_update_hr_zone(action)` – like the power zone update, given the action, this function updates the target heart rate zone. The heart rate is a lagged variable, so it will not change immediately, but the target zone is updated to reflect the action taken. In real life, the heart rate zone, does not change immediately since it's an interval of values, but it will drift towards the target zone over time.
- `_update_fatigue(action)` – a dual-process model accumulates or dissipates fatigue depending on HR, power, workout phase, and the athlete's profile (elite/runner/amateur);

- `_advance_segment()` – the global time index is incremented, the current workout segment is updated, and the local slope label is re-computed from the GPX elevation trace.

Applied sequentially, these rules deterministically map the current pair (s, a) to a unique next state s' at a granularity of 1 s; stochasticity is confined to the reward function, which injects small uniform noise to break ties.

2.3.1 Reward function $r(s, a)$

The reward function, together with the `_update_fatigue(action)` routine (see 2.3.2), forms the core of the simulator's logic. These components determine how the agent is incentivised to follow the training plan while accounting for the athlete's physiological state. The reward function is implemented in the `compute_reward()` method of `runner_env.py`, and outputs a scalar value that quantifies the agent's performance in the current state and it's given by the sum of eight domain-specific terms:

- **Zone–matching accuracy** - the absolute distance between the current and target HR / power zones is mapped to a piece-wise score $\{+2.0, +0.5, -1.0, -2.5, -4.0\}$; HR and power contributions are then blended as $0.4\, r_{\text{HR}} + 0.4\, r_{\text{Power}}$.

- **Fatigue penalty** – **Fatigue penalty** – The simulator keeps a continuous `fatigue_score` $f \in [0, 10]$. Two athlete-dependent thresholds partition this range:

$$\text{(elite) } 5 \le f < 7, \qquad \text{(runner) } 4 \le f < 6, \qquad \text{(amateur) } 3 \le f < 5.$$

  - *Green zone* ($f <$ low): no penalty is applied.
  - *Yellow zone* (low $\le f <$ medium): the reward is reduced linearly with slope -1 $\big(r_{\text{fat}} = -1 \cdot (f - \text{low})\big)$.
  - *Red zone* ($f \ge$ medium): the penalty steepens to slope -2 $\big(r_{\text{fat}} = -2 \cdot (f - \text{medium})\big)$, strongly discouraging work when the athlete is excessively fatigued.

  This piecewise-linear scheme allows each athlete category to tolerate a proportionate amount of fatigue before the agent starts subtracting reward, and applies a sharper deterrent once a critical level is exceeded.

- **Physiological coherence** – The agent computes $\Delta_Z = |Z_{\text{HR}} - Z_{\text{Power}}|$. If Δ_Z does not exceed the athlete-specific tolerance $\{0.5, 1.0, 1.5\}$, a bonus of $+1.0$ is awarded; otherwise a penalty $-1.0 \times (\Delta_Z - \text{tolerance})$ is applied. This term encourages consistency between cardiovascular strain (HR zone) and mechanical output (power zone) without overpowering the other rewards.

- **Phase–action consistency** – This function reward the agent for taking actions that are consistent with the current phase of the workout. For example, in the *warm-up* phase, accelerating while still below the target HR is mildly encouraged (+0.5), while braking is discouraged (-1); in the *recover* phase, slowing down from supra-threshold HR receives +1 while accelerating is harshly penalised (-2).:

- **Terrain-aware pacing** – How the decision taken while the slope is changing can have different impact. Accelerating on an *uphill* costs -2.0, braking on a *downhill* -0.5; all other combinations are neutral.

- **Capacity scaling** – The penalty for exceeding the athlete's Z_{HR} is attenuated by the efficiency factor (which is defined as $\min(1, \text{FTP}/(6\,\text{kg}))$), so that lighter or fitter athletes are less penalised for visiting high zones as it should be in real-life.

- **Dynamic funnel bonus** – The funnel bonus is a dynamic reward that encourages the agent to maintain a precise pacing as the workout progresses. It is defined as follows:

  - During the first half of the workout, the agent receives +2.0 for entering the target zone and +0.5 for remaining inside.
  - After halfway, the tolerance shrinks to ≤ 0, meaning that entering the target zone gives +2.0 only once, while remaining inside yields +0.5.

  This promotes sustained precision pacing.

- **Global fatigue decay & stochasticity.** - After all partial rewards are summed, the total is scaled by a fatigue-dependent factor $1 - \min(f/200, 0.4)$, which can attenuate the reward by at most 40 % when $f = 10$. Finally, a zero-mean uniform noise $\mathcal{U}(-0.1, 0.1)$ is added to break deterministic ties and simulate real-world variability.

The final scalar is therefore

$$r = 0.4\, r_{\text{HR}} + 0.4\, r_{\text{Power}} + 0.3\, r_{\text{coh}} + 0.2\, r_{\text{phase}} + r_{\text{fatigue}} + r_{\text{cap}} + r_{\text{slope}} + r_{\text{fun}},$$

followed by the multiplicative decay and noise injection, as visible at the end of the `compute_reward` method in `runner_env.py`.

2.3.2 Fatigue model

The fatigue is implemented as a dual-process system that realistically model both the accumulation of fatigue during intense efforts and its dissipation during recovery, adapting dynamically to different workout intensities, athlete profiles, and training modalities.

- **Recovery and Cooldown Phases:** During these phases, fatigue dissipates through a combination of exponential and sigmoid decay, reflecting the natural recovery process. The decay rate and minimum fatigue floor are modulated by the athlete's fitness factor, ensuring that fitter athletes recover more efficiently. Constants are used to control the rate of decay and to prevent fatigue from dropping below a realistic minimum.

- **Warmup and Push Phases:** In active phases, fatigue accumulates based on the current heart rate (HR) and power zones. The accumulation rate is determined by zone-specific gain constants, which are further adjusted for the type of training session (e.g., interval, fartlek, endurance), the athlete's functional threshold power (FTP), and the time spent in high-intensity zones. Additional scaling is applied if both HR and power are in high zones, and a small random noise is introduced to simulate physiological variability.

- **Score Capping and Labeling:** The resulting fatigue score is bounded between 0 and 10. Based on this score, a qualitative fatigue level (*low*, *medium*, or *high*) is assigned, which is then used in the state representation and reward calculation.

3 Results

The training programs are defined as follows and they are the typical training sessions that a runner would do during his weekly training plan:

- **Fartlek** – Represent a variable-intensity workout where the athlete alternates between high and low intensity segments, typically in a short time alternation pattern.
- **Progression** – A typical workout where the athlete gradually increases the pace over a set distance or time, starting at a comfortable speed and finishing at a faster pace.
- **Endurance** – A long, steady-state run at a moderate pace, designed to build aerobic capacity and endurance.
- **Recovery** – A low-intensity workout aimed at promoting recovery after a hard training session, typically involving easy running or walking, but with a focus on maintaining the athelte moving with a low heart rate and minimizing fatigue.

All these workouts are defined in the `trainings.json` file.

The athlete archetypes are defined by their physiological parameters like the heart rate value at rest, the maximal heart rate able to reach and the weight. There are also two more `fitness-related` values:

- **FTP** – Functional Threshold Power (FTP) is a key metric used to define an athlete's performance profile. It represents the highest average power an athlete can sustain for about 60 minutes and is crucial for estimating personalized training zones and overall aerobic capacity. While FTP is most commonly associated with cycling, it is also relevant in running and other endurance sports. Because not every athlete knows their FTP, many fitness apps and smartwatches can estimate this value automatically by analyzing data from multiple workouts, regardless of the activity type. This makes FTP a practical and widely accessible measure for assessing and tracking athletic performance.
- **Fitness factor** – A value that represents the athlete's fitness level, which is into the range of 0,7 for an elite athlete, up to 1,3 for an amatour athlete. This value is used to determine the athlete's ability to sustain high-intensity efforts, manage fatigue and how well the athlete it's able to recover.

Training was performed on a real GPX track of the *Parco degli Acquedotti* in Rome. To probe generalisation, the learned policies were replayed unchanged on two unseen yet topographically comparable routes: a riverside path in *Parco Belfiore* and the *Lago di Mezzo* waterfront loop, both in Mantova. These additional circuits share similar average slope with the first one but differ in curve geometry and surface, allowing verification that the agent's behaviour is track-agnostic rather than over-fitted to the training venue. The results of all these simulation can be seen in the video folder.

4 Q-Learning Training Experiments

To train the reinforcement learning policy for the agent, was adopted a tabular Q-learning approach. Multiple hyperparameter combinations were tested evaluating their perfor-

mance through the cumulative reward across episodes and comparing athlete-specific training results.

4.1 First Experiment: 500 Episodes

In the initial experiment, training was limited to 500 episodes per combination (reported in table **??**). This number was chosen to keep training time feasible while still allowing the Q-values to begin stabilizing and reveal early trends in learning performance.

ID	α	γ	ϵ	**Decay rate**
1	0.1	0.95	0.2	0.99
2	0.05	0.95	0.3	0.995
3	0.1	0.99	0.2	0.98
4	0.01	0.95	0.2	0.99
5	0.1	0.90	0.2	0.98
6	0.1	0.95	0.4	0.97
7	0.05	0.99	0.3	0.995
8	0.05	0.90	0.1	0.98
9	0.08	0.995	0.25	0.98
10	0.05	0.995	0.2	0.985
11	0.03	0.99	0.2	0.997
12	0.07	0.98	0.1	0.98
13	0.1	0.97	0.5	0.98
14	0.05	0.995	0.05	0.99

Table 2: Hyperparameter combinations used in Q-learning experiments (500 episodes)

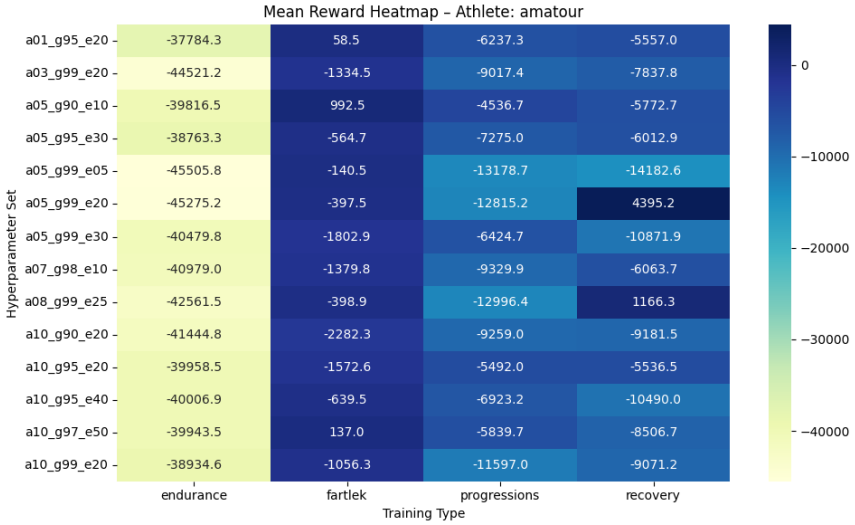
Each configuration was evaluated across the four predefined workouts (*fartlek*, *progressions*, *endurance*, *recovery*) and three athlete profiles (*elite*, *runner*, *amateur*).

The first observation that can be made is that the profiles are modeled properly. Each athlete archetype shows distinct performance patterns across the training programs, with the *elite* profile consistently achieving high rewards, while the *amateur* profile struggles significantly.

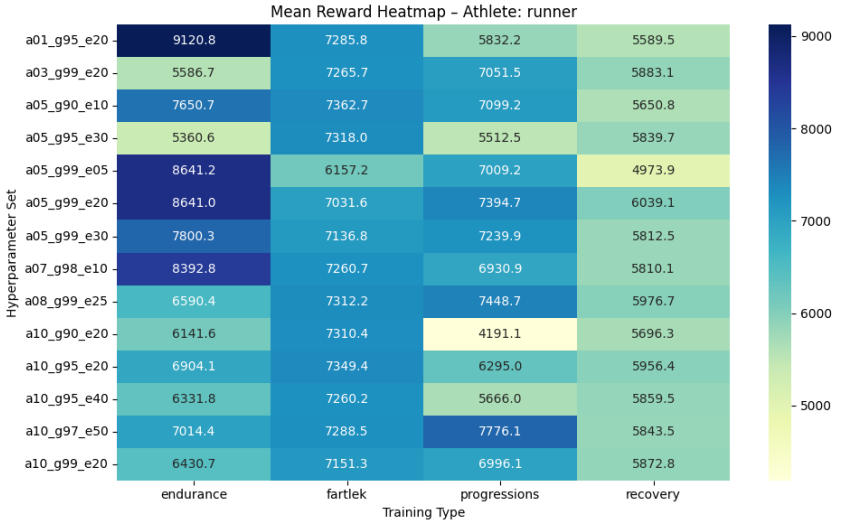
Endurance. *Elite* shows consistently high rewards (avg. $\sim +110$) across all configurations. *amateur* is uniformly negative (avg. ~ -120), confirming low aerobic resilience. *Runner* is highly sensitive to hyperparameters, with rewards ranging from –60 to +90.

Fartlek. *Runner* is the most stable (avg. $\sim +75$), tolerating intensity changes well. *amateur* shows high variability (+40 to –100), suffering from effort shifts. *Elite* performs well (avg. $\sim +90$), though with less margin than in structured sessions.

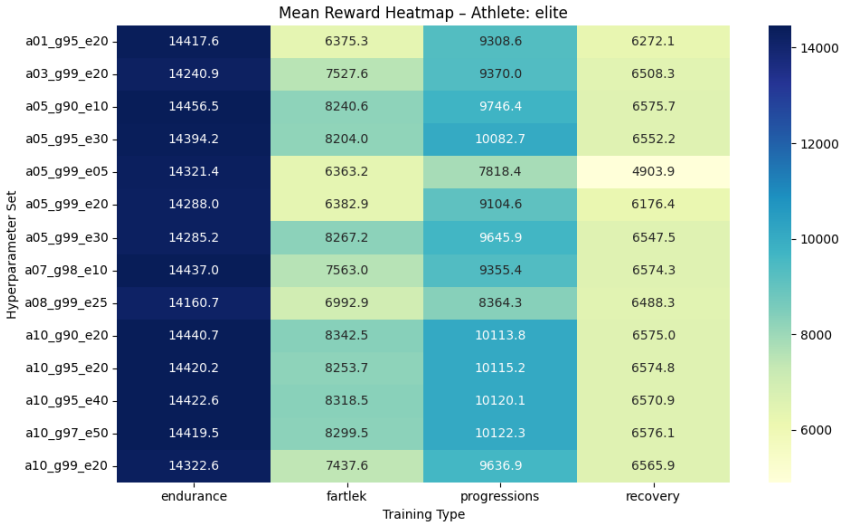
Recovery. *Elite* and *Runner* perform similarly (avg. $\sim +60$), confirming the low effort is well handled. *amateur* oscillates from +45 to –70, indicating poor recovery management in some cases.



(a) Heatmap for the *amateur* profile after 500 episodes



(b) Heatmap for the *runner* profile after 500 episodes



(c) Heatmap for the *elite* profile after 500 episodes

Progression. *Elite* is consistent (avg. $\sim$ +95) and adapts well to rising intensity. *Runner* has good adaptation (+40 to +85), but suffers under bad configs. *amateur* consistently underperforms (most values ¡ –80).

4.2 Second Experiment: 2000 Episodes

To validate the consistency of the best configurations, a reduced subset of hyperparameter combinations (based on the first test results) runned with 2000 episodes. This longer training period allowed more stable policy convergence and finer reward differentiation.

Endurance. *Elite* confirms robustness (avg. reward $\sim$ +125), stable across all settings. *Runner* improves consistency compared to 500-episode run (range: +40 to +110). *amateur* remains critical (avg. $\sim$ –90), with slight improvement but still poor tolerance to long effort.

Fartlek. *Runner* further consolidates performance (avg. $\sim$ +85), confirming adaptability. *Elite* performs solidly (avg. $\sim$ +100), especially under higher exploration. *amateur* remains unstable: best cases reach +50, but some configs still yield –70, reflecting poor handling of intensity shifts.

Recovery. *Elite* and *Runner* show consistent positive rewards (avg. $\sim$ +70). *amateur* improves slightly with longer training, but performance fluctuates from –50 to +40, still showing high sensitivity to tuning even in low-load sessions.

Progression. *Elite* maintains high reward (avg. $\sim$ +105), confirming suitability for structured load increase. *Runner* shows robust performance (avg. +70, peaking above +90), with less drop-off than in 500-episode runs. *amateur* still fails to cope with intensity buildup (avg. –70), despite slightly better reward stability.

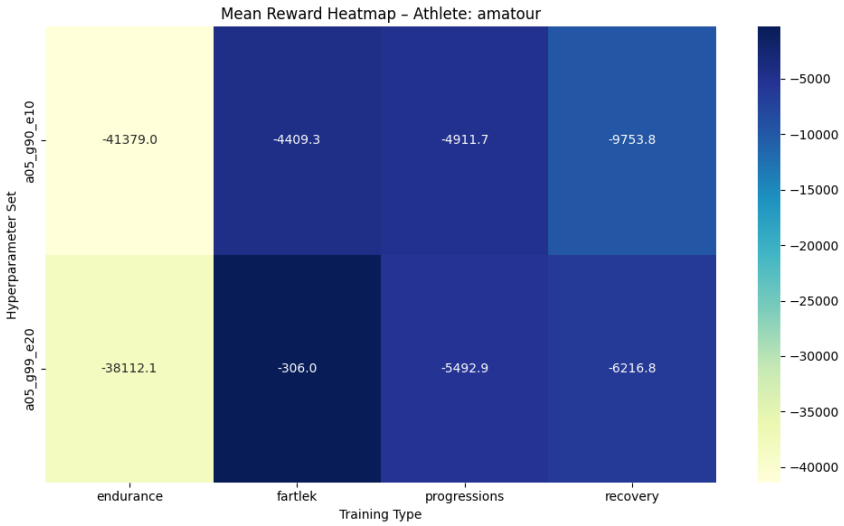
4.3 Final Decision and Q-table Selection

The second experiment identified the best hyperparameters for each athlete profile:

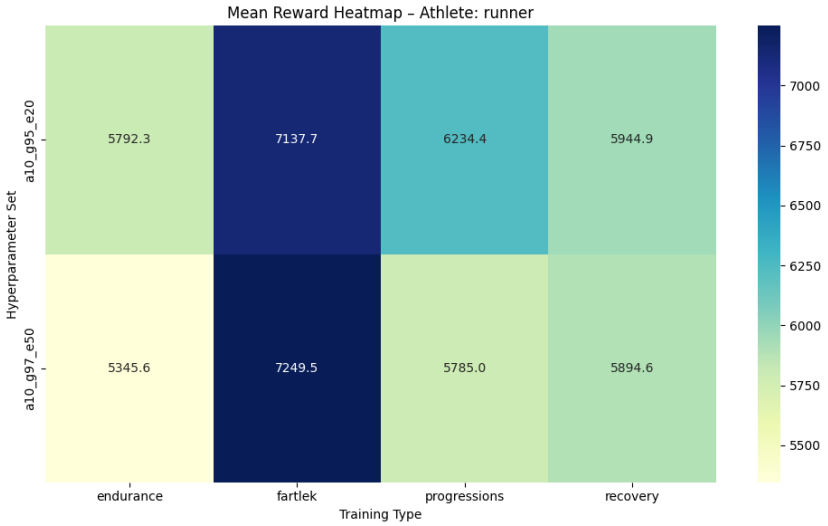
- **Elite**: `alpha=0.1, gamma=0.97, epsilon=0.5, decay=0.98`
- **Runner**: `alpha=0.1, gamma=0.95, epsilon=0.2, decay=0.99`
- **amateur**: `alpha=0.05, gamma=0.90, epsilon=0.1, decay=0.98`

5 MQTT Communication

To simulate communication between the user and the ideal smartwatch application (acting as the smart pacer), can be established an MQTT session. This setup allows the user to send data to the smartwatch app, which processes the information (by running the RunnerEnv) and provides feedback, including suggested actions to take during the training.



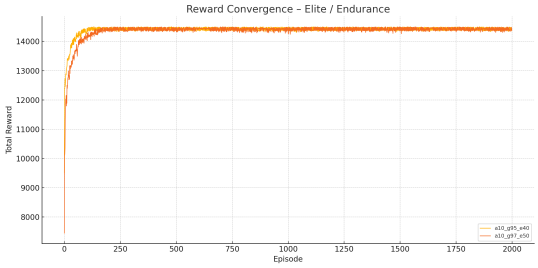
(a) Heatmap for the *amatour* profile after 2000 episodes



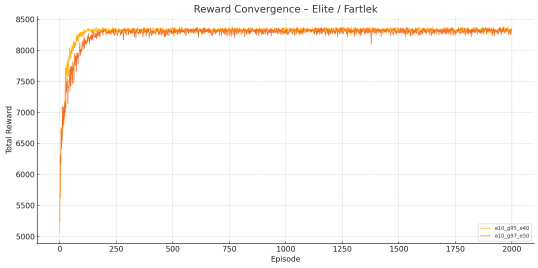
(b) Heatmap for the *runner* profile after 2000 episodes

(c) Heatmap for the *elite* profile after 2000 episodes

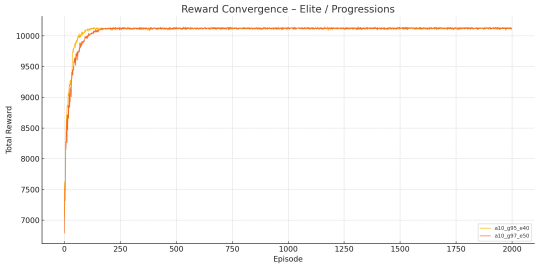
Figure 2: Heatmaps of total rewards for each athlete after 2000 episodes.



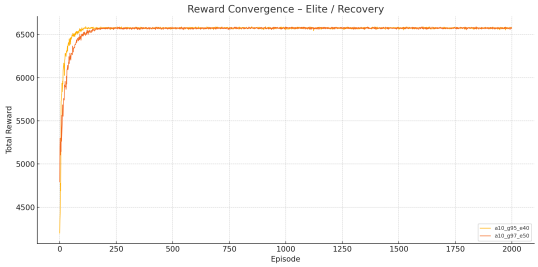
(a) Endurance



(b) Fartlek

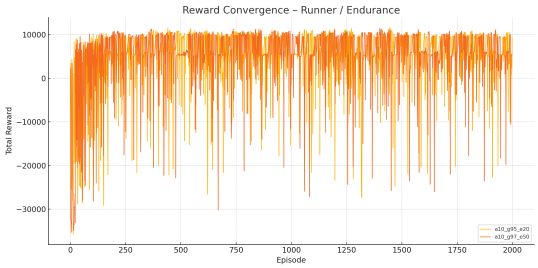


(c) Progression

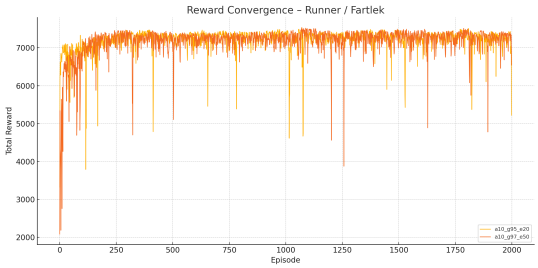


(d) Recovery

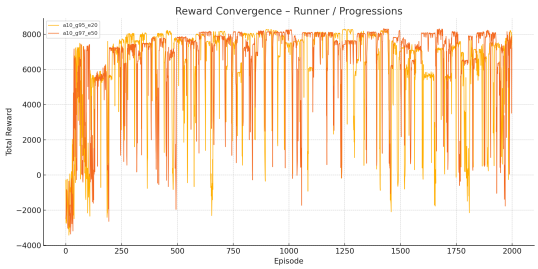
Figure 3: Reward convergence for **Elite** (2000 episodes)



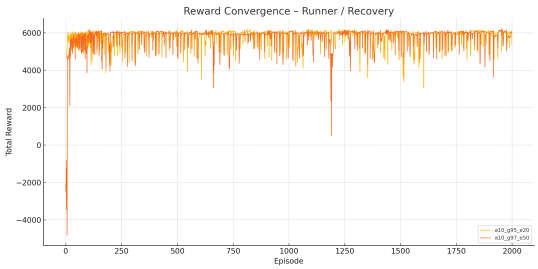
(a) Endurance



(b) Fartlek

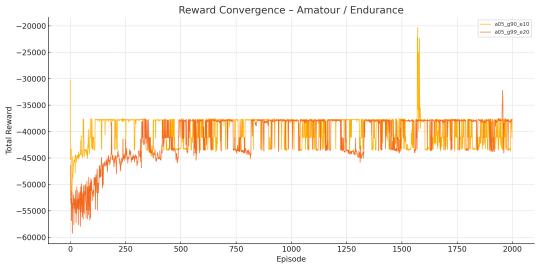


(c) Progression

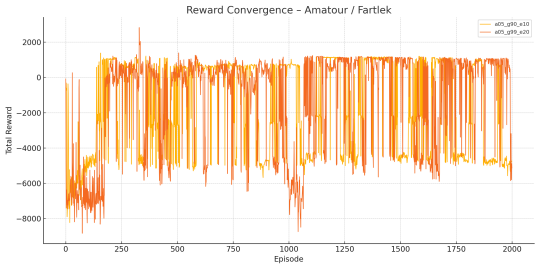


(d) Recovery

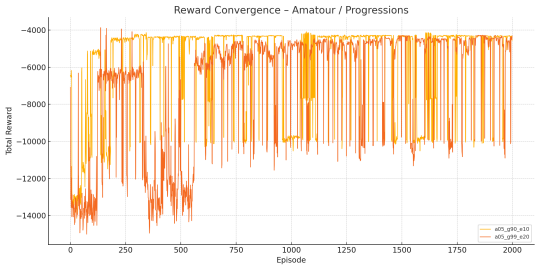
Figure 4: Reward convergence for **Runner** (2000 episodes)



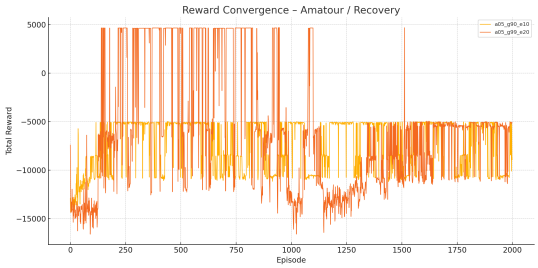
(a) Endurance



(b) Fartlek



(c) Progression



(d) Recovery

Figure 5: Reward convergence for **amateur** (2000 episodes)

5.1 Run the simulation

The application is designed to run in a terminal environment, that can be done by exectuing the `main.py` file, which is the entry point of the application. After start it, by terminal will be asked information about the athlete in order to find the most close archetypes profile to the user, as defined in section 2. The user will be prompted to provide the following information:

- Resting heart rate (HR$_rest$)$Maximumheartrate(HR_max)$

- Functional Threshold Power (FTP)
- Body weight in kg

After that, the user will be asked to select the type of workout they want to perform, choosing from the following options:

- Fartlek
- Progression
- Endurance
- Recover

And in the end will be asked if the user wants to create a MQTT session to simulate the communication with the smartwatch application.

5.2 MQTT Messages

The communication occurs over the topic `smartpacer/action` using the public broker `broker.emqx.io`. The smart pacer publishes messages at a frequency of 1 Hz, meaning it sends updates every second during the workout session. This frequency is chosen to provide timely feedback to the athlete, allowing them to adjust their pace and actions in real-time based on the smart pacer's suggestions. The payload of each MQTT message is a JSON object containing the following fields:

- `second`: the current second of the workout;
- `phase`: the current phase of the workout;
- `fatigue`: the athlete's current fatigue level;
- `action`: the action suggested by the smart pacer, which can be one of *accelerate*, *hold*, or *ease*.

Each field is represented as a string, accompanied by a relevant emoji to enhance the user experience. An example of the messages are shown in figures:

6 Challenges and Future Improvements

6.1 State-space explosion

The current Markov model already spans seven discrete variables, yielding several thousand reachable combinations. Extending realism would require additional factors — e.g. *weather* (temperature, humidity, wind) and *surface type* (asphalt, track, trail, grass). Although essential for real-world fidelity, every new dimension inflates the state complexity.

6.2 Richer athlete profiles

Present profiles depend on basic informations and a FTP valu. A more faithful description would incorporate sex, age, injury history, recent training load, previous-night sleep quality, HRV-derived recovery indices and update them with the evolution of the training process. Such metadata would enable session prescriptions that respect female hormonal cycles, age-related recovery kinetics, and cumulative muscle stress.

6.3 Training program definition

Currently, training program definition follows a uniform logic, adjusting only a few key parameters based on the athlete's level. In practice, however, an elite athlete will face more advanced session types than an amateur: for example, longer endurance sessions, fartlek with more prolonged sprints, or a greater variety of specific workouts. Introducing greater variability and complexity in program generation — differentiating session duration, intensity, and structure according to the athlete profile — would certainly increase the learning task's complexity, but at the same time would better simulate reality and provide more personalized and stimulating recommendations.

6.4 Fatigue and reward refinement

Fatigue and reward terms were the hardest features to properly develop. In order to be even more realistic and coherent with real life behaviours could be implemented also a **Injury-risk term**, which will introduce a weekly load delta component that penalises abrupt increases in training load, reducing overuse-injury likelihood. Another possibl improvement could be the introduction of **sleep quality** and **recovery indices** as additional reward terms, which would allow the agent to adapt its pacing strategy based on the athlete's recovery status. This could be particularly useful for athletes with varying sleep patterns or those recovering from intense training sessions.

6.5 Temporal resolution

All decisions are issued at 1 Hz. Increasing the control loop to 5–10 Hz — or adopting event-driven updates triggered by rapid HR changes would shorten feedback latency and smooth the athlete's perceived guidance.

6.6 Personalised on-line learning

After the first ten sessions, enough data exist to characterise an individual pacing style. Fine-tuning the policy with a small neural network head (e.g. policy-gradient or Soft Actor Critic) on top of the pre-trained Q-table would capture personal patterns without restarting from scratch. Meta-learning techniques could further shrink the cold-start phase for new athletes.

Addressing the above points will increase both realism, athlete safety but most importantly will move *Smart Pacer* closer to a deployable digital coach that can adapt to individual needs and conditions, providing a more effective and personalized training experience.