Escape from Tarkov – Weapon Build Optimiser

Overview

This project is a **heuristic-based weapon build optimiser** for *Escape from Tarkov*.
Given a weapon, a set of attachments, and a set of constraints (budget, forced suppressor, weighting of ergonomics vs recoil), the system attempts to generate **strong, competitive weapon builds** without searching the entire solution space.

The goal is **not** to guarantee the mathematically optimal build. Instead, the optimiser aims to reliably produce builds that are *good enough to be difficult to beat*, within practical time constraints.

This project was built as a **portfolio piece, originating from a passion project,** to demonstrate system modelling, heuristic decision-making, and practical software design rather than UI polish or academic optimisation theory.

The Problem

Weapon configuration in Escape from Tarkov is a **combinatorial problem**:

- Each weapon can accept many attachments
- Attachments occupy mutually exclusive slots
- Some attachments depend on others (e.g. suppressors require a muzzle adapter)
- Each attachment affects multiple competing stats
- The total number of valid combinations grows extremely quickly

A brute-force search over all possible builds is **computationally infeasible** once realistic data is used.

This makes the problem well-suited to a **heuristic approach**, where the goal is to explore promising regions of the solution space rather than enumerate everything.

System Design

The project is modelled as a **small interacting system**, not a script.

Core Domain Objects

- **Weapon**
  Represents immutable base weapon statistics (price, recoil, ergonomics).

- **Attachment**
  Represents a single weapon modification, including:

    - Slot/type
    - Price
    - Stat modifiers

- **Build**
  Represents a *mutable aggregate*:

    - Composed of one weapon and multiple attachments
    - Tracks derived state (total price, total ergonomics, total recoil)
    - Enforces compatibility and slot constraints
    - Provides scoring logic for comparison

Derived values such as total price and stats are **cached and updated incrementally** as attachments are added, rather than recalculated from scratch each time.

Constraints Enforced in Code

- Only one attachment per slot/type
- Suppressors require a compatible muzzle adapter
- Budget limits are enforced during build construction

- Builds track a unique “signature” to avoid duplicates

This allows the optimiser to operate entirely on **valid builds**, rather than generating invalid combinations and filtering them later.

Data Handling

Attachment data is loaded from an **external CSV file**, not hardcoded.

- Raw CSV rows are immediately converted into `Attachment` objects
- Optimisation logic operates on objects, not dictionaries or CSV rows
- This separation allows the system to adapt easily if attachment data changes

The design intentionally decouples **data representation** from **optimisation logic**, reflecting how real-world systems handle evolving datasets.

Heuristic Approach

This project uses **heuristic decision-making**, not brute force.

Why a Heuristic?

The **Real Tarkov** solution space is too large to search exhaustively. Instead, the optimiser focuses on:

- Reasonable trade-offs
- Stat weighting
- Controlled exploration

The result is not guaranteed to be optimal, instead aiming for **consistently strong candidate builds**.

Scoring Logic

Attachments and builds are scored using a **weighted function**:

- Recoil reduction vs ergonomics gain
- Weights are configurable to bias toward different playstyles
- Scores are used to rank attachments and completed builds

Strategies & Exploration

The optimiser combines several simple but effective techniques:

- **Strategy-based selection**
  Attachments may be chosen based on:
    - Best stat value
    - Highest price
    - Lowest price
      This biases exploration toward different regions of the solution space.
- **Controlled randomness (“noise”)**
  Small random variations are introduced during scoring to:
    - Avoid getting stuck in local optima
    - Encourage diversity in generated builds
    - Produce slightly different results across runs

This randomness is deliberate and limited. The optimiser is *stochastic*, not chaotic.

Limitations (by design)

- This is not a genetic algorithm or formal optimiser
- No guarantee of global optimality

- Simplified compatibility rules
- Static prices (no market volatility)

These trade-offs were accepted to keep the system **understandable, fast, and maintainable**.

Interface & Usability

A minimal **Tkinter GUI** is provided to make the system usable:

- Toggle constraints such as forced suppressor
- Adjust budget limits
- Run the optimiser interactively
- View the top candidate builds and their stats

The interface is intentionally lightweight. The goal is **usability**, not visual polish.

Assumptions & Simplifications

To keep the scope reasonable:

- Attachment compatibility rules are simplified
- All prices are treated as static inputs
- Only a fixed set of slots is considered
- Tactical attachments are not fully modelled (placeholder for future expansion)

These assumptions are explicit and intentional.

Future Improvements

Potential extensions include:

- More complete attachment dependency modelling
- Dynamic pricing (market data integration)
- Genetic algorithms or evolutionary strategies
- Better exploration of attachment combinations
- Separation of optimiser logic into a reusable library

What This Project Demonstrates

- System modelling using interacting classes
- Explicit constraint enforcement
- Heuristic decision-making under uncertainty
- Trade-offs between optimality and practicality
- Working with structured external data
- Building usable tooling, not just scripts

This project is not intended to be “finished”, it is a **minimum viable demonstration** of problem-solving, reasoning, and engineering judgement.

A separate version of this project remains in development as a passion project, intended to represent a fully ‘finished’ version.