

Pakiety gier

“ Źródło: [repo luneta/docs/GAME-PACKS.md](#) — edytuj w repo, nie tutaj.

Pakiety gier: obsługa nowej gry bez programowania

Luneta ma dwie wbudowane gry: Blue Archive i NTE. Kolejną grę dodasz **pakiem**. Pakiet to katalog z plikiem `game.json`. Opisujesz w nim, gdzie na zrzucie okna gry są liczby i co z nich wynika. Agent wczytuje pakiety przy starcie.

Pakiet nie zawiera kodu ani skryptów i nie ma dostępu do sieci. Opisuje tylko obraz okna gry, więc zasady bezpieczeństwa Lunety obowiązują bez zmian: zero czytania pamięci gry, zero wstrzykiwania, zero klikania za gracza.

Gdzie leżą pakiety

system	katalog
Linux	<code>~/.local/share/pl.ariku.luneta/games/<klucz>/</code>
Windows	<code>%APPDATA%\pl.ariku.luneta\games\<klucz>\</code>
test / dev	katalog wskazany zmienną <code>LUNETAGAMESDIR</code>

W katalogu pakietu są dwa pliki:

- `game.json`: opis gry (piszesz sam);
- `screens.json`: model punktów charakterystycznych ekranów (generuje `luneta learn`, opcjonalny).

Przykład: `games/nte_demo/game.json`. Sprawdzisz go poleceniem `LUNETAGAMESDIR=games luneta screens nte_demo <klatki>`.

Krok po kroku

1. **Szkielec:** `luneta game new moja_gra --label "moja gra" --process MojaGra.exe`. Klucz musi być taki sam jak klucz gry na serwerze, bo stamtąd przychodzą waluty i zadania.
2. **Proces:** uruchom grę i wpisz `luneta detect`. Do `processes` wpisz nazwę pliku `.exe` z listy. Popraw `windowTitles`, jeśli trzeba.
3. **Klatki:** `luneta map-session --game moja_gra --minutes 20`. Chodź po ekranach, które chcesz śledzić. Klatki zapisują się lokalnie w `mapping/`.
4. **Tekst z klatek:** `luneta ocr-lines <klatka.jpg>` wypisze linie OCR z pozycjami `[x, y, szerokość × wysokość]`. Z nich wybierzesz słowa kluczowe ekranów i obszary pól.
5. **Opis ekranów i pól** w `game.json` (format niżej). Sprawdzaj na bieżąco: `luneta screens moja_gra mapping/<katalog>`, a dla całego pliku `luneta game check moja_gra`.
6. **Punkty charakterystyczne:** `luneta learn moja_gra ~/.local/share/pl.ariku.luneta/games/moja_gra/screens.json mapping/<katalog>`. Wiersz `sprawdzenie:` ma pokazać `inaczej 0`.
7. **Restart agenta.** Gra pojawi się w dniówkach, auto-skanie i wykrywaniu.

Format `game.json`

Wszystkie współrzędne podajesz w pikselach klatki **1920×1080** jako `[x0, y0, x1, y1]`. Inna rozdzielczość jest skalowana do tej.

```
{
  "key": "moja_gra",
  "label": "moja gra",
  "processes": ["MojaGra.exe"],
  "windowTitles": ["Moja Gra"],
  "screens": [
    { "name": "*", "fields": [
      { "key": "gems", "kind": "currency", "rect": [1600, 20, 1760, 70] }
    ] },
    { "name": "Dungeon", "keywords": ["Dungeon"], "any": ["Enter", "Sweep"],
      "keywordArea": [0, 0, 600, 120],
      "fields": [
        { "key": "keys", "kind": "counter", "area": [1300, 900, 1900, 1000], "pattern":
"Keys\\s*(\\d+)\\s*/\\s*(\\d+)" }
      ] }
  ],
  "rules": [
```

```
{ "field": "keys", "task": "daily_dungeon", "done": "zero", "remaining": true }
]
```

Ekrany (screens)

pole	znaczenie
name	nazwa ekranu (dowolna). "*" = pola czytane na każdym ekranie, np. pasek walut
keywords	wszystkie te napisy muszą być na klatce. Wielkość liter ma znaczenie, to fragmenty tekstu z OCR
any	i co najmniej jeden z tych napisów
keywordArea	opcjonalnie: napisy muszą leżeć w tym prostokącie, np. tytuł w rogu

Ekrany sprawdzane są w kolejności z pliku i wygrywa pierwszy pasujący. Dlatego ekrany bardziej szczegółowe wpisz wyżej.

Pola (fields)

kind	wynik	jak czytać
currency	waluta, czyli snapshot na serwer (klucz waluty jak na serwerze)	rect (OCR cyfr w stałym miejscu, najpewniejsze) albo pattern z grupą 1
counter	licznik N/M (bilety, stamina, pity)	rect albo pattern z grupami 1 i 2; przy jednej grupie podaj max. Bez pattern szuka „N/M”
percent	procent	rect albo pattern z grupą 1
timer	czas hh:mm:ss (> 0 = coś w toku)	pattern z 3 grupami
label	tekst, np. nazwa banera	pattern z grupą 1

area zawęży szukanie wzorca do linii OCR, których środek leży w prostokącie. Nie przyspiesza OCR: agent robi jeden OCR całej klatki, bo model detekcji ocrs ma stałe wejście 600×800 i osobny OCR każdego pola kosztuje tyle co cała klatka (pomiar luneta perf 2026-10-05).

pattern to wyrażenie regularne (składnia Rust regex). W JSON odwrotny ukośnik piszesz podwójnie: \\d.

Reguły (rules): wnioski o zadaniach

pole	znaczenie
field	klucz pola
task	klucz zadania na serwerze (dniówki, tygodniówki)
done	zero (0/N, np. bilety zużyte), full (N/N), below_max (< N, np. była walka), nonzero (> 0, np. czas w toku), at_least:<liczba>
remaining	licznik pokazuje, ile zostało (np. „Gift 0/3”). Postęp to wtedy max – wartość

Spełniona reguła odhacza zadanie raz na okres i pokazuje „✓ zaliczone” na nakładce. Niespełniona pokazuje „postęp”.

Dobre praktyki

- **Waluty** czytaj z rect w stałym miejscu. OCR cyfr na małym wycinku jest dokładniejszy niż szukanie wzorcem w całej klatce.
- **Prostokąt** obejmuje samą liczbę, bez ikony. Ikona bywa czytana jako „1”.
- **Słowa kluczowe** wybieraj z napisów, które zawsze są na ekranie: tytuł, zakładki, przyciski. Nie bierz tekstu, który się zmienia, np. nazw banerów.
- **Nowe klatki:** gdy auto-skan czegoś nie rozumie, zapisuje klatkę w dzienniku problemów (luneta review). Te klatki dobrze nadają się do kolejnego luneta learn .
- **Serwer:** gra musi mieć na serwerze ten sam klucz, swoje waluty i zadania. Nieznane serwerowi waluty agent pomija.

Kiedy pakiet nie wystarczy

Logika, której nie da się zadeklarować, wymaga modułu wbudowanego w Ruście (src-tauri/src/modules/ , wpis w builtin()). Przykłady: kolor ikony decyduje o walucie, biblioteka postaci z dopasowaniem do bazy, kilka układów paska zależnie od ekranu. Tak są zrobione Blue Archive i NTE.