

Jenga – Package Python

Ce répertoire contient le **package Python Jenga**, cœur du système de build Jenga.

Il expose l'API DSL, les commandes CLI, le moteur de build, les utilitaires et le framework de tests unitaires.

📋 Sommaire

- [Structure du package](#)
 - [Installation du package](#)
 - [Utilisation en tant que bibliothèque](#)
 - [Modules principaux](#)
 - [Exemple d'intégration](#)
 - [Contribuer au package](#)
-

🏗 Structure du package

```
Jenga/
├── __init__.py          # Exposition de l'API publique, version
├── Api.py               # DSL, context managers, énumérations, dataclasses
├── Jenga.py              # Point d'entrée CLI (console_script)
├── Commands/            # Implémentation des commandes CLI
├── Core/                 # Moteur de build (loader, cache, builder, ...)
├── Unittest/             # Framework de tests C++ (sources et binaires)
└── Utils/                # Utilitaires transversaux (console, fs, process, ...)
```

Le package est conçu pour être **installable** via [pip](#) et fournit un point d'entrée [Jenga](#).

📦 Installation du package

Depuis les sources (développement)

```
git clone https://github.com/RihenUniverse/Jenga.git
cd Jenga
pip install -e .
```

Depuis PyPI (à venir)

```
pip install Jenga
```

Dépendances

Le fichier `requirements.txt` liste les dépendances minimales :

- `watchdog>=2.1.0` (recommandé pour `Jenga watch`)
- `colorama>=0.4.4` (Windows uniquement)
- `requests>=2.28.0` (pour les commandes `publish`, optionnel)

⌚ Utilisation en tant que bibliothèque

Vous pouvez importer Jenga dans vos propres scripts Python pour manipuler des workspaces, projets ou toolchains par programmation.

```
from Jenga import Workspace, Project, project, workspace
from Jenga.Utils import Colored

# Créer un workspace en mémoire
with workspace("MonWorkspace"):
    with project("MonProjet"):
        consoleapp()
        files(["src/**.cpp"])
        Colored.PrintSuccess("Projet configuré")
```

L'API exposée dans `__init__.py` est identique à celle de `Api.py`.

❖ Modules principaux

Module	Description
<code>Api</code>	DSL, context managers, énumérations, dataclasses (Workspace, Project, ...)
<code>Commands</code>	Toutes les commandes CLI (<code>build</code> , <code>run</code> , <code>test</code> , ...)
<code>Core</code>	Moteur de build : Loader, Cache, Builder, Platform, Toolchains, Daemon
<code>CoreBuilders</code>	Implémentations concrètes pour chaque plateforme (Windows, Linux, ...)
<code>Unittest</code>	Framework de tests unitaires C++ (sources et binaires précompilés)
<code>Utils</code>	Outils génériques : console colorée, fichiers, processus, rapports, UI

Chaque module possède son propre fichier `README.md` détaillant son fonctionnement et ses conventions.

🔌 Exemple d'intégration

Utiliser Jenga comme bibliothèque pour automatiser la configuration d'un workspace :

```

import sys
from pathlib import Path
from Jenga import workspace, project, consoleapp, files, includedirs
from Jenga.Core.Loader import Loader
from Jenga.Core.Cache import Cache

# Création programmatique d'un fichier .jenga
def generate_workspace(name: str, path: Path):
    with open(path / f"{name}.jenga", "w") as f:
        f.write(f'from Jenga.Api import *\n\n')
        f.write(f'with workspace("{name}":\n')
        f.write(f'    configurations(["Debug", "Release"])\n')
        f.write(f'    with project("App"):\n')
        f.write(f'        consoleapp()\n')
        f.write(f'        files(["src/**.cpp"])\n')
    print(f"✓ Workspace {name} créé")

if __name__ == "__main__":
    generate_workspace("MyGame", Path.cwd())

```

🤝 Contribuer au package

- Cloner le dépôt** et installer en mode développement (`pip install -e .`).
- Respecter les conventions de nommage** (PascalCase, _PascalCase, _camelCase, UPPER_SNAKE_CASE, lower).
- Tester les modifications** : exécuter les tests unitaires (à venir) et vérifier le bon fonctionnement des commandes.
- Documenter** toute nouvelle fonctionnalité dans le README approprié.
- Soumettre une pull request** sur GitHub.

📄 Licence

Propriétaire (à définir). Pour tout usage, contacter l'équipe Jenga.

🔗 Liens connexes

- Documentation utilisateur (dépôt racine)
- Guide des commandes
- Moteur de build
- Builders
- Framework de tests Unitest
- Utilitaires

Ce fichier fait partie du package Jenga. Toute suggestion est la bienvenue.

```
## 📁 `README.md` (racine du projet)
```

```
```markdown
```

```
Jenga – Build System Cross-Plateforme
```

\*\*Jenga\*\* est un système de build \*\*complet\*\*, \*\*modulaire\*\* et \*\*extensible\*\* pour les projets C, C++, Objective-C, Assembly, Rust, Zig et autres langages natifs.

Il permet de compiler, tester, packager et déployer des applications sur \*\*Windows\*\*, \*\*Linux\*\*, \*\*macOS\*\*, \*\*Android\*\*, \*\*iOS\*\*, \*\*Web (Wasm)\*\*, \*\*Xbox\*\*, \*\*PlayStation\*\* et \*\*Nintendo Switch\*\*.

---

```
📄 Sommaire
```

- [Pourquoi Jenga ?](#-pourquoi-jenga-)
- [Fonctionnalités clés](#-fonctionnalités-clés)
- [Prérequis](#prérequis)
- [Installation](#installation)
- [Premiers pas](#premiers-pas)
- [Documentation](#documentation)
- [Contribuer](#contribuer)
- [Licence](#licence)

---

```
⚙️ Pourquoi Jenga ?
```

Jenga est né du besoin d'un outil de build \*\*unifié\*\* pour des projets complexes ciblant de multiples plateformes, sans sacrifier la performance ni la flexibilité.

Contrairement à CMake, il ne génère pas de fichiers intermédiaires : il \*\*compile directement\*\* en utilisant les toolchains natives.

Contrairement à des systèmes comme Meson ou Scons, il propose un \*\*DSL intégré en Python\*\* extrêmement lisible et puissant.

**Objectifs :**

- **Rapidité** : parsing une seule fois, cache intelligent, daemon en arrière-plan.
- **Simplicité** : une syntaxe déclarative, des conventions fortes, zéro fichier projet à générer.
- **Cross-compilation** : build pour n'importe quelle cible depuis n'importe quel hôte.
- **Extensibilité** : ajout facile de nouvelles plateformes, toolchains, commandes.
- **Intégration** : support natif des tests unitaires, benchmarks, profiling, packaging, déploiement.

## ---

### ## ♦ Fonctionnalités clés

- \*\*DSL Python\*\* avec context managers (`with workspace():`, `with project():`, ...).
- \*\*Gestion intelligente du cache\*\* (SQLite) - rechargement incrémental.
- \*\*Daemon\*\* pour des commandes instantanées (50-200ms).
- \*\*Support de plus de 20 plateformes\*\* (Windows, Linux, macOS, Android, iOS, Web, Xbox, PS4/5, Switch, ...).
- \*\*Détection automatique des toolchains\*\* (MSVC, GCC, Clang, NDK, Emscripten, ...).
- \*\*Compilation incrémentale\*\* par hash de contenu.
- \*\*Tests unitaires intégrés\*\* avec Unittest (framework C++ moderne).
- \*\*Benchmarking et profilage\*\* (Google Benchmark, perf, Instruments, ...).
- \*\*Packaging\*\* : APK, AAB, IPA, MSI, EXE, DEB, RPM, AppImage, DMG, ZIP.
- \*\*Déploiement\*\* : adb, ios-deploy, xbapp, copie locale.
- \*\*Publication\*\* sur registres (NuGet, vcpkg, Conan, npm, PyPI – en cours).
- \*\*Modules C++20\*\* supportés (MSVC, Clang, GCC).
- \*\*Générateurs de projets\*\* : CMake, Makefile, Visual Studio 2022.

## ---

### ## 🎯 Prérequis

- \*\*Python 3.8 ou supérieur\*\*.
- \*\*Compilateurs\*\* : selon les cibles (MSVC, GCC, Clang, NDK, Emscripten, ...).
- \*\*Outils optionnels\*\* :
  - `watchdog` (pour `Jenga watch`)
  - `keytool` (pour `Jenga keygen`)
  - `apksigner`, `bundletool` (Android)
  - `ios-deploy`, `create-dmg`, `pkgbuild` (macOS/iOS)
  - WiX Toolset, Inno Setup (Windows)
  - `dpkg-deb` (Linux)
  - `xbapp`, `MakePkg` (Xbox – nécessite GDK)

## ---

### ## 📦 Installation

#### #### Depuis les sources

```
```bash
git clone https://github.com/RihenUniverse/Jenga.git
cd Jenga
pip install -e .
```

Via pip (bientôt disponible)

```
pip install Jenga
```

Vérifiez l'installation :

```
Jenga --version
```

🚀 Premiers pas

1. Créer un nouveau workspace

```
Jenga workspace MonJeu --interactive  
cd MonJeu
```

2. Ajouter un projet

```
Jenga project Moteur --kind static
```

3. Compiler

```
Jenga build
```

4. Tester

```
Jenga test
```

5. Packager

```
Jenga package --platform windows --type zip
```

📖 Documentation

La documentation complète est organisée par module dans le dépôt :

Module	Description
Jenga/	Package Python – API et point d'entrée
Jenga/Commands/	Toutes les commandes CLI

Module	Description
Jenga/Core/	Moteur de build (loader, cache, builder, ...)
Jenga/Core/Builders/	Implémentations plateforme
Jenga/Unitest/	Framework de tests C++
Jenga/Utils/	Utilitaires transversaux

Consultez également le [Guide de contribution](#) (à créer).

Contribuer

Les contributions sont les bienvenues !

Merci de lire le [guide du contributeur](#) avant de soumettre une pull request.

Rappel des conventions de nommage :

- PascalCase : classes, méthodes publiques, énumérations
- _PascalCase : méthodes privées
- lower : fonctions DSL utilisateur (un seul mot, pas de _)
- _camelCase : attributs privés/protégés
- camelCase : attributs internes (non publics)
- UPPER_SNAKE_CASE : constantes, valeurs d'enum

Licence

Ce projet est sous licence propriétaire.

Pour toute demande d'utilisation, veuillez contacter l'équipe Jenga à l'adresse team@jenga.build.

Remerciements

Jenga s'appuie sur des projets open-source formidables :

- [watchdog](#)
- [colorama](#)
- [requests](#)
- et bien sûr la bibliothèque standard Python.

Documentation générée le 12 février 2026.