Documentation Technique – **Clash of Piglin**

Table des matières

1. Informations générales

Élément	Détail
Titre	Documentation Techinque - Clash of piglin
Version	1.0
Auteur	Zwartkat
Date de révision	18/10/2025
Référence du document	DOC-GD-TECH-001
Public cible	Développeurs, testeurs, intégrateurs
Langage	Python 3.11+
Bibliothèques principales	Pygame 2.6, esper

2. Objet du document

Ce document décrit l'**architecture logicielle**, les **modules**, la **procédure d'installation**, les **configurations**, et la **structure du code** du jeu **Clash of piglin**.

L'objectif est de permettre à tout développeur de comprendre le fonctionnement interne du jeu, de le modifier ou de le maintenir.

3. Références normatives

- ISO/IEC/IEEE 26514:2008 — *Design and development of information for users*
- Documentation Pygame
- Python 3.11

---

4. Vue d’ensemble du jeu

Clash of piglin est un jeu type RTS inspiré de Minecraft Deux joueurs s'affrontent dans l'objectif de détruire le bastion de l'adversaire.

Fonctionnalités principales

- Déplacement des troupes
- Achat de troupes
- Système de génération d'argent automatique
- Interface graphique simple avec menus.

---

5. Environnement logiciel requis

Composant	Spécification
Système d’exploitation	Windows 7 ou supérieur, macOS 10.12 ou supérieur, Linux récent
Mémoire vive (RAM)	2 Go
Stockage	50 Mo d’espace libre
Python	3.11 ou supérieur
Bibliothèques	Pygame 2.6, autres dépendances via `requirements.txt`
Résolution d’écran	800x600 minimum

6. Procédure d’installation

6.1. Cloner le dépôt

```
git clone https://github.com/Zwartkat/Clash-of-Piglin
cd Clash-of-Piglin
```

6.2. Installer les dépendances

```
pip install -r requirements.txt
```

6.3. Lancer le jeu

```
python ./src/main.py
```

7. Exploitation du jeu

Cette section décrit comment **démarrer, arrêter** le jeu Clash of Piglin.

7.1 Démarrage du jeu

- Assurez-vous que toutes les dépendances sont installées :

```
pip install -r requirements.txt
```

Lancer le jeu depuis le répertoire src/ :

```
python main.py
```

Le menu principal apparaît et vous pouvez :

- Démarrer une partie
- Consulter les crédits
- Quitter

7.2 Arrêt du jeu

Le jeu peut être arrêté via :

- Le menu principal → Quitter
- Echap à la fin de partie
- La fermeture de la fenêtre (non recommandé)

8. Structure du code source

```
src/
│
├── main.py                    # Point d'entrée principal du jeu
│
├── components/                # Composants du système ECS
│   ├── base/                  # Composants de base : santé, position, équipe,
etc.
│   ├── gameplay/              # Composants liés aux mécaniques de jeu (attaque,
sélection)
│   └── rendering/             # Composants graphiques (sprites, visuels)
│
├── config/                    # Configuration du jeu (unités, terrains,
couches)
```

```
│   ├── layer.py                # Définit les couches d'affichage
│   ├── terrains.py             # Définit les types de terrains
│   └── units.py                # Définit les caractéristiques des unités
│
├── core/                       # Noyau du moteur de jeu
│   ├── config.py               # Gestion globale de la configuration
│   ├── engine.py               # Boucle principale du moteur de jeu
│   ├── services.py             # Gestion des services internes (event bus,
gestion de joueur, etc.)
│   │
│   ├── ecs/                    # Système ECS (Entity-Component-System)
│   │   ├── component.py        # Classe abstraite représentant un composant
│   │   ├── entity.py           # Classe abstraite représentant une entité
│   │   ├── event.py            # Classe abstraite représentant un événement
│   │   ├── event_bus.py        # Bus d'événements centralisé
│   │   └── iterator_system.py  # Gestion des itératives des entités dans le
moteur ECS
│   │
│   ├── game/                   # Gestion du monde et des entités de jeu
│   │   ├── camera.py           # Gestion de la caméra
│   │   ├── map.py              # Gestion de la carte
│   │   ├── player.py           # Définition du joueur
│   │   ├── player_manager.py   # Gestion multi-joueur / joueurs IA
│   │   └── terrain.py          # Gestion des terrains du jeu
│   │
│   └── input/                  # Gestion des entrées utilisateur
│       └── input_manager.py    # Mappage clavier/souris et actions
│
├── enums/                      # Déclarations d'énumérations globales
│   ├── case_type.py            # Types de cases
│   ├── input_actions.py        # Actions d'entrée (clic, flèches, etc.)
│   ├── source_effect.py        # Origine des effets (terrain, balise, etc.)
│   │
│   └── entity/                 # Enums liées aux entités
│
├── events/                     # Événements ECS (communication entre systèmes)
│   ├── attack_event.py         # Événement d'attaque
│   ├── move_order_event.py     # Ordre de mouvement
│   ├── select_event.py         # Sélection d'unité
│   ├── spawn_unit_event.py     # Apparition d'unité
│   ├── death_event.py          # Mort d'une entité
│   ├── victory_event.py        # Condition de victoire
│   └── ...                     # (autres événements caméra, zoom, etc.)
│
├── factories/                  # Création d'entités et d'unités
│   ├── entity_factory.py       # Fabrique d'entités génériques
│   └── unit_factory.py         # Fabrique d'unités spécifiques (chargées depuis
la configuration)
│
├── systems/                    # Systèmes ECS (logique du jeu)
│   ├── combat/                 # Systèmes de combat (flèches, cibles, troupes)
│   ├── input/                  # Systèmes liés aux entrées et caméra
│   ├── rendering/              # Systèmes d'affichage et HUD
│   ├── world/                  # Systèmes du monde (économie, mouvement,
```

```
collisions)
│   ├── death_event_handler.py   # Gestion de la mort des entités
│   ├── quit_system.py           # Fermeture propre du jeu
│   └── victory_system.py        # Gestion des conditions de victoire
│
└── ui/                          # Interface utilisateur
│   └── hud.py                   # Affichage du HUD (interface de jeu)
```

8.1 Description des modules principaux

Module	Rôle et responsabilités
main.py	Point d'entrée du jeu. Initialise le moteur, charge la configuration et lance la boucle principale. Ouvre le menu principal du jeu.
core/	Contient le moteur principal et les services. Gestion de la boucle de jeu, des entités, événements et entrées.
core/ecs/	Implémente le pattern ECS : définition des entités, composants, systèmes et bus d'événements.
components/	Définit les composants réutilisables : santé, position, attaque, etc.
systems/	Contient la logique du jeu par système : combat, mouvement, rendu, économie, input, etc. Chaque système manipule les entités via leurs composants.
config/	Fichiers de configuration : unités, terrains, couches de rendu. Permet de modifier les paramètres du jeu sans changer le code.
enums/	Déclarations d'énumérations globales pour types d'entités, actions, effets et cases.
events/	Événements ECS pour la communication entre systèmes (attaques, mouvements, sélections, victoire, mort).
factories/	Création et initialisation des entités à partir des composants et des fichiers de configuration.
ui/	Gestion du HUD et de l'interface utilisateur. Affiche les informations du joueur et l'état du jeu. Permet au joueur de voir et d'intéragir avec le jeu.

9. Fichiers de données

Le jeu utilise plusieurs types de fichiers de données externes :

Dossier / Fichier	Rôle	Format / Contenu
`config/layer.py`	Définit les couches d'affichage du moteur (priorité de rendu).	Script Python
`config/terrains.py`	Spécifie les types de terrains et leurs effets sur les unités.	Script Python

Dossier / Fichier	Rôle	Format / Contenu
`config/units.py`	Définit les caractéristiques de chaque unité (PV, attaque, portée, coût).	Script Python
`assets/`	Contient les ressources graphiques et audio du jeu.	Images et sons
`config.yaml`	Contient les configurations générales et les textes	Données au format YAML

10. Sécurité et intégrité

Le jeu applique plusieurs mécanismes pour assurer la sécurité et l'intégrité des données :

- **Validation des entrées utilisateur**
  Toutes les entrées sont contrôlées avant traitement (clavier, souris, sélection d'unité) pour éviter les états invalides.
- **Intégrité des fichiers de configuration et ressources**
  Les fichiers de `/config/` et `/assets/` sont vérifiés au lancement pour garantir qu'ils sont complets et cohérents.
- **Intégrité des entités et du moteur ECS**
  Les systèmes (combat, mouvement, rendu) appliquent des contraintes pour éviter les actions illégales (attaques sur unités mortes, mouvements hors carte).

11. Maintenance et mise à jour

Mise à jour du jeu :

Tirer les dernières modifications :

```
git pull origin main
```

Ajout de nouvelles entités :

- Créer le nouveau type d'entité dans `enums/entity/entity_type.py`
- Ajouter dans la constante `UNITS` la clé correspondant à l'entité et lui assigner un objet entité contenant les composants correspondants à l'entité

```
UNITS = {
    EntityType.NEW : Entity(
        components=[
            #Liste des composants de l''entité
        ])
}
```

12. Erreurs courantes et solutions

Erreur	Cause possible	Solution
`ModuleNotFoundError: No module named 'pygame'`	Pygame non installé ou mauvaise version	Exécuter `pip install -r requirements.txt` ou vérifier la version de Python et Pygame
Une unité ne répond pas aux ordres	Composants manquants ou mal initialisés dans `UNITS`	Vérifier que tous les composants nécessaires sont inclus et correctement importés
Erreur de chargement d'un fichier de configuration	Fichier manquant ou syntaxe invalide dans `config/*.py` ou `config.yaml`	Vérifier la présence du fichier et sa syntaxe. Pour YAML, utiliser un validateur YAML
Actions utilisateur non détectées (clic, touches)	Mauvais mapping clavier/souris ou conflit dans `input_manager.py`	Vérifier que `input_actions.py` correspond aux touches utilisées et que `input_manager` est bien initialisé
Sprites ou ressources graphiques manquants	Fichier absent dans `assets/` ou chemin incorrect	Vérifier la présence et le chemin des fichiers images/audio utilisés par les composants `sprite`
Problème de caméra	Mauvaise récupération de la taille de la fenêtre	Redimensionner la fenêtre
Impossible de cloner ou pull le dépôt Git	Problème de droits ou URL incorrecte	Vérifier l'URL et vos droits Git, utiliser `git clone https://github.com/Zwartkat/Clash-of-Piglin`

13. Glossaire

Terme	Définition
ECS (Entity-Component-System)	Architecture logicielle qui sépare les entités (objets du jeu), leurs composants (données) et les systèmes (logique de traitement).
Entité (Entity)	Objet du jeu unique (unité, bâtiment, projectile) constitué d'un ou plusieurs composants.
Composant (Component)	Conteneur de données d'une entité (ex : santé, position, vitesse, attaque).
Système (System)	Module qui applique la logique du jeu aux entités selon leurs composants (ex : combat, mouvement, rendu).
Event / Événement	Message ou signal envoyé entre systèmes pour déclencher des actions (ex : attaque, déplacement, mort).
Bus d'événements (Event Bus)	Mécanisme centralisé pour diffuser les événements aux systèmes concernés.

Terme	Définition
HUD (Head-Up Display)	Interface utilisateur affichant les informations de jeu : ressources, points de vie, sélection d'unités.
Configuration (Config)	Fichiers ou scripts définissant les paramètres du jeu, comme les unités, terrains et couches de rendu.
Assets	Ressources graphiques et audio utilisées par le jeu (images, sprites, sons).
Python	Langage de programmation utilisé pour développer le jeu.
Pygame	Bibliothèque Python utilisée pour gérer les graphismes, les entrées et le son.
Terrain	Case ou zone du jeu qui peuvent avoir des propriétés particulières (effets sur le mouvement, etc.).
Répertoire racine (`src/`)	Dossier contenant l'ensemble du code source et des modules du jeu.

14. Support et contact

Pour tout problème ou question, contacter :

Zwartkat, lorenzovdkn, darkell, workai, SparkasselaBank ou MatthieuPinceel en passant par GitHub Issues

Lien du dépôt : https://github.com/Zwartkat/Clash-of-Piglin

Cette documentation pour Clash of Piglin respecte les préconisations de la norme ISO/IEC/IEEE 26514.