

Documentation technique

LEAD PROD : Emmanuel Schmück

LEAD GAME DESIGN : Pierre-François Dormegnie

LEAD LEVEL DESIGN : Emeric Dos Santos

LEAD TECH : Hanna Fiani

LEAD GRAPHICS : Rémi Moreaux

LEAD QA : Joris Da Silva

[**1 - Architecture générale**](#_4ijyrvsgoajf) **2**

[**2 - Détails des scripts**](#_z8ylm5som47g) **3**

[2.1 - Scripts principaux](#_tzcrbwz0q8hy) 3

[2.2 - Scripts intermédiaires](#_lza4yjs8xyi) 3

[2.3 - Scripts mineurs](#_7yb53zt15zqm) 3

[**3 - Problèmes et solutions**](#_ufq283683mxn) **4**

[**4 - Perspectives**](#_kkz2fbhuvtot) **4**

# 1 - Architecture générale

L’état du jeu est géré par **GameManager (singleton)** : lancement du niveau, progression, respawn, victoire/défaite, reset de la scène. La plupart des autres managers/controllers communiquent avec GameManager.

Le niveau (ville en voxels) est composé d’environ 100.000 cubes ayant chacun le script **InfectionRaycast** qui gère l’infection / réparation et la transmission aux cubes voisins.

L’infection débute lorsqu’un virus atterrit sur un cube. Les virus sont générés semi-aléatoirement par **WaveController (singleton)** dirigé par GameManager. Chaque virus est porteur d’un script **VirusManager** qui gère son état, et d’un script **VirusAnimation** qui gère le visuel.

Le joueur est un prefab instancié sur un spawner au début de la partie. Il porte le script **FirstPersonController** qui gère le déplacement et la caméra et le script **PlayerManager** qui gère le reste. Si le joueur meurt (contact avec un virus ou avec la killbox sous la ville), il est remplacé par un autre objet qui porte le script **PlayerGhost** (déplacement de la caméra vers le spawn).

Les NPC qui peuplent la ville portent les scripts **NPCManager** (déplacement)et **NPCInfection** (état). Ils se déplacent aléatoirement vers des points proches de leur parent en évitant (partiellement) les murs.

Le “ciel” et la “mer” sont créés avec **OceanSpawner** qui instancie des cubes porteurs du script **Oscillator** (qui anime leur position).

# 

# 

# 2 - Détails des scripts

## 2.1 - Scripts principaux

GameManager: Script principal qui gère les mécaniques principales du jeu: StartLevel, GameOver, GameWon, RestartLevel, SpawnPlayer, etc…



InfectionRaycast: un script attaché sur chaque bloc infectable. Il gère les différents stades des infections d’un bloc, sa destruction (en mode transparent) et sa réparation. On utilise un raycast sur les 6 directions suivantes: up, down, left, right, forward, back, pour reconnaître l’environnement de chaque bloc, et on sauvegarde les références dans une ArrayList.

## 2.2 - Scripts intermédiaires

FirstPersonController (Unity Standard Assets, modifié pour nos besoins)

NPCInfection: PNJ s’infecte si le bloc au dessous est infecté

NPCManager: mouvements des PNJ

PlayerManager: script du joueur. Buff, debuff, shoot, initialization, énergie et crosshair

UI\_Manager: tout ce qui est en rapport avec le UI canvas (boussole, barre energie, loading, victory, gameover, etc…)

VirusManager: infection du bloc en contact, mort par le joueur donc le déclenchement des réparations des blocs infectés

WaveController: chaque niveau a des vagues de certains N virus (N varie selon le niveau) avec un temps variable entre chaque apparition de virus

## 2.3 - Scripts mineurs

DisableRendererAtRuntime: l’objet qui a ce script attaché va être désactivé au moment de création

Explosion: l’explosion du virus

GameMenu:

GlowAnimation: l’effet Glow des blocs du ciel et de la mer

KillBox: si le joueur touche un objet qui a ce script, il meurt

Menu: le script qui gère les boutons des menus

VirusAnimation: l’animation du virus

OceanSpawner: la création du ciel et de la mer

Oscillator: l’effet oscillation des blocs du ciel et de la mer

PlayerGhost: quand le joueur meurt, ce script donne un effet ghost lors du respawn

Projectile: le projectile du joueur qui répare les infections

# 

# 

# 3 - Problèmes et solutions

**Globalement : problèmes de performances liées au très grand nombre d’objets (cubes) dans la scène, autant pour le rendu que pour les scripts qu’ils portent.**

Pertes de performances causés par l’utilisation d’instance de matériaux (get/set du renderer.material à la place du sharedMaterial) résolu partiellement en accédant au sharedMaterial.

Pertes de performances liées aux updates simultanées de 100.000 scripts : résolu partiellement :

* en updatant moins souvent (chaque cube désynchronisé pour éviter les pics de lag),
* en ne faire des raycast qu’une seule fois au début pour trouver les voisins (on y accède ensuite par référence)
* en désactivant le script autant que possible (seulement actif si besoin)

Loading de scène très long (>100 secondes) à cause de 100.000 start() simultanés contenants des raycasts : résolu (loading <5 secondes) en initialisant le script dans la première update plutôt que dans start (pas certain de la raison mais ça fonctionne ??).

**Également, quelques problèmes avec Git (manque d’expérience des LD mais aussi des progs).**

# 4 - Perspectives

Pour gagner encore en performances :

texture & material batching

moteur voxel (collision/render)

manager pour l’infection des cubes, plutôt que 100.000 scripts individuels ?

passer les cubes en static

Pour pousser le concept plus loin :

Niveau procédural et/ou dynamique (parties mobiles)

Interaction joueur/virus et joueur/NPC (combat)

Interaction joueur/cube (construction type minecraft ?)