Stickman Brawl — Post-mortem

> Projet individuel — Automne 2025
> Auteur : Antonin Claudel

---

1) Informations générales

Titre du projet : *Stickman Brawl*
Nature du projet : jeu de combat local (2 joueurs) de type *Smash-like*.
Moteur : Unreal Engine 5.6
Technologies : intégralement développées en **Blueprints** (C++ seulement exploré, puis écarté).
Période de réalisation : du **8 septembre 2025** (prototype Zero-G) au **28 octobre 2025**.

Évolution du concept

Le projet est né d'une première expérimentation, *Stickman Zero-G*, centrée sur la simulation physique et la gravité nulle. Cette orientation, rapidement abandonnée pour des raisons de faisabilité et de rendement pédagogique, a conduit à un pivot conceptuel vers un *Smash-like* exploitant pleinement les systèmes internes du moteur (notamment `LaunchCharacter`, `ApplyDamage`, `Jump` et `AddMovementInput`).

Les principales décisions de cadrage ont consisté à :

- Substituer la **caméra dynamique** par une **caméra latérale fixe**, réduisant la complexité d'intégration ;
- Redéfinir la condition de *KO* : abandon du modèle d'éjection au profit d'une élimination à **santé nulle**, formalisée dans les versions V2 et V2.1 du One-Pager ;
- Restreindre le périmètre à un **multijoueur local à deux joueurs fixes**, l'ajout ou le retrait dynamique via C++ (détection des manettes) ayant été jugé non prioritaire dans la contrainte temporelle.

---

2) Objectifs initiaux

L'objectif principal consistait à produire, dans un cadre individuel, un **prototype jouable et stable** démontrant une compréhension approfondie du moteur et un apprentissage concret, tout en demeurant réalisable dans les délais impartis.
Les intentions initiales portaient sur :

- L'expérimentation des **mécaniques de combat rapproché** ;
- La mise en œuvre d'un **multijoueur local robuste**.

Bien que non planifiée au départ, la **maîtrise du Player State et des Data Assets** est devenue un apprentissage central, structurant la réflexion architecturale finale.

---

3) Points de réussite

L'approche méthodique du refactoring a conduit à une architecture claire et efficace, exclusivement en Blueprints. Les classes de base d'Unreal (**GameMode**, **GameState**, **PlayerState**, **PlayerController**, **Character**)

ont été employées selon leurs rôles respectifs, garantissant une séparation nette des responsabilités et un comportement stable du multijoueur 2P.

Les fonctionnalités essentielles — déplacements latéraux, `LaunchCharacter`, `ApplyDamage`, gestion de la santé et des stocks, détection des vainqueurs — ont fonctionné de manière stable dès leur première intégration, réduisant considérablement le temps de débogage ultérieur.
Des sessions de tests régulières ont permis d'affiner le gameplay : augmentation de l'**Air Control**, réduction du *lag* post-attaque et synchronisation plus fluide de la rotation du joueur.

La gestion de version a reposé sur **Git CLI** avec publication sur **GitHub**, suivant un modèle simple mais rigoureux : commits atomiques sur une seule branche `main`, documentation des changements, et usage de **Git Bash**, **Vim**, **PowerShell** et **GH CLI** pour un contrôle précis de l'environnement. Cette approche minimaliste a favorisé la continuité et la stabilité du projet sans surcoût de gestion.

Les choix techniques structurants incluent :

- L'usage des **Maps** pour un accès et une modification plus efficaces des données ;
- L'adoption de **Widgets imbriqués** pour la hiérarchisation de l'interface ;
- L'utilisation des **Event Dispatchers** pour réduire les appels au Tick et améliorer la performance ;
- Le recours à des **Anim Notifies** et des **Sockets** pour la gestion fine des interactions physiques.

Exemples de refactorisation

Initialement, les données de santé et d'état du joueur étaient stockées dans le **Character**, ce qui rendait leur synchronisation complexe lors des transitions de gameplay : accès depuis le Game Mode ou le Widget Blueprint. Une refactorisation a déplacé ces informations dans le **PlayerState**, permettant un accès cohérent via `GetGameState → GetPlayerArray` et une meilleure isolation des responsabilités. Un autre changement a également simplifié la logique des widgets, désormais mis à jour uniquement par événement plutôt que sur Tick continu.

---

4) Problèmes rencontrés et résolutions

Les principales difficultés ont concerné l'ordre d'initialisation des acteurs (widgets, PlayerState/Controller, ID local −1) et la synchronisation entre les différentes phases du cycle de vie des objets Unreal. Ces problèmes ont été résolus par l'ajout de délais contrôlés (`Delay Until Next Tick`) et la réorganisation des appels dans le **Begin Play** ou le **On Possess** selon la logique d'initialisation.

Le module de sélection des personnages a représenté la tâche la plus longue : environ vingt jours étalés de travail. Sa complexité tenait à la combinaison de Data Assets, Widgets imbriqués et gestion du focus multi-joueurs. L'importation d'assets a été stabilisée par la normalisation des Offsets et le recours au modèle « Without Skin ».

Le prototype initial *Zero-G* s'appuyait sur une simulation physique complète avec ragdoll, mais la complexité d'intégration et les problèmes de scale l'ont rendu peu viable. Ce pivot vers le *Smash-like* a permis de recentrer l'apprentissage sur le gameplay et la logique du moteur.

Les erreurs de conception identifiées concernent :

- Une complexité d'interface évitable, qu'un framework comme **CommonUI** aurait pu simplifier ;

- La rotation du personnage, encore gérée via le Tick ;
- Des collisions capsules à optimiser pour des interactions plus nettes.

---

5) Compétences acquises

Compétences techniques

- **Systèmes de mouvement** : meilleure compréhension de `AirControl`, `Jump`, `LaunchCharacter`, `ApplyDamage` et `AnyDamage` ;
- **Gestion de l'état** : usage approfondi de PlayerState, GameState et PlayerController, et compréhension de leur interconnexion ;
- **Structure de données** : utilisation de **Maps** et de **Data Assets** pour une architecture plus souple et performante ;
- **UI et événements** : maîtrise des **Event Dispatchers**, optimisation du cycle d'update des widgets, conception modulaire de l'UI ;
- **Méthodologie logicielle** : refactorisation progressive et réduction mesurée de la dette technique.

Compétences transversales

- **Gestion du temps et des priorités** : maintien du périmètre initial du One-Pager, renoncement stratégique à certaines idées ambitieuses ;
- **Esprit critique et autonomie** : capacité à identifier les solutions les plus rationnelles et à utiliser les outils du moteur plutôt que de les re-coder.

---

6) Perspectives d'amélioration

Les perspectives peuvent être structurées selon trois horizons :

Court terme

- Optimisation du **cœur du gameplay** : ajustement du ressenti des coups, lag post-attaque, gestion du knockback et du hitstop ;
- Amélioration du **HUD** et du **feedback visuel** (effets, sons, impact frames) ;
- Correction du **focus UI** après *alt-tab*.

Moyen terme

- Introduction de mécaniques avancées : saisie, esquive, dash, bouclier, ledge grab, attaques directionnelles, marche différenciée ;
- Refonte de la **caméra** (dynamique) et enrichissement visuel des arènes ;
- Révision des collisions capsules et de la rotation pour plus de précision.

Long terme

- **Internationalisation** du jeu (localisation multilingue) pour renforcer la cohérence de production et l'accessibilité.

Avec deux mois supplémentaires, les efforts porteraient prioritairement sur l'amélioration du ressenti de combat, la variété des coups et la solidité du HUD. Une publication sur *itch.io* demeure envisagée à des fins de diffusion et de visibilité.

---

7) Bilan

L'expérience confirme l'importance de l'ajustement du périmètre et de la priorisation des fonctionnalités. Les objectifs révisés ont été atteints avec un haut degré de satisfaction, illustrant une véritable démarche agile.

Le projet a renforcé une approche d'ingénierie solide : compréhension fine des systèmes internes d'Unreal, anticipation des dépendances inter-classes et articulation entre ambition technique et faisabilité temporelle.

Le développement total représente environ **77 à 100 heures de travail effectif**, soit une moyenne de **11 à 14 heures par semaine** sur sept semaines, selon l'analyse des commits et l'estimation du temps par itération.

Synthèse personnelle : la réalisation intégrale du projet, menée sans collaboration directe, a renforcé mon autonomie méthodologique et ma confiance technique. Les compétences acquises autour du multijoueur local ouvrent également la voie à de futurs travaux sur la **mise en réseau**, notamment dans le cadre d'un prototype FPS expérimental.
Maxime directrice : *Keep it simple, stupid* — principe dont la rigueur d'application s'avère essentielle dans toute production interactive à contrainte forte.