UQAC
Université du Québec
à Chicoutimi

Stickman Brawl — Post-mortem

Présenté à
Sylvain Boivin

Par
Antonin Claudel

8INF978 — Sujet spécial en jeux vidéo
1537 — Maîtrise en informatique (jeux vidéo)

Automne 2025

Plan

1. Informations générales

1.1 Détails du projet

Titre	*Stickman Brawl*
Auteur	Antonin Claudel
Période	Du **8 septembre** au **5 novembre 2025**
Nature	Jeu de combat local (2 joueurs) de type *Smash-like*
Moteur	Unreal Engine 5.6
Technologie	Intégralement **Blueprints**

1.2 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 la version 2.1 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

2.1 Objectif principal

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**.

2.2 Axes d'apprentissage émergents

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

3.1 Architecture et stabilité

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 du combat rapproché — déplacements latéraux, `MultiSphereTraceforObjects`, `LaunchCharacter`, `ApplyDamage`, gestion de la santé et des stocks, détection du vainqueur — ont fonctionné de manière stable dès leur première intégration, évitant le 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. Ces fonctionnalités, rapidement validées, s'inscrivent pleinement dans les objectifs d'apprentissage initiaux liés au combat rapproché.

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.

3.2 Choix techniques structurants

Les choix techniques structurants incluaient :

- 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.

3.3 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.

3.4 Sélection de la couleur du personnage

La sélection de la couleur du personnage a constitué la tâche la plus exigeante du projet, mobilisant environ vingt jours de travail. Sa conception a reposé sur la combinaison de **Data Assets**, de **Widgets imbriqués** et d'une gestion fine du **focus multi-joueurs**. Ce travail a abouti à un système robuste et réutilisable, offrant une base solide pour une future extension vers des sélections plus dynamiques.

3.5 Multijoueur local

La mise en œuvre du **multijoueur local** représente une réussite majeure du projet, atteignant pleinement l'un des deux objectifs d'apprentissage fixés au départ. Dès l'écran de sélection de la couleur du personnage, le moteur crée un deuxième joueur via la fonction `Create Local Player`. Les joueurs sont référencés dans le tableau des **Local Players** de la **Game Instance**. Ils sont ensuite instanciés dans tous les niveaux, y compris *Brawl*, à partir de ces données, garantissant une initialisation cohérente et sans duplication.

Le système d'Unreal gère automatiquement l'assignation des manettes en fonction de la position des joueurs dans ce tableau, ce qui a permis d'obtenir un comportement plug-and-play immédiat sans code supplémentaire. La seule configuration nécessaire a consisté à **désactiver le split-screen** dans les paramètres du moteur.

Ce résultat, obtenu exclusivement en **Blueprints**, démontre une compréhension approfondie de la hiérarchie interne *Player Controller → Local Player → Game Instance → Game Mode* et la capacité à tirer parti des fonctionnalités natives du moteur plutôt que de les contourner.

4. Problèmes rencontrés et résolutions

4.1 Initialisation et synchronisation

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.

4.2 Importation et normalisation des assets

L'importation d'assets externes a présenté plusieurs difficultés techniques. Les modèles utilisés généraient des incohérences d'échelle et de pivot au moment de l'intégration dans Unreal. Ces écarts provoquaient des anomalies de position lors des animations.

Pour résoudre ce problème, un **travail de normalisation des Offsets** a été entrepris. L'objectif était de garantir que tous les modèles importés respectent les mêmes repères spatiaux.

De plus, le téléchargement d'animations *Without Skin* depuis Mixamo, mais générées à partir du modèle par défaut de la plateforme, entraînait un problème spécifique : lors de l'import dans Unreal, en sélectionnant le squelette du Stickman comme squelette cible, le personnage devenait presque invisible. Pour corriger ce phénomène, il fallait réimporter le modèle du Stickman dans Mixamo, le rigger à nouveau, puis télécharger l'animation appliquée à ce modèle avec le paramètre *Without Skin* activé, garantissant ainsi la compatibilité complète entre le squelette et la géométrie.

4.3 Abandon du prototype Zero-G

Le prototype initial *Zero-G* s'appuyait sur une simulation physique complète avec ragdoll, mais il s'est révélé peu viable en raison de contraintes techniques majeures. L'usage du mode `Simulate Physics`,

nécessaire pour activer le ragdoll, rendait impossible l'emploi de fonctions essentielles telles que `AddMovementInput`, incompatibles avec cette configuration. Il fallait alors recourir à `AddForce` appliqué directement sur le *Skeletal Mesh*, en limitant manuellement la force ajoutée lorsque la vélocité devenait trop élevée. Cette approche, bien qu'expérimentale, impliquait de réimplémenter une logique déjà prise en charge par le moteur, rendant la solution trop artisanale et difficilement extensible. Le pivot vers le *Smash-like* a ainsi permis de recentrer l'apprentissage sur le gameplay et la logique moteur standard.

4.4 Incident de configuration de la Game Instance

Lors d'une refactorisation, le renommage du fichier **Game Instance** a entraîné une réinitialisation automatique du paramètre *Game Instance Class* dans les paramètres du projet Unreal. Le moteur ne trouvait donc plus la classe à exécuter au lancement, provoquant un démarrage incomplet du jeu et l'absence de certaines données initialisées à partir de la Game Instance.

Après diagnostic, la cause a été identifiée dans les **Project Settings** → **Maps & Modes**, où la référence de classe avait été perdue. La correction a consisté à **réassigner la Game Instance Class appropriée** dans le projet. Cette expérience a souligné l'importance d'un contrôle de cohérence post-refactor et d'une vérification systématique des références globales du moteur après tout renommage de fichiers Blueprint.

5. Compétences acquises

5.1 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.

5.2 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 :

6.1 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*.

6.2 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.

6.3 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. Ce que je referais différemment

7.1 Approche modulaire

Si c'était à refaire, j'adopterais plus tôt une **approche modulaire** dans la conception des interfaces, notamment en privilégiant l'imbrication de *Widgets* plutôt que leur duplication. Cette méthode aurait simplifié la maintenance du **HUD** lors de l'introduction des couleurs et facilité l'évolution du projet à moyen terme.

7.2 Fonctions utilitaires réutilisables

Je mettrais également en place dès le départ des **fonctions utilitaires réutilisables**, afin d'éviter la redondance de code et d'améliorer la clarté de la logique globale. Cette pratique aurait permis de mieux isoler les comportements communs et d'accélérer les itérations.

7.3 Meilleure gestion des événements et de l'UI

Par ailleurs, je réfléchirais dès la conception à des **alternatives aux assignations (binds)** des éléments de l'interface. L'utilisation d'une **initialisation contrôlée** et de **répartiteurs d'événements (Event Dispatchers)** aurait réduit les appels au *Tick* et amélioré la performance générale du projet dès le début.

7.4 Intégration précoce de CommonUI

Enfin, je testerais dès la conception des interfaces l'intégration du framework **CommonUI** afin d'évaluer sa pertinence pour la gestion des menus et des interfaces dynamiques via la manette. Ce choix, s'il

s'avérait adapté, permettrait de renforcer la cohérence structurelle et la réutilisabilité de l'interface utilisateur.

8. Bilan

8.1 Résultats et enseignements

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.

8.2 Analyse de l'investissement

Le développement total représente environ **77 à 100 heures de travail effectif**, soit une moyenne de **11 à 14 heures par semaine** sur 8 semaines, selon l'analyse des commits et l'estimation du temps par itération.

8.3 Réflexion personnelle

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 : *Keep it simple, stupid* — principe dont la rigueur d'application s'avère essentielle dans toute production interactive à contrainte forte.