le cnam
enjmin


# Rapport de stage

## Développeur jeu vidéo

*du 04 juillet au 26 août 2016*

*François Rivoire*

Entreprise d'acceuil : ***Bulwark Studios***

Formation : ***Cnam-Enjmin***

*Master Jeux et Médias Interactifs Numériques*

*Spécialité : Programmation*

# Table des matières

# Introduction

Du 04 juillet 2016 au 26 août 2016, j'ai effectué un stage au sein de BulwarkStudios (Angoulême). J'ai pu participer principalement à deux missions différentes que nous avons menées en parallèle.

Pour commencer, le studio m'a mis à contribution sur une prestation avec pour but la réalisation d'un serious game à destination des enseignants. J'ai mis en place des outils pour faciliter l'intégration des nombreux assets et la création des multiples scénarios du jeu. J'ai aussi participé au processus d'intégration lui même.

De plus, l'équipe m'a confié la tâche délicate de participer à la réalisation d'un prototype pour un projet qui leur tenait à cœur depuis longtemps. Au delà d'un simple prototype, il était nécessaire que le résultat soit réutilisable et constitue un socle solide pour la suite de la réalisation qui continuerait évidemment après la fin de la durée du stage.

Ce stage constitue ma première expérience professionnelle dans le secteur du jeu vidéo. J'ai en conséquence pu appréhender le fonctionnement réel d'une société dans ce secteur d'activité.

Plus largement, ce stage a été l'opportunité pour moi d'apprendre de nouvelles compétences techniques, de nouvelles méthodologies de travail, de développer ma rigueur et de constater une nouvelle fois l'importance de l'aspect collaboratif pour la réalisation d'un projet.

Au-delà d'enrichir mes connaissances personnelles, ce stage m'a permis de comprendre dans quelle mesure un studio indépendant diffère d'une structure plus imposante. J'ai pu en l'occurrence apprécier une atmosphère conviviale, tout en constatant les multiples défis à relever en tant qu'indépendant. L'envergure de la société a été importante dans ma décision d'y effectuer un stage. En effet, cela m'a semblé être une bonne occasion de faire l'expérience de ce type de structure pour mieux orienter mes futurs choix professionnels. Travailler dans le secteur du jeu vidéo peut en effet se traduire de beaucoup de façons différentes. Il me paraissait donc important de diversifier mes expérience dans ce secteur, afin d'affiner ma vision des possibilités à ma disposition.

# La société : Bulwark Studios

Bulwark Studios a été crée en novembre 2012 à Angoulême par deux anciens développeurs Web et mobile désireux de concevoir des jeux vidéos mobiles, un marché alors en pleine expansion.

Après deux ans et demi d'activité, la société a produit trois titres parus sur **PC, tablettes et smartphones** tout en proposant ses services en développement d'application mobile dans le cadre de **prestations** ponctuelles.

D'un point de vue technique, le studio développe aujourd'hui la majorité de ses productions vidéo-ludiques via **Unity 3D**. Dans le cadre de prestations, il intervient également sur des développements d'applications web et mobiles.

Enfin, le studio a pu mettre en place des partenariats de **publishing** réussis avec les éditeurs canadiens **Noodlecake Studios** (Android) et allemands **Kalypso Media** (PC et tablettes).

| Emmanuel | Jérémy | Tatiana | Damien | Luc |
|---|---|---|---|---|
| Chef de projet GD | Directeur technique | Directrice artistique | Communication | Directeur audio |

# L’équipe

## *En studio*

### *1) Emmanuel Monnereau*

Emmanuel a plus de 5 ans d’expérience dans le développement d’application web et mobile. Il a été en charge de projets importants comme les plateformes de presse régionale en ligne du groupe SudOuest. Dernièrement, il a intégré le développement de la plateforme d’actualité Web du groupe France Télévisions (francetv.fr/info).

### *2) Jérémy Guéry*

Jérémy a également 5 ans d’expérience dans le développement de site internet dynamique. Depuis 3 ans, il s’est formé puis spécialisé dans la création d’application mobile. Il a commencé par la réalisation d'une série de 8 applications à destination de personnes atteintes d'autisme. Il a par la suite conçu une application de réalité augmentée touristique.

### *3) Tatiana Barbesolle*

Diplômée de l’EMCA, Tatiana a d’abord travaillé dans différents studios d’animations (Les Trois Ours, 2 Minutes, Normaal, les Films du Poisson Rouge, Antefilm) avant de rejoindre Bulwark Studios en tant que directrice artistique. Elle y a rapidement apporté une patte graphique reconnaissable dans l’ensemble des productions du studio.

### *4) Romain Viel*

Dernier arrivé au studio, Romain a terminé ses études à Supinfogame Rubika avant de rejoindre l’équipe. Il a effectué des stages au sein de studios situés en Inde et aux États-Unis ce qui lui confère une culture élargie de l’art au sein du jeu vidéo.

## *En télétravail*

### *1) Damien Morvan*

Après avoir suivi une formation technique en développement web, Damien s’est orienté vers la gestion de projet internet. Il a été en charge d’un projet collaboratif international en ligne chez BNP Paribas. Intéressé par le secteur culturel, il a également travaillé un an pour un centre culturel où il a interconnecté plusieurs systèmes (site internet, billetterie, CRM et réseaux sociaux).

### *2) Luc Blanchard*

Luc est l’oreille du studio. Après avoir passé plusieurs années à faire du sound design pour des courts métrages et des films, il s’est spécialisé dans les applications mobiles et les jeux vidéo. Il apportera sa connaissance technique sur toutes les parties audio de l’application. Il travaille aujourd’hui depuis Montréal au Canada.

## Projets

### *Jeux*

● **Ronin** (*novembre 2012*, **iOS** , jeu de course infinie)

**URL :** https://itunes.apple.com/fr/app/ronin/id583710912?mt=8

*Ronin est un runner publié sur iPhone et iPad en décembre 2012. Le joueur y incarne un samouraï errant à la poursuite d'une mystérieuse voleuse lui ayant dérobé sa dernière paie. Via des gestures intuitives, le joueur doit éviter certains obstacles et éliminer les menaces se dressant sur sa route.*

● **Spin Safari** (*juin 2013*, **iOS / Android** , puzzle game type match3)


**URL :** https://itunes.apple.com/fr/app/spinsafari/id671742267?mt=8

*Spin Safari est un puzzlegame sorti simultanément sur iOS et Android en juin 2013. Version révisée du match3, le joueur doit regrouper par 3 des animaux mélangés sur un disque. Le jeu défie le joueur sur plus d'une centaine de niveaux répartis en plusieurs chapitres reprenant différents écosystèmes du monde.*

● **Crowntakers** (*novembre 2014*, **Steam / iOS / Android** , jeu de rôle tactique)


**URL :** http://store.steampowered.com/app/294370/?snr=1_7_15__13

*Crowntakers associe la stratégie au tour par tour à des éléments typiques du jeu de rôle et place le joueur aux ordres de la couronne, dans un monde médiéval fantastique, riche en rencontres périlleuses et aventures épiques. Grâce à un monde généré aléatoirement, Crowntakers offre une expérience de jeu sans cesse renouvelée.*

## *Prestations*

● **Wasis**

Bulwark Studios est intervenu pour développer l'interface compatible navigateur web PC et tablettes du prototype de l'application sur l'eau Wasis.

● **L'Hermione**

Bulwark Studios fut sollicité pour la réalisation du **visuel promotionnel** du lancement de l'application smartphone et tablette de l'Hermione.

● **Autres**

Nous intervenons occasionnellement en régie technique au sein de sociétés nécessitant un appui lors de montée en charge de la production. Nous avons également pour objectif de partager nos savoirs et expériences, et dans cette optique nous sommes intervenus à plusieurs reprises lors de conférences à l'UUGA.

# Les travaux effectués et les apports du stage

## Description détaillée des missions

Comme dit en introduction, deux missions nous ont été données, qu'il a fallut mener à bien en simultané.

D'une part, nous devions participer à une prestation sur un serious game à destination des enseignants. Le but de l'expérience est d'inciter les enseignants à mieux gérer leur voix au cours de leur journée. Il consiste en une suite de mises en situations scriptées. Plusieurs choix sont proposés au joueur, et il peut ensuite observer les conséquences se dérouler.

Ce projet était géré par plusieurs équipes en simultané, avec un studio s'occupant du game design et de la communication avec le client, un autre prenant en charge la conception visuelle, et enfin BulwarkStudios pour la partie programmation et l'intégration. Il était donc intéressant pour moi de constater les contraintes que cela entraînait en termes de communications entre les différentes participants.

J'ai aussi pu me rendre compte qu'un studio indépendant comptait beaucoup sur les prestations de ce type pour maintenir un bon équilibre financier. Aussi évident que cela puisse paraître, le fait d'en faire l'expérience est souvent plus marquant. C'est une donnée qu'il est important de garder à l'esprit pour les choix professionnels qu'il me faudra faire par la suite.

En parallèle de cette mission, il nous a été demandé de poser les fondations pour un projet interne au studio, Space Ark. Le projet se dirige vers un jeu de stratégie/gestion/survie d'une petite équipe de survivants sur une planète inconnue.

Ce projet étant encore à ses débuts en termes de game design, nous devions fournir un socle qui permette de tester différentes idées de gameplay tout en restant ouvert à l'ajout de fonctionnalités. En somme, le défi était de produire une architecture de code facilement maintenable, relativement découplé tout en restant efficace. Les prestations menées par le studio fournissant un soutien financier, il était possible de prendre le temps pour poser des bases solides et réfléchies. C'est pourquoi cette mission fût une très bonne opportunité pour moi de m'entraîner à architecturer mon code et à connaître et bien utiliser les structures de données et les designs patterns adaptés selon la problématique rencontrée.

Pour ces deux missions, je m'emploierais à détailler ci-après les choix que nous avons menés ensemble tout au long de ces deux mois de stage.

## Partie technique

Je m'efforcerais ici de rapporter mes réflexions sur chaque solution technique en décrivant brièvement la problématique, la réponse choisie et en nuançant ce choix.

### *Command*

Le pattern Command à été au centre du design très tôt dans l'élaboration du projet. Il était en fait déjà présent avant le début du stage. Ce pattern permet d'encapsuler un ou plusieurs comportements (donc des appels de méthodes) dans un objet.

Cette façon de procéder présente de nombreux avantages. De manière générale on obtient une plus grande flexibilité quand a l'exécution des comportements voulus. En effet, on peut maintenant stocker les Commandes, ce qui permet par exemple d'instaurer des priorités. Il devient possible de faire transiter ces Commandes facilement, ce qui est particulièrement utile dans un jeu de stratégie où il est nécessaire de gérer efficacement de multiples unités. Un autre avantage bien connu est de rendre bien plus simple l'implémentation de mécaniques telles que l'Undo/Redo, ce qui encore une fois est souvent nécessaire dans un jeu de stratégie.

Nous avons pu observer l'utilité de ce design pattern quand il a fallut créer un système de prérequis afin de vérifier si une unité était en capacité d'effectuer une Commande qui lui avait été donnée. Pour exemple, si une unité recevait pour commande d'attaquer une autre unité, il lui était demandé de vérifier si elle était à portée de cette unité avant de lui infliger des dégâts. Si cette condition n'était pas remplie, il nous suffisait de créer une Commande de déplacement vers la cible et de mettre une nouvelle Commande d'attaque à la suite de la Commande de déplacement.

### *Observer*

Je connaissais déjà le design pattern Observer avant ce stage, mais nous avons pu en faire une grande utilisation dans ce projet. Ce pattern est souvent utilisé pour permettre à différents objets de communiquer sans les coupler. Il donne la possibilité à un objet de notifier d'autres objets qu'un événement a eu lieu sans pour autant avoir à lier concrètement les différentes classes impliquées.

Comme souvent, nos avons mis ce pattern en place afin de créer un système d'interface utilisateur qui puisse réagir aux changements intervenant au niveau de la logique du jeu. La logique et sa représentation devaient rester indépendantes autant que possible, c'est pourquoi l'utilisation de ce design pattern se justifiait pleinement. Nous avons donc créé un ensemble de classes et interfaces sur le schéma Observer, avec par exemple wrapper générique qui notifiait automatiquement ses observers en cas de modification d'une variable dont il avait la responsabilité. Cela a rendu plus simple la création de variables pouvant être reconnues et surveillées par l'interface.

Bien sûr un tel pattern comporte ses défauts. Tout d'abord, il est moins facile de lire le code et de comprendre le déroulement du programme. Du plus, il faut penser à se désinscrire des objets observés sous peine de constater des fuites de mémoires ou des comportements imprévus. Pour expliciter le contrat qui demande aux observers de se désinscrire proprement, nous avons choisi d'encapsuler ce processus dans une classe dédiée. Cet objet est récupéré par l'observer au moment de son inscription, afin de l'inciter à bien effectuer la désinscription tout en permettant de conserver un minimum de référence à l'objet observé une fois l'inscription faite.

## *Generics*

En début de développement, il est difficile d'estimer avec précision tous les cas qu'il faudra couvrir dans le suite du développement. Cette situation nous a conduit assez rapidement vers l'utilisation de types génériques. Les types générique permettent de mettre en place des algorithmes manipulant des types non précisés. En effet, certains algorithmes restent valides pour de nombreux types de paramètres différents, utiliser les types génériques est alors tout indiqué.

Dans notre cas, les types génériques étaient utilisés en conjonction avec le pattern Observer pour gérer n'importe quel type de message. L'algorithmie derrière le pattern Observer est la même pour tout type de message envoyé, c'est pourquoi il n'est pas nécessaire de restreindre notre implémentation de ce design pattern à un type de message particulier. Cela nous a permis de gérer avec un code similaire plusieurs types de valeurs à communiquer au code gérant l'interface utilisateur.

Bien qu'au fait de l'existence des types génériques, je n'avais jamais vraiment eu l'occasion de les utiliser régulièrement en situation réelle. C'est une technique relativement importante à connaître et qui peut d'avérer très utile si elle est bien utilisée, en conjonction avec certains design patterns.

## *Input et notion de contexte*

Nous avons rencontré une problématique classique en programmation pour le jeu. Cette problématique a trait à l'association entre un input brut (eg : touche du clavier enfoncée) et une action dans le jeu, autrement dit, l'input mapping. En effet, il n'est pas rare dans un jeu de rencontrer plusieurs situations qui correspondent à un input mapping spécifique, différent de celui des autres situations du jeu. Par exemple dans le cas d'un joueur qui rentre dans un véhicule, l'appui d'une touche n'aura bien souvent pas le même effet que lorsqu'il se déplace à pied.

Pour palier à ce problème nous avons créé un système permettant de gérer la transformation d'un input brut en une action au niveau logique en prenant en compte les différents contextes du jeu. Chaque contexte correspond à un input mapping particulier, et les différents contextes interagissent selon un schéma de type Chain of command : ils associent les inputs bruts pour lesquels ils définissent un mapping, et laissent aux contextes moins prioritaires la possibilité

mapper les inputs bruts restants. Il suffit alors d'activer/désactiver un contexte de la liste pour que le jeu réagisse différemment aux inputs effectués par le joueur. On transmet le résultat du processus de mapping au reste du code par le bias du design pattern Observer, ce qui rend le système relativement flexible.

Il faudra sans doute par la suite raffiner le mécanisme d'interaction entre les contextes pour pouvoir gérer des situations plus complexes.

## *Documentation*

Étant en stage pour une durée limitée, il était plus que nécessaire de documenter l'intégralité du code produit pour qu'il puisse être compris et remanié par la suite. Tout le code produit a été systématiquement et rigoureusement documenté dans ce but. J'ai pu a nouveau constater que cela améliorait le code lui même, puisque en se forçant à rédiger une description claire on pouvait plus facilement trouver des failles. Cela conduit aussi à prendre plus soin de l'interface du code, à le penser pour qu'il soit utilisable le plus intuitivement et correctement possible. Ce stage m'aura finalement aidé à renforcer cette bonne pratique.

## *Unity Editor*

Dans un domaine plus pragmatique, j'ai eu l'occasion de développer mes connaissances des possibilités d'extension du moteur Unity3D au niveau de l'éditeur. Le besoin de se pencher sur ce sujet est venu de la prestation, mission consistant à réaliser un serious game. Cette mission comportait une importante partie intégration et level design (pour ainsi dire). Beaucoup de tâches se sont avérées répétitives et longues. Nous avons donc réfléchi à la mise en place d'un certains nombres d'outils qui, bien que plutôt simples, nous ont fait gagner un temps très précieux.

Le moteur Unity3D est extrêmement permissif en ce qui concerne son éditeur. Il est par exemple très facile de rajouter un bouton à l'interface, ou de mettre en place des formulaires pour automatiser des tâches qui sans cela s’avéreraient fastidieuses. Si nous nous sommes limités à ce genre d'extensions, il est aussi possible d'aller bien plus loin et de proposer de véritables outils qui s'intègrent très bien avec l'éditeur existant.

Au delà de ce que j'ai pu apprendre de spécifique au moteur Unity3D, j'ai surtout constaté à quel point la programmation outil en général peut s'avérer cruciale dans le processus de développement. Fluidifier le travail de chacun permet non seulement de gagner du temps, mais peut aussi rendre la tache beaucoup plus agréable. L'ergonomie de ces outils est aussi très importante. Pour exemple, beaucoup des outils que nous avons réalisé permettent en définitive de moins faire d'allers-retours à la souris. En plus de constater son importance, j'ai aussi réalisé que c'était en soi une tâche agréable. Voir le travail de mes collaborateurs facilité par un outil que j'ai mis en place est très gratifiant. Cela renforce aussi la connivence au sein de l'équipe, et pousse ainsi l'aspect collaboratif de ce métier, un aspect qui me tient particulièrement à cœur.

# Conclusion

A travers ce rapport j'espère avoir pu rendre compte de mon expérience en tant que développeur pendant ces deux mois au sein de la société Bulwark Studios. S'il me reste encore beaucoup à découvrir pour appréhender ce métier dans son entier, ce stage m'aura permis d'y faire un premier pas très encourageant. Que ce soit au niveau technique ou au niveau du vécu au quotidien, je suis maintenant convaincu de pouvoir m'y plaire. J'ai pu nuancer l'attrait des petites sociétés indépendante en prenant en compte les difficultés qui accompagnent ce genre de situation, tout en appréciant grandement l'ambiance que j'y pu y rencontrer chaque jour.

# Remerciements

J'aimerais remercier tout particulièrement Emmanuel, Tatiana et Jeremy pour leur bienveillance, leur entrain et leur écoute. Travailler avec eux a été un réel plaisir, et j'espère beaucoup pouvoir retrouver ce type d'ambiance dans la suite de ma carrière professionnelle.

Je remercie également Titouan, stagiaire en programmation également, avec qui j'ai pu partager de très bon moment de travail et de détente, sans oublier de nombreux débats sur le nommage de nos merveilleuse classes.

Enfin, merci à Nyx Studios pour m'avoir permis tester le HTC Vive (c'était vraiment bien).

# Annexes

```csharp
/// <summary>
/// This generic interface represents a Subject/Observable/Provider in an Observer pattern.
/// Any class implementing this interface is supposed to allow registering callbacks functions.
/// They should also implement a private class implementing the IDisposable interface that will unregister a particular callback when its Dispose method is called.
/// That way an instance of this class can be returned when registering to an ISubject, and it can be used to unregister to the ISubject.
/// </summary>
/// <remarks>The Subject class below is a good exemple of implementation of ISubject, using C# events.</remarks>
/// <typeparam name="T">The type of the message that will be sent to registered callbacks.</typeparam>
public interface ISubject<T> : ISubject {

    /// <summary>
    /// Use this method to add a callback function to the Subject.
    /// </summary>
    /// <remarks>
    ///  !!! Don't forget to unregister your callback when you're done or when the object owning it is destroyed
    /// ...or prepare for unforseen consequences.
    /// Use the Dispose method of the returned IDisposable object to unregister.
    /// </remarks>
    /// <param name="listener">The callback function to register.</param>
    /// <param name="unsubscriber">The IDisposable to use to unregister the callback. Use the Dispose method to do that.</param>
    void AddListener(Action<T> listener, out IDisposable unsubscriber);

}
```

*Observer : interface Subject*

```csharp
/// <summary>
/// This abstract generic class provides a simple implementation of the ISubject generic interface.
/// It uses a C# event to register/unregister callbacks.
/// Use the Notify method to fire the event and notify the observers.
/// </summary>
/// <typeparam name="T">The type of the message that wil be sent to registered callbacks.</typeparam>
public abstract class Subject<T> : ISubject<T> {

    /// <summary>
    /// The event acting as a observable.
    /// </summary>
    protected event Action<T> OnNotify;

    /// <summary> Use this method to add a callback function to the Subject.
    public virtual void AddListener(Action<T> listener, out IDisposable unsubscriber) ...

    /// <summary> Use this method to remove a callback function to the Subject.
    protected virtual void RemoveListener(Action<T> listener) ...

    protected bool HasListeners() ...

    /// <summary> Use this method to notify the registered observers. This will trigger every registered callback.
    protected void Notify(T message) ...

    /// <summary> Get the type of message that will be sent to observers upon notification.
    public Type GetMessageType() ...
```

*Observer : exemple d'implémentation de l'interface ISubject*

```
    /// <summary> This class is reponsible for unsubscribing a callback of a Subject instance. It will do so when its Dispos ...
    private class Unsubscriber : IDisposable {

        /// <summary> The subject from whitch the callback will be unregistered.
        private Subject<T> subject;

        /// <summary> The callback to unregister.
        private Action<T> callback;

        /// <summary> Ctor.
        public Unsubscriber(Subject<T> subject, Action<T> callback) {
            this.subject = subject;
            this.callback = callback;
        }

        /// <summary> Call this method to unregister the corresponding callback of the subject it registered to.
        public void Dispose() {
            if (callback != null) subject.RemoveListener(callback);
        }
    }

}
```

*Observer : On utilise une classe de ce type pour gérer la désinscription*

```
/// <summary>
/// This generic class is designed to encapsulate a value along with an event that fires when this value is modified.
/// You can stop the class from firing the event upon value modification with the bNotify boolean (true by default).
/// </summary>
/// <typeparam name="T">The type of the encapsulated value.</typeparam>
public class ModificationNotifier<T> : Subject<T> {

    /// <summary> Whether we should notify the listeners when the value is modified.
    public bool bNotify;

    /// <summary> The encapsulated value.
    private T _value;
    public T Value {
        get {
            return _value;
        }
        set {
            _value = value;
            if (bNotify) Notify(_value);
        }
    }

    /// <summary> Ctor.
    public ModificationNotifier(T value, bool notify = true) {
        _value = value;
        bNotify = notify;
    }

    /// <summary> Use this method to add a callback function to the ModificationNotifier. This will also fire the callback u ...
    public override void AddListener(Action<T> listener, out IDisposable unsubscriber) {
        base.AddListener(listener, out unsubscriber);
        listener(_value);
    }
}
```

*Observer : exemple d'utilisation en conjonction des properties C#*