Faire un jeu vidéo : par où commencer ?

Par Alexandre Laurent

Date de publication : 8 octobre 2012

Dernière mise à jour : 18 août 2014

Vous avez une idée de jeu ou vous débutez en programmation et souhaitez faire un jeu, alors cet article est un excellent point de départ pour vous. Ce document regroupe les informations utiles permettant de prendre une décision afin que le démarrage de votre nouveau projet soit le meilleur possible.

Commentez

I - Introduction

Les jeux vidéo ont toujours été un acteur important dans le monde de l'informatique. Peu de temps après la conception des ordinateurs, des programmes de jeux ont vu le jour. Maintenant, le marché de ce loisir dépasse celui du grand écran.

Tout en jouant, vous avez sûrement eu l'envie de faire votre propre jeu. Ce tutoriel a pour but de poser la première pierre à votre projet afin que celui-ci devienne une réalité.

Ce tutoriel n'est pas écrit que pour les débutants. C'est un guide pour commencer n'importe quel projet de jeu vidéo et rappellera les notions et outils de base permettant de mener votre projet à bien.

I-A - Qu'est-ce qu'un jeu vidéo ?

Avant de commencer, revenons au tout début. Qu'est-ce qu'un jeu vidéo ? C'est en sachant cela qu'il est plus facile d'imaginer les différents éléments nécessaires à sa création.

Tout d'abord, un jeu vidéo n'est pas que l'œuvre d'un seul programmeur (sauf cas très rare). Un jeu vidéo, c'est le rassemblement d'une idée, d'un programme, de dessins et de sons. Ce tout constitue un jeu vidéo. Ces domaines sont eux aussi fragmentés en sous-ensembles :

Programmeur :

- intelligence artificielle - gestion des ennemis et des éléments « neutres » ;
- moteur - bloc qui permet l'orchestration des éléments du jeu ;
- 2D/3D - module qui affiche les images à l'écran ;
- jeu - les règles du jeu, la gestion de la santé, de l'apparition et l'orchestration des éléments et autres ;
- son - module qui gère les sons, la musique selon les événements du jeu ;
- réseau - module permettant de faire des jeux multijoueurs ;
- interface utilisateur - les menus, l'écran de pause, le HUD ;
- outils - l'éditeur de carte et les autres outils nécessaires pour la création du jeu.

Game designer :

- scénariste - créateur de l'histoire, du cadre scénaristique du jeu ;
- gameplay designer - création des règles du jeu (comment gagner/perdre…) ;
- level designer - création des niveaux.

Graphiste :

- artiste 2D ;
- modeleur de personnages ;
- modeleur de décors.

Son :

- effets sonores ;
- musiciens.

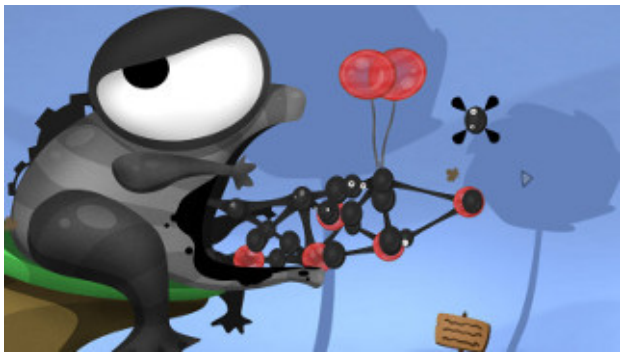
Cette liste est loin d'être complète. Par exemple, il peut aussi y avoir des employés à la communication (site web, communauté) ou, pour les projets internationaux, des traducteurs.

Cette liste est vraie pour les très grands projets. Pour de simples projets amateurs, on peut se retrouver à un programmeur et un artiste et cela peut suffire.

Ce qu'il faut comprendre ici, c'est qu'un jeu est avant tout l'histoire d'un(e) ou de plusieurs passionné(e)s. Cela demande énormément de motivation et de temps. Un jeu tel que God of War 3 est réalisé par une équipe complète de professionnels qui ont passé des journées, des semaines, des mois et souvent même des années entières à la conception du jeu.

God of War 3

Mais ce n'est pas pour cela que vous n'allez pas pouvoir faire un jeu. En effet, il est possible de faire des jeux simples qui sont très intéressants. Prenons des exemples comme Minecraft, Binding of Isaac ou World of Goo. Ces jeux ont été réalisés par des passionnés pendant leur temps libre (même s'il est important de rappeler que Notch a décidé d'arrêter son travail pour développer à plein temps Minecraft).



World of Goo

C'est la raison de cet article. Je vais vous guider afin que vous ne tombiez pas dans les pièges classiques d'un projet amateur et que vous prépariez au mieux le développement du jeu avec les outils adéquats pour réaliser vos rêves.

I-B - Faire un jeu vidéo ?

Mais alors, est-ce possible de faire un jeu vidéo tout seul ?

La réponse est oui. Il suffit d'avoir de la motivation, de la patience, du temps ainsi qu'une bonne base au départ (cet article par exemple).

En fait, le principal problème lorsque l'on veut faire un jeu, c'est que l'on se plonge immédiatement sur le projet sans réfléchir. La première étape est de calibrer ses idées à ses possibilités. Ensuite, nous déterminerons comment réaliser ce projet.

I-B-1 - Calibrer son projet

En effet, de trop nombreuses fois, j'ai pu lire des messages de personnes cherchant à réaliser un MMORPG du style de World of Warcraft. Certes, c'est un rêve de faire un tel projet, mais, soyons honnête, ce projet est immense et bien trop décourageant pour un petit groupe de personnes.

Pour réussir son projet, il faut le ramener à ses possibilités. Le but est que vous appreniez sur la conception d'un jeu en réalisant des petits jeux (ou brouillons). Ainsi, vous aurez un aperçu des difficultés que vous pouvez avoir mais également un début de réponse à des questions fondamentales.

Quepensez-vouspossible pour vous ?

Par exemple, si vous commencez la programmation, il est conseillé de commencer par des projets simples du style Tic-tac-toe, le jeu des allumettes ou la bataille navale (ou encore Mastermind). Oui, ce sont des projets simples, faits déjà mille fois. Mais ne commence-t-on pas chaque langage par un « Hello World » ? Pour ces jeux, je parle d'application en console, toute simple. Par la suite, vous pouvez continuer avec des jeux un peu plus complexes, ou simplement améliorer votre Tic-tac-toe pour lui donner une interface graphique.

Tout projet doit commencer au plus simple. Ensuite, on ajoute des briques, toujours plus complexes, toujours plus grandes. À la fin, on obtient un jeu complet et admirable.

Ne commencez donc pas par un MMORPG, cela serait du suicide et vous découragerait trop tôt (parce qu'en plus, vous ne saurez pas comment le commencer). À la place, faites un jeu en commençant par la base de celui-ci, comme un gestionnaire de combat ou de monstres.

I-B-2 - Faire un projet en équipe ?

Selon la taille de votre projet, vous souhaitez ou devez travailler en équipe.

Gérer une équipe demande plus de travail et de préparation sur le projet. En effet, chef de projet est un métier à part. D'autant plus que contrairement à une entreprise, il n'y a pas de motivation d'argent. Chacun aura ses occupations et ses envies, mais aussi son propre emploi du temps rendant la gestion du projet très difficile.

La gestion peut donc être compliquée, mais pas impossible. Premièrement, il faut déterminer l'organisation du travail (qui fait quoi et comment). Ensuite, il faudra continuellement motiver l'équipe et suivre l'avancement des différentes tâches effectuées. Comme c'est votre projet avec votre idée, les autres membres seront moins motivés que vous. Finalement, la coopération et la communication sont des points importants. La distance entre chaque membre ne facilitant pas le travail, il faudra régulièrement faire des réunions à l'aide d'un logiciel tel que Skype. Finalement, pour le développement et pour la gestion du projet, plusieurs outils existent et sont décrits dans la section « **les outils** ».

I-C - Quelle voie choisir ?

De multiples possibilités s'offrent pour créer votre jeu vidéo. On peut distinguer quatre voies :

- le modding ;
- les game makers ;
- les kits de développement (SDK) ;
- la programmation pure et dure.

L'ordre choisi pour cette liste prend en compte la nécessité de programmer. Ainsi, le premier élément demande peu ou pas de programmation alors que le dernier consiste uniquement en de la programmation.

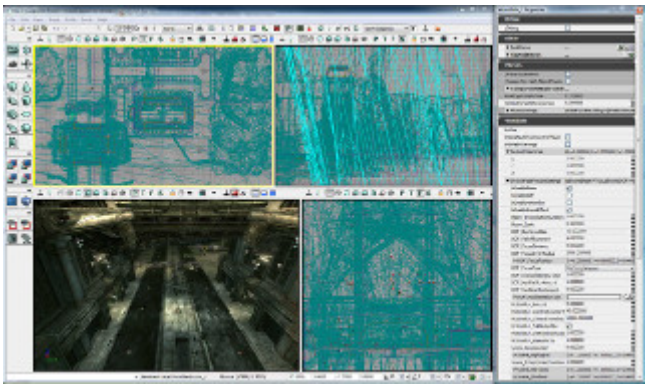
Ainsi, il y en a pour tous les goûts. Un artiste voulant tester ses créations pourra le faire très facilement en l'incorporant dans un autre jeu (modding). Un game designer pourra tester ses idées dans un game maker par exemple. Un programmeur pourra, selon son niveau, soit faire son jeu en utilisant un moteur, soit créer directement le sien.

II - Le modding

Le modding est le processus de modification d'un jeu. Certains développeurs proposent des outils pour créer ou éditer le contenu du jeu afin que la communauté puisse étendre son jeu préféré.

II-A - Éditeur

Des jeux comme Skyrim, Starcraft ou Unreal Tournament 3 proposent un éditeur complet afin de créer de nouvelles cartes et même de nouvelles règles de jeu. Chaque développeur de jeu a son propre éditeur. Ainsi, le meilleur moyen d'avoir de l'aide sur ces outils et de lire la documentation et de parcourir les différents sites des communautés de fans.



Éditeur de Unreal Tournament 3

Cette méthode de créer son jeu ne demande aucune ou très peu de connaissances en programmation et permet de s'amuser tout en restant dans l'univers de son jeu préféré. Un game designer pourra tester ses idées et un artiste pourra intégrer ses nouveaux assets, lorsque cela est possible.

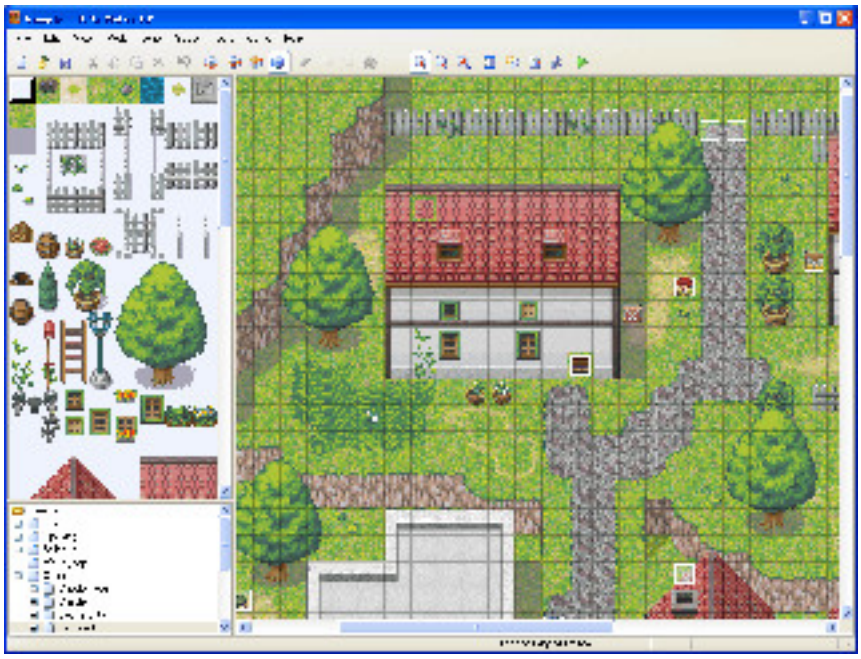
II-B - Modification des fichiers du jeu

Certains jeux ne sont pas accompagnés d'éditeur. Du coup, il faudra modifier directement les fichiers du jeu « à la main ». Vous pouvez néanmoins trouver des logiciels spécialisés sur les sites des fans. Autrement, cela demandera de la recherche, un esprit logique et de la motivation. Toutefois, cela peut être très intéressant, si vous souhaitez savoir comment le jeu fonctionne et si vous aimez tripatouiller.

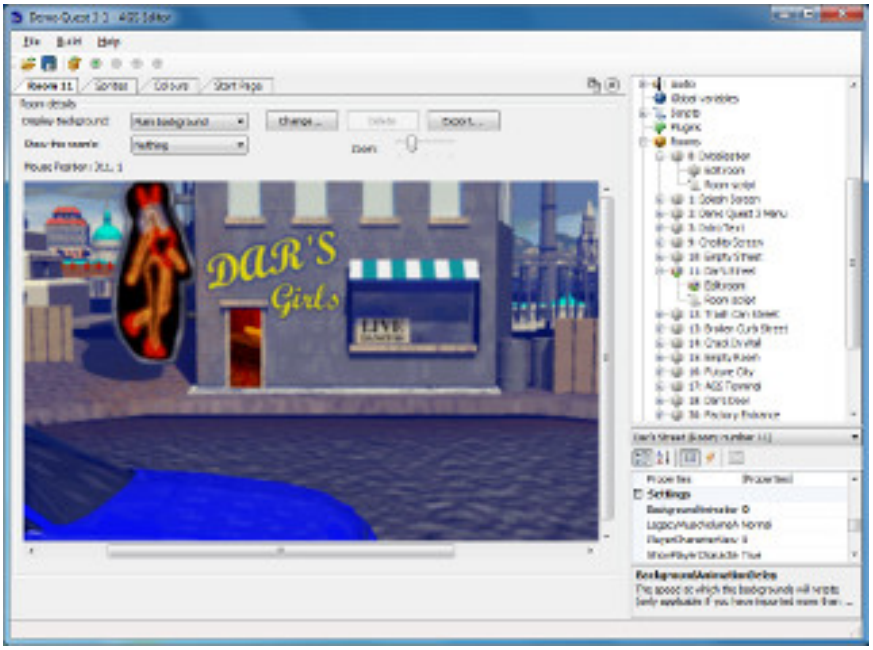
III - Les game makers

Les game makers sont des logiciels aidant à la création d'un jeu d'un type précis mais heureusement, il existe un maker pour chaque type de jeu ou presque. Parmi ceux-ci, nous pouvons citer :

- la gamme **RPG Maker** pour créer ses propres RPG ;
- **MUGEN**, un moteur de jeux de combat ;
- **Adventure Game Studio**, un créateur de jeux d'aventure ;
- **3D Game Studio** ;
- **Game Maker**.



Chacun d'entre eux propose un ensemble d'outils adaptés et spécialisés pour la création d'un jeu vidéo. Pas ou peu de connaissances en programmation sont nécessaires car ce genre de logiciel réimplémente un langage de scripting très simplifié afin de déclencher des actions dynamiques. De plus, la globalité du jeu peut être créée en cliquant sur des boutons et en configurant des propriétés.



Ce type de logiciel permet de mettre en place ses idées et peut donc être une bonne idée pour tester des mécanismes pour les game designers. Toutefois, si vous souhaitez un jeu qui ne peut se fabriquer dans un moule et qui se veut original par son gameplay, ces logiciels ne vous conviendront certainement pas.

IV - Kits de développement (SDK)

Récemment, plusieurs sociétés de jeux vidéo ont distribué les logiciels qu'ils utilisent durant la conception de leurs jeux vidéo. Ainsi Epic Games (Unreal Tournament, Gears of War) et Crytech (FarCry, Crysis) ont mis à disposition du public l'ensemble des logiciels qu'ils utilisent pour créer leurs jeux.

Mais ce ne sont pas les seuls à avoir proposé une telle suite pour la création de jeux vidéo. En effet, la société Unity Technologies ainsi que quelques autres ont aussi créé leur propre solution, offrant une alternative aux grands noms du milieu à un plus faible coût.

Ainsi, les trois kits les plus célèbres sont :

- **Unreal Development Kit** (UDK) par Epic Games ;
- **CryENGINE SDK** par Crytek ;
- **Unity 3D** par Unity Technologies.

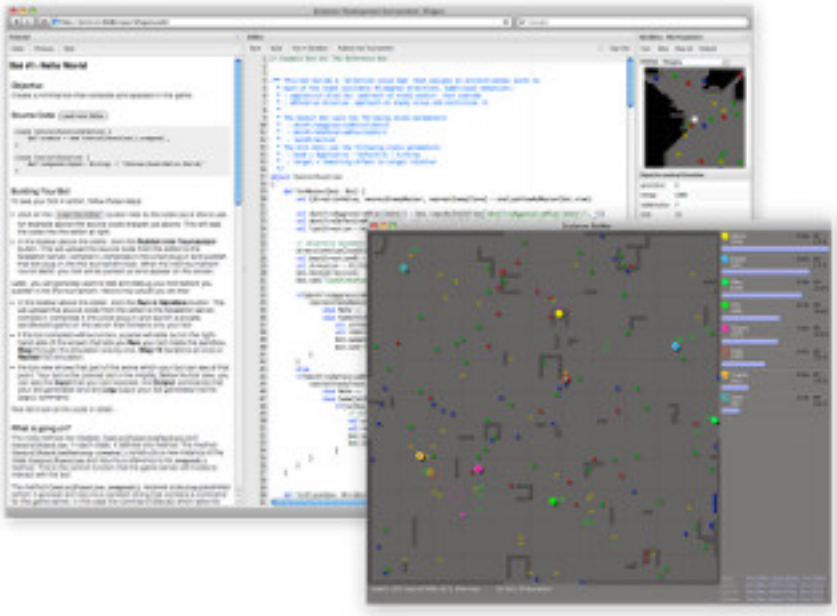
Mais il existe aussi d'autres logiciels dans cette lignée, tels que **NeoAxis**.

Le principal avantage de ce type de solution est la possibilité de créer un jeu rapidement, tout en utilisant les nouvelles techniques et technologies du monde du jeu vidéo.

Toutefois, sachant que l'architecture du jeu est déjà en place, il peut être un peu plus compliqué de faire un jeu dans un style différent de celui pour lequel le kit a été prévu. Cela n'est pas impossible, car les kits laissent une grande liberté, mais pour ce faire il faudra faire de la programmation afin d'implémenter les nouveaux comportements dans le moteur.

Epic Games a créé son propre langage dans l'UDK : l'UnrealScript. Il est avantageux de connaître la programmation avant de partir à la découverte de celui-ci. Unity 3D et NeoAxis se basent sur du C# et le CryENGINE SDK sur du C++.

V - La programmation



Cette section présente les possibilités lorsque l'on souhaite se lancer dans la programmation.

V-A - Le langage

Le premier choix à faire est le langage. En théorie, il est possible de prendre n'importe quel langage pour faire son jeu. En pratique, certains langages seront plus adaptés que d'autres.

Il n'y a pas de langage parfait. Chacun a son point de vue par rapport aux langages, tiré de sa propre expérience. Donc, le premier conseil, c'est de se faire plaisir.

Vous avez commencé un langage et vous l'aimez bien, alors n'hésitez pas à continuer avec celui-ci. En effet, il est déjà assez compliqué de faire son jeu, alors apporter une nouvelle difficulté en choisissant un langage inconnu n'est sûrement pas une bonne idée.

Certaines plateformes ne supportent pas tous les langages. Ainsi, si vous souhaitez faire un jeu pour des plateformes autres que le PC, renseignez-vous des contraintes pour cette plateforme.

Comme première indication, l'**assembleur** est fortement déconseillé. Le langage est trop bas niveau pour pouvoir faire des jeux simplement. Seules les vieilles machines (Game Boy, Amiga, Atari) nécessitent encore l'utilisation de ce langage.

Le **C** et surtout le **C++** sont des très bons choix, car couramment utilisés dans le monde du jeu vidéo. De par ce fait, le langage dispose d'une multitude de bibliothèques permettant de réaliser vos jeux. Malheureusement, ce n'est pas le langage le plus simple (même si ce point de vue est relatif) et celui-ci demande une grande rigueur.

Le **Java** est un langage de choix, car couramment utilisé tout en étant simple. De plus, ce langage est privilégié si vous voulez faire un jeu pour Android.

Le **C#** est semblable au Java. Toutefois, il a souvent été limité aux plateformes Microsoft. Avec, vous pouvez développer des jeux pour Xbox, Windows Phone et Zune avec XNA, pour navigateur avec Silverlight. Finalement, ce langage est utilisé dans les kits Unity 3D et NeoAxis. Ce langage est fortement utilisé chez les professionnels du jeu vidéo pour la création des outils liés à la conception du jeu (éditeurs, logiciels utilitaires, etc.).

Le BASIC, popularisé car implémenté d'office sur les vieux ordinateurs, est un langage assez simple. Il est toujours possible de faire des jeux avec en utilisant des variantes telles que **PureBasic**, mais ces solutions ont vieilli.

Le **Python** est un langage de script simple et puissant. Il est généralement facile à apprendre et est de plus en plus utilisé dans les jeux vidéo pour implémenter la logique du jeu. De plus, il existe de nombreuses bibliothèques aidant à la programmation d'un jeu, comme la très célèbre **pygame**.

Sachant que les jeux pour le web sont une catégorie bien à part, je vous invite à vous référer à la section Jeux web dans la suite de ce tutoriel.

Ceci est un rapide tour des langages les plus importants, mais il reste plein d'autres solutions.

*D'après moi, le **Python** semble un bon choix pour débuter.*

V-B - La bibliothèque

Maintenant que vous avez choisi votre langage, il est nécessaire de déterminer les bibliothèques dont vous avez besoin pour réaliser votre jeu.

Une bibliothèque est une boite contenant une série de fonctions réalisant des tâches simples. Cette boite est nécessaire car il serait très difficile de créer un jeu si nous devions à chaque fois refaire les fonctions de base comme « dessiner un point sur l'écran » ou « ouvrir une fenêtre ».

Comme pour les langages, il existe différents niveaux de bibliothèques. Le bas niveau se retrouve au plus proche de la machine et proposera des fonctions simples du style : « dessiner un point sur l'écran », alors que les bibliothèques de haut niveau proposeront des fonctions permettant de dessiner directement un modèle 3D complet à partir d'un fichier.

*Comme il n'est pas possible d'énumérer tous les choix possibles en termes de bibliothèques et de langages, je vais me concentrer sur les bibliothèques les plus courantes en C/C++. Il est possible de découvrir les autres choix sur la page suivante : **http://jeux.developpez.com/telecharger/index/categorie/557/Bibliotheques***

V-B-1 - Bibliothèques bas niveau

Pour la 3D, le choix est assez simple. Il y a **OpenGL** et **DirectX**. Le choix se fait selon les plateformes visées. En effet, DirectX n'est supporté que sous Windows et sur Xbox. OpenGL est purement C alors que DirectX est en C++, toutefois, des surcouches (wrappers) permettent d'utiliser ces bibliothèques avec d'autres langages. OpenGL existe dans une variante embarquée : OpenGL ES, utilisée dans les plateformes mobiles (iPhone/iPad/Android).

Alors que OpenGL se limite uniquement à la 3D, DirectX permet aussi de lire les fichiers des modèles 3D, de jouer des sons, de gérer les périphériques de jeu.

Pour la 2D, vous pouvez utiliser la **SDL** (qui est très portable (compatible même avec la GP2X)), **SFML** (légèrement moins portable) et **Allegro**. Plus précisément, ces bibliothèques ne se limitent pas à afficher des images en 2D mais gèrent aussi le son, les périphériques et tout ce qui est nécessaire à la création d'un jeu.

Il est aussi possible de faire de la 2D avec OpenGL et DirectX.

Pour la musique et le son, les solutions sont **OpenAL** ou directement utiliser les fonctionnalités données par le système d'exploitation. OpenAL permet une spatialisation du son et permettra de jouer les sons dans un environnement 3D.

Pour la physique, la bibliothèque à choisir dépendra de votre projet. En effet, certaines bibliothèques se spécialisent dans la physique dans un monde 2D, alors que d'autres agiront sur un monde 3D. Parmi celles-ci, nous pouvons citer **Box2D**, **Bullet**, **Newton**.

Pour le réseau, les bibliothèques sont liées au système d'exploitation pour lequel vous développez. En effet, chaque système a sa propre implémentation des sockets qui sont la forme la plus basique pour créer une connexion entre deux machines.

*Un choix souvent fait par les débutants (choix qui se trouve être un des meilleurs pour faire un jeu rapidement) est de créer son jeu avec SDL et ses modules SDL_mixer, SDL_image et SDL_TTF. Vous pouvez retrouver des explications détaillées dans **nos tutoriels**.*

Bien sûr, ce paragraphe ne peut pas lister toutes les bibliothèques disponibles. Je vous invite donc à visiter notre page **Bibliothèques** ou à effectuer des recherches sur Internet.

V-B-2 - Moteurs

Les moteurs sont aussi des bibliothèques mais ceux-ci encapsulent les bibliothèques bas niveau afin de donner un moyen simplifié de faire son jeu. Ainsi, dans les moteurs, des algorithmes avancés seront mis en place vous évitant de devoir les réimplémenter.

Tout comme les bibliothèques bas niveau, les moteurs peuvent se spécialiser :

- **pour la 3D** : **Ogre**, **Irrlicht** et **OpenSceneGraph** ;
- **pour la physique** : **ODE**, **Havok** (souvent utilisé dans les jeux vidéo), **PhysX** et **Tokamak** ;
- **pour le réseau** : **RakNet**, **HawkNL** ou encore **GNE** ;
- **pour le son** : **FModEx** ou **BASS**. FModEx est une bibliothèque couramment utilisée dans le monde des jeux vidéo (Blizzard l'utilise dans Diablo 3 et Starcraft 2).

Cette liste n'est bien sûr pas exhaustive, tant dans les moteurs cités que dans les catégories abordées. On pourra trouver des moteurs un peu plus spécifiques, par exemple des moteurs d'animation 3D, des moteurs de gestion des périphériques, etc.

V-C - Jeux web

Les jeux web sont devenus de plus en plus populaires ces dernières années. En effet, ce succès peut être dû à leur facilité d'accès : il n'est pas nécessaire d'avoir une machine surpuissante, ni même de suivre une procédure d'installation complexe et lente. Il suffit bien souvent de démarrer son navigateur internet pour pouvoir y jouer.

V-C-1 - Côté serveur

Le serveur est la machine à laquelle les utilisateurs se connecteront pour jouer. Dans le cas d'un jeu web, le serveur est une machine proposant un site web et est accessible à l'aide d'une URL.

Que vous fassiez un jeu en HTML, en Flash ou avec Unity, si vous voulez le rendre disponible il faudra l'héberger. Cet hébergement est payant et il peut vous en coûter plus ou moins cher selon vos besoins en espace de stockage et en bande passante.

Pour ceux-ci, le site pourra se résumer à sa plus simple forme, une page HTML contenant le code pour importer le module du jeu.

Il existe cependant un autre style de jeu, appelé jeu par navigateur qui repose essentiellement sur l'utilisation du serveur. Dans cette catégorie nous retrouvons les classiques OGame ou Travian.

Ogame

La conception d'un tel jeu est similaire à celle d'un site web. Ainsi, il sera programmé dans un langage comme le PHP, les pages affichées du côté client en HTML/CSS et du JavaScript pour rendre le tout dynamique aux yeux de l'utilisateur.

Maintenant, il existe des frameworks permettant d'éviter le PHP et d'utiliser d'autres langages. C'est le cas pour **Django**, permettant de faire un site en Python, ou **Ruby on Rails**, utilisant le Ruby.

Comme vous allez avoir besoin de sauvegarder des données sur les joueurs, les monstres ou autres, il vous sera nécessaire d'avoir une **base de données**.

V-C-2 - Flash

Pendant de nombreuses années, le **Flash** a été dominant dans les jeux web. En effet, la technologie de Adobe a été la première à proposer de facilement créer des animations et cela de manière indépendante des navigateurs. Le langage de programmation derrière le Flash s'appelle l'ActionScript (actuellement en version 3). La technologie Flash récemment passée en version 11 permet de faire des applications web en 3D.

Pour réaliser une application Flash, plusieurs **éditeurs** sont disponibles. Adobe développe son propre éditeur : **Adobe Flash CS** mais celui-ci est payant. Un autre éditeur intéressant est **Flash Develop**, car celui-ci est gratuit et vous permet pleinement de réaliser un jeu Flash.

Comme le Flash s'est vite popularisé, de multiples bibliothèques de jeux existent et facilitent la création d'un jeu. En effet, à la base le Flash est prévu pour faire de simples animations et n'est pas spécialement conçu pour les jeux. Les bibliothèques comme **Flixel** permettent d'avoir une gestion simplifiée des composants Flash afin de créer un jeu sans soucis.

*D'un autre côté, la technologie semble perdre des adhérents. D'une part, il est possible de faire les mêmes choses avec le HTML 5 et JavaScript, mais en plus Adobe ne supporte plus d'importantes plateformes comme **Linux** ou **Android**.*

V-C-3 - HTML 5/CSS 3

Depuis quelques années, les langages web ont évolué pour offrir une nouvelle dynamique à nos pages. Ainsi, avec un savant mélange de HTML 5/CSS 3 et de JavaScript (et notamment avec Node.js pour le serveur ou encore avec les **websockets**), il est maintenant aisé de réaliser des **jeux complets** et supportés par la majorité des navigateurs et plateformes.

Malgré le jeune âge de ces technologies, différentes bibliothèques existent pour aider à la création d'un jeu : **Impact**, **LimeJS** et bien d'autres.

Il existe aussi des logiciels spécialisés comme **Construct 2**.

V-C-4 - Unity 3D/Unreal Development Kit

Les kits de développement Unity 3D et Unreal Development Kit permettent d'exporter son jeu dans une page web.

V-C-5 - Facebook

Le développement de jeux vidéo a atteint une nouvelle dimension avec l'arrivée des réseaux sociaux tels que Facebook. Ce dernier a mis en place une bibliothèque afin de réaliser des applications pour le réseau et notamment des jeux. Vous pouvez trouver plus d'informations sur le portail de la firme : **http://developers.facebook.com**.

V-D - Les outils

Lors de la conception d'un jeu vidéo, il est souvent nécessaire d'utiliser des outils additionnels.

Il vous faudra un outil permettant d'écrire du code. Vous pouvez bien sûr utiliser le bloc-notes, mais comme vous allez sûrement passer beaucoup de temps sur votre code (et que celui-ci n'est pas toujours aussi simple à lire qu'un roman), il est préférable d'utiliser un éditeur spécialisé qui vous apportera des fonctionnalités essentielles telles que la coloration syntaxique. Ensuite, il peut être nécessaire de créer un diagramme UML afin de mettre en place

l'architecture de votre jeu. Finalement, et cela devient obligatoire lorsque l'on travaille en groupe, il est nécessaire de mettre en place un logiciel de contrôle de versions.

Ce sont les outils de base pour créer un jeu vidéo. Toutefois, d'autres outils peuvent vous aider, alors n'hésitez pas à jeter un coup d'œil sur **cette page**.

V-D-1 - L'éditeur de code

Un Environnement de Développement Intégré (EDI) est un logiciel regroupant un éditeur de code ainsi que les outils nécessaires pour compiler et produire l'exécutable de ce code. De plus, il contient un débogueur intégré permettant de traquer les bogues de manière simple et efficace.

Pour le C/C++, les EDI les plus connus sont : **Microsoft Visual C++**, **Code::Blocks** et **QtCreator**.

Pour le C#, les EDI sont : **Microsoft Visual C#** et **MonoDevelop**.

Pour le Java, on parlera de **Eclipse** et **NetBeans**.

Bien sûr, cette liste n'est pas exhaustive, de plus un éditeur peut ne pas se cantonner à un seul langage, tel que Eclipse qui est capable aussi de traiter du C ou encore du PHP. Si vous aimez un autre logiciel et que vous vous sentez bien dans celui-ci, alors ne changez pas.

V-D-2 - La modélisation UML

La modélisation est importante afin de ne pas partir tête baissée dans le code et de rencontrer des problèmes d'évolution dans votre logiciel. Bien que la feuille de papier soit le premier outil pour dessiner les diagrammes, il peut être nécessaire d'avoir un logiciel afin de créer son modèle sur ordinateur et de pouvoir ainsi le distribuer aux autres membres de l'équipe.

Les outils disponibles pour créer vos diagrammes sont répertoriés sur **cette page**.

V-D-3 - Logiciels de contrôle de versions

Lorsque l'on travaille à plusieurs sur un même code, il est important de mettre en place un logiciel de contrôle des versions. Celui-ci permet de sauvegarder les différentes modifications de vos fichiers de code et de les gérer. Même si vous travaillez seul, il est conseillé d'en utiliser un afin d'avoir des sauvegardes de votre travail sur un serveur.

Tout d'abord, il existe plusieurs services proposant des serveurs de contrôle de versions (là où votre code sera stocké). Certains de ces services forcent l'utilisation d'une licence où le code doit être ouvert. De plus, le choix d'un de ces services dépendra aussi du protocole que vous voulez utiliser (CVS, SVN, Mercurial, GIT, etc.).

Developpez.com propose ce genre de services, avec un bugtracker et tous les autres outils nécessaires pour faire vivre son projet.

Les autres solutions sont les forges telles que **SourceForge**, **Google Code**, **Assembla** ou **GitHub**.

V-D-4 - Logiciels de suivi des bogues (bugtrackers)

Il peut aussi être intéressant d'utiliser un bugtracker pour suivre l'évolution des bogues ou recevoir les commentaires des utilisateurs de votre jeu. Souvent, les services associés à l'utilisation d'une forge vous donnent l'accès à un bugtracker.

Toutefois, il vous est tout à fait possible de l'installer sur votre propre machine. Les plus connus sont : **Mantis** et **Bugzilla**.

Il peut être aussi utilisé pour gérer les différentes tâches du projet et par la même occasion, voir si le projet se déroule correctement.

V-D-5 - Logiciels de gestion de projet

Si vous souhaitez une solution plus complète qu'un simple bugtracker, il est possible d'installer un logiciel de gestion de projet. Généralement, ceux-ci se composent d'un bugtracker, d'un wiki, d'un forum, d'une gestion des utilisateurs et de leurs droits sur les projets, d'une intégration avec le gestionnaire de versions et de feuilles de route. Bien entendu, il faudra vous renseigner afin de voir celui qui correspond le mieux à vos attentes.

Parmi ces logiciels, je citerais **Redmine** et **Trac** qui ont l'avantage d'être gratuits et de permettre de gérer un projet.

Pour un premier projet, d'autant plus si vous êtes seul et que le projet n'est qu'un simple petit jeu, l'utilisation d'un bugtracker ou d'un logiciel de gestion de projet peut être un peu lourde. Toutefois, cela ne peut qu'être une bonne expérience de découverte. Pour les projets en groupe, le gestionnaire de projet permettra une meilleure communication et un meilleur suivi du projet. Cela permet aussi de garder un historique des discussions et choix effectués pour le projet.

VI - La plateforme

Le choix de la plateforme s'est diversifié ces dernières années. En effet, il est devenu facile de développer des applications pour des plateformes autres que le PC.

Dans tous les cas, vous allez effectuer votre développement sur votre PC, car l'écriture du code et la compilation s'effectuent sur une machine classique.

Il faut savoir que si vous souhaitez cibler une plateforme telle qu'une console ou un smartphone, cela ajoutera un peu de difficulté au développement. Il est donc préférable de commencer avec quelques connaissances en programmation.

VI-A - Les plateformes Apple

Tout d'abord, pour développer pour l'iPhone ou l'iPad, il vous faudra un Mac. Des tutoriels sont à votre disposition à cette adresse : **http://ios.developpez.com/cours/**.

Votre point de départ sera l'installation du SDK : **http://a-renouard.developpez.com/tutoriels/ios/installation-developer-tools/**.

VI-B - Un jeu sur Android

Android est le concurrent direct d'iOS. Vous pouvez trouver plus de tutoriels ici : **http://android.developpez.com/cours/**.

VI-C - PSP/PS3

Il est aussi possible de développer des jeux pour les plateformes portables de Sony en utilisant des kits non officiels tels que : **http://sourceforge.net/projects/minpspw/**.

Toutefois, il vous faudra un émulateur pour tester vos réalisations (ou encore, en débridant votre console).

Malheureusement, aucun kit officiel n'est disponible gratuitement pour ces consoles. Seule une bêta du kit est disponible : **http://www.playstation.com/pss/developer/index_e.html**.

VI-D - Wii/Nintendo DS/Nintendo 3DS

Il est plus simple d'aborder la Nintendo DS que la PSP.

Bien que le kit officiel ne soit pas disponible, il existe plus de ressources sur la question : **http://krachik.developpez.com/tutoriels/pascal/install-config-fpc4nds/** ou **http://ds-homebrew.over-blog.com/**.

Il est aussi possible de faire des jeux sur GBA, comme expliqué dans ce tutoriel : **http://gfx.developpez.com/prog-gba/**.

Pour la Wii il existe ces ressources : **http://www.codemii.com/2008/08/10/tutorial-1-setting-up-the-environment/**.

VI-E - Xbox/Windows Phone

Microsoft est plus ouvert que ses concurrents pour le développement amateur de jeux. La première solution est d'utiliser **XNA** qui est une bibliothèque de jeux utilisable en C#.

La documentation officielle de Microsoft est très complète sur le sujet, comme vous pouvez en juger : **http://msdn.microsoft.com/en-us/library/bb203937.aspx**.

Il vous est aussi possible d'utiliser Kinect, le capteur de Microsoft, à l'aide du kit officiel : **http://www.microsoft.com/en-us/kinectforwindows/develop/**.

Avec ce même kit, il est possible de créer des jeux pour Windows Phone.

VII - Un choix bien trop souvent oublié ?

Il reste une alternative que l'on écarte/oublie trop souvent. Celle de s'intégrer à un projet Open Source. En effet, l'avantage qu'a un projet Open Source, c'est qu'il est déjà assez avancé pour avoir une certitude de se finir (et d'être présentable). Ainsi, un artiste pourra soumettre ses dessins, un game designer ses idées et un programmeur pourra participer à un projet et rajouter des fonctionnalités intéressantes.

VII-A - Licence

Il est très important de lire les documents liés à la licence. En effet, chaque bibliothèque ou logiciel est soumis à des droits d'auteur. La licence permet de déterminer ce qu'il est possible de faire et ce qu'il ne l'est pas.

Les kits de développement (UDK, CryEngine SDK, etc.) possèdent une licence qui permet de réaliser des jeux gratuits, mais dès que vous souhaitez vendre ceux-ci, il faudra payer l'éditeur du kit pour avoir l'autorisation.

Certaines licences pour les bibliothèques peuvent vous obliger à garder votre code ouvert. D'autres n'acceptent pas d'être utilisées dans des applications commerciales.

En réalité, tout est possible en termes de licence. L'utilisation d'un logiciel ou d'une bibliothèque implique que vous acceptiez la licence associée. Donc, lisez bien les licences avant de distribuer votre jeu (le choix des logiciels/bibliothèques est aussi à faire selon la licence).

En conclusion, si vous souhaitez revendre votre projet, ou même, rendre le code disponible librement, il est nécessaire de vérifier que les logiciels et licences des bibliothèques que vous utilisez au sein de votre projet vous laissent la liberté de faire ce que vous souhaitez. Si cela n'est pas le cas, le créateur de la bibliothèque originelle peut toujours demander la fermeture de votre projet.

VIII - Conclusion

N'hésitez pas à demander de l'aide ou des informations supplémentaires sur la **rubrique Jeux** et sur **son forum**. De plus, comme il est difficile d'être exhaustif sur les technologies permettant de créer des jeux, je ne peux que vous conseiller de jeter un coup d'œil à nos pages des **outils** et des **bibliothèques**.

IX - Remerciements

Je souhaite remercier **ovh** et **Neckara**. Finalement, je remercie aussi **_Max_** et **ClaudeLELOUP** pour leurs corrections.