

Conception de jeux massivement multi-joueurs

2010

Conception de jeux massivement multi-joueurs

Mémoire

Jonathan Muller
08/02/2010

Sommaire

Contexte du mémoire :	4
Qu'est ce qu'un jeu massivement multi-joueurs ?	5
Définition:	5
Les précurseurs:	5
De grands succès:	6
Un nouveau secteur:	6
Le marché actuel :	7
Nombre de joueurs par jeu : estimations	7
Côté serveur:	8
Côté client:	9
Interactions côté serveur:	11
Les actions du joueur:	11
Interprétation du serveur:	11
Pourquoi déléguer les calculs ?	12
Alléger le serveur:	13
Disponibilité du serveur:	14
Localisation des serveurs/connexion:	14
Gestion de la mémoire:	15
Sécurisation des serveurs:	15
Protection des sources et des données:	16
Evolution du jeu:	16
Microsoft SQL Server:	17
Oracle:	17
MySQL:	17
Prise en compte des différences de matériel :	18
Pré chargement des données :	18
Sécurisation du client :	19
Latence du jeu :	19
Contenu du client :	19

L'univers du jeu:	20
Économie du jeu:.....	20
Design:.....	20
Musique/ambiance/bruitages:.....	20
Gestion des compétences / égalité entre les classes / armes :	21
Conception des niveaux/cartes/donjons:	21
Évolution du personnage / niveaux / caractéristiques:	21
Additivité du jeu:	22
Gameplays:.....	22
Les P2P.....	23
Les F2P.....	23
Les gratuits à licence payante:	24
Les jeux entièrement gratuits:	24
Communauté:.....	25
Les game masters:.....	25
Viser un public:.....	25

Contexte du mémoire :

Ce mémoire a été réalisé dans le cadre de ma formation à l'ITIN.



Créée en 1988 par un réseau d'entreprises, l'ITIN est une école supérieure publique d'informatique, de réseaux et systèmes d'information. Elle est située au cœur de Cergy Pontoise.

-Site de l'ITIN

Dans ce mémoire, nous allons analyser ce qu'il faut savoir ou faire pour réaliser un jeu massivement multi-joueurs.

Il est clair que le contenu de ce mémoire concerne l'utilisation commerciale des jeux, il n'est en aucun cas un tutoriel, il ne s'adresse pas à une conception amateur. Bien que ce genre de conception soit possible, elle ne sera pas exploitable commercialement et pourra être réalisée de multiples façons. Nous recherchons ici une conception solide, organisée, productive et efficace pour un jeu à plusieurs centaines de milliers de joueurs.

Qu'est ce qu'un jeu massivement multi-joueurs ?

Définition:

Le **jeu en ligne massivement multi-joueurs (MMOG**, de l'anglais *massively multiplayer online game*, parfois encore abrégé en **MMO**) est un genre de [jeu vidéo](#) faisant participer un très grand nombre de joueurs simultanément par le biais d'un [réseau informatique](#) ayant accès à [Internet](#).

- Wikipédia

Les jeux vidéo prennent un nouveau tournant depuis maintenant plusieurs années. L'accès généralisé à internet permet à un nombre de plus en plus élevé de joueurs de participer à ce nouveau type de jeu.

Les précurseurs:

Les premiers jeux massivement multi-joueurs proprement dits sont apparus dans les années 90, avec les premiers accès à internet. Mais le jeu considéré comme le premier à avoir démocratisé le genre est Ultima-online, sorti en 97. Sa première mise à jour, ou extension, sort un an plus tard alors qu'il affiche déjà 100.000 joueurs inscrits. Pendant ce temps le jeu Lineage sort et après un an, il franchit la barre des 3 millions de joueurs.

De grands succès:

C'est à partir de ce moment que les jeux massivement multi-joueurs ont commencé à faire parler d'eux. Ces jeux de niche ont alors attiré les plus grands éditeurs de jeu. Après un tel succès, ce sont des jeux comme Everquest, La 4ème prophétie, Lineage 2, puis World of Warcraft pour ne citer que les plus réputés, qui finissent par sortir et qui ont aujourd'hui le succès qu'on leur connaît. En 2010, plus de 400 jeux sont considérés comme jeux massivement multi-joueurs.

En 2006, on estimait la communauté des jeux massivement multi-joueurs à environ 10 millions de joueurs. Un an plus tard, elle était estimée à plus de 50 millions.

Un nouveau secteur:

C'est alors toute une industrie qui s'ouvre à la concurrence, de nouveaux concepts, de nouveaux succès apparaissent, et ce type de jeu attire un nombre de plus en plus élevé de joueurs, à travers toutes les plates-formes. Sur l'ordinateur, sur console, même via le navigateur ou un simple téléphone mobile, les jeux en ligne massivement multi-joueurs sont devenus un véritable phénomène de société.

Le marché actuel :

Nombre de joueurs par jeu : estimations



World of Warcraft – 13 Millions



Guild Wars – 5 Millions



Knight Online – 5 Millions



Dofus – 3 Millions



Runescape – 1 Million



Lineage 2 – 1 Million

Ces chiffres sont des estimations réalisées par mmogdata, et sont à prendre avec prudence. En effet il est très difficile de récolter ces données, les éditeurs communiquant très peu sur leur nombre de joueurs.

Les Langages utilisées:

Côté serveur:

.NET: Les langages .NET sont tout à fait utilisables dans le cadre du serveur. Le Framework .NET est maintenant assez mature et puissant pour offrir tout ce que nous avons besoin pour notre serveur. Son seul défaut est qu'il nécessite une plate-forme Microsoft Windows. Bien que les serveurs Windows soient très puissants et suffisent à notre projet, il ne faut pas que le langage force une architecture. Si Windows server est notre choix, nous pouvons choisir un langage .NET. De plus, nous pouvons utiliser ASP si notre jeu est basé sur un navigateur, ou même pour faire dialoguer le web avec le serveur de jeu.

Java: Java est un langage puissant, stable et surtout multi plateformes. Cependant dans notre projet nous n'aurons pas besoin d'avoir un serveur multi plateformes. Une fois mis en place, les serveurs principaux utiliseront la même technologie et il est peu probable que la structure change. Java perd alors un peu de son intérêt mais il reste un langage de choix, notamment pour une application web.

C++: Ce langage est très puissant, il permet d'avoir des performances extrêmes car il est très proche du langage machine, contrairement aux autres ci-dessus. Cependant ce langage nécessite une connaissance et une compréhension approfondies de celui-ci. Ainsi il faut avoir une équipe de développeurs solide pour la mettre en place. Ce langage peut être multi plateformes. Cela va dépendre des bibliothèques utilisées. La puissance des machines actuelles nous permet de réaliser le serveur en .NET ou en Java. Est-il bien nécessaire de choisir le C++ qui est bien plus difficile à mettre en place et à maintenir ?

PHP: Un langage de choix, extrêmement puissant et portable, le PHP est sûrement le bon choix si le jeu est basé sur navigateur.

Côté client:

.NET : les langages .NET sont utilisables, à condition d'utiliser la technologie XNA. Elle permet de réaliser des applications 2D ou 3D très simplement sur les plates-formes PC et Xbox. Il n'y a quasiment aucun changement à apporter pour passer d'une version à l'autre, et il supporte les différences d'architecture des machines qui l'exécutent. Si notre choix est de porter le jeu sur console, notamment Xbox, il peut être intéressant de le choisir. C'est une technologie assez jeune, mais son utilisation permet d'obtenir un résultat très rapidement, et déjà quelques moteurs de qualité l'utilisant existent.

C++ : Pour réaliser le jeu côté client, il est difficile de trouver mieux. Le C++ réalise des merveilles, grâce à sa puissance il peut faire tourner des jeux aux graphismes très poussés avec des performances élevées. Cependant vu la puissance des machines actuelles, cet argument est de moins en moins valable par rapport au XNA.

Java : Il est à éviter pour tout client lourd. Mais pour un jeu massivement multi-joueurs basé sur navigateur, il est tout à fait utilisable. En effet Java a beaucoup de mal à calculer et afficher la 3D. Bien que ce soit possible, il faudra se contenter de graphismes médiocres. Cependant sur le web cela est tout à fait correct.

Les principaux Framework:

Abyssal Game Engine: Nécessite Visual studio 2005 ou plus

Le moteur Abyssal est un moteur assez puissant, orienté jeux multi-joueurs avec toute une collection d'outils pour créer son jeu. Il contient par exemple : un éditeur d'animation, de monde, un import de modèles 3DSMAX, un éditeur d'interface utilisateur, avec possibilité de réalisation du client en parallèle, etc.

BigWorld Server:

Développé uniquement pour la réalisation de jeux massivement multi-joueurs, la technologie BigWorld s'avère être un bon choix. Il gère le taux de charge des serveurs pour faire basculer la charge sur un autre en cas de débordement. Il supporte les architectures multi-cœurs et 64 bits. Certains grands éditeurs ont opté pour celui ci.

Visual3D Game Engine:

Ce moteur est développé spécialement pour .NET et XNA, il propose un outil qui permet de gérer tout ce qui s'affiche à l'écran : des éditeurs pour gérer le terrain, les quêtes, les modèles, les scripts, etc. Il est possible de garder la compatibilité avec les Xbox pour un déploiement éventuel.

OpenSpace:

OpenSpace est un éditeur en ligne, basé sur Flash pour réaliser un jeu massivement multi-joueurs basé sur navigateur. Il utilise Action Script 3 et SmartFoxPowerServer. Il ne permet de réaliser que des jeux isométriques.

Conception d'un jeu massivement multi-joueurs:

Interactions côté serveur:

Toutes les interactions sont calculées côté serveur, le client n'a comme rôle que d'afficher les informations que lui envoie le serveur, il ne doit faire que transmettre visuellement au joueur ce que le serveur veut que le joueur voit. Contrairement à un jeu conventionnel hors ligne, où tout est calculé au sein même du jeu, ici le fonctionnement doit être différent.

Les actions du joueur:

Chaque action du joueur requiert la validation du serveur. Il faut vérifier chaque demande du joueur. On peut donc représenter chaque action du joueur comme ceci: le joueur souhaite réaliser une action, il utilise donc le jeu pour faire la demande au serveur. Par exemple, pour avancer, il va appuyer sur une touche ou cliquer avec sa souris là où il veut se diriger.

Interprétation du serveur:

Le client interprète alors la commande du joueur et envoie l'information au serveur via la connexion internet du joueur. Le serveur analyse alors la demande du joueur, il effectue tous ses calculs et renvoie au client un signal pour l'avertir que son action est prise en compte, ou que celle-ci est impossible. Le serveur envoie alors l'information à tous les autres joueurs aux alentours.

Pourquoi déléguer les calculs ?

Il est nécessaire de se baser sur ce modèle si l'on veut éviter tout « hack », autrement dit une modification du logiciel côté client pour éditer le contenu du jeu. Ce genre d'action est dangereux pour notre jeu, dans la mesure où aucune action qui serait réalisée côté client ne doit pouvoir avantager ou désavantager un joueur.

Par exemple, même si le joueur arrive à créer un hack sur son client pour se déplacer rapidement, le serveur considérera son action « avancer » comme déclenchant une augmentation de l'indice x de 1 par exemple. Si son hack augmente x de 100, ses interactions côté serveur seront toujours par rapport à sa position en $x+1$. C'est à dire que même si un joueur change sa position côté client, le serveur a retenu sa position réelle. Si par exemple il essaye de ramasser un objet à la position où il s'est placé le serveur ne lui permettra pas de réaliser cette action: il n'est pas à portée.

On suit le même principe pour toutes les actions réalisables dans le jeu. Ceci nécessite donc un serveur extrêmement puissant, mais aussi une optimisation sans faille de celui-ci. Il est nécessaire d'utiliser un langage rapide et très proche du langage machine. Cependant la puissance des serveurs actuels permet de plus en plus d'utiliser des langages de couche plus haute.

Il y a 5 ans, un serveur de ce type nécessitait plus de 16Go de mémoire vive.

L'autre avantage de déléguer les calculs au serveur est de pouvoir réaliser des mises à jour, majeures ou mineures sans déclencher une mise à jour du client, ce qui obligerait le joueur à télécharger une nouvelle version d'un ou plusieurs fichiers du jeu. Dans le cas d'une mise à jour du noyau du jeu, ce téléchargement peut s'avérer long. Les mises à jour étant plutôt régulières dans tout jeu massivement multi-joueurs, ceci peut s'avérer rapidement handicapant pour le joueur qui possède souvent peu de temps pour jouer.

Alléger le serveur:

Comme nous l'avons vu, le nombre de calculs à réaliser par le serveur est relativement important. Il faut donc penser à un système pour alléger le serveur et permettre à un nombre élevé de joueurs de réaliser tout type d'action. Il est possible de diviser le monde en plusieurs sous-serveurs, par zone.

Ainsi le joueur passe virtuellement d'un serveur à un autre en fonction de la zone dans laquelle il se situe. Cela doit être transparent pour lui et les autres joueurs. Il ne faut pas que les autres joueurs de la zone le voient disparaître d'un coup, ou apparaître juste à côté d'eux. Il faut donc créer une zone de débordement dans laquelle les joueurs se situent sur les deux serveurs. Lorsque la limite des deux zones est atteinte, le joueur passe automatiquement d'un serveur à un autre. La zone d'emboîtement peut être approximativement de la taille de la vision d'un joueur sur l'horizon du jeu.

Cependant pour donner l'impression au joueur qu'il se situe dans le même monde que les autres, il est nécessaire de garder la communication entre eux. On peut penser à un dialogue entre les sous serveurs ou à un serveur dédié aux communications.

On peut donc imaginer que le monde virtuel peut être divisé en plusieurs sous-serveurs pour chaque type d'action.

Il est également possible de diviser ces sous-serveurs en plusieurs canaux (qui peuvent être eux mêmes des sous-serveurs, ou non). Cette technique a été utilisée pour le lancement de jeux à succès, afin d'alléger les zones de départ et réduire le nombre de joueurs affichés à l'écran. Cependant cette technique est déconseillée dans la mesure où elle permet aux joueurs d'en éviter certains autres. Un bon moyen consistera à utiliser cette technique au lancement du jeu pour les zones de départ, puis d'interdire le changement de canal dans les zones où les joueurs commencent à être moins nombreux.

Disponibilité du serveur:

Il est absolument nécessaire d'assurer une disponibilité à 100% pour le serveur. Les joueurs payent pour avoir une qualité de service irréprochable. Il faut soigner la qualité de jeu pendant toute sa durée de vie. Un serveur avec une forte latence, des bugs trop nombreux, et souvent hors ligne discréditera le jeu, notamment lors de sa sortie. Ainsi le jeu sera voué à l'échec. Ceci a été vérifié lors du lancement de Warhammer: Age Of Reconning, jeu avec des bugs trop nombreux, et un serveur avec une trop forte latence. Le jeu est entré en phase de déclin quelques mois après sa sortie.

Localisation des serveurs/connexion:

Avant de lancer notre jeu, il faut analyser la localisation éventuelle des joueurs qui vont rejoindre notre monde virtuel. Une fois effectué, il faut placer nos serveurs de manière à offrir à chacun des joueurs une expérience maximale sur au moins un des serveurs. Une bonne solution consistera à placer dans chaque pays un serveur prêt à accueillir les joueurs aux alentours. Cependant certains pays n'offrent pas un nombre suffisant de joueurs pour offrir plusieurs serveurs, mais en offrent trop pour n'en ouvrir qu'un. Il faut alors penser à placer des serveurs internationaux placés entre plusieurs pays chez un hébergeur qui propose la meilleure qualité de ligne.

Gestion de la mémoire:

En général, le serveur a un cycle de maintenance d'une semaine, ce qui signifie que chaque semaine il sera redémarré pour appliquer une mise à jour, ou simplement pour effectuer une série de tests pour être sûr qu'il n'y aura pas de problème pour la semaine suivante. Il faut que durant une semaine entière il n'y ait pas besoin de redémarrer, et ce quelque soit le nombre de joueurs connectés, d'actions réalisées ou le taux de sollicitation du serveur.

Autre contrainte : On ne peut se permettre de faire des accès entre le serveur et la base de données à chaque fois que l'on doit aller prendre des informations. Par exemple, une solution consisterait à charger en mémoire un personnage et tout ce qui le concerne lors de sa connexion, d'effectuer une série de tests durant sa connexion, et de libérer la mémoire associée lorsqu'il se déconnecte.

La sauvegarde des données aurait lieu lorsque celui-ci se déconnecte, ou après sa connexion toutes les x minutes en fonction de la charge du serveur en utilisant une zone tampon pour effectuer les instructions au serveur de base de données, ceci afin de ne pas surcharger le serveur de jeu avec ces sauvegardes qui ne sont pas critiques pour le déroulement du jeu.

Sécurisation des serveurs:

Les serveurs doivent être sécurisés afin de ne subir aucune perte d'information, même en cas de problème hardware (crash disque, mémoire). Il faut donc installer un système de réplication entre plusieurs serveurs, où les instructions vont s'exécuter en parallèle sur ces serveurs. Ainsi, si l'un est défaillant, un autre peut prendre le relais avec les mêmes données.

On peut installer un système de raid sur les disques du serveur, afin de pouvoir les remplacer à chaud en cas de panne sans subir de perte de données.

Protection des sources et des données:

La protection des comptes et des personnages est vitale. On ne peut pas se permettre d'avoir un bug impactant ceux-ci, ça serait la fin pour notre jeu massivement multi-joueurs.

Par exemple, le célèbre jeu Lineage 2 a connu de nombreuses fuites de ses logiciels. De nombreux serveurs pirates sont donc apparus, et bien que ceux-ci aient d'un côté permis de faire connaître le jeu à un grand nombre de joueurs grâce à sa gratuité, ils ont aussi permis à un nombre considérable de joueurs de quitter les serveurs officiels pour aller jouer gratuitement sur ces serveurs pirates.

Il faut également faire très attention avec les sources logicielles et l'accès des employés à ceux-ci. Lineage 3 a connu ce désagrément lorsque les employés ont quitté l'équipe de développement, ils ont vendu le code source du logiciel à une entreprise concurrente. Il faut donc faire attention aux accès et essayer de limiter l'emprise des employés sur le code, si possible ne jamais leur laisser l'accès au code source complet.

De même, la protection des comptes (incluant les personnages) est vitale. On ne peut pas se permettre d'avoir un bug impactant ceux-ci, ça sonnerait la fin pour notre jeu massivement multi-joueurs. En effet, les données de compte associées à chaque joueur doivent être préservées, car cela représente des centaines d'heures passées sur le jeu. On risque outre de faire perdre son temps au joueur qui devra repartir de zéro, de perdre la confiance des nouveaux joueurs et d'entacher la réputation de l'entreprise. Enfin, cette protection doit permettre d'éviter des actes de malveillance sur ces comptes (piratage pour revente d'objets contre de l'argent réel).

Il faut donc sécuriser les comptes joueurs à l'extrême, d'employer tous les moyens à notre disposition pour les protéger.

Evolution du jeu:

Il faut concevoir le jeu en pensant nécessairement à l'évolution de celui-ci. Si on veut fidéliser le joueur, on ne peut concevoir un jeu figé. Il se doit de proposer aux joueurs une expérience qui change, qui évolue avec lui au fil du temps. De plus, l'équipe qui conçoit le jeu peut changer au cours du temps, il est donc nécessaire de concevoir le jeu en pensant aux évolutions futures.

Les extensions du jeu, payantes ou gratuites, paraissent généralement à un intervalle de 6 mois à 1 an et demi, rarement plus.

Serveur de base de données:

Cette partie est très importante, il est nécessaire d'avoir un serveur stable, extrêmement puissant, sécurisé, et disposant de tout un tas d'outils pour l'administrer avec méthode. Il existe plusieurs serveurs de base de données, avec chaque des avantages et des inconvénients.

Microsoft SQL Server:

Si notre solution est réalisée pour Microsoft Windows, c'est sûrement ce SGBD qu'il nous faut. Microsoft SQL Server est certainement l'un des SGBD les plus performants sur Windows. De plus, il intègre un nombre impressionnant d'outils pour administrer le serveur, ainsi que pour avoir accès à l'audit de celui-ci. Il gère la charge des différents serveurs. Si nous utilisons .NET, il peut être intéressant de bénéficier des objets de DAO du langage pour ce SGBD.

Oracle:

Ce type de SGBD est optimisé pour une utilisation intensive de la base de données. Son principal avantage est qu'il permet de configurer absolument tout dans votre base. Cependant il ne faut pas perdre de vue que ce type de SGBD demande une quantité de ressources non négligeable.

MySQL:

Si nous souhaitons réaliser un jeu massivement multi-joueurs via navigateur, MySQL peut nous intéresser si nous utilisons PHP. En effet MySQL est très bien intégré avec PHP. Bien que les autres SGBD soient tout aussi utilisables, MySQL me semble être le meilleur choix pour une application orienté web.

Le client du jeu:

Prise en compte des différences de matériel :

Pour viser un large public, le jeu doit pouvoir tourner sur les machines les plus modestes, le moteur doit donc être léger et les graphismes n'ont pas besoin d'être très poussés. Cependant, il faut réaliser un environnement riche et un design soigné pour ne pas exclure les joueurs les plus exigeants. Une optimisation est donc nécessaire côté client, et les options d'affichage doivent être très riches pour pouvoir laisser une grande marge entre la version la plus nécessiteuse en ressources et celle avec tout au minimum.

La configuration des postes des joueurs étant très différente, il faut tester le jeu avec un maximum de combinaisons. Cela étant une tâche difficile à réaliser en interne, nous pouvons passer par une phase de tests, appelée bêta test ouverte ou fermée, dont les joueurs sont souvent friands, car cela leur permet de tester eux aussi le jeu. Souvent cela permet également de déceler les éventuels bugs qui n'auront pas pu être vu autrement, notamment pour les tests de stress avec un nombre important de joueurs.

On suit le même principe pour la bande passante, il ne faut pas que les données transférées soient trop importantes pour que les joueurs disposant d'une connexion bas débit puissent avoir la même expérience de jeu.

Pré chargement des données :

Il peut être intéressant de réfléchir à un système qui permette au client de pré-charger les données sans laisser au joueur la possibilité d'y accéder. Par exemple, si nous chargeons les joueurs qui se trouvent aux alentours de la zone où le joueur est, afin de pouvoir l'afficher avec fluidité et ne pas le saturer au cas où 30 joueurs entrent d'un coup dans la zone, il pourra le voir avec un logiciel qui analyse les trames. Il faudrait donc penser à un système pour empêcher cela.

Sécurisation du client :

Il n'existe quasiment qu'un logiciel de sécurisation pour ce type de jeu. Il s'agit de gameguard, qui va surveiller l'activité de l'ordinateur pendant que le jeu tourne, et vérifier si aucun logiciel tiers est actif. Si c'est le cas, il déconnectera l'utilisateur.

Cependant ce type de logiciel est assez mal vu puisqu'il détecte souvent des logiciels comme étant des logiciels de piratage (comme les macros sur les claviers par exemple).

De plus son inefficacité a été prouvée puisque il est possible de le neutraliser en effectuant des modifications dans les fichiers.

Latence du jeu :

Comme nous l'avons abordé en amont dans ce document, chaque action du joueur nécessite l'aval du serveur. Cependant le temps d'attente pour obtenir la réponse, peut se traduire par une impression de lenteur pour le joueur.

On peut gagner en fluidité en réalisant l'action avant la réponse du serveur, et corriger l'information lorsqu'elle est reçue. Par exemple, commencer à faire avancer le joueur avant d'avoir la réponse du serveur, puis lorsque la réponse est reçue, si la position actuelle est différente de la position calculée par le serveur, alors le client replace le joueur au bon endroit.

Contenu du client :

Puisque tout est délégué au serveur, le client ne doit servir qu'à montrer au joueur ce qu'il se passe dans le monde virtuel. Aucune action affectant le joueur, les autres joueurs ou le monde ne doit être effectuée sur le poste du client.

Ainsi le client contiendra uniquement les textures, les modèles, les terrains, les noms d'objets, les messages d'erreur, l'interface utilisateur.

Conception du jeu:

L'univers du jeu:

Il est important de définir l'univers du jeu bien avant de commencer tout développement, puisqu'il va définir la ligne d'orientation de toute la conception. Des annonces marketing, en passant par le design du site ou du jeu, son histoire vont être la base de celui-ci.

Économie du jeu:

Plus important et difficile qu'il n'y paraît, définir l'économie du jeu nécessite de nombreuses simulations, une analyse très poussée du prix des choses comparée à leur facilité d'acquisition. On peut définir le prix d'une chose en fonction du temps qu'il a fallu pour se la procurer.

Design:

Le design est très important, il va définir le public visé, son âge, son type de jeu, etc. Il doit coller à l'histoire du jeu et le joueur ne doit pas se lasser de parcourir le monde à travers ce style graphique.

Musique/ambiance/bruitages:

Ils définissent l'ambiance du jeu, comment le joueur va être immergé dans celui-ci. Cette ambiance participe à l'addiction du joueur pour le jeu.

Gestion des compétences / égalité entre les classes / armes :

Il faut analyser le jeu sous toutes ses facettes, tester ce que les joueurs peuvent incarner, utiliser, faire et essayer toutes les possibilités pour que le jeu soit le plus équilibré possible dans un maximum de situations. Il faut que quelque soit la voie que choisisse le joueur, il puisse y trouver son compte.

Conception des niveaux/cartes/donjons:

Chaque carte, donjon, doivent être analysés par un designer pour transmettre au joueur des sentiments. Le faire se sentir perdu, guidé, angoissé, héroïque etc.

Évolution du personnage / niveaux / caractéristiques:

Il faut que tout au long du jeu, le joueur puisse trouver son compte en fonction de son niveau ou de ce qu'il a obtenu. Il ne faut pas qu'il puisse s'ennuyer, et en même temps il faut lui rendre plus facile ce qui lui a semblé si dur par le passé. Ainsi il pourra rendre service aux autres, utiliser ses capacités pour avoir de l'argent plus rapidement, etc.

Chacune de ces étapes est spéciale et doit être exécutée par des personnes compétentes qui en ont fait leur métier. Ces étapes vont participer au succès de notre jeu.

Additivité du jeu:

Le jeu doit offrir une expérience qui évolue sans cesse. Le joueur ne doit pas avoir l'impression d'arriver à son terme. Il doit avoir le sentiment de participer à l'évolution du serveur, suivre son histoire au fil des extensions et des mises à jour. Cela passe par l'immersion dans le jeu, via les graphismes, les décors, mais aussi la musique et l'histoire du jeu. Il faut aussi qu'il puisse suivre le jeu quand il n'a pas accès à son poste personnel, d'où l'intérêt des forums, des sites et des ressources liés au jeu lui même.

Gameplays:

Diverses méthodes existent pour offrir au joueur ce sentiment d'évolution: l'école occidentale et l'école coréenne.

Le jeu à l'occidentale est plus basé sur l'équipement, les joueurs doivent parcourir le monde et relever des défis pour améliorer sans cesse leur équipement. Lorsqu'une mise à jour est effectuée ou qu'une extension est proposée, les joueurs sont amenés à recommencer cette course à l'équipement pour rivaliser entre eux.

Le jeu à la coréenne se base plus sur la chasse aux monstres, plus le joueur chasse, plus il gagne d'expérience et plus il est puissant. Cela demande beaucoup de temps et est très rébarbatif mais les joueurs qui passent le plus de temps sur le jeu sont très avantagés par rapport aux autres. Cela engendre quelques problèmes, comme la création de robots qui chassent automatiquement tout au long de la journée même lorsque le joueur est absent. Il faut essayer de contrôler cela et d'empêcher l'essor de ces robots pour éviter de décourager les joueurs réguliers.

Il faut penser à tous les « gameplays », le joueur occasionnel doit pouvoir se faire sa place dans le monde, tandis que le joueur régulier doit avoir un challenge constant. Cela est très difficile à réaliser dans la mesure où le joueur régulier n'acceptera pas de perdre face à un joueur débutant. Il faut donc essayer de consolider les deux types de jeux en séparant éventuellement les zones ou en proposant au joueur une expérience épique tout au long de son évolution, ainsi un joueur même à bas niveau s'amusera et pourra jouer avec les autres joueurs de même niveau que lui, sans qu'il se sente obligé de jouer plus pour atteindre un niveau supérieur.

Les modèles commerciaux :

Les P2P: Pay to play, seuls les joueurs qui payent un abonnement peuvent rejoindre le monde virtuel et participer à son évolution. De manière générale, une période dite de trial, ou d'essai, est proposée aux joueurs souhaitant rejoindre la partie. Ils peuvent alors accéder au jeu et essayer quelques fonctionnalités de base avant de s'engager à payer.

Ce style de jeu a de multiples avantages: pour le joueur, il est assuré d'avoir accès à tout le contenu du jeu sans payer davantage. Ce qu'il voit est ce qu'il peut avoir. Il est sur un pied d'égalité avec les autres joueurs.

Pour les producteurs, ce style de paiement permet de prévoir les revenus sur une période plus ou moins longue. De même cela veut dire pour l'équipe de conception qu'un joueur est fatalement un joueur payant, ce qui permet de concevoir le jeu et ses évolutions de manière collective.

Les F2P: Free to play. Ces jeux sont gratuits. Cependant ils restent des jeux commerciaux et doivent rentrer dans leurs frais. Ces jeux proposent donc le plus souvent à leurs joueurs d'acheter des objets dans une boutique virtuelle, afin de prendre le dessus sur les autres joueurs, ou d'accéder à du contenu bloqué. C'est un modèle en plein essor, car très lucratif, même s'il ne possède pas les avantages des P2P, il offre une grande flexibilité dans l'évolution du jeu et est bien plus facile à concevoir. Du fait de sa gratuité, il peut se permettre de sortir plus ou moins en avance et d'avoir quelques bugs. Sa conception est souvent de moins bonne qualité et le style de jeu se rapproche plus des jeux coréens, même si cela tend à changer au fil du temps.

Les gratuits à licence payante:

On achète le jeu une fois, puis on peut y jouer indéfiniment par la suite. A chaque extension du jeu, on peut acheter celle ci puis continuer à jouer. Bien que certains jeux à abonnement fassent payer une licence et un abonnement, ceux ci ne font payer que la licence. Il existe peu de jeu de ce genre, mais ceux qui s'y sont lancés ont connu un véritable succès. (Guild Wars par exemple)

Les jeux entièrement gratuits:

Les jeux basés par navigateur sont souvent entièrement gratuits. Les joueurs peuvent évoluer et jouer tout au long de la journée sans déboursier d'argent. Ce type de jeu est financé grâce à la publicité, et demande souvent à l'utilisateur des informations personnelles, comme l'âge, la profession, pour mieux cibler la publicité et ainsi en augmenter les revenus.

Gérer les joueurs :

Communauté:

Pour gérer la communauté, il faut engager ce qu'on appelle des community managers. Ils sont chargés de gérer la relation entre les équipes de développement et les joueurs. Ils doivent écouter les attentes des joueurs, leurs impressions, faire leur feedback aux développeurs et inversement, rapporter aux joueurs les nouveautés.

Ils sont également chargés de résoudre les conflits entre les joueurs, surveiller ce qu'il se dit sur le jeu dans les forums et sites annexes, favoriser l'essor des fans sites en fournissant des informations, donnant des interviews, réaliser des packs pour créer un site web aux couleurs du jeu. Les choses les plus infimes sont souvent les plus importantes. Il faut créer une véritable petite société autour du jeu. Les joueurs doivent avoir l'impression d'appartenir à une communauté non seulement dans le jeu, mais autour du jeu dans la vie réelle. Un jeu sans communauté est voué à l'échec.

Les game masters:

Chargés de résoudre les conflits et reporter les bugs dans le jeu aux développeurs, ils doivent être là en permanence, et répondre aux questions des joueurs dans le jeu. Ils doivent avoir une connaissance très poussée du jeu, afin de pouvoir répondre à toutes les questions. Ce sont souvent des anciens joueurs d'expérience qui sont recrutés à ce poste.

Dans les jeux internationaux, une bonne maîtrise des langues étrangères est requise, ainsi qu'une bonne diplomatie et une bonne expression écrite. Il faut être neutre, patient, incorruptible, technique, expérimenté. C'est un travail plus difficile qu'il n'y paraît.

Viser un public:

Plutôt mature, jeune ou mixte, le public visé par un jeu est très important. Il va diriger l'évolution du produit de sa création à sa sortie. Le style graphique, le background, le style de jeu, le contenu... Tout est défini autour d'une ambiance, une atmosphère créées pour un public particulier.

Analyser le marché :

Avant de se lancer dans la conception du jeu, une étape préalable est nécessaire. En effet, il ne faut pas se lancer dans un domaine déjà occupé par un autre jeu. Il faut donc d'abord faire une liste de ce qui a été fait, puis la comparer avec nos idées. Comparer nos idées avec les idées déjà réalisées.

Ainsi, nous pouvons récolter les informations, trouver de nouvelles idées, voir les idées qui n'ont pas marché et faire ainsi un concentré d'idées qui sera différent de ce qui existe.

Contrairement à ce que l'on pourrait penser, à un jeu complètement novateur a peu de chance de trouver le succès, il restera un jeu de niche. Avoir une bonne idée et un jeu complètement original ne suffisent pas à conquérir le marché. C'est pour cela que nous retrouvons souvent les mêmes mécanismes dans de nombreux jeux multi-joueurs.

La conception d'un tel jeu requiert des sommes considérables, ainsi on ne peut pas prendre de risques en sortant un jeu sorti des sentiers battus. Cependant, on peut créer l'originalité en prenant l'ensemble des concepts qui ont marché, en allant chercher parmi les succès dans le domaine. Concentrer les bonnes idées, et apporter quelques changements, améliorer et avoir des idées novatrices autour de valeurs sûres.

C'est pour cela que les jeux sont très proches entre eux, peu de choses différent mais à chaque fois, ils semblent être différents. Petit à petit, chaque jeu apporte une nouveauté et ainsi c'est l'ensemble des jeux multi-joueurs qui évolue.

Pour concevoir notre jeu, il faut partir d'une bonne idée et structurer l'ensemble des valeurs sûres dans le domaine autour de cette idée. Ainsi notre jeu aura toutes les chances de rencontrer le succès.

Communication et stratégie d'entrée sur le marché:

Avant, pendant et après la sortie du jeu, il faut communiquer avec les joueurs. Pendant toute la phase de développement, il faut les tenir au courant, les faire participer à l'évolution du jeu, livrer continuellement des images et des vidéos présentant l'avancée du jeu, sans pour autant montrer un type de gameplay pour ne pas décevoir les joueurs.

Le jeu n'étant pas fini à ce stade, il faut être vigilant dans la communication. On ne peut pas montrer au joueur une partie du jeu qui n'est pas terminée, ou il le prendra comme la version finale du jeu et elle sera gravée dans sa mémoire.

Il faut donc montrer une version qui aura été soigneusement analysée, et validée par une équipe de communication dédiée à cela. Le développement d'un tel jeu prenant plusieurs années, il faut savoir communiquer à petites doses, pour tenir le joueur en haleine et être sûr qu'il ne perde pas le jeu de vue pendant toute la phase de développement.

Les « community managers » doivent de leur côté parler avec les joueurs, obtenir des partenariats avec des sites généralistes et promouvoir leur jeu en trouvant des fans qui sont prêts à créer des sites dédiés au jeu. Pour être sûr que le jeu soit attendu du plus grand nombre, il faut se positionner partout.

Pour cela, il faut se placer là où l'on peut en fonction du budget: publicité sur les sites généralistes, mailing lists, spot télévision, radio. Les jeux multi-joueurs sont entrés dans les mœurs au cours de ces dernières années, une publicité à la télé sera de nos jours payante.

Le futur des jeux massivement multi-joueurs:

De tels jeux apportent que peu d'innovation, leur développement étant coûteux et lent, c'est un domaine qui avance doucement. Seulement, cela tend à changer car de plus en plus d'outils permettent de les développer. Ainsi, de nombreux jeux massivement multi-joueurs sortent, avec toute sorte de modèles commerciaux, et il serait probable que dans le futur, ils paraissent pour promouvoir un film, un événement, etc.

L'avènement de la 3D approche, et maintenant des technologies nous permettent de l'avoir à la maison. Il est évident qu'un bon nombre de jeux va se lancer sur ce marché pour proposer aux joueurs des jeux beaucoup plus immersifs.

De nouveaux jeux sortent également sur téléphone mobile. La puissance de ces unités permet de se géo-localiser et d'obtenir des informations, participer à un jeu en fonction de la position réelle de la personne. On peut penser dans le futur à un jeu à jouer chez soi, où l'on peut continuer sa progression dans la vie réelle en se déplaçant. La frontière entre le jeu et la vie est de plus en plus fine, et cela ne peut aller qu'en s'accroissant.

Quoi qu'il en soit, les jeux en ligne massivement multi-joueurs n'en sont qu'à leur début, et les années qui vont suivre nous apporteront beaucoup de nouveauté, et le mouvement ne risque pas de s'essouffler avant longtemps.

Glossaire

Termes utilisés et définition associée :

- MMOG : Anagramme de Massively Multiplayer Online Game, pour jeux en ligne massivement multi-joueurs, abrégé MMO.
- Hardware : Partie matérielle d'un poste, c'est la composition d'un ordinateur.
- Hack : Pratique qui consiste à accéder à des informations ou à modifier celles-ci en effectuant un crochet dans le programme
- Gameplay : Style de jeu, le gameplay représente la manière dont le joueur aborde la partie
- XNA : C'est une technologie Microsoft intégrée au Framework .NET, qui permet au développeur de réaliser un programme. Elle est spécialement dédiée au jeu et au multimédia et comporte une série d'outils permettant sa mise en place rapide.