Rapport personnel
Projet Web-BD

Elodie COCQ

Introduction

Tout d'abord, j'ai activement participé à la mise en place du cahier des charges ainsi que des espaces d'échange qui nous ont permis de nous mettre d'accord sur la base commune, ainsi que de communiquer efficacement tout au long de ce projet. J'étais initialement responsable de la partie moteur du jeu et donc de son implémentation côté serveur.

Principe du jeu

J'ai commencé par réfléchir aux différents éléments que nous voulions initialement pour le village. Nous avions donc deux types de ressources, des citoyens ainsi qu'une caserne, un centre de formation et des habitations. Il a alors fallu définir différentes variables, comme les quantités de ressources nécessaires à la construction mais aussi les fonctions d'évolution des ressources.

J'avais d'abord pensé à une évolution lente des ressources, qui se traduisait comme 1 ressource d'or toutes les 10 secondes par exemple. Mais, lorsque j'ai réfléchi à l'implémentation de la mise à jour des données sur des intervalles de temps, j'ai compris que cela ne fonctionnerait pas de la façon dont nous l'avions pensé. Si l'on simplifie le problème, il aurait alors fallu que la base de données contienne des ressources au type de format réel, mais que l'on n'en considère que leur valeur entière pour les utilisateurs par exemple. Ou alors, il aurait fallu garder la date de dernière mise à jour de chacune des ressources pour pouvoir déterminer le moment où la ressource augmente. Plutôt que de compliquer les algorithmes pour correspondre à cette première idée de vitesse d'évolution, j'ai préféré proposer au groupe un second modèle d'évolution. Il s'agissait alors d'une échelle plus rapide pour être en accord avec les précédents choix d'implémentation. L'unité de base était donc la seconde.

Au fur et à mesure de l'implémentation du jeu, j'ai pu constater qu'il manquait certains types de bâtiments pour le rendre plus intéressant. J'ai alors pu demander l'avis des autres membres du groupe quant à l'ajout de dépôts limitant les ressources ou encore des reliques qui augmentent l'intérêt du jeu par exemple.

De la même façon, les constantes d'évolution, de temps d'attente ou encore celles relatives aux coûts des différents éléments constituant le jeu ont été modifiées au cours du projet. En effet, il a fallu prendre le temps de réfléchir soigneusement à leurs valeurs pour faire en sorte que le jeu soit agréable, intéressant et puisse introduire une notion de challenge. L'ensemble de ces constantes ont été placées dans un fichier créé spécialement pour elles, ce qui permet de n'avoir à modifier qu'à un seul endroit ces dernières pour changer l'ensemble du jeu. L'utilisation d'un tel fichier a d'ailleurs été très utile pour les échanges avec le côté client puisque cela lui a permis de faciliter la récupération d'informations clefs du moteur.

Finalement, j'ai rédigé le contenu de la page d'aide pour que l'utilisateur puisse comprendre les ressorts du jeu, sans rentrer dans les détails d'implémentation. J'ai essayé de m'en tenir à ce qu'un nouvel utilisateur voudrait pouvoir savoir afin de jouer dans des conditions optimales.

Mise à jour des données

Le principal challenge de la mise à jour des données était déjà identifié dès le début dans les informations fournies. En effet, la mise à jour dans la base de données de la quantité de ressources possédée doit être faite au besoin, et non pas en continue, mais les joueurs doivent avoir l'impression que celle-ci est bien faîte en continue. Cela implique que le serveur doit être capable de mettre à jour l'ensemble des informations tout en restant fidèle au modèle défini, même sur un grand intervalle de temps. Alors que le côté client actualise ses données seconde après seconde, le serveur doit pouvoir atteindre les bonnes valeurs qu'importe la durée écoulée entre les deux mises à jour.

Dans un premier temps, il me semblait important que chaque action engagée par l'utilisateur possède son propre script. Encore une fois, cela permettait de simplifier leur appel pour le côté client. Chacun des scripts commence donc par inclure une mise à jour des données, pour que les informations traitées ensuite soient actualisées. Après avoir réalisé toutes les vérifications nécessaires, l'action peut être validée et enregistrée en base de données. Pour chaque action demandée, on donne un retour sur sa validité ou non, avec des explications claires des possibles problèmes rencontrés. Par exemple, si on s'est vu refuser la formation d'un soldat, on est capable de savoir pour quelle raison exactement: si c'est un manque de casernes ou un manque de ressources dans ce cas.

La difficulté de la mise à jour est de bien réfléchir à l'ensemble des éléments qui ont pu se produire pendant l'intervalle de temps sans mise à jour pour obtenir un algorithme correcte. La mise à jour des données avant chaque action permet de retirer certains cas compliqués pour justement engager la mise à jour. Ici, on sait qu'il n'y a pas eu de demande de construction entre deux intervalles de mises à jour par exemple. Mais d'autres cas viennent complexifier l'actualisation des données. Selon moi, les grands cas les plus difficiles à traiter sont les fins de construction de bâtiment. En effet, il faut prendre en compte l'état de stockage du village mais aussi le moment où la construction s'est achevée.

Pour parler d'un exemple plus concret, on peut prendre le cas de la fin de construction d'une grange. Lorsque l'utilisateur demande la construction d'une grange, le serveur lance une mise à jour, vérifie la validité de sa requête puis lance alors la construction. Si l'utilisateur n'effectue une nouvelle action qu'après la fin de la construction, il va falloir savoir si la capacité maximale de bois a été atteinte pendant cette période. Si c'est le cas, il va falloir déterminer si cela s'est produit avant l'apparition de la nouvelle grange ou après. Dans le premier cas, la production est donc bloquée et recommence seulement après que la nouvelle grange soit disponible. Dans le deuxième cas, il faudra bloquer la concentration de bois à la nouvelle capacité de stockage. De tels cas nécessitent donc d'y prêter attention avec des algorithmes robustes. La banque et l'habitation réagissent exactement de la même manière avec leur propre ressource respective: l'or et le citoyen.

Toutefois, ces cas limites n'existent pas pour les autres bâtiments puisqu'ils n'impactent pas les variables mises à jour automatiquement. Par exemple, la caserne n'influence pas l'évolution des ressources, et le soldat est formé à la demande de l'utilisateur.

Il est à noter que j'ai dû faire des compromis pour cette partie. En effet, le mineur était initialement formé de la même façon que le soldat et la relique. Mais, puisqu'il impactait l'évolution de l'une des ressources principales, l'or, il fallait l'intégrer aux cas limites exposés précédemment. Beaucoup trop de cas semblaient à traiter, avec de potentielles nouvelles variables à enregistrer en base de données. Il m'a semblé plus judicieux de rendre alors un mineur directement opérationnel. Il faut tout de même attendre un certain temps avant de pouvoir en former un nouveau, modélisant la préparation d'un nouvel accueil pour le centre de formation. Cela permettait aussi de varier le gameplay, et donc de rendre le jeu encore moins monotone.

Lien avec le côté client

Dans un premier temps, j'ai eu la possibilité de mettre en place beaucoup de scripts avant que le côté client soit prêt à incorporer le moteur du jeu. Dès lors, nous avons pu travailler en très forte corrélation, Paul et moi, pour mettre en place un jeu fonctionnel pour l'utilisateur. En effet, il a pu récupérer mes scripts et les utiliser côté client. Dès qu'un nouveau script était prêt, je l'en informais pour qu'il puisse commencer la transcription en javascript. Parfois même, j'étais amenée à directement lui expliquer le fonctionnement du script serveur et nous implémentions le côté client ensemble pour que l'on soit bien d'accord. On a alors pu discuter des modifications à apporter aux scripts PHP pour qu'ils soient utilisables encore plus facilement du côté client. Nous avons aussi pu vérifier que nous avions bien les mêmes fonctions d'évolution, et que, seconde après seconde ou dans un intervalle donné, les résultats étaient bien les mêmes. D'ailleurs, ces vérifications ont permis de déceler quelques erreurs dans mes algorithmes de mise à jour, qui ont pu être corrigées facilement. Nous avons aussi défini une grande partie de l'interface ensemble.

Fichiers et code

Dans l'ensemble des fichiers et du code que j'ai pu rédiger, j'ai essayé d'être la plus claire possible. Cela passe alors par la séparation des scripts en plusieurs fichiers distincts selon leur rôle, mais aussi par l'utilisation de commentaires et de noms de variables

cohérents. Il s'agit certes de bonnes pratiques que j'essaie habituellement de mettre en place, mais elles avaient encore plus de sens et de force au sein de ce projet: je savais que ces fichiers seraient directement utilisés par d'autres membres du groupe. J'ai donc redoublé d'efforts pour fournir des fichiers compréhensibles et utiles, principalement donc à Paul qui a développé la partie cliente, celle directement en lien avec la mienne. En ce qui concerne la séparation en plusieurs fichiers, j'aurais très bien pu me contenter de conditions pour certains. Par exemple, j'aurais pu regrouper l'ensemble des destructions de bâtiments dans un seul script dans lequel il aurait fallu fournir le type de bâtiment considéré. Cependant, puisque chaque bâtiment a des impacts différents sur le reste de l'environnement du village, ce fichier aurait été constitué de conditions pour chaque partie du code. Cela aurait donc rendu la lecture et la compréhension plus complexe, en plus de son utilisation.

De plus, j'ai pris garde aux accès à la base de données. En effet, j'ai fait attention à ce que chaque connexion à la base soit suivie d'une déconnexion propre, tant en cas de succès qu'en cas d'échec ou d'erreur.

Tests

Pour pouvoir tester les différents scripts au fur et à mesure, j'ai préparé une page de test, moteur.php, qui n'avait aucune prétention de design. J'ai donc pu vérifier que ce que j'implémentais fonctionnait, ou corriger si besoin. J'ai aussi mis en place plusieurs séries de tests plus complexes contenant différentes actions. Par exemple un relatif à la destruction:

- état initial du test : le village contient 2 casernes et 50 soldats,
- on lance la formation d'un soldat,
- avant que la formation soit terminée, on détruit une caserne,
- état final du test : on n'a plus qu'une seule caserne, toujours 50 soldats et la formation du soldat demandée a été annulée : le village est désormais dans l'incapacité de contenir plus de 50 soldats.

Effectuer différents tests de ce type me permettait d'être sûre que l'implémentation fonctionnait, au moins pour l'ensemble des cas que j'ai pu extraire.

Par contre, certains tests n'ont été possibles qu'une fois le client implémenté pour pouvoir voir apparaître des divergences de données ou quelconques autres problèmes d'algorithmes.

On peut noter que le moteur en php n'utilise que la notion de village, et jamais celle d'utilisateur. J'ai en effet souhaité conserver ces parties totalement dissociées pour ne pas cloisonner et restreindre le code. Cela ouvre à de plus grandes possibilités sans contraindre l'implémentation. En effet, j'imaginais qu'un utilisateur puisse avoir plusieurs villages par la suite, selon son expérience et son implication dans le jeu. L'implémentation actuelle aurait donc été valide sans avoir à faire de grosses modifications.

Pour avoir une démonstration plus complète, j'ai pu récupérer plusieurs villages provenant aussi des tests de jouabilité de mes camarades et préparer une base de données pré-remplie. On pourra alors mieux apprécier le classement, les attaques ou tout simplement l'ensemble des interactions entre les joueurs. On y trouve une simulation de litiges entre deux utilisateurs qui vont s'envoyer des mails et s'attaquer. Il s'agit de BaptMan et Ezio. Différents utilisateurs sont donc proposés dans cette base, dont les mots de passe sont

disponibles dans le fichier texte passwords fourni dans le dossier BDD. Les requêtes SQL sont placées dans le fichier BDD_cubeFighter.sql, lui aussi disponible dans le dossier BDD. L'importation de ces données via phpMyAdmin est normalement possible sans problème.

Autres contributions

Une fois le moteur côté serveur terminé, j'ai pu me rendre utile dans différentes autres parties. J'ai pu d'une part effectuer plusieurs tests sur l'inscription et la connexion, mais aussi sur ce qui était déjà prêts sur la page du jeu.

J'ai aussi continué à aider Paul dans sa partie qui n'était pas des moindre. J'ai par exemple implémenté la validité du nom du village saisi pour une première connexion. J'ai aussi pu proposer un design pour les bâtiments ou retoucher certaines images du projet. Avec l'apparition de la messagerie, nous avons eu l'idée d'améliorer l'attaque entre joueurs en envoyant des mails de rapport aux deux opposants.

Je me suis ensuite servie de l'implémentation du classement de Florian pour créer un historique des attaques relatives au joueur connecté. De cette façon, les deux pages pouvaient avoir un même rendu visuel pour conserver une certaine homogénéité.

Une fois l'ensemble des parties réuni, nous avons pu tester le projet dans sa globalité et apporter de nouvelles pistes d'améliorations ou des rapports de bugs. Cela a été très bénéfique pour le projet grâce aux personnes réactives du groupe.

Il me semble toutefois important de mentionner le fait que la page gestion de compte est arrivée tard. J'ai à peine pu la tester et le résultat obtenu n'est pas vraiment optimal: il reste encore plusieurs bugs, pour certains seulement visuels. Malheureusement, le temps imparti restant avant le rendu n'était pas suffisant pour reprendre proprement l'ensemble, nous avons alors tenté de corriger au mieux un maximum d'erreurs lorsque nous avons eu accès au code.

Conclusion

Ce projet m'a réellement permis de comprendre ce que signifiait côté serveur et côté client. Bien que j'avais pu en entendre parler, je ne mesurais pas ce que cela impliquait vis-à-vis de la gestion d'une base de données et du rendu sur navigateur pour un tel type de site. J'ai aussi pu consolider mes acquis sur les bases de données et l'html, mais aussi en apprendre davantage sur le javascript que je n'avais jusqu'alors que survolé. J'ai beaucoup aimé mettre en place le principe du jeu et l'étayer au fur et à mesure de mes implémentations.