DOSSIER

*Valentin MONGEY*

PROJET D’ISN 2013-2014

THE LEGEND OF THE QUEST

Sommaire

1. Le jeu
   1. Principes du jeu
   2. Pourquoi ce projet ?
   3. Répartition du travail
2. Détails de mes programmes
   1. Auberge
      1. Gestion de la vie
      2. Combat d’ivrognes
   2. Combat
      1. Joueur
      2. Ennemis
      3. Heal/tank
      4. Loot
   3. Attaques
   4. Compagnons
      1. Intégration en combat
      2. Trouver un compagnon
      3. Le compagnon et les quêtes
3. Graphismes
4. Conclusion
5. Le jeu

1.1) Principes du jeu

Notre jeu est donc un RPG « Role Playing Game ». Au début du jeu, au moment de la création de la partie, le joueur doit entrer un pseudonyme et choisir entre trois classes : hérétiques, paladin et assassin. Chaque classe a des caractéristiques de départ différentes ainsi que des sorts différents. Il peut alors choisir de passer ou non la cinématique de début. Le joueur commence le jeu dans la partie supérieure gauche de la carte. Proche de lui se trouve un village, qui permet de lancer la première quête dès que le joueur entre dans celui-ci. Le joueur doit donc suivre la suite de quêtes afin d’avancer dans l’histoire. Il va devoir faire de nombreux combats, qui seront de plus en plus dur au fur et à mesure de sa monté en niveau. Pour l’aider dans ses combats, le joueur peut avoir l’aide d’un compagnon, qui sera plus ou moins fort, en fonction du niveau du joueur mais aussi en fonction de sa « qualité ». En effet un esclave sera moins puissant qu’un paladin ou qu’un sorcier. Le joueur a aussi la possibilité de fabriquer des armes et armures : il possède un niveau en forgeage qui monte à chaque fabrication et lui permet de débloquer de nouvelles armes et armures passer certains paliers. La fabrication se fait à partir d’objets récupérés aléatoirement à la fin des combats. Il y a donc un système de vente que le joueur pourra utiliser pour faire de la place dans son inventaire tout en se faisant de l’argent. Il existe en effet plusieurs objets n’ayant pas d’utilité et le joueur devra trouver quels sont ces objets inutiles afin d’éviter de s’encombrer inutilement.

Sur la carte, des évènements aléatoires peuvent se déclencher, comme par exemple une agression de sorcier nous lançant des boules de feu et nous attaquant, ou encore un assassin nous proposant ses services contre quelques pièces d’or.

Plusieurs types de quêtes sont présents :

* Chasse aux monstres : il faut faire un certains nombres de combats pour réussir la quête.
* Transport : il faut transporter un objet d’un village à un autre.
* Récupération : trouver des objets sur certains monstres…

1.2) Pourquoi ce projet ?

Depuis le tout début, nous voulions faire un jeu mais au départ nous avions une autre idée de jeu en tête. En effet, dans un premier temps nous voulions faire un jeu de style « point and click », qui consiste à une suite de tableaux dans lesquels le joueur doit cliquer afin d’intéragir avec les décors. Ce projet avait deux grosses difficultés : la première étant le nombre important d’images à fournir et la seconde, la gestion de la position du clic de la souris et des positions des objets sur lesquels le joueur peut cliquer. Après plusieurs semaines de travail sur ce projet, nous nous sommes rendu compte qu’on n’avançait pas : nous n’arrivions à gérer les clics, alors que le jeu devait tourner autour de ce système… Puis pour se changer les idées, nous avons fait un petit jeu du style « RPG » et après avoir discuté ensemble, nous nous sommes rendu compte que ce projet était plus à notre porté tout en nous permettant d’avoir une plus grande liberté d’action. Nous avons donc continué le projet, tout d’abord sous la bibliothèque Tkinter. Cependant, nous étions limités au niveau de la gestion des images : nous avons donc décidé d’utiliser la bibliothèque Pygame, nous offrant plus de facilité à ce niveau. C’est ainsi que nous sommes arrivé à ce qui sera par la suite notre projet final.

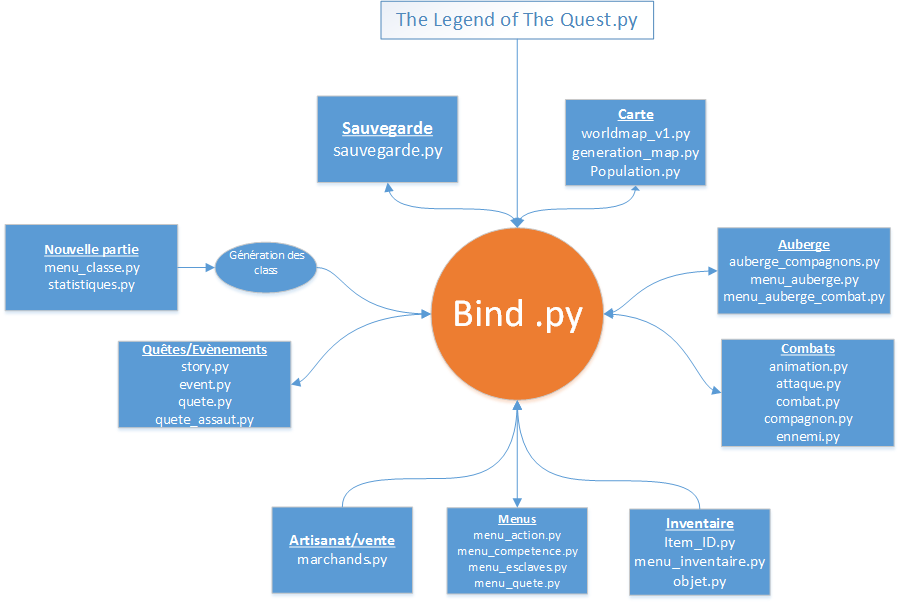
1.3) Répartition du travail

Depuis le tout début, nous nous sommes mis d’accord sur le travail que chacun devait faire. Mais pour éviter tous problèmes, nous avons mis en place une « règle » : on ne devait pas modifier les programmes des autres membres du groupe sans leur accord. Ainsi, nous évitions de perdre des données et nous réduisions les risques d’erreurs. Nous nous sommes donc occupés chacun de plusieurs parties du projet.

**Nathan**: il a principalement fait les images du jeu, notamment les monstres, les personnages contrôlés par le joueur, les bloques constituant la carte, etc. Il a aussi fait le système d’inventaire pouvant gérer les objets gagnés, l’utilisation de potion et l’équipement du joueur.

**Aurélien :** il est à l’origine du programme autour duquel tourne tout notre jeu : les  « binds ». Il s’agit des raccourcis sur le clavier, chaque touche renvoyant une commande spécifique en fonction de la page et principalement du titre de la page sur laquelle se trouve le joueur. Il a fait le programme de la génération de la carte, le système gérant la mini-map avec la position du joueur, l’entré dans les villages, le déclenchement des évènements aléatoires. Il a aussi mis en place le système pour les quêtes ainsi que la distribution des points de compétences du joueur, la sauvegarde et le chargement de la partie. Il a finalement fait la quête finale qui consiste à une bataille durant laquelle le joueur doit placer des hommes sur trois fronts différents, le joueur ayant un combat plus ou moins difficile en fonction du déroulement de la bataille.

**Valentin**: Je me suis occupé du système de l’auberge, du combat, des compagnons, des animations et de graphismes. Les détails sont ci-après.



1. Détails de mes programmes
   1. Auberge
      1. Gestion de la vie

Dans notre jeu, le personnage possède un certain nombre de points de vie qu’il va perdre lors de ces combats. Le joueur a donc plusieurs possibilités pour regagner des points de vie : il peut prendre une potion de soin, gagner un niveau qui lui permet de récupérer toute sa vie ou bien de trouver un village et d’aller à l’auberge.

Dans celle-ci, plusieurs choix s’offre à lui :

* il peut prendre une bière : cela lui coûte cinq pièces d’or et lui rends un point de vie,
* il peut prendre un steak : cela lui coûte vingt pièces d’or et lui rends cinq points de vie,
* il peut prendre une chambre : pour cinquante pièces d’or, il récupère toute sa vie.

L’auberge est donc un lieu important, surtout en début de partie lorsque le joueur n’a pas encore de potion : il doit donc se rendre souvent à l’auberge pour se soigner, ce qui permet au joueur de s’occuper des autres actions disponibles dans le village.

* + 1. Combat d’ivrognes

Les combats d’ivrognes sont un bon moyen de gagner de l’argent. Le joueur possède une renommée qui augmente lorsqu’il gagne un combat et qui diminue lorsqu’il en perd. Lorsque le joueur lance un combat, un nombre aléatoire de spectateur est choisi : plus la renommée du joueur est grande, plus le nombre de spectateur peut être important. Dans ce nombre de spectateur, le nombre de personnes pariant sur notre victoire est aussi tiré aléatoirement. Chaque parieur parie une somme d’argent aléatoire. Si le joueur gagne, il récupère toutes les mises des parieurs et s’il avait lui-même parié, il gagne le double de sa mise.

Le combat se déroule de la manière suivant : un texte apparait aléatoirement à l’écran puis disparait après un certain temps. Ce temps est définie en fonction de la renommée du joueur : en effet, plus elle est élevée, plus le texte disparait rapidement. Ce texte affiche l’attaque que le joueur doit faire pour pouvoir toucher l’adversaire, car le joueur a accès en permanence à trois attaques mais seulement celle affiché permet de toucher, les deux autres entrainant une attaque ratée. L’adversaire peut aussi rater ses attaques, cependant plus la renommée du joueur augmente, plus il se bat contre des adversaires puissants qui raterons de moins en moins leurs attaques, jusqu’aux plus puissants qui eux ne raterons plus aucune attaque : le joueur n’a pas le droit à l’erreur et le texte disparait rapidement !

La plus grosse difficulté a été de rendre l’auberge attractive, d’où la présence des combats d’ivrognes. Cependant, au début les combats étaient totalement aléatoires et ne présentaient pas de réels intérêts. J’ai donc décidé par la suite de refaire totalement les combats en m’inspirant du jeu de la bombe qu’on avait fait en cours. La nouvelle difficulté a été de gérer les binds étant donné qu’une seule attaque tirée aléatoirement à chaque tour peut infliger des dégâts. J’ai cherché de nombreuses façons de faire pour au final me rendre compte que le système tenait en quelques lignes de codes.

* 1. Combat
     1. Joueur

Le combat est l’un des éléments les plus importants de notre projet. En effet, la monté en puissance du personnage et la résolution des quêtes passent par les combats. Les combats sont déclenchés soit aléatoirement sur la world-map, soit lors d’évènements. Lors des combats, il peut y avoir de un à trois monstres et le joueur à plusieurs choix : il peut utiliser son attaque « basique » sur le monstre de son choix, cette attaque permet de faire des dégâts modérés et de réduire les temps de recharge des attaques spéciales et des potions. Chaque classe possède en effet deux attaques spéciales, l’une déclenche un temps de recharge de deux tours, l’autre de quatre tours. Ces attaques seront décrites plus loin dans le dossier. Le joueur a la possibilité d’utiliser deux potions : une potion de soin qui lui rend de la vie et une potion de soin de groupe, qui lui rend moins de vie mais permet de soigner aussi son compagnon. Il ne peut bien sur les utiliser uniquement si elles sont présentes dans l’inventaire du personnage. Finalement le joueur peut choisir d’essayer de fuir, si il rate les ennemis l’attaquent, si il réussit, le joueur est de retour sur la world-map. Si le joueur gagne le combat, il gagne des points d’expériences ainsi que des objets.

Le joueur possède plusieurs caractéristiques :

* Chance de critique : lorsque le joueur attaque, un nombre est tiré aléatoirement, s’il est plus petit ou égale à la chance de critique du joueur, alors l’attaque inflige le double de dégâts, sinon c’est une attaque normale,
* Chance d’esquiver : quand le joueur se fait attaquer, un nombre aléatoire est tiré, s’il est inférieur ou égale à la chance d’esquiver du joueur, alors il esquive l’attaque du monstre, sinon il subit l’attaque normalement,
* La constitution : c’est l’addition de l’amure apporté par l’équipement et la résistance naturelle du personnage. La constitution réduit directement les dégâts infligés par les monstres.

J’ai rencontré plusieurs difficultés, principalement au niveau des attaques spéciales, étant donné que chaque classe possède des attaques différentes. Ces difficultés seront détaillées dans le paragraphe dédié aux attaques. Une autre difficulté a été la gestion de l’attaque sur un monstre mort lorsqu’il y a plusieurs monstres dans le combat. Alors que le compagnon ne peut pas attaquer un monstre mort (et inversement, un monstre ne peut pas attaquer un compagnon mort), le joueur avait toujours la possibilité d’attaquer le monstre mort, puisque ses attaques sont gérées par les binds en fonction du nombre de monstre en début de combat. J’avais donc la potentialité de gérer l’emplacement des textes et l’attaque du joueur en fonction du nombre de monstre encore en vie. Cela représenté cependant un travail considérable car je devais envisager toute les possibilités (par exemple, s’il y trois monstres et que le monstre un est mort, le joueur ne peut attaquer que les monstres deux et trois). Après avoir discuté avec Aurélien et Nathan, on en est venu à la conclusion qu’il fallait laisser la possibilité au joueur d’attaquer un monstre mort, ce qui fait une « attaque dans le vent », le joueur perd donc une attaque inutilement, cela pousse donc le joueur à faire attention à ce qu’il fait. La dernière difficulté a été la gestion de la victoire : je n’ai pas eu une grosse difficulté à passer, mais plusieurs petites, notamment parce que la victoire se fait grâce à de nombreuses vérifications (la mort de tous les monstres, le gain de points d’expériences en fonction des monstres, gain de l’or, le loot, etc.). Il m’est donc arrivé parfois que le combat ne puisse plus se finir alors que tous les monstres étaient morts…

* + 1. Ennemis

Les ennemis sont tirés aléatoirement parmi une liste de monstres. Cette liste regroupe tous les monstres en leur attribuant à chacun toutes les caractéristiques les définissant, dont un ID. Cet ID permet de simplifier la création du monstre, on crée un monstre avec un ID spécifique et cela permet de créer un monstre avec les caractéristiques liés à cet ID. Il est important de noter que les monstres sont choisi certes aléatoirement, mais en fonction de la zone dans laquelle se trouve le joueur : en effet, s’il se trouve dans un désert, le joueur ne pourra combattre que des monstres du désert, pareillement pour les autres zones. Cette sélection est faites grâce au système d’ID. Les monstres possèdent des statistiques similaires au joueur mais ils ne possèdent pas de chance de critique : nous avons décidé de ne pas intégrer de chance de critique pour les monstres car nous trouvions frustrants de pouvoir perdre un combat sur un coup critique d’un monstre. Lorsque le joueur est avec un compagnon, les monstres ont une chance sur deux de taper le joueur et une chance sur deux de taper le compagnon, sauf cas spécial ou le compagnon et du type « tank ». Certains monstres peuvent aussi être du type « heal ». Ces deux types sont expliqués dans le paragraphe suivant.

* + 1. Heal/tank

Les personnages de type « heal » sont des personnages pouvant se soigner et soigner leurs compagnons. Se sont donc en général des sorciers, des mages, etc. Les classes de l’hérétique et du paladin possèdent des compétences de ce type :

* L’hérétique : il a un drain de vie, il inflige des dégâts tout en récupérant une partie de sa vie, en fonction des dégâts infligés.
* Le paladin : il possède un soin qui lui rend une certaine quantité de points de vie.

Certains compagnons ont donc la possibilité de se soigner ou de soigner le joueur. Cependant, ils ne peuvent soigner que si l’un des deux a perdu de la vie, il ne perdra pas en effet un tour pour soigner inutilement, il préfèrera taper. Les monstres fonctionnent aussi avec le même principe, ainsi s’il y a trois ennemis capables de soigner, ils vérifient s’il manque de la vie à l’un des monstres, si oui, alors le premier choisi entre soigner ou taper. Si il soigne, le deuxième monstre va vérifier si le monstre soigné à enfin toute sa vie, sinon, le deuxième monstre fait le même choix que le premier. Si finalement, le monstre soigné a toute sa vie, alors les autres monstres ne feront qu’attaquer. Ce système m’a permis de créer une intelligence artificielle, bien qu’aléatoire, qui ne fait pas choix inutile.

Les compagnons de type « tank » sont des compagnons massifs qui attirent les attaques des ennemis. Lorsque le joueur est accompagné d’un tank, les monstres n’ont plus une chance sur deux d’attaquer le joueur et pareil pour le compagnon : ils ont à présents une chance sur quatre d’attaquer le joueur et trois chances sur quatre d’attaquer le compagnon.

La plus grosse difficulté a été de rendre ces spécialités utiles sans qu’elles ne deviennent pour autant trop puissante.

2.2.4) Loot

Le loot correspond aux objets que le joueur reçoit lors de sa victoire en combat. Pour chaque monstre présent dans le combat, il y a une chance de gagner un objet. Ainsi, lorsqu’il y a trois monstres, le joueur peut gagner de zéros à trois objets. Le loot est géré par une fonction (Cf. annexe), puisqu’il existe plusieurs types d’objets plus ou moins importants, j’ai fait en sorte que certains objets apparaissent plus souvent que d’autre. Il existe aussi des objets qui n’ont pas d’utilité, si ce n’est d’être vendu pour gagner des pièces d’or. En plus de ces objets, le joueur a une chance sur cinq de gagner une potion de soin.

A la fin du combat, si l’inventaire du joueur est plein, il a alors le choix entre abandonner tous les objets gagnés lors du combat ou bien ouvrir son inventaire. S’il ouvre son inventaire, il peut alors choisir de supprimer les objets dans l’un des emplacements de l’inventaire et de récupérer le nouvel objet tout juste gagné ou alors passer cet objet et l’abandonner définitivement s’il ne l’intéresse pas. J’ai malheureusement eu beaucoup de mal à faire ce programme car, bien que je me sois occupé du loot, l’inventaire avait été codé plusieurs semaine avant par Nathan. La question c’est aussi posé à savoir qui devait s’occuper de ce programme : Nathan car c’est une partie de l’inventaire, ou moi car il s’agit de la fin des combats. Je me suis donc occupé de ce programme car il était vraiment relié au combat. J’ai cependant passé plusieurs heures à mettre en place le système de suppression des objets dans l’inventaire et la récupération du nouvel objet, car cela faisait intervenir aussi bien les binds que l’inventaire mais aussi le combat et les monstres. Je me suis donc heurté à une très grosse difficulté et je suis parvenu à la surmonter en avancement doucement et en corrigeant les erreurs une par une. Bien que ce système gère un cas particulier et rare, il démontre à quel point nous avons fait attention à tous les détails, aussi petits soient-ils, car ce sont ces détails qui donnent de la profondeur à un jeu.

* 1. Attaques

Chacune des trois classes possède deux attaques spéciales spécifiques, une avec qui déclenche un temps de recharge de deux tours et l’autre déclenche un temps de recharge de quatre tours. L’hérétique à un vol de vie qui inflige des dégâts tout en soignant le joueur et un météore qui inflige des dégâts à tous les monstres présents. Le paladin possède un soin et une attaque puissante qui inflige de gros dégâts. Enfin l’assassin a une attaque qui permet d’assommer un ennemi qui l’empêche d’attaquer durant ce tour et un assassinat qui n’inflige pas beaucoup de dégâts, cependant si l’ennemi attaqué a moins de 35% de sa vie, cela le tue directement, peu importe sa vie.

J’ai eu des difficultés sur plusieurs attaques : pour le vol de vie de l’hérétique, je voulais que le joueur soit soigné d’un montant inférieur aux dégâts infligés. Il fallait donc que je récupère le montant de ces dégâts et j’ai alors étais obligé de refaire le calcul des dégâts de façon à pouvoir les utiliser plus facilement. L’attaque assommée de l’assassin m’a posé un gros problème : comment faire en sorte que l’ennemi ciblé n’attaque pas pendant ce tour. Je voulais dans un premier temps que tout se fasse dans la fonction de cette attaque spéciale. Cependant, je me suis vu obligé de faire une vérification, dans les attaques des monstres, d’une variable vérifiant si le monstre est assommé ou non. Cette variable que j’ai nommée stunMonstre passe à la valeur 1 lorsque le monstre est assommé et ainsi il n’attaque pas. Finalement, l’attaque météore de l’hérétique permet de toucher tous les monstres présents mais ne vérifie pas s’il y a des monstres morts : par exemple, s’il y a trois monstres lors d’un combat, le monstre un est mort, l’attaque va quand même toucher les trois monstres, cependant alors que les monstres deux et trois vont prendre des dégâts, le monstre un subira une attaque dans le vent, comme expliqué précédemment dans la partie joueur du paragraphe sur le combat. Je pense que j’aurai pu faire en sorte que l’attaque ne touche que les monstres encore en vie, cela m’aurai cependant prit beaucoup de temps que je ne possédais alors pas.

* 1. Compagnons

2.4.1) Intégration en combat

Nous avions pendant plusieurs semaines abandonnées l’idée d’intégrer des compagnons. Cependant, le projet avançant très bien, on a décidé que les compagnons ajouteraient un gros plus à notre jeu et surtout aux combats. J’ai donc commencé à créer les compagnons alors que le combat était déjà finis. Il m’a fallu changer beaucoup de code, notamment sur les attaques des monstres qui devaient prendre en compte la présence ou non d’un compagnon, la position du personnage s’il est avec un compagnon qui est différente de lorsqu’il est seul, l’attaque de ce compagnon, la gestion de ses statistiques, de sa mort, de son type, etc. Cependant, le compagnon reprenait beaucoup de système déjà existant dans notre programme, à quelques différences. Cela m’a donc prit du temps pour l’intégrer et le rendre fonctionnel, je n’ai cependant pas eu de grosse difficulté lors de cette période, si ce n’est la gestion de la mort du compagnon en combat, avec des moments ou le compagnon ressuscité contre ma volonté… Finalement, ces problèmes venaient de la vérification de sa mort à de mauvais moments.

2.4.2) Trouver un compagnon

Nous voulions qu’il y ait plusieurs compagnons dans notre jeu, j’ai donc fait en sorte qu’on puisse les récupérer de plusieurs façons différentes. La première fut le marché aux esclaves qui est un classique dans les jeux-vidéo de style RPG. J’ai donc créé quatre types d’esclaves : un faible, un ordinaire, un puissant de type tank, un sorcier de type heal. Le sorcier coûtant plus d’argent que le faible. Le marchand d’esclaves propose trois esclaves aléatoires, si le joueur achète un des esclaves, le marchand ne propose plus que les deux autres jusqu’à ce qu’il ne propose plus d’esclave. Cependant, à partir de la première fois ou le joueur visite le marché aux esclaves, le « stock » d’esclaves se renouvelle tous les quatre combats. Le joueur peut aussi rencontrer aléatoirement sur la world-map un personnage qui propose ses services en tant que compagnons contre quelques pièces d’or. Si un compagnon vous accompagne, vous avez la possibilité le de tuer joyeusement avec votre nouveau compagnon de route. Il existe aussi trois compagnons uniques, un pour chaque zone, se trouvant dans l’auberge du village de leur zone. Ces compagnons puissants et coûteux sont unique car, s’ils meurent, il ne sera plus possible de les retrouver, leur mort est définitive.

2.4.3) Le compagnon et les quêtes

Le système de compagnon a permis de diversifier les quêtes disponibles. J’ai ainsi fait une quête durant laquelle le joueur se voit attribuer un compagnon scientifique et il doit l’emmener jusqu’à la zone contaminé et tuer des monstres contaminés avec ce scientifique. Cette quête c’est révélée plus difficile à faire que prévu : en effet, j’ai dû gérer le fait que le joueur possède déjà un compagnon (voir l’un des compagnons uniques). J’ai donc fait en sorte que notre compagnon soit « sauvegardé », le scientifique remplaçant ce compagnon et à la fin de la quête le joueur récupère son ancien compagnon. Il m’a aussi fallu empêcher le joueur de récupérer un autre compagnon qui remplacerait le scientifique et entraînerait des erreurs. Finalement, j’ai dû trouver une solution quant à la mort de se compagnon scientifique lorsque la quête est activé : j’ai donc fait en sorte que la quête soit suspendu jusqu’à ce que le joueur retourne rechercher le scientifique.

1. Graphismes

Je me suis occupé de certain graphisme dont dispose notre jeu. Principalement les animations des attaques en combat, le combat étant ma partie, il était plus judicieux que ce soit moi qui m’occupe des animations. J’ai fait mes animations à l’aide des programmes étudiés en cours qui nous permettez de faire se déplacer des boules ainsi que le module de mouvement d’un personnage. Les animations ne sont finalement qu’un déplacement d’images, mais cela m’a cependant demandé beaucoup de temps et j’aurais aimé avoir plus de temps pour pouvoir faire des animations plus détaillés et plus belles. J’ai aussi fait quelques images comme la page de chargement, le menu des statistiques du personnage, l’icône du jeu utilisée sur notre page web, le menu de la bataille finale, etc. Je n’ai eu ici pas de grande difficulté, cela m’a demandé simplement beaucoup de temps de travail.

1. Conclusion

Que dire si ce n’est que j’ai vraiment beaucoup aimé participer à ce projet. Nous avons eu une grande liberté et la possibilité de développer un jeu de la façon dont on le voulait, c’est la raison pour laquelle je me suis énormément investie dans ce projet, tout comme le groupe en général. Dès le début, nous avions un gros projet, pleins d’idées et d’envies… Finalement, même si nous n’avons pas fait tous ce que nous voulions (et fait des choses dont nous n’avions pas pensé), ce jeu est le reflet de tout le temps passé à travailler, à réfléchir, à imaginer, à rêver et à s’énerver. Car ce ne fut pas facile tous les jours, certaines difficultés nous ont semblées insurmontable alors que, le plus souvent, la réponse se trouvait sous notre nez ! Tout cela a créé une expérience vraiment unique qui, en plus de m’avoir appris de nombreuses choses sur le codage, m’a permis de passer de très bons moments en compagnie d’Aurélien et de Nathan.