Cahier des Charges
Universitus

Adrien Boitelle Robin Navarro Florian Correia
Romain Vergé Isak Viste Yann Blanchet

2018-2019

Contents

Partie 1

Étude des besoins

1.1 Objectifs

Pour mettre en place un guide du nouvel étudiant en informatique lui permettant de faire connaissances à la fois des technos/services disponibles mais aussi des espaces d'accueil/informations existants, nous proposons un jeu vidéo accessible par navigateur sur le site de l'Université de Bordeaux. Le tout devant intégrer une partie apprentissage des commandes de base de Linux, cela se fera par le biais des interactions utilisateur, tandis que la découverte du campus et des services délivrés par l'Université sera réalisée par l'accomplissement de quêtes de type RPG (Role-Playing Game).

Le public visé sera majoritairement composé des étudiants de première année en licence informatique, mais les fonctionnalités de base de l'application devront être suffisamment accessibles pour permettre aux étudiants des autres licences de s'y intéresser.

L'application s'inspire du projet Terminus développé par le MIT. Le joueur évoluera au sein du campus de Talence par le biais d'un terminal simulé. C'est dans ce terminal qu'il entrera des commandes similaires aux commandes utilisées sur les systèmes Unix afin d'interagir avec l'environnement et de résoudre des quêtes.

1.2 Besoins fonctionnels

- Un utilisateur doit pouvoir créer un nouveau compte, avec son pseudo et son mot de passe.
- Il doit pouvoir se déconnecter de son compte et s'y reconnecter à partir

de n'importe quel machine

- Il doit pouvoir supprimer son compte
- Un compte permet d'accéder au jeu sur navigateur
- L'utilisateur doit pouvoir interagir avec le jeu par le biais du clavier, en entrant des commandes
- Le jeu doit comporter des éléments rappelant le campus de Talence et son organisation
- Le jeu doit être construit de la même façon qu'une arborescence de fichiers Unix
- Les interactions utilisateur doivent être similaires aux interactions avec un terminal Unix

1.3 Besoins non fonctionnels

- Le jeu ne doit pas être trop gourmand en ressources
- Le lancement du jeu après sélection ou création du compte doit prendre moins de 2 secondes
- Le jeu doit comporter des quêtes optionnelles afin de capter l'attention de l'utilisateur
- Compléter la totalité des quêtes du jeu pour un utilisateur moyen doit prendre au minimum une demi-douzaine d'heures

Partie 2

Fonctionnement du jeu

2.1 Écran de jeu

LL'écran de jeu est extrêmement sobre afin d'habituer les utilisateurs aux terminaux qu'ils seront amenés à utiliser durant toute la durée de leur cursus. En conséquence, l'écran représente un terminal de type Unix et ne comporte que des lignes de texte qui s'affichent les unes après les autres en défilement vertical.

2.2 Terminal et usage des commandes

Le terminal inclus dans le jeu constitue la partie la plus importante de l'application. Le terminal simule un terminal Unix, à partir de là, de nombreuses commandes sont accessibles au joueur afin de lui permettre d'interagir avec son environnement. Il constitue le lien entre le joueur et le jeu.

2.3 Éléments du jeu

Le jeu est constitué des divers éléments listés ci-après

- des lieux, dans lequel l'avatar du joueur se déplace. Par exemple les arrêts de tram, la BU, le bâtiment A22...
- des objets, disposés dans les lieux, avec lesquels le joueur peut interagir, mais avec des comportements limités. Par exemple un formulaire, une clé, une porte...

- des êtres autonomes, situés dans les lieux, très similaires aux objets mais aux comportements plus variés et complexe. Le joueur interagit avec eux par le biais de commandes dédiées. Par exemple d'autres étudiants, des enseignants, des minions...

2.3.1 Lieux

Durant l'aventure, le joueur est amené à traverser l'université en long et en large, de ses plus sombres souterrains à ses étages les plus hauts. Concrètement, les divers environnements qu'il parcourt sont représentés sous forme de lieux.

Un lieu est défini par son **nom**, sa **description**, la **liste des lieux qui lui sont adjacents** et **la liste des objets, des pnjs, des ennemis et des quêtes** qu'il contient.

Afin d'interagir avec le lieu dans lequel il est situé, le joueur a à sa disposition une liste de commandes décrites ci-dessous.

- **cd** : C'est la commande de base, la plus simple et une des commandes que le joueur pourra utiliser dès le début du jeu. Elle permet de passer du lieu courant au lieu spécifié. Elle doit être entrée sous la forme : **cd lieu** où le paramètre lieu est le nom du lieu où le joueur souhaite se déplacer. Lorsque le joueur entre dans un nouveau lieu, la description de ce nouveau lieu s'affiche dans le terminal. Cette commande permet également une interaction avec l'inventaire, mais nous en parlerons dans la section appropriée.

- **ls** : Une commande qui permet de lister dans le terminal tous les objets, les pnjs, les ennemis potentiels et les lieux adjacents du lieu courant. L'inventaire apparaît également au joueur dans la liste. Elle doit être entrée sous la forme **ls** et ne prend aucun paramètre. Cette commande est aussi utilisable dans l'inventaire, mais ce point sera abordé dans la section appropriée.

- **pwd** : Une commande qui permet d'afficher dans le terminal une représentation visuelle du lieu courant. Elle doit être entrée sous la forme **pwd** et ne prend aucun paramètre.

Exemple :

Ici nous allons montrer un exemple typique d'interaction avec le lieu courant.

Le joueur se trouve dans le lieu **Salle de musique**. Il entre la commande **pwd** afin d'observer les lieux :

PlayerName@:/SalleDeMusique$ pwd

Une image du lieu s'affiche dans le terminal.

Puis il entre la commande **ls** afin de lister le contenu du lieu :

PlayerName@:/SalleDeMusique$ ls

La liste apparaît :

Couloir Placard

Il choisit de changer de lieu et d'aller dans le couloir :

PlayerName@:/SalleDeMusique$ cd couloir

Le joueur se trouve désormais dans le couloir ! On remarque que la localisation du joueur a changé, désormais Couloir est affiché à droite de Salle de Musique :

PlayerName@:/SalleDeMusique/Couloir$

2.3.2 Objets

Tout au long de son périple, le joueur est amené à trouver dans les lieux qu'il parcourt une multitude d'objets avec lesquels il pourra interagir de diverses manières. Ils peuvent apparaître sous forme de clés à prendre et utiliser, de portes à ouvrir, d'armes etc. Les objets sont généralement des entités simples que le joueur pourra éventuellement ramasser et placer dans son inventaire.

L'inventaire du joueur est extrêmement important, c'est là qu'il pourra entreposer tous les objets qu'il a acquis au cours du jeu. A l'aide des commandes qui lui sont accessibles, le joueur peut organiser son inventaire exactement comme il l'entend, créer des poches et des sous-poches afin de le trier et d'accéder rapidement aux éléments qu'il recherche.

De la même façon que pour les lieux, le joueur peut interagir avec les objets et l'inventaire avec les commandes suivantes :

- **cd** : Il s'agit de la même commande que celle permettant de se déplacer de lieux en lieux, elle est utilisée aussi afin de parcourir l'inventaire. Pour ouvrir l'inventaire depuis un lieu, il suffit de remplacer le paramètre lieu par le nom de l'inventaire. A partir de là, le joueur peut se déplacer librement entre ses poches comme dans un dossier unix, à l'aide de deux versions de la commande : **cd poche** avec le paramètre poche indiquant la poche désirée (accessible depuis la poche courante ou en entrant le chemin jusqu'à cette poche) ; ou bien **cd ..** pour revenir dans la poche parent de la poche courante. Pour sortir de l'inventaire, deux options sont offertes : se déplacer dans la poche nommée "Quitter" ou revenir à la racine puis entrer **cd ...** Cela aura pour effet de renvoyer le joueur dans le lieu où il se trouvait.

- **ls** : La version de cette commande accessible depuis l'inventaire fonctionne de la même manière que celle accessible depuis un lieu : en entrant **ls** dans l'inventaire, une liste des objets contenus dans la poche courante est affichée dans le terminal.

- **mkdir** : Cette commande permet de créer une poche dans la poche courante. Elle est surtout utile afin d'organiser son inventaire. Elle doit être entrée sous la forme **mkdir nom_poche** où "nom_poche" représente le nom de la poche que l'on souhaite créer.

- **mv** : Cette commande permet de déplacer un objet. L'objet peut être déplacé depuis le lieu courant dans l'inventaire, depuis une poche de l'inventaire au lieu courant, d'une poche à une autre poche, voire d'une poche ou d'un lieu à un objet/pnj dans certains cas particuliers (par exemple déplacer une clé d'une poche à une porte). L'usage de cette commande est sujet à des restrictions en fonction de l'objet ou de l'endroit où l'on souhaite placer l'objet (Une porte ne peut par exemple pas être déplacée, au contraire d'un livre). Elle doit être entrée sous la forme **mv objet destination** où "objet" représente l'objet que l'on souhaite déplacer et "destination" l'endroit où l'on souhaite déposer l'objet.

- **cp** : Cette commande permet de cloner un objet et de le placer à l'endroit voulu. Elle s'utilise de la même façon que la commande **mv**, à l'exception que cette commande agit sur une copie de l'objet sélectionné et non sur l'objet en lui-même. Cette commande est en revanche sujette

à des restrictions extrêmement fortes, seul un petit nombre d'objet peut être affecté. Elle doit être entrée sous la forme ¡ cp objet destination ¿ , les paramètres étant les même que pour la commande mv.

- **cat** : Cette commande permet d'afficher la description d'un objet. Elle doit être entrée sous la forme **cat objet** où "objet" représente l'objet que l'on souhaite examiner. Cette commande est aussi utilisable sur les ennemis et les pnjs, mais nous verrons cela dans la section dédiée.

- **rm** : Cette commande permet de détruire des objets, cependant des restrictions particulières sont appliquées (les objets principaux par exemple ne peuvent pas être détruits). Elle doit être entrée sous la forme **rm objet** où "objet" représente l'objet à détruire.

- **./** : La commande particulière d'exécution. Cette commande permet d'activer un objet (par exemple une potion). La plupart du temps les objets ne seront pas affectés par cette commande. Elle est entrée sous la forme **./objet** où objet représente l'objet à activer. Cette commande est aussi utilisée sur les pnjs et certaines entités.

2.3.3 Entités

Un être est composé des éléments suivants Au cours du jeu le joueur aura l'occasion d'interagir avec de nombreuses entités, que l'on appelera pnjs (Personnage Non Joueur). Certains seront d'une importance capitale et feront avancer l'histoire, d'autres peuvent donner des quêtes secondaires à accomplir afin d'obtenir des récompenses, quelques-uns pourront être combattus, enfin certains ne seront là que pour rendre le monde un peu plus vivants et n'offrir que quelques lignes de dialogue au joueur. Un pnj est défini par son **nom**, sa **description**, ses **caractéristiques** et ses **lignes de dialogue**. Afin d'interagir avec un pnj, les commandes suivantes peuvent être utilisées :

- **talk** : Cette commande permet de lancer le dialogue d'un pnj: Elle doit être entrée sous la forme : **talk pnj** où pnj correspond au nom du pnj auquel on souhaite dialoguer.

2.3.4 Quêtes

Composante incontournable de tout RPG qui se respecte, un système de quête sera implémenté dans le jeu. Tout au long de sa progression dans l'univers du jeu, le joueur sera guidé par un fil rouge sous forme d'une suite

de quêtes aux objectifs variés. La complétion d'une quête pourra débloquer de nombreuses récompenses pour le joueur, tel que de nouvelles commandes, de nouveaux objets ou de nouvelles quêtes.

Une quête est définie par les éléments suivants :

- **nom** : Le nom de la quête qui sera affichée
- **descriptipn** : La description de la quête qui sera affichée à son lancement
- **évènement de lancement** : Un évènement qui sera lancé à la création de la quête. Cela peut être par exemple la création d'un nouveau personnage nécessaire à la quête
- **évènement de résolution** : Un évènement qui sera lancé à la résolution de la quête. par exemple un personnage qui sera déplacé dans un autre lieu
- **sous-quêtes** : Une liste de sous-quêtes dont la complétion est nécessaire dans la complétion de la quêtes courante. Par exemple la quête **Vaincre Voldescript** qui nécessite de remplir la quête **Détruire les sept méthodes** et **Trouver le pouvoir de l'Amitié**
- **conditions** : Les conditions à remplir afin de compléter la quête. Ces conditions s'ajoutent aux sous-quêtes à compléter
- **quêtes suivantes** : Les quêtes débloquées par la complétion de la quête courante

La gestion des quêtes se fait de manière automatique, le joueur n'a pas à les gérer lui-même. Si toutes les conditions d'une quête sont remplies, la quête est automatiquement et immédiatement validée et ses évènements de résolution se lancent aussitôt.

2.3.5 Progression du joueur

Au fur et à mesure que le joueur complète des quêtes, il reçoit des récompenses et ses capacités augmentent. Au commencement, le joueur ne dispose que de quelques commandes et son inventaire est quasiment vide. Ses caractéristiques sont basses et les ennemis ne font qu'une bouchée de lui. Mais plus il avance dans le jeu, plus il devient puissant, plus il obtient des commandes étendant sa liberté d'action et plus il acquiert des objets qu'il pourra utiliser afin de l'aider ou de résoudre des situations

compliquées.

Pour obtenir de nouvelles commandes, le joueur doit accomplir des quêtes. Pour obtenir de nouveaux objets, le joueur peut accomplir des quêtes, vaincre des ennemis ou tout simplement parcourir le monde. Enfin pour augmenter ses caractéristiques il doit obtenir de l'expérience et monter de niveau en affrontant des ennemis.

Une liste de toutes les commandes et de tous les objets que le joueur pourra obtenir dans le jeu sera visibles dans une section ultérieure.

2.4 Univers

2.4.1 Environnement

La quasi-totalité du jeu se déroule au sein dd'une version altérée/parallèle e l'Université de Bordeaux. Chaque lieu visité par le joueur représente un lieu important du campus, tel que la bibliothèque, la cafétéria, le bâtiment A222 etc. Mais ce lieu est légèrement différent de la réalité, représenté sous un spectre fantastique.

L'énergie principal du monde où se déroule le jeu est l'informagie. La pratique de l'informagie est une discipline extrêmement complexe que l'on appelle l'Unixité. C'est au sein d'une université où l'on enseigne l'Unixité que se déroule le jeu. Le joueur incarne un nouvel étudiant qui débute sa formation à l'université, cependant l'année risque de ne pas être de tout repos et de nombreux évènements plus ou moins catastrophiques viendront perturber l'université.

Pour attiser l'intérêt du joueur, ol'nivers psera arsemé de références à une saga bien connue , Harry Potter. L'université s'inspirera ainsi grandement de l'école de magie Poudlard, avec ses escaliers mouvants, ses cachettes, ses créatures étranges et ses secrets.

- Occulus Rift avec le ChoixBot
- 4 grandes écoles selon le langage
  - Griffondor –>Java –>Javador / Grifforacle
  - Serpentard –>Python –>Serpythard
  - Poufsouffle –>PHP –>PHPouffle
  - Serdaigle –>C –>C'erdaigle

2.4.2 Récit

Pour accompagner le joueur, un fil rouge à suivre, représenté par une suite de quêtes principales, le guidera tout le long du jeu. Le joueur devra également évoluer dans un univers inspiré d'Harry Potter dans lequel il pourra apprendre les commandes sous forme de jeu. Voici l'univers dans lequel le joueur devra évoluer :

Dans une université où magie et technologie cohabitent, les étudiants sont formés depuis des décennies à l'usage d'un
pouvoir extrêmement rare appelé l'Unixité.

L'utilisateur est un des nouveaux étudiants, malheureusement le début de son année est marquée par la libération d'un processus daemon particulièrement redoutable, qui se met à répandre des processus zombies dans tout le campus. Il vous faudra parcourir l'université de fond en comble et acquérir les connaissances nécessaires afin de mettre un terme à cette terrible menace, et peut-être de découvrir qui se cache derrière tout ça.

Partie 3

Implémentation

3.1 Technologies

Le jeu sera fournit sous forme d'application web. Ce choix nous permet de toucher un publique plus large.
En effet, l'utilisateur n'a pas a télécharger quoi que ce soit, juste à se connecter à l'aide d'un navigateur web.
Ainsi l'utilisation sur les ordinateurs de la faculté en devient donc simplifié.

3.1.1 Front-End

Côté utilisateur, l'application est simplement du code HTML 5, avec du CSS 3.
Cela est en effet suffisant pour émuler un terminal, et permet de garder une application jouable sur des ordinateurs peu performant.
L'utilisation de javascript avec la norme ES6 est recommandée, car compatible avec la plus part des navigateurs.
(Depuis la version 13 pour Edge, 26 pour Firefox, 10 pour Safari et 39 pour Chrome)

La communication avec le serveur se fait à l'aide de WebSocket, suppporté par tous les navigateurs cités ci-dessus.

3.1.2 Back-End

Côté serveur, nous recommandons NodeJs, version 10, pour simplifier la gestion des conteneurs que nous verrons plus loin.

Conteneur

Pour chaque utilisateur, un conteneur Docker sera créé.
Cela nous permet d'isoler le joueur dans un environnement "bac-à-sable", dans lequel il pourra effectuer toute les actions permises par le jeu, sans affecter les autres joueurs.
Ainsi, même si le joueur trouve un moyen de "casser" le jeu, le conteneur pourra être relancé, et cela sans
affecter l'utilisation de l'application par les autre joueurs. Chaque joueur aura une version du jeu qui tournera dans le conteneur qui lui est dédié.

Nous recommandons le package Dockerode, disponible pour NodeJs, qui permet de simplifier les appels à l'API Docker.
L'espace mémoire utilisé par les conteneurs devra être limité en espace, afin d'éviter qu'un utilisateur ne bloque l'application en créant, par exemple, une "fork-bomb".

Jeu

Grâce à l'utilisation des conteneurs, le jeu peut être créé en utilisant n'importe quel langage.
Cependant, nous conseillons l'utilisation de langage haut niveau.
En effet, le jeu étant événementiel et sans graphismes à proprement parlé, nous n'avons pas besoin de performances très élevées.
Et des langages tel que python permettent une gestion de celles-ci de manière aisée.

Base de donnée

Un système d'authentification devra être mis en place, afin de pouvoir identifier l'utilisateur. Une base de donnée doit être utilisée pour stocker les couples identifiant, mot de passe. Les identifiants devront être uniques, et par soucis de sécurité les mots de passes ne devront pas être stocké en claire.

3.2 Structure

Le jeu en lui-même est une couche qui se place entre l'utilisateur et son container, à la fois pour le guider et pour filtrer ses entrées.
Toute entrée de l'utilisateur est traitée comme une commande shell plus ou

moins bridée. Aucune commande n'est permise tant qu'elle n'est pas déclarée comme une classe héritant de Command.

Le monde du jeu est modélisé comme étant une structure de fichiers dans lequel le joueur se déplace à l'aide de commandes.
Les quêtes sont stockées dans l'objet principal (Game), et déclenchent des évènements (Event) selon des conditions (Condition). Le joueur est confiné dans le dossier world/ mais peut tout de même intéragir avec celui-ci en parlant à des personnages disposant d'arbres de dialogue auxquels peuvent être attachés des évènements.

C'est en parlant à des personnages et en créant des entités que le joueur peut résoudre des quêtes et progresser dans le jeu.