

Light | Dark

**Pour l’obtention du Diplôme Universitaire de Technologies - Informatique**

**Rapport de projet**

**Année universitaire 2013 - 2014**

Réalisé par :   
**Mickaël CAMPMAS – Léo CASTERA– Jean-Joseph MARTY – Théo KRISZT**

Sous la direction de :   
**Dr. Abdelkader GOUAICH**

Remerciements

Sommaire

[1 Cahier des charges : Game design document - 2 -](#_Toc377558269)

[1.1 Introduction - 2 -](#_Toc377558270)

[1.2 Gameplay - 2 -](#_Toc377558271)

[1.2.1 Population Cible - 2 -](#_Toc377558272)

[1.2.2 Classification du jeu - 2 -](#_Toc377558273)

[1.2.3 Règles et structure d’action - 2 -](#_Toc377558274)

[1.2.4 Interactions entre le jour et le micro monde - 7 -](#_Toc377558275)

[1.3 Level Design - 8 -](#_Toc377558276)

[1.3.1 Création de scènes pour le jeu - 8 -](#_Toc377558277)

[1.3.2 Vérifier la cohérence du gameplay par le jeu de plateau - 11 -](#_Toc377558278)

[1.4 Scénario - 16 -](#_Toc377558279)

[1.5 Environnement graphique - 17 -](#_Toc377558280)

[1.6 Environnement sonore - 18 -](#_Toc377558281)

[2 Rapport technique - 19 -](#_Toc377558282)

[2.1 Conception - 19 -](#_Toc377558283)

[2.1.1 Technologies utilisés - 19 -](#_Toc377558284)

[2.2 Conception de Light|Dark - 19 -](#_Toc377558285)

[2.2.1 Les collisions et interactions physiques - 19 -](#_Toc377558286)

[2.2.2 La Shadow Form - 20 -](#_Toc377558287)

[2.2.3 La Light Form - 20 -](#_Toc377558288)

[2.2.4 Les monstres - 21 -](#_Toc377558289)

[2.2.5 Les animaux - 21 -](#_Toc377558290)

[2.3 Les Algorithmes spécifiques - 22 -](#_Toc377558291)

[2.3.1 Le système de coordonnées de libGDX - 22 -](#_Toc377558292)

[2.3.2 Les équations de conversion - 23 -](#_Toc377558293)

[2.3.3 La détection de collision - 25 -](#_Toc377558294)

[2.3.4 Le système de matrice et le système array - 26 -](#_Toc377558295)

[2.3.5 La modélisation des projectiles - 26 -](#_Toc377558296)

[2.3.6 L'intelligence artificielle - 26 -](#_Toc377558297)

[2.4 Architecture du logiciel - 27 -](#_Toc377558298)

[2.4.1 Programmation MVC - 27 -](#_Toc377558299)

[2.4.2 Hiérarchie des classes - 27 -](#_Toc377558300)

[2.4.3 Développement multiplateformes - 27 -](#_Toc377558301)

[2.5 Résultat - 27 -](#_Toc377558302)

[2.5.1 Présentation du jeu - 27 -](#_Toc377558303)

[2.5.2 Fonctionnalités du cahier des charges - 27 -](#_Toc377558304)

[2.5.3 Fonctionnalités abandonnées - 28 -](#_Toc377558305)

[2.6 Développement futur - 28 -](#_Toc377558306)

[2.6.1 Fonctionnalités supplémentaires - 28 -](#_Toc377558307)

[2.6.2 Conception artistique - 29 -](#_Toc377558308)

[2.6.3 Publication sur des boutiques en ligne - 29 -](#_Toc377558309)

[3 Manuel d’utilisation - 30 -](#_Toc377558310)

[3.1 Manuel d’installation - 30 -](#_Toc377558311)

[3.1.1 Installation sur PC - 30 -](#_Toc377558312)

[3.1.2 Installation sous Android - 30 -](#_Toc377558313)

[3.2 Manuel d’utilisation - 30 -](#_Toc377558314)

[3.2.1 Commencer une nouvelle partie - 30 -](#_Toc377558315)

[3.2.2 Continuer une partie - 30 -](#_Toc377558316)

[3.2.3 Se déplacer - 30 -](#_Toc377558317)

[3.2.4 Changer de forme - 31 -](#_Toc377558318)

[3.2.5 Attaquer - 31 -](#_Toc377558319)

[3.2.6 Se glisser dans une ombre - 31 -](#_Toc377558320)

[3.2.7 Contrôler un animal - 31 -](#_Toc377558321)

[4 Rapport d’activité - 32 -](#_Toc377558322)

[4.1 Méthode de développement - 32 -](#_Toc377558323)

[4.1.1 Mise en place de la méthode Agile Scrum - 32 -](#_Toc377558324)

[4.2 Planification - 32 -](#_Toc377558325)

[4.2.1 Planning prévu - 32 -](#_Toc377558326)

[4.2.2 Planning réels. - 33 -](#_Toc377558327)

[4.3 Méthode et outils de travail - 33 -](#_Toc377558328)

[4.3.1 Section modélisation - 33 -](#_Toc377558329)

[4.3.2 Section Codage - 33 -](#_Toc377558330)

[4.3.3 Section organisation. - 34 -](#_Toc377558331)

Table des figures

[Figure 1 - Schéma type d'un niveau - 4 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558348)

[Figure 2 - Niveau 1 - 8 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558349)

[Figure 3 - Niveau 2 - 9 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558350)

[Figure 4 - Niveau 3 - 9 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558351)

[Figure 5 - Niveau 4 - 10 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558352)

[Figure 6 - Niveau 5 - 10 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558353)

[Figure 7 - Niveau 6 - 11 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558354)

[Figure 8 - Présentation du concept au tuteur - 12 -](#_Toc377558355)

[Figure 9 - Début d'une session de jeu - 12 -](#_Toc377558356)

[Figure 10 - Découverte du grappin - 13 -](#_Toc377558357)

[Figure 11 - Niveau 2 et premier contact avec les ennemis - 13 -](#_Toc377558358)

[Figure 12 - Découverte de la Light Form - 14 -](#_Toc377558359)

[Figure 13 - Quatrième niveau - 14 -](#_Toc377558360)

[Figure 14 - Cinquième niveau et accès à l'orbe - 15 -](#_Toc377558361)

[Figure 15 - Dernier niveau et utilisation de l'orbe - 15 -](#_Toc377558362)

[Figure 16 - Shadow Form - 17 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558363)

[Figure 17 - Light Form - 17 -](file:///F:\Rapport%20De%20Projet%20-%20CAMPMAS%20CASTERA%20MARTY%20KRISZT.docx#_Toc377558364)

[Figure 1 Diagramme de cas d'utilisation de la forme d’ombre. - 20 -](#_Toc377558365)

[Figure 2 Représentation d'un vecteur symbolisant un déplacement vers le haut à droite - 22 -](#_Toc377558366)

[Figure 3Différences entre la méthode de positionnement de l'écran et de libGDX - 23 -](#_Toc377558367)

[Figure 4 Tir du personnage par rapport à la position de la souris - 24 -](#_Toc377558368)

Glossaire

**Gameplay**: Terme difficile à expliciter. Série de règles définissant la jouabilité d'un jeu, ce qui comprends les contrôles, l'intéraction entre le jeu et le joueur, les objectifs, le challenge, les récompenses...

**libGDX** : Terme difficile à expliciter. Série de règles définissant la jouabilité d'un jeu, ce qui comprends les contrôles, l'intéraction entre le jeu et le joueur, les objectifs, le challenge, les récompenses...

Introduction

Le but de ce projet est d’élaborer un jeu vidéo en deux dimensions, programmé en Java à l’aide de la librairie libGDX\*. Le jeu suivra des règles précises et essayera d’attirer le joueur et de lui proposer de se divertir tout en se concentrant.

Le gameplay\* du jeu proposera au joueur d’incarner un personnage dont les aptitudes seront influencées par le cycle jour/nuit à travers des phases de combat ou d’infiltration selon une orientation action/puzzle.

L’enjeu principal consistera à élaborer un gameplay\* innovant et indépendant du scénario et ce, malgré nos connaissances dans le domaine de la création de jeu vidéo encore peu développées. Nous avons dû suivre une méthode bien spécifique et propre à ce média. Il s’agit donc là, contrairement à la croyance populaire, de ne pas créer un jeu à partir d’un scénario mais d’élaborer un gameplay\* dont le scénario permet de justifier les actions entreprises dans ce dernier. Cette composante du jeu doit donc être créée avant et séparément du scénario.

Ce projet a pour but de nous initier aux mécanismes principaux de développement d’un jeu vidéo. Il nous montre aussi les aspects vitaux de la réalisation d’un projet en groupe.

Pour réaliser un gameplay\* complet et original, nous avons commencé par nous documenter sur des règles existant et méthodes de game designer\*. Aussi, nous nous sommes fortement inspirés de la mécanique principale des jeux de rôles, en particulier des premiers Zelda. Le concept peut se résumer aux déplacements dans les huit directions et une attaque. Contrairement à la majorité des jeux de rôles, nous n’avons pas intégré de moyens directs de défense. Cependant, notre personnage devait avoir une part importante de faiblesse afin d’accentuer la difficulté du jeu. Nous sommes donc orientés vers un gameplay\* en deux cœurs diamétralement opposé : l’un se basant sur l’action et l’autre se basant sur la réflexion.

Dans un premier temps nous parlerons du document de game design, fruit de cette réflexion, qui explicite ce que nous avons décidé de réaliser. Ensuite, nous expliquerons quels sont les mécanismes qui régissent le jeu, les méthodes, techniques et technologies mises en œuvre afin de réaliser ce projet. Enfin, nous présenterons notre expérience désormais enrichie en tant que développeurs de jeu vidéo grâce à l’utilisation de méthodes et d’outils de travail spécifiques.

# Cahier des charges : Game design document

## Introduction

Le cahier des charges d’un jeu vidéo est différent du cahier des charges d’un programme plus commun : le développement et l’analyse sont fondamentalement différents. Il est alors crucial de procéder, en premier lieu, à l’élaboration d’un document de game design : si la croyance populaire veut qu’un jeu vidéo se construise autour d’un scénario, il est tout autrement. En effet, les game designers doivent tout d’abord imaginer les règles, les mécanismes qui régiront le jeu, tout en faisant avancer le futur joueur au sein d’une boucle dites « OCR » : Objectif – Challenge - Récompense, qui rythme le level design, l’agencement de niveaux qui utilisent les outils mis à disposition par le game design. Le scénario viendra alors justifier le gameplay, pourquoi le joueur doit-il réaliser ces actions et dans quel but, puis vient alors le tour de l’environnement visuel et sonore.

Sans le respect de ces règles essentielles, le projet a de grandes chances de ne jamais voir le jour ou de devenir médiocre, simplement à cause des contraintes posées par le scénario ou de l’incohérence du gameplay\*. Il suffit par exemple de regarder certains jeux réalisés dans un but préventif, où le « scénario » a été réalisé avant de penser à de quelconques règles : le jeu n’est ni ludique ni agréable, il n’est même pas ce que l’on pourrait considérer comme un jeu.

Cette partie est alors construite en respectant l’ordre du processus créatif : game design document, level design, scénario, environnement visuel et sonore.

## Gameplay

### Population Cible

Le jeu s’adresse aux Core-Gamers\*, une catégorie de joueurs habitués aux jeux vidéo et ayant connu les premiers jeux vidéo ou du moins connaissant les grandes licences qui ont influencé ce milieu (notamment The Legend Of Zelda\*). L’âge de ces personnes se situerait donc entre 12 et 35 ans.

### Classification du jeu

Les principaux types sur lesquels le gameplay\* se repose sont action et puzzle, chacun étant relié à l’une des formes du protagoniste, mais nous reviendrons sur ce point plus en détail plus tard.

Un autre type, secondaire cette fois-ci, est l’aventure, qui découlerait de l’expérience du joueur à travers les différents niveaux qu’il parcourrait. L’inspiration vient très clairement des premiers épisodes de la série The Legend Of Zelda\*, dont la structure est facilement reconnaissable et s’adapte parfaitement au jeu que nous souhaitons réaliser.

### Règles et structure d’action

Les différentes actions se déroulant au cours de l’expérience de joueur se déroulent sur des écrans fixes en 2D avec vue de dessus, où le personnage rentre d’un point de l’écran et sort d’un autre.

Les grandes fonctions du gameplay\* se basent donc sur 2 cœurs, qui sont directement régis par les formes que peut porter le personnage :

1. Light Form : orientée action

2. Shadow Form : orientée infiltration, réflexion

Le jeu possède un système jour/nuit, qui influera sur la forme qu’arbore le personnage, et donc sur le gameplay\*. Le joueur ne pourra passer de l'un à l'autre de manière définitive, mais obtiendra tôt dans le jeu un orbe lui donnant la possibilité de passer temporairement d'une forme à l'autre en inversant le cycle en cours, avant de ramener le personnage à sa position initiale et de se désactiver pendant un certain temps. Le joueur sera ainsi amené à utiliser les 2 formes dans certaines scènes, le temps limité offert par l'orbe rajoutant un certain stress et du challenge, tout en apportant dynamisme et richesse à l’expérience de jeu.

#### Actions détachables

Les actions réalisables par l’agent seront très reproductibles, rythmées par les formes du personnage : attaques en Light Form, déplacement d’ombres en ombres et contrôles d’animaux en Shadow Form. Mais d’autres activités seront uniques, tel que l’obtention d’une clef ou le déverrouillage d’une porte, tout en restant un mécanisme que l’on retrouve plusieurs fois dans l’aventure

#### Règles constitutives - Dualité Shadow Form / Light Form

Les règles formelles qui entourent le jeu restent classiques, une perte totale de point de vie entraine la fin de la partie. Toutefois, la façon de jouer étant différente d’une forme à l’autre, elles s’adaptent à celle arborée par l’agent. L'alternance entre ces deux formes sera donc au cœur de gameplay du jeu. Apportant 2 expériences différentes, la forme d'ombre étant orienté puzzle et infiltration tandis que la forme de lumière étant plus simplement orientée action, leur utilisation croisée seront à la base de nombreuses énigmes dont le level design disposera. Le jeu repose sur la différence entre ces deux formes, à l’opposé l’une de l’autre, tant visuellement que sur la façon d’y jouer.

Ce sera donc l’orbe qui permettra l’alternance dynamique entre ces deux formes.

##### Shadow Form

À l'aube, des ombres apparaitront ça et là à l’écran, permettant au joueur, alors transformé en Shadow Form, de se déplacer jusqu'à sa destination. Pour cela, l'agent dispose d'un grappin lui permettant de se déplacer d'ombre en ombre ainsi que d'une capacité lui permettant, en se déplaçant à l'ombre d'un animal, de le contrôler. L'agent ne pourra toutefois pas se déplacer en dehors d'une ombre.

Les 3 verbes d’action sont donc:

* Se déplacer
* S’infiltrer (discrétion et reconnaissance)
* Contrôler (les animaux)

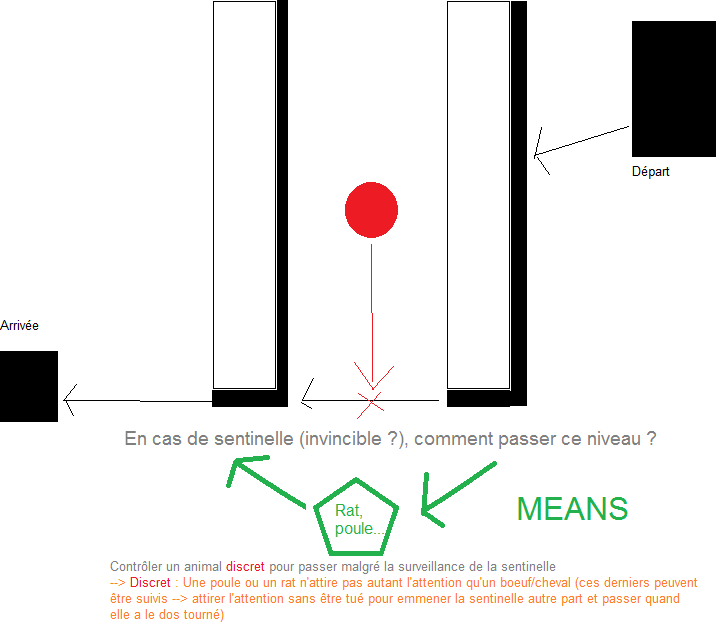
En Shadow Form, le personnage est faible, un coup d’un ennemi suffirait à le défaire. Le joueur peut donc être détecté et tué par des ennemis s’il se déplace d’une ombre à l’autre dans le champ de vision de l’ennemi (dont la taille peut différer selon la qualité de ce dernier). Le joueur doit donc réfléchir et faire preuve d’ingéniosité pour se déplacer d’un point A à un point B sans se faire repérer. Pour cela, le joueur peut avoir recours au contrôle d’animaux qui peuvent permettre au joueur de passer outre la surveillance d’une sentinelle.

Figure - Schéma type d'un niveau

Les animaux terrestres possèdent, en plus de pouvoir se déplacer, une capacité spéciale : un coq peut se mettre à chanter et donc attirer les ennemis à l’endroit où il se trouve, les obligeant à se déplacer. Une salamandre, elle, peut allumer une torche, le sanglier peut détruire une barrière en chargeant. Un poisson, lui, sera un « moyen de transport ». Comprenons par là qu’il ne permet que de se déplacer d’un point à un autre : le poisson remonte une rivière et s’arrête à un point précis.

La Shadow Form étant immatérielle, elle peut se jouer de certains obstacles : une grille, plongée dans la pénombre, pourra être traversée sans problèmes tandis que la Light Form en sera incapable. Elle ne pourra toutefois pas traverser certains passages où les ombres sont hors de portée ou quand le combat est inévitable.

##### Light Form

Lors de la première apparition de cette forme, le joueur devra trouver une épée afin de disposer des capacités offensives de cette forme. La Light Form est à l’opposé de la Shadow Form : résolument orienté action, elle permet au joueur d’attaquer ses ennemis d’une part, au corps à corps dans la direction voulue, ou de charger une attaque en restant appuyé une à deux secondes de plus. Cette dernière attaque permet de relâcher un projectile unidirectionnel, arrêté par tout obstacle, ennemi ou décors, mais possède une plus grande efficacité que l’attaque au corps à corps. Cette forme permettra au joueur de « décompresser », de se défouler après des énigmes complexes avec la Shadow Form, mais elle lui sera aussi complémentaire afin de déloger des ennemis dont la Shadow Form ne peut se débarrasser. Elle apporte ainsi une nouvelle dimension de gameplay\*.

Elle se définit donc en 2 verbes :

* Se déplacer
* Attaquer

#### Orbe et utilisation croisée des formes

Tôt dans l'histoire (vers la scène 5 environ), le joueur obtiendra un orbe lui donnant la possibilité de changer temporairement de cycle, à la manière des sables du temps\* de Prince Of Persia\*. À la suite de son utilisation, l'orbe se désactivera pendant un temps (40 secondes).

Le temps d'utilisation étant limité et son utilisation affublée d'un temps de rechargement, le joueur devra utiliser cet objet à bon escient. Le temps limité impliquera une dose de stress, qui dynamisera le gameplay et offrira un certain challenge.

Certaines scènes ne pourront pas être passées sans utiliser les deux formes. Le joueur devra donc œuvrer avec les propriétés de ces dernières pour surmonter certains obstacles. Une scène pourrait par exemple demander au joueur de passer la surveillance d'ennemis trop nombreux en Shadow Form pour ensuite arriver à une salle close qui nécessitera la Light Form pour se débarrasser d'un garde gênant puis pour abaisser un levier qui pourrait ouvrir une autre salle ou un barrage permettant à un poisson d'aller d'un point A à un point B.

#### Buts

Afin d’encourager le joueur à avancer, il est important de lui fournir des buts variés sans dénaturer le gameplay ou le perdre dans une multitude d’objectifs dans lequel il n’arrive plus à se retrouver : nous entrons ici dans la boucle OCR (Objectif, Challenge, Récompenses) que le joueur suit durant toute sa partie sans s’en forcément s’en rendre compte.

Ces objectifs se doivent d’être ludogènes et atteignables par l’agent en utilisant les moyens mis à sa disposition, mais aussi clairs et simples à comprendre. Un jeu aux mécanismes et buts trop flous risque de rebuter le joueur dès les 30 premières secondes, cette prise en main est d’ailleurs la séquence la plus importante de la session de jeu et va déterminer si le joueur continuera de jouer parce qu’il trouve le jeu amusant, ou s’il va simplement arrêter de jouer.

Lorsque le joueur arrive sur une scène de Light|Dark, son objectif premier est d’atteindre la sortie située autre part sur l’écran : nous avons ici un simple schéma d’un déplacement d’un point A à un point B. Toutefois, l’agent ne peut atteindre l’objectif immédiatement, car il doit suivre les règles explicitées ci-dessus et que certains obstacles apparaitront devant lui (ennemis, énigmes…). Les actions qu’il entreprend pour les surmonter constituent déjà de plus petits objectifs progressifs qu’il devra remplir pour en accomplir un plus grand. Pour cela, certaines scènes demanderont donc au joueur d’effectuer des actions spécifiques bien qu’il n’y ait pas forcément d’ordre précis. Parmi ces actions on retrouve le déplacement d’ombre en ombre, le fait d’éviter un ennemi en Shadow Form ou de l’attaquer en Light Form et enfin, contrôler un animal pour déclencher certains mécanismes (attirer l’attention d’un ennemi à un endroit précis, allumer une torche…).

Ces mécanismes constituent des retardataires qui ralentissent la progression du joueur afin qu’ils représentent une sorte de challenge pour le joueur. Certains d’entre eux ne seront franchissables qu’avec la Shadow Form en tirant parti des propriétés de cette dernière :

* Une barrière (grillage, portail) plongée dans la pénombre ne peut être franchie qu'avec cette forme, étant complètement immatérielle.
* Contrôler certains animaux permettra de briser un obstacle comme un rocher (sanglier) ou de traverser un point d'eau (poisson)

À contrario, certaines ombres étant trop éloignées, ou les animaux étant absents, seule la Light Form pourra passer, mais il devra alors combattre certains ennemis, attirés par la lumière qu’émet l’agent. Ces derniers représentent une menace tant en Light Form qu’en Shadow Form et est l’un des principaux obstacles, retardataire et challenge du jeu.

#### Boucle OCR + Means

Énoncée précédemment, la boule OCR, pour Objectif-Challenge-Récompense, est une boucle qui régit la progression du joueur dans le jeu. L’oubli d’un de ces éléments peut gravement nuire à la qualité du jeu final, et serait une grave erreur de game design.

Pour le jeu final, la boucle OCR de Light|Dark se présenterait comme ceci :

**Objectif** : Obtenir un nouveau fragment de l’orbe en notre possession se trouvant à la fin d’une zone.

**Challenge** : Énigmes, puzzles, obstacles, ennemis et retardataires. Potentiellement un monstre plus puissant gardant ledit fragment.

**Récompense :** Le joueur obtient un nouveau fragment lui offrant plus de temps d’utilisation de l’orbe. Elle donne aussi un indice sur la nouvelle zone à explorer, et donc l’accès à une nouvelle boucle OCR.

#### Actions vivantes, facteurs d’échecs

Toutefois, pour éviter que le joueur ne se sente pris dans un « couloir » et qu’il ait l’impression que toutes ses actions sont prévues, le jeu doit donner une certaine liberté à ses actions et donc lui permettre de manquer certains objectifs ou de les réaliser dans l’ordre qu’il le souhaite. C’est notamment la raison pour laquelle l’orbe existe et que le jeu ne possède pas un cycle jour/nuit régulier qui aurait pu freiner le joueur à de nombreuses reprises dans son avancée. Afin de ne pas trop simplifier le jeu, l’orbe reste affublé d’un temps de rechargement pour amener le joueur à réfléchir avant d’agir.

En plus des retardataires, le jeu possède un mécanisme simple qui encourage le joueur à effectuer ces actions correctement : la mort. En Light Form, elle se traduit par une perte de tous ses points de vie, en Shadow Form, être simplement détecté et touché par un ennemi suffit. Lorsque cela arrive, le personnage réapparait à sa position de départ dans la scène, ce qui signifie qu’il devra répéter les actions précédentes pour revenir à l’endroit où il se trouvait avant de mourir.

### Interactions entre le jour et le micro monde

#### Le système de sauvegarde

Afin que le joueur ne perde pas sa progression entre chaque session de jeu, le programme possède un simple système de sauvegarde qui mémorise automatiquement la position de départ de la dernière scène de départ traversée.

#### Contrôles

Le jeu dispose de 2 configurations différentes selon le support utilisé, mais seront simplement composé de touches pour le déplacement et d’un moyen et pointage et de clics pour les attaques et les actions.

#### Cameras

Vue de dessus, la camera est fixe et le personnage se déplace d’un bord à l’autre de l’écran, chaque scène dispose d’une caméra ou « point de vue » fixes et lorsque le personnage change de scène, il en est de même pour la caméra.

#### Personnages

Pour le prototype, le jeu possède 3 types de personnages :

* L’agent, possédant deux formes
* Les animaux, pouvant être contrôlés par l’agent
* Les ennemis, hostiles à l’agent et étant un des principaux obstacles

Plus tard, des PNJ\* seront ajoutés au jeu afin de renforcer la narration.

#### Feedback

Les feedbacks sont des retours sur actions de l’agent. Lorsqu’une action s’effectue à l’écran, un retour doit être apporté, il peut être d’ordre visuel et/ou sonore, afin d’avertir le joueur d’une action et de confirmer que l’action a bien été réalisée. Pour Light | Dark, ces feedbacks concernent :

* Les dégâts reçus par le personnage : lorsque celui-ci subit une attaque, son apparence clignote et est repoussée.
* Lorsque l’orbe est utilisé, son apparence est grisée jusqu’à son rechargement
* Et d’autres retours visuels et sonores sur les actions des acteurs (attaque, changer d’ombre…)

#### Assistance

Afin de ne pas perdre le joueur durant ses sessions de jeu et de lui faire comprendre les mécaniques du jeu, une assistance, explicite ou non, doit accompagner le joueur dans son expérience. Elle peut aussi se manifester par l’architecture des niveaux : c’est l’approche que nous avons choisie pour ce prototype, nous y reviendrons dans le level design.

Au-delà du projet, des panneaux d’indications et des menus contextuels pour les dialogues permettront d’orienter le joueur et de renforcer la narration.

#### Menus

Les menus du jeu restent assez classiques. On retrouve donc les traditionnels menus de démarrage, permettant de commencer une nouvelle partie ou d’en continuer une précédente, et le menu pause, utilisable durant une partie.

## Level Design

### Création de scènes pour le jeu

Le level design constitue l’ensemble des scènes dans lequel l’agent progresse, il tire parti des outils créés et mis à disposition par le game design.

Pour ce prototype qui est composé des premiers niveaux d’apprentissage, plus souvent appelés tutoriel\*, nous avons choisi l’approche qu’Edmure McMillen présente dans Indie Game : The Movie\* : chacun des premiers niveaux est volontairement simpliste et sert à exploiter une seule mécanique. Ceci permet au joueur d’assimiler pleinement cette technique avant de passer à des niveaux qui incluront cette dernière avec d’autres mécaniques qui auront elles aussi profité d’un niveau semblable.

|  |  |
| --- | --- |
| Pour ce premier niveau, nous avons choisi d’apprendre au joueur le premier des mouvements de la Shadow Form : le déplacement d’ombre en ombre. Le niveau est simplement composé d’ombres sur lesquelles il doit se déplacer et s’agripper.  Ce niveau doit aussi lui permettre de connaitre la longueur du grappin grâce au positionnement d’ombre parfois situé trop loin du personnage et qui amènera le joueur à trouver un chemin alternatif. | Figure - Niveau 1 |
| Le deuxième niveau met le joueur pour la première fois face à des ennemis hostiles, qui l’attaqueront immédiatement s’il venait à changer d’ombre devant eux.  Si le chemin le plus court mène donc irrémédiablement à l’échec, un chemin plus court l’amène vers un animal passant devant l’ennemi. Il peut donc contrôler cet animal pour passer devant l’ennemi sans être repéré et arriver à la sortie. | Figure - Niveau 2 |
| Pour ce troisième niveau, un ennemi bloque l’accès à la sortie : il faudra donc trouver un moyen pour l’en déloger.  Le joueur peut apercevoir un coq à proximité de cet ennemi, et grâce à l’expérience acquise au niveau précédent, il sait qu’il peut contrôler cet animal pour progresser.  C’est en effectuant cette action qu’il apprendra qu’il peut utiliser une compétence propre à l’animal (dans le cas ici présent, chanter), qui peut attirer l’ennemi et donc dégager l’entrée. | Figure - Niveau 3 |
| En arrivant dans le quatrième niveau, la nuit est tombée sur le micro monde\* et le personnage change de forme.  Désormais en Light Form, le personnage peut se déplacer à sa guise sur la carte et possède une nouvelle jauge de santé. Il peut alors se déplacer, en évitant l’attention de son garde, récupérer une épée dans le coffre qui pourra lui permettre d’enfin se défendre contre ses ennemis, dont celui qui gardait l’arme en question. | Figure - Niveau 4 |
| Toujours en Light Form à l’entrée du cinquième niveau, trois ennemis aperçoivent de loin le personnage et se mettre à l’attaquer : le joueur comprend alors qu’il est fortement visible sous cette forme.  En se débarrassant de ces ennemis, il peut accéder au coffre que ces derniers gardaient et récupère l’orbe. | Figure - Niveau 5 |
| Enfin, le dernier niveau tire parti de tous les mécanismes appris grâce aux niveaux précédents. Le personnage arrive en Shadow Form et doit allumer deux brasiers pour déverrouiller la porte. Pour cela, il peut se déplacer derrière la grille, utiliser l’orbe pour déclencher la Light Form, éliminer les ennemis, abaisser le levier pour ouvrir le passage à la salamandre que le joueur pourra contrôler en revenant en Shadow Form pour allumer les deux brasiers à proximité de la porte. | Figure - Niveau 6 |

### Vérifier la cohérence du gameplay par le jeu de plateau

Une bonne pratique pour vérifier la cohérence et la fiabilité du gameplay\* et du level design consiste à adapter le jeu et les niveaux en jeu de plateau. En faisant essayer ce jeu réalisé avec du papier, pièces de jeu d’échecs et dessins à une tierce personne, on peut vérifier que les mécanismes sont facilement assimilés et compréhensibles.

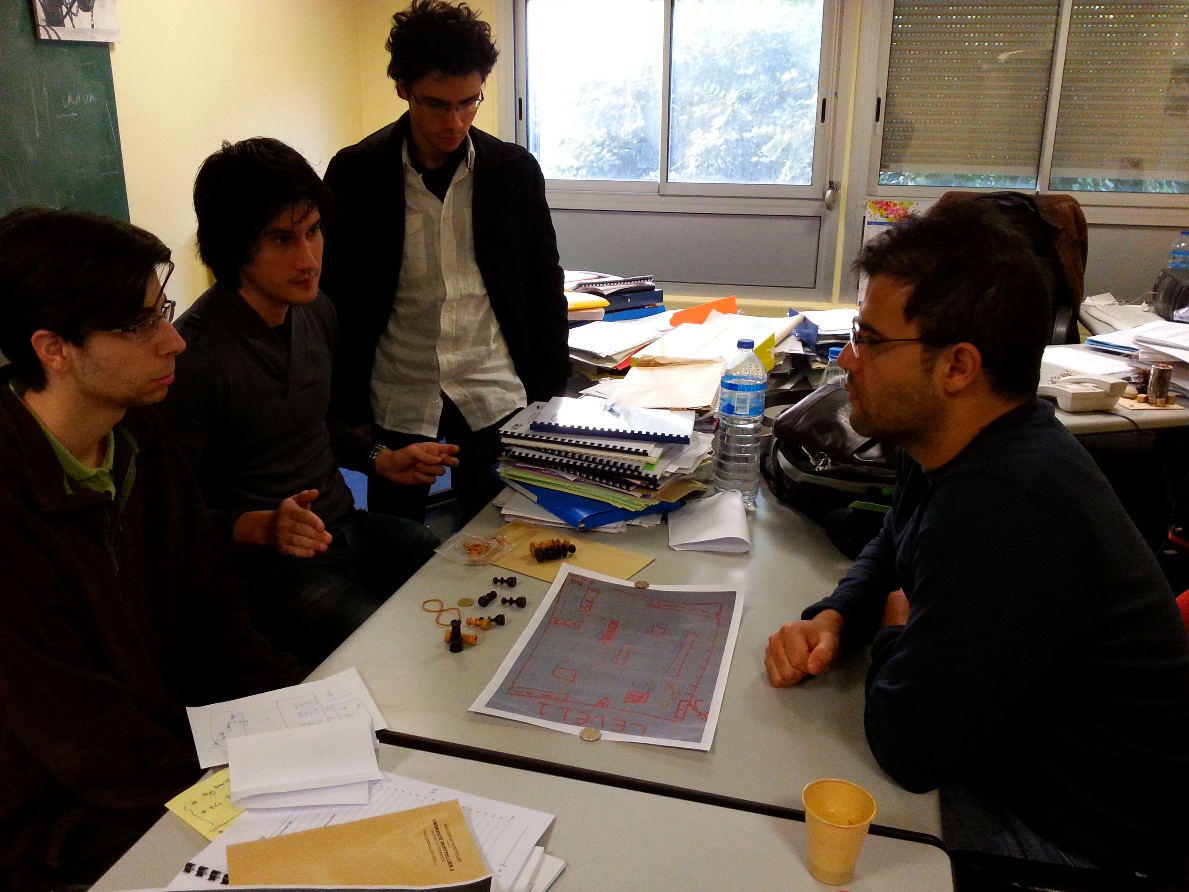


Figure - Présentation du concept au tuteur

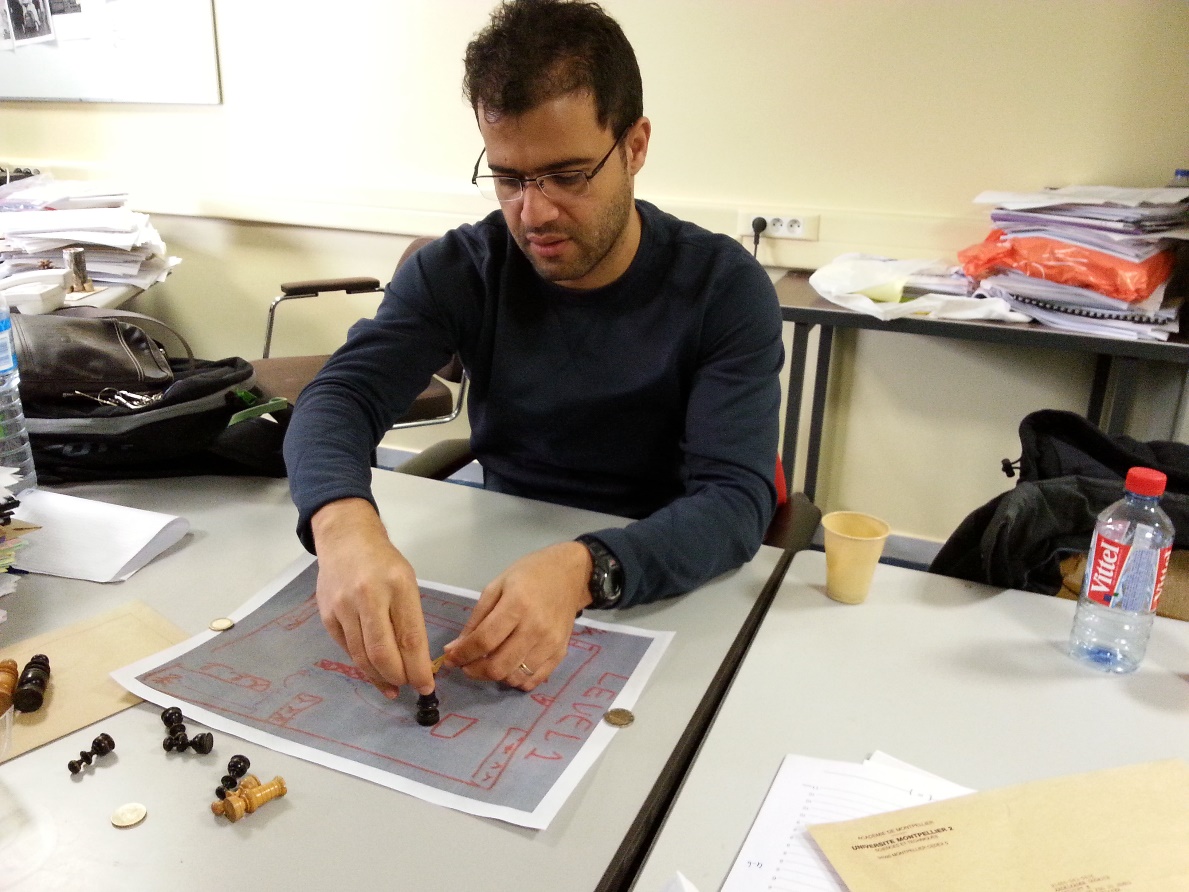


Figure - Début d'une session de jeu

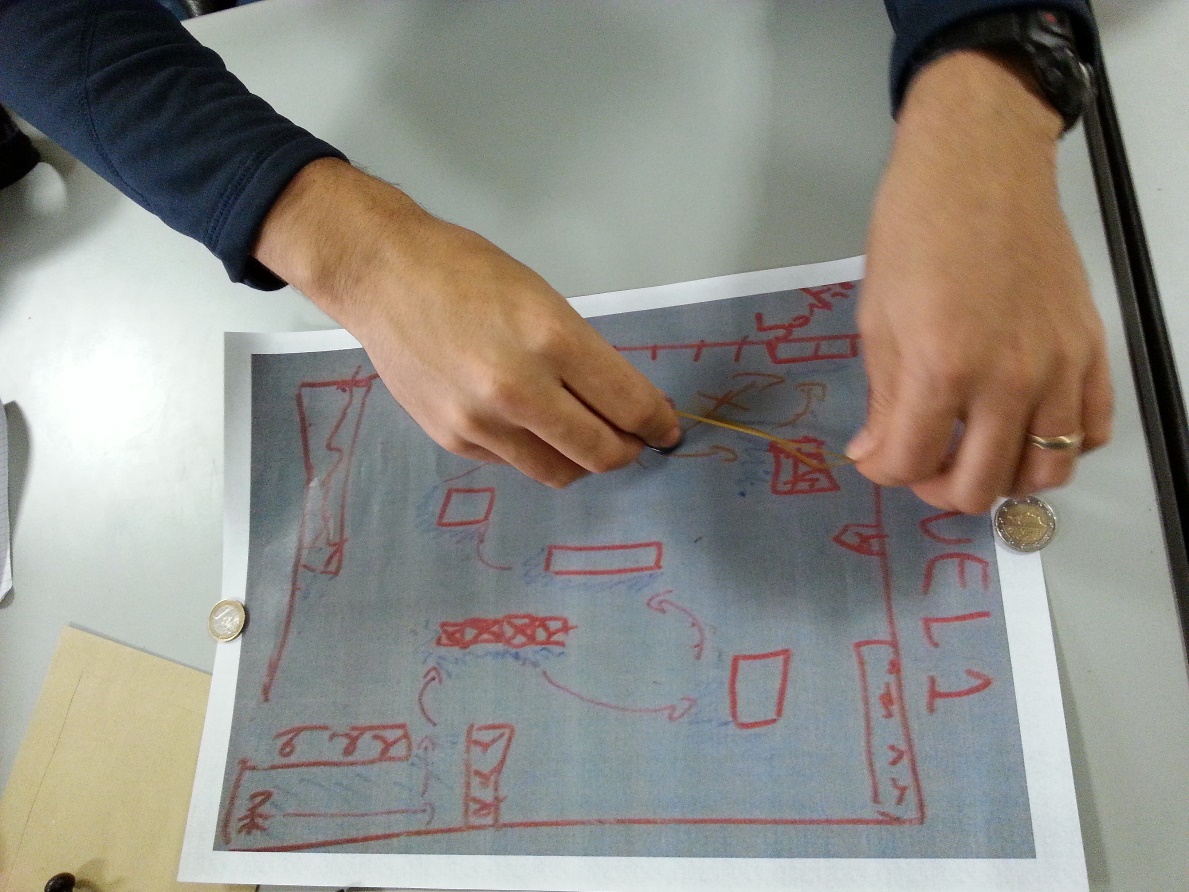


Figure - Découverte du grappin

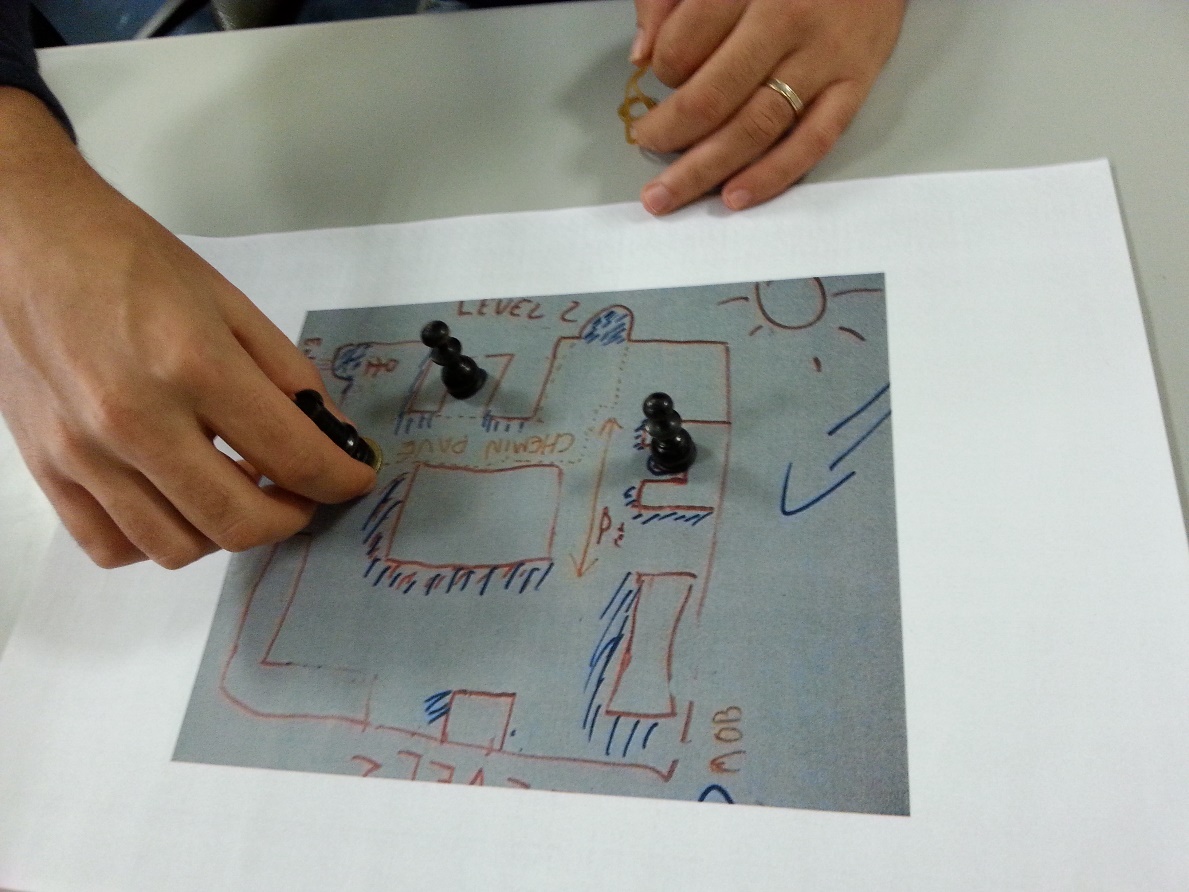


Figure - Niveau 2 et premier contact avec les ennemis

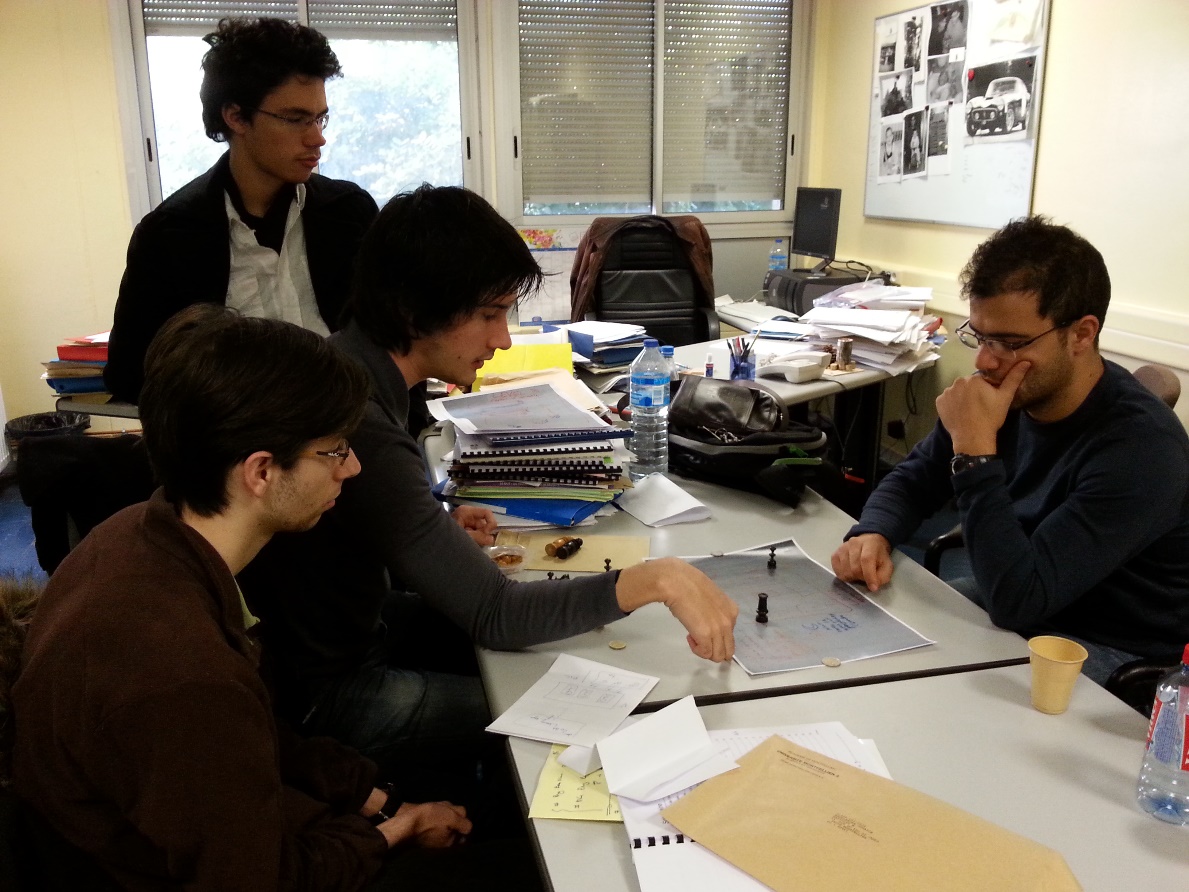


Figure - Découverte de la Light Form

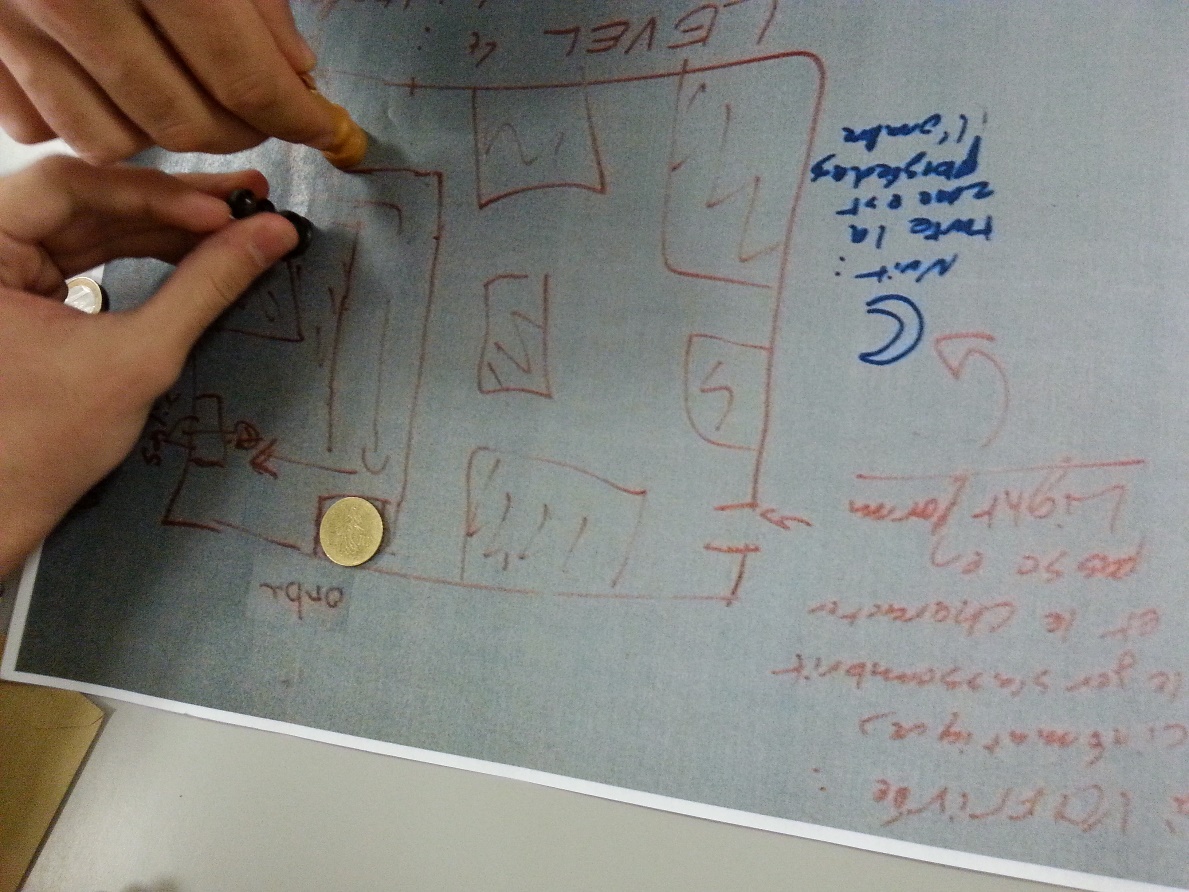


Figure - Quatrième niveau

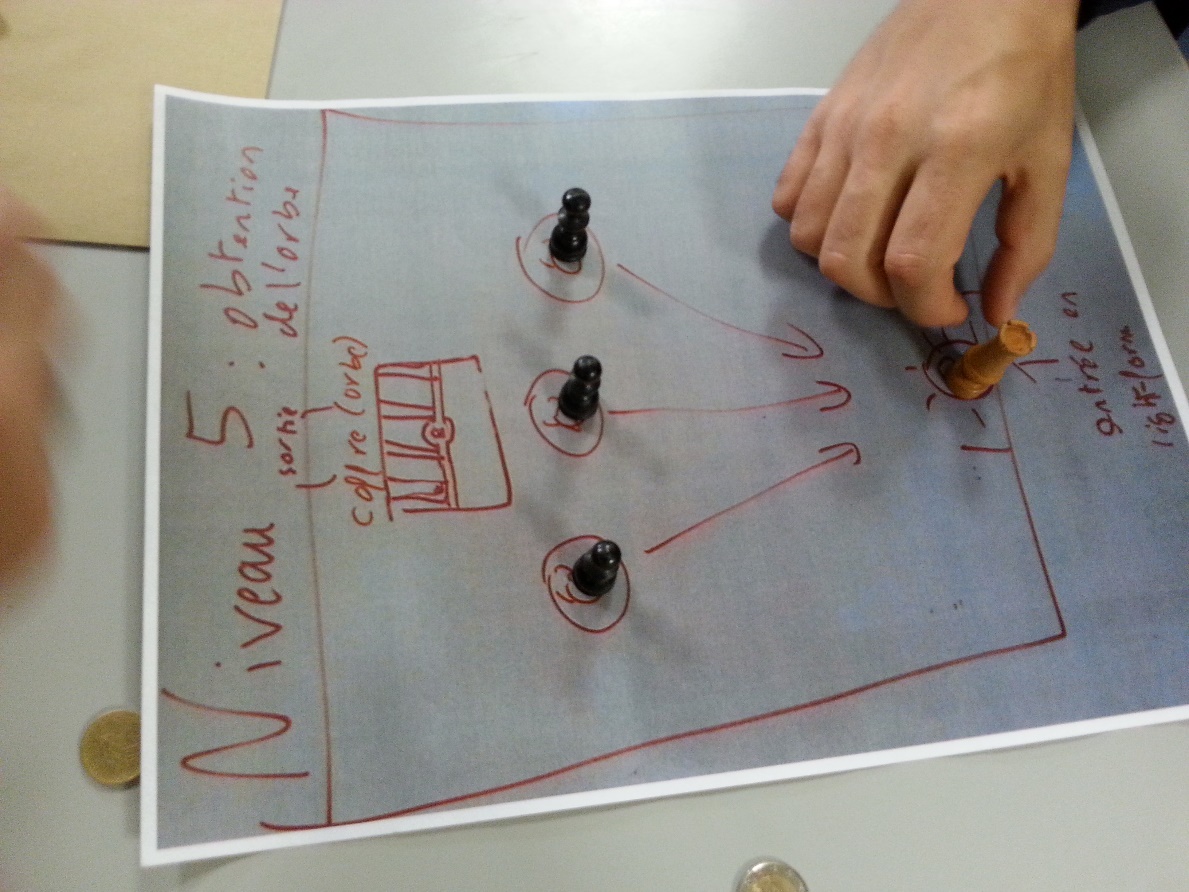


Figure - Cinquième niveau et accès à l'orbe



Figure - Dernier niveau et utilisation de l'orbe

## Scénario

Comme énoncé précédemment, le scénario se doit d’être créé après avoir imaginé le gameplay. Le scénario permet donc de justifier les éléments de ce dernier et non de le créer, un jeu vidéo peut d’ailleurs exister avec un scénario des plus simples voir sans scénario, l’expérience de jeu n’en sera pas altérée.

Le scénario peut toutefois aider non seulement à captiver l’attention du joueur et l’amener à ses objectifs, mais aussi à imaginer les scènes qui composent le level design, ce dernier utilisant les outils mis à disposition par le game design.

Pour Light|Dark, nous avons choisi un scénario simpliste, mais permettant de justifier la totalité des éléments de gameplay, tout en fournissant un background\* crédible pour l’intégration des différents acteurs et la progression du joueur.

« Light | Dark met en scène les Nocten, peuple humanoïde vénérant l’orbe de lune, seul garant de l’équilibre entre la nuit et le jour, la lumière et les ténèbres. À son contact, le corps de ses gardiens a changé et s’adapte au moment de la journée, leur permettant de se cacher du reste du monde : de jour, ils arborent une forme lumineuse, éblouissante et de nuit, ils sont des ombres parfaitement invisibles.

Vous incarnez Ekhan, un Nocten dont l’horloge biologique est inversée par rapport à celle de ses congénères, et ce depuis sa naissance, ce qui lui a valu d’être rejeté par son propre peuple et d’être le premier accusé lorsque l’orbe disparait après avoir été brisée.

Banni puis chassé par les Nocten, il se retrouve ainsi seul dans un monde désormais instable où l’alternance entre le jour et la nuit semble complètement soumise au hasard. Dans sa fuite, Ekhan trouvera un fragment de l'artefact perdu par son peuple. En suivant les traces laissées par celui qui semble être le véritable coupable, Ekhan s'engagera dans une quête pour restaurer la relique sacrée et ainsi rétablir l’équilibre du monde. »

L'objectif principal du jeu sera ainsi de récupérer les tessons de l'orbe, disséminé çà et là par le voleur. À chaque fois que le joueur se rapprochera d'un fragment, il devra se confronter à un boss, par exemple un ennemi corrompu par l'essence de l'orbe. Une fois vaincu, le joueur obtient un nouveau fragment, réduisant le temps de rechargement de l'orbe tout en donnant un indice sur la position du voleur, et donc d'un nouveau fragment. Le dernier fragment le mènera irrémédiablement vers cet ennemi afin d'en comprendre les motivations, mais aussi de découvrir la véritable nature du personnage : pourquoi est-il le seul de sa race à être inversé ?

Le scénario ci-dessus permet non seulement de fournir une raison à la présence des deux formes, de l’orbe, mais permet aussi de fournir au joueur un but plus significatif que de simplement récupérer des objets se trouvant à la fin de donjons. L’écriture d’un tel scénario n’a à aucun moment modifié le document de game design et permet déjà d’entrevoir le cheminement entrepris par le level design.

## Environnement graphique

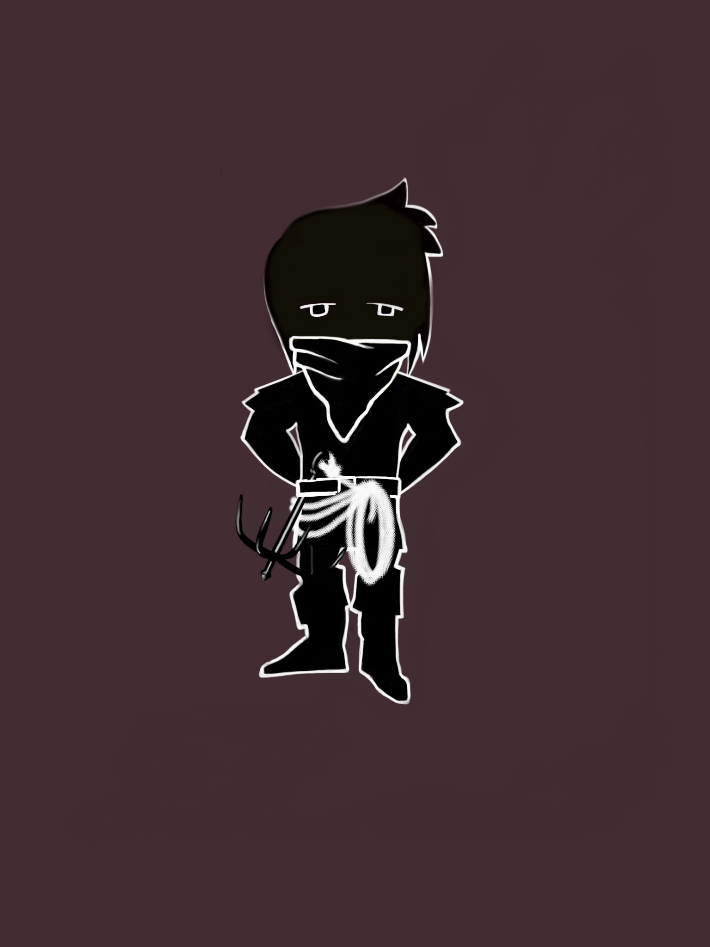
Nous avons choisi pour le prototype de nous servir de banques d’images libres de droits pour concevoir les décors et les ennemis. L’orientation du style de dessin se rapproche donc de ceux utilisés dans de nombreux RPG\* en deux dimensions : le medieval-fantasy.

Figure - Shadow Form

Figure - Light Form

En ce qui concerne le personnage principal, ses formes et ses animations, nous avons toutefois tenu à demander les services d’un artiste-plasticien. Cette personne s’est aussi proposée pour réaliser le reste de l’aspect graphique du jeu au-delà du projet, le temps lui manquant durant la période de développement du prototype pour réaliser le reste des éléments du jeu.

L’une des principales difficultés pour cette tâche est la communication avec le graphiste : les corps de métier étant complètement différents, les hésitations et incompréhensions sont fréquentes. La première chose à effectuer est donc de bien convenir des éléments voulus, du style de dessin, des inspirations possibles, de l’allure générale du personnage et des éléments que l’on souhaiterait apparaitre sur celui-ci.

Dans le cas présent, il s’agit d’avoir une influence médiévale, d’un effet dessiné bien présent et de faire figurer des armes et outils sur chacune des formes : épée pour la Light Form, grappin pour la Shadow Form. Le scénario peut aussi aider le graphiste à trouver les éléments de dessins que nous aurions pu oublier, grâce au passif du personnage, l’environnement dans lequel il évolue, etc.

Un contact fréquent avec celui-ci reste aussi primordial afin de s’assurer que le résultat obtenu correspond bien à nos attentes ou de le corriger avant qu’il ne soit trop tard.

Enfin, les animations sont réalisées par simple enchainement de trois images successives dont un état de début, un état de transition et un état final.

## Environnement sonore

Pour l’environnement sonore, nous avons aussi choisi de nous servir de sons et musiques libres de droits, mais nous pourrons aussi produire nous-mêmes certains bruitages (épée, bruits de pas…).

Nous suivrons aussi les conseils d’un compositeur afin de réaliser une ambiance sonore plus crédible et adaptée.

# Rapport technique

## Conception

### Technologies utilisés

Le langage JAVA est le langage qui nous est demandé par le cahier des charges. De plus ce langage créé en 1995, est particulièrement portable ce qui est un atout dans la fabrication de produit multiplateforme. La programmation-objet est omniprésente sous JAVA ce qui permet généralement de simplifier la relecture du code. Enfin, cette popularité dont JAVA bénéficie explique la documentation particulièrement riche en explication et en exemples.

La bibliothèque libGDX est une liste de fonctionnalités graphiques qui permettent de piloter la librairie OpenGL. L'avantage principal de libGDX réside en la simplification de la programmation graphique. Tout comme le JAVA, libGDX est compatible sur plusieurs plates-formes. Bien qu'au début nous devions utiliser GAME qui est un moteur de jeu développé par le LIRMM nous avons choisis libGDX pour sa diversité en fonctionnalité et sa légèreté.

## Conception de Light|Dark

### Les collisions et interactions physiques

Dans le micro monde du jeu il existe différents types de cases. En effet, le joueur ne peut passer au travers des obstacles ou ne pas subir les inconvénients d'une marre de boue. De plus, notre game design document précise que le joueur doit pouvoir se déplacer dans une zone d'ombre s'il est en forme d’ombre.

#### Les cases bloquantes

Ce type case, comme son nom, l'indique empêche le joueur de passer au travers. Ces cases représentent un obstacle.

Nous modélisons la gestion des obstacles dans le contrôleur du joueur, ainsi que dans les contrôleurs des différents acteurs tels que les monstres, les animaux et les projectiles. De cette façon le micro monde de Light|Dark peut imposer des contraintes sur ses différents acteurs en agissant directement sur leur comportement.

#### Les cases ombres

Ces cases propres à la Shadow Form permettent au joueur de s’infiltrer dans le niveau tout en le protégeant. Les monstres ne peuvent effectivement pas détecter le joueur en forme d’ombre si celui-ci se trouve dans une case de type ombre. Contrairement aux monstres, les animaux peuvent se déplacer dans les cases ombres.

#### Les cases à friction

Ces cases ont une propriété mitigée puisqu’elle se situe sur la frontière entre le refus absolu et la pleine acceptation. La plupart imposent un changement notable dans le déplacement de l’acteur.

### La Shadow Form

Cette forme représente la forme faible du joueur. Elle se caractérise par l'impossibilité de se trouver au contact de la lumière et par des moyens spéciaux de déplacement tel que le grappin. Une des spécificités de la Shadow Form est la possibilité de contrôler un animal, et ce, même si ce dernier se déplace.

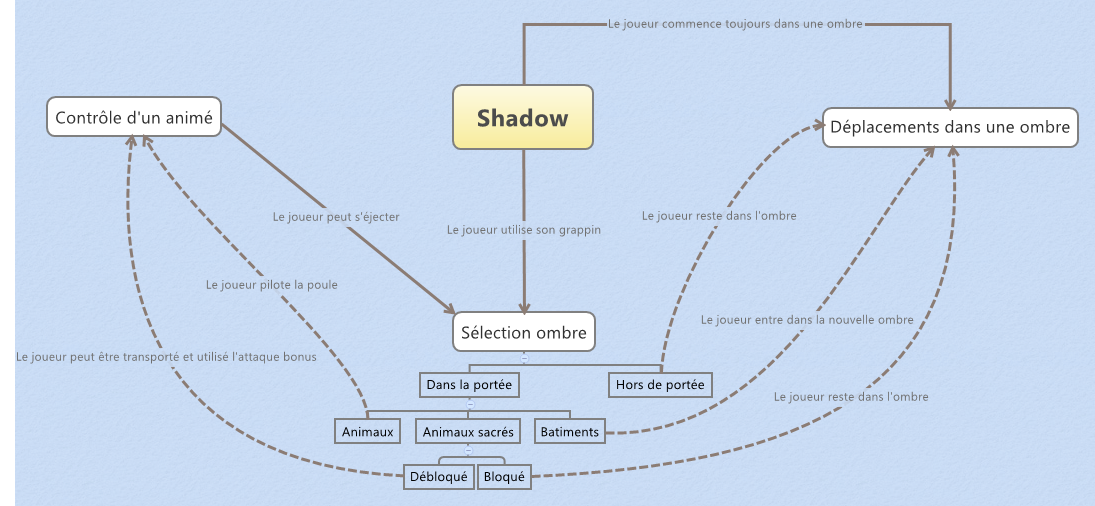


Figure Diagramme de cas d'utilisation de la forme d’ombre.

Ce diagramme montre bien toutes les possibilités de cette forme. Comme indiqué, le joueur ne peut pas sélectionner une ombre ou un animé (un animal) qui est hors de portée. Cependant, il est possible que le joueur se retrouve dans une situation où il est positionné dans la lumière et c'est à ce moment que le programme redirige le joueur dans une ombre. Cette fonction est importante, car les montres ne peuvent pas voir le joueur en forme d’ombre que lorsqu'il traverse une zone de lumière.

### La Light Form

Cette forme est simple puisqu’elle contient les déplacements de la Shadow Form sans les limitations ce cette dernière. De plus, cette forme est particulièrement forte avec ses deux types d’attaques différentes. En effet, l’épée permet une attaque au corps à corps et l’attaque distante permet de toucher un monstre jusqu’à une certaine portée.

[diagramme d’activité]

Cette forme n’est pas la plus importante du game play puisqu’elle ne permet uniquement que de débloquer certains niveaux par la victoire du joueur sur les monstres.

### Les monstres

Cette catégorie d’acteurs, elle la seule qui puisse interagir sur la vie du joueur. Les monstres sont particulièrement sensibles à l’obscurité et ils ne peuvent traverser dans laquelle ils ne peuvent pas voir. Ils peuvent être détruits par le joueur si celui-ci les tue avec un animal ou avec ses moyens de défense en Light Form.

[diagramme use case]

Les monstres se déplacent de manière linéaire en suivant un parcours prédéfini qu’ils exécutent en boucle. Dans le cas où le joueur est détecté par ces derniers, ils peuvent sortir de leurs parcours pour attaquer le joueur. Après s’ils sont toujours en vie, ils rejoignent leurs parcours pour le continuer.

### Les animaux

Les animaux sont complètement passifs par rapport au joueur qui peut les contrôler, mais pas les tuer. Les animaux suivent un parcours prédéfini qu’ils répètent en boucle.

[diagramme d’activité]

Lors d’une sortie de leur parcours, l’intelligence artificielle de l’animal leur permet de revenir sur leur parcours et de le continuer indéfiniment.

## Les Algorithmes spécifiques

### Le système de coordonnées de libGDX

LibGDX n’utilise pas un système de coordonnées par pixel bien que ce dernier est historiquement universel dans le monde des librairies graphiques et dans le monde des moteurs de jeux. Le système de coordonnées proposé par libGDX est assez générique. En effet, libGDX gère l’affichage par un système de flottant (float). Ce genre de nombre permet le calcul des nombres rationnels sur une grande précision ce qui permet de proposer un système universel basé sur des unités virtuelles. Ainsi, le développeur décide sur combien d'unités il définit son écran, puis se positionne en fonction du système de coordonnées qu’il a défini. L’avantage indéniable que ce système apporte est son indépendance par rapport aux dimensions de l'écran.

De plus, un point se définit sous libGDX par un vecteur. La programmation graphique s'en trouve complètement simplifie notamment dans les calculs d'angle et de collision.

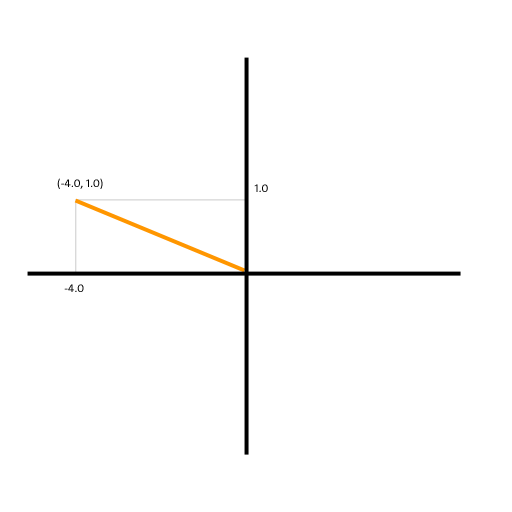


Figure Représentation d'un vecteur symbolisant un déplacement vers le haut à droite

### Les équations de conversion

#### Passage des pixels à la matrice de cases

Les entrées étant toujours historiquement codé dans le système de coordonnées pixel, il nous faut une fonction de conversion pour passer d'un système à l'autre.

① **float** posX = (((**this**.monde.getNiveau().getLargeur() / (**float**) w) \* (**float**) x));

② **float** posY = (**this**.monde.getNiveau().getHauteur() -((**this**.monde.getNiveau().getHauteur() / (**float**) h) \* (**float**) y));

Fonction de conversion pour chaque dimension

Comme le montre la ligne ①, le système se résout grâce à une simple règle de trois. La conversion d'unité à l'autre est proportionnelle ce qui simplifie les calculs. La ligne ② est intéressante car un élément supplémentaire vient s'ajouter : on inverse par rapport à l'écran la position Y calculée par une règle de trois comme dans la ligne précédente. Cette inversion vient du fait que libGDX ne possède pas l'origine au même endroit que celui de l'écran et permet de compenser cette différence.

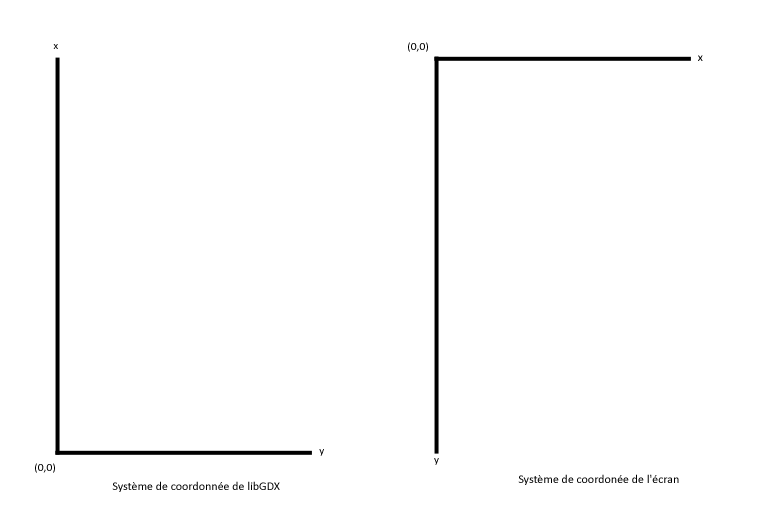


Figure Différences entre la méthode de positionnement de l'écran et de libGDX

#### Calcul de l'angle d'un vecteur

Dans le game play, nous avons défini la direction du tir de projectile par la direction imposé par la position de clic de souris par rapport à la position du personnage.

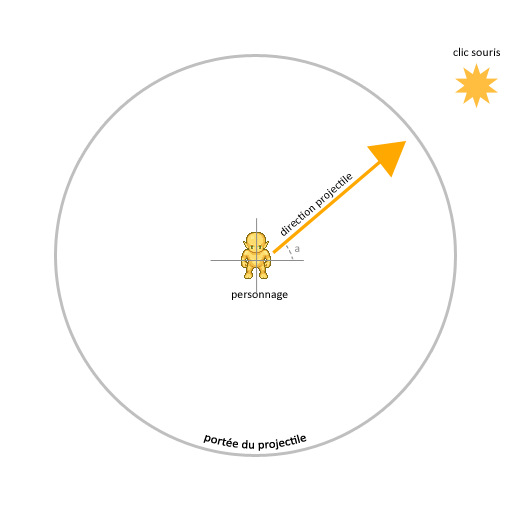


Figure Tir du personnage par rapport à la position de la souris

Aussi, libGDX possède une fonctionnalité qui permet le calcul de l’angle d’un vecteur. Le code ci-dessous montre l’utilisation de cette méthode qui retourne dans la variable angle la valeur de l’angle « a ».

// v est un vecteur

**float** angle = v.angle();

Cependant, nous pouvons réécrire cette méthode à des fins de compréhension et de recherche. Ainsi, pour donner une direction au projectile nous devons tout d'abord calculer l'angle « a » du clic par rapport à l'horizontale du personnage. En utilisant les propriétés trigonométriques du vecteur, on peut en déduire l'angle par la fonction inverse de tangente sur ] ; [ (①). Cependant, nous avons l'angle dans repère polaire que l'on donc doit traduire pour un repère orthonormé afin de pouvoir être exploité par le vecteur. Pour les convertir, nous pouvons toujours utiliser les propriétés trigonométriques des coordonnées polaires (②). Nous pouvons donc construire un vecteur de direction correspondant à la trajectoire voulue.

// v est le vecteur de la position du personnage à la position de la souris

// d est le vecteur représentant la direction du projectile

**float** angle = (**float**) Math.*atan2*(v.y, v.x); ①

d.x =(**float**)Math.*cos*(angle); ②

d.y =(**float**)Math.*sin*(angle);

Calcul d'un vecteur de direction

Le calcul de la portée du projectile se fait par rapport à la distance qui sépare le projectile de son point de lancement. Aussi, cette méthode existe également sous libGDX. De plus, deux méthodes fournissent le même résultat mais pas à la même rapidité.

// v est un vecteur

**float** dist = v.dst( x, y) ;

// v est un vecteur

**float** dist2 = v.dst2(x, y) ;

En effet, la méthode dst() utilise un calcul de racine carrée ce qui n’est pas le cas de dst2() qui est de ce fait plus rapide.

Nous fournissons ci-dessous une implémentation possible de dst(). La distance d’un calcul peut être calculée grâce au théorème du triangle rectangle de Pythagore.

En calculant la racine carrée on obtient la distance. Notons toutefois que la racine carré est un calcul gourmand pour le processeur mais négligeable par rapport à la puissance actuelle des ordinateurs et de la demande de ressources de la part de notre jeu. Voici, une implémentation par rétro-ingénierie de la fonction dst().

// vtemp est un vecteur vide

// posInitial le vecteur de départ du projectile

// position le vecteur de la position actuel du projectile

vtemp.x = (Math.*abs*(position.x - posInitial.x)); // ici la soustraction est détaillée

vtemp.y = (Math.*abs*(position.y - posInitial.y));

// calcul de la distance par le théorème de Pythagore

**float** dist = (**float**) Math.*sqrt*(Math.*pow*((**double**)vtemp.x, 2.0)+Math.*pow*((**double**)vtemp.y, 2.0));

**if** (dist > *DISTANCE\_MAX*){

// détruire le projectile

}

### La détection de collision

Le micro monde impose au joueur certaines règles l’empêchant une liberté totale de déplacement. Ainsi comme détaillé précédemment les case bloquante ou à friction influe le comportement du joueur pendant son déplacement.

Il est donc primordial de se pencher sur la manière de contrôler ces interactions. La librairie libGDX possède une fonction overlaps sur l’objet rectangle qui retourne vrai lorsque celui-ci entre collision avec un mur ou une case spéciale.

perso.getRapidite().scl(delta); ①// on travail au ralenti

Rectangle persoRect = rectPool.obtain();②

persoRect.set(perso.getCadre());

**this**.chargerCollision();③

persoRect.x += perso.getRapidite().x; ④

persoRect.y += perso.getRapidite().y;

**int** i = 0;

boolean ok = true;

**while** (i< collision.size && ok){ ⑤

**if** (collision.get(i) != **null** && persoRect.overlaps(collision.get(i))){ ⑥

perso.getRapidite().x = 0;

perso.getRapidite().y = 0;

ok = **false**;

}

i++;

}

perso.getRapidite().scl(1/delta); ⑦ // on restore la vitesse

Tout d’abord, on adapte la vitesse du déplacement du personnage afin de travailler sur des valeurs convenables et compatibles avec le déplacement. Il est par exemple, dangereux que ces valeurs soit trop grandes puisqu’à un certain seuil elles perdraient leur sens ①. Puis on crée un rectangle invisible qui simulera la présence du personnage après avoir avancé ②. On appelle ce rectangle un rectangle de travail. Ensuite, nous chargeons dans un array la totalité des cases bloquantes dans le cas d’une détection de collision entre le joueur et des murs③. Puis nous mettons à jour les coordonnées du cadre du joueur qui sont modélisées par un rectangle ④. En effet, nous utilisons des rectangles pour optimiser la rapidité des fonctions de contrôle ou de tests sur les éléments. Enfin, pour chaque élément de l’array ⑤, on vérifie que le rectangle de travail n’est pas entré en collision avec un élément bloquant ⑥. Dans le cas où le rectangle de travail entre en collision nous remettons à zéro la vitesse de déplacement du personnage pour lui empêche de continuer. Finalement, nous pouvons restaurer la vitesse du joueur et actualiser la position avec l’avancement ou non du joueur.

### Le système de matrice et le système array

Le système de matrice se défini par une matrice bidimensionnelle dont chaque d’indice x, y représente une case sur la carte par rapport à sa position. L’array quant à lui permet de lister des cases. Les deux systèmes sont tout aussi opérationnels l’un comme l’autre, mais chacun d’eux ont leurs avantages. Aussi, nous utilisons les deux car nous combinons leurs avantages pour améliorer la rapidité du jeu.

En effet, les deux méthodes sont des approches différentes de travailler sur les cases cependant si l’on traiter la liste complète des cases bloquantes par exemple il est plus utile de parcourir une liste plutôt que de tester par un parcours total d’une matrice bidimensionnelle si une case est ou n’est pas une case bloquante. De même, il est plus simple pour vérifier les cases autour du joueur il est plus simple de parcourir les cases sur une matrice dont la position des cases est l’indice plutôt que de parcourir une liste et de tester les positions de toutes les cases.

### La modélisation des projectiles

Nous modélisons les projectiles comme des objets indépendants qui respectent les contraintes de déplacement et qui sont détruits à la collision avec une case bloquante ou au dépassement de la limite de la portée.

### L'intelligence artificielle

Dans le cadre de la gestion des animaux et monstres, nous sommes amenés à implémenter une intelligence artificielle. On peut séparer deux comportements distincts : les déplacements et les réactions à un évènement.

#### Déplacements

Les déplacements se définissent par un parcours en boucle d’un chemin linéaire et prédéfini.

Vector2 v = a.getPath().get(a.getPathStep()); ①

Vector2 p = a.getPosition(); ②

corrigeDirection(a); ③

**float** aprox = 0.1f;

**if** ( Math.*abs*(p.x - v.x )<aprox && Math.*abs*(p.y - v.y )<aprox){ ④

**this**.nextStep(a); ⑤

}

① On récupère dans le vecteur v les coordonnés de la case visée par l’animé. ② On affecte la position de l’animé au vecteur. ③ Ici la direction est corrigée si l’animé n’est pas dans la bonne direction et si on entre en collision avec la case visée ④ le code sélectionne la case suivante ⑤.

#### Réaction à un événement

Les monstres ont en plus des déplacements la possibilité de réagir à un ou des évènements. Par exemple lorsqu’un joueur se téléporte d’une case ombre à l’autre le monstre peut l’intercepter et le tuer.

## Architecture du logiciel

### Programmation MVC

Le modèle MVC se décompose en trois différents éléments :

* Modèle : qui ne contient que la définition des éléments
* Vue : qui ne contient que la manière d’afficher les modèles
* Contrôleur : qui détermine le comportement des modèles

[ image hiérarchie code ]

### Hiérarchie des classes

Notre jeu est conçu selon une certaine arborescence. Lors de l’affichage, par exemple, la classe GameScreen.java est appelle la méthode update de AfficheMonde.java, qui appelle les méthodes updates de tous les modèles.

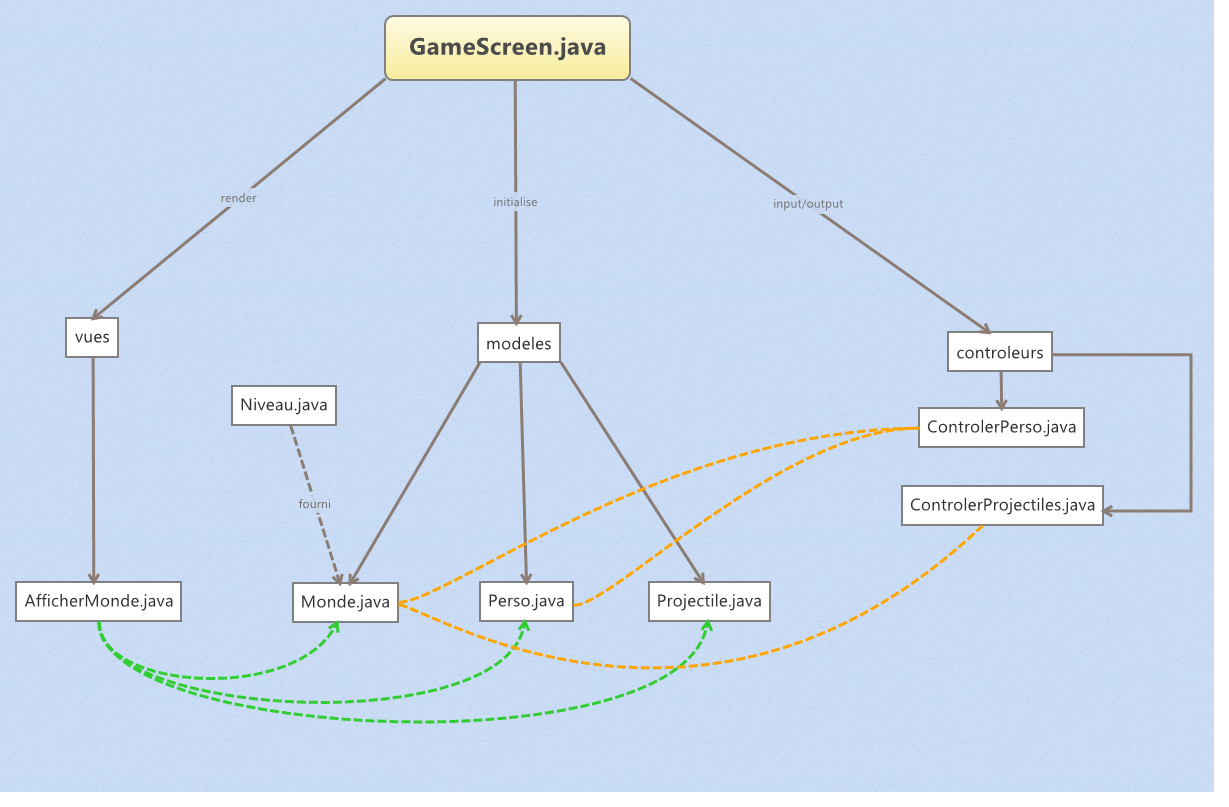


Figure ficgure montrant tous les liens

Comme indiqué sur la figure précédente, la classe GameScreen.java est au centre de l’affichage puisqu’elle exécute indirectement toutes les méthodes de tous les objets affichables du jeu.

### Développement multiplateformes

LibGDX propose par défaut différentes plateformes telles que Windows, Mac, Linux, Android, iOS et HTLM 5 pour une interface web.

Notre projet se décompose en trois sous projet tel que le projet desktop, le projet Android et le moteur du jeu. Aussi, nous avons conçu deux interfaces différentes pour la version bureau et pour la version mobile.

## Résultat

### Présentation du jeu

Le jeu se définit par un personnage principal qui pour avancer dans la carte doit accomplir des niveaux. Face au joueur se lèvent des retardataires tels que des murs, des cases spéciales, des monstres… Mais aussi des aides pour le joueur comme les animaux qui ont des caractéristiques spéciales.

### Fonctionnalités du cahier des charges

Pour ce game play original, nous devions réaliser un système de jeu sur deux cœurs différents. Pour cela, nous avons implémenté deux formes : la Shadow Form et la Light Form.

* Shadow Form
* Déplacement dans les huit directions à condition d’être dans l’ombre.
* Téléportation d’une zone d’ombre à ombre en utilisant qui doit être lancé dans la portée.
* Possibilité de piloter différents animaux.
* Changement de forme.
* Impossibilité de traverser les murs.
* Invisible dans l’ombre pour les animés.
* Light Form
* Déplacement dans les huit directions.
* Possibilité d’attaquer un monstre.
* Impossibilité de traverser les murs

Parallèlement, les animés se décomposent en deux sous-catégories : les animaux et les monstres. Les animaux sont passifs et limités contrairement aux monstres qui interagissent avec le personnage en l’attaquant.

* Animaux
* Possibilité de déplacement linéaire prédéfini et parcourus en boucle.
* Possibilité d’être contrôlé par le joueur.
* Capacité spéciale.
* Impossibilité de traverser les murs.
* Monstres
* Déplacement linéaire prédéfini et parcourus en boucle.
* Capacité de détecter le joueur s’il n’est pas dans une ombre.
* Attaquer le joueur dès que possible.
* Impossibilité de traverser le noir.
* Possibilité de mourir sous les projectiles du joueur.
* Impossibilité de traverser les murs.

### Fonctionnalités abandonnées

## Développement futur

### Fonctionnalités supplémentaires

Nous souhaitons développer notre jeu et éventuellement le continuer après ce projet tuteuré. Aussi, nous pensons ajouter différents biotopes, types de monstres et d’animaux. Nous pensons aussi ajouter de PNJ Parallèlement, le level design doit être complété pour que le jeu complète la boucle OCR définie dans le game design Document. Enfin, nous voulons offrir un produit de qualité et d’utiliser de manière complète les conseils qui nous ont été offerts par les différents professionnels du monde artistique que nous avons contacté. C’est pourquoi nous aimerions amener le projet à une qualité sonore et graphique supérieure.

### Conception artistique

#### Conception graphique

#### Conception musicale

#### Environnement sonore

### Publication sur des boutiques en ligne

# Manuel d’utilisation et d’installation

## Manuel d’installation

### Installation sur PC

#### Environnement Java (JRE)

Une machine virtuelle Java est nécessaire pour lancer le projet. L’installateur est disponible sur le site officiel [www.java.com/fr/](http://www.java.com/fr/). Lancez et complétez l’installation pour disposer de la machine virtuelle Java qui permettra d’exécuter le jeu.

#### Récupération du jeu

La version java du projet est disponible en téléchargement à l’adresse [www.projets-lightdark.fr/](http://www.projets-lightdark.fr/). Téléchargez le fichier .jar et double-cliquez dessus. Le jeu va maintenant se lancer.

Si vous utilisez une console sous Linux, placez-vous dans le répertoire du projet *cd chemin/du/projet* puis exécutez le fichier avec la commande *java jar monfichier.jar*

### Installation sous Android

#### Récupération du jeu

Pour pouvoir installer le jeu sur la mémoire de votre appareil, la première étape consiste à transférer la version Android (.apk) du projet depuis [www.projets-lightdark.fr/](http://www.projets-lightdark.fr/) vers la mémoire de votre appareil.

#### Gestionnaire de fichiers

Ensuite, téléchargez et installez depuis le magasin virtuel de Google® un explorateur de fichiers comme Astro™ (application gratuite). Naviguez ensuite dans la mémoire de votre appareil jusqu’ à l’endroit où le projet a été placé. Sélectionnez le fichier et choisissez d’installer l’application. Cette dernière est désormais disponible sur votre appareil.

## Manuel d’utilisation

### Commencer une nouvelle partie

Dans le menu principal, si vous souhaitez démarrer une nouvelle partie, cliquez simplement sur « Nouvelle partie ».

### Continuer une partie

Pour commencer une nouvelle partie, dans le menu principal, cliquez simplement sur « continuer ».

### Se déplacer

Le personnage peut se déplacer à l’aide des touches ZQSD du clavier ou, sur Android, à l’aide des flèches directionnelles affichées à l’écran. Il est possible de se déplacer dans les quatre directions cardinales et en diagonale.

### Changer de forme

En cliquant ou en appuyant sur l’orbe, le joueur change temporairement de forme. Si l’orbe est grisé alors il n’est pas possible de changer de forme pour le moment.

L’orbe est utilisable uniquement si le joueur est placé sur une ombre. De plus, certaines zones peuvent empêcher l’orbe de fonctionner.

### Attaquer

Quand il est sous sa forme de lumière, le joueur peut attaquer en cliquant ou en appuyant dans la direction voulue. Le personnage porte alors un coup d’épée dans la direction pointée. Il peut ainsi combattre les monstres qui se dressent contre lui.

### Se glisser dans une ombre

Sous forme d’ombre, le joueur se déplacer d’ombre en ombre. Pour ce faire, il dispose d’un grappin qui lui permet de se déplacer sur l’ombre pointée si elle est bien à portée de son grappin. Sinon, le grappin revient à son lanceur.

### Contrôler un animal

Il est également possible de prendre le contrôle d’animaux en se glissant dans leur ombre. Il est ainsi possible de contrôler les mouvements de l’animal : se déplacer et utiliser son action spéciale. Cette action spéciale est propre à chaque animal et se déclenche en gardant le clic (ou le doigt) enfoncé lorsque l’on contrôle l’animal.

# Rapport d’activité

## Méthode de développement

Pour le développement de notre projet, nous avons adopté, à l'initiative de notre tuteur, M Abdelkader Gouaich, la méthode Agile Scrum.

### Mise en place de la méthode Agile Scrum

#### Méthode Scrum : Définition

La méthode Scrum est une méthode Agile pour gérer les projets. Dans cette méthode, le travail à réaliser est découpé en différentes parties, nommées sprints, qui couvrent chacun une courte période de temps (dans notre cas, de une à trois semaines).

Au début de chaque sprint, nous nous retrouvions avec notre tuteur pour faire le point sur le sprint précédent, et fixer les objectifs pour le sprint courant. Nous nous organisions alors pour nous répartir librement les différentes tâches entre nous, selon le principe d’auto-organisation de cette méthode.

Enfin nous nous réunissions fréquemment durant les sprints au cours de réunions de travail ou de réflexions, que nous avons nommé séances de brainstorming.

#### Bilan d'utilisation

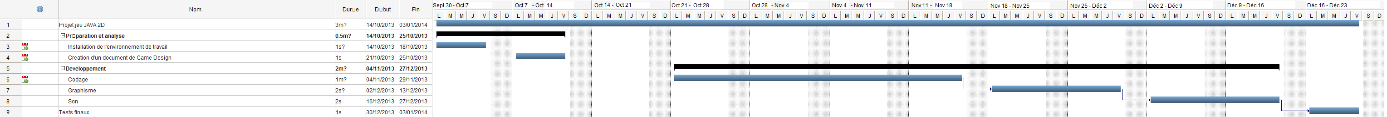
Nous avons rapidement réussi à mettre en place cette technique. L’utilisation de la méthode Agile Scrum nous a permis, rétrospectivement, de structurer et d’organiser l’avancement de notre projet, combiné à l’utilisation de Trello. Cependant nous avons aménagé et modifié quelque peu cette méthode, pour l’adapter avec nos autres contraintes scolaires.

Nous avons suivi cette méthode pour les réunions avec M Gouaich, mais nous l’avons, concernant la tenue de nos réunions (ou brainstorming), quelque peu modifié pour l’adapter à la tenue de réunions plus longues.

## Planification

### Planning prévu

Comme demandé par les consignes, nous avons réalisé un planning initial, sous forme de diagramme de Gantt, couvrant l’intégralité de la durée du projet, en évaluant à priori les taches à réaliser et leur durée.



Ce planning global a au final été peu suivi, pour plusieurs raisons, notamment liées au fait que nous ne savions pas comment créer un jeu vidéo, ni d’idée précise sur les tâches à réaliser. Nous avons notamment sous-estimé la création du game design document, que nous pensions réaliser en une semaine et qui, au final, s’est révélée plus complexe que prévu en occupant près de six sprints.

D’autre part, les autres travaux qui nous ont été confiés à l’IUT nous ont malgré-nous pris beaucoup de temps, qui a d’autant retardé l’avancement du projet.

### Planning réels

#### Des plannings sprints par sprints

Nous avons, en plus du planning global, réalisé pour chaque sprint des plannings des tâches à l’aide de diagrammes de Gantt. Ces planning, plus en adéquation avec la réalité de l’avancement de notre projet, ont été plus réalistes et mieux suivis.

#### Les contraintes

Ces plannings, bien que plus réalistes, ne comprenaient pas nos contraintes, notamment les indisponibilités de chacun ou bien les examens, ils n’ont donc pas été parfaitement suivis.

#### Conclusion sur les plannings

Bien que nous ayons réalisés les plannings, ils n’ont été finalement que moyennement suivi, du fait des différents facteurs cités plus haut. A posteriori, l’utilisation de ces plannings ne nous a apporté que peu de choses, comparé à l’utilisation des outils Wiki et Trello.

## Méthode et outils de travail

### Section modélisation

#### Modélisation UML

Pour la modélisation nous avons choisi d'utiliser surtout le logiciel Modelio, mais aussi parfois le logiciel StarUML, qui sont des logiciels gratuits de modélisation UML.

#### Autres diagrammes

Pour réaliser quelques diagrammes explicatifs sous forme de cartes mentales, à destination du wiki, nous avons également utilisé occasionnellement le logiciel Xmind.

Enfin nous avons réalisé quelques schémas à la main, que nous avons ensuite scannés.

### Section Codage

Pour le codage, nous avons utilisé le langage de programmation JAVA, la plateforme de développement éclipse, et libGDX, avec SVN sur un serveur Google.

#### JAVA

Nous avons dû utiliser, dans le cadre de notre projet, libGDX qui est une librairie écrite en java. Ainsi, le choix du langage de programmation s’est imposé : Java

#### Eclipse

Eclipse est un IDE axée développement Java que nous avions l’habitude d’utiliser dans le cadre de notre formation, nous avons donc naturellement continué avec ce logiciel.

#### LibGDX

LibGDX est une librairie Java, axée sur la création et le développement de jeu vidéo, imposé pour ce projet. Bien que nous ne la connaissions pas, elle s’est révélé être d’une grande aide pour le développement du jeu vidéo, en nous offrant de multiples méthodes déjà réalisée et en nous fournissant un squelette de code type.

#### SVN

SVN est un outil de gestion de version, qui permet de gérer le développement d’un projet à plusieurs membres. Nous avons choisis d’utiliser SVN plutôt que GIT, son concurrent, car c’est un outil sur lequel nous avions appris à utiliser dans le cadre de notre formation. Malgré quelques erreurs, nous a permis de travailler en commun sur une ou parfois plusieurs copies du programme.

Nous avons choisi de stocker notre projet sur un serveur Google, de par sa simplicité d’utilisation et sa gratuité.

#### Utilisation coordonnée

Une séance usuelle de travail sur le code du jeu suivait les étapes suivantes :

* Définition des objectifs de la séance sur Trello
* Récupération de la dernière version du dépôt SVN
* Développement de la fonctionnalité fixée plus tôt
* Mise à jour de la copie de travail puis enregistrement des modifications sur le dépôt

### Section organisation.

#### Wiki

Nous avons réalisé un wiki, pour permettre aux observateurs de suivre l’évolution de notre projet, tel que demandé dans les consignes du projet.

Nous avons préféré l’utilisation d’un wiki contre celle d’un blog, pour plus de simplicité : nous sommes en effets plus habituées à utiliser un wiki qu’un blog, et de plus, l’utilisation d’un wiki nous a semblé plus intéressante.

Ce wiki a régulièrement été mis-à-jour, d’autant plus qu’il s’est rapidement révélé utile pour savoir entre nous l’avancement global du projet, de suivre l'évolution du projet et de stocker les comptes rendus des réunions et de savoir l'avancement général du projet.

#### Trello

Trello est un outil web gratuit permettant d’organiser et de gérer des projets. Les tâches, qui peuvent être ajoutés par chacun des membres, peuvent être ajoutés dans trois colonnes : à faire, en cours, fait, et sont déplaçable de l’une à l’autre. Elles peuvent par ailleurs se voir attribuer des membres.

Bien utilisé, Trello nous a permis de répondre aux questions : « Que faut-il faire ? », « Qui fait quoi ? », « Ou en est-on? », et nous a permis de nous organiser facilement au niveau de chaque tâche.

Cet outil, que nous avons tenté d’associer à chacune de nos séances de travail, en particulier lors du codage ou les petites tâches se sont démultipliées, a pu être d’une grande aide.

#### Communication

Au niveau du travail, nous avons travaillé en groupe et individuellement.

Concernant le travail de groupe, nous avons privilégié autant que possible la rencontre de visu, de préférence dans les locaux de l’IUT de Montpellier. Lors des séances de télétravail, nous avons utilisé Teamspeak comme logiciel de communication.

#### Rapport

Enfin, pour l’écriture du rapport, nous avons choisi d’utiliser Microsoft Word 2013, pour nous simplifier la mise en page et l’écriture du rapport en utilisant une plateforme moderne et identique pour chaque des membres du groupe.

Conclusion

Bibliographie / Sitographie

**Aucune source spécifiée dans le document actif.**

Annexes

**Résumé**

Le projet Light/Dark vise à créer un jeu vidéo en deux dimensions en passant par toutes les étapes de création d’un jeu, du document de Game-Design jusqu’à la réalisation d’un modèle pleinement fonctionnel pouvant être présenté puis joué par des joueurs réguliers.

Il illustre bien les différentes contraintes liées à la réalisation d’un jeu vidéo orienté puzzle/aventure : analyse, ergonomie, mécanismes, environnement visuel et sonore, mais surtout la relation particulière entre le scénario et le gameplay.

Ce jeu a été développé en utilisant le langage de programmation Java et la librairie libGDX.

**Mots clés**

Jeu vidéo, 2D, Game-Design Document, gameplay, scénario, boucle OCR, Java, libGDX.