Plan

Contenu

[Conception 2](#_Toc377298095)

[Technologies utilisés 2](#_Toc377298096)

[Conception de Lightdark 3](#_Toc377298097)

[La sadowform 3](#_Toc377298098)

[La lightform 3](#_Toc377298099)

[Les monstres 3](#_Toc377298100)

[Les animaux 3](#_Toc377298101)

[Les collisions et interactions physiques 3](#_Toc377298102)

[Les Algorithmes spécifiques 3](#_Toc377298103)

[Le système de coordonnées de libGDX 3](#_Toc377298104)

[Les équations de conversion 3](#_Toc377298105)

[La détection de collision 3](#_Toc377298106)

[Le système de matrice et le système array 3](#_Toc377298107)

[La modélisation des projectiles 3](#_Toc377298108)

[L'intelligence artificielle 3](#_Toc377298109)

[Architecture du logiciel 3](#_Toc377298110)

[Programmation MVC 3](#_Toc377298111)

[Hiérarchie des classes 3](#_Toc377298112)

[Développement multiplateformes 3](#_Toc377298113)

[Résultat 3](#_Toc377298114)

[Présentation du jeu 3](#_Toc377298115)

[Fonctionnalités du cahier des charges 3](#_Toc377298116)

[Fonctionnalités abandonnées 4](#_Toc377298117)

[Développement futur 4](#_Toc377298118)

[Fonctionnalités supplémentaires 4](#_Toc377298119)

[Conception artistique 4](#_Toc377298120)

[Publication sur des boutiques en ligne 4](#_Toc377298121)

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 formestelles que les plates-formesmobiles et bureautiques. Bien qu'au début nous devions utiliser GAME qui est un moteur de jeu développé par *lirmm* nous avons choisis libGDX pour sa diversité en fonctionnalité et sa légèreté.

SVN est un logiciel de version. Le but de ce type de logiciel consiste à sauvegarder chaque mise à jour d'un projet sur un serveur centralisé et de coordonner le travail collaboratif entre plusieurs utilisateurs.

Lors de l'organisation, un groupe peut avoir le besoin de visualiser les objectifs restants et les objectifs réalisés. En ce sens, le site web Trello permet aisément de gérer en groupe une liste de rappels.

Pour la modélisation UML, nous utilisons StarUML etModélio 3.0 même si nous préférons utiliser de temps en temps Xmind, qui est un logiciel de conception de carte mentale,pour certains diagrammes puisque ce dernier permet un affichage plus esthétique.

## Conception de Lightdark

### Les collisions et interactions physiques

Dans le micro monde du jeu il existe différents type 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 gamedesign document précise que le joueur doit pouvoir se déplacer dans une zone d'ombre s'il est en mode shadow.

#### 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 tel que les monstres, les animaux et les projectiles. De cette façon le micro monde de lightdark peut imposer des contraintes sur ses différents acteurs en agissant directement sur leur comportement.

#### Les cases ombres

Ces cases propres à la shadowform permettent au joueur de s’infiltrer dans le niveau tout en le protégeant. Les monstres ne peut effectivement pas détecter le joueur en forme shadow si celui-ci se trouve dans une case ombre. Contrairement, aux monstres qui « ont peur du noir », 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 pluparts imposent un changement notable dans le déplacement de l’acteur.

### La sadowform

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 shadowform est la possibilité de contrôler un animal, et ce, même si ce dernier se déplace.

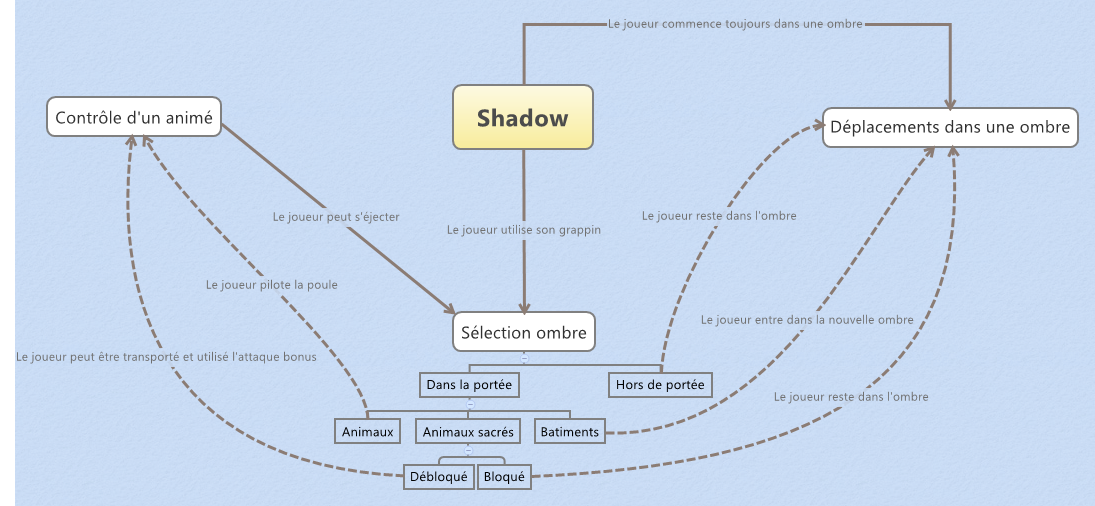


Diagramme de cas d'utilisation de la forme shadow.

Ce diagramme montre bien toutes les possibilités de cette forme. Aussi cette forme est celle par défaut du joueur puisqu'il ne peut pas être tout le temps en lightform.

Comme indiqué sur le diagramme 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 shadow que lorsqu'il traverseune zone de lumière.

### La lightform

Cette forme est simple puisqu’elle contient les déplacements de la shadowform 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 gameplay 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 qui ne peuvent traverser et 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 lightform.

[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]

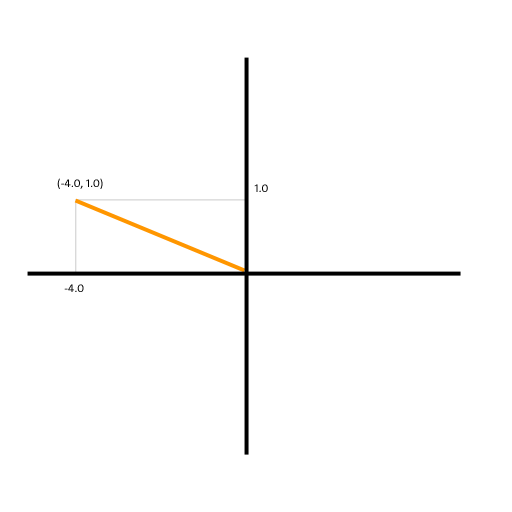
Lors d’une sortie de leur parcours l’intelligence artificielle de l’animal leur permet de revenir sur leur parcours et de le continuer indéfiniment.

## Les Algorithmes spécifiques

### Le système de coordonnées de libGDX

LibGDX n’utilise pas un système de coordonnée 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 coordonnée 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 de d’unités il définit son écran, puis se positionne en fonction du système de coordonnées qu’il a défini. L’avantages 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.



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

### Les équations de conversion

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

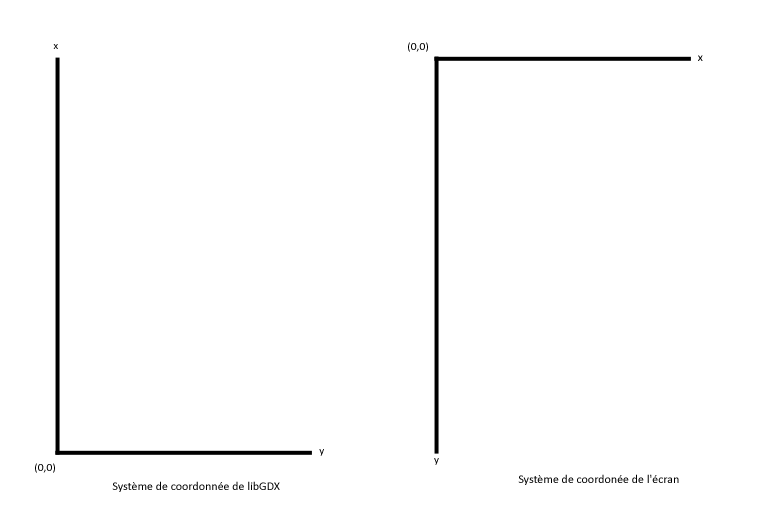
Les entrées étant toujours dans le système de coordonnées pixel et il faut une fonction de conversion pour passer d'un système à l'autre.

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

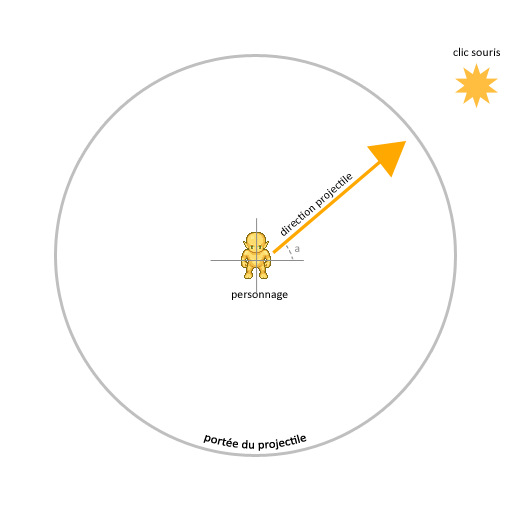
Comme le montre la ligne ①, le système se résout grâce à une simple règle de trois. La conversion d'unité à l'autre est proportionnelle ce qui simplifie les calculs. La ligne ② est intéressante car un élément supplémentaire vient s'ajouter: on inverse par rapport à l'écran la position Y calculé par une règle de trois comme dans la ligne précédente. Cet 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.



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

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

Dans le gameplay, 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.



Tir du personnage par rapport à la position de la souris

Aussi pour donner une direction au projectile nous devons tout d'abord calculer l'angle "a" du clic par rapport à l'horizontal du personnage. En utilisant les propriétés de la trigonométrie 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 doit traduire pour un repère orthonormé. Pour les convertir, nous pouvons toujours utiliser les propriétés trigonométrique des coordonnées polaires (②). Nous pouvons donc construire un vecteur de direction correspondant à la trajectoire voulu.

// 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 porté du projectile se fait par rapport à la distance qui sépare le projectile de son point de lancement. Cela peut être calculé grâce au théorème de Pythagore sur le triangle rectangle.

En calculant la racine carré 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.

// 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));

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 système de matrice et le système array

### La modélisation des projectiles

### L'intelligence artificielle

#### Déplacements intelligents

#### Réaction à un événement

## Architecture du logiciel

### Programmation MVC

### Hiérarchie des classes

### Développement multiplateformes

# Résultat

## Présentation du jeu

### Fonctionnalités du cahier des charges

### Fonctionnalités abandonnées

## Développement futur

### Fonctionnalités supplémentaires

PNJ

### Conception artistique

#### Conception graphique

#### Conception musicale

#### Environnement sonore

### Publication sur des boutiques en ligne