

**CY Cergy Paris Université**

**RAPPORT**

pour le projet d'intégration  
**Licence d'Informatique troisième année**

sur le sujet

**Explorateurs autonomes et communicants**

rédigé par

**Afatchawo Junior - Chriqui Nathan - Da Cruz Mathis**



Avril 2022

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte du projet . . . . .	3
1.2	Objectif du projet . . . . .	3
<b>2</b>	<b>Spécification du projet</b>	<b>4</b>
2.1	Notions de base et contraintes du projet . . . . .	4
2.2	Fonctionnalités attendues du projet . . . . .	4
2.3	IHM . . . . .	4
2.4	Spécificité . . . . .	5
<b>3</b>	<b>Conception et réalisation du projet</b>	<b>6</b>
3.1	Architecture globale du logiciel . . . . .	6
3.2	Conception des traitements (processus) . . . . .	7
3.2.1	Aventurière . . . . .	7
3.2.2	Offensive . . . . .	8
3.2.3	Défensive . . . . .	10
3.3	Conception du multi-threading . . . . .	11
3.4	Conception de l'IHM graphique . . . . .	12
<b>4</b>	<b>Manuel utilisateur</b>	<b>13</b>
4.1	Menu . . . . .	13
4.2	Fenêtre des règles . . . . .	14
4.3	Fenêtre de choix de stratégie . . . . .	15
4.4	Fenêtre principale . . . . .	16
4.5	Fin de jeu . . . . .	18
<b>5</b>	<b>Déroulement du projet</b>	<b>19</b>
5.1	Réalisation du projet par étapes . . . . .	19
5.2	Répartition des tâches entre membres de l'équipe . . . . .	19
<b>6</b>	<b>Conclusion et perspectives</b>	<b>20</b>
6.1	Résumé du travail réalisé . . . . .	20
6.2	Améliorations possibles du projet . . . . .	20
6.3	Derniers mots . . . . .	20

## Table des figures

1	UML simplifié du logiciel . . . . .	6
2	Organisation des différentes fenêtres de l'IHM graphique . . . . .	12
3	Menu . . . . .	13
4	Fenêtre des règles . . . . .	14
5	Fenêtre de choix de stratégie . . . . .	15
6	Fenêtre principale . . . . .	16
7	Carte d'exploration . . . . .	16
8	Affichage utilisateur . . . . .	17
9	Bouton menu . . . . .	17
10	Fin du Jeu . . . . .	18

## **Remerciements**

Tout d'abord, nous tenons à remercier sincèrement M. Tianxiao Liu, qui, en tant qu'accompagnant professeur pour notre Projet d'Intégration, s'est toujours montré à l'écoute, disponible et clair pour notre groupe tout au long de ce projet. En effet, grâce à sa dévotion en tant que superviseur, il nous a fournis beaucoup d'aide durant différentes réunions afin de nous guider aisément durant l'accomplissement de ce travail. Nous remercions également toute l'équipe des Professeurs de CY Cergy Paris Université.

# **1 Introduction**

Dans cette section, nous allons présenter brièvement l'objectif du projet

## **1.1 Contexte du projet**

La motivation de ce projet est de découvrir la programmation multi-threading ainsi que d'implémenter un jeu du même style d'un jeu assez connu : Curious Expedition.

De plus, l'idée générale nous a bien plus : développer une simulation de stratégie.

## **1.2 Objectif du projet**

Ce projet a pour objectif de réaliser une simulation d'explorateurs à la recherche de trésors sur une île dangereuse et inexplorée en IHM graphique.

## 2 Spécification du projet

Nous avons présenté l'objectif du projet dans la section 1. Dans cette section, nous présentons la spécification de notre logiciel réalisé. Ceci correspond principalement au document de spécification du projet (cahier des charges).

### 2.1 Notions de base et contraintes du projet

**Principe Général :** Le but est de récupérer le maximum de trésors possibles parmi ceux disponibles sur la carte, le tout en minimisant les pertes des explorateurs (mort) durant des combats de monstres. Il convient aussi d'être le plus rapide et le plus astucieux lors du choix de la stratégie.

**Explorateurs (logiciel) :** Dans ce projet, on souhaite donc réaliser une simulation permettant de choisir parmi différentes stratégies, l'aventure la plus rapide sans minimiser la sécurité. Cette simulation s'effectue ainsi tour par tour avec des déplacements plus ou moins aléatoires.

**Outils de développement :**

1. Java
2. Eclipse
3. Latex

### 2.2 Fonctionnalités attendues du projet

L'utilisateur du logiciel doit pouvoir :

- Voir apparaître le menu principal
- Décider de commencer une nouvelle partie ou quitter l'application
- Visualiser les différentes règles du jeu
- Opter pour choisir entre 3 stratégies.
- Visualiser la carte et ses limites
- Visualiser les explorateurs
- Visualiser les trésors, montagnes et monstres de la carte
- Connaitre à tout moment le nombre d'explorateurs en vie ainsi que le nombre de trésors restant
- Appercevoir sa bourse d'or s'incrémenter à chaque trésor récupéré
- Retourner au menu principal en fin de partie ou en cours de simulation
- En fin de partie, visualiser son résultat (gold récupéré)

### 2.3 IHM

➤ Jeu

- Carte d'exploration représentée à l'aide d'une image
- Explorateurs débutants tous au même point de départ
- Montagnes, obstacles immobiles
- Griffons, monstres dangereux
- Affichage des trésors et explorateurs restants

- Affichage de la bourse d'or du joueur
- Boutons :
  - \* Menu : retourner au menu du logiciel

➤ Menu

- Boutons :
  - \* Start : lancer le jeu
  - \* Règles : liste des règles du jeu
  - \* Quitter : fermer le logiciel

➤ Choix stratégie

- Boutons :
  - \* Aventurière
  - \* Offensive
  - \* Défensive
  - \* Menu : retourner au menu du logiciel

## 2.4 Spécificité

- Actualisation tour par tour
- Nombre aléatoire d'explorateurs
- Nombre aléatoire de montagnes et griffons
- Déplacements aléatoires des explorateurs en fonction des emplacements des trésors
- Comptage de l'or : Incrémentation de la bourse d'or du joueur de (50 ou 100) à chaque trésor récupéré
- Chaque simulation est différente de la précédente
- La stratégie commune est : L'explorateur se dirige vers un coffre, une fois le coffre atteint, un autre coffre lui sera attribué. Lors d'une confrontation avec un monstre, l'explorateur perdra des hp/pv. Jauge de vie initiale est de 100 hp/pv. La fin du jeu occurre lorsque l'ensemble des trésors a été trouvé ou que l'ensemble des explorateurs soit mort.
- De plus, le choix de la stratégie influe sur le déroulement de la simulation
  - Aventurière : L'explorateur peut escalader les obstacles au lieu de les contourner. Il est donc pas armé s'il croise un monstre il se fait directement tuer.
  - Offensive : L'explorateur est armé d'une épée. Lors d'un affrontement, le monstre adverse est tué, quant à notre explorateur il perdra 50 hp/pv (moitié de sa vie).
  - Défensive : L'explorateur est équipé d'un bouclier. Lors d'un affrontement, le monstre ne prend aucun dégât, quant à l'explorateur il perdra seulement 25 hp/pv (quart de sa vie).

### 3 Conception et réalisation du projet

#### 3.1 Architecture globale du logiciel

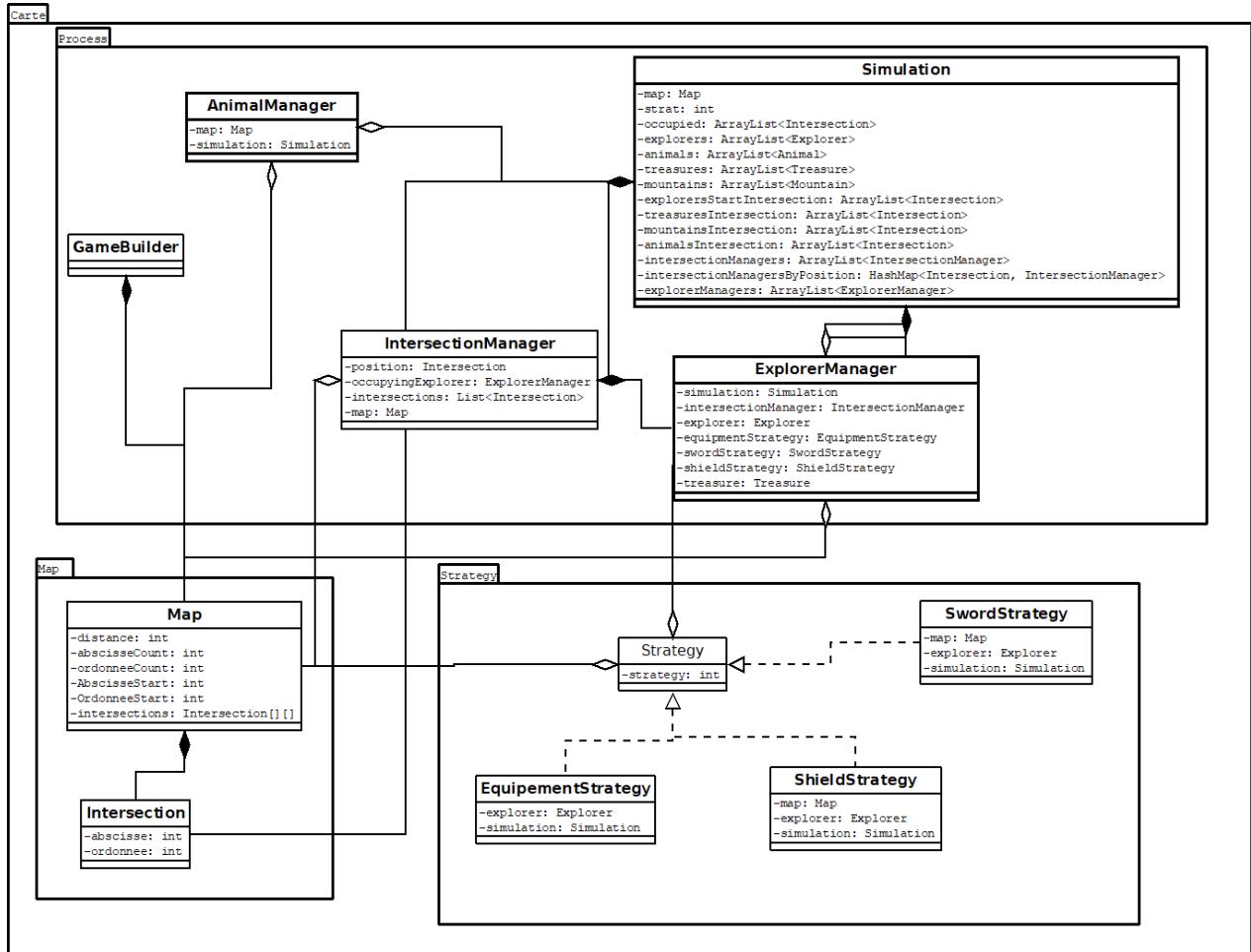


FIGURE 1 – UML simplifié du logiciel

## 3.2 Conception des traitements (processus)

Cette section comporte les algorithmes principaux de notre projet qui ont permis l'implémentation des stratégies ainsi que de prendre en charge certaines exceptions. Ces algorithmes sont volontairement simplifiés afin de faciliter au mieux la compréhension.

### 3.2.1 Aventurière

Dans 1, on gère les événements causés par le déplacement d'un explorateur vers une nouvelle case. Si un explorateur rencontre un trésor, on retire le trésor de la carte. Si un explorateur rencontre un animal sauvage, l'explorateur est K.O, on le retire de la carte.

---

#### Algorithm 1 call(Intersection position, Treasure treasure)

---

```
newPosition ← moveExplorer(position,treasure)
if belongTo(position,getTreasuresIntersection()) then
    getTreasuresIntersection().remove(position)
    getTreasures().remove(treasure)
    if belongTo(position,getAnimalsIntersection()) then
        getTreasures().add(treasure)
        getExplorers().remove(explorer)
    end if
end if
return newPosition
```

---

Dans 2, on s'occupe de la trajectoire de notre explorateur en l'occurrence, l'explorateur doit se rapprocher du trésor qui lui a été affecté.

---

#### Algorithm 2 moveExplorer(Intersection position, Treasure treasure)

---

```
treasurePosition ← treasure.getPosition()
newAbscisse ← treasurePosition.getAbscisse()
newOrdonnee ← treasurePosition.getOrdonnee()
if position.getAbscisse() > treasurePosition.getAbscisse() then
    newAbscisse ← position.getAbscisse() – GameConfiguration.BLOCKSIZE
else if position.getAbscisse() < treasurePosition.getAbscisse() then
    newAbscisse ← position.getAbscisse() + GameConfiguration.BLOCKSIZE
end if
if position.getOrdonnee() > treasurePosition.getOrdonnee() then
    newOrdonnee ← position.getOrdonnee() – GameConfiguration.BLOCKSIZE
else if position.getOrdonnee() < treasurePosition.getOrdonnee() then
    newOrdonnee ← position.getOrdonnee() + GameConfiguration.BLOCKSIZE
end if
return new Intersection(newAbscisse,newOrdonnee)
```

---

### 3.2.2 Offensive

Dans 3, on gère les événements causés par le déplacement d'un explorateur vers une nouvelle case. Si un explorateur rencontre un trésor, on retire le trésor de la carte. Si un explorateur rencontre un animal sauvage, l'animal sauvage est K.O, on le retire de la carte quant à l'explorateur il perdra 50 pv. Si l'explorateur est à 0 pv, l'explorateur est K.O, on le retire de la carte.

---

**Algorithm 3** call(Intersection position, Treasure treasure)

---

```
newPosition ← moveExplorer(position, treasure)
isDelete ← 0
delete ← newAnimal(position)
if belongTo(position, getAnimalsIntersection()) then
    for all animal in getAnimals() do
        if animal.getPosition = position then
            isDelete ← 1
            delete ← animal
        end if
    end for
end if
if belongTo(position, getTreasuresIntersection()) then
    getTreasuresIntersection().remove(position)
    getTreasures().remove(treasure)
end if
if isDelete = 1 then
    getAnimalsIntersection().remove(position)
    getAnimals().remove(delete)
    if explorer.getLife() = 100 then
        explorer.setLife(50)
    else
        getTreasures().add(treasure)
        getExplorers().remove(explorer)
    end if
end if
return newPosition
```

---

Dans 4, on s'occupe de la trajectoire de notre explorateur en l'occurrence, l'explorateur doit se rapprocher du trésor qui lui a été affecté, tout en évitant les obstacles naturels (montagne).

---

**Algorithm 4** moveExplorer(Intersection position, Treasure treasure)

---

```

treasurePosition ← treasure.getPosition()
newAbscisse ← treasurePosition.getAbscisse()
newOrdonnee ← treasurePosition.getOrdonnee()
if position.getAbscisse() > treasurePosition.getAbscisse() then
    newAbscisse ← position.getAbscisse() – GameConfiguration.BLOCKSIZE
else if position.getAbscisse() < treasurePosition.getAbscisse() then
    newAbscisse ← position.getAbscisse() + GameConfiguration.BLOCKSIZE
end if
if position.getOrdonnee() > treasurePosition.getOrdonnee() then
    newOrdonnee ← position.getOrdonnee() – GameConfiguration.BLOCKSIZE
else if position.getOrdonnee() < treasurePosition.getOrdonnee() then
    newOrdonnee ← position.getOrdonnee() + GameConfiguration.BLOCKSIZE
end if
result = newIntersection(newAbscisse, newOrdonnee)
while belongTo(result, getMountainsIntersection()) do
    percent = Math.random()
    if percent <= 0.25 then
        result = getElementPosition(position.getAbscisse(), position.getOrdonnee() +  

GameConfiguration.BLOCKSIZE)
    else if percent <= 0.5 then
        result = getElementPosition(position.getAbscisse(), position.getOrdonnee() –  

GameConfiguration.BLOCKSIZE)
    else if percent <= 0.75 then
        result = getElementPosition(position.getAbscisse() + BLOCKSIZE, position.getOrdonnee())
    else
        result = getElementPosition(position.getAbscisse() - BLOCKSIZE, position.getOrdonnee())
    end if
end while
return result

```

---

### 3.2.3 Défensive

Dans 5, on gère les événements causés par le déplacement d'un explorateur vers une nouvelle case. Si un explorateur rencontre un trésor, on retire le trésor de la carte. Si un explorateur rencontre un animal sauvage, il perdra 25 pv. Si l'explorateur est à 0 pv, l'explorateur est K.O, on le retire de la carte.

---

**Algorithm 5** call(Intersection position, Treasure treasure)

---

```
newPosition ← moveExplorer(position, treasure)
if belongTo(newPosition, getAnimalsIntersection()) then
    explorer.setLife(explorer.getLife() - 25)
    if explorer.getLife() = 0 then
        getTreasures().add(treasure)
        getExplorers().remove(explorer)
    end if
end if
if belongTo(position, getTreasuresIntersection()) then
    getTreasuresIntersection().remove(newPosition)
    getTreasures().remove(treasure)
end if
return newPosition
```

---

Ici moveExplorer est pareil que dans 4.

### 3.3 Conception du multi-threading

Pour la conception du **Multi-threading**, nous nous sommes basés sur le code de démonstration du Train donné par le prof.

Chaque Explorateur est un **thread**. La zone d'entrée des explorateurs se fait par le bloc tout en haut à gauche (le premier bloc). Les explorateurs rentrent alors un à un sur la carte à chaque fois que la zone d'entrée est inoccupée.

Dans la classe de « management » de l'explorateur (**ExplorerManager**), la nouvelle position de l'explorateur est déterminée par la stratégie utilisée et sa position actuelle. On modifie alors la position de l'explorateur grâce à la méthode **setPosition(position)**.

Le Manager du bloc (Intersection) correspondant à cette dernière est récupérée depuis la classe Simulation. L'explorateur peut alors y entrer par la méthode **synchronized void enter** qui permet d'attribuer un nouvel explorateur au bloc et de signaler qu'il est occupé pour empêcher d'autres explorateurs d'y rentrer. La position de l'explorateur est alors actualisée par le « gérant » de cette intersection (IntersectionManager).

Concernant la classe **IntersectionManager**, par la méthode **synchronized void enter** mentionnée plus haut, on modifie la position du « Manager » de l'explorateur (ExplorerManager). On récupère l'IntersectionManager précédemment occupé par l'explorateur et on l'a libère en utilisant la méthode **synchronized void exit** qui mettra alors à jour le fait qu'il n'y a plus d'explorateur dans la zone en rendant l'attribut **occupyingExplorer** de cette classe à null. Et aussi avec la méthode **notify** qui indiquera que le thread de l'explorateur va « abandonner » la zone et dire aux threads des autres explorateurs que cette zone est libre. L'**ExplorerManager** de l'explorateur modifie son IntersectionManager et prends celui actuel ou il se trouve. Ce qui implique que l'attribut **occupyingExplorer** de l'actuel zone n'est plus null et prends le thread de l'explorateur qui vient de rentrer dans la zone.

### 3.4 Conception de l'IHM graphique

- Menu
- Choix de la stratégie
- Règles du jeu
- Fenêtre Principale

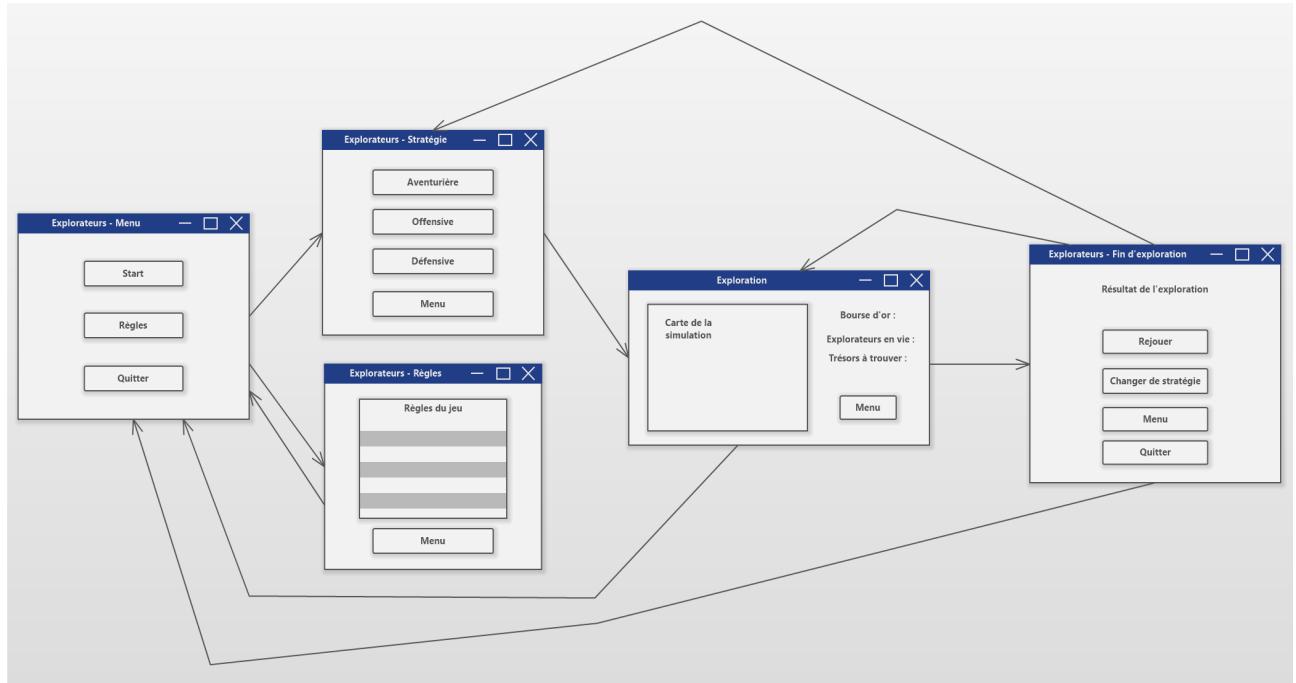


FIGURE 2 – Organisation des différentes fenêtres de l'IHM graphique

## 4 Manuel utilisateur

### 4.1 Menu

Dans la fenêtre d'accueil, les joueurs ont 3 options :

- Start : Lance le jeu et plus particulièrement la fenêtre de choix de stratégie.
- Regles : Lance une fenêtre affichant la liste des règles du jeu.
- Quitter : Quitte le logiciel.

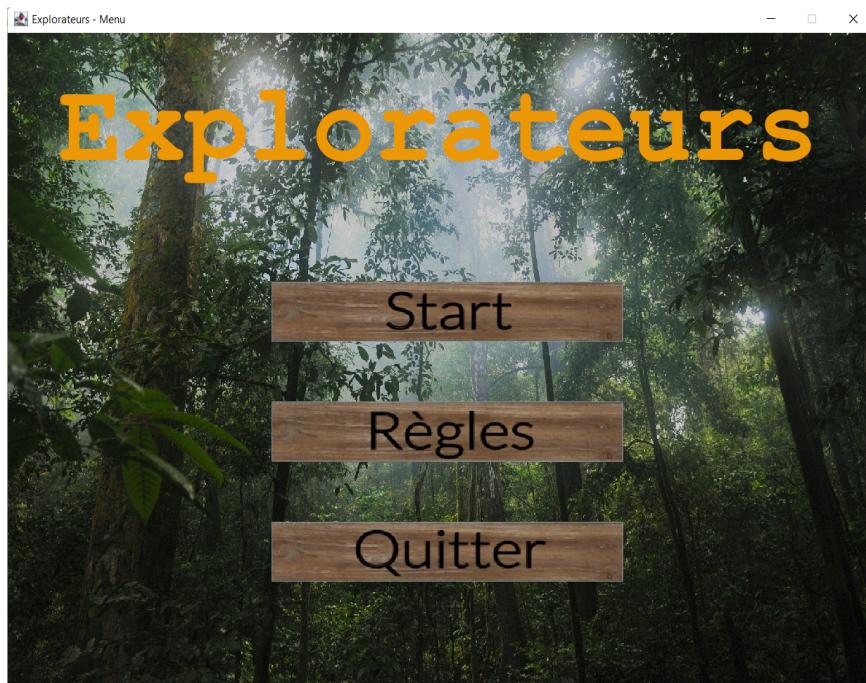


FIGURE 3 – Menu

## 4.2 Fenêtre des règles

Dans la fenêtre des règles, les joueurs ont accès à la liste des règles du jeu



FIGURE 4 – Fenêtre des règles

### 4.3 Fenêtre de choix de stratégie

Dans la fenêtre de choix, les joueurs ont accès à 3 options de stratégie différentes et une option pour retourner au menu du jeu :

- Aventurière : Lance le jeu avec la stratégie d'exploration **aventurière** : L'explorateur peut escalader les obstacles au lieu de les contourner. Il est donc pas armé s'il croise un monstre il se fait directement tuer.
- Offensive : Lance le jeu avec la stratégie d'exploration **offensive** : L'explorateur est armé d'une épée. Lors d'un affrontement, le monstre adverse est tué, quant à notre explorateur il perdra 50 hp/pv (moitié de sa vie).
- Défensive : Lance le jeu avec la stratégie d'exploration **défensive** : L'explorateur est équipé d'un bouclier. Lors d'un affrontement, le monstre ne prend aucun dégât, quant à l'explorateur il perdra seulement 25 hp/pv (quart de sa vie).
- Menu : Retourne au menu du logiciel.



FIGURE 5 – Fenêtre de choix de stratégie

#### 4.4 Fenêtre principale

Dans la fenêtre principale, les joueurs ont accès à 3 sections différentes :

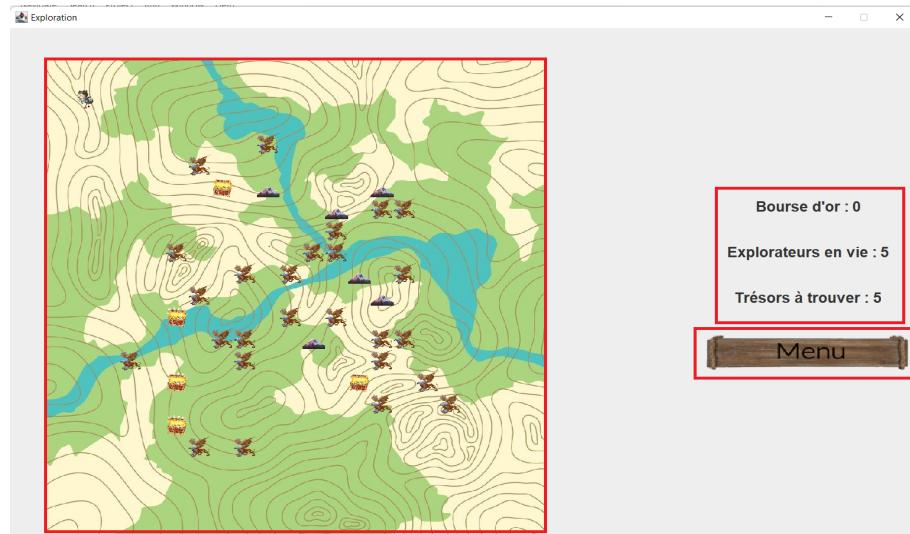


FIGURE 6 – Fenêtre principale

1. Carte d'exploration : Il s'agit du coeur de notre fenêtre, c'est la section qui va être le plus utilisée. Tour par tour, les explorateurs vont se déplacer à la recherche des trésors sur l'île inexplorée.

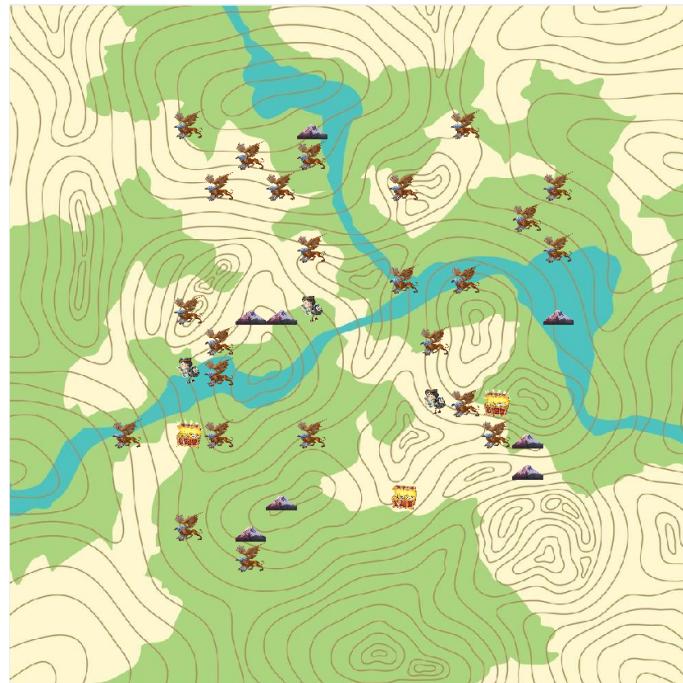


FIGURE 7 – Carte d'exploration

2. Affichage tour par tour : Lorsqu'un trésor est récupéré par un explorateur, le contenu de la bourse d'or est incrémenté par la valeur du trésor (50 ou 100) et le nombre de trésor à trouvé est diminué. De même, lors du décès d'un explorateur, le nombre d'explorateurs restant est décrémenté

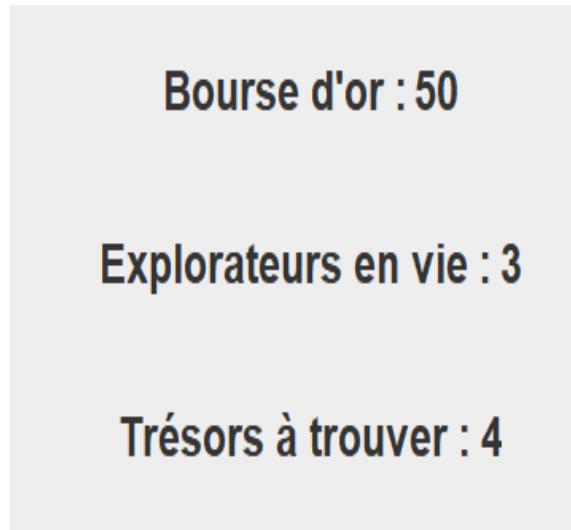


FIGURE 8 – Affichage utilisateur

3. Bouton :

- Menu : Arret de la simulation et retour au menu du logiciel



FIGURE 9 – Bouton menu

#### 4.5 Fin de jeu



FIGURE 10 – Fin du Jeu

A la fin de la partie, un texte indiquant le résultat de l'exploration ainsi qu'un récapitulatif du nombre de pièces d'or récupérées est affiché.

Enfin dans la fenêtre de fin de jeu, les joueurs ont 4 options :

- Rejouer : Relancer une nouvelle exploration avec la même stratégie.
- Changer de stratégie : Renvoie à la fenêtre de choix de stratégie afin de démarrer une une nouvelle exploration avec une stratégie différente.
- Menu : Renvoie au Menu (fenêtre d'accueil).
- Exit : Quitte le logiciel.

## 5 Déroulement du projet

Dans cette section, nous décrivons comment le projet a été réalisé en équipe : la répartition des tâches, la synchronisation du travail en membres de l'équipe, etc.

### 5.1 Réalisation du projet par étapes

Etapes d'avancement :

1. Spécifications du projet : Documentation, cahier des charges et répartition des tâches.
2. Conception des classes de données : UML des classes de données, premières classes.
3. Conception de l'IHM graphique.
4. Conception du noyau fonctionnel : Conceptions des classes et approfondissements...
5. Conception du multi-threading
6. Algorithmes principaux : Règles du jeu, exceptions, résolution des problèmes rencontrés.
7. Latex et documentation du projet : Rédaction du rapport.
8. Préparation à la soutenance finale : Préparation pour l'oral

### 5.2 Répartition des tâches entre membres de l'équipe

Les différents membres de l'équipes ont su faire preuves de cohésion et d'entraide lors de la conception du projet. Cependant, nous nous sommes répartis les tâches afin de respecter au mieux le délai imposé :

- Mathis : Stratégie 2, Stratégie 3...
- Junior : Multi-threading, Stratégie 1...
- Nathan : Interface graphique (IHM), navigations entre fenêtres, affichages, documentation latex...

## 6 Conclusion et perspectives

Dans cette section, nous résumons la réalisation du projet et nous présentons également les extensions et améliorations possibles du projet.

### 6.1 Résumé du travail réalisé

Le cahier des charges est globalement respecté (point de vue moteur, IHM, algo, multi-threading).

### 6.2 Améliorations possibles du projet

Comme améliorations possibles du projet, nous aurions pu ajouter davantages de stratégies, avoir la possibilité de pouvoir créer un compte pour chaque joueur de notre logiciel afin de pouvoir récupérer le contenu de sa bourse d'or. Nous aurions également pu améliorer l'IHM graphique vers une IHM graphique en 3D avec des animations et dessins réalisés à la main.

### 6.3 Derniers mots

Nous avons trouvé très intéressant de travailler sur la conception d'un jeu de simulation. Ce projet nous a permis de savoir comment fonctionne un logiciel avec plusieurs threads, ceci est très intéressant dans un cadre professionnel comme personnel. Et cela nous a également permis d'évoluer dans le monde de la programmation en général ainsi que l'utilisation et l'application de la programmation multi-threading pour le développement d'un logiciel en Java. De plus la répartition des tâches personnelles et les séances de travail en commun nous ont montré l'importance du travail en équipes. Nous en sommes très fiers.