Dossier du Projet (19-20)
Groupe : Steve Xu,Romain Carré & Adham Awad.

Architecture :

Nous avons décidé d'intégrer dans notre package model la classe centrale du jeu : Battlefield. Cette classe permet de gérer la liste des ennemis, la liste des tours ainsi que la liste des projectiles. Elle gère aussi les points de vie du joueur ainsi que son argent. Elle a pour attribut trois autres classes essentiel : WaveManager, Terrain et Graph.

La classe WaveManager est utilisée pour gérer le nombre d'ennemis lors des différentes vagues qui surviennent lors d'une partie. Elle intègre le taux d'apparition des différents ennemis ainsi que le premier tour ou ce type d'ennemi apparaît. Elle permet également de savoir à quel tour nous sommes et quel est le nombre de tours maximum avant la victoire du joueur.

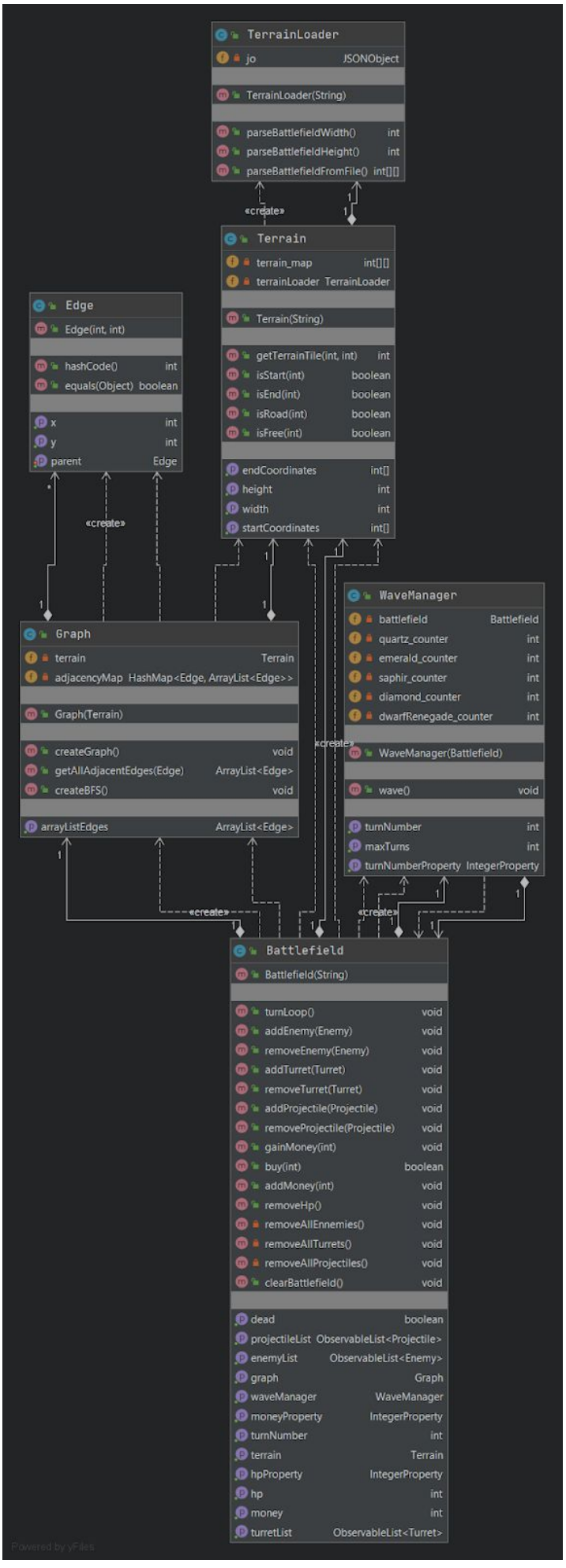
La classe Terrain permet d'avoir le terrain sur la forme d'un tableau d'entier, ainsi que sa longueur et sa hauteur. Le tableau est créé dynamiquement à partir d'un fichier JSON grâce à la classe TerrainLoader. Le tableau obtenu contient différents ids qui correspondent à l'emplacement de la tuile sur un tileset. Il existe une classification des ids qui permet de savoir quel est le type d'une case :

- 1 à 100 : chemins
- 101 à 200 : Nains
- 201 à 300 : Minéraux
- 301 à 400 : Projectiles
- 401 à 500 : Environnements
- 501 à 600 : Case libre

La classe Graph quant à elle modélise le terrain grâce à un graphe en utilisant une liste de sommets modélisés par la classe Edge. Cela permet d'utiliser plusieurs algorithmes de recherches de chemin notamment le BFS. Il permet aux ennemis de suivre un chemin dynamiquement peu importe le terrain.

L'architecture est pensée pour permettre à tout moment de changer de map et d'avoir le moins de choses à modifier possible, c'est pour celà que les chemins sont suivis dynamiquement et le terrain créé dynamiquement à partir d'un fichier json. Toutes les constantes sont gérés dans une classe appeler mRUtil, ainsi le tour maximum de jeu, le tour d'apparition, le taux d'apparition, les dégâts des différents projectiles, la classification des ids ou encore les différentes caractéristiques de tous les ennemis peuvent être modifiés en n'utilisant que ce fichier.

Nous avons créé trois sous-packages dans model : projectile, turret et enemy.
Chaque package contient une classe abstraite qui définit le comportement global de chacunes des sous-classes disponibles dans le package. Par exemple le package projectile contient une classe Projectile qui permet de modéliser l'ensemble des projectiles. La classe Potion qui hérite de la classe projectile se situe dans le même package et adopte un comportement en plus de son comportement de base hérité de Projectile. Il en va de même pour tous les autres classes dans ce package.

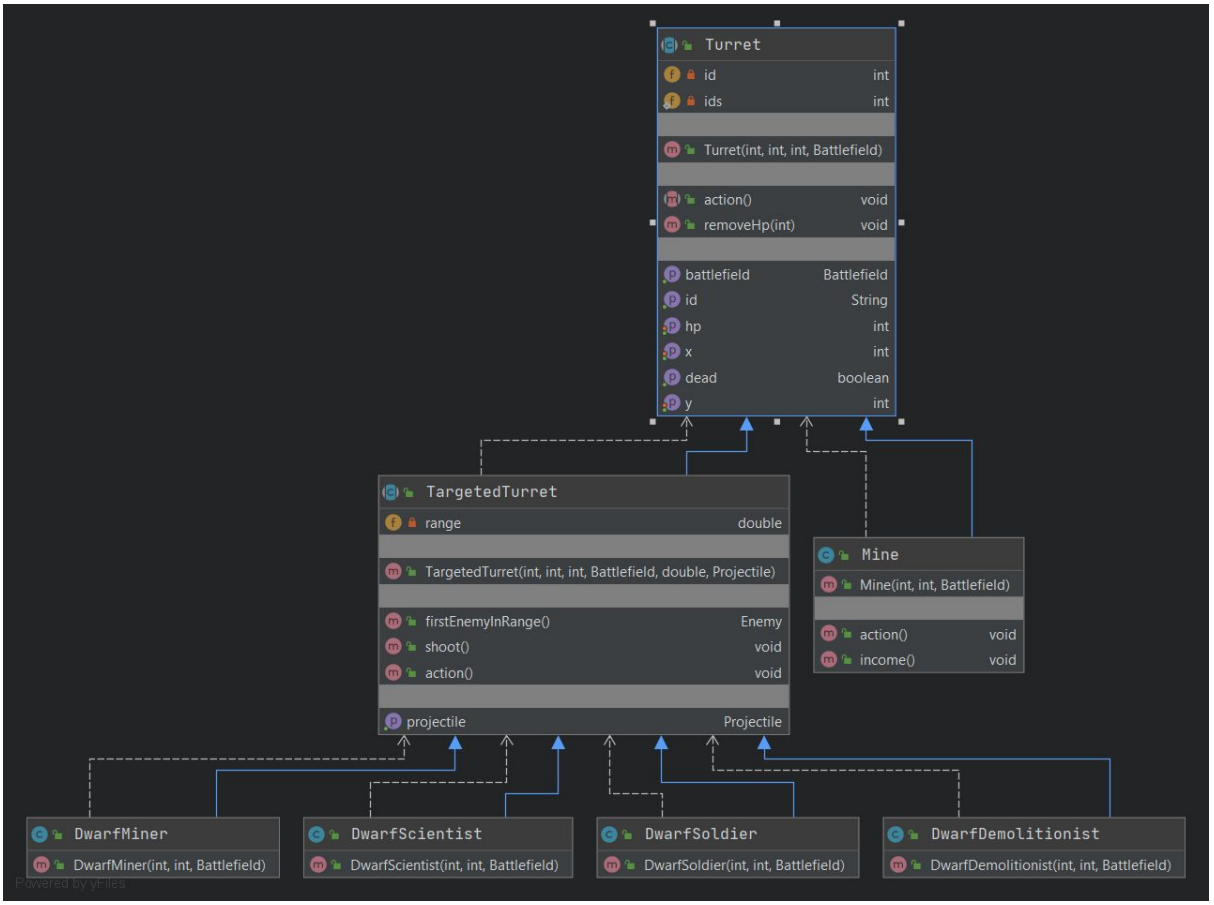

TerrainLoader
jo JSONObject
TerrainLoader(String)
parseBattlefieldWidth() int
parseBattlefieldHeight() int
parseBattlefieldFromFile() int[][]
«create»
1
1
Terrain
terrain_map int[][]
terrainLoader TerrainLoader
Terrain(String)
getTerrainTile(int, int) int
isStart(int) boolean
isEnd(int) boolean
isRoad(int) boolean
isFree(int) boolean
endCoordinates int[]
height int
width int
startCoordinates int[]
Edge
Edge(int, int)
hashCode() int
equals(Object) boolean
x int
y int
parent Edge
*
«create»
1
1
1
WaveManager
battlefield Battlefield
quartz_counter int
emerald_counter int
saphir_counter int
diamond_counter int
dwarfRenegade_counter int
WaveManager(Battlefield)
wave() void
turnNumber int
maxTurns int
turnNumberProperty IntegerProperty
Graph
terrain Terrain
adjacencyMap HashMap<Edge, ArrayList<Edge>>
Graph(Terrain)
createGraph() void
getAllAdjacentEdges(Edge) ArrayList<Edge>
createBFS() void
arrayListEdges ArrayList<Edge>
«create»
1
«create»
«create»
1
1
1
1
1
1
Battlefield
Battlefield(String)
turnLoop() void
addEnemy(Enemy) void
removeEnemy(Enemy) void
addTurret(Turret) void
removeTurret(Turret) void
addProjectile(Projectile) void
removeProjectile(Projectile) void
gainMoney(int) void
buy(int) boolean
addMoney(int) void
removeHp() void
removeAllEnnemies() void
removeAllTurrets() void
removeAllProjectiles() void
clearBattlefield() void
dead boolean
projectileList ObservableList<Projectile>
enemyList ObservableList<Enemy>
graph Graph
waveManager WaveManager
moneyProperty IntegerProperty
turnNumber int
terrain Terrain
hpProperty IntegerProperty
hp int
money int
turretList ObservableList<Turret>
Powered by yFiles

En ce qui concerne la vue elle est gérée par une classe appelée BattlefieldView. Cette classe a pour attribut la classe Battlefield, l'objet Pane et l'objet Tilepane.
Le terrain est créé dynamiquement en créant des tuiles 32*32 depuis le tileset grâce aux ids de la classe Terrain. Les tuiles sont ensuites placées dans le Tilepane. La classe BattlefieldView s'occupe également d'afficher les ennemis, les projectiles ainsi que les tours en allant chercher l'image correspondante en 32*32 dans le tileset mais cette fois ci les images sont placés sur le pane. L'animation des ennemis et des projectiles utilise la classe Timeline de javaFX, elle permet de gérer les propriétés d'un objet javaFX en fonction du temps. Par exemple, on peut dire à l'image d'un ennemi qu'à la seconde 0 il doit être à sa position de base puis à la seconde 1 à sa position d'arrivée. L'animation image par image en temps réel est alors créée.

Diagrammes de classe

Tourelles
Du côté des tourelles, la super classe abstraite est Turret. Elle a comme attributs id, hp, x et y des entiers et battlefield de type Battlefield.
Le constructeur de cette dernière prend en argument (hp, x, y battlefield).
Dans cette classe, on a une méthode abstraite void action(), void removeHp(int) ainsi que des setters et getters.
Ensuite, la classe abstraite TargetedTurret vient extends Turret ayant pour constructeur TargetedTurret(x,y,battlefield, range, projectile). Ces tourelles vont avoir la capacité d'attaquer. Ainsi on a les attributs projectile et range en plus.
C'est d'ailleurs ici où on a les méthodes action(), shoot() et firstEnemyInRange() qui vont permettre aux tourelles d'attaquer les ennemis.
On a donc 4 sous classes qui vont extends cette dernière: DwarfMiner, DwarfSoldier, DwarfDemolitionist et DwarfScientist. Elle vont toutes avoir les mêmes arguments que TargetedTurret.

Et finalement, on a une classe Mine à part qui extends Turret qui admet les mêmes arguments que TargetedTurret toutefois n'étant pas une tourelle offensive elle n'extends donc pas celle-ci. Son seul et unique intérêt est de faire appel à la méthode income() qui rajoute de la monnaie au joueur.



Diagrammes de séquence

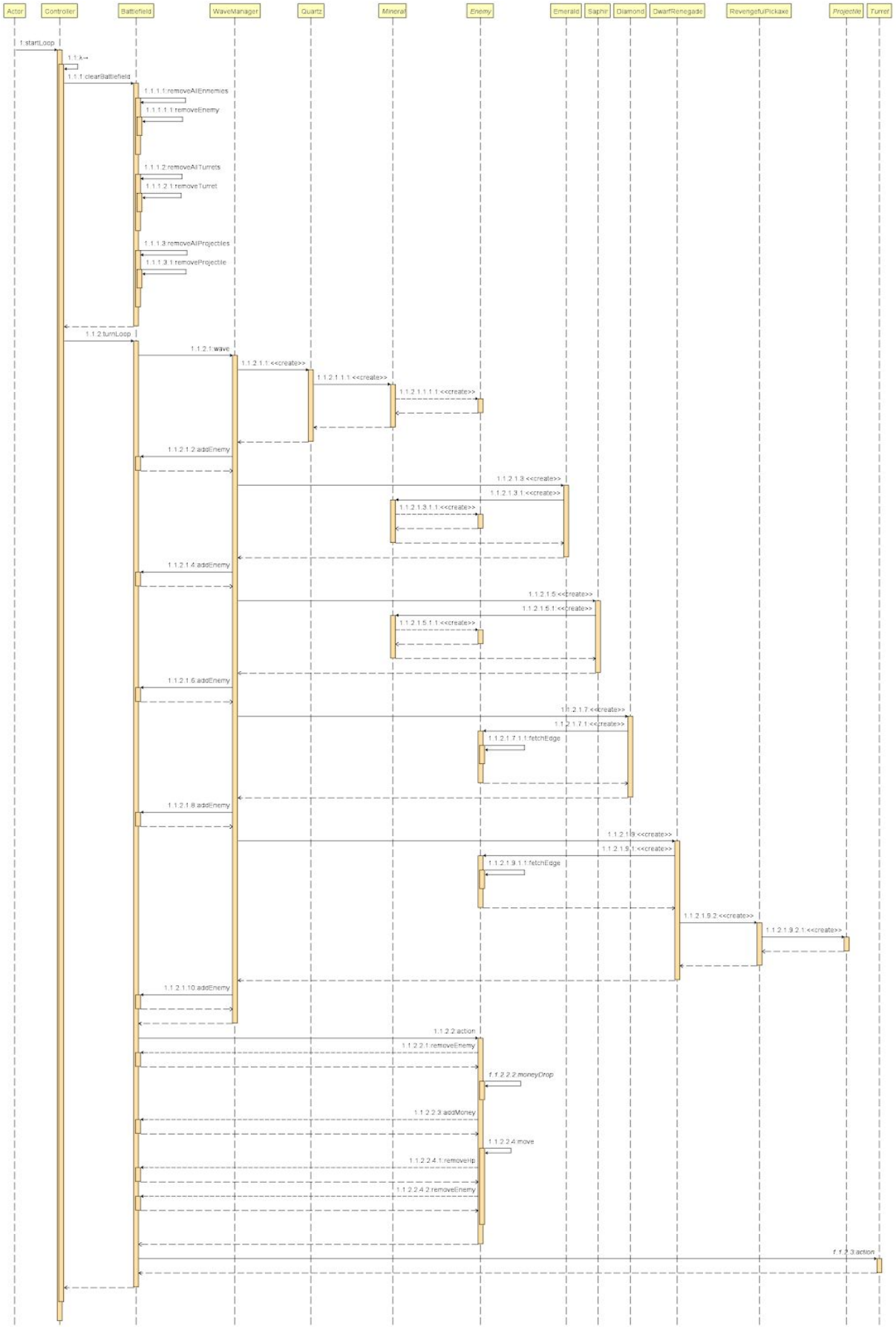
Lorsque la méthode startLoop() est appelée, elle va d'abord créer une keyFrame dans laquelle lorsque que le laps indiqué (ici 0,5s) est écoulé un événement est déclenché.

Dans cette événement, si le nombre de point de vie est égal à 0 ou le nombre de tour maximal est atteint, on appelle la méthode clearBattlefield() dans Battlefield.
clearBattlefield() permet de supprimer tous les éléments des listes crées auparavant à l'aide de méthodes présentant dans cette dernière telles que removeAllEnemies etc... C'est notamment la méthode qui met fin au jeu, avec donc un affichage adéquat selon si le joueur a perdu ou non.

Si la condition dans l'événement n'est pas respectée, dans ce cas turnLoop() de Battlefield est appelée. C'est cette méthode qui va permettre de faire dérouler les vagues grâce à wave() de waveManager. wave() va simplement créer des instances de classes en fonction du nombre de tour. Typiquement, chaque fois que la condition est respectée, on va créer le quartz(x, y, battlefield), qui va appeler le constructeur de la classe abstraite Mineral qui elle-même va utiliser celui de Enemy où l'on retrouve la méthode fetchEdge() permettant d'attribuer le sommet du graphe sur lequel il est.

Cela va être quasiment identique dans les autres cas à l'exception du Nain Renégat auquel on affecte un projectile (qui est le RevengefulPickaxe(x,y)).

Après que wave() se termine, deux boucles sont exécutées à la suite. Dans ces deux boucles, les méthodes action() des ennemis et des tourelles sont effectuées respectivement. Ainsi tous les ennemis vont effectuer leurs actions. Si un ennemi meurt, cela enclenche la méthode moneyDrop() de la sous classe correspondante qui retourne un entier qui est alors ajouté au battlefield avec addMoney(int).
Sinon, la méthode move() de Enemy ou de la sous classe est effectué. Dans tous les cas, si l'ennemi est arrivé à la fin du graphe on retire 1 point de vie au joueur à l'aide de removeHp() de Battlefield et on retire juste ensuite l'ennemi de la liste.
Les tourelles vont à leur tour effectuer leurs actions.


Actor
Controller
Battlefield
WaveManager
Quartz
Mineral
Enemy
Emerald
Saphir
Diamond
DwarfRenegade
RevengefulPickaxe
Projectile
Turret
1:startLoop
1.1:A
1.1.1:clearBattlefield
1.1.1.1:removeAllEnnemies
1.1.1.1.1:removeEnemy
1.1.1.2:removeAllTurrets
1.1.1.2.1:removeTurret
1.1.1.3:removeAllProjectiles
1.1.1.3.1:removeProjectile
1.1.2:turnLoop
1.1.2.1:wave
1.1.2.1.1:<<create>>
1.1.2.1.1.1:<<create>>
1.1.2.1.1.1.1:<<create>>
1.1.2.1.2:addEnemy
1.1.2.1.3:<<create>>
1.1.2.1.3.1:<<create>>
1.1.2.1.3.1.1:<<create>>
1.1.2.1.4:addEnemy
1.1.2.1.5:<<create>>
1.1.2.1.5.1:<<create>>
1.1.2.1.5.1.1:<<create>>
1.1.2.1.6:addEnemy
1.1.2.1.7:<<create>>
1.1.2.1.7.1:<<create>>
1.1.2.1.7.1.1:fetchEdge
1.1.2.1.8:addEnemy
1.1.2.1.9:<<create>>
1.1.2.1.9.1:<<create>>
1.1.2.1.9.1.1:fetchEdge
1.1.2.1.9.2:<<create>>
1.1.2.1.9.2.1:<<create>>
1.1.2.1.10:addEnemy
1.1.2.2:action
1.1.2.2.1:removeEnemy
1.1.2.2.2:moneyDrop
1.1.2.2.3:addMoney
1.1.2.2.4:move
1.1.2.2.4.1:removeHp
1.1.2.2.4.2:removeEnemy
1.1.2.3:action

Structures de données :
La structure de données Map et plus particulièrement hashMap est utilisée dans la classe BattlefieldView lorsqu'il s'agit d'associer une certaine tuile depuis un tileset vers la vue qui est proposé au joueur. Les ids sur le terrain sont utilisés en tant que clé dans la hashmap. Par exemple, lorsqu'on a un rocher qui a pour id : 523. On cherche dans la hashmap avec la clé : 523. On peut alors récupérer l'image du rocher et l'afficher. Cela marche pour plusieurs rochers et pour tous les éléments tant qu'ils ont un id. Cela permet de d'abord remplir la hashmap avec les images dont a besoin en parcourant le terrain, puis de remplir la vue avec les images correspondantes en évitant de les recréer à chaque fois qu'on croise le même élément.
Cette structure de données est aussi utilisée dans la classe Graph pour permettre de retrouver la liste des sommets adjacents à un sommet en fonction de ses coordonnées.

Exception :
La classe Controller contient une gestion d'exception. L'erreur est créée dans la classe TerrainLoader lorsque que le fichier JSON utilisé pour créer la map est corrompu. Une exception est alors créée et récupérée dans le controller. La gestion de cette exception consiste à afficher le message "Loading of Terrain failed" sur l'écran du joueur.

Utilisation maîtrisée d'algorithmes intéressants :
Notre principal algorithme intéressant est le BFS (Breadth-first search) dans la classe Graph. Il nous permet d'attribuer à chaque sommet un parent. Les ennemis se déplacent toujours par rapport au parent de leur sommet actuel et ainsi de suite jusqu'au seul sommet sans parent le sommet qui représente la fin du chemin.

Junits :
Les classes couvertes par les tests Junits sont Enemy et Projectile. Pour la classe Enemy, nous testons l'armure du diamant qui l'empêche de prendre des dégâts tant qu'il a son armure en simulant une attaque de chacun des nains (tourelles) sur lui. Nous testons également la compétence spéciale de l'émeraude qui est de se séparer en deux enfants lors de sa mort en simulant sa mort. Le dernier test sur la classe Enemy concerne le renvoi de dégâts de la part du saphir, nous le testons en simulant une attaque de la part de chaque nain sur un saphir.

Pour la classe Projectile, nous testons le recul infligé aux ennemis par la dynamite en simulant un tire du nain démolisseur sur un quartz et en regardant ses nouvelles coordonnées. Nous testons ensuite les dégâts de zone de la roquette du nain soldat en simulant un tir sur 4 ennemis disposés sur des cases différentes. Enfin nous testons la destruction de l'armure du diamant par la potion du nain scientifique en simulant un tir du nain scientifique sur un diamant pour détruire son armure puis un autre pour vérifier qu'il peut maintenant prendre des dégâts.

Document utilisateur :

Après des années d'exploitation et de torture, les pierres précieuses se rebellent et tentent de renverser le monde pour prendre leur revanche. Pour éviter ce drame, les nains s'allient pour les en empêcher. Grâce à leur savoir-faire, les nains sont capables de façonner des armes adaptées et se spécialisent contre ces dernières.

- Tourelles agressives:
  - Nain mineur:
    
    Ces nains lancent des pioches mono-cible qui causent des dégâts faibles. Ils ont aussi peu de vie. Cependant, ils ont l'avantage d'avoir un prix peu élevé.
  - Nain Démolisseur:
    
    Les dynamites envoyées par ce nain ont la capacité de repousser l'ennemi tout en faisant des dégâts moyens. Cette capacité est extrêmement pratique pour temporiser et permettre aux nains d'avoir le temps de se préparer contre les minéraux. Il a un prix très élevé et a une vie moyenne.
  - Nain Militaire:
    
    Le nain militaire utilise des rockettes. Elles font des dégâts à l'impact et l'explosion fait des dégâts de zone qui sont réduits en fonction de la distance. Les dégâts de zones ne sont pas renvoyés par les Saphir, ce nain constitue donc une bonne réponse à ce type d'ennemi. Ils ont beaucoup de vie mais coûtent cependant cher.
  - Nain Scientifique:
    
    Ces nains se sont spécialisés dans la chimie afin de réussir à détruire les minerais les plus puissants. Le nain scientifique est en effet le seul nain capable de faire perdre son armure à un diamant. Il a des points de vie moyens et un prix moyen.

- Tourelles passives:
  - Mine d'or:
    
    La mine d'or est une structure coûteuse qui produit de l'argent de manière passive pour obtenir un complément de revenu sur l'argent récupéré sur les ennemis. Elle ne peut être détruite que par les nains renégats il est donc conseillé de la placer loin du passage des ennemis.

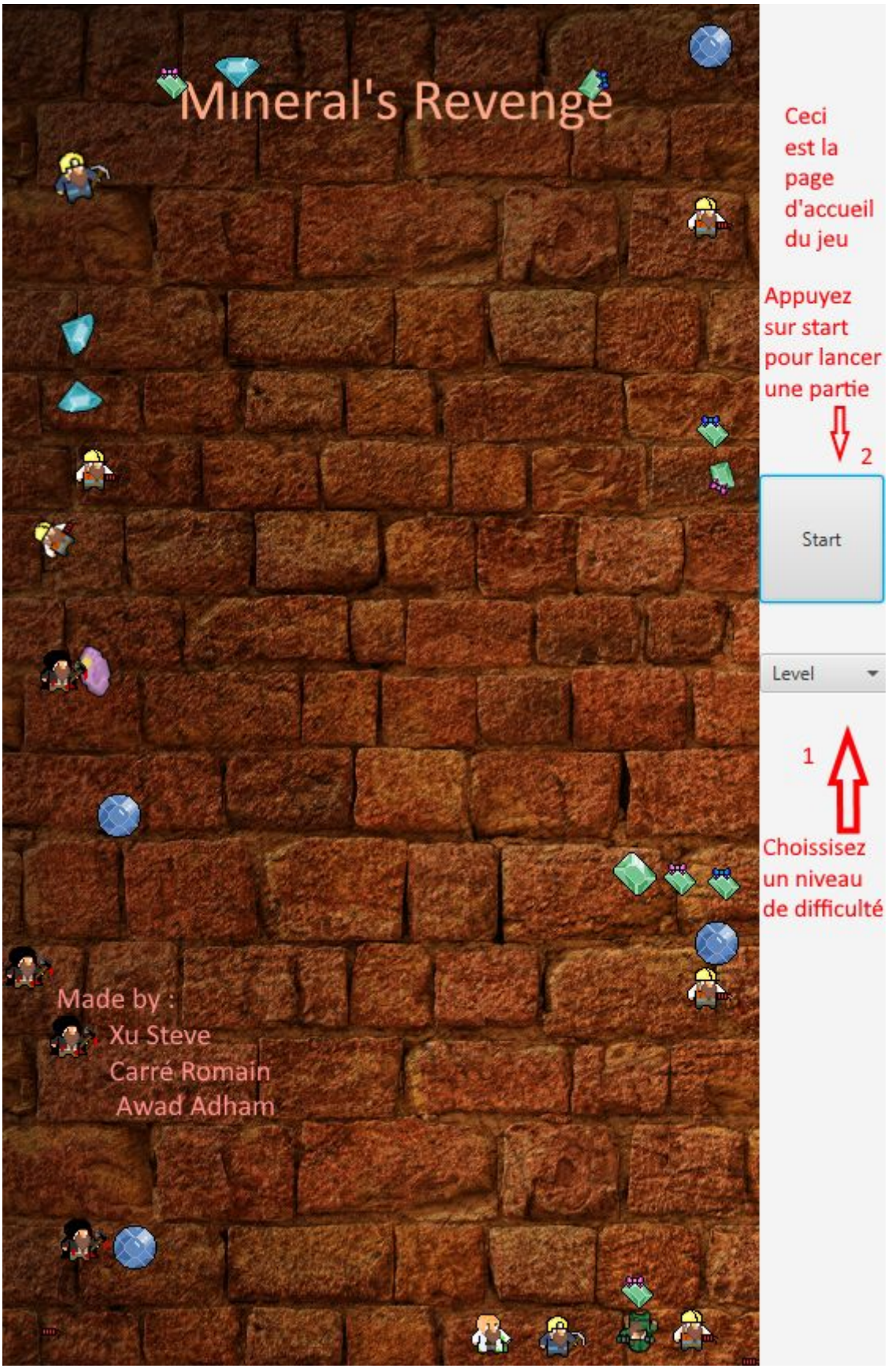
Ennemis:
Classés par ordre croissant en fonction de leur dureté d'après l'échelle de Mohs.

- Quartz:
  
  Ennemi standard et faible, ils n'ont que le nombre pour eux. Il donne un peu d'argent.
- Emeraude:
  
  Cette pierre précieuse à la particularité de donner deux enfants lorsqu'elle meurt. Leurs pvs sont réduits de moitié. Elle donne très très peu d'argent mais ses enfants en donnent aussi.
- Saphir:
  
  Cette pierre précieuse est fière et puissante. Elle renvoie une partie des dégâts qu'elle subit lorsqu'elle reçoit une attaque directe. C'est une unité qui a un grand nombre de pv et qui donne une quantité d'argent moyenne.
- Diamant:
  
  Ennemi tenace avec une armure résistante. Le diamant ne peut-être détruit par des dégâts tant qu'il possède une armure. Un nain scientifique est nécessaire pour réussir à la lui enlever afin qu'il prenne des dégâts. Il donne une quantité d'argent élevée.
- Nain renégat:
  
  Ce nain est un traître qui a été racheté par les pierres précieuses qui ont vendu leur camarade pour l'avoir dans leur rang. C'est un nain dangereux car il n'hésite pas à attaquer les tours. Il représente une vraie menace et peut très vite devenir un problème. Il donne une quantité d'argent très élevée.

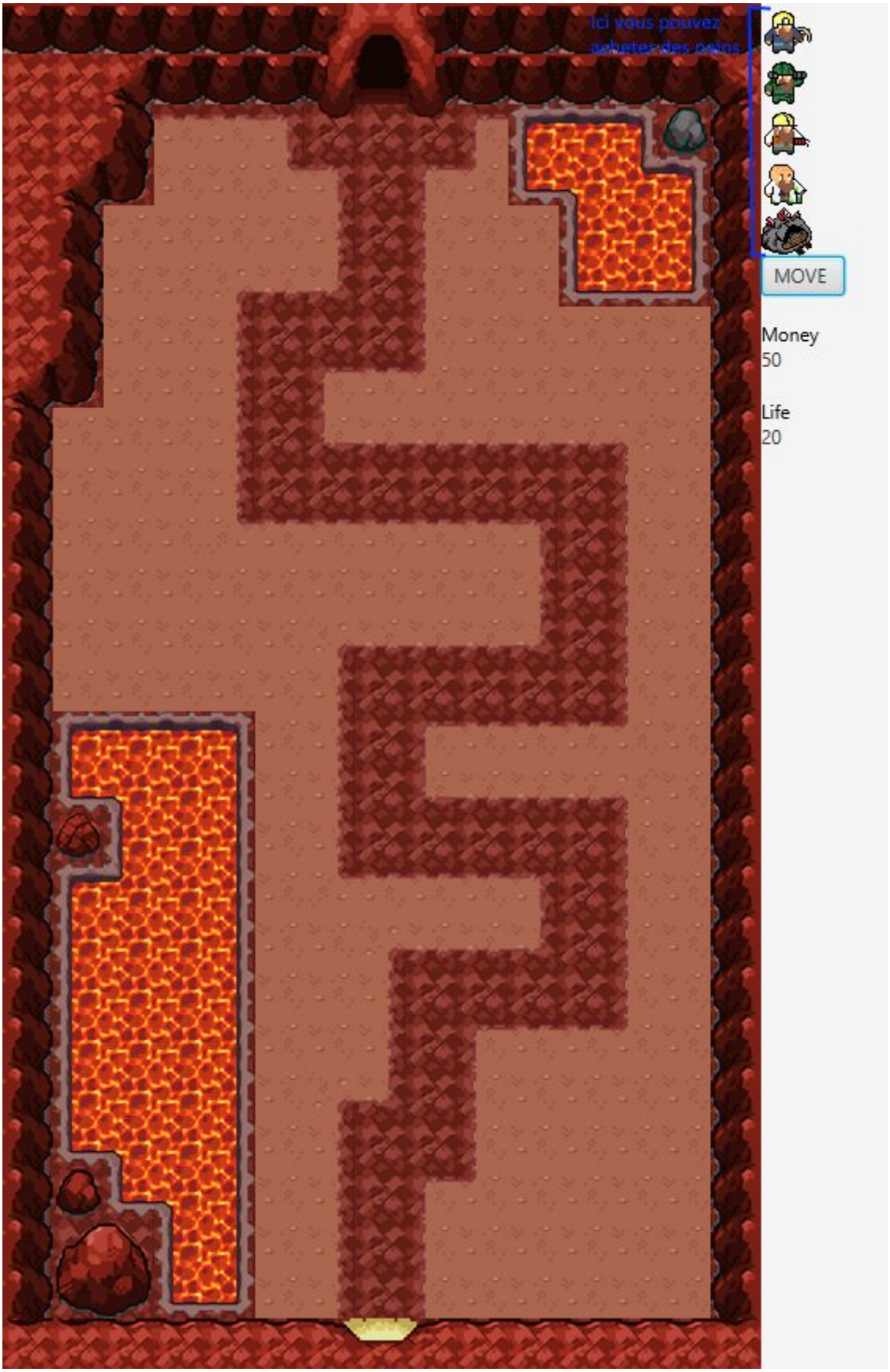
Ce tableau décrit l'utilité des nains par rapport aux différents minéraux :
Plus le nombre de "+" est important plus le nain est efficace face à l'ennemi.
Plus le nombre de "-" est important plus le nain est faible face à l'ennemi.
Si c'est un "0" le nain est inutile face à l'ennemi.

					
	+	++	++	0	0
	+	++	++++	0	0
	+	+	+++	-	0
	0	++	0	++++	0
	--	--	---	---	----

Guide de jeu :


Mineral's Revenge
Made by :
Xu Steve
Carré Romain
Awad Adham
Ceci est la page d'accueil du jeu
Appuyez sur start pour lancer une partie
2
Start
Level
1
Choissisez un niveau de difficulté


Si les enemis
arrivent ici
vous perdez
1 point de vie
Les enemis arrivent
par ici
MOVE
Money
50
Life
20


Ici vous pouvez
acheter des ogres
MOVE
Money
50
Life
20

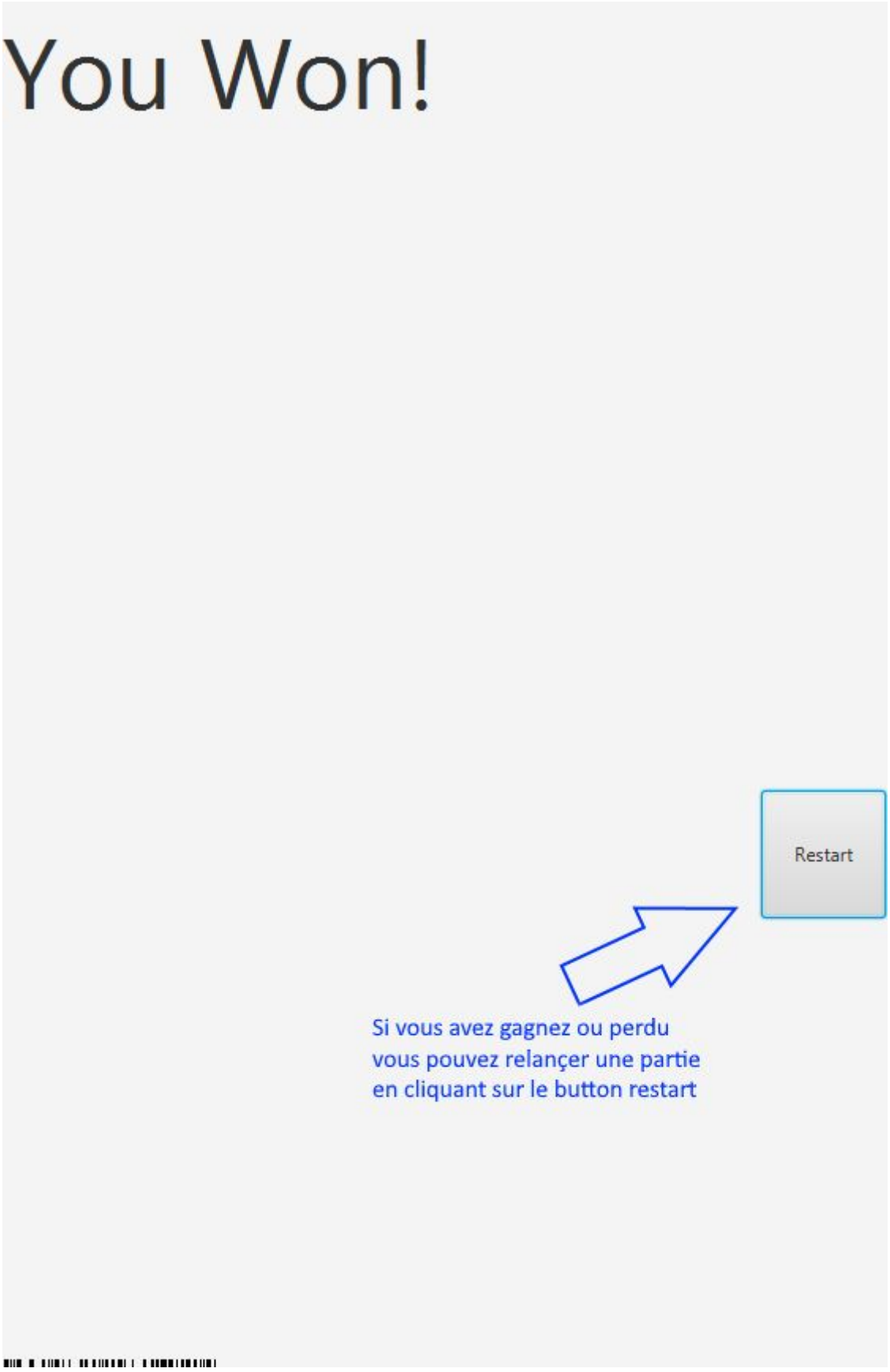

Maintenez et
glissez un nain
avec la souris
MOVE
Money
50
Life
20


MOVE
Money
35
Life
20
Lâchez avec votre souris
là ou vous voulez le placer,
les nains ne peuvent être placer
que sur des cases libres et
le placement coûte de l'argent


MOVE
Money
5
Life
20
Après avoir
placé quelques
défenses cliquez
sur move
pour commencer
la partie


MOVE
Money
5
Life
20
Vous pouvez
enlever un nain
en utilisant
le clic droit de la
souris
Attention ça ne
rends pas d'argent


MOVE
Money
17
Life
20
Les enemis arrivent et vos nains tirent. Bonne chance !


You Won!
Restart
Si vous avez gagnez ou perdu
vous pouvez relançer une partie
en cliquant sur le button restart