

Spécification et Modélisation

DWARVES mANAGER



Alexandre HAMANN, Clément HERLEDAN, Yannick GUERN, Clément PERREAU

S6P 2013

Table des matières

I Spécification	2
Introduction	3
UseCase	4
Scénario	5
II GameDesign	12
Introduction	13
Mécanique de jeu	14
Contrôles	15
Direction artistique	16
Background	17
III Spécification technique	18
Choix technologiques	19
Architecture du projet	20
Modélisation	21
Diagrammes de classe	21
Diagrammes d'activités	27
Machine à états	38

Première partie

Spécification

Introduction

Dwarves Manager est un jeu de gestion qui prend place dans un univers fantastique classique. Le joueur est aux commandes d'une mine dans laquelle des nains (créatures fantastiques de la mythologie scandinave) travaillent à extraire toute sorte de ressources. Le but du jeu est de gérer cette petite entreprise, et de l'aider à remplir ses objectifs.

Le joueur doit pour cela recruter des nains, mettre en place des structures pour répondre à leurs besoins, afin de maximiser la productivité. Il définit également les zones que les nains doivent excaver pour récolter des ressources. Le but du jeu, dans un premier prototype sera d'acquérir une certaine quantité de ressources, avant la fin d'un temps imparti.

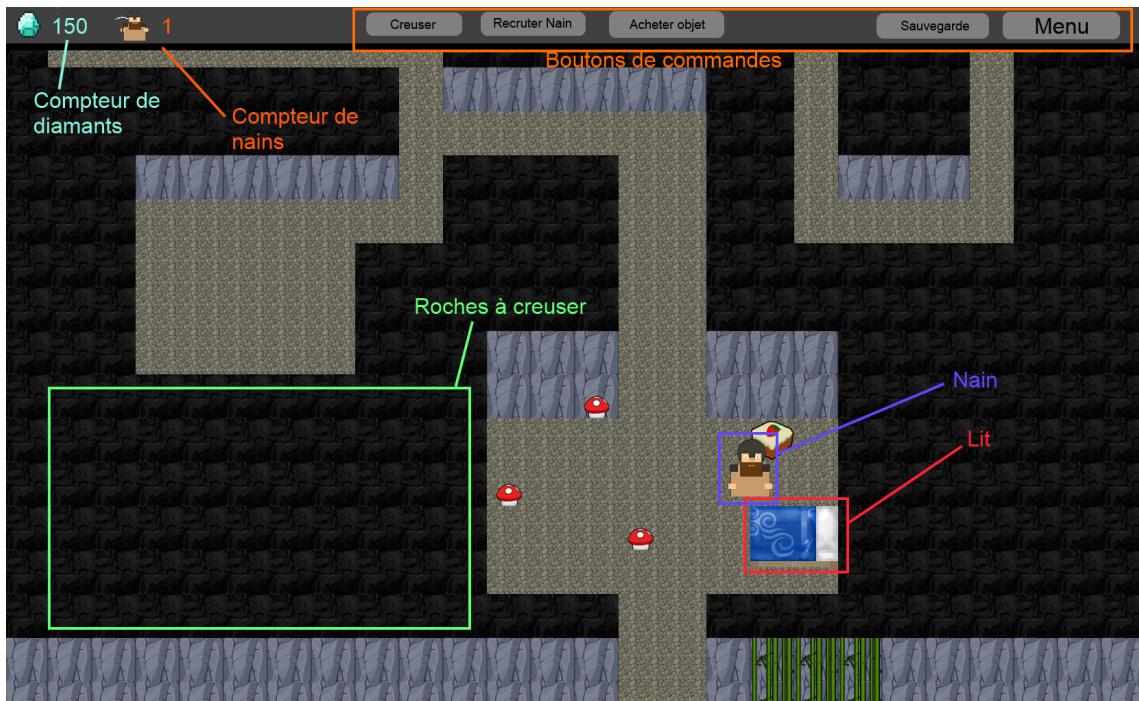
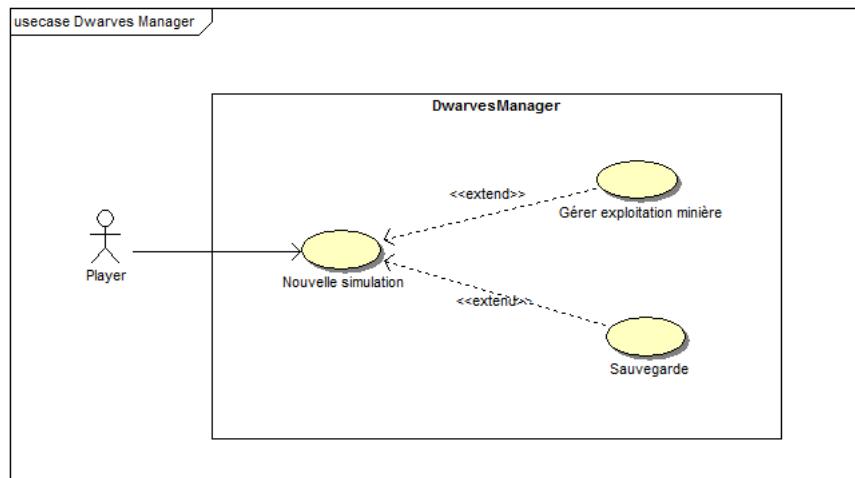


FIGURE 1: Aspect prévu du jeu

UseCase



Nouvelle simulation :

Le joueur sélectionne un scénario ou une partie enregistrée, afin de lancer une nouvelle instance de simulation.

Gérer exploitation minière :

Le joueur gère l'exploitation. Cela inclue la possibilité de recruter/renvoyer de nouveaux nains, placer des objets pour leur confort. Le joueur définit également les zones du monde à creuser.

Sauvegarde :

Sauvegarde de l'état de la simulation dans un fichier.

Scénario

Nouvelle Simulation

		Nouvelle Simulation
Sommaire d'identification		Le joueur sélectionne un scénario ou une partie enregistrée, afin de lancer une nouvelle instance de simulation.
Description scénarios	Nominal	* Le joueur choisit une scénario * Le joueur valide son choix
	Alternatif	* Le joueur choisit une partie sauvegardée * Le joueur valide son choix
	Erreur	* Le joueur choisit une partie sauvegardée * Le niveau ne peut pas être chargé(niveau inexistant)

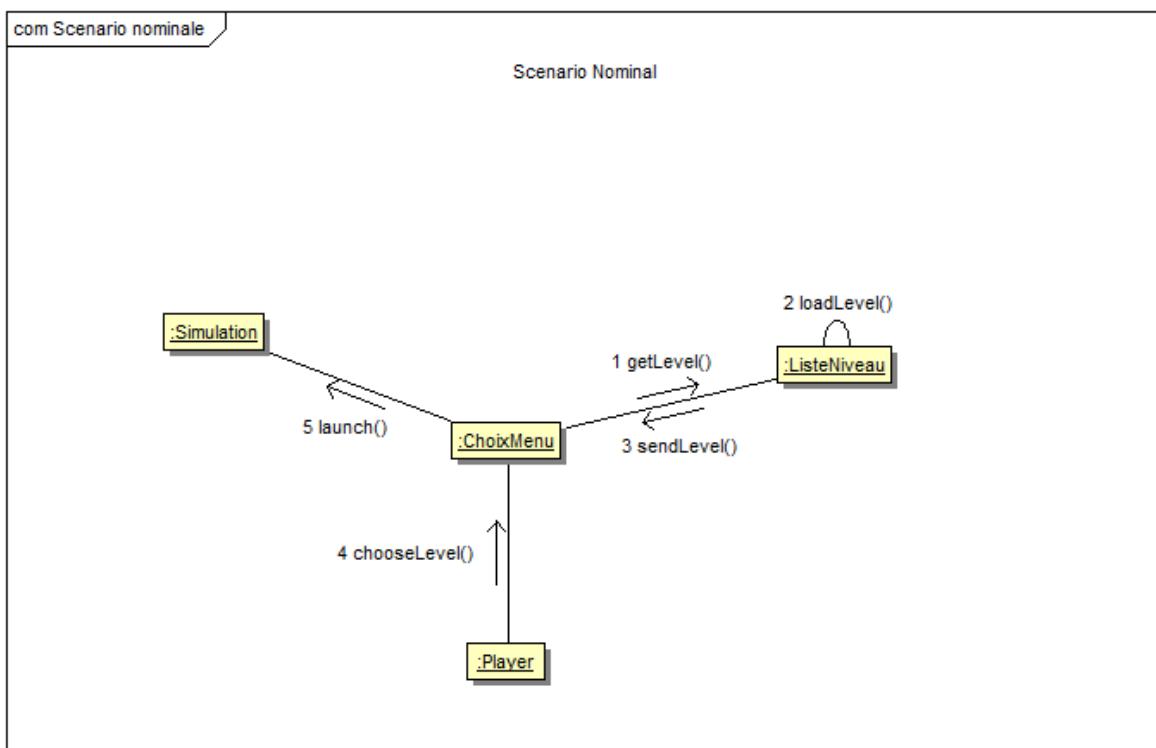


FIGURE 2: Scénario nominal

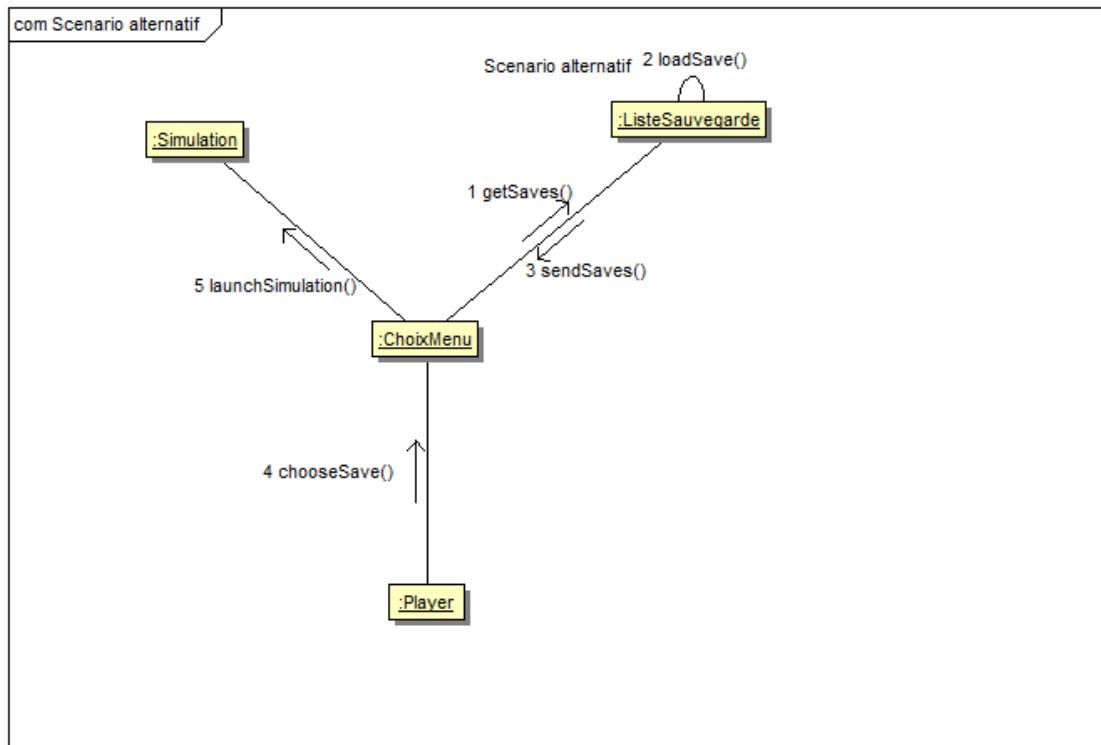


FIGURE 3: Scénario alternatif

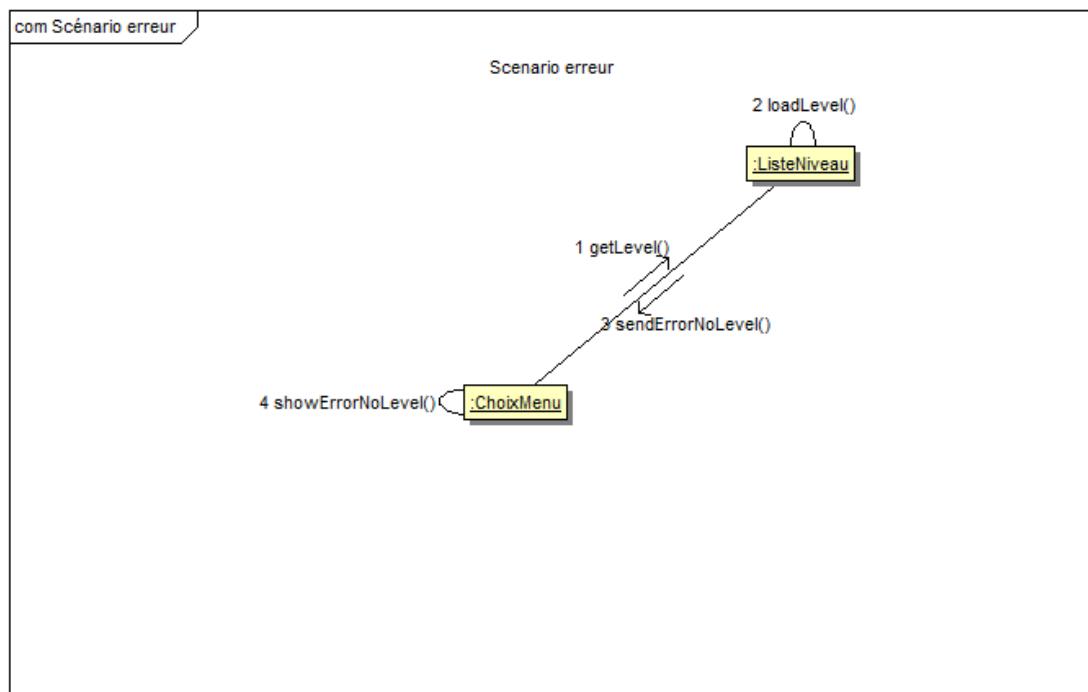


FIGURE 4: Scénario erreur

Gérer exploitation minière

Sommaire d'identification	Nouvelle Simulation	
	<ul style="list-style-type: none"> * Le joueur gère l'exploitation. * Cela inclut la possibilité de recruter/renvoyer de nouveaux nains, placer des objets pour leur confort. * Le joueur définit également les zones du monde à creuser. 	
Sommaire d'identification	Nominale	<ul style="list-style-type: none"> * Le joueur recrute un nain * Le joueur place un lit * Le joueur définit une zone à miner * Le nain commence à miner * Le nain fatigué, rejoint son lit pour dormir * Le nain repart travailler et fini de miner * Le nain récolte des diamants pour le joueur * Le joueur ayant récolté assez de diamants remporte la partie
	Alternatif	<ul style="list-style-type: none"> * Le joueur recrute un nain * Le joueur définit une zone à miner * Le nain commence à miner * Le nain fatigué refuse de continuer à travailler (pas de lit pour dormir) * Le joueur perd la partie parce que le temps qui lui était alloué est écoulé

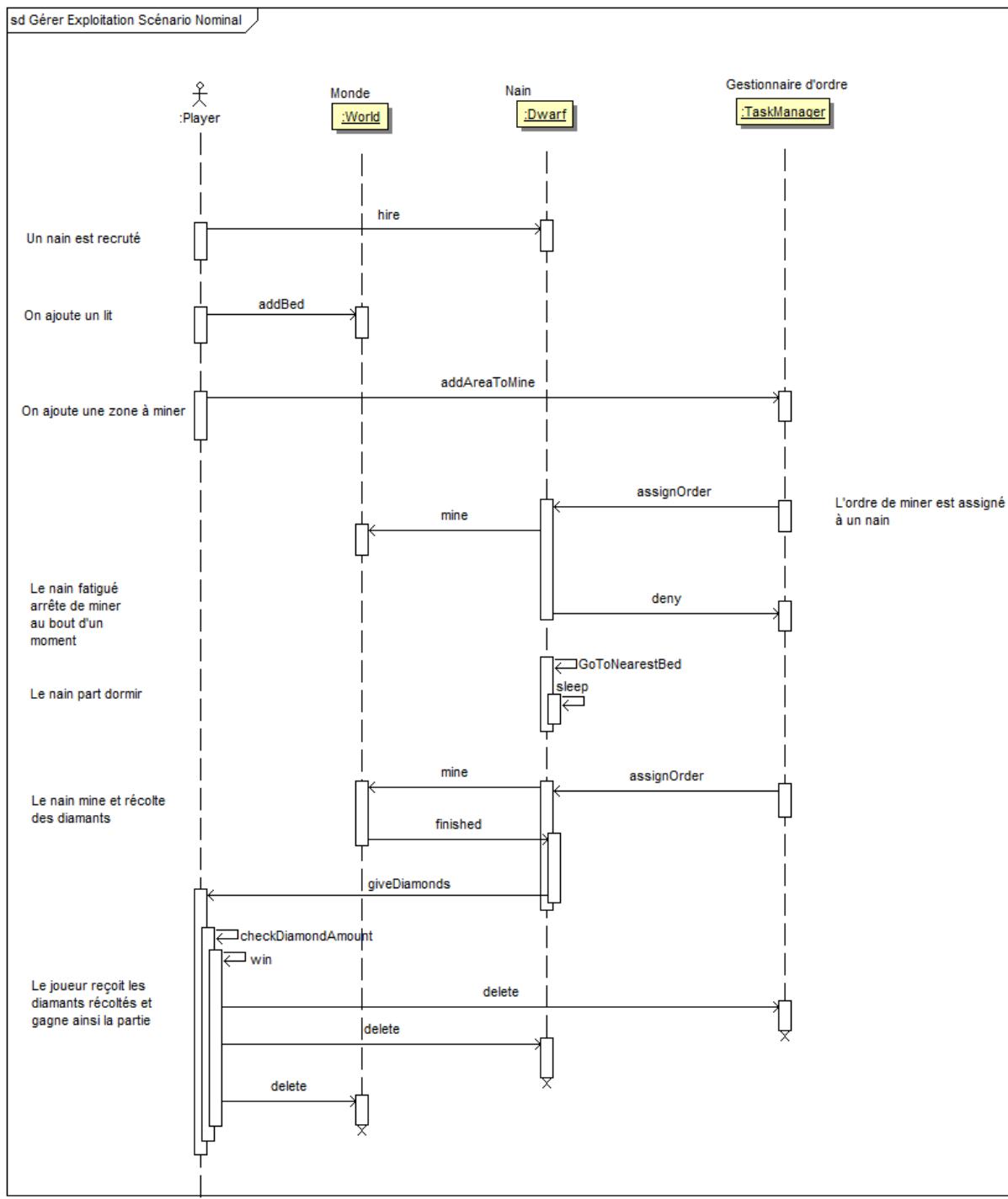


FIGURE 5: Scénario nominal

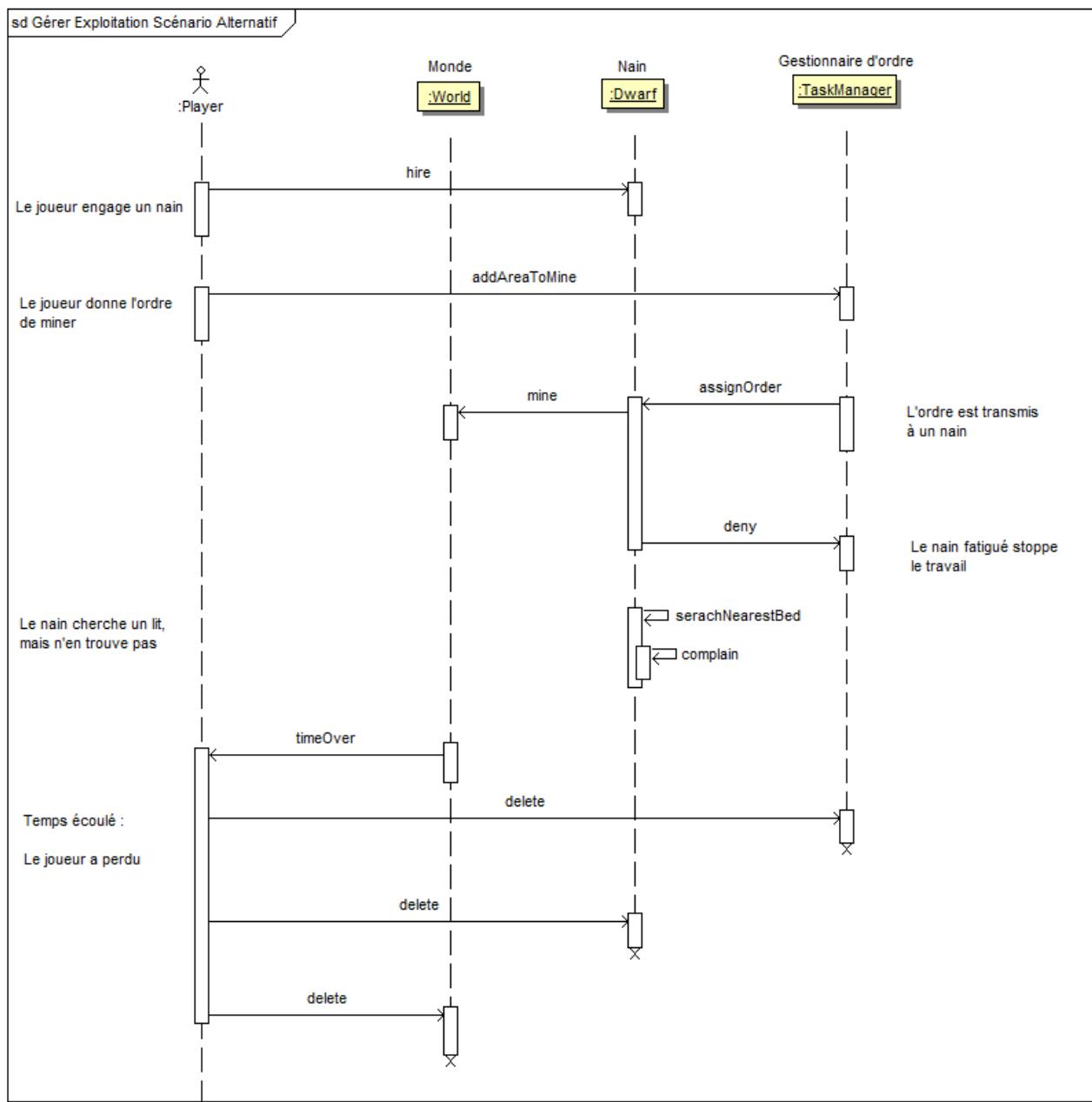


FIGURE 6: Scénario alternatif

Sauvegarde

Sommaire d'identification		Sauvegarde
Description scénarios	Nominale	Sauvegarde de l'état de la simulation dans un fichier. * Le joueur rentre un nom * Le joueur valide * Sauvegarde effectuée
	Erreur	* Le joueur rentre un nom * Le joueur valide * Échec de la sauvegarde (Nom invalide, ou chemin inaccessible en écriture)

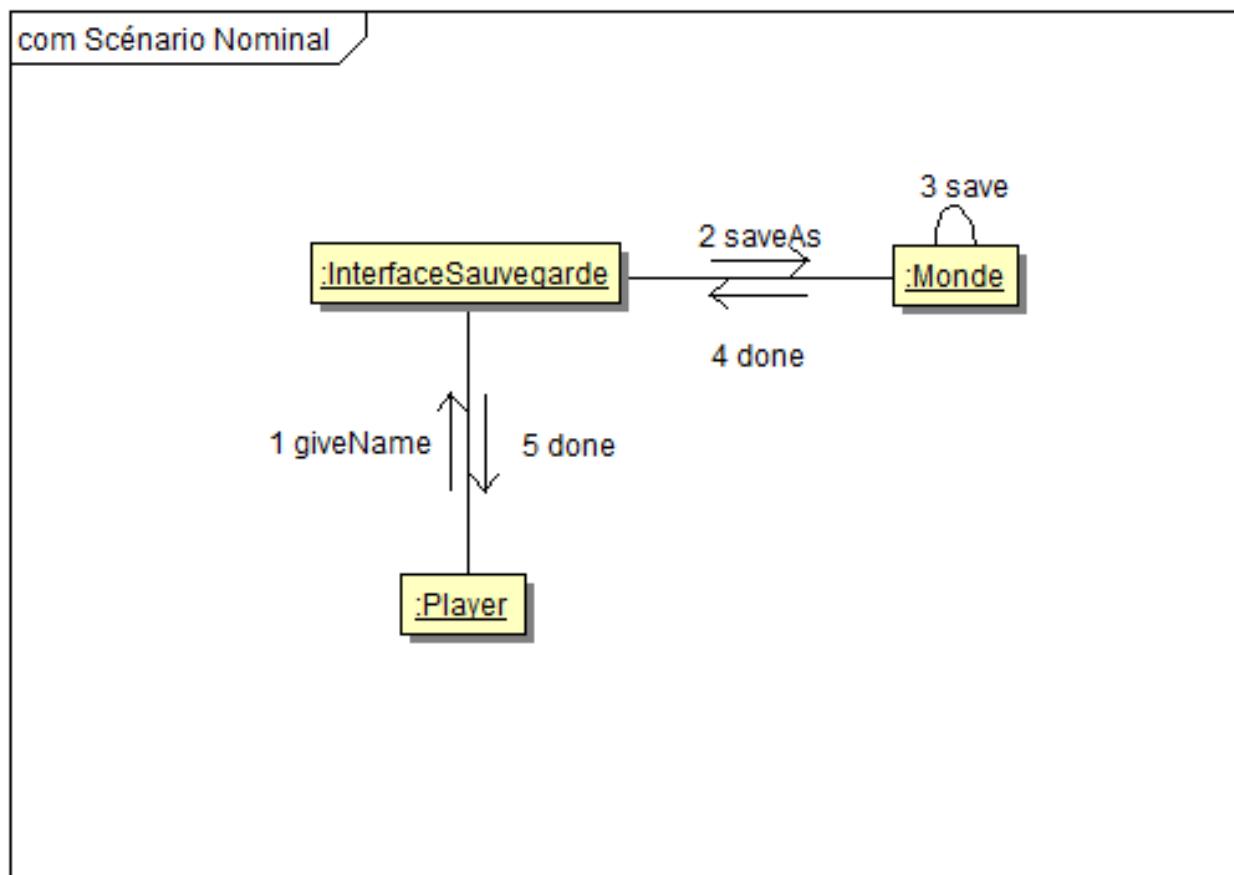


FIGURE 7: Scénario nominal

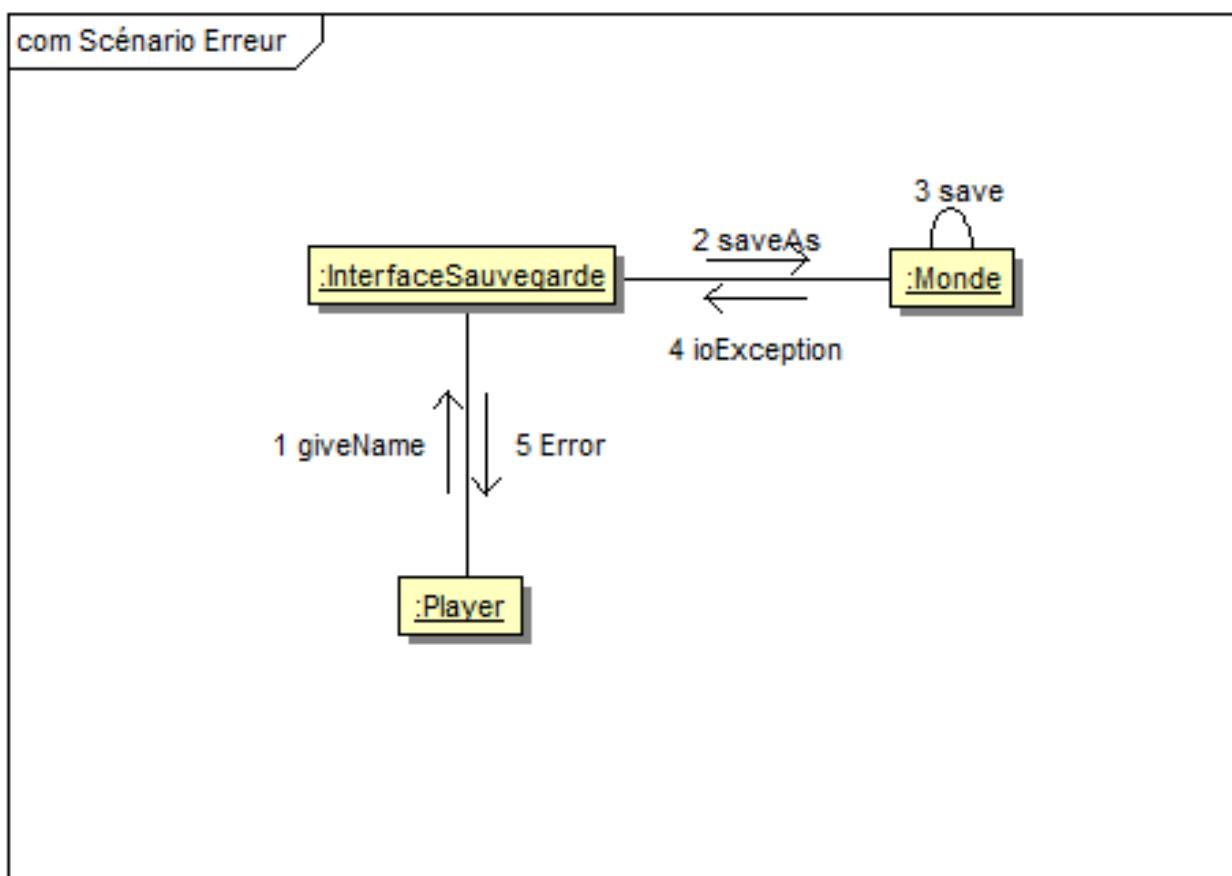


FIGURE 8: Scénario erreur

Deuxième partie

GameDesign

Introduction

Le but de cette partie est de décrire précisément les mécaniques de jeu que nous souhaitons implémenter, de décrire le style artistique et graphique du jeu, ainsi que son contexte imaginaire.

Mécaniques de jeu

PERSONNAGES :

Le nain mineur : Le nain mineur travaille à extraire les ressources dans les zones décrites par le joueur. Il a besoin de se reposer régulièrement et de se détendre à la taverne, et aussi d'une pioche en bon état pour miner.

Le nain artisan : Le nain artisan répare et fabrique des pioches pour les mineurs.

Le nain tavernier : Le nain tavernier sert les bières à la taverne

PIÈCES ET OBJETS :

Le dortoir : Le dortoir est la pièce de la mine où sont installés les lits où dormiront les nains.

La taverne : La taverne est l'endroit où viennent se détendre les nains après une dure journée de labeur. La taverne a besoin d'un comptoir, d'un fût à bière et d'au moins une table pour fonctionner.

L'atelier : L'atelier est l'antre du nain artisan. Il y a une enclume et un râtelier pour permettre au nain artisan de travailler. Avant d'aller travailler, les nains viennent y chercher leur pioche, et au retour du travail ils viennent y déposer leur pioche dans un râtelier prévu à cet effet.

LES BESOINS :

Le sommeil : Pour répondre à un manque de sommeil, le nain doit dormir dans un lit dans un dortoir.

La bière : Les nains doivent aller à la taverne boire des bières régulièrement.

Chaque nain recruté est différent et possède une certaine prédisposition à la fainéantise et à l'alcoolisme déterminée aléatoirement par une loi normale. Un nain peut donc être paresseux et passer tout son temps à dormir, ou encore passer tout son temps à boire des bières à la taverne, mais cela restera relativement rare.

GESTION DU PERSONNEL :

Le joueur est libre de renvoyer n'importe lequel des nains, néanmoins, il devra payer une prime de licenciement. De même quand on embauche un nain, il faudra payer une prime d'embauche.

Remplacer un nain fainéant par exemple peut se révéler très coûteux, surtout si son remplaçant n'est pas meilleur.

Régulièrement, il faut aussi payer les salaires, recruter trop de nains peut vite mener une exploitation à la faillite, et lorsque le mineraï s'épuise, il faut envisager de lourds plans sociaux.

Contrôles :

Les actions du joueur sont relativement limitées. Il lui est possible de recruter de nouveaux nains, de délimiter les zones à miner, de délimiter des pièces, de placer des objets de mobilier et de confort.

Toute la subtilité du jeu relève du fait que tous les contrôles sont indirects, le joueur n'a pas à sélectionner et à donner un ordre précis à chacun des personnages, il définit juste les zones de travail et aménage l'espace de ses employés.

Direction artistique :

Nous avons réalisé quelques dessins des personnages dans un style minimaliste dès le début de projet afin de pouvoir imaginer au plus tôt l'aspect final du jeu. Voir figure 9, page 16.

Néanmoins, la réalisation de toutes les ressources graphiques n'étant pas notre priorité, les personnages seront les seuls éléments que nous dessineront. Nous tenterons bien sur de réunir des ressources graphiques et sonores qui collent au mieux à l'univers du jeu.



FIGURE 9: Style des personnages

Background :

Les nains :

Le nain est une créature humanoïde imaginaire souterraine de petite taille, dont la figure actuelle est principalement issue de la mythologie nordique et des croyances germaniques médiévales.

Mythologie nordique :

Les nains (dvergr en vieux norrois) sont des créatures vivant sous terre, dans les pierres ou les montagnes, ils sont originellement des vers trouvés dans la dépouille du géant Ymir, auxquels les dieux donnent forme humaine et intelligence, mais qui du fait de leur origine continuent à vivre sous terre et dans les pierres. Les nains sont souvent rusés, et se caractérisent par leur habileté, surtout en tant que forgerons. En effet, ils sont responsables de la création de la majorité des attributs divins, dont le marteau de Thor. Les textes mythologiques ne donnent aucune information relative à la taille des nains, et il n'y a pas de raison de penser qu'ils étaient de petite taille. Cette représentation apparaît dans les sagas tardives, où ils sont décrits petits et généralement laids. La croyance en des nains caricaturaux, de petite taille et généralement malins et mystérieux, est restée dans le folklore populaire germanique après la christianisation.

Cet aspect des nains est repris dans une grande majorité d'histoire fantastique et de jeux. L'un des exemples est le seigneur des anneaux où les nains sont des créatures humanoïdes de petite taille, vraisemblablement entre 1 mètre 20 et 1 mètre 50. Ils sont robustes, font d'excellents combattants et sont dotés d'une grande résistance à la faim et à la douleur. Leur espérance de vie moyenne est de deux cent cinquante ans. Un aspect important de leur physique est leur barbe, qu'ils ne rasent jamais et qui est portée par les hommes comme par les femmes. Les Nains minent et travaillent les métaux précieux et la pierre avec un talent inégalé dans la terre du milieu. Ils sont également de grands forgerons, créateurs de nombreuses armes de légende. Ils sont aussi réputés pour l'usage de hache comme arme de prédilection.



FIGURE 10: Une représentation de nain : Gimli de la Communauté de l'Anneau

Troisième partie

Spécification technique

Choix technologiques

Nous avons choisi de développer Dwarves Manager en Java avec le framework LibGDX

Pourquoi Java ?

La portabilité de notre programme sur différents systèmes étant l'un de nos objectifs prioritaires, nous estimons que Java est le langage le plus adapté.

Pourquoi LibGdx ?

LibGdx est un framework Open Source, dont l'un des atouts principaux est d'assurer la compatibilité d'une application entre les Systèmes les plus courants (Windows, Linux, Android, MacOs, Ios, Web) sans avoir à en modifier, ou à en adapter le code.

La librairie dispose d'une bonne documentation et de toutes les fonctionnalités nécessaires pour pouvoir développer confortablement un jeu vidéo ou une application graphique complexe. Pour plus d'informations : [urlhttp://libgdx.badlogicgames.com/](http://libgdx.badlogicgames.com/)

Éditeur de contenu

Afin de concevoir les niveaux du jeu, nous avons décidé d'utiliser l'éditeur de niveau « Tiled Map Editor ». Ce logiciel libre permet de dessiner les niveaux du jeu, d'éditer leurs propriétés, de placer des objets, et d'exporter le résultat dans un format XML que nous pouvons lire avec LibGdx pour ensuite l'intégrer sans difficultés dans le jeu. Pour plus d'informations : [urlhttp://www.mapeditor.org/](http://www.mapeditor.org/)

Résumé

Langage privilégié : Java

Framework : LibGDX

Editeur de contenu : Tiled

Plateformes cibles : Windows, Linux, Android (Tablette et Smartphone)

Architecture du projet

Le projet est constitué de 4 projets Eclipse, c'est une structure recommandée, lorsque l'on utilise LibGDX en vue d'une application multi-plateforme. Voir figure 11

DwarvesManager : C'est le projet principal, qui contient tout le code de l'application, et inclus la librairie GDX.

DwarvesManager-android : C'est le projet Android, qui référence le projet principal. Il définit toutes les informations pour l'exécution de Dwarves Manager sur le système Android (Manifeste Android), contient également les sources compilées utilisées par LibGdx pour s'exécuter sur Android, ainsi que les librairies du SDK Android. Un seul fichier de code est présent, c'est une classe qui étend la classe AndroidApplication du SDK, afin de pouvoir lancer le jeu.

DwarvesManager-desktop : Le principe est le même sauf que ce projet contient un main capable d'exécuter l'application sous Windows, Linux et MacOs.

DwarvesManager-html : Même principe encore, on a ici une classe qui étend GwtApplication du framework Gwt (Google Web Toolkit), qui permet d'inclure le jeu dans une page web et de le faire exécuter par un navigateur internet.

NB : LibGDX nous avait également généré un projet IOs (Pour Iphone/Ipad), mais n'ayant pas le budget pour acheter une licence de développement, nous avons préférer abandonner la compatibilité avec cette plate-forme.

MVC :

Nous avons choisi le pattern MVC comme modèle d'architecture, car c'est une solution relativement simple qui nous permet néanmoins d'assurer une bonne extensibilité de notre programme.

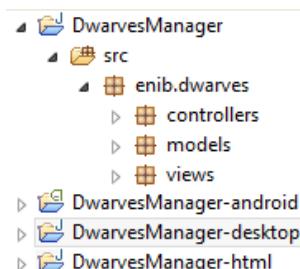


FIGURE 11: Architecture du projet

Modélisation

Diagrammes de classe

Sur les diagrammes nous ne montrons pas les liens entre les « modèles » et leurs « vues ». Nous ne détaillerons pas ici les vues. Les attributs montrés publics seront implémenté comme étant privé ou protégé dans le code selon les cas, mais disposerons d'accesseurs et modifieurs publics. Les modifieurs et accesseurs ne sont par conséquent pas non plus détaillés.

Il nous était impossible de représenter toute l'architecture sur un seul diagramme de classe, nous avons donc essayer de découper au mieux les différentes parties du projet.

class GameModels

Ce diagramme (Figure 12) présente la structure générale de notre code métier. La classe Game est la classe centrale, on l'instancie quand on souhaite créer une nouvelle simulation, elle est composé de plusieurs objets.

Un « Characters » : un objet chargé de gérer la population de la simulation

Un « Rooms » : un objet qui gère toutes les pièces créées par le joueur.

Un « Objects » : un objet qui gère tout les objets placés par le joueur.

Un « Level » : un objet qui représente le monde du jeu

Un « TaskManager » : un objet qui gérera les ordres donnés par le joueur et les attribuera aux nains.

Remarque sur class WorldView (Figure 17)

Comme on peut le voir une partie des données qui constituent le monde sont stockées dans des classes du package view. Ces classes d'affichage avaient déjà été créées par l'un d'entre nous avant le projet, et comme elles fonctionnaient bien, nous avons choisis de les réutiliser. Néanmoins, elles n'ont pas été conçues pour une architecture MVC, le but de la classe Level est donc de servir d'interface entre les modèles et les données contenues dans ces classes.

Remarque sur class Pathfinding (Figure 18)

Le pathfinder utilise l'algorithme A*, d'où cette structure.

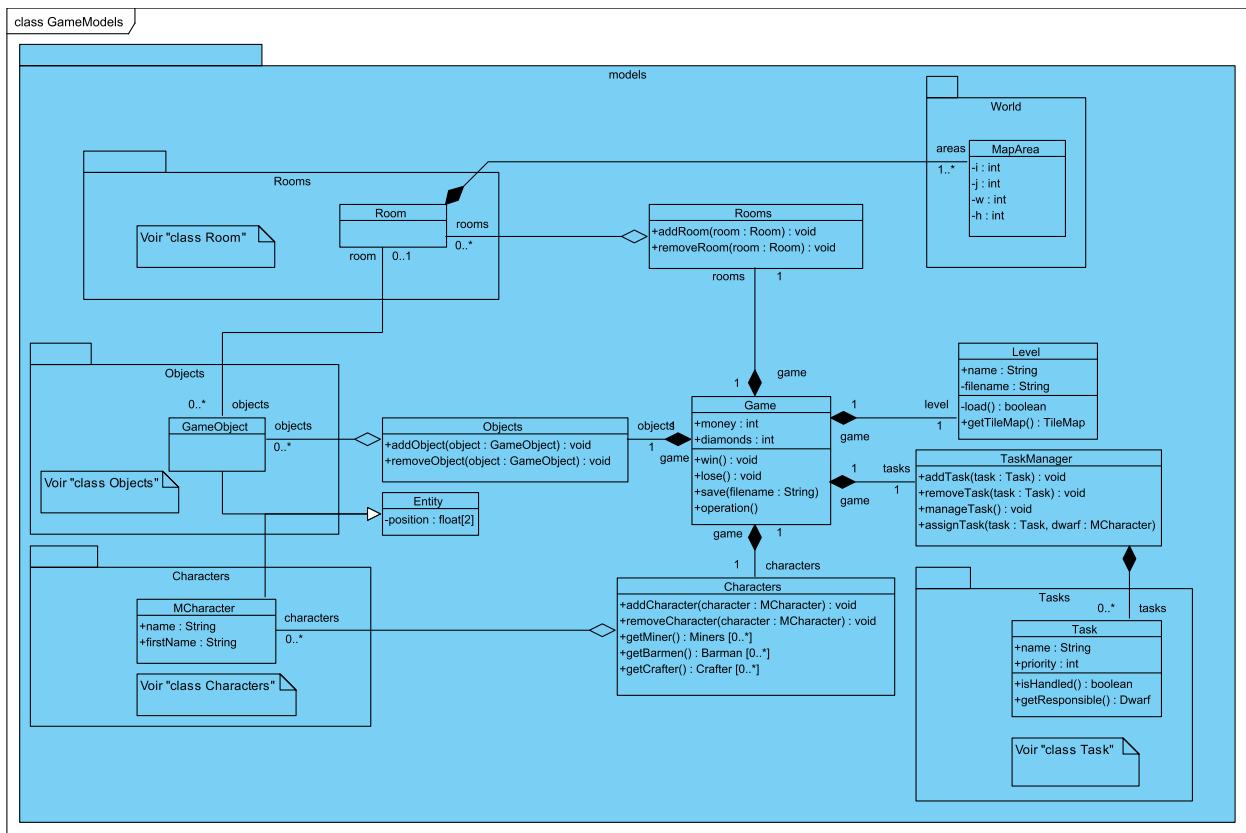


FIGURE 12: Architecture globale du code métier

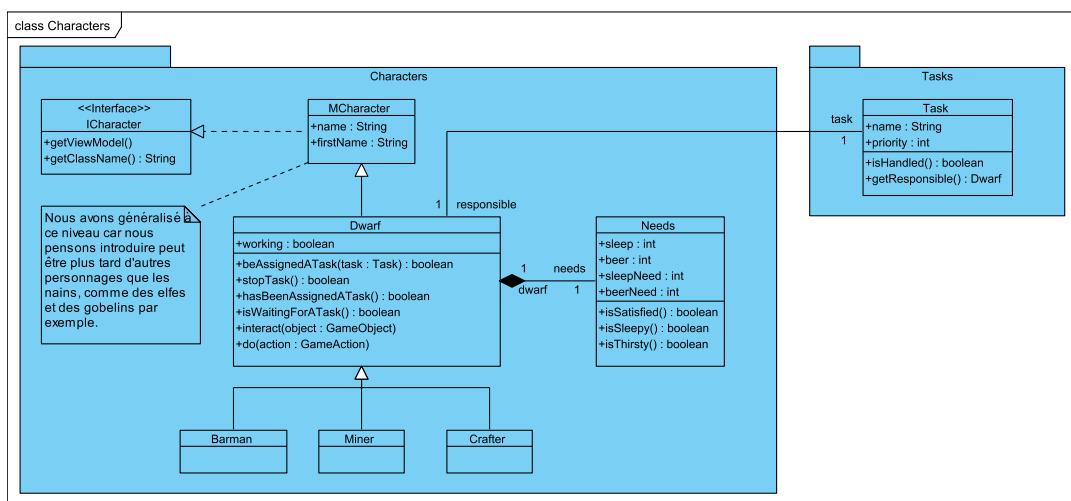


FIGURE 13: Modèle des personnages

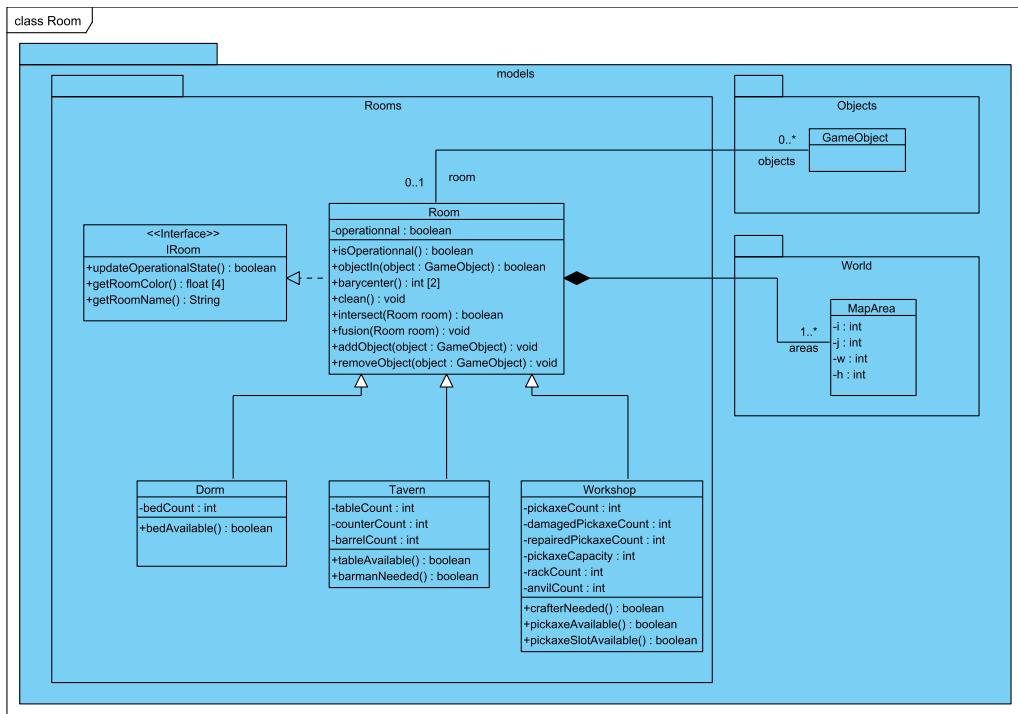


FIGURE 14: Modèle des pièces

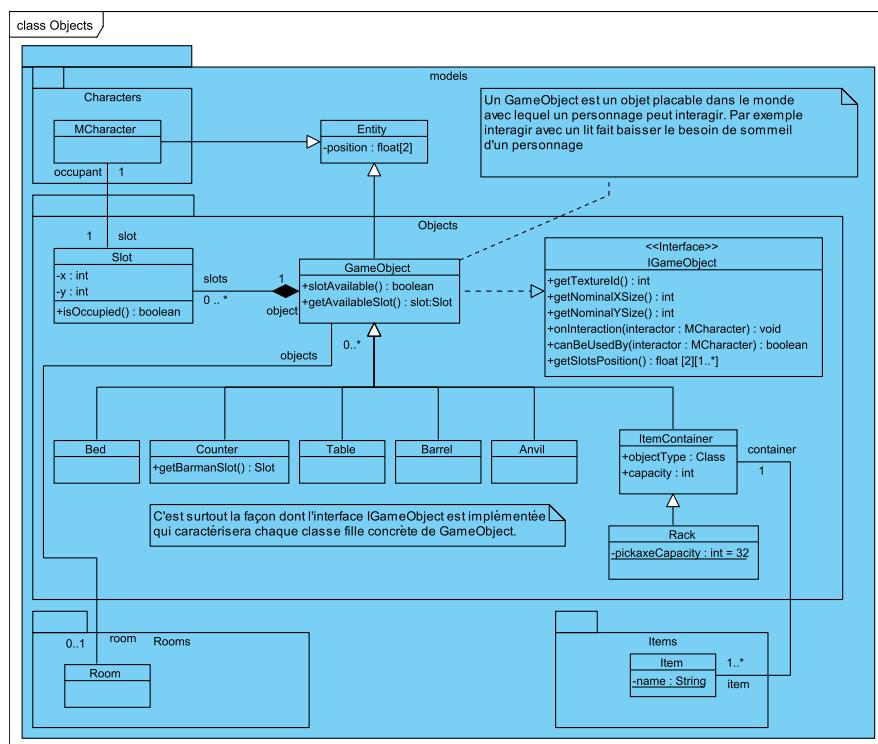


FIGURE 15: Modèle des objets

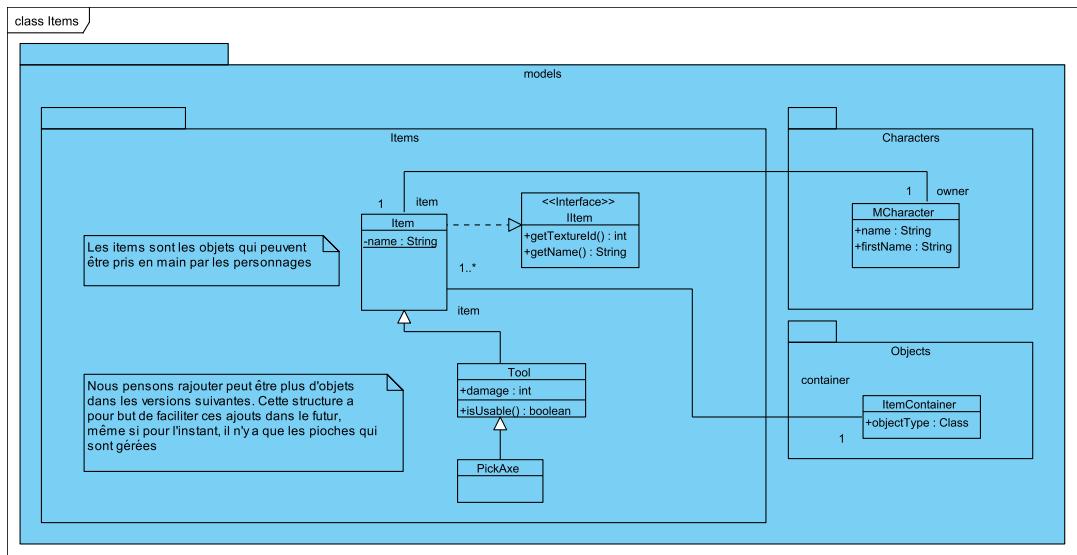


FIGURE 16: Modèle des items

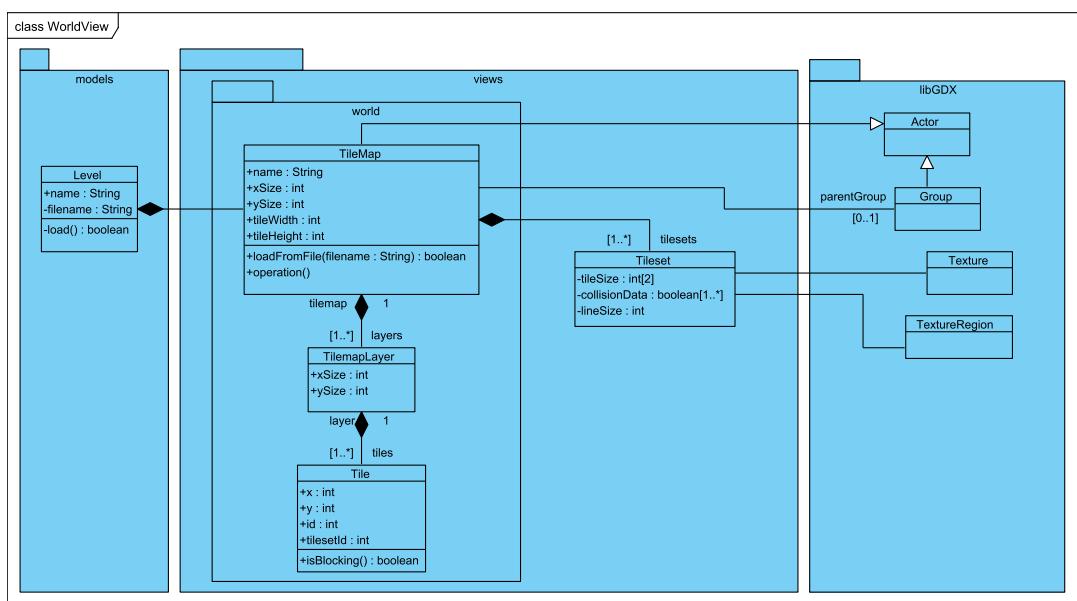


FIGURE 17: Modèle d'affichage du monde

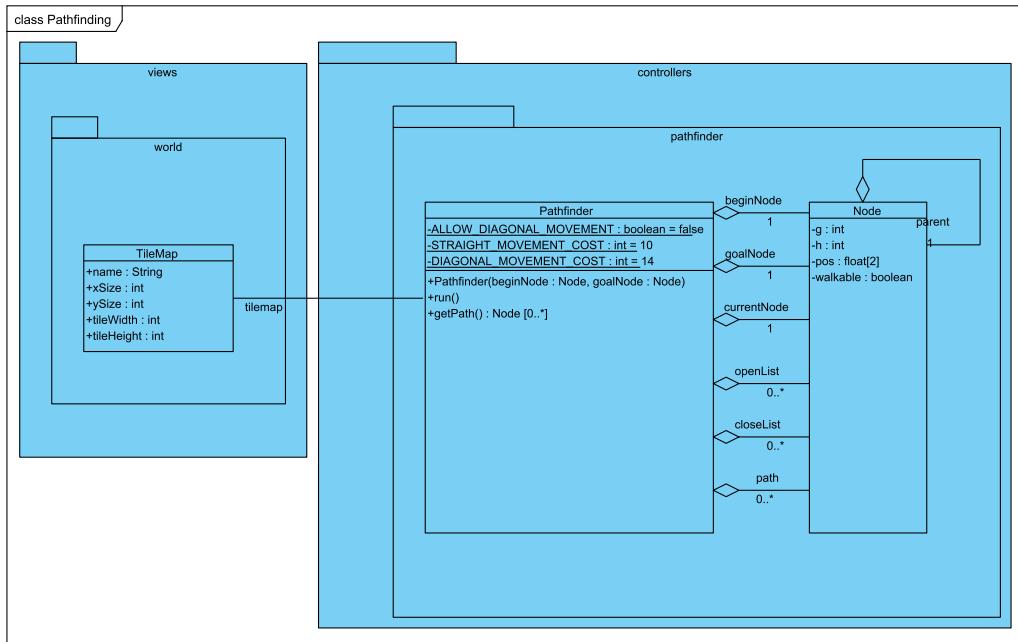


FIGURE 18: Modèle du pathfinder

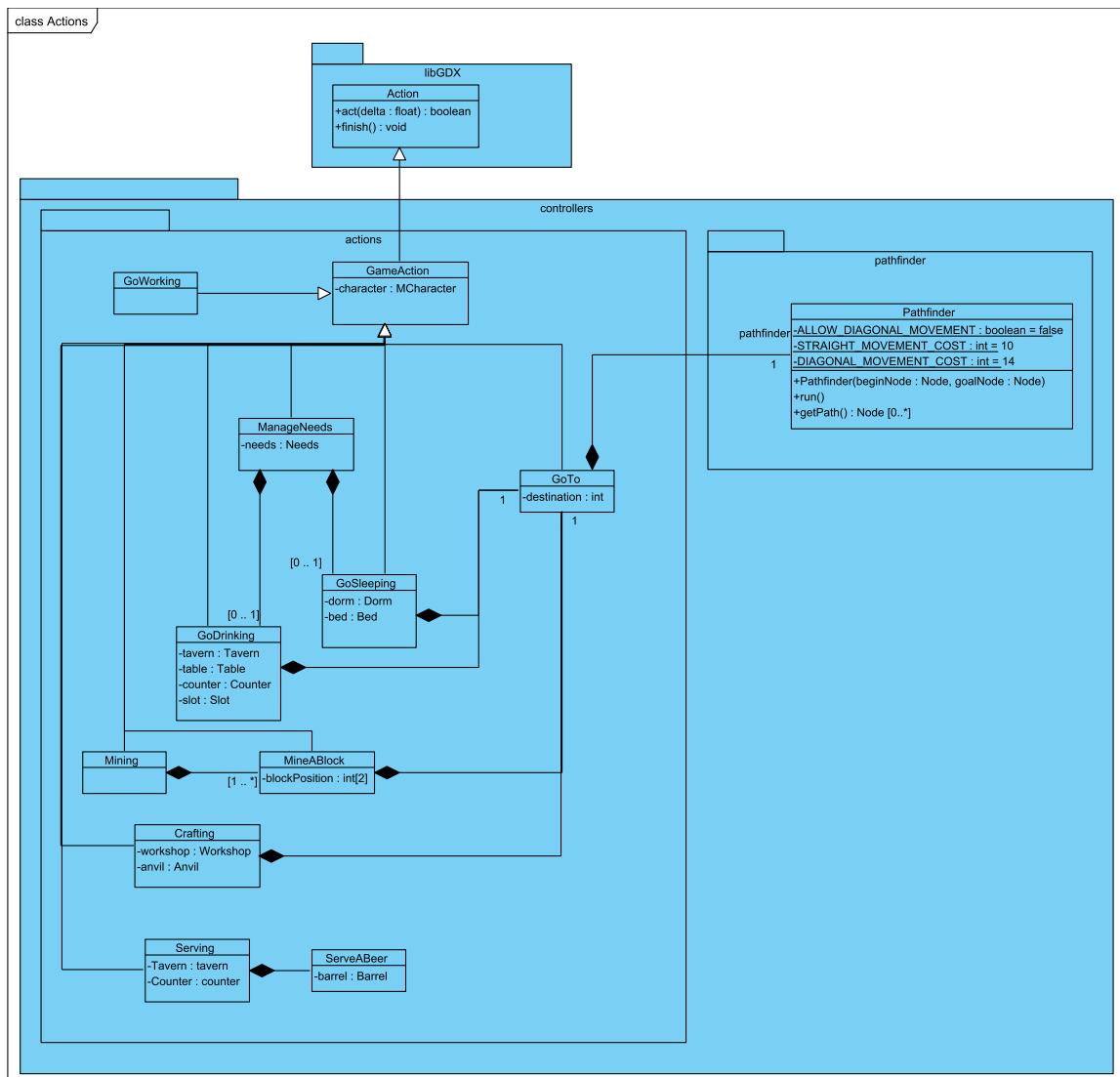


FIGURE 19: Modèle des actions

Remarque sur class Actions (Figure 19)

Ce modèle a été conçu en ayant en tête la façon dont fonctionne les « Actions » de LibGdx. Dans LibGdx, il nous suffira grossièrement de faire :

```
1 model.getView().addAction(new Action(model));
```

Cela aura pour effet d'ajouter une action à réaliser au modèle et à sa vue. Ces actions peuvent être parallélisées ou organisées en séquence de manière très simple grâce à libGdx.

Par exemple dans le code de l'action ManageNeeds on aura :

```
1 public void entry(){
2     this.target.clearActions() // Le nain stoppe tout pour passer a la gestion de ses besoins
3     this.goSleep = new GoSleeping(this.target);
4     this.goDrink = new GoDrinking(this.target);
5     this.target.getView().addAction(this.goSleep)
6     this.target.getView().addAction(this.goDrink) // Ce n'est pas le code final, mais un exemple bien
7 }
```

De ce principe découle tout le système de composition mis en place dans ce diagramme.

NB : Nous pensons que donner une idée de ce à quoi ressemblera le code permet de mieux comprendre ce diagramme.

Diagrammes d'activités

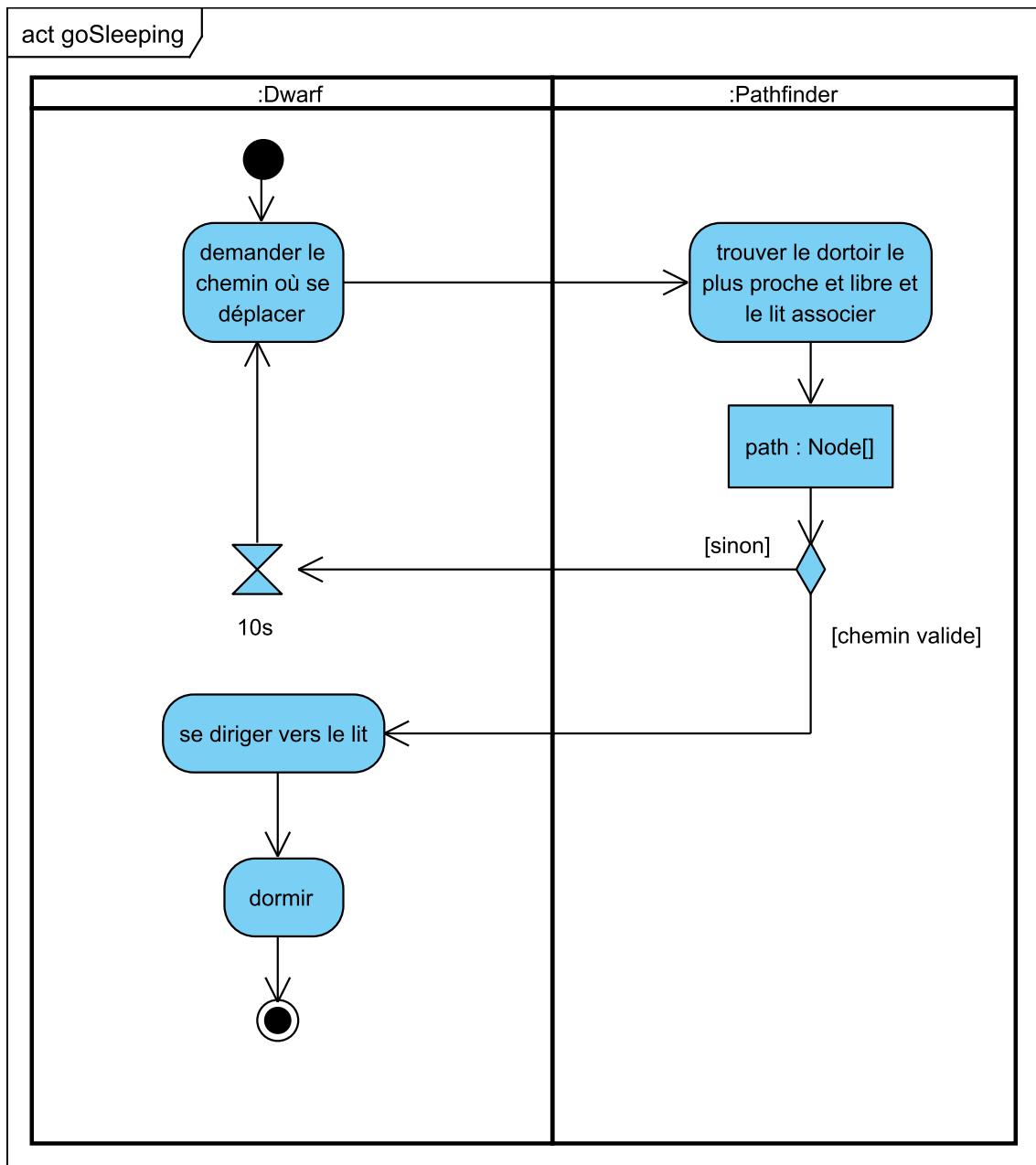


FIGURE 20: Activité aller dormir

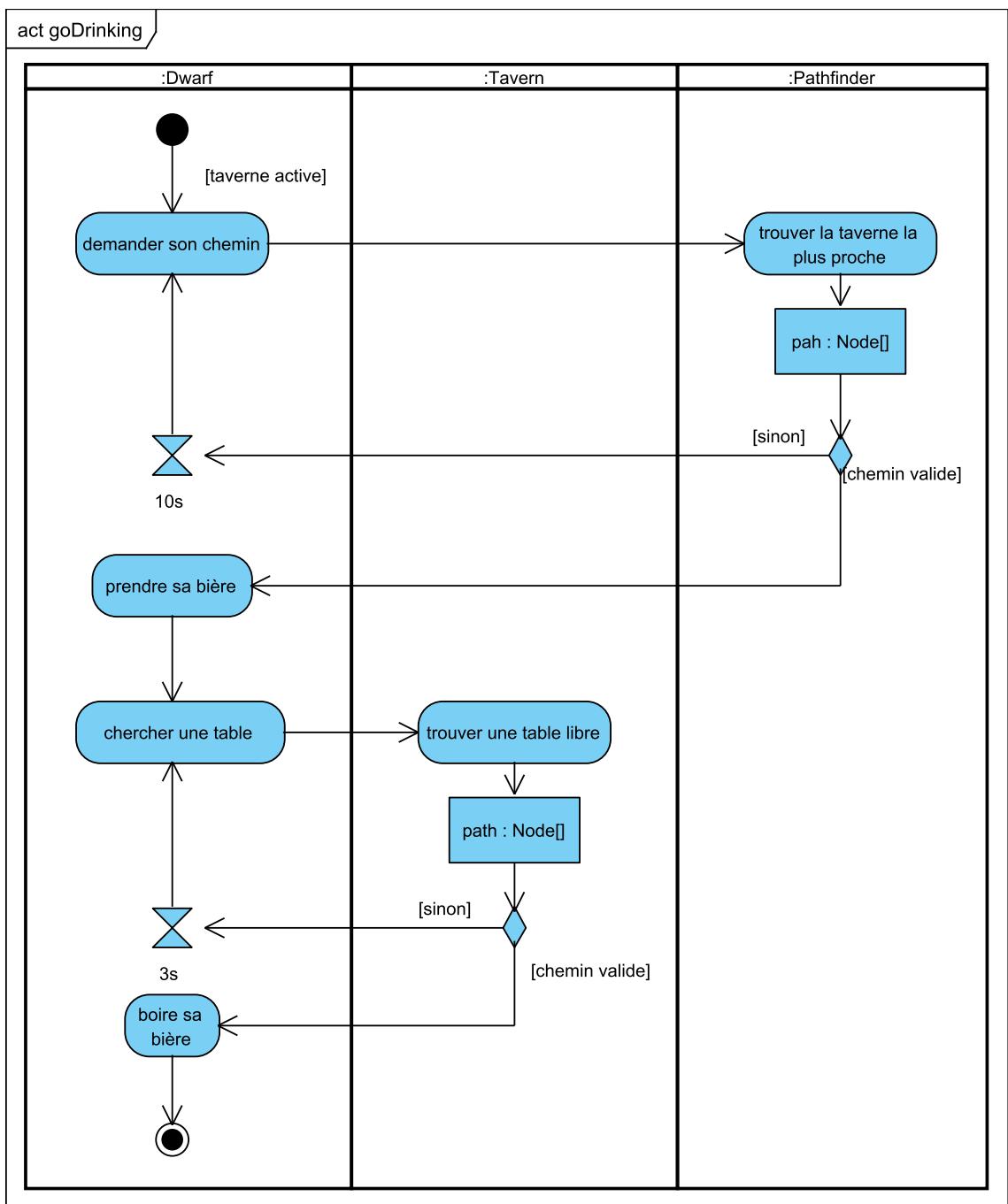


FIGURE 21: Activité aller boire

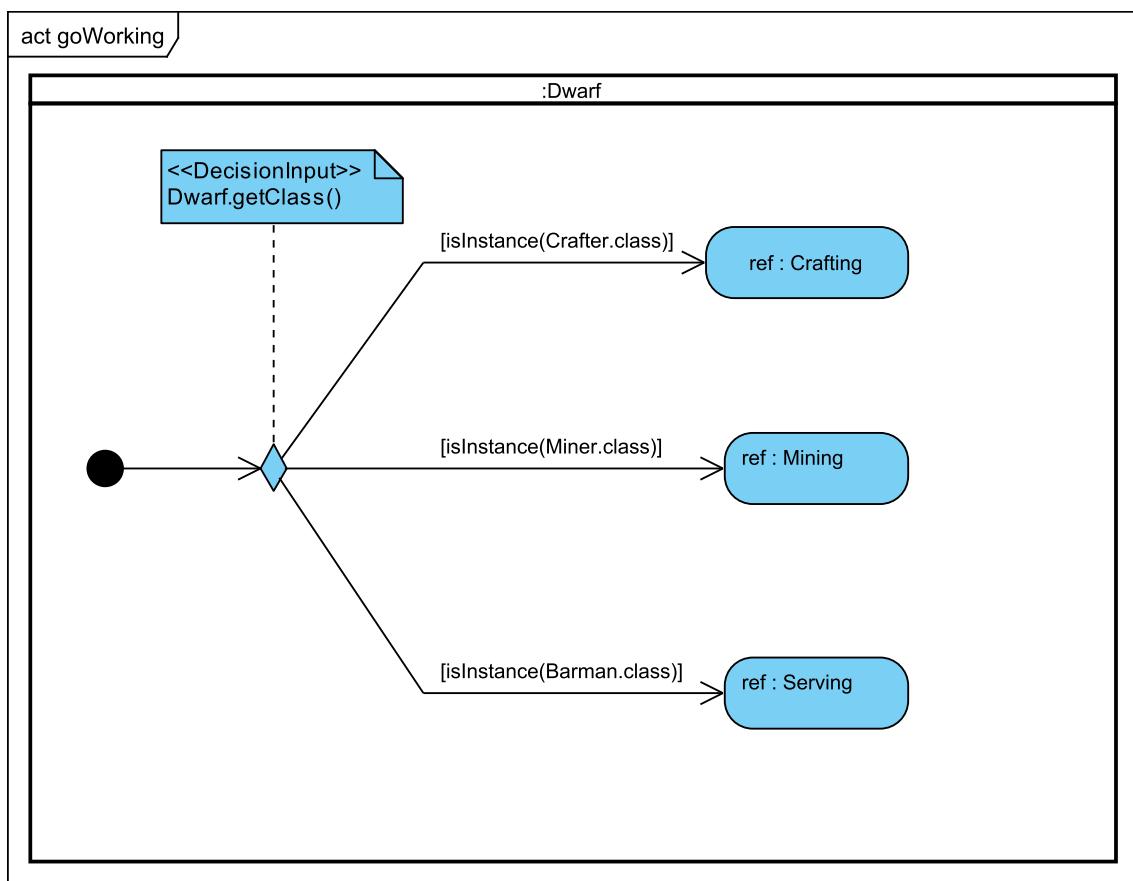


FIGURE 22: Activité aller travailler

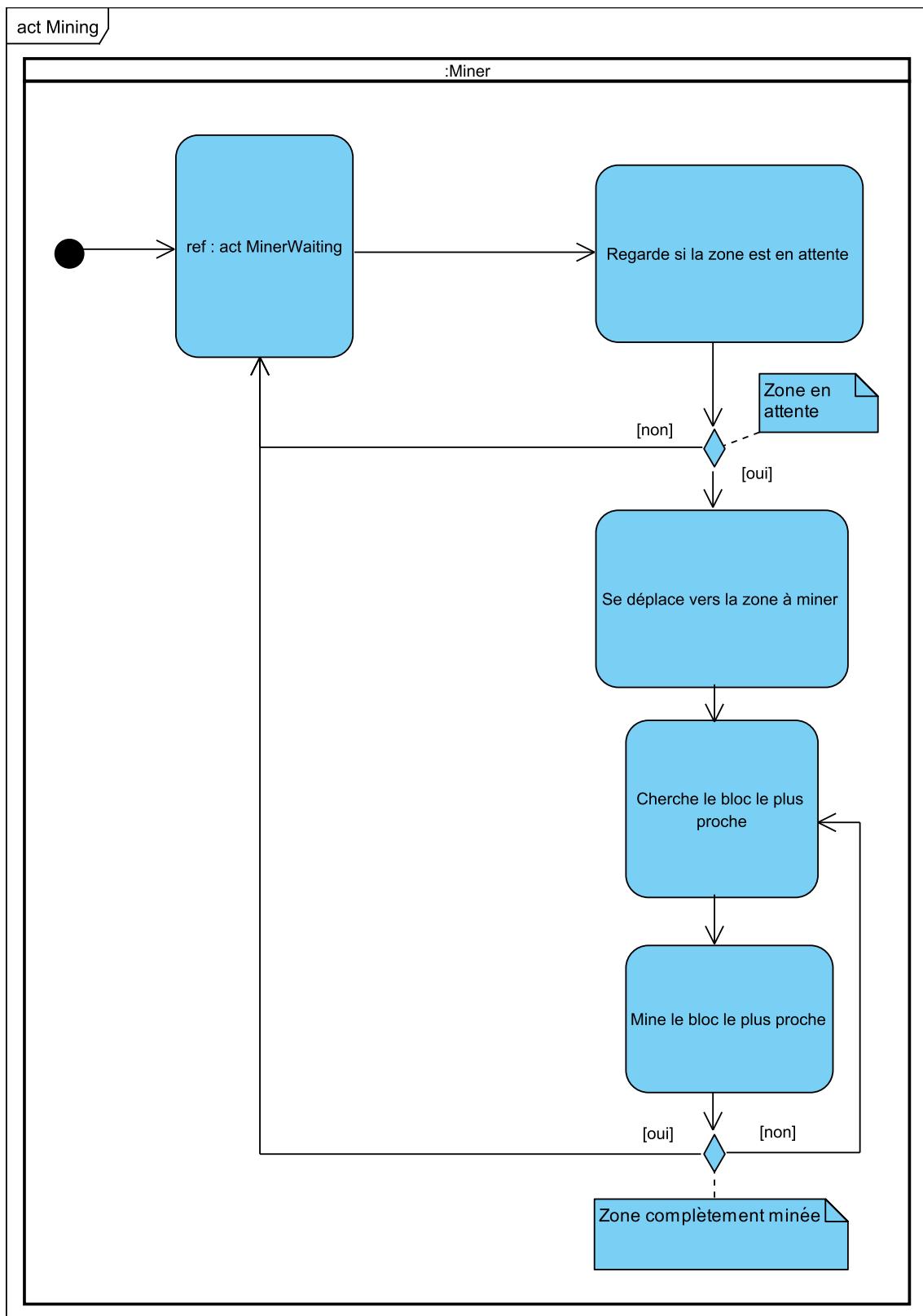


FIGURE 23: Activité aller miner

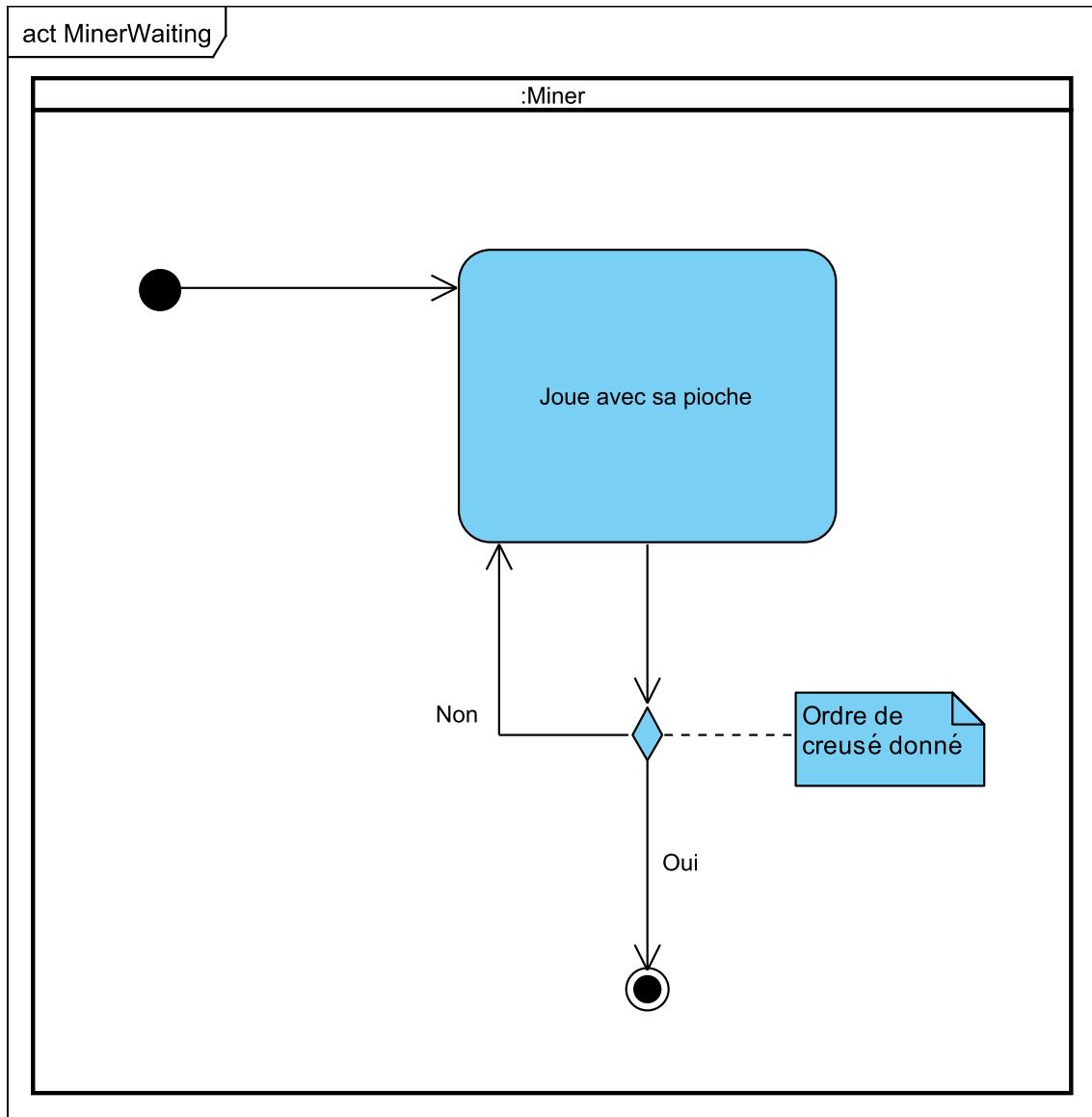


FIGURE 24: Activité d'attente du mineur

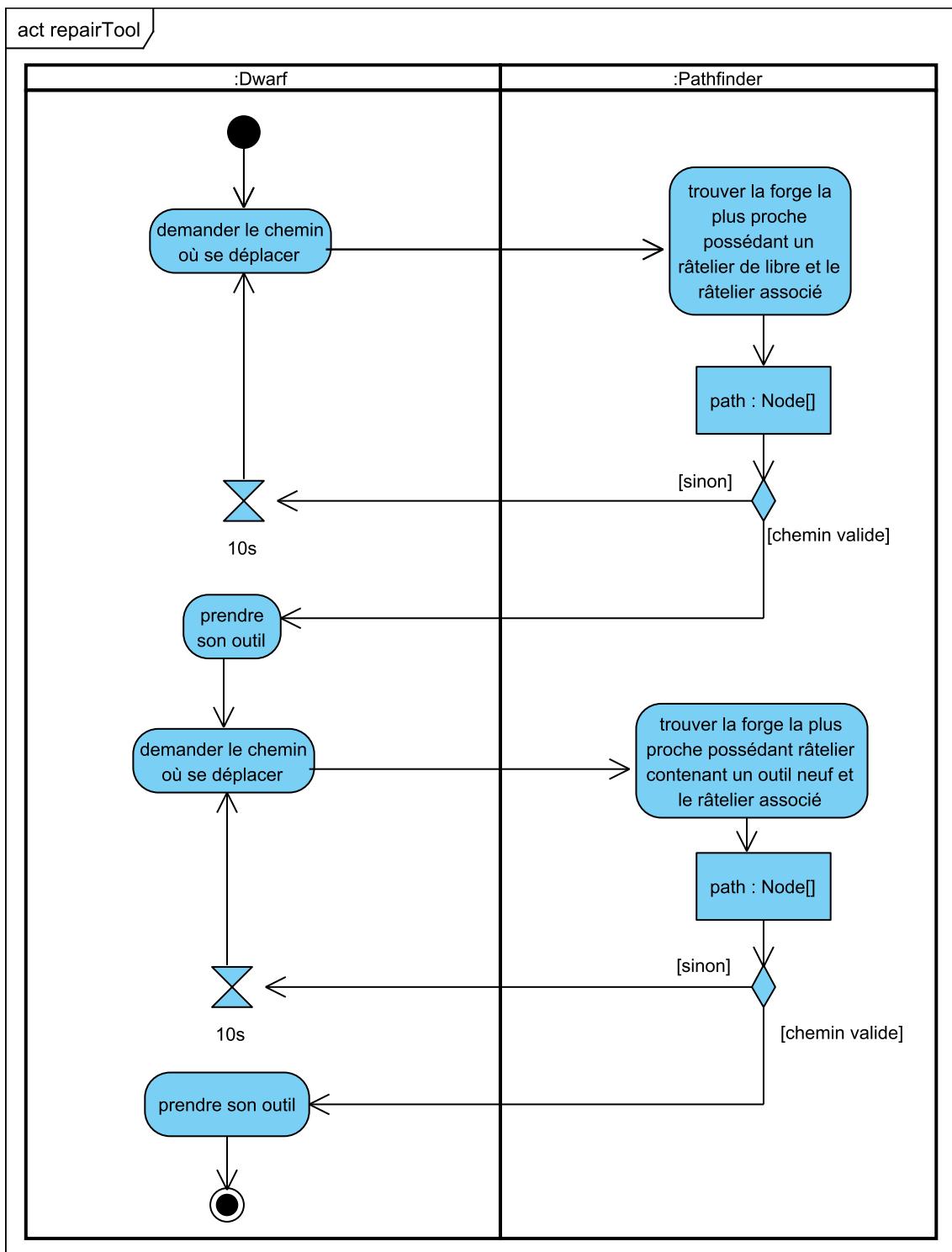


FIGURE 25: Activité aller réparer son outil

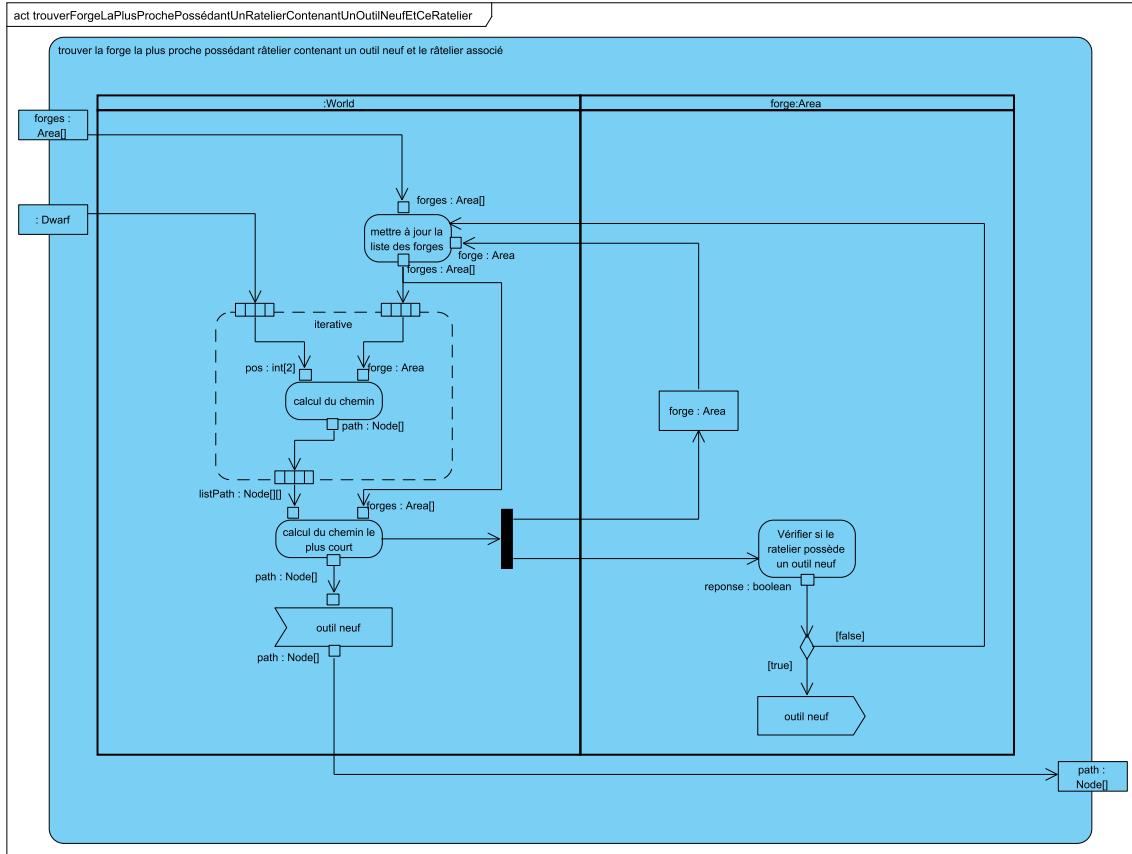


FIGURE 26: Activité de recherche de l'atelier le plus proche possédant au moins un outil neuf

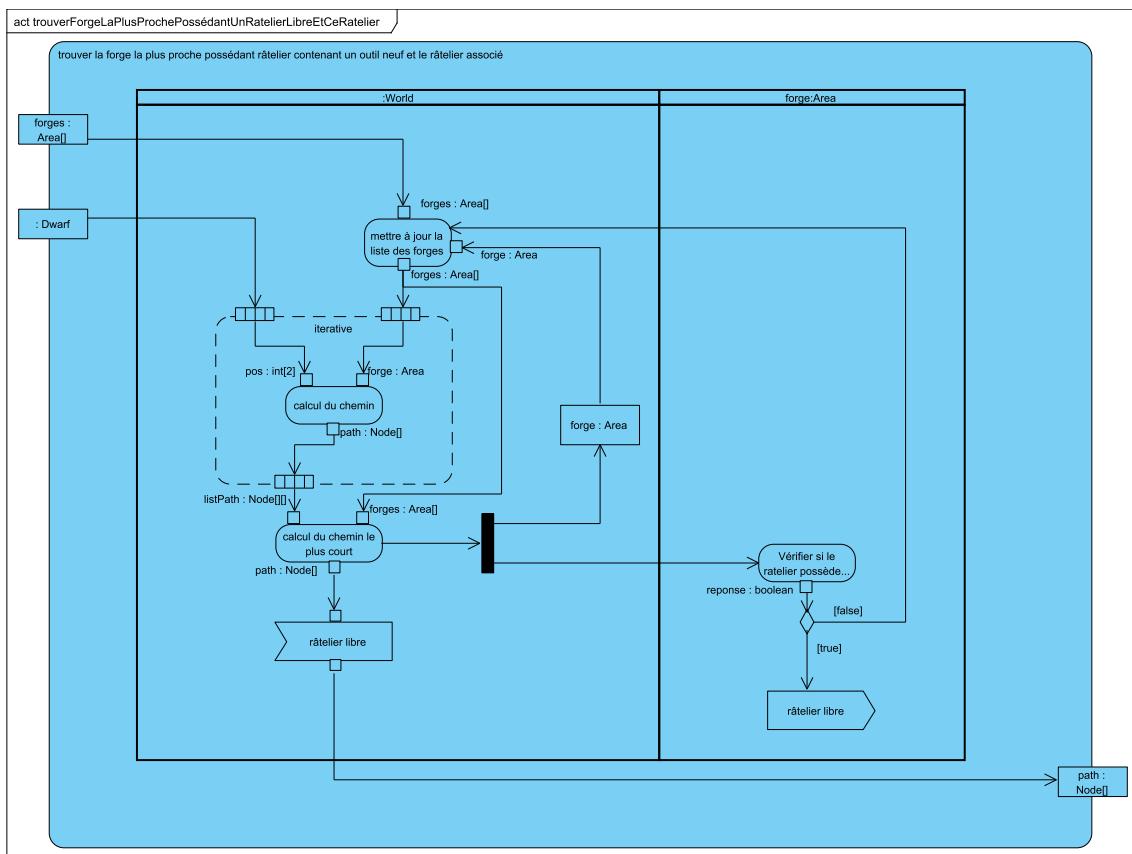


FIGURE 27: Activité de recherche de l'atelier le plus proche possédant au moins un ratelier libre

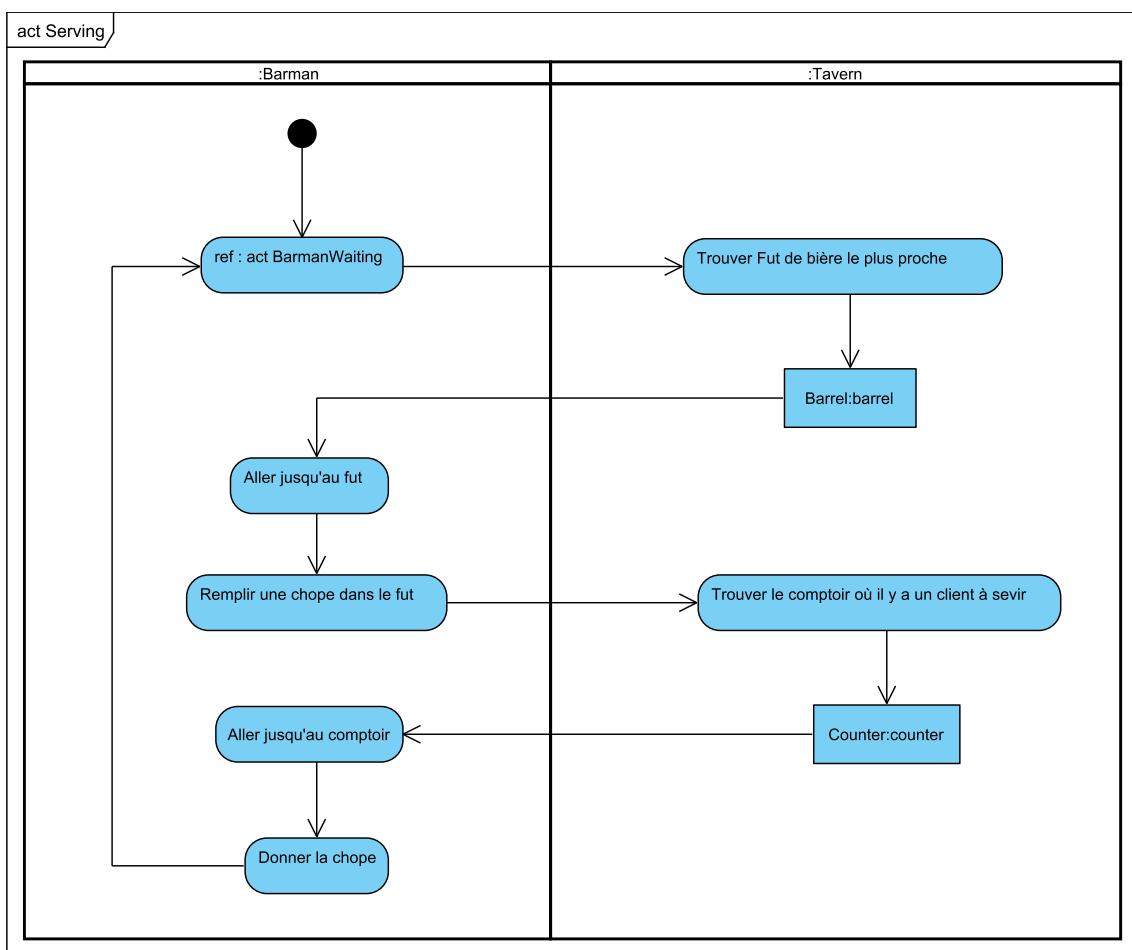


FIGURE 28: Activité de travail du barman

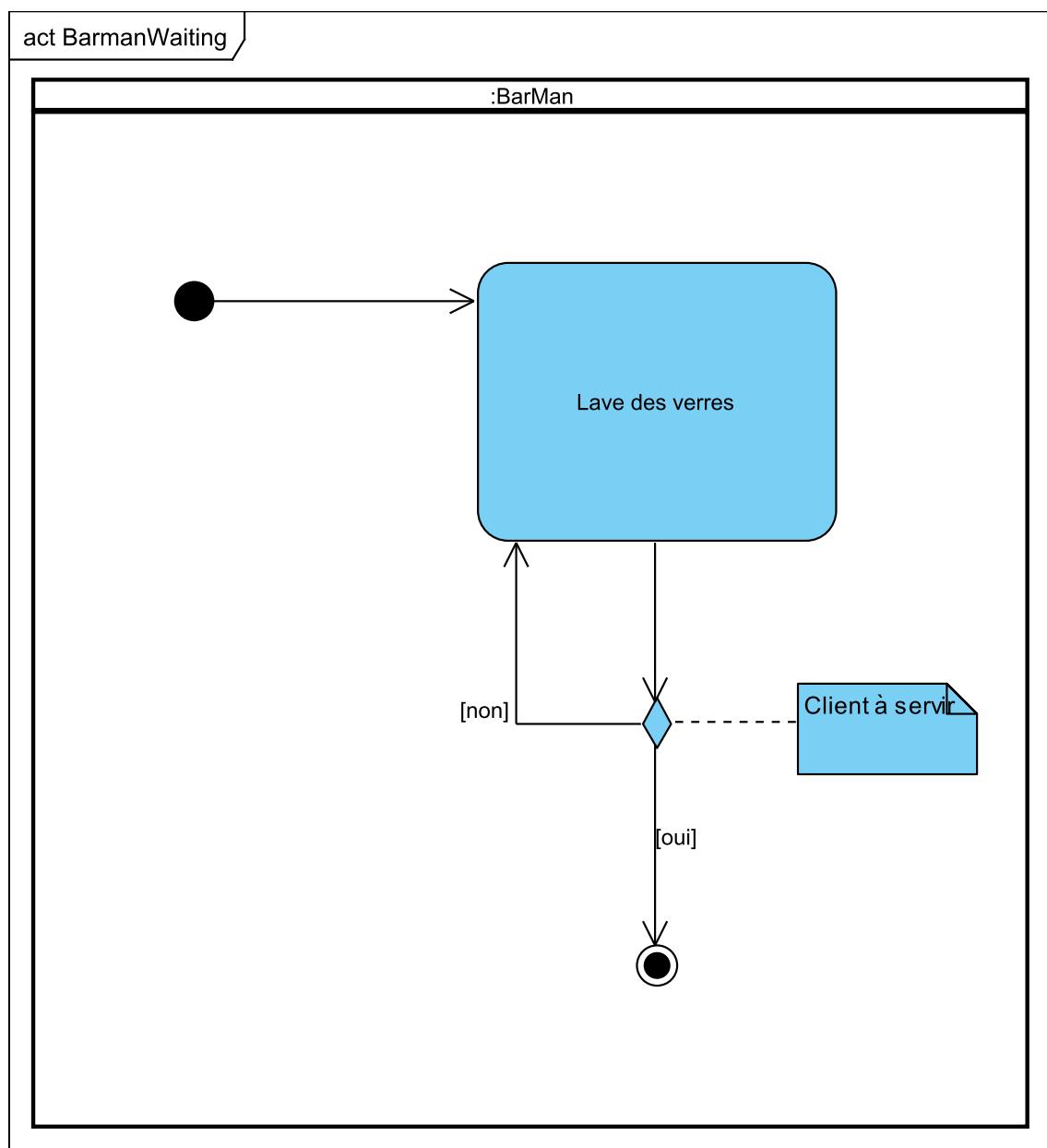


FIGURE 29: Activité d'attente du barman

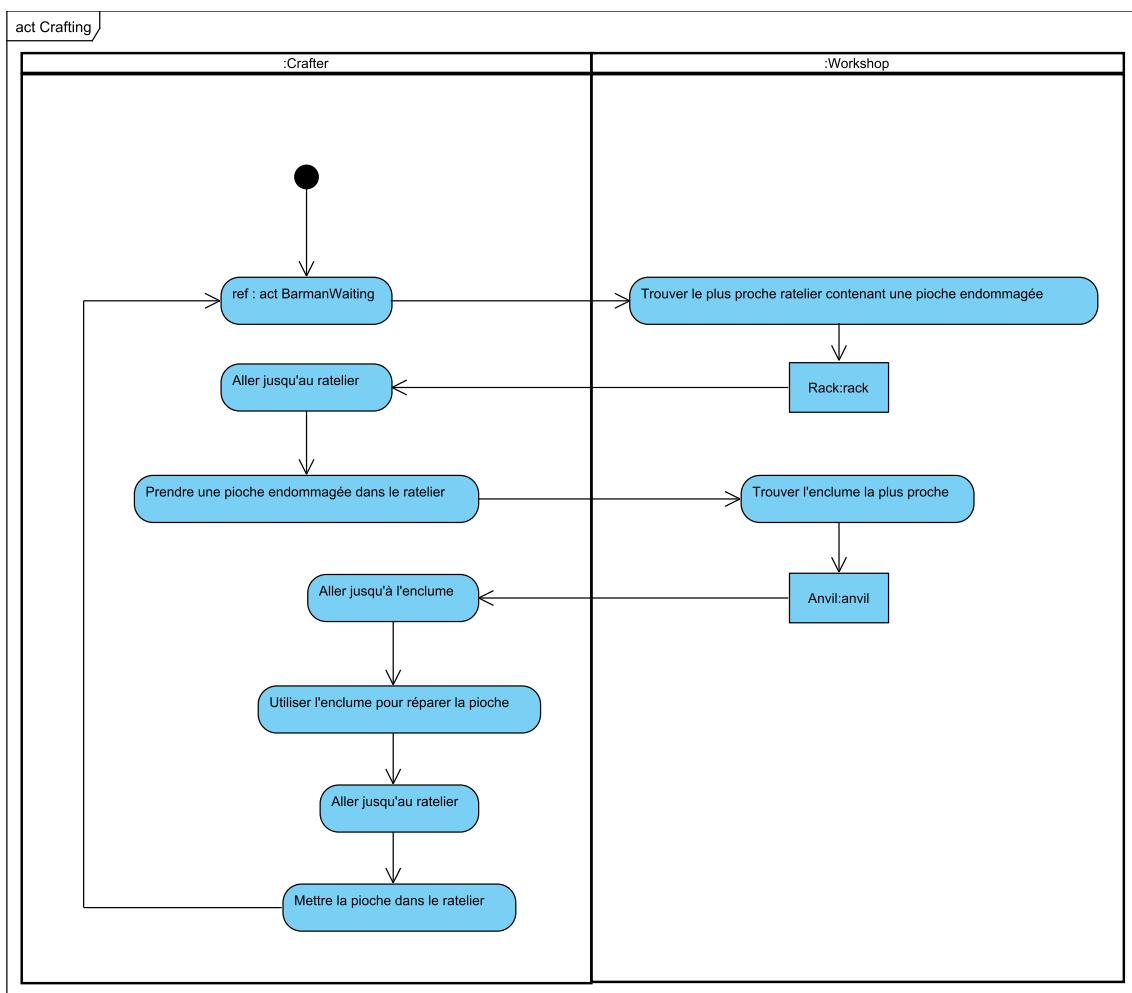


FIGURE 30: Activité de travail de l'artisan

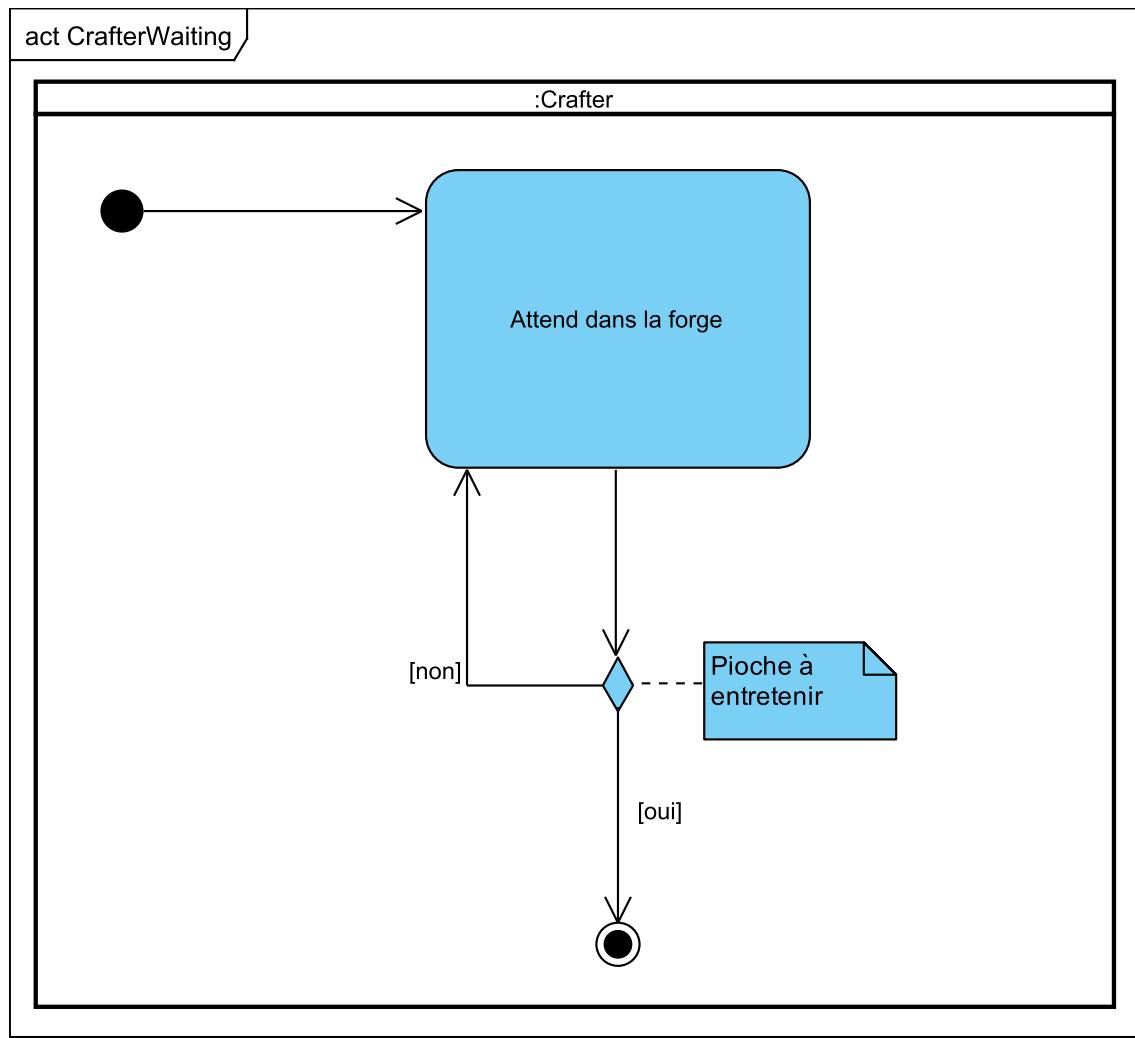


FIGURE 31: Activité d'attente de l'artisan

Machines à états

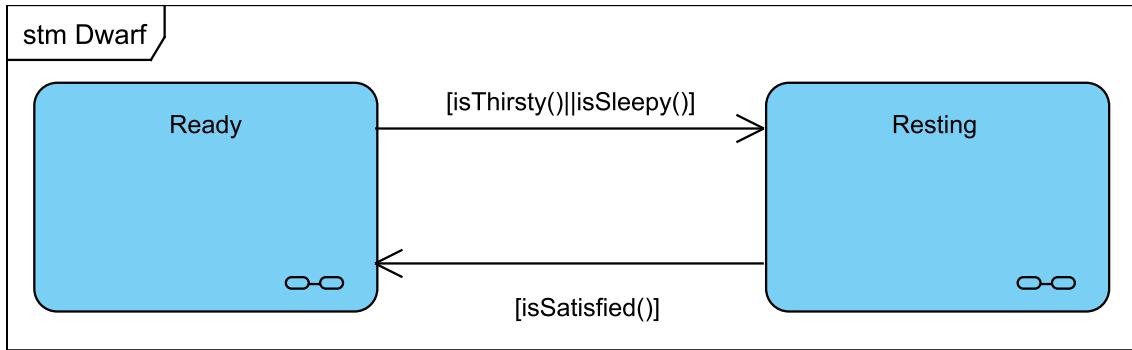


FIGURE 32: Machine à états du nains

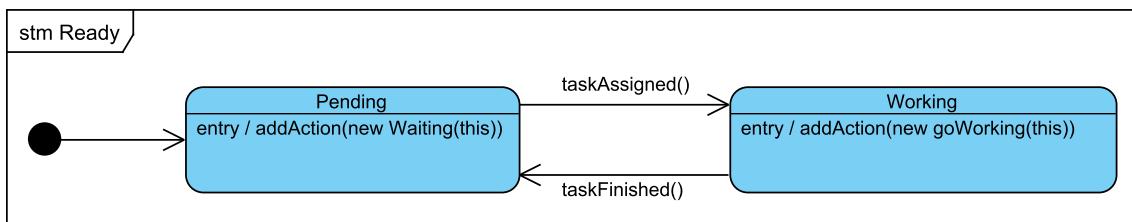


FIGURE 33: Détail de la sous-machine Ready du diagramme Dwarf

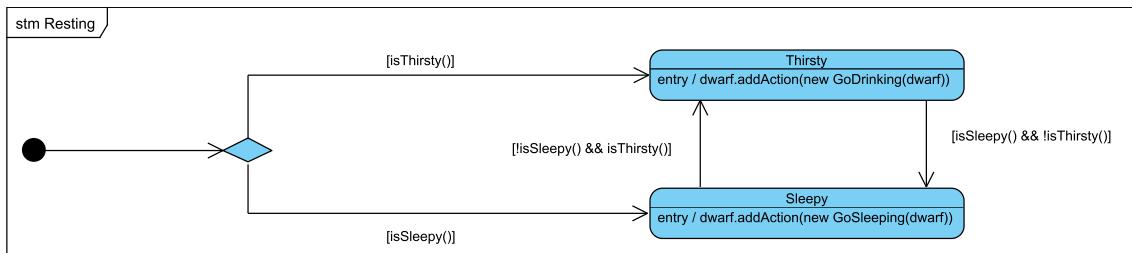


FIGURE 34: Détail de la sous-machine Resting du diagramme Dwarf

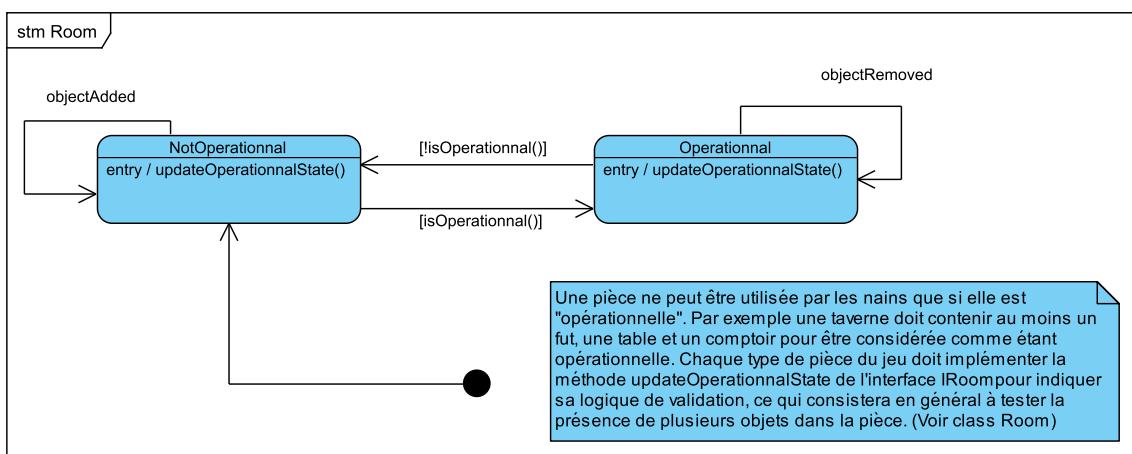


FIGURE 35: Machine à état des pièces



Merci d'avoir lu