

RAPPORT DE PROJET

Groupe H

Leglise Sébastien

Mansouri Nihed

Pessel Malo

Cours de Programmation Concurrentes et Interfaces Interactives

Année universitaire 2024-2025

Sommaire

I. Introduction	4
II. Analyse globale	5 à 14
A. Fonctionnalités principales	5 à 12
B. Analysé détaillée	12 à 14
III. Plan de développement	15 à 18
1. Phase de recherche et documentation	15
2. Conception de la vue	15 à 16
3. Conception Contrôleur	17
4. Conception Modèle	17 à 18
5. Conception générale	18
IV. Conception générale	19 à 20
A. Conception générale	19
1. Modèle (Model)	19
2. Vue (View)	19
3. Contrôleur (Controller)	20
V. Conception détaillée	21 à 59
● Déplacement des unités	21 à 23
● Gestion des collisions et interactions	24 à 27
● Gestion GameMaster.....	28 à 31
● Gestion des sélections	32 à 34
● Gestion des zones de fonctionnement	35 à 36
● Gestion des Spawns ennemis.....	37 à 39
● Gestion des unités non contrôlables	40 à 43
● Gestion des unités contrôlables	44 à 47
● Gestion Affichage des Images	48 à 49
● Sac et inventaire	50 à 51
● Sous Marin.....	51 à 52

• Système de victoire et paramétrage	52 à 53
• Génération et évolution des ressources	54 à 56
• Carte des profondeurs et gestion du terrain	56 à 57
• Barres de progression personnalisés	58 à 59
VI. Résultat	60 à 63
VII. Documentation utilisateur	64 à 66
VIII.Documentation développeur	67 à 68
IX. Conclusion et perspectives	69 à 70

I- Introduction :

Dans un futur post-apocalyptique où les océans ont englouti une grande partie de la surface terrestre, les vestiges d'une civilisation oubliée ont refait surface. Ces ruines sous-marines, renfermant des artefacts puissants et mystérieux, convoités tant pour leur valeur scientifique que commerciale. C'est dans ce monde que s'inscrit notre projet, qui plonge le joueur au cœur d'une expédition ambitieuse mêlant exploration, stratégie et gestion des ressources.

Le joueur incarne le directeur d'une expédition sous-marine, à la tête d'un groupe de plongeurs qu'il doit recruter, et diriger pour récolter les artefacts disséminés dans les profondeurs. En début de partie, le joueur définit une durée limitée ainsi qu'une somme d'argent initiale, son objectif étant alors d'atteindre un certain nombre de points de victoire avant la fin du temps imparti. Il faudra donc optimiser les déplacements des plongeurs, gérer les ressources récupérées, et faire face à un environnement dangereux et à des créatures hostiles.

Ce projet mêle plusieurs mécaniques de jeu : la collecte et la gestion d'objets (artefacts), le recrutement d'unités (plongeurs), la navigation dans des environnements à profondeur variable, ainsi que la confrontation à des ennemis intelligents et perturbateurs (comme les pieuvres ou les calamars). Une interface permet au joueur d'interagir avec ses unités, d'accéder à ses ressources, et de prendre des décisions stratégiques en temps réel.

Le jeu a été conçu avec l'objectif de proposer une expérience dynamique, entre la progression rapide et les dangers croissants des profondeurs. De futures évolutions du projet pourraient ajouter de nouvelles unités, des événements aléatoires, ou des mécanismes de coopération et de compétition.

II- Analyse globale:

A. Fonctionnalités principales :

- Interface principale et gestion des sessions de jeu :

Interface graphique complète permettant de démarrer une partie et de configurer précisément les paramètres de jeu (temps limite, argent initial, points de victoire nécessaires).

- Affichage du jeu et interface utilisateur :

Interface graphique basée sur un JPanel divisé en deux sections clairement délimitées :

- **Zone de jeu** : représente visuellement la carte sous-marine où évoluent les unités contrôlables et non contrôlables, avec une caméra dynamique permettant de naviguer sur un grand terrain.
- **Panneau de contrôle** : affiche en temps réel les informations essentielles telles que les unités sélectionnées, l'état des ressources, l'inventaire actuel, les performances du joueur (score, argent disponible, carburant, oxygène restant), et permet d'effectuer diverses actions.

- Gestion des artefacts et stockage :

Permet au joueur de manipuler les artefacts découverts pendant l'exploration. Chaque artefact récupéré peut être placé dans l'inventaire des plongeurs et/ou vendu contre de l'argent et des points de victoire. Un inventaire centralisé affiche en temps réel tous les objets stockés, facilitant leur gestion et vente.

- Administration des unités et équipements :

Propose une interface détaillée pour gérer les plongeurs et sous-marins disponibles. Le joueur peut recruter différents types de plongeurs (normaux ou armés), acheter des sous-marins, et équiper les unités avec des ressources spécifiques telles que l'oxygène ou l'essence. Chaque recrutement ou amélioration impacte les performances globales, les possibilités d'exploration, ainsi que le score.

- **Inventaire et collecte des artefacts :**

Un mécanisme dynamique permet aux plongeurs de collecter les artefacts trouvés en les plaçant directement dans leur sac à dos (backpack). Une fois leur capacité maximale atteinte, ou sur décision stratégique du joueur, les plongeurs retournent à la base pour livrer les artefacts. Ceux-ci sont alors transférés de l'inventaire des plongeurs vers l'inventaire central de la base et deviennent disponibles sur le marché (market), où ils pourront être vendus pour générer des revenus financiers ainsi que des points de victoire supplémentaires. L'interface utilisateur reflète en temps réel ces étapes de collecte, livraison et mise à disposition au market, offrant au joueur une gestion fluide et interactive de ses ressources.

- **Système de score et conditions de victoire/défaite:**

Le système de score est géré par une classe dédiée appelée **Referee**, responsable du suivi détaillé des performances du joueur. Les points sont attribués selon les règles suivantes :

- **Vente des artefacts :**
 - Collier : **+10 points**.
 - Bague : **+20 points**.
 - Relique : **+50 points**.
- **Embauche d'unités :**
 - Plongeur normal : **+10 points**.
 - Plongeur armé : **+20 points**.
- **Achat de sous marin : + 30 points**

Chaque fois qu'une action éligible est réalisée (vente ou embauche), les points correspondants sont immédiatement ajoutés au compteur général. Ce compteur est affiché en permanence et mis à jour en temps réel. La classe Referee utilise également un mécanisme de chronomètre précis basé sur System.currentTimeMillis() afin de mesurer le temps écoulé. Une partie est remportée dès que le joueur atteint l'objectif prédéfini en début de partie, avant que le temps limite fixé ne soit dépassé. À l'inverse, si le temps imparti s'écoule sans avoir atteint l'objectif, la partie est perdue.

- **Gestion du Sous-marin :**

Un système dédié permettant au joueur de gérer le fonctionnement spécifique des sous-marins, incluant :

- Embarquement/débarquement des plongeurs :
Les plongeurs peuvent embarquer dans le sous-marin lorsque celui-ci est à proximité, devenant invisibles sur la carte, et débarquer automatiquement quand le carburant est épuisé.
- Gestion du carburant (essence) :
Le sous-marin consomme du carburant lorsqu'il est en mouvement. Le carburant doit être surveillé en permanence, nécessitant des ravitaillements via le marché.

- **Gestion du Backpack des plongeurs :**

Chaque plongeur possède un inventaire limité (backpack) pour stocker les artefacts collectés. Lorsque le backpack atteint sa capacité maximale, ou à tout moment selon la décision stratégique du joueur, le plongeur retourne à la base pour livrer les artefacts collectés. Ceux-ci sont alors transférés vers le marché, où ils deviennent disponibles à la vente.

- **Embauche des plongeurs :**

Le joueur peut embaucher de nouvelles unités (plongeurs normaux ou armés) depuis l'interface de gestion des unités. Chaque nouveau plongeur recruté est ajouté automatiquement à la carte du jeu, à une position aléatoire située exclusivement dans la zone correspondant à la **profondeur 1**. Ce mécanisme d'apparition stratégique limite initialement l'accès aux zones profondes et pousse le joueur à optimiser les déplacements et les stratégies d'exploration au fil de la partie.

- **Économie et système de boutique (Market) :**

Permet l'achat d'unités supplémentaires (sous-marins, plongeurs normaux ou armés), ainsi que de ressources essentielles (oxygène, essence). Chaque achat déduit automatiquement l'argent disponible, influence le score et améliore les capacités opérationnelles du joueur sur le terrain.

- **Recharge des unités contrôlables :**

Un mécanisme simple et efficace permettant au joueur d'acheter des recharges (via la boutique) pour remettre au maximum des ressources critiques :

- Essence : Recharge immédiatement à 100 % le carburant de tous les sous-marins.
- Oxygène : Recharge immédiatement à 100 % l'oxygène de tous les plongeurs.

- **Minimap et navigation rapide :**

La MiniMap fournit une représentation miniature et dynamique du terrain sous-marin. Elle permet au joueur d'avoir une vue d'ensemble en temps réel de l'environnement, même lorsque la carte principale est agrandie.

Fonctionnalités clés :

- Affiche la position de toutes les unités contrôlables, des ressources visibles, de la base et des ennemis détectés.
- Les éléments sont dessinés à l'échelle, avec des couleurs différentes pour chaque type :
 - Plongeur : carré bleu (rouge s'il est sélectionné).
 - Ressources : cercle jaune ou vert selon l'état, uniquement si détecté.
 - Ennemi : cercle rouge, uniquement si détecté.
 - Base : rectangle cyan.
- Un rectangle blanc transparent indique la zone visible par la caméra actuelle.
- Navigation rapide possible via clic sur la minimap pour déplacer la caméra.
- **Terrain et profondeur :**

Le terrain est représenté sous forme d'une grille 2D très étendue (par exemple 12000 x 12000 pixels) et intègre des zones de profondeur simulant les fonds marins.

Caractéristiques :

- Carte des profondeurs (depthMap) générée procéduralement avec bruit aléatoire, pour créer un effet naturel.
- Le terrain est divisé en 4 zones de profondeur :
 - Profondeur 1 : Peu profond (ressources fréquentes comme colliers).
 - Profondeur 2 : Moyenne profondeur (bagues, colliers).
 - Profondeur 3 : Profonde (bagues, reliques rares).
 - Profondeur 4 : Très profonde (reliques spéciales, trésors).
- Chaque zone a une limite de ressources simultanées, pour inciter à la stratégie.
- Les textures de fond varient selon la profondeur, renforçant l'immersion visuelle.

- **Caméra :**

Le système de caméra dynamique permet au joueur d'explorer un vaste terrain avec fluidité.

Fonctionnalités :

- La caméra se déplace en suivant les actions du joueur (clics, ou raccourcis (clic molette)).
- Sa position est synchronisée avec la MiniMap pour indiquer la zone actuellement visible.
- Le joueur peut naviguer horizontalement et verticalement sur l'ensemble de la carte.

- **Déplacement des différentes unités :**

Toutes les unités se déplacent toutes en même temps et pour permettre à nos unités de se déplacer, nous leur donnons un **vecteur vitesse** pour simplifier la gestion des déplacements (tout mouvement peut être décrit comme une combinaison de deux mouvements élémentaires vx,vy), et donc chaque unité garde sa position courante et ce vecteur. Lorsqu'un unité **accélère** sa vitesse courante incrémenté selon un taux d'accélération défini, évitant ainsi un démarrage brusque qui romprait l'illusion de mouvement naturel, et ensuite à l'approche de la destination la vitesse perd graduellement son intensité.

- **Gestion des unités contrôlables :**

Sur le terrain il existe une base principale à laquelle on doit ramener les ressources, il s'agit aussi de l'endroit où il va remplir son oxygène et munition s'il en a.

Le joueur peut effectuer plusieurs actions en fonction de l'unité sélectionnée:

- **Déplacement** : choix précis des destinations.
- **Récupération des ressources** : les plongeurs peuvent récolter automatiquement des artefacts une fois à proximité, remplies progressivement leur Sac à dos.
- **Interaction avec les unités ennemis** : le joueur peut choisir d'effrayer certains adversaires pour les faire fuir ou d'engager le combat en utilisant des unités armées, que ce soit à distance ou en mêlée.

On peut sélectionner jusqu'à **10 unités** en même temps, les actions en communs sont donc affichées permettant ainsi de donner des ordres à nos unités plus facilement.

La plupart de nos unités ont une quantité d'**oxygène** limitée et doivent donc la recharger constamment, si une unité se retrouve sans oxygène elle est retirée du jeu. L'oxygène diminue constamment et la quantité d'oxygène perdue augmente lorsqu'on vas plus profond ou on fait fuire des ennemis.

En plus les unités possèdent ce qu'on appelle "**stamina**" qui décroît lorsqu'on se déplace ou lorsqu'on fait fuire des ennemis. Une stamina trop réduite rend les unités plus lentes ou même incapables de se déplacer obligeant nos unités à prendre des pauses ou tout simplement d'arrêter l'action en cours.

Toutes les unités dans le jeu, ont des heat points indiquant leurs points de vie, si il arrive à 0 il sont automatiquement retirés du jeu.

- **Gestion des collisions et interactions :**

Parallèlement au reste des fonctionnalités du jeu, un **gestionnaire de proximité** détecte en permanence quels objets sont à proximité de nos unités pour pouvoir gérer les interactions et les collisions. Pour cela on utilise une **grille** qui existe en parallèle avec le terrain de jeu.

Ce gestionnaire travaille en parallèle avec un **deuxième gestionnaire** qui met à jour les coordonnées des objets par rapport à cette grille.

Le principe est simple, pour chacune de nos unités on récupère ses coordonnées dans cette grille et on récupère les objets voisins. De cette façon, le gestionnaire gère les collisions entre notre unité et ses voisins et évite que nos unités chevauchent avec d'autres objets. En plus, on gère les différentes fonctionnalités demandant que deux ou plus objets interagissent avec nos unités.

- **Gestion des zones de fonctionnement :**

Les **zones de fonctionnement** sont des régions définies sur la carte qui permettent de limiter les calculs et les interactions à des zones spécifiques. Elles servent à **optimiser les performances** du jeu en ne traitant que les objets que le joueur peut percevoir.

On a donc une zone principale qui se trouve dans la zone visible du jeu et plusieurs zones qui sont créées lorsque les unités contrôlables ne sont plus visibles.

- **Gestion des unités non contrôlables :**

Le **GameMaster** est l'un des gestionnaires central du jeu. Il coordonne les interactions entre les ennemis et les zones de fonctionnement. Il s'agit d'un chef d'orchestre qui s'assure que tous les éléments du jeu fonctionnent correctement et de manière optimisée.

Il maintient la liste de toutes les unités non contrôlables et gère leur bon fonctionnement, et exécute les actions des ennemis en parallèle pour cela on crée une **deuxième grille** en parallèle du terrain et de la grille du gestionnaire de proximité.

Les unités non contrôlables sont toutes des **ennemis** pour le joueur, elles interagissent avec les ressources et les unités contrôlables. Chaque type d'ennemi a des caractéristiques et des comportements uniques.

Elles peuvent avoir 3 état, **attente, fuite, ou vadrouille**, et actuellement il existe 4 types d'ennemis, Calamar, Pieuvre, des Bébés Pieuvres, et Kraken.

Le Calamar parcourt la carte pour collecter les ressources disponibles. Après un certain temps, il quitte le terrain, emportant avec lui ce qu'il a réussi à amasser.

La Pieuvre détecte les plongeurs à proximité et leur dérobe des ressources. Lorsqu'aucun plongeur n'est détecté, elle reste en attente. Les jeunes pieuvres adoptent un comportement similaire, à la différence qu'elles restent proches de leur parent et lui remettent les ressources qu'elles ont pu subtiliser.

Le Kraken est un adversaire redoutable qui attaque les unités contrôlables tout en restant proche de son point d'origine. Bien qu'il soit lent, ses attaques infligent de lourds dégâts. De plus, chaque coup réussi lui permet de récupérer des heat points.

On a deux façons de faire apparaître des ennemis sur le terrain.

Aléatoirement, un point est créé à un endroit aléatoirement et génère un type aléatoire d'ennemis (pas de Kraken) avec une fréquence qui augmente plus le point de spawn se trouve profond et une fois un certain nombre d'ennemis à été créé il est détruit.

Ou **statiquement**, avec des points de spawn qui se trouvent sur le terrain dès le début et ne disparaissent jamais, mais avec une fréquence très réduite et une petite chance de faire apparaître un Kraken.

B. Analysé détaillée :

Fonctionnalité	Difficulté	Priorité
Représenter les unités contrôlables	Facile	1
Représenter les ressources	Facile	1
Représenter les unités ennemis	Facile	1
Menu d'accueil	Facile	3
Déplacement des unités	Moyen	1

Sélection Simple	Facile	1
Sélection Multiple	Moyen	3
Embauche	Facile	2
Vendre	Facile	2
Acheter	Facile	3
Action : se déplacer	Facile	1
Action : récupérer	Facile	1
Action : Attaquer	Facile	2
Action : Shoot	Moyen	2
Action : Stop	Facile	3
Action : Défend	Moyen	3
BackPack + Livraison	Facile	3
Implémenter Sous marin	Moyen	3
Embarquer dans un sous marin	Moyen	3
TimeLine de la ressource	Facile	1
Panneaux de contrôle des actions	Moyen	1
Gestion du Temps de jeu	Facile	2
Gestion des points/ gains	Facile	1
Mécanisme de victoire	Facile	2
Paramétriser une partie	Moyen	3
Terrain avec profondeurs	Moyen	3
Terrain plus grand	Facile	2
MiniMap	Moyen	2
Gestion proximités/Collisions	Difficile	1
GameMaster	Difficile	1
Gestion des spawns enemies	Moyen	1

Gestion des unités ennemis	Difficile	1
Gestion de l'oxygène/stamina/HP	Facile	2
Design pour l'affichage	Facile	1
Gestion des zones de fonctionnement	Moyen	1
Affichage des informations d'une unité	Facile	2

III- Plan de développement :

1. Phase de recherche et documentation :

- Analyse du cahier des charges et définition des objectifs du projet : 1h
- Documentation sur la synchronisation : 20 min
- Documentation intermédiaire : 2h
- Rédaction du rapport final et de la documentation technique complète expliquant les choix d'implémentation, difficultés rencontrées et solutions adoptées : 5h

2. Conception de la vue :

a) Panneau principal et interface globale (GamePanel):

- Mise en place de la grille, affichage des objets (unités, ressources, ennemis) et rendu graphique général : 1h
- Intégration des panneaux d'informations (InfoPanel, et InfoPanelUNC) : 20 min
- Mise en œuvre du rafraîchissement continu via le thread (classe Redessine) : 30 min

b) Market et dialogues associés:

- Implémentation du Market (MarketPopup) permettant l'accès aux actions d'embauche et de vente : 1h
- Implémentation du dialogue d'embauche (EmbaucheDialog), avec prévisualisation de l'unité et déduction du coût + choix: 40 min
- Implémentation du dialogue de vente (SellDialog) avec affichage de la liste des ressources à vendre et gestion de la sélection : 20 min
- Implémentation des achats : 30 min

c) Panneaux d'actions et d'informations:

- Implémentation du panneau de contrôle (InfoPanel) pour afficher les informations de l'unité sélectionnée et déclencher les actions « Se déplacer » et « Récupérer » : 35 min.
- Implémentation de la barre de progression dans le panel des ressources (InfoPanelUNC), avec mise à jour en temps réel selon l'état de la ressource : 20 min.

- Implémentation du panneau de contrôle, pour la sélection multiples et les actions : 1 h.
- Affichage des unités sélectionné et des actions en commun: 30 min
- Afficher toutes les actions pour les différentes unités: 15 min.

d) Augmentation de la taille du terrain :

- Modification des constantes de dimensions du terrain et ajustement des paramètres d'affichage : 30 min
- Mise à jour de la gestion de la caméra pour s'adapter aux nouvelles dimensions : 40 min
- Tests et débogage de la navigation et des interactions avec le terrain agrandi : 30 min

e) MiniMap :

- Mise à jour du ratio d'affichage pour correspondre aux nouvelles dimensions du terrain : 20 min
- Ajout d'un indicateur dynamique de la zone visible par la caméra : 25 min
- Tests et ajustements de la représentation des unités et ressources à l'échelle de la minimap : 30 min

f) Menu et paramétrage :

- Création du menu principal permettant de lancer le jeu, d'accéder aux réglages (paramétrage) et de quitter : 30 min
- Implémentation des options de paramétrage initial (temps de partie, etc.) : 30 min

g) Gestion de fin de partie :

- Intégration de la logique de fin de partie dans VictoryManager : 15 min

h) Graphisme

- Dessin et ajout des images pour les objets et backgrounds : 2 h.

3. Conception Contrôleur :

- Implémenter la sélection simple des unités contrôlables : 1h
- Implémenter la logique des boutons du panneau d'actions pour lancer les modes déplacement et récupération : 15 min
- Gestion des actions liées au Market (ouverture des dialogues) : 30 min

- Modifier la sélection d'un objet, le fonctionnement des actions pour la sélection multiple et débogage : 2h
- Gestion des munitions pour les actions ayant besoin d'attaquer à distance: 30 min
- Gestion de la stamina et de l'oxygène :20 min
- Gestion des proximités et des collisions: 4 h
- Gestion de la création de nouveaux ennemis de manière aléatoire: 2 h
- Mise en place GameMaster: 1h
- Gestion des zones de fonctionnement: 1:30 h
- Optimisation GameMaster avec les zones : 1h
- Optimisation des sélections: 15 min

4. Conception Modèle :

a) Unité et ressources :

- Implémenter les ressources avec leur évolution d'état (**EN_CROISSANCE** / **PRÊTE**) et le code couleur associé : 1h
- Implémenter le sous marin :
- Implémentation des mécanismes spécifiques du sous-marin (embarquement/débarquement automatique des plongeurs, consommation d'essence) : 2h

- implémenter le déplacement des unités : 2h
- rendre le déplacement plus naturel : 30 min
- Implémenter le Plongeur : 30 min
- Implémenter le PlongeurArme: 15 min
- Implémenter les actions attaque, shoot, défend du plongeur armé : 1h
- Implémenter Calamar avec la fuite: 30 min
- Implémenter Pieuvre et leurs bébés : 2 h

- Implémenter Kraken : 30 min
- Implémenter EnnemySpawnPoint : 20 min
- implémenter EpicSpawnPoint: 15 min

b) Système économique et gestion du score :

- Intégrer la gestion des gains et des points de victoire (via la classe Referee) : 30 min
- Mécanismes de paiement pour l'embauche, vente et achat d'items (essence, oxygène, etc.) : 30 min

c) Construction et base :

- Implémentation de la Base : 30 min
- Intégration des interactions entre la Base et les unités (livraison du sac ...etc) : 30 min

d)

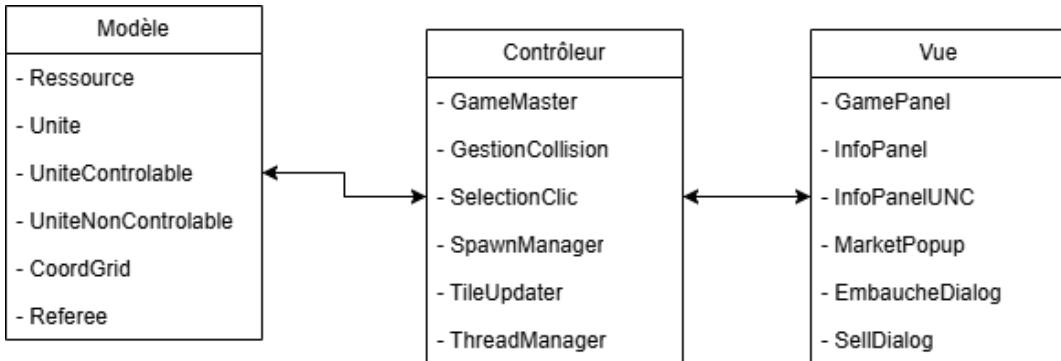
5. Conception générale :

- Intégration de l'ensemble des composants dans l'architecture globale du jeu : 1h
- Tests d'intégration et débogage des interactions entre les différentes classes : 3h
- Tests et débogage des interactions des unités, du fonctionnement des actions, comportement correct des unités non contrôlables et affichage correct du graphisme: 4h

Lien du diagramme de Gantt : [!\[\]\(0b0636dbae614f97346d733ac650473d_img.jpg\) Diagramme de Gantt](#)

IV- Conception :

A) Conception générale :



1. Modèle (Model) :

Les classes du modèle représentent les données et la logique métier de l'application.

- Ressource : Représente une ressource dans le jeu.
- Unite : Classe de base pour toutes les unités du jeu.
- UniteControlable : Unités contrôlables par le joueur (ex: Plongeur).
- UniteNonControlable : Unités non contrôlables par le joueur (ex: Calamar, Pieuvre).
- CoordGrid : Représente les coordonnées d'une case dans une grille.
- Position : Représente une position dans le jeu.
- Referee : Gère les gains et les points du joueur.
- GestionRessource : Gère le cycle de vie des artefacts
- ResourceSpawner : Gère la génération procédurale des ressources.

2. Vue (View) :

Les classes de la vue sont responsables de l'affichage des données à l'utilisateur.

- GamePanel : Panneau principal du jeu qui affiche la grille, les objets, et les unités.
- InfoPanel : Affiche les informations sur les unités sélectionnées.
- InfoPanelUNC : Affiche les informations sur les ressources sélectionnées.
- MarketPopup : Fenêtre popup pour le marché (embauche, vente).
- EmbaucheDialog : Dialogue pour embaucher de nouvelles unités.

- SellDialog : Dialogue pour vendre des ressources.
- MiniMap : Panneau affichant une représentation réduite du terrain avec les unités, ressources et la zone visible par la caméra.

3. Contrôleur (Controller) :

Les classes du contrôleur gèrent les interactions entre le modèle et la vue.

- GameMaster : Contrôleur principal qui gère la logique du jeu.
- GestionCollisions : Gère les collisions entre les objets.
- ProximityChecker : Vérifie la proximité entre les objets.
- SelectionClic : Gère les sélections de l'utilisateur via la souris.
- SpawnManager : Gère la génération des ennemis et des ressources.
- TileUpdater : Met à jour les positions des objets dans la grille.
- ThreadManager : Gère les threads du jeu.
- VictoryManager : Gère les conditions de victoire.

a) Explication des Relations :

- **Modèle** : Les classes du modèle contiennent les données et la logique métier. Elles sont indépendantes de la vue et du contrôleur.
- **Vue** : Les classes de la vue affichent les données du modèle et interagissent avec l'utilisateur. Elles ne contiennent pas de logique métier.
- **Contrôleur** : Les classes du contrôleur gèrent les interactions entre la vue et le modèle. Elles reçoivent les entrées de l'utilisateur, mettent à jour le modèle, et rafraîchissent la vue en conséquence.

V- Conception Détailé :

- Déplacement des unités

1. Structures de données utilisées

Unite : Classe représentant une unité en mouvement, avec des attributs pour sa position (Position), sa destination (Position), sa vitesse (double), et ses vitesses selon les axes (vx, vy).

GamePanel : Classe responsable de l'affichage des objets du jeu et de leur mise à jour graphique.

ThreadManager : Classe qui gère le comptage des threads actifs.

2. Constantes du modèle

DELAY = 30 : Temps d'attente entre chaque itération du thread de déplacement, contrôlant la fluidité du mouvement. Ce delay doit être le même dans la classe GameMaster et doit être supérieure ou égal à celui dans ProximityChecker.

3. Algorithme (description abstraite)

1. Initialiser le thread et enregistrer l'unité associée.

2. Tant que le thread est actif (`running = true`) :

a. Si le jeu est en pause :

- Mettre le thread en attente (pause) jusqu'à ce que le jeu reprenne.
b. Récupérer la destination de l'unité de manière sécurisée (synchronisée).

c. Si la destination est `null` :

- Arrêter le thread (le déplacement est terminé).

d. Calculer le vecteur directionnel vers la destination :

- Calculer la différence en X (`dx`) et en Y (`dy`) entre la position actuelle et la destination.

- Calculer la distance totale à parcourir.

- Calculer l'angle de direction à l'aide de la fonction `atan2` .

e. Si la distance initiale n'a pas encore été définie :

- Enregistrer la distance initiale.

f. Si la distance restante est très faible (proche de la destination) :

- Fixer directement la position de l'unité à la destination.
- Réinitialiser la vitesse et l'accélération de l'unité.
- Arrêter le thread.

g. Sinon (la distance est encore significative) :

- Ajuster la vitesse de l'unité :

- Si la vitesse actuelle est inférieure à la vitesse maximale :
 - Augmenter progressivement la vitesse (phase d'accélération).

- Si la distance restante est supérieure à 50% de la distance initiale

:

- Maintenir la vitesse maximale.

- Si la distance restante est inférieure à 50% de la distance initiale :

- Réduire progressivement la vitesse (phase de décélération).

- Calculer les composantes du vecteur vitesse ('vx' et 'vy') en fonction de l'angle et de la vitesse actuelle.

- Mettre à jour la position de l'unité en ajoutant les composantes de vitesse à sa position actuelle.

h. Mettre à jour l'affichage du jeu ('repaint').

i. Mettre le thread en pause pour un court délai (par exemple, 30 ms) avant de recalculer.

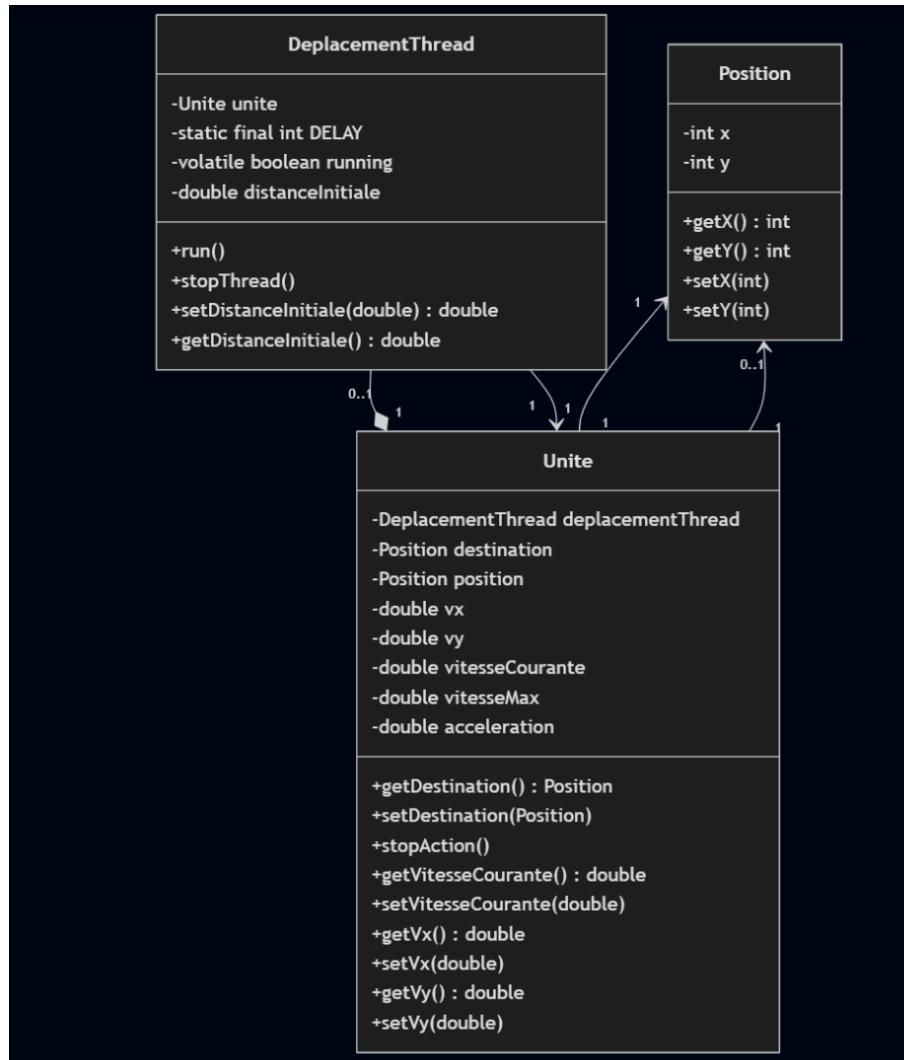
4. Utilisation du code par d'autres fonctionnalités :

Gestion du déplacement : Chaque unité ayant un DeplacementThread peut être déplacée indépendamment.

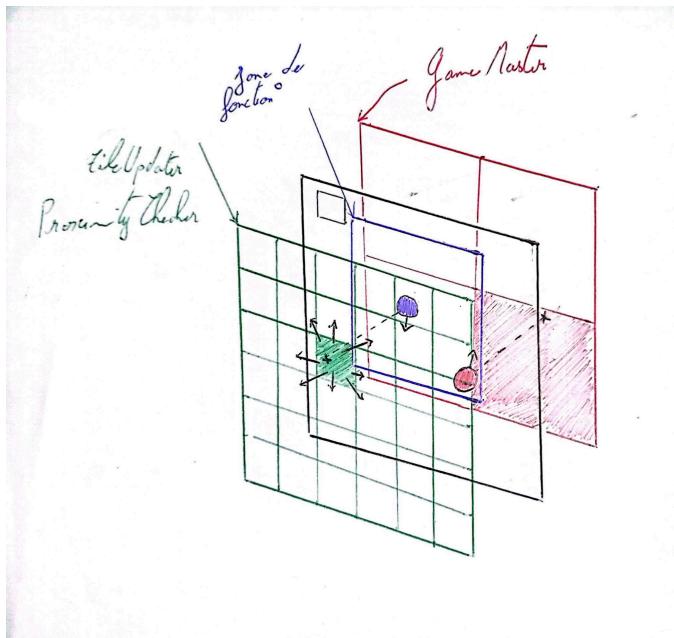
Gestion de la suppression d'unités : GamePanel.removeObjet() est utilisé pour retirer une unité du jeu.

Intégration avec ThreadManager : Permet de suivre et contrôler l'exécution des threads de déplacement.

5. Diagramme de classes simplifié



- **Gestion des collisions et interactions:**



1. Structures de données utilisées

a) Gestion des objets et de leurs positions

CoordGrid : Coordonnées discrètes des objets dans une grille de tuiles.

ConcurrentHashMap<CoordGrid, CopyOnWriteArrayList> : Structure permettant d'associer chaque tuile à une liste d'objets présents. C'est le conteneur principal de tous les objets sur le jeu.

b) Gestion des interactions

GestionCollisions : Contient les fonctions de détection de collision entre objets.

TileManager : Gère la division de la carte en tuiles et assigne des objets à ces tuiles.

TileUpdater : Met à jour la position des objets et leur appartenance aux tuiles.

ProximityChecker : Vérifie quels objets sont en collisions/proximités

2. Constantes du modèle

1. Runtime.getRuntime().availableProcessors()

- Rôle : Détermine le nombre de processeurs disponibles sur la machine pour créer un pool de threads.
- Utilisation : Permet de paralléliser les mises à jour des objets sur plusieurs threads pour améliorer les performances.

2. 16 (dans Thread.sleep(16))

- Rôle : Définit le délai entre chaque itération du thread TileUpdater.
- Utilisation : Ce délai correspond à environ 60 FPS (1000 ms / 60 ≈ 16 ms), garantissant une mise à jour fluide des objets.

3. batchSize = Math.max(1, objets.size()) /

Runtime.getRuntime().availableProcessors()

- Rôle : Définit la taille des lots d'objets à traiter par chaque thread dans le pool.
- Utilisation : Permet de diviser les objets à mettre à jour en lots équilibrés pour chaque processeur disponible.

4. TILESIZE = 200

- Rôle : Définit la taille (en pixels) d'une tuile sur la carte.
- Utilisation : Permet de diviser la carte en une grille régulière où chaque tuile mesure 200x200 pixels. Cette taille est utilisée pour convertir des positions en coordonnées de grille et pour gérer les objets dans des zones spécifiques.

5. nbTilesWidth = GamePanel.TERRAIN_WIDTH / TILESIZE

- Rôle : Définit le nombre total de tuiles en largeur sur la carte.
- Utilisation : Permet de calculer combien de tuiles couvrent la largeur totale du terrain. Cela dépend de la largeur du terrain définie dans GamePanel.

6. nbTilesHeight = GamePanel.TERRAIN_HEIGHT / TILESIZE

- Rôle : Définit le nombre total de tuiles en hauteur sur la carte.
- Utilisation : Permet de calculer combien de tuiles couvrent la hauteur totale du terrain. Cela dépend de la hauteur du terrain définie dans GamePanel.

7. coordTiles = new CoordGrid[nbTilesWidth][nbTilesHeight]

- Rôle : Représente une grille bidimensionnelle contenant les coordonnées de chaque tuile.
- Utilisation : Permet de stocker et d'accéder rapidement aux coordonnées de chaque tuile sur la carte. Chaque élément de cette grille correspond à une instance de CoordGrid.

3. Algorithme (description abstraite)

1. Initialiser le thread avec :

- Une carte des objets (`objetsMap`) organisée par coordonnées de grille.
- Une liste des unités contrôlables (`unitesEnJeu`).

2. Tant que le thread est actif (`running = true`):

a. Parcourir toutes les unités contrôlables dans `unitesEnJeu`.

b. Pour chaque unité :

i. Vérifier si l'unité est en dehors de la zone visible de la caméra :

- Si oui, créer une zone dynamique autour de l'unité.
- Sinon, supprimer la zone dynamique associée à l'unité.

ii. Récupérer tous les objets proches de l'unité :

- Objets dans la même tuile (`getObjetDansMemeTile`).
- Objets dans les tuiles voisines (`getObjetTilesVoisines`).

iii. Pour chaque objet voisin :

- Si l'unité est un **Plongeur** :

- Si le voisin est une ressource ciblée :
- Vérifier si le plongeur est à portée.
- Si oui, collecter la ressource et réinitialiser la cible.
- Si le voisin est un ennemi :
- Si le plongeur est armé et en mode défense :
- Vérifier si l'ennemi est dans le cercle de défense.
- Si oui, tirer sur l'ennemi.
- Si le plongeur est en mode attaque :
- Vérifier si l'ennemi est à portée.
- Si oui, infliger des dégâts à l'ennemi.

- Si le voisin est un calamar :

- Vérifier si le calamar est dans le périmètre de fuite.
- Si oui, forcer le calamar à fuir.
- Si le voisin est une pieuvre ou un céphalopode :
- Marquer l'unité comme cible pour la pieuvre. (notre unité à était

détecté)

- Si c'est une pieuvre bébé, transmettre la cible aux autres bébés.
- Si le voisin est une base :
- Vérifier si le plongeur est à l'intérieur de la base.
- Si oui :
- Livrer le sac à dos.
- Recharger l'oxygène.
- Si le plongeur est armé, recharger ses munitions.

- Vérifier les collisions avec le voisin :
- Si une collision est détectée, empêcher le chevauchement.

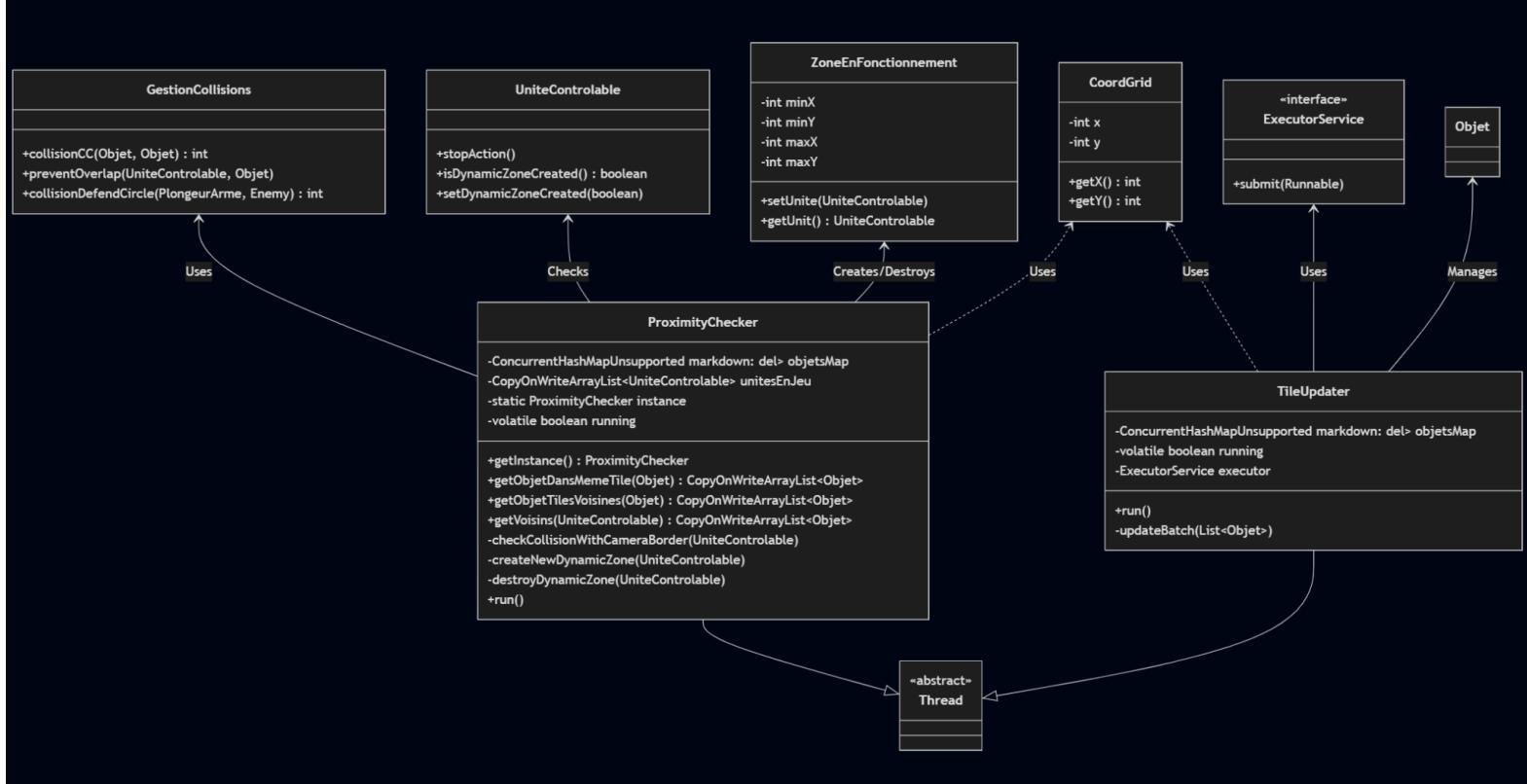
c. Attendre un court délai (par exemple, 30 ms) avant de recommencer.

4. Utilisation du code par d'autres fonctionnalités :

TileUpdater met à jour la position des objets et les assigne aux bonnes tuiles.

ProximityChecker est utilisé pour détecter les proximités entre les objets et les unités contrôlables, principalement pour assurer qu'il n'y est pas de chevauchement entre objets.

5. Diagramme de classes simplifié



- **Gestion gamemaster**

1. Structures de données utilisées

ConcurrentHashMap<CoordGrid, CopyOnWriteArrayList<Enemy>>
enemiesGrid: Organise les ennemis en fonction de leur position dans une grille spatiale.

CopyOnWriteArrayList<Enemy> enemies : Stocke la liste globale des ennemis actifs.

CoordGrid[][] Grid : Représente une grille fixe pour localiser les cellules spatiales.

ExecutorService enemyExecutor: Exécute les actions des ennemis en parallèle.

Set<Enemy> processedEnemies: Suit les ennemis déjà traités pour éviter les doublons.

2. Constantes du modèle

DELAY : Définit le délai entre les itérations de la boucle principale.

CELL_SIZE : Définit la taille d'une cellule dans la grille spatiale.

GRID_WIDTH : Définit le nombre total de cellules en largeur dans la grille spatiale.

GRID_HEIGHT : Définit le nombre total de cellules en hauteur dans la grille spatiale.

Grid : Représente une grille bidimensionnelle contenant les coordonnées des cellules.

3. Algorithme (description abstraite)

1. Initialiser le thread

2. Récupérer les coins de la base principale pour vérifier les collisions avec les ennemis.

3. Tant que le thread est actif :

a. Mettre à jour la grille spatiale (`updateGrid`):

- Effacer la grille actuelle.

- Pour chaque ennemi vivant :

- Calculer la cellule de la grille correspondant à sa position.

- Ajouter l'ennemi à la cellule correspondante dans la grille.

b. Créer un ensemble temporaire `processedEnemies` pour suivre les ennemis déjà traités.

c. Gérer les ennemis dans la zone principale :

i. Récupérer tous les ennemis dans la zone principale (`getEnemiesInZone`).

ii. Pour chaque ennemi dans la zone principale :

- Si l'ennemi n'a pas encore été traité :

- Ajouter l'ennemi à `processedEnemies`.

- Soumettre une tâche au pool de threads (`enemyExecutor`)

pour gérer cet ennemi :

- Vérifier si l'ennemi est hors des limites du terrain ou dans la base principale :

- Si oui, supprimer l'ennemi.

- Si l'ennemi est un calamar :

- Mettre à jour les ressources disponibles pour le calamar.

- Exécuter l'action principale de l'ennemi (`enemy.action()`).

d. Gérer les ennemis dans les zones dynamiques :

i. Pour chaque zone dynamique :

- Récupérer tous les ennemis dans la zone (`getEnemiesInZone`).

- Pour chaque ennemi dans la zone dynamique :

- Ajouter l'ennemi à `processedEnemies`.

- Soumettre une tâche au pool de threads (`enemyExecutor`)

pour gérer cet ennemi :

- Si l'ennemi est un calamar :

- Mettre à jour les ressources disponibles pour le calamar.

- Exécuter les actions de vadrouille et d'interaction de l'ennemi :
 - `enemy.vadrouille()`
 - `enemy.action()`
 - Vérifier si l'ennemi est hors des limites du terrain ou dans la base principale :
 - Si oui, supprimer l'ennemi.
- e. Gérer les ennemis en dehors des zones actives :
- i. Pour chaque ennemi dans la liste globale des ennemis (`enemies`) :
- Si l'ennemi n'est pas dans `processedEnemies` :
 - Arrêter les actions de l'ennemi (`enemy.stopAction()`).
 - Si l'ennemi est marqué comme étant dans une zone active :
 - Arrêter tous les threads associés à l'ennemi.
 - Marquer l'ennemi comme étant en dehors des zones actives.
- f. Attendre un court délai (`Thread.sleep(Delay)`) avant de recommencer.

4. Utilisation du code par d'autres fonctionnalités :

GamePanel : Gère l'affichage et les interactions utilisateur. Ajout/suppression d'objets, mise à jour des cibles, gestion des unités tuées.

SpawnManager : Gère la création et la gestion des points de spawn. Ajout d'ennemis, mise à jour des cibles après génération.

ProximityChecker : Déetecte les interactions entre unités contrôlables et objets proches. Accès aux ennemis et ressources, mise à jour des cibles.

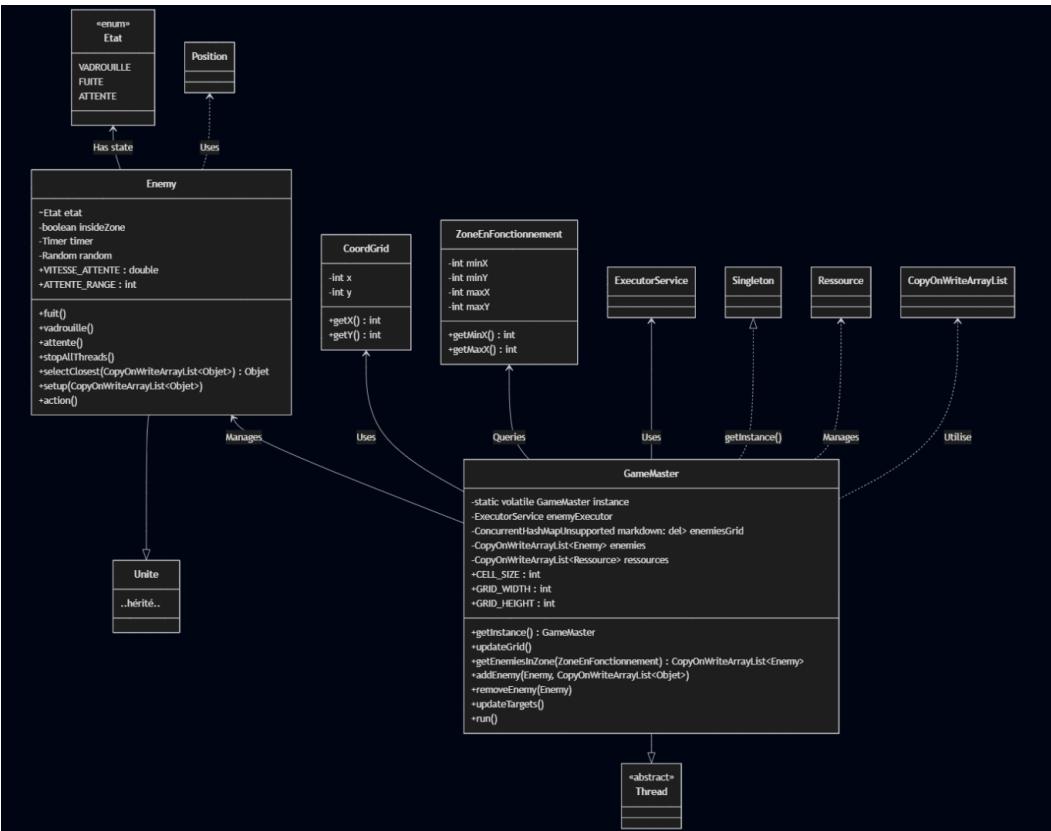
VictoryManager : Gère les conditions de victoire et de défaite. Suivi des ressources et des ennemis pour vérifier les conditions de victoire.

SelectionClic : Gère les clics de souris pour sélectionner des unités ou interagir avec des objets. Ciblage des ennemis et interaction avec les ressources.

TileUpdater : Met à jour les objets sur la carte en fonction de leur position. Accès aux ennemis dans les cellules de la grille.

ZoneMover : Gère les zones dynamiques autour des unités contrôlables. Accès aux ennemis dans les zones dynamiques pour mettre à jour leurs interactions.

5. Diagramme de classes simplifié



- **Gestion des sélections**

1. Structures de données utilisées

panel.getUnitesSelected() : Gère la liste des unités sélectionnées.
Ajout/suppression d'unités sélectionnées, réinitialisation de la sélection.

CopyOnWriteArrayList<Objet> objectsInCell : Stocke les objets dans une cellule spécifique. Vérifie les objets affectés par un clic ou une sélection.

ArrayList<CoordGrid> neighbors: Stocke les coordonnées des cellules voisines. Vérifie les objets dans les cellules voisines lors d'un clic proche des bords.

Rectangle selectionRect: Représente une zone de sélection. Vérifie quelles unités se trouvent dans la zone de sélection dessinée par le joueur.

2. Constantes du modèle

_MAX_SELECTED_UNITS : Limite le nombre d'unités sélectionnées.
Empêche la sélection de plus d'unités que le maximum autorisé.

TileManager.TILESIZE : Définit la taille d'une cellule sur la carte. Vérifie si un clic est proche des bords d'une cellule et accède aux cellules voisines.

GamePanel.TERRAIN_WIDTH et GamePanel.TERRAIN_HEIGHT :
Définissent les dimensions totales du terrain. S'assurent que les coordonnées calculées restent dans les limites du terrain.

GamePanel.VIEWPORT_WIDTH et GamePanel.VIEWPORT_HEIGHT :
Définissent les dimensions de la zone visible. Vérifient si un clic est à l'intérieur de la zone visible.

GamePanel.MINIMAP_SCALE_X et GamePanel.MINIMAP_SCALE_Y :
Définissent les échelles de conversion entre minimap et monde. Convertissent les coordonnées d'un clic sur la minimap en coordonnées du monde.

3. Algorithme (description abstraite)

mousePressed :

1. Convertir les coordonnées du clic de la souris en coordonnées du monde.
2. Enregistrer les coordonnées de départ pour la sélection (startXWorld, startYWorld).
3. Si le clic est en dehors des limites du terrain :
 - Afficher un message et arrêter le traitement.
4. Si le clic est sur la minimap :
 - Convertir les coordonnées de la minimap en coordonnées du monde.
 - Centrer la caméra sur ce point.
 - Arrêter le traitement.
5. Si le clic est un clic gauche (BUTTON1) :
 - a. Si une action de tir est en attente :
 - Gérer l'action de tir ('handleShootingAction').
 - Réinitialiser l'état de l'action de tir.
 - Arrêter le traitement.
 - b. Gérer la sélection d'objets ou d'unités ('handleSelection').
 - c. Si le mode récupération est activé :
 - Gérer la récupération ('handleRecuperationMode').
 - d. Si le mode attaque est activé :
 - Gérer l'attaque ('handleAttackingMode').
 - e. Si le mode déplacement est activé :
 - Gérer le déplacement ('handleDeplacementMode').
6. Si le clic est un clic droit (BUTTON3) :
 - Gérer l'ordre de déplacement ('handleRightClickOrder').

handleSelection :

1. Si aucun mode (récupération, attaque, déplacement) n'est activé :
 - Réinitialiser la sélection actuelle ('dropUnitesSelectionnees').
2. Activer le mode de sélection ('isSelecting = true').
3. Calculer les coordonnées de la cellule cliquée et les restes (restX, restY).
4. Récupérer les objets dans la cellule cliquée.
5. Si le clic est proche des bords de la cellule :
 - a. Calculer les coordonnées des cellules voisines.
 - b. Ajouter les objets des cellules voisines à la liste des objets à vérifier.
6. Pour chaque objet dans les cellules pertinentes :
 - a. Vérifier si le clic est à l'intérieur des limites de l'objet.
 - b. Si l'objet est une unité contrôlable :
 - Ajouter l'unité à la sélection.
 - Mettre à jour l'interface utilisateur pour afficher les informations de l'unité.
 - c. Si l'objet est une ressource :
 - Mettre à jour l'interface utilisateur pour afficher les informations de la ressource.
 - Si le mode récupération est activé, définir la ressource comme cible.

d. Si l'objet est un ennemi :

- Mettre à jour l'interface utilisateur pour afficher les informations de l'ennemi.

- Si le mode attaque est activé, définir l'ennemi comme cible.

7. Si aucun objet n'est sélectionné :

- Réinitialiser l'état de la sélection et de l'interface utilisateur.

selectUnitsInRectangle()

1. Calculer les dimensions du rectangle de sélection en fonction des coordonnées de départ et de fin.

2. Pour chaque unité contrôlable dans le jeu :

- a. Si la position de l'unité est à l'intérieur du rectangle de sélection :

- Ajouter l'unité à la sélection (si la limite maximale n'est pas atteinte).
- Marquer l'unité comme sélectionnée.

3. Mettre à jour l'interface utilisateur pour afficher les informations des unités sélectionnées.

4. Utilisation du code par d'autres fonctionnalités :

GamePanel : Gère l'affichage et les interactions utilisateur. Ajout comme écouteur de souris, gestion des sélections, modes d'interaction (déplacement, attaque, etc.).

KeyboardController : Gère les interactions utilisateur via le clavier. Applique les actions du clavier aux unités sélectionnées via SelectionClic.

MinimapPanel : Affiche une vue réduite de la carte. Gère les clics sur la minimap pour centrer la caméra.

InfoPanel : Affiche des informations sur les unités ou objets sélectionnés. Met à jour les informations affichées en fonction des sélections.

5. Diagramme de classes simplifié



- **Gestion des zones de fonctionnement**

- 1. Structures de données utilisées**

minX, minY, maxX, maxY : Définissent les limites de la zone en termes de coordonnées.

unit : Représente l'unité associée à une zone dynamique.

- 2. Constantes du modèle**

MAX_WIDTH : Largeur maximale d'une zone, incluant un buffer.

MAX_HEIGHT : Hauteur maximale d'une zone, incluant un buffer.

- 3. Algorithme (description abstraite)**

- **Définition des Zones :**

1. **Une zone est définie par ses limites (minX, minY, maxX, maxY), qui représentent un rectangle sur la carte.**
2. **Les zones peuvent être statiques (comme la zone principale) ou dynamiques (comme les zones autour des unités contrôlables).**

- **Vérification de la Position :**

1. **Les méthodes `isInsideMain` et `isInsideDynamic` permettent de vérifier si une position donnée se trouve à l'intérieur des limites de la zone.**

- **Gestion des Zones Dynamiques :**

1. **Une unité contrôlable peut être associée à une zone dynamique via `setUnite`. Cette unité est utilisée pour ajuster les limites de la zone en fonction de sa position.**

4. Utilisation du code par d'autres fonctionnalités :

GamePanel : Gère la zone principale et les zones dynamiques.

Ajoute/supprime des zones dynamiques, vérifie si une position est dans une zone.

ProximityChecker : Vérifie les interactions entre unités et objets proches.

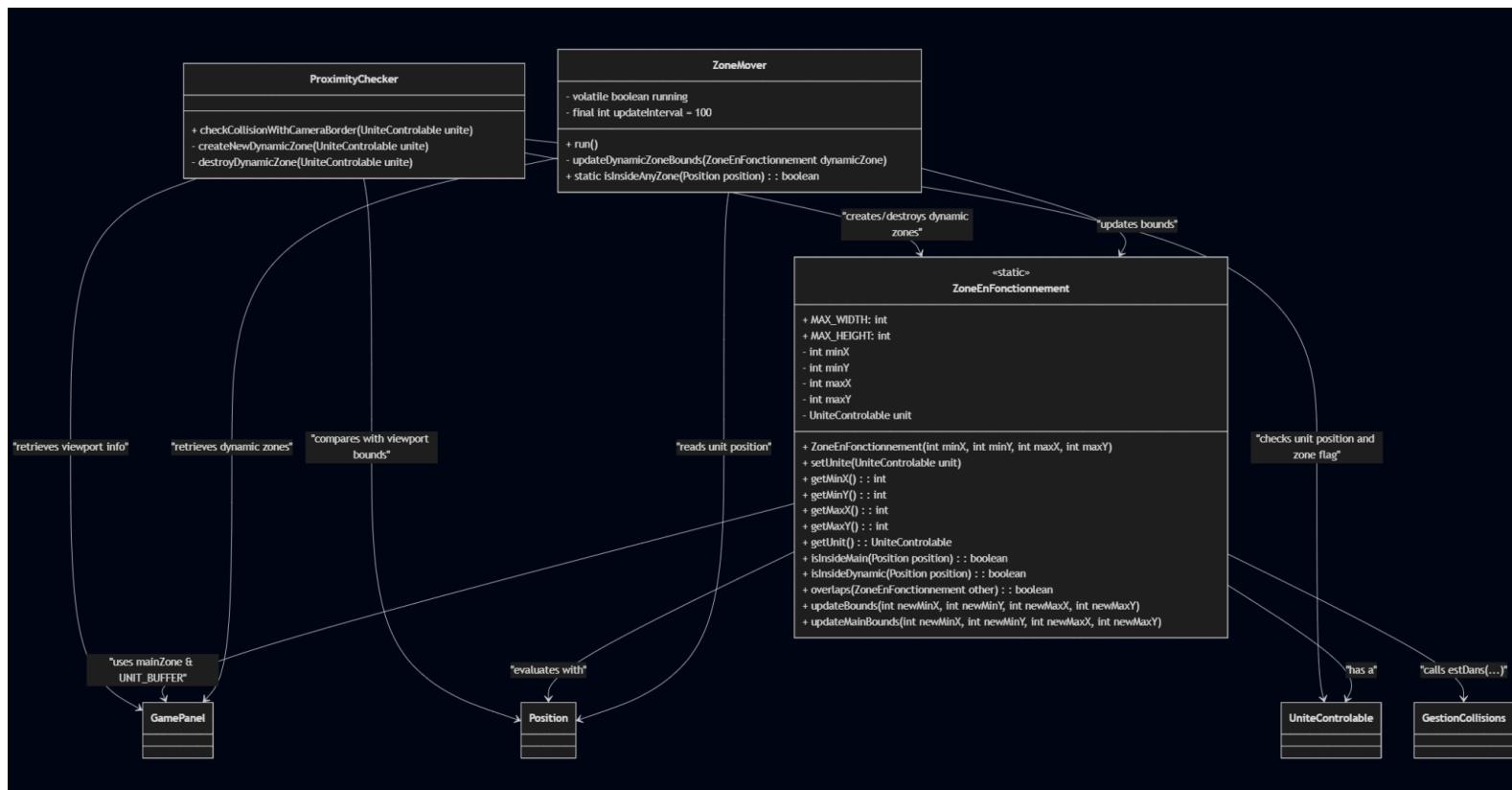
Crée ou supprime des zones dynamiques en fonction des déplacements des unités.

ZoneMover : Met à jour les limites des zones dynamiques. Ajuste les limites des zones dynamiques en fonction des déplacements des unités associées.

GameMaster : Gère les ennemis dans les zones actives. Récupère les ennemis dans les zones actives, met à jour leurs cibles.

SpawnManage : Génère des points de spawn pour les ennemis. Sélectionne une zone active pour y placer un point de spawn.

5. Diagramme de classes simplifié



- **Gestion des Spawns enemies**

1. Structures de données utilisées

CopyOnWriteArrayList<EnemySpawnPoint> spawnPoints : Stocke les points de spawn standards.

CopyOnWriteArrayList <EpicSpawnPoint>epicSpawnPoints : Stocke les points de spawn épiques, c'est-à-dire celles qui ne disparaissent jamais.

Random random : Génère des valeurs aléatoires pour les positions, types d'ennemis, et intervalles.

ZoneEnFonctionnement zone: Limite les spawns à des zones actives spécifiques.

2. Constantes du modèle

RAYON : Définit le rayon autour d'un point de spawn pour générer des ennemis.

CONSTANT_SPAWN_INTERVAL : Définit l'intervalle constant entre les spawns d'ennemis épiques.

maxAttempts : Limite le nombre de tentatives pour générer une position valide.

spawnInterval: Définit l'intervalle entre les spawns pour un point de spawn spécifique.

depth : Ajuste l'intervalle de spawn en fonction de la profondeur de la carte.

3. Algorithme (description abstraite)

SpawnManager :

1. **Initialiser les points de spawn épiques via `epicSpawnPoints()` :**
 - Ajouter des points de spawn épiques à des positions prédefinies.
 - Démarrer un thread pour chaque point de spawn épique.
2. **Tant que le thread de SpawnManager est actif :**

- a. Générer un point de spawn standard aléatoire via `generateRandomSpawnPoint` :
 - Sélectionner une zone active (zone principale ou dynamique).
 - Générer une position aléatoire dans cette zone.
 - Ajouter un point de spawn standard à cette position.
- b. Attendre un intervalle aléatoire entre 1 et 60 secondes.

generateRandomSpawnPoint :

1. Récupérer toutes les zones actives (zone principale et zones dynamiques).
2. Sélectionner une zone active aléatoire.
3. Générer une position aléatoire dans la zone sélectionnée via `generateRandomPositionInZone` :
 - Répéter jusqu'à trouver une position valide ou atteindre le nombre maximum de tentatives.
4. Calculer la profondeur de la carte à cette position.
5. Ajouter un point de spawn standard à cette position avec un intervalle ajusté en fonction de la profondeur :
 - Intervalle = max(60 secondes - profondeur * 1 seconde, 20 secondes).

EpicSpawnPoint :

Tant que le thread est actif :

- a. Générer un ennemi via `spawnEnemy` :
 - Générer une position aléatoire dans le rayon du point de spawn.
 - Générer un nombre aléatoire pour déterminer le type d'ennemi :
 - 70% de chance : Ne rien générer.
 - 25% de chance : Générer une `Pieuvre`.
 - 5% de chance : Générer un ou plusieurs `Krakens` (en fonction du nombre d'unités contrôlables).
 - Ajouter les ennemis générés au jeu via `GameMaster.addEnemy`.
- b. Attendre l'intervalle constant défini (60 secondes).

4. Utilisation du code par d'autres fonctionnalités :

GamePanel : Gère l'affichage des points de spawn et des ennemis générés. Affiche les points de spawn, fournit les zones actives pour limiter les spawns.

GameMaster : Gère les ennemis générés par les points de spawn. Ajoute les ennemis générés aux listes globales, met à jour leurs cibles.

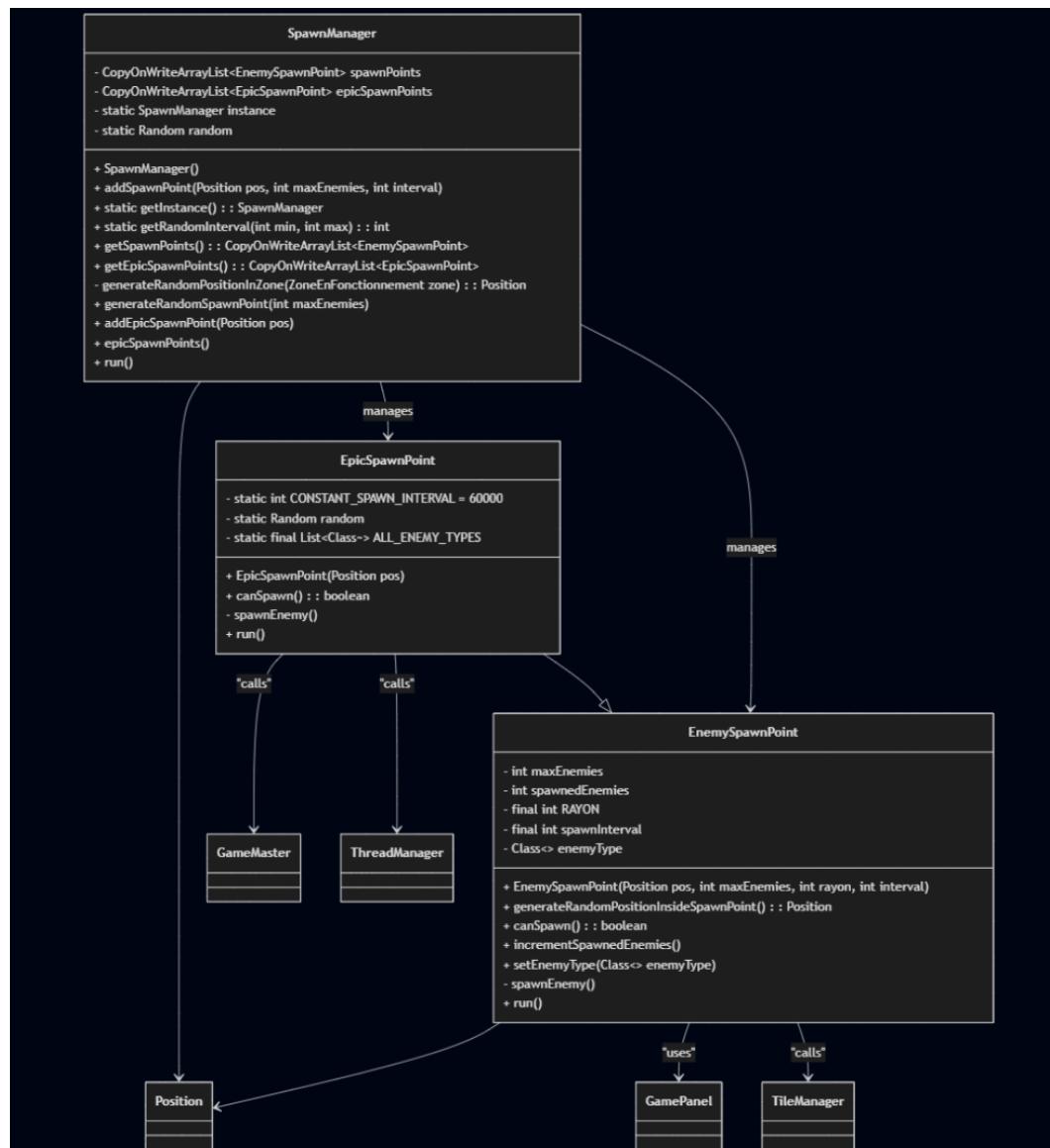
SpawnManager : Classe principale pour gérer les points de spawn et les spawns d'ennemis. Ajoute des points de spawn, génère des positions aléatoires, et contrôle les intervalles.

EnemySpawnPoint : Représente un point de spawn standard pour les ennemis. Génère des ennemis à des intervalles réguliers et une fois un certain nombre atteint il est détruit.

EpicSpawnPoint : Représente un point de spawn épique pour les ennemis rares. Génère des ennemis rares comme les Krakens à des intervalles constants et n'est jamais détruit.

ZoneEnFonctionnement : Limite les spawns à des zones actives spécifiques. Utilisé pour générer des positions aléatoires dans des zones actives.

5. Diagramme de classes simplifié



- **Gestion des unités non contrôlables**

1. Structures de données utilisées

CopyOnWriteArrayList<UniteControlable> targetsDisponibles : Stocke les cibles potentielles pour l'unité non contrôlable.

UniteControlable target : Représente la cible actuelle de l'unité non contrôlable.

CopyOnWriteArrayList<Ressource> ressourcesDisponibles : Stocke les ressources disponibles pour l'unité non contrôlable.

Random random : Génère des valeurs aléatoires pour les actions et déplacements.

Timer timer : Gère les délais pour des actions spécifiques.

Etat etat : Représente l'état actuel de l'unité non contrôlable.

Position destination : Représente la destination actuelle de l'unité non contrôlable.

2. Constantes du modèle

VITESSE_ATTENTE : Définit la vitesse de déplacement en mode attente.

ATTENTE_RANGE : Définit la portée maximale pour les déplacements aléatoires en mode attente.

MAX_DISTANCE : Définit la distance maximale qu'une unité peut parcourir avant de revenir à l'origine.

SAFE_STALKING_DISTANCE : Définit la distance minimale de sécurité pour traquer une cible.

STALKING_DISTANCE : Définit la distance idéale pour traquer une cible.

DAMAGETENTACLE : Définit les dégâts infligés par le Kraken.

ATTACK_INTERVAL : Définit l'intervalle minimum entre deux attaques du Kraken.

3. Algorithme (description abstraite)

1. Calamar

Comportements Principaux :

1. Collecte de Ressources :

- Le calamar sélectionne la ressource la plus proche parmi les ressources disponibles.
- Si une ressource est à portée, il la collecte et l'ajoute à son inventaire.
- Une fois la ressource collectée, elle est retirée de la liste des ressources disponibles.

2. Fuite :

- Si le calamar est menacé (par exemple, lorsqu'un plongeur le fait fuir), ou le temps le timer qu'il possède c'est écoulé il entre en mode fuite.
- Il se déplace vers le bord le plus proche de la carte et quitte la zone.

3. Vérification des Objectifs :

- Si l'objectif actuel (ressource ciblée) n'est plus valide (par exemple, si elle a été collectée par un autre calamar), il sélectionne une nouvelle ressource.

États :

- Vadrouille : Se déplace aléatoirement lorsqu'il n'a pas de ressource ciblée.
- Fuite : Quitte la zone lorsqu'il est menacé.

2. Pieuvre

Comportements Principaux :

1. Vol de Ressources :

- La pieuvre cible un plongeur et tente de voler toutes les ressources de son sac à dos.
- Si elle réussit, les ressources sont rajoutée à son sac.

2. Gestion des Enfants :

- La pieuvre peut avoir des bébés pieuvres associés.
- Lorsqu'elle repère une cible, elle transmet cette cible à ses bébés pieuvres pour qu'ils l'attaquent ou la volent.

3. Déplacement :

- Se déplace vers sa cible (plongeur ou ressource) pour interagir avec elle.

États :

- **Attente** : Se déplace aléatoirement lorsqu'elle n'a pas de cible.
- **Vadrouille** : Se déplace vers un plongeur pour voler une ressource.

3. Pieuvre Bébé

Comportements Principaux :

1. Vol de Ressources :

- Comme la pieuvre adulte, la pieuvre bébé cible un plongeur pour voler une ressource.
- Une fois la ressource volée, elle la ramène à sa mère (la pieuvre adulte).

2. Transmission des Cibles :

- Si une cible est identifiée, la pieuvre bébé peut transmettre cette cible à ses frères et sœurs pour coordonner une attaque.

3. Retour à la Mère :

- Après avoir volé une ressource, elle retourne à la position de sa mère pour déposer la ressource.

États :

- Attente : Se déplace aléatoirement lorsqu'elle n'a pas de cible.
- Vadrouille : Se déplace vers un plongeur pour voler une ressource ou retourne à la position de sa mère après avoir volé une ressource.

4. Kraken

Comportements Principaux :

1. Attaque des Plongeurs :

- Le Kraken cible un plongeur et se déplace vers lui.
- Si le plongeur est à portée d'attaque, le Kraken inflige des dégâts.

2. Régénération :

- Lorsqu'il attaque un plongeur, le Kraken récupère des points de vie en fonction des dégâts infligés.

3. Limite de Distance :

- Si le Kraken s'éloigne trop de sa position d'origine, il retourne à cette position.

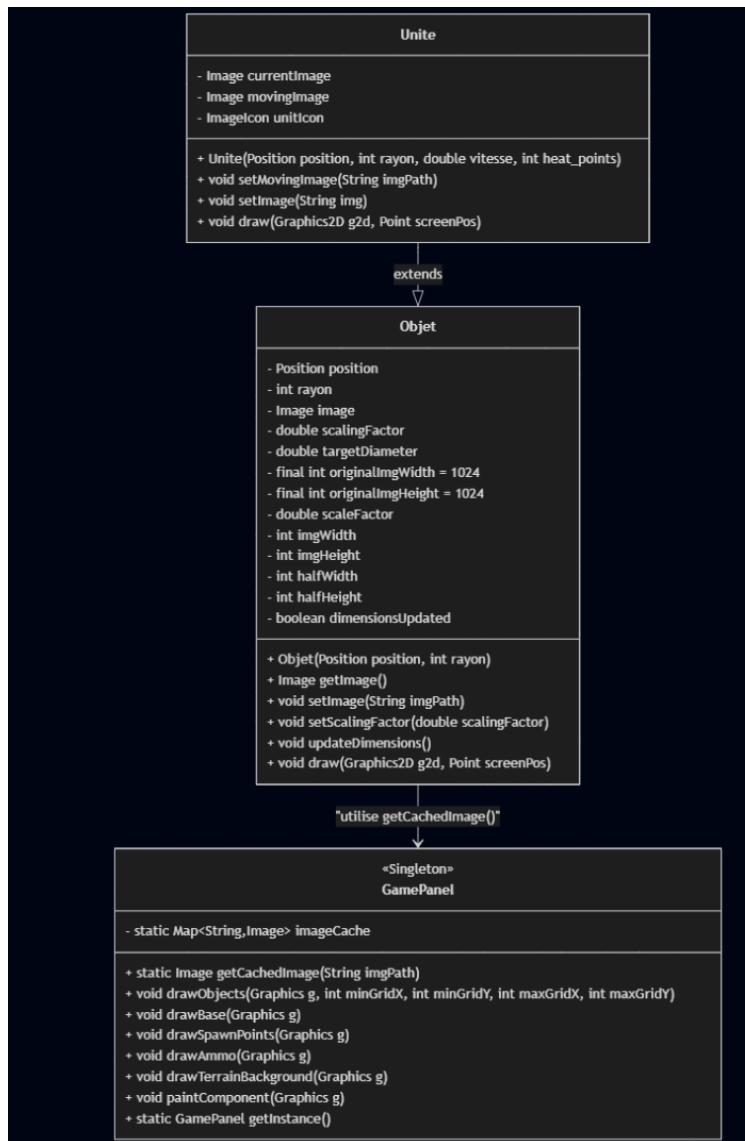
États :

- Attente : Se déplace aléatoirement lorsqu'il n'a pas de cible.
- Vadrouille : Se déplace vers un plongeur pour l'attaquer ou Retourne à sa position d'origine lorsqu'il s'éloigne trop.

4. Utilisation du code par d'autres fonctionnalités :

- **GamePanel** : gère l'affichage et la suppression des objets.
- **GameMaster** : suit l'état global des ennemis.
- **GestionCollisions**: Vérifie les collisions et applique des ajustements de position ou des rebonds.
- **SpawnManager** : assure le renouvellement des ennemis.
- **ProximityChecker** : Gère les collisions, les vols de ressources, et les attaques.

5. Diagramme de classes simplifié



- **Gestion des unités contrôlables**

1. **Structures de données utilisées**

List<Ressource>backpack : Représente le sac à dos du plongeur.

Ressource targetResource : Représente la ressource actuellement ciblée par le plongeur.

Position defendCircleCenter : Centre du cercle de défense du plongeur armé.

defendCircleRadius : Rayon du cercle de défense.

boolean isDefending : Indique si le plongeur armé est en mode défense.

ammo : Nombre de munitions disponibles pour le plongeur armé.

Unite target : Cible actuelle du plongeur armé.

long lastShotTime : Moment du dernier tir effectué par le plongeur armé.

AmmoManager : Gère le fonctionnement des munitions une fois le plongeurArme tire sur quelque chose, son fonctionnement ressemble celui de proximityChecker.

2. **Constantes du modèle**

CAPACITE_SAC : Capacité maximale du sac du plongeur.

MAX_OXYGEN : Niveau maximal d'oxygène du plongeur.

MAX_STAMINA : Niveau maximal d'endurance du plongeur.

BACKPACK_CAPACITY : Capacité maximale du sac à dos du plongeur.

MAX_AMMO : Nombre maximum de munitions du plongeur armé.

SHOOTING_RANGE : Portée maximale des tirs du plongeur armé.

SHOOTING_INTERVAL : Intervalle minimum entre deux tirs.

DAMAGE : Dégâts infligés par un tir du plongeur armé.

COOLDOWN_FUITE : Temps de recharge avant que le plongeur puisse fuir à nouveau.

3. Algorithme (description abstraite)

1. Plongeur

Comportements Principaux :

1. Collecte de Ressources :

- Le plongeur cible une ressource proche et se déplace vers elle.
- Une fois à portée, il collecte la ressource et l'ajoute à son sac à dos.

2. Livraison des Ressources :

- Une fois à la base, il décharge les ressources collectées.

3. Déplacement :

- Le plongeur se déplace vers une position spécifiée par le joueur.
- Si dans la base, il récupère son oxygène.

4. Fuite :

- Le plongeur fait fuire les calamars qui se retrouvent dans un certain périmètre.

5. Gestion de l'Oxygène :

- Le plongeur consomme de l'oxygène en fonction de sa profondeur.
- Si l'oxygène atteint zéro avant qu'il n'atteigne une base, le plongeur meurt.

6. Gestion de l'Endurance :

- Le plongeur utilise son endurance pour se déplacer rapidement.
- Si l'endurance est faible, sa vitesse de déplacement diminue.
- L'endurance se régénère lorsque le plongeur est immobile.

2. Plongeur Armé

Comportements Principaux :

1. Défense :

- Le plongeur armé peut entrer en mode défense, où il surveille un périmètre défini autour de lui (cercle de défense).
- Si un ennemi entre dans ce périmètre, il tire automatiquement pour l'éloigner ou le détruire.

2. Attaque :

- **Le plongeur armé peut recevoir un ordre d'attaquer un ennemi spécifique.**
- **Il se déplace vers l'ennemi.**
- **Une fois à portée, il attaque l'ennemi une seule fois.**

3. Gestion des Munitions:

- **Le plongeur armé utilise des munitions pour tirer sur les ennemis.**
- **Si la balle arrive à la position elle est détruite et lorsqu'elle touche une unité pendant qu'elle se déplace alors l'unité reçoit des dégâts et la balle est détruite .**
- **Si ses munitions sont épuisées, il doit retourner à la base pour se réapprovisionner.**
- **Une fois à la base, il recharge ses munitions et son oxygène.**

4. Déplacement :

- **Comme le plongeur standard, le plongeur armé peut recevoir un ordre de déplacement vers une position spécifique.**

5. Fuite :

- **Pareil qu'avec le plongeur.**

6. Gestion de l'Oxygène :

- **Comme le plongeur standard, le plongeur armé consomme de l'oxygène en fonction de sa profondeur.**

4. Utilisation du code par d'autres fonctionnalités :

GamePanel : Affiche les plongeurs, met à jour leurs informations dans l'interface utilisateur.

ProximityChecker: Gère la collecte de ressources et les tirs sur les ennemis. Vérifie les interactions entre plongeurs et objets proches.

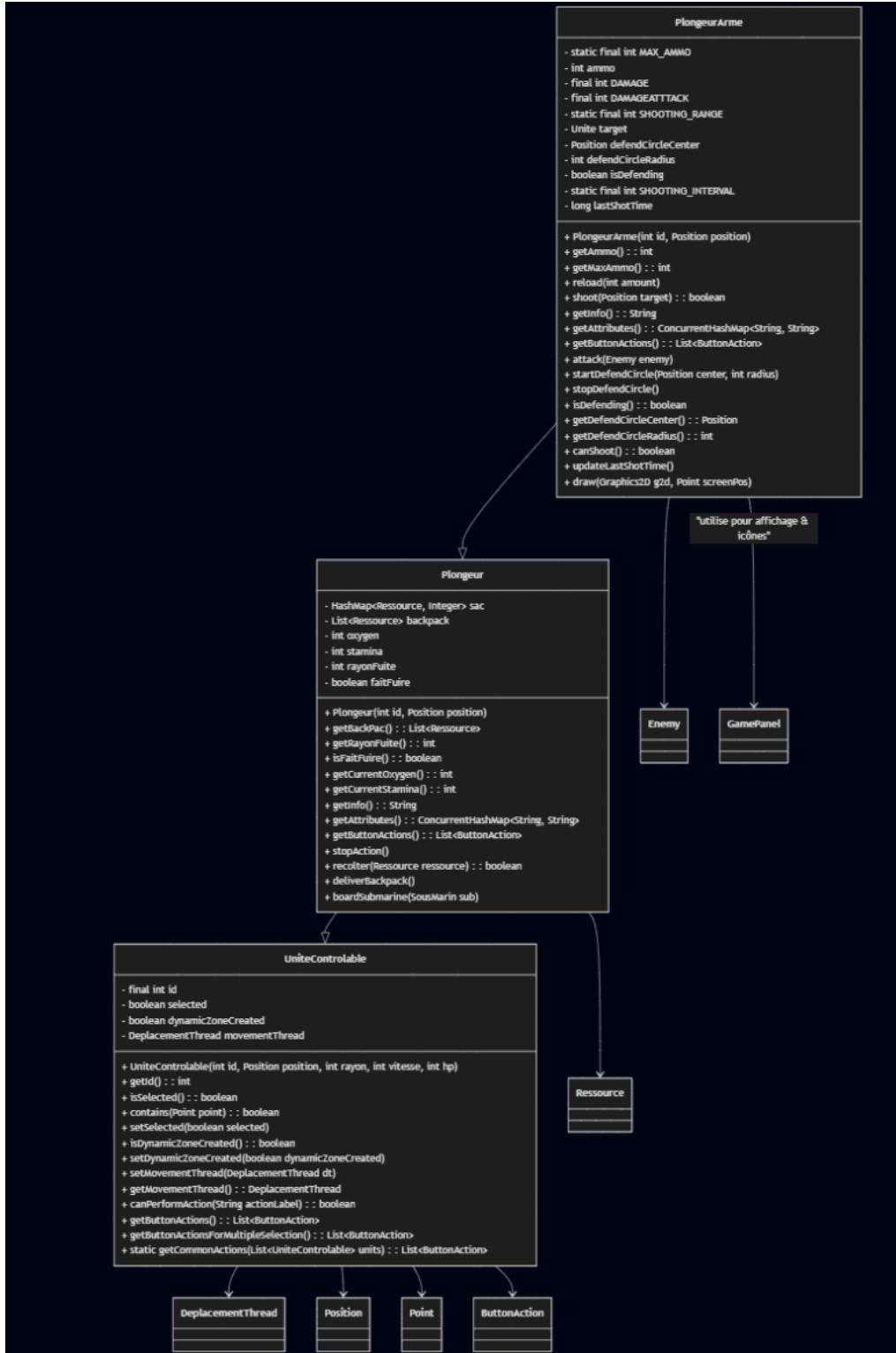
OxygenHandler : Gère la consommation d'oxygène des plongeurs. Réduit l'oxygène en fonction de la profondeur, tue les plongeurs si nécessaire.

StaminaRegenHandler : Gère la régénération ou la diminution de l'endurance des plongeurs. Ajuste l'endurance en fonction des actions des plongeurs.

SelectionClic: Gère les clics de souris pour sélectionner et donner des ordres aux plongeurs. Permet de sélectionner des plongeurs et de leur donner des ordres.

FuiteHandler : Gère l'action fait fuir des plongeurs.

5. Diagramme de classes simplifié



- **Gestion Affichage des Images**

1. Structures de données utilisées

Image image : Stocke l'image principale d'un objet.

Image movingImage : Stocke l'image utilisée pour une unité en mouvement.

Image currentImage : Représente l'image actuellement affichée pour une unité.

ImageIcon unitIcon : Représente une icône associée à une unité.

Map<String, Image> imageCache : Stocke en cache les images chargées pour éviter les rechargements.

Map<Integer, Image> depthBackgroundImages : Associe des images de fond à différents niveaux de profondeur.

Image backgroundImage : Représente l'image de fond d'un panneau.

originalImgWidth, originalImgHeight : Dimensions par défaut des images utilisées pour les objets.

targetDiameter : Définit le diamètre cible pour l'affichage d'une image.

2. Constantes du modèle

CUSTOM_FONT : Police personnalisée utilisée pour dessiner du texte sur les images ou les panneaux.

3. Algorithme (description abstraite)

Chargement des Images : Les images sont chargées via GamePanel.getCachedImage et mises en cache pour éviter les rechargements.

Mise à l'Échelle : Les dimensions des images sont ajustées en fonction de leur rôle et de leur taille cible.

Dessin des Objets : Les objets sont dessinés sur le panneau principal avec des transformations pour les unités en mouvement.

Arrière-Plans : Les images de fond sont associées à des niveaux de profondeur et dessinées en premier.

Panneaux d'Interface : Les panneaux affichent des informations visuelles sur les unités, les ressources, et les ennemis.

4. Utilisation du code par d'autres fonctionnalités :

Objet : Gère l'image principale associée à chaque objet.

Unité : Gère les images statiques et en mouvement des unités.

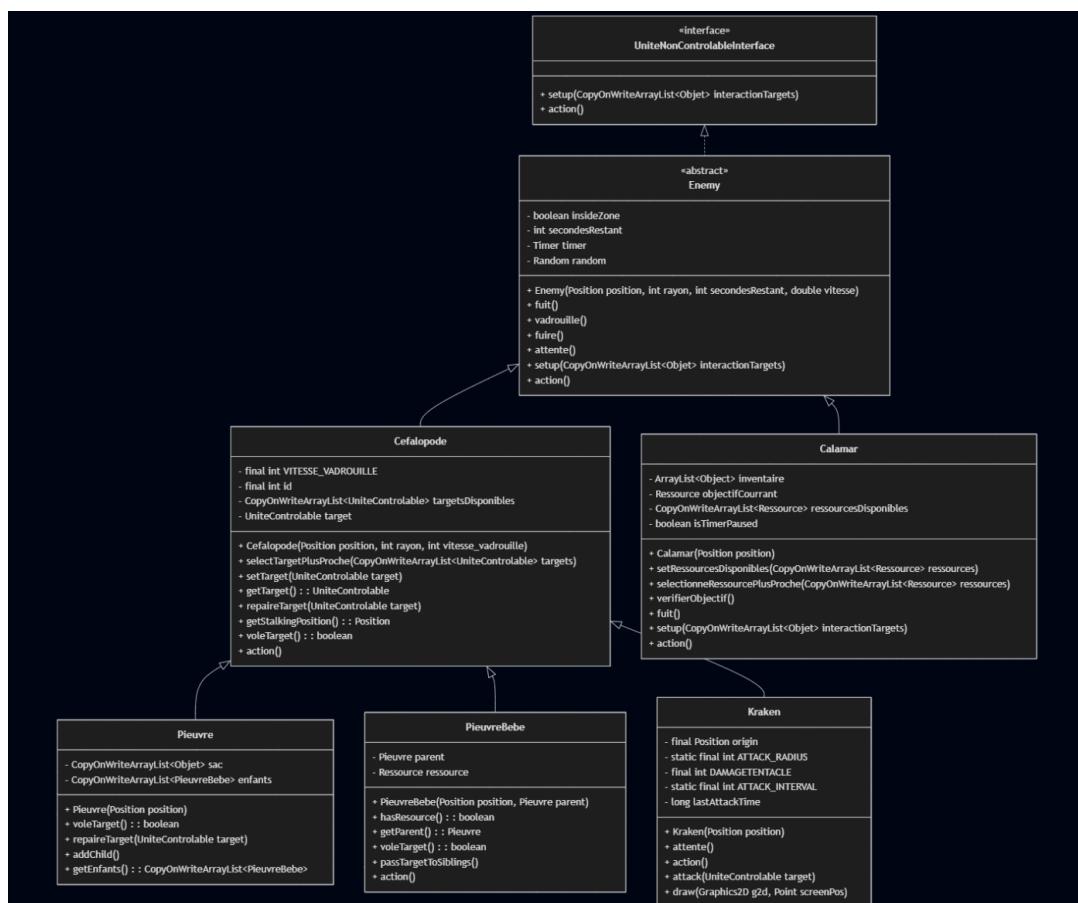
GamePanel : Charge et met en cache les images, dessine les objets et les arrière-plans.

Terrain : Gère les images de fond associées aux niveaux de profondeur. Fournit les images de fond pour dessiner l'arrière-plan.

BackgroundPanel : Gère une image de fond pour les panneaux d'interface utilisateur.

InfoPanel : Utilise des icônes ou des images pour représenter visuellement les unités ou les ressources.

5. Diagramme de classes simplifié



- **Sac (Inventaire) :**

1. **Structure de donné utilisée :**

Classes et Interfaces

- Plongeur : Contient une structure de données (un ArrayList) pour stocker les ressources (Colliers) ramassées.

Collections Utilisées

- ArrayList<Ressource> : Stocke les ressources dans le sac d'un Plongeur.
- HashMap<Ressource, Integer> : Possibilité d'implémenter un inventaire avec quantités si besoin.

2. **Algorithme :**

- **Collecte**

- Lorsqu'un Plongeur collecte une ressource, celle-ci est ajoutée au sac si la capacité n'est pas atteinte.

- **Livraison / Vente**

- Quand le Plongeur atteint la Base, il déclenche l'action de livraison, le contenu du sac est transféré dans le market pour être vendu ou comptabilisé dans les gains du joueur.

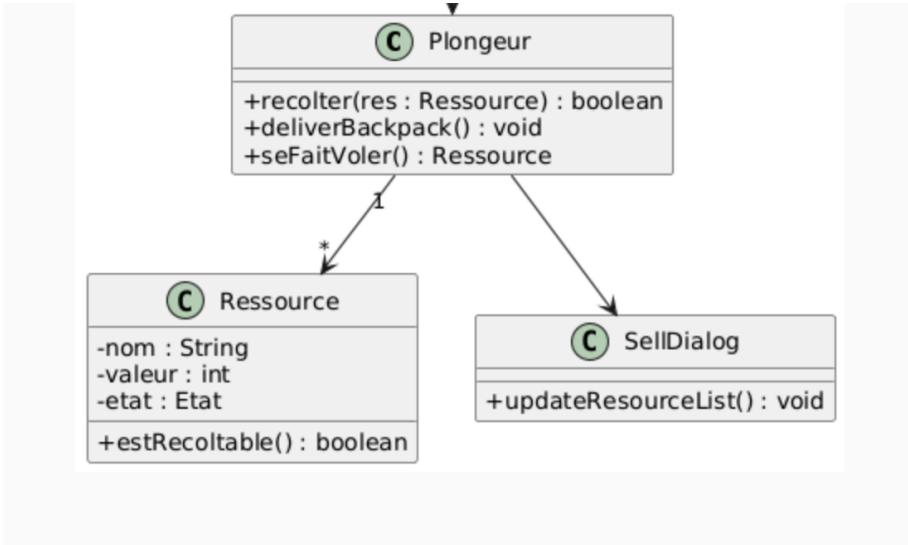
- **Vol**

- Certains ennemis peuvent déclencher une méthode pour retirer une ressource du sac.

3. **Utilisation du code par d'autres fonctionnalités :**

- Market : La vente des ressources du sac est gérée par SellDialog, qui parcourt le sac de chaque unité sélectionnée.
- Interactions ennemis : Les ennemis peuvent accéder aux méthodes du sac pour en voler le contenu.

4. **Diagramme de classe simplifié :**



- **Sous Marin :**

1. **Structure de donnée utilisée :**

SousMarin : Contient une référence (`boardedDiver`) pour stocker le plongeur embarqué ainsi qu'une variable entière (`fuel`) représentant le carburant disponible pour ses déplacements.

Attributs :

`Plongeur boardedDiver` : Stocke une référence au plongeur actuellement embarqué (ou null si aucun plongeur à bord).
`int fuel` : Quantité actuelle d'essence disponible pour les déplacements du sous-marin.
`final int MAX_FUEL` : Quantité maximale de carburant (fixée à 100).

2. **Algorithme :**

- a) **Embarquement :**

Lorsqu'un plongeur entre en collision avec un sous-marin, une vérification est déclenchée. Si aucun autre plongeur n'est déjà à bord, le plongeur devient invisible sur la carte et est référencé par le sous-marin (`boardedDiver`).

- b) **Débarquement :**

Le plongeur débarque automatiquement si le carburant (`fuel`) atteint 0. Le plongeur est rendu visible à proximité immédiate du sous-marin, remis en jeu et disponible pour de nouvelles actions.

3. Utilisation du code par d'autres fonctionnalités :

Déplacement des unités :

La gestion du carburant influence directement la capacité de déplacement du sous-marin.

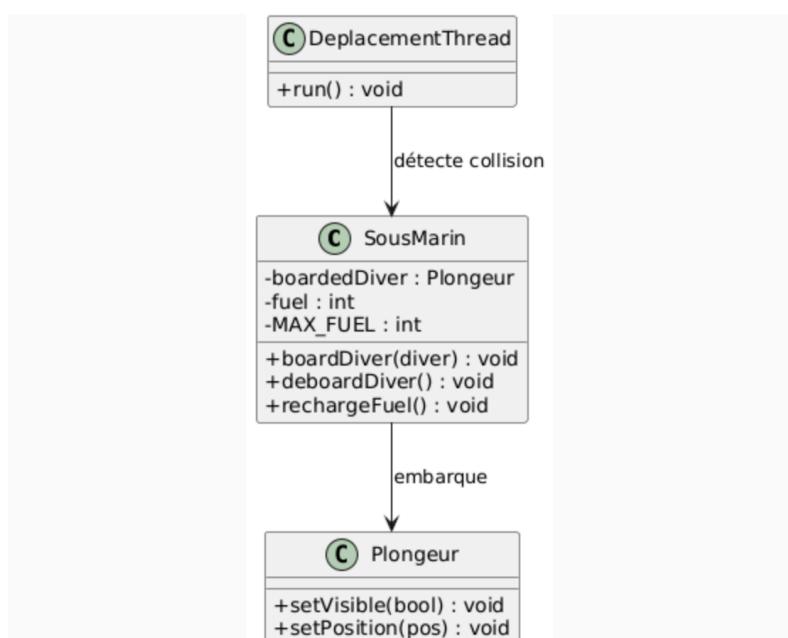
Market :

L'achat d'essence recharge automatiquement le carburant de tous les sous-marins présents en jeu.

Gestion des collisions :

La collision entre un plongeur et un sous-marin déclenche l'action automatique d'embarquement.

4. Diagramme de classe simplifié :



- Système de Victoire et paramétrage :**

1. Objectif :

Permettre au joueur de :

- Définir la durée de la partie (en minutes)
- Choisir la somme d'argent de départ
- Calculer dynamiquement un objectif de score à atteindre
- Gérer automatiquement les conditions de victoire ou de défaite en fonction du temps imparti et des points de victoire

2. Paramétrage du joueur (GameLaunchDialog)

Composants :

- JSpinner timeSpinner → Choix du **temps de jeu** (1 à 60 minutes)
- JSpinner moneySpinner → Choix de l'**argent de départ**
- JLabel pointsLabel → Affichage dynamique du **score cible à atteindre**
- calculateTargetPoints(...) → Méthode de **calcul du score objectif**

Déclenchement de la partie :

Appui sur le bouton « Lancer le jeu » :

- Réinitialise les scores et l'argent via la classe Referee
- Lance un VictoryManager qui surveille le temps et les points
- Affiche GamePanel et ferme le GameLaunchDialog

3. Suivi de la condition de victoire (VictoryManager)

Fonctionnalités :

- **Vérification périodique** :
 - Si le temps est écoulé et que le joueur a atteint les points cibles
→ victoire
 - Si le temps est écoulé sans atteindre le score → défaite
- **Fin de partie** déclenchée automatiquement avec écran GameOverDialog

4. Interactions avec d'autres composants :

Referee : Centralise l'argent et les points du joueur

GameLaunchDialog : Interface de lancement et de configuration de partie

VictoryManager : Gère le timing et les conditions de victoire/défaite

- **Génération et évolution des ressources :**

1. **Objectif**

Le système de génération des ressources (*Resource Spawner*) permet de faire apparaître dynamiquement des artefacts à la surface du terrain sous-marin, en fonction de la profondeur, de la rareté, et du niveau de progression du joueur. Il vise à équilibrer exploration, stratégie et rejouabilité grâce à une répartition intelligente et progressive des objets à collecter.

2. **Structure et classes impliquées**

- **ResourceSpawner** : Thread autonome responsable de la génération procédurale des ressources.
- **GestionRessource** : Thread de gestion du cycle de vie de chaque ressource, de son état initial (**EN_CROISSANCE**) à son état final (**PRET_A_RECORDER**).
- **Ressource** et sous-classes : Collier, Bague, Tresor, Coffre.

3. **Algorithme de génération**

Le **ResourceSpawner** s'exécute tant que le nombre maximal de ressources n'est pas atteint. À chaque itération :

1. Un nombre aléatoire de ressources est défini (entre **spawnCountMin** et **spawnCountMax**).
2. Pour chaque ressource à générer :
 - Une position valide est calculée aléatoirement sur la carte (via **generateValidPosition()**).
 - La profondeur est déterminée à cette position.
 - Un type de ressource est sélectionné selon la profondeur :
 - Profondeur 1 : uniquement **Collier**.
 - Profondeur 2 : **Collier** ou **Bague** (50% chacun).
 - Profondeur 3 : **Bague** ou **Tresor**.

- Profondeur 4 : **Bague** ou **Tresor**, avec 3 % de chances d'obtenir un **Coffre** (victoire instantanée).
3. La ressource est ajoutée au terrain via **gamePanel.addObject(...)**.
 4. Un thread **GestionRessource** est démarré pour faire évoluer la ressource après 20 secondes.

Un délai aléatoire est appliqué entre chaque vague de génération (**spawnIntervalMin** à **spawnIntervalMax** millisecondes).

4. Cycle de vie des ressources

Chaque ressource est initialement créée avec l'état **EN_CROISSANCE**. Elle évolue automatiquement vers l'état **PRET_A_RECOLTER** après un intervalle défini dans **GestionRessource**. Ce changement déclenche une notification à la vue (**InfoPanelUNC**) via un listener.

Etat	Durée	Action possible	Couleur UI
EN_CROISSANCE	20sec	Aucune	Jaune
PRET_A_RECOLTER	∞	Collecte par plongeur	Vert

5. Contraintes de génération

- La position générée doit être :
 - **libre** (pas trop de ressources déjà placées),
 - **dans une zone active** du jeu (**ZoneMover.isInsideAnyZone(...)**).
- Un maximum de **tentatives** (100) est autorisé pour éviter les boucles infinies.

6. Interactions avec les autres composants

- **Plongeurs** : peuvent collecter les ressources une fois prêtées.
- **Base** : réceptionne les artefacts livrés, déclenchant leur mise en vente.

- **Market / Referee** : chaque ressource livrée et vendue génère des points et de l'argent.
- **VictoryManager** : une ressource rare comme le **Coffre** déclenche une victoire immédiate.
- **Carte des Profondeurs et Gestion du Terrain**

1. Objectif

Le système de terrain représente la carte du monde sous-marin. Il gère les zones de profondeurs, détermine où les ressources peuvent apparaître, limite leur densité par région, et permet un rendu visuel immersif adapté à la profondeur. Il joue un rôle central dans l'équilibrage du gameplay, en guidant l'exploration stratégique des joueurs vers des zones de plus en plus riches mais dangereuses.

2. Génération procédurale de la profondeur

Le terrain est une grille 2D (**int[][] depthMap**) dont chaque cellule représente un niveau de profondeur (1 à 4). La génération repose sur deux éléments :

- **Une valeur diagonale** : plus on se déplace vers le bas-droit de la carte, plus la profondeur augmente.
- **Un bruit aléatoire** : introduit une variation pour rendre la transition plus naturelle.

Valeur finale	Profondeur assignée
< 0.25	1 (peu profond)
0.25 à < 0.5	2
0.5 à < 0.75	3
≥ 0.75	4 (très profond)

3. Zones de profondeur et limitation de ressources

Chaque niveau de profondeur est modélisé par une classe interne **DepthZone**, contenant :

- Un identifiant de profondeur
- Un **nombre maximal de ressources simultanées**

- Un compteur en temps réel (**currentResources**)

Cela permet :

- De **contrôler la densité** des ressources,
- D'éviter la **surcharge** graphique ou stratégique d'une zone.

L'ajout ou la suppression de ressources met à jour dynamiquement ces compteurs via :

- **incrementResourcesAt(x, y)**
- **decrementResourcesAt(x, y)**

4. Conversion des coordonnées et gestion d'affichage

La classe **Terrain** propose également des méthodes pour convertir les coordonnées :

- **panelToTerrain(...)** et **terrainToPanel(...)** permettent la navigation fluide avec la caméra.

Un fond visuel spécifique est appliqué selon la profondeur, grâce à :

- un découpage du terrain en cubes (grille secondaire pour l'optimisation du rendu),
- un chargement d'images thématiques (**sea0.png** à **sea3.png**).

5. Intégration avec d'autres systèmes

- **RessourceSpawner** interroge **Terrain** via **canAddResourceAt(...)** pour vérifier s'il est possible de générer une ressource à une position donnée.
- **GamePanel** utilise **getDepthAt(...)** pour adapter les comportements et visuels.
- La **MiniMap** exploite les données de profondeur pour colorer les zones.

- **Barres de progression personnalisées**

1. Objectif

La classe **Barre** fournit une barre de progression graphique personnalisée utilisée dans plusieurs contextes du jeu :

- **Suivi de l'oxygène** des plongeurs,
- **Suivi de la stamina**,
- **Indication du carburant** des sous-marins,
- **Affichage du temps restant** avec format minute:seconde.

Elle permet d'alerter visuellement le joueur lorsque les ressources sont faibles via un **système de seuils colorés dynamiques**.

2. Fonctionnalités

- Personnalisation de la taille (**width, height**) et de la couleur principale.
- **Affichage textuel** de la valeur actuelle.
- Mode spécial **showAsTime** : convertit la valeur en format horloge (mm:ss), utile pour le chronomètre de la partie.
- **Seuil d'alerte (warningThreshold)** :
 - Si le pourcentage passe en-dessous de ce seuil, la couleur passe automatiquement en rouge, signalant un état critique.

3. Comportement dynamique

Méthode **updateProgress(int currentValue, int maxValue)** :

- Met à jour la valeur courante et maximale de la barre.
- Recalcule automatiquement la couleur selon le seuil de danger défini.

Extrait :

```
if (percent < warningThreshold) {  
    setForeground(Color.RED); // état critique  
} else {  
    setForeground(baseColor); // état normal  
}
```

4. Intégration

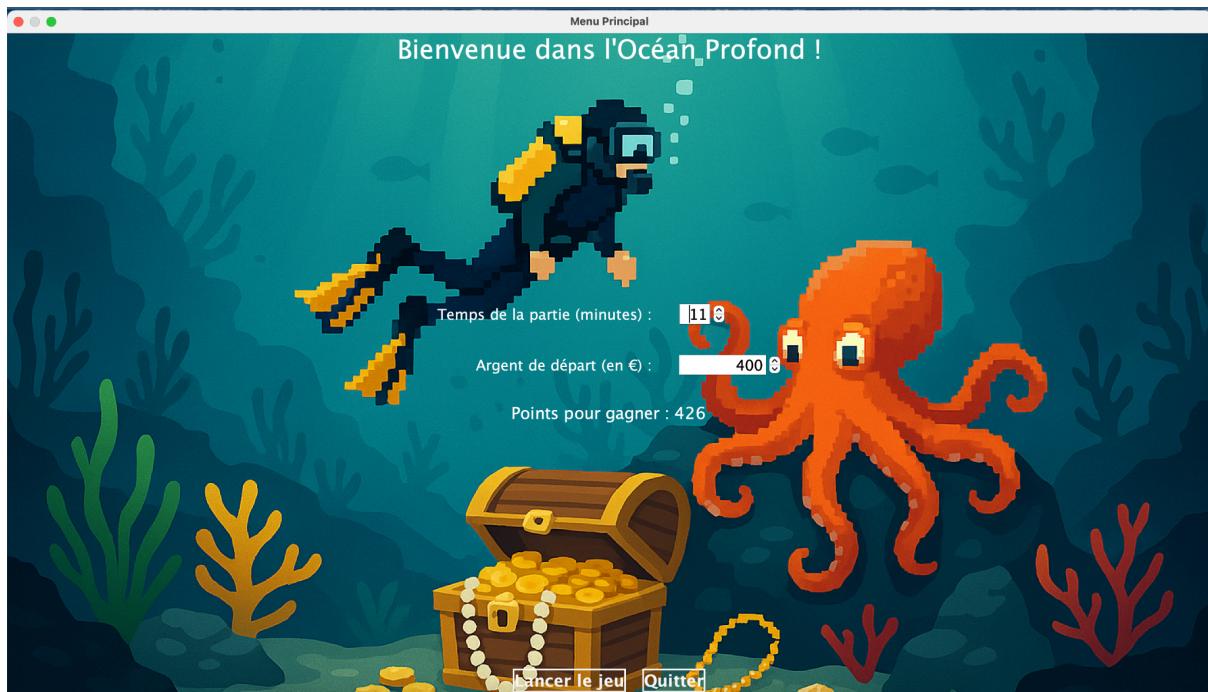
- Utilisée dans les panneaux d'information (**InfoPanel**, c , etc.).
- Synchronisée en temps réel avec les valeurs des unités.
- Pour le **compte à rebours de la partie**, la barre affiche le temps restant et devient rouge en-dessous de 20 %.

5. Bénéfices UX

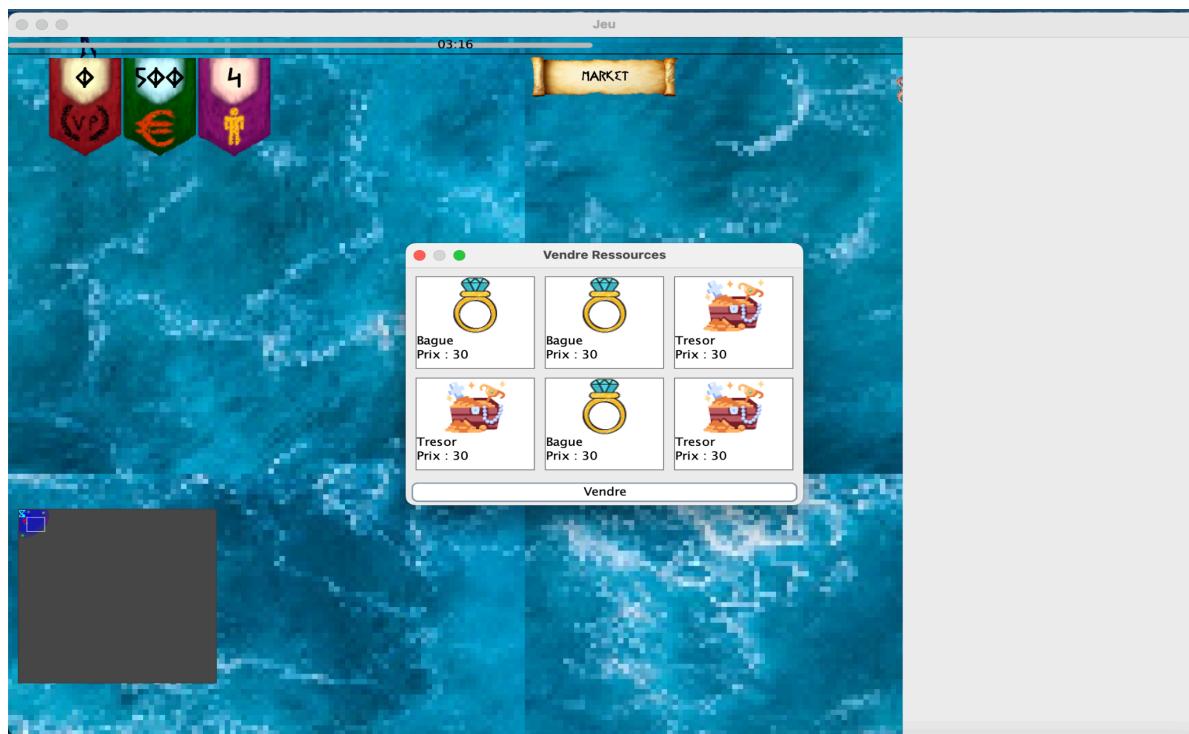
- **Clarté immédiate** pour le joueur sur l'état de ses unités.
- Permet d'anticiper les dangers (manque d'oxygène, essence, fatigue).
- Apporte une **cohérence visuelle** à l'interface du jeu.

V- Résultats :

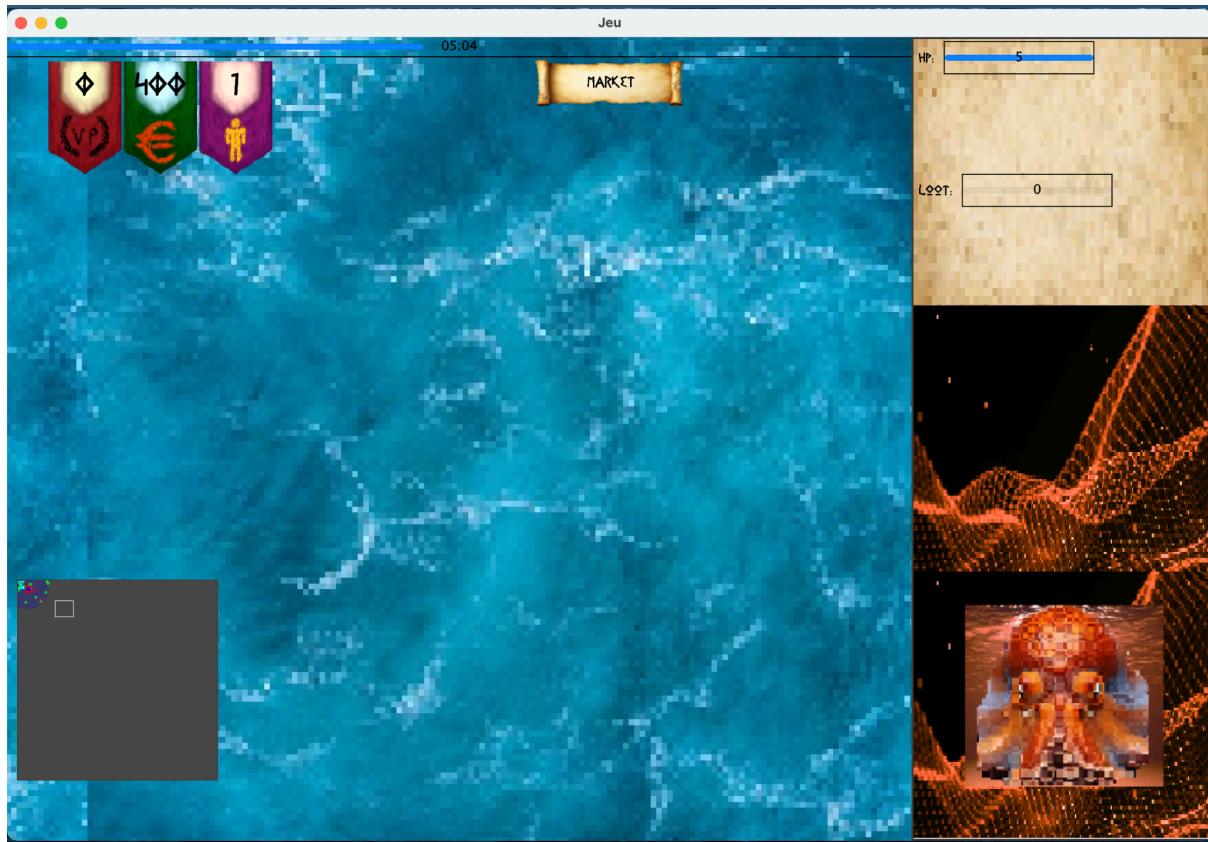
- Menu d'accueil :



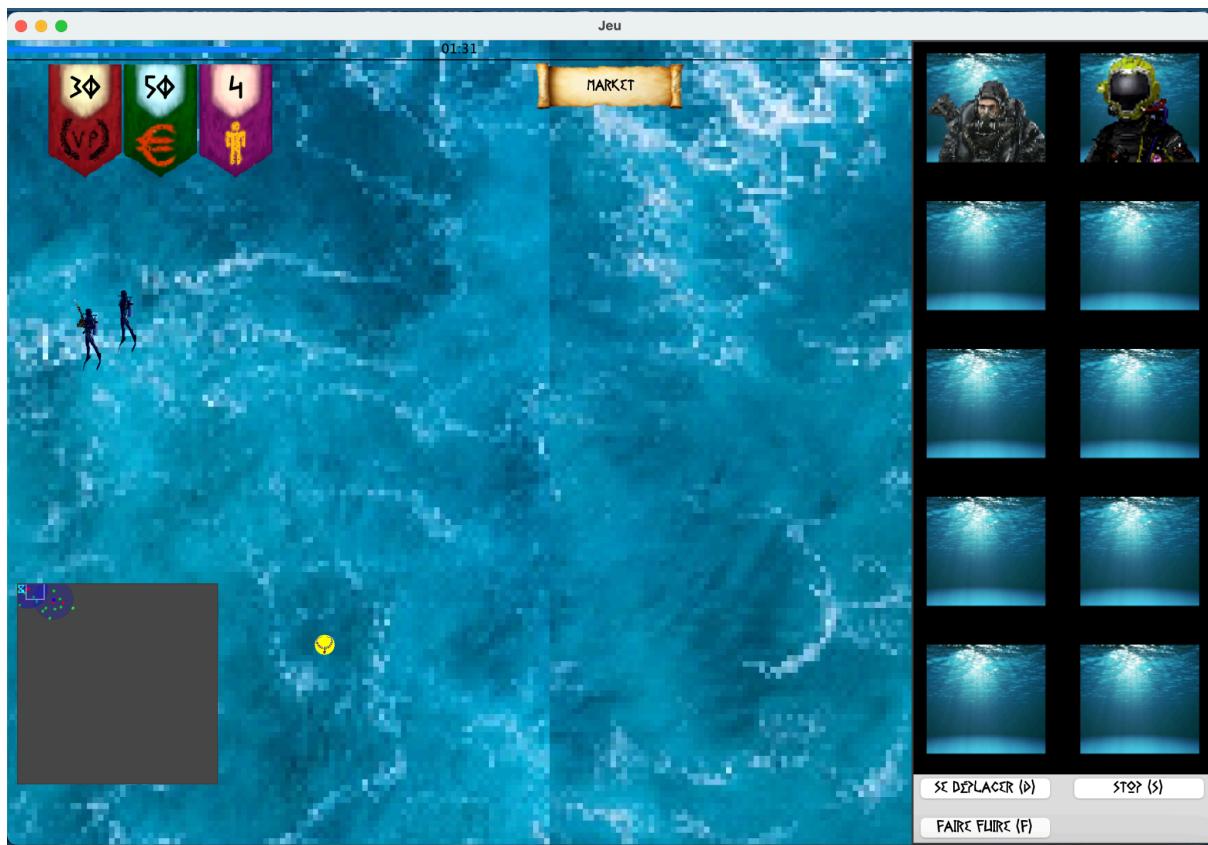
- Market : Vendre



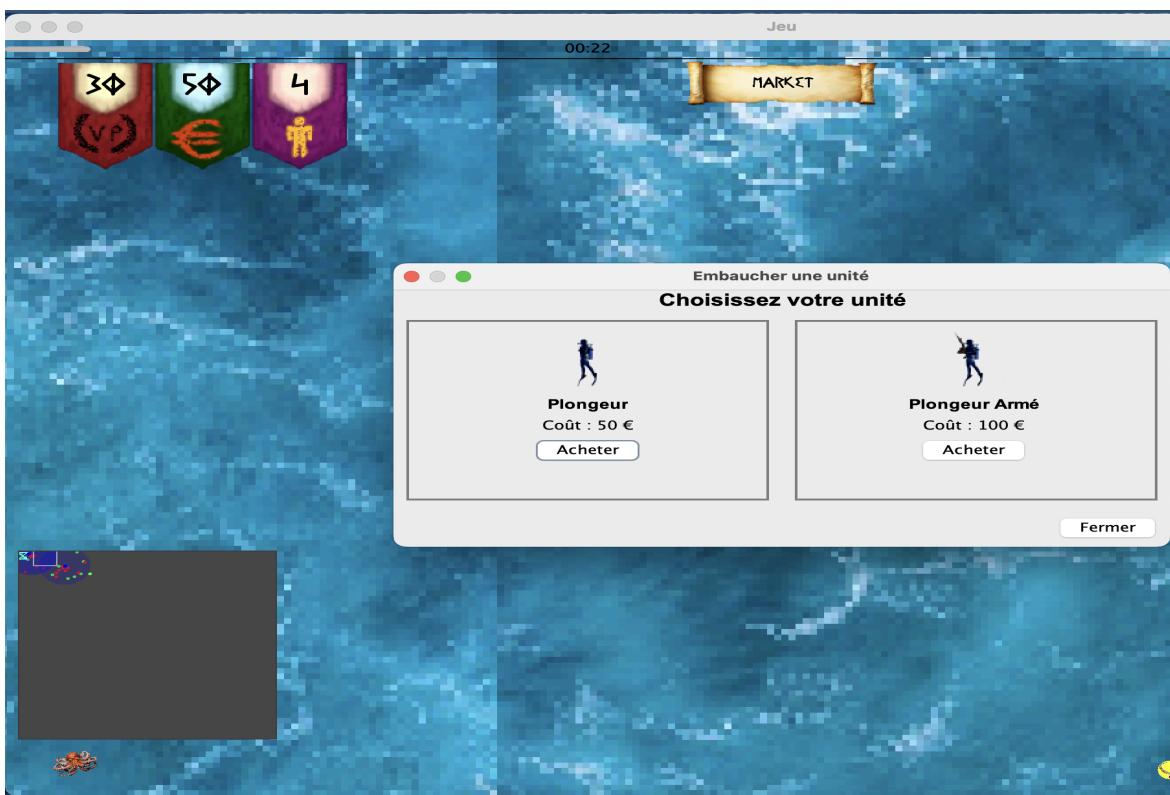
- Panel info ennemi :



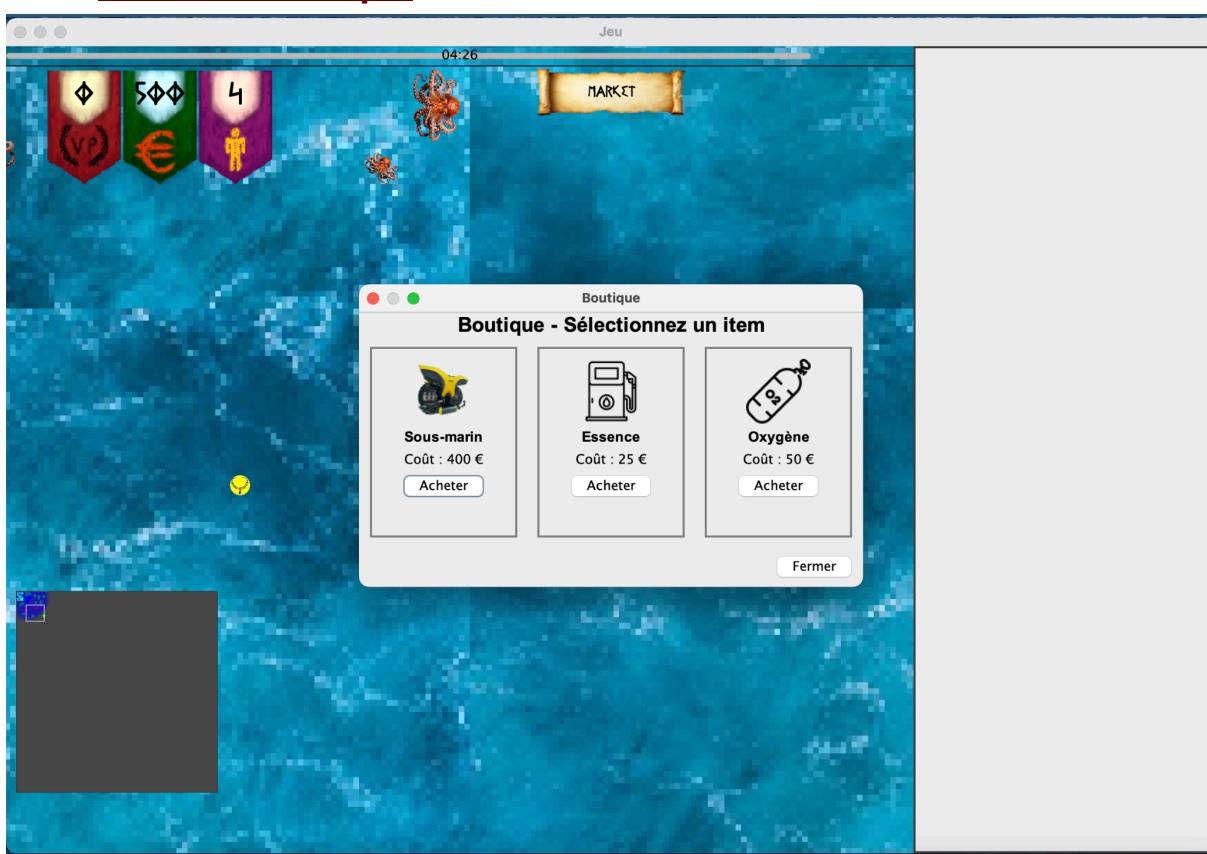
- Panel info sélection multiple :

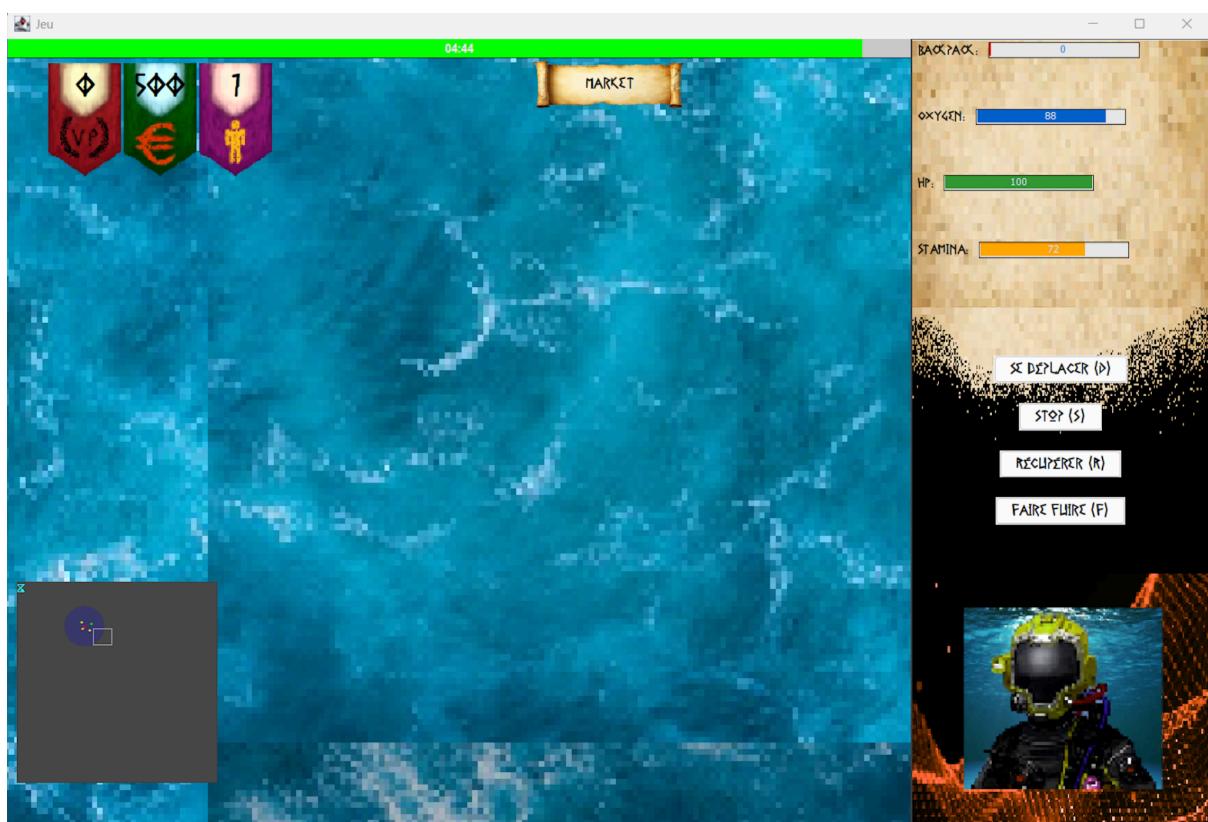
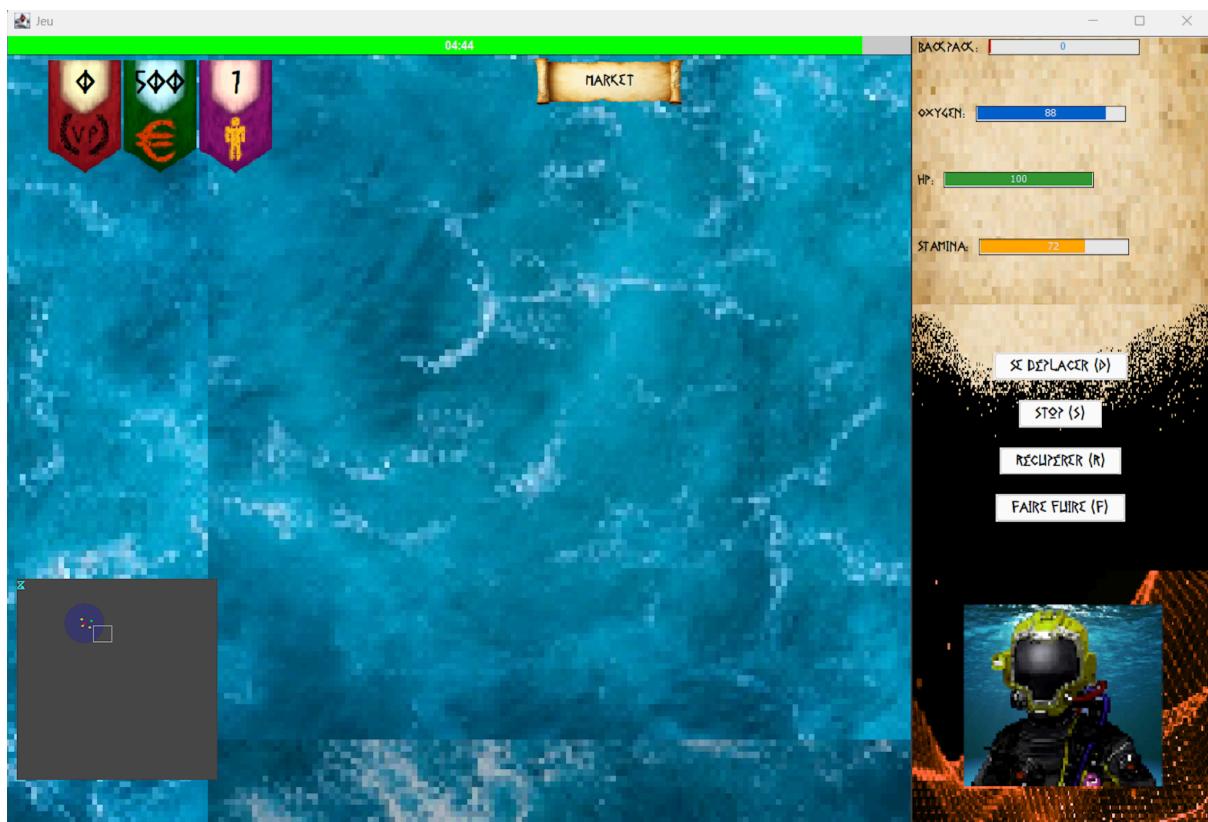


- Market : Embaucher



- Market : Boutique





VI- Documentation Utilisateur :

- Début de Partie:

Au lancement d'une nouvelle partie, vous commencez avec une somme d'argent initiale ainsi qu'une durée limitée définie par vos soins. Un plongeur initial apparaît à une position aléatoire en zone peu profonde (**profondeur 1**). Vous pouvez recruter des plongeurs supplémentaires et des sous-marins via le **Market** afin d'augmenter vos capacités d'exploration.

- Collecte d'artefacts :

Collecte :

Sélectionnez un plongeur sur la carte. Dans le panneau de droite, cliquez sur « Récupérer » puis sur l'artefact souhaité. L'artefact sera automatiquement ajouté dans le sac à dos (**backpack**) du plongeur.

Livraison :

Lorsque le backpack d'un plongeur est plein, ou à tout moment selon votre choix, le plongeur peut retourner vers la base afin de livrer les artefacts. Ceux-ci deviennent alors disponibles dans votre inventaire global pour être vendus au Market.

- Se déplacer :

Sélectionnez l'unité (plongeur ou sous-marin) que vous souhaitez déplacer. Les actions disponibles s'affichent sur le panneau latéral droit. Cliquez ensuite sur « Déplacer » puis indiquez précisément la destination souhaitée sur la carte.

- Market :

Pour accéder au Market, cliquez sur le bouton dédié dans l'interface de jeu. Le Market propose plusieurs actions :

- Embaucher :

Sélectionnez l'option « Embaucher » pour recruter des unités supplémentaires (plongeurs normaux, plongeurs armés). Chaque unité a un coût indiqué clairement, qui sera immédiatement déduit de votre argent disponible. Les unités recrutées apparaissent à des positions aléatoires en profondeur 1. Cela génère également des points de victoire.

- Vendre des artefacts :

Sélectionnez « Vendre » puis choisissez l'artefact que vous souhaitez vendre depuis votre inventaire global. Chaque vente génère des fonds supplémentaires et des points de victoire.

- Acheter de l'essence ou de l'oxygène :

Vous pouvez acheter de l'essence pour recharger le carburant de tous vos sous-marins. L'oxygène permet de recharger les réserves d'oxygène de tous vos plongeurs à 100 %. Chaque achat a un coût affiché clairement et sera immédiatement déduit de vos fonds.

- **Conditions de victoire :**

Votre objectif est d'atteindre le nombre de points de victoire défini en début de partie, avant que le temps limite ne soit écoulé. Les points de victoire s'obtiennent par la vente d'artefacts (colliers, bagues, reliques) et par l'embauche de nouvelles unités. Si l'objectif est atteint avant la fin du temps imparti, vous gagnez la partie. Sinon, c'est perdu !

Si le joueur récupère un **coffre**, événement très rare ayant seulement **3 % de chances** de se produire à la profondeur 4, la partie se termine immédiatement et le joueur remporte automatiquement la victoire

- **Gestion du sous-marin :**

A partir de la profondeur 3 vous pouvez recruter un sous-marin depuis le Market. Celui-ci consomme du carburant (essence) lorsqu'il est en déplacement. Pour embarquer un plongeur dans le sous-marin, il faut déplacer le plongeur vers le sous-marin .

- **Système de Timer de victoire :**

Barre de progression visuelle avec seuil d'avertissement (20% du temps restant)

Calcul en temps réel avec **System.currentTimeMillis()**

Gestion via **TimerTask** avec rafraîchissement toutes les secondes

- **États des Ressources :**

État	Durée	Action Possible	Couleur UI
EN_CROISSANCE	20s	Aucune	Jaune
PRET_A_RECORDER	Illimité	Collecte	Vert

- **Légende des entités dans la MiniMap :**

Entité	Représentation	Couleur	Taille
Plongeur	Cercle	Bleu	Rayon*3
Artefact	Cercle	Jaune/Vert	Rayon*2
Ennemi	Cercle	Rouge	Rayon*3
Base	Rectangle	Cyan	Taille fixe

VII- Documentation développeur :

a) Description des Classes Principales

GameMaster

La classe GameMaster est responsable de la gestion globale du jeu. Elle maintient les listes de ressources et d'ennemis, et met à jour ces listes en fonction des objets présents sur le panneau de jeu (GamePanel). Elle contient également la boucle principale du jeu dans la méthode run.

GamePanel

La classe GamePanel représente le panneau de jeu où les objets sont affichés et les interactions se produisent. Elle gère l'affichage des unités, des ressources, et des points de spawn des ennemis. Elle contient également des méthodes pour ajouter et supprimer des objets du jeu.

SpawnManager

La classe SpawnManager gère les points de spawn des ennemis. Elle génère des points de spawn à des intervalles aléatoires et crée des ennemis à ces points.

b) Constantes Modifiables

GamePanel

PANELDIMENSION: La dimension du panneau de jeu. Modifier cette constante changera la taille de la zone de jeu.

TERRAIN_MIN_X, TERRAIN_MAX_X, TERRAIN_MIN_Y, TERRAIN_MAX_Y:
Les limites du terrain de jeu.Modifier ces constantes changera les limites dans lesquelles les unités peuvent se déplacer.

SpawnManager

getRandomInterval(int min, int max): La méthode pour obtenir un intervalle aléatoire entre les générations de points de spawn. Modifier les paramètres min et max changera la fréquence de génération des points de spawn.

c) Améliorations Possibles :

Interface Utilisateur

L'interface utilisateur pourrait être enrichie avec des informations supplémentaires sur les unités et les ressources. Par exemple, afficher des barres de santé pour les unités ou des indicateurs de croissance pour les ressources.

IA des Ennemis

L'intelligence artificielle des ennemis pourrait être améliorée pour rendre le jeu plus intéressant. Actuellement, les ennemis ont des comportements de base comme la fuite et la vadrouille. Ajouter des comportements plus complexes, comme la coopération entre ennemis ou des stratégies d'attaque, rendrait le jeu plus dynamique.

Sauvegarde et Chargement

Ajouter des fonctionnalités de sauvegarde et de chargement permettrait aux joueurs de sauvegarder leur progression et de reprendre le jeu plus tard. Cela nécessiterait l'implémentation de méthodes pour visualiser l'état du jeu et le restaurer à partir d'un fichier.

Conclusion et perspectives :

Ce projet nous a permis de concevoir et développer un jeu de gestion et d'exploration en environnement sous-marin. L'objectif était de proposer une expérience stratégique mêlant collecte de ressources, gestion d'unités, affrontement avec des ennemis variés et optimisation dans un temps limité. Nous avons réussi à mettre en place une interface graphique fluide, une architecture logicielle modulaire (MVC), ainsi qu'un gameplay complet comprenant des mécaniques de score, de commerce et de victoire/défaite.

Nous avons ainsi implémenté l'ensemble des fonctionnalités essentielles du jeu : une interface utilisateur interactive, une carte 2D dynamique avec gestion des profondeurs, des unités contrôlables et non contrôlables aux comportements variés, un système d'économie complet (marché, achat, vente), un détecteur de collision performant, et une logique de victoire conditionnée par le temps et les points de victoire.

Pendant le développement du projet, nous avons rencontré plusieurs difficultés techniques. L'une des plus importantes a été la gestion des threads, notamment pour que l'affichage reste fluide tout en faisant tourner en parallèle tous les éléments de jeu. Nous avons aussi passé beaucoup de temps sur le débogage, car certaines fonctionnalités ne réagissaient pas comme prévu ou provoquaient des bugs difficiles à détecter. Pour résoudre ces problèmes, nous avons fait de nombreux tests, ajouté des messages de suivi (logs) dans le code, et corrigé petit à petit les erreurs en isolant les éléments concernés. Ces difficultés nous ont appris à être plus rigoureux et organisés dans notre façon de coder.

Ce projet nous a ainsi permis de développer une expertise en programmation orientée objet et en architecture MVC, éléments essentiels pour garantir la lisibilité et l'évolution du code. Nous avons également acquis une solide maîtrise des threads et de la programmation concurrente, en assurant la synchronisation efficace entre les différentes mécaniques du jeu, telles que l'affichage, les interactions entre les différentes unités, ou encore les minuteries de victoire. Par ailleurs, nous avons appris à concevoir un véritable système de jeu stratégique, dans lequel le joueur est constamment amené à faire des choix tactiques : recruter ou économiser, explorer ou se replier, utiliser un sous-marin ou nager, affronter un ennemi ou battre en retraite. Ces mécaniques ont enrichi notre réflexion autour de la conception ludique, de l'équilibrage du gameplay et de la courbe de difficulté. Enfin, ce projet nous a offert une expérience concrète du développement d'un jeu en

temps réel, intégrant interfaces interactives, animations dynamiques et comportements complexes des entités.

De nombreuses perspectives d'amélioration s'ouvrent pour enrichir davantage l'expérience de jeu. Il serait intéressant de renforcer la dimension stratégique en introduisant d'autres technologies à débloquer, ou encore des choix de missions avec objectifs secondaires, permettant au joueur d'adapter ses décisions selon le contexte. L'ajout d'un mode multijoueur, qu'il soit coopératif ou compétitif, offrirait une nouvelle profondeur stratégique et une rejouabilité accrue. Un système de sauvegarde et de chargement permettrait quant à lui de conserver sa progression sur plusieurs sessions de jeu. Enfin, l'introduction d'événements aléatoires – comme des tempêtes, l'apparition de krakens ou la découverte d'épaves rares – encouragerait le joueur à s'adapter en permanence. Pour parfaire l'immersion, des améliorations visuelles et sonores seraient également à envisager, rendant l'univers sous-marin encore plus vivant et engageant.