Analyse du projet

Contexte

Le projet de développement du serveur de jeu consiste à améliorer et à faire évoluer une application existante. Initialement basé sur Java et JBox2D, le serveur de jeu permet à plusieurs joueurs de se connecter et de contrôler des véhicules virtuels.

L'objectif principal est de renforcer la qualité et les fonctionnalités de l'application tout en maintenant la compatibilité avec les clients déjà développés, tels que les applications mobiles et les bots. Pour atteindre cet objectif, plusieurs approches sont envisageables : la ré-écriture du serveur dans un autre langage ou l'amélioration du code existant.

Le choix a été fait de migrer le projet vers le moteur de jeu Godot, offrant ainsi de nouvelles opportunités en termes de flexibilité, de performances et de possibilités de développement. Cette décision découle d'une analyse approfondie des besoins du projet, ainsi que des avantages et des inconvénients de chaque technologie disponible.

Dans cette optique, le présent document d'analyse vise à fournir une évaluation détaillée de l'existant, des propositions d'amélioration, des choix techniques argumentés et une planification pour la mise en œuvre de ces propositions. Il servira de guide pour le développement du projet vers un état plus robuste et plus fonctionnel.

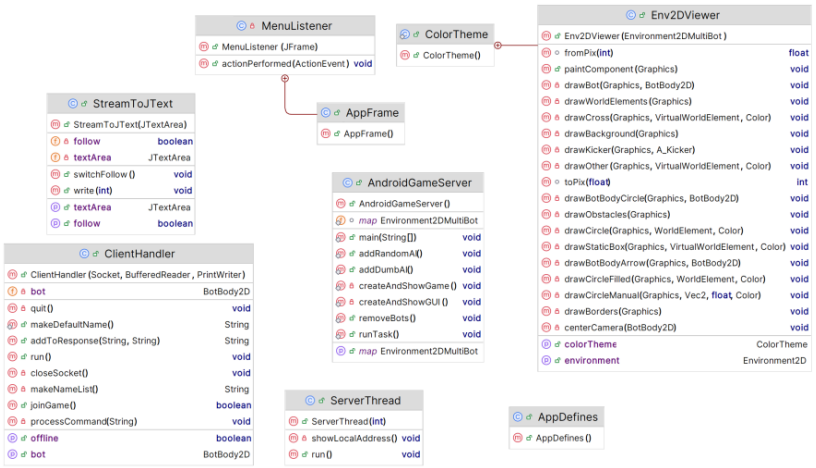
Projet existant

Présentation de l'existant en termes de structure générale du code

Le serveur de jeu est bien structuré, avec une organisation claire des packages et des classes. La structure générale du code est organisée en plusieurs packages distincts, chacun ayant une responsabilité spécifique. Par exemple, les classes liées aux bots sont regroupées dans le package `bot`, tandis que les éléments de l'environnement se trouvent dans le package `worldElements`. Cela montre une bonne séparation des préoccupations, ce qui facilite la navigation et la compréhension du code.

Le cœur du serveur est représenté par la classe `AndroidGameServer`, qui initialise le serveur et gère les connexions des clients via le `ServerThread`. Ce dernier maintient une liste de clients et distribue les tâches à `ClientHandler`, qui gère la communication avec chaque client connecté. Les interactions critiques, telles que la gestion des connexions des clients, impliquent plusieurs composants du système. Par exemple, lorsqu'un client se connecte, le `ClientHandler` est créé pour gérer cette connexion, et le `ServerThread` ajoute ce client à sa liste, permettant ainsi une communication bidirectionnelle entre le client et le serveur.

Le code utilise des interfaces et des classes abstraites pour définir les comportements des différents éléments du jeu, ce qui favorise l'extensibilité. Par exemple, les éléments du bot et les éléments physiques de l'environnement héritent de classes de base telles que `BotElement` et `PhysicalElement2D`, ce qui permet d'ajouter facilement de nouveaux types d'éléments sans modifier le code existant.



Analyse de l'existant en termes de qualité

La qualité du code peut être évaluée sous trois aspects principaux : la lisibilité, la maintenabilité et la robustesse.

Lisibilité

En termes de lisibilité, le code présente plusieurs points forts. L'organisation en packages clairement définis et l'utilisation de noms explicites pour les classes et les méthodes rendent le code plus compréhensible. Cependant, un manque de documentation complète et de commentaires explicatifs est notable. Les commentaires sont essentiels pour aider les développeurs à comprendre le flux du programme et les parties complexes du code. De plus, certaines méthodes sont assez longues et gagneraient à être divisées en sous-méthodes plus petites et spécifiques, ce qui améliorerait encore la lisibilité.

Maintenabilité

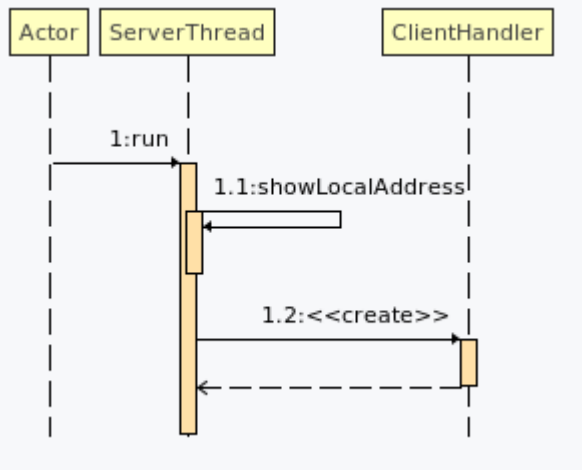
La maintenabilité du code est facilitée par sa modularité. La séparation des responsabilités entre les classes permet une maintenance plus aisée et l'ajout de nouvelles fonctionnalités sans perturber le reste du système. L'utilisation d'interfaces et de classes abstraites montre une bonne compréhension des principes de la programmation orientée objet et favorise l'extensibilité. Cependant, l'absence de tests unitaires est un point faible majeur. Les tests unitaires sont cruciaux pour valider les modifications et détecter les régressions. Une meilleure gestion des exceptions est également nécessaire pour prévenir les plantages imprévus et améliorer la résilience du système.

Robustesse

En termes de robustesse, le code bénéficie d'une séparation claire des responsabilités, ce qui améliore la robustesse globale du système. Cependant, la gestion des erreurs pourrait être améliorée pour garantir une meilleure résilience du serveur en cas de situations imprévues. De plus, certaines méthodes critiques du système pourraient être optimisées pour améliorer les performances du serveur. Utiliser des outils de profilage pour identifier et corriger les goulots d'étranglement potentiels serait bénéfique.

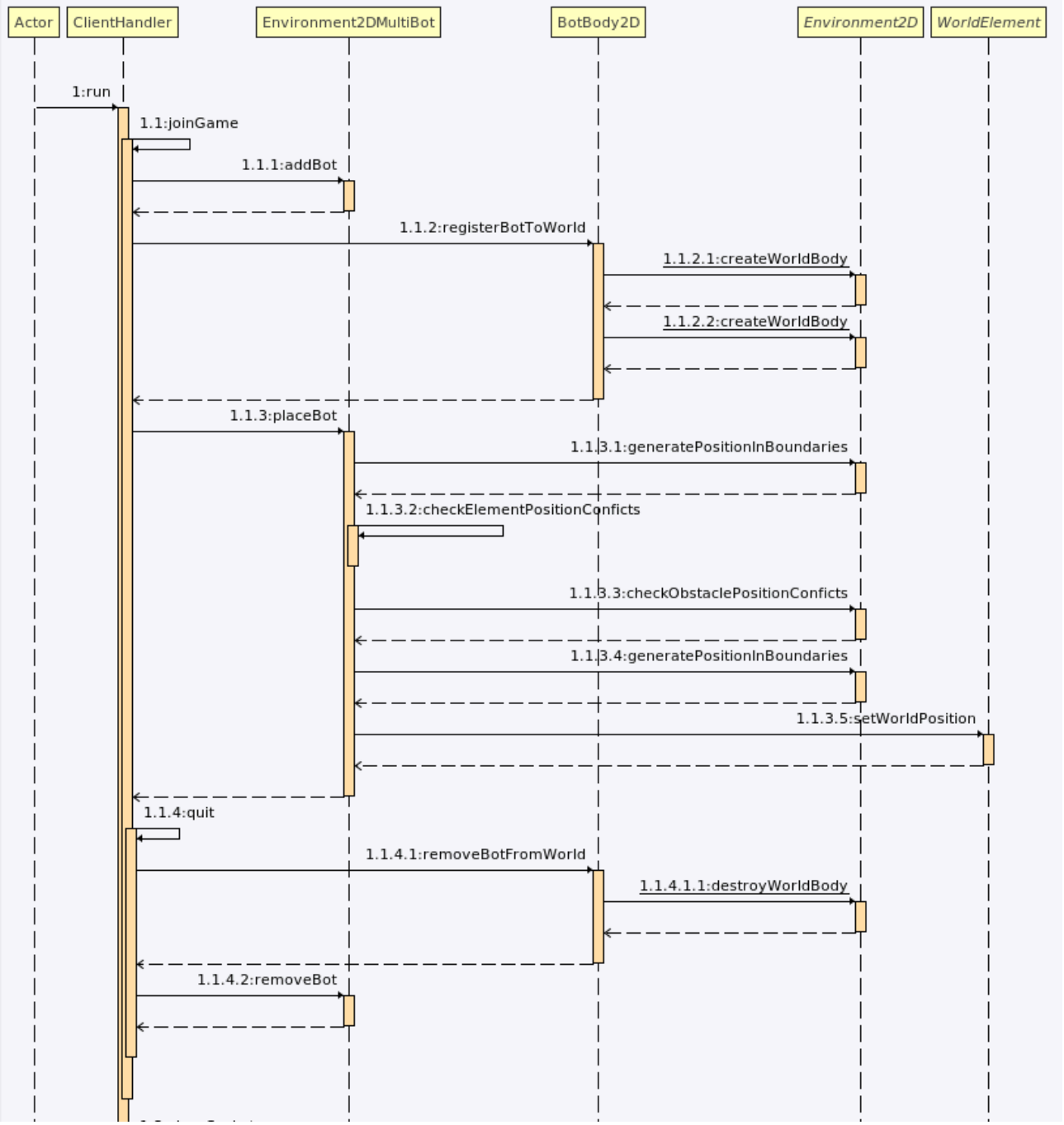
Analyse du projet

Projet existant



Analyse du projet

Projet existant



Analyse du projet

Projet existant

Fonctionnalité

Commandes	Paramètre	Effet	Retour
NAME	String	Change le nom du joueur	N/A
COL	int	Change la couleur du joueur	N/A
COL	int, int, int	Change la couleur du joueur	N/A
EXIT	N/A	Déconnecte le joueur	N/A
LIVE	N/A	Indique que nous somme toujours en ligne	N/A
MSG	N/A	Ajout un message court sous le joueur	N/A
NLIST	N/A	Renvoie la liste des joueurs	Array
USRMSG	String	Renvoie le message du joueur	String
Orient	N/A	Renvoie l'orientation absolue du joueur 2PI	Float
CBOT	N/A	Retourne des informations sur l'adversaire le plus proche orientation/distance	String
NBOT	String	Envoie des info sur l'adversaire choisi	String
CPROJ	Float	Renvoie le projectile le plus proche orientation/distance	String
MOTL	Float	Puissance du moteur gauche (0,5 stop)	N/A
MOTR	Float	Puissance du moteur droite (0,5 stop)	N/A
GUNTRIG	Float	Tire > 0,5. Arrête de tirer < 0,5	N/A
GUNTRAV	Float	Défini l'orientation du canon en fonction de l'orientation du joueur (0,5 devant)	N/A

Choix techniques

L'élaboration de notre solution nécessite une évaluation approfondie des choix technologiques afin de garantir la pertinence, la robustesse et la maintenabilité de notre architecture. Le programme actuel est tel un serveur de jeu avec affichage graphique. Notre analyse s'est donc concentrée sur plusieurs moteurs de jeu et langages de programmation, en mettant en balance leurs avantages et leurs inconvénients respectifs.

Godot

Godot émerge comme une solution attrayante en tant qu'outil open source et communautaire offrant une gamme étendue de fonctionnalités pour le développement de jeux en 2D, 3D et réalité virtuelle. Son support pour plusieurs langages de script, notamment Python, C++, et GDScript, lui confère une flexibilité appréciable pour répondre à divers besoins de développement.

Technologie	Confiance	Qualité	Fonctionnalité native	Optimisation	Total
COEFFICIENT	5	5	3	1	N/A
Python	3	4	2	2	43
C++	3	5	3	4	53
GDScript	5	4	5	3	63

Unity

Unity, bien que populaire et largement utilisé, a été écarté de nos options pour plusieurs raisons. Tout d'abord, ses performances ne répondent pas à nos critères, car notre projet n'exige pas une optimisation particulièrement poussée. Ensuite, son historique de dette technique, incluant des problèmes de stabilité et de maintenabilité, nous a incités à rechercher des alternatives plus fiables et pérennes pour notre solution.

Unreal

Unreal Engine est reconnu pour sa puissance, en particulier dans le domaine de la 3D où il est particulièrement utilisé quand il s'agit d'avoir des graphismes poussés. Cependant, son utilisation peut être entravée par une dette technique substantielle, ce qui peut compromettre la maintenabilité et l'évolutivité de notre projet sur le long terme.

Java

Bien qu'une partie du code existant soit en Java, son manque de qualité et l'absence de tests remettent en question son adéquation pour notre projet. Intégrer du nouveau développement dans cet écosystème pourrait être complexe et risqué.

Rust

Rust présente des avantages significatifs en termes de qualité de code et de sécurité. Sa conception moderne et sa forte communauté en font une option attrayante. Cependant, en tant que langage relativement récent, des préoccupations persistent quant à sa stabilité et à sa maturité pour les projets de grande envergure.

Python

Python est réputé pour sa simplicité et sa richesse en bibliothèques, ce qui en fait un choix séduisant pour le développement de jeux vidéos. Les bibliothèque PyGame, Pyglet et PyOpenGL sont capable de soutenir le développement de jeux vidéo en 2D. Cependant, la performance de Python et plutôt limitée dans des contextes de jeux vidéos complexes qui nécessite des calculs intensifs.

Analyse du projet

Choix techniques

Technologie	Confiance	Qualité	Fonctionnalité native	Optimisation	Total
COEFFICIENT	5	5	3	1	N/A
Godot	5	4	5	4	64
Unity	1	4	5	5	45
Unreal	3	4	4	4	51
Java	3	2	2	2	33
Rust	3	5	2	4	50
Python	3	4	3	2	46

Plan de développement

Nous avons découpé le développement de l'application en 7 grandes étapes. Chaque étape de développement se focalise sur une partie du projet ou une fonctionnalité. Nous avons ordonné les différentes fonctionnalités dans l'ordre du plus important au moins important.

Sprint	Date de début de sprint	Date de fin de sprint	Objectif
Analyse et Planification	02/05/2024	03/05/2024	Analyser le projet existant et faire les choix techniques
Ré-écriture	06/05/2024	01/06/2024	Ré-écrire tout le serveur de jeu sans ajouter de fonctionnalité majeur
Nouvelle fonctionnalité	03/06/2024	07/06/2024	Ajout d'un mode de jeu, Ajout de classe etc...
Refacto et Qualité	10/06/2024	14/06/2024	Vérification de la qualité de l'intégralité du code. Mise au propre des éventuelles coquilles.
Soutenance et Manuel Technique	17/06/2024	21/06/2024	Préparer la soutenance et Finaliser les différents rendu