

UNIVERSITÉ DE STRASBOURG

CURSUS MASTER INGÉNIERIE – INFORMATIQUE

L3S6

rapport projet intégrateur

Auteur :
Timothée OLIGER
Adrien OSSYWA

Responsable :
Stephane CATELOIN

SUJET
Tiny Empire



May 2, 2019

Contents

Introduction	2
Contexte	2
Choix du jeu	2
Conception du jeu	2
Technologies	3
Langage de programmation	3
Architecture	3
Micro-services	4
API authentification	4
API de classement	4
API de gestion des parties	5
Instance de partie	5
Client	5
Organisation	5
Formations des équipes	6
Front	6
Noyaux	6
Réseaux	6
GRPC	6
Répartition des tâches	7
Gitlab	7
Discussions	7
Discord	7
CI / CD	7
gitlab CI	7
Webhook gitlab	8
miroir github / docker cloud	8
Developement	8
S'adapter aux configs	8
docker	8
git	8
branches	8
Deploiement	9
Kubernetes	9
HA	9
Montée en charge	9
Test	9
Go test	9

Data race	10
Partie Personel	10
Timothée Oliger	10

Introduction

Ce projet nous permet d'appliquer les compétences que nous avons acquises durant notre licence, dans les différents domaines étudiés, notamment l'image et le réseau en CMI. Il s'agit de créer un jeu multijoueur en temps réel, reprenant un jeux des années 1980-1990.

Contexte

Les CMI en option Imagerie ont été amenés à utiliser un moteur de jeu 3D, ici Unity, lors d'un projet au semestre 5. Cette matière a permis d'approcher l'utilisation de scripts et la manipulation de l'interface de Unity. Ces connaissances préalables ont permis d'aborder plus rapidement la conception du jeu, ainsi de connaître la solutions à certains problèmes auxquels ils auraient pu faire face auparavant. En choisissant d'utiliser Unity plutôt qu'un moteur de jeu inconnu des CMI Image du groupe, il a été possible d'avancer plus vite dans l'ensemble de la conception du jeu.

Choix du jeu

Un premier brainstorming a été réalisé lors des réunions de début, afin de trouver le jeu le plus adapté aux demandes et contraintes du sujet. Les avis se sont majoritairement dirigés vers un Real Time Strategy (RTS) game, soit un jeu de stratégie en temps réel. Après reflexion, nous avons fini par choisir Age Of Empire, car cela motivait tous les membres de l'équipe, et qu'une version allégée nous parassait accessible. Cependant nous avons sous-estimé la charge de travail que représente un RTS tel que Age Of Empire. De ce fait nous avons du limiter énormément les fonctionnalités et nous avons fait impasse sur la version mobile et Web.

Conception du jeu

Plusieurs réunions de groupes ont été organisées surtout au début du semestre, afin de se décider sur les technologies à utiliser et l'architecture du projet à venir.

Nous avons réalisé des schémas pour prévoir au mieux une architecture robuste et cohérente avec le jeu Age Of Empire original.

Technologies

Un des premiers problèmes rencontrés a été l'hétérogénéité entre les compétences techniques des différents membres du groupe. Nous avons du faire des compromis pour que tous le monde puisse progresser dans de bonne conditions, et que les personnes avançant plus rapidement puisse opérer sur un nombre plus important de tâches.

Langage de programmation

Pour l'instance de jeu, nous avons hésité entre JAVA, Python et GO. JAVA étant lourd et en voie d'extinction, ilPython et GO était les deux choix restants. Après discussion GO nous a paru plus adapté, dans le sens où il ressemble fortement au C et possède les avantages: * d'être compilable * d'intégrer des outils de test * d'intégrer une gestion des dépendances * d'intégrer une documentation * de très bien s'intégrer à une architecture micro-service

Architecture

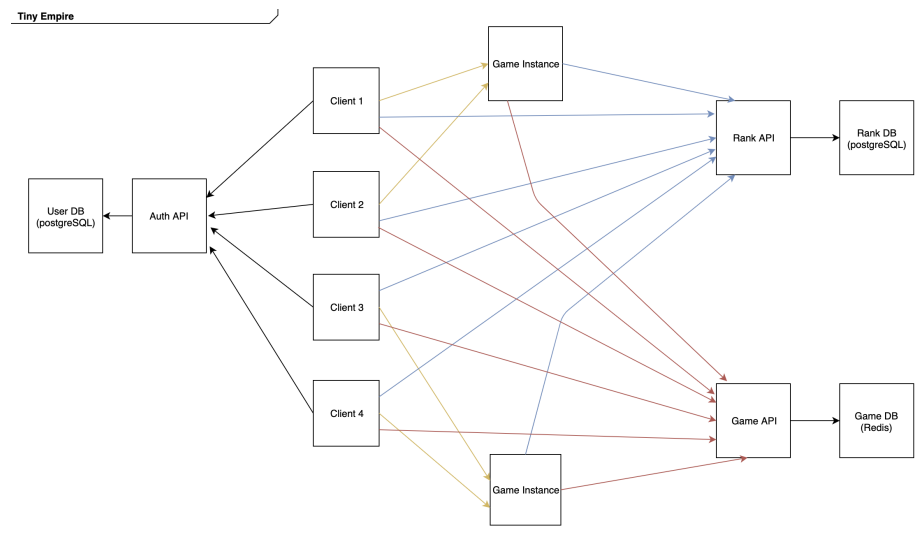


Figure 1: Architecture

En discutant entre membres de l'équipe, nous nous sommes tournés vers une architecture de type “micro-services” : c’est celle qui nous a semblée la plus propie à l’application des connaissances respectives de chacun.

Micro-services

Les avantages d’une architecture de type “micro-services” sont : *

- * l’indépendance entre les services
- * la quantité de code raisonnable et facilement assimilable contenue dans chaque service
- * la possibilité d’augmenter la robustesse de l’application en dupliquant uniquement les services les plus sollicités.

API authentication

Nous avons décidé de choisir la réalisation d’une API GraphQL afin de se connecter/enregistrer et obtenir la liste des serveurs disponibles.

GraphQL GraphQL est un langage créé par Facebook. Même si REST (Representational State Transfer) demeure le format standard des API, certains développeurs décident de se tourner vers GraphQL pour combler les lacunes majeures de ce format. Contrairement à REST, et son modèle relativement structuré basé sur les ressources, GraphQL intervient avec une approche plus flexible : on crée un schéma de requête, puis le serveur l’analyse et renvoie les informations demandées. De plus, l’utilisation du projet “Apollo Server” permet de monter une API GraphQL très simplement et rapidement.

Knex / bookshelf Nous avons choisi d’utiliser un Object-Relational Mapping (ORM) pour la base de données contenant les informations sur les parties et les joueurs. Ce type de programme se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet. De manière imagée, un ORM peut être décrit comme une couche d’abstraction entre le “monde objet” et “monde relationnel”.

API de classement

Ce Service permet de calculer le niveau de chaque joueur, il exploite pour cela les résultats des parties.

Ligues Le système de classement classe les joueurs dans 4 ligues différentes. Le joueur commence dans la ligue la plus basse, et lorsqu’il gagne des parties, il monte doucement le classement jusqu’à terminer dans la ligue qu’il lui correspond

Avantages L’avantage de ce système de classement et que le niveau de chaque partie correspond au niveau des joueurs. Cela permet d’améliorer l’expérience de jeu grâce à un système équilibré.

Avancement A l’heure actuelle, nous l’avons pas pu l’implémenter car c’est un module non nécessaire au déroulement de la partie qui est la priorité.

API de gestion des parties

Ce service permet de rejoindre des partie, obtenir des informations sur les parties comme la composition des équipes.

Instance de partie

Les instances de partie sont écrites en GO et conteneuriser comme les autres services grâce à docker. Cela permettra à terme de générer une partie à la demande et de la supprimer facilement.

Client

Concernant le client, nous hésitions entre Godot et unity

Unity Unity est un moteur de jeux propriétaire possédant un nombre conséquent d’assets et de fonctionnalités puissantes permettant de construire un jeu avec des outils performants. L’avantage de Unity comparé à Godot, est que nous avons appris à le manipuler lors de l’UE “Moteur de Jeux 3D” ce qui a été décisif sur notre choix en plus du grand nombre d’utilisateurs. A la vue de l’ampleur de notre projet, nous ne pouvions pas nous permettre de perdre un temps considérable sur la prise en main de Godot.

Godot Godot est un moteur de jeux open source, il est léger et facilement intégrable avec git. Cependant Godot souffre d’un manque cruel de communauté comparé à Unity et la documentation présentait des lacunes. Pour départager un vote a été réalisé et notre choix s’est dirigé vers unity.

Organisation

Pour créer le jeu dans les meilleures conditions, il a fallu diviser le projet en plusieurs modules

Formations des équipes

La formation des équipes est venue naturellement en fonction des différents CMI ' Image / Réseau '.

Front

L'équipe front qui est composé des CMI Image est chargé de développer le client car ces derniers ont appris à utiliser Unity et son plus à l'aise en UI / Animations ... Elle est composée d'Adrien, Chloé et Marie.

Noyaux

L'équipe noyaux est chargé de développer le noyaux. E

Réseaux

L'équipe réseaux est chargé de faire le lien entre le client et l'instance de la partie. Le framework utilisé pour établir cette communication est "GRPC" pour C# (unity)et GO.

GRPC

Cette librairie permet d'établir une communication stable reposant sur les standards HTTP/2. Cette technologie a été retenu car elle peut être utilisée pour plusieurs langage de développement et est facilement intégrable au code source. Plusieurs services peuvent être implémentés et générer dans le langage cible voulu à partir d'un fichier .proto. Ces services doivent être décrites dans ce fichier pour être ensuite interprétés par le compilateur protoc. Un code va être généré à l'aide de ce fichier contenant l'équivalence des services spécifiées écrits en langage cible. Ainsi, les fonctions/classes créés permettent de récupérer/envoyer des informations de manière transparente lors du développement du jeu. Le fichier .proto doit être le même pour code grpc go et grpc C#. Ainsi les services doivent être formalisés de manière générique pour les 2 langages. Les modifications de ce fichier implique qu'une nouvelle compilation des fichiers GRPC dans tous les langages cibles du projet doit être réalisé.

Dans le dossier utilisé pour Unity, il faut insérer dans le dossier "Assets/Plugin" les éléments du Framework GRPC qui va permettre d'interpréter les services écrits langage cible issus du fichier .proto

Répartition des tâches

La répartition des tâches se fait grâce à gitlab, les issues sont créés et attribué ou choisi par les personnes disponibles. Ce système permet de d'attribuer des tâches correspondantes aux personnes augmentant ainsi la productivité.

Gitlab

L'utilisation de gitlab est très agréable, contrairement à github, la version gratuite contient un large pannel de fonctionnalités.

Issues Les issues nous permettent de demander des features et de déclarer qu'on travail sur une feature. Le gros problèmes que nous avons rencontré par rapport aux issues est qu'au début du projet certaines issues se faisaient sur le long termes ou alors d'autres issues imprévues auxquelles on avait pas pensé sont apparues au fil de l'avancement du projet.

Pull request Lorsqu'une branche est stable, c'est à dire qu'elle passe les tests de l'intégration, une Pull request est créé. Elle est validée après un test effectué par un membre et après avoir passé tout les test de l'intégration continu.

Discussions

Les issues sont fort pratique, hélas pour des petites question technique il est utile d'utiliser de la communication sous forme de chat qui n'est pas le cas du système de discussion directement intégré aux issues.

Discord

Nous avons choisi discord comme logiciel de chat. En plus d'être gratuit il permet de faire des groupes d'utilisateurs et différents salons textuel et vocaux.

CI / CD

Pour pouvoir garantir un maximum de stabilité, à chaque publication de commit, des scripts de test sont lancé sur des runner gitlab.

gitlab CI

Gitlab permet grace au fichier .gitlab-ci.yml de déclarer un pipeline qui peut tester, publier et deployer des solutions logiciels

Webhook gitlab

Lorsqu'il y a un événement sur un des projet git du groupe gitlab AOEINT, un message apparait dans le salon CI du discord, cela permet de prendre connaissance d'un commit ou d'une issue.

mirroir github / docker cloud

L'instance gitlab de l'université de possède pas de registry docker. Dockercloud perme Dockercloud n'est pas compatible avec gitlab, pour pouvoir profiter de la ci il a falu faire un mirroir github.

Developement

Le fait que tout le monde ne travaillait pas sous le même OS, le developpement à posé quelques soucis en début de projet.

S'adapter aux configs

Il a falu faire du cas par cas afin d'installer go, unity et des dépendances sur chaque machines.

docker

Pour palier à ce problème, l'utilisation de docker à permis de faciliter le développement. Par ailleurs cela à était long et plusieurs personnes ont du passer de windows familial à windows pro.

git

Pour developper, l'utilisation de git semblait évidente.

branches

Pour ne pas se marcher dessus et travailler en parallèle chaque feature était développer sur des branches indépendantes excepté pour l'équipe front.

b -> develop Dès qu'une feature est jugée terminé. Une pull request est ouverte pour merge les modifications. Cela permet d'intégrer les features au fur et à mesure et d'avancer par itérations

Deploiement

La phase de deployment conventionel peut être compliqué et demander des manipulations spécifique pour metre à jour un service. Nous avons décidé d'intégrer nos service dans un cloud privée en conteneurisant nos services dans des conteneurs.

Kubernetes

Kubernetes est un orchestrateur de conteneur. Grace aux api de kubernetes, il est facile d'augmenter la charge de calcul, gagner en redondance et réduire le taux de panne. Kubernetes est aussi penser pour faciliter l'intégration continu, il est très facile de metre à jour des service sur un nombre de serveur infini.

HA

La Haute disponibilité permet un taux de panne proche de 0. Pour ce faire, nous augmentant le nombre de serveurs physiques, nous générons plusieurs instances de chaque service sur les différents noeuds. Nous profitons d'un système de stockage redondant reduisant le risque de perdre des données.

Montée en charge

A terme grace aux outils cités précédément, notre infrastructure permetra d'automatiquement ajuster le nombre de noeuds de chaque service pour répondre à des montés en charge.

Test

Nous avons essayé d'intégrer un maximum de test unitaire pour détecter et corriger un maximum d'erreurs. Cela nous permet de gagner du temps en s'investissant moins dans la recherche de bugs.

Go test

L'un des avantages de go et l'outils go test, il permet de lancer très facilement nos tests unitaires.

Data race

L'outil go test permet également de détecter les data races, nous en avons rencontrés un très grand nombre.

Partie Personel

Timothée Oliger

Tout d'abord je tiens à remercier mes collaborateurs, c'est ensemble que nous avons pu terminer ce jeux.