\begin{titlepage}

UNIVERSITÉ DE STRASBOURG\[1cm] \textsc{Cursus
Master Ingénierie – Informatique\[0.5cm] L3S6\[0.5cm]

\[0.4cm] { **Rapport projet intégrateur}\[0.4cm] \[1.5cm]**

Auteurs :
Timothée Oliger
Adrien Ossywa

~

Responsable :
Stephane Cateloin

\[1cm]

Sujet\ *Tiny Empire*\[1.5cm]

~

\[1cm]

{May 2, 2019}

\end{titlepage}

Contents

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'approcer l'utilisation de scripts et la manipulation de l'interface de Unity. Ces connaissance 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 posible 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 alégée 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'arcitecture 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 peronnes 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, ilPyhton 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

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

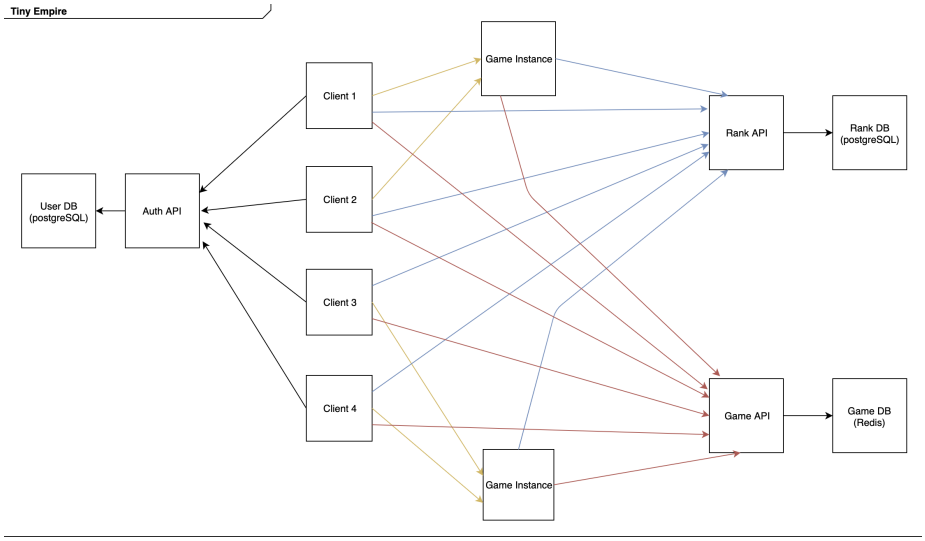

Figure 1: infrastructure

Micro-services

Les avantages d'une architecture de type "micro-services" sont : * l'indépendance entre les services * la quantité de code raisonable et facilement assimilable contenue dans chaque service * la possibilité d'augmenter la robustesse de l'application en dupliquant uniquement les services les plus solicités.

API authentification et d'informations sur les lobbies

Nous avons décidé de choisir la réalistion d'une API GraphQL afin de se connecter/enregister et obtenir la liste des serveurs disponibles.

GraphQL **GraphQL est un language créé par Facebook. Même si REST (Representational State Tranfer) demeure le format standart des API, certains développeurs décident de se tourner vers GraphQL pour combler leslacunes 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 partie 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

Pas faite

Instance de partie

Les instances de partie sont écrites en GO et conteneurisées 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 moteur de jeu 3D côté clien, nous avons eu des hésitations entre GoDot et Unity. Comme précisé plus haut, les membres de l'équipe en CMI Image ont déjà utilisé l'interface de Unity et sont donc déjà habitués à son utilisation. Malgré des doutes sur la compatibilité entre Git et le fonctionnement de Unity, contrairement à GoDot

Unity **Unity est un moteur de jeux propriétaire possédant un nombre conséquent d'assets et de fonctionnalités puissantes permetant de construire un jeux avec des outils performants. L'avantage de Unity comparé à Godot, est que nous avon 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 départagé un vote à été réalisé et notre choix s'est dirigé vers unity.**

Organisation

Pour créer le jeux 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.

Répartition des taches

La répartition des taches se fait grâce à gitlab, les issues sont créés et attribué ou choisi par les personnes disponibles. Ce système permet de d'attribuer des taches 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 fonctionnabilités.

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 thermes 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.yaml de déclarer un pipeline qui peut tester, publier et deployer des

solutions logiciels

Webhook gitlab

Lorsqu'il y a un évenement 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 chaques machines.

docker

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

git

Pour developer, 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 deployement conventionel peut être compliqué et demander des manipulations spécifique pour metre à jour un service. Nous avons décidé d'intégré 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.