Werewolf game on Android

Conception d’une application Android implémentant le jeu du loup-garou

Une image contenant texte

Description générée automatiquement

Screenshot de l’application ici

!!! VIDEO !!!

Samir ALMOU, Anuraag POTHULA, Harjeet NAGI

3 Janvier 2021

HES-SO MSE – T-MobOp – Mini-projet

# Table des matières

[1. Table des matières 2](#_Toc59783610)

[2. Introduction 3](#_Toc59783611)

[3. Analyse 3](#_Toc59783612)

[4. Conception 3](#_Toc59783613)

[5. Implémentation 3](#_Toc59783614)

[6. Problèmes rencontrés 3](#_Toc59783615)

[7. Conclusion 3](#_Toc59783616)

[8. Annexes/Références/Bibliographies 4](#_Toc59783617)

# Introduction

Dans le cadre du cours T-MobOp sur Android, nous avons choisis de concevoir une application implémentant le jeu du loup-garou.

[Expliquer le jeu du loup-garou en quelques lignes]

Avec ce choix, nous souhaitions réaliser une application différente par rapport à ce qui a été choisis par les autres étudiants du cours. bla bla bla

[but et motivation]

# Analyse

[Comment on a souhaité faire l’application ?]

Le jeu est multi-joueur en ligne avec la possibilité d’ouvrir plusieurs salles. L’accès à une salle se fait par son nom.

Un serveur héberge les données et permet à chaque joueur de recevoir toutes les infos du jeu.

Un ‘host’ ouvre une salle et communique le nom de la salle aux autres joueurs. Son appareil sera ‘maitre’ de la salle et de la database.

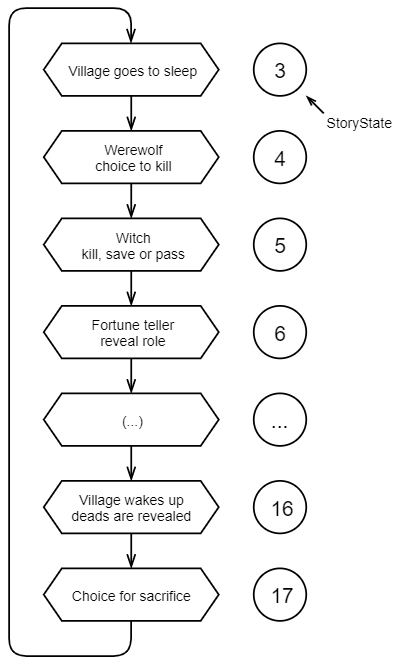
## Ouverture d’une salle, configuration et attente

Tout le monde peut créer une salle et configurer le nb de joueur et la distribution des rôles.

WaitingRoom

## Rejoindre une salle

## Déroulement global du jeu et variable *StoryState*

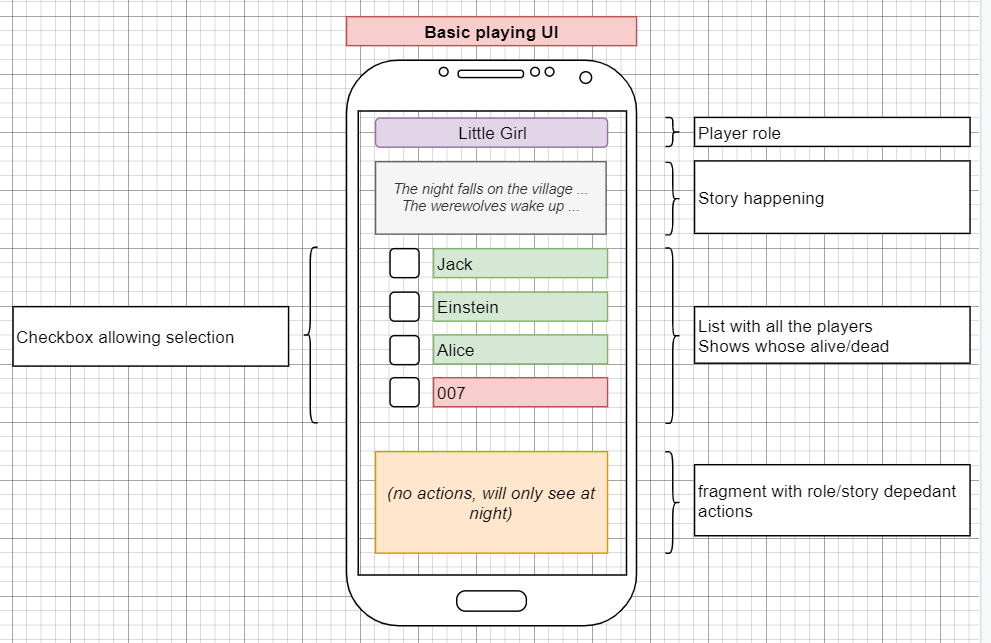
Le jeu suit le déroulement classique du jeu original :

* Le village s’endort.
* Les loups-garous se réveillent, choisissent leur victime et se rendorment.
* La sorcière joue son tour, ensuite la voyante, etc ...
* Une fois tous les rôles s’étant succédés, le village se réveille, les morts sont dévoilés et le village choisit le sacrifice.

Une variable nommée *StoryState* permet de suivre l’évolution du jeu. A chaque changement d’étape, cette variable est incrémentée [par le host] du jeu de façon automatique. (Les valeurs suggérées dans le digramme ci-contre sont indicatives.)

Le jeu s’adapte selon les rôles en cours ; si la voyante a été tuées, l’étape 6 est automatique sautée.

## Interface générale lors du jeu



## Serveur : Firebase, Realtime Database

Afin de communiquer toutes les données à tous les joueurs participant à une partie, une base de données en ligne est utilisée. Dès l’ouverture de l’application, chaque appareil reçoit une copie de l’ensemble de données sauvegardées.

Firebase est une plateforme de développement supportée par Google qui offre de nombreuses fonctionnalités, API et outils. Pour notre projet, nous avons utilisé la *Realtime Database*, une base de données en temps réel. Elle permet de facilement écrire des données et génère des events à chaque modification de sorte à informer tout appareil connecté en temps réel.

Firebase est un outil très intéressant pour le développement de petites applications telles que la nôtre.

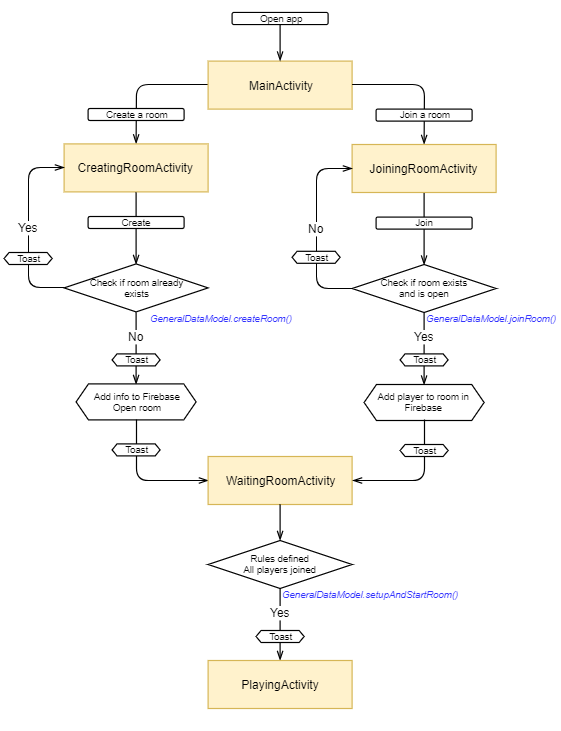
serveur, code backend, gestion automatique ressource, gratuit jusqu’à un certain quota d’utilisation ...

Afin d’assurer le fonctionnement du jeu, un paquet similaire à celui présenté ci-contre est utilisé. Le format utilisé est l’arbre JSON.

# Conception

## Conception de l’interface utilisateur UI (icon, thème, ...)

## Déroulement des activités (Main, joining, waiting room, playing activity)



## Realtime Database de Firebase

L’entièreté de l’implémentation de la communication se fait à travers d’un unique objet appelé *GeneralDataModel*.

bla bla bla functions /

# Implémentation

## Code des fragments des rôles (xml + activity)

## WaitingRoomActivity et création des règles personnalisées

## GeneralDataModel : base de données

## Fonctionnement du gameplay

# Problèmes rencontrés

problème avec firebase / manque de documentation, exemple

pas connaisseur Java ... ?

# Conclusion

Amélioration à apporter

Safeguard : protection contre trop de joueur, check doublon joueur, joueur qui quitte sans prévenir, timer par actions

Test avec plusieurs salles ouvertes (manque de tel pour tester)

Salles publiques, n’importe qui peut rejoindre

Firebase functions, tout le code en backend avec la Database

Qui a fait quoi dans le projet ?

Samir : WaitingRoom, ...

Anou : Interface, ...

Nagi : Firebase, ...

# Annexes/Références/Bibliographies