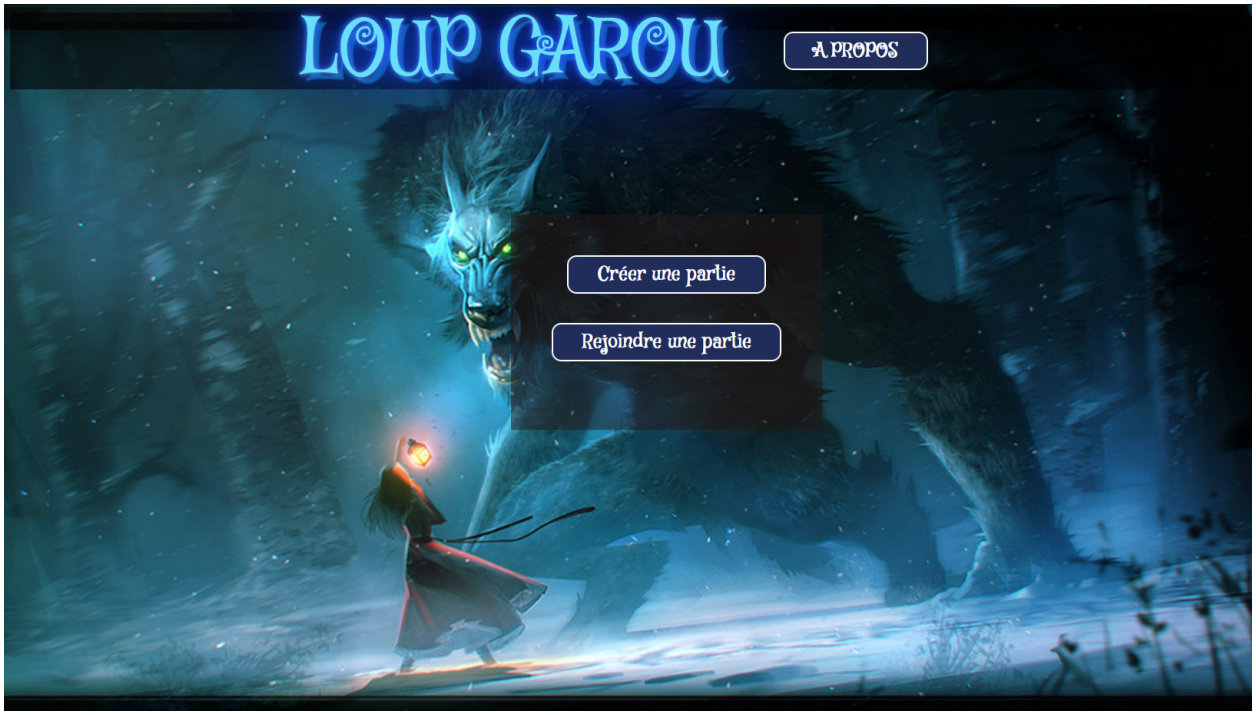
Rapport LIFProjet : Jeu du Loup Garou



Sommaire :

27/04/2021

Par : Nicolas CHAMAND, Vak Sam Sophie HAM, Duc Anh NGUYEN

Encadrant : Gabriel RADANNE

UNIVERSITÉ DE LYON

Université Claude Bernard Lyon 1

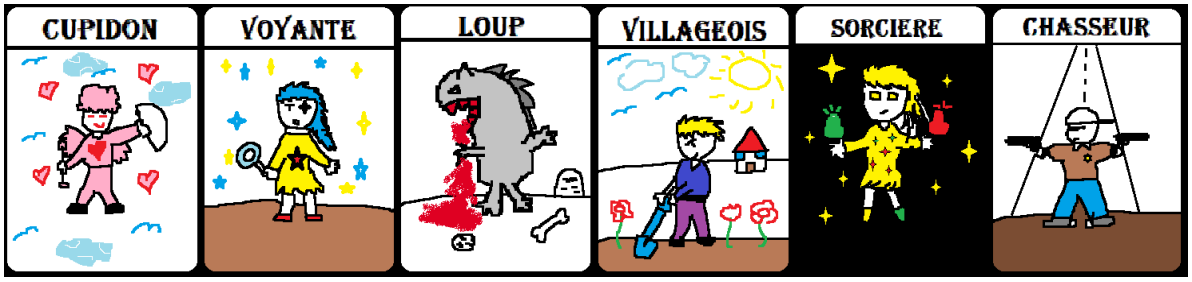
I - Vue d'ensemble

1) Description du projet

Ce semestre, en LIFProjet, nous avons choisi de réaliser un jeu de plateau multijoueur en ligne. Notre choix s'est vite porté sur le jeu du Loup Garou. C'est un jeu de société paru en 2001 et qui est de type **jeu de rôle** où chaque joueur incarne soit un **villageois** soit un **loup garou**. Le but des villageois est de démasquer et tuer tous les loups garous avant que les loups garous les tuent en premier. Il existe également **d'autres personnages** dont le but est soit d'aider les villageois à remporter la partie soit de gagner seul. Une partie doit être menée par une personne ne prenant pas part au jeu : il s'agit du **narrateur**.

Chaque personnage a des capacités différentes :

- **Les villageois** ont pour but d'exécuter tous les loup-garous.
- **Les loups garous** ont pour but d'éliminer tous les villageois.
- **Le maire** compte pour 2 voix lors des votes.
- **La voyante** peut découvrir l'identité d'une personne à chaque tour.
- **Le chasseur** peut choisir une personne à exécuter lorsqu'il meurt.
- **Cupidon** lie le destin de 2 personnes, si l'un meurt l'autre meurt aussi.
- **La sorcière** a le pouvoir de ressusciter la victime des loups garous ou de tuer un joueur.
- **La petite fille** peut, pendant la nuit, entrouvrir les yeux pour démasquer les loups garous. Mais si elle se fait prendre, elle peut être tuée.



Différentes cartes des personnages de notre jeu du Loup Garou

Le jeu est jouable sur un navigateur web. Il est réalisé avec 3 langages web principaux : HTML, CSS et Javascript. Nous utilisons **ReactJS** comme bibliothèque principale pour la création de l'application. ReactJs nous permet de bien spécifier sous forme de composants, les différents éléments de notre page et de les rendre dynamiques plus facilement. En autre, le rendu de la page complète se résume au rendu des différents composants que contient la page et la modification d'un de ses composants n'entraîne pas le rechargement complet de la page. (on parle de "single page application")

Un serveur de la bibliothèque **PeerJS** est implémenté pour la mise en place des connexions et des échanges entre clients. Nous n'avons donc aucune donnée à stocker sur un serveur distant et donc pas besoin d'utiliser le langage PHP. De plus, nous avons une totale liberté sur le type de données que nous souhaitons envoyer et sur comment elles seront traitées à l'arrivée chez le client.

Nous utilisons également **Google Drive** pour l'élaboration de notre cahier des charges mais aussi pour la rédaction du rapport ainsi que le PowerPoint pour l'oral. Notre projet est sauvegardé dans un répertoire **GIT** pour permettre de garder une trace de toutes nos modifications.

Pour la création de cette interface, nous nous sommes fixés différents objectifs à réaliser :

- Un module de connexion permettant de créer ou de rejoindre une session de jeu pouvant aller jusqu'à 10 joueurs.
- Un formulaire de chat permettant l'envoie et la réception de messages par tous les joueurs de la session.
- Un narrateur donnera des indications sous forme de texte pour faire avancer le jeu. (bonus : possibilité de le faire en audio pour rendre le jeu plus vivant)
- Un système de vote permettant aux joueurs de choisir quel autre joueur ils souhaiteraient éliminer pour le tour (ou sauver/élire dans les autres cas). Au moment du vote, les joueurs auront un temps limité pour choisir quelqu'un : il faut réaliser un timer.
- Un affichage prenant en compte les dimensions de l'appareil utilisé : page web responsive.

2) Organisation

Pour organiser notre travail, nous avons tout d'abord réalisé un diagramme de classes qui nous permettait de séparer et visualiser les différents composants du jeu et les méthodes utiles à son fonctionnement. Nous avons surtout représenté, dans le diagramme ci-dessous, la partie modèle du jeu, c'est-à-dire la partie qui gère les données du site donc tout ce qui était gestion, organisation et assemblage des informations. Nous avons abouti à la création de trois classes différentes :

- Une classe ***Connexion*** qui va permettre de gérer les connexions entrantes et sortantes entre les différents joueurs d'un partie au travers d'un serveur PeerJS. Elle conserve toutes les informations concernant les joueurs et gère l'envoie des informations dans le jeu et dans le chat.
- Une classe ***Joueur*** qui permet de stocker les informations concernant le joueur comme son identifiant, la carte qui lui a été attribuée durant la partie et un statut qui permet de savoir s'il est maire.
- Une classe ***Jeu*** qui comporte toutes les méthodes principales du jeu, donc les fonctions qui vont permettre l'avancement et le bon fonctionnement d'une partie comme choisir une victime, sauver ou empoisonner une personne, élire un maire et retirer le joueur de la partie.

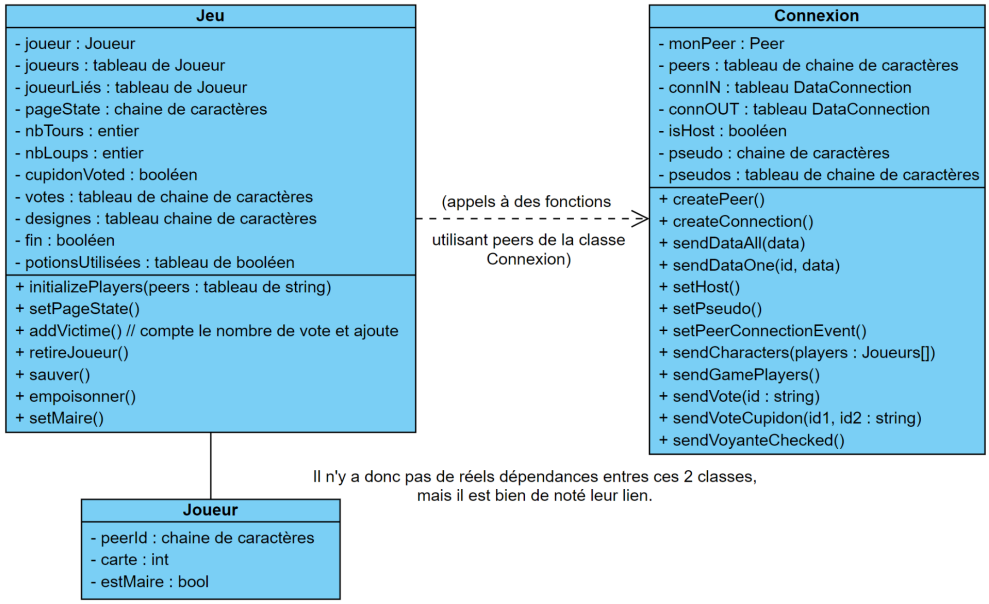


Diagramme de classes du jeu du Loup Garou

Nous avons également un diagramme d'états qui représente les différents états dans lequel passe le jeu au cours d'une partie pour savoir quelles actions doivent être effectuées à ce moment précis. Le jeu passe par six états différents au cours d'une partie. Nous nous retrouvons d'abord dans l'état "Homepage" donc l'état d'accueil où un joueur va créer une partie en tant qu'hôte et va attendre que d'autres personnes le rejoignent d'où l'état "Attente" suivant. Lorsque la partie démarre, un rôle sera attribué aléatoirement à chaque joueur avant de passer à l'état "Tombée de la nuit". Pendant la première nuit, Cupidon se réveille pour choisir les destins liés puis la voyante se réveille à son tour. Ensuite, c'est aux loups garous de choisir leur victime au cours d'un vote puis un comptage se fait, la sorcière se réveille pour user ou non de ces pouvoirs, ce qui peut ainsi remettre en question le choix précédent. Après l'état de "Vote Loup", nous arrivons au "Levé du Jour" qui va permettre d'élire un maire et de passer à l'état de 'Vote Village" pour choisir le coupable. A la fin de ce jugement, la nuit retombe et tout recommence jusqu'à ce qu'il ait un vainqueur.

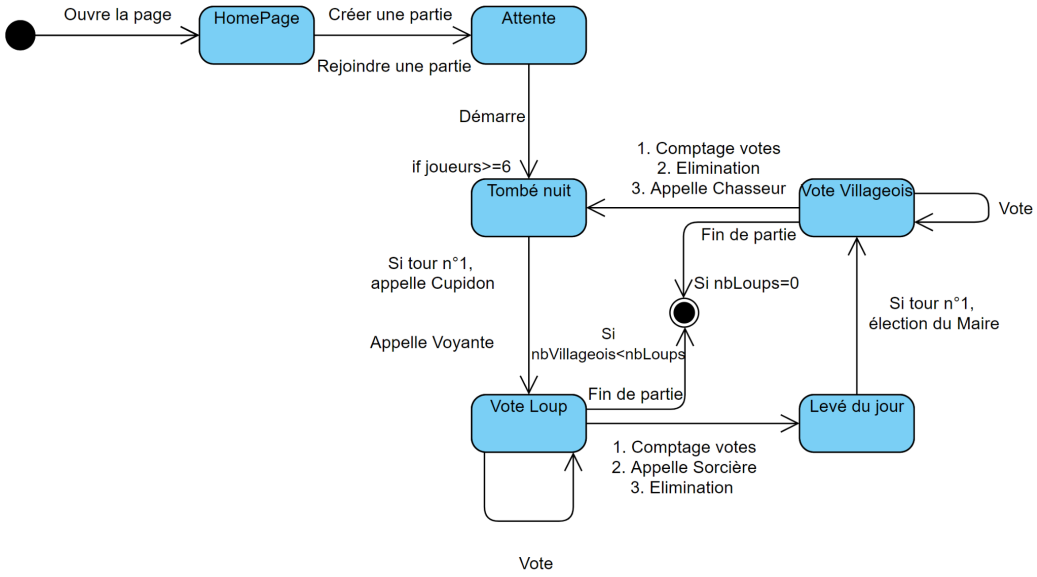


Diagramme d'états du jeu du Loup Garou

Nous avons également utilisé un Scrum Board pour pouvoir répercuter nos tâches et éviter que tout le monde se mette à modifier le même morceau de code pour créer une même fonctionnalité. Cependant, nous avons vite laissé tomber cet outil étant donné que nous nous entraidons pour coder.

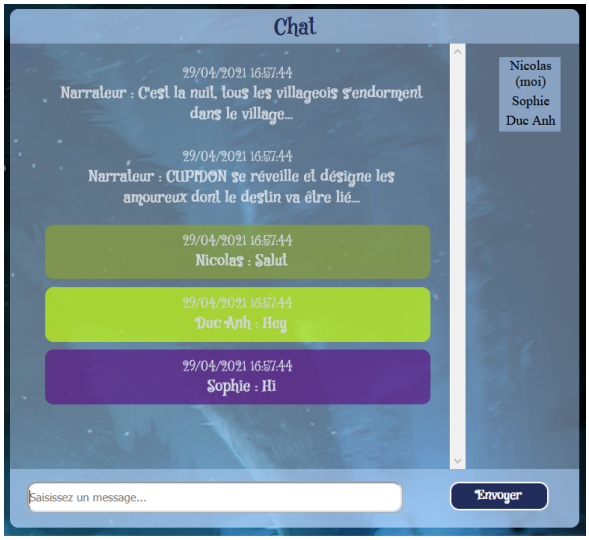
II - Contexte de la réalisation

1) Travail réalisé

A la suite de la création des différents diagrammes, nous avons codé notre jeu suivant un schéma Modèle-Vue. On retrouve d'une part un répertoire "Components", avec l'ensemble des composantes graphiques du jeu comme le chat, les cartes ou les votes… Et d'une autre part, la partie modèle qui regroupe les trois classes vu précédemment pour la gestion des données et les fonctionnalités du jeu.

L'un des modules le plus important est le module de connexion. Il est basé sur les connexions P2P (Peer to Peer) comme nous l'avions annoncé. Le module contient le pseudo de tous les joueurs

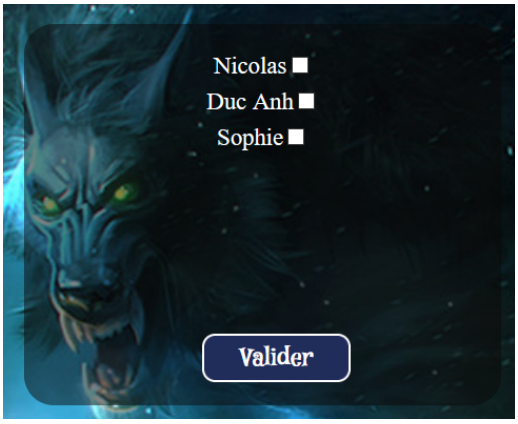
connectées. Il permet de créer un “peer” permettant la connexion d’autres peers. A noter, que les peers doivent être connectés dans les “deux sens” pour l’envoie ET la réception de données. Les données sont en général envoyées sous forme de tableau associatif et nous avons décidé de leur associer un attribut “type” qui permet de déterminer si le type de données correspond à un message, un vote de Cupidon, un vote de Loup etc… Un aspect “non visible” de la conception de ce module a été la création des différents types de données que l’on s’attend à recevoir au cours du jeu.



Capture du chat de jeu

Un chat a été ajouté pour permettre aux joueurs de discuter ensemble tout au long de la partie. Pour mettre en place ce chat, il a fallu créer différents composants : un espace pour afficher la liste des messages échangés, la liste des joueurs et un formulaire pour envoyer des messages. Tous ces composants sont regroupés en un seul composant principal : le chatbox. Les données envoyées au travers du formulaire passent par une fonction sendAll() dans le modèle qui va pouvoir renvoyer le message à l’ensemble des utilisateurs. Le chat inclut également le narrateur.

C’est l’hôte qui décide du lancement de la partie (lorsqu’il y a plus de 6 joueurs). Nous avons réalisé un component “Vote” qui permet d’implémenter les différentes fonctionnalités des personnages : Cupidon pour choisir les deux amants, la Voyante pour dévoiler une carte et les Loups pour éliminer un joueur. La fonctionnalité de votes peut être facilement adaptée en fonction du personnage. Le jeu s’arrête aux votes des Villageois. Les fonctionnalités concernant le comptage de votes et l’élimination d’un joueur ont été réalisées dans la partie modèle.



Capture d’écran du formulaire de vote

En clair, par rapport à ce qui avait été prévu dans le cahier des charges, nous avons réalisé le module de connexion, le chat, les système de vote et son timer. En revanche, nous n’avons pas pu réaliser l’aspect responsive de la page et un affichage plus dynamique avec React pour le narrateur.

2) Difficultés rencontrées

a) Avec React

Nous avons rencontré pas mal de difficultés au cours de la programmation du jeu étant donné que ReactJS était un tout nouvel outil pour nous. Etant donné qu’aucun de nous n’avait utilisé React auparavant, nous avons passé pas mal de temps à apprendre à utiliser cette bibliothèque. Nous avons rencontré des difficultés lors de la programmation avec la création de composant en JSX, qui n’est pas très loin du HTML mais qui avait ses spécificités.

b) Dans la classe Jeu

Une des difficultés rencontrée a été de gérer le changement d'état du jeu, de faire appel à des fonctions dans d'autres classes. L'état de la page est une donnée qui devrait être stockée dans le modèle, par exemple dans la classe Jeu (qui ferait office d'état du jeu). En utilisant React, nous avons compris que le rechargement de la page s'effectuent lorsque l'état (state) d'un composant a été changé. Les données que nous recevons par PeerJS se trouvent, au niveau de notre code, dans la partie modèle (précisément dans le module de connexion). Il est alors facile de changer l'état de la page dans la classe Jeu mais il faut aussi changer l'état des composants React pour faire le rechargement (sauf que nous n'y avons pas accès directement). Nous avons alors mis des références de certains composants dans le modèle (composants globaux comme "App") nous permettant ainsi de modifier l'état de celui-ci quand l'état du jeu change et donc de rafraîchir la page avec les composants modifiés.

c) Dans le module de connexion

La mise en place du module de connexion nous a également pris du temps sachant qu'au cours de la réalisation du projet, les serveurs PeerJS sont tombés en panne donc nous avons dû créer un serveur local pour pouvoir avancer dans notre projet. Pour négocier les connexions, PeerJS se connecte à un PeerServer. Le serveur agit uniquement en tant que relais de connexion mais aucune donnée ne passe par le serveur. L'inconvénient avec le serveur local et qu'il n'était pas possible de tester le jeu en multijoueur sur des ordinateurs différents mais le problème des serveurs PeerJS a été résolu.

Une des difficultés a été également de prendre en compte l'arrivée des différentes données lors de la connexion. En effet, lorsqu'un peer se connecte à un autre, il doit envoyer son pseudo à ce dernier. De son côté, le peer ajoute l'id de celui qui se connecte mais il doit faire attention à comment ajouter le pseudo, puisque des retards à l'envoi du message et la connexion de plusieurs autres peers peuvent entraîner des mélanges de pseudos.

Une autre difficulté a été de choisir la manière dont s'interconnecte les joueurs. Nous avons opté pour une interconnexion totale (chaque peer est connecté à tous les autres peers). Ainsi, si "l'hôte" se déconnecte, les données peuvent quand même circuler. En revanche, pour des raisons de simplicité, les données de votes sont envoyées à l'hôte de la partie pour comptage. Le calcul n'est ainsi effectué que sur une seule machine et le résultat est retourné à chacun. Le problème ici est que si l'hôte se déconnecte, la partie ne peut plus avancer.

Une difficulté cette fois-ci liée à l'organisation du travail pour les éléments du modèle. En effet, nous n'avions que 3 classes à réaliser. Bien que ses classes comportaient beaucoup de fonctionnalités, nous ne codions pas en général à plus d'un sur le même fichier de modèle pour des raisons d'organisation et de séparation de code. Pour la plupart du temps, les fonctionnalités ont été codées à deux par le biais de Discord ce qui facilitait la programmation en elle-même, étant à deux pour réfléchir sur un problème et qui permettait de régler le problème de séparation des fonctionnalités à coder dans le modèle.

III - Bilan

Malgré pas mal de difficultés rencontrées au cours de la programmation de ce jeu, nous sommes assez satisfait de ce que nous avons pu réaliser tous ensemble. Il est vrai qu'on aurait bien voulu terminer le jeu et ajouter des personnages en plus comme le maire et la sorcière ainsi que d'autres personnages bonus comme la petite fille, le chasseur et le voleur. Par manque de temps, la partie responsive reste à revoir mais dans l'ensemble ce projet a été très enrichissant pour nous trois. Nous avons appris à travailler

ensemble, en équipe, et surtout en distanciel. Nous avons également pu monter en compétence technique avec l'apprentissage d'une toute nouvelle bibliothèque tel que React.

Avec React, nous avons appris à séparer les éléments qu'on souhaite obtenir en différents composants ayant des fonctionnalités distinctes (voir exemple Chat). Nous avons appris également à faire des affichages dynamiques avec l'utilisation de l'état du jeu mais également à faire des appels de fonctions d'un composant vers une autre au travers d'un attribut JSX appelé "props".

Ceci est l'un de nos premiers projets en Web n'utilisant pas de serveur externe pour l'envoi de données. L'utilisation de PeerJS a été tout nouveau pour nous et cela nous a laissé une grande liberté dans le traitement des données. Elle nous a permis d'apprendre à mettre en place un petit système de traitement des données que l'on reçoit par les peers (en particulier, différencier les types de données qui arrive, message ou vote). En revanche, au cours de ce projet, nous ne nous sommes pas préoccupés de la partie sécurité des données.

Sources :

Règles du jeu :
https://fr.wikipedia.org/wiki/Les_Loups-garous_de_Thiercelieux
http://lesloupsgarous.free.fr/htm/regles.html

Exemples de jeu du Loup Garou :
https://wolfy.fr/
https://www.loups-garous-en-ligne.com/

React :
https://fr.reactjs.org/
https://openclassrooms.com/fr/courses/4664381-realisez-une-application-web-avec-react-js/4664388-decouvrez-lutilite-et-les-concepts-cles-de-react

PeerJS :
https://peerjs.com/docs.html#start
https://github.com/peers/peerjs