HACK.ME

DOCUMENTATION TECHNIQUE

Table des matières

Prérequis

- Un serveur Web
- Une base de données MySQL
- Node
- Java 9

Comment ça marche

Tout d'abord vous devez démarrer votre serveur web ainsi que votre base de données, une fois cela fait lancez l'API node puis le site web (hackmeweb)

Ensuite lancer le setup.exe pour installer l'application sur votre ordinateur, puis lancer l'application.

Ensuite une fois le programme lancer une fenêtre de connexion apparaitra, une fois connecter vous pourrez vous déplacer sur l'application et ainsi vous pourrez télécharger des plugins, des maps ou jouer.

Developpement

Api node

Les modèles :

User : Email, Username, Password, Admin

Les champs Email et Username devront être unique, et deux utilisateurs ne pourront pas avoir les même.

Le champ password devra être chiffrer en base pour ne pas être visible.

Le champ admin permettra de différencier les différents droit d'accès de chaque utilisateur 0 pour un user normal, 1 pour un modérateur, 2 pour un admin, envisager le -1 en cas de bannissement.

Map : Nom, Description, Zip des sprites, Zip des énigmes, Fichier de la carte

Le nom sera unique à chaque carte, deux cartes ne peuvent pas avoir le même nom.

Les énigmes seront contenues dans le dossier zip au format json et auront entre 1 et 3 enigmes possible. Le Zip des sprites doit impérativement être un Zip et non un RAR (ou autre format de compression) pour faciliter le téléchargement des cartes.

Le fichier de la carte sera un fichier .tmx qui est fournit après la création sous Tiled (logiciel de création de carte)

Plugin : Nom, Description, Jar du plugin

Le nom est unique à chaque plugin, deux plugins ne peuvent pas avoir le même nom.

Score : Nom du joueur, nom de la carte, score

Chaque pair joueur et carte est unique, si ce couple existe le score de ce dernier est mis à jour, dans le cas contraire, ce couple est créer et le score y est affecté

Les controllers :

Tous les controllers doivent permettre la création, et la mise à jour de certains champs du modèle attitrer.

Ils devront permettre la recherche des modèles dans les cas nécessaires (un utilisateur selon son email, une carte selon son nom, etc...)

Le controller du modèle User devra comporter une méthode la création d'un token JWT lors de la connexion réussi d'un utilisateur (plateforme web ou client lourd) ainsi qu'une méthode permettant l'identification de ce token lors d'un demande de connexion d'un utilisateur.

Une méthode doit aussi être capable de vérifier si l'utilisateur est un admin, modérateur, ou simple user.

Dans le cas du controller de score, il devra avoir une méthode trouvant tous les scores pour une carte, tous les scores pour un utilisateur, et le score d'un utilisateur sur une carte particulière.

Les routes :

Toutes les routes HORS GET devront être protéger par la fonction qui permet de vérifier si le token renvoyer par un utilisateur est valide, les fonctions GET seront accessibles à tout le monde.

Map : Une route d'upload qui prendra tous les fichiers nécessaires au bon fonctionnement de la carte.

Une route de download permettant de récupérer l'ensemble des fichiers d'une carte dans un fichier zip.

Une route permettant d'accéder à toutes les maps disponible sur le serveur.

User : Une route d'inscription, acceptant un email, un nom et un mot de passe.

Une route de connexion demandant un nom et un mot de passe pour renvoyer (en cas de succès) un token jwt permettant l'accès aux autres routes.

Plugin : Comme pour la carte, une route d'upload prennant un nom et une description ainsi que le fichier jar.

Une route permettant de download ce .jar.

Une route permettant de voir les plugins disponible sur le serveur.

Score : Une route d'ajout de score qui permettra d'ajouter ou de mettre à jour le score du joueur ou de la carte choisis.

Il devra contenir une route pour chaque cas possible, afficher chaque score d'une map, chaque score d'un joueur, ou le score d'un joueur sur une map.

Client lourd

Les IHM

Les IHM ont été créée à partir du scène builder d'Oracle, puis nous devons les placer dans le dossier view de notre application.

Ensuite chaque vu (fichier fxml créé avec le scène builder) possède un controller, dans le controller pour accéder au éléments de notre vu il faut créer une variable du même nom que le fx:id de l'élément et du même type que l'élément (AnchorPane, Button, …) avec l'annotation @FXML au-dessus.

Vous pouvez désormais utiliser vos éléments dans votre code.

Vous pouvez aussi créer une méthode nommé "public void initialize()" cette méthode définie le traitement à exécuter avant l'affichage de la page.

Le jeu

Librairies

La partie jeu est développer avec la librairie slick2D, qui est une librairie se basant sur openGL.

Utilisation de qj.util, une librairie permettant de recharger des classes dynamiquement, dans le but de chargé les classes créer par l'utilisateur pour pouvoir récupéré les annotations à l'intérieur de son code.

Utilisation de jackson pour pouvoir instancier des objets grâce à un JSON.

Partie jeu

Le jeu fonctionne avec des classes d'états, quand on lance le jeu on dirige le programme vers un certain états qui derrière va s'initialiser et lancer une boucle update et render qui seront nos boucles de jeu.Il suffira ensuite de passer entre chacun des états
Le jeu est découpé en 3 états:
MapState : La phase ou l'utilisateur se trouve sur la carte le personnage.
CodeState: La phase ou l'utilisateur doit coder.
EndState: L'écran de fin.

Des classes modèles ont été créer:

La map.
Créer à partir d'un .tmx, cette extension permet de créer des maps facilement a partir d'un éditeur de map.

Le Joueur.

L'énigme.
C'est l'énigme de code proposer au joueur.

L'épreuve.
Une surcouche de l'énigme faisant le lien entre l'énigme et la compilation.

Un groupe de classe HUD est utilisé par les classes d'états pour afficher certain composant graphique du jeu.

TriggerController est une classe permettant de faire la jonction entre les états et entre différentes classes d'affichage et de données.

Partie compilation

Il existe une petite partie compilation permettant d'exécuter le code de l'utilisateur et de récupérer la sortie d'erreur en cas de besoin.
Le parseur d'annotation se trouve dans cette partie car les annotations sont récupérées depuis le code de l'utilisateur.

La librairie

Il y a une librairie "hackmelibrairie" possédant les interfaces utiles au chargement et à la création des plugins. Pour créer cette librairie nous avons donc créé un projet maven dans lequel nous avons créé des interfaces avec des méthodes que l'utilisateur devra redéfinir.

Cette librairie sera accessible via la dépendance maven :

```
<dependency>
    <groupId>com.esgi</groupId>
    <artifactId>hackme-library</artifactId>
    <version>1.0.0</version>
</dependency>
```

Les plugins

Charger un plugin

Pour cela une classe PluginLoader à été créée permettant de charger la classe principale d'un jar.

Elle possède un constructeur par défaut dans lequel un chemin vers le dossier de plugin par défaut est donné et un second constructeur dans lequel vous pouvez définir le chemin vers le dossier que vous souhaitez.

Ensuite elle possède une méthode loadPlugin qui prend en paramètre une String qui est le nom du jar que vous souhaitez charger et elle vous renvois un objet que vous devrez caster dans l'objet que vous désirez pour l'utiliser.

Cette classe ouvrira votre jar et ira lire le fichier Manifest.mf pour déterminer quelle est la class main de votre plugin, ensuite elle chargera cette Class grâce à un ClassLoader avant de vous la retourner sous la forme d'un objet.

Créer un Plugin

Pour créer un Plugin vous devez importer une librairie contenant les interfaces que vous pourrez utiliser dans vos plugins :

```
<dependency>
    <groupId>com.esgi</groupId>
    <artifactId>hackme-library</artifactId>
    <version>1.0.0</version>
</dependency>
```

Ainsi que la dépendance du jeu :

```
<dependency>
    <groupId>com.esgi</groupId>
```

```
    <artifactId>hackme</artifactId>
    <version>1.0.0</version>
</dependency>
```

Pour pouvoir utilisez les Class du Client lourd ainsi que leurs méthodes comme celle reliant l'application à l'API node.

Ensuite vous devrez créer une Class main implémentant l'interface que vous souhaitez utiliser dans la librairie et définir ce que vous souhaitez faire dans les méthodes qui vous seront donnée.

Site web

La technologie choisie pour le site Web est le React.

Le site se décomposera en plusieurs page (en dehors de la page d'accueil)

Une page de création de compte simple, demandant l'email, le username ainsi que le mot de passe de l'utilisateur avant de le rentrer en base, sauf en cas de conflit.

Une page d'authentification, qui en cas de succès gardera en localStorage le token afin de pouvoir le réutiliser.

Une page plugins et maps qui proposeront un système similaire. Voir en base les plugins (ou maps) disponible et les afficher sur la page. On pourra y voir le nom et la description de chacun des élements ainsi qu'un bouton de téléchargement.

Dans le cas de la page des cartes, proposer un bouton menant sur la page des Scores.

La page des scores proposera d'afficher l'intégralités des scores de la carte, c'est-à-dire le score de chaque joueur sur cette dernière.

La page de maps proposera un bouton d'upload afin d'ajouter une carte à la communauté. Pour cela, il faudra être connecté, dans le cas contraire, demander à l'utilisateur de s'authentifier avant.

Une fois sur la page proposer un ensemble de champ qui permettra d'ajouter au serveur l'ensemble des fichiers nécessaires à une carte. Ces champs seront les suivants, un nom (unique) une description, un fichier .tmx pour la carte, un Zip pour les sprites et un json pour une énigme. Permettre à l'utilisateur de rajouter une ou deux enigmes (ou les retirer si il change d'avis)

La page plugin devra également proposer d'upload un plugin. La page d'upload proposera plusieurs champs, dont le nom, la description et la possibilités d'ajouter un fichier jar.

Utilisation

Commandes pour le client lourd

```bash

git clone https://github.com/bibeul/Projet_annuel_3AL1_2018.git

cd Projet_annuel_3AL1_2018

cd client_lourd
```

mvn clean install

mvn exec:java -Dexec.MainClass="hackme.Main"

ou

java -Djava.library.path=src/main/resources/lib/natives -jar target/hackme-jar-with-dependencies.jar

```

Ou

Lancer le setup.exe pour installer le client lourd

### Commandes pour l'api node

```bash

git clone https://github.com/bibeul/Projet_annuel_3AL1_2018.git

cd Projet_annuel_3AL1_2018

cd api_PA

npm install

npm start

```

Commandes pour le site web

```bash

git clone https://github.com/bibeul/Projet_annuel_3AL1_2018.git

cd Projet_annuel_3AL1_2018

cd hackmeweb

npm install

npm start

```

Liens

http://deptinfo.unice.fr/twiki/pub/Minfo/GenieLog1415/cours6-GL-minfo-1415.pdf

http://miageprojet2.unice.fr/@api/deki/files/1399/=02_Chargement_dynamique.pdf
```