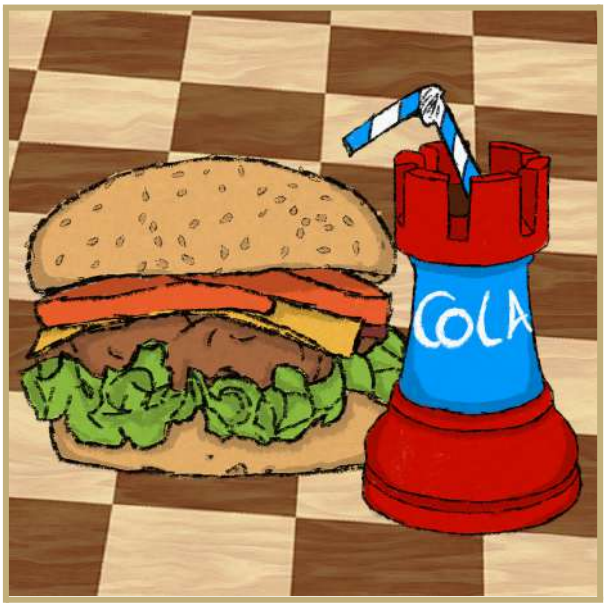
Rapport de projet

Développement d'application mobile d'échec en ligne



Réalisé par Ange CLEMENT **21800812**
Erwan REINDERS **21805614**
Virgil ROUQUETTE--CAMPREDON **21801892**

Dépôt github ChessBurger 2 version Mobile :
https://github.com/virgil-rouquettecampredon/ProjetMobile

Dépôt github ChessBurger 2 version Web :
https://github.com/virgil-rouquettecampredon/ChessBurger2_Web_Client

Lien de la vidéo :
https://drive.google.com/file/d/1Jh-9agK1tKyRXss6qBweY_J-CnUghRWD/view?usp=sharing

Année Universitaire 2021-2022

Remerciements

Nous tenons à remercier Abdelhak Seriai pour ses enseignements des bonnes pratiques de programmation en développement d'applications mobiles, et également Théo Bargiacchi pour avoir fait le logo. Nous tenons aussi à remercier Hugo BEC qui était développeur lors du projet Chessburger premier du nom.

Introduction

De nos jours, il est difficile de concevoir un projet ayant pour vocation d'être partagé au plus grand nombre d'utilisateurs, sans penser à une solution mobile ou WEB. Que cela soit pour promouvoir un nouveau produit ou service, ou encore mettre en place un endroit de partage et d'interactions entre différents utilisateurs, une application mobile est un moyen efficace pour porter ces solutions sur différentes plateformes.

Dans ce souci de présentation d'un service mobile et WEB, nous avons voulu reprendre un précédent projet que nous avons eu l'occasion de développer durant notre formation de licence informatique, en l'ajustant aux besoins des nouvelles contraintes qu'impliquent une telle solution.

Nous allons donc vous présenter notre rendu, en production d'application, du projet ChessBurger, qui permet de jouer aux échecs en local et en ligne contre des utilisateurs connectés. Ces derniers pourront mettre à jour leur profil de joueur, jouer contre d'autres joueurs sous la forme de proposition de défis, et se comparer selon un système de classement aux autres utilisateurs.

Dans un premier temps, nous détaillerons la partie serveur de notre application, développée via Firebase, puis nous présenterons l'aspect visuel de notre solution. Enfin, nous incluons une partie de présentation des classes modèles, importante dans la création d'une mécanique de gestion d'un jeu de plateau où plusieurs joueurs s'affrontent.

1. Back-end

Pour permettre la mise en place d'une application utilisable par le plus grand nombre, il est important de pouvoir intégrer à notre solution un système de gestion de comptes utilisateurs par exemple. Pour cela, et afin de développer les deux clients que sont la version mobile et web, nous avons donc eu besoin d'un endroit pour stocker les données.

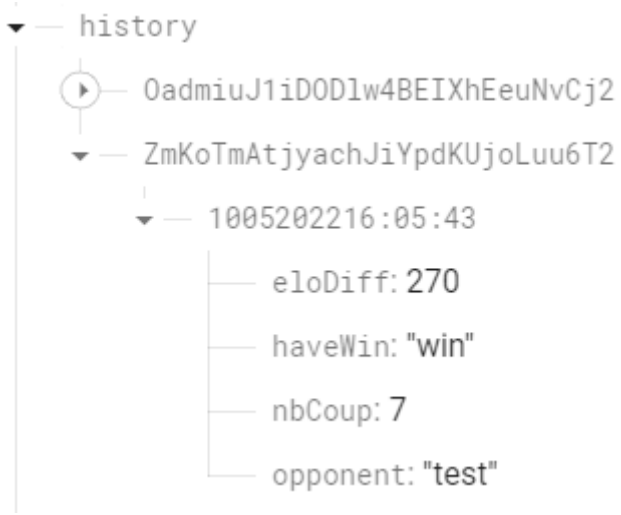
1.1. Technologies utilisée

Pour le stockage des données de jeu et utilisateurs, nous avons utilisé **Firebase**. C'est une solution assez facile à déployer et gratuite, qui nous a permis la mise en place de communications entre tous les joueurs. Nous avons également pu établir des mécaniques de connexions par des fonctionnalités d'authentification présentes sur Firebase.

1.2. Structures données

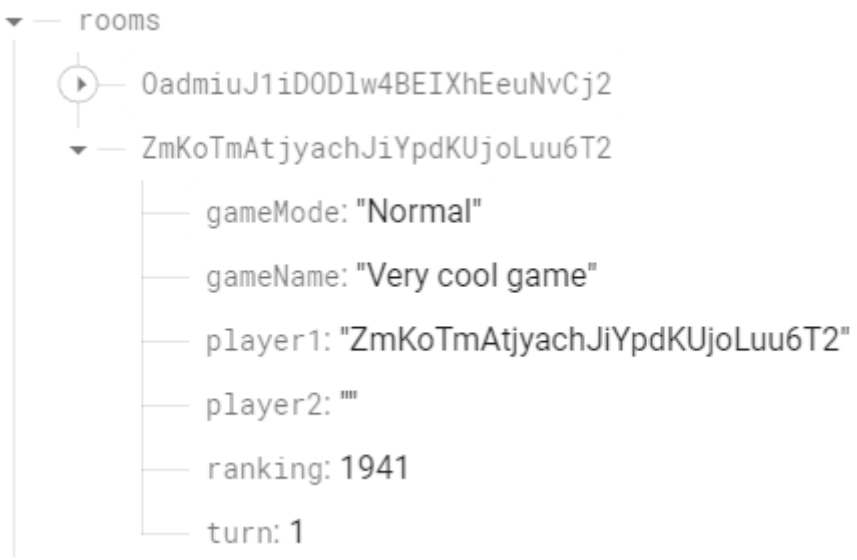
Pour que toute l'architecture du modèle de jeu fonctionne correctement, il a fallu réfléchir à l'organisation des données. Pour cela, nous avons créé 3 grandes classes (même si firebase marche en noSQL) qui sont :

- **History**, qui permet de stocker les historiques de parties. Elle possède comme attributs le pseudo du joueur perdant, l'elo perdu ou gagné et le nombre de coups joués dans la partie de jeu.



Architecture BDD de history

- **Rooms**, qui permet de connecter les joueurs entre eux pour lancer une partie et qui sert de salle d'attente quand il n'y a pas le nombre de joueurs nécessaires pour lancer la partie de jeu. Elle contient les attributs suivants :
  - idJoueur1 (identifiant BDD du premier joueur)
  - idJoueur2 (identifiant BDD du second joueur)
  - elo du joueur qui crée la salle
  - piece1 (déplacement d'une pièce sur le plateau de jeu)
  - piece2 (déplacement possible d'une seconde pièce de jeu, cas spécial du Roque)
  - turn (indication sous forme de jeton du joueur courant).



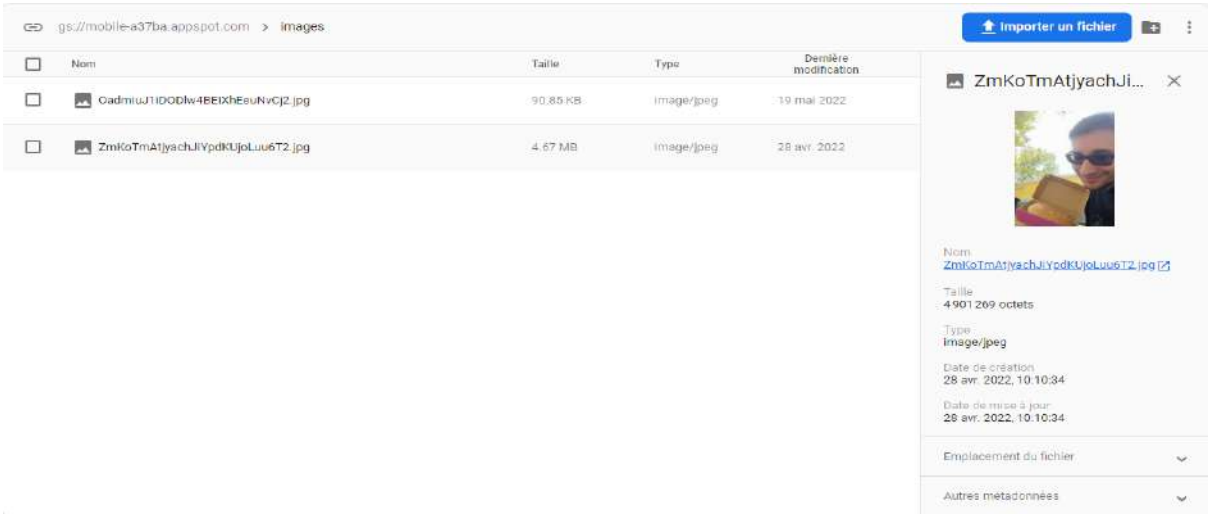
Architecture BDD de rooms

- **Users**, qui permet de stocker les informations du joueur tel que le pseudo, elo, biographie, email et animations, indiquant si l'utilisateur désire afficher ou non le déplacement continu des pièces du plateau par exemple.

Architecture BDD de users

Ces trois classes contiennent un ID unique, l'uid du joueur, afin de ne pas écraser les données d'autres utilisateurs. La classe Rooms, quant à elle, est effacée à la fin de la partie de jeu (victoire, abandon ou égalité). Elle ne sert qu'à donner les informations pour la partie en cours. Dans notre modèle tel qu'il a été pensé, un joueur peut ne créer qu'une seule room à la fois. C'est à dire que s'il quitte sa partie alors qu'elle n'est pas finie et qu'il crée une nouvelle room, l'ancienne sera alors effacée.

Pour la gestion de la photo de profil de chaque joueur, on utilise à nouveau Firebase. Cependant, on utilise un module de stockage différent, appelé "Storage", qui nous permet d'enregistrer des fichiers. Ici donc, quand l'utilisateur voudra changer de photo de profil, il va devoir l'envoyer à notre serveur qui la nommera par l'uid du joueur connecté, afin de n'avoir qu'une seule photo stockée par joueur.



a

1.3. Exemple d’utilisation

Voici un diagramme de séquences pour la gestion de notre système d’un coup de jeu.

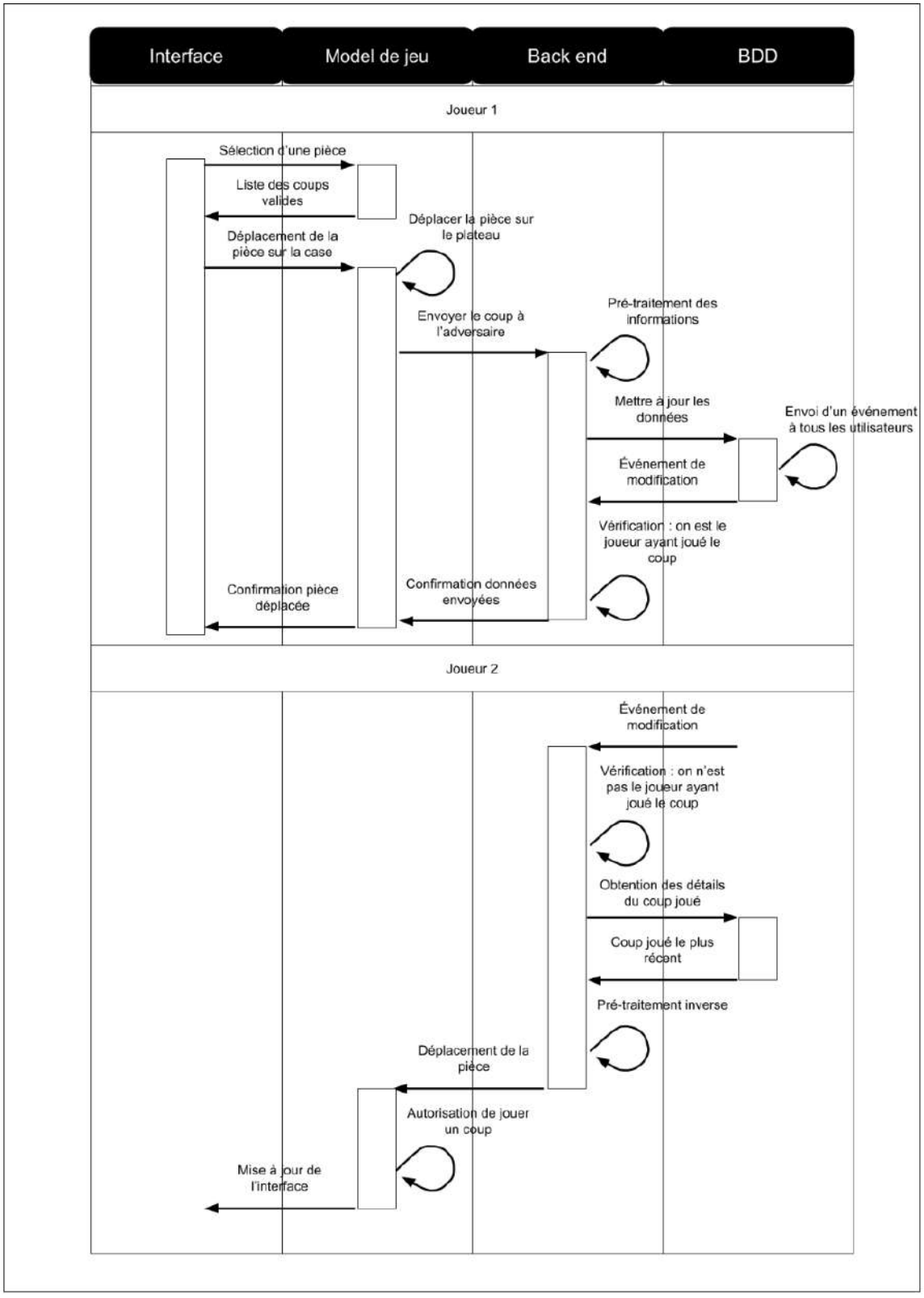


Diagramme de séquence d'un coup de jeu

Dans un premier temps, il y a une vérification côté modèle de jeu d'un coup de jeu jugé comme valide par le système (voir la partie modèle de jeu), puis ce coup est envoyé à la BDD dans la bonne room, modélisant une instance de partie en ligne côté Firebase. Chaque client possède son propre listener de modification de données sur la partie de jeu

en cours (la room). Les deux adversaires sont liés à la même partie, référencée par un identifiant unique correspondant à l'identifiant du joueur qui la construit.

Dépendant de la valeur "tour" que va lire le listener dans la room, il va pouvoir interpréter s'il s'agit d'un coup qu'il doit faire exécuter au modèle de son client ou non. Cette valeur est mise à jour à chaque nouvel envoi d'un coup d'un client à la BDD, permettant une gestion coup par coup synchrone des actions des utilisateurs sur la partie de jeu (on attend que l'adversaire joue pour retranscrire ce coup sur notre modèle de jeu et opérer le calcul de nos prochain mouvements de jeu).

L'action que l'on va opérer en lisant un coup adverse depuis la room dans la BDD est supposée valide, car elle est issue du modèle du joueur adverse, possédant une instance à jour de l'état de la partie de jeu en ligne. Si un joueur ne peut pas jouer de coup d'après son modèle, il se met en attente d'un coup d'un joueur adverse, pour le retranscrire et lancer une nouvelle boucle de jeu (voir partie modèle de jeu).

De même, un autre listener qui est basé sur loose dans la rooms, permet à signaler à l'autre joueur quand un des joueurs abandonne la partie. Donc en reprenant notre architecture, le joueur va alors recevoir la notification, supprimer la rooms, les listeners et va écrire dans History les informations nécessaires pour l'historique de la partie.

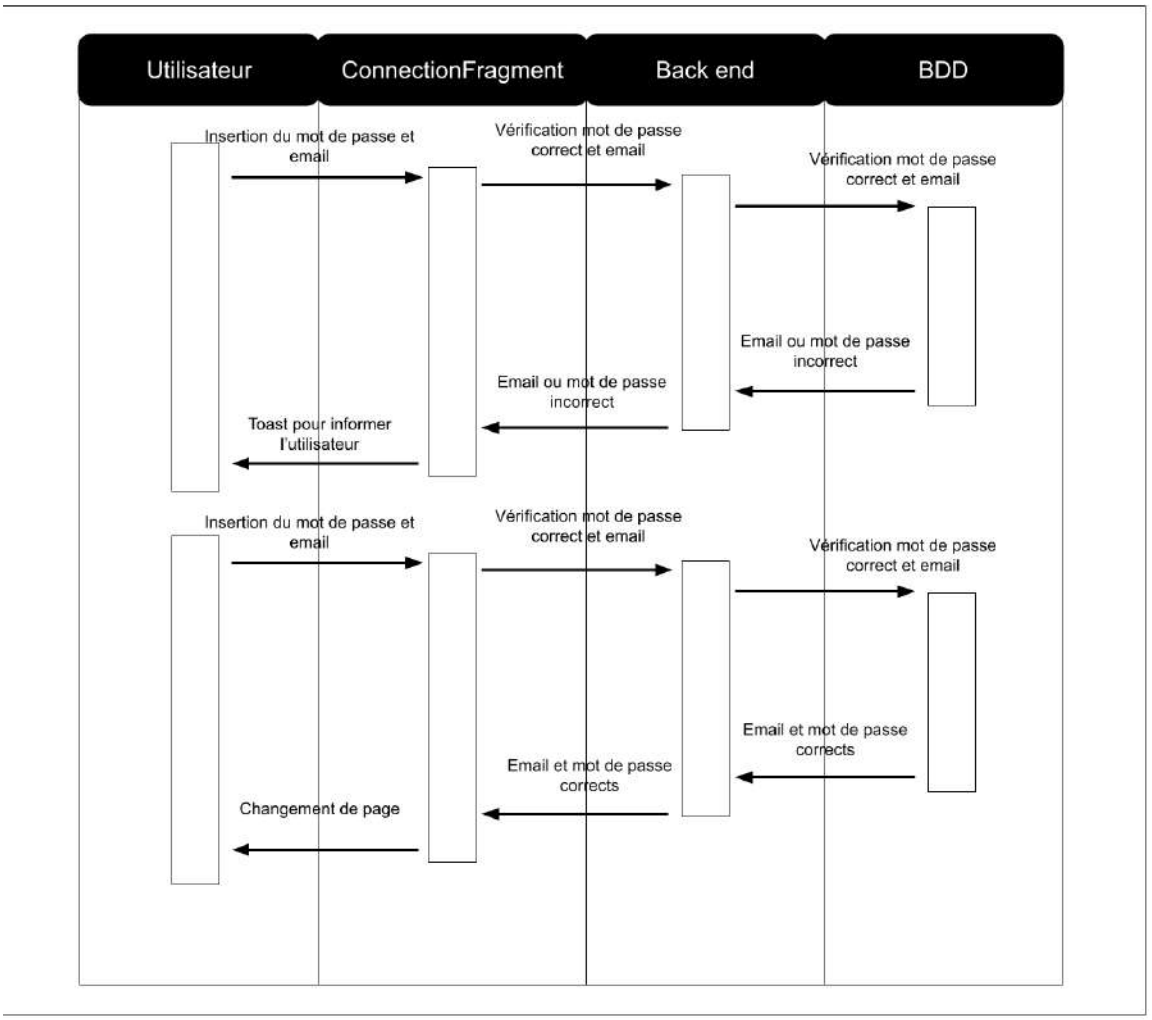


Diagramme de séquence de connexion d'un utilisateur

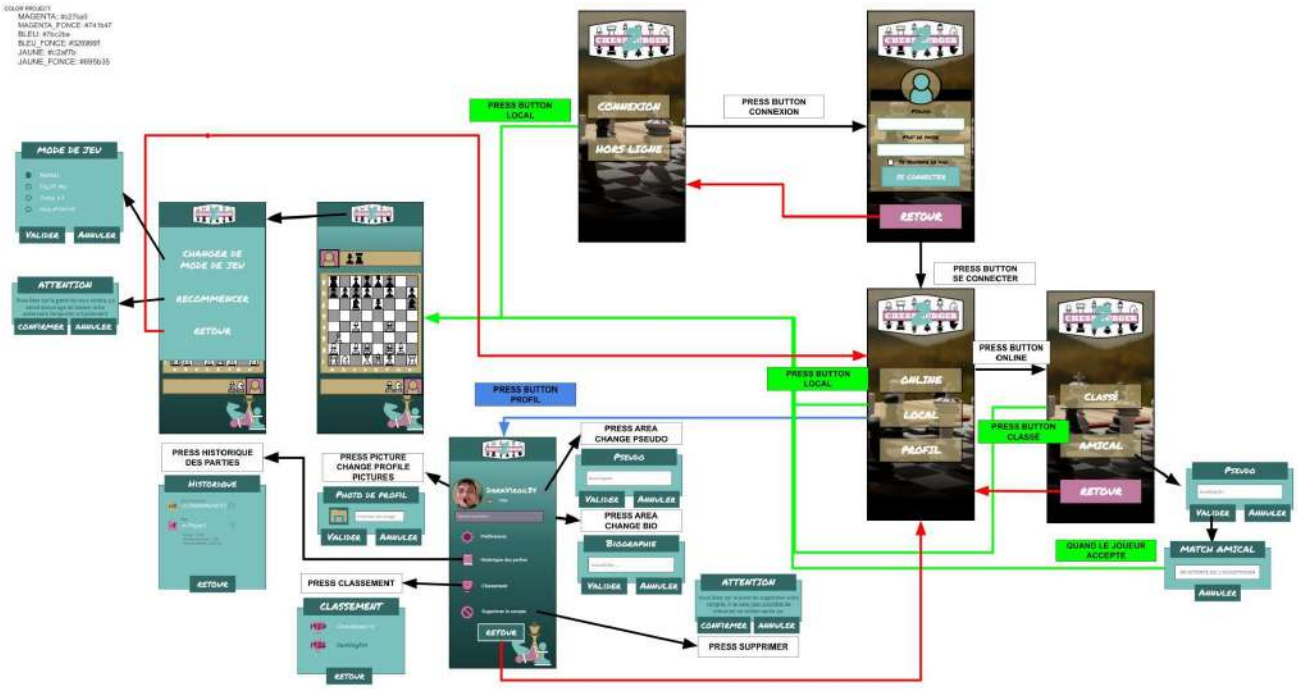
Un second diagramme de séquence intéressant à voir est celui de la connexion. On se rend compte que lorsque l'utilisateur va se connecter, l'application va directement faire appel à Firebase pour vérifier si l'utilisateur avec les informations sont correctes, alors la BDD va retourner un succès que l'on va récupérer pour lancer la nouvelle page pour cet utilisateur, sinon elle nous retourne une erreur. Quand une erreur est retourné alors nous la traitons pour informer l'utilisateur par le biais d'un Toast que ses informations sont incorrectes

2. Front-end

La gestion et manipulation des données ne sont rien dans une application mobile ou web sans leur affichage côté utilisateur. Ainsi, les parties back et front d'une application sont liées et doivent être prises en compte ensemble dans le processus de développement. On doit donc produire l'affichage et la gestion d'interactions utilisateurs en fonction de l'idée originelle, en l'occurrence un jeu d'échecs en ligne dans notre cas.

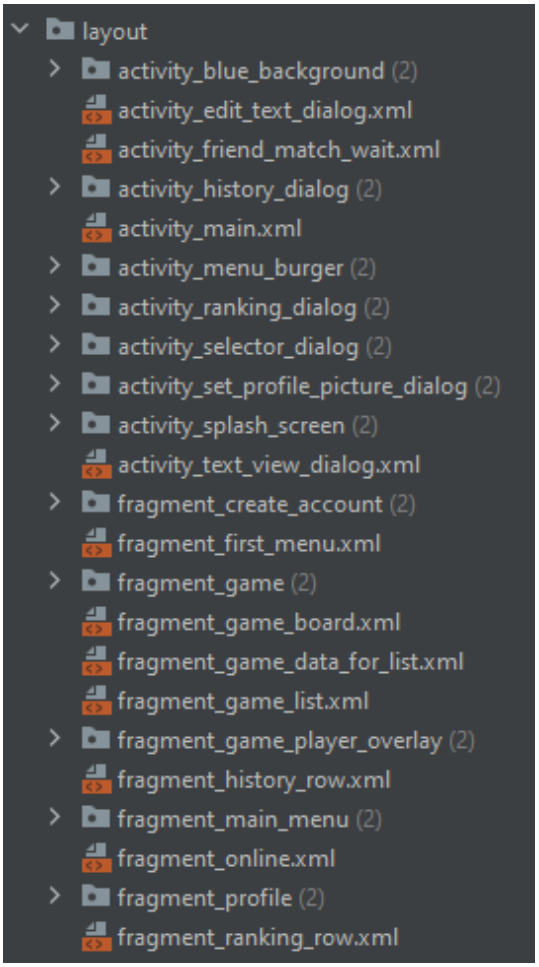
2.1. Programmation mobile

Le but étant tout de même de concevoir une application mobile, nous avons élaboré une maquette des interfaces et des interactions possibles au sein de celle-ci. Cela nous a permis de développer efficacement tous les fragments de jeu et de gestion d'utilisateur par exemple.



Maquette interface application mobile

À partir de ce schéma, nous avons implémenté une liste de layouts, dont en voici un extrait :



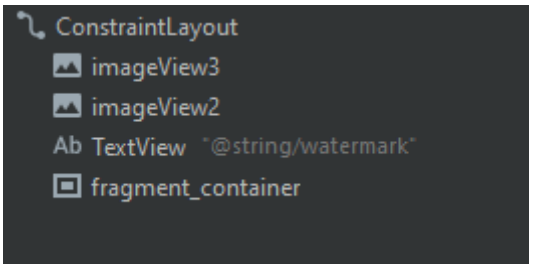
Liste des layouts dans la hiérarchie du projet

2.1.1. Explication de différents types de menus

Sur la maquette, on peut distinguer trois types de menus. Il y a premièrement les écrans ayant comme fond un rendu que nous avons fait sur Blender en modélisant des pièces d'échecs, en leur appliquant une texture “effet bois” et en les disposant dans une scène 3D.

Rendu 3D d'un plateau d'échec

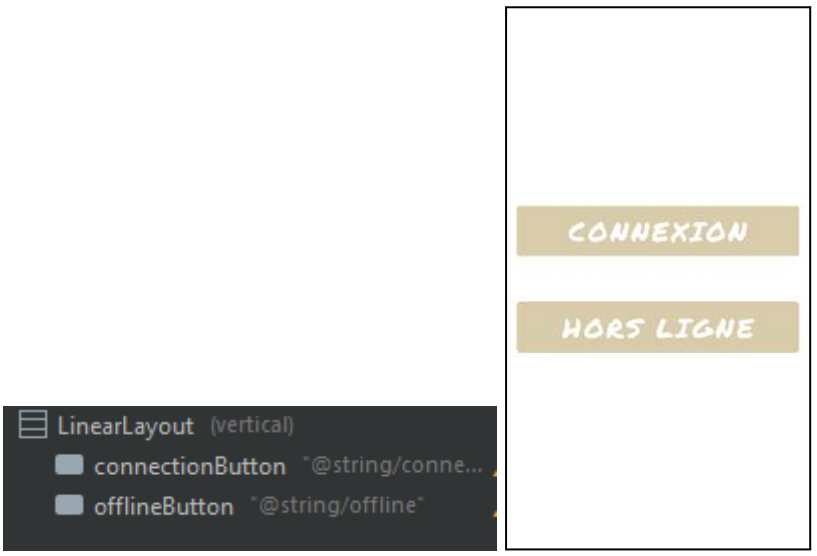
Comme ces menus possèdent la même image de fond, et afin d'éviter la duplication de code, nous avons mis en place un système de fragments. On sépare la vue de fond d'un fragment qui va s'afficher au-dessus de ce fond. Ce fragment peut s'afficher sur la totalité de l'écran.



Hiérarchie et pré-rendu du fichier XML appelé "Activity_main"

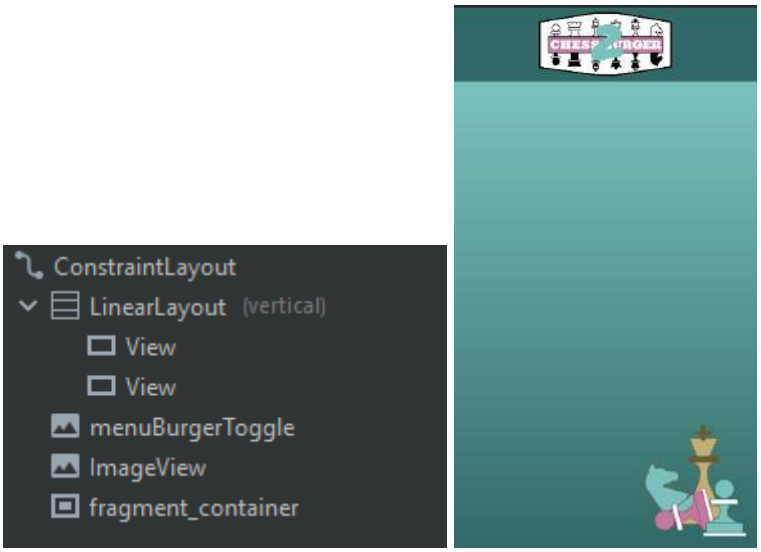
Ce fichier contient le fond de ce groupe de menu ainsi qu'un conteneur de fragment. L'idée de ces menus est donc de charger ce même fichier XML, puis de charger le fragment qui leur correspond. Prenons par exemple le premier menu, celui composé des boutons

“Connexion” et “Hors ligne”, appelé “MainActivity”. Cette activité va charger le fichier XML “Activity_main”. Elle va ensuite créer le fragment “FirstMenuFragment” et l’ajouter au conteneur de fragments.



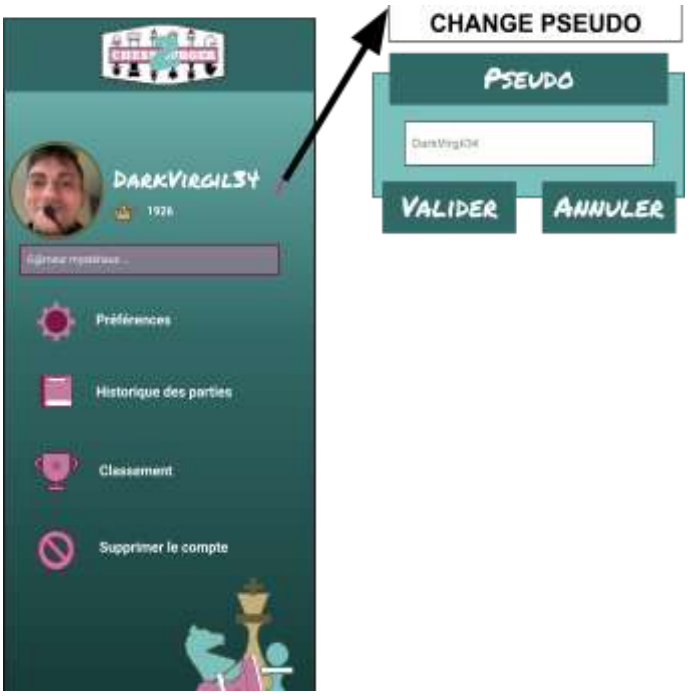
XML “fragment_first_menu” utilisé par “FirstMenuFragment”

Similairement, il y a les menus sur fond bleu. Ils utilisent le même principe : on crée une activité avec comme fond un fichier XML unique et on ajoute le fragment correspondant. Sur cette vue, le fragment ne peut pas s’afficher sur la bannière contenant le logo.



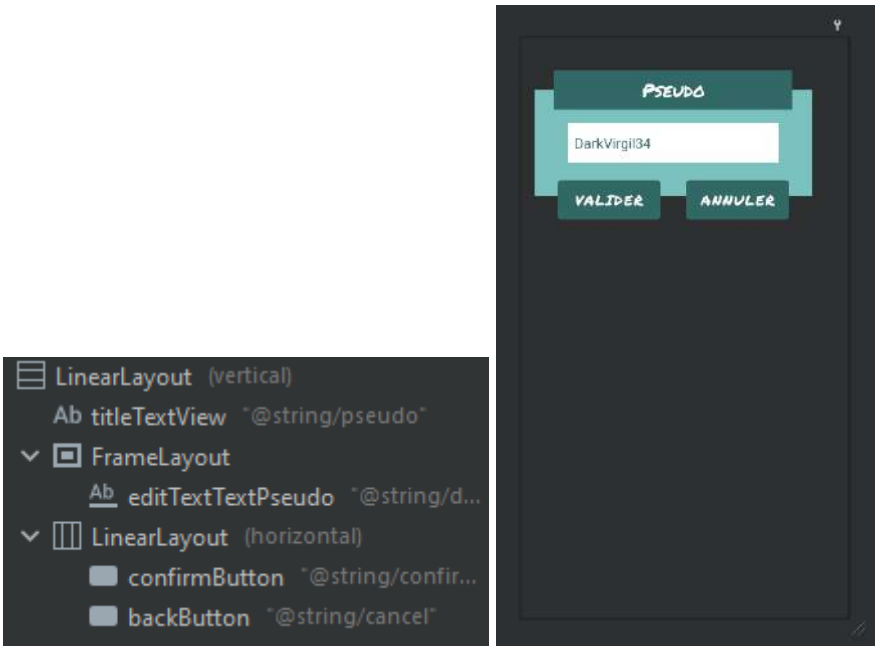
XML “activity_blue_background”

Enfin, il y a quelques activités que l’on montre à l’utilisateur au-dessus de l’interface courante : les “dialogs”. Ce sont des fenêtres flottantes, plus petites que la taille de l’écran, et qui ont généralement moins d’informations ou d'interactions qu’une véritable fenêtre. Ces “dialogs” sont, dans le code, gérés comme des activités. Cependant, elles renvoient un résultat. Les activités qui les lancent doivent donc attendre la réception de ce résultat. Prenons par exemple le cas où l’utilisateur, sur sa page de profil, souhaite changer son pseudo. On modélise donc cette interaction :



L'utilisateur souhaite modifier son pseudo

On a donc l'activité "ProfileActivity" qui possède le bouton pour changer de pseudo. Elle possède aussi un lanceur d'activité ("ActivityResultLauncher") qui permet de lancer une activité et de récupérer la valeur de retour. Lorsque l'utilisateur appuie sur ce bouton, cette activité va donc utiliser son lanceur pour afficher le "dialog" appelé "EditTextDialogActivity", en précisant le titre à afficher et le texte d'aide à la saisie.



XML "activity_edit_text_dialog" utilisé par "EditTextDialogActivity"

À partir d'ici plusieurs scénarios sont possibles. Soit l'utilisateur valide son entrée et on change le pseudo, soit il annule et on ne fait rien. Ce comportement est défini dans le

lanceur utilisé, donc dans l'activité de profil. Du point de vue du "dialog", si l'utilisateur valide l'action, alors on met dans l'Intent de résultat le texte saisi ainsi que la valeur de résultat "OK", sinon, on met la valeur de résultat "CANCELED". Le lanceur va donc récupérer ces résultats et effectuer le traitement adapté.

2.1.2. Thèmes, styles et dimensions

Certaines conceptions de vues contiennent des données redondantes. Afin de limiter la duplication de code (principe **DRY** : Don't Repeat Yourself) et de pouvoir accéder à des données communes, android studio propose différents systèmes regroupant différentes informations.

2.1.2.1. Thèmes

Les thèmes permettent de définir des couleurs ainsi que des paramètres d'affichage en fonction des activités. On a donc défini un thème principal nous permettant d'accéder aux couleurs souvent utilisées et donc de pouvoir les changer facilement.

Ce thème hérite d'un thème de la convention "Material Design". Cela va modifier tous nos boutons pour qu'ils suivent cette convention. En premier lieu, on définit les couleurs principales, leurs variantes et la couleur du texte (s'il se situe sur la couleur). Comme ce thème correspond à la catégorie de menus ayant comme fond le rendu 3D, on définit les couleurs de la barre de navigation en haut et en bas comme étant jaune et noir. Enfin, on définit des valeurs pour afficher le plateau de jeu.

```
<style name="Theme.ProjetMobile"
parent="Theme.MaterialComponents.DayNight.NoActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/alpha_yellow</item>
    <item name="colorPrimaryVariant">@color/yellow</item>
    <item name="colorPrimaryVariantDark">@color/dark_yellow</item>
    <item name="colorOnPrimary">@color/white</item>
    <item name="colorOnPrimaryVariant">@color/black</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/blue</item>
    <item name="colorSecondaryVariant">@color/dark_blue</item>
    <item name="colorOnSecondary">@color/dark_blue</item>
    <item name="colorOnSecondaryVariant">@color/white</item>

    <item name="colorTertiary">@color/magenta</item>
    <item name="colorTertiaryVariant">@color/dark_magenta</item>
    <item name="colorOnTertiary">@color/white</item>

    <item name="fontFamilyVariant">@font/permanent_marker</item>

    <!-- Status bar color. -->
```

```
    <item name="android:statusBarColor"
tools:targetApi="l">?attr/colorPrimary</item>
    <item name="android:navigationBarColor">@color/black</item>

    <!-- For the color theme game -->
    <item name="white_case_color">?attr/colorSecondary</item>
    <item name="black_case_color">?attr/colorSecondaryVariant</item>
    <item name="selection_case_color">?attr/colorPrimaryVariant</item>
    <item name="eat_case_color">?attr/colorTertiaryVariant</item>
    <item
name="confirmation_case_color">?attr/colorPrimaryVariantDark</item>
    <item name="menaced_case_color">?attr/colorTertiary</item>
    <item name="rock_case_color">@color/light_green</item>


    <item name="GameMainColor">?attr/colorPrimaryVariant</item>
    <item name="GameSecondColor">?attr/colorPrimaryVariantDark</item>
    <item name="GameColorPlayerTurn">?attr/colorOnPrimaryVariant</item>
    <item name="colorTextIndicatorBoard">?attr/colorOnPrimary</item>

    <item name="changePiece_bgColorScreen">@color/alpha_black</item>
    <item name="changePiece_textColor">@color/white</item>

    <item name="colorBorderTopFinishScreen">?attr/colorTertiary</item>
    <item
name="colorBorderBottomFinishScreen">?attr/colorTertiaryVariant</item>
    <item
name="colorTextTopBottomFinishScreen">?attr/colorSecondary</item>
    <item name="colorTextFinishScreen">?attr/colorPrimaryVariant</item>
    <item name="colorBGFinishScreen">@color/alpha_black</item>


    <item name="colorBGStartScreen">@color/blue</item>
    <item name="android:windowDisablePreview">true</item>
</style>
```

La plupart des attributs ont été définis arbitrairement dans le fichier “attr.xml”, et correspondent à des besoins de spécification d'options de rendus interfaces. Suivant ce même principe, on définit les couleurs dans le fichier “colors.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
```

```
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>

    <color name="magenta">#C27BA0</color>
    <color name="dark_magenta">#741B47</color>
    <color name="blue">#7BC2BE</color>
    <color name="dark_blue">#326966</color>
    <color name="yellow">#C2AF7B</color>
    <color name="alpha_yellow">#A4C2AF7B</color>
    <color name="dark_yellow">#695b35</color>
    <color name="brown">#3d351e</color>
    <color name="black">#FF000000</color>
    <color name="alpha_black">#88000000</color>
    <color name="white">#FFFFFFFF</color>

    <color name="red">#FF0000</color>
    <color name="light_green">#A6F1A6</color>
</resources>
```

On définit également un sous-thème pour les activités sur fond bleu. On va par exemple changer la couleur des barres de navigation.

```
<style name="AppBlueActivityTheme" parent="Theme.ProjetMobile">
    <item name="android:statusBarColor"
tools:targetApi="l">?attr/colorSecondaryVariant</item>
    <item
name="android:navigationBarColor">?attr/colorSecondaryVariant</item>
</style>
```

Dans le fichier “AndroidManifest.xml”, il faut donner à ces activités le bon thème correspondant. Par exemple, pour l’activité de profil, on va lui donner le thème bleu :

```
<activity
    android:name=".ProfileActivity"
    android:exported="false"
    android:theme="@style/AppBlueActivityTheme" />
```

On définit également le thème pour les fenêtres de “dialog”. Ce thème permet d’afficher les activités par-dessus l’activité en cours. Pour cela, on l’indique comme étant flottante et ayant un fond semi-transparent, permettant de visualiser l’activité courante et de ne pas perdre l'utilisateur, tout en détachant la fenêtre de dialogue afin de la mettre en valeur et indiquer à l’utilisateur plus efficacement l'action qu’il est sur le point d’entreprendre (accompagnement par le design de l’utilisateur dans le processus d’interaction avec notre application).

```
<style name="AppDialogTheme" parent="AppBlueActivityTheme">
   <item name="android:windowFrame">@null</item>
   <item name="android:windowIsFloating">true</item>
   <item name="windowActionBar">false</item>
   <item name="android:windowNoTitle">true</item>

   <item name="android:background">@android:color/transparent</item>
   <item
name="android:windowBackground">@android:color/transparent</item>
   <item name="android:colorBackgroundCacheHint">@null</item>
   <item name="android:windowIsTranslucent">true</item>
</style>
```

2.1.2.2. Styles

Au-delà des thèmes, on dispose de la possibilité de définir des styles. Les styles renseignent les méthodes d'affichage des éléments des vues XML. Prenons par exemple les boutons du premier menu. Premièrement, regardons le fichier XML du menu. Il est constitué d'un layout linéaire qui va afficher le contenu dans l'axe vertical et de manière centrée. Ce layout contient deux boutons ayant le style "btn_menu_default". Le premier possède une marge afin de les séparer sur l'interface. Le style permet de mettre en un seul emplacement les attributs communs, et ainsi généraliser le style d'un élément d'interface pour pouvoir le dupliquer plus facilement sur l'ensemble de l'application. Outre une meilleure réutilisabilité, cela inclut une meilleure cohérence visuelle. En effet, cela normalise les éléments importants de design, favorisant une meilleure prise en main par celui-ci de notre application. Cet élément peut paraître anodin, mais devient important quand on cherche à développer une application intuitive offrant un bon confort d'utilisation, essentiel pour ce genre de services répandu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".FirstMenuFragment"
   android:orientation="vertical"
   android:padding="@dimen/fab_margin"
   android:gravity="center">

   <Button
       android:id="@+id/connectionButton"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginBottom="50dp"
```

```
        style="@style/btn_menu_default"
        android:text="@string/connection" />

    <Button
        android:id="@+id/offlineButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/btn_menu_default"
        android:text="@string/offline" />

</LinearLayout>
```

Le style contient donc les attributs en commun entre les boutons. On définit la police, la taille et la couleur du texte ainsi que l'espacement intérieur. On hérite du style de la convention Material Design "UnelevatedButton", car on ne veut pas avoir d'ombres, un peu à la manière de "flat button". En effet, le fond est transparent, donc l'effet d'ombre ne fonctionne pas correctement.

```
<style name="btn_menu_default"
parent="Widget.MaterialComponents.Button.UnelevatedButton">
    <item name="android:fontFamily">?attr/fontFamilyVariant</item>
    <item name="android:textSize">@dimen/txt_size_btn_menu</item>
    <item name="android:textColor">?attr/colorOnPrimary</item>
    <item name="android:padding">@dimen/padding_btn_menu</item>
</style>
```

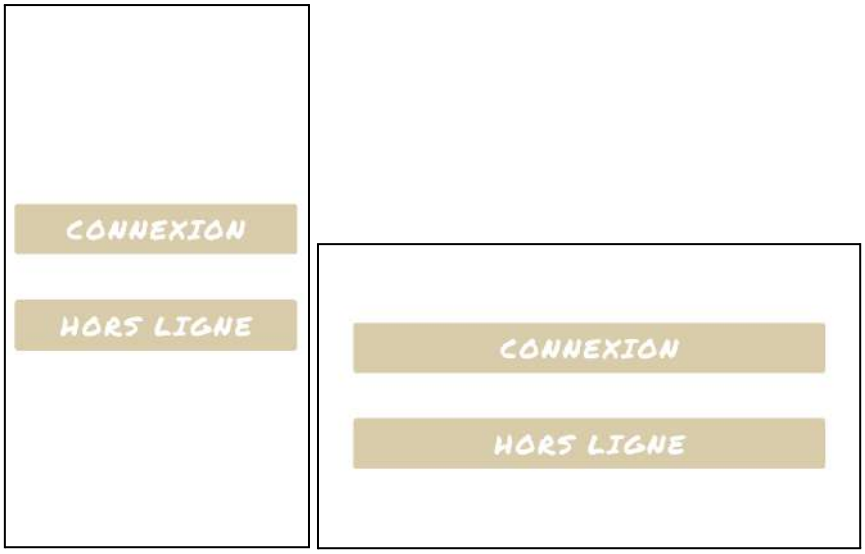
2.1.2.3. Dimensions

Vous avez peut-être remarqué que l'on ne donne pas directement les dimensions. On utilise à la place des variables définies dans le fichier "dimension.xml". Cela nous permet de définir des valeurs de dimensions comme on peut le faire pour les couleurs. Ces dimensions sont ensuite utilisées par les fichiers XML, les styles et dans le code. Cela nous permet de modifier plusieurs valeurs rapidement, d'être plus clair sur les valeurs et d'éviter la duplication de code (normalisation de l'interface). De plus, on a la possibilité de créer un fichier de dimensions différentes en fonction de la taille d'affichage, et donc d'avoir une application adaptative à la taille de l'écran, offrant une meilleure ergonomie pour tous les types d'utilisateurs.

2.1.3. Ergonomie et adaptivité sur les différentes orientations

Une application mobile doit être pensée pour tout type d'appareils et tout type d'écrans. Nous avons donc exploré différents moyens de rendre les différentes activités plus adaptatives. Trois moyens pour adapter la taille d'une activité ont été trouvés ; sans passer par le code.

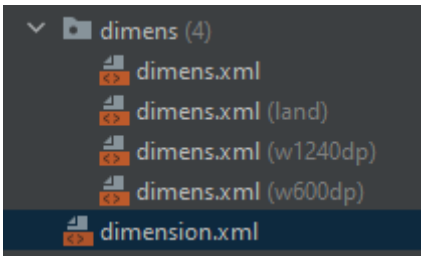
Premièrement, pour les activités simples, on peut concevoir le fichier XML de manière à ce qu'il fonctionne peu importe les dimensions de l'écran. Par exemple, si on reprend l'activité du premier menu, il est conçu pour s'adapter aux changements sans que l'on aie de choses à changer.



Pré-rendu du premier menu en mode vertical et horizontal

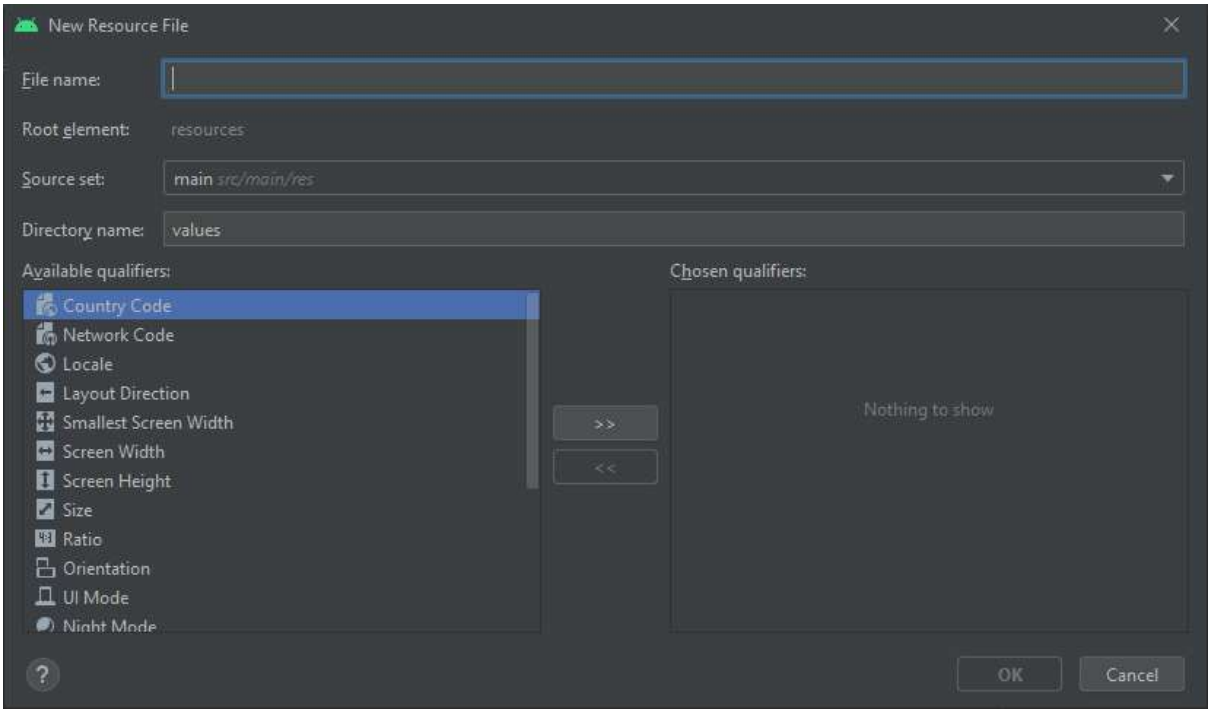
En vérité, dans ce menu, il y a une dimension qui change lorsqu'on passe en horizontal grâce à la deuxième technique présentée ci-dessous, mais c'est plus pour un côté esthétique que pratique.

Deuxièmement, on peut définir des dimensions différentes en fonction des tailles d'écran. En effet, c'est l'un des avantages d'avoir défini les valeurs dans le fichier de dimension.



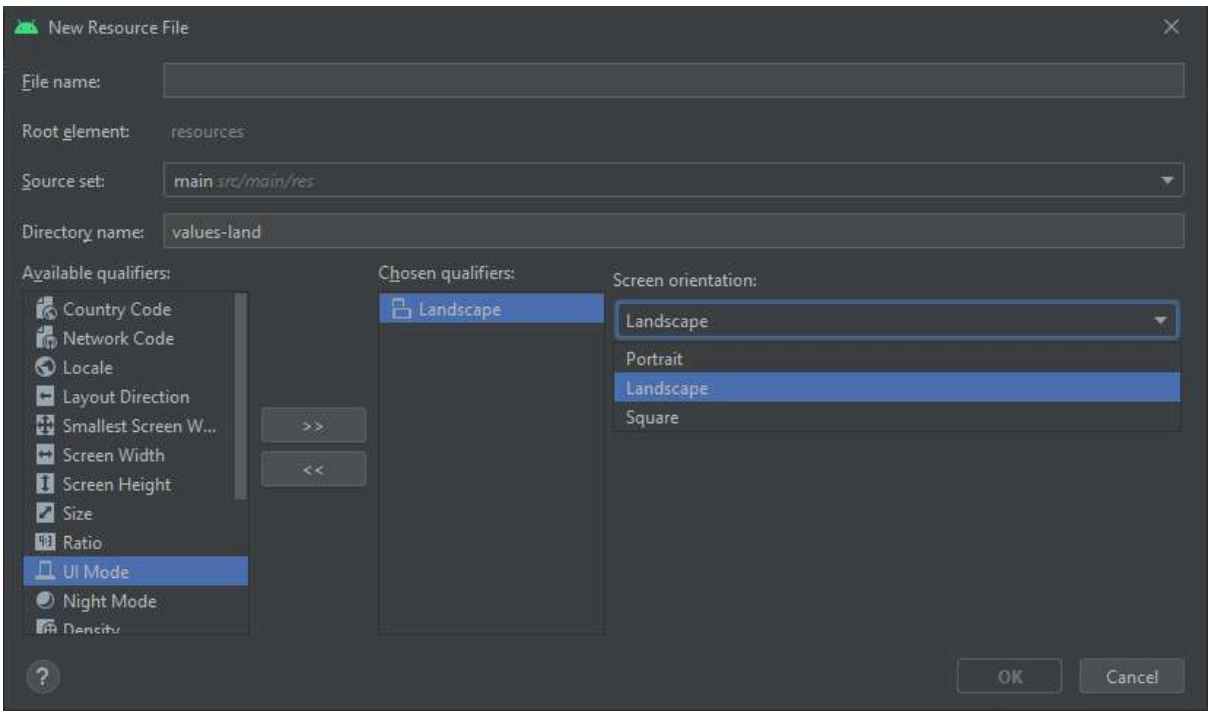
Hiérarchie des différentes dimensions

Ils peuvent être créés à l'aide d'Android studio, en faisant un clic droit → "new" → "value resource file", et en choisissant les bons qualificateurs.



Fenêtre d'android studio pour la création d'un fichier de valeur

On peut par exemple prendre un qualificateur sur la taille de l'écran, sur l'orientation de l'écran, etc.
Pour prendre en compte l'orientation de l'appareil, on crée un fichier avec comme qualificateur l'orientation, et on choisit l'orientation horizontale appelée "landscape".



Fenêtre d'android studio pour la création d'un fichier de valeur sur l'orientation

Ainsi, si l'on redéfinit une valeur dans ce fichier et que le que l'on est dans un format paysage, alors la valeur du fichier est appliquée.

Faisons par exemple le suivi de la dimension “dialog_width”. Cette dimension est utilisée pour les “dialogs”. Elle spécifie la largeur que doit prendre l’activité. En effet, comme ces activités sont flottantes, leurs largeurs ne dépendent pas de la taille de l’écran. On a donc spécifié, dans le fichier de dimensions, cette largeur.

```
<dimen name="dialog_width">350dp</dimen>
```

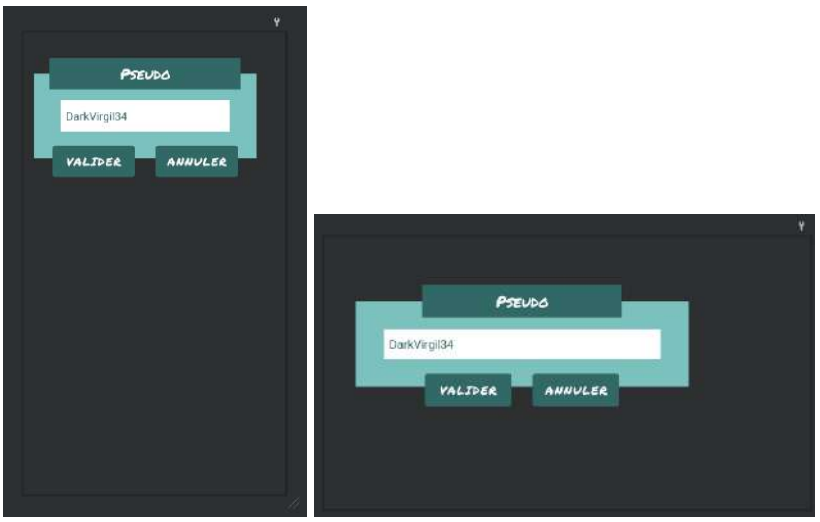
Cette valeur est également définie dans le fichier de dimensions qui détecte le format paysage :

```
<dimen name="dialog_width">500dp</dimen>
```

Elle est ensuite utilisée par les “dialogs”. Par exemple, dans le layout “activity_edit_text”, on l’applique en tant que largeur du “LinearLayout”.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="@dimen/dialog_width"
   android:layout_height="wrap_content"
   tools:context=".EditTextDialogActivity"
   android:layout_margin="@dimen/fab_margin"
   android:gravity="center|top"
   android:orientation="vertical">

    ...

</LinearLayout>
```

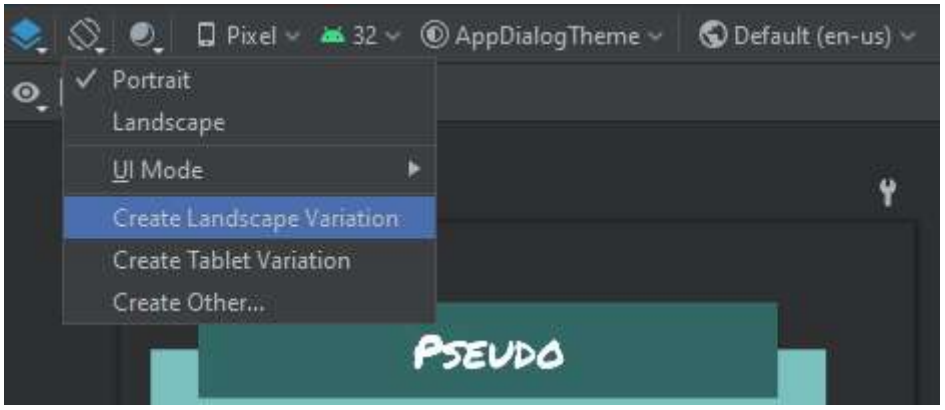
Ainsi, on peut s’adapter aux changements d’orientation en modifiant les différentes valeurs possibles pour un élément d'interface.



Pré-rendu du dialog d'édition de texte en mode vertical et horizontal

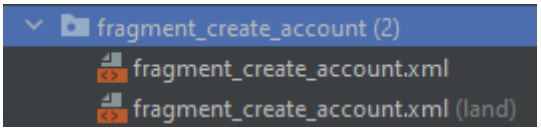
Cette technique est claire, mais, dans certains cas plus complexes, n'est pas suffisante. En effet, son utilisation nous limite à seulement changer des valeurs de taille. Cependant, dans certains cas, on souhaite également changer l'agencement des vues.

Troisièmement, on peut définir un nouveau layout en fonction de l'orientation. Pour cela, il suffit d'ouvrir la vue dans android studio et de créer une variation.



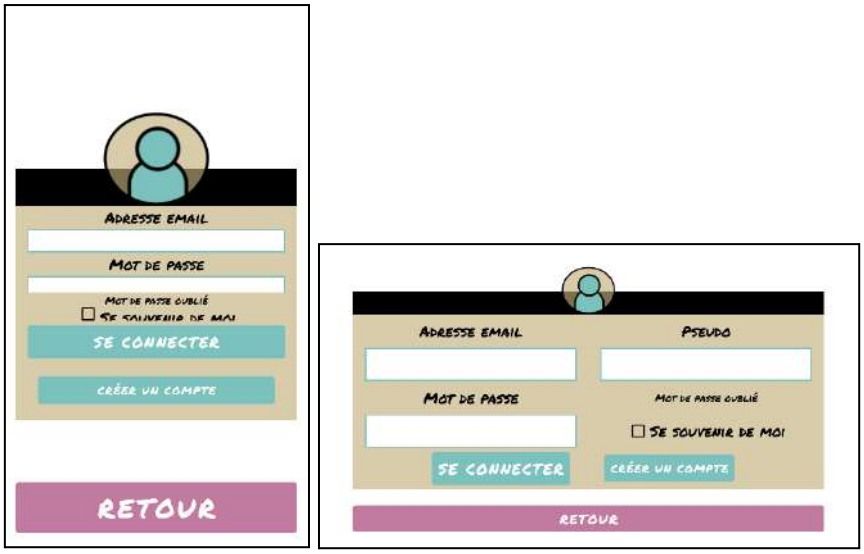
Création d'une variation de layout en fonction de l'orientation

Ainsi, le fichier XML chargé lors de la création de l'activité est choisi automatiquement. Cependant, il faut recréer un nouveau fichier, mais cela est tout de même utile pour les vues plus complexes. Par exemple, pour le menu de connexion, on a également défini un moyen de créer un compte grâce à un bouton "Créer un compte". Ce bouton va montrer davantage de champs afin que l'on puisse créer le compte. Cependant, pour la version en format paysage, on peut plus facilement montrer ces champs. En effet, on peut les afficher sur le côté, prenant l'avantage du format horizontal.
Ainsi, deux fichiers XML sont créés.



Hiérarchie des fichiers XML du menu de connection

Les deux fichiers sont automatiquement organisés dans un dossier à l'intérieur de la racine des layouts. Dans le deuxième fichier, on peut donc modifier et déplacer les éléments.

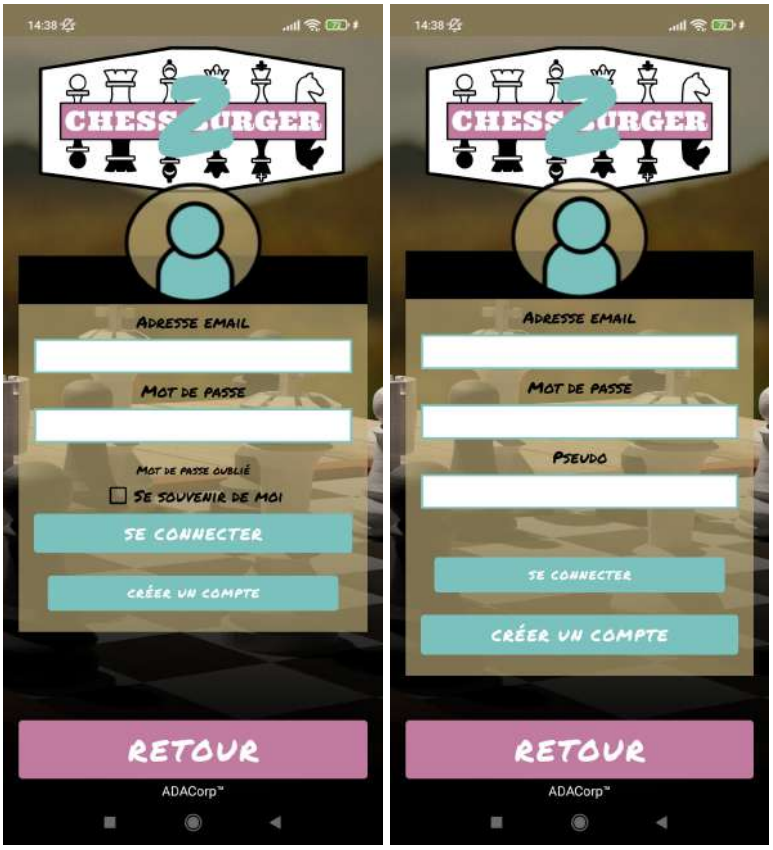


Pré-rendu de la vue “fragment_create_account” en mode vertical et horizontal

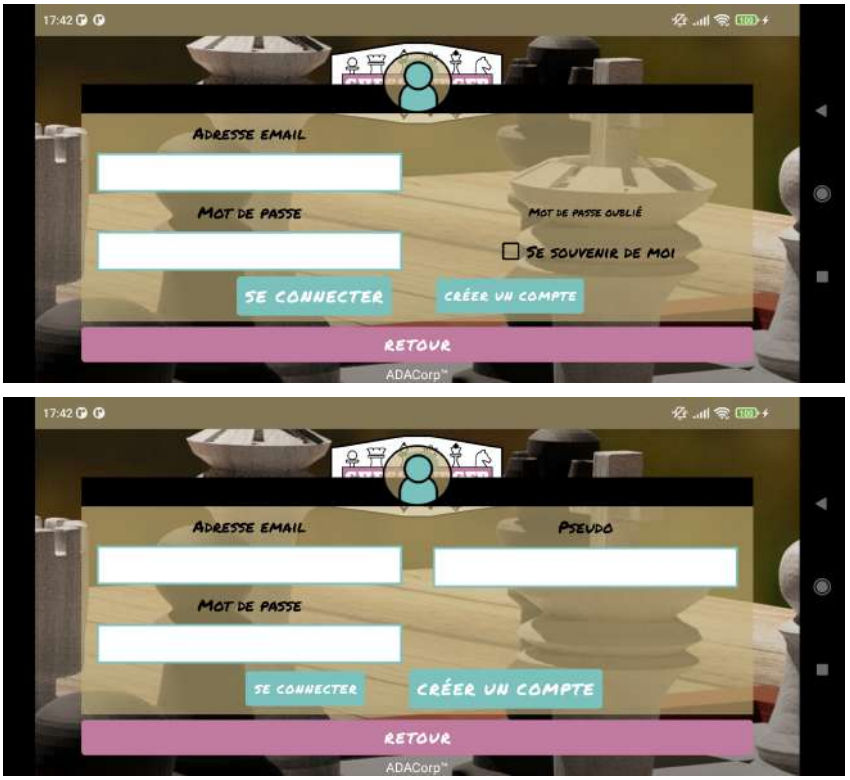
2.1.4. Détails des différents menus

Dans cette partie, des détails sur différentes fonctionnalités et leurs implémentations vont être donnés.

Premièrement, certains menus présentent des animations. Par exemple, le menu de connexion présent ci-dessus va en réalité étendre son menu, en fonction que l’on soit dans l’état de connexion ou de création de compte (un même menu qui gère les deux comportements). Cela va induire le fait que l’on ne va pas changer d’activité entre la connexion et la création de compte par exemple. On limite ainsi l’effort de compréhension de l’utilisateur, qui voit dynamiquement le choix d’interaction voulu se proposer à lui. Il pourra ainsi assimiler ces deux actions comme des actions relativement similaires, ce qui est le cas, car elles permettent un accès à un profil pour un utilisateur, qu’il soit nouveau sur l’application ou non.



Capture du menu de connexion en mode “connexion” et "création de compte” en vertical



Capture du menu de connexion en mode “connexion” et "création de compte” en horizontal

Si l'on est dans l'état de connexion et que l'on appuie sur le bouton "Créer un compte", alors on doit montrer le nouveau champ "Pseudo" qui, comme son nom l'indique, permet de modifier son pseudo. On doit également cacher le bouton "Mot de passe oublié" et "Se souvenir de moi", qui n'ont plus de sens ici. Inversement, si l'on est dans l'état de création de compte et que l'on appuie sur le bouton "Se connecter", alors on cache le champ "Pseudo" et on affiche les options "Mot de passe oublié" et "Se souvenir de moi".
Pour cela, on passe par des animations. Par exemple, pour cacher les options "Mot de passe oublié" et "Se souvenir de moi", on doit tout d'abord récupérer le "LinearLayout" parent. Ensuite, on lui donne une animation afin de modifier sa position et sa transparence.

Pour le cacher, on le déplace vers le bas, on le met transparent, on lui attribue une durée d'apparition. On peut même ajouter une fonction callback à exécuter lorsqu'il à fini de s'animer.

```
ll_options.animate()
        .translationY(ll_options.getHeight())
        .alpha(0.0f)
        .setDuration(animation_create_account_duration)
        .setListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
            }
        });
```

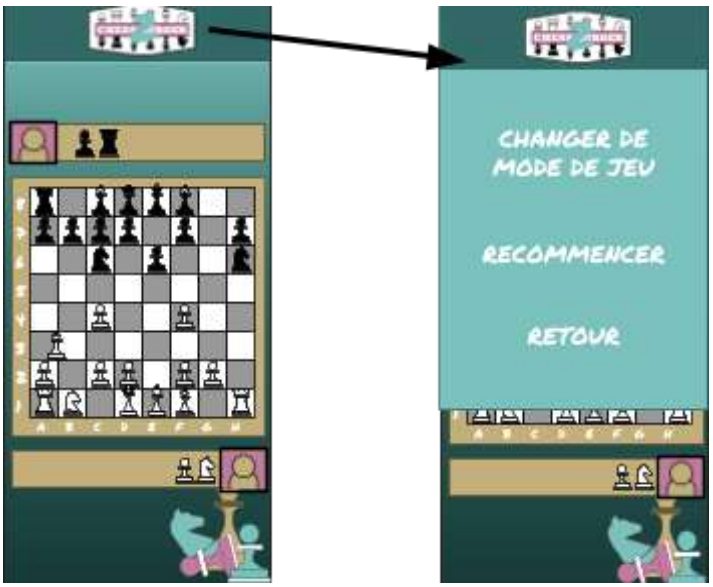
Pour le montrer, on doit faire les actions inverses, donc remettre l'attribut de déplacement à 0 et la transparence à 1.

```
ll_options.animate()
        .translationY(0)
        .alpha(1.0f)
        .setDuration(animation_create_account_duration);
```

Bien sûr, on ne doit pas effectuer exactement la même animation si l'utilisateur est au format paysage, on doit donc détecter l'orientation du téléphone. On peut obtenir cette information en cherchant dans les configurations des ressources.

```
boolean land = getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE;
```

Durant une partie de jeu, on a la possibilité d'ouvrir un menu burger en appuyant sur le logo.



Ouverture du menu burger

Le menu burger se déroule du haut vers le bas et se comporte comme un “dialog”, mais doit intégrer une animation lors de son lancement. On lui affecte donc un nouveau thème appelé “MenuBurgerTheme”. Un menu burger est un élément d’interface répandu et assimilé par bon nombre d'utilisateurs. Ils pourront ainsi comprendre rapidement notre design, et ne pas abandonner la navigation dans notre application, car trop complexe à assimiler (trop éloignée des standards de designs).

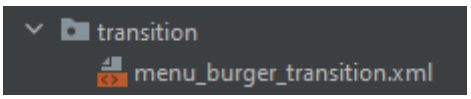
```
<activity
    android:name=".MenuBurgerActivity"
    android:exported="false"
    android:theme="@style/MenuBurgerTheme" />
```

Ce thème n’hérite pas du thème des “dialogs”, car certains comportements ne doivent pas être définis. On affecte donc une transition d’entrée et de sortie ainsi qu’un fond transparent.

```
<style name="MenuBurgerTheme" parent="AppBlueActivityTheme">
    <item name="android:windowActivityTransitions">true</item>
    <item
name="android:windowEnterTransition">@transition/menu_burger_transition<
/item>
    <item
name="android:windowExitTransition">@transition/menu_burger_transition</
item>

    <item
name="android:navigationBarColor">@android:color/transparent</item>
    <item name="android:statusBarColor"
tools:targetApi="l">@android:color/transparent</item>
```

```
    <item name="android:background">@android:color/transparent</item>
    <item name="android:windowBackground">@color/alpha_black</item>
    <item name="android:colorBackgroundCacheHint">@null</item>
    <item name="android:windowIsTranslucent">true</item>
</style>
```

On affecte au menu burger la transition appelée “menu_burger_transition”. Cette transition est définie dans le dossier de transition situé dans la racine des ressources.

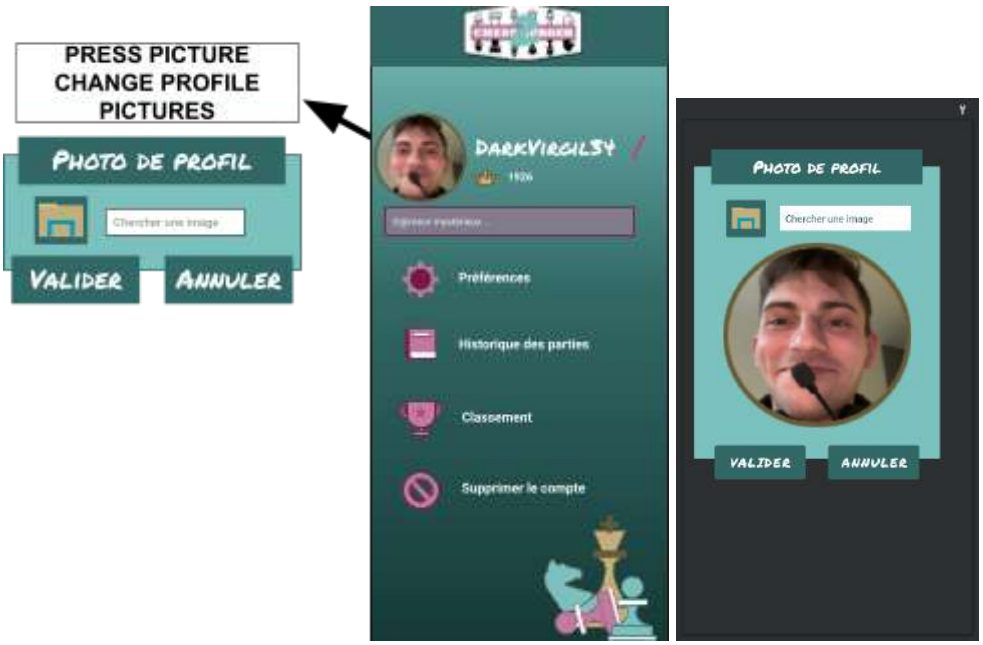


Emplacement du fichier de transition.

La transition “menu_burger_transition” définit simplement une transition à partir du haut de l’écran.

```
<?xml version="1.0" encoding="utf-8"?>
<transitionSet
xmlns:android="http://schemas.android.com/apk/res/android">
    <slide android:slideEdge="top"/>
</transitionSet>
```

Parlons un peu du menu du choix d’image de profil. Lorsque l’utilisateur appuie sur son image dans le menu d’édition de profil, on ouvre un “dialog” de sélection d’images.

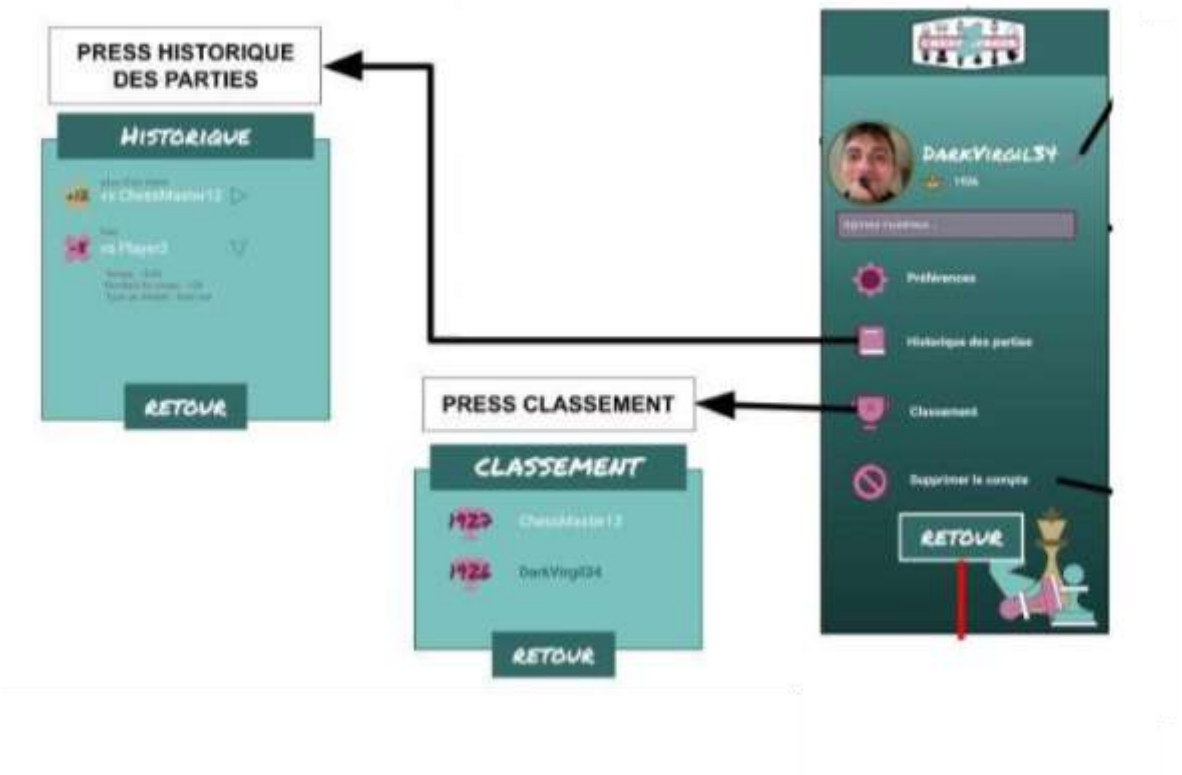


Conception de l’action de choix d’image et pré-rendu de la fenêtre de dialogue réellement créée.

Cette activité présente une prévisualisation de l'image choisie, deux boutons pour valider ou annuler et enfin un bouton pour lancer le choix d'images du téléphone. Pour cela, on doit demander au système de lancer une application de choix d'image et attendre le résultat.
On crée donc un lanceur qui précise le comportement à adapter lors du résultat et on affecte au bouton le bon comportement.

```
ActivityResultLauncher<String> getImage = registerForActivityResult(new
ActivityResultContracts.GetContent(),
       resultURI -> {
           if (resultURI != null) {
               result = resultURI;
               previewImageView.setImageURI(result);
           }
       });

ImageButton setFileButton = findViewById(R.id.setFileButton);
setFileButton.setOnClickListener(v -> {
   getImage.launch("image/*");
});
```

Il y a aussi d'autres "dialogs" ayant un comportement spécifique : les menus d'historique et de classement.



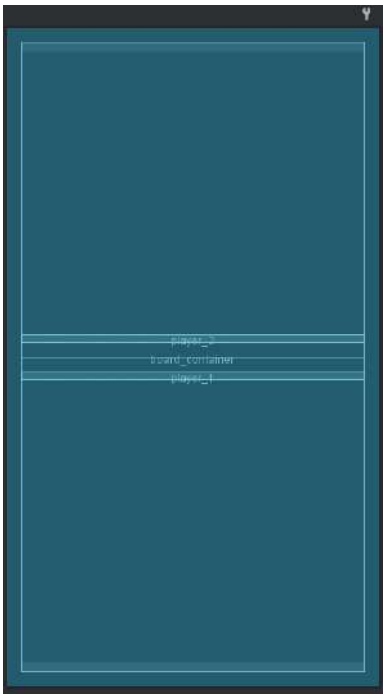
Conception des menus d'historique et de classement.

Ces menus présentent une liste horizontale de fragments contenue dans une "ScrollView", permettant à l'utilisateur de défiler la liste. Lors du lancement de ces activités, on va donc remplir cette liste en ajoutant des fragments représentant des lignes. Le choix de la "ScrollView" est d'autant plus pertinent que l'on ne connaît pas à l'avance le nombre de parties jouées par un utilisateur ou le nombre de joueurs dans la BDD.

Grâce à la modélisation du plateau que l'on précise dans une prochaine partie, on peut créer un layout contenant le plateau de jeu ainsi que les noms des joueurs. Le layout "fragment_game" affiche le plateau de jeu en utilisant des fragments pour mettre le plateau de jeu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".GameFragment"
    android:orientation="vertical"
    android:gravity="center"
```

```
    android:layout_margin="@dimen/fab_margin">

    <FrameLayout
        android:id="@+id/player_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/game_box"
        android:padding="@dimen/drawable_box_margin"
        android:layout_marginBottom="@dimen/common_margin" />

    <FrameLayout
        android:id="@+id/board_container"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="@dimen/common_margin"
        />

    <FrameLayout
        android:id="@+id/player_1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="@dimen/drawable_box_margin"
        android:background="@drawable/game_box"
        />

</LinearLayout>
```



Blueprint des fragments de joueurs et de plateau

Le fragment de plateau est défini via le fichier “fragment_game_player_overlay” qui utilise les vues définies dans “Model”.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:app="http://schemas.android.com/apk/res-auto"
       xmlns:tools="http://schemas.android.com/tools"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       tools:context=".GameBoardFragment">

   <com.example.projetmobile.Model.Board
       android:id="@+id/board_game"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:background="@drawable/game_box"
       android:paddingLeft="@dimen/board_margin_max"
       android:paddingTop="@dimen/board_margin_min"
       android:paddingRight="@dimen/board_margin_min"
       android:paddingBottom="@dimen/board_margin_max"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintTop_toTopOf="parent"
       app:nb_column="8"
       app:nb_row="8" />

   <com.example.projetmobile.Model.ChangePieceScreen
       android:id="@+id/transform_screen"
       android:layout_width="0dp"
       android:layout_height="0dp"
       app:layout_constraintBottom_toBottomOf="@+id/board_game"
       app:layout_constraintEnd_toEndOf="@+id/board_game"
       app:layout_constraintStart_toStartOf="@+id/board_game"
       app:layout_constraintTop_toTopOf="@+id/board_game"
       android:gravity="center"
       android:visibility="invisible"
       android:orientation="vertical">
       <TextView
           android:gravity="center"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:fontFamily="@font/permanent_marker"
           android:text="@string/on_board_screen"
```

```
            android:textColor="?attr/changePiece_textColor"
            android:textSize="@dimen/txt_size_sub_btn_menu"
            android:layout_weight=".5"/>
    </com.example.projetmobile.Model.ChangePieceScreen>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Par exemple, la classe “Board” hérite de “TableLayout”, et doit donc redéfinir des méthodes afin de s’afficher correctement (taille induite par ses éléments fils, affichage global par rapport aux autres éléments d’interface, notion de transparence et de profondeur comme en HTML).

Enfin, qui dit développement d’application mobile, dit gestion des capteurs présents sur le téléphone. Nous avons donc mis en place une mécanique de jeu qui marche seulement pour une partie locale. On souhaite détecter une grande secousse afin de redémarrer la partie, mimant la secousse d’un échiquier faisant renverser ses pièces de jeu. On adapte donc l’activité “GameActivity” afin de lui donner ce comportement. Premièrement, “GameActivity” implémente l’interface “SensorEventListener”.

```
public class GameActivity extends AppCompatActivity implements
SensorEventListener {
…
}
```

Ensuite, on récupère le “SensorManager” et l'accéléromètre, si le téléphone en possède un, dans la fonction “onCreate”.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

...
      sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

      accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
      if (accelerometer == null) {
            Toast.makeText(this, "Pas d'accelerometre :(",
Toast.LENGTH_SHORT).show();
      }

      previousAcceleration = new float[3];
      previousAcceleration[0] = 0;
      previousAcceleration[1] = 0;
      previousAcceleration[2] = 0;
```

```
}
```

Enfin, on définit les fonctions nécessaires au fonctionnement du capteur. Pour détecter la secousse, on passe par la notion de “jerk”, la dérivée de l’accélération sur le temps. On compare donc la longueur du vecteur de “jerk” par rapport à une limite, et on relance la partie si sa longueur est trop grande.

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
   float x, y, z;
   x = sensorEvent.values[0] - previousAcceleration[0];
   y = sensorEvent.values[1] - previousAcceleration[1];
   z = sensorEvent.values[2] - previousAcceleration[2];
   double length = Math.sqrt(x*x+y*y+z*z);

   boolean greatJerk = length > jerkRequired;
   if (greatJerk) {
       restartGame();
   }

   previousAcceleration[0] = sensorEvent.values[0];
   previousAcceleration[1] = sensorEvent.values[1];
   previousAcceleration[2] = sensorEvent.values[2];
}

@Override
public void onAccuracyChanged(Sensor sensor, int i) {

}



@Override
protected void onResume() {
   super.onResume();
   sensorManager.registerListener(GameActivity.this, accelerometter,
SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
   super.onPause();
   sensorManager.unregisterListener(this);
}
```

2.2. Programmation web

Second exemple de rendu obtenu via Blender et présent dans le site WEB

2.2.1. Utilisation de librairies graphiques

Pour les besoins de ce projet, nous avons dû mettre en place un client WEB, en plus de l'application mobile initiale. Comme ce client n'est pas l'acteur principal du projet, qui est l'application Android, nous avons pris la liberté de réutiliser des librairies de design et d'aide à la mise en page. Nous avons par exemple utilisé Bootstrap v5.2. Cette librairie propose une liste d'éléments de design et de classes CSS permettant la mise en place intuitive d'interfaces adaptatives et ergonomiques. Cela apporte au passage un aspect de standardisation d'éléments graphiques, notion importante déjà présente dans le développement de notre application mobile.

Cette librairie apporte également des comportements JavaScript, utile pour la mise en place de Toast côté site WEB par exemple.

Exemple de Toast de connection au site

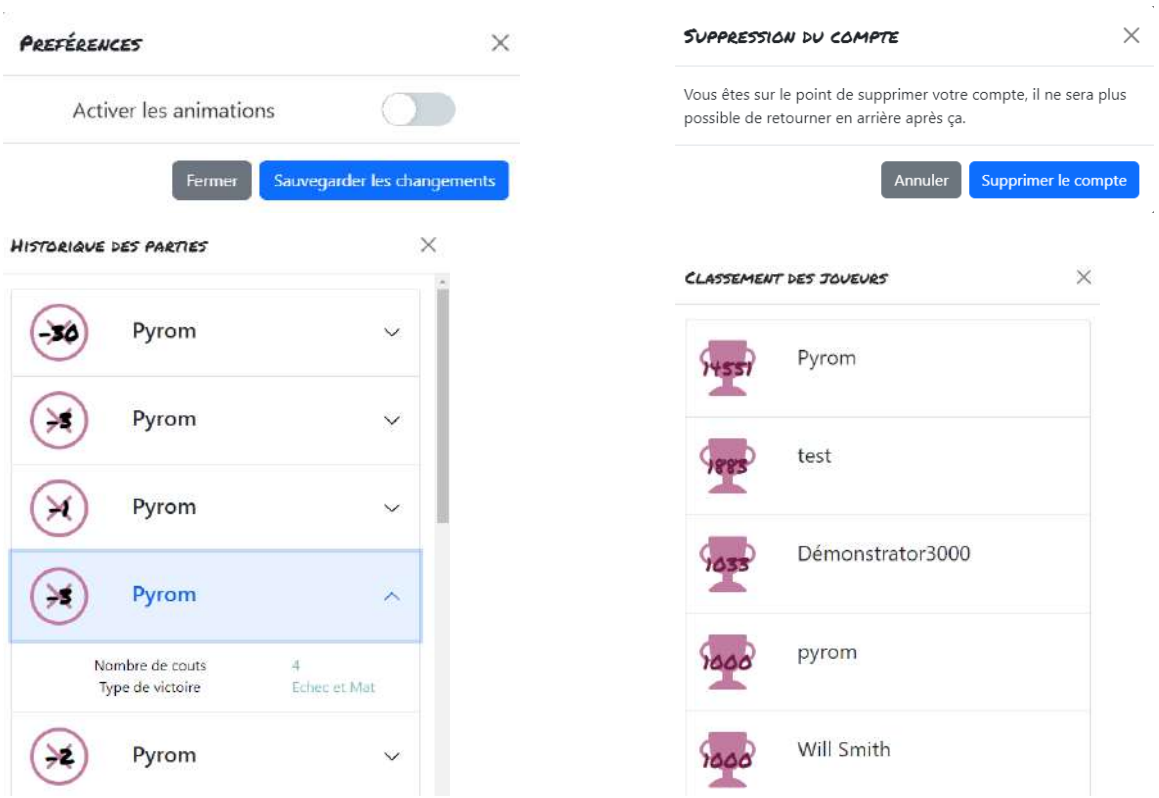
Pour les Toast par exemple, on utilise un unique élément HTML, dont on va simplement venir modifier le corps du message affiché en fonction du contexte d'appel de ce Toast (toastValue).

```
const dom_alert_toast = document.getElementById("AlertMessageToast");

function showToast(time){
   let myToast = new bootstrap.Toast(dom_alert_toast, {
       delay: time ?? 5000
   });
   myToast.show();
}
```

Ce Toast va être affiché depuis le code JavaScript, afin de pouvoir lui donner une durée d'affichage à l'écran par exemple. En effet, cet élément est par défaut invisible sur notre site, mais présent quand même dans la page. C'est en générant un nouveau Toast Bootstrap avec le bon élément DOM que l'on va pouvoir le faire apparaître ensuite (méthode show).

```
<div id="AlertMessageToast" class="toast" role="alert"
aria-live="assertive" aria-atomic="true">
   <div class="toast-header d-flex justify-content-around
align-items-center">
       <img src="./res/drawable/img/img.png" class="rounded mr-2"
alt="logoPasBo">
       <strong class="title">ChessBurger2</strong>
           <button type="button" class="ml-2 mb-1 btn-close"
data-bs-dismiss="toast" aria-label="Close">
               <span aria-hidden="true">&times;</span>
           </button>
   </div>
   <div id="toastValue" class="toast-body">
       JE SUIS UN TOAST TOUT BO
   </div>
</div>
```

Toujours à propos de la librairie Bootstrap, on implémente également dans notre client WEB d'autres éléments de design, comme les "Modal". Les "Modals" sont des éléments de design comparables dans l'utilisation que l'on en fait, à des "dialogs" en Android.



Exemple de Modals de jeu

Ces éléments possèdent dans leur construction un titre renseignant le type de l'action demandée à l'utilisateur, un corps de contenu variable (boutons, texte, champs de saisie d'informations textuelles, listes d'éléments dynamiques ou non…) et un affichage par-dessus la page originelle, mimant le comportement des "dialogs" que l'on utilise dans notre client Android (c'est-à-dire avec un fond semi-transparent et au milieu de la page). Les "Modals" de saisie d'informations possèdent également des boutons de validation ou d'annulation des champs saisies, pour un accompagnement au plus près de l'utilisateur dans son usage du site. Plus un utilisateur se sentira guidé dans sa manipulation de notre client WEB (mais aussi Android), et plus il sera à même de réaliser les actions que l'on souhaite qu'il fasse simplement.

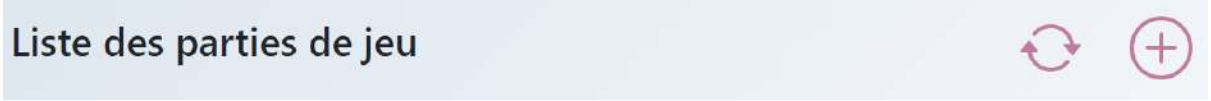
Une "Modals" se manipule par le biais d'un bouton renseignant sur son activation, la bonne fenêtre à afficher.

```
<div class="modal fade" id="CreatePartyModal" tabindex="-1"
aria-labelledby="CreatePartyModalLabel" aria-hidden="true">
   <div class="modal-dialog modal-dialog-centered">
       <div class="modal-content">
           <div class="modal-header">
               <h5 class="modal-title" id="CreatePartyModalLabel">Créer
```

```
une partie</h5>
                <button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">
                <label class="form-label" for="partyName">Nom de la
partie</label>
                <input type="text" id="partyName" class="form-control
form-control-lg" placeholder="Nom de partie">
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary"
data-bs-dismiss="modal">Fermer</button>
                <button type="button" class="btn btn-primary"
id="createGameButton" data-bs-dismiss="modal">Créer une partie</button>
            </div>
        </div>
    </div>
</div>

[...]

<!--bouton d'activation de la Modal de création d'une partie de jeu-->
<p data-bs-toggle="modal" data-bs-target="#CreatePartyModal">
```

Ici, on a par exemple la présence d'une “Modal” utilisée pour la création d’une partie de jeu, avec son bouton d'activation (lancement de son affichage).

Enfin, Bootstrap met à disposition un bon nombre d'icônes gratuitement, et ce, afin de rendre les interfaces développées plus simples dans leur compréhension par un utilisateur (plus accessibles). Une icône va venir traduire graphiquement un concept ou un objet, de représentation souvent simplifiée, voire imagée.



Exemple d’utilisation d'icônes pour l'esthétique (en haut) ou un réel comportement (en bas)

2.2.2. Exemples de composants graphiques importants

Bootstrap permet également la mise en place des éléments dans une page HTML sous la forme de grilles intuitives (reprenant le comportement natif des éléments de style “flex”). Pour la mise en page de la partie de jeu par exemple, je sépare la partie du plateau de jeu des éléments graphiques des joueurs dans deux colonnes séparées.

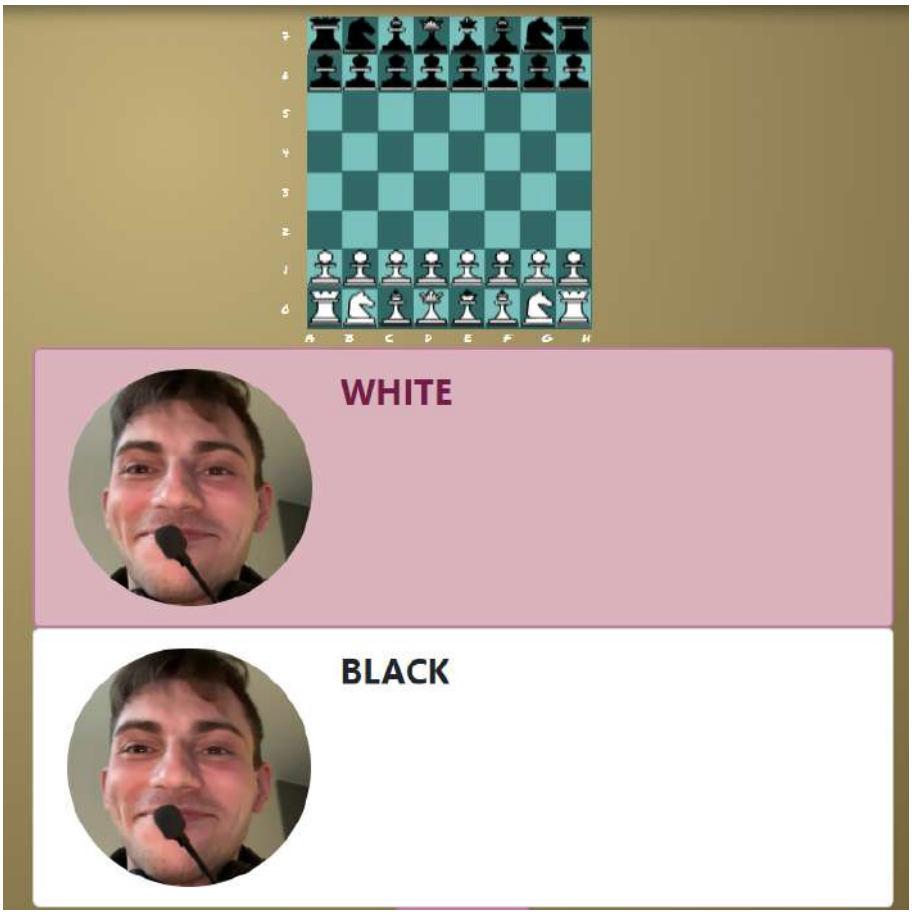
```
<div class="mt-5 container justify-content-center align-content-center">
   <div id="chessGame" class="row flex-lg-row d-flex flex-column">
       <div id="game" class="col-lg-6 col-md-12 h-100">
       </div>
       <div id="gameUI" class="col-lg-6 col-md-12 d-flex flex-column">
       </div>

       [...]
   </div>

</div>
```

On indique ici que pour une taille d’écran large, les deux éléments (de jeu et de données joueurs), seront disposés sur la même ligne, mais que pour une taille d’écran médium, ils seront disposés les uns à la suite des autres.

Breakpoint	Class infix	Dimensions
Extra small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Tableau des dimensions disponible sur le site internet de Bootstrap

Exemple d'affichage obtenu avec une taille d'écran large (ou supérieure)



Exemple d'affichage obtenu avec une taille d'écran médium (ou inférieure)

On remarque que le plateau garde une taille carrée, comportement qui est rendu possible par l'application d'un ratio 1:1 sur l'élément.

```
<div id="game" class="col-lg-6 col-md-12 h-100">
  <div id="root_board" class="container">
```

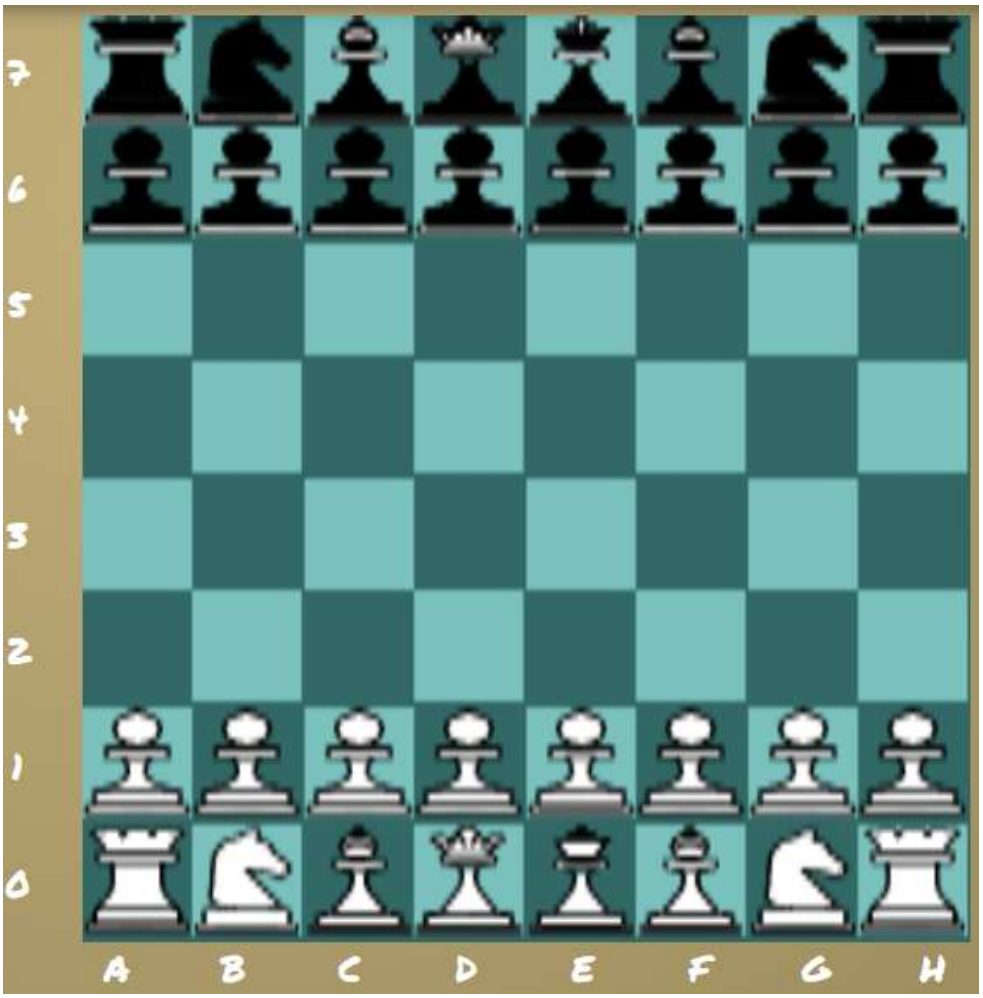
```
    <div id="boardWrapper" class="row">
<!--INDICATION DES COLONNES DU PLATEAU-->
<div class="col_infos col-2 col-sm-1 d-flex flex-column" style="color:
white; font-family: &quot;Permanent Marker&quot;, cursive;">
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">7</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">6</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">5</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">4</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">3</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">2</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">1</div>
          <div class="p-2 text-center" style="display: flex; flex: 1 1
0%; justify-content: center; font-size: 1.1vw; align-items:
center;">0</div>
   </div>
   <div class="col-10 col-sm-11">
     <div id="canvasWrapper" class="row ratio ratio-1x1">
       <canvas id="gameBoardCanvas"></canvas>

        <!--ECRAN DE TRANSFORMATION D'UNE PIECE-->
        <div id="transform_screen" class="cover-container d-flex h-100
p-3 mx-auto flex-column text-center" style="display: none !important;">
            <h4 class="player"></h4>
                <main>
                  <h3 class="message">Veuillez choisir une pièce à
transformer</h3>
                  <div class="row mt-3 justify-content-around">
                       <img class="p-2 imgtransform col-5 mt-2" alt="img
tower">
                       <img class="p-2 imgtransform col-5 mt-2" alt="img
bishop"><img class="p-2 imgtransform col-5 mt-4" alt="img queen">
                       <img class="p-2 imgtransform col-5 mt-4" alt="img
```

```
knight">
                    </div>
                </main>
                <footer class="mt-auto">
                    <div class="inner">(click to chose)</div>
                </footer>
            </div>

        <!--ECRAN DE FIN DE PARTIE-->
        <div id="end_screen" class="cover-container d-flex h-100 p-3
mx-auto flex-column text-center" style="display: none !important;">
            <h3 class="start"></h3>
             <main>
                <h1 class="message"></h1>
             </main>
             <footer class="mastfoot mt-auto">
                 <div class="inner">
                     <h3 class="end cover-heading"></h3>
                 </div>
              </footer>
          </div>
      </div>
  </div>

  <!--INDICATION DES LIGNES DU PLATEAU-->
  <div class="row row_infos" style="color: white; font-family:
&quot;Permanent Marker&quot;, cursive;">
<div class="col-2 col-sm-1"></div>
<div class="col" style="text-align: center; font-size: 1.1vw;">A</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">B</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">C</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">D</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">E</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">F</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">G</div>
<div class="col" style="text-align: center; font-size: 1.1vw;">H</div>
  </div>
</div>
</div>
</div>
```

On affiche un certain nombre d'éléments d'indication du plateau, comme les informations de lignes (0 - 9) et de colonnes (A - H), présentes aussi en Android. On met également en place des écrans de fin de partie et de choix de changement de pièces. Ces écrans sont contextuels et seront affichés (display block) au bon moment suivant le déroulement de la partie de jeu. Ce sont des éléments DOM en position “absolute”, ce qui

veut dire qu'ils vont se positionner relativement à la dernière balise de position “relative”, en l'occurrence la balise englobante du canvas de dessin du plateau de jeu. Ils vont donc, le cas échéant, venir couvrir la totalité du plateau de jeu.

Il faut également tenir en compte dans la version en ligne de ne pas afficher ces écrans pour les deux clients, ou avec des informations différentes, en fonction du contexte d'appel et de l'avancement de la partie.



Plateau de jeu

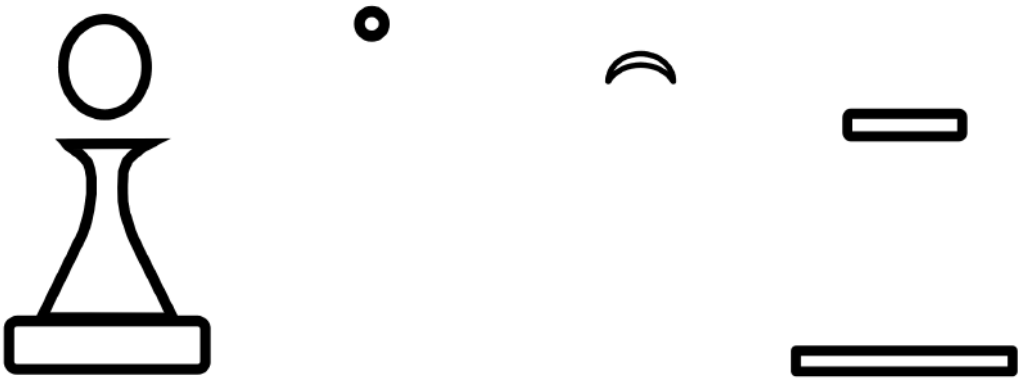
Pour l'affichage du plateau côté WEB, on utilise une balise particulière en HTML, la balise canvas. Elle sert à afficher du contenu 2D et 3D sur une page internet. Cette balise représente une toile de dessin, depuis laquelle il devient possible de peindre des formes géométriques, ou plaquer des images dans une dimension précise.

Cet ensemble d'éléments graphiques pour l'affichage de la partie de jeu est généré procéduralement depuis le code JavaScript, afin de pouvoir opérer une plus grande manipulation et génération de parties graphiques liées aux modèles de jeu.

Dans notre implémentation, une case est donc vue graphiquement comme une zone carrée du canvas de plateau, possédant un certain nombre d'informations de rendu possibles (pièce qui possède une apparence, case sélectionnée). L'apparence des pièces est ici gérée par une classe à part : la classe “ComposedDrawing”.

```
export let main_color_piece                 = ["#FFFFFF", "#000000"];
export let border_color_piece               = ["#000000", "#000000"];
export let color_piece_second               = ["#000000", "#FFFFFF"];
export let plate_color_piece                = ["#FFFFFF","#FFFFFF"];

let p1_bishop1 = new Bishop(players[0], main_color_piece[0],
plate_color_piece[0], border_color_piece[0], color_piece_second[0]);
```

Cette classe permet de renseigner un élément de dessin comme une composition de couches de dessin, souvent définie en SVG, dont leurs couleurs de bordure et de remplissage sont paramétrables. Cela nous permet de ne générer qu'un design en couche par pièces, puis de le customiser à volonté ensuite.



Vision en couche de la pièce du fou

Dans les options d'utilisateurs, on alloue la possibilité de désactiver ou pas les animations du déplacement des pièces sur le plateau de jeu. En JavaScript, on ne possède pas de classes Animator comme sur Android. En effet, pour la gestion des déplacements continus sur le plateau de jeu côté client mobile, on passe par un ValueAnimator entre 0 et 1.

```
ValueAnimator anim_piece = ValueAnimator.ofFloat(0.0f, 1.0f);
anim_piece.setDuration(this.piece_globalDuration);
anim_piece.addUpdateListener(animation -> {
   float value = (Float) animation.getAnimatedValue();
   top_pieceToMove = (int) (start_y + (end_y - start_y) * value);
   left_pieceToMove = (int) (start_x + (end_x - start_x) * value);
   bottom_pieceToMove = top_pieceToMove + h_piece;
   right_pieceToMove = left_pieceToMove + w_piece;
   invalidate();
});
anim_piece.addListener(onTheEnd);

anim_piece.setInterpolator(new AccelerateInterpolator());
```

```
anim_piece.start();
```

Pour la partie JavaScript, on a dû générer notre propre objet Animator, se comportant de la même manière que celui présent en Java.

```
function animationCycle(t, object, start, end) {
   if (object.startTime === undefined) object.startTime = t;
   let relativeTime = (t - object.startTime) / object.duration;
   if (relativeTime <= 1) {

       //Draw elements
       object.board.clear();
       object.board.drawBoard();

       let valAnim = ((object.getVal)(relativeTime));

       object.drawingElement.bounds.right    = start.right     +
(end.right -  start.right)    * valAnim;
       object.drawingElement.bounds.left     = start.left      +
(end.left -   start.left)     * valAnim;
       object.drawingElement.bounds.top      = start.top       +
(end.top -    start.top)      * valAnim;
       object.drawingElement.bounds.bottom   = start.bottom    +
(end.bottom - start.bottom)   * valAnim;

       //Draw the element
       object.drawingElement.draw(object.canvasContext);
       //LOOP
       object.id = requestAnimationFrame((t)=> animationCycle(t,object,
start, end));

   } else if (object.animationOn) {
       cancelAnimationFrame(object.id);
       object.animationOn = false;
       object.animation_cpt++;

       if (object.animation_cpt<object.animation_loop_cpt) {
           object.startTime = t;
           requestAnimationFrame((t)=> animationCycle(t,object, start,
end));
       }else{
           object.onEnd();
       }
   }
}
```

2.2.3. Code en JavaScript

Concernant les appels à la base de données, côté JavaScript, ils sont généralement utilisés pour : soit ajouter de l'information, soit la modifier ou encore se mettre un event listener afin de réceptionner tous les changements de certains attributs spécifiques de la BDD. Très intéressant lorsque l'on veut voir, par exemple, si un joueur vient de jouer.

Ici le code pour le listener de loose, attributs dans rooms, utilisé pour voir si le joueur a abandonné

```
export function looseListener(uid, playerIndex, obj) {
    console.log("LOOSE LISTENER");
    refLoose = ref(db, "rooms/" + uid + '/loose');
    onValue(refLoose, (snapshot) => {
        //turn listener simplification

        if (snapshot.val() == playerIndex) {
            obj.winByFF();
        }
        console.log(" ========== ============== ========== ");
    }, /*{
        onlyOnce: true
    }*/);
}
```

Ou encore, par exemple lors de la création d'un joueur côté realtime database, nous avons instancié les attributs pour les joueurs tels que leur pseudo, elo, etc …. Permettant donc dans la page profil ou encore lorsque l'utilisateur joue en ligne d'afficher ses informations aux autres joueurs.

```
function writeUserData(userId, mail, pseudo) {
    const db = getDatabase();
    set(ref(db, 'users/' + userId), {
        pseudo: pseudo,
        elo: 1000,
        email: mail,
        bio: "Un g@meur avec un @ à la place du a",
        useAnimations : 0
    });
}
```

Cette fonction est surtout encapsulée avec d'autres fonctions, permettant aux autres développeurs de ne pas avoir accès directement à la base de données, mais d'utiliser des fonctions générique développeur qui sont utilisables dans un maximum de cas d'utilisation. Cela permet par la même occasion d'éviter de réécrire du code.

3. Modèle de jeu

Nous sommes partis sur une application proposant la possibilité pour un utilisateur de jouer aux échecs. Nous avons donc dû implémenter tout un système de gestion de jeu d'échec, pour le faire ensuite correspondre aux interactions d'un utilisateur lors d'un match en local, ou en ligne. Nous allons détailler le modèle pour la partie Java, car elle est identique, à peu de choses près, à la version JavaScript.

3.1.1. Pièces et déplacement

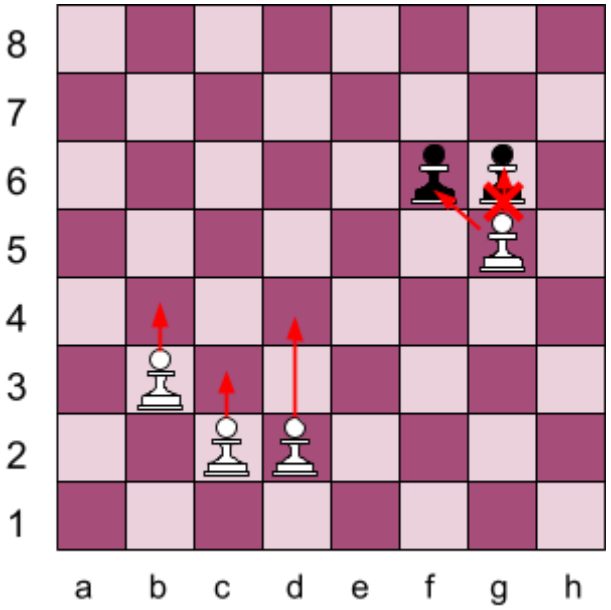
Nous avons dans un premier temps modélisé le concept de déplacement de jeu sur un plateau de cases. Pour cela, nous distinguons deux types de déplacements :

- déplacement ponctuels (une seule fois, téléportation)
- déplacement vectoriels (potentiellement infini, s'arrêtant au premier obstacle)

On peut aussi distinguer un type d'action pour un déplacement précis, comme le fait de manger une pièce adverse. Cela va avoir pour effet de générer des déplacements conditionnels, ne pouvant se déclencher que pour des conditions précises (ici une pièce adverse à portée).

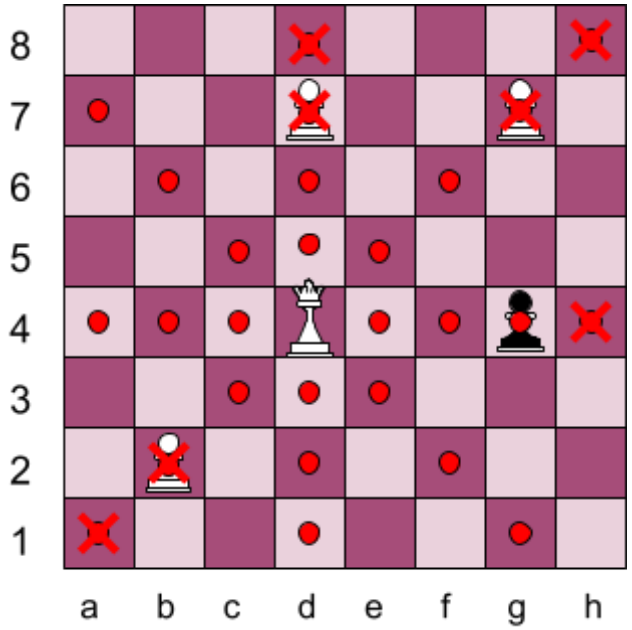
Par la suite, on peut combiner différents types de déplacements, et même en générer de nouveaux, afin de spécifier des comportements encore plus riches (mécanique de variantes de jeu). On a voulu générer un code de modèle le plus générique possible afin d'allouer la possibilité future de pouvoir implémenter des variantes de jeu, avec des objets de jeu aux comportements différents du jeu original.

Pour la pièce du pion par exemple, on peut distinguer un déplacement vectoriel vers l'avant d'une ou deux cases (si c'est son premier déplacement ou non), et des déplacements conditionnels ponctuels en diagonale s'il peut manger une cible. On renseigne à la construction d'un pion sa direction de déplacement, afin de garder un comportement générique pour les pions noirs ou blancs.



Déplacements d'un pion de jeu

Pour la pièce de la reine par exemple, on peut distinguer un déplacement vectoriel conditionnel dans toutes les directions des points cardinaux, détectant si une pièce adverse est à portée.



Déplacements de la reine

Ainsi, chaque type de pièces est une sous-classe de la classe abstraite "Piece", qui renseigne le comportement général d'une pièce d'un jeu d'échec.

```
public abstract class Piece implements GameObject {
    public enum DIRECTION { UP, DOWN, LEFT, RIGHT}

    protected boolean movedYet;
    protected ComposedDrawing appearance;
    protected Player possessor;
    private List<Piece> tastyPieces;

    protected Piece lastShape;
    private boolean deapCloneOnGame;

    public Piece(Player p) {
        this.movedYet = false;
        this.possessor = p;
        appearance = new ComposedDrawing();
        tastyPieces = new ArrayList<>();
    }

    /**
```

```
     * Return the visual appearance of a piece
     *
     * @return return all Drawables needed to be drawn
     **/
    public ComposedDrawing getAppearances() {...}

    /**
     * Get all the possible movement that a piece can perform
     *
     * @param col : column since the piece start to compute mvt
     * @param row : row since the piece start to compute mvt
     * @return list of movements that the piece can achieve
     **/
    public abstract List<Movement<? extends GameObject>>
getAllPossibleMvt(int col, int row);


    /**
     * Set that a piece has moved
     *
     * @param b : value to set to the current moving detector of the
piece
     **/
    public void setMoved(boolean b) {...}

    /**
     * Get the possessor of a piece
     *
     * @return : Player that control the piece this
     **/
    public Player getPossessor() {...}

    /**
     * Get if the piece is a victory condition piece
     **/
    public boolean isVictoryCondition() {...}

    /**
     * Get all the piece that this can eat currently
     *
     * @return List of tasty pieces
     **/
    public List<Piece> getTastyPieces() {...}

    /**
     * Get the value of the displacement of a piece
```

```
     **/
    public boolean isMovedYet() {...}

    /**
     * Clear all the pieces that this can eat currently
     **/
    public void clearTastyPieces() {...}

    /**
     * Say if a piece can be transformed or not
     **/
    public boolean canBeTransformed() {...}

    /**
     * Get the last shape of a piece, just before its get cloned
     **/
    public Piece getLastShape() {...}

    /**
     * Get back to the precedent shape for a piece
     **/
    public void getBackPrecedentShape() {...}
}
```

Une pièce possède une apparence, qui sert à la caractériser visuellement sur le plateau de jeu. Elle possède aussi des attributs permettant de renseigner si celle-ci peut être transformable (comme le pion s'il se trouve sur la dernière rangée), si elle a déjà bougé au cours de la partie ou encore s'il s'agit d'une pièce de condition de victoire, comme c'est le cas du roi. Elle possède aussi une référence vers son processeur courant (Joueur).

À côté de ça, une pièce va renseigner la liste des mouvements possibles qu'elle peut réaliser sous la forme d'une liste d'objets de la classe "Movement".

```
public abstract class Movement<T extends GameObject> {
    protected Action<T> action;
    protected Position start;
    protected Position incrementation;

    public Movement(Action<T> action, Position start, Position
incrementation) {
        this.action = action;
        this.start = start;
        this.incrementation = incrementation;
    }
```

```
    /**Compute all the possible position that a Mouvement can perform
     * @param b : Board of the current game
     * @return a list of Position computed by this mouvement**/
    public abstract List<Position> getAllPositions(Board b);
}
```

C'est cet objet "Movement" qui va être évalué en fonction du contexte de jeu courant (Board), afin de générer une liste de positions, couple de valeurs (coordonnées X et Y), qui représenteront les mouvements valides de la pièce de jeu. Par la suite, ces mouvements vont être filtrés afin de ne garder que ceux pour lesquels le Joueur pourra réellement choisir.

Ce comportement traduit le fait qu'une pièce possède toujours le même nombre de mouvements, mais qu'elle ne pourra pas toujours les effectuer à un instant donné de la partie de jeu.

L'objet Action traduit ici la typologie du mouvement, soit la condition à vérifier pour que le mouvement soit valide (comme le déplacement conditionnel si une pièce ennemie est à portée).

```
public List<Position> getAllPositions(Board b) {
        List<Position> res = new ArrayList<>();
        int cpt_dep = this.cpt_mvt_performed;
        Position p = start.addAndReturn(new Position());

        while (b.isGoodPos(p.getX(),p.getY()) && (cpt_dep != 0)){
            p = p.addAndReturn(incrementation);
            Action.ActionState ac =
action.isValidated(b.getACase(p.getX(),p.getY()));

            if(Action.ActionState.VALID == ac || (canStillMove && ac ==
Action.ActionState.STILLGOOD)){
                res.add(p);
                if(Action.ActionState.VALID == ac){
                    break;
                }
            }else{
                break;
            }
            cpt_dep --;
        }
        return res;
}
```

Sur le code précédent, on détaille la boucle de génération des positions possiblement atteignables (avant filtration) pour une pièce de jeu. Un mouvement peut avoir

plusieurs états possibles durant son évaluation pour une “Position” donnée. Soit il est “INVALID”, et on arrête sans prendre en considération la position regardée, soit il est “VALID”, alors on s'arrête, mais on prend cette fois-ci en considération la “Position” observée comme possiblement atteignable par la Piece en question, soit la Position est “STILLGOOD”, alors on peut continuer d'observer les “Positions” suivantes (mécanique des déplacements vectoriels).

3.1.2. Le plateau de jeu

Le plateau de jeu centralise la plupart des données de connaissance de jeu pour les Joueurs. Cette structure de données se compose d’une liste de Cases, qui vont elles-mêmes contenir des informations telles qu’une possible “Pièce” sur elle, ou des informations de mise à jour graphique (“Case” précédemment sélectionnée par exemple).

Un plateau de jeu va entre autres construire les différentes instances de jeu pour pouvoir jouer (Pieces qu’il va positionner sur le plateau de jeu, et qu’il va attribuer à la bonne instance du Joueur) et retourner les Joueurs d’une partie de jeu.

```
public class Player implements GameObject{
    private Map<Integer, ComposedDrawing> appearancesPieceToTransform;

    private String pseudo;
    private Map<Piece, List<Position>> piecesPlayer;

    private Map<Piece, List<Association_rock>> rockPieces;

    private List<Piece> cimetary;
    private int color;


    public Player(String pseudo,int color){
        this.piecesPlayer = new HashMap<>();
        this.rockPieces = new HashMap<>();

        this.pseudo = pseudo;
        this.color = color;
        this.cimetary = new ArrayList<>();

        appearancesPieceToTransform = new HashMap<>();
    }
[...]
}
```

Un “Joueur” est défini dans notre implémentation comme processeur de “Piece”, se distinguant des autres Joueurs par un pseudonyme et une couleur d’affichage.

Le plateau de jeu va également venir renseigner tous les comportements d'interactions possibles, comme le déplacement d'une "Piece", ou déterminer si un déplacement est valide.

Pour les mises à jour graphiques, nous ne redessinons que les "Cases" qui ont subi un changement au cours du tour de jeu, et pas l'entièreté du plateau. Cela permet d'économiser des performances de jeu, qui est un point important en programmation mobile.

3.1.3. GameManager

Le GameManager est la classe qui va modéliser l'ordonnanceur d'une partie de jeu. C'est lui qui va s'assurer du bon déroulement d'un tour de jeu, et plus généralement de la partie, en passant la main d'un Joueur à l'autre.

Dans sa version locale, un GameManager va se charger de définir la fonction principale de callback sur les événements de clic sur le plateau. Il va également manipuler la boucle principale de jeu. Pour cela, il va devoir calculer, pour le Joueur courant, tous les mouvements qu'il peut effectuer. Il va donc récupérer la liste des coups parmi les mouvements que peuvent faire les "Pieces" qu'il lui reste sur le plateau. Il va ensuite réduire ces mouvements en calculant leur validité par rapport à l'état de menace de ses "Piece" condition de victoire. En effet, si un mouvement rend le Joueur en question en échec, alors ce mouvement est retiré des mouvements possibles qu'il peut faire. Si à la fin de cette phase, le Joueur courant n'a plus de mouvement possibles, alors la partie se termine. S'il est menacé, ce joueur sera déclaré comme perdant, sinon il y a juste égalité entre les deux joueurs.

```
/**
 * ======== For launch and play a game ========
 **/
public void start() {
    //Clear the board
    this.board.clear();

    //Init the board
    this.players = this.board.initGame_players();
    majPlayerLayout();

    if (startANewTurn()) {
        onEndingGame();
    } else {
        computeOnclkListener();
    }
}

 //Function for complete the main onclick listener of the board app
```

```
    protected boolean startANewTurn() {
        this.currentPlayer = getCurrentPlayer();

        //Maj UI for better understanding in interface of current player
        majLayoutPlayerTurn(this.currentPlayer);

        //Next we can compute the special rock movement
        computeRockInGame(this.currentPlayer);

        //We create all the movement for all the players
        for (Player p : players) {
            computePossibleMvts(p);
        }
        //And we restrict the current player movements if he is in
danger
        performMenaced(this.currentPlayer);

        //We can independently compute the dangerous Case by calculating
the possible position for each dangerous enemy neighbour
        performDanger(this.currentPlayer);

        return this.isFinished();
    }

    /**
     * ======== For game stopping mechanics ========
     **/
    public boolean isFinished() {
        //Detect if current player can still perform at least one
movement
        for (Piece p : currentPlayer.getPiecesPlayer()) {
            if (currentPlayer.getPositionsPiece(p).size() > 0) return
false;
        }
        return true;
    }
```

Il va également devoir gérer la mise à jour des affichages des éléments de jeu (cimetière, joueur qui doit jouer).

Il va devoir ensuite gérer les différents type de mouvements (animés ou non), que cela soit des déplacements simples ou plus complexes, comme celui du roque. Dans un premier temps, nous avons voulu modéliser ce mouvement avec notre système de mouvements généralistes, puis nous sommes revenus à une approche plus simple mais fonctionnelle.

Pour la promotion, le “GameManager” assure également la possible transformation d'une “Piece”, en appelant le bon écran de choix de “Piece” pour le Joueur concerné. Il en va de même pour la fin de partie, avec le message de fin de partie adéquat.

On conserve aussi un état constant de la partie de jeu sous la forme d'une pile de coups joués.

3.1.4. GameManager en ligne

Le principe du “GameManagerOnline” reste le même que “GameManager”. Seulement quelques éléments différents de celui-ci. Le principe est le suivant, quand un joueur effectue un coup, il va alors l’envoyer à la BDD qui va faire appeler un “pieceListener” qui va mettre à jour le “turn” dans la BDD (pour éviter les problèmes ou le “turn” est mis à jour avant le déplacement de la pièce). Puis le “turnListener” va être appelé, ce qui va permettre au joueur d’effectuer le coup de son adversaire, puis de pouvoir être débloqué pour jouer son propre coup et l’envoyer à la BDD, etc.

Concernant la mécanique d’abandon, c’est quant à lui l’attribut “loose” qui est mis à jour. Étant donné qu’un listener attend une modification de cet attribut, lorsqu’un des joueurs modifie “loose” par son “indexPlayer”, cela notifie l’autre que le joueur a abandonné.

Dernièrement, pour signaler qu’un joueur à perdu par échec et mat ou qu’ils sont en pat, le joueur qui perd ou qui reçoit le pat, va envoyer un coup vide à la BDD. Il va donc modifier “turn” par la même mécanique qu’auparavant et donc réveiller le joueur et récupérer le déplacement. Mais ce déplacement est un coup vide, ce qui va être traité comme un recalcul du plateau pour voir si le joueur en face a perdu de Pat ou d'échec et mat. Enfin, cela va signaler aux joueurs, par le biais de l’interface, la raison de fin de partie.

3.1.5. Java vs JavaScript

Comme énoncé précédemment, la version du modèle en Java ou en JavaScript ne change pas dans son comportement global. Cependant, la version JavaScript diffère un peu de la version Java dans sa gestion des événements de clic sur le plateau (gestion des interactions utilisateur).

En effet, en version Java, le plateau se compose de cases qui sont des vues personnalisées. Le plateau de jeu est lui-même une vue personnalisée, héritant de la vue TableLayout. On peut ainsi renseigner un traitement visuel (fonction onDraw) et un traitement au niveau des interactions utilisateurs, par une fonction callback sur les clics.

La version JavaScript se compose quant à elle d’un unique canvas pour symboliser le plateau de jeu. On va devoir récupérer les événements de clic sur cette balise spéciale, et les transformer en des coordonnées plateau de jeu (ligne et colonne).

```
fromCanvasCoordLinearToCaseInstance(x, y) {
```

```
    let row = Math.floor(y * this.board.nb_row);
    let col = Math.floor(x * this.board.nb_col);
    return this.board.getACase(col, row);
}
```

Pour la version web, nous ne chargeons qu'une seule fois les apparences des "Piece", puis nous en sauvegardons l'aspect dans une image tampons, que nous réutilisons au moment de redessiner la "Piece" sur le plateau. Nous passons également par des interpréteurs de fichiers SVG afin de pouvoir les manipuler efficacement sur canvas.

4. Déploiement/Installation

4.1. Installation mobile

Afin d'installer l'application sur mobile, on peut cloner le répertoire git avec android studio et compiler avec celui-ci.

4.2. Déploiement web

4.2.1. Back end : Firebase

La partie Firebase, grâce à son système, est déjà déployée dès sa création..

4.2.2. Front end : HTML

Pour le front end, il suffit de cloner dans un serveur le deuxième répertoire git du client web. Le site démarre à partir du fichier "index.html".

5. Manuel d’utilisation

Fonctionnalités	Android	Web
Créer un compte	**X**	**X**
Supprimer un compte	**X**	**X**
Se connecter	**X**	**X**
Voir les informations du profil	**X**	**X**
Changer de pseudo	**X**	**X**
Changer de photo de profil	**X**	**X**
Changer de biographie	**X**	**X**
Voir son historique de partie	**X**	**X**
Voir le classement des joueurs	**X**	**X**
Créer une partie	**X**	**X**
Rejoindre une partie	**X**	**X**
Jouer une partie	**X**	**X**
Abandonner une partie	**X**	**X**
Capteur pour redémarrer une partie (local)	**X**	

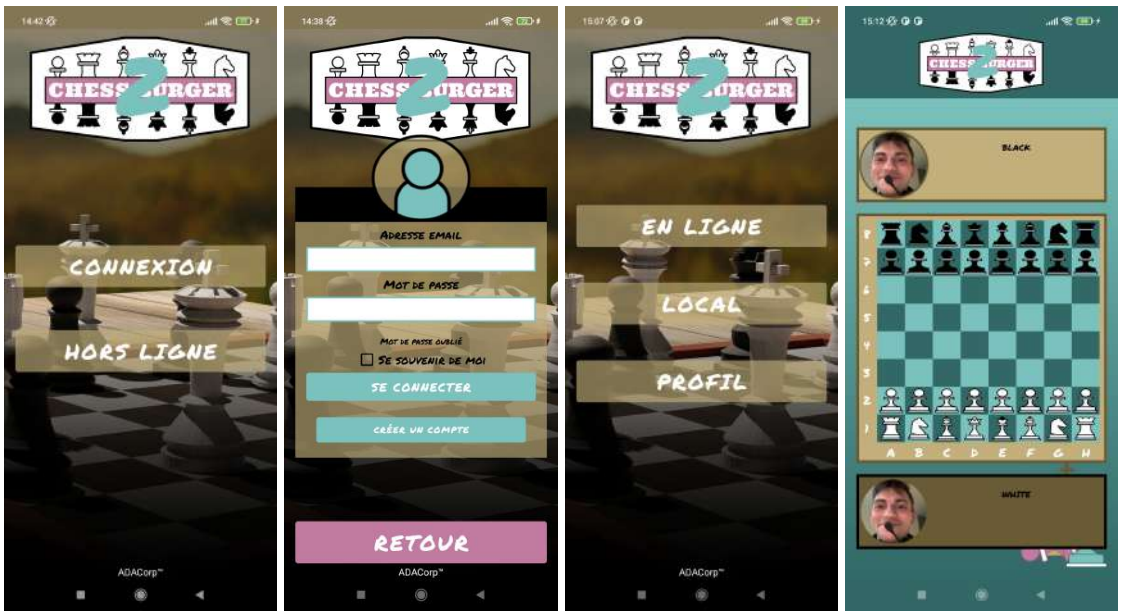
5.1. Fonctionnalité mobile

5.1.1. Jouer une partie en local

Pour jouer en local, on peut, dès le premier menu, appuyer sur le bouton “Hors Ligne”. Le plateau de jeu s’affiche et nous permet de jouer une partie.

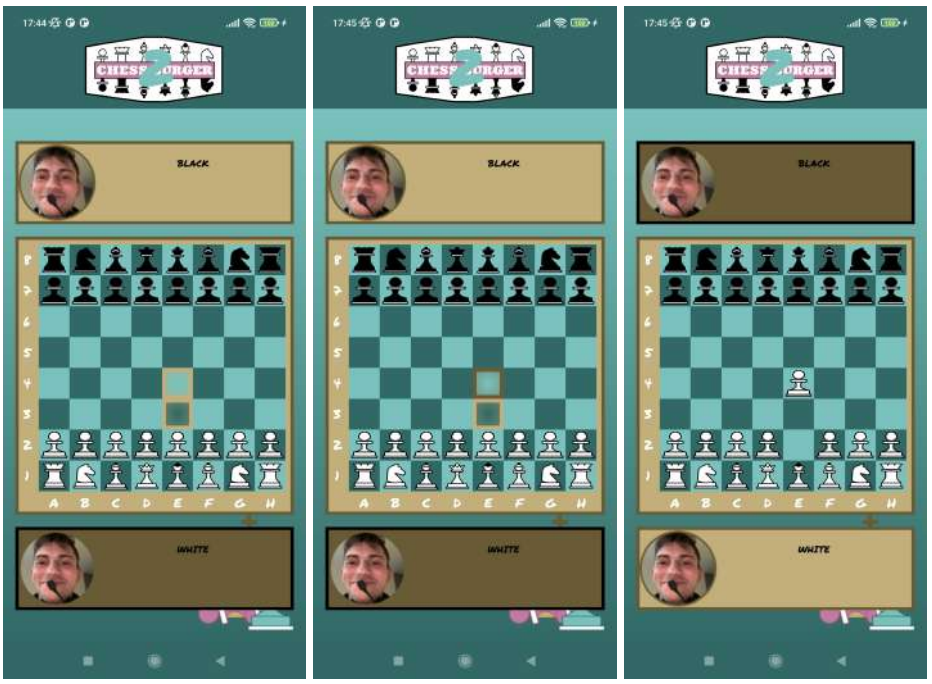
Chemin jusqu'au jeu local.

On peut également accéder à une partie en local après s'être connecté. Il faut donc aller dans "Connexion", puis remplir ses informations, ensuite appuyer sur "Se connecter" et enfin sur "Local".



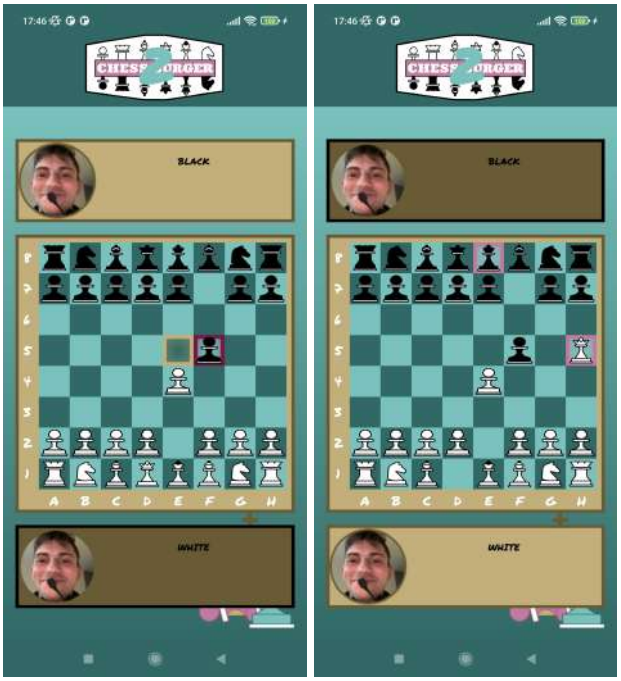
Deuxième chemin jusqu'au jeu local.

À partir de ce menu, on peut jouer une partie d'échec classique ; les blancs commencent. Pour cela, on peut appuyer sur la pièce que l'on souhaite déplacer. On voit alors l'ensemble des cases où l'on peut déplacer cette pièce. Ensuite, on sélectionne la case où l'on souhaite déplacer la pièce. Enfin, on confirme la case en appuyant une deuxième fois dessus. Ce principe en trois étapes pour un déplacement a été pensé pour le mobile puisque la précision que l'on peut avoir sur des petites cases n'est pas parfaite.



Les trois étapes de déplacement de pièce.

De plus, il y a des colorations en fonction de l'état des cases. Si l'on peut manger une autre pièce, alors la case devient rouge et si le roi est en échec, sa case devient rose.



Différentes couleurs de cases

Enfin, on dispose d'un écran de victoire qui se présente avec une animation magnifique.

Écran de fin de partie

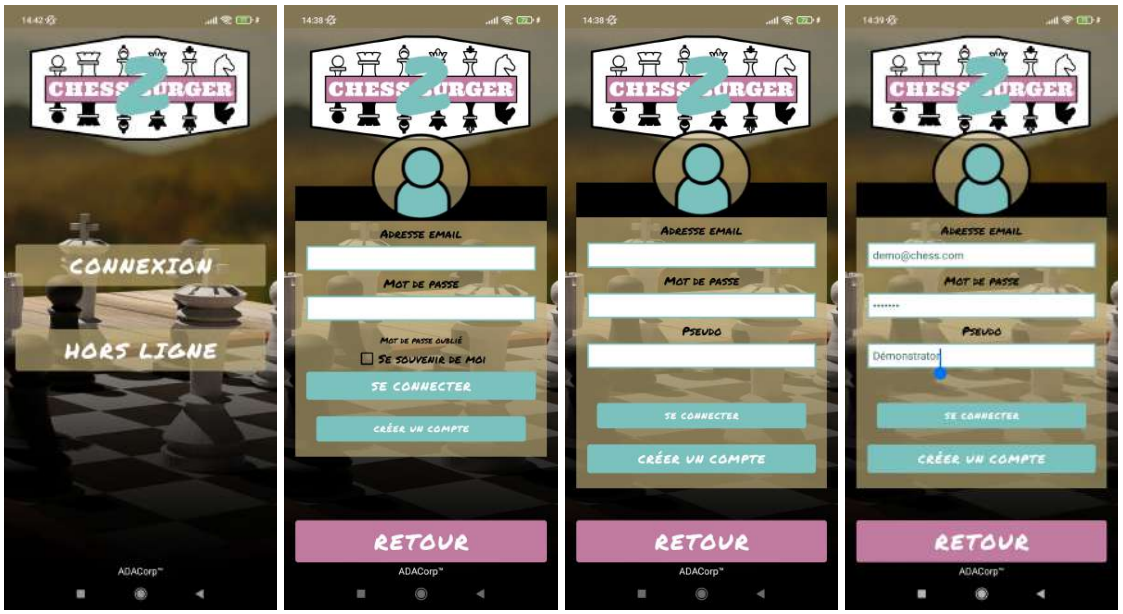
En plus de la partie, on a la possibilité d'ouvrir le menu burger. Pour ce faire, on doit appuyer sur le logo. Dans ce menu, on a la possibilité de recommencer ou de quitter la partie. Les modes de jeu n'ont finalement pas été implémentés.

Menu burger

On dispose également d'une autre manière pour recommencer la partie. Sur le menu de jeu, si l'on secoue le téléphone assez fort, alors la partie recommence ; mais il faut secouer très fort.

5.1.2. Créer un compte

Pour créer un compte, il faut, depuis le premier menu, aller dans “Connexion”, puis appuyer sur le bouton “Créer un compte”. On peut maintenant saisir les informations nécessaires à la création de compte. On finalise la création en appuyant à nouveau sur “Créer un compte”.



Chemin de création de compte

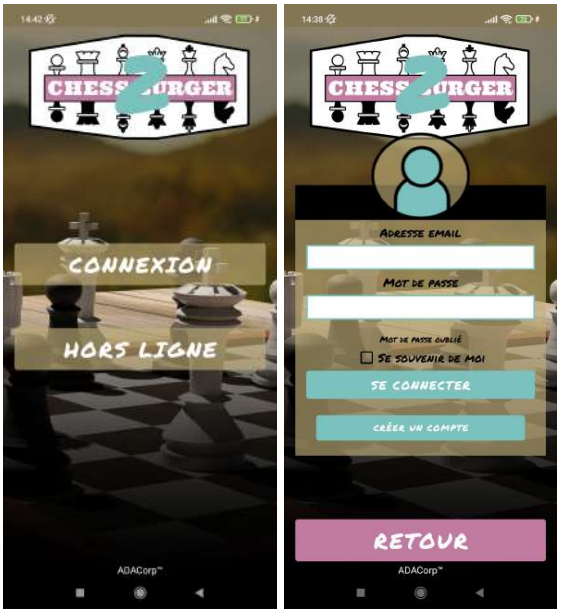
Bien sûr, on ne peut pas créer un utilisateur s'il manque des informations.

Il manque des informations

Une fois que l’utilisateur a créé son compte, il est automatiquement connecté et peut donc accéder à son profil et aux parties en ligne.

5.1.3. Se connecter

Pour se connecter, il faut aller, toujours depuis le premier menu, dans “Connexion”. L’utilisateur peut alors saisir ses informations et valider en appuyant sur “Se connecter”.



Chemin pour se connecter.

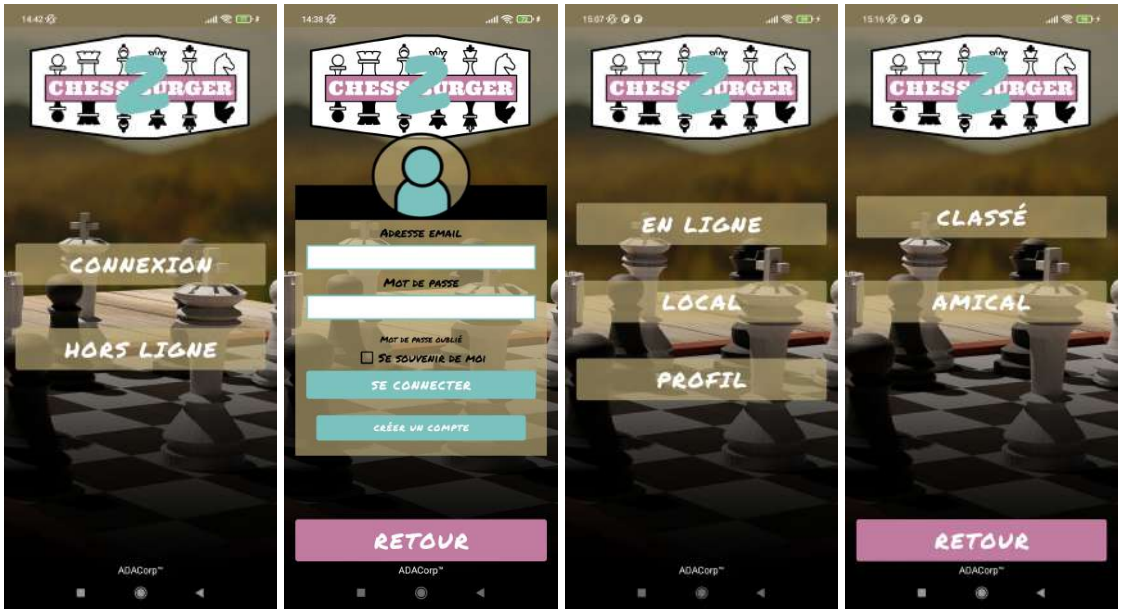
Bien sûr, si l’on donne de mauvaises informations, on ne peut pas se connecter. On ne doit pas laisser de champs vides et on doit mettre le bon mot de passe.

Mauvaises informations de connexion.

Une fois connecté, on est présenté par un autre menu permettant de jouer en ligne, jouer en local ou modifier son profil.

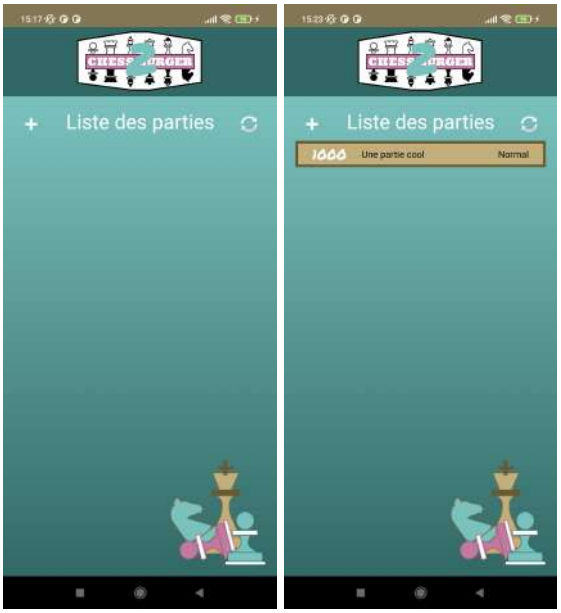
5.1.4. Jouer une partie en ligne

Pour jouer une partie en ligne, il faut tout d'abord naviguer jusqu'à la liste des parties en ligne. Pour cela, on doit aller dans “Connexion”, puis on se connecte avec “Se connecter”, ensuite, on appuie sur “En ligne”, et enfin “Amical”.



Chemin vers la liste de partie (les parties classés n'ont pas été implémentés)

On est donc présenté devant le menu qui liste les parties. En utilisant ce menu, on peut synchroniser les parties disponibles en appuyant sur le cercle de synchronisation en haut à droite.



Liste des parties : vide à gauche et contenant une partie à droite.

Bien sûr, s'il y a de nombreuses parties, on peut les faire défiler. Dans les parties en ligne, le comportement pour quitter est différent. En effet, cela correspond à abandonner la partie, on

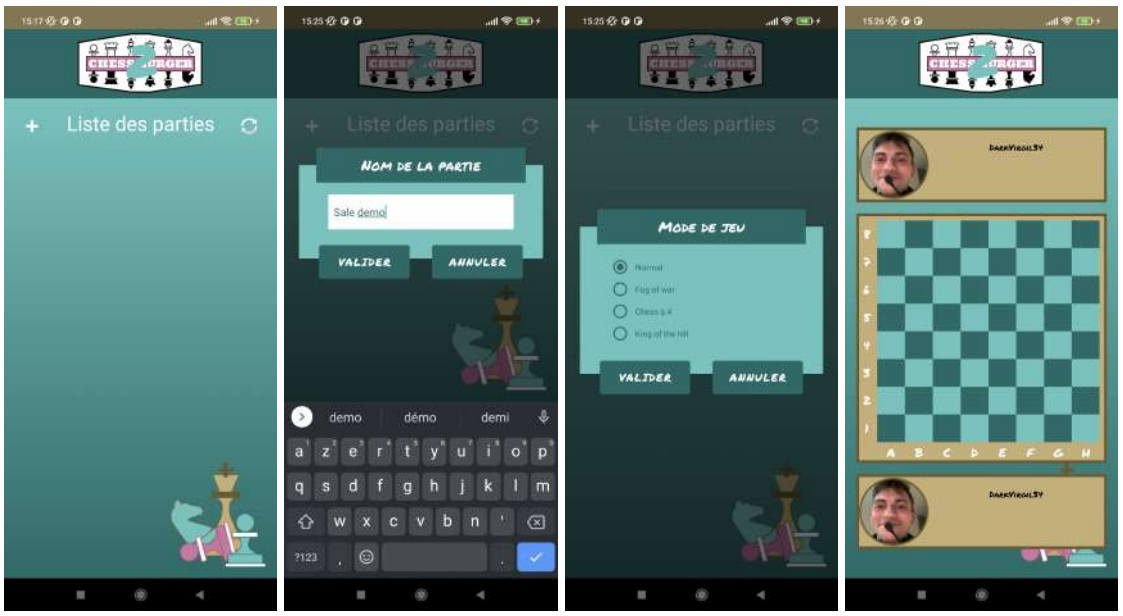
nécessite donc une vérification lors de l’appui de la touche de retour. On met donc en place un système qui nécessite d’appuyer deux fois sur la touche de retour afin de quitter la salle.

Message de confirmation lors de l’appui sur la touche de retour arrière.

Lorsque l’on quitte la partie, on perd automatiquement.

5.1.4.1. Créer une salle

À partir du menu de liste de parties, on peut créer une salle en appuyant sur l'icône de “plus” (+) en haut à gauche. Un “dialog” se présente alors pour saisir le nom de la partie, il faut ensuite valider en appuyant sur “Valider”. Un nouveau “dialog” apparaît pour sélectionner un mode de jeu. Les modes de jeu ne sont pas encore implémentés, donc ce menu ne sert à rien. On peut donc valider. La salle est alors créée et on est automatiquement mis à l'intérieur.

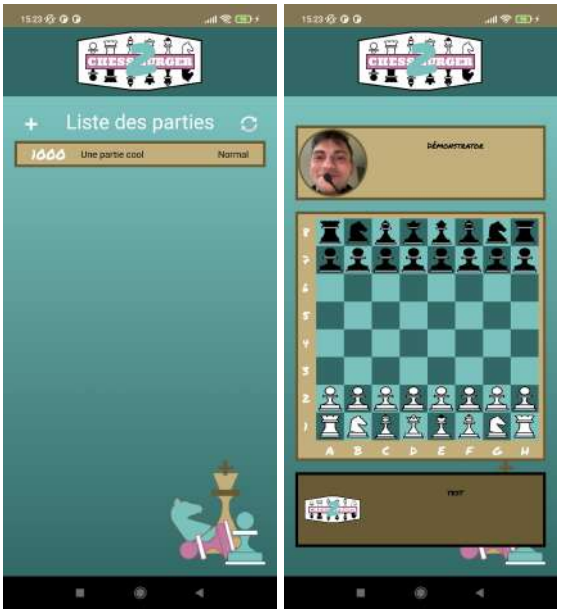


Chemin depuis la liste de partie vers la création de salle.

Une fois à l'intérieur de la salle, on est en attente qu'un autre utilisateur rejoigne. Lorsque cet autre utilisateur nous parvient, la partie démarre ; le joueur ayant créé la salle joue les blancs, donc en premier.

5.1.4.2. Rejoindre une salle

À partir du menu de liste de parties, on peut rejoindre un jeu en appuyant sur une salle de la liste.

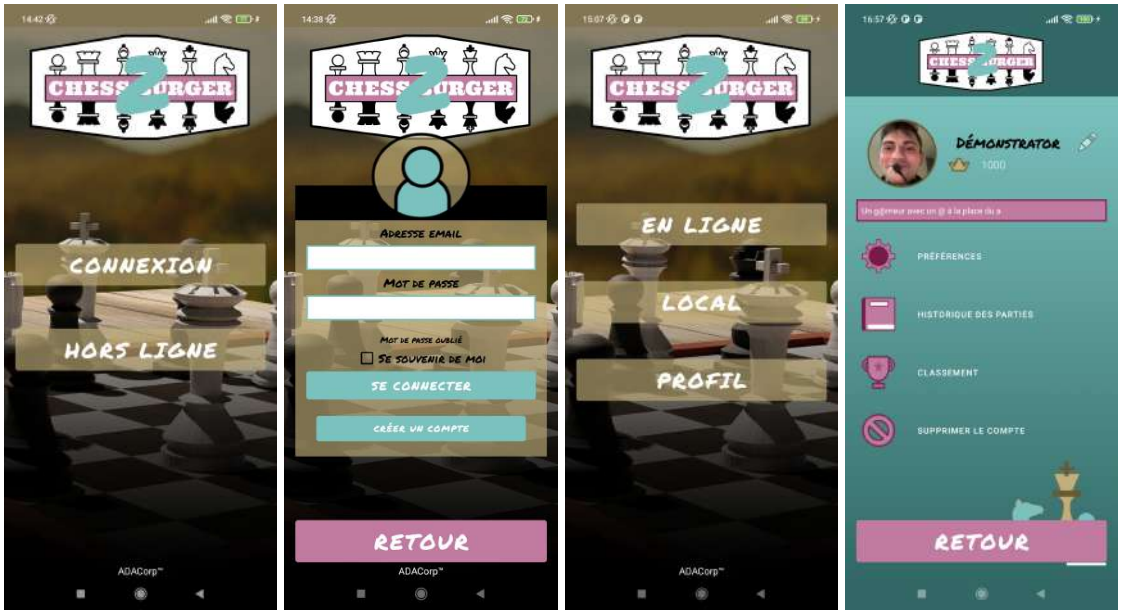


Chemin depuis la liste de partie pour rejoindre une salle.

Ainsi, on est directement en partie contre la personne qui a créé la salle ; c'est lui qui commence.

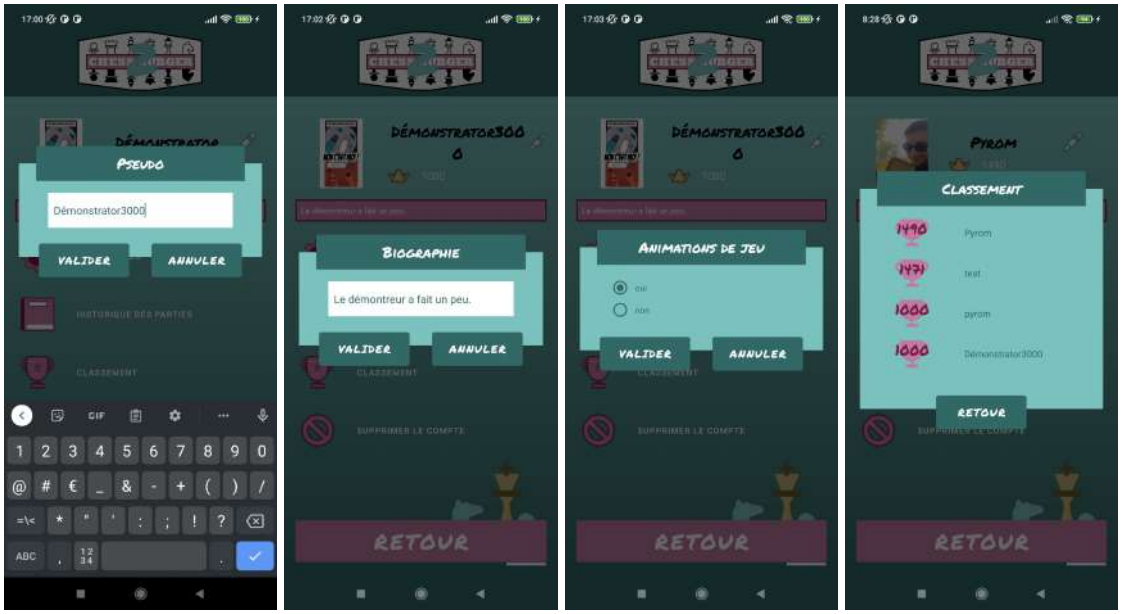
5.1.5. Modifier son profil

Pour modifier son profil, il faut, depuis le premier menu, aller dans “Connexion”, puis appuyer sur le bouton “Se connecter”. Enfin, on accède au profil via le bouton “Profil”.



Chemin vers le menu de profil.

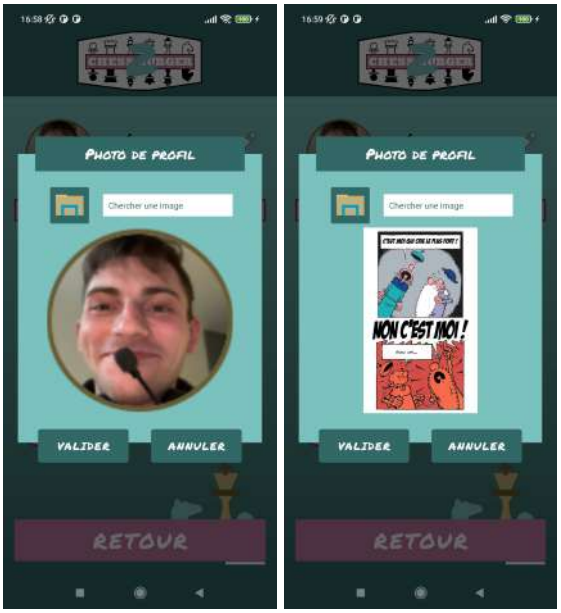
Ce menu comporte différentes fonctionnalités d’édition du profil. On peut changer de pseudo en appuyant sur son pseudo, on peut changer sa biographie de la même manière, on peut changer le mode d'animation de jeu en appuyant sur le bouton “Préférences”, on peut visualiser son classement avec le bouton “Classement”. Enfin, on peut consulter son historique avec le bouton “Historique des parties”.



Différentes fonctionnalités : édition de pseudo, biographie, préférences, visualisation du classement.

Affichage de l'historique des parties. Menu de base à gauche et menu étendu à droite.

On peut également changer d'image en appuyant sur son image. Cela ouvre un "dialog" permettant de visualiser l'image. On peut alors choisir une nouvelle image en appuyant sur le bouton icône en forme de dossier.



Changement d'image de profil avant et après la sélection d'image.

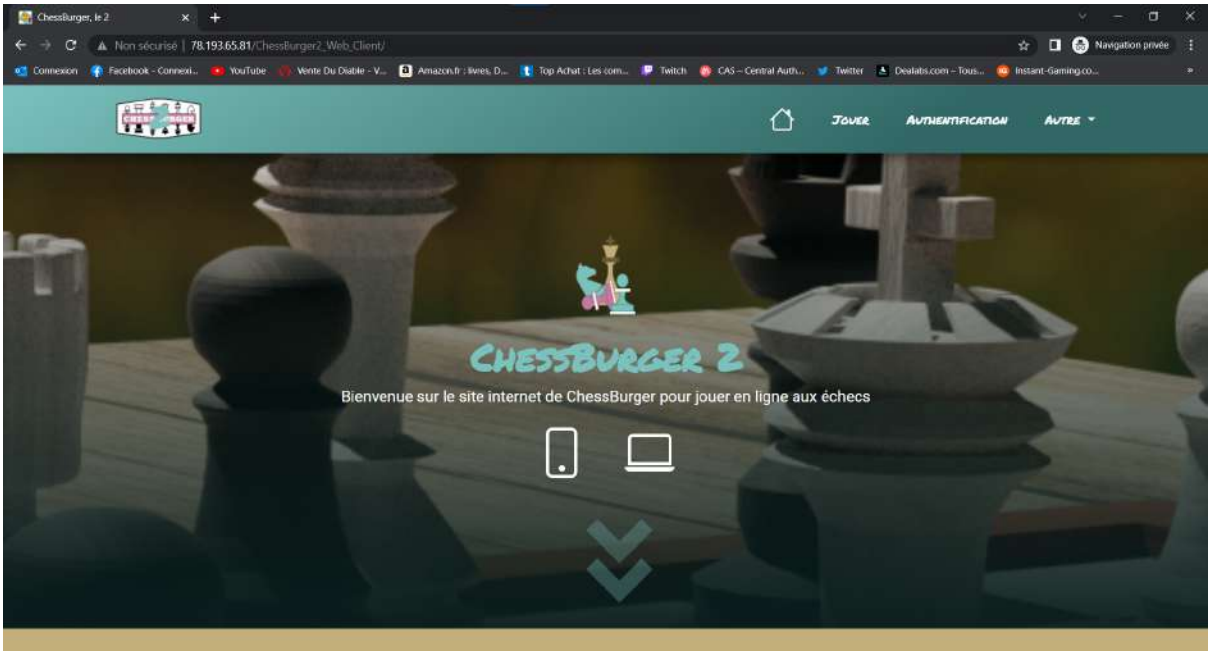
La dernière fonctionnalité de ce menu permet de supprimer le compte. Il y a bien sûr une confirmation.

Confirmation de suppression de compte.

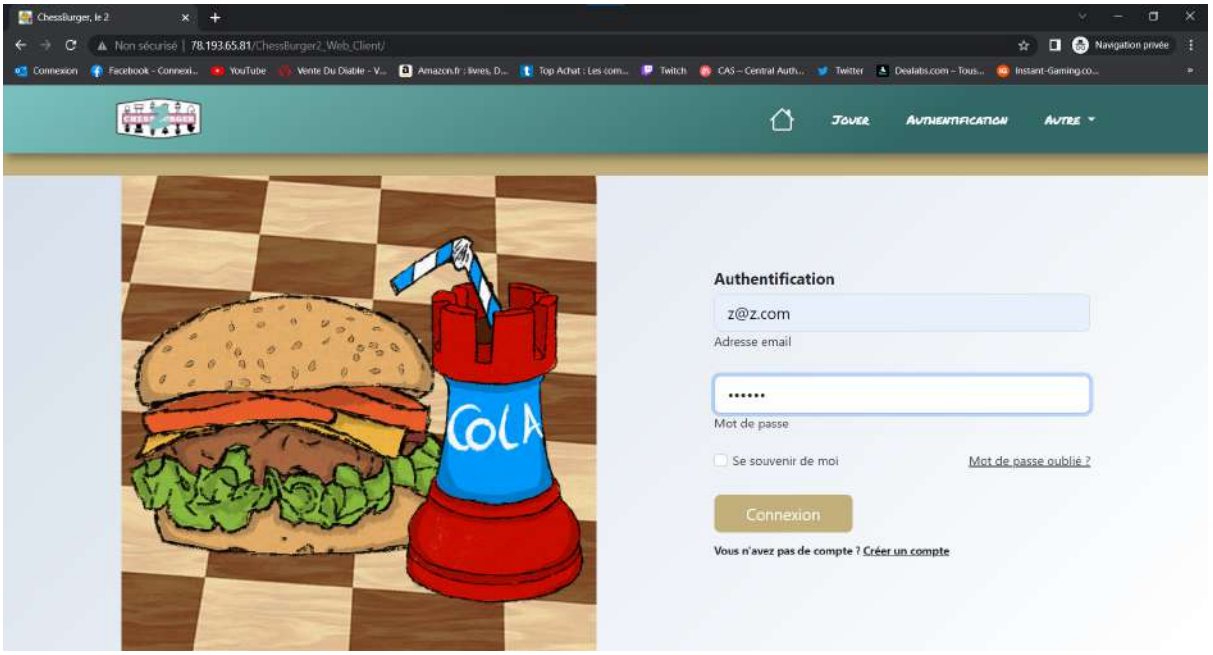
5.2. Fonctionnalités Web

5.2.1. Créer un compte

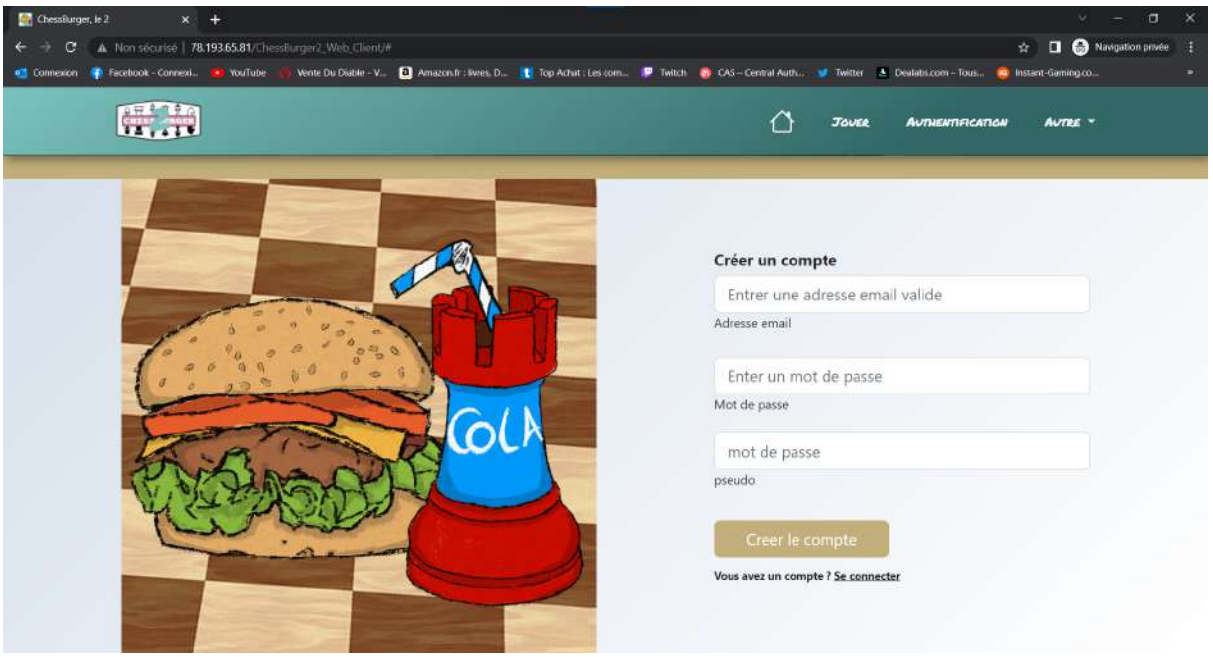
Quand l'utilisateur se connecte sur la page, il arrive sur le menu d'accueil suivant :



À ce moment, les seules actions possibles sont jouées en local (que nous verrons plus tard), se connecter et se créer un compte. Pour cela, il devra scroller vers le bas pour faire apparaître le formulaire de connexion ou de création de compte. Il va donc arriver à un moment où cet affichage va lui être proposé :

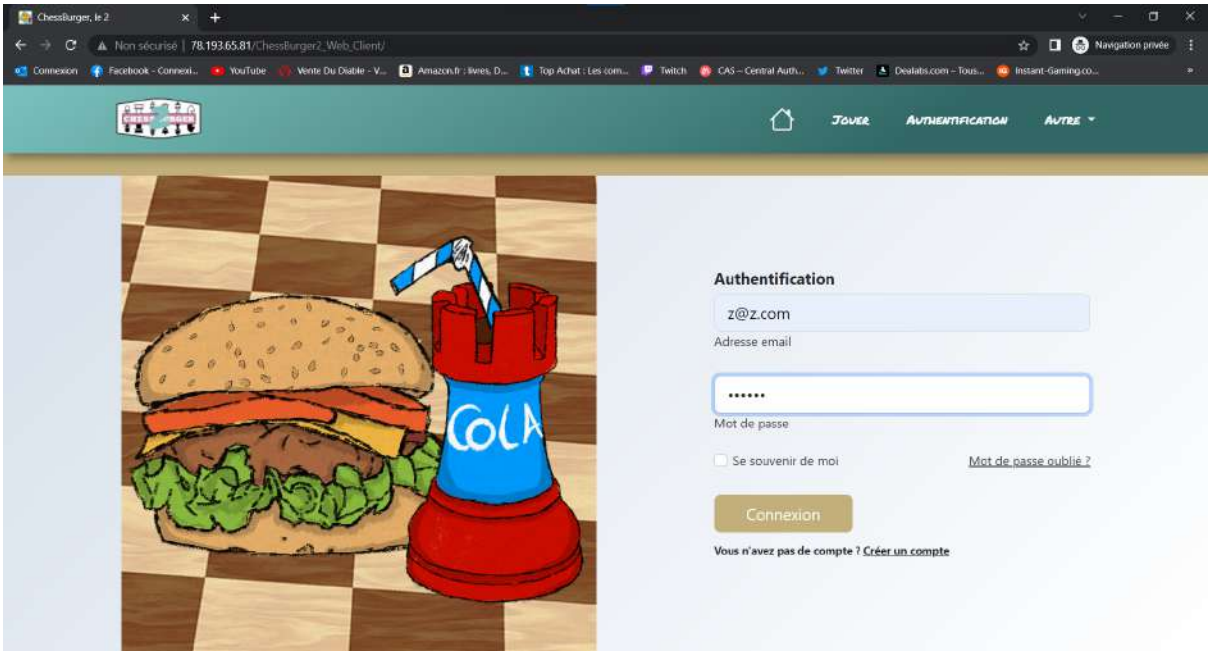


Pour créer le compte, il suffira à l'utilisateur de cliquer sur "Créer un compte" se situant en dessous du bouton connexion pour faire apparaître le formulaire de création de compte suivant.

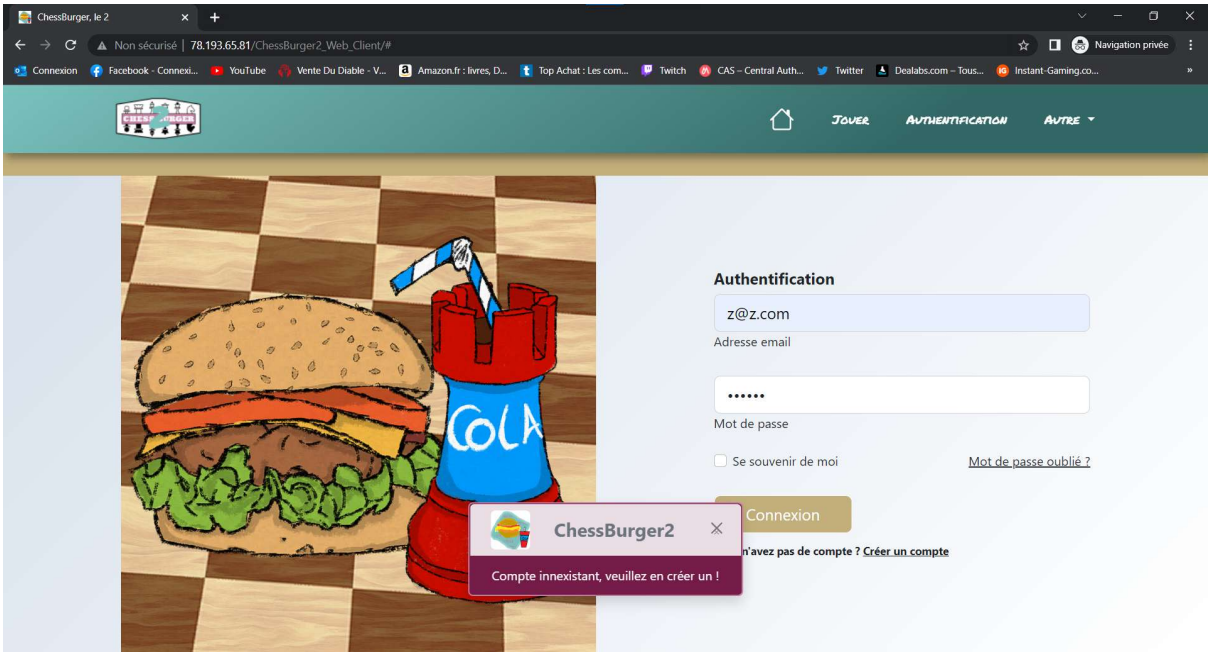


5.2.2. Se connecter

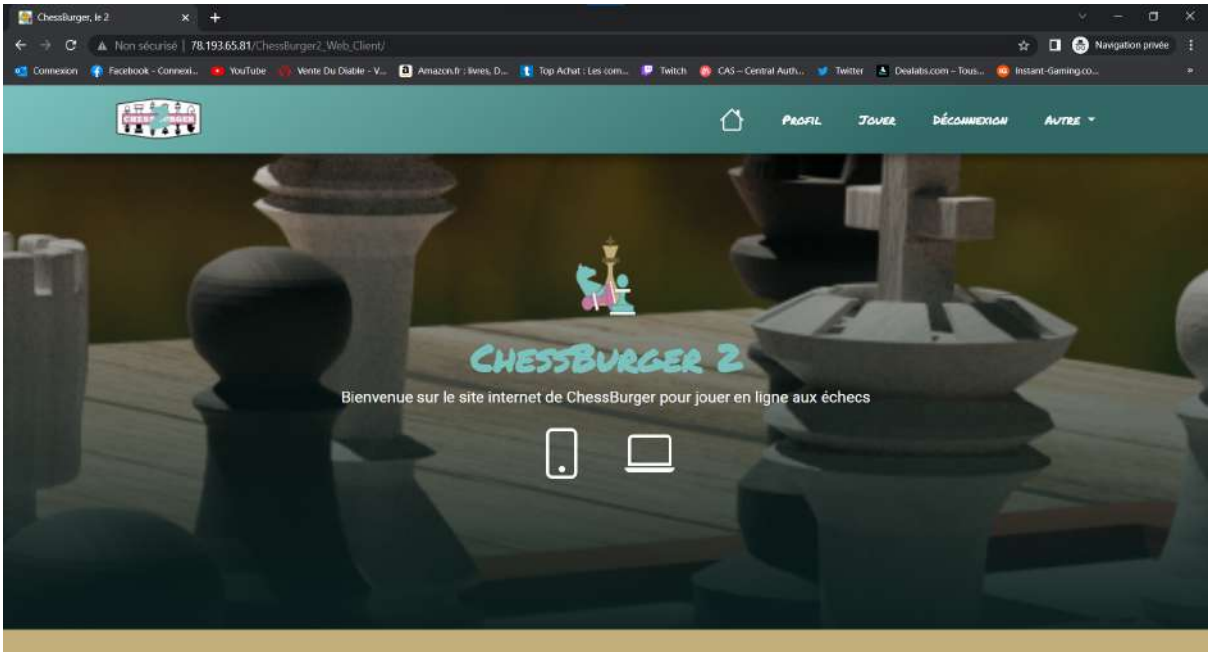
Ici, il suffit à l'utilisateur de faire la même chose qu'auparavant, sauf qu'il ne cliquera pas sur "Créer un compte" mais remplira seulement les informations de son compte comme après.



Si ses informations sont erronées, un toast apparaîtra alors pour le notifier.



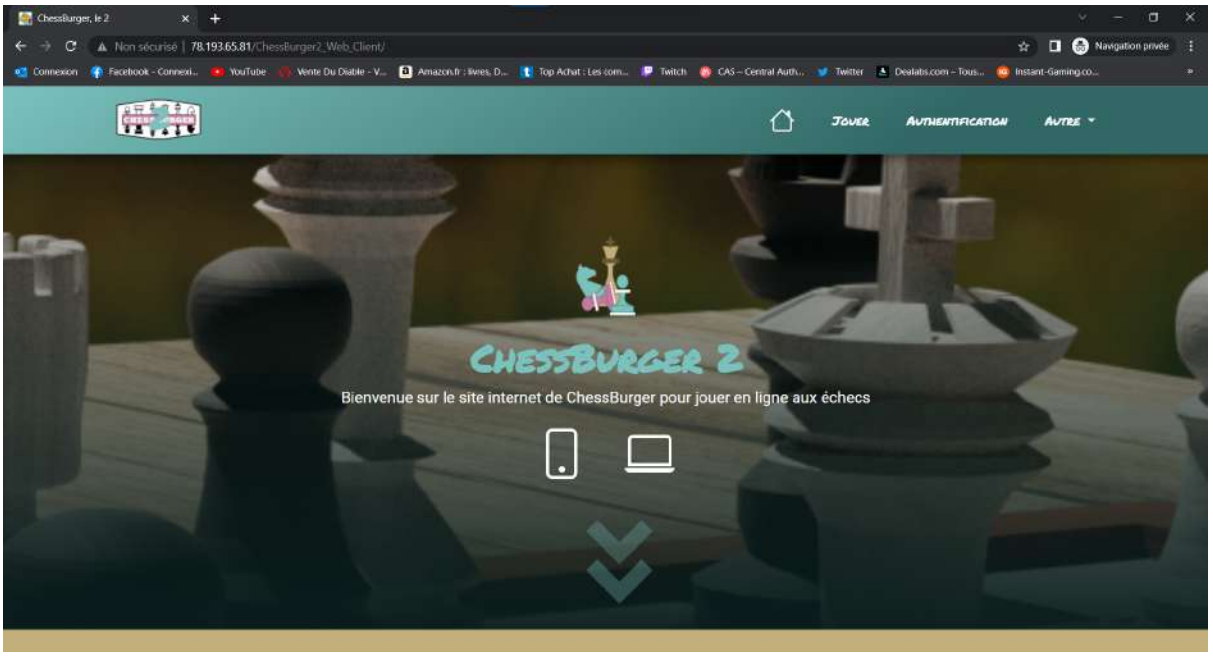
Si ces informations sont correctes, alors il sera directement amené sur la page suivante.



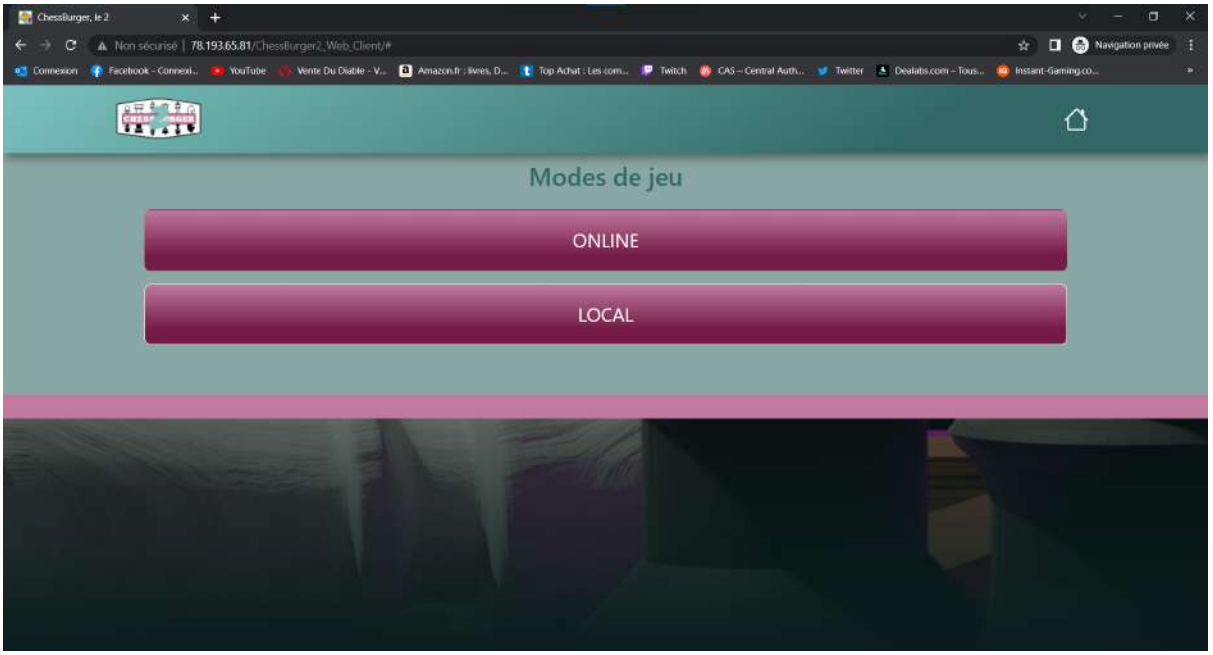
5.2.3. Jouer une partie en local

Si le joueur veut jouer une partie en local, deux solutions s'offrent à lui :

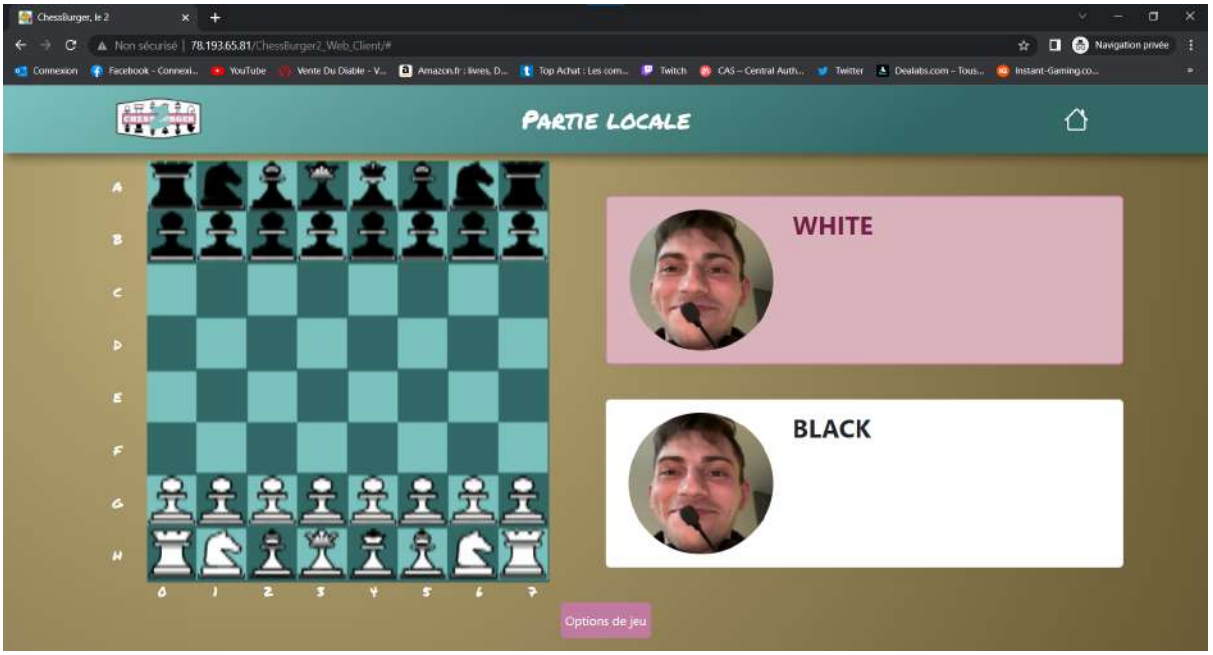
- Soit il clique sur jouer sans même se connecter en haut à droite



- Soit il se connecte comme expliqué avant, puis clique sur jouer au même endroit pour être amené sur cette page.



Et ici, il devra alors cliquer sur le bouton local pour finalement arriver sur la page de jeu en local.

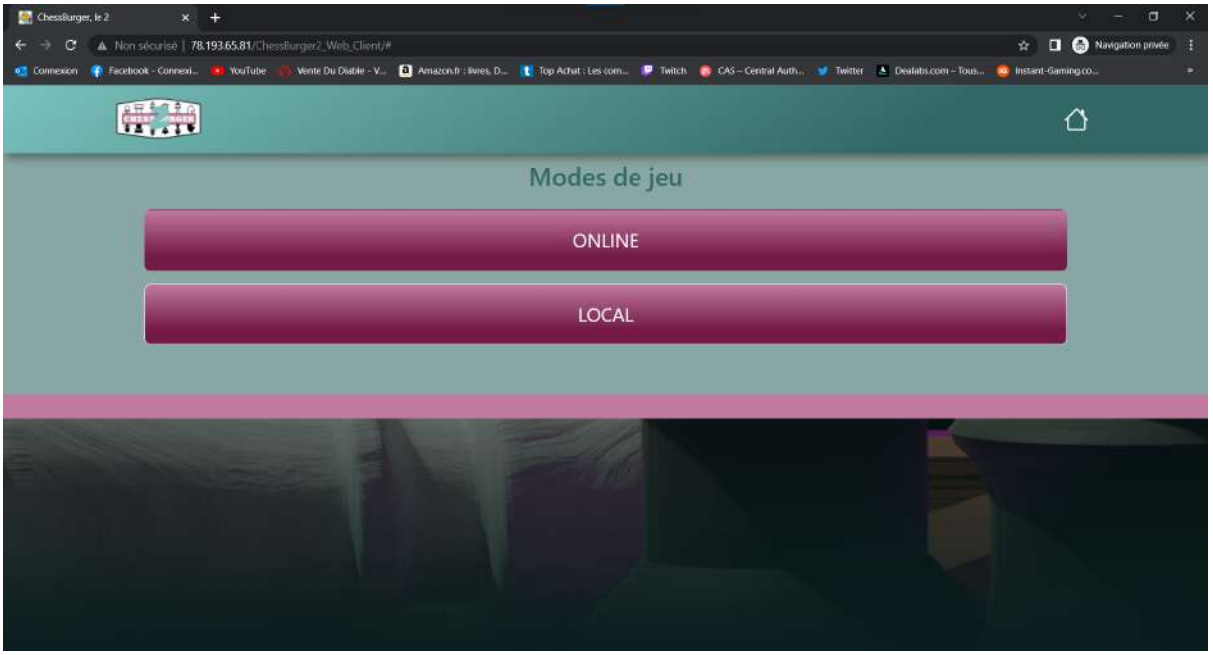


5.2.4. Jouer une partie en ligne

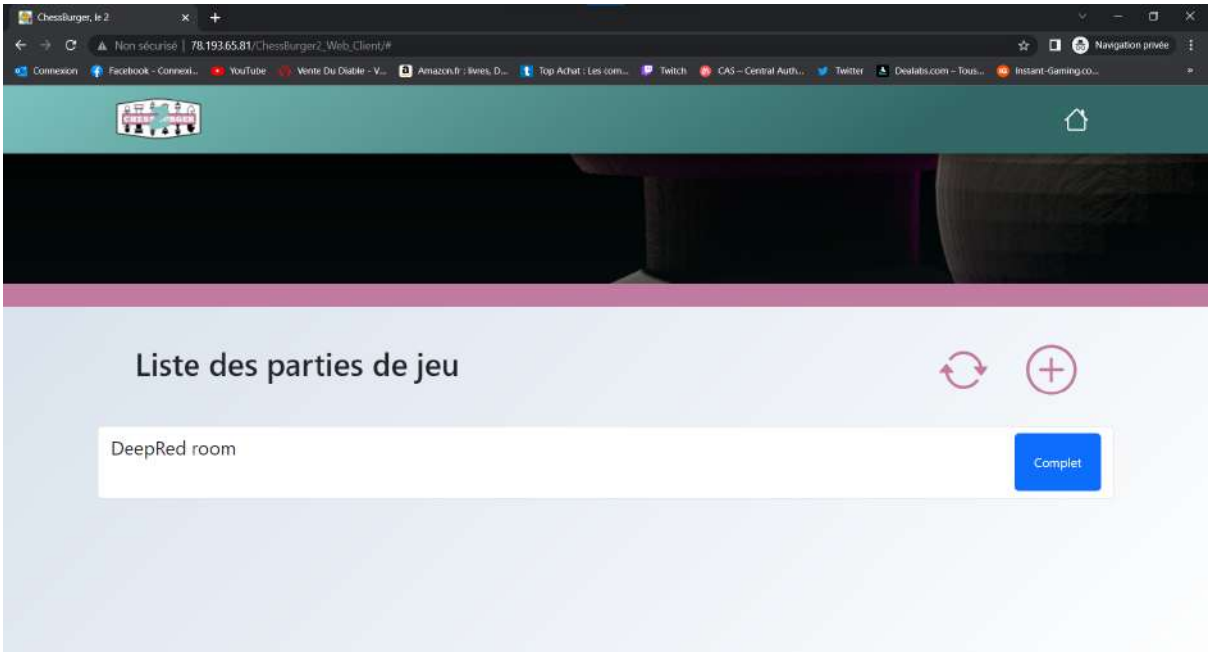
Ici par contre, l'utilisateur doit forcément se connecter pour pouvoir effectuer les actions qui suivent.

5.2.4.1. Créer une salle

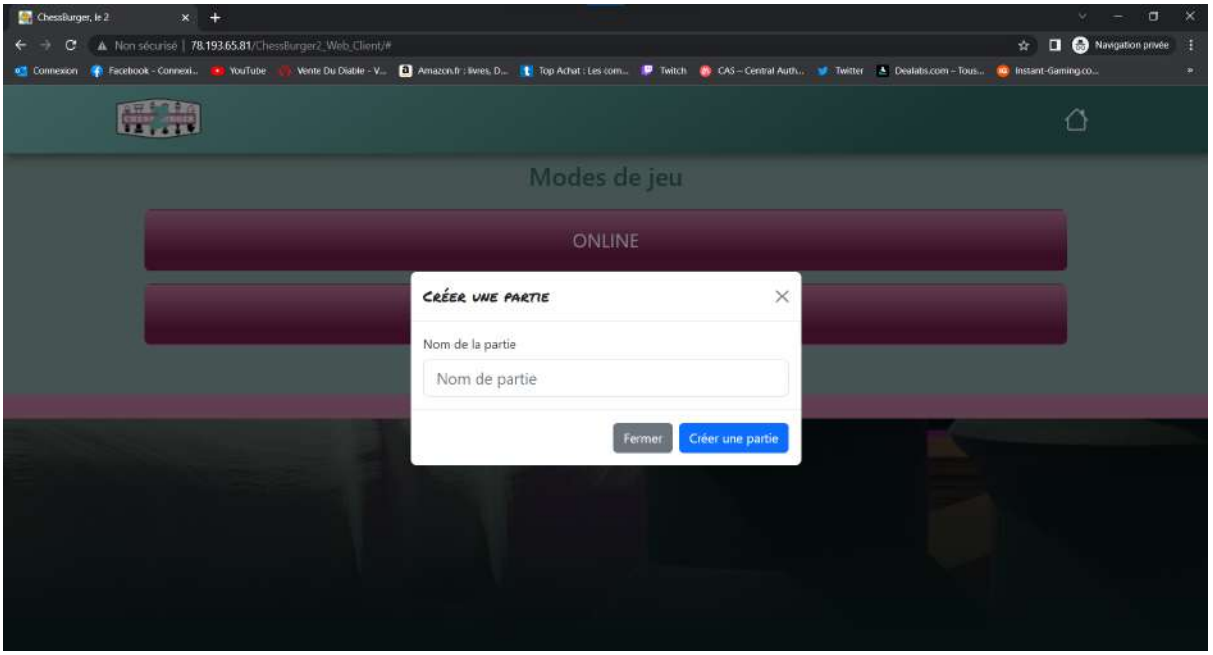
En étant sur la page après sa connexion, il va alors effectuer l'action pour jouer. Mais au lieu de cliquer sur "local", cliquera sur "online".



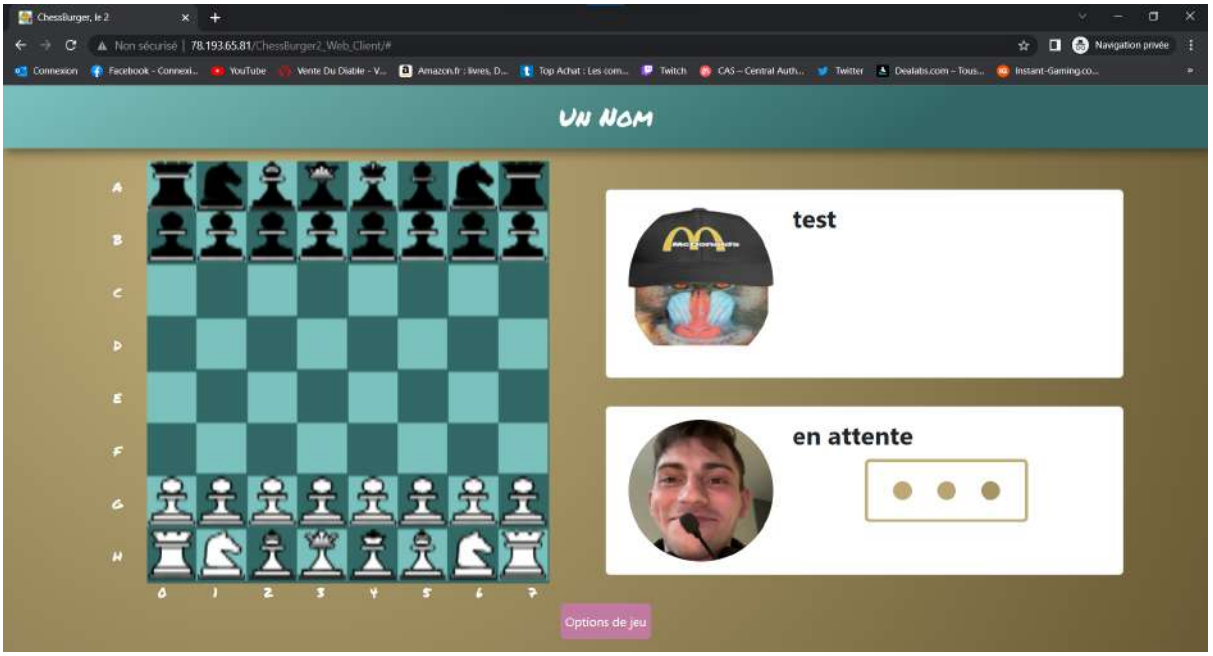
Ce qui va l'amener ici.



Ici, pour créer une partie, il faut cliquer sur le plus à droite de l'écran ce qui fera apparaître le dialogue suivant demandant à l'utilisateur de rentrer un nom à sa partie pour qu'elle soit plus facilement identifiable par les autres joueurs.



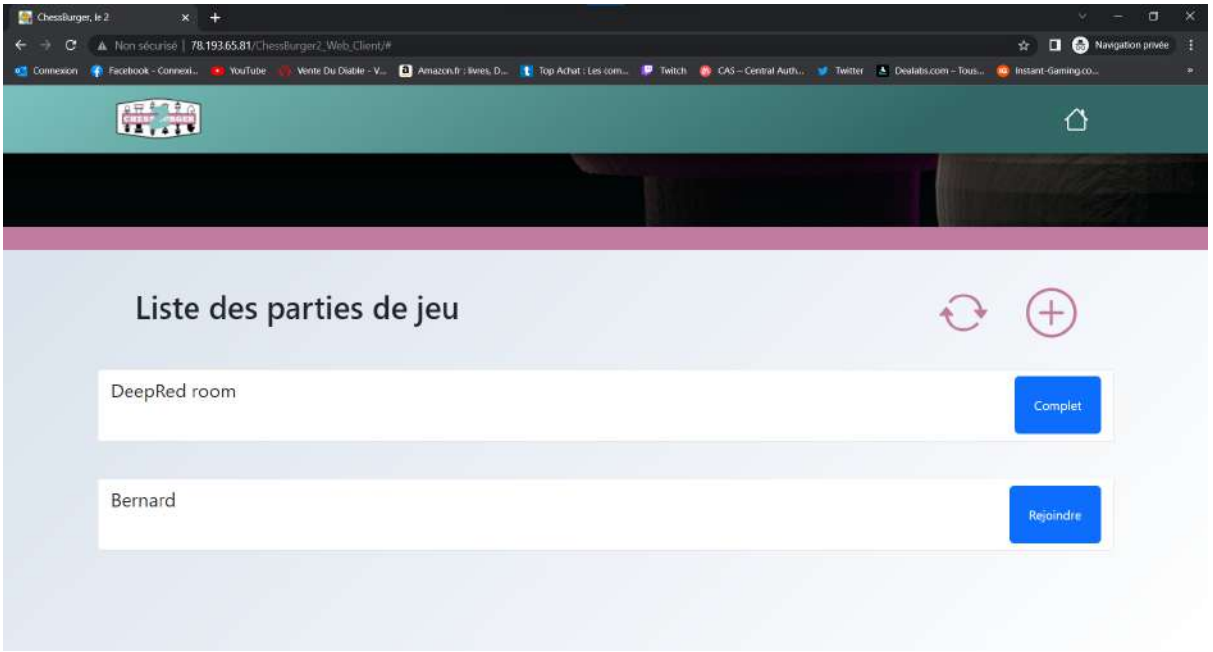
Puis l'utilisateur rentre alors le nom de sa partie pour être amené ici.



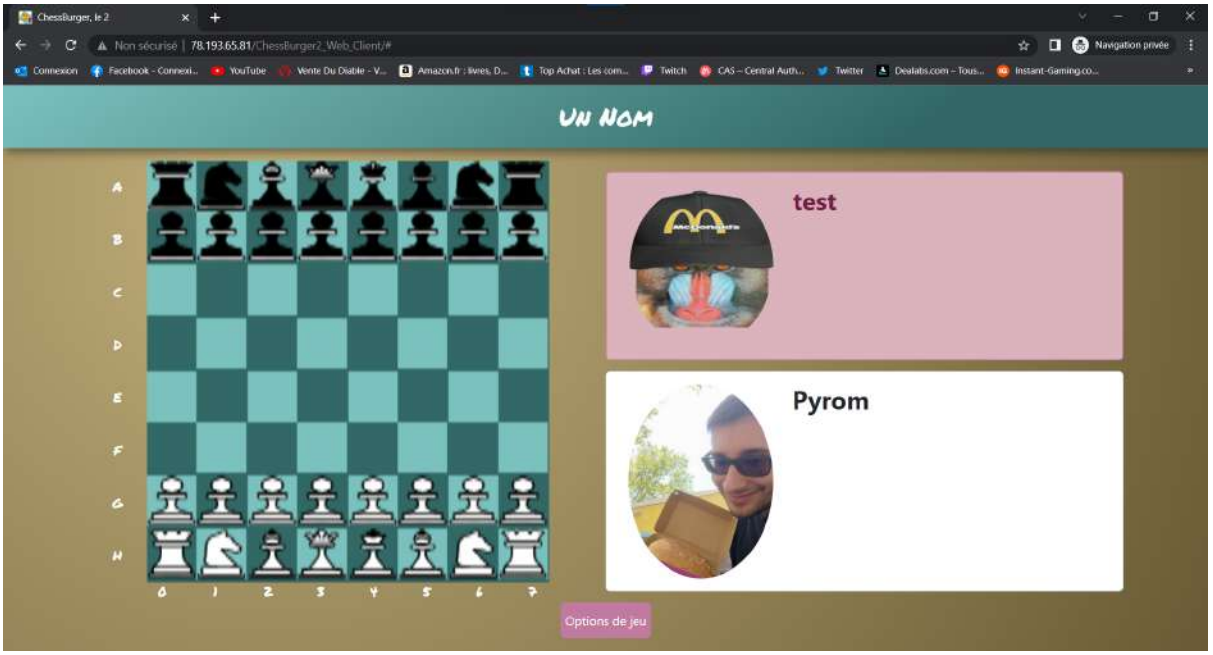
Alors à ce moment, le joueur doit attendre l'arrivée d'un autre joueur pour pouvoir commencer sa partie est jouée comme en local.

5.2.4.2. Rejoindre une salle

Pour rejoindre une partie, il faudra effectuer le click sur le bouton “online”, sauf qu’au contraire de créer une partie, il faudra cliquer sur le bouton à gauche de plus, qui va rafraîchir la liste des parties disponibles.

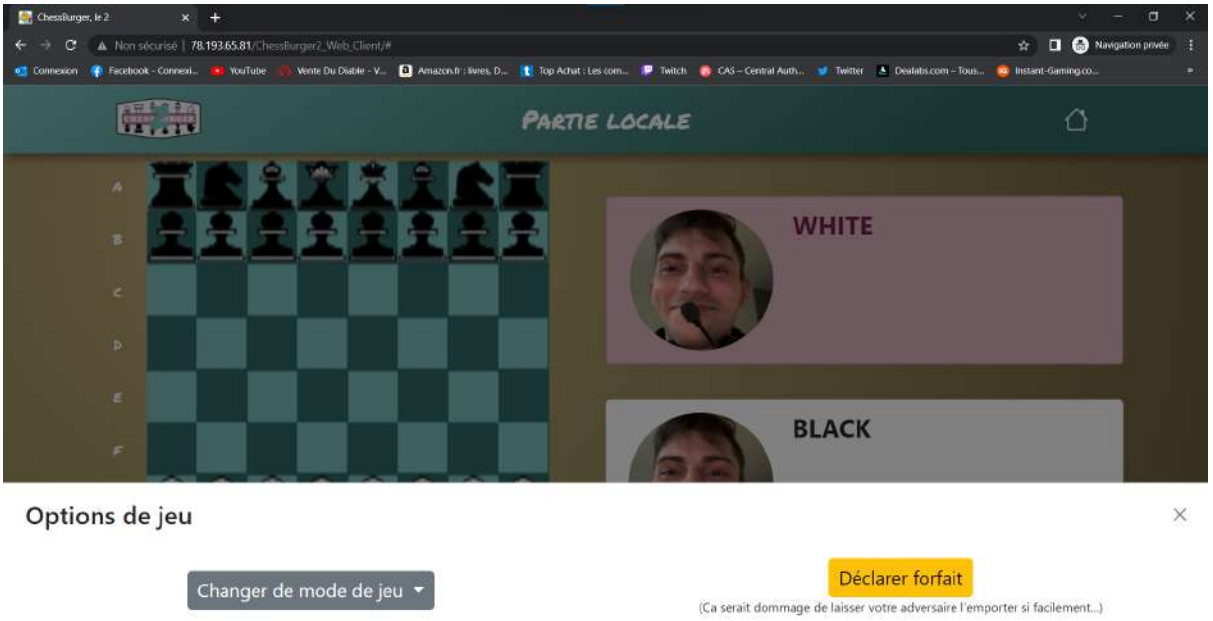


Il devra alors cliquer sur le bouton rejoindre à droite de la partie ce qui l'amènera dans la partie avec le joueur se trouvant déjà dans la room.



5.2.4.3. Abandonner une partie

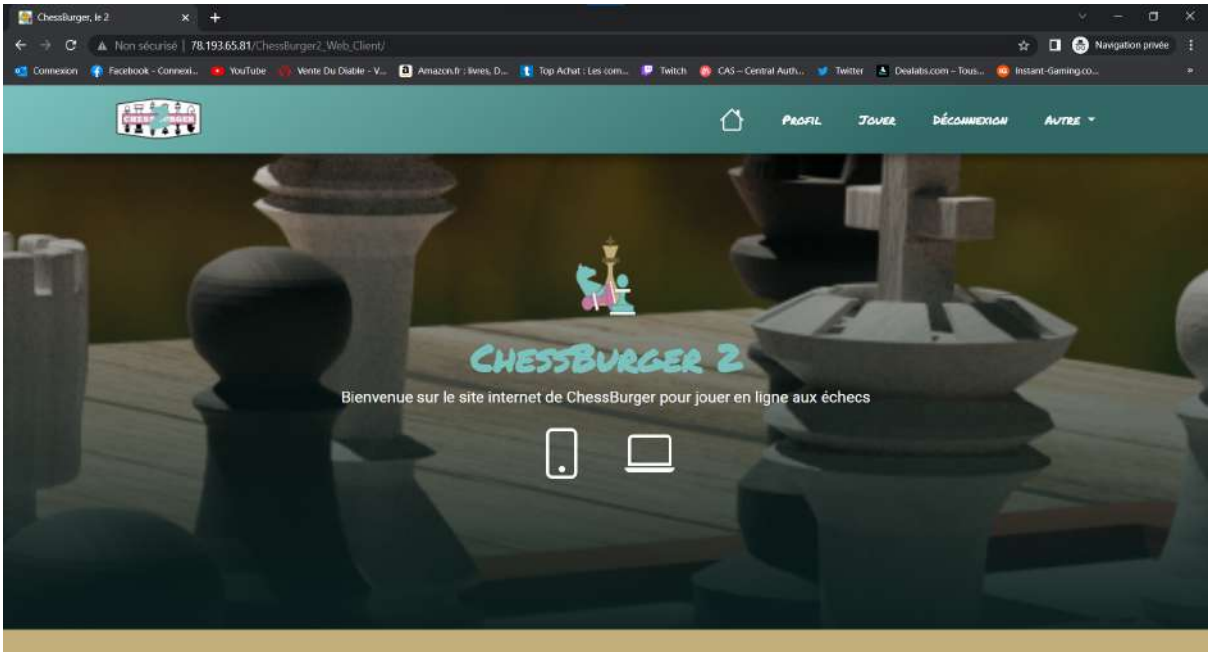
Pour abandonner, il faut donc se mettre dans une partie puis cliquer sur les options de jeux au milieu en bas de l'écran qui ouvre un dialogue qui permet au joueur s'il le souhaite d'abandonner la partie.



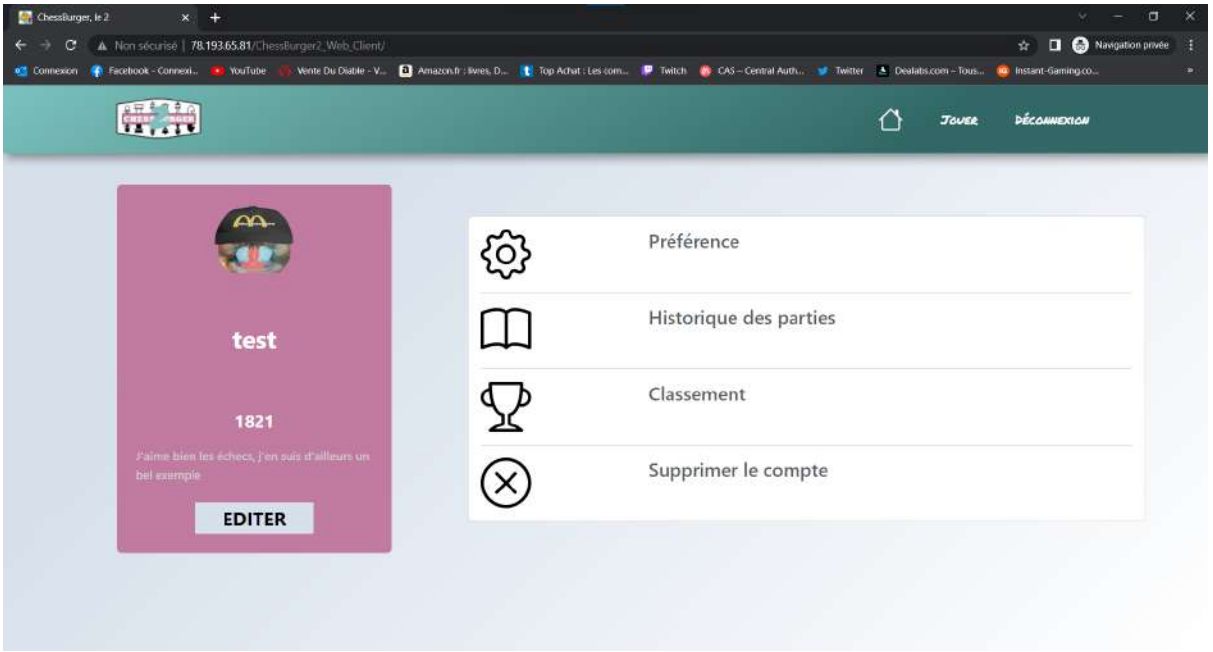
5.2.5. Modifier son profil

Pour modifier ou voir ses informations de profil. Il faut que l'utilisateur soit connecté à son compte. Puis quand il est sur la page d'accueil, clique sur le bouton profil en haut à droite de son écran.

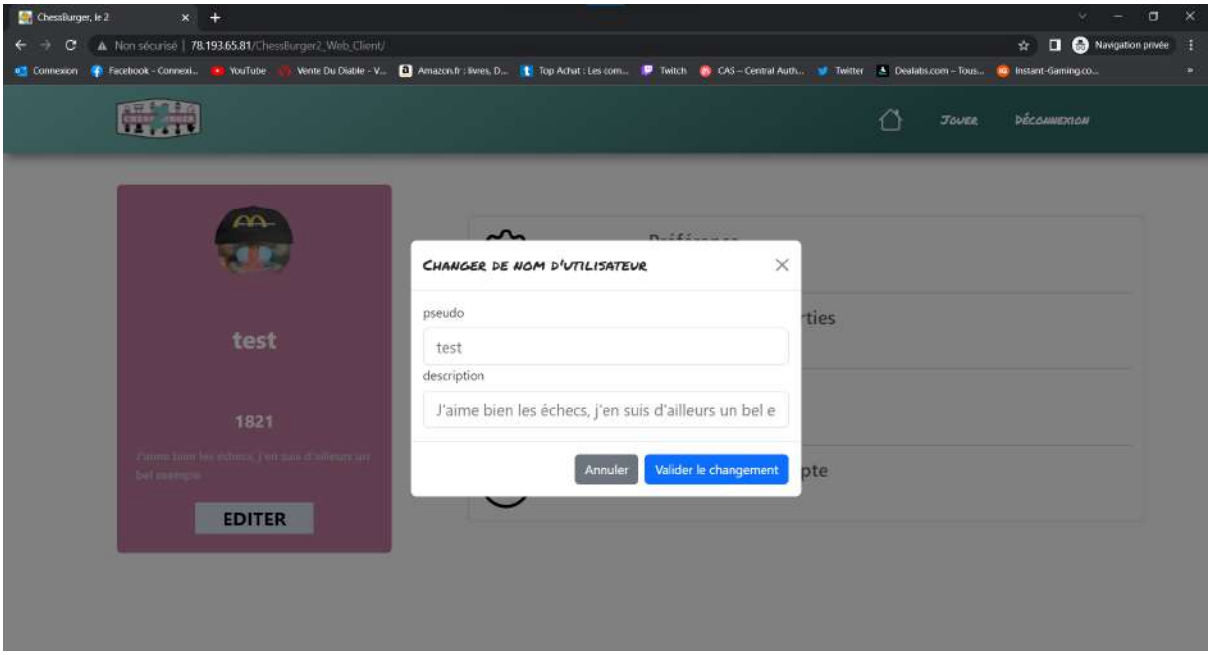
.



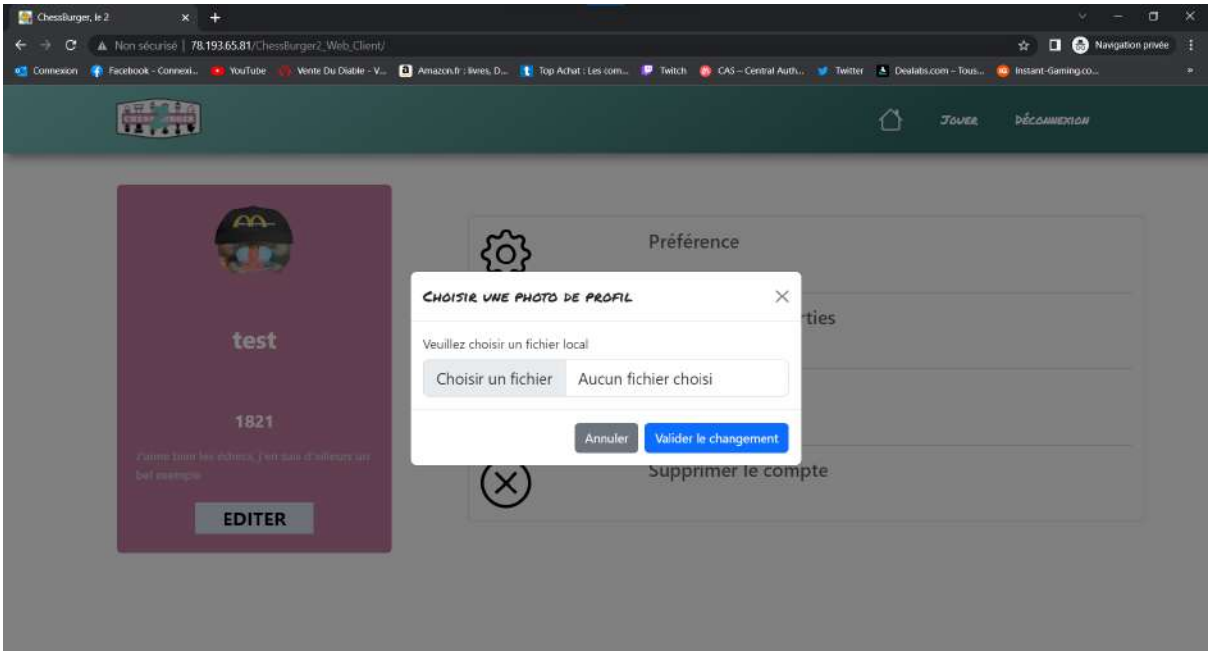
Puis à ce moment, il sera alors conduit sur la page suivante qui contient toutes ses informations.



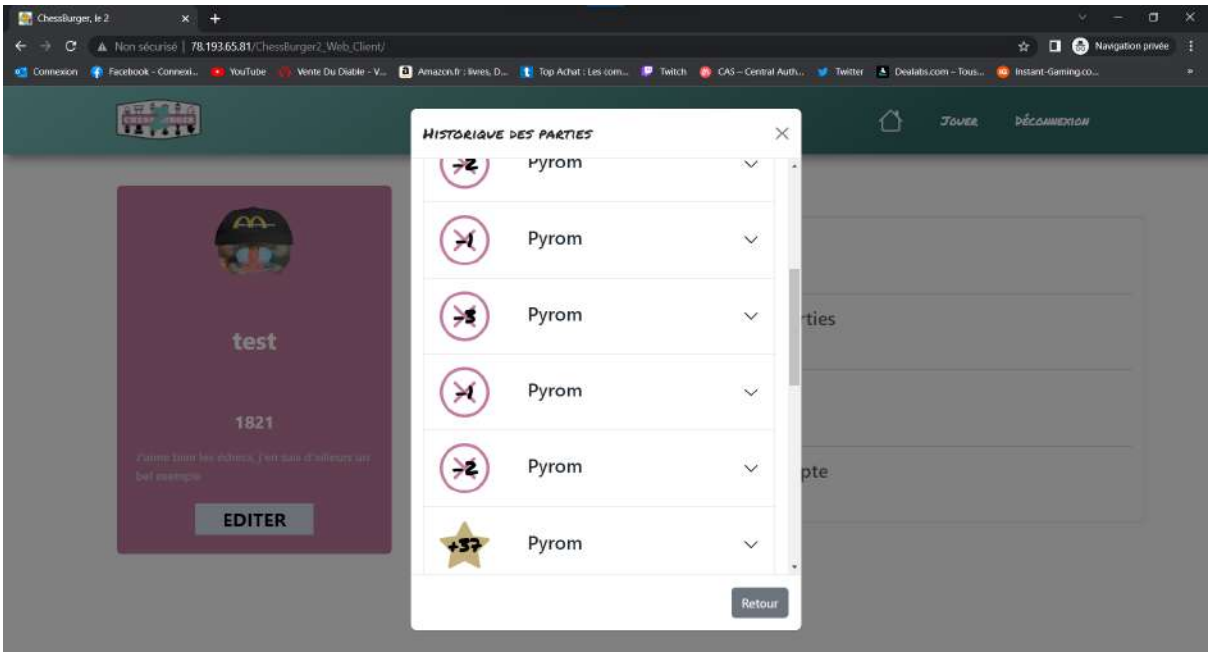
Ici, il pourra alors modifier ses informations telles que son pseudo et sa biographie en cliquant sur le bouton “éditer”.



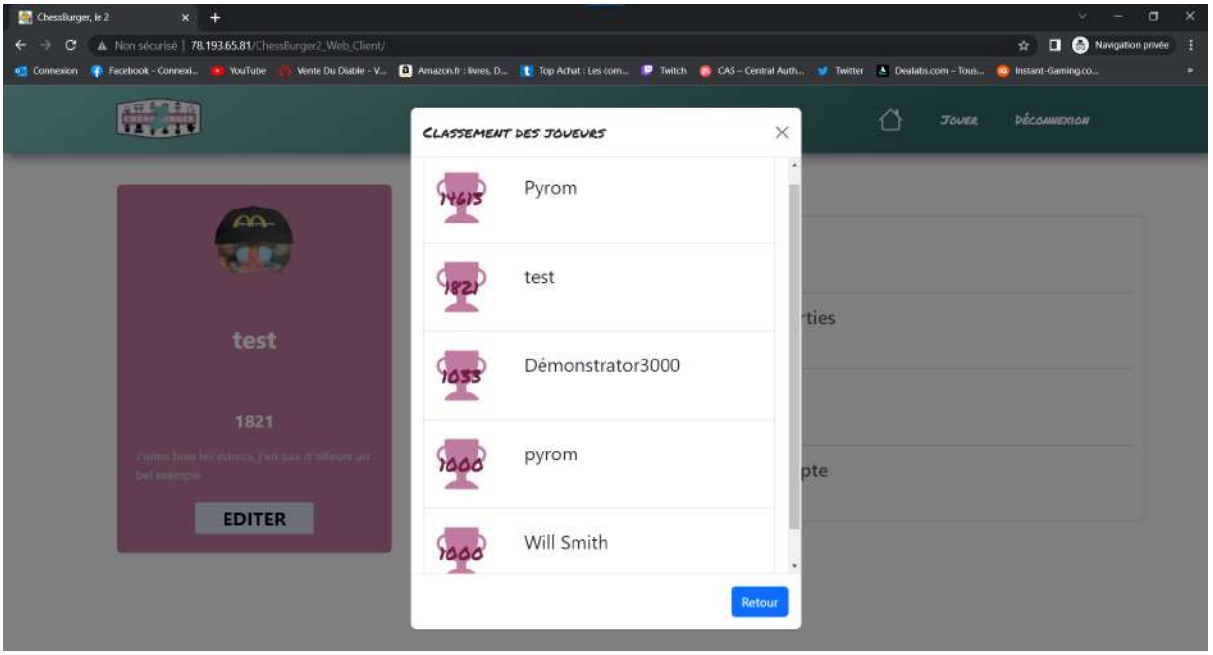
Ou encore sa photo de profil en cliquant sur sa photo puis en choisissant la photo voulue se situant sur son ordinateur.



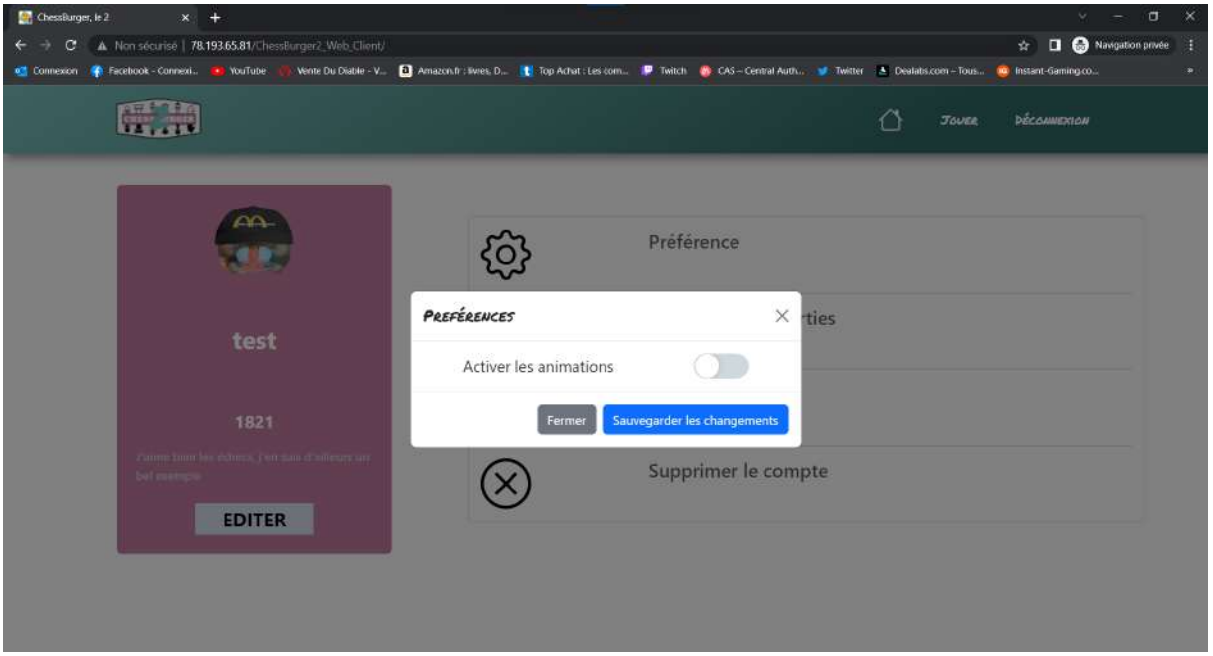
S'il veut voir son historique de partie, il lui suffit alors de cliquer sur "historique de parties" afin de lui montrer contre qui il a joué, s'il a gagné/perdu, combien de "élo", en combien de coups, etc.



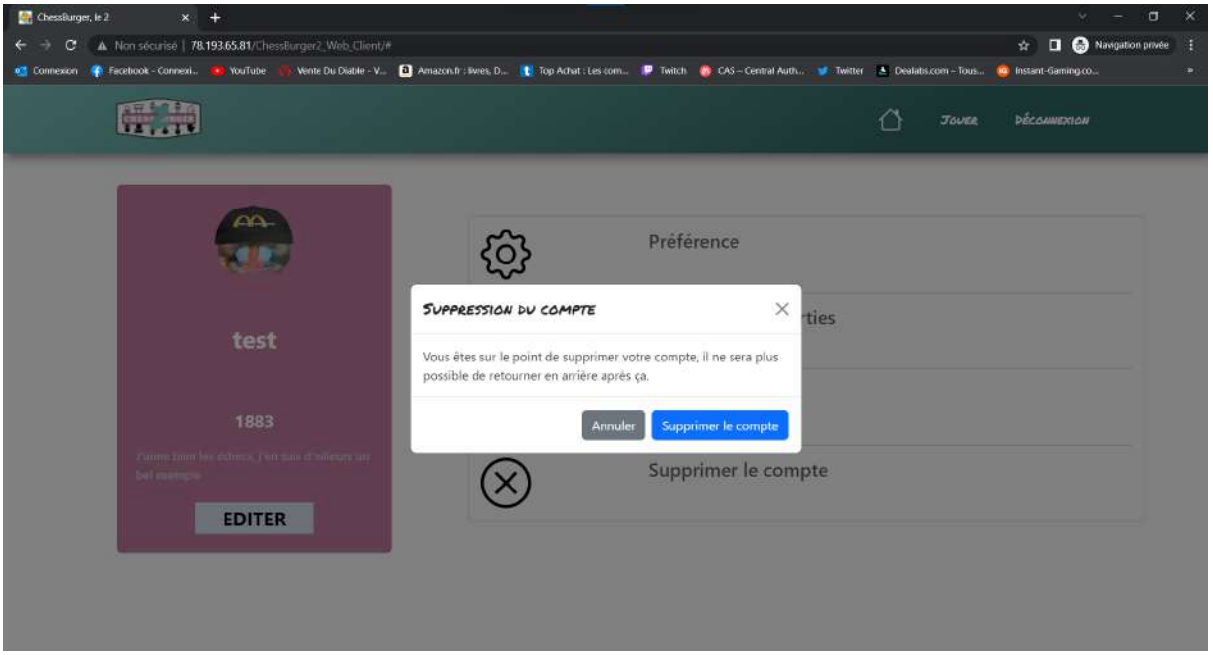
Il peut aussi afficher le classement de tous les joueurs pour se comparer à eux. En cliquant sur le bouton classement.



Ou encore afficher ses préférences afin d'afficher ou non en jeux les animations.



Et finalement, il peut supprimer définitivement son compte, cela amènera la destruction totale de toutes ses informations. Il devra alors cliquer sur Supprimer son compte et valider.


ChessBurger, le 2
Non sécurisé | 78.193.65.81/ChessBurger2_Web_Client/#
Navigation privée
Connexion
Facebook - Connexi...
YouTube
Vente Du Diable - V...
Amazon.fr : livres, D...
Top Achat : Les com...
Twitch
CAS – Central Auth...
Twitter
Dealabs.com – Tous...
Instant-Gaming.co...
JOUER
DÉCONNEXION
test
1883
EDITER
Préférence
Supprimer le compte
SUPPRESSION DU COMPTE
Vous êtes sur le point de supprimer votre compte, il ne sera plus possible de retourner en arrière aprés ça.
Annuler
Supprimer le compte

6. Rétrospective

Nous avons développé une application de jeu d'échecs en ligne et multi-plateforme ; ce qui représente déjà un grand accomplissement. Mais il existe différentes implémentations que l'on voudrait modifier. Par exemple, on peut concevoir une meilleure organisation des méthodes d'accès à la base de données Firebase, on peut même encapsuler tous les appels à Firebase dans une structure de classes. De la même manière, on peut davantage séparer les comportements des vues de plateau de jeu et de son affichage, ce qui nous permettrait d'accéder à des fonctionnalités plus facilement, et donc de par exemple faire une méthode pour abandonner la partie. Sur l'aspect visuel du plateau de jeu côté client web, la qualité des images reste assez faible (dû au canvas). Si on avait voulu l'améliorer, on aurait dû partir sur une implémentation différente et mettre les images directement dans des balises HTML par exemple (HTML supporte nativement les images au format SVG).

Certaines fonctionnalités n'ont pas pu être complètement implémentées par manque de temps. Même si notre site web possède énormément de charisme, il y a quelques fonctionnalités que l'on voudrait améliorer. Par exemple, l'affichage du plateau de jeu peut être grandement amélioré. En effet, pour une raison inconnue, il ne fonctionne pas sur le navigateur Firefox. De plus, l'implémentation des parties classées nécessite de concevoir un système de matchmaking permettant de trouver deux joueurs ayant le même niveau. Enfin, certains problèmes subsistent dans cette application, il serait évident de prendre le temps de résoudre ces problèmes.

7. Perspective

Les perspectives que l'on avait prévues et pensées pour ce projet seraient d'implémenter des modes de jeux ainsi qu'un interpréteur de règles. C'est-à-dire que l'utilisateur peut créer des règles pour une partie les enregistrer s'il le veut et pouvoir y jouer avec d'autres joueurs. Cela permettrait par la suite de jouer à des variantes telles que "King of the hill" ou encore "Horde", etc.

De plus, il faudrait qu'un système de notification soit mis en place afin de notifier quand des parties sont créées, pouvoir inviter d'autres joueurs à jouer dans sa room, etc. Cela permettrait d'avoir une application plus interactive.

Un autre point important serait la sécurité. De nos jours, le monde de l'informatique est assujetti à des attaques, il serait donc important que l'application soit parée à ce genre d'attaques éventuelles.

Aussi, un système de communauté/ami serait prévu, permettant plus facilement d'inviter d'autres joueurs à jouer. Cette fonctionnalité est pas mal liée aux notifications, car quand un joueur en invite un autre, il faut le notifier.

8. Annexes

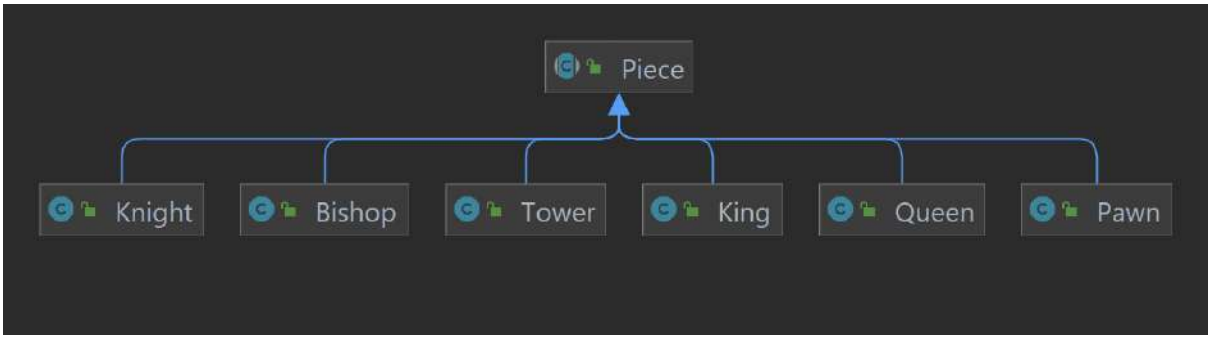


Diagramme de classe Pièces

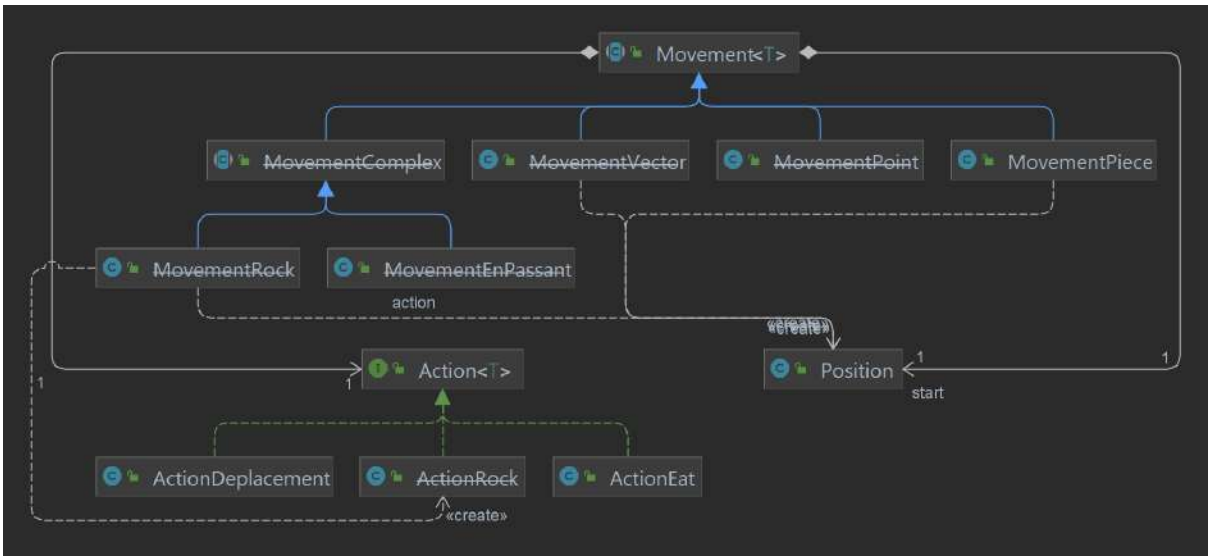


Diagramme de classes Mouvement

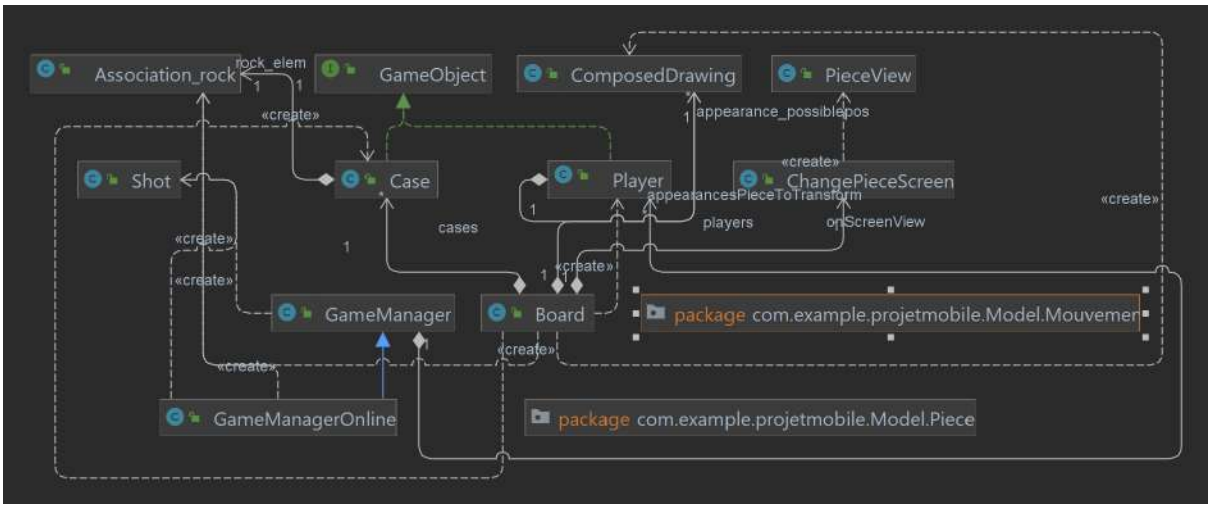


Diagramme de classe Model

8.1.1. Sitographie

Client WEB :

Dépot GITHUB : https://github.com/virgil-rouquettecampredon/ChessBurger2_Web_Client

Site Boostrap : https://getbootstrap.com/
Tutoriels pour la programmation WEB : https://www.w3schools.com/bootstrap/
Exemples de code WEB : https://codepen.io/
Génération de palettes de couleurs : https://coolors.co/

Client Android :

Dépot GITHUB : https://github.com/virgil-rouquettecampredon/ProjetMobile

Doc : https://developer.android.com/docs
Aides diverses: https://stackoverflow.com/