

**Département Informatique**

 71 rue Peter Fink

01000 Bourg-en-Bresse

 04 74 45 50 59

*Roberjot Aloïs, Bodin Mathis, Caron Maxence, Namani Samuel*

Promotion 2023/2024

Mahjong Calculator

Manuel d’utilisation

Fabrice Jaillet

Diplôme Universitaire de Technologie

Informatique

|  |
| --- |
| [Nom de la société] |
|  |

Parcours IEM :   
Informatique Et Mobilité

Table des matières

[I. Contexte 0](#_Toc162364437)

[I.1 État de l’application fin semestre 1 0](#_Toc162364438)

[I.1.1 Détection des tuiles 0](#_Toc162364439)

[I.1.2 Problèmes de performance et d’optimisation 0](#_Toc162364440)

[I.1.3 Manque de fonctionnalités 0](#_Toc162364441)

[I.1.4 Compatibilité et responsivité 0](#_Toc162364442)

[I.2 Améliorations et changement prévues 0](#_Toc162364443)

[II. Amélioration 1](#_Toc162364444)

[II.1 Améliorations terminées 1](#_Toc162364445)

[II.1.1 Optimisation du code JavaScript 1](#_Toc162364446)

[II.1.2 Optimisation du code Java 2](#_Toc162364447)

[II.1.3 Amélioration de la détection des tuiles 2](#_Toc162364448)

[II.1.4 Ajout des règles 2](#_Toc162364449)

[II.1.5 Gestion d’une base de données 2](#_Toc162364450)

[II.1.6 Refonte de l’interface utilisateur 3](#_Toc162364451)

[II.2 Améliorations en cours 3](#_Toc162364452)

[II.2.1 Editeur de règles 3](#_Toc162364453)

[II.2.2 Algorithme de cluster encore possible à améliorer 3](#_Toc162364454)

* Changement :
  + Suppression drag and drop
  + Sélection des tuiles sur le site
* Améliorations :
  + Terminé
    - Refactoring du code JS/Java
    - Règles du jeu
    - Détection d’image
    - Création base de données / algo de connexion / Page de login site
    - Correction de bugs
  + En cours
    - Utilisation de la base de données pour l’enregistrement des scores et des règles
    - Détection d’image
* Conclusion

# Contexte

## État de l’application fin semestre 1

L’application que nous avions développée à la fin du semestre 1 est un site web visant à fournir un calculateur de score pour le jeu de Mahjong. A ce moment, nous avions une application fonctionnelle mais comportant plusieurs lacunes.

### Détection des tuiles

L’une des principales limitations de l’application était la fiabilité de la détection automatique des tuiles. Bien que fonctionnelle, cette fonctionnalité n’était pas toujours précise à 100%, ce qui pouvait entraîner des erreurs dans les tuiles retourner dans l’application.

//ajouter des exemples de type de photos avec des erreur/fond blanc etc…

### Problèmes de performance et d’optimisation

Nous avions également rencontré des problèmes de performance et d’optimisation dans le code de l’application (du coté du JavaScript et du JAVA). Cela se manifestait par des temps de chargement plus long (même si déjà très court) et une expérience utilisateur parfois moins fluide.

### Manque de fonctionnalités

En plus de ces problèmes techniques, notre application manquait de certaines fonctionnalités essentielles. Par exemple, il n’y avait pas beaucoup de règles disponibles pour les joueurs sur le site, il n’y avait pas d’éditeur de règles pour la possibilité d’ajout des propres règles des utilisateurs. Pour mettre en place ces fonctionnalités, il manquait également un système de connexion avec base de données sur le site.

### Compatibilité et responsivité

Enfin, l’application n’était pas très responsive sur les ordinateurs de bureau, étant principalement conçue pour les appareils mobiles qui peuvent plus facilement prendre en photo des mains de Mahjong. Cela limitait l’expérience utilisateur pour les utilisateurs accédant au site à partir de leurs PC.

## Améliorations et changement prévues

Au début du semestre 2, nous avons identifié plusieurs domaines d’amélioration pour notre application. Tout d’abord, nous avons prévu d’améliorer la détection des tuiles en réalisant une étude sur les différents types d’images que les utilisateurs peuvent fournir et les résultats attendus. Cette étude nous permettrait de déterminer les paramètres optimaux pour la détection d’image, améliorant ainsi la précision et la fiabilité de cette fonctionnalité.

Ensuite, nous avons planifié d’optimiser le code Java et JavaScript de l’application. Cela inclut un refactor total du code JavaScript pour améliorer sa lisibilité et sa performance. Nous envisagions également de supprimer la fonctionnalité de drag and drop, qui s’est avérée être mal optimiser, au profit d’une approche plus efficace. De plus, nous avions prévu d’optimiser certains algorithmes en Java, tels que l’algorithme de clustering, afin d’améliorer la rapidité de la détection d’image.

Une autre amélioration importante que nous avons prévue est l’ajout des règles de base du Mahjong sur le site. Cela permettrait aux joueurs de calculer leurs scores correctement. De plus, nous envisagions d’intégrer une base de données et de mettre en place une connexion sur le site. Cela permettrait de développer un éditeur de règles et de permettre l’enregistrement des scores et des règles des joueurs.

Enfin, nous avions l’intention de refaire le design du site pour le rendre plus beau sur les ordinateurs de bureau. Cela implique une refonte de l’interface utilisateur pour améliorer la navigation et l’ergonomie sur les grands écrans.

# Amélioration

## Améliorations terminées

### Optimisation du code JavaScript

Pour améliorer la structure et les performances de notre code JavaScript, nous avons entrepris un processus complet de refactoring et d’optimisation. Avant cette refonte, le code JavaScript était mal organisé, avec toutes les fonctionnalités mélangées dans un seul fichier, et il n’y avait pas assez de classes ou de structure clair.

Nous avons donc réorganisé le code en le séparent en modules distincts, avec chaque fonctionnalité regroupée dans des fichier dédiés. Cela nous a permis de mieux organiser notre code et de le rendre plus maintenable à long terme. De plus, nous avons introduit l’utilisation de classes JavaScript pour encapsuler le comportement et les données liés à chaque composant de l’application, ce qui a rendu le code plus lisible et plus facile à comprendre.

Une des améliorations significatives que nous apportés concerne la manipulation dynamique du contenu sur certaine pages, comme celle de la sélection des tuiles ou des paramètres. Avant, chaque interaction/ajout de tuiles avec l’utilisateur entraînait un rechargement complet de la page, ce qui pouvait entraîner des problèmes perceptibles dans l’expérience utilisateur. Pour remédier à cela, nous avons mis en place une approche basée sur la manipulation du DOM (Document Object Model), où nous utilisons des références directes aux éléments HTML et déplaçons dynamiquement les parties sélectionnées au lieu de recharger toute la page.

De plus, nous avons simplifié la gestion des évènements et des interactions utilisateur en introduisant notre propre bibliothèque de signaux, que nous avons nommée « Signal ». Cette bibliothèque nous permet de connecter facilement des fonctions à des éléments HTML spécifiques et d’exécuter ces fonctions lorsque l’utilisateur interagit avec ces éléments. Cela rend le code plus modulaire et évite la nécessité d’utiliser des identifiant uniques pour chaque élément, ce qui simplifie la maintenance et la gestion du code.

Nous avons aussi pris la décision de supprimer la fonctionnalité de drag and drop sur la page de sélection des tuiles. Cette fonctionnalité, implémentée en JavaScript pur, présentait des performances insatisfaisantes et aurait nécessité une refonte complète avec un Framework JavaScript dédié pour obtenir des résultats satisfaisants. Par soucis de propreté du code et de simplicité, nous avons temporairement retiré cette fonctionnalité en attendant une réintégration future avec une approche optimisé.

Schéma + explication du nouveau design pattern JS, expliquer qu’avant tout même fichier etc…

### Optimisation du code Java

Pour améliorer les performances et l’efficacité de notre code Java, nous avons aussi entrepris un processus de refactoring visant à réduire les temps d’attentes des calculs et optimiser l’algorithme de clustering pour une meilleure détection des tuiles.

Tout d’abord, nous avons identifié les parties du code Java qui étaient responsables des longs temps de calcul et avons effectué des ajustements pour les optimiser. Avant ces modifications, l’utilisation de l’algorithme SIFT (Scale-Invariant Feature Transform) de la librairie OpenCV entraînait des temps de calcul assez longs, allant jusqu’à environ 30 secondes pour compléter la détection des tuiles dans une image. Nous avons optimisé cet aspect en révisant et en améliorant les paramètres et les méthodes utilisées par l’algorithme, ce qui a permit de réduire considérablement les temps d’attente à moins d’une seconde.

Ensuite, nous avons aussi fait des améliorations sur l’algorithme de clustering utilisé pour détectés les groupes de tuiles dans les images fournit. Auparavant, l’algorithme détectait systématiquement cinq clusters, ce qui pouvait fausser les résultats car il n’y avait pas forcement tout le temps 5 cluster, parfois plus ou moins. Pour remédier à cela, nous avons améliorer l’algorithme pour qu’il trouve plutôt les x meilleurs clusters dans l’image, ou x est le nombre optimal de cluster détectés pour une image donné.

Expliquer si possible les différences avant après dans le code + mettre images avec temps de calcul avant après

### Amélioration de la détection des tuiles

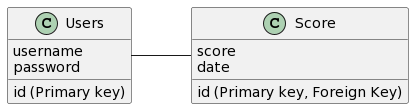
Montrer la détection selon les différents paramètres, montrer qu’on a pu choisir les params optimaux

### Ajout des règles

Expliquer comment marche les règles (dans le code) et dire quelles règles ont été ajouter

### Gestion d’une base de données

Pour améliorer la gestion des données de notre application, nous avons utilisé la librairie JDBC (Java database Connectivity) pour établir une connexion à une base de données SQLite présente dans le projet. Cette base de données a été conçue pour stocker les utilisateur inscrit sur le site ainsi que leurs scores.  
L’architecture de notre base de données comprend deux tables : « Users » et « Scores.



Nous avons veillé à mettre en place des contraintes appropriées dans notre base de données, notamment des contraintes de clés étrangères avec des actions ON DELETE CASCADE pour assurer l’intégrité des données. Cela garanti que les données sont correctement gérées lorsqu’un utilisateur est supprimé.

Bien que les tables pour les règles n’aient pas encore été implémentées, notre architecture Java et la façon dont la base de données peut être gérée permettra l’ajout de nouvelle fonctionnalité. Nous prévoyons d’ajouter des tables supplémentaires pour stocker les règles personnalisées des joueurs.

### Refonte de l’interface utilisateur

Pour améliorer l’interface utilisateur sur notre application, nous avons effectué une refonte de l’interface utilisateur, en mettant particulièrement l’accent sur la page de sélection des tuiles.   
Auparavant, cette page présentait des limitations qui rendaient certaines règles de jeu impossible à appliquer. En effet, les utilisateurs étaient restreints à remplir uniquement cinq slots pour une main, chaque slots pouvant contenir au maximum quatre tuiles. Cela posait des problèmes pour des règles telles que les « Grand jeu » et d’autres.

Pour remédier à cette limitation, nous avons repensé le fonctionnement de la page de sélection. Désormais, un seul slot est visible initialement, et lorsqu’il est rempli, un nouveau slot apparait automatiquement en dessous. Il n’y a plus de limite de slots et de limite de tuiles par slot, ce qui permet de résoudre tous les problèmes liés aux règles impossibles à appliquer.

------ Image avant après -------------

De plus, nous avons améliorer l’affichage pour la détection des tuiles. Avant, une fois la détection terminée, les tuiles étaient remplies instantanément, ce qui pouvait parfois être déroutant pour l’utilisateur. Désormais, nous avons mis en place un affichage plus clair, ou chaque tuile détecté dans l’image est affichée à gauche, avec sa correspondance à droite. L’utilisateur peut ensuite sélectionner les tuiles détectées qu’il souhaite conserver.

------ Image présentatrice -------------

Auparavant, lors de l’envoie d’une image pour la détection des tuiles, en cas de problème coté serveur, tel qu’un serveur indisponible ou non fonctionnel, l’utilisateur était confronté à une pop-up affichant un cercle de chargement infini, ce qui pouvait être frustrant. Désormais, en cas de problème coté serveur, nous avons mis en place un pop-up explicative, informant l’utilisateur du problème rencontré. Cette approche permet à l’utilisateur de comprendre la situation et de ne pas rester bloqué, améliorant ainsi son expérience globale sur notre application.

--- setting custom/nouveau parametres------------

## Améliorations en cours

### Editeur de règles

Malheureusement, nous n’avons pas pu intégrer un editeur de règles dans notre application comme nous l’avions initialement prévu. Avec les nombreuses autres améliorations et optimisation importantes que nous avions planifiées, l’ajout d’un éditeur de regles s’est avéré être un défi de taille et nous avons rapidement réalisé que nous manquions de temps pour le mettre en œuvre de manière satisfaisantes et correctement.

Cependant, il est important de noter que l’architecture que nous avons mise en place tant du côté JavaScript que du côté Java permettra l’ajout ultérieur d’un éditeur de règles. En utilisant le design pattern actuel des règles qui est adapté et en utilisant la flexibilité de notre base de données, nous avons créé une infrastructure solide sur laquelle nous pourrons construire à l’avenir. Ainsi, bien que l’éditeur de règles n’ai pas été implémenté dans cette version de l’application, il reste une possibilité pour des développements futurs.

### Algorithme de cluster encore possible à améliorer

Expliquer le problèmes avec le calcul avec le centre de la tuiles, montrer les shéma et dire qu’il faut calculer avec les bords etc