Tak&Bin

Cahier de conception

Auteur : SPI3 Projet Groupe 2	**À destination de :** **M. Jacoboni et M. Després**
Crée-le : 13 mars 2015	**Dernière modification : 03/04/2015**
Objectifs : Création d'un Takuzu exécutable sur un support informatique répondant à une liste de fonctionnalités préétablies.	

Introduction

Le but de ce document est de présenter l'architecture du logiciel Tak&Bin développé dans le cadre du projet de L3 ainsi que la démarche de conception qui fut adoptée afin de le réaliser.

Ce projet consiste en la réalisation d'un logiciel permettant de jouer des parties de Takuzu tout en proposant un système de persistance des données utilisateurs et un suivi des scores de l'ensemble des joueurs.

Le projet fut réalisé en Ruby en suivant une méthode Modèle Vue Contrôleur.
Ce document de conception décrit la structure générale de l'application dans un premier temps puis détaille le fonctionnement des différents modules de l'application.

Interface

A.Interface utilisateur

Le système de jeu se compose d'une application exécutable sur les principaux OS. L'interface se présente sous la forme de page navigable.
Cette application comporte à son point d'accès un système de connexion par mot de passe ou de création d'un nouveau compte pour l'utilisateur qui souhaite accéder au service proposé (cf fig 1.).

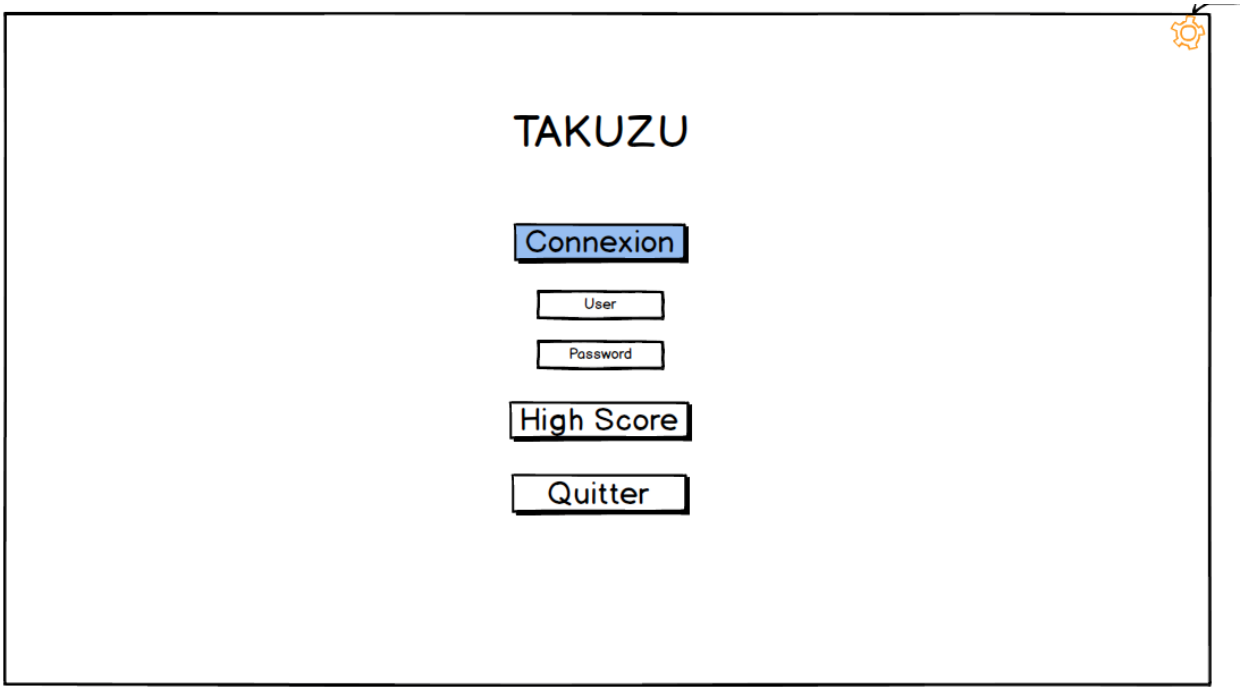

Fig 1. Mock up préliminaire de l'interface d'entrée du système Tak&Bin

L'interface utilisateur se compose également des pages suivantes :

- page principale de l'application (sélection de parties, defi day, résumé profil utilisateur, prévisualisation grille, score de la grille sélectionné). (Fig. 2)
- page de jeu (grille de jeu, option de jeu). (Fig. 3)
- page de classement des scores. (Fig. 4)
- page de gestion de compte utilisateur et option générale de l'application.

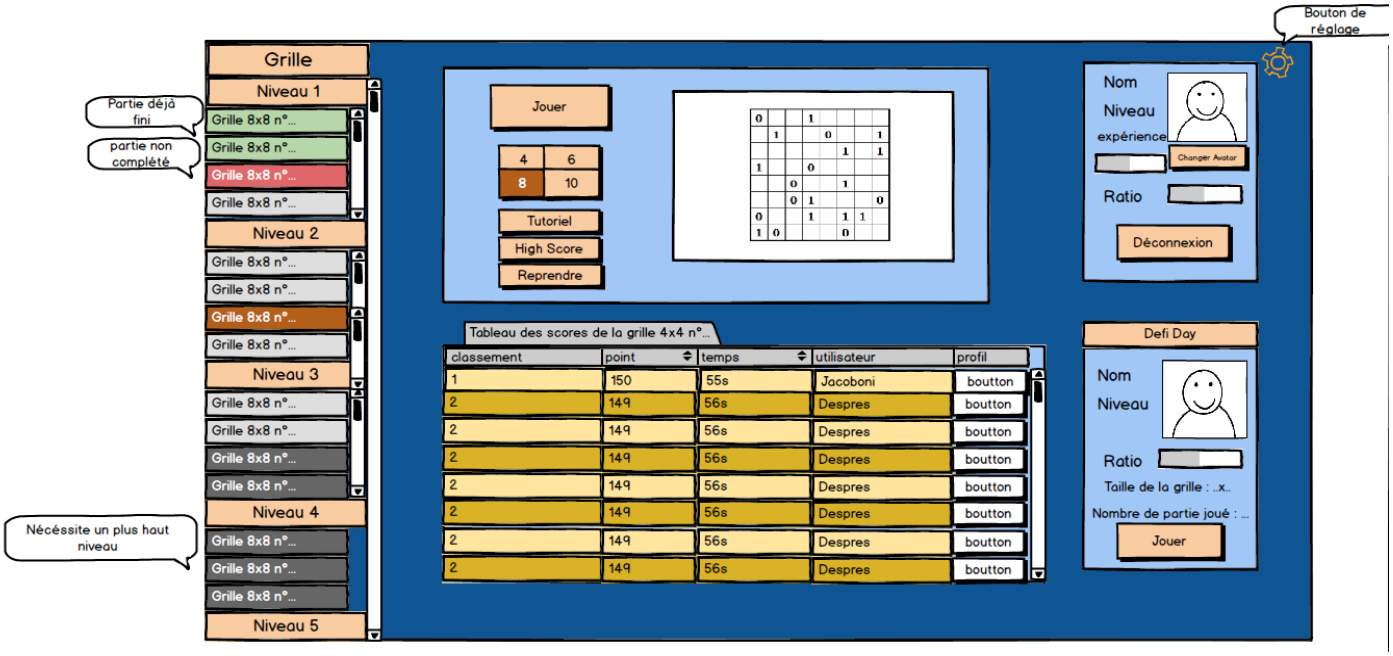

Fig 2. Mock up préliminaire de la page principale de l'application

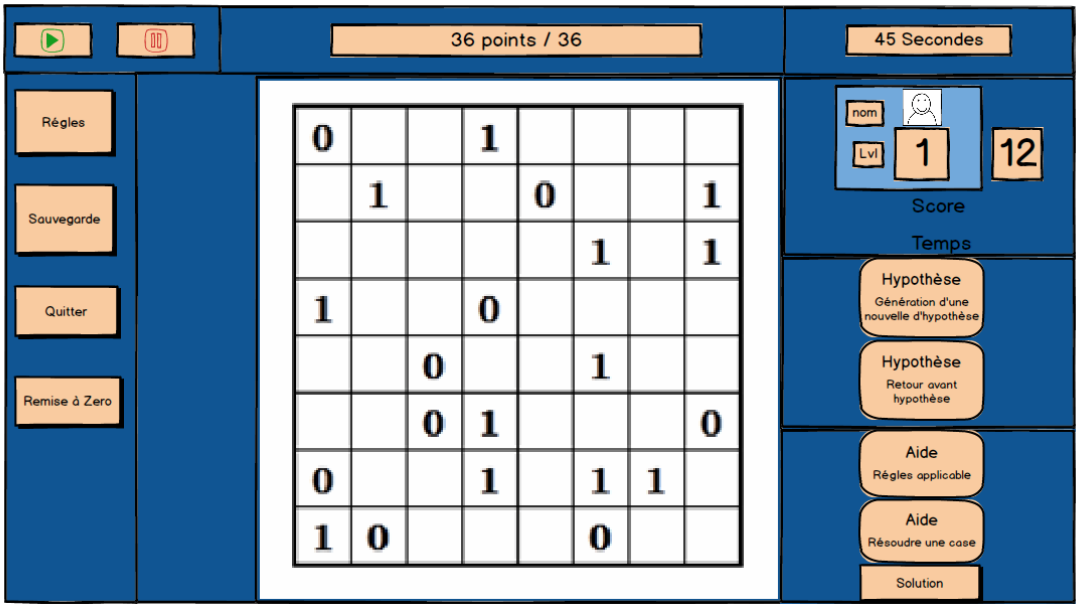

Fig 3. Mock up préliminaire de la page de jeu

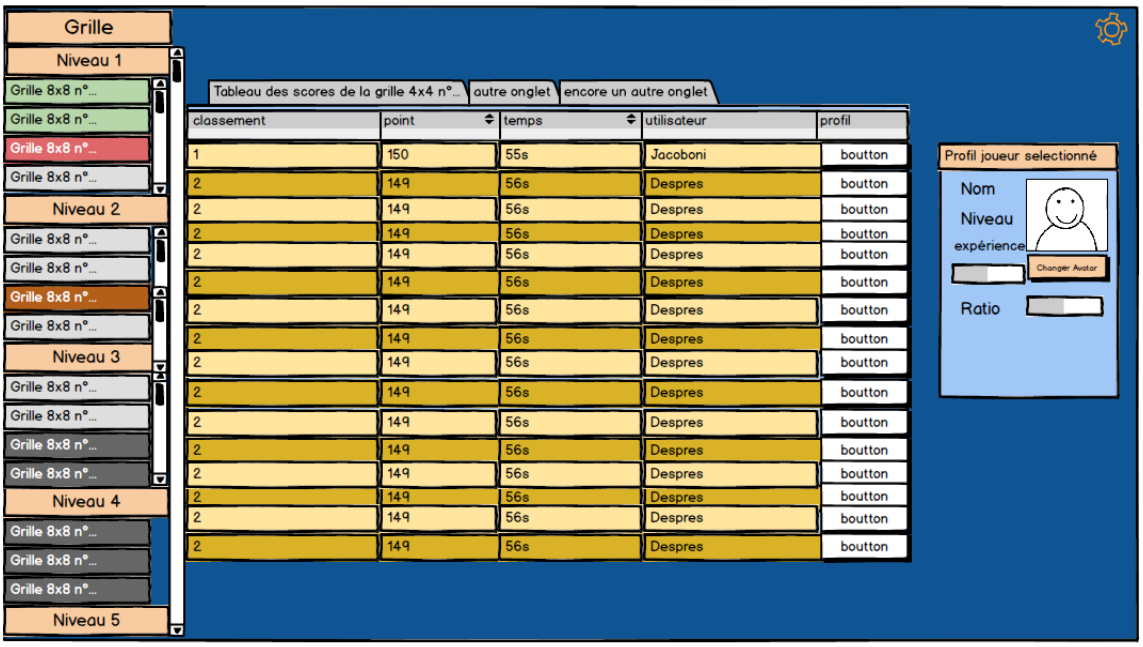

Fig 4. Mock up préliminaire de la page de classement des scores

Après une première itération de code, l'interface graphique se découpe de la façon suivante.

- interface d'entrée du système Tak&Bin (Fig 5.)
- formulaire d'inscription (Fig 6.)
- une page de sélection des parties (Fig 7.)
- une page de jeu (Fig 8.)

Fig 5. (Itération 1) interface d'entrée du système Tak&Bin

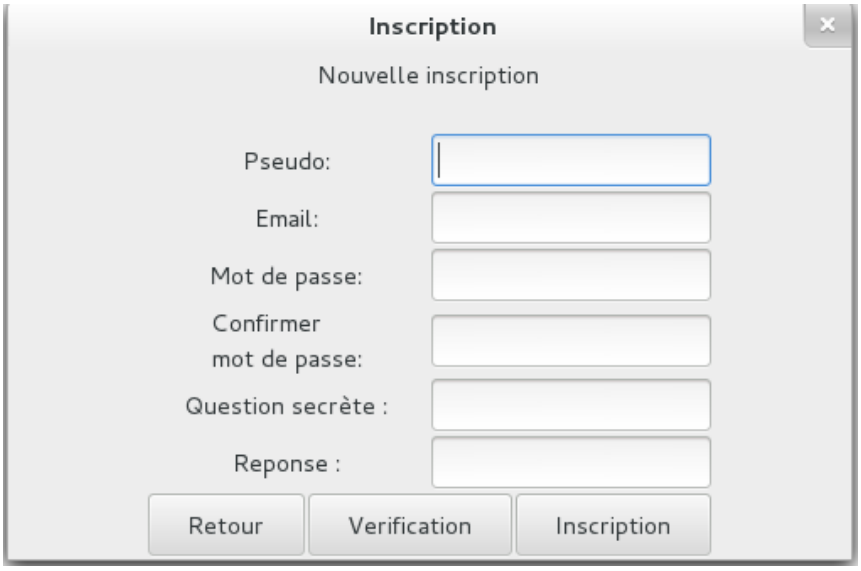

Fig 6. (Itération 1) formulaire d'inscription

Fig 7. (Itération 1) page de sélection des parties

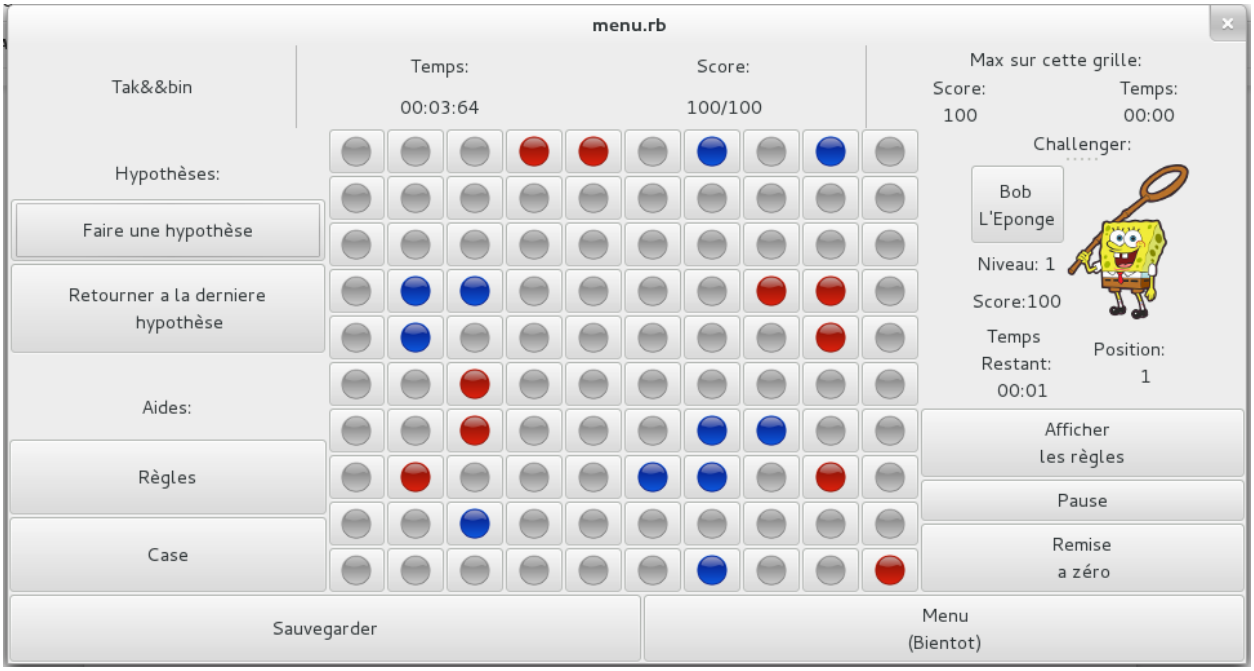

Fig 8. (Itération 1) Page de jeu

B.Interface logicielle

Les interfaces logicielles mentionnées ici sont des interfaces internes au système.

- Interface avec la base de données : l'application Tak&Bin doit communiquer avec sa base de données afin de permettre une persistance des informations des utilisateurs. Une base de données SQLite sera implémentée à cette fin.

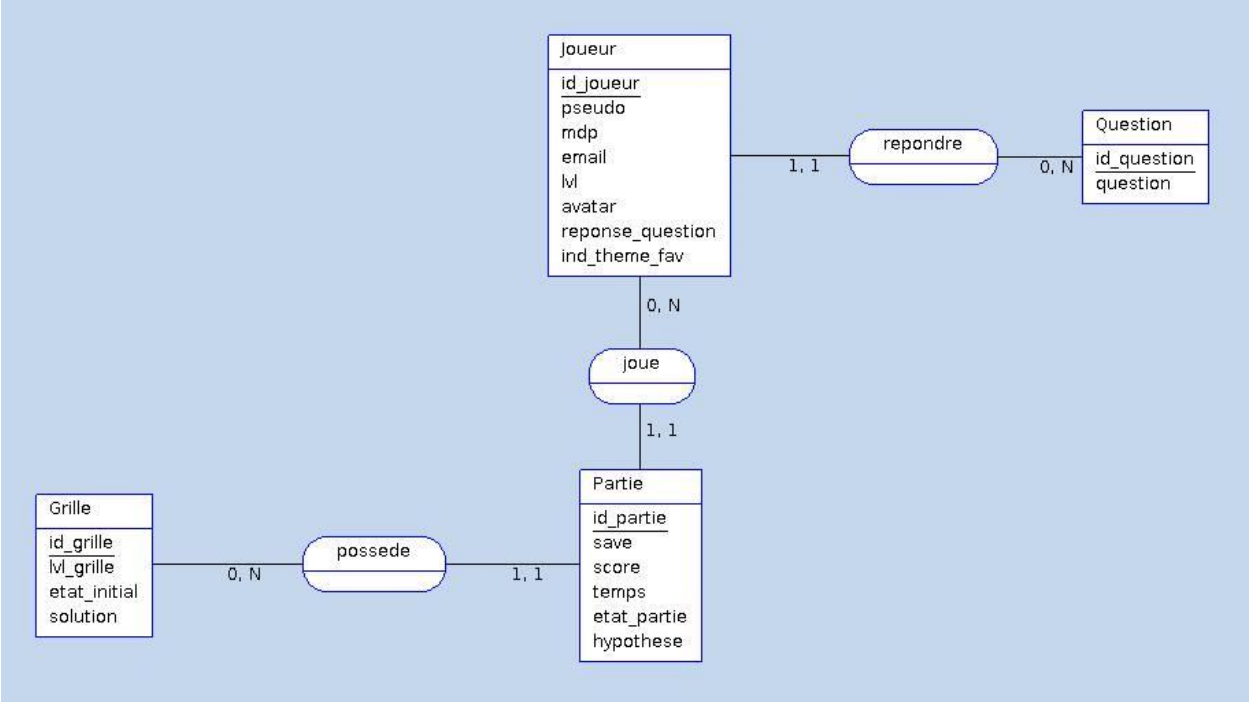

Fig 9. Modèle entité association de la base de données du Tak&Bin

Joueur (id_joueur, pseudo, mdp, email, lvl, avatar, reponse_question, ind_theme_fav, #id_question)

Question (id_question, question)

Grille (id_grille, lvl_grille, etat_initial, solution)

Partie (id_partie, save, score, temps, etat_partie, hypothèse, #id_grille, #id_joueur)

Contraintes générales de conception

Les directives qui nous ont été données lors de l'initialisation du projet :

- Travail par groupe de 7
- livrable le 27 avril 2015
- Il doit s'agir d'un jeu de Takuzu
- programmation en Ruby et interface en GTK
- Utilisation de UML et des principes de génie logiciel
- protection de la base de données
- Éviter les possibilités de triche
- Fournir les artéfacts suivants :
    - Cahier des charges
    - Cahier de conception
    - Cahier de documentation
    - Manuel utilisateur
    - Code source

Architecture générale de l'application

L'architecture MVC

Pour le développement de l'application, nous avons essayé d'appliquer une méthode Modèle Vue Contrôleur. Cette architecture permet de séparer le fonctionnel de l'interface.
Le modèle correspond à la base de données qui n'est accessible que par le serveur.
La (les) vue a été créée à l'aide du générateur Glade, qui nous permet une meilleure maintenabilité pour les éléments statiques, qui surchargeraient inutilement le code utile.
La partie contrôleur comprend tous les codes écrits en Ruby, que ce soit la gestion du serveur et des accès à la base de données, ou bien de la partie en elle même.

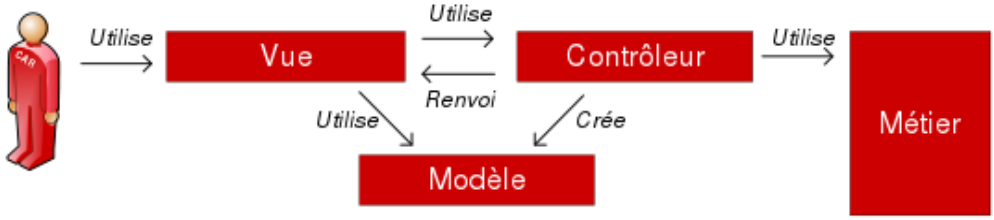

Fig 2. Schématisation d'une architecture Modèle-Vue-Contrôleur

Fonctionnement interne de l'application

Structure logicielle

Convention
Par convention on indiquera entre parenthèses après la description d'une procédure le nom de la fonction associé dans le code.
Chaque dossier décrit ci-dessous comporte également un ou plusieurs fichiers .glade comportant l'architecture graphique des différentes pages de l'application.

Version 1

Le fichier de lancement principal se situe dans le répertoire ~/menu du projet il s'agit du fichier menu.rb

Le fichier menu.rb contient les procédures suivantes :

- ouverture de la page de nouvelle partie (nouvPartie).
- ouverture du formulaire d'inscription(inscrire).
- ouverture de la page de HighScore (showHigh).
- fermeture de la page actuelle et de toutes les fenêtres filles lancées (onDestroy).

Le fichier FormInscription.rb contient les procédures suivantes :

- vérification du contenu du formulaire et demande de rectification en cas de champs non conforme (vérif).
- envoie à la base de données du contenu du formulaire une fois celui-ci correctement rempli (envoieForm).

Dans le répertoire ~/Partie

Le fichier partie.rb gère l'affichage et l'initialisation des parties de jeu et contient les procédures suivantes :

- une fonction d'initialisation de la grille qui génère la partie demandée
- une fonction de sauvegarde d'une hypothèse (pushHypo)
- une fonction de retour à l'état avant dernière hypothèse(popHypo)
- une fonction de remise à zéro de la grille (razGrille)
- une fonction de mise en pause de la partie et de masquage de la grille (pause)
- une fonction d'affichage des règles mettant en évidence les cases ne respectant des règles pouvant s'appliquer à ce moment-là.
- une fonction de sauvegarde de la partie actuelle (sauvegarde)
- une routine de vérification des règles du takuzu (vérif)

Le fichier cellule.rb gère le comportement des cellules composant la grille de partie et contient les procédures suivantes :

- une fonction d'initialisation des cellules selon leur état (bloqué ou non et couleur) (initialize(bloque,depart,soluce))
- une fonction pour modifier l'état d'une cellule(Cellule.grilleAjour(etat))
- une fonction pour vérifier si l'état d'une cellule correspond à l'état souhaité(Cellule.grilleAjour?)
- une fonction qui retourne si une cellule est vide(vide?)
- une fonction permettant de mettre la cellule à son état solutionné (resoudre)
- une fonction pour vérifier si le contenu d'une cellule correspond à celui attendu dans la correction(estCorrect)
- une fonction pour remettre l'état d'une cellule dans son état d'initialisation de départ(raz)
- une fonction pour mettre à jour l'image de la cellule après un changement d'état(maj)
- une fonction permettant de changer l'état d'une cellule par un clic (onClic(button))

Dans le répertoire ~/Regles

Le fichier affichRegles.rb permet l'affichage des règles et contient les procédures suivantes

- une fonction d'initialisation de la page de règle (initialize)
- une fonction de fermeture de la page de règle (onDestroy)

Dans le répertoire ~/timer

Le fichier Timer.rb permet l'implémentation d'un timer pour chronométrer les résultats des joueurs, il contient les procédures suivantes :

- une fonction d'initialisation du timer(initialize)
- une fonction pour mettre le timer en pause(pause)
- une fonction pour remettre le timer en route(reprendre)
- une fonction d'affichage du statut du timer en heure minute seconde (to_s)
- une fonction de remise à zéro du timer (raz)
- une fonction de test du timer (Timer.test)

Dans le répertoire ~/User

-L'ensemble de fichiers user (user.rb, et son homonyme glade) sert à définir les attributs nécessaires à un joueur pour jouer une partie.

Il est « lié » a la base de données dans le sens ou certains de ces attributs seront stockés à long terme, alors que d'autres non.
Il permet aussi grâce a la méthode display d'afficher une petite fenêtre de présentation rapide du joueur avec quelques informations, telle que son nom, son niveau, son avatar. Ceci dans le but d'appliquer au mieux le principe de « Don't repeat yourself », si cher au langage Ruby.

-L'ensemble de fichiers userInterface est l'interface dont dispose l'utilisateur après s'être identifié. Elle lui permet de:

-commencer ou reprendre une partie, d'un niveau au choix, à partir du volet de gauche.

-visualiser le plus simplement les informations concernant son compte.

-modifier l'apparence du jeu, en choisissant parmi 16 777 216 couleurs, et ceci pour différents éléments du jeu, comme le fond, et les boutons.

-d'afficher l'actuel meilleur score au défiday, et proposer de jouer une partie pour le détrôner.

-d'afficher les meilleurs scores pour une grille sélectionnée dans le volet de gauche.

Gestion de la base de données

Joueur :

id_joueur (Integer, Clé primaire):

- Identifiant unique d'un joueur. Il s'agit d'un entier incrémenté automatiquement pour chaque nouveau joueur.

pseudo (Text) :

- Correspond au pseudo du joueur sous forme de chaîne de caractère.
- Correspond à l'identifiant de l'utilisateur.
- Correspond au nom visible par tous les utilisateurs.

mdp (Text) :

- Correspond au mot de passe du joueur voulant se connecter.

email (Text) :

- Correspond à l'adresse email du joueur.
- Permet à un joueur de récupérer son mot de passe perdu

lvl (Integer) :

- Correspond à l'expérience du joueur.
- Permet de calculer le niveau du joueur.

avatar (Text) :

- Correspond à l'avatar du joueur sérialisé sous forme binaire (en Marshall).

reponse_question (Text):

- Correspond à la réponse de la question de l'utilisateur
- Permet la vérification de l'authenticité du joueur en cas de mots de passe perdus.

ind_theme_fav (Integer) :

- Correspond à l'indice du thème dans le tableau des thèmes.

#id_question (Integer):

- Correspond à un identifiant de la question auquel a répondu le joueur

Question :

id_question (integer, primary key) :

- Correspond à l'identifiant d'une question

question (text) :

- Correspond à l'intitulé de la question

Grille :

id_grille (integer, primary key) :

- Correspond à l'identifiant d'une grille

lvl_grille (integer):

- Correspond au niveau de la grille

etat_initial (text):

- Correspond à l'état d'initialisation de la grille

solution(text) :

- *Correspond à la solution de la grille*

Partie :

id_partie (integer, primary key) :

- Corresponds à un ID unique lié à une partie et incrémenté automatiquement pour chaque nouvelle partie créée.

save (text) :

- Corresponds à l'état de la grille lors de la sauvegarde.

score (integer) :

- Corresponds au score obtenu par le joueur sur cette partie.

temps (integer) :

- Corresponds au temps écoulé durant lequel la partie était active.

etat_partie (boolean) :

- Corresponds au statut de la partie (fini ou en cours).

hypothèse (text) :

- Corresponds à la pile contenant les hypothèses effectuées durant la partie.

#id_grille (integer) :

- Corresponds à un identifiant d'une grille associé à la partie.

#id_joueur (integer):

- Corresponds à un identifiant d'un joueur associé à la partie.

Cas d'utilisation

Connexion au Takuzu

Titre : Connexion au Takuzu
Acteur principal : Utilisateur/Joueur
Préconditions : Ruby et gtk sont installés sur la machine de l'utilisateur.
Postconditions : L'utilisateur a accès au contenu de jeu correspondant à son profil.

Action des acteurs	Réponse du système
1. L'utilisateur lance l'exécutable du jeu.	2. Le système ouvre l'interface de connexion du jeu.
3. L'utilisateur clique sur le bouton Connexion.	4. Le système affiche les zones de saisie User/Password afin d'effectuer la connexion.
5. L'utilisateur rentre un nom d'utilisateur et un mot de passe.	6. Le système vérifie l'existence d'un compte existant avec ce nom et la correspondance avec le mot de passe rentrée. 7. Le système charge les données utilisateur et ouvre la page de lancement de partie.

Scénario alternatif

*a. À tout moment l'application peut rencontrer un problème et devoir fermer
1. Le système informe l'utilisateur qu'un problème a eu lieu et que l'application va fermer.

5. L'utilisateur rentre un nom comportant des caractères non valides
5a. Le système indique que des caractères inappropriés sont présents et rappelle les caractères autorisés.
5b. Le système remet à vide les cases User et Password.

6a. Aucun compte n'existe dans la base de données.

6a-1. Le système crée un nouveau compte utilisateur avec le nom d'utilisateur et le mot de passe rentré.

6b. Un compte possédant ce nom d'utilisateur existe déjà et le mot de passe ne correspond pas.
6b-1. Le système informe l'utilisateur que la combinaison mot de passe/nom d'utilisateur ne correspond pas.
6b-2. Le système informe l'utilisateur que le nom d'utilisateur est déjà pris s'il souhaitait créer un compte.

7. La communication avec le serveur ne peut pas être établie.
7a. Le système informe l'utilisateur que la connexion n'a pas pu s'établir et propose de réessayer.

<u>Lancer une partie</u>
Titre : Lancer une partie de takuzu
Acteur principal : Utilisateur/Joueur
Préconditions: L'utilisateur a lancé le jeu et effectué une connexion avec succès.
Postconditions : L'utilisateur a accès à l'interface de jeu et peut débuter sa partie.

Actions des acteurs	Réponse du système
1. L'utilisateur sélectionne la taille de la grille auquel il souhaite jouer.	
	2. Le système charge et affiche les différentes parties disponibles de cette taille triée par niveau de difficulté.
3. L'utilisateur sélectionne la grille auquel il souhaite jouer.	
	4. Le système affiche une vue réduite de la grille sélectionnée.
	5. Le système charge les scores liés à la grille et les affiches dans le tableau des scores.
6. L'utilisateur clique sur le bouton Jouer pour lancer la partie correspondant à la grille sélectionnée.	
	6. Le système ouvre la page de jeu et charge la partie sélectionnée.

Scénario alternatif

*a. À tout moment l'application peut rencontrer un problème et devoir fermer

1.Le système informe l'utilisateur qu'un problème a eu lieu et que l'application va fermer.

*b. À tout moment l'application peut rencontrer un problème dans la communication avec le serveur.

1. Le système informe l'utilisateur que la connexion n'a pas pu s'établir et propose de réessayer.

3. L'utilisateur sélectionne une grille sur laquelle il possède une partie sauvegardée.

3a. Le système dégrise le bouton « Reprendre » et permet de recharger la grille en question dans l'état où elle était lors de sa sauvegarde.

Générer une hypothèse et revenir à l'état précédent.

Titre : Hypothèse et retour d'hypothèse

Acteur principal : Utilisateur/Joueur

Préconditions: L'utilisateur a lancé une partie.

Postconditions : L'état de la partie a été sauvegardé et l'utilisateur dispose d'un moyen pour retourner à l'état précédent la sauvegarder.

Actions des acteurs	Réponse du système
1. L'utilisateur clique sur le bouton « génération d'une nouvelle hypothèse »	
	2. Le système sauvegarde l'état de la grille, le timer, le score, le nombre d'aides utilisé et les places dans une pile.
Les étapes 1-2 peuvent être répétées.	
3. L'utilisateur clique sur le bouton « Retour avant hypothèse ».	
	4. Le système charge la dernière sauvegarde d'hypothèse effectuée et la retire de la pile.
Les étapes 3-4 peuvent être répétées.	

Scénario alternatif

*a. À tout moment l'application peut rencontrer un problème et devoir fermer

1.Le système informe l'utilisateur qu'un problème a eu lieu et que l'application va fermer.

*b. À tout moment l'application peut rencontrer un problème dans la communication avec le serveur.

1. Le système informe l'utilisateur que la connexion n'a pas pu s'établir et propose de réessayer.

3. L'utilisateur clique sur le bouton « Retour avant hypothèse » alors qu'aucune sauvegarde d'hypothèse n'est présente dans la pile.

3a. Le système bloque le signal et ne change pas l'état du jeu.