Application de création et d'aide à la résolution de puzzle *picross*

Dossier d'analyse et de conception

Groupe n°2 :
Brinon Baptiste
Brocherieux Thibault
Cohen Mehdi
Debonne Valentin
Lardy Anthony
Mottier Emeric
Pastouret Gilles
Pelloin Valentin

Licence Informatique
Le Mans Université
27 février 2018

Sommaire

1 | Présentation

1.1 Introduction

Dans le cadre de l'unité d'enseignement « Génie logiciel 2 » de la Licence d'informatique de Le Mans Université, les étudiants de troisième année sont amenés à travailler sur un projet de développement d'une application.

Ce document permet de présenter les solutions au cahier des charges, ainsi que l'architecture du programme demandé. Celui-ci est divisé en plusieurs parties : l'architecture du jeu, la gestion d'une partie, l'aide à la résolution du picross,

1.2 Objectif de l'application

Nous devons réaliser un jeu de type picross (aussi appelé *nonogramme*, *logigramme* ou *hanjie*) permettant à un utilisateur de résoudre des grilles et de l'aider dans sa réalisation. Une proposition de nom d'application et de logo se trouve en annexe (4.3).

1.3 Outils

Afin de réaliser notre application de *picross*, nous utilisons plusieurs outils.

LaTeX est utilisé pour rédiger et mettre en forme les différents livrables avant de les convertir en format PDF.

Git et *GitHub* permettent le partage d'informations et la mise en commun du code ainsi que les livrables associés. Nous nous servons de *Discord* afin de communiquer au sein du groupe, partager nos idées, nos informations et notre travail.

Pour concevoir les différents diagrammes UML, nous avons recours au logiciel *Astah* et au site internet *Tom's Planner*[1].

Afin de créer les maquettes du logiciel (disponibles en Annexe) permettant d'avoir un rendu de la future interface graphique, nous avons utilisé le logiciel *Balsamiq*[2].

Pour simplifier la vérification de la qualité du code produit, nous utilisons Code Climate (maintenabilité) ainsi que Travis CI (compilation et tests).

Le langage de programmation Ruby est utilisé pour programmer le logiciel, incluant une documentation générée par *RDoc*[3]. Les bibliothèques de *GTK* seront utilisées pour la réalisation de l'interface graphique.

1.3.1 Gems utilisés

Nous utilisons les *gems* Ruby suivants :

RSpec : écriture et compilation des tests unitaires

simplecov : calcul et affichage du taux de couverture des tests unitaires

RDoc : génération de la documentation en ligne[3]

hanna-bootstrap : thème responsive pour *RDoc*

rake : gestions des taches automatiques de notre projet (compilation de la documentation, tests unitaires, création de l'exécutable, ...)

rmagick : conversion automatique d'images de picross trouvés sur internet

nokogiri : *parser* de pages HTML pour convertir les picross trouvés sur internet

gtk3 : liens Ruby vers les bibliothèques d'interface graphique GTK3

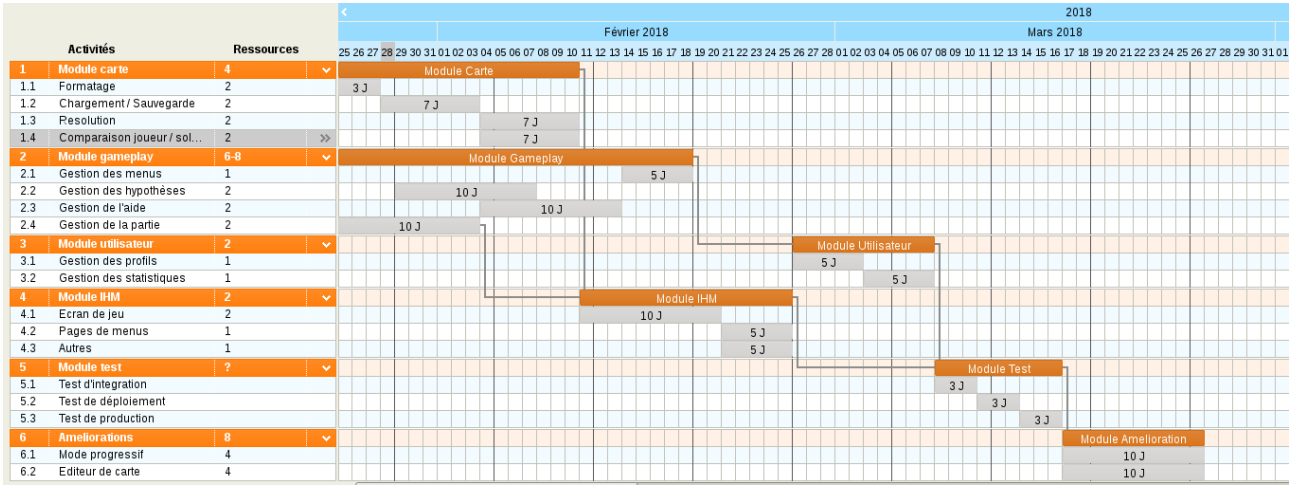
Bundler : installation automatique de tous les *gems*

1. Site web de *Tom's Planner* : `http://tomsplanner.fr/`
2. Nos maquettes sont disponibles ici : `https://balsamiq.cloud/sxl7v/pw3tb/r599E`
3. Notre documentation en temps réel est disponible à l'adresse `https://picross.vlntn.pw/doc/`

2 | Conception générale

2.1 Diagramme de Gantt détaillé

FIGURE 2.1 – Diagramme de *Gant détaillé*



À travers ce diagramme de Gantt, nous observons l'organisation du projet sur les trois mois à venir. Le projet se découpe en plusieurs modules : carte, *gameplay*, utilisateur, IHM, test et améliorations.

Chaque module a une durée de vie, nous lui accordons un temps de travail et un certain nombre de personnes, les modules sont également divisés en différentes étapes ce qui correspond aux fonctionnalités de l'application.

Les modules sont imbriqués les uns dans les autres, c'est-à-dire que certains dépendent d'autres modules. Sans les modules gameplay et carte réalisés, le module IHM ne peut être réalisé.

2.2 Besoins fonctionnels

En plus de la génération et de la gestion d'une partie, le programme doit contenir les fonctionnalités suivantes :

Table 2.1 – Besoins fonctionnels

Catégorie	Besoins Fonctionnels
Aide	BF1.1 : Gagner une utilisation d'aide
	BF1.2 : Acheter une aide
	BF1.3 : Utiliser une aide gratuite
Chronomètre	BF2.1 : Accéder au score à la fin d'une partie
	BF2.2 : Mettre la partie en pause
	BF2.3 : Reprendre une partie mise en pause
Hypothèses	BF3.1 : Créer une hypothèse
	BF3.2 : Valider une hypothèse
	BF3.3 : Annuler une hypothèse
Sauvegarde	BF4.1 : Effectuer les sauvegardes automatiquement
	BF4.2 : Charger automatiquement une partie à son lancement
	BF4.3 : Stocker localement les sauvegardes
IHM	BF5.1 : Afficher la grille en noir et blanc
	BF5.2 : Être compatible avec plusieurs périphériques (clavier, souris)
	BF5.3 : Mettre en valeur des indices associés aux cases remplies
	BF5.4 : Afficher la quantité de cases sélectionnées (sélection multiple)

1 Aide

1.1 Gagner une utilisation d'aide

L'utilisateur a la possibilité de gagner des aides gratuites en complétant des grilles et en terminant les niveaux.

1.2 Acheter une aide

Durant une partie, si il en a besoin, le joueur peut utiliser une aide en échange d'une pénalité de temps, sauf s'il en dispose d'une gratuite.

1.3 Utiliser une aide

L'utilisateur peut utiliser l'aide à tout moment en échange d'une perte de temps, sauf si celui-ci dispose d'une aide gratuite, en quel cas, l'aide n'entraînera aucune pénalité de temps.

2 Chronomètre

2.1 Accéder au score à la fin d'une partie

À la fin de la partie, un nombre d'étoiles est attribué au joueur en fonction du temps écoulé.

2.2 Mettre la partie en pause

À n'importe quel moment de la partie, le joueur peut la mettre en pause. Elle est alors masquée.

2.3 Reprendre une partie mise en pause

Suite à une mise en pause, le joueur peut dès qu'il est prêt, relancer la partie.

3 Hypothèses

3.1 Créer une hypothèse

Durant la partie, le joueur peut créer une nouvelle hypothèse en cliquant sur le bouton associé. Les cases colorées seront alors d'une couleur différente.

3.2 Valider une hypothèse

Lorsque l'hypothèse s'avère juste, l'utilisateur peut choisir de la valider. Les cases colorées deviennent alors noires et l'hypothèse disparaît de celles en cours.

3.3 Annuler une hypothèse

Lorsque l'hypothèse est fausse, l'utilisateur peut l'annuler. Les cases colorées suite à la création de l'hypothèse sont alors blanchies.

4 Sauvegarde

4.1 Effectuer les sauvegardes automatiquement

A chaque modification de la grille, une nouvelle sauvegarde est effectuée automatiquement, sauvegardant aussi la valeur du chronomètre.

4.2 Charger automatiquement une partie à son lancement

Lorsque le joueur relance une partie qu'il n'avait pas terminée, le chronomètre reprend là où il était arrêté, et la grille est affichée là où l'avait laissée le joueur.

4.3 Stocker localement les sauvegardes

Les sauvegardes sont conservées sur la machine de l'utilisateur. Elles ne sont donc pas en ligne.

5 Interface Homme Machine

5.1 Afficher la grille en noir et blanc

Les cases sont blanches, et sont colorées en noir lors de leur validation par le joueur. Les cases appartenant à une hypothèse doivent néanmoins être identifiées comme telles.

5.2 Être compatible avec plusieurs périphériques

Le joueur peut jouer avec les périphériques qu'il désire, que ce soit seulement au clavier, seulement à la souris, ou les deux en même temps.

5.3 Mettre en valeur des indices associés aux cases remplies

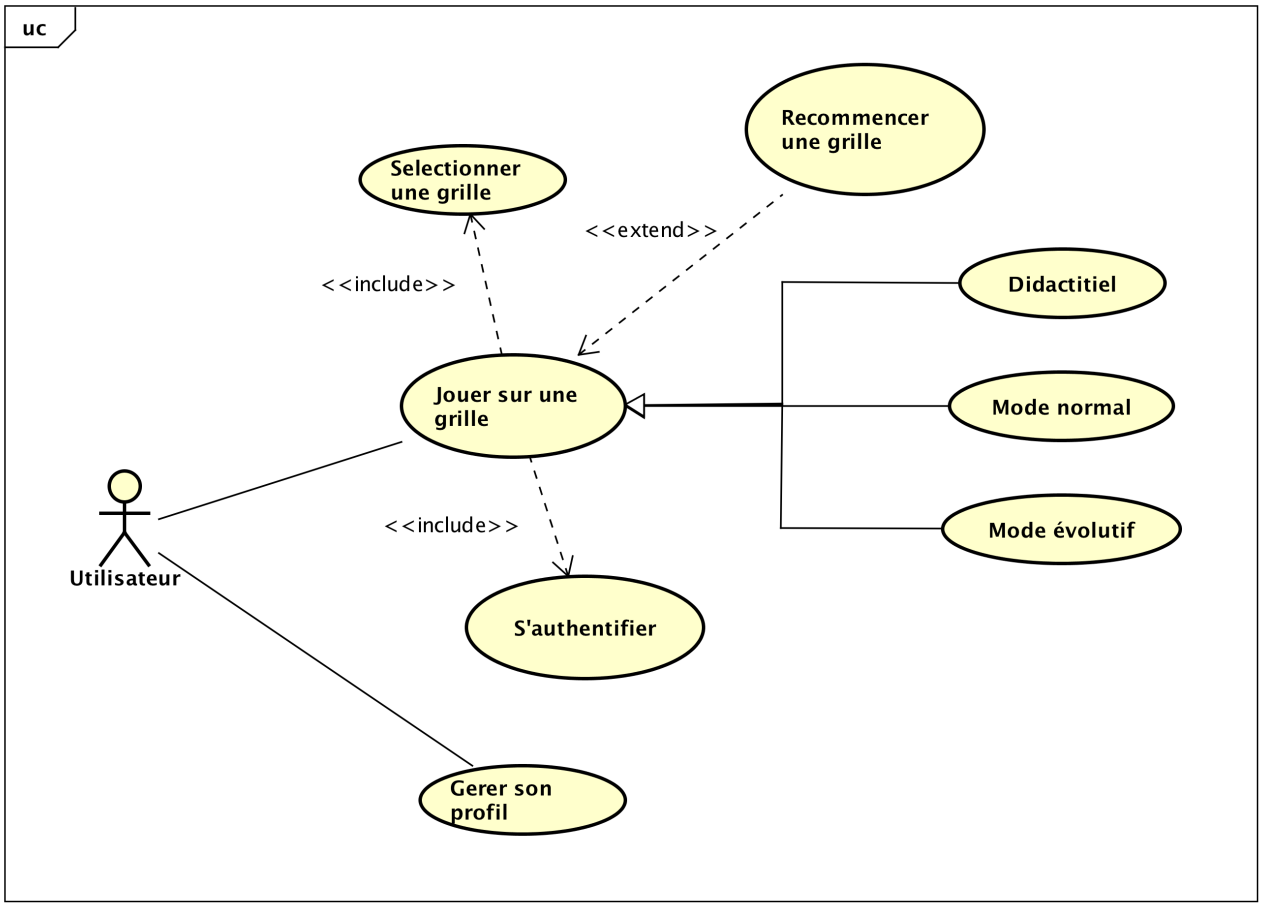
Quand le joueur colorie des cases, le programme identifie de quel bloc de cases il s'agit et met en valeur l'indice associé, de façon à faciliter la lisibilité de la grille.

5.4 Afficher la quantité de cases sélectionnées

Quand l'utilisateur sélectionne plusieurs cases en une fois, un petit chiffre à côté de la souris (ou de la case sur laquelle se trouve le curseur s'il joue au clavier) indique le nombre de cases actuellement sélectionnées.

2.3 Diagramme de cas d'utilisations

FIGURE 2.2 – Diagramme de *Cas d'utilisation*



Le but de l'utilisateur est de jouer sur une grille de *picross*. Pour cela, il doit avoir un profil. Il peut donc en créer un nouveau s'il en a envie, ou s'authentifier à un profil déjà existant. Le joueur doit ensuite sélectionner une grille à l'aide du menu. Il a le choix entre trois types de jeu :

— Le didacticiel, permettant d'apprendre les règles du jeu.
— Le mode classique, permettant de jouer sur plusieurs grilles de tailles et difficultés variables.
— Le mode évolutif, agrandissant la taille de la grille au fur et à mesure que le joueur résoudra celle-ci.

À tout moment, celui-ci peut utiliser l'aide intégrée au jeu pour se débloquer. Il peut aussi choisir de recommencer la grille s'il le souhaite.

2.4 Mode classique

Dans le mode classique, le joueur a accès à plusieurs fonctionnalités. Il doit tout d'abord choisir un chapitre débloqué puis une grille. Ensuite, lorsqu'il se trouve en partie, il peut tout d'abord interagir avec la grille en cochant une case afin de se repérer dans les cases qui ne doivent pas être coloriées ou en coloriant une case. Il peut également enlever une coche ou une couleur d'une case s'il s'aperçoit qu'il s'est trompé. Il peut aussi sélectionner un groupe de cases à colorier, cocher ou vider.

Au niveau de l'interface, l'utilisateur peut, à l'aide de boutons, réinitialiser la grille s'il souhaite repartir de zéro. S'il termine la grille, il sera renvoyé dans le chapitre en cours. Il peut également utiliser différentes aides (en cliquant sur un bouton) qui diffèrent par la clarté de l'indice, leur coût et leur pénalité. Il est aussi possible pour l'utilisateur de créer ou d'enlever des hypothèses afin de partir dans une direction dont il n'est pas sûr. Enfin, l'utilisateur peut quitter la partie quand il le souhaite, la grille est sauvegardée automatiquement.

2.5 Mode évolutif

Lors d'une partie en mode évolutif, le joueur choisit une des grilles du mode. Chaque grille choisie a une taille fixe de 5x5 (cases). Pour résoudre cette grille, il a accès aux mêmes fonctionnalités qu'une partie classique soit interagir avec la grille et avec l'interface.

Une fois la grille réussie, celle-ci évolue en une grille plus grande (10x10) tout en gardant le coin supérieur gauche identique à la grille 5x5 réalisée par l'utilisateur (voir exemple en Annexe). Après cela, le joueur doit remplir la grille 10x10 en ayant les mêmes fonctionnalités que précédemment puis la grille évolue une nouvelle fois en 15x15 et ainsi de suite jusqu'à atteindre la taille maximale de 25x25. Une fois la grille 25x25 réussie, le joueur gagne la partie et est renvoyé sur le choix de la grille.

En annexe (4.1) se trouvent des exemples de parties en mode évolutif.

2.6 Didacticiel

Une partie en mode didacticiel est une partie guidée afin que l'utilisateur comprenne la logique de résolution d'une partie de *picross*. Les mouvements de celui-ci sont donc restreints aux coups (cocher ou colorier) que lui dit de jouer l'application.

2.7 Erreurs

E1 : Sélection d'un chapitre non déverrouillé.
Peut arriver lors de la sélection d'un chapitre.
Traitement : accès bloqué au chapitre et message rappelant le nombre d'étoiles nécessaires pour y accéder.

E2 : Arrêt de la partie en cours.
Peut arriver en cours de partie.
Traitement : lors du redémarrage de la partie, reprise au dernier point de sauvegarde.

E3 : Lancement d'une deuxième instance.
Dans ce cas, un message d'erreur affiche que le lancement a été bloqué.

E4 : Quitter le menu Options sans validation de celles-ci.
Affichage d'un message prévenant que les modifications n'ont pas été enregistrées avant de fermer le menu.
Le message propose d'enregistrer les modifications ou de les perdre, avant de quitter le menu.

2.8 Aides

L'utilisateur possède un nombre d'aides gratuites quand il commence à jouer et peut en récupérer en terminant un certain nombre de grilles. Quand il est en jeu, il peut les utiliser afin de se débloquer sans pénalité de temps. S'il ne possède plus d'aides gratuites, il peut toujours en utiliser une, mais celle-ci aura un coût (en temps) proportionnel à la difficulté du picross qui sera appliqué à chaque utilisation. La difficulté influence aussi le type d'aide appliqué :

En mode facile : coloriage de toute une ligne ou colonne

En mode normal : coloriage d'une case correcte

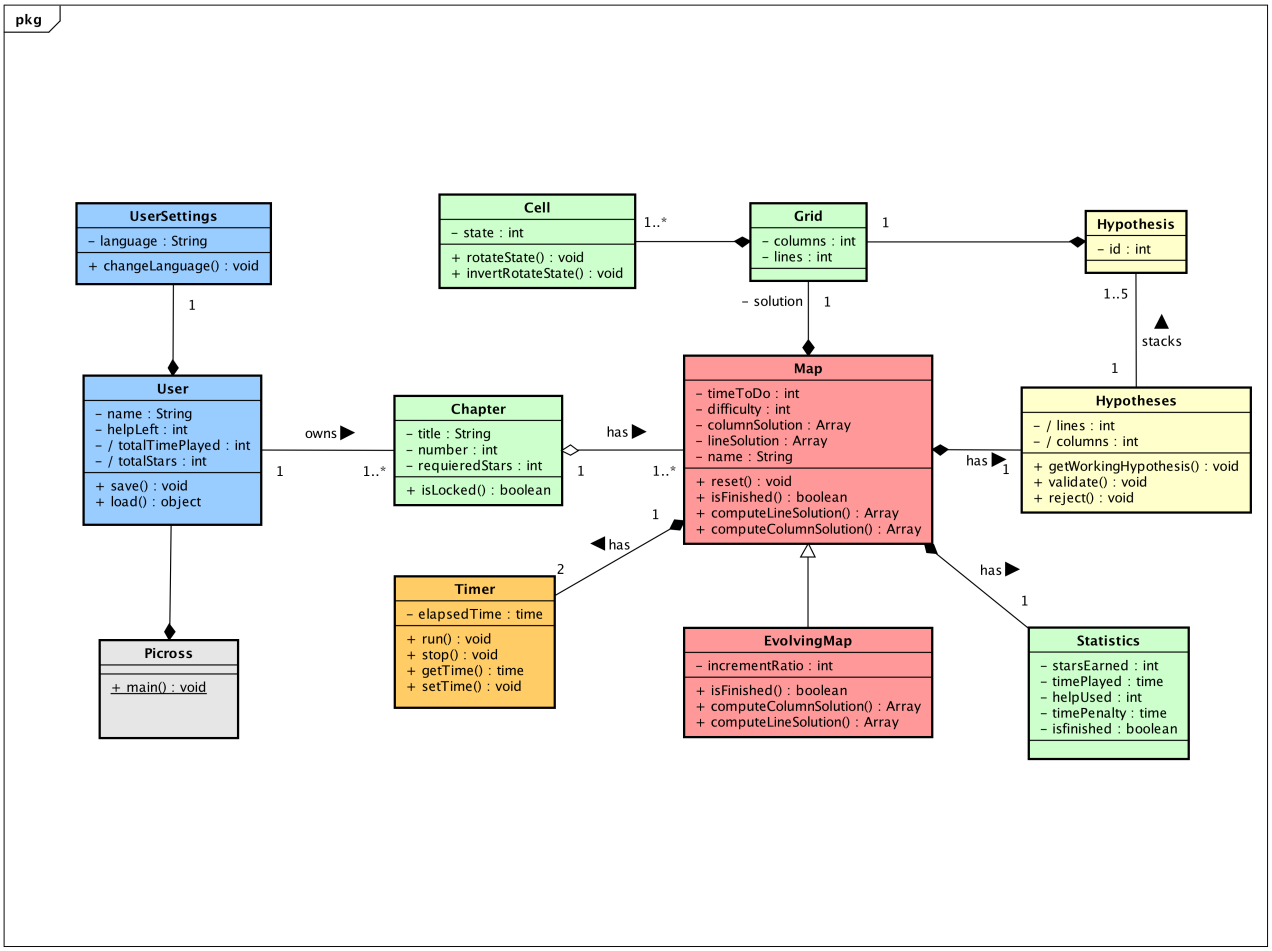
En mode difficile : indication d'une ligne ou colonne où se trouve une case correcte non coloriée

L'aide donnée en mode normal et difficile indique une case que le joueur aurait pu trouver par lui-même, sans connaître la solution.

3 | Conception détaillée

3.1 Diagramme de classe

FIGURE 3.1 – Diagramme de *Classes*



Le diagramme de classe ci-dessus présente les différentes classes pour notre application. Les classes nécessaires à l'interface graphique ne sont pas présentes afin de ne pas compliquer le diagramme.

Nous avons tout d'abord la classe *Picross* qui est la racine représentant l'ensemble du jeu et permettant de le lancer. Cette classe est liée à la classe *User* possédant un nom et un nombre d'aides gratuites restantes. Elle possède aussi des statistiques sur le temps total joué et le nombre d'étoiles totales acquises afin de permettre l'utilisation du jeu par plusieurs personnes différentes et ainsi avoir des sauvegardes différentes. Ces sauvegardes sont automatiques (méthode `save()`) et sont rechargées à la reconnexion du joueur (méthode `load()`). Chaque utilisateur possède des paramètres tels que celui de la langue grâce à la classe *UserSettings* qu'il peut modifier par l'intermédiaire de la méthode `changeLanguage()`.

Un utilisateur possède plusieurs chapitres grâce à la classe *Chapter* possédant un nom, un numéro de chapitre, ainsi que le nombre d'étoiles nécessaires au déverrouillage. Lorsque le joueur possède un nombre d'étoiles supérieur ou égal à celui nécessaire au déverrouillage, le chapitre est considéré comme déverrouillé. La méthode `locked?()` l'indique en renvoyant `false`.

Un chapitre possède plusieurs cartes de classe *Map* possédant une difficulté et le temps moyen pour réaliser la grille, nécessaire au calcul du score une fois la map finie. Une map peut également être réinitialisée par l'utilisateur avec la méthode `reset()` afin de recommencer sa partie.

Une map est composée d'une grille de picross avec sa solution (classe *Grid*) elle-même composée de plusieurs lignes et colonnes de cellules (classe *Cell*) noires, blanches ou cochées (marquées comme étant réellement blanches). Chaque map possède ses propres statistiques (classe *Statistiques*) composées (si la grille a été finie au moins une fois) du nombre d'étoiles gagnées, du temps joué, des aides utilisées et du temps de pénalité. Chaque carte possède également deux *Timer*s permettant de connaître le temps écoulé en partie ainsi que de calculer le score final une fois la grille réussie. Avoir deux *timers* permet de séparer le temps réel du temps de pénalité associé aux aides utilisées.

Une map est également composée d'un objet hypothèse (classe *Hypotheses*) comportant plusieurs grilles d'hypothèse (classe *Hypothesis*) possédant chacune un numéro et une couleur unique. Le joueur joue sur la première grille d'hypothèse et peut, s'il le souhaite, en ouvrir d'autres. Le mode hypothèse contient plusieurs méthodes : `reject()` permettant de rejeter l'hypothèse et donc de l'effacer, `validate()` permettant de valider l'hypothèse, de sortir de cette hypothèse tout en modifiant la grille de base avec les changements effectués dans l'hypothèse. Enfin, il y a la méthode `getWorkingHypothesis()` pour démarrer une nouvelle hypothèse.

4 | Annexes

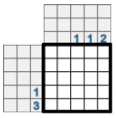
4.1 Exemple partie évolutive

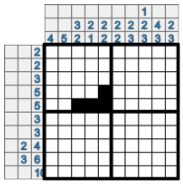
Ci-joint sont présents des exemples de grilles évolutives, c'est-à-dire de grilles qui s'agrandissent au fil de la partie. Deux exemples de grilles sont disponibles sous forme de GIF animé aux liens suivants :

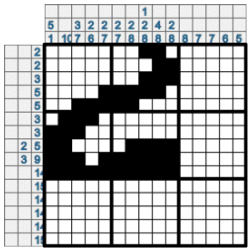
— `https://picross.vlntn.pw/Documents/Images/cat.gif`

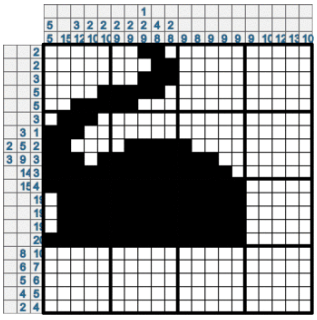
— `https://picross.vlntn.pw/Documents/Images/mouse.gif`

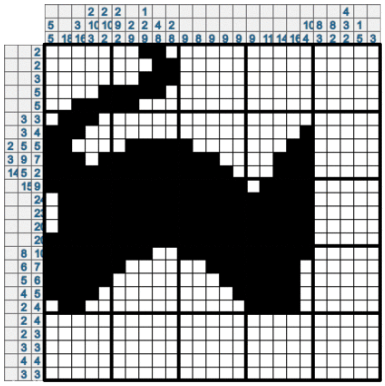
Le premier gif est également disponible sous format d'un ensemble d'images :

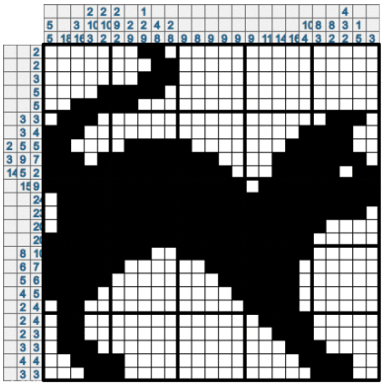












4.2 Maquettes de l'interface

Ci-dessous les maquettes réalisées pour l'interface de notre application en français.

Lorsque le joueur lance l'application, il arrive sur l'écran (4.1) lui demandant de rentrer son pseudo pour permettre à chaque joueur d'avoir son propre profil. S'il rentre un pseudo déjà existant, il charge son profil avec ses statistiques et ses chapitres débloqués sinon un nouveau profil est créé.

L'utilisateur arrive ensuite sur la page d'accueil du logiciel (4.2) lui permettant de choisir parmi les différentes fonctionnalités. Le bouton *Quitter* ferme l'application.

Le bouton *Options* renvoie sur la page d'options (4.3) qui permet à l'utilisateur de modifier ses préférences de jeu.

Le bouton *Règles* permet d'afficher les règles générales du *picross* (4.4). L'utilisateur peut voir ses statistiques et celles des autres utilisateurs en cliquant sur le bouton *Classement*.

Enfin, en cliquant sur le bouton *Jouer*, l'utilisateur est envoyé sur la sélection des chapitres (4.5) comportant tous les chapitres du jeu, didacticiel et mode évolutif inclus.

Après avoir choisi un chapitre, il arrive sur la sélection de la grille (4.6) ainsi que le nombre d'étoiles obtenues sur les grilles déjà réussies. Lorsqu'il a choisi sa grille, la partie commence (4.7).

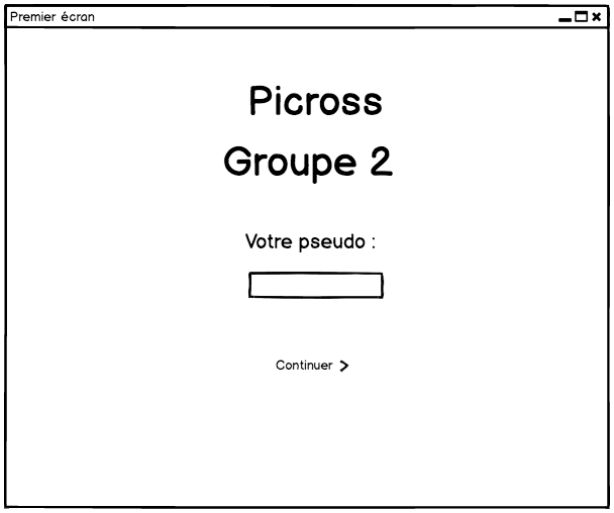


FIGURE 4.1 – Écran de connexion

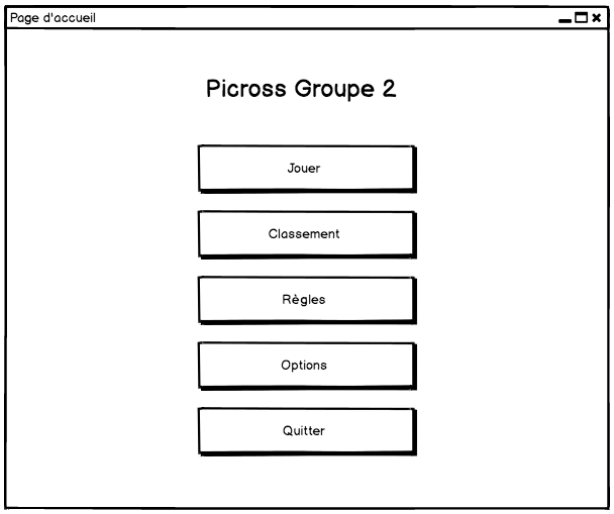


FIGURE 4.2 – Page d'accueil

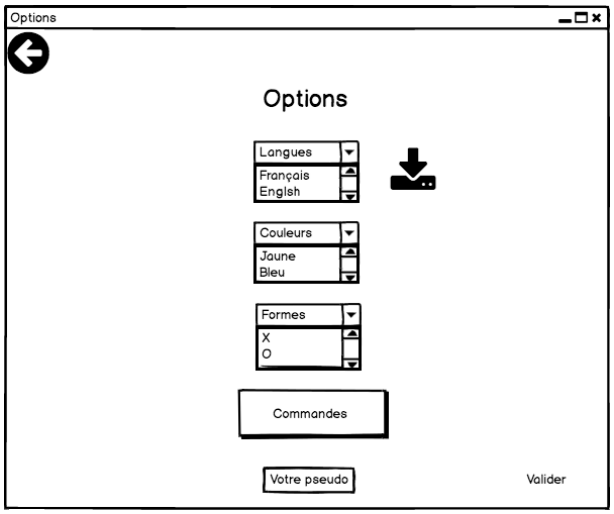


FIGURE 4.3 – Options

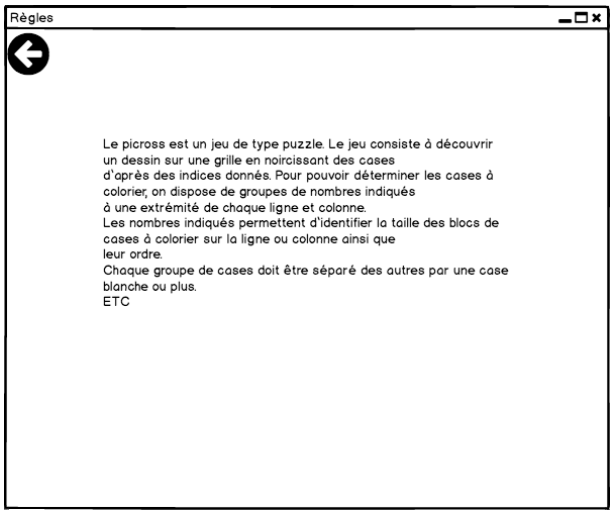


FIGURE 4.4 – Règles

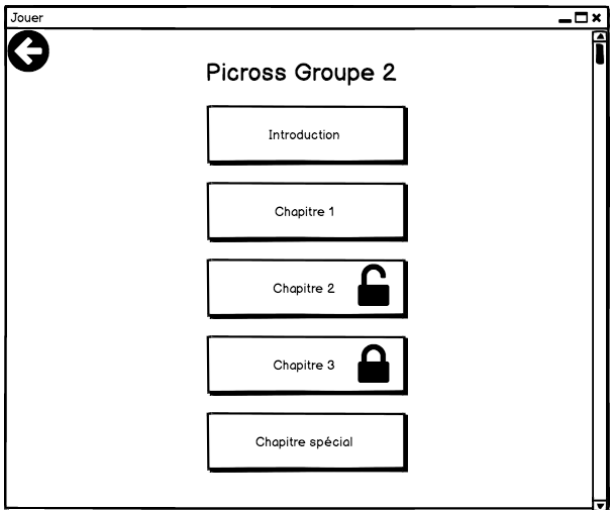


FIGURE 4.5 – Sélection du chapitre

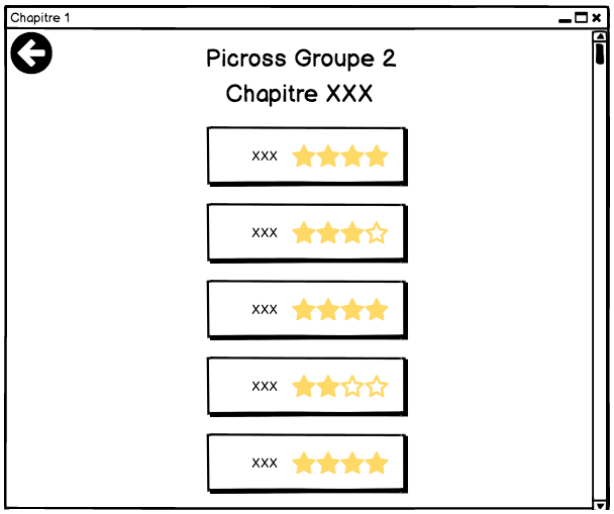


FIGURE 4.6 – Sélection de la grille

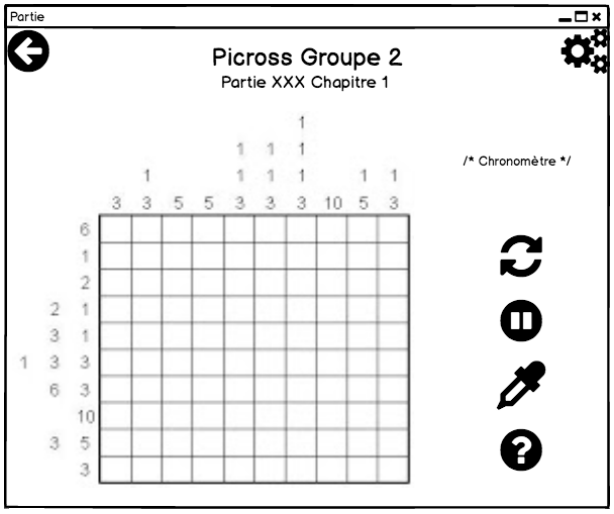


FIGURE 4.7 – Partie

4.3 Proposition de nom d'application et de logo

Nous avons pensé à un nom d'application *Ruþycross* (Ruþycross). Celui-ci est la combinaison de *Ruby* et de *Picross*, en utilisant le caractère ASCII «*þ*». En utilisant la couleur principale du logo de *Ruby* (`#AA1401`), et un picross représentant le «*þ*» de *Ruþycross*, nous avons réalisé un logo pour notre application. Celui-ci se trouve ci-dessous.

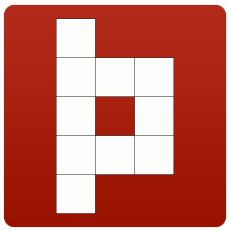


FIGURE 4.8 – Logo de l'application