# 3I008 — Projet
# ORush
# Partie 2 : Interface graphique

Ariana Carnielli

## Table des matières

**1 Introduction** **1**

**2 Fichier HTML** **1**

**3 Fichiers de la partie 1** **2**

**4 Maps** **2**

**5 ORush_js** **3**
5.1 Début d'une partie . . . . . 3
5.2 Solution automatique . . . . . 3
5.3 Sélection d'un bateau . . . . . 4
5.4 Mouvement d'un bateau . . . . . 4

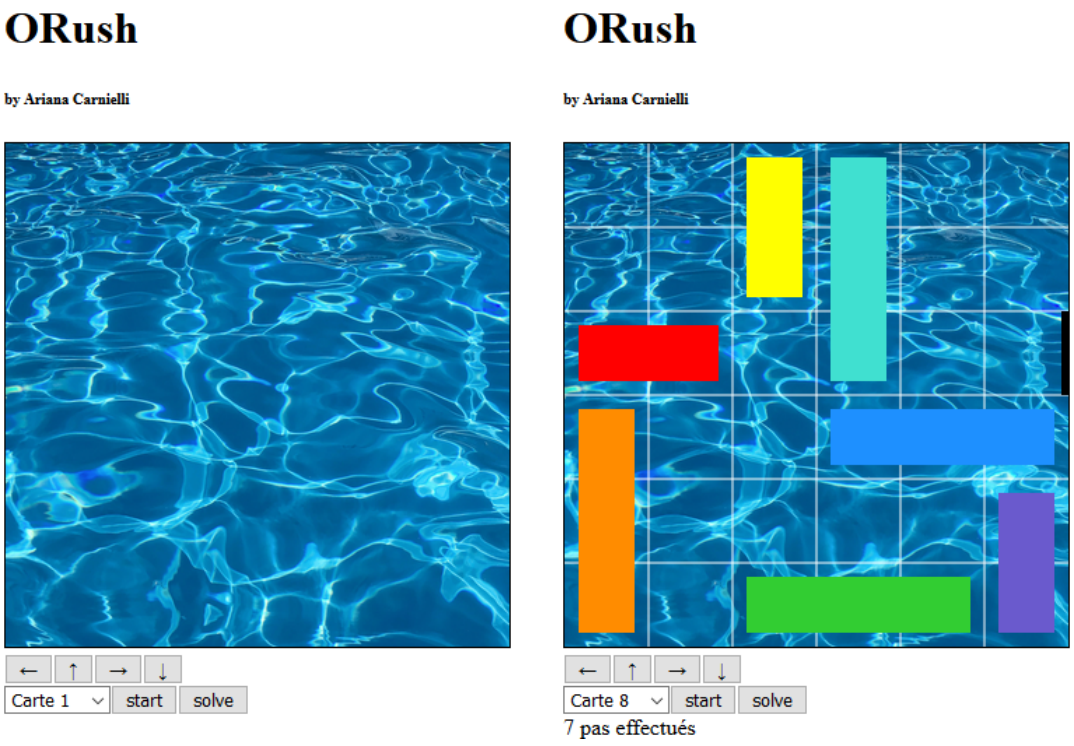
**6 Makefile** **4**

## 1 Introduction

Dans cette deuxième partie du projet, après l'implémentation du noyau du jeu *ORush* avec des fonctions permettant la manipulation des états, l'implémentation des mouvements et la résolution automatique d'une grille, cette deuxième partie du projet s'intéresse à l'implémentation d'une interface graphique pour le jeu. Pour cela, on a utilisé l'interopérabilité entre OCaml et JavaScript à l'aide de l'API `gen_js_api`, qui se base sur le compilateur `js_of_ocaml`. L'interface du jeu codé est donné dans la Figure 1. La suite de ce document décrit les techniques utilisées et les principaux choix d'implémentation faits.

## 2 Fichier HTML

Le jeu est chargé à travers un fichier `index.html`. Ce fichier est minimaliste : il crée une balise `h1` pour le titre, un `canvas` pour le jeu en soi et charge le script




(a) (b)

FIGURE 1 – Interface du jeu *ORush* (a) à l'initialisation et (b) au cours de la résolution d'une grille.

principal contenant le jeu à travers une balise `script`. Tous les autres éléments nécessaires pour le jeu, comme les éléments sur le *canvas*, les boutons et le titre de la page sont crées à partir du code OCaml.

# 3 Fichiers de la partie 1

Les fichiers `port.ml`, `moves.ml` et `solver.ml` codés à la partie 1 de ce projet sont utilisés dans cette partie et constituent le noyau du jeu. Les versions utilisées sont essentiellement identiques à celles de la partie 1, l'unique différence étant la création de deux fonctions dans `port.ml` pour permettre d'extraire plus facilement des informations sur un bateau.

# 4 Maps

Un fichier `maps.ml` a été crée, contenant 14 cartes différentes pour le jeu *ORush* et la palette de couleurs utilisées pour dessiner les bateaux. Il s'agit des 12 cartes fournies en exemple pour la partie 1 plus deux autres crées pour tester d'autres configurations de jeu.

Ce fichier a été crée car il n'a pas été possible de charger automatiquement les cartes depuis des fichiers `.txt`. En effet, la lecture des fichiers à l'aide de la fonction OCaml `open_in` levait toujours une exception indiquant que le fichier n'avait pas été trouvé. Après des tests en utilisant les fonctions `Sys.getcwd` et `Sys.readdir`,

on a remarqué que le code JavaScript obtenu à partir de la compilation d'OCaml travaillait sur un dossier virtuel `/static/` qui était vide. La lecture en utilisant le chemin relatif du fichier de tests n'était donc pas possible, et comme la lecture par chemin absolu rendrait le jeu inutilisable sur toute autre machine, on a décidé d'implémenter les cartes comme code en OCaml, permettant donc leur utilisation partout. Cela a cependant l'inconvénient de rendre plus difficile la création de nouvelles cartes.

# 5 ORush_js

Le fichier `orush_js` contient la partie principale du code de l'interface graphique. Il contient une fonction `main`, appelée au démarrage du jeu, qui fait les configurations principales, fixant la taille du *canvas* à 360px × 360px, affichant le titre du jeu et les boutons de contrôle et déclarant toutes les *listeners* associés aux boutons, au *canvas* et aux flèches du clavier. Cette fonction principale contient aussi des variables mutables contenant l'état courant du jeu, le numéro de la carte courante, le bateau sélectionné, un compteur de mouvement et la solution donnée par le solveur automatique. Ces variables sont modifiées par les fonctions *listeners* quand nécessaire, le choix de les implémenter comme mutables provient du fait que les fonctions *listeners* doivent obligatoirement retourner un `unit`.

## 5.1 Début d'une partie

Le jeu commence lorsque l'utilisateur appuie sur le bouton `start`, qui appelle la fonction homonyme. Cette fonction nettoie l'écran à l'aide d'une fonction dédiée `clear_canvas`, dessine les lignes horizontales et verticales constituant le grid et la sortie à l'aide de la fonction `set_grid`, cherche l'état sélectionné dans le menu et redémarre toutes les variables mutables créées dans `main`. Elle garantit aussi que les flèches sont habilitées et que le bouton permettant de faire un nouveau pas de la solution automatique n'est pas affiché.

La fonction `start` se charge aussi d'afficher l'état initial du jeu. L'affichage d'un état du jeu est fait par la fonction `draw_state`, qui ne fait que parcourir la liste de bateaux d'un état et les afficher par la fonction `draw_boat`. Cette fonction dessine chaque bateau comme un rectangle, à une distance de 10px des bords de la case, avec des couleurs différentes basées sur la lettre qui représente le bateau.

## 5.2 Solution automatique

L'utilisateur a toujours l'option de demander la solution du jeu, à travers le bouton `solve`. Ce bouton appelle la fonction `solve_start`, qui calcule la solution automatique à partir de l'état courant, fait apparaitre un nouveau bouton `next step`, affiche la solution dans une balise `div` créée au préalable pour ce but et désactive les touches du clavier, les boutons de flèche et le bouton `solve` lui-même. Chaque clic dans `next step` appelle la fonction `solve_step`, qui met l'écran à jour avec le prochain pas de la solution, incrémente le compteur de mouvements et, à la fin, réactive les boutons qui étaient déactives et cache le bouton `next step`. Si l'utilisateur appuie sur `start` avant la fin de la solution, le jeu redémarre et les boutons

sont réactivés. Si l'utilisateur a choisi de rester sur la même grille, l'affichage de la solution reste à l'écran ; sinon, il est effacé.

### 5.3 Sélection d'un bateau

La sélection d'un bateau se fait par un clic de la souris sur le bateau correspondant (ou sur l'espace entre le bateau et le bord de sa case). Par défaut, le bateau A est sélectionné lorsqu'une grille est chargée. La sélection d'un bateau lors d'un clic se fait par un *listener* de clics sur le *canvas*, qui appelle la fonction `boat_of_click`. Cette fonction calcule la case correspondante à partir des coordonnées du clic relatives au coin en haut à gauche du *canvas* et utilise la fonction `Port.grid_of_state` pour déterminer quel bateau a été sélectionné. Un clic dans une case ne contenant pas de bateau désélectionne tout bateau (ce qui correspond à affecter le caractère `'~'` à la variable de `main` contenant le bateau courant).

### 5.4 Mouvement d'un bateau

Le mouvement d'un bateau peut être fait par les boutons de flèches de l'interface ou par les flèches du clavier. Pour ce faire, outre les *listeners* de chaque bouton, un *listener* a aussi été codé pour le clavier en attendant l'évènement `keydown`. Lors de cet évènement, si la touche appuyée est une des flèches du clavier (codes 37 à 40) et on n'est pas dans le mode de solution automatique, on appelle la fonction `move_of_click`, qui est aussi appelée par les boutons.

La fonction `move_of_click` vérifie d'abord qu'un bateau est bien sélectionné avant de faire tout mouvement. Elle crée ensuite un mouvement et essaie de l'appliquer à l'aide de la fonction `Moves.apply_move`, en ignorant tout mouvement à gauche ou à droite des bateaux dans la verticale ou vers le haut ou le bas des bateaux à l'horizontale. Si le mouvement peut être appliqué, cette fonction se charge aussi de le faire et de mettre à jour l'état courant et le dessin à l'écran. Si l'utilisateur gagne la partie, elle affiche aussi une fenêtre avec un message contenant le nombre de pas réalisés. Si le mouvement n'est pas possible, la fonction l'ignore.

## 6 Makefile

Pour compiler le jeu, un fichier `Makefile` a été crée basé sur celui donné dans le TME11. On a rajouté la compilation des fichiers de la partie 1 du projet ainsi que du nouveau fichier `maps.ml` et adapté la compilation par `js_of_ocaml` de notre fichier principal `ocaml_js.ml`.

Ce fichier `Makefile` doit connaitre l'emplacement du dossier `opam` et des fichiers de la version `4.06.1` d'OCaml. Ce dossier est typiquement `~/.opam/4.06.1`, cependant l'utilisation de ce chemin donne une erreur de compilation avec `js_of_ocaml`. En effet, `js_of_ocaml` semble incapable de trouver le fichier `findlib.conf` dans `~/.opam/4.06.1/lib/`, mais il le trouve sans problèmes si `~` est remplacé par le chemin absolu du répertoire personnel. Ainsi, une variable `OPAM_PATH` est rajoutée au début du `Makefile` et doit être modifiée avant la compilation pour contenir le chemin absolu vers la version `4.06.1` dans `.opam`.