Objectif

Réaliser une application autonome javascript qui communique avec une api web REST.

Besoin

Le but du jeu est de faire le plus grand nombre de récoltes possibles.

Le principe est de devoir remplir les citernes d'eau de 3 champs qui ont besoin d'être irrigués tous les jours (seconde dans le jeu) pour pousser.
Le joueur a une réserve d'eau globale qu'il peut remplir en achetant de l'eau. Il irrigue ensuite ses champs en puisant dans cette réserve globale. Le joueur vend les champs arrivés à maturité pour récupérer de l'argent qu'il réinvestira dans de l'eau.
Si un champ est en rupture d'eau (plus d'eau dans la citerne du champ) sa récolte est perdue. Quand la citerne sera réalimentée, le décompte de maturité du champ repart à 0.

La consommation des champs va augmenter pour simuler une aridité. Le rythme de jeu va donc s'accélérer jusqu'à avoir les trois champs sans eau dans leurs citernes. La partie est alors terminée. Le nombre de récoltes effectuées est le score de la partie.

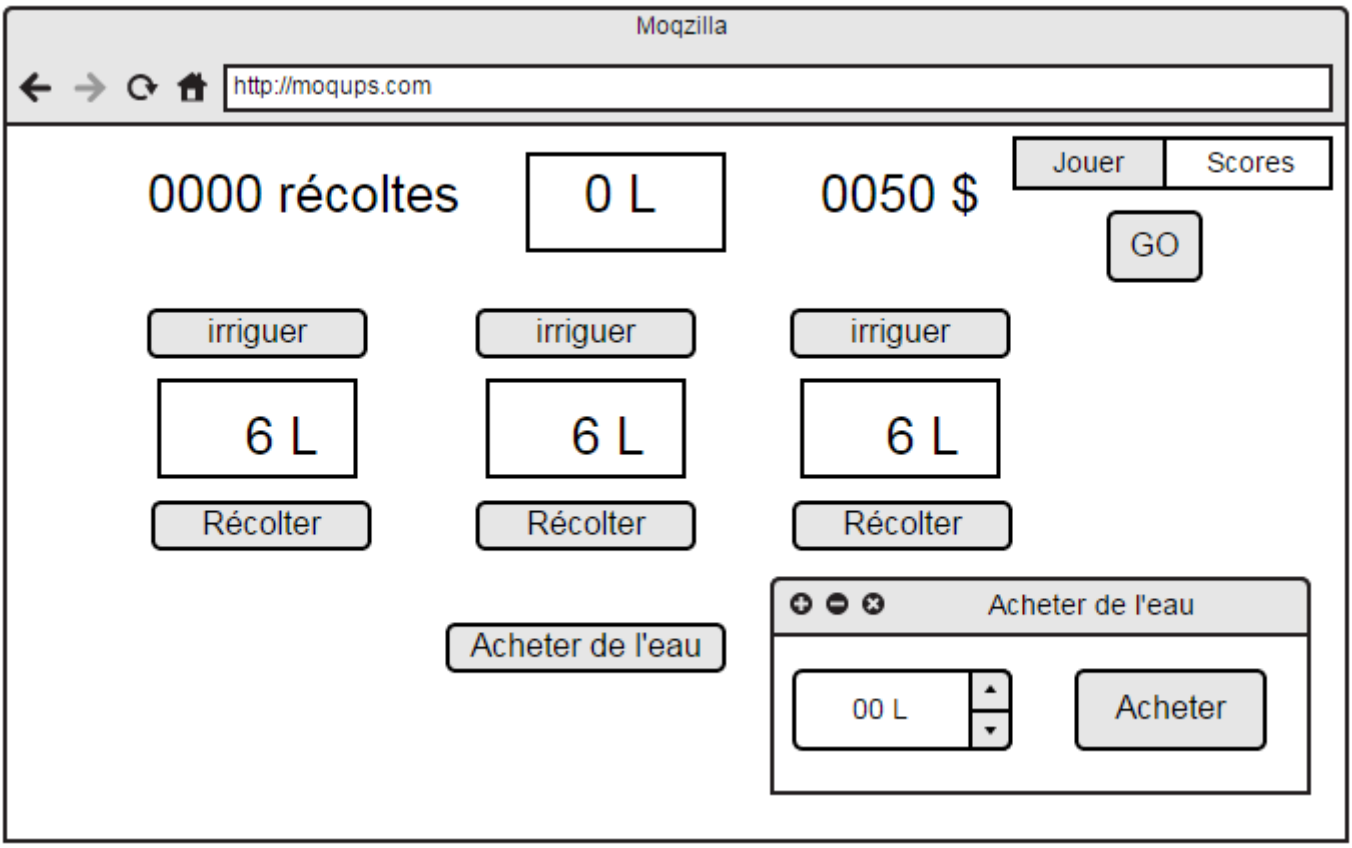
Le score est envoyé sur un serveur qui met à disposition une API web Service REST pour recevoir les nouveaux scores et mettre à disposition les scores déjà enregistrés. Le joueur peut consulter les scores de tout le monde.

L'objectif de temps moyen de jeu est de 10 minutes et le temps maximum 15 minutes.

Règles de gestion

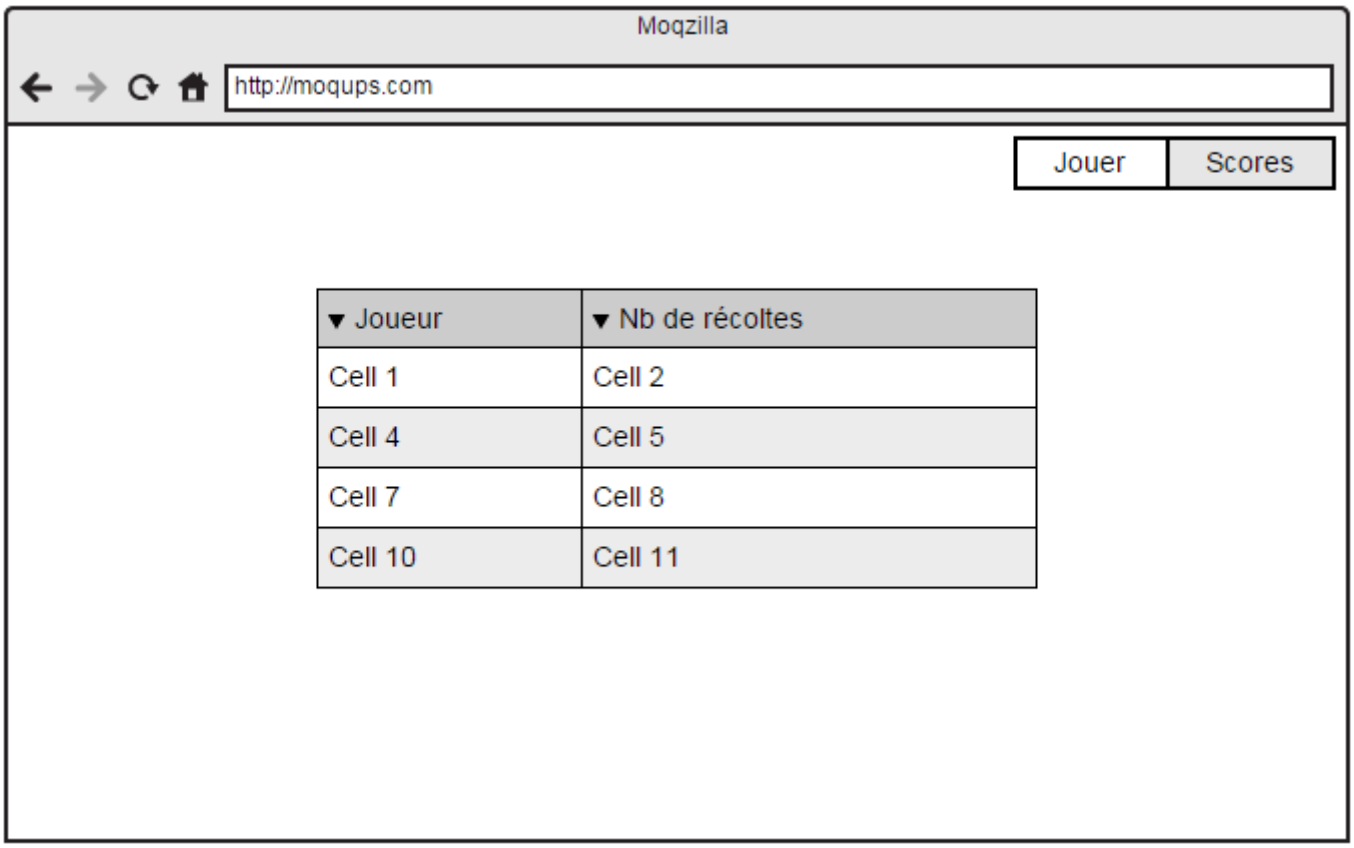
- Argent initial: 50$,
- Réserve initiale: 3L / citerne + 3L global,
- Consommation initiale: 1L / seconde,
- Consommation maximum: 2L / seconde,
- Vous devez trouver une règle de l'augmentation de la consommation pour atteindre les objectifs de temps de jeu,
- Nombre de secondes pour qu'un champ soit mûr: 20 secondes,
- Prix de l'eau: 1$ / L,
- Prix d'une récolte: 40$,
- Un champ prêt à récolter ne consomme pas d'eau.

Ecran principal



Interactions :
L'achat d'eau ouvre un popin et met en pause le jeu.
La couleur du bouton récolter change de couleur quand le champ est mûr.
Une fenêtre s'ouvre à la fin du jeu pour saisir le nom du joueur.

Ecran des scores



Contraintes techniques

1. Le serveur REST est fourni par le formateur,
2. Utilisation de JQuery pour la manipulation du DOM et les appels AJAX,
3. Application du MVC sur toute la conception de l'application JavaScript. Vous devez avoir des objets View, des objets Controller, et un objet Model qui communiquent selon les principes du MVC. Le passage de la vue Jeu à la vue Score n'arrête pas le modèle qui applique les règles de gestion toutes les secondes.

API

Route	Verbe	Description
/scores/	GET	Récupère la liste des scores.

Exemple de résultat récupéré :

```
{
  "status": "OK",
  "list": [
    {
      "name": "Erwan",
      "score" : 24
    },
    {
      "name": "Gabriel",
      "score" : 30
    },
    {
      "name": "Benjamin",
      "score" : 56
    }
  ]
}
```

Route	Verbe	Description
/scores/	POST	Permet d'envoyer un score (nécessite un attribut 'name' et 'score' dans le body).

Exemple de résultat envoyé :

```
{
  "name": "Gabriel",
  "score": 24
}
```

Le retour de cette route est identique à la route /scores en GET.

Coup de pouce

- Utiliser setInterval() pour la gestion du temps.
- Les scores des joueurs récupérés en AJAX modifient le modèle qui notifie la vue.