GuessWhat

4 - GO

Description du document

Titre	GuessWhat
Date	03/08/2017
Auteur	thomas@idequanet.com
Responsable	Sophie Viger
E-mail	pedagowac@epitech.eu
Sujet	GuessWhat
Mots-clés	GO, JS, sockets, JSON
Version	1.0

Tableau des révisions

Date	Auteur	Sections	Commentaire
03/08/2017	Thomas Solignac	Toutes	Création du document.

Modalités de rendu

Serveur	Git epitech
Authentification	Blih
Dépôt	guesswhat
Droits	ramassage-tek :r odin1.duclos@epitech.eu:r

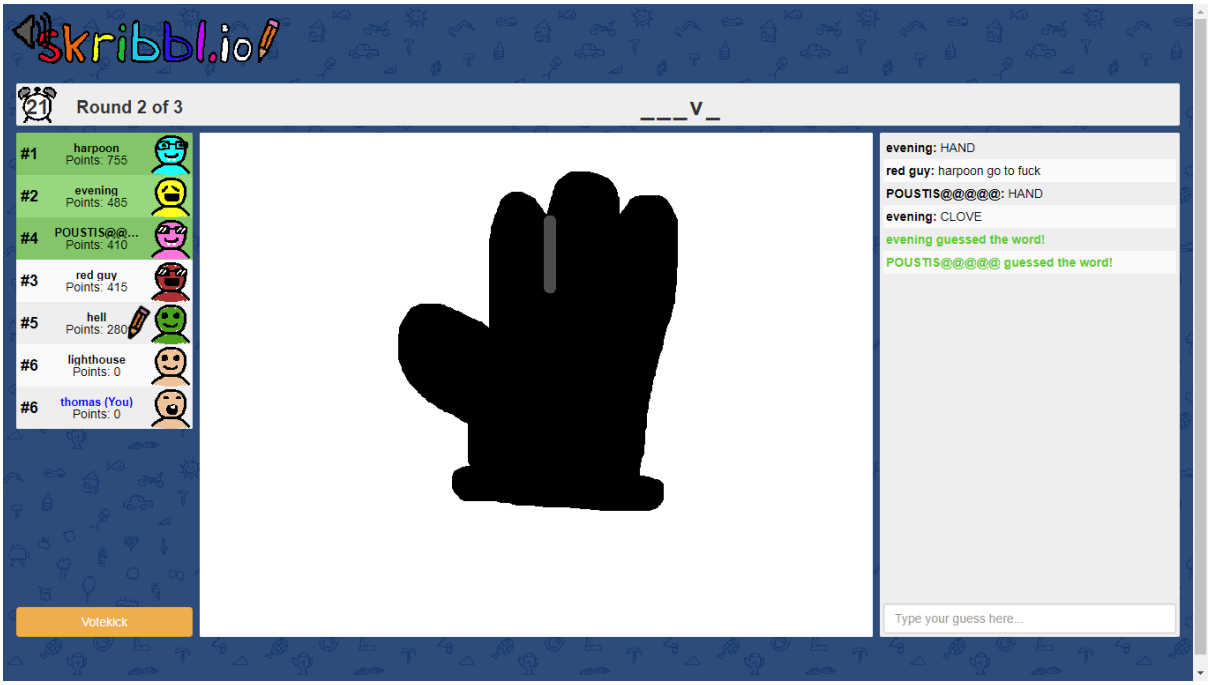
Contents

Architecture 5
Serveur Go 5
Front web 5
Communication front-serveur 5
Proposition de déroulé 6
1- Un simple ping-pong server front 6
2- La même chose en JSON 6
3- Un chat minimaliste 7
4- Un chat un peu plus complet 7
5- Ajouter le dessin 8
Et la suite ? 8
Conseils 9
Améliorations possibles 9

Introduction

Le projet GuessWhat consiste à créer un équivalent en ligne et simplifié du jeu Pictionary. Le principe est simple : un joueur reçoit un mot, et il doit le faire deviner à son équipe au plus vite. Ici nous allons faire légèrement différent : il n'y a pas d'équipe, le premier joueur qui devine le mot gagne des points, proportionnels à sa rapidité de réponse.

On trouve quelques sites en ligne qui proposent un concept similaire :



Exemple de site similaire : Skribbl.io

Une fois le jeu de base créé, vous serez invités à y ajouter une touche d'originalité. Je vous proposerai quelques directions possibles. Un des intérêts de ce projet étant la possibilité de mettre un lien vers votre projet dans votre CV/LinkedIn.

Notions

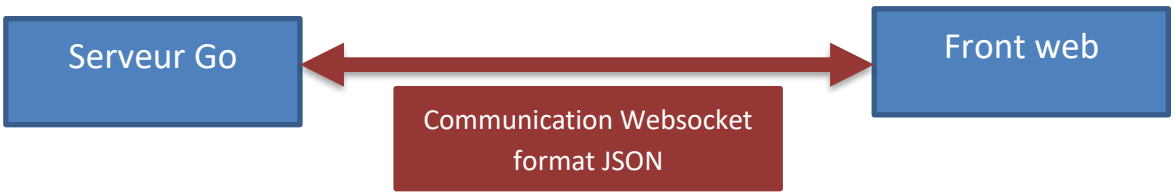
De nombreuses notions seront abordées dans ce projet. On y trouve entre autres :

- GO
- Moteur de template JS
- Websockets
- JSON
- API
- Problématiques de parallélisation

Procédure

Architecture

Le projet sera composé de deux parties. Un serveur en Go et un front web.



Serveur Go

Le serveur Go permet d'avoir plusieurs joueurs qui partagent la même partie, depuis différentes pages web.

Front web

Vous avez toute liberté de technologie et de réutilisation de projets passés. Il va communiquer les interactions utilisateur au serveur, et utiliser les informations que lui envoie le serveur pour afficher l'avancement jeu (et la gestion des rooms).

Communication front-serveur

Voici une idée générale des échanges entre le serveur Go et le Front Web

D'abord, un joueur crée une room :

1. Le front envoie au serveur une demande de création de room, avec son pseudo
2. Le serveur renvoie le code de la room
3. Le front redirige le joueur vers la room, en tant qu'administrateur de la room
4. Le front affiche le code de la room pour que d'autres puissent rejoindre le jeu
5. Quand l'administrateur appuie sur le bouton « start », le front transmet la demande de démarrage du jeu au serveur.

Ensuite, un joueur rejoint la room

1. Le front demande la liste des rooms publiques.
2. Le serveur envoie cette liste.
3. L'utilisateur clique sur une room, et le front charge l'URL correspondante
4. Le front se connecte au serveur en précisant le code de la room
5. Le serveur renvoie la liste des joueurs connectés. Pour le dessinateur, il envoie le mot à faire deviner.
    a. Si le jeu a déjà commencé il envoie également le dessin déjà amorcé.
    b. Sinon, le serveur attend que l'administrateur appuie que sur le bouton start
6. Le front envoie au serveur les interactions utilisateurs

   a. Pour la personne qui dessine : le dessin au fur et à mesure
   b. Pour les autres : les messages dans le chat
7. Quand une personne gagne ou que le temps est écoulé, le serveur envoie un message au front avec le résultat de la partie. Le serveur ferme la room.

Proposition de déroulé

La difficulté première de ce projet réside dans l'ordre de création des éléments. Par quoi commencer ?

Effectivement il sera difficile de tester le serveur sans le front, et le front sans le serveur. On va commencer par créer une page simple qui permet de faire discuter le serveur et le front web, qu'on enrichira petit à petit. Le système de room viendra en tout dernier.

1- Un simple ping-pong server front

Je vous invite pour commencer, dès que vous avez les bases du Go, à faire une simple communication entre le front et le serveur.

Coté serveur Go :

1. Ouvrir un serveur websocket
2. Quand un client se connecte, rentrer dans une boucle qui lui renvoie tout ce qu'on reçoit, passé en majuscule.
   a. Voir le package « strings » pour passer en majuscule.

Coté front web :

1. Mettre un champ texte avec un bouton « envoyer »
2. Se connecter à la websocket (le serveur Go doit être lancé)
3. Quand on clique sur le bouton « envoyer », envoyer le contenu du champ texte à la websocket
4. Afficher la réponse en console.log ou ailleurs

Note importante :
Je vous invite à utiliser au maximum tous les packages standards à disposition.

Pour les websockets, je vous invite à utiliser le package suivant :

https://godoc.org/golang.org/x/net/websocket

2- La même chose en JSON

On va maintenant faire en sorte que cet échange soit entièrement en JSON :

```
{
  message : « test »
}
```

A partir de là, vous avez acquis la base des échanges websocket en JSON.

Utilisez bien tous les outils à disposition pour vous faciliter l'encodage et le décodage du JSON.

Note : Je vous invite à utiliser le JSON « decoder » plutôt que de faire des « read » directement. https://golang.org/pkg/encoding/json/#example_Decoder_Decode_stream

Parfois, les websockets segmentent les messages !

3- Un chat minimaliste

A ce stade, on va faire un chat simple. Toutes les pages partageront le même chat. Vous allez donc :

- Coté front, afficher clairement les messages reçus du serveur
- Coté serveur, quand un client se connecte, l'ajouter à la liste de clients connectés.
    - Il faudra donc conserver la liste des clients en globale, et la maintenir en fonction des connexions/déconnexions
- Coté serveur, quand vous recevez un message, le renvoyer à tous les clients.

Note importante - Une nouvelle notion indispensable ici : les mutex

Coté Go, quand une websocket reçoit une connexion, la fonction appelée (le « handler ») est lancée dans un processus parallèle (une goroutine plus exactement). Ça veut dire que le code s'exécute à plusieurs endroits en même temps, au même moment. Ça permet d'utiliser toute la puissance de votre processeur, mais ça implique quelques contraintes.

A un moment donné, vous aurez besoin de stocker des éléments en globale. Par exemple vous aurez besoin de stocker la liste de tous les clients connectés. Ainsi, dans le chat, quand un client vous envoie un message, vous le transmettez à tous les clients connectés.

Si deux goroutines modifient la même variable au même moment, vous allez avoir un crash. Pour remédier à ça, vous utiliserez une notion très importante, les mutexs : https://golang.org/pkg/sync/

4- Un chat un peu plus complet

On va rajouter les nicknames et garder l'historique du chat : quand quelqu'un se connectera, il verra tous les messages passés.

- Coté serveur, il serait utile dans un premier temps, de faire un objet qui contienne la connexion websocket et le nickname du client.

Maintenant, il va falloir un type de message supplémentaire pour pouvoir changer son nickname.

```
{
  nickname : « guillaume »
}
```

Le serveur reçoit maintenant deux paquets possibles : un nouveau nickname ou un message. Comment les différencier ?

Il y a deux solutions principales. Comme vous le savez maintenant, l'outil du Go qui permet de décoder un message JSON peut transformer le JSON texte en plusieurs objets de nature différentes. Par exemple, une map ou un objet.

Décoder dans une map oblige à effectuer des manipulations complexes de « cast » que je vous invite à éviter.

Le décodage vers un objet est plus simple : le décodeur remplit tout l'objet avec ce qu'il trouve dans le JSON. Dans cette deuxième possibilité, plus facile, je vous invite à mettre un champ « type » en haut de tous vos messages.

Exemple :

```
{
  type : « nickname »,
  nickname : « guillaume »
}
{
  type : « message »,
  message : « hello ! »,
  nickname : « guillaume »
}
```

Dans ce cas, je vous invite à avoir une première structure qui contient juste le champ « type ». Une fois que vous avez identifié de quel type de message il s'agit, vous décodez à nouveau le texte reçu, cette fois dans le bon type de structure.

En particulier avec les structures très différentes à venir, ça va vous gagner du temps.

5- Ajouter le dessin

Sur la partie front, ajouter un Canvas qui permet de dessiner dedans, et envoyer les mises à jour du dessin au serveur, pour qu'il les redistribue à tous les clients connectés.

Et la suite ?

A partir de là, vous avez résolu toutes les problématiques techniques de base. Félicitations ! Il s'agit maintenant de compléter les fonctionnalités.

1. Mettre un bouton start
2. Faire que le serveur envoie le mot à deviner, au hasard depuis une liste
3. Spécifier le joueur qui a gagné
4. Tirer au sort le prochain qui va dessiner
5. Ajouter le système de room.

Conseils

- Affichez systématiquement tous les retours d'erreur, et proprement. Le debug deviendra un enfer sans ça. Le Go vous permet de le faire très finement, c'est un intérêt fort du langage.
- Les packages standards du Go sont très nombreux et puissants. Usez et abusez !
- Le Go fournit pas mal d'outils puissants, dont certains bien pratiques, comme « go fmt » par exemple.
- Certaines notions abordées ici sont nouvelles. Prenez le temps de lire un peu plus globalement sur ces notions qui vous sont nouvelles, ça vous facilitera la compréhension de leur utilisation.
- Avoir le même protocole qu'un autre groupe peut éventuellement vous faciliter le travail.

Améliorations possibles

- Avatars personnalisés
- Système de room publique/privée
- Match équipe contre équipe
- Mots par thématique
- Outils de dessin (couleur, spray, épaisseur, etc.)
- Système de score
- Page d'administration des mots/thèmes