Gothello - Go 言語と WebGL によるオセロ(3D)

Web ブラウザを用いてオンラインで対戦できるオセロです。サーバー側はGo言語、クライアント側はJavascriptを利用しました。これを作った理由は単純で、作れる気がしたからです。これまで「オセロを作ろう」的なチュートリアルを見たことはなく、オセロのコードなどは全て自分で考えました。２Ｄだとありきたりな気がしたので３Ｄにしました。制作期間は一か月程ですが、そのほとんどが WebGL との戦いでした。Three.js 使えばもっと早く終わった気がします。

目次

1. 機能の要約

サーバー側

- WebSocketの接続待機
- プレイヤー同士を対戦させる
- ゲームの進行
- 進行状態の通信

利用したライブラリ (サーバー側)

- github.com/gorilla/websocket

クライアント側

- サーバーとWebsocketで接続する
- サーバーに対戦相手の検索を依頼
- サーバーから受け取ったゲーム情報をクライアントに反映
- WebGLを利用してゲームの状況を画面に描画

利用したライブラリ (クライアント側)

- svelte
- gl-matrix

利用した Web サイト、書籍

- WebGL: ウェブの 2D および 3D グラフィック
- WebGL の基本
- wgld.org

- API Docs - Svelte
- 実践プログラミング WebGL HTML & JavaScript による 3D グラフィックス開発

その他に利用したソフトウェアなど

- Blender (ボードとオセロの石をモデリングするために利用)

2. サーバーのビルド

サーバーはGo 言語(v1.16.5)で開発しました。 Go をビルドする際は Go の**プロジェクトルート**`./src/server/`**に移動**してから行います。

```
cd ./src/server
go -o gothello build main.go
```

無事完了すると`src/server`の中に`gothello.exe`が作成されているはずです。

3. クライアントのビルド

クライアントをビルドするには、`yarn`が必要です。(`npm`は未検証) また、Node.jsも必要になります。*(開発環境の Node.js は、v12.13.1 です)*

初めに、プロジェクトのルートに移動してから以下のコマンドを実行します。

yarnを使う場合

```
yarn install
yarn build
```

特にエラーが出なければクライアントのビルドは完了です。

また、クライアントとサーバーをビルドするこれらのタスクをスクリプトにしてあります。

```
./build.sh
```

4. サーバーの起動

まず、`gothello.exe`を`public`と**同じ階層に配置** します。 (`build.sh`を利用してビルドした場合は自動で配置されます。)

そして、このバイナリファイルを実行するために以下のコマンドを入力します。

```
#デフォルトではポート番号80で起動します
./gothello.exe
```

```
#また、特定のポート番号を使用したい場合は以下のようにします
./gothello.exe -port=8080

#実行権限がないとき
chmod u+rx gothello.exe
```

5. プレイ方法

Web ブラウザを起動し、アドレスバーにアドレスを入力します。

```
#サーバーを起動しているコンピューターでプレイする場合
http://localhost

#ポート番号を指定した場合
http://localhost:8080

#例: ローカルネットワークからプレイ
http://192.168.3.2
```

1. 「Click to find opponent」をクリックすると対戦相手の検索を開始します
2. 対戦相手が見つかるとゲームが開始します。
3. 「Your turn」と表示されているときがあなたのターンです。
4. ボードをクリックして石を設置できます。
5. ルールは一般的なオセロと変わらないので割愛
6. 石を設置できなくなるか、対戦相手が切断するとゲームが終了します。
7. 結果が表示されます。結果表示をクリックすると初期画面に戻ります。

6. 詳しいプログラムの流れ

マッチング

- 対戦相手の検索を要求されると、キューの最後尾にプレイヤーを追加する
- 次にキューの長さが 2 以上のとき、先頭の 2 プレイヤーで新しいゲームを開始する
- `Player`の線形リスト
- `push`の操作は、新しい要素を最後尾に連結するのみ
- `pop`の操作は、先頭の次の要素を先頭の要素にするのみ

ゲーム進行プログラム

- ゲームを開始すると、初期状態やプレイヤーの情報をクライアントに送る
- 現在ターン中のプレイヤーからクリック位置を受け取る
- 受け取った位置に石を設置した場合、「どの石を裏返せるか？」を調べる
    - 一枚も裏返せないなら、そこは設置できないので何もしない
    - そうでないなら裏返し、石を設置する。そして、ターンを更新する
- ターンを更新すると、そのターンで設置できる石の枚数を調べる
    - 一枚も設置できないなら`pass`を 1 増やし、さらにターンを更新する
    - `pass`が 2 を超えたとき、これ以上ゲームを進めることは不可能なので終了する

    - また、石の設置と同時に`pass`は 0 に戻される
- そして、ゲームの状態やターンの更新をプレイヤーに送る
- ゲームが終了すると結果情報をプレイヤーに送る

WebGL

- `WebGL Context`を取得する。（ブラウザが対応していない場合はエラーを表示）
- 3D モデルをロードする。(OBJ ファイルから頂点や法線の情報を取得し GPU にアップロード)
- ロードが終了すると描画ループに入る
- 発生した`DOMイベント`はキューに格納され、描画ループで処理される

7. 工夫したところとか

- `Go`と`Javascript`の両方で扱うのが簡単な`JSON`でデータをやり取りする
- サーバーはクリックの情報だけを受け取るのではプレイヤー不正が難しい
- ゲームプログラム (`./src/server/game/game.go`) は全て自分で考えた。
- マッチングには FIFO 線形リストを用いた。計算量の削減と、先に待機を始めたプレイヤーからゲームを開始するため。
- `Disc(disc.go)`は`int8`のエイリアスである。`-1`を掛けることで裏返しを表現している
- `Manager(manager.go)`ではハッシュテーブルを実装している