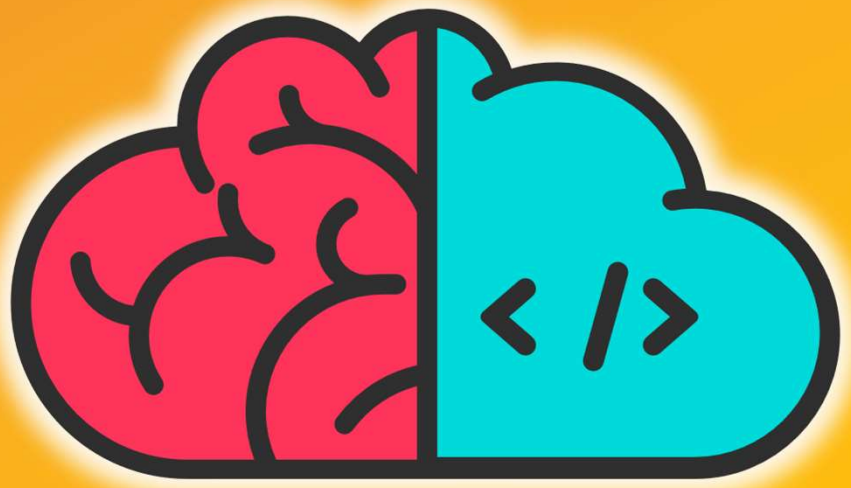
COURS C# / ASP.NET



Christophe MOMMER

https://www.hts-learning.com

PROJET

Pour apprendre ASP.NET, nous allons créer deux applications :

- Une API en ASP.NET Web API (Minimal API)
- Une application graphique (Blazor WebAssembly)

L'application sera le jeu de bataille navale

L'API sera chargée de gérer le jeu. Nous allons faire ensemble le développement de la partie solo contre l'ordinateur

Cet exercice servira de TP noté et sera fait en groupe de 2. L'évaluation tiendra compte de la qualité du code (ce que ChatGPT fait plutôt mal) et des fonctionnalités implémentées

Démonstration

ÉTAPE 1 : CRÉER LES PROJETS

Le projet se compose de 3 projets C# :

1. L'API
2. L'application Blazor
3. La librairie d'échange (les modèles)

Créer l'api avec la commande
« dotnet new webapi –minimal -n BattleShip.API »

Créer l'application Blazor WebAssembly
« dotnet new blazorwasm –e -n BattleShip.App »

Créer la librairie de modèles
« dotnet new classlib –n BattleShip.Models »

Créer la solution : « dotnet new sln –n BattleShip » et faire les liens (dotnet sln add [PROJET])

THÉORIE : LES APIS EN .NET

ASP.NET propose deux modes de fonctionnement des APIs : MVC et Minimal APIs (controlerless)

Cette deuxième approche est recommandée pour la performance et la simplicité

Les méthodes MapGet/MapPost (et pour tous les autres verbes HTTP) doivent être appelées sur la variable app :

C#

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

Les paramètres de la lambda peuvent être [FromBody] (POST), [FromServices] (injection de dépendances) ou encore [FromRoute] pour extraire une valeur de l'URL

THÉORIE : LES APIS EN .NET

La sérialisation des éléments se fait avec JSON. Il suffit de décrire un modèle (une classe) pour la lire, comme pour l'écrire :

```
app.MapGet("bonjour/{nom}/{prenom}",(
    [FromRoute] string nom,
    [FromRoute] string prenom) =>
    {
        return TypedResults.Ok(
            new Personne
            {
                Nom = nom,
                Prenom = prenom
            }
        );
    }
);
```

```
1 reference
class Personne
{
    1 reference
    public string Nom{ get; set; }
    1 reference
    public string Prenom { get; set; }
}
```

THÉORIE : LES APIS EN .NET

La désérialisation est automatique depuis le body d'une requête POST

```
app.MapPost("bonjour", (
    [FromBody] Personne personne) =>
    {
        return TypedResults.Ok(
            $"Bonjour {personne.Nom} {personne.Prenom}");
    }
);
```

THÉORIE : LES APIS EN .NET

L'injection de dépendances permet de configurer les services de l'application au niveau d'un point central pour que le container les fournisse à chaque élément le demandant (endpoint, classe, etc.)

Pour enregistrer un service, il faudra le faire sur builder.Services avec la méthode correspondant à sa durée de vie :

- Transient : éphémère
- Scoped : durée de vie limitée à un périmètre
- Singleton : instance unique partagée

```
builder.Services.AddSingleton<GameService>();
```

ÉTAPE 2 : CRÉATION DE GRILLE

Le jeu de bataille navale va être simplifié car c'est l'ordinateur qui va générer la position des bateaux (grille de 10x10)

On va utiliser un tableau à double dimension de char afin de déterminer les cases (char[,])

L'API doit se charger de générer deux grilles : une pour le joueur et une autre, gardée secrète, représentant le jeu de l'IA

Pour générer une grille :

- Un bateau aura une lettre attitrée (A => F) et une taille (1 => 4)
- Un bateau peut être placé en horizontal ou en vertical
- Il ne doit pas dépasser de la grille
- On obtient un aléatoire avec Random.Shared.Next(X) (X étant la valeur maximum - 1)
- La grille vide est composée uniquement de '\0', sinon elle contient la lettre du bateau

ÉTAPE 3 : API DE JEU

Préparons maintenant les endpoints afin de pouvoir exploiter notre jeu.
Au niveau de l'API, il sera nécessaire de proposer les opérations suivantes :

- Démarrer une nouvelle partie ➔ le retour de cet appel renverra les bateaux du joueur pour que l'application puisse l'afficher ainsi que l'identifiant de la partie qui a été créé

- Faire une attaque (passer l'identifiant du jeu ainsi que les coordonnées de l'attaque) ➔ le retour de cet appel renverra l'état du jeu (l'identité du gagnant si existant), l'état du tir (touché/raté) et l'attaque éventuelle de l'autre joueur (l'IA ici)

Pour que l'IA puisse répondre, il faudra prévoir une liste de coups possibles et les mélanger de façon aléatoire afin de les prendre dans l'ordre

ÉTAPE 4 : ALGORITHME DE JEU

Le joueur a tiré, il faut vérifier ce qu'il y a aux coordonnées indiquées. Si c'est '\0', cela signifie que la case est vide (raté). Si c'est une lettre, c'est qu'un bateau a été touché

Lorsque le tir a été effectué, vérifier si le jeu est terminé
Si oui, renvoyer comme quoi le jeu est terminé et que le joueur est gagnant
Si non, faire tirer l'IA et vérifier si le jeu est terminé. Si oui, l'IA est gagnante

Dans tous les cas, renvoyer toutes les informations à l'application graphique pour la mise à jour de l'interface

Pour vérifier si le jeu est terminé, la façon « rapide » est de parcourir les grilles et voir si elles contiennent encore une lettre de A à F

Il faut marquer les cases jouées dans la grille avec X (touché) ou O (raté). Il est possible de compter le nombre de X qui doit être égal à 13