

MASTER : M2 III

Durée : 4 heures

Enseignant : Guillaume Chervet

Nb page(s) : 12

- Le plagiat reste proscrit, une vérification pourra être effectuée.

TYPE DE BESOINS :			
<input checked="" type="checkbox"/> INTERCALAIRE SIMPLE	<input type="checkbox"/> INTERCALAIRE DOUBLE	<input type="checkbox"/> CALCULATRICE	<input type="checkbox"/> SANS CALCULATRICE
<input type="checkbox"/> COPIES DS	<input type="checkbox"/> SALLE INFO	<input checked="" type="checkbox"/> ORDI PERSO	<input type="checkbox"/> CLE USB
<input checked="" type="checkbox"/> AVEC DOCUMENTS (Internet)	<input type="checkbox"/> SANS DOCUMENTS	<input checked="" type="checkbox"/> SURVEILLANTS	<input checked="" type="checkbox"/> BROUILLONS
<input type="checkbox"/> PLAN COMPTABLE	<input type="checkbox"/> CODE (A PRECISER)	<input type="checkbox"/> DICTIONNAIRE (A PRECISER)	<input type="checkbox"/> AUTRES (A PRECISER)

Merci d'apporter vos précisions au niveau des consignes particulières svp.

CONSIGNES PARTICULIERES : Ce devoir nécessite votre ordinateur personnel ainsi qu'une connexion à internet et un accès à une prise électrique. Tous les supports sont autorisés. Chaque étudiant a un contenu d'examen et un rendu d'examen qui est différent.....

Examen Janvier 2024 – Session 1

Microsoft Technologies

N° Etudiant : .....

# Microsoft Technologies

Tous les supports sont autorisés (votre ordinateur, internet). Il n'y a aucun piège dans ce partielle, l'idée est de parcourir sous une autre forme ce qui a été réalisé en cours.

## 1. Compétences attendues

- Respecter organisation et nommage d'un projet C#
- Créer une librairie indépendante
- Gérer l'asynchronisme/parallélisme pour un calcul nécessitant du CPU
- Consommer une librairie tierce disponible depuis Nuget
- Tester unitairement une librairie/débugger
- Savoir créer, manipuler un projet solution .sln
- Savoir créer, manipuler un projet C# .csproj
- Gérer les références entre les projets de la même solution
- Savoir compiler un projet avec le dotnet cli
- Mettre en place un pipeline d'intégration continue avec Github Action
- Mettre en place une minimal Web Api
- Être Autonome

## 2. Objectifs

Vous allez créer une librairie qui va permettre de recevoir en entrée une image d'un objet à détecter, plus une ou plusieurs images dans lesquelles il faut rechercher l'objet en question. En sortie, vous retrouverez un tableau des coordonnées de l'objet détecté pour chaque image ainsi que l'image avec le contour de la détection dessinée.

La technique utilisée n'utilise pas d'Intelligence Artificielle, cela s'appelle le « Feature matching » cette technique était utilisée auparavant pour cartographier et rechercher les empreintes digitales.



Exemple de scène



Exemple d'objet à détecter



Exemple d'image de sortie

Pour cela vous allez utiliser la librairie tierce et open source OpenCVSharp : <https://github.com/shimat/opencvsharp>

Votre librairie permettra de détecter un objet dans plusieurs images à la fois sans bloquer le thread courant.

Les exécutions seront réalisées en parallèle et par conséquent la charge sera mieux répartie sur tous vos processeurs ce qui va accélérer le temps de traitement.

### 3. A Rendre avant la fin du temps imparti

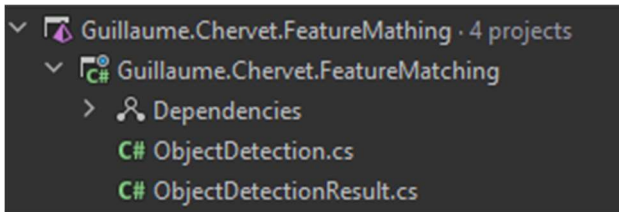
Votre projet « zippé » avec le format de nommage « **numéro-la-catho**.FeatureMatching.zip » dans la section « Icampus » de ce cours qui a été prévu pour déposer cet examen. Attention, il vous faut supprimer les répertoires "/bin" dans chaque projet sinon le « zip » sera trop gros. Il doit faire moins de 100 Mo.

### 4. Prérequis

Télécharger, installer et utiliser le sdk version SDK 8.0.100:

<https://dotnet.microsoft.com/en-us/download/dotnet/8.0>

## 5. Initiation de la solution (.sln) et création du projet librairie (3 points)



Vous allez créer une librairie qui réalise le FeatureMatching dont le « TargetFramework » est « net8.0 ». Son nom et namespace doivent respecter la convention `VotrePrénom.VotreNom.FeatureMatching` (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom)

### Fichier « ObjectDetectionResult.cs » Type de retour de votre librairie

```
namespace VotrePrénom.VotreNom.FeatureMatching;

public record ObjectDetectionResult
{
    public byte[] ImageData { get; set; }
    public IList<ObjectDetectionPoint> Points { get; set; }
}

public record ObjectDetectionPoint
{
    public double X { get; set; }
    public double Y { get; set; }
}
```

### Fichier « ObjectDetection.cs » classe qui expose la méthode publique de votre librairie

```
namespace VotrePrénom.VotreNom.FeatureMatching;

public class ObjectDetection
{
    public Task<IList<ObjectDetectionResult>> DetectObjectInScenesAsync(byte[]
objectImageData, IList<byte[]> imagesSceneData)
    {
        // TODO implement your code here
        throw new NotImplementedException();
    }
}
```

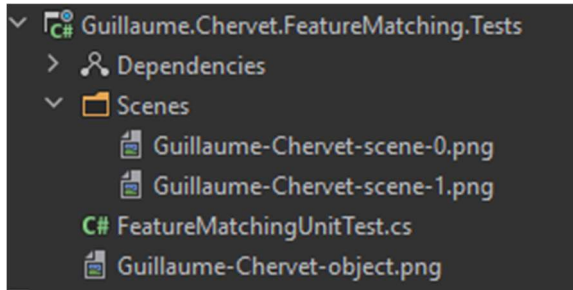
### A Réaliser :

1. Créer un projet Solution `VotrePrénom.VotreNom.FeatureMatching.sln` à la racine de votre projet (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom)
2. Créer le projet « .csproj » pour initier votre librairie de détection d'objet en respectant les consignes de nommage ci-dessus. A vous de bien respecter la hiérarchie des répertoires des projets .NET sur le disque dur.
3. Associer votre nouveau projet à la solution « .sln »
4. Créer les fichiers ObjectDetectionResult.cs et ObjectDetection.cs en positionnant le code dans l'énoncé ci-dessous. Comme vous pouvez le constater, pour le moment votre librairie retournera une Exception « NotImplementedException » via la méthode « DetectObjectInScenes ».

## 6. Création d'un projet de test unitaire (4 points)

Attention : Lisez et prenez le temps de lire toute la section avant de commencer à coder.

Le test unitaire ci-dessous (cf. code) consommera la librairie que vous venez de créer. Il utilise deux images « scène » qui seront associées au projet de test unitaire dans le sous répertoire « Scenes » et une image « objet » qui sera à la racine de votre projet, exemple :



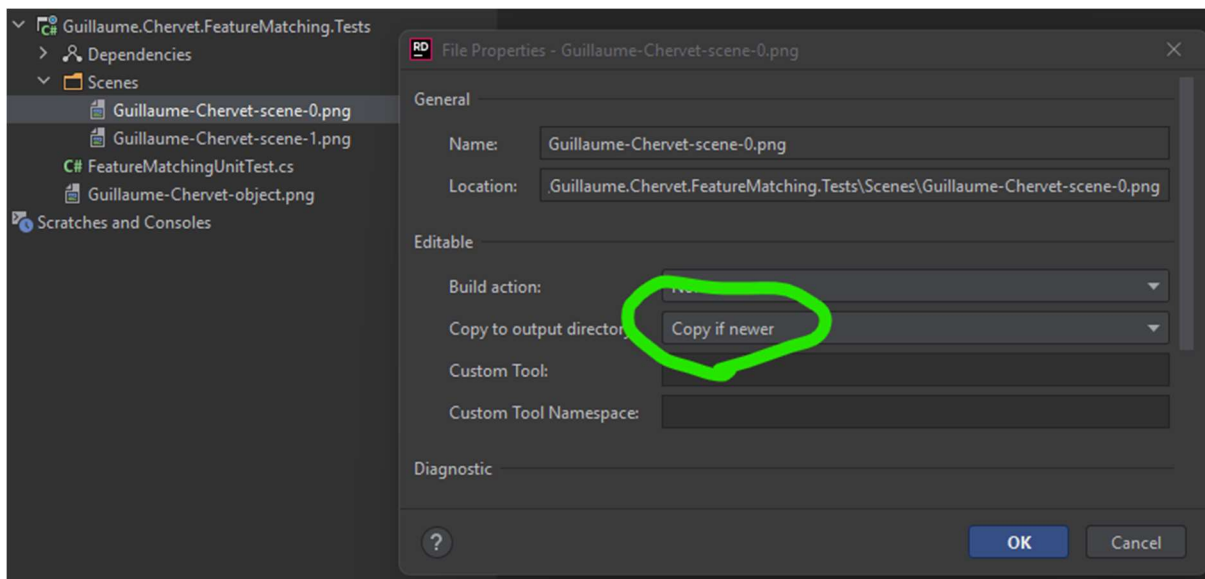
Vous avez trois images à télécharger via le lien one drive ci-dessous :

[https://1drv.ms/u/s!ApubmQnxll\\_okMsklMa9er-QalKnQ?e=eINyEl](https://1drv.ms/u/s!ApubmQnxll_okMsklMa9er-QalKnQ?e=eINyEl)

- VotrePrénom-VotreNom-scene-0.jpg => à mettre dans le répertoire « Scenes »
- VotrePrénom-VotreNom-scene-1.jpg => à mettre dans le répertoire « Scenes »
- VotrePrénom-VotreNom-object.jpg => à mettre à la racine de votre projet de Test Unitaire

(Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom)

Pour que ce test fonctionne, il faut bien configurer les images afin qu'elles soient automatiquement copiées dans le répertoire de sortie de « build » des librairies C#. Exemple :



Clic droit sur l'image puis sélectionner « Properties ». Puis dans le champ « Copy to output directory » sélectionner « Copy if newer ».

```

using System.Collections.Generic;
using System.IO;
using System.Reflection;
using System.Text.Json;
using System.Threading.Tasks;
using Xunit;

namespace VotrePrénom.VotreNom.FeatureMatching.Tests;

public class FeatureMatchingUnitTest
{
    [Fact]
    public async Task ObjectShouldBeDetectedCorrectly()
    {
        var executingPath = GetExecutingPath();
        var imageScenesData = new List<byte[]>();
        foreach (var imagePath in Directory.EnumerateFiles(Path.Combine(executingPath,
"Scenes")))
        {
            var imageBytes = await File.ReadAllBytesAsync(imagePath);
            imageScenesData.Add(imageBytes);
        }

        var objectImageData = await File.ReadAllBytesAsync(Path.Combine(executingPath,
"Guillaume-Chervet-object.png"));

        var detectObjectInScenesResults = await new
ObjectDetection().DetectObjectInScenesAsync(objectImageData, imageScenesData);

Assert.Equal("[{\"X\":116,\"Y\":158},{\"X\":87,\"Y\":272},{\"X\":263,\"Y\":294},{\"X\":
276,\"Y\":179}]", JsonSerializer.Serialize(detectObjectInScenesResults[0].Points));

Assert.Equal("[{\"X\":116,\"Y\":158},{\"X\":87,\"Y\":272},{\"X\":263,\"Y\":294},{\"X\":
276,\"Y\":179}]", JsonSerializer.Serialize(detectObjectInScenesResults[1].Points));
    }

    private static string GetExecutingPath()
    {
        var executingAssemblyPath = Assembly.GetExecutingAssembly().Location;
        var executingPath = Path.GetDirectoryName(executingAssemblyPath);
        return executingPath;
    }
}

```

#### A Réaliser :

- Créer un projet de test unitaire `VotrePrénom.VotreNom.FeatureMatching.Tests` qui utilise le framework de test « xunit » et dont le « TargetFramework » est « net8.0 ». A vous de bien respecter la hiérarchie des répertoires des projets .NET sur le disque dur.
- Associer votre projet de test au projet solution.
- Copier le test unitaire ci-dessus toujours en remplaçant « VotrePrénom » par votre prénom et « VotreNom » par votre nom.
- Depuis votre projet de test unitaire, vous devez référencer votre librairie de FeatureMatching.
- Télécharger, positionner, configurer les trois images afin qu'elles permettent au test de bien récupérer les images lors de l'exécution.

En l'état le test unitaire compile correctement, mais son exécution doit échouer (**rouge**) car votre librairie lance une Exception.

## 7. Implémenter votre librairie de FeatureMatching (4 points)

Attention : Lisez et prenez le temps de lire toute la section avant de commencer à coder.

Votre librairie devra référencer les packages nugets ci-dessous :

- **OpenCvSharp4.Extensions version 4.8.0.20230708**
- **Plus une librairie ci-dessous à adapter selon votre Operating System (OS) version 4.8.0.20230708**

Si vous êtes sur MAC votre librairie devra référencer les packages nugets ci-dessous (si cela ne fonctionne pas, mettez un code « mocké » qui retourne un faux résultat) :

- **OpenCvSharp4.Extensions version 4.6.0.20230105**
- **OpenCvSharp4.runtime.osx.10.15-x64 4.6.0.20230105**

Package	Description	Link
OpenCvSharp4.runtime.win	Native bindings for Windows x64/x86 (except UWP)	
OpenCvSharp4.runtime.uwp	Native bindings for UWP (Universal Windows Platform) x64/x86/ARM	
OpenCvSharp4.runtime.ubuntu.20.04-x64	Native bindings for Ubuntu 20.04 x64	
OpenCvSharp4.runtime.linux-arm	Native bindings for Linux Arm	
OpenCvSharp4.runtime.wasm	Native bindings for WebAssembly	

Exemple fonctionnel et synchrone d'utilisation de OpenCVSharp C# (lié à la configuration ci-dessus) :

```
// Import
using OpenCvSharp;
```

```
// Code
private ObjectDetectionResult DetectObjectInScene(byte[] imageObjectData, byte[]
imageSceneData)
{
    using var imgobject = Mat.FromImageData(imageObjectData, ImreadModes.Color);
    using var imgScene = Mat.FromImageData(imageSceneData, ImreadModes.Color);

    using var orb = ORB.Create(10000);
    using var descriptors1 = new Mat();
    using var descriptors2 = new Mat();
    orb.DetectAndCompute(imgobject, null, out var keyPoints1, descriptors1);
    orb.DetectAndCompute(imgScene, null, out var keyPoints2, descriptors2);

    using var bf = new BFMatcher(NormTypes.Hamming, crossCheck: true);
    var matches = bf.Match(descriptors1, descriptors2);

    var goodMatches = matches
        .OrderBy(x => x.Distance)
        .Take(10)
        .ToArray();
}
```



```

var srcPts = goodMatches.Select(m => keyPoints1[m.QueryIdx].Pt).Select(p => new
Point2d(p.X, p.Y));
var dstPts = goodMatches.Select(m => keyPoints2[m.TrainIdx].Pt).Select(p => new
Point2d(p.X, p.Y));

using var homography = Cv2.FindHomography(srcPts, dstPts, HomographyMethods.Ransac,
5, null);

int h = imgobject.Height, w = imgobject.Width;
var img2Bounds = new[]
{
    new Point2d(0, 0),
    new Point2d(0, h-1),
    new Point2d(w-1, h-1),
    new Point2d(w-1, 0),
};
var img2BoundsTransformed = Cv2.PerspectiveTransform(img2Bounds, homography);

using var view = imgScene.Clone();
var drawingPoints = img2BoundsTransformed.Select(p => (Point) p).ToArray();
Cv2.PolyLines(view, new []{drawingPoints}, true, Scalar.Red, 3);

// Uncomment to see the image result
/*
using (new Window("view", view))
{
    Cv2.WaitKey();
}
*/

var imageResult = view.ToBytes(".png");

return new ObjectDetectionResult()
{
    ImageData = imageResult,
    Points = drawingPoints.Select(point => new ObjectDetectionPoint() {X=point.X,
Y=point.Y}).ToList()
};
}

```

Votre librairie devra réaliser tous les traitements de détection en parallèle.

### A Réaliser :

1. Dans votre librairie de détection d'objet, importer les librairies « OpenCVSharp » depuis nuget. Ajouter la librairie adaptée à votre Operating System (OS). Si vous bloquez à faire fonctionner la librairie, passez à la suite en retournant des faux résultats en dur afin de pouvoir continuer l'examen.
2. Réaliser l'implémentation complète du code pour que plusieurs images de type « scènes » puissent être traitées en parallèle via l'utilisation de « Task.Run ».
3. Une fois votre librairie fonctionnelle, dans le test unitaire, remplacer les résultats attendus de type `"[{\"X\":117,\"Y\":158},{\"X\":87,\"Y\":272},{\"X\":263,\"Y\":294},{\"X\":276,\"Y\":179}]"` par vos propres valeurs afin que votre test unitaire soit passant (au vert).

### Exemple de code « mocké » qui retourne un faux résultat :

```
public async Task<IList<ObjectDetectionResult>> DetectObjectInScenesAsync(byte[]
objectImageData, IList<byte[]> imagesSceneData)
{
    IList<ObjectDetectionResult> results = new List<ObjectDetectionResult>();
    results.Add(new ObjectDetectionResult
        { ImageData = new byte[] {0},
          Points = new List<ObjectDetectionPoint> { new() {X=1,Y=2} } } );
    results.Add(new ObjectDetectionResult
        { ImageData = new byte[] {0},
          Points = new List<ObjectDetectionPoint> { new() {X=1,Y=2} } } );

    return results;
}
```

## 8. Création d'un projet console qui consomme votre librairie (3 points)

### A Réaliser :

1. Créer un projet « Console » qui consomme votre librairie.  
Son nom et namespace doivent respecter la convention **VotrePrénom.VotreNom.FeatureMatching.Console** (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom). A vous de bien respecter la hiérarchie des répertoires des projets .NET. A vous de bien associer ce projet à la solution .sln.

Le premier argument injecté à l'application doit être le chemin complet vers l'image de l'objet à détecter.  
Le second argument injecté doit être le chemin complet du répertoire qui contient les images de « scène » à la racine de celui-ci.

Exemple d'utilisation de votre projet console :

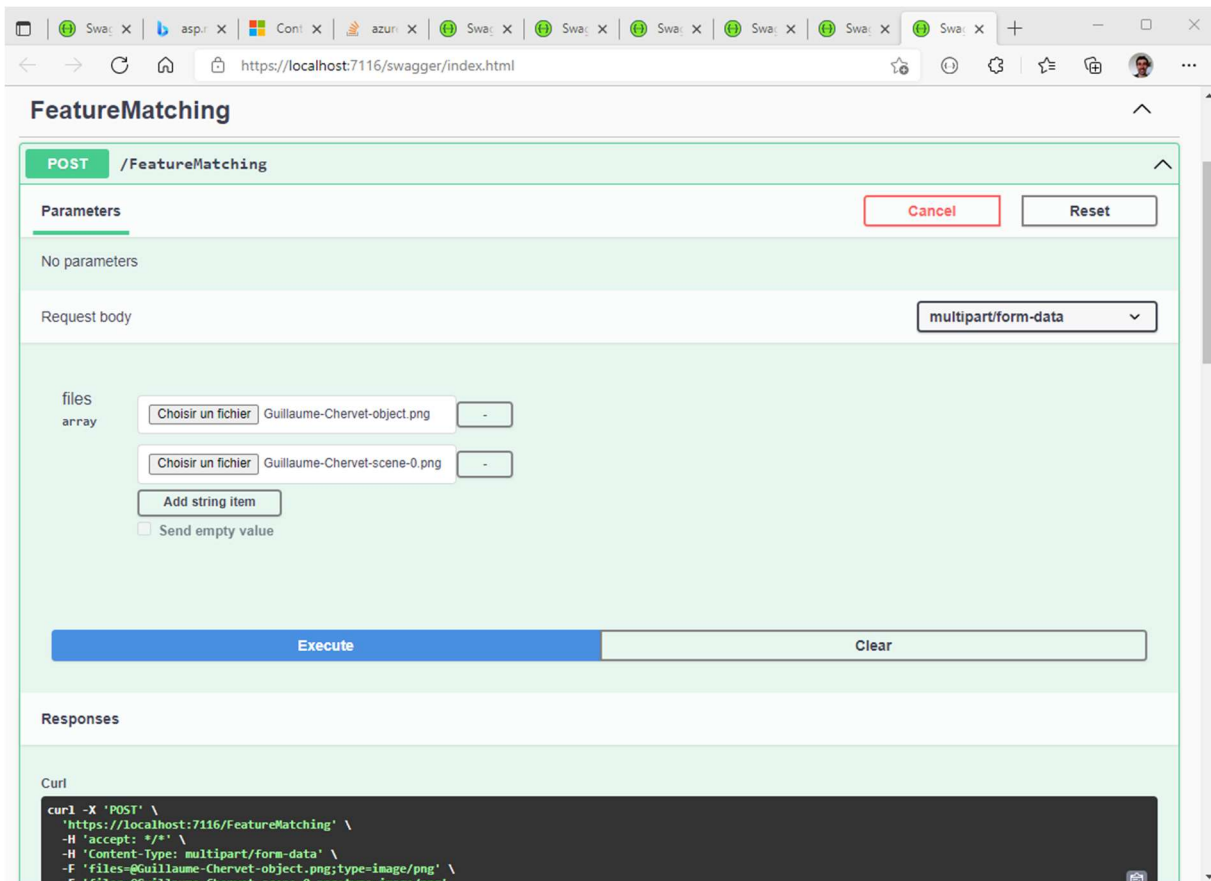
➤ *dotnet run C:\object.png C:\Scenes*

Le projet console doit imprimer dans la console toutes les listes de points détectés au format JSON pour toutes les images présentes dans le répertoire qui contient les « Scènes ».

```
foreach (var objectDetectionResult in detectObjectInScenesResults)
{
    System.Console.WriteLine($"Points:
{JsonSerializer.Serialize(objectDetectionResult.Points)}");
}
```

## 9. Exposition via une Api Web (3 points)

L'objectif de cette section est d'exposer votre librairie via une WebApi et de la rendre testable manuellement dans un browser via Swagger.



### A réaliser :

1. Créer un projet « WebApi » de type minimal API qui consomme votre librairie. Son nom et namespace doivent respecter la convention **VotrePrénom.VotreNom.FeatureMatching.WebApi** (Remplacer « VotrePrénom » par votre prénom et « VotreNom » par votre nom). A vous de bien respecter la hiérarchie des répertoires des projets .NET. A vous de bien associer ce projet au projet solution.

Si vous utilisez le bon « template », il y a déjà l'interface swagger configurée avec une route de démonstration de configuré.

2. Ajouter une route HTTP POST « FeatureMatching » qui contient le code ci-dessous puis compléter l'implémentation de la méthode afin qu'elle retourne l'image qui affiche la zone détectée et retournée par votre librairie. Ainsi dans le « Browser » vous pourrez voir visuellement la détection.

```

app.MapPost("/FeatureMatching", async ([FromForm] IFormFileCollection files) =>
{
    if (files.Count != 2)
        return Results.BadRequest();

    using var objectSourceStream = files[0].OpenReadStream();
    using var objectMemoryStream = new MemoryStream();
    objectSourceStream.CopyTo(objectMemoryStream);
    var imageObjectData = objectMemoryStream.ToArray();

    using var sceneSourceStream = files[1].OpenReadStream();
    using var sceneMemoryStream = new MemoryStream();
    sceneSourceStream.CopyTo(sceneMemoryStream);
    var imageSceneData = sceneMemoryStream.ToArray();

    // Your implementation code
    throw new NotImplementedException();

    // La méthode ci-dessous permet de retourner une image depuis un tableau de bytes,
    var imageData = new bytes[];
    return Results.File(result[0].ImageData, "image/png");
}).DisableAntiforgery();

```

-----

## 10. Intégration continue avec GitHubAction (3 points)

### A réaliser :

1. Ajouter un fichier « .gitignore » adapté à votre projet C#.
2. Open sourcer votre projet sur github en suivant la convention <https://github.com/your-account/VotrePrénom-VotreNom-FeatureMatching>
3. **Important :** Mettre à la racine de votre projet un README.md avec dedans un lien vers votre projet github. Cela permettra de retrouver votre github pour l'évaluation.
4. Ajouter un fichier « global.json » adapté à votre version du SDK.
5. Le but de cette question est de créer une build d'intégration continue en utilisant les GithubAction. Ajouter un fichier « ./.github/workflows/main.yml » dans votre projet. Configurer le fichier pour qu'il :
  - a. Télécharge la bonne version du SDK dotnet
  - b. Exécute le test unitaire (sans se soucier de la couverture de code)
  - c. Publie votre projet Console en sortie de votre build. Le projet publié devra être un .exe autonome compatible Windows 10.