# Teste d’intégration

## Partie 1

**Push sur le git**

git init

clear

git status

git add -A

git commit -m " start project"

clear

git status

git remote add origin https://github.com/Jackot/DarkTP.git

git push -u origin master

### Etape 1

On commence par vérifier que tout fonctionne :

Choisissez un répertoire et faites l'initialisation à la main.

Ouvrez VS Code, enregistrez en tant que Workspace le répertoire contenant la solition puis dans

la barre du haut de l'éditeur cliquez sur l'onglet Terminal puis choississez l'option New Terminal.

Un Terminal s'est ouvert en bas de la fenêtre de code, n'hésitez pas à l'agrandir un peu si

necessaire pour y voir.

Vous devriez voir un résultat de ce style :

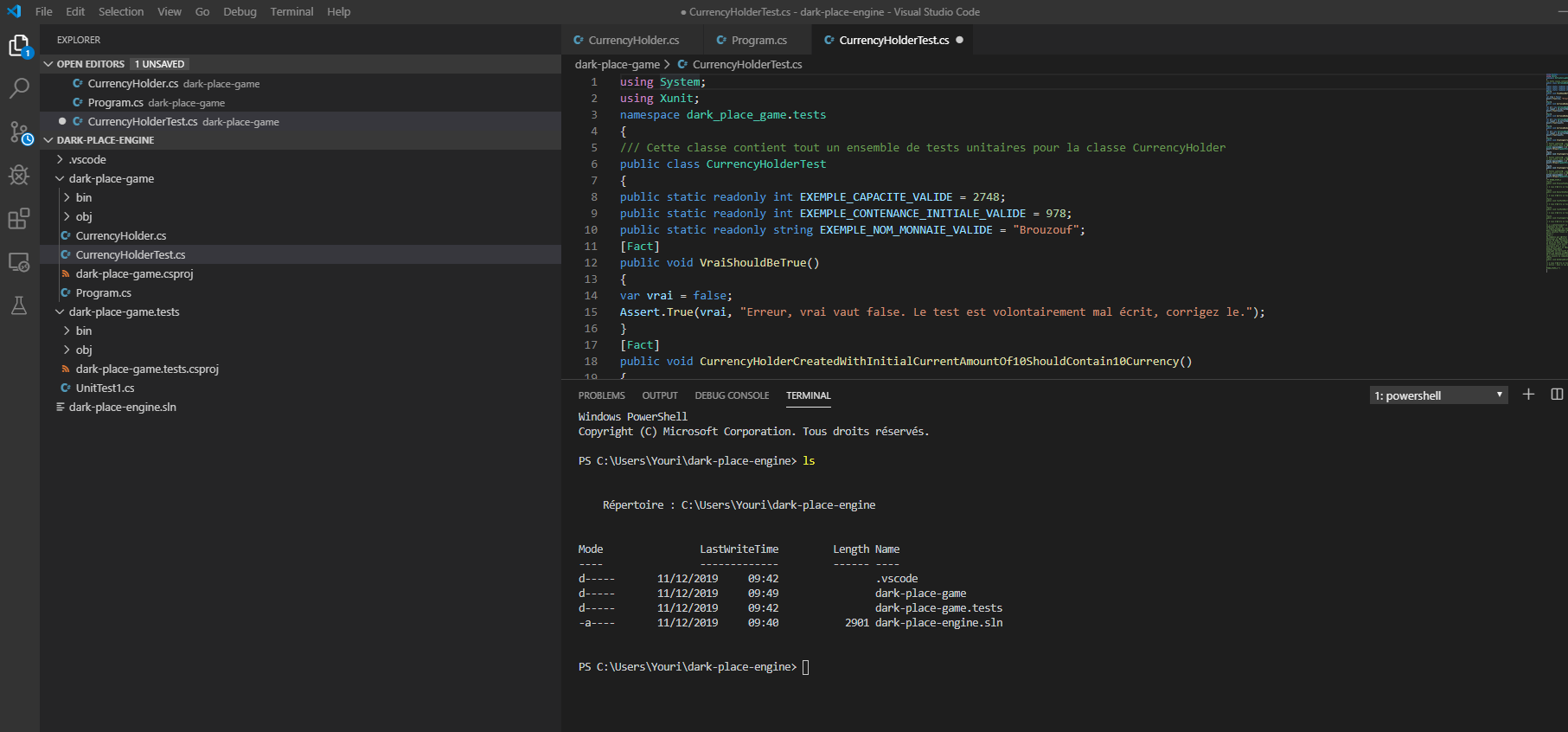
Windows PowerShell

Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS F:\IPI\_TP\test\dark-place-engine>

tapez ensuite la commande 'ls' pour vérifier que votre répertoire contient bien la solution :



Si vous ne voyez pas ce résultat vous n'êtes pas dans le bon répertoire ou avez mal récupéré le projet. Ensuite nous allons tester l'execution du programme et l'execution des tests. Tapez d'abord la commande "dotnet run --project .\dark-place-game" pour essayer d'executer le projet. Vous devriez voir quelque chose comme ca :

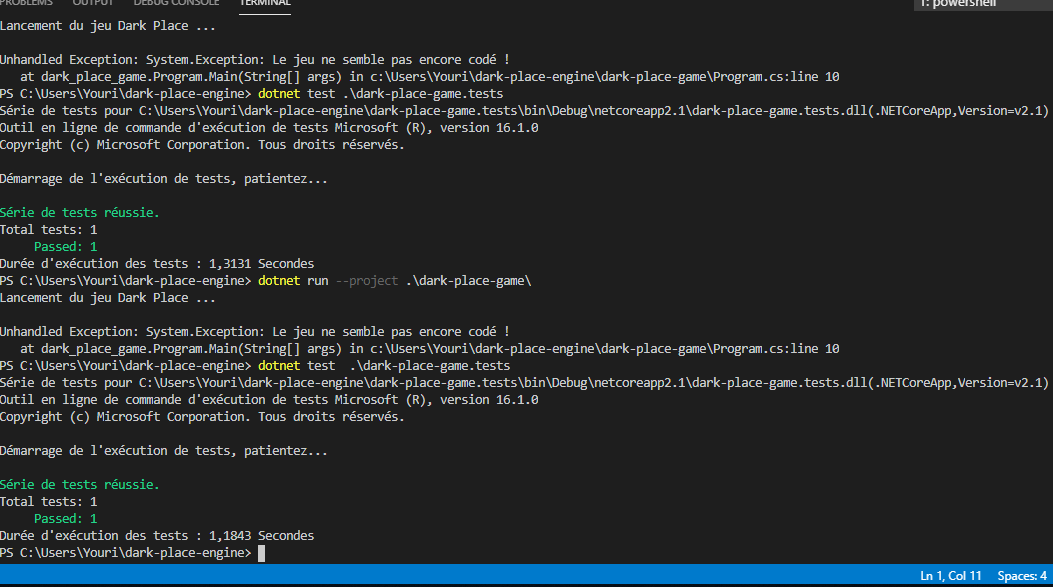
Ensuite executez les tests de l'application en tapant dans le terminal la commande "dotnet test .\dark-place-game.tests". Vous devriez voir un résultat de ce style :

PS F:\IPI\_TP\test\dark-place-engine> dotnet run --project .\dark-place-game\ Lancement du jeu Dark Place ...

Unhandled Exception: System.Exception: Le jeu ne semble pas encore codé ! at dark\_place\_game.Program.Main(String[] args) in F:\IPI\_TP\test\dark-place-engine\dark-place-game\Program.cs: PS F:\IPI\_TP\test\dark-place-engine>

Il est normal que la plupart des tests échouent à ce stade. Anecdotiquement certains tests réussissent par accident.

PS F:\IPI\_TP\test\dark-place-engine> dotnet test .\dark-place-game.tests\ Série de tests pour F:\IPI\_TP\test\dark-place-engine\dark-place-game.tests\bin\Debug\netcoreapp2.2\dark-place-gam Outil en ligne de commande d'exécution de tests Microsoft (R), version 16.3.0 Copyright (c) Microsoft Corporation. Tous droits réservés.



Démarrage de l'exécution de tests, patientez...

Au total, 1 fichiers de test ont correspondu au modèle spécifié.

[xUnit.net 00:00:00.42] dark\_place\_game.tests.CurrencyHolderTest.Test1 [FAIL]

[xUnit.net 00:00:00.43] dark\_place\_game.tests.CurrencyHolderTest.Test2 [FAIL]

[xUnit.net 00:00:00.43] dark\_place\_game.tests.CurrencyHolderTest.Test4 [FAIL]

X dark\_place\_game.tests.CurrencyHolderTest.Test1 [9ms] Message d'erreur : Un porte monnaie crée avec un contenu initial de 10 doit contenir 10 Expected: True Actual: False Arborescence des appels de procédure : at dark\_place\_game.tests.CurrencyHolderTest.Test1() in F:\IPI\_TP\test\dark-place-engine\dark-place-game.test

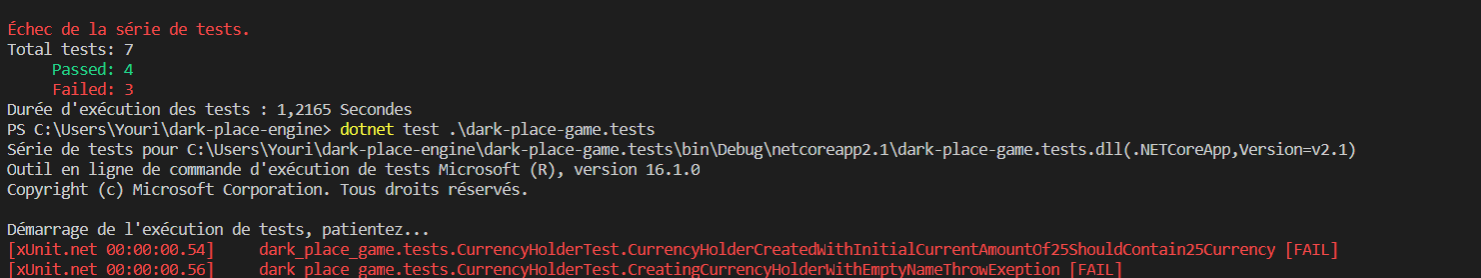
X dark\_place\_game.tests.CurrencyHolderTest.Test2 [1ms] Message d'erreur : Un porte monnaie crée avec un contenu initial de 25 doit contenir 25 Expected: True Actual: False Arborescence des appels de procédure : at dark\_place\_game.tests.CurrencyHolderTest.Test2() in F:\IPI\_TP\test\dark-place-engine\dark-place-game.test

X dark\_place\_game.tests.CurrencyHolderTest.Test4 [6ms] Message d'erreur : Assert.Throws() Failure Expected: typeof(System.ArgumentException) Actual: (No exception was thrown) Arborescence des appels de procédure : at dark\_place\_game.tests.CurrencyHolderTest.Test4() in F:\IPI\_TP\test\dark-place-engine\dark-place-game.test

Échec de la série de tests. Nombre total de tests : 4 Réussi(s) : 1 Non réussi(s) : 3 Durée totale : 0,9356 Secondes

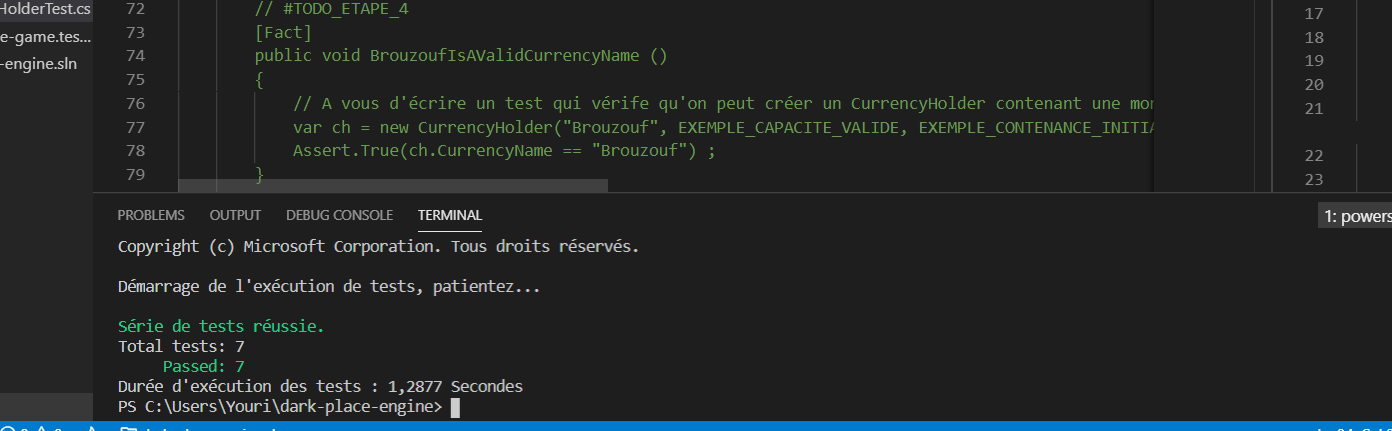
## Etape 2

Afin de vous préparer à la rédaction de vos propres tests plus tard, le test0 est volontairement mal codé. Corrigez-le. A part si le contraire est indiqué dans l'exercice, les tests que j'ai rédigés pour la suite du TP sont toujours valides. Lisez la trace de l'échec du test dans le terminal, trouver la ligne à modifier, et corrigez l'erreur évidente. Relancez le test,



## Etape 3

Vous allez pouvoir profiter des tests pour coder CurrencyHolder. Modifiez le code de CurrencyHolder jusqu'à ce que tous les tests passent.

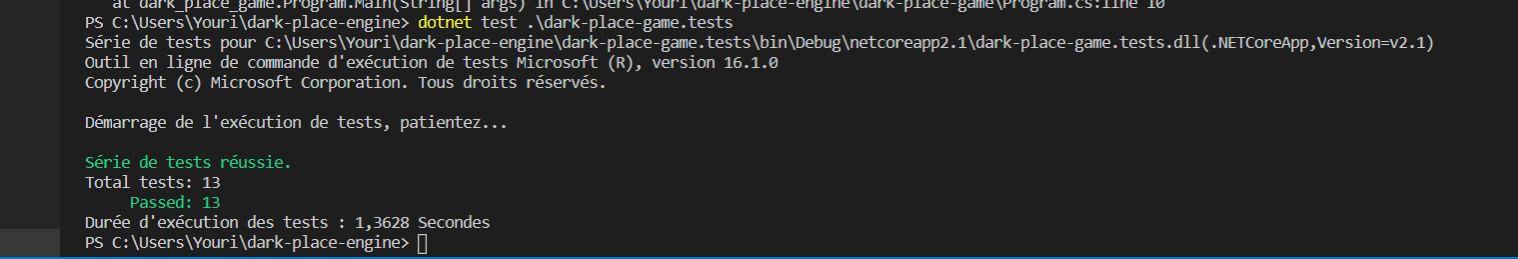


## Etape 4

Trouvez le bloc de code commenté avec le tag #TODO\_ETAPE\_4. Décommentez le Remplissez les tests vides. Exécutez les.

## Etape 5

Corrigez le code de CurrencyHolder jusqu'a faire passer vos tests. (en cas de doute sur vos tests demandez à vérifier qu'ils soient valides)



Etape 6

Voici quelques spécifications et ou/description de tests. Implantez ces tests dans le code.

* Un nom de currency doit faire entre 4 et 10 characteres : Ecrivez un test pour un nom de douze caracteres
* On ne peux pas mettre (méthode) put une quantité négative de currency dans un CurrencyHolder
* **On ne peux pas mettre des currency dans un CurrencyHolder si ça le fait depasser sa capacité**
* On ne peux pas ajouter ou retirer 0 currency (lever expetion) (2 tests)
* Un nom de currency ne doit pas commencer par la lettre a majuscule ou minuscule (2 tests) Un CurrencyHolder ne peux avoir une capacité inférieure à 1 (2 tests)
* Faire 2 tests unitaires pertinents pour la methode IsEmpty Un CurrencyHolder est plein (IsFull) si son contenu est égal à sa capacité (4 test, y a pas de piege)

Etape 7

Corrigez le code de CurrencyHolder jusqu'a faire passer vos tests. (en cas de doute sur vos tests demandez à vérifier qu'ils soient valides) Etape 8 Félicitation Vous avez terminé l'exercice sur les tests unitaires. Demandez l'accès à la partie 2.

Faites une copie archivée de votre travail.

using System;

namespace dark\_place\_game

{

    [System.Serializable]

    /// Une Exeption Custom

    public class NotEnoughtSpaceInCurrencyHolderExeption : System.Exception {}

    public class CurrencyHolder

    {

        public static readonly string CURRENCY\_DEFAULT\_NAME = "Unnamed";

        /// Le nom de la monnaie

        public string CurrencyName {

            get {return currencyName;}

            private set {

                currencyName = value;

            }

        }

        // Le champs interne de la property

        private string currencyName = CURRENCY\_DEFAULT\_NAME;

        /// Le montant actuel

        public int CurrentAmount {

            get {return currentAmount;}

            private set {

                currentAmount = value;

            }

        }

        // Le champs interne de la property

        private int currentAmount = 0;

        /// La contenance maximum du conteneur

        public int Capacity {

            get {return capacity;}

            private set {

                capacity = value;

            }

        }

        // Le champs interne de la property

        private int capacity = 0;

        public CurrencyHolder(string name,int capacity, int amount) {

            if (name == "") {

                throw new System.ArgumentException("Parameter name cannot be empty", "name") ;

            }

            if (name == null) {

                throw new System.ArgumentException("Parameter name cannot be null", "name") ;

            }

            if (amount < 0) {

                throw new System.ArgumentException("Parameter amount cannot be less than 0", "amount") ;

            }

            if (name.Length < 4 || name.Length > 11){

                throw new System.ArgumentException("Parameter name have not the good size", "name") ;

            }

            if (capacity < amount){

                throw new System.ArgumentException("Parameter name have not the good size", "name") ;

            }

            CurrencyName = name;

            Capacity = capacity;

            CurrentAmount = amount;

        }

        public bool IsEmpty() {

            if (CurrentAmount == 0)

                return true ;

            return false ;

        }

        public bool IsFull() {

            if ( CurrentAmount >= Capacity )

                return true ;

            return false ;

        }

        public void Store(int amount) {

            if(amount < 0 ){

                throw new System.ArgumentException("Parameter capacity must be positive ", "amount") ;

            }

            if (CurrentAmount+amount <= Capacity )

                CurrentAmount += amount ;

            else

                throw new NotEnoughtSpaceInCurrencyHolderExeption() ;

        }

        public void Withdraw(int amount) {

            if (CurrentAmount >= amount)

                Capacity -= amount ;

            else

                throw new System.ArgumentException("Impossible de retirer plus que le montant actuel.", "amount") ;

        }

    }

}

Teste

using System;

using Xunit;

namespace dark\_place\_game.tests

{

    /// Cette classe contient tout un ensemble de tests unitaires pour la classe CurrencyHolder

    public class CurrencyHolderTest

    {

        public static readonly int EXEMPLE\_CAPACITE\_VALIDE = 2748;

        public static readonly int EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE = 978;

        public static readonly string EXEMPLE\_NOM\_MONNAIE\_VALIDE = "Brouzouf";

        [Fact]

        public void VraiShouldBeTrue()

        {

            var vrai = true;

            Assert.True(vrai, "Erreur, vrai vaut false. Le test est volontairement mal écrit, corrigez le.");

        }

        [Fact]

        public void CurrencyHolderCreatedWithInitialCurrentAmountOf10ShouldContain10Currency()

        {

            var ch = new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE,EXEMPLE\_CAPACITE\_VALIDE , 10);

            var result = ch.CurrentAmount == 10;

            Assert.True(result);

        }

        [Fact]

        public void CurrencyHolderCreatedWithInitialCurrentAmountOf25ShouldContain25Currency()

        {

            var ch = new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE, EXEMPLE\_CAPACITE\_VALIDE, 25);

            var result = ch.CurrentAmount == 25;

            Assert.True(result);

        }

        [Fact]

        public void CurrencyHolderCreatedWithInitialCurrentAmountOf0ShouldContain0Currency()

        {

            var ch = new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE,EXEMPLE\_CAPACITE\_VALIDE, 0);

            var result = ch.CurrentAmount == 0;

            Assert.True(result);

        }

        [Fact]

        public void CreatingCurrencyHolderWithNegativeContentThrowExeption()

        {

            // Petite subtilité : pour tester les levées d'exeption en c# on est obligé d'utiliser un concept un peu exotique : les expression lambda.

            // sans entrer dans le détail pour déclarer une lambda respectez la syntaxe ci dessous, pour rédiger des tests unitaires elle devrais vous suffire.

            Action mauvaisAppel = () => new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE,EXEMPLE\_CAPACITE\_VALIDE , -10);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

        [Fact]

        public void CreatingCurrencyHolderWithNullNameThrowExeption()

        {

            // Petite subtilité : pour tester les levées d'exeption en c# on est obligé d'utiliser un concept un peu exotique : les expression lambda.

            // sans entrer dans le détail pour déclarer une lambda respectez la syntaxe ci dessous, pour rédiger des tests unitaires elle devrais vous suffire.

            Action mauvaisAppel = () => new CurrencyHolder(null,EXEMPLE\_CAPACITE\_VALIDE , EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

        [Fact]

        public void CreatingCurrencyHolderWithEmptyNameThrowExeption()

        {

            // Petite subtilité : pour tester les levées d'exeption en c# on est obligé d'utiliser un concept un peu exotique : les expression lambda.

            // sans entrer dans le détail pour déclarer une lambda respectez la syntaxe ci dessous, pour rédiger des tests unitaires elle devrais vous suffire.

            Action mauvaisAppel = () => new CurrencyHolder("",EXEMPLE\_CAPACITE\_VALIDE , EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

        // #TODO\_ETAPE\_4

        [Fact]

        public void BrouzoufIsAValidCurrencyName ()

        {

            // A vous d'écrire un test qui vérife qu'on peut créer un CurrencyHolder contenant une monnaie dont le nom est Brouzouf

            var ch = new CurrencyHolder("Brouzouf", EXEMPLE\_CAPACITE\_VALIDE, EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE) ;

            Assert.True(ch.CurrencyName == "Brouzouf") ;

        }

        [Fact]

        public void DollardIsAValidCurrencyName ()

        {

            // A vous d'écrire un test qui vérife qu'on peut créer un CurrencyHolder contenant une monnaie dont le nom est Dollard

            var ch = new CurrencyHolder("Dollard", EXEMPLE\_CAPACITE\_VALIDE, EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE) ;

            Assert.True(ch.CurrencyName == "Dollard") ;

        }

        [Fact]

        public void TestPut10CurrencyInNonFullCurrencyHolder()

        {

            // A vous d'écrire un test qui vérifie que si on ajoute via la methode put 10 currency à un sac a moité plein, il contient maintenant la bonne quantité de currency

            var ch = new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE, EXEMPLE\_CAPACITE\_VALIDE, EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE) ;

            ch.Store(10) ;

            Assert.True(ch.CurrentAmount == 10 + EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE) ;

        }

        [Fact]

        public void TestPut10CurrencyInNearlyFullCurrencyHolder()

        {

            // A vous d'écrire un test qui vérifie que si on ajoute via la methode put 10 currency à un sac quasiement plein, une exeption NotEnoughtSpaceInCurrencyHolderExeption est levée.

            Action mauvaisAppel = () => {

                var ch = new CurrencyHolder(EXEMPLE\_NOM\_MONNAIE\_VALIDE, EXEMPLE\_CAPACITE\_VALIDE, EXEMPLE\_CAPACITE\_VALIDE-9) ;

                ch.Store(10) ;

            };

            Assert.Throws<NotEnoughtSpaceInCurrencyHolderExeption>(mauvaisAppel) ;

        }

        [Fact]

        public void CreatingCurrencyHolderWithNameShorterThan4CharacterThrowExeption()

        {

            // A vous d'écrire un test qui doit échouer s'il es possible de créer un CurrencyHolder dont Le Nom De monnaie est inférieur a 4 lettres

             Action mauvaisAppel = () => new CurrencyHolder("fr",EXEMPLE\_CAPACITE\_VALIDE , EXEMPLE\_CONTENANCE\_INITIALE\_VALIDE);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

        [Fact]

        public void CreatingCurrencyHolderpositivecapacityThrowExeption()

        {

            // A vous d'écrire un test qui doit échouer s'il es possible de créer un CurrencyHolder dont la capacité est positive

             Action mauvaisAppel = () => new CurrencyHolder("uuuuu",EXEMPLE\_CAPACITE\_VALIDE , -10);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

       [Fact]

        public void limiteCurrencyHoldercapacityThrowExeption()

        {

            // tester la capaciter

             Action mauvaisAppel = () => new CurrencyHolder("uuuuu",10 , 10);

            Assert.Throws<ArgumentException>(mauvaisAppel);

        }

        [Fact]

        public void WithdrawMoreThanCurrentAmountInCurrencyHolderThrowExeption()

        {

            // A vous d'écrire un test qui vérifie que retirer (methode withdraw) une quantité negative de currency leve une exeption CantWithDrawNegativeCurrencyAmountExeption.

            // Asruce : dans ce cas prévu avant même de pouvoir compiler le test, vous allez être obligé de créer la classe CantWithDrawMoreThanCurrentAmountExeption (vous pouvez la mettre dans le meme fichier que CurrencyHolder)

        }

    }

}