# Suite de votre TP

**Après avoir réalisé votre « memory », nous allons à présent créer un programme de tests unitaire puis dans un second temps élaborer un espace projet. Par ailleurs, vous trouverez ci-joint la grille de notation(Annexe : grille de notation).**

## Créer un programme de test unitaire :

Pour utiliser ces tests avec Espresso et configurer votre environnement de test, voici les étapes à suivre :

### Étape 1 : Configurer votre projet avec Espresso

1. **Ouvrez build.gradle (Module: app)** : Assurez-vous d'ajouter les dépendances nécessaires pour Espresso dans votre fichier build.gradle (voir ci-dessous)

plugins **{** id 'com.android.application'  
**}**android **{** namespace 'com.game.memory'  
 compileSdk 33  
  
 defaultConfig **{** applicationId "com.game.memory"  
 minSdk 24  
 targetSdk 33  
 versionCode 1  
 versionName "1.0"  
  
 testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
 **}** buildTypes **{** release **{** minifyEnabled false  
 proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
 **}  
 }** compileOptions **{** sourceCompatibility JavaVersion.*VERSION\_1\_8* targetCompatibility JavaVersion.*VERSION\_1\_8* **}  
}**dependencies **{** implementation 'androidx.appcompat:appcompat:1.4.1'  
 implementation 'com.google.android.material:material:1.5.0'  
 implementation 'androidx.constraintlayout:constraintlayout:2.1.3'  
 testImplementation 'junit:junit:4.13.2'  
 androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
 androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
  
 // Dépendances de test pour Espresso  
 //androidTestImplementation 'androidx.test.ext:junit:1.1.5'  
 //androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'  
  
 // Dépendances pour éviter des conflits  
 implementation 'androidx.test:core:1.5.0'  
 androidTestImplementation 'androidx.test:rules:1.5.0'  
**}**

1. **Ouvrez build.gradle (Project : memory)** : Assurez-vous d’avoir la même chose (voir ci-dessous)

// Top-level build file where you can add configuration options common to all sub-projects/modules.  
plugins **{** id 'com.android.application' version '7.4.2' apply false  
 id 'com.android.library' version '7.4.2' apply false  
**}**

1. **Synchronisez le projet** après avoir modifié les dépendances.

### Étape 2 : Ajouter un fichier de test

1. Placez le fichier de tests généré dans le répertoire approprié :
   1. Naviguez dans l'explorateur de projet vers :  
      app/src/androidTest/java/com/game/memory/.
   2. Créez un nouveau fichier nommé MainActivityTest.java.
   3. Collez le code chapitre « programme de verificaiton ».
2. Vérifiez que le fichier a accès aux ressources de votre projet (boutons, TextView, etc.).

### Étape 3 : Exécuter les tests

1. **Sélectionnez la configuration de tests :**

* Cliquez avec le bouton droit sur le fichier MainActivityTest.java ou sur le dossier androidTest.
* Choisissez Run 'Tests in com.game.memory'.

1. **Visualisez les résultats des tests :**

* Une fenêtre s’ouvrira dans Android Studio montrant les résultats. Les tests réussis seront marqués en vert, tandis que les tests échoués seront en rouge.

### Étape 4 : Complétez votre fiche de test

Rajoutez des cas de tests permettant de fiabiliser votre solution logicielle

## Programme de verification:

package com.game.memory;  
  
import android.view.View;  
import android.widget.ImageButton;  
import android.widget.TextView;  
  
import androidx.test.espresso.ViewAction;  
import androidx.test.espresso.ViewAssertion;  
import androidx.test.ext.junit.rules.ActivityScenarioRule;  
import androidx.test.ext.junit.runners.AndroidJUnit4;  
  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
  
import static androidx.test.espresso.Espresso.*onView*;  
import static androidx.test.espresso.action.ViewActions.*click*;  
import static androidx.test.espresso.assertion.ViewAssertions.*matches*;  
import static androidx.test.espresso.matcher.ViewMatchers.*isDisplayed*;  
import static androidx.test.espresso.matcher.ViewMatchers.*withId*;  
import static androidx.test.espresso.matcher.ViewMatchers.*withText*;  
  
import java.lang.reflect.Field;  
  
@RunWith(AndroidJUnit4.class)  
public class MainActivityTest {  
  
 @Rule  
 public ActivityScenarioRule<MainActivity> activityRule = new ActivityScenarioRule<>(MainActivity.class);  
  
 @Test  
 public void testInitialSetup() {  
 // Vérifie que tous les boutons sont affichés  
 for (int i = 0; i < 20; i++) {  
 int buttonId = getButtonId(i);  
 *onView*(*withId*(buttonId)).check(*matches*(*isDisplayed*()));  
 }  
  
 // Vérifie que le TextView de message est affiché  
 //onView(withId(R.id.)).check(matches(isDisplayed()));  
 }  
  
 @Test  
 public void testClickButton() {  
 // Simule un clic sur le premier bouton  
 *onView*(*withId*(R.id.*Btn0*)).perform(*click*());  
  
 // Vérifie que l'image change après un clic  
 *onView*(*withId*(R.id.*Btn0*)).check(*matches*(*isDisplayed*())); // L'image doit être visible  
 }  
  
 @Test  
 public void testMatchCards() {  
 // Simule deux clics sur des cartes qui correspondent  
 *onView*(*withId*(R.id.*Btn0*)).perform(*click*());  
 *onView*(*withId*(R.id.*Btn1*)).perform(*click*());  
  
 // Vérifie que les cartes restent visibles et désactivées si elles correspondent  
 *onView*(*withId*(R.id.*Btn0*)).check(*matches*(*isDisplayed*()));  
 *onView*(*withId*(R.id.*Btn0*)).check((view, noViewFoundException) -> {  
 ImageButton button = (ImageButton) view;  
 assert !button.isEnabled(); // Bouton désactivé  
 });  
 }  
  
 @Test  
 public void testMismatchCards() {  
 // Simule deux clics sur des cartes qui ne correspondent pas  
 *onView*(*withId*(R.id.*Btn0*)).perform(*click*());  
 *onView*(*withId*(R.id.*Btn2*)).perform(*click*());  
  
 // Vérifie que les cartes retournent à l'état initial après une seconde  
 *onView*(*withId*(R.id.*Btn0*)).check(*matches*(*isDisplayed*()));  
 *onView*(*withId*(R.id.*Btn2*)).check(*matches*(*isDisplayed*()));  
 }  
  
 @Test  
 public void testWinningCondition() {  
 // Simule une série de clics pour trouver toutes les paires  
 int score = 0;  
 for (int i = 0; i < 10; i++) {  
 int firstButtonId = getButtonId(i);  
 int secondButtonId = getButtonId(i + 10);  
  
 *onView*(*withId*(firstButtonId)).perform(*click*());  
 *onView*(*withId*(secondButtonId)).perform(*click*());  
 score += 2;  
 }  
 assert score == 20 : "Le score final doit être 20, mais il est: " + score;  
 }  
 private int getButtonId(int index) {  
 try {  
 // Récupère l'ID du bouton dynamiquement à partir de son nom  
 String buttonId = "Btn" + index;  
 Field idField = R.id.class.getField(buttonId);  
 return idField.getInt(null);  
 } catch (Exception e) {  
 throw new RuntimeException("Erreur lors de la récupération de l'ID pour Btn" + index);  
 }  
 }  
}

## Elaborer un espace projet

Nous allons créer un compte sur une plateforme de projet puis déployer notre solution pour qu’elle soit accessible par GIT et créer une page d’accueil de votre projet.

### Exemple du wiki :

## Cours : Créer un espace projet avec Azure DevOps (ex Visual Studio Team Services)

### **1. Introduction à Azure DevOps**

**Azure DevOps** est un ensemble de services de collaboration et de gestion de projet destiné aux équipes de développement de logiciels. Il fournit des outils pour la gestion de code source, l'intégration continue, la gestion de projet agile, les tests automatisés, et bien plus encore. Un espace projet dans Azure DevOps permet à une équipe de collaborer efficacement, de suivre l'évolution du projet, et d'automatiser de nombreux aspects du développement logiciel.

**Objectifs de l'espace projet** :

* Gérer le code source avec Git ou Team Foundation Version Control (TFVC)
* Suivre l'avancement des tâches avec des tableaux Kanban, Scrum, ou Agile
* Automatiser les processus de build et de déploiement
* Organiser les tests et suivre les bugs

### **2. Création d'un compte Azure DevOps**

Avant de créer un espace projet, vous devez avoir un compte Azure DevOps. Si vous n’en avez pas encore, voici les étapes pour en créer un :

1. Allez sur [Azure DevOps](https://dev.azure.com/).
2. Cliquez sur **Start free** pour créer un compte.
3. Connectez-vous avec votre compte Microsoft ou créez-en un nouveau.
4. Une fois connecté, vous serez redirigé vers le tableau de bord d'Azure DevOps.

### **3. Créer un nouveau projet**

Un projet Azure DevOps est une collection de repos, de pipelines, de tableaux et d'artefacts associés. Pour créer un nouveau projet, voici les étapes :

1. Sur le tableau de bord d'Azure DevOps, cliquez sur **New Project** (Nouveau projet).
2. **Nommer le projet** : Choisissez un nom pertinent pour le projet.
3. **Description (optionnelle)** : Ajoutez une description qui explique brièvement l’objectif du projet.
4. **Visibilité** : Vous pouvez choisir entre un projet public ou privé, en fonction de vos besoins.
5. **Gestion du code** : Sélectionnez le type de gestion de code que vous préférez, Git étant le plus couramment utilisé.
6. **Agile, Scrum ou CMMI** : Sélectionnez un modèle de processus de gestion de projet en fonction de votre méthodologie (Agile, Scrum, ou CMMI).

Cliquez sur **Create** pour lancer la création du projet.

### **4. Configuration de l’espace projet**

Une fois le projet créé, vous avez accès à différentes sections pour gérer et configurer votre projet :

* **Repos** : Le dépôt de code source, où vous allez gérer votre code avec Git.
* **Pipelines** : La partie CI/CD pour automatiser la construction et le déploiement de votre code.
* **Boards** : Les tableaux Kanban, Scrum ou Agile pour suivre les tâches et l'avancement du projet.
* **Artifacts** : Pour gérer les paquets et artefacts de votre projet.

##### a) **Configurer un dépôt Git**

1. Allez dans **Repos**.
2. Si vous avez choisi Git lors de la création du projet, Azure DevOps va automatiquement créer un dépôt Git pour vous. Vous pouvez alors y pousser vos premiers commits en suivant les instructions données sur la page d’accueil du dépôt (URL de clonage, commandes Git).
3. Vous pouvez créer de nouvelles branches pour le développement, gérer les demandes de tirage (pull requests), et effectuer des revues de code.

##### b) **Configurer un pipeline (CI/CD)**

1. Allez dans **Pipelines** et cliquez sur **Create Pipeline**.
2. Sélectionnez le dépôt où se trouve votre code.
3. Choisissez un modèle pour le pipeline ou créez-le de manière personnalisée.
4. Configurez des étapes comme la compilation du code, les tests, et le déploiement automatique.
5. Une fois configuré, votre pipeline se déclenchera automatiquement chaque fois que vous ferez un commit dans le dépôt.

##### c) **Créer un tableau pour suivre les tâches (Boards)**

1. Allez dans **Boards** pour créer des éléments de travail comme des **User Stories**, des **Tâches**, des **Bugs** et des **Épics**.
2. Utilisez un tableau Kanban pour suivre l'état des tâches (À faire, En cours, Terminé).
3. Vous pouvez ajouter des **sprints** si vous utilisez Scrum ou planifier des itérations.

##### d) **Configurer des tests (Test Plans)**

Si vous voulez intégrer des tests dans votre processus, vous pouvez utiliser l'onglet **Test Plans** :

1. Créez des **plans de tests** pour définir les scénarios de tests à automatiser ou à exécuter manuellement.
2. Associez des tests aux tâches et assurez-vous que le code respecte les normes de qualité avant de le déployer.

#### **5. Ajouter des membres à votre équipe**

Pour collaborer efficacement, vous devez ajouter des membres à votre projet.

1. Allez dans **Project Settings** (Paramètres du projet).
2. Dans la section **Permissions**, cliquez sur **Add** pour ajouter des utilisateurs.
3. Assurez-vous que chaque membre a le rôle approprié (par exemple, Développeur, Administrateur, Testeur).

#### **6. Bonnes pratiques**

* **Gestion des branches** : Utilisez des branches pour chaque fonctionnalité ou bug, et ne travaillez jamais directement sur la branche principale.
* **Revue de code** : Utilisez des **Pull Requests** pour effectuer des revues de code avant de fusionner les branches dans la branche principale.
* **Communication** : Tenez des réunions régulières de mise à jour de l’avancement du projet, soit via un outil de chat intégré à Azure DevOps ou des outils externes comme Teams.
* **Automatisation** : Automatisez les tests et les builds pour garantir que les modifications du code sont validées avant d’être intégrées.

#### **7. Conclusion**

Vous avez maintenant créé un espace projet dans Azure DevOps, configuré des dépôts Git, mis en place des pipelines de CI/CD, et organisé les tâches avec des boards. Azure DevOps vous offre un ensemble d'outils puissants pour gérer tous les aspects du développement logiciel, du code source au déploiement. En suivant ces étapes, vous serez en mesure de travailler de manière plus collaborative, agile et automatisée, en améliorant la productivité de votre équipe.

### **Exercices pratiques :**

1. Créez un projet Azure DevOps pour intégrer votre application memory.
2. Créez un tableau Kanban pour suivre les tâches de développement de l'application.
3. Ajoutez au moins deux autres membres à votre projet et attribuez-leur des rôles spécifiques.

## Grille de notation

|  |  |
| --- | --- |
| Objectifs | **Note sur 2** |
| **Les 16 boutons sont affichés** |  |
| **Les boutons présentent un "?" ou une image** |  |
| **Quand je clique sur un bouton/image, l'image apparait** |  |
| **Quand une paire est identique, alors les cartes restent visibles et ne peuvent pas se retournées** |  |
| **Quand les cartes sont retournées, nous ne pouvons pas recliquer dessus** |  |
| **A chaque paire cliquée, un score est incrémenté** |  |
| **Un message est affiché en haut de l'écran** |  |
| **Le programme est installé sur la tablette ou sur le téléphone** |  |
| **Un programme de test est développé et permet de tester le programme** |  |
| **Le programme est disponible dans devops : dev.azure** |  |
|  |  |
| **Bonus (toutes nouvelles fonctionnalités/idées ajoutées)** |  |