S 1.01

DEV

# TP - Tests unitaires

## 1 Présentation générale

La SAÉ est pour vous l'occasion de réaliser pour la première fois un projet de programmation plus conséquent en équipe. A cette occasion, vous allez devoir mettre en place ce que l'on appelle des *tests unitaires*. Il s'agit de codes servant à vérifier le bon fonctionnement de chaque partie de votre programme (hors IHM). Cette pratique, assez repandue en entreprise, a de nombreux avantages comme par exemple d'assurer que seul un code fonctionnel est intégré au projet principal ou encore d'assurer la non-régression de votre code, c'est-à-dire d'éviter qu'une modification de votre code ne vienne supprimer une fonctionnalité codée précédemment.

L'objectif de ce TP est de vous apprendre à utiliser un petit framework maison de tests unitaires en Free-Pascal en vous faisant coder les tests unitaires pour un jeu de bataille navale dont le code vous est fourni afin de trouver et de corriger quelques erreurs insérées volontairement dans celui-ci.

Commencez par récupérer le projet Lazarus du jeu bataille navale présent sur le commun dans le répertoire "BatailleNavale" au même niveau que ce pdf. Ce projet est composé de plusieurs unités :

- **UnitBatailleNavaleLogic** contient la base du code métier du jeu (ce fichier ne contient aucune erreur).
- **UnitBatailleNavaleJoueurs** contient la partie du code métier du jeu que nous souhaitons tester.
- **UnitBatailleNavaleIHM** contient la partie graphique du jeu (ce fichier ne contient aucune erreur).
- **UnitBatailleNavaleTestUnitaire** contiendra les tests unitaires que vous allez coder.
- **TestUnitaire** est la bibliothèque de tests unitaires que nous allons utiliser, parcourez rapidement l'interface de cette unité pour voir les fonctions et procédures à votre disposition.

Vous pouvez tester le jeu en lançant le projet. Comme vous pouvez le voir, le jeu rencontre quelques soucis... Après ce test, commentez la ligne 6 du programme principale et décommentez la ligne 7.

## 2 Réalisation d'un premier test

Commencez par regarder la procédure **test** de l'unité **UnitBatailleNavaleTestUnitaire**. Cette méthode, appelée maintenant par le programme principal, va lancer successivement 7 autres procédures qui effectueront chacune des tests sur une partie du code du jeu puis exécute la procédure **Summary** qui affiche à l'écran un bilan des tests réalisés. Votre objectif est de coder les 7 procédures de tests. Pour cela, nous allons commencer par la procédure **initialisationNumeroJoueurEnCours_test** qui a pour but de tester la procédure **initialisationNumeroJoueurEnCours** présente dans l'unité **UnitBatailleNavaleJoueurs** et dont le rôle est d'initialiser le numéro du joueur en cours lors du début de partie. Les deux joueurs étant numéroté 0 et 1, il nous faut nous assurer qu'après cette initialisation, le numéro du joueur courant est bien 0.

Dans la procédure **initialisationNumeroJoueurEnCours_test** :

1. Créez une nouvelle série de tests en utilisant la commande :

```
newTestsSeries('Initialisation du numéro du joueur');
```

2. Ajoutez un nouveau test à cette série en utilisant la commande :

   newTest('Initialisation du numéro du joueur','A l"initialisation') ;

3. Exécutez la procédure initialisant le numéro du joueur :

   initialisationNumeroJoueurEnCours() ;

4. Testez si le numéro du joueur en cours est bien 0.

   testIsEqual(getNumJoueurEnCours(),0) ;

Exécutez votre projet et constatez que le test passe.

Cet exemple permet de voir la procédure à respecter pour créer un test à l'aide de notre framework :

1. On commence par créer une série de tests (qui pourra contenir plusieurs tests).
2. On crée ensuite un nouveau test dans cette série (vos tests doivent porter des noms différents).
3. On exécute le morceau de programme à tester.
4. On utilise l'une des procédures testIsEqual fournie par le framework pour tester si le résultat obtenu est bien celui attendu. (Une seule procédure testIsEqual peut être appelée par test !).

## 3 Tests du passage de tours

Dans la procédure **JoueurSuivant_test**, nous allons tester la procédure **JoueurSuivant**. Pour cela, créez une nouvelle série de tests nommée "Passage de tour". Créez ensuite deux tests :

- "Après premier tour" qui teste si après un premier passage de tour, c'est bien au joueur 1 de jouer.
- "Après second tour" qui teste si après un second passage de tour, c'est bien au joueur 0 de jouer.

Exécutez les tests. Vous constaterez que l'un d'eux ne passe pas. Corrigez le code de la procédure **JoueurSuivant** pour que le test passe.

## 4 Test de l'initialisation des joueurs

Dans la procédure **initialisationJoueur_test**, nous allons tester la procédure **initialisationJoueur**. L'objectif de cette procédure est d'initialiser le joueur (dont le numéro est donné en paramètre) en mettant son nombre de bateaux à 0 et en remplissant les deux cartes 10x10 (carteBateau et carteTir) de 0. Pour cela, créez une nouvelle série de tests nommée "Initialisation du joueur". Créez ensuite trois tests :

- "Nombre de bateau = 0 - 0" qui teste si, après initialisation du joueur 0, celui-ci ne possède aucun bateau.
- "Carte des bateaux vide - 0" qui teste si la carte des bateaux du joueur 0 ne contient bien que des 0.
- "Carte des tirs vide - 0" qui teste si la carte des tirs du joueur 0 ne contient bien que des 0.

Pour les deux tests sur les cartes, il vous est conseillé d'utiliser un boolean (estVide) initialisé à vrai, de parcourir le tableau que vous étudiez en mettant à faux le boolean si vous rencontrez une case contenant autre chose que 0. Enfin vous devrez utiliser la fonction testIsEqual(boolean) pour tester si le boolean est bien à vrai.
Réalisez aussi les tests pour le joueur 1 en n'oubliant pas de passer au joueur suivant.
Trouvez et corrigez les erreurs du code métier pour que les tests passent.

## 5 Test du placement des bateaux

Dans la procédure **placementDesBateauxJoueur_test**, nous allons tester la procédure **placementDesBateauxJoueur** dont l'objectif est de placer aléatoirement 5 bateaux (d'une case) dans la carte des bateaux du joueur dont le numéro est donné en paramètre. Ces bateaux sont représentés dans la carte des bateaux respectivement par des nombres de 1 à 5 (le premier bateau est représenté par un 1, le deuxième par un 2...). Pour cela, créez une nouvelle série de tests nommée "Placement des bateaux" dans laquelle, après avoir initialisé le joueur 0 et placez ces bateaux (en appelant les procédures respectives) vous créerez trois tests :

- "Nombre de bateaux - 0" qui vérifie que le joueur 0 possède bien 5 bateaux.
- "Nombre de bateaux sur la carte - 0" qui vérifie que 5 bateaux sont bien présents sur la carte des bateaux du joueur 0.

Faire de même pour le joueur 1.
Les tests devraient passer sans modification du code.

## 6 Test "gagné" et "perdu"

Dans les procédures **aPerdu_test** et **aGagne_test** on cherchera a tester les procédures **aPerdu** et **aGagne** qui déterminent respectivement si le joueur (dont le numéro est donné en paramètre) a perdu ou gagné. Dans chacune de ces deux procédures de test, créez une nouvelle série de tests contenant chacune deux tests :

- Le premier test vérifie que juste après l'initialisation des joueurs (les bateaux n'ont pas été placés), le joueur 0 a perdu (car il n'a plus de bateaux) et gagné (car l'autre joueur n'a plus de bateaux).
- Le second test vérifie qu'après avoir placé les bateaux le joueur 0 n'a ni perdu ni gagné.

## 7 Test des tirs

Dans la procédure **tirer_test**, proposez une série de tests permettant de test le mieux possible la procédure **tirer**. Pour indication, un 1 dans la table des tirs indique un tir n'ayant pas touché de bateau adverse et un 2 indique un tir ayant touché un bateau adverse. Corrigez les erreurs du code (si vous en détectez).