Bordeaux INP
ENSC

RAPPORT

Génie Logiciel – Projet 2016

Tests pour la sélection des astronautes de l'ESA

Émilie ROGER – Camille TARTARE
ENSC – Promotion 2018

Vendredi 16 décembre 2016

Plan

1. Introduction

L'objectif de ce projet est de réaliser un programme de tests similaire à celui qui est utilisé par l'agence spatiale européenne pour la sélection des astronautes.

Le programme de tests proposé dans ce projet comprend 5 tests différents : un test sur la perception et la mémoire associative, un test sur l'attention et la concentration, un test de calcul mental, un test sur des problèmes mathématiques et un test sur des problèmes physiques.

Pour chaque test, deux niveaux de difficultés sont proposés : un niveau facile, correspondant à une absence de limite de temps ou à des questions plus simples par exemple, et un niveau difficile.

À l'issue de chaque test, la proportion de bonnes réponses de l'utilisateur est donnée. Par ailleurs, tout au long du test, l'application indique à l'utilisateur si sa réponse est correcte ou non. En cas de mauvaise réponse, on précise à l'utilisateur la réponse qu'il aurait dû donner.

2. Spécifications générales

2.1. Analyse fonctionnelle

L'application s'articule autour de 5 tests différents, chacun ayant ses spécificités.

Nous avons décidé de mettre en place une première interface de bienvenue expliquant l'objectif de l'application. Si l'application ne l'intéresse pas, l'utilisateur peut alors la quitter. Sinon, il accède à une seconde interface qui lui permet de choisir quel test il veut faire et avec quel niveau de difficulté. Une fois son choix effectué, le test se charge dans une nouvelle fenêtre.

Le déroulement de chaque test est identique. Dans un premier temps, on affiche la consigne du test (ce qu'il faut que l'utilisateur sache et fasse pour réussir) illustrée au moyen d'une courte démonstration (3 à 6 images). Ensuite, l'utilisateur peut commencer le test. Une fois le test terminé, on lui affiche son pourcentage de réussite.

À chaque fois que l'utilisateur valide une réponse, une fenêtre pop-up (boîte de dialogue) lui indique si sa réponse est juste ou non, et si jamais sa réponse est incorrecte, on lui indique la réponse qu'il aurait dû renseigner.

L'interface de bienvenue et l'interface de menu sont toujours les mêmes quel que soit l'utilisateur et ne nécessitent pas de vérifications particulières. Pour la première interface, il n'y a que 2 boutons : un pour quitter l'application et un pour accéder au menu. Pour la seconde interface, un bouton permet de revenir à l'interface de bienvenue et les 10 autres ouvrent chacun un test différent.

En revanche, l'enchaînement des questions, une fois que l'utilisateur a choisi un test, est bien plus complexe car fortement dépendant du choix de l'utilisateur. Suivant s'il répond juste ou faux, les événements à déclencher ne vont pas être les mêmes. Par exemple, s'il répond juste, il faut ouvrir le pop-up de "bonne réponse" et incrémenter son score, tandis que s'il répond faux, il faut ouvrir le pop-up de "mauvaise réponse" et lui donner la réponse correcte. Par ailleurs, suivant le test choisi, la validation d'une réponse n'aura pas les mêmes conséquences.

L'interface montrant la consigne et la démonstration est sous forme de "ruban défilant". Lorsque l'utilisateur arrive sur l'interface, par défaut, elle lui affiche la consigne. Ensuite de part et d'autre de celle-ci, se trouvent des flèches qui permettent de faire défiler le "ruban", afin d'accéder à la démonstration. En bas, un bouton permet d'accéder directement au test.

N.B. : Le fonctionnement détaillé de chaque test est décrit plus loin dans la partie "Spécifications détaillées > Formulaires de l'application".

2.2. Analyse des besoins pour l'interface avec l'utilisateur

L'utilisateur doit pouvoir quitter l'application uniquement à partir de la page d'accueil. De plus, pour éviter qu'il ne quitte involontairement l'application, une fenêtre pop-up doit demander une confirmation à l'utilisateur.

L'utilisateur doit pouvoir revenir à l'interface de menu à tout moment et quel que soit le test qu'il est en train d'effectuer, cela grâce à un bouton dédié et à la "croix" fermant le formulaire.

L'utilisateur doit pouvoir connaître sa progression dans un test (à quelle question il est en train de répondre) et, si la réponse à la question est limitée dans le temps, connaître le nombre de secondes qui lui reste avant échéance.

Certains tests doivent afficher le pop-up de résultat durant 3 secondes. Nous avons décidé que l'utilisateur n'était pas obligé d'attendre 3 secondes pour passer à la question suivante, et pouvait donc fermer la boîte de dialogue à tout moment *via* un bouton "OK" et la croix rouge.

L'utilisateur ne doit pas pouvoir ouvrir 2 tests en même temps. Cela nécessite donc des formulaires modaux.

L'utilisateur ne peut pas agrandir l'interface comme il le souhaite, mais peut cependant réduire la fenêtre.

2.3. Schémas UML et autres schémas

2.3.1. Diagramme de classes

Nous avons décidé de mettre en place une classe abstraite "Test" définissant les méthodes et les attributs dont on aura besoin dans chaque test. Ces derniers ayant tous un

fonctionnement différent, cela justifie le fait qu'on ne puisse pas implémenter directement les méthodes dans "Test".

Les attributs de la classe abstraite sont les suivants : le nom du test, sa consigne, les "écrans" de démonstration, la liste des questions à poser au joueur, la liste des images à afficher au joueur (relatives à une question donnée), la liste des réponses justes, le score du joueur (initialement mis à zéro), l'intervalle de temps limite pour répondre à une question et le niveau de difficulté.

Les méthodes de la classe abstraite sont les suivantes :

- la récupération de la consigne du test et de la démonstration
- la récupération des questions du test, avec ses éventuelles images, ses réponses, etc.
- la vérification de la réponse donnée par le joueur
- en cas d'erreur, l'affichage du message d'erreur adéquate avec la réponse correcte
- le calcul de la proportion de réponses justes

Ensuite, nous avons 4 classes qui héritent de la classe abstraite et implémentent donc ses méthodes. Elles sont chacune dotées de 2 constructeurs : un pour créer un "test à jouer" et un destiné aux tests unitaires.

Certaines classes nécessitent, par ailleurs, des méthodes qui leur sont propres. De même chacune de ces classes "filles" possèdent des attributs inhérents au test qu'elles représentent, notamment le nombre de questions et leur identifiant.

Il a été décidé de rassembler les tests "Problèmes mathématiques" et "Problèmes physiques" dans une même classe, car ces 2 tests fonctionnent exactement de la même manière.

Ci-dessous le diagramme de classes associé à notre application :

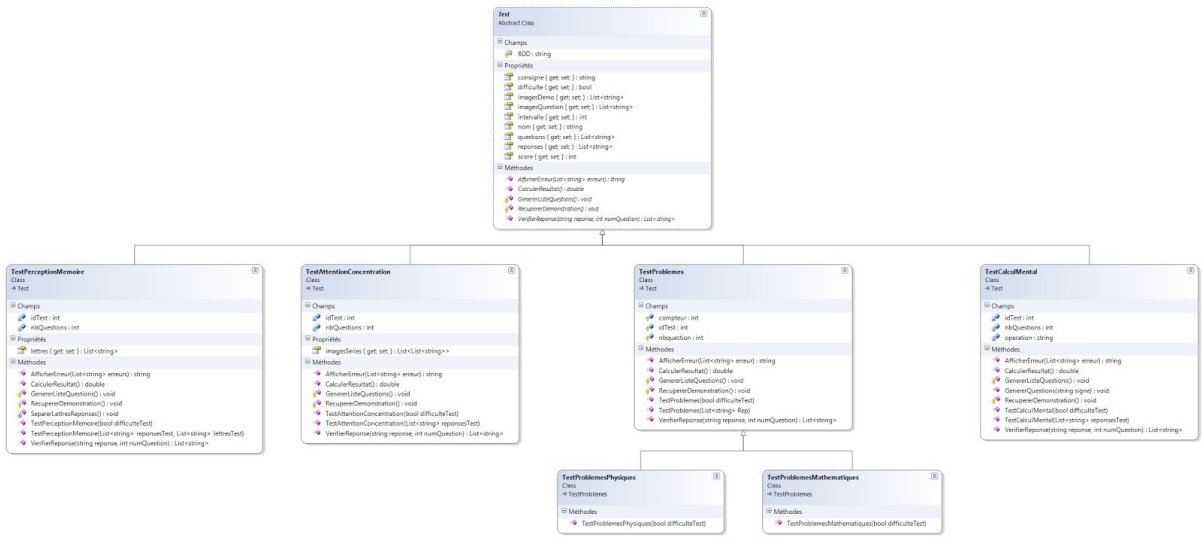


Diagramme de classes

2.3.2. Diagramme de séquences

Au niveau du fonctionnement des tests, on commence par construire un test grâce au constructeur.

Celui-ci appelle alors les méthodes “RecupererDemonstration” et “GenererListeQuestion” afin de compléter certains de ces attributs. Le constructeur peut également appeler des méthodes inhérentes à la classe, à l’instar de “SeparerImages” pour la classe “TestAttentionConcentration”.

Ensuite, *via* les formulaires qui vont bien, on affiche la consigne et la démonstration (formulaire “TestDemonstration”) puis on procède à l’enchaînement des questions (formulaires “TestPerception”, “TestAttention”, “TestCalcul”, “TestMaths” ou “TestPhysique”).

L’enchaînement des questions nécessitent l’appel à plusieurs méthodes de la classe. En effet, à chaque fois que l’utilisateur valide une réponse (ou pas si le temps de réponse est limité), on vérifie s’il a répondu juste avec la méthode “VerifierReponse”. Cette dernière renvoie alors une liste composée de la (des) réponse(s) fausse(s) et de la (des) réponse(s) correcte(s). Cette méthode permet également d’incrémenter le score du joueur.

La liste d’erreurs renvoyée par la méthode “VerifierReponse” permet alors l’affichage du résultat de l’utilisateur dans une boîte de dialogue *via* la méthode “AfficherErreurs” qui renvoie le message à afficher au joueur en cas de mauvaise réponse. En cas de bonne réponse, le message étant toujours le même pour n’importe quel utilisateur, nous avons jugé qu’il n’était pas nécessaire d’implémenter une méthode.

À la fin du test, on appelle la méthode “CalculerResultat” qui renvoie le taux de réussite de l’utilisateur pour ce test.

Voici ci-dessous le diagramme de séquences illustrant l’appel successif des méthodes de nos méthodes depuis les formulaires :

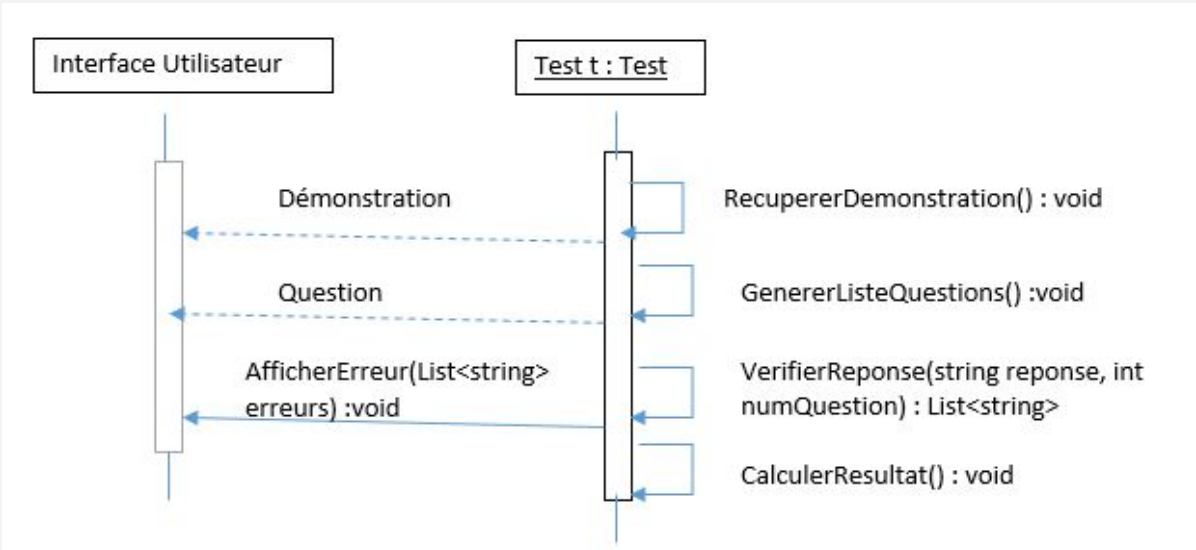
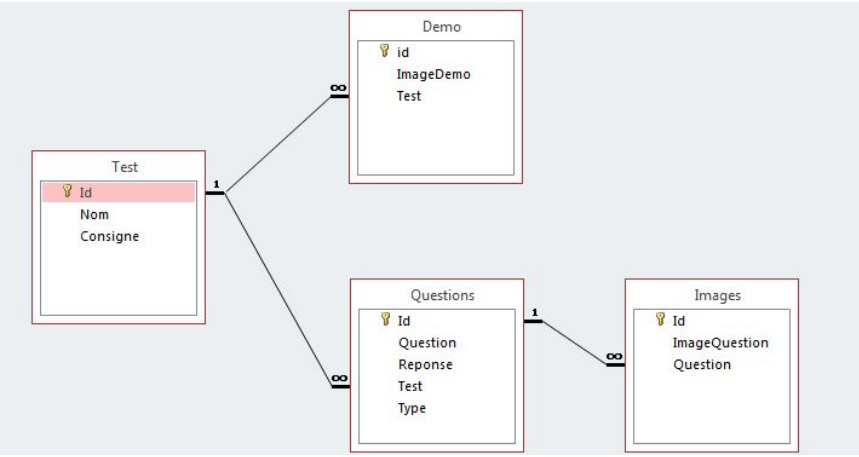


Diagramme de séquences

2.3.3. Modèle conceptuel des données

Voici comment nous avons construit la base de données qui répertorie les tests disponibles (avec leur consigne et leurs “écrans de démonstration”) et l’ensemble des questions utilisées par ces tests (avec leurs réponses et leurs éventuelles images).



Modèle conceptuel des données

2.3.4. Interactions entre l’utilisateur, les formulaires et les classes

Voici les interactions qu’il y a – généralement – entre l’utilisateur, les différents formulaires et les classes. Par rapport à la réalité, le formulaire de résultat n’est pas apparent et l’appel aux fonctions “VerifierReponse” et “AfficherErreur” n’est représenté qu’une seule fois alors qu’elles sont appelées autant de fois qu’il y a de questions dans le test.

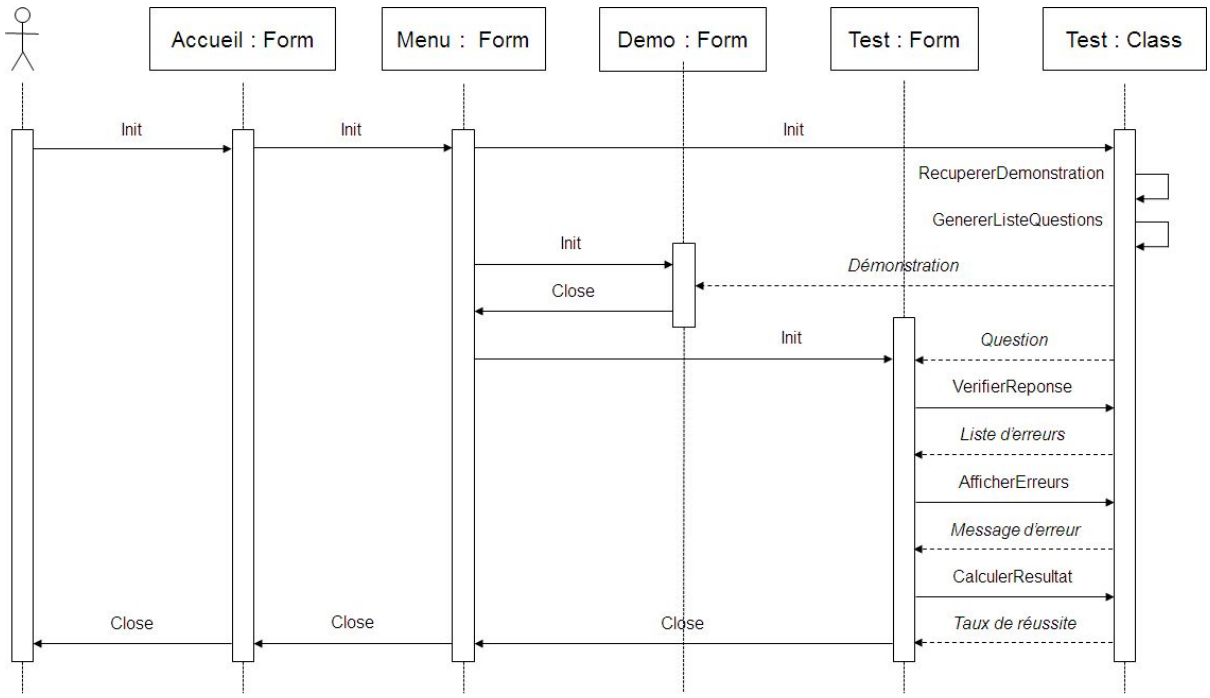


Schéma des interactions

2.4. Description de l'architecture de l'application

L'application doit permettre d'effectuer des tests similaires à ceux utilisés pour sélectionner les astronautes de l'agence spatiale européenne (ESA).

L'application est codée en C# sous Miscrosoft Visual Studio 2010 et la base de données utilisée par l'application est une base de données Access.

L'application est structurée comme suit :

- une Bibliothèque de classes "EMACClass" permettant de coder l'ensemble des classes nécessaires au projet, à savoir notre classe abstraite "Test", et ses 4 classes héritées
- une Application Windows Form "EMACApp" permettant de coder les différentes interfaces nécessaires au projet, à savoir l'interface de bienvenue "Home", l'interface de menu "Menu", l'interface affichant consigne et démonstration "TestDemonstration", les interfaces inhérentes à chaque test ainsi que l'interface "TestResultat" permettant de générer la boîte de dialogue qui indique si la réponse de l'utilisateur est juste
- un Projet de test "EMACTest" contenant l'ensemble de nos tests unitaires

"EMACApp" contient également la base de données utilisée pour faire fonctionner l'application (dans AppData), et les images nécessaires pour illustrer les démonstrations et certaines questions (dans AppImages).

Il est nécessaire d'ajouter la référence "EMACClass" à l'Application Windows Form "EMACApp" afin de pouvoir générer les tests et faire tourner l'application. De même, il faut ajouter "EMACClass" au Projet de test "EMACTest" pour pouvoir exécuter les tests unitaires.

Un dossier "EMACPicture" a également été créé. Celui-ci contient les images utilisées pour le design de nos formulaires (fond, logos, etc.). Il n'interagit absolument pas avec le reste de l'application et n'est destiné qu'à stocker nos images afin de ne pas les perdre.

2.5. Répartition des tâches et planning

2.5.1. Planning prévisionnel

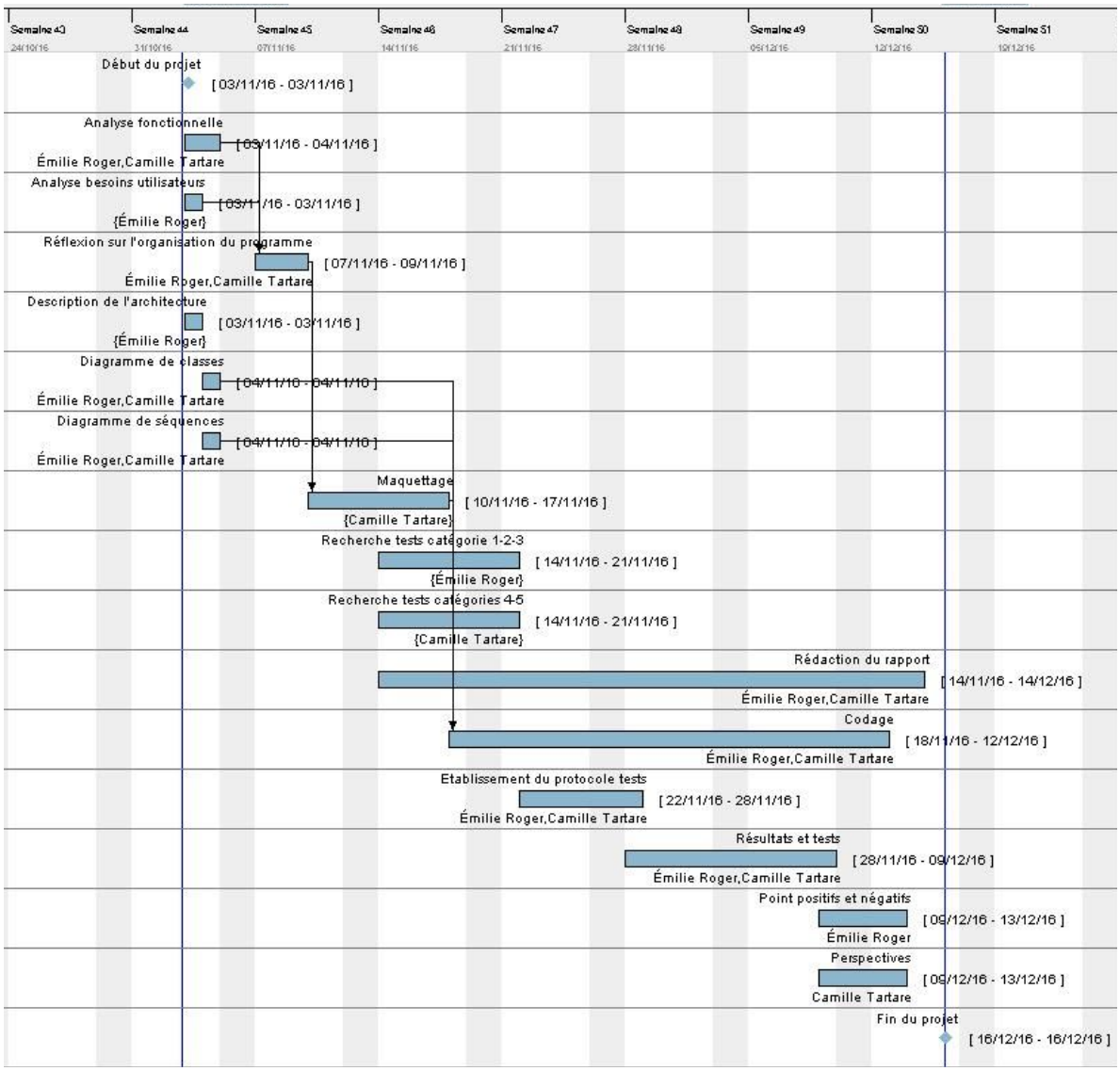
Le planning ci-dessous présente les différentes tâches que nous avions prévues de faire, quand, par qui et en combien de temps.

Ce planning est non-exhaustif, la majorité des tâches inscrites étant bien souvent divisées en sous-tâches, voire sous-sous-tâches, notamment pour les tâches "Rédaction du rapport" et "Codage".

Notre démarche a été, globalement, la suivante :

- une première période de réflexion (analyse fonctionnelle et des besoins, architecture logiciel, schémas UML, formulaires et classes à coder)

- une deuxième période pour effectuer le travail préliminaire (logo, nom de l'application, charte graphique, création des questions, maquettage)
- une troisième période de codage et de rédaction du rapport en parallèle, cette dernière période englobant l'écriture des protocoles de tests et la mise en œuvre de ces tests
- une dernière période de relecture et de corrections



Planning prévisionnel du projet et répartition des tâches

2.5.2. Planning effectif

Dans l'ensemble, nous avons respecté le planning établi. Ainsi, par exemple, nous avons commencé le rapport dès le début du projet afin de ne pas le bâcler en fin de projet, ce qui était planifié. Nous avons cependant allongé légèrement la phase de codage afin d'apporter des optimisations finales auxquelles nous n'avions pas songé en commençant le projet.

3. Spécifications détaillées

3.1. Formulaires de l'application

Nous avons 9 formulaires différents :

- un pour l'interface de bienvenue
- un pour l'interface de menu
- un pour l'interface affichant la consigne et la démonstration du test choisi par l'utilisateur
- un pour chaque test
- un pour les boîtes de dialogue qu'on n'affiche sur une durée limitée

3.1.1. Formulaire de bienvenue "Home"

Très peu d'interactions sont possibles dans ce formulaire.

La croix rouge et le bouton "Quitter" déclenchent la fermeture du formulaire. Une demande de confirmation *via* une fenêtre pop-up est demandée.

Le bouton "Continuer" permet d'accéder au formulaire de menu. Lorsqu'on appuie sur ce bouton, le formulaire actuel est caché et le formulaire de menu est généré de façon modale : le formulaire de menu est comparable à une boîte de dialogue.

3.1.2. Formulaire de menu "Menu"

La croix rouge et le bouton "Retour" permettent de revenir au formulaire de bienvenue, le résultat du formulaire (*i.e.* de la boîte de dialogue) prenant la valeur "OK".

L'ensemble des boutons restant déclenchent la création d'un test (appel du constructeur adéquate avec l'argument "true" si l'utilisateur a cliqué sur un bouton portant la mention "Difficile" ou "false" si l'utilisateur a cliqué sur un bouton portant la mention "Facile) et l'affichage, c'est-à-dire l'ouverture, du formulaire de consigne et démonstration.

3.1.3. Formulaire de consigne et démonstration "TestDemonstration"

La création de ce formulaire nécessite un argument de type "Test" ; il s'agit du test créé dans le formulaire précédent. Cela permet de récupérer la consigne et les "écrans" de démonstration appropriés, chaque test ayant une consigne et des "écrans" de démonstration différents.

Bien que les consignes et les "écrans" de démonstration soient inhérents à un type de test, nous avons pris le parti de faire un seul formulaire pour afficher les consignes et les "écrans" de démonstration de tous les tests. Cela s'explique par le fait que le fonctionnement du formulaire est exactement identique quel que soit le test choisi par l'utilisateur.

Le fonctionnement de ce formulaire est le suivant :

- La croix rouge déclenche la fermeture du formulaire.
- Les chevrons “>” et “<” permettent le défilement de la consigne et des “écrans” de démonstration.
- Le bouton “Jouer” permet de fermer le formulaire actuel et d’accéder au formulaire propre au test choisi par l’utilisateur. Par exemple, si l’utilisateur avait choisi le test “Calcul mental” dans le formulaire de menu, en cliquant sur le bouton “Jouer”, cela fermera le formulaire de consigne et démonstration et ouvrira le formulaire de calcul mental (“TestCalcul”).

L’ouverture du formulaire de consigne et démonstration n’entraîne pas la fermeture du formulaire de menu. Ainsi, l’utilisateur peut à tout moment revenir au menu.

3.1.4. Formulaire de perception et mémoire associative “TestPerception”

Écran 1 : Tout d’abord, un premier écran permet d’afficher la question en cours (*e.g.* “Question 4 sur 10”), la règle à suivre pour cette question (*e.g.* “Retenez les chiffres dans les carrés bleus”) et un bouton “Suivant”.

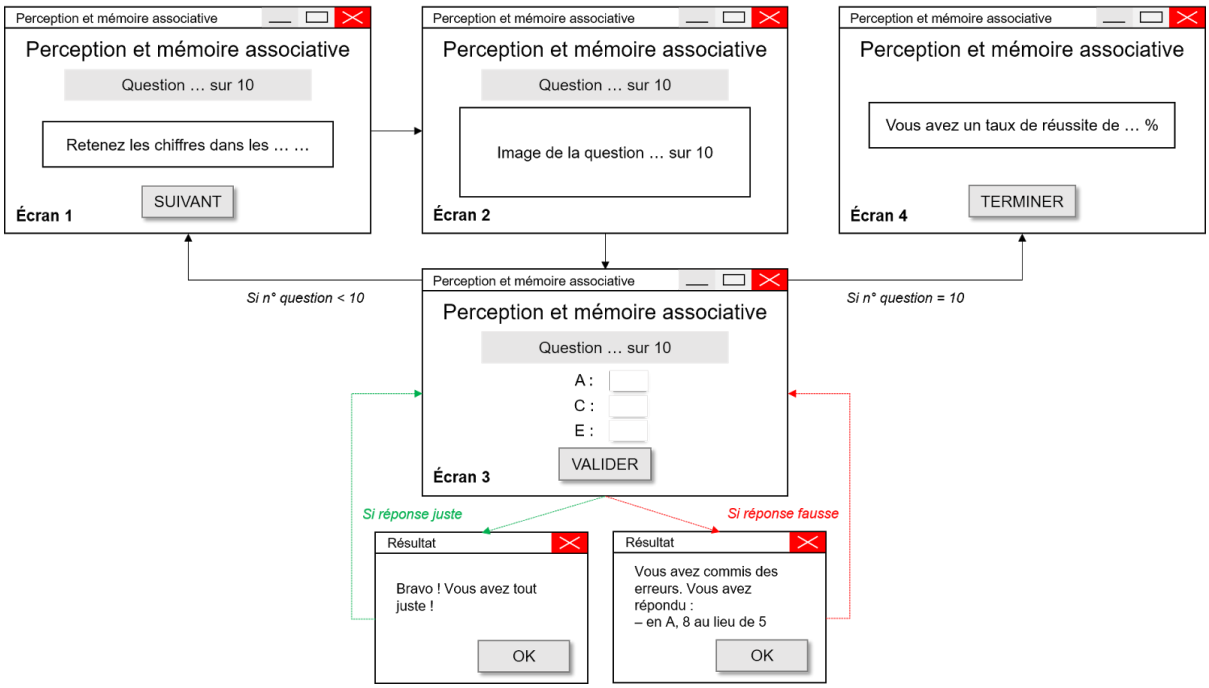
Écran 2 : Lorsque l’utilisateur a lu la règle et a appuyé sur le bouton “Suivant”, la règle et le bouton “Suivant” sont cachés, et à leur place s’affiche l’image relative à la question. Cet affichage dure 4 secondes en mode “Facile” et 2 secondes en mode “Difficile”.

Écran 3 : Ensuite, l’image est cachée et est remplacée par un formulaire avec des champs à compléter et un bouton de validation “Valider”. Cela permet à l’utilisateur d’entrer les chiffres qu’il a retenu et de valider sa réponse.

> Quand l’utilisateur clique sur le bouton “Valider”, une boîte de dialogue s’ouvre avec la mention “Bravo ! Vous avez tout juste !” s’il a fait un sans-faute ou la mention “Vous avez commis des erreurs. Vous avez répondu : [– en …, … au lieu de …]” s’il a mal rempli certains champs, voire rien rempli du tout.
>
> Il faut alors que l’utilisateur clique sur le bouton “OK” de la boîte de dialogue pour qu’elle se ferme. On efface alors l’écran en cours et on réaffiche l’écran **Écran 1** dans le cas où on n’en était pas à la dernière question (10 questions).

Écran 4 : Dans le cas où l’utilisateur était en train de répondre à la dernière question, on efface l’écran en cours (**Écran 3**) et on affiche à la place son résultat (proportion de réponses justes) avec un bouton “Terminer”. Si l’utilisateur appuie sur le bouton “Terminer”, le formulaire se ferme.

Ci-dessous l’illustration du fonctionnement du test de perception et mémoire associative :



3.1.5. Formulaire d'attention et concentration "TestAttention"

Écran 1 : Tout d'abord, un premier écran permet d'afficher la série et la question en cours (*e.g.* "Série 1 sur 3 – Question 2 sur 5"), la règle à suivre pour cette série (même règle pour les 3 séries en mode "Facile", règles différentes pour les 3 séries en mode "Difficile") et un bouton "Suivant".

Écran 2 : Lorsque l'utilisateur a lu la règle et a appuyé sur le bouton "Suivant", la règle et le bouton "Suivant" sont cachés, et à leur place s'affiche l'image relative à la question 1 de la série, ainsi que le bouton "Bouton 3". Cet affichage dure tant que l'utilisateur n'a pas appuyé sur le bouton. En mode "Difficile", cet affichage n'excède pas 5 secondes.

> Quand l'utilisateur clique sur le bouton "Bouton 3", on efface l'écran en cours et une boîte de dialogue s'ouvre avec la mention "Bravo ! Vous avez cliqué sur le bon bouton !". En mode "Difficile", s'il n'a pas cliqué sur le bouton "Bouton 3" dans le temps imparti, la boîte de dialogue indique "Vous n'avez cliqué sur aucun bouton. Pour information, il fallait cliquer sur le bouton 3".
>
> Il faut alors que l'utilisateur clique sur le bouton "OK" de la boîte de dialogue ou attende 3 secondes pour qu'elle se ferme. On affiche alors l'écran **Écran 3**.

Écran 3 : Cet écran affiche l'image relative à la question (2, 3, 4 ou 5), ainsi que les boutons "Bouton 1", "Bouton 2" et "Bouton 3". Cet affichage dure tant que l'utilisateur n'a pas appuyé sur un bouton. En mode "Difficile", cet affichage n'excède pas 5 secondes.

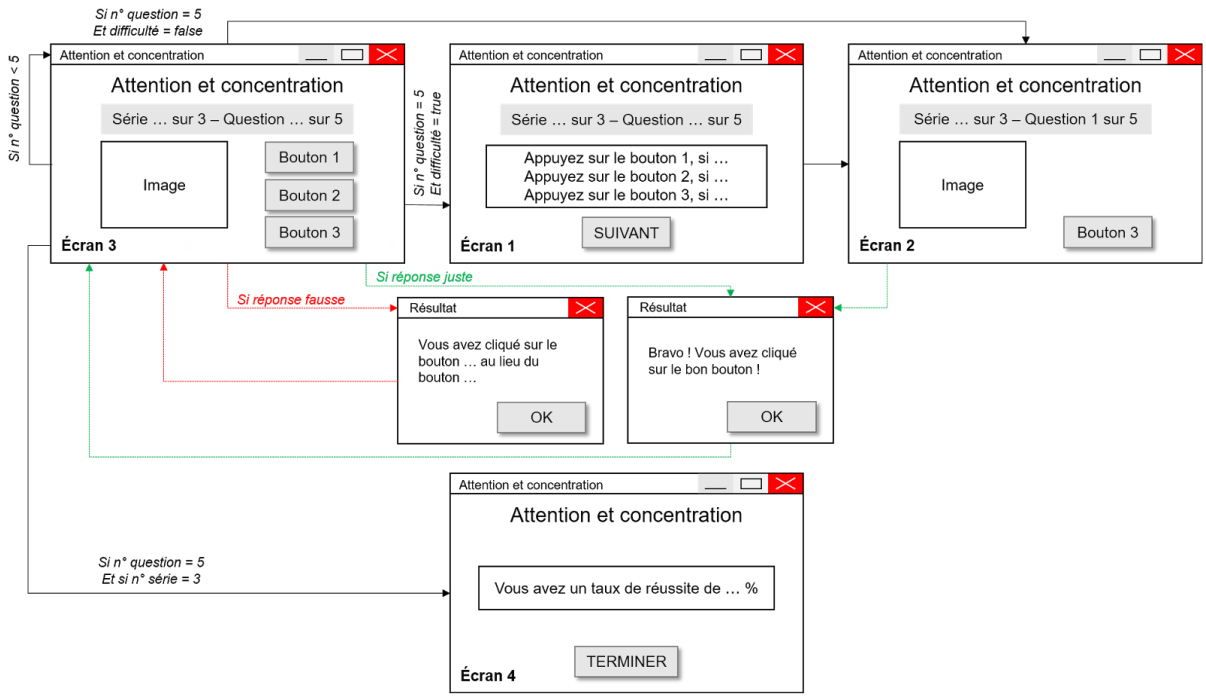
> Quand l'utilisateur clique sur un bouton, on efface l'écran en cours et une boîte de dialogue s'ouvre avec la mention "Bravo ! Vous avez cliqué sur le bon bouton !" si c'était le bon bouton, et la mention "Vous avez cliqué sur le bouton … au lieu du bouton …" si c'était le mauvais bouton. En mode "Difficile", s'il n'a pas cliqué sur un

bouton dans le temps imparti, la boîte de dialogue indique “Vous n’avez cliqué sur aucun bouton. Pour information, il fallait cliquer sur le bouton …”.

Il faut alors que l’utilisateur clique sur le bouton “OK” de la boîte de dialogue ou attende 3 secondes pour qu’elle se ferme. On affiche alors l’écran **Écran 3** si les 5 questions de la série en cours ne sont pas terminées, l’écran **Écran 2** si les 5 questions de la série en cours sont terminées en mode “Facile”, l’écran **Écran 1** si les 5 questions de la série en cours sont terminées en mode “Difficile” ou l’écran **Écran 4** si les 5 questions de la série en cours et que les 3 séries sont terminées.

Écran 4 : Dans le cas où l’utilisateur était en train de répondre à la dernière question de la dernière série, on efface l’écran en cours et on affiche à la place son résultat (proportion de réponses justes) avec un bouton “Terminer”. Si l’utilisateur appuie sur le bouton “Terminer”, le formulaire se ferme.

Ci-dessous l’illustration du fonctionnement du test d’attention et concentration :



3.1.6. Formulaire de calcul mental “TestCalcul”

Écran 1 : Tout d’abord, un premier écran permet au joueur de choisir sur quelle opération il veut se tester grâce à 4 boutons portant le signe de chacune de 4 opérations possibles.

Écran 2 : Lorsque l’utilisateur a cliqué sur le bouton correspondant à l’opération sur laquelle il veut se tester, on affiche le numéro de la question, le calcul à faire, un champ à compléter par le joueur (pour écrire le résultat du calcul) et un bouton “Valider” pour confirmer sa réponse. Cet écran reste affiché tant que l’utilisateur n’a pas cliqué sur ce dernier ou, en mode “Difficile”, tant que l’affichage a duré moins de 5 secondes.

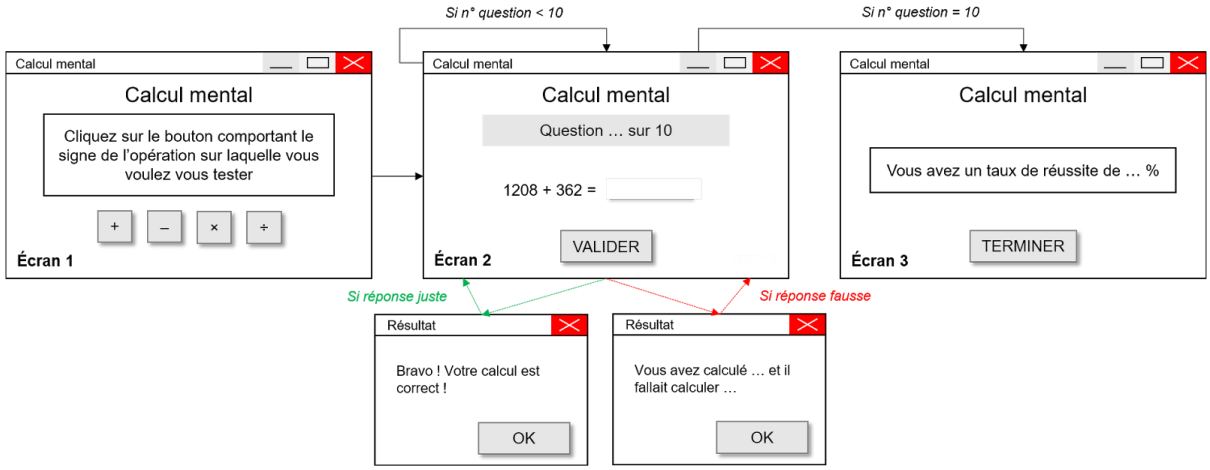
Quand l'utilisateur clique sur le bouton "Valider", une boîte de dialogue s'ouvre avec la mention "Bravo ! Votre calcul est correct !" s'il a inscrit le bon résultat ou la mention "Vous avez calculé … et il fallait calculer…" s'il a inscrit le mauvais résultat.

En mode "Difficile", s'il n'a pas cliqué sur le bouton "Valider" dans le temps imparti, la boîte de dialogue indique "Vous n'avez pas entré de résultat. Pour information, il fallait trouver …".

Il faut alors que l'utilisateur clique sur le bouton "OK" de la boîte de dialogue ou attende 3 secondes pour qu'elle se ferme. On efface alors l'écran en cours et on réaffiche l'écran **Écran 2** avec la question suivante si l'utilisateur n'a pas fini les 10 calculs du test, ou l'écran **Écran 3** dans le cas où le test est terminé.

Écran 3 : Dans le cas où l'utilisateur était en train de répondre à la dernière question, on efface l'écran en cours (**Écran 2**) et on affiche à la place son résultat (proportion de réponses justes) avec un bouton "Terminer". Si l'utilisateur appuie sur le bouton "Terminer", le formulaire se ferme.

Ci-dessous l'illustration du fonctionnement du test de calcul mental :



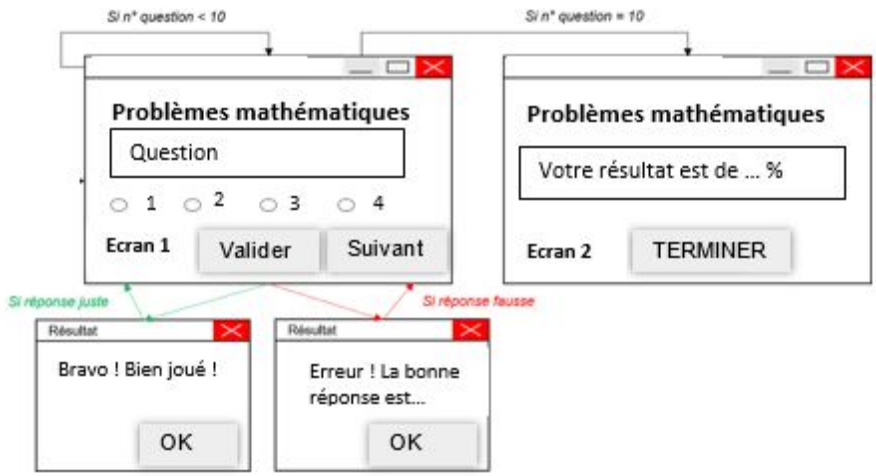
3.1.7. Formulaire de problèmes mathématiques "TestMaths"

Écran 1 : Cet écran présente la question avec les quatre propositions de réponse. L'utilisateur cochera 1 ou 2 ou 3 ou 4 en accord avec la proposition qu'il juge la plus pertinente.

S'il valide et que sa réponse est fausse, un message d'erreur s'affiche. Si le cas inverse se produit, un pop-up de bonne réponse s'affiche alors.

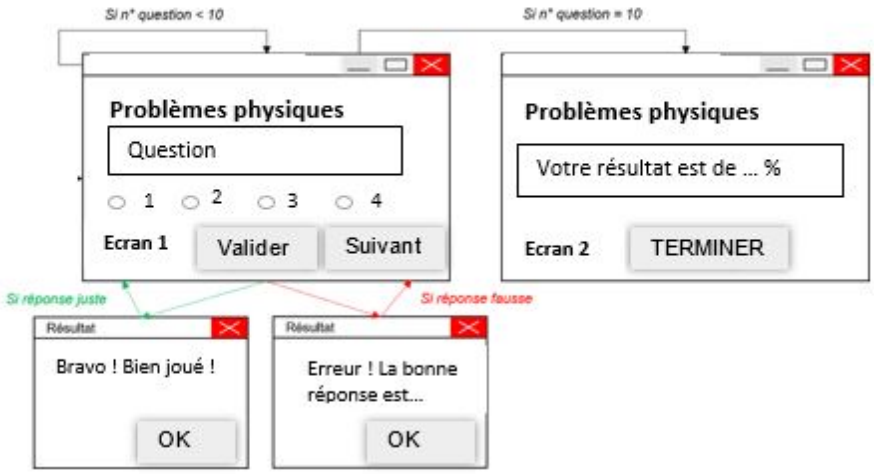
Écran 2 : Cet écran s'affiche quand l'utilisateur a répondu aux 10 questions. Il lui indique son résultat.

La différence entre le niveau facile et difficile est la nature des questions.



3.1.8. Formulaire de problèmes physiques “TestPhysique”

Ce formulaire présente le même principe de fonctionnement que la partie Problèmes Mathématiques.



3.1.9. Formulaire de résultat “TestResultat”

Ce formulaire permet d’afficher le message mentionnant le résultat de l’utilisateur pour une question donnée.

Il possède un bouton “OK” qui permet de fermer le formulaire au même titre que la croix rouge mais se ferme automatiquement au bon de 3 secondes.

Ce formulaire possède un attribut privé : le message à afficher. Le constructeur prend donc en argument une chaîne de caractères correspondant au message que l’on souhaite montrer à l’utilisateur.

3.2. Choix de conception et de production du code source

3.2.1. Gestion des données

Nous avons choisi de mettre en place une base données pour stocker l'ensemble des éléments nécessaires à la mise en place des formulaires (questions, images, consigne etc.)

Cela nous semblait plus pertinent que la sérialisation car ainsi nous pouvions récupérer les différents éléments au moment voulu par des requêtes précises, et non l'ensemble des données d'un seul coup.

Par ailleurs, ayant déjà utilisé la sérialisation lors du projet de programmation orientée objet de l'année passée, nous trouvions cela plus formateur.

3.2.2. Choix de conception

Nous n'avons pas utilisé de patron de conception particulier.

Nous avons cependant établi certains principes afin que notre code soit le plus propre possible.

Ainsi, nous avons respecté le principe de **partage des responsabilités** et **de responsabilité unique**. En effet, nous avons divisé notre projet en deux grands parties : une partie bibliothèque de classes et une partie application Windows Form. Ce sont donc dans les classes que nous définissons les attributs et les méthodes. C'est donc également dans les classes que nous définissons les requêtes à effectuer dans notre base de données Access.

De plus, au sein de chaque classe il n'y a pas de définition de code permettant d'afficher des informations, cette fonction étant réservée aux formulaires. La partie Windows Form est quant à elle destinée à comporter le code servant pour l'affichage utilisateur, les éventuels timers, ainsi qu'à appeler les méthodes adéquates en fonction des choix de l'utilisateur.

Lorsque nous avons mis en place notre diagramme de classe, nous avons également réfléchi à respecter le principe **Don't Repeat Yourself** afin d'éviter les redondances de code. Par exemple, les problèmes Mathématiques et Physiques ayant des modes de fonctionnement similaires, nous avons implémenté les méthodes dans une classe parente intitulée "TestProblemes" afin d'éviter de retrouver deux fois le même code dans les classes "TestProblemesMathematiques" et "TestProblemesPhysiques".

De plus, nous avons utilisé la même interface, *i.e.* le même formulaire, pour l'affichage des consignes et des écrans de démonstration, le fonctionnement étant le même pour chaque type de tests.

Nous avons également essayé de respecter le principe **Keep It Simple Stupid**, c'est-à-dire que nous avons autant que possible privilégié la simplicité. Ainsi, par exemple pour la partie

Calcul Mental, nous utilisons les calculs contenus dans notre base de données, nous n'avons pas créé une fonction qui génère aléatoirement un calcul pour chaque question.

De même pour les tests 1 et 2, nous utilisons des images "pré-faites" dont l'emplacement et le nom sont stockés dans la base de données. Cela nous évite ainsi de devoir générer des formes dans notre application, de devoir ainsi mettre en place de nombreuses vérifications pour savoir si l'image générée répond aux critères demandés, et d'avoir à créer les réponses "sur le tas".

3.2.3. Production du code source

Pour produire le code source, nous avons choisi d'utiliser GitHub afin de faciliter le partage de code et d'avoir un historique de nos versions. Cela nous a permis de nous familiariser avec les commandes Git et à utiliser une invite de commandes.

Cela nous a également incité à davantage travailler sur nos parties respectives afin d'éviter au maximum les conflits. Ainsi, nous n'empiétions pas sur le travail de l'autre.

Nous nous sommes donc répartis les différents tests et formulaires et avons codé plus ou moins séparément en ayant préalablement fixé des conventions à respecter et en ayant déjà la structure de l'application (bibliothèque de classes, application Windows Form, méthodes et attributs des classes, etc.).

3.2.4. Conventions de nommage

Tous les noms de nos variables et méthodes sont en français.

Nous avons cependant laissé le nom des composants des formulaires ainsi que leurs événements en anglais (par exemple : "Button", "Label", "TextBox", "Click", "FormLoad").

On obtient donc les conventions suivantes :

- Les noms des variables et propriétés sont de la forme : maVariable, maPropriete
- Les noms des méthodes sont de la forme : MaMethode
- Les noms des composants sont de la forme : MonComposant_TypeOfComponent
- Les noms des méthodes associées à des actions sur les composants des formulaires sont de la forme : MonComposant_TypeOfComponant_TypeOfEvent

4. Résultats et tests

4.1. Captures d'écran de l'application

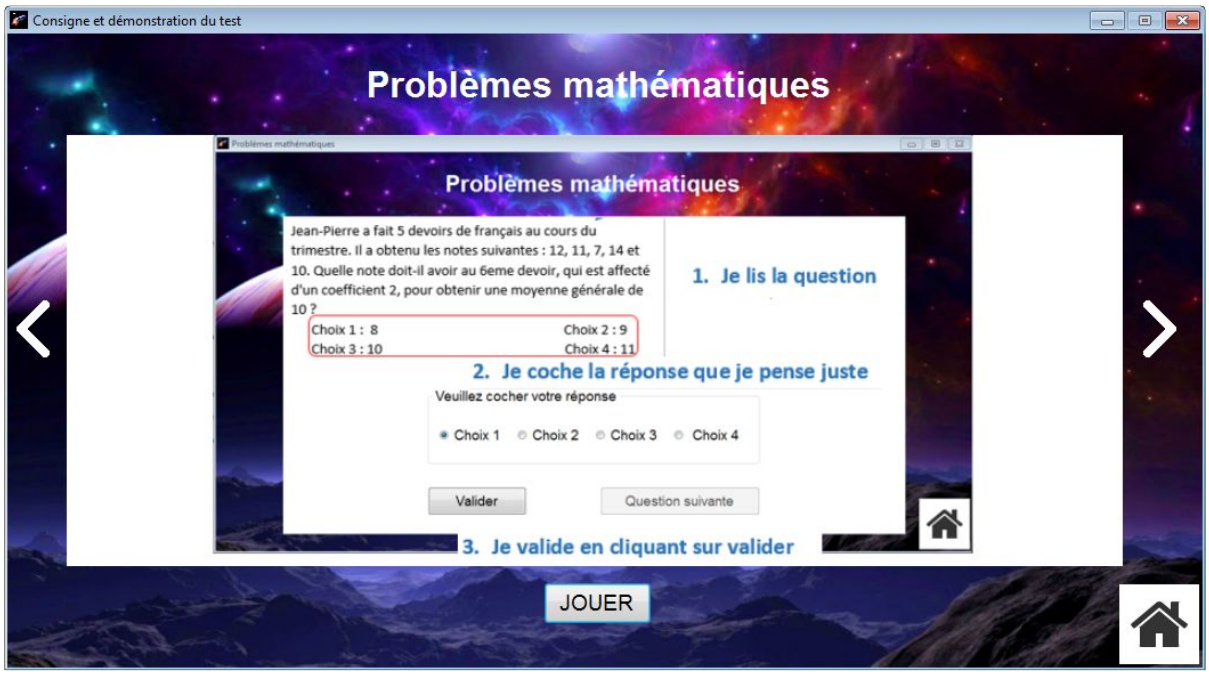
N.B. : Pour éviter de surcharger le rapport, nous n'avons pas mis de captures de chaque test.

Voici des captures d'écran de notre application à des moments clés :

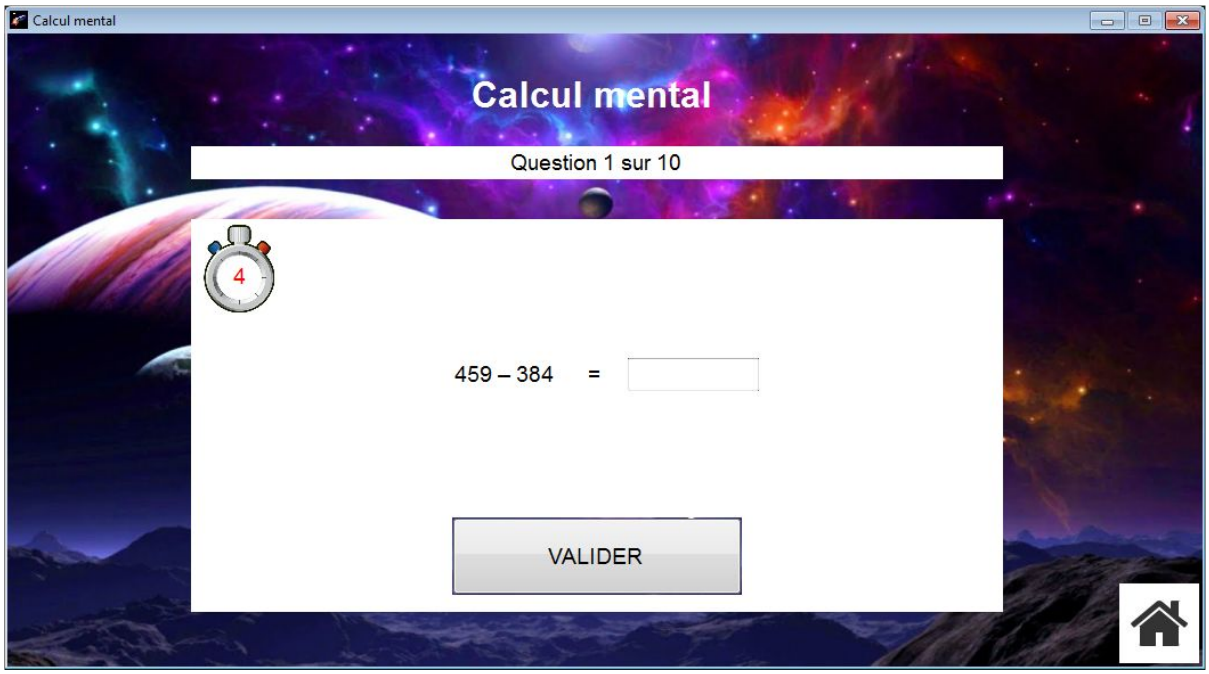
Écran de bienvenue

Écran de menu

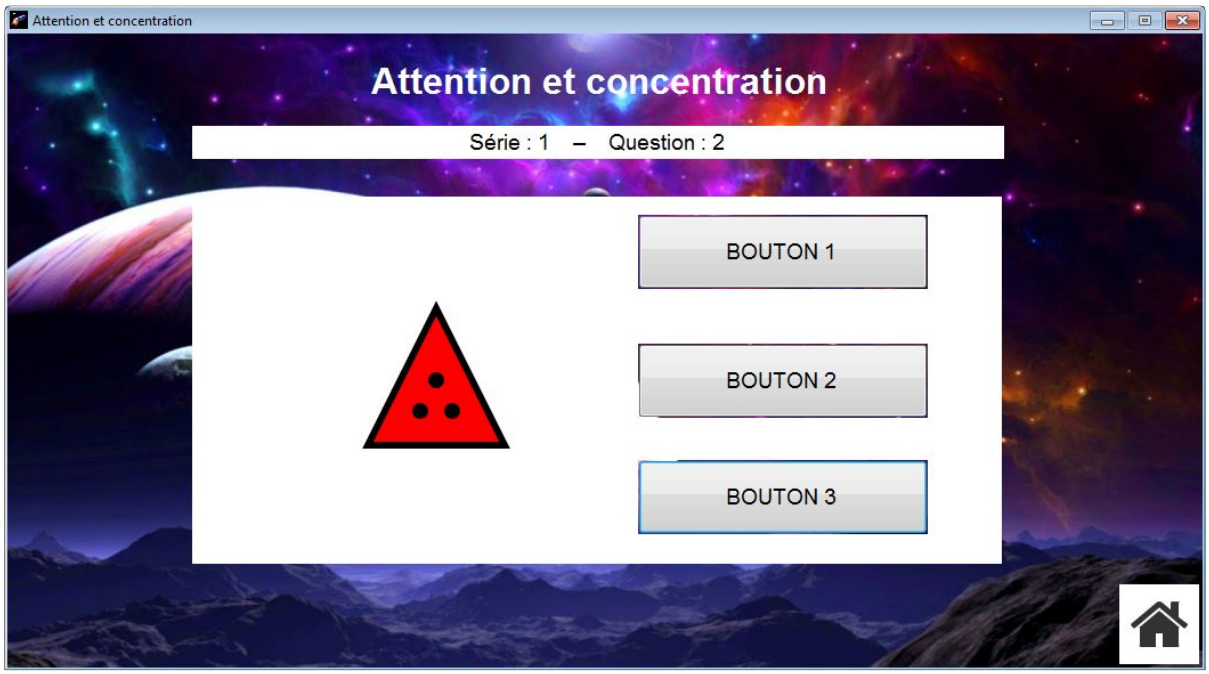
Écran de consigne



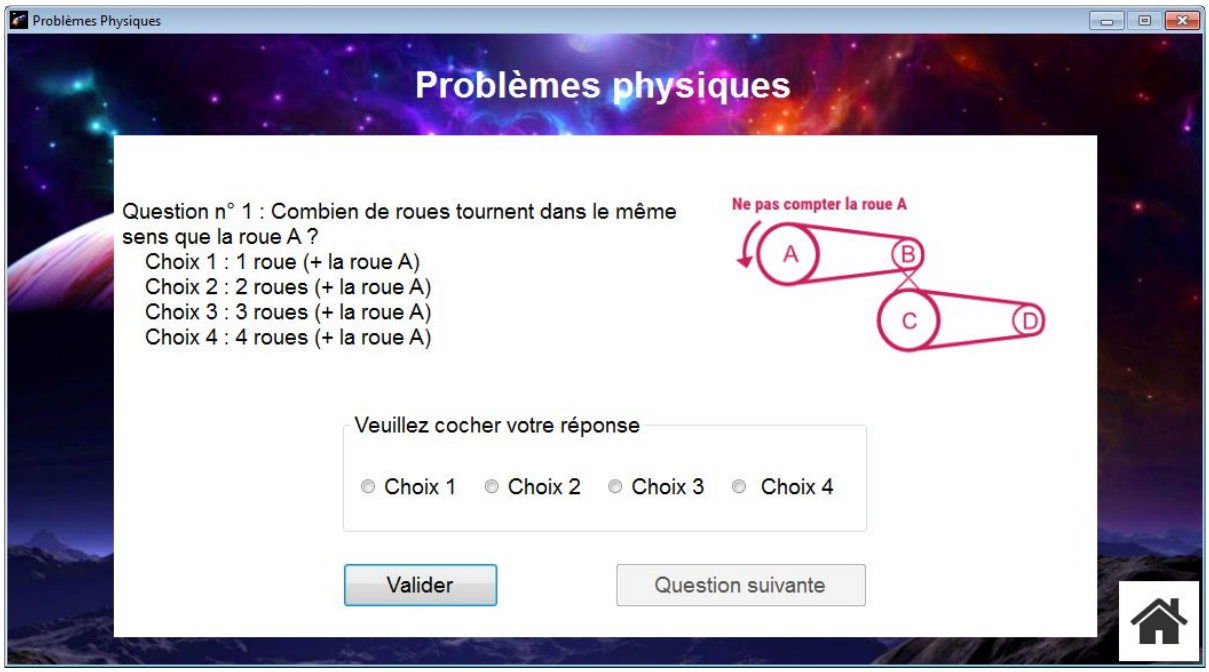
Écran de démonstration



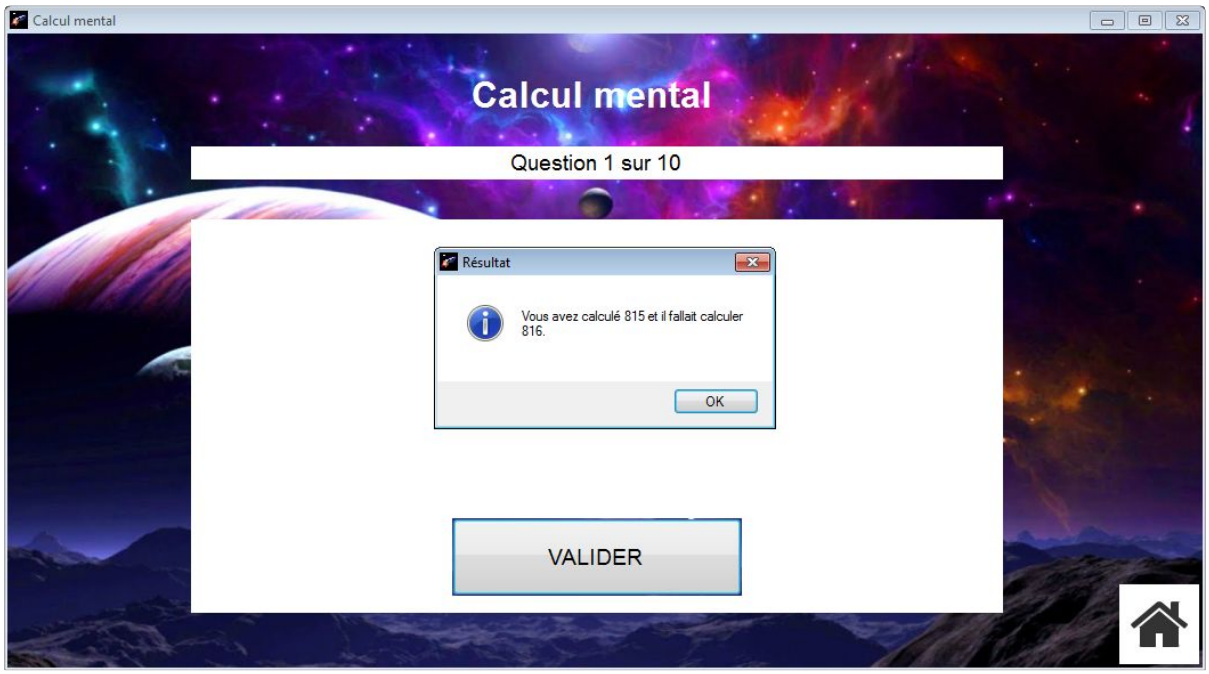
Question limitée dans le temps



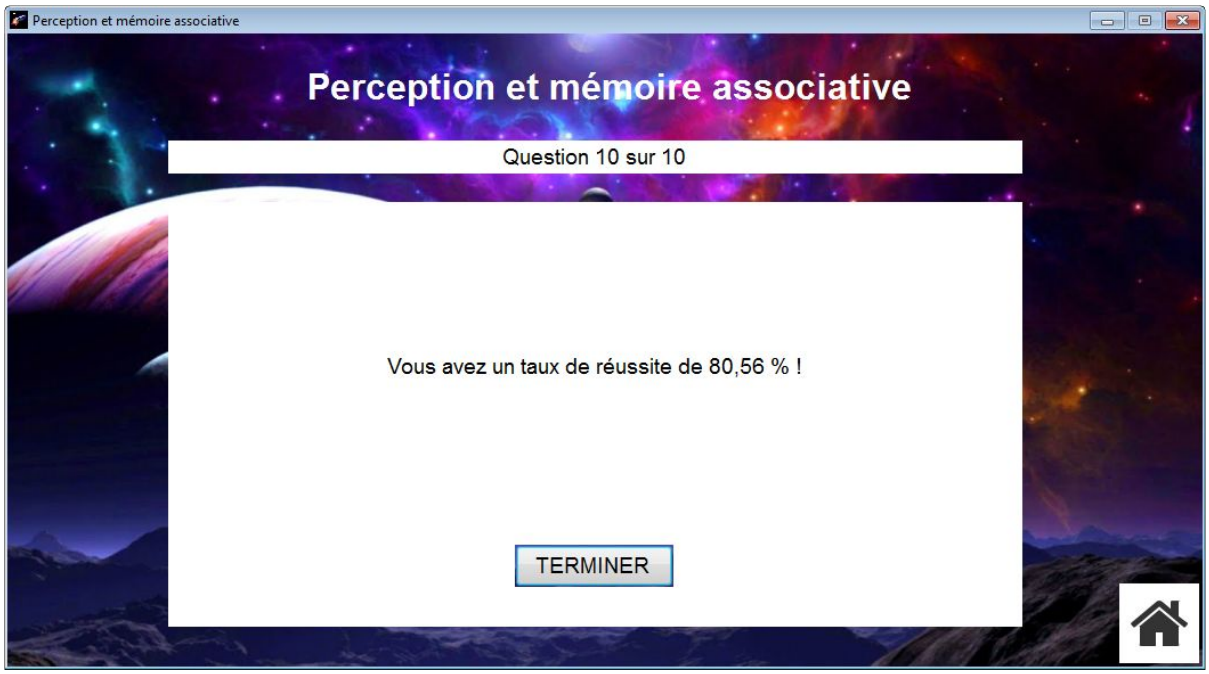
Question avec affichage d'une image



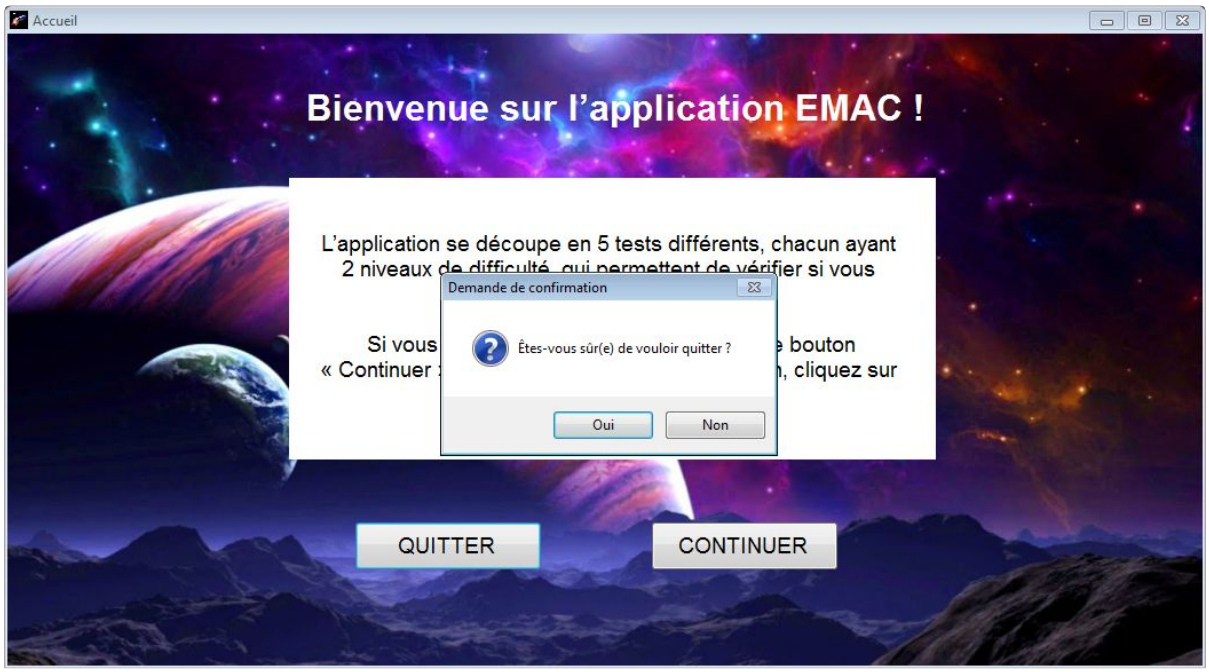
Question d'un problème



Pop-up limité dans le temps de résultat



Écran de résultat



Pop-up pour quitter l'application

4.2. Protocoles de tests

Nous avons fait des tests unitaires sur les différentes méthodes implémentées au sein de nos classes, et des tests fonctionnels sur les formulaires et l'ensemble de l'application.

4.2.1. Tests unitaires

Nous avons mis en place des tests unitaires pour tester les méthodes de nos classes (dans “EMACClass”), à savoir “VerifierReponse”, “AfficherErreur” et “CalculerResultat”.

Cela nous permet de vérifier que ces méthodes, qui sont appelées dans nos formulaires, renvoient bien ce que l’utilisateur doit voir.

La construction des tests étant aléatoire (on ne peut pas prédire à l’avance quelles questions, et donc quelles réponses, il va y avoir dans le test), nous avons créé des constructeurs exprès pour les tests unitaires.

Ces derniers ne nécessitent pas la déclaration de tous les attributs. En effet, les méthodes sur lesquelles on fait les tests n’utilisent généralement que la liste des réponses. Ils ne prennent donc en argument que la liste de réponses (et éventuellement un autre argument selon le type de test).

Ainsi, grâce à ces constructeurs il nous est permis de connaître exactement les réponses aux tests dans la mesure où c’est nous qui les choisissons.

Le protocole utilisé est identique pour chaque test unitaire. D’abord on construit une instance de la classe avec les constructeurs dédiés. Ensuite on déclare les éléments nécessaires à l’appel de la méthode à tester. Puis on déclare l’élément attendu et récupère l’élément réel *via* un appel à la méthode testée. Enfin, on compare les éléments attendu et réel. Dans le cas d’une liste, on vérifie le nombre d’éléments dans la liste puis on vérifie un à un ces éléments.

Chaque test unitaire doit être testé dans un cas favorable “Succès” et défavorable “Échec”, et au moins 2 fois (avec des paramètres différents) dans le cas favorable. De cette façon, nous saurons avec une plus grande fiabilité que nos méthodes renvoient bien ce qu’on souhaite obtenir.

4.2.2. Tests fonctionnels

Les tests fonctionnels sont divisés en 2 catégories : les tests vérifiant les exigences fonctionnelles (EF) et les tests vérifiant les besoins utilisateurs (BU).

Voici le tableau recensant l’ensemble de nos tests fonctionnels :

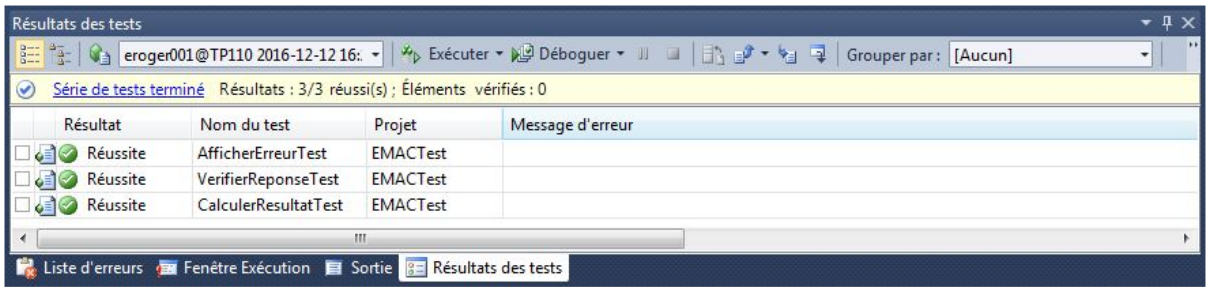
Référence	**Exigences**	**Importance**
EF_01	La page de Bienvenue explique l’objectif et le fonctionnement de l’application	Faible
EF_02	Sur la page de Menu, pour chaque test lorsque l’utilisateur clique sur un bouton Facile/Difficile, les questions qui s’affichent correspondent au bon type de test et au niveau sélectionné.	Forte

EF_03	Pour chaque test, les questions doivent être générées aléatoirement	Moyenne
EF_04	Si le test comporte des images, ces images s'affichent pour la question adéquate	Forte
EF_05	Pour chaque question, la réponse générée correspond bien à celle stockée dans la base de données pour la question	Forte
EF_06	Les tests Problèmes Physiques/Problèmes mathématiques/ Calcul Mental comportent 10 questions	Faible
EF_07	Une consigne et une démonstration sont présentées avant de commencer chaque test	Forte
EF_08	L'interface montrant la consigne et la démonstration est sous forme de "ruban défilant"	Faible
EF_09	Au cours d'un test, l'utilisateur doit pouvoir revenir à chaque instant au menu principal	Faible
EF_10	Lorsque l'utilisateur valide sa réponse, le pop-up correct s'affiche (erreur si sa réponse est erroné, correct si sa réponse est juste)	Forte
EF_11	Pour les catégories Attention Concentration et Calcul Mental, le pop-up de validation affiché a une durée de trois secondes.	Moyenne
BU_01	L'utilisateur peut seulement quitter l'application depuis la page de Bienvenue	Moyenne
BU_02	Lorsque l'utilisateur veut quitter l'application, une fenêtre pop-up s'affiche pour lui demander confirmation	Moyenne
BU_03	L'utilisateur doit pouvoir connaître sa progression au cours d'un test	Faible
BU_04	Si, au cours d'un test, la réponse à la question est limitée dans le temps, l'utilisateur doit connaître le nombre de secondes qui lui reste avant échéance.	Moyenne

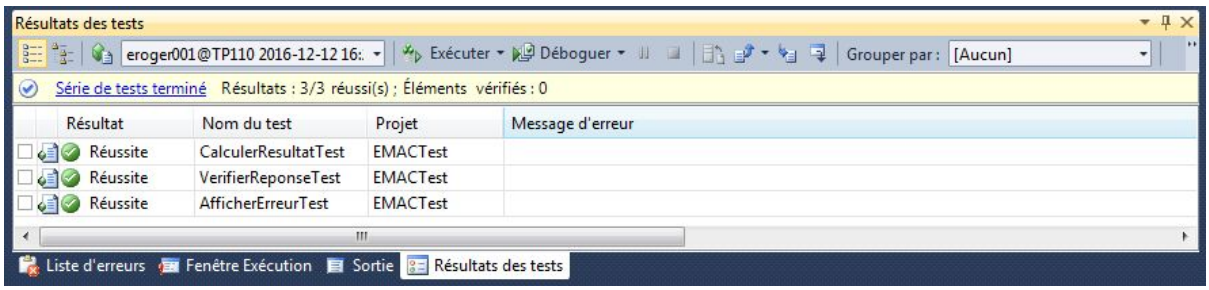
4.3. Résultats des tests

4.3.1. Tests unitaires dans un cas favorable

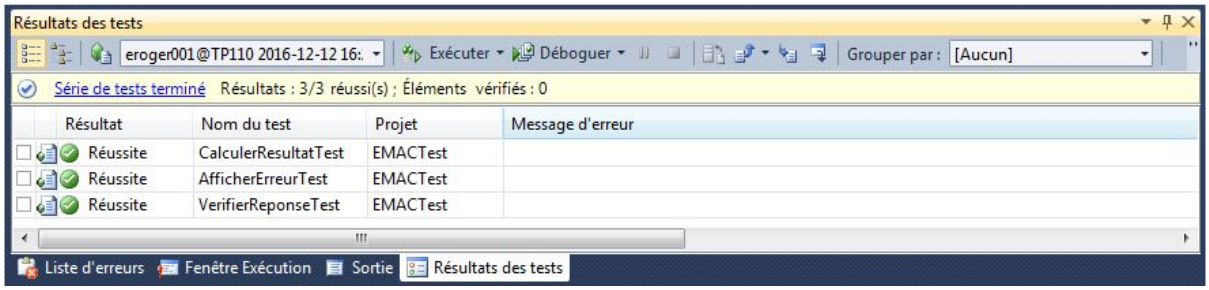
Tests des méthodes de la classe "TestPerceptionMemoire" :



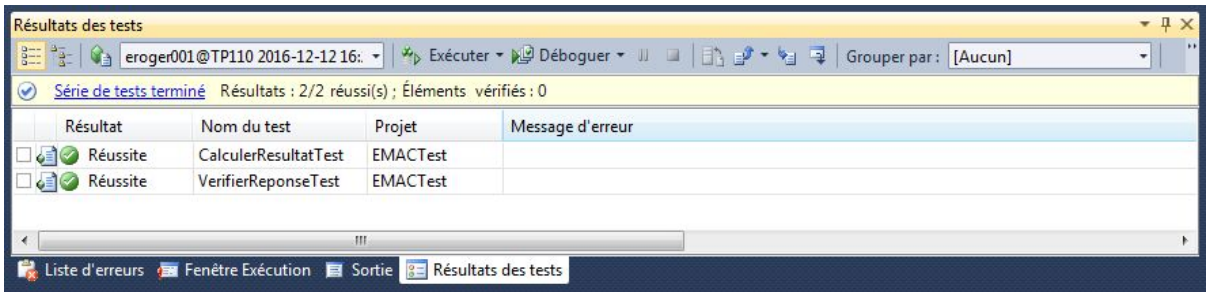
Tests des méthodes de la classe “TestAttentionConcentration” :



Tests des méthodes de la classe “TestCalculMental” :



Tests des méthodes de la classe “TestProblemes” (Mathématiques et Physiques) :



4.3.2. Tests fonctionnels

Voici les résultats obtenus en faisant nos tests fonctionnels :

Réf. exigence	Test	Résultat
EF_01	Vérification en affichant la page au lancement de l’application	OK

EF_02	Test de l'ensemble des boutons Facile/Difficile de la page Menu puis vérification pour chaque question du test si elle correspond au test et niveau défini en regardant dans la base de données	OK
EF_03	Vérification pour chaque Test que les questions sont générées aléatoirement avec la base de données	OK
EF_04	Vérification pour chaque les images s'affichent au bon endroit à l'aide de la base de données	OK
EF_05	Pour deux séries de chaque Test, vérification que la réponse affichée est bien celle contenu dans la base de données	OK
EF_06	Vérification pour chacun des tests mentionnés en comptant le nombre de question et en vérifiant à l'aide des numéros de questions affichés	OK
EF_07	Vérification de la présence d'une consigne et d'une démonstration pour chacun des tests	OK
EF_08	Vérification pour chaque test que l'intégralité de la consigne s'affiche en cliquant sur les flèches à gauche et à droite de l'écran pour faire défiler la démonstration	OK
EF_09	Vérification de la possibilité de revenir au menu principal en cliquant sur le bouton de retour dans chacun des formulaires de Test	OK
EF_10	Vérification de l'affichage du pop-up correct en comparant l'affichage du pop-up et la réponse à la question stockée dans la base de données	OK
EF_11	Vérification en décomptant à l'aide d'une montre chronomètre le temps d'affichage des pop-up à temps limité	OK
BU_01	Vérification en essayant de quitter l'application depuis les formulaires de Test	OK
BU_02	Vérification en essayant de quitter l'application	OK
BU_03	Vérification que le numéro de la question est présent pour chaque question de l'ensemble des formulaires	OK
BU_04	Vérification du décompte du temps pour les questions et les niveaux à temps limitée	OK

5. Bilan et perspectives

5.1. Points positifs et négatifs

5.2.1. Points positifs et apport pédagogique

Ce projet nous a permis de voir comment connecter une base de données à Visual Studio et d'aborder la syntaxe afin de mettre en place des requêtes par rapport à cette base de données. Il nous a également permis d'utiliser le logiciel GitHub pour le partage de code, ce qui n'a pas toujours été aisé mais qui a été enrichissant et nous a permis de prendre conscience de sa puissance et de son utilité.

Nous avons aussi pu perfectionner nos connaissances et compétences sur l'utilisation des Windows Forms. Nous avons notamment pu voir la gestion des formulaires et comment passer d'un formulaire à un autre de façon modale. Nous avons également pu voir comment concilier l'utilisation d'une bibliothèque de classes et d'application Windows Form, ce qui au final s'est révélé plus "propre" pour la gestion du code.

Nous n'avons pas rencontré de difficultés quant au travail en binôme car dès le départ nous nous étions bien réparties le travail. Cela a permis notamment d'éviter la résolution de conflits.

Pour finir, ce projet a été l'occasion de mettre en place des tests unitaires et des tests fonctionnels, ce que nous ne faisions pas l'année passée dans nos différents projets d'informatique. Cela s'est révélé très intéressant dans la mesure où cela nous aide à mieux définir quelles fonctionnalités et quelles exigences on fixe.

5.2.2. Points négatifs et difficultés

La prise en main de GitHub nous a demandé un certain temps d'adaptation (une demi-séance de TD pour l'utiliser convenablement).

Certaines difficultés ont également été rencontrées lors du codage de l'application. Par exemple, la gestion des timers qu'on n'oubliait d'arrêter ou encore la récupération aléatoire des questions dans la base de données. Initialement, nous avions une requête SQL qui nous renvoyait 10 questions aléatoirement, mais à chaque fois qu'on créait un nouveau test, nous obtenions les dix mêmes questions.

Nous avons encore quelques difficultés pour mettre en place des exceptions et ne savons pas trop où il est vraiment pertinent de les placer.

5.2. Évolutions possibles

Une évolution possible à notre projet serait de mettre en place une classe utilisateur. De cette manière, un compte utilisateur pourrait être créé et il y aurait possibilité de connexion/déconnexion. Cela permettrait donc de personnaliser notre application. On

pourrait également stocker les résultats de l'utilisateur afin de lui montrer l'évolution de ses performances.

D'autre part, pour les problèmes physiques et mathématiques, des questions pourraient être ajoutées afin de diversifier les questions proposées et que l'utilisateur ne connaissent pas d'avance les réponses s'il utilise notre application plusieurs fois.

De même, il serait intéressant d'enrichir notre base de données avec d'autres questions pour les autres tests, voire de les générer directement aléatoirement de notre application.

Enfin, il serait intéressant de proposer ce logiciel dans différentes langues étant donné que l'ESA est l'Agence Spatiale Européenne et qu'en Europe on ne parle pas que le français…

MERCI D'AVOIR PRIS LE TEMPS DE LIRE TOUT NOTRE RAPPORT !

Nous espérons sincèrement qu'il vous a plu !