



AstronautComplex



Rapport de projet

Application de tests pour la sélection d'astronautes de l'ESA

Document	
Référence	GL_RP_001_2016_12_13
Projet	Application de tests pour la sélection des astronautes de l'ESA
Tuteur	Baptiste PESQUET
Date de début	13 décembre 2016
Membres	
Corentin BIENASSIS	Thibaud LAURENT

Sommaire

Sommaire

Introduction

I. Spécifications générales

- A. Analyse des exigences fonctionnelles
 - 1. *Exercice de perception et de mémoire associative*
 - 2. *Exercice d'attention et de concentration*
 - 3. *Exercice de calcul mental*
 - 4. *Exercice de questions de culture scientifique*
- B. Analyse des exigences interface
- C. Diagramme de classes
- D. Diagramme de séquences
 - 1. *ExercicePerception*
 - 2. *ExerciceFocus*
 - 3. *ExerciceCalculus*
 - 4. *ExerciceQuestion*
- E. Architecture
- F. Organisation

II. Spécifications détaillées

- A. Liste des formulaires de l'interface
- B. Choix de conception et de réalisation

III. Résultats et tests

- A. Captures d'écran
 - 1. *Menu principal*
 - 2. *Sélection de l'exercice de Mathématiques et choix de difficulté*
 - 3. *Affichage des instructions*
 - 4. *Affichage d'une question générée aléatoirement*
 - 5. *Affichage de la bonne réponse (en l'ayant trouvée)*
 - 6. *Affichage du pourcentage de bonnes réponses (à la fin)*
 - 7. *Retour au menu principal*
- B. Tests
 - 1. *Protocoles*
 - 2. *Résultats*
 - 3. *Tests fonctionnels*

Conclusion

- A. Bilan
- B. Perspectives et évolutions

Annexe 1 : Matrice des fonctionnalités

Annexe 2 : Diagrammes de classes

Annexe 3 : Diagramme de séquences

Annexe 4 : Planning prévisionnel

Annexe 5 : Planning final

Annexe 6 : Maquette

Introduction

L'ESA a besoin d'un logiciel pour tester des candidats souhaitant devenir astronautes. Pour assurer leur sélection selon des critères cognitifs, il a été demandé de développer une solution d'interface d'exercices permettant de les tester individuellement. Dans ce rapport sont détaillées les étapes mises en place pour répondre à ce besoin, ainsi les résultats de la solution dont le code source est disponible à l'adresse <https://github.com/Grox2006/AstronautComplex>.

Afin de ne pas porter confusion entre le concept de test dans la solution et celui de test en développement informatique, les tests pour la sélection des astronautes sont nommés dans ce projet "exercices".

Ce projet s'inscrit dans le cadre du module "Génie Logiciel" de l'ENSC.

I. Spécifications générales

A. Analyse des exigences fonctionnelles

Après étude du cahier des charges (fourni sous forme de sujet en deux parties), plusieurs fonctionnalités ont été relevées. Une matrice les regroupant, disponible en **Annexe 1**, a été réalisée afin de vérifier leurs avancements, mais aussi leurs états de tests. Voici leurs descriptions :

- **[AC_G_01]** possibilité de réaliser des exercices avec un niveau *facile* ou *difficile*
- **[AC_G_02]** menu principal permettant l'accès aux différents exercices, accessible depuis partout dans la solution
- **[AC_G_03]** explications à l'utilisateur avant chaque exercice
- **[AC_G_04]** affichage du pourcentage de bonnes réponses à la fin de chaque exercice

1. Exercice de perception et de mémoire associative

Cet exercice consiste à tester les capacités de mémoire associative du candidate grâce à des formes à retenir rapidement. Les fonctionnalités demandées sont :

- **[AC_TP_01]** affichage d'une règle de mémorisation aléatoire
- **[AC_TP_02]** affichage de 12 formes de nature (carrée ou ronde), de couleur (jaune ou bleue) et de valeur (entre 0 et 9) différentes et aléatoires, dont entre 3 et 4 correspondent à la règle de mémorisation
- **[AC_TP_03]** effacement de l'écran
- **[AC_TP_04]** requête à l'utilisateur de saisir la valeur de la forme correspondant à la règle de mémorisation
- **[AC_TP_05]** comptabilisation des bonnes réponses
- **[AC_TP_06]** répétition de l'exercice pour 10 règles différentes

2. Exercice d'attention et de concentration

Cet exercice consiste à tester les capacités de concentration du candidat grâce à des séries de formes à étudier rapidement. Les fonctionnalités demandées sont :

- **[AC_TA_01]** affichage d'une règle comportementale aléatoire
- **[AC_TA_02]** répétition de la même règle sur 5 formes différentes (appelée série)
- **[AC_TA_03]** affichage d'une forme de nature (carrée, ronde ou triangulaire), couleur (jaune, bleue, verte ou rouge) et d'un nombre de points (entre 0 et 4) aléatoires
- **[AC_TA_04]** distribution des rôles de 3 boutons (*bonne réponse*, *mauvaise réponse* et *aucune réponse*) de façon à ce qu'un bouton soit le bon au moins une fois dans une série
- **[AC_TA_05]** affichage de la distribution des rôles des 3 boutons
- **[AC_TA_06]** requête à l'utilisateur, dans un temps limité (3 secondes), d'appuyer sur le bouton de bonne réponse si les deux objets affichés consécutivement respectent la règle comportementale, ou d'appuyer sur le bouton de mauvaise réponse sinon

- **[AC_TA_07]** affichage d'erreur si l'utilisateur a donné une mauvaise réponse
- **[AC_TA_08]** effacement de l'écran
- **[AC_TA_09]** répétition de l'exercice pour 3 règles différentes

Pour le niveau de difficulté *facile*, les règles restent les mêmes mais aucun temps limite n'est donné pour répondre.

3. Exercice de calcul mental

Cet exercice consiste à tester la capacité de calcul d'opérations simples. Les fonctionnalités demandées sont :

- **[AC_TC_01]** affichage du choix de l'opération
- **[AC_TC_02]** sélection de l'opération
- **[AC_TC_03]** affichage d'une opération
 - addition avec opérandes comprises entre 100 et 999
 - soustraction avec la première opérande comprise entre 100 et 999, et la deuxième comprise entre 10 et la première
 - multiplication avec la première opérande comprise entre 1 et 99, et la deuxième comprise entre 1 et 9
 - division entière (arrondie au plus proche) avec la première opérande comprise entre 10 et 999, et la deuxième comprise entre 1 et 9
- **[AC_TC_04]** requête à l'utilisateur, dans un temps limité (5 secondes), de saisir le résultat de l'opération et de valider
- **[AC_TC_05]** affichage de la solution d'une opération
- **[AC_TC_06]** répétition de l'exercice pour 10 opérations différentes
- **[AC_TC_07]** affichage du pourcentage de bonnes réponses

Pour le niveau de difficulté *facile*, les règles restent les mêmes mais aucun temps limite n'est donné pour répondre. De plus, les intervalles spécifiés sont tous inclusifs.

4. Exercice de questions de culture scientifique

Cette partie se divise en deux exercices similaires, chacun consistant à tester les connaissances d'une discipline scientifique (mathématiques et physique). Les fonctionnalités demandées sont :

- **[AC_TQ_01]** sélection d'un problème aléatoire parmi une liste
- **[AC_TQ_02]** affichage d'un problème (inférieur à 4 lignes, avec un éventuel dessin pour l'illustrer)
- **[AC_TQ_03]** affichage des réponses possibles
- **[AC_TQ_04]** requête à l'utilisateur de sélectionner l'unique bonne réponse
- **[AC_TQ_05]** affichage de la bonne réponse au problème
- **[AC_TQ_06]** répétition de l'exercice pour 10 question différentes
- **[AC_TQ_07]** affichage du pourcentage de bonnes réponses

B. Analyse des exigences interface

La solution est une interface qui doit, selon les exigences, être à la fois le plus ergonomique et le plus convivial possible. Ainsi, plusieurs critères ergonomiques stricts ont été définis :

- **[AC_I_01]** menu d'accueil explicite
- **[AC_I_02]** clarté de l'interface
- **[AC_I_03]** adaptation au changement de taille de l'interface
- **[AC_I_04]** barre d'outils accessible depuis n'importe quel écran de l'interface
- **[AC_I_05]** style graphique cohérent et harmonieux
- **[AC_I_06]** le moins d'écrans possibles (changement d'écran sur la même fenêtre)

Le type d'interface à concevoir n'étant pas précisé, et compte tenu de ces exigences internes et du contenu de la formation reçue au préalable, il paraît nécessaire de devoir comparer les méthodes possibles de réalisation de cette interface.

Type d'interface	Avantages	Inconvénients
Bureau (C#, Console)	<ul style="list-style-type: none"> • Léger 	<ul style="list-style-type: none"> • Peu ergonomique et peu conviviale • Concepteur et générateur d'interface inexistant
Bureau (C#, WinForms)	<ul style="list-style-type: none"> • Concepteur et générateur d'interface simple • Utilisation de contrôles préfabriqués 	<ul style="list-style-type: none"> • Portabilité sur ordinateurs Windows uniquement • Vétusté esthétique des contrôles préfabriqués
Web (HTML/CSS)	<ul style="list-style-type: none"> • Style personnalisable • Portabilité totale sur tous les navigateurs internet 	<ul style="list-style-type: none"> • Style coûteux en temps de création

L'interface de type bureau en WinForms est préférée aux autres, notamment grâce à la grande capacité de son concepteur à pouvoir réaliser une maquette prête à être poursuivie en interface fonctionnelle avec très peu de coût horaire, mais aussi son utilisation de contrôles préfabriqués permettant une harmonisation certaine de l'interface. De plus, sa faible portabilité n'est en soi qu'un problème potentiel, étant donnée l'absence de précision quant à l'environnement sur lequel sera utilisé l'application.

C. Diagramme de classes

L'analyse des besoins fonctionnels et interface a conduit à l'élaboration d'un diagramme de classes, travail préliminaire permettant de faire le lien entre exigences et produit, en modélisant les concepts clés.

Ce diagramme de classes est disponible en **Annexe 2**.

D. Diagramme de séquences

Après élaboration du diagramme de classe, une différence a été faite entre les classes d'interface et les classes communs. Ainsi, 3 sous-interfaces ont été déterminées :

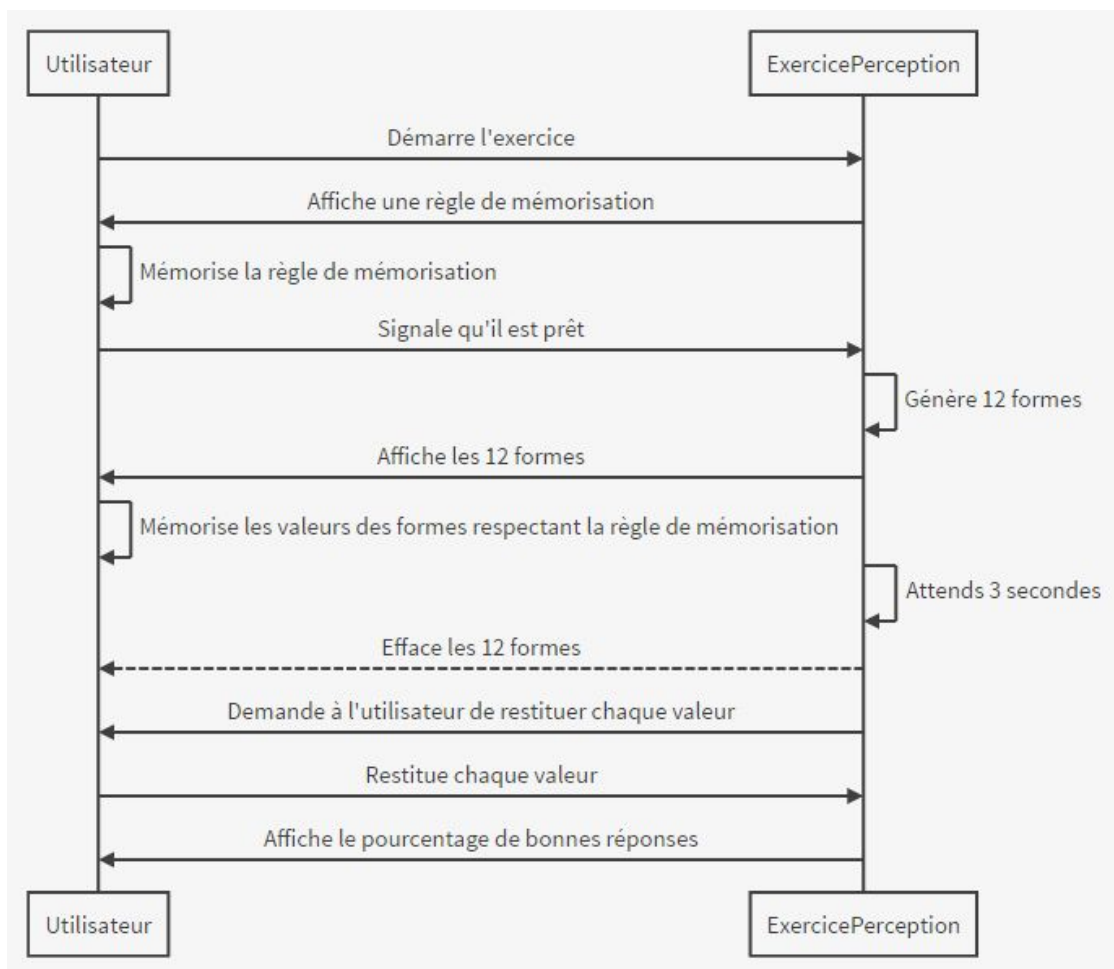
- **ExerciceForm**, la fenêtre d'affichage principale
- **ExerciceDialogDifficulty**, la boîte de dialogue permettant le choix de difficulté d'un exercice
- **Exercice**, le contrôle d'affichage générique d'un exercice

En effet, cette dernière classe, en étant héritée, permet de définir un nombre illimité de nouveaux exercices, répondant tous aux mêmes règles qu'elle établit.

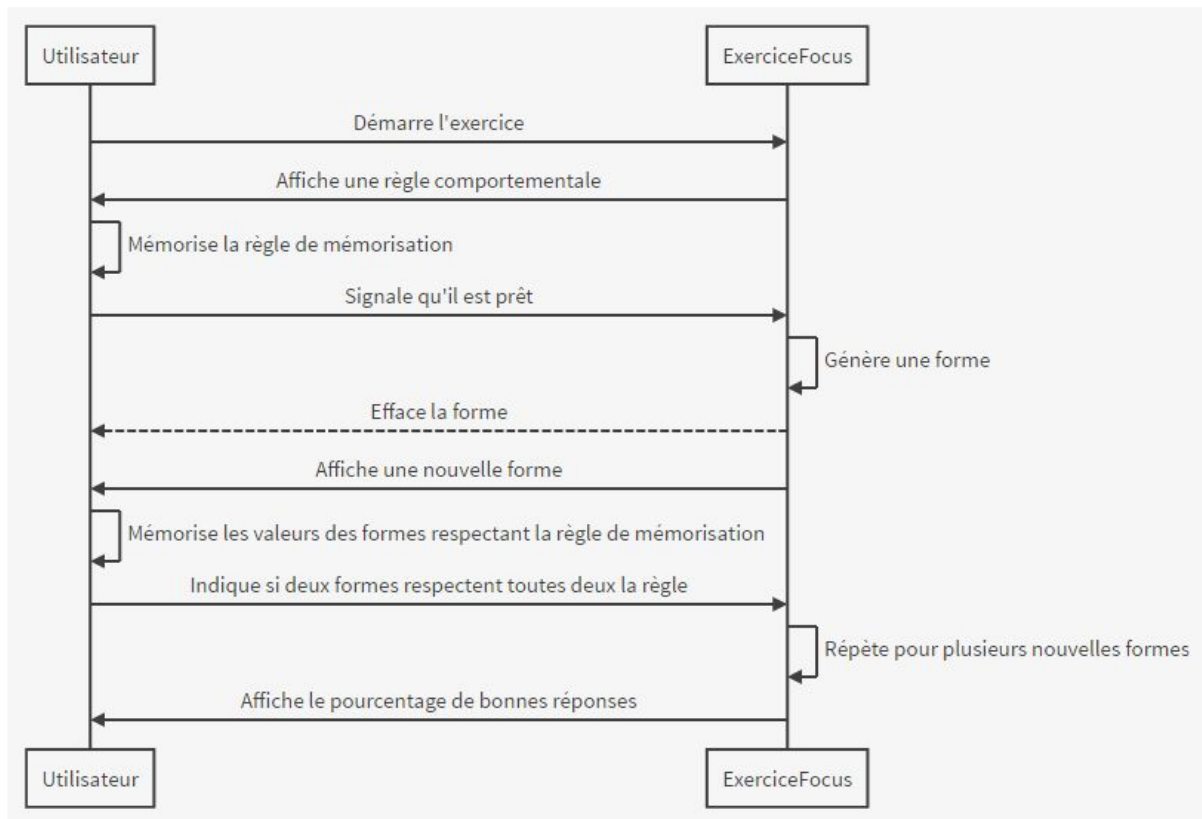
Ce diagramme de séquences en **Annexe 3**.

L'exécution d'un exercice est propre à ce dernier et n'opère de manière dépendante qu'en implémentant les différentes méthodes de **Exercice**. Les diagrammes de séquence de ces exercices sont décrits ci-dessous.

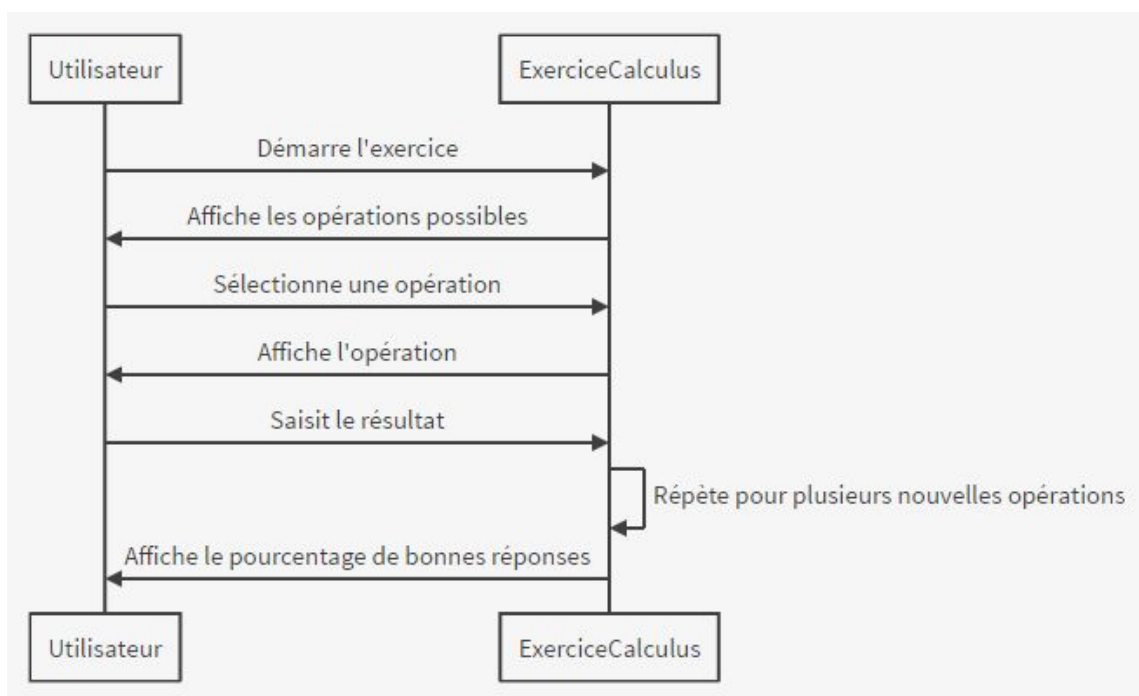
1. ExercicePerception



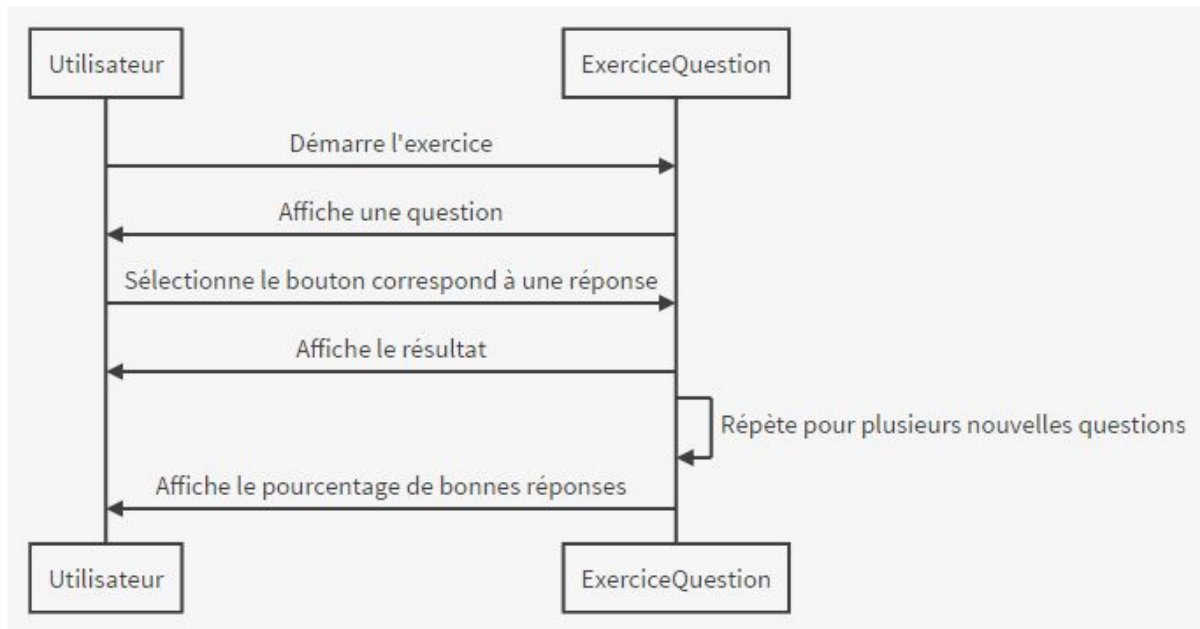
2. ExerciceFocus



3. ExerciceCalculus



4. ExerciceQuestion



E. Architecture

Le développement de nouveaux exercices est un travail répétitif et en partie automatisable. Ce constat a mené à une réflexion concernant l'architecture du projet : la sélection des astronautes étant sujette à de permanentes évolutions, comment permettre à quiconque d'ajouter ses propres exercices ?

L'architecture de la solution a été pensée pour pallier ce problème. Le principe est donc de pouvoir créer un nombre illimité d'exercices ; et pour ce faire, la solution a été découpée en deux parties :

- **AstronautComplex**, le moteur générique
- **AstronautComplexBasicPack**, un regroupement d'exercices correspondant aux exercices initialement demandés

Cette architecture respecte donc un patron de conception de type "moteur-plugin", inspiré du patron "bridge". Son fonctionnement est détaillée ci-dessous :



Les packs sont chargés par un algorithme de **ExerciceForm**, qui va chercher toutes les bibliothèques respectant ce format à l'endroit spécifié par le moteur, et charger à la volée tous les exercices qu'il trouve. Pour chaque nouvel exercice chargé, le moteur permet son accès de la manière précisée dans le diagramme de séquences.

Cette méthode permet ainsi de pouvoir créer un nombre illimité d'exercices pour la solution.

F. Organisation

L'équipe de développement de ce projet est composée des deux personnes rédigeant ce rapport :

- Corentin BIENASSIS
- Thibaud LAURENT

Les tâches ont été réparties de manière équilibrée, en fonction des souhaits et compétences de chaque membre. Le moteur est réalisé en parallèle des exercices ; les deux parties ne dépendant que d'une relation d'implémentation, l'adaptation d'un exercice au changement du moteur ne cause qu'une charge de travail mineure. Cette tâche étant lourde en travail, Thibaud s'occupe des deux exercices de culture scientifique (similaires), quand Corentin s'occupe des trois autres ainsi que de la réalisation des tests unitaires.

Le planning prévisionnel est disponible en **Annexe 4**, et le planning final en **Annexe 5**.

Afin de partager les sources de la solution, ces dernières ont été hébergées sur le site GitHub et leurs versions gérées avec Git. De plus, afin de pouvoir connaître l'état du code hébergé, un outil d'intégration continue, Travis CI, a été greffé au dépôt Git.

II. Spécifications détaillées

A. Liste des formulaires de l'interface

Afin de ne pas surcharger inutilement l'utilisateur avec des informations et des fenêtres supplémentaires, le choix a été fait de développer le plus de fonctionnalités dans le moins d'interfaces. Un seul formulaire complet est donc présent **ExerciceForm**, ainsi qu'une boîte de dialogue (pour choisir la difficulté) **ExerciceDialogDifficulty**, auxquels s'ajoutent les quatre formulaires d'exercices, chacun remplissant la zone centrale de **ExerciceForm**.

La maquette de ces formulaires, réalisée sous le Designer de Visual Studio 2015, est disponible en **Annexe 6**.

B. Choix de conception et de réalisation

Les exigences interface internes spécifient l'utilisation de la technologie WinForms avec le langage C#. La réalisation s'est axée sur des classes héritant de concepts abstraits :

- tous les exercices sont des implémentations de **Exercice**
- toutes les questions (générées aléatoirement dans le cadre d'un exercice de type "question") sont des implémentations de **Question**
- tous les polygones en 3D (qui permettent une représentation graphique de certaines questions) sont des implémentations de **Polygon3D**

Afin de modéliser les deux parties, la solution est divisée en deux projets, correspondant chacun à une partie. Le projet **AstronautComplex** charge tous les fichiers DLL présents dans un dossier *Plugins*, et pour chacun de ces fichiers, charge toutes les classes non-abstraites héritant de **Exercice** dans une liste d'exercices. Ce faisant, il est possible de développer de manière totalement indépendante un projet de type "bibliothèque de classes" référençant **AstronautComplex** pour créer des exercices inédits, en plaçant le fichier DLL de sortie dans le répertoire *Plugins*.

Enfin afin de respecter les standards du langage, le code respecte sa syntaxe par défaut et est écrit intégralement en anglais, commentaires inclus. Ces derniers sont présents à chaque en-tête de fonction afin de préciser sa description, ses arguments et son retour, et à chaque en-tête de fonction afin de préciser ce qu'elle représente.

III. Résultats et tests

A. Captures d'écran

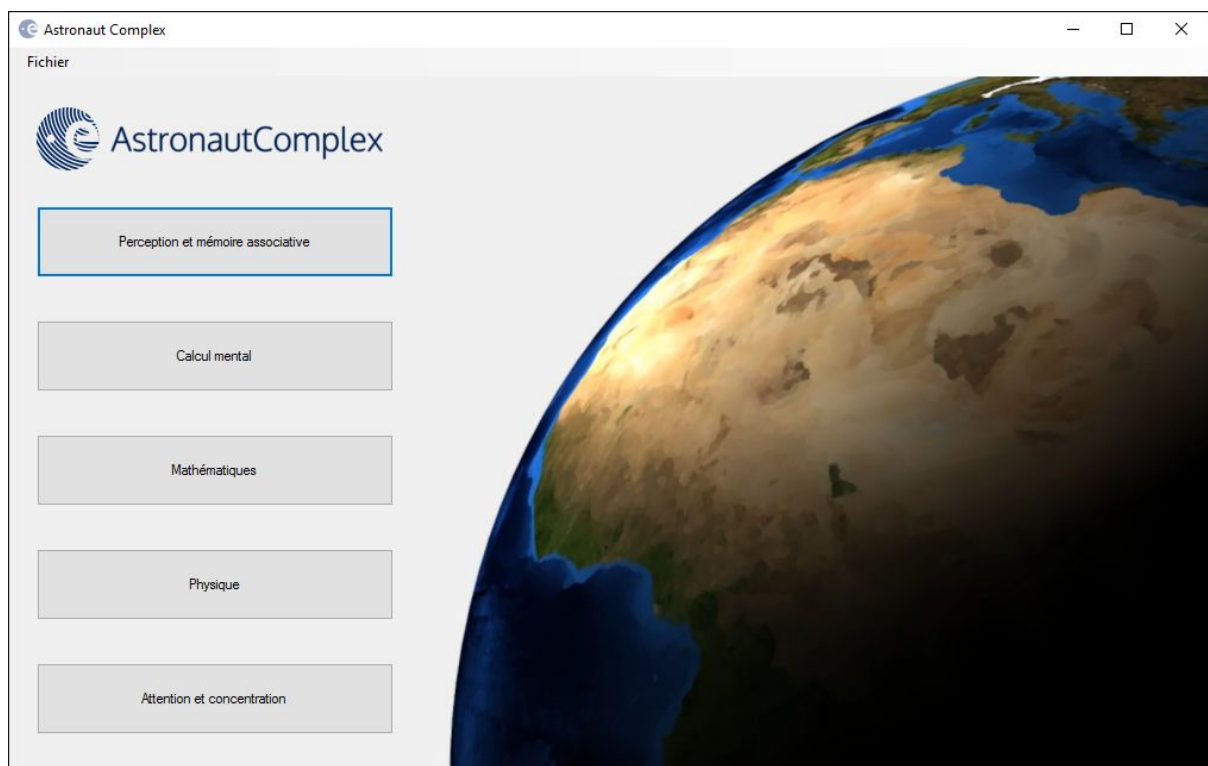
Le scénario exemple correspondant aux captures d'écran présentées est le suivant :

- Accéder au menu principal
- Réaliser un exercice mathématique
- Connaître son résultat final

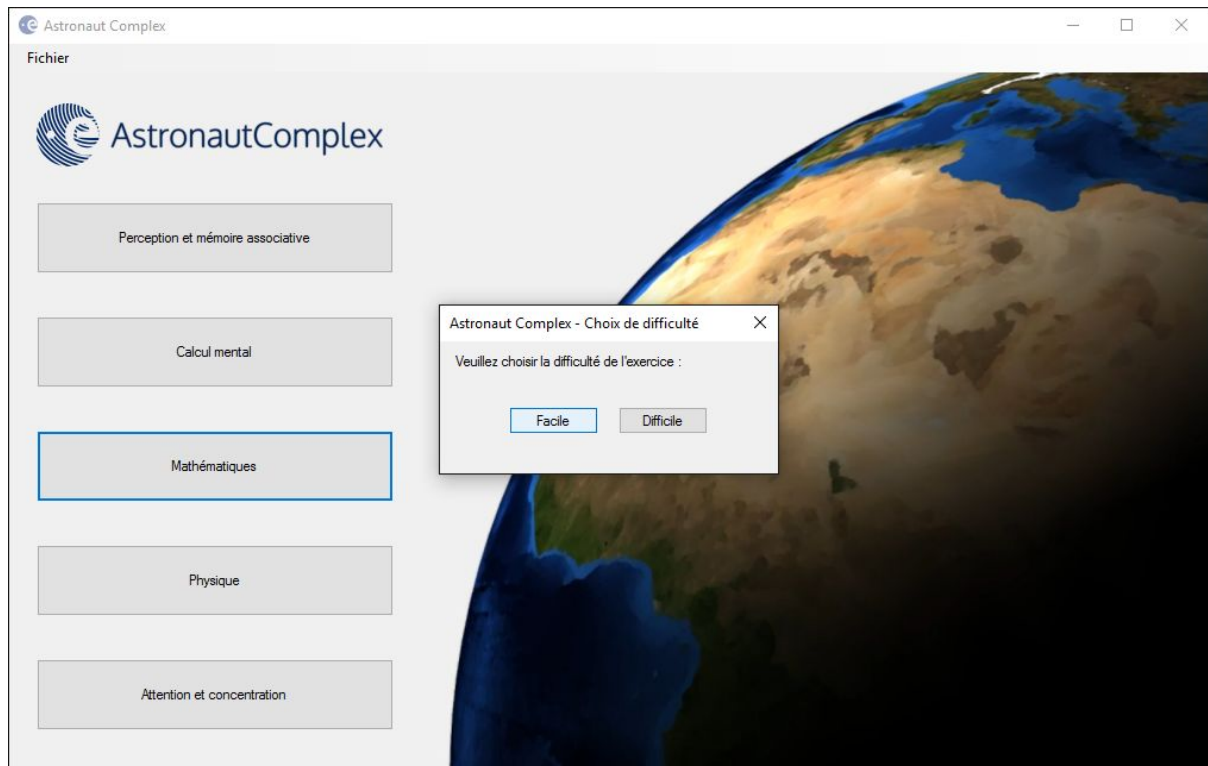
Le logo de l'application utilise celui de l'ESA (libre de droit) qui a été étendu par Thibaud en y ajoutant le nom "AstronautComplex".

Le fond d'écran du menu principal est disponible en deux versions : une animée (nécessitant Windows Media Player sur l'ordinateur exécutant le programme) et une statique (au cas où l'utilisateur ne le possède pas). L'introduction de cette fonctionnalité purement esthétique a demandé un travail de recherche pour comprendre comment utiliser des bibliothèques externes d'interopérabilité et aborder ce concept de manière superficielle. L'animation a été entièrement réalisée par Thibaud sous le logiciel Unity (utilisant une texture libre de droit qu'il a traité par la suite pour un rendu plus pastel), et montée par Corentin sous Blender Video Editing. Cependant, la lenteur au chargement et la difficulté de gestion d'erreurs entre ordinateurs équipés des bonnes bibliothèques et les autres nous ont poussés à commenter temporairement le code d'affichage de cette vidéo afin de garantir la stabilité du programme.

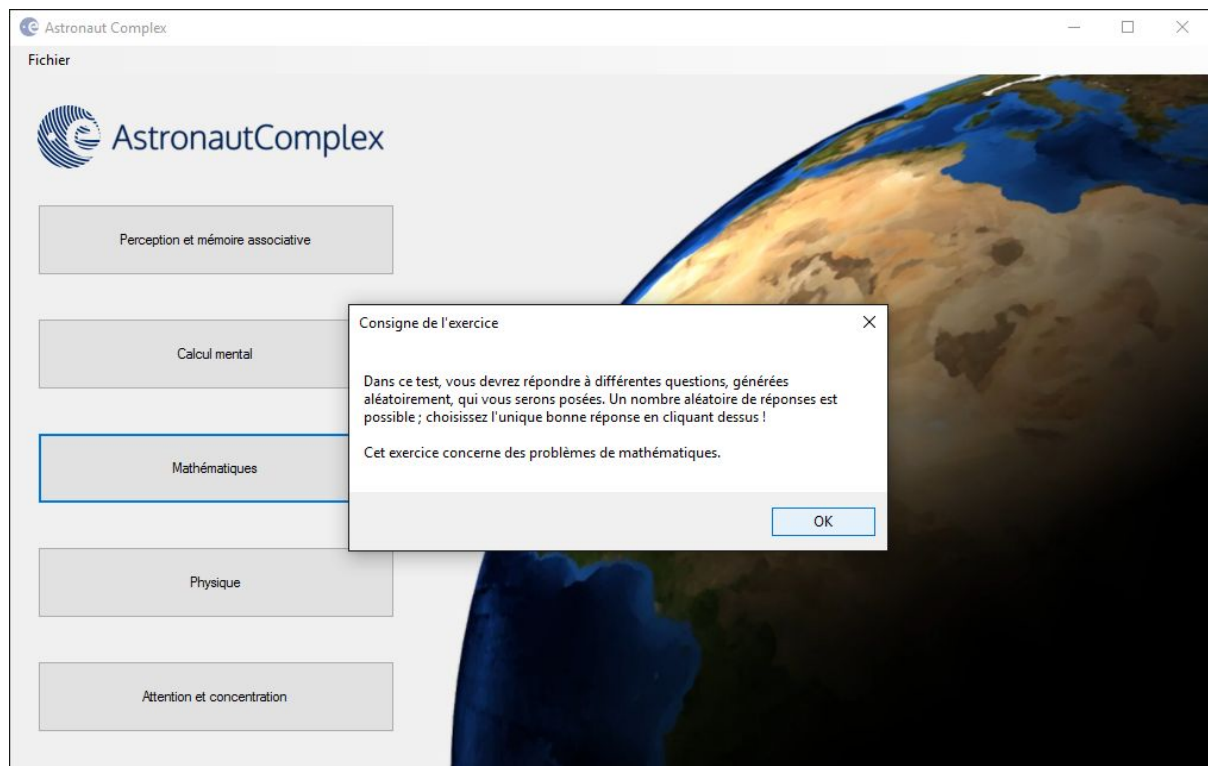
1. Menu principal



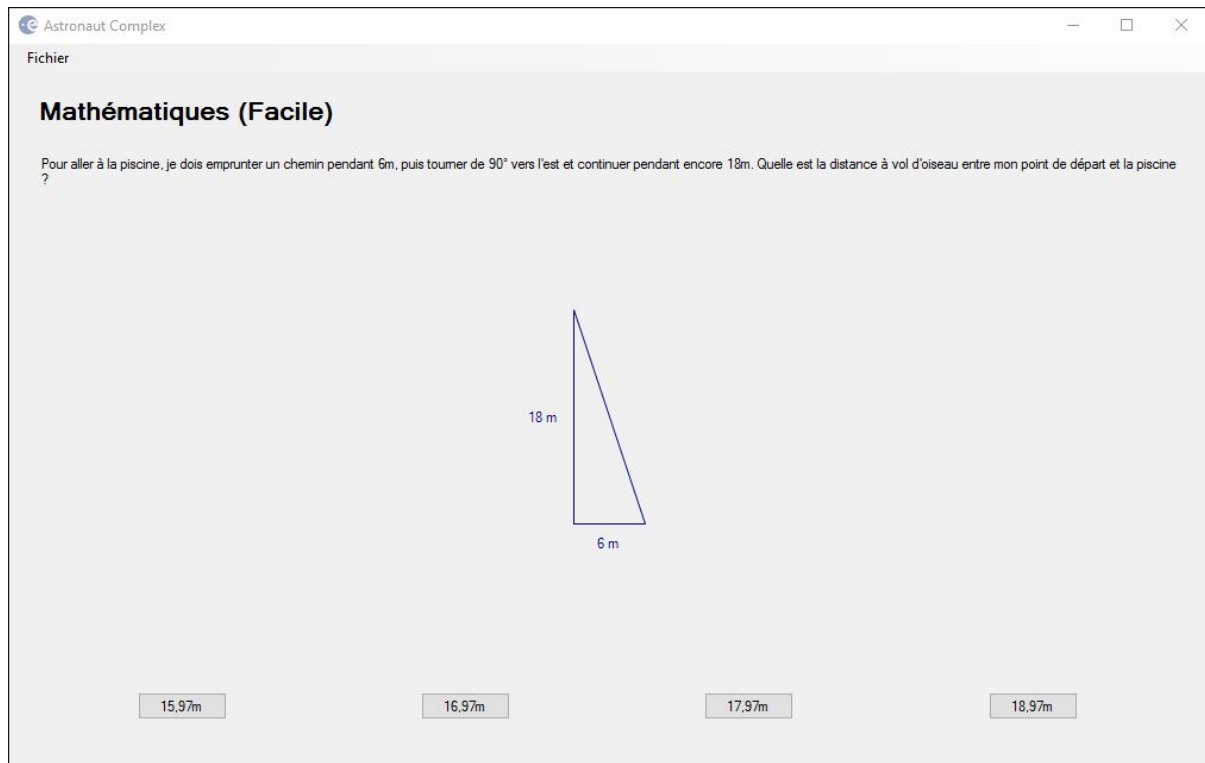
2. Sélection de l'exercice de Mathématiques et choix de difficulté



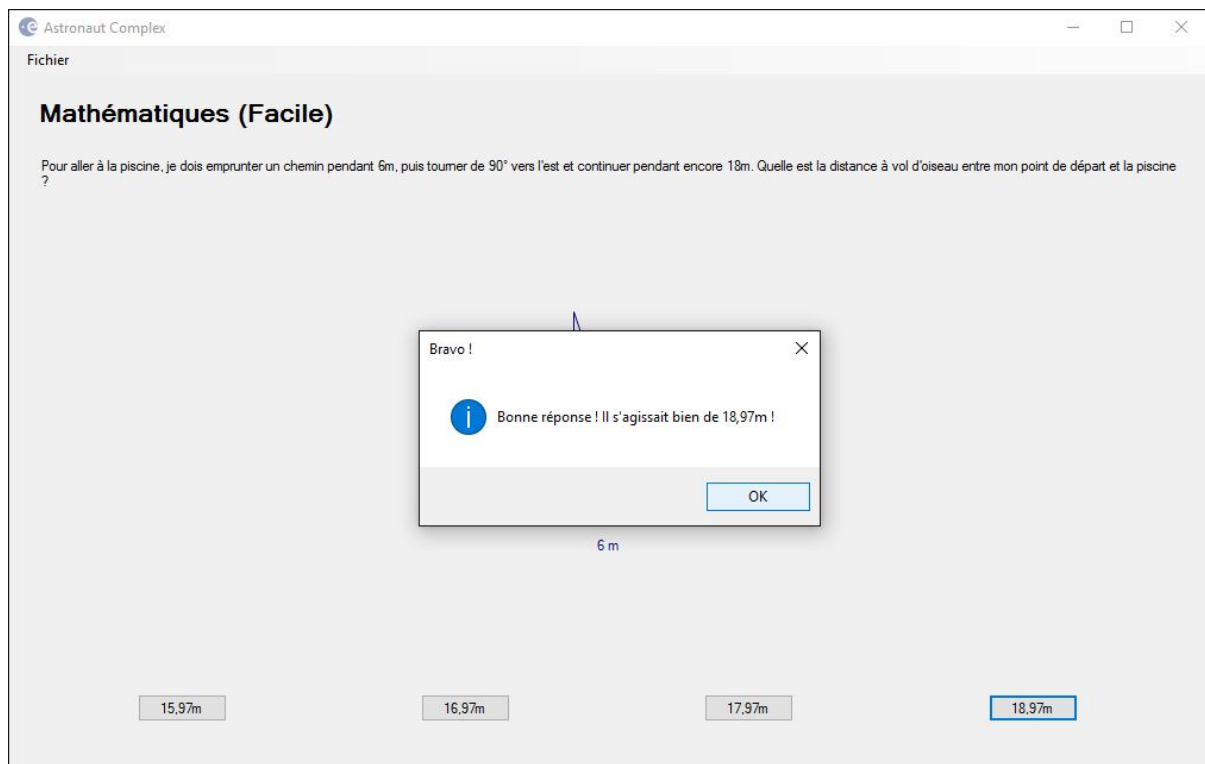
3. Affichage des instructions



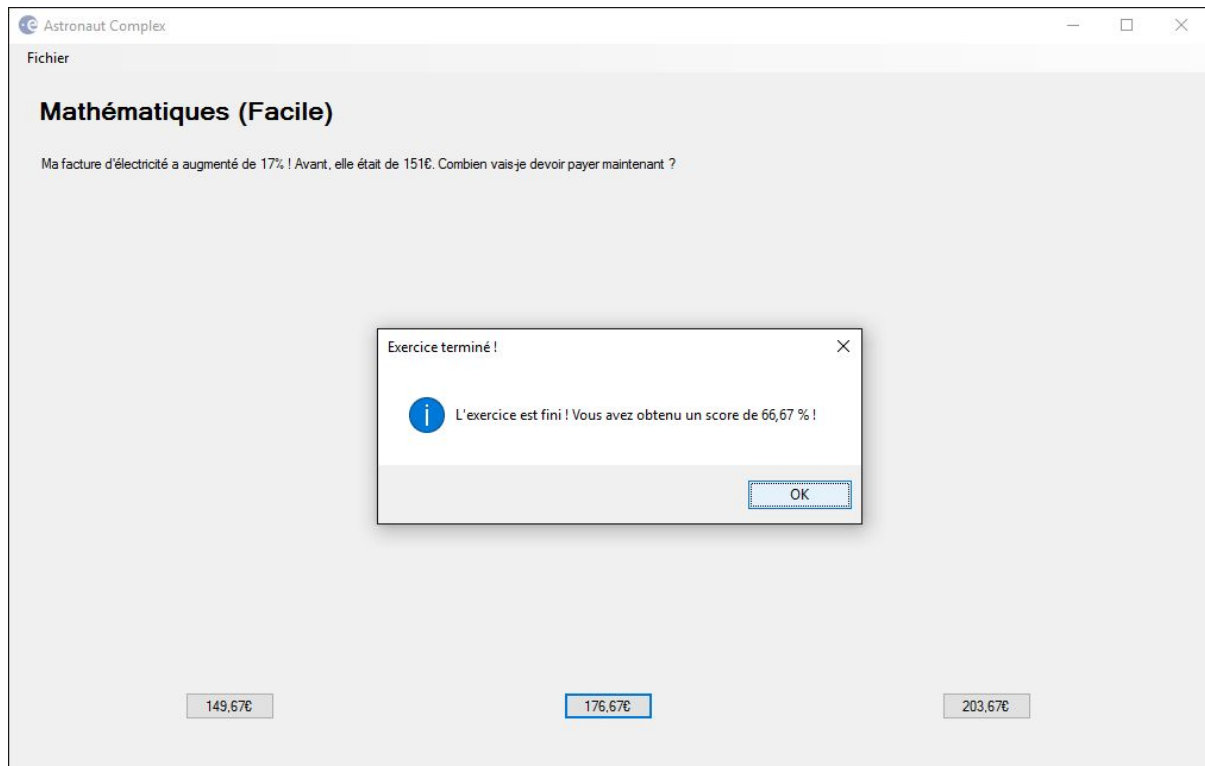
4. Affichage d'une question générée aléatoirement



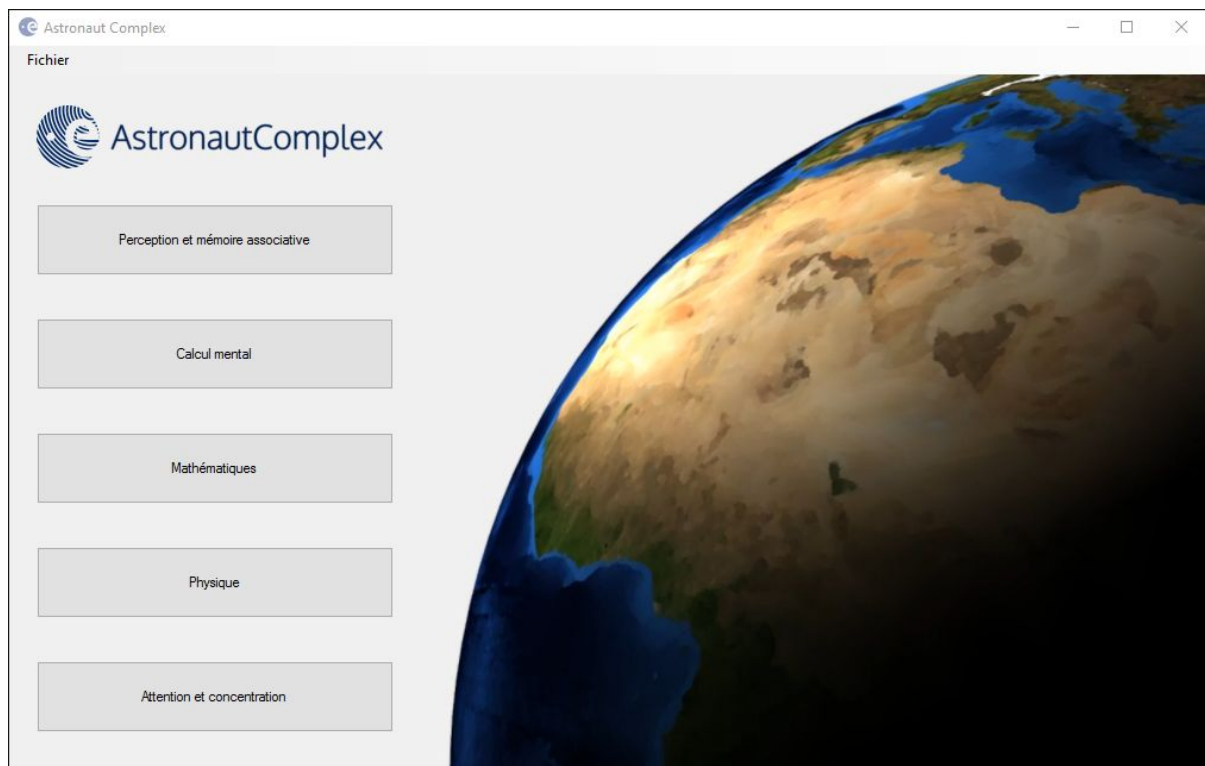
5. Affichage de la bonne réponse (en l'ayant trouvée)



6. Affichage du pourcentage de bonnes réponses (à la fin)



7. Retour au menu principal



B. Tests

En dehors des tests continus menés tout au long du développement, des tests unitaires et fonctionnels ont été développés à la fin du projet pour garantir le bon fonctionnement des classes de base. Notons qu'ils sont légèrement biaisés dans leur écriture par le développement qui a été fait avant ; il aurait peut-être été plus efficace de procéder à du développement piloté par test (DPT) en les réalisant dès le début et en axant le développement sur eux.

1. Protocoles

Les tests unitaires ont été réalisés avec l'aide de l'outil intégré à Visual Studio, par la création d'un 3e projet **AstronautComplexTests** contenant plusieurs classes, chacune préfixée du mot **Test_** et correspondant à une classe du projet. Chacune des fonctions de cette classe de test est nommée selon la fonction qu'elle teste, préfixée du mot **Test_** et du cas d'utilisation.

Par exemple, la classe **Test_ExerciseScore** contient les méthodes :

- **Test_NTotalAnswers_ToString** qui permet de tester le calcul du pourcentage de bonnes réponses d'un exercice avec un nombre total de réponses normal (supérieur à 1)
- **Test_0TotalAnswers_ToString** qui permet de tester le même calcul mais pour un nombre total de réponses de 0
- **Test_NotEnoughTotalAnswers_ToString** qui permet de tester le même calcul mais pour un nombre total de réponses inférieur au nombre de bonnes réponses

Le corps de ces fonctions est composé d'une instanciation de la classe à tester, des opérations nécessaires pour faire correspondre le cas d'utilisation, et enfin les résultats (grâce à, par exemple, une assertion).

À la génération du projet de tests, ceux-ci sont automatiquement vérifiés et les assertions non valides sont indiquées par l'outil.

2. Résultats

Tous les tests unitaires n'ont pas été écrits (un certain nombre de fonctions sont encore à tester unitairement parlant), mais tous les résultats obtenus sont positifs.

3. Tests fonctionnels

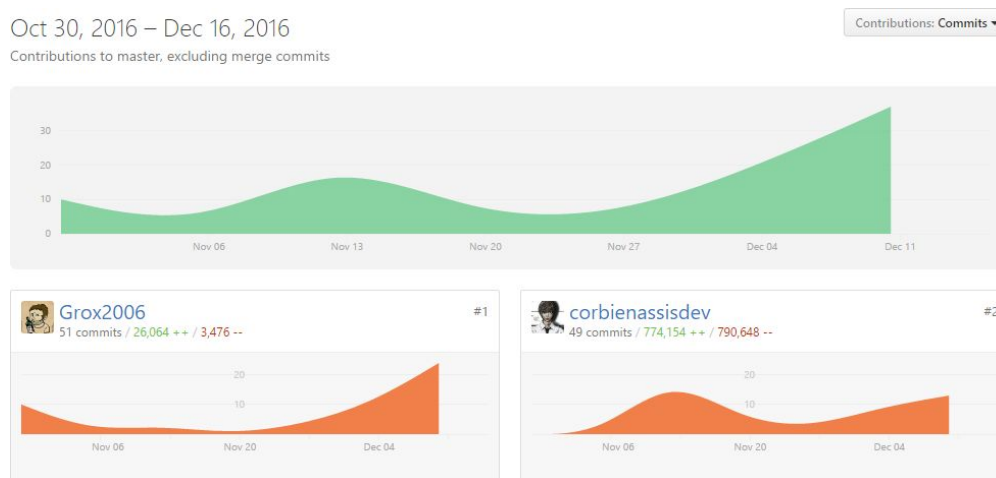
Afin de compléter les tests unitaires, testant des fonctions uniques et pas nécessairement l'agencement entre elles, des tests fonctionnels ont été menés et leur état inscrit dans la matrice des fonctionnalités disponible en **Annexe 1**.

Conclusion

A. Bilan

Le planning a pu être complété et toutes les fonctionnalités demandées sont présentes dans la solution. Un effort particulier a été donné sur les critères ergonomiques des exigences interface afin de fournir une application conviviale, simple et efficace. Le développement suivant un patron "moteur-plugin" a permis le développement en parallèle des différents exercices demandés. Les exercices de type "questions" ont été développés de manière à générer aléatoirement tout type de question pourvu qu'il soit défini dans une classe indépendante. Cependant, toutes les questions proposées n'ont pas été implémentées, et l'utilisation de la fonction Thread.Sleep dans **ExercicePerception** bloque l'utilisation du programme. Ce problème est réglable en utilisant d'autres fonctions, mais elles ne sont disponibles qu'en compilant la solution vers une cible de .NET Framework plus récente (version 4.5, pas disponible à l'ENSC).

L'expérience préalable du binôme a permis un développement fluide, aisé et sans problèmes majeurs pour les deux membres de l'équipe. Chacun a autant participé que l'autre, comme le prouve ce graphique représentant la contribution de chaque membre au projet (issu du site GitHub) :



Nous pouvons voir le travail également réparti mais hétérogène. En effet, alors que Thibaud a produit plus au début et à la fin (développement du moteur initial et production des derniers exercices), c'est Corentin qui a assuré le développement en milieu de projet. Le très haut nombre de lignes manipulées par Corentin correspond à l'ajout et la suppression d'une grosse partie du projet qu'il développait indépendamment, et qui est devenue **ExercicePerception** et **ExerciceFocus**.

B. Perspectives et évolutions

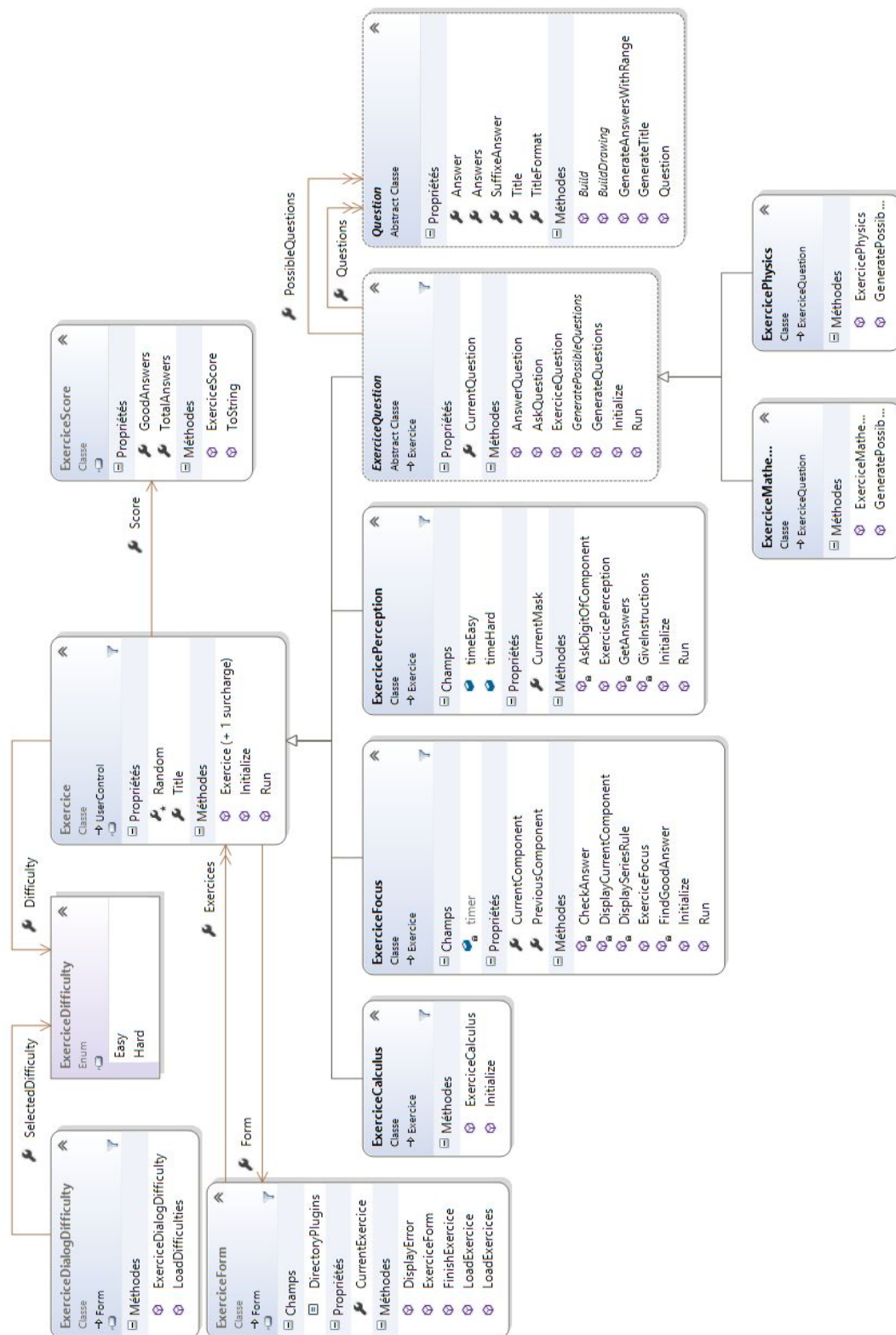
L'architecture de la solution étant ouverte et disponible sur GitHub, le développement de nouveaux exercices ou l'ajout de nouvelles possibilités aux existants (par exemple, ajouter des questions aux exercices de type "question") se fait par simple héritage de fonctions abstraites ; cela nécessite tout de même des connaissances, même superficielles, de développement en C#. Enfin, il est envisageable d'étendre l'utilisation de l'application à d'autres champs que la sélection d'astronautes : par exemple, des tests médicaux sur patients atteint de soucis cognitifs cliniques.

Annexe 1 : Matrice des fonctionnalités

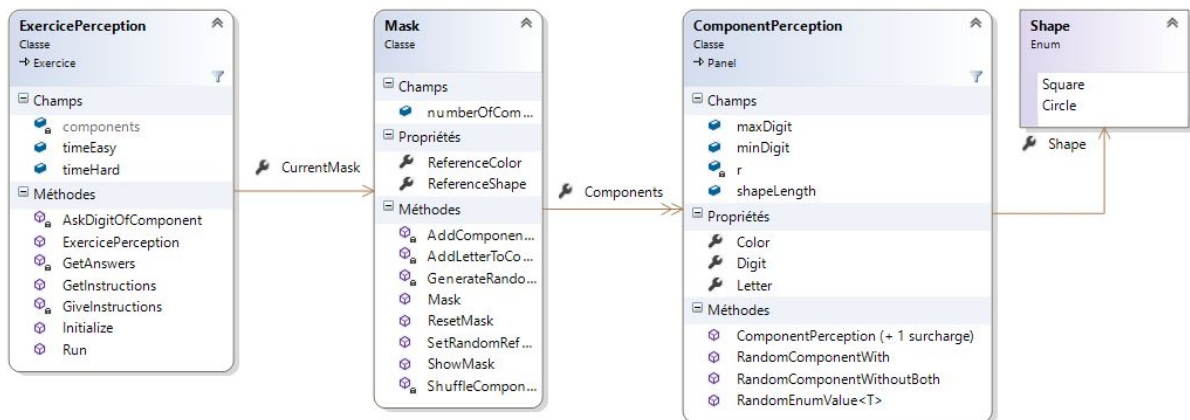
Catégorie	Fonctionnalité	Avancement	Tests unitaires	Test fonctionnel
Général	[AC_G_01]	100%		Validé
	[AC_G_02]	100%		Validé
	[AC_G_03]	100%		Validé
	[AC_G_04]	100%	Validé	Validé
Exercice de perception	[AC_TP_01]	100%		Validé
	[AC_TP_02]	100%	Validé	Validé
	[AC_TP_03]	100%	Validé	Validé
	[AC_TP_04]	100%		Validé
	[AC_TP_05]	100%		Validé
	[AC_TP_06]	100%		Validé
Exercice d'attention	[AC_TA_01]	100%		Validé
	[AC_TA_02]	100%		Validé
	[AC_TA_03]	100%		Validé
	[AC_TA_04]	100%		Validé
	[AC_TA_05]	100%		Validé
	[AC_TA_06]	100%		Validé
	[AC_TA_07]	100%		Validé
	[AC_TA_08]	100%		Validé
	[AC_TA_09]	100%		Validé
Exercice de calcul	[AC_TC_01]	100%		Validé
	[AC_TC_02]	100%		Validé
	[AC_TC_03]	100%		Validé
	[AC_TC_04]	80%		Validé
	[AC_TC_05]	100%		Validé
	[AC_TC_06]	100%		Validé
	[AC_TC_07]	100%		Validé
Exercice de questions	[AC_TQ_01]	100%		Validé
	[AC_TQ_02]	100%		Validé
	[AC_TQ_03]	100%		Validé
	[AC_TQ_04]	100%		Validé
	[AC_TQ_05]	100%		Validé
	[AC_TQ_06]	100%		Validé
	[AC_TQ_07]	100%		Validé

Annexe 2 : Diagrammes de classes

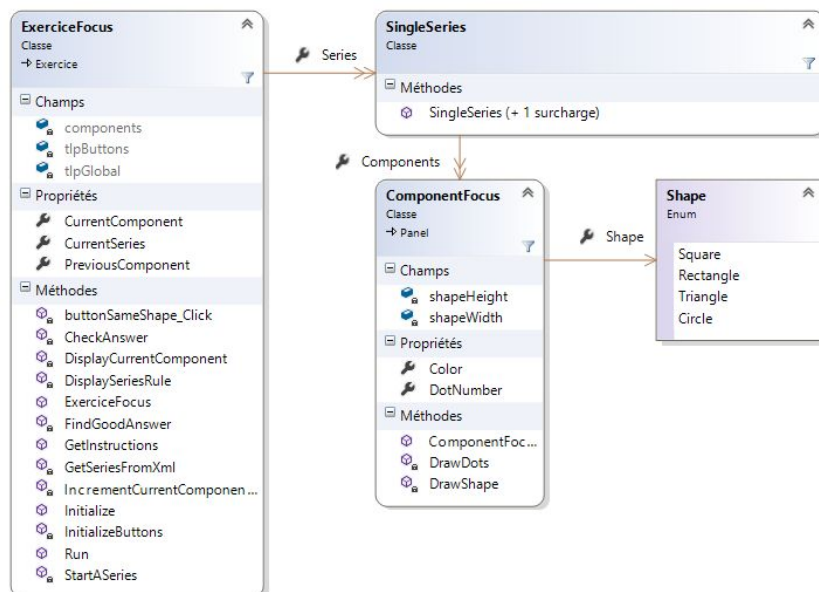
AstronautComplex et AstronautComplexBasicPack



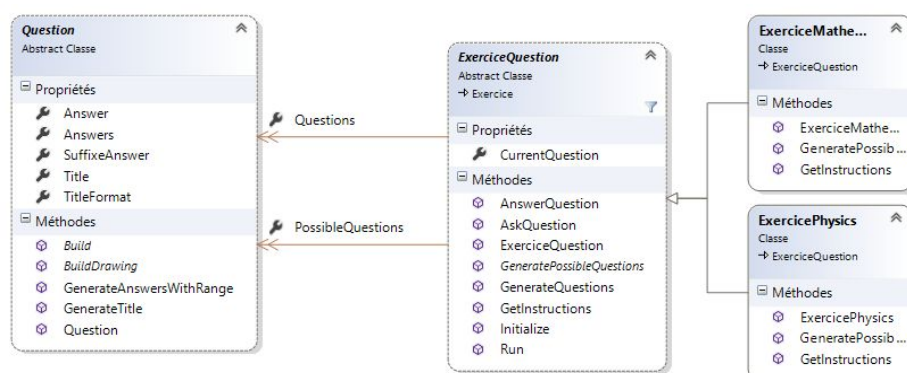
ExercicePerception



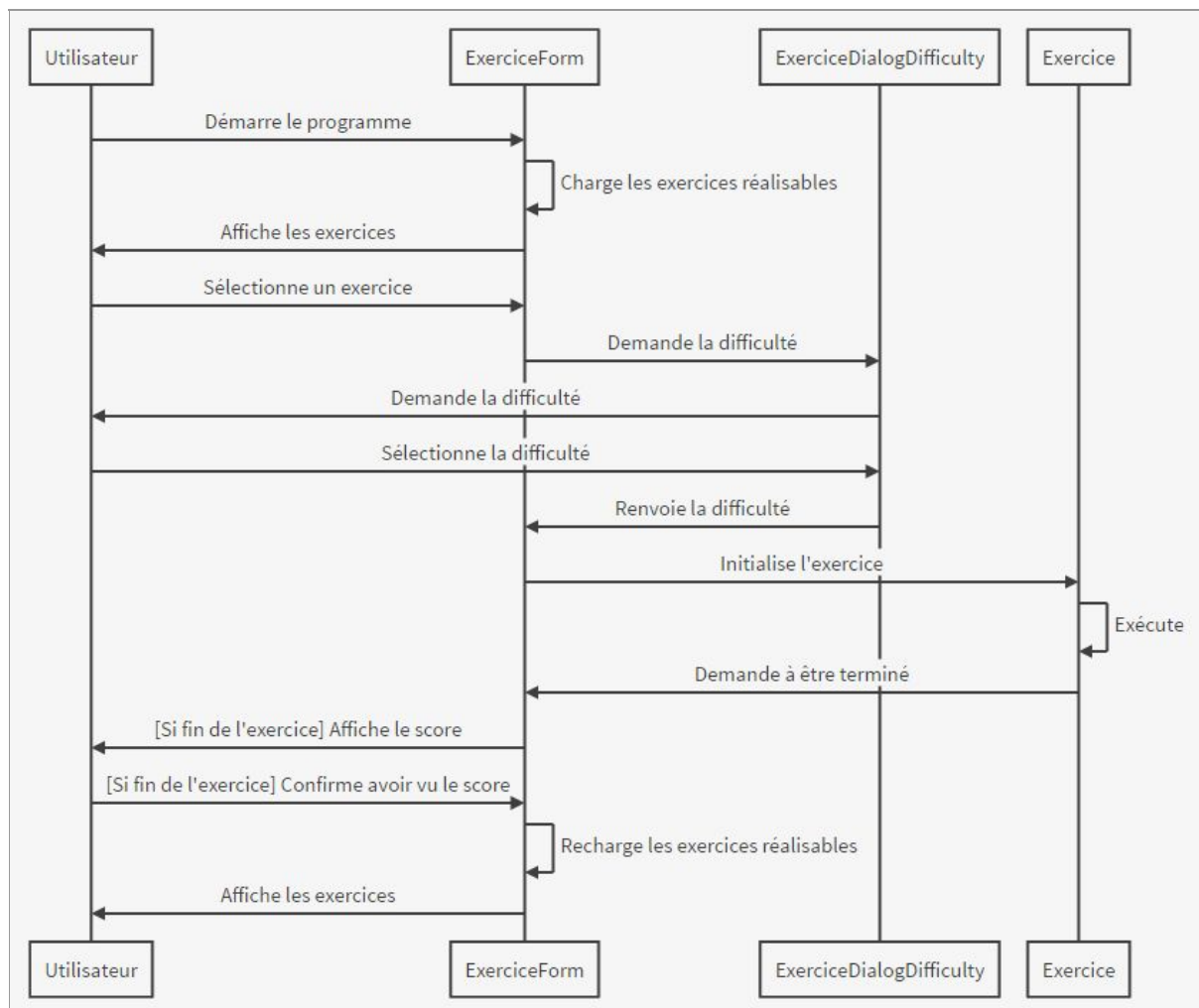
ExerciceFocus



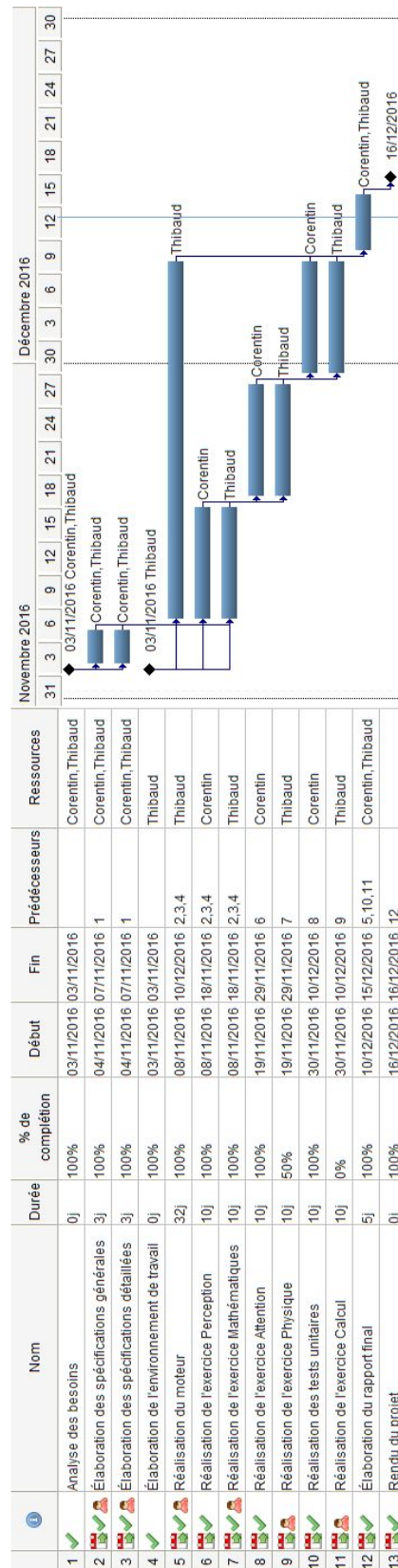
ExerciceQuestion



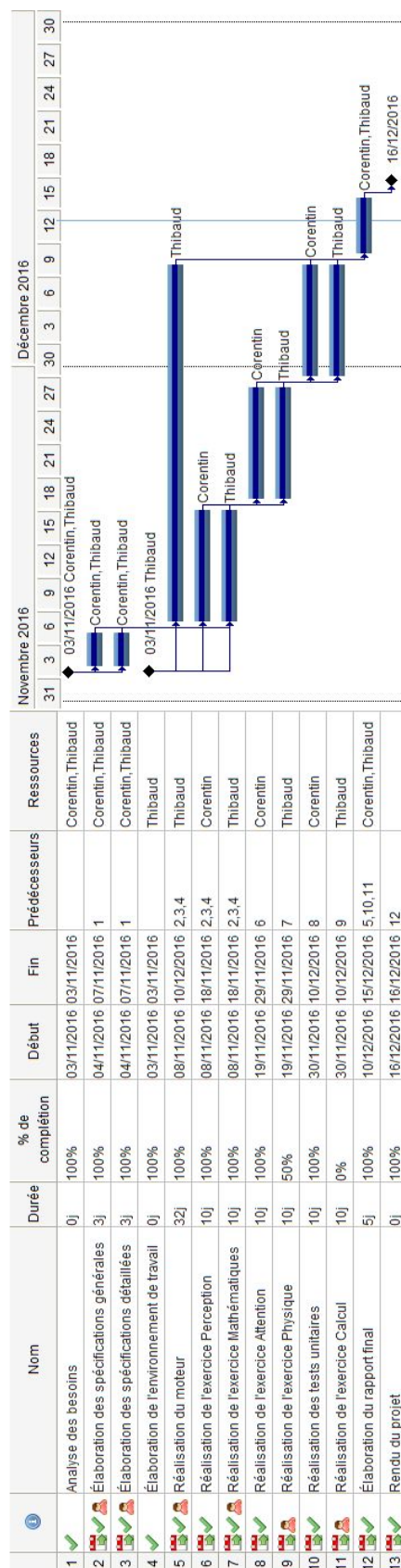
Annexe 3 : Diagramme de séquences



Annexe 4 : Planning prévisionnel

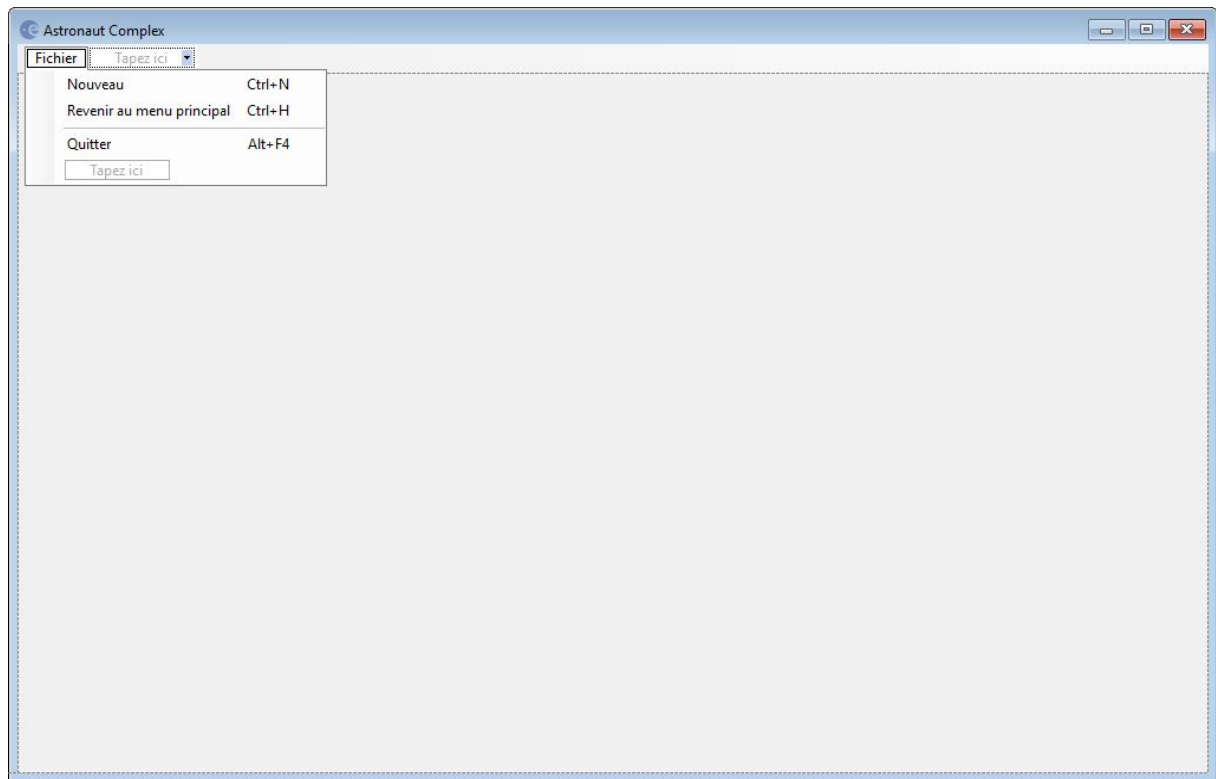


Annexe 5 : Planning final

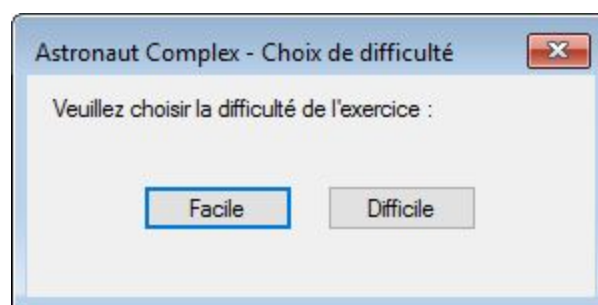


Annexe 6 : Maquette

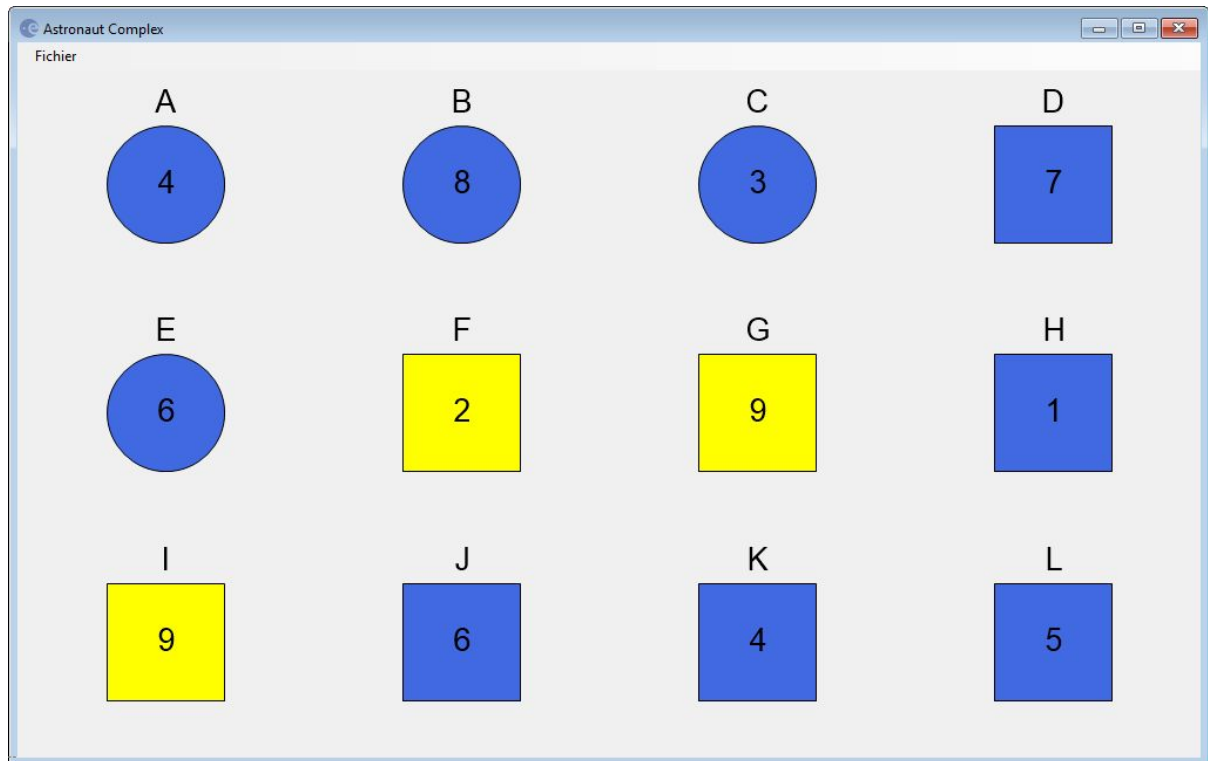
ExerciceForm



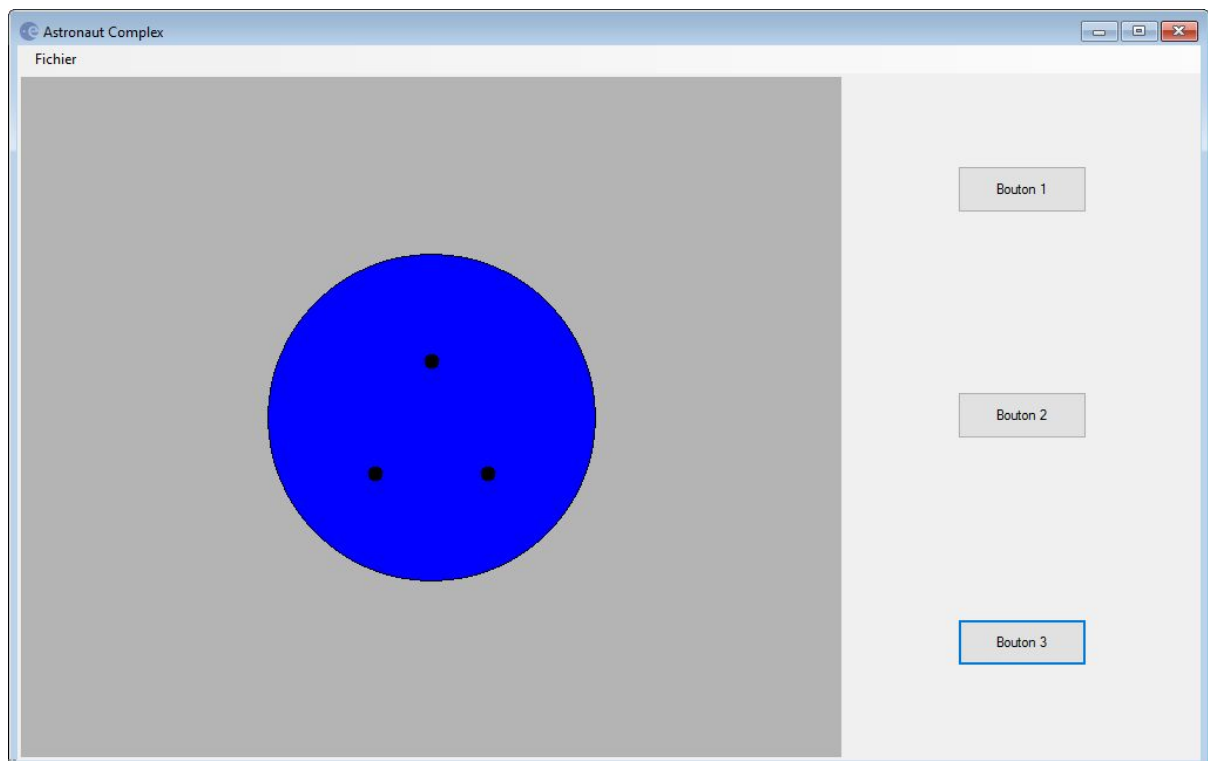
ExerciceDialogDifficulty



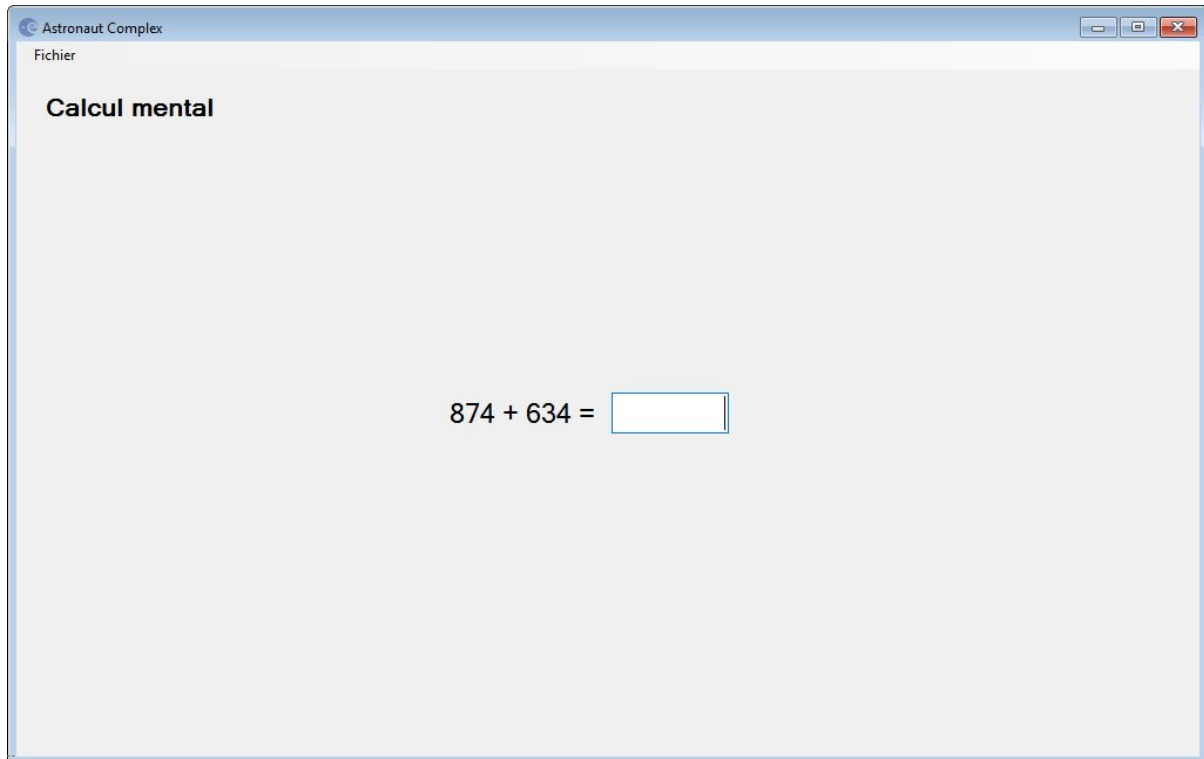
ExercicePerception



ExerciceFocus



Exercice Calculus



The screenshot shows a window titled "Astronaut Complex" with a menu bar containing "Fichier". The main content area is titled "Calcul mental" and displays the equation $874 + 634 =$ followed by a text input field for the user's answer.

Exercice Question



The screenshot shows a window titled "Astronaut Complex" with a menu bar containing "Fichier". The main content area is titled "Mathématiques (Facile)" and contains the text: "Ma facture d'électricité a augmenté de 5% ! Avant, elle était de 63€. Combien vais-je devoir payer maintenant ?". At the bottom of the window, there are three radio button options: "59,15€", "66,15€", and "73,15€".