



Rapport du Projet Génie Logiciel



Emile NETTER

Thomas CASADO

Introduction

Ce projet de génie logiciel consiste à implémenter un programme de tests semblable à celui utilisé par l'agence spatiale européenne pour la sélection des astronautes. Ce projet sera développé en utilisant la programmation événementielle avec les Windows Forms. Cette technologie permet de réaliser facilement des interfaces graphiques Homme-Machine et gère notamment les interactions riches avec l'utilisateur. Le langage de programmation utilisé sera le C#.

Ce rapport traitera de notre démarche, de nos choix de conception, de l'architecture de l'application et des tests que nous avons réalisés afin de valider le bon fonctionnement de notre application.

1. Spécifications générales

Analyse fonctionnelle

Cette partie traite de l'analyse du besoin liée au cahier des charges. Chaque formulaire de test correspond à un tableau qui regroupe les fonctionnalités attendues.

Légende :

FP : fonction principale

FC : fonction contrainte

FCom : fonction complémentaire

Fonctionnalité	Expression
FP 1	Doit permettre à l'utilisateur d'effectuer un test de perception
FC1.1	Doit afficher une règle
FC1.2	Doit afficher 12 formes différentes
FC1.2.1	Doit être un rond ou un carré
FC1.2.2	Doit être bleu ou jaune
FC1.3	Doit afficher 1 chiffre dans chaque forme
FC1.4	Doit permettre à l'utilisateur de saisir des valeurs
FC1.5	Doit effectuer 10 tests
FC1.6	Doit compter les bonnes et mauvaises réponses

Tableau 1 - Test de perception

Fonctionnalité	Expression
FP 2	Doit permettre à l'utilisateur d'effectuer un test d'attention
FC2.1	Doit afficher une règle
FC2.2	Doit afficher une forme
FC2.2.1	Doit être un rond un carré ou un triangle
FC2.2.2	Doit être bleu, jaune ou rouge
FC2.3	Doit afficher entre 0 et 4 points dans une forme
FC2.4	Doit afficher 5 écrans consécutif
FC2.5	Doit afficher 1 bouton sur l'écran 1
FC2.5.1	Doit afficher le bouton 3 sur l'écran 1
FC2.6	Doit afficher 3 boutons sur les autres écrans
FC2.7	Doit effectuer 3 séries

Tableau 2 - Test d'attention

Fonctionnalité	Expression
FP 3	Doit permettre à l'utilisateur d'effectuer un test de calcul mental
FP 3.1	Doit permettre à l'utilisateur de choisir parmi les 4 additions
FP 3.1.1	Doit permettre d'effectuer une addition
FC 3.1.1	Les opérandes doivent avoir 3 chiffres chacun
FP 3.1.2	Doit permettre d'effectuer une soustraction
FC 3.1.2	Le premier nombre doit avoir 3 chiffres et être toujours supérieur au deuxième qui lui peut comporter entre 2 et 3 chiffres.
FP 3.1.3	Doit permettre d'effectuer une multiplication
FC 3.1.3	Le premier nombre peut avoir entre 1 et 2 chiffres et le second 1 seul chiffre.
FP 3.1.4	Doit permettre d'effectuer une division
FC 3.1.4	Le dividende doit avoir 2 ou 3 chiffres et le diviseur 1 seul chiffre.
FC 3.1.5	La division doit être entière

FC 3.1.6	Si le résultat n'est pas un nombre entier, l'utilisateur doit trouver le résultat entier arrondi au plus proche
FC 3.2	Doit procéder à 10 tests pour un opérateur donné
FC 3.3	Doit indiquer le pourcentage de bonnes réponses
FC 3.4	Doit afficher la réponse pendant quelques secondes en cas d'erreur
FC 3.5	Doit afficher "OK" en cas de bonne réponse

Tableau 3 - Test de calcul mental

Fonctionnalité	Expression
FP 4	Doit permettre à l'utilisateur d'effectuer une série de 10 problèmes mathématiques
FC 4.1	Le texte de la réponse ne doit pas dépasser 4 lignes
FC 4.2	La réponse doit être de type QCM
FC 4.3	Doit choisir les tests de manière aléatoire
FC 4.4	Ne doit pas faire apparaître 2 fois le même problème sur une série de 10
FC 4.5	Les tests doivent être différenciés en fonction du niveau de difficulté
FCom 4.1	Doit pouvoir afficher un dessin à côté du texte

Tableau 4 - Test de mathématiques

Fonctionnalité	Expression
FP 5	Doit permettre à l'utilisateur d'effectuer une série de 10 problèmes de physique
FC 5.1	Le texte de la réponse ne doit pas dépasser 4 lignes
FC 5.2	La réponse doit être de type QCM
FC 5.3	Doit choisir les tests de manière aléatoire
FC 5.4	Ne doit pas faire apparaître 2 fois le même problème sur une série de 10
FC 5.5	Les tests doivent être différenciés en fonction du niveau de difficulté
FCom 5.1	Doit pouvoir afficher un dessin à côté du texte

Tableau 5 - Test de physique

Fonctionnalité	Expression
FP 6	Doit permettre à l'utilisateur de choisir parmi les 5 tests
FC 6.1	Doit permettre à l'utilisateur de choisir le niveau de difficulté entre facile et difficile.

Analyse des besoins pour l'interface avec l'utilisateur

L'interface de l'application doit être sobre et intuitive afin que l'utilisateur soit concentré sur les tests qu'il doit passer. En effet, cette application sera utilisée pour sélectionner les astronautes susceptibles de rejoindre l'Agence Spatiale Européenne. Si l'application est trop confuse, le candidat ne sera pas dans les bonnes conditions pour passer les différents tests. Tests qui seront d'autant plus compliqués à passer si l'interface n'est pas ergonomique. Pour faciliter la tâche du candidat, chaque test sera précédé d'un message affiché à l'écran expliquant le déroulement du test ainsi que d'une démonstration simple. Enfin, le candidat doit pouvoir retourner sur le menu principal à tout moment.

Architecture de l'application

Notre application est divisée en plusieurs formulaires. Le formulaire principal (MainMenu) permet d'accéder aux différents tests. Ensuite, chaque test a son propre formulaire. Il existe néanmoins des cas où un test ne correspond pas forcément à un seul formulaire. C'est le cas pour le test de calcul mental, de perception ainsi que les problèmes de mathématiques et de physique. Les cas seront détaillés ci-après en fonction du test.

Du point de vue de l'architecture logiciel, nous avons essayé de suivre un patron logiciel qui nous avait été présenté en cours à savoir le Model - View - Presenter. Ce modèle nous paraissait le plus approprié pour une application tournant avec des Windows Forms. Nous avons bien compris le principe de ce modèle qui permet de différencier la partie graphique (affichages, boutons etc.) et le code (fonctions, méthodes...). Cependant, nous n'avons pas réussi à mettre en place la partie Presenter (qui doit faire le lien entre la vue et le modèle) malgré de nombreuses recherches et l'exemple qui était disponible sur github. Notre code est, de manière générale, quand même plus lisible et plus facile à maintenir que si nous n'avions pas utilisé ce patron logiciel. Ainsi, il y a d'un côté les formulaires qui permettent d'afficher les tests et de l'autre côté les classes associées à ces formulaires et qui portent généralement le même nom que le formulaire en question. Dans ces classes sont créées toutes les méthodes utiles pour afficher et envoyer les données vers la vue. La vue les récupère, procède aux différents affichages puis renvoie si nécessaire des données (résultats du test par exemple).

D'autre part, nous n'avons pas voulu implémenter plus de fonctionnalités que nécessaires. Notre objectif était avant tout de produire une solution fonctionnelle, agréable et facile à utiliser. Le but étant plutôt de produire le code le plus clair possible. Notre code est donc orienté sur le principe du Keep It Simple. De plus, nous avons veillé à ce que le code soit le plus optimisé possible, et donc cherché à alléger le code, à ne pas nous répéter. Nous nous sommes orientés sur le principe Dont Repeat Yourself. Le meilleur exemple est la création d'une classe forme avec 2 constructeurs

différents(un pour une forme correspondante à l'exercice Perception, l'autre pour l'exercice Concentration) de façon à n'avoir qu'une seule fonction de création de forme pour ces 2 exercices. Globalement, sur l'ensemble de l'application nous avons veillé à ce qu'aucune répétition inutile de code ne soit présente dans notre code.

Enfin, toutes nos bases de données de questions et règles sont stockées dans des fichiers XML pour assurer une plus grande interopérabilité si par exemple l'ESA souhaite changer ses banques de données. De plus, le code est plus aéré qu'avec des pages et des pages de problèmes de mathématiques et de physique. Cette modification de base de données de problèmes est expliquée dans le manuel de fonctionnement, section manuel administrateur.

Voir le diagramme de classe et les maquettes en annexe.

Répartition des tâches et planning

La date de début du projet est le 14 Novembre et le rendu est prévu le 16 Décembre. La première partie du projet est surtout une phase de conception et de maquettage. Nous avons réfléchi ensemble à l'architecture du logiciel et aux différentes classes que nous allions implémenter (cette phase a duré environ une semaine). Les maquettes ont permis de nous donner un aperçu de l'interface finale.

Après avoir fait la conception de l'application nous avons pu passer à la production du code source. La fonctionnalité de création de classes de Visual Studio nous a permis de gagner du temps puisqu'elle permet de générer automatiquement nos classes, méthodes et attributs. La partie programmation devait durer une quinzaine de jours. Certains tests ont été beaucoup plus faciles à réaliser que d'autres, notamment ceux ne nécessitant pas la génération de formes. Ainsi, une personne s'est mise au développement des tests avec formes et l'autre sur ceux restants. Des points étaient régulièrement faits pour suivre l'avancement et l'utilisation de git nous a permis de travailler chacun de notre côté plus sereinement. Les derniers jours ont permis d'harmoniser le code, de travailler la partie graphique de notre application et de finir le rapport. L'écriture du rapport s'est aussi faite en binôme, chacun était chargé d'écrire une partie.

Par rapport au planning initial, nous avons surtout pris du retard sur la phase de conception de l'application. Cette phase a demandé beaucoup de réflexion et nous avons dû revenir plusieurs fois sur des points qui ne nous convenaient pas ou qui étaient trop durs à implémenter. Ce retard a ensuite été rattrapé ou du moins compensé par phase de programmation. Les diagrammes de gantt ci-dessous illustrent notre gestion du temps durant ce projet.

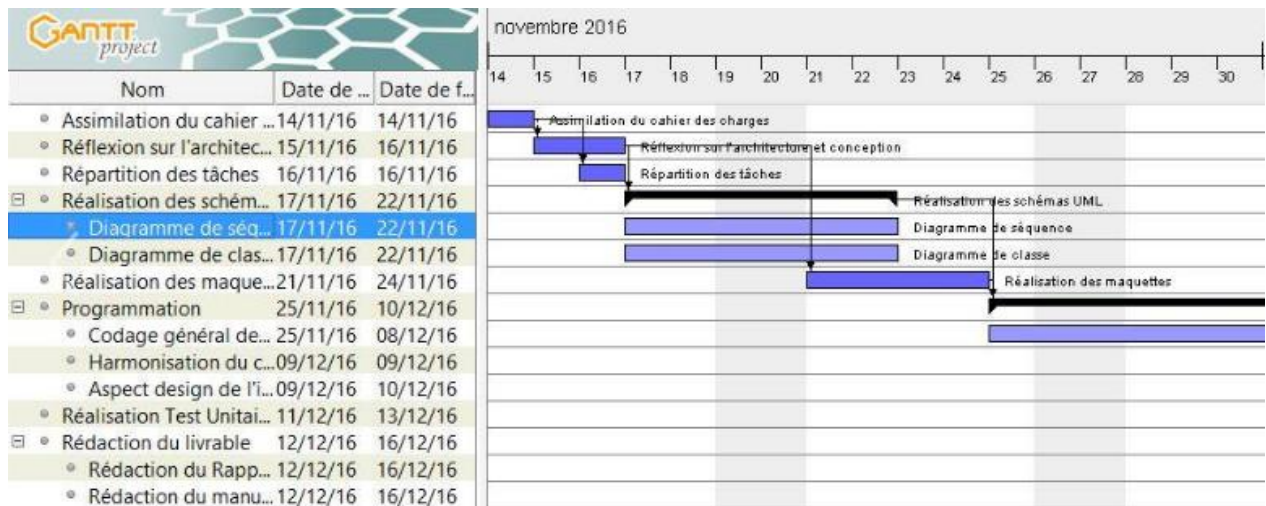


Figure 1 Planning prévisionnel

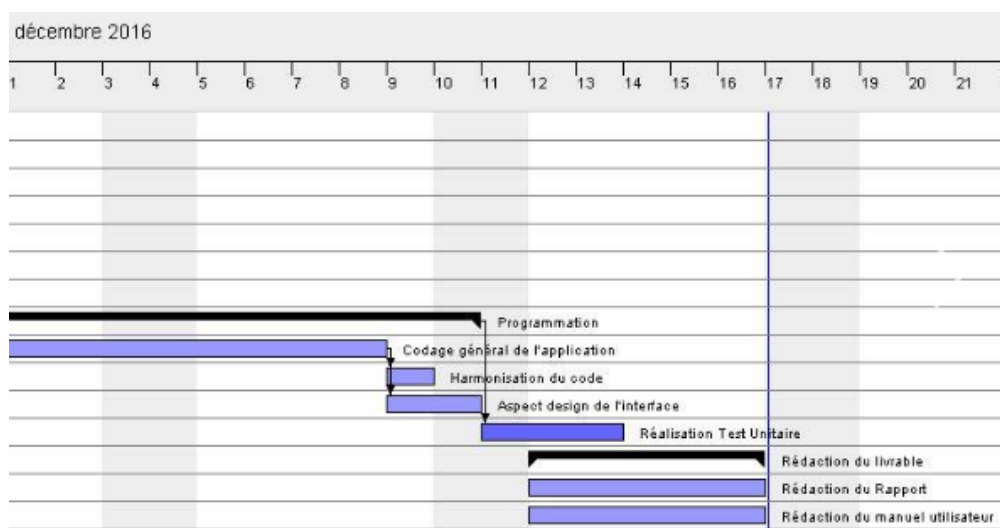


Figure 2 Planning prévisionnel (suite)

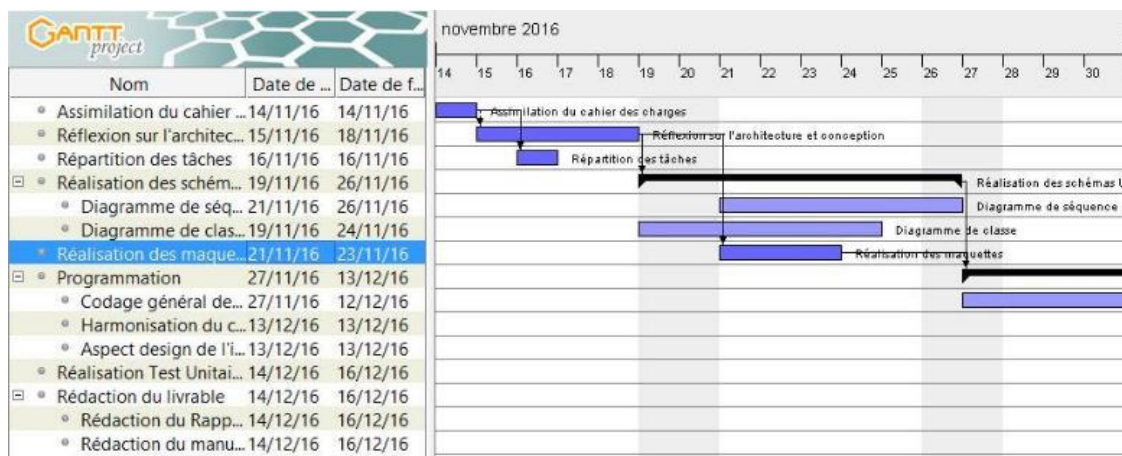


Figure 3 Planning effectif

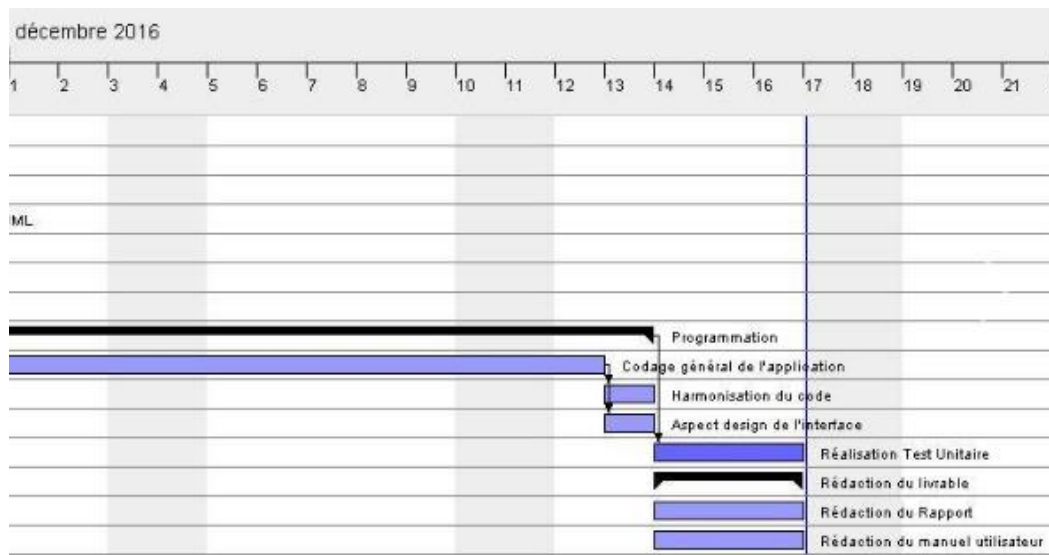


Figure 4 - Planning effectif (suite)

2. Spécifications détaillées

Nous allons ici parler plus en détails de l'implémentation de chaque formulaire et de la logique algorithmique que nous avons mise en place.

MainMenu :

Figure 5 - Menu principal

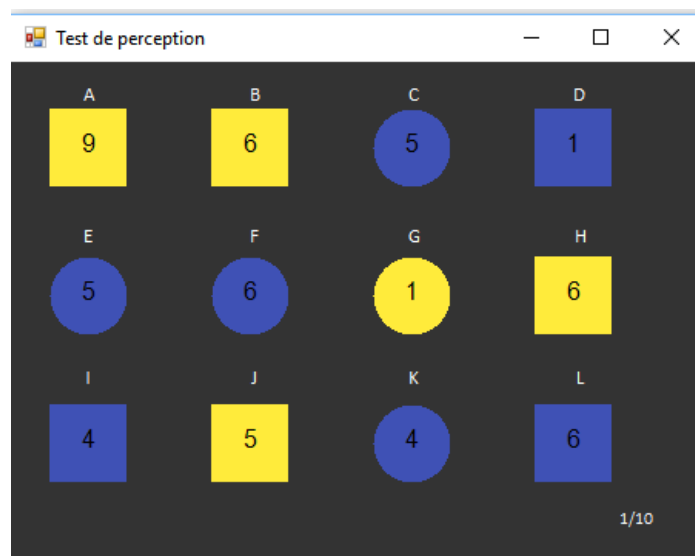
Le menu principal permet d'accéder aux tests. On peut également choisir la difficulté grâce à deux "radio buttons". Par défaut la difficulté est réglée sur Facile. L'utilisateur peut également entrer un nom s'il souhaite sauvegarder ses résultats. Cette option est facultative. Enfin, après chaque test

effectué, le pourcentage de bonnes réponses vient s'afficher à côté du test. Cela permet à l'utilisateur d'avoir un aperçu de son score quand il retourne sur le menu principal.

L'interface a été simplifiée au maximum afin de faciliter l'utilisation du logiciel. Comme on peut le voir sur la capture d'écran, l'application n'occupe pas la totalité de l'écran. Nous avons fait ce choix car les tests ne demandent pas beaucoup de place pour les réaliser et une fenêtre prenant toute la taille de l'écran aurait sûrement paru vide. Ici l'information est concentrée au milieu de l'écran ce qui facilite la concentration.

Le menu principal est le noyau de l'application. En effet, les tests ne peuvent être lancés que depuis ce menu. Une méthode commune à tous les boutons a été mise en place afin de lancer le test correspondant au bouton que l'on a cliqué. Une fois le test lancé, un nouveau formulaire apparaît. Ce formulaire n'interagit que très peu avec les autres : il se contente de récupérer les résultats des tests stockés dans la classe Sauvegarde.cs afin de les afficher. Un timer (1seconde) permet de rafraîchir l'affichage du menu afin de chercher constamment s'il y a de nouveau résultat à afficher. De plus, un timer de 10 secondes a été mis en place: ce timer permet la saisie du nom de l'utilisateur. Au-delà de ce temps, l'interface est de nouveau rafraîchie.

FormPerception



A	B	C	D
9	6	5	1
E	F	G	H
5	6	1	6
I	J	K	L
4	5	4	6

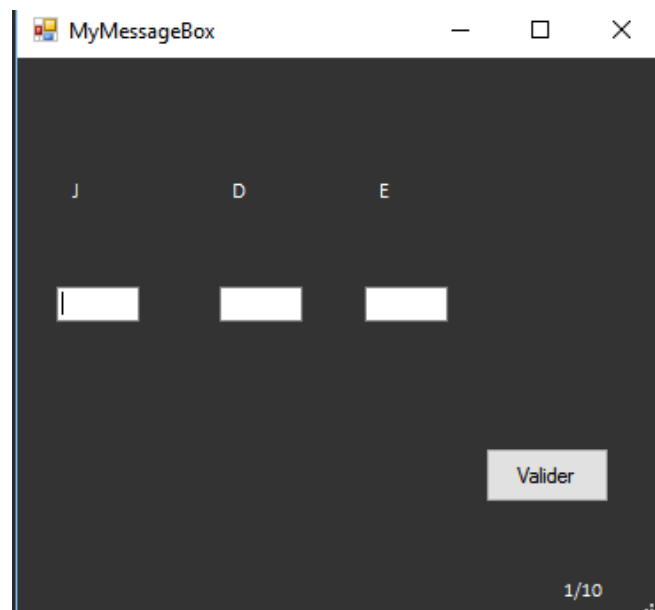
1/10

Ce formulaire correspond au test de perception. Le formulaire de base ne comporte que les lettres A,B,C...,L qui sont placées manuellement. Un compteur en bas à droite indique le nombre de séries à effectuer et permet de savoir à quel test on se trouve.

Ce test se décompose en trois temps. Dans un premier temps, lorsque l'utilisateur clique sur le bouton pour lancer le test, un messageBox fait apparaître la consigne. Cette consigne est récupérée depuis un fichier XML puis générée aléatoirement grâce à la méthode selectionRegle() de la classe Perception.cs. Ensuite, le formulaire apparaît comme sur la capture. Les formes et les chiffres à l'intérieur sont aussi générés de manière aléatoire. Pour les chiffres, on génère un tableau [3,4] dans

lequel on y ajoute des chiffres aléatoires. Pour les formes, on crée aussi un tableau [3,4] d'objets de type `Forme` qui possèdent différents attributs (couleur, forme, position en x, position en y...). On insère dans un premier temps les formes correspondantes à la règle (à des positions [i,j] aléatoires et au nombre de 3 ou 4), puis on remplit le reste du tableau avec d'autres formes, grâce à la méthode `selectionForme` de la classe `Perception.cs`. On fait ensuite appelle à la méthode `OnPaint` afin de dessiner les formes puis les chiffres grâce aux méthodes `Forme.creationFormeColorée(Forme f, Form form)` et `Perception.dessineNombres(PaintEventArgs e)`. A noter que les chiffres doivent être dessinés après l'affichage des formes car sinon elles les recouvrent. Le choix de tout générer aléatoirement nous a demandé plus de réflexion que si nous avions utilisé une base de données mais cela rend l'application plus réelle et le candidat ne pourra jamais "apprendre" les solutions de chaque grille s'il relance le test de nombreuses fois. Un timer permet de modifier en fonction de la difficulté choisie le temps de mémorisation des formes. A la fin du temps (2 ou 4 secondes) la dernière étape se produit. Un autre formulaire apparaît permettant de saisir les chiffres correspondant aux lettres. Ce formulaire s'appelle `MyMessageBox` et nous allons en parler juste après.

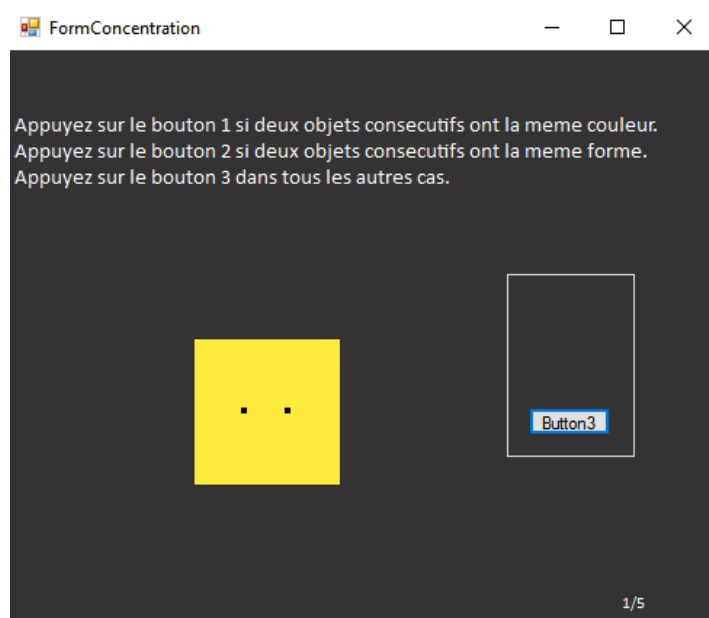
MyMessageBox



Nous voulions au départ utiliser un `messageBox` afin de permettre à l'utilisateur de saisir les résultats. Malheureusement, les `messageBox` ne sont pas modulables (on ne peut pas rajouter de composants par dessus) et il a donc fallu créer le nôtre. C'est donc un formulaire composé de quatre `TextBox` (la quatrième est invisible si le nombre de formes à mémoriser est trois), de quatre labels (trois si le nombre de formes à mémoriser est trois) qui permettent d'afficher la lettre et d'un bouton. L'utilisateur doit saisir les valeurs qui étaient affichées dans les formes qui correspondent aux lettres affichées. A la création du tableau de `Formes`, on en crée un autre dans lequel on ne stocke que les bonnes formes c'est-à-dire celles qui doivent être mémorisées. De ce fait, on peut facilement récupérer à quelle lettre correspond chaque forme puisqu'une forme possède une lettre parmi ses attributs. Enfin, un message s'affiche sur le formulaire avec le nombre de bonnes réponses. Puis une deuxième série commence et ainsi de suite jusqu'à la 10 série.

FormConcentration

Ce formulaire correspond au test de concentration. Le formulaire de base ne contient que la consigne, les 3 boutons de réponses et le compteur indiquant à quel test on est de la série. Ce test se lance lorsque l'utilisateur clique sur le bouton du Menu correspondant et une messageBox affiche la consigne du test à suivre. La consigne est récupérée dans un fichier XML et générée aléatoirement grâce à la méthode `selectionRegle` de la classe `Concentration.cs`. De plus, grâce à ce fichier XML on récupère l'indice du bouton correspondant au changement de paramètre (forme, couleur, nombre de points) suivant la consigne afin de pouvoir savoir quel bouton correspond à la bonne réponse. Ensuite, le formulaire apparaît comme sur la capture ci-dessous. Les formes et le nombre de points sont générés aléatoirement, comme pour `Perception`. Pour les chiffres, on génère un tableau [3,4] dans lequel on y ajoute des chiffres aléatoires. Pour les formes, on crée un objet de type `Forme` qui possèdent différents attributs (couleur, forme, position en x, position en y, largeur, hauteur). On sélectionne les paramètres de la forme à générer grâce à la méthode `selectionForme` de la classe `Perception.cs`. On conserve ces paramètres de façon à ce que la génération de la forme suivante ne conserve qu'un seul paramètre par rapport à l'image précédente. Cette conservation de paramètre se fait également au sein de cette méthode avec un `random` qui choisit un nombre entre 1 et 3. Suivant le nombre tiré, on sait quel paramètre est conservé. Puis on associe ce paramètre conservé au bouton correspondant dans le formulaire `Concentration`, ainsi on connaît la bonne réponse. Tout comme pour `Perception`, on fait ensuite appel à la méthode `OnPaint` afin de dessiner les formes puis les points grâce aux méthodes `Forme.creationFormeColorée(Forme f, Form form)` et `Forme.creationPoint(int nbPoint, Form form)`. Le choix de tout générer aléatoirement nous a ici aussi demandé plus de réflexion que si nous avions utilisé une base de données mais cela présente les mêmes avantages que pour `Perception`. Un `timer` permet de limiter le temps de réponse à 5 secondes lorsque la difficulté de l'exercice est difficile. A la fin du temps ou au clic sur un des boutons, la forme suivante est générée et cela 5 fois par série. Il y a en tout 3 séries à exécuter; en mode difficile, une nouvelle consigne est exécutée à chaque nouvelle série.



FormCalculMental

Vous allez devoir réaliser des calculs mentaux. Sélectionner l'opération souhaitée !

+

-

X

/

Lorsque l'on lance le test de calcul mental, on obtient le formulaire ci-dessus qui permet de choisir entre les quatre opérations. Le clic sur l'un des quatre boutons envoie sur le formulaire FormOperation décrit ci-dessous.

FormOperation

Ce test était un des plus faciles à mettre en oeuvre, en fonction de l'opération on génère des nombres aléatoires respectant les règles précisées dans le sujet. On stocke alors le résultat et les 2 chiffres aléatoires générés de l'opération et on vérifie si la réponse de l'utilisateur correspond. Si la réponse est fausse, on affiche la bonne réponse et on passe à la suite. A la fin de la série, un message affiche le nombre de bonnes réponses.

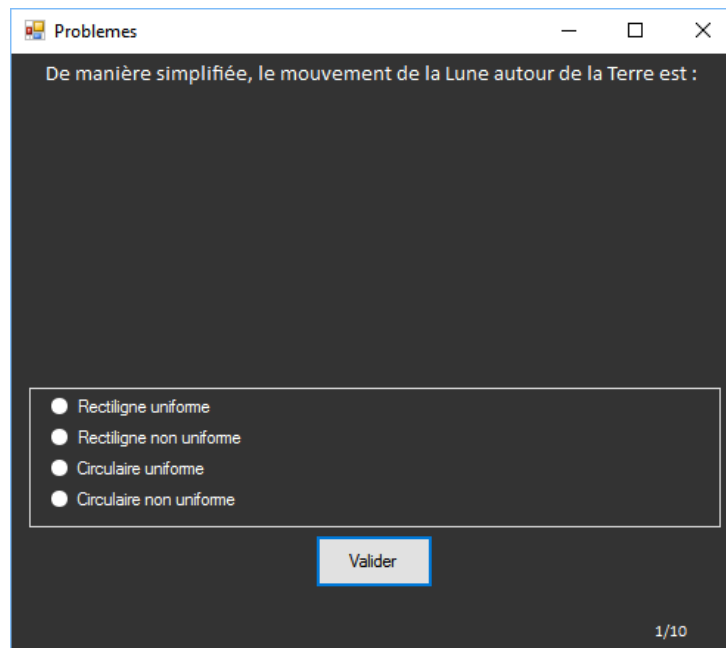
Résoudre

865 + 195 =

Valider

1/10

FormProbleme



Dernier formulaire de notre application. Ce formulaire est commun aux problèmes de mathématiques et de physique. En effet, ces deux tests sont exactement similaires, seules les questions changent. Ainsi, en fonction du test choisi, on chargera l'une ou l'autre des bases de données grâce à la méthode `selectionPbm` de la classe `Probleme.cs`. Le formulaire est constitué de l'énoncé, d'une image qui peut afficher un schéma ou un graphique en fonction de la question, quatre `RadioButton` qui servent à répondre à la question et d'un bouton pour valider son choix. La base de données des questions étant grande, il ne faut sélectionner qu'une partie des problèmes à chaque série. Cette sélection se fait aléatoirement pour avoir des séries uniques à chaque fois. Les problèmes sont stockés dans un tableau. Une fois le tableau récupéré, il suffit d'afficher chaque problème.

Pour chacun de ces Formulaires, on récupère les résultats finaux que l'on conserve dans la classe `Sauvegarde` afin de les exporter dans un XML. Cela permet de conserver les résultats des utilisateurs. La sauvegarde se réalise lorsqu'on quitte l'application ou lorsque l'on change d'utilisateur, soit quand on écrit un nouveau nom d'utilisateur.

3. Résultats et tests

Afin d'être sûr que notre application était fonctionnelle et répondait à l'ensemble des exigences stipulées dans le cahier des charges nous avons réalisé des tests.

Cependant, la majorité de nos méthodes sont des `Void`, elles ne retournent rien. Nous ne pouvons donc pas exercer de tests unitaires sur ces dernières. De plus, pour les méthodes restantes, les paramètres en entrée de ces méthodes ne permettent pas d'anticiper la valeur qui sera retournée. En effet, ces méthodes sont composées de traitements qui reposent sur des "Random". Nous ne pouvons donc pas prévoir, bien que connaissant les paramètres d'entrées de la méthode, ce qui sera retournée par cette dernière.

Les tests unitaires semblent donc compromis pour notre application.

Nous avons cependant pu élaborer des tests fonctionnels sur l'ensemble de nos formulaires qui font appels à toutes les méthodes présentes dans nos classes exercices. L'ensemble des méthodes de notre application ont donc pu être testées grâce à ces tests fonctionnels.

En voici les spécifications de fonctionnalités détaillées. Vous trouverez le détail de ces tests (Risque métier, risque technique, criticité résultante, statut d'exécution) dans le fichier excel :

Resultats Test Fonctionnels.xlsx

Nous avons décidé de ne pas apporter de description à chaque test, l'exigence jugée suffisamment explicite. Cependant, pour la réalisation de chacun des protocoles, nous avons exécuté la partie concernée de l'application et vérifié que tout fonctionnait correctement.

Spécifications fonctionnelles détaillées

1. CU_001 – Actions réalisées depuis le Menu Principal

1.1. Description

Le menu principal permet de choisir le test et sa difficulté. Il permet également de saisir le nom de l'utilisateur et d'afficher ses derniers résultats.

1.2. Description des champs (CU_001_RG_000)

Nom du champ	Type du champ	Remarques
Perception	Bouton	-
Concentration	Bouton	-
Calcul Mental	Bouton	-
Mathématiques	Bouton	.
Physiques	Bouton	-
Difficile	Radio Button	
Facile	Radio Button	
Nom Utilisateur	TextBox	
Checked	PictureBox	
Résultats actuel (pour chaque test)	Label	
Page de chargement	PictureBox	

1.3. Règles de gestion

Identifiant	Description
CU_001_RG_001	La difficulté est définie par un booléen static qui prend la valeur true lorsque <i>Difficile</i> est sélectionné.

CU_001_RG_002	La difficulté est accessible depuis toutes les autres classes grâce à sa propriété static.
CU_001_RG_003	Le Nom Utilisateur est définie par une chaîne de caractères static qui prend la valeur entrée dans la TextBox lors du clic sur <i>Checked</i>
CU_001_RG_004	L'utilisateur doit entrer son nom dans <i>NomUtilisateur</i> de façon à sauvegarder ses résultats. Il a 10 secondes pour saisir son nom et le valider, sinon la NomUtilisateur se réinitialisera avec sa valeur actuelle.
CU_001_RG_005	Les labels de résultats actuels sont définis par des chaînes de caractère static qui récupère le taux de bonne réponse en pourcentage au test associé. Ils sont modifiés uniquement si l'utilisateur réalise le test en entier.
CU_001_RG_006	Chaque bouton renvoie au <i>Form</i> du test associé
CU_001_RG_007	Il y a en tout 5 possibilités de test
CU_001_RG_008	La page de chargement permet de présenter et actuellement de simuler un chargement du programme.

2. CU_002 – Test de Perception

2.1. Description

Ce cas d'utilisation permet de réaliser le test de Perception.

2.2. Description des champs (CU_002_RG_000)

Form Perception

Nom du champ	Type du champ	Remarques
Formes	Classe Forme	Différentes selon les paramètres de création et consigne donnée
NbDePoint	Int	
Lettres	Label	
Consigne	Texte	Générée dans une MessageBox
TimerReponse	Timer	
compteur	Label	

Form MyMessageBox

Nom du champ	Type du champ	Remarques
Lettres	Label	

Saisie réponse	Text Box	
verif	Label	
Valider	Bouton	
TimerAffichageResultat	Timer	
compteur	Label	

2.3. Règles de gestion

Identifiant	Description
CU_002_RG_001	Une forme est définie par une couleur, un type de forme, une position en abscisse, une position en ordonnée, une largeur, une hauteur, une lettre et une valeur.
CU_002_RG_002	Une forme est dépendante de la <i>Consigne</i>
CU_002_RG_003	Affichage de la <i>consigne</i> dans un MessageBox
CU_002_RG_004	Au chargement du test, 3 lignes de 4 colonnes de formes sont générées avec les lettres et valeurs correspondantes grâce aux méthodes de la classe <i>Perception.cs</i> et <i>Forme.cs</i>
CU_002_RG_005	Le temps d'affichage des formes (<i>TimerReponse</i>) est initialisé à 4 secondes en difficulté facile et à 2 secondes en difficulté difficile
CU_002_RG_006	Les formes sont affichées jusqu'à écoulement de <i>TimerReponse</i>
CU_002_RG_007	Mettre l'indice du test actuel dans le label <i>compteur</i>
CU_002_RG_008	<i>MyMessageBox</i> se lance à l'écoulement de <i>TimerReponse</i> .
CU_002_RG_009	Affichage des lettres présentes dans les formes à repérer comme spécifiées dans la consigne. Suivant le nombre de formes à repérer on aura 3 ou 4 lettres.
CU_002_RG_010	Affichage des zones de saisie de réponses (3 ou 4)
CU_002_RG_011	Comparaison des réponses entre celles données par l'utilisateur et les réponses attendues
CU_002_RG_0112	Affichage du nombre de bonnes réponses sur un test dans le label <i>verif</i> tant que le <i>timerAffichageResultat</i> n'est pas écoulé.
CU_002_RG_0113	Relancer le test tant que la série de 10 n'est pas atteinte
CU_002_RG_0114	A la fin du test, sauvegarde du résultat en implémentant la variable static <i>rPerception</i> de la classe <i>Sauvegarde.cs</i>
	Affichage du taux de bonne réponse en pourcentage à l'ensemble des tests
	Retour au menu à tout moment

3. CU_002 – Test de Concentration

3.1. Description

Ce cas d'utilisation permet de réaliser le test de Concentration.

3.2. Description des champs (CU_002_RG_000)

Nom du champ	Type du champ	Remarques
Forme	Classe Forme	Différentes selon les paramètres de création
NbDePoints	Int	Différents selon les paramètres de création
ConsigneLB	Label	
compteur	Label	
Consigne	Texte	Générée dans une MessageBox
TimerConcentration	Timer	
button1	Bouton	Non présent sur le premier écran
button2	Bouton	Non présent sur le premier écran
button3	Bouton	
Valider	Bouton	

3.3. Règles de gestion

Identifiant	Description
CU_002_RG_001	Une forme est définie par une couleur, un type de forme, une position en abscisse, une position en ordonnée, une largeur, une hauteur
CU_002_RG_002	Une forme est dépendante de la forme précédente
CU_002_RG_003	Affichage de la <i>consigne</i> dans un MessageBox
CU_002_RG_004	Au chargement du test, une forme est générée avec le nombre de points correspondants grâce aux méthodes de la classe <i>Perception.cs</i> et <i>Forme.cs</i>
CU_002_RG_005	Mettre l'indice du test actuel dans le label <i>compteur</i>
CU_002_RG_006	En difficulté facile, au clic sur un bouton de réponse, une nouvelle forme apparaît. En difficile, une nouvelle forme apparaît au clic sur un bouton de réponse ou après 5 secondes.
CU_002_RG_007	Il est obligatoire d'appuyer sur le <i>bouton3</i> au premier écran.
CU_002_RG_008	Il y a 3 séries de 5 écrans successifs en tout. En difficile, à chaque série il y a une nouvelle consigne.
CU_002_RG_009	Relancer le test tant que la 3ème série n'est pas terminée
CU_002_RG_010	Comparaison des réponses entre celles données par l'utilisateur et les réponses

	attendues
CU_002_RG_011	A la fin de chaque série, on affiche le taux de bonne réponse avec un MessageBox
CU_002_RG_0112	A la fin des 3 séries, on affiche le taux de bonne réponse total en pourcentage avec un MessageBox
CU_002_RG_0113	A la fin du test entier, sauvegarde du résultat en implémentant la variable static <i>rConcentration</i> de la classe <i>Sauvegarde.cs</i>
	Retour au menu à tout moment

4. CU_002 – Test de Calcul Mental

4.1. Description

Ce cas d'utilisation permet de réaliser le test de CalculMental.

4.2. Description des champs (CU_002_RG_000)

FormCalculMental

Nom du champ	Type du champ	Remarques
Consigne	Label	
+	Bouton	Différents selon les paramètres de création
-	Bouton	
x	Bouton	Générée dans une MessageBox
/	Bouton	

FormOperation

Nom du champ	Type du champ	Remarques
Consigne	Label	
ch1	Label	Différents selon l'opération choisie
ch2	Label	Différents selon l'opération choisie
Signe	Label	Récupère le signe de FormCalculMental
=	Label	Générée dans une MessageBox
Valider	Bouton	
verif	Label	
compteur	Label	
TimerDifficile	Timer	
TimerAfficheResultat	Timer	

4.3. Règles de gestion

Identifiant	Description
-------------	-------------

CU_002_RG_001	Affichage de la consigne en haut de l'écran
CU_002_RG_002	Récupération du signe
CU_002_RG_003	Sélection aléatoire des chiffres suivant l'opération dans la classe Operation.cs
CU_002_RG_004	Mettre l'indice du test actuel dans le label <i>compteur</i>
CU_002_RG_005	En difficulté facile, l'utilisateur valide sa réponse en cliquant sur le bouton Valider ou en appuyant sur Entrée. En difficile, il y a une limite de temps de 5 secondes pour répondre
CU_002_RG_006	Comparaison des réponses entre celles données par l'utilisateur et les réponses attendues
CU_002_RG_007	Affichage de la bonne réponse sur un test dans le label <i>verif</i> tant que le timerAffichageResultat n'est pas écoulé.
CU_002_RG_008	Relancer le test tant que l'on n'a pas atteint le 10ème test
CU_002_RG_009	A la fin des 10 tests, on affiche le taux de bonne réponse en pourcentage avec un MessageBox
CU_002_RG_010	A la fin des 10 tests, sauvegarde du résultat en implémentant la variable static <i>rCm</i> de la classe <i>Sauvegarde.cs</i>
CU_002_RG_011	Retour au menu à tout moment

5. CU_002 – Test de Problèmes

5.1. Description

Ce cas d'utilisation permet de réaliser les tests de mathématiques et physiques

5.2. Description des champs (CU_002_RG_000)

Nom du champ	Type du champ	Remarques
ConsigneLB	Label	
Consigne	Texte	Générée dans une MessageBox
imgTest	PictureBox	
Choix1	Radio Bouton	
Choix2	Radio Bouton	
Choix3	Radio Bouton	
Choix4	Radio Bouton	
Valider	Bouton	
verif	Label	
compteur	Label	
TimerAfficheResultat	Timer	

5.3. Règles de gestion

Identifiant	Description
CU_002_RG_001	Affichage de la <i>consigne</i> dans un MessageBox
CU_002_RG_002	Récupération de la matière et de la difficulté
CU_002_RG_003	Sélection aléatoire des problèmes suivant la matière et la difficulté dans la classe <i>Probleme.cs</i>
CU_002_RG_004	Mettre l'indice du test actuel dans le label <i>compteur</i>
CU_002_RG_005	Récupérer la potentielle image en supplément de la consigne texte
CU_002_RG_006	Comparaison des réponses entre celles données par l'utilisateur et les réponses attendues
CU_002_RG_007	Affichage de la bonne réponse sur un test dans le label <i>verif</i> tant que le <i>timerAffichageResultat</i> n'est pas écoulé.
CU_002_RG_008	Relancer le test tant que l'on n'a pas atteint le 10ème test
CU_002_RG_009	A la fin des 10 tests, on affiche le taux de bonne réponse en pourcentage avec un MessageBox
CU_002_RG_010	A la fin des 10 tests, sauvegarde du résultat en implémentant la variable static <i>rMaths</i> ou <i>rPhysique</i> de la classe <i>Sauvegarde.cs</i>
CU_002_RG_011	Retour au menu à tout moment

Nous avons également réalisé des tests utilisateurs afin de s'assurer que notre interface était claire, intuitive et esthétique. Pour cela, nous avons choisi des personnes non concernées par ce projet (pas étudiants à l'ENSC) et ne possédant pas une grande habitude des technologies, c'est à dire des membres de notre entourage. Bien que ces derniers ne soient pas astronautes, ils ont pu nous faire des retours sur notre interface. Leur non habitude avec les interfaces ordinateurs nous ont permis d'être sûr de la clarté et la facilité d'utilisation de notre application. Nous nous sommes basés sur ces remarques pour améliorer continuellement notre interface

4. Bilan et perspectives

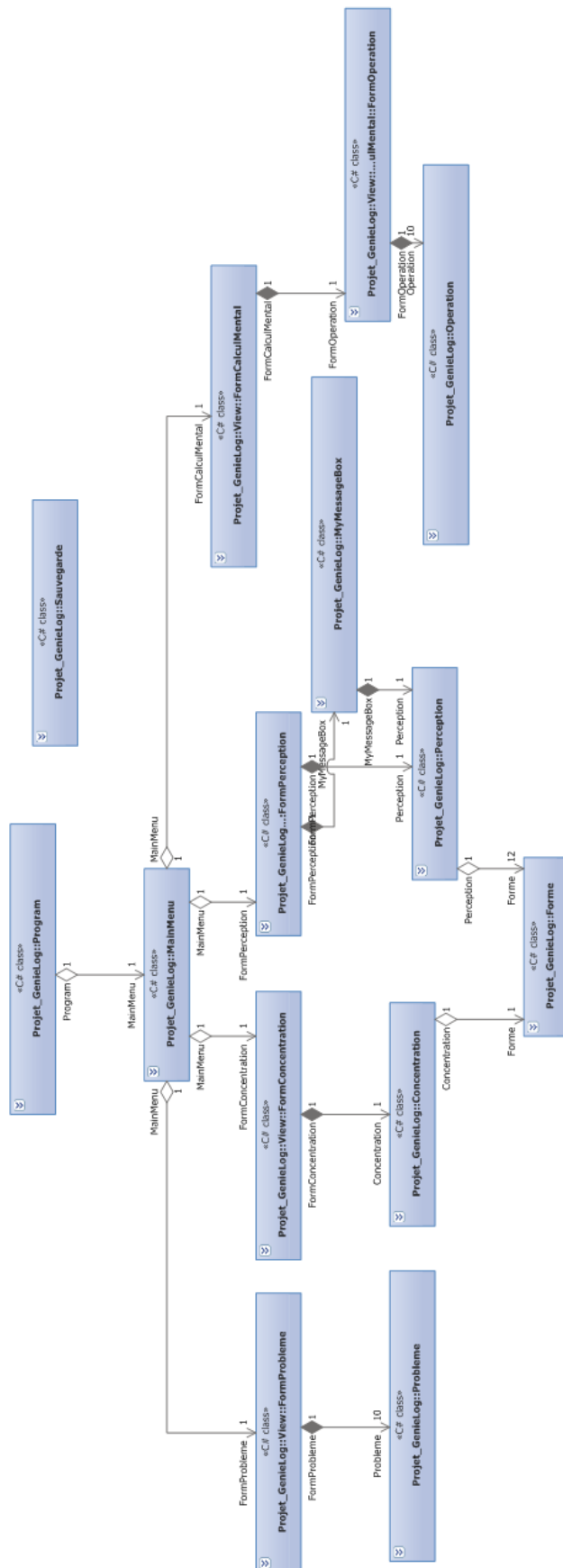
Notre application dispose de toutes les fonctionnalités qui étaient demandées dans le cahier des charges. Tous les tests ont été implémentés et fonctionnent. Nous avons aussi ajouté quelques fonctionnalités supplémentaires comme par exemple l'affichage des résultats sur le menu principal ou encore la sauvegarde des données dans un fichier XML afin de pouvoir comparer et étudier les résultats des astronautes. Au niveau de l'interface, une attention particulière a été portée sur

l'aspect graphique afin qu'elle soit la plus agréable possible. Le thème sombre rappelle l'obscurité de l'espace.

Ce projet de génie logiciel nous a permis de mettre en œuvre les pratiques propres au génie logiciel afin de maximiser la réussite de ce projet. Il y a eu des phases de réflexion et de remise en question des choix qui avaient été faits mais aussi des moments de satisfaction lorsque nous avançons dans le projet. Nous avons rencontré quelques difficultés par moment notamment lors de la génération des formes de couleur. La technologie des winforms est assez contraignante et ne permet pas une grande modularité au niveau des traitements graphiques. Mis à part cela, il est très facile d'obtenir des résultats rapidement puisque l'on voit tout de suite les éléments que l'on place sur nos différents forms. Nous avons également pu apprécier l'utilité et la facilité d'utilisation de GitHub pour pouvoir travailler séparément et rester au courant de l'avancement de chacun. Nous l'utilisons déjà tous les deux avant ce projet et nous sommes persuadés que c'est un excellent outil de travail.

D'un point de vue général, nous sommes plutôt satisfaits de notre application. Pour la rendre encore meilleure, on pourrait imaginer encore plus d'options comme par exemple une base de données afin de pouvoir stocker les utilisateurs en ligne, une plus grande base de données de questions, une interface plus personnalisée... Avec toutes ces options, peut-être que l'ESA utiliserait notre logiciel !

Annexes



Annexe1 : Diagramme de classe

Menu Principal

Perception

Concentration

Calculs
mentaux

Mathématiques

Physiques

Niveau de difficulté

☐ Facile


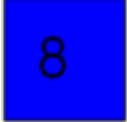










☐ Difficile

PERCEPTION

Vous allez devoir mémoriser les valeurs
de tous les objets de la forme X donnée et
de la couleur Y donnée, comme par
exemple : mémorisez les
valeurs de tous les ronds bleus.
Etes vous prêt ?

Ready !

Mémoirisez les valeurs des ronds bleus

A 	B 	C 	D 
E 	F 	G 	H 
I 	J 	K 	L 

CONCENTRATION

Vous devrez appuyer sur le bouton 1, le bouton 2 ou le bouton 3 selon que 2 objets présentés sur 2 écrans consécutifs respectent ou pas une certaine règle.
Etes vous prêt ?

Ready !

Règles:

Appuyez sur le bouton 1 si deux objets consécutifs ont la même couleur.

Appuyez sur le bouton 2 si deux objets consécutifs ont le même nombre de points.

Appuyez sur le bouton 3 dans tous les autres cas.



BUTTON 1

BUTTON 2

BUTTON 3

CALCUL

Vous allez devoir réaliser des calculs mentaux. Sélectionner l'opération souhaitée !

+

-

x

/

Addition

Résoudre :

$$514 + 612 =$$

MATHEMATIQUES

Vous allez devoir résoudre une série de
10 problèmes de mathématiques portant
sur des thèmes différents.
Etes vous prêt ?

Ready !

1) Hors-saison, un hôtel propose une chambre double à 80 Euros. En pleine saison, le prix augmente de 15%. Quel est le prix de la chambre double en pleine saison ?

☐ 81.5€

☐ 92€

☐ 95€

☐ 120€

Valider

Vous avez eu 90 % de réussite !