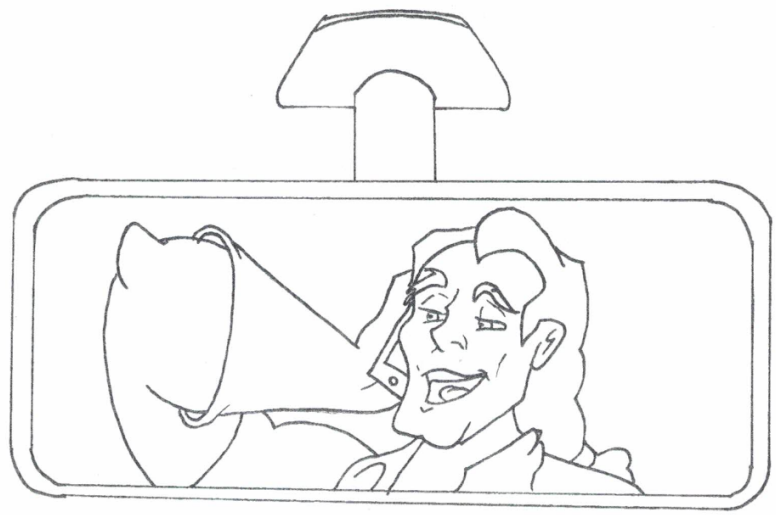
SCIENCES SORBONNE UNIVERSITÉ

POLYTECH SORBONNE

Rapport projet C++

There is no Planet B



DELABORDE Romane - MAIN4
JOLIVET Maëlle - EISE4

Année 2021-2022

Introduction

Lors de la découverte du thème “There is no Planet B”, de nombreuses pistes avaient été envisagées. Cependant, nous avions particulièrement été séduites à l’idée d’exploiter le côté obscur que pourrait évoquer cette thématique. En particulier, nous avions envie de présenter notre travail sous la forme d’un jeu afin de marquer les esprits, et faire passer un message sous forme ludique.

Notre projet dans le cadre de ce cours de langage objet C++ est donc un jeu d’énigmes intitulé “There is no Planet B” dans lequel le joueur va devoir faire des choix.

Il est à noter que ce rapport dévoile entièrement les mécanismes du jeu et par conséquent les réponses aux énigmes : vous souhaiterez peut-être jouer d’abord, puis lire ce rapport ensuite. Vous pouvez alors directement vous référer à la partie qui suit.

Procédure d’installation et d’exécution

Le projet se trouve à l’adresse github :
https://github.com/RomaneDelaborde/There-is-no-Planet-B
Il est à noter que nous n’avons pas testé notre jeu sous Windows, nous ne pouvons donc pas garantir son fonctionnement sous cet OS, ou aider au téléchargement des librairies, ni à la compilation. Vous pouvez soit cloner le projet (git clone), soit le télécharger en format zip. Une fois le projet installé, naviguez jusque dans le dossier.
Les librairies nécessaires à la compilation du Jeu sont les librairies standards du C++, ainsi que la librairie Catch pour les Testcases, et la librairie Gtkmm3 pour le graphisme.

Cette dernière s’installe comme suit :

- sous Linux : `sudo apt-get install libgtkmm-3.0-dev`
- sous MacOs avec Homebrew : `brew install gtkmm3`

Pour compiler, entrez la commande `make`. *(note : sous MacOs, le compilateur clang peut afficher des warnings : ils ne portent que sur la non-utilisation de librairies).*

Pour exécuter, entrez la commande `./planetB` . Le jeu est lancé, vous n’avez qu’à suivre les instructions. Bonne chance !

Pour les tests, naviguez jusqu’au dossier nommé **tests**. Puis entrez `make`, puis entrez `./testcase`.

Description de l’application

There is no Planet B est un jeu d’énigmes dans lequel le joueur va devoir faire des choix.

Résumé du jeu *(à lire de préférence après avoir joué)* :

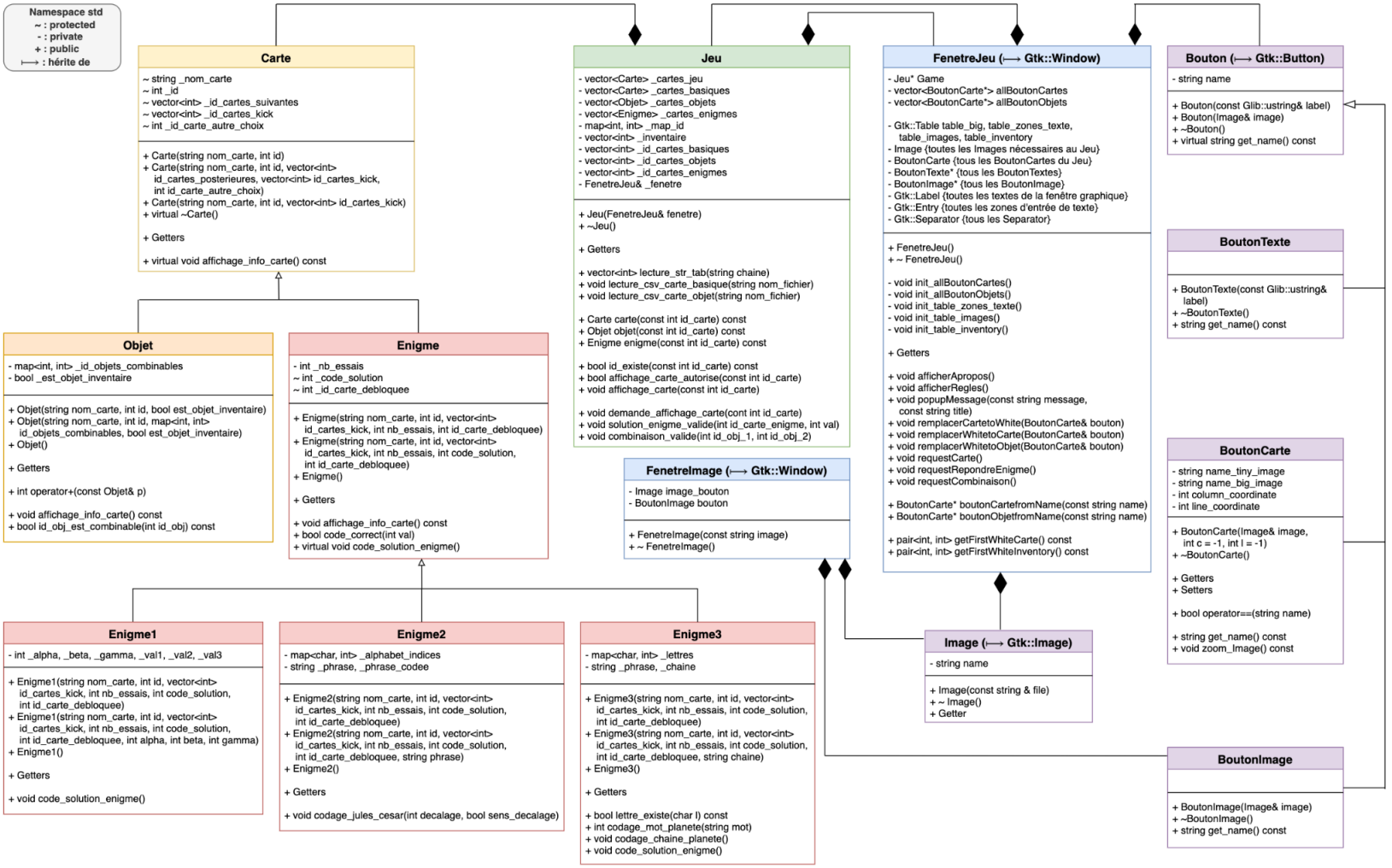
La première partie de l’histoire se déroule dans une voiture : vous incarnez le chauffeur privé d’un Suisse plutôt fortuné. Après une annonce gouvernementale à la radio au sujet de l'environnement, vous entendez des bribes de conversation entre votre patron et l’un de ses confrères. Ils semblent évoquer une réception de colis importante, ce qui éveille des soupçons en vous. Vous avez alors deux choix possibles : suivre votre intuition en menant votre enquête chez votre patron (le jeu continue) OU obéir à votre patron en vous rendant au pressing (le jeu s’arrête).

Si le jeu continue, vous arrivez dans le garage de votre patron. En vous introduisant par la suite dans la maison, vous vous retrouvez face à une bibliothèque qui semble cacher une porte. Une fois la porte secrète atteinte, vous découvrez un bureau. Après avoir mis la main sur un certificat et un ticket doré prouvant clairement l’existence d’une planète de secours (une planète B), en cas de problème sur la Terre, vous faîtes face à votre patron.
Deux possibilités sont alors proposées : assommer votre patron OU le tuer.

Peu importe le choix fait ci-dessus, vous devez à nouveau prendre une décision. Soit révéler au monde entier la supercherie : vous ne pourrez vous-même pas profiter de ce ticket vers la planète B. Soit ne pas révéler au monde la supercherie : vous pouvez envisager un jour de quitter la Terre grâce à votre ticket. Dans les deux situations, vous aurez des comptes à régler avec la justice à cause de votre patron.

Sur la page suivante, nous présentons le diagramme UML complet de notre application (en incluant également toutes les classes en lien avec l’affichage graphique du Jeu). La qualité de l’image n’étant pas idéale, vous pouvez retrouver ce diagramme dans le dépôt git, sous le nom ***uml.png***.

Diagramme UML de l'application



Mise en valeur de l'utilisation des contraintes

Notre jeu est composé au total de **14 classes**, dont 8 qui sont primordiales au fonctionnement de notre projet : Carte, Objet, Enigme, Enigme1, Enigme2, Enigme3, Jeu, FenetreJeu.

Les classes Enigme1, Enigme2 et Enigme3 sont les filles de la classe Enigme, qui est elle-même la fille de la classe Carte. Cette **hiérarchie** est cohérente car il existe des points communs entre chaque énigme même si les énigmes ne sont pas toutes construites de la même façon. Par ailleurs, pour les objets comme pour les énigmes, ils ne peuvent pas exister sans que ces derniers soient reliés à une carte.

Carte::affichage_info_carte() est **virtuelle** puisqu'elle est redéfinie dans les classes filles Objet et Enigme. Elle n'est pas virtuelle pure puisque sa définition est nécessaire dans Carte, pour l'affichage des infos des cartes basiques.

Le getter *Bouton::get_name()* est **virtuel pur**. En effet, le nom recherché n'est pas le même dans le cas d'un BoutonTexte (pas de nom), d'un BoutonAccueil (nom de l'image en fond) ou d'un BoutonCarte (nom de la Carte correspondante). Il est donc obligatoirement redéfinie dans les classes filles.

Enigme::calcul_code_solution() est **virtuelle** car généralement pour chaque énigme, la méthode de calcul de la solution est unique. Il est à noter que cette méthode virtuelle n'est pas utilisée dans la classe Enigme2, puisque le code solution est connu en amont. Il n'y a donc aucune raison de le calculer.

BoutonCarte::operator==(string name) est **surchargé**, il permet de faciliter la recherche des correspondances Carte-BoutonCarte

int Objet::operator+(const Objet & p) est **surchargé**, il permet de simplifier l'accès à la carte déverrouillée par la combinaison des deux objets, en renvoyant la valeur de son id.
Si cette valeur n'existe pas (en particulier si les deux objets ne sont pas combinables) alors cette surcharge renvoie -1.

Il a été pertinent d'utiliser des **vectors** afin de stocker de nombreuses informations et en particulier les id des cartes, que cela concerne la classe Jeu ou encore la classe Carte.

Par ailleurs, il a été pertinent de créer des **maps** afin de faciliter l'implémentation du jeu. Plus précisément, il a fallu créer un map dans la classe Jeu avec comme clés les id de toutes les cartes du jeu et comme valeurs, un entier indiquant son statut d'affichage dans le jeu (1 si la carte est affichée, 0 si elle ne l'est pas et -1 si elle n'est pas affichable).

Enfin, nous utilisons également des **pairs**, qui ne sont pas des conteneurs à proprement parler mais qui permettent de contenir 2 éléments du même type ou de types différents : nous utilisons des pair<int, int> pour renvoyer des coordonnées (x,y).

Les **commentaires** dans le code ont facilité la mise en œuvre du jeu car malgré le nom des variables assez explicites, il est pratique de se rappeler rapidement du rôle de ces

dernières. Les commentaires nous ont également permis de rapidement comprendre les parties du code implémentées par l'autre personne du binôme.

Les **tests unitaires** ont permis de s'assurer de la bonne implémentation des classes au fur et à mesure de la création du jeu. Par ailleurs, ils permettent de tester les énigmes avec des valeurs autres que celles utilisées dans le jeu. De plus, cela permet de faire une simulation du jeu sans affichage graphique *(ou du moins avec un affichage graphique allégé)*.

Fiertés

Maëlle : D'une manière générale, je suis très fière que nous soyions arrivées à créer un jeu cohérent, jouable et original, en partant de zéro : nous avons créé le scénario et dessiné toutes les images du jeu. Ce projet est construit d'une manière qui rend possible sa complexification, c'est-à-dire que dans le futur nous pourrions repartir de ce projet pour créer d'autres Escape Game / Jeux de décisions. Nous avons réussi à implémenter tout ce que nous souhaitions au départ, y compris pour l'interface graphique. La dérivation des classes de la librairie Gtkmm a été très pratique pour pouvoir réaliser une interface correspondant parfaitement aux besoins de notre Jeu.

Romane : Je pense que même si les énigmes n'étaient pas la partie du code la plus difficile à implémenter, j'en suis plutôt fière. J'ai trouvé effectivement cette partie très intéressante non seulement car les trois types d'énigmes étaient assez diversifiés mais aussi parce que j'ai appris à manipuler les maps. En outre, si nous avons l'occasion de coder à nouveau un jeu dans ce style, il pourra être envisageable de réutiliser l'ensemble des classes créées au cours de ce projet.

Conclusion

“There is no Planet B” est un jeu que nous avons conçu entièrement, de la rédaction des règles à l’implémentation du code, en passant par le processus artistique.
Même si nous sommes vraiment satisfaites de l’ensemble de notre projet, il existe néanmoins des points d’amélioration. Dans un premier temps, il peut être envisageable de proposer des indices au joueur si jamais celui-ci est bloqué dans le jeu. De plus, il faudrait imposer un nombre limité de tentatives au joueur pour résoudre une énigme. Enfin, notre programme semble causer des problèmes mémoire détectés avec Valgrind, que nous n’avons pas eu le temps de corriger : une amélioration du projet est alors de supprimer toute fuite de mémoire.

Concernant la réalisation de futurs projets du même type, il peut être tout à fait envisageable de s’appuyer sur le code que nous avons implémenté dans le cadre de ce cours. En outre, il peut être intéressant de s’intéresser à d’autres types d’énigmes comme par exemple des codes contenant des lettres.

Enfin, nous aimerions remercier Cécile Braunstein d’avoir proposé un sujet riche et intriguant qui nous a laissé un grand choix de libertés, ainsi que Johann Huber pour nous avoir aidé lors des séances de projet à bien définir notre projet, et qui a validé notre direction artistique et technique.

Annexes : Règles du Jeu

There is no Planet B est un jeu d'énigmes dans lequel vous allez devoir faire des choix.
Avant de jouer, nous vous conseillons d'avoir de quoi noter à proximité.

Les règles sont les suivantes :
Chaque carte possède un code unique indiqué en haut à droite de cette dernière.
Les cartes énigmes ont leur code en couleur verte.
Il existe des cartes objets qui peuvent être ou non ajoutées à l'inventaire.
Un objet peut être inutile ou au contraire être utilisé plusieurs fois.
Un objet reste dans l'inventaire même si celui-ci a déjà servi.
Les codes rouges se trouvant sur une carte correspondent en général à des codes de cartes déblocables, mais ce n'est pas toujours le cas.

Au fil du jeu, certaines cartes disparaîtront automatiquement de l'écran. Cela signifie que vous avez exploité tout leur potentiel, elles ne sont donc pas réaffichables.

Voici comment jouer :
Vous avez 3 possibilités d'entrées de code en haut de l'écran :

→ **Affichage d'une nouvelle carte :** Entrez le code rouge correspondant à la carte à afficher
Précision : Lorsque vous rentrez un code correspondant à un objet à ramasser, celui-ci sera ajouté à votre inventaire, en bas de l'écran. Pour ce type d'objet, vous devez obligatoirement les avoir dans votre inventaire avant de pouvoir les combiner avec d'autres objets.

→ **Combinaison de deux objets :** Entrez dans la première case un code correspondant à un objet et dans la seconde, un autre code du même type. L'ordre n'a pas d'importance.

→ **Réponse à une énigme :** Entrez dans la première case le code de la carte énigme et dans la seconde, votre réponse à cette énigme.

Vous pouvez afficher les règles du jeu à n'importe quel moment dans le jeu en appuyant sur le bouton "?".

P.S. : Le début du jeu se déroule dans une voiture. Il est important de rappeler le fait que dans la réalité il ne faut pas conduire et résoudre des énigmes en même temps.

Annexes : Solutions du Jeu

Ci-dessous, vous retrouverez toutes les "réponses" du jeu.

Assommer le patron : vous devez combiner le pied de biche (carte n°205) avec la tête du patron (n°404).

Tuer le patron : vous devez combiner le revolver (n°401) avec la tête du patron (n°404).

Ouverture de la porte électronique du garage (n°203) : attention pour pouvoir déverrouiller cette porte, vous devez avoir tout fouillé en amont dans le garage. Si cela est fait, vous devez utiliser le pied de biche (n°205) trouvé dans la boîte à outils (n°24) et le combiner avec la porte électronique (n°203).

Ouverture de la porte secrète (n°302) : Vous devez combiner l'épingle à cheveux (n°101) trouvée dans la boîte à gants de la voiture (n°12) avec la porte secrète (n°302).

Solutions des énigmes :

Énigme de la radio : carte n°13. | Cartes utiles à la résolution : 10, 11 et 12

Solution : Dans un premier temps, il faut résoudre le système inscrit sur le post-it de la carte n°12. Les valeurs obtenues sont les suivantes : α = 8, β = 7 et γ = 9.
Il suffit ensuite de regarder la liste des radios données sur la carte n°11. Sachant que sur la carte n°10 il est indiqué qu'il est 17h, il n'y a que la radio Rouge FM qui émet à cette heure. Sa fréquence associée est αβ soit 87 MHz. La solution de l'énigme est donc **87**.

Énigme de la bibliothèque : carte n°32. | Cartes utiles à la résolution : 204, 31, 32

Précisions sur les conditions d'obtention de la carte n°204 : Après avoir trouvé 2 morceaux de papier dans le garage (cartes 22 et 23 qui impliquent respectivement les cartes 201 et 202), vous devez les rassembler en combinant les cartes 201 et 202. Le résultat de cette combinaison est un dessin d'enfant. En retournant ce dessin avec le code 26, vous obtenez un dessin étrange au verso. Avec le code 204, vous récupérez cet objet dans votre inventaire (carte 204).

Solution : Il s'agit de décrypter le message inscrit sur la carte 31 grâce à la carte 204. Le décodage donné sur la carte n°204 correspond au code de César : il s'agit dans ce cas d'un décalage de 7 dans l'alphabet. Ici : I devient B, C devient V, B devient U etc. Ainsi, le message caché **"APYLY SL SPCYL KL YVHSK KHOS"** de la carte 31 devient **"TIRER LE LIVRE DE ROALD DAHL"**. Il faut donc tirer le livre "Charlie et la Chocolaterie" de la carte 32, qui correspond au code 311. La solution de l'énigme est donc **311**.

Énigme du coffre-fort : carte n°50. | Carte utile à la résolution : 42

Solution : En bas du tableau de la carte 42, il y a écrit UNJUST + B =_ _ _ _ _ _
Cela signifie qu'il faut convertir UNJUST et B en chiffres. Toujours sur la même carte et en regardant l'ordre des planètes par rapport au soleil, il est possible d'en déduire la valeur des lettres : V=2, T=3, J=5, S=6, U=7, N=8, B=9. Les valeurs correspondantes à la lettre M ne sont pas prises en compte car une lettre ne peut pas avoir plusieurs valeurs. En outre, la lettre M n'apparaît pas dans les mots à décoder. En convertissant les lettres en chiffres, UNJUST devient 785763 et B vaut 9. UNJUST+B revient donc à faire 785763+9=**785772**, qui est le code pour ouvrir le coffre-fort.

En résumé :

- Solution énigme de la radio (carte 13) : 87
- Solution énigme de la bibliothèque (carte 32) : 311
- Solution énigme du coffre-fort (carte 45) : 785772

Actions capitales à réaliser pour avancer dans le jeu :

→ Avoir ramassé l'épingle à cheveux (carte 101) dans la boîte à gants (carte 12) avant de répondre à l'énigme de la radio (carte 13)

→ Avoir ramassé la feuille avec la roue de César (carte 204) avant de forcer la porte → électronique (carte 203) avec le pied de biche (carte 205)
→ Avoir ramassé le ticket doré (carte 402), le certificat (carte 403) et le revolver (carte 401) avant de déclencher l'arrivée du patron dans le bureau

Clins d'œil :

Des références à la planète B *(mais pas que)* ont été glissées tout au long du jeu :

- la liste de courses commençant avec la lettre B dans la boîte à gants de la voiture
- la chanson "one way ticket" en référence au ticket d'or aller simple vers la planète B
- le paillasson dans le garage avec toutes les lettres de l'alphabet en minuscule sauf la lettre B en majuscule
- le livre de roald dahl en référence au ticket d'or
- la boîte de chocolat Willy Wonka toujours en rapport avec Charlie et la Chocolaterie
- la planète B ajoutée au système solaire sur le tableau avec les planètes
- le code "UNJUST+B" sur le tableau pour la traduction anglaise de "INJUSTE" et pour la planète "B"

Annexes : Description des Classes

La description détaillée des Classes étant très intéressante mais également très longue, nous la passons en annexes, puisque le diagramme UML est assez complet dans le corps du rapport. Avant de détailler les classes, il est important de prendre en considération les précisions suivantes :

→ Une carte basique est une carte qui n'est ni un objet, ni une énigme.
→ Les cartes dites "secrètes" sont uniquement déblocables par une énigme ou une combinaison d'objets.
→ Une carte où le joueur doit prendre une décision, par exemple aller au pressing ou non (voir carte 16) donne lieu à deux cartes "choix" (cartes 17 et 20 dans notre exemple).
→ Lorsqu'il y a écrit la carte X, il s'agit en réalité de la carte d'id X.

La classe **Carte** contient toutes les informations nécessaires à la création d'une carte. Ces informations correspondent aux attributs suivants :

- **_nom_carte** : nom de l'image associée à la carte (sans l'extension du format)
- **_id** : identifiant propre à chaque carte
- **_id_cartes_suivantes** : liste des id des cartes suivantes à afficher (ne contient pas les cartes secrètes déblocables par une énigme ou une combinaison d'objets)
- **_id_cartes_kick** : liste des id des cartes permettant de kicker la carte de l'affichage du jeu (i.e. quand elle n'est plus utile)
- **_id_carte_autre_choix** : id de la carte de l'autre choix possible si la carte est une carte choix, sinon cet attribut vaut 0 (pour pouvoir empêcher son affichage)

Il est à noter que chaque id de carte a été déterminé manuellement afin de pouvoir contrôler plus facilement le déroulement du jeu. De plus, il existe une logique dans l'attribution des id :

- la carte 1 est la première à apparaître donc il s'agit de la carte d'id 1
- la plupart des cartes basiques sont composées de deux chiffres dont le premier correspond à la pièce avec laquelle elle est reliée
- les cartes choix et objets sont composées de trois chiffres

Par conséquent, il n'était pas envisageable d'initialiser cet id via un static.

Il existe 3 constructeurs différents pour cette classe : le premier est utile à la classe fille Objet, le second est dédié à la classe fille Énigme tandis que le dernier permet la création d'une carte basique. Les méthodes de la classe Carte sont essentiellement composées de getters. Il y a également une méthode virtuelle, affichage_info_carte, permettant d'afficher tous les attributs d'une carte, qui fût très utile pour l'implémentation du code. Cette méthode est redéfinie pour Objet et Enigme.

La classe **Objet** est une classe fille de **Carte** car un objet est simplement une carte avec quelques spécificités en plus. Elle possède en supplément des attributs de sa classe mère, les attributs suivants :

- **_id_objets_combinables** : map ayant les id des objets combinables avec cet objet en clés et l'id de la carte résultante de cette combinaison en valeur

- **_est_objet_inventaire** : booléen valant 1 si l'objet peut être déposé dans l'inventaire et 0 sinon

Certains objets ne sont pas combinables avec d'autres (par exemple le ticket doré ou encore le certificat), ce qui justifie l'existence de deux constructeurs. En addition des getters dans les méthodes, il existe une méthode id_obj_est_combinable, qui selon l'id de l'objet passé en paramètre, renvoie 1 si celui-ci est combinable avec l'objet en question et 0 sinon.

La classe **Enigme** est une classe fille de **Carte** car tout comme pour la classe **Objet**, une énigme est simplement une carte avec quelques spécificités en plus.
Elle possède en supplément des attributs de sa classe mère, les attributs suivants :

- **_nb_essais** : nombre de tentatives autorisées pour réussir l'énigme (pas utilisé)
- **_code_solution** : solution de l'énigme; il s'agit toujours d'un entier
- **_id_carte_debloquee** : carte débloquée lorsque l'énigme est réussie

En addition des getters dans les méthodes, il existe une méthode code_correct, qui selon la valeur entière passée en paramètres, renvoie 1 si la solution proposée est correcte et 0 sinon. Il est important de préciser que la classe Enigme n'est appelée uniquement dans ses classes filles : elle n'est pas instanciable. En effet, cette classe sert uniquement de base aux différentes énigmes de notre jeu, étant donné que chaque énigme possède ses propres caractéristiques.

La classe **Enigme1** se situe au troisième niveau de hiérarchie de notre diagramme UML.
Elle est effectivement la fille de classe **Enigme**, qui est elle-même la fille de la classe **Carte**.
Il en est de même pour les classes **Enigme2** et **Enigme3** (voir prochains paragraphes).
La classe **Enigme1** permet de calculer les valeurs à droite des équations du système suivant (à partir de l'initialisation des valeurs de α, β et γ) :
(voir solution de l'énigme 1 dans la section "Annexes" pour plus de détails)

$$\alpha + \alpha + \alpha = ?$$

$$\beta + \beta + \alpha = ?$$

$$\gamma + \gamma + \beta = ?$$

La classe **Enigme2** permet de coder un message par substitution monoalphabétique, suivant un décalage de N lettres dans l'alphabet. La valeur du décalage peut être modifiée manuellement dans la classe. Contrairement aux autres énigmes, le code correspondant à la solution de l'énigme est déjà connu car il s'agit de l'id d'un objet. Par conséquent cette classe est uniquement utile lors du processus de création du jeu, permettant ainsi d'automatiser le codage d'un message. Cependant, ce code peut être réutilisé pour la réalisation d'un nouveau jeu (généralisable également aux codes correspondants aux autres énigmes).

La classe **Enigme3** permet de coder des mots en chiffres suivant une certaine logique (1 lettre = 1 chiffre donné). Il est également possible de "sommer" des mots pour complexifier légèrement l'énigme.

La classe **Jeu** permet de regrouper les classes **Carte**, **Objet** et celles correspondant aux énigmes. En effet, le constructeur de Jeu va recueillir toutes les informations sur les cartes du jeu, fournies en grande partie grâce à des fichiers csv.
Dans un premier temps, le constructeur fait appel à une méthode qui permet de lire le fichier csv contenant les données sur les cartes basiques. Puis, il répète l'opération avec une méthode différente avec le fichier csv lié aux cartes objets. Quant aux cartes énigmes, il n'en existe seulement trois, il n'est donc pas nécessaire de créer un fichier csv spécifique aux énigmes. À chaque fois qu'une nouvelle carte basique ou non est lue, ses informations sont enregistrées dans diverses listes (vector) correspondant à des attributs de la classe Jeu.
En addition des getters dans les méthodes, il existe plusieurs fonctions dont celles qui suivent sont les plus importantes :

- **demande_affichage_carte** : si l'affichage de la carte passée en paramètre est autorisé, elle peut être ajoutée à l'interface graphique, sinon un pop-up d'erreur apparaît à l'écran
- **solution_enigme_valide** : si le code entier passé en paramètre correspond à la solution de l'id de l'énigme également passée en paramètres, alors la carte secrète correspondant à l'énigme est débloquée, sinon un pop-up d'erreur apparaît à l'écran
- **combinaison_valide** : si les deux objets passés en paramètres (sous forme d'id) sont combinables, un pop-up apparaît sur l'écran indiquant quelle carte est déverrouillée, sinon un pop-up d'erreur apparaît à l'écran

Précisions - En supposant que son id existe, une carte ne peut pas être affichée dans les cas suivants :
- si elle est déjà affichée à l'écran
- si elle n'a pas encore été débloquée : il faut que la carte donnant accès à cette carte soit déjà affichée à l'écran
- si elle a déjà été affichée au cours de la partie et qu'elle n'est plus utile à l'avancement du jeu : elle a donc été kickée automatiquement par le jeu et n'est pas réutilisable
- si elle correspond à la carte reliée à l'autre choix possible : par exemple si le joueur a décidé de tuer le patron, il ne doit pas pouvoir afficher la carte où le patron est assommé

En utilisant la bibliothèque graphique Gtkmm, nous avons pu créer les **classes graphiques** nécessaires à notre jeu en dérivant les classes existantes de la bibliothèque. Ainsi, nous obtenons 7 classes supplémentaires : FenetreImage, FenetreJeu, Bouton (et ses 3 filles BoutonTexte, BoutonCarte et BoutonImage), et enfin Image. Nous ne détaillons que la plus intéressante : **FenetreJeu**.

Cette classe permet de gérer la partie visuelle du jeu, ainsi que de lier les actions du Joueur (*cliquer sur un bouton, entrer un nombre..*), à des actions sur le Jeu (*débloquer une carte*). Son constructeur initialise tous les **Widgets** (boutons, images, zone d'entrée de texte..) qui seront affichés au cours du Jeu, place ceux s'affichant en premier, et enfn lance le constructeur d'un **Jeu** (par son attribut Jeu* Game).