# Charte de développement

* convention de nommage en ‘CamelCase’ pour une meilleure compréhension (gangsneedingpots--> gangsNeedingPots) ;
* les noms de constantes sont en majuscule, les mots sont séparés par un caractère de soulignement ‘\_’ ;
* utilisation du caractère de soulignement ‘\_’ n’est autorisé que pour les constantes et en deuxième position des noms de champs privés ;
* les champs privés commencent par ‘m\_’ suivi du nom de la variable en ‘CamelCase’ (majuscule initial), on garde la convention d’origine ;
* les noms variables locales, paramètres de fonctions sont en ‘camelCase’ (avec minuscule initiale). Les variables locales peuvent être précédé d’un ‘l’ soit ‘l{CamelCaseName}’. De même les paramètres de fonction peuvent être précédé d’un ‘p’ soit ‘p{CamelCaseName}’.
* les noms des classes, structures, énumérations, propriétés et fonctions sont en ‘CamelCase’ (majuscule initial) ;
* les attributs génériques doivent être nommés de manière explicite ([TKey, TValue] au lieu de [T, U]) ;
* l’utilisation du type variant ‘var’ n’est autorisé que lors de l’utilisation de Linq ou type anonyme (permet la lecture du code lorsque l’IntelliSense n’est pas actif, dans le corps d’un email par exemple…) ;
* les noms de variables sont le plus explicite possible, préférer « gangMemberNum » au lieu de « num » ou « security » au lieu de « sec ».

# Refactoring

* proscrire tout champ public, les remplacer par des propriétés ;
* déplacer toute instruction en rapport avec l’IHM (couleur, taille de composant, etc) dans le projet dédié. Exposer ces éléments sous forme d’interface (pattern GOF et interface C#) utilisé par les processus du jeu pour échanger les informations de manière générique. ;
* déplacer toute instruction spécifique au système d’exploitation (accès fichier, …) vers le projet spécifique et implémenter ces appel grâce à des définitions d’interfaces (pattern GOF et interface C#) utilisé par les processus du jeu pour échanger les informations de manière générique ;
* extraire tous les textes du code pour les centraliser dans un container de ressource nécessaire pour une traduction intégrale du jeu
* .

# Travail en cours

* les attributs Skills et Stats ont été convertis en objet, conservant l’historique des valeurs (semaine précédente, année précédente) et permettant d’assurer que la valeur est toujours à l’intérieur des limites paramétrées, cette classe permet de fournir la valeur de base (sans modificateurs appliquée tel que drogues, objets, charmes) ainsi que la valeur après modificateurs appliquée, valeur courante après application des modificateurs. Des collections spécifiques ont été créé pour faciliter la gestion des attributs. Vois le projet Models pour un diagramme de classe de la nouvelle structure. Cette modification a permis de régler quelques erreurs d’accès aux tableaux d’attributs (enemy\_gang->m\_Skills[STAT\_CONSTITUTION]). La nouvelle structure empêche ce genre d’erreur ;
* déplacement de la notion d’argent (classe Gold) vers la classe Player. Cette classe dérivera d’une classe PlayerCaracter dont dérivera également la classe Rivale. Elle stockera toutes les propriétés et toutes les fonctions nécessaires à la vie du personnage. Par la suite ça permettra d’ajouter une IA aux Rivals pour ajouter un certain chalenge au jeu.
* les missions de gang ont été restructurées en une hiérarchie d’objet. Pour aouter une nouvelle mission :
  + Premièrement, ajouter une entrée à l’énumération GangMission,
  + Ensuite créer la classe de mission correspondante, elle doit hériter de la classe abstraite GangMissionBase,
  + Ensuite surcharger les fonction virtuelle PreExecuteJob, ExecuteJob et PostExecuteJob selon les besoins,
  + Enfin aouter à la fabrique GangMissionBase.GetMission(GangMission newMissionType, Gang gangToAssigne) l’instanciation de cette nouvelle classe ;

Pour qu’un gang effectue la mission, appelez la méthode DoWork : GangManager.foreach(Gang gang in playerGang) { gang.Mission.DoWork(); }

* déplacer l’inventaire de la classe Brothel vers la classe Player (et Rivals par extension)
* Remplacer les constantes EVENT\_{something} par une énumération EventType
* recoder la sérialisation XML (ressources, fichier de configuration, fichier de sauvegarde, fichiers externe) en utilisant les classe du C#.
* appliquer incom/outcom factor définit dans le fichier de configuration aux incoms/outcome de la classe GoldBase quand nécessaire et/ou absent (slave\_sales, brothel\_work, etc), ajout des income/outcome factor manquant au fichier de configuration tel que CreatureSales, BarIncome, etc.