# Development standard

* Naming convention is 'CamelCase' for better understanding (gangsneedingpots -> gangsNeedingPots).
* Constants names are in uppercase, words are separated by an underscore '\_'.
* Use of underscore '\_' is allowed only for constants and in second position of private field names.
* The private fields begin with 'm\_' followed by the name of the variable as 'CamelCase' (starting with an uppercase), keeping the original convention.
* Local variable names, function parameters are 'camelCase' (starting with a lowercase). Local variables can be preceded by an 'l' like 'l {CamelCaseName}'. Similarly, the function parameters can be preceded by a 'p' like 'p {CamelCaseName}'.
* The names of classes, structures, enumerations, properties and functions are in 'CamelCase' (starting with an uppercase).
* Generic attributes must be explicitly named ([TKey, TValue] instead of [T, U]).
* The use of type 'var' is allowed only when using Linq or anonymous type (to allow reading code when IntelliSense is not active, in the body of an email for example).
* Variable names are as explicit as possible, prefer "gangMemberNum" instead of "num" or "security" instead of "sec".

# Refactoring

* Prohibit all public fields, to replace them with properties.
* Move any instruction related to the HMI (color, component size, etc.) in the dedicated project. Expose these elements as an interface (GOF pattern and C# interface) used by the game's processes to exchange information generically.
* Move any operating system-specific instruction (file access, etc.) in the dedicated project. Expose these elements as an interface (GOF pattern and C # interface) used by the processes of the game to exchange information in a way Generic.
* Extract all the texts inline to centralize them in a resource container necessary for a complete translation of the game.

# Work in progress

* The attributes Skills and Stats have been converted into objects, keeping the history of the values (previous week, previous year) and making sure that the value is always within the parameterized limits, this class makes it possible to provide the value (Without modifiers applied such as drugs, objects, charms) as well as the current valueafter applied modifiers. Specific collections have been created to facilitate the management of attributes. See the Models project for a class diagram of the new structure. This modification made it possible to correct some access errors to the attribute tables (enemy\_gang->m\_Skills [STAT\_CONSTITUTION]). The new structure prevents this kind of error.
* Move the notion of money (Gold class) to the Player class. Player class will derive from a PlayerCaracter class from which the Rival class will also derive. It will store all properties and functions necessary for the character's life, to finally add an AI to the Rivals to get some more challenge to the game.
* Gang missions have been restructured into an object hierarchy. For a gang to perform the mission, call the DoWork method: GangManager.foreach (Gang gang in playerGang) {gang.Mission.DoWork (); }. To add a new mission:
  1. Add an entry to the GangMission enumeration.
  2. Create the corresponding mission class, it must inherit the abstract class GangMissionBase.
  3. Override the PreExecuteJob, ExecuteJob, and PostExecuteJob virtual functions as needed.
  4. Add to the GangMissionBase.GetMission factory (GangMission newMissionType, GangToAssign Gang) the instantiation of this new class.
* Move the inventory of the Brothel class to the Player class (and Rivals by extension), treat it as an item manager, an extended list of items to manage it. Control of inventory filling, control that two items of the same type cannot be worn simultaneously: two pairs of boots, automatic equipment of category of items according to the context: helmet, sword, armor for a combat, etc.)
* Replace the constants EVENT\_ {something} with an EventType enumeration
* Recode XML serialization (resources, configuration file, backup file, external files) using the C # classes.
* Apply income / outcome factor of configuration file to incomes / outcomes of GoldBase class when necessary and / or missing (slave\_sales, brothel\_work, etc), add the missing income / outcome factor to the configuration file such as CreatureSales, BarIncome , Etc.
* Isolate managers and encapsulate them in a Singleton to facilitate their access (global access), define their responsibilities and move the methods and properties in the most appropriate classes.
* The Brothel manager ensuring the overall management of the game becomes GameManager to avoid any confusion with BrothelManager managing the buildings.
* Extract all messages and store them to resources container. Set reference to the message by an identifier and automated loading from OS dependant project. Isolate the generation of reports in dedicated functions to centralize the work of writing by passing in parameter a structure containing the data necessary to this generation (contextual data).

# Norme 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 fonctions virtuelles 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), le traiter sous forme de gestionnaire d’items, une liste d’items étendu assurant la gestion des items et de l’inventaire (quantité d’un item, control du remplissage de l’inventaire, control que deux item de même type ne peuvent pas être porté en simultané : deux paire de bottes, équipement automatique de catégorie d’items en fonction du contexte : casque, épée, armure pour un combat, etc)
* 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.
* isoler les gestionnaires et les encapsuler dans un Template Singleton pour faciliter leur accès (accès global), définir leurs responsabilités, déplacer les méthodes dans les classes les plus adaptées.
* Le gestionnaire Brothel assurant la gestion globale du jeu devient Game pour éviter toute confusion avec un gestionnaire Brothel assurant la gestion des bâtiments.