# Personnel

<http://www.lequydonhanoi.edu.vn/upload_images/S%C3%A1ch%20ngo%E1%BA%A1i%20ng%E1%BB%AF/Rich%20Dad%20Poor%20Dad.pdf>

# Concept

## Idée générale

Un afficheur / gestionnaire de carte hexagonale pour Oltréé, qui doit permettre d’afficher la carte du jeu actuel, sélectionner chaque hexagone pour en voir les informations, et donner aux joueurs un visuel sur le territoire, pour informer et faciliter leurs décisions.

## Fini

### La Base / V1

Offre le minimum vital.

* Afficher la carte aux dimensions pré-fixées.
* Zoom / Dézoom / Déplacement de la carte
* Charger / sauvegarder
* Visualisation d’Hexagone:
  + Couleur & Image (Carte)
  + Degré d’exploration (Carte)
  + Label de coordonnées en accord avec celles de la carte initiale (Carte)
  + Description (Interface – si sélectionné seulement)
* Modification d’Hexagone :
  + Couleur
  + Image
  + Description
  + Degré d’exploration
* Sélection d’hexagone :
  + Par clic dans le polygone (Carte)
  + Applique un liseré vert pétant à l’intérieur (Carte)
  + Affiche la Description de l’Hexagone (Interface)
* 3 Modes :
  + Création de la carte : modification des éléments graphiques
  + Edition de la carte : modification des éléments descriptifs
  + Visualisation de la carte : sélection d’hexagone 🡺 visualisation des éléments descriptifs

#### TODO

##### DONE - Sélection d’hexagone : Clic + liseré + affiche description

* DONE Lors de la sélection d’un hexagone, ajout du liseré

##### DONE - Visualisation d’Hexagone: couleur, image, degré d’exploration & label des coordonnées (carte), description si sélectionné (interface)

* DONE Objet Hexagone :
  + Propriétés :
    - Potentiel ajout d’un nouveau type : HexPosition
    - Ajout de GetLabel au M (Colonne-Ligne), un Label au VM
    - Ajout de Description : texte M et VM
    - Ajout de Degré d’Exploration : entier (0 à 6) en M et VM
* DONE MVVM :
  + DONE Dans le TileEditor, rajout des nouveaux champs de l’hexagone :
    - Couleur & Image : noms
    - Degré d’exploration : x/6
    - Description : zone de texte
  + DONE Assurer son remplissage aisé sur source M, aisance de changement
* DONE Interface :
  + DONE Carte :
    - DONE Ajout du Label dans le Polygone, basé sur celui de honeycomb
    - DONE L’exploration doit être visible
  + DONE Dans l’interface gauche, ajout du V de l’hexagone, activé par la sélection d’un hexagone
  + DONE Rendre l’activation / désactivation de la modification lié à l’état de sélection

##### DONE - Modification d’Hexagone : Couleur, image, description, degré d’exploration

* DONE Modification de V et VM de l’Hexagone
  + Degré d’exploration : ajout de + / -
  + Description : devient un champ de texte modifiable

##### DONE - Afficher la carte aux dimensions pré-fixées.

* DONE Fournir une couleur de fond « foncée » pour chaque englobé, et une zone claire pour chaque englobant. Les englobants doivent donc prendre toute la hauteur / largeur disponible, et les englobés non (initialement).
* DONE Propagation des Width et Height des parents (les grid de GameView)
* DONE Une fois cela testé pour les 4 zones interface, mettre à -1 toutes sauf la gauche

##### DONE - Zoom / Dézoom / Déplacement de la carte

* DONE - Centrer la carte lors de son dessin
  + A retravailler
* DONE - Dessin des hexagones : à rendre plus robuste.
* DONE - Initialisation : pas de carte chargée.
* DONE - New 🡺 Nouvelle carte
  + DONE – Le dessin occupe toute la zone, l’hexagone 0-0 est collé en haut à gauche. Cela doit se trouver dans la fonction même de construction des dimensions et positions de l’Hexagone, la carte se contente d’avoir le bon format.
* DONE – Mouvements et zooms
  + DONE – L’information sur la position du centre est conservée à travers les mouvements.
  + DONE – Quand on zoome, on redessine, puis on compare la localisation du nouveau centre par rapport à l’ancien, et on en déduit le mouvement, qui est la différence.
* DONE – La carte doit être dessinée avec son centre au centre de la zone concernée
* DONE – Le contenant de la carte doit avoir une taille réelle suffisante pour y dessiner toute la carte
* DONE - Fournir 4 boutons de déplacement de la carte (ou un glisser déposé)
* Fournir un bouton de « retour au centre »
* DONE – ZIndex :
  + DONE – Ajouter un Enum des ZIndex
  + DONE – Y coller les différents Zindex « utiles »
* DONE Utiliser un évènement GeneralMapTransformation
  + Contient : XMove, YMove, ZoomMultiplicator
  + Reçu par HexMap, qui doit stocker les valeurs et comparer :
    - XMove / YMove : déplacement 🡺 mise à jour de XCenter, YCenter 🡺 Move sur tous les HexViewModel
    - ZoomMultiplicator: zoom 🡺 mise à jour de XCenter, YCenter, CellSize 🡺 UpdateDrawing sur tous les HexViewModel

##### DONE – Modes : Création / Edition / Visualisation

* DONE - Un enum
* DONE - Une propriété générale (Model, VM) liée à un élément de View pour l’afficher (liste déroulante, checkbox, onglet)
* DONE - Sur changement de Mode, envoi d’un message
* DONE - Des actions sur bouton gauche / droit différentes selon le Mode (vérification de la valeur en Model)
* DONE – Mise en place d’un vrai système définissant les actions à effectuer sur les clics gauche et droit de la souris.
  + DONE - La partie EventLogic ne doit faire que souscrire aux évènements, avoir les fonctions liées aux évènements, et appeler les fonctions de GameLogic.
  + DONE - GameLogic contient les fonctions qui « agissent », gèrent les sélections, lancent les actions, etc.
  + DONE - 2 fonctions, bouton gauche et bouton droit, faisant switch sur le mode de jeu, lançant la fonction concernée.
* DONE – Une action de changement de l’interface générale selon le mode (commencer par une Liste, mais si on peut avoir des Onglets bien gérés, ce serait impec)
  + DONE – EventLogic reçoit l’évènement de changement de mode, et lance une fonction UpdateGameMode dans GameLogic.
* DONE – Création de la carte (modification des éléments graphiques)
  + DONE – Gauche : application des données couleur / image en cours sur l’hexagone ciblé
  + DONE – Droite : sélection + récupération des données couleur / image de l’hexagone sélectionné
  + DONE – Etat interface
    - Toute la partie TileEditor est grisée
* DONE – Edition d’hexagones (modification des éléments descriptifs)
  + DONE – Gauche : sélection de l’hexagone + affichage des éléments descriptifs modifiables
    - Description
    - Degré d’exploration (ce qui se répercute sur la Carte)
  + DONE – Droit : NA
  + DONE – Sur sélection :
    - le champ Description est affiché, modifiable
    - les Couleurs et Images sont grisées (si elles peuvent continuer à être mises à jour avec ce qui est sélectionné) ou dissimulés et remplacés par des éléments indicatifs (sinon)
    - Conservation de la sélection ? Si oui, la description et l’exploration sont remplies
* DONE – Visualisation de la carte (sélection d’hexagone 🡺 visualisation des éléments descriptifs)
  + DONE – Gauche : sélection de l’hexagone + affichage des éléments descriptifs non modifiables
  + DONE – Droit : NA
  + DONE – Sur sélection :
    - Le champ Description est affiché, non modifiable
    - Les couleurs et images sont grisées ou dissimulées
    - Conservation de la sélection ? Si oui, la description et l’exploration sont remplies

##### DONE - Nouvelle carte – demander dimensions

* New > Demander dimensions carte (pré remplissage 10 x 10)
* Sur validation, création d’une carte idoine

##### DONE - Charger / sauvegarder

* DONE - Identique à ce que l’on a déjà
* DONE - Assure la non régression
* DONE - Régler le problème de bug GDI+ quand pon fait plusieurs New / Load de suite.

### Evolution – v1.1.0 – Forcer l’implémentation – 1

Etudier le cas des classes Partielles et Abstraites, pour identifier les usages possibles.

* <https://openclassrooms.com/courses/programmez-en-oriente-objet-avec-c/notions-avancees-de-poo-en-c-1>
  + Abstraite : ce qui fournit une base d’implémentation forçant à avoir les mêmes méthodes
    - Exemple : les ViewModel des conteneurs d’interface
  + Partielle : répartir une grande classe en plusieurs fichiers
    - Exemple : GameLogic, pour séparer les différents aspects.
    - Note : l’intérêt peut être limité. L’un des usages peut être la séparation logique. Un autre est d’y mettre l’implémentation d’une interface si elle prend trop de place.
* Classe virtuelle : peut-être réécrite

Classe sealed : ne peut pas être réécrite.

Le but est de créer des Interfaces permettant de planifier et forcer la structure du code, de façon à rendre l’ensemble « portable » pour d’autres types de cartes d’hexagones. Une Interface pour la carte, contenant telles et telles données (ou bien une classe à étendre si les données ne passent pas), une interface pour les Models, et pour les ViewModels – cette dernière pouvant demander un Model en entrée. Il serait potentiellement utile d’avoir aussi des classes à surcharger permettant d’automatiser certaines tâches des interfaces.

#### Idée

* Analyser et rassembler / faire une évolution à part à partir de : Interfaces & Héritage || Abstract, virtual, etc.: permet de « forcer » l’implémentation des modifications subséquentes (probablement abstract)
* Data > Model > ViewModel > View

Il est important de ne PAS modifier les fonctions et la structure en dehors de cela, le but étant de faciliter les futures modifications (qui créeront probablement de nouvelles classes abstraites)

#### TODO

* DONE – Créer les classes abstraites / Interfaces :
  + DONE – Model<Data>
  + DONE – ViewModel<Model>
  + DONE – IViewWithModel<ViewModel>
* DONE – Implémenter ces classes sur un premier élément
* DONE – Renseigner les fonctions des classes abstraites
* DONE – Etendre aux autres éléments
  + DONE – Par voie de fait, cela provoque la création de plusieurs sets de Data et Models (comme pour le menu).
  + DONE – Et donc une évolution de certains messages.
* DONE – Note : pas de Data pour les Panel 🡺 besoin ? Pas nécessairement.

### Evolution – v1.1.1 - Travail sur DATA / Libraries

#### Idée

Ne pas conserver les Data en mémoire :

* Les Data correspondent aux données en BDD – on n’a pas à les conserver en mémoire une fois la création effectuée, ça prend inutilement de la place. Il faut purger.
* Du coup, la sauvegarde « recréée » des Data.

Séparer Data de Libraries ? A terme, les Data seront les types renvoyés par les WS. Il faut réfléchir à comment les choses devraient s’organiser dans ce cas.

* + Il est possible que je doive mettre les Data à part, en vue de la migration avec des WS à terme.
  + Du coup, le contenu des Librairies devra être repensé
  + Peut-être scinder les Libraries : il y a du dessin, des évènements, etc. Ou peut-être pas. Mais alors mieux séparer l’ensemble à l’intérieur.

#### Réflexion

* Réflexion : quid de enum image type ?
  + <https://stackoverflow.com/questions/34826111/how-to-get-all-images-in-the-resources-as-list>
  + En récupérant les ressources, on récupère automatiquement les noms, donc on peut créer tous les TileImage et virer tileimagetype
  + L’enum peut être déplacé dans Libraries
  + Fonction dans libraries permettant de généérer un TileImage à partir de l’enum
  + Fonction get all
* Réflexion : Libraries ?
  + Contenu :
    - Toutes les fonctions
    - Tous les Enums ?
  + Hérite de Data
* Réflexion : Data
  + Contenu : Seulement les Data
  + Pas de fonction en dehors de la création
  + Data n’hérite d’aucun autre projet

#### TODO

* Modifier le fonctionnement de TileImage & TileImageType
  + DONE – TileImage : remplacer TileImageType par un TileID (string)
  + DONE – Supprimer TileImageType
  + DONE – Remplacer les usages de TileImageType par le string
  + DONE – Récupérer all image type remplacé par la récupération des ressources.
* DONE – Créer un projet DataLibrary
* DONE – Ajouter les objets Data :
  + DONE – HexData
  + DONE – HexMapData
  + DONE – MenuData
  + DONE – MenuItemData
  + DONE – TileImage
  + DONE – TileColorData
  + DONE – TileData
* DONE – Renommer HexDrawingData en HexDrawingInformations

### Evolution – v1.1.2 - Refondre la séparation des projets

#### Idée

* Toute la partie graphique, les « contrôles », devrait être dans l’Application.

#### TODO

* DONE – Déplacer tout le contenu de Controls.Library dans VersionBase
* DONE – Dialogs doit être au même niveau que les dossiers MVVM.

### Evolution – v1.1.3 - Séparer UI et Carte

#### TODO

* DONE – Renommer l’interface en UI (préfixe ou suffixe)
* DONE – Créer un ensemble MVVM pour l’UI
* DONE – GameXXXX devient ApplicationXXXX, contenant UIXXXX et HexMapXXXXs
* DONE – Création d’un GameData contenant, pour le moment, seulement HexMapData
* DONE – GameData est chargée et sauvée, recréée à chaque Nouvelle carte
* DONE – UIData est créée une seule fois
* DONE – On ne sauvegarde pas UIData

### Evolution – v1.1.4 - Steamlined Save

#### Idée

* Réfléchir à la méthodologie de sauvegarde. Par exemple, l’affaire des Deregister / Register dans et hors de l’ApplyModel / ImportData
* Disposer d’un modèle général peut être utile, une interface pour chaque niveau « sauvegardable »
* Ajouter à Model<TData> une fonction TData SaveToData

#### TODO

* DONE – « Purifier » les Data
  + DONE – Les Data doivent avoir le minimum de données, et ces données être de types simples, ou d’autres Data
  + DONE – Virer les Fonctions des Data : les Data ne doivent contenir que des données
  + DONE – Les constructeurs des Data doivent les construire à partir de données « simples », comme provenant d’une BDD, et de données Data.
    - DONE – Pour Menu, il faut fournir la donnée du MenuClickedMessage, et re-créer un message qui les récupère.
    - DONE – MeniItemData et MenuData 🡺 UI
* DONE – Les fonctions de sauvegarde doivent être placées de façon logique :
  + DONE – Créer un SaveHelper qui regroupera toutes les fonctions de sauvegarde, qui s’appelleront entre elles
* DONE – On ne doit pas conserver les Data post création :
  + DONE – Bien remplir le helper de création des Data
  + DONE – Les Data doivent pouvoir être créées et supprimées de façon pérenne. S’il le faut, ajouter une fonction de suppression (ils ne sont pas conservés).
  + DONE – Chaque création / sauvegarde / load doit se faire avec un nouveau set de Data.
* DONE – Divers
  + DONE – Enlever le préfixe UI des modèles de couleur et d’image – ce ne sont des UI qu’en tant que VM.
  + DONE – AbstractModel devient IModel, une interface
  + DONE – Doit-on supprimer quelque chose côté VM des hexagones quand on charge une nouvelle carte ? 🡺 non, c’est le polygone qui contient tout.

### Evolution – v1.1.5 - Limiting Image in Code

#### Idée

HexData, HexModel et HexViewModel ne devraient avoir aucun lien avec les TileColorXXX et TileImageXXX autre que leur ID.

Il faut rendre l’accès aux TileXXXYYY facile – du statique ? En tout cas, via ID.

OU BIEN : en s’assurant que les TileXxxYyy ne contiennent pas de donnée « lourde », ça peut passer, MAIS : on devrait quand même ne faire que référencer (donc ID ou ref). Cohérence avec côté « BDD » pour le DATA, légèreté pour le reste.

* A quel moment est-ce que l’Image est importante ?
  + Quand on construit les TileImageViewModel
  + Quand on change l’image d’un Hexagone au niveau ViewModel.
* Il n’est donc pas utile d’avoir une Image avant le ViewModel.

#### TODO

* DONE - Mettre « à disposition » les TileImage et TileColor par leur ID.
  + DONE - La liste des Images doit être disponible, via message 🡺 nouveau GetBitmapImageByNameMessage, entrée image name, sortie BitmapImage, via HexMapDrawingHelper.GetBitmapImageFromTileImageData.
  + DONE - Ce serait bien de sauvegarder les BitmapImage déjà récupérées. Une classe ImageHolder ?
* DONE - TileImageModel : virer Bitmap et BitmapImage
* TileImageViewModel :
  + DONE - A la création, appeler GetBitmapImageByNameMessage pour récupérer le BitmapImage
  + DONE - Utiliser GenerateTileImageSource pour avoir une couleur de fond (vert clair)
  + DONE - L’ImageSource sera utilisée à la place du BitmapImage
* DONE - HexViewModel > UpdateTileData
  + DONE - A la place d’un Bitmap, demander image name.
  + DONE - Récupérer l’image bitmap via GetBitmapImageByNameMessage
* DONE - Propager les modifications

## En Cours

### Evolution – Communautés 1/x

#### Idées

* Objet Communauté
  + Nom
  + Type
  + Taille
  + Nombre et type des Habitants
  + Description
* Création sur carte
  + Un bouton « Ajouter Communauté »
  + Bouton cliqué + clic sur carte 🡺 Fenêtre de création qui reprend les données ci-dessus
* Sélection sur carte 🡺 interface de visualisation / modification, reprenant les informations

#### TODO

* Note :
  + Il faut bouger les points des Drawing informations ET les polygones. Ça, c’est fait.
  + Quand on bouge un polygone, on lui recréée un ensemble de points « bougés » 🡺 ce serait plus simple de bouger d’abord les DrawingInformations, puis de fournir les points bougés, plutôt qu’un mouvement. (on peut garder l’ancienne fonction en la renommant, si on veut garder de l’historique)
  + Quand on zoome, on update le CellRadius (ce qui régénère les points), on recentre la HexMap (ce qui bouge tous les centres, et régénère tous les points, puis bouge tous les polygones), avant de faire un dernier mouvement (ce qui re-bouge tous les centres, régénère les points, et bouge tous les polygones)
  + Fuck it, optimisation. Fonctions de « dessin » qui manipule seulement les centres, etc
* Test carte :
  + DONE - Bouton « Add »
  + « Add » + clic 🡺 ajout point noir au centre de l’hexagone
  + DONE - Ajout sur le lieu exact de clic
  + Ajout sur le point le plus proche
  + Affichage fenêtre sur clic, validation 🡺 ajout.
  + Objet ajouté dispose d’un nom « Bob » en dessous
  + Fenêtre demande nom, validation 🡺 ajout avec nom
  + Bouton cancel de la fenêtre
  + Bouton valider seulement si nom fourni
* Community Data / Model
  + Nom : texte
  + Type : texte
  + Taille : entier
  + Nombre et type des Habitants : entier, texte
  + Description : texte
* Création sur carte
  + Un bouton « Ajouter Communauté »
  + Bouton cliqué + clic sur carte 🡺 Fenêtre de création qui reprend les données ci-dessus
  + L’ajout ne se fait que sur validation
  + Pour l’identification / sauvegarde des points : le clic est pris dans un Hex, donc on peut récupérer son centre, l’éloignement X et Y du centre, et le pourcentage de Width et Height correspondant. Cela fait donc : Column, Row, %Width, %Height. Ensuite l’objet créé aura ses propres fonctions de « redessin ».
  + A noter qu’une Communauté peut se permettre d’être intégrée à un Hex, mais que, pour viser long terme / ligne, l’en sortir peut être une bonne idée.
* Sélection sur carte 🡺 interface de visualisation / modification, reprenant les informations
* Modes de jeu
  + Création : Placement d’une communauté, pas de description
  + XXXXX
* Enum CommunityType
  + City / Cité
  + Town / Ville
  + Burg / Bourgade
  + Village / Village
  + AgriculturalDomain / Domaine Agricole
* Enum divers :
  + AgriculturalDomainActivity
    - AnimalHusbandryType
    - MineType
    - QuarryType
  + VillageType
  + BurgType
  + CityTownType
* Les enums liés aux communautés

## Planification

### Roadmap

* Arriver à l’état de la carte des joueurs :
  + Communautés 1 :
    - Permet la mise en place des points, rendant Lignes plus facile.
    - Début de complexification de l’interface gauche : ce qui est disponible pour la création va différer de ce qui l’est pour la modification / visualisation
  + Lignes 1
* A cette étape, j’ai une Map correspondant à ce que les joueurs ont. Il faut donc faire attention à ce que les modifications suivantes puissent être utilisées sans détruire une éventuelle sauvegarde (même si l’export par Obsidian Portal permet de limiter la casse).
* Ajout des « aides » simples :
  + Patrouille 1 : permet d’avoir le visuel de composition du groupe, les ressources emportées, et la consommation
  + Calendrier 1 : permet d’avoir la date actuelle du jeu.
  + Météo 1 : permet de générer rapidement le temps.
* Cette partie-là verra probablement l’UI droite apparaître.

### Evolutions Fonctionnelles

#### Evolution – Types de Terrain 1/x

En s’inspirant des terrains d’Hexographer, créer des Types de Terrain fixes et les proposer au lieu des Couleurs & Images.

#### Evolution – Lignes 1/2

* Création d’une Ligne :
  + Mode de placement (point cliqué, proche sommet, point fixe)
  + Nom (qui sera placé au milieu du segment milieu)
  + Définition d’une Couleur (liste habituelle, probablement à généraliser donc) et d’une Largeur (liste statique)
  + Définition des points par clics :
    - Sur le point cliqué, simple
    - Sur le « sommet polygonal » le plus proche (on prend le polygone cliqué, on cherche son point le plus proche)
    - Sur le « point fixe » le plus proche : sommet polygonal, centre polygone, milieu des côtés, milieux sommet-centre.
* Modification d’une Ligne :
  + Mode de placement (point cliqué, proche sommet, point fixe)
  + Nom
  + Couleur
  + Largeur
  + Note : Pas de déplacement des points
* Sélection d’une ligne :
  + Par clic directement dessus
  + Indiqué par un liseré vert

#### Evolution – Lignes 2/2

* <http://csharphelper.com/blog/2014/10/let-the-user-draw-polygons-move-them-and-add-points-to-them-in-c/>
* Déplacement des points d’une ligne :
  + Sélection de la ligne 🡺 modification des « Points » de façon à les rendre plus facilement sélectionnables
  + Déplacement du point + relâchement : selon Mode.

#### Evolution – Calendrier 1/x

* Elément Calendrier – Top
  + Jour, Mois, Année
  + Clic dessus pour afficher une fenêtre Calendrier
* Fenêtre Calendrier
  + Indique mois et année
  + Tableau des Jours
    - Indication du jour en cours
  + Boutons pour avancer / reculer dans les jours
* Pas d’agenda

#### Evolution – Calendrier 2/x

* Elément Agenda
  + Contient des jours
    - Date
    - Description
  + Seuls les jours où quelque chose a été renseigné existent
* Sur le Calendrier
  + Les jours avec une correspondance dans l’Agenda apparaissent différemment, cliquables.
  + Cliquer sur un Jour Agenda ouvre une autre fenêtre (ou modifie celle du Calendrier), et afficher la Date et la Description

#### Evolution – Météo 1/x

* Possibilité d’afficher une interface Météo
  + On fournit la Saison et le Temps Actuel
  + Génère, sur clic, la Météo suivante
    - Temps
    - Durée (s’il y a)
  + Prend en compte les Evènements météorologiques particuliers
  + Indique les effets météorologiques

#### Evolution – Météo 2/x

* Lien entre l’Interface Météo et le Calendrier / Agenda
  + Ajout d’une donnée Météo dans les Jours de l’Agenda (ce qui créée un Jour s’il n’y en a pas)
  + Un bouton « Inscrire dans l’Agenda » permet d’ajouter une météo donnée sur un jour donné.
  + Il peut être utile de rendre visible un symbole météo sur les jours du calendrier – si existant. Sinon, un indice identifiant que le jour a, ou non, une Météo remplie
* Réfléchir sur l’utilité d’une fonctionnalité de remplissage automatique sur toute une période (et qui ne surcharge pas une météo déjà remplie)

#### Evolution – Météo 3/x

* Lien entre Météo et Déplacements
  + Tableau de coûts en déplacements selon la météo
  + Prendre en compte les modes de déplacement : route embouée, neige, etc.

Evolution – Météo 4/x

* Effets Météo cumulatifs :
  + Compteurs incrémentables pour les « situations » :
    - Neige : augmente par jour de neige, reste dans certaines conditions, etc.
    - Pluie : autant une pluie fine n’a guère d’effet, autant des pluies diluviennes impactent divers éléments
    - Chaleur : jour après jour de sécheresse a un impact
  + Ces compteurs doivent avoir des effets, probablement par « crans »
    - Handicaps de déplacement
    - Hausse de la consommation de ressources
    - Dégâts de fatigue
    - Impacts sur les bâtiments / productions / populations
    - Etc.

#### Evolution – Patrouilleur 1/x

* Elément Patrouilleur
  + Informations générales
    - Nom
    - Joueur
    - Peuple (liste)
    - Motivation (liste)
    - Initiation (liste)
  + Vocations & Sauvegardes
    - Vocations
    - Sauvegardes (sous forme de bonus)
  + Carnet de Patrouille
    - DV, PV max, PM max
    - Cartes
    - Vigilance
    - Déni
    - CA
    - Initiative
  + Métiers
    - Un tableau avec tous les métiers, score, bouton +/-
    - Tous les métiers ont un score, même à 0
  + Harnois : des cases à cocher
  + Langues : un tableau, deux cases chacune (écrit / parlé)
  + Ressources immobilisées
* Interface « Feuille de Personnage »
  + Mode « Edition »
  + Mode « visualisation »
* Cette 1e version n’a pas à afficher grand-chose en dehors du nom de chaque élément, ni fournir de calcul automatique
* Elle permet par contre de renseigner l’objet Patrouille

#### Evolution – Patrouilleur 2/x

* Elément Patrouilleur
  + Informations générales :
    - Initiation : fournit les points lors de la Création, ainsi que les Cartes.
  + Vocations & Sauvegardes
    - Vocations : boutons + / - avec Source
    - Sauvegardes : boutons + / - avec Source (sauf le -)
  + Carnet de Patrouille
    - PV max : fournis par le joueur, seulement à la Création et en cas de Gain de DV
    - PM max : calcul auto
    - Cartes : automatiquement, par initiation. Possible de faire en sorte qu’en cas de Gain de DV, il y ait choix Carte / normal.
    - Vigilance (calcul auto)
    - CA : à partir des sources sur la fiche (armure, bouclier, guerrier, custode), et faire le calcul automatique
    - Initiative : calcul auto
  + Métiers
    - Tableau 🡺 Liste : un bouton Ajout + Source
    - Eléments déjà ajoutés : boutons + / - & Source.
  + Harnois : des cases à cocher
  + Langues : un tableau, deux cases chacune (écrit / parlé)
  + Ressources immobilisées (le max étant calculé auto)
  + Notes – les Sources :
    - Gains : Création, Dé de vie, Objectif de Motivation, Carte d’Exaltation, Trésor, Guerrier (pour les sauvegardes).
    - Pertes : erreur (n’enregistre rien, supprime juste), perte de Dé de Vie, autre (champ à remplir)
    - Cet aspect implique de sauvegarder un historique des modifications
* Interface « Feuille de Personnage »
  + Mode Création, qui va demander le nombre de Vocations & Métiers bonus, et ne pas demander de Source.
  + Mode « Edition », pour la modification « en jeu »
    - Il peut être utile d’avoir des boutons « Objectif de patrouille réussi », « Objectif de Motivation Réussi », « Carte d’exaltation – Vocation », « Carte d’Exaltation – Métier », « Trésor – Vocation », « Trésor – Métier »
    - Ces boutons fourniront un certain nombre de « points d’amélioration » des types idoines, ouvriront la modification des PV, etc.
    - C’est vraiment secondaire
  + Mode « visualisation complète »
  + Mode « visualisation simple » (identique à la précédente)
* Ajout majeur : automatisation

#### Evolution – Patrouilleur 3/x

Le but ici est d’ajouter la gestion de certains métiers à sélection simple :

* Druide : liturgies chtoniennes (sélection simple)
* Exorciste : exorcismes (sélection simple)
* Maître de guerre : ordres de bataille (sélection simple)

#### Evolution – Patrouilleur 4/x

Le but est de s’occuper des métiers à sélection moyenne

* Aberrant : pouvoirs (sélection simple, possible de la perdre) et faiblesses (sélection simple, impossible à perdre)
* Marqué : type de marque (sélection définitive, seule la suppression pour Erreur permet de s’en débarrasser) et marques (sélection définitive, suppression dernière en date)
* Prêtre : panthéon (sélection définitive, seul la suppression pour Erreur permet de s’en débarrasser), divinité (sélection définitive, suppression dernière en date)

#### Evolution – Patrouilleur 4/x

Le but est de s’occuper des métiers à sélection complexe

* Alchimiste : différents types de préparations
* Magicien : les sorts, avec obligation des rangs inférieurs, prix en mana, etc.

#### Evolution – Patrouilleur 6/x

Le but est de s’occuper des métiers ajoutant des créatures au personnage

* Chevalier : montures, leurs améliorations
* Maître des bêtes : compagnons animaux, leurs augmentations
* Noble : suivants, type, DV, etc.

#### Evolution – Patrouille 1/x

* Ajout d’un élément « Patrouille »
  + Patrouilleurs : nombre de Patrouilleurs, de Montures, d’animaux de bât, de packs de suivants
  + Ressources transportables et transportées – à remplir à la main (Interface)
  + Consommation journalière – calcul automatique
* Dans l’interface
  + Indique et modifie le nombre de Patrouilleurs, de Montures, d’animaux de bât, de packs de suivants (Interface)
  + Indique et modifie Ressources transportables et transportées – à remplir à la main (Interface)
  + Indique la consommation journalière – calcul automatique (Interface)
  + Bouton « Fin de journée », « Consommer X », « Combat », « Blessé », « Rajout X » (Interface)

#### Evolution – Patrouille 2/x

* Lie les Patrouilleurs à l’objet Patrouille, qui désormais utilise ces informations pour renseigner tout ou partie de ses données

#### Evolution – Patrouille 3/x

* Rajout d’un élément sur la carte
  + Pion (avec image ?)
  + Déplaçable (bouton Interface)
  + Indication de la sélection (liseré vert)

#### Evolution – Déplacement & Actions 1/x

Rendre le déplacement de la Patrouille conforme aux règles

* La Patrouille dispose d’un total de Points d’Actions (PA)
* Chaque Type de Terrain doit avoir un « coût » en PA
* Le déplacement d’un hexagone à un autre coûte la moyenne de PA entre la source et la cible
* Les différentes actions d’Exploration doivent être implémentées (chasse, etc.), et leurs effets de base rappelés, coûts en actions aussi
  + Voyage normal (1 carte, x1)
  + Chasse (2 cartes, x2)
  + Observation (2 cartes, x1)
  + Exploration prudente (1 carte, 2g1 jets, x2)
  + Marche forcée (déplacement entier normal +1, jet +n+1/hex parcourus, x2 PR)
  + Fouille (0 déplacement, toute la journée, jet < degré sécurisé)
* Bouton pour « passer la nuit », sans action associée

#### Evolution – Déplacement & Actions 2/x

* Fournir un plus large éventail d’actions
  + Dangers nocturnes 🡺 action de Passer la nuit en zone marquée « dangereuse »
  + Attendre
  + Récolter des rumeurs (automatique avec certaines actions)
* « Passer la nuit »
  + Demande s’il y a hébergement (impact potentiel des éléments, des Dangers Nocturnes, etc.)
  + Nourriture fournie ou pas ?
  + Selon les choix, impacte les réserves, tire une carte, etc.

#### Evolution – Communautés 1/x

* Objet Communauté
  + Nom
  + Type
  + Taille
  + Nombre et type des Habitants
  + Description
* Création sur carte
  + Un bouton « Ajouter Communauté »
  + Bouton cliqué + clic sur carte 🡺 Fenêtre de création qui reprend les données ci-dessus
* Sélection sur carte 🡺 interface de visualisation / modification, reprenant les informations

#### Evolution – Communautés 2/x

* Relation, avant poste, etc.
* Types plus détaillés en accord avec les règles
* Factions
* Dirigeants & PNJs

#### Evolution –

### Evolutions Code

#### Evolution – Plus belle IHM

##### Idée

Mettre des couleurs tolérables, améliorer le style général, la taille des éléments, le texte, etc.

##### TODO

* Changer la couleur du fond de la carte en un gris foncé
* Changer la couleur du fond de l’élément de gauche en un gris clair.
* Changer la couleur extérieure des polygones de déplacement / zoom en blanc, et l’intérieur en gris clair

#### Evolution –

### Evolutions Techniques

#### Evolution – Site 1/x : Créer un serveur Web

* http://www.finalclap.com/faq/266-serveur-web-hebergement-maison-chez-soi
* https://www.assistance.bouyguestelecom.fr/internet-bbox/services-bbox/heberger-serveurs-dns-dynamique-pro
* Avoir possiblement un moyen d'accéder directement à ce "serveur" (remote desktop par exemple)
* Note : si j’utilise mon PC portable comme serveur, je n’y ai certes accès que quand il est activé, mais c’est une bonne 1e étape pour tester, avant de reporter la chose sur le PC fixe

#### Evolution – Site 2/x : Créer un BDD

Disposer d’une BDD accessible en jeu, mutuelle, permettant de sauvegarder / recharger la carte. Hébergée en local.

* <http://blog.mounier-logiciels.fr/comment-creer-une-connection-mysql-local-en-CSharp.html>
* Serveur Apache
* MySQL

#### Evolution – Site 3/x : Lien Code / BDD

Utiliser un lien BDD / Code qui automatise les choses.

* Utiliser Entity Framework, en mode « code first »
* http://pmusso.developpez.com/tutoriels/dotnet/entity-framework/introduction/
* https://msdn.microsoft.com/fr-fr/library/gg696172(v=vs.103).aspx
* Cela permettra d'implémenter les Data correctement - mais il est possible de s'en inspirer à l'avance. Un projet à part, donc
* Il faut prendre en compte qu’avec une BDD unique, ce qui est fourni aux joueurs ne DOIT PAS permettre la modification
* Une BDD unique, du code Open Source, il faut un MDP « général » aussi (ou disposer d’exports réguliers de la BDD)

#### Idée  – Normes & Checklist (probablement à découper)

Il serait bon d’avoir des normes claires à respecter.

* Le Complément du Guide contient des informations à ce sujet. Point 4.4 – Normes supplémentaires.
* La Checklist (chapitre 5, complément du guide) peut s’avérer précieuse.
* Mes Data sont des Entités.
* L’Architecture de Fixe doit être analysée avec un gros grain de sel : une bonne partie est probablement contre-productive vu la taille de mon projet.
* Modifier la classe par défaut (entre autres défauts) permet de garantir une structure cohérente : <https://stackoverflow.com/questions/8745688/how-can-i-change-the-default-visual-studio-c-sharp-new-class-file-template>
  + Par exemple, des Régions avec les différents types d’éléments à ajouter

#### Idée  – Des Etats / Enum plus étendus (exemple de découpage)

* Cf. Machine à états, page 8 du Complément du guide, et page 26.
* En dehors de la création d’un AbstractEtat et de son implémentation pour fournir des fonctions à un état, il faudra probablement attendre d’avoir travaillé dessus pour comprendre ce qui est nécessaire – et récupérer le code des machines à état.
* Un Etat d’objet a 2 objets paramètres, l’Objet dont il est l’état et l’Enum état.
* Un simple Enum peut utiliser cette méthodologie, mais son Objet n’aura qu’un paramètre, l’Enum.

#### Idée  – Interfaces & Héritages

Le but est de créer des Interfaces permettant de planifier et forcer la structure du code, de façon à rendre l’ensemble « portable » pour d’autres types de cartes d’hexagones. Une Interface pour la carte, contenant telles et telles données (ou bien une classe à étendre si les données ne passent pas), une interface pour les Models, et pour les ViewModels – cette dernière pouvant demander un Model en entrée. Il serait potentiellement utile d’avoir aussi des classes à surcharger permettant d’automatiser certaines tâches des interfaces.

#### Evolution –

### Evolutions Bonnes Pratiques

#### Idée  1 – Mettre en place SonarCloud pour faire de l'Analyse Statique de programmes

Mettre en place SonarCloud, gratuit pour les projets Open Source, et très rapide à configurer

* https://fr.wikipedia.org/wiki/Analyse\_statique\_de\_programmes
* L’analyse statique du code sera effectuée par l’outil SonarQube : https://fr.wikipedia.org/wiki/SonarQube , https://www.sonarqube.org/
* SonarLint est une extension pour VisualStudio permettant d’avoir un retour directement dans l’IDE de l’évaluation des règles définies dans SonarQube : <http://www.sonarlint.org/visualstudio/index.html>
* SonarCloud est gratuit pour les projets Open Source

#### Idée  2 – Installer GoCD pour l’Intégration continue

* Lecture - http://skalp.developpez.com/traductions/martin-fowler-integration-continue/#LV
* Lecture - https://openclassrooms.com/courses/integration-continue
* Lecture - https://stackoverflow.com/questions/1991071/continuous-integration-for-a-small-net-open-source-project
* Outils : https://www.gocd.org, Visual Studio Team Services
* Au travail, ils utilisent Team Foundation Server (qui est salement payant)

#### Idée  – Test Driven Developpement

* Définition : https://fr.wikipedia.org/wiki/Test\_driven\_development
* Introduction : http://igm.univ-mlv.fr/~dr/XPOSE2009/TDD/pagesHTML/PresentationTDD.html
* Mise en œuvre du TDD dans le cycle de développement : http://www.les-traducteurs-agiles.org/tdd/2015/01/06/les-cycles-du-developpement-pilote-par-les-tests.html
* Exemple concret d’application du TDD : http://bruno-orsier.developpez.com/tutoriels/TDD/pentaminos/
* Test unitaire : https://fr.wikipedia.org/wiki/Test\_unitaire
* Framework de tests unitaires : NUnit
* Mock : https://fr.wikipedia.org/wiki/Mock\_(programmation\_orient%C3%A9e\_objet)
* Librairie pour faciliter Mock : MOQ
* Tuto : <https://openclassrooms.com/courses/programmez-en-oriente-objet-avec-c/les-tests-unitaires-5>

#### Idée  – Installer un Système de génération de documentation

* Doxygen : https://fr.wikipedia.org/wiki/Doxygen

#### Evolution –

# Idées générales à relire, traiter, et rajouter dans le Doc :

* D'après le document IBM 0.9.2 & le code :
  + Ce pourrait être une bonne idée que certains évènements contiennent l'objet qui les a généré (mais c'est déjà le cas pour plusieurs évènements)
  + Différenciation XxxxEvent / XxxxCommand ? (tous deux sont des objets Event, mais une Commande représente une commande lancée ?)
  + Réfléchir à avoir une Application englobant le projet
* Lectures diverses :
  + GIT : https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices
  + https://www.gocd.org/2017/11/28/confessions-continuous-delivery-experts-gocd-open-source.html
* Relire https://blog.rsuter.com/recommendations-best-practices-implementing-mvvm-xaml-net-applications/
  + Implémenter mieux les Commands, de façon à avoir un fonctionnement générique ?
  + A relire régulièrement au fur et à mesure de l'implémentation de nouvelles logiques
  + Chercher d'autres documents de best practices
* Se renseigner sur WCF : https://fr.wikipedia.org/wiki/Windows\_Communication\_Foundation

# Roadmap (récupérée)

* [X] Merge couleur et image
* [X] Terrain = image OU couleur + image, liste
* [X] Virer la lettre
* [X] Virer la limitation de taille
* [X] Virer le Switch Terrain
* [X] Interface gauche
* [X] Menu déroulant Terrain
* [X] Clic + Terrain = changement terrain cible
* [X] Mettre en place impérativement une communication par évènements, tester sur clic et récupération selected tileimage et tilecolor :
  + [X] Choix : https://rachel53461.wordpress.com/2011/10/09/simplifying-prisms-eventaggregator/
  + EventAggregator https://stackoverflow.com/questions/36621679/communication-between-two-user-controls-one-contains-the-other-with-mvvm
  + Communication entre viewmodel https://stackoverflow.com/questions/19686382/wpf-communication-between-user-controls
  + MVC, les ViewModel sont les controller, qui gèrent les actions ? https://softwareengineering.stackexchange.com/questions/352481/mvvm-communicate-between-views
* [X] Déplacer CellSize de HexData vers HexDrawingData
* [X] Réfléchir à ce que doit contenir HexData, HexDrawingData, TileData, et où ils doivent être les uns par rapport aux autres, (cf. ci-dessous aussi). Peut être le dessin doit il toujours fournir un CellSize ? CellSize dans un HexModel ?
* [X] Faire en sorte que les objets de base - HexData, HexDrawingData, etc. - n'aient que des types de donnée simples (ex: au lieu d'avoir les Color en donnée, n'avoir que le ARGB en string)
* [X] Faut-il virer le Tag du Polygon, ce qui évite d'avoir à fournir les coordinate au lieu du drawing data ?
* [X] Création d'un objet Map, contenant les données de dimension, la liste des Hex, etc., tout ce qui est nécessaire pour la gestion de la Map (un Model, probablement ?)
* [X] Possibilité de passer par référence les hexagones / polygones ? Vérifier qu'il y a unicité. Si oui, un objet HexModel peut être utile, non ?
* [X] Mettre en place un vrai MVVM au niveau des models et des view models : il faut que les M&VM parents référencent les M&VM enfants, que les M soient utilisé dans la construction des VM : VM(M) ! Et donc, les deux listes déroulantes devraient s'appeler List, avec un VM(M), et un M contenant la liste.
  + [X] HexModel : virer le HexData
  + [X] HexModel : le HexDrawingData a sa place ailleurs que dans Libraries ?
  + [X] TileColorModel : virer le tilecolor, prendre le ARGB
  + [X] TileImageTypeModel : virer le TileImageType, conserver le reste
* [X] Possibilité d'écouter le changement d'une donnée dans le ViewModel ? Comme un Selected, dans le setter ?
* [X] Ajouter les Ressources > Images dans le projet (où ?), et les utiliser (path relatif)
* [X] Interface UI : afficher le type de terrain sélectionné sur clic droit (sélectionner dans listes ?)
* [X] Lire https://blog.rsuter.com/recommendations-best-practices-implementing-mvvm-xaml-net-applications/
  + [X] Mettre en place une ViewModelBase et une INotifyPropertyChanged base classes, qui implémentent les fonctions nécessaires
  + [X] Toujours assigner le ViewModel en DataContext
  + [X] Ajouter Mytoolkit au projet : https://github.com/RSuter/MyToolkit/blob/master/README.md - le récupérer par NuGet
  + [X] Sa logique de commande ==> Menu ?
  + [X] Ne pas set le datacontext par binding dans xaml qui instancie la vue; Cf son code ==> méthode perso : get set vm dans le view, view.viewmodel = xxx pour le rajouter
  + [X] Explorer sa méthode de comm' pr message, surtout callback ! => à mettre en place d'abord dans le clic sur polygon pour les échanges
  + [X] Le choix entre user control et templated control peut attendre d'avoir plusieurs applications
* [ ] Interface UI : Menu en haut
  + [X] Première implémentation
  + [X] Menu en haut : récupérer le bouton cliqué , Sinon une fonction différente par bouton
  + [X] Menu en haut : command = msg
  + [ ] Menu en haut : gérer la structure du menu par un fichier de configuration ? Clairement pas urgent
* [X] Lien Models <=> Data : on récupère Data, on créée les Models, mais met-on à jour Data quand Model change ? Idéalement, V => VM => M
* [X] ABSOLUMENT : un système pour gérer toutes ces commandes lancées de partout. On va possiblement devoir disposer de classes supplémentaires pour composer l'application. On a déjà le système des EventMessage d'un côté, EventLogic de l'autre. Il faudrait rassembler les fonctions ainsi : EventMessage les messages, EventLogic les liens EventMessage / GameLogic, et GameLogic les fonctions
* [X] Sauvegarde / rechargement de Map par Sérialisation
* [ ] Préparation passage BDD
  + [ ] Récupération enum par string (pour BDD)
  + [ ] Fichier "constant" pour les couleurs et les images, gestion par ID
* [X] Faire un MVVM pour la Carte
* [ ] Faire un MVVM pour l'Interface Gauche (LeftPanel) (au lieu de TileEditor)
  + [X] UserControl LeftPanel (simple pour le moment)
  + [ ] Menu déroulant contenant les "modes". Edition Hex, Selection Hex, Edition Road/River, etc.
  + [ ] Rajouter un champ de texte contenant la Description
* [ ] Rivières et Routes
  + [ ] Réfléchir à comment enregistrer les données de localisation des points d'une rivière. Possiblement s'organiser avec les points centraux comme référence, et des fonctions de traduction
  + [ ] River(Data,Model,ViewModel,View)
  + [ ] Road(Data,Model,ViewModel,View)
  + [ ] Dessiner Rivière, route
  + [ ] Sélection Rivière, route
  + [ ] Rivière : possibilité d'identifier les hexagones où elle passe ? La direction ?
* [ ] Au lancement, demander dimensions (et type terrain de base ?)
* [ ] Regénération d'une carte (avec demande dimensions)
* [ ] Créer un objet TileSet, contenant la liste des types de terrain, référencé dans Map ?
* [ ] Ajouter toutes les tiles d'origine dans l'objet TileSet (interface ?), et en faire une version pour chaque set de tiles que j'ai, et faire un Enum + ue fonction pour récupérer le bon
* [ ] Séparer les fonctions de Save / Load / New dans un autre fichier
* [ ] Redimensionnement avec boutons +/-
* [ ] Redimensionnement en cas de redimensionnement de fenêtre