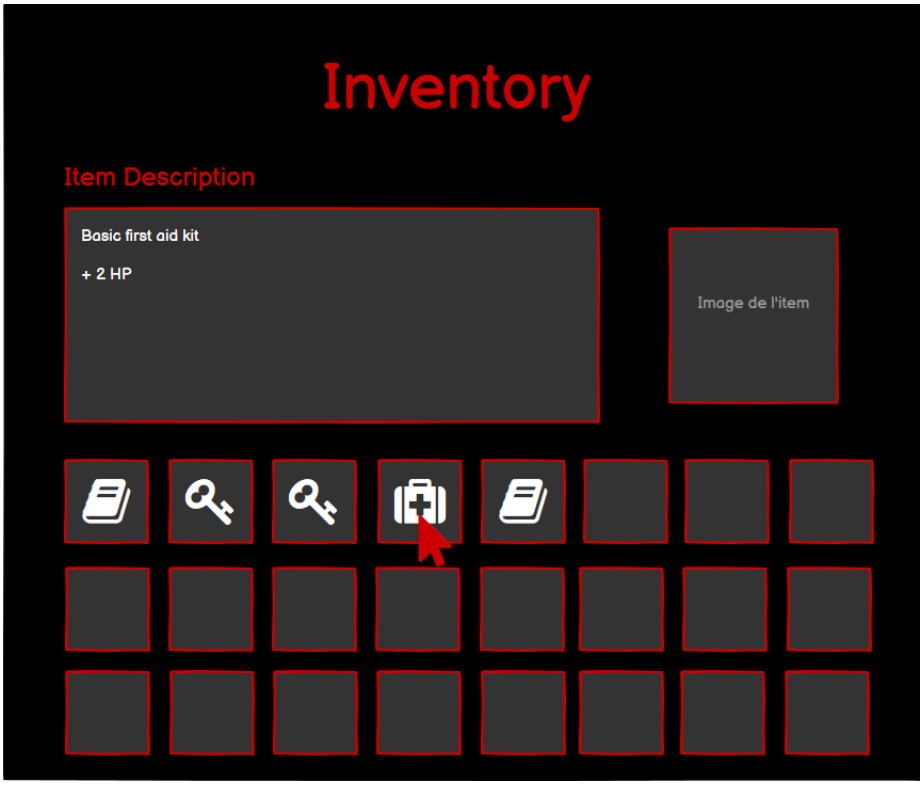
1 - Maquettes

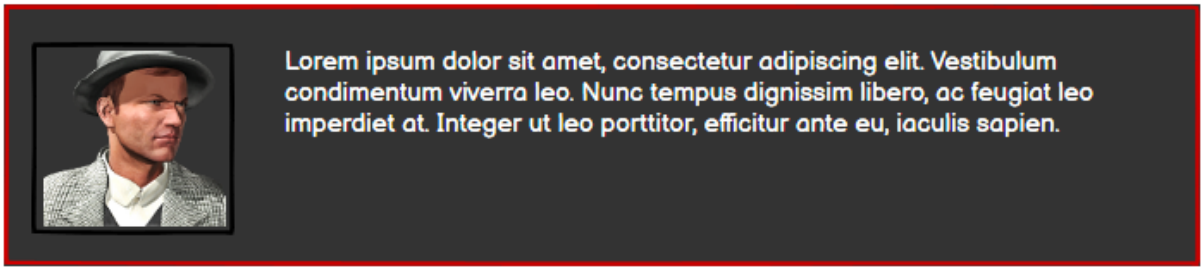
1.1 Menu Principal

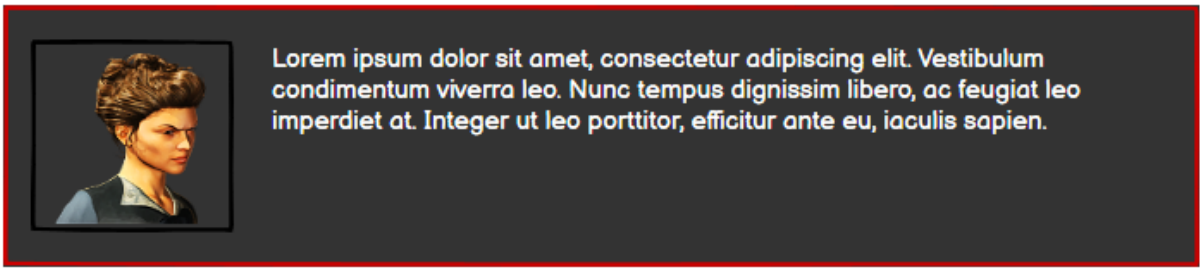
1.2 Menu d'inventaire



1.3 Journal

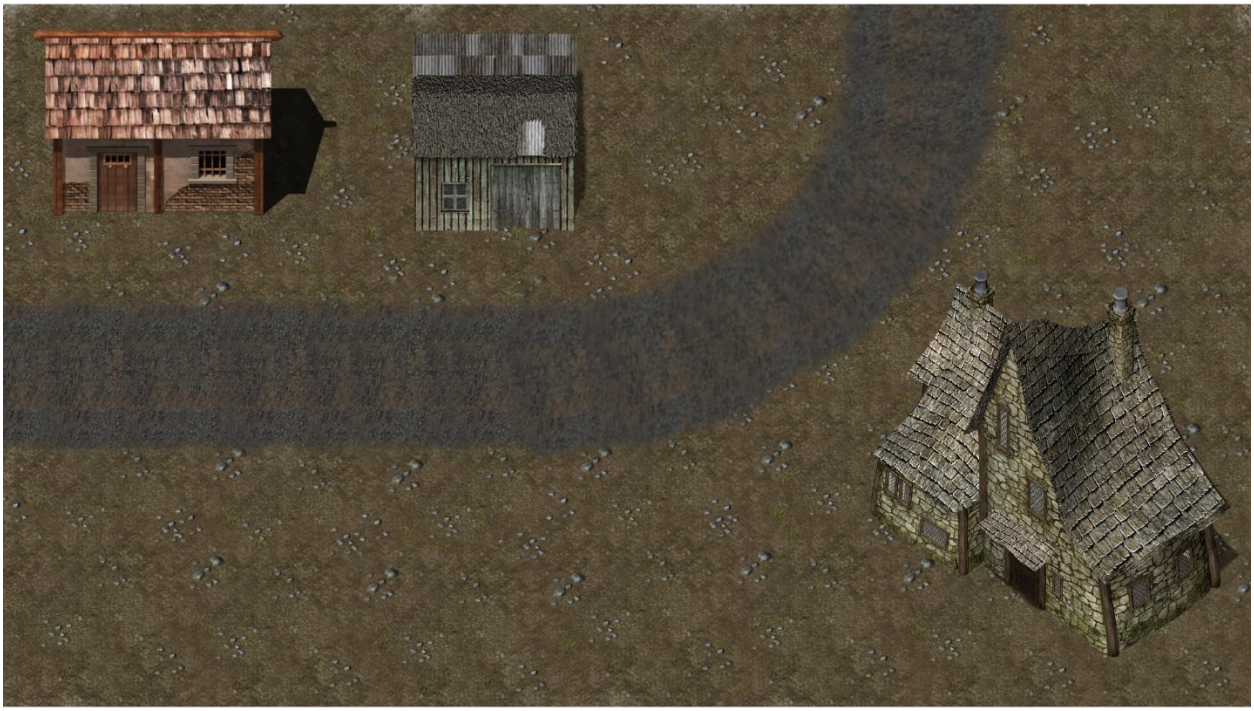
1.4 Boîtes de dialogues





Les prochaines maquettes donnent un aperçu des maps, extérieures et intérieures, dans lesquelles le joueurs se déplacera. À noter qu'il ne s'agit que des exemples et que des objets (meubles, arbres, maisons, etc) seront rajoutés par la suite dans Unity.

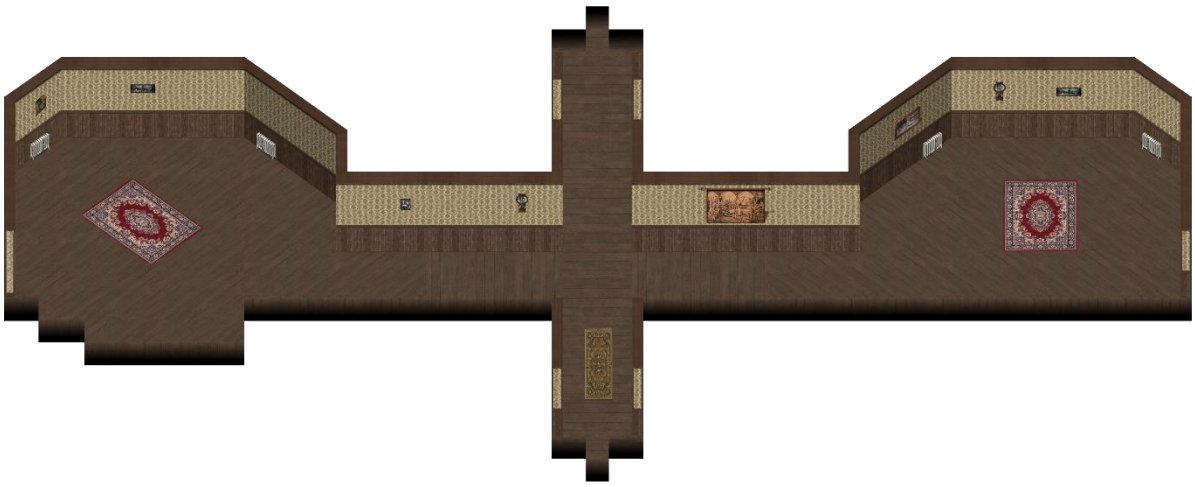
1.5 Town Entrance



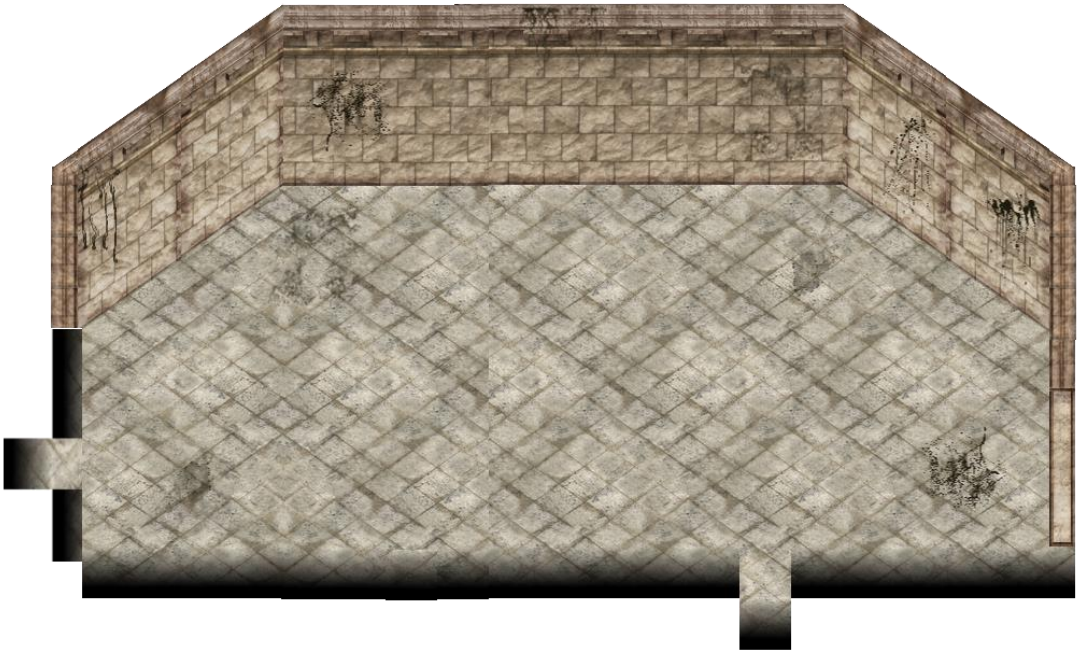
1.6 Asylum Exterior

1.7 Church Exterior

1.8 Asylum Front

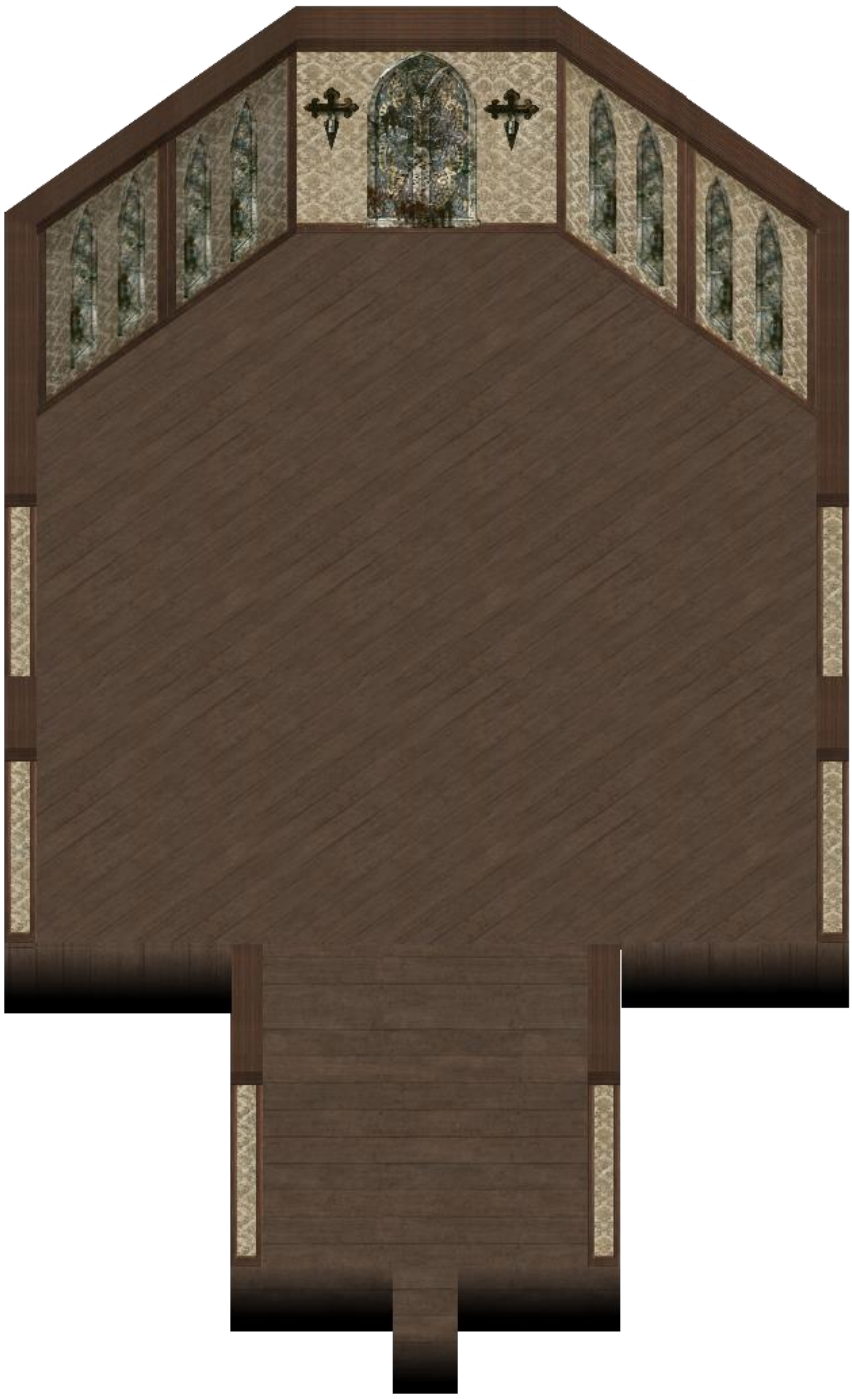


1.8 Asylum Back Right

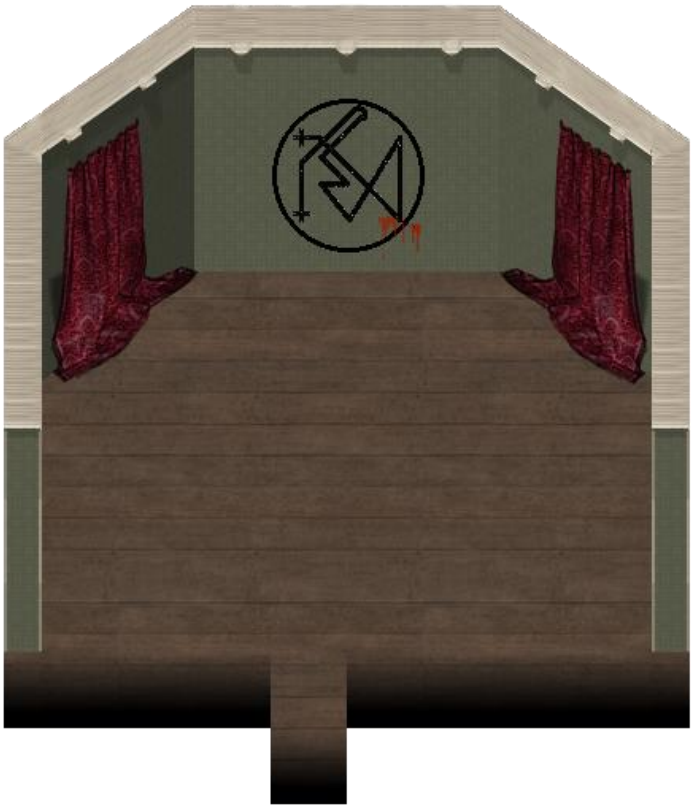


1.9 Asylum Back Left

1.10 Church



1.11 Church Secret Room



2- Design Patterns

2.1 – Factory (PrefabManager)

Le concept de Prefab dans Unity permet de créer, de configurer (par l'ajout de Components et de différents scripts), ainsi que de sauvegarder des objets afin de les réutiliser. Ces objets pré-configurés deviennent donc des clones du prefab original, ce qui facilite grandement l'instanciation dynamique des GameObjects qui doivent se comporter de la même façon. Par exemple, inutile de créer (*new*) et de configurer un objet de type Bullet à chaque fois qu'un personnage utilise une arme à feu, ou un objet de type Enemy pour « ennemi générique #6 » : une simple copie suffit.

La classe PrefabManager agira donc à titre de *factory* et permettra de gérer ces instanciations dynamiques de GameObjects (balles, ennemis, effets spéciaux tels que explosions ou magie, etc), peu importe le type d'objets demandé. Cela permettra à la fois de réutiliser ces mêmes prefabs dans différentes scènes.

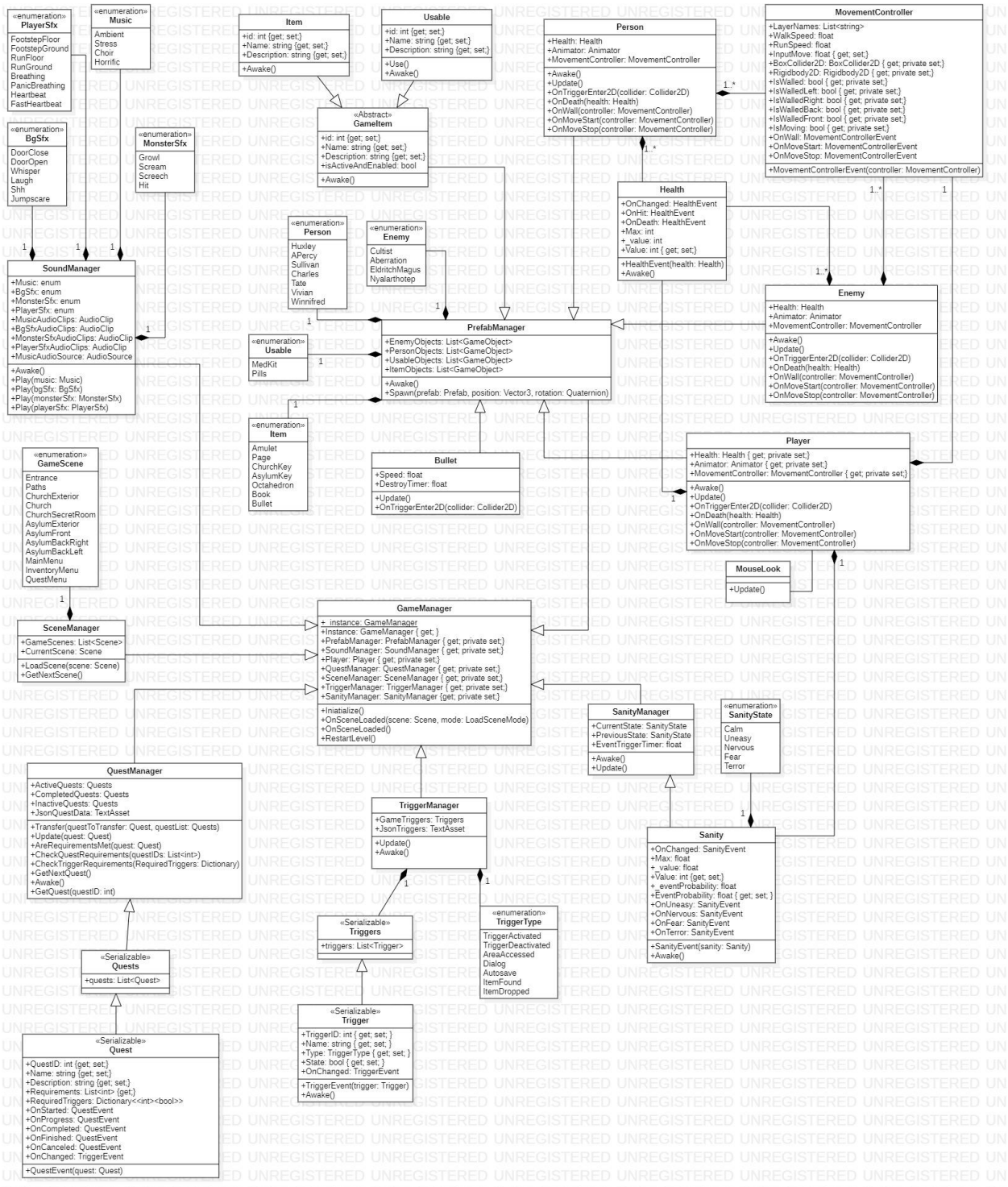
2.2 – Observers

Plusieurs évènements *in-game* devront être monitorés et gérés, parfois de manière différente. Il y aura notamment des *HealthEvents* à l'affût des changements dans le nombre de points de vie, si les personnages ont été touchés ou s'ils sont morts. Ces évènements devront être monitorés pour le joueur et pour tous les NPCs. Il y aura également des *TriggerEvents* (porte ouverte par le joueur, objet trouvé, etc), des *QuestEvents* (en progression, complétée, etc), des *SanityEvents* (selon x niveau de santé mentale). Vérifier et mettre à jour toutes ces données dans une *for loop* serait très lourd.

À l'aide de *delegate functions*, on observe et agit seulement lorsque l'évènement observé se produit sur l'objet concerné : si le joueur se fait toucher/tirer, seul le *OnHit HealthEvent* du joueur est appelé et mis à jour. Toutes ces fonctions *delegate* doivent être définies dans chaque type d'objet qui les utilisent, ce qui permet un grand contrôle et une granularité dans le code. Un *Trigger* de type *DoorOpen* pourrait provoquer X s'il s'agit de la porte A, mais provoquer Y s'il s'agit plutôt de la porte B.

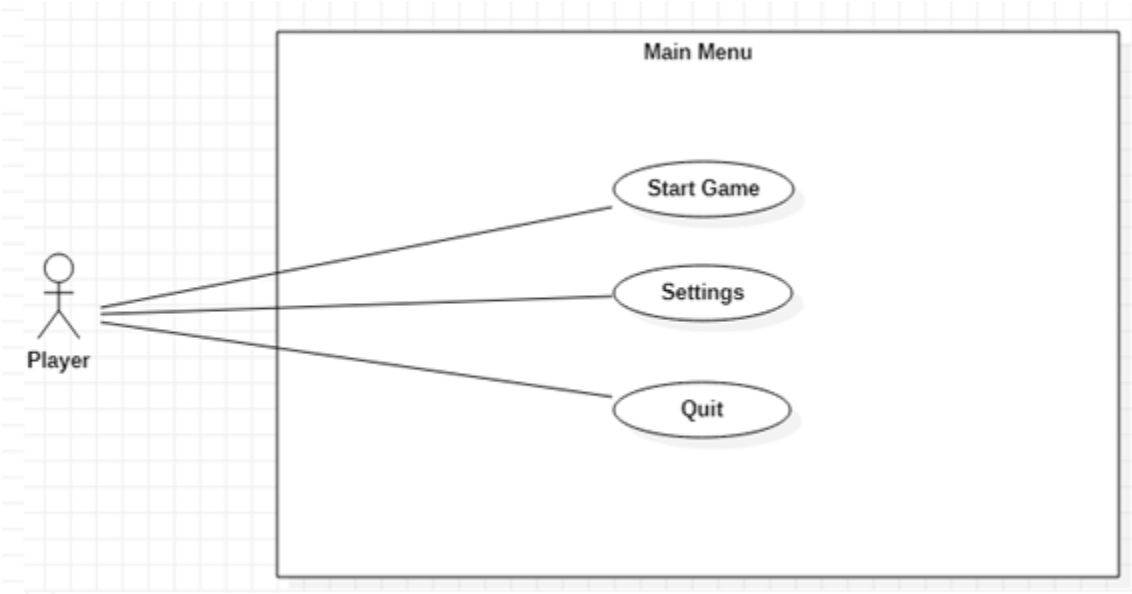
3– UML

3.1 Diagramme de classes

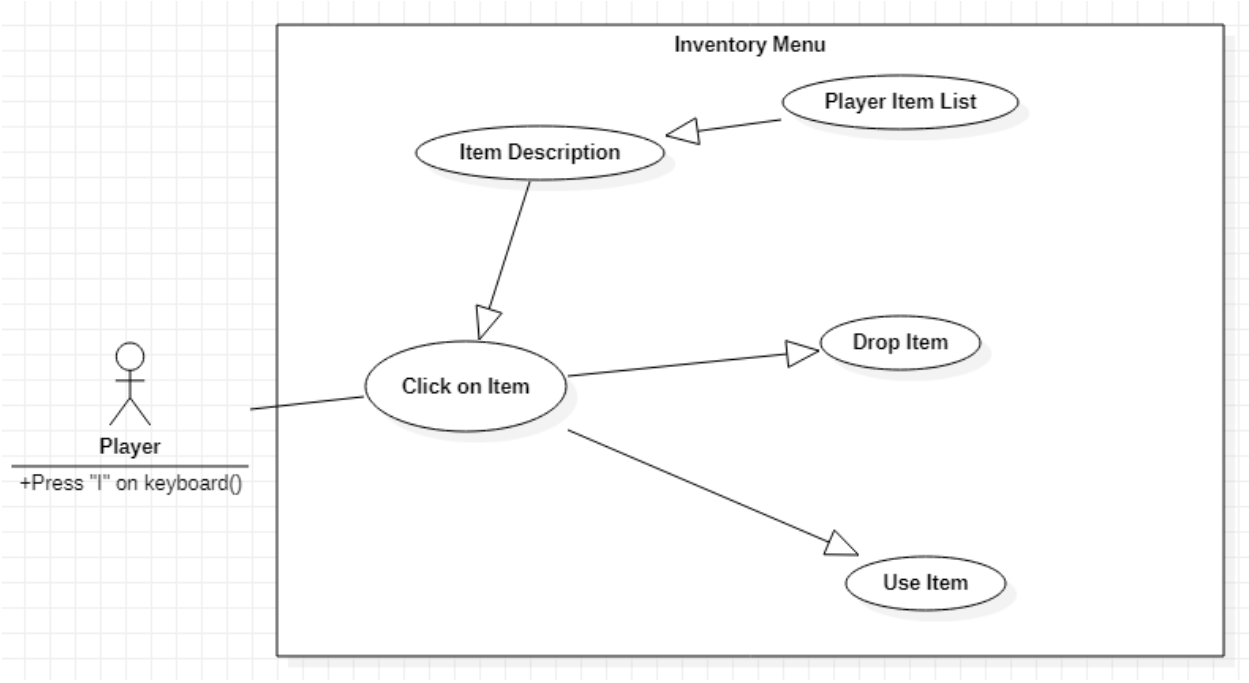


3.2 Diagramme de cas d'usages

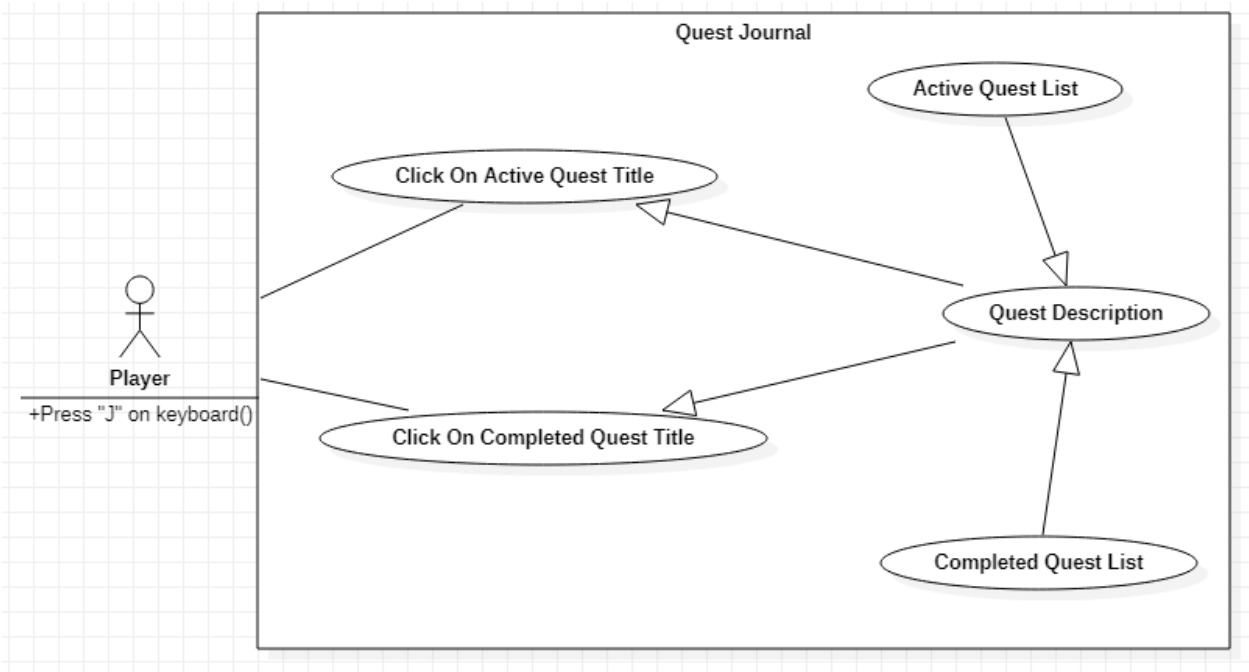
3.2.1 Main Menu



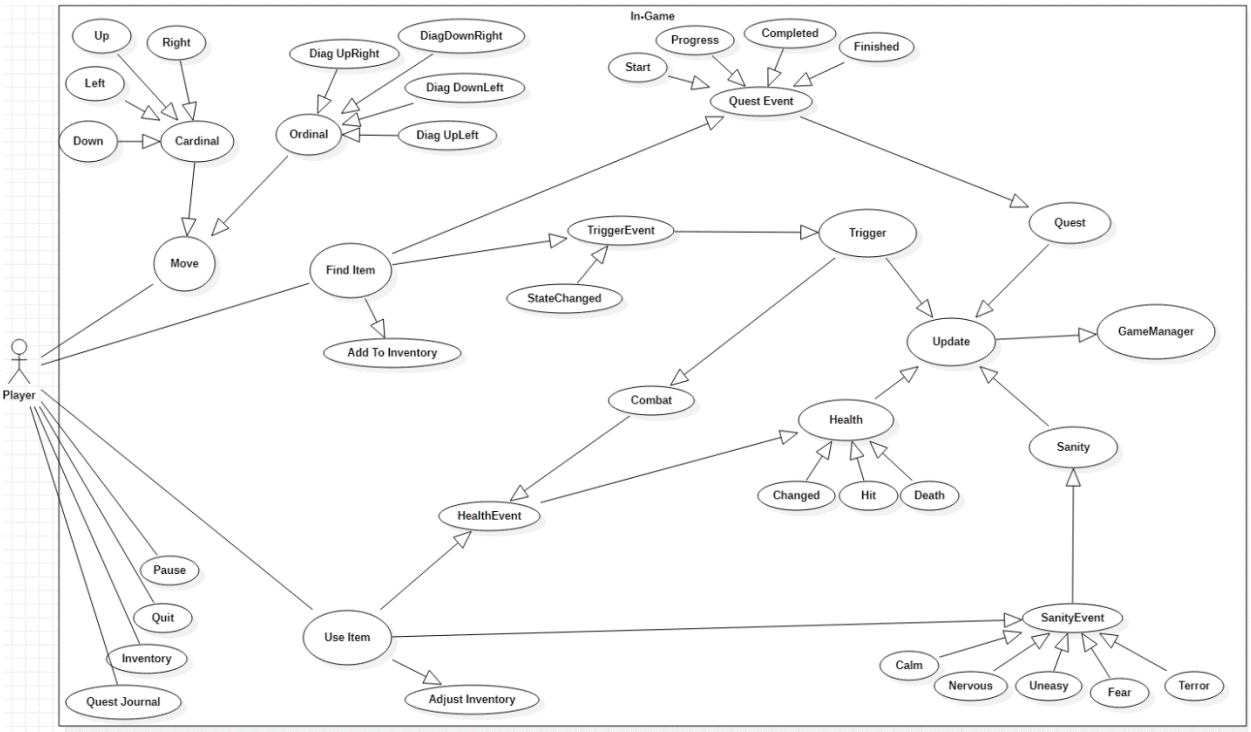
3.2.2 Inventory Menu



3.2.3 Quest Menu

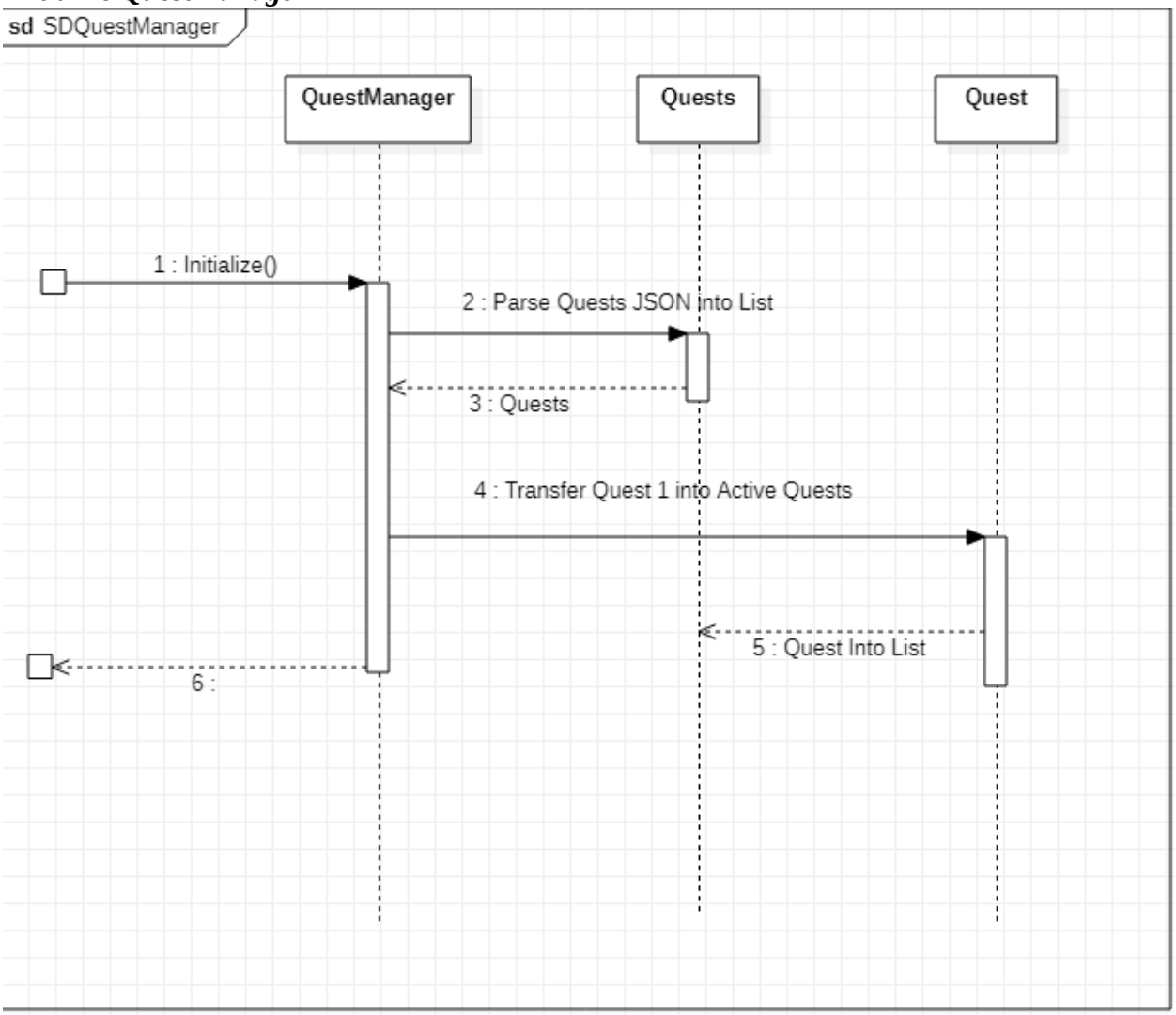


3.2.4 In-Game

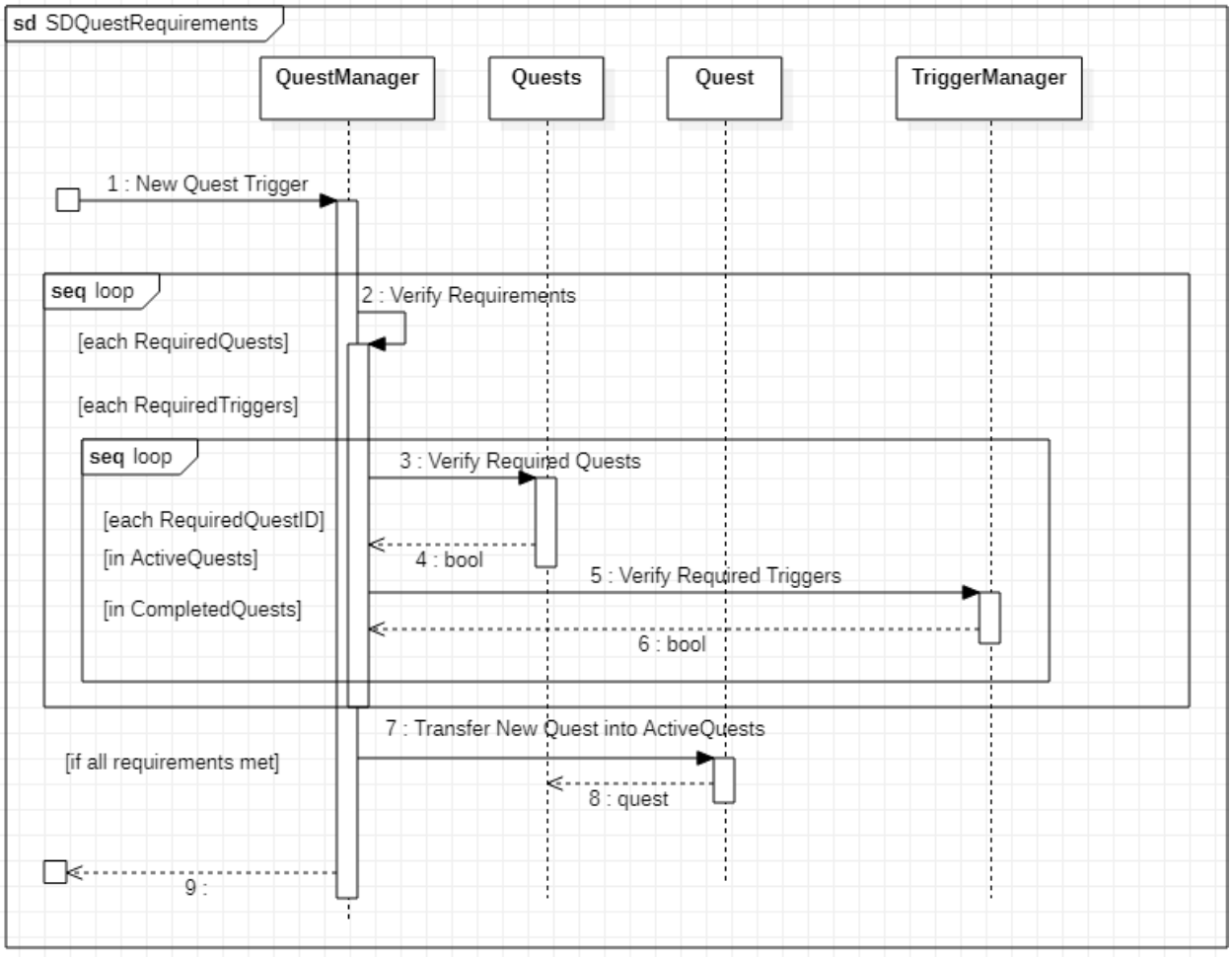


3.3 Diagramme de séquence

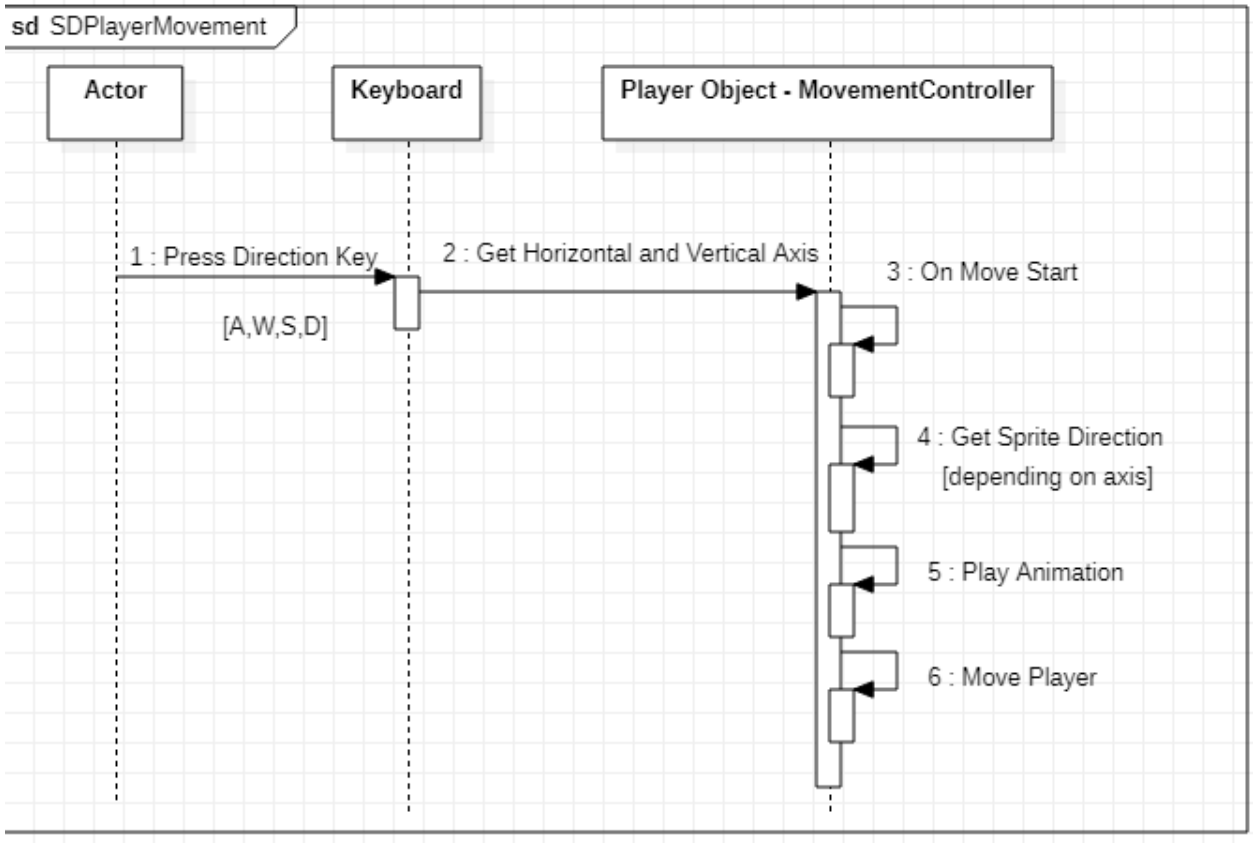
3.3.1 Initialize QuestManager



3.3.2 QuestManager Verify if new quest can be unlocked



3.3.3 Player Movement



4- Structure de données externe

La structure de données externe sera sous forme de documents JSON, qui seront lus lors du démarrage du jeu. Un des documents rassemblera les informations nécessaires à la création de chaque *Quest* du jeu : ID, nom, description, ainsi que les numéros d'IDs des *Quests* et des *Triggers* requis pour être débloquée. Les *Triggers* seront sous forme de dictionnaire, afin d'inclure leur état requis avec un booléen.

```
{
   "quests":[
      {
         "QuestID:":"1",
         "Name":"Find the key to the church",
         "description":"The Church's doors are locked tight, and it seems the windows are boarded up as well. There should be a key around here somewhere.",
         "requirements":[

         ],
         "triggers":[
            {
               "id":"4",
               "state":"false"
            },
            {
               "id":"5",
               "state":"true"
            }
         ]
      },
      {
         "QuestID:":"2",
         "Name":"Find the key to the Asylum",
         "description":"The note I found about Doctor Danforth was somewhat bizarre. I should find the key to the Asylum, and then search his office.",
         "requirements":[
            "1"
         ],
         "triggers":[

         ]
      }
   ]
}
```

Le second document JSON rassemblera de façon similaire les données des *Triggers* et incluera le type de chacun. Il est possible que d'autres documents JSON s'ajoutent au fil du temps.

```
{
    "triggers": [
        {
          "id:": "1",
          "name": "ChurchDoor",
          "type": "DoorOpen",
          "state": "false"
        },
        {
          "id:": "2",
          "name": "ChurchKey",
          "type": "ItemFound",
          "state": "false"
        }

    ]
}
```

5- Sources et références

1- Exercices en classe et code dans le cadre du cours Développement de jeux vidéos C63

2- https://gist.github.com/grofit/651efe441472da4e29fa
   - Concernant la structure de *quest system*, comment séparer tous ses éléments

3- https://gameprogrammingpatterns.com/observer.html
   - Concernant les Observers en jeux vidéos, avec exemples de code.

4- https://eternaldarkness.fandom.com/wiki/Sanity_Effects
   - Concernant le fonctionnement d'un Sanity Meter et les effets/events que cela peut engendrer

5- HP Lovecraft