Retour sur la séance 1:

La première séance était une découverte de l'objectif et du java en général. Une approche naïve d'un problème permet souvent d'en cerner les enjeux et les obstacles.

Il n'était pas encore question de résoudre tous les problèmes, seulement de les rencontrer pour apprécier les solutions que vous apprendrez par la suite. J'ai tout de même été surpris de voir un grand nombre d'entre vous passer le premier boss et adopter des programmes inventifs. Félicitations :)

- **Premier enjeu: Enregistrer les valeurs**

Pour commencer à donner des instructions, vous devez normalement être capable d'enregistrer les valeurs du plateau pour vos traitements ultérieurs. La plupart d'entre vous avez déjà eu une bonne intuition en créant votre premier objet: Site.

```
class Site {

    public int id;
    public int x;
    public int y;
    public int radius;

}
```

Actualisons notre class Player pour prendre en compte cet état. On va ici se concentrer uniquement sur le début

```
class Player {

    public static Map<Integer, Site> listeSite = new HashMap<>();

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int numSites = in.nextInt();
        for (int i = 0; i < numSites; i++) {
            int siteId = in.nextInt();
            int x = in.nextInt();
            int y = in.nextInt();
            int radius = in.nextInt();

            Site site = new Site();
            site.x = x;
            site.y = y;
            site.id = siteId;
```

```
            site.radius = radius;

            listeSite.put(siteId, site);

        }

        while (true) {
              // votre code ici
        }
    }
}
```

Ici une petite difficulté: Votre méthode main étant static (elle ne demande pas qu'on instancie l'objet -> Personne n'a fait "new Player()", c'est obligatoire pour la méthode main) elle ne peut utiliser que des attribut d'objet static.

- **Etablir une stratégie**

Il semble que construire un bâtiment au tour 1 et engager une escouade de soldat ensuite soit une bonne idée pour passer le premier boss. :P

Séance 2: Modéliser la situation

Le premier avantage de l'objet est qu'il permet de modéliser le monde de façon intuitive. Si notre premier essai permet de retenir les valeurs et de les enregistrer dans une Map, cette approche va rapidement arriver à ses limites.

Aujourd'hui l'objectif est de conceptualiser le problème et d'établir un premier modèle qui décrit la situation!

Comme c'est votre premier TP, je vais un peu vous guider en vous proposant une solution. Vous retrouverez peut-être certaines de vos idées de la semaine dernière?

- **<u>Des indices de liens?</u>**

En étudiant les différents concepts que le jeu semble nous présenter on voit rapidement émerger deux thèmes que l'on a envie de regrouper. Pour l'exercice ici je vais me baser sur les consignes à la sortie du tutoriel, en ligue bronze. Ca vous fera gagner du temps (mais en réalité si votre modélisation est bien faite, vous verrez que ça ne coûte rien d'ajouter des objets)

Reine, Caserne-Chevalier, Archer, Mine, Tour, Géant, Caserne-Archer, Caserne-Géant

Comment relier ces différents objets entre eux?
Quels sont les comportements communs entre eux?

Astuce: Il est possible de décrire des concepts sous entendu mais pas exprimés dans cette liste. Généralement quand on procède ainsi, on a un indice qu'on parle d'une classe abstraite.

exemple: Une voiture et un camion semblent reliés par la class véhicule. Personne n'aurait idée d'instancier un véhicule, qui décrit un concept et pas un objet. Mais impossible de se passer de cette notion pour décrire votre modèle.

- **Des indices de différences?**

En regardant la carte, on remarque rapidement des sites qui peuvent être soit vide, soit transformés via la reine en bâtiment. Ils semblent partager énormément de points communs comme des coordonnées et un radius. On pourrait facilement penser qu'en réalité, un site est juste un "bâtiment vide", c'est-à-dire que Site hériterait de Bâtiment.

Mais cette hypothèse rencontre rapidement deux soucis:

- param1 et param2 ne semble servir à rien dans le cas d'un site. Ils renvoient simplement -1, qui ressemble même au code erreur en C++. Un objet hérité qui n'utilise pas les valeurs de son parent, c'est souvent le signe que le lien est erroné.
- Le comportement, c'est-à-dire ce qu'on modélise via les méthodes est complètement différent. L'intérêt de l'héritage est de pouvoir utiliser les méthodes sur les différents descendants, et lorsqu'on doit coder une fonction "produire(){ // ne fait rien} c'est très souvent le signe d'un problème dans la relation.

Quelle est la bonne relation entre le site et le bâtiment?

- **Un petit conseil:**

Les énums sont un outil incroyable pour décrire certains concepts, mais on a tendance à le confondre avec l'héritage. Pour savoir si c'est un objet, il faut regarder si il a un comportement: Une voiture va avoir une fonction avancer et freiner, c'est donc un objet. La couleur de la voiture ne va avoir aucun comportement, elle sera rouge ou verte, c'est donc une enum.

Ouverture au prochain TP

Vous verrez la semaine prochaine les interfaces qui sont encore un autre type de liens. Les interfaces décrivent des comportements communs entre des objets différents.

Un petit problème qui vous aidera à comprendre ce que je veux dire: La libellule hérite de l'insecte, et le pigeon de l'oiseau, mais les deux peuvent voler. Comment éviter de dupliquer cette fonction partout dans le code? Comment éviter de donner à la fourmi ou un pingouin une fonction “vol(){// ne vole pas}”?

L'interface résout ce souci.