odalités

Type : Soutenance orale
Durée : 15 minutes
Date : 3ème semaine de janvier
Responsable de l'évaluation : Adrien Krähe

Critères

À titre indicatif, voici le genre de critèr
l'évaluation :

- Répartition du temps de parole
- Présentation de la conception haut-niveau
    - Choix et cohérence du schéma
    - Clarté du discours
- Présentation de la conception bas-niveau
    - Choix et cohérence du schéma
    - Clarté du discours
- Pertinence de l'analyse des choix de conc
- Réponses aux questions

Contenu

Vous présenterez vox choix de conception en quelques slides en incluant notamment :

- un schéma de l'architecture de votre application avec ses différentes entités et leurs relations. Il accompagnera vos explication sur vos choix de conception haut niveau : Pourquoi cette architecture logicielle plutôt qu'une autre ? Quel est le rôle des principales entités ? Quelles sont leurs responsabilités ? etc.

- un diagramme de classes UML de votre application, ou un équivalent si vous ne programmez pas selon le paradigme objet. Il accompagnera vos explications sur vos choix de conception bas niveau : Comment avez-vous choisi de gérer les données au sein des entité ? Comment sont créées les entités ? Avez-vous utilisé des Design Pattern ? Si oui, pourquoi ?

Ces points seront à critiquer au sens 1er du terme, c'est-à-dire en indiquant quels ont été selon vous vos bons et mauvais choix, ceux qu vous conserveriez et ceux que vous changeriez si c'était à refaire. N'essayez pas de cacher des erreurs, au contraire, montrez que vous les avez identifiées et que vous avez pris le recul nécessaire pour n plus les refaire à l'avenir.

(Notez que le mot "entité" utilisé ci-dessus est suffisamment vague pour être adapté à tous vos projets. Il peut représenter les éléments d'un client/serveur, d'un MVC, une classe, un objet, une fonction, etc.)

Projet T3

Fernandes Samuel, Le Roux Aymeric, Gillig Mattéo, Khenissi Tejeddine

Sujet - Objectif

Développer un jeu sérieux en fonction d’une problématique donnée afin d’apporter des connaissances aux joueurs.

Ce jeu doit être pensé pour être maintenable, debuggable, voire amélioré par la suite et par des équipes extérieures au projet.

Sujet - Problématique

Re-territorialiser par la matière. Approche du métabolisme urbain à l'échelle d'un quartier.

Koenigshoffen-Est à Strasbourg TW :

Géographie

Sujet - Explications

La gestion d’un territoire, une ville ou un village par exemple, passe par différents flux de matières.

Les flux économiques (financiers), écologiques et sociaux (démographie, ressources humaines) sont autant de facteurs garantissant ou non l’attractivité d’un territoire et son niveau de vie.

Pour fonctionner, ce territoire a également besoin de ressources et marchandises aussi diverses que variées (nourriture, eau, électricité, matières premières et biens de consommation de toute sorte !)

Notre sujet se concentre sur l’échelle d’un quartier pour mieux appréhender l’impact qu’à la matière sur ce dernier. Elle décompose cette tâche complexe qu’est la gestion d’un territoire.

Sujet - Connaissances à apporter

Aider le joueur à comprendre **l'impact des flux de marchandise sur l'occupation d'un territoire.**

Outils - Explications

Godot C# 4.0

- Moteur de jeu ⇒ simplification des créations graphiques
- Facilité de prise en main ⇒ temps de “formation” réduit
- C# - langage objet familier ⇒ temps de “formation” réduit, pas nécessaire d’apprendre complètement un nouveau langage
- Outil familier ⇒ certain des membres connaissaient déjà

Conception haut niveau - Choix

- Une fenêtre de jeu principale (pas de fioritures, d'accueil, config, etc)
- Une grille toujours visible
- Une liste de bâtiments toujours accessible
- Cliquer sur la grille puis un bâtiment de la liste (ou inversement) le place sur la grille
- Système d'inventaire, d'import et export via lui
- Jauges (économique, écologique, sociale)
- Système de tour
- Relation, échanges entre bâtiments
- Des notifications

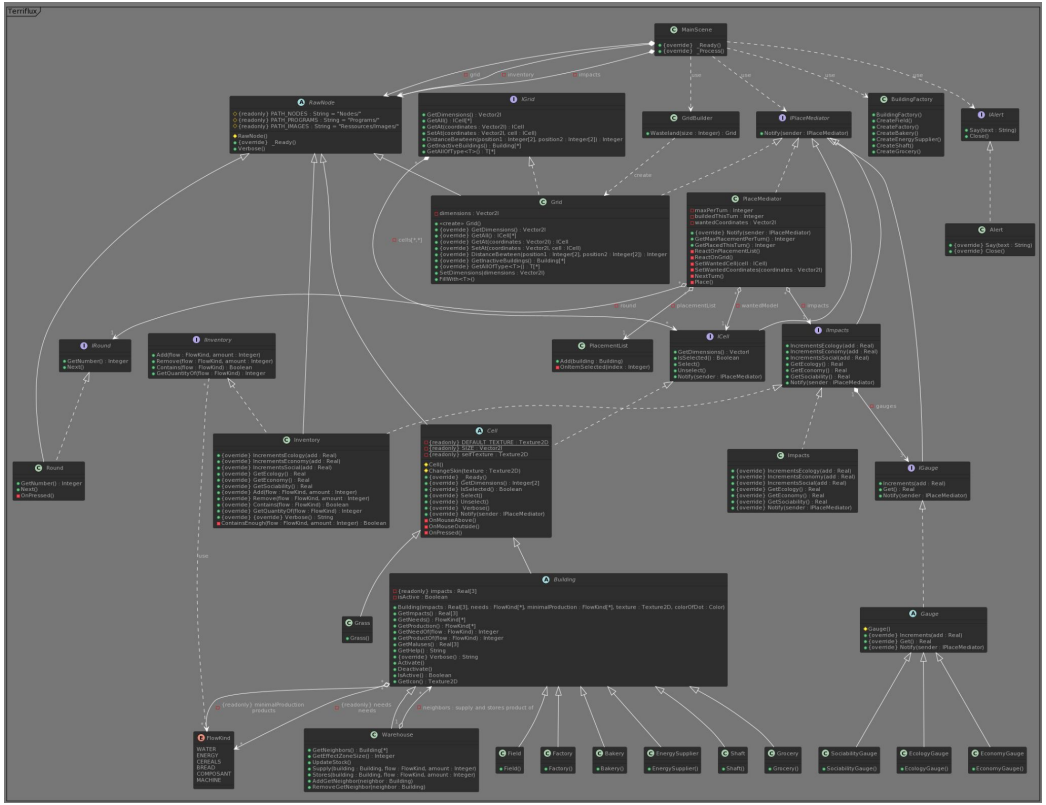
Conception haut niveau - Rôles

- La fenêtre de jeu principale créée et gère l'affichage des différentes entités principales (inventaire, notification, grille et liste de bâtiments)
- La grille gère les bâtiments, leur position et leur stockage
- La liste de bâtiments n'est qu'un outil pour placer rapidement un bâtiment sur la grille
- L'inventaire contrôle l'affichage des quantités des différents flux, leur import et export

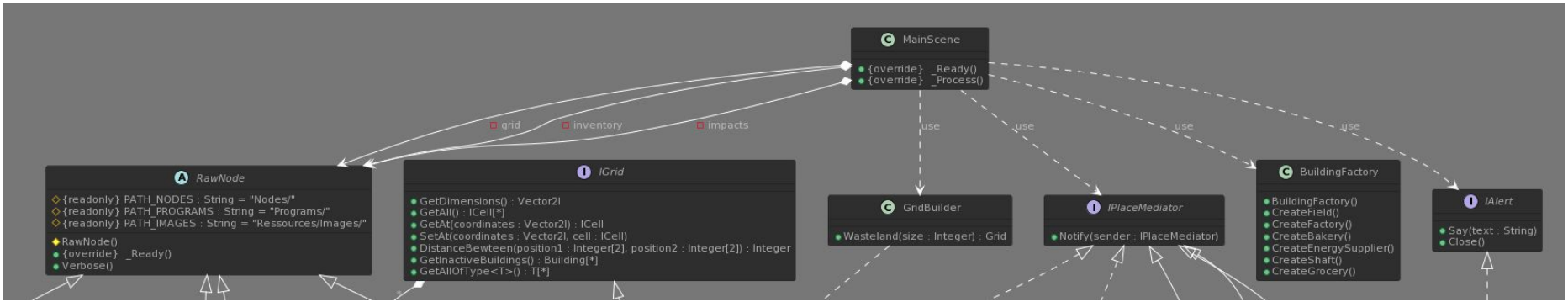
Conception haut niveau - Rôles

- Les jauges affichent au joueur l'état global de son territoire, leurs valeurs déterminent la victoire ou défaite du joueur en fin de partie
- Le système de tour déclenche toutes les mises à jour graphiques, les modifications sur les jauges et les calculs d'import-export dans l'inventaire (sans le faire lui-même !)
- L'entrepot (un type de bâtiment) gère les transferts de flux des bâtiments vers l'inventaire, et inversement
- Les notifications servent à communiquer avec le joueur

Conception bas niveau - Schéma

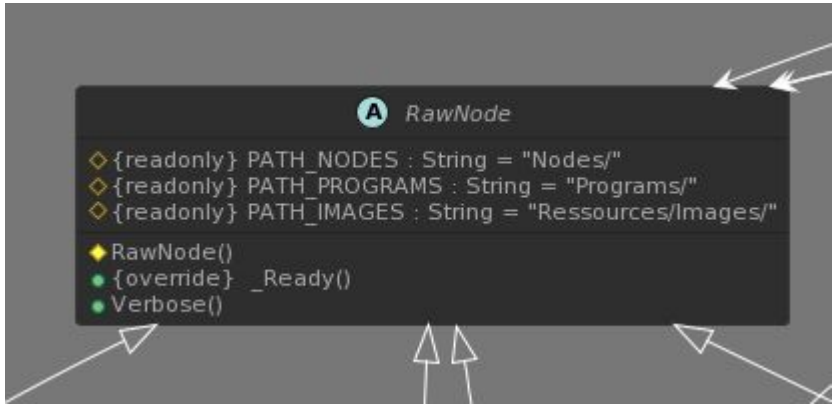


Conception bas niveau - Justifications



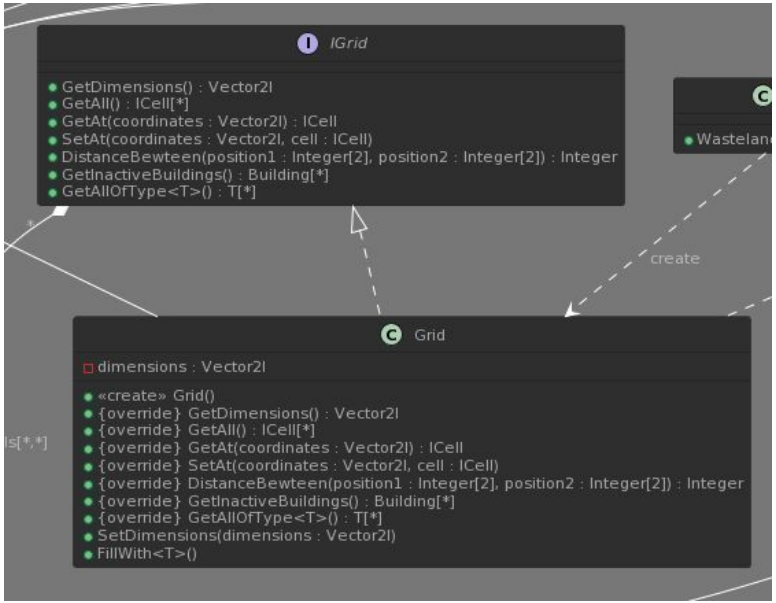
La fenêtre de jeu principale créée et stocke directement les différentes scènes “**RawNode**” (notamment la **grille**) et s’occupe d’afficher les notifications “**Alert**”

Conception bas niveau - Justifications



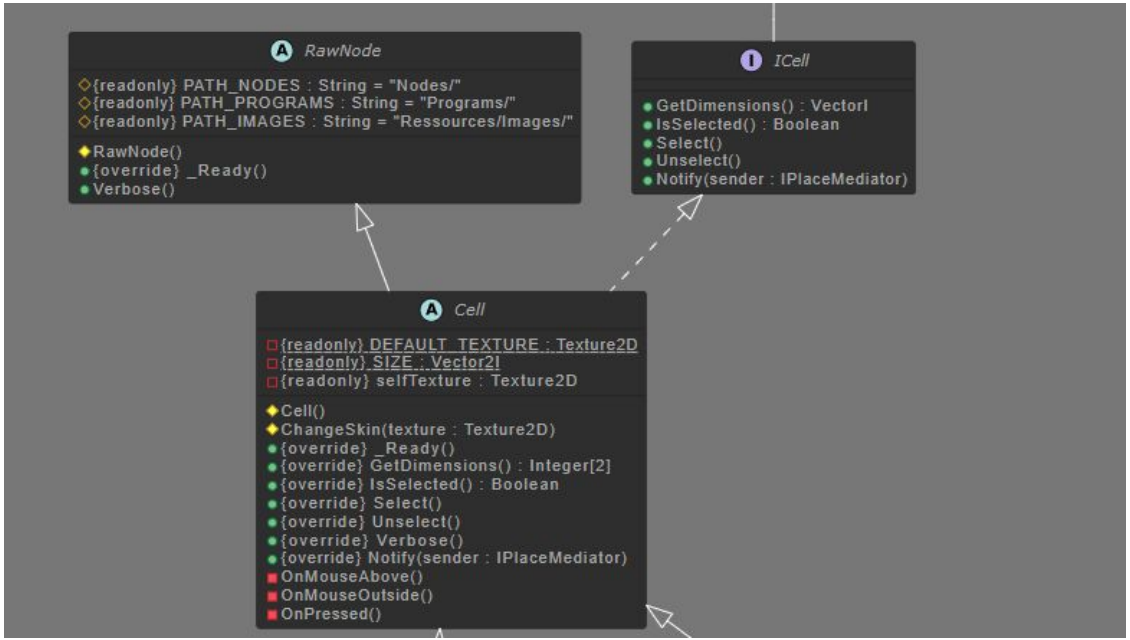
Godot permet d'instancier des scènes et des objets "graphiques" via une fonction assez lourde à écrire, en utilisant le chemin vers la scène ("classe graphique") dans les fichiers du projet. Nos scène héritent d'une même abstract "**RawNode**", offrant une façon simplifiée de les instancier.

Conception bas niveau - Justifications



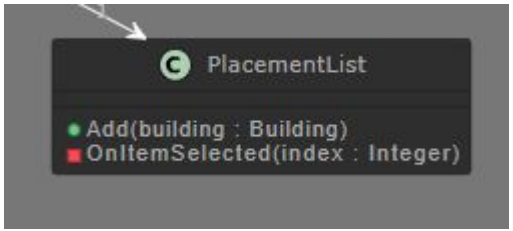
La grille est un tableau 2D contenant des objets ICell (bâtiments, terrains vagues, ...).

Conception bas niveau - Justifications



... voici les fameuses cellules “ICell” en question

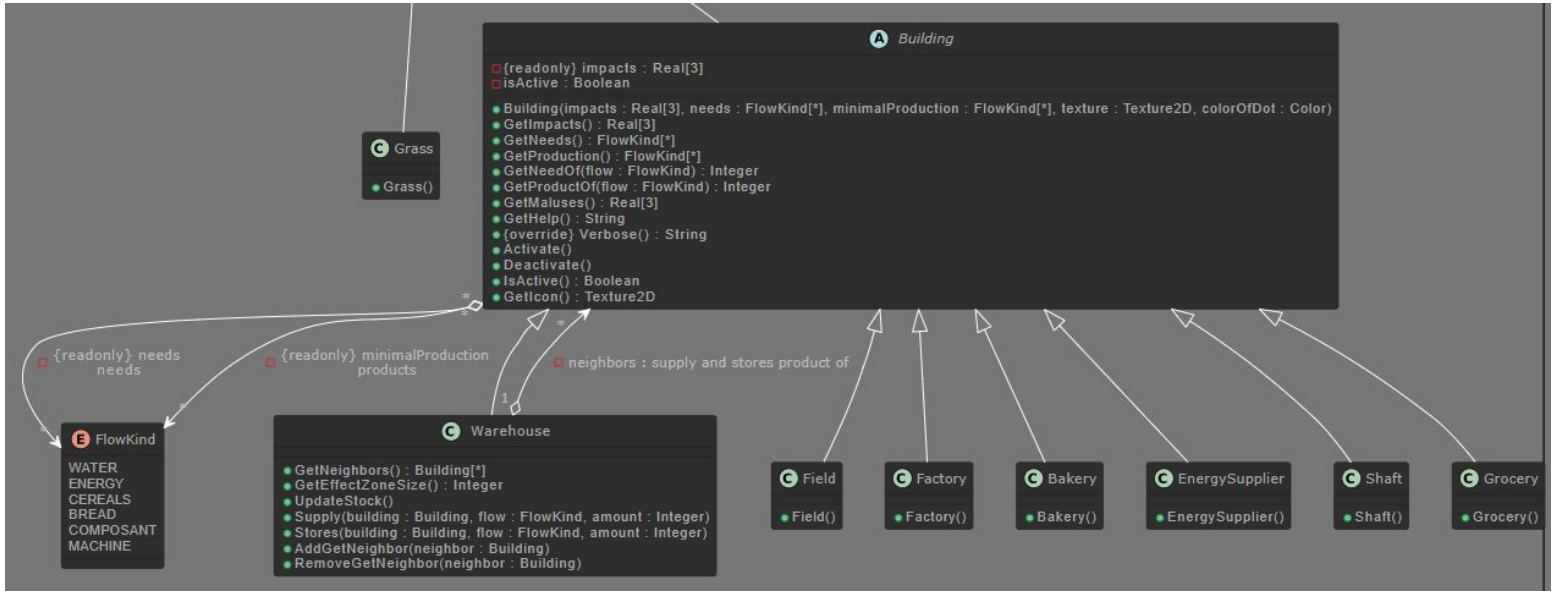
Conception bas niveau - Justifications



La liste des bâtiment est une ItemList fourni par Godot, agissant un peu comme une liste classique, affichant l'icône de chaque valeur au côté de son nom – pratique pour le joueur.

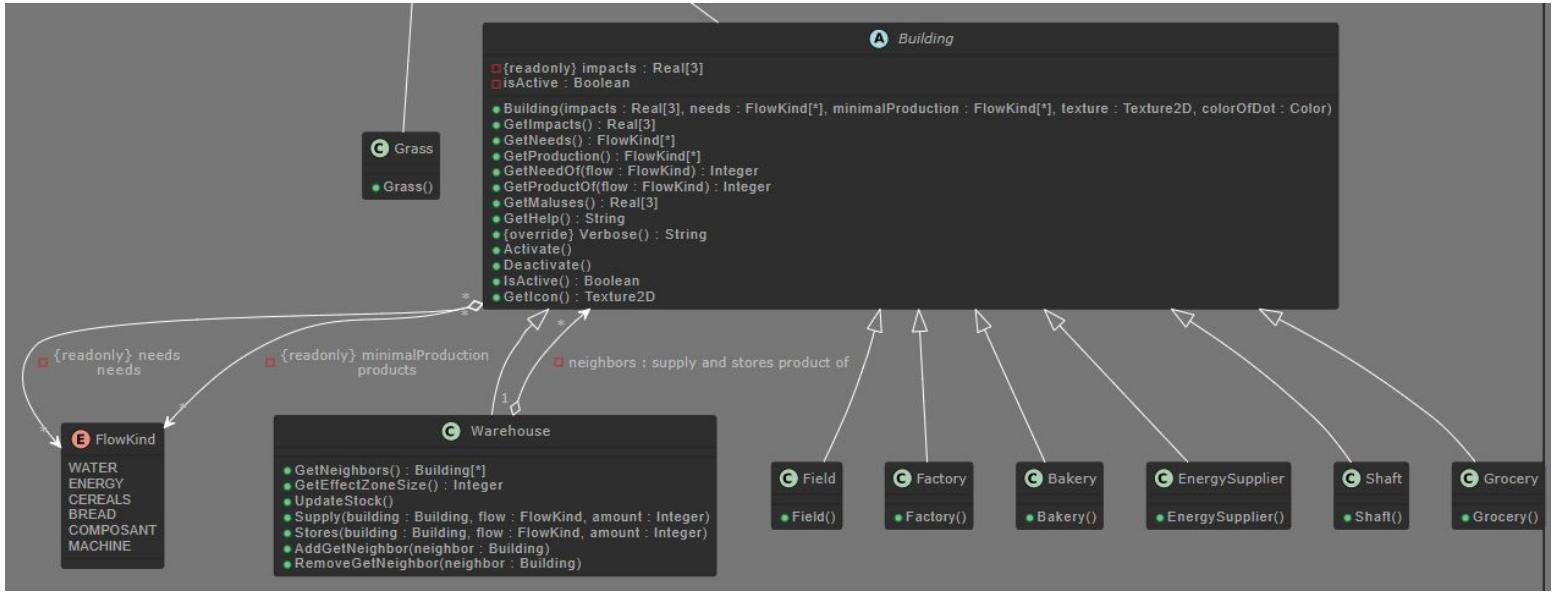
Inconvénient : impossible de récupérer un objet directement, seulement son nom. D'où l'utilité de RawNode permettant d'instancier via le nom d'une scène !

Conception bas niveau - Justifications



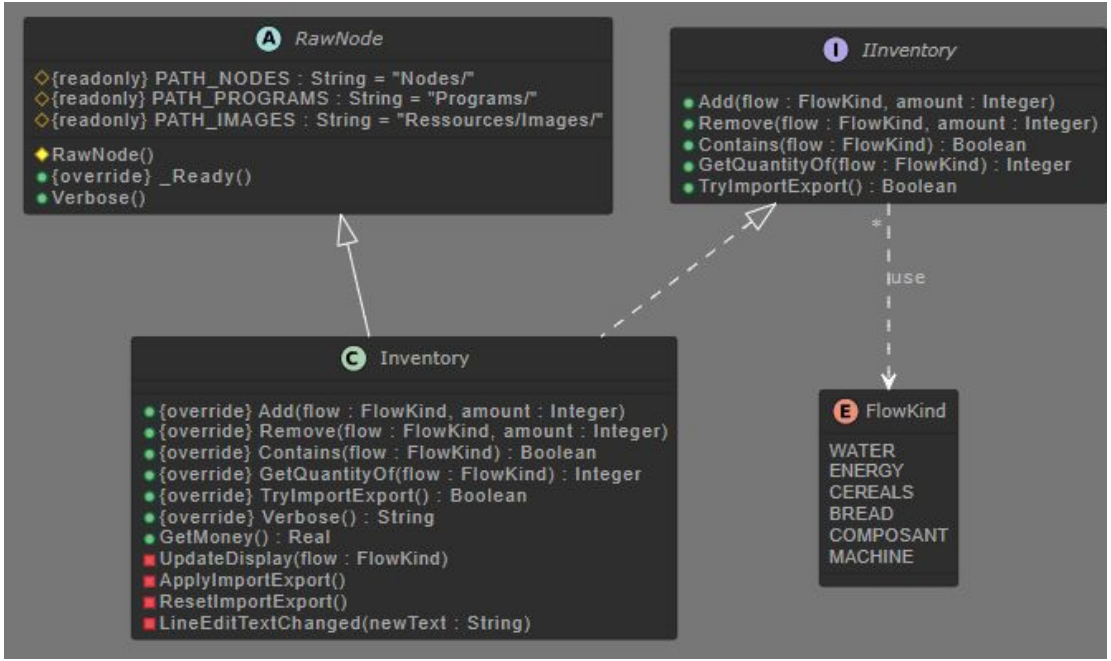
... les bâtiments en question, héritant de l'abstract **Cell**. Les bâtiments ont tous des besoins et des productions propre à chacun, exprimés en flux "**FlowKind**" ; l'association type de flux - quantité nécessaire / produite est stockée dans un **Dictionnaire**.

Conception bas niveau - Justifications



Les échanges et transfert des flux vers l'inventaire se font via le bâtiment spécifique Warehouse (entrepôt).

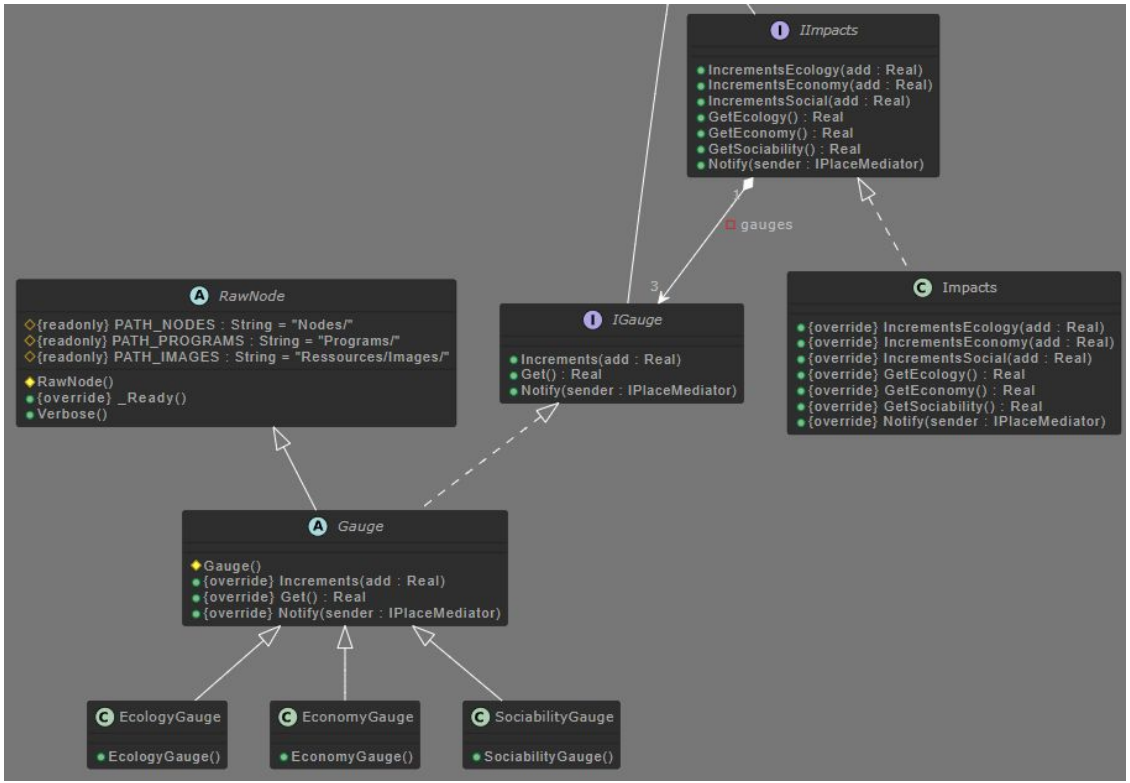
Conception bas niveau - Justifications



L'inventaire gère les transferts de flux via un **Dictionnaire** <type de flux, int>, hormis pour l'argent qui lui est sous forme de **double**.

Conception bas niveau - Justifications

Les jauges sont des TextureBar de Godot. En fonction de la valeur qu'elles stockent (**double**), elles se remplissent / vident de leur texture



Conception bas niveau - Design pattern

Nous utilisons dans ce projet 4 Design pattern différents :

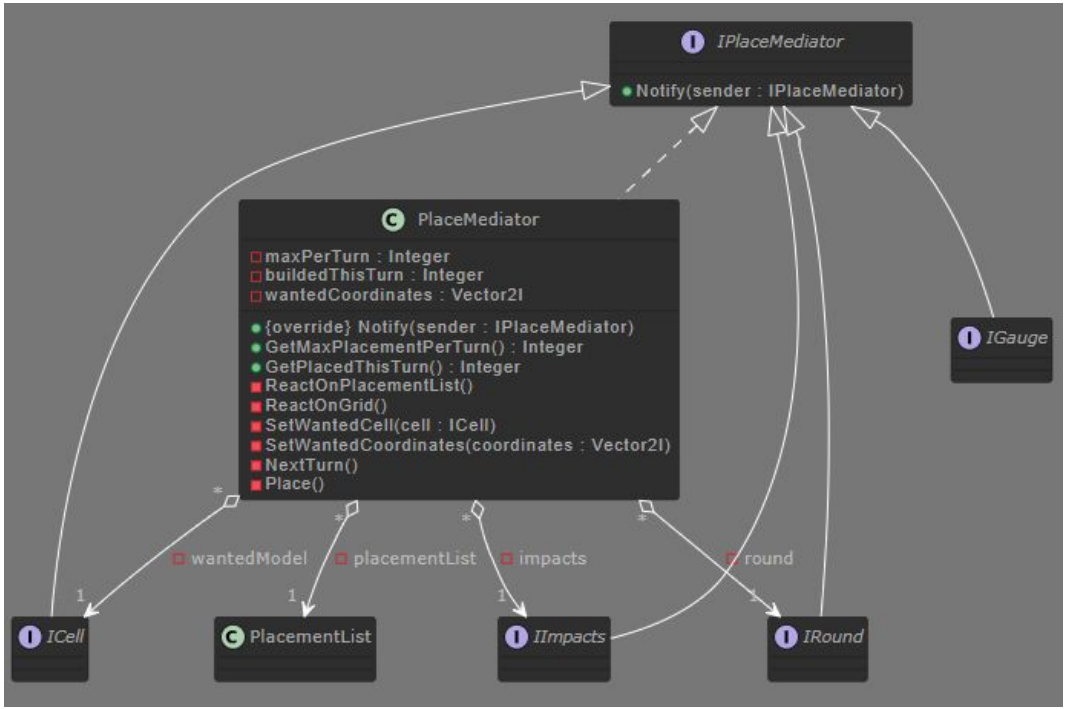
- mediator
- singleton
- factory
- builder

Au départ nous utilisions aussi un MVC, rendu inutile par Godot, et donc complexifiant l'architecture sans raison.

Conception bas niveau - Design pattern - Mediator

***Mediator** : oblige plusieurs classes à communiquer à travers lui plutôt que directement entre elles afin d'organiser au mieux les échanges*

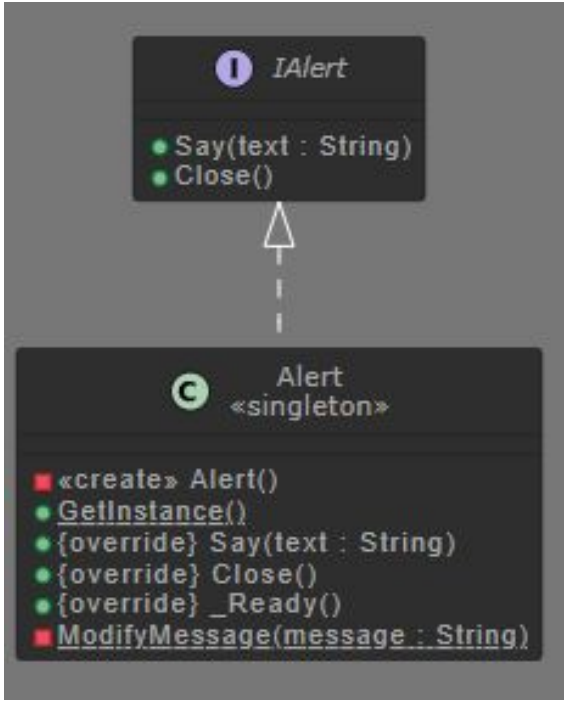
Ici, il sauvegarde le bâtiment à placer et les coordonnées où le placer, autorise ou non la construction, sauvegarde les valeurs à modifier au prochain tour et les transmet quand celui-ci est passé.



Conception bas niveau - Design pattern - Singleton

***Singleton** : il n'existe qu'une seule et unique instance de la classe, partagée*

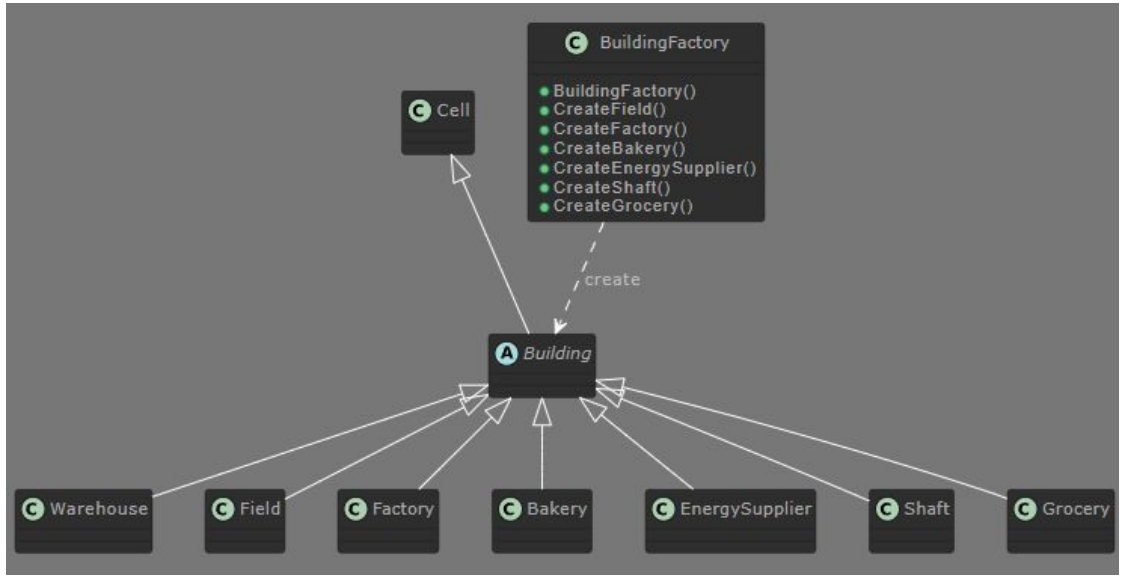
Ici, la notification est placée sur l' écran principale, au-dessus de toutes les scènes. Mais n'importe qui peut l'appeler !



Conception bas niveau - Design pattern - Factory

***Factory** : crée des instances de classes dérivées d'une autre, sans avoir à connaître spécifiquement ces classes dérivées*

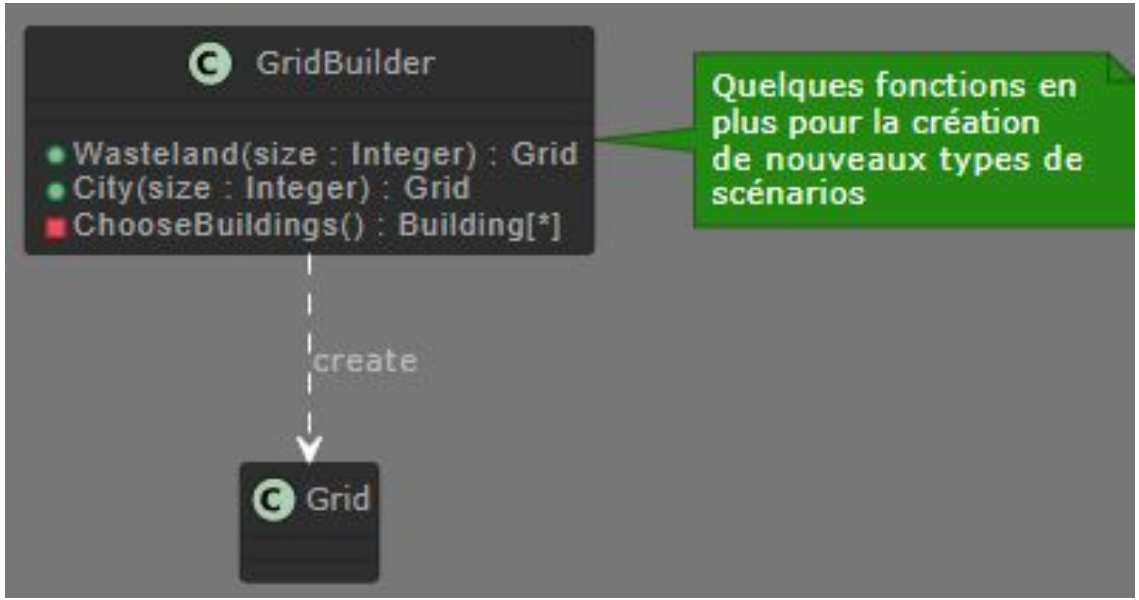
Ici, elle permet d'instancier tout type de bâtiment !



A ne pas confondre avec le bâtiment "Factory"

Conception bas niveau - Design pattern - Builder

Builder *: simplifie la création d'un objet complexe*



Ici, il permet d'instancier une grille déjà remplie avec de la terre.

Si un futur développeur souhaite ajouter un scénario (avec une ville existante dès le départ par exemple), ce builder lui permettra de créer cette grille préremplie.

Quelques retours des joueurs

Voici les points à améliorer si nous ou un autre développeur continue le projet, selon nos testeurs :

- Fournir plus d'informations au joueur sur les prix et le rôle important du Warehouse directement en jeu
- **Ajouter des feedbacks en masse (manque de transition entre les tours, de musiques, de bruitages, d'*animations*)**
- **Placer une sécurité empêchant le joueur de passer des tours sans construire**
- **Augmenter drastiquement le nombre de tours jouables et réduire la taille de la grille afin de pouvoir construire entièrement le quartier, le détruire, le modifier dans le temps * – phase de territorialisation puis déterritorialisation / reterritorialisation**
- Régler les bugs liés à l'écologie (qui ne peut que descendre, *étrangement*)

** le joueur devait ici perdre en ~2min et finir sa partie en environ 5min, il était difficile d'atteindre ces objectifs avec un grand nombre de tours et l'avons donc réduit à 10 tours maximum.*