

On Tue, Oct 3, 2023 at 4:13 PM Pierre Kraemer <kraemer@unistra.fr> wrote:

Bonjour,

Le projet T3 sera évalué sous plusieurs aspects. Avec M. Zimmermann, nous allons évaluer, pour le compte de l'UE2, la partie "structure de données et algorithmes" (dans le style de ce qui nous intéressera en P31).

Après les vacances de fin d'année, chaque groupe nous présentera :

- le thème du projet (très rapidement)
- les structures de données choisies pour représenter les données principales du jeu
- les algorithmes mis en œuvre pour la mécanique du jeu
- une justification de ces choix, pourquoi ils sont bons ou finalement mauvais et dans ce cas, ce que vous auriez pu faire de mieux.

Nous passerons ensuite à une série de questions, éventuellement individualisées, qui pourront notamment porter sur votre code. Vous pourrez donc préparer un éditeur de code ouvert avec votre projet.

Selon l'environnement technique choisi, il se peut que vous soyez très contraints sur la représentation des données et les algorithmes mis en œuvre. Dans ce cas, vous devez être capables de présenter / critiquer les structures principales qui sont au cœur des outils que vous utilisez.

Les détails d'organisation de ces présentations (date, durée, etc.) vous seront donnés plus tard. Le temps alloué à chaque groupe sera forcément relativement court. La pertinence de ce que vous aurez jugé important de mettre en avant dans cette présentation sera un critère d'évaluation.

Pierre Kraemer.

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.**



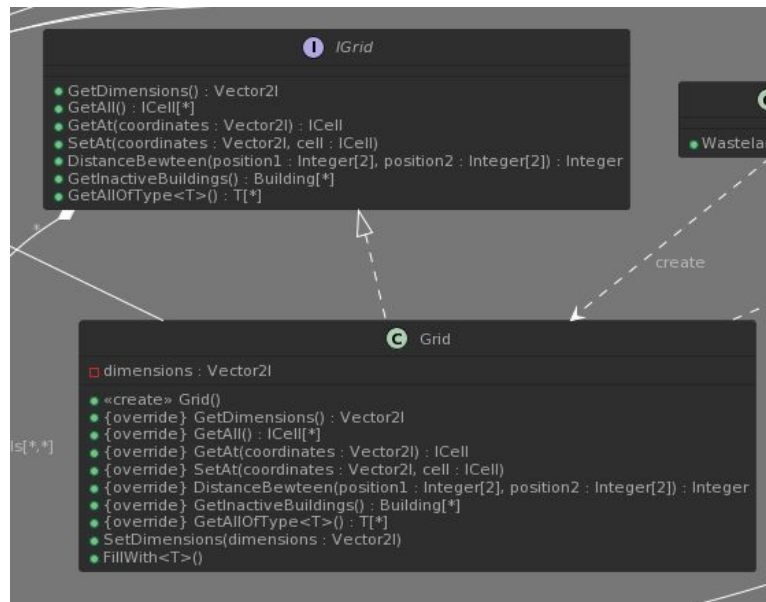
* pas possible d'agrandir le terrain courant de la partie, par exemple si on avait mis un système d'achat de nouvelles terres



Structures de données principales - Justifications

La grille de jeu

- ❖ tableau 2D
 - Avantages
 - coordonnées comme sur une carte (line, colonne), vecteur
 - Inconvénients
 - taille immuable *



















Structures de données principales - Justifications

La “liste” des bâtiments constructibles

- ❖ ItemList = dictionnaire <string, image>
 - Avantages
 - fourni par godot avec ses éléments graphiques
 - associe nom du bâtiment et la l'icône (texture) le représentant en jeu
 - Inconvénients
 - doit récupérer le nom (string) du bâtiment *puis* instancier un bâtiment du même nom au lieu d'obtenir une instance directement *

	Energysupplier	
	Shaft	
	Field	
	Factory	
	Bakery	
	Grocery	
	Warehouse	



Structures de données principales - Justifications

La “liste” des bâtiments constructibles

Sur la grille de jeu :

- récupération du nom du bâtiment et instantiation d'un nouveau via une fonction communes à nos scènes

Instantiate(name : String) : RawNode

```
private readonly List<Building> drafts;  
  
// to see needs and production
```

Pour les classes utilisant des données propre aux **classes** de bâtiments :

- sauvegarde d'une instance “brouillon” de chaque bâtiment avec les mêmes index que dans l'ItemList à son initialisation
- récupération de l'**index** du bâtiment sélectionné (fourni par l'ItemList)
- passe la référence du brouillon au même index dans la liste “draft” (> script de notre l'itemlist)



Structures de données principales - Justifications

Un bâtiment

```
A Building

{readonly} impacts : Real[3]
isActive : Boolean

«create» Building(impacts : Real[3], needs : FlowKind[*], minimalProduction : FlowKind[*], texture : Texture2D, colorOfDot : Color)
GetImpacts() : Real[3]
GetNeeds() : FlowKind[*]
GetProduction() : FlowKind[*]
GetNeedOf(flow : FlowKind) : Integer
GetProductOf(flow : FlowKind) : Integer
GetMaluses() : Real[3]
GetHelp() : String
{override} Verbose() : String
Activate()
Deactivate()
IsActive() : Boolean
GetIcon() : Texture2D
```



Structures de données principales - Justifications

Un bâtiment (2)

- ❖ Impacts = **double[3]**
 - Avantages
 - peu d'utilisation mémoire
 - accès rapide aux données

- ❖ Besoins ("needs") & Productions ("minimalProduction") = **dictionnaire<type de flux, int>**
 - Inconvénients
 - plus d'encapsulation (une fonction pour les clés, une pour les valeurs)
 - + lourd qu'une liste en mémoire
 - Avantages
 - meilleures organisation des données



Algorithme - Fonctionnement

- Le joueur choisit une case de la grille (click sur la case)
- Le joueur choisit un bâtiment à construire (click dans la liste)
- Le bâtiment est placé sur la grille
- Les jauges d'état du territoire (économie, écologie, social) sont mises à jour
- Le joueur choisit (via l'inventaire) les ressources à importer et exporter
- Le joueur décide de passer au tour suivant
- L'inventaire est mis à jour en fonction des besoins et productions des bâtiments ainsi que de l'import-export
- Le numéro de tour est changer.



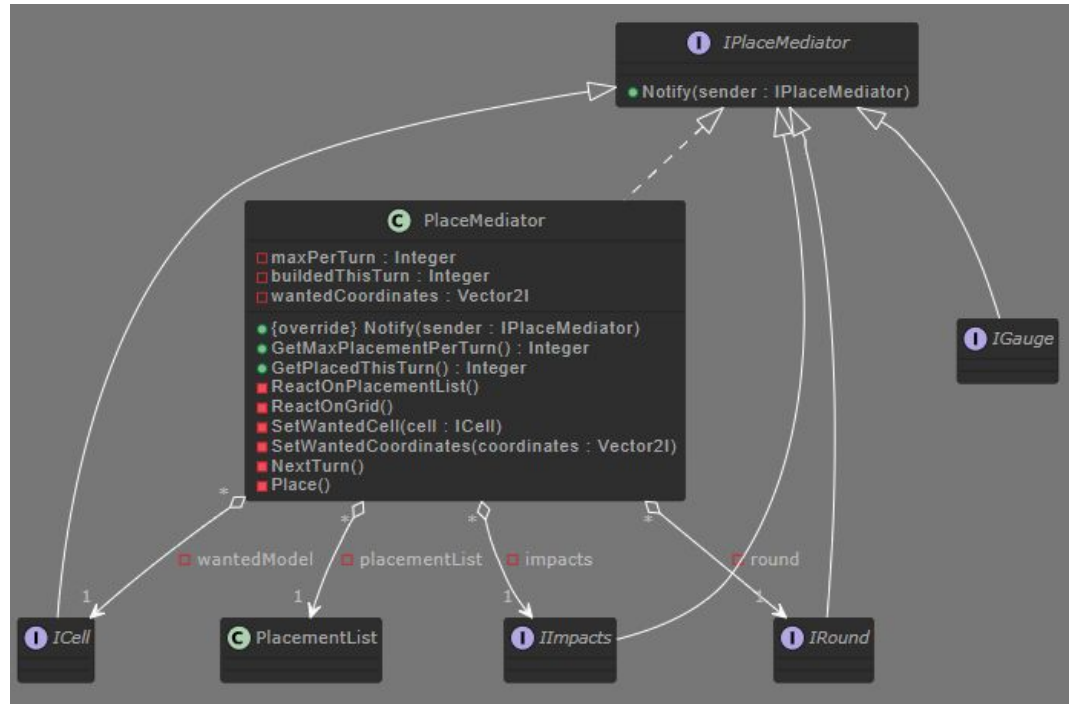
*Possibilité d'intervvertir
et répéter l'opération*

Fin, on boucle

Algorithme - Implémentation

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

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.

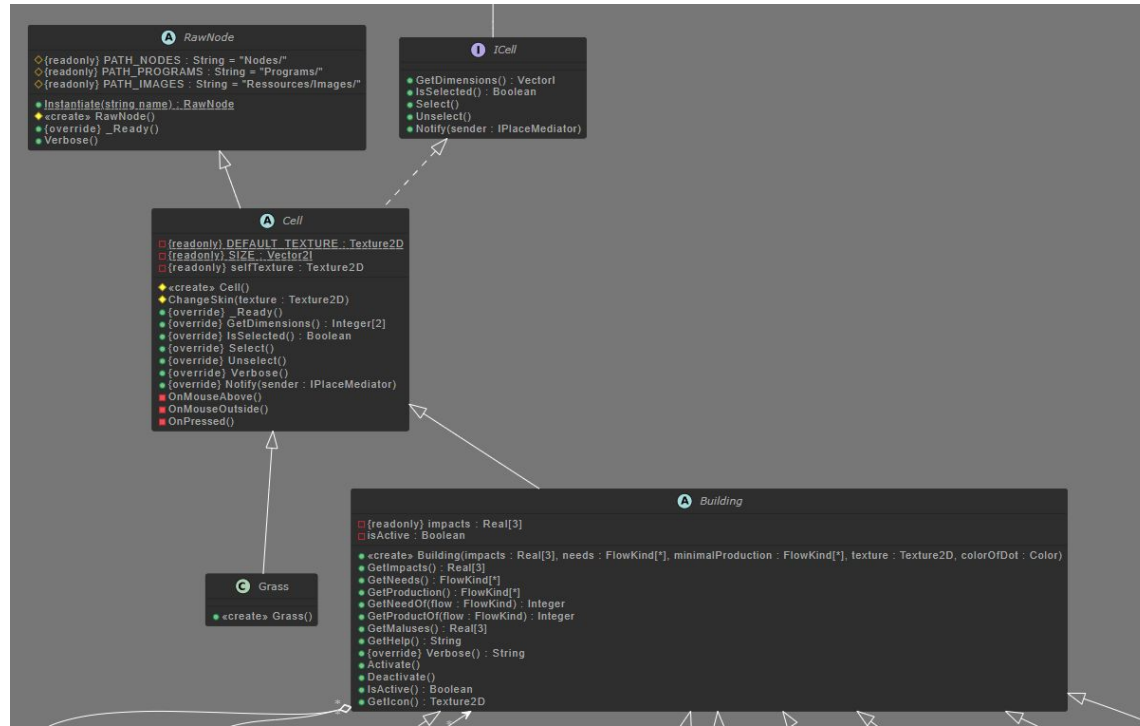


Annexe - RawNode

A *RawNode*

- ◆{readonly} PATH_NODES : String = "Nodes/"
- ◆{readonly} PATH_PROGRAMS : String = "Programs/"
- ◆{readonly} PATH_IMAGES : String = "Ressources/Images/"
- ◆«create» RawNode()
- ◆RawNode()
- {override} _Ready()
- Verbose()

Annexe - Building



Annexe - Building (2)

