

# Architecture d'un programme interactif graphique

## Partie 1: MVC

### Objectifs:

- Organiser un projet selon l'architecture Model-View-Controller

### Plan:

- Les 3 types de dialogue utilisateur
- Architecture Model-View-Controller

# Les 3 styles de dialogues humain-machine

- **ligne de commande** (commandes UNIX-LINUX)
  - passage de paramètres au programme avec **argc** et **argv**
- **Conversational** (programmes semestre d'automne)
  - **cout**, **cin**, etc disponible avec la bibliothèque standard C++
- **Interface Graphique Utilisateur** (*Graphic User Interface* / GUI)
  - widgets (boutons, etc...) disponible dans **bibliothèques**

**Bibliothèque** (*Library*) : regroupe le code objet de plusieurs modules réalisant une tâche bien définie (exemple: interface graphique, dessin, etc...)

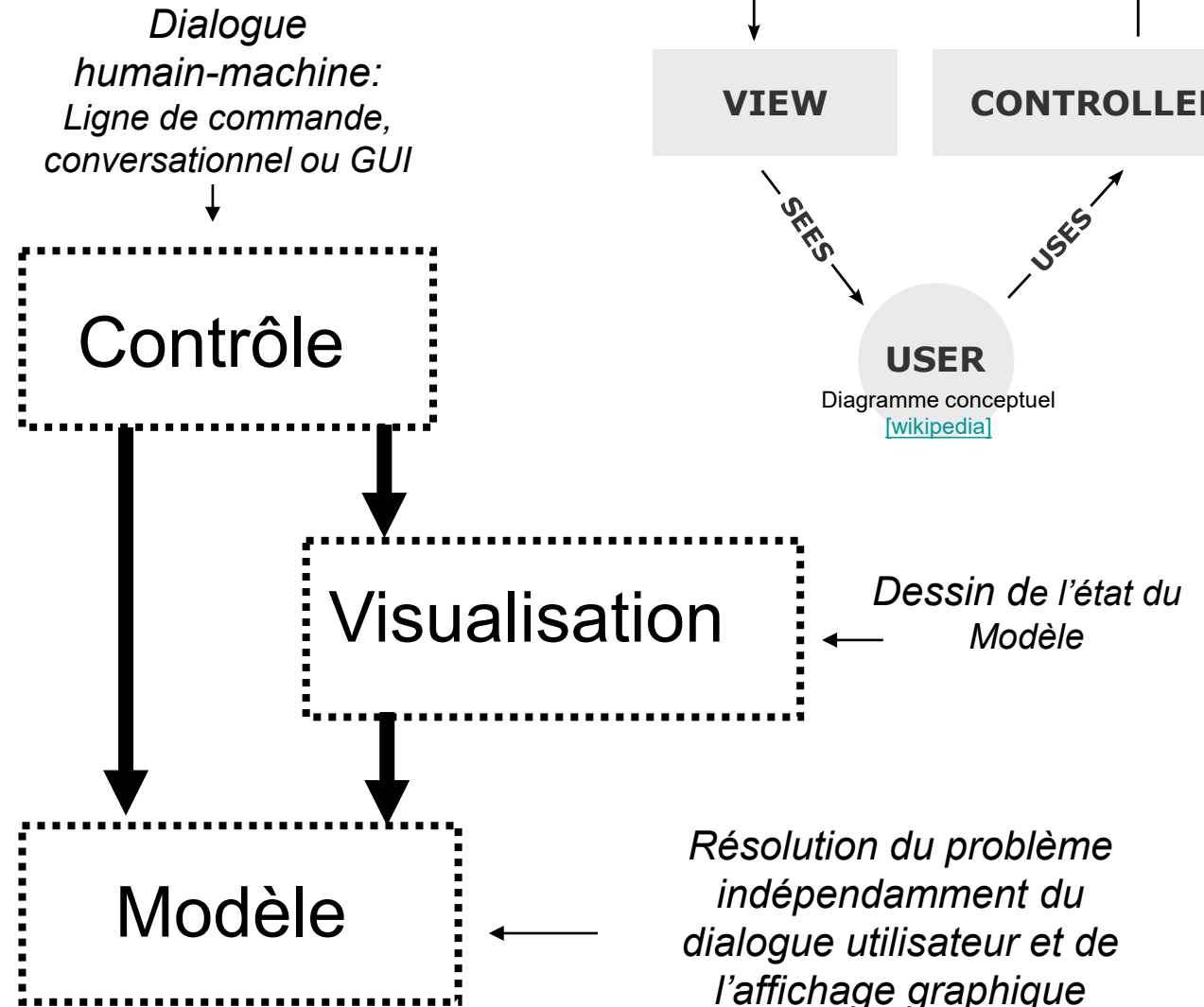
**API** (*Application Programming Interface*) : prototypes des fonctions exportées par une bibliothèque dans un fichier en-tête (exemple: **gtkmm.h**) = fichier d'interface de la bibliothèque

# L'architecture MVC: Model-View-Controller

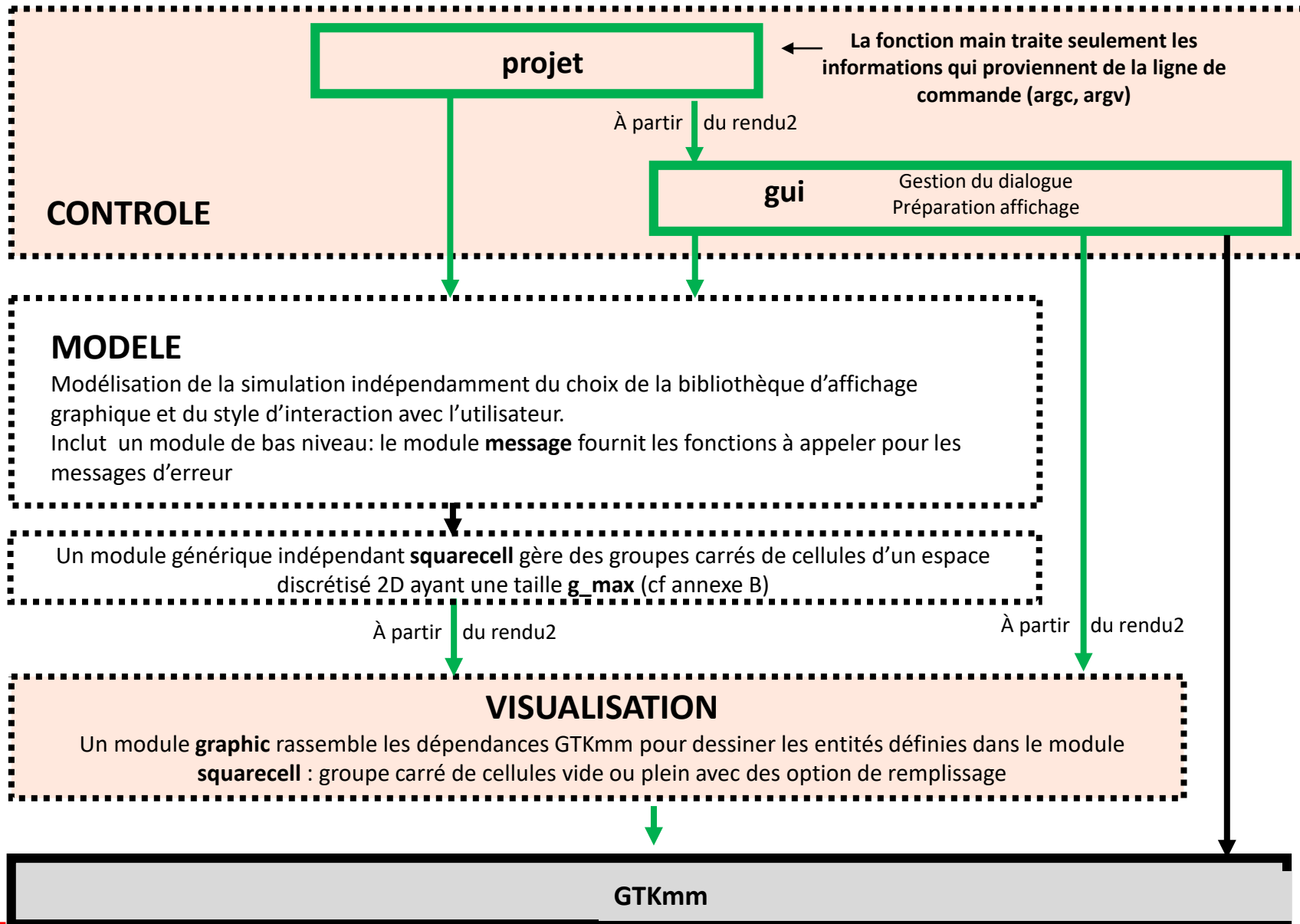
Principe : *Separation of concerns*  
séparation des responsabilités

## Objectifs:

- Permettre **plusieurs styles de Visualisations** et de **Contrôle/Dialogue** sur un Modèle.
- S'adapter facilement à l'évolution rapide de la technologie des parties "Visualisation" et "Contrôle utilisateur" en **gardant stable la partie "Modèle métier"**
- Dans l'industrie, **les compétences sont souvent focalisées sur un sous-système**, rarement sur l'ensemble des trois.



# L'architecture MVC: Model-View-Controller(2)



## Simplifications dans le cadre du cours/projet :

Le **Modèle** n'a aucune dépendance graphique ; seul **squarecell** exploite les fonctions mises à disposition dans l'interface **graphic.h**

Le module **graphic** regroupe les dépendances de dessin vis-à-vis d'une bibliothèque externe (GTKmm).

Il offre le moyen de dessiner des éléments simples en 2D.

# Résumé

- En vertu du principe de **séparation des responsabilités/fonctionnalités** nous adoptons l'approche **Model-View-Controller** pour l'architecture d'une application interactive:
  - Le sous-système de **Contrôle** pilote celui de **Visualisation** et celui du **Modèle** du problème traité.
  - Le sous-système du **Modèle** n'a aucune dépendance vis à vis du dialogue humain-machine ni des fonctions de dessins
- GTKmm offre une hiérarchie de classes C++ pour construire une interface utilisateur