# Kronologic

Dietrich Corentin,
Dziezuk Mathieu,
Mougin Enzo





# Sommaire

- Introduction
- La Programmation par Contraintes
- Liste des fonctionnalités
- Maquette de l'application
- Diagrammes
- Planning des itérations et rôle de chacun
- Bilan itération 1
- Démonstration Choco-Solver



# Introduction

### Objectifs du projet:

- Assimilation et intégration des règles
- IA déduction dont une IA de déduction heuristique et une IA de déduction Choco-Solver
- IA assistante dont une IA assistante heuristique et une IA assistante Choco-Solver
  - IA joueuse
  - Générateur de scénarios



### La Programmation par Contraintes

• Les variables

Pi,t : Position du personnage i au temps t (1≤ t ≤ 6).

Les domaines

Les salles sont numérotées de 1 à 6, donc la variable Pi,t a un domaine : {1, 2, 3, 4, 5, 6}.

Les contraintes

### Contraintes de règles

- 1. Déplacement obligatoire
- Déplacement dans des salles adjacentes

### Contraintes de début de partie

1. Au début de la partie (**t** = **1**), la position initiale de chaque personnage est annoncée

### Contraintes au cours de la partie

- 1. Public (Temps choisi)
- 2. Privé (Temps choisi)
- 3. Public (Personnage choisi)
- 4. Privé (Personnage choisi)

### Fonctionnalités liées à :

### l'interface utilisateur

- Créer une page d'accueil
- Créer l'interface de jeu
- Permettre au joueur de poser une question
- Permettre au joueur de formuler une hypothèse
- Permettre au joueur de formuler des déductions

- Permettre au joueur d'accéder aux déductions de l'IA
- Permettre au joueur de demander de l'aide à l'IA assistante
- Permettre au joueur de changer d'affichage
- Permettre au joueur de visualiser les règles
- Permettre au joueur de visualiser le film des déplacements des personnages

### Fonctionnalités liées à :

### l'IA de déduction

On disposera de deux IA de déduction : une heuristique et une utilisant Choco-Solver. Bien qu'ayant des fonctionnements différents, elles devront posséder les mêmes fonctionnalités.

Analyser les indices collectés et en tirer des conclusions logiques

Proposer une hypothèse logique basée sur les indices

Composer une hypothèse logique basée sur les indices

### Fonctionnalités liées à :

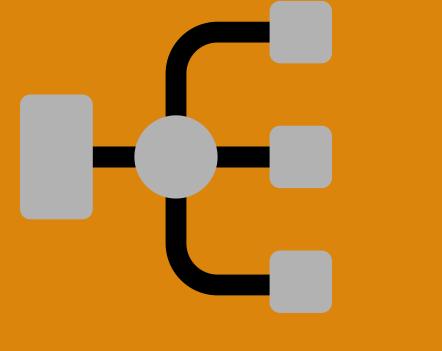
### l'IA d'assistance

On disposera également de deux IA d'assistance, une heuristique et une utilisant Choco-Solver. Ayant le même objectif, elles possèderont les mêmes fonctionnalités.

Recommander une question optimale à poser



Simuler les conséquences de différentes questions



Corriger les déductions du joueur



### Partie optionnelle

## IA joueuse

Raisonner sur les indices collectés et en tirer des déductions logiques

Poser les questions permettant d'obtenir les informations les plus pertinentes pour la complétion de l'enquête

Générer des hypothèses intermédiaires qui seront ajustées en fonction des nouveaux indices

Éliminer les options impossibles au fur et à mesure afin de pouvoir raisonner sur un ensemble d'informations plus petit

Formuler une déduction une fois qu'elle a trouvé le coupable, le temps et le lieu du crime

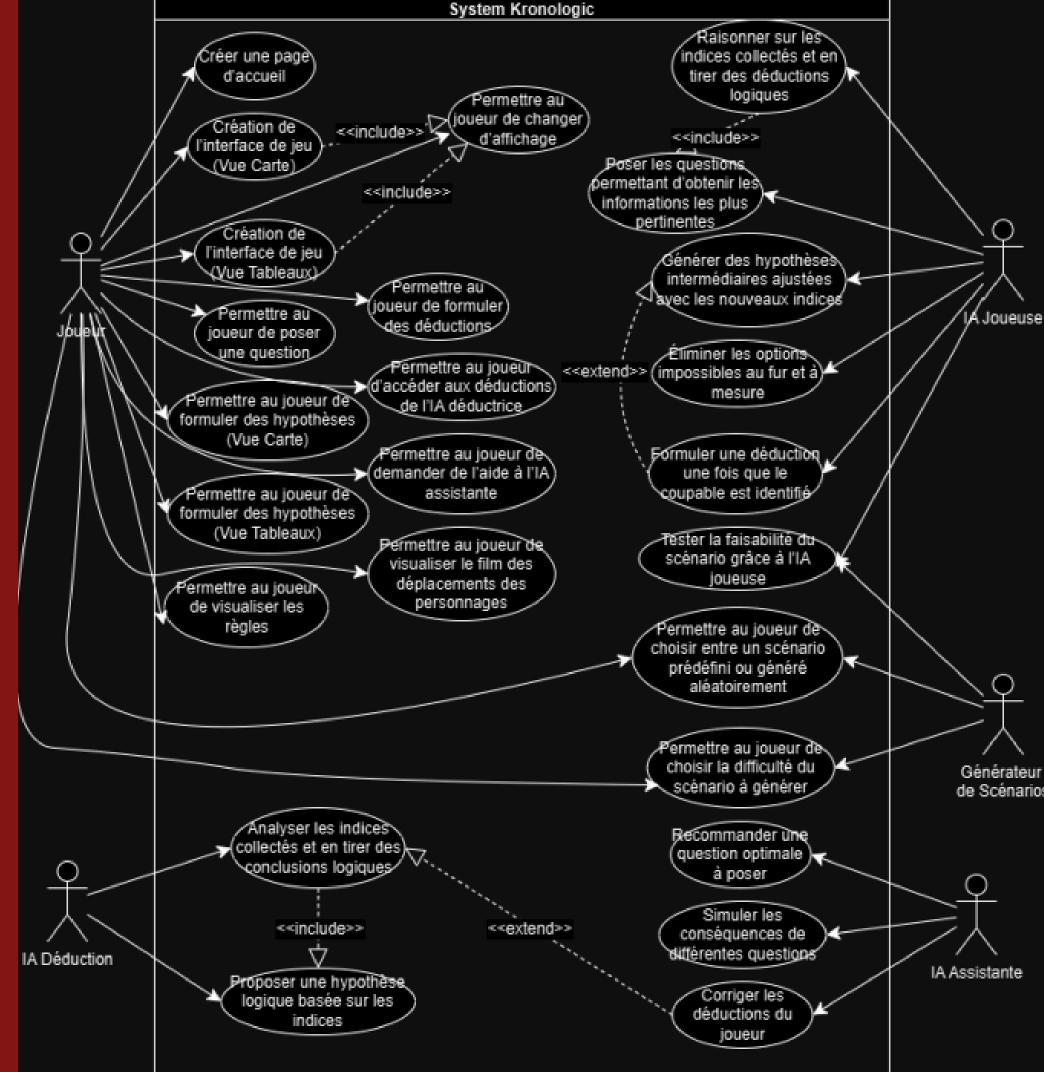
# Générateur de scénarios

Permettre au joueur de choisir entre un scénario prédéfini ou un nouveau scénario généré aléatoirement

Permettre au joueur de choisir la difficulté du scénario à générer

Tester la faisabilité du scénario grâce à l'IA joueuse avant de le proposer au joueur (une seule solution possible et trouvable)

# Diagramme de Cas d'Utilisations



# Kronologic

Jouer

IA Joueuse

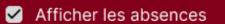
Quitter





Joueur





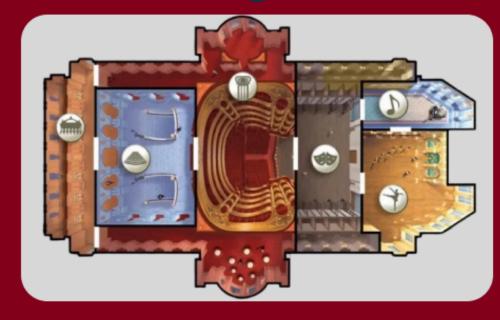














Historique des indices :

Tour 2: Il y a O personne La Salle au Temps 1.

Tour 1: L'Aventurière est passée 0 fois dans La Salle.





Hypothèse









Absence

✓ Masquer les hypothèses

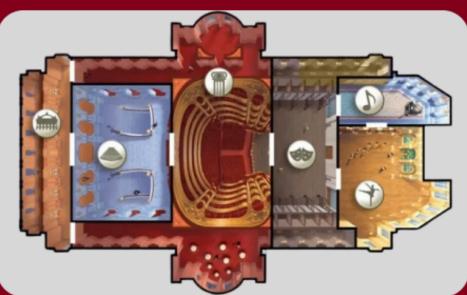


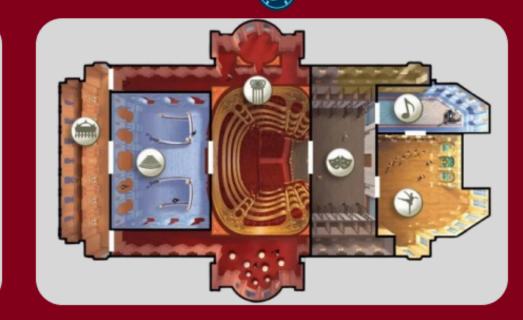
Déduction de l'IA











Faire une déduction

Poser une question

Demander un indice

Pour augmenter le nombre de personnage il faut double cliquer sur le pion. Cela vous permet de choisir un nombre entre 0 et 6.

Pions d'hypothèse de présence



Pions de présence





Pions d'hypothèse de nombres



Pions de nombres



Pions d'hypothèse d'absence



Pions d'absence













#### Changer affichage

Déduction de l'IA

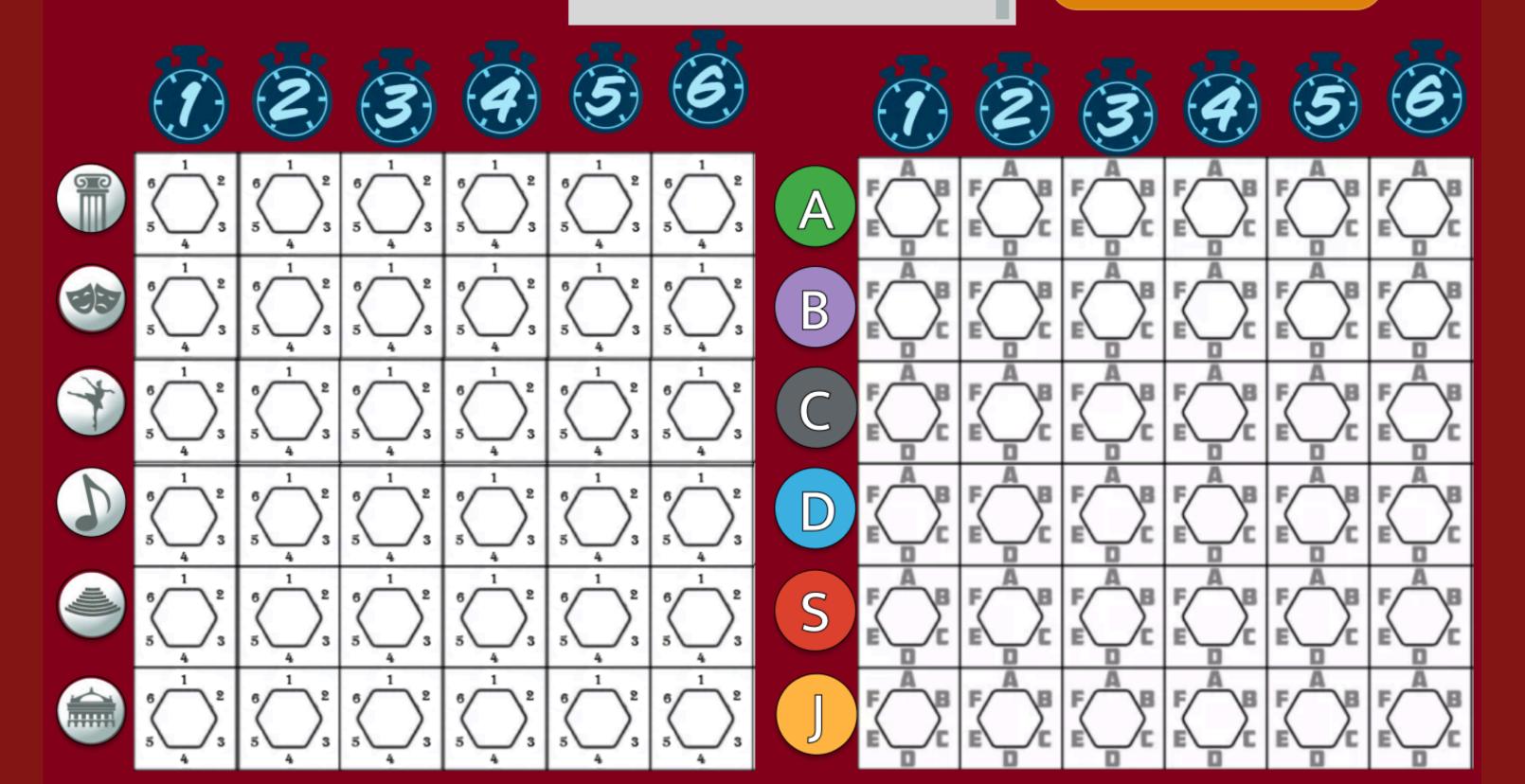
Historique des indices :

Tour 2: Il y a O personne La Salle au Temps 1.

Tour 1 : L'Aventurière est passée O fois dans La Salle.

Demander un indice

Faire une déduction



### Posez une question:

Choisissez un Lieu:













Choisissez un Temps ou un Personnage:





















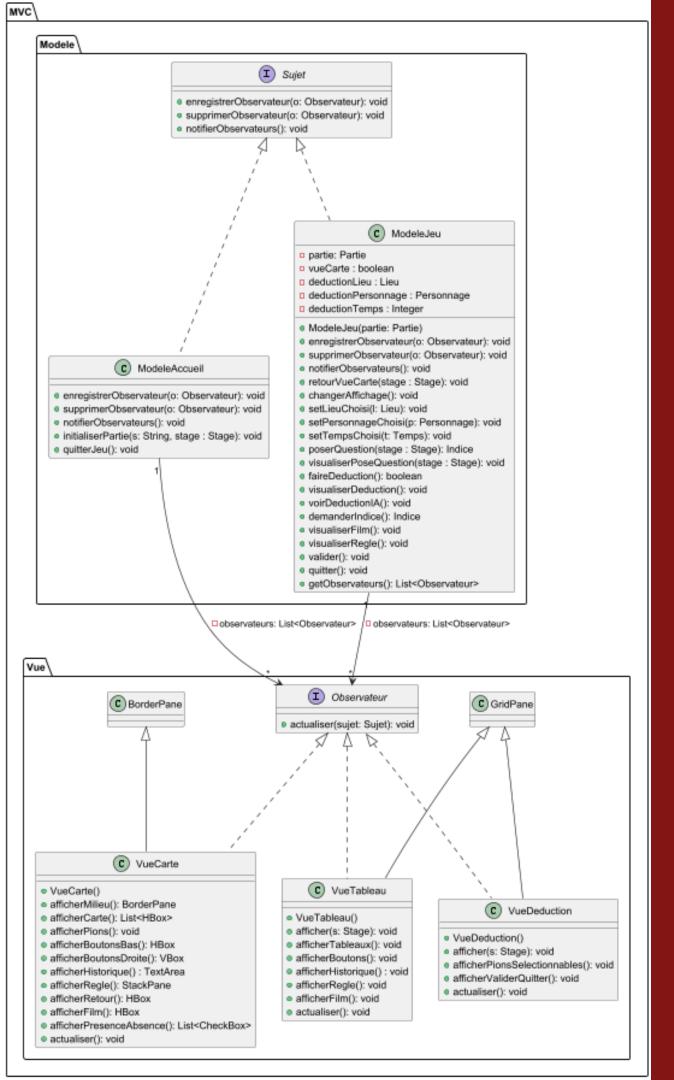






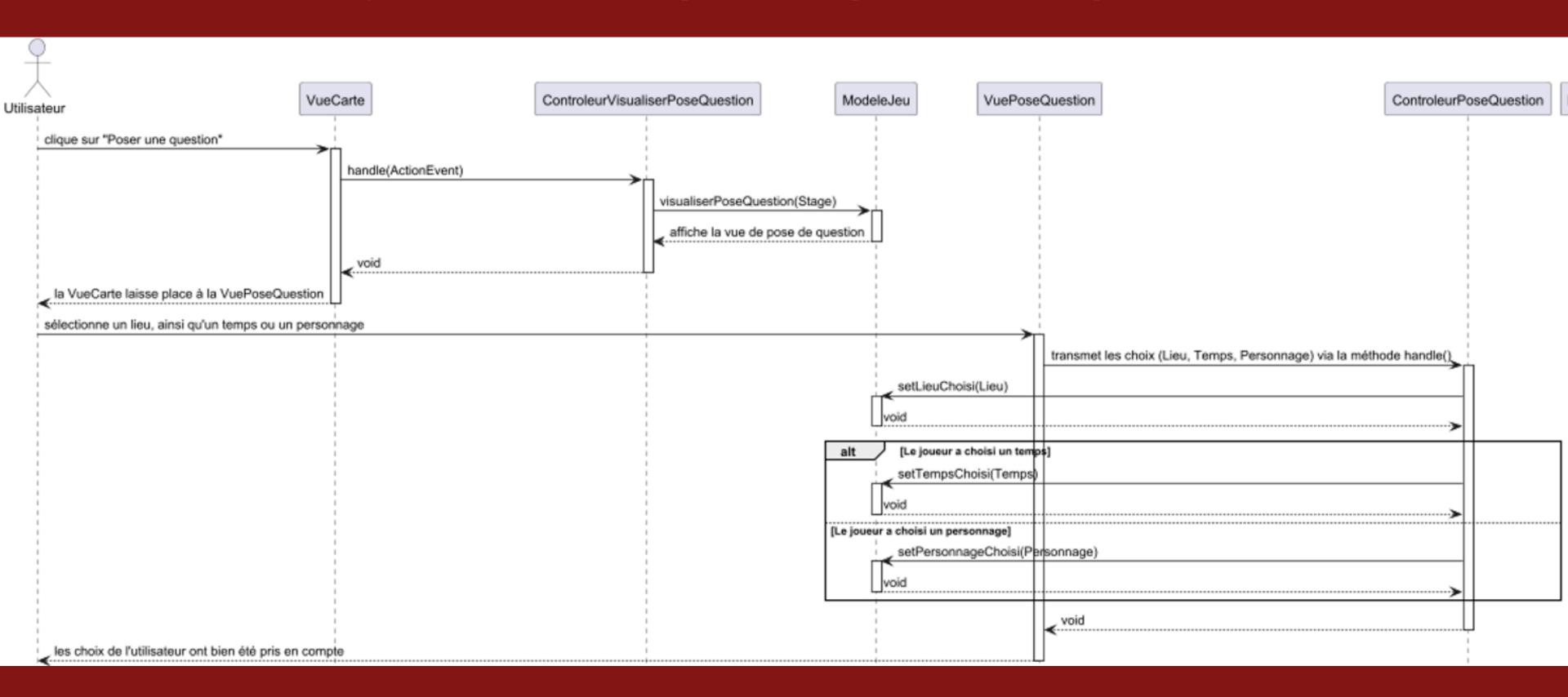
#### C Partie nbQuestion: Integer · Partie(e: Enquete, d: Deroulement, gn: GestionnaireNotes, gi: GestionnaireIndices) poserQuestionPersonnage(l: Lieu, p: Personnage): Indice poserQuestionTemps(I: Lieu, t: Temps): Indice · ajouterIndice(i: Indice): void · faireDeduction(I: Lieu, p: Personnage, t: Temps): boolean · demanderIndice(): Indice Indice C GestionnaireIndices · GestionnaireIndices(I : List<Indice>) indicesDecouverts poserQuestionPersonnage(l: Lieu, p: Personnage): Indice o poserQuestionTemps(l: Lieu, t: Temps): Indice enquete (A) Indice □ lieu: Lieu public: Integer o Indice(I: Lieu, p: Integer) o toString(): String · getLieu(): Lieu getInfoPublic(): Integer C Enquete idEnquete : int nomEnquete : String synopsis : String enigme : String loupeOr:int loupeBronze : int C Deroulement A Position · Enquete(id: int, n: String, s: String, e: String, lo: int, lb: int) C IndiceTemps C IndicePersonnage Deroulement(lp: List<Position>) · Position(I: Lieu, t: Temps, p: Personnage) · verifierLoupe(int nbQuestion): String prive: String prive: String positionsAuTemps(t: Temps): List<Position> getLieu(): Lieu faireDeduction(l: Lieu, p: Personnage, t: Temps): boolean positionsDansLieu(l: Lieu): List<Position> getTemps(): Temps getIdEnquete(): int IndiceTemps(I: Lieu, p: Integer, t: Temps, p: String) IndicePersonnage(I: Lieu, p: Integer, perso: Personnage, p: String) positionsDuPersonnage(p: Personnage): List<Position</li> · getNomEnquete(): String · getSynopsis(): String · getEnigme(): String getMeurtrier(): Personnage getLieuMeurtre(): Lieu getTempsMeurtre(): Temps getLoupeOr(): int getLoupeBronze(): int listePositions temps C Lieu C Realite C Personnage nom: String C Temps C Hypothese Id : Integer Realite(I: Lieu, t: Temps, p: Personnage) nom: String □ listeLieuxAdiacents temps : Integer و Lieu(n: String, i: Integer, I: List<Lieu>) getLieu(): Lieu Servira à la gestion des notes du joueur (qu'on développera dans l'itération2) Personnage(n: String, I: List<Lieu>) getTemps(): Temps Temps(t: Integer) agetNom(): String e getNom(): String · getId(): Integer getPersonnage(): Personnage getListeLieuxAdjacents(): List<Lieu>

# Diagramme de classe partie Kronologic

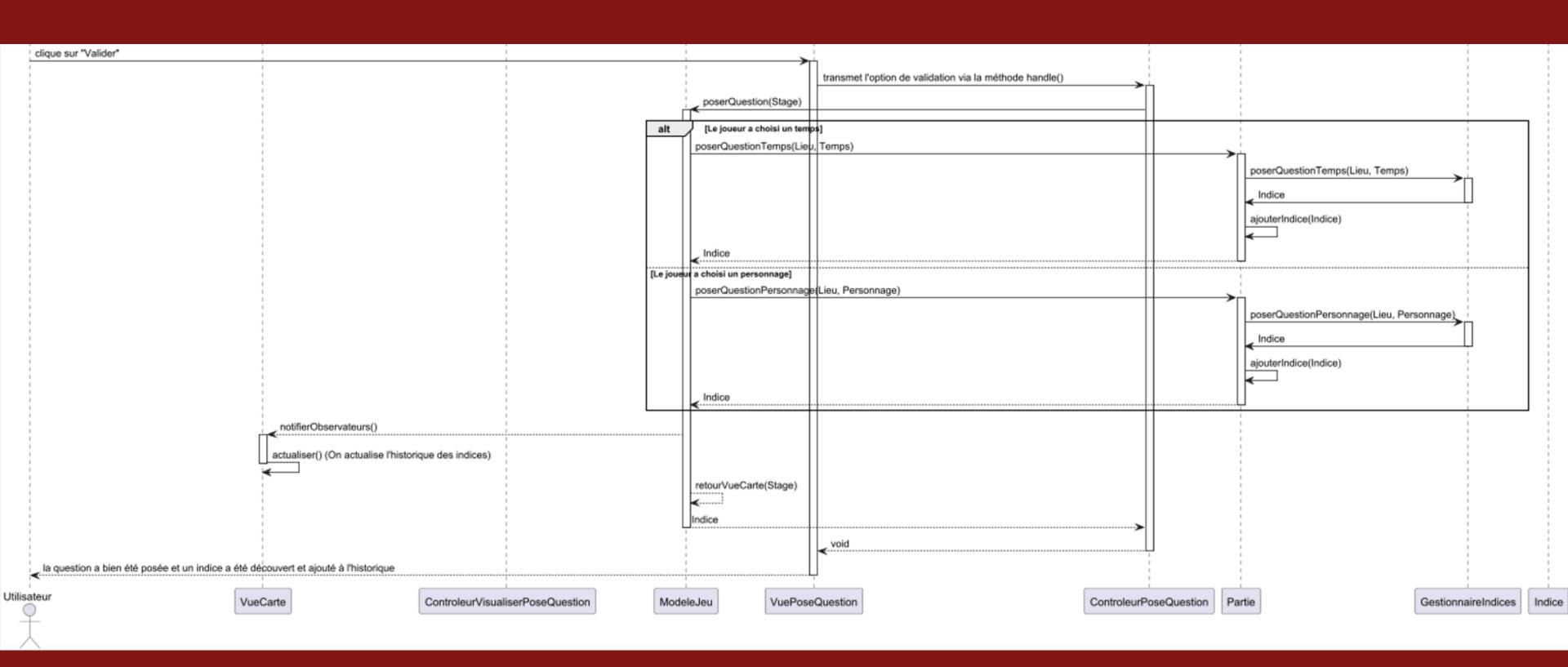


# Diagramme de classe partie MVC

### Diagramme de séquence (poser une question)



### Diagramme de séquence (poser une question)



	Objectifs principaux	Rôle des membres du groupe
Itération 1	<ul> <li>Compréhension de Choco-Solver</li> <li>Réaliser des prototypes</li> <li>Mettre en place les bases du jeu</li> </ul>	<ul> <li>Mathieu : se familiariser avec Choco-Solver</li> <li>Enzo et Corentin : mettre en place les bases du jeu</li> </ul>
Itération 2	<ul> <li>Développer l'interface graphique</li> <li>Implémenter les fonctionnalités principales de jeu</li> </ul>	Chacun s'occupera de différentes fonctionnalités et de leurs représentations graphiques
Itérations 3 et 4 (IA de déduction)	<ul> <li>Développer deux versions</li> <li>Comparer les performances</li> <li>Intégrer les suggestions de l'IA à l'interface graphique</li> </ul>	<ul> <li>Mathieu: Implémenter l'IA reposant sur Choco-Solver</li> <li>Enzo et Corentin: Implémenter l'IA heuristique</li> </ul>
ltérations 5 et 6 (IA d'assistance)	<ul> <li>Développer deux versions</li> <li>Comparer les performances</li> <li>Permettre une interaction fluide entre le joueur et l'IA</li> </ul>	<ul> <li>Mathieu: Implémenter l'IA reposant sur Choco-Solver</li> <li>Enzo et Corentin: Implémenter l'IA heuristique et les interactions</li> </ul>
Itération 7	<ul> <li>Corriger les éventuels bugs et erreurs</li> <li>Finaliser ce qui n'a pas pu être terminé</li> </ul>	Aucune répartition n'a été définie pour le moment, car nous ne pouvons pas nous projeter aussi loin.

### Itération 1

### Choco-Solver (Comparaison de performance)

# Traveling Salesman Problem (TSP)

### Contraintes

#### 17 villes

- Heuristique : 327 ms
- Choco Solver: 4171 ms

#### 30 villes:

- Heuristique : Java heap space (mémoire insuffisante)
- Choco Solver: 13295 ms

- Chaque ville doit être visitée une seule fois.
- Le trajet commence et se termine à la même ville.
- Trouver un chemin qui minimise la distance totale ou le coût total.

# Présentation Interface Kronologic