Basic_dev (groupe 1)

Henri BOST
Ivann VASIC

Rapport de conception projet AP4B

Table des matières

Introduction

Le projet s'inscrit dans le cadre de la conception et du développement d'un jeu inspiré de *Turing Machine*, adapté à une interface informatique interactive. L'objectif principal est de proposer une expérience utilisateur fluide et engageante, à travers une interface graphique développée avec la bibliothèque Swing en Java. Le jeu repose sur des mécanismes de logique et de déduction, où les joueurs doivent valider des hypothèses en fonction de critères donnés, en utilisant un plateau de jeu et une feuille de notes virtuelle.

La phase de conception de ce projet a été guidée par des principes fondamentaux de modélisation orientée objets, permettant une structuration claire et évolutive du système. Les diagrammes UML, tels que les cas d'utilisation, les diagrammes d'états, de classes et de séquence, ont été utilisés pour modéliser les interactions, les transitions et les relations entre les différentes entités du jeu. Ces outils garantissent une traduction fluide de la conception en code, tout en facilitant les futures adaptations et extensions.

Ce rapport présente les fondements de la conception du projet, en s'appuyant sur les diagrammes UML et les choix techniques effectués. La conception met en avant une architecture modulaire, où les classes abstraites et les scénarios spécifiques assurent une logique de jeu adaptable. L'utilisation de Swing permet d'offrir une interface intuitive, avec un plateau de jeu et une feuille de notes pour améliorer l'expérience des joueurs.

Ainsi, ce document constitue une base solide pour comprendre l'architecture actuelle du projet et servir de référence pour les itérations futures dans le développement.

Organisation des tâches

L'ensemble des éléments de conception du projet ont été réalisés en collaboration, lors des séances de TP ou lors de séances hors emploi du temps. Ainsi la compréhension du projet est maximale pour tous les membres du groupe et le travail est équitablement réparti.

Pour la partie implémentation, nous avons divisé le travail, Henri a fait la partie interface et Ivann s'est concentré sur la partie MVC (Mouvement Vue Contrôle).

Enfin nous avons procédé ensemble à la mise en commun et à la finalisation du projet.
La quantité de travail est équivalente entre les différents membres du groupe.

Adaptation thématique et présentation du jeu

La consigne de ce projet nous indiquait de transposer les règles du jeu à des éléments en rapport avec l'UTBM. Le jeu n'étant plus le même qu'avant, il s'appelle désormais **UTBMachine.**
Nous avons créé et implémenté un total de 6 scénarii différents, dont 2 à 4 vérificateurs, 2 à 5 vérificateurs et enfin 2 à 6 vérificateurs.

Ces scénarii n'utilisent plus les comparaisons logiques mais différentes phrases, affirmations et hypothèses pour permettre au joueur de découvrir **où se déroulera le prochain évènement UTBM, qui l'organisera et quel est l'effectif attendu.** Tout comme dans le jeu original, le joueur peut choisir 1 élément par catégories et chaque catégorie comporte 5 choix différents.

Voici les choix que nous avons retenus :

Lieux : Axone, Foyer Belfort, MDE Sevenans, La Poudrière, Foyer Montbéliard.
Organisateur : AE, SkiUT, BDS, CrunchTime, Gala.
Nombre d'invités : 30, 75, 120, 200, 350.

Exemple du vérificateur no1 du scénario no1 :

Vérificateur 1

- Critère A (vrai) : Le lieu est un Foyer.
- Critère B : Le lieu est Axone ou La Poudrière.
- Critère C : Le lieu est MDE Sevenans uniquement.

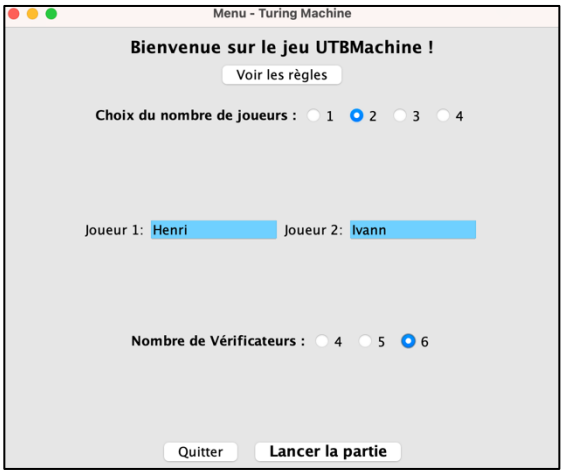
(*Options restantes après test : Foyer Belfort, Foyer Montbéliard.*)

Par exemple le premier vérificateur du scénario no1 est composé de 3 critères dont deux faux.

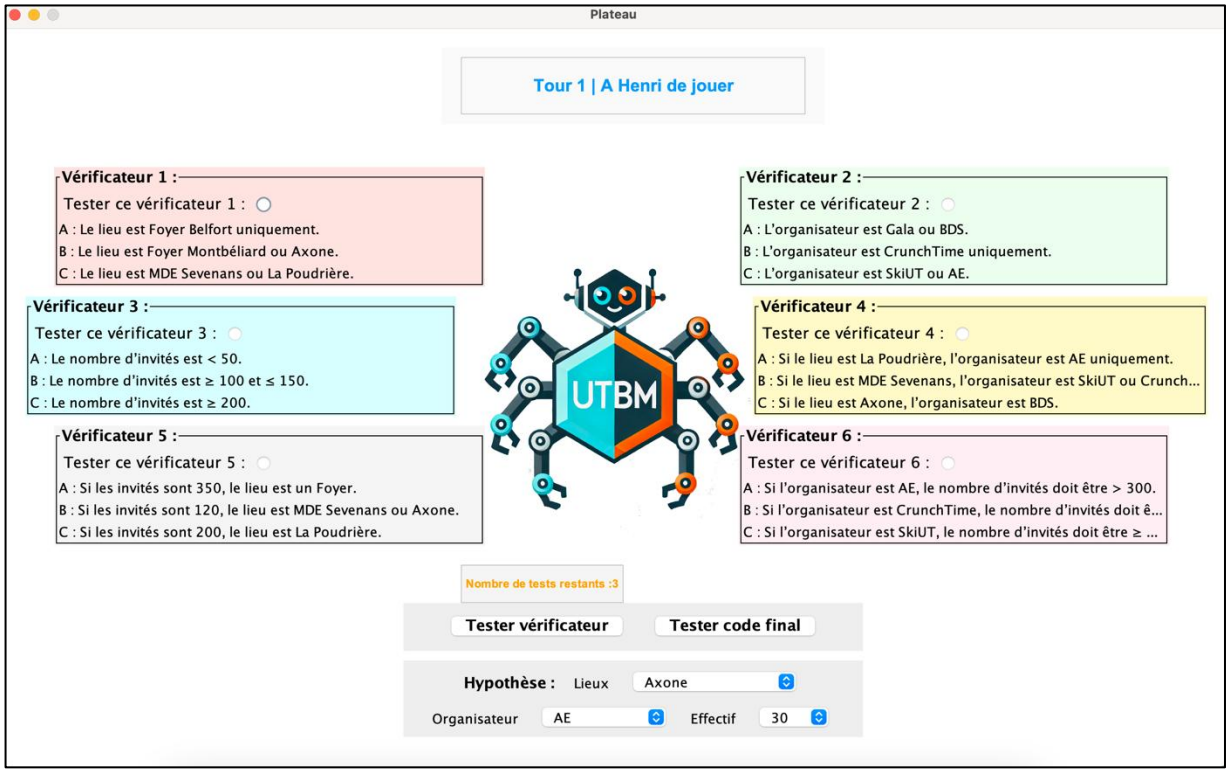
Une fois la bon critère déterminé le joueur n'aura plus que deux possibilités pour le choix des lieux.
Vous trouverez en annexe de ce rapport l'ensemble de nos scenarii et les réponses associées.

Déroulement d'une partie :

Le but du jeu est le suivant : être le premier à deviner où se déroulera le prochain évènement UTBM, qui l'organisera et quel sera l'effectif !

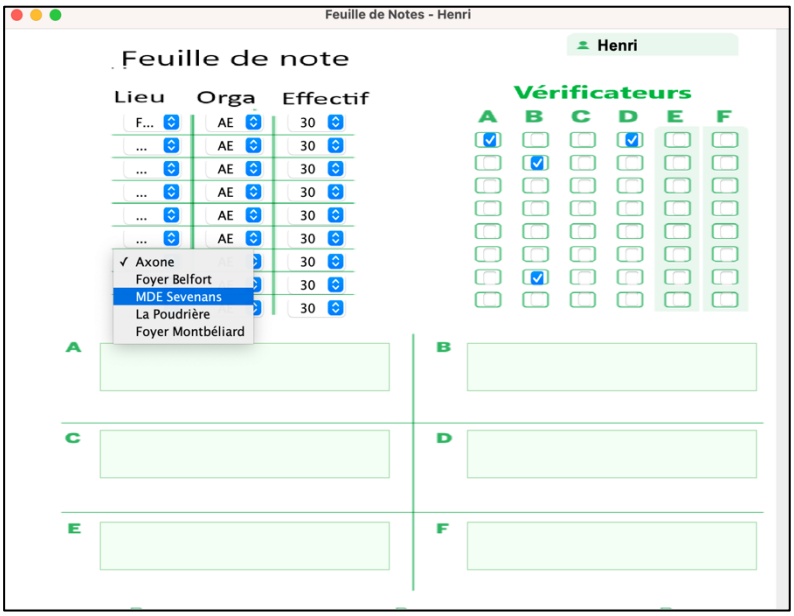


En premier on sélectionne le nombre de joueurs, on entre leurs noms et on choisit le nombre de vérificateurs. Le menu offre également la possibilité de lire les règles et de quitter.



Le plateau du jeu permet au(x) joueur(s) de testes les vérificateurs, prendre des notes via la feuille de notes ou encore de tenter un code final afin de gagner.
Le plateau est interactif, il affiche le joueur qui doit jouer, le numéro du tour ainsi que le nombre de tests restants.

Les joueurs testent des vérificateurs jusqu'à ce que l'un d'entre eux obtienne la bonne combinaison.



La feuille de note permet au joueur de noter ses tentatives et indices.
Elle affiche également le nom de son propriétaire.

Lorsqu'un joueur trouve le bon code, une fenêtre apparait pour le féliciter et la partie s'achève.

Diagrammes UML et conception

Cette partie du rapport se divise en deux sous parties, la première sera la conception et traite des choix de conception autours du projet. La deuxième est consacrée aux diagrammes UML et à leur explication.

Conception

Cette section détaille les choix de conception effectués pour structurer et développer le projet. L'objectif principal était de garantir une architecture claire, modulaire et évolutive, tout en respectant les exigences du jeu inspiré de *Turing Machine*. Les soussections suivantes expliquent les objectifs de conception, la structure architecturale du projet, les interactions entre ses composants, et les choix techniques.

1.Objectifs de la conception

Les principaux objectifs de la conception étaient les suivants :

- Assurer la modularité et l'évolutivité du code : afin d'ajouter facilement de nouveaux scénarios ou des fonctionnalités supplémentaires sans perturber l'ensemble du système.

- Créer une structure globale du projet basée sur les objets (POO)

2. Structure et architecture du projet

Le projet est structuré selon une architecture orientée objets, où chaque entité principale du jeu est représentée par une classe ou un ensemble de classes. L'architecture du projet est la suivante :

- La classe abstraite *Scenarii* encapsule la logique de base commune à tous les scénarios, et ses sous-classes (*Cas1*, *Cas2*, etc.) implémentent les règles spécifiques à chaque scénario.

- La classe *Joueur* modélise les actions possibles d'un joueur (proposer une hypothèse, consulter les notes, etc.).

- La classe *Game* centralise la logique principale du jeu, en orchestrant les interactions entre les scénarios, le joueur, et l'interface utilisateur.

Cette architecture favorise la modularité, car chaque composant peut être modifié ou étendu indépendamment des autres.

3. Interaction entre les composants

Les interactions entre les différentes entités du système suivent une logique simple et structurée :

- **Flux principal du jeu** :

    - L'utilisateur choisit un mode de jeu (solo, compétition, coopération) via l'interface.

    - Le système choisit aléatoirement un scénario basé sur le nombre de vérificateurs choisis.

    - Le joueur soumet une hypothèse via l'interface, qui transmet les informations à la classe *Game*.

    - *Game* compare l'hypothèse soumise avec les réponses du scénario courant (gérées par les sous-classes de *Scenarii*).

    - Les résultats sont retournés et affichés à l'utilisateur.

- **Plateau et feuille de notes** :

- L'interface graphique propose une feuille de notes où le joueur peut enregistrer ses observations tout comme dans le jeu physique.

- Les vérificateurs disponibles sont affichés sur le plateau, et le joueur peut interagir avec eux pour tester ses hypothèses.

- **Limitation des actions** :
  - Le joueur est limité à trois vérifications par tour, conformément aux règles inspirées de *Turing Machine*.

4. Choix techniques

Certains choix techniques ont été effectués pour répondre aux objectifs de conception et garantir la compatibilité avec le langage Java et ses outils. Ces choix incluent :

- **Utilisation de Swing** :

  - Swing a été choisi pour l'interface utilisateur en raison de sa simplicité, de son intégration native avec Java, et de sa documentation abondante. Swing permet de créer des interfaces multi-fenêtres et des interactions utilisateur claires grâce à ses composants comme JFrame, JPanel, et JTextArea.

- **Modélisation des scénarios** :

  - Une classe abstraite Scenarii définit une structure commune pour tous les scénarios. Chaque sous-classe (Cas1, Cas2, etc.) implémente les vérificateurs et les réponses spécifiques à un scénario donné. ○ Cette approche facilite l'ajout de nouveaux scénarios en étendant simplement la classe abstraite.

- **Gestion des hypothèses** :

  - Les hypothèses du joueur sont saisies via des champs de texte (JTextField) dans l'interface, puis transmises à la classe Game pour validation. ○ Un bouton "Vérifier" compare l'hypothèse avec la réponse correcte et retourne un résultat (victoire, erreur, ou nouvelle tentative).

- **Génération aléatoire de scénarios** :

  - Une méthode dédiée dans la classe Joueur génère un scénario aléatoire en fonction du nombre de vérificateurs choisi par le joueur.

5. Conception modulaire et extensible

L'architecture est conçue pour être modulaire, permettant des évolutions futures avec un impact minimal sur le code existant. Les points suivants garantissent cette modularité :

- **Extensibilité des scénarios** :

- La classe abstraite Scenarii permet d'ajouter facilement de nouveaux scénarios en définissant les vérificateurs et les réponses spécifiques à ces scénarios. ○ Les relations d'héritage entre Scenarii et ses sous-classes (Cas1, Cas2, etc.) simplifient la gestion des scénarios.

- **Modularité de l'interface** :

  - L'interface Swing est indépendante de la logique métier. Cela permet de remplacer ou d'améliorer l'interface graphique sans affecter la logique du jeu. ○ Les panneaux (plateau et feuille de notes) peuvent être adaptés ou étendus pour inclure de nouvelles fonctionnalités, comme des statistiques de jeu ou des animations.

- **Séparation logique et graphique** :

  - La logique du jeu est isolée dans *Game* et *Scenarii*, ce qui facilite les tests et la maintenance.
  - L'interface graphique agit comme une couche indépendante pour interagir avec l'utilisateur.

Diagrammes UML

La phase de création des diagrammes UML a été le centre de notre réflexion, elle nous a permis de mettre au clair nos pensées d'une manière structurée et d'arriver à un résultat final fonctionnel.

Après une analyse du fonctionnement de notre projet nous avons conclu que son fonctionnement d'un point de vue utilisateur et développeur peut être entièrement saisi avec 4 diagrammes : le diagramme de cas d'utilisations (use case), le diagramme de classes, le diagramme de séquence et le diagramme d'états-transitions

Les diagrammes UML que nous avons élaborés ont joué un rôle crucial dans la phase de conception de notre projet. Ils nous ont permis de structurer efficacement les interactions utilisateur, les entités principales du système, et les flux dynamiques du jeu. En représentant graphiquement les différentes dimensions du projet, nous avons pu identifier les relations et les responsabilités des différents composants, tout en anticipant les interactions entre l'utilisateur et le système.

Après une analyse approfondie, nous avons identifié quatre types de diagrammes essentiels pour modéliser notre projet :

1. **Diagramme de cas d'utilisation (Use Case)** : Représente les interactions entre les acteurs (joueurs) et les fonctionnalités principales du système, telles que la proposition d'un code ou la validation des hypothèses.

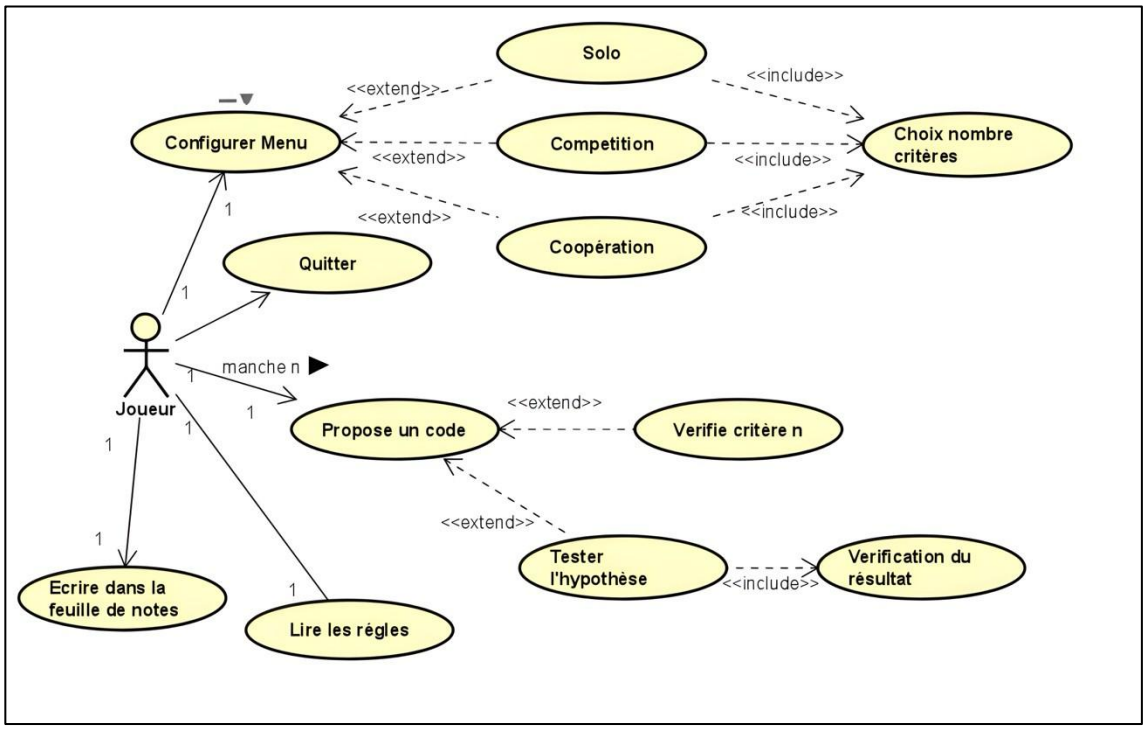
2. **Diagramme de classes** : Définit la structure du système en modélisant les entités principales et leurs relations, comme les scénarios, le joueur, et l'interface.

3. **Diagramme de séquence** : Modélise les interactions dynamiques entre les composants pendant une partie, notamment les échanges entre le joueur, l'interface utilisateur, et le système du jeu.

4. **Diagramme d'états-transitions** : Décrit les différents états possibles du jeu et les transitions entre ces états, comme la progression d'un tour ou la validation d'une hypothèse.

Ces diagrammes offrent une vision d'ensemble du projet, en répondant aux questions fondamentales : **que fait le système, comment est-il structuré, et comment les entités interagissent-elles ?** Dans les sections suivantes, nous décrivons chacun de ces diagrammes en détail, en soulignant leur rôle et leur contribution à la mise en œuvre du projet.

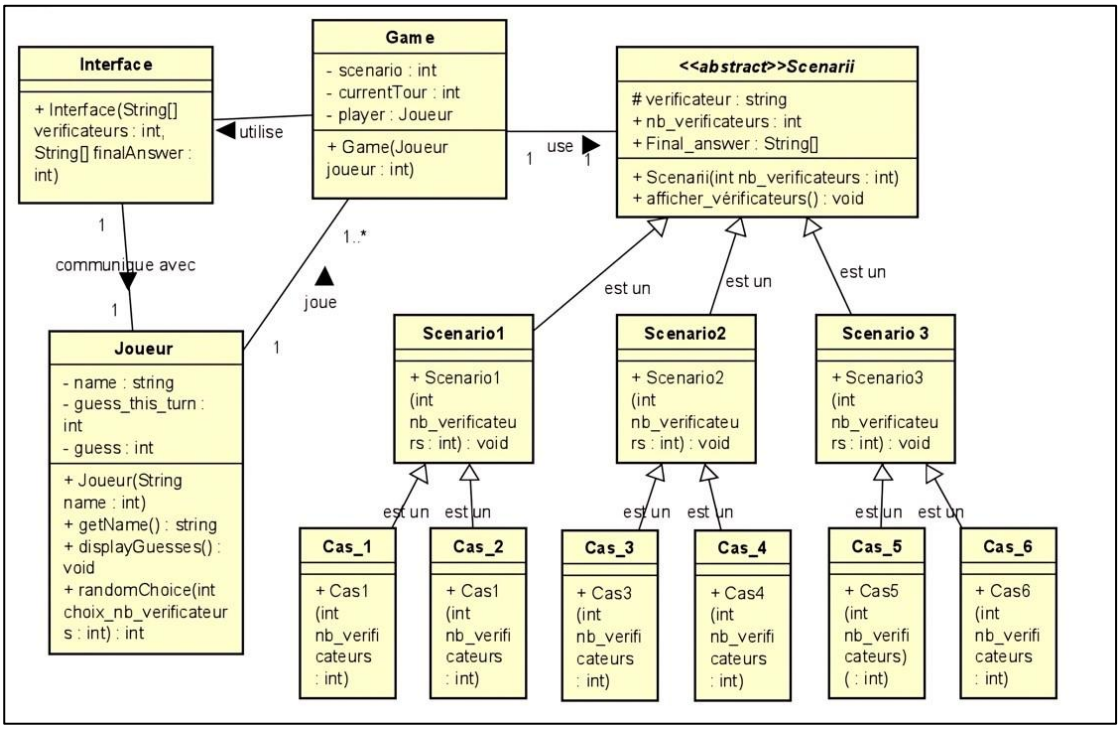
Diagramme de cas d'utilisation (Use Case) :



Analysedu Use Case :

Le diagramme de cas d'utilisation illustre les interactions principales entre le joueur et le système. Il met en évidence les fonctionnalités essentielles du jeu, comme la configuration du menu, le choix du mode (solo, compétition ou coopération), la proposition d'un code, et la vérification des hypothèses.
Chaque action inclut ou étend d'autres cas, montrant la flexibilité du système, comme la validation d'un critère ou l'écriture dans la feuille de notes.
Ce diagramme met également en lumière les différentes étapes du jeu, garantissant que toutes les interactions utilisateur sont bien prises en compte. Enfin, il sert de base pour structurer l'interface utilisateur et la logique métier.

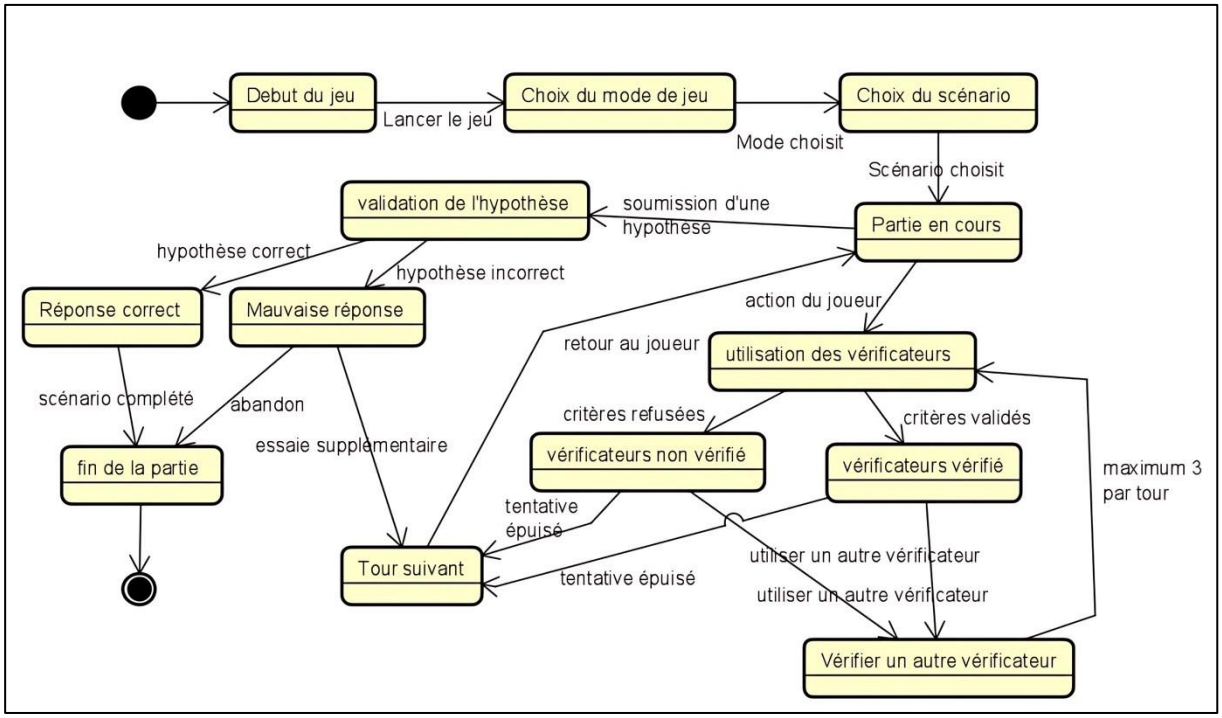
Diagramme de classes :



Analyse du Diagramme de Classes :

Le diagramme de classes représente la structure statique du projet en mettant en avant les entités principales (classes) et leurs relations.
La classe abstraite Scenarii joue un rôle central en définissant une architecture commune pour les sous-classes (Cas1 à Cas6), chacune représentant un scénario spécifique.
Les relations entre *Game*, *Interface*, et *Joueur* montrent une séparation claire des responsabilités : *Game* gère la logique, *Interface* s'occupe de l'interaction utilisateur, et *Joueur* modélise les actions du joueur. Ce diagramme facilite la compréhension de la modularité et de l'extensibilité du système.

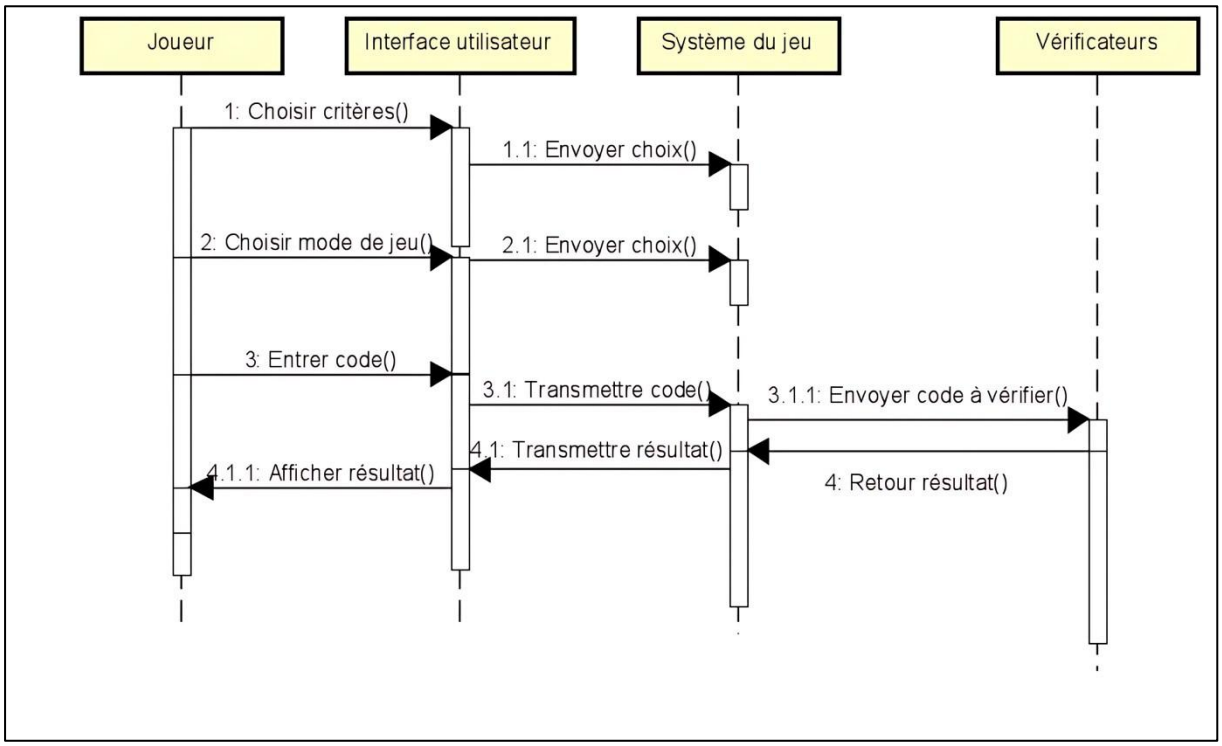
Diagramme d'États-Transitions :



Analyse du Diagramme d'États-Transitions :

Le diagramme d’états-transitions modélise le fonctionnement du jeu à travers les différents états possibles, tels que le début de la partie, le choix du mode de jeu, la validation des hypothèses, ou la fin de la partie. Il illustre les transitions qui dépendent des actions du joueur (soumission d'une hypothèse, utilisation des vérificateurs) et des résultats obtenus (hypothèse correcte ou incorrecte). Chaque transition est conditionnée par les règles du jeu, comme la limite de trois vérifications par tour. Ce diagramme structure la progression du jeu et aide à anticiper les cas limites ou scénarios complexes.

Diagramme de Séquence :



Analyse du Diagramme de Séquence
Le diagramme de séquence modélise les interactions dynamiques entre les différents composants pendant une partie. Il illustre le flux des messages échangés : le joueur interagit avec l'interface pour choisir un critère ou soumettre un code, l'interface transmet ces données à Game, qui les valide en appelant les vérificateurs. Le résultat est ensuite renvoyé à l'interface et affiché au joueur. Ce diagramme capture les étapes clés d'une interaction typique, garantissant une synchronisation correcte entre l'utilisateur, l'interface, et la logique métier.

Conclusion

Ce projet de jeu inspiré de *Turing Machine* a mis en évidence l'importance d'une conception bien structurée pour le développement d'un système interactif et évolutif. La programmation orientée objets (POO) a joué un rôle central dans cette démarche, en permettant une séparation claire des responsabilités entre les entités principales, telles que les scénarios, le joueur, l'interface graphique, et la logique de gestion du jeu. Cette architecture modulaire garantit non seulement la clarté du code, mais aussi sa maintenabilité et son extensibilité, facilitant l'ajout de nouveaux scénarios ou de fonctionnalités futures.

Les diagrammes UML ont été essentiels pour conceptualiser et organiser les idées, en passant de l'abstraction à une implémentation pratique. Le diagramme de cas d'utilisation a permis de définir les interactions principales entre l'utilisateur et le système, tandis que le diagramme d'états-transitions a modélisé les différentes étapes du jeu. Le diagramme de classes a fourni une vision claire des relations entre les entités, et le diagramme de séquence a détaillé les interactions dynamiques entre les composants pendant une partie. Ces outils de modélisation ont assuré une cohérence globale tout en anticipant les besoins futurs du projet.

Implémentation du pattern MVC

Model

Le **Model** regroupe toutes les classes et la logique qui décrivent :

- L'état du jeu,
- Les règles,
- Les scénarios,
- Les entités nécessaires au fonctionnement interne.

Dans notre cas :

- La classe Game représente la logique principale du jeu :
  - Elle stocke la liste des joueurs et le nombre de vérificateurs.
  - Elle sélectionne et contient un scénario particulier.
  - Après le pattern MVC implémenté, elle ne gère plus directement l'interface utilisateur (elle ne propose plus de saisie au clavier, par exemple).
- La classe Scenarii (et ses sous-classes comme Cas1, Cas2, etc.) décrit les différents scénarios possibles.
  - Elle définit notamment les vérificateurs (verfificateurs[]), les choix corrects (correctChoices[]) et la réponse finale (Final_answer).
  - Elle offre des méthodes pour valider un vérificateur (validateVerifier(...)) et pour valider l'hypothèse finale (validateGuess(...)).
- La classe Joueur est également un élément du Modèle (elle définit le nom du joueur et la façon dont il choisit un scénario aléatoire, etc.).

L'ensemble de ces classes gère la **logique** et l'**état** du jeu, sans s'occuper de l'affichage graphique.

View

La **Vue** correspond aux interfaces graphiques qui interagissent avec l'utilisateur :

1. La classe Menu constitue la **Vue** du **Menu** d'accueil.
    - Elle affiche une fenêtre permettant de sélectionner :
        - Le nombre de joueurs,
        - Le nom de chaque joueur,
        - Le nombre de vérificateurs,
        - Et de lancer la partie ou de consulter les règles.
    - Elle délègue les données (liste de joueurs, nb de vérificateurs) à son contrôleur (MenuController) et lance ensuite la partie (Plateau, Feuilles de notes, etc.).
2. La classe Plateau constitue la **Vue** principale du **jeu** lui-même.
    - Elle affiche les vérificateurs, les boutons pour tester un vérificateur, pour tester le code final, etc.
    - Elle affiche aussi le numéro du tour, le joueur courant, le nombre de tests restants, etc.
    - Lorsque l'utilisateur clique sur un bouton (ex. "Tester vérificateur"), Plateau ouvre une fenêtre d'options (A, B, C) puis délègue l'action à un **GameController**.
    - Elle ne gère pas la logique des tours ou la validation interne. Elle se contente de **demander** au contrôleur si la validation est réussie ou non, et **met à jour** l'interface en conséquence.
3. Pour chaque joueur, une **FeuilleNotes** (autre vue optionnelle) peut être ouverte afin de prendre des notes. Elle aussi ne gère que l'aspect d'affichage/édition, sans logique métier.

La Vue n'a pas de code métier (règles du jeu) : elle reçoit simplement des instructions du contrôleur, et affiche ou récupère les informations auprès de l'utilisateur.

Controller

Le **Controller** relie la Vue et le Modèle. Il reçoit les requêtes de la Vue (ex. « le joueur veut tester le vérificateur n°2 avec le choix C ») et appelle les méthodes appropriées du Modèle pour effectuer la validation. Ensuite, il informe la Vue du résultat (ex. « vérificateur validé ! » ou « choix incorrect »).
On distingue deux contrôleurs principaux :

1. **MenuController**
    - Il stocke/récupère les infos saisies dans le menu : liste de joueurs, nombre de vérificateurs.
    - Le Menu met à jour ces informations dans MenuController quand on clique sur « Lancer la partie ».
    - Ensuite, on utilise ces données pour lancer le jeu (création du Game, etc.).
2. **GameController**
    - Il gère la logique des tours : combien de vérifications il reste, quel est le joueur actuel, quand passer au tour suivant, etc.
    - Il fait le lien entre la classe Plateau (View) et la classe Game (Model).
    - Lorsque Plateau appelle gameController.testVerifier(index, choix), c'est le contrôleur qui :
        1. Appelle game.getScenario().validateVerifier(index, choix).
        2. Met à jour le nombre de tests restants, change éventuellement de joueur.
        3. Retourne un booléen indiquant si le test est validé ou non, pour que la Vue affiche le message correspondant.
    - Lorsque Plateau appelle gameController.testFinalCode(guess), c'est lui qui interroge le scenario.validateGuess(guess) et informe la Vue du résultat.

Ainsi, le contrôleur **centralise** la logique de progression de la partie et s'assure que la Vue se contente d'afficher et de saisir les infos, tandis que le Modèle gère l'état et les règles fondamentales du jeu.

Difficultés rencontrées

Les difficultés du projet se concentrent autour de la partie mise en relation des différents éléments du MVC.

Le développement de chaque partie d'un point de vue unique n'a pas représenté de difficulté majeure bien que la prise en main de Java SWING représente un temps d'apprentissage non négligeable.

La création de l'interface graphique du plateau a été une tache fastidieuse qui nous a demandé beaucoup de réflexion, en particulier pour l'agencement des différents éléments, l'ergonomie et la jouabilité.

La mise en relation de la partie vue avec le modèle et le contrôle nous a demandé de créer des relations entre les différentes classes du projet, vous retrouverez ces relations dans les diagrammes UML.

RETEX et améliorations possibles

Le développement du jeu UTBMachine nous a permis de nous améliorer dans divers domaines de l'informatique en particulier l'architecture (modèle MVC) et la mise en place d'une interface graphique en utilisant Java SWING.

Le modèle MVC permet une relation saine entre les différents éléments du projet, il est idéal pour la gestion de ce genre de projets et permet de modifier l'interface sans altérer la logique du jeu, et inversement.

Bien que complet, le projet est toujours améliorable : l'exploitation du modèle MVC nous permettrais d'apporter ces modifications sans une restructuration des éléments du projet.

On pourrait par exemple imaginer le développement d'un jeu jouable en ligne, ou encore différents modes de jeu, de nouvelles règles et même une interface graphique plus récente en utilisant une autre bibliothèque tel que JavaFX.

Annexes

1) Les diagrammes UML

USE CASE

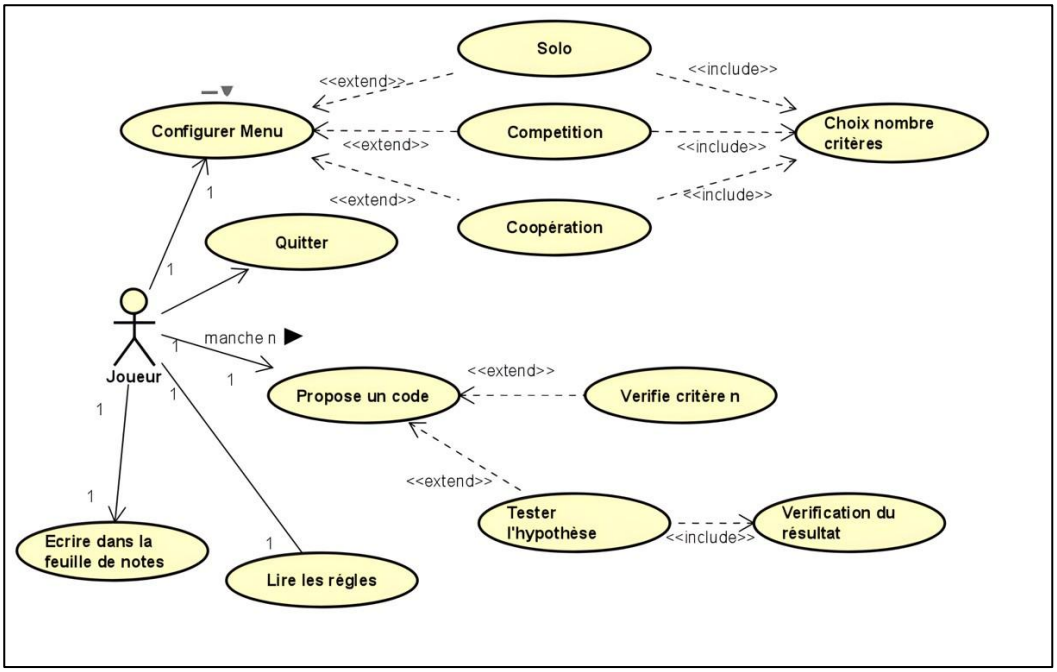


DIAGRAMME DE CLASSES

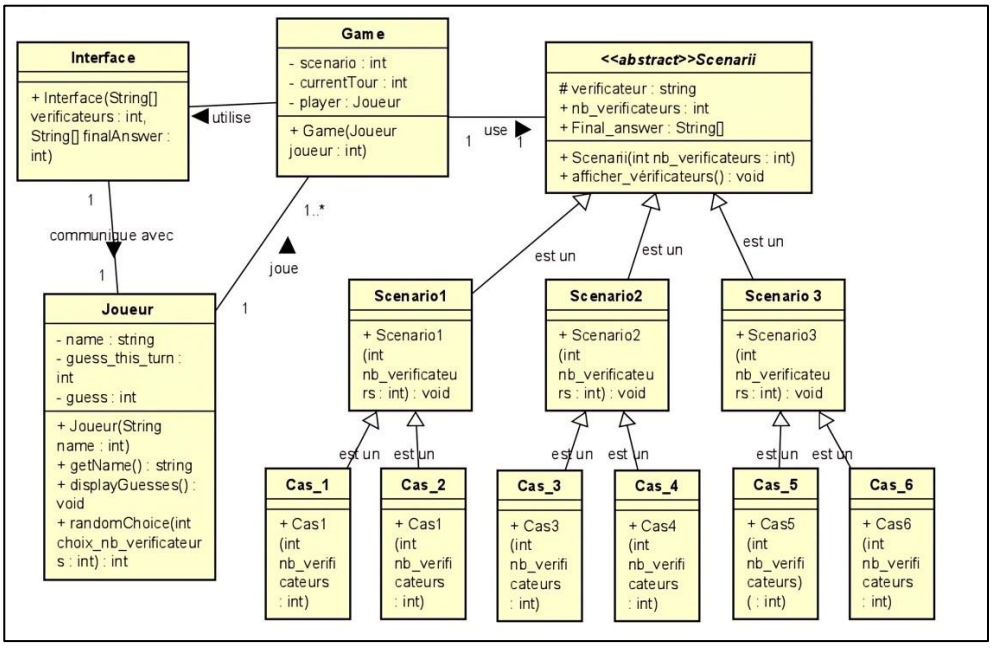


DIAGRAMME DE SEQUENCE

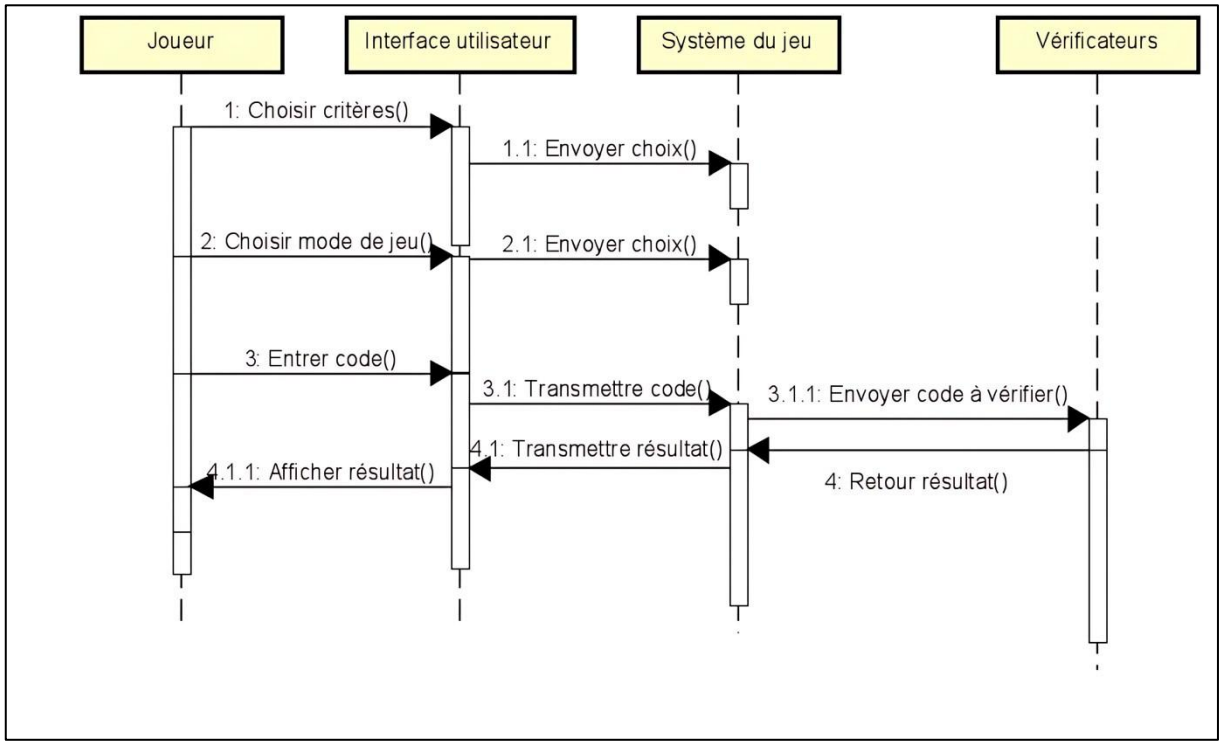
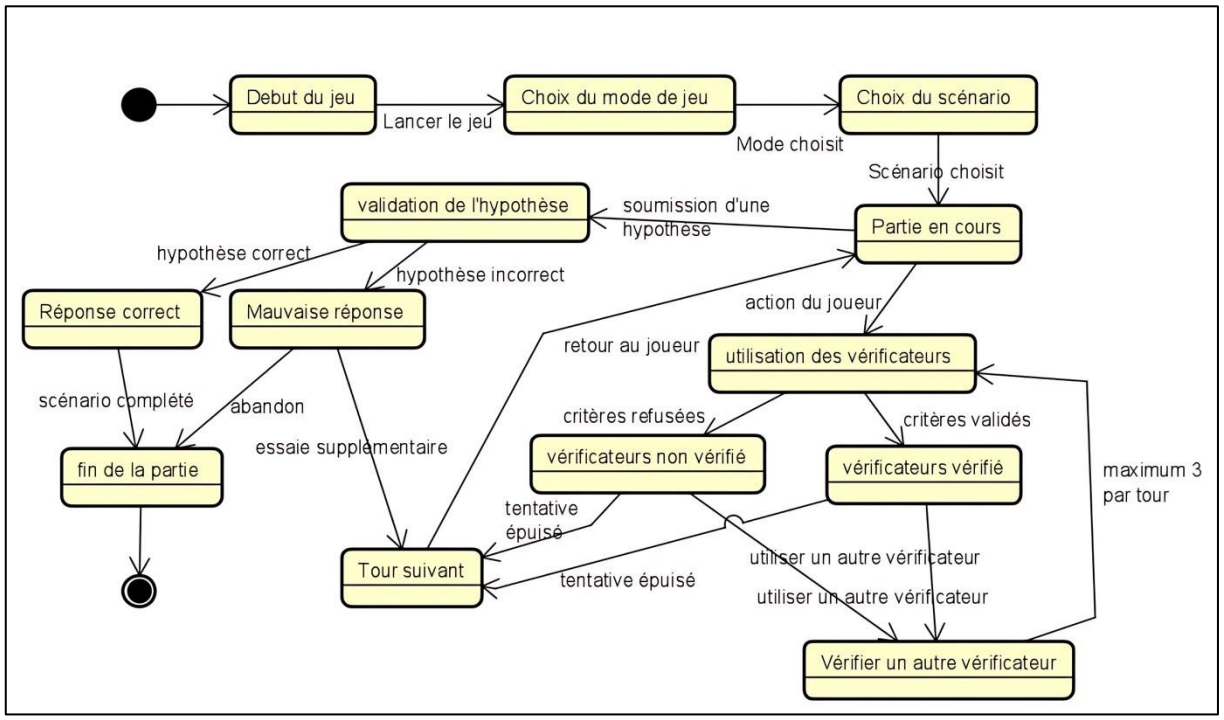


DIAGRAMME D'ETATS-TRANSITIONS



2) Les scenarii du jeu

Paramètres possibles :

1. **Lieu : Axone, Foyer Belfort, MDE Sevenans, La Poudrière, Foyer Montbéliard.**
2. **Organisateur : AE, SkiUT, BDS, CrunchTime, Gala.**
3. **Nombre d'invités : 30, 75, 120, 200, 350.**

Scénario 1 : 4 vérificateurs

Code à deviner : *Foyer Belfort, BDS, 75 invités*.

Vérificateur 1

- Critère A (vrai) : Le lieu est un Foyer.
- Critère B : Le lieu est Axone ou La Poudrière.
- Critère C : Le lieu est MDE Sevenans uniquement.
  (*Options restantes après test : Foyer Belfort, Foyer Montbéliard.*)

Vérificateur 2

- Critère A (vrai) : L'organisateur est BDS ou CrunchTime.
- Critère B : L'organisateur est AE uniquement.
- Critère C : L'organisateur est Gala ou SkiUT.
  (*Options restantes après test : BDS, CrunchTime.*)
-

Vérificateur 3

- Critère A (vrai) : Le nombre d'invités est ≤ 100.
- Critère B : Le nombre d'invités est > 200.
- Critère C : Le nombre d'invités est < 50.
  (*Options restantes après test : 75 uniquement.*)

Vérificateur 4

- Critère A (vrai) : Si le lieu est le Foyer Belfort, l'organisateur est BDS ou AE.
- Critère B : Si le lieu est Axone, l'organisateur est SkiUT uniquement.
- Critère C : Si le lieu est MDE Sevenans, l'organisateur est CrunchTime.
  (*Options restantes après test : Confirme Foyer Belfort et BDS.*)

Scénario 2 : 5 vérificateurs

Code à deviner : *La Poudrière, CrunchTime, 200 invités*.

Vérificateur 1

- Critère A (vrai) : Le lieu est La Poudrière ou Axone.
- Critère B : Le lieu est Foyer Montbéliard ou MDE Sevenans.
- Critère C : Le lieu est Foyer Belfort uniquement.
  (*Options restantes après test : La Poudrière, Axone.*)

Vérificateur 2

- Critère A (vrai) : L'organisateur est CrunchTime ou Gala.
- Critère B : L'organisateur est AE ou SkiUT.
- Critère C : L'organisateur est BDS uniquement.
  (*Options restantes après test : CrunchTime, Gala.*)

Vérificateur 3

- Critère A (vrai) : Le nombre d'invités est ≥ 100.
- Critère B : Le nombre d'invités est ≤ 150.
- Critère C : Le nombre d'invités est < 50.
  (*Options restantes après test : 200 uniquement.*)

Vérificateur 4

- Critère A (vrai) : Si le lieu est La Poudrière, les invités doivent être ≥ 150.
- Critère B : Si le lieu est Axone, les invités doivent être < 100.
- Critère C : Si le lieu est MDE Sevenans, les invités doivent être ≥ 300.
  (*Options restantes après test : Confirme 200 invités et La Poudrière.*)

Vérificateur 5

- Critère A (vrai) : Si l'organisateur est CrunchTime, le lieu est La Poudrière ou Axone.
- Critère B : Si l'organisateur est AE, le lieu est un Foyer.
- Critère C : Si l'organisateur est Gala, le lieu est Axone uniquement.
  (*Options restantes après test : Confirme CrunchTime.*)

Scénario 3 : 6 vérificateurs

Code à deviner : *MDE Sevenans, SkiUT, 120 invités*.

Vérificateur 1

- Critère A (vrai) : Le lieu est MDE Sevenans ou La Poudrière.
- Critère B : Le lieu est Foyer Montbéliard ou Axone.
- Critère C : Le lieu est Foyer Belfort uniquement.
  (*Options restantes après test : MDE Sevenans, La Poudrière.*)

Vérificateur 2

- Critère A (vrai) : L'organisateur est SkiUT ou AE.
- Critère B : L'organisateur est CrunchTime uniquement.

- Critère C : L'organisateur est Gala ou BDS.
  (*Options restantes après test : SkiUT, AE.*)

Vérificateur 3
- Critère A (vrai) : Le nombre d'invités est ≥ 100 et ≤ 150.
- Critère B : Le nombre d'invités est ≥ 200.
- Critère C : Le nombre d'invités est < 50.
  (*Options restantes après test : 120 uniquement.*)

Vérificateur 4
- Critère A (vrai) : Si le lieu est MDE Sevenans, l'organisateur est SkiUT ou CrunchTime.
- Critère B : Si le lieu est La Poudrière, l'organisateur est AE uniquement.
- Critère C : Si le lieu est Axone, l'organisateur est BDS.
  (*Options restantes après test : Confirme MDE Sevenans et SkiUT.*)

Vérificateur 5
- Critère A (vrai) : Si les invités sont 120, le lieu est MDE Sevenans ou Axone.
- Critère B : Si les invités sont 200, le lieu est La Poudrière.
- Critère C : Si les invités sont 350, le lieu est un Foyer.
  (*Options restantes après test : Confirme 120 invités et MDE Sevenans.*)

Vérificateur 6
- Critère A (vrai) : Si l'organisateur est SkiUT, le nombre d'invités doit être ≥ 100 et ≤ 150.
- Critère B : Si l'organisateur est CrunchTime, le nombre d'invités doit être < 100.
- Critère C : Si l'organisateur est AE, le nombre d'invités doit être > 300.
  (*Options restantes après test : Confirme 120 invités et SkiUT.*)

Scénario 4 : 4 vérificateurs
Code à deviner : Foyer Montbéliard, AE, 250 invités.

Vérificateur 1
- Critère A (vrai) : Le lieu est Foyer Montbéliard ou Axone.
- Critère B : Le lieu est La Poudrière ou MDE Sevenans.
- Critère C : Le lieu est Foyer Belfort uniquement.
  (*Options restantes après test : Foyer Montbéliard, Axone.*)

Vérificateur 2
- Critère A (vrai) : L'organisateur est AE ou BDS.
- Critère B : L'organisateur est Gala ou CrunchTime.
- Critère C : L'organisateur est SkiUT uniquement.
  (*Options restantes après test : AE ou BDS.*)

Vérificateur 3

- Critère A (vrai) : Le nombre d'invités est ≥ 200.
- Critère B : Le nombre d'invités est < 100.
- Critère C : Le nombre d'invités est ≥ 150 et ≤ 200.
  (*Options restantes après test : 250 invités.*)

Vérificateur 4

- Critère A (vrai) : Si l'organisateur est AE, le lieu est Foyer Montbéliard ou La Poudrière.
- Critère B : Si l'organisateur est SkiUT, le lieu est Axone.
- Critère C : Si l'organisateur est BDS, le lieu est MDE Sevenans.
  (*Options restantes après test : Confirme Foyer Montbéliard et AE.*)

Scénario 5 : 5 vérificateurs

Code à deviner : La Poudrière, BDS, 100 invités.

Vérificateur 1

- Critère A (vrai) : Le lieu est La Poudrière ou Foyer Montbéliard.
- Critère B : Le lieu est Foyer Belfort ou MDE Sevenans.
- Critère C : Le lieu est Axone uniquement.
  (*Options restantes après test : La Poudrière, Foyer Montbéliard.*)

Vérificateur 2

- Critère A (vrai) : L'organisateur est BDS ou Gala.
- Critère B : L'organisateur est AE ou CrunchTime. • Critère C : L'organisateur est SkiUT uniquement.
  (*Options restantes après test : BDS ou Gala.*)

Vérificateur 3

- Critère A (vrai) : Le nombre d'invités est < 150.
- Critère B : Le nombre d'invités est > 250.
- Critère C : Le nombre d'invités est ≥ 150.
  (*Options restantes après test : 100 invités.*)

Vérificateur 4

- Critère A (vrai) : Si le lieu est La Poudrière, l'organisateur est BDS ou Gala.
- Critère B : Si le lieu est Foyer Montbéliard, l'organisateur est AE.
- Critère C : Si le lieu est Axone, l'organisateur est SkiUT.
  (*Options restantes après test : Confirme La Poudrière et BDS.*)

Vérificateur 5

- Critère A (vrai) : Si le nombre d'invités est 100, le lieu est La Poudrière.
- Critère B : Si le nombre d'invités est 200, le lieu est MDE Sevenans.
- Critère C : Si le nombre d'invités est < 100, le lieu est Axone.
  (*Options restantes après test : Confirme 100 invités et La Poudrière.*)

Scénario 6 : 6 vérificateurs
Code à deviner : Axone, Gala, 150 invités.

Vérificateur 1
- Critère A (vrai) : Le lieu est Axone ou La Poudrière.
- Critère B : Le lieu est MDE Sevenans ou Foyer Belfort.
- Critère C : Le lieu est Foyer Montbéliard uniquement.
  (*Options restantes après test : Axone, La Poudrière.*)

Vérificateur 2
- Critère A (vrai) : L'organisateur est Gala ou SkiUT.
- Critère B : L'organisateur est AE ou BDS.
- Critère C : L'organisateur est CrunchTime uniquement.
  (*Options restantes après test : Gala ou SkiUT.*)

Vérificateur 3
- Critère A (vrai) : Le nombre d'invités est ≥ 100 et ≤ 150.
- Critère B : Le nombre d'invités est < 100.
- Critère C : Le nombre d'invités est ≥ 200.
  (*Options restantes après test : 150 invités.*)

Vérificateur 4
- Critère A (vrai) : Si le lieu est Axone, l'organisateur est Gala ou SkiUT.
- Critère B : Si le lieu est La Poudrière, l'organisateur est AE ou BDS.
- Critère C : Si le lieu est Foyer Montbéliard, l'organisateur est CrunchTime.
  (*Options restantes après test : Confirme Axone et Gala.*)

Vérificateur 5
- Critère A (vrai) : Si l'organisateur est Gala, le nombre d'invités est ≤ 150.
- Critère B : Si l'organisateur est SkiUT, le nombre d'invités est > 150.
- Critère C : Si l'organisateur est AE, le nombre d'invités est ≥ 200.
  (*Options restantes après test : Confirme 150 invités et Gala.*)

Vérificateur 6
- Critère A (vrai) : Si le lieu est Axone, le nombre d'invités est ≤ 150.
- Critère B : Si le lieu est La Poudrière, le nombre d'invités est > 150.
- Critère C : Si le lieu est MDE Sevenans, le nombre d'invités est ≥ 200.
  (*Options restantes après test : Confirme 150 invités et Axone.*)