

Décisions et génération des résultats By Year

Principe

Le framework By Year ne connaît pas la structure détaillée des décisions. Il délègue à `utilsByGameUid[gameUid]` et au moteur `generateResultsByGameUid[gameUid]`.

```
sequenceDiagram
    participant U as Unity Decisions UI
    participant B as BackEndBrainCloud
    participant C as gameId_custom_command
    participant T as teams
    participant J as generateResultsJob
    participant G as GenerateResultsGame
    participant R as results

    U->>B: CustomCommand / décision
    B->>C: script game-specific
    C->>T: sauvegarde TeamDecisions
    J->>G: generateCurrentYearResults(actData)
    G->>T: lit décisions de toutes les équipes
    G->>G: marché / RH / production / features...
    G->>R: saveResults
    G->>T: met à jour cash/état année suivante
```

Pattern des moteurs de résultats

Chaque jeu possède un objet `GenerateResults*` avec généralement :

- collecte des décisions de toutes les équipes;
- calcul d'averages/références;

- calcul des scores ou du marché;
- events;
- `finalizeTeamsData()`;
- `saveResults()`;
- reset/préparation de l'année suivante;
- `deleteResults()` pour réouverture/rollback.

Factories serveur

`utils_factory_generate_results.js` instancie le moteur adapté à `ent1`, `hr1`, `mgt1`, `mk1` ou `mk2`.
Ajouter un nouveau jeu By Year exige donc de le brancher dans cette factory.

UI Unity

Le socle visuel se trouve dans `DecisionsByYearControl`, `DecisionsPanelByYear` et les contrôles `SimByYear*`. Les écrans de décisions réels sont souvent dans `GameSpecific/<gameUid>/Src/UI`.

Ajouter une nouvelle décision qui influence les résultats

Ce workflow est le cœur d'une feature By Year.

1. Static data/config

Ajouter la donnée de référence dans la Sheet/JSON game-specific si la décision dépend d'un coût, seuil, segment, feature, pays, employé, etc.

2. Modèle Unity

Ajouter la valeur dans `TeamDecisions<Game>/YearData` et l'UI qui l'édite.

3. Sauvegarde serveur

Ajouter la custom command game-specific. Valider cash, permissions et contraintes côté serveur, même si l'UI fait déjà la validation.

4. Synchronisation équipe

Émettre un message game-specific afin que les autres membres voient la décision. Ajouter la constante dans `GameSpecific/<gameUid>/Src/Messages/MessageType*.cs` si nécessaire.

5. Résultats

Consommer la décision dans `GenerateResults<Game>`. Choisir l'étape correcte : gather → averages → scores → sales/hiring/production → finalize/save.

6. Préparer l'année suivante

Si la décision crée un état durable, vérifier `finalizeTeamsData()` / reset team pour transporter seulement ce qui doit survivre.

7. Évaluation, ranking, report

Si la nouvelle décision change un score, mettre à jour Eval et ranking. Si elle produit un nouveau résultat, ajouter le modèle Unity et le report.

8. Bots

Pour les jeux async avec bots, décider explicitement quelle valeur le bot choisit. Sinon les bots peuvent générer des résultats invalides ou avoir un avantage/désavantage artificiel.

Debug d'un résultat “bizarre”

1. Inspecter la décision persistée avant génération.
 2. Inspecter les données agrégées de toutes les équipes.
 3. Inspecter averages/références.
 4. Inspecter score intermédiaire.
 5. Inspecter résultat final.
 6. Inspecter cash/team state après `finalizeTeamsData()`.
 7. Comparer le modèle Unity désérialisé.
-

Revision #2

Created 2026-09-18 19:19:44 UTC by Mathis

Updated 2026-09-18 20:17:53 UTC by Mathis