

Système d'évaluation commun

Couche commune

Les deux familles utilisent un socle d'évaluation partagé : `StudentEvaluationBase`, `TeamEvaluationBase`, leurs controllers, les stubs et les commandes BrainCloud communes.

Sources de notes

- performance d'activité;
- présence;
- logbook;
- scenario / segments;
- pair evaluation;
- custom evaluations;
- teacher overrides;
- By Module : scores de modules;
- By Year : scores annuels game-specific.

Commandes communes

Le contrat backend inclut notamment `GetEvaluation`, `GetStudentEvaluationStub`, `GetActPerfEvaluation`, `GetLogbookEvaluation`, `GetCustomEvaluation`, `GetPresenceEvaluation`, création/modification/suppression de custom eval, pair eval et teacher overrides.

Pondérations

La simulation conserve les ponderations d'évaluation et d'ActPerf. Les changements sont persistés serveur puis propagés en RTT afin que les contrôles professeur/étudiant se mettent à jour.

Stubs

`SimulationBaseController` maintient des caches de `EvalStudentStub` et `EvalTeamStub`. Ils sont marqués dirty lorsque certains messages arrivent afin d'éviter de recharger toutes les évaluations en permanence.

Spécialisation

```
flowchart TD
    BASE[Évaluation commune] --> BM[By Module]
    BASE --> BY[By Year]
    BM --> MOD[Module scores / correction]
    BY --> YR[Year results / game-specific Eval]
    BASE --> LOG[Logbook]
    BASE --> PRES[Presence]
    BASE --> CUST[Custom / Pair / Overrides]
```

Règle de maintenance

Les calculs propres à un jeu restent dans son moteur Eval; la composition, les ponderations, les stubs et les mécanismes de sauvegarde/override restent communs.

Ajouter une nouvelle composante d'évaluation

Avant d'ajouter une note, décider si elle appartient à **evaluation** (étudiant), **evaluationTeam** (équipe), **ActPerf**, ou si elle est seulement un détail de résultat game-specific.

1. Définir la source de vérité

- score individuel → document `evaluation`;
- score d'équipe/performance → `evaluationTeam`;
- score de module → généré dans le pipeline By Module puis propagé vers `evaluationTeam.actPerf`;
- score annuel game-specific → calcul du moteur Eval du jeu.

2. Définir la pondération

Les évaluations supportent plusieurs stratégies : annuelle/global, individuelle/équipe. Pour ActPerf, `getActPerfScore()` utilise la pondération enregistrée dans la simulation ou, à défaut, la static data `act_perf_pond`.

3. Sauvegarder la note

Ajouter/adapter la commande serveur responsable. Après sauvegarde, envoyer le stub RTT approprié plutôt que de forcer tous les clients à recharger toutes les évaluations.

4. Ajouter le calcul de stub

`utils_eval.js` contient les fonctions de score compact : `getActPerfScore`, `getModuleTypeScore`, `getPresenceScore`, `getLogbookScore`, `getCustomEvalScore` et les builders de stubs étudiant/équipe.

5. Invalider les caches Unity

`SimulationBaseController` et les controllers d'évaluation cachent certains stubs. Le nouveau message doit soit mettre à jour directement la valeur, soit marquer le cache dirty.

Valeur -1 : signification importante

Plusieurs calculs retournent `-1` pour signifier "ignorer / pas encore évalué", tandis que `0` signifie réellement une note nulle. Ne pas convertir automatiquement un `-1` en 0 dans l'UI ou dans un nouvel agrégat.

Checklist évaluation

- Document correct choisi.
- Pondération et type annuel/global définis.

- Cas ongoing vs finished testé.
 - -1 conservé lorsque la note doit être ignorée.
 - RTT/stub envoyé.
 - Cache Unity invalidé.
 - Export Excel/rapport vérifié si la nouvelle note doit y apparaître.
 - By Year sync, async et By Module testés selon le scope.
-

Revision #2

Created 2026-09-18 19:23:41 UTC by Mathis

Updated 2026-09-18 20:08:02 UTC by Mathis