

Évaluations, logbook, scénarios et rankings By Year

Évaluations

`_utils/utils_eval.js` fournit le cadre commun pour activité, présence, logbook, custom evaluations et stubs. Chaque jeu fournit ensuite son `Eval<Game>` via la factory d'évaluation.

Scores game-specific

Les classes `EvalEnt1`, `EvalHr1`, `EvalMgt1`, `EvalMk1` et équivalents calculent notamment scenario score, segment score et activity performance à partir des résultats propres au jeu.

Logbook

Les données de logbook sont persistées dans `teamsLogbooks`. `utils_by_year.deleteYearLogbook()` montre que le logbook et ses évaluations sont liés au cycle annuel et doivent être nettoyés correctement lors d'une réouverture/suppression d'année.

Présence

`startYear()` crée un nouveau stub de présence pour les étudiants. Les helpers `setYearPresence()` / `deleteYearPresence()` maintiennent ces données lors des changements d'état.

Scénarios

Le client utilise `StudentPlayScenarioByYearState` et les pages/scénarios propres au jeu. Le serveur peut calculer le scenario score via la factory d'évaluation game-specific.

Rankings

`_utils/utils_rankings.js` gère les scores et mises à jour de rankings. Chaque jeu peut appliquer son propre calcul avec des fonctions comme `updateRankingsEnt1`, `updateRankingsHr1`, etc.

```
flowchart LR
    RESULTS[Year Results] --> PERF[Act Performance]
    SCEN[Scenario] --> EVAL[Student Evaluation]
    LOG[Logbook] --> EVAL
    PRES[Presence] --> EVAL
    PERF --> EVAL
    RESULTS --> RANK[Rankings]
```

UI

`EvaluationByYearSyncControl`, `EvaluationByYearAsyncControl`, lignes étudiant, detailed reports, logbook controls, presence popup et ranking controls composent la couche commune.

Ajouter un score ActPerf ou une composante de ranking

ActPerf

1. Ajouter/calculer la métrique dans les résultats du jeu.
2. Modifier `Eval<Game>.computeActPerfScore()`.
3. Ajouter la pondération dans `act_perf_pond` si la métrique est configurable.
4. Vérifier le stub envoyé à `evaluationTeam`.
5. Ajouter le label/report Unity.

6. Tester score d'année en cours vs année terminée.

Ranking

Chaque jeu peut définir `updateRankings<Game>()`. Une nouvelle métrique de ranking doit être calculable depuis la donnée persistée après génération; éviter un ranking basé sur un état transitoire uniquement présent en mémoire.

Scénario

Les réponses scénario sont distinctes des décisions annuelles. Si une nouvelle question/segment influence l'évaluation :

1. mettre à jour la static data/scenario;
2. mettre à jour le prefab/page Unity si nouveau type visuel;
3. vérifier `computeScenarioScoreAndSave()` ou `computeSegmentScoreAndSave()` du jeu;
4. vérifier les `Scenario.eScenarioStep` uniquement s'ils contrôlent un flow/tutoriel, pas comme stockage des réponses.

Logbook et présence lors d'une nouvelle année

`startYear()` initialise la présence; reopen doit la supprimer/restaurer correctement. Le logbook est par activité/année et son évaluation est séparée du texte. Toute nouvelle logique annuelle doit conserver ces index d'année.

Debug d'une note globale fausse

1. Inspecter le résultat game-specific brut.
2. Appeler `computeActPerfScore()` avec le même year/team.
3. Inspecter `evaluationTeam`.
4. Inspecter les pondérations simulation/static data.
5. Inspecter le stub Unity et les caches.

Revision #2

Created 2026-09-18 19:19:45 UTC by Mathis

Updated 2026-09-18 20:17:57 UTC by Mathis