

Cycle d'une année, calendrier et jobs

startYear()

`utils_by_year.startYear(actData, updateYearFinished)` incrémente l'année si nécessaire, passe l'état à `ACT_YEAR_STARTED`, enregistre `startTime`, appelle `startYearGameSpec()`, initialise les stubs de présence puis envoie `act_year_started`.

```
stateDiagram-v2
    [*] --> NotStarted
    NotStarted --> YearStarted: startYear
    YearStarted --> Generating: fin de période / commande
    Generating --> YearFinished: génération terminée
    YearFinished --> YearStarted: année suivante
    YearFinished --> [*]: dernière année
```

nextGenerate

`updateNextGenerateDate()` calcule quand le prochain traitement automatique doit avoir lieu. En pratique, les années n'ont pas de génération automatique. En mode calendrier, `nextGenerate` pointe vers la fin de l'année active ou le début de la suivante.

Validation des dates

`validateDates()` corrige les années mal ordonnées ou placées dans le passé et garantit notamment un intervalle minimal de cinq minutes. Si des dates sont déplacées, le professeur peut recevoir `years_date_modified`.

Job planifié

`gatherActsForJob()` récupère les activités dont `nextGenerate < now`. `runActJob()` traite une activité active : si l'année est commencée, elle passe à generating et appelle `generateResultsJob()` ; sinon elle peut démarrer l'année suivante.

generateResultsJob()

1. envoie `act_generating` ;
2. fait jouer les bots si l'activité est async ;
3. génère les events game-specific ;
4. appelle `generateResultsByGameUid[gameUid].generateCurrentYearResults(actData)` ;
5. décide s'il faut démarrer automatiquement l'année suivante ;
6. met à jour `nextGenerate` ;
7. sauvegarde l'activité ;
8. envoie `act_year_finished` lorsque requis.

Scripts de job

Les wrappers se trouvent sous `server_scripts/_job/`, notamment `generate_results_job.js` et le scheduler commun.

Modifier le cycle annuel ou le calendrier sans casser les jobs

Invariants du cycle

- `startYear()` initialise l'année et appelle `startYearGameSpec()` ;
- la génération passe par `generateResultsJob()` ou la commande manuelle équivalente ;
- `nextGenerate` détermine le prochain traitement automatique ;
- les années de pratique ne suivent pas forcément le même calendrier automatique ;
- le client est averti via les messages `act_year_started`, `act_generating`, `act_year_finished`.

Changer la logique de dates

Les fonctions à auditer sont `updateNextGenerateDate()`, `validateDates()`, `dateToUpper()`, `dateToLower()` et `gatherActsForJob()`.

1. Définir le comportement attendu pour sync, async et practice.
2. Vérifier que les dates passées sont corrigées de façon déterministe.
3. Vérifier l'intervalle minimal attendu par `validateDates()`.
4. Mettre à jour les messages professeur si des dates sont déplacées.
5. Tester le job avec une activité dont `nextGenerate` est juste avant/après "now".

Modifier startYear()

Toute nouvelle initialisation annuelle doit être faite soit dans le socle commun si elle concerne tous les jeux, soit dans `startYearGameSpec()`. Ne pas ajouter des branches GameUid directement dans `utils_by_year.startYear()`.

Tester un changement de cycle

Cas	À vérifier
Année 1 manuelle	start state, presence, year data, RTT.
Fin manuelle	generating → results → finished.
Calendrier automatique	<code>nextGenerate</code> , scheduler, next year.
Async	bots avant génération.
Dernière année	pas de démarrage fantôme d'une année suivante.
Reopen	état + results + evals + logbook restaurés/cohérents.

Debug d'un job qui ne part pas

1. Lire `activity.nextGenerate`.
2. Vérifier que l'activité est active.
3. Vérifier `gatherActsForJob()`.
4. Exécuter `runActJob()` avec l'UID.
5. Inspecter l'état annuel avant/après.
6. Vérifier le log de `generateResultsJob()`.

