

MGT1 — architecture et logique game-specific (DEPRECATED)

Positionnement

MGT1 modélise la gestion d'une équipe de développement : employés, temps de travail, features, segments, budget et lancement d'un jeu.

Static data

Les datasets comprennent `employees`, `events`, `features`, `scenario`, `segments` et `act_perf_pond`, plus `constants`.

Modèles Unity

- `Employee`, `EmployeesData` ;
- `Feature`, `FeatureResults`, `GameFeatureData` ;
- `Segment` ;
- `LaunchedGame`, `GameResults` ;
- `TeamDecisionsMgt1`, `TeamMgt1`, `YearData`.

Logique serveur

`GameSpecificUtilsMgt1` gère notamment :

- activation/inactivation d'employés;

- création et noms d'employés;
- coûts salaire et coûts additionnels;
- temps de travail;
- affectation d'employés aux features;
- travail requis et complétion de feature;
- events et rankings.

Résultats

`GenerateResultsMgt1` calcule le travail requis, les bonus énergie/discipline/bonheur, les employés à inactiver, les résultats par équipe et la transition des employés vers l'année suivante.

```

flowchart LR
    EMP[Employés] --> WORK[Temps de travail]
    FEAT[Features] --> NEED[Travail requis]
    WORK --> PROG[Progression features]
    NEED --> PROG
    PROG --> COMPLETE[Features complétées]
    COMPLETE --> GAME[Résultat du jeu]
    EMP --> BONUS[Energy / discipline / happiness]
    BONUS --> GAME
  
```

UI

L'UI game-specific contient employees list, team leader, producer, president, feature selectors, budgets, work allocation, lancement du jeu et rapports.

Debug

Pour une feature qui progresse mal, vérifier `setWorkOnFeature()`, le temps/coût de l'employé, `getNeededWork()` et les bonus appliqués dans la génération annuelle.

Ajouter une feature MGT1 de A à Z

Custom commands réelles

`mgt1_custom_command.js` gère :

- `launch_game` / `delete_game` ;
- `add_feature` / `delete_feature` ;
- `hire_employees` / `dismiss_employee` ;
- `change_work_time` ;
- `save_social_budget` / `save_remote_budget` ;
- `save_feature_ratios` ;
- `change_employee_active_status` ;
- `employee_level_up` / `employee_level_down` .

Carte du moteur

- `getEmployeeSalaryCost()` / `getEmployeeAdditionalCost()` — coûts;
- `getEmployeeWorkTimeBonus()` / `getEmployeeWorkTimeCost()` — temps;
- `getFeatureMinimumBudget()` ;
- `setWorkOnFeature()` ;
- `completeFeature()` ;
- `getNeededWork()` ;
- `getEnergyBonus()` , `getDisciplinaryBonus()` , `getHappinessBonus()` ;
- `computeTeamsResults()` / `generateCurrentYearResults()` .

Ajouter une nouvelle feature de jeu MGT1

1. Ajouter la static data sous `features` si c'est une feature configurable.
2. Ajouter le modèle Unity `Feature/GameFeatureData` si structure nouvelle.
3. Ajouter le contrôle UI et les ratios/budget si nécessaire.
4. Ajouter la command de création/édition.
5. Vérifier les employés déjà assignés lors d'une suppression : `deleteEmployeeFeature()` doit nettoyer les travaux devenus invalides.
6. Ajouter la contribution au `workNeeded` et à `computeTeamsResults()` .
7. Vérifier la complétion via `completeFeature()` .
8. Ajouter le résultat/report.

Employés : index et noms

Le code associe certains employés par index dans `yearData.employees` et par nom dans `employeesNames`. Toute opération d'ajout/suppression doit préserver l'alignement attendu; une suppression mal gérée peut faire pointer un nom vers le mauvais employé.

Work time : validation serveur obligatoire

`changeWorkTime()` valide game, feature, task level et niveau réel de l'employé. Une nouvelle allocation de travail doit conserver ce type de validation côté serveur.

Debug MGT1

Pour une feature qui ne progresse pas : feature exists → employee works structure → level/workTime → needed work → bonuses → computeTeamsResults → final result.

Revision #3

Created 2026-09-18 19:21:50 UTC by Mathis

Updated 2026-09-18 20:20:12 UTC by Mathis