

Bootstrap et composition runtime

Responsabilité

Le bootstrap commun décide **où le client se connecte, quel jeu est chargé, quelle famille de simulation est utilisée et quelles classes concrètes sont créées**. Le point central est

`Assets/Src/Game/Game.cs`.

```
flowchart TD
    AWAKE[Game.Awake] --> ENV[SetConnectionType]
    AWAKE --> SERVICES[Services globaux]
    ENV --> BC[BrainCloudWrapper]
    LOGIN[Login] --> UID[SetGameUid]
    UID --> SIMTYPE[CurrentSimType]
    UID --> TEXTS[LoadGameSpecificTexts]
    UID --> FACTORY[Factory.CreateStaticDataController]
    FACTORY --> BM[By Module]
    FACTORY --> BY[By Year]
```

Game.Awake()

`Awake()` initialise le singleton `Game.Instance`, `PoolsManager` et les objets backend. En Editor, `SetConnectionType(GameConnectionType.eDev)` est utilisé par défaut. Les wrappers `GameSparks/BrainCloud` sont désactivés avant la sélection du backend puis le wrapper nécessaire est activé/configuré.

Environnements BrainCloud

`SetConnectionType()` sélectionne l'application BrainCloud correspondant à l'environnement courant (Dev, Dev2, Demo ou Live), initialise `BrainCloudWrapper`, active son `GameObject` et configure notamment la profondeur JSON maximale.

Les identifiants/secrets d'environnement sont du code opérationnel sensible : la documentation décrit leur emplacement et le flux, mais ne doit pas recopier les secrets.

InitBackendState → AuthState

```
sequenceDiagram
    participant S as InitBackendState
    participant G as Game
    participant B as Backend
    participant A as AuthState
    S->>G: SetBackend(backEnd)
    S->>G: GlobalTime.GetTime()
    S-->>A: username + pass + gameId
    A->>B: Login(...)
    B-->>A: profileId + createdAt + displayName
    A-->>G: LoadUserState
```

SetGameUid()

Après chargement de `UserDetails`, `Game.SetGameUid(gameUid, skinName)` :

1. convertit le string en `SharedConstants.GameUid.eGameUid`;
2. renseigne `CurrentGameUidEnum` et `CurrentGameUid`;
3. détermine `CurrentSimType` avec `SharedConstants.SimType.GetSimTypeFromGameUid()`;
4. charge les textes game-specific;
5. crée le controller de static data via `Factory.CreateStaticDataController()`;
6. détermine le type d'école et poursuit l'initialisation.

Factory

`Assets/Src/Game/Factory.cs` est le point de composition. Il sélectionne les classes concrètes selon le `GameUid`, au lieu de disperser des `if (gameUid)` dans le projet.

GameUid	Famille	StaticDataController
ent1	By Year	StaticDataControllerEnt1
mk1	By Year	StaticDataControllerMk1
hr1	By Year	StaticDataControllerHr1
mk2	By Year	StaticDataControllerMk2
mgt1	By Year	StaticDataControllerMgt1
acc1	By Module	StaticDataControllerByModule / spécialisation Acc1
acc2	By Module	StaticDataControllerByModule / spécialisation Acc2

Factory : autres objets construits

La Factory ne sert pas seulement à la static data. Les controllers communs appellent aussi des méthodes comme `CreateSimulationBase()`, `CreateActivityBase()`, `CreateTeamBase()`, `CreateTeam()`, `CreateStudentStub()`, `CreateTeamStub()` et `CreateResults()`. Le GameUid/SimType choisit ainsi le modèle concret à tous les niveaux.

LoadUserState

`LoadUserState` charge `UserDetails`, crée `Teacher` ou `Student`, applique la langue, appelle `SetGameUid()`, démarre le heartbeat hors Editor et, en WebGL, tente de mettre à jour dynamiquement textes et écoles.

Version client

Le runtime charge `Assets/Resources/data/version.txt` et `sub_version.txt`. Le workflow de build et l'absence d'incrément automatique sont documentés dans **Build Unity, WebGL et versioning client**.

Ajouter un nouveau GameUid

1. Ajouter le GameUid dans `SharedConstants`.

2. Mapper le GameUid vers le bon `SimType`.
3. Ajouter les classes concrètes nécessaires dans `Factory`.
4. Créer `Assets/GameSpecific/<gameUid>` et ses Resources.
5. Ajouter les static data/textes dans les pipelines Node.
6. Ajouter les Server Scripts game-specific et les factories serveur si la logique diffère.
7. Tester le flow complet : login → SetGameUid → static data → simulation → commande serveur.

Diagnostic

Si le mauvais jeu, la mauvaise UI ou le mauvais modèle est chargé, vérifier dans l'ordre :

`UserDetails.gameUid` → `SetGameUid()` → `CurrentSimType` → `Factory` → `StaticData*.Initialize()`.

Revision #2

Created 2026-09-18 19:08:50 UTC by Mathis

Updated 2026-09-18 19:30:28 UTC by Mathis