

MK2 — architecture et logique game-specific

Positionnement

MK2 reprend le cœur marketing de MK1 mais ajoute notamment la logique pays/segments, relance, stock, loyauté et notoriété. Il doit donc être traité comme un moteur distinct plutôt qu'une simple skin MK1.

Static data

Les datasets comprennent `product_specs`, `comms`, `distrib`, `medias`, `events`, `price_specs`, `scenario`, `segments`, `countries`, `shops` et `act_perf_pond`, plus `constants`.

Modèles Unity

Les classes sont proches de MK1 mais ajoutent notamment `Country`. L'UI ajoute `CountryLaunchControl` et des variantes MK2 du marché et des rapports.

Logique serveur spécifique

- `launchTypeCalculation()`;
- `relaunchSegment()`;
- `launchCountryCalculation()`;
- `analyseRentability()`;
- `redistributeStockAndRefund()`;
- `getLaunchedProductsOfProduct()`;
- création/lancement de produits avec paramètres pays, supports de communication et stock.

Résultats

```
flowchart TD
    P[Products by country/segment] --> AVG[Market averages]
    AVG --> SCORE[Product + Comm + Distrib]
    SCORE --> SALES[Sales]
    SALES --> LOY[Loyalties]
    LOY --> NOT[Notoriety]
    NOT --> RES[Team Results]
    SALES --> STOCK[Stock / redistribution]
    STOCK --> RES
```

`GenerateResultsMk2` ajoute notamment `computeLoyalties()` et persiste la notoriété en plus des ventes et résultats.

Bots

`BotsMk2` génère les décisions marketing async en tenant compte des règles MK2.

Debug

Si un résultat diffère de MK1, vérifier d'abord les branches pays/relaunch/loyalty/stock avant de comparer les formules de score communes en apparence.

Ajouter une feature MK2 sans supposer MK1

MK2 partage des concepts avec MK1, mais son moteur n'est pas un alias. Il possède son propre `GameSpecificUtilsMk2`, son propre générateur de résultats et ses propres modèles Unity.

Fonctions spécifiques MK2 à connaître

- `launchTypeCalculation()`;

- `relaunchSegment()`;
- `launchCountryCalculation()`;
- `analyseRentability()`;
- `createProduct()` / `launchProduct()`;
- `getLaunchedProductsOfProduct()`;
- `redistributeStockAndRefund()`;
- `computeLoyalties()`;
- notoriété persistée via `saveResults()`.

Custom commands

Les noms d'entrée sont similaires à MK1 : `save_product`, `save_launched_product`, `delete product/launch/calc`. Ils appellent toutefois `utilsMk2`, dont les structures incluent notamment `country/stock/relaunch`.

Ajouter un nouveau comportement de lancement

1. Définir si le changement concerne produit, lancement initial, relaunch, country ou stock.
2. Ajouter la donnée dans `Product` / `LaunchedProduct` / `Country` côté Unity.
3. Ajouter la saisie UI, notamment `CountryLaunchControl` si géographique.
4. Ajouter la valeur aux save commands.
5. Adapter `launchTypeCalculation` / `relaunchSegment` / `launchCountryCalculation` selon le cas.
6. Vérifier les refunds/redistribution de stock lors d'une modification/suppression.
7. Ajouter l'effet aux scores/ventes.
8. Vérifier loyalty + notoriety sur l'année suivante.
9. Adapter `BotsMk2`.

Stock MK2

Le stock de lancement MK2 est distinct du système d'inventaire Accounting. Ne pas réutiliser mentalement les règles FIFO/periodic stock de By Module : ici il s'agit de quantités marketing liées aux lancements et à la redistribution/refund.

Debug différence MK1/MK2

Quand MK2 donne un résultat différent d'une intuition basée sur MK1, vérifier d'abord country, relaunch, stock, loyalties et notoriety. Ensuite seulement comparer product/comm/dist score.

Test de régression MK2

- premier lancement sans stock historique;
- relaunch même segment;
- lancement nouveau pays;
- modification avec refund de stock;
- loyalty année suivante;
- notoriety sauvegardée/rechargée;
- bot + génération async.

Revision #2

Created 2026-09-18 19:21:54 UTC by Mathis

Updated 2026-09-18 20:19:26 UTC by Mathis