RomM — Manettes dans EmulatorJS

> *Dernière mise à jour : 2026-09-06*

Cette page décrit deux correctifs appliqués à **EmulatorJS**, le moteur d'émulation que RomM embarque pour jouer **directement dans le navigateur**. Sans eux, certaines manettes sont vues par le navigateur mais déclarées absentes par l'émulateur, et seul le joueur 1 obtient une configuration de boutons.

Résultat obtenu après les deux correctifs :

```
Joueur 1 → détecté + touches configurées automatiquement
Joueur 2 → détecté + touches configurées automatiquement
Joueur 3 → détecté + touches configurées automatiquement
Joueur 4 → détecté + touches configurées automatiquement
```

> ⚠ **Ne pas confondre avec l'Emulator Streaming.** RomM propose deux façons de jouer, et elles n'ont **aucune pièce en commun** :
>
> - **EmulatorJS** — l'émulation tourne dans le navigateur du joueur, les manettes passent par la *Gamepad API* du navigateur. **C'est le sujet de cette page.**
> - **Emulator Streaming** — Dolphin ou Eden tournent en conteneur sur le serveur et diffusent une image ; les manettes passent par Selkies. Voir les pages « Dolphin pour RomM », « Eden pour RomM » et « Accès distant, publication HTTPS et isolation réseau ».
>
> Un problème de manette ne se diagnostique donc pas de la même manière selon le mode utilisé.

> ◫ **Il existe un second patch EmulatorJS sur cette instance**, décrit dans la page **« RomM — Forcer un core EmulatorJS par plateforme »**. Il impose un

> core pour la N64 et l'Arcade, et modifie un **fichier différent** :
>
> | Patch | Fichier modifié | Ce que c'est | Persistance |
> |---|---|---|---|
> | **Manettes** (cette page) | `assets/emulatorjs/data/src/emulator.js` | le **moteur** EmulatorJS | bind mount Unraid |
> | **Cores** (l'autre page) | `assets/EmulatorJS-*.js` | le **bundle frontend de RomM** | script relancé par *Post Arguments* |
>
> Les deux coexistent sans conflit, mais leurs noms se ressemblent assez pour qu'on les confonde en cherchant une panne. Le `debug: true` du §7 ne concerne **que** le patch manettes.

> ◫◫ **Contexte de ces correctifs** : EmulatorJS **4.2.3**, tel qu'embarqué dans **RomM 5.2.0**. Serveur Unraid, image `rommapp/romm`, conteneur `romm`. Ce sont des contournements liés à ces versions — voir §10 avant toute mise à jour.

---

1. Ce que corrigent ces deux patchs

#	Problème	Symptôme visible	Nature du correctif
1	Index de manette « à trous » mal interprété	`Game Pad not connected` alors que le *Controller Debug* de RomM fonctionne	Chercher la manette par son `index` réel au lieu de sa position dans le tableau
2	Contrôles par défaut vides pour les joueurs 2 à 4	Chaque joueur supplémentaire doit refaire son mapping à la main	Copier le mapping **manette** du joueur 1 vers les joueurs 2, 3 et 4

Les deux patchs vivent dans le **même fichier**, `emulator.js`, conservé dans l'`appdata` d'Unraid et monté dans le conteneur (§8).

---

2. Problème n°1 — la manette est vue, mais EmulatorJS la dit absente

2.1 Symptômes

Dans RomM, le diagnostic intégré répond correctement :

```
Controller Debug → OK
```

Tous les boutons et tous les axes réagissent. Mais dès qu'un jeu démarre dans EmulatorJS :

```
Game Pad not connected
```

Et dans la console du navigateur (`F12`) :

```
Uncaught TypeError: Cannot read properties of undefined (reading 'id')
    at Object.connected (emulator.min.js:1:237337)
    at GamepadHandler.dispatchEvent (emulator.min.js:1:206364)
    at emulator.min.js:1:205943
    at Array.forEach (<anonymous>)
    at GamepadHandler.updateGamepadState (emulator.min.js:1:204942)
    at GamepadHandler.loop (emulator.min.js:1:204688)
```

C'est la signature du bug : l'émulateur lit une case de tableau qui n'existe pas.

2.2 Diagnostic — ce que le navigateur expose vraiment

À exécuter dans la console du navigateur, manette branchée et un bouton pressé au moins une fois (la *Gamepad API* n'expose rien avant la première interaction) :

```
Array.from(navigator.getGamepads()).map((g, slot) =>
  g ? {
```

```
    slot,
    index: g.index,
    id: g.id,
    mapping: g.mapping
  } : null
)
```

Résultat observé dans notre installation :

```
0: null

1:
  id: "Controller (STANDARD GAMEPAD)"
  index: 1
  mapping: "standard"
  slot: 1

2: null
3: null
```

La manette est donc **parfaitement fonctionnelle** au niveau du navigateur. Le point qui compte : elle occupe le **slot 1**, avec `index = 1`, alors que le **slot 0 est vide**.

C'est ce qu'on appelle un index *sparse* — un emplacement libre avant une manette. Cela arrive dès qu'une manette a été débranchée, qu'un récepteur occupe un slot, ou simplement selon l'ordre de branchement.

2.3 La cause — un index qui n'est pas une position

EmulatorJS 4.2.3 confond deux choses différentes :

```
index navigateur   ≠   position dans son tableau interne
```

Le navigateur annonce `gamepadIndex = 1`. Mais le tableau interne d'EmulatorJS, lui, ne garde que les manettes réellement présentes :

```
[
    {
        id: "Controller (STANDARD GAMEPAD)",
```

```
        index: 1
    }
]
```

La manette est donc **en position 0** de ce tableau, tout en portant `index = 1`. L'ancien code fait :

```
this.gamepad.gamepads[1].id
```

alors que `this.gamepad.gamepads[1]` vaut `undefined` — d'où l'erreur `Cannot read properties of undefined (reading 'id')`, et la manette déclarée non connectée.

> □□ Tant que la manette occupe le slot 0, tout fonctionne : `index 0` et `position 0` coïncident par hasard. C'est ce qui rend le bug déroutant — il apparaît et disparaît selon l'ordre de branchement.

---

3. Correctif n°1 — chercher la manette par son index réel

Le principe : cesser de traiter `gamepadIndex` comme une position, et **rechercher** la manette dont l'`index` correspond.

Une fonction `getGamepadSelectionValue(gamepadIndex)` est ajoutée, bâtie autour de :

```
const gamepad = this.gamepad.gamepads.find(
    candidate =>
        candidate &&
        candidate.index === gamepadIndex
);
```

Les emplacements du code qui lisaient directement le tableau appellent désormais cette fonction. Le cas problématique se résout alors correctement :

```
slot 0 → vide
slot 1 → manette index 1   ✓ trouvée
```

---

4. Problème n°2 — seul le joueur 1 est configuré

Une fois les manettes détectées, un second comportement apparaît. La première manette joue immédiatement, les autres non :

```
Player 1 → touches déjà configurées
Player 2 → touches à configurer manuellement
Player 3 → touches à configurer manuellement
Player 4 → touches à configurer manuellement
```

Ce n'est pas un bug, mais le comportement d'origine d'EmulatorJS : ses contrôles par défaut ne renseignent que le premier joueur.

```
0: {
    // mapping complet du joueur 1
},
1: {},
2: {},
3: {}
```

---

5. Correctif n°2 — hériter le mapping manette du joueur 1

Les joueurs 2, 3 et 4 reprennent le mapping du joueur 1 :

```
Player 1
   │
   ├── A / B / X / Y
   ├── Start / Select
   ├── D-Pad
   ├── sticks analogiques
   ├── L / R
   └── L2 / R2
```

```
                │
                ├──────► Player 2
                ├──────► Player 3
                └──────► Player 4
```

> ⚠ **On ne copie que `value2`, jamais `value`.** Dans EmulatorJS, `value2` porte le mapping **manette** et `value` le mapping **clavier**. Copier `value` donnerait les mêmes touches clavier à plusieurs joueurs à la fois — une seule frappe piloterait quatre personnages.

Ce correctif suppose que les manettes sont exposées par le navigateur avec `mapping: "standard"` : EmulatorJS peut alors s'appuyer sur les positions normalisées de la *Gamepad API*. C'est le cas des manettes Xbox, des manettes PlayStation correctement reconnues, des 8BitDo et de beaucoup d'autres modèles. Une manette exotique remontée en `mapping: ""` devra toujours être configurée à la main.

---

6. Mise en place

6.1 Le dossier persistant

Sur Unraid :

```
mkdir -p /mnt/user/appdata/romm-ejs-gamepad-patch
```

Ce dossier contiendra trois fichiers :

Fichier	Rôle
`emulator.js`	**le fichier réellement servi à RomM** (les deux correctifs)
`emulator.js.orig`	copie du fichier fourni par l'image Docker, avant toute modification
`emulator.js.bak-multiplayer`	état intermédiaire : correctif n°1 seul

6.2 Retrouver le conteneur RomM

Son nom peut varier selon le template ; on le détecte par son image :

```
ROMM="$(docker ps --format '{{.Names}} {{.Image}}' | awk '$2 ~ /rommapp\/romm/ {print $1; exit}')"
echo "$ROMM"
```

Si la commande ne renvoie rien, le conteneur est arrêté ou son image diffère — vérifier avec `docker ps -a`.

6.3 Sauvegarder le fichier d'origine

```
docker cp \
  "$ROMM:/var/www/html/assets/emulatorjs/data/src/emulator.js" \
  "/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js.orig"
```

6.4 Appliquer le correctif n°1

> ⚠ **Ce correctif a été appliqué par édition directe du fichier**, sans commande d'injection automatique — contrairement au correctif n°2 (§6.6). Il n'existe donc **pas de recette pour le régénérer** : c'est le fichier `emulator.js` de l'`appdata` qui fait office de référence.
>
> **Conséquence pratique : ce dossier doit être sauvegardé.** S'il était perdu, il faudrait refaire l'analyse du §2 et réécrire le patch à la main.

Le fichier de travail est :

```
/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js
```

Vérification de la présence du patch :

```
grep -n "getGamepadSelectionValue" \
  /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js
```

Attendu dans notre installation — une définition et trois appels :

```
1142: const gamepadSelection = this.getGamepadSelectionValue(e.gamepadIndex);
1153: const gamepadSelection = this.getGamepadSelectionValue(e.gamepadIndex);
1194: getGamepadSelectionValue(gamepadIndex) {
3361: const gamepadSelection = this.getGamepadSelectionValue(e.gamepadIndex);
```

Les numéros de ligne sont indicatifs et bougeront à la moindre évolution du fichier ; c'est la **présence** des quatre occurrences qui compte.

6.5 Sauvegarder avant le correctif n°2

```
FILE="/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js"

cp "$FILE" "$FILE.bak-multiplayer"
```

Cette sauvegarde contient le correctif n°1 **seul** : c'est le point de retour si l'auto-mapping multijoueur pose problème (§11.1).

6.6 Injecter le correctif n°2

```
FILE="/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js"

grep -q "AUTO-MAP PLAYERS 2-4" "$FILE" || sed -i '/if (this\.config\.defaultControllers)
this\.defaultControllers = this\.config\.defaultControllers;/a\
\
        // AUTO-MAP PLAYERS 2-4: copy gamepad mappings from player 1\
        for (let player = 1; player < 4; player++) {\
            if (!this.defaultControllers[player] ||
Object.keys(this.defaultControllers[player]).length === 0) {\
                this.defaultControllers[player] = {};\
                for (const [button, mapping] of Object.entries(this.defaultControllers[0] ||
{})) {\
                    if (mapping.value2 !== undefined) {\
                        this.defaultControllers[player][button] = { value2: mapping.value2 };\
                    }\
                }\
            }\
        }\
' "$FILE"
```

Le `grep -q ... ||` en tête rend la commande **rejouable sans risque** : si le marqueur `AUTO-MAP PLAYERS 2-4` est déjà là, le `sed` n'est pas exécuté et le bloc n'est pas injecté une seconde fois.

Vérification :

```
grep -n -A18 -B3 "AUTO-MAP PLAYERS 2-4" \
  /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js
```

On doit lire :

```
if (this.config.defaultControllers)
    this.defaultControllers = this.config.defaultControllers;

// AUTO-MAP PLAYERS 2-4: copy gamepad mappings from player 1
for (let player = 1; player < 4; player++) {
    if (
        !this.defaultControllers[player] ||
        Object.keys(this.defaultControllers[player]).length === 0
    ) {
        this.defaultControllers[player] = {};

        for (const [button, mapping] of Object.entries(
            this.defaultControllers[0] || {}
        )) {
            if (mapping.value2 !== undefined) {
                this.defaultControllers[player][button] = {
                    value2: mapping.value2
                };
            }
        }
    }
}
```

> □□ La boucle part de `player = 1` : l'indice `0` est le joueur 1, déjà configuré. Elle ne touche qu'aux joueurs dont le mapping est **vide** — une configuration personnalisée déjà présente n'est jamais écrasée.

---

7. Faire servir les sources par RomM (`debug: true`)

Par défaut, EmulatorJS charge sa version **minifiée** :

```
emulator.min.js
```

Or nos correctifs portent sur le fichier **source** :

```
data/src/emulator.js
```

Sans réglage supplémentaire, ils ne seraient donc jamais chargés. Il faut demander à RomM de servir les sources, dans son `config.yml` (le même fichier que celui qui porte le bloc `streaming` des pages Dolphin et Eden) :

```
emulatorjs:
  debug: true
```

Si une section `emulatorjs:` existe déjà, n'ajouter que la ligne `debug: true` en dessous.

> ⚠ **Ce réglage a un coût.** `debug: true` fait charger les fichiers EmulatorJS non minifiés : le démarrage d'un jeu est plus lent et le volume téléchargé plus important, surtout sur une connexion distante. C'est le prix à payer pour que les patchs s'appliquent — il n'y a pas d'autre moyen tant qu'on corrige la source plutôt que le fichier minifié.

Redémarrer RomM après modification.

---

8. Rendre le tout persistant (bind mount Unraid)

Un fichier modifié **à l'intérieur** d'un conteneur est perdu à la première mise à jour, recréation ou réinstallation. Le fichier corrigé doit donc venir de l'`appdata`.

Dans le template Docker RomM d'Unraid, ajouter un **Path** :

Champ	Valeur
Host Path	`/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js`
Container Path	`/var/www/html/assets/emulatorjs/data/src/emulator.js`

Champ	Valeur
Access Mode	**Read Only**

```
/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js
                          |
                          | bind mount Unraid (ro)
                          ▼
/var/www/html/assets/emulatorjs/data/src/emulator.js
```

Le fichier de l'`appdata` **remplace** alors celui de l'image, et devient celui réellement servi aux navigateurs.

> C'est un montage de **fichier**, pas de dossier : le chemin hôte doit exister avant le démarrage du conteneur, sinon Docker crée un dossier vide à sa place et RomM ne trouve plus son `emulator.js`.

Après une mise à jour de l'image RomM :

```
nouvelle image RomM
       +
bind mount Unraid
       =
notre emulator.js corrigé
```

Les deux correctifs restent donc actifs après un redémarrage, une recréation du conteneur ou une mise à jour Docker.

---

9. Redémarrer et valider

```
ROMM="$(docker ps --format '{{.Names}} {{.Image}}' | awk '$2 ~ /rommapp\/romm/ {print $1; exit}')"

docker restart "$ROMM"
```

Puis, dans le navigateur, forcer le rechargement du cache — sans quoi l'ancien fichier reste servi :

```
Windows / Linux : Ctrl + Shift + R
macOS           : Cmd + Shift + R
```

Points à vérifier :

- ☐ `Controller Debug` de RomM répond ;
- ☐ EmulatorJS annonce la manette **connectée** (plus de `Game Pad not connected`) ;
- ☐ la console (`F12`) ne montre plus `Cannot read properties of undefined (reading 'id')` ;
- ☐ le joueur 1 joue sans rien configurer ;
- ☐ les joueurs 2, 3 et 4 jouent **également** sans rien configurer ;
- ☐ les touches clavier ne pilotent qu'un seul joueur.

---

10. Après une mise à jour de RomM ou d'EmulatorJS

Ces correctifs sont des contournements **taillés pour EmulatorJS 4.2.3**. Une nouvelle version de RomM peut très bien corriger le bug d'index nativement, réorganiser la gestion des contrôleurs, ou changer la structure de `emulator.js`.

> ⚠ **Le piège, c'est justement la persistance.** Le bind mount a priorité sur l'image : après une mise à jour, RomM continuera de servir **notre ancien fichier** à un runtime plus récent, même si l'image en fournit une version corrigée. Selon l'ampleur des changements, cela va du simple patch devenu inutile au **plantage de l'émulateur**.
>
> Toute mise à jour majeure de RomM demande donc de relire ce point — ce n'est pas optionnel.

Vérification rapide que le fichier persistant contient toujours les deux patchs :

```
# correctif n°1 — détection
grep -n "getGamepadSelectionValue" \
  /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js

# correctif n°2 — multijoueur
```

```
grep -n "AUTO-MAP PLAYERS 2-4" \
  /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js
```

Si les deux commandes renvoient des lignes, le fichier est intact. **Cela ne dit rien de sa compatibilité** avec la nouvelle version de RomM : à tester en lançant un jeu, console ouverte.

Marche à suivre conseillée après une montée de version majeure :

1. récupérer le nouveau `emulator.js` de l'image dans un fichier à part : `docker cp "$ROMM:/var/www/html/assets/emulatorjs/data/src/emulator.js" /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js.new` ;
2. y chercher `getGamepadSelectionValue` ou toute recherche par `index` — si le bug est corrigé en amont, le correctif n°1 n'a plus lieu d'être ;
3. si seul le correctif multijoueur reste nécessaire, repartir du fichier neuf et lui appliquer le `sed` du §6.6.

---

11. Revenir en arrière

11.1 Annuler le seul correctif multijoueur

Pour retrouver le fichier avec le correctif de détection mais sans l'auto-mapping :

```
cp \
  "/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js.bak-multiplayer" \
  "/mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js"

ROMM="$(docker ps --format '{{.Names}} {{.Image}}' | awk '$2 ~ /rommapp\/romm/ {print $1; exit}')"
docker restart "$ROMM"
```

11.2 Retirer complètement les patchs

Le fichier d'origine est conservé dans `emulator.js.orig`, mais **le restaurer ne suffit pas** : il correspond à la version de RomM installée au moment de la copie, pas forcément à celle en service aujourd'hui. La méthode fiable :

1. **supprimer le Path** dans le template Docker Unraid (celui du §8) ;
2. recréer le conteneur RomM — il retrouve alors le `emulator.js` exact de son image ;

3. si le mode debug n'a plus d'utilité, remettre `debug: false` dans `config.yml` ou supprimer la ligne, pour retrouver les fichiers minifiés et un chargement plus rapide.

Le dossier de l'`appdata` peut être conservé : il ne gêne plus une fois le montage retiré, et garde la trace du travail fait.

---

12. Dépannage rapide

Symptôme	Cause probable	Solution
`Game Pad not connected`, alors que `Controller Debug` fonctionne	Correctif n°1 absent, ou fichier minifié servi	§6.4 puis §7
`Cannot read properties of undefined (reading 'id')` dans la console	Index *sparse* non géré — le correctif n°1 n'est pas chargé	Vérifier le grep du §10, puis le bind mount du §8
Les patchs sont bien dans le fichier, mais rien ne change	`debug: true` manquant : RomM sert toujours `emulator.min.js`	§7
Rien ne change après modification du fichier	Cache du navigateur	Rechargement forcé (§9)
Joueurs 2 à 4 toujours vides	Correctif n°2 absent, ou mappings déjà non vides	§6.6 — la boucle ne touche que les mappings vides
Une même touche clavier pilote plusieurs joueurs	`value` a été copié en plus de `value2`	Restaurer `emulator.js.bak-multiplayer` (§11.1) et réinjecter §6.6
Les patchs disparaissent après mise à jour Docker	Le bind mount n'est pas dans le template	§8
RomM ne démarre plus / plus de fichier `emulator.js`	Docker a créé un **dossier** à la place du fichier hôte manquant	Vérifier que le Host Path existe bien comme fichier, puis recréer le conteneur
Une manette exotique reste non configurée	Le navigateur ne l'expose pas en `mapping: "standard"`	Diagnostic du §2.2 ; mapping manuel à faire dans EmulatorJS
Jeu plus lent à démarrer qu'avant	Normal : `debug: true` charge les sources non minifiées	§7

---

13. À compléter

- **Le correctif n°1 n'a pas de recette rejouable** (§6.4). Idéalement, produire un `diff` entre `emulator.js.orig` et `emulator.js` et le conserver ici — il permettrait de réappliquer le patch sur une version future : `diff -u /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js.orig /mnt/user/appdata/romm-ejs-gamepad-patch/emulator.js`

- **Le dossier `romm-ejs-gamepad-patch` est-il couvert par la sauvegarde de l'`appdata` ?** À vérifier : c'est aujourd'hui le seul exemplaire du correctif n°1.
- **Bug remonté en amont ?** Vérifier si le problème d'index *sparse* est déjà signalé au projet EmulatorJS — si oui, suivre le ticket permettrait de savoir quand retirer le patch.
- **Reprendre la méthode du patch cores ?** Celui-ci **repatche** le fichier après chaque recréation du conteneur au lieu de le **remplacer** par bind mount — ce qui lui évite le défaut décrit au §10 (servir un vieux fichier à un runtime récent). La même approche serait transposable ici, au prix d'un script à écrire. Voir « RomM — Forcer un core EmulatorJS par plateforme », §7.

Revision #4
Created 2026-09-06 09:52:39 UTC by thymon
Updated 2026-09-06 12:28:42 UTC by thymon