3D avec le Web Assembly

- Unity Web & WebAssembly

Unity Web & WebAssembly

Unity

Unity est un moteur de jeux vidéo multiplateforme qui fournit aux développeurs un éditeur.

Parmi les plateformes cibles supportées par Unity, le développeur peut choisir WebGL pour compiler une version web de son jeu.

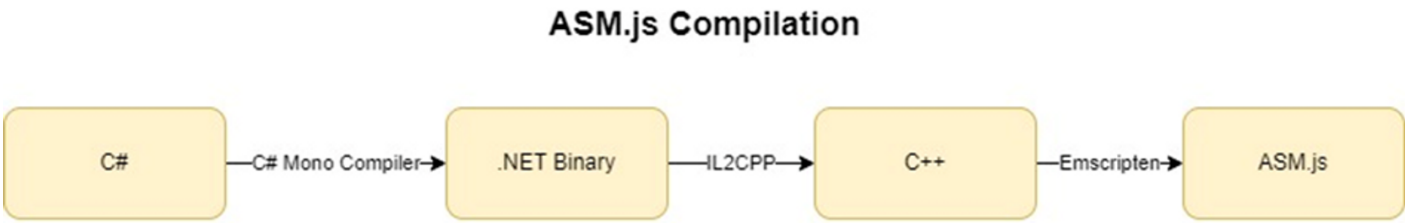
Auparavant, Unity utilisait ASM.js pour transpiler le code du jeu en JavaScript, car c'était le moyen standard d'exécuter des instructions dans un navigateur. Depuis 2018, Unity se base sur le WebAssembly, car cette solution apporte un gain de performance et de nombreux avantages par rapport à ASM.js.

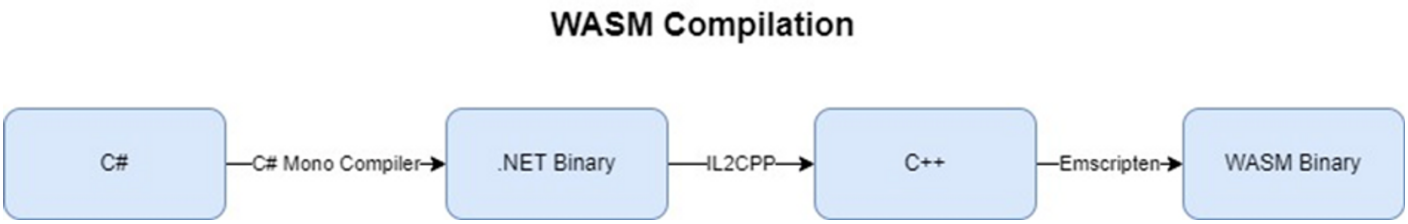
WASM vs ASM.js

Les performances sont similaires entre WASM et ASM.js, dans le cas où le navigateur est optimisé pour ASM.js. Ce qui rend WASM meilleur est son temps de démarrage plus rapide, car le JavaScript doit être parsé et interprété. De plus, la taille du binaire est réduite d'environ 10-15%, ce qui est significatif, car le fichier doit être téléchargé à chaque chargement de page s'il n'est pas mis en cache ou en cas de mise à jour.

Les deux solutions s'appuient sur les mêmes étapes et outils de compilation pour être compilées. Ainsi, le processus de compilation d'un jeu pour la plateforme web n'a pas beaucoup changé.

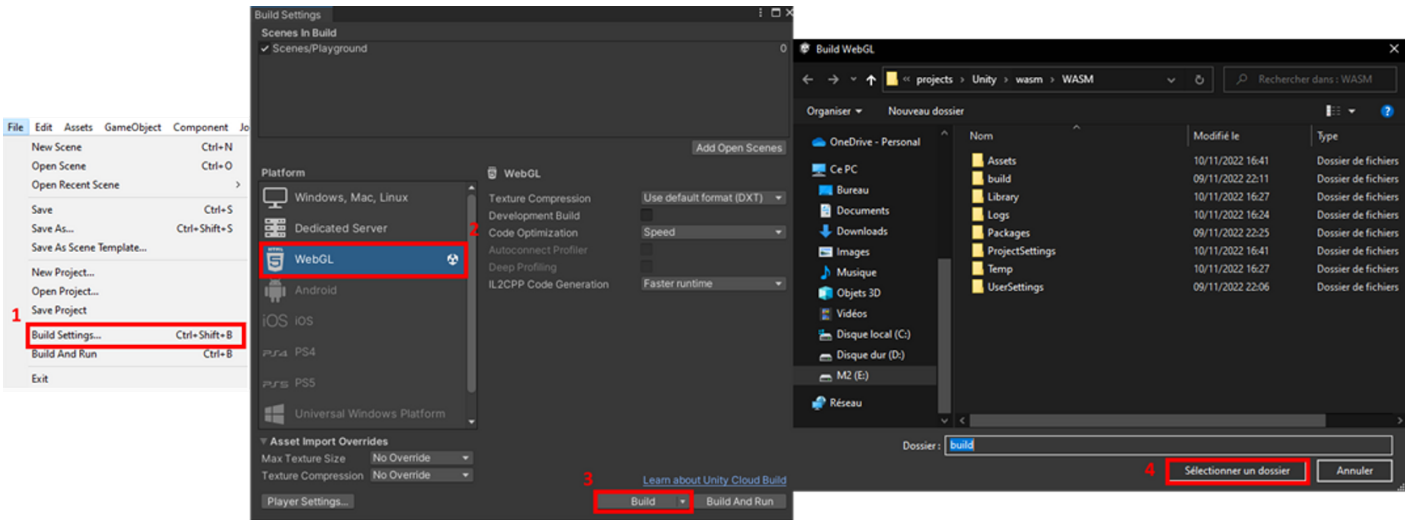
- Le code écrit par le développeur en C# est compilé
- Le binaire est ensuite transpilé à l'aide de IL2CPP en code C++
  - Cela permet ensuite à l'outil Emscripten de compiler le code C++ en un binaire WASM





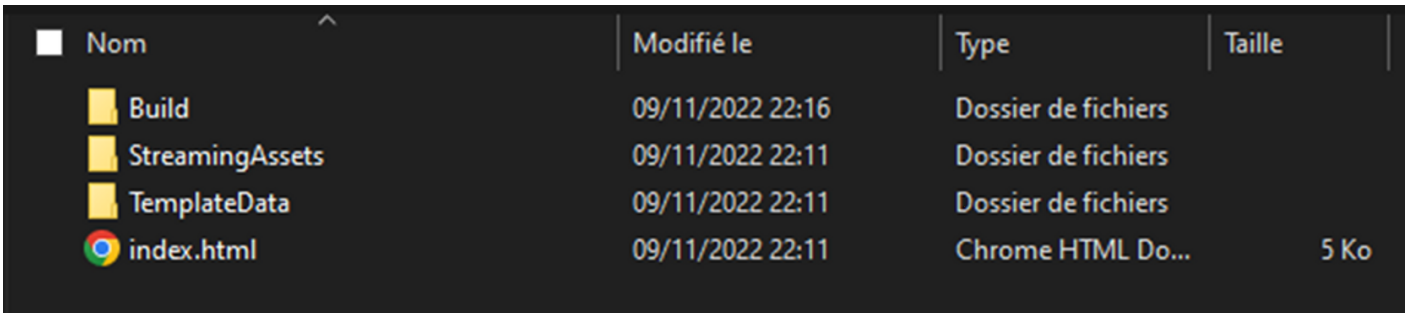
Démonstration

Nous allons partir d'un projet existant, puis définir le web comme plateforme cible et analyser la composition du build avec WASM sur Unity.



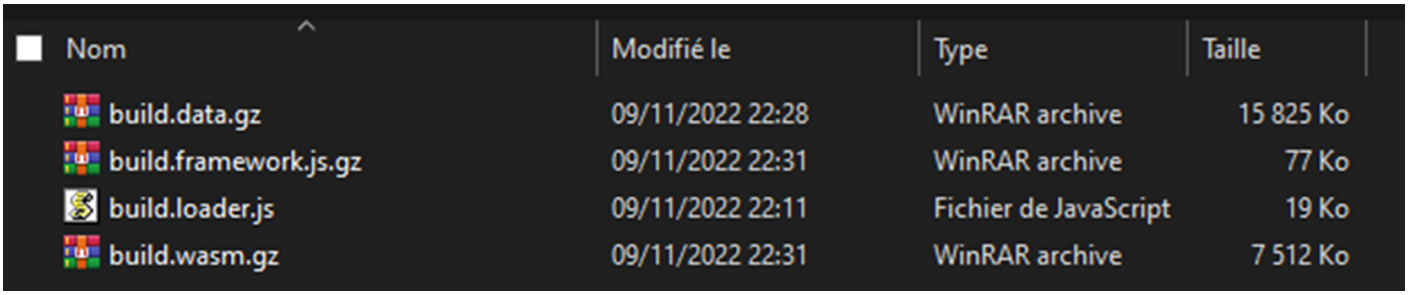
Afin de définir la plateforme cible, nous devons sur notre éditeur Unity :

1. Aller dans le menu file et sélectionner l'option **Build Settings…** qui contient tous les paramètres relatifs à la création d'un livrable de notre jeu
2. Ensuite, nous sélectionnons dans la liste des plateformes l'option **WebGL** pour créer une version WebAssembly de notre jeu.
3. En cliquant sur le bouton **Build**, l'éditeur lancera les différentes étapes de construction.
4. Et pour finir, nous sélectionnons le dossier dans lequel notre **build** sera situé.



Une fois le build effectué, on retrouve les fichiers suivants :

- Le fichier **index.html**, qui sert de wrapper pour afficher notre jeu
- Le dossier **TemplateData** qui contient des assets pour l'affichage de la page HTML
- Le dossier **StreamingAssets** qui contient un fichier JSON avec des métadonnées relatives au build Unity
- Le dossier **Build** qui nous intéresse, il contient tous les fichiers relatifs au jeu qui sera exécuté dans la page HTML.

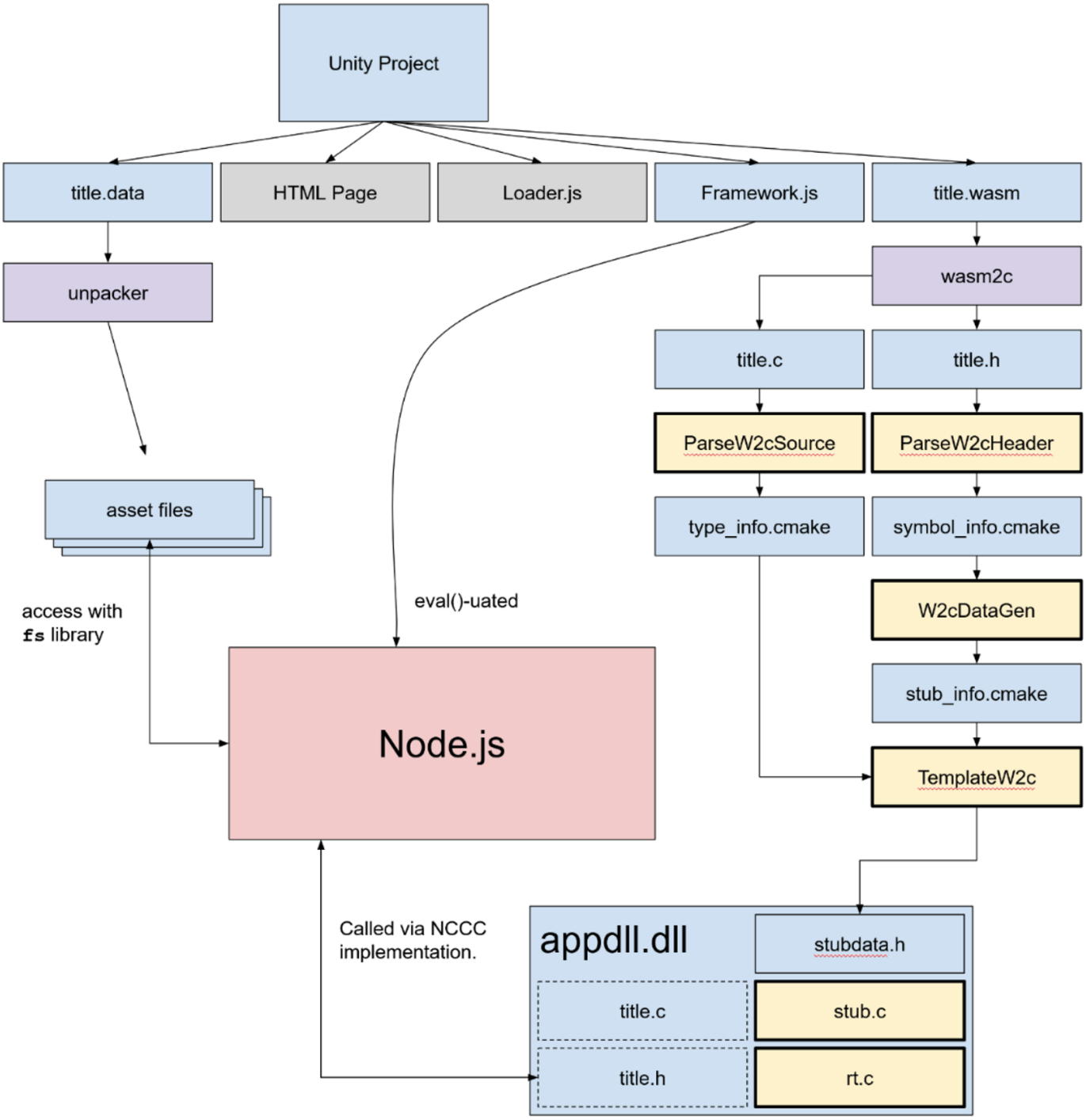


Dans le dossier build, nous retrouvons les fichiers suivants :

- **build.data**, qui contient tous les assets du jeu qui seront mis à disposition du wasm;
- **build.framework.js**, qui comprend tous les bindings permettant au wasm d’interagir avec le navigateur ;
- **build.loader.js**, qui permet de mettre un chargement en attendant que le fichier wasm soit téléchargé ;
- **build.wasm**, qui est le fichier qui contient toute la logique du jeu ainsi que le moteur de jeux de Unity.

Ressources

Organisation d’un projet Unity



https://zenn.dev/okuoku/articles/5a7a04e75234b3

https://medium.com/wasmer/wasmer-io-devices-announcement-6f2a6fe23081

https://blog.unity.com/technology/webassembly-is-here