

ISOMAKER

README



Epitech EIP 2024/2025
Tom Bariteau-Peter, Léa Guillemard, Alessandro Tosi

IsoMaker est un game engine dédié à la création de jeux isométriques. Il est écrit en C++ et utilise la Raylib pour l'affichage.

NORME C++

La fondation de la norme de programmation appliquée à ce projet est une adaptation au C++ de la norme de programmation en C d'Epitech. En voici les principes fondamentaux :

CONVENTIONS D'APPELLATION

- Tous les **sous dossiers** et **fichiers** contenus dans les dossiers /src et /includes devraient être composés d'un seul mot si possible. Ceux ci, ainsi que les noms des **classes** devraient suivre la convention UpperCamelCase, selon laquelle chaque mot composant le nom doit commencer par une majuscule.
- Les **interfaces** et **classes abstraites** doivent avoir des noms commençant respectivement par un I majuscule ou un A majuscule (ex. IHandler, AHandler).
- Les **namespaces**, **fonctions** et **variables** devraient suivre la convention lowerCamelCase, selon laquelle le premier mot composant le nom commence par une minuscule et les suivants par une majuscule (ex. input::KeyboardHandler::startLoop(bool status)).

ARBORESCENCE DU DOSSIER PROJET

- Les fichiers .hpp contenant des **interfaces**, **implémentations de classes abstraites**, **types spécifiques au projet** et autres choses similaires devraient se trouver dans /includes/<sous dossier> (ex. IHandler.hpp, AHandler.hpp et Types.hpp dans /includes/Input).
- Les fichiers source .cpp nécessaires à la compilation du projet devraient se trouver dans /src/<sous dossier>, accompagnés de leurs fichiers .hpp respectifs (ex. Keyboard.cpp et Keyboard.hpp dans /src/Input).

FICHIERS

- Tous les fichiers devraient commencer par le header Epitech standard, obtenu grâce à l'extension VSCode dédiée.
- Le contenu des fichiers .hpp devrait être précédé de #pragma once.
- Tous les fichiers devraient se finir par \n (retour à la ligne).

CLASSES

- Les **spécificateurs d'accès** d'une classe devraient être dans l'ordre suivant: public, protected, private
- Les **prototypes de fonction** devraient être déclarés avant les **variables**.
- Les **noms des variables** de la classe devraient commencer par un tiret bas (ex. _position, _size, _scale)

STANDARD DE MESSAGES DE COMMIT

Tous les commits contribuant à ce projet doivent être accompagnés d'un message de commit adhérent au format suivant, basé sur les standards de Commits Conventionnel:

<type>(<portée>): <description>

STRUCTURE

- **type:** Décrit l'objectif du commit. Un des types prédéfinis suivants devrait être utilisé:
 - **feat:** Une nouvelle fonctionnalité.
 - **fix:** Une réparation de bug.
 - **refacto:** Changements dans le code qui ne résolvent pas de bug et n'ajoutent pas de fonctionnalité.
 - **doc:** Changements dans la documentation (ex. modifications du README).
 - **style:** Changements dans la norme du code (ex. format, caractères manquants) qui n'affectent pas de fonctionnalités.
 - **test:** Ajout ou modification de tests.
 - **build:** Maintenance (ex. scripts de compilation, mise à jour des dépendances).
- **portée:** Identifie la partie du code affectée. Devrait rester concis, par exemple:
 - Le nom d'un module (ex. graphics, input handling).
 - Le nom d'un composant (ex. button, keyboard handler).
 - * pour un changement global, impactant de multiples parts.
- **description:** Un résumé bref des changements, à l'impératif.
 - Devrait commencer par un verbe à l'impératif présent.
 - Ne pas commencer par une majuscule ou finir par un point.

Exemples

Ajouter une fonctionnalité: feat(input handling): add support for joypad controls

Réparer un bug: fix(graphics): resolve crash on object rendering

Mettre à jour la documentation: doc(readme): improve commit convention examples

Mettre à jour du code: refacto(engine): simplify event loop logic

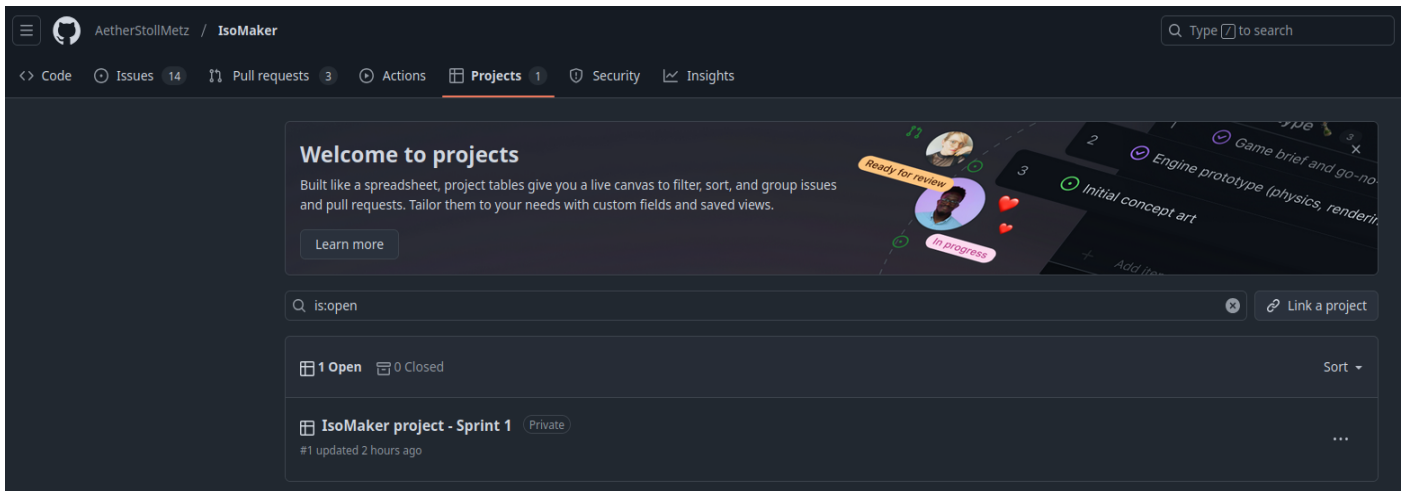
Mettre à jour des tests: test(graphics): add tests for sprite rendering\

POURQUOI UTILISER CE FORMAT ?

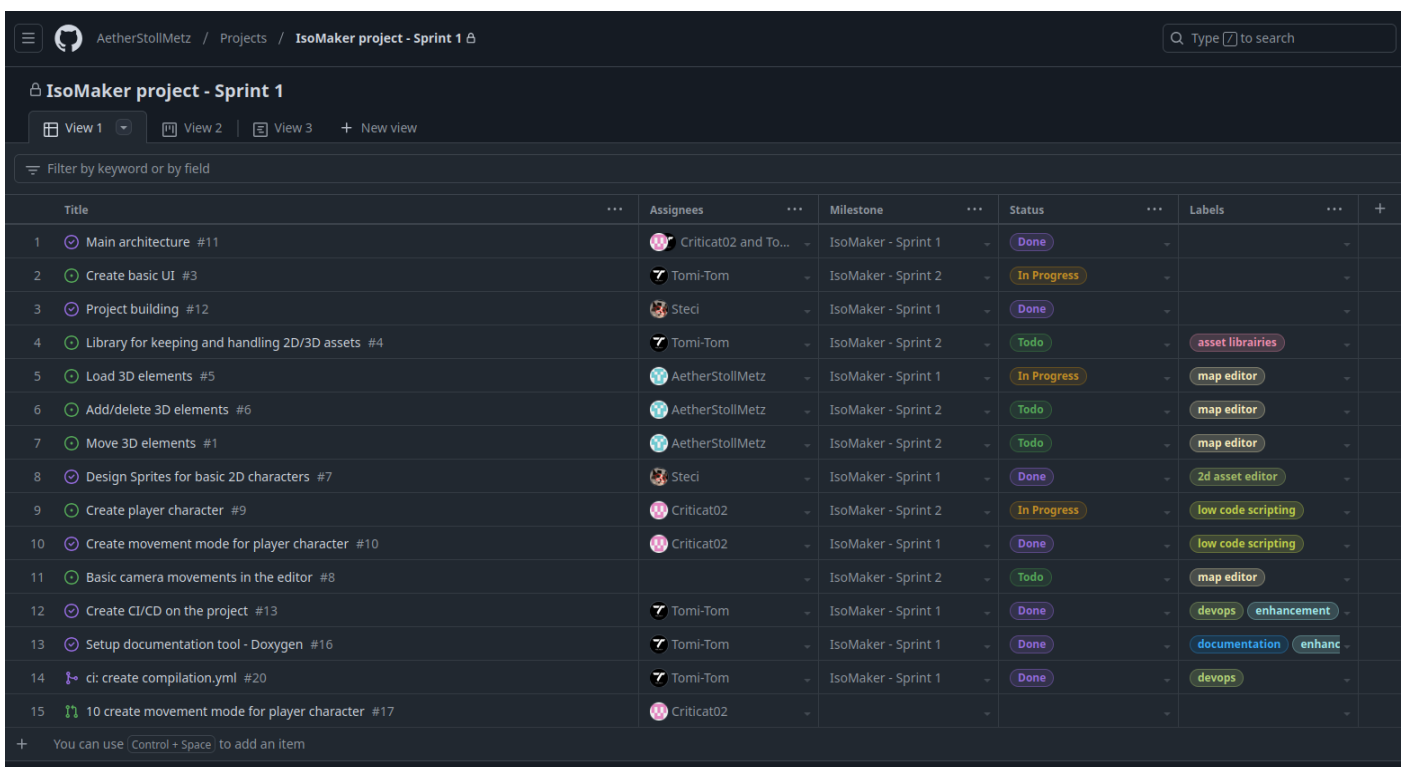
- **Clarté:** Rend la compréhension du but d'un commit plus facile et rapide
- **Automatisation:** Permet d'utiliser des outils pour générer des notes de version et des journaux de modifications.
- **Uniformité:** Encourage les contributeurs à documenter les changements de façon uniforme

COMMENT CREER UNE FONCTIONNALITE ET UNE PULL REQUEST

1. Ouvrir le Github project du repertoire



2. Ajouter un item à la liste



3. Ajouter un nom et une description

Create new issue in AetherStollMetz/IsoMaker

Blank issue in AetherStollMetz/IsoMaker

Add a title *

[New feature]

Add a description

WritePreview

H B I | | < > | | | | | | @ | | |

Type your description here...

Paste, drop, or click to add files

AssigneeLabelIsoMaker project - Sprint 1Milestone

☐ Create moreCancelCreate

4. Cliquer sur le petit bouton “Create a branch”/”Créer une branche”

[New feature] #31

Edit | | | |

OpenAetherStollMetz/IsoMakerPublic

Criticat02 opened now

No description provided.

Add a comment

WritePreview

H B I | | < > | | | | | | @ | | |

Use Markdown to format your comment

Paste, drop, or click to add files

Assignees

No one - Assign yourself

Labels

No labels

Projects

IsoMaker project - Sprint 1

Status No status

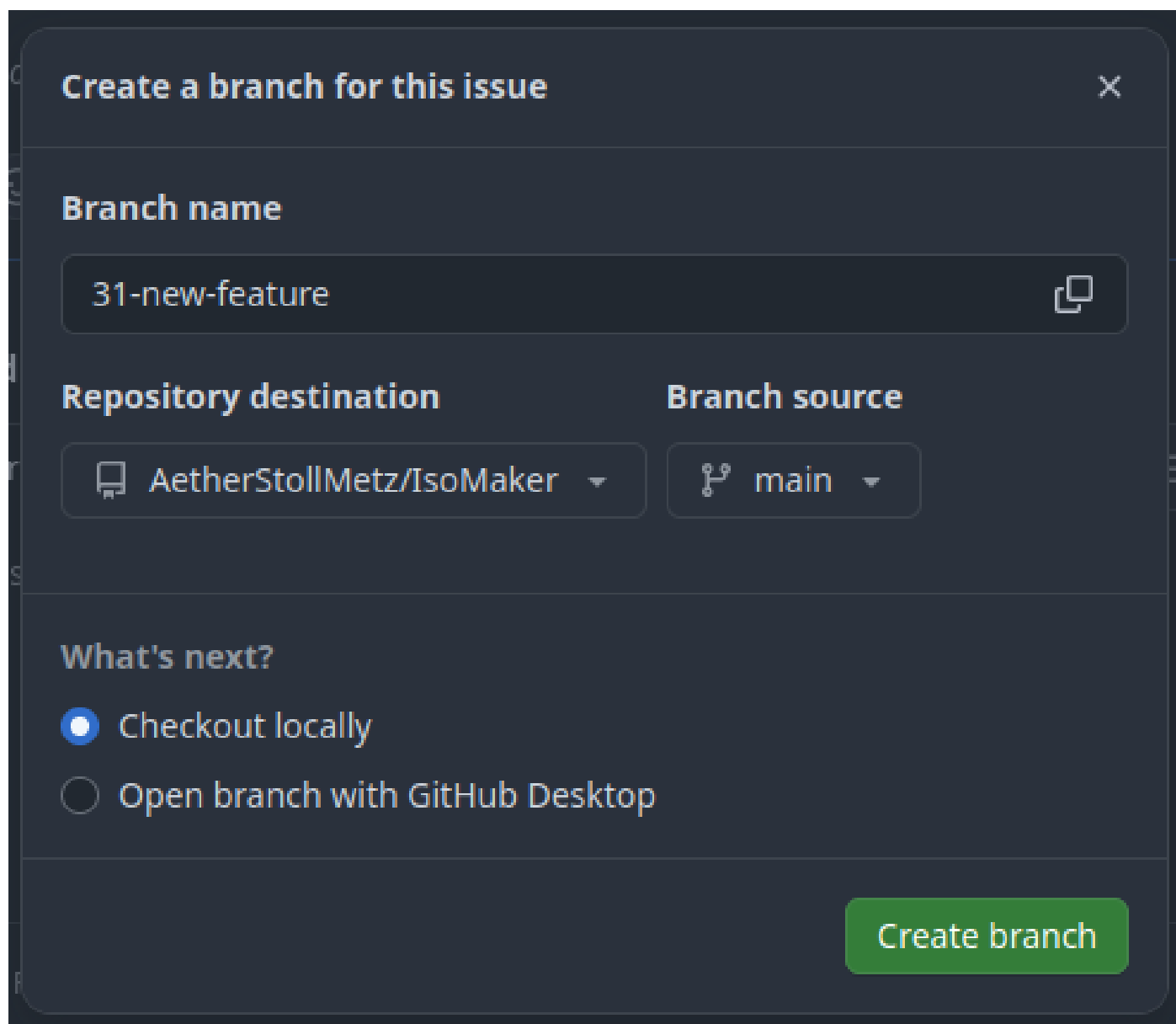
Milestone

No milestone

Development

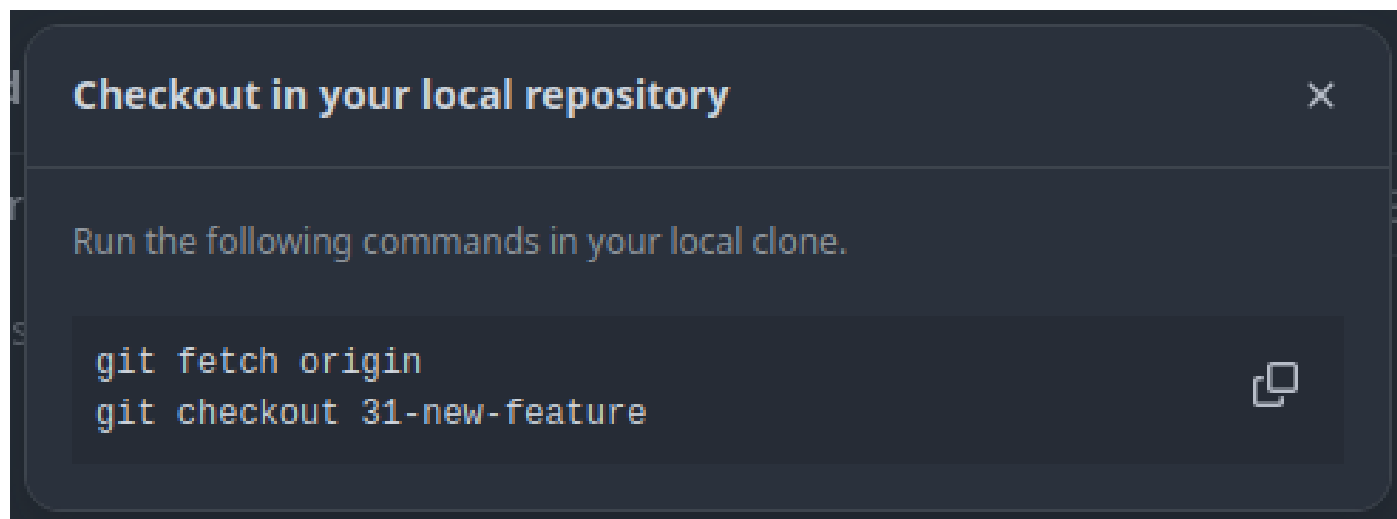
Create a branch for this issue or link a pull request.

5. Vous voudrez peut être changer la branche source pour autre chose que le main. Sinon, laissez tout tel quel et créez la branche.



The screenshot shows a dark-themed dialog box titled "Create a branch for this issue" with a close button (X) in the top right corner. The dialog is divided into several sections. The first section, "Branch name", contains a text input field with the value "31-new-feature" and a copy icon to its right. The second section is split into two parts: "Repository destination" with a dropdown menu showing "AetherStollMetz/IsoMaker" and a "Branch source" dropdown menu showing "main". The third section, "What's next?", contains two radio button options: "Checkout locally" (which is selected) and "Open branch with GitHub Desktop". At the bottom right of the dialog is a green button labeled "Create branch".

6. Faites un check out local de la branche et créez votre fonctionnalité (faites attention à utiliser le standard de messages de commit)



The screenshot shows a dark-themed dialog box titled "Checkout in your local repository" with a close button (X) in the top right corner. The dialog contains a text instruction: "Run the following commands in your local clone." Below this instruction is a code block containing two lines of text: "git fetch origin" and "git checkout 31-new-feature". A copy icon is located to the right of the code block.