Roadmap PROJET SIMILI :

Principes de base du projet :

* On a un monde 3D qui est simulé
* Et un moteur 3D qui permet de créer le monde et de jouer dedans

Etape 1 : Les bases du logiciel 3D :

* Avoir une interface Utilisateur qui peut accueillir une Scène 3D : ✅
* Avoir un grillage et une caméra qui montre cette scène 3D : ✅
* Positionner des objets 3D dans une scène 3D : ✅
* Sélectionner des objets 3D et indiquer leur position dans l’interface : ✅
* Donner à un Gizmo la bonne position correspondant à un objet 3D sélectionné : ✅
* Dessiner le Gizmo : ✅
* Dessiner le Gizmo au centre de l’objet sélectionné : ✅
* HightLight les axes du gizmo lorsque la souris passe au-dessus : ✅
* Cliquer sur les axes du Gizmo : ✅
* Faire en sorte que cliquer sur le Gizmo ne dessélectionne pas l’objet 3D : ✅
* Le Gizmo est toujours visible par-dessus l’objet sélectionné : ✅
* En maintenant le clic enfoncé sur l’axe du Gizmo, on déplace l’objet sélectionné dans l’axe choisi (application de transforms) : ✅
* Pouvoir de nouveau sélectionner l’objet modifié : ✅
* Faire des options de scale et de rotation de l’objet 3D : ✅
* Faire de la Caméra un objet à part dans la scène

avec des fichiers HPP et CPP : ✅

* Faire le rendu de la scène 3D à travers la Caméra : ✅
* On peut sélectionner des objets depuis la caméra : ✅
* Définir des objets sélectionnables ou non : ✅
* Faire une classe Objet 3D de base dont vont hériter les Mesh et la Caméra : ✅
* Pouvoir déplacer l’axe de la caméra (avec bouton central de la souris et touche maj + bouton central pour déplacement latéral) : ✅
* Fonction de Zoom in et Zoom out : ✅
* La caméra peut orbiter autour d’un axe : ✅
* On préserve l’axe de la caméra lorsqu’on la déplace : ✅
* Faire un inspecteur, une fenêtre imgui à part qui va lister les objets 3D, qui devra montrer notre cube et la Caméra par exemple : sous une forme hiérarchique : ✅
* Lorsqu’on sélectionne un objet 3D : on doit voir l’objet 3D sélectionné dans l’inspecteur : ✅
* On peut sélectionner un objet 3D via l’inspecteur, faisant apparaître son gizmo sur l’affichage : ✅
* Docker / fixer les fenêtres : ✅
* Implémenter les bases de directX12 et un début de stack bas niveau : ✅
* Lancer une fenêtre directX12 : ✅
* La fenêtre de droite doit montrer les datas des objets sélectionnés que ce soit depuis l’inspecteur ou l’espace 3D (on se contentera pour le moment du nom et des coordonnées dans l’espace 3D)  : ✅
* Création d’un système permettant de sauvegarder et de charger un GUI et de la maintenir de façon permanente avec un auto-save : ✅
* On veut modifier le nom des objets via l’inspecteur et leurs datas manuellement : ✅
* Menu contextuel qu’on fait apparaître avec le clic gauche et disparaître ensuite avec le clic droit : ✅
* Créer des objets 3D simple via une fenêtre contextuelle : ✅
* Détruire les objets 3D via une fenêtre contextuelle qui apparaît lorsqu’on clique sur souris gauche, sur un objet sélectionné : ✅
* Gérer la sélection multiple : ✅
* Faire un premier système de parent enfant pour les objets 3D : ✅
* L’inspecteur doit pouvoir gérer un système parent-enfant via drag and drop :
* Ce qu’on veut c’est qu’en maintenant l’objet avec un clic gauche enfoncé, celui-ci doit devenir un enfant d’un objet précédant et adopter ses propriétés quand on le lâche : ✅
* Terminer le refactoring de la hierarchy List : ✅
* Faire en sorte que l’enfant reproduise les propriétés du parent(tester avec de simples transforms) : ✅
* Développer le système coordonnées locales et globales : ✅
* Corriger un bug faisant que les déplacements de l’objet sont éclatés si c’est un enfant : ✅
* Si on détruit un objet 3D parent, les enfants sont également détruits : ✅
* Dessin des vertices : ✅
* Dessin des edges : ✅
* Dessin des faces : ✅
* Sélection de vertices : ✅
* Sélection de Edges : ✅
* Sélection de faces : ✅
* Manipulation de vertices : ✅
* Manipulation de edges : ✅
* Manipulations de faces : ✅
* Fonction : Insert edge loops avec CTRL + E : ❌
* Extrusion de faces : ✅
* Créer un historique permettant de remonter

les transformations d’un objet : ✅

* Merge de vertice : ❌
* Fusion de mesh : ❌
* Selection loop (selected element + arrow key) pour :
  + Faces : ❌
  + Vertices : ❌
  + Edges : ❌
* Faire un DNA\_MESH inspiré de Blender qui permettra de suivre les modifications sur les mesh dans une fenêtre d’historique dédié : ✅
* Annuler les modifications du mesh en cliqant sur les éléments de l’historique : ✅
* Faire un scene\_DNA qui va lire les DNA de chaque élément de la scène : ❌
* Faire un CTRL+Z qui annule les modifications en remontant dans l’historique : ❌
* Enregistrer et annuler le parenting : ❌
* Implémenter le reste des modifications qu’on fait au Mesh dans l’historique : ❌
* Création d’un classe « Mesh » qui centralise la création de maillage : ✅
* Exporter et importer un objet 3D (et pouvoir lire le mesh créé dans un autre logiciel) : ❌
* Explorer un début de système pour faire un suivi de projet avec un project manager, qui sera affiché via du XML et du CSS : ❌
* Sauvergarder le projet dans un dossier. On accèdera au projet via un format spécifique « .SIM » : ❌
* Faire un fichier .Scn qui va contenir les scènes : ❌
* Charger la scène 3D sauvegardée : ❌
* Sauvegarder l’UI et la charger manuellement : ✅
* Sauvegarder et charger l’UI de manière automatique : ✅
* Activer le support mutli-écran : ✅
* Caméra non orthographique : ❌
* Changement de Caméra : ❌

Etpae de développement du moteur 2 :  Les bases d’un bon logiciel pixel art / de dessin 2D / Modélisation 3D avancée :

On va faire un système de dessin pixelart qui sera interne au moteur, et qui pourra être posé sur des planes 3D qui pourront servir de tuiles 3D

* Faire un système de suivi de projet : ❌
* Faire une deuxième GUI qui pasculera vers le mode Pixel Art : ❌
* Créer les premiers mods de dessin (on s’inspirera d’Aseprite) : ❌
* Appliquer les dessins à des tiles : ❌
* Créer des tiles par snapping, ils pourront être à plat, à 90° ou à 45° : ❌
* Faire un système d’animation 2D : ❌
* Faire un mode preview du sprite : ❌
* Faire un système de déplacement avec sprite animés : ❌
* Transformer ça en composant : ❌

Etape de développement du moteur 3 : Modélisation avancée, texturing avancé, importation, lighting et intégration de Bullet, animation 2D et 3D

* On va faire trois modes d’affichage :
* Shading, qui représentera l’objet 3D avec un Shader et des textures : ❌
* Vertices, qui représentera les sommets de l’objet 3D sous forme de points noirs : ❌
* Edges qui représentera l’objet sous forme d’arrêtes (on veut pouvoir modifier l’affichage de leur épaisseur librement) : ❌
* On veut pouvoir sélectionner les sommets individuellement et modifier leur position avec le gizmo : ❌
* On applique le principe au Edge (dont on voudrait aussi faire une rotation : ❌
* On veut sélectionner les faces individuelles de l’objet 3D : ❌
* Systèmes d’UV et de normales à un objet 3D qui pourra avoir des textures : ❌
* Début de système de lighting : ❌
* Importer et lire des animations 3D : ❌

Etape de développement du jeu 1  : Commencer à rendre le monde vivant

* On va virer la grille, et faire un système de maillage avec des cubes 3D / voxel
* Un voxel dans notre cas ne sera pas juste un voxel. Ce sera un contenant. Il pourra donc stocker une certaine quantité d’éléments qui exerceront une action sur d’autres éléments qui leur seront superposés. Ce sera un layer de base.

Optionnels :

* Faire son propre système de gizmo

Elements à refactorer et choses à faire :

* Un UML
* Refaire certaines fonctionnalités du hierarchy inspector

Journal de Développement du projet

21 avril 2025 :

Déjà nous avons la preuve que Qt peut tenir de l’OpenGL sans aucun problème. Donc l’idéal maintenant est de représenter un sprite 2D dans un espace 3D.

* Le sprite serait un simple quadrilatère avec une texture
* On va commencer par afficher un simple quadrillage pour représenter le sol, et les coordonnées 0 / le centre de l’univers
* On aura une caméra avec plusieurs types de vue, dont une vue 2D non orthographique et sans perspective top down
* Une unité de base représentera 1 mètre.
* Il faut afficher la texture via SLD2 donc binder Qt OpenGL et SDL2

22 Avril 2025 :

Maintenant on peut afficher des textures via SDL2 dans un contexte openGL. Donc on peut utiliser à la fois, QT, OpenGL et SDL2.

26 Avril 2025 :

On a choisi de virer Qt du projet de passer à imgui, car Qt ne dispose pas vraiment de sélecteur 3D ou de Gizmos, puis les performances peuvent laisser à désirer. Donc nous avons fait une refonte complète du projet et fait une vaste migration. On a d’abord réussi à implémenter ImGUI. Mais on avait du mal à recréer le contexte openGL, donc on a implémenté glad, et refait les fichiers cpp pour tenir le contexte openGL avec un VAO, et un VBO notamment.

27 Avril 2025 :

On va tenter une expérience. Techniquement dans notre projet on a une fenêtre Imgui qui tient la scène 3D. Donc elle existe dans notre code.

Ce qu’on veut c’est lui ajouter clairement la scène 3D via une flèche.

Exemple de pseudo code

Fenêtre qui tient la scène 3D => Ajouter Scène 3D => Ajouter grillage => Ajouter composant 2D

Note : On a pas pu faire de fonction fléchée, mais on a créé un fonction add, parameter de gauche et parameter de droite. La fonction add permet à un objet, le paramètre de gauche de se voir ajouter un autre objet, le paramètre de droite.

28 Avril 2025 :

J’ai tenté de séparer le rendu d’objet 3D, de la Scène 3D et l’instanciation du Contexte OpenGL. Au final il en a résolu un bug que j’ai été incapable de résoudre.

Problèmes rencontrés :

La caméra n’était jamais dans le bon sens, on aurait dit qu’elle était override. Donc j’ai galéré pendant trois heures pour la paramétrer correctement. Ensuite l’image 2D était rendue de manière relative à la caméra, et pas en tant qu’objet 3D. Ce qui n’était pas bon, car si l’objet 2D était rendu correctement,

Donc j’ai opté pour la décision suivante :

On se contente d’ajouter à une fenêtre ImGUI enfant un contexte OpenGL dont la taille est attachée à la taille de la fenêtre imGui, et on a jouté des fonctions get return pour retourner la hauteur et la largeur du contexte OpenGL. Le contexte OpenGL seul sera ajouté à la fenêtre.



29 Avril 2025 :

* On a réussi à ajouter un cube 3D dans la scène.
* Tentative de création d’un Gizmo et d’un sélecteur.
* Question : Est-ce que ce n’est pas la fenêtre 3D qui devrait tenir la logique de élection d’objet vu qu’elle tient le contexte openGL ?
* Dans ce cas, autant créer un ThreeDWindow.cpp ce qu’on a fait, maintenant la fenêtre 3D spécialisée peut détecter les clics via glfw

A faire : Associer le sélecteur et la fenêtre 3D

* Si on y arrive, il faudra ajouter la propriété nom aux cubes, et faire un listing imgui pour afficher le nom du cube, ses coordonnées dans l’espace, etc.

Idée pour plus tard :

* L’idéal serait de commencer à plancher sur un début de système de sauvegarde scène. On pourrait stocker le monde 3D, les objets, leurs positions, via un JSON

01er Mai 2025 :

On a un maintenant un sélecteur qui est capable de sélectionner un objet à sa bonne position dans un espace 3D.

Note : J’ai également commencé à bosser sur une roadmap. On va d’abord développer des fonctionnalités de moteur 3D.

05 mai 2025 :

J’ai fait une petite pause de deux jours, sur le projet, mais maintenant on a un gizmo avec des flèches qui donnent un feedback visuel lorsque la souris passe au-dessus d’elle. On peut également, et enfin, déplacer les objets dans l’espace 3D. Par contre on ne peut plus les sélectionner une fois déplacés.

J’ai corrigé le problème qui empêchait la sélection de l’objet une fois qu’il était déplacé.

Penser à implémenter un Ribbon élément plus tard.



Note : sur ImGui on parle de TabBar

11 mai 2025 :

J’ai terminé le développement de la caméra 3D. La caméra 3D est un objet 3D dans la scène 3D. Elle peut se déplacer latéralement, zoomer, dézoomer, orbiter. Le rendu 3D est également fait à travers elle.

12 mai 2025 :

J’ai amélioré la sélection de l’objet 3D. Maintenant ça tient vraiment compte de la position de la caméra et ça détecte beaucoup mieux la position de l’objet. On tient compte de la taille et de la largeur de la fenêtre ImGUI et plus vraiment de celle du contexte openGL. La déselection et sélection fonctionnent également mieux.

14 mai 2025 :

Il est temps de commencer à plancher sur ce début d’inspecteur et de hiérarchy window.

15 mai 2025 :

J’ai un inspecteur hiérarchique qui permet de voir quels sont mes objets dans ma scène 3D

17 mai 2025 :

Après une bonne journée dans les transports en commun, j’ai pu trouver du temps pour bosser l’inspecteur hiérarchique. J’ai mis en place des fonctions pour créer un système de surbrillance, lorsque la souris arrive à un item de la liste. Lorsqu’on fait un clic gauche sur l’item, il passe du jaune au rouge.

L’objet sélectionné va sélectionner l’objet correspondant dans la scène 3D avec un gizmo, qui appliquera les transforms souhaités sur l’objet. On peut déselectionner l’objet depuis l’inspecteur ce qui fait disparaître le gizmo.

Le reste de l’après-midi, j’ai tenté de comprendre comment à partir de la scène 3D, je pouvais indiquer dans l’inspecteur hiérarchique l’objet actuellement sélectionné. J’ai réussi en faisant en sorte que la classe hiérarchy inspecteur soit membre de la classe ThreeDWindow et en faisant de ThreeDWindow une classe membre de l’inspecteur. La déselection fonctionne aussi.

22 mai 2025 :

J’ai dû lâcher le projet deux trois jours par flemme, et aussi parce que ben, je dois trouver un travail IRL (mais je me fais aucun soucis de ce côté-là). Mais aujourd’hui j’ai pu reprendre le projet. J’ai activé le mode Docking d’ImGui.

Ce qui nous donne ce rendu :

J’ai commencé à faire un système qui permettra la lecture et la sauvegarde d’un fichier de config.



J’ai commencé à implémenter les bases de DirectX 12 dans le programme et à le faire fonctionner avec OpenGL (DirectX 12 étant notamment très utilisé dans l’industrie). Le but final est de proposer un moteur proposant plusieurs API graphiques. OpenGL pour les projets simple, DirectX 12 pour les projets plus complexes. Comme ça les gens qui utilisent l’un ou l’autre pourront coder leur shaders sur Simili.

26 mai 2025 :

C’est un bon mois de développement qui s’achève mais on a une bonne base du projet. Le projet peut initier une fenêtre openGL dans une fenêtre ImGUI, et on peut lancer une fenêtre DirectX 12 dans une fenêtre indépendante. On est même capables de détecter le hardware utilisé par le PC.

En théorie, il y aura juste besoin de continuer le développement de la stack bas niveau de DirectX 12 quand on décidera de faire des projets qui tourneront sur DirectX 12 plutôt que sur OpenGL.

Screnshot de mon projet de moteur 3D. On peut voir 5 fenêtres :

- Le hierarchy insepctor qui liste les objets présents dans la scène 3D
- La fenêtre qui contient la scène 3D, soit un cube bleu sur un grillage
- Une fênêtre en bas qui contient un simple " Bienvenue dans la première fenêtre ! c'est une belle fenêtre ! "
- Une fenêtre bleue indépendante qui tient le contexte de rendu pour DirectX 12
- Une fenêtre de debug

Ceci fait, j’ai commencé à implémenté l’inspecteur d’objet.



29 mai 2025 :

L’inspecteur d’objet est opérationnel, il montre les propriétés de l’objet lorsqu’on clique sur l’un des items de la liste de l’inspecteur hiérarchique ou sur l’objet 3D l’espace 3D. J’ai aussi créé un système de sauvegarde, chargement d’UI, et d’UI permanente grâce à un système d’autosave.

31 mai 2025 :

J’ai corrigé le code du fichier MainSoftwareGUI.CPP pour le rendre plus rationnel. J’ai créé des fonctions comme tryLoadLayout, autoSaveLayout, mainWindowOptions, popUpModal, pour rationnaliser au maximum le code, séparer les responsabilités, et rendre le code plus lisible. J’ai également activé le multiScreenSupport, cela a necessité bien sûr de corriger un peu la boucle de rendu, car ImGUi peut imporairement changer le contexte openGL quand il dessine ses platform windows (les fenêtre flottantes détachables). Il peut donc swapper le mauvais buffer, ce qui provoque des artefacts visuels. On doit donc «  forcer » la gestion du contexte OpenGL.

01 juin 2025 :

* J’ai réussi à faire en sorte qu’on puisse modifier les propriétés des objets, comme le scale, la rotation, la position, le nom, etc. depuis l’éditeur.
* J’ai fait créé un premier système de pop apparaît lorsqu’on fait un clic gauche, ainsi qu’un début de création d’objet via cette même fenêtre
* Le système de création depuis le menu contextuel est mainte opérationnel. J’ai corrigé un bug qui faisait que lorsqu’on sélectionnait un objet créé dan l’inspecteur hiérarchique, la fenêtre d’inspection d’objet continuait d’afficher qu’il n’y avait aucun objet sélectionné.

17 juin 2025 :

* J’ai pas repris ce journal depuis un moment. J’ai réussi à implémenter le système de la sélection multiple depuis l’inspecteur hiérarchique et la fenêtre 3D.

18 juin 2025 :

J’ai fait une correction du système de déplacement. Maintenant quand on manipule les objets, on le fait à partir du centre d’un gizmo dessiné par le centre de la position des objets de la liste des objets sélectionnés. Conséquence, les Objets peuvent être scale ou orbiter autour d’un centre lors des rotations

19 juin 2025 :

C’est un projet qui avance bien ! j’ai réussi à faire un début de système de hiérarchisation d’abord en bossant sur la liste de l’inspecteur hiérarchique. Il a fallut revoir la structure de sa fonction de rendering et corriger quelques absurdités.

Maintenant il faudrait que lorsqu’on détruit un objet enfant, il soit aussi retiré de la liste, que cette destruction s’applique aussi au reste de ses descendants.

22 juin 2025 :

J’ai implémenté la destruction récursive des objets 3D pour les enfants. Lorsqu’on détruit un objet, s’il s’avère qu’il a des enfants, ses enfants seront détruits.

J’ai rendu explicite aussi la matrice modèle du monde 3D dans le fichier ThreedSceneDrawer. On a maintenant de vraies coordonnées 3D globales. Quand on set la position d’un objet 3D à 0.0f , ça le set réellement au centre du monde.

03 Juillet 2025 :

Salut le journal ça faisait longtemps ! J’ai entièrement refactorisé mon système de hiérarchisation. Maintenant on fonctionne entièrement sur un système de slots, et j’ai rendu le drag and drop plus clair.

16 juillet 2025 :

Resalut le journal ! ça faisait également longtemps. J’ai terminé la refactorisation du système de hiérarchisation et j’en suis un peu fier.

Chaque slot est un bouton imgui, il peut s’agir soit d’un objet 3D, soit d’un empty (j’ai créé des emptys qui héritent des objets 3D afin d’occuper une place dans la mémoire et de permettre l’échange de slots).

Chaque slot contenant un objet peut avoir un enfant, et chaque enfant peut être déplacé librement.

18 juillet 2025 :

Règlement d’un bug qui faisait que les objets ayant été parentés peuvent avoir des transformations saccadées

20 juillet 2025 :

Simili devient petit un vrai moteur 3D. Chère journal de développement, désormais les parents appliquent leurs transformations sur leurs enfants. Et les enfants peuvent être transformés au niveau local.

Je ne sais pas encore comment ça va tourner pour la suite, et j’hésite quant à la direction à prendre. Je pense que que je vais problème travailler sur la modélisation 3D en elle-même avec les faces, vertices, et edges. Mais une chose est sûre : On avance.

* Mise à jour d’OpenGL vers la version 4.6
* J’ai commencé à créer le début des sommets 3D.

21 juillet 2025 :

* Début du dessin des Edges.
* Fin de la création des Faces et «  connexion » à leur modèle.

03 Août 2025 :

* On peut sélectionner les sommets des objets et faire apparaître le imGuizmo sur le sommet sélectionné
* On a un début de mode de sélection : Sélection normale de modèles 3D et Vertices

04 Août 2025 :

* J’ai commencé à refactoré plus sérieusement le dossier src/UI, notamment pour le Hierarchy Inspector. J’ai créé une classe HandleHierarchyInteractions qui est déclarée comme friendly class dans la classe Hierarchy Inspector, permettant d’employer les membres privés de Hierarchy Inspector à la première. Et j’ai séparé les responsabilités et je me retrouve avec un fichier moins conséquent (450 lignes environ contre 800).

08 Août 2025 (et depuis le 4 août) :

* J’ai fini de refactorer et de corriger le Hierarchy Inspector.
* J’ai fait un autre design pour l’inspecteur hiérarchique.
* J’ai fini de refactorer la manipulation des objets depuis la fenêtre 3D. J’ai mis tout ça dans le dossier Engine.
* J’ai fini de bosser sur la sélection et le déplacement des vertices et j’ai corrigé la mise à jour des faces lors du déplacement des vertices.
* J’ai perdu une bonne journée de développement car j’ai tenté d’intégrer CEF Chromium Embded Framework afin de voir si je pouvais remplacer l’UI par des fenêtres HTML. Ben j’ai laissé tombé. Et j’ai choisi arbitrairement de me contenter d’apprendre à réellement manier ImGUI.

11 Août 2025 :

* J’ai fini d’implémenter la sélection et la manipulation de Vertices
* J’ai créé le mode de sélection 3D pour les faces dans la fenêtre 3D
* J’ai implémenté la sélection et la transformation de faces.

12 Août 2025 :

* J’ai implémenté la sélection de edges ainsi que la manipulation

13 Août 2025 :

J’ai commencé à réorganiser le dossier de travail et les fichiers et je me suis débarassé du fichier AssemblerLogic. C’était cool mais ça servait à rien. Et j’ai corrigé le main en conséquence.

15 Août 2025 :

J’ai refactorisé le script ThreeDWindow.cpp et en déplaçant toute la fonction de click Handling dans une classe spécialisée clickHandler située dans le même dossier.

J’ai à peu près fini d’implémenter l’extrusion de face. La face extrude

16 Août 2025 :

J’ai commencé à bosser sur le concept de Mesh\_DNA, l’ADN du mesh. Et j’ai crée un système d’historique qui permet de remonter les transformations. Ce sera la base d’un système qui permette de remonter la scène.

17 Août 2025 :

* Correction d’un soucis avec le positionnement de la fenêtre de l’historique, et correction d’un problème de calcul des events de transformation qui causait un décalage entre le numéro de l’event et le nombre réel d’events. Maintenant il faut appliquer ce principe à chaque élément modifié du projet.
* J’ai viré la vieille classe cube. J’ai créé une classe Mesh qui centralise l’ajout de vertices, edges, et faces, et les initialise dans un maillage. Et j’ai créé des fichiers CreatePrimitive.cpp et hpp qui eux vont donner la forme d’un cube au maillage, qui aura toujours pour type Mesh. J’ai ensuite remplacé tous les dynamic cast cube par des dynamic cast Mesh dans le code.

18 Août 2025 :

* L’History peut suivre chaque transformation des objets 3D.