Projet TPI - Conventus

Application de conception et simulation de circuits logiques



**Mathias Renoult**

Étudiant - Développeur

CPNV - SI-MI4B

[Mathias.renoult@cpnv.ch](mailto:Mathias.renoult@cpnv.ch)

**Loïc Viret**

Chef de projet

CPNV

[Loic.viret@cpnv.ch](mailto:Loic.viret@cpnv.ch)

**Jonathan Melly**

Expert

ETML

Jonathan.melly@eduvaud.ch

**Romain Gehrig**

Expert

[Romain.gehrig@gmail.com](mailto:Romain.gehrig@gmail.com)

Glossaire

|  |  |
| --- | --- |
| Aggrégation | C'est le fait de pouvoir stocker un objet dans un autre objet. |
| Algèbre de bool | Approche algébrique de la logique. Permet de représenter des schémas logique sous forme d'expressions. |
| ALU (Arithmetic Logic Unit) | Composant chargé d'effectuer des calculs. Souvent faisant partie d'un microprocesseur. |
| AND | Porte logique "ET" (⋀) requérant toutes ses entrées étant allumée pour renvoyer un signal positif. |
| Asset | Entité Unity que l'on peut voir dans l'application ou dans le projet. Il peut venir de l'extérieur ou être créé par l'utilisateur. |
| Asset Store | Magasin en ligne permettant de télécharger et ajouter des *assets* créé par d'autres personne dans notre projet Unity. |
| Attribut | Variable en C# étant définie par un type, un niveau de protection et un nom. Les attributs composent les classes. |
| C# | Language de programmation orienté objet développé par Microsoft et proche du Java. C'est le language de *scripting* d'Unity |
| Classe | Moule permettant de servir de patrons lors de la création d'objets. Lorsqu'on utilise une classe pour créer un objet, on crée une *instance* de cette classe. |
| Component | *Asset* Unity ou classe ajoutable à un *GameObject* pour interagir avec lui. |
| Conventus | Nom de l'application. Signifie "Assemblée", "Assemblage" et plus rarement "Circuit" en latin. |
| CPNV (Centre Professionnel du Nord Vaudois) | École professionnelle dans laquelle se déroule le projet. |
| Encapsulation | On cache la structure de l'objet et on propose plutot des méhodes pour manipuler les propriétés de cet objet afin de s'assurer de la bonne manipulation de ceux-ci. |
| GameObject | Élément présent dans la hiérarchie du projet *Unity*. Chaque *GameObject* à un *Transform* qui lui est attaché et qui gère sa position, rotation et échelle. On peut lui ajouter des *components* au besoin. |
| Git Extensions | Interface visuelle pour *Git*. |
| GitHub | Service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de *versioning* *Git.* |
| Héritage | Un des quatre éléments centraux de la *POO.* L'*héritage* permet de transmettre des *attributs* et *méthodes* aux classe qui lui sont enfant. Permet l'utilisation du *polymorphisme*. |
| Instance | Objet créé à partir d'une classe source. |
| LabView | Logiciel informatique développé par National Instruments et qui permet de créer des systèmes de mesure de contrôle via une interface graphique. |
| LogiSim | Simulateur de circuits logiques libre et gratuit écrit en Java. |
| MA-01 | Module "Base du numérique" donné généralement en première année d'Informatique au CPNV. On y apprend notamment le binaire, l'hexadécimal et l'algèbre *booléenne.* |
| Méthode | Portion de code présent dans une classe pouvant être exécuté suite à un appel. Une méthode se caractérise par son niveau de protection, son type de retour, son nom et les arguments qu'elle demande. |
| NAND | Porte logique "NON-ET" (⊼) laissant passer le signal tant que les deux entrées ne sont pas activées. C'est l'équivalent théorique d'un transistor classique. |
| Nesting (nidification) | Procédé visant à encapsuler des composants pour en créer un nouveau. |
| NOR | Porte logique "NON-OU" (⊽) qui est laisse passer le signal seulement si aucune entrée n'est activée. |
| NOT | Porte logique "NOT" (¬) qui inverser le signal. |
| OOP (Object Oriented Programmation) | Se référer à *POO.* |
| OR | Porte logique "OU" qui est laisse passer le signal si au moins l'une des deux entrées est activée. |
| Polymorphisme | Un des quatre éléments centraux de la *POO.* Le polymorphisme permet d'exécuter un code spécifique en fonction du type de l'objet sur lequelle on appelle une *méthode.* |
| POO (Programmation Orientée Objet) | Paradigme de programmation centrée autour des objets. Ce paradigme à quatre piliers : *l'héritage,* le *polymorphisme,* *l'encapsulation* et *l'aggrégation* |
| Porte logique | Une porte logique est un élément physique ou logique qui prend généralement deux entrées binaires et renvoie un seul signal de sortie, également en binaire. |
| POS (Product Of Sum) | Expression booléenne faite à partir de la somme d'une ou plusieurs variables. |
| Scalable (échelonnable) | Augmenter ou réduire un ensemble de valeurs tout en gardant le rapport de grandeur entre elles. |
| Script | Fichier contenant du code. Désigne traditionnellement uniquement le code exécuté sur le CPU. |
| Shader | Fichier contenant du code. Désigne traditionnellement uniquement le code exécuté sur le GPU. |
| Singleton | Le *singleton* est un patron de conception dont l'objectif est de restreindre l'instantiation d'une classe à un seul et unique objet. Appelé parfois manager. |
| SMART | Acronyme signifiant Spécifique, Mesurable, Atteignable, Réalisable et Temporellement ancré. Cette méthode à pour but de forcer la création d'objectifs précis et cohérents. |
| SOC (Security Operations Center) | Acronyme désignant la partie sécurité d'un élément. |
| SOP (Sum Of Products) | Expression booléenne faisant la somme des produits d'un ensemble de variables étant liées entre elles par des portes *AND* et *OR.* |
| Static | Propriété du *C#* pouvant être appliqué à un *attribut* ou une *classe.* Cela permet d'interdire l'instanciation d'une classe ou d'un attribut. |
| Transistor | Élément électronique utilisé pour faire des processeurs notamment. C'est l'équivalent de la porte *NAND* logique. |
| Trigonométrie | Ensemble de méthodes de calcul permettant de calculer des angles et des distance dans des triangles, le plus souvent rectangles. |
| Unity | Moteur de jeux-vidéos 2D ou 3D utilisant le *C#* comme langage de scripting. Logiciel utilisé pour créer l'application de ce projet. |
| XNOR | Porte logique "NON-OU-EXCLUSIF" (=) |
| XOR | Porte logique "OU-EXCLUSIF" (⊻) qui laisse passer le signal si les deux entrées sont différentes. |

Table des matières

Table des matières

[Glossaire 1](#_Toc73622759)

[Table des matières 4](#_Toc73622760)

[Résumé 7](#_Toc73622761)

[Analyse préliminaire 7](#_Toc73622762)

[Introduction 7](#_Toc73622763)

[Buts 7](#_Toc73622764)

[Objectifs 8](#_Toc73622765)

[Analyse 9](#_Toc73622766)

[Concept 9](#_Toc73622767)

[Difficultés attendues 9](#_Toc73622768)

[Planification initiale 11](#_Toc73622769)

[Planification détaillée 12](#_Toc73622770)

[Sprint 1 12](#_Toc73622771)

[Sprint 2 13](#_Toc73622772)

[Sprint 3 13](#_Toc73622773)

[Dossier de conception 14](#_Toc73622774)

[Matériel 14](#_Toc73622775)

[Hardware 14](#_Toc73622776)

[Système d’exploitation 14](#_Toc73622777)

[Logiciels et sites Internet 14](#_Toc73622778)

[Modèle conceptuel de données 15](#_Toc73622779)

[Programmation 16](#_Toc73622780)

[Versioning 17](#_Toc73622781)

[Conception des maquettes 17](#_Toc73622782)

[Présentation des maquettes 19](#_Toc73622783)

[Palette de couleurs 21](#_Toc73622784)

[Réalisation 22](#_Toc73622785)

[Structure du projet 22](#_Toc73622786)

[22](#_Toc73622787)

[Description de l’application 23](#_Toc73622788)

[Les components 23](#_Toc73622789)

[Déroulement du développement 24](#_Toc73622790)

[Interface 24](#_Toc73622791)

[Components : 24](#_Toc73622792)

[Câbles 31](#_Toc73622793)

[Export en PNG 33](#_Toc73622794)

[Synergie câbles – components 34](#_Toc73622795)

[Analyse du journal de travail 37](#_Toc73622796)

[Problèmes rencontrés 43](#_Toc73622797)

[Rétrospective 43](#_Toc73622798)

[Erreurs restantes 44](#_Toc73622799)

[Pour aller plus loin 44](#_Toc73622800)

[Tests effectués 44](#_Toc73622801)

[Documents fournis 44](#_Toc73622802)

[Conclusions 44](#_Toc73622803)

[Annexes 44](#_Toc73622804)

[Manuel d’utilisation 44](#_Toc73622805)

[Archives du projet 44](#_Toc73622806)

Résumé

Analyse préliminaire

Introduction

*Conventus* est une application de conception et de simulation de circuits logiques. Elle permet de placer des composants logiques (soit des portes logiques[[1]](#footnote-1) basiques, soit des composants créés par l’utilisateur) sur un canevas, de les relier et de tester le circuit en choisissant le nombre d’entrées, de sorties, et leur état. L’un des objectifs non-officiel de ce projet est de servir de support pour le module *CPNV MA-01*. Bien qu’il existe déjà des applications permettant de faire de genre de simulations, je pense que celle-ci est unique dans son concept de création quasi automatique de composants personnalisés.

Buts

Étant donné que je souhaite livrer une application contenant plus de choses que demandé dans le cahier des charges, je vais séparer mes buts en deux : les buts impératifs et les buts circonstanciels

Impératifs :

* Se concentrer sur l’aspect ergonomique et « *easy-to-use* » de l’application
* Porter une attention particulière à la communication avec le chef de projet et les experts
* Livrer une application agréable à utiliser et conforme au cahier des charges
* Développer une interface responsive, interactive et *scalable*

Circonstanciels :

* Ajouter la fonctionnalité de *nesting* (création d’un nouveau composant à partir d’autres)
* Ajouter la fonctionnalité de simplification des circuits et de traduction en expressions algébriques booléennes
* Ajouter une section « Apprentissage » qui permettrait d’apprendre pas à pas comment fonctionne les circuits logiques et qu’est-ce qu’on peut faire avec

Objectifs

Les objectifs doivent respecter la méthode *SMART*. Ils servent à donner des cibles concrètes à viser lors de la réalisation du projet.

La grille d’évaluation reste le document d’autorité, néanmoins, la liste d’objectif ci-dessous ne perd pas sa valeur en production. La composante « T » du *SMART* dans les objectifs ci-dessous est implicitement « lors du rendu du projet ».

1. L’application est créée avec *Unity* et est en *C#*
2. L’utilisateur peut ajouter des portes logiques (*NOT, AND, OR, XOR, NAND, NOR, XNOR*) sur le canevas
3. L’utilisateur peut ajouter et supprimer les entrées et sorties du schéma. Il peut modifier l’état des entrées
4. L’application contient une partie « conception » qui lui permet de créer son circuit
5. L’application contient une partie « simulation » qui permet de tester le circuit créer par l’utilisateur. Il peut modifier l’état des entrées pendant la simulation
6. L’utilisateur peut enregistrer le schéma sur lequel il est en train de travailler et le rouvrir plus tard pour le modifier à sa guise
7. L’application supporte l’exportation des schémas sous forme de fichier image

Analyse

Concept

Avant toute chose, j’apprécie donner un nom aux application que je suis en train de développer. Cela me permet de mettre un mot sur le projet et de me motiver à faire quelque chose de réellement concret.

Pour mon projet Pré-TPI, j’avais choisis le nom *Spiti*. Cela signifie « maison » en grec et l’application devait me permettre de dessiner des plans de bâtiments et d’y placer des meubles en vue d’un déménagement. Pour rester dans la même logique, j’ai cherché un nom en fonction d’un élément central du projet. N’ayant rien trouvé de convaincant en grec, j’ai décidé de chercher du côté du latin. J’ai fini par trouvé un nom qui me plaisait bien : *Conventus*. Cela peut se traduire par « assemblage » ou plus rarement par « circuit ».

Il est clair que *Conventus* n’a pas la prétention de révolutionner le marché des simulations de circuits logiques. Il y a déjà beaucoup d’applications très complètes qui permettent de réaliser ce genre de choses et bien plus encore. On peut citer par exemple *MultiSim* ou *LabView*. En revanche, ce que j’espère fera la force de mon application, c’est son côté très simple à prendre en main et surtout pédagogique.

Difficultés attendues

Lors de ce projet, il y a plusieurs éléments nouveaux auxquels j’aurai à faire face. Afin de limiter au maximum les risques de blocage lors de l’implémentation des fonctionnalités, je vais lister ces potentiels points « à problème » et réfléchir à comment je pourrais limiter leur risque dans la suite de l’analyse et conception du projet. Voici donc les points que j’ai relevés :

1. Interface efficace et *scalable*

Je n’ai jamais réalisé d’interfaces réellement dynamiques. Je ne sais pas vraiment s’il y a un moyen assez « simple » d’automatiser la mise à l’échelle de celle-ci.

1. Portes bouclant sur elles-mêmes

J’ai peur qu’il y ait des soucis de clignotement s’il y a des boucles qui se forment dans le circuit. Une des solutions serait de détecter les boucles et de renvoyer un état « indéfini » à la place de 0 ou 1. Bien que j’aie déjà un début de solution, j’ai peur que cela soit techniquement ardu à mettre en place.

1. Simulation « au ralenti » du circuit

La simulation devrait pouvoir être jouée à différentes vitesses. C’est important pour le côté pédagogique de l’application. Le problème que cela me pose c’est que je vais devoir synchroniser d’une manière ou d’une autre les signaux et je n’ai pas encore trouvé de manière simple de faire cela.

1. Exportation du circuit en PNG

L’exportation du schéma en image était déjà un des points de mon Pré-TPI. Je n’ai pas réussi à mettre en place cette solution dans les temps, ni même de commencer les recherches à vrai dire. C’est pour ça que cela me fait un peu peur de devoir faire cela de nouveau. Je vais essayer de trouver une librairie ou un *asset* qui fait ça de manière quasi automatique.

Planification initiale

Le Logiciel **Gantt Project 2.8** a été utilisé pour mettre en place la planification initiale du projet. Les tâches sont regroupées par type **(analyse, conception, réalisation, tests et documentation**).

|  |  |  |
| --- | --- | --- |
| Nom | Date de début | Date de fin |
| Analyse | 03.05.2021 | 06.05.2021 |
| Création de la planification initiale | 03.05.2021 | 03.05.2021 |
| Créer le Git, le projet Unity, le journal de bord, le rapport | 03.05.2021 | 03.05.2021 |
| Recherche d'informations sur le sujet | 04.05.2021 | 04.05.2021 |
| Maquettes de l'app | 04.05.2021 | 04.05.2021 |
| Choix des conventions de nommage et architecteur du code | 04.05.2021 | 04.05.2021 |
| Début du rapport | 06.05.2021 | 06.05.2021 |
| Analyse du code d'applications similaires | 06.05.2021 | 06.05.2021 |
| Conception | 10.05.2021 | 11.05.2021 |
| Diagramme de flux sur les parties les plus critiques de l'app | 10.05.2021 | 11.05.2021 |
| Choix de la technologie de sauvegarde de fichier | 11.05.2021 | 11.05.2021 |
| Finalisation du dossier de l'analyse et du dossier de conception dans le rapport | 11.05.2021 | 11.05.2021 |
| Implémentation | 17.05.2021 | 27.05.2021 |
| Création de l'interface de base | 17.05.2021 | 17.05.2021 |
| Recherche des ressources (images et sons) | 17.05.2021 | 17.05.2021 |
| Création des managers et des classes "de base" | 17.05.2021 | 17.05.2021 |
| Création des managers et classes "de base" | 18.05.2021 | 18.05.2021 |
| Codage des "câbles" qui relient les éléments | 18.05.2021 | 18.05.2021 |
| Codage des portes logiques de base | 18.05.2021 | 18.05.2021 |
| Fonctionnalité "simulation" du circuit | 20.05.2021 | 21.05.2021 |
| Affichage des états des portes et des câbles | 21.05.2021 | 21.05.2021 |
| Enregistrement des fichiers | 25.05.2021 | 25.05.2021 |
| Exportation de fichiers image | 25.05.2021 | 27.05.2021 |
| Tests et Finalisation | 28.05.2021 | 04.06.2021 |
| Session de tests approfondie | 28.05.2021 | 28.05.2021 |
| Session de test approfondie | 31.05.2021 | 31.05.2021 |
| Terminer le rapport | 31.05.2021 | 01.06.2021 |
| Correction du rapport, enjolivage, manuel utilisateur | 03.06.2021 | 03.06.2021 |
| Rendu final TPI | 04.06.2021 | 04.06.2021 |

Planification détaillée

Comme solution de gestion de projet, j’ai décidé de partir sur la solution intégrée à *GitHub* directement dans l’onglet « Projects » du dépôt sur les conseils de Mr. Viret. J’ai utilisé le modèle « Standard Kanban ».

C’est une méthode de gestion *agile* tout en restant simple et légère à utiliser. Cela signifie que j’ai des *sprints* qui segmentent mon projet et une liste de tâche dans chacune de celles-ci.

La durée des *sprints* n’est pas fixe, comme le demande normalement la norme agile. C’est parce qu’avec les fête de l’ascension et la pentecôte, un des *sprints* aurait été très court. J’ai préféré me concentrer sur des sprints fonctionnels.

Voici les sprints et les tâches :

Sprint 1

**03.05.21 - 17.05.21** Analyse et conception. Création de l'interface de base, recherche des ressources nécessaire au projet, création des managers et des classes "de base".

* Création de tests
* Finaliser la partie analyse et conception du rapport
* Trouver comment exporter le projet en tant que PNG et expliquer le processus de fonctionnement
* Choisir la technologie de sauvegarde de fichier et réaliser un diagramme expliquant le processus de sauvegarde
* Diagramme de flux du calcul de la *table de vérité*
* Diagramme de flux du « *nesting »* des composants
* Diagrammes de flux de la simulation d’un circuit
* Choisir les conventions de nommage et l’architecture du code
* Maquettes de l’application
* Création du Git
* Créer le document du rapport et documenter l’analyse
* Analyser le code d’applications similaires
* Création de la planification initiale
* Création du journal de bord
* Recherche d’infos sur le sujet

Sprint 2

**18.05.21 - 28.05.21** Création de l'application en soi. Placement des portes logiques, simulation, sauvegarde et export des fichiers.

* Exportation d’un projet en PNG
* Enregistrement du projet & gestion de projets
* Affichage de l’état des câbles et des composants
* Programmation de la simulation des circuits
* Programmation des portes logiques
* Programmation des « câbles » qui relieront les éléments entre eux
* Création des managers et des classes « de base »

Sprint 3

* **28.05.21 - 04.06.21** Déroulement des tests. Correction d'un maximum de bugs. Finalisation du rapport.
* Préparation de la défense TPI
* Envoyer le mail de rendu
* Impression du rapport et reliage
* Création du manuel utilisateur
* Corriger le rapport
* Terminer le rapport
* Session de test approfondie
* Session de test approfondie 2

Dossier de conception

Matériel

Le développement d’une application de si petite envergure ne nécessite pas de matériel très spécifique. J’utiliserai mon poste de travail au *CPNV*. En voici les spécifications :

Hardware

* Dell OptiPlex 7050
* ACPI x64 based PC
* Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
* Intel(R) HD Graphics 530
* 16 GO de RAM
* Souris Microsoft HID
* Calvier Dell Standard

Système d’exploitation

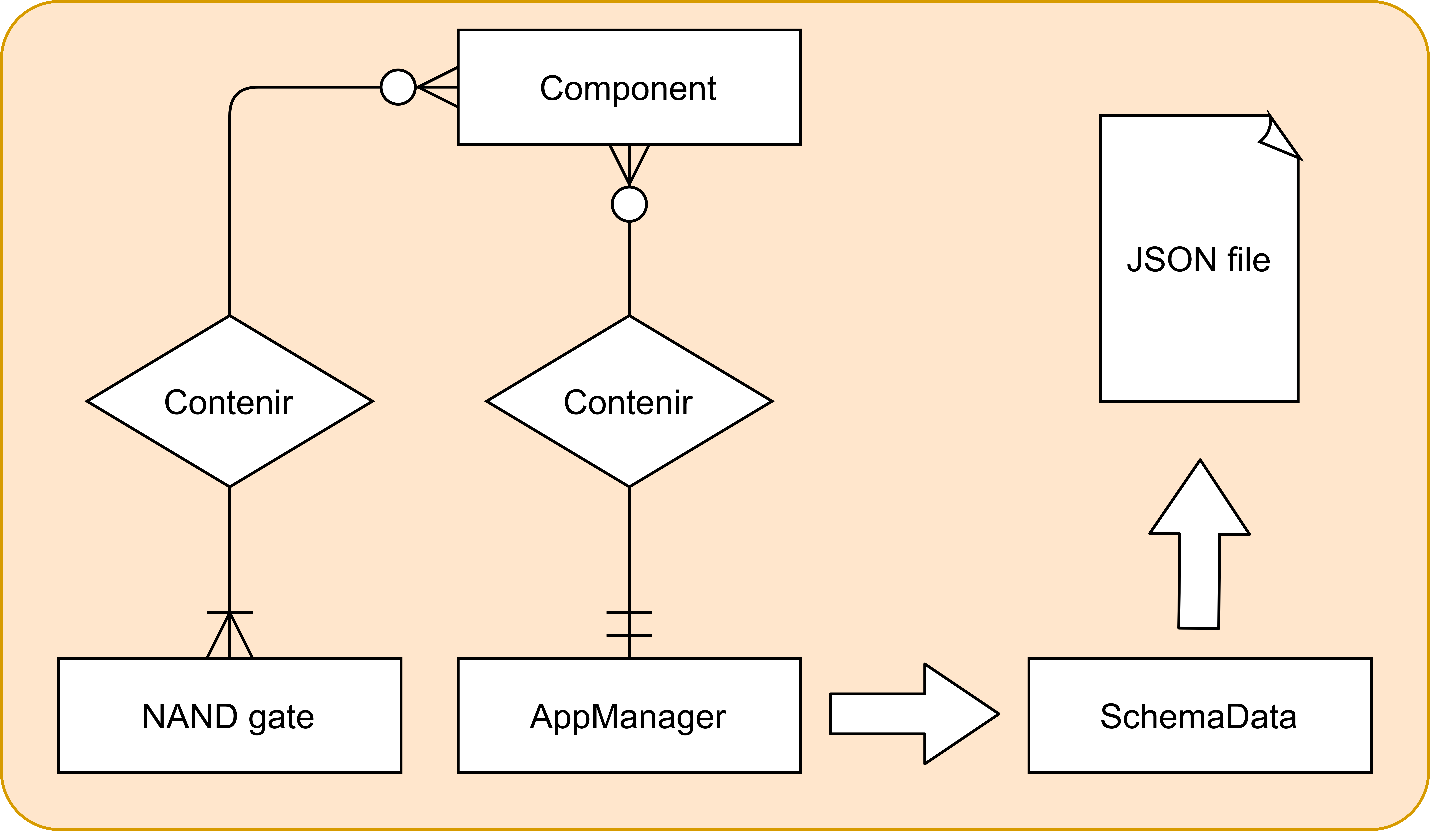
Le système d’exploitation est Microsoft Windows 10 Education.

Logiciels et sites Internet

J’essaye au maximum de prendre le plus possible des logiciels gratuits et si possible open-source. J’aimerais que tout le monde puisse, à moindre coûts, suivre cette documentation et tester / réaliser les mêmes choses que moi. Voici les logiciels principaux que j’utiliserai :

* Unity 2020.2.2f1
* Visual Studio Code (version 1.53)
* Gantt Project Manager 2.8
* Git Extensions
* Draw.io
* Coolors.co
* NounProject.com
* Photopea.com
* Suite Office
* GitHub.com
* Jimdo.com

Modèle conceptuel de données

Pour ce projet, je souhaiterais faire deux types de fichiers de sauvegarde : un fichier complet contenant les informations sur tous les composants du schéma et un fichier ne contenant que l’expression algébrique du schéma. La logique de sauvegarde derrière restant la même, je ne vais présenter qu’un seul et unique schéma.

J’ai décidé d’utiliser du *JSON* comme format de fichier parce que c’est un format flexible et que je connais bien. Je l’ai utilisé pendant mon Pré-TPI et je n’ai pas eu de problème avec. De plus, *Unity* intègre directement dans ses librairies un utilitaire *JSON* qui permet de le *parser/déparser* très simplement.

Programmation

*Unity* propose deux manières de programmer : soit via les *scripts* en *C#,* soit via les *blueprints*. Ces derniers permettent de faire théoriquement presque les mêmes choses que les *scripts C#* mais sans avoir à taper une seule ligne de code.

J’ai décidé d’utiliser la version textuelle pour plusieurs raisons :

1. Je connais beaucoup mieux le *C#*
2. La majorité des gens ne travaillent pas avec les *blueprints*, il y a donc moins d’aide en ligne
3. La technologie *blueprints* est plus difficile à optimiser (en terme de performances)

Ce n’est pas encore une certitude, mais j’aimerai bien toucher un petit peu au *shaders.* J’ai eu l’opportunité de m’y frotter un peu lors de mon stage où j’en ai créé un en *HLSL* pour *Unity*. La création de *shaders* est une discipline très spécifique et assez particulière. Si je n’ai pas le temps d’en faire ce ne sera pas un problème.

Pour le nommage j’utiliserai du *CamelCase* et du *pascalCase*. Ce sont les conventions de nommage que je préfère car elles sont rapides à écrire et élégante.

En *pascalCase* :

* Les *attributs*

En *CamelCase* :

* Les *classes*
* Les *méthodes*
* Les *dictionnaires*
* Les *énums*
* *GameObjects*

Lors de ce projet, j’aurai besoin de managers. N’appréciant pas travailler avec les *classes statiques*, je vais préférer l’utilisation de *singletons* lors de ce projet. Ceux-ci sont pratique à utiliser, optimisés, et j’ai l’habitude de travailler avec lors de ce genre de projets.

Versioning

Il y a plusieurs solutions pour assurer le *versioning* d’un projet sur *Unity* : *Colaborate*, *GitHub*, SourceTree, *Git Kraken*, etc…Ayant travaillé lors de mon année de stage sur *Unity*, et ayant utilisé *Colaborate* également, j’ai pu me rendre compte de l’inefficacité de cette solution. Lors de mon pré-TPI j’ai utilisé un *asset* téléchargé sur l’*asset store* s’appelant *GitHub for Unity*; c’était une catastrophe. J’ai eu énormément de problème de *merge* parce que j’avais travaillé sur deux ordinateurs différents.

Il me faut donc trouver une alternative gratuite, pratique et efficace. Suite aux conseils de Mr. Viret, je suis parti sur *Git Extensions*. C’est une interface graphique pour *Git* qui comporte beaucoup d’outils intéressant tout en restant relativement simple dans son fonctionnement.

Conception des maquettes

Étant à l’aise avec la manipulation du *webgiciel* *Draw.io*, j’ai décidé de l’utiliser pour réaliser les maquettes de l’application. J’aime cette application parce qu’elle est extrêmement complète tout en restant très simple dans sa mise en œuvre,

Contrairement à mon Pré-TPI, cette application n’aura qu’une seule et unique *scène*. Cela me simplifie beaucoup la gestion des managers, et cela me demande également moins de travail sur le design et la création de l’interface car tout est regroupé au même endroit.

J’ai néanmoins plusieurs maquettes car j’ai également conceptualisé les panneau « pop-up » (*nesting* et *editing*) et le menu de gestion des composants.

Sur les deux pages suivantes se trouvent mes maquettes que je viens de mentionner.

Stratégie de tests

Méthode

Il y a deux types de tests : les scénarios et les tests unitaires. Les scénarios se concentrent sur l’expérience utilisateur et les test unitaires plutôt sur la fiabilité des résultats engendrés suite à une action utilisateur. *Unity* possède un système interne permettant de facilement créer des tests unitaires. Je n’ai cependant encore jamais utilisé cet outil, je vais donc devoir me documenter un peu. Ne considérent pas les tests unitaires comme étant absolument déterminants dans le cas du développement de mon application, je ne prévois d’en faire qu’un seul. À l’inverse, les scénarios me permettront de traquer les bugs restant plus facilement et ainsi de pouvoir reprendre le projet en solo en partant sur de bonnes bases.

Les données seront testées à deux endroits différents. Pour le test unitaire ce sera fait sur l’ordinateur du CPNV depuis l’éditeur d’*Unity*. Pour les scénarios, l’ordinateur utilisé sera celui de Mr. Olivier Renoult, un proche. J’ai décidé de faire les tests des scénarios sur une autre machine et par une autre personne afin de me rapprocher au plus d’une expérience utilisateur classique.

Mr. Olivier Renout se verra remettre un exemple du tableau ci-dessous des tests à effectuer ainsi que d’un fichier à remplir avec les résultats des tests.

Je m’assurerai moi-même de déterminer la cause des potentiels problèmes repérés par le testeur ; la source de ces problèmes étant soit une erreur de ma part dans la conception même de l’application (bug) soit une incompréhension de la part de l’utilisateur (instructions pas claires / applications peu intuitive ou peu ergonomique).

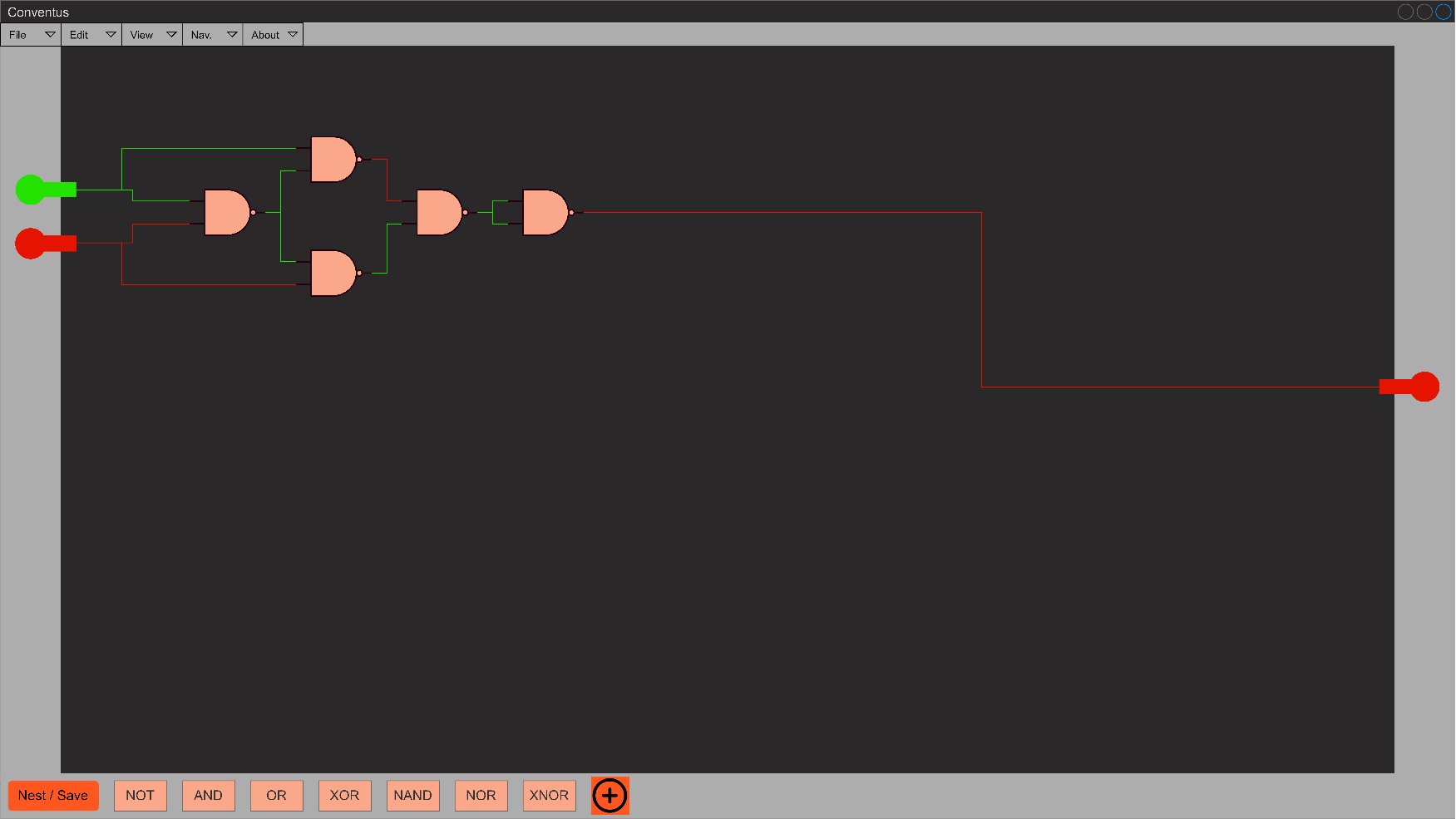
Test unitaire

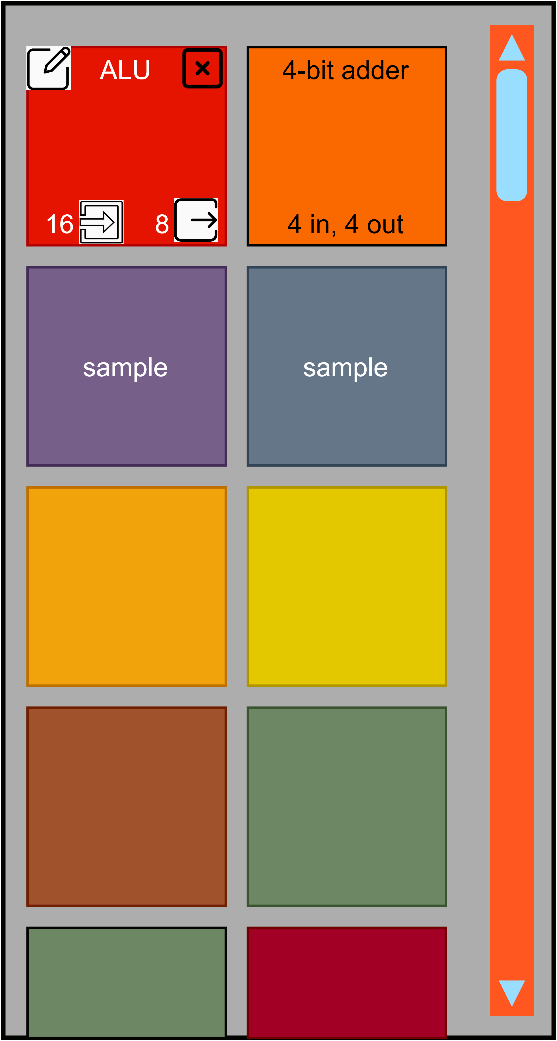
Une porte *XOR* est créée. Deux inputs et un output sont rajoutés au canevas. On relie les deux inputs du canevas aux entrées de la porte *XOR*. On relie la sortie de la porte *XOR* à l’output du canevas. On lance la simulation. On bascule l’état des deux entrées du canevas sur « *true* » On test que la sortie du canevas soit bien « *false* ». On allume la première entrée, la sortie devrait être sur « *true* ». On éteint la première entrée et on allume la deuxième, la sortie devrait rester sur « *true* ». On allume à nouveau la première entrée, la sortie devrait être sur « *false* ».

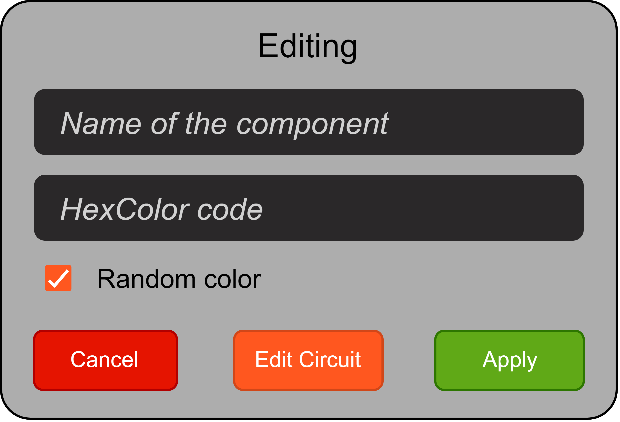
Si les quatre conditions de ce test unitaire sont remplies, cela veut dire que la porte *XOR* fonctionne comme elle le devrait, c’est-à-dire conformément à la table de vérité officielle de cette porte.

Liste des scénarios

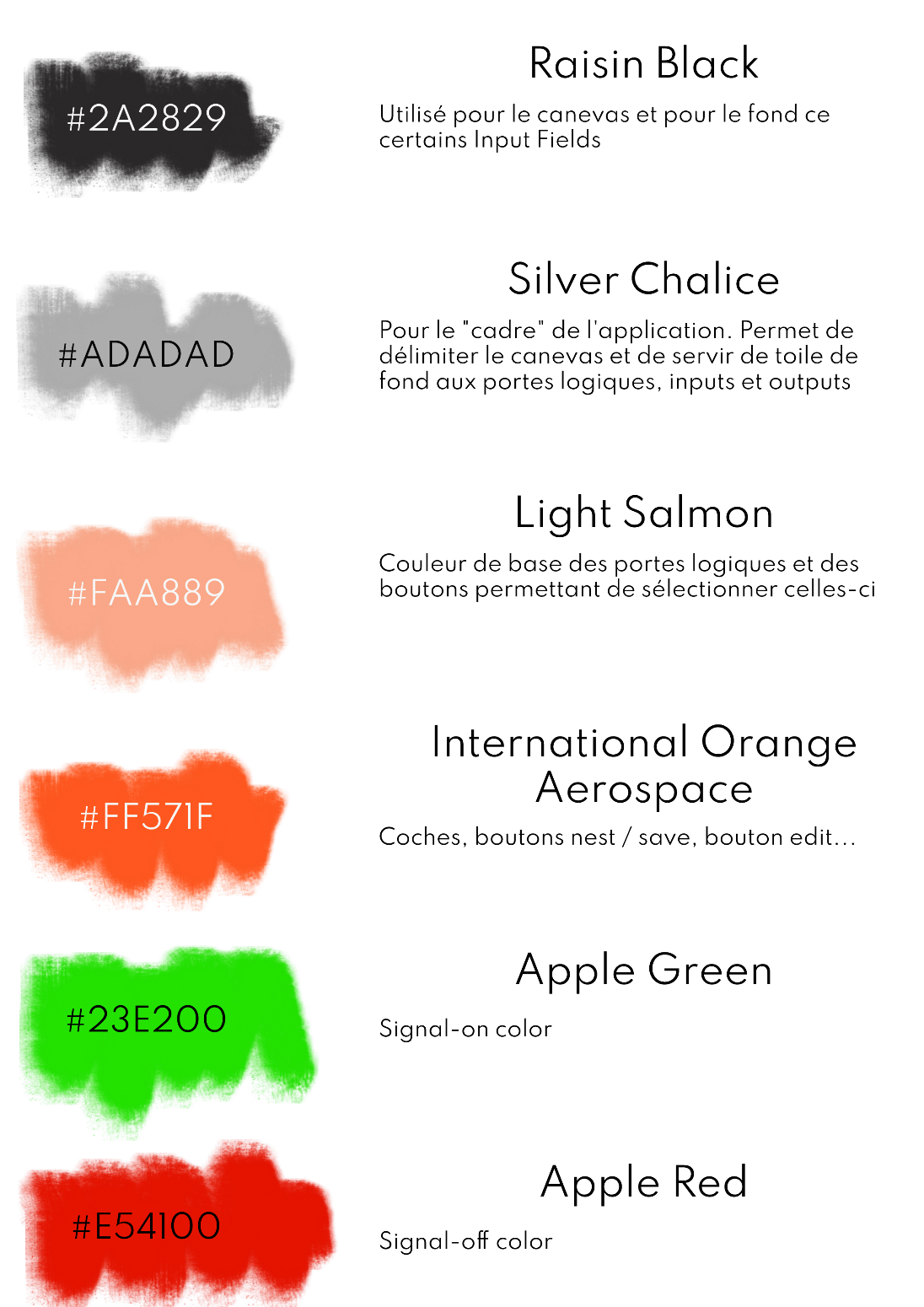
|  |  |  |  |
| --- | --- | --- | --- |
| ID | Étant donné | Action | Résultat attendu |
| 1.1 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur passe sa souris sur le bouton « Fichiers » en haut de l’écran. | Quatre onglets se déroule en dessous du bouton survolé. |
| 1.2 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur passe sa souris sur le bouton « Édition » en haut de l’écran. | Deux onglets se déroule en dessous du bouton survolé. |
| 1.3 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur passe sa souris sur le bouton « Vue » en haut de l’écran. | Quatre onglets se déroule en dessous du bouton survolé. |
| 1.4 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur passe sa souris sur le bouton « À propos » en haut de l’écran. | Deux onglets se déroule en dessous du bouton survolé. |
| 1.5 | L’utilisateur survole le bouton « Fichiers » avec son curseur. | L’utilisateur clique sur l’onglet « Exporter PNG ». | Un retour visuel indique à l’utilisateur que l’export est réussi. L’export se situe dans le dossier « Background » du répertoire permanent de l’application. |
| 1.6 | L’utilisateur survole le bouton « Fichiers » avec son curseur. | L’utilisateur clique sur l’onglet « Sauvergarder ». | Un retour visuel indique à l’utilisateur que la sauvegarde est réussie. Les components qui n’étaient pas déjà sauvegardés se sont sauvegardés. |
| 1.7 | L’utilisateur survole le bouton « Fichiers » avec son curseur. Des nouveaux components sous forme de fichiers json ont été glissés dans le dossier « components » de l’application pendant qu’elle était allumée. | L’utilisateur clique sur l’onglet « Charger ». | Un retour visuel indique à l’utilisateur que la charge des components est réussie. Les components récemment ajouté au dossier « Components » se retrouve maintenant dans le panel components de l’application. |
| 1.8 | L’utilisateur survole le bouton « Fichiers » avec son curseur. | L’utilisateur clique sur l’onglet « Quitter » | L’application se ferme. |
| 1.9 | L’utilisateur survole le bouton « Édition » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. La simulation n’est pas lancée. | L’utilisateur clique sur l’onglet « Simuler ». | Les câbles et les inputs et outputs du canevas se colorent alors en vert et rouge en conformément à l’état qu’ils possèdent. |
| 1.10 | L’utilisateur survole le bouton « Édition » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. La simulation est déjà lancée. | L’utilisateur clique sur l’onglet « Simuler ». | Les câbles et les inputs et outputs du canevas reprennent leur couleur blanche. |
| 1.11 | L’utilisateur survole le bouton « Édition » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. | L’utilisateur clique sur l’onglet « Vider le canevas ». | Tous les composants et les câbles du canevas se suppriment sans délai. |
| 1.12 | L’utilisateur survole le bouton « Vue » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. | L’utilisateur clique sur l’onglet « Centrer ». | La vue se centre sur les components. (moyenne de la position de tous les components à l’écran) |
| 1.13 | L’utilisateur survole le bouton « Vue » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. La norme de portes « IEC » est activée. | L’utilisateur clique sur l’onglet « ANSI ». | Les portes changent d’apparence pour correspondre à celle de la norme « ANSI ». |
| 1.14 | L’utilisateur survole le bouton « Vue » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. La norme de portes « ANSI » est activée. | L’utilisateur clique sur l’onglet « IEC ». | Les portes changent d’apparence pour correspondre à celle de la norme « IEC ». |
| 1.15 | L’utilisateur survole le bouton « Vue » avec son curseur. Un circuit basique est dessiné sur la canevas et relié aux inputs et outputs. La norme de portes « ANSI » est activée. | L’utilisateur clique sur l’onglet « DIN ». | Les portes changent d’apparence pour correspondre à celle de la norme « DIN ». |
| 1.16 | L’utilisateur survole le bouton « À propos » avec son curseur. | L’utilisateur clique sur l’onglet « Tutoriel ». | La fenêtre du tutoriel s’ouvre. |
| 1.17 | L’utilisateur survole le bouton « À propos » avec son curseur. | L’utilisateur clique sur l’onglet « Crédits ». | La fenêtre des crédits s’ouvre. |
| 1.18 | La fenêtre du tutoriel est ouverte. | L’utilisateur clique sur la croix en haut à droite de la fenêtre. | La fenêtre se ferme. |
| 1.19 | La fenêtre des crédits est ouverte. | L’utilisateur clique sur la croix en haut à droite de la fenêtre. | La fenêtre se ferme. |
| 2.1 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur clique sur le petit « + »vert en bas à gauche. | Un input s’ajoute au canevas. |
| 2.2 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. Un input est relié à un élément du canevas. | L’utilisateur clique sur le petit « - »rouge en bas à gauche. | L’input le plus haut se supprime. Le câble qui lui était lié l’est aussi. |
| 2.3 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur clique sur le petit « + »vert en bas à droite. | Un output s’ajoute au canevas. |
| 2.4 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. Un output est relié à un élément du canevas. | L’utilisateur clique sur le petit « - »rouge en bas à droite. | L’output le plus haut se supprime. Le câble qui lui était lié l’est aussi. |
| 2.5 | La simulation est active. | L’utilisateur clique sur un input. | La couleur de l’input passe de rouge à vert ou de vert à rouge. |
| 2.6 | La simulation est active. | L’utilisateur clique sur un output. | La couleur de l’output passe de rouge à vert ou de vert à rouge. |
| 3.1 | La simulation est active. | L’utilisateur conçoit un circuit logique à base de portes dont il connaît d’avance le fonctionnement. | Les couleurs des câbles et des outputs du canevas sont exactement celles attendues. |
| 3.2 | L’utilisateur a conçu un schéma logique complet et sans boucles sur lui-même | L’utilisateur clique sur le bouton « Créer » en bas à gauche. | La fenêtre de création de components s’ouvrent. |
| 3.3 | L’utilisateur a conçu un schéma logique puis a cliqué sur le bouton « Créer » en bas à gauche. | L’utilisateur remplit les champs de la fenêtre puis clique sur « Créer ». | Le component se sauvegarde bien et est ajouté dans le panel des components accessible en cliquant sur le « + » orange en bas. |
| 3.4 | L’utilisateur a déjà enregistré un component « Custom ». | L’utilisateur clique sur le bouton « + » en bas. | Le panel des components s’ouvre et il y a un item correspondant au component précédemment créé par l’utilisateur. |
| 3.5 | L’utilisateur a ouvert le panel des components et dedans se trouve au moins un component « Custom ». | L’utilisateur clique sur un des component. | Le component apparaît sur le canevas au milieu de l’écran. Il est de la bonne couleur. |
| 3.6 | L’utilisateur a ouvert le panel des components et dedans se trouve au moins un component « Custom ». | L’utilisateur clique sur l’icône de croix en haut à droite d’un item | L’item se supprime. |
| 3.7 | L’utilisateur a ouvert le panel des components et dedans se trouve au moins un component « Custom ». | L’utilisateur clique sur l’icône de calepin en haut à gauche d’un item | La fenêtre de modification d’un component s’ouvre. |
| 3.8 | L’utilisateur a ouvert la fenêtre de modification d’un component. | L’utilisateur change le nom du component ou sa couleur. | Les modifications apportées se répercutent sur l’item dans le panel des components. |
| 4.1 | L’application est ouverte normalement. Aucune fenêtre n’est ouverte. | L’utilisateur glisse et dépose une porte logique en cliquant sur un des boutons prévus à cet effet en bas de l’écran. | La porte logique correspondant à l’inscription sur le bouton apparaît sur le canevas. |
| 4.2 | Un component est placé sur le canevas. | L’utilisateur clique (gauche) sur le component, laisse enfoncé le bouton de la souris, déplace sa souris puis relâche son clic. | Le component se déplace en suivant le curseur de la souris. |
| 4.3 | Deux components sont placé sur le canevas. | L’utilisateur relie output vers un output. | Un câble relie correctement les deux IO sélectionnés par l’utilisateur. |
| 4.4 | Deux components sont placé sur le canevas. | L’utilisateur essaie de relié un output à un output ou un input à un input. | Au moment où l’utilisateur lâche son clic de souris, le câble se détruit. |
| 4.5 | Un component est placé sur le canevas. | L’utilisateur commence à faire un câble depuis un IO du component mais lâche sa souris dans le vide. | Le câble de détruit. |
| 4.7 | Un component est placé sur le canevas. | L’utilisateur déplace le component sur la barre en bas de l’écran (vers la poubelle orange). | Le component se détruit ainsi que tous les câbles qui lui sont reliés. |
| 4.8 | L’utilisateur a placé un component « Custom » dont il connaît le comportement sur le canevas. Ce component est relié à d’autres éléments de manière correcte. La simulation est active. | Rien | Le component « Custom » agit comme il devrait. (Sa table de vérité est respectée) |
| 4.9 | L’utilisateur a placé deux components sur le canevas et ils sont reliés de manière logique. | L’utilisateur crée un nouveau component « Custom » à partir du canevas. | Un nouveau component se crée conformément au schéma conçu par l’utilisateur. Sa table de vérité est conforme à celle qu’on attend de lui. |

Présentation des maquettes





Palette de couleurs

Afin de s’assurer que le visuel de l’application soit agréable à l’œil, j’ai utilisé le site *coolors.co* pour générer une palette de couleurs cohérente. Je me suis inspiré pour les couleurs principale d’élément naturels, passant l’orange melon à l’orange cucurbitacée. J’ai ensuite « peint » mes maquettes avec cette dernière. Je trouve que le résultat est satisfaisant. Voici la palette en question :

Réalisation

Structure du projet

La structure de ce projet est plutôt simple et classique. À la racine du projet, il y a beaucoup de dossiers mais un seul nous intéresse, le dossier « *Assets* ». C’est dans celui-ci que se trouve toute notre application : *scripts*, images, polices, *shaders*, *scenes*, etc.

Le schéma ci-dessous devrait être assez explicite.

## 

Je vais quand-même expliquer ce qui se trouve dans les dossiers principaux :

**Assets :** Contient tout ce qui se trouve dans l’application. C’est le contenu de l’application.

**TextMeshPro :** C’est un plugin permettant d’avoir des textes sous forme de *meshs* (maillages / modèle 3D). Donne également plus de fonctionnalités sur la customisation des textes.

**Plugins :** Contient les plugins. Il y a notamment le *SVG* Importer.

**Prefabs :** Les *prefabs* sont des objets Unity stockés afin d’être ré-instanciés plus tard. Typiquement, on retrouve les canevas des *câbles, portes logiques* et des *components*.

**Scenes :** Ce dossier contient les *scènes* utilisées dans l’app. Il y en a que deux qui nous intéresse : le menu et la *scène* principale.

**Scripts :** Contient tous les *scripts* utilisés dans l’app. Je les ai classés par catégories qui me parlaient. Il y aurait plein d’autres manières de les classer...

**Shaders :** Dans ce projet, les *shaders* n’ont pas une très grande importance. Mais en général, c’est dans de dossier qu’on les stocke.

**Ressources :** C’est ici que se trouvent les images utilisées, les polices et les matériaux.

**Library :** Dossier où se trouvent toutes les librairies utilisées. Je n’ai pas eu besoin d’utiliser des libraires externes à *Unity*.

**ProjectSettings :** Paramètres du projet

**UserSettings :** Paramètres utilisateur

Description de l’application

Les components

La logique utilisée pour les *portes logiques* est assez simple, presque archaïque. En fait, c’est un *GameObject* auquel est attaché la *classe* *<Component* >. Cette dernière est commune à tous les éléments présents sur le canevas (à l’exception des câbles). Les *portes logiques* sont en quelques sortes un « cas particulier » de *component*. Cette classe à un *enum* intitulé sobrement « Type ». Il y a 10 types de *components* *: Buffer, NOT, AND, OR, XOR, NAND, NOR, XNOR*, Custom et Unset. Chaque *porte logique* à son type propre et il y a le type « Custom » pour les *components* créés par les utilisateurs. Le type « Unset » est utilisé lorsque l’on ne sait pas quel rôle est sensé joué le *component*.

Déroulement du développement

Interface

Components :

Il existe 10 types de composants que l’on peut classer en 2 catégories :

**Portes logiques :**

* Buffer
* NOT
* AND
* OR
* XOR
* NAND
* NOR
* XNOR

**Custom** :

* Custom

**Non-classée :**

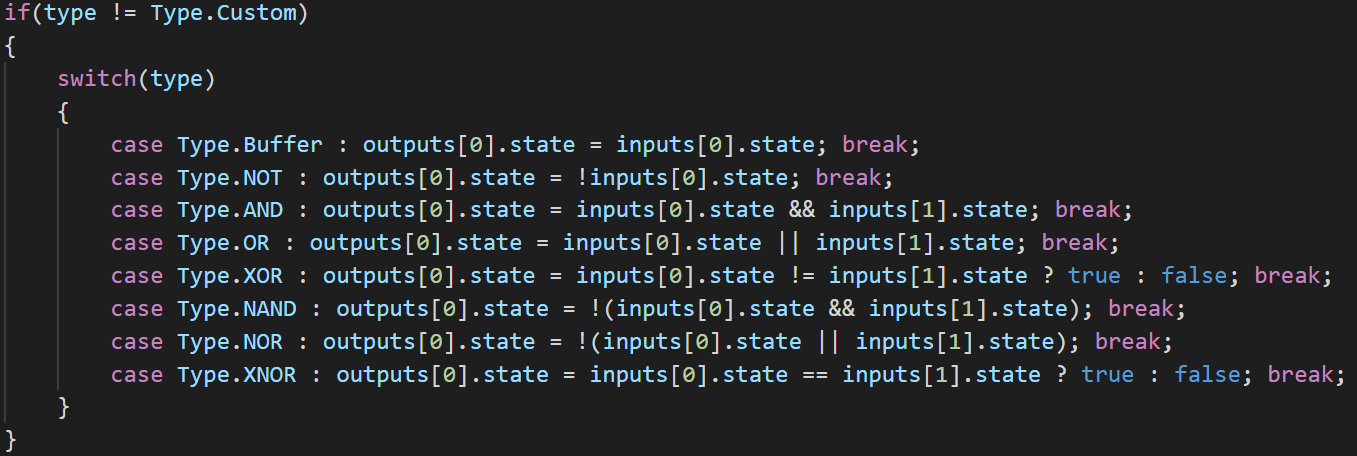
* Unset

Ces deux grandes catégories font réellement la différence en terme de logique d’utilisation de la classe « Component.cs ». Ci-dessous une petite clarification :

Portes logiques

Originellement, je me suis dit qu’il serait plus qualitatif d’utiliser des fichiers *SVG* pour représenter les *portes logiques*. J’ai trouvé une page WikiCommons mettant à disposition toutes les portes logiques des normes ANSI, IEL et DIN en téléchargement gratuitement. J’ai donc utilisé ces fichiers pour la première version des portes logiques dans l’application. Afin de pouvoir afficher du *SVG* dans *Unity*, il est nécessaire d’ajouter un *asset* s’appelant *SVG Importer*. C’est un *asset* développé par l’équipe d’*Unity* qui permet d’importer et de modifier les fichiers *SVG.* Malheureusement, c’est *asset* est encore ne développement et il existe que lamentablement peu de documentation dessus. J’ai essayé de faire des opérations simples avec comme par exemple changer la couleur des traits ou encore leur largeur, mais je n’y suis pas arrivé. Décidant de ne pas perdre du temps avec une technologie encore en développement, je suis passé sur du *PNG*. Avec le *PNG*, on perd en qualité et en flexibilité mais on gagne en facilité de manipulation.

Les portes logiques sont simulées presque directement dans le code. En effet, étant donné la simplicité de leur fonctionnement, l’on peut directement utiliser des opérateurs logiques pour calculer leur état. Voici le code permettant de simuler toutes les portes logiques de manière efficace et optimisée :



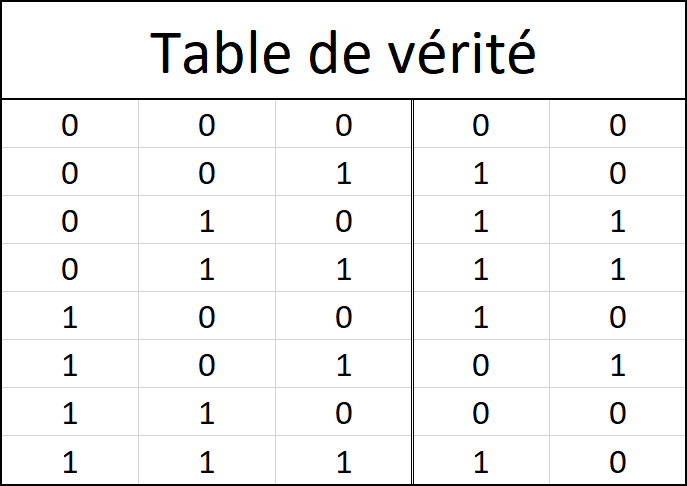
Cette optimisation permet d’accélérer un peu la simulation car, contrairement aux components, il n’y a pas besoin d’itérer dans un tableau pour déterminer l’état des *outputs*.

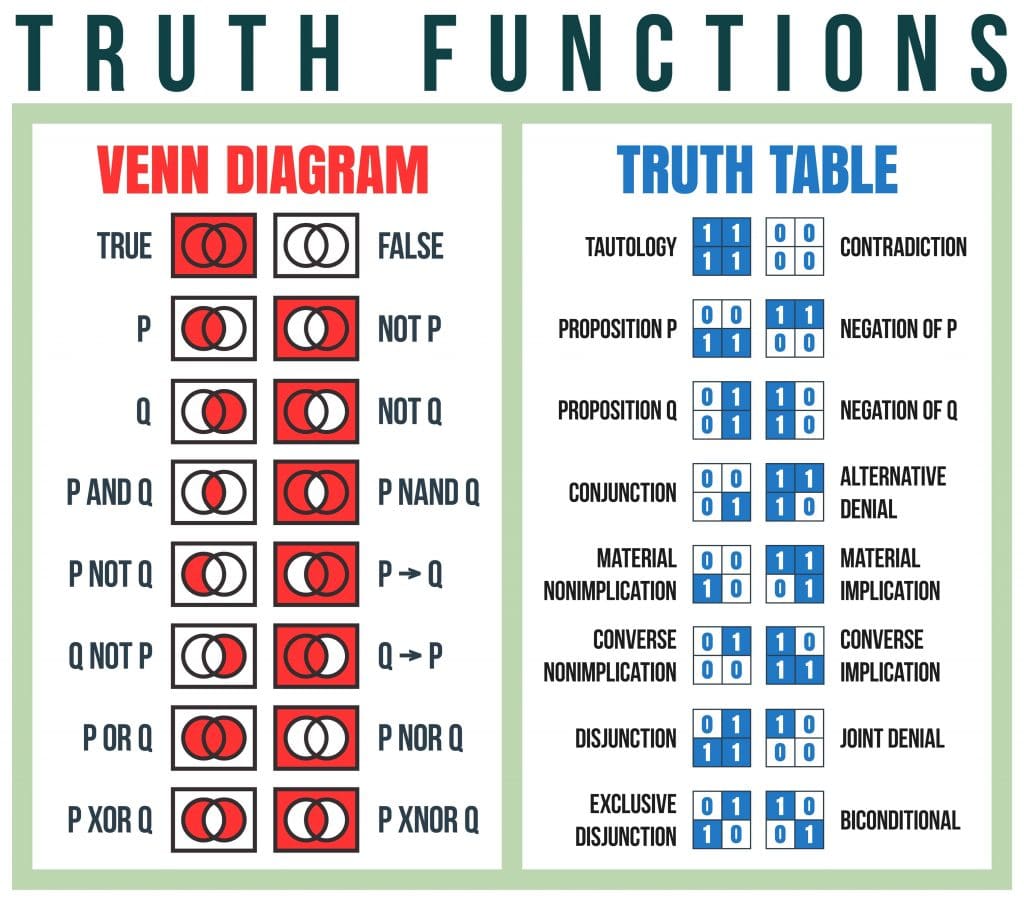
Custom

Les composants customs sont, selon moi, le point central de cette application. En effet, c’est grâce à cette fonctionnalité que l’on peut réaliser des projets d’envergure, dans les limites de la machine bien-sûr.

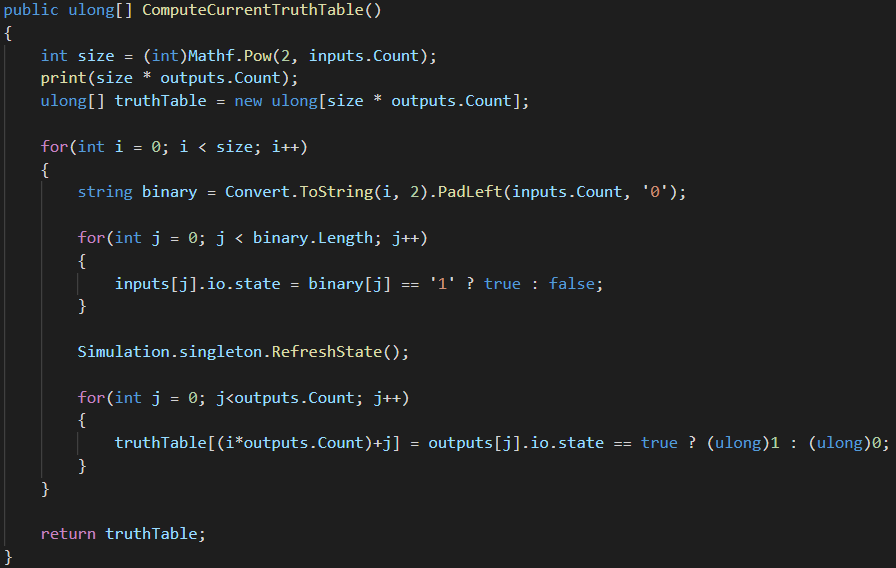
Il n’a pas été si simple de permettre aux utilisateurs de créer leurs propres composants et cela pour une raison assez simple : comment faire pour s’assurer que le composant nouvellement créer se comporte exactement de la manière attendue sans pour autant avoir à simuler en interne le comportement complexe de celui-ci ?

Au début du projet, j’avais plutôt privilégier une approche assez « fidèle » à la réalité. Je souhaitais que tout à chacun puisse comme « pénétrer » à l’intérieur de chaque composant afin de voir ce quoi il est composé. Le problème de cette approche est double : premièrement elle est techniquement assez compliquée à réaliser car cela nécessite de stocker l’équivalent d’un canevas complet dans chaque composant, sachant que ce composant peut contenir d’autres composants, et ainsi de suite. De plus, pour sérialiser un component au complet ce n’est pas si simple… Il y aurait facilement pu y avoir des dizaines de « couches » de profondeur dans les composants les plus complexes. Deuxièmement et conséquemment, la taille du stockage de ces composants aurait été relativement gigantesque car il aurait fallu stocker, plusieurs dizaines, voire centaines, de composants pour en créer un nouveau. Il aurait été possible de fonctionner par références uniquement mais cela aurait rendu la tâche encore plus ardue.

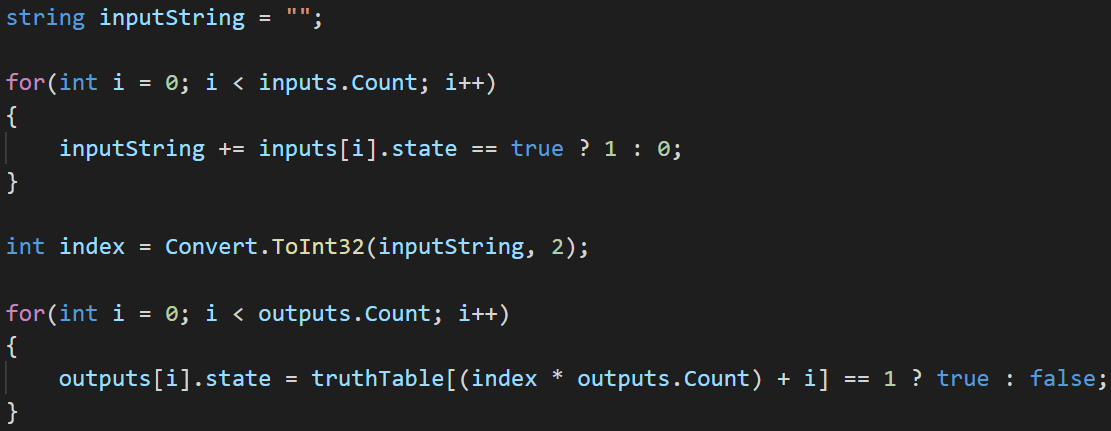


Je suis donc parti sur une approche moins « risquée » et sûrement aussi moins « aboutie » mais, qui fonctionne presque sans restrictions ! Lorsque l’utilisateur crée un component, je simule TOUS les états possibles du circuit et je stocke les résultats des outp uts. En fait, c’est tout simplement une table de vérité avec uniquement les sorties et pas les entrées. Pourquoi ne pas stocker aussi les entrées ? Tout simplement parce que c’est très simple de déterminer quelle sortie est associée à quelle entrée en fonction de son index. En effet, les tables de vérités étant toujours calculée selon la même procédure, je sais que la conversion de binaire à décimal des entrées est égal à l’index du tableau correspondant à la sortie. Si je souhaite donc savoir quelle entrée à défini la case n°17 de ma table de vérité (je précise que c’est une table à une dimension pour l’exemple), il me suffit de convertir 17 en binaire => 10001 => et ensuite d’associer chaque entrée à celles de mon canevas. Donc ici j’aurais dans l’ordre : *true, false, false, false* et *true*. Il suffit ensuite de comparer les entrées calculées et de voir si elles correspondent effectivement aux entrées réelles du composant pour déterminer si la simulation a renvoyé le bon résultat. 

Voici la méthode permettant de calculer la table de vérité du component créé :



Et la méthode utilisant la table de vérité pour calculer l’état des sorties :



À noter que si j’utilise un tableau de *ulong* à la place de *bool*, c’est parce que je préfère gagner un peu de place sur le disque dur. Cela me contraint donc à utiliser des ternaires pour convertir les 1 et les 0 en *true* et *false*.

Cependant, c’est encore un peu plus compliqué que cela. *Unity* intègre une *serializer* JSON, qui fonctionne assez bien d’ailleurs, mais qui a des limitations. Le plus gros problème de ce *serializer*, c’est qu’il n’est pas capable de sérialiser les tableaux à plus d’une dimension. Il m’était donc impossible de stocker des tables de vérités avec plus d’une sortie par état possible. J’ai donc dû, afin de quand-même pouvoir stocker mes tables de vérité, aplatir mon tableau. Je multiplie ou divise donc l’index de ma table en fonction du nombre de sorties par ligne.

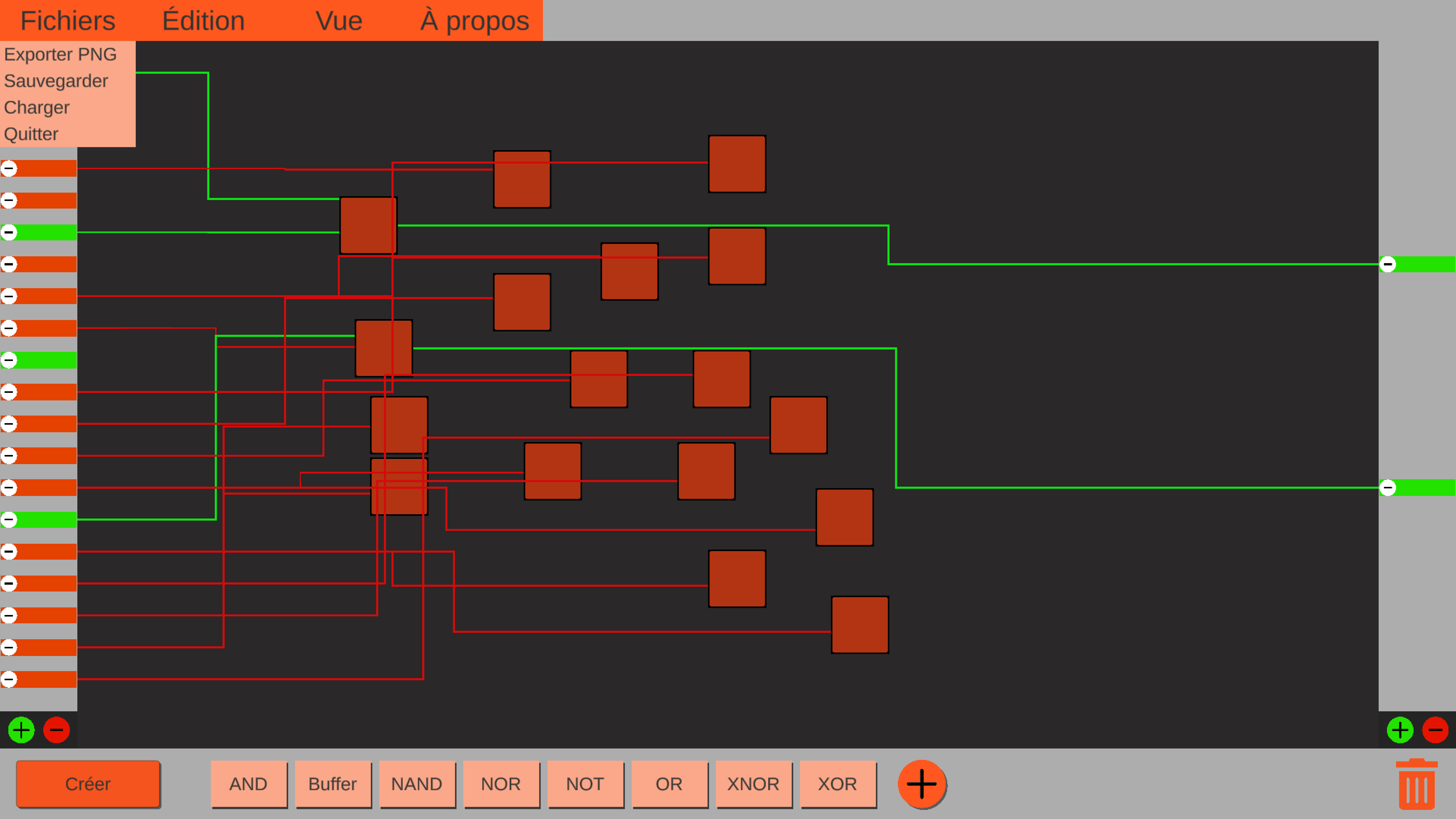
Quelle est l’efficacité de cet algorithme de création des tables de vérité ? Et surtout, est-ce que les gros composants ne risquent pas de prendre beaucoup trop de temps à se créer et de place sur le disque dur ?

La taille de la table de vérité se calcul selon cette formule avec *i* étant le nombre d’entrées du canevas et *o* le nombre de sorties :

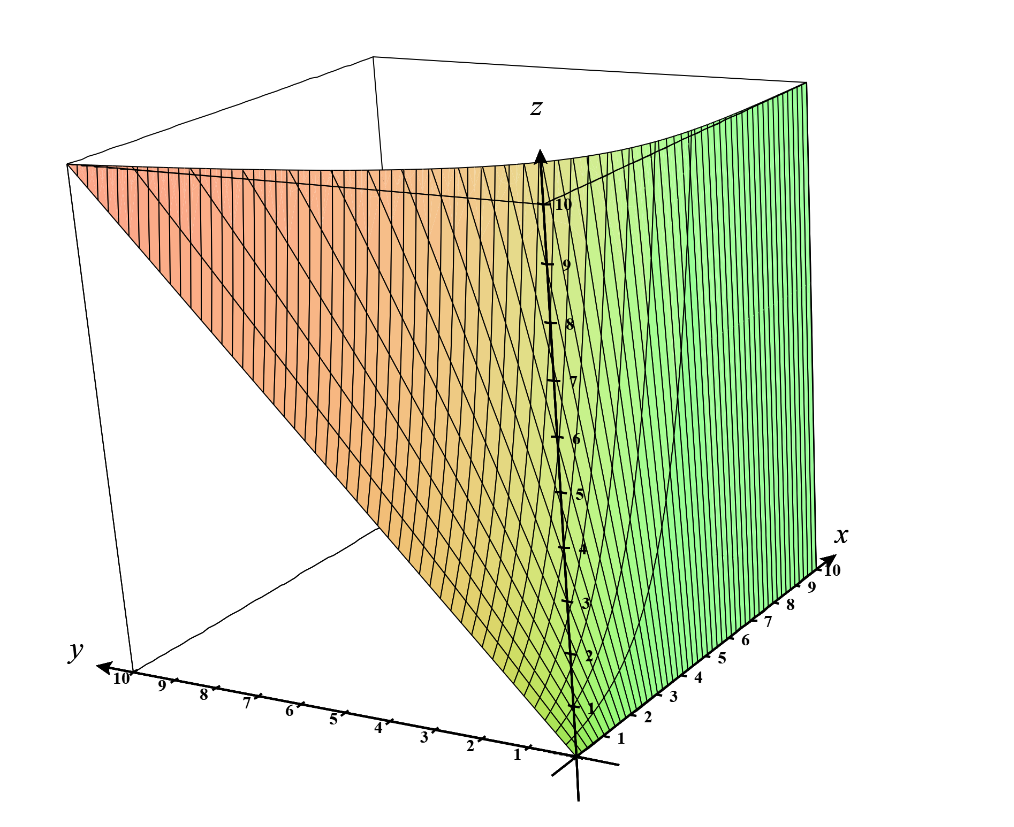
La performance de cet algorithme est donc de O(2^n), ce qui n’est pas très bon… La question intéressante cependant est : « À partir de combien d’entrées est-ce que l’application peine à tourner ? » Le nombre de sorties n’influent que très peu la complexité de l’algorithme, on peut le négliger dans les calculs. Malheureusement, il est difficile de créer les components avec plus de 20 entrées (pas assez de RAM disponible), sous risque de faire crasher l’application…

Une solution envisageable aurait été de ne pas stocker la table de vérité sous forme de tableau mais sous forme de long *string.* Cela aurait fait virtuellement sauté la limite à 20 entrées.

Par contre, en terme de simulation, je n’ai pas réussi à faire baisser les performances de l’application, même avec le schéma ci-dessous qui est un agrégat de composants prenant 20 inputs et donnant un output.

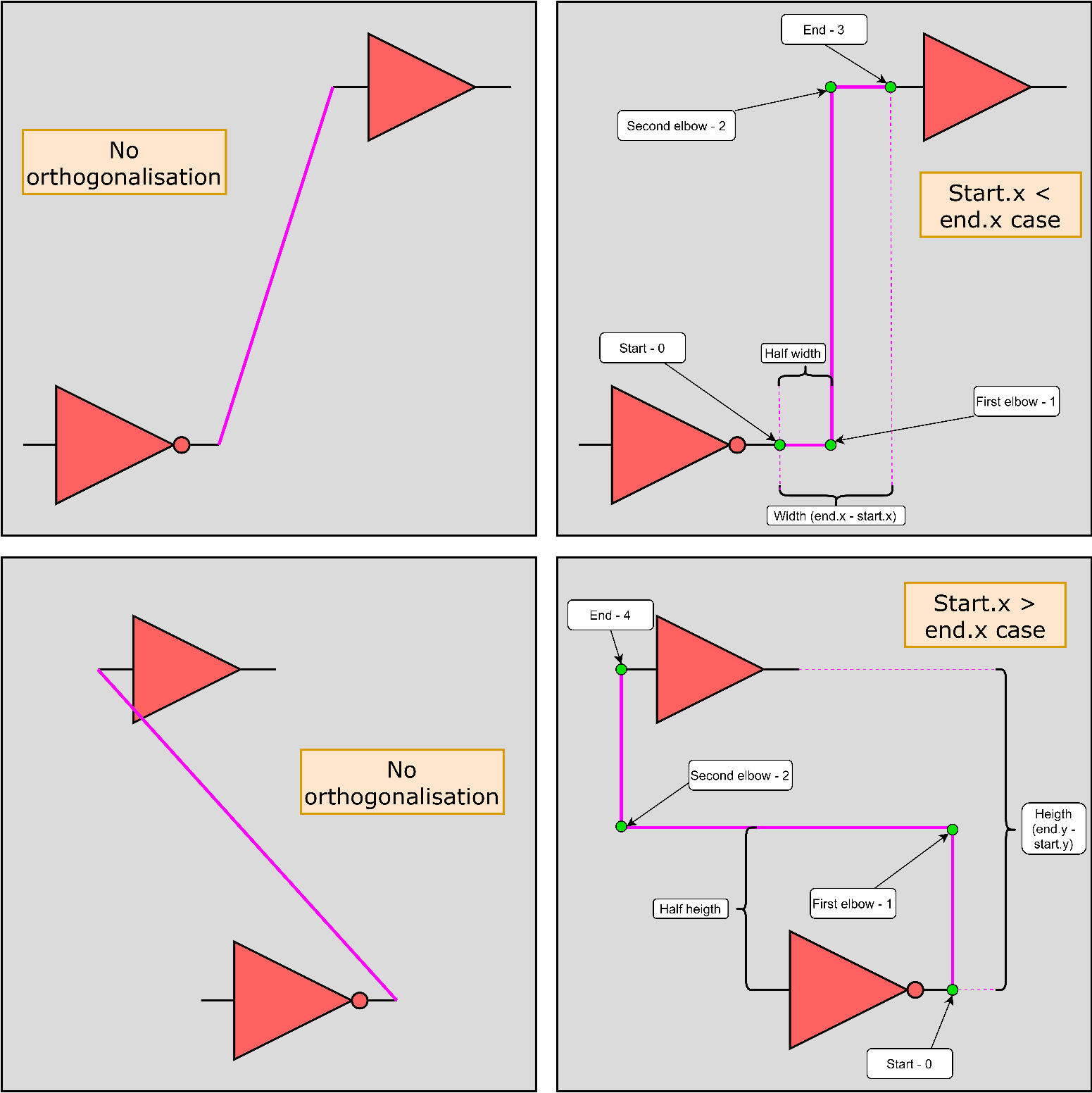


Ci-dessous le graphique 3D de l’évolution de la fonction



Câbles

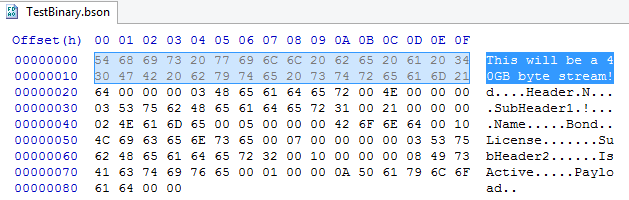
Les câbles relient deux IO entre eux. Ils adoptent automatiquement une forme orthogonale afin d’obtenir un rendu plus facilement lisible. Le *Line Renderer* permet de changer dynamiquement et facilement les propriétés du câble comme la couleur, l’épaisseur ou alors la position. On peut également utiliser un dégradé pour colorer le câble ; c’est pratique pour afficher visuellement la « progression » du signal dans ceux-ci. Ci-dessous un schéma expliquant comment fonctionne l’orthogonalisation des câbles (ne pas confondre avec l’orthogonalisation vectorielle via la méthode de Gram-Schmidt).

L’algorithme d’implémentation est très simple. Il suffit de prendre la position latérale entre l’IO de départ et celui d’arrivée et d’en faire la différence. Si le premier est plus grand que le second, on effectue une orthogonalisation sur l’axe Y, sinon sur l’axe X. Lorsque l’on déplace une porte reliée à une autre, la transition entre les deux types d’orthogonalisation se fait sans qu’on la remarque parce que pour passer de l’un à l’autre on passe par un stade où la différence vaut 0.

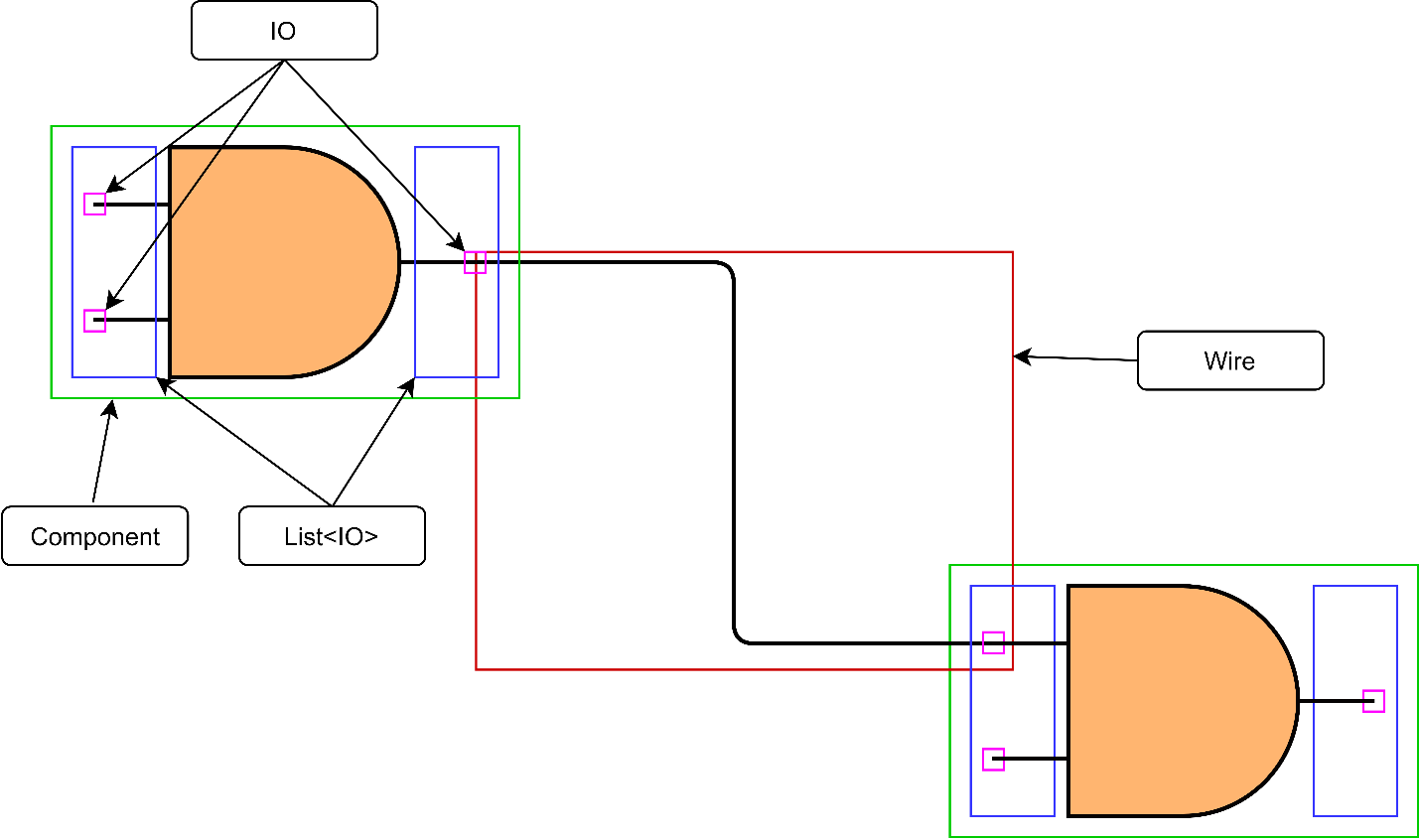
Export en PNG

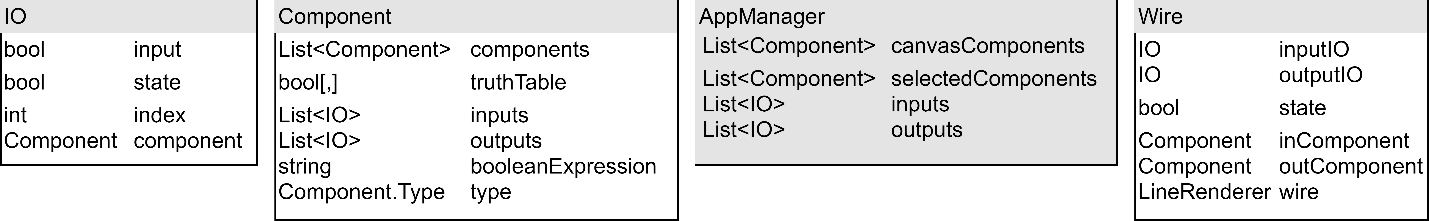
Pour exporter une image des schémas, je ne me casse honnêtement pas trop la tête… *Unity* intègre une classe nomée « ScreenCapture.cs » qui contient une méthode appelée « CaptureScreenshot » qui fait quasiment tout le boulot à ma place.

À la base, j’avais envisagé prendre chaque pixel rendu par la caméra, le transposé sur une texture que j’aurais ensuite stockée sur un *array* de *bytes* et encodé en PNG avant de l’exporter. Mais n’ayant pas de temps à perdre, j’ai plutôt utilisé la fonction intégrée d’Unity.



Synergie câbles – components

Un des premiers problèmes sérieux rencontré lors du développement de l’application, c’est le fait que les câbles ne se déplaçaient en même temps que les *components*, ce qui pose un problème de taille. Après avoir essayé plusieurs méthodes de fonctionnement différentes, une solution a finalement été trouvée. La première solution envisagée était bonne sur le concept mais très complexe dans les faits, rendant la solution difficile à débugger. Voici un schéma représentant le fonctionnement premièrement envisagé : 



Explication des classes :

**IO**

* Input : booléen permettant de savoir si ce *script* est attaché sur un *component* en tant qu’*input* ou en tant qu’*output*.
* State : état de la broche. Soit allumé (reçoit ou envoie le signal « true ») soit éteint (reçoit ou envoie le signal « false »)
* Index : donne la position du *script* dans la liste
* Component : indique quel est le *component* auquel est relié cet IO. Valeur « null » si pas relié

**Component**

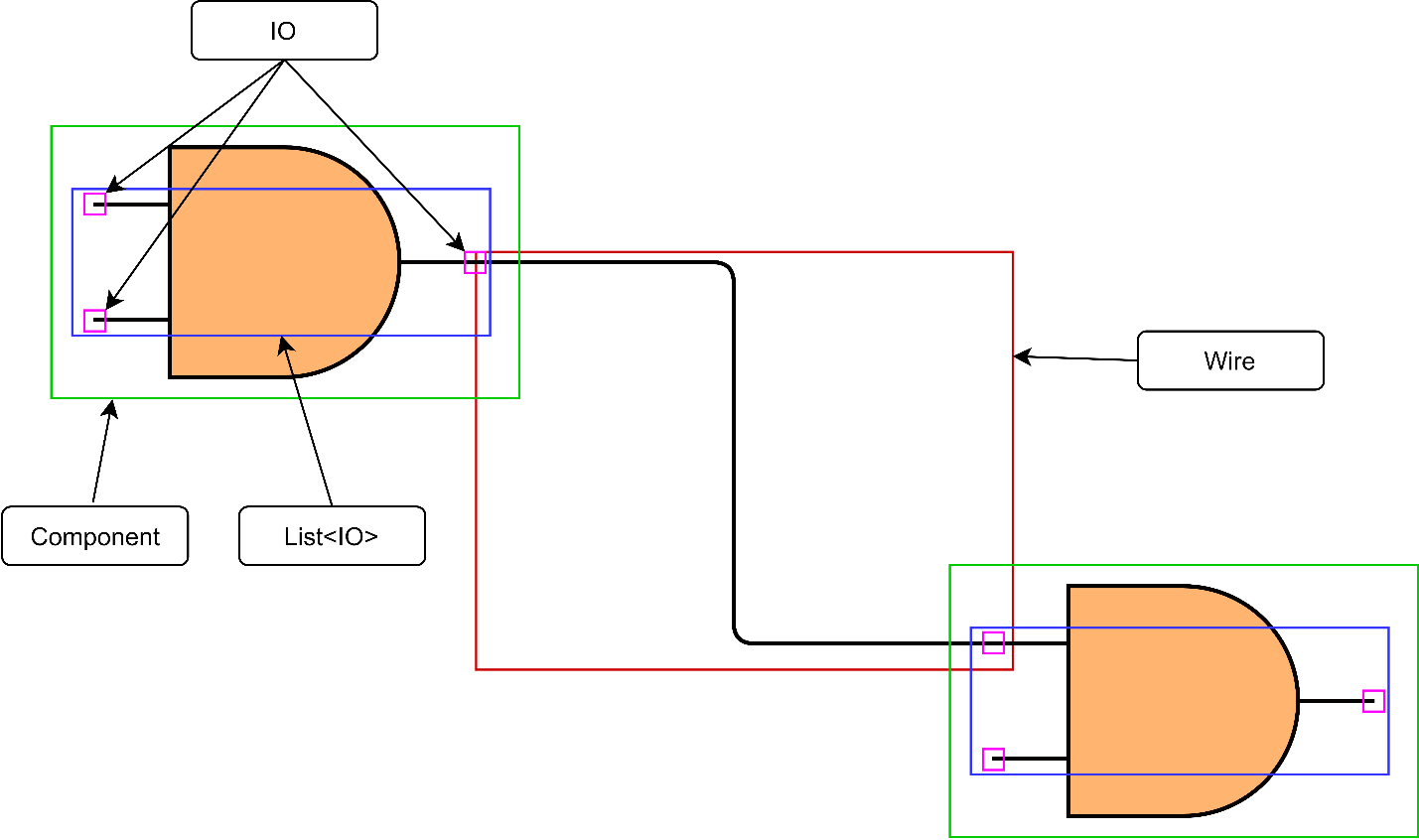
* Components : Liste de tous les composants qui compose ce composant. Cela permet d’avoir des *components* fait par d’autres *components*, et ce de manière récursive.
* Truth Table : tableau en deux dimensions permettant de stocké la table de vérité du composant.
* Inputs : liste de tous les *scripts* IO jouant le rôle d’*inputs* sur ce composant
* Outputs : liste de tous les *scripts* IO jouant le rôle d’*outputs* sur ce composant
* Boolean Expression : expression booléenne sous forme de texte représentant le composant
* Type : donne le type du composant. C’est-à-dire, soit une *porte logique* soit un composant « Custom »

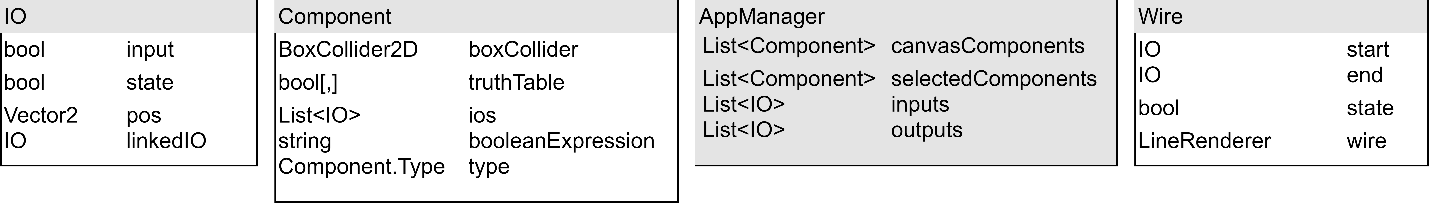
**AppManager :**

* Canvas Components : liste contenant tous les *components* sur le canevas
* Selected Components : liste contenant tous les *components* actuellement sélectionnés
* Inputs : liste de toutes les entrées du canevas
* Outputs : liste de toutes les sorties du canevas

**Wire :**

* Input IO : référence vers l’IO relié en entrée
* Output IO : référence vers l’IO relié en sortie
* State : état du câble (allumé étant « true » et éteint étant « false »)
* In Component : composant relié à l’entrée du câble
* Out Component : composant relié à la sortie du câble
* Wire : un *Line Renderer* est un composant *Unity* permettant de dessiner des lignes. Il y a beaucoup de paramètres utiles

Après avoir constaté l’inefficacité de ce modèle, j’ai décidé de supprimer les IO et de directement relier les *components* entre eux. Je me suis dit que ça allait simplifier le tout. Grave erreur ! en effet ne pas avoir de *classe* « parapluie » qui stocke toutes les infos relatives à une entrée ou une sortie signifie qu’il est obligatoire de stocker toutes ces informations dans la classe *component* directement. Après avoir essayé, sans succès, j’ai décidé de réintégrer la classe « IO » au système mais en simplifiant un peu la synergie entre les différentes classes. Voici un schéma montrant le nouveau fonctionnement ainsi que les nouvelles *classes* et leurs *attributs* : 



Explication des classes :

Je ne vais pas réexpliquer ce qui l’a déjà été par soucis de concision.

**IO :**

* Pos : position du *GameObject* sur la *scène* auquel le *script* est attaché
* Linked IO : référence à l’IO auquel celui-ci est lié. Valeur nulle s’il n’est pas relié

**Component :**

* Box Collider : *collider* correspondant à la taille de la *porte logique*
* IOS : liste de toutes les entrées et sorties du composant

**Wire :**

* Start : référence vers l’IO relié en entrée
* End : référence vers l’IO relié en sortie

Analyse du journal de travail

| Analyse du journal de travail | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Date** | **Début** | **Fin** | **Durée** | **Type** | **Tâche** |
| 1 | 03.05.2021 | 08:08 | 08:13 | 00:05 | Analyse | Découverte du CDC |
| 2 | 03.05.2021 | 08:14 | 08:25 | 00:11 | Analyse | Mise en place du GIT, début d'analyse |
| 3 | 03.05.2021 | 08:30 | 08:58 | 00:28 | Misc. | Discussion sur le CDC avec Mr. Melly et Mr. Gehrig |
| 4 | 03.05.2021 | 09:00 | 09:35 | 00:35 | Analyse | Création de la planification initiale |
| 5 | 03.05.2021 | 09:55 | 10:18 | 00:23 | Analyse | Création de la planification initiale |
| 6 | 03.05.2021 | 10:19 | 10:41 | 00:22 | Documentation | Création du projet Unity et implémentation du GIT |
| 7 | 03.05.2021 | 10:42 | 10:45 | 00:03 | Documentation | Création du fichier Word pour le rapport |
| 8 | 03.05.2021 | 10:46 | 11:17 | 00:31 | Analyse | Mise en place du Kanban sur GitHub et des sprints |
| 9 | 03.05.2021 | 11:18 | 11:31 | 00:13 | Analyse | Recherche d'informations sur le sujet |
| 10 | 03.05.2021 | 11:32 | 12:16 | 00:44 | Documentation | Documentation du projet |
| 11 | 03.05.2021 | 13:30 | 15:06 | 01:36 | Documentation | Difficultés attendues et Glossaire |
| 12 | 03.05.2021 | 15:25 | 15:32 | 00:07 | Documentation | Finition du glossaire |
| 13 | 03.05.2021 | 15:33 | 16:03 | 00:30 | Documentation | Mise en forme du rapport et table des matières |
| 14 | 03.05.2021 | 16:04 | 16:52 | 00:48 | Documentation | Planifaication initiale et Début du dossier de conception |
| 15 | 04.05.2021 | 08:05 | 08:32 | 00:27 | Documentation | Convention de nommage, versioning |
| 16 | 04.05.2021 | 09:07 | 09:35 | 00:28 | Conception | Maquettes |
| 17 | 04.05.2021 | 10:00 | 12:15 | 02:15 | Conception | Maquettes |
| 18 | 04.05.2021 | 13:30 | 14:08 | 00:38 | Conception | Palette de couleur |
| 19 | 04.05.2021 | 14:10 | 14:43 | 00:33 | Documentation | Documentation des maquettes |
| 20 | 04.05.2021 | 14:44 | 15:00 | 00:16 | Documentation | Planification détaillée en sprints et tâches |
| 21 | 04.05.2021 | 15:30 | 16:05 | 00:35 | Documentation |  |
| 22 | 06.05.2021 | 08:05 | 09:17 | 01:12 | Misc. | Discussion avec le chef de projet |
| 23 | 06.05.2021 | 09:17 | 09:36 | 00:19 | Documentation | Mise en forme |
| 24 | 06.05.2021 | 09:56 | 12:15 | 02:19 | Documentation | Complétion du glossaire, de l'analyse et de la conception |
| 25 | 06.05.2021 | 13:32 | 16:50 | 03:18 | Conception | Refonte des maquettes, flowcharts |
| 26 | 07.05.2021 | 08:03 | 11:25 | 03:22 | Réalisation | Lancement du projet Unity |
| 27 | 10.05.2021 | 08:18 | 09:35 | 01:17 | Réalisation | Développement de l'interface |
| 28 | 10.05.2021 | 09:51 | 12:15 | 02:24 | Réalisation | Développement de l'interface |
| 29 | 10.05.2021 | 13:30 | 15:05 | 01:35 | Réalisation | Développement des portes |
| 30 | 10.05.2021 | 15:20 | 16:55 | 01:35 | Réalisation | Développement des portes |
| 31 | 11.05.2021 | 08:03 | 09:35 | 01:32 | Réalisation | Développement des portes |
| 32 | 11.05.2021 | 09:52 | 12:12 | 02:20 | Réalisation | Développement des câbles |
| 33 | 11.05.2021 | 13:30 | 15:06 | 01:36 | Réalisation | Développement des câbles |
| 34 | 11.05.2021 | 15:20 | 16:05 | 00:45 | Réalisation | Développement des câbles |
| 35 | 17.05.2021 | 08:15 | 08:52 | 00:37 | Conception | Reconsidération de la logique programmatoire inhérente au projet |
| 36 | 17.05.2021 | 08:52 | 09:35 | 00:43 | Réalisation | Fixing du code. Recréation de la logique IO |
| 37 | 17.05.2021 | 09:52 | 10:29 | 00:37 | Réalisation | Logique des câbles |
| 38 | 17.05.2021 | 10:30 | 12:15 | 01:45 | Réalisation | Logique des câbles |
| 39 | 17.05.2021 | 13:32 | 14:45 | 01:13 | Réalisation | Refactoring du code |
| 40 | 17.05.2021 | 14:45 | 15:15 | 00:30 | Misc. | Visite de Mr. Gehrig |
| 41 | 17.05.2021 | 15:20 | 16:50 | 01:30 | Réalisation | Refactoring du code |
| 42 | 18.05.2021 | 08:02 | 08:42 | 00:40 | Réalisation | Fin de refacotring du code. |
| 43 | 18.05.2021 | 08:42 | 09:35 | 00:53 | Réalisation | Photoshoppage de tous les composants |
| 44 | 18.05.2021 | 09:53 | 12:15 | 02:22 | Réalisation | Adaptation des SVG pour Unity |
| 45 | 20.05.2021 | 08:01 | 08:15 | 00:14 | Réalisation | Débugging des câbles |
| 46 | 20.05.2021 | 08:15 | 08:55 | 00:40 | Misc. | Discussion avec Mr. Viret |
| 47 | 20.05.2021 | 09:00 | 09:36 | 00:36 | Documentation | Début du dossier de réalisation |
| 48 | 20.05.2021 | 09:55 | 12:15 | 02:20 | Documentation | Rédaction d'un certain nombre de problèmes rencontrés |
| 49 | 21.05.2021 | 08:00 | 09:35 | 01:35 | Réalisation | Suppression des câbles et amélioration de la visibilité |
| 50 | 21.05.2021 | 09:50 | 11:02 | 01:12 | Réalisation | Inputs et outputs du canevas |
| 51 | 21.05.2021 | 11:03 | 11:26 | 00:23 | Réalisation | synérgie canevas et inputs |
| 52 | 25.05.2021 | 08:06 | 08:58 | 00:52 | Réalisation | Fin de la synérgie entre canevas et inputs |
| 53 | 25.05.2021 | 08:58 | 09:19 | 00:21 | Réalisation | Gestion de la caméra et épaisseur dynamique des câbles |
| 54 | 25.05.2021 | 09:20 | 10:15 | 00:55 | Réalisation | Gestion des outputs du canevas |
| 55 | 25.05.2021 | 09:50 | 10:15 | 00:25 | Réalisation | Gestion des outputs du canevas |
| 56 | 25.05.2021 | 10:15 | 11:24 | 01:09 | Réalisation | Simulation des portes logiques |
| 57 | 25.05.2021 | 11:24 | 11:41 | 00:17 | Réalisation | Réparation des dégâts causés par Jonas |
| 58 | 25.05.2021 | 11:41 | 12:10 | 00:29 | Réalisation | Correction de divers bugs liés aux câbles et à leur updates |
| 59 | 25.05.2021 | 13:30 | 15:05 | 01:35 | Réalisation | Correction de divers bugs liés aux câbles et à leur updates |
| 60 | 25.05.2021 | 15:20 | 16:05 | 00:45 | Réalisation | Dossier de réalisation |
| 61 | 27.05.2021 | 13:30 | 15:05 | 01:35 | Réalisation | Management des composants dans le menu |
| 62 | 27.05.2021 | 15:20 | 16:55 | 01:35 | Réalisation | Sauvegarde et composants |
| 63 | 28.05.2021 | 08:05 | 08:20 | 00:15 | Tests | Tests divers |
| 64 | 28.05.2021 | 08:20 | 08:50 | 00:30 | Misc. | Discussion avec Mr. Viret |
| 65 | 28.05.2021 | 08:50 | 09:30 | 00:40 | Réalisation | Sérialisation des components |
| 66 | 28.05.2021 | 09:50 | 11:25 | 01:35 | Réalisation | Sérialisation des components |
| 67 | 31.05.2021 | 08:03 | 09:35 | 01:32 | Réalisation | Sérialisation des components |
| 68 | 31.05.2021 | 09:50 | 10:16 | 00:26 | Réalisation | Sérialisation des components |
| 69 | 31.05.2021 | 10:16 | 11:48 | 01:32 | Réalisation | Truth table computing |
| 70 | 31.05.2021 | 11:48 | 12:15 | 00:27 | Réalisation | Components instantiation |
| 71 | 31.05.2021 | 13:30 | 14:46 | 01:16 | Réalisation | Custom components simulation |
| 72 | 31.05.2021 | 14:47 | 16:06 | 01:19 | Réalisation | Small quality of life tweaks |
| 73 | 31.05.2021 | 16:06 | 16:34 | 00:28 | Documentation | Implementing Doxygen |
| 74 | 01.06.2021 | 08:05 | 09:24 | 01:19 | Réalisation | Delete et Edit des components dans le menu |
| 75 | 01.06.2021 | 09:24 | 09:27 | 00:03 | Réalisation | Taille du rond de selection automatiquement ajustée à la camera |
| 76 | 01.06.2021 | 09:28 | 09:35 | 00:07 | Réalisation | Travail sur la up bar |
| 77 | 01.06.2021 | 09:55 | 12:08 | 02:13 | Réalisation | Travail sur la up bar |
| 78 | 01.06.2021 | 12:08 | 12:15 | 00:07 | Réalisation | PNG export |
| 79 | 01.06.2021 | 13:30 | 14:23 | 00:53 | Réalisation | Pimp de l'appli |
| 80 | 01.06.2021 | 14:23 | 15:06 | 00:43 | Réalisation | Autres options de la up bar |
| 81 | 01.06.2021 | 15:25 | 15:58 | 00:33 | Documentation | Commentaires |
| 82 | 03.06.2021 | 08:03 | 08:19 | 00:16 | Réalisation | Correction d'un bug empêchant de charger les portes logiques lors du premier lancement de l'app |
| 83 | 03.06.2021 | 08:19 | 08:43 | 00:24 | Réalisation | Retour avec Mr. Viret et Mr. Evangelastique |
| 84 | 03.06.2021 | 08:43 | 09:23 | 00:40 | Réalisation | Finished commenting the code. |
| 85 | 03.06.2021 | 09:23 | 09:35 | 00:12 | Documentation | Documentation |
| 86 | 03.06.2021 | 09:53 | 12:12 | 02:19 | Documentation | Documentation |
| 87 | 03.06.2021 | 12:12 | 12:15 | 00:03 | Tests | Création tests unitaires |
| 88 | 03.06.2021 | 13:30 | 13:59 | 00:29 | Tests | Création tests unitaires |
| 89 | 03.06.2021 | 13:59 | 16:50 | 02:51 | Documentation | Peaufinage du dossier |
| 90 | 04.06.2021 | 08:00 | 09:35 | 01:35 | Documentation | Peaufinage du dossier et impression |

Et ci-dessous figure le résumé de la répartition des heures et les totaux.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Étiquettes de lignes** | **Analyse** | **Conception** | **Documentation** | **Misc.** | **Réalisation** | **Total général** |
| **Total général** | **00:01:58** | **00:07:16** | **00:19:33** | **00:03:20** | **02:07:03** | **03:15:10** |
| **Total en heure** | **1.97h** | **7.27h** | **19.55h** | **3.33h** | **55.05h** | **87.17h** |

Ayant été rigoureux sur la tenue du journal de travail, je suis plutôt assez confiant sur la qualité de ce résumé. Je pense que le temps a été bien utilisé.

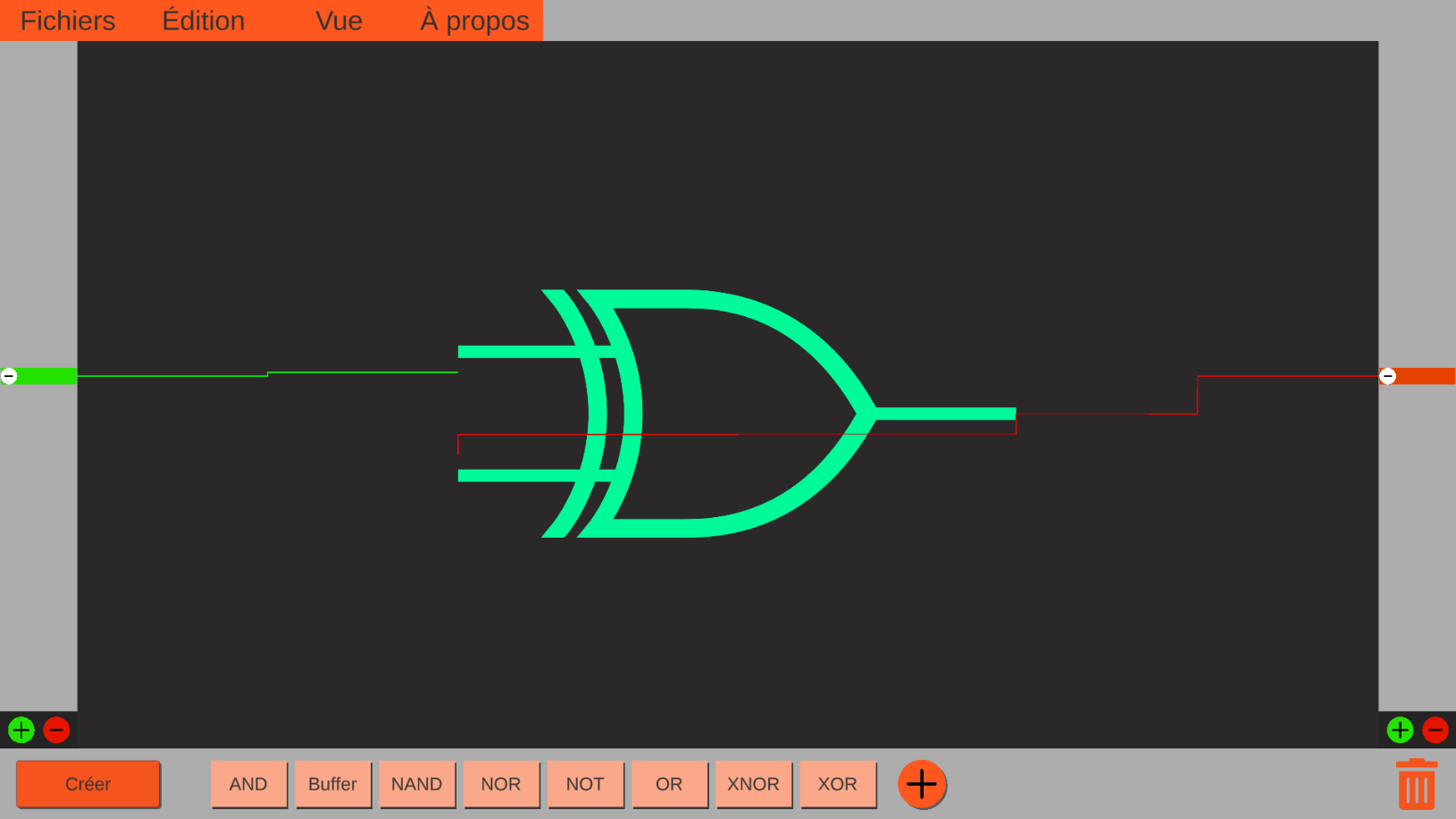
Problèmes rencontrés

Rétrospective

Erreurs restantes

Les circuits auto-dépendants :

Pour l’instant, l’application ne supporte pas les circuits bouclant sur eux-mêmes, c’est-à-dire des circuits dont l’état de certains éléments de celui-ci dépendent de leur propre état. Voici un exemple :



Ici, la sortie devrait « clignoter » car dès lors que la sortie de la porte est allumée, elle s’annule directement, puis se rallume, etc..

Dans tous les cas testés, en cas de « clignotement », c’est l’état *false* qui est retenu.

Il est à noter que l’application est tout de même protéger de la majorité des conflits de ce genre. Dans l’algorithme de calcul à partir des tables de vérité, il y a une liste des composants déjà « traités » et une limite au nombre d’opérations à effectuer par frames (empêchant donc une boucle de récursivité infinie).

L’export des schémas en PNG

Je n’ai pas réussi à faire en sorte que les fichiers exportés ne prennent que le schéma et omettent l’interface. J’ai essayé de désactivé l’interface juste avant de faire la capture d’écran et de la réactiver après mais rien à faire, cela ne fonctionne pas. Pourtant, j’ai même manuellement demandé le rendu de la caméra juste après avoir désactivé l’interface…

Je suis sûr que la solution à ce problème n’est pas sorcier, il faut juste que je prenne un peu de temps pour me pencher dessus…

Bug lors de la suppression d’éléments

Parfois, lorsque l’on glisse un élément sur la barre en bas, celui-ci ne se supprime pas comme il devrait. Je ne sais pas exactement pourquoi cela fait ça mais je suppose que c’est parce que le curseur n’est pas placé correctement sur l’élément d’interface contenant le déclencheur de la poubelle. Je pense qu’un petit coup d’œil dans la documentation d’*Unity* à la section « GU I » et « Event triggers » devrait m’aider.

Bug lors du changement de couleur d’un component

Malheureusement, bien que l’interpréteur de couleur fonctionne correctement, les couleurs pseudo-aléatoires données par le programme écrase celle rentrée par l’utilisateur rendant donc impossible pour le moment de donner une couleur personnalisée à ses components.

Il est très probable que ce bug se résolve en mettant un « if » au bon endroit dans la création des components.

Pour aller plus loin

Afin d’approfondir le programme il y a deux fonctionnalités que je trouverais pertinentes à rajouter s’il fallait continuer son développement.

Sauvegarde des circuits en eux-mêmes

À l’heure actuelle, l’application est capable de sauvegarder des circuits au travers des components. Le problème c’est que les circuits « en-soi » ne sont pas stockés, c’est leur comportement (c’est-à-dire leur table de vérité) qui est sauvegardé. Il serait intéressant de permettre à l’utilisateur de sauvegarder non pas leur component mais leur schéma complet ; ils pourraient alors ensuite le rouvrir et continuer de travailler dessus.

Ajouts de composants divers

C’est déjà bien de pouvoir travailler avec les 8 portes logiques de base et de pouvoir créer ses propres composants à partir de ceux de base, mais cela reste limité. Voci les composants que je souhaiterai rajouter si d’aventure je devais continuer le développement de cette application :

* Une lampe / LED qui s’allumerait si on lui apporte un signal « true »
* Un affichage décimal qui prendrait du binaire en entrée
* Un çifteli qui jouerait des notes de musique en fonction des inputs apportés
* Un affichage 7-segments
* Un bouton (qu’on devrait laisser appuyer pour envoyer du courant)
* Un switch (interrupteur donnant du courant on/off)

Tests effectués

Documents fournis

Conclusions

Je trouve que ce projet est globalement une réussite. Il y a quelques points dont je suis un peu déçu, notamment les bugs restants et le fait que l’on ne puisse pas encore enregistrer ses schémas pour les continuer plus tard. J’aurais aimé pouvoir bosser encore quelques semaines sur l’application pour vraiment lui donner plus de fonctionnalités et un ressentit plus « professionnel ».

Annexes

Manuel d’utilisation

Archives du projet

1. [↑](#footnote-ref-1)