|  |
| --- |
| Eurêka ! |

# Table des matières

[1 Table des matières 2](#_Toc103669751)

[2 Analyse préliminaire 4](#_Toc103669752)

[2.1 Introduction 4](#_Toc103669753)

[2.2 Matériel à disposition 4](#_Toc103669754)

[2.3 Prérequis 4](#_Toc103669755)

[2.4 Objectifs 4](#_Toc103669756)

[2.5 Planification initiale 5](#_Toc103669757)

[2.6 Planification détaillée 5](#_Toc103669758)

[3 Analyse / Conception 7](#_Toc103669759)

[3.1 Analyse 7](#_Toc103669760)

[3.1.1 Uses cases et scénarios 7](#_Toc103669761)

[3.2 Concept 9](#_Toc103669762)

[3.2.1 Diagramme de flux 9](#_Toc103669763)

[3.2.2 Design de l’UI 13](#_Toc103669764)

[3.3 Stratégie de test 16](#_Toc103669765)

[3.3.1 Test Unitaire 16](#_Toc103669766)

[3.3.2 Tests d’acceptation 16](#_Toc103669767)

[*3.4* Risques techniques 17](#_Toc103669768)

[4 Réalisation 17](#_Toc103669769)

[4.1 Dossier de réalisation 17](#_Toc103669770)

[4.1.1 Scripts 17](#_Toc103669771)

[4.1.2 Dossier 17](#_Toc103669772)

[4.1.3 Scripts 18](#_Toc103669773)

[4.1.4 Version de windows 19](#_Toc103669774)

[4.1.5 Version de Unity 19](#_Toc103669775)

[4.1.6 Version de visual studio 20](#_Toc103669776)

[4.2 Description des tests effectués 20](#_Toc103669777)

[4.2.1 Tests d’acceptations 20](#_Toc103669778)

[4.2.2 Test unitaire 21](#_Toc103669779)

[4.3 Erreurs restantes 22](#_Toc103669780)

[4.4 Liste des documents fournis 22](#_Toc103669781)

[5 Conclusions 22](#_Toc103669782)

[6 Annexes 23](#_Toc103669783)

[6.1 Résumé du rapport du TPI / version succincte de la documentation 23](#_Toc103669784)

[6.2 Sources – Bibliographie 23](#_Toc103669785)

[6.3 Journal de travail 23](#_Toc103669786)

[**6.4** 23](#_Toc103669787)

[6.5 Manuel d'Installation 23](#_Toc103669788)

[6.6 Manuel d'Utilisation 23](#_Toc103669789)

[6.7 Archives du projet 23](#_Toc103669790)

[7 Glossaire 23](#_Toc103669791)

# Analyse préliminaire

## Introduction

Eurêka est une serious game de résolution de puzzle, créer avec le langage C# et le moteur graphique unity, il a pour but de proposer au joueur la possibilité de résoudre des puzzles sous la forme d’un schéma avec des portes logiques généré aléatoirement, le joueur devra deviner le résultat de tous le résultat afin de réussir le puzzle, plusieurs niveaux de difficulté s’offre à lui, il peut aussi créer son propre puzzle.  
Il a aussi la possibilité de prendre en photo son puzzle, afin de le sauvegarder sur le disque.

## Matériel à disposition

Liste de matériel physique et de logiciel mis à disposition

* 1 PC du CPNV
* Unity 2020
* Visual Studio 2020
* Visual Studio Code
* Suite office

## Prérequis

* Formation de base du CPNV
* Connaissances en POO
* Maitrise de Unity

## Objectifs

Ce chapitre énumère les objectifs du projet. L'atteinte ou non de ceux-ci devra pouvoir être contrôlée à la fin du projet. Les objectifs pourront éventuellement être revus après l'analyse.

Ces éléments peuvent être repris des spécifications de départ.

Les objectifs du projet étant :

* Résolution de puzzle

Le but étant que le joueur puisse résoudre le puzzle en devinant la sortie(résultat) du puzzle (Exemple : une porte AND avec en entrée 1 et 0, donne 0 comme sortie)

Puzzle Aléatoire

Création Aléatoire des puzzles, chaque porte est générée aléatoirement, un nombre fixe de porte est présent, qui change en fonction du niveau de difficulté choisi par le joueur : en mode facile il y’a 3 portes logique, en mode moyen, il y’a 9 portes logique, en mode difficile, il y’a 19 portes logique.

* Puzzle manuel

Création manuel des puzzles, avec la possibilité de choisir les logiques portes dans le jeu, comme pour le puzzle aléatoire, les portes sont déjà prédéfinie par la difficulté, ils ne peuvent être déplacée, par contre le

joueur peut choisir quelle

type de porte (AND, XOR, etc…), et par la suite, de complété le puzzle par la suite, comme un puzzle aléatoire

* Capture de puzzle

Permet de prendre en photo le puzzle courant, qui est sauvegarder dans sur le disque

* Théorie des portes logiques

Affiche sur l’écran comment les portes logique courante sur le puzzle courant du comment fonctionne, leur entrée, sortie, avec une table de vérité

## Planification initiale



## Planification détaillée











# Analyse / Conception

## Analyse

### Uses cases et scénarios

Afin de comprendre comment, chaque fonctionnalité doit être programmé, et aussi d’avoir un scénario type pour gérer les exceptions j’ai fait des uses cases et scénario, afin de pouvoir bien se mettre d’accord sur ce qu’il faut faire

#### Puzzle aléatoire

Un puzzle aléatoire est généré lors du lancement d’une partie en tant que puzzle aléatoire.

Si le jeu est en échec, c’est-à-dire que le résultat attendu, n’est pas trouvé par le joueur, le jeu ne peut pas se terminer, ce use case et scénario ne gère pas la résolution de puzzle. Â l’inverse, si le résultat attendu, est trouvé par le joueur, le jeu est considéré comme trouvé, et un nouveau puzzle est généré, sauf si le joueur décide de quitter le jeu ou de passer sur un puzzle manuel.

Si le joueur décide de fermer le jeu quand un puzzle est en cours, le jeu demande une confirmation, si le joueur décide quand même de quitter le jeu, le puzzle ne seras pas sauvegarder, et un nouveau puzzle seras généré au prochain lancement du jeu, s’il n’y a pas de jeu en cours, le jeu se ferme sans donner de confirmation.



1 cas d'utilisation : puzzle aléatoire

#### Puzzle Manuel

Un puzzle manuel est généré lors du lancement de la partie, les ports sont vides mais déjà présente, les liaisons sont aussi déjà présente.  
Le joueur peux donc choisir le type de portes (porte AND, OR, par exemple).

Après avoir confirmé son choix, le puzzle passe en mode résolution de puzzle, le joueur peux donc compléter son propre puzzle.

Une partie peux être considéré comme en échec si le joueur ne choisit pas toutes les portes logique (laisse une vide par exemple) et le jeu peux être considéré comme terminé, si toutes les portes ont été choisies et le joueur aille confirmer son puzzle. Comme pour les puzzles aléatoires, le jeu demande une confirmation, si une partie est en cours, et ne demande pas si le jeu est déjà terminé.



2 cas d'utilisation : puzzle manuel

#### Résolution de puzzle

C’est la partie la plus importante du jeu, c’est ce qui gère toute la partie de résolution des puzzles, aléatoire et manuel, le résultat que le joueur doit trouver, et soit 1 ou 0 (true ou false), il faut donc vérifier si la valeur est dans cette fourchette de données ou que la valeur entrée est bien du décimal, si ce n’est pas le cas, le résultat est considéré comme faux, et un erreur est du coup affichée, si le joueur ne rentre aucun résultat rien ne se passe, si le joueur trouve le bon résultat, le jeu se termine.



3 cas d'utilisation : résolution de puzzle

#### Capture d’écran

Le système de capture d’écran, permet la sauvegarde en photo du puzzle actuel, le jeu ne peux pas relire l’image pour charger une partie, cela est juste une photo.  
Si la théorie des portes logique, ou le menu démarrée est affiché, le bouton pour prendre en capture le jeu, n’est ni actif, ni visible, à l’inverse on peut y accéder sans soucis.

Si le jeu ne peut pas avoir accès au fichier de sauvegarde d’image pour quelconques raisons, le jeu retourne une erreur.



4 case d'utilisation : capture d'écran

#### Théories des portes logiques

A chaque lancement de partie, soit en lançant un puzzle aléatoire, ou après la création d’un puzzle manuel, le jeu affiche un jeu d’icône de portes logiques, et en appuyant sur une icône, un descriptif détaillé, leur icone, et une table de vérité sera possible d’afficher. Il est aussi possible de réafficher cette fenêtre en appuyant sur F1.



5 cas d'utilisation : Théorie des porte logique

## Concept

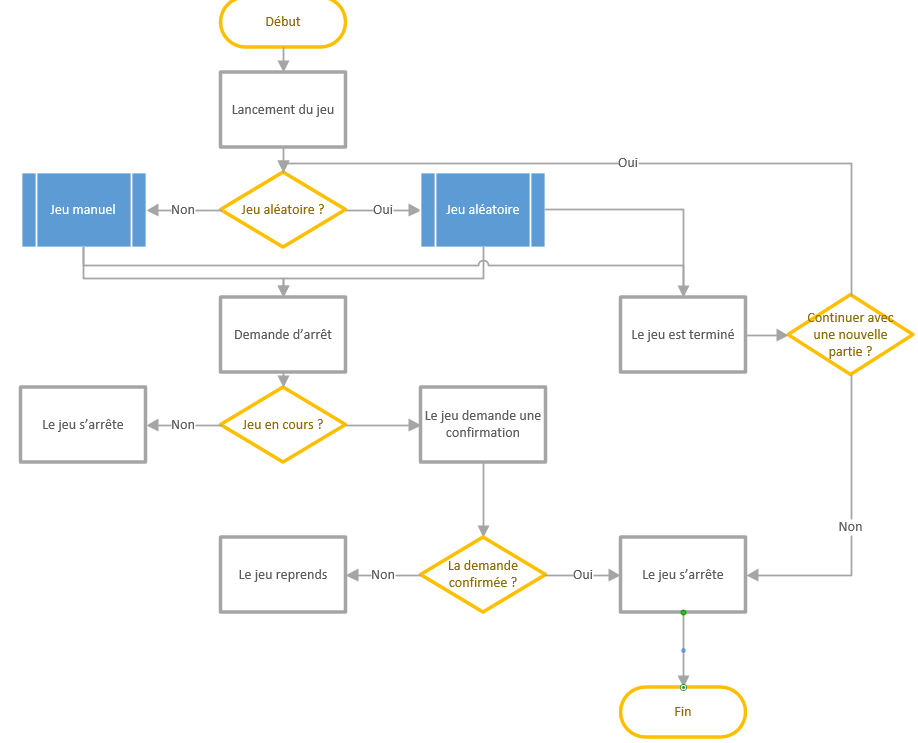
### Diagramme de flux

Pour avoir une idée plus précise du comment une fonctionnalité doit être implémentée, j’ai fait plusieurs diagrammes de flux, dont on peut voir comment les fonctions interagisse entre elles

#### Main

Le code main, est le script maitre qui gère l’appelle des autres fonctions, et gère aussi la fermeture du programme, le jeu commence toujours par demander si le joueur veut commencer par un puzzle manuel ou un puzzle aléatoire, et appelle la fonction qui gère le type de puzzle choisit. Le joueur peut choisir de continuer ou pas après chaque partie, si le joueur choisit de continuer, le jeu repose la question au joueur sur quel type de puzzle il veut, s’il ne veut pas, le jeu se ferme.

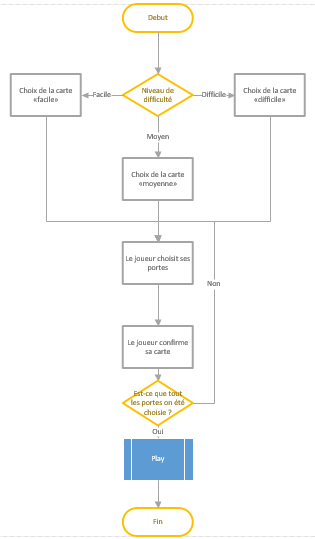
Il peut arriver que le joueur veut quitter le jeu, suivant le cas, si un puzzle est en cours, le jeu demande une confirmation au joueur. Au contraire, si aucun puzzle est en cours, le jeu se quitte sans demander de confirmations



6: Diagramme de flux global de l'application

#### Jeu Manuel

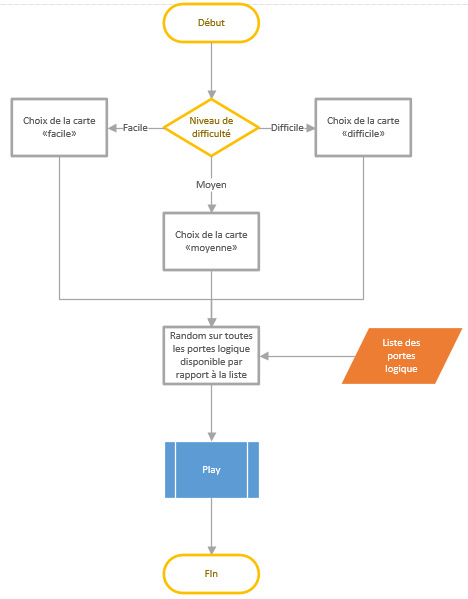
Si le joueur choisit un puzzle manuel, il doit aussi choisir la difficulté du jeu (un puzzle facile a 3 portes, moyen 9 portes et difficile 19 portes), les portes et fils sont déjà présent, le joueur peut juste choisir le type de portes par portes présente, si le joueur ne choisit pas tous ses portes, le jeu ne continue pas, à l’inverse s’il choisit toute ces portes, le jeu continue et appelle la fonction play.



7 Diagramme du puzzle manuel

#### Jeu aléatoire

Comme pour le puzzle manuel, le puzzle aléatoire commence par un choix de niveau de difficulté, avec le même nombre de portes, le jeu après sélection de la difficulté prend toute les portes du puzzle, et la liste de porte disponible et fait un random sur toutes les portes, et ensuite le joueur peu enfin participer au puzzle

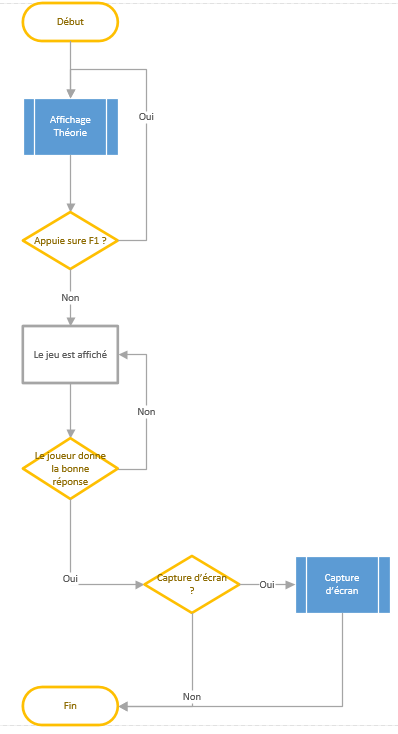


8 Diagramme du puzzle aléatoire

#### Play

Au début de chaque partie, la théorie sur chaque porte présente sur le puzzle courant est affiché, et peux être réafficher en appuyant sur F1.

Après, cela, le jeu est affiché, et tant que le joueur n’a pas trouvé la bonne réponse, le joueur ne pourras pas passer à autre chose, s’il trouve la bonne réponse, il peut prendre en photo le puzzle, et le jeu est ensuite considéré comme terminé et retourne au main.



9 diagramme de la résolution de puzzle

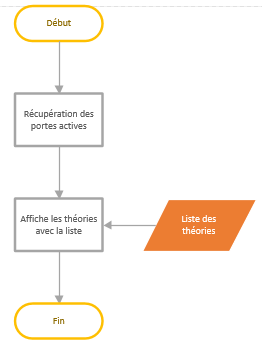
#### Capture d’écran

Permet la sauvegarde du puzzle courant par le biais d’une photo de ce dernier

#### Théorie des portes logique

Permet l’affichage de la théorie des port logiques

10 diagramme pour la théorie des portes logique



11 Diagramme pour la capture d'écran

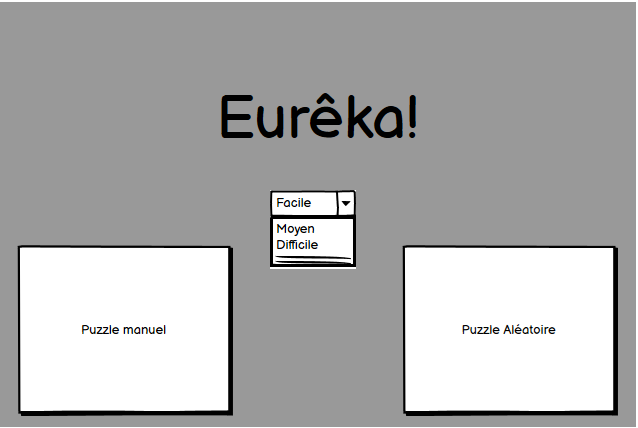
### Design de l’UI

Afin d’avoir une idée de comment l’UI du serious game, va ressembler, il est important d’avoir un design à quoi on peut s’attacher pour plus tard designer l’UI du jeu.

L’UI est pensé pour être accessible par PC et téléphone, les boutons sont grands, facile à voir.

#### Menu principale/jeu suivant

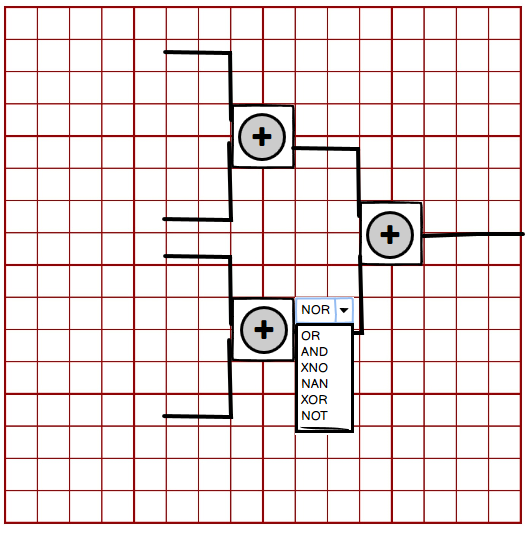
Cette « page » peut être utilisée pour le menu principale, mais aussi comme sélection de mode de jeu pour le prochain puzzle



12 maquette menu principale

#### Menu de création de puzzle

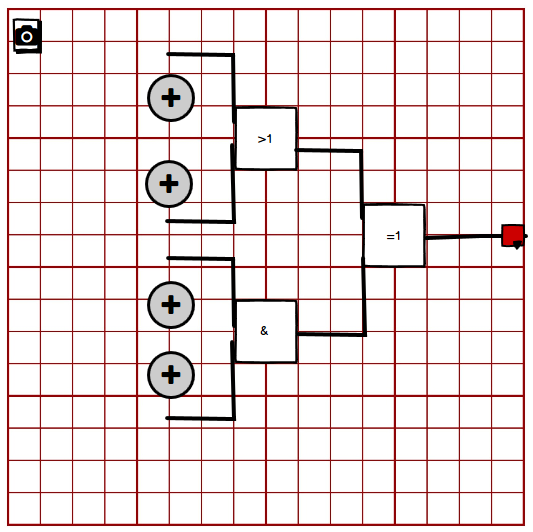
Le joueur aura la possibilité de créer son propre puzzle, les portes sont déjà présents, et le nombre de porte ne change que si le niveau de difficulté change, dans ce cas, le niveau de difficulté est facile, il peut, en cliquant sur la porte, changer de type de porte.



13 maquette : création de puzzle

#### Menu de résolution de puzzle

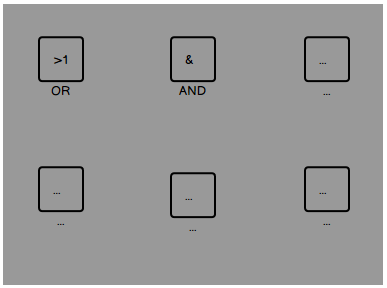
Après avoir terminé la création manuel d’un puzzle ou d’avoir choisi un puzzle aléatoire, le joueur devras compléter le puzzle, si le joueur choisit la bonne réponse, la sortie passe au vert, et le joueur à gagner



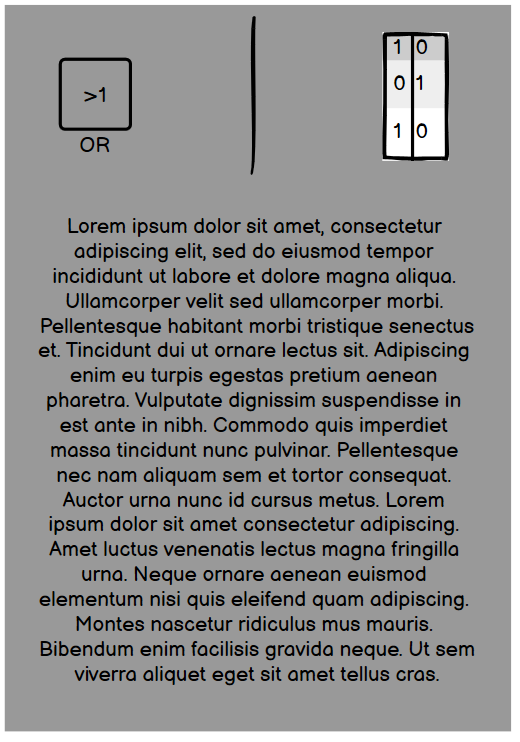
14 maquette : puzzle aléatoire

#### Théorie des portes logiques

Un choix de portes logique seras affiché, en début de partie et en appuyant sur F1, en appuyant sur les différentes icônes de la première fenêtre, il est possible d’avoir un descriptif détaillé de la porte logique.



15 théorie : jeu d’icônes de porte logique



16 théorie détaillé des portes logique

## Stratégie de test

### Test Unitaire

Un test Unitaire seras effectué sur la fonctionnalité de la résolution de puzzle, afin de bien vérifier le bon fonctionnement de cette fonctionnalité, avec le module de tests proposer par visual studio.

### Tests d’acceptation

Plusieurs tests d’acceptation seront mis sur IceScrum, je demanderais à plusieurs camarades de classe et membres de ma famille pour tester le produit, et de vérifier que toutes les tests d’acceptation passent

## Risques techniques

### Risque techniques

Quelque chose dont je n’ai pas l’habitude de faire, du au faîte que j’ai très rarement pratiqué, étant les tests unitaires en C#, est j’en n’ai jamais fait avec tout l’infrastructure d’Unity

# Réalisation

## Dossier de réalisation

### Scripts

#### Classes

* LogicGate : une classe contenant des types de porte logique

#### Gates

* GateManager : le script gérant une porte logique, chaque porte logique à se script, il gère entrées et sorties
* CustomGateManager : le script gérant la création des portes logique dans la création de puzzle

#### Puzzle

* CheckDifficulty : script qui prends en compte la difficulté choisit par le joueur, et choisis la bonne carte
* GlobalControl : script qui permet de garder en mémoire des variables, pour qu’elles ne soient pas perdue au changement de scène
* MapInfo : script qui donne les infos, de toutes les portes logique courante pour la difficulté choisie
* GeneratePuzzle : récupère toutes les portes logiques de la difficulté choisie et change toutes les port (sauf pour moyen et difficile ou la dernière porte est de tout de façon un AND), si le joueur a créer un puzzle auparavant, il génère le puzzle par rapport au choix du joueur
* ResolvePuzzle : Script qui gère la complétion du puzzle, si la dernière porte est en « Vert », le jeu retourne au menu principale, et il est possible de rechoisir un nouveau puzzle.
* CustomPuzzle : gère la confirmation du choix du joueur pour la création du puzzle, et sauvegarde le tout dans GlobalControl

#### UI

* MainMenu : script qui gère le menu démarré, notamment le changement de scène et la difficulté
* Zoom : script qui permet de zoomer sur le puzzle

### Dossier

* Assets : répertoire principal, englobent tous les autres dossiers, fichier et script

#### Ressources

* GameObjects : répertoire contenant diffèrent gameobjects utile pour le projet.
* Prefabs : répertoire contenant les préfabs utilisé pour le projet
* Sprites : répertoire contenant tous les sprites du jeu

#### Scenes

* Répertoire contenant toutes les scènes du jeu

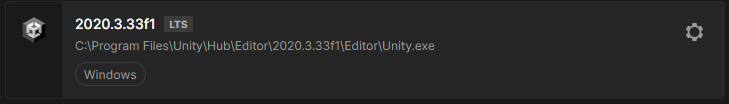
### Scripts

* Classes : répertoire contenant toutes les classes utilisé dans le jeu, notamment LogicGate
* Gates : répertoire avec les scripts gérant les portes logiques
* Puzzle : répertoire avec les scripts gérant les puzzles
* UI : répertoire contenant les scripts qui gère les UI

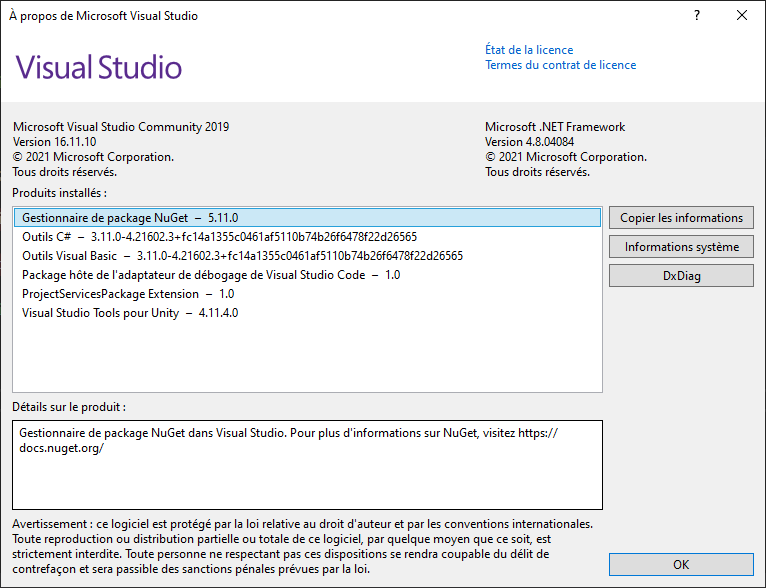
### Version de windows



### Version de Unity



### Version de visual studio



### Version du programme

* les répertoires où le logiciel est installé
* la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)
* les versions des systèmes d'exploitation et des outils logiciels
* la description exacte du matériel
* le numéro de version de votre produit !
* programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.

NOTE : Evitez d’inclure les listings des sources, à moins que vous ne désiriez en expliquer une partie vous paraissant importante. Dans ce cas n’incluez que cette partie…

## Description des tests effectués

### Tests d’acceptations

#### résolution du puzzle

* Les portes logiques fonctionnent correctement
  + Quand le joueur change l'entrée des portes, la sortie des portes logique réagisse correctement
* Les entrées changent quand je joueur veux
  + Si le joueur change l'entrée, le changement est pris en compte et affecte les portes logiques
* Le système détecte quand le puzzle est terminé
  + Quand le puzzle est réussi, et que la dernière porte logique est en vert, le jeu se termine correctement

#### création du puzzle aléatoire

* Le système change aléatoirement toutes les portes
  + Le système génère aléatoire les portes logiques, en prenant en compte les portes à 2 entrées et 1 entrée
* Le système génère aléatoirement à chaque fois
  + Le système génère aléatoirement, et ne régénère jamais le même puzzle

#### création du puzzle customisé

* Quand le joueur choisit sa porte logique la porte change
  + Le visuel change, et le système prend en compte
* Quand le joueur confirme son choix le système prend en compte le choix
  + Quand le joueur as terminé de créer son puzzle, et confirme son choix, le système passe en mode création, avec les portes logique à la bonne place, qui fonctionne correctement

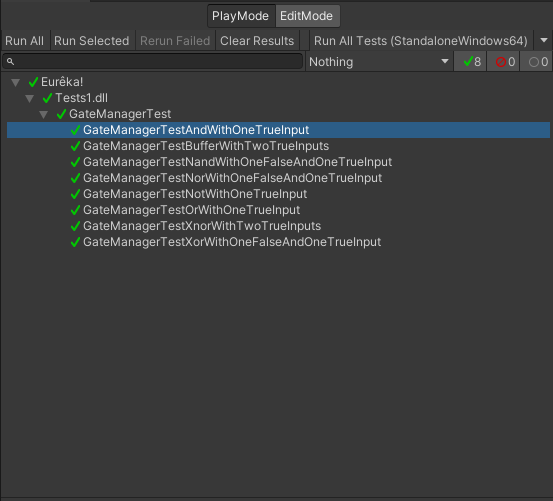
#### Création du système des théorie

* Les boutons réagissent correctement
  + Les boutons affichent et caches les bonnes dalles

#### système de capture d’écran

* Quand le bouton est pressé une photo est prise
  + Une photo est prise au moment du pressage du bouton, et le l’image est bien capturée
* Si pas de dossier existe, un dossier est créé
  + Si le joueur prend une photo et le dossier de photo n’est pas présent, le dossier est créé

### Test unitaire

Le test unitaire se porte sur le script de la gestion de porte logique, c’est-à-dire, le script, qui gère les entrées, et sortie des portes, le test est réalisé en C# sur Visual Studio.

#### GateManager Test And With One True Inputs

Test sur une porte AND avec une entrée TRUE, et qui retourne un FALSE, les entrées qui ne sont pas spécifié, sont de base false

#### GateManager Test Buffer with two true inputs

Test sur une porte Buffer, avec deux entrées TRUE, et qui retourne un TRUE, les portes logiques avec une seule entrée, ne prennent en considération seulement que la première entrée

#### GateManager Test Nand with one false and one true input

Test sur une porte NAND, avec une entrée TRUE et une entrée FALSE, et qui retourne un FALSE

#### Gatemanager test nor with one false inpute and one true input

Test sur une porte NOR, avec une entrée FALSE et une entrée TRUE, qui retourne un FALSE

#### GateManager Test Not with one true input

Test sur une porte NOT, avec une entrée TRUE, qui retourne un false

#### GateManager Test or with one true input

Test sur une porte OR, avec une entrée TRUE, qui retourne un TRUE

#### GateManager Test XNOR with two true inputs

Test sur une porte logique XNOR, avec deux entrée TRUE, qui retourne un TRUE

#### GateManager Test XOR with one fasle and one true input

Test sur une porte logique XOR, avec une entrée TRUE, et une entrée FALSE, qui retourne un TRUE

* les conditions exactes de chaque test
* les preuves de test (papier ou fichier)
* tests sans preuve: fournir au moins une description

## Erreurs restantes

S'il reste encore des erreurs:

* Description détaillée
* Conséquences sur l'utilisation du produit
* Actions envisagées ou possibles

## Liste des documents fournis

* Diagrammes de flux
* Journal de travail
* Maquettes
* Uses case et scénario
* Résumé du projet

# Conclusions

* Développez en tous cas les points suivants:
* Objectifs atteints / non-atteints
* Points positifs / négatifs
* Difficultés particulières
* Suites possibles pour le projet (évolutions & améliorations)

# Annexes

## Résumé du rapport du TPI / version succincte de la documentation

## Sources – Bibliographie

tutorialspoint. «Logic gates.» *tutorialspoint.* 16 05 2022. https://www.tutorialspoint.com/computer\_logical\_organization/logic\_gates.htm.

Unity. «Screenshot, but from a certain camera.» *Unity Forum.* 16 05 2022. https://forum.unity.com/threads/screenshot-but-from-a-certain-camera.69645/#:~:text=%20%20%201%20Disable%20all%20activated%20cameras,is%20done%2C%20you%20can%20switch%20cameras...%20More%20.

—. «Unity - Application.persistentDataPath.» *Unity documentation.* 16 05 2022. https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html.

—. «Unity - ScreenCapture.CaptureScreenshot.» *Unity Documentation.* 16 05 2022. https://docs.unity3d.com/ScriptReference/ScreenCapture.CaptureScreenshot.html.

Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)… Et de toutes les aides externes (noms)

## Journal de travail

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Durée | Activité | Remarques |
|  |  |  |  |
|  |  |  |  |

## 

## Manuel d'Installation

## Manuel d'Utilisation

## Archives du projet

Media, … dans une fourre en plastique

# Glossaire

A

AND

porte logique ET 3, 6, 16, 19

Assets

dossier maître dans unity 16

B

Buffer

porte logique entrée = sortie 19, 20

C

C#

Language de programmation 3, 19

F

FALSE

FAUX 19, 20

G

Gates

Portes logique 17

N

NAND

porte logique NON-ET 20

NOR

Porte logique NON-OU 20

NOT

Porte logique NON 20

O

OR

Porte logique OU 6, 20

P

POO

Programmation orientée object 3

R

random

aléatoire 10

T

TRUE

VRAI 19, 20

U

UI

User interface, interface graphique 12, 16, 17

Unity

Moteur graphique de jeu 17

V

Visual Studio

Application pour programmer 3, 19

X

XNOR

Porte logique NON-OU exclusif 20

XOR

Porte logique OU exclusif 3, 20