Le jeu de la Réformation

L’explication humoristique et algorithmique des critiques de Luther



Lucas Cranach the Elder, 1532

Table des matières

[Description du projet 3](#_Toc113032782)

[Notice d’installation : Comment le jeu fonctionne 6](#_Toc113032783)

[Choix des librairies 7](#_Toc113032784)

[Algorithme utilisé 9](#_Toc113032785)

[Structures de données 13](#_Toc113032786)

[Manuel utilisateur : Le guide d’utilisation 14](#_Toc113032787)

[Documentation du code 15](#_Toc113032788)

# Description du projet

Intéressé depuis toujours par la religion et ses différents mouvements, le master MIMO ne paraît pas être celui qui me permettrait de réfléchir sur ce sujet. Suivant des discussions intéressantes avec certains de mes collègues, l’idée de réaliser un projet centré autour d’une vision décalée, simpliste et humoristique de la Réforme protestante commencée officiellement par Luther en 1517 est devenue assez évidente. Elle n’intéressait pas pour autant mes camarades de travail, mais c’était un beau défi : malgré tous les rendus universitaires de cette année, un retard conséquent et une mauvaise organisation, l’objectif était devenu celui d’être capable de réaliser un code seul, tenant relativement la route en prenant en compte le contexte et la solitude. Nous préciserons par ailleurs que ce jeu n’a aucun but d’évangélisation, de discours théologique ou de débat politico-historique. Il se comprend simplement par un certain contexte qui se veut au mieux amusant, et de mon envie d’allier une thématique religieuse à un jeu-vidéo comme certains autres l’ont fait beaucoup mieux que moi (*The Binding of Isaac*, Edmund McMillen, Florian Himsl).

C’est tout ce contexte qui mène donc au jeu de la Réformation. Sa simplicité s’inscrit dans une vision assez absurde : on peut imaginer que Luther, sur son imprimerie toute nouvelle, se met à mettre en place un algorithme permettant d’expliquer le problème des indulgences. Le jeu propose donc une parodie d’un certain style de jeu bien connu, qu’on pourrait appeler un *Cookie Clicker-like*, qui reprend un principe simple : un clic signifie un objet en question qui est compté. Ces clics peuvent devenir une monnaie, ensuite utilisable pour améliorer d’autres éléments, ses propres clics, et au final finir le jeu. Concept simple certes, mais qui prend une toute nouvelle approche quand on prend en compte le contexte de notre jeu, et l’utilisation principalement de la librairie *turtle* en Python. L’utilisateur du monde moderne, contemporain de Luther, décontenancé par cette nouvelle technologie algorithmique, n’est pas pour autant déboussolé par le fait de cliquer sur un élément pour obtenir la ressource.

Pour se distinguer des autres jeux du même type : chaque *Cookie-Clicker-like* a son propre élément particulier, que ce soit au niveau graphique ou au niveau du *gameplay*. Pour notre cas, l’interface graphique est simple et suit de nombreuses connotations religieuses. Au niveau du *gameplay*, le jeu de la Réformation se distingue par le concept de « péchés » : peut-importe l’action de l’utilisateur, même une action ratée qui ne mène concrètement à rien pour l’utilisateur, augmentera le compteur de péchés. S’il est trop haut, l’utilisateur a perdu.

Le fait de travailler sur ce que l’on peut appeler un « *point-and-click* » ramène aux temps immémoriaux des premiers jeux-vidéos. S’ils fonctionnaient aussi bien au début des jeux-vidéos sur ordinateur, c’est par leurs dimensions intuitives : on voit un objet, celui-ci est mis en avant par rapport au reste du décor.

Notre *point-and-click* utilisant *turtle* s’accompagne également d’une partie secrète, à découvrir pour l’utilisateur s’il gagne le premier jeu, utilisant cette fois-ci la librairie *pygame*. Il représente la dernière action de l’utilisateur, afin d’atteindre la victoire finale, qui se solutionne à contre-courant. Nous en parlerons dans la suite de la documentation, en gardant le suspense dans la partie de la « Description du projet ».

Techniquement parlant, le projet allie donc deux jeux codés dans deux cadres de travail bien différents, l’un via *turtle*, l’autre via *pygame*. Plusieurs inspirations sont à comprendre afin d’observer la pensée et la démarche entreprise. Le jeu est notamment inspiré par la réalisation de Lixian réalisé pour l’anniversaire du YouTuber Markiplier[[1]](#footnote-1). Le second jeu est notamment inspiré par les très nombreuses réalisations de *snake* sur Python[[2]](#footnote-2).

Afin de mieux comprendre l’objectif du jeu, il semble nécessaire de faire une petite contextualisation historique. En 1517, le moine augustin Martin Luther cloue sur son église en Allemagne les 95 thèses, une critique acerbe du commerce des indulgences réalisé par les papes de l’époque. En échange d’un gros don monétaire, le pape distribuait en effet des indulgences, c’est-à-dire des pardons des péchés réalisés. Les personnes riches pouvaient alors, en échange d’un gros don à l’Eglise, payer directement leur voyage au Paradis, ce qui paraissait pour Luther une incroyable remise en question des dogmes de la religion catholique. Rappelons également que ce commerce des indulgences incluait des dons de terres par les seigneurs, notamment vers la fin de leurs vies, en échange d’un ticket au Paradis. Ces dons de terres ont rendu petit à petit l’Eglise et ses Etats très puissants et riches, ce qui est une critique de Luther. La mise en place du premier protestantisme officiel s’est donc centrée autour de cette critique de Luther et des autres théoriciens. Notre jeu et algorithme a pour objectif, de manière humoristique, de montrer comment Luther aurait expliqué le commerce des indulgences à notre époque avec une capacité de codage relative.

L’objectif du jeu est donc d’atteindre 10000 pardons, aucun péchés et 50 dons de terres, de vaincre le péché originel, afin d’être sûr d’avoir sa place au Paradis.

Le code du programme sera partagé sur la plateforme GitHub, permettant de mettre en place l’ensemble des fichiers nécessaires à son bon fonctionnement. Cela permettra également d’avoir un certain historique sur les avancées du code qui seront réalisées sur le long terme. En le partageant au public, on a également la possibilité d’avoir des retours sur des possibles avancées ou sur la qualité du code.

# Notice d’installation : Comment le jeu fonctionne

Pour le bon fonctionnement du jeu, il est nécessaire d’avoir plusieurs *packages* supplémentaires pour Python : la librairie *pygame* et la librairie *python-vlc*. Si l’utilisateur veut faire fonctionner la musique dans le jeu, détail esthétique secondaire, il est nécessaire d’avoir sur son pc le logiciel *vlc* et installer dans Python la librairie en lien avec le programme. Il faut aussi prendre en compte, dans le code lui-même, une modification nécessaire pour le bon fonctionnement de la musique : la ligne 12 doit être modifiée.

Lorsque l’utilisateur fait fonctionner le programme, un écran *turtle* apparaît. Celui-ci, nommé « Le commerce des indulgences vu par Luther ». C’est cette interface graphique que l’utilisateur doit comprendre et qui va le suivre tout au long du jeu. L’utilisateur n’a le choix que d’une seule action : cliquer. Un clic, dans le jeu, est censé représenter un jour. Si l’utilisateur clique sur un espace non *mappée*, c’est-à-dire sans action enregistrée, il n’obtiendra aucun effet positif, par contre, un jour sera quand même passé, donc des péchés seront enregistrés. En échange, l’endroit où il clique est la donnée importante. En fonction de la zone cliquée, et du nombre de « monnaies » disponibles, l’utilisateur peut soit obtenir « des pardons » en cliquant sur la croix, obtenir des bonus en fonction de ce qu’il clique, ou limiter les malus dont son principal ennemi : « les péchés ». En fonction de son choix, les différents compteurs en question sont mis à jour automatiquement. Ils sont au nombre de 4 : les pardons, l’information à collecter principale du jeu, les indulgences (ayant un impact sur le nombre de pardons par clics), représentant la monnaie principale, les dons, nécessaires pour gagner la partie, et les péchés, qu’il faut limiter afin de ne pas perdre la partie.

Si l’utilisateur obtient 1000 péchés ou plus, ou qu’il se met dans une situation de déficit trop élevé en indulgences ou en pardon, il aura perdu, et sera accueilli par un écran de défaite rassurant. A l’inverse, l’utilisateur doit réussir à gérer ses ressources et obtenir 10 000 pardons, 50 dons à l’Eglise, et aucun péché, il sera accueilli par un jeu classique de *snake*, représentant le péché originel. Si l’utilisateur arrive à vaincre cette dernière épreuve, il sera accueilli par la plus grande des réussites : un simple écran de victoire.

A la différence d’un *snake* classique, l’objectif n’est pas de manger la pomme, justement, c’est la représentation dudit péché. Foncer vers la pomme ne mène à rien. La seule façon de se sortir du jeu, c’est de sortir de l’écran, une sorte de représentation d’un dépassement dudit péché originel, menant à la victoire.

# Choix des librairies

Plusieurs librairies ont été choisies pour le bon fonctionnement du code et son optimisation. D’abord, le projet a utilisé la librairie classique de Python dans PyCharm, afin de faire tous les traitements classiques. Nommé ***pip***, il est nécessaire pour le bon fonctionnement d’un code Python et l’installation de nouveaux packages. La librairie ***setuptools*** est égalemet installée directement dans PyCharm, utile pour certains modules et packages Python. ***wheel*** également est présent, mais non utilisé.

Les autres *packages* installées sont notamment mis en avant via des imports au début du code. Ils sont externes à toutes les fonctions car utilisés dans l’ensemble du programme. Nous les décrivons ici par la suite :

* Le programme utilise principalement tout au long de son fonctionnement la librairie *turtle*, qui permet de créer toute sorte de choses via un écran. Ce package est utilisé massivement dans le cadre du premier jeu de notre programme. Il est utilisé dans la création d’un écran et d’une interface graphique sur laquelle l’utilisateur peut cliquer. Cet écran répond au choix de l’utilisateur en fonction de ce dernier, et via *turtle.mainloop*, ne s’arrête que lorsque l’utilisateur l’a décidé. La librairie *turtle*, concrètement permet de réaliser graphiquement n’importe quoi sur un plan d’une certaine taille définie par l’utilisateur. Ce dernier a une tortue, qu’il peut déplacer à des coordonnées x,y et former les formes de son choix. Cela permet de mettre en place un nombre infini de graphiques et de représentations.
* La librairie *random* est principalement utilisée à travers la méthode *random.randInt(x,y)*, permettant pour simplifier de donner un chiffre aléatoire, un *integer* entre x et y. Cette méthode est essentielle pour la mise en place aléatoire de la place de la pomme dans le jeu utilisant *pygame*, mais l’est également pour définir le nombre de péchés ajoutés aléatoirement lors de l’action de l’utilisateur.
* L’une des librairies installées expressément pour notre code se nomme ***pygame***. Celle-ci est reconnue comme étant l’une des principales librairies utilisées pour créer des jeux plus ou moins complexes en utilisant le langage Python. Elle nous permet dans notre code de mettre en place un jeu du *snake* classique et simple. On utilise un ensemble de méthodes présentes dans *pygame*, permettant l’initialisation d’un écran, la création du serpent et son déplacement, ainsi que la création de la pomme. La librairie est massive et peut être utilisée de très nombreuses manières pour la création d’un jeu-vidéo bien plus important que celui codé dans le programme.
* La librairie nommée ***vlc*** permet justement d’utiliser de manière efficace du code reprenant le logiciel français *vlc* pour *VideoLan Client Media Player*. Elle est utilisée afin de mettre en place la musique dans le jeu, que ce soit pendant le jeu en lui-même ou lors des écrans de victoire et de défaite. La longueur des musiques permet de ne pas se poser la question de potentielles boucles si la musique est terminée.

# Algorithme utilisé

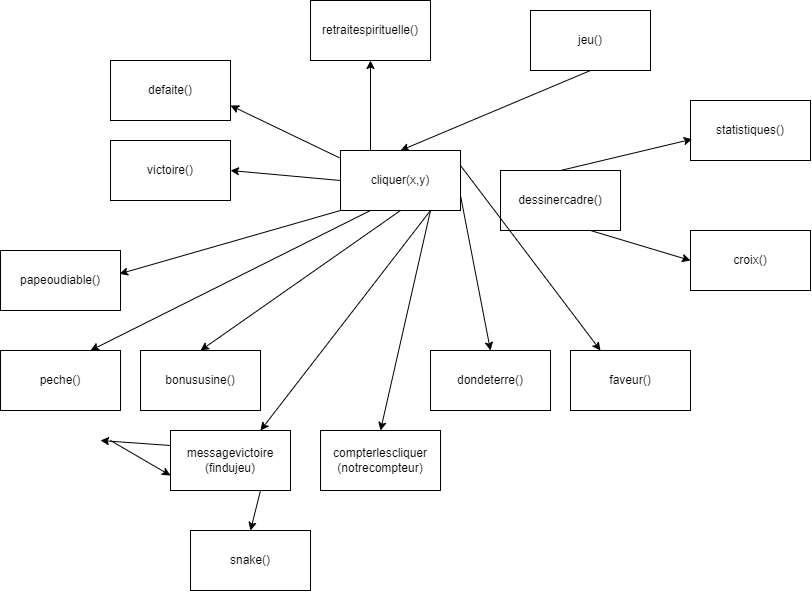
Pour faciliter le traitement, le code a utilisé de manière importante des variables globales, au lieu d’utiliser des *return* par exemple dans les fonctions. C’est un choix de facilité qui a été réalisé, notamment pour permettre un traitement cohérent de nos variables qui sont régulièrement modifiés entre toutes les fonctions.

Le jeu a été étudié de base comme comportant de très nombreuses fonctions jouant entre elles pour avoir des traitements efficaces des variables. C’est ce qui a été réalisé au final : un code comportant de très nombreuses fonctions, aucune classe, fonctionnant en appelant consécutivement chaque variable grâce au *mainloop()* de *turtle*. Toutes les variables utilisées globalement sont initialisés au début du code. Plusieurs *turtles* sont initialisées, chacune, indépendantes, aura un objectif précis, la représentation du nombre précis de l’une des quatre statistiques du code. Séparer les *turtle* nous permet d’optimiser les traitements de mise à jour des statistiques et donc de permettre à l’utilisateur de cliquer aussi vite qu’il le souhaite. C’est pour cela que l’on inclue dans le GitHub un petit logiciel d’autoclicker pour faciliter la vie de l’utilisateur.

Nous allons maintenant expliquer l’intérêt de chaque fonction pour le code :

* La fonction **statistiques()** a pour objectif de créer spécialement l’interface représentant les différentes statistiques de l’utilisateur. C’est un fonctionnement simple de *turtle* qui a été réalisé principalement par tâtonnement et formant une interface simple mais en conséquence facile à comprendre pour l’utilisateur.
* La fonction **croix()** a pour objectif d’utiliser la *turtle* pour créer une croix, symbole de la religion chrétienne et représentante de ce qu’il faut cliquer dans notre jeu. Le fonctionnement de la fonction est très simple, suivant des boucles permettant de créer facilement notre représentation. L’utilisateur cliquera sur cette croix pour obtenir des pardons par jour.
* La fonction **dessinercadre()** a pour but de dessiner le reste des éléments d’interface, le cadre général, et appelle les fonctions croix et statistiques.
* Notre fonction **jeu()** comporte la fonction de *turtle* pour les clics, ainsi que le mainloop. C’est elle qui sera utilisée à la fin du code.
* La fonction **cliquer(x,y)** est la fonction globale comportant le bon fonctionnement du bon programme. Quand on clique sur l’écran, cela nous donne des coordonnées, et seulement en fonction de ces dernières, on fait un traitement particulier. Chaque traitement particulier est représenté par une fonction appelée, traitant la demande de l’utilisateur, que cela soit un achat d’un bonus particulier, l’ajout d’un pardon, ou même l’ajout aléatoire de péchés. Cette fonction continue jusqu’à ce que la variable jeutermine passe à 0, ce qui est le cas dans le cas précis d’une victoire ou d’une défaite. En plus des simples changements de statistiques, on a aussi ajouté une possibilité aléatoire d’avoir des bonus ou des malus, c’est-à-dire beaucoup plus de péchés ou d’indulgences de manière temporaire, tout ceci également représenté dans des fonctions. On vérifie à chaque fois si les conditions de victoire ou de défaites sont acceptés, et si c’est le cas, on traite les fonctions en conséquence. L’utilité des différentes *turtles* se ressent particulièrement ici pour le traitement des différents changements de statistiques.
* La fonction **dondeterre()** met en place l’ajout d’un don en échange de cinquante indulgences.
* La fonction **compterlescliquer(notrecompteur)** ajoute simplement un pardon ou plus, en fonction du nombre d’indulgences, dans le compteur des pardons.
* La fonction **bonususine()** met en place l’ajout d’une indulgence. Cet ajout d’indulgence peut aléatoirement augmenter le taux de péché.
* La fonction **retraitespirituelle()** échange cent pardons contre un taux possible de péché.
* La fonction **peche()** ajoute des péchés à chaque clic de l’utilisateur en fonction du taux de péché, s’il existe, et met à jour la *turtle* en question.
* La fonction **faveur()**, en échange de 20 indulgences, supprime cent péchés.
* La fonction **papeoudiable()** est celle qui induit que l’utilisateur a une chance sur 250 d’avoir soit une bénédiction pontificale qui lui ajoute 50 indulgences pendant 50 jours, ou l’inverse pour le taux de péché. C’est une sorte de rematérialisation du cookie d’or dans *Cookie Clicker*.
* La fonction **victoire()** vérifie si l’utilisateur a l’ensemble des statistiques pour pouvoir tester le jeu du péché originel.
* La fonction **defaite()** est l’exact opposé de cette dernière.
* La fonction **messagevictoire(findujeu)** permet de définir si l’utilisateur a le droit de finir le jeu ou non, s’il est dans une position de victoire il doit d’abord vaincre le jeu du serpent avant le titre de fin, dans tous les autres cas, on l’amène vers l’écran de défaite.
* La fonction **snake()** comporte l’ensemble du code utilisant *pygame* et formant le jeu du serpent. C’est un code très simple, avec un écran, un certain positionnement qui s’adapte en fonction du mouvement de l’utilisateur. Une pomme est mise aléatoirement sur l’écran, mais l’objectif est de l’éviter et de directement foncer à l’extérieur de l’écran pour gagner, ce qui est contre-intuitif. Le retour du findujeu décide si l’utilisateur a le droit a son écran de victoire ou non. L’algorithme permet de prendre en compte tous les cas possibles pour les positionnements.

Voici le fonctionnement théorique simplifié sur drawio du programme :



# Structures de données

Peu de données extérieures sont utilisées au sein de notre code. On peut penser à deux types de données en particulier, des musiques, pour former une pseudo bande-son, et des images, récompensant l’utilisateur qui a gagné ou perdu.

On utilise, grâce à l’import de *vlc*, des .mp3 pour les musiques, mais l’utilisation de .wav était également possible via winsound.PlaySound. Nous avons fait exprès de faire prendre des fichiers d’une très grande taille et particulièrement longs pour ne pas avoir à s’occuper du temps de la musique, en effet, l’utilisateur n’aura jamais le temps de finir les musiques en question[[3]](#footnote-3).

Les autres fichiers sont des .gif, et non des autres types d’images possibles, notamment car *turtle* ne prend en compte que des fichiers .gif pour pouvoir mettre en place des images dans son écran. C’est pour cela que l’on utilise la fonction *turtle.bgpic*.

Taille moyenne des données des fichiers utilisés :

|  |  |
| --- | --- |
| Nom du fichier | Taille totale |
| defeat.mp3 | 8208 ko |
| music.mp3 | 648 309 ko |
| victory.mp3 | 42 192 ko |
|  |  |
| taille totale | 693709 ko |
|  |  |
| Nom du fichier | Taille totale |
| hell.gif | 59 ko |
| victoiredeux.gif | 67 ko |
|  |  |
| taille totale | 126 ko |
|  |  |
| taille finale | 693835 ko |
|  |  |

# Manuel utilisateur : Le guide d’utilisation

Le fonctionnement du code est très simple : il faut cliquer pour que des pardons s’ajoutent au nombre de pardons. Quand on a assez de pardons, on peut acheter des indulgences, nous permettant d’acheter des dons, nécessaires pour gagner la partie. Tout ceci doit se faire tout en ayant une certaine gestion du nombre de péchés, ne devant pas dépasser 1000 sinon c’est une défaite. On doit donc gérer toutes nos ressources, profiter des bonus positifs, et accepter la défaite et recommencer en cas de malchances. Le rôle du serpent est secondaire, servant de représentation de la victoire de l’utilisateur s’il comprend l’objectif du jeu.

Voici les avancées prévues sur le code :

* Une meilleure esthétique générale, que ce soit des ajouts d’effets sonores ou des meilleurs graphismes par exemple.
* Plus de mini-jeux, avec des étapes à entreprendre en cas d’arrivées à certains jalons (par exemple 5000 pardons). Voir le jeu du *frog* par exemple.
* Améliorer le *snake* et le rendre pleinement fonctionnel
* Une meilleure et plus grande utilisation générale des musiques et des images
* Faire des nouveaux types d’achats de ressources possibles, en essayant par exemple d’implémenter une fonction utilisant *turtle.ontimer* pour par exemple donner des pardons par jour sans clic.
* Améliorer le code en général en limitant l’utilisation de variables globales.
* Séparer les différentes fonctions en classes pour une meilleure compréhension du code

# Documentation du code

Voir dans le code les différents commentaires, et les fichiers présents dans le GitHub, servant de documentation du code

1. https://www.youtube.com/watch?v=X8KBBmaEbJs [↑](#footnote-ref-1)
2. https://www.edureka.co/blog/snake-game-with-pygame/ [↑](#footnote-ref-2)
3. <https://www.youtube.com/watch?v=bUQJlCmd9oY&t=10008s>

   <https://www.youtube.com/watch?v=9tHHiYx-5RE&t=9s>

   <https://www.youtube.com/watch?v=UHknrRJPMZI> [↑](#footnote-ref-3)