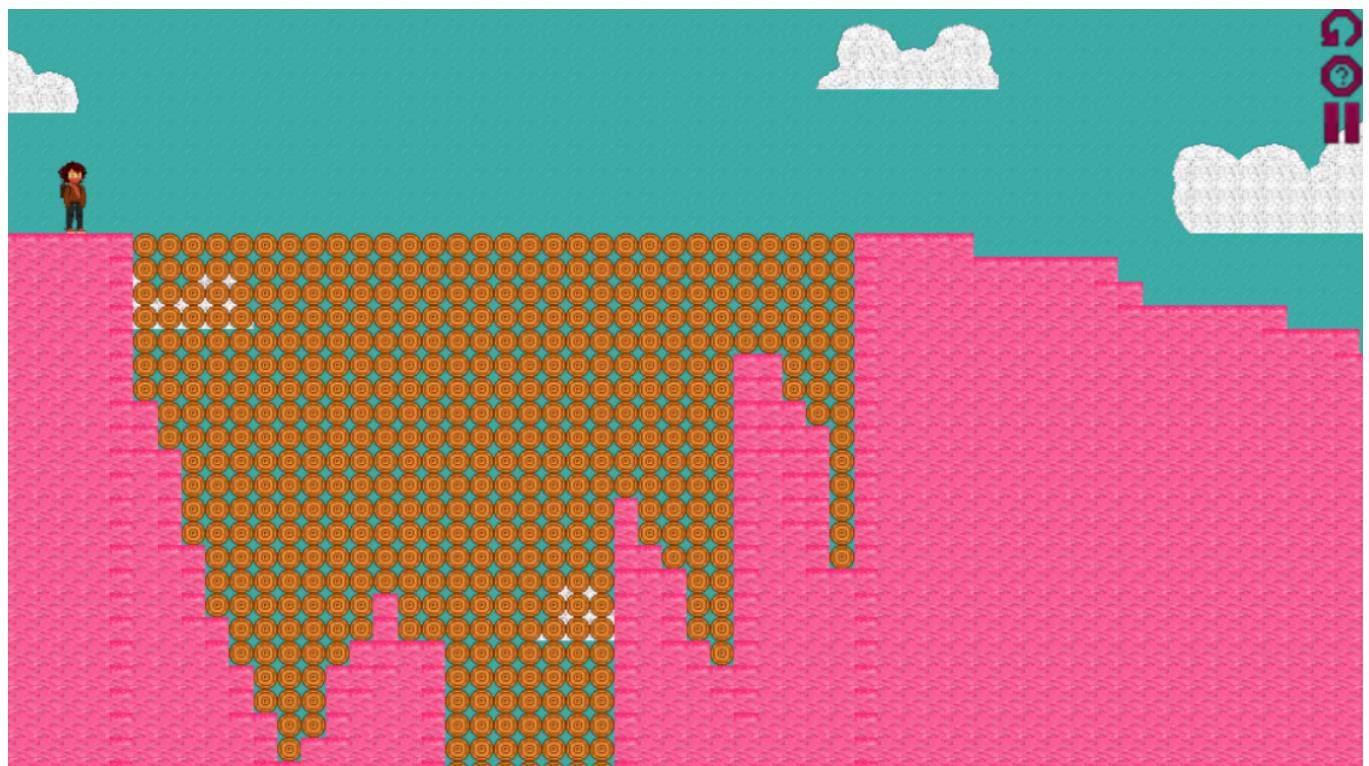


Rapport de TX :

Froggy and Frankie take a nice walk

Lauranne Fossat, Page Magnier-Slimani, Maxence Vahedi



I-Cahier des charges

Le but de cette TX est de créer un jeu ludique et pédagogique qui permette aux étudiant·e·s de l'UV INF1 d'apprendre le python et de progresser sur les différentes notions du cours et notamment celle des boucles. En effet, le but est de créer un jeu avec des niveaux à difficultés croissantes qui parte de la base du concept des boucles à son approfondissement, en permettant une valorisation de la progression. Le projet doit pouvoir être utilisé facilement par les étudiants et étudiantes de l'UV, en autonomie sur un créneau d'une heure de TP. Le but est également de fournir un produit fini esthétique et plaisant qui donne envie aux étudiants et étudiantes.

De notre côté, il était attendu de nous que ce projet nous permette d'approfondir nos connaissances du langage de programmation Python en travaillant sur un projet à portée globale, regroupant des savoir-faire artistiques et informatiques divers. Le but était de proposer un produit qui permette une application de nos connaissances en informatique acquises à l'UTC pour les mettre aux services d'un dispositif pédagogique, que nous aurions aimé nous-même voir lors de notre formation.

II-Compétences à transmettre

Notre première réflexion concerne les compétences à transmettre. Certes, nous réalisons un jeu éducatif ayant pour but de familiariser les élèves avec l'utilisation des boucles mais le cahier des charges reste imprécis.

Avant d'écrire la moindre ligne de code, nous devons nous assurer que nous avons une idée claire des compétences précises à transmettre, ainsi que des profils et connaissances préalables des élèves.

Nous nous sommes basés sur les sujets de TP des années précédentes, les diapositives du cours ainsi qu'une fiche pédagogique que nous avons remplie à partir des appels avec Domitile Lourdeaux. Les compétences à transmettre sont donc ces dernières :

- les boucle *for*
- les boucle *for* utilisant la variable i dans le corps de la boucle
- les boucle *for* en décrémentant l'indice i
- les boucles *for* ne commençant pas nécessairement à 0
- les boucles imbriquées

- les boucles à indices différents (pairs par exemple)
- les boucles *while* conditionnelles
- les boucles comprenant plusieurs conditions d'arrêt

Le tout doit s'adapter au niveau inégal des élèves et donc être le plus intuitif possible tout en proposant des défis optionnels, plus durs pour les élèves en avance. De plus, l'installation doit être la plus simple possible pour minimiser le temps qui y est consacré. Cette définition des objectifs, si elle doit être claire avant de commencer notre projet, ne doit pas être figée.

En effet, nous avons étayé cette dernière au regard de tests du jeu à différentes étapes de développement pour saisir les problèmes rencontrés par les élèves. Par exemple, une première démo fonctionnelle a révélé des difficultés à savoir si leurs codes étaient dysfonctionnels, ou juste en train de s'exécuter pour plusieurs étudiants et étudiantes. Ce manque de clarté a pu être réglé par un *timeout*, des messages d'erreurs plus clairs ainsi que la réduction de la taille des niveaux et donc du temps d'exécution.

III-Réflexions autour du benchmark

Mais alors, comment faire pour transmettre ces connaissances ? Afin de gagner en pertinence, nous avons commencé par nous intéresser à ce qui s'est déjà fait en termes de jeu éducatif pour apprendre les boucles en Python. Bien sûr, nous ne sommes pas les premières à s'atteler à une telle conception : nombreux ont été les essais, se soldant souvent à notre sens par des échecs.

Là où l'intégralité des jeux que nous avons croisés échouent, c'est à être des jeux. Dans les faits, bien souvent, le jeu s'apparente plutôt à un TP déguisé en jeu vidéo. Notre ambition avec cette TX était donc de faire plus qu'un "TP ludique", un véritable jeu intéressant et plaisant, qui a la propriété de transmettre les attendus pédagogiques d'un TP - en l'occurrence sur les boucles.

La très grande partie des jeux éducatifs pour apprendre les boucles en Python que nous avons croisés fonctionnent de la même manière : un personnage doit aller d'un point A à un point B en évitant des obstacles, traversant un labyrinthe, et parfois même combattre des monstres. On programme donc les déplacements du *sprite*. Le reproche que l'on peut faire à de tels jeux est leur rythme très lent et la distance mise avec le jeu : le·a joueur·euse n'incarne pas le personnage, iel le programme. Ainsi le jeu est déshumanisé et perd beaucoup de son potentiel immersif.

C'est pourquoi nous proposons une approche en rupture avec ce paradigme : les joueuses et joueurs peuvent ainsi contrôler plus directement (avec les flèches du clavier) et donc incarner

le personnage. Mais alors, où intervient le code ? Pas de *spoil* : nous reviendrons à l'idée centrale de notre jeu, celle qui fait sa spécificité, dans la section de ce rapport portant sur le *game design*.

IV-Réflexions préliminaires et méthodologie

L'informatique n'est pas neutre. Son développement est marqué par l'exclusion systématique des minorités, et ce depuis son émergence de l'industrie militaire. Il s'agit encore aujourd'hui d'un milieu masculin et peu inclusif, et ce malgré toutes ses promesses émancipatrices. Ce projet prend tout son sens en notre profonde volonté d'aller vers une informatique centrée sur l'humain – toutes et tous les humains.

Le geste créatif lui non plus n'est pas neutre et ce particulièrement lorsqu'on le mène à des enjeux éducatifs. Nous avons le parti pris de défendre une informatique libre, accessible à toutes et à tous et qui sert d'outil convivial au sein de notre société.

En ce sens, réaliser un jeu éducatif permettant au plus grand nombre d'acquérir une compréhension basique de l'informatique s'ancre profondément dans nos valeurs. À une échelle plus globale, il s'agit de démystifier la science pour que le plus grand nombre puisse s'en saisir et comprendre la manière dont cette dernière n'est ni neutre, ni absolue, mais bien la somme de toutes celles et ceux qui la font exister. C'est pourquoi nous préférons profiter de notre parenté sur notre code pour le diffuser le plus largement possible en *open source*, puis si possible sur des plateformes éducatives afin que notre jeu ne serve pas qu'à des futurs ingénieurs.

C'est avec cette réflexion que nous avons débuté notre production. Afin qu'elle ne reste pas que théorique, nous nous sommes assurées qu'elle structure notre production en adéquation avec notre contenu, et ce avec une attention toute particulière à l'épanouissement de chacune d'entre nous et sans hiérarchie. Nous avions conscience que les ambitions que nous avions pour notre jeu dépassent le simple TP ludique et qu'il en résultera une charge de travail conséquente.

Nous avons donc pris soin de partager nos ressentis par rapport au projet régulièrement afin de s'assurer que personne ne se sente surchargé ou encore dévalorisé dans son travail. La communication a été un des éléments centraux de notre travail. En complément des réunions hebdomadaires qui suivaient la présentation de notre avancée, nous nous réunissions régulièrement en groupe de travail autour de différents éléments du projet (moteur, direction artistique, menu, etc.).

Nous avons également défini des dates clés pour lesquelles nous souhaitions avoir un moteur fonctionnel, puis une première démo, un produit minimum viable et enfin le jeu complet. Pour tenir ce rythme, chaque semaine, les tâches à faire étaient déterminées de manière collégiale puis disposées sur un tableau commun permettant de suivre nos avancées. Les réunions de groupe de travail (souvent en mi-semaine) permettaient une flexibilité cruciale au fonctionnement de ce formalisme. Certains chantiers particulièrement longs tels que les visuels du jeu ou les fonctions utilisables par le·a joueur·euse nécessitaient beaucoup d'anticipation.

Tout au long du projet, nous nous sommes assurées d'obtenir en temps et en heure un produit minimal viable, abandonnant lorsque nécessaire les éléments trop chronophages. Faire un jeu demande beaucoup de temps, bien plus que 125 heures par personne pendant un semestre. C'est pourquoi nous avons essayé de rester lucides quant aux compromis nécessaires.

V-Outils utilisés

Créer un jeu relève d'une grande variété de tâches et de domaines. Nous avons donc utilisé de nombreux outils aux utilités et compétences très variées.

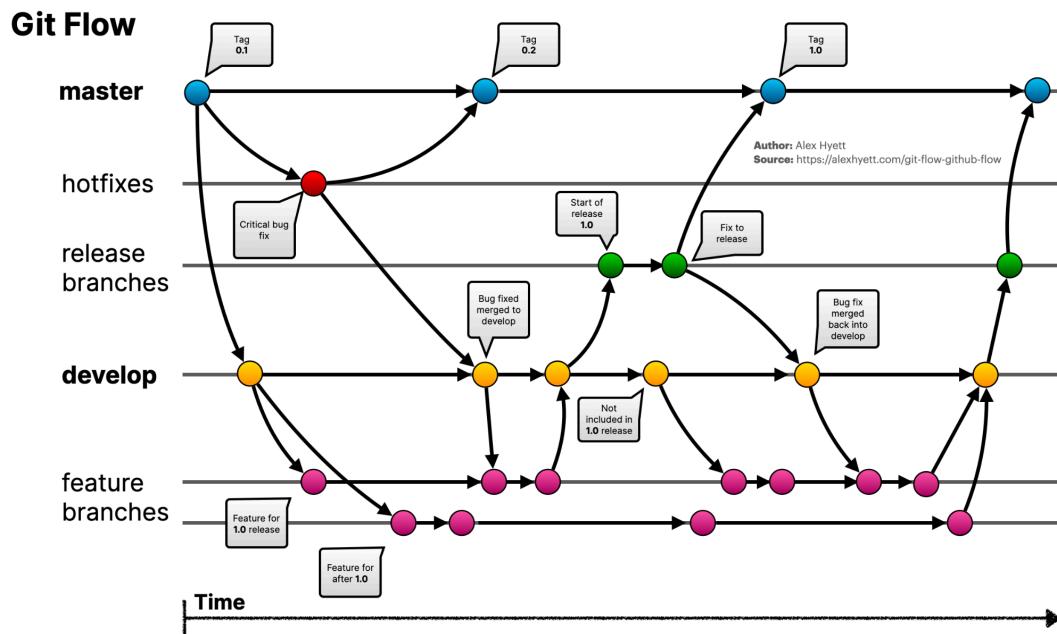
Organisation : Notion et approche agile

Un des premiers défis de la TX a été pour nous de s'organiser : même à trois personnes qui se connaissaient bien et savaient travailler ensemble, il s'agissait d'un projet particulièrement ambitieux. Ainsi fluidifier l'attribution des tâches, la planification des réunions et leurs ordres du jour étaient des objectifs que *Notion* nous a permis d'atteindre. La particularité de *Notion* est sa modularité : ainsi nous avons conçu notre propre espace de travail personnalisé, en nous appuyant notamment sur un modèle s'apparentant à une forme d'approche agile, décomposant le développement en projets, eux-mêmes découpés en tâches. Bien sûr, l'échelle réduite de notre équipe nous a permis de circuler entre les projets et ainsi d'acquérir une grande variété de compétences ; cependant, nous avons fait le choix d'attribuer le rôle de “(co-)responsable” sur chaque projet afin de garantir son avancement. En synthèse, *Notion* a été le plus important support logistique du projet, des *deadlines* de réunion aux tâches en passant par la documentation de notre avancement. Le défi qui succède immédiatement à cette organisation est celle de notre code : comment gérer l'avancement parallèle de tant de projets sur une même *codebase* ?

The screenshot shows the Notion interface for the 'Projets' page. On the left, there's a sidebar with navigation links like 'Search', 'Updates', 'Settings & members', 'New page', and 'Teams'. Under 'Teamspaces', 'Projets' is selected. The main area has a header 'Projet / Projets' and a top bar with user icons and filters. Below is a title 'Projets' with tabs for 'Actifs', 'Gantt', 'Tous les projets' (which is selected), 'All', and 'New'. There are three cards representing tasks: 'Moteur de jeu' (78.4% complete), 'Game Design (1er TP Boucles)' (85.7% complete), and 'DA Visuelle'. Each card has a 'New' button. The top right has 'Sort', 'Filter', and other search/refresh buttons.

Code : *Git, Github, PyCharm & Sphinx*

Nous avons choisi d'utiliser *Git* comme gestionnaire de versions, et *GitHub* comme service de stockage de dépôt distant. Ainsi en travaillant avec le CLI de *Git* ou *GitHub Desktop* selon la préférence de chacune, nous avons pu organiser notre code selon un *git flow* défini au préalable et minimisant les conflits de code entre les centaines de *commits* répartis sur des dizaines de branches :



Source : <https://www.alexhyett.com/git-flow-github-flow/>

En tant qu'IDE, nous travaillons toutes déjà sur *PyCharm* par habitude ; ainsi nous avons utilisé celui-ci tout au cours du projet pour le développement. Notons que c'est sur ce type de grands projets très focalisés sur une technologie (en l'occurrence, Python) que ces IDE spécialisés révèlent toute leur utilité : ses nombreuses fonctions taillées pour le langage ont grandement aidé à accélérer le processus de production du code. Nous avons d'ailleurs produit une documentation de celui-ci, à l'aide d'un autre outil, *Sphinx*.

Parmis les nombreux avantages que présente *Sphinx* (par rapport à un simple langage de prise de note comme *Markdown*, que nous avons aussi envisagé), la génération semi-automatique de documentation par modules nous a fait gagné un temps monumental et nous a permis – et nous permettra – de maintenir cette documentation à jour à l'aide d'une simple commande.

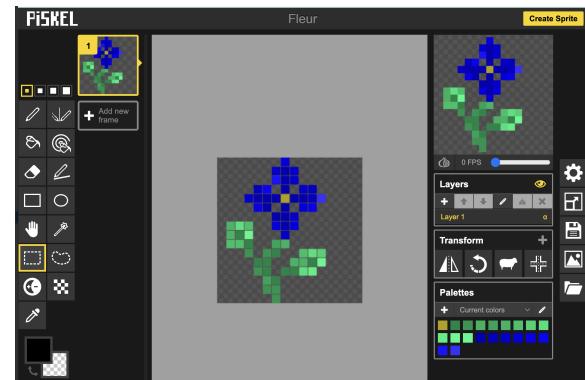
Il s'agit d'une documentation technique (plus précisément, une *API reference*) qui s'avérera – nous l'espérons – très utile à l'éventuelle passation vers de nouveaux étudiants ou simplement pour permettre à quiconque de comprendre le code source un peu plus facilement. Il s'agissait d'une condition nécessaire à la satisfaction de notre exigence d'*open-source* pour le projet : nous voulons que ce projet soit libre d'accès et de contribution à quiconque !

Direction artistique & schémas techniques : *Figma*

Figma est un outil collaboratif de design et de brainstorm que nous avons utilisé principalement à sa seconde fin (qui est assurée par le sous-outil *Figjam*). Ainsi la production des palettes et du *moodboard* y ont été faites, tout comme les réflexions sur la manière dont il était possible de surmonter des défis techniques – comme par exemple celui de l'*artificial buffer*.

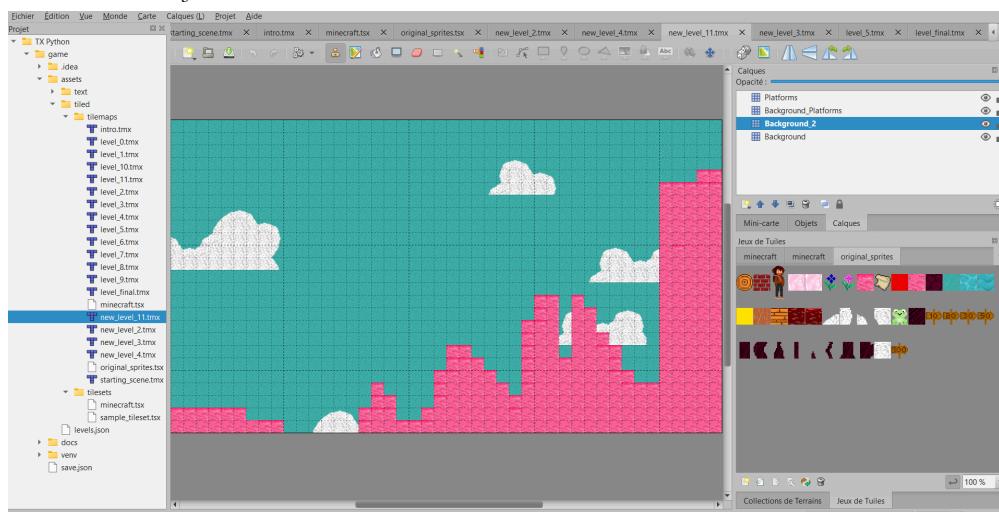
Pixel art : *Piskel*

Piskel est un logiciel web, facile à utiliser et adapté aux débutant·e·s (grâce au nombre réduit de fonctionnalités proposées et à la simplicité de celles-ci) permettant la création et l'édition de *sprites*. Il permet de dessiner une image pixel par pixel, et offre la possibilité de décomposer un dessin en plusieurs *layers* se superposant. De plus, il permet l'animation de *sprites* en différentes *frames* avec une visualisation du résultat en temps réel, selon une vitesse en images par seconde à définir. Cet outil est également très pratique en termes de sauvegarde. Il permet d'enregistrer directement un *sprite* sous la forme d'un PNG et propose même la possibilité de l'enregistrer au format *piskel*, très utile pour partager des *sprites* complexes et avoir accès à toutes les *layers* et/ou *frames* du dessin.



Game design : *Tiled* & croquis papier

Pour ce qui est du *game design*, la plupart des niveaux ont été conçus sur papier, pour pouvoir faire émerger les idées plus facilement. Le logiciel *open source Tiled* nous a permis de facilement incorporer les images que nous avions créées au sein de fichiers JSON. Ainsi, nous avons pu effectuer assez rapidement des changements à l'architecture de nos niveaux avec une interface visuelle claire. Le temps de dessiner nos propres images, nous avons temporairement utilisé celles de *Minecraft*, facilement accessibles.

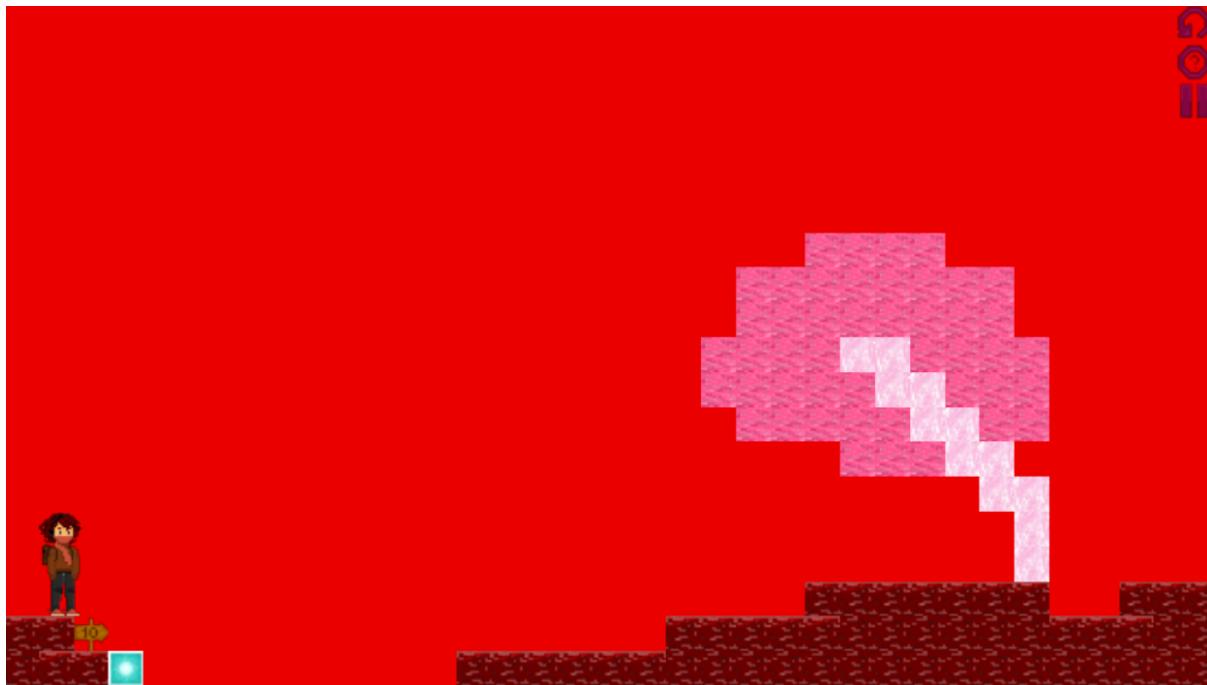


VI-Game Design

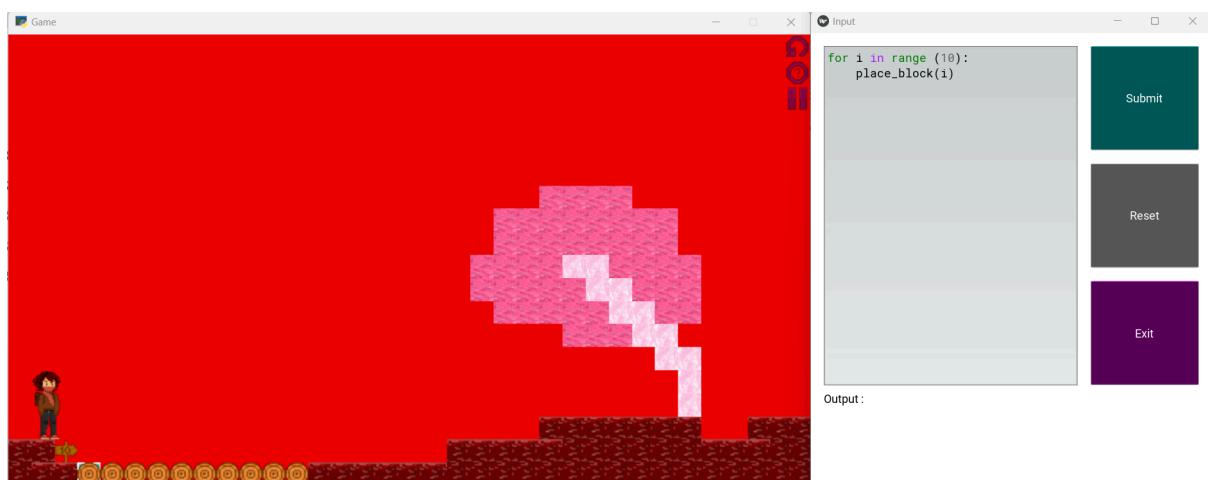
Mais alors, comment transmettre la compréhension des boucles en Python ? Le *benchmark* évoqué précédemment est soit très généraliste, soit très peu inspiré. Notre réflexion de *game design* prend ses racines dans la question de l'usage direct des boucles en informatique. Leur utilité première est d'exécuter de manière répétée des instructions et d'automatiser le traitement de différents cas de figure. Il serait donc intéressant de proposer aux joueurs et aux joueuses une manière de visualiser l'automatisation des actions à répéter.

C'est à force d'itération que nous sont venus les fondements de notre *game design*. En opposition aux jeux très peu ludiques offrant la possibilité de contrôler le personnage au moyen de code, notre jeu permettra de manipuler son environnement. Ainsi ; au sein de cette expérience, les joueurs et joueuses pourront franchir des obstacles en posant des blocs au moyen de commandes informatiques dans une deuxième fenêtre prévue à cet effet.

Par exemple, dans ce niveau :

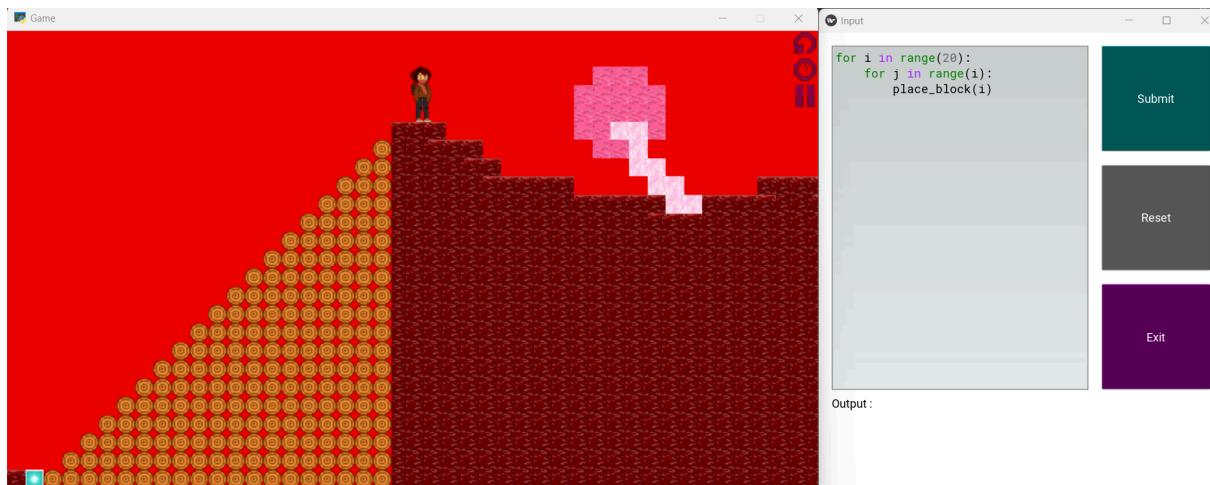


L'exécution du code suivant dans la deuxième fenêtre pose 10 blocs à la suite aux coordonnées correspondantes et résout donc le niveau :



À partir de cette logique de base, de nombreuses variations sont possibles. Nous avons donc repris la liste de compétences à transmettre et tenté de trouver les manières les plus élégantes de montrer les différentes utilités des boucles en *Python* pour permettre leur appréhension.

Ainsi, pour saisir la notion de boucles imbriquées, nous utilisons par exemple un escalier comme dans l'exemple ci-dessous :

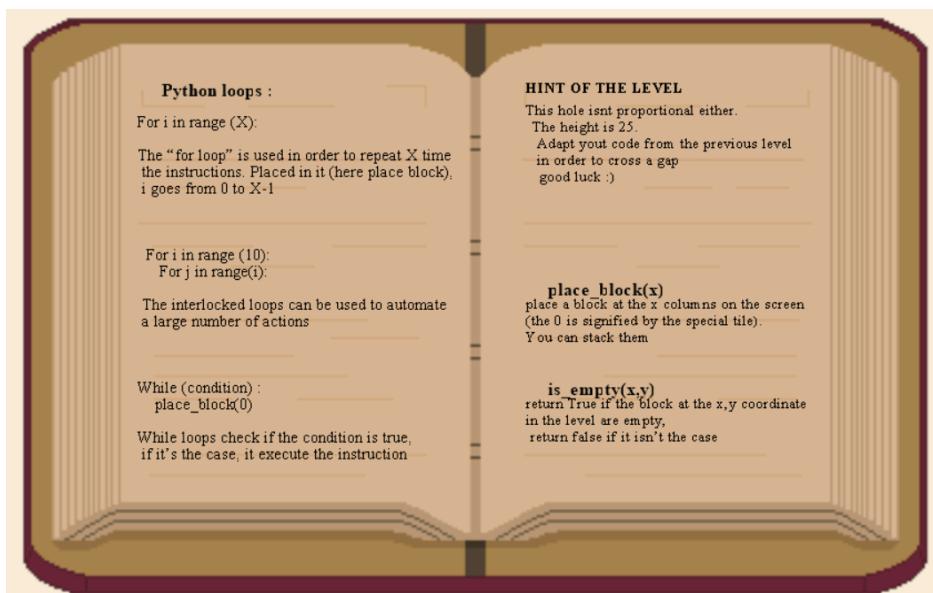


Nous avons dû faire face à la difficulté de devoir développer un jeu à destination de profils très variés. Comment faire en sorte que des étudiantes et étudiants n'ayant jamais programmé auparavant puissent apprécier l'expérience que nous proposons, tout en réservant une difficulté plus corsée à celles et ceux avec un bagage informatique préalable ?

Nous avons donc décidé de séparer l'expérience en 9 premiers tableaux suivis de 4 niveaux optionnels à la difficulté élevée. Ainsi, les différents profils de joueurs et joueuses peuvent avoir une expérience satisfaisante.

Nous avons beaucoup travaillé sur l'inclusivité de notre jeu. Dès l'installation, nous avons veillé à ce qu'avec une simple exécution d'un fichier compilé, les librairies nécessaires soient installées et que le jeu se lance.

Au sein du jeu, un livre accessible à tout moment par les joueurs et joueuses contient l'intégralité des fonctions du jeu, des rappels sur les boucles ainsi qu'un indice unique à chaque niveau. De plus, des personnages non joueur·euse·s donnent des indications de manière régulière.



Nous avons implémenté les fonctions *place_block(x)* et *is_empty(x,y)* comme éléments centraux de game design. Leur évolution est en trois temps. Au début, elles servent à construire des structures basiques telles que des ponts ou des escaliers. Une fois une compréhension basique des structures de code acquises, des trous suivant des fonctions mathématiques font leur apparition et les joueurs et joueuses sont amené·e·s à répliquer par exemple la suite de fibonacci ou une fonction quadratique.

Dans un dernier temps, les niveaux amènent les élèves à trouver une manière générique de résoudre n'importe quel niveau, peu importe son contenu en utilisant l'intégralité de ce qu'ils ont pu voir jusqu'ici, incluant des boucles imbriquées, les fonctions *place_block* et *is_empty* et des tests avec *if*. Les élèves étant parvenus à la fin du jeu devraient donc avoir une solide compréhension de la mise en synergie de ces outils. Ils seront récompensé·e·s par un easter egg - une fonction leur permettant de transformer leur personnage en grenouille.

Néanmoins, la création d'un jeu ne se fait pas dans l'application directe d'un plan d'action. De nombreux changements ont dûs être faits au fur et à mesure pour rendre l'expérience plus agréable. Certaines de ces évolutions rétroactives sont indispensables.

Un des exemples principaux de ces petits changements qui impactent l'expérience des utilisateur·ice·s de manière conséquente est par exemple l'ajout de panneaux indiquant la taille des trous à franchir. Le livret d'indice a aussi été ajouté en réaction aux différentes difficultés constatées des testeurs et testeuses.

VII-Production artistique

Plutôt que de rajouter une histoire générique à un produit fini, nous nous sommes posées la question de ce que racontent nos mécaniques de jeu. Nous avons décidé qu'il serait intéressant d'utiliser les boucles pour parler de santé mentale et de la nécessité de réussir tous les jours à effectuer des tâches basiques qui peuvent pourtant être très éprouvantes.

Nous voulions représenter qu'au fur et à mesure que les joueur·euse·s maîtrisent mieux les boucles, leur personnage devient également de plus en plus capable d'effectuer des tâches complexes et va de mieux en mieux. Néanmoins, nous avons dû adapter notre narration au fait que le sujet de la santé mentale puisse être sensible pour certain·e·s élèves et que d'autres puissent vouloir simplement faire leur TP sans être ralenti·e·s par la narration. Nous avons donc opté pour une narration environnementale où le changement d'atmosphère progressif, mêlé à la satisfaction de résoudre des problèmes de plus en plus complexes, permet à chacun et chacune d'avoir sa propre interprétation de l'histoire du personnage.

Pour transmettre cette évolution, nous avons commencé par réaliser trois palettes de couleurs pour les différents moments clés de l'aventure ainsi qu'un descriptif des ambiances que nous recherchions. Nous avons également fait un *moodboard* à partir des différentes inspirations que nous avions.

1^{ère} palette de couleurs



2^{ème} palette de couleurs



3^{ème} palette de couleurs



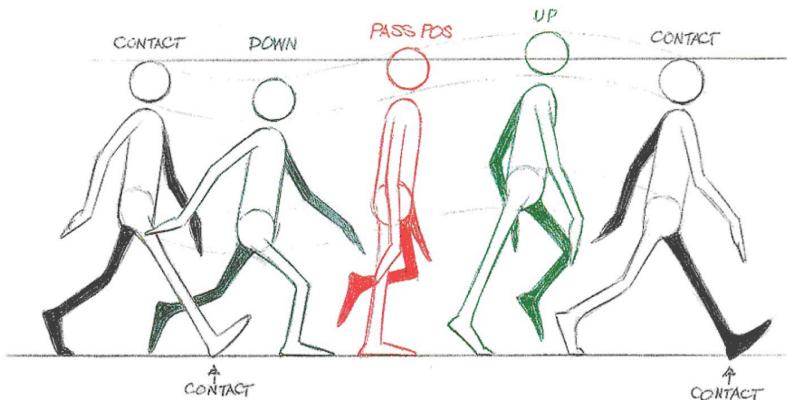
Les écrans de fin, avec leurs tons pastels, contrastent avec les nuances de rouge très intenses du début, permettant une satisfaction contemplative en les atteignant. Le *sound design* a été retiré du jeu, ce dernier étant joué en salle de TD, sans écouteurs avec plusieurs dizaines d'élèves.



Par ailleurs, l'histoire de notre jeu passe également par notre personnage. Celui-ci, à l'apparence timide, caché derrière son écharpe, emmitouflé dans son pull en laine, et coupé du monde extérieur par son casque audio, nous emmène avec lui dans sa découverte des boucles en *Python*.

Un des enjeux de son implémentation a donc été d'animer ce *sprite* pour visualiser son mouvement au moment où celui-ci était déplacé par le joueur ou la joueuse. Pour cela, nous

avons construit un enchaînement de 8 *frames*, décomposant le mouvement de marche du personnage selon un schéma basique d'analyse de poses d'un déplacement comme ci-dessous :



Source : <https://idearocketanimation.com/2-walk-cycle/>

Une fois les poses dessinées, il suffit de les assembler et d'itérer sur chacune d'entre elles dès que le personnage est mis en mouvement, et on obtient l'illusion que celui-ci avance.



VII-Structure du moteur

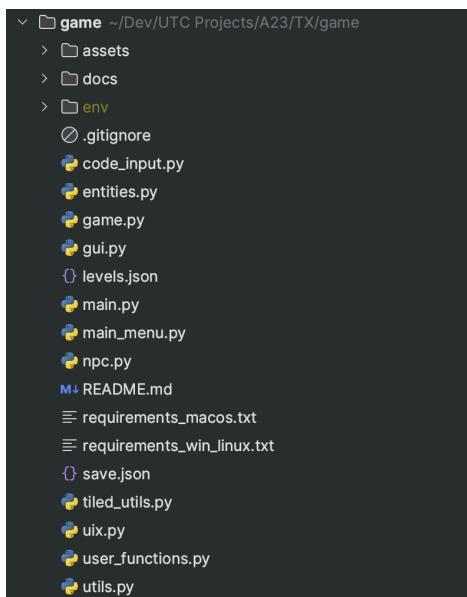
Notre jeu se présente sous la forme de deux fenêtres qui interagissent entre elles. La première fonctionne comme un *platformer* presque lambda, dans le sens où le joueur ou la joueuse contrôle un personnage qui doit se déplacer sur des plateformes et éviter des obstacles. La seule subtilité est que le personnage ne pouvant pas sauter, la seule manière de parvenir à traverser les failles qui se présentent devant lui est pour le·a joueur·euse d'utiliser la deuxième fenêtre. Celle-ci se présente sous la forme d'un IDE simplifié où l'utilisateur·ice peut écrire du code.

Le moteur de notre jeu utilise principalement deux bibliothèques, chacune gérant le fonctionnement d'une des deux fenêtres. Ainsi, la fenêtre de déplacement du personnage est gérée par la bibliothèque *Arcade* et la seconde par la bibliothèque *Kivy*. Le choix de ces bibliothèques nous est apparu comme plutôt évident dans la mesure où *Arcade* est une bibliothèque de référence dans le développement de jeu vidéo en *Python* et *Kivy* est une des seules bibliothèques proposant le développement d'application GUI (Graphic User Interface) permettant de gérer du code. Ainsi, avec nos deux bibliothèques, il ne nous reste plus qu'à imaginer un moyen de relier les deux fenêtres entre elles pour permettre leurs interactions.

Le problème qui nous apparaît alors est que ces deux bibliothèques fonctionnent sur le même principe de programmation événementielle, c'est-à-dire que le code s'effectue en boucle en attendant que l'utilisateur·ice interagisse avec l'application, que ce soit en cliquant sur un bouton, en relâchant une touche de clavier ou autrement. Par conséquent, lorsque l'une des deux applications est lancée, l'autre ne s'ouvre qu'à la fermeture de la première, lors de l'arrêt de sa *mainloop*. L'enjeu est alors de séparer les deux boucles en deux processus différents pour permettre leur exécution en parallèle.

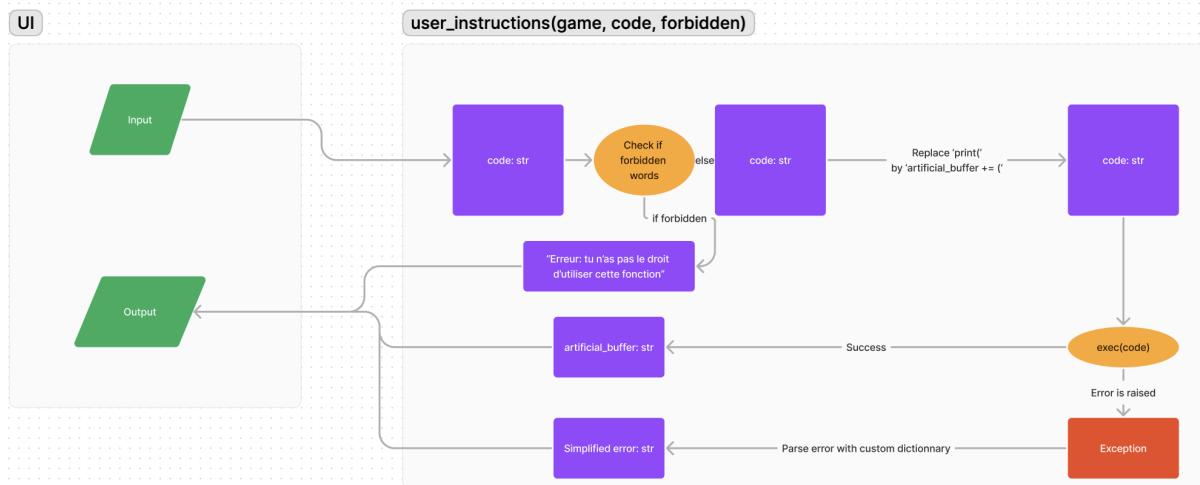
C'est pour permettre cette séparation qu'apparaît la dernière bibliothèque principale utilisée par notre jeu, *Multiprocessing*. Celle-ci permet de lancer différents processus en même temps et de permettre de les relier, pour échanger des informations de l'un à l'autre, à l'aide d'un objet *pipe*. C'est cette structure compartimentalisée parallèlement qui permet d'exécuter les deux applications en même temps sans que les bibliothèques n'entrent en conflit.

Le code du jeu est très conséquent et complexe, ainsi il convient de le structurer pour ne pas s'y perdre entre menus, fenêtres, entités, ressources graphiques *etc.* La capture d'écran ci-dessous rend compte de la segmentation en modules de celui-ci :



Une fois le système basique de notre moteur posé, il nous reste à expliciter le lien entre les deux fenêtres du jeu et notamment comment l'exécution du code est gérée. En effet, la fenêtre utilisant *Kivy* ne permet que l'affichage de l'input utilisateur sous forme de code et non son traitement. En réalité, une fois que l'utilisateur·ice appuie sur le bouton *submit*, le code, tel qu'il est écrit, est envoyé sous forme d'une chaîne de caractère à la seconde fenêtre par l'intermédiaire du *pipe*. Du côté de la fenêtre *Arcade*, le code est modifié, vérifié puis exécuté, de sorte à ce que les changements en lien avec la partie *platformer* soit exécuté à l'intérieur de l'instance de notre jeu et que les données retournées par l'exécution du code soient renvoyées vers la fenêtre de code pour être affichées à l'écran.

Un exemple parlant de difficulté que nous avons eue lors de la conception de la fonction `user_instructions()` exécutant le code entré par les joueur·euse·s est la manière de retourner le *buffer*, i.e. de ce qui a éventuellement été *print*. Après avoir envisagé de nombreuses possibilités, résultant de nos recherches et de nos réflexions, nous avons abouti au schéma qui suit :



La solution pour laquelle nous avons opté est la création d'une variable `artificial_buffer` stockant la string résultante du script de l'utilisateur. Après avoir vérifié le code entré, les “`print`” sont remplacés par “`artificial buffer += ()`” ce qui a pour effet de stocker dans la variable tout ce qui aurait en temps normal été `print` dans la console du processus `main`, inaccessible aux *end users*. Cette variable est transmise du processus *Arcade* au processus *Kivy* au travers du *pipe*, et le résultat est affiché sur la fenêtre de code.

Il ne s'agit là que d'un exemple des nombreuses ruses que nous avons dû mettre en place pour réaliser techniquement nos idées de *gameplay* qui s'éloignent considérablement de ce qu'il est typiquement prévu de faire avec les *frameworks* utilisés. Pour plus d'informations sur le détail des modules et des fonctions, se référer à l'*API reference* mentionnée plus tôt.

Il reste enfin, à déterminer les fonctions appelables par l'utilisateur. Elles sont au nombre de deux¹. La première, `place_block`, permet de placer un bloc sur l'emplacement le plus bas disponible à la position horizontale passée en paramètre. La seconde, `is_empty`, permet de vérifier si une position (x, y) contient ou non déjà un bloc.

Ces deux fonctions ayant une portée dans la fenêtre de déplacement du personnage (*Arcade*) et étant appelées dans la fenêtre d'affichage du code (*Kivy*), une modification du code en `input` est d'ajouter en paramètre l'instance de notre jeu pour que les modifications puissent être prises en compte. Ainsi, sur le même schéma que la variable `artificial_buffer`, la chaîne de caractère qui compose le code entré par l'utilisateur est modifiée pour faire apparaître en premier paramètre, la variable instanciée de notre jeu, en cours d'utilisation.

¹ Sans compter `frog`, un *easter egg*. Essayez pour voir !

VIII-Conclusion et pistes d'améliorations

Ainsi, *Froggy and Frankie take a nice walk* était un pari ambitieux visant à dépasser le concept de “TP ludique” pour tenter de proposer une expérience pédagogique autant que vidéoludique. Cet objectif a nécessité une organisation très structurée de notre part pour s’assurer d’avoir un livrable dont nous puissions être fières et qui puisse remplir les objectifs pédagogiques et s’adapter aux retours de notre responsable de TX très à l’écoute, et ce en préservant nos santé mentales.

Pour accomplir notre vision artistique, nous avons dû pousser nos outils dans leurs retranchements. *Arcade Python* est un *framework* intéressant, bien documenté, mais un peu trop rigide pour créer une expérience en décalage avec celles existantes. Nous avons donc dû faire usage de plusieurs processus simultanés et modules ainsi que d’une grande planification artistique pour mettre sur pied un prototype fonctionnel puis un jeu complet.

Nous avons conscience que notre travail ne s’arrête pas ici. Il faudra au prochain semestre faire de l’assistance auprès des élèves pour pouvoir faire évoluer notre jeu en fonction des retours, *bugs* et difficultés rencontrées. Ce constat nous invite à penser aux différentes pistes d’améliorations de notre jeu.

Par exemple, il aurait été intéressant de proposer des manières alternatives et plus complexes de résoudre les niveaux au moyen de collectibles situés à des endroits stratégiques. Cela aurait été une manière de renforcer la rejouabilité et de s’assurer que celles et ceux qui ont de l’avance puisse profiter d’une expérience épanouissante.

Nous avions également pour projet d’ajouter des cinématiques. Malheureusement, la rigidité d’*Arcade Python* aurait forcé cette implémentation à se faire image par image en recréant un lecteur vidéo nous même. Renforcer la scénarisation de notre jeu par des moyens alternatifs tels que des éléments disséminés au fur et à mesure serait une manière intéressante de rendre l’expérience plus immersive.

Nous sommes fières de ce que nous avons créé. Cette TX a été très enrichissante, nous permettant d’apprendre, en plus de l’utilisation de nombreux modules et de fluidifier notre maîtrise de Python, à travailler en équipe autour d’un projet de code, à structurer notre travail et à maîtriser des outils tels que *Git*. Cette UV a donc a bien des égards été pédagogique vis-à-vis de nos futurs métiers dans l’informatique, en plus de nous permettre de travailler de manière épanouissante avec des amies que l’on aime fort.

Et qui sait, peut être que l’on pourra créer notre studio de jeu vidéo indépendant un jour.