TomNoob's blender

Rapport de soutenance 3

Table des matières

1 Introduction

Nous voici à la fin de notre projet de S4. Vous pourrez lire dans ce rapport tout ce qui concerne la conception et la réalisation de notre jeu TomNoob's blender. Tout d'abord faisons un rappel de ce qu'est TomNoob's blender. TomNoob's blender est un jeu du même type de Candy Crush ce fameux jeu sur téléphone portable. Nous avons choisi comme thème les fruits d'Animal Crossing.

Le but du jeu est d'aligner des fruits pour gagner un maximum de points. Or nous avons rajouté une intelligence artificielle qui tentera d'avoir le meilleur score. Pour résu-

mer, TomNoob's blender est un jeu de type Candy Crush où une intelligence artificielle pourra jouer.

2 EMSY

Qui sont donc les membres du groupe EMSY ? C'est ce que vous trouverez ci-après. Tout d'abord EMSY est un groupe d'amis. C'est aussi un groupe d'étudiants partageant ce point commun qu'est l'informatique. Nous avons connu des hauts et des bas mais nous avons surmonté les difficultés. Nous avons réussi à rester souder et à communiquer correctement pour vous présenter le jeu TomNoob's blender !

2.1 Dragan

Eldaram alias Dragan. Toujours présent mais souvent en retard.

2.2 Céline

Mikutakitoku ou bien Maktu Maktu, c'est Céline. Cheffe de projet mais fauteuse de troubles de sa belle voix.

2.3 Katia

SKYWy ou bien Katia. On l'appelle aussi la sorcière car elle connaît beaucoup trop de formules magiques.

2.4 Kenjy

Yjnek ou Kenjy à l'endroit. C'est un homme mystérieux.

3 Reprise du cahier des charges

3.1 Présentation du projet

3.1.1 L'histoire de Tom Noob

Tom Noob est le frère de Tom Nook. Commençons par une brève présentation de Tom Nook. Tom Nook, connu au Japon sous le nom de Tanukichi, est un personnage de jeu vidéo dans la série Animal Crossing. Il est le gérant du magasin du village. C'est un Tanuki, un raton laveur. L'avis des joueurs est mitigé, nous avons d'un côté ceux qui admirent Tom Nook et ceux qui le détestent. Mais pour quelle raison ? Certains se sentent exploités car il demande beaucoup mais ne donne rien en échange. Tom Noob, l'opposé de Tom Nook, est un chômeur qui passe la journée dans sa cuisine pour faire son plat préféré : le smoothie. Ses fruits préférés sont la pomme, l'orange, la pêche et la

poire. Le mélange de ces fruits donne un smoothie très sucré et doux.

3.1.2 Nature du projet

Tous les plus grands projets sont nés d'un rêve. Notre projet ne fait pas exception à cette règle. Candy Crush est un jeu très connu qui est réalisable à notre niveau mais trop simple pour un projet d'un semestre. Pour rendre la chose plus intéressante, nous avons décidé de créer une intelligence artificielle qui va jouer et tenter de battre son propre record. Pour plus de "fun", le thème du jeu ne sera pas des bonbons mais des fruits. C'est pour cela que le jeu s'appellera Tom Noob's blender (cf. l'histoire de Tom Noob).

3.1.3 Objet de l'étude

L'expérience apportée par ce projet nous permettra de voir une nouvelle méthode de progression de notre intelligence artificielle, ainsi que de créer des interfaces plus dynamiques et plus développées que celles que nous avons déjà eu l'occasion de réaliser avec SDL.

3.1.4 Découpage du projet

— Site web : Cette partie englobe tout le travail qui est fait sur le site web du projet. Il sera maintenu à jour au fil du temps.

— Structure du réseau de neurones : cette partie va traiter de l'organisation du réseau de neurones. Pour le fonctionnement de ce projet il est essentiel d'avoir un réseau général et pas spécialisé, il doit pouvoir prendre n'importe quelle forme avec un nombre de couches variant.

— Entraînement du réseau de neurones : dans cette partie on trouve simplement la façon d'entraîner le réseau et de lui apprendre sa tâche. On notera ici qu'il n'est pas question de backward propagation mais de mutations aléatoires.

— Interface : il s'agit de l'affichage et de l'interaction avec le programme. Cette partie est importante pour avoir un retour visuel sur les actions qui sont effectuées par notre IA.

— Le terrain : la grille de jeu en somme, c'est un peu

le coeur de notre projet car tous les éléments vont se servir des données fournies par le terrain. Il contient aussi une méthode pour donner au réseau de neurones la liste des actions possibles avec les modalités qu'elles présentent.

— Le jeu : c'est le fonctionnement en soit, l'ajout de tous les éléments les uns avec les autres, le fait de rendre la grille dynamique et d'enregistrer un score, d'ajouter une option de Game Over... Bref le ciment entre les morceaux du projet pour qu'ils puissent tous fonctionner sous le même programme.

3.2 État de l'art

Il y a deux méthodes principales pour la résolution de niveaux telle que nous l'entendons. La première n'est pas optimale mais elle peut rester efficace. Nous écrivons nous-même un algorithme qui décide de quel mouvement est le plus rentable en appliquant ce que l'on pense être le mieux ! Bien sûr, la complexité de cet algorithme est bien moindre et en plus il n'est pas nécessaire d'entraîner un réseau mais si jamais notre idée de ce qui est un bon mouvement se

trouve en fait être fausse, alors jamais il n'y aura de bon score par cette méthode.

L'autre méthode connue est de passer par une intelligence artificielle pour décider de quel est le meilleur mouvement possible ! Elle va traiter un grand nombre de parties pour tenter de "comprendre" comment il est possible de faire le meilleur score possible et elle-même s'améliorer avec le temps pour faire de plus en plus de points. Il s'agit de la solution vers laquelle nous avons décidé de nous diriger.

3.3 Les features

Notre projet proposera une belle interface graphique pour montrer les prouesses de l'IA. Le jeu sera intégralement codé pour que l'IA puisse jouer et son but est de faire le plus haut score possible. Pour ce faire, il va d'abord falloir créer notre propre version du jeu Candy Crush pour servir de base de fonctionnement. Ensuite réaliser l'IA puis relier le jeu avec l'IA.

Pour pimenter un peu le tout, nous allons implémenter un fruit spécial qui va détruire tout ceux de la couleur avec laquelle il est lié. Cela nous permettra de mettre un peu de difficulté pour le bot afin qu'il ait plus de mouvements possibles.

Pour le calcul des points, nous allons faire quelque chose de plus complexe que simplement compter le nombre de fruits détruits, il entrera en compte les fruits détruits en réaction en chaîne et le nombre de fruits détruits total au même moment.

3.4 Intérêt algorithmique

Parmi les solutions qui étaient à notre disposition, nous avons choisi de faire une IA qui est supposée résoudre au mieux le puzzle pour avoir le plus de points. La dernière fois pour l'OCR nous avions fait un projet qui utilisait une IA qui déduisait des caractères d'une image, ici nous avons décidé de faire une IA qui va apprendre à jouer à un jeu. Nous avons à notre disposition de nombreuses méthodes d'apprentissage et de développement pour aider notre IA à comprendre la façon d'avoir le meilleur score lors de ses

parties endiablées.

3.4.1 Nos décisions sur l'IA

Pour le moment, nous réfléchissions à utiliser un algorithme qui va apprendre en générant des IA qui vont jouer au jeu, et les meilleurs d'entre elles, celles avec les meilleurs scores, seront gardées pour générer une autre génération issue des meilleures IA. Il nous sera aussi possible de faire un réseau de neurones adapté pour fonctionner avec ce type de jeu.

Pour aider l'IA dans ses choix, nous allons inclure dans le jeu beaucoup d'analyse de terrain, savoir à quoi il ressemble et quels sont les mouvement possibles, une sorte de pré-traitement pour le réseau de neurones... Le but du réseau de neurones sera uniquement de déterminer à quel point un mouvement est un bon mouvement. Il prendra donc en entrée toutes les statistiques que nous aurons effectué sur le pré-traitement (par exemple combien de points rapporte le mouvement, s'il fait un fruit spécial, si le mouvement engendre la possibilité de nouveau mouvement ou en retire...) et en sortie il donnera une statistique pour dire

si c'est une bonne chose à faire ou non.

Son intérêt algorithmique est aussi notable bien que moindre par rapport à un réseau de neurones. Ce prétraitement va demander de faire l'étude d'une grille, probablement sous la forme d'un tableau même s'il peut être envisageable de le traiter comme un graphe (les liens étant entre les voisins). Sachant que cette étude qualitative s'effectuera très souvent, il est important qu'elle soit optimisée afin d'avoir la vitesse de résolution et d'entraînement la plus rapide possible.

3.4.2 L'apprentissage de l'IA

Sur le réseau de neurones de l'OCR nous avions fait une "backward-propagation" qui permet à l'IA de modifier son réseau afin de savoir s'il est bon ou pas. Ici le contexte est différent car nous ne pouvons pas déterminer si oui ou non le mouvement est intelligent, il faut que l'IA fasse elle-même ce travail. Comme vu plus haut, nous pensons faire usage d'un apprentissage par génération, nous prenons les meilleurs d'une génération aléatoire. De cette façon, l'IA évoluera petit à petit mais apprendra de ses erreurs.

Passons à quelques modalités pour cette méthode d'évolution. Nous avons donc décidé d'utiliser plusieurs réseaux dont le nombre de couches et le nombre de neurones seront générés aléatoirement pour donner de la diversité au départ. Par la suite, les meilleurs réseaux resteront et ils leurs seront appliqués des mutations mineures ou majeures (avec un moindre pourcentage de chance) aléatoires en espérant qu'elles finissent par apprendre doucement à avoir le plus de points possible après de nombreuses générations.

3.4.3 La méthode traditionnelle

Les tests dans les jeux sont utilisés pour comprendre l'expérience du joueur et peuvent avoir différentes perspectives, l'équilibrage des difficultés et les tests de collision sont deux exemples courants.

Ajouter un nouveau niveau dans un jeu tel que Candy Crush peut être divisé en trois phases :

— Création : Les level designers utilisent leur imagination et leurs compétences créatives pour créer un nouveau puzzle amusant. C'est là que l'idée centrale du niveau est créée.

— Équilibrage : Les level designers font appel à d'autres personnes ou des sociétés spécialisées dans les tests de jeu pour s'assurer que leur niveau est stimulant. Les concepteurs doivent souvent modifier leur niveau et le peaufiner afin de le rendre agréable pour tout le monde.
— Lancement : Le niveau est publié et disponible dans le monde entier dans le jeu

L'équilibrage est la phase qui prend le plus de temps, mais ce n'est pas celle qui demande le plus de travail de la part des développeurs. Pour la majeure partie du temps, les développeurs attendent les résultats de la part des testeurs qui leur sont nécessaire pour savoir dans quelle but modifier le niveau. Malheureusement, cela prend une semaine à des testeurs humain de tester les nouveaux niveaux. Ce qui fait que les développeurs alternent entre développer un nouveau niveau et modifier un autre.

3.4.4 Les bénéfices des tests autonomes

L'idée derrière les tests autonomes est de créer une IA qui va jouer juste comme un humain va jouer. En lançant ces scripts, il est possible de savoir si un niveau sera ou non adapté et pouvoir l'adapter en fonction pour le rendre plus intéressant à jouer sans avoir à attendre qu'un humain teste les niveau et fasse un retour.

4 Reprise du rapport de soutenance 1

4.1 Site Web

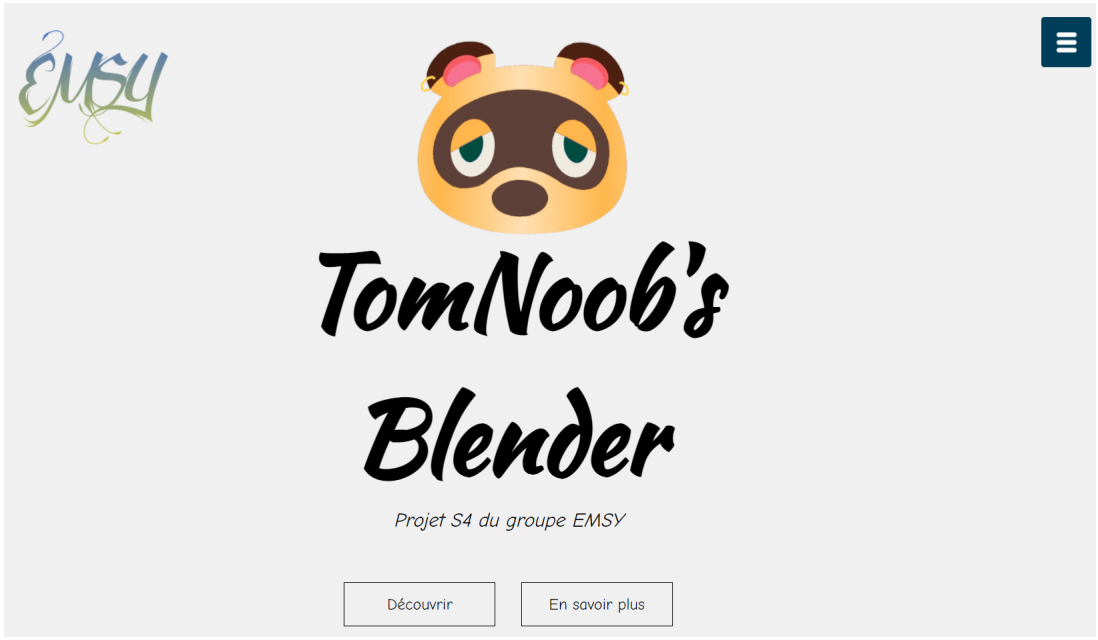
Le site est terminé, en effet le squelette du site est prêt et les documents tels que les rapports et plans de soutenance et manuel d'utilisateur y seront ajoutés plus tard, le cahier des charges en revanche est déjà accessible. Le site sera régulièrement mis à jour pour tenir les utilisateurs au courant des nouveautés et des changements.

Le site est hébergé par GITHUB, le lien est le suivant :

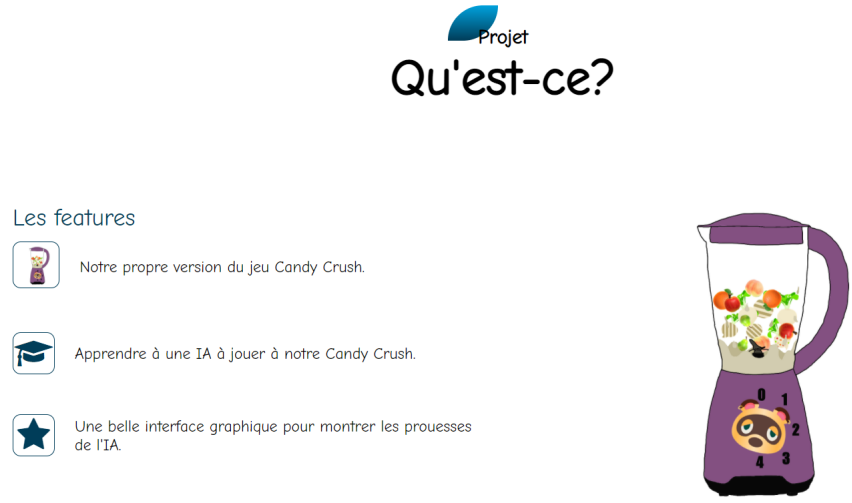
https://epiemsy.github.io.

Le site est composé de quatre parties :

— La bannière

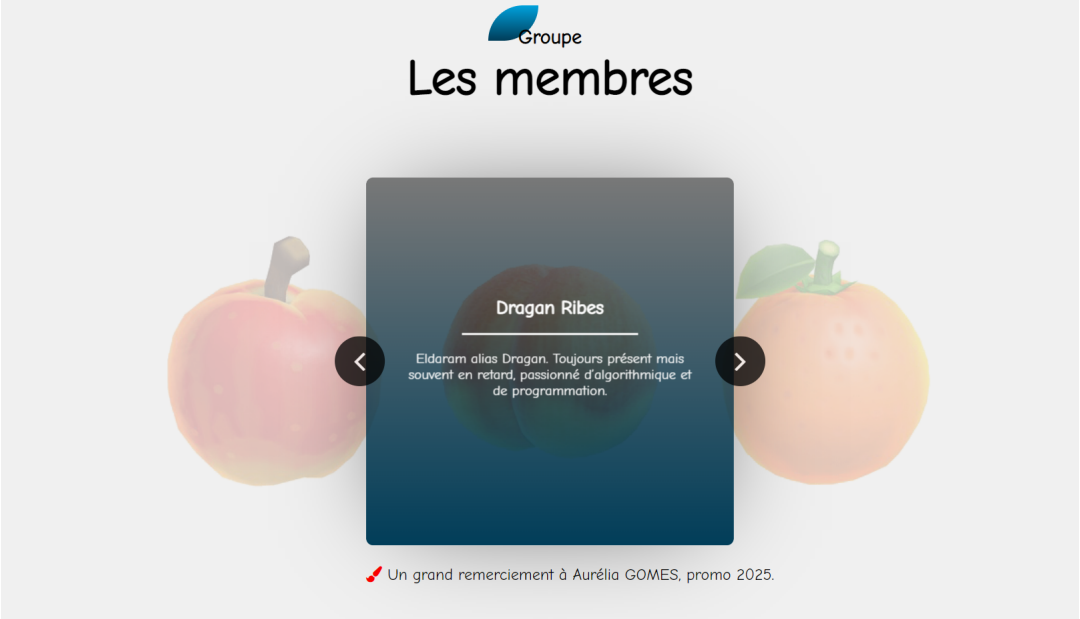


— Le projet avec quelques un de ses features

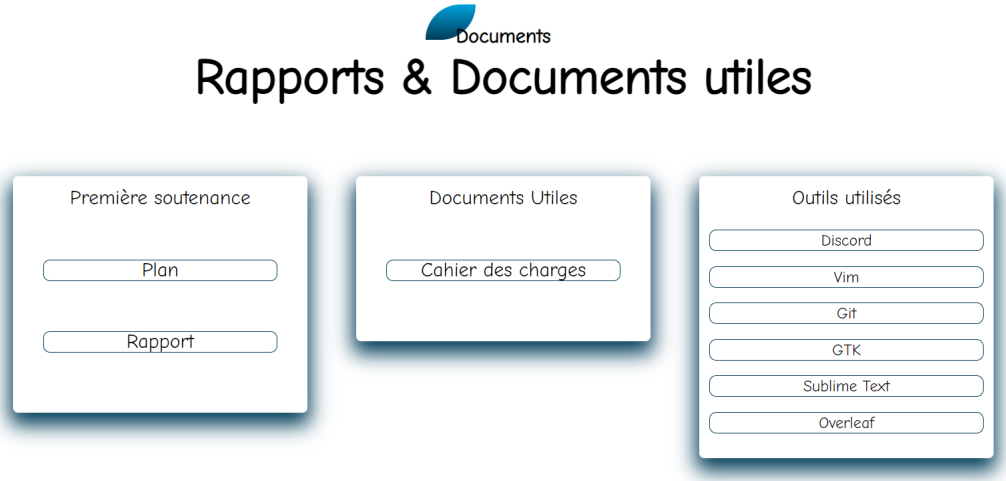


— Le groupe avec une simple présentation de chaque

membre sous forme de carrousel



— Les documents, on peut y retrouver notamment le cahier des charges, plans et rapports de soutenance ainsi que les outils utilisés



Il y a aussi un menu sur la droite qui permet de faire défiler la page jusquà la partie désirée. Du Javascript a été utilisé pour cette fonction, ainsi que pour le carrousel et le menu.

4.2 Réseau de neurones

4.2.1 Structure

Comme promis la structure du réseau de neurones est terminée ! Le but de cette structure est de faire quelque chose de très très modelable car le nombre de couches et de neurones par couche doit pouvoir varier. En réalité, il y a quelques similarités avec l'OCR, mais l'OCR était moins complexe, simplement parce que le réseau de neurones était déjà verrouillé sur un type. Ici nous devons gérer bien plus de variables et il faut bien plus détailler les calculs pour les modifier. Il a aussi fallu créer un nouvel

objet de type Network** qui sert à faire une liste de réseau de neurones, accompagnée d'une fonction qui génère des réseaux de neurones. La structure est ainsi terminée et nous permettra de développer sereinement l'entraînement de ce dernier.

Majoritairement les erreurs rencontrées sont assez communes, beaucoup d'erreurs vis-à-vis des pointeurs. L'utilisation des pointeurs simplifie énormément le travail de création du réseau mais il demande une certaine rigueur et énormément de précisions. L'allocation de mémoire dynamique pose aussi ses propres problèmes, en effet les fonctions comme "Malloc" ou "Free" rendent la gestion de la mémoire nécessaire pour un projet de cette envergure. À l'avenir, il faudra être très précautionneux vis-à-vis de l'entraînement car il y aura de nombreux réseaux de neurones et à chaque génération 90% d'entre eux ne seront plus utilisés, il faudra donc penser à les libérer.

4.2.2 Entraînement

L'entraînement a été commencé, majoritairement en adaptant la procédure "forward propagation", la procédure de prédiction.

4.3 Interface

L'interface à été codée en utilisant la librairie GTK. Le code est donc divisé en deux parties : Un code en C et un code en XML. Je me suis énormément aidé de la documentation GTK disponible sur le site développer.gnome.org pour appréhender la librairie. Pour créer l'interface, j'ai commencé par la construire avec un peu de code XML que j'ai récupéré dans le tp pong réalisé au S3 puis je l'ai modifié pour séparer l'écran en 2 parties aux dimensions définies. Une de ces parties accueille le score et le nombre de fruits détruits et l'autre est une grille représentant le terrain qui accueille les fruits. La grille faite 8 fruits par 8. J'ai ensuite créé les 64 fruits à la main sur XML pour les gérer plus facilement individuellement. Une fois le code XML réalisé, l'interface était prête, seulement tous

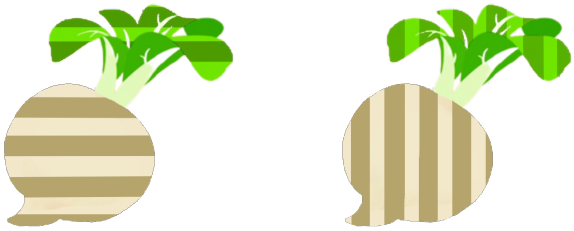
les fruits étaient les mêmes. C'est là qu'intervient le code en C. J'ai récupéré la fonction de Céline qui génère une matrice de int aléatoire, chaque int représentant un fruit différent, puis j'ai créé une matrice de GtkImage auxquelles j'ai modifié le fichier d'image. De la l'interface générait une matrice de fruits aléatoire bien visuelle. J'ai continué en reprenant la fonction de Céline vérifiant si une une matrice possède des enchaînements de plus de trois fruits et je l'ai adapté à ma matrice de GtkImage.

4.4 Jeu

4.4.1 Les fruits

Tout d'abord présentons les fruits normaux sans effet spécial. Pour ce qui est des fruits nous avons choisi la pomme (0), la pêche (1), l'orange (2) et la poire (3).



Le fruit ligne (FL : 5) et le fruit colonne (FC : 6) détruisent respectivement une ligne et une colonne lorsqu'elles sont sélectionnées.

Nous avons ajouté quelques particularités. Lorsqu'un FL et un FL ou un FL et un FC ou un FC et un FC sont sélectionnés, la ligne d'un FC ou FL et la colonne de l'autre fruit sont détruites (ce choix dépend de l'ordre des fruits sélectionnés).

Et enfin le fruit spécial (SF : 4). Ce fruit, lorsqu'il est sélectionné avec un fruit normal, détruit tous les fruits du même type que le fruit normal choisi. Le fruit spécial avec

un FC ou un FL détruit tous les fruits d'un type normal au hasard de la grille. Deux fruits spéciaux sélectionnés détruisent TOUTE la grille.

Pour l'histoire, ces fruits sont des navets qui sont présents dans Animal Crossing : New Horizons. Porcelette les vend tous les dimanches sur notre île. Nous avons demandé à Aurélia GOMES en SUP (l'artiste renommée d'EPITA) de modifier le navet originel pour avoir les fruits colonne, ligne et spécial.

4.4.2 Mécaniques

Les mécaniques comprennent, comme dit précédemment, l'échange des deux fruits sélectionnés, la tombée des fruits après ce mouvement ou encore les fruits spéciaux et leur effet.

Nous allons maintenant lister les fonctions importantes réalisées :

— void generategrid(int matrix[][]) remplit la matrice ma-

trix avec des fruits au hasard choisis entre la pomme, la poire, la pêche et l'orange. Pour cette fonction nous utilisons la fonction rand(), provenant de la librairie stdlib.h, avec % 4 pour avoir un chiffre entre 0 et 3.

— void fruitsfall(int matrix[][], size_t column) fait tomber les fruits de la colonne column après la destruction de fruits. Les fruits du dessus tombent et lorsque tous les fruits sont tombés, de nouveaux fruits sont générés pour compléter le nombre de fruits manquants.

— void destroyline(int matrix[][], size_t line) détruit la ligne line entièrement et appelle la fonction fruitsfall appelée pour chaque case pour régénérer la ligne.

— void destroycolumn(int matrix[][], size_t column) détruit la colonne column entièrement et appelle la fonction fruitsfall pour régénérer la colonne.

— void transSF(int matrix[][], int type) détruit tous les fruits type de la grille et fait appel à la fonction fruitsfall pour faire retomber les fruits lorsqu'un fruit type est détruit.

— int checkaround(int matrix[][], size_t l, size_t c, int condition) vérifie autour du fruit aux coordonnées (l, c). Si une

ligne ou une colonne d'au moins 3 fruits existe alors ces fruits vont être détruits et la fonction fruitsfall va être appelée pour faire tomber les autres fruits. Si la condition est START alors le score ne changera pas malgré les modifications de la grille sinon le score est incrémenté selon le mouvement réalisé et le compteur de fruits détruits change également. Cette fonction retourne TRUE s'il y a eu un changement dans la matrice sinon FALSE.

— void cmd(int matrix[][], size_t l1, size_t c1, size_t l2, size_t c2) va tester les deux fruits sélectionnés. S'ils sont identiques et des fruits normaux alors rien ne se passe. Sinon les deux fruits sélectionnés sont échangés par une fonction swap puis les fruits sont testés et les appels aux fonctions sont faits par rapport aux fruits sélectionnés.

— int checkgrid(int matrix[][], int condition vérifie toute la grille en appelant la fonction checkaround. checkgrid est appelé après un mouvement car la retombée des fruits peut créer des combinaisons. Cette fonction retourne TRUE si un quelconque changement est réa-

lisé sinon elle retourne FALSE.

4.4.3 Combinaisons

SF = fruit spécial FL = fruit ligne FC = fruit colonne

FN = fruit normal (pomme, pêche, poire, orange)

Nous avons fait une liste des combinaisons possibles dans TomNoob's Blender :

- 3 FN = 120 points. Les 3 FN sont détruits.
- 4 FN = 160 points. Les 4 FN sont détruits et FC (ou FL) apparaît si les 4 FN sont sur la même ligne (ou respectivement même colonne).
- 5 FN = 200 points. Les 5 FN sont détruits et SF apparaît.
- FN + SF = 40 * (nb FN) + 240 points. Tous les fruits FN de la grille sont détruits.
- FN + FC/FL = 180 + (nb FN) * 40 + (nb FL) * 40 + (nb FC) * 40 points. La colonne/la ligne des coordonnées d'arrivée du FC/FL.
- SF + FC/FL = 240 + 180 + (nb FN) * 40 points. Tous les fruits d'un type choisi au hasard sont détruits de la grille.

— FC/FL + FL/FC = 180 * 2 + (nb FN) * 40 + (nb SF) * 240 + (nb FC/FL) * 180 points. La ligne et la colonne sont détruites formant donc une croix.

— SF + SF = 240 * 2 + (nb FN) * 40 + (nb FL/FC) * 180 + (nb SF) * 240 points. Toute la grille est détruite et est régénérée entièrement.

5 Reprise du rapport de soutenance 2

5.1 Site Web

Le site n'a pas changé, si ce n'est que le plan ainsi que le rapport de soutenance ont été ajoutés. Dans la partie documents, nous avons ajouté un bloc pour les documents de la deuxième soutenance mais mis à pas cela nous pensions pas que le site nécessite de changement pour le moment.

5.2 Réseau de neurones

5.2.1 Informations sur les mouvements

Pour que le réseau de neurones sache quel mouvement est bien ou non, il faut lui donner des informations pour qu'il puisse évaluer ce qui va lui faire gagner le plus de points. Pour cela, nous avons réalisé une fonction qui va récupérer les informations de tous les mouvements possibles comprenant les coordonnées des fruits à échanger, le score engendré par le mouvement et si le mouvement engendre un SF, un FL ou un FC. Toutes ces informations seront stockées dans une liste contenant des listes avec les

informations de chaque mouvement possible. Cette liste contient des double, c'est-à-dire des flottants pour permettre au réseau de neurones de Dragan de pouvoir utiliser ces informations.

Cette liste (double *info_list) est composée de listes qui contiennent :

- info_list[nb*8 + 0] = numéro de ligne du premier fruit / 10
- info_list[nb*8 + 1] = numéro de colonne du premier fruit / 10
- info_list[nb*8 + 2]= numéro de ligne du deuxième fruit / 10
- info_list[nb*8 + 3] = numéro de colonne du deuxième fruit / 10
- info_list[nb*8 + 4] = score engendré par le mouvement / 10000 (il faut un flottant pour que le réseau de neurones puisse l'utiliser correctement)
- info_list[nb*8 + 5] = 1 si un SF apparaît suite au mouvement sinon 0
- info_list[nb*8 + 6] = 1 si un FC apparaît suite au

mouvement sinon 0

— info_list[nb*8 + 7] = 1 si un FL apparaît suite au mouvement sinon 0

Tout d'abord il faut récupérer tous les mouvements possibles. Pour cela on a une fonction qui va tester pour tout fruit aux coordonnées i et j si ce fruit peut engendrer un mouvement. Si c'est le cas alors la fonction va remplir la liste avec les informations obtenues. Les tests sont très explicites, il va tester tous les cas de mouvements possibles, c'est-à-dire 28 cas. (La fonction est très très longue). Si un mouvement possible est détecté alors la fonction va changer les valeurs de la liste de liste correspondante à ce mouvement.

Pour calculer le score engendré par un mouvement, on a la fonction qui "simule" un swap de fruit. Les fruits tombent, cependant, contrairement à lorsqu'on joue, de nouveaux fruits aléatoire ne tombent pas. Cela permet d'avoir un score minimum obtenu pour le mouvement. Minimum car en fonction des fruits qui tomberont lorsqu'on qu'on fait réellement le mouvement des combinai-

sons peuvent se créer et donc ajouter de nouveaux points. Cette partie n'est pas contrôlable car ces fruits sont générés aléatoirement et nous voulons laisser la chance faire son travail.

Pour éviter de réecrire les fonctions fruitsfall, cmd, parmi d'autres (en réalité presque toutes les fonctions), nous avons ajouté un condition à ces fonctions, qui correspond à si oui ou non nous voulions de nouvelles de nouveaux fruits aléatoire, ainsi les fonctions se font passer la condition.

Il aurait été impossible de ne générer au moins une erreur, non ? Vous voyez où je veux en venir, il y a des situations que nous n'avions pas prévu. Ces problèmes sont décris dans la section "Les problèmes rencontrés".

5.2.2 L'évolution

Elle se passe en trois étapes, la 1ère est une étape d'évaluation. Chaque réseau est évalué individuellement et un score lui est attribué. Pour le moment nous faisons des

tests avec XOR, l'évaluation est donc faite en fonction de la différence qu'a le résultat attendu avec le résultat espérée. Plus la différence est petite, plus le réseau est réussi. Mais quand nous en arriverons à la phase finale, il suffira de changer la méthode d'évaluation. Le réseau de neurones jouera et le score obtenu sera alors la note. Ensuite les réseaux sont triés du plus efficace au moins efficace.

Finalement on applique les diverses modifications au réseau, le but est de faire des modifications aléatoires. Pour cela on a deux opérations, la mutation et le croisement qui sont détaillés un peu plus bas dans ce rapport. Dans les détails de la phase de modifications, le nombre de réseaux va être découpé en 3 parties, 10% d'entre eux seront considérés comme les meilleures réseaux, 10% seront les moins efficaces donc les derniers et les 80% restants. Les meilleurs réseaux sont conservé tels quels dans la génération suivante, pour pouvoir piocher des réseaux fonctionnels. Les derniers réseaux, eux, sont supprimés et remplacés par des réseaux tout nouveaux pour avoir de nouveaux réseaux utilisables pour les croisements, pour ajouter de la diversité aux réseaux. Ensuite, le reste des réseaux est déterminé en

piochant au hasard dans les réseaux de la précédente génération. Les meilleurs réseaux ont plus de chance d'être choisis que les autres pour la mutation et le croisement, afin d'avoir un résultat plus proche de quelque chose de fonctionnel. Après que toute la génération soit remplie, le cycle évolutif recommence.

5.2.3 La mutation

Pour la première soutenance, nous avions présenté une fonction nommée "mutate" qui fait muter aléatoirement une matrice. Nous avons maintenant utilisé cette fonction pour faire muter un réseau de neurones. Nous partons du principe que la mutation se fait uniquement sur les matrices poids et biais pour les matrices type F, les matrices A, elles, sont simplement régénérées. Pour cette fonction, elle fut assez simple à réaliser mais son application reste très limité.

5.2.4 Les croisements

Contrairement à la mutation, il est en général plus probable de pouvoir utiliser des croisements pour avoir des

réseaux performants. Le but du croisement est de mélanger deux réseaux de neurones ensemble en espérant obtenir un réseau plus avancé et mieux construit sur la base de deux autres réseaux. Il est probable que le croisement soit un véritable avancement pour l'apprentissage évolutif mais, même si la fonction "cross" marche correctement, nous rencontrons quelques problèmes à faire marcher proprement l'arithmétique des pointeurs qui va avec. En réalité, toute les fonctions qui en découlent fonctionnent aussi, mais elles se heurtent à quelques problèmes sur l'espace mémoire et la libérations des précédents réseaux de neurones. Il faudra donc faire un peu de ménage avant de pouvoir utiliser de façon optimale toutes les possibilités qu'offrent le croisement.

5.3 Interface & terrain

Pour réaliser le drag and drop, j'ai dû utiliser un aspect de GTK que j'avais très peu utilisé jusqu'alors durant ce projet : les signaux. Le terrain étant composé de GTKImages, cliquer sur celles-ci ne produisait aucun signal. J'ai donc rajouté un GTKEventbox par dessus le GTKGrid

contenant les images, ce qui a eu pour effet de détecter les clics dans l'espace de la grille.

J'ai ensuite utilisé les coordonnées de la souris ainsi que les signaux on-click et on-release des GTKButton pour rendre le drag and drop fonctionnel. Pour ça, j'ai créé une variable globale de type int*, qui allait contenir les coordonnées des cases à échanger, puis j'ai créé des fonctions qui se font appeler chaque fois que les signaux on-click et on-release apparaissent. Ces fonctions stockent alors les coordonnées de la souris au moment du click et au moment où on relâche le clic dans la liste.

Il ne me restait plus qu'à utiliser les fonctions de Céline et Katia pour rendre le jeu fonctionnel. En créant cette liste de coordonnées, j'avais facilement accès aux fruits à échanger. Et avec mes précédentes fonctions, j'avais facilement accès à la matrice de int représentant le plateau. Le drag and drop est alors fonctionnel.

5.4 Jeu (Candy Crush)

Le plus gros a été fait pour la première soutenance, il ne restait qu'à corriger les bugs. Par exemple lorsqu'un fruit spécial était échangé vers le haut avec un fruit normal, le fruit normal ne disparaissait pas. Plusieurs bugs identiques étaient présents et nous avons dû trouver leur origine et les corriger.

De plus certaines fonctions qui avaient été réalisées pour la première soutenance n'étaient pas explicites rendant leur compréhension à la lecture confuses (cf. le rapport de soutenance 1). C'était notamment le cas de la fonction cmd qui comprenait de nombreux if dans un switch et aussi un switch dans un switch. Nous avons décidé de pallier cette confusion en supprimant les switchs et en les remplaçant par des simples if pour chaque cas possible. Pour rappel, la fonction cmd teste le type des deux fruits sélectionnés et fait appel aux fonctions correspondantes à leur type. Après ce changement, la fonction est beaucoup plus longue mais plus explicite pour, vous, le jury à lire.

5.5 Les problèmes rencontrés

5.5.1 Le réseau de neurones

Nous avons rencontré au moins une infinité de problèmes durant la conception de l'évolution pour le réseau de neurones. Le 1er d'entre eux, et un problème que nous rencontrons encore est la gestion de la mémoire. Vous imaginez bien que créer des générations de 500 réseaux de neurones environ 1000 fois pour les entraîner est relativement coûteux en espace mémoire. Il faut donc faire très attention à la gestion de la mémoire, à l'ensemble des réservations qui sont faites pour le réseau de neurones et à où ces références de réservations sont stockées.

Dans la même veine de problèmes, peu de problèmes sur le réseau en soi ont été rencontrés. Mais il y avait un moment où l'algorithme de notation se basait sur la matrice de sortie de la 1ère couche au lieu de travailler avec les résultats de la dernière couche du réseau de neurones. Cela rendait assez évidemment l'entraînement du réseau de neurones très complexe car il ne lisait pas du tout les bonnes valeurs de sortie.

Pour faire de la résolution de bug dans le réseau de neurones, j'ai effectué beaucoup de changements internes, dont un "léger" changement sur la compilation pour avoir des codes d'erreur plus précis (comme par exemple l'ajout du flag "fsanitize=address"), l'erreur que j'ai eu n'était pas en C, mais dans le Makefile en lui-même. En réalité il y avait bien une variable avec tous les flags de compilation mais la commande de compilation était exécutée sans ces "flags" rendant l'identification des lignes impossible par exemple (pas le flag -g). Suite à la résolution de ce petit problème, nous avons pu régler de nombreux petits problèmes au sein du programme, comme par exemple les boucles "for" faisait des comparaisons entre des "int" (la valeur qui grandit entre chaque itération) et des "size_t" (la valeur qui correspond à la "taille" d'un réseau, son nombre de couches par exemple).

5.5.2 Informations sur les mouvements

Les modifications faites sur les fonctions précédemment implémentées nous ont fait comprendre que nous n'avions

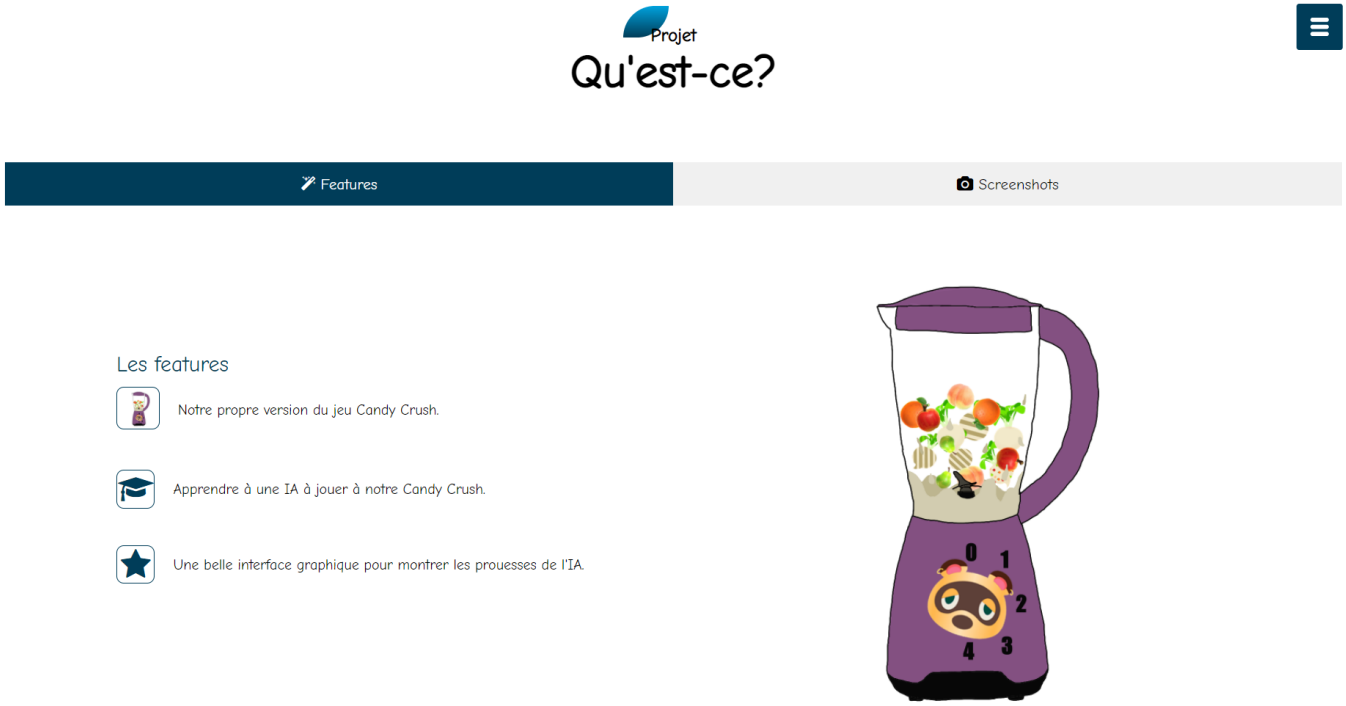
pas pensé à tous les cas possibles. Par exemple, un alignement de plus de trois "cases" vides qui a provoqué une boucle infinie, que nous avons heureusement rapidement corrigé.

Un problème où nous avons eu plus de mal à corriger est dû au type. Le score est un entier non signé que nous avons stocké dans un tableau d'entiers. Originellement, nous avions cast lors de l'affectation au tableau, mais le programme n'était pas content et nous donnait des 0. La solution est de cast lors du return de la fonction simulation.

6 Avancée globale

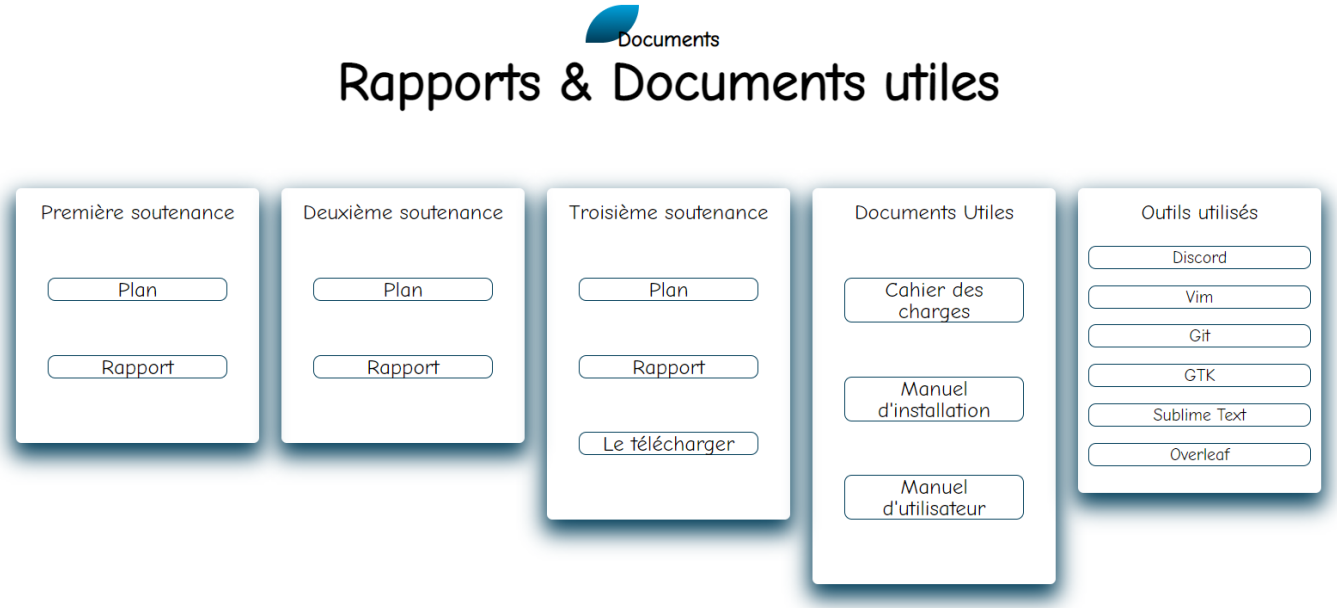
6.1 Site Web

Un ajout récent est un onglet pour montrer les screenshots de notre jeux dans la partie Projet ainsi qu'une vidéo du réseau de neurones faisant une partie qui sert de teaser.



Bien sûr il y a l'ajout des documents, je parle de ceux pour la troisième soutenance (plan et rapport de soutenance) mais aussi du lien pour télécharger le fichier zip qui contient le projet ainsi que des manuels d'utilisateur

et de compilation.



6.2 Réseau de neurones

Pour lier le réseau de neurones au jeu, on a utilisé des fonctions similaires à celles utilisées pour le XOR. L'idée est toujours d'évaluer chaque neurone auquel on a donné les informations d'un mouvement, et ce pour chaque mouvement possible, récupérées par save_info (précédemment le main du fichier informations.c).

La fonction d'évaluation se répartit en simples étapes :

— Étape 1 : on récupère les informations concernant les mouvements possibles
— Étape 2 : on donne les informations d'***un*** mouvement à chaque neurone (chaque mouvement est différent)
— Étape 3 : on trie les neurones pour récuperer celui avec le meilleur score
— Étape 4 : on effectue ce mouvement
— Étape 5 : on répète depuis l'Étape 1 jusqu'à ce qu'il n'y ait plus de mouvement possible (on parle ici de la limite de mouvement que nous avons mis en place)
— Étape 6 : On récupère le score final

6.2.1 Save and load

Nous avons donc ajouté deux fonctions pour permettre à notre réseau de neurones d'être entraîné sur différentes sessions. Le save fonctionne en enregistrant dans un fichier différent chaque matrice utilisée par le réseau de neurones et enregistre dans un autre fichier l'architecture générale du réseau de neurones. Cela utilise des nouvelles librairies et fonctions de sauvegarde de variable sous format binaire. La fonction load ne fait que fabriquer un réseau de neu-

rones dont les couches sont composées des différents fichiers binaires sauvegardés précédemment. Load est aussi capable de savoir s'il y a ou non réseau de neurones présent, s'il n'y en a pas, elle génère un nouveau réseau de neurones qu'elle renvoie après.

Il y a eu de nombreuses erreurs durant la création de ces deux fonctions, et ce fut assez difficile de se décider sur les divers formats d'enregistrement pour le réseau de neurones. Pour les fichiers qui enregistrent les matrices du réseau de neurones, la solution qui nous a paru la plus simple a été de simplement faire des fichiers binaires. Cependant pour la création du fichier contenant les informations sur le réseau de neurones nous a demandé plus de réflexion. Au début nous avions pensé à utiliser un fichier texte au format json, mais finalement nous avons utilisé un fichier texte qui contient "couche*3+1" lignes.

6.3 Interface

Une fois l'interface créée et le drag and drop fonctionnel, il me restait encore quelques améliorations à apporter à l'interface. Tout d'abord, la partie gauche de l'interface

(le score) n'étant pas implémentée, j'ai commencé par implémenter le score, le compteur de fruits détruits et enfin le meilleur score. Pour ce faire, j'ai récupéré le score qui s'actualisait déjà grâce aux fonctions de Céline et Katia. Puis à l'aide des fonctions asprintf() et gtk_label_set_text(), j'ai créé des nouveaux labels pour les fruits normaux et fruits spéciaux qui s'actualisent à chaque mouvement.

Pour ce qui est du meilleur score, il est stocké dans un fichier extérieur, bestscore. J'ai ensuite créé une fenêtre de fin de partie, la fenêtre game over, qui affiche le score obtenu lors de la partie ainsi que le meilleur score. Si le score obtenu est meilleur que le best score, celui-ci est actualisé à l'appui du bouton OK. J'ai ensuite créé un indicateur permettant de savoir quand le joueur peut jouer et quand le jeu est en train de faire tomber des nouveaux fruits. En effet, j'ai ajouté une GtkImage contenant la tête de Tom Noob que j'ai rendu invisible. Après cela j'ai modifié ma fonction movement() pour qu'elle affiche cette image lorsque le jeu fait tomber de nouveaux fruits.

J'ai ensuite ajouté un nombre de mouvements pour que les parties puissent se finir et j'ai créé un compteur sur l'interface pour que le joueur puisse savoir combien de coups il lui reste. Une fois tout ceci fait, j'ai relié le jeu avec le réseau de neurones de Dragan. Pour ce faire, j'ai créé une nouvelle fenêtre s'affichant au lancement de l'application possédant 4 boutons :

- Le bouton **Player mode** pour que l'utilisateur joue.
- Le bouton **AI mode** pour que l'IA ayant un algorithme bruteforce joue
- Le bouton **NN mode** pour que le réseau de neurones joue
- Le bouton **Train NN** pour entraîner le réseau de neurones.

J'ai ensuite lié ces boutons aux diverses fonctions que m'ont fourni Céline, Katia et Dragan pour que les différents modes soient fonctionnels. Il m'a fallu créer encore une nouvelle fenêtre pour la fonction train NN. Cette fenêtre contient deux GtkInput pour que l'utilisateur puisse rentrer le nombre de réseaux de neurones à entraîner ainsi

que le nombre de sessions d'entraînement. La fenêtre contient également un bouton pour lancer l'entraînement et des labels pour indiquer à l'utilisateur que le réseau est entraîné. Une fois tout ceci fait, l'interface était fonctionnelle et permettait au joueur de choisir les différents modes de jeu.

6.4 Jeu

Pour finaliser ce projet, nous avons décidé de rendre le jeu plus visuel. Nous avons modifié l'affichage du jeu. Chaque fois qu'une combinaison est détruite, l'interface est actualisée pour comprendre ce qui se passe. Avant l'ajout de cette mécanique, l'interface ne s'actualisait pas à chaque combinaison. C'était donc impossible de comprendre d'où venaient les points et les combinaisons détruites qui s'enchaînent suite à un mouvement.

7 Problèmes rencontrés

L'un des plus gros problèmes rencontrés concernant le jeu était d'afficher les étapes, c'est-à-dire de montrer à chaque fois les fruits se détruire au fur et à mesure et

pas juste disparaître. Ce mécanique devait être implémenté pour permettre au joueur de bien visualiser d'où proviennent les points qu'il gagne. De plus cela rendait le jeu plus compréhensible, en effet avant l'ajout de cette mécanique les combinaisons disparaissaient tout simplement. Cela rendait le jeu moins visuel et donc moins compréhensible. En raison de la richesse de la librairie GTK et de sa man page, cela nous a pris beaucoup de temps. Nous avons tenté beaucoup de choses pour enfin réussir à ajouter cette mécanique grâce à la fonction gtk_events_pending qui renvoit un booléen en fonction de si l'interface a des instructions en attente ce qui nous permet d'ajouter un délai afin de pouvoir voir les étapes.

Nous avons également rencontré de nombreux problèmes lorsqu'il fallait lier le jeu avec le réseau de neurones. En effet, il a fallu réunir les deux dossiers, CandyCrush et NeuralNetwork, pour pouvoir compiler le projet ensemble. Il fallait simuler l'échange de deux fruits au lieu d'utiliser le drag and drop.

8 Chefs d'œuvre

Un grand remerciement à Aurélia GOMES qui est actuellement dans la promo 2025 en SUP. Nous la connaissons depuis plus d'un an, malheureusement nos chemins se sont séparés dû à son redoublement. Cependant nous restons toujours en contact car c'est une très bonne amie pour tous les membres du groupe. Il n'est pas faux de dire que Aurélia fait partie de l'équipe EMSY. En effet, la majorité des dessins qui sont présents dans notre projet viennent d'elle. Nous avons suscité ses talents et elle a accepté volontiers.

Voici tous les dessins qu'elle a réalisés :

Bon courage à elle pour cette année!!!

9 Planning

Quelques modifications ont été faites quant à la répartition des tâches.

C = Commencé B = Bien avancé T = Terminé

P = Participation (correction/ajustement)

9.1 Soutenance 1

Tâches	Dragan	Kenjy	Katia	Céline
Site Web			T	P
Structure du réseau de neurones	T		T	
Entraînement du réseau de neurones	B		B	
Interface		T		P
Le terrain		C		C
Le jeu			B	B

9.2 Soutenance 2

Tâches	Dragan	Kenjy	Katia	Céline
Site Web			T	P
Structure du réseau de neurones	T		T	
Entraînement du réseau de neurones	B		B	
Lien réseau / jeu			B	B
Interface		T		P
Le terrain		B		B
Le jeu			T	T

9.3 Soutenance 3

Tâches	Dragan	Kenjy	Katia	Céline
Site Web			T	P
Structure du réseau de neurones	T		T	
Entraînement du réseau de neurones	T		T	
Lien réseau / jeu			T	T
Interface		T		P
Le terrain		T		T
Le jeu			T	T

10 Conclusion

Pour finir cette année riche en émotions, voici TomNoob's Blender terminé. En résumé de ce qui a été fait, nous avons un site internet regroupant toutes les informations sur le jeu, sur l'équipe EMSY et aussi les documents réalisés lors de ce projet. Une interface a été réalisée pour permettre au joueur de profiter pleinement de son expérience TomNoob's Blender mais également de pouvoir voir une intelligence artificielle jouer. Pour réaliser la première fonctionnalité qui est de jouer, nous avons donc implémenté des mécaniques similaires au jeu Candy Crush. Et pour la deuxième fonctionnalité qui est de laisser une intelligence artificielle jouer, nous avons repris les mécaniques réalisées avant et avons appris au réseau de neurones quelles étaient les meilleures combinaisons qui rapporteraient le plus de points.

Le but étant d'avoir le meilleur score, qui de l'humain ou de la machine sera meilleur ?