Rapport du projet d’animation

TAG – Équipe n°4 (G1)

# Introduction au sujet

Dans le cadre du module M3202C, notre groupe a été divisé en plusieurs équipes avec pour objectif la réalisation d’un programme à base de mathématiques codé en

Python, devant fonctionner avec le logiciel SageMath.

# Présentation du projet

Tout d’abord nous avons choisi de réaliser une animation 2D car un rendu graphique sera toujours plus agréable à regarder que de simples lignes de code. Ainsi, nous avons eu l’idée d’un programme qui génèrerait une image affichant un mot saisit par l’utilisateur au clavier, lettre par lettre, sous forme de graffiti avec la couleur souhaitée.

# Aspects techniques du projet

## Outils utilisés

* **Python (Langage de programmation) :** langage compatible avec SageMath, utilisé afin de coder notre projet.
* **SageMath (Logiciel) :** compilation de notre code en Python afin de générer l’animation.
* **Geogebra (Application web) :** obtenir un rendu graphique pour chaque lettre avant de commencer à les coder une par une.
* **Format d’image APNG (Animated Portable Network Graphics) :** format d’export de notre animation, plus efficace et de meilleure qualité que le format habituel GIF.

## Structure du projet

**Étapes de fonctionnement :**

* Import du module « sys » permettant de choisir le système d’encodage par défaut, ici UTF-8 (à décommenter seulement en cas de problème de compilation).
* Initialisation de la fonction « showInfos » qui affiche sur la console les informations à prendre en compte pour les différents paramètres que l’utilisateur devra saisir au clavier.
* Initialisation de la fonction « setDemande » permettant à l’utilisateur de saisir au clavier le mot qu’il souhaite en boucle tant que le mot saisit n’est pas valide (seulement les lettres de l’alphabet sont acceptées), la boucle se termine lorsque la valeur du compteur « cpt » est égale à la longueur de la chaine saisie au clavier. Elle est ensuite stockée dans la variable globale nommée « demande ».
* Initialisation de la fonction « checkDemande » prenant en paramètre un caractère, qui analyse le code ASCII (au format décimal) de celui-ci afin d’en déduire s’il appartient bien à l’alphabet ou pas (sachant que les caractères minuscules de l’alphabet ont un code ASCII compris entre 97 et 122). Elle retourne 1 lorsque c’est valide, 0 dans le cas contraire, puis cette valeur sera additionnée au compteur de la fonction « setDemande ».
* Initialisation de la fonction « setColor » permettant à l’utilisateur de saisir au clavier la couleur qu’il souhaite, autant de fois qu’il le faudra. La couleur sera stockée dans la variable globale « printColor ».
* Initialisation de la fonction « checkColor » qui vérifie une chaine de caractères en paramètre, afin de savoir si elle appartient dans un tableau « colors » contenant l’ensemble des couleurs disponibles. Si la couleur est bien contenue dans le tableau, alors elle renvoie la même chaine passée en paramètre, sinon elle renvoie une chaine qui n’est pas contenue dans le tableau, ici « 0 ».
* Initialisation de la fonction « setSize » qui prend en paramètre un booléen, permet de définir l’épaisseur du point pour l’affichage, s’il vaut vrai, dans ce cas-là elle retourne la valeur initiale définie pour « pointSize », dans le cas contraire elle renvoie 0, afin que le point soit invisible.
* Initialisation de la fonction « getScaling » qui gère les paramètres d’affichage de l’animation, elle initialise la variable globale « pointSize » qui est la taille du point affiché et « scale » qui est l’échelle de l’affichage pour l’animation (définie à 2 après plusieurs tests, de préférence il ne vaut mieux pas y toucher car il risque d’y avoir des problèmes à cause du format APNG n’acceptant seulement des images au format [1:1] de même taille). La fonction retourne les valeurs maximales et minimales en abscisse et en ordonnée ainsi que l’échelle du graphique en fonction de la longueur du mot saisit.
* Initialisation de la fonction « getSpace » prenant en paramètre l’index du caractère dans la chaine afin d’obtenir sa position en abscisse. Retourne le résultat du calcul effectué permettant à chaque lettre d’être positionnée de telle sorte à ce que le mot soit centré dans l’animation finale.
* Initialisation de la fonction « getTransition » prenant en compte trois paramètres, « lastframe » qui est la dernière image du tableau contenant l’ensemble des images pour l’animation, « begin » un tableau contenant les coordonnées (x,y) du dernier point de la lettre venant d’être tracée et « end » un second tableau contenant les coordonnées (x,y) du premier point de la prochaine lettre à tracer. Elle va calculer le coefficient directeur « a » et ensuite l’ordonnée à l’origine « b » de la fonction affine qui sera tracée dans la transition entre chaque lettre. Elle retourne le tableau d’images contenant les images du tracé de la fonction point par point.
* Initialisation de la fonction « tag » prenant en compte deux paramètres qui sont les coordonnées (x,y) du point de base sur lequel chaque élément est positionné par rapport à celui-ci. La bombe de peinture est composée de différents éléments comme un polygone, une ellipse et un cercle. Retourne un polygone.
* Initialisation de la fonction « tagspraying » qui est une copie de la fonction « tag » en y ajoutant un polygone en plus, imitant un spray de peinture, utilisée lors du tracé d’une lettre. Retourne également un polygone.
* Initialisation de la fonction « DroiteH » prenant six paramètres en compte dont la dernière image du tableau contenant toutes les images, la valeur minimale et maximale en abscisse, le pas définissant le nombre d’images qui seront générées pour le tracé, l’ordonnée et la valeur booléenne permettant de choisir si l’on affiche le tracé ou pas. Pour chaque valeur de « x » dans l’intervalle, création d’une image contenant le contenu de l’ancienne image et ajout d’un point aux coordonnées définies. On sauvegarde la nouvelle image créée dans la variable de l’ancienne image. Si le booléen « visible » vaut vrai alors on ajoute la bombe de peinture avec l’imitation de spray dans l’image sinon on rajoute simplement la bombe sans spray. L’image est ensuite ajoutée dans le tableau d’images de la fonction. Une fois la boucle terminée on rajoute en dernière position l’image du dernier point sans la bombe de peinture. Retourne ensuite le tableau.
* Initialisation de la fonction « DroiteV » qui est une copie de la fonction « DroiteH » en y inversant les coordonnées (x,y) afin d’obtenir une droite verticale.
* Initialisation de la fonction « getBDiag » permettant d’obtenir l’ordonnée à l’origine d’une fonction affine pour n’importe quel point en abscisse, prend en paramètre le coefficient directeur, l’ordonnée à l’origine initiale et le nouveau point de départ en abscisse. Retourne le calcul effectué.
* Initialisation de la fonction « Diagonale » qui ressemble fortement à la fonction « DroiteH » mais les coordonnées du point seront définies grâce à une fonction affine (ax+b).
* Initialisation de la fonction « getXdegree » avec trois paramètres, le premier étant un angle, le deuxième étant l’échelle pour l’abscisse et le troisième, la position en abscisse. Retourne la coordonnée en abscisse.
* Initialisation de la fonction « getYdegree » comme la fonction précédente mais retourne la coordonnée en ordonnée.
* Initialisation de la fonction « Courbe » qui ressemble à la fonction « DroiteH » sauf que cette fois ci nous utilisons une fonction trigonométrique. Retourne le tableau d’images de la courbe.
* Initialisation de la fonction « Animate » qui prend en paramètre le tableau contenant l’ensemble des images pour l’animation, les paramètres d’affichage du graphique retournés par la fonction « getScaling », le délai d’affichage entre chaque image et le nombre d’itérations (répétitions) de l’animation. Estime le temps de génération, créée l’animation et la sauvegarde au format APNG sous le nom du mot saisi par l’utilisateur.
* Initialisation de la fonction « main » qui affiche les informations à l’écran, demande à l’utilisateur le mot, la couleur et le nombre de répétitions pour l’animation qu’il souhaite. Création d’un tableau « frames » qui contiendra l’ensemble des images pour la génération de l’animation. On ajoute au tableau l’image initiale « tmpframe » afin que le tableau soit initialisé en tant que tableau d’images. Pour chaque caractère dans « demande », on récupère dans le tableau « activeLetter » les valeurs retournées par la fonction appelée par rapport au caractère actuel, si c’est le premier caractère, il n’y a pas besoin de transition, on initialise alors « tmpEnd » qui va stocker les coordonnées (x,y) du dernier point du caractère actuel, « letterFrames » qui récupère les images pour l’animation du caractère, « tmpframe » qui prend la dernière image de « letterFrames » ne contenant pas de bombe de peinture, on supprime juste après cette image afin qu’il n’y est pas de problème de liaison entre chaque ensemble d’images retourné par chaque fonction de caractère et enfin on ajoute les images de la lettre actuelle au tableau contenant l’ensemble des images. Si le mot saisit par l’utilisateur est plus d’un caractère dans ce cas-là, avant d’afficher le caractère suivant on insère une transition. Une fois la boucle terminée, on rajoute la dernière image du dernier caractère sans bombe de peinture pour qu’à la fin de l’animation, on ne voit plus la bombe. Retourne le tableau d’images « frames ».
* Initialisation de toutes les fonctions concernant chaque caractère. Retourne toutes un tableau d’images contenant le tracé du caractère.
* Définition du tableau « functions » contenant la fonction propre à chaque caractère.
* Appel de la fonction « Animate » qui appelle la fonction « main » pour générer l’animation.

**Informations complémentaires :**

* La génération de l’image peut prendre un certain temps, plus il y a de lettres, donc plus il y aura d’images à générer (sachant que le temps en moyenne pour la génération de l’animation d’une lettre est compris entre 30 secondes et 1 minute).
* Le code est également commenté pour voir en détail ce qu’il se passe dans le programme.