

Rapport projet KWYK

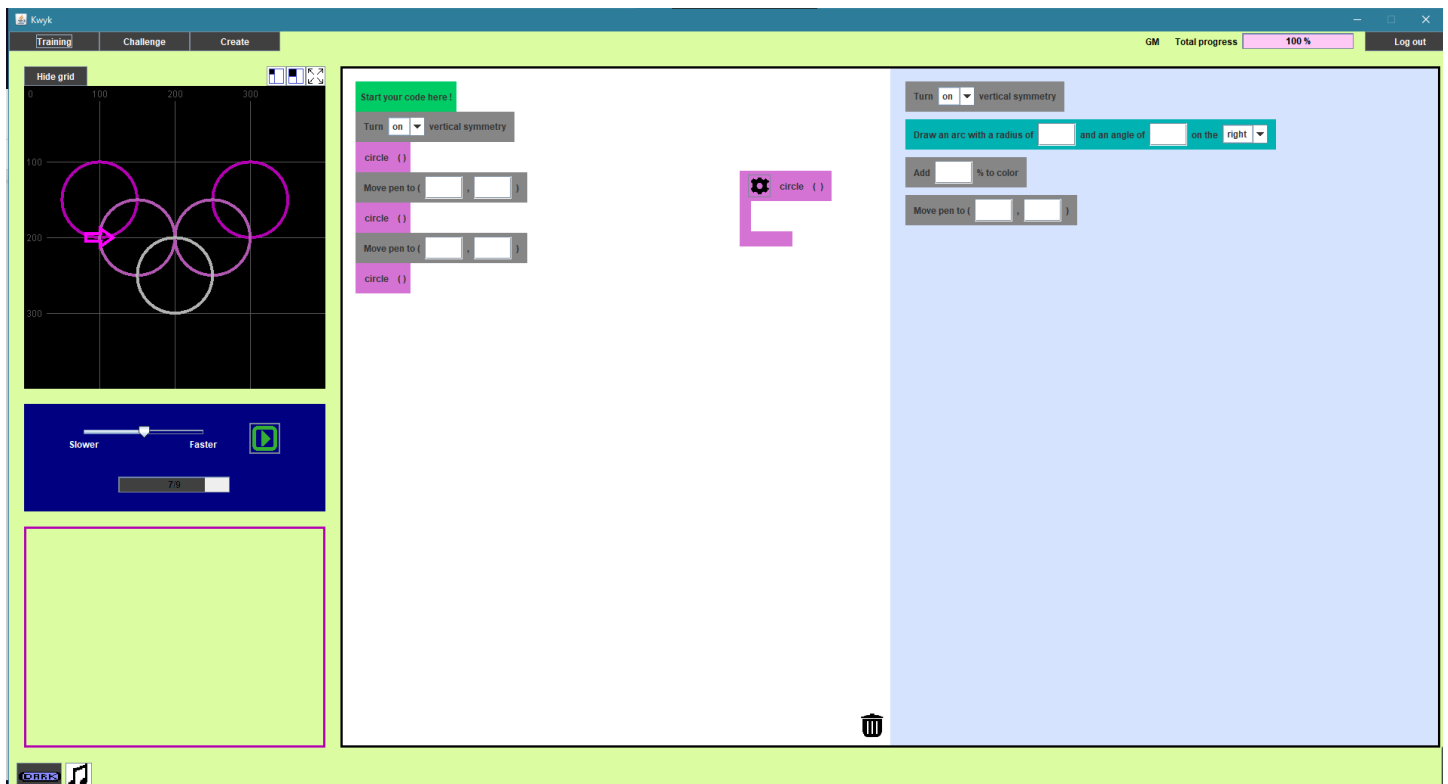
Etudiants en charge du projet :

- SAMSON Pierre 21956843
- DAI Anna 21953144
- RABEHANTA Line 21960168
- SAILLY Steven 21954116

Cahier des charges : https://docs.google.com/document/d/1sz2GuABkuvMOMiyGI-5VNK7ukjiEI1cBDi0V4gH_9ZQ/edit?usp=sharing

Description du projet :

Notre projet se base sur *Kwyk*, *Algoblocs* ainsi que *Scratch*. Le but est de créer un jeu pédagogique permettant l'apprentissage de la programmation, via des dessins à reproduire, grâce à un langage en bloc.



Un niveau du jeu, sur le thème des fonctions

Présentation détaillée :

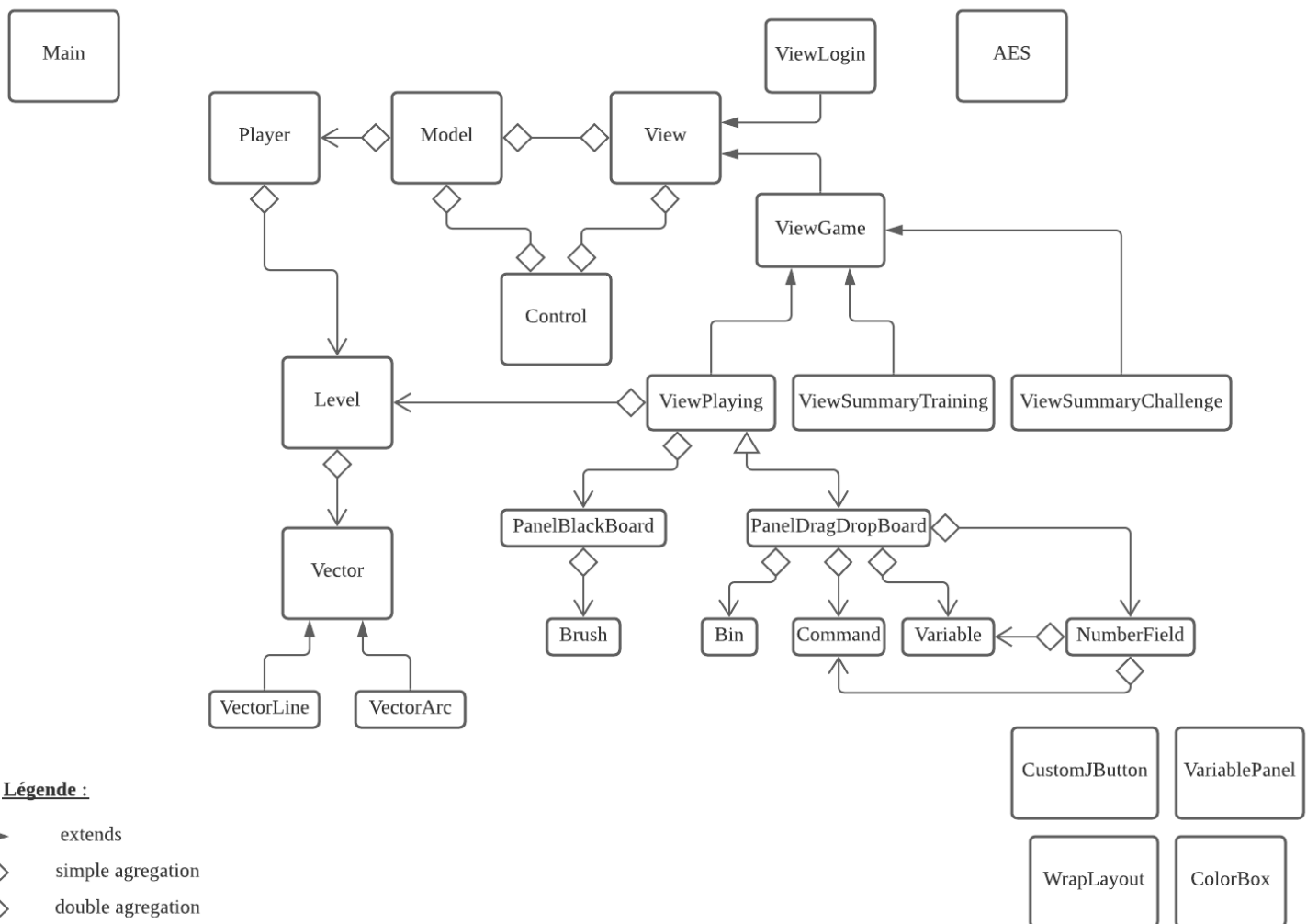


Diagramme des classes

Joueur :

- Système de multi-comptes avec deux comptes spéciaux : GM (administrateur)¹ et default, qui sont créés à chaque démarrage du jeu (pour éviter les problèmes de suppression).
- Le joueur se crée un compte et/ou se connecte grâce à la page View Login.
- Les mots de passe sont cryptés via AES (Advanced Encryption Standard).
- Les données du joueur sont serialisées dans un fichier .player.
- La progression du joueur est mise à jour au moment de la réussite d'un niveau grâce à la fonction save() : le fichier du joueur est écrasé par un nouveau fichier à jour à chaque appel de la fonction.
- Nous utilisons une `LinkedList<String>` pour sauvegarder les challenges réussis (sauvegarde du nom) et un `boolean[][]` pour les training (met à true le boolean suivant).

¹ Mot de passe : Azozo

Niveaux :

- Chaque niveau est composé d'un fichier .lvl (objet de la classe Level) et d'une image (qui représente le dessin à reproduire), respectivement sauvegardés dans les répertoires levels/ et preview/.
- Pour le jeu, les niveaux sont séparés en deux catégories :
 - D'une part, **Training** où les niveaux sont rangés par catégories pour apprendre la programmation en faisant découvrir les différentes options petit à petit. Ici, le joueur doit faire les niveaux dans l'ordre que nous avons établi (éventuellement dans plusieurs catégories différentes).
 - D'autre part, **Challenge** qui contient des niveaux plus difficiles, et les niveaux créés par les joueurs. Ici, le joueur peut faire n'importe quel niveau sans ordre prédéfini, et nous y introduisons une présentation randomisée, pour faire varier les niveaux en tête de page.
- Pour la création d'un niveau, nous faisons une différence entre le GM, le joueur lambda, et le compte default :
 - Le GM a plus d'options pour la création. Il peut choisir la position initiale du pinceau et sa couleur, enregistrer du code et des fonctions (pour créer les Training), et choisir le dossier dans lequel sera sauvegardé le niveau.
 - Le joueur lambda, quant à lui, ne peut que sauvegarder ses niveaux dans Challenge, et son pinceau aura des attributs par défaut.
 - Le compte default ne peut, en revanche, pas créer de niveau.
- La création des niveaux se passe sur la page **Create** :
 - Le joueur crée son niveau et peut ensuite le submit.
 - Si le dessin du joueur est vide ou que le code est en train de run, le bouton submit est désactivé.
 - Une fenêtre apparaît ensuite pour laisser le joueur choisir un nom pour son niveau. Ce nom doit respecter des normes, puisqu'il sera aussi intégré dans le nom du fichier .lvl (pas de caractères spéciaux notamment).

Dessin :

- Les dessins sont enregistrés dans des LinkedList<Vector>. Cette classe Vector ne représente pas des vecteurs classiques, mais nous y enregistrons les valeurs nécessaires pour utiliser les fonctions de dessin de Graphics (drawLine et drawArc).
- Les valeurs capturées, à mettre en paramètres des fonctions de dessin, sont aussi arrangées et/ou calculées, pour pouvoir proposer un résultat plus "logique" au vu du dessin (arc de cercle dans la continuation du pinceau, calcul de la destination des traits).
- Le dessin est simplifié quand il est submit dans **Create**, ou quand il est run dans un niveau. C'est-à-dire que les mêmes vecteurs et les superpositions sont enlevés du dessin, et deux vecteurs qui seraient le prolongement l'un de l'autre sont fusionnés. Cette simplification passe par une comparaison de chaque vecteur avec tous les autres déjà traités (que nous enregistrons dans une liste).
- Dans un niveau, à la fin de la simplification, les vecteurs du dessin du joueur sont comparés avec ceux du patron, pour vérifier si le dessin reproduit est celui attendu ou non.

Langage en bloc :

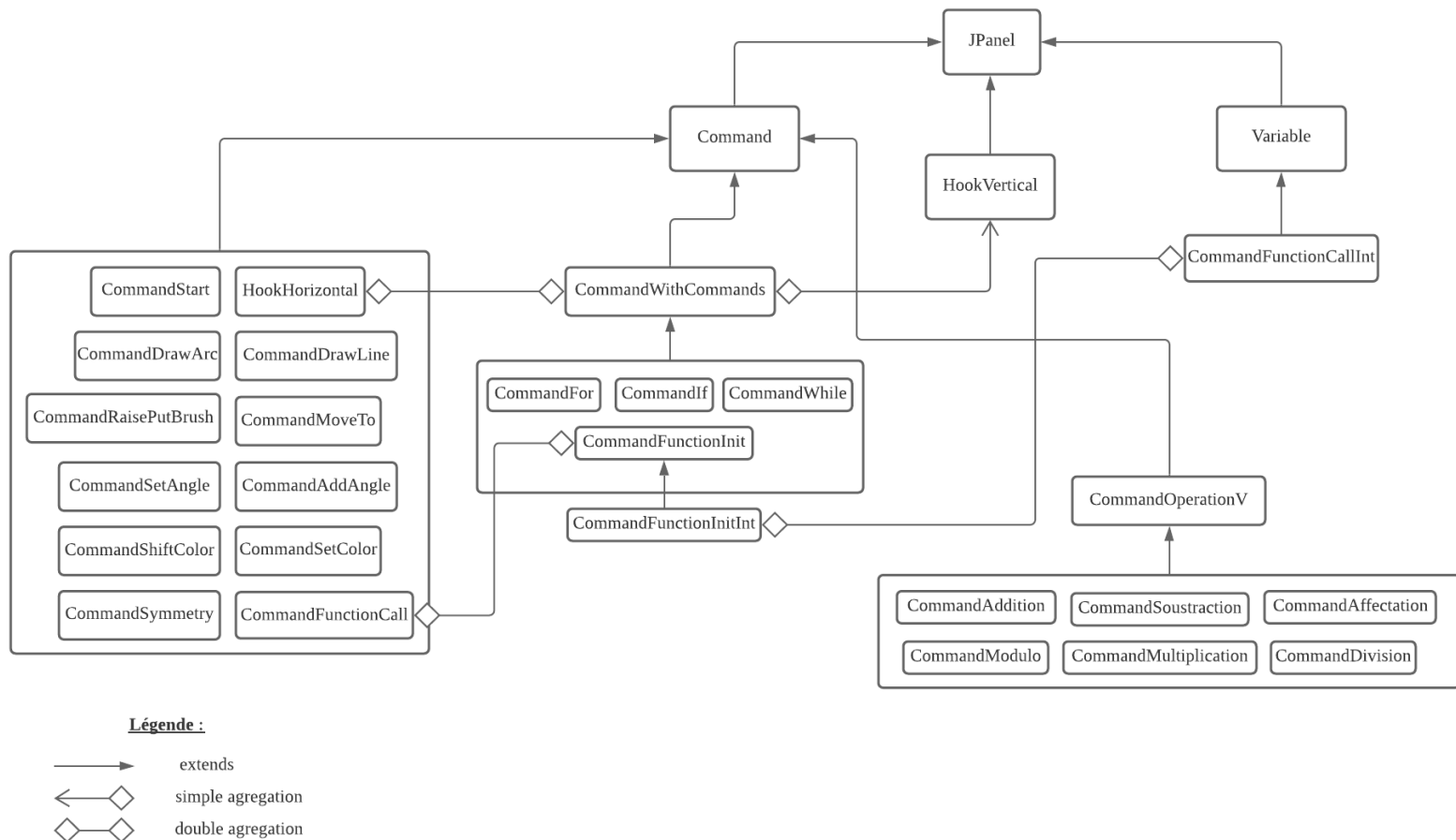
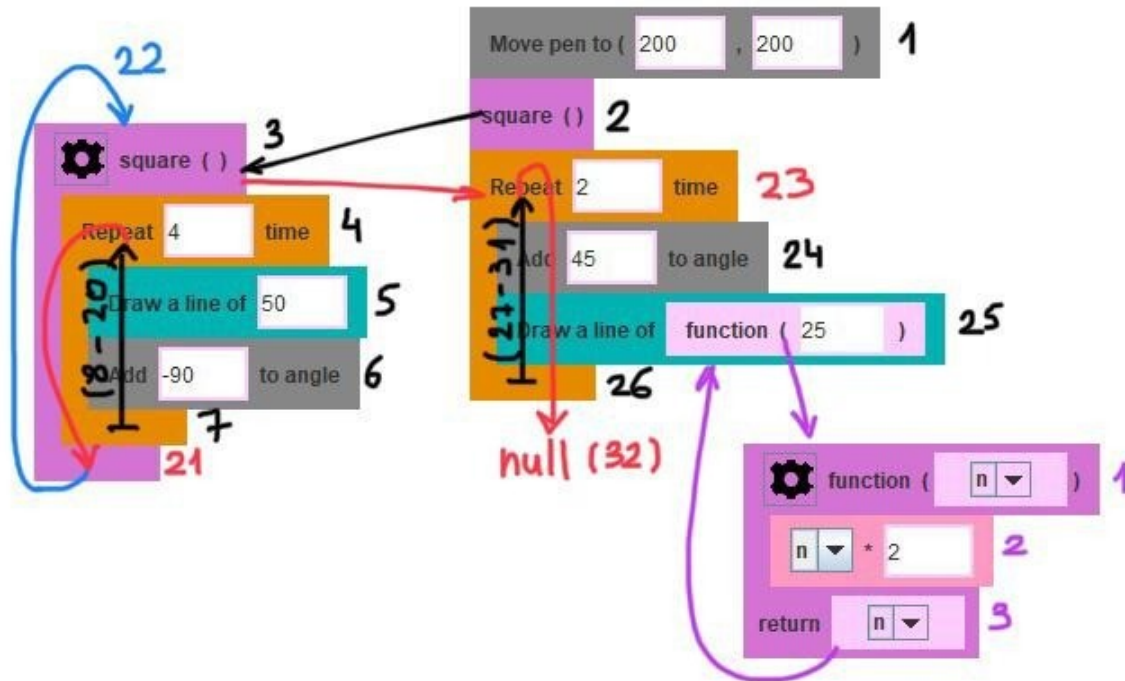


Diagramme détaillé des classes du langage

- Chaque bloc est un objet de la classe Command ou Variable (qui sont des JPanel). Il existe deux grands types de Command : les commandes “générales” (représentées par un unique JPanel) et les CommandWithCommands qui peuvent contenir des commandes à l’intérieur d’elles, comme les boucles par exemple (représentées par trois JPanel, dont deux Command qui ont des attributs pointant l’un vers l’autre). Les Variable ont une implémentation assez similaire avec les Command, à quelques détails près.
- Les commandes peuvent être attachées à d’autres commandes. Pour cela, nous sauvegardons préalablement dans une liste les liens vers toutes les commandes (sinon nous n’aurons pas accès à l’objet auquel nous voulions accrocher une commande) et nous créons manuellement une LinkedList<Command> qui relie les commandes entre elles (attributs Command previous et Command next). La référence vers la commande précédente ne sera utile que pour l’affichage graphique, et celle vers la suivante sera utilisée pour lancer le code (exécution de this, puis du suivant, ...).
- Chaque commande représente des points communs avec les autres : gestion des déplacements, accrochage/décrochage, suppression, régénération, mise à l’avant-plan, redimensionnement. Ces méthodes sont écrites dans la classe mère Command. Les différences sont gérées plus spécifiquement dans les classes enfants : vérifier si la commande est exécutable, l’exécution.
- Les variables sont enregistrées dans des HashMap, globale ou locales selon si elles sont dans une fonction qui retourne un entier, ou ailleurs. Les HashMap

locales serviront lors de la récurrence de ces fonctions avec retour.

- La récurrence est d'ailleurs interdite dans les fonctions sans retour, puisqu'elles sont utilisées pour dessiner, alors que les fonctions avec retour servent à calculer. Du fait de la récurrence possible dans les fonctions avec retour, nous devons faire une copie de la fonction avant de l'utiliser. Sinon, la gestion des for, if et while pose des problèmes, notamment dû à des reset() de CommandWithCommands qui doivent s'opérer correctement.



Étapes dans l'exécution du code (la command Move pen est accrochée à Start)

Erreurs et gestion :

- Quand nous démarrons le jeu, les répertoires sont initialisés s'ils n'existent pas. Dans ce cas-là, ils sont vides, sauf pour players/, qui contient par défaut les deux comptes GM et default. Cette initialisation permettrait un accès minimum au jeu, quitte à ne pas avoir de niveaux (si les données externes ne sont pas présentes dans le bon répertoire ou supprimées).
- Il y a des conditions à respecter pour se connecter ou créer un compte. Par exemple, il ne faut pas que le champ password soit vide, nous interdisons les caractères spéciaux, ... Nous affichons des messages d'erreur appropriés selon les cas.
- Avant l'exécution du code du joueur, nous vérifions préalablement les champs (champs vides, division par 0, CommandWithCommands vides, boucles infinies), et s'il y a des problèmes, nous montrons au joueur ce qui ne va pas avec un affichage graphique (commandes/variables/JTextField qui deviennent rouges).
- Pendant l'exécution, si un trait sort du BlackBoard (de dimension 400*400), ou qu'il est trop petit (distance/angle/rayon insuffisant), un popup apparaît à l'écran pour expliciter la nature de l'erreur (out of bounds, too small), et un affichage graphique montre aussi le champ problématique. Il est d'ailleurs impossible de toucher au code pendant une exécution, sinon le dessin change dynamiquement aussi, et nous aurions alors un dessin incorrect.
- Il y a des conditions strictes pour (re)nommer les variables/fonctions (seules les

lettres sont autorisées, et aussi les nombres pour les fonctions), pour se rapprocher le plus possible d'un vrai langage.

→ Nous avons essayé de rapprocher le plus possible la gestion des commandes de notre jeu et celle d'un véritable IDE, notamment dans la gestion des erreurs et des oublis, bien que le tout soit simplifié.

Aide dans le codage :

- Nous avons inséré des aides graphiques pour la compréhension des commandes et pour faciliter la prise en main du jeu : image du cercle trigonométrique, blocs qui s'allument ou non selon si il est possible de coller des blocs entre eux, catégorisation des commandes avec des couleurs...
- Les niveaux dans **Training** ont une difficulté croissante : nous abordons les concepts les plus importants de la programmation petit à petit.
- Dans certains niveaux, le code et/ou les fonctions sont enregistrés, pour éventuellement initier à certains principes du codage. Le joueur disposera alors d'une aide pour résoudre le niveau, et il n'aura plus qu'à y insérer les bonnes valeurs. Nous avons pu utiliser cette fonctionnalité pour montrer la factorisation du code, la création de fonction, l'utilisation de variables globale/locale, ...
- Un affichage dynamique des valeurs des variables est proposé, et le résultat du code du joueur est directement visible dans BlackBoard, afin de mieux rendre compte de ce qu'il se passe à l'exécution du code, cela peut permettre également la détection des erreurs ou des dysfonctionnements.
- Lorsque le joueur souhaite créer/supprimer une variable ou créer/renommer une fonction, cela lui est rendu possible grâce à des boutons. La suppression/renomination est semblable au raccourci Ctrl-R des IDE.

Graphisme :

- Pour rendre le jeu plus agréable visuellement, nous avons introduit deux graphismes : un mode nuit avec des couleurs sombres, et un mode jour avec des couleurs pastels.
- Nous avons aussi pris en compte les différentes tailles possibles des écrans, et rendu la présentation relative. Trois tailles de BlackBoard sont aussi proposées, pour voir le dessin du joueur en plus grand. Mais si l'écran du joueur est trop petit, certaines fonctionnalités lui seront ôtées (la taille médium, ou bien l'affichage dynamique des variables sous certaines tailles). De plus, plusieurs JPanel sont rendues scrollables pour éviter ce genre de problème.
- Dans le jeu, le redimensionnement des commandes se fait dynamiquement selon les cas (accroche ou création de variables, accrochage dans les CommandWithCommands, ...). La commande active (et celles qui lui sont collées) est mise au premier plan. Les variables, s'il y en a, sont toujours au premier plan, au-dessus des JTextField.
- Concernant l'animation du code, à chaque fois qu'un trait est tracé dans le dessin, il y a un petit temps d'arrêt (réglable par le joueur) avant que le prochain trait soit à son tour tracé.

Répartition du travail :

- Nous avons tout d'abord fait plusieurs réunions à quatre pour réfléchir à la structure globale du jeu, que ce soit les codes ou nos idées pour implémenter les premières choses qui nous venaient à l'esprit avant de s'attaquer à la programmation plus

concrètement. Dans la suite, il y a eu de nombreux échanges d'idées, bien que chacun n'apporte pas sa contribution directe au code correspondant : ce type d'échange ne se reflète bien entendu pas dans les commits.

- Pierre : gestion des données des joueurs (création de compte et connexion, sauvegarde de la progression) et des niveaux (création, affichage dans les sommaires, chargement), élaboration de commandes et des training
- Anna : principalement la gestion des commandes et variables (fonctionnement et affichage), les vecteurs et la création de training/challenge, ainsi que participation à l'amélioration de certaines fonctionnalités par-ci et par-là
- Line : principalement l'amélioration de l'aspect graphique, la gestion des erreurs en rapport à la création de comptes, l'ajout/l'amélioration de quelques fonctionnalités, et participation à l'élaboration de commandes
- Steven : gestion des erreurs liées aux vecteurs, élaboration de commandes, amélioration de l'expérience du joueur (notamment facilitation de la navigation et des entrées)

Recherches doc :

- Dans le cadre de la création de compte pour les joueurs nous avons eu besoin d'un système pour crypter les mots de passe et ainsi garantir la sécurité. Nous avons donc opté pour l'option AES :
<https://howtodoinjava.com/java/java-security/java-aes-encryption-example/>
- Lors de la création des sommaires, afin d'avoir un résultat plus esthétique et plus parlant pour les joueurs, nous avons décidé d'ajouter une prévisualisation des niveaux (c'est-à-dire une image) dans les boutons permettant d'accéder aux niveaux. Ainsi, nous avons cherché un moyen de prendre un screenshot de l'écran (plus précisément du BlackBoard) et nous avons trouvé un morceau de code satisfaisant en faisant des recherches sur internet :
<https://www.codota.com/code/java/methods/java.awt.Robot/createScreenCapture>
- Puisque nous n'avons pas participé à l'UE préprofessionnalisation au semestre dernier, nous n'avons pas les bases pour utiliser git. Pour palier à cela et avoir au moins les bases pour pouvoir commit, push, pull nous avons suivi des tutoriels, comme celui-ci :
<https://www.vogella.com/tutorials/EclipseGit/article.html>
- Pour une meilleure navigation dans le jeu, nous avons implémenté la possibilité pour le joueur d'utiliser la touche Entrée pour remplacer certains clics, avec l'interface KeyListener. Elle a aussi permis la gestion des erreurs de saisie lors de la connexion/l'inscription du joueur. Le tutoriel qui a servi de référence :
<https://waytolearnx.com/2020/05/keylistener-java.html>
- Pour se renseigner sur les différents composants de Swing nous avons globalement utilisé la documentation oracle sur swing ainsi que différents tutoriels ou forum selon les composants :
<https://docs.oracle.com/javase/7/docs/api/javafx/swing/package-summary.html>
<https://stackoverflow.com/questions/11093326/restricting-jtextfield-input-to-integers/11093360> (restriction de la saisie dans les JTextField)
<https://tips4java.wordpress.com/2008/11/06/wrap-layout/> (WrapLayout, une sorte de FlowLayout scrollable)
<http://www.java2s.com/Code/Java/Swing-JFC/Colorcomboboxrenderer.htm> (colorBox)
- Nous avons regardé la documentation oracle pour comprendre comment fonctionnent les HashMap :
<https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>

- Pour comprendre comment dessiner avec Graphics, et notamment pour réadapter drawArc(), nous avons étudié en détail le fonctionnement des fonctions en question : https://www3.ntu.edu.sg/home/ehchua/programming/java/J4b_CustomGraphics.html
- Nous avons cherché dans différents forums le moyen de mettre de la musique dans le jeu.

Problèmes :

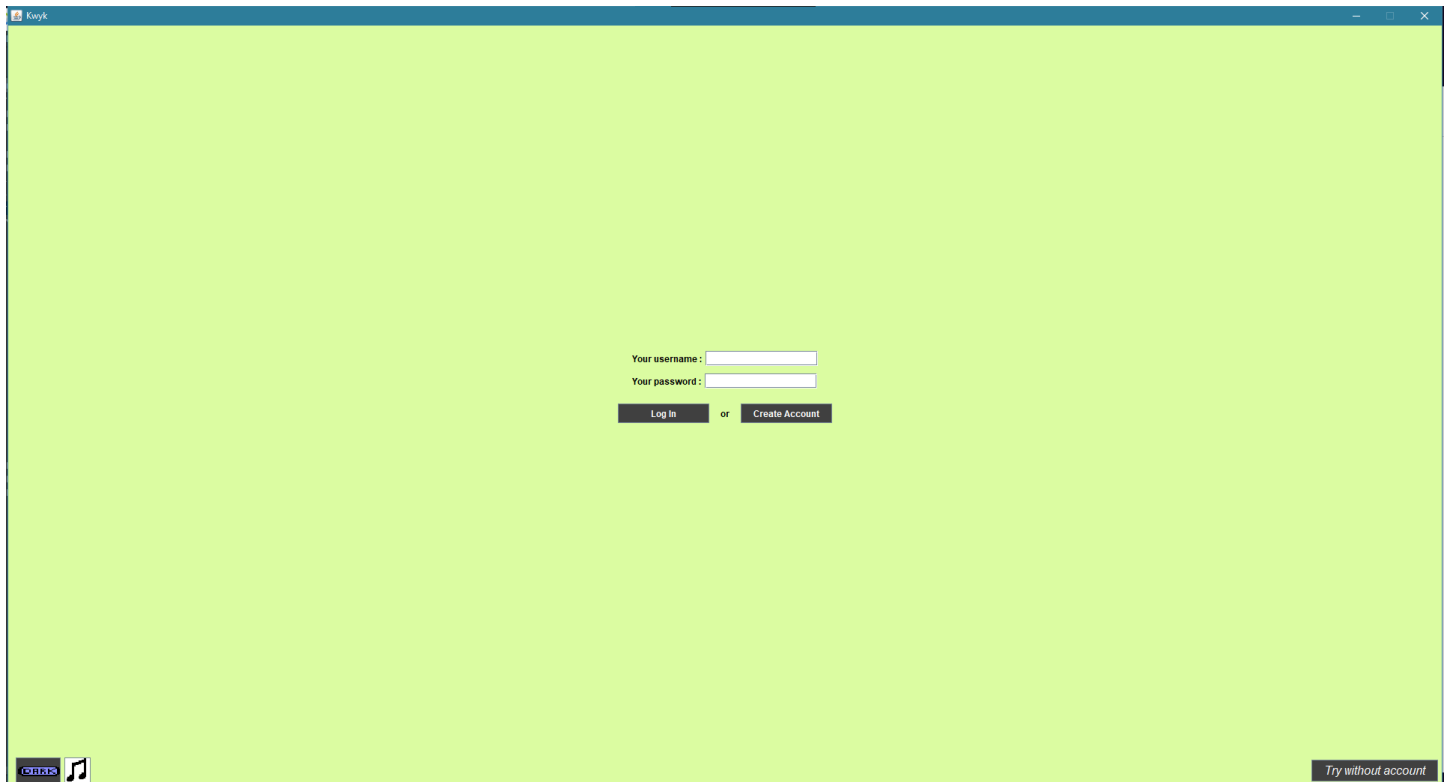
- N'ayant pas eu l'UE préprofessionnalisation au semestre dernier, nous n'avons pas été formés à l'utilisation de git, ce qui a conduit à plusieurs problèmes (jusqu'à la fin du projet) comme la gestion hasardeuse des conflits entre commits.
- Concernant le jeu, l'aspect graphique a été le plus difficile à gérer, probablement en rapport aux différentes versions de java : la méthode getPreferredSize() (pour tous les composants utilisés) ne semble pas donner la bonne valeur, ce qui crée des problèmes d'affichage (sur la version la plus récente de java).
- Lorsqu'il a fallu finaliser le projet en créant les niveaux, de nombreux bugs ont été découverts, comme la sauvegarde de code (option **Save code** dans le mode **Create**) qui était incomplète lors d'un appel de fonction dans une fonction ou de l'utilisation de variables.

Extensions :

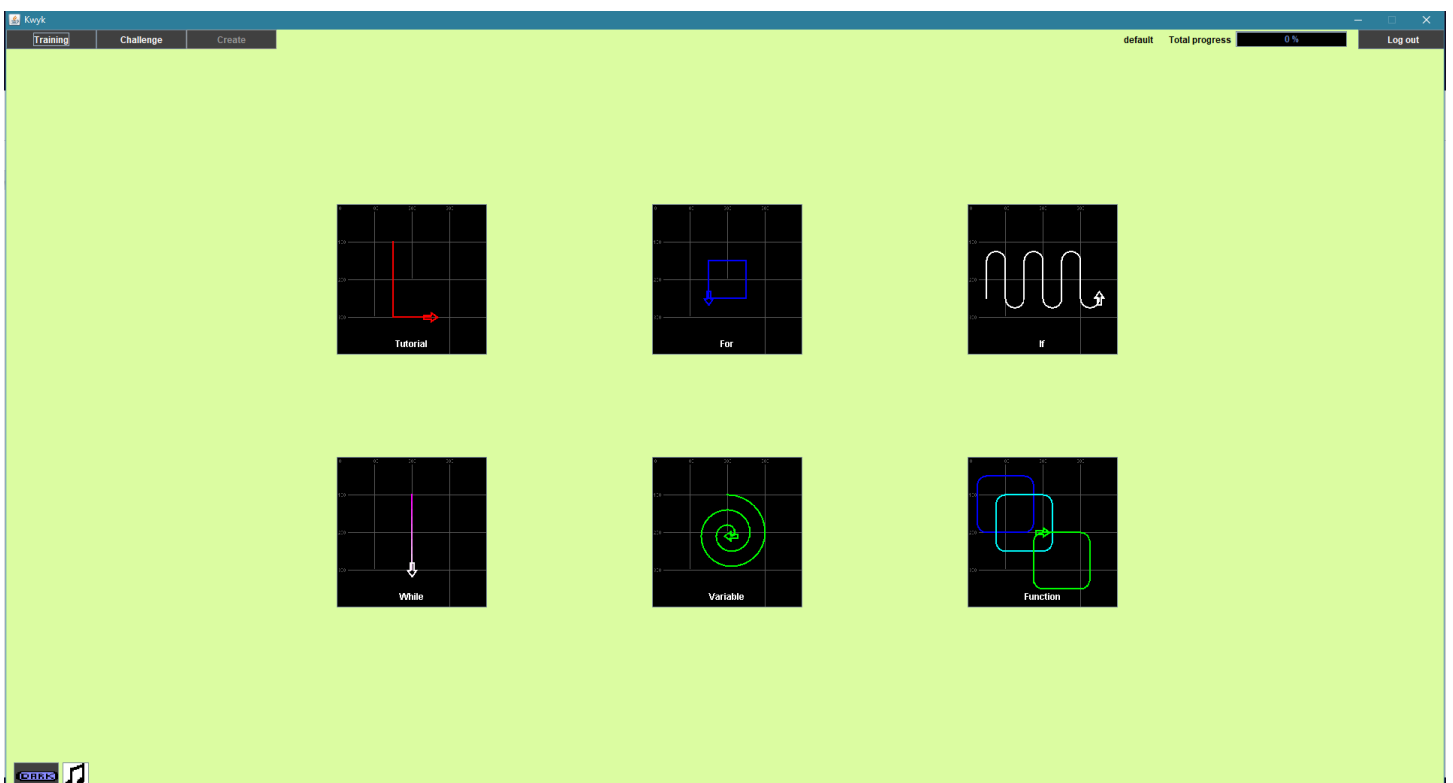
Néanmoins, notre jeu est sujet à diverses améliorations et/ou changements tels que :

- Ajout d'une fonction copier-coller afin de rendre beaucoup plus pratique la création et la résolution des niveaux
- Introduire une sensibilité aux clics pour détacher une commande (problème que nous rencontrons lorsque nous faisons un missclick sur un bloc, qui détache une commande sans que nous puissions nous en apercevoir)
- Possibilité d'ajouter du multijoueur dans le jeu. Par exemple, la page **Challenge** fonctionnerait à la manière d'un workshop où nous pouvons télécharger les niveaux créés par d'autres joueurs, qui sont sur une autre machine. Ou encore la possibilité de résoudre certains niveaux en collaboration, avec un fonctionnement similaire à celui de Google Docs.
- Ajouter des pinceaux à textures (épaisseur des traits, crayon/pinceau/stylo) afin de développer plus en profondeur l'aspect dessin et visuel du jeu
- Exporter ses dessins = pouvoir récupérer les images
- Création d'un panel d'administrateur pour les GM qui permettraient de supprimer des niveaux/joueurs plus facilement sans avoir à toucher aux fichiers du jeu
- Tuto vidéo et personnalisation (avec des popup) de chaque niveau d'entraînement (explications des commandes/principes de programmation)
- Augmentation du nombre d'arguments permis dans les fonctions avec retour, afin de créer des fonctions plus intéressantes (calcul de l'hypoténuse ou de la distance par exemple)
- Ajouter des attributs "nombre de fois ouverte" et "nombre de fois réussie" pour chaque challenge, et faire un classement du dessin le plus populaire (nombre de fois ouverte) et du plus difficile (nombre de fois réussie/nombre de fois ouverte)
- Rendre plus fluide le changement des pages
- Rendre l'exécution du code plus dynamique (la commande en cours qui s'allume)

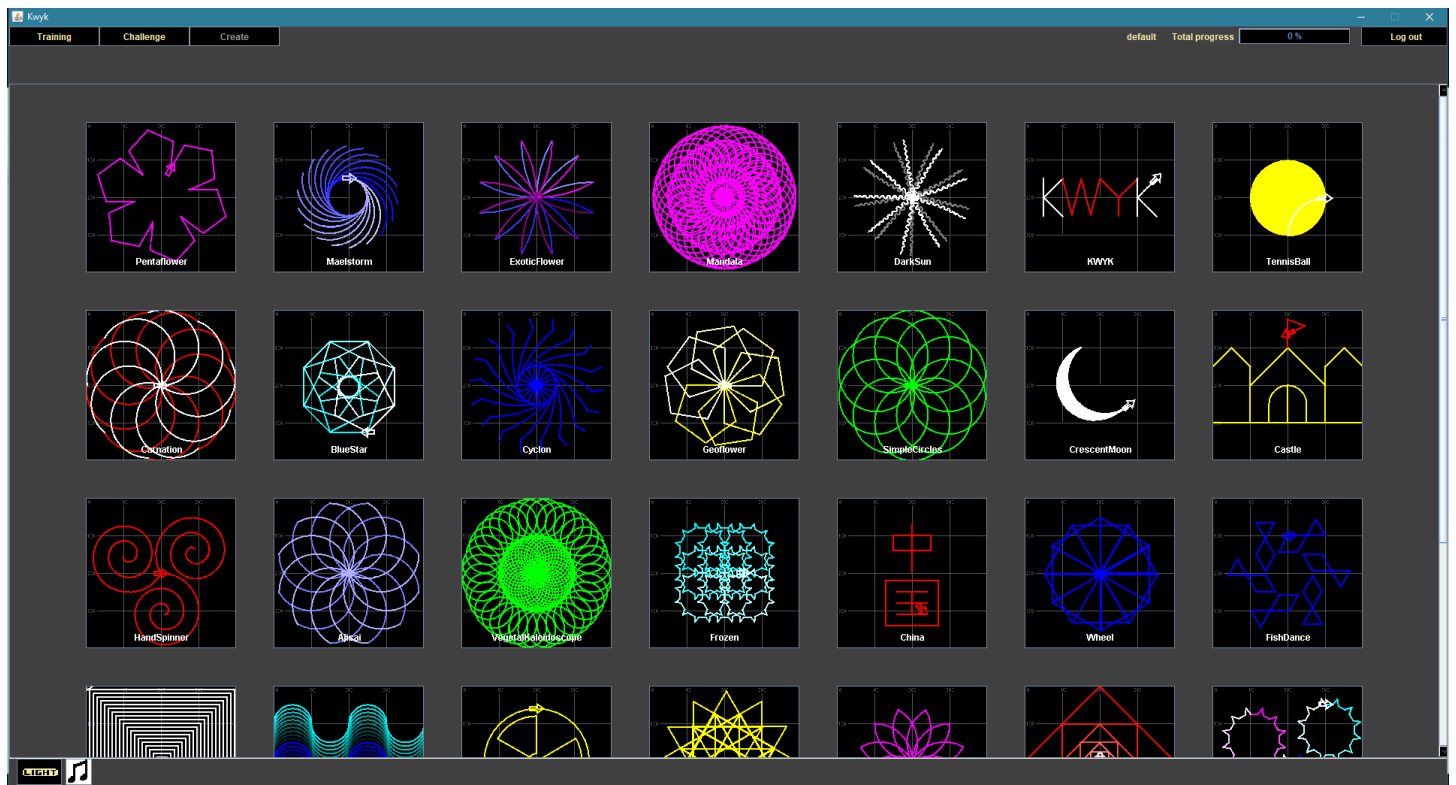
Différentes images utiles pour présenter le jeu :



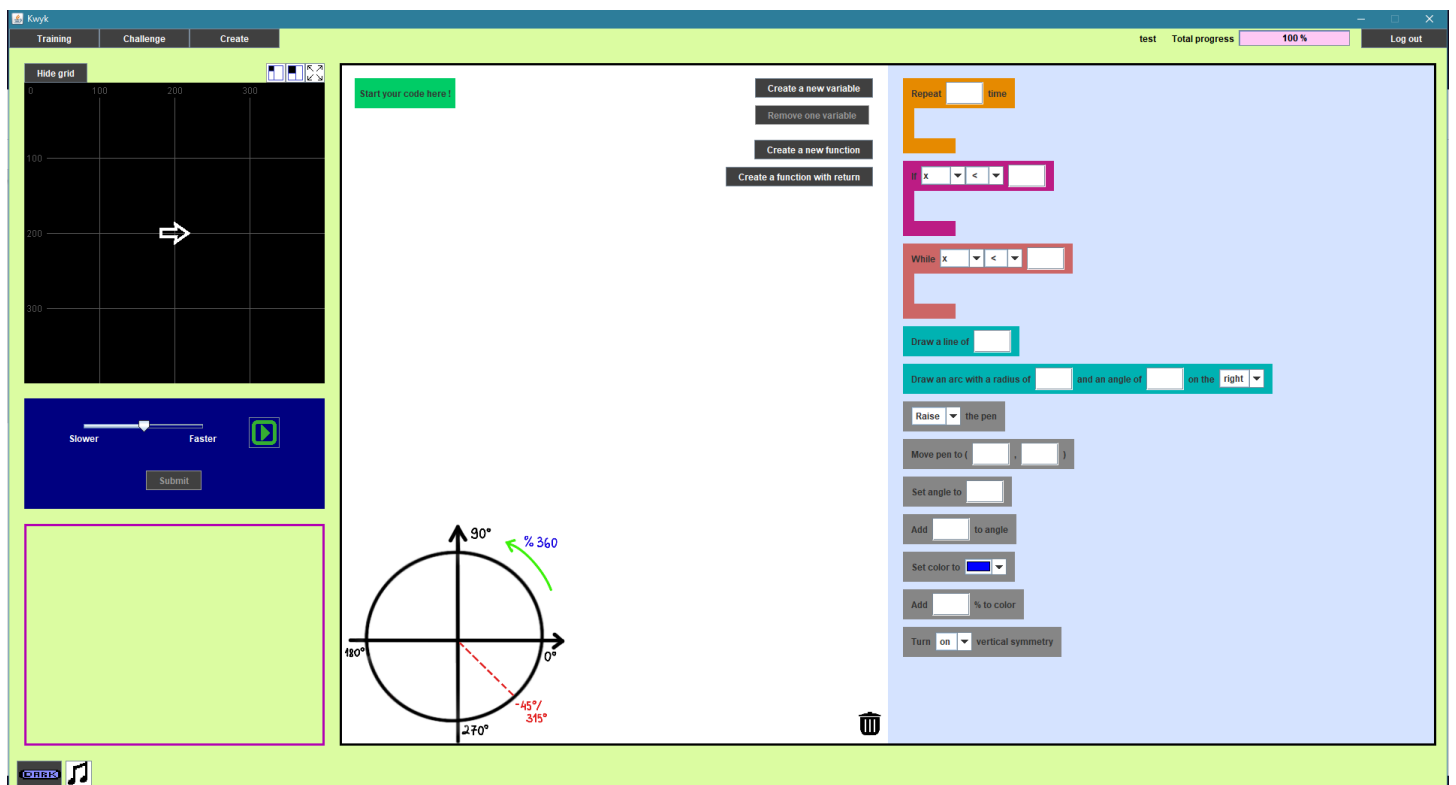
Kwyk Login (light theme)



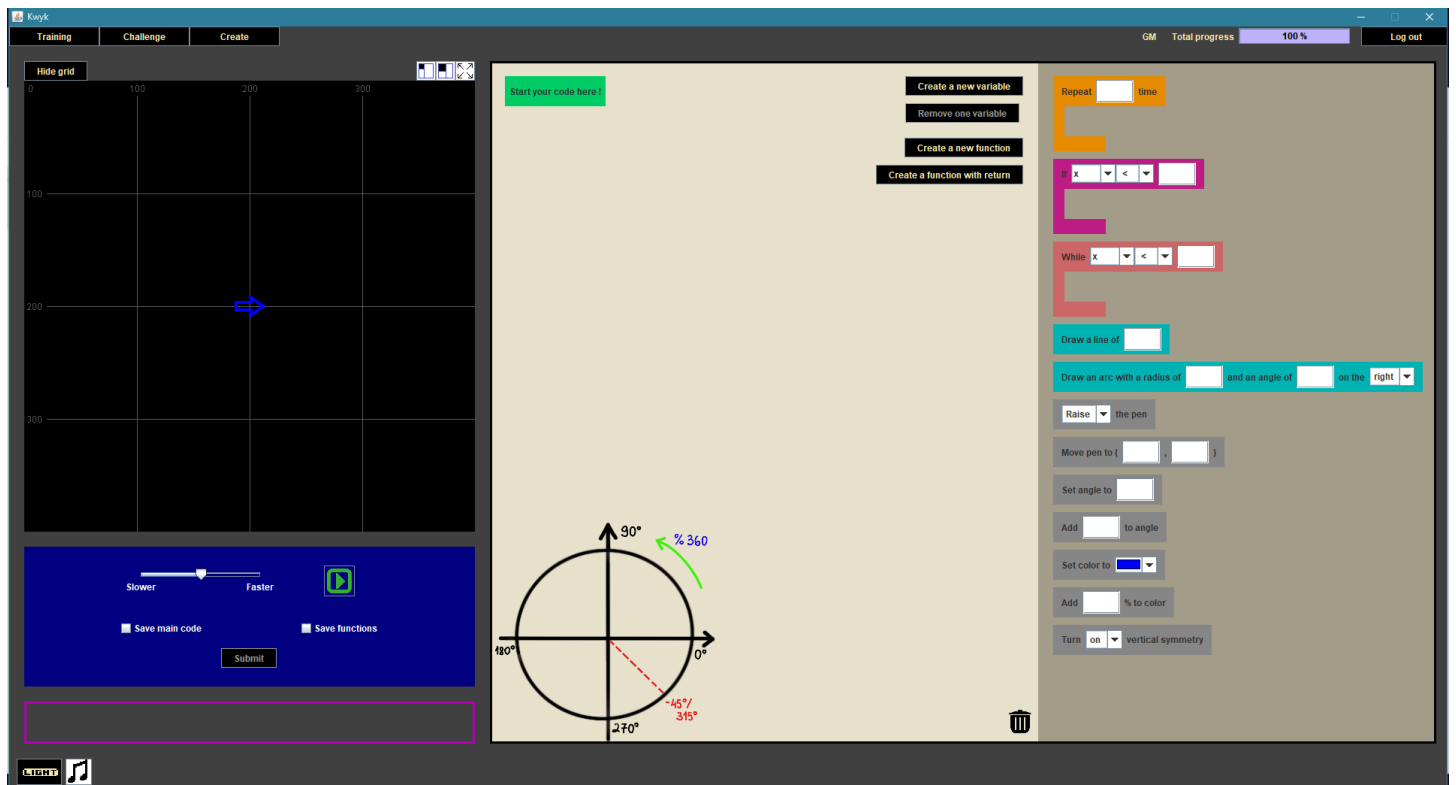
Kwyk Summary Training (page principale des catégories, light theme)



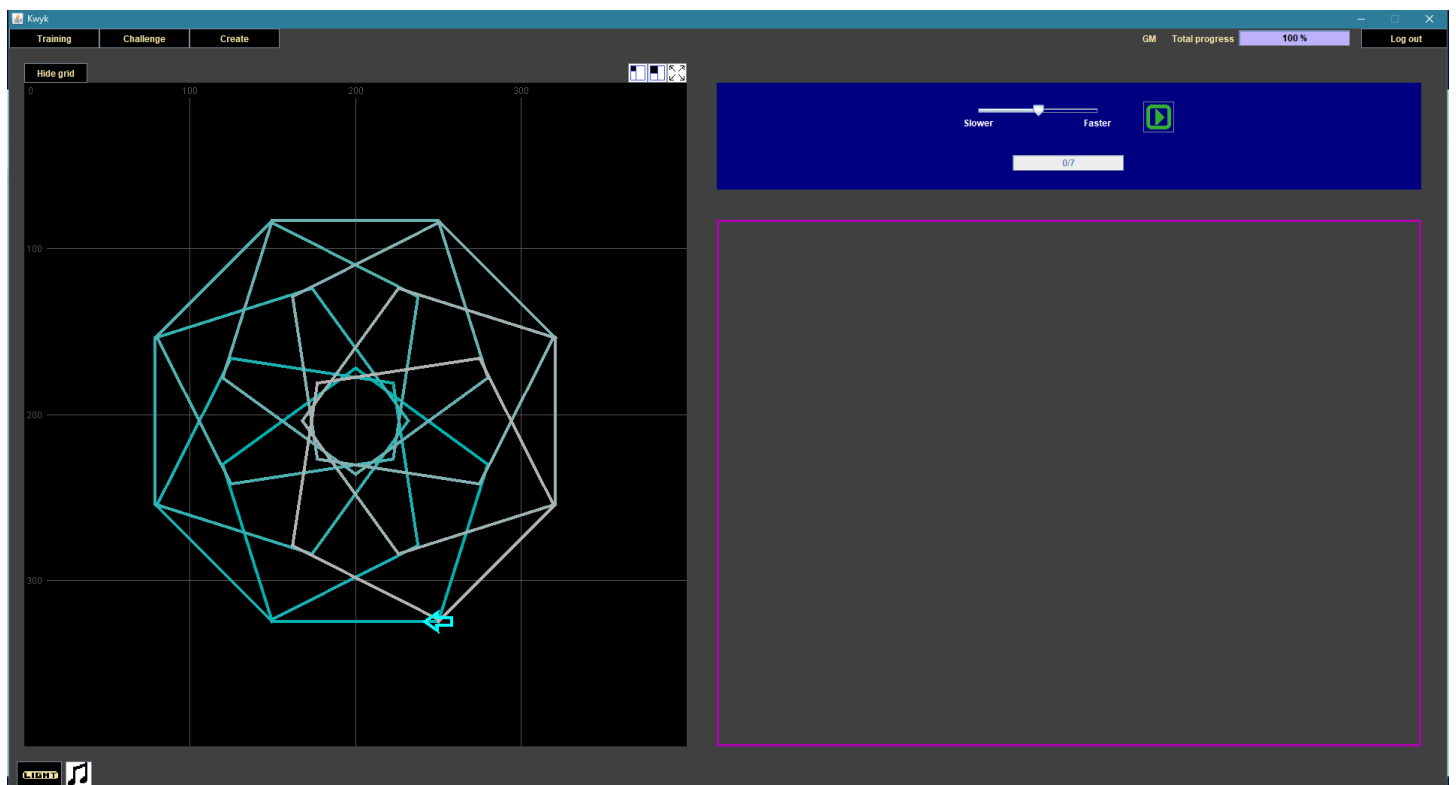
Kwyk Summary Challenge (dark theme)



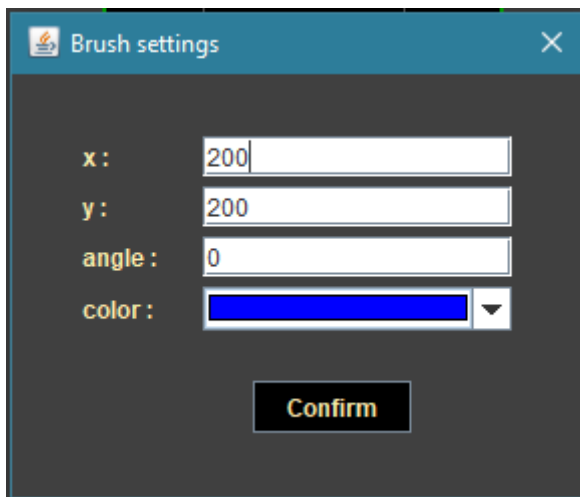
Kwyk Create chez un player lambda (taille classique et light theme)



Kwyk Create chez le GM (taille M et dark theme)



Kwyk Playing (plein écran, dark theme)



Brush settings

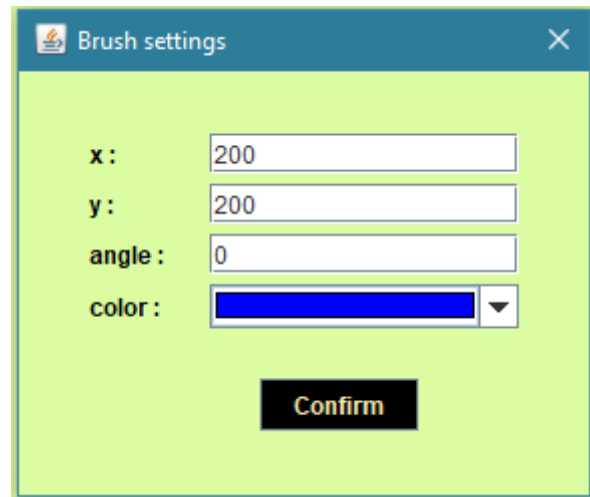
x : 200

y : 200

angle : 0

color : ▼

Confirm



Brush settings

x : 200

y : 200

angle : 0

color : ▼

Confirm

Fenêtre pour choisir les attributs du pinceau (GM seulement)