# Contrôle Continu Patron Conception

Sélian Arsène, Guillaume Lericheux, Brice Andrieux

April 15, 2025

## Contents



# 1 Projet Pixel Perfect

Le projet évalué de l'unité d'enseignement Patrons de conception est un jeu de dessin de formes à reproduire. Le concept est trivial : l'utilisateur peut choisir de reproduire des formes générées aléatoirement sur une feuille de dessin ou bien jouer contre un autre joueur, où le premier joueur doit reproduire les formes que le deuxième joueur aura dessiné, puis inversement. Un score à la fin de la partie est attribué à un joueur en fonction des formes qu'il a dessiné.

# 2 Utilisation du programme

Notre programme peut se lancer avec la commande "ant run". Une fois celle-ci effectuée, vous pouvez choisir de jouer entre une partie en solo ou à 2 joueurs.

## 2.1 Partie en solo

Sur une partie en solo, 10 formes seront générées aléatoirement et ne sont affichées que pendant 10 secondes.
Une fois ce temps écoulé, vous pouvez créer de nouvelles formes avec le premier bouton situé en haut de l'écran, qui vous amène dans un menu de sélection de la forme que vous voulez placer. Vous pouvez ensuite créer la forme à l'aide d'un glisser-déposer.
Avec les 2 boutons suivants, vous pouvez choisir de déplacer ou supprimer des formes existantes. Une fois l'une de ces 2 actions sélectionnées, vous pouvez respectivement glissez-déposez une forme pour la déplacer ou cliquez une forme pour la supprimer. Vous pouvez aussi déplacer une forme en maintenant enfoncé le clic droit de votre souris.
Avec les 2 derniers boutons, vous pouvez annuler des actions, ou rétablir des actions annulées.
Une fois que vous êtes satisfait de votre disposition de formes, vous pouvez valider votre disposition avec le bouton en bas de l'écran. Un score vous sera donné. Ce score varie de 0 à 100, et plus il est élevé, mieux vous avez reproduit les formes qui vous ont été montrées au lancement de la partie.
Vous pourrez ensuite retourner au menu de lancement de parties.

## 2.2 Partie à 2 joueurs

Dans une partie à 2 joueurs, chaque joueur va pouvoir, à tour de rôle, librement créer des formes. Il pourra ensuite passer la partie à l'autre joueur, qui devra mémoriser cet ensemble de forme, et la recréer, comme dans une partie solo.
Ces actions se répèteront 10 fois, où chaque joueur devra créer librement des formes 5 fois, et reporduire les formes de l'autre joueur 5 fois. Les scores de chaque partie sont sommés au fur et à mesure, et celui qui marque le plus de points durant la partie est déclaré vainqueur.

# 3 Design Patterns implémentés

## 3.1 Modèle Vue-Contrôleur

Notre projet suit principalement l'architecture Modèle Vue-Contrôleur.

### 3.2 Singleton

Afin d'éviter de recréer plusieurs fois une instance qui n'a besoin d'être créée qu'une seule fois, on a implémenté le pattern Singleton. La classe Dessin, dans laquelle nous enregistrons nos formes, est un singleton car nous effaçons à chaque fin de tour de jeu les formes dessinées. Le gestionnaire de commandes CommandHandler, la classe de détection de formes DetecterForme ou bien GenererFormesAleatoires sont aussi des singleton.

### 3.3 State

Il s'agit d'un design pattern essentiel au bon fonctionnement du jeu, le pattern State nous permet d'interagir avec les différents éléments graphiques Swing et la souris.
Pour ce faire , le pattern utilise la classe Interaction située dans src/control/state , cette classe est la clé de voûte du pattern .
En effet, la classe crée une multitude d'instances de classes situées elles aussi dans src/control/state et les enregistre en public final .
On peut retrouver les classes secondaires suivantes :

- Déplacer : Pour déplacer une forme.
- Effacer : Pour effacer une forme.
- Dessiner : qui ouvre un menu avec une multitudes de formes à créer, listées ci dessous.
- DessinerRectangle : Pour dessiner un rectangle.
- DessinerCercle : Pour dessiner un cercle.
- DessinerTriangle : Pour dessiner un triangle.
- DessinerAnyPolygone : Pour dessiner un polygone à plus de 4 cotés.

Dans la classe Interaction on y retrouve une implémentation de MouseMotionListener et de MouseListener.
Malheureusement il n'est pas possible hériter MouseAdapter, car notre classe hérite déjà AbsModeleEcoutable, afin de prévenir la vue et le modèle des futurs changements.
Toutes ces fonctions sont donc déléguées aux autres classes mentionnées précédemment , selon l'état actuel de l'interaction .
Le pattern state possède comme état initial le déplacement, et après avoir exécuté une action comme la création d'une forme ou bien la suppression d'une forme, l'état retourne à son état initial.
Dans l'interaction nos classes secondaires sont du type "EtatInteraction" qui elles n'hérite pas le MouseAdapter (ce qui n'est pas le cas de AbsEtatInteraction qui lui implémente EtatInteraction et hérite ce MouseAdapter).
Cette situation nous force donc à convertir toutes nos classes secondaires via

leurs types, on ne peut pas convertir ou créer nos classes sur AbsEtatInteraction (qui lui contient le MouseAdapter et l'EtatInteraction) car par définition on n'initialise pas une classe Abstraite.
Lors de l'ajout de formes via les classes secondaires : DessinerRectangle , DessinerCercle etc ... nos classes font appel aux différentes méthodes de la classe Dessin dans le modèle (qui s'occupe du stockage et de la gestion des formes) et qui par la même occasion applique le Pattern Command.

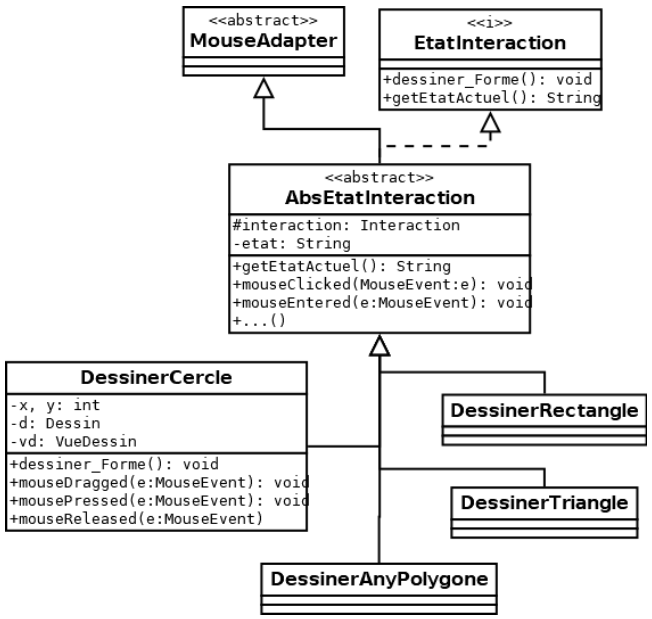


Figure 1: Diagramme UML de l'interaction

## 3.4 Command

Le pattern Command permet d'appliquer les fonctionnalités suivantes : Undo et Redo ce qui correspond à Ctrl-Z et Ctrl-Y.
La clé de voûte de ce pattern est la classe "CommandHandler" située dans src/control/command. Ici le principe est assez simple , Dessin va appeler le CommandHandler et va créer deux piles qui devront contenir des OperationCommand.
Une OperationCommand est une classe qui implémente deux états :

- void Operate() : Permet d'appliquer l'action.
- void Compensate() : Permet de faire machine arrière.

Par exemple: la classe CommandMoveForme qui doit donc appliquer le déplacement d'une forme (ainsi que son opposé), va devoir prendre une forme, et changer sa

position via des Setters.
De plus, dans le cas ou notre forme est un polygone, les Setters s'occuperont aussi de changer la position de touts les points de la forme afin de rendre son apparition sur la vue authentique.

Pour appliquer un Undo et un Redo il suffit juste d'appuyer sur les boutons $\leftarrow$ et $\rightarrow$ situé sur la fenêtre de dessin.
La fenêtre principale (celle ou l'on peut dessiner), possède donc elle aussi un accès au CommandHandler et elle peut interagir dessus grâce à l'instance du pattern singleton.

### 3.5 Chain of Responsability

Ce pattern permet de générer une partie de jeu joueur contre ordinateur et joueur contre joueur. Les clés de voûtes de ce pattern sont : Jeu et MaillonJeu situés dans src/control/maillon.
Tout commence sur le Menu qui hérite le MaillonJeu, ce menu possède deux boutons : le premier joueur contre ordi génère la chaîne de maillons suivants :
Initialisation d'un nouveau jeu , le jeu permet de démarrer la chaîne de maillon via des threads et la chaîne s'arrêtera uniquement quand la boucle while true situé dans la méthode run() s'arrêtera.

- Maillon FenetreMemorisation : le joueur doit mémoriser le maximum de formes avant la fin du temps imparti. Il est possible d'appuyer sur un bouton pour passer le temps d'attente.
- Maillon FenetrePrincipale : le joueur reproduit les formes qu'il a vu précédemment.
- Maillon Score : ce maillon affiche le score et efface les formes situées dans Dessin afin de préparer la prochaine partie.
- Maillon Menu : ce maillon affiche au début le menu du jeu et arrête le thread lancé par jeu, cela permet de ne pas relancer la chaîne de maillon une nouvelle fois quand une partie est terminée.

Deuxième façon de jouer, le joueur contre joueur. Le joueur peut choisir de jouer contre une autre personne. le premier tour d'un jeu multijoueur se déroule de la façon suivante:
Initialisation d'un nouveau jeu , le jeu permet de démarrer la chaîne de maillon via des threads et la chaîne s'arrêtera uniquement quand la boucle while true situé dans la méthode run() s'arrêtera.

1. Maillon FenetrePrincipale : JoueurA dessine des formes sur une fenêtre de dessin que le JoueurB devra reproduire.

2. Maillon FenetreMemorisation : JoueurB doit mémoriser les formes dessinées par JoueurA avant la fin du temps imparti. Il est possible d'appuyer sur un bouton pour passer le temps d'attente.

3. Maillon FenetrePrincipale : JoueurB doit reproduire les formes qu'il a mémorisé sur une fenêtre de dessin.

4. Maillon Score : une fenêtre affiche le score accumulé par chacun des joueurs au fil des tours. Une fois atteint un certains nombre de tours, la partie se termine et un bouton permet de revenir au menu.

5. Maillon Menu : ce maillon affiche au début le menu du jeu et arrête le thread lancé par jeu, cela permet de ne pas relancer la chaîne de maillon une nouvelle fois quand une partie est terminée.

On répète ces 4 étapes en alternant JoueurA et JoueurB jusqu'à un certain nombre de tours. Une partie de jeu multijoueur se fait en un certain nombre de tours (par défaut 10). Une fois la partie terminée, les scores finaux de chacun des joueurs sont comparés afin de désigner un gagnant. Un joueur gagne s'il a obtenu un score plus élevé que celui de son adversaire.

# 4 Création des formes

Pour pouvoir créer différentes formes en restant le plus générique que possible, les formes sont séparées en 2 catégories : celles qui ont une forme ovale, et peuvent être dessiné avec la fonction drawOval de graphics, et celles qui ont une forme polygonale et pouvant être dessiné avec la fonction drawPolygon.

## 4.1 forme Ovale

Il n'y a qu'une forme ovale d'implémentée, le cercle. Le drawOval de graphics a besoin d'un point de départ (x et y), d'une largeur et d'une hauteur, qui sont le diamètre du cercle (donc 2 fois sont rayon)
Pour sa création manuelle, le joueur sélectionne en premier le centre du cercle, et le rayon dépend de la distance parcouru avec le glisser-déposer.
A noter que le rayon n'est pas la distance entre le début (point P1) et la fin (P2) du trajet de la souris, mais le maximum entre la distance en abscisse ou en ordonnée.

## 4.2 Forme Polygonale

Plusieurs formes polygonales sont implémentées : Les triangles, les rectangles, et les polygones régulier de 5 à 10 côtés (techniquement des polygones avec plus de côtés sont faisable, mais ils sont peu lisible à petite échelle donc nous avons préféré limiter à 10 côtés)

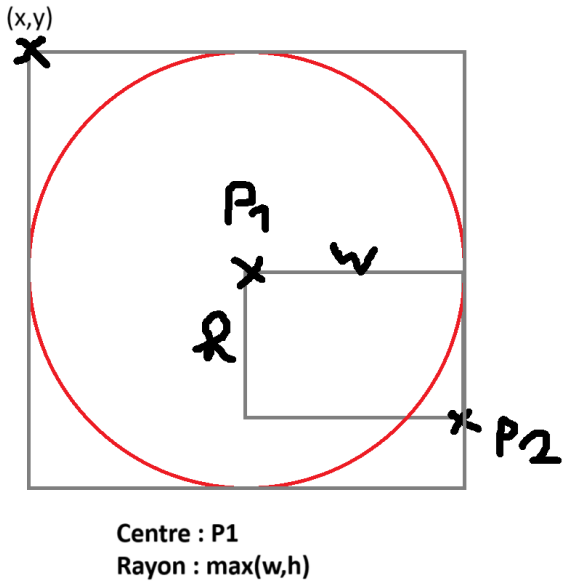


Figure 2: Création d'un cercle

Le drawPolygon de graphics a besoin d'une liste de coordonnées en x, d'une autre en y, et du nombre de point de la forme. Il faut donc donner la liste de tout les points de notre forme.
Pour créer un triangle ou un rectangle manuellement, le joueur va devoir glisser-déposer, ce qui créera un point de départ (P1) et d'arrivée (P2). On délimite ensuite un rectangle qui contiendra la forme, et on génère les points dedans de façon a maximiser l'espace pris par la forme dans la délimitation. Par exemple le rectangle prend exactement la même forme que la délimitation.
Pour les polygones régulier, le principe est le même excepté que la délimitation est un carré, ou similairement à la création de cercle c'est la plus grande distance entre abscisse ou entre ordonnée qui prime.

A noter que l'ordre des points dans les listes de coordonnées est toujours dans le même ordre : le premier point est toujours le point le plus en haut, à gauche. Les autres points sont placé dans le sens des aiguilles d'une montre en partant du premier.

## 5 Score d'une partie

Pour savoir le score d'un joueur sur une partie, on compare la liste de formes à reproduire, et celle qui ont été créé par le joueur.

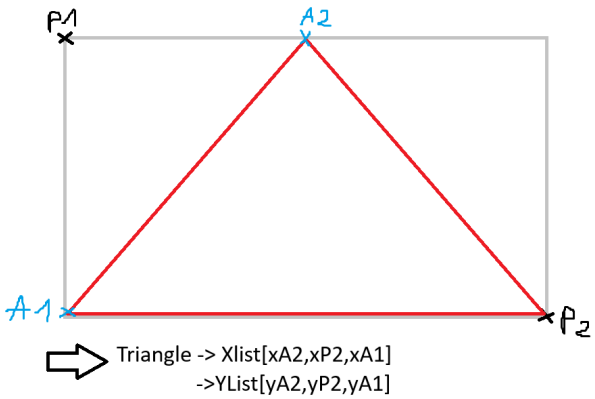


Figure 3: Création d'un triangle

Pour chaque forme à reproduire, on regarde comment ont été placées les formes de même type (entre les formes ovale ou polygone), et pour les polygones si elle ont le même nombre de côté. Pour chaque forme compatible, on commence à 100 points, que l'on diminue selon les écarts à la forme recherchée :

- Pour une forme Ovale, l'écart entre les centres des 2 formes fait perdre des points selon la distance ; puis l'écart de rayon entre les 2 formes fait de nouveau perdre des points.
- Pour une forme Polygone, on compare point par point la figure ; chaque écart entre les point correspondant d'une figure à l'autre fait perdre des points ; étant donné que les formes polygones commencent toujours par leur point le plus en haut à gauche, et continuent dans la même direction, on a juste besoin de comparer les points des 2 figures dans l ordre.

Si plusieurs formes compatibles ont un score positif, on garde la meilleure d'entre elles et on comptabilise son score. cette figure qui a marqué des points ne pourra plus en marquer plus.
Enfin, on fais la moyenne des scores, en divisant par la plus grande quantité parmi les formes à reproduire et celles placées par le joueur. Cela permet de limiter le score entre 0 et 100 quoi qu'il en soit, et de pénaliser un joueur qui a mis trop ou trop peu de forme.

# 6 Annexes


Figure 4: Choix du mode de jeu

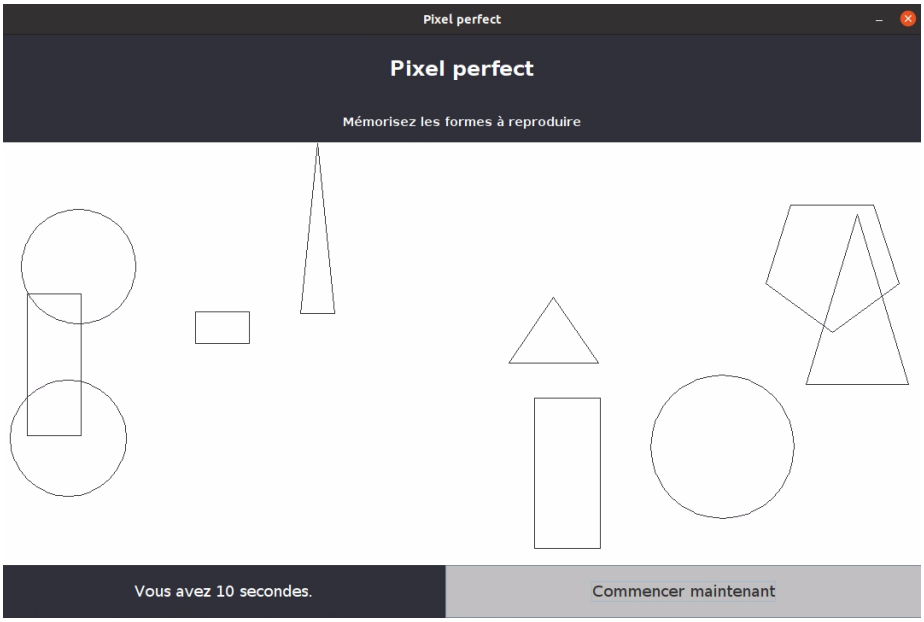


Figure 5: Formes à mémoriser

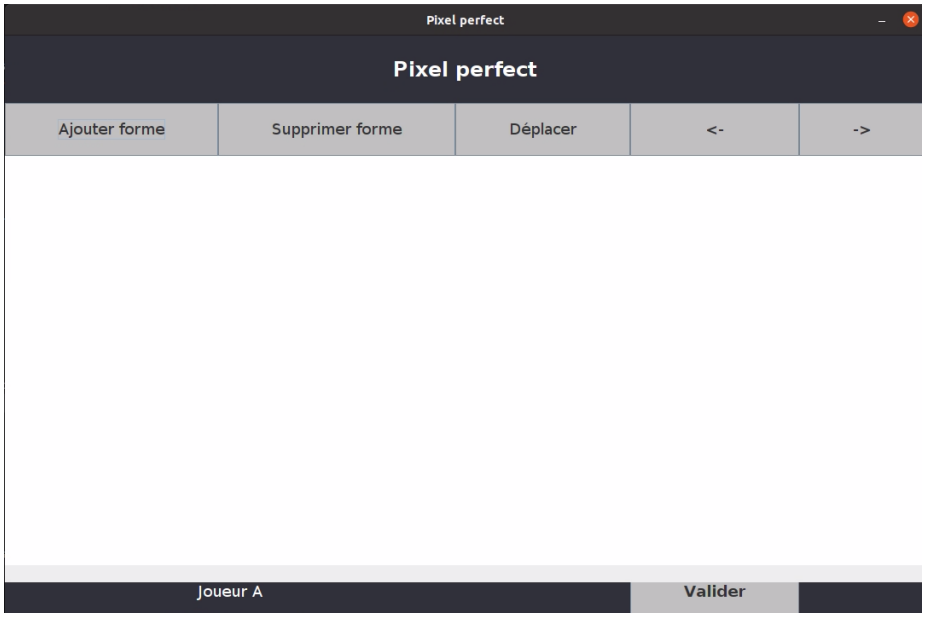


Figure 6: Écran de jeu (vide)

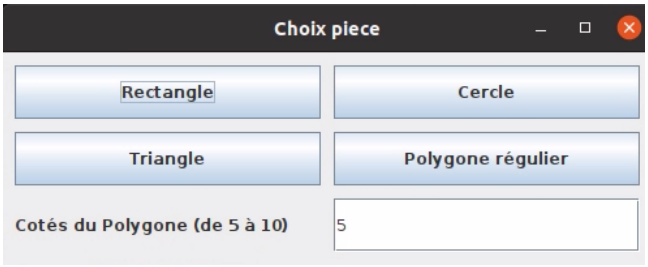


Figure 7: Choix de figure à ajouter

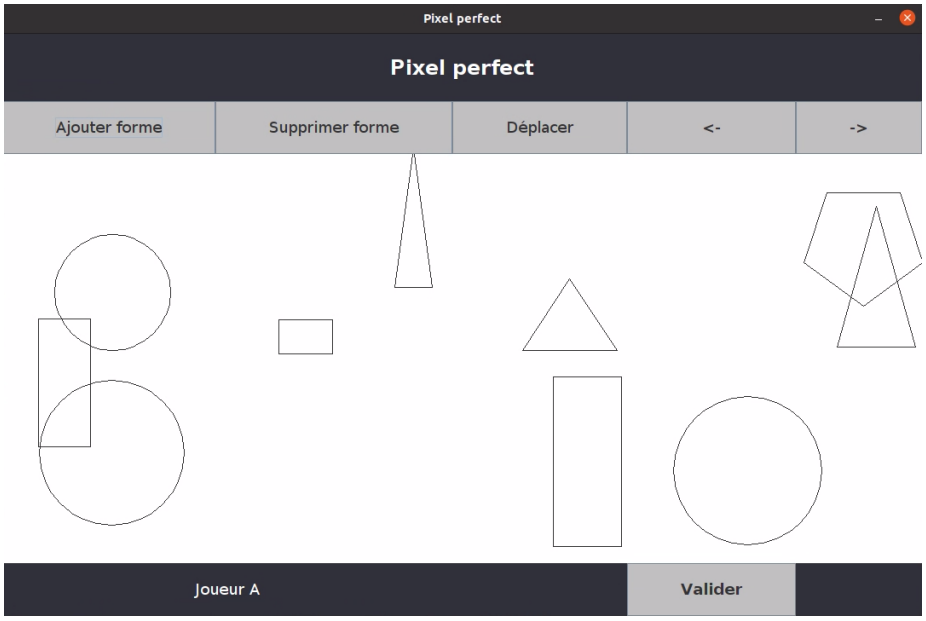


Figure 8: Écran de jeu (après reproduction)


Figure 9: Score de la partie


Figure 10: Score d'une partie à 2 joueurs