GOSSET Séverin

BIGUENET Denis

Université Gustave Eiffel

Rapport

Projet Tangram

Table des matières

[Table des matières 2](#_Toc29581061)

[Introduction 3](#_Toc29581062)

[Manuel utilisateur 4](#_Toc29581063)

[Schéma UML 5](#_Toc29581064)

[Rôle des classes 6](#_Toc29581065)

[Point 6](#_Toc29581066)

[Triangle & Pièce 6](#_Toc29581067)

[Shape 6](#_Toc29581068)

[Button 6](#_Toc29581069)

[Clickable 6](#_Toc29581070)

[Menu 6](#_Toc29581071)

[Drawable 6](#_Toc29581072)

[Action 7](#_Toc29581073)

[Game 7](#_Toc29581074)

[Main 7](#_Toc29581075)

[FileUtils 7](#_Toc29581076)

[MVC 8](#_Toc29581077)

[Compilation 9](#_Toc29581078)

[Conclusion 10](#_Toc29581079)

[Bibliographie 11](#_Toc29581080)

# Introduction

Le but de ce projet est de créer un jeu de Tangram classique, c’est-à-dire reproduire une figure avec des pièces de différentes formes. On souhaite également pouvoir sauvegarder la partie en cours afin de pouvoir la reprendre plus tard ?, ainsi qu’avoir la possibilité de créer un niveau grâce à un éditeur basique intégré au jeu.

# Manuel utilisateur

## Liste des commandes

* Sélectionner une pièce : clic gauche
* Relâcher une pièce : clic gauche
* Faire tourner une pièce : molette de la souris

Ajouter des raccourcis claviers (par exemple les flèches pour déplacer…) ?

## Charger un niveau

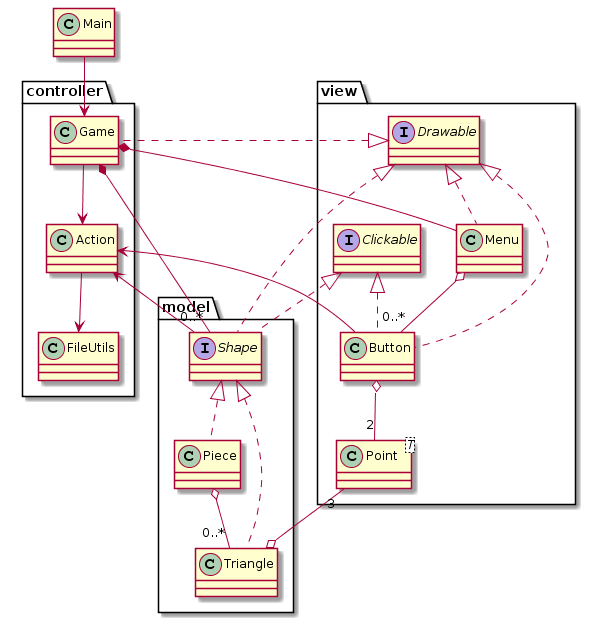
TODO

## Créer un niveau

TODO

# Schéma UML

TODO : mettre à jour l’UML



# Rôle des classes

## Point

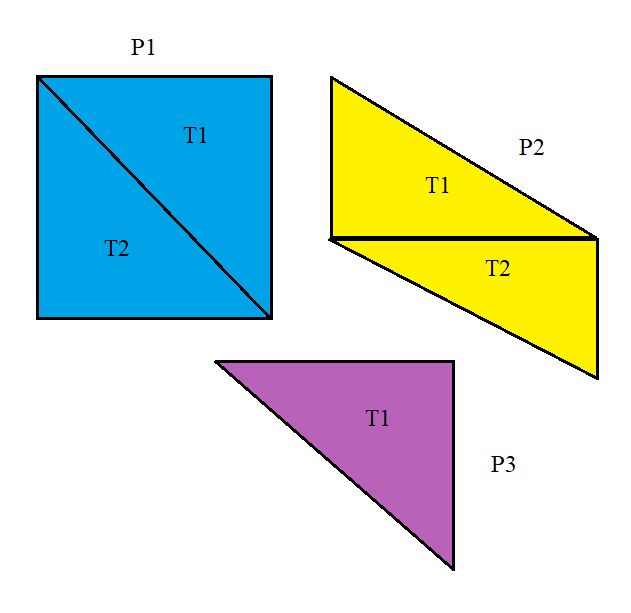
La classe la plus basique du projet est la classe Point, qui représente un point dans un espace deux dimensions, un point possède donc deux attributs x et y pour ses coordonnées. Il peut être judicieux de faire de cette classe un template, cela pourrait avoir plusieurs avantages.

Premièrement, les coordonnées x et y pourront être de n’importe quel type (int, double ou float) sans que l’on ait à surcharger les méthodes ou modifier leur prototype.

Deuxièmement, si le code est récupéré par une autre personne pour le modifier, il n’aura pas à se sourcier du type des valeurs entrées lorsqu’il crée des points.

## Triangle & Pièce

Un triangle est composé de trois points distincts (relation d’agrégation). Ces triangles servent à créer nos pièces. En effet, plutôt que de créer une classe par type de pièce (triangle, parallélogramme…) implémentant une interface, il peut être plus logique de composer nos pièces de triangles (relation d’agrégation). Chaque pièce est donc composée d’une liste de triangles, sur lesquels on peut effectuer des opérations.



Figure

Dans la figure ci-dessus, le carré bleu est constitué de 2 triangles, de même que le parallélogramme jaune. Les autres pièces sont constituées d’un unique triangle.

Il y a plusieurs avantages dans ce design pattern :

* Inutile de créer une classe par type de pièce.
* Une seule implémentation pour chaque méthode peu importe le nombre de forme de pièce possible, toute pièce est traitée comme un ensemble de triangles.

## Shape

Shape permet de définir une interface commune entre les pièces et les triangles, car certaines pièces sont en fait composées d’un unique triangle. Cette interface implémente elle-même l’interface Clickable, car une pièce doit être sélectionnable, ainsi que l’interface Drawable, car on peut dessiner une pièce.

## Button

Un bouton est un carré ou un rectangle définit par un point haut-gauche et un point bas-droit. Etant donné que l’on peut interagir avec un bouton, cette classe implémente donc l’interface Clickable.

## Clickable

Interface commune pour tous les objets alors lesquels le joueur peut interagir, en l’occurrence, les pièces et les boutons.

## Menu

La classe Menu est une agrégation de boutons accompagnés d’une interface, un menu peut être dessiné et implémente donc l’interface Drawable.

## Drawable

L’interface Drawable permet de définir une interface commune pour tout ce qui peut être dessiné, notamment les pièces, les boutons, et le menu.

## Action

Une action est représentée par la fonction à appeler lorsqu’une certaine touche est utilisée par l’utilisateur. SFML propose un système de traitement d’événements (pollEvent et waitEvent), avec lequel on peut récupérer les touches utilisées par l’utilisateur. On peut alors associer un événement à une fonction, pour cela, on a donc ajouté une map statique qui associe un événement à une action.

La classe Action est un foncteur.

## Game

La classe Game sert à définir ce qu’est une partie, c’est-à-dire :

* Une liste de pièces
* Un objectif : une forme à obtenir avec les pièces à disposition

Cette classe implémente l’interface Drawable car elle contient tout ce qui peut être et doit être dessiné à un moment donné de la partie.

## Main

Cette classe crée une partie et n’appelle que des méthodes de celle-ci (condition de victoire, affichage…), afin de déléguer un maximum de tâches et d’écrire un minimum de code dans ce Main.

## FileUtils

Cette classe statique sert à gérer les écritures et lectures de fichiers (charger une partie, sauvegarder une partie).

Dans notre Tangram, on souhaite pouvoir créer un niveau à partir des sept pièces de base, ainsi que charger un niveau. Aussi sauvegarder/charger partie en cours ?

Etant donné que cette classe ne contient pas d’attribut et que l’on ne souhaite pas l’instancier, il est alors logique d’en faire une classe statique, cette classe ne contiendra alors que des méthodes statiques que l’on peut appeler depuis Game ou Action.

# Principales fonctions

## Magnétisme

Un problème est apparu lorsque l’on essayait de reproduire la figure. En effet, il est en effet très difficile de placer une pièce au pixel près, or les coordonnées des pièces placées sont très importantes dans la vérification des conditions de victoires.

Pour résoudre ce problème il y a plusieurs possibilités :

1. Mettre en place une tolérance dans la vérification des conditions de victoire, afin de pouvoir gagner même si les pièces ne sont pas exactement bien placées.
2. Ajouter du magnétisme entre les pièces, c’est-à-dire faire en sorte qu’une pièce soit attirée par une autre pièce lorsqu’on la relâche, mais aussi attirée par la figure à reproduire. Une pièce est attirée par les sommets d’une autre forme si elle est suffisamment à proximité.

La seconde méthode est plus élégante que la première et permet d’éviter en outre un certain nombre de phénomènes comme le chevauchement des pièces.

Le principe du magnétisme est assez simple : on vérifie s’il existe un sommet d’une pièce ou un point de la forme à reproduire à proximité de la pièce que l’on vient de relâcher. On calcule alors la distance entre les deux pièces, dans notre cas, on peut la définir comme la plus petite distance entre deux points de chaque pièce. Si cette distance est inférieure à la distance de magnétisme donnée, alors la pièce relâchée est attirée par l’autre pièce. Dans ce cas on translate la pièce relâchée par le vecteur obtenu à partir des deux points.

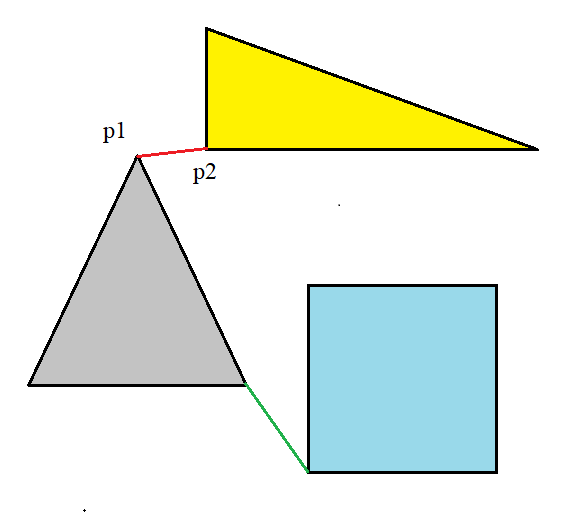


Figure 2

Dans l’exemple si dessus, la pièce grise vient d’être relâchée. Le segment en rouge représente la distance entre les deux pièces, si cette distance est suffisamment petite, alors la pièce grise est attirée par la pièce jaune. On translate alors le triangle gris par le vecteur p2 – p1. Cependant, la pièce grise peut être à proximité d’autres pièces comme le carré bleu. Dans ce cas, la pièce grise est attirée par la pièce la plus proche, le triangle jaune en l’occurrence, puisque le segment rouge est plus court que le segment vert.

TODO : décrire la fonction

## La condition de victoire

TODO

## Charger un niveau



## Sauvegarder un niveau

# MVC

Ce projet suit une architecture MVC, on peut alors le diviser en trois packages Model View et Controller dans lesquels sont répartis nos classes :

* Model : Shape, Piece, Triangle, Point
* View : Drawable, Clikable, Menu, Button
* Controller : Game Action FileUtils Clickable

# Compilation

mkdir build

cd build

cmake ..

make

Pour lancer le projet :

./Tangra

# Conclusion

# Bibliographie