GOSSET Séverin

BIGUENET Denis

Université Gustave Eiffel

Rapport

Projet Tangram

Table des matières

[Introduction 3](#_Toc32404288)

[Manuel utilisateur 4](#_Toc32404289)

[Menu 4](#_Toc32404290)

[Charger un niveau 4](#_Toc32404291)

[Jouer un niveau 4](#_Toc32404292)

[Créer un niveau 4](#_Toc32404293)

[Liste des commandes lors d’une partie 4](#_Toc32404294)

[Schéma UML 5](#_Toc32404295)

[Rôle des classes 6](#_Toc32404296)

[Point 6](#_Toc32404297)

[Triangle & Pièce 6](#_Toc32404298)

[Shape 6](#_Toc32404299)

[Button 6](#_Toc32404300)

[Clickable 7](#_Toc32404301)

[Menu 7](#_Toc32404302)

[Drawable 7](#_Toc32404303)

[Action 7](#_Toc32404304)

[Game 7](#_Toc32404305)

[Main 7](#_Toc32404306)

[FileUtils 7](#_Toc32404307)

[Principales fonctions 9](#_Toc32404308)

[Magnétisme 9](#_Toc32404309)

[La condition de victoire 10](#_Toc32404310)

[Charger un niveau 10](#_Toc32404311)

[Sauvegarder un niveau 10](#_Toc32404312)

[MVC 11](#_Toc32404313)

[Compilation 12](#_Toc32404314)

[Conclusion 13](#_Toc32404315)

[Bibliographie 14](#_Toc32404316)

# 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 la possibilité de créer un niveau grâce à un éditeur basique intégré au jeu, ainsi que de le sauvegarder pour le rendre jouable.

Le jeu doit-être avant tout intuitif et ergonomique à l’utilisation. Pour cela, nous avons mis en œuvre plusieurs solutions.

Tout d’abord, nous avons ajouté du magnétisme entre les pièces, afin que les pièces s’attirent entre elles, et soient attirées par la figure à reproduire. Cette solution évite de devoir placer au pixel près les pièces : le magnétisme le fait tout seul lorsque la pièce est placée à proximité de l’emplacement voulu.

Une tolérance à l’erreur a aussi été mise en place afin que l’utilisateur gagne même si une ou plusieurs pièces ne sont pas parfaitement bien placées. Enfin, l’interface doit-être dans l’idéal *responsive*, et s’adapter à la taille de l’écran.

Parler des limitations

# Manuel utilisateur

## Menu

Le jeu se lance sur un menu principal qui offre plusieurs actions possibles, telles que jouer un niveau, créer un niveau, ou quitter.

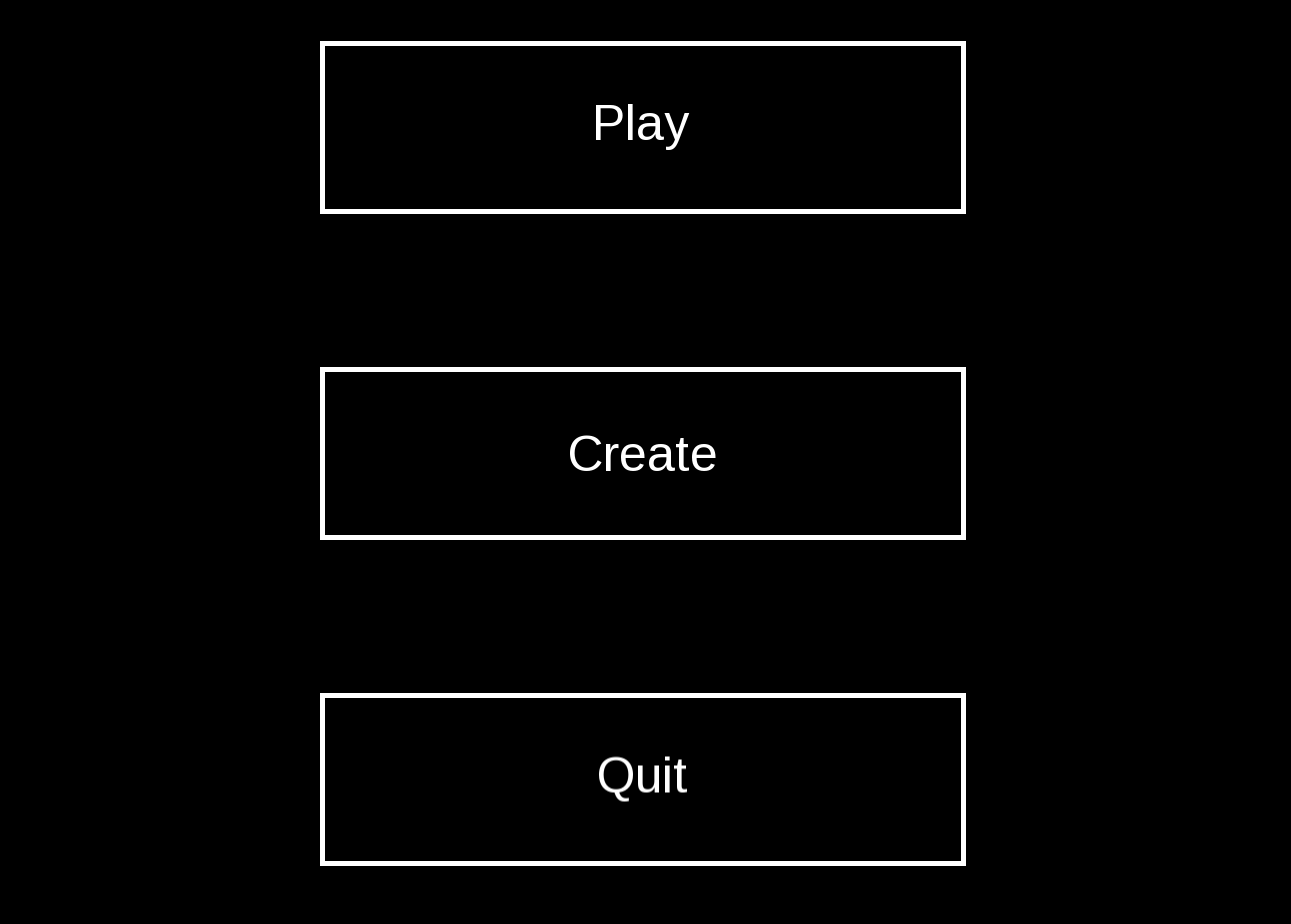


Figure 1 : menu principal du jeu

## Jouer un niveau

Pour jouer un niveau, cliquer sur « Play », un écran de sélection apparait alors avec tous les niveaux proposés, chaque niveau est représenté par un rectangle et la figure à reproduire à l’intérieur. Pour naviguer entre les différents niveaux, cliquer sur les boutons « << «  et « >> », respectivement à gauche et à droite de l’écran. Pour jouer un niveau, cliquer sur le rectangle correspondant.

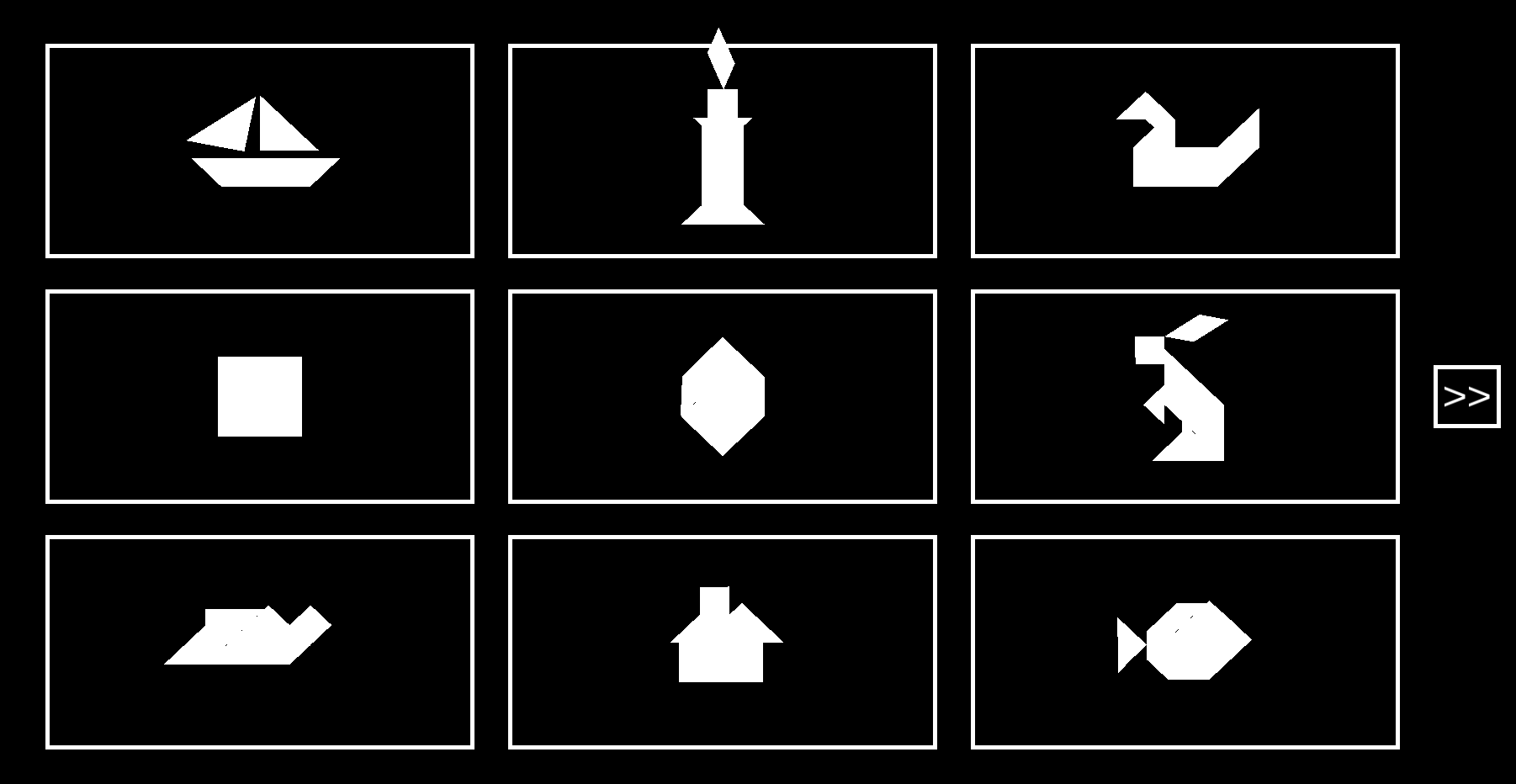


Figure 2 : menu de sélection de niveau

Les boutons « << » et « >> » servent à afficher les autres niveaux quand il y en a trop.

## Jouer un niveau

Une fois que le joueur a sélectionné le niveau de son choix. Il a accès à une interface composée de plusieurs boutons :

* « Load », afin de charger un autre niveau
* « Hint », qui donne un indice à l’utilisateur TODO
* « Menu », pour revenir au menu principal
* « Quit », afin de quitter le jeu

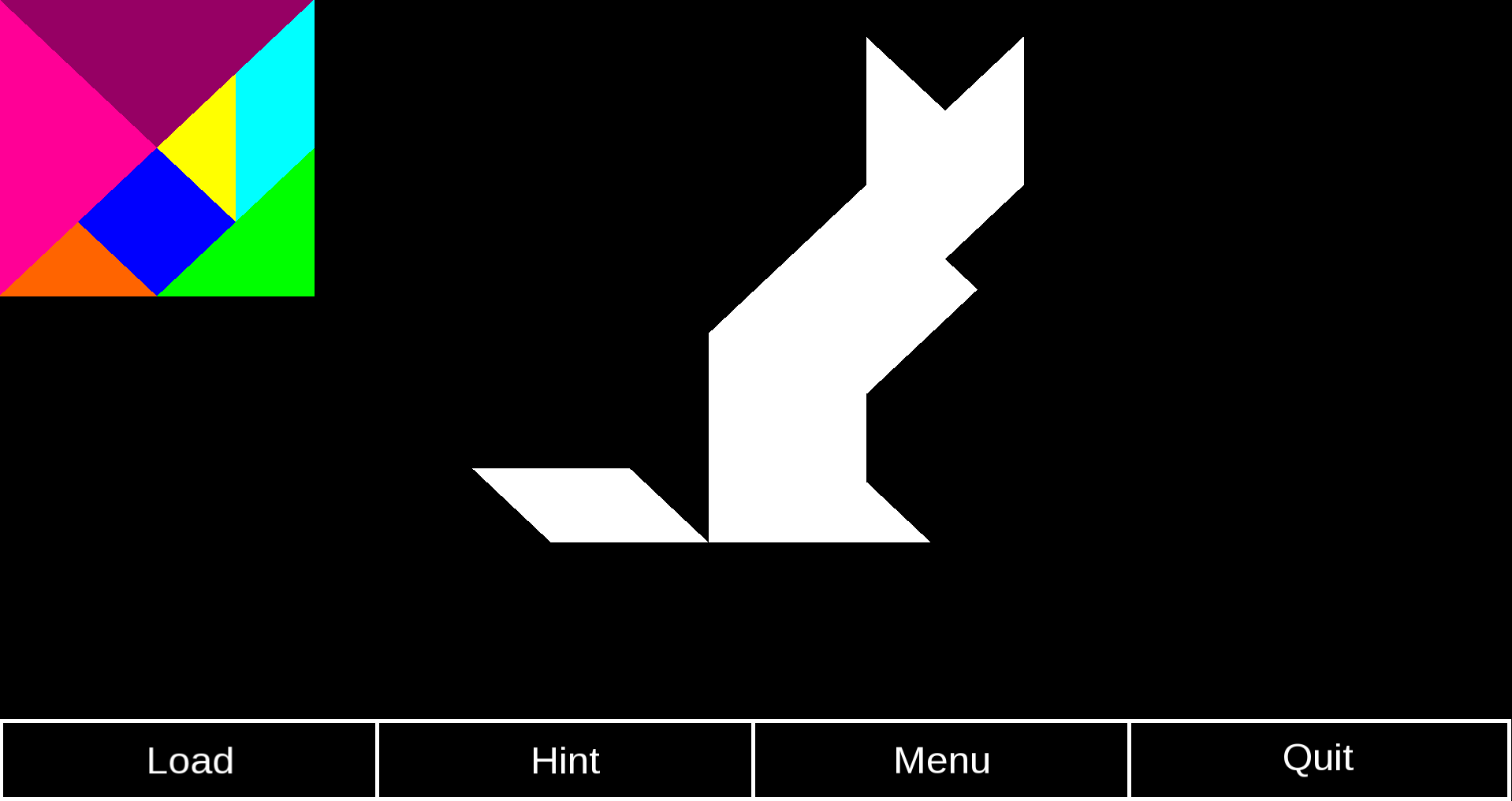


Figure 3 : écran de jeu au cours d’une partie

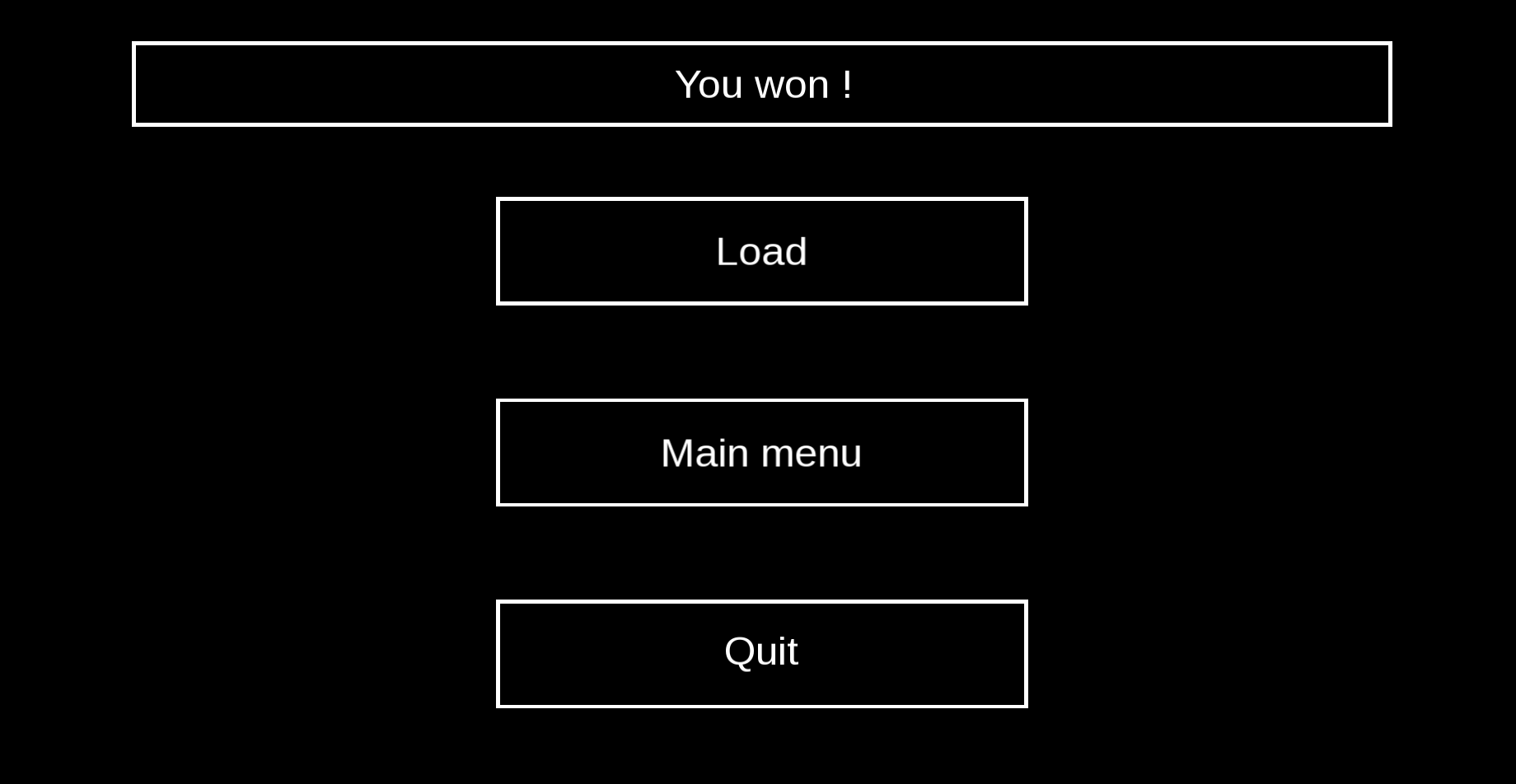


Figure 4 : écran de victoire

## Créer un niveau

Pour créer un niveau, cliquer sur le bouton « Create level » à partir du menu du jeu. Le joueur dispose des sept pièces du Tangram, qu’il peut bouger, faire tourner, afin d’obtenir la forme voulue.

Pour sauvegarder le niveau, cliquer sur le bouton « Save level».

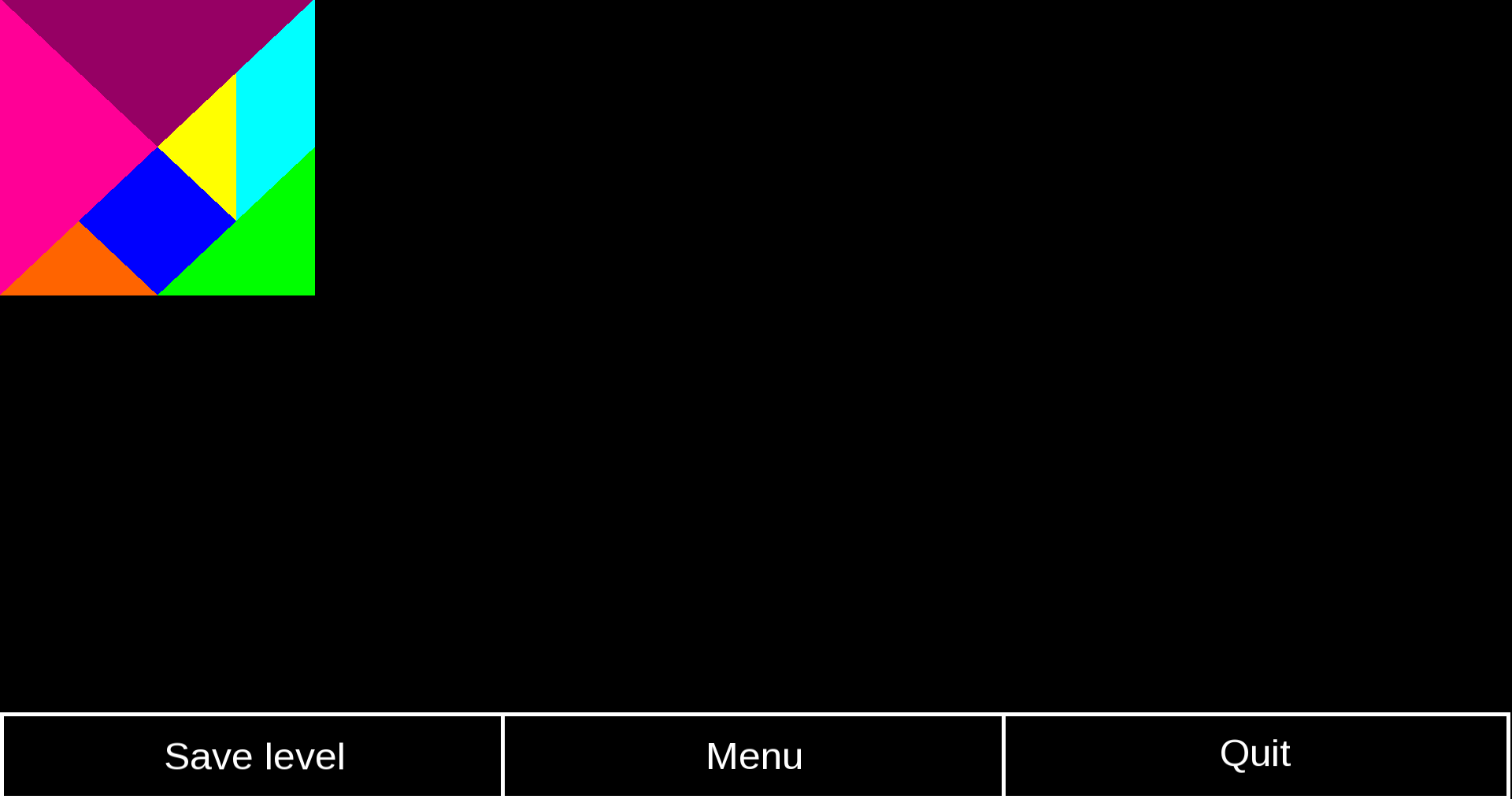


Figure 5 : écran de création d’un niveau

L’utilisateur devrait pouvoir choisir le nom du fichier TODO

## Liste des commandes lors d’une partie

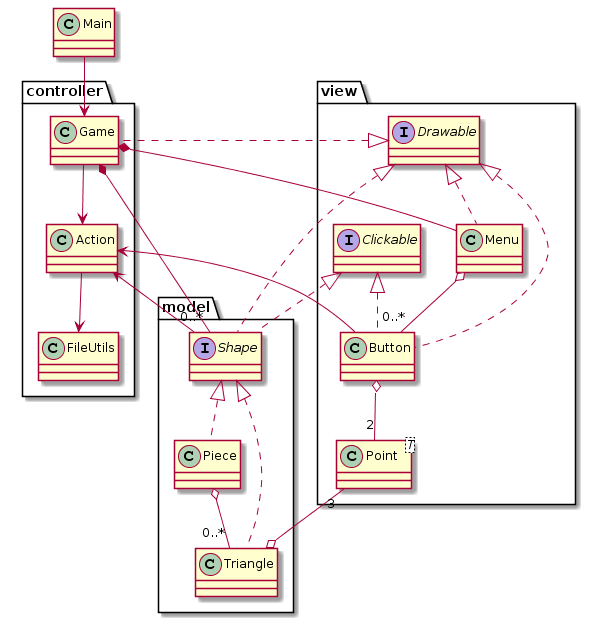
* Sélectionner une pièce : clic gauche
* Relâcher une pièce : clic gauche
* Faire tourner une pièce : molette de la souris

Ajouter d’autres raccourcis claviers (par exemple les flèches pour déplacer…) ? Voir un raccourcis par bouton, exemple : q -> quitter

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 2D, un point possède donc deux attributs x et y pour ses coordonnées. Il peut être judicieux de faire de cette classe un template, nous permettant de choisir le type de données que l’on souhaite (int, float ou double).

## Triangle & Pièce

Un triangle est composé de trois points distincts (relation de composition), ainsi que d’un angle et de trois points bruts, qui correspondent aux trois points précédents sans rotations. En effet, les rotations successives peuvent déformer le triangle ou donner une nouvelle position approximative, il faut alors garder en mémoire la position brute du triangle afin de n’effectuer qu’une seule rotation de thêta degrés dessus.

Chaque pièce est composée d’un certain nombre de triangles, ce qui est très intéressant pour la condition de victoire : chaque pièce ainsi que la forme à reproduire, peut être vue comme un ensemble de triangles basiques. La condition de victoire sera détaillée plus tard dans ce document.

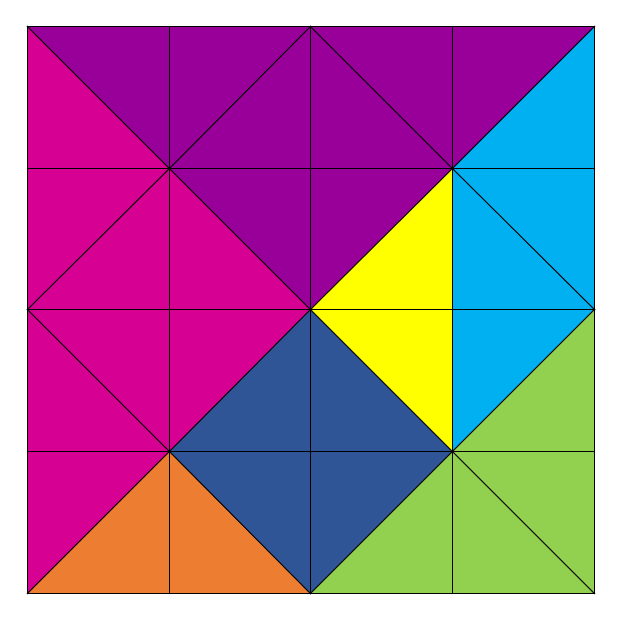


Figure 6 : chaque pièce est composée de plusieurs triangles de base

Il y a plusieurs avantages au design pattern de composition :

* Inutile de créer une classe par type de pièce, il n’y aura que deux classes : Triangle et Piece, peu importe le nombre de formes.
* 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, qui se traitent de la manière. Cette interface implémente elle-même l’interface Clickable, car une pièce doit être sélectionnable par l’utilisateur, 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, un texte et une fonction à appliquer lorsque l’on clique dessus (il s’agit donc d’une lambda). Etant donné que l’on peut interagir avec un bouton, cette classe implémente donc l’interface Clickable mais aussi Drawable.

## Clickable

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

## 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.

## ActionManager

## Menu

Un menu est une agrégation de boutons avec lesquels peut interagir le joueur. Un menu peut être dessiné et implémente donc l’interface Drawable.

## Game

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

* Un ensemble de pièces
* Un objectif : une forme à reproduire avec les pièces à disposition
* Une pièce sélectionnée parmi les pièces à disposition
* Un état (en cours ou terminée)

Cette classe sert à donner un comportement à gérer le déroulement d’une partie : vérifier la condition de victoire, effectuer les transformations sur les pièces (comme les rotations ou les translations). Elle est directement reliée à la classe ActionManager, qui appelle des méthodes de Game selon les événements.

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

## GameManager

Le GameManager est composé d’une Game, d’un Menu, d’un ActionManager, ainsi que d’une fenêtre SFML. Cette classe permet de gérer le jeu dans sa globalité : modifier le Menu et la Game selon les boutons utilisés, dessiner ce qui doit l’être sur la fenêtre, ou même libérer l’espace mémoire quand une partie vient d’être terminée ou abandonnée.

## 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.

## Preferences

Cette classe est un singleton qui sert à stocker les différentes globales, comme le magnétisme, ou la tolérance pour la condition de victoire.

## 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 ou sauvegarder une partie).

Dans notre Tangram, on souhaite pouvoir créer un niveau à partir des sept pièces de base, ainsi que charger un niveau.

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, elle ne contiendra alors que des méthodes statiques que l’on peut appeler depuis les autres classes.

# 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, et ce 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.

Le principe du magnétisme est assez simple : on vérifie s’il existe un sommet d’une pièce ou 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 sommets 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.

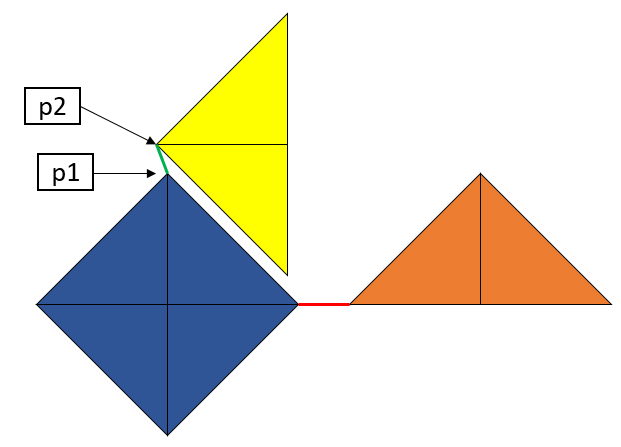


Figure 7

Dans l’exemple si dessus, la pièce bleue vient d’être relâchée. Le segment vert représente la distance entre la pièce bleue et la pièce jaune. Si cette distance est inférieure à la distance de magnétisme donnée dans Preferences.hpp, alors la pièce bleue est attirée par la pièce jaune. On translate alors la pièce bleue par le vecteur p2 – p1. Cependant, la pièce bleue peut être à proximité d’autres pièces comme le triangle orange. Dans ce cas, la pièce bleue est attirée par la pièce la plus proche, le triangle jaune en l’occurrence, puisque le segment rouge est plus long que le segment vert.



Figure  : code de la fonction de magnétisme

finir

## La condition de victoire

Tout d’abord, il paraît évident de ne vérifier cette condition de victoire uniquement lorsque l’utilisateur pose une pièce, puisque l’état du jeu ne devrait pas changer autrement.

La condition de victoire dans notre jeu est assez simple, comme déjà expliqué précédemment dans le document, notre figure à reproduire est en fait une grille composée de triangles élémentaires.



Figure  : code de la condition de victoire

finir

# 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 :

./Tangram

# Conclusion

# Bibliographie