Université de Montpellier
L2 informatique

GOLF MATHÉMATIQUE

RAPPORT DE PROJET T.E.R
PROJET INFORMATIQUE HLIN405

Etudiants:
M. Mike Germain
M. Benjamin Baska
M. Kevin Lastra

Encadrante:
Mme. Annie Chateau

Table de Matières

Introduction

Sous la direction de Mme. Annie Chateau, notre groupe, composé de Mike Germain, Benjamin Baska, Kevin Lastra, a travaillé sur le développement du jeu "Golf Mathématique" comme projet du module HLIN405.

1 Organisation du projet

1.1 Objectifs et cahier des charges

Le but est de créer un jeu de golf, version mathématique.

Base et Règles

Les règles sont simple:

- Partir du point de départ et aller jusqu'au trou en faisant le moins de coup.

- Pour se déplacer, on ne peut que de la portée indiqué sur notre case, dans 8 positions différentes, haut, bas, droite, gauche, ainsi que, haut droite, haut gauche, bas droite, bas gauche.

Interface Graphique

Pour la partie graphique, le but est d'apprendre à utiliser une interface graphique.

Génération automatique de la carte

x2

Intelligence Artificielle

au secours

1.2 Division du travail

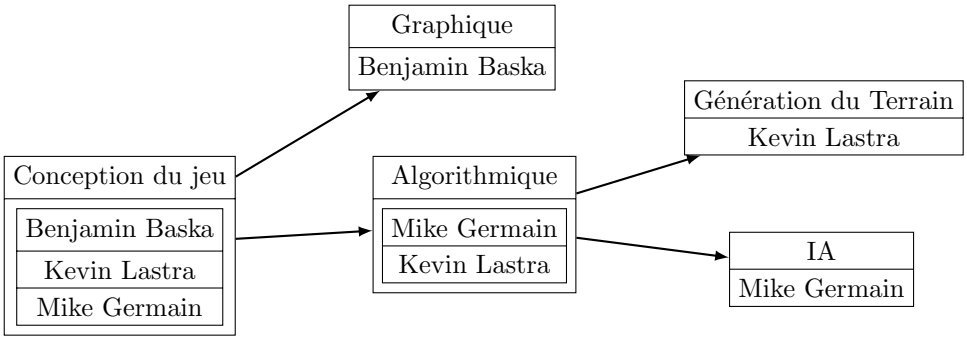

Figure 1: Diagramme de répartition du travail.

1.3 Outils de travail

Langage de programmation
Le langage qu'on a choisi pour le développement du jeu, est le C++ pour 2 raisons principales:

1. Ce langage est un langage de "programmation orienté aux objets" (POO).
2. Grâce aux modules de HLIN202 et HLIN302, on a une base de connaissance avec laquelle on peut travailler de manière confortable.

Outil de modélisation graphique
On a choisi la librairie QT pour les différents avantages qu'elle nous apporte. Cette librairie dispose d'une bonne documentation, elle est adapté au langage C++ et elle dispose aussi d'un outil de travail intéressant appelé "QT Creator", qui rend le travail plus facile.

Travail collaboratif
Nous avons utilisé de multiples programmes:

1. GitHub. Ce logiciel nous permet de partager les avancements du travail réalisé par chacun depuis différents ordinateurs et aussi de sauvegarder plusieurs versions du travail, ce qui est rassurant en cas de perte.
2. Discord. Celui-ci nous permet le partage d'écran, la communication orale et écrit, ce qui est utile pour le développement du jeu.

Éditeur de texte
La production du projet est réalisé grâce à plusieurs éditeur de texte:

1. Éditeur de code - Emacs, sublime et QTCreator.
2. Éditeur LaTeX - TexMaker

2 Conception

Dès la première réunion, en utilisant l'image qui nous a été donné dans notre sujet de projet, on a cherché une architecture de programmation pour laquelle ce jeu s'adapterait au mieux.

La première chose qu'on a défini est la forme du terrain de jeu, Terrain de NxM cases, après la classe Terrain crée on a structuré une classe qui manipulerait toutes les entrées/sorties ("ToutEnUn") et finalement les joueurs avec une méthode "QEvent".

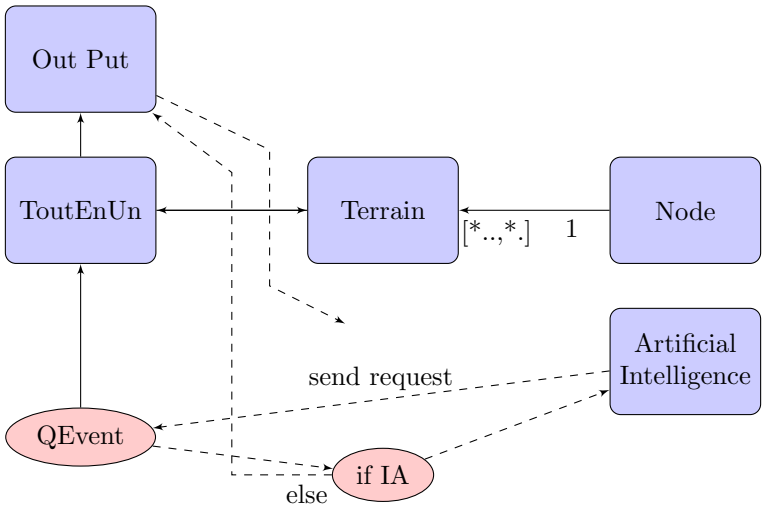

Figure 2: Représentation du flux du jeu.

2.1 Base du jeu

Golf mathématique est une autre version du golf traditionnel, donc cela signifie qu'on modifie ainsi le jeu sans perdre l'esprit du golf.
Le golf est un jeu où plusieurs joueurs jouent tour par tour, la personne gagne si elle a fait le moins de coup, donc le moins de points.

Donc pour définir les bases de notre jeu on doit utiliser les règles du jeu original.
Dans un Terrain de golf, on a:

- Départ (zone où les joueurs commencent)
- Cible (le trou)
- Obstacles (zone d'eau)

Et puis, le joueur se déplace et tape la balle avec une puissance, la portée.
Avec tout ça, on a des éléments avec lesquels on peut travailler: un terrain que l'on interprétera comme un tableau d'entier, les valeurs étant la portée de la balle.
Un terrain sera divisé en 4 zones différentes:

- Départ
- Eau
- Herbe
- Trou

2.2 Graphique

Choix de la librairie graphique

Au début du projet, nous nous avions décidé d'utiliser la librairie graphique :

Mais le fait qu'elle soit adapté au langage C, et donc pas prévu pour de l'orienté objet nous a posé quelque soucis.
Après cela, nous avons chercher ainsi une autre librairie, celle ci adapté au C++. Nous nous sommes alors mis d'accord sur Qt. Cette librairie offre une grande documentation et un éditeur de texte adapté ("Qt Creator").

Les premiers essais pour la partie graphique ont pu commencer, avec en premier les cases de jeu :

Figure 3: Première version des dessins

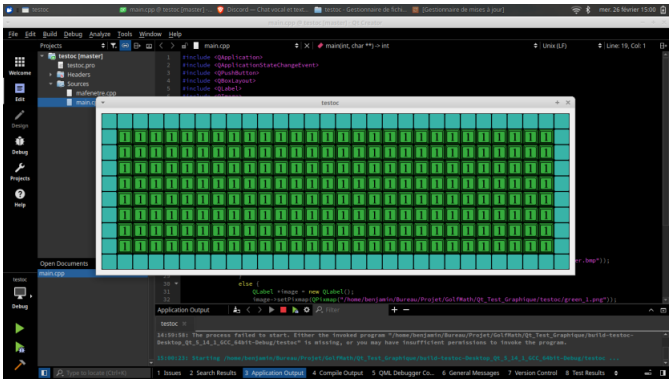

Figure 4: Premier test de terrain

Le résultat étant trop sombre, on a refait les cases avec un rendu plus clair et plus lisible.

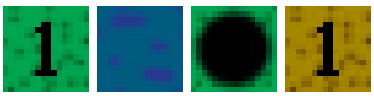

Figure 5: Rendu deuxième version

Figure 6: Rendu final

2.3 Qt

La création d'une interface

Pour pouvoir créer une interface graphique, on utilise la librairie et les objets de Qt, tel que QApplication qui permet de gérer l'interface graphique de l'utilisateur (ou GUI en anglais), ainsi que de contrôler le flux d'entrée et de sortie et les paramètre principaux de notre interface.
Voilà à quoi ressemble le code principal pour créer une fenêtre :

```
#include <QApplication>
#include "toutenun.h"
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow fen;
    fen.show();

    return app.exec();
}
```

```
QApplication app(argc, argv);
return app.exec();
```

C'est cela qui va créer une première fenêtre.
Ensuite les autres objets que nous appellerons, hériteront de la classe QObject, une autre classe importante de Qt qui permet l'affichage de widget.

```
MainWindow fen;
fen.show();
```

Cela appelle notre classe MainWindow qui est la fenêtre d'accueil de notre jeu.

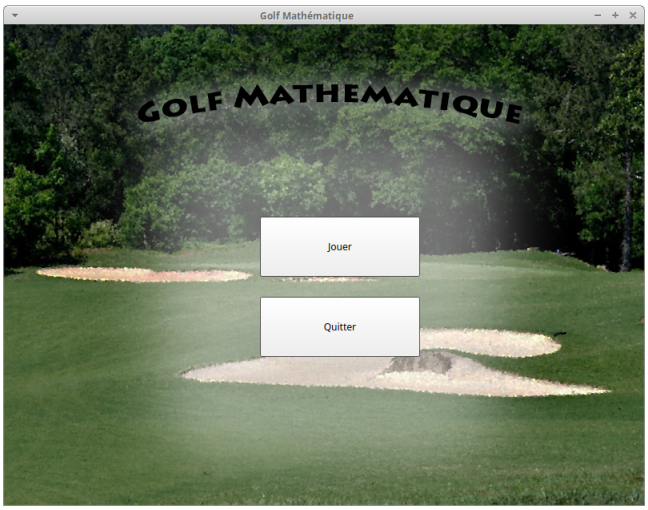

Figure 7: Accueil

La classe de cette fenêtre est :

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
    setWindowIcon(QIcon("./image/ballegolf.png"));
    setWindowTitle("Golf Mathématique");
    setFixedSize(800, 600);
    QPalette fond;
    fond.setBrush(backgroundRole(), QBrush(QPixmap("./image/v2/background.jpg")));
    setPalette(fond);

    jouer = new QPushButton("Jouer", this);
    quitter = new QPushButton("Quitter", this);
    jouer->setFixedSize(200, 75);
    quitter->setFixedSize(200, 75);
    jouer->move(320, 240);
    quitter->move(320, 340);

    connect(jouer, SIGNAL(clicked()), this, SLOT(lancer()));
    connect(jouer, SIGNAL(clicked()), this, SLOT(close()));
    connect(quitter, SIGNAL(clicked()), qApp, SLOT(quit()));
}
```

```
void MainWindow::lancer()
{
    Niveaux *niv = new Niveaux;
    niv->show();
}
```

Ici nous pouvons voir plusieurs nouvel chose.

```
setWindowIcon(QIcon("./image/ballegolf.png"));
    setWindowTitle("Golf Mathématique");
    setFixedSize(800, 600);
    QPalette fond;
    fond.setBrush(backgroundRole(), QBrush(QPixmap("./image/v2/background.jpg")));
    setPalette(fond);
```

Toute cette partie s'occupe de l'apparence de la fenêtre, son nom et son icône.

```
    jouer = new QPushButton("Jouer", this);
    quitter = new QPushButton("Quitter", this);
    jouer->setFixedSize(200, 75);
    quitter->setFixedSize(200, 75);
    jouer->move(320, 240);
    quitter->move(320, 340);
```

Celle ci s'occupe des widgets comme les boutons.

```
    connect(jouer, SIGNAL(clicked()), this, SLOT(lancer()));
    connect(jouer, SIGNAL(clicked()), this, SLOT(close()));
    connect(quitter, SIGNAL(clicked()), qApp, SLOT(quit()));
```

Ensuite, on attribut un effet à nos boutons.

```
void MainWindow::lancer()
{
    Niveaux *niv = new Niveaux;
    niv->show();
}
```

Et enfin, cette méthode appelle notre nouvel fenêtre qui gère nos différents niveaux.

Les niveaux lance alors différentes seeds (référence des niveaux). Prenons comme exemple le niveau 1 :

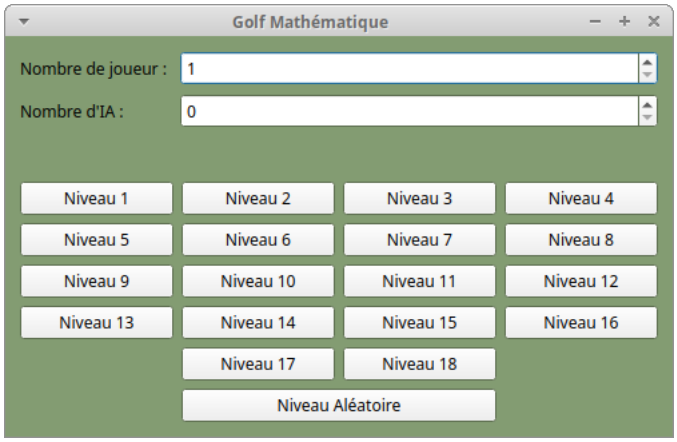

Figure 8: Page des niveaux

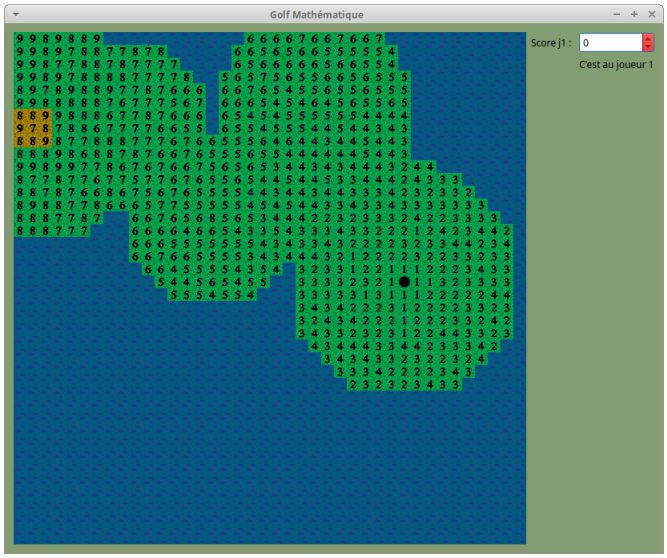

Figure 9: Niveau 1

2.4 Intelligence Artificielle - IA

2.4.1 Une IA pas très futée

La première idée pour faire l'ia était de faire une basiques qui avance tout droit vers le trou sans réfléchir. Son objectif a chaque coup calculer simplement le coup qui se rapprochait le plus du trou avec une boucle 'for' très simple.
Un problème en est ressorti, c'est peu efficace. J'ai donc décidé de reprendre du début.

2.4.2 Une IA un peu plus complexe

Et donc après avoir recherché des algorithmes de pathfinding sur internet un algorithme en particulier en est ressortit assez souvent : A* (ou A star)
C'est un algorithme de pathfinding très connu qui ressemble a ceci :

```
Fonction cheminPlusCourt(g:Graphe, objectif:Nœud, depart:Nœud)
       closedList = File()
       openList = FilePrioritaire(comparateur=compare2Noeuds)
       openList.ajouter(depart)
       tant que openList n'est pas vide
           u = openList.depiler()
           si u.x == objectif.x et u.y == objectif.y
               reconstituerChemin(u)
               terminer le programme
           pour chaque voisin v de u dans g
               si non(v existe dans closedList ou si v existe
                dans openList avec un cout inférieur)
                    v.cout = u.cout +1
                    v.heuristique = v.cout + distance([v.x, v.y],
                    [objectif.x, objectif.y])
                    openList.ajouter(v)
           closedList.ajouter(u)
       terminer le programme (avec erreur)
```

J'ai ensuite du évidemment adapter l'algorithme à notre projet, pour se faire j'ai d'abord du créer une classe qui a une 'node' associe un 'cout' (le nombre de coups jusqu'ici), une heuristique (la distance "a vol d'oiseau" jusqu'à l'arrivée) et un père, pour à la fois pouvoir recréer le chemin après le pathfinding mais aussi tout simplement pour le bon fonctionnement de l'algorithme.
Et après plusieurs complications cette IA était enfin terminée et fonctionnelle. A savoir que, bien que l'IA trouve le meilleur chemin à chaque fois, elle n'est pas invincible, il n'est pas compliqué de faire égalité ou même de la battre car j'ai fais en sorte qu'elle ne choisisse pas son point de départ de manière optimale.

2.5 Génération du Terrain

2.5.1 Génération Automatique

Pour générer un terrain de manière automatique on doit placer des règles:

- Le Terrain doit être connexe ou au moins cheminable.
- La génération des portée doit être presque linéal.

Après avoir testé différentes idées, nous avons trouvé une solution très convenable pour construire un terrain. Pour générait notre terrain on va produire un chemin est sur celui la on va faire apparaître la surface jouable.
Cette algorithme est divise en deux passage sur une grille:

Premier Passage

On produit un premier point de manière aléatoire ($P_{x,y}$), et à partir de ce point on génère n-1 autres points ($P_{x,y}$,...,Pn_{x_n,y_n}).

$$k \leq \text{n et } k > 0, P_k = P_{k-1} + \lambda(a,b)$$

Telle que le vecteur (a, b)$\in$V, avec V l'ensemble des 7 différentes directions valides représenté avec des vecteurs.

$$\text{V}=\{(-1,-1),(-1,0),...,(1,1)\}$$
$$\text{Et } \lambda = B_i \text{ avec B} = \{\text{n-1,n-1,n-2,n-3,...,4,3,2,2}\}.$$

Avec cet algorithme on trouverait que:

X:"l'ensemble des terrains générés"
Y:"l'ensemble des chemins possibles"
$\forall x \in X, \exists y \in Y$ telle que "y" est un chemin valide.

Avec cette phrase on pouvait générer le terrain sans avoir à penser aux possibles contraintes comme la connexité.

Pseudo Code

Algorithm 1: GénérationChemin(d lo: entier, d la: entier,d n: entier): Tableau d'entier

variables : Tab: tableau d'entier bidimensionnel de taille n*m.
i: entier.
v: structure vecteur contenant une paire d'entier "x" et "y".

début algorithme:

```
i ← 1;
// on définit v le premier point du chemin.
v.x ← random()%(lo/2);
v.y ← random()%(la/2);
Tab[v.x][v.y ] ← 1;
while i < n do
    dir ← getdirection();
    // renvoi un nombre entre 0-7 aléatoirement, qui
       représente les 8 diffèrent directions
    v ← nouvelPosition(v,dir);
    // renvoi un vecteur qui représente la nouvel position
       v_{n+1} par rapport a la direction et la position v_n;
    if v est valide && Tab[v.x][v.y ] == 0 then
        Tab[v.x][v.y ] ← 1;
return Tab;
```

Cet algorithme va nous renvoyer un tableau d'entier, le quelle on peut représente avec un graphe:
GénérationTerrain(n,m,5);

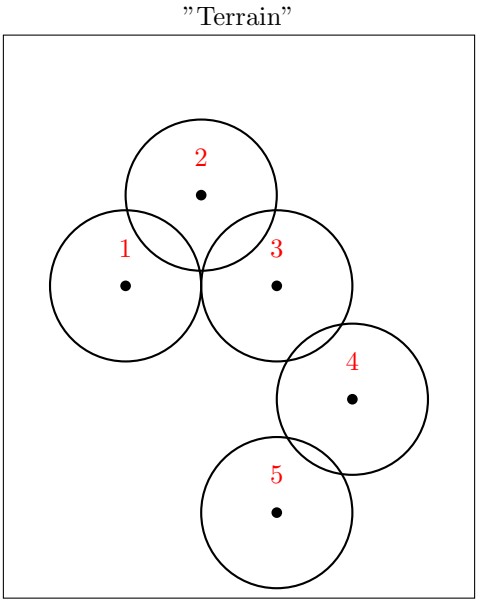

Figure 10: les cercles représente la zone ou le Terrain va ce générait.Anexe 0.

Deuxième Passage

Le chemin ainsi généré, on va passer case par case de notre grille et on va teste si la distance entre la case est un point du chemin est inférieur à un certain nombre.

On pose (x, y) une case.
T l'ensemble des points du chemin.
λ le rayon.
Si distance((x, y), T_i) $< \lambda$ alors la case est valide.

si une case est valide ça veut dire que cette case est partie de la surface de notre terrain, donc on doit lui donne une portée, cette portée est définit par la formule suivante:

$$\text{portée} = (2 + (\frac{dist*8}{sranyon}\%8)) + (((rand()\%5) - 2)\%2)$$

Cette formule est divisée en 2 partie, représenté par les couleurs:

1. $(2 + (\frac{dist*8}{sranyon}\%8))$

dist est la distance entre le dernier point du chemin et la case actuel.
srayon est la distance entre le dernier point du chemin et le point valide le plus éloigner du dernier point du chemin.
A l'aide de GeoGebra on peut représenter cette formule de manière plus graphique.

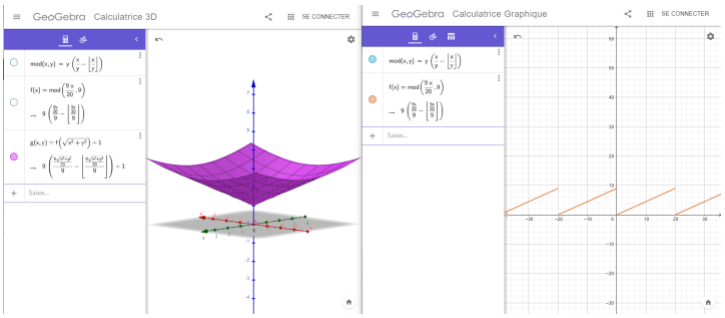

Figure 11: srayon = 20, 9 au lieu 8 et 1 au lieu 2. La portée d'un point est équivalent a l'axe z.

2. $(((rand()\%5) - 2)\%2)$

rand() un nombre aléatoire entre 0 et RAND_MAX(un nombre très grand qui dépend du langage);

$rand()\%5 \leftarrow$ est un nombre entre 0 et 4
$((rand()\%5) - 2) \in \{-2, -1, 0, 1, 2\}$

3 nombre pair(60%) et 2 nombre impair(40%).

$$(((rand()\%5) - 2)\%2) \in \{-1, 0, 1\},$$

ça veut dire qui est plus probable d'avoir 0 que 1 ou -1 donc 60% de probabilité de ne pas modifier et 40% pour modifier.

Pseudo Code

Algorithm 2: GénérationTerrain(d Tab: tableau d'entier,d Ch:tableau de vecteur(les points du chemin),d lo:entier,d la:entier): Tableau d'entier

```
variables :
début algorithme:
ext ← extrem(Tab);
// renvoi le point le plus éloigner du dernier point du
   chemin(Ch[n-1]).
for x ← 0 to lo do
    for y ← 0 to la do
        if dansRayon(Tab[x][y ],Ch) et Tab[x][y ] ∉ Ch then
            // dansRayon renvoi vrai ou faux dépendant si un
               point (x, y) est dans le rayon d'un point du
               chemin.
            Tab[x][y ] ← calcPortee(ext,Ch[n-1],Tab[x][y ]);
            // calcPortee renvoi la porte d'un point utilisant
               la formule vue avant.
return Tab;
```

"Terrain"

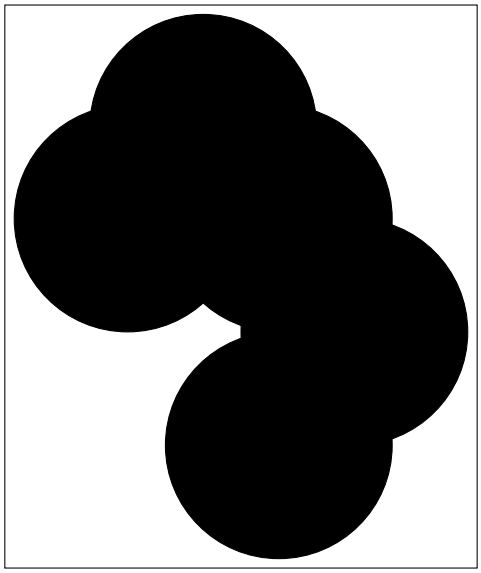

figure 0: les zones noire représenter la surface jouable.Anexe 0.

3 Bibliographie

Pour la partie graphique on a utilisé les sites suivants :

- wiki.libsdl.org/FrontPage
- openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/17117-installation-de-la-sdl
- doc.qt.io/
- openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1898935-initiez-vous-a-qt
- www.youtube.com/channel/UC6CdzK3QAxtr7giBwqk5eUA