# 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:**
Mm. Annie Chateau

# Table de Matières

# Introduction

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

# 1 Organisation du projet

## 1.1 Objectifs et cahier des charges

wtf

### Base et Règles

aucune idée

### Interface Graphique

pire

### Génération automatique de la carte

x2

### Intelligence Artificielle

au secours

## 1.2 Division du travail

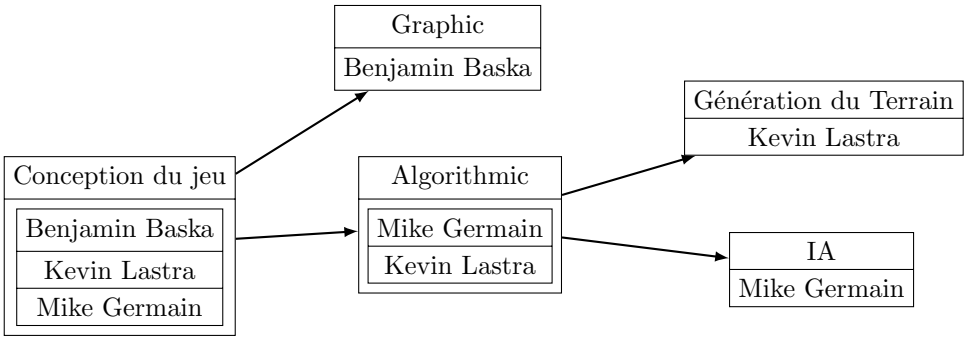


Figure 1: Diagramme de répartition du travail.

## 1.3 Outils de travail

### Langage de programmation

Le langage qu'on à choisi pour le développement du jeux, est le C++ pour 2 raisons principales:

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

### Software de programmation graphique

On a choisit la librairie QT pour les différents avantages qu'elle nous apporte. Cette librairie dispose d'une bonne documentation, elle é adapté au langage C++ et elle dispose aussi d'un outil de travail très 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 très rassurant en cas de perte.
2. Discord. Ce logiciel nous permet le partage d'écran, la communication orale et écrit, ce qui est très utile pour le développement du jeux.

### Éditeur de texte

La production du projet est réalise grâce a plusieurs éditeur de text:

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

# 2 Conception

De la premier réunion, utilisant l'image proportionnée dans le sujet du projet, on à travaillé dans une architecture, pour le quel ce jeux s'adapterait mieux.

La premier chose qu'on à définit est la structure du terrain de jeux, Terrain de NxM Node, après la classe Terrain crée on a structuré une classe qui manipulerait tout les entre/sortie ("GameMaster") et finalement les joueur avec une classe "PlayerController".

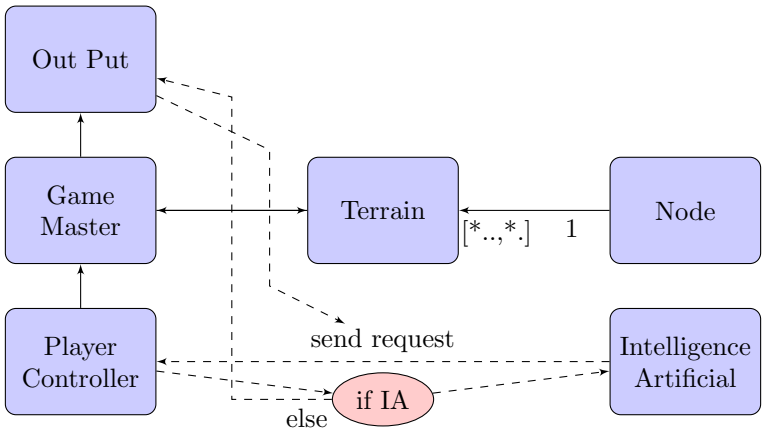


Figure 2: Représentation du flux du jeu.

## 2.1 Base du jeu

Golf mathématique est un sous type du golf original, donc cela signifie qu'on doit modifier le jeu sont perd l'esprit du golf. Le Golf est un jeux de plusieurs joueur par tour, la personne qui gagner est celui qui a moins de points, etc... Donc pour définir les bases de notre jeu on doit utiliser les règles du jeu original et les objets qui lui entour. Dans un Terrain de golf, on a un:

- Départ(zone ou les jouer commence.)
- Cible(le trou)
- Obstacles(zone d'eau ou sable)

Un joueur peut avoir plusieurs position et tape la balle avec une portée. Avec tous ça on trouve des éléments avec le quelle on peut travailler. Un terrain on

l'interprétera comme un tableau d'entier les quelles la valeur sera la portée de la balle. Un terrain sera divise on 3 zones différents:

- Eau
- sable
- Herbe

## 2.2 Graphic

## 2.3 Intelligence Artificielle - IA

## 2.4 Génération du Terrain

### 2.4.1 Génération Automatique

Pour générait un terrain de manière automatique on doit place des règles:

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

Après avoir testé différents 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 apparente la surface.
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)∈V, avec V l'ensemble des 7 différents directions valides représente avec des vecteurs.

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

Avec cette algorithme ont 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 pouvais générer le terrain sans avoir a 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;
```

Cette algorithme va nous rendre un tableau d'entier, le quelle on peut représente avec un graph:
GénérationTerrain(n,m,5);

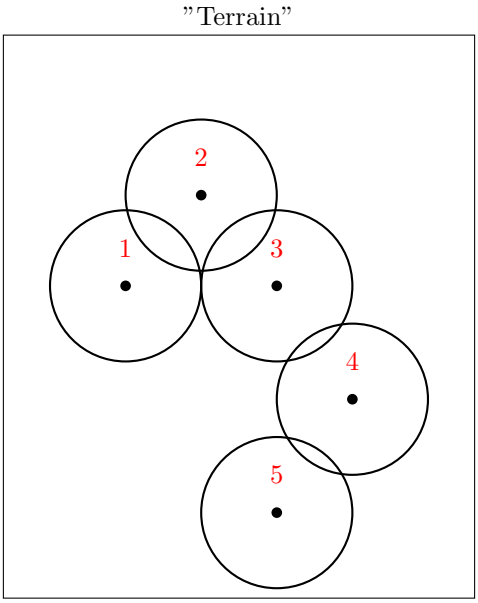


Figure 3: les cercles représente la zone ou le Terrain va ce générait.Anexe 0.

## Deuxième Passage

Déjà générait le chemin, 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 a un certain nombre.

On pose (x, y) une case.
T l'ensemble des points du chemin.
$\lambda$ 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 porté, cette porté est définit par la formule suivant:

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

Cette formule est divisé on 2 partie, représente par les couleur:

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 graphic.

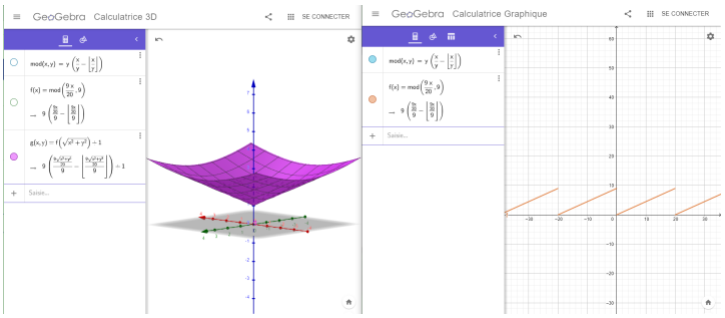


Figure 4: srayon = 20, 9 au lieu 8 et 1 au lieu 2. La portée d'un point est équivalent a l'axe z.

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

rand() un nombre aléatoire entre 0 et RAND_MAX(un nombre très grand qui dépens 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"

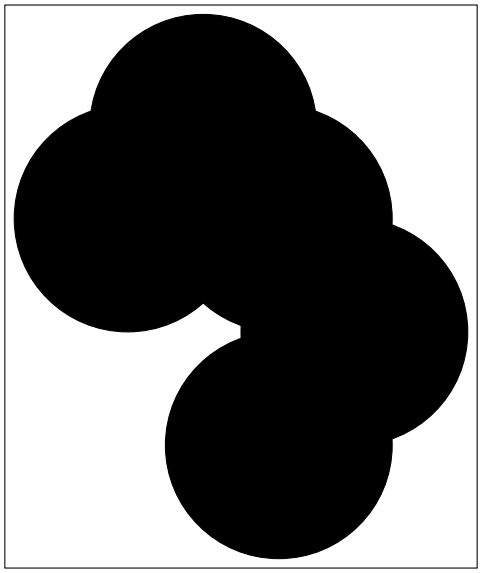

figure 0: les zones noire représenter la surface jouable.Anexe 0.

# 3 Bibliographie

# 4 Annexes