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 simples:

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

- On ne peut que se déplacer de la portée indiquée sur notre case, dans 8 positions différentes, haut, bas, droite, gauche, ainsi que, haut droite, haut gauche, bas droite, bas gauche (case entouré en noir sur la figure ci-dessous).

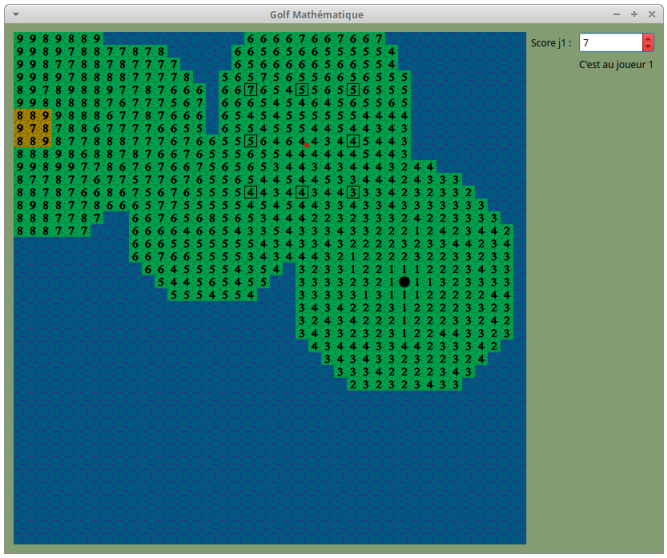

Figure 1: Exemple de déplacement

Interface Graphique

Pour la partie graphique, le but est d'apprendre à utiliser une interface graphique, et rendre plus fluide et convivial le jeu.

Génération automatique de la carte

Pouvoir générer des nouveaux terrains pour le jeux et c'était aussi un défi algorithmique.

Intelligence Artificielle

au secours

1.2 Division du travail

Nous avons tout d'abord, pris le temps de bien structurer le jeu, ce mettre tous d'accord sur comment on voyait le golf mathématiques.
Puis, nous avons séparé le travail en fonction des affinités de chacun, Kevin pour la partie de création d'un algorithme de génération de terrain, Mike pour ce qui est de la création d'une intelligence artificielle et Benjamin pour la partie visuel du jeu et l'adaptation avec la librairie graphique.

Figure 2: Diagramme de Gantt

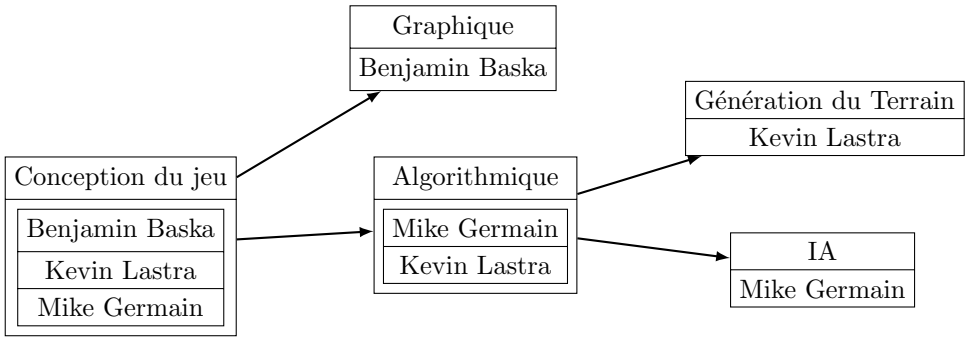

Figure 3: Diagramme de répartition du travail.

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 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 plusieurs programmes:

1. GitHub. Ce gestionnaire de version 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. https://github.com/kevinlastra/GolfMath.git
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 réalisée 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. Nous avons opté pour une modélisation objet qui permet de bien délimiter chaque élément du jeu.

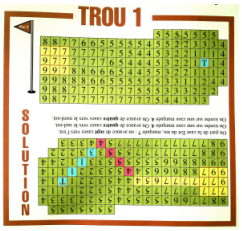

Figure 4: Image du sujet

La première chose qu'on a défini est la forme du terrain de jeu, Terrain de NxM cases, la forme rectangulaire est la plus adaptée pour gérer le terrain dans l'interface mais le "green" lui-même a sa forme propre à l'intérieur de ce rectangle. La classe Terrain crée on a structuré une classe qui manipulerait toutes les entrées/sorties ("ToutEnUn") et finalement les joueurs avec une méthode "QEvent".

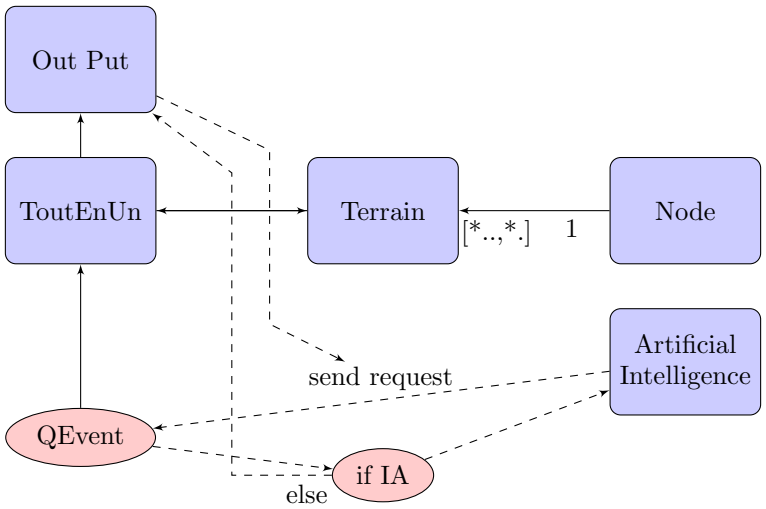

Figure 5: Représentation du flux du jeu.

Quand le joueur lance le jeu, il appel la classe "ToutEnUn", qui crée un terrain avec la classe "Terrain", qui celui-ci appel plusieurs case de la classe "Node". Le terrain est ainsi généré.
Ensuite quand le joueur clique, il envoie une requête, "send request", à "QEvent", et fait ainsi avancé le jeu, couleur rose du graphique.
Si il y a une IA, "if IA", au moment de la requête, l'IA est donc appelé grâce à une méthode, et ainsi retourne à "QEvent" et le jeu continue.

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)

Ici pour des raisons de simplicité nous avons modélisé une seule sorte d'obstacles, mais il serait envisageable d'enrichir le jeu avec différents types (forêt, sable, pierre) qui entraîneraient différents comportements.

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'entiers, les valeurs étant la portée de la balle.
Un terrain sera divisé en 4 zones différentes (voir aussi figure 9):

- Départ, case jaune, c'est sur celle-ci que la partie commence mais elles ont les attributs d'une case herbe.
- Eau, case bleu, cette case nous fait perdre un tour si on y va dessus et nous renvoie à notre dernière position.
- Herbe, case verte, elles sont les briques de notre jeu, elles ont chacune une portée.
- Trou, case avec un rond noir, elle est unique et met fin à la partie.

Un dernier type "sable" avait été envisagé pour des cases de faibles portées, mais du au confinement cela a été enlevé.

2.2 Graphique

Choix de la librairie graphique

Au début du projet, nous nous avions décidé d'utiliser la librairie graphique SDL[1] :

Figure 6: Librairy SDL.

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 cherché ainsi une autre librairie, celle ci adapté au C++. Nous nous sommes alors mis d'accord sur Qt [2]. 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 7: Première version des dessins

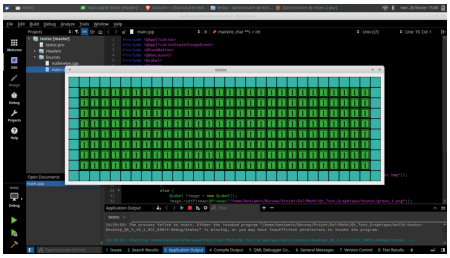

Figure 8: Premier test de terrain

[1] https://www.libsdl.org/
[2] https://www.qt.io/

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

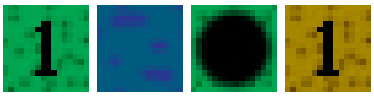

Figure 9: Rendu deuxième version

Figure 10: 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 paramètres 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.

La classe de cette fenêtre est :

```
#include "mainwindow.h"
```

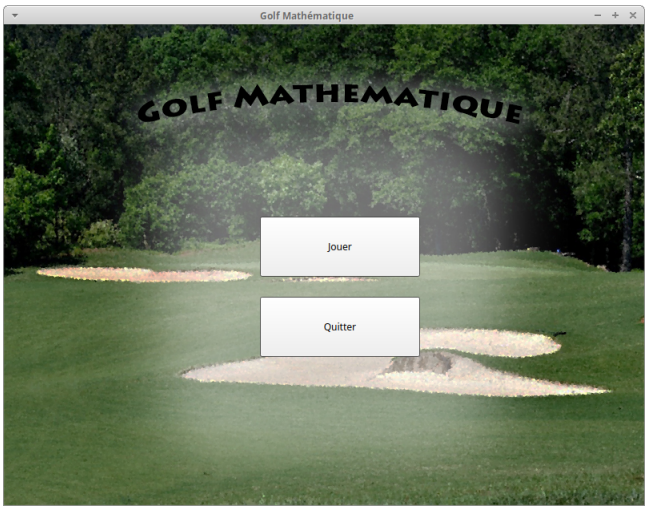

Figure 11: Accueil

```
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 nouvelles choses.

```
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 nouvelle fenêtre qui gère nos différents niveaux (figure 12).

Les niveaux lancent alors différentes seeds (référence des niveaux), ces seeds servent à générer à la volée de nouveaux terrains (figure 13).

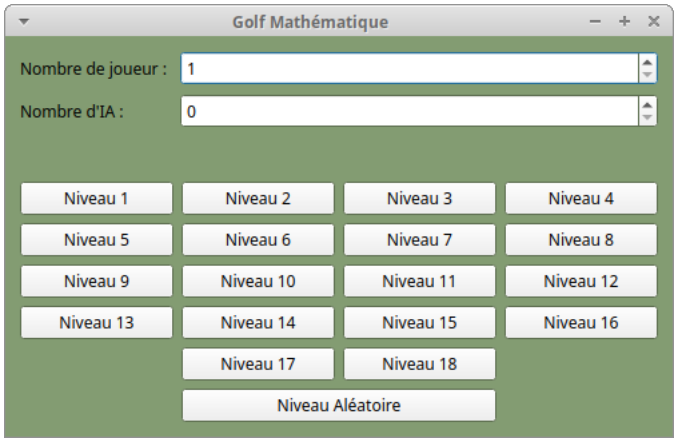

Figure 12: Page des niveaux

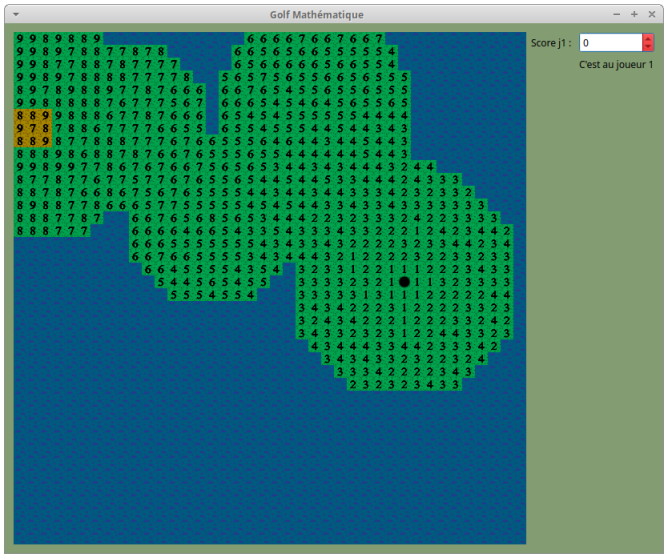

Figure 13: Niveau 1

2.4 Intelligence Artificielle - IA

2.4.1 Une IA pas très futée

La première idée pour faire l'IA était de faire une basique 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 dans le sens où le nombre de coups était très élevé par rapport à ce que pouvaient faire les joueurs humains. Cela manque d'intérêt et il vaut mieux jouer contre un adversaire un peu plus fort. Nous avons donc décidé de reprendre du début.

2.4.2 Une IA un peu plus complexe

Après avoir recherché des algorithmes de recherche de chemins optimaux (pathfinding) sur internet, un algorithme en particulier en est ressorti assez souvent : A* (ou A star)
C'est un algorithme de pathfinding très connu qui ressemble à 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)
```

Nous avons ensuite du évidemment adapter l'algorithme à notre projet, pour se faire nous avons d'abord du créer une classe qui à une case associe un 'coût' (le nombre de coups jusqu'ici), une heuristique (la distance "à vol d'oiseau" jusqu'à l'arrivée ce qui permet de minorer la distance réelle à parcourir) 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 nous avons fait 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 portées doit être presque linéaire.

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

Premier Passage

On produit un premier point de manière aléatoire P_0 = (x, y), et à partir de ce point on génère n-1 autres points (P_0,...,P_n).

$$P_k = P_{k-1} + B_k(a, b)$$

Telle que $k \leq$ n, $k > 0$, le vecteur (a , b) $\in$ V valide (V l'ensemble des 8 différentes directions valides représenté avec des vecteurs).

$$\text{V} = \{(-1,-1),(-1,0),...,(1,1)\}$$

Un vecteur (a , b) est valide uniquement si la distance entre P_0 et P_{k-1} est inférieur a la distance entre P_0 et P_k.
B est un tableau d'entier arbitrairement sélectionnée, par exemple :

$$\text{B} = \{8,8,7,6,5,4,3,2,2\}$$

dans ce cas B est un tableau de 9 entier avec les extrémités doubles, pour augmente la vitesse du jeux au début et éviter le nombre 0 a la fin, a cause du deuxième passage.

$$B_n = B_{n-1} \text{ et } B_0 = B_1.$$

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.

On définit un chemin valide, le chemin connexe et jouable dans 15 ou moins mouvement.
Avec cette phrase on peut 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();
    // renvoie un nombre entre 0-7 aléatoirement, qui
       représente les 8 différentes directions
    v ← nouvellePosition(v,dir);
    // renvoie un vecteur qui représente la nouvelle position
       v_{n+1} par rapport à 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'entiers, que l'on peut représenter avec un graphe :
GénérationTerrain(n,m,5);

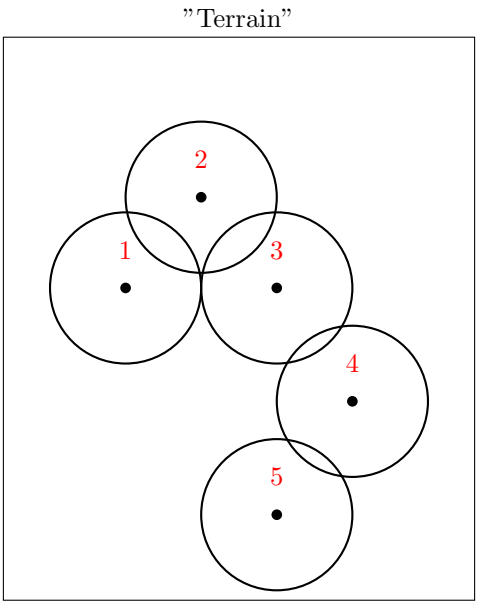

Figure 14: les cercles représentent la zone ou le Terrain va se générer.

Deuxième Passage

Le chemin ainsi généré, on va passer case par case dans notre grille et on va tester si la distance entre la case et un point du chemin est inférieure à un certain nombre. On pose (x, y) une case, T l'ensemble des points du chemin, λ le rayon.

$$\text{Si distance}((x, y), T_i) < \lambda \text{ alors la case est valide.}$$

Si une case est valide ça veut dire que cette case fait partie de la surface de notre terrain, donc on doit lui donner une portée, cette portée est définie par la formule suivante:

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

Cette formule est divisée en 2 parties, représentées par les couleurs:

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

dist est la distance entre le dernier point du chemin et la case actuelle.
srayon est la distance entre le dernier point du chemin et le point valide le plus

éloigné du dernier point du chemin.
A l'aide de GeoGebra on peut représenter cette formule de manière plus graphique.

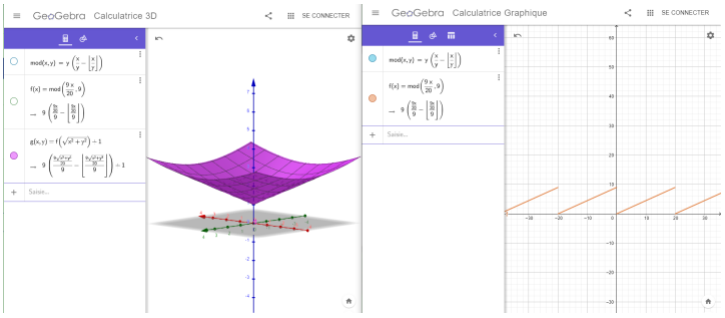

Figure 15: srayon = 20, 9 au lieu 8 et 1 au lieu 2. La portée d'un point est représentée sur l'axe z.

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

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

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

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);
// renvoie le point le plus éloigné 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 renvoie vrai ou faux dépendamment 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 renvoie la portée d'un point
               utilisant la formule vue avant.
return Tab;
```

"Terrain"

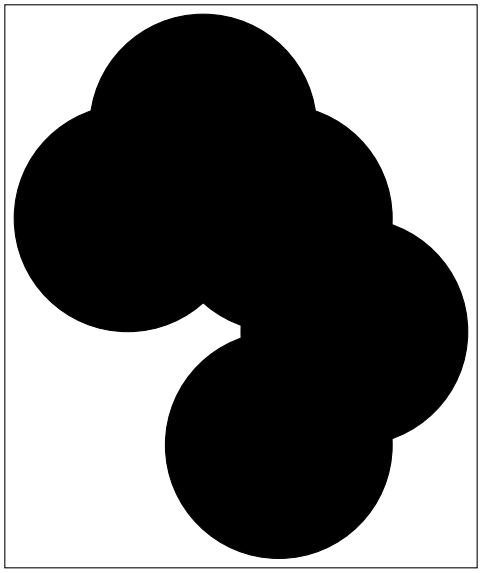

Figure 16: les zones noires représentent la surface jouable.

3 Résultat

3.1 Le jeu

Le jeu est terminé, il est possible de jouer dans 3 mode, tout seul pour s'entraîner, avec d'autre personne jusqu'à 4 joueurs, ou alors avec une IA.
Le joueur humain choisit son niveau entre 18 niveaux qui forment un parcours sans monter en difficulté, et un bouton de génération de niveau aléatoire.

Le jeu se déroule en cliquant sur le terrain, comme montrer sur la figure 1, chaque joueur à une couleur définie. Rouge pour le joueur 1, bleu pour le joueur 2, jaune pour le joueur 3 et enfin vert pour le joueur 4.

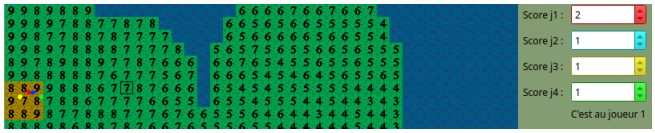

Figure 17: Les différents joueurs

Quand on ajoute une IA, on ne peut jouer que jusqu'à trois joueurs et l'IA prend la place du dernier joueur.

3.2 CONCLUSION

Pour ce qui est de la programmation, nous avons pu mettre en œuvre dans un projet concret ce que l'on a appris en cours. Nous avons amélioré nos compétences techniques personnelles mais aussi notre capacités à s'adapter dans un projet de groupe.
Nous avons appris à gérer un projet en équipe avec les atouts de chacun, et aussi apprendre à combler les lacunes que chacun pouvaient avoir.

Cependant, en ce qui concerne le projet, nous pouvons imaginer comme amélioration de rajouter plusieurs niveaux d'IA, mettre une difficulté croissante pour les niveaux, on pourrait aussi rajouter de la diversité dans les obstacles du jeu comme des bancs de sable ou encore des arbres. En ce qui concerne l'interface, on pourrait y ajouter des "animations" pour les mouvements de la balles. Le dernier ajout pour rendre le jeu plus optimal serait d'ajouter une partie réseaux.

4 Bibliographie

En general:

- www.cplusplus.com/reference

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