

Camp de jour Folie Technique

Été 2019

Informatique Avancé

Thématique 14-17 ans

POLYTECHNIQUE MONTRÉAL
Par Charly Jeffrey

Table des matières

Introduction	2
Horaire de la semaine	2
Conseils	2
Présentation de la semaine	3
Introduction à C++	4
Mandelbrot	6
Connect 4	12
Annexe	15

Introduction

La semaine *Informatique Avancé* introduira les jeunes au langage de programmation *C++* et leur montrera comment manipuler des variables, fonctions, classes, pointeurs et autres pour créer des applications interactives. Les deux principales activités sont *Connect 4* et *Mandelbrot Fractal*.

La semaine s'adresse à des jeunes ayant, idéalement, déjà programmé dans le passé et cela peu importe le langage utilisé. Un jeune qui a déjà fait une des semaines *Informatique* du camp est parfaitement en mesure de suivre la semaine *Informatique Avancé*.

Horaire de la semaine

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
9:00	Spectacle du lundi				
9:30					
10:00		Présentation: Images et Fractales			
10:30	Introduction Semaine				
11:00					
11:30	Diner	Diner	Diner	Diner	Diner
12:00					
12:30	Activité à l'extérieur	Activité à l'extérieur	Activité à l'extérieur	Activité à l'extérieur	Activité à l'extérieur
13:00					
13:30					
14:00					
14:30	Introduction à C++	Mandelbrot	Connect4	Visite Unity	Finalisation
15:00					
15:30					
16:00					

Conseils

Lien vers le [github](#) de la semaine.

Kermesse: Habituellement, les jeunes de 14-17 n'aiment pas vraiment aller à la Kermesse; demandez leur s'ils préfèrent aller à la Kermesse ou rester au local pour finaliser leurs projets ou avoir période libre si les projets sont complétés.

Présentation de la semaine

Sommaire

Tel que mentionné plus haut, la semaine *Informatique Avancé* introduira les jeunes à la programmation orientée objet, ou *OOP*, à l'aide du langage *C++*. Les activités qu'ils feront durant la semaine serviront à les familiariser avec certaines notions de bases de la programmation, telles que les variables, les fonctions, etc, ainsi que de voir certaines applications pratiques de la programmation telles que l'apprentissage machine, traitement d'image et autres.

Commencez par vous présenter et faire un tour de table pour déterminer l'intérêt que les jeunes pour la programmation ainsi que leur compétences en informatique; normalement la semaine s'adresse à des jeunes qui ont déjà programmé dans le passé, mais ils y en a toujours qui toucheront à la programmation pour la première fois. Si certains des jeunes n'ont jamais programmé antérieurement, vous allez devoir ajuster le débit et certaines présentations en conséquence. Rassurez toutefois ces jeunes pour qu'ils ne se sentent pas à l'écart des autres et pour qu'ils sachent que la semaine est quand même réalisable pour leur niveau.

Par la suite, à l'aide de la présentation *Introduction à la semaine*, présentez-leur le contenu de la semaine et les activités qu'ils feront; il s'agit d'une briève explication qui doit susciter leur intérêt et les motiver.

Durée approximative 1h	Coût approximatif par jeune 0.00\$
Niveau de difficulté (1 à 5) 1	Groupe d'âge 14-17 ans
Matériel - Aucun	Notions abordées - Aucune

Présentation suggérée

Uniquement le *powerpoint Introduction à la semaine* sera utile et nécessaire pour cette activité.

Introduction à C₊₊

Sommaire

Cette activité/présentation a pour but d'introduire les jeunes aux concepts fondamentales de la programmation et du langage *C₊₊*; c'est également à l'aide de cette activité qu'ils se familiariseront avec le logiciel *VisualStudio*.

<u>Durée approximative</u> 3h	<u>Coût approxatif par jeune</u> 0.00\$
<u>Niveau de difficulté (1 à 5)</u> 3	<u>Groupe d'âge</u> 14-17ans
<u>Matériel</u> - Aucun	<u>Notions abordées</u> <ul style="list-style-type: none"> - VisualStudio - Variable - Fonction - Boucle - Tableau - Pointeur

Théorie

Cette activité a pour but d'introduire les notions de bases du langage *C₊₊* et s'assurer que les jeunes deviennent habiles à les manipuler. Les notions qui seront abordées sont les types, les variables, les fonctions, les *arrays*, les pointeurs ainsi que d'autre notions utiles tel que les expressions logiques. Certaines notions seront trivialement simples pour certains jeunes, mais complètement nouvelles pour d'autres c'est pourquoi il est important de s'assurer que tous les jeunes les comprennent bien.

Les types:

Un type en programmation sert à définir la nature d'une variable, et même d'une fonction, et ainsi déterminer quelles opérations peuvent être appliquées sur la variable; le type d'une variable est donc déterminer selon le rôle que cette variable a dans le programme. Les types de base du langage *C₊₊* sont:

- | | |
|-----------|--|
| bool(ean) | Un seul bit, deux valeurs possibles: 0 ≡ <i>false</i> ou 1 ≡ <i>true</i> . |
| int | Représente les nombres entiers entre -2^{32} et $2^{32} - 1$. |
| float | Représente les nombres décimaux, aussi appelés nombre flottants. |
| char | Représente un caractère seul. |
| void | Type réservé aux fonctions qui retournent aucune valeur. |

Il existe également les types *long*, *long long* et *double*, mais ceux-ci sont uniquement des extensions des types *int* et *float* pour encoder des nombres entiers plus grands ou encoder des nombres décimaux avec une précision plus élevée.

Les variables:

Une variable peut être vue comme une petite boîte que le programmeur crée en lui donnant un nom, un type et possiblement une valeur. Cette petite boîte peut être accédée n'importe quand (ou presque) dans le programme pour soit utiliser sa valeur, ou la modifier. Il est important de savoir qu'en *C++*, contrairement à un langage tel que *Python*, le type d'une variable ne peut pas être modifié!

Les fonctions:

Les fonctions servent à encapsuler une procédure qui peut être appelé par la suite à l'intérieur du programme principale. Puisque *C++* est un langage typé, alors les fonctions doivent également avoir un type; ce type est le même que celui de la valeur qui sera retournée par la fonction. Exemple, si la fonction retourne un nombre entier, alors son type doit forcément être *int*, mais si une fonction ne retourne rien, alors son type est *void*.

Les arrays:

Les arrays servent à ordonner plusieurs éléments de même type et d'accéder à ces objets à l'aide d'une seule variable. Ces éléments sont sauvegardés sur la mémoire de manière consécutive, ils sont donc ordonnés les uns à la suite des autres. En *C++*, la taille d'un array doit être connue à la compilation et ne peut pas être modifiée par la suite. Par contre, ses éléments, eux, peuvent être accédés et modifiés à n'importe quel moment. Le type d'un array est tout simplement le même que le type de ses éléments; un array qui contient des nombres entiers est donc de type *int*.

Les pointeurs:

Tous les objets d'un programme doivent être sauvegardés quelque part sur l'espace mémoire allouée pour le programme. Les pointeurs servent à accéder à l'emplacement où un certain objet est sauvegardé sur la mémoire; ainsi, ils «pointent» à quel emplacement une variable, une fonction, etc se trouve et donc, peuvent être utilisés pour directement modifier la valeur qui se trouve à cet emplacement. Le type d'un pointeur est simplement le même que celui de l'objet qu'il pointe. Ils sont utilisé, entre autre, lorsqu'une fonction doit modifier la valeur de plusieurs variables simultanément, ou lorsqu'une fonction fait appelle à une classe ou autres structures plus complexes que les types de base. Un array n'est en fait qu'un pointeur qui pointe vers l'emplacement mémoire où tous ses éléments sont sauvegardés; accéder au i^e élément du array ne fait que déplacer le pointeur initial de i position sur la mémoire.

Manipulation

Les manipulations pour cette activité sont assez simples: suivez simplement les diapos de la présentation suggérée; la théorie sur les notions et des exercices récapitulatifs s'y trouvent. Tel que mentionné plus haut, il faut simplement s'assurer que les jeunes comprennent bien la matière, sans toutefois la maîtriser. Cette activité doit les rendre plus habiles à travailler et programmer avec *C++* pour que les deux projets se réalisent plus fluidement.

Présentation suggérée

Seul le powerpoint *Introduction à C++* est nécessaire.

Mandelbrot

Sommaire

Le premier projet consiste à recréer le fractale de Mandelbrot semblable à celui de la figure 1. Cette activité montrera aux jeunes comment les images et les couleurs sont encodées dans un ordinateur, tout en mettant en pratique les notions abordées plus tôt.

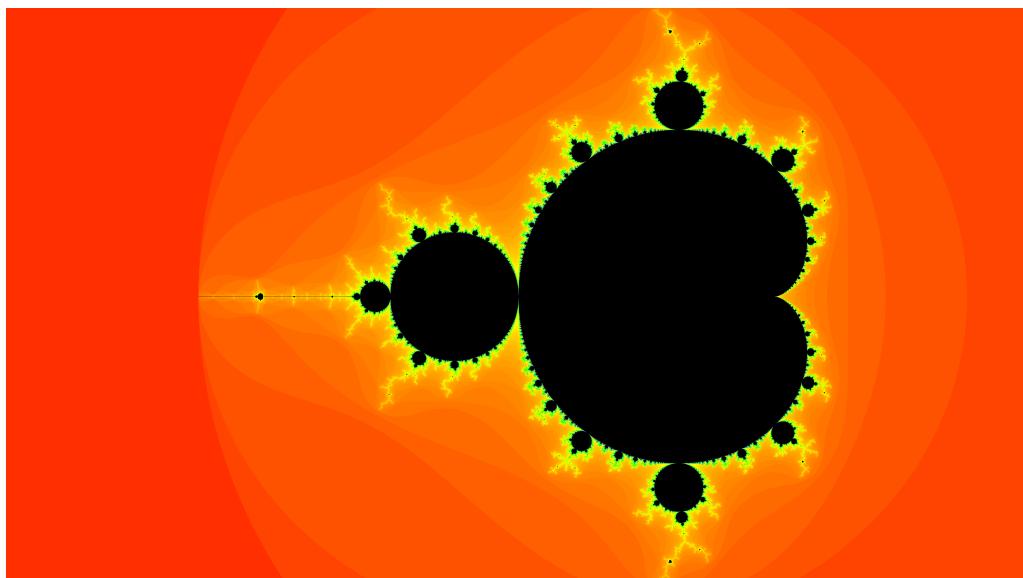


Figure 1: Fractale de Mandelbrot

<u>Durée approximative</u> 5h	<u>Coût approximatif par jeune</u> 0.00\$
<u>Niveau de difficulté (1 à 5)</u> 3	<u>Groupe d'âge</u> 14-17 ans
<u>Matériel</u> - Aucun	<u>Notions abordées</u> - Fractale - Nombres complexes - Image/Couleurs Encoding

Théorie

Cette activité n'introduit pas beaucoup de nouvelles notions de programmation, mais utilise certains concepts mathématiques que les jeunes n'ont probablement pas vu. Certaines parties théoriques parmi celles qui suivent vont plus loin que le nécessaire pour réaliser le projet, mais il est important de brièvement en parler avec les jeunes pour qu'ils puissent comprendre d'où l'image qu'ils feront venir.

Fractale:

Un fractale est un objet mathématique dont la structure est invariante à un changement d'échelle; donc peu importe l'échelle utilisée la même forme est retrouvée. Exemple:

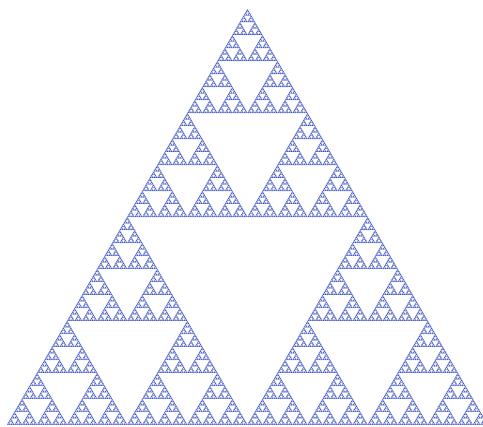


Figure 2: Triangle de Sierpiński

Nombres complexes:

Puisque le fractale de Mandelbrot est obtenu à l'aide d'une équation faisant intervenir les nombres complexes, il serait alors intéressant d'en parler avec les jeunes. La théorie ci-dessous n'est pas obligatoire pour réaliser l'activité et est donc optionnelle; donnez en tant que les jeunes semblent intéressés et n'en donner pas trop pour ne pas les perdre!

Un nombre complexe est un nombre à deux composantes: une partie réelle ainsi qu'une partie imaginaire; il s'agit donc d'un vecteur dans le plan complexe. Soit z , un nombre complexe quelconque,

$$z = a + ib$$

la partie réelle de z vaut a et la sa partie imaginaire vaut b . Le nombre i correspond au nombre *imaginaire* dont la valeur est $\sqrt{-1}$; la partie imaginaire d'un nombre correspond toujours à la composante qui multiplie le nombre i . La norme d'un nombre complexe, ou sa grandeur, est obtenue à l'aide de la relation de Pythagore:

$$|z| = \sqrt{a^2 + b^2}$$

Soit z_1 et z_2 , deux nombres complexes, alors l'addition et le produit de z_1 avec z_2 valent

$$\begin{aligned} z_1 + z_2 &= a_1 + a_2 + i(b_1 + b_2), \\ z_1 \cdot z_2 &= a_1 a_2 - b_1 b_2 + i(a_1 b_2 + a_2 b_1). \end{aligned}$$

L'addition de deux nombres complexes revient donc à additionner la partie réelle du premier avec celle du deuxième et de même pour les parties imaginaires. La multiplication, elle, est un peu plus compliquée puisqu'il faut distribuer les termes -incluant le nombre i - et regrouper les parties réelles et imaginaires.

Mandelbrot fractale:

Pour obtenir le fractale de Mandelbrot, il faut utiliser l'algorithme suivant:

```

Mandelbrot(iterMax, D) :
Pour tout nombre C de D :
    iter ← 0;
    z ← C;
    Tant que iter < iterMax :
        z ← z · z + C;
        Si |z| >= 2 :
            Colorier le pixel associé à C blanc;
            fin;
        iter ← iter + 1;
        Colorier le pixel associé à C en noir;
    
```

Autrement dit, si on itère sur le domaine D suivant:

$$D = \{z \in \mathbb{C} \mid \operatorname{Re}(z) \in [-2, 2], \operatorname{Im}(z) \in [-2, 2]\}$$

alors le nombre C va prendre toutes les valeurs possibles pour ce domaine et pour chaque valeurs de C , on applique la récursion suivante:

$$z_{n+1} = z_n^2 + C$$

où z_n est la valeur de z à la n^e itération et $z_0 = C$. On applique la récursion tant la norme de z est inférieure à 2. Si on sort de la boucle avant d'atteindre la valeur maximal permise pour le nombre d'itération, alors on colorie le pixel en blanc et sinon, on le colorie en noir.

Images et pixels:

Une image a comme propriétés une largeur ainsi qu'une hauteur, ce qui définit sa résolution. Une image de largeur 1080 avec une hauteur de 720 a donc une résolution de 1080x720. L'image est formée de pixels et chaque pixel est défini selon 3 propriétés: l'intensité des couleurs rouge, vert et bleu que le pixel projette; aussi nommé RGB. Ces propriétés peuvent uniquement avoir des valeurs entre 0 et 255; un pixel est donc encodé sur 3 *bytes* (ou 24 *bits*). Ainsi, une image est complètement définie si on connaît ses dimensions et l'intensité RGB de chacun de ses pixels.

Pour cette activité, les images que les jeunes feront seront encodées sous le format *.ppm* suivant:

```

P2/P3
width height
255
pixel00 pixel01 ... pixel0w
pixel10 pixel11 ... pixel1w
:
pixelh0 pixelh1 ... pixelhw

```

La première ligne correspond au type d'encodage utilisé pour les couleurs; P2 signifie que l'image est en *greyscale* et P3, pour une image en couleur. La seconde ligne définit la dimension de l'image; *width* sera la largeur et *height*, la hauteur. La troisième ligne définit la valeur qui sera utilisée pour normaliser les valeurs qui seront données plus bas; 255 est une bonne valeur à utiliser. Finalement, le reste du fichier contient la valeur de chaque pixel de l'image, ligne par ligne. Il faut noté que si l'image est de format P2, alors il y aura une seule valeur par pixel, mais si l'image est en P3, alors il y aura 3 valeurs par pixel (rouge-vert-bleu).

Couleurs:

Il existe plusieurs manières différentes pour définir une couleur. La plus populaire est en la définissant par la quantité de rouge, vert et bleu que la couleur contient; il s'agit de l'encodage *RGB*. C'est sous ce format qu'une couleur est encodée dans un pixel. Une autre manière est d'utiliser le système *HSV*: *hue*, *saturation* et *value/brightness*.

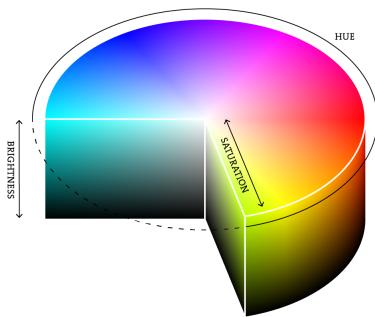


Figure 3: Encodage HSV

Le *hue* correspond à la teinte que la couleur aura; la valeur doit être comprise entre 0 et 360. La saturation, comme son nom l'indique, correspond à la saturation de la couleur; la valeur doit être comprise entre 0.0 et 1.0 tel que 1.0 correspond à blanc. Finalement, la valeur, aussi appelée *brightness*, correspond à la quantité l'intensité de la couleur; la valeur doit être entre 0.0 et 1.0 tel que 0.0 correspond à noire et 1.0 sera la couleur pure. L'avantage d'utiliser le format *HSV* est qu'il est beaucoup plus facile se «promener» dans cet espace et ainsi créer une image avec de belles teintes; on obtient d'abord le pixel sous le format *HSV*, puis on l'encode sous le format *RGB* à l'aide d'une transformation pour encoder l'image. La fonction permettant passer de *HSV* à *RGB* sera donnée.

Manipulation

Les manipulation pour cette activité sont séparées en 5 itération du code; chaque itération ajoute de nouveaux éléments au code et modifie les anciens pour améliorer le programme.

Première itération:

La première itération du code introduira les jeunes à l'allocation de mémoire dynamique via la fonction *malloc()* pour créer des *arrays* à taille variable en *C++*. Ils apprendront aussi à créer une image à partir des valeurs contenues dans un *array*. L'image finale de cette itération est donnée par la figure 4 de l'annexe.

Seconde itération:

La seconde itération du code montrera aux jeunes comment définir leur propres fonctions en *C++*. Aucun élément est ajouté à cette itération; les jeunes devront définir des fonctions pour l'allocation de mémoire, l'initialisation de *array* ainsi qu'une fonction pour enregistrer un *array* en image.

Troisième itération:

La troisième itération introduit la récursion de Mandelbrot pour obtenir une première version du fractale. Les jeunes définiront la fonction qui permettra d'obtenir la couleur d'un pixel selon sa position dans le domaine complexe de l'image. L'image finale obtenue en noir et blanc est donnée par la figure 5 de l'annexe.

Quatrième itération:

La quatrième itération introduira les notions de couleurs sous les format *HSV* et *RGB* ainsi que les structures du langage *C++*. Les jeunes devront donc modifier leur code pour ajouter les structures *pixelHSV* et *pixelRGB*, et devront également modifier comment le *array* utilisé est défini pour permettre d'associer 3 valeurs à chaque pixel de l'image; les valeurs seront utilisés pour stocker la couleur sous le format *RGB*. L'image obtenue à la fin de cette itération ne devrait pas être différente de celle obtenue à l'itération précédente, malgré les changements fait au code.

Cinquième itération:

La cinquième, et dernière, itération ajoutera de la couleur au fractale. Les jeunes utiliseront les structures *HSV* et *RGB* précédemment définis pour créer le fractale de Mandelbrot sous toute sa splendeur. Cette étape est celle qui permettra aux jeunes d'être créatifs puisqu'ils pourront modifier comment la couleur de chaque pixel sera définis tout en se «promenant» à différent endroits du fractal. L'image de base obtenue à la fin de cette itération est donnée par la figure 6; les autres images de l'annexe sont obtenues selon les coordonnées utilisées pour centrer l'image, le nombre d'itérations maximale , le zoom fait ainsi que les couleurs utilisées.

Exploration du fractale:

Si le temps le permet, laissez les jeunes explorer le fractale en modifiant le point central, défini par *center_x*, *center_y* dans le code, tout en modifiant comment la couleur varie selon l'itération. Pour ce faire, les jeunes peuvent modifier la fonction *ScalingFunction* du code comme bon leur semble, mais cette fonction doit absolument retourner une valeur entre 0.0 et 1.0, inclusivement. Modifier le nombre d'itération maximal ne fait que mieux définir les contours des structures du fractale; plus d'itération permet de mieux définir les structures à plus petites échelles.

Présentation suggérée

Seule la présentation *Le fractale de Mandelbrot* est nécessaire à l'activité.

Sources d'erreur

La première source d'erreur pour cette activité sera probablement une mauvaise indexation sur les *arrays* du programme; un pointeur qui pointe nul part. Cette erreur survient lorsque l'indice donné à un *array* ne se trouve pas dans l'intervalle valide, soit entre 0 et la taille du *array*. Pour corriger ces erreurs, il suffit de déterminer pourquoi l'indice fourni est hors domaine; par exemple, la condition d'arrêt d'une boucle n'est pas cohérente avec le *array* qu'on parcourt/modifie à l'intérieur de celle-ci.

La seconde source d'erreur pourrait être la définition de la fonction *Mandelbrot()*; puisque celle-ci se base sur les nombres complexes et que les jeunes n'auront pas vu ce qu'ils sont, il se pourrait que des erreurs de signes surviennent lorsque les jeunes transcrivent ce que vous leur montrer. Prenez simplement votre temps et assurez-vous que les jeunes ont bien réécrit le code que vous leur montrer au tableau; dans le cas où un jeune fait une petite erreur, l'image obtenue à la fin de l'exécution du programme ne sera pas la bonne.

Une autre source d'erreur pourrait se trouver dans les définitions des fonctions pour enregistrer les *arrays* en image *greyscale* et *RGB*; vérifiez que l'en-tête et le format des images est bien écrit. Exemple, l'en-tête P2 doit être utilisé avec les images *greyscale* tandis que P3 est pour les images *RGB*.

Connect 4

Sommaire

L'activité *Connect 4* introduira les jeunes aux notions de base du langage *C++*. Durant cette activité, ils apprendront comment manipuler les variables, les fonctions ainsi que les opérations de bases touchant les classes et les pointeurs. Le but de cette activité est de programmer une version *simplifiée* du fameux jeu, *Connect 4*.

Durée approximative 5h	Coût approximatif par jeune 0.00\$
Niveau de difficulté (1 à 5) 3	Groupe d'âge 14-17 ans
Matériel Un ordinateur fonctionnel	Notions abordées <ul style="list-style-type: none"> - Variables - Arrays - Fonctions - Classes - Pointeurs

Théorie

Les classes:

Les classes sont des types personnalisés créés par le programmeur et encapsulent des variables et fonctions qui seront propres à la classe créée; les variables et fonctions définies par une classe sont nommées *attributs* et *méthodes*. Les attributs et méthodes qui peuvent être accédés et utilisés hors de la classe sont dits *publiques* et si ils sont uniquement utilisable par la classe elle-même, ils sont dits *privés*.

Manipulation

1) Explication du jeu

La première étape est de s'assurer que tous les jeunes comprennent bien les mécanismes du jeu *Connect 4*. Cela comprend comment les joueurs interagissent avec le jeu, comment vérifier si un des joueurs a gagné, etc. *Connect 4* est un jeu relativement simple, et les jeunes vont probablement penser que cette étape est inutile, mais c'est bien important que tout soit clair avant de débuter la programmation du jeu.

2) Créer le projet

La seconde étape est de créer le projet qui contiendra trois fichiers: *Source.cpp*, *Game.cpp* et *Game.h*. Le premier fichier sera celui qui s'occupera des interactions entre les joueurs et la classe de jeu; c'est ce fichier qui contiendra la fonction *main*. Les deux autres fichiers sont respectivement la définition des méthodes de

la classe *Game* et son en-tête; c'est à l'aide de ces fichiers que les mécanismes du jeu seront définis.

3) Première itération de la classe

Lorsque le projet est créé, commencez par définir les attributs ainsi que les méthodes de bases de la classe *Game*; soit ses dimensions, la grille de jeu ainsi que les symboles possibles pour les attributs et le constructeur, *reset()* ainsi que *printGrid()* pour ses méthodes. La première itération de chaque fichier devrait ressembler à celle contenue dans le dossier *v1* de l'activité.

4) Seconde itération

Game.h:

Dans le fichier en-tête de la classe, l'enum *EChoiceStatus* devra être ajoutée; elle contient tous les status possibles qu'un choix d'un joueur fait. Les définitions *IsColumnFull()*, *GetChoiceStatus()*, *PlaceJeton()* et *ChangeJoueur()* seront également ajoutées à la classe. Les attributs *colsCounter*, *quiJoue* et *joueurs* doivent également être ajoutés comme attributs privés à la classe. Les définitions des nouvelles méthodes se retrouvent dans le répertoire *v2*.

Source.cpp:

Dans le fichier *Source.cpp*, la fonction *GetValidChoice()* doit être ajoutée. De plus, la fonction *main()* doit être modifiée pour pouvoir obtenir le nom des joueurs, instancier un objet *Game* et avoir la première version de la boucle de jeu principale.

5) Troisième itération

Game.h:

Les méthodes *IsGameOver()* et *CheckGrid()* seront ajoutées à cette itération. La première méthode permet de vérifier si une partie est terminée (soit un joueur à gagné ou la partie est nulle) et la seconde méthode permet de vérifier si le dernier coup joué est un coup gagnant; leur définition se trouve dans le répertoire *v3*. Les attributs *gameOver*, *gameWon*, *valeurSymboles* et *valeurGagnante* doivent également être ajoutées à la classe.

Source.cpp:

Uniquement la boucle de jeu principale du fichier *Source.cpp* sera modifiée lors de cette itération; les méthodes *CheckGrid()*, *ChangeJoueur()* et *IsGameOver()* seront utilisées.

6) Dernière itération

La dernière itération est celle qui contient le plus de modifications et il faudra s'Adapter au rythme des jeunes.

Game.h:

Les méthodes *GetCurrentPlayer()*, *Play()*, *PlayGame()* et *PrintGameStatus()* seront ajoutées à la classe. La méthode *PlayGame()* contient maintenant la boucle de jeu principale qui se trouvait auparavant dans la fonction *main* du fichier *Source.cpp*. Le constructeur de la classe sera également modifier pour permettre aux joueurs de préciser le nombre de partie qui seront jouées par la méthode *Play()*. La méthode *PrintGameStatus()* ne fait qu'afficher le gagnant d'une partie ainsi que le nombre de victoire de chacun des joueurs. Les attributs *nombreParties* ainsi que *score* doivent être ajoutés à la classe.

Source.cpp:

Le fichier *Source.cpp* sera grandement simplifier puisque la boucle de jeu et la fonction *GetValidChoice()* se trouvent maintenant dans la classe *Game*; la fonction *GetNames()* sera toutefois ajoutée pour obtenir le nom des joueurs.

Les modifications se trouvent dans le répertoire *v4*.

Présentation suggérée

Les présentations recommandées pour cette activité sont «*Introduction à C++*» et «*Introduction aux Classes*». Pour la première partie de l'activité, il faut avoir présenté *Introduction à C++* qui, sans surprise, introduira aux jeunes les notions de bases du langage *C++*. Pour la seconde partie de l'activité, soit celle qui introduit la classe *Game*, il faut avoir fait la présentation *Introduction aux Classes* et idéalement avoir fait l'activité inclue à la fin.

Sources d'erreur

La principale source d'erreur, qui sera l'erreur la plus rencontrée durant toute la semaine, sera l'oubli des point-virgules «;» à la fin de chaque *statement*. Vous n'allez jamais assez répéter aux jeunes de vérifier si toutes les lignes de leur code finissent par un «;».

De plus, puisque les jeunes travailleront avec des pointeurs et des *arrays*, une erreur qui risque de survenir assez souvent sera l'erreur *Segmentation fault*; cette erreur se produit lorsque le programme essaie d'accéder à un élément d'un *array* avec un indice invalide. Il faut donc vérifier si l'indice utilisé est dans l'intervalle valide du *array*.

Finalement, la méthode *CheckGrid()* peut être assez complexe à première vue; assurez-vous donc que les jeunes comprennent bien comment vérifier si un joueur a gagné suite à son dernier coup joué.

Annexe

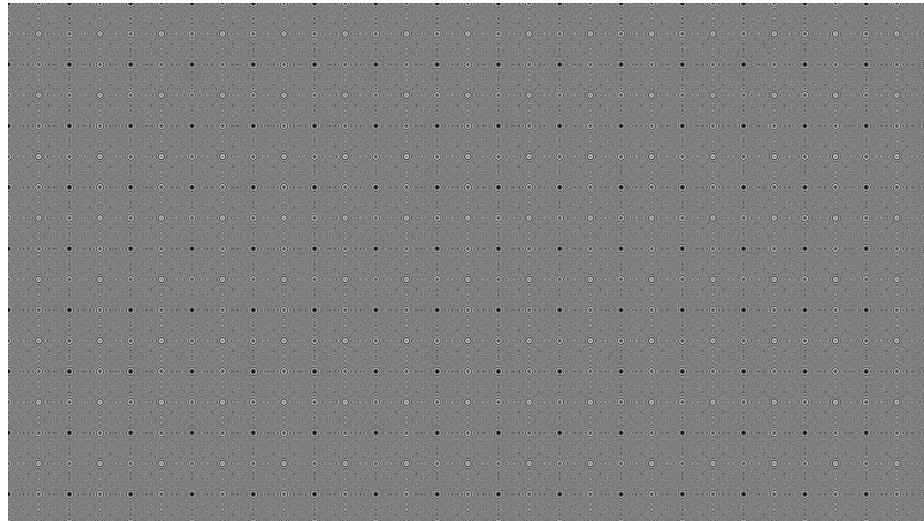


Figure 4: Première itération

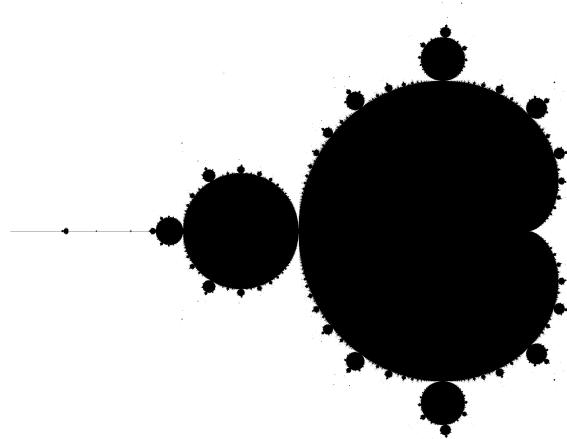


Figure 5: Troisième itération

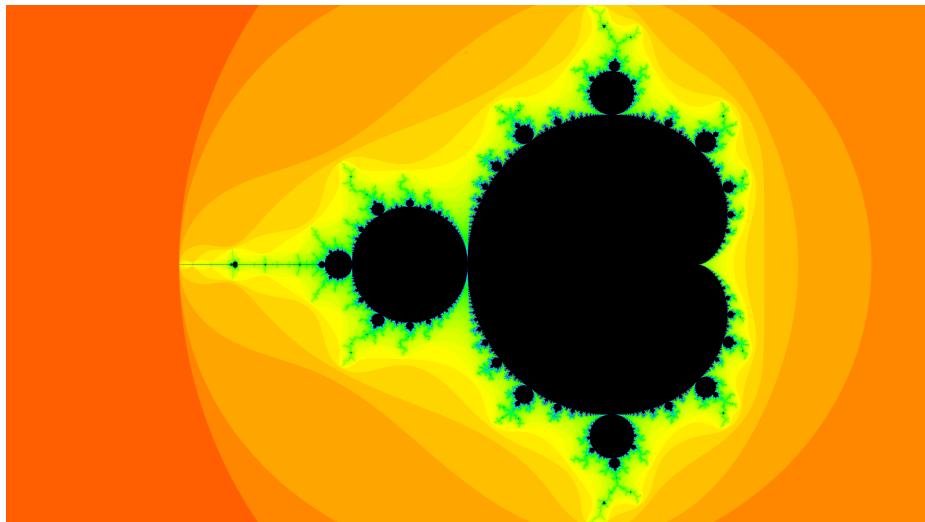


Figure 6: Cinquième itération: colorized

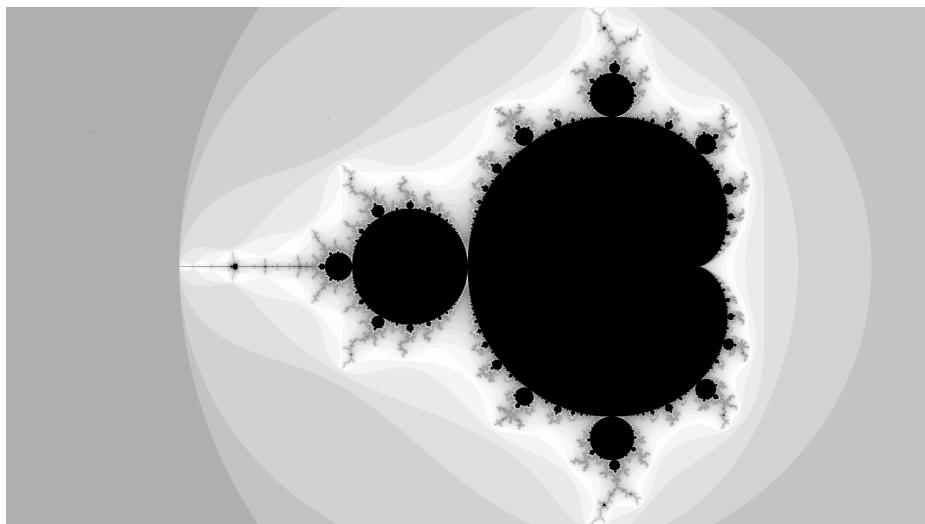


Figure 7: Cinquième itération: greyscale

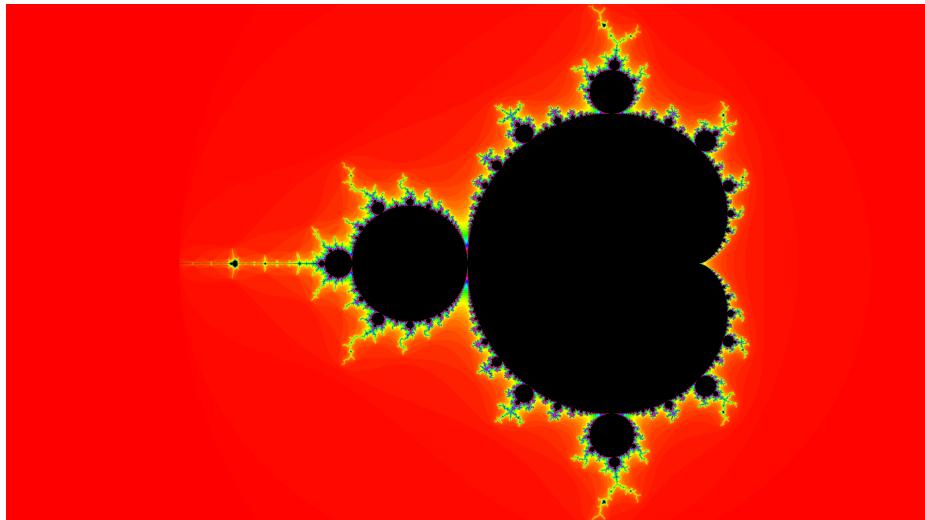


Figure 8: $(-0.75, 0.0)$; zoom=1.0

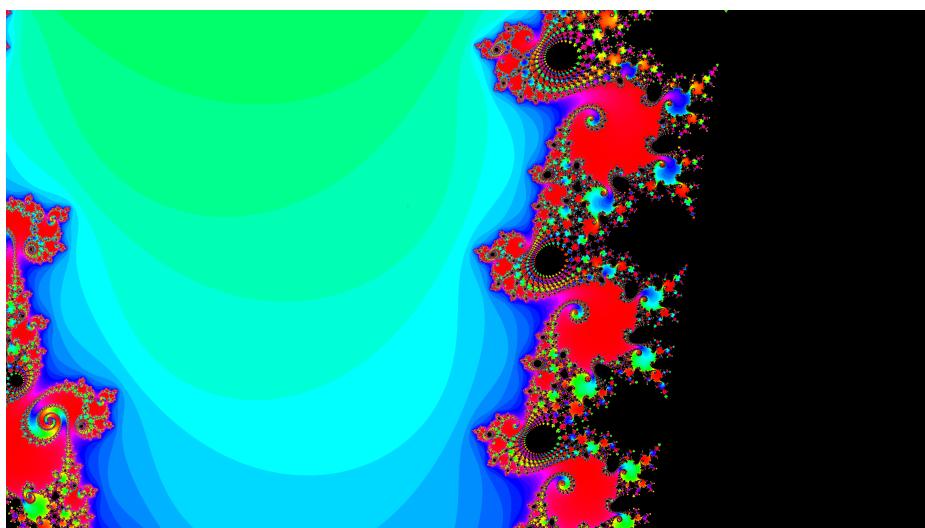


Figure 9: $(-0.75, 0.1)$; zoom=64.0

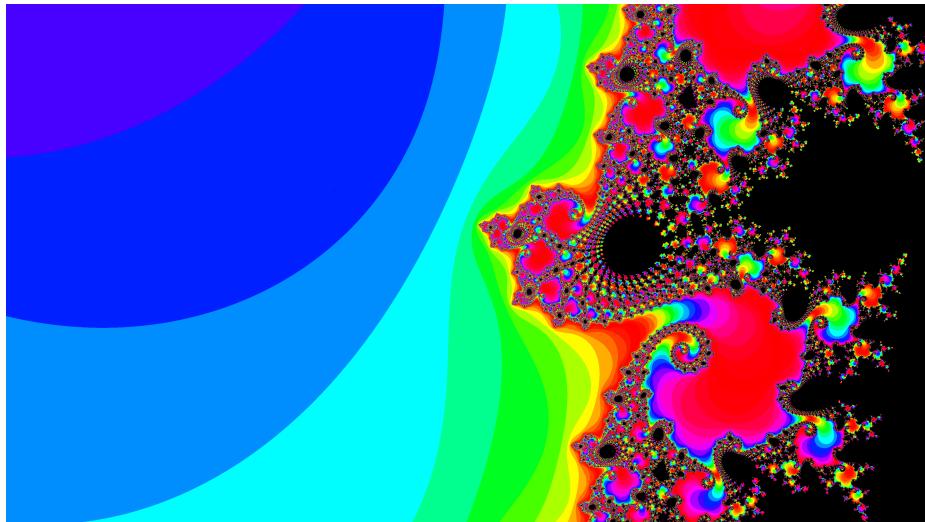


Figure 10: $(-0.75, 0.0)$; zoom=128.0

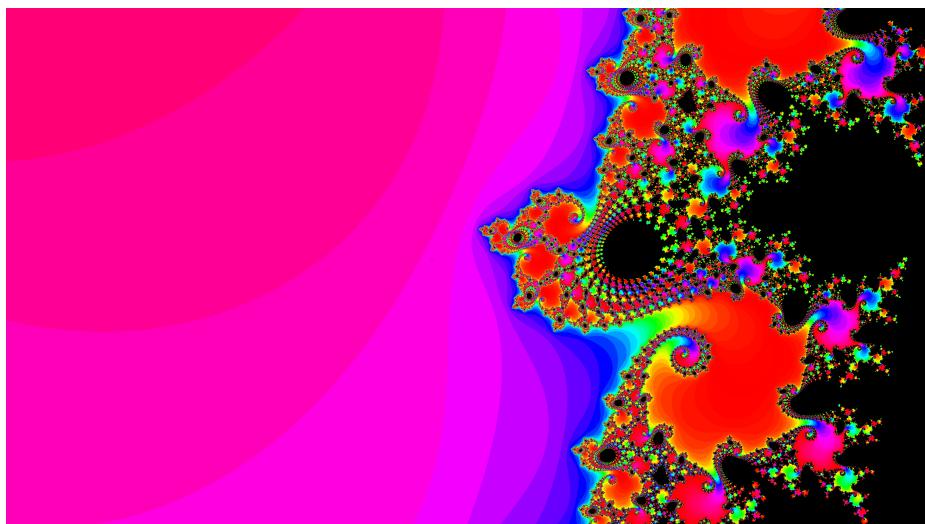


Figure 11: $(-0.75, 0.1)$; zoom=128.0



Figure 12: (-0.748, 0.1); zoom=1024.0; MAXITER=256



Figure 13: (-0.748, 0.1); zoom=1024.0; MAXITER=1024