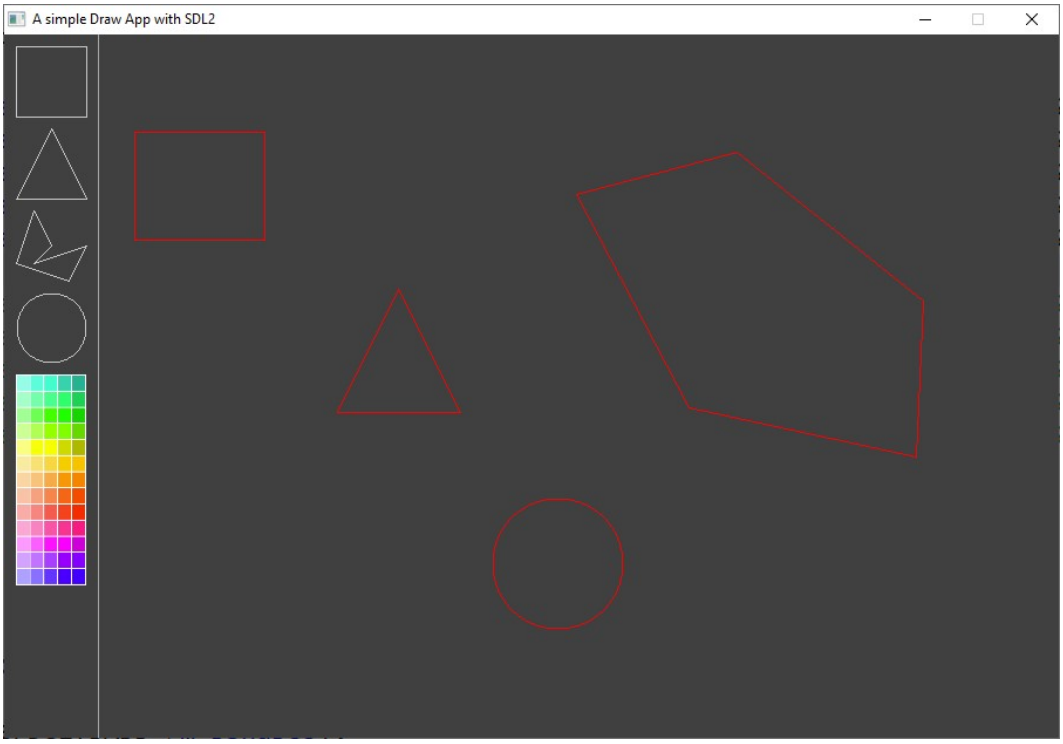
CONCEPTION LOGICIELLE

SUPPORT LANGAGE C

« MinDRAW»



1 CONTEXTE

1.1 Objectif – Description

Il s'agit de concevoir une application minimaliste de dessin de formes géométriques.

A partir d'une toolbox proposant un ensemble d'outils de traçage, formes et palette de couleurs, l'utilisateur ayant choisi la forme et la couleur, peut tracer à volonté, avec la souris, des formes à sa guise. Ces formes pourront ensuite être déplacées, modifiées ou supprimées.

L'interface présente donc deux zones, une pour la toolbox située verticalement sur la gauche, et la zone de tracé des formes occupant le reste de la fenêtre.

MinDRAW s'utilise essentiellement à la souris et quelques touches du clavier pour des opérations ponctuelles.

1.2 Représentation – Symbolisation – Modélisation

Entité « Toolbox » :

La **Toolbox** est supportée par une entité « **Container** » rassemblant les outils mis à disposition de l'utilisateur. Chaque outil est une entité de la catégorie **Tool**.

Entité « Tool » :

Chaque outil proposé dans l'interface est une entité « **Tool** » dont le descriptif est le suivant :

Structures de données :

```
typedef enum e_toolType{    /* Tool type enumerate definition        */
    TOOL_UNDEFINED  = 0,    /* undefined tool: no tool               */
    TOOL_CIRCLE     = 1,    /* for drawing circles                   */
    TOOL_TRIANGLE   = 4,    /* for drawing triangles                 */
    TOOL_SQUARE     = 5,    /* for drawing squares or rectangles     */
    TOOL_POLYGON    = 7,    /* for drawing free polygons             */
    TOOL_COLORCHART,        /* for placing a color-chart in tool-box,
                               in purpose to peek a color            */
}t_toolType;

struct s_tool {
    t_toolType      m_toolType;     /* enumerate tool type, identifying the specific tool                   */
    size_t          m_szSize;       /* tool frame size: square icon for all, except for the color-chart tool */
    SDL_Point       m_location;     /* tool location in the tool-box, window relative                       */
    SDL_Point*      m_pVextexes;    /* for tool vertexes                                                    */
    SDL_Texture*    m_pTexture;     /* only for the color-chart tool                                        */
    SDL_Color       m_colorAt;      /* only use with the color-chart tool                                   */
};
```

Fonctionnalités :

```
t_tool*ToolNew(t_toolType toolType, size_t szToolSize, int iLocX, int iLocY, ...);
```

Permet la création d'une nouvelle entité « Tool » :

toolType : une des constantes énumérées

szToolSize : la taille en de l'icône représentative de l'outil

iLocX : la position horizontale du coin haut gauche de l'icône (relative à la fenêtre)

iLocY : la position verticale du coin haut gauche de l'icône (relative à la fenêtre)

... : paramètres supplémentaires à fournir uniquement pour l'outil TOOL_COLORCHART *(expliqué plus loin pour ce cas particulier)*

Retourne le pointeur sur l'entité nouvellement créée.

```
t_tool*ToolDel(t_tool*pTool);
```

Permet la destruction d'une entité « Tool » :

pTool : pointeur sur l'entité à détruire

Retourne le pointeur NULL.

```
t_tool*ToolDraw(const t_tool*pTool, SDL_Renderer*pRenderer);
```

Permet le dessin d'une entité « Tool » :

pTool : pointeur sur l'entité à dessiner

pRenderer : pointeur sur le renderer dans lequel dessiner

Retourne le pointeur NULL pour utilisation en parsing de conteneur.

```
t_tool*ToolAt(const t_tool*pTool, SDL_Point*pAt);
```

Permet d'indiquer si une entité « Tool » est concernée par un point particulier :

pTool : pointeur sur l'entité à tester

pAt : pointeur sur le point particulier à tester

Retourne le résultat du test au format pointeur t_tool*.

```
t_toolType ToolGetType(const t_tool*pTool);
```

Permet d'obtenir le type d'outil d'une entité « Tool » :

pTool : pointeur sur l'entité à interroger

Retourne le type de l'outil, l'un des types énumérés.

```
SDL_Color ToolGetColor(const t_tool*pTool);
```

Permet d'obtenir la couleur d'une entité « Tool » :

pTool : pointeur sur l'entité à interroger

Retourne la couleur RGBA de l'outil.

Remarque : cette information n'a de sens que pour l'outil TOOL_COLORCHART. Sera développé plus loin pour ce cas particulier.

Entité « Shape » :

Cette catégorie permet de représenter les formes géométriques dessinées par l'utilisateur.

Structures de données :

```
typedef enum e_shapeType{   /* Shape type enumerate definition       */
    SHAPE_UNDEFINED  = 0,   /* undefined shape: no shape             */
    SHAPE_CIRCLE     = 1,   /* for circle shapes                     */
    SHAPE_TRIANGLE   = 4,   /* for triangle shapes                   */
    SHAPE_SQUARE     = 5,   /* for square or rectangle shapes        */
    SHAPE_POLYGON    = 7,   /* for free polygon shapes               */
}t_shapeType;

struct s_shape {
    t_shapeType     m_type;         /* enumerate shape type, identifying the specific shape  */
    SDL_Rect        m_rFrame;       /* shape rectangular frame, enclosing the entire shape   */
    SDL_Color       m_color;        /* shape drawing RGBA color                              */
    SDL_Point*      m_pVextexes;    /* shape vertexes dynamic array: use only with free
                                       polygon shapes                                        */
    int             m_iVertexes;    /* shape number of vertexes: use only with free polygons */
};
```

Fonctionnalités :

```
t_shape*ShapeNew(t_shapeType shapeType, int iLocX, int iLocY, int iWidth, int iHeight, SDL_Color color,...);
```

Permet la création d'une nouvelle entité « Shape » :

shapeType : une des constantes énumérées

iLocX : la position horizontale du coin haut gauche de la forme (relative à la fenêtre)

iLocY : la position verticale du coin haut gauche de la forme (relative à la fenêtre)

iWidth : la largeur en pixel de la forme (largeur globale de la forme)

iHeight : la hauteur en pixel de la forme (hauteur globale de la forme)

color : la couleur RGBA de la forme

... : paramètres supplémentaires à fournir uniquement pour la forme SHAPE_POLYGON *(expliqué plus loin pour ce cas particulier)*

Retourne le pointeur sur l'entité nouvellement créée.

```
t_shape*ShapeDel(t_shape*pShape);
```

Permet la destruction d'une entité « Shape » :

pShape : pointeur sur l'entité à détruire

Retourne le pointeur NULL.

```
t_shape*ShapeDraw(t_shape*pShape, SDL_Renderer*pRenderer);
```

Permet le dessin d'une entité « Shape » :

pShape : pointeur sur l'entité à dessiner

pRenderer : pointeur sur le renderer dans lequel dessiner

Retourne le pointeur NULL pour utilisation en parsing de conteneur.

```
t_shape*ShapeDrag(t_shape*pShape, int iDx, int iDy);
```

Permet le déplacement d'une entité « Shape » :

pShape : pointeur sur l'entité à déplacer

iDx : valeur du déplacement horizontal donnée en pixels

iDy : valeur du déplacement vertical donnée en pixels

Retourne le pointeur sur l'entité elle-même.

Entité « App » :

L'application est développée sur la base de la bibliothèque SDL2, une application fenêtrée selon l'architecture classique d'une programmation événementielle. Elle est supportée par les fichiers **app.h** et **app.c**.

Structure de données :

```
struct s_app {
    t_status        m_uStatus;              /* application status flags                          */
    SDL_Rect        m_rWindowFrame;         /* application main window frame                     */
    SDL_Color       m_colorBkgnd;           /* application main window background color           */
    char*           m_pTitle;               /* application main window title                     */

    SDL_Window*     m_pWindow;              /* application main window structure pointer         */
    SDL_Renderer*   m_pRenderer;            /* application main renderer structure pointer       */
    SDL_TimerID     m_iTimerID;             /* application main timer ID                         */

    SDL_Rect        m_rDrawingArea;         /* application drawing frame area                    */
    SDL_Rect        m_rToolboxArea;         /* application tool-box frame area                   */

    t_container*    m_pToolbox;             /* application tool-box container pointer            */
    t_tool*         m_selectedTool;         /* application selected tool pointer                 */
    SDL_Color       m_selectedToolColor;    /* application selected tool color                   */

    SDL_Point       m_ptStart,              /* points structures for drawing mechanism purpose   */
                    m_ptEnd;

    SDL_Point*      m_pVertexes;            /* dynamic points array for drawing free polygons    */
    int             m_iVertexesNumber;      /* current or total number of polygon vertexes       */
    int             m_iVertexesCapacity;    /* current size of the dynamic vertexes array        */

    t_container*    m_pShapes;              /* the drawn shapes container pointer                */
};
```

Fonctionnalités : (NON COMMENTEES, NI DETAILLEES, QUE DU CLASSIQUE !)

```
t_app*AppNew(int iWidth, int iHeight, int iColorBkgnd, const char*pAppTitleStr);
t_app*AppDel(t_app*pApp);
int   AppRun(t_app*pApp);
Uint32 _AppTimerCallback(Uint32 interval, void*pParam)
t_app*_AppDraw(t_app*pApp);
```

Entité « Main » :

La fonction principale main() initie la création d'une instance dynamique de l'application, la fait vivre et procède à sa destruction une fois l'application terminée. *Là aussi, que du classique !*

2 DEVELOPPEMENT – TRAVAIL DEMANDE

Le développement est à conduire sur la base du projet Eclipse DS1-23-09-2022-DRAW-C-BASE fourni. Consulter les directives en § 3 en fin de sujet avant de commencer.

Le développement est à conduire étape par étape. Le projet final sera rendu après archivage selon les modalités données en § 3 en fin de sujet.

2.1 Implémentation complète du module « Container »

S'agissant d'un module on ne peut plus classique, aucune directive n'est donnée ici. Les seules fonctionnalités requises pour la plus grande partie de l'application sont celles présentes dans les fichiers correspondants.

Remarque : certaines parties de la suite de l'application peuvent être traitées sans la nécessité du module conteneur : les éléments pouvant être testés individuellement.

☞ Implémenter le code du module « Container ».

2.2 Construction & affichage de la toolbox :

La toolbox est complétée avec des outils relevant de l'entité « Tool ». Cette entité doit être implémentée en amont.

```
t_tool*ToolNew(t_toolType toolType, size_t szToolSize, int iLocX, int iLocY, ...);
/**
 * @todo: initializing the new pTool structure
*/
```

Ne seront initialisés avec les paramètres passés à ToolNew(), que les champs :

- m_toolType
- m_szSize
- m_location

```
/**
 * @todo: allocating the dynamic vertexes array
 */
```

Allouer dynamiquement ici le vecteur de SDL_Point nécessaire à chaque type d'outil, à savoir :

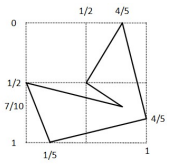
- *TOOL_SQUARE : 5 points : 4 points du carré + 1 point rendu identique au premier pour fermer la figure.*

- *TOOL_TRIANGLE : 4 points : 3 points du triangle + 1 point rendu identique au premier pour fermer la figure.*

- *TOOL_CIRCLE : 1 point pour la position du centre, le rayon étant déduit du champ m_szSize de la structure associée à l'entité.*

- *TOOL_POLYGON : 7 points : 6 points du polygone + 1 point rendu identique au premier pour fermer la figure.*

Détails pour la forme du polygone de TOOL_POLYGON :
Les ratios donnés sur la figure ci-contre permettent de positionner chaque vertex du polygone représentatif de l'outil TOOL_POLYGON, avec 6 points et le 7ième rendu identique au premier pour fermer la figure lors du tracé avec la fonction SDL_RenderDrawLines().



- *TOOL_COLORCHART : ne nécessite aucune allocation dynamique de mémoire.*

L'implémentation des codes suivant nécessite bien évidemment, d'exploiter les paramètres passés à la fonction.

```
/**
  * @todo: defining the vertexes for TOOL_SQUARE tool
  */
```

Définir ici les 4 vertex du carré représentatif de l'outil TOOL_SQUARE. Le 5ième vertex est rendu égal au premier.

```
/**
  * @todo: defining the vertexes for TOOL_TRIANGLE tool
  */
```

Définir ici les 3 vertex du triangle isocèle représentatif de l'outil TOOL_TRIANGLE. Le 4ième vertex est rendu égal au premier.

```
/**
  * @todo: defining the vertexes for TOOL_POLYGON tool
  */
```

Définir ici les 6 vertex du polygone représentatif de l'outil TOOL_POYGON. Le 7ième vertex est rendu égal au premier.

```
/**
  * @todo: defining the vertexes for TOOL_CIRCLE tool
  */
```

Le seul vertex du cercle représentatif de l'outil TOOL_CIRCLE est destiné à recevoir les coordonnées de son centre. Définir ici ce vertex.

☞ Implémenter le code de la fonction « ToolNew() ».

Pour tester l'implémentation faite précédemment, il faut remplir le conteneur « Toolbox » avec des outils. La disposition des outils dans la Toolbox respecte les contraintes suivantes :

- Les dimensions de la frame englobant chaque icône représentative d'un outil seront largeur = hauteur = APP_TOOLBOX_TOOL_SIZE
- Les icônes seront espacées de la valeur APP_TOOLBOX_TOOL_SPACING

Remarque : sur la capture d'écran représentant la toolbox, l'outil TOOL_COLORCHART occupe une hauteur valant 3 × APP_TOOLBOX_TOOL_SIZE. Cette spécificité est directement gérée par la fonction ToolDraw(). Le coefficient multiplicatif, ici valant 3, est paramétré par TOOL_COLORCHART_HEIGHT_RATIO situé dans le fichier <tool.h>. Ceci serait à prendre en considération, si on devait placer d'autres outils en dessous de celui-ci.

```
/*********************************************************************************
 * @todo:   Populating here the tool-box with the different tools by pushing new tool
 *          in the tool-box container.
 *
 * @example: ContainerPushback(pApp->m_pToolbox, ToolNew(TOOL_SQUARE, APP_TOOLBOX_TOOL_SIZE,
 *
 */
```

En codant la ligne donnée en exemple, un outil de type TOOL_SQUARE doit apparaître dans la toolbox à l'exécution du code.

Remarque : dès à ce stade, les événements de la souris sont réactifs :

- *Un clic sur l'outil TOOL_SQUARE doit faire apparaître dans la barre de titre de la fenêtre les caractéristiques de celui-ci. L'outil est ainsi sélectionné pour les futurs tracés.*
- *Un clic dans la zone de la toolbox, hors outil, permet d'effacer la sélection ; plus aucun outil n'est actif, aucun tracé ne sera exécuté. La fenêtre retrouve son titre par défaut.*
- *Lorsqu'un outil est sélectionné, dans la zone de tracé, bouton enfoncé et mouvements de la souris, montrent la délimitation de la future forme en cours de traçage. Noter également dans cette phase, le clipping de la souris à la zone de dessin.*

  Concernant le clipping du curseur de la souris, les dernières versions de la SDL intègrent la fonction SDL_SetWindowMouseRect() qui permet cette opération. Dans le fichier <app.h>, en activant la macro `#define APP_USE_SDLSETWINDOWMOUSERECT`*, la fonction est alors activée dans le code. Si compilation du code se fait alors sans erreur, la version de la SDL est correcte, sinon il faudra désactiver la macro en la remettant en commentaire.*

☞ Sur ce principe, ajouter les autres outils pour compléter la toolbox. Vérifier les réactions de l'application au vu de la remarque faite ci-dessus.

ATTENTION : pour éviter les plantages de l'application sur l'utilisation de l'outil TOOL_POLYGON à ce stade du développement, les fonctionnalités associées ont été désactivées avec des directives de compilation conditionnelles. Ces fonctionnalités seront activées par la mise en commentaire de la macro `#define APP_DONT_PROCESS_POLYGON_TOOL` *du fichier <app.h>.*
Cependant, si cet outil est sélectionné, et qu'un semblant de tracé a été commencé, on sortira du tracé que par un clic de la souris et avec la touche Left CTRL maintenu enfoncée.

2.3 Tracé de formes – Création d'éléments « Shape » :

```
t_shape*ShapeNew(t_shapeType shapeType, int iLocX, int iLocY, int iWidth, int iHeight, SDL_Color color,...);

/**
 * @todo: initializing the new pShape structure with parameters
 */
```

Ne seront initialisés avec les paramètres passés à ShapeNew(), que les champs :

- m_type
- m_rFrame
- m_color

☞ Implémenter le code de la fonction « ShapeNew() ».

Pour tester l'implémentation faite précédemment, il faut remplir le conteneur des formes géométriques avec des entités « Shape ».

```
/**
 * @todo: for this tools, process are the same: pushing a new Shape in the shapes container
 *        extract asked parameters by ShapeNew() from the application data
 *
 *      i.e.  ContainerPushback(pApp->m_pShapes, ShapeNew(...));
 */
```

Remarques :

- *Dans la chronologie des événements lors du traçage, pApp->m_ptStart contient les coordonnées du point du premier clic, et pApp->m_ptEnd celles du point lors du relâché du bouton.*
- *Les largeurs et hauteurs déduites de ces paramètres devront être positives. Cela ne sera pas toujours le cas suivant les positions relatives des points de départ et d'arrivée. Pour garantir cet impératif, on utilisera la fonction SDL_abs() qui renvoie la valeur absolue de l'expression ou de la valeur passée en paramètre.*

☞ Implémenter le remplissage du conteneur des formes tracées, et valider son opérabilité avec quelques essais sur les différents outils disponibles (hors TOOL_POLYGON non opérationnel à ce stade).

Remarque : le nuancier, outil TOOL_COLORCHART, n'est également pas opérationnel à ce stade de développement de l'application.

2.4 Déplacement des formes dessinées

Pour les formes dessinées dans l'interface, on souhaite pouvoir les déplacer à l'aide de la souris une fois le curseur au-dessus de la forme choisie et la touche **Left SHIFT** du clavier maintenue enfoncée. Cette action ne pouvant se faire que si l'application n'est pas déjà utilisée pour tracer une forme.

Algorithme :

- En mode libre (application non utilisée pour tracer une nouvelle forme).
- Si la touche **Left SHIFT** est enfoncée, et si aucune forme dessinée n'est déjà pas en déplacement, sur les événements mouvement de la souris, parcourir le conteneur des formes tracées pour trouver une forme susceptible d'être au dessous du pointeur de la souris.
- Si une forme est trouvée, et si la touche **Left SHIFT** est toujours enfoncée, appliquer à celle-ci le décalage correspondant au déplacement de la souris.
- Le déplacement s'arrête lorsque la touche **Left SHIFT** est relâchée. Il faut tracer le fait qu'il n'y a plus de forme sélectionnée pour un éventuel déplacement.

Support :

- La fonction **ShapeDrag()** doit être implémentée.
- Le module « **Shape** » doit être enrichie de la fonction prédicat dont le prototype est :
  `t_shape*ShapeIsAt(t_shape*pShape, SDL_Point*piAt);`
  retournant une valeur non nulle si le point **piAt** se situe dans la frame de la forme passée par **pShape**.
- Les événements mouvement souris (motion) disposent des champs **xrel** et **yrel** qui véhiculent les déplacements relatifs de celle-ci.
- On peut doter le module « **App** » d'un champ supplémentaire de type **t_shape*** destiné à recueillir le pointeur sur la forme survolée par la souris.
- L'état de la touche **Left SHIFT** est codé dans le flag *ST_APP_LSHIFT_PRESSED*.

☞ Implémenter la fonctionnalité de déplacement des formes dessinées.

Remarques : la « pollution » de la zone de la toolbox par les formes géométriques en déplacement peut se résoudre par le changement de l'ordre d'affichage de ces éléments dans la fonction _AppDraw() et en rajoutant un effacement préalable du fond de la toolbox.
Le clipping de la souris peut être activé dans cette phase afin d'éviter les débordements intempestifs.

☞ Optionnellement, proposer une amélioration de cette fonctionnalité en tenant compte des remarques précédentes.

2.5 Prise en charge de l'outil TOOL_POLYGON et des tracés associés

La saisie d'un polygone suit un procédé un peu différent de celui mis en place pour la saisie des autres formes. En effet, il faut enregistrer chaque point au clic de la souris et le rajouter à la structure qui contient les précédents points du polygone en cours de saisie.
Comme le nombre de sommets n'est pas connu en amont, on doit recourir à des allocations dynamiques en partant d'une structure de stockage d'une capacité fixée par défaut. Au fil de la saisie des sommets du polygone, si cette structure venait à manquer de place, il faudra procéder à une réallocation dynamique de cette structure avec une nouvelle capacité pris égale à 1,5 fois celle actuelle.
Du côté graphique, lors de la saisie, on ne peut plus tracer un rectangle englobant à la manière de ce qui se fait pour les autres formes. Ici il faut tracer au fur et à mesure les arrêtes du polygone en cours de saisie.
De même, la terminaison de la saisie d'un polygone diffère également. Puisqu'en phase de saisie le bouton de la souris est régulièrement enfoncé et relâché, l'événement du relâché doit être accompagné d'une information supplémentaire : le maintient de la touche Left CTRL enfoncé lors du relâché du bouton de la souris.

Procédure :

- Désactiver la macro `#define APP_DONT_PROCESS_POLYGON_TOOL` du fichier <**app.h**>.
- Implémenter toutes les sections concernées par la mise en commentaire de la macro précédente. Les indications sont directement en commentaire dans le code.
- Pour la création d'une entité **Shape** de type **SHAPE_POLYGON**, la fonction **ShapeNew()** attend en plus des paramètres habituels, deux paramètres « optionnels » (indiqués par la présence des **…** dans le prototype de la fonction) : **POUR SHAPE_POLYGON CES DEUX PARAMETRES SONT OBLIGATOIRES** :
  Il s'agit dans, l'ordre de passage, du **nombre de vertexes** constituant le polygone et du **pointeur sur la structure contenant l'ensemble des points** associé.

Support :

- L'entité « **App** » possède les champs suivants entrant dans le processus de saisie des polygones.

```
SDL_Point*      m_pVertexes;
int             m_iVertexesNumber;
int             m_iVertexesCapacity;
```

☞ Implémenter la fonctionnalité de dessin de polygones.

Remarques : à ce stade les polygones dessinés ne peuvent être déplacés par le mécanisme décrit pour les autres formes. La raison est lors de la création d'une entité « Shape » avec l'option SHAPE_POLYGON, le champ m_rFrame n'est pas initialisé, et le test de survol par la souris échoue donc.
On peut remédier à cela en recherchant sur l'ensemble des vertexes les coordonnées minimales en x et y, et la largeur et la hauteur maximales, et initialiser m_rFrame en conséquence. Attention les largeurs et hauteurs calculées devront êtres positives.
Attention, le déplacement d'une entité SHAPE_POLYGONE, pour qu'elle soit effective,il faut recalculer la nouvelle position de tous ses vertexes, et non pas simplement une simple mise à jour du champ m_rFrame.

☞ Implémenter le calcul et l'initialisation du m_rFrame des entités SHAPE_POLYGON. Faire un test de validation.

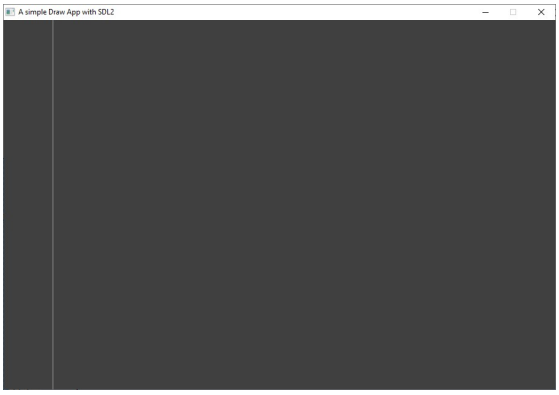
3 DOCUMENT FOURNI – DOCUMENT A RENDRE

Base de travail :

Il est fourni le fichier compressé « DS1-23-09-2022-DRAW-C-BASE.zip » ainsi que ce document au format pdf.

Sous Eclipse, importer l'archive de ce projet. Une fois le projet ouvert, le renommer aussitôt en <NomPrenom>.

Effectuer une première compilation du projet, il ne doit pas y avoir ni erreur ni warning. Lancer une première fois l'application. Vous devez obtenir la fenêtre ci-contre. Le développement de l'application peut être entamé selon les étapes précisées dans le sujet.



Les entêtes des fichiers doivent être nommés avec vos nom et prénom.

Rendu :

Le dossier du projet archivé et compressé au format **.zip** à rendre sous Moodle. Le fichier archive aura pour nom la concaténation de votre Nom et de votre Prénom, sans accent, avec les premières lettres en majuscule ; exemple : « **NomPrenom.zip** ».