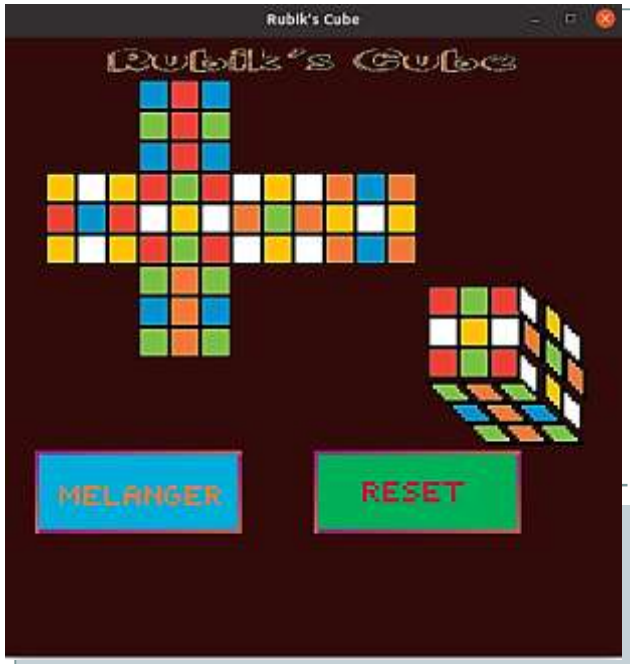
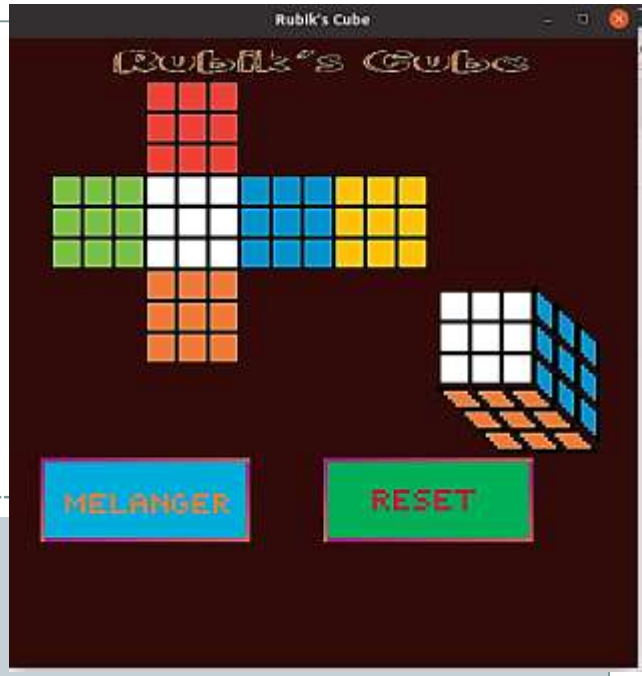
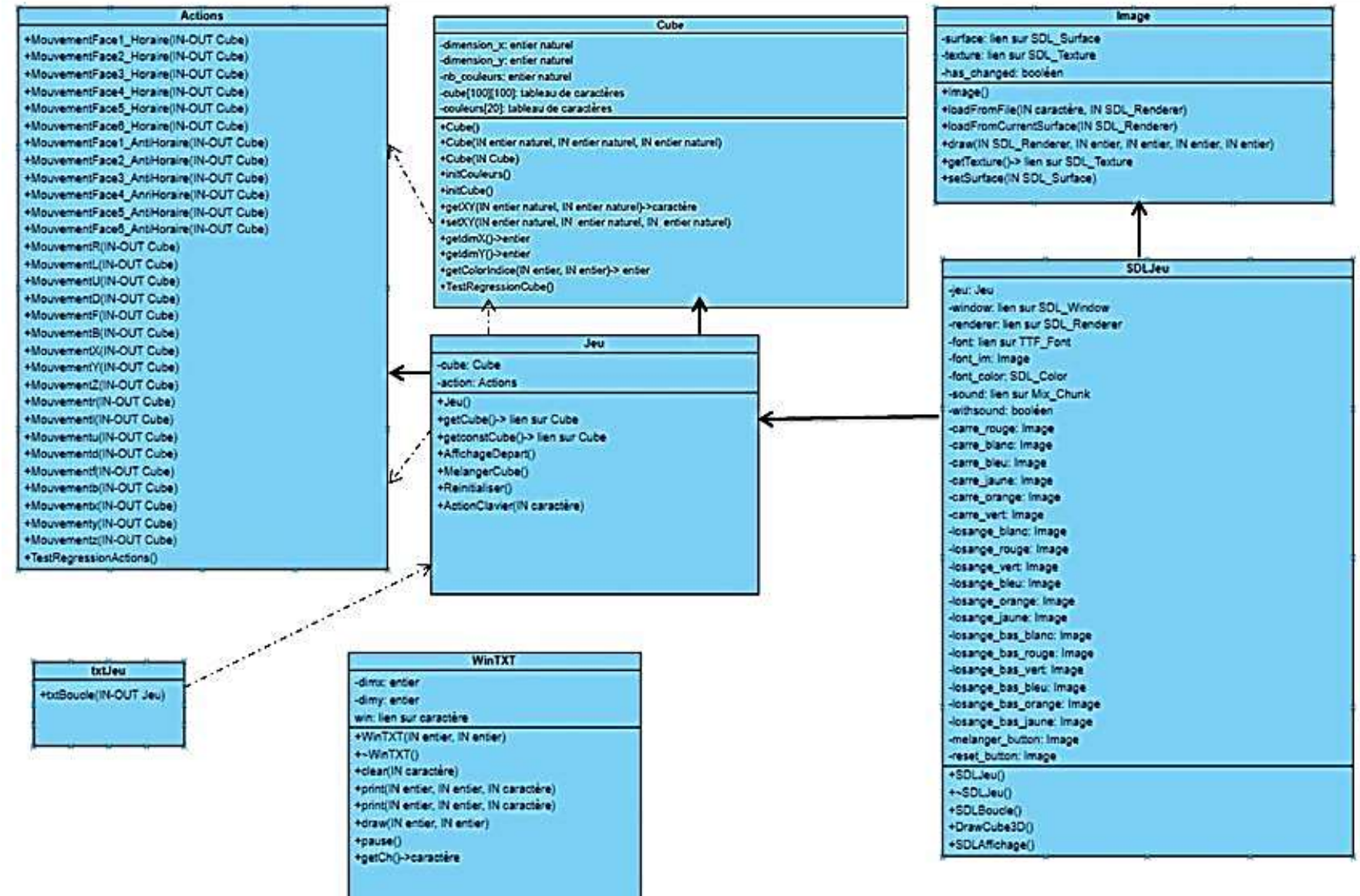



# RUBIK'S CUBE

FAIT PAR L'UNITÉ DES CASSE-TÊTEURS
MEMBRE: AHMED ATANANE P1905392
UE LIFAP4
L2 INFORMATIQUE

# Diagramme des classes



# La classe Cube

```
enum Couleur: char{Blanc='W',Bleu='B',Jaune='Y',Vert='G',Rouge='R',Orange='O',NoColor='#'};

 /** @class La classe Cube permettant de gérer un patron du Rubik's Cube en deux dimensions.
 */
class Cube
{
    private:
    unsigned int dimension_x;
    unsigned int dimension_y;
    unsigned int nb_couleurs;

    char cube[100][100];
    char couleurs[20];

    public:
    Cube();
    Cube(unsigned int dimx, unsigned int dimy, unsigned int colors);
    Cube(const Cube& c);
    void initCouleurs();
    void initCube();
    char getXY(const unsigned int x, const unsigned int y) const;
    void setXY(unsigned int x, unsigned int y, unsigned int c);
    int getdimX() const;
    int getdimY() const;
    int getColorIndice(int x, int y) const;
    void TestRegressionCube();

};
```

# La classe Actions

```
/** @class: La classe Actions permettant de gérer tous les mouvements possibles pour un Rubik's Cube.
 */
class Actions
{
    public:
    void MouvementFace1_Horaire(Cube& c); //rotation à 90º au sens horaire de la face avant.
    void MouvementFace2_Horaire(Cube& c); //rotation à 90º au sens horaire de la face de droite.
    void MouvementFace3_Horaire(Cube& c); //rotation à 90º au sens horaire de la face arrière.
    void MouvementFace4_Horaire(Cube& c); //rotation à 90º au sens horaire de la face de gauche.
    void MouvementFace5_Horaire(Cube& c);  //rotation à 90º au sens horaire de la face du haut.
    void MouvementFace6_Horaire(Cube& c);  //rotation à 90º au sens horaire de la face du bas.

    void MouvementFace1_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face avant.
    void MouvementFace2_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face de droite.
    void MouvementFace3_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face arrière.
    void MouvementFace4_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face de gauche.
    void MouvementFace5_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face du haut.
    void MouvementFace6_AntiHoraire(Cube& c); //rotation à 90º au sens anti-horaire de la face du bas.
```

# La classe Actions

```
    void MouvementR(Cube& c);
    void MouvementL(Cube& c);
    void MouvementU(Cube& c);
    void MouvementD(Cube& c);
    void MouvementF(Cube& c);
    void MouvementB(Cube& c);
    void MouvementX(Cube& c);
    void MouvementY(Cube& c);
    void MouvementZ(Cube& c);

    void Mouvementr(Cube& c);
    void Mouvementl(Cube& c);
    void Mouvementu(Cube& c);
    void Mouvementd(Cube& c);
    void Mouvementf(Cube& c);
    void Mouvementb(Cube& c);
    void Mouvementx(Cube& c);
    void Mouvementy(Cube& c);
    void Mouvementz(Cube& c);

    void TestRegressionActions();
};
```

Les différentes procédures de mouvements du cube

# La classe Jeu

```
#include "Cube.h"
#include "Actions.h"

class Cube;
class Actions;

/** @class La classe Jeu permettant de gérer les paramètres d'une partie de jeu.
 */
class Jeu
{
    private:
    Cube cube;
    Actions action;

    public:
    Jeu();

    Cube &getCube();
    const Cube &getconstCube() const;
    void AffichageDepart();
    void MelangerCube();
    void Reinitialiser();
    void ActionClavier(const char touche);
};
```

La classe Jeu gérant les paramètres du jeu Rubik's Cube

# La classe SDLJeu

```
class SDLJeu
{
    private:
    Jeu jeu;
    SDL_Window* window;
    SDL_Renderer* renderer;

    TTF_Font* font;
    Image font_im;
    SDL_Color font_color;

    Mix_Chunk* sound;
    bool withsound;

    Image carre_rouge;
    Image carre_bleu;
    Image carre_vert;
    Image carre_blanc;
    Image carre_orange;
    Image carre_jaune;

    Image melanger;
    Image reset;
```

Classe SDLJeu gérant le jeu avec un affichage graphique SDL2

```
    Image losange_blanc;
    Image losange_rouge;
    Image losange_bleu;
    Image losange_orange;
    Image losange_jaune;
    Image losange_vert;

    Image losange_bas_blanc;
    Image losange_bas_rouge;
    Image losange_bas_bleu;
    Image losange_bas_orange;
    Image losange_bas_jaune;
    Image losange_bas_vert;

    public:
    SDLJeu();
    ~SDLJeu();
    void SDLBoucle();
    void DrawCube3D();
    void SDLAffichage();
};
```

# Conclusion

- Objectifs atteints:
  - Mode texte et Mode Graphique SDL sont fonctionnels
  - Les mouvements du Rubik's Cube fonctionnent
- Objectifs non atteints:
  - Affichage d'un Cube en 3D
  - Implémentation d'un chronomètre
  - Aide à la résolution du cube,
- Difficultés rencontrées:
  - Les erreurs de segmentations
  - L'utilisation de Valgrind
  - Le manque de temps

- Merci pour votre attention!