

Programación orientada a objetos

Proyecto final

Simulador de memoria

Profesor: Esp. Ing. Cesar Omar Aranda

Integrantes:

ARIAS, Stefania

ARTEAGA, Araceli

GARCIA, Maximiliano

Universidad Nacional de Cuyo

Facultad de Ingeniería

Mendoza- Argentina

2015

Introducción.

En el presente informe procederá a describir los aspectos teóricos involucrados en la realización del Proyecto Final Integrador a modo de incorporar aspectos avanzados del lenguaje c++. Para ello se realizó un trabajo de investigación e implementación de los siguientes temas:

1. SDL2.
2. Threads.
3. Mutex. (Elegido por el grupo)
4. SDL_ttf (Elegido por el grupo)

El informe está dividido en secciones donde se explicará el marco teórico de los temas investigados, se describirán las clases utilizadas, la lista de archivos, un anexo con los programas y conclusiones.

Consiste en la realización de un simulador de memoria ocupada por procesos, donde el enfoque principal debe hacerse sobre la gestión de memoria. El sistema debe permitir observar gráficamente la ocupación de memoria según los requerimientos realizados por diferentes procesos. Se prevén 2 estrategias de gestión de memoria: aleatoria o secuencial.

Este informe fue realizado a pedido del Ingeniero Cesar Aranda para cumplimentar una instancia evaluativa en la materia Programación Orientada a Objetos de la Facultad de Ingeniería en la “Universidad Nacional de Cuyo”, Mendoza, Argentina.

SDL2 “Simple DirectMedia Layer”.

En el programa se ha incorporado el manejo de la librería externa **SDL2 Simple DirectMedia Layer (SDL)** es un conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes. También proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia. Pese a estar programado en C, tiene *wrappers* a otros lenguajes de programación como C++ (el utilizado para el desarrollo de este proyecto).


Una de sus grandes virtudes es el tratarse de una biblioteca multiplataforma, siendo compatible oficialmente con los sistemas **Microsoft Windows, GNU/Linux, Mac, etc.**

Para la correcta implementación y ejecución de la misma se ha instalado el compilador, de acuerdo a la arquitectura del ordenador, MinGW en este caso 64 bits (para poder utilizar hilos en el IDE ZingAI).








1-Se va a la ubicación donde está guardada esta librería y eligiendo la arquitectura correcta, copiamos la carpeta **SDL2** que se encuentra en el include. En nuestro caso:

 C:\SDL2-2.0.5\x86_64-w64-mingw32\include

Esta carpeta se pega en la ubicación de la carpeta del compilador MinGW, eligiendo la arquitectura correcta, en la carpeta include. En este caso:


 C:\mingw64\mingw64\x86_64-w64-mingw32\include

2- Se va nuevamente a la ubicación de la librería, recordando siempre elegir la arquitectura correcta, se ingresa a la carpeta “lib” y se copian los archivos seleccionados:

	cmake	09/11/2016 15:33	Carpeta de archivos	
	pkgconfig	09/11/2016 15:33	Carpeta de archivos	
<input checked="" type="checkbox"/>	 libSDL2.a	20/10/2016 0:59	Archivo A	8.268 KB
<input checked="" type="checkbox"/>	 libSDL2.dll.a	20/10/2016 0:59	Archivo A	349 KB
	 libSDL2.la	20/10/2016 0:59	Archivo LA	2 KB
	 libSDL2_test.a	20/10/2016 0:59	Archivo A	487 KB
<input checked="" type="checkbox"/>	 libSDL2main.a	20/10/2016 0:59	Archivo A	12 KB

Estos archivos se pegan en la carpeta “lib” del MinGW, de la arquitectura correcta

3- En la ubicación de la librería, vamos a la carpeta “bin” y copiamos el .dll, este se pega en la carpeta “dist” del proyecto, en el debug correspondiente. En este caso:

 C:\Users\Araceli\Documents\NetBeansProjects\SDL2project\dist\Debug64\MinGW-Windows

Luego de haber copiado correctamente los archivos, en IDE a utilizar, en este caso NETBEANS 8.02, se configuran las propiedades del proyecto. En el linker, en “Additional options” se agregan los parámetros extra de enlazado. En este caso:

Categories:

- General
- Build
 - C Compiler
 - C++ Compiler
 - Fortran Compiler
 - Assembler
 - Linker**
 - Packaging
- Run
- Debug
- Related Projects
- Formatting

Configuration: Debug (active) Manage Configurations...

General

Output: \${CND_DISTDIR}/\${CND_CONF}/\${CND_PLATFORM}...

Additional Library Directories: ...

Options

Strip Symbols: ☐

Input

Additional Dependencies: ...

Tool

Tool: g++

Libraries

Libraries: ...

Compilation Line

Additional Options: **-lmingw32 -lSDL2main -lSDL2 -lSDL2_ttf**

Debug

OK Cancel Apply Help

NOTA: Para el correcto funcionamiento de la librería SDL2, en el main de este proyecto conservar los argumentos predefinidos, si estos se borran aparecerá un error al compilar:

“undefined reference to ‘SDL_main’ ”

Inicializando la librería

Se debe incluir el archivo SDL2/SDL.h:

#include <SDL2/SDL.h>

Para inicializar la SDL2 se debe llamar a la función `SDL_Init (SDL_INIT_VIDEO)`, la cual retorna un -1 si algo falla y 0 si tuvo éxito. El parámetro que toma es para especificar que partes de SDL inicializa, en este caso inicializa el subsistema de video. Se puede inicializar uno o varios subsistemas (separando con ||).

```
void graficar::VentanaPpal() {  
    SDL_Init(SDL_INIT_EVERYTHING);  
    TTF_Init();  
    SDL_CreateWindowAndRenderer(800, 600, SDL_WINDOW_SHOWN, &window, &renderer);  
    GenerarFondo(800, 600);  
    TextGen();  
    SDL_SetWindowTitle(window, "Memoria");  
}
```

En este proyecto se llama a `SDL_Init` para inicializar todos los subsistemas.

Como se puede apreciar en la imagen anterior, para crear la ventana y el renderer utilizó la función:

`SDL_CreateWindowAndRenderer` (ancho, alto, flags, &window, &renderer)

Con la función `SDL_SetWindowTitle` se le agrega un título a la ventana.

Algunas de las estructuras que se usaron en el proyecto son:

- `SDL_Rect` define un área rectangular, su definición es:

```
typedef struct {  
    Sint16 x, y;  
    Uint16 w, h;  
}SDL_Rect
```

x, y: definen la posición de la esquina superior izquierda del rectángulo.

w, h: definen el alto y ancho del rectángulo

En este proyecto todos los rectángulos se dibujan con esta estructura. Por ejemplo en el método `drawGrid()` de la clase `Graficar` se dibuja la grilla de la ventana.

```
void graficar::drawGrid() {
    SDL_Rect GridRect;
    int i;
    GridRect.w = 2;
    GridRect.h = 34 * 15;
    GridRect.y = 70;
    for (i = 0; i < 31; i++) {
        GridRect.x = i * 15 - 1 + 7;
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
        SDL_RenderFillRect(renderer, &GridRect);
    }
    GridRect.w = 30 * 15;
    GridRect.h = 2;
    GridRect.x = 7;
    for (i = 0; i < 35; i++) {
        GridRect.y = i * 15 - 1 + 70;
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
        SDL_RenderFillRect(renderer, &GridRect);
    }
    SDL_RenderPresent(renderer);
}
```

Accediendo a x, y, w y h se le dan las dimensiones al rectángulo.

SDL_SetRenderDrawColor: se utiliza para configurar el color utilizado para las operaciones de dibujo (Rect, Line y Clear)

SDL_RenderFillRect: para rellenar un rectángulo en el destino de representación actual con el color de dibujo.

SDL_RenderPresent: para actualizar la pantalla con cualquier representación realizada desde la llamada anterior.

- **SDL_Surface:** una superficie básicamente es una área de memoria, ya sea memoria del sistema (RAM) o memoria de video (de la tarjeta de video). En esta área de memoria podemos almacenar imágenes, dibujar sobre ella, etc.

```
typedef struct SDL_Surface {
    Uint32 flags;
    SDL_PixelFormat *format;
    int w, h;
    Uint16 pitch;
    void *pixels;
    SDL_Rect clip_rect;
    int refcount;
} SDL_Surface;
```

format: un puntero a una estructura que indica el formato de cada pixel.

w, h: ancho y alto de la superficie.

pitch: corresponde a la longitud de una línea de escaneo (scanline) de la superficie, medido en bytes.

pixels: puntero al comienzo de los datos (píxeles) de la superficie.

clip_rect: estructura que indica el área (un simple rectángulo) de clipping de la superficie.

refcount: contador de referencia, usado cuando se libera la superficie.

```
void graficar::GenerarFondo(int winsizeX, int winsizeY) {
    SDL_Rect Rectangulos;
    // Clear screen
    SDL_Rect fontRect;
    TTF_Font* fuente;
    fuente = TTF_OpenFont("times.ttf", 60);
    SDL_Color fColor;
    fColor.r = 136;
    fColor.g = 0;
    fColor.b = 21;

    Rectangulos.x = 0;
    Rectangulos.y = 0;
    Rectangulos.w = winsizeX;
    Rectangulos.h = winsizeY;
    SDL_SetRenderDrawColor(renderer, 192, 192, 192, SDL_ALPHA_OPAQUE);
    SDL_RenderFillRect(renderer, &Rectangulos);

    Rectangulos.x = 7;
    Rectangulos.y = 70;
    Rectangulos.w = 30 * 15;
    Rectangulos.h = 34 * 15;
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, SDL_ALPHA_OPAQUE);
    SDL_RenderFillRect(renderer, &Rectangulos);

    Rectangulos.x = 500;
    Rectangulos.y = 90;
    Rectangulos.w = 780 - 500;
    Rectangulos.h = 450;
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);
    SDL_RenderFillRect(renderer, &Rectangulos);

    SDL_Surface* surfaceMessage = TTF_RenderText_Solid(fuente, "Memoria RAM", fColor);
    SDL_Texture* Message = SDL_CreateTextureFromSurface(renderer, surfaceMessage);
    fontRect.x = 135;
    fontRect.y = 35;
    fontRect.w = 300 - 135;
    fontRect.h = 65 - 35;
    SDL_RenderCopy(renderer, Message, NULL, &fontRect);

    surfaceMessage = TTF_RenderText_Solid(fuente, "Proceso en ejecucion", fColor);
    Message = SDL_CreateTextureFromSurface(renderer, surfaceMessage);
    fontRect.x = 540;
    fontRect.y = 60;
    fontRect.w = 730 - 540;
    fontRect.h = 85 - 60;
    SDL_RenderCopy(renderer, Message, NULL, &fontRect);
}
```

```
if (!UsoMem) {  
    Rectangulos.x = 7;  
    Rectangulos.y = 70 + 17 * 15;  
    Rectangulos.w = 30 * 15;  
    Rectangulos.h = 17 * 15;  
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, SDL_ALPHA_OPAQUE);  
    SDL_RenderFillRect(renderer, &Rectangulos);  
}  
SDL_RenderPresent(renderer);  
}
```

SDL_CreateTextureFromSurface: se utiliza para crear una textura a partir de una superficie existente.

SDL_RenderCopy: se utiliza para copiar una parte de la textura al destino de renderización actual.

Nuevos “tipos de datos” en SDL:

Uint8 : unsigned char a 8 bit (1 byte)

Uint16 : unsigned int a 16 bit (2 byte)

Uint32 : unsigned int a 32 bit (4 byte)

Uint64 : unsigned int a 64 bit (8 byte)

Sint8 : signed char a 8 bit (1 byte)

Sint16 : signed int a 16 bit (2 byte)

Sint32 : signed int a 32 bit (4 byte)

Sint64 : signed int a 64 bit (8 byte)

- SDL_Event: el manejo de eventos se inicializa junto con la inicialización del subsistema de video, es decir SDL_Init (SDL_INIT_VIDEO). SDL internamente almacena sin procesar en una cola de eventos, los trata a través de una unión SDL_Event :

```
Typedef union {  
    Uint8 type;  
    SDL_ActiveEvent active;  
    SDL_KeyboardEvent key;  
    SDL_MouseMotionEvent motion;  
    SDL_MouseButtonEvent button;  
    SDL_JoyAxisEvent jaxis;  
    SDL_JoyBallEvent jball;  
    SDL_JoyHatEvent jhat;  
    SDL_JoyButtonEvent jbutton;  
    SDL_ResizeEvent resize;  
    SDL_QuitEvent quit;  
    SDL_UserEvent user;  
    SDL_SysWMEvent syswm;  
} SDL_Event;
```

type: el tipo de evento.

active: evento de activación.

key: evento de teclado.

motion: evento de movimiento del mouse.

button: evento de botón del mouse.

jaxis: evento de movimiento de eje de joystick.

jball: evento de movimiento del trackball del joystick.

jhat: evento de movimiento del minijoystick (hat) del joystick.

jbutton: evento de botón del joystick.

resize: evento de redimensionado de ventana.

quit: evento de petición de cierre de aplicación.

user: evento definido por el usuario.

syswm: evento indefinido del gestor de ventanas.

Antes del salir del programa se debe llamar a la función `SDL_Quit()`, la cual cierra todos los subsistemas creados inicialmente y libera los recursos ocupados por ellos.

```
void graficar::Salir() {  
    while (event.type != SDL_QUIT) {  
        SDL_PollEvent(&event);  
    }  
    SDL_DestroyWindow(window);  
    SDL_Quit();  
}
```

En el while, verifica que mientras no haya un evento generado por el usuario voy a llamar a `SDL_PollEvent`.

`SDL_PollEvent`: se utiliza para realizar sondeos de eventos pendientes.

`SDL_DestroyWindow`: se utiliza para destruir la ventana.

Módulo TTF de SLD2.

En el programa se ha incorporado al manejo de la librería `SDL2` el módulo independiente `SDL2_TTF`, el cual se utiliza para renderizar texto, para ello se siguen los siguientes pasos:

Se instala de la misma manera que `SDL2` y se agregan los archivos .dll donde se encuentra el ejecutable .exe

Se debe incluir el archivo `SDL2/SDL_TTF.h`:

#include <SDL2/SDL_TTF.h>

Incluir en la carpeta de proyecto el archivo de fuente con extensión .ttf para poder cargar la fuente a utilizar, en este caso "tahoma.ttf"

El módulo debe inicializarse con la función `TTF_Init()`;

En este proyecto en concreto se utilizó una función de la siguiente forma:

```

void Graficar::TextGen() {
    SDL_Rect fontRect;
    TTF_Font* fuente;
    fuente = TTF_OpenFont("Tahoma.ttf", 30);
    SDL_Color fColor;
    fColor.r = 0;
    fColor.g = 255;
    fColor.b = 0;
    char* text;
    for (int i = 0; i < auxiliarVectProc.size(); i++) {
        text = auxiliarVectProc[i];
        SDL_Surface* surfaceMessage = TTF_RenderText_Solid(fuente, text, fColor);
        SDL_Texture* Message = SDL_CreateTextureFromSurface(renderer, surfaceMessage);
        fontRect.x = 500;
        fontRect.y = 95+i*20;
        if(auxiliarVectType[i]==0){
            fontRect.w = 15*10;
        }
        else{
            fontRect.w = 15*10;
        }
        fontRect.h = 20;
        SDL_RenderCopy(renderer, Message, NULL, &fontRect);
    }
    SDL_RenderPresent(renderer);
}

```

SDL_Rect fontRect: definimos el rectángulo donde estará el texto

TTF_Font* fuente: definimos el apuntador de la fuente

fuente = TTF_OpenFont ("tahoma.ttf", 30): abrimos el archivo y tamaño de fuente, en este caso tahoma

SDL_Color fColor: definimos el color

El bucle se utiliza para múltiples impresiones, tomamos el texto de tipo char*

SDL_Surface* surfaceMessage = TTF_RenderText_Solid (fuente, text, fColor): generamos una superficie el texto y su color

El bucle if para saber si es proceso o subprocesso y definir el tamaño según cantidad de caracteres

SDL_RenderCopy (renderer, Message, NULL, &fontRect): agrega el cuadro de texto al render

Threads.

En este proyecto se hizo uso de threads para gestionar los procesos y subprocessos por separado.

Para implementar el uso de Threads se creó un vector instancia de la clase thread provista por la biblioteca thread. Para que funcione hay que utilizar una versión del compilador gcc+11 en adelante.

Para la correcta implementación y ejecución de los mismos se declara la cabecera <thread>.

#include <thread>

```

for (int i = 0; i < cant; i++) {
    v.at(i) = thread(&graficar::ProcesosHilos, &MapaMemoria, Vector1[i], Vector2[i], procesos[i].getPila(), procesos[i].getCodigo(),
                    procesos[i].getDatos(), procesos[i].getTiempoVida(), i + 1, procesos[i].getNroProceso(), procesos[i].getNroSubProc());
}
for (int j = 0; j < procesos.size(); j++) {
    v[j].join();
}

```

Se creó un vector de objetos hilos, para gestionar con cada hilo un proceso o subproceso distinto. Como se puede observar en la imagen anterior, el hilo se define colocando entre paréntesis la función a ejecutar por el mismo, seguido de los argumentos de dicha función.

Se utiliza la función `join()` para esperar que el hilo termine su proceso. Es necesario agregarla porque sino el hilo no se crea, ya que lo corta al pasar a la siguiente instrucción.

Mutex

Se utilizó mutex para que los procesos tengan acceso a las mismas funciones sin pisarse entre sí.

Para la correcta implementación y ejecución se declara la cabecera `<mutex>`

`#include <mutex>`

En el método `procesosHilos` de la clase `Graficar`

```
int graficar::ProcesosHilos(int x, int y, int tamano1, int tamano2, int tamano3, int tpo, int hilo, int nroproceso, int nrosubproc) {
    int i, posicion2 = 0, posicion3 = 0;
    posicion2 = x + tamano1;
    int aux1 = 0, aux2 = 0, aux3 = 0;
    for (i = 0; i < (tpo * 40); i++) {
        if (x + tamano1 < 29) {
            mtx.lock();
            dibujar(255, 0, 0, x, y, tamano1);
            mtx.unlock();
        } else {
            mtx.lock();
            dibujar(255, 0, 0, x, y, 30 - x);
            mtx.unlock();
            if (x + tamano1 - 30 > 0) {
                aux1 = x + tamano1 - 30;
                y++;
                mtx.lock();
                dibujar(255, 0, 0, 0, y, aux1);
                mtx.unlock();
                y--;
            }
        }
        if (posicion2 + tamano2 <= 29) {
            mtx.lock();
            dibujar(0, 255, 0, posicion2, y, tamano2);
            mtx.unlock();
        } else {
            if (30 - posicion2 > 0) {
```

```
mtx.lock();
dibujar(0, 255, 0, posicion2, y, 30 - posicion2);
mtx.unlock();
aux2 = posicion2 + tamano2 - 30;
y++;
mtx.lock();
dibujar(0, 255, 0, 0, y, aux2);
mtx.unlock();
y--;
} else {
    aux2 = tamano2;
    y++;
    mtx.lock();
    dibujar(0, 255, 0, aux1, y, aux2);
    mtx.unlock();
    y--;
}
}
if (tamano3 != 0) {
    posicion3 = posicion2 + tamano2;
    if (posicion3 + tamano3 <= 29) {
        mtx.lock();
        dibujar(0, 0, 255, posicion3, y, tamano3);
        mtx.unlock();
    } else {
        if (30 - posicion3 > 0) {
            mtx.lock();
            dibujar(0, 0, 255, posicion3, y, 30 - posicion3);
            mtx.unlock();
            aux3 = posicion3 + tamano3 - 30;
            y++;
            mtx.lock();
            dibujar(0, 0, 255, 0, y, aux3);
            mtx.unlock();
            y--;
        } else {
            aux3 = aux1 + aux2;
            y++;
            mtx.lock();
            dibujar(0, 0, 255, aux3, y, tamano3);
            mtx.unlock();
            y--;
        }
    }
}
Sleep(25);
}
mtx.lock();
QuitarTextoProc(nroproceso, nrosubproc);
mtx.unlock();
mtx.lock();
GenerarFondo(800, 600);
mtx.unlock();
mtx.lock();
drawGrid();
mtx.unlock();
mtx.lock();
TextGen();
mtx.unlock();
return 0;
```

Función lock (): El hilo llamante bloquea el mutex, bloqueando si es necesario:

Si el mutex no está bloqueado actualmente por ningún hilo, el hilo llamante lo bloquea (desde este punto, y hasta que se llama a su desbloqueo de miembro, el hilo posee el mutex).

Si el mutex está actualmente bloqueado por otro subproceso, la ejecución del subproceso de llamada se bloquea hasta que se desbloquee por el otro subproceso (otros subprocesos no bloqueados continúan su ejecución).

El bloqueo de función no miembro permite bloquear más de un objeto mutex simultáneamente, evitando los bloqueos potenciales que pueden ocurrir cuando varios hilos bloquean / desbloquean objetos mutex individuales en órdenes diferentes.

Función unlock (): desbloquea el mutex, liberando la propiedad sobre él. Si otros hilos están actualmente bloqueados intentando bloquear este mismo mutex, uno de ellos adquiere la propiedad sobre él y continúa su ejecución.

Si el mutex no está bloqueado actualmente por el subproceso de llamada, provoca un comportamiento indefinido.

Todas las operaciones de bloqueo y desbloqueo del mutex siguen un solo orden total, con todos los efectos visibles sincronizados entre las operaciones de bloqueo y las operaciones de desbloqueo anteriores en el mismo objeto.

En este método el hilo bloquea a la función dibujar, y hasta que no termina de dibujar el proceso no la desbloquea. Luego la libera para que otro hilo pueda utilizarla.

Diagrama de clases:

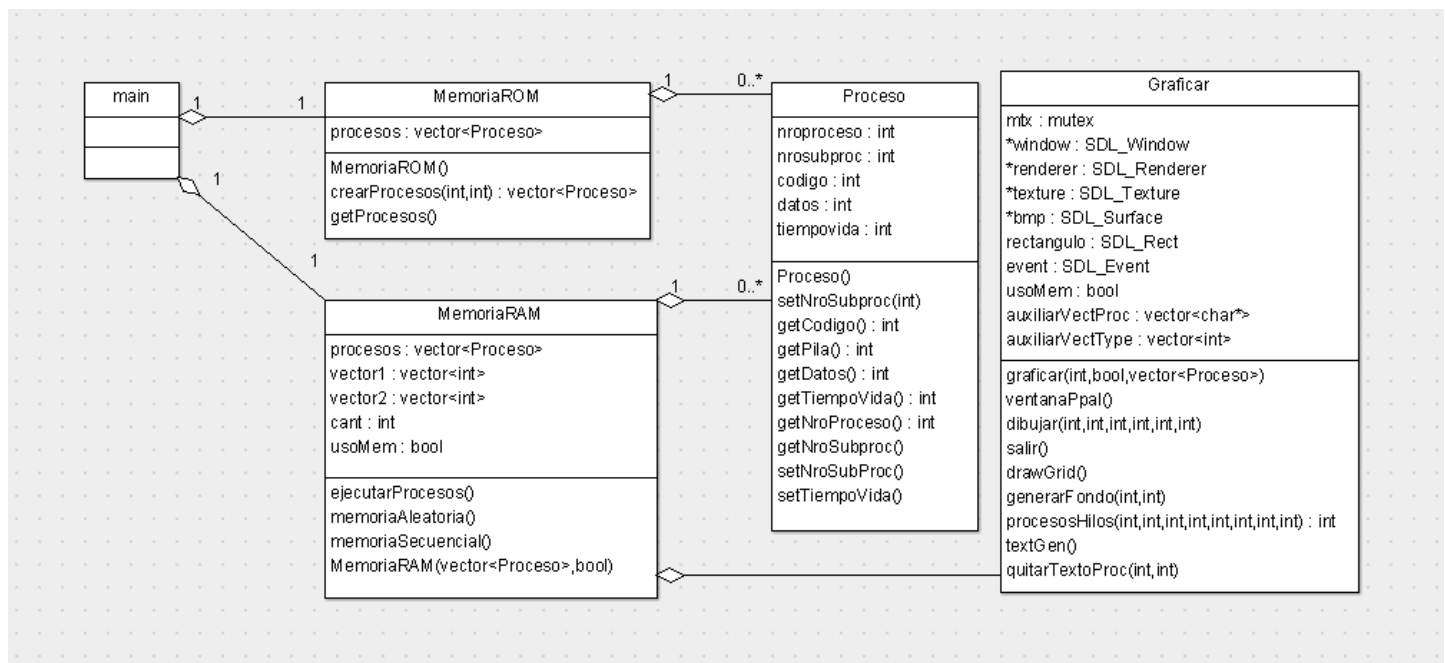


Diagrama de secuencia:

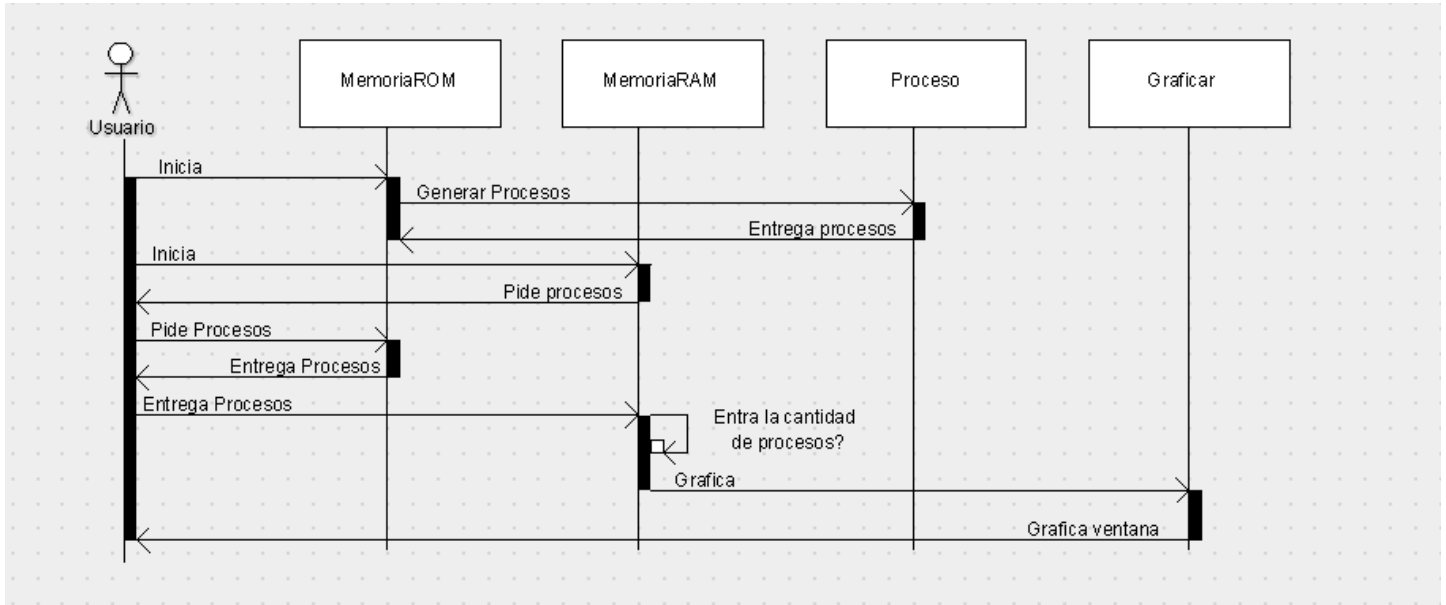
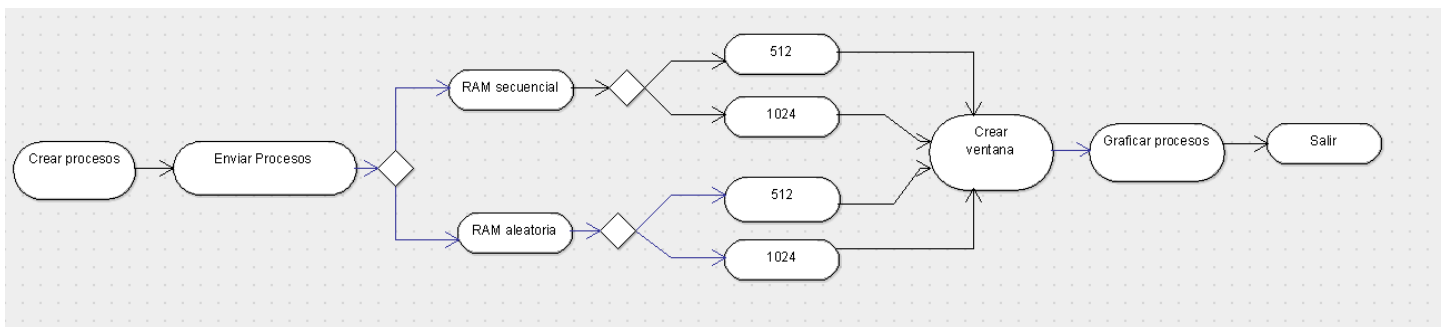


Diagrama de actividad:



Diseño del modelo planteado

Proceso.h

```
#ifndef PROCESO_H
#define PROCESO_H
#include <ctime>
#include <vector>
#include <stdlib.h>
#include <iostream>
#include <string>
using namespace std;
class Proceso {
public:
    Proceso();
    Proceso(int,int,int,int,int);
    virtual ~Proceso();
    int getCodigo();
    int getPila();
    int getDatos();
    int getTiempoVida();
    int getNroProceso();
    int getNroSubProc();
    void setNroSubproc();
    void setNroSubProc(int);
    void setTiempoVida();

private:
    int nroproceso,nrosubproc,codigo,pila,datos,tiempovida;
};
#endif /* PROCESOS H */
```

Esta clase se encarga de inicializar los procesos, dándoles un nombre de proceso o subproceso, un valor de código, pila, datos y un tiempo de vida.

MemoriaRAM.h

```
#ifndef MEMORIARAM_H
#define MEMORIARAM_H
#include "Proceso.h"
#include "Graficar.h"

class MemoriaInsuficiente{
public:
    MemoriaInsuficiente(){};
};

class MemoriaRAM {
private:
    vector<Proceso> procesos;
    vector< int > Vector1;
    vector< int > Vector2;
    int cant;
    bool UsoMem;

public:
    void EjecutarProcesos();
    void MemoriaAleatoria();
    void MemoriaSecuencial();
    MemoriaRAM(vector<Proceso> procesos, bool UsoMem);
    ~MemoriaRAM();
};

#endif
```

Esta clase analiza los procesos que recibe, de manera secuencial o aleatoria y los gráfica.

MemoriaROM.h

```
#ifndef MEMORIAROM_H
#define MEMORIAROM_H
#include "Proceso.h"

using namespace std;

class MemoriaROM {
public:
    MemoriaROM();
    virtual ~MemoriaROM();
    void CrearProcesos(int, int);
    vector<Proceso> getProcesos();
private:
    vector<Proceso> procesos;
};

#endif /* MEMORIAROM_H */
```

Esta clase se encarga de crear un vector de procesos para enviárselos al main y de ahí a la Memoria RAM

Graficar.h

```
#ifndef GRAFICAR_H
#define GRAFICAR_H
class Graficar {
private:
    mutex mtx;
    SDL_Window* window;
    SDL_Renderer* renderer;
    SDL_Texture *texture ;
    SDL_Surface *bmp ;
    SDL_Rect rectangulo ;
    SDL_Event event;
    bool UsoMem;
    vector<char*> auxiliarVectProc;
    vector<int> auxiliarVectType;
public:
    Graficar(int a, bool UsoMem, vector<Proceso> procesos);
    void dibujar(int r, int g, int b, int x, int y, int tamano);
    void VentanaPpal();
    void Salir();
    void drawGrid();
    void GenerarFondo(int winsizeX, int winsizeY);
    int ProcesosHilos(int x, int y, int tamano1, int tamano2, int tamano3, int tpo, int hilo, int nroproceso, int nrosubproc);
    void TextGen();
    void QuitarTextoProc(int nroproceso, int nrosubproc);
};

#endif
```

Esta clase se encarga de crear la ventana escribir y graficar sobre ella los procesos en ejecución.

A continuación podemos ver el despliegue de la salida del programa.

```
sdl2project
Cuantos procesos desea crear como maximo?
```

```
sdl2project
Cuantos procesos desea crear como maximo?
15
Cuantos subprocesos desea crear como maximo?
```

Acá se puede observar la primera bifurcación

```
sdl2project
Cuantos procesos desea crear como maximo?
15
Cuantos subprocesos desea crear como maximo?
15
Que memoria desea ejecutar
0. Memoria Secuencial
1. Memoria Aleatoria
```

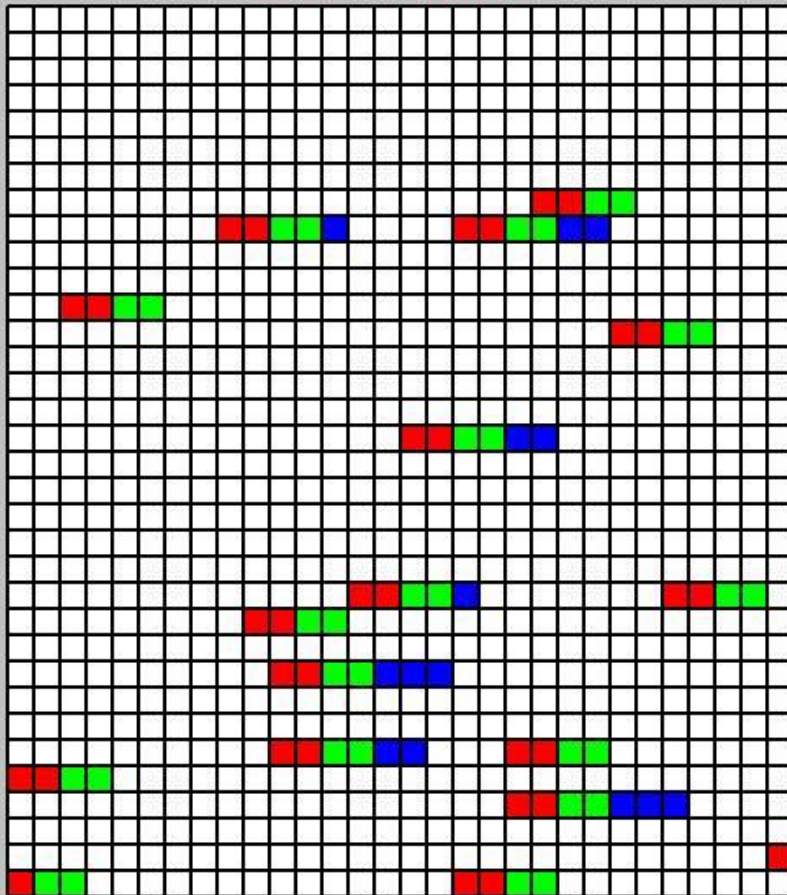
```
sdl2project
Cuantos procesos desea crear como maximo?
15
Cuantos subprocesos desea crear como maximo?
15
Que memoria desea ejecutar
0. Memoria Secuencial
1. Memoria Aleatoria
1
Que Espacio de memoria desea tener
1. 1020
0. 510
```

Dependiendo el tamaño de memoria que se elija podemos obtener las siguientes respuestas, para memoria aleatoria

Memoria 1024

Memoria

Memoria RAM



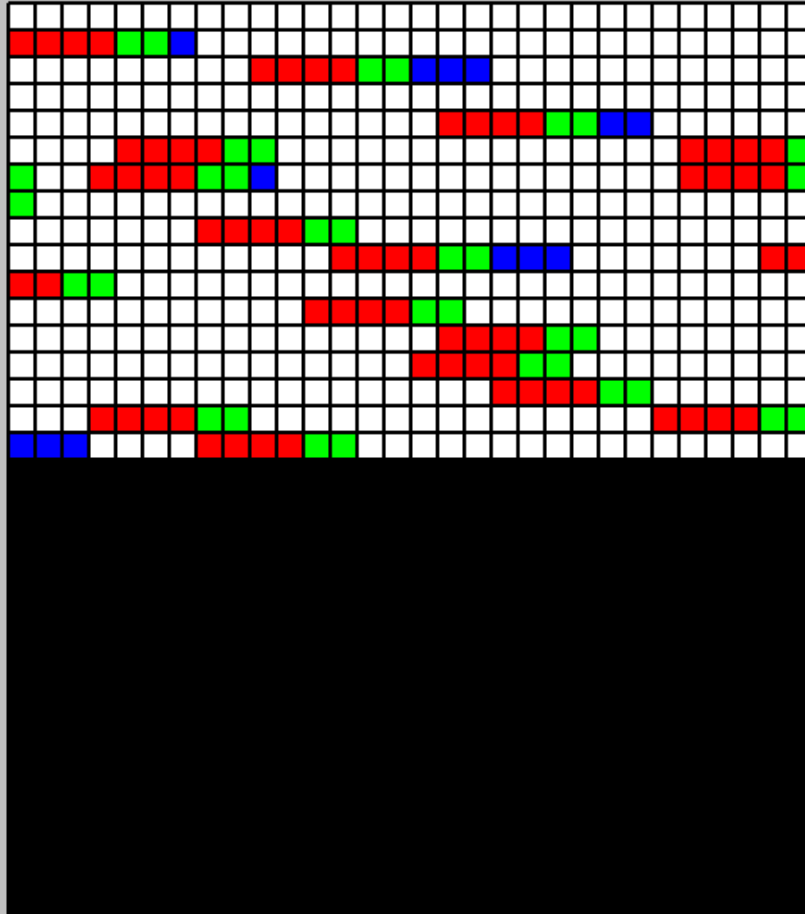
Proceso en ejecucion

```
Proceso A
Proceso B
Proceso C
Proceso D
Proceso E
Proceso F
Proceso G
Proceso G1
Proceso D1
Proceso C1
Proceso D2
Proceso E1
Proceso E2
Proceso E3
Proceso F1
Proceso G2
```

Memoria 512

Memoria

Memoria RAM

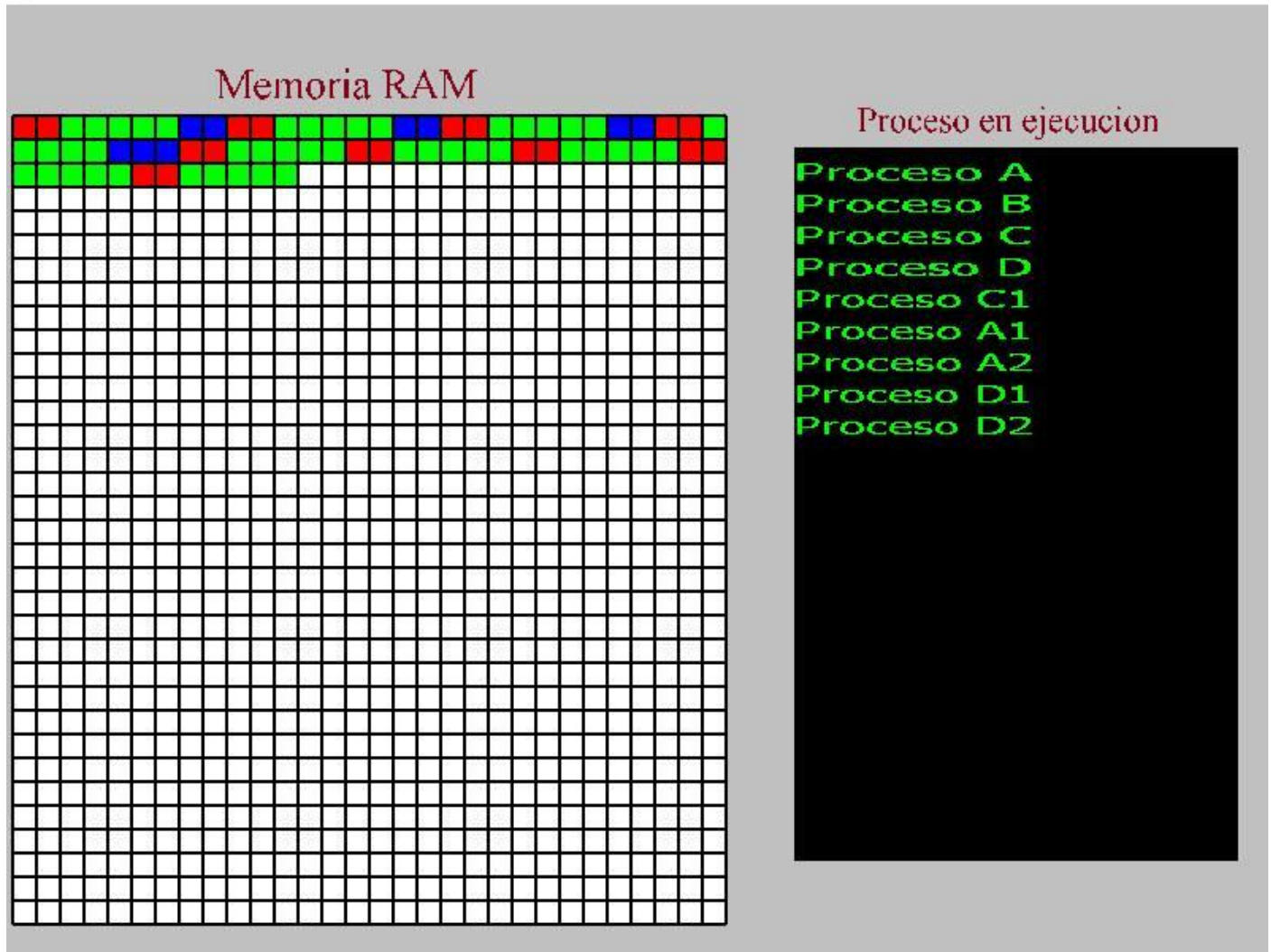


Proceso en ejecucion

Proceso A
Proceso B
Proceso C
Proceso D
Proceso E
Proceso F
Proceso A1
Proceso A2
Proceso B1
Proceso C1
Proceso C2
Proceso A3
Proceso A4
Proceso E1
Proceso C3
Proceso F1
Proceso A5

Para Memoria Secuencial se pueden obtener las siguientes respuestas:
Memoria 1024

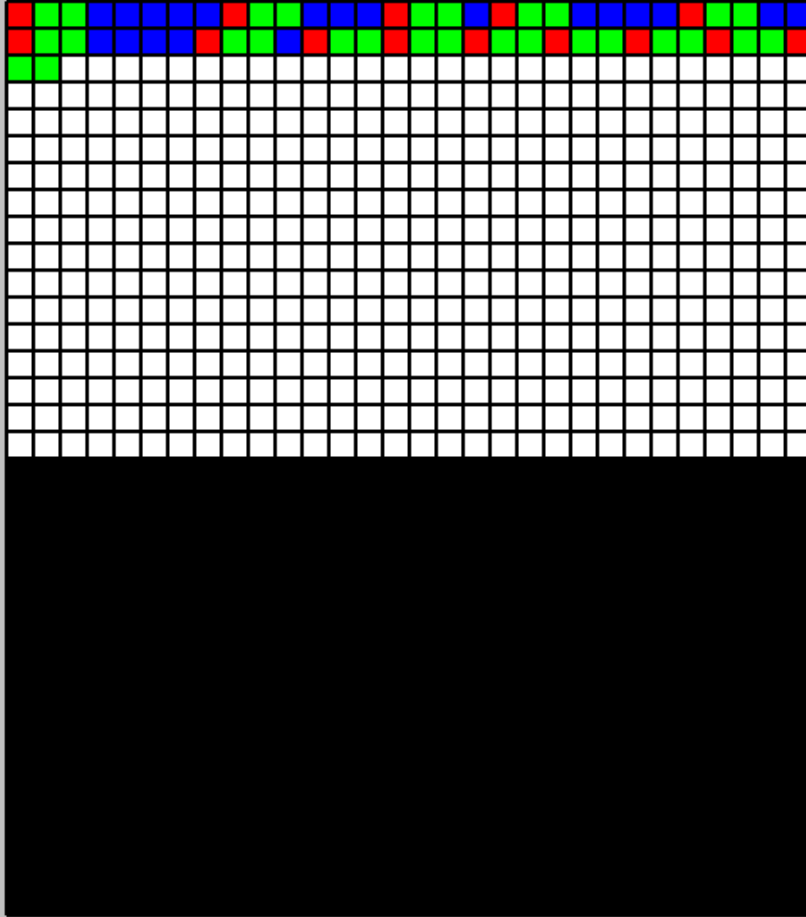
Memoria



Memoria 512

 Memoria

Memoria RAM



Proceso en ejecucion

Proceso A
Proceso B
Proceso C
Proceso D
Proceso E
Proceso F
Proceso G
Proceso G1
Proceso A1
Proceso C1
Proceso D1
Proceso F1
Proceso C2
Proceso E1

Conclusiones

Conclusiones del trabajo

En el presente trabajo pudimos ahondar sobre aspectos útiles del lenguaje C++ que desconocíamos. Pudimos evidenciar las facilidades que nos ofrece el lenguaje a la hora de interactuar con el sistema operativo en los temas que implementamos. Esta interacción está oculta para el programador que simplemente hace uso de las funciones provistas por las librerías estándares de C++ sin preocuparse por cuestiones de bajo nivel.

Comprendimos la importancia y versatilidad que ofrece el uso de hilos y mutex, simplificó mucho el trabajo con la gestión de memoria al momento de hacerla secuencial o aleatoria.

También ejercitamos la implementación del paradigma de programación orientada a objetos a la hora de decidir el diseño de nuestro modelo.

Conclusiones del grupo

Realizar este trabajo en equipo fue una experiencia diferente ya que todas las partes se relacionaban entre sí. Tuvimos que afianzar la comunicación y el trabajo en equipo.

Nos enfrentó al desafío de tener que buscar y utilizar las librerías necesarias, así como también al hecho de aprender a utilizar la librería externa SDL2.

Descubrimos que zinjaI no compila threads con minGW32, tuvimos que descargar minGW64.

Bibliografía

- <https://wiki.libsdl.org>
- <http://cerkala.blogspot.com.ar/2015/08/sdl2-203-in-netbeans-ide-802.html>
- <http://stackoverflow.com/questions/22886500/how-to-render-text-in-sdl2>
- <http://stackoverflow.com/questions/21007329/what-is-a-sdl-renderer>
- <http://www.sdltutorials.com/sdl-tutorial-basics>
- <http://windrealm.org/tutorials/sdl/sdl2-base-cpp.php>
- http://www.losersjuegos.com.ar/_media/referencia/libros/capitulo_01/capitulo_01.pdf
- http://www.losersjuegos.com.ar/_media/referencia/libros/capitulo_02/capitulo_02.pdf