Battle City

Andrés Santiago Duque Escobar
Valentina Roquemen Echeverry

Física Computacional II
Universidad de Antioquia
Medellín
2020-2

Tabla de contenidos

Link al repositorio: https://github.com/vlt-ro/ProyectoFinal_BattleCity

Battle City

Historia

Battle City es un videojuego de tanques producido y publicado por Namco como una adaptación del clásico arcade Tank Battalion.

Está disponible para las videoconsolas de Nintendo NES, Game Boy y Wii.

Battle City

¿En qué consiste el juego?

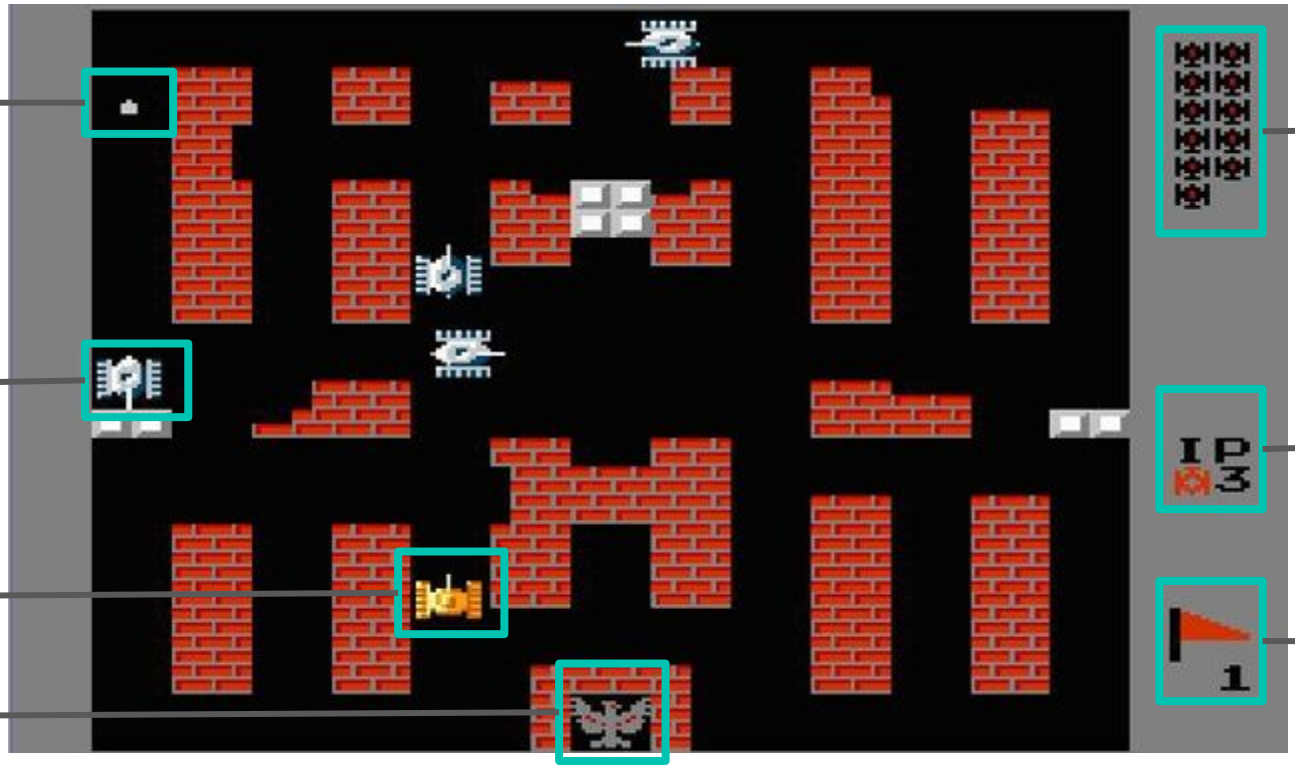
Objetivo: Controlar un tanque sobre un escenario plagado de tanques enemigos, evitando que destruyan su base militar.

Gana: Se completa el nivel cuando haya destruido todos los tanques enemigos.

Pierde: El juego termina si el enemigo destruye la base o el jugador gasta todas sus vidas.

Battle City

¿En qué consiste el juego?



Versión C++

SDL: Simple DirectMedia Layer

- Librería de desarrollo multiplataforma diseñada para proporcionar acceso de bajo nivel a hardware de audio, teclado, mouse, joystick y gráficos a través de OpenGL y Direct3D.
- Compatible oficialmente con Windows, Mac OS X, Linux, iOS y Android.
- Está escrita en C, funciona de forma nativa con C++ y hay enlaces disponibles para otros lenguajes, incluidos C# y Python.
- Se distribuye bajo la licencia zlib. Esta licencia le permite utilizar SDL libremente en cualquier software.

SDL: Simple DirectMedia Layer

Librerías

```
#include <SDL2/SDL.h>
```

`#include <SDL2/SDL_image.h>` → Permite cargar imágenes con diferentes formatos

`#include <SDL2/SDL_ttf.h>` → Permite usar fuentes

Inicialización de las librerías

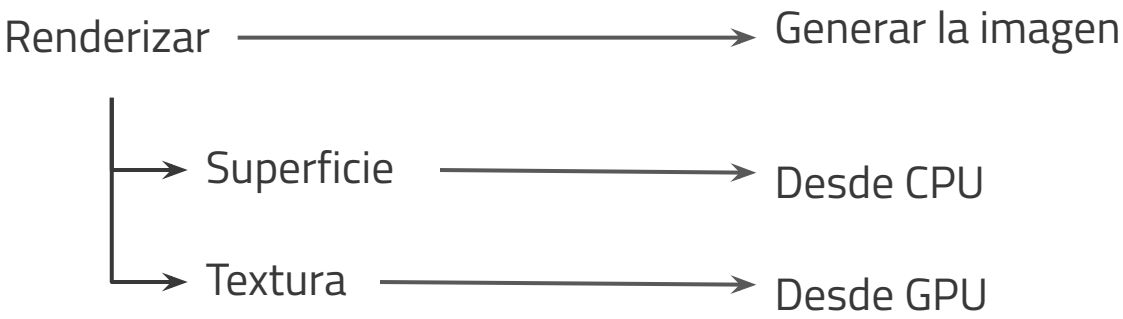
- SDL_Init()
- IMG_Init()
- TTF_Init()

Limpieza de los subsistemas

- SDL_Quit()
- IMG_Quit()
- TTF_Quit()

SDL: Simple DirectMedia Layer

Conceptos claves



Tipos

- SDL_Renderer
- SDL_Texture
- SDL_Surface

Funciones

- SDL_CreateRenderer( )
- SDL_DestroyRenderer( )
- SDL_RenderCopy( )
- SDL_RenderPresent( )

Versión C++

Otras funciones y tipos importantes

IMG_Load( )

TTF_RenderTextSolid( )

SDL_CreateTextureFromSurface()

SDL_RenderFillRect()

SDL_Delay( )

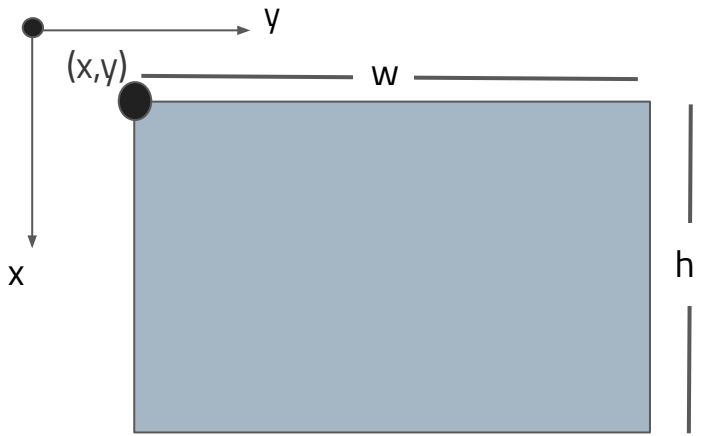
SDL_DestroyWindow( )

SDL_FreeSurface()

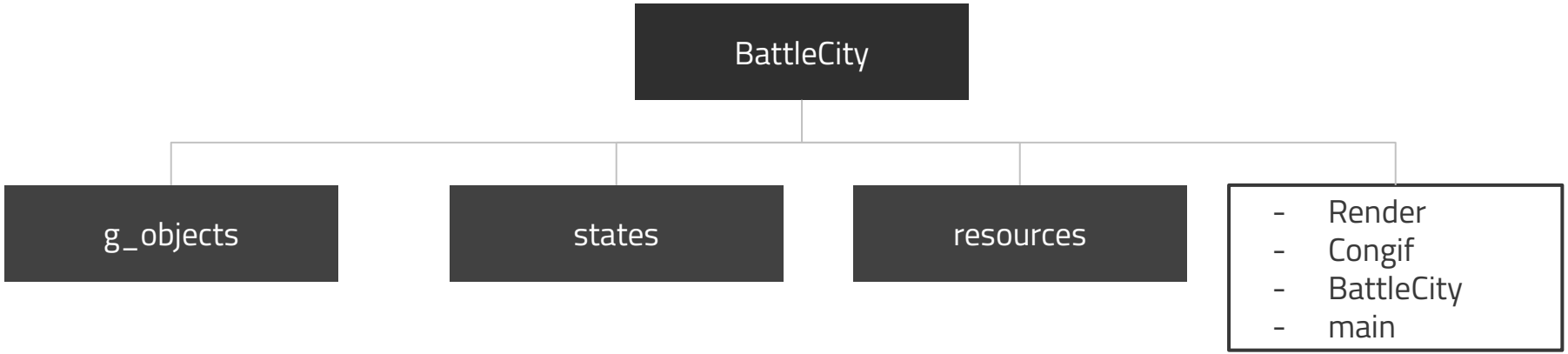
TTF_Font

SDL_Color

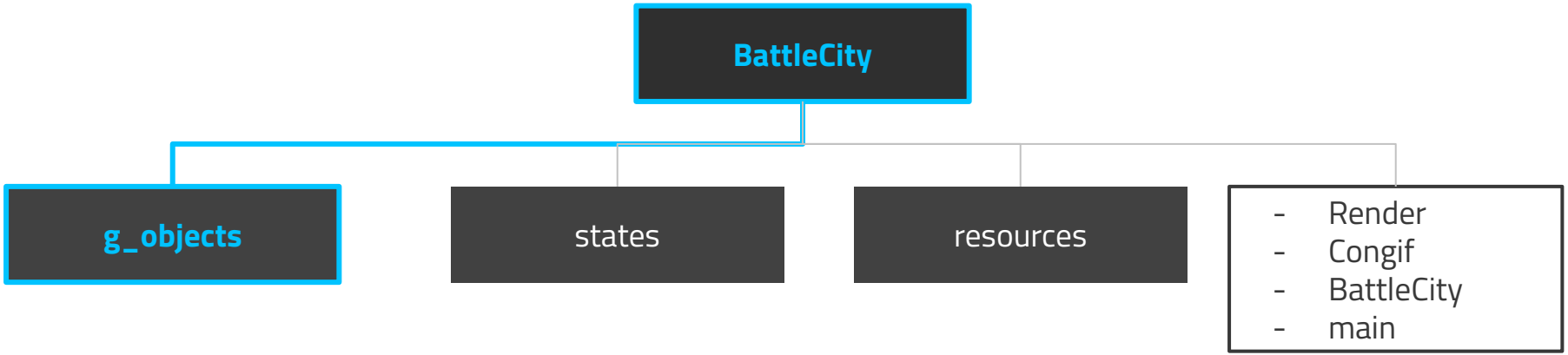
SDL_Rect rect={x,y,w,h};



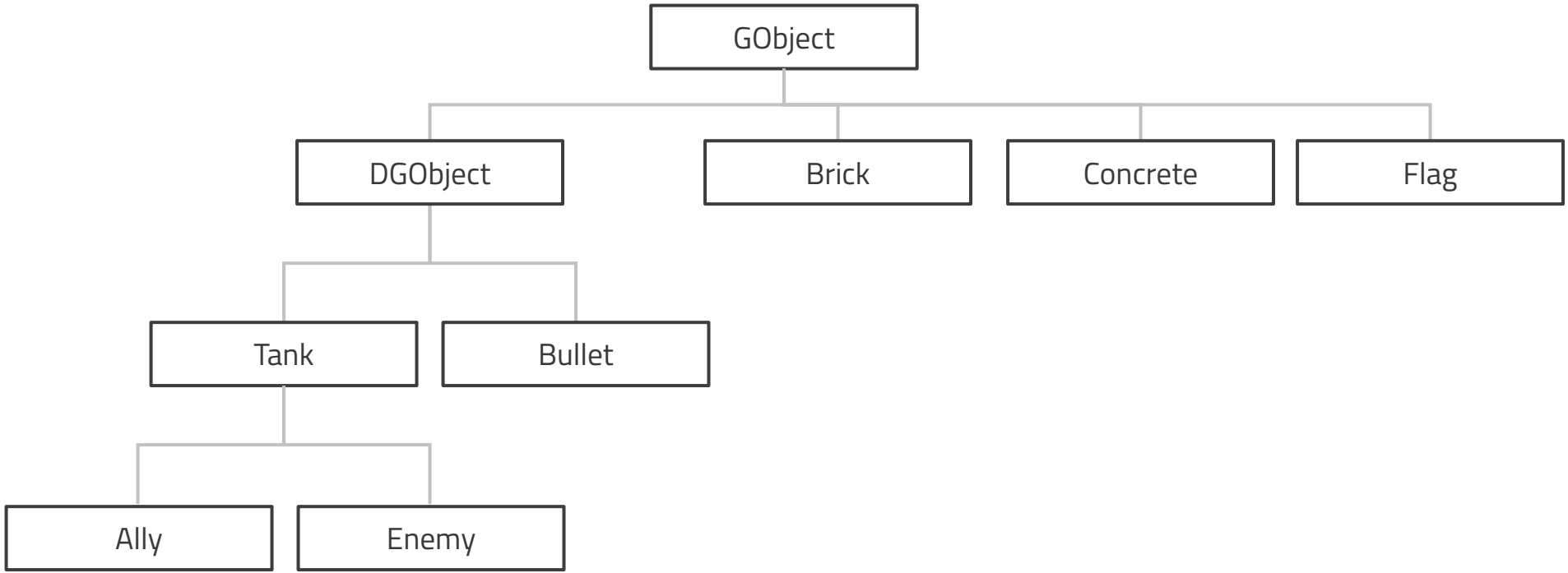
Estructura del repositorio



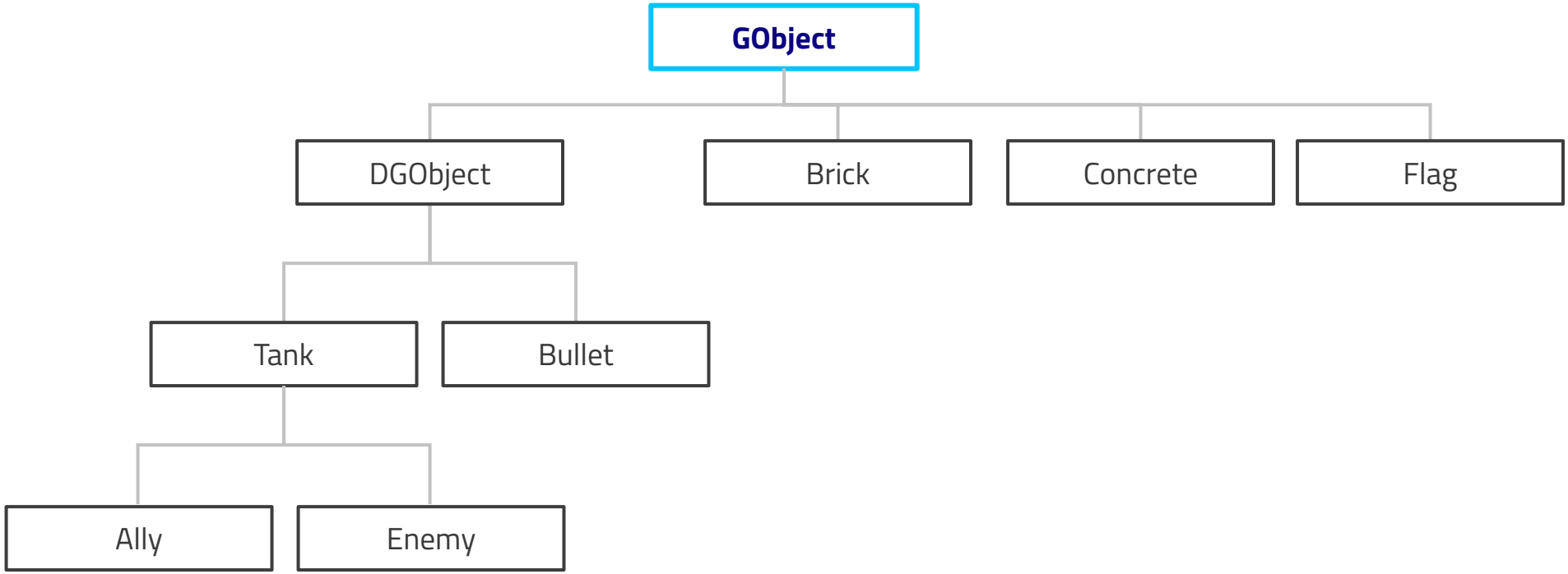
Estructura del repositorio



Estructura del juego: Clase GObject



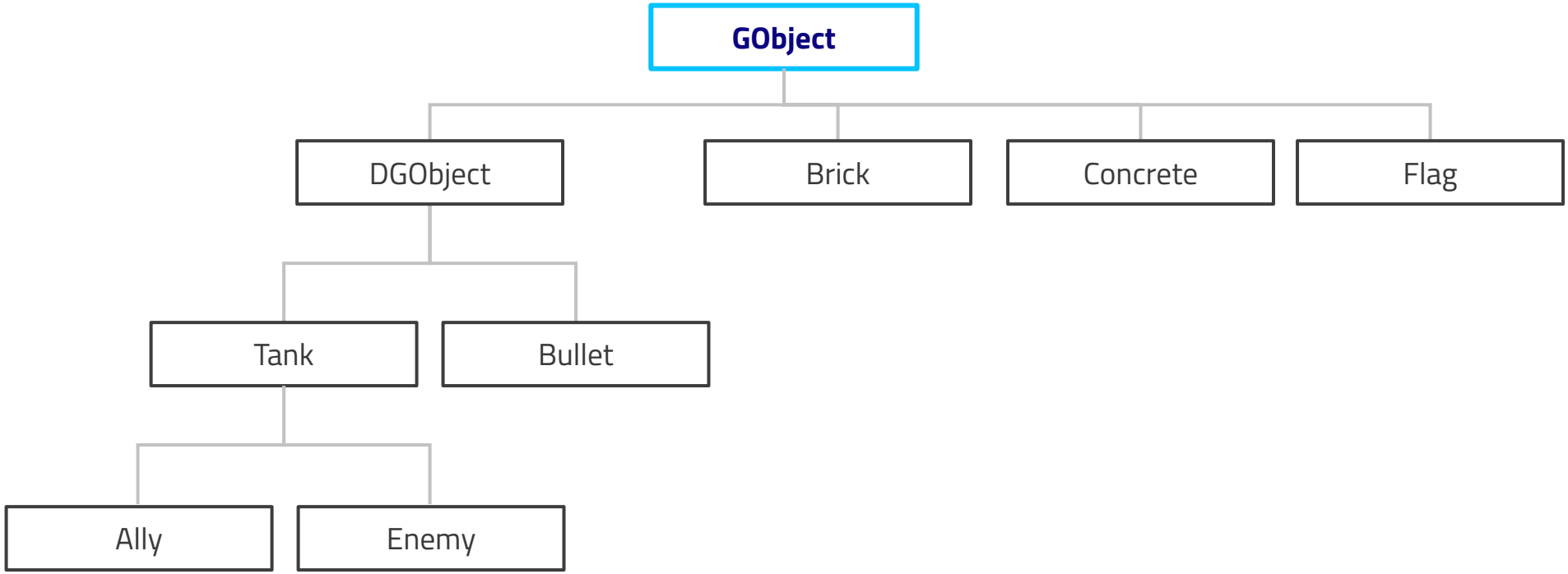
Estructura del juego: Clase GObject



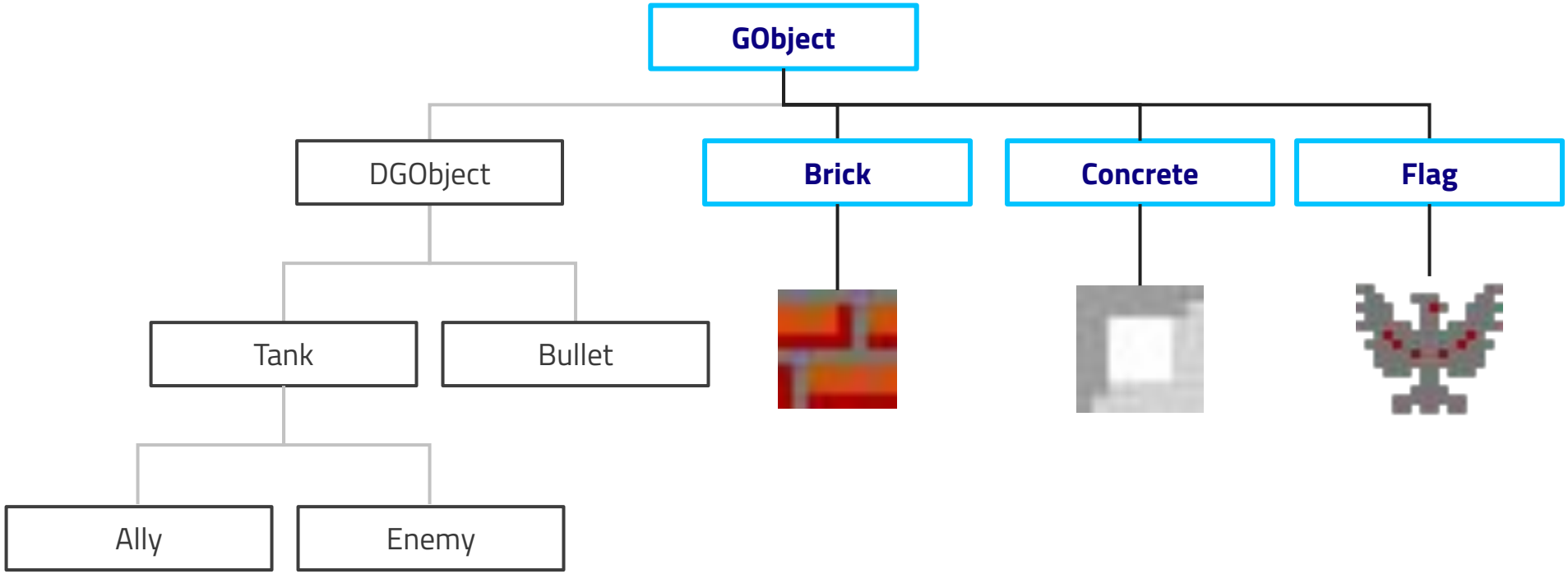
Estructura del juego: Clase GObject

```cpp
class GObject
{
public:

    /**
     * @param height    Altura de la imagen.
     * @param width     Ancho de la imagen.
     * @param xPos      Coordenada x desde la esquina superior izquierda.
     * @param yPos      Coordenada y desde la esquina superior izquierda.
     */
    GObject(size_t height=0, size_t width=0, int xPos=0, int yPos=0);

    virtual ~GObject(){};

    /**
     * Establece una nueva posición.
     */
    void setPosition(int x,int y);

    /**
     * @return Posición del objeto.
     */
    SDL_Point getPosition();

    /**
     * @return Una estructura con la información de la posición y de la
     *         dimension del objeto.
     */
    SDL_Rect getDimension();

    /**
     * @return Región donde se encuentra la textura del objeto.
     */
    const SDL_Rect * getTexture();

    /**
     * Establece la región donde se encuentra la textura del objeto.
     */
    void setTexture(const SDL_Rect *texture);

    /**
     * @return Obtiene el ID del objeto.
     */
    unsigned getID();

private:
    unsigned id;
    SDL_Rect rect; // Posición y dimensión del objeto.
    SDL_Rect texture; // Posición y dimensión de la textura.

protected:
    /**
     * Establece el ID del objeto.
     */
    void setID(unsigned);
};

#endif /* GOBJECT_H_ */
```

Estructura del juego: Clase GObject



Estructura del juego: Clase GObject



Estructura del juego: Clase GObject

```
class Flag: public GObject
{
public:
    Flag(int x, int y);
    virtual ~Flag();
};
```

```
class Concrete: public GObject
{
public:
    Concrete(int x, int y);
    virtual ~Concrete();
};
```

```
class Brick: public GObject
{
public:
    Brick(int x, int y);
    virtual ~Brick();
};
```

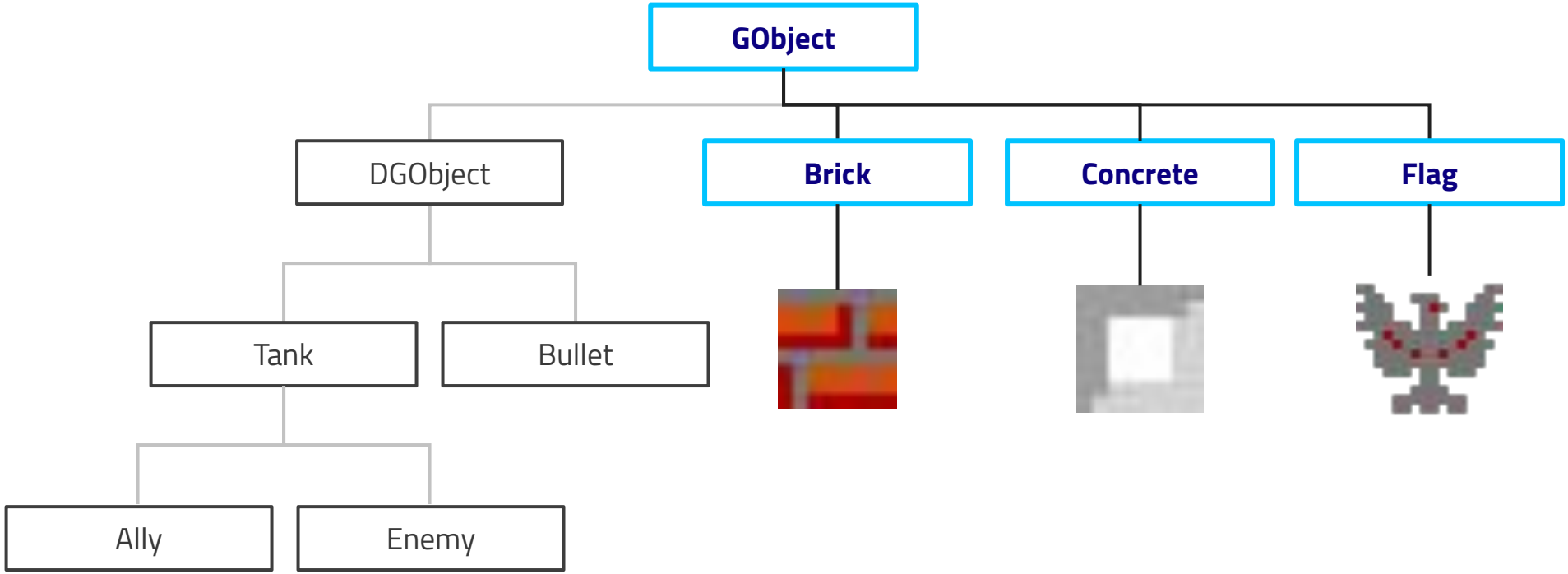
Estructura del juego- Clase GObject

```
class Flag: public GObject
{
public:
    Flag(int x, int y);
    virtual ~Flag();
};
```

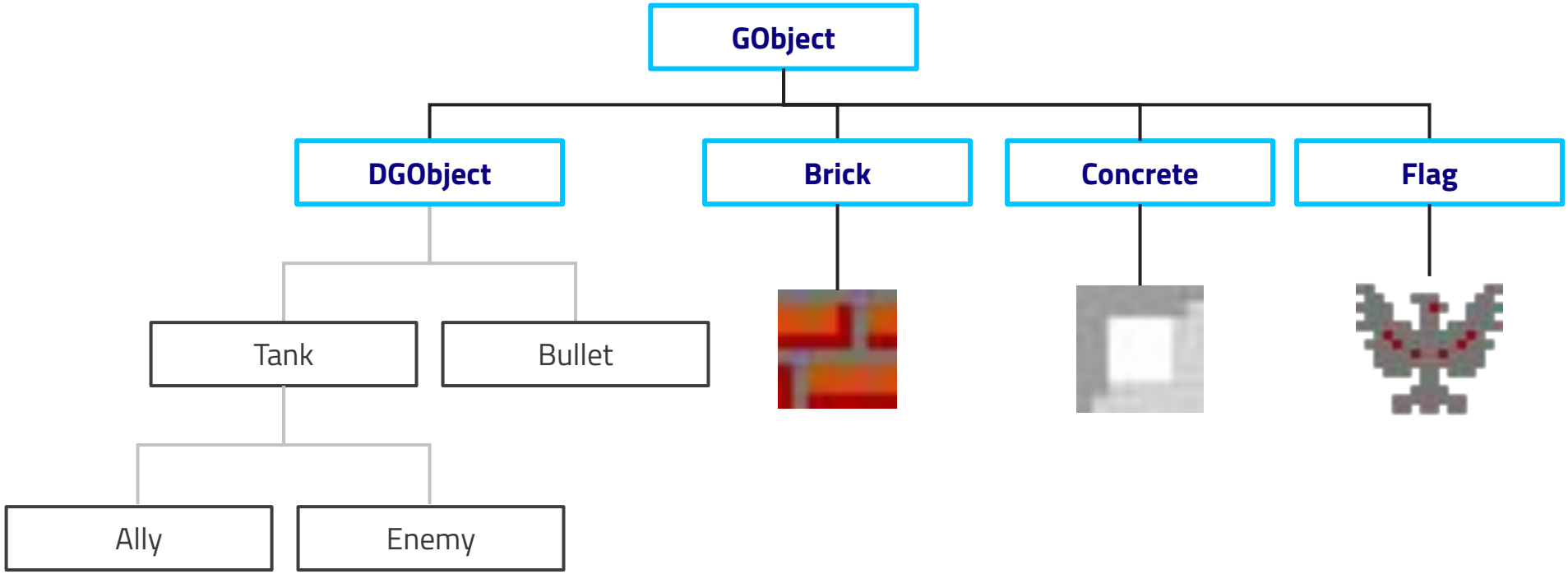
```
class Concrete: public GObject
{
public:
    Concrete(int x, int y);
    virtual ~Concrete();
};
```

```
Flag::Flag(int x, int y):
    GObject(Config::txFlag.h, Config::txFlag.w, x, y)
{
    setTexture(&Config::txFlag);
    setID(Config::FLAG);
}
```

Estructura del juego: Clase GObject



Estructura del juego: Clase GObject



Estructura del juego: Clase DGObject

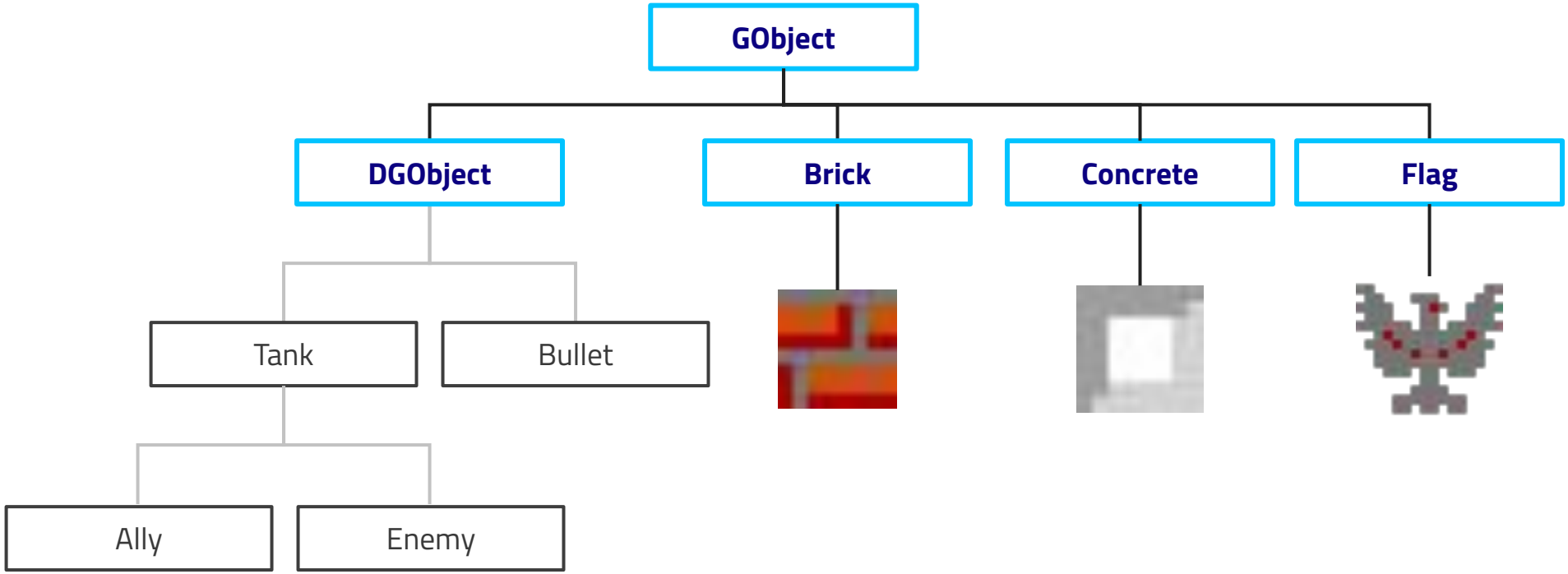
```
class DGObject: public GObject
{
public:

    enum{eUp=0, eDown=1, eLeft=2, eRight=3};

    DGObject(int h, int w, int x, int y);
    virtual ~DGObject();

    /**
     * Mueve el objeto step píxeles en la dirección predeterminada
     */
    int move(int step);

    /**
     * Mueve el objeto step píxeles y modifica la dirección
     */
    int move(int direction, int step);

    /**
     * @brief Mueve el objeto step píxeles, modifica la dirección en caso de
     * no colisionar con ningún objeto del vector obj.
     * @return -1: Si no choca con ningún objeto. En otro retorna el índice
     *         en el que se encuentra el objeto con qué colisiona.
     */
    int move(int step, vector<GObject*> &obj, int direction =-1);

    /**
     * Revisa si el objeto de la clase se intersecta con rect.
     */
    bool collide(SDL_Rect rect);

    /**
     * Revisa si rect1 y rect2 se intersectan.
     */
    bool collide(SDL_Rect rect1, SDL_Rect rect2);

protected:

    SDL_Rect textures[4]; //Orientaciones de las texturas:
                          //Arriba, abajo, izquierda, derecha
    int orientation; // Orientación actual del objeto.

private:

    /**
     * Mueve un rectángulo evitando que choque con los bordes.
     */
    static SDL_Rect moveRect(SDL_Rect rect, int step, int orientation);
};
```

Estructura del juego: Clase DGObject

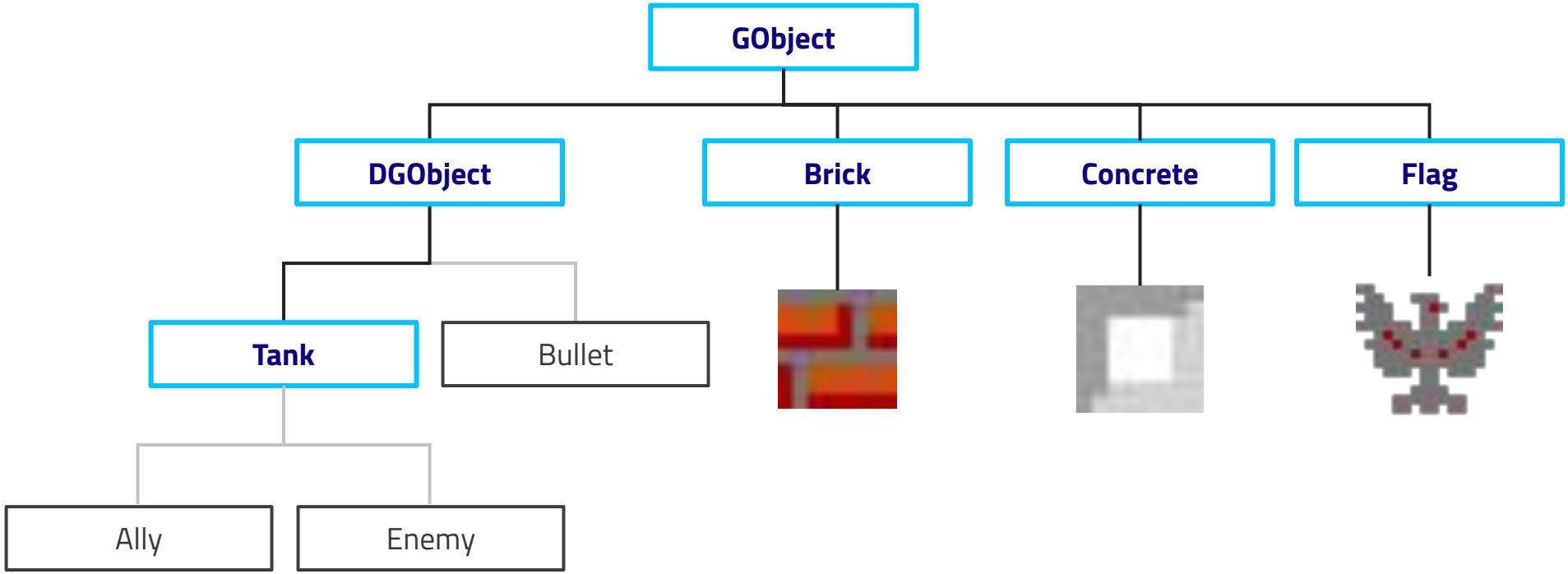
```
class DGObject: public GObject
{
public:

    enum{eUp=0, eDown=1, eLeft=2, eRight=3};

    DGObject(int h, int w, int x, int y);
    virtual ~DGObject();

    /**
     * Mueve el objeto step píxeles en la dirección predete
     */
    int move(int step);

    /**
     * Mueve el objeto step píxeles y modifica la dirección
     */
    int move(int direction, int step);

    /**
     * @brief Mueve el objeto step píxeles, modifica la di
     * no colisionar con ningún objeto del vector obj.
     * @return -1: Si no choca con ningún objeto. En otro
     *         en el que se encuentra el objeto con qué co
     */
    int move(int step, vector<GObject*> &obj, int directio

    /**
     * Revisa si el objeto de la clase se intersecta con re
     */
    bool collide(SDL_Rect rect);
```

```
int DGObject::move(int step, vector<GObject*> &obj, int dir)
{
    int r = -1;

    if(dir != orientation)
    {
        if(dir==eUp || dir==eDown || dir==eLeft || dir==eRight)
        {
            setTexture(&textures[dir]);
            orientation = dir;
        }
    }


    SDL_Rect d = moveRect(getDimension(), step, orientation);
    if(d.x != getPosition().x || d.y != getPosition().y)
    {
        for( auto it=obj.begin(); it<obj.end(); it++ )
        {
            if(collide(d, (*it)->getDimension()) && (this != (*it)))
                return it - obj.begin();
        }
        setPosition(d.x, d.y);
    }
    return r;
}
```

Estructura del juego: Clase GObject



Estructura del juego: Clase GObject



Estructura del juego: Clase Tank

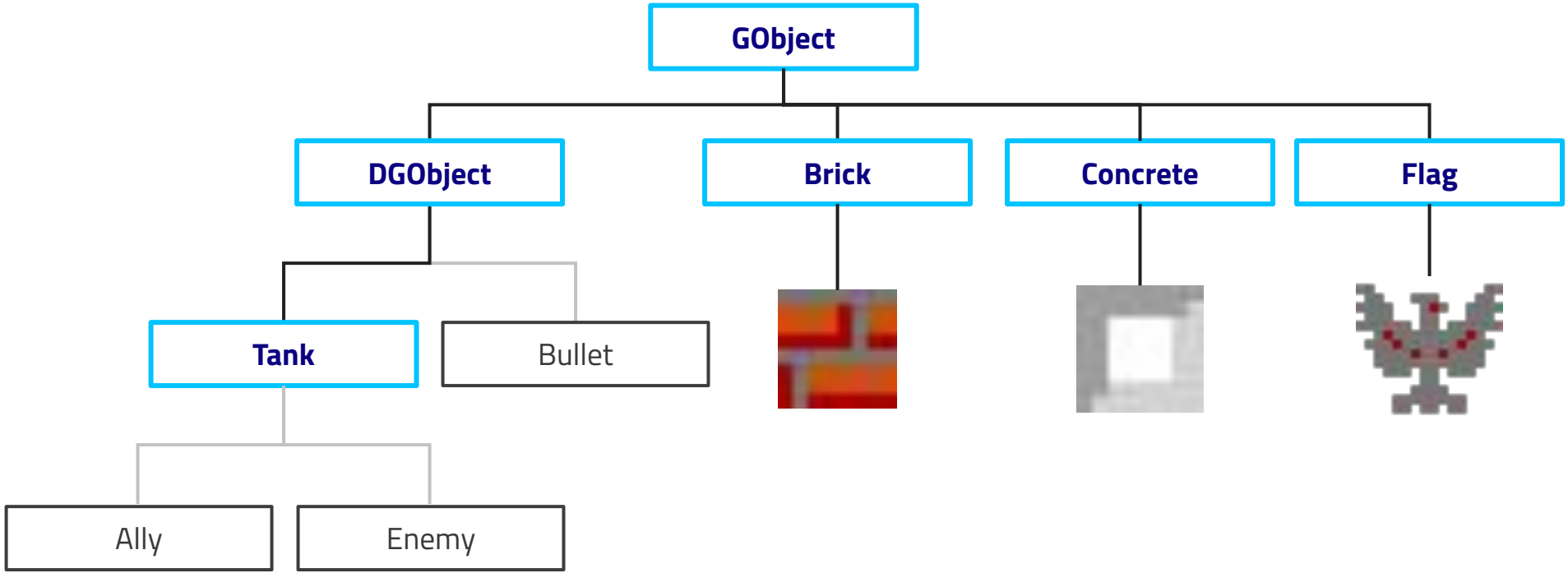
```
class Tank: public DGObject
{
public:
    Tank(size_t height, size_t width, int xPos, int yPos);
    virtual ~Tank();

    /**
     * Genera un objeto de la clase Bullet con la misma
     * orientación del objeto Tank.
     */
    Bullet* shoot();
};
```

Estructura del juego: Clase Tank

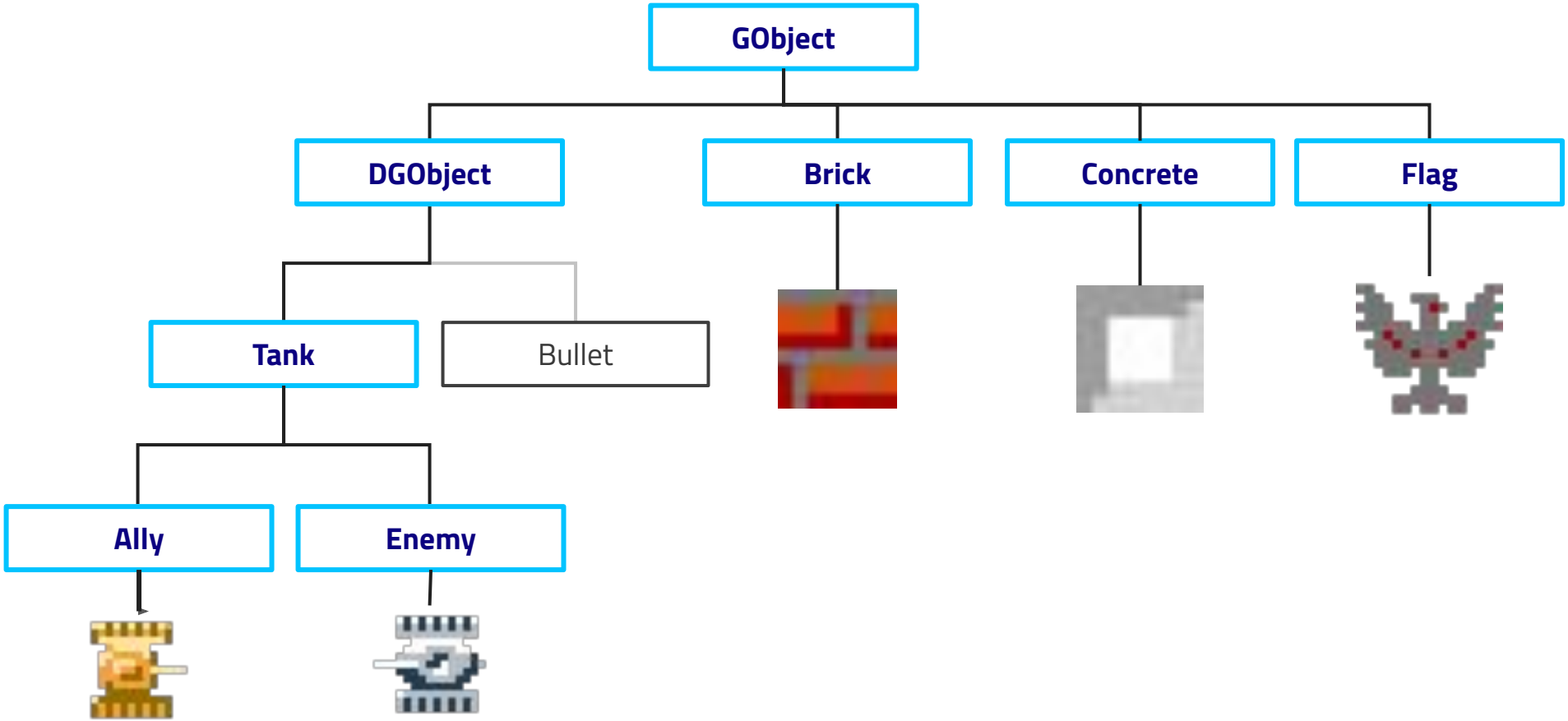
```
class Tank: public DGObject
{
public:
    Tank(size_t height, size_t width, int
    virtual ~Tank();

    /**
     * Genera un objeto de la clase Bullet
     * orientación del objeto Tank.
     */
    Bullet* shoot();
};
```

```
Bullet* Tank::shoot()
{
    Bullet *b = nullptr;
    auto d = getDimension();

    switch(orientation)
    {
    case eUp:
        b = new Bullet(d.x+12, d.y+5, eUp);
        break;
    case eDown:
        b = new Bullet(d.x+12, d.y+d.h-5, eDown);
        break;
    case eLeft:
        b = new Bullet(d.x+5, d.y+12, eLeft);
        break;
    case eRight:
        b = new Bullet(d.x+d.w-5, d.y+12, eRight);
        break;
    }

    return b;
}
```

Estructura del juego· Clase GObject



Estructura del juego· Clase GObject



Estructura del juego: Clase Tank

```
class Enemy: public Tank
{
public:
    Enemy(int x, int y);
    virtual ~Enemy(){};
    int move(int step, vector<GObject*> &obj, int direction =-1);

    Bullet* shoot();

private:
    unsigned int ticks;
    unsigned int ticksShoot;
    static bool isSeedInitilized;
};
```

```
class Ally: public Tank
{
public:
    Ally(int x, int y);
    virtual ~Ally();
};
```

Estructura del juego: Clase Enemy

```
class Enemy: public Tank
{
public:
    Enemy(int x, int y);
    virtual ~Enemy(){};
    int move(int step, ve

    Bullet* shoot();

private:
    unsigned int ticks;
    unsigned int ticksSho
    static bool isSeedIni
};
```

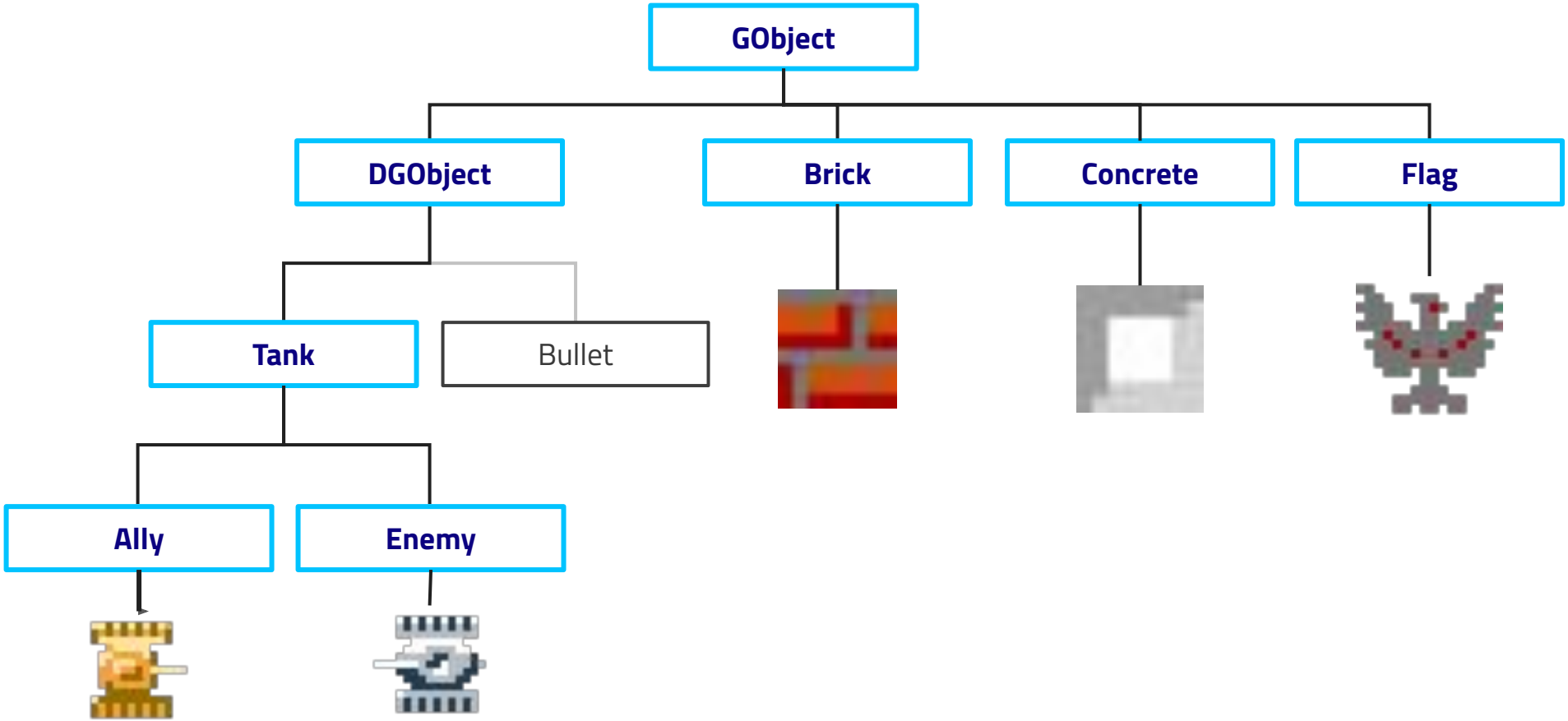
```
int Enemy::move(int step, vector<GObject*> &obj, int direction)
{
    const SDL_Point target_position={13*Config::UN,25*Config::UN};
    unsigned int cticks = SDL_GetTicks();
    float p = static_cast<float>(rand()) / RAND_MAX;

    if(cticks > ticks)
    {
        ticks = cticks + rand()%800 + 100; //Update the "timer"
        if(p < 0.8  && target_position.x > 0 && target_position.y > 0)
        {
            int dx = target_position.x - (getPosition().x + getDimension().w / 2);
            int dy = target_position.y - (getPosition().y + getDimension().h / 2);

            p = static_cast<float>(rand()) / RAND_MAX;

            if(abs(dx) > abs(dy))
                direction = p < 0.7 ? (dx < 0 ? eLeft : eRight) : (dy < 0 ? eUp : eDown);
            else
                direction = p < 0.7 ? (dy < 0 ? eUp : eDown) : (dy < 0 ? eLeft : eRight);
        }
        else
            direction = rand() % 4;
    }
    else
        direction = orientation;

    return Tank::move(step,obj,direction);
}
```

Estructura del juego: Enemy

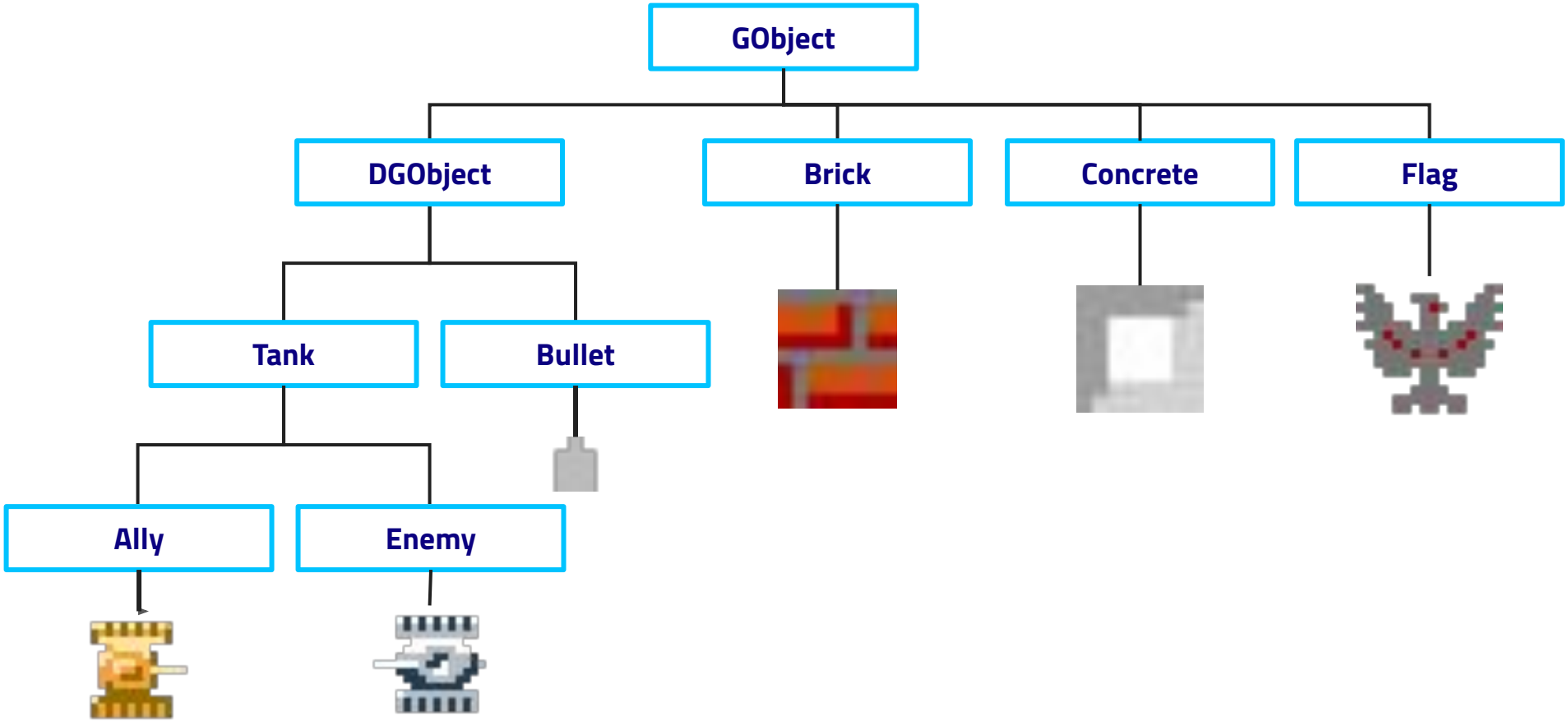
```
class Enemy: public Tank
{
public:
    Enemy(int x, int y);
    virtual ~Enemy(){};
    int move(int step, vector<

    Bullet* shoot();

private:
    unsigned int ticks;
    unsigned int ticksShoot;
    static bool isSeedInitiliz
};
```

```
Bullet* Enemy::shoot()
{
    unsigned int cticks = SDL_GetTicks();
    if(cticks > ticksShoot)
    {
        ticksShoot = cticks + rand()%2000 + 1000;
        return  Tank::shoot();
    }
    return nullptr;
}
```

Estructura del juego· Clase GObject



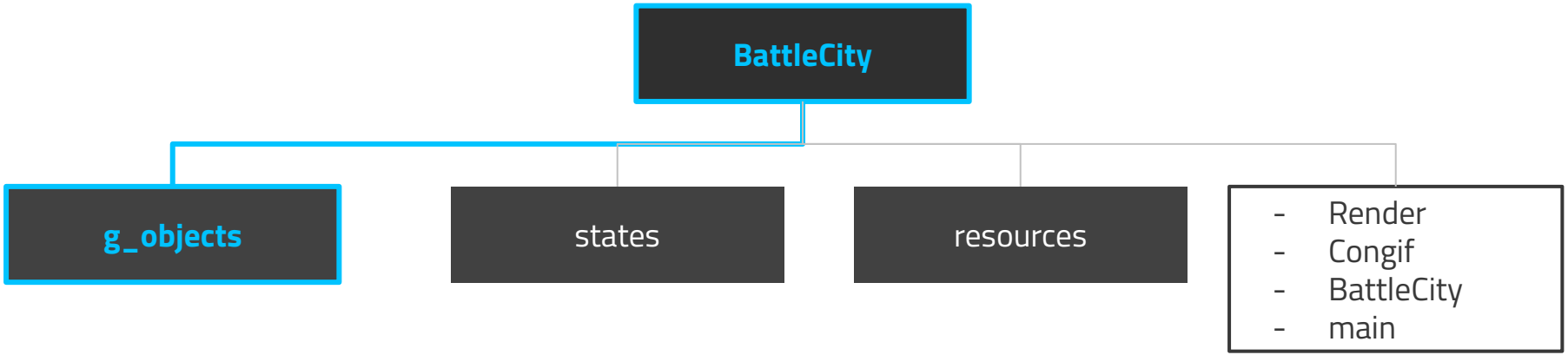
Estructura del juego· Clase GObject



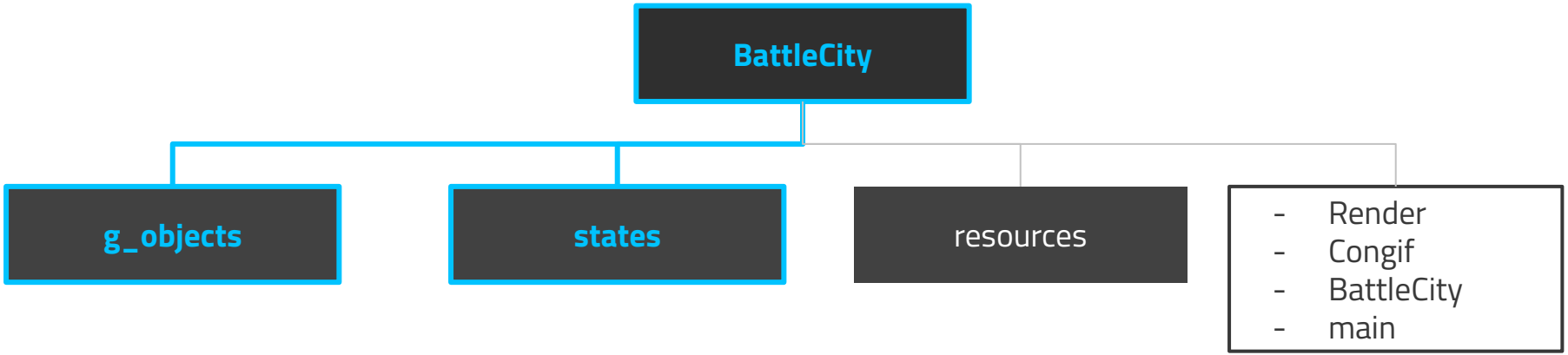
Estructura del juego: Clase Bullet

```
class Bullet: public DGObject
{
public:
    Bullet(int x, int y,int direction);
    virtual ~Bullet();
};
```

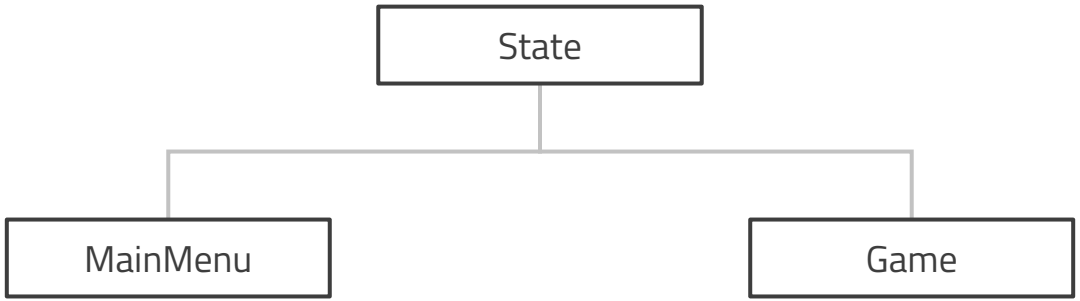
Estructura del repositorio



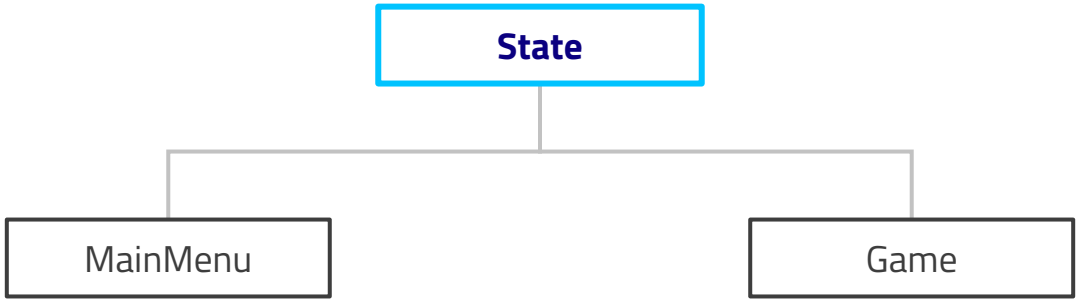
Estructura del repositorio



Estructura del juego: Clase State



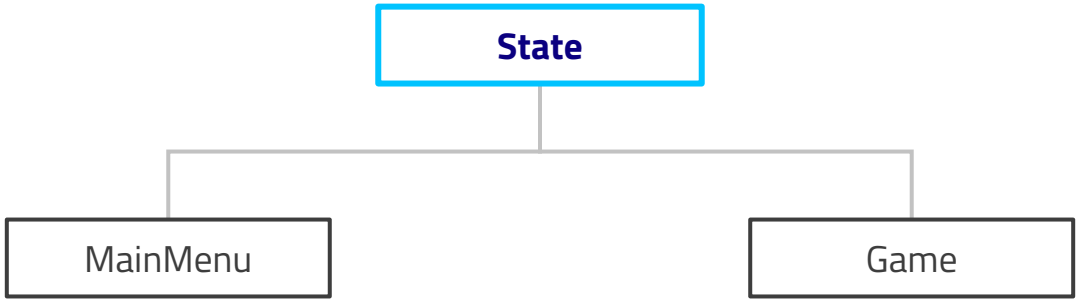
Estructura del juego: Clase State



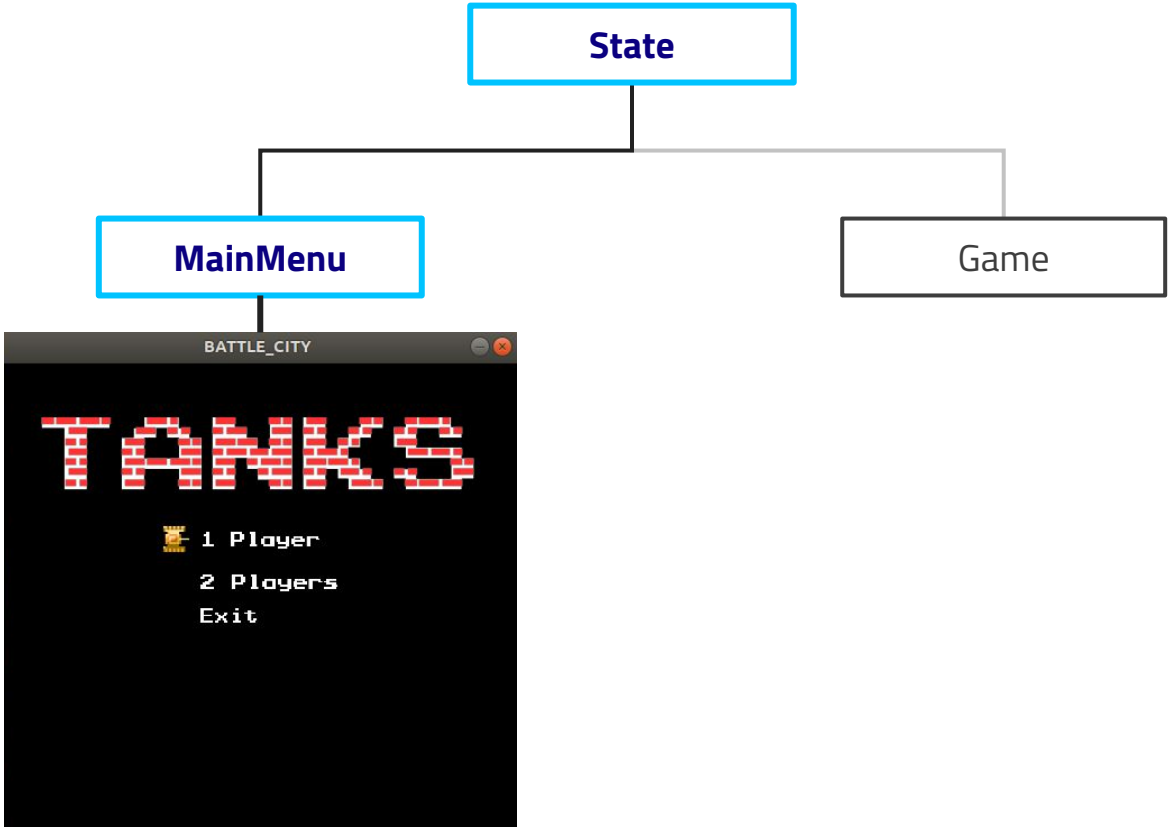
Estructura del juego: Clase State

```cpp
class State
{
public:

    virtual ~State(){};

    /**
     * Pasa el indicador de la tecla presionada.
     */
    virtual void inputKey(string key) = 0;

    /**
     * @brief Tarea del estado actual.
     * @note Debe de ser llamada periodicamente.
     */
    virtual int task() = 0;

    /**
     * Inicializa el estado
     */
    virtual bool start() = 0;

    /**
     * Detiene el estado y libera memoria
     */
    virtual bool stop() = 0;

};
```

Estructura del juego· Clase State



Estructura del juego· Clase State



Estructura del juego· Clase MainMenu

```
class MainMenu: public State
{
public:

    /**
     * Eventos que serán retornados por método task.
     */
    enum
    {
        eONE_PLAYER = 0x01,///< Modo: un jugador
        eTWO_PLAYERS,      ///< Modo: dos jugadores
        eEXIT              ///< Salir del juego
    };

    MainMenu();
    virtual ~MainMenu();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();

private:

    map<string, SDL_Texture*> texturaMainMenu;
    int yPos;
    int counter;
    string currKey="";
};
```

Estructura del juego- Clase MainMenu

```
class MainMenu: public State
{
public:

    /**
     * Eventos que serán retorna
     */
    enum
    {
        eONE_PLAYER = 0x01,///<
        eTWO_PLAYERS,      ///<
        eEXIT              ///<
    };

    MainMenu();
    virtual ~MainMenu();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();

private:

    map<string, SDL_Texture*> te
    int yPos;
    int counter;
    string currKey="";
};
```

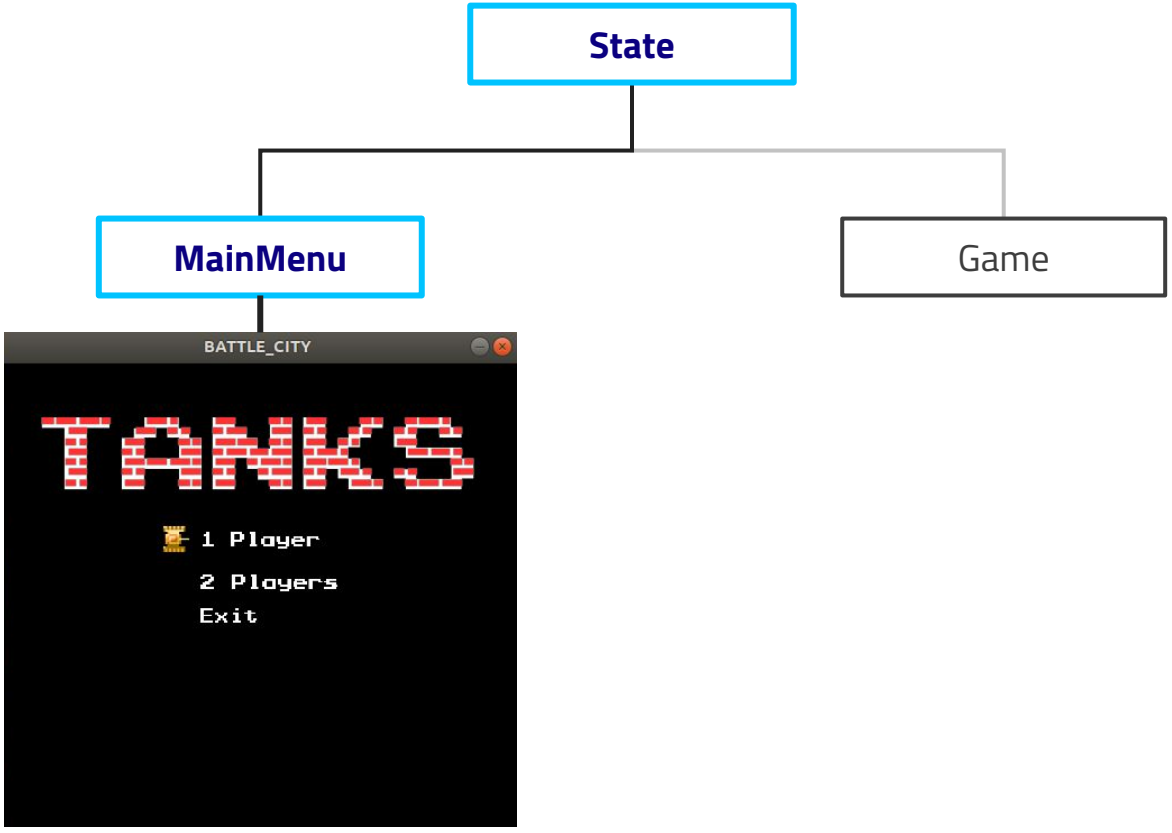
```
bool MainMenu::start()
{
    Render::drawRect(0,0, Config::SCREEN_WIDTH, Config::SCREEN_HEIGHT, Render::black, true);

    ifstream file1(Config::textureMainMenu.c_str());
    ifstream file2(Config::textureMainMenu_names.c_str());

    string str;
    vector<string> texturas;
    vector<string> texturas_names;

    while (getline(file1,str)){ texturas.push_back(str);}
    while (getline(file2,str)){ texturas_names.push_back(str);}

    for (int i = 0; i < texturas.size(); ++i)
    {
        texturaMainMenu.insert(pair<string, SDL_Texture*>(texturas_names[i], Render::loadText
    }

    Render::drawText((Config::SCREEN_WIDTH-404)/2, 48, 404, 74, Render::white, Config::font_p
    Render::drawText((Config::SCREEN_WIDTH-404)/2, 48, 404, 74, Render::red, Config::font_tan
    Render::drawText(181, 150, 108, 15, Render::white, Config::font_prstartk, 58, "1 Player")
    Render::drawText(181, 188, 124, 15, Render::white, Config::font_prstartk, 58, "2 Players"
    Render::drawText(181, 218, 52, 15, Render::white, Config::font_prstartk, 58, "Exit");
    Render::drawObject(145, yPos, 29, 29, texturaMainMenu["ally"]);

    Render::presentRender();

    return false;
}
```

Estructura del juego- Clase MainMenu

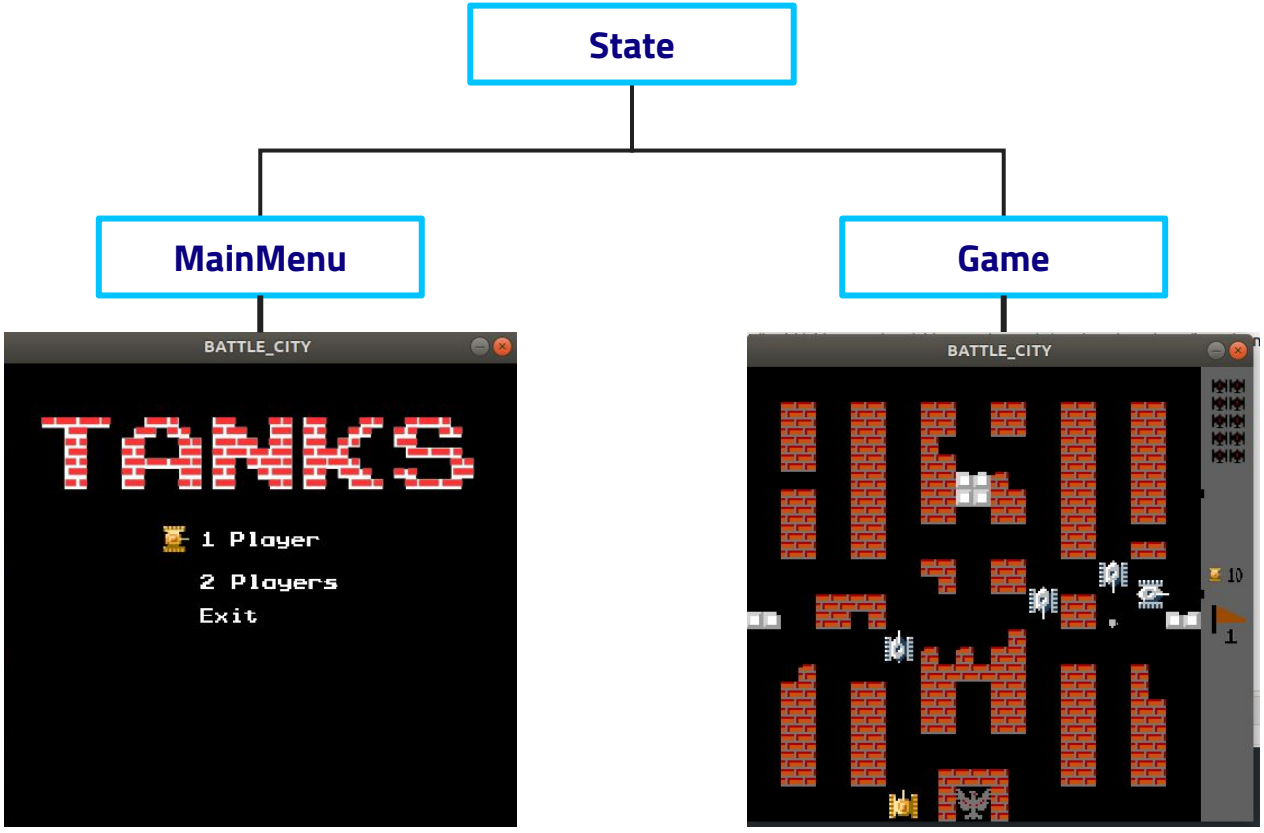
```
class MainMenu: public State
{
public:

    /**
     * Eventos que serán retornados por método
     */
    enum
    {
        eONE_PLAYER = 0x01,///< Modo: un jugad
        eTWO_PLAYERS,      ///< Modo: dos juga
        eEXIT              ///< Salir del jueg
    };

    MainMenu();
    virtual ~MainMenu();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();

private:

    map<string, SDL_Texture*> texturaMainMenu;
    int yPos;
    int counter;
    string currKey="";
};
```

```
int MainMenu::task()
{
    int result = 0;
    if (currKey=="Down" or currKey=="Up")
    {
        Render::drawRect(145, yPos, 29, 29, Render::black, true);

        if (counter%3==2) yPos = 211;
        else if (counter%3==1) yPos = 179;
        else if (counter%3==0) yPos = 142;

        Render::drawObject(145, yPos, 29, 29, texturaMainMenu["ally"]);
        Render::presentRender();
    }
    else if (currKey=="Return")
    {
        if (counter%3==0)
            result = eONE_PLAYER;
        else if (counter%3==1)
            result = eTWO_PLAYERS;
        else if (counter%3==2)
            result = eEXIT;
    }

    currKey="";
    SDL_Delay(100);
    return result;
}
```

Estructura del juego· Clase State



Estructura del juego· Clase State



Estructura del juego: Clase Game

```
class Game: public State
{
public:

    /**
     * Eventos que serán retornados por método task.
     */
    enum
    {
        eCONTINUE = 0,  //< Estado: Continua el juego
        eVICTORY = 0x05,//< Estado: Gana el juego
        eDEFEAT,        //< Estado: Pierde el juego
    };

    Game();
    virtual ~Game();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();
```

```
private:
    map<string, SDL_Texture*> texturaGame;
    vector<GObject*> objects;
    Ally *ally;
    int status;
    vector <Bullet*> bullets;
    SDL_Keycode currKey;
    vector<vector<int>> enemy_counter;
    int n_lifes;
    unsigned int enemyTicks;

    void moveAlly();
    void moveBullets();
    void moveEnemies();

    void destroyFlag();
    void destroyEnemy(int ind);
    void gameOver();
    void score();
    void drawObject(GObject &obj);
    void destroyObject(int ind);
    void createAlly();
    void destroyAlly();
};
```

Estructura del juego- Clase Game

```
class Game: public State
{
public:

    /**
     * Eventos que serán retor
     */
    enum
    {
        eCONTINUE = 0,   //< Es
        eVICTORY = 0x05,//< Es
        eDEFEAT,         //< Es
    };

    Game();
    virtual ~Game();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();
```

mapa.csv

```
while (getline(file,str)){ mapa.push_back(str);}

objects.clear();
int xPos,yPos;
for (unsigned i = 0; i < mapa.size(); ++i)
{
    for (unsigned j = 0; j < mapa[0].size(); ++j)
    {
        xPos = j*Config::UN;
        yPos = i*Config::UN;
        switch(mapa[i][j])
        {
            case 'b':
                objects.push_back(new Brick(xPos,yPos));
                Render::drawObject(objects.back()->getTexture(), xPos, yPos);
                break;
            case 'c':
                objects.push_back(new Concrete(xPos,yPos));
                Render::drawObject(objects.back()->getTexture(), xPos, yPos);
                break;
            case 'f':
                objects.push_back(new Flag(xPos,yPos));
                Render::drawObject(objects.back()->getTexture(), xPos, yPos);
                break;
            case 'e':
                Render::drawObject(xPos-6, yPos-6, Config::UN-2, Config::UN-2, texturaGame["enemy_g"]);
                enemy_counter.push_back({xPos-6, yPos-6});
                break;
            case 'a':
                Render::drawObject(xPos-10, yPos-10, Config::UN, Config::UN, texturaGame["ally"]);
                Render::drawText(xPos+8, yPos-8, 15, 15, Render::black, Config::font_prstartk, 32, to_string(n_l
                break;
            case 'g':
                Render::drawObject(xPos-6, yPos-6, 31, 26, texturaGame["flag_g"]);
                Render::drawText(xPos+4, yPos+Config::UN , 15, 15, Render::black, Config::font_prstartk, 32, "1"
                break;
```

Estructura del juego: Clase Game

```
class Game: public State
{
public:

    /**
     * Eventos que serán retornados por método task.
     */
    enum
    {
        eCONTINUE = 0,  //< Estado: Continua el juego
        eVICTORY = 0x05,//< Estado: Gana el juego
        eDEFEAT,        //< Estado: Pierde el juego
    };

    Game();
    virtual ~Game();

    void inputKey(string key);
    int task();
    bool start();
    bool stop();
```

```
void Game::inputKey(string key)
{
    if(key == "Up")
        currKey = SDLK_UP;
    else if(key == "Down")
        currKey = SDLK_DOWN;
    else if(key == "Right")
        currKey = SDLK_RIGHT;
    else if(key == "Left")
        currKey = SDLK_LEFT;
    else if(key == "Space")
        currKey = SDLK_SPACE;
}
```

```
int Game::task()
{
    moveAlly();
    moveEnemies();
    moveBullets();

    if(status == eCONTINUE)
    {
        Render::presentRender();
        SDL_Delay(70);
    }
    return status;
}
```

Estructura del juego- Clase Game

```cpp
int Game::task()
{
    moveAlly();
    moveEnemies();
    moveBullets();

    if(status == eCONTINUE)
    {
        Render::presentRender();
        SDL_Delay(70);
    }
    return status;
}
```

```cpp
void Game::moveEnemies()
{
    unsigned n_enemytemp = 0;
    for(auto obj : objects)
    {
        if(obj->getID() == Config::ENEMY)
        {
            Render::drawRect(obj->getDimension(),Render::black,true);
            Enemy *pObj = dynamic_cast<Enemy*>(obj);
            if(pObj)
            {
                pObj->move(Config::UN/4,objects);
                Bullet *b = pObj->shoot();
                if(b)
                    bullets.push_back(b);
            }
            drawObject(*obj);
            n_enemytemp += 1;
        }
    }

    unsigned int cticks = SDL_GetTicks();
    if (n_enemytemp < std::min(NPOS, enemy_counter.size()) &&
        cticks > enemyTicks)
    {
        enemyTicks = cticks + rand()%3000 + 1000;
        curEnemyPos = (curEnemyPos + 1)%NPOS;
        objects.push_back(new Enemy(EnemyPos[curEnemyPos].x, EnemyPos[curEnemyPos].y));
        drawObject(*objects.back());
    }
}
```

Estructura del juego: Clase Game

```cpp
int Game::task()
{
    moveAlly();
    moveEnemies();
    moveBullets();

    if(status == eCONTINUE)
    {
        Render::presentRender();
        SDL_Delay(70);
    }
    return status;
}
```

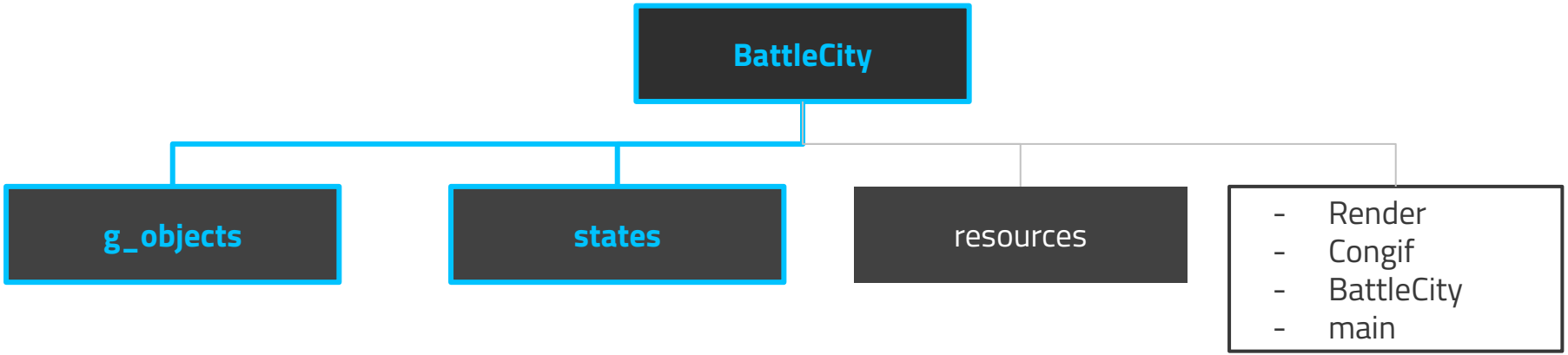
```cpp
void Game::moveBullets()
{
    for(auto bIt = bullets.begin(); bIt < bullets.end(); bIt++)
    {
        auto &b = (*bIt);

        SDL_Point oldPos = b->getPosition();
        Render::drawRect(b->getDimension(), Render::black, true);
        int ind = b->move(Config::UN,objects);
        if( ind >= 0)
        {
            switch(objects[ind]->getID())
            {
            case Config::CONCRETE:
                break;
            case Config::ALLY:
                destroyAlly();
                break;
            case Config::ENEMY:
                destroyEnemy(ind);
                break;
            case Config::FLAG:
                destroyFlag();
                break;
            default:
                destroyObject(ind);
            }
        }

        if(oldPos.x == b->getPosition().x && oldPos.y == b->getPosition().y)
        {
            delete b;
            bullets.erase(bIt);
            continue;
        }
        drawObject(*b);
    }
```

Estructura del juego: Clase Game

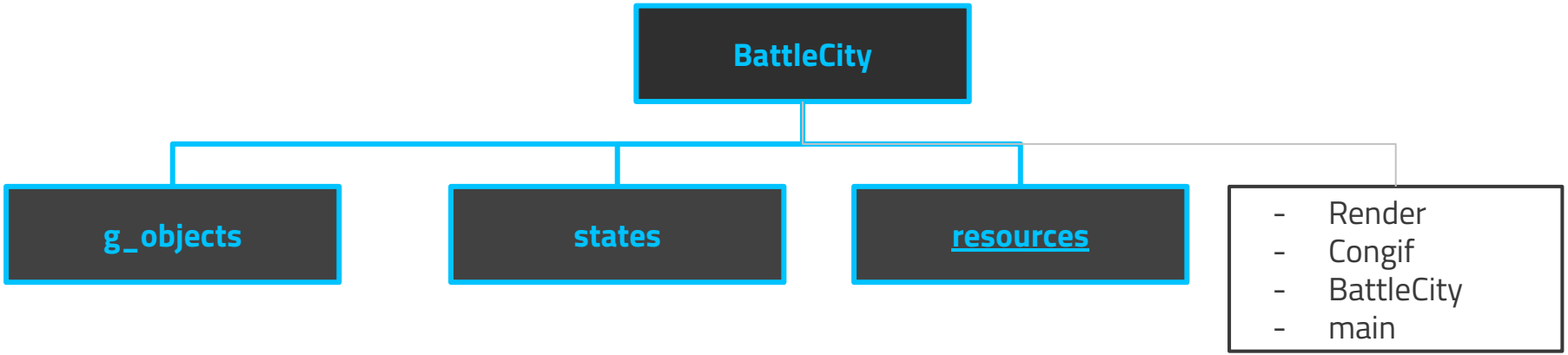
```
void Game::destroyFlag()
{
    for(auto it=objects.begin(); it<objects.end(); it++)
    {
        if((*it)->getID() == Config::FLAG)
        {
            Render::drawRect((*it)->getDimension(), Render::black, true);
            delete (*it); //free memory
            objects.erase(it); //Remove the ally from the obstacles vector
            Render::drawObject(&Config::txDestroyedFlag, 192, 384);
            Render::presentRender();
            SDL_Delay(2000);
            gameOver();
            status = eDEFEAT;
            return;
        }
    }
}
```

```
void Game::moveBullets()
{
    for(auto bIt = bullets.begin(); bIt < bullets.end(); bIt++)
    {
        auto &b = (*bIt);

        SDL_Point oldPos = b->getPosition();
        Render::drawRect(b->getDimension(), Render::black, true);
        int ind = b->move(Config::UN,objects);
        if( ind >= 0)
        {
            switch(objects[ind]->getID())
            {
            case Config::CONCRETE:
                break;
            case Config::ALLY:
                destroyAlly();
                break;
            case Config::ENEMY:
                destroyEnemy(ind);
                break;
            case Config::FLAG:
                destroyFlag();
                break;
            default:
                destroyObject(ind);
            }
        }

        if(oldPos.x == b->getPosition().x && oldPos.y == b->getPosition().y)
        {
            delete b;
            bullets.erase(bIt);
            continue;
        }
        drawObject(*b);
```

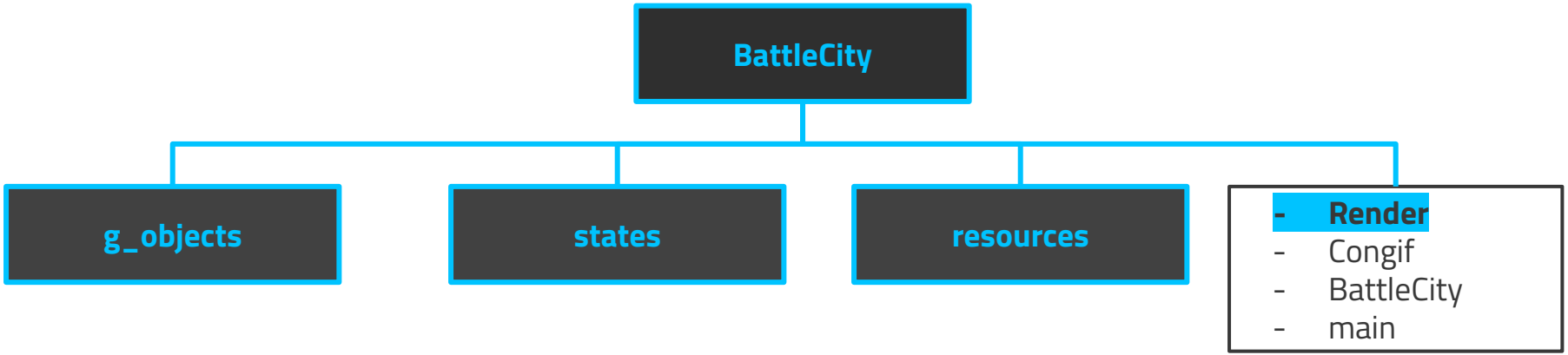
Estructura del repositorio



Estructura del repositorio



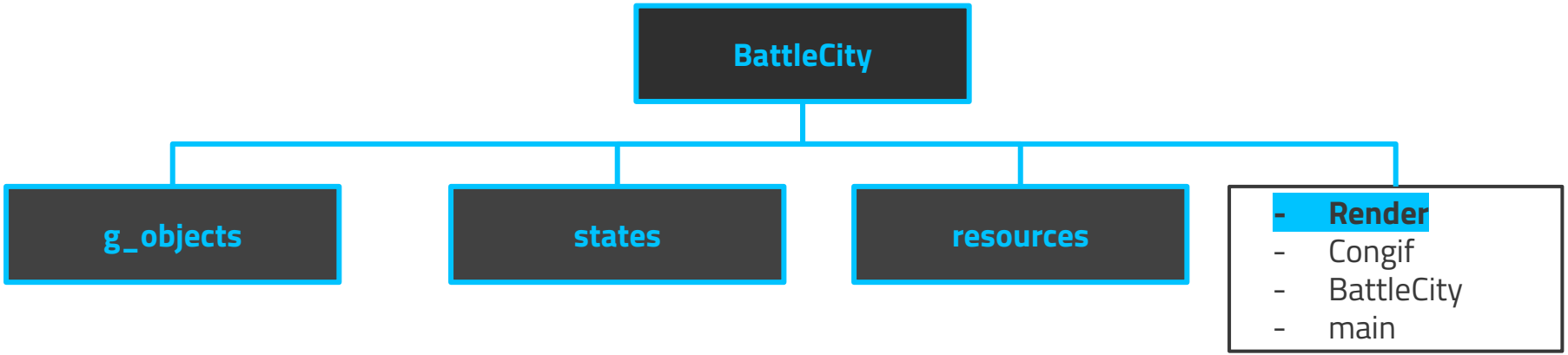
Estructura del repositorio



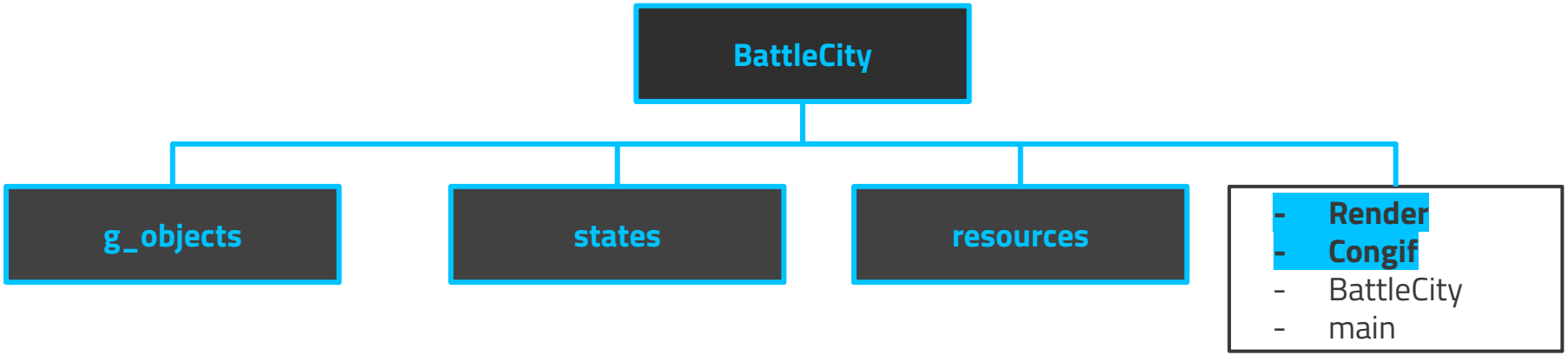
Estructura del juego: Clase Render

```
class Render
{
public:
    Render(SDL_Window *);
    virtual ~Render();

    static void init(SDL_Window *);
    static SDL_Texture* loadTexture(std::string path);
    static void drawObject(int ObjectXPosition, int ObjectYPosition, int ObjectWidth, int ObjectHei
    static void drawObject(const SDL_Rect *source, int x, int y);
    static void drawText(int TextXPosition, int TextYPosition, int TextWidth, int TextHeight, SDL_C
    static void drawRect(int RectangleXPosition, int RectangleYPosition, int RectangleWidth, int Re
    static void drawRect(SDL_Rect rect, SDL_Color RectangleColor, bool Filled);
    static void presentRender();

    static void close();

    static const SDL_Color red;
    static const SDL_Color white;
    static const SDL_Color black;
    static const SDL_Color gray;

private:
    static SDL_Renderer *render;
    static SDL_Texture *texture;
};
```

Estructura del repositorio



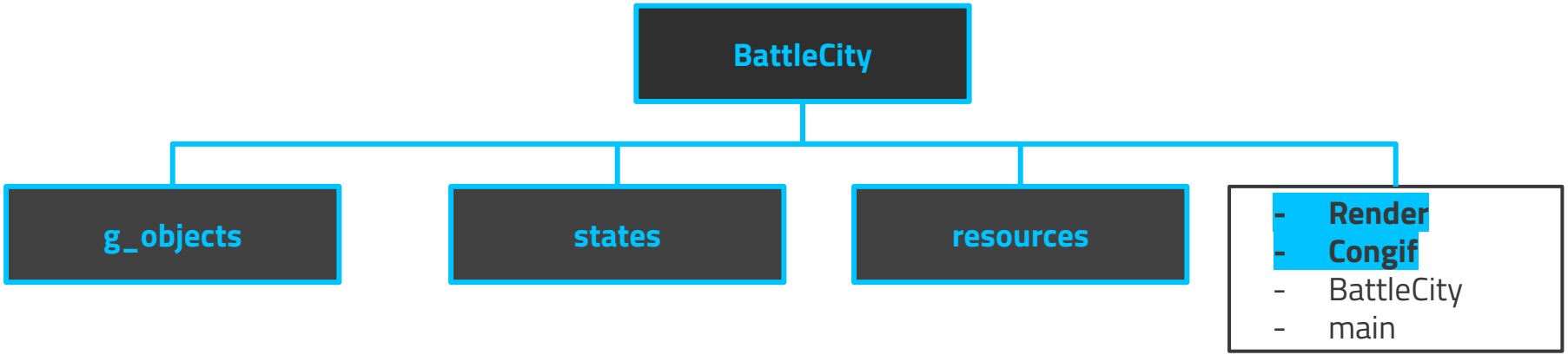
Estructura del repositorio



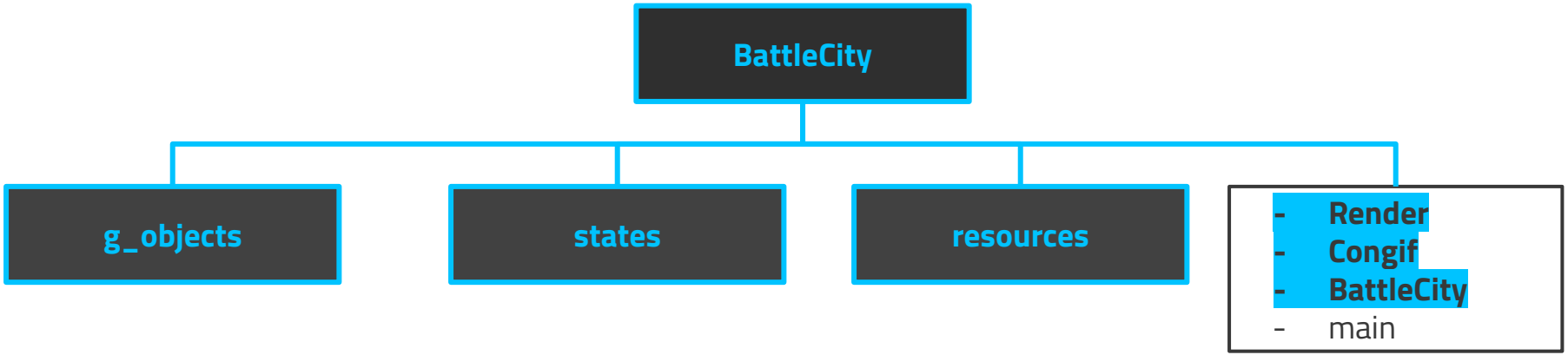
Estructura del juego- Clase Config

```
class Config{

public:

    static const int UN;
    static const int SCREEN_WIDTH;
    static const int SCREEN_HEIGHT;
    static const string textureMainMenu;
    static const string textureMainMenu_names;
    static const string textureGame;
    static const string textureGame_names;
    static const string mapLevel1;
    static const string font_tank;
    static const string font_prstartk;
    static const string path_global_texture;

    /* Región donde se encuentran las texturas de los objetos */
    static const SDL_Rect txBrick;
    static const SDL_Rect txConcrete;
    static const SDL_Rect txFlag;
    static const SDL_Rect txDestroyedFlag;
    static const SDL_Rect txAlly[];
    static const SDL_Rect txEnemy[];
    static const SDL_Rect txBullet[];

    /* ID de los objetos */
    enum {BRICK=1,CONCRETE,FLAG,BULLET,ENEMY,ALLY};
};
```

Estructura del repositorio



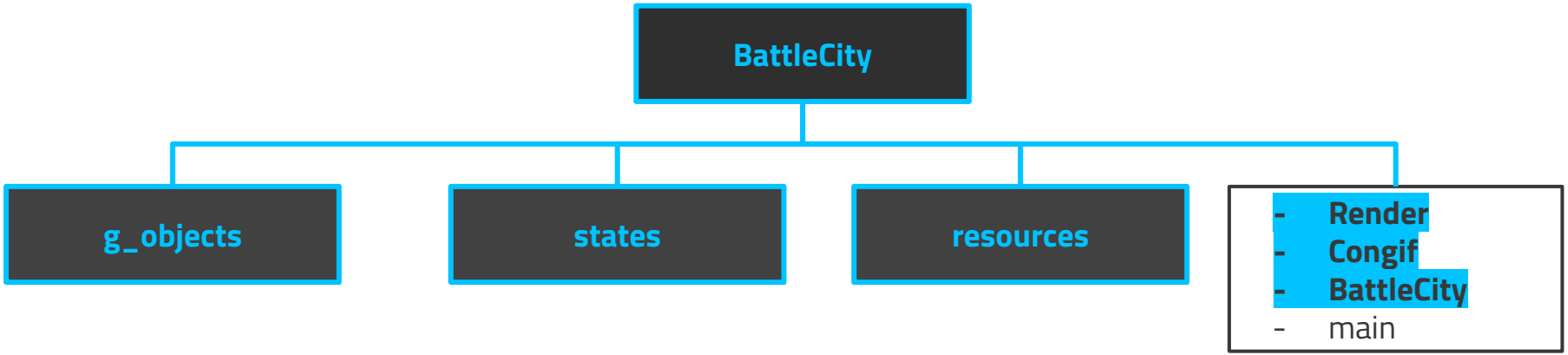
Estructura del repositorio



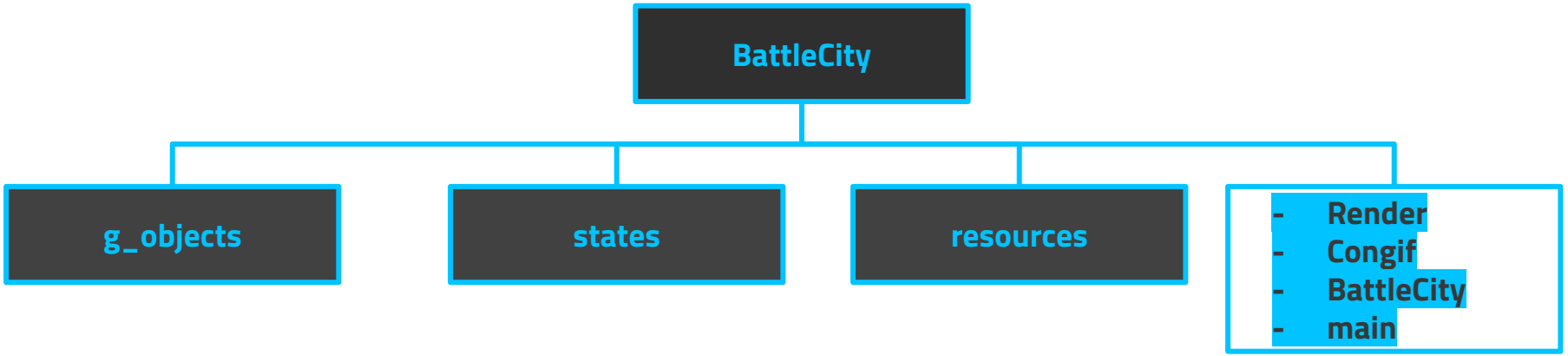
Estructura del juego: Clase BattleCity

```
class BattleCity
{
public :

  BattleCity();
  ~BattleCity();

  /**
   * Inicia y controla el estado del juego y los eventos del teclado.
   */
  void start();

private :

  State *currState; //Estado actual
  SDL_Window *window; // Ventana del juego

  /**
   * Inicia las librerias de SDL y la ventana
   */
  bool init();

};
```

Estructura del repositorio



Estructura del repositorio



Estructura del juego: main

```
/**
 * @brief Se prueba Battle City
 */
#include "battlecity.h"

int main()
{
  BattleCity myGame;

  myGame.start(); // Se empieza el juego

  return 0;
}
```

Referencias

- https://lazyfoo.net/tutorials/SDL/index.php
- https://github.com/krystiankaluzny/Tanks
- https://en.wikipedia.org/wiki/Battle_City
- https://www.libsdl.org/

Gracias