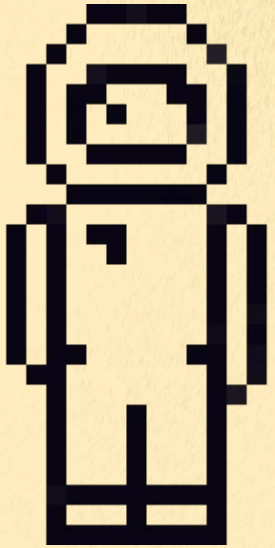
DiveUP



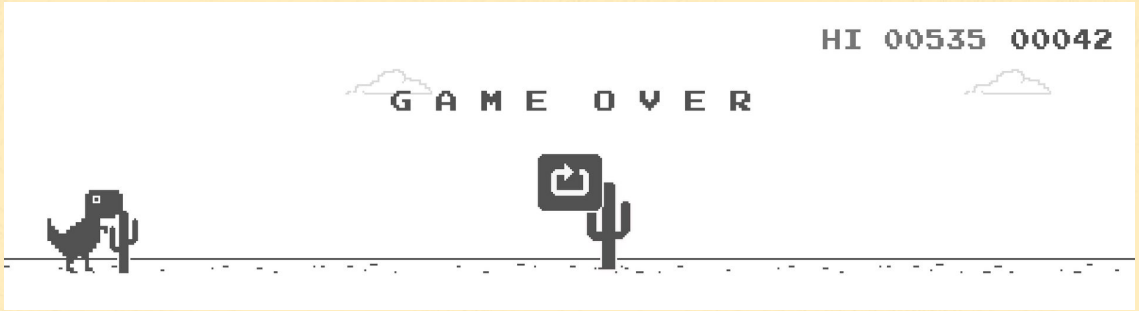
Laura Sofía Arango
Cristian Serna

Indice

1. Motivación
2. Demo
   a. Demostración
   b. Compilación
3. Estructura
4. Estados e interfaz
5. Objetos

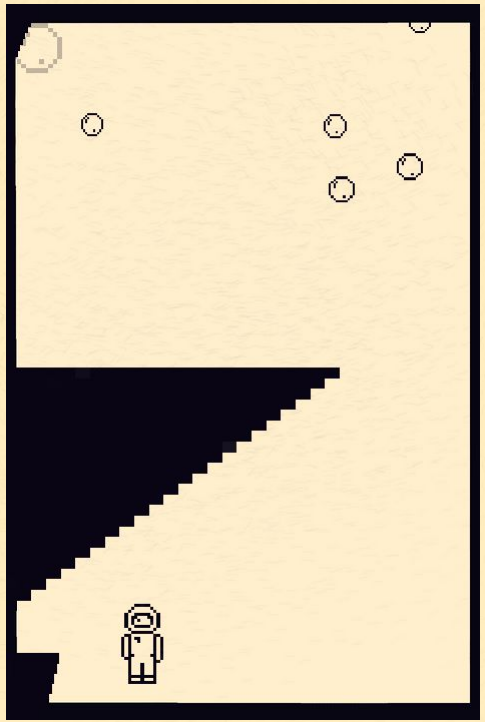
Motivación

Juegos de plataformas



Demo

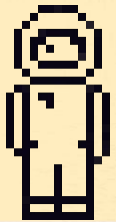
DiveUP en ejecución



Compilación

Dependencias e instrucciones

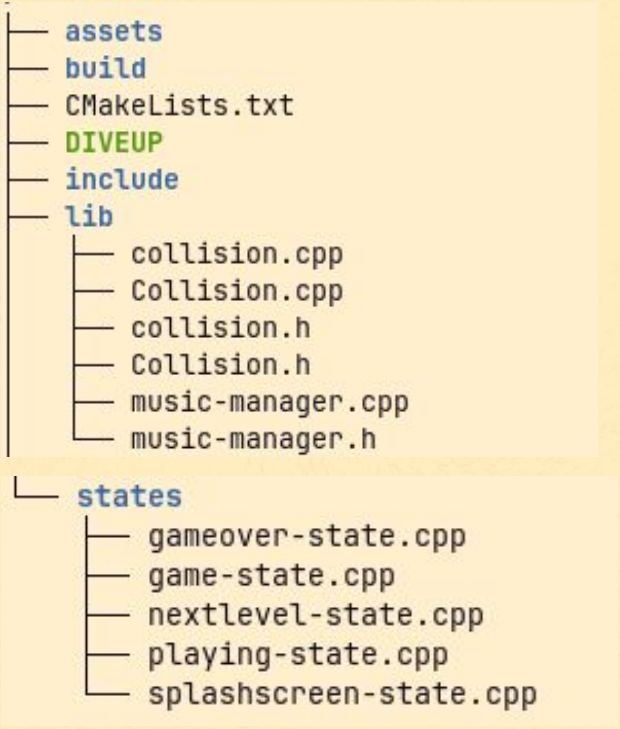
1. Clonar el repositorio https://github.com/KaboomPhysicist/diveup
2. Instalar SFML
   ```
   sudo apt-get install libsfml-dev
   ```
3. Compilar:
   a. `mkdir build`
   b. `cd build && cmake ..`
   c. `cd .. && cmake --build build`
4. `./DIVEUP`

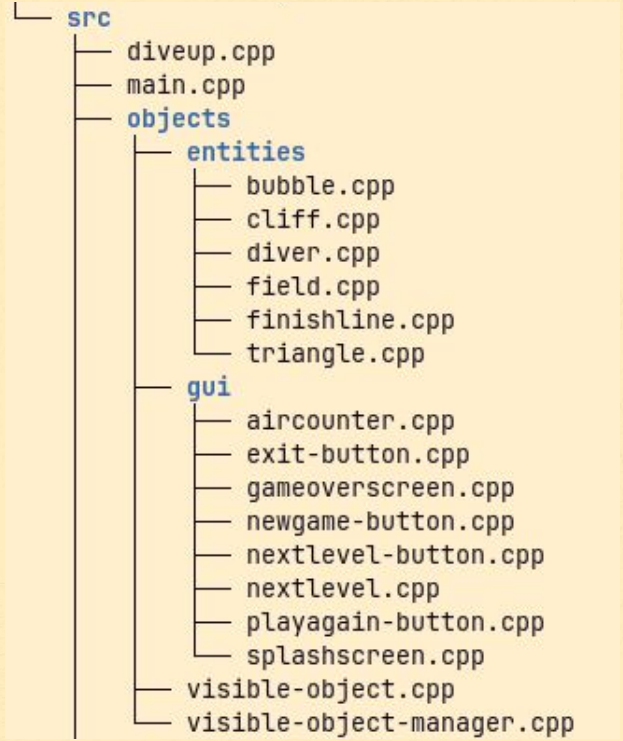


Proyecto basado en:
https://medium.com/achiev/game-from-scratch-with-c-and-sfml-1-f17dcc2b6092

Estructura

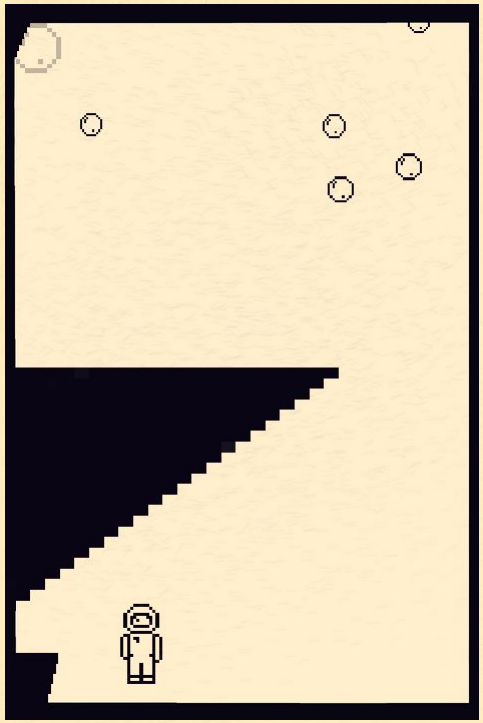
Estructura general del proyecto





Estructura

Composición de un estado



Estado

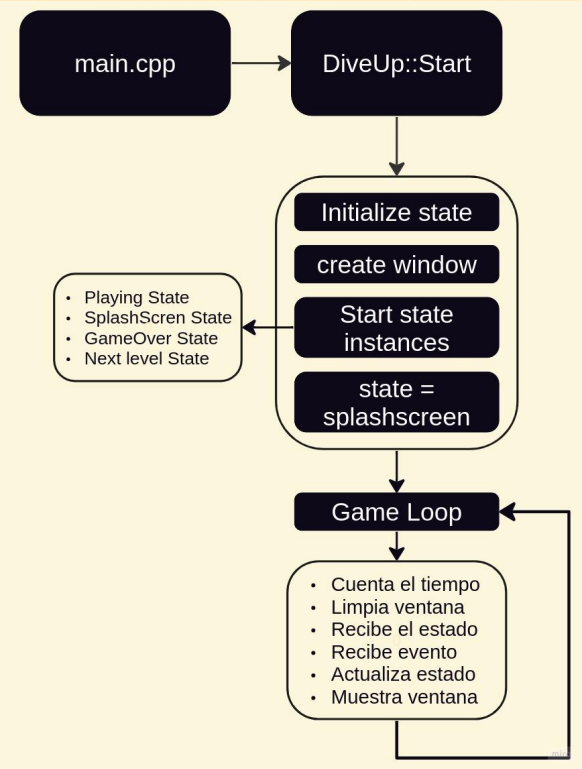
- SplashScreen
- Playing
- NextLevel
- GameOver

Objetos:

- Jugador
- Riscos
- Burbujas
- Contador
- Fondo
- Final
- Triángulo

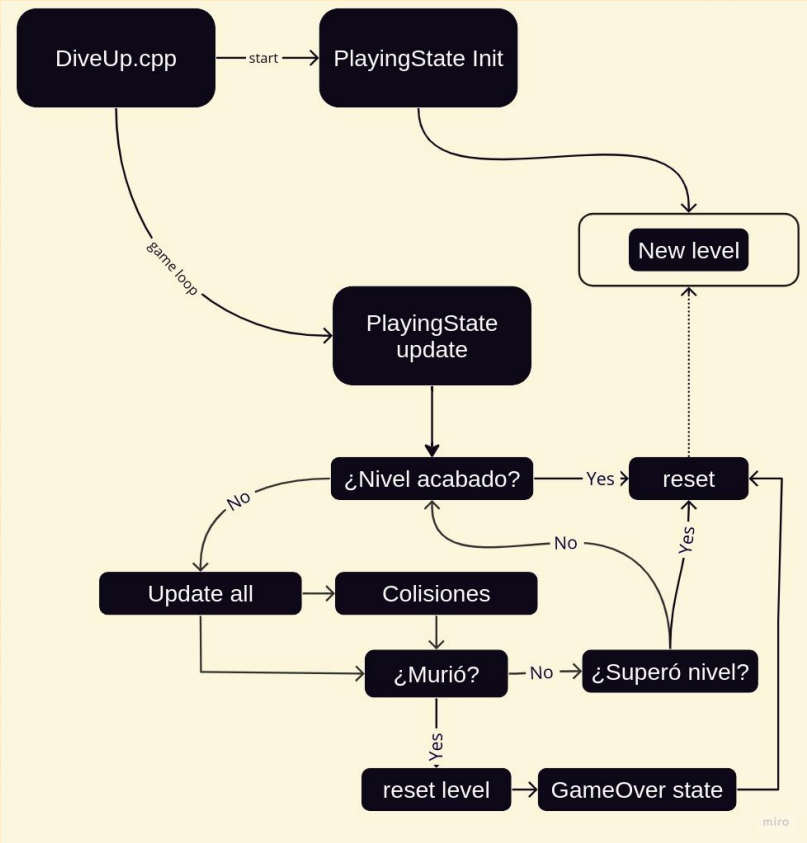
Estructura

Diagrama de flujo del juego



Estructura

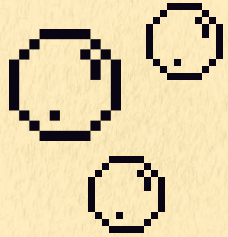
Playing state



Estados e interfaz

Playing State

```
1  class PlayingState : public GameState{
2      public:
3          void init() override;
4          void handleInput(sf::Event *event) override;
5          void update(float timeElapsed) override;
6          void draw(sf::RenderWindow *window) override;
7          void BubblePopulation();
8          ~PlayingState() override;
9
10         void newLevel();
11         void initAirmarker(int, int);
12         void opacityupdate(int);
13
14         void reset();
15         void setEnded(bool);
16         void resetLevel();
```



Estructura

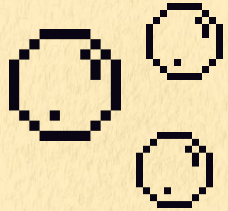
Objetos: Visible object class

```
class VisibleObject {
    public:
      VisibleObject(std::string textureFilename);
      virtual ~VisibleObject();
      virtual void handleInput(sf::Event *event);
      virtual void update(float timeElapsed);
      virtual void draw(sf::RenderWindow *window);

      virtual void collideWith(VisibleObject *target);

      virtual void move(float x, float y);
      virtual void setPosition(float x, float y);
      virtual sf::Vector2<float> getPosition();

      virtual float getTop();
      virtual float getBottom();
      virtual float getLeft();
      virtual float getRight();
```

```
 virtual sf::Rect<float> getBoundingRect();

  virtual void setPriority(size_t priority);
  virtual size_t getPriority();

  virtual sf::Sprite getSprite();

protected:
    sf::Sprite _sprite;
    sf::Texture _texture;
    bool _isLoaded;

private:
  size_t _priority;
};
```

Estructura

Objetos: Visible object manager



```cpp
void VisibleObjectManager::updateAll(float timeElapsed) {
  auto itr = _objects.begin();
  while (itr != _objects.end()) {
    itr->second->update(timeElapsed);
    itr++;
  }  // Detect collision by bounding rect.
  auto originItr = _objects.begin();  while (originItr != _objects.end()) {
  sf::Rect<float> originBound = originItr->second->getBoundingRect();
  auto targetItr = _objects.begin();    while (targetItr != _objects.end()) {
    if (targetItr == originItr) { targetItr++; continue; }

      sf::Rect<float> targetBound = targetItr->second->getBoundingRect();

      if (originBound.intersects(targetBound)) {
        if(colision::checkCollision(originItr->second, targetItr->second)){
          originItr->second->collideWith(targetItr->second);
        }
      }
      targetItr++;
    }
    originItr++;
  }
```

```cpp
class VisibleObjectManager {
public:
  ~VisibleObjectManager();
  void add(std::string name, VisibleObject *object);
  void remove(std::string name);
  VisibleObject *get(std::string name);

  void handleInputAll(sf::Event *event);
  void updateAll(float timeElapsed);
  void drawAll(sf::RenderWindow *window);

private:
  std::map<std::string, VisibleObject*> _objects;
};
```

Objetos

Objetos: Diver

```cpp
void Diver::handleInput(sf::Event *event) {
    if (event->type == sf::Event::KeyPressed) {
        if (event->key.code == sf::Keyboard::Left) {
            _direction = Left;
            this->Set_texture("assets/DEBUG-diver-left.png");
        } else if (event->key.code == sf::Keyboard::Right) {
            _direction = Right;
            this->Set_texture("assets/DEBUG-diver-right.png");
        }
    } else if (event->type == sf::Event::KeyReleased) {
            _direction = NONE;
            this->Set_texture("assets/stand-diver3.png");
        }
    }
```

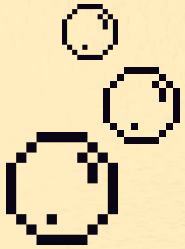
```cpp
void Diver::update(float timeElapsed) {

    float x_velocity = 0.0f;

    if (_direction == Left) {x_velocity = _speed * -1;}
    else if (_direction == Right) {x_velocity = _speed;}

    if(_finishing){y_velocity = -400;}
    else y_velocity = 0;

    move(x_velocity * timeElapsed, y_velocity * timeElapsed);
```

```cpp
# pragma once

#include "objects/visible-object.h"
#include "objects/entities/bubble.h"

class Diver : public VisibleObject {
    public:
        Diver(float constraintLeft, float constraintRight);
        void handleInput(sf::Event *event);
        void update(float timeElapsed);
        void collideWith(VisibleObject *target);

        void Set_texture(std::string textureFilename);
        void setFinishing(bool);

        float getOxygen();
        void setOxygen(float);


    private:
        enum Direction { Left, Right ,NONE};
        Direction _direction = NONE;
        float _speed = 100.0f;
        float _constraintLeft;
        float _constraintRight;
        bool _finishing;
        float _oxygen;
        float y_velocity;
};
```

Objetos

Objetos: Cliff

```
class Cliff: public VisibleObject {
public:
    Cliff(int,int,float,float);
    void update(float timeElapsed);
    void handleInput(sf::Event *event);
    void scaleCliff(float, float);
    static void verifyCliffs(std::vector<Cliff*> &, Cliff& );
    static void generateCliffs(std::vector<Cliff*> &cliffs,
     float velocity, int cliffsMax);
    static void verifySpace(std::vector<Cliff*> &_cliffs,
    Cliff &cliff);

    void collideWith(VisibleObject *target);

    private:
        float velocity;
};
```



```
void Cliff::generateCliffs(std::vector<Cliff*> &_cliffs, float velocity,int cliffsMax){

    int _cliffsMax = cliffsMax;
    for (int i=0; i< _cliffsMax ;i++){

        std::random_device rd;
        std::mt19937 gen(rd());

        std::normal_distribution<float> ypos(0, 50);
        std::normal_distribution<float> size(100, 50);

        float direction = (rand() % 2 == 0);
        float altura = ypos(gen);

        std::vector<float> _cliffPos= {direction*DiveUp::SCREEN_WIDTH, altura};

        int sizex = size(gen)+100;
        int sizey = size(gen)+100;

        Cliff *cliff = new Cliff(sizex, sizey, direction, velocity);

        cliff->setPosition(_cliffPos.at(0), _cliffPos.at(1));

        if (i > 0){
            verifyCliffs(_cliffs, *cliff);
            verifySpace(_cliffs, *cliff);}

        _cliffs.push_back(cliff);

    }
    //the lowest cliff must be over 500, so let's move all cliffs up
    for (Cliff *i : _cliffs){ i->setPosition(i->getPosition().x, i->getPosition().y-500); }
}
```

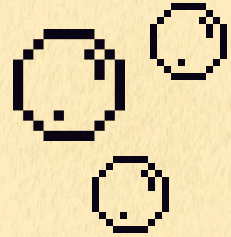
Objetos

Objetos: Bubble

```
1  void Bubble::GenerateBubble(short int index, float velocity_factor, float velocity_bias,
2   sf::Rect<float> constraints, std::vector<int> SCREEN_RANGE, std::vector<Bubble*>& _bubbles,
3                                                 VisibleObjectManager& visibleObjectManager){
4      // Generate random position for bubble
5
6      std::random_device rd;
7      std::mt19937 gen(rd());
8
9      std::uniform_int_distribution<> disx(SCREEN_RANGE.at(0), SCREEN_RANGE.at(1));
10     std::uniform_int_distribution<> disy(SCREEN_RANGE.at(2), SCREEN_RANGE.at(3));
11
12     int x = disx(gen);
13     int y = disy(gen);
14
15     _bubbles.at(index) = new Bubble(constraints, velocity_factor, velocity_bias);
16
17     _bubbles.at(index)->setPosition(x, y);
18     _bubbles.at(index)->setPriority(1);
19
20     std::ostringstream bubbleName;
21     bubbleName << "bubble" << index;
22
23     visibleObjectManager.add(bubbleName.str(), _bubbles.at(index));
24 }
```

Objetos

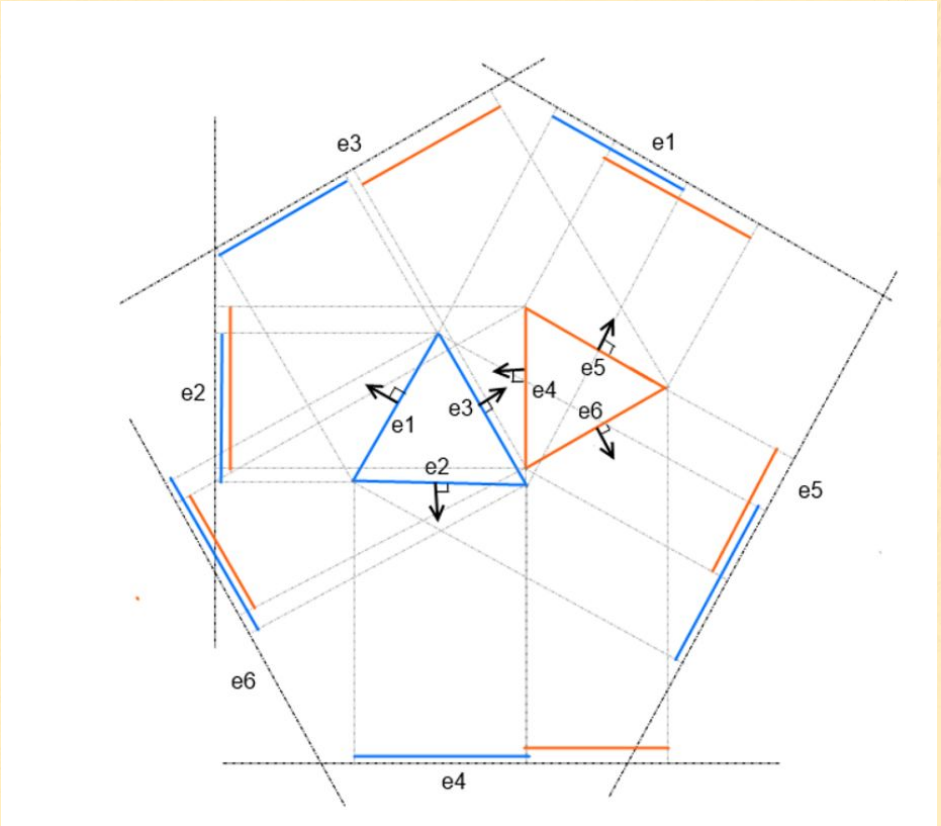
Objetos: FinishLine



```cpp
FinishLine::FinishLine(int position, float speed) : VisibleObject("assets/finishline.png") {
    this->setPosition(0, position);
    velocity = speed;
}

void FinishLine::collideWith(VisibleObject *target) {
    if(!dynamic_cast<Diver*>(target) ) return;
    std::cout << "Llegaste a la meta" << std::endl;
    dynamic_cast<PlayingState*>(DiveUp::getState())->setEnded(true);
    DiveUp::setState(DiveUp::NextLevel);
};

void FinishLine::update(float timeElapsed) {
    //std::cout << "Line position:" << this->getPosition().y << std::endl;
    if(_finishing){
        this->move(0, 0);
    }
    else{
        this->move(0, velocity * timeElapsed);
    }
}
```

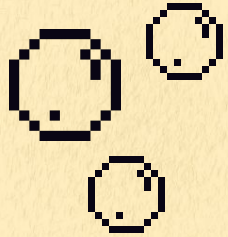
Objetos

Colisiones y el teorema de ejes de separación



Objetos

Collision box



```cpp
Triangle::Triangle(const sf::Vector2f& vertex1, const sf::Vector2f& vertex2,
                   const sf::Vector2f& vertex3, const sf::Vector2f& position) {
    vertices[0] = vertex1;
    vertices[1] = vertex2;
    vertices[2] = vertex3;
    this->position = position;
}

    // Calculate the axes perpendicular to each edge
std::vector<sf::Vector2f> Triangle::calculateAxes() {
    std::vector<sf::Vector2f> axes;

    for (int i = 0; i < 3; i++) {
        sf::Vector2f edge = vertices[(i + 1) % 3] - vertices[i];
        float norm = std::sqrt(edge.x * edge.x + edge.y * edge.y);
        sf::Vector2f axis(-edge.y/norm, edge.x/norm);  // Perpendicular axis
        axes.push_back(axis);
    }

    return axes;
}
```

```cpp
sf::Vector2f Triangle::projectOntoAxis(const sf::Vector2f& axis) {
    float minProjection = std::numeric_limits<float>::max();
    float maxProjection = -std::numeric_limits<float>::max();


    for (int i = 0; i < 3; i++) {
        //std::cout << "Vertices: " << vertices[i].x << " " << vertices[i].y << std::endl;
        float projection = axis.x * (vertices[i].x) + axis.y * (vertices[i].y);
        minProjection = std::min(minProjection, projection);
        maxProjection = std::max(maxProjection, projection);
    }

    //std::cout << "min: " << minProjection << " max: " << maxProjection << std::endl;

    return sf::Vector2f(minProjection, maxProjection);
}

```

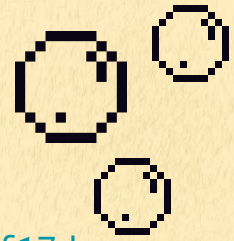
Objetos

Música

```
1  class Musicmanager  {
2      public:
3          Musicmanager();
4          void startMusic();
5          void stopMusic();
6          void selectMusic();
7          sf::Music music;
8  };
```

```
sf::Music music;
if (!music.openFromFile("music.ogg"))
    return -1; // error
music.play();
```

Referencias



- Código basado en:
  - https://medium.com/achiev/game-from-scratch-with-c-and-sfml-1-f17dcc2b6092
- Música sacada de:
  - https://pixabay.com/music/upbeat-immortality-148306
  - Youtube Audio Library
- Código de colisiones:
  - https://github.com/proyectos-ce/XeonWars/blob/master/Collision.h
- Teorema de ejes de separación:
  - https://research.ncl.ac.uk/game/mastersdegree/gametechnologies/previousinformation/physics4collisiondetection/2017%20Tutorial%204%20-%20Collision%20Detection.pdf

Gracias, jueguen