Jungle Arcade

JUEGO CREADO DE C++ Y SFML

/luisdavaria /Davaria

Mario Bros Role Play

VSCODE

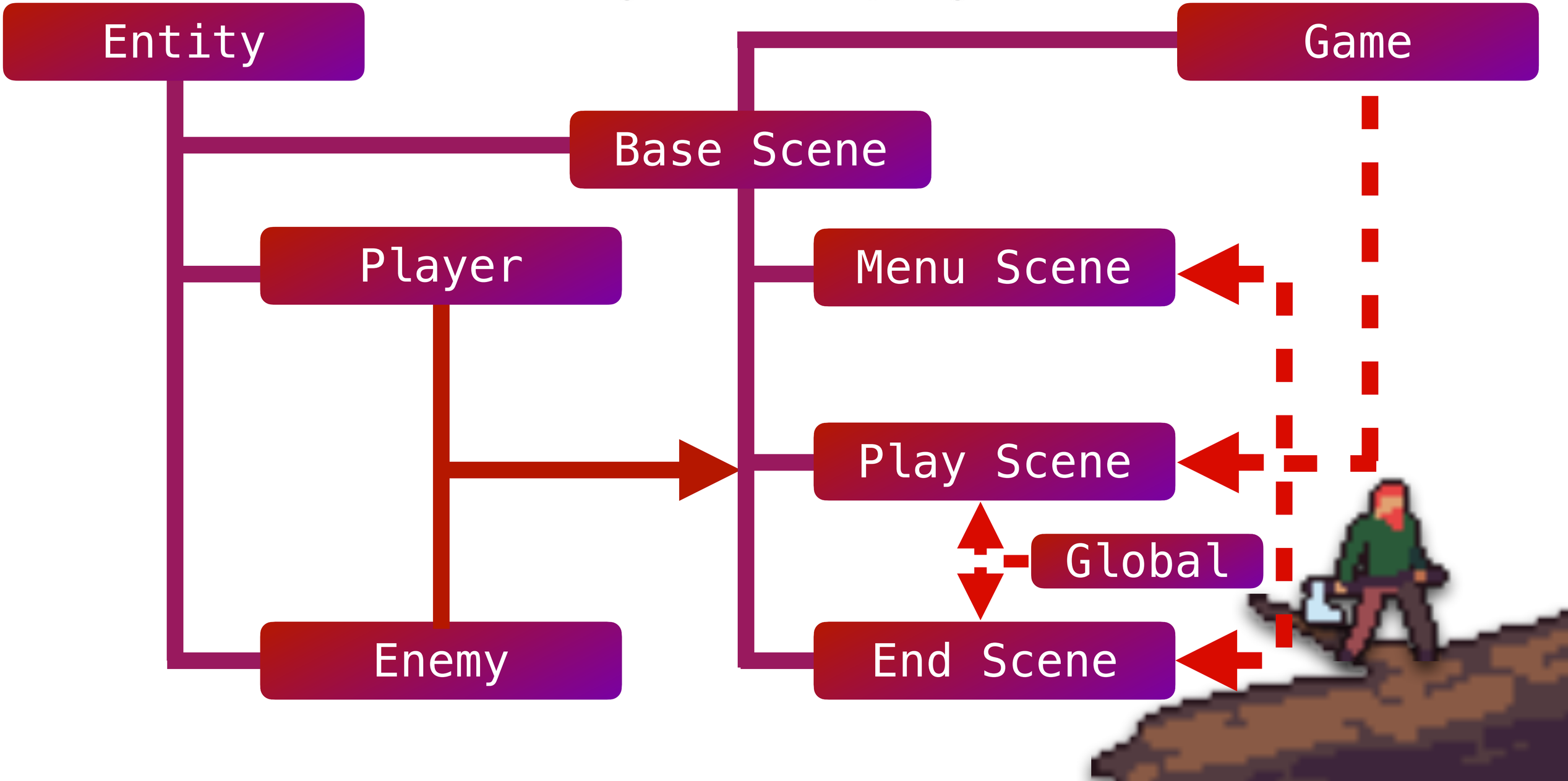
SFML

Patron Singleton

Clases Abtractas
POO

Funciones Virtuales
Herencia

Diagrama de juego

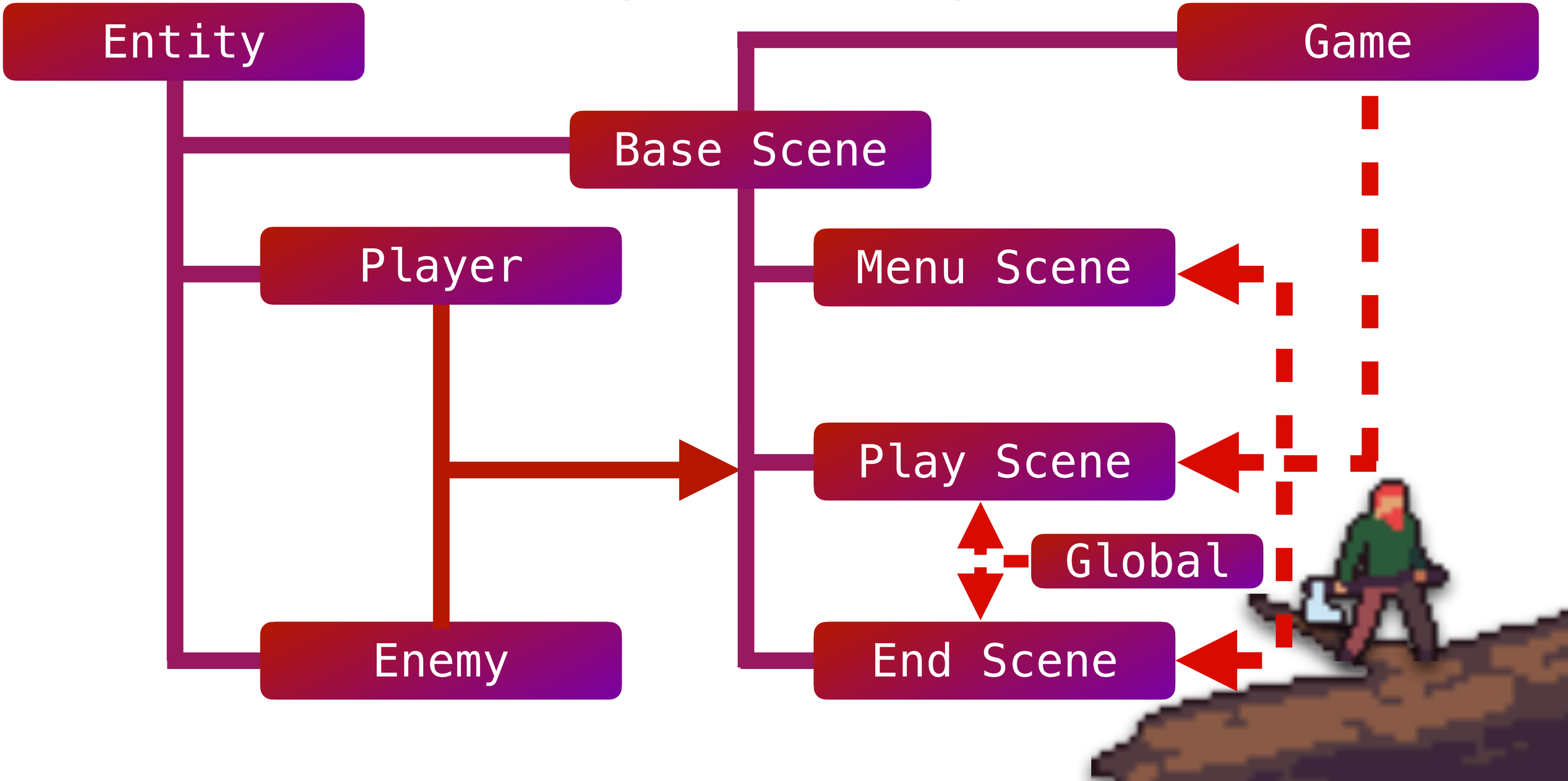


Clase Abstracta ~ Entity

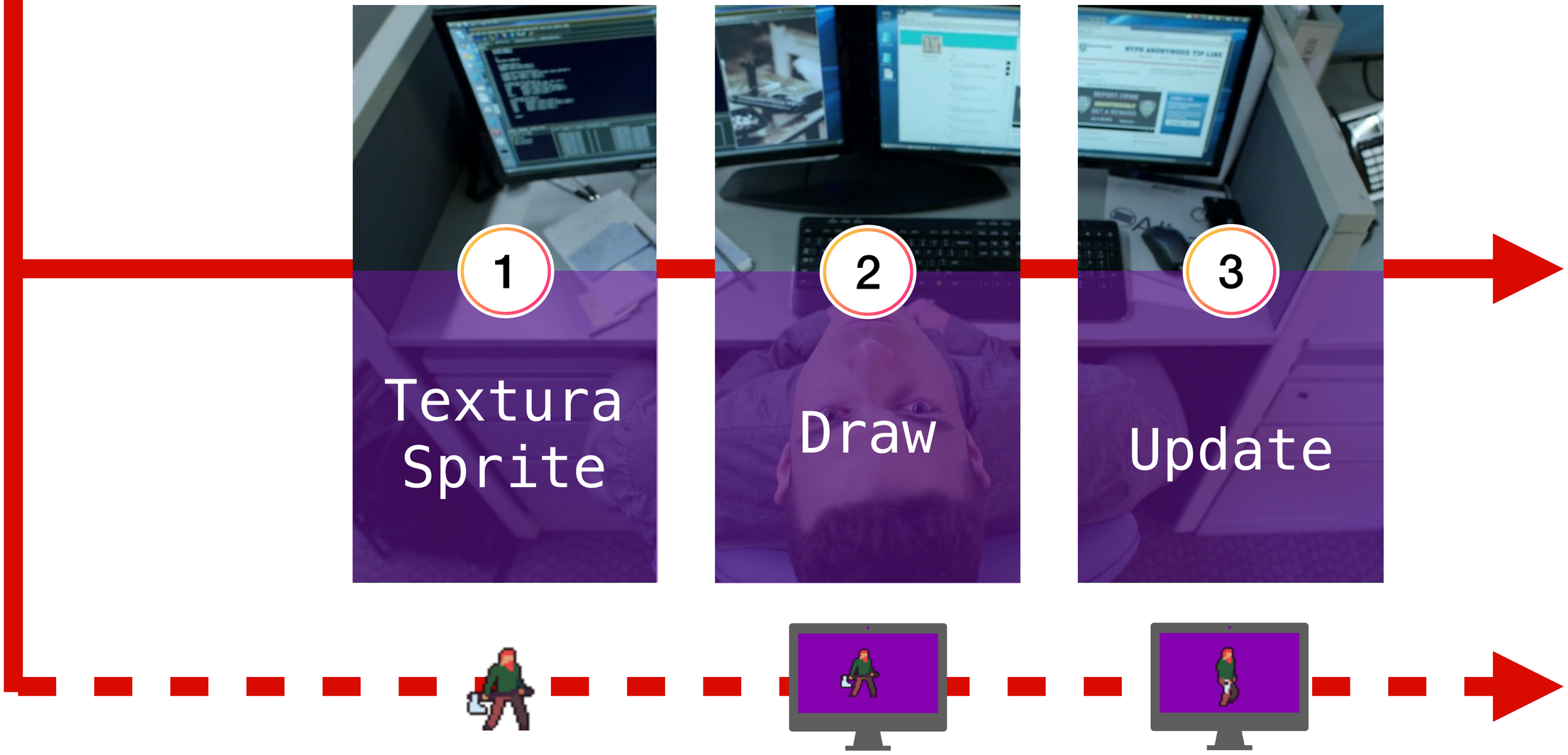
```
#ifndef ENTITY_H
#define ENTITY_H
#include <SFML/Graphics.hpp>
using namespace sf;

class Entity
{
public:
  virtual void update(float) = 0;
  virtual void draw(RenderWindow &w) = 0;
};

#endif
```

Davaria/Proyecto-CC2

Diagrama de juego



Antes de empezar



Clase Enemy

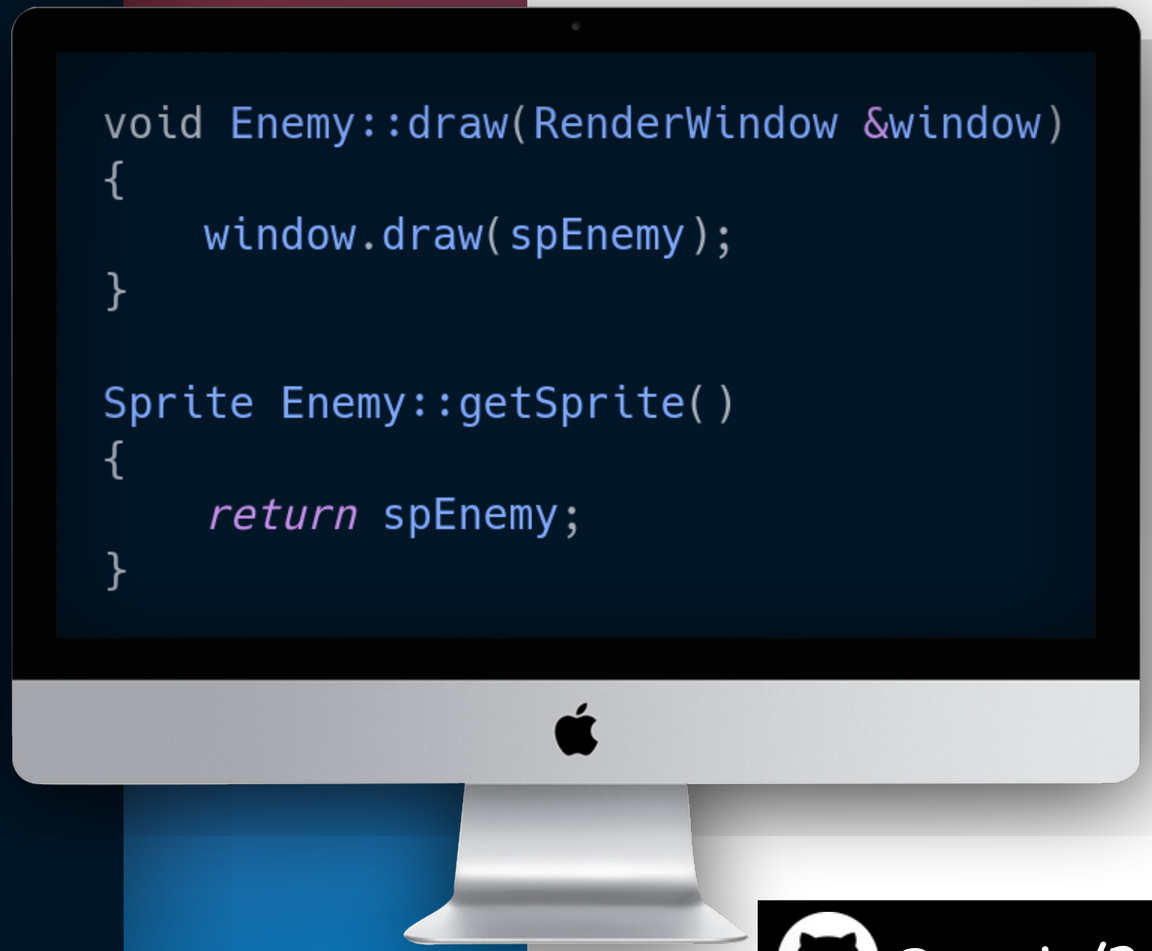
```cpp
class Enemy : public Entity
{
private:
  Sprite spEnemy;
  Texture textEnemy;
  // Velocidad del enemigo
  Vector2f vellEnemy;
  const float EnemyVel = 1;
  // Valores de acorde al tamaño de juego
  int valueX = 300, valueY = 200;
  // Animacion
  Animation enemyAnim;
  float deltaTime = 0.0f;
  Clock clock;
  bool faceR;

public:
  Enemy();
  // Manejo de escena
  void update(float) override;
  void draw(RenderWindow &) override;
  // Metodo que devuelva el sprite
  Sprite getSprite();
};
```

```cpp
Enemy::Enemy()
{
    // Cargando texturas
    textEnemy.loadFromFile("./assets/desert-enemys/4 Vulture/Vulture_walk.png");
    spEnemy.setTexture(textEnemy);

    // Modificar la velocidad del enemigo
    srand(time(0));
    spEnemy.setPosition(rand() % valueX, rand() % valueY);
    vellEnemy.x = 2 + rand() % 2;
    vellEnemy.y = 2 + rand() % 2;

    // Animacion
    enemyAnim.setAnim(&textEnemy, 4, 0.1f);
}
```

Davaria/Proyecto-CC2

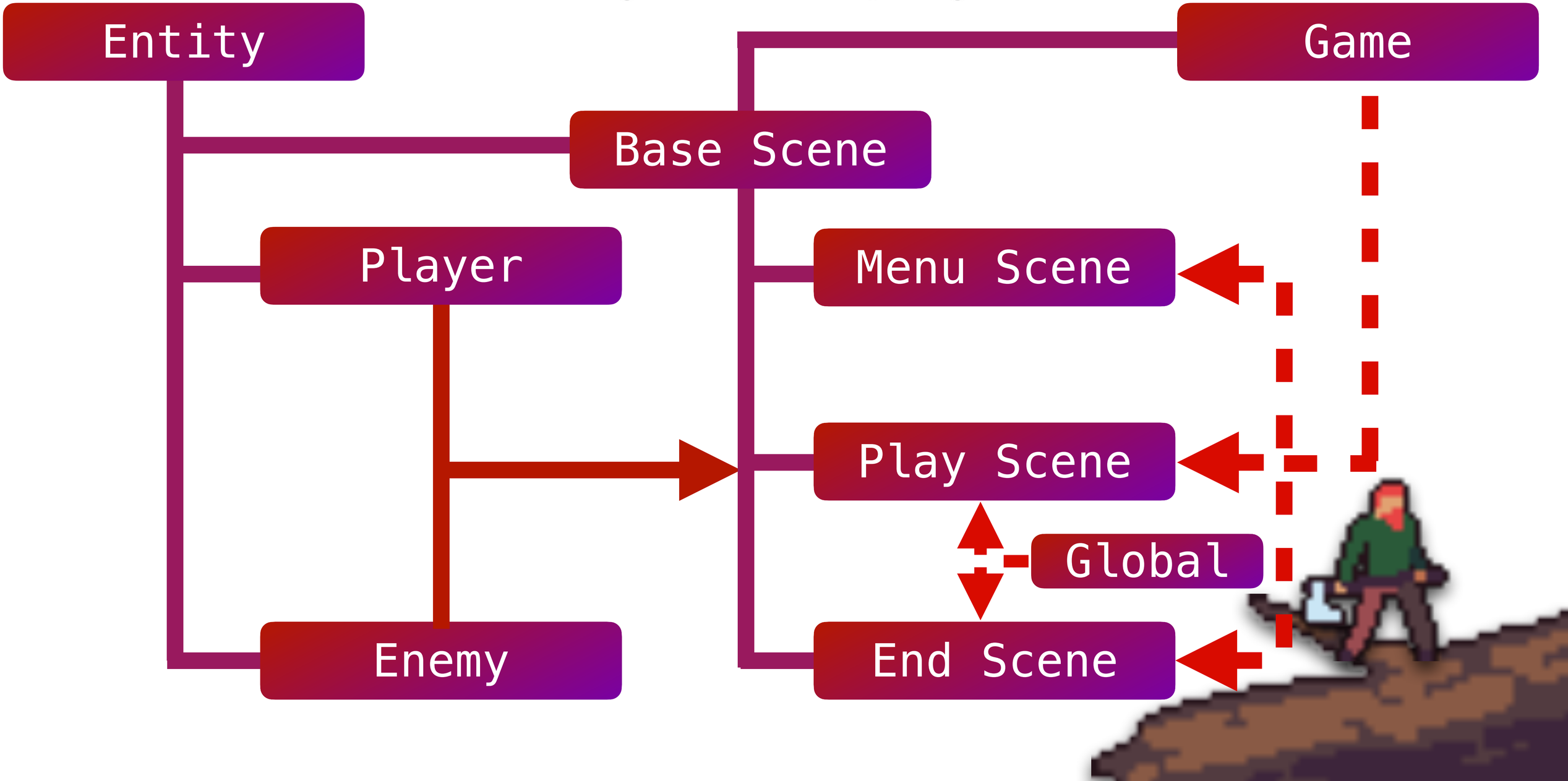
Clase Enemy

```
void Enemy::update(float elapsed)
{
    deltaTime = clock.restart().asSeconds();
    // mover al enemigo
    spEnemy.move(vellEnemy);
    // limitar el movimiento del enemigo
    Vector2f charPosEnemy = spEnemy.getPosition();
    if (charPosEnemy.x < 0)
    {
        charPosEnemy.x = 0;
        vellEnemy.x *= -1;
        faceR = false;
    }
    if (charPosEnemy.x > 500)
    {
        charPosEnemy.x = 500;
        vellEnemy.x *= -1;
        faceR = true;
    }
    if (charPosEnemy.y < 0)
    {
        charPosEnemy.y = 0;
        vellEnemy.y *= -1;
    }
    if (charPosEnemy.y > 300)
    {
        charPosEnemy.y = 300;
        vellEnemy.y *= -1;
    }

    enemyAnim.updateAnim(deltaTime, faceR);
    spEnemy.setTextureRect(enemyAnim.uvRect);
    spEnemy.setPosition(charPosEnemy);
}
```



Davaria/Proyecto-CC2

Diagrama de juego



Clase BaseScene

- Representa una escena del juego.
- Para crear una nueva clase de escena, heredar de esta clase
- Redefinir el comportamiento de los metodos

```
class BaseScene
{
    vector<Entity *> entities;
    vector<Entity *> to_delete;

public:
    // funcion que sera invocada para actualizar la escena
    virtual void update(float);

    // funcion que sera invocada para dibujar la escena
    virtual void draw(RenderWindow &);

    // agrega un nuevo actor a la escena
    void add(Entity *);

    // eliminar un actor de la escena
    void remove(Entity *e);
};
```

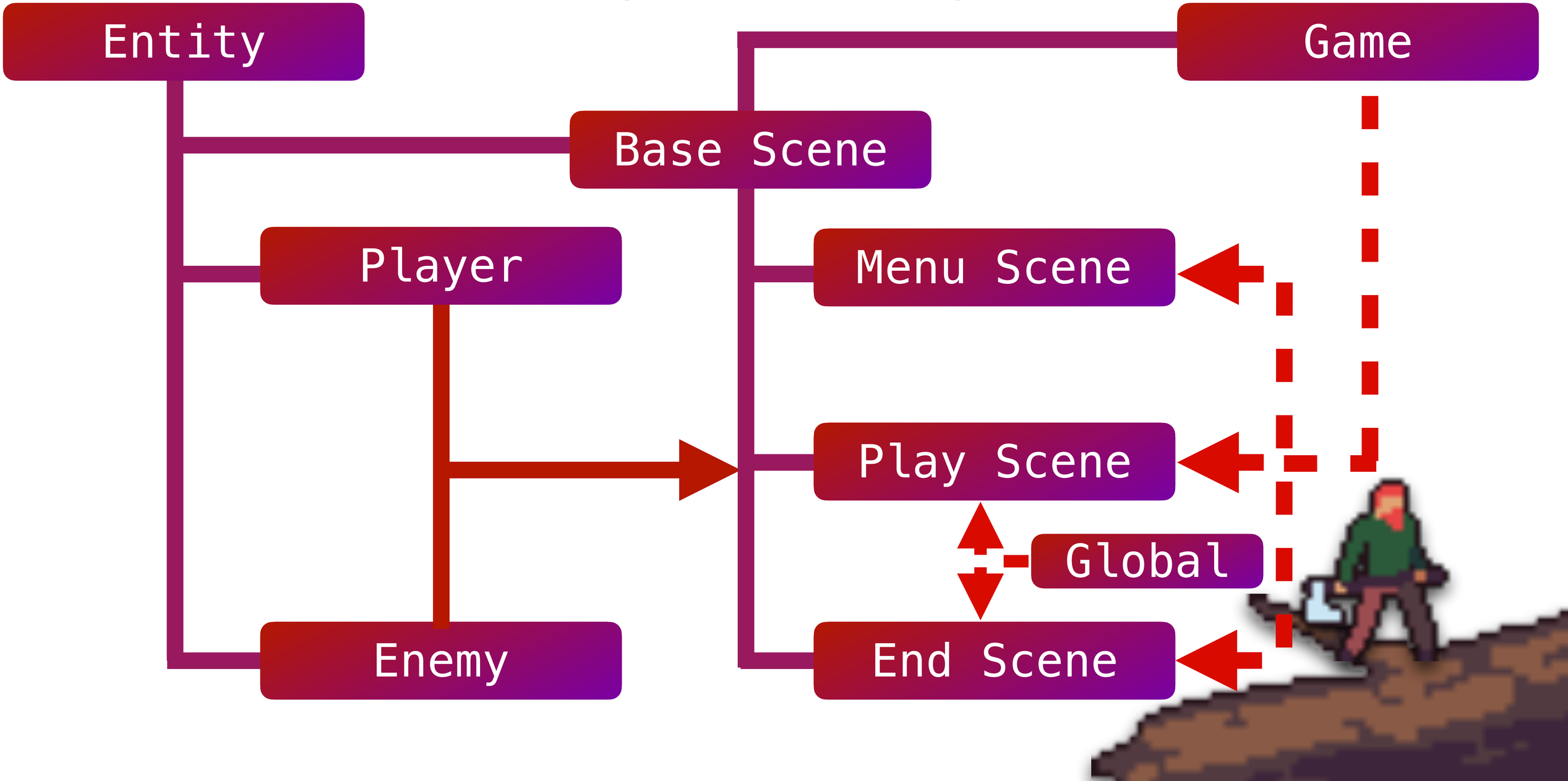
Clase BaseScene

```
void BaseScene::draw(RenderWindow &w)
{
    for (auto e : entities)
        e->draw(w);
}

void BaseScene::add(Entity *e)
{
    entities.push_back(e);
}

void BaseScene::remove(Entity *e)
{
    to_delete.push_back(e);
}
```

```
void BaseScene::update(float elapsed)
{
    for (auto e : entities)
        e->update(elapsed);

    // elimina actores
    for (auto d : to_delete)
    {
        auto it = find(entities.begin(), entities.end(), d);
        if (it != entities.end())
            entities.erase(it);
    }
    to_delete.clear();
}
```

Davaria/Proyecto-CC2

Diagrama de juego



Clase PlayScene

```
class PlayScene : public BaseScene
{
    Character *player;      //nuestro personaje
    vector<Enemy *> enemies;  //arreglo con enemigos
    float minVel;           //Velocidad en la que un enemigo se creara
    Clock timeToNewEnemy;    // tiempo para crear x enemigos
    float score;            // puntaje
    Text scoreText;         // texto para mostrar el puntaje
    Font fontScoreText;     // fuente del texto
    Texture textBackground; // textura del fondo
    Sprite spBackground;    // sprite del fondo
    Music bgm;

    void createNewEnemy(float); // crear una nuevo enemigo

public:
    PlayScene();
    void update(float);
    void draw(RenderWindow &);
};
```

Davaria/Proyecto-CC2

PlayScene

```cpp
PlayScene::PlayScene()
{
    // Crear el personaje
    player = new Character(Vector2f(100, 200));
    add(player);

    minVel = 150;
    createNewEnemy(minVel);
    timeToNewEnemy.restart();

    // Music
    if (!bgm.openFromFile("./assets/music/astronomia_bgm.wav"))
        cout << "error" << endl;
    else
    {
        bgm.setLoop(true);
        bgm.play();
    }
    // Music

    // Puntaje
    if (!fontScoreText.loadFromFile("./assets/font/Cave-Story.ttf"))
    {
        cerr << "ERROR: no se encontro la fuente "
            << "PlayScene.cpp"
            << endl;
    }
    // Puntaje

    // Texturas de nuestros videojuego
    textBackground.loadFromFile("./assets/swamp-game-tileset/2 Background/Background.png");
    spBackground.setTexture(textBackground);
    // Texturas de nuestros videojuego

    score = 0;
    scoreText.setFont(fontScoreText);
    scoreText.setFillColor(Color::White);
}
```

```cpp
void PlayScene::update(float elapsed)
{
    //  actualiza el personaje
    BaseScene::update(elapsed);
    // creamos nuevo enemigo
    if (timeToNewEnemy.getElapsedTime().asSeconds() > 20)
    {
        minVel += 20;
        createNewEnemy(minVel);
        timeToNewEnemy.restart();
    }
    // Colisiones entre el personaje y las pelotas
    for (const auto &enemy : enemies)
    {
        if (player->collidesWithEnemy(enemy))
        {
            if (score > Global::highScore)
                Global::highScore = score;
            Game::getInstance().switchScene(new EndScene());
        }
    }

    // actualiza el puntaje
    score += elapsed;
    //stringstream leer o escribir sobre una cadena
    stringstream tmp;
    tmp << "PUNTAJE: " << int(score);
    scoreText.setString(tmp.str());
}
```

Davaria/Proyecto-CC2

PlayScene

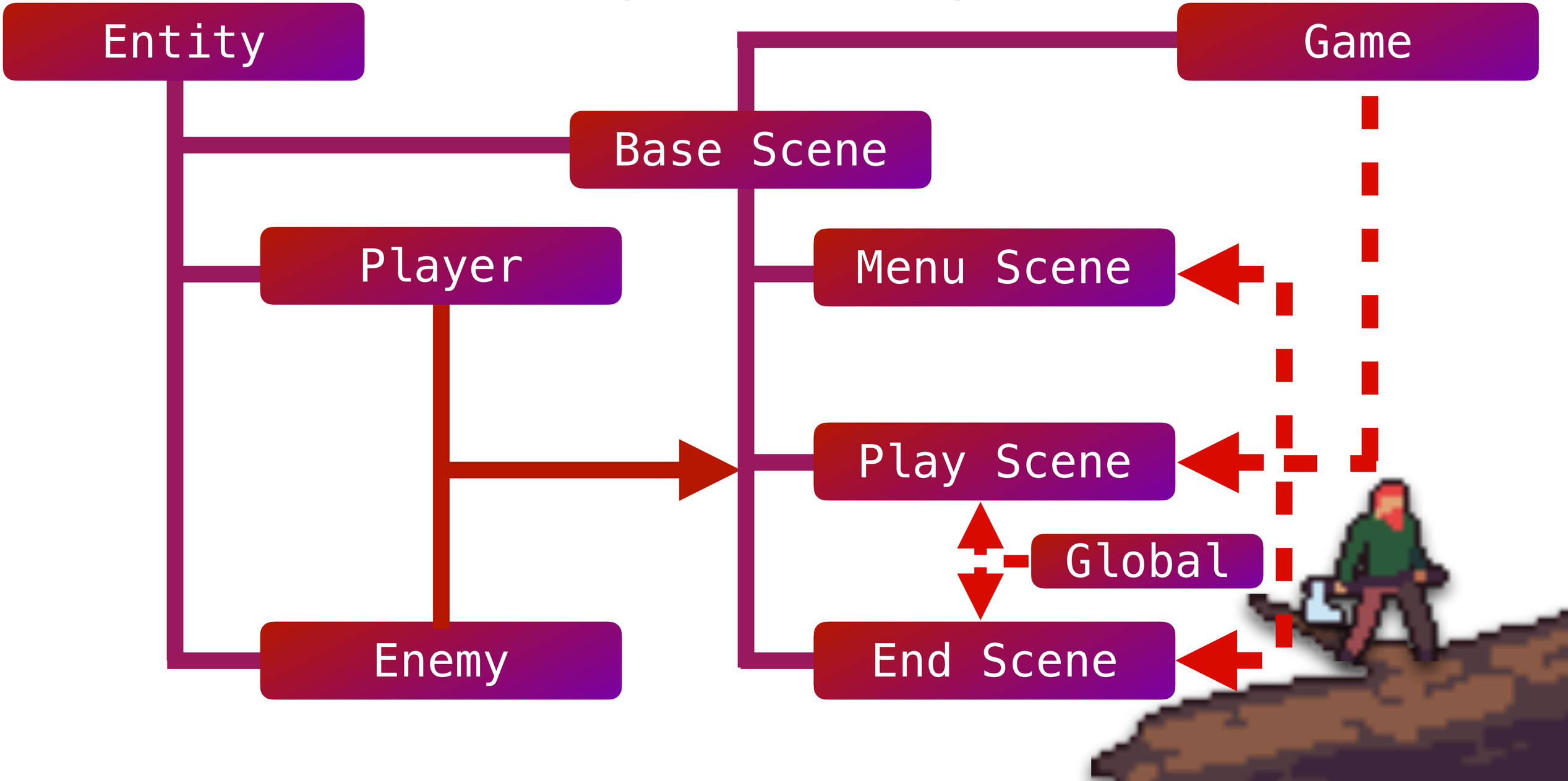
```
void PlayScene::draw(RenderWindow &window)
{
    // Limpia la pantalla
    window.draw(spBackground);
    BaseScene::draw(window);
    window.draw(scoreText);
}

void PlayScene::createNewEnemy(float minVel)
{
    Enemy *b = new Enemy();
    enemies.push_back(b);
    add(b);
}
```

Global

```
class Global
{
public:
  static int highScore;
};

int Global::highScore = 0;
```

Davaria/Proyecto-CC2

Diagrama de juego



Clase Game

Clase para manejar un bucle de juego con distintas escenas

```
class Game
{

  RenderWindow window;
  BaseScene *currentScene, *nextScene;
  Clock clock;

  void processEvents();
  void update();
  void draw();
  static Game *instance;
  Game();

public:
  /// comenzar el bucle de juego
  void run();

  /// cambiar la escena actual por otra
  void switchScene(BaseScene *scene);

  /// obtener la instancia de juego (singleton)
  static Game &getInstance();

  /// crear un juego especificando el modo de video y la escena inicial
  static Game &create(const VideoMode &videoMode, BaseScene *scene, const string &name = "");
};
```

Patron singleton

```
Game *Game::instance = nullptr;
Game::Game() {}

Game &Game::create(const VideoMode &videoMode, BaseScene *scene, const string &name)
{
    if (instance)
        cerr << "ERROR: can't call create(), game already running." << endl;
    else
    {
        Game &g = getInstance();
        g.window.create(videoMode, name, Style::Default);
        g.nextScene = nullptr;
        g.currentScene = scene;
        g.window.setFramerateLimit(60);
        g.clock.restart();
    }
    return getInstance();
}
```

Permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

```
Game &Game::getInstance()
{
    if (!instance)
        instance = new Game();
    return *instance;
}
```

Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

Class Game

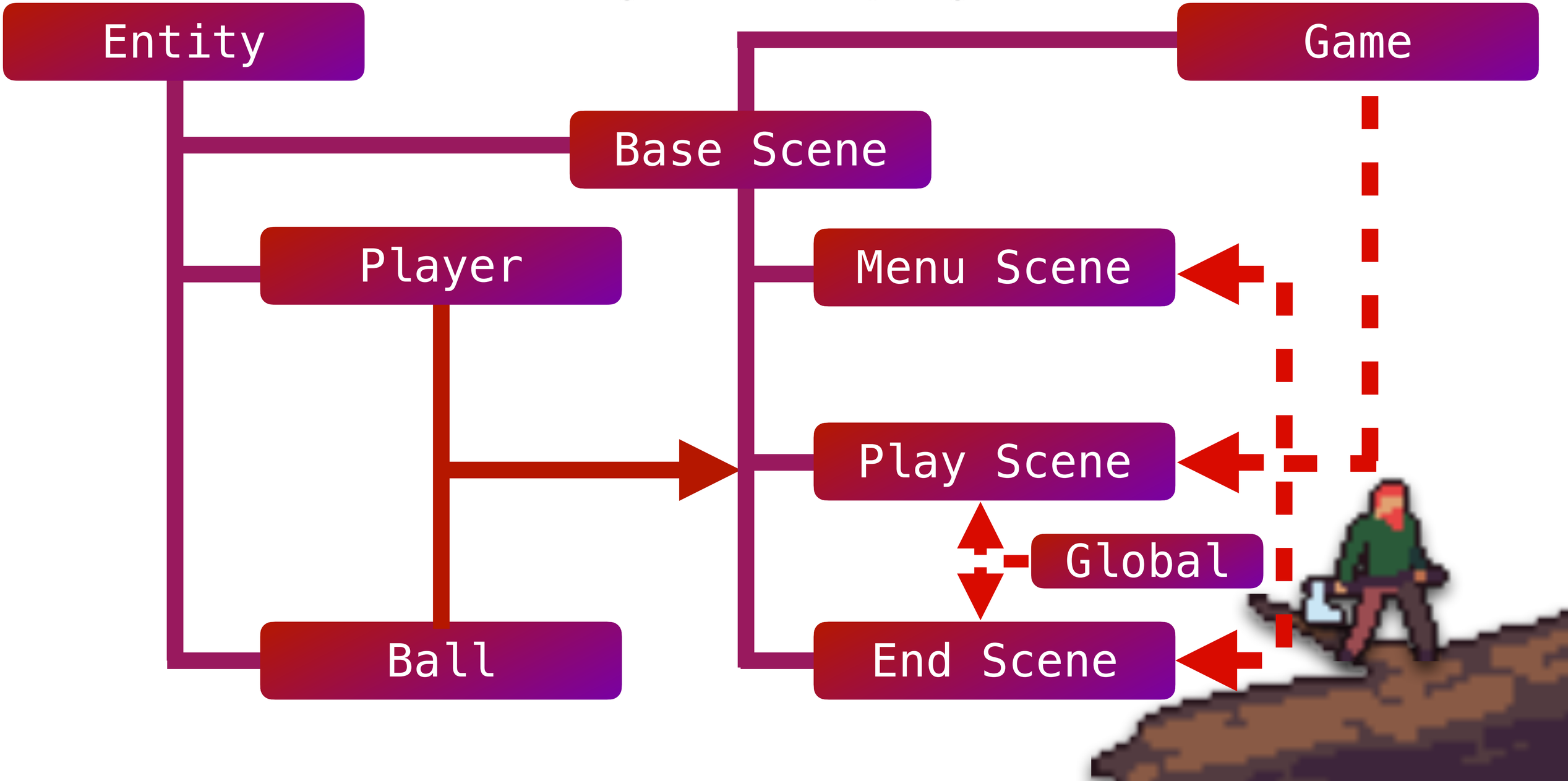
1

```
void Game::run()
{
    while (window.isOpen() && currentScene != nullptr)
    {
        Event e;
        while (window.pollEvent(e))
        {
            if (e.type == Event::Closed)
                window.close();
        }
        update();
        draw();
        if (nextScene != nullptr)
        {
            delete currentScene;
            currentScene = nextScene;
            nextScene = nullptr;
        }
    }
}
```

2

```
void Game::update()
{
    currentScene->update(clock.getElapsedTime().asSeconds());
    clock.restart();
}

void Game::draw()
{
    window.clear(Color(0, 0, 0, 255));
    currentScene->draw(window);
    window.display();
}

void Game::switchScene(BaseScene *scene)
{
    nextScene = scene;
}
```

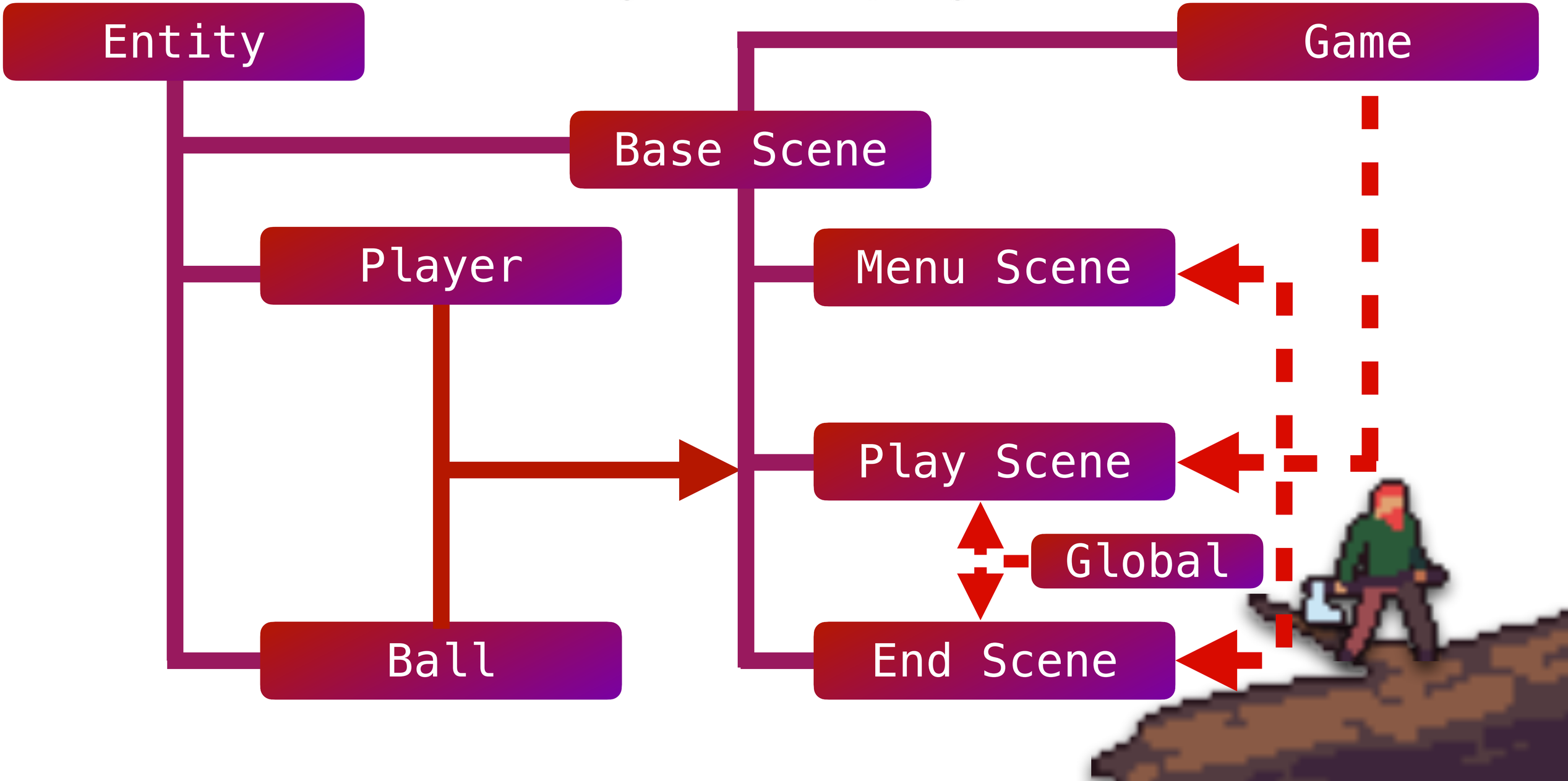
Diagrama de juego



```
int main(int argc, char const *argv[])
{
    BaseScene *scene = new MenuScene();
    Game &g = Game::create(VideoMode(576, 324), scene, "Jungle Arcade");
    g.run();
    return 0;
}
```

Davaria/Proyecto-CC2

Diagrama de juego



Conclusiones Finales

Cada escena hereda de BaseScene es responsable de ***actualizar/dibujar*** todos los objetos en su conjunto y agregar(redifinir) comportamientos extra (que no son responsabilidad exclusiva de un objeto)

Cada objeto hereda de Entity y es reponsable de redifinir como se actualiza y dibuja el mismo.

La clase Game se encarga de gestionar las escenas.

Davaria/Proyecto-CC2

Fondo de color

Tipografias

TITULOS /Thonburi

Texto/Menlo