CHULETA DE LO APRENDIDO (LABORATORIOS)

# Laboratorio 1 (03/02/2025):

## Crear y meter objetos en escena (fighter).

* Declaramos en **Game.h** el objeto (private): **Container \*\_fighter;**
* En **void Game::init()**, dejamos la inicialización de SDLUtil e InputHandler y creamos un **\_fighter**.
  + Lo creamos.
  + Le damos posición.
  + Le damos tamaño.
  + Le damos rotación.
  + Le añadimos los componentes.
  + Lo metemos al juego.

|  |
| --- |
| **void Game::init() {**  **// initialize the SDL singleton**  **if (!SDLUtils::Init("Ping Pong", 800, 600,**  **"resources/config/test.resources.json")) {**  **std::cerr << "Something went wrong while initializing SDLUtils"**  **<< std::endl;**  **return;**  **}**  **// initialize the InputHandler singleton**  **if (!InputHandler::Init()) {**  **std::cerr << "Something went wrong while initializing SDLHandler"**  **<< std::endl;**  **return;**  **}**  **// the fighter.**    **// creation**  **\_fighter = new Container();**  **// pos (dado que origen es top left, lo centramos quitando la mitad del ancho y el alto)**  **\_fighter->getPos().set(sdlutils().width() / 2 - 25.0f, sdlutils().height() / 2 - 25.0f);**  **// size**  **\_fighter->setWidth(50.0f);**  **\_fighter->setHeight(50.0f);**  **//Setea la rotación**  **\_fighter->setRotation(90.0f);**  **// add components**  **\_fighter->addComponent(new ImageRenderer(&sdlutils().images().at("fighter"))); //ImageRenderer**  **\_fighter->addComponent(new FighterCtrl()); //FighterCtrl**  **// add to objects vector list of the game.**  **\_objs.push\_back(\_fighter);**  **}** |

|  |
| --- |
| Nota: el método **init()** indica qué se va a inicializar al comienzo del juego (Game), por eso es el init del **Game.h/Game.cpp**. |

## Añadir propiedad a un GameObject (rotation).

|  |
| --- |
| Nota: **GameObject** solo tiene **.h** porque es abstracta, lo que quiere decir que solo está creada para ser clase base. |

* Nos vamos a GameObject.h y le metemos la propiedad rotación (protected, para que solo los gameObjects que hereden de ella puedan acceder): **float \_rot;**
* Se podría inicializar su valor de dos formas:

|  |  |
| --- | --- |
| LA FORMA QUE PREFIERE SAMIR | LA FORMA QUE TAMBIÉN LE VALE A SAMIR (en caso de ir deprisa/estar vagas podemos valernos de esta, es igual de válida) |
| **GameObject() :**  **\_pos(), \_vel(), \_width(), \_height(), \_rot(0.0f) {**  **}** | **protected:**  **Vector2D \_pos;**  **Vector2D \_vel;**  **float \_width;**  **float \_height;**  **float \_rot = 0.0f;** |

* Le metemos los getters y setters (también están los de las demás propiedades):

|  |
| --- |
| **//Rotation**  **inline float getRotation() const {**  **return \_rot;**  **}**  **inline void setRotation(float rotation) {**  **\_rot = rotation;**  **}** |

## Modificar un componente (ImageRender).

* Modificamos ImageRender para que haga render con la rotación (ImageRender hereda de RenderComponent, RenderComponent hereda de Component):

|  |  |
| --- | --- |
| Antes era:  void ImageRenderer::render(Container \*o) {  SDL\_Rect dest = build\_sdlrect(o->getPos(), o->getWidth(), o->getHeight());  float r = o->getRotation();  **\_img->render(dest);**  } | Ahora es:  void ImageRenderer::render(Container \*o) {  SDL\_Rect dest = build\_sdlrect(o->getPos(), o->getWidth(), o->getHeight());  float r = o->getRotation();  **\_img->render(dest, r);**  } |

|  |  |  |
| --- | --- | --- |
| En Texture.h…   |  |  | | --- | --- | | // renders the complete texture at a destination rectangle (dest)  inline void render(**const SDL\_Rect &dest**) {  SDL\_Rect src = { 0, 0, \_width, \_height };  **render(src, dest);**  } | // renders the complete texture at a destination rectangle (dest), with rotation  inline void render(**const SDL\_Rect &dest, float rotation**) {  SDL\_Rect src = { 0, 0, \_width, \_height };  **render(src, dest, rotation);**  } | |

Ya se renderiza la rotación, se puede setear en el init ya: **\_fighter->setRotation(90.0f);**

## Crear un componente (FighterCtrl).

## Renderización de pantalla de SDL.

