

---

# SGG player\_orbit.h Library

---

Επιμέλεια: Α.Δρακόπουλος

---

## Πίνακας περιεχομένων

Βιβλιοθήκη player_orbit.h.....	3
player_orbit.h ( τεκμηρίωση στα Αγγλικά ).....	3
player_orbit_es.h ( τεκμηρίωση στα Ισπανικά ).....	6
player_orbit_gr.h ( τεκμηρίωση στα Ελληνικά ).....	9
Πρόγραμμα main.cpp για χρήση των συναρτήσεων.....	11
Επεξήγηση των Τροποποιήσεων.....	12

# Βιβλιοθήκη player\_orbit.h

## player\_orbit.h ( τεκμηρίωση στα Αγγλικά )

```
#pragma once
#ifndef PLAYER_ORBIT_H
#define PLAYER_ORBIT_H

#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

/**
 * Structure that defines the properties of a player, represented as a circle.
 */
struct Player {
    float x, y;          ///< Center position of the player.
    float radius;        ///< Radius of the circle representing the player.
    float speed;         ///< Speed of the player's movement.
    float angle;         ///< Angle in degrees for directional movements like inclined
throws.
    float time;          ///< Time variable for time-based movements (e.g., free fall).
};

/**
 * Moves the player horizontally by a specified speed.
 * @param player Reference to the Player structure to be moved.
 * @param speed Horizontal speed to be applied to the player's movement.
 */
void moveHorizontal(Player& player, float speed) {
    player.x += speed;
}

/**
 * Moves the player vertically by a specified speed.
 * @param player Reference to the Player structure to be moved.
 * @param speed Vertical speed to be applied to the player's movement.
 */
void moveVertical(Player& player, float speed) {
    player.y += speed;
}

/**
 * Moves the player in a circular motion around a specified center.
 * @param player Reference to the Player structure to be moved.
 * @param centerX X-coordinate of the center of the circular path.
 * @param centerY Y-coordinate of the center of the circular path.
 * @param radius Radius of the circular path.
 * @param speed Angular speed of the circular motion in degrees per update.
 */
void moveCircular(Player& player, float centerX, float centerY, float radius, float
speed) {
    player.angle += speed;
    player.x = centerX + radius * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + radius * sin(player.angle * M_PI / 180.0f);
}

/**
 * Simulates free fall motion for the player under gravity.
 * @param player Reference to the Player structure to be moved.
 * @param gravity Acceleration due to gravity (default is 9.8 m/s^2).
 */
void freeFall(Player& player, float gravity = 9.8f) {
```

```

    player.y += 0.5f * gravity * player.time * player.time;
    player.time += 0.1f; // Updates time in each frame
}

/**
 * Simulates a vertical throw upwards for the player.
 * @param player Reference to the Player structure to be moved.
 * @param initialVelocity Initial vertical velocity of the throw.
 * @param gravity Acceleration due to gravity (default is 9.8 m/s^2).
 */
void verticalThrow(Player& player, float initialVelocity, float gravity = 9.8f) {
    player.y -= initialVelocity * player.time - 0.5f * gravity * player.time *
player.time;
    player.time += 0.1f;
}

/**
 * Simulates an inclined throw at an angle for the player.
 * @param player Reference to the Player structure to be moved.
 * @param initialVelocity Initial velocity of the throw.
 * @param angle Angle of the throw in degrees.
 * @param gravity Acceleration due to gravity (default is 9.8 m/s^2).
 */
void inclinedThrow(Player& player, float initialVelocity, float angle, float gravity =
9.8f) {
    float radAngle = angle * M_PI / 180.0f;
    player.x += initialVelocity * cos(radAngle) * player.time;
    player.y -= (initialVelocity * sin(radAngle) * player.time - 0.5f * gravity *
player.time * player.time);
    player.time += 0.1f;
}

/**
 * Moves the player in an elliptical path with a constant speed.
 * @param player Reference to the Player structure to be moved.
 * @param centerX X-coordinate of the center of the elliptical path.
 * @param centerY Y-coordinate of the center of the elliptical path.
 * @param a Semi-major axis length of the ellipse.
 * @param b Semi-minor axis length of the ellipse.
 * @param speed Angular speed of the elliptical motion in degrees per update.
 */
void ellipticalMotion(Player& player, float centerX, float centerY, float a, float b,
float speed) {
    player.angle += speed;
    player.x = centerX + a * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + b * sin(player.angle * M_PI / 180.0f);
}

/**
 * Draws the player as a circle at the current position.
 * @param player Constant reference to the Player structure to be drawn.
 */
void drawPlayer(const Player& player) {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f; // Blue color for the player
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 1.0f;
    graphics::drawDisk(player.x, player.y, player.radius, brush);
}

/**
 * Resets the time variable of the player for movement calculations.
 * @param player Reference to the Player structure whose time is to be reset.
 */
void resetTime(Player& player) {

```

```
    player.time = 0.0f;  
}  
#endif // PLAYER_ORBIT_H
```

## player\_orbit\_es.h ( τεκμηρίωση στα Ισπανικά )

```
#pragma once
#ifndef PLAYER_ORBIT_ES_H
#define PLAYER_ORBIT_ES_H

#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

/**
 * Estructura que define las propiedades de un jugador, representado como un círculo.
 */
struct Player {
    float x, y;          ///< Posición central del jugador.
    float radius;        ///< Radio del círculo que representa al jugador.
    float speed;         ///< Velocidad de movimiento del jugador.
    float angle;         ///< Ángulo en grados para movimientos direccionales, como
    lanzamientos inclinados.
    float time;          ///< Variable de tiempo para movimientos basados en tiempo (por
    ejemplo, caída libre).
};

/**
 * Mueve el jugador horizontalmente a una velocidad especificada.
 * @param player Referencia a la estructura Player que será movida.
 * @param speed Velocidad horizontal que se aplicará al movimiento del jugador.
 */
void moveHorizontal(Player& player, float speed) {
    player.x += speed;
}

/**
 * Mueve el jugador verticalmente a una velocidad especificada.
 * @param player Referencia a la estructura Player que será movida.
 * @param speed Velocidad vertical que se aplicará al movimiento del jugador.
 */
void moveVertical(Player& player, float speed) {
    player.y += speed;
}

/**
 * Mueve el jugador en un movimiento circular alrededor de un centro especificado.
 * @param player Referencia a la estructura Player que será movida.
 * @param centerX Coordenada X del centro de la trayectoria circular.
 * @param centerY Coordenada Y del centro de la trayectoria circular.
 * @param radius Radio de la trayectoria circular.
 * @param speed Velocidad angular del movimiento circular en grados por actualización.
 */
void moveCircular(Player& player, float centerX, float centerY, float radius, float
speed) {
    player.angle += speed;
    player.x = centerX + radius * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + radius * sin(player.angle * M_PI / 180.0f);
}

/**
 * Simula el movimiento de caída libre para el jugador bajo la influencia de la
 gravedad.
 * @param player Referencia a la estructura Player que será movida.
 * @param gravity Aceleración debido a la gravedad (por defecto 9.8 m/s^2).
 */
void freeFall(Player& player, float gravity = 9.8f) {
    player.y += 0.5f * gravity * player.time * player.time;
}
```

```

    player.time += 0.1f; // Actualiza el tiempo en cada fotograma
}

/**
 * Simula un lanzamiento vertical hacia arriba para el jugador.
 * @param player Referencia a la estructura Player que será movida.
 * @param initialVelocity Velocidad vertical inicial del lanzamiento.
 * @param gravity Aceleración debido a la gravedad (por defecto 9.8 m/s^2).
 */
void verticalThrow(Player& player, float initialVelocity, float gravity = 9.8f) {
    player.y -= initialVelocity * player.time - 0.5f * gravity * player.time *
player.time;
    player.time += 0.1f;
}

/**
 * Simula un lanzamiento inclinado en un ángulo para el jugador.
 * @param player Referencia a la estructura Player que será movida.
 * @param initialVelocity Velocidad inicial del lanzamiento.
 * @param angle Ángulo del lanzamiento en grados.
 * @param gravity Aceleración debido a la gravedad (por defecto 9.8 m/s^2).
 */
void inclinedThrow(Player& player, float initialVelocity, float angle, float gravity =
9.8f) {
    float radAngle = angle * M_PI / 180.0f;
    player.x += initialVelocity * cos(radAngle) * player.time;
    player.y -= (initialVelocity * sin(radAngle) * player.time - 0.5f * gravity *
player.time * player.time);
    player.time += 0.1f;
}

/**
 * Mueve el jugador en una trayectoria elíptica con velocidad constante.
 * @param player Referencia a la estructura Player que será movida.
 * @param centerX Coordenada X del centro de la trayectoria elíptica.
 * @param centerY Coordenada Y del centro de la trayectoria elíptica.
 * @param a Longitud del semi-eje mayor de la elipse.
 * @param b Longitud del semi-eje menor de la elipse.
 * @param speed Velocidad angular del movimiento elíptico en grados por actualización.
 */
void ellipticalMotion(Player& player, float centerX, float centerY, float a, float b,
float speed) {
    player.angle += speed;
    player.x = centerX + a * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + b * sin(player.angle * M_PI / 180.0f);
}

/**
 * Dibuja el jugador como un círculo en la posición actual.
 * @param player Referencia constante a la estructura Player que será dibujada.
 */
void drawPlayer(const Player& player) {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f; // Color azul para el jugador
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 1.0f;
    graphics::drawDisk(player.x, player.y, player.radius, brush);
}

/**
 * Restaura la variable de tiempo del jugador para los cálculos de movimiento.
 * @param player Referencia a la estructura Player cuyo tiempo será restaurado.
 */
void resetTime(Player& player) {
    player.time = 0.0f;
}

```

```
}
```

```
#endif // PLAYER_ORBIT_ES_H
```



## player\_orbit\_gr.h ( τεκμηρίωση στα Ελληνικά )

```
#pragma once
#ifndef PLAYER_ORBIT_GR_H
#define PLAYER_ORBIT_GR_H

#include "sgg/graphics.h"
#include <cmath>
#define M_PI 3.14159265358979323846

struct Player {
    float x, y;           // Θέση του κέντρου του παίκτη
    float radius;         // Ακτίνα του κύκλου που αναπαριστά τον παίκτη
    float speed;          // Ταχύτητα του παίκτη
    float angle;          // Κλίση σε μοίρες για τις βολές
    float time;           // Χρόνος για κίνηση βασισμένη στον χρόνο (ελεύθερη πτώση κτλ)
};

// Ομαλή ευθύγραμμη οριζόντια κίνηση
void moveHorizontal(Player& player, float speed) {
    player.x += speed;
}

// Ομαλή ευθύγραμμη κατακόρυφη κίνηση
void moveVertical(Player& player, float speed) {
    player.y += speed;
}

// Κυκλική κίνηση
void moveCircular(Player& player, float centerX, float centerY, float radius, float
speed) {
    player.angle += speed;
    player.x = centerX + radius * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + radius * sin(player.angle * M_PI / 180.0f);
}

// Ελεύθερη πτώση
void freeFall(Player& player, float gravity = 9.8f) {
    player.y += 0.5f * gravity * player.time * player.time;
    player.time += 0.1f; // Ενημέρωση του χρόνου σε κάθε καρέ
}

// Κάθετη προς τα πάνω βολή
void verticalThrow(Player& player, float initialVelocity, float gravity = 9.8f) {
    player.y -= initialVelocity * player.time - 0.5f * gravity * player.time *
player.time;
    player.time += 0.1f;
}

// Βολή με κλίση
void inclinedThrow(Player& player, float initialVelocity, float angle, float gravity =
9.8f) {
    float radAngle = angle * M_PI / 180.0f;
    player.x += initialVelocity * cos(radAngle) * player.time;
    player.y -= (initialVelocity * sin(radAngle) * player.time - 0.5f * gravity *
player.time * player.time);
    player.time += 0.1f;
}

// Ελλειπτική κίνηση με σταθερή ταχύτητα
void ellipticalMotion(Player& player, float centerX, float centerY, float a, float b,
float speed) {
    player.angle += speed;
```

```
    player.x = centerX + a * cos(player.angle * M_PI / 180.0f);
    player.y = centerY + b * sin(player.angle * M_PI / 180.0f);
}

// Σχεδίαση του παίκτη ως κύκλου
void drawPlayer(const Player& player) {
    graphics::Brush brush;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 1.0f;
    graphics::drawDisk(player.x, player.y, player.radius, brush);
}

// Επαναφορά του χρόνου
void resetTime(Player& player) {
    player.time = 0.0f;
}

#endif // PLAYER_ORBIT_GR_H
```

## Πρόγραμμα main.cpp για χρήση των συναρτήσεων

Το παρακάτω πρόγραμμα δείχνει την εφαρμογή των παραπάνω κινήσεων με χρήση των βελών του πληκτρολογίου για εναλλαγή κινήσεων.

```
#include "Player_orbit.h"
#include "sgg/graphics.h"
#include <iostream>

// Ρυθμίσεις παραθύρου
const float WINDOW_WIDTH = 800.0f;
const float WINDOW_HEIGHT = 600.0f;

// Αρχικοποίηση του παίκτη
Player player = { WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 15.0f, 2.0f, 0.0f, 0.0f };
int movementType = 0; // Τύπος κίνησης

// Συνάρτηση σχεδίασης
void draw() {
    // Καθαρισμός παραθύρου
    graphics::Brush bgBrush;
    bgBrush.fill_color[0] = 1.0f;
    bgBrush.fill_color[1] = 1.0f;
    bgBrush.fill_color[2] = 1.0f;
    graphics::drawRect(WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, WINDOW_WIDTH,
WINDOW_HEIGHT, bgBrush);

    // Σχεδίαση του παίκτη
    drawPlayer(player);
}

// Συνάρτηση ενημέρωσης για τη μετακίνηση του παίκτη
void update(float ms) {
    if (graphics::getKeyState(graphics::SCANCODE_RIGHT)) movementType = 1;
    if (graphics::getKeyState(graphics::SCANCODE_LEFT)) movementType = 2;
    if (graphics::getKeyState(graphics::SCANCODE_UP)) movementType = 3;
    if (graphics::getKeyState(graphics::SCANCODE_DOWN)) movementType = 4;
    if (graphics::getKeyState(graphics::SCANCODE_SPACE)) movementType = 5;
    if (graphics::getKeyState(graphics::SCANCODE_LSHIFT)) movementType = 6;
    if (graphics::getKeyState(graphics::SCANCODE_TAB)) movementType = 7;

    // Ανάλογα με τον τύπο κίνησης, καλείται η αντίστοιχη συνάρτηση
    switch (movementType) {
    case 1:
        moveHorizontal(player, player.speed);
        break;
    case 2:
        moveVertical(player, player.speed);
        break;
    case 3:
        moveCircular(player, WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 100, 1.0f);
        break;
    case 4:
        freeFall(player);
        break;
    case 5:
        verticalThrow(player, 20.0f);
        break;
    case 6:
        inclinedThrow(player, 20.0f, 45.0f);
        break;
    case 7:
        ellipticalMotion(player, WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2, 150, 100, 1.0f);
    }
```

```

        break;
    }

    // Επαναφορά του παίκτη αν υπερβεί τα όρια
    if (player.x < 0 || player.x > WINDOW_WIDTH || player.y < 0 || player.y >
WINDOW_HEIGHT) {
        player.x = WINDOW_WIDTH / 2;
        player.y = WINDOW_HEIGHT / 2;
        resetTime(player); // Επαναφορά του χρόνου για ελεύθερη πτώση/βολή
    }
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Player's Movements");

    // Ορισμός συνάρτησης σχεδίασης και ενημέρωσης
    graphics::setDrawFunction(draw);
    graphics::setUpdateFunction(update);

    // Έναρξη κύκλου μηνυμάτων
    graphics::startMessageLoop();

    return 0;
}

```

## Επεξήγηση

- Οριζόντια και Κατακόρυφη Κίνηση:** Οι συναρτήσεις `moveHorizontal` και `moveVertical` μετακινούν τον παίκτη είτε κατά μήκος του άξονα x είτε κατά μήκος του άξονα y αντίστοιχα, με βάση την παραμετροποιημένη ταχύτητα.
- Κυκλική Κίνηση:** Η συνάρτηση `moveCircular` μετακινεί τον παίκτη σε κυκλική τροχιά γύρω από το κέντρο του κύκλου, με προκαθορισμένη ακτίνα και ταχύτητα.
- Ελεύθερη Πτώση:** Η συνάρτηση `freeFall` αυξάνει τη θέση του παίκτη στον άξονα y με βάση τον χρόνο και την επιτάχυνση της βαρύτητας.
- Κάθετη και Κεκλιμένη Βολή:** Οι συναρτήσεις `verticalThrow` και `inclinedThrow` υπολογίζουν την κίνηση του παίκτη ανάλογα με την ταχύτητα εκκίνησης και την κλίση της βολής.
- Ελλειπτική Κίνηση:** Η συνάρτηση `ellipticalMotion` υπολογίζει την κίνηση σε ελλειπτική τροχιά γύρω από το κέντρο, βασισμένη σε δύο ημιάξονες *a* και *b*.