CGG\_Code\_Explanation

Here's a detailed explanation of the given code snippet:

**### 1. `void character\_destroy( Character \*character )`**

This function is responsible for freeing up the memory associated with the textures used by a `Character` object. The character's images (for standing, jumping, and walking animations in both directions) are unloaded using `UnloadTexture`, which is a function provided by the \*\*raylib\*\* library to release textures when they are no longer needed.

```c

void character\_destroy( Character \*character ) {

UnloadTexture( character->image.stand\_right );

UnloadTexture( character->image.stand\_left );

UnloadTexture( character->image.jump\_right );

UnloadTexture( character->image.jump\_left );

UnloadTexture( character->image.walk1\_right );

UnloadTexture( character->image.walk1\_left );

UnloadTexture( character->image.walk2\_right );

UnloadTexture( character->image.walk2\_left );

}

```

- \*\*Purpose\*\*: This function is part of memory management to ensure textures used by a character are properly unloaded, preventing memory leaks.

**### 2. `typedef enum { PLATFORM, FLOOR } PlatformType;`**

This `enum` defines different types of platforms that exist in the game. There are two types:

- `PLATFORM`: Represents a regular platform the character can jump onto.

- `FLOOR`: Represents the ground or floor of the game.

**### 3. `typedef struct { ... } Platform;`**

This `struct` defines the properties of a platform object in the game:

- `x` and `y`: Coordinates of the platform's top-left corner.

- `width` and `height`: Dimensions of the platform.

- `type`: The type of platform, either `PLATFORM` or `FLOOR`.

```c

typedef struct {

int x;

int y;

int width;

int height;

PlatformType type;

} Platform;

```

**### 4. `float rand\_float()`**

This is a simple utility function that generates a random float between 0 and 1. It does this by dividing the result of `rand()` (which returns an integer) by `RAND\_MAX` (the maximum possible value of `rand()`).

```c

float rand\_float() {

return (float) ( (float) rand() / (float) RAND\_MAX );

}

```

**### 5. `int character\_on\_platform( Character character, Platform platforms[], int count )`**

This function checks if the character is currently on any platform. It does so by iterating through an array of platforms and checking if the character's rectangle overlaps with any platform's rectangle using `CheckCollisionRecs()` (a raylib function).

```c

int character\_on\_platform( Character character, Platform platforms[], int count ) {

for( int i = 0; i < count; i++ ) {

Rectangle platform\_rec = {

.x = platforms[i].x,

.y = platforms[i].y,

.width = platforms[i].width,

.height = platforms[i].height,

};

Rectangle character\_rec = {

.x = character.x + 10,

.y = character.y + character.height - character.height \* 0.2,

.width = character.width - 15,

.height = character.height \* 0.2 + 1,

};

if( CheckCollisionRecs( character\_rec, platform\_rec ) ) {

return i; // The character is on the platform

}

}

return -1; // The character is not on any platform

}

```

- The `Rectangle` structures represent the character's and platform's areas.

- If a collision is detected, the function returns the index of the platform the character is on.

- If no platform is detected, the function returns `-1`.

### 6. `int main()`

The `main()` function initializes the game window, sets up the characters, and prepares the game loop. Here's a breakdown of what is happening:

1. \*\*Initialize random seed\*\* using `srand(time(NULL));` to ensure the random number generator generates different sequences on each run.

2. \*\*Window setup\*\*: The game window is initialized using `InitWindow` with a resolution of `800 \* SCALE\_FACTOR` by `600 \* SCALE\_FACTOR`, and the FPS is capped at 60 using `SetTargetFPS(60)`.

3. \*\*Character setup\*\*:

- The main character and a monster are created, each with its position, size, velocity, speed, and walking/jumping state.

- The textures for the characters' animations (standing, walking, jumping) are loaded using `character\_image\_load`.

4. \*\*Monster Setup\*\*:

- The monster is created similarly to the character, but it is positioned off-screen (`window\_width + 500`) to enter from the right.

```c

int main() {

srand(time(NULL));

int window\_width = 800 \* SCALE\_FACTOR;

int window\_height = 600 \* SCALE\_FACTOR;

InitWindow( window\_width, window\_height, "My Game" );

SetTargetFPS( 60 );

bool game\_over = false;

CharacterImage character\_image = {0};

character\_image\_load( &character\_image, "img/standing.png", "img/jumping.png", "img/walk1.png", "img/walk2.png", NULL );

CharacterImage monster\_image = {0};

Vector2 monster\_gun\_position = {

.x = 400,

.y = 170,

};

character\_image\_load( &monster\_image, "img/monster\_right\_1.png", "img/monster\_right\_1.png", "img/monster\_right\_1.png", "img/monster\_right\_2.png", &monster\_gun\_position );

Bullets character\_bullets = {0};

Character character = {

.x = window\_width/2,

.y = window\_height/2,

.width = 101,

.height = 260,

.velocity = 200 \* SCALE\_FACTOR,

.speed = 250,

.walking = false,

.jumping = false,

.jump\_strength = 1000,

.image = character\_image,

.jump\_key = KEY\_SPACE,

.direction = 1,

.should\_reappear = false,

.bullets = &character\_bullets,

};

Bullets monster\_bullets = {0};

int monster\_width = 406;

Character monster = {

.x = window\_width + 500,

.y = window\_height/2,

.width = monster\_width,

.height = 339,

.velocity = 200 \* SCALE\_FACTOR,

.speed = 200,

.walking = true,

.jumping = false,

.jump\_strength = 1000,

.image = monster\_image,

.jump\_key = KEY\_F,

.direction = -1,

.should\_reappear = true,

.bullets = &monster\_bullets,

};

}

```

**### \*\*Summary\*\***

- The code sets up a basic platform game where a character can move, jump, and interact with platforms.

- Textures are loaded and unloaded appropriately for the character and monster.

- Collision detection checks whether the character is standing on any platform.

- The `main` function initializes the game window, loads textures for the character and monster, and sets up their properties.

The core logic involves handling texture management, detecting platform collisions, and setting up the game window and character behaviors.