```c
//
//  raycaster.h
//  Raycaster
//
//  Created by ------- on 1/27/24.
//

#ifndef raycaster_h
#define raycaster_h

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_mixer.h>
#include <stdbool.h>

#define SPRITE_IMAGES 9 // Number of images to load into play

// Player info
float PlayerX = 20; // Player position
float PlayerY = 2;
float DirectionX = -1; // Player directional vector
float DirectionY = 0;
float DirectionAbsoluteRadian; // Player's direction vector angle in standard
 position
float PlayerSpeed = 0;
int const FieldOfView = 60; // Field of view in degrees
int PlayerHealth = 100;
float PlayerStamina = 100;
int PlayerAmmo = 10;

// Sprite information. Abusing liberally for global use across functions.
typedef struct {
    int UniqueID; // calling the sprite using this id, because Sprites[i] will
     be sorted constantly
    int ImageID; // what image the sprite uses
    float LocationX;
    float LocationY;
    int SpriteType; // is sprite projectile or enemy?

    // Non-critical info that is declared as needed and needs to be
     transferred across functions
    int SpriteHealth; // ONLY SPRITES
    float Pixel;
    float DistanceToPlayer;
    float AngleToPlayer;
    float PlayerToWallDistance;
    float AngleIfProjectile; // ONLY PROJECTILES
    float SeenByPlayer; // ONLY SPRITES; default false, as soon as true, is
     pernament, enemy sprites start moving to player and firing
    int FireTimer; // ONLY SPRITES to determine interval of firing
} Sprite;
```

```
int NumberOfSprites = 11;
int UniqueIDCounter = 11; // Each sprite is a new unique ID
Sprite* Sprites; // List of sprites
int SpritesCapacity; // Stores sprites memory allocation and reallocates if
 necessary

// Physical window's dimensions for SDL package
int const ScreenWidth = 1280;
int const ScreenHeight = 800;
// Size of world in 0s and 1s
int const MapWidth = 22;
int const MapHeight = 22;


// Iniitalizing global variables used to maintain information across frames +
 used across functions generally created for readability
bool keyState[SDL_NUM_SCANCODES] = {false}; // Generated by language model.
 Handles whether a key is continously being held down
bool GunReload;
float VerticalView = 0; // Bob effect to simulate walking
bool DirectionBobUp = true;
int BobCounter = 0;


bool NoViolence = false; // In case I want to showcase a pure
 raycaster...without the blood and iron. Enabled through F2



int const Map[MapWidth][MapHeight] = // Scans this whether to draw walls or not
{
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1},
    {1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1},
    {1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1},
    {1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
```

```
    {1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};

float CalculateRay(float RayRadian);

void PlayerInput(int *RecoverStaminaTimer);
void PlayerMovement(bool Forward, bool Left, bool Right, bool Backward, bool
 Running);
void HandleBob(void);

void AdjustSpriteInfo(Sprite *Sprites);
int Comparator(const void *a, const void *b);

void DrawMap(Sprite* Sprites, SDL_Renderer* Renderer);
void DrawHUD(SDL_Renderer* Renderer);
void DrawWeapon(SDL_Renderer* Renderer, SDL_Texture* GunTextures[], bool
 *GunShot, int GunShotRenderTime, bool *GunReload, int GunReloadRenderTime);

void InitializeCreateSprite(Sprite* Sprites, int ImageID, float StartX, float
 StartY, int SpriteType, float AngleIfProjectile);
Sprite CreateSprite(Sprite* Sprites, int ImageID, float StartX, float StartY,
 int SpriteType, float AngleIfProjectile);
void HandleSpriteBehavior(Sprite* Sprites, bool *NoViolence, Mix_Chunk*
 FireSound, Mix_Chunk* PotionSound, Mix_Chunk* SpriteDeathSound);
void DestroySprite(Sprite* Sprites, int UniqueID);

#endif /* raycaster_h */
```