For the project for Structured Programming Language, we have chosen to recreate one of the classic games, ” Arkanoid”. It was a very popular arcade game and was later introduced for PCs and Laptops. In order to recreate this nostalgic game, we are going to use C programming language along with the raylib. Raylib is a simple and easy-to-use library for programming languages like C/C++, to enjoy video games programming. Unlike, usual UIs that take time to understand and implement, raylib provides easy-to-understand structures and a makefile that reduces our workload.

So, here is a code to build Arkanoid: 2D classic game, with the help of C language and raylib library:

| #include "raylib.h"  #include <stdio.h> #include <stdlib.h> #include <math.h>  #define PLAYER\_MAX\_LIFE 5 #define LINES\_OF\_BRICKS 7 #define BRICKS\_PER\_LINE 20 #define BG CLITERAL(Color) { 0, 4, 53, 255 }  **typedef** **struct** **Player** {  Vector2 position;  Vector2 size;  **int** life; } Player;  **typedef** **struct** **Ball** {  Vector2 position;  Vector2 speed;  **int** radius;  **bool** active; } Ball;  **typedef** **struct** **Brick** {  Vector2 position;  **bool** active; } Brick;   **int** screenWidth = 800; **int** screenHeight = 450;  **bool** isMenu = true; **bool** gameOver = false; **bool** pause = false; **int** score = 0; Player player = {0}; Ball ball = {0}; Brick brick[LINES\_OF\_BRICKS][BRICKS\_PER\_LINE] = {0}; Vector2 brickSize = {0};  **void** **InitGame**(); // Initialize game **void** **UpdateGame**(); // Update game (one frame) **void** **DrawGame**(); // Draw game (one frame) **void** **UnloadBricks**(); // Unload game **void** **UpdateDrawFrame**(); // Update and Draw (one frame)  **int** **main**() {  InitWindow(screenWidth, screenHeight, "classic game: arkanoid");  InitGame();  SetTargetFPS(60);  **while** (!WindowShouldClose())  {  // Update here  **if** (isMenu)  {  **if** (IsKeyPressed(KEY\_ENTER))  isMenu = false;  }  **else**  {  **if** (IsKeyPressed(KEY\_M))  isMenu = true;  }  UpdateDrawFrame();  }  CloseWindow();  **return** 0; }  **void** **InitGame**(**void**) {  brickSize = (Vector2){GetScreenWidth() / BRICKS\_PER\_LINE, 30};   // Initialize player  player.position = (Vector2){screenWidth / 2, screenHeight \* 7 / 8};  player.size = (Vector2){screenWidth / 10, 20};  player.life = PLAYER\_MAX\_LIFE;   // Initialize ball  ball.position = (Vector2){screenWidth / 2, screenHeight \* 7 / 8 - 30};  ball.speed = (Vector2){0, 0};  ball.radius = 7;  ball.active = false;   // Initialize bricks  **for** (**int** i = 0; i < LINES\_OF\_BRICKS; i++)  {  **for** (**int** j = 0; j < BRICKS\_PER\_LINE; j++)  {  brick[i][j].position = (Vector2){j \* brickSize.x + brickSize.x / 2, i \* brickSize.y + brickSize.y / 2};  brick[i][j].active = true;  }  } }  // Update game (one frame) **void** **UpdateGame**() {  **if** (!gameOver)  {  **if** (IsKeyPressed('P'))  pause = !pause;   **if** (!pause)  {  // Player movement logic  **if** (IsKeyDown(KEY\_LEFT))  player.position.x -= 5;  **if** ((player.position.x - player.size.x / 2) <= 0)  player.position.x = player.size.x / 2;  **if** (IsKeyDown(KEY\_RIGHT))  player.position.x += 5;  **if** ((player.position.x + player.size.x / 2) >= screenWidth)  player.position.x = screenWidth - player.size.x / 2;   // Ball launching logic  **if** (!ball.active)  {  **if** (IsKeyPressed(KEY\_SPACE))  {  ball.active = true;  ball.speed = (Vector2){0, -5};  }  }   // Ball movement logic  **if** (ball.active)  {  ball.position.x += ball.speed.x;  ball.position.y += ball.speed.y;  }  **else**  {  ball.position = (Vector2){player.position.x, screenHeight \* 7 / 8 - 30};  }   // Collision logic: ball vs walls  **if** (((ball.position.x + ball.radius) >= screenWidth) || ((ball.position.x - ball.radius) <= 0))  ball.speed.x \*= -1;  **if** ((ball.position.y - ball.radius) <= 0)  ball.speed.y \*= -1;  **if** ((ball.position.y + ball.radius) >= screenHeight)  {  ball.speed = (Vector2){0, 0};  ball.active = false;  player.life--;  }   // Collision logic: ball vs player  **if** (CheckCollisionCircleRec(ball.position, ball.radius, (Rectangle){player.position.x - player.size.x / 2, player.position.y - player.size.y / 2, player.size.x, player.size.y}))  {  **if** (ball.speed.y > 0)  {  ball.speed.y \*= -1;  ball.speed.x = (ball.position.x - player.position.x) / (player.size.x / 2) \* 5;  }  }   // Collision logic: ball vs bricks  **for** (**int** i = 0; i < LINES\_OF\_BRICKS; i++)  {  **for** (**int** j = 0; j < BRICKS\_PER\_LINE; j++)  {  **if** (brick[i][j].active)  {  // Hit below  **if** (((ball.position.y - ball.radius) <= (brick[i][j].position.y + brickSize.y / 2)) && ((ball.position.y - ball.radius) > (brick[i][j].position.y + brickSize.y / 2 + ball.speed.y)) && ((**fabs**(ball.position.x - brick[i][j].position.x)) < (brickSize.x / 2 + ball.radius \* 2 / 3)) && (ball.speed.y < 0))  {  brick[i][j].active = false;  ball.speed.y \*= -1;  score++;  }  // Hit above  **else** **if** (((ball.position.y + ball.radius) >= (brick[i][j].position.y - brickSize.y / 2)) && ((ball.position.y + ball.radius) < (brick[i][j].position.y - brickSize.y / 2 + ball.speed.y)) && ((**fabs**(ball.position.x - brick[i][j].position.x)) < (brickSize.x / 2 + ball.radius \* 2 / 3)) && (ball.speed.y > 0))  {  brick[i][j].active = false;  ball.speed.y \*= -1;  score++;  }  // Hit left  **else** **if** (((ball.position.x + ball.radius) >= (brick[i][j].position.x - brickSize.x / 2)) && ((ball.position.x + ball.radius) < (brick[i][j].position.x - brickSize.x / 2 + ball.speed.x)) && ((**fabs**(ball.position.y - brick[i][j].position.y)) < (brickSize.y / 2 + ball.radius \* 2 / 3)) && (ball.speed.x > 0))  {  brick[i][j].active = false;  ball.speed.x \*= -1;  score++;  }  // Hit right  **else** **if** (((ball.position.x - ball.radius) <= (brick[i][j].position.x + brickSize.x / 2)) && ((ball.position.x - ball.radius) > (brick[i][j].position.x + brickSize.x / 2 + ball.speed.x)) && ((**fabs**(ball.position.y - brick[i][j].position.y)) < (brickSize.y / 2 + ball.radius \* 2 / 3)) && (ball.speed.x < 0))  {  brick[i][j].active = false;  ball.speed.x \*= -1;  score++;  }  }  }  }   // Game over logic  **if** (player.life <= 0)  gameOver = true;  **else**  {  gameOver = true;   **for** (**int** i = 0; i < LINES\_OF\_BRICKS; i++)  {  **for** (**int** j = 0; j < BRICKS\_PER\_LINE; j++)  {  **if** (brick[i][j].active)  gameOver = false;  }  }  }  }  }  **else**  {  **if** (IsKeyPressed(KEY\_ENTER))  {  InitGame();  gameOver = false;  }  } }  // Draw game (one frame) **void** **DrawGame**() {  BeginDrawing();  **if** (isMenu)  {  ClearBackground(RAYWHITE);  DrawText("ARKAN0ID: 2D Classic GAME", GetScreenWidth() / 2 - 350, GetScreenHeight() / 2, 50, BLACK);  DrawText("Press ENTER to start the GAME", 50, screenHeight - 150, 20, DARKGRAY);  DrawText("Press P to pause the GAME", 50, screenHeight - 125, 20, DARKGRAY);  DrawText("Press M to return to MENU", 50, screenHeight - 100, 20, DARKGRAY);  DrawText("Press ESC to exit the GAME", 50, screenHeight - 75, 20, DARKGRAY);  }  **else**  {  ClearBackground(BG);   **if** (!gameOver)  {  // Draw player bar  Rectangle paddle = {player.position.x - player.size.x / 2, player.position.y - player.size.y / 2, player.size.x, player.size.y / 2};  DrawRectangleRounded(paddle, 10.0, 4, WHITE);   // Draw player lives  **for** (**int** i = 0; i < player.life; i++)  {  Rectangle life = {20 + 40 \* i, screenHeight - 30, 35, 10};  DrawRectangleRounded(life, 10.4, 4, LIGHTGRAY);  }   // Draw ball  DrawCircleV(ball.position, ball.radius, MAROON);   // Draw bricks  UnloadBricks();   **if** (pause)  DrawText("GAME PAUSED", screenWidth / 2 - MeasureText("GAME PAUSED", 40) / 2, screenHeight / 2 - 40, 40, WHITE);  }  **else**  {  **if** (score != (LINES\_OF\_BRICKS \* BRICKS\_PER\_LINE))  {  DrawText("PRESS [ENTER] TO PLAY AGAIN", GetScreenWidth() / 2 - MeasureText("PRESS [ESC..] TO PLAY AGAIN", 20) / 2, GetScreenHeight() / 4 \* 3 - 50, 20, WHITE);  DrawText("PRESS [ESC] TO EXIT the GAME", GetScreenWidth() / 2 - MeasureText("PRESS [ESC..] TO PLAY AGAIN", 20) / 2, GetScreenHeight() / 4 \* 3, 20, WHITE);  // Draw Scoreboard  DrawText(TextFormat("SCORE%4i", score), GetScreenWidth() / 2 - MeasureText("SCORE", 40), 50, 50, WHITE);  }  **else**  {  DrawText("PRESS [ENTER] TO PLAY AGAIN", GetScreenWidth() / 2 - MeasureText("PRESS [ESC..] TO PLAY AGAIN", 20) / 2, GetScreenHeight() / 4 \* 3 - 50, 20, WHITE);  DrawText("PRESS [ESC] TO EXIT the GAME", GetScreenWidth() / 2 - MeasureText("PRESS [ESC..] TO PLAY AGAIN", 20) / 2, GetScreenHeight() / 4 \* 3, 20, WHITE);  // Draw Scoreboard  DrawText(TextFormat("CONGRATULATIONS"), GetScreenWidth() / 2 - 250, 50, 50, WHITE);  DrawText("You Earned the Highest Score!", GetScreenWidth() / 2 - 230, 100, 30, WHITE);  }  }  }  EndDrawing(); }  // Update and Draw (one frame) **void** **UpdateDrawFrame**() {  UpdateGame();  DrawGame(); }  **void** **UnloadBricks**() {  **for** (**int** i = 0; i < LINES\_OF\_BRICKS; i++)  {  **for** (**int** j = 0; j < BRICKS\_PER\_LINE; j++)  {  **if** (brick[i][j].active)  {  **if** (i == 0)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, YELLOW);  }  **else** **if** (i == 1)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, PINK);  }  **else** **if** (i == 2)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, GREEN);  }  **else** **if** (i == 3)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, BLUE);  }  **else** **if** (i == 4)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, PURPLE);  }  **else** **if** (i == 5)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, BROWN);  }  **else** **if** (i == 6)  {  DrawRectangle(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, RED);  }  DrawRectangleLines(brick[i][j].position.x - brickSize.x / 2, brick[i][j].position.y - brickSize.y / 2, brickSize.x, brickSize.y, BG);  }  }  } } |
| --- |