**Minecraft jarock**

#include "renderer.h"﻿#include <vulkan/vulkan.h>﻿#include <GLFW/glfw3.h>﻿#include <vector>﻿#include <cstring>﻿#include <stdexcept>﻿#include <iostream>﻿namespace {﻿GLFWwindow\* window = nullptr;﻿// Core﻿VkInstance instance{};﻿VkSurfaceKHR surface{};﻿VkPhysicalDevice pdev{};﻿uint32\_t gfxQFamily=0, presentQFamily=0;﻿VkDevice device{};﻿VkQueue gfxQ{}, presentQ{};﻿// Swapchain﻿VkSwapchainKHR swapchain{};﻿VkFormat swapFmt{};﻿VkExtent2D swapExtent{};﻿std::vector<VkImage> swapImages;﻿std::vector<VkImageView> swapViews;﻿// Render pass / pipeline﻿VkRenderPass renderPass{};﻿VkPipelineLayout pipeLayout{};﻿VkPipeline pipeline{};﻿// Framebuffers﻿std::vector<VkFramebuffer> framebuffers;﻿// Command pool / buffers / sync﻿VkCommandPool cmdPool{};﻿std::vector<VkCommandBuffer> cmdbufs;﻿std::vector<VkSemaphore> imgAvail, renderDone;﻿std::vector<VkFence> inFlight;﻿size\_t frameIndex=0;﻿const int MAX\_FRAMES=2;﻿// Instance buffer (SSBO) cho vị trí instanced﻿VkBuffer instBuf{};﻿VkDeviceMemory instMem{};﻿uint32\_t instCount=0;﻿VkDescriptorSetLayout dsl{};﻿VkDescriptorPool dsp{};﻿VkDescriptorSet dset{};﻿// Helpers﻿uint32\_t findMemoryType(uint32\_t typeBits, VkMemoryPropertyFlags flags){﻿  VkPhysicalDeviceMemoryProperties mp{};﻿  vkGetPhysicalDeviceMemoryProperties(pdev,&mp);﻿  for(uint32\_t i=0;i<[mp.me](mp.me#mp.me)moryTypeCount;i++)﻿    if((typeBits & (1u<<i)) && ([mp.me](mp.me#mp.me)moryTypes[i].propertyFlags & flags)==flags) return i;﻿  throw std::runtime\_error("No mem type");﻿}﻿void createBuffer(VkDeviceSize sz, VkBufferUsageFlags usage, VkMemoryPropertyFlags props,﻿                  VkBuffer& buf, VkDeviceMemory& mem){﻿  VkBufferCreateInfo bi{VK\_STRUCTURE\_TYPE\_BUFFER\_CREATE\_INFO};﻿  bi.size=sz; bi.usage=usage; bi.sharingMode=VK\_SHARING\_MODE\_EXCLUSIVE;﻿  if(vkCreateBuffer(device,&bi,nullptr,&buf)!=VK\_SUCCESS) throw std::runtime\_error("buf");﻿  VkMemoryRequirements mr{}; vkGetBufferMemoryRequirements(device,buf,&mr);﻿  VkMemoryAllocateInfo ai{VK\_STRUCTURE\_TYPE\_MEMORY\_ALLOCATE\_INFO};﻿  ai.allocationSize=mr.size; [ai.me](ai.me#ai.me)moryTypeIndex=findMemoryType([mr.me](mr.me#mr.me)moryTypeBits, props);﻿  if(vkAllocateMemory(device,&ai,nullptr,&mem)!=VK\_SUCCESS) throw std::runtime\_error("mem");﻿  vkBindBufferMemory(device,buf,mem,0);﻿}﻿void recordCmd(VkCommandBuffer cb, uint32\_t imgIdx){﻿  VkClearValue clear{{{0.07f,0.08f,0.12f,1.f}}};﻿  VkRenderPassBeginInfo rp{VK\_STRUCTURE\_TYPE\_RENDER\_PASS\_BEGIN\_INFO};﻿  rp.renderPass=renderPass; [rp.fr](rp.fr#rp.fr)amebuffer=framebuffers[imgIdx];﻿  rp.renderArea.extent=swapExtent; rp.clearValueCount=1; rp.pClearValues=&clear;﻿  vkCmdBeginRenderPass(cb,&rp,VK\_SUBPASS\_CONTENTS\_INLINE);﻿  vkCmdBindPipeline(cb,VK\_PIPELINE\_BIND\_POINT\_GRAPHICS,pipeline);﻿  vkCmdBindDescriptorSets(cb,VK\_PIPELINE\_BIND\_POINT\_GRAPHICS, pipeLayout,0,1,&dset,0,nullptr);﻿  // Vẽ point list instanced (mỗi instance đại diện 1 mob/item → vertex shader expand thành billboard/mesh)﻿  vkCmdDraw(cb, 1, instCount, 0, 0);﻿  vkCmdEndRenderPass(cb);﻿}﻿} // ns﻿// ================== PUBLIC API ==================﻿bool vk\_init(int width, int height, const char\* title){﻿  if(!glfwInit()) return false;﻿  if(!glfwVulkanSupported()) return false;﻿  glfwWindowHint(GLFW\_CLIENT\_API, GLFW\_NO\_API);﻿  window = glfwCreateWindow(width,height,title,nullptr,nullptr);﻿  // Instance﻿  uint32\_t extCount=0; const char\*\* exts = glfwGetRequiredInstanceExtensions(&extCount);﻿  VkApplicationInfo app{VK\_STRUCTURE\_TYPE\_APPLICATION\_INFO};﻿  app.pApplicationName="Jarock"; app.apiVersion=VK\_API\_VERSION\_1\_2;﻿  VkInstanceCreateInfo ici{VK\_STRUCTURE\_TYPE\_INSTANCE\_CREATE\_INFO};﻿  ici.pApplicationInfo=&app; ici.enabledExtensionCount=extCount; ici.ppEnabledExtensionNames=exts;﻿  if(vkCreateInstance(&ici,nullptr,&instance)!=VK\_SUCCESS) return false;﻿  // Surface﻿  if(glfwCreateWindowSurface(instance,window,nullptr,&surface)!=VK\_SUCCESS) return false;﻿  // Pick device & queues﻿  uint32\_t devCount=0; vkEnumeratePhysicalDevices(instance,&devCount,nullptr);﻿  std::vector<VkPhysicalDevice> devs(devCount); vkEnumeratePhysicalDevices(instance,&devCount,devs.data());﻿  pdev = devs[0];﻿  uint32\_t qCount=0; vkGetPhysicalDeviceQueueFamilyProperties(pdev,&qCount,nullptr);﻿  std::vector<VkQueueFamilyProperties> qprops(qCount); vkGetPhysicalDeviceQueueFamilyProperties(pdev,&qCount,qprops.data());﻿  for(uint32\_t i=0;i<qCount;i++){﻿    VkBool32 present=VK\_FALSE; vkGetPhysicalDeviceSurfaceSupportKHR(pdev,i,surface,&present);﻿    if((qprops[i].queueFlags&VK\_QUEUE\_GRAPHICS\_BIT) && present){ gfxQFamily=presentQFamily=i; break; }﻿  }﻿  float prio=1.f;﻿  VkDeviceQueueCreateInfo qci{VK\_STRUCTURE\_TYPE\_DEVICE\_QUEUE\_CREATE\_INFO};﻿  qci.queueFamilyIndex=gfxQFamily; qci.queueCount=1; qci.pQueuePriorities=&prio;﻿  const char\* devExts[] = {VK\_KHR\_SWAPCHAIN\_EXTENSION\_NAME};﻿  VkDeviceCreateInfo dci{VK\_STRUCTURE\_TYPE\_DEVICE\_CREATE\_INFO};﻿  dci.queueCreateInfoCount=1; dci.pQueueCreateInfos=&qci;﻿  dci.enabledExtensionCount=1; dci.ppEnabledExtensionNames=devExts;﻿  vkCreateDevice(pdev,&dci,nullptr,&device);﻿  vkGetDeviceQueue(device,gfxQFamily,0,&gfxQ);﻿  presentQ=gfxQ;﻿  // Swapchain﻿  VkSurfaceCapabilitiesKHR caps{}; vkGetPhysicalDeviceSurfaceCapabilitiesKHR(pdev,surface,&caps);﻿  uint32\_t fmtCount=0; vkGetPhysicalDeviceSurfaceFormatsKHR(pdev,surface,&fmtCount,nullptr);﻿  std::vector<VkSurfaceFormatKHR> fmts(fmtCount); vkGetPhysicalDeviceSurfaceFormatsKHR(pdev,surface,&fmtCount,fmts.data());﻿  swapFmt = (fmts[0].format==VK\_FORMAT\_UNDEFINED)? VK\_FORMAT\_B8G8R8A8\_UNORM : fmts[0].format;﻿  swapExtent = caps.currentExtent.width!=UINT32\_MAX ? caps.currentExtent : VkExtent2D{(uint32\_t)width,(uint32\_t)height};﻿  VkSwapchainCreateInfoKHR sci{VK\_STRUCTURE\_TYPE\_SWAPCHAIN\_CREATE\_INFO\_KHR};﻿  sci.surface=surface; sci.minImageCount=std::max(2u,caps.minImageCount);﻿  sci.imageFormat=swapFmt; sci.imageColorSpace=fmts[0].colorSpace;﻿  sci.imageExtent=swapExtent; sci.imageArrayLayers=1; sci.imageUsage=VK\_IMAGE\_USAGE\_COLOR\_ATTACHMENT\_BIT;﻿  sci.imageSharingMode=VK\_SHARING\_MODE\_EXCLUSIVE;﻿  sci.preTransform=caps.currentTransform; [sci.com](sci.com#sci.com)positeAlpha=VK\_COMPOSITE\_ALPHA\_OPAQUE\_BIT\_KHR;﻿  sci.presentMode=VK\_PRESENT\_MODE\_FIFO\_KHR; sci.clipped=VK\_TRUE;﻿  vkCreateSwapchainKHR(device,&sci,nullptr,&swapchain);﻿  uint32\_t imgCount=0; vkGetSwapchainImagesKHR(device,swapchain,&imgCount,nullptr);﻿  s[wapImages.resi](wapImages.resi#wapImages.resi)ze(imgCount); vkGetSwapchainImagesKHR(device,swapchain,&imgCount,s[wapImages.data](wapImages.data#wapImages.data)());﻿  s[wapViews.resi](wapViews.resi#wapViews.resi)ze(imgCount);﻿  for(uint32\_t i=0;i<imgCount;i++){﻿    VkImageViewCreateInfo iv{VK\_STRUCTURE\_TYPE\_IMAGE\_VIEW\_CREATE\_INFO};﻿    iv.image=swapImages[i]; iv.viewType=VK\_IMAGE\_VIEW\_TYPE\_2D; iv.format=swapFmt;﻿    iv.subresourceRange.aspectMask=VK\_IMAGE\_ASPECT\_COLOR\_BIT; iv.subresourceRange.levelCount=1; iv.subresourceRange.layerCount=1;﻿    vkCreateImageView(device,&iv,nullptr,&swapViews[i]);﻿  }﻿  // RenderPass﻿  VkAttachmentDescription color{};﻿  color.format=swapFmt; color.samples=VK\_SAMPLE\_COUNT\_1\_BIT;﻿  color.loadOp=VK\_ATTACHMENT\_LOAD\_OP\_CLEAR; color.storeOp=VK\_ATTACHMENT\_STORE\_OP\_STORE;﻿  color.initialLayout=VK\_IMAGE\_LAYOUT\_UNDEFINED; color.finalLayout=VK\_IMAGE\_LAYOUT\_PRESENT\_SRC\_KHR;﻿  VkAttachmentReference colorRef{0,VK\_IMAGE\_LAYOUT\_COLOR\_ATTACHMENT\_OPTIMAL};﻿  VkSubpassDescription sub{}; sub.pipelineBindPoint=VK\_PIPELINE\_BIND\_POINT\_GRAPHICS;﻿  sub.colorAttachmentCount=1; sub.pColorAttachments=&colorRef;﻿  VkRenderPassCreateInfo rpci{VK\_STRUCTURE\_TYPE\_RENDER\_PASS\_CREATE\_INFO};﻿  rpci.attachmentCount=1; rpci.pAttachments=&color; rpci.subpassCount=1; rpci.pSubpasses=&sub;﻿  vkCreateRenderPass(device,&rpci,nullptr,&renderPass);﻿  // Pipeline (vertex = point → billboard/mesh trong VS; fragment tô màu)﻿  // (Nhúng SPIR-V từ headers tạo sẵn: shaders.spv.h)﻿  extern const uint32\_t VS\_CODE[]; extern const size\_t VS\_SIZE;﻿  extern const uint32\_t FS\_CODE[]; extern const size\_t FS\_SIZE;﻿  auto makeShader=[&](const uint32\_t\* code,size\_t bytes){﻿    VkShaderModuleCreateInfo sm{VK\_STRUCTURE\_TYPE\_SHADER\_MODULE\_CREATE\_INFO};﻿    sm.codeSize=bytes; sm.pCode=code;﻿    VkShaderModule m{}; vkCreateShaderModule(device,&sm,nullptr,&m); return m;﻿  };﻿  VkShaderModule vs=makeShader(VS\_CODE,VS\_SIZE), fs=makeShader(FS\_CODE,FS\_SIZE);﻿  VkPipelineShaderStageCreateInfo stg[2]{};﻿  stg[0].sType=VK\_STRUCTURE\_TYPE\_PIPELINE\_SHADER\_STAGE\_CREATE\_INFO; stg[0].stage=VK\_SHADER\_STAGE\_VERTEX\_BIT;   stg[0].module=vs; stg[0].pName="main";﻿  stg[1].sType=VK\_STRUCTURE\_TYPE\_PIPELINE\_SHADER\_STAGE\_CREATE\_INFO; stg[1].stage=VK\_SHADER\_STAGE\_FRAGMENT\_BIT; stg[1].module=fs; stg[1].pName="main";﻿  // No vertex bindings (dùng instanced positions từ SSBO), viewport/scissor dynamic﻿  VkPipelineVertexInputStateCreateInfo vi{VK\_STRUCTURE\_TYPE\_PIPELINE\_VERTEX\_INPUT\_STATE\_CREATE\_INFO};﻿  VkPipelineInputAssemblyStateCreateInfo ia{VK\_STRUCTURE\_TYPE\_PIPELINE\_INPUT\_ASSEMBLY\_STATE\_CREATE\_INFO};﻿  ia.topology=VK\_PRIMITIVE\_TOPOLOGY\_POINT\_LIST;﻿  VkPipelineViewportStateCreateInfo vp{VK\_STRUCTURE\_TYPE\_PIPELINE\_VIEWPORT\_STATE\_CREATE\_INFO};﻿  vp.viewportCount=1; vp.scissorCount=1;﻿  VkPipelineRasterizationStateCreateInfo rs{VK\_STRUCTURE\_TYPE\_PIPELINE\_RASTERIZATION\_STATE\_CREATE\_INFO};﻿  rs.polygonMode=VK\_POLYGON\_MODE\_FILL; rs.cullMode=VK\_CULL\_MODE\_NONE; rs.lineWidth=1.f;﻿  VkPipelineMultisampleStateCreateInfo ms{VK\_STRUCTURE\_TYPE\_PIPELINE\_MULTISAMPLE\_STATE\_CREATE\_INFO};﻿  ms.rasterizationSamples=VK\_SAMPLE\_COUNT\_1\_BIT;﻿  VkPipelineColorBlendAttachmentState cba{}; cba.colorWriteMask=0xF; cba.blendEnable=VK\_FALSE;﻿  VkPipelineColorBlendStateCreateInfo cb{VK\_STRUCTURE\_TYPE\_PIPELINE\_COLOR\_BLEND\_STATE\_CREATE\_INFO};﻿  cb.attachmentCount=1; cb.pAttachments=&cba;﻿  VkDynamicState dyns[2]={VK\_DYNAMIC\_STATE\_VIEWPORT,VK\_DYNAMIC\_STATE\_SCISSOR};﻿  VkPipelineDynamicStateCreateInfo dyn{VK\_STRUCTURE\_TYPE\_PIPELINE\_DYNAMIC\_STATE\_CREATE\_INFO};﻿  dyn.dynamicStateCount=2; dyn.pDynamicStates=dyns;﻿  // Descriptor (binding 0: SSBO positions)﻿  VkDescriptorSetLayoutBinding b0{}; b0.binding=0; b0.descriptorCount=1;﻿  b0.descriptorType=VK\_DESCRIPTOR\_TYPE\_STORAGE\_BUFFER; b0.stageFlags=VK\_SHADER\_STAGE\_VERTEX\_BIT;﻿  VkDescriptorSetLayoutCreateInfo dli{VK\_STRUCTURE\_TYPE\_DESCRIPTOR\_SET\_LAYOUT\_CREATE\_INFO};﻿  dli.bindingCount=1; dli.pBindings=&b0; vkCreateDescriptorSetLayout(device,&dli,nullptr,&dsl);﻿  VkPipelineLayoutCreateInfo pl{VK\_STRUCTURE\_TYPE\_PIPELINE\_LAYOUT\_CREATE\_INFO};﻿  pl.setLayoutCount=1; pl.pSetLayouts=&dsl; vkCreatePipelineLayout(device,&pl,nullptr,&pipeLayout);﻿  VkGraphicsPipelineCreateInfo pci{VK\_STRUCTURE\_TYPE\_GRAPHICS\_PIPELINE\_CREATE\_INFO};﻿  pci.stageCount=2; pci.pStages=stg; pci.pVertexInputState=&vi; pci.pInputAssemblyState=&ia;﻿  pci.pViewportState=&vp; pci.pRasterizationState=&rs; pci.pMultisampleState=&ms;﻿  pci.pColorBlendState=&cb; pci.pDynamicState=&dyn; pci.layout=pipeLayout; pci.renderPass=renderPass;﻿  vkCreateGraphicsPipelines(device,VK\_NULL\_HANDLE,1,&pci,nullptr,&pipeline);﻿  vkDestroyShaderModule(device,vs,nullptr);﻿  vkDestroyShaderModule(device,fs,nullptr);﻿  // Framebuffers﻿  framebuffers.resize(s[wapViews.size](wapViews.size#wapViews.size)());﻿  for(size\_t i=0;i<s[wapViews.size](wapViews.size#wapViews.size)();++i){﻿    VkImageView atts[1]={swapViews[i]};﻿    VkFramebufferCreateInfo fbi{VK\_STRUCTURE\_TYPE\_FRAMEBUFFER\_CREATE\_INFO};﻿    fbi.renderPass=renderPass; fbi.attachmentCount=1; fbi.pAttachments=atts;﻿    fbi.width=s[wapExtent.widt](wapExtent.widt#wapExtent.widt)h; fbi.height=s[wapExtent.heig](wapExtent.heig#wapExtent.heig)ht; fbi.layers=1;﻿    vkCreateFramebuffer(device,&fbi,nullptr,&framebuffers[i]);﻿  }﻿  // Cmd pool/buffers﻿  VkCommandPoolCreateInfo cp{VK\_STRUCTURE\_TYPE\_COMMAND\_POOL\_CREATE\_INFO};﻿  cp.queueFamilyIndex=gfxQFamily; cp.flags=VK\_COMMAND\_POOL\_CREATE\_RESET\_COMMAND\_BUFFER\_BIT;﻿  vkCreateCommandPool(device,&cp,nullptr,&cmdPool);﻿  cmdbufs.resize(MAX\_FRAMES);﻿  VkCommandBufferAllocateInfo cai{VK\_STRUCTURE\_TYPE\_COMMAND\_BUFFER\_ALLOCATE\_INFO};﻿[cai.com](cai.com#cai.com)mandPool=cmdPool; cai.level=VK\_COMMAND\_BUFFER\_LEVEL\_PRIMARY; [cai.com](cai.com#cai.com)mandBufferCount=cmdbufs.size();﻿  vkAllocateCommandBuffers(device,&cai,cmdbufs.data());﻿  imgAvail.resize(MAX\_FRAMES); renderDone.resize(MAX\_FRAMES); inFlight.resize(MAX\_FRAMES);﻿  for(int i=0;i<MAX\_FRAMES;i++){﻿    VkSemaphoreCreateInfo si{VK\_STRUCTURE\_TYPE\_SEMAPHORE\_CREATE\_INFO};﻿    vkCreateSemaphore(device,&si,nullptr,&imgAvail[i]);﻿    vkCreateSemaphore(device,&si,nullptr,&renderDone[i]);﻿    VkFenceCreateInfo fi{VK\_STRUCTURE\_TYPE\_FENCE\_CREATE\_INFO}; fi.flags=VK\_FENCE\_CREATE\_SIGNALED\_BIT;﻿    vkCreateFence(device,&fi,nullptr,&inFlight[i]);﻿  }﻿  // Instance buffer + descriptor﻿  createBuffer(1024\*1024, // 1MB cho demo﻿               VK\_BUFFER\_USAGE\_STORAGE\_BUFFER\_BIT, ﻿               VK\_MEMORY\_PROPERTY\_HOST\_VISIBLE\_BIT|VK\_MEMORY\_PROPERTY\_HOST\_COHERENT\_BIT,﻿               instBuf, instMem);﻿  VkDescriptorPoolSize dps{VK\_DESCRIPTOR\_TYPE\_STORAGE\_BUFFER,1};﻿  VkDescriptorPoolCreateInfo dpci{VK\_STRUCTURE\_TYPE\_DESCRIPTOR\_POOL\_CREATE\_INFO};﻿  dpci.maxSets=1; dpci.poolSizeCount=1; dpci.pPoolSizes=&dps; vkCreateDescriptorPool(device,&dpci,nullptr,&dsp);﻿  VkDescriptorSetAllocateInfo dsai{VK\_STRUCTURE\_TYPE\_DESCRIPTOR\_SET\_ALLOCATE\_INFO};﻿  dsai.descriptorPool=dsp; dsai.descriptorSetCount=1; dsai.pSetLayouts=&dsl; vkAllocateDescriptorSets(device,&dsai,&dset);﻿  VkDescriptorBufferInfo dbi{instBuf,0,VK\_WHOLE\_SIZE};﻿  VkWriteDescriptorSet w{VK\_STRUCTURE\_TYPE\_WRITE\_DESCRIPTOR\_SET};﻿  w.dstSet=dset; w.dstBinding=0; w.descriptorCount=1; w.descriptorType=VK\_DESCRIPTOR\_TYPE\_STORAGE\_BUFFER; w.pBufferInfo=&dbi;﻿  vkUpdateDescriptorSets(device,1,&w,0,nullptr);﻿  return true;﻿}﻿void vk\_upload\_instances(const float\* xyz, int count){﻿  instCount = (count<0)?0:static\_cast<uint32\_t>(count);﻿  if(instCount==0) return;﻿  void\* p=nullptr; vkMapMemory(device,instMem,0,VK\_WHOLE\_SIZE,0,&p);﻿  std::memcpy(p, xyz, sizeof(float)\*3\*instCount);﻿  vkUnmapMemory(device,instMem);﻿}﻿void vk\_draw\_frame(){﻿  glfwPollEvents();﻿  uint32\_t imgIdx=0;﻿  vkWaitForFences(device,1,&inFlight[frameIndex],VK\_TRUE,UINT64\_MAX);﻿  vkResetFences(device,1,&inFlight[frameIndex]);﻿  vkAcquireNextImageKHR(device,swapchain,UINT64\_MAX,imgAvail[frameIndex],VK\_NULL\_HANDLE,&imgIdx);﻿  vkResetCommandBuffer(cmdbufs[frameIndex],0);﻿  VkCommandBufferBeginInfo bi{VK\_STRUCTURE\_TYPE\_COMMAND\_BUFFER\_BEGIN\_INFO};﻿  vkBeginCommandBuffer(cmdbufs[frameIndex],&bi);﻿  // dynamic viewport/scissor﻿  VkViewport vp{0,0,(float)s[wapExtent.widt](wapExtent.widt#wapExtent.widt)h,(float)s[wapExtent.heig](wapExtent.heig#wapExtent.heig)ht,0.f,1.f};﻿  VkRect2D sc{{0,0},swapExtent};﻿  vkCmdSetViewport(cmdbufs[frameIndex],0,1,&vp);﻿  vkCmdSetScissor(cmdbufs[frameIndex],0,1,&sc);﻿  recordCmd(cmdbufs[frameIndex], imgIdx);﻿  vkEndCommandBuffer(cmdbufs[frameIndex]);﻿  VkPipelineStageFlags waitStage = VK\_PIPELINE\_STAGE\_COLOR\_ATTACHMENT\_OUTPUT\_BIT;﻿  VkSubmitInfo si{VK\_STRUCTURE\_TYPE\_SUBMIT\_INFO};﻿  si.waitSemaphoreCount=1; si.pWaitSemaphores=&imgAvail[frameIndex]; si.pWaitDstStageMask=&waitStage;﻿[si.com](si.com#si.com)mandBufferCount=1; si.pCommandBuffers=&cmdbufs[frameIndex];﻿  si.signalSemaphoreCount=1; si.pSignalSemaphores=&renderDone[frameIndex];﻿  vkQueueSubmit(gfxQ,1,&si,inFlight[frameIndex]);﻿  VkPresentInfoKHR pi{VK\_STRUCTURE\_TYPE\_PRESENT\_INFO\_KHR};﻿  pi.waitSemaphoreCount=1; pi.pWaitSemaphores=&renderDone[frameIndex];﻿  pi.swapchainCount=1; VkSwapchainKHR scs[1]={swapchain}; pi.pSwapchains=scs; pi.pImageIndices=&imgIdx;﻿  vkQueuePresentKHR(presentQ,&pi);﻿  frameIndex = (frameIndex+1)%MAX\_FRAMES;﻿}﻿void vk\_shutdown(){﻿  vkDeviceWaitIdle(device);﻿  vkDestroyDescriptorPool(device,dsp,nullptr);﻿  vkDestroyDescriptorSetLayout(device,dsl,nullptr);﻿  vkDestroyBuffer(device,instBuf,nullptr); vkFreeMemory(device,instMem,nullptr);﻿  for(auto fb:framebuffers) vkDestroyFramebuffer(device,fb,nullptr);﻿  vkDestroyPipeline(device,pipeline,nullptr);﻿  vkDestroyPipelineLayout(device,pipeLayout,nullptr);﻿  vkDestroyRenderPass(device,renderPass,nullptr);﻿  for(auto v:swapViews) vkDestroyImageView(device,v,nullptr);﻿  vkDestroySwapchainKHR(device,swapchain,nullptr);﻿  for(int i=0;i<MAX\_FRAMES;i++){ vkDestroySemaphore(device,imgAvail[i],nullptr); vkDestroySemaphore(device,renderDone[i],nullptr); vkDestroyFence(device,inFlight[i],nullptr); }﻿  vkDestroyCommandPool(device,cmdPool,nullptr);﻿  vkDestroyDevice(device,nullptr);﻿  vkDestroySurfaceKHR(instance,surface,nullptr);﻿  vkDestroyInstance(instance,nullptr);﻿  glfwDestroyWindow(window); glfwTerminate();﻿}﻿#version 450﻿layout(std430, binding=0) buffer Instances { vec3 positions[]; };﻿layout(location=0) out vec3 vColor;﻿void main(){﻿    vec3 p = positions[gl\_InstanceIndex];﻿    gl\_Position = vec4(p, 1.0);     // demo: chưa có proj/view → bạn thay bằng MVP﻿    gl\_PointSize = 6.0;﻿    vColor = vec3(0.9, 0.8, 0.2);﻿}﻿#version 450﻿layout(location=0) in vec3 vColor;﻿layout(location=0) out vec4 outColor;﻿void main(){ outColor = vec4(vColor,1.0); }﻿// Cargo.toml: [lib] crate-type=["cdylib"]﻿use std::collections::HashMap;﻿use std::time::{Instant,Duration};﻿use std::ffi::c\_void;﻿#[derive(Clone)]﻿pub struct Mob { pub id:u32, pub x:f32, pub y:f32, pub z:f32, pub vx:f32, pub vy:f32, pub vz:f32, pub hp:f32, pub alive:bool }﻿#[derive(Clone)]﻿pub struct Item{ pub id:u32, pub x:f32, pub y:f32, pub z:f32, pub spawn:Instant, pub picked:bool }﻿static mut MOBS: Option<HashMap<u32,Mob>> = None;﻿static mut ITEMS: Option<HashMap<u32,Item>> = None;﻿#[no\_mangle] pub extern "C" fn rs\_init(){﻿    unsafe{ MOBS=Some(HashMap::new()); ITEMS=Some(HashMap::new()); }﻿}﻿fn height(x:i32,z:i32)->i32{﻿    // value noise siêu đơn giản (demo)﻿    let n = ((x\*[73856093](73856093#73856093)) ^ (z\*[19349663](19349663#19349663))) as i64;﻿    ((n % 11 + 50) as i32)﻿}﻿#[no\_mangle] pub extern "C" fn rs\_world\_spawn(radius:i32){﻿    unsafe{﻿        let mobs = MOBS.as\_mut().unwrap();﻿        let items = ITEMS.as\_mut().unwrap();﻿        let mut idc=1u32;﻿        for x in -radius..=radius {﻿            for z in -radius..=radius {﻿                let y = height(x,z) as f32;﻿                if (x+z) % 13 == 0 {﻿                    mobs.insert(idc, Mob{ id:idc, x:x as f32, y, z:z as f32, vx:0.,vy:0.,vz:0., hp:100., alive:true }); idc+=1;﻿                }﻿                if (x\*z) % 17 == 0 {﻿                    items.insert(idc, Item{ id:idc, x:x as f32+0.2, y, z:z as f32+0.2, spawn:Instant::now(), picked:false }); idc+=1;﻿                }﻿            }﻿        }﻿    }﻿}﻿#[no\_mangle] pub extern "C" fn rs\_mob\_damage(id:u32, dmg:f32){﻿    unsafe {﻿        if let Some(m)=MOBS.as\_mut().unwrap().get\_mut(&id) {﻿            if m.alive { m.hp -= dmg; if m.hp<=0.0 { m.alive=false; } }﻿        }﻿    }﻿}﻿#[no\_mangle] pub extern "C" fn rs\_update(players:\*const (f32,f32,f32), n:usize){﻿    unsafe {﻿        let mobs = MOBS.as\_mut().unwrap();﻿        for m in mobs.values\_mut(){﻿            if !m.alive { continue; }﻿            // gravity﻿            m.vy -= 0.02; m.y += m.vy; if m.y < 0.0 { m.y=0.0; m.vy=0.0; }﻿            // follow nearest﻿            if n>0 {﻿                let mut best=0usize; let mut d2=f32::MAX;﻿                for i in 0..n {﻿                    let (px,py,pz) = \*players.add(i);﻿                    let dd = (m.x-px)\*(m.x-px) + (m.y-py)\*(m.y-py) + (m.z-pz)\*(m.z-pz);﻿                    if dd<d2 { d2=dd; best=i; }﻿                }﻿                let (tx,\_,tz) = \*players.add(best);﻿                let dx=tx-m.x; let dz=tz-m.z; let len=(dx\*dx+dz\*dz).sqrt();﻿                if len>1e-4 { m.vx = dx/len\*0.06; m.vz = dz/len\*0.06; }﻿            }﻿            // avoid each other (simple)﻿            for o in mobs.values(){﻿                if o.id==m.id || !o.alive { continue; }﻿                let dx=m.x-o.x; let dz=m.z-o.z; let d2=dx\*dx+dz\*dz;﻿                if d2<1.2 { m.vx += dx\*0.02; m.vz += dz\*0.02; }﻿            }﻿            // integrate﻿            m.x += m.vx; m.z += m.vz;﻿            // attack if close (demo: giảm hp player server-side sau)﻿        }﻿        // items despawn 5'﻿        let items = ITEMS.as\_mut().unwrap();﻿        let now = Instant::now();﻿        items.retain(|\_,it| !it.picked && now.duration\_since(it.spawn) < Duration::from\_secs(300));﻿        // remove dead mobs﻿        mobs.retain(|\_,m| m.alive);﻿    }﻿}﻿// Xuất positions sang renderer (xyz liên tiếp)﻿#[no\_mangle] pub extern "C" fn rs\_copy\_mob\_positions(out\_xyz:\*mut f32) -> i32 {﻿    unsafe{﻿        let mobs=MOBS.as\_ref().unwrap();﻿        let mut i=0;﻿        for m in mobs.values(){﻿            if !m.alive { continue; }﻿            unsafe{﻿                \*out\_xyz.add(i\*3+0)=m.x;﻿                \*out\_xyz.add(i\*3+1)=m.y;﻿                \*out\_xyz.add(i\*3+2)=m.z;﻿            }﻿            i+=1;﻿        }﻿        i as i32﻿    }﻿}﻿// JarockGame.java﻿import java.nio.\*;﻿import java.util.\*;﻿import java.util.concurrent.\*;﻿public class JarockGame {﻿    static {﻿        System.loadLibrary("jarock\_vk");   // C++ Vulkan﻿        System.loadLibrary("jarock\_rs");   // Rust core﻿    }﻿    // VK﻿    private static native boolean vk\_init(int w,int h,String title);﻿    private static native void vk\_upload\_instances(FloatBuffer xyz, int count);﻿    private static native void vk\_draw\_frame();﻿    private static native void vk\_shutdown();﻿    // RS﻿    private static native void rs\_init();﻿    private static native void rs\_world\_spawn(int radius);﻿    private static native void rs\_update(float[] playerXYZ, int playerCount);﻿    private static native int  rs\_copy\_mob\_positions(FloatBuffer out);﻿    // ---- State ----﻿    private final List<float[]> players = new ArrayList<>();﻿    private final ExecutorService exec = Executors.newFixedThreadPool(2);﻿    // Input (đơn giản – bạn thay bằng GLFW/JNativeHook sau)﻿    volatile boolean W,A,S,D,SPACE;﻿    public void start(){﻿        rs\_init();﻿        rs\_world\_spawn(32);﻿        if(!vk\_init(1280,720,"Jarock Vulkan")) throw new RuntimeException("VK init failed");﻿        // Player 0﻿        players.add(new float[]{0f, 5f, 0f});﻿        // Game loop thread﻿        new Thread(()->{﻿            long last = System.nanoTime();﻿            FloatBuffer mobBuf = ByteBuffer.allocateDirect(3\*4\*65536).order(ByteOrder.nativeOrder()).asFloatBuffer();﻿            float[] pflat = new float[3];﻿            while(true){﻿                // Input → move player 0﻿                float[] p0 = players.get(0);﻿                if(W) p0[2]+=0.15f;﻿                if(S) p0[2]-=0.15f;﻿                if(A) p0[0]-=0.15f;﻿                if(D) p0[0]+=0.15f;﻿                if(SPACE) p0[1]+=0.2f; // demo jump (không physics client)﻿                // Update RS﻿                pflat[0]=p0[0]; pflat[1]=p0[1]; pflat[2]=p0[2];﻿                rs\_update(pflat,1);﻿                // Copy mob positions → upload VK﻿                mobBuf.clear();﻿                int n = rs\_copy\_mob\_positions(mobBuf);﻿                mobBuf.limit(n\*3);﻿                vk\_upload\_instances(mobBuf, n);﻿                // Draw﻿                vk\_draw\_frame();﻿                // Simple frame cap ~60﻿                try { Thread.sleep(16); } catch(Exception ignore){}﻿                long now = System.nanoTime();﻿                last = now;﻿            }﻿        },"game").start();﻿    }﻿    // Fake input toggles (thay bằng key listener thực sự)﻿    public void setKey(char key, boolean down){﻿        switch(Character.toUpperCase(key)){﻿            case 'W': W=down; break; case 'S': S=down; break;﻿            case 'A': A=down; break; case 'D': D=down; break;﻿            case ' ': SPACE=down; break;﻿        }﻿    }﻿    public static void main(String[] args){﻿        JarockGame g = new JarockGame();﻿        g.start();﻿        // Demo input: bạn thay bằng layer input thật﻿        new Timer().scheduleAtFixedRate(new TimerTask(){ public void run(){﻿            // no-op; giữ JVM sống﻿        }},0,1000);﻿    }﻿}﻿#[repr(C)]﻿pub struct ItemStack { pub item\_id:u16, pub count:u16 }﻿#[no\_mangle] pub extern "C" fn inv\_add\_item(inv:\*mut ItemStack, cap:usize, item\_id:u16, add:u16) -> u16 {﻿    // trả về còn thừa﻿    let slots = unsafe { std::slice::from\_raw\_parts\_mut(inv, cap) };﻿    let mut remain = add;﻿    // stack lên cùng item﻿    for s in slots.iter\_mut(){﻿        if remain==0 { break; }﻿        if s.item\_id==item\_id && s.count<64 {﻿            let space = 64 - s.count;﻿            let put = space.min(remain);﻿            s.count += put; remain -= put;﻿        }﻿    }﻿    // điền vào slot trống﻿    for s in slots.iter\_mut(){﻿        if remain==0 { break; }﻿        if s.count==0 { let put = remain.min(64); \*s = ItemStack{item\_id, count:put}; remain -= put; }﻿    }﻿    remain﻿}﻿// Crafting mẫu: 2 gỗ → 4 plank﻿#[no\_mangle] pub extern "C" fn craft\_wood\_to\_plank(inv:\*mut ItemStack, cap:usize) -> bool {﻿    let slots = unsafe { std::slice::from\_raw\_parts\_mut(inv, cap) };﻿    // tìm 2 gỗ (id=1)﻿    let mut need=2usize;﻿    for s in slots.iter\_mut(){﻿        if s.item\_id==1 && s.count>0 {﻿            let take = need.min(s.count as usize);﻿            s.count -= take as u16; need -= take;﻿            if need==0 { break; }﻿        }﻿    }﻿    if need==0 {﻿        // add plank (id=2) \*4﻿        let rem = inv\_add\_item(inv, cap, 2, 4);﻿        if rem>0 { /\* thả rơi ra đất nếu đầy \*/ }﻿        true﻿    } else {﻿        false﻿    }﻿}﻿#[repr(C)]﻿pub struct ItemStack { pub item\_id:u16, pub count:u16 }﻿#[no\_mangle] pub extern "C" fn inv\_add\_item(inv:\*mut ItemStack, cap:usize, item\_id:u16, add:u16) -> u16 {﻿    // trả về còn thừa﻿    let slots = unsafe { std::slice::from\_raw\_parts\_mut(inv, cap) };﻿    let mut remain = add;﻿    // stack lên cùng item﻿    for s in slots.iter\_mut(){﻿        if remain==0 { break; }﻿        if s.item\_id==item\_id && s.count<64 {﻿            let space = 64 - s.count;﻿            let put = space.min(remain);﻿            s.count += put; remain -= put;﻿        }﻿    }﻿    // điền vào slot trống﻿    for s in slots.iter\_mut(){﻿        if remain==0 { break; }﻿        if s.count==0 { let put = remain.min(64); \*s = ItemStack{item\_id, count:put}; remain -= put; }﻿    }﻿    remain﻿}﻿// Crafting mẫu: 2 gỗ → 4 plank﻿#[no\_mangle] pub extern "C" fn craft\_wood\_to\_plank(inv:\*mut ItemStack, cap:usize) -> bool {﻿    let slots = unsafe { std::slice::from\_raw\_parts\_mut(inv, cap) };﻿    // tìm 2 gỗ (id=1)﻿    let mut need=2usize;﻿    for s in slots.iter\_mut(){﻿        if s.item\_id==1 && s.count>0 {﻿            let take = need.min(s.count as usize);﻿            s.count -= take as u16; need -= take;﻿            if need==0 { break; }﻿        }﻿    }﻿    if need==0 {﻿        // add plank (id=2) \*4﻿        let rem = inv\_add\_item(inv, cap, 2, 4);﻿        if rem>0 { /\* thả rơi ra đất nếu đầy \*/ }﻿        true﻿    } else {﻿        false﻿    }﻿}﻿// NetServer.java (server tick đơn giản)﻿import [java.net](java.net#java.net).\*;﻿import java.nio.\*;﻿import java.nio.channels.\*;﻿import java.util.\*;﻿public class NetServer {﻿  private final DatagramChannel ch;﻿  private final ByteBuffer buf = ByteBuffer.allocateDirect(1400);﻿  public NetServer(int port) throws Exception{﻿    ch = DatagramChannel.open(StandardProtocolFamily.INET);﻿    ch.bind(new InetSocketAddress(port));﻿    ch.configureBlocking(false);﻿  }﻿  public void tick(){﻿    buf.clear();﻿    SocketAddress from = ch.receive(buf);﻿    if(from!=null){﻿      buf.flip();﻿      byte type = buf.get();﻿      if(type==1){ // C2S: player pos﻿        float x=buf.getFloat(), y=buf.getFloat(), z=buf.getFloat();﻿        // cập nhật vị trí player vào state server, rồi broadcast﻿      }﻿      // S2C: ví dụ gửi snapshot mobs﻿      ByteBuffer out = ByteBuffer.allocateDirect(256);﻿      out.put((byte)2); // packet type﻿      out.putInt(0);    // mobs count (demo)﻿      out.flip();﻿      try { ch.send(out, from); } catch(Exception ignore){}﻿    }﻿  }﻿  public static void main(String[] a) throws Exception {﻿    NetServer s=new NetServer(25565);﻿    while(true){ s.tick(); Thread.sleep(5); }﻿  }﻿}﻿// renderer.cpp﻿#include "renderer.h"﻿void Renderer::createRenderPass() {﻿    VkAttachmentDescription colorAttachment{};﻿    colorAttachment.format = swapchainImageFormat;﻿    colorAttachment.samples = VK\_SAMPLE\_COUNT\_1\_BIT;﻿    colorAttachment.loadOp = VK\_ATTACHMENT\_LOAD\_OP\_CLEAR;﻿    colorAttachment.storeOp = VK\_ATTACHMENT\_STORE\_OP\_STORE;﻿    colorAttachment.initialLayout = VK\_IMAGE\_LAYOUT\_UNDEFINED;﻿    colorAttachment.finalLayout = VK\_IMAGE\_LAYOUT\_PRESENT\_SRC\_KHR;﻿    VkAttachmentReference colorAttachmentRef{};﻿    colorAttachmentRef.attachment = 0;﻿    colorAttachmentRef.layout = VK\_IMAGE\_LAYOUT\_COLOR\_ATTACHMENT\_OPTIMAL;﻿    VkSubpassDescription subpass{};﻿    subpass.pipelineBindPoint = VK\_PIPELINE\_BIND\_POINT\_GRAPHICS;﻿    subpass.colorAttachmentCount = 1;﻿    subpass.pColorAttachments = &colorAttachmentRef;﻿    VkRenderPassCreateInfo renderPassInfo{};﻿    renderPassInfo.sType = VK\_STRUCTURE\_TYPE\_RENDER\_PASS\_CREATE\_INFO;﻿    renderPassInfo.attachmentCount = 1;﻿    renderPassInfo.pAttachments = &colorAttachment;﻿    renderPassInfo.subpassCount = 1;﻿    renderPassInfo.pSubpasses = &subpass;﻿    if (vkCreateRenderPass(device, &renderPassInfo, nullptr, &renderPass) != VK\_SUCCESS) {﻿        throw std::runtime\_error("failed to create render pass!");﻿    }﻿}﻿void Renderer::recordCommandBuffer(VkCommandBuffer commandBuffer, uint32\_t imageIndex) {﻿    VkCommandBufferBeginInfo beginInfo{};﻿    beginInfo.sType = VK\_STRUCTURE\_TYPE\_COMMAND\_BUFFER\_BEGIN\_INFO;﻿    vkBeginCommandBuffer(commandBuffer, &beginInfo);﻿    VkRenderPassBeginInfo renderPassInfo{};﻿    renderPassInfo.sType = VK\_STRUCTURE\_TYPE\_RENDER\_PASS\_BEGIN\_INFO;﻿    renderPassInfo.renderPass = renderPass;﻿[renderPassInfo.fr](renderPassInfo.fr#renderPassInfo.fr)amebuffer = swapchainFramebuffers[imageIndex];﻿    renderPassInfo.renderArea.offset = {0, 0};﻿    renderPassInfo.renderArea.extent = swapchainExtent;﻿    VkClearValue clearColor = {{{0.1f, 0.1f, 0.2f, 1.0f}}};﻿    renderPassInfo.clearValueCount = 1;﻿    renderPassInfo.pClearValues = &clearColor;﻿    vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK\_SUBPASS\_CONTENTS\_INLINE);﻿    // TODO: Bind pipeline + draw mobs/items﻿    // vkCmdBindPipeline(commandBuffer, VK\_PIPELINE\_BIND\_POINT\_GRAPHICS, graphicsPipeline);﻿    vkCmdEndRenderPass(commandBuffer);﻿    vkEndCommandBuffer(commandBuffer);﻿}﻿// logic.rs﻿use std::time::{Instant, Duration};﻿pub struct Mob {﻿    pub x: f32,﻿    pub y: f32,﻿    pub z: f32,﻿    pub alive: bool,﻿    pub last\_damage: Option<Instant>,﻿}﻿pub struct Item {﻿    pub x: f32,﻿    pub y: f32,﻿    pub z: f32,﻿    pub spawn\_time: Instant,﻿    pub picked: bool,﻿}﻿impl Mob {﻿    pub fn update(&mut self, player\_x: f32, player\_z: f32, dt: f32) {﻿        if !self.alive { return; }﻿        // Simple AI: move towards player﻿        let dx = player\_x - self.x;﻿        let dz = player\_z - self.z;﻿        let len = (dx\*dx + dz\*dz).sqrt();﻿        if len > 0.1 {﻿            self.x += dx / len \* 2.0 \* dt; // 2.0 = speed﻿            self.z += dz / len \* 2.0 \* dt;﻿        }﻿        // Gravity﻿        self.y -= 9.8 \* dt;﻿        if self.y < 0.0 { self.y = 0.0; }﻿        // Die after damage﻿        if let Some(last\_hit) = self.last\_damage {﻿            if last\_hit.elapsed() < Duration::from\_secs(1) {﻿                self.alive = false;﻿            }﻿        }﻿    }﻿    pub fn take\_damage(&mut self) {﻿        self.last\_damage = Some(Instant::now());﻿    }﻿}﻿impl Item {﻿    pub fn update(&mut self) {﻿        if self.picked { return; }﻿        if self.spawn\_time.elapsed() > Duration::from\_secs(300) {﻿            self.picked = true; // Despawn after 1min﻿        }﻿    }﻿}﻿// Game.java﻿public class Game {﻿    private boolean running = true;﻿    public void start() {﻿        Thread logicThread = new Thread(this::gameLogicLoop, "LogicThread");﻿        Thread renderThread = new Thread(this::renderLoop, "RenderThread");﻿        logicThread.start();﻿        renderThread.start();﻿    }﻿    private void gameLogicLoop() {﻿        long lastTime = System.nanoTime();﻿        while (running) {﻿            long now = System.nanoTime();﻿            float dt = (now - lastTime) / 1\_000\_000\_000f;﻿            lastTime = now;﻿            // Call into Rust for mob/item logic﻿            RustBridge.updateEntities(dt);﻿            try { Thread.sleep(2); } catch (InterruptedException e) {}﻿        }﻿    }﻿    private void renderLoop() {﻿        while (running) {﻿            // Call into C++ Vulkan renderer﻿            Renderer.renderFrame();﻿        }﻿    }﻿    public void stop() {﻿        running = false;﻿    }﻿}﻿jarock/﻿├─ native/                # C++ rendering (Vulkan)﻿│  ├─ CMakeLists.txt﻿│  ├─ src/﻿│  │  ├─ jarock\_core.h﻿│  │  ├─ jarock\_core.cpp﻿│  │  ├─ jarock\_renderer.h﻿│  │  ├─ jarock\_renderer.cpp﻿│  ├─ shaders/﻿│  │  ├─ vs\_chunk.spv﻿│  │  ├─ fs\_chunk.spv﻿│  ├─ textures/﻿│  │  └─ atlas.png﻿│  └─ external/﻿│     ├─ vulkan/﻿│     ├─ FastNoiseLite.h﻿├─ rust/                  # Rust game logic﻿│  ├─ Cargo.toml﻿│  ├─ src/﻿│  │  ├─ lib.rs﻿│  │  ├─ logic.rs﻿│  │  ├─ shared.rs  # New: Shared memory﻿├─ csharp/                # C# gameplay/tools﻿│  ├─ JarockGame.csproj﻿│  ├─ src/﻿│  │  ├─ Program.cs﻿│  │  ├─ ModApi.cs﻿├─ java/                  # Java orchestration﻿│  ├─ build.gradle﻿│  ├─ lib/﻿│  │  ├─ jarock\_core.so﻿│  │  ├─ jarock\_rust.so﻿│  │  └─ mono.dll﻿│  └─ src/main/java/com/jarock/﻿│     ├─ engine/﻿│     │  ├─ Engine.java﻿│     │  ├─ NativeBridge.java﻿│     │  ├─ math/Vec3.java﻿│     │  ├─ ecs/World.java﻿│     │  ├─ ecs/Transform.java﻿│     │  ├─ ecs/Health.java﻿│     │  ├─ ecs/Velocity.java﻿│     │  ├─ ecs/Faction.java﻿│     │  ├─ ecs/AIThink.java﻿│     │  ├─ ecs/ItemStack.java﻿│     │  ├─ ecs/SpatialGrid.java﻿│     │  ├─ ecs/SharedState.java  # New: Shared memory﻿│     │  ├─ status/Status.java﻿│     │  ├─ combat/CombatMath.java﻿│     │  ├─ cmd/CommandRouter.java﻿│     ├─ modapi/﻿│     │  ├─ Block.java﻿│     │  ├─ Entity.java﻿│     │  ├─ Spell.java﻿│     │  ├─ Item.java﻿│     │  ├─ MobType.java﻿│     │  ├─ events/EventBus.java﻿│     │  ├─ registry/Registry.java﻿│     │  ├─ loader/ModLoader.java﻿│     │  ├─ loader/ScriptLoader.java﻿│     ├─ world/﻿│     │  ├─ WorldGen.java﻿│     │  ├─ Re[almManager.java](almManager.java#almManager.java)﻿│     ├─ net/﻿│     │  ├─ UdpServer.java﻿│     │  ├─ UdpClient.java﻿│     ├─ nativecore/﻿│     │  ├─ NativeCore.java﻿│     │  ├─ RustCore.java﻿│     │  └─ CSharpCore.java﻿│     └─ samplemods/﻿│        ├─ HolyLightSpell.java﻿│        ├─ DarkBindingSpell.java﻿│        ├─ items/AdrenalineSyringe.java﻿│        └─ mobs/MiniOceanDragon.java﻿├─ js/                    # JavaScript modding﻿│  ├─ package.json﻿│  ├─ mods/﻿│  │  ├─ holy\_light.js﻿│  │  ├─ dark\_binding.js﻿│  │  └─ mini\_ocean\_dragon.js﻿├─ JarockExtremeDatapack/ # Vanilla-compatible datapack﻿│  ├─ pack.mcmeta﻿│  ├─ data/﻿│  │  ├─ minecraft/﻿│  │  │  └─ tags/functions/load.json﻿│  │  └─ jarock/﻿│  │     ├─ functions/﻿│  │     │  ├─ load.mcfunction﻿│  │     │  ├─ tick.mcfunction﻿│  │     │  ├─ spells/holy\_light.mcfunction﻿│  │     │  ├─ spells/dark\_binding.mcfunction﻿│  │     │  ├─ boss/ocean\_mini\_dragon\_spawn.mcfunction﻿│  │     │  ├─ boss/ocean\_mini\_dragon\_tick.mcfunction﻿│  │     │  ├─ items/give\_adrenaline.mcfunction﻿│  │     │  └─ util/damage\_aoe.mcfunction﻿│  │     ├─ loot\_tables/﻿│  │     │  └─ mobs/ocean\_mini\_dragon.json﻿│  │     ├─ tags/entity\_types/undead.json﻿│  │     ├─ predicates/is\_player.json﻿│  │     └─ advancements/jarock/root.json﻿├─ README.md﻿├─ build.sh﻿package com.jarock.ecs;﻿import java.nio.ByteBuffer;﻿public final class SharedState {﻿    private final ByteBuffer buffer;﻿    private final int maxEntities;﻿    private static final int ENTITY\_STRIDE = 16; // id (4), x,y,z (12)﻿    public SharedState(int maxEntities) {﻿        this.maxEntities = maxEntities;﻿        this.buffer = ByteBuffer.allocateDirect(maxEntities \* ENTITY\_STRIDE);﻿    }﻿    public void updateEntity(int id, float x, float y, float z) {﻿        int offset = id \* ENTITY\_STRIDE;﻿        if (offset + ENTITY\_STRIDE <= buffer.capacity()) {﻿            buffer.putInt(offset, id);﻿            buffer.putFloat(offset + 4, x);﻿            buffer.putFloat(offset + 8, y);﻿            buffer.putFloat(offset + 12, z);﻿        }﻿    }﻿    public ByteBuffer getBuffer() { return buffer; }﻿    public int getMaxEntities() { return maxEntities; }﻿}