WYKŁAD 10

Programowanie Gier Mobilnych

Przedmiot: Programowanie Aplikacji Mobilnych

dr inż. Mateusz Pomianek

Silniki, Fizyka, Grafika, AI, Optymalizacja

- Unity & Unreal Engine
- Grafika 2D i 3D / Shadery
- Fizyka i Kolizje
- AI w Grach Mobilnych
- Optymalizacja i Monetyzacja

Plan Wykładu

Slajdy 1–3

Przemysł gier mobilnych - rynek, trendy, historia

Slajdy 4–6

Silniki gier: Unity, Unreal, Godot, Cocos

Slajdy 7–9

Grafika 2D i 3D - rendering, shadery, animacje

Slajdy 10–12

Fizyka gier - silniki, kolizje, symulacje

Slajdy 13–15

AI i proceduralne generowanie świata

Slajdy 16–18

Wejście użytkownika, UI/UX i dźwięk

Slajdy 19–22

Optymalizacja i profilowanie na mobile

Slajdy 23–25

Sieciowość - multiplayer i backend

Slajdy 26–28

Monetyzacja, dystrybucja, live ops

Slajdy 29–30

Case Studies i podsumowanie

Rynek gier mobilnych: skala i trendy

Gry mobilne to największy segment branży gamingowej - przekraczający łącznie konsole i PC

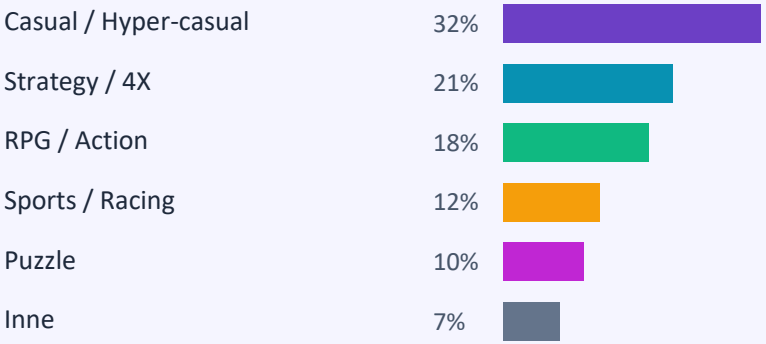
$98.7 mld
Przychody gier mobilnych 2024

3.2 mld
Graczy mobilnych na świecie

57%
Udział mobile w całym rynku gier

91 min
Średni czas gry dziennie na użytkownika

Podział przychodów według gatunku (2024)



Kluczowe trendy 2024–2026

- Generatywna AI w content creation
- Cloud gaming (GeForce Now Mobile)
- Cross-platform play (PC ↔ Mobile)
- Subscription models (Apple Arcade, GP Pass)
- AR/MR gaming (Vision Pro, AR glasses)

Silniki Gier Mobilnych - Przegląd i Porównanie

Unity

C#

Udział rynku: 72%

Zalety: Największy ekosystem, Asset Store, szeroka dokumentacja, DOTS/ECS, ML-Agents

Wady: Runtime overhead, licencjonowanie (runtime fee 2024)

Platformy: iOS, Android, PC, Console, WebGL

Unreal Engine 5

C++ / Blueprint

Udział rynku: 13%

Zalety: Nanite, Lumen, fotorealizm, MetaHuman, Chaos Physics - AAA quality

Wady: Wysoki koszt sprzętowy, trudna optymalizacja mobile

Platformy: iOS, Android (ograniczenia), PC, Console

Godot 4

GDScript / C# / C++

Udział rynku: 8%

Zalety: Open source (MIT), lekki, szybki start, Vulkan renderer, 2D pierwszej klasy

Wady: Mniejszy ekosystem, brak AAA mobile case studies

Platformy: iOS, Android, PC, Web

Cocos Creator

TypeScript / JS

Udział rynku: 5%

Zalety: Dominacja rynku chińskiego, optymalizacja dla casual mobile, mini-games (WeChat)

Wady: Mniejsza społeczność poza Azją

Platformy: iOS, Android, H5, Mini-games

Architektura silnika Unity i cykl życia sceny

Model GameObject–Component

GameObject

Kontener bez zachowania - jedynie transformacja (pozycja, rotacja, skala) w hierarchii sceny. Wszystko w Unity jest GameObject.

Component (MonoBehaviour)

Zachowanie przypisane do obiektu. Klasy dziedziczące z MonoBehaviour implementują Start(), Update(), FixedUpdate() i obsługują zdarzenia.

ScriptableObject

Dane niezwiązane z sceną - konfiguracja, balans gry, definicje przedmiotów. Redukuje coupling między komponentami.

DOTS / ECS (Data-Oriented Tech)

Entity Component System - dane liniowo w pamięci, burst compilation, job system. 10–100× szybszy niż klasyczne MonoBehaviour dla dużych liczb obiektów.

Cykl życia MonoBehaviour

`Awake()`
1× - inicjalizacja referencji, niezależnie od stanu aktywności

`OnEnable()`
przy włączeniu obiektu - resetowanie stanu, subskrypcje zdarzeń

`Start()`
1× przed pierwszą klatką - setup zależny od innych komponentów

`FixedUpdate()`
stały timestep (domyślnie 50Hz) - fizyka, ruchy

`Update()`
każda klatka - logika gry, wejście użytkownika

`LateUpdate()`
po wszystkich Update() - kamera, finalizacja pozycji

`OnDisable() / OnDestroy()`
cleanup - unsubscribe, zwalnianie zasobów

skryptowanie C# i wzorce projektowe w grach mobilnych na Unity

Stosowanie wzorców projektowych w grach poprawia testowalność, redukuje coupling i ułatwia skalowanie projektu

Singleton (GameManager)

Globalny dostęp do menedżerów (Audio, Scene, Save). Używaj ostrożnie - utrudnia testy. Prefer Dependency Injection.

Observer / Event System

UnityEvent, C# events, ScriptableObject events. Decoupling między UI a logiką gry - UI nasłuchuje na zmianę stanu.

State Machine (Enum / Class)

Animator State Machine dla animacji, własny FSM dla AI przeciwnika i ekranów menu. Czytelna nawigacja stanów.

Object Pooling

Kluczowe dla mobile - unikanie GC Alloc przez recykling obiektów. Unity 2021+ ma wbudowany ObjectPool<T>.

Command Pattern

Historia akcji, undo/redo (puzzle games), replay systems, command queue dla AI.

Service Locator / DI

Zenject / VContainer - IoC container dla Unity. Wstrzykiwanie zależności zamiast Find(), zwiększa testowalność.

Zasada: preferuj kompozycję nad dziedziczenie. Unity favors component-based design, nie class hierarchies

Grafika 2D: Sprite, Tilemap i Animacje

2D pozostaje dominującym wyborem dla gier mobilnych ze względu na niższe wymagania sprzętowe i niższy koszt produkcji contentu

Pipeline grafiki 2D w Unity

Sprite Atlas (Sprite Packer)
Scalanie tekstur w atlasy - redukcja draw calls z N do 1 dla elementów tego samego atlasu. Krytyczne dla mobile.

Sprite Renderer
Komponent renderujący sprite - warstwy sortowania (Sorting Layer, Order in Layer), kolor i materiał.

Tilemap + Rule Tiles
Automatyczne kafelkowanie - rule tiles generują odpowiedni sprite na podstawie sąsiadów. Grid 2D, Isometric, Hexagonal.

Animator / Spine / DragonBones
Animator Controller (stany + blend trees). Spine 2D / DragonBones - animacja szkieletowa ze zredukowaną liczbą spritów.

2D Lighting (URP)
Normal maps dla 2D, Point Light, Global Light - efekt głębi bez 3D. Universal Render Pipeline.

Optymalizacja Draw Calls

C#
```
// Sprite Atlas - programowe ładowanie
var spriteAtlas = Resources.Load<SpriteAtlas>(
    "UI/MainAtlas");
var sprite = spriteAtlas
    .GetSprite("btn_play");
GetComponent<Image>().sprite = sprite;

// GPU Instancing - wiele identycznych obj.
var mat = GetComponent<Renderer>().material;
mat.enableInstancing = true;

// SpriteRenderer.drawCallCount
// (Frame Debugger → sprawdź batching)
Debug.Log(Camera.main.commandBufferCount);
```

Zasady optymalizacji 2D

- Maksymalnie 1024×1024 px atlas per scena
- Wyłącz Read/Write Enable na teksturach (–50% RAM)
- Mip-maps WYŁĄCZONE dla UI 2D (zbędne)
- Sprite compression: ASTC dla iOS/Android

Grafika 3D - Rendering, PBR i Shadery na mobile

Rendering mobilny to sztuka kompromisu między jakością wizualną a wydajnością na ograniczonym GPU/SoC

Ścieżki renderowania Unity

Built-in RP
Legacy - pełna kompatybilność, ale brak optymalizacji mobilnych. Nie polecany dla nowych projektów.

Universal RP (URP)
Rekomendowany dla mobile - single-pass rendering, GPU Instancing, optimized batching, 2D Renderer. Znacząco lepszy performance.

High Definition RP (HDRP)
PC/Console - ray tracing, volumetric lighting, fotorealizm. NIE nadaje się dla mobile (zbyt wymagający).

PBR (Physically Based Rendering) na Mobile

Albedo / Base Color
Kolor bazowy powierzchni bez oświetlenia. Format: sRGB, maks. 512×512 px na mobile.

Metallic + Smoothness
Upakowane w jedną teksturę - metallic w kanale R, smoothness w A. Redukcja samplerów.

Normal Map
Iluzja geometrii - tangens space, DirectX vs OpenGL konwencja (Unity używa OpenGL Y+). Max 256×256 mobile.

Occlusion Map
Ambient occlusion baked - cienie w szczelinach bez kosztownych obliczeń RT.

Shader Graph / HLSL - prosty Toon Shader

HLSL

```
Shader "Mobile/ToonLit" {
  Properties {
    _MainTex ("Texture", 2D) = "white" {}
    _RampTex ("Toon Ramp", 2D) = "white" {}
    _OutlineWidth ("Outline", Range(0,0.1)) = 0.02
  }
  CGPROGRAM
  #pragma surface surf ToonRamp
  fixed4 LightingToonRamp(SurfaceOutput o,
      fixed3 lightDir, fixed atten) {
    float diff = dot(o.Normal, lightDir);
    float h = diff * 0.5 + 0.5;
    float2 rh = float2(h, 0.5);
    fixed4 c;
    c.rgb = o.Albedo * tex2D(_RampTex, rh).rgb;
    return c;
  }
  ENDCG
}
```

Animacje, VFX i systemy cząsteczkowe

Animacje i efekty wizualne są kluczowe dla game feel. Muszą być efektowne przy minimalnym koszcie obliczeniowym

Systemy animacji

Mechanim / Animator
Graf stanów animacji - stany (Idle, Run, Jump), przejścia z warunkami (parametry: float, bool, trigger). Blend Tree dla smooth blending kierunków ruchu.

Animation Rigging (IK)
Inverse Kinematics - naturalny ruch kończyn, ograniczenia joints. Proceduralne dostosowanie animacji do geometrii terenu.

Timeline + Cinemachine
Kinematyki - wyreżyserowane sekwencje (cutscenes), kamera Cinemachine z Virtual Cameras, damping, noise dla organic feel.

DOTween (Tweening Library)
Najpopularniejsza biblioteka tweenów - płynne animacje pozycji, skali, koloru bez Animatora. Sekwencje, callback, ease functions.

VFX Graph i Particle System

Shuriken (Legacy PS)
Klasyczny system cząsteczkowy - CPU-bound, max ~1000 cząstek na mobile. Over lifetime modules: color, size, velocity.

VFX Graph (GPU)
GPU compute - miliony cząsteczek na PC/high-end mobile. Wymaga URP/HDRP i Compute Shader support (OpenGL ES 3.1+).

Shader-based Effects
Wiele efektów jako shadery zamiast cząsteczek - dissolve, hologram, force field - zero overhead CPU.

⚡ Performance tips: bake animacje gdy możliwe · GPU Skinning w Player Settings · ogranicz Particle System do max 300 cząstek na mobile · użyj LOD dla animowanych postaci

Fizyka gier: silniki i symulacje

Realistyczna fizyka to fundament immersji w grach akcji, platformówkach i symulatorach - wymaga starannej konfiguracji na mobile

PhysX (Unity 3D)

- Rigidbody z mass, drag, angular drag
- Collider types: Box, Sphere, Capsule, Mesh, Convex
- Joints: Fixed, Hinge, Spring, ConfigurableJoint
- Physics Materials: friction, bounciness
- Layer Collision Matrix - selektywne kolizje

Box2D (Unity 2D)

- Rigidbody2D - kinematic, dynamic, static
- Composite Collider, Edge Collider, Polygon
- DistanceJoint2D, WheelJoint2D, SliderJoint2D
- Effectors: AreaEffector, SurfaceEffector (conveyor)
- PhysicsMaterial2D - tarcie i odbicie 2D

Havok Physics (ECS)

- Deterministyczna fizyka dla multi-player
- Integracja z DOTS/ECS - dane liniowo w pamięci
- Znacząco wydajniejsza niż PhysX dla dużych scen
- Burst Compiler - SIMD optymalizacje
- Stationary sleep - uśpienie nieaktywnych ciał

Sterowanie postacią z Rigidbody

```C#
void FixedUpdate() {
  // Ruch na podstawie wejścia
  float h = Input.GetAxis("Horizontal");
  float v = Input.GetAxis("Vertical");
  Vector3 move = new Vector3(h, 0, v) * speed;
  rb.MovePosition(rb.position + move * Time.fixedDeltaTime);
  // Skok z detekcją podłoża
  if (isGrounded && Input.GetButtonDown("Jump"))
    rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
}
```

Najlepsze praktyki fizyki mobile

- FixedUpdate dla fizyki (NIE Update)
- Ustaw Fixed Timestep na 0.02 lub 0.033 (mobile)
- Używaj Layer Matrix - pomija zbędne kolizje
- Convex Mesh Collider zamiast Concave (10× szybszy)
- Tryb Continuous Collision Detect tylko dla szybkich obiektów

Detekcja kolizji: algorytmy i implementacja

Algorytmy detekcji kolizji

AABB (Axis-Aligned Bounding Box)

Najprostszy i najszybszy - prostokąt wyrównany do osi. O(1) test. Używany jako broad phase. Wadą jest brak obrotu.

GJK + EPA (Convex Hulls)

Gilbert-Johnson-Keerthi - dokładna detekcja wypukłych brył. Expanding Polytope Algorithm - głębokość penetracji. Używany przez PhysX.

Broad Phase (BVH / SAP)

Bounding Volume Hierarchy i Sweep and Prune - redukcja $O(n^2)$ do $O(n \log n)$. Eliminacja par niekolizyjnych przed dokładną fazą.

Raycast / Overlap

Physics.Raycast, OverlapSphere, SphereCast - dyskretne zapytania do silnika fizyki. Efektywne dla hitscan, pola widzenia, line of sight.

Callbacki kolizji w Unity

C#

```
public class PlayerCollision : MonoBehaviour {
    // --- Fizyczne kolizje (Collider + Rigidbody) ---
    void OnCollisionEnter(Collision col) {
        if (col.gameObject.CompareTag("Enemy")) {
            TakeDamage(col.impulse.magnitude);
        }
        // Punkty styku
        foreach (ContactPoint p in col.contacts)
            Debug.DrawRay(p.point, p.normal, Color.red);
    }
    void OnCollisionStay(Collision col) { /* tarcie */ }
    void OnCollisionExit(Collision col) { /* cleanup */ }

    // --- Triggery (Is Trigger = true) ---
    void OnTriggerEnter(Collider other) {
        if (other.TryGetComponent<Coin>(out var coin))
            coin.Collect();
    }
    // --- 2D wersje (Physics2D) ---
    void OnCollisionEnter2D(Collision2D col) { }
    void OnTriggerEnter2D(Collider2D other) { }
}
```

Raycast - line of sight

C#

```
// Sprawdź czy gracz widzi cel
RaycastHit hit;
int mask = LayerMask.GetMask("Obstacle");
bool hasLOS = !Physics.Raycast(
    transform.position,
    (target.position - transform.position).normalized,
    out hit, viewDistance, mask);
if (hasLOS) AttackPlayer();
```

Proceduralne generowanie świata

Proceduralne generowanie pozwala tworzyć nieograniczoną ilość unikalnego contentu przy minimalnym zasobach artystycznych - fundament roguelite i sandbox games

Szum Perlin / Simplex Noise

Generowanie terenu, map wysokości, tekstur chmur. Octaves + persistence + lacunarity - fraktalne detale. Unity: Mathf.PerlinNoise().

BSP (Binary Space Partitioning)

Rekurencyjny podział przestrzeni - dungeons, mapy pomieszczeń. Każdy liść drzewa BSP to pomieszczenie lub korytarz.

Wave Function Collapse (WFC)

Algorytm generowania map z zachowaniem lokalnej spójności wzorców. Tile-based - analizuje przykład i generuje nieskończone warianty.

L-Systems (Rośliny)

Gramatyki Lindenmayera - formalne systemy przepisywania dla roślin, miast, jaskiń. Rekurencyjne rozwijanie symboli w geometrię 3D.

Delaunay Triangulation

Triangulacja Delaunay'a - generowanie dróg, rzek, rozkład punktów zainteresowania na mapie świata. Optymalne pokrycie.

Marching Cubes / Squares

Ekstrakcja izopowierzchni ze scalar field - generowanie grot, terenu wokseli (jak Minecraft). Voxel terrain z smooth mesh.

Seed-based generation: ten sam seed zawsze generuje identyczny świat - kluczowe dla sharowalnych map i debugowania

Sztuczna Inteligencja w Grach Mobilnych

AI w grach nie wymaga realizmu - wymaga iluzji inteligentnego zachowania przy minimalnym koszcie obliczeniowym

Techniki AI w grach

Finite State Machine (FSM)
Stany (Idle, Patrol, Chase, Attack) z przejściami warunkowanymi. Prosta implementacja, trudna w skalowaniu powyżej ~10 stanów.

Behavior Trees
Hierarchiczna struktura - Selector, Sequence, Decorator, Leaf (Task). Modularność i reużywalność. NodeCanvas, Behavior Designer (Unity).

GOAP (Goal Oriented Action Planning)
Agent planuje sekwencję akcji do osiągnięcia celu. Dynamiczne planowanie ścieżki przez problem space. Używany w F.E.A.R., Tomb Raider.

Utility AI
Ocenianie każdej akcji liczbową wartością użyteczności. Wybór akcji z najwyższym score. Naturalne zachowania, jak NPC w The Sims.

A* vs NavMesh - kiedy co używać

NavMesh (Unity wbudowany)
3D, złożona geometria, dynamiczne agenty, płynny ruch

A* Pathfinding Project
Grid-based 2D/3D, wiele typów agentów, dynamiczne mapy, mobile-friendly

Pathfinding - NavMesh

C#

```
using UnityEngine.AI;

public class EnemyAI : MonoBehaviour {
    NavMeshAgent agent;
    Transform player;
    float attackRange = 2f;

    void Update() {
        float dist = Vector3.Distance(
            transform.position, player.position);

        if (dist > attackRange) {
            // Pathfinding do gracza
            agent.SetDestination(player.position);
            ChangeState(State.Chase);
        } else {
            agent.isStopped = true;
            ChangeState(State.Attack);
        }
    }
    // Dynamiczne przeszkody: NavMeshObstacle
    // NavMesh.SamplePosition() dla cover points
}
```

Machine Learning w grach - ML-Agents i uczenie przez wzmocnienie

Unity ML-Agents Toolkit umożliwia trenowanie agentów z użyciem reinforcement learning i imitacji - tworzenie nieprzewidywalnych, adaptacyjnych przeciwników

Agent

Byt podejmujący decyzje na podstawie obserwacji środowiska. Implementuje metodę OnActionReceived() i CollectObservations().

Środowisko (Environment)

Symulacja gry dostarczająca stany (observations) i nagrody (rewards). Może działać jako wiele równoległych instancji (multi-environment training).

Policy (Polityka)

Sieć neuronowa mapująca obserwacje na akcje. Proximal Policy Optimization (PPO) - domyślny algorytm ML-Agents.

Reward Function

Kluczowy element projektowania - reward shaping, dense vs sparse rewards. Złe nagrody prowadzą do reward hacking (agent exploituje system).

Imitation Learning (BC/GAIL)

Trenowanie przez demonstrację - nagrywanie ludzkich rozgrywek jako dane treningowe. GAIL (Generative Adversarial Imitation Learning).

Inference na Urządzeniu

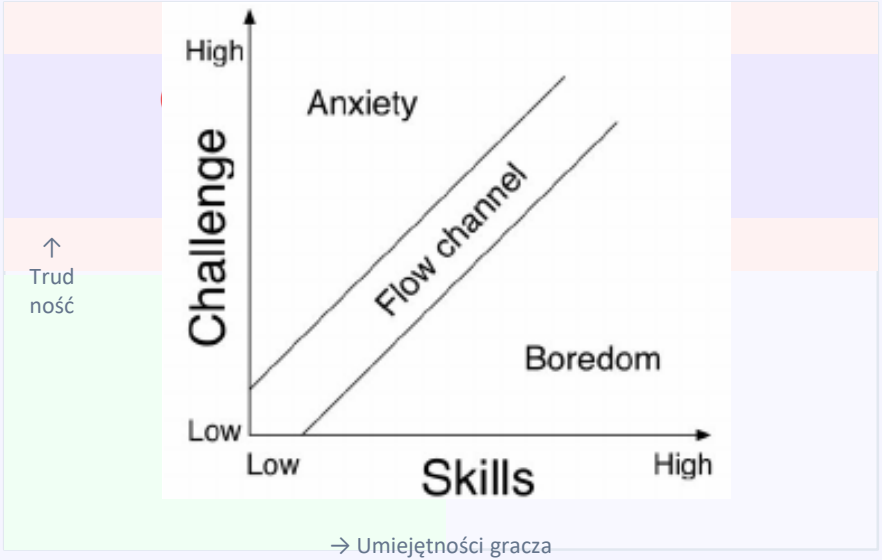
Wytrenowany model (.onnx) eksportowany do Unity → Barracuda runtime. CPU/GPU inference na mobile - realtime w 60 FPS.

Zastosowania ML w grach mobilnych: adaptacja trudności (DDA), zachowania NPC, generowanie muzyki, anti-cheat detection, matchmaking skill-based

Projektowanie gameplayowe, balansowanie i DDA

Dynamic Difficulty Adjustment (DDA) dostosowuje wyzwanie do umiejętności gracza - utrzymuje flow state i redukuje frustrację/nudę

Model Flow (Csikszentmihalyi) w projektowaniu gier



Strategie DDA

Rubber Band AI

Przeciwnicy dostosowują prędkość/agresję do wyniku gracza - Mario Kart, racing games. Kontrowersyjna mechanika.

Adaptive Spawn Rate

Liczba i typ wrogów zależny od win/loss ratio gracza w ostatnich N runach. Spawn budgets.

Statistical DDA

Monitorowanie metryk: śmierci, czasu levelów, zasobów - automatyczna korekta parametrów.

Stealth Assistance

Ukryta pomoc - crash prevention, ostatnie sekundy gracza są łatwiejsze (np. auto-aim buff).

Kluczowe metryki: śmierci/minutę · retry rate · level completion rate · sessjon length · rage quit events

Input System: obsługa dotyku i gestów na mobile

Mobile input to dotyk, gesty, akcelerometr i żyroskop - zupełnie inny paradygmat niż klawiatura i mysz

Unity Input Systems

Legacy Input Manager
Input.GetAxis(), Input.GetButtonDown() - prosty, ale nieelastyczny. Brak remappingu i multi-platform abstrakcji.

New Input System (2019+)
Action Maps, Input Actions, Binding - pełna abstrakcja platformy. PlayerInput component, Callbacks i UnityEvent. Rekomendowane.

Input.touches (Legacy Touch)
Touch[] - fazy: Began, Moved, Stationary, Ended, Canceled. Multi-touch, deltaPosition, pressure.

Rozpoznawanie gestów

```C#
// Pinch to zoom (multi-touch)
if (Input.touchCount == 2) {
    Touch t0 = Input.GetTouch(0);
    Touch t1 = Input.GetTouch(1);
    float prevDist = Vector2.Distance(
        t0.position - t0.deltaPosition,
        t1.position - t1.deltaPosition);
    float currDist = Vector2.Distance(
        t0.position, t1.position);
    float delta = currDist - prevDist;
    Camera.main.orthographicSize -= delta * zoomSpeed;
}
```

Żyroskop i Akcelerometr

```C#
// Żyroskop - VR, AR, tilt controls
Input.gyro.enabled = true;
Quaternion gyroRot = Input.gyro.attitude;
// Konwersja z Right-handed do Left-handed
Quaternion unity = new Quaternion(
    gyroRot.x, gyroRot.y,
    -gyroRot.z, -gyroRot.w);
transform.localRotation = unity;

// Akcelerometr - tilt steering
Vector3 accel = Input.acceleration;
float tilt = Mathf.Clamp(accel.x, -1f, 1f);
rb.AddForce(Vector3.right * tilt * steerForce);
```

Wirtualny joystick i HUD mobile

Float Joystick (dynamic)
Joystick pojawia się w miejscu dotyku - bardziej ergonomiczny niż statyczny

Dead Zone
Ignoruj ruch joysticka < threshold (0.1) - eliminacja szumu i drgań palca

Haptic Feedback
Handheld.Vibrate() / iOS CoreHaptics - feedback przy strzelaniu, kolizji, zbieraniu

Safe Area Layout
RectTransform safe area dla notch (iPhone X+) - UI nie przykrywa elementów systemu

UI/UX w grach mobilnych: projektowanie interfejsu

UI gry mobilnej musi być czytelne na ekranie 5–7 cali, obsługiwane jednym kciukiem i nie rozpraszać od rozgrywki

Systemy UI Unity

Canvas (UGUI) *Legacy*

RenderMode: Screen Space Overlay / Camera / World Space. Layout Groups (Vertical, Horizontal, Grid). Rector Transform. Powszechnie używany, ale kosztowny przy wielu elementach.

UI Toolkit (UIElements) *Nowe*

UXML + USS (jak HTML+CSS). Responsive design, data binding, UI Builder. Rekomendowane dla złożonych UI i cross-platform.

World Space UI *(Diegetic)*

UI osadzony w świecie gry. Np. pasek życia nad głową, interaktywne ekrany w świecie. Immersja bez HUD nakładki.

Zasady UX dla gier mobilnych

Thumb-friendly zones

Dolna część ekranu - kluczowe przyciski akcji.
Górny HUD - pasywna informacja.
Unikaj centrum - zasłania akcję.

Hierarchia ekranów i nawigacja

Splash Screen → Main Menu

max 3 sekundy, auto-skip po załadowaniu, logo animacja

Main Menu → Level Select

wyraźny CTA, progress widoczny na mapie poziomów

Gameplay HUD

minimalistyczny - HP, score, special. Ukryj zbędne informacje

Pause Menu (ESC/Back)

nie wyłącza muzyki, wyraźny Resume, Settings dostępne

Game Over / Victory

wynik, nowe rekordy podkreślone, retry jako domyślna akcja

In-App Store UI

FOMO triggers, bundles, limited time offers - etyka monetyzacji!

Audio w grach mobilnych: dźwięk i muzyka

Audio to 50% doświadczenia gry. Często niedoceniane, ale kluczowe dla game feel, immersji i informacji zwrotnej

Architektura audio w Unity

AudioSource + AudioClip

Podstawowe API: PlayOneShot() dla SFX, Loop dla muzyki. AudioSource.Play() bez allokacji GC.

Audio Mixer

Grupy Master/Music/SFX z efektami (Reverb, Echo, Compressor). Ducking: ściszanie muzyki podczas dialogów.

FMOD Studio (Middleware)

Profesjonalne narzędzie audio, adaptacyjna muzyka, interactive music z transitions, parametry wywołujące efekty. Wwise jako alternatywa.

Spatial Audio (3D Sound)

AudioSource.spatialBlend = 1 dla dźwięków 3D. Rolloff curves, logarytmiczne dla realistycznych spadków głośności. Audio Listener na kamerze.

Optymalizacja audio mobile

- Format: Vorbis/OGG dla muzyki, ADPCM dla krótkich SFX
- Load Type: Compressed In Memory dla muzyki BGM
- Mono audio dla dźwięków 3D - stereo tylko dla muzyki
- Max Voices: 32 jednoczesnych kanałów na mobile
- AudioSettings.outputSampleRate: 44100 Hz standard

AudioManager: Object Pool dla SFX

C#

```
public class AudioManager : MonoBehaviour {
    [SerializeField] AudioSource[] sfxSources;
    int _idx = 0;

    public void PlaySFX(AudioClip clip,
        float volume = 1f, float pitch = 1f) {
        var src = sfxSources[_idx % sfxSources.Length];
        src.clip = clip;
        src.volume = volume;
        src.pitch = pitch + Random.Range(-0.05f, 0.05f);
        src.Play();
        _idx++;
    }

    // Muzyka z cross-fade
    public IEnumerator CrossfadeMusic(
        AudioClip newTrack, float duration = 1f) {
        float t = 0;
        while (t < duration) {
            musicSource.volume = Mathf.Lerp(1, 0, t/duration);
            t += Time.deltaTime;
            yield return null;
        }
        musicSource.clip = newTrack;
        musicSource.Play();
        // ... fade in
    }
}
```

Optymalizacja CPU: profilowanie i wydajność kodu

Cel dla gier mobilnych: 60 FPS przy max 16.6 ms na klatkę. Każda milisekunda ma znaczenie na słabszym SoC

Narzędzia profilowania

Unity Profiler

Wbudowany - CPU, GPU, Memory, Audio, Physics frame-by-frame. Deep Profiling (wolniej, ale pełne callstacks). Profile Build na urządzeniu.

Frame Debugger

Podgląd każdego draw call - batching, SetPass calls, materiały. Kluczowy do identyfikacji zbędnych renderów.

Memory Profiler

Heap snapshot - które obiekty zajmują pamięć, referencje do GC roots. Managed vs Native memory.

Android GPU Profiler / Xcode Instruments

Natywne narzędzia platform - GPU counters, shader stalls, bandwidth. Snapdragon Profiler (Qualcomm), Mali Graphics Debugger.

Typowe wąskie gardła CPU i rozwiązania

`GetComponent<T>() w Update()`	Cache w Start() → prywatne pole. GetComponent() to kosztowny reflection-based lookup.
`GC Allocation - garbage`	Unikaj new w Update(), string concatenation, LINQ. Użyj struct zamiast class dla małych danych.
`Coroutines vs async-await`	Coroutine ma overhead GC. UniTask (Cysharp) jako zero-alloc async/await dla Unity.
`Instantiate / Destroy`	Object Pooling - recykl zamiast tworzenia nowych. Unity 2021: ObjectPool<T> wbudowany.
`Find() / FindObjectOfType()`	Cache referencji w Awake/Start lub ScriptableObject Event. Nigdy w Update().
`Physics.OverlapSphere() per frame`	Zmniejsz częstotliwość lub użyj NonAlloc variants: OverlapSphereNonAlloc(buffer).

Optymalizacja GPU i zarządzanie pamięcią

Na mobile GPU i CPU współdzielą RAM. Każdy megabajt pamięci tekstur bezpośrednio ogranicza możliwe zasoby gry

Redukcja Draw Calls (SetPass Calls):

Static Batching

Nieporuszające się obiekty z tym samym materiałem - scalane w jeden mesh przed grą. Zero overhead runtime. Mark as Static.

Dynamic Batching

Małe meshe (<300 werteksów) z tym samym materiałem - automatyczne łączenie w runtime. Ograniczone przez werteksy.

GPU Instancing

Setki identycznych obiektów (drzewa, trawy, wrogowie) - jeden draw call. Wymaga instancing shader. enable: material.enableInstancing = true

SRP Batcher

URP Batcher - grupuje materiały o tym samym shaderze, nie obiekty. Znacząca redukcja CPU overhead.

Zarządzanie pamięcią RAM:

Texture Compression

ASTC (iOS i Android Vulkan) - najlepszy compression ratio, 4×4 block. ETC2 dla Android OpenGL ES. PVRTC dla starsze iOS. Redukcja 75–87% rozmiaru.

Asset Bundle / Addressables

Dynamiczne ładowanie zasobów - tylko potrzebne assety w pamięci. Addressables System: asynchroniczne ładowanie, reference counting, hot reload.

Texture Atlasing

Łączenie wielu tekstur → jeden atlas. Redukcja draw calls i VRAM fragmentation. TexturePacker, Unity Sprite Atlas.

LOD (Level of Detail)

LOD Group - automatyczna zamiana na prostszy mesh z odległości. LOD0 (blisko) → LOD1 → LOD2 → Culled. Ogromna redukcja polygon count.

Budżety pamięci: iOS 500 MB max recommended · Android 350 MB (low-end), 750 MB (high-end) · Tekstury: max 50% RAM budget

Zarządzanie scenami, stanem gry i assetami

Architektura zasobów i scen determinuje czas ładowania, zużycie pamięci i możliwości skalowania gry

Strategie zarządzania scenami

Single-scene z Additive Loading
Jedna trwała scena (GameManager, AudioManager) + dynamiczne ładowanie/odładowywanie sub-scen przez LoadSceneAsync(scene, Additive). Brak freezeframe.

Addressable Scenes
Sceny jako Addressable Assets - ładowanie po kluczu/label, grupowanie, dependency tracking. Ideal dla dużych open-world.

Loading Screen Pattern
Async ładowanie z paskiem postępu. AsyncOperation.progress (0–0.9, 0.9–1.0 to activation). allowSceneActivation = false dla ładowania w tle.

System zapisu gry

```
[Serializable] class SaveData {
    public int level, score, coins;
    public float[] playerPos;
}
// Zapis: JSON + AES encryption
string json = JsonUtility.ToJson(data);
File.WriteAllText(savePath, EncryptAES(json));
// Wczytanie
string raw = DecryptAES(File.ReadAllText(savePath));
var d = JsonUtility.FromJson<SaveData>(raw);
```
C#

Unity Addressables workflow

```
// ładowanie assetu po kluczu (adres/label)
var handle = Addressables.LoadAssetAsync<Sprite>(
    "UI/hero_portrait");
handle.Completed += (op) => {
    if (op.Status == AsyncOperationStatus.Succeeded)
        portraitImage.sprite = op.Result;
};

// ładowanie sceny
var sceneHandle = Addressables.LoadSceneAsync(
    "Level_01", LoadSceneMode.Additive);
await sceneHandle.Task;

// WAŻNE: zwalniaj po użyciu
Addressables.Release(handle);
Addressables.UnloadSceneAsync(sceneHandle);

// Preload w tle (warm-up)
Addressables.DownloadDependenciesAsync("level_pack_1");
```
C#

Lifecycle zasobów

Load → Use → Unload (Resources.UnloadUnusedAssets)
wywołuj po scenie loading, NIE w Update

GC.Collect()
callback dla iOS/Android

Application.lowMemory

Profilowanie na Urządzeniu i Debugowanie

Profilowanie w edytorze jest niewystarczające - zawsze profiluj na docelowym urządzeniu, szczególnie low-end Androidach

Workflow profilowania na urządzeniu:

1. Build Development Build + Profiler

Player Settings → Development Build + Autoconnect Profiler. IL2CPP release build dla finalnych benchmarków.

2. Uruchom na urządzeniu przez USB

Android: ADB over USB lub Wi-Fi. iOS: Xcode. Unity Remote nie jest wystarczające dla pomiarów wydajności.

3. Zidentyfikuj hotspoty

Deep Profiler: hierarchia wywołań. Sortuj po Self Time - metody pochłaniające czas bez dzieci.

4. Wdróż fix i zmierz ponownie

Nigdy nie optymalizuj bez mierzenia przed i po. Zmiana może nie mieć efektu lub go pogorszyć.

Custom Profiler Markers:

```C#
using Unity.Profiling;
// Niealokujące markery do własnego kodu
static readonly ProfilerMarker k_GenChunk =
    new ProfilerMarker("WorldGen.GenerateChunk");

void GenerateChunk(Vector2Int coords) {
    using (k_GenChunk.Auto()) {
        // ten kod będzie widoczny w Profilerze
        GenerateTerrain(coords);
        PlaceObjects(coords);
    }
}
```

Typowe problemy na mobile:

Thermal Throttling
Przegrzanie → CPU/GPU obniżają taktowanie. Rozwiązanie:

Memory Warning → Crash
iOS: didReceiveMemoryWarning - zwalniaj cache, tekstury. Brak

Shader Compilation Stutter
Pierwsze użycie shadera → hitches.

GFX Jobs Single-threaded
Domyślnie Unity renderuje single-thread. Włącz Graphics Jobs w Player Settings (eksperymentalne, ale +20% mobile)

Multiplayer w Grach Mobilnych - Architektura Sieciowa

Sieciowość gier to jeden z najtrudniejszych problemów inżynieryjnych - latencja, desync, cheating i skalowalność

Topologie sieciowe:

Client-Server (Authoritative)

Serwer jako jedyna prawdziwa symulacja - klient wysyła wejście, serwer odsyła stan. Odporność na cheating. Latencja widoczna. Standard AAA.

Peer-to-Peer (P2P)

Gracze komunikują się bezpośrednio - brak kosztów serwera, ale podatność na cheat, problemy NAT traversal (STUN/TURN). Używane w racing, fighting.

Listen Server

Jeden z klientów pełni rolę serwera - prostsze wdrożenie, but host advantage. Popularne w indie games.

Relay Server (Steam/Epic)

Pośredni serwer relay eliminuje problemy NAT bez kosztów dedykowanego. Epic Online Services, Steam Networking, Unity Relay.

Protokoły i SDKi multiplayer:

Netcode for GameObjects (Unity)

Oficjalny SDK Unity - snapshot synchronisation, RPCs, NetworkVariable. Dobry start dla projektów Unity.

Mirror (Open-Source)

Fork UNET - stabilny, duże community, battle-tested. High-Level API z wieloma transportami (TCP, KCP, WebTransport).

Photon Fusion / PUN

BaaS dla mobile - managed servers, global regions. Fusion: deterministic rollback netcode. PUN2: event-based, prosty start.

Fish-Net

Nowoczesna alternatywa Mirror - performance-first design, Prediction, LOD, wsparcie dla ECS.

Kompensacja Latencji i Synchronizacja Stanu

Gracze mobilni mają typowo 50–150 ms latencji - gra musi być grywalna pomimo opóźnień transmisji

Dead Reckoning

Przewidywanie przyszłej pozycji na podstawie ostatniego stanu i prędkości - interpolacja i ekstrapolacja. Smooth movement mimo pakiet drops.

Client-Side Prediction

Klient natychmiast aplikuje własne wejście lokalnie - serwer potwierdza lub koryguje (reconciliation). Eliminuje odczucie lagowania sterowania.

Server Reconciliation

Po potwierdzeniu od serwera: klient porównuje stan, cofa symulację do potwierdzonego stanu, reaplikuje niepotwierdzony input. Roll-forward.

Lag Compensation (Hitscan)

Serwer "cofa czas" - przy strzale sprawdza pozycje przeciwników w chwili T-latency, nie T-now. Valobraнт, Counter-Strike.

Interpolation Buffer

Bufor stanu z opóźnieniem ~100 ms - płynne wyświetlanie stanu innych graczy przez interpolację między punktami. Masło smooth.

Lockstep / Deterministic

Wszyscy gracze wykonują identyczne obliczenia → identyczny stan. Zero transferu stanu, tylko wejście. Wymagana pełna determinizm (Fixed timestep, seed).

Kluczowe metryki: RTT (Round Trip Time) < 150 ms dla real-time gier · Packet loss < 5% · Jitter < 30 ms · Tick rate: 20–128 Hz (zależy od gatunku)

Backend dla gier mobilnych - Usługi i Infrastruktura

Game Backend (BaaS - Backend as a Service) dostarcza infrastrukturę serwerową bez konieczności budowania jej od zera

Platformy BaaS dla gier:

Unity Gaming Services

- Authentication (anon, Steam, Apple)
- CloudSave - cross-device progression
- Leaderboards, Economy
- Matchmaking, Relay, Lobby
- Cloud Code (serverless JS)

Firebase (Google)

- Realtime Database / Firestore
- Authentication (Email, Google, Apple)
- Remote Config - feature flags
- Analytics + Crashlytics
- Cloud Functions (Node.js)

PlayFab (Microsoft Azure)

- Entity player data model
- Virtual economy (currencies)
- Server hosting & matchmaking
- LiveOps: tournaments, events
- Economy v2, Leaderboards v2

Nakama (Open-Source)

- Self-hosted / Heroic Cloud
- Social: friends, groups, chat
- Multiplayer (authoritative)
- Lua/JS/Go server runtime
- Full real-time multiplayer

Kluczowe funkcjonalności backend gier:

Player Authentication
Anonimowe → linked (Apple/Google) → pełne konto. Device ID jako seed.

Player Progression
XP, poziomy, odblokowania. Weryfikacja serwer-side (anti-cheat).

Anti-cheat
Signature validation, hash checksum save file, server authority.

Analytics Events
Funnel analysis, monetization metrics, LTV calculation.

LiveOps Events
Rotujące eventy, sezonowe treści, time-limited offers.

CDN Delivery
Delta patching, hot content delivery, A/B testing.

Monetyzacja Gier Mobilnych - Modele Biznesowe

Wybór modelu monetyzacji to jedna z najważniejszych decyzji projektowych - wpływa na design, retention i etykę gry

Premium (Paid App)

$0.99–$9.99

✓ Brak reklam, uczciwe, prosty model

✗ Mała baza graczy (płatna bariera)

ARPU: $2–5 jednorazowo

Przykład: Minecraft, Monument Valley

Free-to-Play + IAP

Darmowa + mikrotransakcje

✓ Ogromna baza użytkowników

✗ Ryzyko pay-to-win, koszty UA

ARPU: $0.01–3 avg, wieloryby 20%

Przykład: Clash of Clans, Pokémon GO

Subscrypcja

$2.99–9.99/mies.

✓ Przewidywalny przychód, stabilność

✗ Wysokie oczekiwania aktualizacji

ARPU: $36–120 LTV

Przykład: Apple Arcade, Royal Match+

Reklamy (Ad-based)

Darmowa + CPM/CPC

✓ Zero bariery wejścia, masowy reach

✗ Złe UX, niskie CPM w Polsce

ARPU: $0.5–2 ARPDAU

Przykład: Candy Crush (hybryd), Wordle

Hybrydowy model (F2P + Premium Battle Pass + Cosmetics) = dominujący trend 2022–2025. Kluczowe: brak pay-to-win dla retencji społeczności.

In-App Purchases: Implementacja i Projektowanie Sklepu

IAP muszą być zaprojektowane z myślą o wartości dla gracza - nie tylko maksymalizacji przychodu, ale i pozytywnym doświadczeniu

Typy IAP:

Consumable (Zużywalne)

Waluta gry, boosters, życia - mogą być kupowane wielokrotnie. Baza systemu ekonomii gry.

Non-Consumable (Trwałe)

Odblokowanie poziomów, usunięcie reklam, dodatkowe postacie - jednorazowy zakup per konto.

Subscription (Subskrypcja)

VIP membership, battle pass, premium currency tygodniowo - automatyczne odnowienie, wymaga specjalnej obsługi anulowania.

Implementacja IAP (Unity IAP):

```C#
// Unity IAP: IStoreListener.ProcessPurchase
// Validate receipt server-side! (nie client-side)
```

Walidacja receiptu - kluczowe!

```Java/Kotlin
// NIE ufaj klientowi - waliduj server-side
// 1. Klient: kup → receipt (base64)
// 2. Klient → własny serwer: POST /validate
// 3. Serwer → Apple/Google API:
//    POST https://buy.itunes.apple.com/verifyReceipt
// 4. Serwer waliduje i dopiero WTEDY przyznaje item

// Google Play Billing (Android IAP)
BillingClient billing = BillingClient.newBuilder(ctx)
    .setListener((result, purchases) -> {
        if (result.getResponseCode() == OK && purchases != null)
            for (var p : purchases)
                verifyWithServer(p.getPurchaseToken());
    })
    .enablePendingPurchases()
    .build();
// Zawsze consumuj DOPIERO po potwierdzeniu z serwera
```

Zasady projektowania sklepu

- **Anchoring cen**
- **Limited-time offers**
- **Bundle value framing**
- **First purchase discount**

50% zniżka na pierwszy zakup - onboarding do płatności

Dystrybucja, App Stores i Live Operations

Sukces gry mobilnej to nie tylko launch - LiveOps i ciągłe aktualizacje decydują o długoterminowej retencji

Wymagania App Store / Google Play:

Privacy Manifest (iOS 17+)

Apple wymaga deklaracji używanych API (tracking domains, NSPrivacyAccessedAPITypes). Brak manifest → odrzucenie aplikacji.

Data Safety (Android 2022+)

Google Play: deklaracja zbieranych danych, cel, udostępnianie third-party. RODO compliance obowiązkowe w EU.

IDFA Consent (ATT Framework)

iOS 14.5+: App Tracking Transparency - pytanie o zgodę przed śledzeniem. Opt-in rate ~25–40%. Wpływ na mobile advertising.

Content Rating (PEGI/ESRB)

Questionnaire przy submission. Wynik wpływa na widoczność w sklepie i dostępność dla młodszych użytkowników.

Live Operations - utrzymanie zaangażowania:

Sezonowe eventy

Halloween, Boże Narodzenie, rocznice - limitowane skórki, questy, waluta sezonowa. +40% DAU w evencie.

Battle Pass

Sezonowe drzewko nagród płatne + bezpłatne - Fortnite model. Zwiększa sesja length i LTV.

Turnieje i Rankingi

Weekly leaderboards, endgame PvP. Poczucie progressu dla hardcorowych graczy, prestige.

Push Notifications

Firebase FCM / APNs - przypomnienia o energii, eventy, powrót gracza. Personalizacja zwiększa CTR.

Remote Config / A/B Testing

Firebase Remote Config - zmiana parametrów bez update. A/B test cen IAP, trudności, UI layoutu.

D1/D7/D30 Retention: Day-1 >40%, Day-7 >20%, Day-30 >10% - benchmark top quartile mobile games

Case Study 1 - Indie Puzzle Game: od prototypu do 1M pobrań

Analiza procesu tworzenia niezależnej gry puzzle na iOS/Android - 1 developer, 8 miesięcy, budget $0 → 1M+ pobrań

1.2 mln	4.7★	D30: 12%	$4,200
Pobrania pierwsze 3 miesiące	Rating App Store	Day-30 Retention	Przychód pierwszy miesiąc

Stos technologiczny

Unity 2022 LTS + URP	Stabilna wersja LTS dla komercyjnych projektów
C# + Zenject (DI)	Dependency Injection od dnia 1, czysta architektura, testowalna
DOTween Pro	Wszystkie animacje UI i efekty, zero Animatora dla UI
Unity Addressables	Rozdzielenie contentu od kodu, szybkie patche bez pełnego update
Firebase Analytics + Crashlytics	Metryki i raporty crashy, identyfikacja problemów na device
Unity IAP + Admob (hybryd)	Rewarded ads + optional IAP, nie-inwazyjny model monetyzacji

Kluczowe wnioski

Soft launch w Kanadzie/Australii
Test metrics (CPI, Retention) przed globalnym launche'm. Iteracja bez utraty App Store rating.

ASO (App Store Optimization)
Ikona i screenshots to 60% konwersji. Iteruj ikonę A/B testem. Keyword research przed launch.

Core loop w 3 zdania
Jeśli nie możesz opisać core loop w 3 zdaniach, mechanika jest zbyt złożona dla casual mobile.

Onboarding bez tekstu
Pierwsze 60 sekund bez ekranów z tekstem. Nauka przez doing, UI overlay na żywym gameplayu.

Performance budgety od dnia 1
iPhone 8 jako minimum target. Codzienne profiling buildy na low-end device zapobiegły performance debt.

Trendy i podsumowanie

Generatywna AI w grach

LLM dla dynamicznych dialogów NPC, AI Game Masters, infinite quest generation - Inworld AI, Convai

Apple Vision Pro / MR

Mixed Reality gaming - overlay gier na rzeczywistości, nowe paradygmaty interakcji przestrzennej

Cloud Gaming Mobile

GeForce Now, Xbox Cloud Gaming - gry AAA na słabym sprzęcie, 5G jako enabler

AI-Assisted Development

GitHub Copilot, Unity Muse, Bolt.new - AI przyspiesza prototypowanie 3–10×, obniża barierę wejścia

Blockchain / NFT Games (Web3)

Play-to-Earn - kontrowersyjny trend. Regulacje i krach 2022 spowolniły. Concept własności cyfrowej pozostaje interesujący.

Cross-Platform Play & Progression

PC ↔ Mobile ↔ Console. Jeden account, jeden progress. Wymagane na rynku premium 2025+

Kluczowe wnioski wykładu

- Wybór silnika (Unity/Unreal/Godot) determinuje architekturę, narzędzia i czas produkcji - decyzja nieodwracalna
- Optymalizacja mobile to nie etap końcowy, lecz ciągła praktyka od dnia 1 - performance budgety na low-end device
- Game Feel > Feature Count - gra z dobrą mechaniką core loop i polskim feel pokona grę z setkami ficzerów
- Data-driven development: metryki (D1/D30 retention, ARPU, session length) zastępują intuicję na etapie skalowania

Pytania, dyskusja i koniec