Graviton Phaze — Game Design Document

Jake Zivontsis

1) Concept

Graviton Phaze is a 2D top-down, touch-friendly physics game with a single loop:

(1) Plan by launching unmanned probes (projectiles) that latch/capture and deploy local field nodes or trigger switches.

(2) Fly a skiff using thrust and rotation through the field you engineered. Fuel is limited and a timer tracks your time. Gravity wells can swallow the player; counts as a failed attempt.

(3) Reach the goal to complete the level in the fewest attempts and shortest time.

The levels take place in space with gravity wells and field nodes. The learning heart: summing forces, conservative fields, momentum, drag, and numerical integration the player can feel.

2) Platforms & Controls (Mobile — Unity Input System)

Platform: Android (mobile-only). Resolution: 20:9 and 16:9 phones.

2.1 Required Unity Assets / Classes

- InputActionAsset: GravitonInput.inputactions (Generate C# Class on).
- Action Maps: Plan (probe), Fly (skiff).
- Control Scheme: TouchGyro (Touchscreen + AttitudeSensor; Accelerometer fallback).
- PlayerInput component: switches action maps.
- InputSystemUIInputModule on EventSystem: UI consumes touches; gameplay ignores touches over UI.

2.2 Action Maps & Bindings

A) Plan (Probe Phase) — drag to aim, release to fire

- AimPoint (Vector2) — Touchscreen/primaryTouch/position.
- AimHold (Button) — Touchscreen/primaryTouch/press with Press (default). Use events: started → begin aim; performed (held) → update aim; canceled (on finger release) → fire.
- SwapProbe (Button) — UI button (icon).

B) Fly (Skiff Phase) — tilt to steer; touch anywhere to thrust (except UI buttons)

- Thrust (Button) — Touchscreen/primaryTouch/press with Press&Hold; gameplay is disabled under UI via InputSystemUIInputModule.
- Burst (Button) — Touchscreen/primaryTouch/tap with Tap (TapCount=2).
- Sustained (Button) — Touchscreen/primaryTouch/press with Hold (MinDuration≈0.35s).
- Turn (Vector2) — AttitudeSensor/attitude (yaw from device tilt); Accelerometer fallback.
- Calibrate (Button) — UI button to set neutral orientation.
- Pause (Button) — UI button (optional three-finger tap gesture).

2.3 PlayerInput Wiring & Map Switching

```
// Pseudocode

OnPhaseChange(Plan): PlayerInput.SwitchCurrentActionMap("Plan")

OnPhaseChange(Fly):  PlayerInput.SwitchCurrentActionMap("Fly")
```

2.4 Runtime Pseudocode

Plan — Probe Aiming / Firing

```
AimHold.started    → BeginAim(AimPoint.value)         // record origin

AimHold.performed → UpdateAim(AimPoint.value)        // while finger held

AimHold.canceled  → FireProbe()                      // release fires


BeginAim(p0):      Aim.origin = Cannon.pos ; Aim.active = true

UpdateAim(p):      Aim.v0 = mapDragToVelocity(Aim.origin, p) ;
Predictor.DrawProbe(Aim.origin, Aim.v0)

FireProbe():       if Aim.active: SpawnProbe(Aim.origin, Aim.v0,
SelectedProbeType) ; Aim.active = false
```

Fly — Skiff (Gyro + Touch Thrust)

```
Turn.read → heading = YawFromDevice(attitude, calibration)  // [-1..1]

Thrust.performed → thrustOn = true

Thrust.canceled  → thrustOn = false

Burst.performed  → ApplyBurstImpulse()

Sustained.performed → EnterSustainedMode()  // (optional modifier)
```

```
Ship.Tick(dt):

    if thrustOn and fuel>0: v += headingDir(heading) * thrustAccel * dt ; fuel
-= rate * dt

    v += AccelAt(x, v, t) * dt

    x += v * dt
```

2.5 Notes

- Touch anywhere for thrust: bindings use primaryTouch; UI module consumes UI touches so gameplay doesn't trigger under buttons.
- Use processors (Scale, Deadzone) on Turn to tune sensitivity.
- Provide a Calibrate button to store neutral attitude; apply inverse each frame.

3) Core Loop

Start Level → Plan Phase: Fire N Probes → (each probe resolves to rest → deploys node / toggles switch)

→ Fly Phase: Pilot Skiff → (fuel constraints; collisions) → Goal Reached?

Yes → Score/Stars → Next ; No → Retry Fly (keep anchors) / Reset Plan (remove anchors)

4) World & Object Model

Coordinate system: 2D top-down in Unity units (u). Fixed physics tick at 60 Hz; predictor uses identical math (optional sub-step dt/2).

Key entities

• FieldSource: generic force emitter (well/repulsor/stabilizer/jetstream/vortex/nebula).

• Probe: projectile with initial impulse only; becomes Anchor or triggers Switch after rest.

• Anchor: persistent FieldSource deployed by a probe.

• Auto-Dock Node: floating receptor: large trigger; entering it snaps probe to center, zeroes v, and deploys Anchor.

• Switch Ring: fly-through trigger; toggles doors/fields on timer.

• Door/Wall: solid obstacles to traverse around.

• Skiff: player craft; thrust + rotate; fuel limited.

• Goal Pad: level exit.

5) Physics Model

5.1 Net Acceleration

All forces superpose. For position x, velocity v:

$$\vec{a}(\vec{x},\vec{v},t) = \sum_i \vec{a}_i(\vec{x},\vec{v},t)$$

Notation & Variables:

• x = (x, y): world position in 2D

• v = (vx, vy): velocity

• t: time

• c: center of a field source

• r = x − c: displacement vector from field center

• S, U_0, Γ, k, c: tunable strengths/coefficients for each field

• ε: small soft-core to avoid singularities near centers

• R: characteristic radius for Gaussian fields

5.2 Force Modules (active and player-placed)

A) Inverse-Square Gravity Well (attractor)

Equation: $\vec{a}_{well}(\vec{x}) = S\,\frac{\vec{c}-\vec{x}}{(\|\vec{x}-\vec{c}\|^2+\varepsilon)^{3/2}}$

Summary: produces an attraction toward center c that grows stronger when closer; ε prevents infinite acceleration at r=0.

Parameters: S (strength), ε (soft-core).

B) Inverse-Square Repulsor

Equation: same as A but with negative S.

Summary: pushes away from center; useful as a “soft wall” near hazards.

Parameters: S (negative), ε (soft-core).

C) Gaussian Stabilizer (localized attractor)

Potential: $\Phi(r) = -U_0 * e^{-r^2 / R^2}$

Acceleration: $\vec{a} = \left(\frac{2U_0}{R^2}\right) * e^{-r^2 / R^2} * (\vec{c} - \vec{x})$ with optional accel clamp.

Summary: a local “centering” well that fades quickly outside ~2R; strong near the gap, negligible elsewhere.

Parameters: U_0 (depth), R (radius), $\vec{a}_{Max}$ (clamp).

D) Uniform Field (jetstream)

Equation: $\vec{a} = \vec{E}$ (constant vector inside a region)

Summary: produces steady drift in a direction; combine with anchors to shape lanes.

Parameters: E (vector).

E) Quadratic Drag (nebula/gel)

Equation: $\vec{a} = -k \, \|\vec{v}\| \, \vec{v}$

Summary: dissipative force that slows moving bodies proportional to speed^2 (in magnitude); great for capture nets and comfort.

Parameters: k (drag coefficient).

F) Vortex / Tangential Field

Equation: $\vec{a} = \Gamma \frac{\hat{z} \times (\vec{x} - \vec{c})}{\|\vec{x}-\vec{c}\|^2 + \varepsilon}$

Summary: sideways acceleration around a point; curves trajectories without pulling to center.

Parameters: Γ (circulation), ε (soft-core).

5.3 Collisions & Rest States

Skiff collisions

- Walls/Doors: reflect velocity about surface normal with restitution $e \approx 0.2$ (light bounce).
- Well cores (fail): if distance to any well center < r_fail (core radius) → Fail and present Retry/Reset options.
- Out of bounds (fail): if position leaves the playable bounds → Fail.
- Receptors: receptors are non-solid triggers; skiff ignores them (only their fields affect motion).

Probe collisions

- Receptors (Auto-Dock): if probe center enters receptor snap radius R_snap → set v=0, snap x to receptor center, Deploy Anchor immediately.
- Wells (consume): if distance to well center < r_consume (core radius) → Destroy probe (no deploy).
- Walls/Doors: same response as skiff (light bounce).
- Out of bounds: Destroy probe.

Notes

- Use per-well radii r_fail (skiff) and r_consume (probe) to tune forgiveness.
- Receptors are triggers only; all gameplay effect comes from active FieldSources after a probe docks.
- If a probe reaches a velocity of zero without being destroyed or activated, it is destroyed.

5.4 Integrator & Stability

Semi-Implicit Euler (symplectic):

1. v = v + a*dt
2. x = x + v*dt

Stability tactics:

- fixed dt=1/60
- clamp |a| for anchors
- soft-core ε
- (optional) sub-step near strong fields

5.5 Trajectory Predictor (Stretch)

Runs off-sim with the same AccelAt() and dt; renders a line. Includes all active sources and timers so previews are honest.

6) Game Flow & UI

Phases: Plan / shoot probes → Fly → Score → Retry / next level.

HUD: fuel bar during Fly phase, probe count during Plan phase, timer, pause: star goals, retry, quit.

Physics Overlay (toggle): show v arrow and Σa arrow on active body (stretch goal).

Menus: level select, settings (volume, gyro sensitivity), success / fail screen.

7) Levels (first arc)

L1 — First Light

• Two fixed wells create a pinch; dock at center.

• Probe docks → Gaussian stabilizer turns on.

• Fly skiff through stabilizer to goal.

• Stars: 1 probe, ≥50% fuel, <20s.

L2 — Bank Shot

• Place repulsor on well's inner flank; second probe rides banked curve to a Switch Ring (opens door 12s).

• Fly through door while repulsor keeps skiff off the well.

L3 — Jetstream Alley

• Global downward uniform field; use Jetstream probe, anchor to blend/equalize drift.

L4 — Nebula Gate (stretch)

• Large nebula patch and limited fuel.

• Fire a vortex probe through the lower friction left side of the patch into a receptor.

• Shoot a Jetstream probe toward the vortex to arc a shot around the higher friction side of the nebula patch into a receptor, activating a Jetstream through the nebula.

• Fly through the Jetstream.

8) Scoring & Progression

• Stars (0–3): probes used, fuel remaining, completion time.

• Fail: OOB, fuel zero, hazard core contact.

• Assist/overlay toggles; default off for clarity.

9) Art & Asset List (minimal)

Sprites/VFX: Skiff, Probe, Well Core, Stabilizer/Anchor Node, receptor, Switch Ring, Door/Barrier, Uniform Field Strip, Nebula/Drag Zone, Goal Pad; UI icons.

10) Audio Direction & Assets

SFX: probe launch, docking latch, anchor pulse, switch ring pass, door open/close, thrust loop, collision thud, goal stinger, UI taps.

Music: 2× short loops (planning, flight). AlkaKrab's Sci-Fi music pack.

11) Technical Architecture

Core Scripts: FieldSource2D; FieldManager (SumAccelAt); ProbeController (aim/launch/integrate/rest→Deploy); AnchorDeployed (spawns FieldSource2D); BuoyCapture; LatchPad; SwitchRing→Door; ShipController (gyro rotation, touch thrust, fuel, integrate, collisions); TrajectoryPredictor; LevelDirector; HUDController.

Data: ScriptableObjects for FieldParams and LevelConfig.

12) Pseudocode (physics & flow)

12.1 Core Physics

```
AccelAt(x, v, t):           // sum active field accelerations

    a = (0,0)

    for F in ActiveFields: a += F.Accel(x, v, t)

    return a


PhysicsStep(body, dt):      // semi-implicit Euler

    body.v += AccelAt(body.x, body.v, time) * dt

    body.x += body.v * dt
```

12.2 Field Modules

```
Well.Accel(x):              // inverse-square (softened)

    r = c - x ; d2 = |r|^2 + eps ; return S * r / d2^(3/2)


Gauss.Accel(x):             // localized stabilizer

    r = c - x ; g = exp(-|r|^2 / R^2) ; a = (2U0/R^2) * g * r ; return clamp(a,
aMax)


Uniform.Accel(x):           // region-gated constant field

    return region.contains(x) ? E : (0,0)


Drag.Accel(v):              // quadratic drag (nebula)

    return -k * |v| * v


Vortex.Accel(x):            // tangential bend (use lightly)

    r = x - c ; d2 = |r|^2 + eps ; a = Gamma * perp(r) / d2 ; return clamp(a,
aMax)
```

12.3 Probes — Aiming, Firing, Lifecycle

```
BeginAim(touchStart): Aim.active = true ; Aim.origin = Cannon.x
```

```
UpdateAim(touchPos):        Aim.v0 = mapDragToVelocity(Aim.origin, touchPos) ;
Predictor.DrawProbe(Aim.origin, Aim.v0)

FireProbe():

    if not Aim.active: return

    p = SpawnProbe(Aim.origin, Aim.v0, SelectedProbeType) ; Aim.active = false


Probe.Tick(dt):

    PhysicsStep(self, dt)

    if enters(AutoDockNode): snapToCenter(); DeployAnchor(node, self.type);
Destroy(self)

    if near(WellCore): Destroy(self)

    if OutOfBounds(self.x): Destroy(self)

    if not docked and |v| ≤ v_eps for ≥ t_stationary: Destroy(self)
```

12.4 Deploying Anchors / Switches

```
DeployAnchor(node, probeType):

    switch probeType:

        case Stabilizer: spawn GaussField at node.center with node.params

        case Repulsor:   spawn WellField  (S < 0) at node.center with params

        case Jetstream:  spawn UniformField (+direction) in node.region

        case Vortex:     spawn VortexField at node.center with params

    if node.opensDoor: Doors[node.doorId].OpenFor(node.doorSeconds)
```

12.5 Trajectory Prediction (Probes)

```
Predictor.DrawProbe(x0, v0):

    x = x0 ; v = v0 ; t = 0

    repeat N steps:

        v += AccelAt(x, v, t) * dt_pred

        x += v * dt_pred

        drawDot(x)

        if enters(AutoDockNode) or hitsWall(x) or OutOfBounds(x): break
```

12.6 Ship Tick (Gyro + Touch Thrust)

```
Ship.Tick(dt):

    heading = GyroHeading()                          // unit vector

    if TouchPressed and fuel>0: v += heading * thrustAccel * dt ; fuel -= rate
* dt

    PhysicsStep(self, dt)

    if hits(Wall): v = Reflect(v, normal) * e_wall   // or stop

    if near(WellCore): Fail("Well core")

    if OutOfBounds(x): Fail("Out of bounds")

    if at(Goal): Win()
```

12.7 Phase Flow / State Machine

```
Phase ∈ { Plan, Fly, Outcome }

StartLevel(config):

    Load(config) ; Phase = Plan


Update(Plan):

    HandleAimFire() ; UpdateProbesAndFields()

    if PlayerPressesFly or AutoAdvanceReady(): Phase = Fly


Update(Fly):

    Ship.Tick(dt) ; UpdateFields()

    if Win: Phase = Outcome ; ShowWinScreen()

    if Fail: Phase = Outcome ; ShowFailScreen()


Outcome:

    // Buttons: Next, Retry Fly (keep anchors), Reset Plan (clear anchors)
```

13) Tuning & Defaults (Level 1 reference)

Physics dt: 1/60 s.

Big wells: S=+1600, ε=25 (effective min radius ≈5u).

Stabilizer (Gaussian): R=12, U0 tuned for peak |a|≈2–3 u/s^2, aMax=3.

Ship: thrust accel 10 u/s^2, passive drag c=0.2, fuel 5.0s.

Collisions: restitution e=0.2.

Predictor: 360 steps @ 1/60 s (6 s horizon), sub-step 2× near strong fields.

14) Performance & Mobile Notes

Avoid GC in FixedUpdate; reuse arrays; pool probes/VFX. Cap line points; LOD for halos; target 60 FPS (30 FPS saver).

Stretch Goals:
Accessibility: Large touch targets; colorblind-safe palette.

15) QA Plan & Test Cases

Conservative field energy check; predictor honesty (within 1u rest); docking robustness; fail routes handled; mobile input ergonomics; gyro calibration.

16) Risks & Mitigations

Predictor mismatch → same AccelAt/dt, sub-steps, clamps.

Singularities → ε + accel caps.

Overpowered anchors / level balancing → cap ≤30% of thrust accel and prefer Gaussian. Dock instances set pre-determined params per probe type.

UX confusion → clear halos, arrows, brief tooltips.

Scope creep → one new field per level; art minimal.