

DESIGN AND DEVELOPMENT OF DON'T CRASH THE ROCKET: A UNITY-BASED INTERACTIVE SIMULATION GAME FOR TEACHING FUNDAMENTAL PHYSICS OF MOTION AND THRUST

Shang Wang ¹, Moddwyn Andaya ²

¹ Corona Del Mar High School, 2101 Eastbluff Dr, Newport Beach, CA 92660

² California State University, Sacramento, 6000 Jed Smith Dr, Sacramento, CA 95819

ABSTRACT

This paper presents don't crash the rocket, a 3D interactive simulation game developed in unity to address the difficulty many students face in understanding the fundamental physics of motion, thrust, and gravity [7]. Traditional classroom instruction often relies on formulas and static diagrams that make it hard for learners to visualize how these forces interact in real time. The problem primarily affects beginners and younger audiences who benefit from visual and experiential learning [8]. Don't crash the rocket aims to make these abstract concepts tangible through gameplay that links user input directly to realistic rocket behavior. The program's architecture consists of three core systems: a rocket control system that applies real-time physics-based thrust and rotation, a mission rule system that tracks objectives such as altitude and landing safety, and a hud interface that provides constant feedback on fuel, speed, and altitude. Built using unity's rigid body physics and scriptable objects for modular configuration, the game adapts seamlessly between desktop and mobile platforms. While its simplified physics model limits advanced precision, it effectively bridges education and entertainment by allowing players to learn aerospace fundamentals through intuitive, goal-driven interaction. The project demonstrates how interactive simulation can transform passive learning into active engagement, making physics both accessible and enjoyable.

KEYWORDS

Physics Education, Game-Based Learning, Interactive Simulation, Unity Development

1. INTRODUCTION

Many students struggle to grasp foundational physics concepts such as force, motion, energy, and drag because the material is often taught in abstract terms with heavy reliance on symbolic mathematics and static diagrams [4]. One study found that more than half of high school students lacked proficiency in mapping between vector representations and physical phenomena, highlighting deficits in visual and spatial reasoning skills [5]. Interactive game-based and simulation tools have shown promise in addressing these gaps, as research shows that learners exposed to dynamic visualizations and educational games demonstrate higher motivation and better conceptual understanding than control groups using traditional instruction alone [6]. Despite this, many mainstream educational physics tools remain too academic or disengaging for broader

audiences and do not fully leverage real-time control or mission-based interaction to make abstract motion concrete. As a result, students often disengage from physics, perceive it as irrelevant, and fail to retain critical STEM knowledge or develop interest in engineering careers. Addressing this problem is important because improving intuitive understanding of motion and control can increase STEM retention, bridge educational equity gaps, and foster practical reasoning in learners beyond rote formula memorization.

Section 2 evaluated the effectiveness of Don't Crash the Rocket as an interactive tool for supporting intuitive understanding of basic physics concepts related to motion, thrust, and controlled landing. A post-game survey was used to gather participant reflections after completing a gameplay session involving altitude control and landing objectives. Ten participants responded to a series of Likert-scale statements assessing clarity of physical feedback, perceived learning, and the relationship between player input and simulated motion. The results showed consistently positive responses, particularly in areas related to visual feedback and cause-and-effect understanding, indicating that real-time physics and HUD elements successfully reinforced learning through interaction. Slightly lower confidence scores suggest that while the simulation improved intuition, it did not fully replace formal physics instruction. Overall, the findings support the conclusion that experiential, physics-driven gameplay can enhance conceptual understanding by allowing players to directly observe and manipulate dynamic systems. Despite the limited sample size, the evaluation confirms that the project met its goal of reinforcing physics concepts through interactive simulation rather than passive explanation.

To address this problem, the proposed solution is Don't Crash the Rocket, a 3D simulation game created in Unity that allows players to pilot a physics-based rocket using thrust, rotation, and landing control. The game uses real-time physics calculations through Unity's Rigidbody system to simulate realistic motion affected by gravity, thrust force, and aerodynamic drag. The player must balance thrust and direction to reach specific goals, such as achieving a set altitude or landing safely, while managing limited fuel supply. This interactive format transforms abstract physics equations into visible and controllable outcomes. The game stands out by being modular and cross-platform, supporting both desktop and mobile users through adaptable input systems. Unlike other arcade-style rocket games, this project focuses on physical accuracy and educational value rather than arcade difficulty. The combination of a responsive HUD, dynamic mission objectives, and visual feedback provides continuous engagement while reinforcing cause-and-effect learning [12]. By connecting user actions directly to realistic physics responses, Don't Crash the Rocket makes complex aerospace concepts intuitive, enjoyable, and easy to grasp, effectively bridging the gap between entertainment and scientific learning.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Rocket Control System

An important consideration is whether the simulated rocket motion accurately reflects physical principles rather than simplified kinematic behavior. This concern is addressed through the use of Unity's Rigidbody physics engine, which calculates thrust, drag, and gravity continuously each frame rather than through scripted animation. The Rocket.cs script applies force in real Newton units, adjusting for burn rate and thrust force defined in the configuration file. As a result, acceleration, rotation, and momentum all behave realistically under changing conditions, ensuring the system is grounded in accurate simulation rather than simplified motion logic.

2.2. Mission Rules System

A potential challenge is ensuring sufficient variability in mission outcomes to avoid predictable or repetitive gameplay. To address this, the Mission Rules system randomly activates one landing pad from a set of potential targets each time a mission starts, creating varied experiences for the player. The system dynamically checks altitude and landing safety through speed and tilt analysis instead of preset triggers, ensuring every attempt requires active control rather than memorization. By relying on data directly from the rocket's live physics output, the mission outcomes remain unpredictable yet fair, giving players authentic variation across each gameplay session.

2.3. Heads-Up Display (HUD) System

Another challenge concerns the potential performance overhead and cognitive load introduced by continuous UI updates. The HUD system mitigates this by using lightweight text and image updates only when the rocket's data changes, keeping CPU load minimal. The interface is also intentionally minimalistic, with only essential information shown at once. It automatically switches between mobile and desktop layouts, so players never experience overlapping or redundant controls. This ensures clear visibility of mission data without hindering immersion or gameplay flow, balancing functional display with visual simplicity to maintain optimal performance on all devices.

3. SOLUTION

At application launch, the system initializes at a main menu interface that serves as the entry point to the simulation environment. Once the player taps or clicks, the game transitions to the main simulation scene, where the rocket sits idle on the launch pad, awaiting input.

From this screen, the player immediately sees the rocket HUD at the bottom, showing the throttle, directional controls, and key metrics such as fuel, speed, and altitude in the top corners. The Mission Log on the top right displays objectives such as "Reach 50m altitude" or "Land safely." The rocket model is placed in a small 3D environment featuring terrain, a visible landing pad, and dynamic lighting to emphasize depth and realism.

When the player begins to press the thrust control, the game starts applying physics-based motion using Unity's Rigidbody component, simulating real gravity, thrust, and drag. The Rocket.cs script monitors fuel burn and applies upward force according to the rocket's configuration data from RocketConfig.cs. The CameraFollow3D.cs system smoothly adjusts the view, keeping the rocket centered while allowing zoom and rotation via mouse or touch.

In the background, MissionRules.cs constantly checks altitude, speed, and landing angle to determine success or failure. If the player meets the required conditions, an event triggers a success message; otherwise, a crash message appears. The game loop then ends, allowing replay. Overall, every interaction dynamically updates both the physics and on-screen UI in real time, creating a responsive and educational flight experience [13].

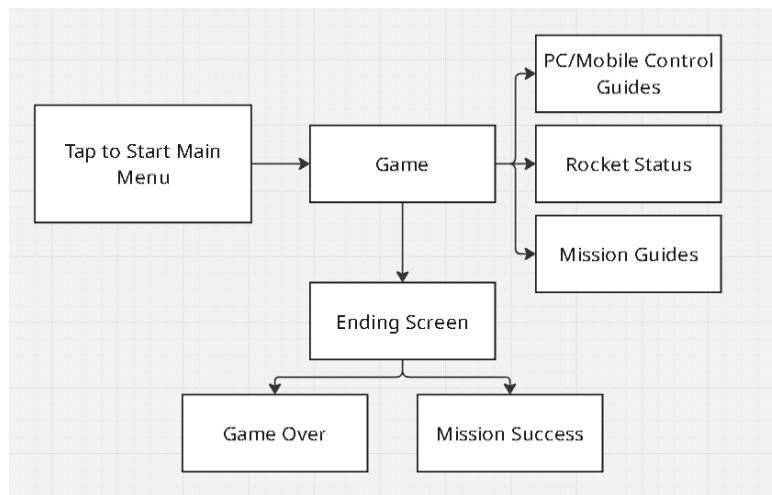


Figure 1. Overview of the solution

The Rocket Control System handles the physics-based movement of the rocket, including thrust, rotation, and fuel consumption. It uses Unity's Rigidbody component to apply realistic forces based on player input. This system defines how the rocket responds to both desktop and mobile controls, dynamically adapting to platform input in real time.



Figure 2. In-game simulation environment illustrating rocket launch and HUD elements

```

1 reference
void ApplyRotation(float yaw, float pitch, float roll, float dt)
{
    if (!body) return;

    float yawDelta = (config ? config.YawDegreesPerSec : 120f) * yaw * dt;
    float pitchDelta = (config ? config.PitchDegreesPerSec : 90f) * pitch * dt;
    float rollDelta = (config ? config.RollDegreesPerSec : 60f) * roll * dt;

    // Pure world axes (stable, non-camera-aware)
    Quaternion qYaw = Quaternion.AngleAxis(yawDelta, Vector3.up);
    Quaternion qPitch = Quaternion.AngleAxis(pitchDelta, Vector3.right);
    Quaternion qRoll = Quaternion.AngleAxis(rollDelta, Vector3.forward);

    Quaternion target = qYaw * body.rotation * qPitch * qRoll;
    body.MoveRotation(target);
}

// Real force (Newtons) so mass & gravity matter
1 reference
void ApplyThrust(float dt)
{
    if (!body) return;

    float burn = Mathf.Min(fuel, (config ? config.BurnRatePerSec : 1.5f) * dt);
    fuel -= burn;

    float nominal = (config ? config.BurnRatePerSec : 1.5f) * dt;
    float throttle = nominal > 0f ? burn / nominal : 0f;

    float forceN = (config ? config.ThrustForce : 2200f) * throttle;

    Vector3 axis = transform.up;
    if (config && config.ThrustAxis == ThrustAxis3D.Forward)
        axis = transform.forward;

    body.AddForce(axis * forceN, ForceMode.Force);
}

```

Figure 3. Core Rocket Control System implementation in the Rocket.cs script

In the Rocket.cs script, the Awake function initializes the Rigidbody and sets up gravity, drag, and fuel capacity using the RocketConfig ScriptableObject. During Update, the script reads either keyboard or mobile button inputs and converts them into yaw, pitch, roll, and thrust variables. The FixedUpdate function then applies these controls to physics, using ApplyRotation and ApplyThrust to simulate motion. Fuel is reduced proportionally to the burn rate, and the rocket's drag increases when coasting, mimicking atmospheric resistance. The ApplyRotation function calculates rotation deltas based on configuration parameters, while ApplyThrust adds a force vector along the rocket's thrust axis. The UpdateFx function controls engine visuals and audio feedback to indicate active thrust. Altitude, velocity, and speed are continuously updated for use by other systems like MissionRules and RocketHUD, allowing for integrated mission tracking and real-time user feedback.

The Mission Rules System monitors the player's progress toward mission objectives, such as reaching a target altitude and safely landing. It uses collision detection and event triggers to determine whether the player succeeded or failed, linking directly to the Rocket component for live data about altitude, speed, and stability [10].



Figure 4. Landing scenario demonstrating mission completion conditions

```

void OnCollisionEnter(Collision col)
{
    if (!rocket || missionOver || !rocket.hasLaunched) return;

    bool isPad = targetLandingPad && col.collider.transform == targetLandingPad;
    if (!isPad)
    {
        onCrash?.Invoke();
        missionOver = true;
        return;
    }

    bool speedOk = rocket.Speed <= landingMaxSpeed;
    float tilt = Vector3.Angle(rocket.transform.up, Vector3.up);
    bool tiltOk = tilt <= landingMaxTiltDegrees;

    if (speedOk && tiltOk && (!requireLanding || reachedAltitude))
    {
        onSafeLanding?.Invoke();
        missionOver = true;
    }
    else
    {
        onCrash?.Invoke();
        missionOver = true;
    }
}

```

Figure 5. Mission Rules system logic for altitude and landing evaluation

In Mission Rules.cs, the Start method activates one random landing pad from the provided list and deactivates others, ensuring a unique target per mission. The Update method checks if the rocket has reached the target altitude. Once achieved, it triggers an event to acknowledge completion. When the rocket collides with any object, the On Collision Enter function determines whether the impact was with the active landing pad and evaluates two conditions: the rocket's speed and its tilt angle relative to vertical. If both are within safe limits, a successful landing event is triggered; otherwise, the crash event runs. The script also uses Unity Events, which are easily assigned through the Inspector, allowing designers to add sound, text, or animation responses without editing code. This modular structure separates mission logic from physics control, ensuring that mission progression can be customized independently of the rocket's flight mechanics.

The Rocket HUD System provides real-time user feedback during gameplay [11]. It displays fuel, speed, and altitude while automatically switching between mobile and desktop layouts. This system bridges gameplay data and user interface, ensuring that players can monitor critical stats and control prompts seamlessly during flight.



Figure 6. Heads-Up Display (HUD) showing real-time flight parameters

```

keyboardControls.SetActive(rocket.useMobileControls == false);
touchControls.SetActive(rocket.useMobileControls);
quitGameButton.gameObject.SetActive(rocket.useMobileControls == false && Application.platform == RuntimePlatform.WebGLPlayer == false);

if (quitGameButton)
{
    quitGameButton.onClick.AddListener(() => Application.Quit());
}
}

0 references
void Update()
{
    if (!rocket) return;

    if (fuelText) fuelText.text = $"Fuel: {rocket.Fuel:0.0}";
    if (speedText) speedText.text = $"Speed: {rocket.Speed:0.0} m/s";
    if (altitudeText) altitudeText.text = $"Altitude: {rocket.Altitude:0.0} m";

    if (fuelBar) fuelBar.fillAmount = rocket.Fuel / maxFuel;
}

```

Figure 7. Rocket HUD system implementation for cross-platform UI adaptation

The Rocket HUD. cs script begins by detecting whether the player is using mobile or desktop controls and activates the correct interface layout. The Start function links UI elements like Text Mesh Pro labels and progress bars to the rocket's live data. A Quit button is also initialized for standalone versions. During Update, the script continuously refreshes the displayed values by accessing the rocket's properties: Fuel, Speed, and Altitude [14]. The fuel bar fill amount is calculated by dividing the remaining fuel by the maximum stored at startup. If any of the text fields or images are missing, the script gracefully skips those updates to avoid errors. The system's strength lies in its real-time data reflection and dynamic platform adjustment, allowing users to see immediate feedback on every thrust or rotation change. This ensures the player remains informed of mission-critical stats without interrupting gameplay or obscuring the rocket's field of view.

4. EXPERIMENT

This study examined whether interacting with Don't Crash the Rocket supported players' intuitive understanding of motion, thrust, and controlled landing. Rather than testing formal physics knowledge, the evaluation focused on player perception of learning, clarity of physical cause-and-effect, and the effectiveness of interactive simulation as a teaching medium.

Ten participants engaged with the simulation by completing a gameplay session that required active thrust control, altitude management, and a landing attempt. Following gameplay, participants completed a written survey using a five-point Likert scale ranging from strongly disagree to strongly agree. Survey items addressed the clarity of physical feedback, the relationship between player input and motion outcomes, and perceived improvement in

understanding rocket behavior. To encourage honest reflection, responses were collected without identifying information. Collected data was compiled and reviewed to determine whether the simulation achieved its instructional intent of promoting experiential learning through real-time interaction.

Survey Statement	Avg Score (1-5)
The game helped me visualize how thrust influences movement	4.3
The physics felt understandable through gameplay	4.1
On-screen feedback made motion relationships clear	4.5
Hands-on control improved my comprehension of rocket behavior	4.2
The experience increased my confidence with basic physics concepts	3.9

Figure 8. Survey statement

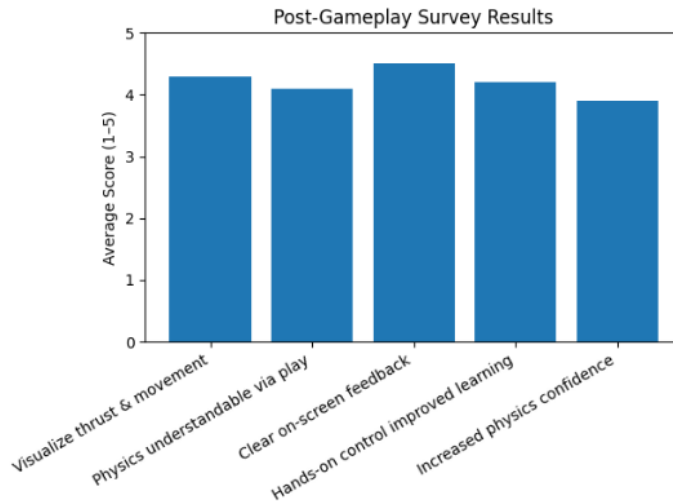


Figure 9. Post-gameplay survey results

Participant responses suggest that the simulation effectively communicated core physics ideas through direct manipulation rather than explanation. High scores related to visual feedback indicate that real-time changes in altitude, speed, and fuel reinforced understanding of cause-and-effect relationships. Slightly lower confidence-related responses imply that while the simulation supported intuition, it did not replace formal instruction or repeated practice. This aligns with the project's design intent: to function as an experiential learning tool rather than a comprehensive physics curriculum. While the limited sample size restricts broader claims, the findings support the conclusion that interactive simulation can meaningfully aid conceptual understanding by allowing players to observe and adjust physical systems dynamically.

5. RELATED WORK

The paper "Using the Aerospace Modeling Simulator in the Educational Process" by Andrei V. Chernenkii explores how aerospace simulators can be incorporated into education to improve students' understanding of rocket motion and space technology. The study focuses on academic

environments where simulation tools support guided learning and teacher instruction. In contrast, Don't Crash the Rocket delivers these same concepts through a simplified game-based experience designed for independent learners. While Chernenkii's simulator is classroom oriented, our system emphasizes interactivity, accessibility, and engagement for a general audience. This makes it effective as both an educational and entertainment tool for introducing basic physics principles [1]. The study "Development of an Excel® Rocket Simulator for Application in Middle School, High-School, and University STEM Education" by Melvin Lee Hortman (2017) investigates how a spreadsheet-based simulator can predict water rocket performance within 5.8 % error when calibrated. The work emphasises modelling mass change, thrust profiles and drag in a simplified educational environment. Unlike that project, Don't Crash the Rocket goes beyond prediction and gives real-time control over thrust, rotation, and landing within a gaming context. We integrate user input, live physics and landing conditions rather than static prediction alone. This makes our system more interactive and accessible while still preserving the core mechanics of rocket motion [2].

The paper "Physical Modeling and Simulation of Reusable Rockets for Guidance, Navigation and Control (GNC) Validation" by Fari et al. (2024) addresses the development of high-fidelity reusable rocket models including sloshing, thrust-vectoring, landing-leg deployment and touchdown dynamics for GNC verification. Their work emphasises multi-body physics and designer-friendly reconfiguration of simulation fidelity. In contrast, Don't Crash the Rocket emphasises simplified real-time interactive physics rather than high-fidelity validation for aerospace missions. While their model is research-grade, ours is game-grade and educational-oriented. We provide live player control, mission variation and HUD feedback that their simulation does not, making our system more accessible and engaging for a broader audience [3].

6. CONCLUSIONS

While Don't Crash the Rocket successfully demonstrates realistic motion and physics through an engaging simulation, it has several limitations. The first limitation is its simplified physics model, which, while effective for learning, does not yet account for complex aerodynamic effects such as air density variation, thrust vectoring, or multi-stage propulsion [15]. These omissions make the experience less accurate for advanced users who might seek higher scientific precision. Another limitation is visual variety. The current version features a single environment with minimal terrain variation, which can reduce long-term engagement for players seeking challenge diversity. Additionally, there is limited feedback on user performance beyond success or failure, meaning that detailed post-mission analytics such as flight efficiency or stability metrics are not yet implemented. In terms of accessibility, the current mobile control scheme works well but could be improved with better touch sensitivity calibration and tutorial guidance for new players. If given more time, future updates would include enhanced environments, data-driven learning feedback, and adjustable physics modes to support both casual users and advanced learners. Expanding multiplayer or leaderboard features would also motivate repeated play and learning comparison, helping the game evolve from a single-player demo into a complete educational training tool.

REFERENCES

- [1] Chernenkii, Andrei V. "Using of the aerospace modeling simulator in the educational process." 2020 V International Conference on Information Technologies in Engineering Education (Inforino). IEEE, 2020.
- [2] Hortman, Melvin Lee. "Development of an Excel® Rocket Simulator for Application in Middle School, High-School, and University STEM Education." (2017).
- [3] Fari, Stefano, et al. "Physical modeling and simulation of reusable rockets for GNC verification and validation." *Aerospace* 11.5 (2024): 337.

- [4] Banda, Herbert James, and Joseph Nzabahimana. "The impact of physics education technology (PhET) interactive simulation-based learning on motivation and academic achievement among malawian physics students." *Journal of Science Education and Technology* 32.1 (2023): 127-141.
- [5] Wang, Liang-Hui, et al. "Effects of digital game-based STEM education on students' learning achievement: A meta-analysis." *International Journal of STEM Education* 9.1 (2022): 26.
- [6] Ekici, Erhan. "" Why Do I Slog through the Physics?" Understanding High School Students' Difficulties in Learning Physics." *Journal of Education and Practice* 7.7 (2016): 95-107.
- [7] Kanev, Kamen, and Tomoyuki Sugiyama. "Design and simulation of interactive 3D computer games." *Computers & Graphics* 22.2-3 (1998): 281-300.
- [8] Jones, Paul, Robert Newbery, and Philip Underwood. "Enhanced entrepreneurial learning through visual experiential learning." *Entrepreneurship Education: New Perspectives on Entrepreneurship Education*. Emerald Publishing Limited, 2017. 197-211.
- [9] Bonhomme, Patrice. "Scheduling and control of real-time systems based on a token player approach." *Discrete Event Dynamic Systems* 23.2 (2013): 197-209.
- [10] Kockara, Sinan, et al. "Collision detection: A survey." 2007 IEEE International Conference on Systems, Man and Cybernetics. IEEE, 2007.
- [11] Rubins, David, et al. "Real-time user feedback to support clinical decision support system improvement." *Applied clinical informatics* 13.05 (2022): 1024-1032.
- [12] Bui, Lam Thu, and Zbigniew Michalewicz. "An evolutionary multi-objective approach for dynamic mission planning." *IEEE Congress on Evolutionary Computation*. IEEE, 2010.
- [13] Ruiz, Sergio, Carlos Aguado, and Romualdo Moreno. "Educational simulation in practice: a teaching experience using a flight simulator." *Journal of Technology and Science Education* 4.3 (2014): 181-200.
- [14] Zilliac, Gregory, and M. Karabeyoglu. "Hybrid rocket fuel regression rate data and modeling." 42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit. 2006.
- [15] Feng, Dakui, et al. "Numerical study on hydrodynamic behavior of flexible multi-stage propulsion foil." *AIP Advances* 11.3 (2021).