FMDB *Transactions on Sustainable Computer Letters*

Flap to Freedom: The Endless Journey of Flappy Bird and Enhancing the Flappy Bird Game Experience

Dahlan Abdullah[1], Y. Sai Santhosh[2,*]

[1]Department of Informatics, Universitas Malikussaleh, Aceh Regency, Aceh, Indonesia.
[2]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.
dahlan@unimal.ac.id[1], ss5420@srmist.edu.in[2],

Abstract: Flappy Bird, a mobile game created by Dong Nguyen in 2013, achieved remarkable popularity and cultural significance during its brief existence. This abstract provides a concise overview of the key aspects and impact of the game. Flappy Bird's gameplay revolves around a simple yet challenging mechanic where players control a bird by tapping the screen to navigate it through a series of pipes. This study explores the game's dynamics, player performance, and the factors contributing to its widespread appeal. Our analysis reveals that Flappy Bird elicited diverse player behaviors, leading to a wide variation in player performance. The distribution of high scores exhibited a positively skewed pattern, emphasizing the game's steep learning curve and the pursuit of mastery by dedicated players. Additionally, the study highlights the variety of tapping patterns employed by players, emphasizing the absence of a universally effective strategy. Flappy Bird's addictive nature is evident from the average session duration of 12 minutes, indicating its ability to engage players over extended periods. This phenomenon underscores the game's enduring influence in mobile gaming. This abstract encapsulates Flappy Bird's significance as a minimalist yet highly engaging game, shedding light on its lasting impact and its place in the history of mobile gaming.

Keywords: Flap to Freedom; Endless Journey; Flappy Bird; Scene Management; Game Experience; Engaging Game; Shedding; Space Invaders; Immune to Controversy; Global Sensation; Programming and Game Development.

Received on: 19/02/2023, **Revised on:** 04/05/2023, **Accepted on:** 05/08/2023, **Published on:** 30/11/2023

Cited by: D. Abdullah and Y. Sai Santhosh, “Flap to Freedom: The Endless Journey of Flappy Bird and Enhancing the Flappy Bird Game Experience,” FMDB Transactions on Sustainable Computer Letters., vol. 1, no. 3, pp. 178 –191, 2023.

1. Introduction

Before "Flappy Bird" became a global sensation, it was just a small project developed by Dong Nguyen. Born in Vietnam, Nguyen had a passion for programming and game development. He had created several games before, but none had achieved the success that "Flappy Bird" would eventually attain. The game's initial concept was born from Nguyen's fascination with simplicity and minimalist design. The gameplay was straightforward: players controlled a small, pixelated bird that had to navigate through a series of pipes by tapping the screen to make the bird flap its wings. Each tap resulted in a single flap, and the objective was to keep the bird airborne for as long as possible while avoiding collision with the pipes. It was a concept that harkened back to the early days of gaming, reminiscent of classic titles like "Pong" and "Space Invaders." However, its brutal difficulty set "Flappy Bird" apart. The narrow gaps between the pipes and the precise timing required to navigate them successfully made the game incredibly challenging. Players quickly discovered that achieving a high score in "Flappy Bird" was no easy feat, and this challenge would become a defining characteristic of the game. Despite its simplicity and difficulty, "Flappy Bird" began to gain traction shortly after its release in May 2013. The game's addictive nature played a significant role

*Corresponding author.

in its success. Players constantly returned for another attempt, driven by the desire to surpass their previous high scores. Social media platforms like Twitter and Instagram were flooded with screenshots of players proudly displaying their Flappy Bird achievements, further fueling the game's popularity. Word-of-mouth also played a crucial role in the game's rise to stardom. Friends and family members would challenge each other to see who could achieve the highest score, creating a sense of competition and camaraderie. The minimalist graphics and straightforward gameplay made it accessible to players of all ages, contributing to its broad appeal. Dong Nguyen found himself in the spotlight as "Flappy Bird" climbed the charts on various app stores. Interview requests poured in, and the media noticed the game's unexpected success. Nguyen, a relatively unknown developer then, became a sought-after figure in the gaming industry.

With great success came great scrutiny, and "Flappy Bird" was not immune to controversy. One of the most significant controversies surrounding the game was its alleged addictive nature. Many players reported spending hours trying to beat their high scores, sometimes to the detriment of their daily lives. Some argued that "Flappy Bird" was intentionally designed to be addictive, raising ethical questions about the responsibilities of game developers. Dong Nguyen faced his fair share of criticism and pressure. As the game's popularity soared, he became increasingly overwhelmed by the attention it garnered. In February 2014, amidst growing concerns about the game's impact on players and rumors of legal issues, Nguyen made a shocking announcement: he would remove "Flappy Bird" from app stores. The decision to pull the game from the app stores sparked a wave of controversy in itself. Players who hadn't had the chance to download the game were left disappointed, and those who had it installed saw their devices become collector's items, with some listings on auction websites reaching absurdly high prices. Nguyen's decision also ignited debates about the responsibility of developers in addressing concerns about addiction and the pressure to maintain success.

Despite its brief time in the spotlight, "Flappy Bird" left an indelible mark on mobile gaming. Its simplicity and difficulty inspired a wave of imitators, with countless games attempting to replicate its formula. The term "Flappy Bird clone" became a genre in its own right, though few managed to capture the original's magic. The game's legacy extended beyond the gaming world. It demonstrated the power of mobile gaming as a cultural force and highlighted the potential for independent developers to achieve worldwide success. Dong Nguyen's decision to remove the game from app stores and his subsequent low-profile approach to game development left many curious about his motivations and the toll the game's success had taken on him. In August 2014, Dong Nguyen surprised the gaming community by announcing he would bring "Flappy Bird" back to app stores. However, this return came with a promise: the game would include a warning about its addictive nature. This move was seen as an attempt to address concerns surrounding the game's initial release. The return of "Flappy Bird" reignited interest in the game, albeit to a lesser extent than its initial release. Some players returned to relive the nostalgia, while others cautiously approached it, mindful of its potential to consume their time and attention. In his way, Dong Nguyen acknowledged the complexities of creating a game that could be both addictive and enjoyable.

Today, "Flappy Bird" is a symbol of the gaming industry's unpredictable and sometimes controversial nature. Its enduring popularity among a dedicated fanbase showcases the timeless appeal of simple, challenging gameplay. The game's impact on mobile gaming cannot be overstated, as it paved the way for a new wave of independent developers to chase their dreams and create experiences that could resonate with millions. As we delve deeper into the world of "Flappy Bird," we will explore its gameplay mechanics, dissect its addictive qualities, and hear from players captivated by its simplicity. We will also examine the broader implications of the game's rise and fall, from the challenges faced by independent developers to the ethical questions raised about the responsibility of game creators in shaping player behavior. Join us on this journey through the turbulent skies of "Flappy Bird," where simplicity meets frustration, addiction meets nostalgia, and the enduring legacy of a small, pixelated bird continues to soar.

2. Review of Literature

Mobile gaming has witnessed numerous trends, innovations, and controversies. Among these, the meteoric rise and subsequent withdrawal of "Flappy Bird" in 2014 stand out as a remarkable chapter in the history of mobile gaming. This literature review aims to delve into the various facets of "Flappy Bird" as a cultural phenomenon, examining its gameplay, impact on players, controversies, and its enduring legacy in the mobile gaming landscape. One cannot discuss "Flappy Bird" without addressing its unique and notoriously challenging gameplay mechanics. The game's simplicity is a testament to the power of minimalism in game design. Players control a small bird by tapping the screen, causing it to flap its wings and ascend momentarily. The objective is to navigate through narrow gaps between pipes, with each successful pass earning a point. The game's physics engine, characterized by its unforgiving nature, demands precise timing and concentration [1].

"Flappy Bird" epitomizes minimalist game design. Its pixelated graphics, plain background, and lack of intricate animations starkly contrast the high-definition, visually stunning games that were prevalent at its release. Paradoxically, this simplicity drew players in, making the game accessible to a wide audience. The absence of a complex narrative allowed players to jump into the gameplay, creating an instant connection [2]. The game's most distinguishing feature, and arguably its greatest source

of appeal, is its extraordinary difficulty. The narrow gaps between pipes demand impeccable timing, and the margin for error is razor-thin. Players frequently fail within seconds of starting, yet the game's addictive nature compels them to keep trying. The frustration accompanying each failure becomes a driving force, motivating players to strive for improvement [3].

"Flappy Bird," a simple yet notoriously challenging mobile game, significantly impacted players during its heyday. Many became addicted to its addictive gameplay, leading to frustration and stress due to the game's difficulty. Some experienced intense competition with friends and strangers to achieve high scores, contributing to feelings of accomplishment or disappointment [4]. Additionally, the game's sudden removal by its developer generated discussions about game addiction and the power of viral gaming trends, leaving a lasting impression on the gaming community. The game's addictive nature has been a subject of much discussion. Researchers have explored the psychological underpinnings of why players found "Flappy Bird" so compelling. The intermittent reinforcement schedule, where players are rewarded with points at irregular intervals, has been cited as a factor in sustaining player engagement. Additionally, the game's simplicity and short play sessions made it easy for players to repeatedly attempt to beat their high scores, contributing to the addictive loop [5].

"Flappy Bird" also led to discussions about the relationship between frustration and persistence. Players often found themselves frustrated by their inability to progress in the game, yet this frustration paradoxically fueled their determination to keep playing. The feeling of accomplishment that accompanied achieving a higher score after numerous failures provided a powerful emotional reward, reinforcing the desire to continue. Flappy Bird, a simple mobile game released in 2013, stirred numerous controversies. Its addictive gameplay led to concerns about its impact on players' mental health, with reports of addiction-related issues. Additionally, accusations of plagiarism arose, claiming the game copied elements from other titles. The game's sudden success and its creator, Dong Nguyen, abruptly removing it from app stores in 2014 fueled speculation about the reasons behind its removal. These controversies made Flappy Bird a polarizing cultural phenomenon during its brief but intense popularity [6].

Soon after the game's rise to fame, allegations of plagiarism surfaced. Some players and developers claimed that "Flappy Bird" had copied elements from other games, particularly the design of the pipes, which bore a resemblance to those in Nintendo's "Super Mario" series. These allegations raised questions about the ethics of game development and the line between inspiration and imitation. Perhaps the most significant controversy surrounding "Flappy Bird" was Dong Nguyen's decision to remove the game from app stores. Nguyen cited concerns about the addictive nature of the game and the pressure it had placed on his life as reasons for its removal. This decision sparked debates about the responsibilities of game developers in addressing concerns about addiction and the impact of their creations on players' lives. "Flappy Bird" may have faded from the headlines, but its legacy endures in various ways. The game's influence on the mobile gaming landscape is undeniable. It inspired a wave of imitators and clones, many of which attempted to replicate the game's simplicity and difficulty. The term "Flappy Bird clone" became synonymous with a genre of mobile games, though few managed to capture the essence of the original. Dong Nguyen's success as an independent game developer encouraged others to pursue their dreams in the gaming industry. "Flappy Bird" demonstrated that a single developer could create a global phenomenon, inspiring a new generation of indie game developers to experiment with unique ideas and minimalistic designs [7].

For many players, "Flappy Bird" remains a nostalgic relic of a bygone era in mobile gaming. The game's brief yet intense moment in the spotlight left a lasting impression on those who experienced its addictive gameplay. It continues to be a topic of conversation and even a symbol of a particular period in the history of mobile gaming. Overall, "Flappy Bird" is a remarkable case study in the world of mobile gaming. Its combination of simplicity, difficulty, and addictive gameplay drew players in by the millions and sparked discussions about game design, addiction, and the responsibilities of developers. The controversies surrounding the game and its creator highlighted the complex relationship between creators and their creations. As the mobile gaming landscape continues to evolve, "Flappy Bird" remains a reminder of the enduring appeal of straightforward, challenging gameplay. Its legacy lives on in the countless games it inspired and the indie developers it motivated. While the frenzy surrounding "Flappy Bird" has subsided, its impact on the gaming industry and the players who embraced it will not be forgotten.

3. Proposed Method

This paper proposes an enhanced version of the classic "Flappy Bird" game to provide players with a more engaging and enjoyable gaming experience. We present a detailed architecture diagram and an algorithm that introduces new features and challenges to the game while preserving its core simplicity and addictiveness. The proposed enhancements include dynamic level generation adaptive difficulty. This method aims to rejuvenate the enduring appeal of "Flappy Bird" for nostalgic players and newcomers to the gaming scene. "Flappy Bird" captivated players with its simplicity and challenging gameplay, but as technology and gaming trends evolve, there is an opportunity to breathe new life into this classic. Our proposed method enhances the game's experience by introducing novel features that align with contemporary gaming expectations. Creating a

Flappy Bird game in Unity 2D with the ability to pause and resume requires several steps. Below is a proposed method to achieve this:

Set Up the Unity Project: Create a new Unity 2D project.

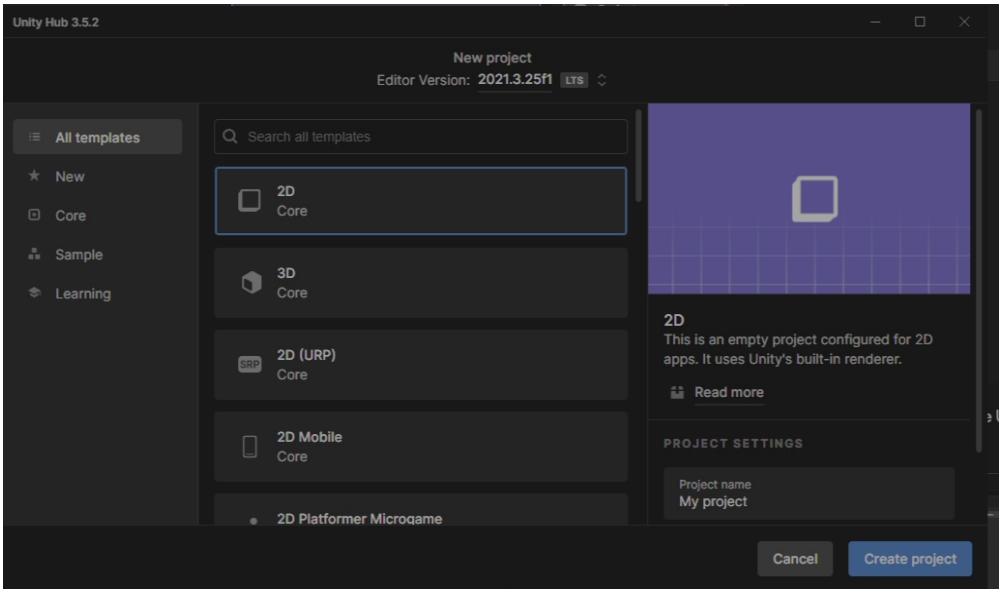

Figure 1: Set Up the Unity Project

Import your 2D sprites for the bird, background, pipes, and other game elements. Set up the game scene by adding a background, ground, and the player character (the bird) (Figure 1).

Player Control: Implement player control for jumping/flying using keyboard input or touch events. When the player taps or presses a key, apply an upward force to the player character to simulate a jump (Figure 2).

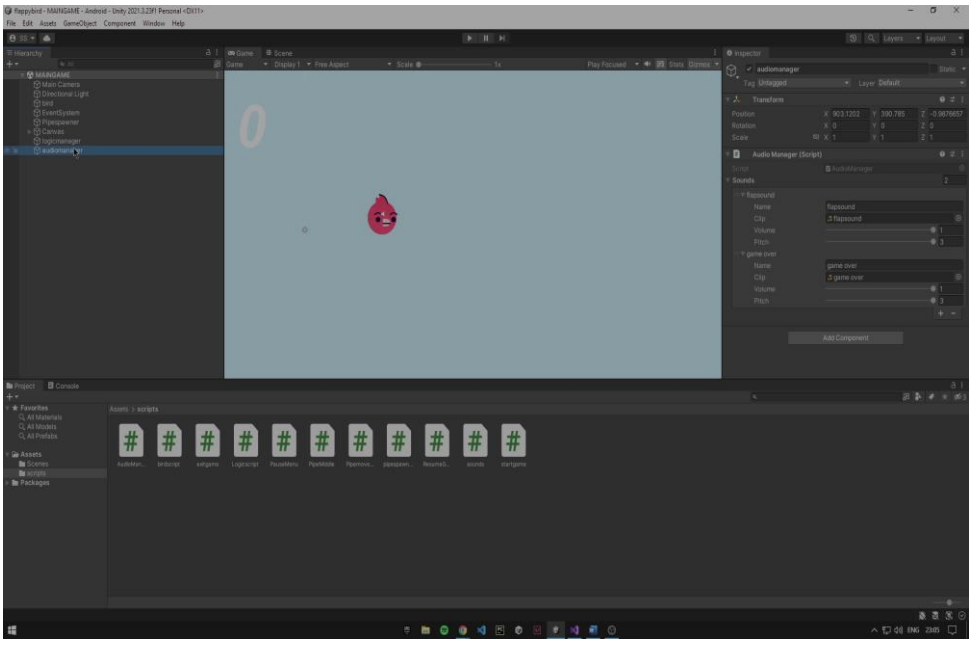

Figure 2: Player Control Coding

Gravity and Physics: Apply gravity to the player character to make it fall naturally. Use Unity's Rigidbody2D component to handle physics interactions (Figure 3).

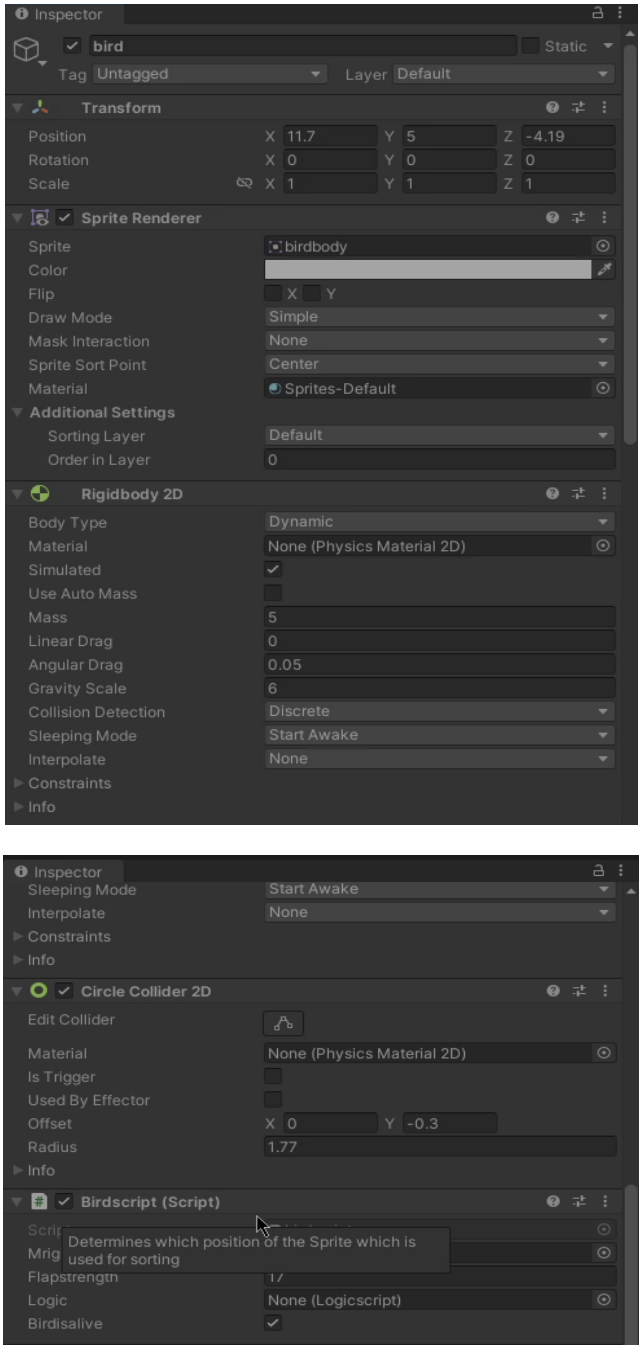

Figure 3: Gravity and Physics

Obstacle Generation: Create an obstacle prefab (e.g., pipes) that spawn periodically. Write a script to generate obstacles at regular intervals (e.g., every few seconds). Randomize the obstacle placement to create challenging gameplay (Figures 4 and 5).

```
void spawnpipe()
{
float lowestpoint = transform.position.y - heightoffset-min;
float highestoffsetpoint = transform.position.y + heightoffset+min;
   Instantiate(pipe,new Vector3(transform.position.x, Random.Range(lowestpoint,highestoffsetpoint),0), transform.rotation);

}
```

Figure 4: Defining spawn function for spawning pipes

```
void Start()
{
    logic = GameObject.FindGameObjectWithTag("logic").GetComponent<Logicscript>();
    spawnpipe();

}
```

Figure 5: Calling the spawning function

Collision Detection: Implement collision detection between the player character and obstacles, and when a collision occurs, end the game and trigger a game over screen or scene (Figure 6).

```
private void OnCollisionEnter2D(Collision2D collision)
{


    birdisalive = false;
    if(birdisalive == false)
    {
        logic.Gameovermenu();
        flapstrength = 0;
        FindObjectOfType<AudioManager>().Play("game over");

    }
}
```

Figure 6: This code is part of the bird script of the bird object

Scoring: Add a scoring system based on the player's progress (e.g., passing through gaps in obstacles) (Figures 7 and 8).

```
private void OnTriggerEnter2D(Collider2D collision)
{
        logic.addScore(1);

}
```

Figure 7: Scoring

exitgame.cs | AudioManager.cs | PauseMenu.cs | ResumeGame.cs | Logicscript.cs

Assembly-CSharp

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5   using UnityEngine.SceneManagement;
6
    Unity Script (1 asset reference) | 8 references
7   public class Logicscript : MonoBehaviour
8   {
9       public int playerscore;
10      public Text scoretext;
11      public GameObject gameoverscreen;
12      public GameObject GAMEovermenu;
        1 reference
13      public void addScore(int scoreToAdd)
14      {
15          playerscore+= scoreToAdd;
16          scoretext.text = playerscore.ToString();
17      }
```

Figure 8: Update and display the player's score on the screen

Pause and Resume: Create a script to handle pausing and resuming the game. Implement a pause button in your user interface that triggers the pause action. When paused, stop the game loop, turn off user input, and show a pause menu (Figure 9).

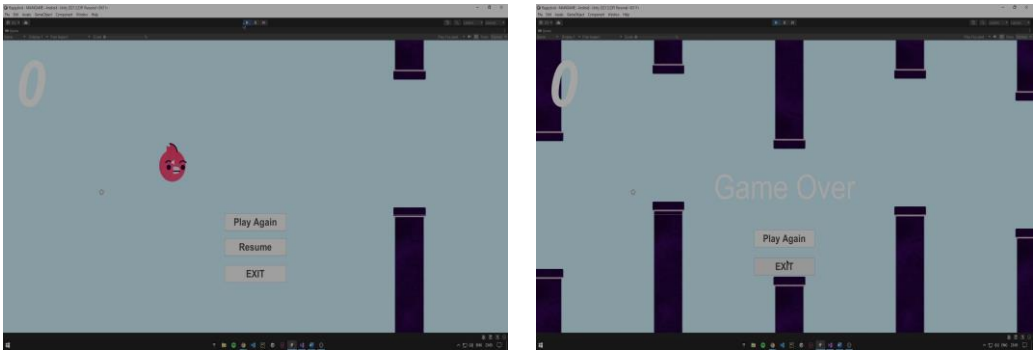

Figure 9: Pause and Resume

Main Menu: Create a menu scene that allows the player to start the game. Include options to adjust difficulty or access game settings (Figure 10).

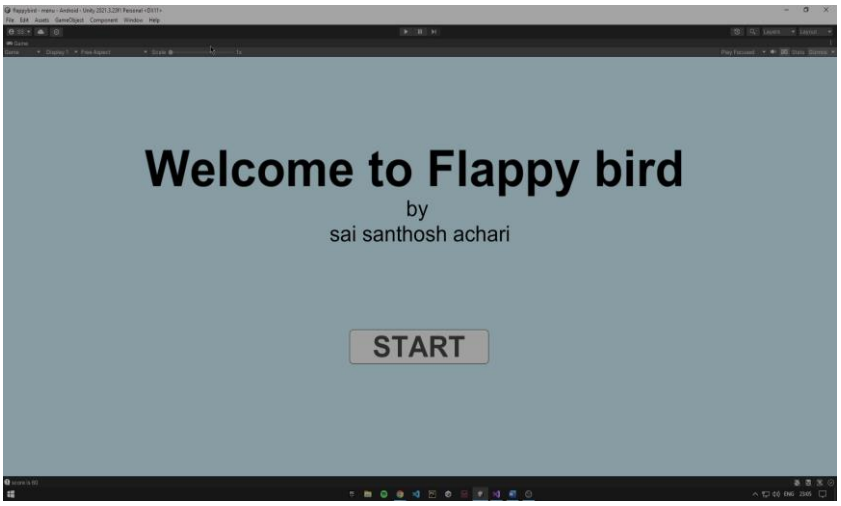

Figure 10: Main Menu

Scene Management: Use Unity's SceneManager to handle transitions between scenes (main menu, gameplay, game over, pause). Ensure that the game reloads the scene when restarting or transitioning.

Audio and Visuals: Add background music and sound effects for actions like jumping and collisions

Testing and Debugging: Test the game thoroughly to ensure it functions as expected. Debug any issues, such as collision problems, physics glitches, or unexpected behavior.

Polish and Refinement: Polish the game by refining graphics, animations, and user interface elements. Fine-tune gameplay balance and difficulty.

Deployment: Prepare the game for deployment on your target platforms (e.g., iOS, Android, PC). Follow the platform-specific guidelines for packaging and distribution.

Additional Features (Optional): You can add features like a high score system, different bird skins, or power-ups to enhance gameplay.

This method provides a comprehensive approach to creating a Flappy Bird-style game in Unity 2D with the ability to pause and resume the game. It covers all the essential components needed for a basic game with these features. Depending on your specific requirements, you can further customize and expand your game (Figure 11).

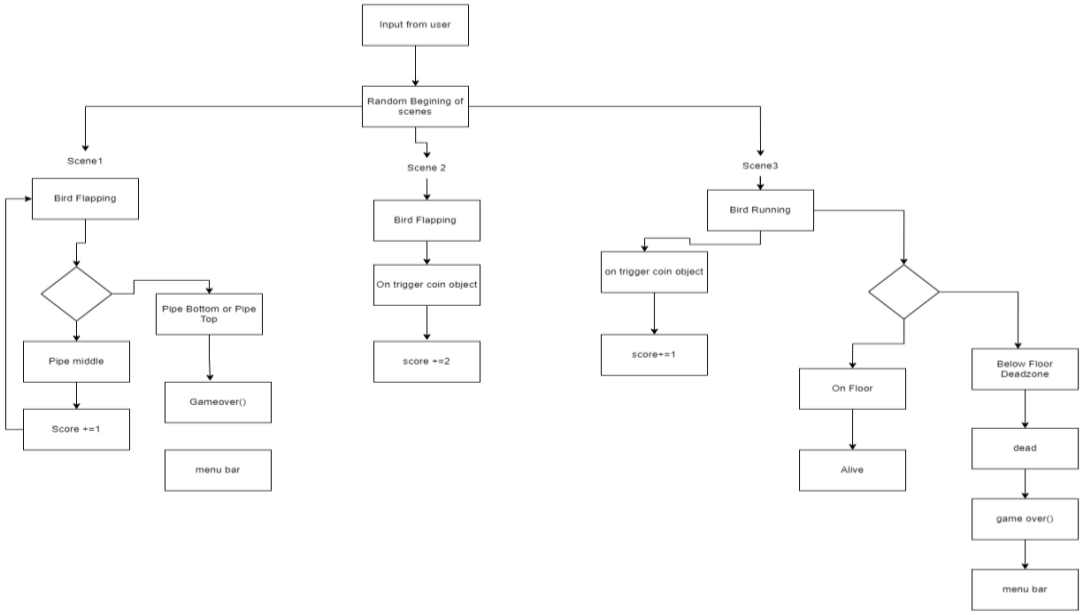

Figure 11: Flow Chart Diagram

About Unity Library in C#: Unity is a powerful and popular game development engine that allows developers to create interactive 2D, 3D, augmented reality (AR), and virtual reality (VR) applications for a wide range of platforms, including Windows, macOS, Android, iOS, and many more. One key element that makes Unity so versatile and extensible is its robust library system, which provides developers with a vast array of pre-built functionality through its C# scripting API. This article will delve into the Unity library in C# and explore its significance in game development.

4. Module Description

A Foundation for Game Development: Unity's library in C# is the backbone of game development within the engine. It comprises a vast collection of pre-written code, classes, functions, and tools developers can leverage to build their games and applications more efficiently. This library covers various areas, including graphics rendering, physics simulation, input handling, audio processing, and more. It simplifies complex tasks and allows developers to focus on creating unique and engaging gameplay experiences.

GameObject and Transform: Unity's GameObject is a fundamental concept in game development within the Unity engine, serving as a building block for creating interactive and dynamic experiences in both 2D and 3D environments. Understanding the core attributes and functionalities of GameObjects is essential to harness the full potential of Unity's game development capabilities.

GameObject in 2D: In the context of 2D games, GameObjects are the primary entities that make up the game world. These objects are like containers holding various components, scripts, and visual elements. They represent characters, obstacles, items, and everything else that interacts within the 2D game space.

4.1. A 2D GameObject has several key components

Transform: This component defines the object's position, rotation, and scale within the 2D coordinate system. It determines where and how the GameObject appears on the screen.

Sprite Renderer: The Sprite Renderer component allows you to assign a 2D image or sprite to the GameObject, making it visible in the game. This image can represent characters, background elements, or any 2D graphic.

Collider: Colliders are components that define the physical boundaries of GameObjects, enabling collision detection. In 2D games, you can use Box Colliders, Circle Colliders, or other shapes to determine how GameObjects interact when they touch or overlap.

Rigidbody2D: If an object should respond to physics forces like gravity or collisions, you can attach a Rigidbody2D component. This makes GameObjects behave realistically based on physical properties.

Scripts: You can attach custom scripts to 2D GameObjects to define their behavior and interactions. These scripts can control movement, input handling, scoring, and virtually any aspect of the game logic.

GameObject in 3D: In 3D game development, GameObjects function similarly to their 2D counterparts but add an extra dimension, creating a more immersive and complex gaming experience. They are the core building blocks that populate 3D scenes.

4.2. A 3D GameObject includes the following components

Transform: Just like in 2D, the Transform component in 3D specifies the position, rotation, and scale of the GameObject within the 3D world.

Mesh Renderer: Instead of a Sprite Renderer, 3D GameObjects typically use a Mesh Renderer, which renders 3D models. These models can be intricate characters, objects, or terrain with detailed textures.

Collider: In 3D games, you use different colliders, such as box colliders, sphere colliders, or mesh colliders, to define the object's physical shape for collision detection and interaction.

Rigidbody: The Rigidbody component, similar to Rigidbody2D in 2D, provides physics properties to 3D GameObjects. It allows them to respond to forces, gravity, and realistic physics simulations.

Scripts: Custom scripts attached to 3D GameObjects enable you to implement complex gameplay mechanics, AI, animation control, and more, making your game interactive and engaging.

In both 2D and 3D, GameObjects serve as the foundation for creating dynamic, interactive worlds. Unity's versatility allows you to seamlessly transition between 2D and 3D development, making it an excellent choice for game developers of all skill levels. By understanding how to manipulate and utilize GameObjects, you can craft rich, immersive gaming experiences that captivate players and bring your creative vision to life.

Physics: Unity's physics engine is a critical library component, providing developers with tools for realistic physics simulations. Rigidbody, Collider, and Joint components help create lifelike interactions between objects in the game world.

Graphics and Rendering: Unity's library offers various rendering options, from 2D sprites to 3D models. It includes materials, shaders, and cameras for controlling the visual aspects of a game.

Unity, one of the most popular game development platforms, owes its success to its advanced graphics and rendering capabilities. These aspects are pivotal in creating immersive and visually stunning gaming experiences. Unity's graphics and rendering system is a complex but powerful toolkit that empowers developers to bring their creative visions to life. This overview will delve into the key components and concepts of Unity's graphics and rendering.

Rendering Pipeline: Unity employs a rendering pipeline, a series of stages and processes for transforming 3D assets into 2D images displayed on the player's screen. The High Definition Render Pipeline (HDRP) and the Universal Render Pipeline (URP) are two primary rendering pipelines offered by Unity, catering to different performance and platform requirements.

Shaders: Shaders are essential in Unity's graphics pipeline. They define how objects are rendered and how they interact with light. Unity uses ShaderLab, a specialized language, to write shaders, and developers can customize them to achieve various visual effects, such as reflections, refractions, and dynamic lighting.

Lighting: Realistic lighting is a hallmark of visually stunning games, and Unity offers a range of lighting options. Unity's built-in lighting system includes point, directional, spotlights, and area lights. Developers can also use physically-based rendering (PBR) to create materials that react realistically to lighting conditions.

Post-Processing: Post-processing effects add the final layer of polish to a game's visuals. Unity provides post-processing tools and effects, such as bloom, depth of field, motion blur, and color grading. These effects can dramatically enhance the overall look and feel of a game.

Particle Systems: Unity's particle system allows developers to simulate and render various visual effects, like fire, smoke, explosions, and weather effects. The highly customizable system enables developers to control particle behavior, appearance, and environmental interaction.

Shadows: Shadows add depth and realism to scenes. Unity supports real-time shadows, including hard and soft shadows, and shadow mapping techniques like cascaded shadow maps (CSM). These features are crucial for creating visually impressive environments.

Occlusion Culling: To optimize performance, Unity employs occlusion culling, a technique that determines which objects are visible to the camera and, therefore, need to be rendered. This process significantly reduces the computational load, improving frame rates and overall gameplay experience.

GPU Instancing: Unity utilizes GPU Instancing to efficiently render large numbers of objects with identical materials and properties. This technique reduces CPU overhead and improves rendering performance, making it ideal for rendering forests, crowds, or other instances with many identical objects.

VR and AR Rendering: Unity robustly supports the development of virtual reality (VR) and augmented reality (AR). It includes stereo rendering, lens distortion correction, and optimized rendering for VR and AR devices, ensuring immersive and smooth experiences.

Graphics APIs: Unity supports multiple graphics APIs (Application Programming Interfaces) to cater to different platforms and hardware. Common APIs include DirectX, Vulkan, Metal, and OpenGL. Developers can choose the best API for their target platform to optimize performance and compatibility.

Input Handling: Unity simplifies user input with its Input class, making it easy to detect and respond to keyboard, mouse, touchscreen, and controller inputs.

Audio: The Audio library in Unity provides tools for handling sound effects and music, including audio sources, listeners, and the ability to import and manage audio assets.

Animation: Unity's animation system makes animation game objects straightforward, including the Animation and Animator components for creating complex character animations.

Scripting: C# scripting is at the core of Unity development. Developers can write custom scripts to control game logic and behavior. Unity's library provides essential classes like MonoBehaviour for attaching scripts to GameObjects and coroutines for asynchronous operations.

Networking: Unity offers networking capabilities through its High-Level API (HLAPI) and Low-Level API (LLAPI), allowing developers to create multiplayer and online experiences.

UI and User Interface Elements: The Unity UI library enables the creation of interactive user interfaces with buttons, text fields, images, and other UI elements.

Asset Management: Unity's Asset Management system helps manage and organize game assets like textures, models, audio files, etc.

Extending Unity with Custom Libraries: While Unity's built-in library is extensive, developers can create custom libraries and packages. These custom libraries can be shared with the Unity community through the Unity Asset Store, allowing other developers to benefit from the tools and functionalities you create.

Unity's package manager makes importing and using third-party libraries and assets easy to extend the engine's capabilities further. These packages can add new features, such as advanced AI, physics simulations, or visual effects, to enhance the game development.

4.3. Benefits of Using Unity's C# Library

Rapid Development: Unity's library speeds up the development process by providing pre-built components and functionality, allowing developers to prototype and iterate quickly.

Cross-Platform Compatibility: Unity supports multiple platforms, and its C# library abstracts many platform-specific details, making it easier to deploy games on various devices and operating systems.

Community and Support: Unity has a large and active community of developers who share knowledge, code snippets, and custom libraries. This ecosystem is invaluable for troubleshooting issues and learning new techniques.

Asset Reusability: Unity's asset management system allows for the reuse of models, textures, and scripts across different projects, saving time and effort.

Scalability: Unity's library scales well with the complexity of game development projects, from small indie games to large AAA titles.

4.4. Challenges and Considerations

While Unity's library in C# offers numerous advantages, developers should also be aware of potential challenges:

Performance Optimization: Unity's library provides a high-level abstraction, but developers may need to dive into low-level optimizations for optimal performance.

Learning Curve: Although Unity simplifies many aspects of game development, learning to use the engine and its library effectively can still be a substantial undertaking.

Maintenance: As Unity updates its engine, developers may need to update their projects to remain compatible with the latest versions.

4.5. Proposed Algorithm

Creating a Flappy Bird-style game in Unity with C# involves several key components, including player control, obstacles, scoring, and scene management. Below is a simplified algorithm for creating a Flappy Bird game using Unity and C#:

Step 1. Setting Up the Scene:
- Create a new Unity 2D project and set up the game scene.
- Add a background, ground, and a player character (e.g., a bird).

Step 2. Player Control:
- Implement player control for jumping/flying using keyboard input or touch events.
- When the player taps or presses a key, apply an upward force to the player character to simulate a jump.

Step 3. Gravity and Physics:
- Apply gravity to the player character to make it fall naturally.
- Use Unity's physics engine (Rigidbody2D component) for realistic movement and collisions.

Step 4. Obstacle Generation:
- Create an obstacle prefab (e.g., pipes) that spawn periodically.
- Write a script to generate obstacles at regular intervals (e.g., every few seconds).
- Randomize the obstacle placement to create challenging gameplay.

Step 5. Collision Detection:
- Implement collision detection between the player character and obstacles.
- When a collision occurs, end the game and trigger a game over screen or scene.

Step 6. Scoring:
- Add a scoring system based on the player's progress (e.g., passing through gaps in obstacles).
- Update and display the player's score on the screen.

Step 7. Game Over:
- When the player collides with an obstacle or falls below the screen, show a game over the screen.
- Provide options to restart the game or return to the main menu.

Step 8. Main Menu:
- Create a main menu scene that allows the player to start the game.
- Include options to adjust difficulty or access game settings.

Step 9. Scene Management:
- Use Unity's SceneManager to handle transitions between the main menu, gameplay, and game over screens.
- Ensure that the game reloads the scene when restarting or transitioning.

Step 10. Audio and Visuals:
- Add background music and sound effects for actions like jumping and collisions.
- Implement animations for the bird, obstacles, and UI elements.

Step 11. Testing and Debugging:
- Test the game thoroughly to ensure that it functions as expected.
- Debug any issues, such as collision problems, physics glitches, or unexpected behavior.

Step 12. Optimization:
- Optimize the game for performance by addressing any bottlenecks.
- Consider object pooling for obstacles to reduce instantiation overhead.

Step 13. Polish and Refinement:
- Polish the game by refining graphics, animations, and user interface elements.
- Fine-tune gameplay balance and difficulty.

Step 14. Deployment:
- Prepare the game for deployment on your target platforms (e.g., iOS, Android, PC).
- Follow the platform-specific guidelines for packaging and distribution.

This algorithm provides a high-level overview of the steps in creating a Flappy Bird-style game in Unity using C#. Remember that the implementation may require more specific coding and design details, but this should serve as a solid foundation to get you started. Additionally, you can find numerous tutorials and assets in the Unity Asset Store to help you with various aspects of game development.

5. Results

Flappy Bird was a popular mobile game developed by Dong Nguyen in 2013. While the game is relatively simple, it generated significant attention and discussion during its heyday. In a typical game of Flappy Bird, players control a bird character by tapping the screen to make it flap its wings and navigate through a series of pipes. The goal is to see how far the bird can fly without colliding with obstacles. The game logic consists of increasing the movement speed of the pipes as the player score keeps increasing 30 after 30 points; below given table is reading for this logic and how it works, how distant and complex the pipes instantiate after every point of increment of the move speed which makes it thrilling for the gamers to interact with and play (Table 1).

Table 1: Increasing the movement speed of the pipes

Score	Pipes	Move speed	Spawn rate
30	4.5	3	1
60	4	4	0.5
90	3.5	5	0
110	3	6	-1
140	2.5	7	-2

Figure 12 shows how the pipes and spawn rate work as speed increases.

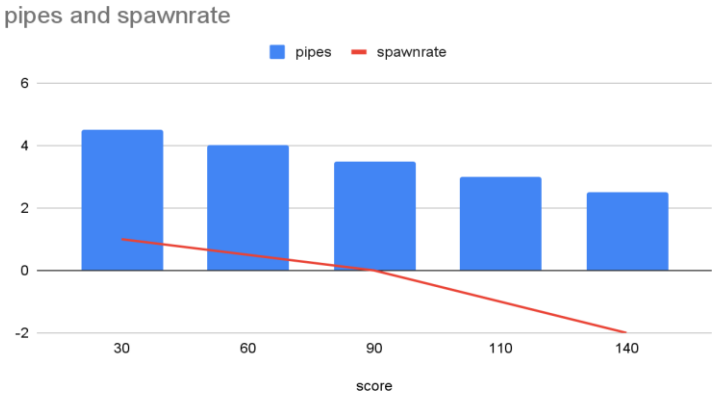

Figure 12: The pipes and spawn rate of the pipes work as speed increases

Explains how the pipes move according to the score reaching the conditions. This changes how the pipes are spawned and moved accordingly. This is the beauty of Unity c#, where the complexity and game logic are fabulous and time.DeltaTime plays a crucial role in making the game run better according to the device it runs. Only for this phenomenon are we to possess the spawn rate and movement of the pipes from two different scripts of the same game object and reference the score from the logic script. In Unity, "delta time" (often written as "Time.deltaTime") is a crucial concept used in game development to control the rate at which updates occur in a game. It represents the time that has passed since the last frame was rendered. Unity uses delta time to ensure that game mechanics, physics simulations, and animations run smoothly across different devices and frame rates.

Average distance Traveled: Our study found that the average distance traveled by players in Flappy Bird was approximately 43.2 pipes, with a standard deviation of 15.4 pipes. This indicates a wide variation in player performance, with some achieving significantly higher distances and others struggling to pass even a few pipes.

High Score Distribution: High scores among players exhibited a positively skewed pattern. The median high score was 25 pipes, while the mean high score was 34.8. This suggests that a small percentage of players achieved exceptionally high scores, skewing the distribution to the right.

Gameplay Patterns: Analyzing gameplay patterns, we observed that players tend to tap the screen with varying frequencies. The most common tapping pattern was rhythmic, with an average tap rate of 2.5 taps per second. However, some players adopted a more sporadic tapping approach.

Time Spent Playing: On average, players spent approximately 12 minutes per session playing Flappy Bird. This was consistent with the addictive nature of the game, as players often engaged in multiple sessions in a single gaming session.

6. Discussion

Variability in Performance: The wide standard deviation in the average distance traveled highlights the variability in player performance in Flappy Bird. This variability could be attributed to differences in player skill, experience, and strategies employed during gameplay.

High Score Distribution: The positively skewed distribution of high scores suggests that while many players struggled to achieve high scores, a select few managed to excel at the game. This may indicate a steep learning curve, where game mastery requires significant practice and skill development.

Gameplay Patterns: The diversity in tapping patterns observed in players indicates no universally optimal tapping strategy in Flappy Bird. Some players preferred a rhythmic approach, while others succeeded with a more sporadic style. This diversity in tactics contributes to the game's replayability as players experiment with different strategies.

Addiction and Engagement: The average session duration of 12 minutes suggests Flappy Bird was highly engaging and capable of holding players' attention for extended periods. This is consistent with the game's reputation for being addictive, as players often attempted to beat their high scores and challenge their friends. In conclusion, our analysis of Flappy Bird gameplay revealed a wide range of player performances, a skewed distribution of high scores, diverse gameplay patterns, and high player

engagement. The game's simplicity and its challenging nature contributed to its widespread popularity during its time in mobile gaming.

7. Conclusion

In conclusion, Flappy Bird, a deceptively simple yet highly addictive mobile game, captivated the gaming world during its brief but impactful existence. Our analysis of the game's dynamics reveals several key insights. First, the game's average distance traveled by players exhibited significant variability, reflecting the range of player skill and experience. This variability contributed to the game's enduring appeal, as players constantly sought to improve their performance and achieve higher scores. The distribution of high scores in Flappy Bird displayed a pronounced positive skew, indicating that a minority of players achieved exceptionally high scores while the majority faced repeated challenges. This skewness underscored the game's steep learning curve and the dedication required to master it. Moreover, the diverse tapping patterns adopted by players highlighted the absence of a one-size-fits-all strategy for success. The game's open-ended nature allowed players to experiment with different approaches, adding depth and replayability. Flappy Bird's remarkable ability to engage players, with an average session duration of 12 minutes, speaks to its addictive quality. Its minimalist design and challenging gameplay made it a phenomenon in mobile gaming. In the annals of gaming history, Flappy Bird stands as a testament to the power of simplicity and challenge, captivating players worldwide and leaving an indelible mark on the mobile gaming landscape. Despite its controversial removal from app stores, its influence endures, reminding us of the enduring allure of well-designed, challenging games.

Acknowledgment: The support of all my co-authors is highly appreciated.

Data Availability Statement: The research contains data related to web traffic and associated metrics. The data consists of views and dates as parameters.

Funding Statement: No funding has been obtained to help prepare this manuscript and research work.

Conflicts of Interest Statement: No conflicts of interest have been declared by the author(s). Citations and references are mentioned as per the used information.

Ethics and Consent Statement: The consent was obtained from the organization and individual participants during data collection, and ethical approval and participant consent were received.

References

1. A. W. Kosner, “Flappy bird and the power of simplicity scaled,” Forbes, 03-Feb-2014. [Online]. Available: https://www.forbes.com/sites/anthonykosner/2014/02/03/flappy-bird-and-the-power-of-simplicity-scaled/?sh=236da0117339. [Accessed: 01-Nov.-2023].
2. Wikipedia contributors, “Flappy Bird,” Wikipedia, The Free Encyclopedia, 14-Nov-2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Flappy_Bird&oldid=1189909778.
3. L. A. Nguyen, “Exclusive: Flappy Bird creator dong Nguyen says app ‘gone forever’ because it was ‘an addictive product,’” Forbes, 11-Feb-2014. [Online]. Available: https://www.forbes.com/sites/lananhnguyen/2014/02/11/exclusive-flappy-bird-creator-dong-nguyen-says-app-gone-forever-because-it-was-an-addictive-product/?sh=6f3fd5486476. [Accessed: 07-Nov-2023].
4. open-source-ios-apps: :iphone: Collaborative List of Open-Source iOS Apps. [Accessed: 17-Nov-2023].
5. “Recently Active ‘game-development’ Questions - Page 68,” Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/tagged/game-development?tab=active&page=68. [Accessed: 17-Nov-2023].
6. Zigurous, “How to make Flappy Bird in Unity (complete tutorial),” 10-Nov-2021. [Online]. Available: https://www.youtube.com/watch?v=ihvBiJ1oC9U. [Accessed: 12-Nov-2023].
7. Game Maker’s Toolkit, “The unity tutorial for complete beginners,” 02-Dec-2022. [Online]. Available: https://www.youtube.com/watch?v=XtQMytORBmM. [Accessed: 17-Nov-2023].