Flappy Bird Game

Project Documentation

Philipp Borkovic, 5BHITM
Gustav Grillitsch, 5BHITM

Department of IT
HTBLuVA Villach

Version 1.0
January 20, 2026

Abstract

This document presents a comprehensive implementation of an enhanced Flappy Bird game developed in Godot using C#. The game builds upon the classic mechanics while introducing advanced features and multiple game modes to provide an engaging player experience.

Core Gameplay Features

The game implements dynamic difficulty scaling where pipes gradually converge as the player's score increases, creating progressively challenging gameplay. Additionally, the game speed increases every 10 points, requiring players to adapt their reflexes and timing continuously. Colored pipes introduce strategic elements by offering bonus points, encouraging players to take calculated risks.

Visual Enhancements

The game features a dynamic day-to-night cycle that transitions from day through sunset to night, providing visual variety throughout extended play sessions. Weather effects including rain, snow, and fog add atmospheric depth and additional visual complexity to the gameplay environment.

Power-ups and Collectibles

Three distinct power-ups enhance gameplay strategy:

- **Shield**: Provides protection from a single collision, allowing recovery from mistakes
- **Slow-motion**: Grants 2 seconds of reduced game speed for precise maneuvering
- **Magnet**: Automatically collects coins within range, maximizing score potential

Stars serve as achievement markers, providing goals and milestones for player progression.

Game Modes

The implementation includes three distinct game modes:

- **Endless Mode**: Classic Flappy Bird gameplay with progressive difficulty
- **Time Attack**: Challenge players to achieve maximum score within 60 seconds
- **Survival Mode**: Features progressively harder pipe configurations, testing player skill limits

This enhanced implementation demonstrates modern game design principles while maintaining the accessibility and addictive nature of the original Flappy Bird concept.

Contents

Chapter 1

Game Entities

1.1 Bird

The player-controlled entity implemented as a `CharacterBody2D`.

1.1.1 Physics Constants

Parameter	Value	Description
Gravity	980.0	Downward acceleration (px/s^2)
JumpVelocity	-350.0	Upward impulse on input
MaxRotationDown	$\pi/2$	Maximum downward tilt (90°)
MaxRotationUp	$-\pi/6$	Maximum upward tilt (-30°)
RotationSpeed	0.003	Rotation interpolation factor

Table 1.1: Bird physics parameters

1.1.2 Input Handling

The bird responds to:

- `ui_accept` action
- Spacebar (`Key.Space`)
- Left mouse button (`MouseButton.Left`)

1.1.3 State Management

```
public void Kill()     // Sets _isAlive = false
public void Reset()    // Restores initial state
public bool IsAlive() // State getter
```

1.2 Pipe

Obstacle entity implemented as `Area2D` with collision detection.

1.2.1 Configuration

Parameter	Value	Description
ScrollSpeed	150.0 (static)	Horizontal movement speed
DestroyPositionX	-200.0	X position for cleanup

Table 1.2: Pipe configuration

1.2.2 Signals

```
1 public delegate void PipePassedEventHandler();
```

Emitted when pipe passes the bird's X position (100px threshold).

1.2.3 Collision Handling

On `BodyEntered`: If body is `Bird`, calls `bird.Kill()`.

1.3 Ground

Static collision boundary implemented as `StaticBody2D`.

- Position: Y = 1030
- Collision shape: Rectangle 1920 × 100
- Visual: `TextureRect` with `borderBG.png`

Chapter 2

Database Architecture

2.1 Overview

SQLite database using `Microsoft.Data.Sqlite`. Database file stored at `OS.GetUserDataDir()/flappy`

2.2 Schema

2.2.1 GameStatistics Table

Singleton record (Id = 1) for aggregate statistics.

Column	Type	Description
Id	INTEGER	Primary key (autoincrement)
HighScore	INTEGER	All-time highest score
TotalGamesPlayed	INTEGER	Total game sessions
TotalDeaths	INTEGER	Total death count
TotalPipesPassed	INTEGER	Cumulative pipes passed
LastPlayedDate	TEXT	ISO 8601 timestamp
AverageScore	INTEGER	Calculated average score

Table 2.1: GameStatistics schema

2.2.2 GameSessions Table

Individual session records for history tracking.

Column	Type	Description
Id	INTEGER	Primary key (autoincrement)
Score	INTEGER	Session final score
PipesPassed	INTEGER	Pipes passed in session
PlayedDate	TEXT	ISO 8601 timestamp
SessionDuration	REAL	Duration in seconds

Table 2.2: GameSessions schema

2.3 Data Models

2.3.1 GameStatistics

```
public class GameStatistics
{
    public int Id { get; set; }
    public int HighScore { get; set; }
    public int TotalGamesPlayed { get; set; }
    public int TotalDeaths { get; set; }
    public int TotalPipesPassed { get; set; }
    public DateTime LastPlayedDate { get; set; }
    public int AverageScore { get; set; }
}
```

2.3.2 GameSession

```
public class GameSession
{
    public int Id { get; set; }
    public int Score { get; set; }
    public int PipesPassed { get; set; }
    public DateTime PlayedDate { get; set; }
    public double SessionDuration { get; set; }
}
```

2.4 DatabaseService API

Method	Description
`SaveGameSession(score, pipes, duration)`	Persists session and updates stats
`GetStatistics()`	Returns `GameStatistics` record
`GetRecentSessions(limit)`	Returns last N sessions
`ResetAllStatistics()`	Clears all data

Table 2.3: DatabaseService public methods

2.5 Transaction Handling

All write operations use transactions with rollback on exception:

```
using var transaction = connection.BeginTransaction();
try {
    // Insert/Update operations
    transaction.Commit();
} catch {
    transaction.Rollback();
```

```
7       throw;
8   }
```

Chapter 3

Game Logic

3.1 GameManager

Central controller for game state, scoring, and system coordination.

3.1.1 Game States

```
public enum GameState
{
    Menu,
    Playing,
    GameOver
}
```

3.1.2 State Transitions

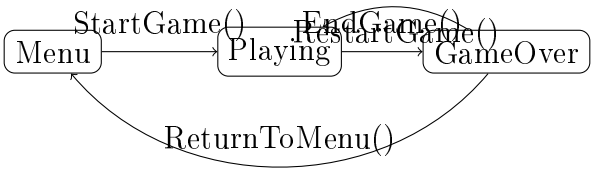

Figure 3.1: Game state machine

3.1.3 Signals

Signal	Payload
`GameStarted`	None
`GameOver`	`int finalScore`
`ScoreChanged`	`int newScore`

Table 3.1: GameManager signals

3.1.4 Core Methods

- `StartGame()`: Initializes game state, resets bird/difficulty, starts spawning
- `EndGame()`: Stops spawning, saves session to database, emits GameOver
- `RestartGame()`: Calls EndGame() then deferred StartGame()
- `OnPipePassed()`: Increments score, updates difficulty

3.2 SpawnManager

Handles pipe instantiation with configurable intervals.

3.2.1 Configuration

Parameter	Value	Description
SpawnInterval	3.0s	Default time between spawns
SpawnPositionX	2000	Spawn X coordinate (off-screen right)
MinGapY	400	Minimum gap Y position
MaxGapY	650	Maximum gap Y position

Table 3.2: SpawnManager configuration

3.2.2 Pipe Grouping

All spawned pipes are added to the `"pipes"` group for batch operations:

```
1 pipe.AddToGroup("pipes");
```

3.3 DifficultyScaler

Dynamic difficulty adjustment based on player score.

3.3.1 Scaling Parameters

Parameter	Base	Limit	Rate
ScrollSpeed	150	400 (max)	+3/point
SpawnInterval	3.0s	1.5s (min)	-0.02s/point

Table 3.3: Difficulty scaling parameters

3.3.2 Difficulty Level Calculation

```
public int GetDifficultyLevel()
{
    var speedProgress = (_currentScrollSpeed - BaseScrollSpeed)
                      / (MaxScrollSpeed - BaseScrollSpeed);
    return Mathf.Clamp((int)(speedProgress * 10) + 1, 1, 10);
}
```

Returns level 1-10 based on current scroll speed progression.

3.3.3 Integration Flow

1. Player passes pipe → `OnPipePassed()`
2. `_difficultyScaler.UpdateDifficulty(_score)`
3. `Pipe.ScrollSpeed = _difficultyScaler.GetScrollSpeed()`
4. `_spawnManager.SetSpawnInterval(...)`

Chapter 4

User Interface

4.1 Screen States

The UI manages three distinct visual states controlled by `UIController`.

Element	Menu	Playing	GameOver
StartButton	Visible	Hidden	Hidden
ScoreLabel	Hidden	Visible	Hidden
GameOverContainer	Hidden	Hidden	Visible

Table 4.1: UI element visibility per state

4.2 UI Hierarchy

```
UI (CanvasLayer)
+-- StartButton (Button)
+-- ScoreLabel (Label)
+-- GameOverContainer (Control)
    +-- Panel (ColorRect)
    +-- GameOverLabel (Label)
    +-- FinalScoreLabel (Label)
    +-- HighScoreLabel (Label)
    +-- RestartButton (Button)
```

4.3 UIController

4.3.1 Exported Node Paths

```
[Export] public NodePath GameManagerPath;
[Export] public NodePath StartButtonPath;
[Export] public NodePath ScoreLabelPath;
[Export] public NodePath GameOverContainerPath;
[Export] public NodePath FinalScoreLabelPath;
[Export] public NodePath HighScoreLabelPath;
```

```
7 [Export] public NodePath RestartButtonPath;
```

4.3.2 Signal Connections

Source	Signal	Handler
StartButton	Pressed	`OnStartButtonPressed()`
RestartButton	Pressed	`OnRestartButtonPressed()`
GameManager	GameStarted	`OnGameStarted()`
GameManager	GameOver	`OnGameOver(finalScore)`
GameManager	ScoreChanged	`OnScoreChanged(newScore)`

Table 4.2: UIController signal connections

4.4 Menu Screen

- Displays START GAME button centered at (810, 490)
- Button dimensions: 300 × 100
- Background: `backgroundSundown.png` (full screen)

4.5 Gameplay Screen

- Score label at top center (910, 50)
- Font size: 48pt
- Real-time score updates via `ScoreChanged` signal

4.6 Game Over Screen

- Semi-transparent panel (660, 340) to (1260, 740)
- Panel color: RGBA(0.2, 0.2, 0.2, 0.9)
- Displays:
  - "GAME OVER" header (48pt)
  - Final score (32pt)
  - High score from database (28pt)
  - RESTART button

4.7 Scene Integration

The UI is instanced in `main.tscn` with exported paths configured:

```
[node name="UI" parent="." instance=ExtResource("6_ui")]
GameManagerPath = NodePath("../GameManager")
StartButtonPath = NodePath("StartButton")
ScoreLabelPath = NodePath("ScoreLabel")
...
```

Chapter 5

Weather System

5.1 Overview

Dynamic weather and time-of-day system rendered via `CanvasLayer` (layer 10).

5.2 Enumerations

5.2.1 WeatherType

```
public enum WeatherType
{
    Clear,  // No weather effects
    Rain,   // Particle-based rain
    Fog     // Screen overlay
}
```

5.2.2 TimeOfDay

```
public enum TimeOfDay
{
    Day,    // No overlay
    Night   // Dark tint overlay
}
```

5.3 WeatherController

5.3.1 Configuration Constants

Parameter	Value	Description
WeatherChangeInterval	30.0s	Auto-change period
TransitionDuration	2.0s	Effect fade duration

Table 5.1: WeatherController configuration

5.3.2 Visual Effects

Effect	Implementation	Properties
Night	ColorRect overlay	Color(0, 0, 0.15, 0.6)
Rain	GpuParticles2D	300 particles, diagonal fall
Fog	ColorRect overlay	Color(0.85, 0.85, 0.9, 0.5)

Table 5.2: Weather visual effects

5.3.3 Public API

```
// Set specific weather/time
void SetWeather(WeatherType weather)
void SetTimeOfDay(TimeOfDay timeOfDay)

// Randomize both
void RandomizeWeather()

// Control auto-cycling
void SetAutoChangeWeather(bool enabled)

// Reset to Clear/Day
void ResetWeather()

// Getters
WeatherType GetCurrentWeather()
TimeOfDay GetCurrentTimeOfDay()
```

5.4 Scene Structure

```
WeatherSystem (CanvasLayer, layer=10)
+-- WeatherController (Node)
+-- NightOverlay (ColorRect)
|   +-- mouse_filter = IGNORE
|   +-- anchors = full screen
+-- FogOverlay (ColorRect)
|   +-- mouse_filter = IGNORE
|   +-- anchors = full screen
+-- RainParticles (GpuParticles2D)
    +-- amount = 300
    +-- position = (960, -50)
```

Property	Value
Direction	(0.2, 1, 0)
Spread	8.0°
Initial velocity	600-800
Gravity	(50, 400, 0)
Scale range	0.8-1.2
Blend mode	Additive

Table 5.3: Rain particle material settings

5.5 Rain Particle Configuration

5.6 Transition System

Effects use Godot's `Tween` for smooth transitions:

```
_currentTween?.Kill();
_currentTween = CreateTween();
_currentTween.TweenProperty(
    _nightOverlay,
    "color:a",
    0.6f,
    TransitionDuration
);
```

5.7 Input Handling

All overlay `ColorRect` nodes have `mouse_filter = 2` (IGNORE) to prevent blocking UI interactions.