Flappy Bird Game

Project Documentation

Philipp Borkovic, 5BHITM
Gustav Grillitsch, 5BHITM

Department of IT
HTBLuVA Villach

Version 1.0
January 26, 2026

Contents

Chapter 1

Design

1.1 Game Concept

The game is a recreation of the classic *Flappy Bird* mobile game, featuring **AV Eder** as the playable bird character. The player controls Eder by tapping or clicking to make him flap and stay airborne, while navigating through gaps between pipes. The goal is to survive as long as possible and achieve the highest score.

1.2 Art Style

The visual design follows a pixel art aesthetic, staying true to the original Flappy Bird's retro charm while incorporating custom character designs.

1.2.1 Design Tools

All pixel art assets were created using **Aseprite**, a professional pixel art and animation tool. Aseprite was chosen for its:

- Intuitive pixel-perfect drawing tools
- Layer management for complex sprites
- Animation timeline for sprite animations
- Export options optimized for game engines

1.3 Character Design

1.3.1 AV Eder (The Bird)

The main character, AV Eder, replaces the traditional Flappy Bird with a custom-designed pixel art representation. The character was designed to be:

- Recognizable and charming
- Appropriately sized for gameplay visibility
- Animated with smooth flapping motion

1.4 Environment Design

1.4.1 Pipes

The iconic green pipes serve as the main obstacles. They were designed with:

- Clear visual boundaries for fair collision detection
- Consistent sizing for predictable gameplay
- Classic styling with a modern pixel art touch

1.4.2 Background

The background features a sundown sky theme, creating a warm and inviting atmosphere for the game. The design includes gradient colors transitioning from warm oranges to cooler blues.

1.4.3 Coins

Collectible coins were added as bonus items throughout the game. They feature:

- Bright, eye-catching design
- Subtle bounce animation for visual feedback
- Clear visibility against the background

1.5 User Interface

The UI design prioritizes clarity and simplicity:

- **Start Menu:** Displays game title and player statistics
- **In-Game HUD:** Shows current score and coins collected
- **Game Over Screen:** Presents final score, high score, and restart option

All UI elements were designed with readable fonts and semi-transparent panels for better visibility against the game background.

Chapter 2

Game Entities

2.1 Bird

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

2.1.1 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 2.1: Bird physics parameters

2.1.2 Input Handling

The bird responds to:

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

2.1.3 State Management

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

2.2 Pipe

Obstacle entity implemented as `Area2D` with collision detection.

2.2.1 Configuration

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

Table 2.2: Pipe configuration

2.2.2 Signals

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

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

2.2.3 Collision Handling

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

2.3 Ground

Static collision boundary implemented as `StaticBody2D`.

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

Chapter 3

Database Architecture

3.1 Overview

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

3.2 Schema

3.2.1 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 3.1: GameStatistics schema

3.2.2 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 3.2: GameSessions schema

3.3 Data Models

3.3.1 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; }
}
```

3.3.2 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; }
}
```

3.4 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 3.3: DatabaseService public methods

3.5 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 4

Game Logic

4.1 GameManager

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

4.1.1 Game States

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

4.1.2 Signals

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

Table 4.1: GameManager signals

4.1.3 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

4.2 SpawnManager

Handles pipe instantiation with configurable intervals.

4.2.1 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 4.2: SpawnManager configuration

4.2.2 Pipe Grouping

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

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

4.3 DifficultyScaler

Dynamic difficulty adjustment based on player score.

4.3.1 Scaling Parameters

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

Table 4.3: Difficulty scaling parameters

4.3.2 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.

4.3.3 Integration Flow

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

Chapter 5

User Interface

5.1 UIController

5.1.1 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;
[Export] public NodePath RestartButtonPath;
```

5.1.2 Signal Connections

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

Table 5.1: UIController signal connections

5.2 Menu Screen

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

5.3 Gameplay Screen

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

5.4 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

5.5 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 6

Weather System

6.1 Overview

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

6.2 Enumerations

6.2.1 WeatherType

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

6.2.2 TimeOfDay

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

6.3 WeatherController

6.3.1 Configuration Constants

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

Table 6.1: WeatherController configuration

6.3.2 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 6.2: Weather visual effects

6.3.3 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()
```

6.4 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 6.3: Rain particle material settings

6.5 Rain Particle Configuration

6.6 Transition System

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

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

6.7 Input Handling

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

Chapter 7

Division of Work

7.1 Development

7.1.1 Programming

All C# code for this project was developed by **Philipp Borkovic** using the following tools:

- **IDE:** JetBrains Rider
- **Game Engine:** Godot Mono Engine (for development and testing)

7.1.2 Art & Assets

All game assets were created by **Gustav Grillitsch** using the following tools:

- **Pixel Art:** Aseprite
- **Scene Creation:** Godot Mono (for base screen creation)

7.2 Time Spent

Team Member	**Role**	**Hours**
Philipp Borkovic	Programming	30–40 hours
Gustav Grillitsch	Art & Assets	25–30 hours
Total		**55–70 hours**

Table 7.1: Time Investment by Team Member

7.3 Tools & Technologies

Category	**Tool**
Game Engine	Godot Mono 4.x
Programming Language	C#
IDE	JetBrains Rider
Pixel Art Editor	Aseprite
Version Control	Git

Table 7.2: Development Tools Used

Chapter 8

Resources

8.1 Programming Resources

The following resources were used during the development of this project:

8.1.1 C# Documentation

- **C# Language Guide**
  https://learn.microsoft.com/en-us/dotnet/csharp/
- **C# Events Programming Guide**
  https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
- **Understanding Events in C# with Practical Examples**
  https://medium.com/@ravipatel.it/understanding-events-in-c-with-practical-exampl

8.1.2 Godot Engine Documentation

- **Godot Engine Documentation (General)**
  https://docs.godotengine.org/en/stable/index.html
- **C# Basics in Godot (German)**
  https://docs.godotengine.org/de/4.x/tutorials/scripting/c_sharp/c_sharp_basics.html
- **C# in Godot Documentation**
  https://docs.godotengine.org/en/stable/tutorials/scripting/c_sharp/index.html