Documentación Técnica Completa - Game Aletheia Cross

Tabla de Contenidos

Descripción General

Game Aletheia Cross es un juego educativo de plataformas 2D desarrollado en **C#** con **Avalonia UI** y **.NET 9.0**. Combina mecánicas de plataformeo con desafíos de programación en **Java**, donde los jugadores deben resolver puzzles algorítmicos para avanzar a través de diferentes niveles mientras interactúan con NPCs y toman decisiones que afectan su progresión.

Características Principales

- **Plataformeo 2D**: Física realista con gravedad, saltos y colisiones
- **Puzzles de Programación**: Desafíos de código Java con compilación y ejecución en tiempo real
- **Sistema de Facciones**: Tres facciones con filosofías distintas
  - Gobierno Aletheia
  - Redline Corporation
  - Resistencia - Biblioteca Libre
- **NPCs Interactivos**: Personajes con diálogos que enriquecen la narrativa
- **Progresión de Niveles**: 7 niveles con dificultad creciente
- **Sistema de Ranking**: Tabla de líderes basada en puntuación
- **Tutorial Interactivo**: Guía inicial para aprender los controles

---

□ Arquitectura del Sistema

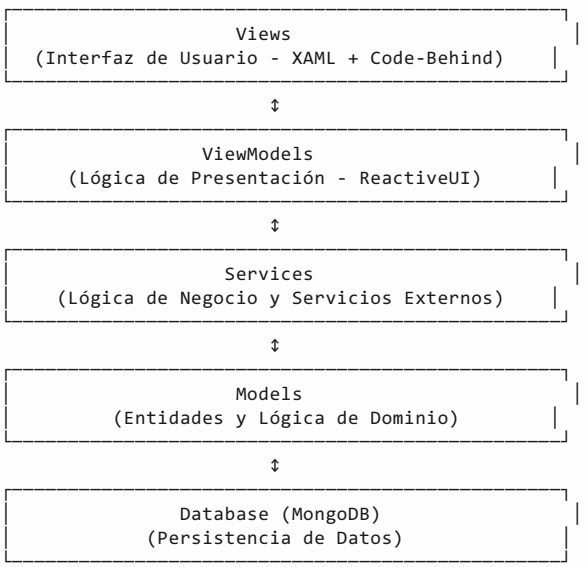
El proyecto sigue el patrón **MVVM (Model-View-ViewModel)** con las siguientes capas:

Diagramas UML

- Diagrama de Actividades
- Diagrama de Casos de Uso

- Diagrama de Clases
- Diagrama de Componentes
- Diagrama de Comunicación
- Diagrama de Despliegue
- Diagrama de Estados
- Diagrama de Objetos
- Diagrama de Paquetes
- Diagrama de Perfil
- Diagrama de Secuencia
- Diagrama de Tiempos

Diagrama de Capas



Flujo de Datos

1. Usuario interactúa con la **Vista** (View)
2. Vista notifica al **ViewModel** mediante bindings
3. ViewModel procesa la lógica y llama a **servicios**
4. Servicios ejecutan operaciones de negocio y acceden a **repositorios**
5. Repositorios realizan operaciones CRUD en **MongoDB**
6. Datos se propagan de vuelta a través de las capas usando **ReactiveUI**

□ Tecnologías Utilizadas

Framework y Lenguajes

Tecnología	Versión	Propósito
.NET	9.0	Framework base
C#	Latest	Lenguaje principal
Avalonia UI	11.3.8	Framework multiplataforma para UI
ReactiveUI	19.5.1	Implementación de MVVM reactivo

Base de Datos

Tecnología	Versión	Propósito
MongoDB	3.5.0	Base de datos NoSQL
MongoDB.Driver	-	Driver oficial para .NET

Librerías Adicionales

Librería	Versión	Propósito
SkiaSharp	3.119.1	Renderizado de gráficos 2D
Newtonsoft.Json	13.0.4	Serialización JSON
Splat	14.7.1	Inyección de dependencias ligera

Herramientas de Desarrollo

- **Visual Studio 2022**: IDE principal
- **JDK**: Requerido para compilar y ejecutar código Java de los puzzles

---

Estructura del Proyecto

```
GameAletheiaCross/
│
├── Assets/
│   └── Images/              # Sprites y fondos
│       ├── playerH.png
│       ├── playerM.png
│       ├── npc1.png - npc4.png
│       ├── platform1.png - platform7.png
│       ├── Portal.png
│       ├── *Piso.png
│       └── *Fondo.png
│
├── Data/
│   └── SeedData.cs          # Generación de puzzles
│
├── Models/
│   ├── Player.cs            # Modelo del jugador
│   ├── Level.cs             # Modelo de niveles
│   ├── Puzzle.cs            # Modelo de puzzles
│   ├── NPC.cs               # Modelo de NPCs
│   ├── Faction.cs           # Modelo de facciones
│   └── Decision.cs          # Modelo de decisiones
│
├── Services/
│   ├── Database/
│   │   ├── MongoDbService.cs
│   │   └── Repositories/
│   │       ├── PlayerRepository.cs
│   │       ├── LevelRepository.cs
│   │       ├── PuzzleRepository.cs
│   │       └── NPCRepository.cs (implícito)
│   ├── CollisionManager.cs
│   ├── PhysicsEngine.cs
│   ├── LevelManager.cs
│   ├── NPCInteractionManager.cs
│   ├── PuzzleService.cs
│   ├── JavaCompilerService.cs
│   ├── LevelGenerator.cs
│   └── LevelsWithNPCsService.cs
│
├── ViewModels/
│   ├── MainWindowViewModel.cs
│   ├── MainMenuViewModel.cs
│   ├── CharacterSelectViewModel.cs
│   ├── TutorialViewModel.cs
│   ├── FactionSelectViewModel.cs
│   ├── GameViewModel.cs
│   ├── DialogueViewModel.cs
│   ├── TerminalViewModel.cs
│   ├── LoadGameViewModel.cs
│   ├── RankingViewModel.cs
│   └── ViewModelBase.cs
│
├── Views/
│   ├── MainWindow.axaml[.cs]
│   ├── MainMenuView.axaml[.cs]
│   ├── CharacterSelectView.axaml[.cs]
│   ├── TutorialView.axaml[.cs]
│   ├── FactionSelectView.axaml[.cs]
│   ├── GameView.axaml[.cs]
│   ├── DialogueView.axaml[.cs]
│   ├── TerminalView.axaml[.cs]
│   ├── LoadGameView.axaml[.cs]
│   └── RankingView.axaml[.cs]
│
├── Utils/
│   └── BooleanNegationConverter.cs
│
├── App.axaml[.cs]           # Configuración de la aplicación
├── Program.cs               # Punto de entrada
└── GameAletheiaCross.csproj # Archivo de proyecto
```

Modelos de Datos

Player

Representa a un jugador en el sistema.

```
public class Player : ReactiveObject
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }  // "Hombre" o "Mujer"
    public string Avatar { get; set; }
    public int CurrentLevel { get; set; } = 1;
    public string Faction { get; set; }
    public int TotalScore { get; set; } = 0;
    public int Health { get; set; } = 100;
    public Position Position { get; set; }
    public Velocity Velocity { get; set; }
    public bool IsJumping { get; set; }
    public bool IsFacingRight { get; set; } = true;
    public DateTime CreationDate { get; set; }
    public DateTime LastPlayed { get; set; }
}
```

Propiedades Clave: - `CurrentLevel`: Nivel actual del jugador (1-7) - `Position` y `Velocity`: Coordenadas y movimiento en el espacio 2D - `IsJumping`: Estado de salto para control de física - `Faction`: Facción elegida por el jugador

Level

Define la estructura de un nivel del juego.

```
public class Level
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public int Difficulty { get; set; }
    public int OrderNumber { get; set; }
    public string Description { get; set; }
    public int TimeLimit { get; set; }
    public Floor FloorPlatform { get; set; }  // Piso principal
    public List<Platform> Platforms { get; set; }
    public List<Enemy> Enemies { get; set; }
    public List<string> NPCIds { get; set; }
    public string Background { get; set; }
    public List<NPC> NPCs { get; set; }  // [BsonIgnore]
}
```

Componentes:

Floor (Piso Principal)

```
public class Floor
{
    public float X { get; set; }
    public float Y { get; set; }
    public float Width { get; set; }
    public float Height { get; set; }
    public bool IsSolid { get; set; }
    public string FloorType { get; set; }
    // Tipos: Pasto, Hielo, Cristal, RedLine, PiedraTutorial
}
```

Platform (Plataforma)

```
public class Platform
{
    public float X { get; set; }
    public float Y { get; set; }
    public float Width { get; set; }
    public float Height { get; set; }
    public bool IsSolid { get; set; }
}
```

Puzzle

Representa un desafío de programación.

```
public class Puzzle
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public string Description { get; set; }
    public string ExpectedOutput { get; set; }
    public string StarterCode { get; set; }
    public string LevelId { get; set; }
    public int Difficulty { get; set; }
    public List<string> Hints { get; set; }
    public int Points { get; set; }
    public bool IsCompleted { get; set; }
}
```

Tipos de Puzzles: - Tutorial - Aritmética - Condicionales - Bucles - Recursión - Arrays - Algoritmos

NPC

Personaje no jugable con diálogos e interacciones.

```
public class NPC
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Role { get; set; }
    public string FactionId { get; set; }
    public float PositionX { get; set; }
    public float PositionY { get; set; }  // Coordenada base (pies del NPC
    public List<string> DialogueList { get; set; }
    public bool IsActive { get; set; }
    public string? LevelId { get; set; }
    public NPCStats Stats { get; set; }
}
```

NPCs Principales: - **El Archivero** (Gobierno): Líder del Gobierno Aletheia - **Decano Villanueva** (Redline): Líder de Redline Corporation - **Noa Espectra** (Resistencia): Líder de la Resistencia - **NPCs Secundarios**: Custodio Alfa, Reportero Fantasma, Analista Fractal, etc.

Faction

Define las facciones del juego.

```
public class Faction
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Type { get; set; }
    public string Leader { get; set; }
    public string Description { get; set; }
    public string ThemeColor { get; set; }
}
```

Facciones: 1. **Gobierno Aletheia**: Orden, control y preservación de datos 2. **Redline Corporation**: Corporativismo, eficiencia y lucro 3. **Resistencia - Biblioteca Libre**: Libertad de información y caos creativo

---

□ Servicios

MongoDbService

Servicio central para conexión con MongoDB.

```
public class MongoDbService
{
    private readonly IMongoDatabase _database;

    public MongoDbService(string connectionString, string dbName)
    {
        var client = new MongoClient(connectionString);
        _database = client.GetDatabase(dbName);
    }

    public IMongoCollection<T> GetCollection<T>(string name);
    public bool Ping();
}
```

PlayerRepository

Gestión de operaciones CRUD para jugadores.

Métodos principales: - `GetByIdAsync(string id)` - `GetByNameAsync(string name)` - `CreateAsync(Player player)` - `UpdateAsync(string id, Player player)` - `GetTopPlayersByScoreAsync(int limit)` - `UpdateFactionAsync(string id, string factionName)`

PhysicsEngine

Motor de física para el movimiento del jugador.

```
public class PhysicsEngine
{
    private const float GRAVITY = 0.8f;
    private const float MAX_FALL_SPEED = 20f;

    public void ApplyGravity(Player player)
    {
        player.Velocity.Y += GRAVITY;
        if (player.Velocity.Y > MAX_FALL_SPEED)
            player.Velocity.Y = MAX_FALL_SPEED;
    }

    public void UpdatePosition(Player player)
    {
        player.Position.X += player.Velocity.X;
        player.Position.Y += player.Velocity.Y;
    }
}
```

CollisionManager

Detección y manejo de colisiones.

```
public void CheckPlatformCollisions(
    Player player,
    List<Level.Platform> platforms,
    Level.Floor floor)
{
    bool onGround = false;

    // Verificar colisión con el piso principal
    if (floor != null && IsCollidingWithFloor(player, floor))
    {
        player.Position.Y = floor.Y;
        player.Velocity.Y = 0;
        player.IsJumping = false;
        onGround = true;
    }

    // Verificar colisiones con plataformas
    foreach (var platform in platforms)
    {
        if (IsCollidingWithPlatform(player, platform))
        {
            player.Position.Y = platform.Y;
            player.Velocity.Y = 0;
            player.IsJumping = false;
            onGround = true;
        }
    }
}
```

JavaCompilerService

Compilación y ejecución de código Java.

```
public async Task<CompilationResult> CompileAndRunAsync(
    string code,
    string expectedOutput)
{
    // 1. Extraer nombre de clase
    string className = ExtractClassName(code);

    // 2. Escribir archivo .java
    string sourceFile = Path.Combine(_tempDirectory, $"{className}.java");
    await File.WriteAllTextAsync(sourceFile, code);

    // 3. Compilar con javac
    var compileResult = await CompileAsync(sourceFile, className);

    // 4. Ejecutar con java
    var executeResult = await ExecuteAsync(className);

    // 5. Comparar salida
    return CompareOutput(executeResult.Output, expectedOutput);
}
```

NPCInteractionManager

Gestión de interacciones con NPCs.

```
public class NPCInteractionManager
{
    private const float INTERACTION_DISTANCE = 50f;

    public NPC? FindNearestNPC(Player player, List<NPC> npcs)
    {
        // Encuentra el NPC más cercano dentro del rango
    }

    public bool IsNearNPC(Player player, List<NPC> npcs)
    {
        return FindNearestNPC(player, npcs) != null;
    }
}
```

PuzzleService

Validación y gestión de puzzles.

```
public async Task<PuzzleValidationResult> ValidateSolutionAsync(
    string playerId,
    string puzzleId,
    string code,
    string output)
{
    var puzzle = await _puzzleRepo.GetByIdAsync(puzzleId);

    // Comparar salida esperada vs actual
    bool isValid = puzzle.ExpectedOutput.Trim() == output.Trim();

    if (isValid)
    {
        puzzle.IsCompleted = true;
        await _puzzleRepo.UpdateAsync(puzzleId, puzzle);

        var player = await _playerRepo.GetByIdAsync(playerId);
        player.TotalScore += puzzle.Points;
        await _playerRepo.UpdateAsync(playerId, player);
    }

    return new PuzzleValidationResult {
        IsValid = isValid,
        PointsEarned = puzzle.Points
    };
}
```

ViewModels

MainWindowViewModel

ViewModel raíz que maneja la navegación entre vistas.

```
public class MainWindowViewModel : ViewModelBase
{
    private ViewModelBase _currentView;

    public ViewModelBase CurrentView
    {
        get => _currentView;
        set => this.RaiseAndSetIfChanged(ref _currentView, value);
    }

    public void NavigateTo(ViewModelBase viewModel)
    {
        CurrentView = viewModel;
    }
}
```

GameViewModel

ViewModel principal del juego con lógica de gameplay.

Responsabilidades: - Game loop a 60 FPS - Manejo de input del jugador - Actualización de física - Detección de colisiones - Gestión de interacciones con NPCs - Control de diálogos - Verificación de completitud de puzzles - Progresión entre niveles

Constantes de Física:

```
private const float PLAYER_SPEED = 5f;
private const float JUMP_FORCE = -15f;
private const float EXIT_DETECTION_DISTANCE = 80f;
```

Game Loop:

```
private void GameLoop()
{
    if (_isPaused) return;

    UpdatePlayerMovement();
    _physics.ApplyGravity(Player);
    _physics.UpdatePosition(Player);
    _collision.CheckPlatformCollisions(
        Player,
        CurrentLevel.Platforms,
        CurrentLevel.FloorPlatform
    );

    CheckNearbyInteractions();
    CheckLevelExit();

    this.RaisePropertyChanged(nameof(Player));
}
```

TutorialViewModel

ViewModel para el tutorial interactivo con seguimiento de progreso.

```
public class TutorialViewModel : ViewModelBase
{
    private int _currentStep = 0;
    private bool _hasMovedLeft = false;
    private bool _hasMovedRight = false;
    private bool _hasJumped = false;
    private bool _hasReachedEnd = false;

    private void CheckTutorialProgress()
    {
        if (_currentStep == 0 && _hasMovedLeft && _hasMovedRight)
        {
            _currentStep = 1;
            UpdateInstructions();
        }

        if (_currentStep == 1 && _hasJumped)
        {
            _currentStep = 2;
            UpdateInstructions();
        }

        if (_currentStep == 2 && Player.Position.X > 1000)
        {
            _hasReachedEnd = true;
            this.RaisePropertyChanged(nameof(CanContinue));
        }
    }
}
```

TerminalViewModel

Gestión de la terminal de programación.

Funcionalidades: - Editor de código con sintaxis Java - Compilación y ejecución de código - Sistema de pistas progresivas - Validación de soluciones - Reinicio de código

```
private async void OnCompile()
{
    var compilationResult = await _compiler.CompileAndRunAsync(
        Code,
        CurrentPuzzle.ExpectedOutput
    );

    if (compilationResult.Success)
    {
        var validationResult = await _puzzleService.ValidateSolutionAsync(
            _playerId,
            CurrentPuzzle.Id,
            Code,
            compilationResult.Output
        );

        if (validationResult.IsValid)
        {
            await _gameViewModel.OnPuzzleCompletedAsync(CurrentPuzzle.Id);
            await Task.Delay(3000);
            OnClose();
        }
    }
}
```

DialogueViewModel

Gestión de diálogos con NPCs con temporizador automático.

```
public class DialogueViewModel : ViewModelBase
{
    private int _currentLineIndex = 0;
    private int _timeRemaining = 5;

    private void StartTimer()
    {
        Task.Run(async () =>
        {
            for (int i = 5; i >= 0; i--)
            {
                await Dispatcher.UIThread.InvokeAsync(() =>
                {
                    TimeRemaining = i;
                });

                if (i == 0)
                {
                    await Dispatcher.UIThread.InvokeAsync(() =>
                    {
                        OnContinue();
                    });
                    return;
                }

                await Task.Delay(1000);
            }
        });
    }
}
```

□ Vistas

Estructura de Vistas

Cada vista está compuesta por: - **Archivo XAML**: Define la interfaz visual - **Code-Behind**

(.cs): Lógica adicional y manejo de eventos

GameView

Vista principal del juego con renderizado de sprites.

Capas de Renderizado: 1. **Fondo**: Imagen de fondo según el nivel 2. **Piso**: Sprite del piso principal 3. **Plataformas**: Sprites de plataformas 4. **NPCs**: Sprites de NPCs con nombres 5. **Jugador**: Sprite del jugador con flip horizontal 6. **Portal**: Sprite del portal de salida 7. **UI Overlay**: Información del juego, controles, botones

Renderizado de Sprites:

```
private void RenderLevel()
{
    _gameCanvas.Children.Clear();

    // 1. Fondo
    if (_backgroundBitmaps.TryGetValue(level.Background, out var bg))
    {
        var bgImg = new Image {
            Source = bg,
            Width = 1280,
            Height = 720
        };
        Canvas.SetLeft(bgImg, 0);
        Canvas.SetTop(bgImg, 0);
        _gameCanvas.Children.Add(bgImg);
    }

    // 2. Piso, Plataformas, NPCs, Jugador, Portal...
}
```

Manejo de Input:

```
private void OnKeyDown(object? sender, KeyEventArgs e)
{
    switch (e.Key)
    {
        case Key.A:
        case Key.Left:
            vm.KeyLeft = true;
            break;
        case Key.D:
        case Key.Right:
            vm.KeyRight = true;
            break;
        case Key.W:
        case Key.Up:
            vm.KeyUp = true;
            break;
        case Key.E:
            vm.KeyE = true;
            break;
        case Key.T:
            vm.OnToggleTerminal();
            break;
        case Key.Escape:
            vm.OnPauseRequested();
            break;
    }
}
```

TerminalView

Editor de código con consola de salida.

Layout: - **Editor de Código**: 60% del ancho - **Consola de Salida**: 40% del ancho - **Botones**: COMPILAR, PISTA, RESET, CERRAR

DialogueView

Vista de diálogo superpuesta en la parte superior de la pantalla con temporizador visible.

Flujo de Juego

Flujo Completo

```
1. Inicio de Aplicación
   ↓
2. Menú Principal
   ↓
3. Nueva Partida → Creación de Personaje
   ↓
4. Tutorial Interactivo
   ↓
5. Selección de Facción
   ↓
6. Juego Principal (Niveles 1-7)
   │
   ├─→ Exploración de nivel
   ├─→ Interacción con NPCs
   ├─→ Resolución de puzzles (Terminal)
   ├─→ Llegada al portal
   └─→ Avance al siguiente nivel
   ↓
7. Completar todos los niveles
   ↓
8. Ranking Final
```

Ciclo de un Nivel

```
Inicio de Nivel
   ↓
Spawn del Jugador (X=100, Y=400)
   ↓
Game Loop (60 FPS)
   │
   ├─ Procesar Input
   ├─ Actualizar Física
   ├─ Detectar Colisiones
   ├─ Verificar Interacciones
   └─ Renderizar
   ↓
¿Puzzle requerido?
   ├─ Sí → Abrir Terminal → Resolver → Marcar completado
   └─ No → Continuar
   ↓
Llegar al Portal
   ↓
Verificar Puzzles Completados
   ↓
Avanzar al Siguiente Nivel
```

Progresión de Niveles

Condiciones para avanzar: - Llegar al portal (distancia < 80 píxeles) - Tener todos los puzzles del nivel completados (si existen) - Estar en el piso (no en el aire)

Al completar un nivel:

```
private async Task OnLevelComplete()
{
    PauseGameLoop();

    // 1. Otorgar puntos
    int levelPoints = 100 * CurrentLevel.Difficulty;
    Player.TotalScore += levelPoints;
    await playerRepo.UpdateAsync(_playerId, Player);

    // 2. Avanzar nivel
    bool advanced = await _levelManager.AdvanceToNextLevelAsync(_playerId)

    if (advanced)
    {
        // 3. Recargar jugador y nivel
        Player = await playerRepo.GetByIdAsync(_playerId);
        await LoadCurrentLevel();
        ResumeGameLoop();
    }
    else
    {
        // Juego completado
        _navigate(new MainMenuViewModel(_navigate));
    }
}
```

Sistema de Niveles

Configuración de Niveles

Los 7 niveles del juego se generan automáticamente al inicio mediante `LevelGenerator`.

Nivel	Nombre	Dificultad	Fondo	Piso	Plataformas	NPCs
1	El Despertar Digital	1	Forest	Pasto	6	Ninguno
2	Ruinas del Firewall Antiguo	1	Ruins	RedLine	9	Custodio Alfa
3	Ciudad de las Contraseñas Perdidas	2	City	Hielo	12	Reportero Fantasma, Bibliotecario Errante
4	Laberinto de Algoritmos	2	Digital	Cristal	13	Decano Villanueva
5	Santuario de los Datos Sagrados	3	Temple	Hielo	10	IA Centinela R-07, Noa Espectra
6	Torre Corporativa Redline	3	Cyber	RedLine	12	Analista Fractal, Guardián de Memoria
7	El Archivo Prohibido	4	Archive	Cristal	16	El Archivero

Generación de Plataformas

Las plataformas se generan mediante el método `GeneratePlatformsForLevel()`:

```
private List<Level.Platform> GeneratePlatformsForLevel(int levelNumber)
{
    var platforms = new List<Level.Platform>();

    if (levelNumber == 1)
    {
        // Progresión lineal ascendente
        platforms.Add(new Platform {
            X = 150, Y = 520, Width = 120, Height = 20
        });
        platforms.Add(new Platform {
            X = 320, Y = 470, Width = 140, Height = 20
        });
        // ... más plataformas
    }
    else if (levelNumber == 3)
    {
        // Patrón zigzag complejo
        platforms.Add(new Platform {
            X = 40, Y = 520, Width = 100, Height = 20
        });
        // ... patrón irregular
    }

    return platforms;
}
```

Posicionamiento del Portal

El portal se posiciona dinámicamente en la plataforma más alta:

```
// Buscar la plataforma más alta (menor Y)
var highestPlatform = level.Platforms[0];
foreach (var platform in level.Platforms)
{
    if (platform.Y < highestPlatform.Y)
        highestPlatform = platform;
}

// Portal en el centro de la plataforma más alta
float portalX = highestPlatform.X + (highestPlatform.Width / 2) - 25;
float portalY = highestPlatform.Y - 100;
```

Sistema de Puzzles

Categorías de Puzzles

Los puzzles están organizados por tipo y dificultad:

Tipo	Niveles	Dificultad	Conceptos
Tutorial	1-2	1	System.out.println(), variables básicas
Aritmética	2	1	Operadores matemáticos, tipos de datos
Condicionales	3	2	if-else, operadores lógicos
Bucles	4	2	for, while, acumuladores
Recursión	5	3	Funciones recursivas, casos base
Arrays	6	3	Arreglos, búsqueda, manipulación
Algoritmos	7	4-5	Fibonacci, números primos, ordenamiento

Estructura de un Puzzle

```
new Puzzle
{
    LevelId = "nivel_id",
    Name = "Bienvenida Digital",
    Type = "Tutorial",
    Description = "Tu primera misión: Imprime 'Bienvenido a Aletheia'",
    ExpectedOutput = "Bienvenido a Aletheia",
    StarterCode = @"public class Main {
    public static void main(String[] args) {
        // Escribe tu código aquí

    }
}",
    Difficulty = 1,
    Hints = new List<string>
    {
        "Usa System.out.println() para imprimir en consola",
        "El texto debe ir entre comillas dobles",
        "No olvides el punto y coma (;) al final"
    },
    Points = 50
}
```

Validación de Soluciones

El proceso de validación sigue estos pasos:

1. Extracción del nombre de clase

```
private string ExtractClassName(string code)
{
    var match = Regex.Match(code, @"public\s+class\s+(\w+)");
    return match.Success ? match.Groups[1].Value : null;
}
```

2. Compilación

```
var startInfo = new ProcessStartInfo
{
    FileName = "javac",
    Arguments = $"\"{sourceFile}\"",
    WorkingDirectory = _tempDirectory,
    RedirectStandardOutput = true,
    RedirectStandardError = true,
    UseShellExecute = false,
    CreateNoWindow = true
};
```

3. Ejecución

```
var startInfo = new ProcessStartInfo
{
    FileName = "java",
    Arguments = className,
    WorkingDirectory = _tempDirectory,
    RedirectStandardOutput = true,
    RedirectStandardError = true
};
```

4. Comparación de salida

```
private bool CompareOutput(string actual, string expected)
{
    actual = actual?.Trim().Replace("\r\n", "\n") ?? "";
    expected = expected?.Trim().Replace("\r\n", "\n") ?? "";
    return actual.Equals(expected, StringComparison.Ordinal);
}
```

Sistema de Pistas

Las pistas se revelan progresivamente:

```
private void OnShowHint()
{
    var hint = _puzzleService.GetNextHint(CurrentPuzzle, _currentHintIndex

    if (_currentHintIndex < TotalHints)
    {
        _currentHintIndex++;
        Output = $" PISTA {_currentHintIndex}/{TotalHints}\n\n{hint}";
    }
    else
    {
        Output = " NO HAY MÁS PISTAS";
    }
}
```

Ejemplos de Puzzles

Puzzle Simple: Bienvenida Digital

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Bienvenido a Aletheia");
    }
}
```

Puzzle Intermedio: Detector de Pares

```
public class Main {
    public static void main(String[] args) {
        int numero = 8;
        if (numero % 2 == 0) {
            System.out.println("PAR");
        } else {
            System.out.println("IMPAR");
        }
    }
}
```

Puzzle Avanzado: Factorial Recursivo

```
public class Main {
    public static int factorial(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return n * factorial(n - 1);
    }

    public static void main(String[] args) {
        System.out.println(factorial(5));  // Output: 120
    }
}
```

Sistema de NPCs

Distribución de NPCs por Nivel

Nivel	NPCs	Facción
1	Ninguno	-
2	Custodio Alfa	Gobierno
3	Reportero Fantasma, Bibliotecario Errante	Gobierno, Resistencia
4	Decano Villanueva	Redline
5	IA Centinela R-07, Noa Espectra	Redline, Resistencia
6	Analista Fractal, Guardián de Memoria	Redline, Resistencia
7	El Archivero	Gobierno

Posicionamiento de NPCs

Los NPCs se posicionan en plataformas específicas del nivel:

```
new NPC
{
    Id = "671000000000000000000102",
    Name = "Custodio Alfa",
    Role = "Agente del Archivo",
    FactionId = "671000000000000000000001",
    PositionX = 200,   // Centro de la plataforma
    PositionY = 520,   // Y de la plataforma (base del NPC)
    DialogueList = new List<string>
    {
        "Acceso restringido. Tu presencia es una anomalía.",
        "Archivar es purificar."
    },
    Stats = new NPCStats {
        Strength = 10,
        Defense = 9,
        Intelligence = 12,
        Agility = 8
    }
}
```

> **Nota importante**: `PositionY` representa la base del NPC (sus pies). El sprite se renderiza 32 píxeles arriba.

Interacción con NPCs

La interacción se activa cuando el jugador está cerca (< 50 píxeles):

```
private void CheckNearbyInteractions()
{
    var nearestNPC = _npcManager.FindNearestNPC(Player, CurrentLevel.NPCs)

    if (nearestNPC != null)
    {
        InteractionHint = $" Presiona E para hablar con {nearestNPC.Name}"
    }
    else
    {
        InteractionHint = "";
    }
}

private void CheckInteractions()
{
    var nearestNPC = _npcManager.FindNearestNPC(Player, CurrentLevel.NPCs)

    if (nearestNPC != null)
    {
        ActiveDialogue = new DialogueViewModel(_navigate, nearestNPC, this
        IsDialogueActive = true;
    }
}
```

Sistema de Diálogos

Los diálogos tienen las siguientes características:

- **Múltiples líneas**: Los NPCs pueden tener varios mensajes
- **Temporizador automático**: 5 segundos por mensaje
- **Avance manual**: Presionar E para continuar
- **Pausa del juego**: El gameplay se detiene durante el diálogo

```
public class DialogueViewModel : ViewModelBase
{
    private int _currentLineIndex = 0;
    private int _timeRemaining = 5;

    private void OnContinue()
    {
        if (_currentLineIndex < _npc.DialogueList.Count - 1)
        {
            _currentLineIndex++;
            this.RaisePropertyChanged(nameof(DialogueText));
            StartTimer();  // Reiniciar temporizador
        }
        else
        {
            _game.CloseDialogue();  // Cerrar al terminar
        }
    }
}
```

Mapeo de Sprites de NPCs

Cada NPC tiene un sprite específico basado en su nombre:

```
private Bitmap? GetNPCSpriteByName(string npcName)
{
    string cleanName = npcName.Split('—')[0].Trim().ToLower();

    if (cleanName.Contains("custodio") && cleanName.Contains("alfa"))
        return LoadBitmap("custodioalfa.png");
    else if (cleanName.Contains("reportero"))
        return LoadBitmap("reporterofantasma.png");
    else if (cleanName.Contains("bibliotecario"))
        return LoadBitmap("Bibliotecarioerrante.png");
    else if (cleanName.Contains("decano"))
        return _npcBitmaps[1];  // npc2.png
    else if (cleanName.Contains("noa"))
        return _npcBitmaps[2];  // npc3.png
    else if (cleanName.Contains("archivero"))
        return _npcBitmaps[3];  // npc4.png

    return _npcBitmaps[0];  // Sprite genérico
}
```

Físicas del Juego

Constantes de Física

```
// Movimiento
private const float PLAYER_SPEED = 5f;

// Salto
private const float JUMP_FORCE = -15f;

// Gravedad
private const float GRAVITY = 0.8f;
private const float MAX_FALL_SPEED = 20f;
```

Sistema de Gravedad

```
public void ApplyGravity(Player player)
{
    player.Velocity.Y += GRAVITY;

    // Limitar velocidad de caída
    if (player.Velocity.Y > MAX_FALL_SPEED)
        player.Velocity.Y = MAX_FALL_SPEED;
}
```

Sistema de Movimiento

```
private void UpdatePlayerMovement()
{
    // Movimiento horizontal
    if (KeyLeft)
    {
        Player.Velocity.X = -PLAYER_SPEED;
        Player.IsFacingRight = false;
    }
    else if (KeyRight)
    {
        Player.Velocity.X = PLAYER_SPEED;
        Player.IsFacingRight = true;
    }
    else
    {
        Player.Velocity.X = 0;  // Detener si no hay input
    }

    // Salto (solo si está en el suelo)
    if (KeyUp && !Player.IsJumping)
    {
        Player.Velocity.Y = JUMP_FORCE;
        Player.IsJumping = true;
    }
}
```

Sistema de Colisiones

Detección de Colisiones con Plataformas

```
private bool IsCollidingWithPlatform(Player player, Level.Platform platfor
{
    float playerLeft = player.Position.X - 20;
    float playerRight = player.Position.X + 20;
    float playerTop = player.Position.Y - 60;
    float playerBottom = player.Position.Y;

    float platformLeft = platform.X;
    float platformRight = platform.X + platform.Width;
    float platformTop = platform.Y;
    float platformBottom = platform.Y + platform.Height;

    return playerRight > platformLeft &&
           playerLeft < platformRight &&
           playerBottom > platformTop &&
           playerTop < platformBottom;
}
```

Resolución de Colisiones

```
public void CheckPlatformCollisions(
    Player player,
    List<Level.Platform> platforms,
    Level.Floor floor)
{
    bool onGround = false;

    // Verificar piso principal
    if (floor != null && floor.IsSolid &&
        IsCollidingWithFloor(player, floor))
    {
        if (player.Velocity.Y >= 0 &&
            player.Position.Y <= floor.Y + floor.Height)
        {
            player.Position.Y = floor.Y;
            player.Velocity.Y = 0;
            player.IsJumping = false;
            onGround = true;
        }
    }

    // Verificar plataformas
    foreach (var platform in platforms)
    {
        if (!platform.IsSolid) continue;

        if (IsCollidingWithPlatform(player, platform))
        {
            if (player.Velocity.Y >= 0 &&
                player.Position.Y <= platform.Y + platform.Height)
            {
                player.Position.Y = platform.Y;
                player.Velocity.Y = 0;
                player.IsJumping = false;
                onGround = true;
            }
        }
    }

    if (!onGround)
    {
        player.IsJumping = true;
    }
}
```

Límites del Mundo

```
// Límites horizontales
if (Player.Position.X < 0) Player.Position.X = 0;
if (Player.Position.X > 1240) Player.Position.X = 1240;

// Caída al vacío (respawn)
if (Player.Position.Y > 800)
{
    ResetPlayerPosition();
}
```

Ciclo de Física (Game Loop)

```
private void GameLoop()  // ~60 FPS
{
    if (_isPaused) return;

    // 1. Procesar input del jugador
    UpdatePlayerMovement();

    // 2. Aplicar física
    _physics.ApplyGravity(Player);
    _physics.UpdatePosition(Player);

    // 3. Detectar y resolver colisiones
    _collision.CheckPlatformCollisions(
        Player,
        CurrentLevel.Platforms,
        CurrentLevel.FloorPlatform
    );

    // 4. Aplicar límites del mundo
    if (Player.Position.X < 0) Player.Position.X = 0;
    if (Player.Position.X > 1240) Player.Position.X = 1240;

    // 5. Respawn si cae al vacío
    if (Player.Position.Y > 800)
    {
        ResetPlayerPosition();
    }

    // 6. Verificar interacciones y completitud
    CheckNearbyInteractions();
    CheckLevelExit();

    // 7. Notificar cambios a la vista
    this.RaisePropertyChanged(nameof(Player));
}
```

Renderizado Visual

Sistema de Sprites

El juego utiliza sprites PNG para todos los elementos visuales:

Sprites del Jugador

- `playerH.png`: Jugador masculino (32x32)
- `playerM.png`: Jugador femenino (32x32)

Sprites de NPCs

- `npc1.png`: NPCs genéricos/Gobierno
- `npc2.png`: Decano Villanueva
- `npc3.png`: Noa Espectra
- `npc4.png`: El Archivero
- `custodioalfa.png`
- `reporterofantasma.png`
- `Bibliotecarioerrante.png`
- `Guardiandelamemoria.png`
- `Iarobot.png`
- `Analistafractal.png`

Sprites de Plataformas

- `platform1.png` a `platform7.png`: Variaciones de plataformas

Sprites de Pisos

- `PastoPiso.png`: Piso de pasto
- `HieloPiso.png`: Piso de hielo
- `CristalPiso.png`: Piso de cristal
- `RedLinePiso.png`: Piso corporativo
- `PiedraTutorial.png`: Piso del tutorial

Fondos de Niveles

- `ForestFondo.png`: Nivel 1
- `RuinsFondo.png`: Nivel 2
- `CityFondo.png`: Nivel 3
- `CityDigitalFondo.png`: Nivel 4
- `TemploFondo.png`: Nivel 5
- `CyberFondo.png`: Nivel 6
- `ArchivoFondo.png`: Nivel 7

Fondos de Menús

- `InicioFondo.png`: Menú principal
- `DefaultFondo.png`: Fondos por defecto
- `GobiernoFondo.png`: Selección de Gobierno
- `RedlineFondo.png`: Selección de Redline
- `BibliotecaFondo.png`: Selección de Resistencia
- `RankedFondo.png`: Pantalla de ranking

Otros Sprites

- `Portal.png`: Portal de salida (50x100)

Carga de Sprites

```
private Bitmap? LoadBitmap(string filename)
{
    try
    {
        var uri = new Uri(
            $"avares://GameAletheiaCross/Assets/Images/{filename}"
        );
        var stream = AssetLoader.Open(uri);
        var bitmap = new Bitmap(stream);
        Console.WriteLine($" {filename} cargado correctamente");
        return bitmap;
    }
    catch (Exception ex)
    {
        Console.WriteLine($" No se pudo cargar {filename}: {ex.Message}");
        return null;
    }
}
```

Orden de Renderizado (Z-Index)

1. **Fondo del nivel** (z-index: 0)
2. **Piso principal** (z-index: 1)
3. **Plataformas** (z-index: 2)
4. **NPCs** (z-index: 3)
5. **Jugador** (z-index: 4)
6. **Portal** (z-index: 5)
7. **UI Overlay** (z-index: 10)
8. **Diálogos** (z-index: 20)

Renderizado del Jugador con Flip Horizontal

```
if (_playerBitmap != null)
{
    _playerImage = new Image
    {
        Source = _playerBitmap,
        Width = 32,
        Height = 32
    };

    Canvas.SetLeft(_playerImage, Player.Position.X - 16);
    Canvas.SetTop(_playerImage, Player.Position.Y - 32);

    // Flip horizontal si mira a la izquierda
    if (!Player.IsFacingRight)
    {
        _playerImage.RenderTransform = new ScaleTransform(-1, 1);
        _playerImage.RenderTransformOrigin = new RelativePoint(
            0.5, 0.5, RelativeUnit.Relative
        );
    }

    _gameCanvas.Children.Add(_playerImage);
}
```

Renderizado de NPCs con Nombres

```
// Sprite del NPC
var npcImg = new Image
{
    Source = npcSprite,
    Width = 32,
    Height = 32
};
Canvas.SetLeft(npcImg, npc.PositionX - 16);
Canvas.SetTop(npcImg, npc.PositionY - 32);
_gameCanvas.Children.Add(npcImg);

// Nombre del NPC
var nameText = new TextBlock
{
    Text = npc.Name.Split('—')[0].Trim(),
    FontSize = 10,
    Foreground = new SolidColorBrush(Colors.White),
    Background = new SolidColorBrush(Color.FromArgb(180, 0, 0, 0)),
    Padding = new Thickness(4, 2),
    TextAlignment = TextAlignment.Center
};
Canvas.SetLeft(nameText, npc.PositionX - 40);
Canvas.SetTop(nameText, npc.PositionY - 50);
_gameCanvas.Children.Add(nameText);
```

Fallback para Sprites Faltantes

Si un sprite no se carga, el sistema usa figuras geométricas:

```
// Fallback para jugador
var rect = new Avalonia.Controls.Shapes.Rectangle
{
    Fill = new SolidColorBrush(Color.Parse("#e94560")),
    Stroke = new SolidColorBrush(Color.Parse("#ff0080")),
    StrokeThickness = 2,
    Width = 40,
    Height = 60
};
Canvas.SetLeft(rect, Player.Position.X - 20);
Canvas.SetTop(rect, Player.Position.Y - 60);
_gameCanvas.Children.Add(rect);
```

Almacenamiento de Datos

MongoDB

El juego utiliza MongoDB como base de datos NoSQL.

Conexión:

```
var client = new MongoClient("mongodb://localhost:27017");
var database = client.GetDatabase("HackerFantasmaDB");
```

Colecciones

players

Almacena información de los jugadores.

```
{
  "_id": ObjectId("..."),
  "name": "JugadorX",
  "gender": "Hombre",
  "avatar": "default",
  "currentLevel": 3,
  "faction": "Gobierno Aletheia",
  "totalScore": 850,
  "health": 100,
  "position": { "x": 350.5, "y": 420.0 },
  "velocity": { "x": 0.0, "y": 0.0 },
  "isJumping": false,
  "isFacingRight": true,
  "creationDate": ISODate("2025-01-15T10:30:00Z"),
  "lastPlayed": ISODate("2025-01-16T14:20:00Z")
}
```

levels

Almacena la configuración de niveles.

```
{
  "_id": ObjectId("..."),
  "name": "El Despertar Digital",
  "type": "Plataformero",
  "difficulty": 1,
  "orderNumber": 1,
  "description": "Tu primera inmersión en la red",
  "timeLimit": 300,
  "floor": {
    "x": 0,
    "y": 600,
    "width": 1280,
    "height": 120,
    "isSolid": true,
    "floorType": "Pasto"
  },
  "platforms": [
    {
      "x": 150,
      "y": 520,
      "width": 120,
      "height": 20,
      "isSolid": true
    }
  ],
  "enemies": [],
  "npcIds": [],
  "background": "forest"
}
```

puzzles

Almacena los puzzles de programación.

```
{
  "_id": ObjectId("..."),
  "name": "Bienvenida Digital",
  "type": "Tutorial",
  "description": "Tu primera misión: Imprime 'Bienvenido a Aletheia'",
  "expectedOutput": "Bienvenido a Aletheia",
  "starterCode": "public class Main {...}",
  "levelId": ObjectId("..."),
  "difficulty": 1,
  "hints": [
    "Usa System.out.println() para imprimir en consola",
    "El texto debe ir entre comillas dobles"
  ],
  "points": 50,
  "isCompleted": false
}
```

npcs

Almacena los NPCs del juego.

```
{
  "_id": ObjectId("671000000000000000000102"),
  "Name": "Custodio Alfa",
  "Role": "Agente del Archivo",
  "FactionId": ObjectId("671000000000000000000001"),
  "PositionX": 200,
  "PositionY": 520,
  "Dialogue": [
    "Acceso restringido. Tu presencia es una anomalía.",
    "Archivar es purificar."
  ],
  "IsActive": true,
  "LevelId": null,
  "Stats": {
    "Strength": 10,
    "Defense": 9,
    "Intelligence": 12,
    "Agility": 8
  }
}
```

factions

Almacena las facciones del juego.

```
{
  "_id": ObjectId("671000000000000000000001"),
  "name": "Gobierno Aletheia",
  "type": "Gobierno",
  "leader": "El Archivero — Julián Casablancas",
  "description": "El Gobierno de Aletheia cree que la información...",
  "themeColor": "#0080FF"
}
```

Operaciones CRUD

Crear Jugador

```
var player = new Player
{
    Name = "NuevoJugador",
    Gender = "Hombre",
    CurrentLevel = 1
};

await _playerRepo.CreateAsync(player);
```

Actualizar Nivel del Jugador

```
player.CurrentLevel++;
player.LastPlayed = DateTime.UtcNow;
await _playerRepo.UpdateAsync(playerId, player);
```

Obtener Nivel Actual

```
var level = await _levelRepo.GetByOrderNumberAsync(player.CurrentLevel);
```

Marcar Puzzle como Completado

```
puzzle.IsCompleted = true;
await _puzzleRepo.UpdateAsync(puzzleId, puzzle);
```

Inicialización de Datos

Al iniciar la aplicación, se ejecuta el sistema de seed data:

```
// Program.cs
var dbService = new MongoDbService(
    "mongodb://localhost:27017",
    "HackerFantasmaDB"
);

// Verificar conexión
if (!dbService.Ping())
{
    Console.WriteLine(" ERROR: No se pudo conectar a MongoDB.");
    return;
}

// Generar niveles si no existen
var levelRepo = new LevelRepository(dbService);
var puzzleRepo = new PuzzleRepository(dbService);
var generator = new LevelGenerator(levelRepo, puzzleRepo);
await generator.GenerateDefaultLevelsAsync();

// Generar puzzles avanzados
var advancedSeed = new AdvancedSeedData(dbService);
await advancedSeed.SeedAdvancedPuzzlesAsync();
```

Instalación y Configuración

Requisitos Previos

Software Requerido

Software	Versión	Descarga
.NET SDK	9.0+	Descargar
MongoDB Community	6.0+	Descargar
JDK	17+	Descargar
Visual Studio 2022	Community+	Descargar

> **Importante**: JDK debe estar configurado en el PATH del sistema

Instalación Paso a Paso

1. Clonar el Repositorio

```
git clone https://github.com/tu-usuario/GameAletheiaCross.git
cd GameAletheiaCross
```

2. Configurar MongoDB

En Windows:

```
# Iniciar MongoDB como servicio
net start MongoDB

# O ejecutar manualmente
mongod --dbpath "C:\data\db"
```

En Linux/macOS:

```
# Iniciar MongoDB
sudo systemctl start mongod

# O ejecutar manualmente
mongod --dbpath /data/db
```

Verificar conexión:

```
mongosh
> show dbs
> exit
```

3. Configurar Variables de Entorno (Opcional)

Si MongoDB no está en el puerto por defecto (27017), editar `Program.cs`:

```
var dbService = new MongoDbService(
    "mongodb://localhost:27017",  // Cambiar puerto si es necesario
    "HackerFantasmaDB"
);
```

4. Restaurar Paquetes NuGet

```
dotnet restore
```

5. Verificar JDK

```
# Verificar que javac y java estén disponibles
javac -version
java -version
```

Si no están en el PATH, agregarlos:

Windows: - Panel de Control → Sistema → Configuración avanzada del sistema → Variables de entorno - Agregar a PATH: `C:\Program Files\Java\jdk-17\bin`

Linux/macOS:

```
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk
export PATH=$PATH:$JAVA_HOME/bin
```

6. Compilar el Proyecto

```
dotnet build
```

7. Ejecutar el Juego

```
dotnet run
```

Configuración de Assets

El proyecto incluye sprites en `Assets/Images/`. Si faltan sprites:

- Los sprites deben estar en formato PNG
- Deben colocarse en: `GameAletheiaCross/Assets/Images/`
- El sistema usa fallbacks geométricos si no encuentra sprites

Estructura esperada:

```
Assets/
└── Images/
    ├── playerH.png
    ├── playerM.png
    ├── npc1.png - npc4.png
    ├── platform1.png - platform7.png
    ├── Portal.png
    ├── PastoPiso.png
    ├── HieloPiso.png
    ├── CristalPiso.png
    ├── RedLinePiso.png
    ├── ForestFondo.png
    ├── RuinsFondo.png
    └── ... (otros fondos)
```

Configuración de Desarrollo

appsettings.json (Opcional)

Crear un archivo de configuración para desarrollo:

```
{
  "MongoDB": {
    "ConnectionString": "mongodb://localhost:27017",
    "DatabaseName": "HackerFantasmaDB"
  },
  "Java": {
    "CompilerTimeout": 10000,
    "ExecutionTimeout": 5000
  }
}
```

Configurar Hot Reload

En Visual Studio: 1. Habilitar “Hot Reload on File Save” 2. Editar `Properties/launchSettings.json`:

```
{
  "profiles": {
    "GameAletheiaCross": {
      "commandName": "Project",
      "hotReloadProfile": "aspnetcore"
    }
  }
}
```

Solución de Problemas Comunes

MongoDB no se conecta

Problema: Error: No se pudo conectar a MongoDB

Soluciones:

1. Verificar que MongoDB esté ejecutándose:

```
mongosh
```

2. Verificar puerto en `Program.cs`:

```
var dbService = new MongoDbService(
    "mongodb://localhost:27017",
    "HackerFantasmaDB"
);
```

3. Verificar firewall (permitir puerto 27017)

JDK no encontrado

Problema: Error ejecutando javac: No se pudo iniciar javac

Soluciones:

1. Verificar instalación:

```
javac -version
```

2. Agregar al PATH:
   - **Windows**: Variables de entorno → PATH → Agregar ruta de JDK
   - **Linux/macOS**: Editar `.bashrc` o `.zshrc`
3. Reiniciar el IDE después de cambiar PATH

Sprites no se cargan

Problema: El juego muestra figuras geométricas en lugar de sprites

Soluciones:

1. Verificar que los archivos PNG existen en `Assets/Images/`
2. Verificar nombres de archivo (case-sensitive en Linux/macOS):
   - `playerH.png`
   - `PlayerH.png`
3. Verificar que los archivos son recursos de Avalonia:

```
<ItemGroup>
  <AvaloniaResource Include="Assets\**\*" />
</ItemGroup>
```

4. Limpiar y recompilar:

```
dotnet clean
dotnet build
```

Errores de compilación de puzzles

Problema: Error de compilación: class Main is public, should be declared in a file named Main.java

Solución: Esto es normal. El sistema usa nombres de clase dinámicos. Asegurarse de que el código del puzzle tenga una clase pública.

Problema: Timeout compilando código

Solución: Aumentar timeout en `JavaCompilerService.cs`:

```
await process.WaitForExitAsync(TimeSpan.FromSeconds(30));
```

□ Arquitectura de Navegación

Sistema de Navegación

El juego usa un sistema de navegación basado en delegados:

```
public class MainWindowViewModel : ViewModelBase
{
    private ViewModelBase _currentView;

    public ViewModelBase CurrentView
    {
        get => _currentView;
        set => this.RaiseAndSetIfChanged(ref _currentView, value);
    }

    public void NavigateTo(ViewModelBase viewModel)
    {
        CurrentView = viewModel;
    }
}
```

Flujo de Navegación

```
MainWindow (MainWindowViewModel)
    ↓
    └─ CurrentView [ContentControl con DataTemplate]
         │
         ├─ MainMenuViewModel → MainMenuView
         │    └─ Opciones: Nueva Partida, Cargar, Ranking, Salir
         │
         ├─ CharacterSelectViewModel → CharacterSelectView
         │    └─ Crear jugador y navegar a Tutorial
         │
         ├─ TutorialViewModel → TutorialView
         │    └─ Completar tutorial y navegar a FactionSelect
         │
         ├─ FactionSelectViewModel → FactionSelectView
         │    └─ Elegir facción y navegar a Game
         │
         ├─ GameViewModel → GameView
         │    ├─ DialogueViewModel → DialogueView (Overlay)
         │    └─ TerminalViewModel → TerminalView (Overlay)
         │
         ├─ LoadGameViewModel → LoadGameView
         │    └─ Seleccionar jugador y navegar a Game
         │
         └─ RankingViewModel → RankingView
              └─ Ver tabla de líderes y volver a MainMenu
```

Navegación entre ViewModels

```
// Desde MainMenuViewModel hacia CharacterSelect
private void OnNewGame()
{
    _navigate(new CharacterSelectViewModel(_navigate));
}

// Desde CharacterSelect hacia Tutorial
private async void OnContinue()
{
    var player = new Player {
        Name = PlayerName,
        Gender = IsMale ? "Hombre" : "Mujer"
    };
    await playerRepo.CreateAsync(player);

    _navigate(new TutorialViewModel(_navigate, player.Id, player.Name));
}

// Desde Tutorial hacia FactionSelect
private async void OnContinue()
{
    _navigate(new FactionSelectViewModel(
        _navigate,
        _playerId,
        _playerName
    ));
}

// Desde FactionSelect hacia Game
private async void OnConfirmFaction()
{
    await playerRepo.UpdateFactionAsync(_playerId, SelectedFaction.Name);
    _navigate(new GameViewModel(_navigate, _playerId, _playerName));
}
```

Sistema de Puntuación

Fuentes de Puntos

1. Completar Puzzles

Según dificultad del puzzle:

Dificultad	Puntos
Tutorial	50
Fácil	75-100
Medio	100-150
Difícil	150-225
Muy Difícil	250-350

2. Completar Niveles

Según dificultad del nivel:

```
int levelPoints = 100 * CurrentLevel.Difficulty;
// Nivel 1 (Dif 1): 100 puntos
// Nivel 7 (Dif 4): 400 puntos
```

Cálculo de Puntuación Total

```
// Al completar un puzzle
private async Task<bool> CompletePuzzleAsync(
    string playerId,
    string puzzleId)
{
    var player = await _playerRepo.GetByIdAsync(playerId);
    var puzzle = await _puzzleRepo.GetByIdAsync(puzzleId);

    player.TotalScore += puzzle.Points;
    await _playerRepo.UpdateAsync(playerId, player);

    return true;
}

// Al completar un nivel
private async Task OnLevelComplete()
{
    int levelPoints = 100 * CurrentLevel.Difficulty;
    Player.TotalScore += levelPoints;
    await playerRepo.UpdateAsync(_playerId, Player);
}
```

Puntuación Máxima Posible

Concepto	Puntos
Puzzles Nivel 1	50
Puzzles Nivel 2	75
Puzzles Nivel 3	320
Puzzles Nivel 4	275
Puzzles Nivel 5	425
Puzzles Nivel 6	595
Puzzles Nivel 7	1,150
Completar Niveles 1-7	1,900
TOTAL	**~4,790 puntos**

Sistema de Ranking

```
public async Task<List<Player>> GetTopPlayersByScoreAsync(int limit = 10)
{
    return await _players.Find(_ => true)
        .SortByDescending(p => p.TotalScore)
        .Limit(limit)
        .ToListAsync();
}
```

Optimizaciones y Rendimiento

Game Loop Optimizado

```
// Game loop a ~60 FPS
Task.Run(async () =>
{
    while (_isRunning && !_gameLoopCts.Token.IsCancellationRequested)
    {
        GameLoop();
        await Task.Delay(16);  // ~60 FPS (1000ms / 60 ≈ 16ms)
    }
}, _gameLoopCts.Token);
```

Cache de Sprites

Los sprites se cargan una sola vez al inicio:

```
private void LoadImages()
{
    // Cargar todos los sprites al inicio
    _playerBitmap = LoadBitmap("playerH.png");
    _npcBitmaps.Add(LoadBitmap("npc1.png"));
    // ... más sprites

    _spritesLoaded = _playerBitmap != null;
}
```

Renderizado Selectivo

Solo se actualiza la posición del jugador en cada frame, no se re-renderiza todo el nivel:

```
private void UpdatePlayerPosition()
{
    if (_playerImage != null)
    {
        Canvas.SetLeft(_playerImage, _viewModel.Player.Position.X - 16);
        Canvas.SetTop(_playerImage, _viewModel.Player.Position.Y - 32);

        // Actualizar flip horizontal si es necesario
        if (_viewModel.Player.IsFacingRight)
            _playerImage.RenderTransform = null;
        else
            _playerImage.RenderTransform = new ScaleTransform(-1, 1);
    }
}
```

Limpieza de Recursos

```
public void Dispose()
{
    CleanupTempFiles();  // Limpiar archivos Java compilados
    _gameLoopCts?.Cancel();  // Cancelar game loop
}

private void CleanupTempFiles()
{
    try
    {
        if (Directory.Exists(_tempDirectory))
        {
            var files = Directory.GetFiles(_tempDirectory);
            foreach (var file in files)
            {
                File.Delete(file);
            }
            Directory.Delete(_tempDirectory, false);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($" Error limpiando archivos: {ex.Message}");
    }
}
```

Testing y Debugging

Logs del Sistema

El juego incluye logging extensivo para debugging:

```
// Logs de niveles
Console.WriteLine($" Nivel cargado: {level.Name}");
Console.WriteLine($"   Plataformas: {level.Platforms?.Count ?? 0}");
Console.WriteLine($"   NPCs: {level.NPCs?.Count ?? 0}");

// Logs de física
Console.WriteLine($" Jugador en ({Player.Position.X:F1}, {Player.Position.
Console.WriteLine($"   Velocidad: ({Player.Velocity.X:F1}, {Player.Velocit

// Logs de puzzles
Console.WriteLine($" Puzzle resuelto: {puzzle.Name} (+{puzzle.Points} pts)

// Logs de NPCs
Console.WriteLine($" Interacción con: {npc.Name}");
```

Comandos de Debugging

MongoDB:

```
// Ver todos los jugadores
use HackerFantasmaDB
db.players.find().pretty()

// Ver niveles
db.levels.find({}, {name: 1, orderNumber: 1, npcIds: 1})

// Ver puzzles de un nivel
db.puzzles.find({levelId: ObjectId("...")})

// Resetear progreso de un jugador
db.players.updateOne(
  {_id: ObjectId("...")},
  {$set: {currentLevel: 1, totalScore: 0}}
)
```

Consola del Juego:

Los logs aparecen en la consola de la aplicación:

```
 Conexión a MongoDB establecida
 7 niveles creados
 9 NPCs creados con posiciones ajustadas
 GameView inicializado
=== RENDERIZANDO NIVEL ===
 Nivel renderizado. Total: 45 elementos
```

Breakpoints Útiles

GameViewModel.cs:

- `GameLoop()`: Inspeccionar cada frame
- `CheckLevelExit()`: Verificar detección del portal
- `OnLevelComplete()`: Verificar progresión

JavaCompilerService.cs:

- `CompileAsync()`: Verificar compilación Java
- `ExecuteAsync()`: Verificar ejecución Java
- `CompareOutput()`: Verificar validación de salida

LevelRepository.cs:

- `GetByOrderNumberAsync()`: Verificar carga de niveles
- `LoadNPCsForLevel()`: Verificar carga de NPCs

Mejoras Futuras

Funcionalidades Planeadas

Sistema de Combate

- Implementar `Enemy` en niveles
- Mecánicas de ataque/defensa
- Sistema de daño y salud

Sistema de Decisiones

- Implementar modelo `Decision`
- Diálogos con opciones múltiples
- Consecuencias basadas en facción

Más Niveles

- Expandir a 10-15 niveles
- Mayor variedad de puzzles
- Desafíos opcionales

Sistema de Inventario

- Items coleccionables
- Power-ups temporales
- Objetos especiales por facción

Multijugador

- Modo cooperativo local
- Ranking online
- Desafíos compartidos

Más Lenguajes de Programación

- Python
- JavaScript
- C#

Editor de Niveles

- Crear niveles personalizados
- Compartir con la comunidad
- Workshop integrado

Música y Sonido

- Banda sonora dinámica
- Efectos de sonido
- Voces de NPCs (opcional)

Optimizaciones Técnicas

Entity Component System (ECS)

- Mejorar rendimiento con muchos NPCs
- Componentes reutilizables

- Mejor organización del código

Asset Pipeline

- Compresión de sprites
- Atlas de texturas
- Carga asíncrona

Networking

- API REST para ranking online
- WebSockets para multijugador
- Cloud save

Internacionalización

- Soporte multi-idioma
- Archivos de recursos `.resx`
- Textos configurables

Licencia y Créditos

Licencia

Este proyecto está bajo la licencia MIT. Ver archivo `LICENSE` para más detalles.

Tecnologías Utilizadas

- **Avalonia UI**: https://avaloniaui.net/
- **ReactiveUI**: https://www.reactiveui.net/
- **MongoDB**: https://www.mongodb.com/
- **SkiaSharp**: https://github.com/mono/SkiaSharp
- **.NET**: https://dotnet.microsoft.com/

Créditos

- **Diseño y Desarrollo**: Jax & Jona
- **Assets**: Jona
- **Testing**: Jax & Lexor

Contacto y Soporte

Git GIJAXF ### Reportar Bugs

Para reportar bugs, crear un issue en GitHub con:

- Descripción del problema
- Pasos para reproducir
- Comportamiento esperado vs actual
- Logs relevantes
- Sistema operativo y versión de .NET

Contribuciones

Las contribuciones son bienvenidas. Para contribuir:

1. Fork el repositorio

2. Crear una rama para tu feature:

   ```
   git checkout -b feature/AmazingFeature
   ```

3. Commit tus cambios:

   ```
   git commit -m 'Add some AmazingFeature'
   ```

4. Push a la rama:

   ```
   git push origin feature/AmazingFeature
   ```

5. Abrir un Pull Request

Guía de Estilo de Código

- Usar `PascalCase` para clases y métodos públicos
- Usar `camelCase` para variables privadas y parámetros
- Prefijo _ para campos privados
- Comentarios XML para métodos públicos
- Logs descriptivos con emojis para facilitar debugging

```
/// <summary>
/// Valida la solución de un puzzle y otorga puntos al jugador.
/// </summary>
/// <param name="playerId">ID del jugador</param>
/// <param name="puzzleId">ID del puzzle</param>
/// <param name="code">Código fuente Java</param>
/// <param name="output">Salida del programa</param>
/// <returns>Resultado de la validación</returns>
public async Task<PuzzleValidationResult> ValidateSolutionAsync(
    string playerId,
    string puzzleId,
    string code,
    string output)
{
    // Implementación...
}
```

Glosario

Término	Definición
Artifact	Contenedor visual independiente en Avalonia
MVVM	Model-View-ViewModel, patrón arquitectónico
ReactiveUI	Framework para programación reactiva en .NET
MongoDB	Base de datos NoSQL orientada a documentos
Sprite	Imagen 2D usada para representar personajes/objetos
Canvas	Control de Avalonia para posicionamiento absoluto
Game Loop	Ciclo principal que actualiza el juego cada frame
Collision Detection	Sistema para detectar solapamiento de objetos
Repository Pattern	Patrón para abstraer acceso a datos
Seed Data	Datos iniciales para poblar la base de datos

Apéndices

A. Estructura Completa de la Base de Datos

```
// HackerFantasmaDB
```

```
// Colección: players
{
  _id: ObjectId,
  name: String,
  gender: "Hombre" | "Mujer",
  avatar: String,
  currentLevel: Number,
  faction: String,
  totalScore: Number,
  health: Number,
  position: { x: Number, y: Number },
  velocity: { x: Number, y: Number },
  isJumping: Boolean,
  isFacingRight: Boolean,
  creationDate: ISODate,
  lastPlayed: ISODate
}

// Colección: levels
{
  _id: ObjectId,
  name: String,
  type: String,
  difficulty: Number,
  orderNumber: Number,
  description: String,
  timeLimit: Number,
  floor: {
    x: Number,
    y: Number,
    width: Number,
    height: Number,
    isSolid: Boolean,
    floorType: "Pasto" | "Hielo" | "Cristal" | "RedLine" | "PiedraTutorial
  },
  platforms: [{
    x: Number,
    y: Number,
    width: Number,
    height: Number,
    isSolid: Boolean
  }],
  enemies: [{
    x: Number,
    y: Number,
    type: String,
    patrolRoute: [String]
  }],
  npcIds: [ObjectId],
  background: String
}

// Colección: puzzles
{
  _id: ObjectId,
  name: String,
  type: String,
  description: String,
  expectedOutput: String,
  starterCode: String,
  levelId: ObjectId,
  difficulty: Number,
  hints: [String],
  points: Number,
  isCompleted: Boolean
}

// Colección: npcs
{
```

```
{
  _id: ObjectId,
  Name: String,
  Role: String,
  FactionId: ObjectId,
  PositionX: Number,
  PositionY: Number,
  Dialogue: [String],
  IsActive: Boolean,
  LevelId: ObjectId | null,
  Stats: {
    Strength: Number,
    Defense: Number,
    Intelligence: Number,
    Agility: Number
  }
}

// Colección: factions
{
  _id: ObjectId,
  name: String,
  type: String,
  leader: String,
  description: String,
  themeColor: String
}

// Colección: decisions
{
  _id: ObjectId,
  playerId: ObjectId,
  levelId: ObjectId,
  choice: String,
  factionAffected: String,
  factionPoints: Number,
  decisionDate: ISODate
}
```

B. Comandos Útiles

Compilar:

```
dotnet build
```

Ejecutar:

```
dotnet run
```

Limpiar:

```
dotnet clean
```

Publicar (Release):

```
dotnet publish -c Release -o ./publish
```

Restaurar paquetes:

```
dotnet restore
```

Actualizar paquetes:

```
dotnet list package --outdated
dotnet add package NombreDelPaquete --version X.X.X
```

C. Atajos de Teclado en el Juego

Tecla	Acción
← / A	Mover izquierda
→ / D	Mover derecha
↑ / W	Saltar
E	Interactuar con NPC
T	Abrir Terminal
ESC	Pausar / Salir
Space	(Reservado para futuro uso)

Fin de la Documentación Técnica

Versión 1.0 - Enero 2025

> **Nota**: Esta documentación está en constante evolución. Para la versión más actualizada, consulta el repositorio oficial del proyecto.

> **¿Encontraste un error en la documentación?** Por favor, repórtalo como un issue en GitHub con la etiqueta `documentation`.

> **¿Te gustó el proyecto?** ¡Déjanos una estrella en Github!