Table des matières

Listings

1 Structure projet P_SantaClash

Le projet est organisé dans une arborescence de dossiers.

2 Code source

2.1 Program.cs

```
using var game = new P_SantaClash.Game1();
game.Run();
```

Listing 1 – Program.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using System;
using System.Collections.Generic;
using System.Linq;

namespace P_SantaClash
{
    public class Game1 : Game
    {
        private GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;

        // Parallaxe (2 couches)
        private Texture2D _background1;
        private Texture2D _background2;
        private float _bg1Offset;
        private float _bg2Offset;

        // Texture "pixel" (1x1)
        private Texture2D _pixel;
        private Texture2D santaTexture;
        private Texture2D _projectileTexture;

        // Entités
        private Santa _santa;
        private Player _p1;
        private Player _p2;

        private readonly List<Enemy> _enemies = new();
        private readonly List<Projectile> _projectiles = new();

        private WaveManager _waveManager;
        private readonly GameStateManager _state = new();

        // Arena de jeu
        private Rectangle _arena;

        // Fin de partie / stats
        private string _gameOverText = "";

        public Game1()
        {
            _graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            IsMouseVisible = true;
        }

        protected override void Initialize()
        {
            base.Initialize();
            _arena = new Rectangle(0, 0, GraphicsDevice.Viewport.Width, ↩
                GraphicsDevice.Viewport.Height); // zone de jeu
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            // Fond écran
            _background1 = Content.Load<Texture2D>("wallpaper");
            _background2 = _background1;

            _pixel = new Texture2D(GraphicsDevice, 1, 1);
            _pixel.SetData(new[] { Color.White });

            // Santa au centre
            santaTexture = CreateSolidTexture(22, 22, Color.White);


            Vector2 santaPos = new Vector2(_arena.Width / 2f - santaTexture.Width ↩
               / 2f, _arena.Height / 2f - santaTexture.Height / 2f);
            _santa = new Santa(santaPos, 100, santaTexture, targetPosition: new ↩
               Vector2(_arena.Width / 2f, _arena.Height / 2f));

            // Joueurs (rectangles colorés via pixel "étiré")
            // On crée des textures dédiées pour avoir des tailles visibles.
            Texture2D p1Tex = CreateSolidTexture(22, 22, Color.Red);
            Texture2D p2Tex = CreateSolidTexture(22, 22, Color.Blue);
            Texture2D enemyTex = CreateSolidTexture(18, 18, Color.Green);
            Texture2D projTex = CreateSolidTexture(8, 8, Color.Yellow);

            _p1 = new Player(1, new Vector2(_arena.Width * 0.25f, _arena.Height * ↩
               0.70f), p1Tex);
            _p2 = new Player(2, new Vector2(_arena.Width * 0.75f, _arena.Height * ↩
               0.70f), p2Tex);

            _waveManager = new WaveManager(_enemies, enemyTex);

            // On stocke la texture projectile via un champ "hack" simple :
            _projectileTexture = projTex;

            UpdateWindowTitle();
        }

        private Texture2D CreateSolidTexture(int w, int h, Color color)
        {
            Texture2D t = new Texture2D(GraphicsDevice, w, h);
            Color[] data = Enumerable.Repeat(color, w * h).ToArray();
            t.SetData(data);
            return t;
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ↩
               ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            switch (_state.State)
            {
                case GameState.Menu:
                    UpdateMenu();
                    break;

                case GameState.Playing:
                    UpdatePlaying(gameTime);
                    break;

                case GameState.GameOver:
                    UpdateGameOver();
                    break;
            }

            base.Update(gameTime);
        }

        private void UpdateMenu()
        {
            var k = Keyboard.GetState();
            var gp = GamePad.GetState(PlayerIndex.One);

            Window.Title = "Santa Clashs MENU (Enter / Start pour jouer)";
            if (k.IsKeyDown(Keys.Enter) || gp.Buttons.Start == ButtonState.Pressed)
```

```
            {
                StartNewGame();
            }
        }

        private void StartNewGame()
        {
            _state.StartGame();
            _enemies.Clear();
            _projectiles.Clear();
            _waveManager.Reset();

            // Reset Santa / joueurs
            _santa.ApplyDamage(-999999); // noop visuel, on ne veut pas ↩
               d'overcomplication ici
            // on recrée proprement
            Vector2 center = new Vector2(_arena.Width / 2f, _arena.Height / 2f);
            _santa = new Santa(new Vector2(center.X - 16, center.Y - 16), 100, ↩
               santaTexture, center);

            _p1 = new Player(1, new Vector2(_arena.Width * 0.25f, _arena.Height * ↩
               0.70f), CreateSolidTexture(22, 22, Color.Red));
            _p2 = new Player(2, new Vector2(_arena.Width * 0.75f, _arena.Height * ↩
               0.70f), CreateSolidTexture(22, 22, Color.Blue));

            _gameOverText = "";
            UpdateWindowTitle();
        }

        private void UpdatePlaying(GameTime gameTime)
        {
            // Parallaxe simple (défilement horizontal)
            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
            _bg1Offset = (_bg1Offset + 20f * dt) % _arena.Width;
            _bg2Offset = (_bg2Offset + 40f * dt) % _arena.Width;

            _p1.Update(gameTime);
            _p2.Update(gameTime);
            _santa.Update(gameTime);

            ClampToArena(_p1);
            ClampToArena(_p2);
            ClampToArena(_santa);

            _waveManager.Update(gameTime, _arena);

            foreach (Enemy e in _enemies.Where(x => x.IsAlive))
                e.Update(gameTime, _santa.Position);

            foreach (Projectile p in _projectiles.Where(x => x.IsAlive))
                p.Update(gameTime);

            // Tir
            HandleShooting(_p1);
            HandleShooting(_p2);

            // ennemis dangereux (proches de Santa)
            List<Enemy> dangerousEnemies = _enemies
                .Where(e => e.IsAlive)
                .Where(e => Vector2.Distance(e.Position, _santa.Position) < 80f)
                .ToList();

            // Contact ennemis -> Santa
            foreach (Enemy e in dangerousEnemies)
            {
                if (e.Hitbox.Intersects(_santa.Hitbox))
                {
                    _santa.ApplyDamage(e.ContactDamage);
                    e.IsAlive = false; // "s'écrase" sur Santa
                }
            }

            // couples projectile/enemy en collision -> met en liste les ↩
               projectiles et les enemies qui se touchent
            var hitPairs =
              (from proj in _projectiles
                where proj.IsAlive
                from enemy in _enemies
                where enemy.IsAlive && proj.Hitbox.Intersects(enemy.Hitbox)
                select (proj, enemy))
              .ToList();

            // tuer les deux instances
            foreach (var (proj, enemy) in hitPairs)
            {
                proj.IsAlive = false;
                enemy.IsAlive = false;

                if (proj.OwnerPlayerId == 1) _p1.AddKill();
                else _p2.AddKill();
            }

            // Nettoyage (hors écran ou morts)
            foreach (Projectile p in _projectiles.Where(p => p.IsAlive))
            {
                if (!_arena.Contains(p.Hitbox))
                    p.IsAlive = false;
            }

            _projectiles.RemoveAll(p => !p.IsAlive);
            _enemies.RemoveAll(e => !e.IsAlive && Vector2.Distance(e.Position, ↩
                _santa.Position) > 500f);

            UpdateWindowTitle();

            if (!_santa.IsAlive)
            {
                BuildGameOverStats();
                _state.GameOver();
            }
        }

        private void HandleShooting(Player player)
        {
            if (!player.WantsToShoot()) return;

            player.MarkShotFired();

            // Direction de tir simple : vers le centre (Santa) mais inversée = on ↩
                tire vers les ennemis autour
            Vector2 dir = player.Velocity;
            if (dir == Vector2.Zero) dir = new Vector2(0, -1);
            else dir.Normalize();

            Vector2 velocity = dir * 420f;
            Vector2 spawn = player.Position + new Vector2(10, 10);

            _projectiles.Add(new Projectile(spawn, velocity, _projectileTexture, ↩
                player.PlayerId));
        }

        private void ClampToArena(GameObject obj)
        {
            // clamp position dans la fenêtre
            float x = MathHelper.Clamp(obj.Position.X, 0, _arena.Width - 1);
            float y = MathHelper.Clamp(obj.Position.Y, 0, _arena.Height - 1);
            obj.Position = new Vector2(x, y);
        }

        private void UpdateGameOver()
        {
            KeyboardState k = Keyboard.GetState();
            GamePadState gp = GamePad.GetState(PlayerIndex.One);

            Window.Title = $"Santa Clash GAME OVER | {_gameOverText} (R pour ↩
                rejouer / Esc pour quitter)";
            if (k.IsKeyDown(Keys.R) || gp.Buttons.Start == ButtonState.Pressed)
                _state.GoToMenu();
        }

        private void BuildGameOverStats()
        {
            // classement final
            var ranking = new[]
                {
                    new { Player = _p1, Name = "Joueur 1 (Manette)" },
                    new { Player = _p2, Name = "Joueur 2 (Clavier)" },
                }
                .OrderByDescending(x => x.Player.Score)
                .ThenByDescending(x => x.Player.Accuracy)
                .ToList();

            var winner = ranking.First();

            _gameOverText =
                $"Gagnant: {winner.Name} | Scores: P1={_p1.Score} P2={_p2.Score} | ↩
                    " +
                $"Precision: P1={_p1.Accuracy:P0} P2={_p2.Accuracy:P0}";
        }

        private void UpdateWindowTitle()
        {
            if (_state.State != GameState.Playing) return;
            Window.Title = $"Santa Clash | Vie Santa: ↩
               {_santa.CurrentHealth}/{_santa.MaxHealth} | " +
                           $"P1 Score: {_p1.Score} (Acc {_p1.Accuracy:P0}) | " +
                           $"P2 Score: {_p2.Score} (Acc {_p2.Accuracy:P0}) | Vague ↩
                               {_waveManager.Wave}";
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Black);

            _spriteBatch.Begin();

            DrawParallaxBackground();

            // Santa + joueurs + ennemis + projectiles
            _santa.Draw(_spriteBatch);
            _p1.Draw(_spriteBatch);
            _p2.Draw(_spriteBatch);

            foreach (Enemy e in _enemies.Where(x => x.IsAlive))
                e.Draw(_spriteBatch);

            foreach (Projectile p in _projectiles.Where(x => x.IsAlive))
                p.Draw(_spriteBatch);

            DrawHudBars();



            _spriteBatch.End();

            base.Draw(gameTime);
        }

        private void DrawParallaxBackground()
        {
            // couche 1
            DrawTiled(_background1, _bg1Offset, Color.White * 0.85f);
            // couche 2 (plus rapide)
            DrawTiled(_background2, _bg2Offset, Color.White * 0.65f);
        }

        private void DrawTiled(Texture2D tex, float offset, Color color)
        {
            int w = _arena.Width;
            int h = _arena.Height;

            // on dessine 2 fois pour boucler
            Rectangle r1 = new Rectangle((int)-offset, 0, w, h);
            Rectangle r2 = new Rectangle((int)(w - offset), 0, w, h);

            _spriteBatch.Draw(tex, r1, color);
            _spriteBatch.Draw(tex, r2, color);
        }
        private void DrawHudBars()
        {
            // HUD sans texte : barres (vie Santa + 2 barres score)
            int pad = 10;
            int barW = 220;
            int barH = 14;

            // Vie Santa
            float healthRatio = _santa.MaxHealth == 0 ? 0 : ↩
                (float)_santa.CurrentHealth / _santa.MaxHealth;
            DrawBar(new Rectangle(pad, pad, barW, barH), healthRatio, ↩
                Color.DarkRed, Color.Red);

            // Scores (échelle arbitraire)
            float p1Ratio = MathHelper.Clamp(_p1.Score / 30f, 0, 1);
            float p2Ratio = MathHelper.Clamp(_p2.Score / 30f, 0, 1);

            DrawBar(new Rectangle(pad, pad + 22, barW, barH), p1Ratio, ↩
                Color.DarkGray, Color.Red);
            DrawBar(new Rectangle(pad, pad + 44, barW, barH), p2Ratio, ↩
                Color.DarkGray, Color.Blue);
        }

        private void DrawBar(Rectangle area, float ratio, Color back, Color fill)
        {
            _spriteBatch.Draw(_pixel, area, back);
            Rectangle filled = new Rectangle(area.X, area.Y, (int)(area.Width * ↩
                ratio), area.Height);
            _spriteBatch.Draw(_pixel, filled, fill);
        }
    }
}
```

Listing 2 – Game1.cs

2.2 Class

2.2.1 Enemy

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;

namespace P_SantaClash
{
    public abstract class Enemy : GameObject
    {
        protected readonly Texture2D Texture;
        protected readonly Random Random = new Random();

        public int ContactDamage { get; protected set; } = 5;

        protected float Speed = 80f;
        protected float MaxAngleOffset = MathF.PI / 12f;

        public Rectangle Hitbox =>
            new Rectangle((int)Position.X, (int)Position.Y, Texture.Width, ↩
                Texture.Height);

        protected Enemy(Vector2 position, Texture2D texture, float speed, float ↩
           maxAngleOffset, int contactDamage)
            : base(position, Vector2.Zero)
        {
            Texture = texture;
            Speed = speed;
            MaxAngleOffset = maxAngleOffset;
            ContactDamage = contactDamage;
        }

        /// <summary>
        /// Déplacement pseudo-aléatoire vers Santa (zigzag/rotation) comme dans ↩
           le PDF.
        /// </summary>
        public void Update(GameTime gameTime, Vector2 santaPosition)
        {
            if (!IsAlive) return;

            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;

            Vector2 dir = santaPosition - Position;
            if (dir == Vector2.Zero) return;
            dir.Normalize();

            float angleOffset = (float)(Random.NextDouble() - 0.5) * 2f * ↩
                MaxAngleOffset;
            float cos = MathF.Cos(angleOffset);
            float sin = MathF.Sin(angleOffset);

            Vector2 dirRotated = new Vector2(
                dir.X * cos - dir.Y * sin,
                dir.X * sin + dir.Y * cos
            );

            Position += dirRotated * Speed * dt;
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (!IsAlive) return;
            spriteBatch.Draw(Texture, Position, Color.White);
        }
    }

    public sealed class SlowEnemy : Enemy
    {
        public SlowEnemy(Vector2 position, Texture2D texture)
            : base(position, texture, speed: 60f, maxAngleOffset: MathF.PI / 16f, ↩
                contactDamage: 8) { }
    }

    public sealed class FastEnemy : Enemy
    {
        public FastEnemy(Vector2 position, Texture2D texture)
            : base(position, texture, speed: 120f, maxAngleOffset: MathF.PI / 10f, ↩
                contactDamage: 4) { }
    }
}
```

Listing 3 – Enemy.cs

2.2.2 GameObject

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace P_SantaClash
{
    /// <summary>
    /// Base POO imposée dans le sujet : position 2D, vitesse, IsAlive, Update(), ↩
        Draw()
    /// </summary>
    public abstract class GameObject
    {
        public Vector2 Position { get; set; }
        public Vector2 Velocity { get; set; }
        public bool IsAlive { get; set; } = true;

        protected GameObject(Vector2 position, Vector2 velocity)
        {
            Position = position;
            Velocity = velocity;
        }

        public virtual void Update(GameTime gameTime) { }
        public virtual void Draw(SpriteBatch spriteBatch) { }
    }
}
```

Listing 4 – GameObject.cs

2.2.3 GameStateManager

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

    public class GameStateManager
    {
        public GameState State { get; private set; } = GameState.Menu;

        public void GoToMenu() => State = GameState.Menu;
        public void StartGame() => State = GameState.Playing;
        public void GameOver() => State = GameState.GameOver;
    }
}
```

Listing 5 – GameStateManager.cs

2.2.4 IDamageable

```
using Microsoft.Xna.Framework;

namespace P_SantaClash
{
    public interface IDamageable
    {
        void ApplyDamage(int amount);
        bool IsAlive { get; }
    }
}
```

Listing 6 – IDamageable.cs

2.2.5 Player

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace P_SantaClash
{
    public class Player : GameObject
    {
        private const float MoveSpeed = 200f;
        private const float ShootCooldown = 0.25f;

        private readonly Texture2D _texture;
        private readonly int _playerId;

        private float _shootTimer;

        public int PlayerId => _playerId;

        public int Score { get; private set; }
        public int ShotsFired { get; private set; }
        public int ShotsHit { get; private set; }

        public Rectangle Hitbox => new Rectangle((int)Position.X, (int)Position.Y, ↩
            _texture.Width, _texture.Height);
        public Player(int playerId, Vector2 position, Texture2D texture) : ↩
            base(position, Vector2.Zero)
        {
            _playerId = playerId;
            _texture = texture;
        }

        public float Accuracy => ShotsFired == 0 ? 0f : (float)ShotsHit / ShotsFired;

        public void AddKill()
        {
            Score++;
            ShotsHit++;
        }

        public override void Update(GameTime gameTime)
        {
            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
            if (_shootTimer > 0) _shootTimer -= dt;

            Vector2 move = Vector2.Zero;

            if (_playerId == 1)
            {
                GamePadState gp = GamePad.GetState(PlayerIndex.One);

                // Use D-pad for movement
                if (gp.DPad.Up == ButtonState.Pressed) move.Y -= 1;
                if (gp.DPad.Down == ButtonState.Pressed) move.Y += 1;
                if (gp.DPad.Left == ButtonState.Pressed) move.X -= 1;
                if (gp.DPad.Right == ButtonState.Pressed) move.X += 1;
            }
            else
            {
                KeyboardState k = Keyboard.GetState();
                if (k.IsKeyDown(Keys.W)) move.Y -= 1;
                if (k.IsKeyDown(Keys.S)) move.Y += 1;
                if (k.IsKeyDown(Keys.A)) move.X -= 1;
                if (k.IsKeyDown(Keys.D)) move.X += 1;
            }

            if (move != Vector2.Zero)
            {
                move.Normalize();
                Position += move * MoveSpeed * dt;
            }
        }

        public bool WantsToShoot()
        {
            if (_shootTimer > 0) return false;

            if (_playerId == 1)
            {
                GamePadState gp = GamePad.GetState(PlayerIndex.One);
                return gp.IsButtonDown(Buttons.A);
            }
            else
            {
                KeyboardState k = Keyboard.GetState();
                return k.IsKeyDown(Keys.Space);
            }
        }

        public void MarkShotFired()
        {
            ShotsFired++;
            _shootTimer = ShootCooldown;
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(_texture, Position, Color.White);
        }
    }
}
```

Listing 7 – Player.cs

2.2.6 Projectile

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

namespace P_SantaClash
{
    public class Projectile : GameObject
    {
        private readonly Texture2D _texture;
        public int OwnerPlayerId { get; }

        public Rectangle Hitbox => new Rectangle((int)Position.X, (int)Position.Y, ↩
            _texture.Width, _texture.Height);

        public Projectile(Vector2 position, Vector2 velocity, Texture2D texture, ↩
            int ownerPlayerId) : base(position, velocity)
        {
            _texture = texture;
            OwnerPlayerId = ownerPlayerId;
        }

        public override void Update(GameTime gameTime)
        {
            if (!IsAlive) return;

            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
            Position += Velocity * dt;
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (!IsAlive) return;
            spriteBatch.Draw(_texture, Position, Color.White);
        }
    }
}
```

Listing 8 – Projectile.cs

2.2.7 Santa

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;

namespace P_SantaClash
{
    public class Santa : GameObject, IDamageable
    {
        public int MaxHealth { get; }
        public int CurrentHealth { get; private set; }

        private readonly Texture2D _texture;
        private readonly Vector2 _targetPosition;
        private readonly Random _random = new Random();

        private float _attractionStrength = 50f;
        private float _noiseStrength = 20f;
        private float _damping = 0.6f;

        public Rectangle Hitbox => new Rectangle((int)Position.X, (int)Position.Y, ↩
            _texture.Width, _texture.Height);

        public Santa(Vector2 position, int maxHealth, Texture2D texture, Vector2? ↩
            targetPosition = null) : base(position, Vector2.Zero)
        {
            MaxHealth = maxHealth;
            CurrentHealth = maxHealth;
            _texture = texture;
            _targetPosition = targetPosition ?? position; // centre de la map
        }


        // Gestions des dégats
        public void ApplyDamage(int amount) => TakeDamage(amount);
        public void TakeDamage(int dmg)
        {
            if (!IsAlive) return;

            CurrentHealth -= dmg;
            if (CurrentHealth <= 0)
            {
                CurrentHealth = 0;
                IsAlive = false;
            }
        }

        public override void Update(GameTime gameTime)
        {
            if (!IsAlive) return;

            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;

            // Mouvement chaotique

            // bougers vers cible
            Vector2 toTarget = _targetPosition - Position;
            if (toTarget != Vector2.Zero)
                toTarget.Normalize();

            // bruit aléatoire
            float noiseX = (float)(_random.NextDouble() - 0.5f);
            float noiseY = (float)(_random.NextDouble() - 0.5f);
            Vector2 noise = new Vector2(noiseX, noiseY);
            if (noise != Vector2.Zero)
                noise.Normalize();

            // vitesse et accelerations
            Vector2 acceleration = toTarget * _attractionStrength + noise * ↩
               _noiseStrength;

            // lent? plus vite.
            if (Velocity.Length() < 15f)
                Velocity += noise * 30f;

            Velocity *= _damping; // amortir
            Position += Velocity * dt;
        }

        public override void Draw(SpriteBatch spriteBatch)
        {
            if (!IsAlive) return;
            spriteBatch.Draw(_texture, Position, Color.White);
        }
    }
}
```

Listing 9 – Santa.cs

2.2.8 WaveManager

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;

namespace P_SantaClash
{
    public class WaveManager
    {
        private readonly Random _random = new Random();
        private readonly List<Enemy> _enemies;
        private readonly Texture2D _enemyTexture;

        private float _spawnTimer;
        private float _spawnInterval = 1.1f;

        private int _wave = 1;

        public int Wave => _wave;

        public WaveManager(List<Enemy> enemies, Texture2D enemyTexture)
        {
            _enemies = enemies;
            _enemyTexture = enemyTexture;
        }

        public void Reset()
        {
            _enemies.Clear();
            _spawnTimer = 0;
            _spawnInterval = 1.1f;
            _wave = 1;
        }

        public void Update(GameTime gameTime, Rectangle arena)
        {
            float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;
            _spawnTimer -= dt;

            // Exemple simple : toutes les X secondes on ajoute un ennemi. Le ↩
                rythme augmente avec les vagues.
            if (_spawnTimer <= 0)
            {
                SpawnEnemy(arena);
                _spawnTimer = _spawnInterval;

                // Une vague monte quand il y a déjà beaucoup d'ennemis (vivants) à↩
                     l'écran.
                int aliveCount = _enemies.Count(e => e.IsAlive);
                if (aliveCount >= 8 + _wave * 2)
                {
                    _wave++;
                    _spawnInterval = MathF.Max(0.45f, _spawnInterval - 0.08f);
                }
            }
        }

        private void SpawnEnemy(Rectangle arena)
        {
            // Spawn sur les bords
            int side = _random.Next(4);
            Vector2 pos = side switch
            {
                0 => new Vector2(arena.Left - 20, _random.Next(arena.Top, ↩
                    arena.Bottom)),   // gauche
                1 => new Vector2(arena.Right + 20, _random.Next(arena.Top, ↩
                    arena.Bottom)),  // droite
                2 => new Vector2(_random.Next(arena.Left, arena.Right), arena.Top ↩
                    - 20),     // haut
                _ => new Vector2(_random.Next(arena.Left, arena.Right), ↩
                    arena.Bottom + 20)  // bas
            };

            // 2 types d'ennemis minimum
            Enemy e = (_random.NextDouble() < 0.5)
                ? new SlowEnemy(pos, _enemyTexture)
                : new FastEnemy(pos, _enemyTexture);

            _enemies.Add(e);
        }
    }
}
```

Listing 10 – WaveManager.cs

2.3 TestCases et Modules

2.3.1 PlayerStats

```
namespace P_SantaClash.Core
{
    public class PlayerStats
    {
        public int PlayerId { get; }
        public int ShotsFired { get; private set; }
        public int ShotsHit { get; private set; }
        public int Score { get; private set; }

        public PlayerStats(int playerId)
        {
            PlayerId = playerId;
        }

        public void RegisterShot() => ShotsFired++;
        public void RegisterHit()
        {
            ShotsHit++;
            Score++;
        }

        public double Accuracy()
        {
            if (ShotsFired == 0) return 0.0;
            return (double)ShotsHit / ShotsFired;
        }
    }
}
```

Listing 11 – PlayerStats.cs

2.3.2 StatsService

```
using System.Collections.Generic;
using System.Linq;

namespace P_SantaClash.Core
{
    public static class StatsService
    {
        /// <summary>
        /// Classement final : score desc, précision desc.
        /// </summary>
        public static List<PlayerStats> RankPlayers(IEnumerable<PlayerStats> players)
        {
            return players
                .OrderByDescending(p => p.Score)
                .ThenByDescending(p => p.Accuracy())
                .ToList();
        }

        /// <summary>
        /// Exemple de LINQ : combien d'ennemis par type.
        /// </summary>
        public static Dictionary<EnemyType, int> ↩
           EnemiesByType(IEnumerable<EnemySpawnInfo> spawns)
        {
            return spawns
                .GroupBy(s => s.Type)
                .ToDictionary(g => g.Key, g => g.Count());
        }
    }
}
```

Listing 12 – StatsService.cs

2.3.3 AccuracyTests

```csharp
using NUnit.Framework;
using P_SantaClash.Core;

namespace P_SantaClash.Tests
{
    public class AccuracyTests
    {
        [Test]
        public void Accuracy_WhenZeroShots_ReturnsZero()
        {
            PlayerStats p = new PlayerStats(1);
            Assert.That(p.Accuracy(), Is.EqualTo(0.0));
        }

        [Test]
        public void Accuracy_NormalCase()
        {
            PlayerStats p = new PlayerStats(1);
            p.RegisterShot();
            p.RegisterShot();
            p.RegisterShot();
            p.RegisterHit(); // 1 hit / 3 shots
            Assert.That(p.Accuracy(), Is.EqualTo(1.0 / 3.0).Within(1e-9));
        }

        [Test]
        public void Score_IncrementsOnHit()
        {
            PlayerStats p = new PlayerStats(1);
            p.RegisterShot();
            p.RegisterHit();
            Assert.That(p.Score, Is.EqualTo(1));
        }
    }
}
```

Listing 13 – AccuracyTests.cs

2.3.4 RankingTests

```csharp
using NUnit.Framework;
using P_SantaClash.Core;
using System.Collections.Generic;

namespace P_SantaClash.Tests
{
    public class RankingTests
    {
        [Test]
        public void RankPlayers_SortsByScoreDesc()
        {
            PlayerStats p1 = new PlayerStats(1);
            PlayerStats p2 = new PlayerStats(2);

            // p1 score 2
            p1.RegisterShot(); p1.RegisterHit();
            p1.RegisterShot(); p1.RegisterHit();

            // p2 score 1
            p2.RegisterShot(); p2.RegisterHit();

            List<PlayerStats> ranked = StatsService.RankPlayers(new ↩
               List<PlayerStats> { p2, p1 });
            Assert.That(ranked[0].PlayerId, Is.EqualTo(1));
        }

        [Test]
        public void RankPlayers_TieBreakByAccuracy()
        {
            PlayerStats p1 = new PlayerStats(1);
            PlayerStats p2 = new PlayerStats(2);

            // même score : 2
            p1.RegisterShot(); p1.RegisterHit();
            p1.RegisterShot(); p1.RegisterHit(); // 2/2 => 100%

            p2.RegisterShot(); p2.RegisterHit();
            p2.RegisterShot();
            p2.RegisterHit(); // 2/3 => 66%

            List<PlayerStats> ranked = StatsService.RankPlayers(new ↩
                List<PlayerStats> { p2, p1 });
            Assert.That(ranked[0].PlayerId, Is.EqualTo(1));
        }
    }
}
```

Listing 14 – RankingTests.cs

3 Annexes

Table des figures

17 / 17