Geometry Wars

INF01121 - MLP

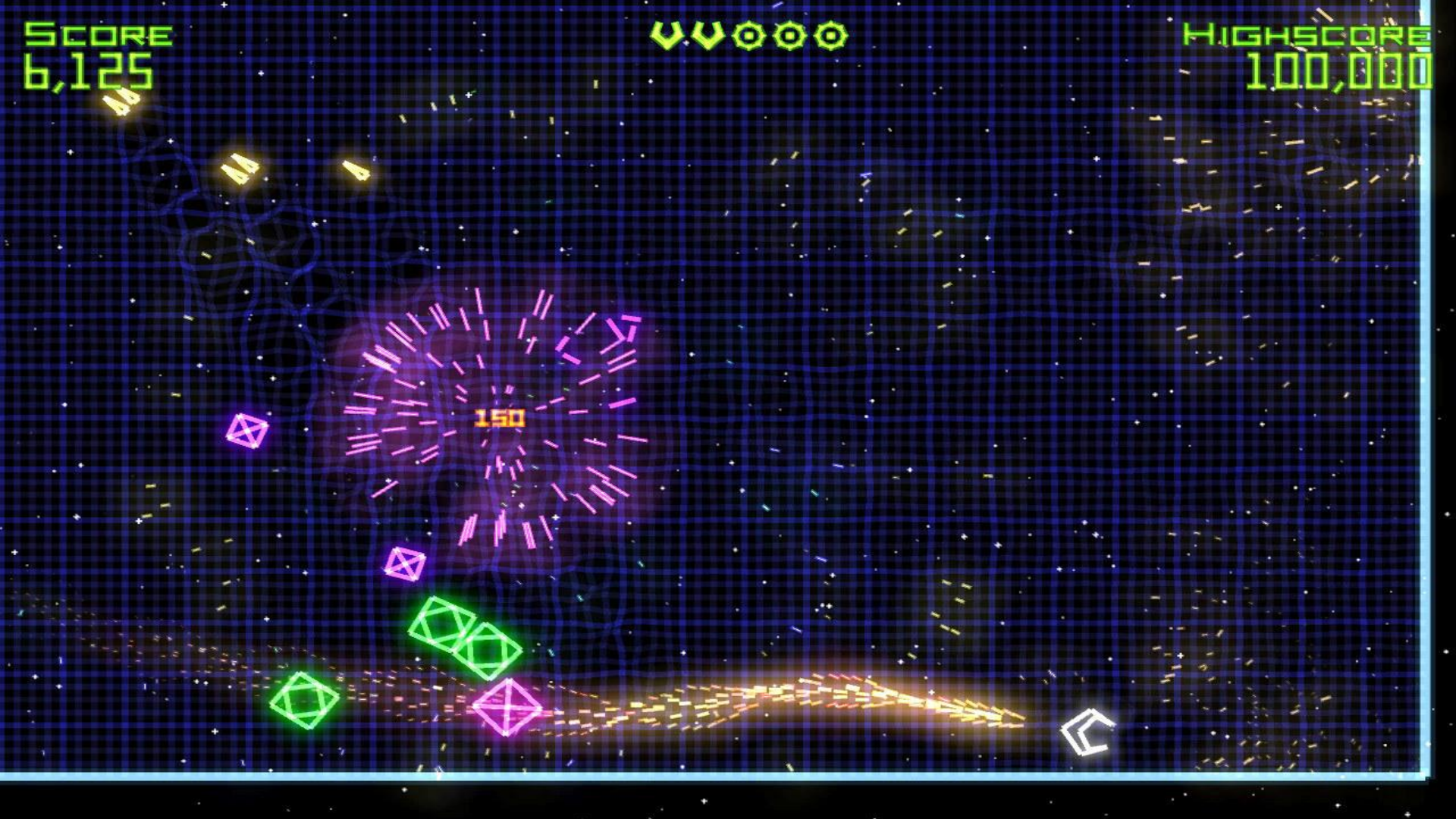
Aluno Luis Filipe Rodrigues

Geometry Wars

- Origem: criado como um minigame dentro de "Project Gotham Racing 2" (2003), Geometry Wars é um jogo de tiro arcade em estilo "twin-stick" (dois analógicos).
- Gameplay: O jogador controla uma pequena nave que deve sobreviver a ondas de inimigos geométricos, usando dois analógicos (um para movimento e outro para atirar em qualquer direção). O objetivo é alcançar a maior pontuação possível.

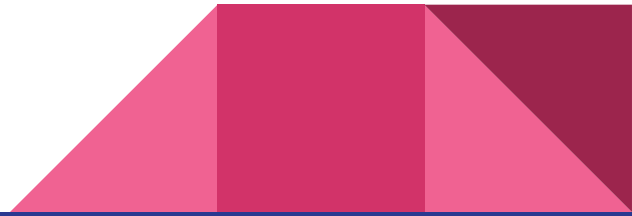
Geometry Wars

- Estilo Visual: Visual minimalista e neon, com efeitos de partículas vibrantes. A trilha sonora é eletrônica e intensa, complementando a ação frenética.
- Evolução: De um simples minigame, tornou-se uma franquia com vários títulos, incluindo "Geometry Wars: Retro Evolved" e "Geometry Wars 3: Dimensions".
- Influente no gênero arcade moderno, conhecido por sua jogabilidade viciante e alto nível de desafio.


Score
6,125
Highscore
100,000
150

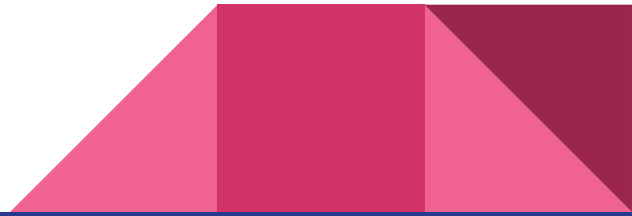
O que foi utilizado?

- C++20
- Backend de renderização DirectX11
- IDE Visual Studio
- Herança
- Polimorfismo
- Álgebra Linear



Classes

- Classe Engine: contém funções para renderização de gráficos e texto, áudio, cálculos de física e tipos de dados. Inicializado a partir de um Game e contém o loop principal do jogo (enquanto o usuário não fechar a janela).
- Classe Game: oferece uma interface mínima para a criação de um jogo com métodos virtuais de Inicialização, Atualização, Desenho e Finalização.



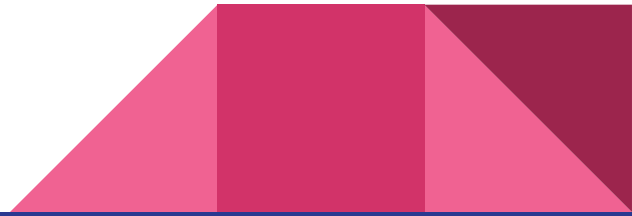
Classes

```
// cria motor do jogo
Engine * engine = new Engine();

// configura janela
//engine->window->Mode(WINDOWED);
//engine->window->Size(1152, 648);
engine->window->Mode(BORDERLESS);
engine->window->Color(0, 0, 0);
engine->window->Title("GeoWars");
engine->window->Icon(IDI_ICON);
engine->window->Cursor(IDC_CURSOR);
engine->window->HideCursor(true);
//engine->graphics->VSync(true);
```

```
// cria o jogo
Game * game = new GeoWars();

// configura o jogo
game->Size(3840, 2160);

// inicia execução
int status = engine->Start(game);

// destrói motor
delete engine;

// encerra
return status;
```

Classes

- Classe Object: Oferece atributos e métodos básicos para um objeto além de métodos virtuais sobrescritos de atualização e desenho
- Classe Scene: basicamente um contêiner de Objects, com funções para adicionar, remover objetos de uma cena e aplicar os métodos de atualização e desenho para cada



Classes

```
// carrega imagens das geometrias
blue    = new Image("Resources/Blue.png");
green   = new Image("Resources/Green.png");
magenta = new Image("Resources/Magenta.png");
orange  = new Image("Resources/Orange.png");

// carrega/incializa objetos
backg   = new Background("Resources/Space.jpg");
player  = new Player();
scene   = new Scene();
hud     = new Hud();

// adiciona objetos na cena
scene->Add(player, MOVING);
scene->Add(new Delay(), STATIC);
```

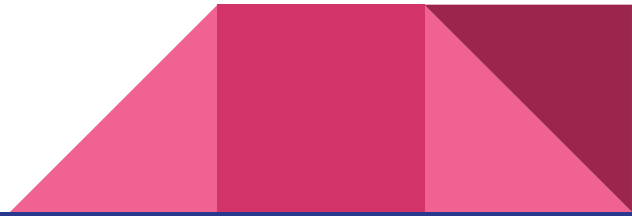
Classes

```
// atualiza cena e calcula colisões
scene->Update();
scene->CollisionDetection();
```

```
// desenha pano de fundo
backg->Draw(viewport);

// desenha a cena
scene->Draw();

// desenha o painel de informações
if (viewHUD)
    hud->Draw();

// desenha bounding box
if (viewBBox)
    scene->DrawBBox();
```

Principais objetos

- Player
- Missile
- Enemy Blue, Green, Magenta e Orange
- Explosion (usa sistema de partículas da Engine)
- Wave (gerador de inimigos)



Principais objetos

```
void Green::OnCollision(Object * obj)
{
    if (obj->Type() == MISSILE)
    {
        GeoWars::scene->Delete(obj, STATIC);
        GeoWars::scene->Delete(this, MOVING);
        GeoWars::scene->Add(new Explosion(x, y), STATIC);
        GeoWars::audio->Play(EXPLODE);
    }
}
```

Principais objetos

```
class Explosion : public Object
{
private:
    Particles * sparks;                 // sistema de partículas

public:
    Explosion(float pX, float pY);      // construtor
    ~Explosion();                       // destrutor

    int Size();                         // quantidade de partículas
    void Update();                      // atualização
    void Draw();                        // desenho
};
```

```
class Particles
{
private:
    Generator config;                                   // configuração do gerador de partículas
    Sprite * sprite;                                    // sprite da partícula
    list<Particle*> particles;                          // lista de partículas

    Timer timer;                                        // controle de tempo
    RandF spread { -config.spread/2, config.spread/2 }; // valores aleatórios para o espalhamento
    RandF speed  { config.minSpeed, config.maxSpeed };  // valores aleatórios para a velocidade

    void Build(float posX, float posY);                 // cria uma partícula

public:
    Particles(const Generator & generator);             // construtor
    ~Particles();                                       // destrutor

    uint Size();                                        // retorna o número de partículas
    bool Inactive();                                    // retorna o estado das partículas
    Generator& Config();                                // retorna referência para gerador
    void Generate(float x, float y, int count = 1);     // gera novas partículas
    void Update(float delta);                           // atualiza posição das partículas por delta
    void Draw(float z, float factor = 0.0f);            // desenha partículas
};
```

Demo!