Beyond the Dome

Guilherme Weimer[1], **Eduardo Ibarr de Paula**[1]

[1]Universidade Regional Integrada do Alto Uruguai e das Missões (URI)
Avenida Batista Bonoto Sobrinho, 733 – 97711-500 – Santiago – RS – Brasil

062282@urisantiago.br, eduardoibarr56@gmail.com

Abstract. *This paper details the design and implementation process of 'Beyond the Dome', a 2D action-adventure game developed as part of the Game Development course curriculum at URI - Campus Santiago. Built using Python and the Pygame library, the game immerses the player in a post-apocalyptic world where the protagonist, Kael, must explore hostile environments outside a protective dome to secure vital resources. Key technical features include procedural level generation based on noise algorithms, physics-driven character mechanics, diverse AI-controlled enemies, environmental hazards such as radiation, and multiple collectible items enhancing survival. The project emphasized a modular software architecture for maintainability, applied optimization techniques for performance, and focused on creating an engaging player experience through responsive controls and coherent audiovisual feedback. This document outlines the core systems, technical challenges, and potential future directions for the project.*

Resumo. *Este artigo detalha o processo de design e implementação do jogo 'Beyond the Dome', desenvolvido como projeto da disciplina de Desenvolvimento de Jogos na URI - Campus Santiago. Construído em Python com a biblioteca Pygame, o jogo imerge o jogador em um mundo pós-apocalíptico onde o protagonista, Kael, deve explorar ambientes hostis fora de uma cúpula protetora para obter recursos vitais. Funcionalidades técnicas chave incluem geração procedural de níveis baseada em algoritmos de ruído, mecânicas de personagem orientadas por física, inimigos diversos controlados por IA, perigos ambientais como radiação, e múltiplos itens coletáveis que auxiliam na sobrevivência. O projeto enfatizou uma arquitetura de software modular para manutenibilidade, aplicou técnicas de otimização para desempenho, e focou na criação de uma experiência de jogador engajadora através de controles responsivos e feedback audiovisual coerente. Este documento descreve os sistemas centrais, desafios técnicos e potenciais direções futuras para o projeto.*

1. Introdução

Este artigo apresenta o projeto de desenvolvimento do jogo digital 2D intitulado "Beyond the Dome", concebido e implementado no âmbito da disciplina de Desenvolvimento de Jogos. Desenvolvido primariamente para a plataforma PC, o jogo combina elementos de ação, aventura e sobrevivência, inseridos em uma narrativa de ficção científica pós-apocalíptica.

O enredo centraliza-se em Kael, um técnico residente na cidade tecnologicamente avançada, porém isolada, de Ômega-7. Esta metrópole é protegida por uma cúpula

energética, gerenciada por uma Inteligência Artificial (IA), que a defende de um ambiente externo devastado por fenômenos tóxicos e altos níveis de radiação. A premissa do jogo é acionada pela escassez de recursos vitais dentro da cúpula, levando a IA a designar Kael para uma missão crítica no exterior: localizar e recuperar Módulos de Filtro essenciais e outros suprimentos.

O jogador assume o controle de Kael, navegando por um mundo gerado proceduralmente, enfrentando ameaças ambientais e combatendo inimigos hostis. O objetivo primário é coletar os Módulos de Filtro para garantir a funcionalidade do sistema de suporte à vida de Ômega-7. No entanto, a exploração revela inconsistências e segredos sobre a verdadeira natureza da catástrofe externa e as motivações da IA, introduzindo um conflito moral e narrativo que desafia as percepções iniciais do jogador.

A escolha tecnológica recaiu sobre a linguagem Python e a biblioteca Pygame, devido à sua acessibilidade, vasta comunidade de suporte, rica documentação e adequação para o desenvolvimento rápido de jogos 2D. Esta combinação permitiu à equipe focar nos aspectos de design e jogabilidade, aproveitando a flexibilidade do Python e as funcionalidades multimídia do Pygame. As seções seguintes aprofundam a arquitetura de software, os sistemas de jogo implementados, as mecânicas de jogabilidade, os desafios técnicos enfrentados e as perspectivas futuras do projeto.

2. Arquitetura e Implementação

A estrutura de ”Beyond the Dome”foi concebida visando a clareza, manutenibilidade e a capacidade de expansão, utilizando as funcionalidades oferecidas pelo Pygame.

2.1. Sistema Principal e Estrutura

O motor do jogo segue o padrão de um loop principal (game loop), que orquestra as operações essenciais a cada quadro (frame):

1. **Processamento de Eventos:** Captura de entradas do usuário (teclado, mouse).
2. **Atualização Lógica (Update):** Modificação do estado dos objetos do jogo (posição, vida, IA, física).
3. **Renderização (Draw):** Desenho dos elementos visuais na tela.

Adotou-se uma abordagem de arquitetura modular, dividindo o código em classes e módulos Python distintos. Essa separação de responsabilidades simplifica o desenvolvimento, o teste e a depuração. As principais abstrações incluem entidades (personagem, inimigos), itens coletáveis, projéteis, sistemas gráficos (sprites, animações) e a lógica de geração de níveis.

2.2. Geração Procedural de Níveis

Um dos pilares do jogo é a geração procedural de níveis, que garante uma nova disposição do mundo a cada partida, aumentando a rejogabilidade. Algoritmos de ruído, como Perlin Noise ou Simplex Noise, são empregados para gerar mapas de altura ou mapas de características, que determinam a topografia do terreno e a distribuição de diferentes zonas ou biomas:

- Zonas Industriais (fábricas, refinarias, etc.)
- Zonas Urbanas (ruínas)

- Zonas Radioativas (perigo ambiental)
- Áreas Naturais (florestas, lagos)

Estes algoritmos produzem valores pseudoaleatórios, mas coesos espacialmente, ideais para criar características naturais como montanhas, vales ou a distribuição de recursos. Sobre este terreno base, estruturas como edifícios, paredes, tanques e outros elementos são posicionados, seja de forma procedural ou a partir de modelos pré-definidos (prefabs), para adicionar complexidade visual e tática ao ambiente. A representação do nível pode ser baseada em tiles (grade) para facilitar colisões e posicionamento.

3. Personagem Principal e Mecânicas

As mecânicas associadas a Kael foram projetadas para oferecer controle preciso e engajador, integrando movimento, combate e sistemas de estado.

3.1. Movimentação e Física

O controle de Kael utiliza um sistema de física 2D simplificado. A movimentação não é instantânea; em vez disso, aplica-se uma força de aceleração quando o jogador pressiona uma tecla de movimento. A velocidade aumenta gradualmente até um limite máximo. Ao soltar a tecla, a fricção do ”solo”desacelera Kael até parar. Parâmetros como força de aceleração, velocidade máxima e coeficiente de atrito são ajustáveis para balancear a responsividade. Diferentes tipos de terreno (definidos pelos tiles do mapa) podem modificar esses parâmetros, tornando o movimento mais lento em água ou lama, por exemplo.

3.2. Sistema de Combate

O combate oferece opções táticas ao jogador:

- **Corpo a Corpo:** Um ataque rápido de curto alcance, útil para conservar munição ou lidar com inimigos que se aproximam demais. Pode envolver uma animação de ataque e uma área de efeito temporária.
- **Combate à Distância:** Utiliza a pistola Beretta M9. Este sistema inclui gerenciamento de munição (contagem finita de balas), um tempo de recarga após esvaziar o pente, e um cursor de mira controlado pelo mouse para direcionar os disparos.

Os projéteis são entidades independentes com velocidade e trajetória próprias. O sistema de detecção de colisão verifica impactos com inimigos e elementos do cenário. Feedback visual, como clarões de disparo, partículas de impacto e ejeção de cartuchos, reforça a ação.

3.3. Animações e Estados do Personagem

Um sistema de animação baseado em spritesheets (folhas de sprites) é usado para visualizar as diferentes ações de Kael (parado, correndo em 8 direções, atirando, atacando corpo a corpo, recebendo dano). Uma máquina de estados finitos pode gerenciar a transição entre essas animações de forma coesa. Kael possui atributos de estado como:

- **Vida (Health Points):** Reduzida ao receber dano; chegar a zero resulta em game over.
- **Nível de Radiação:** Aumenta progressivamente em zonas radioativas. Níveis altos podem causar dano contínuo, reduzir a velocidade ou afetar a visão (efeitos de pós-processamento simples). A Máscara Reforçada mitiga ou anula esse efeito temporariamente.

- **Invencibilidade Temporária:** Um curto período (e.g., 1 segundo) após sofrer dano, durante o qual Kael não pode ser ferido novamente, sinalizado visualmente (piscando o sprite).

4. Inimigos e Inteligência Artificial

Os adversários em "Beyond the Dome"são controlados por scripts de IA que definem seus comportamentos básicos, criando desafios reativos.

4.1. Tipos de Inimigos

Os inimigos implementados incluem:

- **Saqueadores:** Equipados com armas brancas ou de fogo simples. Podem patrulhar áreas ou guardar locais específicos.
- **Cães Selvagens:** Unidades rápidas focadas em detecção e ataque corpo a corpo rápido. Sua velocidade os torna uma ameaça diferente dos saqueadores.

4.2. Comportamento (IA)

A IA utiliza uma combinação de estados e regras simples:

- **Patrulha:** Inimigos se movem entre pontos pré-definidos (waypoints) ou vagueiam aleatoriamente dentro de uma zona designada. Pode incluir pausas e mudanças de direção para simular vigilância.
- **Detecção:** Utiliza verificação de linha de visão (line-of-sight) e/ou um raio de detecção. Se Kael entra na área de detecção e está visível, o inimigo transita para o estado de perseguição. Ruídos feitos pelo jogador (tiros) também podem alertar inimigos próximos.
- **Perseguição:** O inimigo se move em direção à última posição conhecida de Kael. Algoritmos simples de pathfinding (como A*, se o cenário for complexo) podem ser usados para navegar obstáculos, ou uma abordagem mais direta se o terreno for aberto.
- **Ataque:** Ao se aproximar o suficiente de Kael, o inimigo inicia seu padrão de ataque (disparar, avançar para ataque corpo a corpo). Pode haver um tempo de espera (cooldown) entre ataques.

A IA pode ser aprimorada com comportamentos de fuga se com pouca vida, ou táticas de grupo básicas (flanqueamento simples).

5. Itens e Coletáveis

Itens são essenciais para a progressão e sobrevivência, distribuídos pelo mundo gerado proceduralmente.

- **Módulos de Filtro:** Objetivos principais, talvez encontrados em locais específicos ou protegidos por inimigos mais fortes.
- **Máscaras Reforçadas:** Fornecem imunidade ou resistência à radiação por um tempo limitado.
- **Pacotes de Saúde:** Restauram a vida.
- **Munição:** Coletável para a pistola.

A disposição dos itens pode seguir regras procedurais, garantindo que itens essenciais sejam encontráveis, mas incentivando a exploração para encontrar recursos adicionais.

6. Interface, Câmera e Áudio

Estes sistemas são vitais para a comunicação de informações e a imersão do jogador.

6.1. Interface do Usuário (HUD e Menus)

O HUD exibe continuamente a vida, nível de radiação e munição. Pode usar barras, ícones ou números. Menus para iniciar o jogo, ver a introdução, pausar e lidar com o fim de jogo (game over) são implementados usando os recursos de desenho de texto e imagem do Pygame. O cursor de mira para a pistola é um sprite personalizado que segue o ponteiro do mouse.

6.2. Sistema de Câmera

A câmera 2D segue Kael, mantendo-o geralmente no centro da tela. A interpolação linear (lerp) é usada para suavizar o movimento da câmera, calculando sua nova posição como uma fração da distância entre sua posição atual e a posição-alvo (Kael). Isso evita saltos abruptos. O culling garante que apenas os tiles e sprites visíveis na janela da câmera sejam desenhados, uma otimização crucial para níveis grandes.

6.3. Sistema de Áudio

Utilizando o módulo ‘pygame.mixer‘, o sistema de áudio gerencia:

- **Sons Ambientais:** Loops de áudio de fundo que mudam conforme a zona (vento, ruído industrial, silêncio tenso). Sons de passos que variam com o terreno.
- **Efeitos Sonoros (SFX):** Sons curtos e pontuais para ações (tiros, recarga, coleta de item, dano recebido, passos de inimigos, alertas).

O mixer permite controlar o volume global e individual dos sons, tocar múltiplos sons simultaneamente em canais diferentes e gerenciar a reprodução de música de fundo.

7. Aspectos Técnicos e Otimizações

Garantir um desempenho fluido, especialmente com geração procedural e múltiplos inimigos, requer atenção a aspectos técnicos.

7.1. Física e Colisões

A detecção de colisão é majoritariamente realizada usando os retângulos (‘pygame.Rect‘) associados aos sprites. Funções como ‘pygame.sprite.spritecollide()‘ ou ‘pygame.sprite.groupcollide()‘ são eficientes para verificar colisões entre um sprite e um grupo, ou entre dois grupos. Para colisões com o cenário (baseado em tiles), verifica-se a colisão do retângulo do jogador/inimigo com os tiles marcados como sólidos na vizinhança. A resposta à colisão normalmente envolve reposicionar a entidade para fora do objeto colidido.

7.2. Sistemas de Partículas

Sistemas de partículas simples são implementados para adicionar polimento visual. Cada sistema (emissor) gera múltiplas partículas (pequenos sprites ou formas) com propriedades como posição inicial, velocidade, cor, tempo de vida e talvez física básica (gravidade). São usados para sangue, faíscas, fumaça, efeitos de radiação visual, etc. Gerenciar muitas partículas pode impactar o desempenho, exigindo otimizações (reutilização de partículas, limitação do número).

7.3. Otimização de Renderização

O culling da câmera é a principal otimização de renderização. Usar ‘pygame.sprite.Group‘ e suas subclasses otimizadas ajuda a gerenciar e desenhar grandes números de sprites eficientemente. Evitar cálculos complexos ou carregamento de arquivos dentro do loop principal também é fundamental para manter uma taxa de quadros (FPS) estável.