SISTEMA DE INVENTÁRIO PARA JOGOS INDIE COM SQLITE E GODOT ENGINE: aplicação em gênero ARPG

Henrique de Carvalho Meirelles
Graduando em Banco de Dados pela Fatec Bauru
henrique.meirelles@fatec.sp.gov.br

João Victor Veronez Martins
Graduando em Banco de Dados pela Fatec Bauru
joao.martins43@fatec.sp.gov.br

Rodrigo Garcia de Carvalho
Graduando em Banco de Dados pela Fatec Bauru
rodrigo.carvalho19@fatec.sp.gov.br

Alexandre Galvani
Doutor em Mídia e Tecnologia e Docente na Fatec Bauru
alexandre.galvani@fatec.sp.gov.br

RESUMO

Action Role-Playing Game (ARPG) ou Jogo de Interpretação de Papéis de Ação é um subgênero derivado do famigerado Role-Playing Game (RPG). O tipo de jogo em que os participantes assumem personagens, criam narrativas colaborativamente e podem batalhar contra criaturas em tempo real. Seu precursor surgiu na década de 70, em registros oficiais, com o lançamento de Dungeons & Dragons. Uma mecânica fundamental e amplamente utilizada é o inventário: projetado para armazenar, organizar e exibir itens de um personagem in-game. O material disponível (em português) no repositório oficial do Godot Engine não abrange de forma clara a construção da tal. Portanto, este trabalho tem por objetivo apresentar o desenvolvimento de um sistema de inventário para jogos digitais focado em ARPG. Para validar a proposta optou-se pelo Framework Kanban como metodologia, além de títulos com diferentes abordagens sobre a temática, elencados para análise e comparação de pontos relevantes. A base de dados relacional foi previamente armazenada em SQLite, com possibilidade de migração para arquivo(s) no formato JSON. A mudança é proposital para tornar o projeto altamente escalável, considerando a distribuição posterior em diversas plataformas: Web, Android, iOS, Linux, MacOS e Windows. O resultado obtido consiste em um produto completamente funcional e flexível o suficiente para ser reutilizado. Conclui-se que a ideia da adoção do JSON mostrou ser a melhor decisão durante a prototipagem, por se tratar de um formato leve de troca de informações e pela praticidade em manter as tabelas com um simples editor de código.

Palavras-chave: Sistema de Inventário; Jogos Indie ARPG; SQLite; Godot Engine; Framework Kanban.

1 INTRODUÇÃO

Independente de plataforma, o brasileiro é um grande jogador. A Pesquisa Game Brasil (PGB) em sua 8ª edição entrevistou 12.498 pessoas no Brasil, em 26 estados e no Distrito Federal, entre os dias 7 e 22 de fevereiro de 2021. Dados publicados apontam que 72% da população do país tem o costume de jogar jogos eletrônicos, reforçando este hábito em seu cotidiano.

De fato, desenvolver jogos é extremamente recompensador. Entretanto, é um grande desafio que requer planejamento, habilidades técnicas distintas e comprometimento. Para Novak (2015, p. 340-352) o processo possui as seguintes fases: conceito, pré-produção, protótipo, produção, alfa, beta, ouro e pós-produção. Adotadas por todos os gêneros e com variantes.

O Role-Playing Game (RPG) é um gênero bastante abrangente. Dentre as modalidades existentes destacam-se: o de mesa, o tático, o de jogador único e o massivo para multijogadores, assim como o Action Role-Playing Game (ARPG).

Uma mecânica fundamental é o inventário, entretanto, pouco se discute a respeito de sua construção, consequentemente há escassez de material (em português). Seguindo essa linha, o presente trabalho aborda de forma objetiva o desenvolvimento de um sistema de inventário para jogos digitais ARPG com tecnologias totalmente gratuitas e de código aberto.

2 REFERENCIAL TEÓRICO

Este capítulo contextualiza os temas: jogos digitais independentes, gêneros e subgêneros, motores de jogos e sistema de inventário, detalhando-o.

2.1 Jogos digitais

Jogo digital é um termo referente a jogos projetados para serem jogados em dispositivos eletrônicos. Em que há interação entre humano e computador. Para JUUL (2005) são conteúdos de entretenimento que usam variedades de tecnologias para sua apresentação.

2.1.1 Gêneros

Atualmente podemos encontrar mais de 100 tipos de jogos. Segundo Novak (2011) os gêneros mais comuns são: 1. Ação: *First and third person shooter* (FPS e TPS), *Shoot'em up (Shmup)*, *Survival Horror*, *Stealth Action*, Plataforma; 2. Aventura: *Open World*, *Point and Click*, Aventura Textual, Roguelike; 3. Luta: 2D, 3D, *Beat'em up*, *Hack & Slash*; 4. Corrida: Kart, Futurista, *Arcade*, Simulação; 5. *Role-Playing*: Oriental (JRPG), Estratégico, Ação (ARPG), Dungeon Crawl; 6. Simulação: Civilização, Avião, Trem, F1, Parque de diversões, Animal; 7. Estratégia: Estratégia baseada em turnos (TBS), Estratégia em tempo real (RTS), *Tower Defense*, RPG Tático, *Multiplayer Online Battle Arena (MOBA)*; 8. Esportes: Futebol, Basquete, Beisebol, *Skate*, *Golf*, Tênis; 9. *Parlor*: Educativo, *Microgame* ou minigame, Musical, *Puzzle*; 10. *Massive Multiplayer Online (MMO)*: MMORPG, MMOFPS, MMORTS.

2.1.2 Independência

Jogos independentes são produzidos por uma pequena equipe ou por um indivíduo, sem o auxílio de empresas publicadoras. Possuem temática, estética e distribuição diferenciada dos demais jogos dentro do mercado (KOVANTO, 2013).

2.1.3 Interação

O jogador visualiza estatísticas básicas (vida, mana, experiência) de seu personagem por meio de uma área de exibição, denominada Heads-Up Display (HUD) e pode interagir com o jogo de duas formas: 1. User Interface (UI): periféricos diversos como cliques do mouse, entradas no teclado, controles com e sem fio e touch; 2. Graphical User Interface (GUI): componentes visuais que “invadem” o espaço do jogo, como textos, botões, menus, entre outros (NOVAK, 2011).

2.2 Motores de jogos

Um motor é uma ferramenta de desenvolvimento de jogos digitais. Oferece “blocos de construção” que estruturam objetos. Pode trabalhar com arquivos, gráficos 2D e 3D, animações, detectar colisões, sons e inteligência artificial (NOVAK, 2011). Destacam-se: Godot Engine, Construct, GameMaker Studio, Unity e Unreal.

2.3 Sistema de Inventário

A mecânica de inventário é amplamente utilizada, projetada para armazenar, organizar e exibir itens de um personagem: a mochila é o símbolo mais comum.

2.3.1 Origem

O primeiro jogo a implementar um sistema de inventário é incerto. Dois fortes candidatos são Hamurabi (desenvolvido por Doug Dyment e publicado pela Creative Computing em 1968) ou The Oregon Trail (desenvolvido por Don Rawitsch, Bill Heinemann e Paul Dillenberger e produzido pelo Minnesota Educational Computing Consortium em 1971). Para ambos, uma representação textual do acesso aos itens: matérias-primas, armas e armaduras. Jeannie Novak apresenta o jogo Colossal Cave (desenvolvido por Will Crowther e Don Woods e publicado pela CRL em 1976) em seu livro (NOVAK, 2011).

2.3.2 Funções

A adoção de um inventário ocorre pelos jogos com grande movimentação de itens pela mão do jogador. A mecânica simplifica a jornada permitindo armazenar diversos itens, evitando repetir caminhos para coletar o que ficou para trás. O pensamento estratégico é muito valorizado em sua gestão, implicando na escolha de itens de acordo com a necessidade atual (NOVAK, 2011).

2.3.3 Limitações

Um grande obstáculo durante a implementação é que o desenvolvedor não consegue determinar quantos itens o jogador possuirá em um determinado momento, e o jogador não consegue afirmar quantos itens serão necessários para concluir o jogo. Além disso, existem vários tipos de jogadores: os cautelosos que relutam em usar seus itens, porque acreditam que serão necessários no futuro, principalmente para completar missões. Os mais apressados gastam-nos rapidamente e depois percebem o erro cometido. Outros trocam seus itens por dinheiro ou materiais, sem levar em conta o progresso da jornada. Para fornecer um nível adequado de desafio ao jogador, limitar o inventário é extremamente necessário (NOVAK, 2011).

2.3.4 Modelos

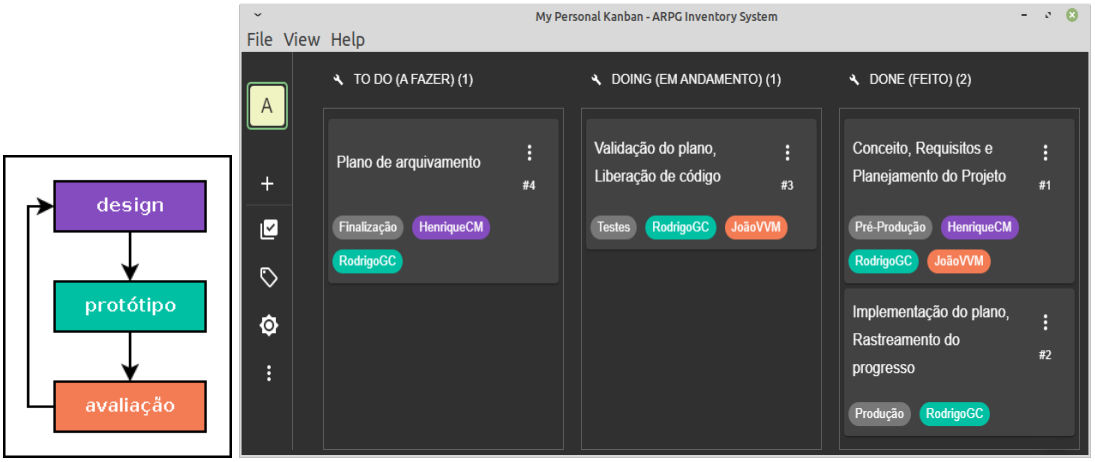
Esta seção apresenta os modelos de sistema de inventário e particularidades: 1. Regra dos 99 (noventa e nove): itens podem ser armazenados com limite fixo por *slot*, tipicamente 99. A limitação pode ser alterada; 2. Ponderado: cada item recebe um valor numérico que representa seu peso, gerando efeitos negativos no personagem conforme a carga aumenta; 3. Grade Visual: itens de estoque ocupam campos de acordo com seu tamanho. Gerando a necessidade de organização; 4. Grade Empilhável: alguns itens podem ser empilhados e recebem um indicador numérico para representar sua quantidade; 5. Grande Slot: todo o inventário é composto por um único campo. Pode herdar características do modelo ponderado; 6. Inventário rápido: uso de itens pré-selecionados por meio de atalhos; 7. Expansão de inventário: considerada uma mecânica de progressão que permite escalar o inventário.

3 MATERIAIS E MÉTODOS

Este capítulo apresenta a metodologia e as tecnologias adotadas. A metodologia escolhida foi o Kanban (MOURA, 2007), em conjunto com o Desenvolvimento Iterativo (NOVAK, 2011). Todos os programas utilizados são gratuitos: 1. DIA Diagram Editor: cria diagramas (DIA DIAGRAM EDITOR, 2022); 2. LibreSprite: permite trabalhar com arte em pixels 2D (LIBRESPRITE, 2022); 3. SFXR: gerador de efeitos sonoros básicos e personalizados (SFXR, 2022); 4. SQLite: biblioteca que implementa um BD embutido (SQLITE, 2022); 5. Structured Query Language (SQL): linguagem de pesquisa declarativa padrão para bancos de dados relacionais (SQL, 2022); 6. DB Browser for SQLite (DB4S): GUI para SQLite (DB4S, 2022); 7. Godot Engine: motor multiplataforma repleto de recursos (GODOT, 2022); 8. GDNative: possibilita executar código C (GODOT DOCS, 2022); 9. GDScript: linguagem própria do motor (GODOT DOCS, 2022); 10. Godot SQLite: plugin para usar SQLite (GODOT ASSET LIBRARY, 2022); 11. JSON: formato leve de intercâmbio de dados entre sistemas (JSON, 2022); 12. Geany: editor de código estável e leve (GEANY, 2022).

O desenvolvimento de um jogo assemelha-se mais ao desenvolvimento de produtos de software do que ao de outras formas de entretenimento. Segundo Novak (2011, p. 354) o modelo de desenvolvimento iterativo “incorpora um processo cíclico em três estágios: design, protótipo e avaliação [...]”. Segue:

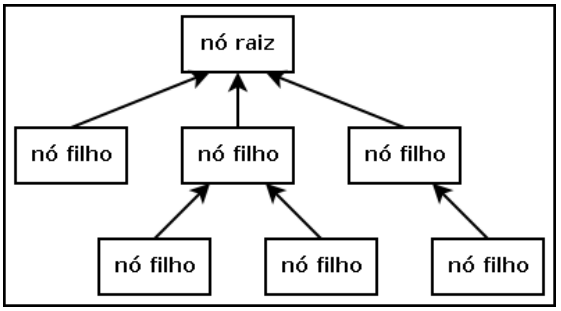
Figura 1 - Diagrama do desenvolvimento iterativo e Configuração do My Personal Kanban



Fonte: Elaborado pelos autores

À esquerda (Figura 1) os estágios do processo. Após a fase de design (que abrange o planejamento e a pré-produção), um protótipo é desenvolvido. Logo que o protótipo é executado e testado durante a fase de avaliação, a equipe decide o que funciona e o que não funciona e volta à fase de design para modificar o protótipo (disponibilizando-o novamente para testes). O processo é repetido indefinidamente, conforme a necessidade, até a conclusão do projeto: a chave desse modelo é o refinamento/aprimoramento contínuo. À direita (Figura 1) a configuração do programa My Personal Kaban com a equipe e tarefas ao final do projeto: em cada cartão há detalhes (não exibidos) sobre o desenvolvimento do projeto. O motor de jogo Godot possui um conceito interessante - e poderoso - de árvore (GODOT DOCS, 2022), onde cada nó pode conter mais e mais nós filhos (Figura 2).

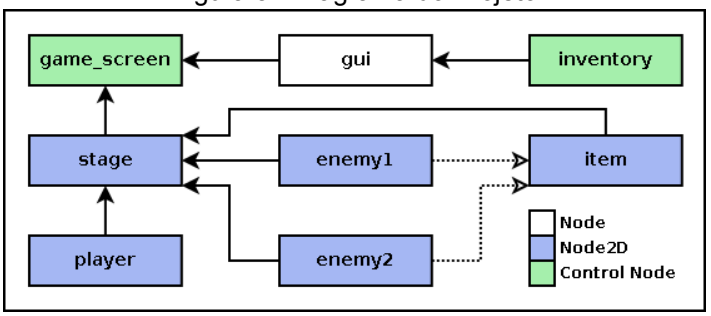
Figura 2 - Conceito de árvore



Fonte: Elaborado pelos autores

A Figura 3 apresenta o diagrama do projeto (simplificado) que contém os nós necessários para sua construção. O fluxo das setas indica que a cena principal é a *game_screen*, enquanto as demais compõem a estrutura.

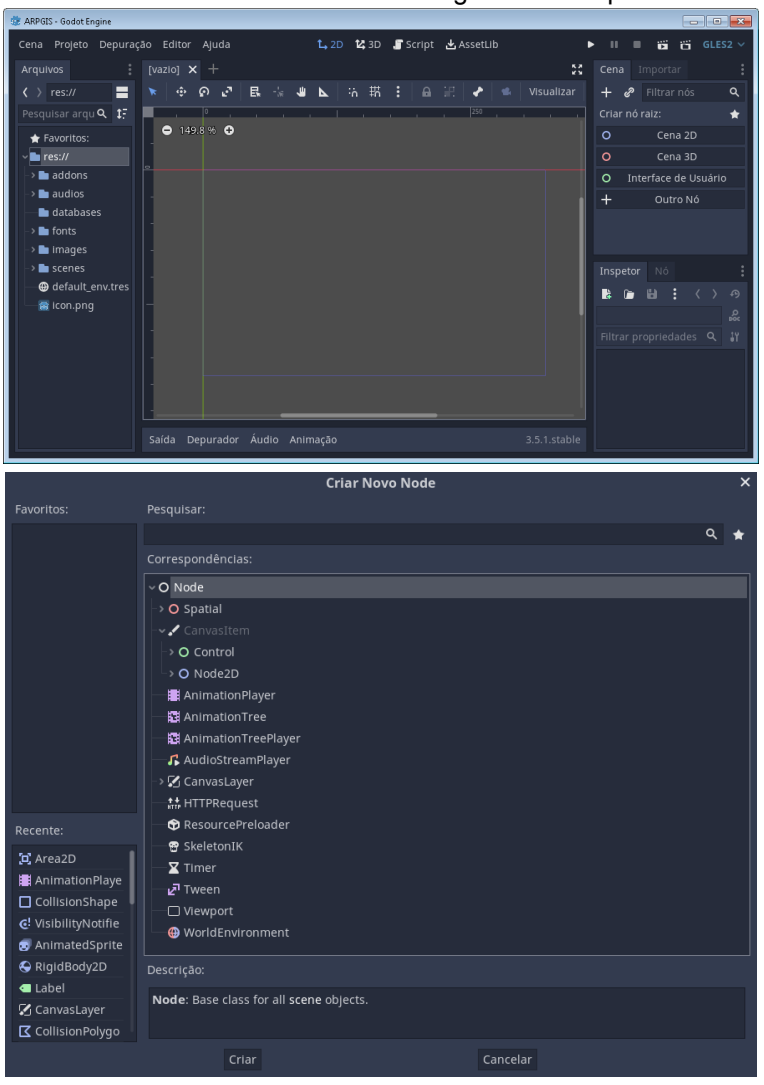
Figura 3 - Diagrama do Projeto



Fonte: Elaborado pelos autores

A Figura 4 exibe a interface do Godot e seu editor de jogos em duas dimensões. No lado esquerdo é possível visualizar o diretório “res://”, que contém todas as pastas criadas para o projeto. Na pasta *addons* temos o *plugin* Godot SQLite; Na pasta *databases* temos as bases de dados, neste caso o arquivo de banco de dados relacional “arpgis.db”; Na pasta *scenes* temos as cenas, que nada mais são do que conjuntos de componentes menores, os nós. Na parte inferior é apresentada uma lista com vários tipos de nós, cada qual com sua função.

Figura 4 - Interface e Editor 2D do Godot Engine - Correspondências de nós



Fonte: Elaborado pelos autores

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta o projeto do sistema de inventário desenvolvido de acordo com os procedimentos metodológicos adotados.

4.1 Design

A Figura 5 apresenta o esboço do sistema de inventário com a disposição geral dos elementos. O lado esquerdo conta com a barra de vida, *slots* do item e subitem principal e o botão para abrir o inventário. O lado direito conta com os *slots* (ou espaços) dos itens e subitens e o botão para fechá-lo.

Figura 5 - Esboço do Sistema de Inventário

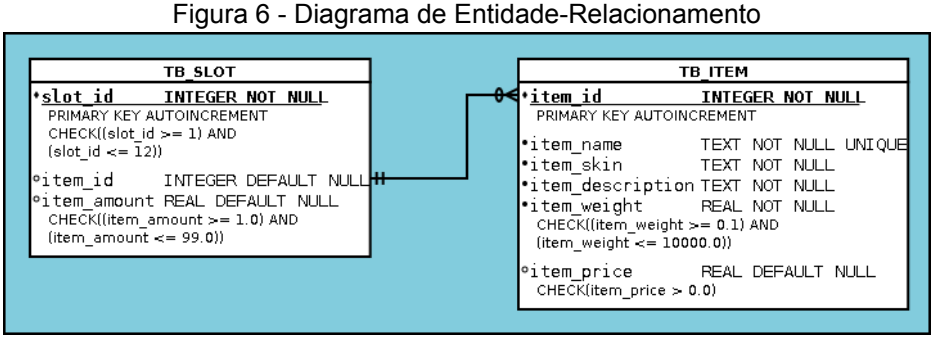
Fonte: Elaborado pelos autores

4.2 Prototipagem

No Diagrama de Entidade-Relacionamento (DER) é apresentada a relação entre as tabelas *tb_slot* e *tb_item*, onde um e somente um espaço do inventário pode conter zero ou “n” itens (Figura 6). Existem restrições de dados para algumas colunas como: *slot_id* em relação ao número máximo de espaços que inventário possui (intervalo de 1 a 12), *item_amount* em relação a quantidade de um item (intervalo de 1.0 a 99.0), *item_name* em relação ao nome que um item recebe (deve ser único), *item_weight* em relação ao peso do item (intervalo de 0.1 a 10000.0), *item_price* em relação ao preço do item (acima de 0.0).

A notação utilizada para representar a cardinalidade entre as tabelas é conhecida como Notação James Martin ou informalmente como “pé de galinha”.

Figura 6 - Diagrama de Entidade-Relacionamento



Fonte: Elaborado pelos autores

Com base no diagrama os comandos SQL foram programados (Figura 7). Para melhor interpretação a equipe comentou todo o código. Também há trechos com diferentes cores onde: a cor laranja representa as palavras reservadas (*CREATE TABLE* e etc). A cor branca representa o nome da tabela e colunas, juntamente dos parênteses, vírgulas e dos operadores de comparação. A cor amarela representa o tipo primitivo de cada coluna. A cor vermelha representa os valores (inteiros e pontos flutuantes). A cor verde representa *strings* (textos) e comentários. O padrão dos comentários segue dois traços, um espaço, texto minúsculo e sem acentuação.

O sistema de inventário depende de outros comandos para funcionar corretamente. Os mais relevantes são apresentados abaixo:

Figura 7 - Código SQL

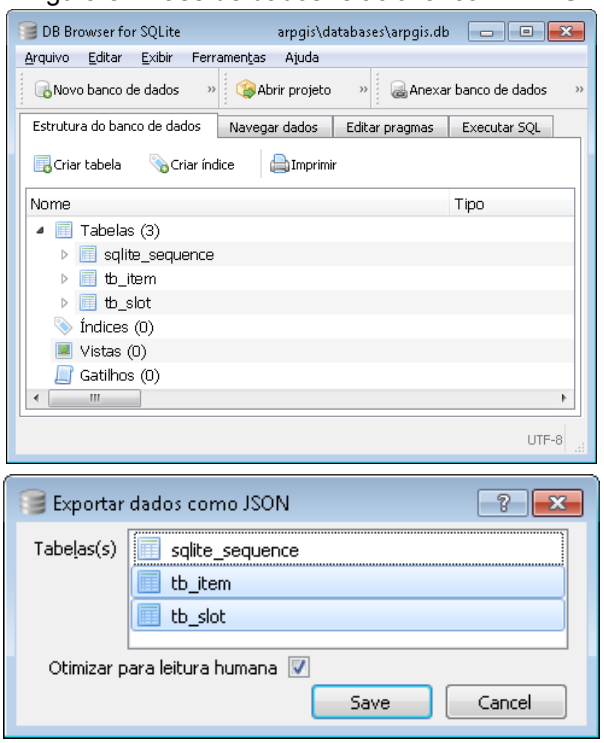
```sql
-- comando para criar a tabela item
CREATE TABLE IF NOT EXISTS tb_item (
	item_id INTEGER NOT NULL,
	item_name TEXT NOT NULL UNIQUE,
	item_skin TEXT NOT NULL,
	item_description TEXT NOT NULL,
	item_weight REAL NOT NULL
		CHECK((item_weight >= 0.1) AND (item_weight <= 10000.0)),
	item_price REAL DEFAULT NULL CHECK(item_price > 0.0),
	PRIMARY KEY (item_id AUTOINCREMENT)
);


-- comando para criar a tabela inventario
CREATE TABLE IF NOT EXISTS tb_slot (
	slot_id INTEGER NOT NULL
		CHECK((slot_id >= 1) AND (slot_id <= 12)),
	item_id INTEGER DEFAULT NULL,
	item_amount REAL DEFAULT NULL
		CHECK((item_amount >= 1.0) AND (item_amount <= 99.0)),
	PRIMARY KEY (slot_id AUTOINCREMENT),
	FOREIGN KEY (item_id) REFERENCES tb_item(item_id)
);


-- comando para inserir um espaco no inventario - executado 12 vezes
INSERT INTO tb_slot DEFAULT VALUES;


-- comando para inserir um item
INSERT INTO tb_item VALUES (1001, "short sword",
	"res://images/items/skins/short_sword.png", "a short iron sword", 32.0, 128.0);


-- comando para atualizar o inventario no decimo segundo espaco
UPDATE tb_slot SET item_id = 1001, item_amount = 1.0 WHERE slot_id = 12;
```

Fonte: Elaborado pelos autores

Estes foram executados com auxílio da ferramenta DB Browser for SQLite (DB4S) de acordo com a Figura 8. Primeiro criou-se as tabelas *tb_item* e *tb_slot*. Em seguida ocorreu a inserção dos dados em cada tabela alternando entre os comandos *INSERT*. Por fim, as modificações foram escritas no banco (tornaram-se permanentes no arquivo com extensão .db). Há também o comando *UPDATE* para atualizar um item específico.

Na parte inferior da figura nota-se uma janela cuja função é exportar os dados como JSON. Ao clicar no botão *Save*, uma caixa de diálogo abriu para definir o local onde os arquivos podiam ser salvos.

Figura 8 - Base de dados relacional com DB4S



Fonte: Elaborado pelos autores

A Figura 9 exibe os arquivos *items.json* e *slots.json*, respectivamente, após exportar para o formato JSON.

Figura 9 - Código JSON

```
[
    {
        "_items_comment1_": "exemplo de um item - items.json",
        "item_id": 1001,
        "item_name": "short sword",
        "item_skin": "res://images/items/skins/short_sword.png",
        "item_description": "a short iron sword",
        "item_weight": 32.0,
        "item_price": 128.0
    }
]

[
    {
        "_slots_comment1_": "primeiro espaco do inventario - slots.json",
        "inventory_slot_id": 1,
        "item_id": 1001,
        "item_amount": 1.0
    },
    {
        "_slots_comment2_": "segundo espaco do inventario - slots.json",
        "inventory_slot_id": 2,
        "item_id": null,
        "item_amount": null
    }
]
```

Fonte: Elaborado pelos autores

A Figura 10 apresenta um exemplo de código utilizando o *plugin* Godot SQLite e a linguagem GDScript e SQL.

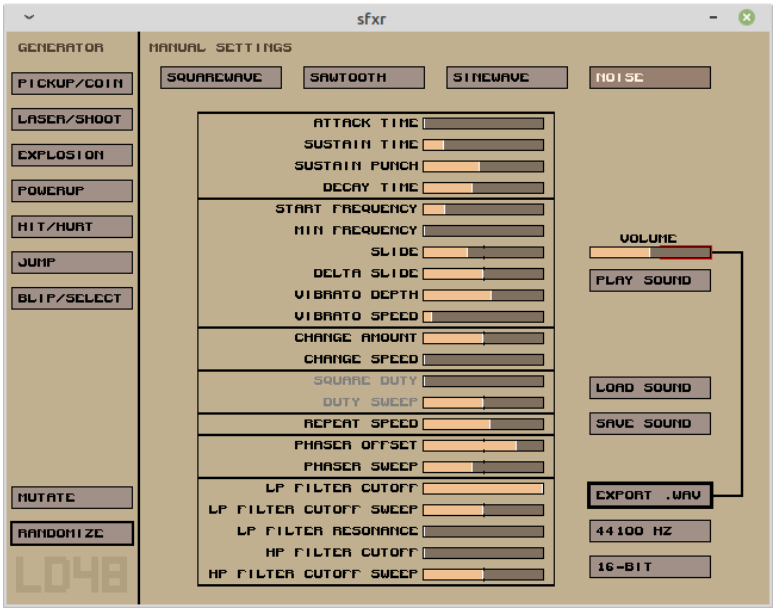
Figura 10 - Código GDScript e SQL

```
extends Node


# classe customizada SQLite
const SQLITE_ADDON = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")
var database_path: String = "res://databases/arpgis"


func query(command_sql: String) -> void:
	# metodo para consultar com codigo SQL
	var database
	var database_query_result: String
	database = SQLITE_ADDON.new()
	database.path = database_path
	database.verbose_mode = true
	database.open_db()
	database.query(command_sql)
	database_query_result = database.query_result
	database.close_db()
	return database_query_result


func _ready() -> void:
	# printa na tela a consulta de todos os itens
	print(query("SELECT * FROM tb_item;"))
```

Fonte: Elaborado pelos autores

A Figura 11 exibe o programa SFXR, um simples gerador de efeitos sonoros.

Figura 11 - Gerador de efeitos sonoros



Fonte: Elaborado pelos autores

4.3 Avaliação

Nesta etapa todos os códigos foram devidamente testados e liberados.

4.4 Finalização

A Figura 12 exibe o resultado final do projeto, com ajustes na paleta de cores para trazer harmonia. O sistema de inventário aceita comandos via teclado, apertando a tecla “i” para acessar o inventário, além de cliques e arraste do mouse

para pegar e soltar os itens nos espaços vazios ou na tela, para derrubá-los quando não houver mais serventia. Ao matar monstros há uma chance de derrubar novos itens para progredir na jornada.

Figura 12 - Sistema de Inventário

Fonte: Elaborado pelos autores

5 CONSIDERAÇÕES FINAIS

Os jogos digitais são parte integral da vida de muitos, principalmente dos mais jovens. Trata-se de um segmento em constante crescimento que proporciona “oportunidades de ouro”. Para incentivar a exploração de soluções acessíveis, optou-se por ferramentas gratuitas.

Os dados foram armazenados em SQLite com o DB4S. Porém, ao considerar a implementação do sistema de inventário em jogos com modo online e posterior distribuição deste em diversas plataformas, parte de uma solução para banco de dados distribuídos fez-se necessária, onde novas versões do sistema podem vir a trabalhar com MongoDB. Desta forma ocorreu a transferência dos dados para um arquivo em formato JSON.

De acordo com os resultados obtidos, constatou-se que o objetivo do trabalho foi plenamente atendido. Através dos procedimentos metodológicos adotados, a ideia de desenvolver um sistema de inventário simples e escalável se tornou possível. Contribuindo assim com o aprendizado da comunidade brasileira que utiliza o motor de jogo Godot em projetos pessoais ou comerciais, bem como a biblioteca de ativos. Foi preciso omitir informações não tão importantes acerca do desenvolvimento, a fim de apontar a direção para aqueles que desejarem se aprofundar no assunto. Vale ressaltar ainda que os envolvidos continuam a planejar melhorias e novas funcionalidades para o sistema.

6 REFERÊNCIAS

ARPGIS. **A ARPG Inventory System**. V1.0. Bauru - São Paulo. Rogarca, 2022. Disponível em: https://rogarca.itch.io/arpgis. Acesso em: 10 abr. 2022.

DB4S. **The Official home of the DB Browser for SQLite**. Disponível em: https://sqlitebrowser.org. Acesso em: 10 abr. 2022.

DIA DIAGRAM EDITOR. **Versatile diagramming tool**. Disponível em: https://wiki.gnome.org/Apps/Dia. Acesso em: 10 abr. 2022.

GEANY. **The IDE**. Disponível em: https://geany.org. Acesso em: 23 abr. 2022.

GODOT. **The game engine you waited for**. Disponível em: https://godotengine.org. Acesso em: 23 abr. 2022.

GODOT ASSET LIBRARY. **Biblioteca de ativos do Godot**. Disponível em: https://godotengine.org/asset-library/asset. Acesso em: 23 abr. 2022.

GODOT DOCS. **Documentação traduzida do Godot**. Disponível em: https://docs.godotengine.org/pt_BR/stable. Acesso em: 23 abr. 2022.

JSON. **Starting**. Disponível em: https://json.org. Acesso em: 23 abr. 2022.

JUUL, J. **Half-Real: Video Games between Real Rules and Fictional Worlds**. Cambridge, Mass: MIT Press, 2005.

KOVANTO, A. **The Improvements for Indie Game Development**. Joensuu: Karelia University of Applied Sciences, 2013.

LIBRESPRITE. **A program for creating and animating your sprites**. Disponível em: https://libresprite.github.io. Acesso em: 23 abr. 2022.

MANZUR, A.; MARQUES, G. **Godot Engine Game Development in 24 Hours, Sams Teach Yourself: The Official Guide to Godot 3.0**. Indianapolis: Pearson, 2018.

MOURA, R. A. **Kanban: a simplicidade do controle da produção**. 7 ed. São Paulo: Imam, 2007.

MPK. **Software My Personal Kanban by Greg Gigon's**. Disponível em: https://greggigon.com/my-personal-kanban-2-0. Acesso em: 23 abr. 2022.

NOVAK, J. **Desenvolvimento de games**. 2 ed. São Paulo: Cengage Learning, 2011.

PGB. **Pesquisa aponta que 72% dos brasileiros jogam games**. Disponível em: https://pesquisagamebrasil.com.br/pt/ebooks. Acesso em: 09 abr. 2022.

SFXR. **A little tool by DrPetter's**. Disponível em: https://drpetter.se/project_sfxr.html. Acesso em: 10 abr. 2022.

SQL. **Home**. Disponível em: https://sql.org. Acesso em 18 ago. 2022.

SQLITE. **Latest**. Disponível em: https://sqlite.org. Acesso em: 02 ago. 2022.