Pathfinder Helper

Base de dados de gestão de informação do TTRPG Pathfinder_2e

Universidade de Aveiro

Licenciatura em Engenharia de Computadores e Informática

E

Licenciatura em Física

Base de Dados P6G3

Regente: Prof. Carlos Costa

Professor Orientador: Prof. Joaquim Sousa Pinto

João Gomes, 84754

Mohamed Haddadi, 107193

5 de junho 2024

Índice

Introdução

No âmbito da unidade curricular de Base de Dados, da Licenciatura em Engenharia de Computadores e Informática e da Licenciatura em Física foi-nos proposto a criação de um projeto que fizesse a gestão de um sistema funcional e com complexidade razoável aplicável ao mundo real. O seguinte relatório irá descrever sucintamente o nosso projeto dando enfase nas partes que achamos mais fulcrais para o funcionamento do mesmo.

O nosso projeto consiste em desenvolver uma ferramenta para criação de personagens e pesquisa de informação no TTRPG(Tabletop Role-Playing Game): Pathfinder, 2nd Edition.
TTRPGs são experiências imersivas e colaborativas de contagem de histórias.
Um grupo de amigos tomam os papéis de DM(Dungeon Master) ou Players e, em comunidade e interagindo socialmente, com imaginação e criatividade, utilizam um sistema de regras e mecânicas de jogo para contar colaborativamente uma história.
O sistema de Pathfinder tem por base um processo de geração aleatória de números que é obtido através de dados.
Estes dados, em TTRPGs, são essencialmente ferramentas que utilizamos para determinar resultados, resolver ações e adicionar um elemento de imprevisibilidade à jogabilidade.
Utilizam-se dados com 4, 6, 8, 10, 12 e 20 faces, sendo o mais comum o dado de 20 lados, ou o D20.
O nosso objetivo é simplificar o processo de criação de personagens para os Players e facilitar a pesquisa de informação de feitiços para ambos o DM e os Players.

O arquivo enviado em anexo contém o código usado para criar a base de dados e utilizar a mesma no diretório **scripts**. Dentro do diretório **scripts** temos os ficheiro **DDL**, **insert_all** e **SP** que são os ficheiros de criação das tabelas, do preenchimento das mesmas e da criação de Stored Procedures, respetivamente. Preenchemos as tabelas criadas a partir de ficheiros JSON que estão dentro do diretório **json**. Desenvolvemos um WFA em C# e o código relativo à mesma encontra-se no diretório **Interface**.

É necessário instalar a package system.configuration para a comunicação entre o database e a aplicação usar a segurança do Windows.

Há duas maneiras de testar a nossa base de dados:

- Localmente com o servidor local: Deverá executar os ficheiros DDL.sql e insert_all.sql por essa ordem. O DDL para criação das tabelas e indexes e o insert_all para a inserção de dados nas tabela e a criação de algumas tabelas temporárias e funções para ajudar na inserção de dados nas tabelas N:M. Para tal é necessário mudar o diretório descrito em cada um dos scripts de inserção de dados no ficheiro insert_all.sql para o diretório local. Este passo é necessário e não pode ser evitado pois usámos ficheiros JSON para a transferência de dados e população das tabelas. Não é possível usar caminhos do diretório relativos para este passo. No nosso caso temos de mudar este diretório entre C:\Users\moham\Desktop\Uni\3º ano\2º semestre\DB-project\json\Classes_data.json e C:\Users\joaog\Documents\GitHub\DB-project\json\Languages.json enquanto estamos a trabalhar no projeto. O passo de segurança para o acesso à base de dados pela interface é feito nativamente pelo Windows.

```
FROM OPENROWSET
(
    BULK 'C:\Users\joaog\Documents\GitHub\DB-project\json\Spells_data.json', SINGLE_CLOB
) AS datasource;
```

Figura 1 - Exemplo de diretório a ser mudado

- Remotamente com o servidor da UA: Deverá executar os passos relativos ao ficheiro insert_all.sql (a sua execução após a mudança dos paths) e certificar-se que tem acesso/permissão ao uso da função BULK. Para além disso tem que alterar no ficheiro App.config a Password:password para Joao84754+Mohamed107193 na linha comentada. Depois deverá descomentar essa linha e comentar a linha acima. Optámos por este passo para não publicar a nossa password no GitHub.

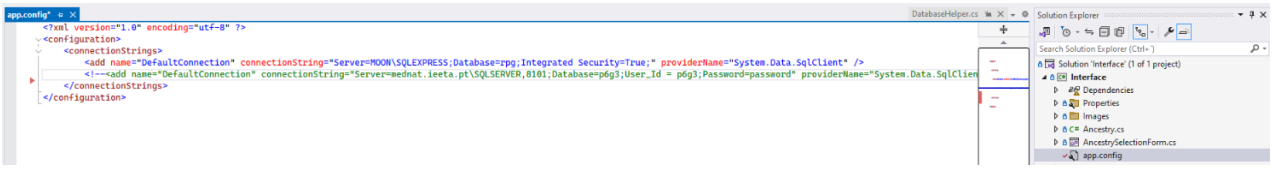

Figura 2 - Ficheiro App.confic onde podemos alterar a password utilizada.

Segue também um vídeo demonstrativo usado para a apresentação dentro do diretório de **vídeos**.

Motivação

Decidimos desenvolver esta base de dados porque, em conjunto com a Unidade Curricular 41549-ihc, queremos criar uma aplicação android para ajudar o nosso grupo de amigos a melhorar a eficiência e jogabilidade das sessões semanais de Pathfinder_2e.

Ao podermos obter facilmente feedback dos utilizadores finais, conseguimos perceber as principais funcionalidades requisitadas pelos mesmos, e assim, concentrar os nossos esforços em desenvolver um produto eficaz e eficiente.

Análise de Requisitos

Podem registar-se utilizadores que vão criar Characters, personagens do jogo, e vão consultar informação sobre o jogo, nomeadamente Feats, Spells e Equipment.

- Existem 2 tipos de User: utilizadores normais e Admin. Os utilizadores Admin podem editar, criar e apagar entradas em algumas tabelas.
- Para um User criar um Character, tem de fazer várias escolhas (nomeadamente Ancestry, Class, Background)
- Escolha de ancestry muda atributos (hp, languages e speed) de cada Character
- Ability scores mudam por causa das escolhas de Ancestry, Background e Class.
- Ability modifiers mudam por causa dos ability scores(Modifier = (Score – 10) / 2)
- Skills, attack rolls, saving throws e HP mudam por causa dos ability modifiers.
- Escolha da class e o nível da class alteram os valores de diferentes proficiencies.
- Escolha da class pode dar acesso ao uso de feitiços. As classes com acesso a feitiços têm uma progressão específica de feitiços.
- Proficiencies alteram os valores de skills, attack rolls e saving throws.

Entidades e multiplicidade

- Cada **utilizador** é identificado por um ID único e possui um nome de utilizador único, uma hash de senha, um salt para a senha, um papel (role) no sistema e a data de criação do registo. O nome de utilizador é único e obrigatório.
- Cada utilizador pode criar vários **Characters** (os personagens do sistema). Cada personagem possui um ID único, Ability Scores, uma Class, uma Ancestry, um Background, modificadores de habilidade (derivados de Ability_scores. Cada Modificador é derivado a partir do seu score respectivo pela fórmula mod = (score – 10) / 2), velocidade (derivada de Ancestry), class DC (derivada de Class), nível (de 1 a 20), nome, pontos de vida (derivados de Class e Ancestry). Algumas personagens ganham acesso a feitiços, dependendo da escolha da sua Class. Cada personagem também tem listas de equipamentos, spells, feats, skills, saving

throws, attack rolls e languages. Os skills, saving throws e attack rolls são derivados a partir da escolha da Class. Languages são derivadas a partir da escolha da Ancestry.

- **Ability_scores** guarda as pontuações das habilidades físicas e mentais dos personagens, que incluem Força, Destreza, Constituição, Inteligência, Sabedoria e Carisma. Cada conjunto de pontuações de habilidades tem um ID único. Estas pontuações são atribuídas automaticamente com valores padrão de 10.
- **Ancestry** armazena as linhagens dos personagens, fornecendo características e habilidades únicas. Cada ancestralidade tem um ID único, nome, pontos de vida, tamanho (pequeno ou médio), velocidade, defeitos e incrementos de habilidade, tipo de visão e raridade. A escolha da ancestralidade também define a Linguagem que o Personagem vai falar.
- **Background** regista os antecedentes dos personagens, representando a sua educação e experiências de vida. Cada background tem um ID único, nome, skills, feats, um bónus num ability score, raridade e uma descrição. A escolha de um Background pode dar acesso ao character a Feats.
- **Class** regista as classes dos personagens, definindo a profissão ou vocação e a progressão das habilidades. Cada classe tem um ID único, nome, pontos de vida por nível, proficiências de ataque e defesa, ability score chave e uma referência opcional à tabela de progressão de feitiços. A própria class tem uma progressão única que depende do nível do Character. As classes que têm uma progressão de feitiços também têm uma Tradition.
- **Spell_progression** regista a progressão dos feitiços para diferentes classes. Inclui o nível do personagem, número de cantrips disponíveis e o número de slots de feitiço para cada nível de 1 a 10.
- **Saving_throws** regista os modificadores de resistência dos personagens, incluindo a proficiência, designação (Reflex, Fortitude, Will, Perception) e valor.
- **Skills** regista as habilidades dos personagens, refletindo o seu treino e perícia em várias tarefas. Cada habilidade possui um ID único, designação e detalhes adicionais. Qualquer Character pode selecionar as mesmas skills.
- **Attack_rolls** regista os modificadores de ataque dos personagens. Cada entrada possui um ID único, um nível de proficiência derivado da classe do personagem e um tipo de ataque, que pode ser desarmado, simples, marcial ou avançado.
- **Equipment** armazena itens e equipamentos que os personagens podem possuir. Cada item possui um ID único, nome, categoria, subcategoria, uso, volume/peso, raridade, categoria da arma, nível necessário e preço. Cada equipamento pode ou não ter um ou mais **Traits** associados. Qualquer Character pode selecionar os mesmos equipamentos.
- **Traits** regista as características associáveis a spells, feats ou equipments. Cada trait possui um ID único, designação e detalhes adicionais.
- **Feats** armazena habilidades especiais ou talentos que os personagens podem adquirir. Cada feito tem um ID único, raridade, pré-requisitos opcionais, um sumário, nome e nível necessário. Cada feat pode ou não ter um ou mais **Traits** associados. Qualquer Character pode selecionar os mesmos feats.
- **Spells** armazena os feitiços que podem ser utilizados por determinadas classes. Cada feitiço possui um ID único, tipo de feitiço, nome, ações necessárias, defesa associada, alvo, raridade, gatilho, área de efeito, nível, detalhes de aumento, duração e alcance. Cada feitiço pode ou não ter um ou mais Traits associados. Alguns feitiços podem ter Tradition diferentes. As classes que têm acesso a magia criam os seus feitiços de maneiras distintas. Qualquer Character pode selecionar os mesmos spells, desde que tenha nível suficiente para os selecionar e que o feitiço pertença à Tradition da sua class.

- **Tradition** regista as tradições mágicas associadas aos feitiços. Cada tradição tem um ID único, nome e detalhes adicionais.
- **Language** regista as línguas que os personagens podem falar. Cada língua é identificada por um ID único e possui uma designação. Várias Ancestries dão acesso à mesma Language. Qualquer Character pode ter a mesma Language selecionada.

Diagrama Entidade Relação (DER)

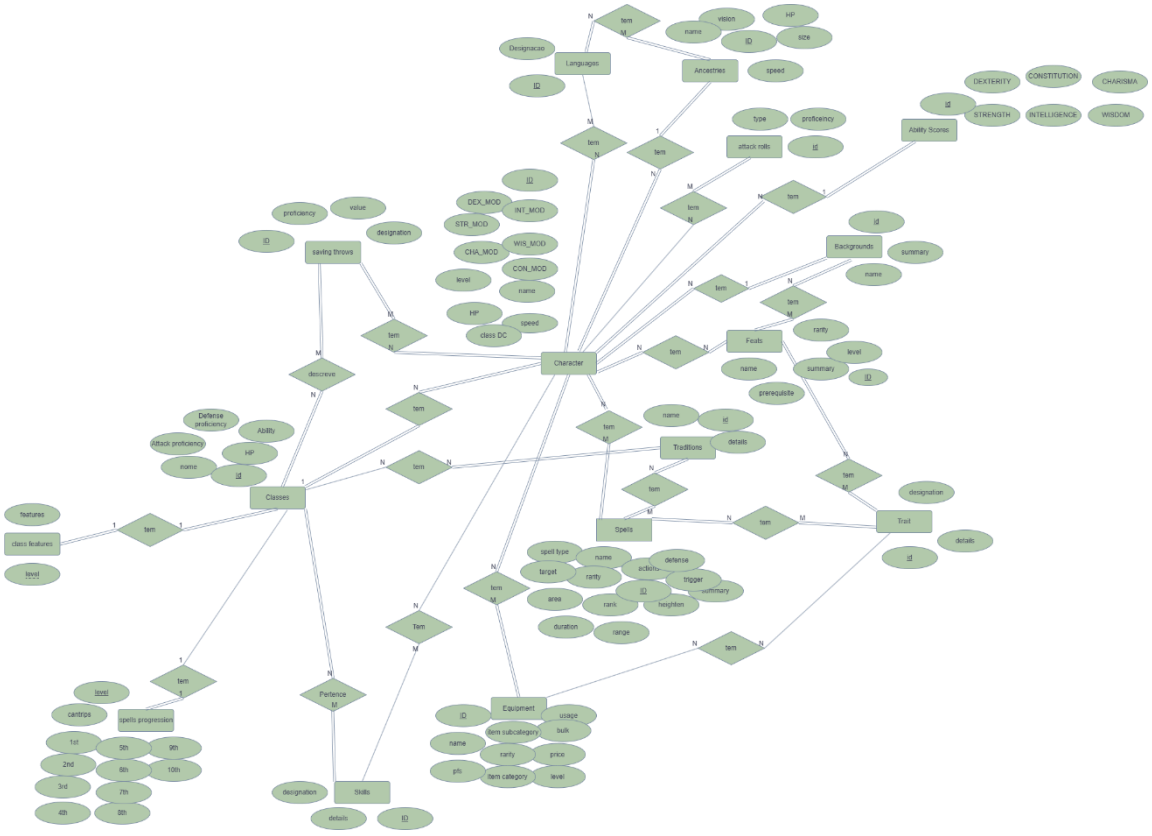

Figura 3 - Diagrama Entidade Relação

Esquema Relacional (ER)

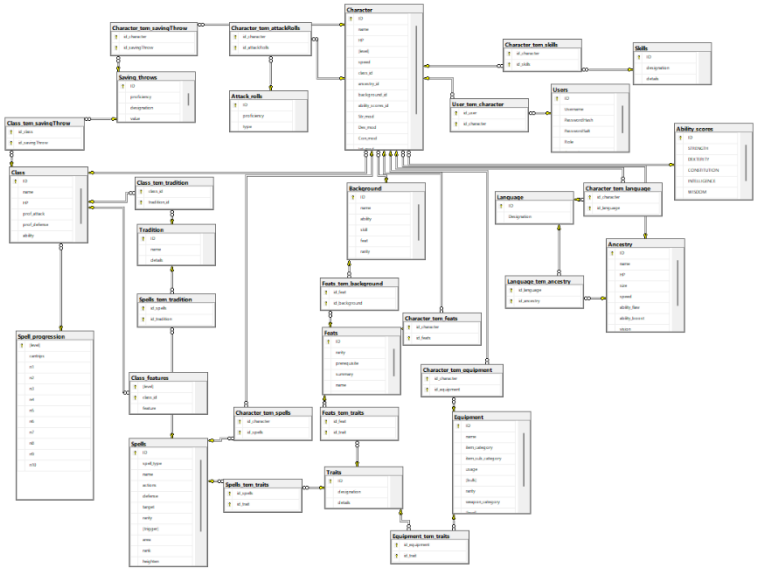

Figura 4 - Esquema Relacional

Stored Procedure (SP)

Criámos uma stored procedure para retirar as Languages a partir da escolha de Ancestry e para obter os valores de Feats, Equipments e Spells da base de dados. Aqui está um excerto do ficheiro SP.sql do diretório scripts.

```
USE rpg
-- Criação da stored procedure
CREATE PROCEDURE GetLanguagesByAncestry
    @AncestryName VARCHAR(28)
AS
BEGIN
    SELECT l.Designation
    FROM Ancestry a
    JOIN Language_tem_ancestry la ON a.ID = la.id_ancestry
    JOIN [Language] l ON la.id_language = l.ID
    WHERE a.[name] = @AncestryName
END

GO

drop procedure GetLanguagesByAncestry
```

Figura 5 – Excerto do ficheiro de criação das Stored Procedures

User Defined Function (UDF)

De forma a preencher a tabela Equipments, precisámos de retirar caracteres não numéricos da coluna price e passar os valores dos diferentes tipos de moedas (cp, sp, gp e pp) para um valor inteiro global que pudéssemos usar para pesquisa, por exemplo.

```sql
CREATE FUNCTION dbo.ConvertPriceToCP (@price VARCHAR(50))
RETURNS INT
AS
BEGIN
    DECLARE @converted_price INT;

    DECLARE @numeric_value VARCHAR(50) = LEFT(@price, PATINDEX('%[a-zA-Z]%', @price) - 1);

    DECLARE @unit_of_measure VARCHAR(50) = SUBSTRING(@price, PATINDEX('%[a-zA-Z]%', @price), LEN(@price));

    SET @converted_price =
        CASE
            WHEN @unit_of_measure = 'pp' THEN CAST(REPLACE(@numeric_value, ',', '') AS INT) * 1000
            WHEN @unit_of_measure = 'gp' THEN CAST(REPLACE(@numeric_value, ',', '') AS INT) * 100
            WHEN @unit_of_measure = 'sp' THEN CAST(REPLACE(@numeric_value, ',', '') AS INT) * 10
            WHEN @unit_of_measure = 'cp' THEN CAST(REPLACE(@numeric_value, ',', '') AS INT)
            ELSE 0
        END;

    RETURN @converted_price;
END;
```

Figura 6 - UDF para converter os diferentes tipos de moedas

```sql
CREATE FUNCTION dbo.RemoveNonNumericChars (@input VARCHAR(50))
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @output VARCHAR(50);
    SET @output = '';

    DECLARE @i INT = 1;
    DECLARE @len INT = LEN(@input);

    WHILE @i <= @len
    BEGIN
        IF SUBSTRING(@input, @i, 1) LIKE '[0-9 ]'
        BEGIN
            SET @output = @output + SUBSTRING(@input, @i, 1);
        END
        SET @i = @i + 1;
    END

    RETURN @output;
END;
GO
```

Figura 7 - UDF para remover caracteres não numéricos

Index

Com vista a iterar mais eficazmente pelos Spells, Feats e Equipments existentes, organizou-se a tabela de Spells pelos nomes, raridade e nível, visto que o acesso mais complexo à mesma verifica os dados por esses atributos. Tomámos essa decisão visto que estas tabelas não iriam ser alteradas, nem iriam ter novos elementos inseridos pelos utilizadores.

```
-- Indexes for the Spells table
CREATE INDEX idx_spells_name ON Spells ([name]);
CREATE INDEX idx_spells_rarity ON Spells (rarity);
CREATE INDEX idx_spells_rank ON Spells ([rank]);

-- Index for the name attribute in the Equipment table
CREATE INDEX idx_equipment_name ON Equipment ([name]);

-- Index for the name attribute in the Feats table
CREATE INDEX idx_feats_name ON Feats ([name]);
```

Figura 8 - Index por data

Discussão

Criar uma interface para a criação de um personagem num jogo mostrou ser uma tarefa bastante complexa. As diferentes e múltiplas implicações entre escolhas na criação da personagem e as várias dependências e derivações de atributos elevaram a complexidade do projeto. A escolha de class deveria dar acesso a diferentes Proficiencies para attack, saving_throws e skills ao longo da progressão do nível do Character, mas não conseguimos criar essa implementação. A escolha de equipamento, no caso de ser da categoria “Weapon” deveria criar uma entrada na tabela Character_tem_attackRolls associada a essa arma escolhida, com base nas várias proficiências de ataque do personagem. Também não foi implementado o atributo de defesa “AC” em Character, que seria derivado a partir da proficiência de defesa da escolha da class e do equipamento (neste caso, da armadura), pois existem vários tipos de proficiência de defesa e vários tipos de armadura e uns usariam o modificador de Destreza, outros teriam um limite no uso de destreza e outros não a usariam.

A tabela de skills não é populada durante a criação da database, e é mais tarde criada durante a criação de Character na interface em C# e enviada para a database.

As tabelas de saving_throws, attack_rolls não foram populadas com valores úteis.

O jogo tem várias restrições e pré-requisitos que não conseguimos implementar. Há vários Feats que não podem ser escolhidos por todas as Ancestries, ou só podem ser escolhidos se o personagem tiver uma certa proficiency num tipo de Attack, Defense, ou Skill, ou até um pré-requisito num Ability Score.

Algo que também não foi implementado foi o uso de Views. Estamos a fazer o acesso direto à Base de Dados e não um acesso a uma interface mediadora através do uso de funções. Este acesso direto pode criar brechas ou quebras na segurança da base de dados e é algo a ser evitado.

Também demos ao Admin a possibilidade de criar novos Spells, Feats e Equipment. Achamos que estas possibilidades não faziam muito sentido no âmbito do jogo, mas queríamos mostrar que o sabíamos fazer.

Conclusão

Podemos concluir que os principais objetivos e as principais funcionalidades da aplicação para a gestão de informação do TTRPG: Pathfinder_2e foram cumpridas. O User pode adicionar Characters, procurar por Spells, Equipments e Feats e um utilizador com privilégios de Admin pode criar Feats, Spells e Equipment e editar/apagar Ancestries e Spells.

Um Character tem atributos derivados de escolhas feitas na sua criação (como as escolhas de Ancestry, Background e Class), e esses atributos vão influenciar valores específicos que seriam utilizados durante uma sessão de jogo com esse personagem a partir dos seus Modificadores (como as skills, HP).

A progressão do nível do Character está definida e tem vários valores a ser utilizados pelo jogador desse personagem, como várias Proficiencies.
Apesar de algumas funcionalidades não estarem implementadas na entrega final, consideramos que temos um produto funcional e útil para um utilizador que queira criar um Character de Pathfinder_2e e procurar por Spells, Feats e Equipments.

Ambos os elementos do grupo concordaram na divisão de trabalho ser atribuída em:
João Gomes – 40%

Mohamed Haddidi – 60%