UNIVERSIDADE FEDERAL DE PELOTAS

YURI DE MELO ZORZOLI NUNES

DUNGEON POO

MARÇO, 2024

1 INTRODUÇÃO

Este relatório descreve o desenvolvimento de um jogo em Java que apresenta uma dinâmica de batalhas entre personagens de diferentes classes. Cada classe possui habilidades específicas e atributos únicos, como força física, agilidade e inteligência, que influenciam seu desempenho em combate. Inicialmente, as classes disponíveis são arqueiro, guerreiro, mago e monstro, cada uma com suas próprias características e estratégias de jogo. Os personagens também possuem pontos de vida, pontos de magia e pontos de experiência, elementos essenciais para sua progressão e evolução ao longo do jogo. A flexibilidade do sistema permite a introdução de novas classes e ajustes nos atributos das classes existentes, garantindo a diversidade e o equilíbrio das batalhas. Este relatório detalhará o processo de desenvolvimento e implementação do jogo.

2 FUNCIONALIDADES

O projeto possui, além da classe App, usada apenas para iniciar o jogo, mais 10 classes que são:

- Character: classe para armazenar as informações e métodos dos personagens
- CharacterClass: classe abstrata para armazenar informações de atributos e habilidades dos personagens
- Monster / Warrior / Archer / Wizard: extende a classe CharacterClass e é usada apenas para instanciar a classe CharacterClass
- Skill: classe para armazenar as informações e métodos das habilidades do jogo
- Team: classe para armazenar as informações e métodos sobre a equipe de personagens
- WeightAttributes: classe para armazenar os pesos dos atributos de cada classe para o cálculo de dano e mana.
- Game: classe onde ocorre todo o processamento do jogo e da interface gráfica

As classes CharacterClass, WeightAttributes e Skill contem apenas os métodos getters e setters de seus atributos, as demais, além dos getters e setters, possuem métodos usados nas batalhas e são os seguintes:

1. Classe **Team**
    1. `insertTeamMate()` permite adicionar um personagem à equipe.
    2. `getFullTeam()` retorna a lista completa de personagens na equipe.
    3. `getCharacter(int i)` permite acessar um personagem específico na equipe com base no índice.
    4. `totalLife()` calcula e retorna a vida total de todos os membros da equipe.
    5. `totalTeamMates()` retorna o número total de membros na equipe.
    6. `nextToPlay()` determina qual personagem da equipe deve jogar a próxima rodada, com base no menor tempo de recarga.
    7. `decreaseAllCooldown()` reduz o tempo de recarga de todas os membros da equipe após uma rodada.
    8. `getOtherTeamMate(Character current)` retorna o outro membro da equipe com base no membro atual.
    9. `removeTeamMate(Character charToRemove)` remove um membro específico da equipe.
2. Classe **Character**
    1. `getMaxLife()`: Calcula e retorna a vida máxima do personagem.
    2. `getMaxMana()`: Calcula e retorna a mana máxima do personagem.
    3. `damage(Skill skill)`: Calcula e retorna o dano causado pela habilidade do personagem.
    4. `mana(Skill skill)`: Calcula e retorna o custo de mana da habilidade do personagem.
    5. `decreaseCooldown()`: Reduz o tempo de cooldown das habilidades do personagem.
    6. `restoreCooldown()`: Restaura o tempo de cooldown das habilidades do personagem.
    7. `setCooldown(int time)`: Define o tempo de cooldown de uma habilidade.
    8. `setStuuned(boolean stuuned)`: Define se o personagem está atordoado.

9. `setExp(int levelDefeated)`: Incrementa a experiência do personagem após uma batalha.
10. `setLevelUp()`: Aumenta o nível do personagem e ajusta seus atributos.
11. `getEnemyMove(Team mainTeam, Team enemyTeam)`: Determina a habilidade que o inimigo utilizará com base em várias ponderações.
12. `hasMana(float attackMana)`: Verifica se o personagem possui mana suficiente para lançar uma habilidade.
13. `downMana(float attackMana)`: Reduz a quantidade de mana do personagem após lançar uma habilidade.
14. `hurtChar(float damage)`: Reduz a vida do personagem com base no dano recebido.
15. `setLifeHeal(float heal)`: Restaura a vida do personagem com base na quantidade de cura recebida.

3. Classe **Game**
    1. `runGame(String file)`: Controla o fluxo do jogo, carrega a história do arquivo fornecido, inicializa os personagens do jogador e dos inimigos, e inicia as batalhas.
    2. `runBattle()`: Gerencia o processo de batalha entre os personagens do jogador e os inimigos.
    3. `playerTurn(Character currentPlayer)`: Controla o turno do jogador, permitindo que ele escolha uma habilidade e um inimigo para atacar.
    4. `enemyTurn(Character currentPlayer)`: Controla o turno dos inimigos, permitindo que eles escolham uma habilidade e um alvo para atacar.
    5. `checkClassName(String charClassName)`: Verifica se o nome da classe de personagem fornecido é válido.
    6. `initializeAndExtractCharacterData(String name, String charClassName, int level)`: Inicializa um novo personagem com base nos dados fornecidos.
    7. `initializeSkills(CharacterClass charClass, String charClassName)`: Extrai as habilidades do arquivo de inicialização para uma determinada classe de personagem.

8. `getStory()`: Extrai a história do arquivo fornecido.
9. `checkTeams()`: Verifica se ambos os times (do jogador e dos inimigos) estão prontos para a batalha.
10. `getSelectedIndexSkill()`: Retorna o índice da habilidade selecionada pelo jogador.
11. `getSelectedIndexEnemy(ButtonGroup buttonGroupEnemy)`: Retorna o índice do inimigo selecionado pelo jogador.
12. `setDelay(int seconds)`: Introduz um atraso na execução do programa para efeitos de visualização.
13. `rechargeButtons(ButtonGroup buttonGroup)`: Limpa a seleção dos botões de um grupo.
14. `prepareUI()`: Prepara a interface gráfica para a próxima ação do jogador.
15. `setPlayerIcon(Character player)`: Define os ícones dos personagens com base em sua classe.

3 DIFICULDADES

Durante o desenvolvimento do jogo, enfrentei desafios iniciais devido à minha limitada experiência em projetos de grande escala em Java. No entanto, rapidamente superei essas dificuldades. A lógica do jogo foi compreensível e fácil de implementar, mas a criação da interface gráfica representou um desafio significativo devido à minha falta de experiência prévia nessa área. No entanto, com dedicação e aprendizado contínuo, fui capaz de superar esses obstáculos e concluir o projeto com sucesso.

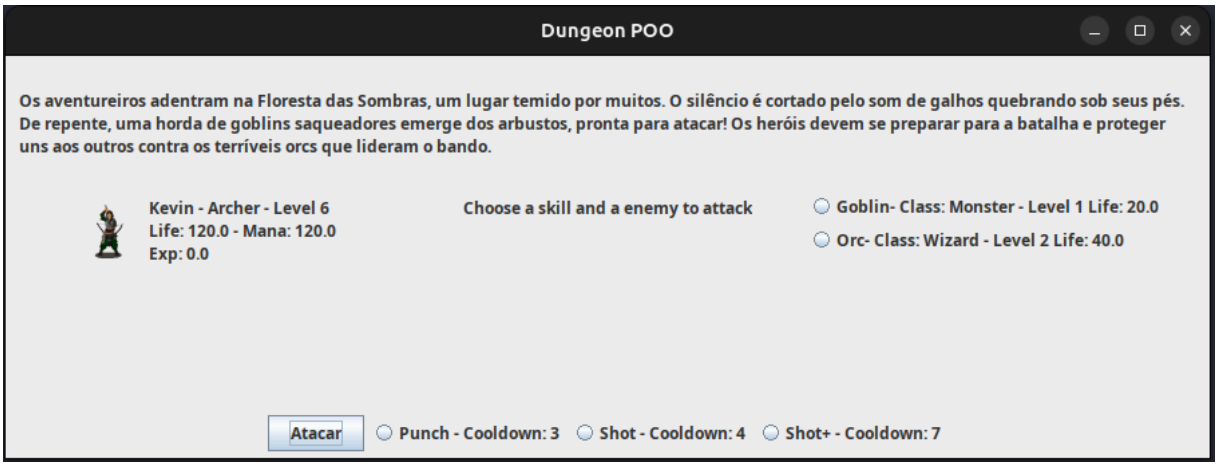
4 PASSOS PARA EXECUTAR

Não foi necessária nenhuma biblioteca externa, então para executar basta apenas extrair o arquivo .ZIP e executar dentro da pasta Dungeon/src o arquivo App.java.

5 UTILIZAÇÃO

O design do jogo é extremamente simples e intuitivo, apresentando uma única tela. Na parte superior, o jogador é imerso na história da batalha em curso. No centro da tela, durante o turno, são exibidos os eventos e informações relevantes sobre a batalha. Durante o turno do inimigo, apenas as ações dele são exibidas. Já durante o turno do jogador, as informações do

personagem, as opções de inimigos, as habilidades disponíveis e um botão para atacar são apresentados, permitindo que o jogador faça suas escolhas de forma clara e eficaz.



6 DIAGRAMA DE CLASSES

7 PROGRAMAÇÃO ORIENTADA A OBJETOS

O design do jogo é extremamente simples e intuitivo, apresentando uma única tela. Na parte superior, o jogador é imerso na história da batalha em curso. No centro da tela, durante o turno, são exibidos os eventos e informações relevantes sobre a batalha. Durante o turno do inimigo, apenas as ações dele são exibidas. Já durante o turno do jogador, as informações do personagem, as opções de inimigos, as habilidades disponíveis e um botão para atacar são apresentados, permitindo que o jogador faça suas escolhas de forma clara e eficaz.

8 CONCLUSÃO

Em resumo, o desenvolvimento do jogo em Java apresentou desafios iniciais, especialmente na criação da interface gráfica. O design simples e intuitivo do jogo proporciona uma experiência clara para o jogador, com uma única tela para imersão na história e tomada de decisões durante as batalhas. A aplicação consistente de métodos ao longo do projeto demonstra a eficácia da Programação Orientada a Objetos, facilitando a organização do código e promovendo a reutilização de funcionalidades. Embora tenha sido uma escolha deliberada não explorar elementos mais complexos da POO, como interfaces e enumerações, o projeto alcançou seus objetivos ao implementar os conceitos básicos de forma sólida e funcional.

9 PROPOSTA

Acredito que para projetos futuros seja interessante implementar coisas que gostamos, assim como esse RPG, poderia ser feito também outros tipos de jogos por turno, talvez um jogo de batalha de navios ou um jogo de cartas simples com diferentes tipos de classes como o RPG. Acredito que fazer projetos que fujam um pouco do padrão sejam mais motivadores.