Relatório do Projeto Final de PEOO

Prof: Rafael Lopes Gomes

Equipe:

- Wosley Mendes Rocha

- Antônio Emílio Guilhon Lôbo Filho

- Francisco Wesley Mendes Maciel

- Pedro Henrique Lorenzom

1. Introdução

Para o projeto decidimos por fazer um jogo de Battle Royale com um conceito um pouco diferente. Para que não repetíssemos a mesma fórmula comum de jogos deste estilo que usam apenas o PvP (Player vs Player), usamos como inspiração o Battle Royale de Tetris, o Tetris 99. Neste jogo há PvP mas ele acontece de maneira indireta. A base do jogo é um PvE (Player vs Environment) que acaba influenciando no jogo dos outros jogadores. Na próxima sessão explicaremos um pouco da história do nosso jogo para depois explicarmos como ele funciona.

É importante lembrar que mesmo usando um conceito diferente, tentamos sempre nos atentar a manter as funcionalidades exigidas no trabalho. Às vezes fizemos apenas pequenas adaptações para encaixar as funcionalidades no conceito do jogo, mas acreditamos que essas adaptações acabam não afetando o aprendizado consequente das propostas exigidas.

2. Resumo da História do Jogo

O nosso jogo, que se chama “O Último Teleportador!”, se passa no ano de 2232 e o jogo representa um torneio criado pela sociedade desta época futurista. Neste futuro não tão próximo, foi criado o “Teleportador de Mão” ou “Teleportador Portátil”. Com ele o ser humano poderia teleportar coisas pequenas para outros lugares instantaneamente. Com esse conceito, logo surgiu a ideia de criar uma nova modalidade esportiva que tivesse o Teleportador Portátil como instrumento principal. Rapidamente essa nova modalidade, que leva o mesmo nome do nosso jogo, se tornou um grande sucesso no planeta. Na próxima sessão explicaremos como o jogo funciona.

3. Funcionalidades e Regras do Jogo

3.1. Mapas dos Personagens

Neste jogo, que precisa ser jogado por pelo menos por duas pessoas, os jogadores são dispostos em um “mapa” que terá as seguintes coordenadas:

[0] [1] [2]

[0] x x x

[1] x x x

[2] x x x

[3] x x x

[posição do jogador]

(figura 1)

Cada jogador terá seu próprio mapa e inimigos (controlados pela CPU) aparecerão nos pontos onde há os ‘x’. Se houver um número n no lugar de um x, há n inimigo naquela coordenada, se houver 0, não há inimigos ali. Exemplo abaixo de um mapa do jogo:

[0] [1] [2]

[0] 0 0 0

[1] 1 0 0

[2] 0 1 2

[3] 0 0 0

[posição do jogador]

(figura 2)

Como podemos ver na figura 2 acima, há 1 inimigo nas coordenadas (1,0) e (2,1) e 2 inimigos na coordenada (2,2).

3.2. Rodadas

Uma das exigências do trabalho é que o jogo seja feito por rodadas. Jogos neste estilo requerem mais estratégia do que habilidade, por isso escolhemos o conceito deste jogo. Abaixo explicaremos como funcionará o passo-a-passo de cada rodada:

Passo 01 - Sorteio de ordem das jogadas: Esta também é uma exigência do trabalho. Ao início de cada rodada será sorteada a ordem de turnos daquela rodada e essa informação aparecerá para os jogadores.

Passo 02 – Os inimigos andam uma casa para frente (linha++) em direção à posição do seu personagem. Caso estejam na linha 3 e andem mais uma vez, eles entrarão em contato com o personagem e darão 1 de dano para cada inimigo naquela coordenada. É importante ficar atento.

Passo 03 – Escolha da ação do jogador no turno: Quando for sua vez, você terá que realizar uma ação. As ações permitidas são:

1 – Atacar: Esta ação determina que você usará o seu teleportador em algum inimigo que está no seu mapa. Basta informar as coordenadas do inimigo no formato a seguir: linha coluna (Exemplo: Se digitar 1 0 você atacará o inimigo na linha 1, coluna 0).

1.1 – Consequência de Atacar: Esta ação é importante para evitar que inimigos se aproximem da posição do personagem e assim evitar de tomar dano. Mas há também a consequência que define o conceito de batalha do jogo. O inimigo teleportado do seu mapa, aparecerá no mapa do próximo personagem a jogar (se você for o último da rodada, ele teleportará para o primeiro). É importante lembrar que o ataque gasta 1 de munição do personagem.

2 – Defender: Esta ação faz com que você gaste seu turno para recuperar 1 de Vida. Use-a com sabedoria, ficar um turno sem atacar pode ser perigoso!

3 – Recuperar Munição: Aqui substituímos o conceito de Power-Up, pelo de munição. Fizemos isso pois acreditamos que faça mais sentido dentro do contexto do nosso jogo. Ao usar esta ação o personagem recupera 2 de munição. É importante ficar atento à munição, gerenciá-la bem é crucial para a vitória.

4 – Especial: Esta ação só pode ser realizada uma vez por partida. Ainda falaremos de Classes, mas é importante saber que cada Classe do jogo possui um especial diferente. Na sessão de Classes explicaremos a diferença entre eles.

Passo 04 – Repetição dos *Passos 02 e 03* para cada jogador até terminar a rodada.

Passo 05 – Retorna ao *Passo 01* até encontrar um vencedor.

Vencedor: Quando sobrar apenas um último jogador vivo o vencedor será anunciado! (Conceito de Battle Royale).

4. Atributos e Classes

4.1. Atributos

Cada personagem do jogo possui atributos específicos que os diferenciam dos outros personagens. Os atributos são determinados por qual classe o personagem pertence. Segue abaixo a lista de atributos e quanto cada classe possui de cada um deles:

- Ataque (atq) – Determina quantos inimigos/coordenada serão teleportados. Exemplo: Com 2 de atq o personagem pode teleportar até dois inimigos por ação de ataque. Contanto que os inimigos estejam na mesma coordenada.
- Defesa(def) – Determina quanto o personagem aguenta sofrer dano. Este atributo está diretamente ligado com a Vida Máxima de um personagem.
- Vida Máxima – Determina quanto de vida tem o personagem. O cálculo para Vida Máxima é feito por (defesa +2).
- Munição Máxima – Determina quanto de munição cada personagem pode ter no máximo.

4.2. Classes

Atualmente o jogo possui três classes. Cada classe possui atributos específicos e também um poder especial diferente. Para as particularidades das classes, temos a lista abaixo:

4.2.1. Atirador de Elite

Ataque = 2; Defesa = 1; Vida Máxima = 3; Munição Máxima = 3;

Especial = Causa diretamente 1 de dano ao próximo jogador da rodada (se você for o último da rodada, ele causará dano no primeiro personagem da rodada).

4.2.2. Defensor

Ataque = 1; Defesa = 2; Vida Máxima = 4; Munição Máxima = 4;

Especial = Recupera 3 pontos de Vida.

4.2.3. Estrategista

Ataque = 1; Defesa = 1; Vida Máxima = 3; Munição Máxima = 5;

Especial = Rouba 3 Munições do próximo jogador da rodada (se você for o último da rodada, você roubará do primeiro personagem da rodada).

5. Implementação do Código do Jogo

Nosso código possui 7 classes e 1 interface. São elas:

5.1. Jogo (Main)

Na classe Jogo definimos o nosso Main. Aqui acontecerá as instruções para que a partida seja realizada. Definimos aqui também o Salvamento e Carregamento do jogo.

5.2. PersonagemAbs

Esta é uma classe abstrata que define o conceito de Personagem. Seja ele jogável ou não. Nesta versão do jogo ainda não há NPCs (Personagens não jogáveis) definidos, mas é importante definir uma classe abstrata que atenda a este conceito para possíveis mudanças no futuro.

5.3. PersonagemJogavel

Aqui definimos melhor o que é um personagem jogável e quais são seus atributos e métodos. Esta classe herda a classe abstrata PersonagemAbs.

5.4. AtiradorDeElite, Defensor e Estrategista

Estas três classes definem as classes do jogo. Nela também definimos o método de Especial usando Polimorfismo para determinar as diferenças dos especiais de cada classe. Estas classes herdam a classe PersonagemJogavel.

5.5. Ambiente (Batalha)

Esta classe constrói os mapas de cada personagem e é responsável por aplicar os métodos de batalha do jogo.

5.6. Especial (Interface)

Esta classe define o texto de explicação do Especial de cada classe

6. Checklist das Funcionalidades Exigidas no Projeto

6.1. Jogo no Estilo Battle Royale

- O jogo segue o estilo de jogos Battle Royale com PvP indireto, como no exemplo do jogo Tetris 99. Portanto o conceito está presente. ✔

6.2. Criar, Deletar, Salvar e Carregar Personagens

- Ao criarmos um novo jogo nós podemos criar novos personagens. (Linhas 42 a 66 da classe *Jogo*). ✔

- Se já tivermos personagens salvos podemos deletá-los, sobrescrevendo novos personagens no arquivo. (Linhas 86 a 134 da classe *Jogo*). ✔

- No menu inicial, se escolhermos a segunda opção nós carregaremos um novo jogo com os personagens já salvos. (Linhas 139 a 176 da classe *Jogo*). ✔

Observação: No salvamento e carregamento do jogo usamos arquivos .json aplicando a biblioteca gson, do Google.

6.3. Criar um Menu Inicial

- O menu foi criado com um *switch()* que está nas linhas 36 a 203 da classe Jogo. ✔

6.4. Realizar batalhas

- O jogo cria batalhas se escolhermos a opção 1 do menu principal, criando um novo jogo. Ou carregando personagens já criados, usando a opção 2 do menu principal. ✔

6.5. Batalha Feita em Rodadas Com Ações Sorteadas

- As batalhas por turno acontecem a partir da linha 206 até a linha 312 da classe *Jogo*. ✔

- O sorteio das ações por rodada é feito na linha 221 da classe *Jogo*. ✔

6.6. Tipos de Ações: Atacar, Defender e Power-Ups.

- Escolha de Atacar está a partir da linha 260 da classe Jogo. ✔

- Escolha de Defender está a partir da linha 284 da classe Jogo. ✔

- Escolha de Power-Ups, como informado anteriormente, foi substituída por Recuperar Munição para melhor se encaixar no conceito do jogo. Esta opção está a partir da linha 288 da classe Jogo. ✔

6.7. Funcionalidade dos Personagens (Atributos e Classes)

- A sessão 4 deste relatório descreve bem as funcionalidades, atributos, classes e especiais de cada personagem. ✔

6.8. Critérios de Avaliação

6.8.1. Uso de Herança

- A classe *PersonagemJogavel* herda as informações da classe *PersonagemAbs* e as classes *AtiradorDeElite*, *Defensor* e *Estrategista* herdam as informações da classe *PersonagemJogavel*. ✔

6.8.2. Encapsulamento de Atributos e Métodos

- As classes criadas usam atributos *private* ou *protected* e possuem *gets* e *sets* adequadamente. ✔

6.8.3 Aplicação de Classes Abstratas, Polimorfismo e Interfaces

- Criamos a classe abstrata para definir o conceito básico de personagem. Ver classe *PersonagemAbs*. ✔

- Usamos polimorfismo para determinar os especiais de cada classe do jogo. Os métodos de especiais sobrescrevem (*override*) o método da classe mãe. ✔

- Criamos a interface *Especial* para definir o texto de Especial de cada personagem. ✔

6.8.4. Tratamento de Exceções

- Fizemos tratamento de Exceções nos momentos de salvar (a partir da linha 99 da classe *Jogo*) e ao carregar (a partir da linha 139 da classe Jogo). ✔

7. Próximos Passos

Segue uma pequena lista de próximos passos para este projeto:

- Correção de possíveis *bugs*.

- Aproveitar a criação da classe abstrata *PersonagemAbs* para criar uma nova classe de personagens não jogáveis (NPCs) e defini-los como inimigos com funções diferentes para inserir no mapa de cada personagem.

- Criar novas classes e, consequentemente, novos especiais.

- Criar novos métodos aleatórios de alocação e movimentação de inimigos dentro do mapa de cada personagem.