Engenharia de Software - Profª Ana C. V. de Melo
Projeto Kindred - Fase 1

Bruno Guilherme Ricci Lucas (4460596)
Gustavo Chicato Wandeur (7557797)
Leonardo Pereira Macedo (8536065)
Vinícius Bitencourt Matos (8536221)

15 de outubro de 2015

1 Dicionário de Dados

- **Exército:** Conjunto de *unidades* controladas por um jogador.
- **Unidade:** Uma personagem do *exército* do jogador. Possui *hp*, *atq*, *def*, *agi*, *mov* e uma *classe*, e pode equipar uma *arma*. Se um jogador não possuir nenhuma *unidade*, ele perde a partida.
- **Turno:** Rodada de um jogador. Nela, a pessoa de quem é a vez pode mover suas *unidades* e atacar o *exército* do oponente enquanto este assiste o que ocorre. Ao encerrar o *turno*, passa-se a vez para o adversário jogar.
- **Classe:** Grupo à qual uma *unidade* pertence. Afeta o *HP*, *mov*, *atq*, *def*, *agi* e limita as *armas* usadas pela *unidade*.
- **HP:** Hit Points. A quantidade de dano que uma *unidade* pode receber antes de ser destruída.
- **Atq:** Representação numérica do dano que uma *unidade* pode causar.
- **Def:** Representação numérica da redução do dano sofrido por uma *unidade*.
- **Agi:** Representação numérica da agilidade de uma *unidade*. Afeta a chance de se esquivar de um ataque.
- **Mov:** Representa o número de *tiles* que uma *unidade* pode se deslocar a cada *turno*, horizontalmente e verticalmente.
- **Range:** Alcance do ataque de uma *unidade*. Uma *unidade* pode atacar um oponente que esteja até um número *range* de *tiles* à distância. O valor varia de acordo com a *arma* equipada.
- **Arma:** Arma que uma unidade pode equipar. Cada tipo de arma tem seu alcance. Pode afetar o *atq* da *unidade* de acordo com o *range* do ataque.
- **Dano:** Representa o *HP* perdido por uma *unidade* que recebeu um ataque. O cálculo do *dano* envolve o *atq* do atacante e os atributos *def* e *agi* do defensor.
- **Tile:** É um "quadrado" que compõe o *mapa*. Cada *tile* tem um *terrain* e pode conter, no máximo, uma *unidade*.
- **Terreno:** Espécie de "ambiente" que ocorre em um *tile* (por exemplo, floresta ou planície). Pode afetar o *mov*, *def* ou *agi*, dando benefícios ou prejuízos à *unidade* que está no *tile*.
- **Mapa:** Um conjunto de *tiles* no qual os jogadores controlam seus *exércitos* e se enfrentam.

2 Descrição dos Use-Cases

2.1 Nome do Use-Case

Criação e realização de uma partida do jogo Kindred.

2.2 Resumo

Entrada do usuário numa sala para jogar uma partida de Kindred e a realização da mesma.

2.3 Atores

Jogador cliente e jogador hospedeiro.

2.4 Precondição

Jogo já está aberto no menu principal.

2.5 Fluxo Normal dos Eventos

1. O usuário escolhe jogar uma partida.
2. O usuário decide hospedar uma partida.
3. O usuário escolhe as opções da partida (mapa, etc.).
4. O usuário hospedeiro aguarda um outro jogador cliente se conectar.
5. Os usuários hospedeiro e conectado confirmam a partida.
6. Os jogadores posicionam suas unidades no mapa para a batalha começar.
7. Turno do jogador A. Este pode mover e atacar com suas unidades.
8. O jogador A encerra seu turno.
9. Turno do jogador B. Este pode mover e atacar com suas unidades.
10. O jogador B encerra seu turno.
11. Turno do jogador A novamente (retorna-se ao passo 7).
12. Eventualmente, um jogador destruirá todas as unidades do oponente.

2.6 Fluxos Alternativos

2.6.1 Fechamento do Jogo

No passo 1, o usuário pode decidir sair do jogo, o que encerra o programa.

2.6.2 Fechamento do Jogo

Em vez de hospedar uma partida, o jogador pode optar, no passo 2, por juntar-se a uma sala existente:

1. O usuário procura alguma sala pela qual tenha interessante nas existentes.
2. O usuário conecta-se à sala.
3. Retorno ao passo 5 do fluxo normal.

2.6.3 Cancelamento da Partida

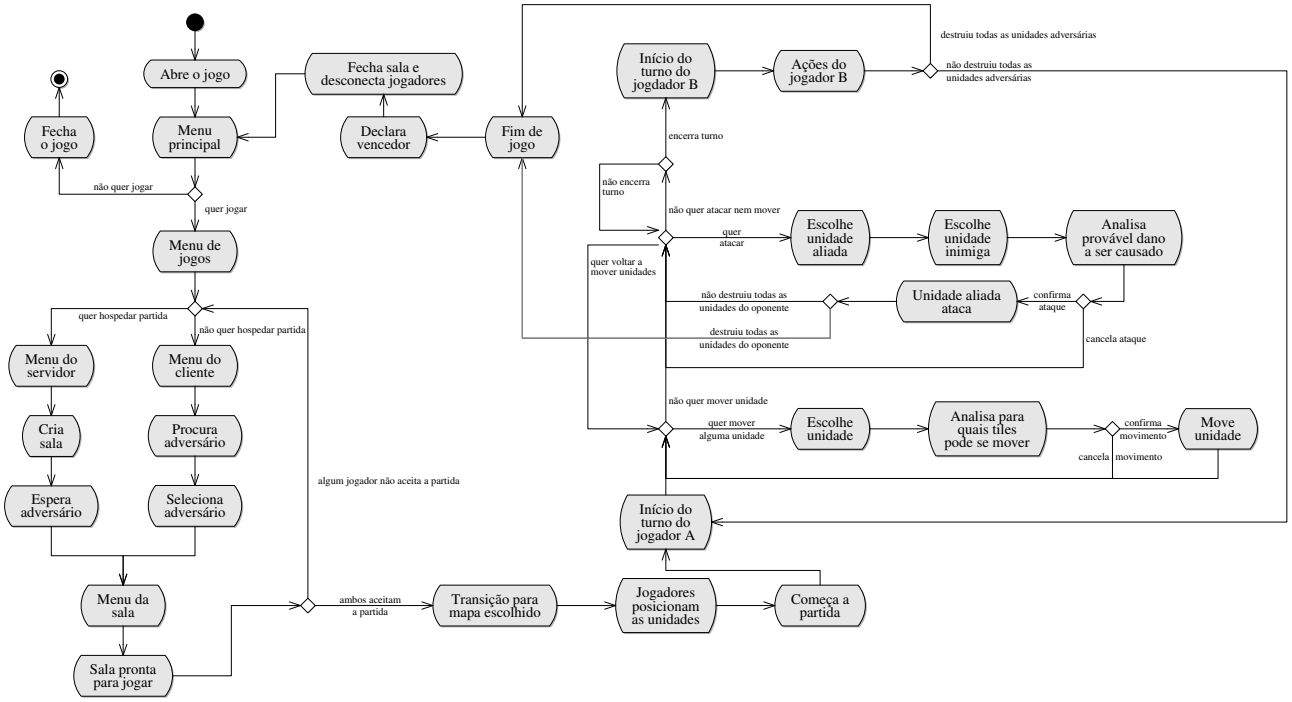
No passo 5, se algum dos jogadores (hospedeiro ou conectado) recusar a partida, retorna-se ao passo 1 do fluxo normal.

2.7 Condições Posteriores

2.7.1 Fim do Jogo

Quando o passo 12 ocorrer, a declaração do jogador vencedor é feita e a partida é finalizada, retornando os usuários ao passo 1 do fluxo normal.

3 Diagrama de Atividades

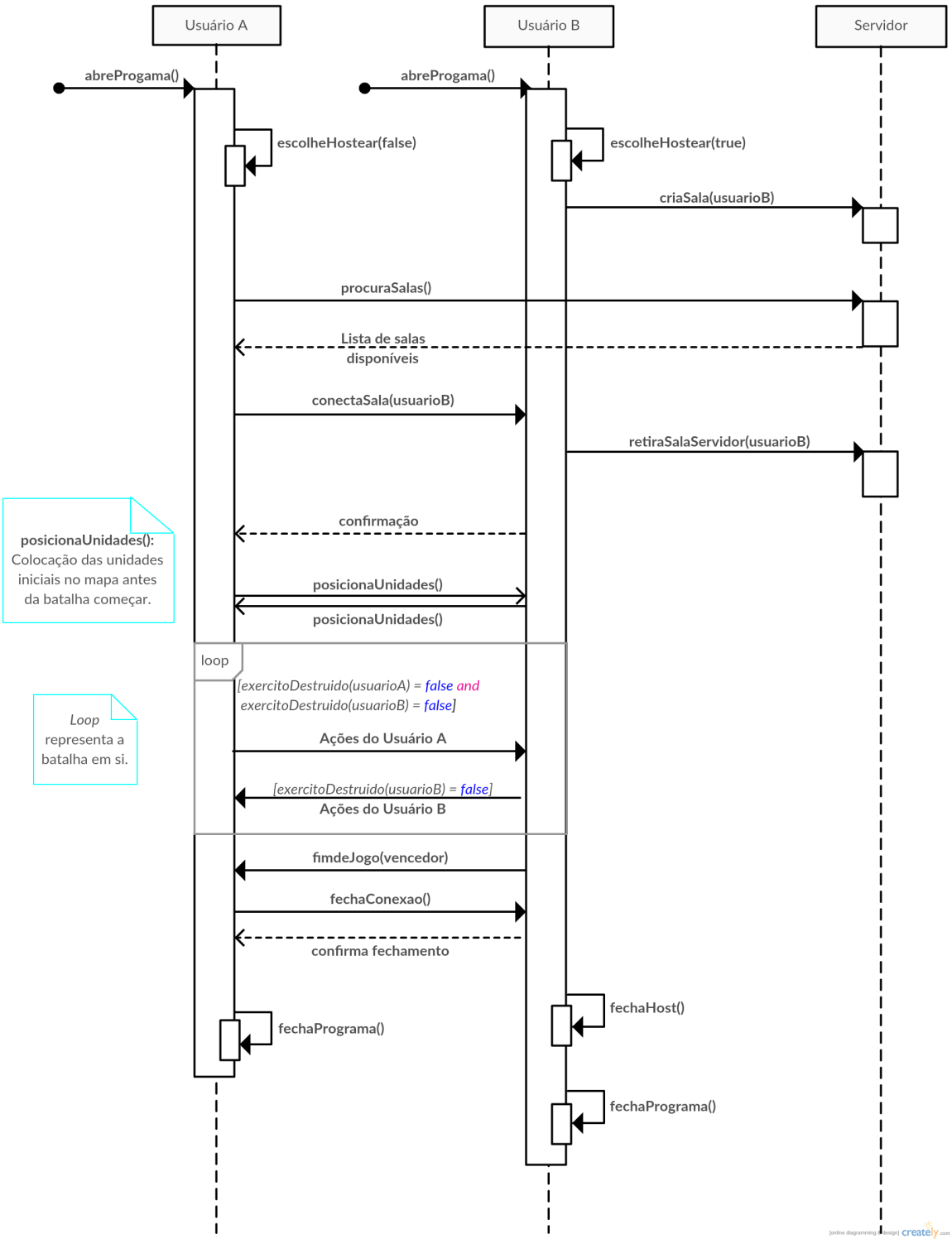


4 Diagrama de Sequência

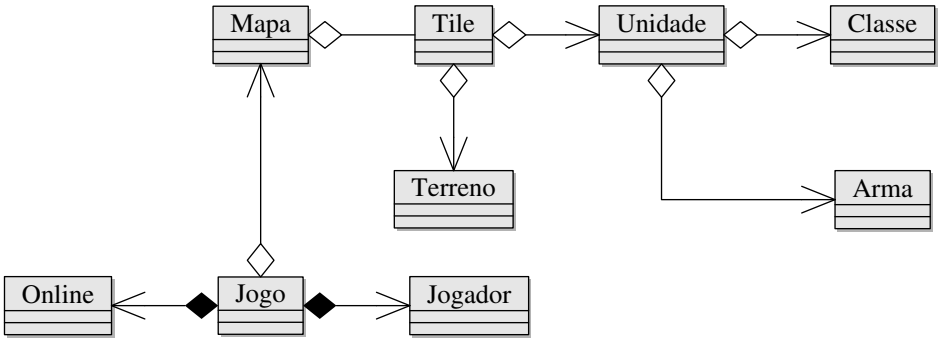
A seguir, representamos o diagrama de sequência em uma situação de uso ideal do programa.

O usuário A representa um jogador que irá se conectar a uma sala existente e o jogador B, um jogador que irá hospedar uma sala.

O servidor armazena todas as salas sendo hospedadas no momento para que os jogadores não hospedeiros possam escolher um adversário.


Usuário A
Usuário B
Servidor
abreProgama()
abreProgama()
escolheHostear(false)
escolheHostear(true)
criaSala(usuarioB)
procuraSalas()
Lista de salas disponíveis
conectaSala(usuarioB)
retiraSalaServidor(usuarioB)
confirmação
posicionaUnidades(): Colocação das unidades iniciais no mapa antes da batalha começar.
posicionaUnidades()
posicionaUnidades()
loop
[exercitoDestruido(usuarioA) = false and exercitoDestruido(usuarioB) = false]
Loop representa a batalha em si.
Ações do Usuário A
[exercitoDestruido(usuarioB) = false]
Ações do Usuário B
fimdeJogo(vencedor)
fechaConexao()
confirma fechamento
fechaHost()
fechaPrograma()
fechaPrograma()

5 Identificação dos objetos/classes necessários ao projeto (esboço inicial)



- **Classe:** Contém a *classe* à qual a *unidade* pertence e muda seus atributos de acordo.
- **Arma:** Contém a *arma* que a *unidade* está usando. Muda a *range* da *unidade* e o *dano* dela.
- **Unidade:** Contém os atributos (*HP*, *atq*, *def*, *agi*, *mov*) de uma *unidade* do *exército* do jogador, bem como a *arma* que ela está usando e a *classe* à qual pertence.
- **Tile:** Guarda até uma *unidade* e contém um *terreno*. Afeta a performance da *unidade* (se houver uma) de acordo.
- **Mapa:** Armazena todos os *tiles* e suas posições (ou seja, como estão relacionados entre si).
- **Online:** Responsável por cuidar da conexão pela Internet, enviando e recebendo dados dos jogadores.
- **Jogador:** Cuida de todas as ações que o jogador pode exercer, como mover *unidade*, atacar, etc.
- **Jogo:** A classe central do sistema. Cuidará das atualizações das demais classes, manter o loop principal do jogo, inicializar as demais classes e encerrar o programa se o jogador o fechar. Também guarda o *turno* atual.

6 Documento de Especificação dos Requisitos do Sistema

6.1 Requisitos Funcionais

- Programa escrito em Java.
- Jogo de estratégia.
- Ambiente medieval.
- Voltado totalmente para multiplayer.
- Servidor contendo salas sendo hospedadas.
- Um dos jogadores hospeda a partida, atuando como servidor, e espera outro cliente conectar-se a ele.
- O jogador pode escolher um oponente para se conectar e jogar uma partida.
- Cada jogador controla um exército.
- O jogo é dividido em turnos.

6.2 Requisitos Não-Funcionais

- Segurança na troca de dados com outros jogadores.
- Estabilidade nas conexões.
- Servidor estável, com produção de um log contendo ações realizadas.
- Tentativa de reenvio de dados ou reconexão em caso de falha de conexão.
- Jogabilidade simples, ou seja, o modo de jogar deve ser intuitivo.