Alunos: Pedro Fatuch
Ricardo Carvalho

Simulação de jogo de RPG em turnos. O jogo consiste em dois personagens, o Bruiser (atacante) e o Healer (curandeiro). Também temos um player, que não é uma classe, mas sim um número aleatório gerado para nossos personagens tomarem dano. Colocamos esse elemento aleatório para simular a ação imprevisível do jogador. A idéia é que esses dois personagens trabalhem em conjunto com um curando enquanto o outro ataca o jogador. Quando a vida dos personagens é zerada, o jogador ganha. Se o loop acabar a IA ganhou. Quando a vida zera os personagens não executam as ações dos estados.

O agente bruiser é o nosso “lutador” ele é responsável por causar dano ao nosso suposto jogador, como é uma simulação com o propósito de utilizar técnicas de máquina de estado, o jogador não foi criado, então o agente lutador causa um dano fictício, sem ter um alvo para receber o dano, possuindo dois estados, descritos posteriormente. Além disso, quando o outro agente (healer) morre o bruiser entra em estado de fúria, causando mais dano.

O agente healer é o nosso “curandeiro” ele é responsável por manter a equipe viva, tentamos fazer o código de maneira que ele possa ter diferentes “aliados”, supondo que possam ter outros agentes diferentes posteriormente criados, este possui quatro estados, também descritos posteriormente, e utiliza mana no lugar de energia. Como a variável de energia é pertencente a todos os inimigos, apesar de não termos feito aqui, o curandeiro poderia atacar assim como o lutador com pequenas alterações.

Variáveis do agente Bruiser

Nome	Valor inicial	Descrição
int health	100	Vida atual, caso seja 0 o personagem “morre”
int maxHealth	100	Vida máxima, limita a vida do personagem após curado
int energy	50	Energia atual, gasta ao atacar e troca de estado ao zerar
int maxEnergy	50	Energia máxima, limita a energia do personagem após “descansar”

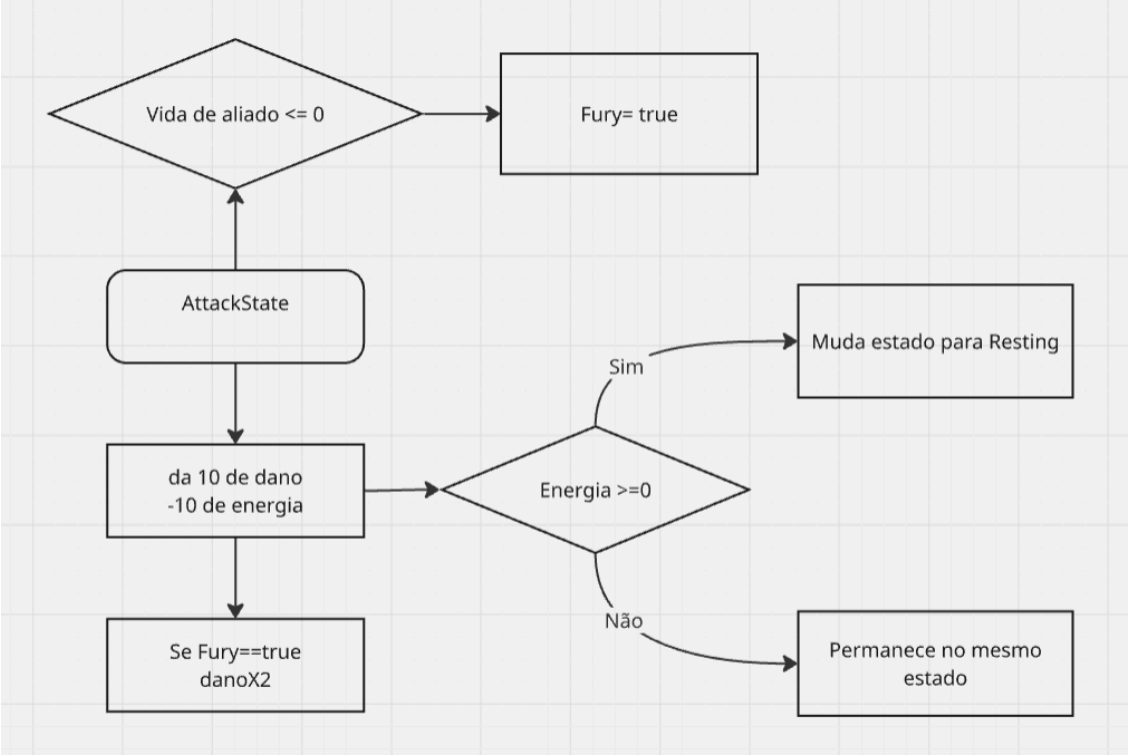
bool fury	false	Quando true, o personagem causa o dobro do dano

Possui dois estados: Resting e Attack, sendo Attack o estado padrão. No Resting ele recupera 10 de energia. Ele só troca de estado após sua energia voltar ao máximo, e entra nele quando sua energia zera.

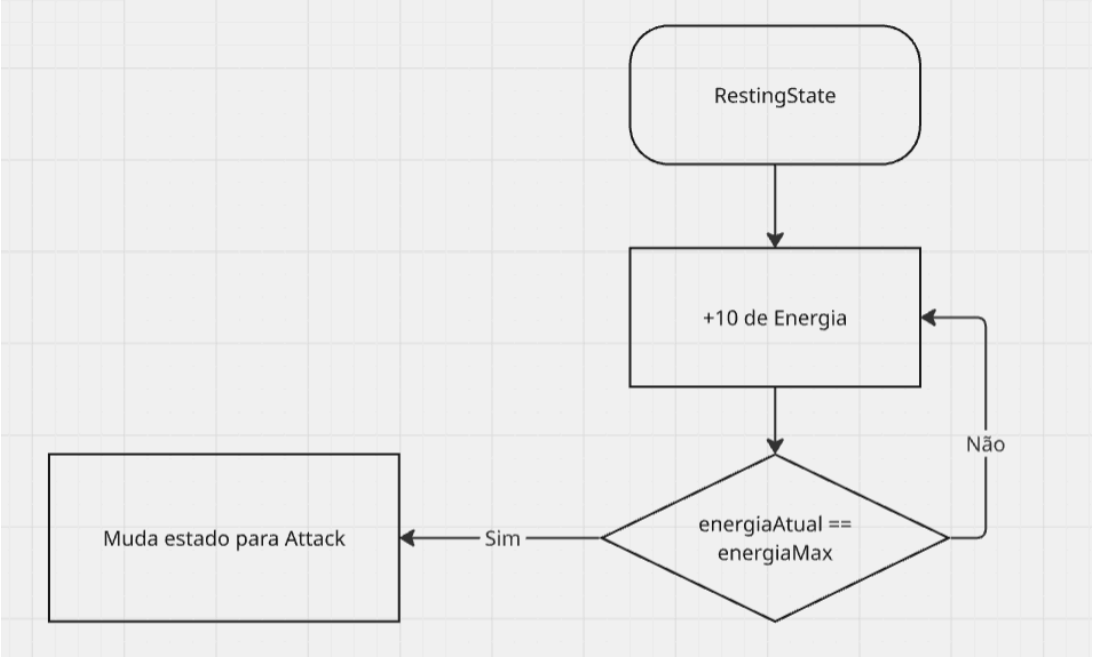
O Attack ele dá dano ao jogador de valor fixo no jogador e perde 10 de energia. Se a vida do aliado for menor ou igual a 0, fury vira true. Com essa variável em true, o valor do dano dobra.

Estados	Ação de entrada	Execução	Transições
Attack	Escreve no console informando a entrada no estado de ataque	Causa 10 de dano, ou 20 caso esteja em fúria	Caso fique sem energia muda de estado para Resting
Resting	Escreve no console informando entrada no estado de descanso	Aumenta em 10 a energia	Caso a energia esteja cheia muda de estado para Attack

Estado de Ataque



Estado de Resting(bruiser)



Healer:

Variáveis do agente Healer

Nome	Valor inicial	Descrição
int health	70	Vida atual, caso seja 0 o personagem “morre”
int maxHealth	70	Vida máxima, limita a vida do personagem após curado
int mana	80	Gasta mana para realizar os feitiços

Possui quatro estados: Resting, Area Heal, Healing e Sacrifice.

Area Heal é seu estado padrão. Ele vai restaurar 5 de vida para o Bruiser e para ele mesmo. Também vai perder 5 de mana.

Em Healing ele irá curar um único aliado, que pode ser ele mesmo ou o Bruiser. Para entrar neste estado um dos dois precisa estar com 50% ou menos de vida. Ele cura em 20 e perde 10 de mana.

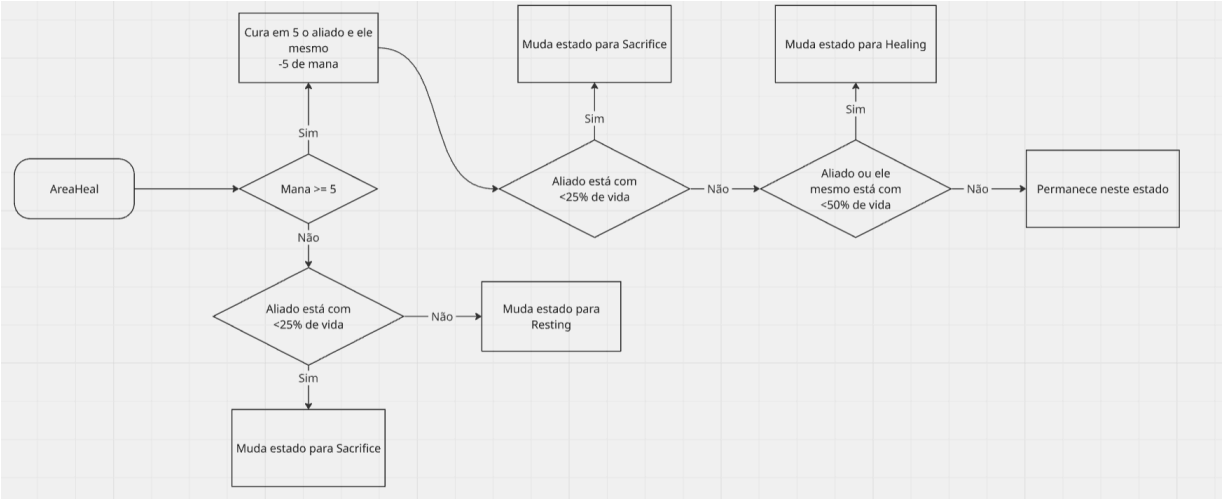
Sacrifice é um estado que executa se o bruiser estiver com 25% ou menos de sua vida máxima. O Healer zera sua vida e maximiza a vida do aliado.

Por último temos o Resting, aqui o Healer recupera 10 de mana e troca de estado logo depois. Ele só entra neste estado se não possuir mana para realizar alguma magia.

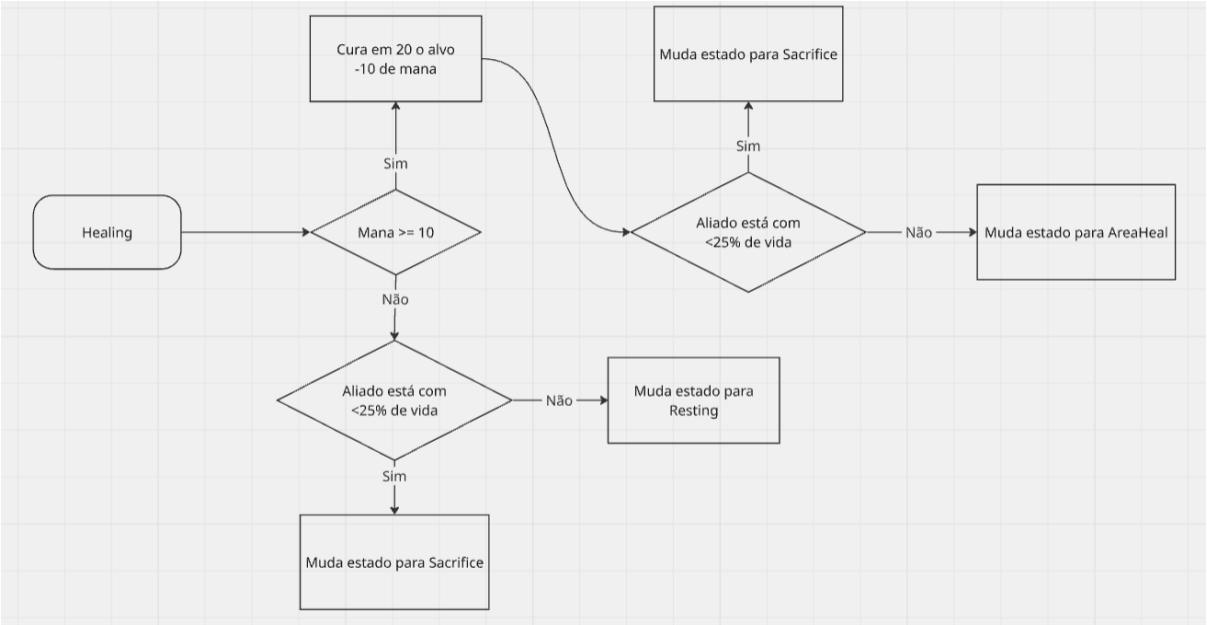
Estados	Ação de entrada	Execução	Transições
Area Heal	Escreve no console informando a entrada no estado de cura em area	Perde 5 de mana e cura 5 de vida de ambos os personagens	Caso aliado tenha menos de 25% de vida troca para Sacrifice, caso alguém tenha menos que 50% da vida, troca para Healing, caso tenha menos que 5 de mana, troca para Resting
Healing	Escreve no console informando entrada no estado de cura	Perde 10 de mana e cura 20 do personagem com a menor vida	Caso aliado tenha menos de 25% de vida troca para Sacrifice. Caso tenha menos que 5 de mana, troca para Resting. Caso ambos personagens tenham

			mais que 50% de vida, troca para Area Heal
Sacrifice	Escreve no console informando a entrada no estado de sacrifício	Perde 100 de vida (morrendo) para curar 100 de vida de seu aliado	Não tem transição pois o personagem morre
Resting	Escreve no console informando a entrada no estado de descanso	Aumenta em 5 sua mana	Caso aliado tenha menos de 25% de vida troca para Sacrifice. Caso alguém tenha menos de 50% da vida, troca para Healing. Caso todos tenham mais que 50% de vida, muda para Area Heal

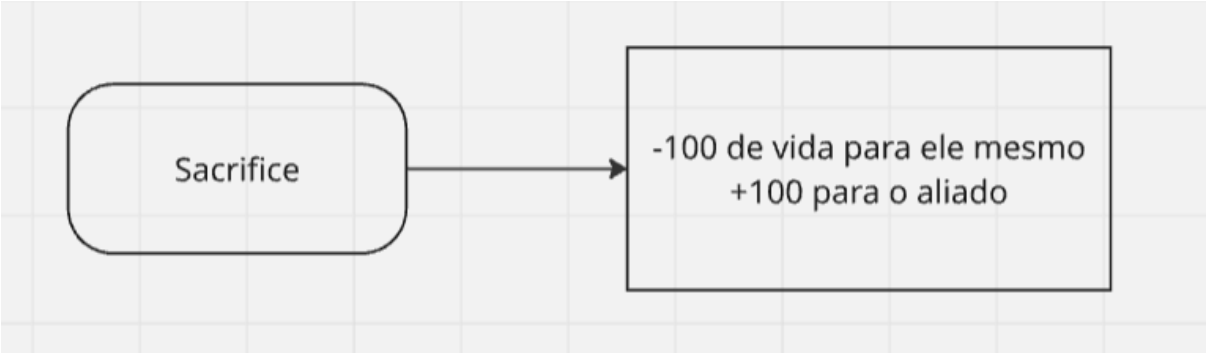
Estado de Area Heal



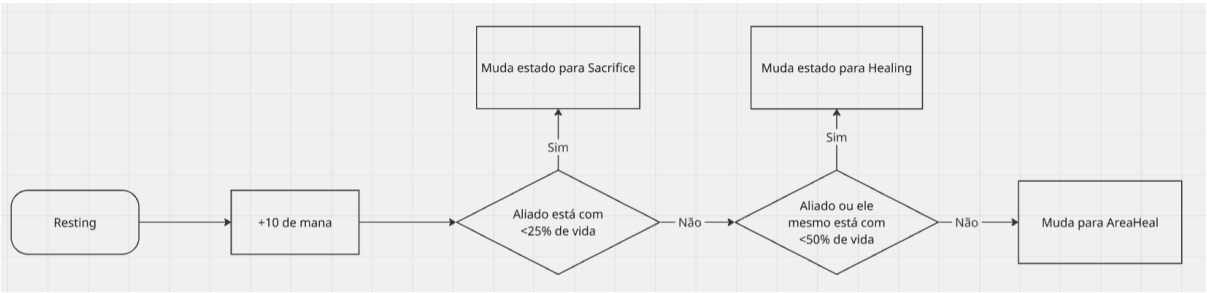
Estado de Healing



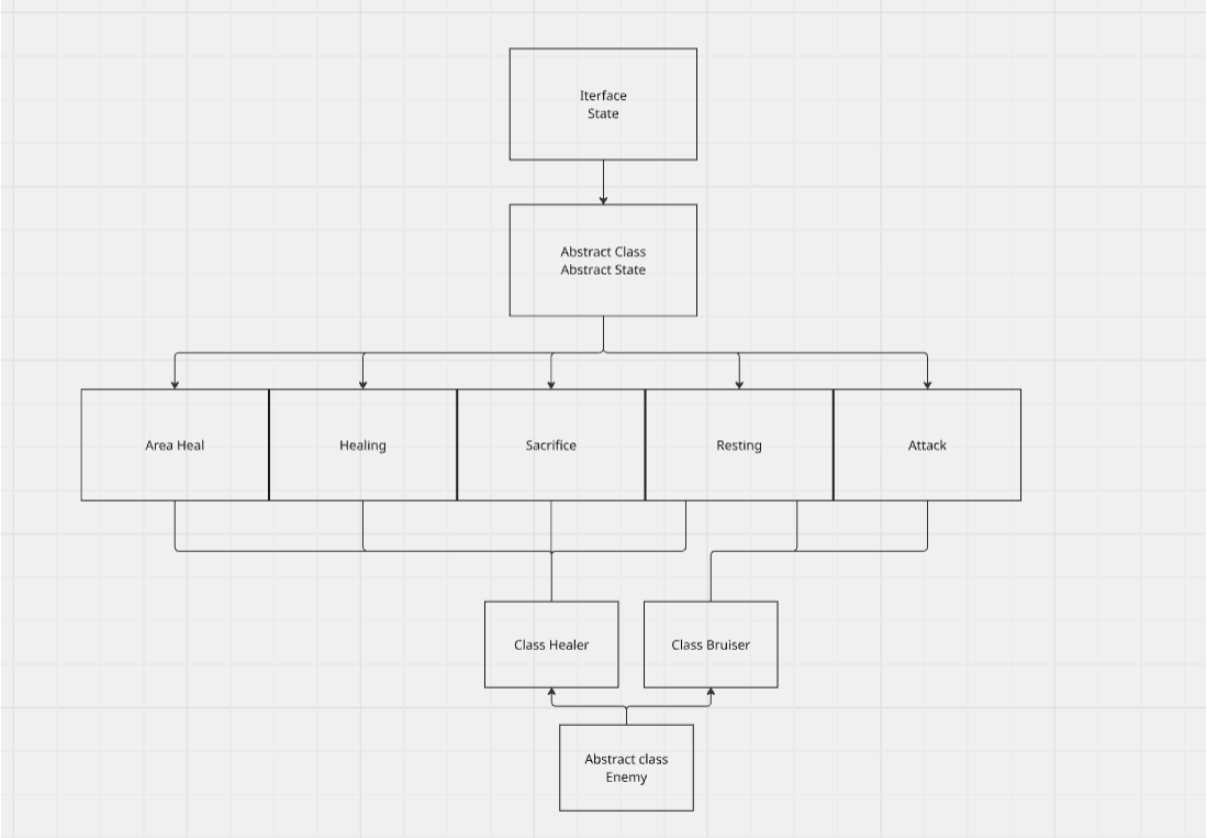
Estado de Sacrifício



Estado de Resting(Healer)



Estrutura do Código:



Temos primeiramente a interface State, que define o padrão de funções dos estados desejados. Implementando a interface temos o Abstract State, a nossa classe abstrata, que define a execução padrão de três funções que serão utilizadas em todos os estados, sendo elas: printStats, enter, leave. E sobre, Area Heal, Healing, Sacrifice, Resting e Attack, são os estados que nossos agentes podem estar, sendo que o Bruiser só pode estar em Attack ou Resting, e o Healer só pode estar em Area Heal, Healing, Sacrifice e Resting. Todos estes estados estendem do Abstract State, utilizando a mesma função de enter, leave e printStats, tendo apenas uma função de execute diferente para cada um.

Referente aos agentes, temos a Abstract Class Enemy, que define os padrões dos agentes. Dentro dele temos todas as variáveis que são utilizadas por ambos agentes, sendo elas: name, health, maxHealth, energy, maxEnergy, currentState e ally. Além disso, temos diversas funções, responsáveis por: devolver o valor de cada uma das variáveis, aumentar o valor das variáveis, diminuir o valor das variáveis, trocar de estado e checar se o agente está vivo. Estendendo da classe Enemy, temos os agentes Bruiser e Healer, Bruiser adiciona a variável de fúria, além de funções relacionadas à variável (verificar e alterar), já o agente adiciona a variável mana, além de funções relacionadas à variável(verificar e alterar).

Para finalizar temos o arquivo Main, que garante o loop do sistema, definindo o número de ciclos e que simula o dano do jogador aleatorizando um valor de dano e em qual dos agentes. Além disso, a condição de fim de jogo, de ambos os agentes estarem mortos, acontece também na main.

Resultados:
Quanto aos resultados esperados, abaixo temos um exemplo do log, mas o nosso arquivo “main” garante a repetição do loop x vezes, utilizamos um loop for então o limite de tempo existe, para sabermos em que etapa (ou ciclo) estamos nosso log inicia informando o ciclo de repetição atual. Depois disso recebemos as informações quanto ao player (dizendo em qual dos agentes causou dano, que por sua vez dispara uma mensagem do agente informando que recebeu dano. Depois, caso ambos estejam vivos, o primeiro agente a ser ativado é o Healer, então informamos isso escrevendo seu nome seguido de sua ação e seus resultados e por fim os status do agente após a ação (HP, Energia e Mana). Depois disso repetimos o processo para o agente Bruiser, mostrando sua ação e seus status após ela

Exemplo de log esperado:
==========INICIO DO CICLO 2 ==========

****Player*****
Jogador causou 18 de dano em Heitor
Heitor recebeu 18 de dano

****Heitor*****
Bruna recebeu 5 de cura
Heitor recebeu 5 de cura
Heitor curou Bruna
Status Heitor
HP: 46/70
Energia: 40/40
Mana: 70

****Bruna*****
Bruna atacou causando 10 de dano
Status Bruna
HP: 100/100
Energia: 30/50

Limitações:
Gostaríamos de ter adicionado uma classe player mais complexa e interativa, permitindo que nossa máquina de estados fosse “desafiada” por um jogador, além disso, tentamos deixar o código de uma maneira que fosse fácil a adição de novos tipos de agentes (além de bruiser e healer) mas pecamos em pensar em mais personagens existindo simultaneamente em alguns pontos, também poderíamos ter melhorado essa interação para permitir múltiplos agentes.