PROJETO. Jogos do tipo "ataque a torre" (*tower offense*) são bastante comuns atualmente. Nele, o objetivo é enviar tropas contra um defensor, causando dano suficiente para sua destruição dentro de um número máximo de rodadas. O objetivo deste projeto é criar um hardware capaz de simular um jogo deste tipo: o objetivo é destruir a famosa Enterprise, nave espacial da série Star Trek, em uma versão simplificada (e invertida) do jogo de tabuleiro "Star Trek Panic".

Comportamento desejado

O jogo a ser construído considera os seguintes elementos:

- O jogo é composto por rodadas, que têm duração de 1 ciclo de clock. Assim, o sinal de saída `turn`, composto por 5 bits, é incrementado a cada borda de subida do clock.
- O sistema é reinicializado colocando a entrada `reset` em `'1'`. Para iniciar o jogo em si, o jogador deve colocar o valor da entrada `reset` em '0', o que leva o valor inicial de `turn` para 1. A entrada de `reset` também pode reinicializar o sistema a qualquer momento, bastando para isso ser colocada em '1'.
- A cada rodada, o dano causado à Enterprise é fornecido ao sistema a ser projetado, na forma da entrada `damage` de 8 bits sem sinal.
    1. Nota: o valor do dano é calculado por outro sistema em função da quantidade de naves do jogador em campo – mas projetar esse outro sistema está fora do escopo deste projeto.
- A Enterprise é representada por 2 parâmetros: sua energia (`health`) e a proteção de seu escudo (`shield`), ambas representadas como variáveis de 8 bits sem sinal.
- Inicialmente, a nave está com a energia completa e com seus escudos ativos ("*Shields up, Mr. Sulu!*"): logo, tanto `health` como `shield` têm o valor 255.
- O dano causado à nave, em função do valor de `damage`, depende da situação de seu escudo (veja exemplos no final do documento):
    0. Assim que o jogo é iniciado (`reset` = '0'), o dano de entrada é sempre 0. Logo, seu projeto não precisa se preocupar com o valor da entrada `damage` durante essa primeira rodada.
    1. Enquanto não for causado dano de pelo menos 32 em uma única rodada, os escudos ignoram esse dano sem qualquer problema: não há decremento de `health` ou `shield` nesse caso.
    2. Um dano de 32 ou mais em uma rodada, por outro lado, afeta a integridade do escudo: `shield` é decrementado do valor do dano e, a partir de então, esse decremento passa a ocorrer para qualquer valor de dano (mesmo se inferior

a 32); além disso, qualquer dano superior ao valor do escudo passa a ser decrementado de `health`. Por outro lado, o escudo ainda mantém sua capacidade máxima de regeneração: a cada rodada, `shield` é incrementado de 16.
    - Perceba que, nessa situação, um dano menor que 16 em uma rodada permite à Enterprise elevar o valor de `shield`, podendo manter ou recuperar o valor máximo de 255. Além disso, calcula-se o decremento de `health` considerando o valor de `shield` <u>após</u> sua regeneração.
3. Se, a qualquer momento, o dano de entrada for tal que se chega a `shield < 128`, a capacidade de regeneração do escudo se deteriora permanentemente: o incremento de `shield` passa a ser apenas de 2 por rodada
    - Essa situação se mantém mesmo se `shield` voltar a ter um valor de 128 ou mais em rodadas subsequentes ("*Shields collapsing, Captain!*").
4. Se, a qualquer momento antes da rodada 16, o dano à Enterprise for suficiente para reduzir o valor de `health` a 0 ou menos, o jogador vence: o valor mostrado de `health` passa a ser 0, e a saída W passa a valer 1 na borda de subida seguinte do clock. Porém, se a rodada 16 for atingida, o jogador perde mesmo que ainda cause algum dano relevante: nesse caso, a saída L passa a valer 1 após a borda de subida seguinte do clock. Em qualquer dos casos, o sistema interrompe a execução até que o botão `reset` seja ativado (colocado em '1'): essa ação leva o jogador de volta ao estado inicial do jogo, a partir de onde ele pode reiniciar o jogo colocando a entrada `reset` em '0'.
    - Perceba que o jogador pode perder e ganhar ao mesmo tempo: para isso, o dano que entrar na rodada 16 deve ser suficiente para levar `health` a 0 naquela rodada! Nesse caso, a saída WL valerá "11" logo após a borda de subida do clock.

As tabelas no final deste documento ilustram esse comportamento, considerando a inicialização do jogo, cenários de vitória do jogador, cenário de derrota, e cenários de vitória + derrota simultâneas (a nave só é destruída quando se chega à rodada 16).

<u>Intruções para Entrega</u>

A entidade a ser desenvolvida tem a seguinte interface:

```
entity StarTrekAssault is
  port (
    clock, reset: in  bit;                    -- sinais de controle globais
    damage:       in  bit_vector(7 downto 0); -- Entrada de dados: dano
    shield:       out bit_vector(7 downto 0); -- Saída: shield atual
    health:       out bit_vector(7 downto 0); -- Saída: health atual
    turn:         out bit_vector(4 downto 0); -- Saída: rodada atual
    WL:           out bit_vector(1 downto 0)  -- Saída: vitória e/ou derrota
  );
end entity;
```

Você deve acessar o link específico para o projeto dentro do tópico "Projetos" no e-Disciplinas, já logado com seu usuário e senha, o que o levará à página apropriada do juiz eletrônico. Nessa atividade, você pode enviar **um único arquivo** codificado em UTF-8. O nome do arquivo não importa, mas sim a descrição VHDL que está dentro: a entidade na sua solução deve ser idêntica à "`StarTrekAssault`" apresentada neste enunciado ou o juiz

não irá processar corretamente seu arquivo. Caso você use outras entidades para auxiliar na sua solução, coloque-as nesse **mesmo arquivo**.

Quando acessar o link no e-Disciplinas, o navegador abrirá uma janela para envio do arquivo. Selecione-o e envie para o juiz. Jamais recarregue a página de submissão pois seu navegador pode enviar o arquivo novamente, o que vai ser considerado pelo juiz como um novo envio e pode prejudicar sua nota final. Caso desista do envio, simplesmente feche a janela. Depois do envio, a página carregará automaticamente o resultado do juiz, quando você poderá fechar a janela. A nota dada pelo juiz é somente para a submissão que você acabou de fazer.

O prazo para a submissão das soluções no Juiz é aquele definido no e-disciplinas. O Juiz aceitará um número limitado de submissões para cada atividade deste projeto. Sua submissão será corrigida imediatamente e sua nota será apresentada. A maior nota dentre as submissões será considerada. Apesar dessa facilidade de fazer algumas submissões, **não use o juiz como ferramenta de debug**: como boa prática de engenharia, faça seus testbenchs e utilize o GHDL ou o EDAPlayground para validar suas soluções **antes** de submetê-las ao juiz. Se você não fizer isso, há grandes chances de que você desperdice oportunidades de submissão para erros simples, que poderiam ser corrigidos antes da submissão... **O testbench** em si **não deve fazer parte da submissão**, pois o juiz tem seu próprio testbench. Entretanto, para facilitar um pouco sua vida, estamos fornecendo um testbench parcial (com apenas algumas entradas e saídas esperadas).

Dicas:

1. Para facilitar o gerenciamento de incrementos e decrementos nos valores de `health` e `shield`, uma estratégia é criar **registradores especiais**, que se **saturam** sempre que os valores saírem dos limites aceitos. Ou seja, para guardar o valor de `shield`, pode ser usado um registrador que fixa seu conteúdo interno em 0 (zero) caso seu decremento levasse a um valor negativo, e para 255 caso seu incremento levasse a um valor acima de 255; similarmente, para o valor de `health`, pode ser usado um registrador similar que satura em 0 (zero). Caso deseje usar essa estratégia, são fornecidos registradores especiais com essa capacidade: seu conteúdo tem mais do que 8 bits, embora a saída tenha apenas 8 bits (conforme solicitado na `entity` a ser utilizada). Por exemplo, ao fazer um cálculo como "`shield = (shield + regen - damage)`" o resultado pode ser um valor entre -253 (quando `damage = 255, shield = 0, regen = 2`) e +271 (quando `damage = 0, shield = 255, regen = 16`); para capturar os valores nessa faixa diretamente no registrador, sem recorrer a lógica adicional, internamente é usado um inteiro com sinal de 10 bits isso permite representar entre -512 e +511), e então são usadas atribuições condicionais para trocar por 0 os valores negativos, e por 255 os valores maiores que 255. Estratégia similar pode ser útil em outras partes do seu projeto.
2. A forma mais fácil de construir o sistema consiste em separá-lo em: uma unidade de controle (UC), que cuida das transições de estados, e opera nas bordas de subida do clock; e um fluxo de dados (FD), que consiste nos componentes da solução interligados de forma adequada, e atua na borda de descida do clock. Assim, quando a UC envia os sinais de controle logo após uma borda de subida, a FD atualiza os registradores logo em seguida, na borda de descida do clock
3. Sinais definidos como saída de um circuito (i.e., declarados como "**`out`**") não podem ser usados no lado direito de uma atribuição. Por exemplo, o VHDL não permite que seja

feito algo como "`x <= shield(7)`", pois o sinal `shield` é uma saída do circuito. Caso precise fazer alguma operação sobre um dado que está ligado em uma saída, crie um sinal interno intermediário (e.g., "`shieldBuff`"), utilize-o para fazer as atribuições que desejar (e.g., "`x <= shieldBuff(7)`"), e então faça a sua ligação com a saída (e.g., "`shield <= shieldBuff`").

EXEMPLOS DE COMPORTAMENTO ESPERADO: TESTE-OS EM SEU TESTBENCH!

(Re)Inicializando o jogo ("X" significa "don't care")																	
reset	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0
damage:	X	X	X	0	Jogo acontecendo (vide abaixo)			X	X	X	X	X	X	Jogo acontecendo (vide abaixo)			
shield	0	0	0	255				0	255	0	0	0	255				
health:	0	0	0	255				0	255	0	0	0	255				
turn:	0	0	0	1				X	1	X	X	X	1				
W/L	0/0	0/0	0/0	0/0				0/0	0/0	0/0	0/0	0/0	0/0				

Cenário de vitória do jogador ("X" significa "don't care")																	
damage:	0	31	55	16	0	120	17	0	231	1	4	200	X	X	X	X	X
shield:	255	255	216	216	232	128	127	129	0	1	0	0	**0**	0	0	0	0
	regen: 16							regen: 2									
health	255	255	255	255	255	255	255	255	155	155	154	0	**0**	0	0	0	0
turn:	1	2	3	4	5	6	7	8	9	10	11	12	**12**	12	12	12	12
W/L	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	**1/0**	1/0	1/0	1/0	1/0

Cenário de vitória avassaladora do jogador ("X" significa "don't care")																	
damage:	0	255	255	255	**X**	**X**	X	X	X	X	X	X	X	X	X	X	X
shield:	255	16	0	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0
	regen: 16		regen: 2														
health	255	255	18	0	**0**	0	0	0	0	0	0	0	0	0	0	0	0
turn:	1	2	3	4	**4**	4	4	4	4	4	4	4	4	4	4	4	4
W/L	0/0	0/0	0/0	0/0	**1/0**	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0

Cenário de derrota do jogador ("X" significa "don't care")																	
damage:	0	31	32	9	16	4	216	2	112	0	94	1	93	12	0	9	**X**
shield:	255	255	239	246	246	255	55	55	0	2	0	1	0	0	2	0	**2**
	regen: 16							regen: 2									
health	255	255	255	255	255	255	255	255	200	200	110	110	20	10	10	5	**5**
turn:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	**16**
W/L	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	**0/1**

Cenário de vitória + derrota simultâneas do jogador ("X" significa "don't care")																	
damage:	0	31	35	6	1	17	216	2	156	0	59	1	3	82	12	12	**X**
shield:	255	255	236	246	255	254	54	54	0	2	0	1	0	0	0	0	**0**
	regen: 16							regen: 2									
health	255	255	255	255	255	255	255	255	155	155	100	100	100	20	10	0	**0**
turn:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	**16**
W/L	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	**1/1**