Desenvolver elementos autônomos concorrentes para um jogo de terminal

Objetivo

O objetivo do trabalho é desenvolver novos elementos concorrentes e mecânicas para um jogo de terminal já existente. O código-fonte inicial fornecido inclui o controle da interface em modo texto e a mecânica básica de movimentação do personagem. A partir dessa base, você deverá implementar funcionalidades concorrentes utilizando threads/goroutines e canais de acordo com os requisitos estabelecidos nesta especificação.

Descrição do Jogo

O jogo permite que o jogador movimente um personagem por um mapa usando coordenadas x e y. O mapa possui tamanho de 30 linhas por 60 colunas (esse tamanho pode ser modificado). O mapa é representado por uma matriz e cada elemento do mapa é posicionado em uma única célula. O personagem se movimenta nas 4 direções (cima, baixo, esquerda e direita) uma célula por vez. O jogador deve utilizar as teclas do teclado (WASD, E e ESC) para interagir com o jogo.

O mapa é carregado na inicialização a partir de um arquivo de texto. Cada caractere representa um elemento que será desenhado no mapa do jogo. Por exemplo, no código disponibilizado temos os seguintes caracteres (ver arquivo mapa.txt):

- Espaço: deixa a célula vazia no mapa;
- ▤: desenha uma parede de tijolos;
- ♣: desenha vegetação;
- ☺: representa a posição inicial do personagem no mapa.

O posicionamento dos elementos do mapa nas células e a movimentação e ações do personagem são realizados diretamente no sobre a matriz. A função interfaceDesenharJogo() deve ser chamada cada vez que algo mudar na tela. A *struct* Elemento define os elementos do mapa. Veja mais informações no código-fonte.

A implementação atual do jogo não possui *threads* e todas as ações são disparadas pela movimentação do personagem iniciada pelo jogador através do teclado/botões. Sua tarefa é criar novos elementos para o mapa do jogo que funcionem de forma autônoma e concorrente utilizando múltiplas goroutines.

Requisitos Mínimos de Concorrência e Sincronização

Os elementos implementados no jogo devem ser executados de forma **concorrente** e utilizar **canais** para comunicação entre si ou com o personagem. Além disso, é necessário garantir **segurança no acesso a estruturas compartilhadas**, evitando condições de corrida e comportamentos inesperados. Toda comunicação e sincronização do jogo, incluindo exclusão mútua, deve ser realizada utilizando **canais**.

A seguir estão listados os requisitos mínimos que devem ser obrigatoriamente atendidos no projeto:

Novos elementos concorrentes (mínimo 3 tipos)

Você deve implementar pelo menos **três tipos diferentes de elementos autônomos**, cada um com um comportamento próprio, que sejam **executados em goroutines separadas**. Esses elementos devem funcionar de forma **independente da lógica principal do jogo**, ou seja, seu comportamento deve continuar ocorrendo mesmo que o jogador não esteja interagindo diretamente com eles.

Os elementos devem apresentar **comportamentos visíveis e distintos no mapa**, como movimentar-se periodicamente ou de forma aleatória; alternar entre estados (por exemplo, ligado/desligado, visível/invisível); surgir e desaparecer após um intervalo de tempo; patrulhar áreas do mapa ou perseguir o jogador; reagir à presença do jogador ou de outros elementos.

Cada elemento deve possuir **uma representação visual própria** (símbolo e cor) e **interagir com o ambiente ou com o personagem**, podendo, por exemplo: bloquear caminhos ou alterar o mapa; causar efeitos sobre o personagem (dano, cura, bônus, etc.); ativar/desativar portais, armadilhas ou outros mecanismos; iniciar diálogos ou disparar eventos quando o jogador estiver próximo.

Exclusão mútua

Todas as seções críticas do jogo devem ser protegidas para evitar condições de corrida. Exemplos de situações que podem exigir exclusão mútua incluem:

- Alterações simultâneas no mapa (matriz de elementos);
- Atualização da interface de forma concorrente via interfaceDesenharJogo();
- Modificação de estado do jogo ou de elementos concorrentes.

A proteção deve ser feita utilizando canais apensa (é vedado o uso de qualquer outro mecanismo de sincronização adicional disponível no Go).

Comunicação entre elementos por canais

Pelo menos um dos elementos implementados deve **trocar mensagens com outro(s) elemento(s) ou com o personagem por meio de canais**, em vez de acessar diretamente estruturas compartilhadas. Situações possíveis incluem:

- Um inimigo que recebe comandos de movimento por meio de um canal;
- Um item que aguarda um “sinal” para aparecer no mapa;
- Um elemento oculto que é ativado apenas após uma mensagem de outro componente do jogo.
- Um inimigo que recebe uma notificação quando o personagem entra em sua área de vigilância e muda de comportamento.

Escuta concorrente de múltiplos canais

Pelo menos um elemento do jogo deve ser capaz de **ouvir dois ou mais canais ao mesmo tempo**, reagindo a diferentes tipos de eventos simultaneamente. Exemplos:

- Um personagem que escuta comandos do jogador ao mesmo tempo em que monitora eventos do ambiente, como alertas ou armadilhas sendo ativadas.
- Um mecanismo que alterna entre dois comportamentos distintos dependendo do canal que receber mensagem primeiro;
- Um inimigo que alterna entre patrulhar e perseguir o jogador, dependendo de sinais recebidos.

Comunicação com timeout

Ao menos um componente do jogo deve utilizar **um canal com tempo limite de espera para receber uma mensagem**. Se nenhuma mensagem for recebida dentro do tempo esperado, um comportamento alternativo deve ser executado. Exemplos:

- Um portal que se fecha automaticamente após alguns segundos se o jogador não entrar;
- Uma armadilha que aguarda a ativação por tempo limitado;
- Um diálogo com NPC que é encerrado se o jogador não responder em tempo.

Entrega

O trabalho pode ser realizado em grupos de até 3 integrantes desde que a equipe esteja cadastrada como “grupo” no Moodle. A entrega deverá ser feita como um arquivo “.zip” contendo os arquivos fonte desenvolvidos, bem como informações para sua compilação e execução. Escreva também um pequeno relatório explicando os novos elementos e interações implementadas. Atente para os prazos de entregas e apresentação definidos no Moodle.

Critérios de Avaliação do Trabalho

Critério	Pontos
O programa funciona corretamente e sem erros ou travamentos	2.0
Foram implementados ao menos 3 tipos de elementos concorrentes autônomos com comportamentos visíveis e distintos no mapa	1.5

Há uso de canais para comunicação e sincronização entre elementos	1.0
Pelo menos um elemento escuta múltiplos canais e isso é demonstrável	1.0
Pelo menos um elemento utiliza canais com timeout de forma testável	1.0
Há controle de exclusão mútua nas regiões críticas do jogo utilizando canais	1.0
O relatório descreve claramente os elementos criados e o emprego de canais para realizadas sincronização e comunicação	0.5
Os alunos demonstram domínio sobre a implementação e são capazes de responder perguntas sobre o funcionamento e decisões tomadas.	2.0