DEATHread STARuby

Geovane, Henrique

Universidade Estadual do Centro Oeste do Paraná

25 de novembro de 2011

Sumário

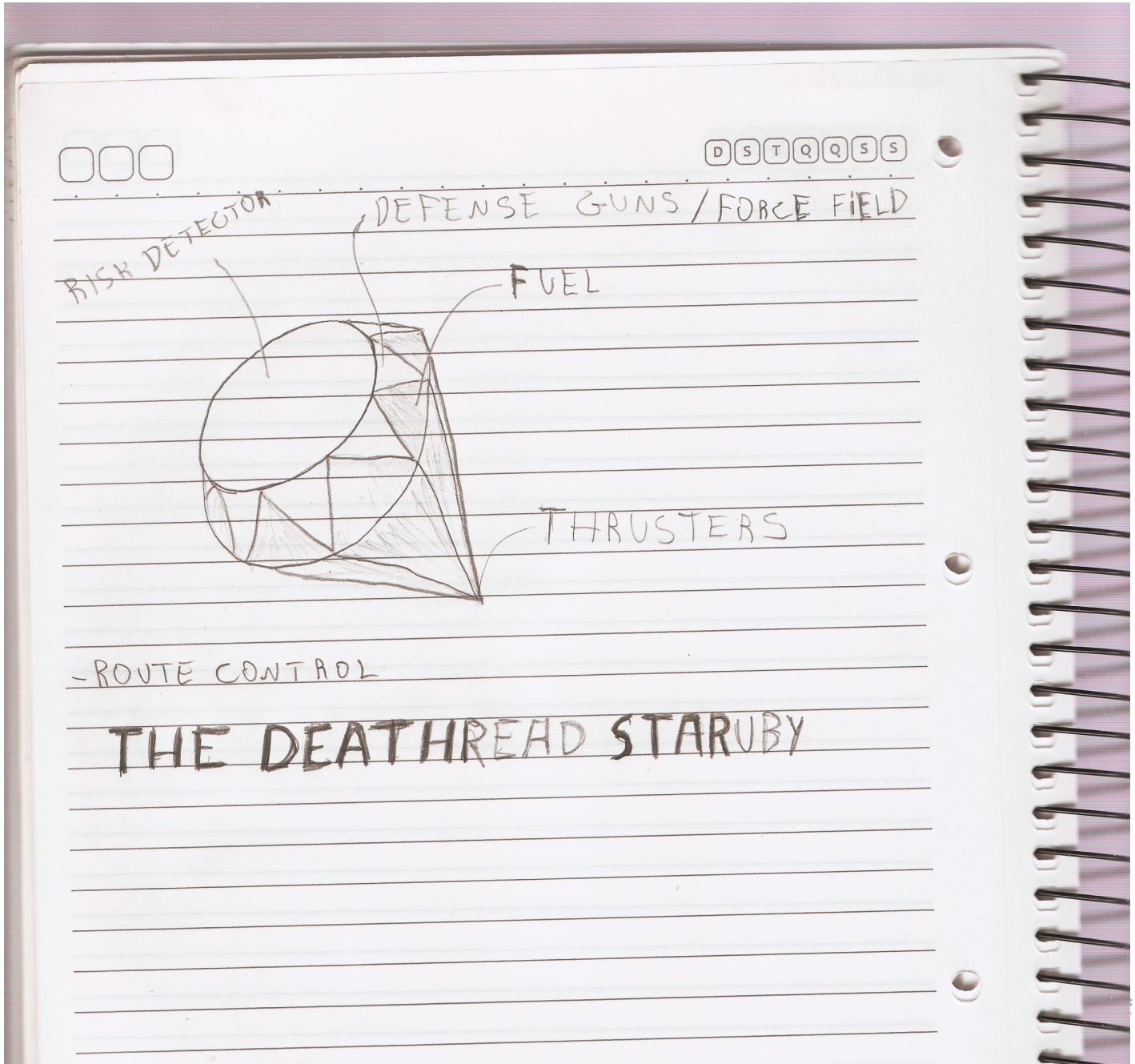
Sumário

Sumário

Sumário

Sumário

Sumário


D S T Q Q S S
RISK DETECTOR
DEFENSE GUNS / FORCE FIELD
FUEL
THRUSTERS
-ROUTE CONTROL
THE DEATHREAD STARUBY

Problema escolhido

O problema escolhido é um simulador simples destinado a, obviamente, simular uma nave espacial em viajem. Os recursos principais da nave são separados em módulos, e cada módulo é executado por uma Thread.

Contexto

- A nave possui uma classe que representa um sensor, com o status de recursos como energia, combustivel e dano.
- Ao inicial a nave, os módulos são ligados, e cada Thread que representa a execução de um módulo começa a ser executada.

Contexto

- A nave possui uma classe que representa um sensor, com o status de recursos como energia, combustivel e dano.
- Ao inicial a nave, os módulos são ligados, e cada Thread que representa a execução de um módulo começa a ser executada.

Por que usar Threads no problema?

Os modulos precisam notificar ao sensor o quanto estão gastando de recursos, ou se a nave recebeu algum dano, e aí se encontra a solução pelas Threads, já que todos devem notificar os sensores de forma concorrente.

Threads: explicações

As Threads servem para separar a aplicação a ser executada em várias linhas de execução, de forma que estas linhas seja executadas concorrentemente e sincronizadamente.

Threads: explicações

As Threads servem para separar a aplicação a ser executada em várias linhas de execução, de forma que estas linhas seja executadas concorrentemente e sincronizadamente.

Threads no ruby

No ruby, as Threads são implementadas a nivel de aplicação, até mesmo porque é a maquina virtual do ruby que vai gerenciar a concorrencia das threads.

Pseudo Código principal

Como foi visto no pseudocódigo, foi nescessário sincronizar o acesso aos sensores. Já que todos os modulos vão notificar os sensores concorrentemente, se isso não for sincronizado, dados não consistentes poderão ser colocados nos sensores.

Sobre o sistema

Para tornar a notificação dos sensores Thread-safe (poder ser chamado por diversas threads sem causar inconsistencia nos dados), foi necessário implementar um semáforo em que cada thread passa por esse semáforo para notificar os sensores.

Sobre o sistema

Para tornar a notificação dos sensores Thread-safe (poder ser chamado por diversas threads sem causar inconsistencia nos dados), foi necessário implementar um semáforo em que cada thread passa por esse semáforo para notificar os sensores.

Sobre o sistema

O semáforo no caso, é uma classe chamada Mutex (mutual exclusion). Sendo assim, é criada uma instancia da classe Mutex, e essa instancia é chamada por cada Thread para sincronizar a parte devida do codigo.

Desenvolvimento

O desenvolvimento foi feito na maior parte do tempo remotamente, compartilhando o espaço de desenvolvimento por meio de um repositorio no github, com debates via messenger.

Sobre o sistema

Os testes foram feitos basicamente por testes de unidade, assim, a cada classe criada, era testado a classe individualmente, e, só posteriormente, integrar as classes para executar o problema.

Conclusão

O uso de Threads demonstrou algumas dificuldades, como por exemplo, entender como a maquina virtual faz a concorrencia das Threads; foi visto, por exemplo, que ao chamar uma Thread pelo metodo “run”, a aplicação não executa nenhuma outra Thread enquanto a execução desta não terminar, sendo nescessário usar o método “join” para repartir o tempo da CPU para diversas Threads que estão executando loops.