UM FRAMEWORK PARA APOIO AO ENSINO DE ALGORITMOS DE BUSCA EM ÁRVORE DE SOLUÇÕES PARA PROBLEMAS DE INTELIGÊNCIA ARTIFICIAL

Marco Simões, MSc.
marcosimoes@fib.br

Mario Jorge Pereira, Bel.
mariojpereira@hotmail.com

Agenda

- Motivação
- Objetivos do Projeto
- Problemas de Busca
- Arquitetura do *Framework*
- Exemplo de Uso
- Resultados
- Considerações Finais
- Trabalhos Futuros

Motivação

- Para desenvolver aplicações com IA é necessário:
  - Conhecer algoritmos e técnicas de IA
  - Modelar o problema de forma a permitir o uso de IA
  - Implementar o algoritmo do agente
- Alunos de graduação demonstram dificuldades com o conhecimento de programação de computadores
- Dificuldade para abordagem prática da IA devido a etapa de implementação do algoritmo

Objetivos do Projeto

- Especificar um *framework* para desenvolvimento de agentes solucionadores de problemas de busca.
- Permitir o uso dos algoritmos de busca de forma simplificada.
- Atender a demanda acadêmica de ferramentas de apoio ao ensino do projeto de agentes inteligentes.

Problemas de Busca

- Caracterizados por:
  - Espaço de Estados
  - Estado inicial
  - Estados finais (objetivos)
  - Ações
  - Função Custo do Caminho (g)
  - Função Heurística (h)

Arquitetura do *Framework*

Controle de ações

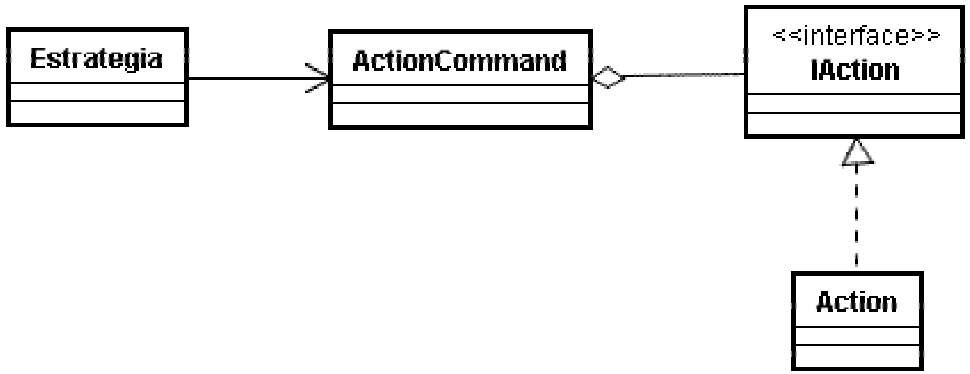


Diagrama classes parcial (Controle de ações)

- Permitir a execução das ações sem a necessidade de conhecê-las previamente.
- Padrão de projeto *Command*

Estratégias

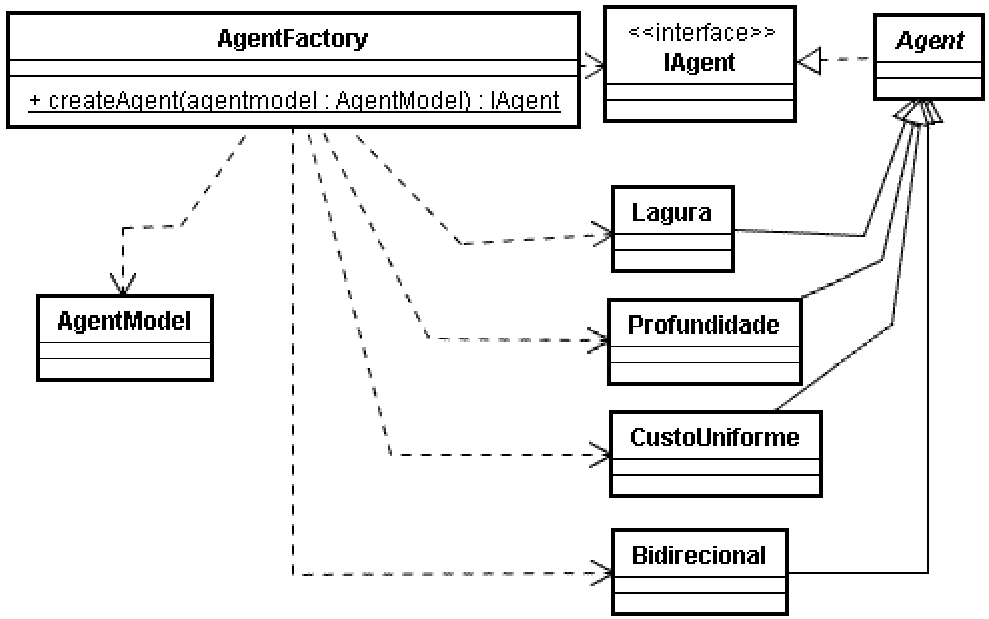


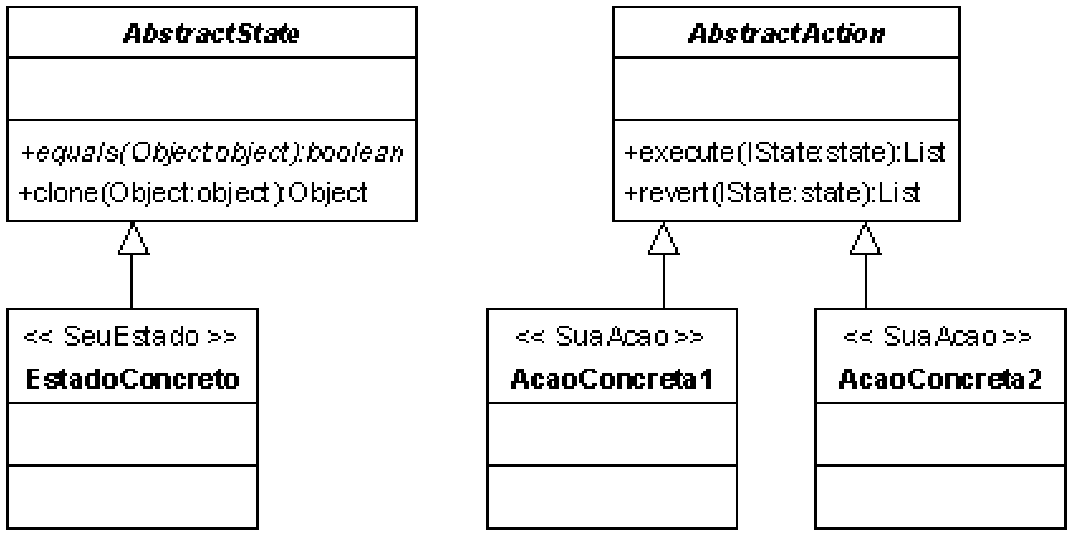
Diagrama de classes parcial (estratégias)

- Permitir o mesmo tratamento para qualquer estratégia.
- Padrão de projeto *Factory Method*

Outros módulos

- Exportação
  - Responsável por exportar a árvore de busca gerada pelos algoritmos para formato XML
- Núcleo
  - Responsável pela configuração dos parâmetros iniciais do framework e por controlar o fluxo de execução dos algoritmos de busca

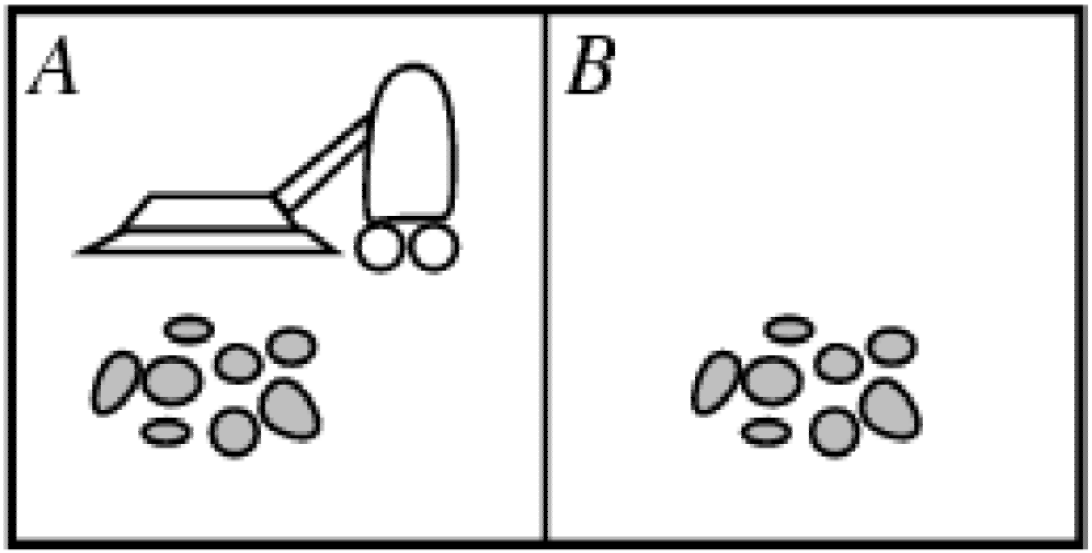
Como estender o *framework*



- *AbstractState*
- *AbstractAction*

Exemplo de Uso: Problema do aspirador de pó

Um agente que controla um aspirador de pó em um mundo com duas salas. O agente possui sensores de sujeira em ambas as salas e através de seus efetuadores pode apenas se mover entre as salas e aspirar.



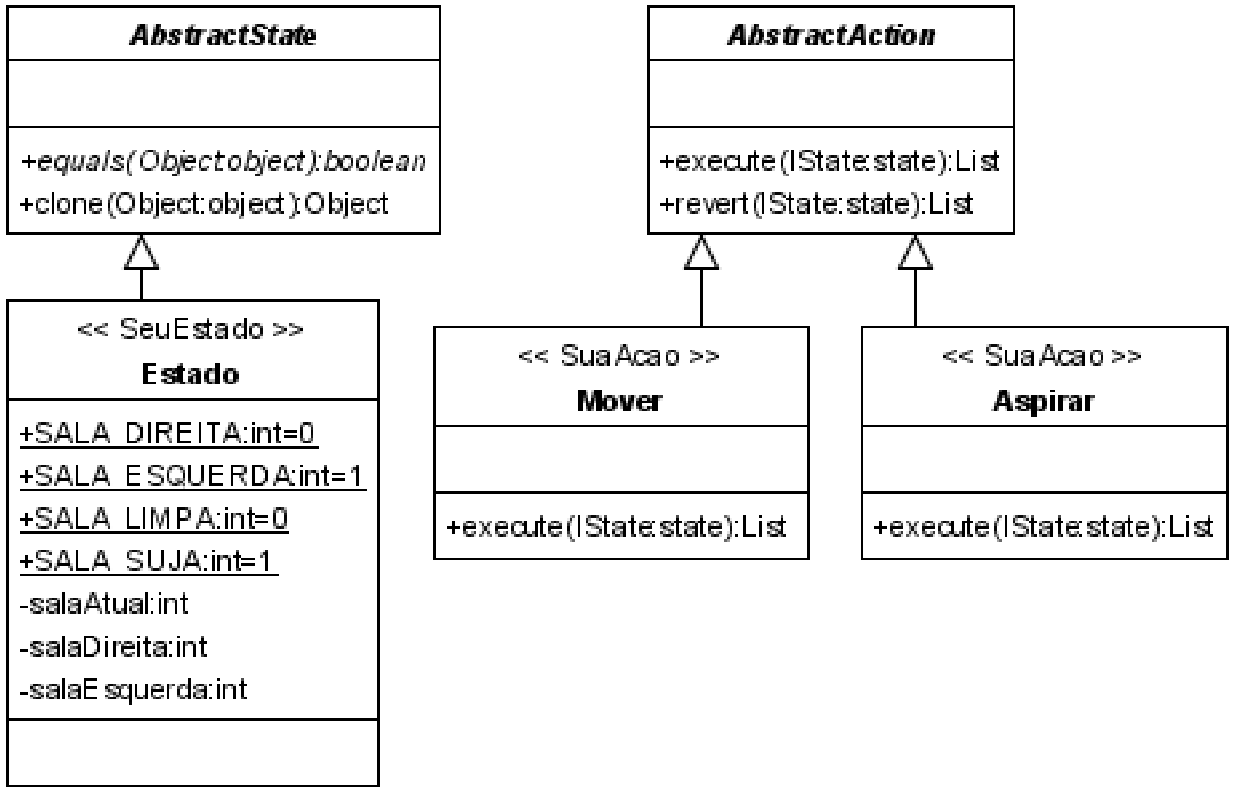
Modelagem

- Estados:
  - Sala atual (Direita ou Esquerda)
  - Situação da Sala direita (Suja ou Limpa)
  - Situação da Sala esquerda (Suja ou Limpa)
- Ações:
  - Mover
  - Aspirar

Modelagem

- Estado inicial:
  - Sala atual = sala esquerda
  - Situação da sala direita = suja
  - Situação da sala esquerda = suja
- Estado Objetivo:
  - Sala atual = sala esquerda ou sala direita
  - Situação da sala direita = limpa
  - Situação da sala esquerda = limpa

Solução


AbstractState
+equals(Object:object):boolean
+clone(Object:object):Object
AbstractAction
+execute(IState:state):List
+revert(IState:state):List
<< SeuEstado >>
Estado
+SALA_DIREITA:int=0
+SALA_ESQUERDA:int=1
+SALA_LIMPA:int=0
+SALA_SUJA:int=1
-salaAtual:int
-salaDireita:int
-salaEsquerda:int
<< SuaAcao >>
Mover
+execute(IState:state):List
<< SuaAcao >>
Aspirar
+execute(IState:state):List

Implementando Estado

```
public class Estado extends AbstractState {

    // ... constantes

    private int salaAtual;
    private int salaDireita;
    private int salaEsquerda;

    //...construtores, get e set.

    public Object clone() { ...}

    public boolean equals(Object arg0) {...}

    public String toString(){...}
}
```

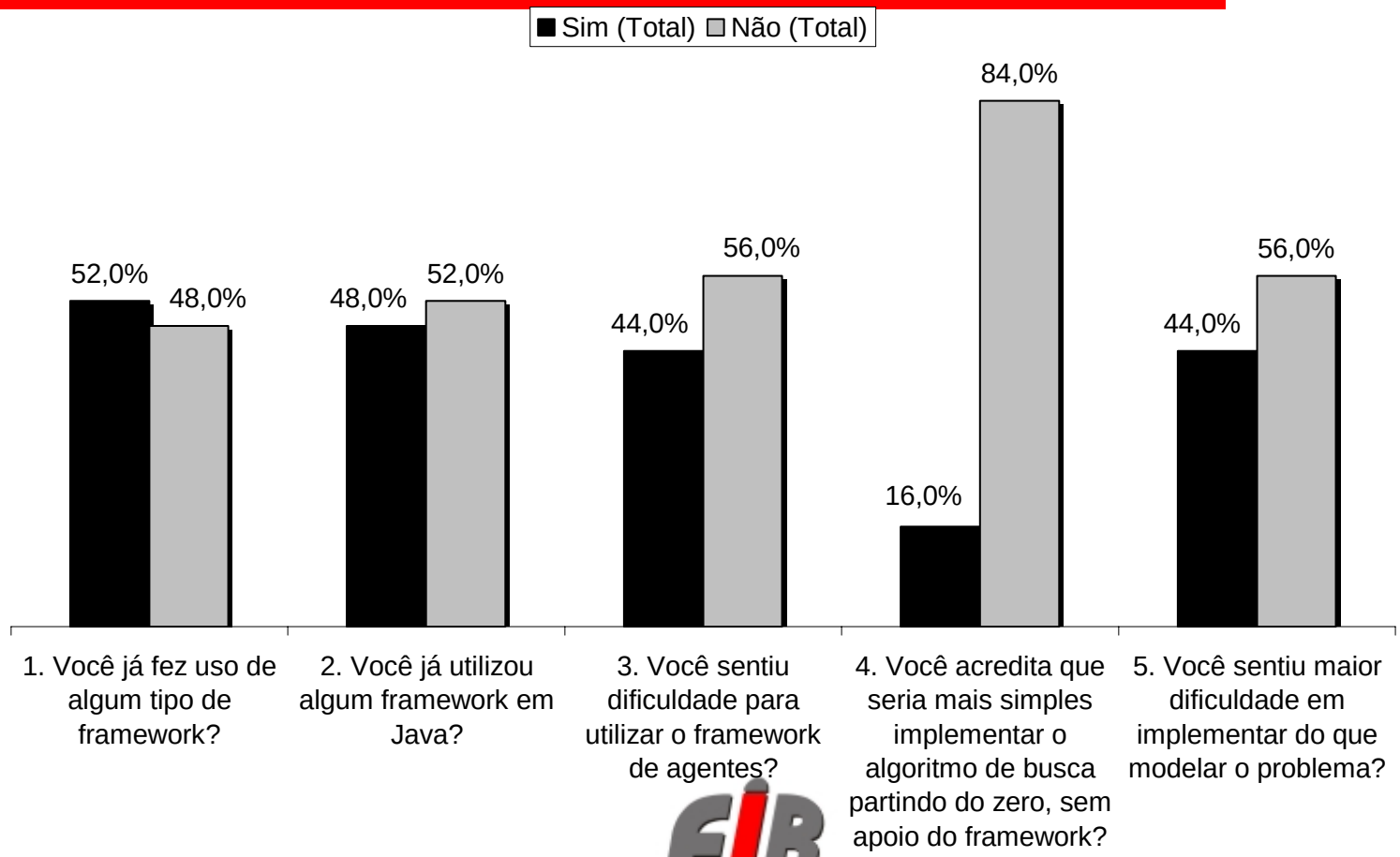
Ação Aspirar

```
public List<IState> execute(IState estado) throws ImpossibleActionException
    {
    List<IState> novosestados = new ArrayList<IState>();
    Estado o = (Estado) estado;
    Estado novoestado = (Estado) o.clone();
    if ( (o.getSalaAtual() == Estado.SALA_DIREITA) &&
                        (o.getSalaDireita() == Estado.SALA_SUJA)){
        novoestado.setSalaDireita(Estado.SALA_LIMPA);
    }else if((o.getSalaAtual() == Estado.SALA_ESQUERDA) &&
                        (o.getSalaEsquerda() == Estado.SALA_SUJA)){
        novoestado.setSalaEsquerda(Estado.SALA_LIMPA);
    }else{
        throw new ImpossibleActionException("Aspirar");
    }
    novosestados.add(novoestado);
    return novosestados;
}
```

Casos Especiais

- Problemas em que o custo das ações é diferente de 1
  - Estender a classe Functions (Núcleo) e redefinir o método *g()*
- Estratégias de Busca Heurística
  - Estender a classe Functions e redefinir o método *h()*
- Busca Competitiva (minimax)
  - Estender a classe Functions e redefinir o método *calculaUtilidade()*
- Objetivo não pode ser um conjunto enumerável de estados
  - Estender a classe Functions e redefinir o método *funcaoObjetivo()*

Resultados da aplicação do framework com alunos



Perguntas qualitativas

- Os alunos auto-avaliaram seu conhecimento de programação como mediano
- Avaliaram que o framework deu uma boa ajuda no desenvolvimento do trabalho
- Consideraram excelente a iniciativa de desenvolvimento do trabalho
- Avaliaram o framework como muito bom (nota 4 numa escala de 0 a 5)

Considerações Finais

- A pesquisa apresentou um ambiente favorável e de grande aceitação de ferramentas desse tipo.
- A pesquisa mostra também que a dificuldade no aprendizado de programação afeta o aprendizado da construção de agentes inteligentes.
- O *framework* permite ao desenvolvedor abstrair as peculiaridades de implementação dos algoritmos e se ater à descrição e modelagem do problema.
- O *framework* permite utilizar e comparar diferentes estratégias de busca com um pequeno esforço de programação.

Trabalhos Futuros

- Adicionar uma interface gráfica para acompanhamento passo-a-passo da construção da árvore de busca pelo agente
- Aplicar o framework com um universo maior de alunos visando confirmar os resultados parciais obtidos
- Permitir o uso de Busca Bidirecional com outras estratégias de busca
- Criação de um shell gráfico para tornar mais fácil a formulação do problema, sem necessidade do uso de linguagem de programação
- Verificar a eficácia da ferramenta para solução de problemas reais