|  |
| --- |
| Universidade federal fluminense |
| Raciocínio |
| Introdução a Sistemas Multiagentes |
|  |
| **Troy Kohwalter** |
| **6/14/2012** |

|  |
| --- |
|  |

Sumário

[Tabela de Figuras 3](#_Toc327891520)

[Capítulo 1 Introdução 4](#_Toc327891521)

[Capítulo 2 Árvore de Decisão 5](#_Toc327891522)

[2.1 Criando uma árvore de decisão 5](#_Toc327891523)

[2.2 Árvore de Decisão Preditiva 7](#_Toc327891524)

[2.3 Indução de Árvores de Decisão 8](#_Toc327891525)

[2.4 Algoritmo ID3 10](#_Toc327891526)

[2.4.1 Entropia de informação e Ganho de informação 11](#_Toc327891527)

[2.5 Vantagens de usar Árvores de Decisões 12](#_Toc327891528)

[2.6 Desvantagens de usar Árvores de Decisões 13](#_Toc327891529)

[Capítulo 3 Árvore de Decisão para agentes BDI 14](#_Toc327891530)

[3.1 Adaptando a Árvore de Decisão 16](#_Toc327891531)

[Capítulo 4 Lógica *Fuzzy* 18](#_Toc327891532)

[4.1 Inferência *Fuzzy* 18](#_Toc327891533)

[4.1.1 Representação de Conhecimento 19](#_Toc327891534)

[4.2 Raciocínio *Fuzzy* 19](#_Toc327891535)

[Capítulo 5 Software Development Manager (SDM) 21](#_Toc327891536)

[5.1 Funcionários 22](#_Toc327891537)

[5.1.1 Papéis 22](#_Toc327891538)

[5.1.2 Atributos 22](#_Toc327891539)

[5.1.3 Especializações 23](#_Toc327891540)

[Capítulo 6 Abordagem Proposta 24](#_Toc327891541)

[6.1 Mapeamento BDI 24](#_Toc327891542)

[6.2 Alterações no jogo SDM 25](#_Toc327891543)

[6.3 Árvores de Decisão por papel 28](#_Toc327891544)

[Capítulo 7 Implementação 30](#_Toc327891545)

[7.1 Introdução ao Unity3D 30](#_Toc327891546)

[7.2 Desenvolvimento 31](#_Toc327891547)

[Capítulo 8 Conclusão 35](#_Toc327891548)

[Referencias Bibliográficas 36](#_Toc327891549)

# Tabela de Figuras

[Figura 1: Exemplo de árvore de decisão 5](file:///D:\SVN\sdm\trunk\Documents\Raciocionio.docx#_Toc327891550)

[Figura 2: Exemplo do cálculo de nó de incerteza 6](file:///D:\SVN\sdm\trunk\Documents\Raciocionio.docx#_Toc327891551)

[Figura 3: Exemplo de cálculo do valor de nó de decisão 7](#_Toc327891552)

[Figura 4: Exemplo de uma árvore de decisão de aprendizagem 8](#_Toc327891553)

[Figura 5: Exemplo de um conjunto de treinamento 9](#_Toc327891554)

[Figura 6: Árvore de decisão do conjunto de teste 10](#_Toc327891555)

[Figura 7: Exemplo de árvore de decisão antes da transformação 17](#_Toc327891556)

[Figura 8: Árvores de decisão geradas com a transformação 17](#_Toc327891557)

[Figura 9: Exemplo de valor fuzzy 18](#_Toc327891558)

[Figura 10: Diagrama de Classe parcial do SDM na sua primeira versão desenvolvido no inicio de 2011. 21](#_Toc327891559)

[Figura 11: Interface do jogo SDM 30](#_Toc327891560)

[Figura 12: Área de trabalho do Unity3D, desenvolvendo o SDM 31](#_Toc327891561)

[Figura 13: Janela de Configuração de tarefas 32](#_Toc327891562)

[Figura 14: Acessando a janela Task Configuration 32](#_Toc327891563)

[Figura 15: Janela ilustrativa do processo de escalonamento de papeis do gerente 33](#_Toc327891564)

[Figura 16: Janela de exibe todas as ações realizadas pelos funcionários 34](#_Toc327891565)

# Introdução

Um sistema multiagente é composto de vários agentes "inteligentes" que interagem com o ambiente. Este tipo de sistema pode ser usado para resolver problemas que são difíceis para um agente individual. O tipo de inteligência usada por estes agentes pode incluir: buscas algorítmicas, procedural, funcional, ou metódica; e abordagem de processamento.

Este trabalho tem como foco a área de raciocínio de um sistema multiagente. No entanto, como não foi encontrado na literatura artigos sobre este tema para sistemas multiagentes, o material utilizado neste trabalho é baseado em técnicas de Inteligência Artificial.

O Raciocínio pode ser visto como uma operação lógica discursiva e mental. Para isso, o intelecto humano utiliza uma ou mais proposições para chegar a conclusão de algo através de mecanismos de comparações e abstrações, e quais são os dados que levam às respostas verdadeiras, falsas ou prováveis. Sendo assim, partindo de premissas chega a conclusões. Algumas técnicas usadas de Inteligência Artificial para raciocínio são árvores de decisão e lógica fuzzy, que serão apresentadas neste trabalho.

Por fim, raciocínio pode ser considerado como um dos integrantes de mecanismos dos processos [cognitivos](http://pt.wikipedia.org/wiki/Cogni%C3%A7%C3%A3o) superiores para formação de conceitos e da solução de problemas, sendo parte do [pensamento](http://pt.wikipedia.org/wiki/Pensamento). Em um sistema multiagente, a racionalização é definida como um processo cognitivo de ver a razão para uma determinada crença, conclusão ou ação. Este processo de racionalização pode ser de dois tipos: Dedutivo e indutivo.

Sendo assim, este trabalho esta dividido da seguinte forma: Capitulo 2 apresenta a base teórica sobre árvores de decisões, mostrando como elas são geradas e como podem ser utilizadas para modelos preditivos e indutivos; Capitulo 3 apresenta um exemplo de utilização de árvores de decisões em um sistema que utiliza agentes BDI; Capitulo 4 faz uma introdução a lógica *fuzzy* e explica como que é o processo de raciocínio *fuzzy*; Capitulo 5 apresenta a primeira versão do jogo SDM, desenvolvido em 2011, que será utilizado como base na implementação do trabalho; Capitulo 6 apresenta a proposta desse trabalho, as modificações realizadas no jogo, e o mapeamento teórico do modelo BDI; Capitulo 7 mostra alguns detalhes de implementação do trabalho, mostrando os pontos que foram alterados no SDM para suportar o que foi proposto; e por fim o Capitulo 8 apresenta a conclusão do trabalho.

# Árvore de Decisão

Árvores de decisão (MORET, 1982) é uma ferramenta visual utilizada para modelar decisões e suas consequências, incluindo possibilidades de ocorrência de eventos, custos de recursos, e utilidade de um determinado resultado. Ela pode ser considerada como um algoritmo determinístico para decidir que variável testar em seguida, baseado nas variáveis já testadas e nos resultados de sua avaliação. Estas árvores são utilizadas em diversas aplicações, como por exemplo, em analises de decisões, identificação de estratégias apropriadas para alcançar um determinado objetivo, e em investigações operacionais.

Figura 1: Exemplo de árvore de decisão

As árvores são representadas por grafos orientados em formato de árvores e tem a característica de representar regras. O grafo de uma árvore de decisão é composto de três tipos de nós distintos: nó de decisão, nó de incerteza e nó terminal. As árvores de decisões normalmente são criadas da esquerda para direita, e possui a característica de que todos os nós da árvore são do tipo *burst*, ou seja, geram novas ramificações e os ramos da árvore nunca convergem. Por esta razão, as árvores podem crescer exponencialmente e possuir repetições. A ilustra um exemplo típico de árvore de decisão, desenhada a mão.

## Criando uma árvore de decisão

Em um artigo disponível em MindTools (“Decision Tree Analysis”, 2012) é feita uma descrição detalhada do procedimento para criar árvores de decisões. O processo inicia com uma decisão que precisa ser realizada, que é a raiz da árvore. A partir desta decisão, gere ramos para cada resultado possível da decisão e, no final dos ramos, considere o resultado. Se o resultado deste ramo for uma tomada de decisão que possui incerteza, coloque o nó de incerteza. Caso o resultado do ramo seja outra decisão, então coloque o nó de decisão, e caso seja um ramo terminal, coloque o nó terminal. A partir destes novos nós gerados na árvore, crie os ramos que representem as possíveis opções. Novamente avalie o resultado de cada ramo e coloque o nó apropriado. Repita o processo até que todos os ramos terminem com o nó terminal.

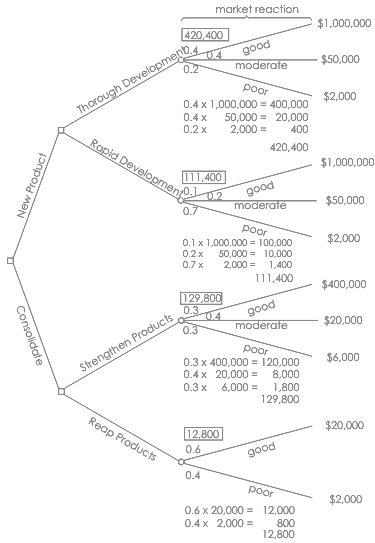
O próximo passo é calcular os valores na árvore. Em todos os nós de incerteza, atribua a probabilidade de entrar em cada ramo resultante. Depois atribui um valor resultante nos nós terminais. Agora é necessário calcular o valor dos resultados de cada ramo através do seu tipo de nó. Este calculo é feito no sentido do nó terminal até a raiz da árvore. Em nós do tipo de incerteza, o valor será multiplicado pela sua probabilidade de ocorrência e depois é feito um somatório destes resultados para ser colocado no valor do nó de incerteza. A ilustra um exemplo deste calculo.

Figura 2: Exemplo do cálculo de nó de incerteza

Para o calculo dos nós de decisão, atribua um valor para cada ramo resultante da decisão. Depois, para cada ramo, subtraia do valor resultante o valor do ramo obtendo como resultado o valor de beneficio dessa decisão. Agora escolha dentro de todos os ramos, o valor que tenha um maior beneficio e coloque este valor no nó de decisão. A ilustra esta etapa. Repita este processo até chegar a raiz da árvore.

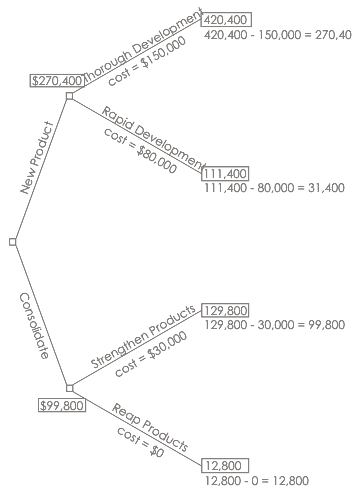


Figura 3: Exemplo de cálculo do valor de nó de decisão

## Árvore de Decisão Preditiva

Árvores de decisão também podem ser utilizadas como um modelo preditivo em que mapeia observações de um determinado fato e / ou item a conclusões de seu valor, sendo assim conhecidas como *Decision Tree Learning*. Este modelo de árvore de decisão é bastante utilizado em estatística, mineração de dados e aprendizado de maquina. Nestas áreas, a árvore de decisão é utilizada como um modelo preditivo onde mapeia observações de um item e conclusões sobre seu valor. Em análise de decisão, a árvore de decisão é usada para explicitamente e visualmente representar decisões e tomadas de decisões.

Uma árvore pode "aprender predições" pela divisão da origem em subconjuntos baseados em testes de valor do atributo. Esse processo é repetido em cada subconjunto derivado em uma forma recursiva chamada de *recursive partitioning*. A recursão é completada quando o subconjunto em um nó possui todos os mesmos valores da variável alvo, ou quando a divisão não gera nenhum novo valor para a predição. Esse processo é chamado de *Top-down induction of decision tree* (QUINLAN, 1986) e é um exemplo de um algoritmo guloso. A ilustra um exemplo deste tipo de árvore.

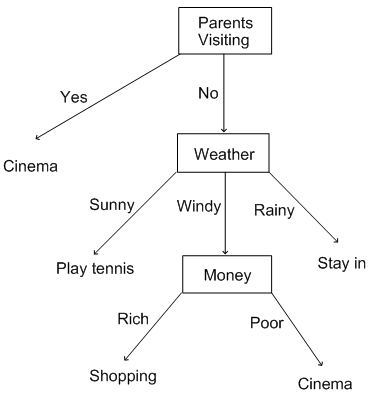


Figura 4: Exemplo de uma árvore de decisão de aprendizagem

## Indução de Árvores de Decisão

Árvores de decisões também podem ser utilizadas para realizar induções. Para isso, utiliza-se uma base que representa um universo de objetos que são descritos em termos de coleções de atributos. Cada atributo possui alguma importância para o objeto e é limitado a um conjunto de valores discretos e mutuamente exclusivos. Cada objeto no universo pertence a um conjunto de classes mutuamente exclusivas. Outro componente importante é o conjunto de treinamento (*training set*) de objetos cuja classe é conhecida. A tarefa de indução é desenvolver uma regra de classificação que possa determinar em que classe cada objeto pertence de acordo com seus valores de atributos.

Como mencionado, a regra de classificação dos objetos será expressa como uma árvore de decisão. A ilustra um exemplo de um pequeno conjunto de treinamento que usa atributos de tipo: aspecto{sunny, overcast, rain}, temperatura{cool, mild, hot}, umidade{high, normal}, windy{true, false}.

****

Figura 5: Exemplo de um conjunto de treinamento

Cada valor dos atributos é mostrado na figura, junto com a classe em que o objeto pertence. A árvore de decisão que corretamente classifica cada objeto do conjunto de treinamento é dada na . As folhas da árvore de decisão são os nomes das classes, e os outros nós representam testes baseados em atributos com um ramo para cada possível resultado. Para classificar um objeto, inicia-se pela raiz da árvore, avalia o teste, e siga o ramo de acordo com o resultado. O processo continua até que a folha seja encontrada, onde o objeto é afirmado pertencer a classe com o nome da folha. Se os atributos são adequados, é sempre possível construir uma árvore de decisão que corretamente classifica cada objeto do conjunto de treinamento, e usualmente existem várias árvores de decisões que classifica corretamente.



Figura 6: Árvore de decisão do conjunto de teste

A essência da indução é mover além do conjunto de treinamento, como por exemplo, construir uma árvore de decisão que corretamente classifique objetos que também não pertençam ao conjunto de treinamento. Para conseguir isto, a árvore de decisão precisa capturar alguns relacionamentos significantes entre a classe do objeto e seus valores de atributos.

## Algoritmo ID3

Uma abordagem de fazer a tarefa de indução de árvores seria gerar todas as possíveis árvores decisões que classifique corretamente o conjunto de treinamento e selecionar o mais simples deles. No entanto, o número de árvores pode ser muito grande, tornando esta abordagem apenas viável para pequenas tarefas de indução.

O algoritmo ID3 (QUINLAN, 1986) foi concebido para a outra extremidade do espectro, onde existem muitos atributos e o conjunto de treinamento contem muitos objetos. No entanto é necessária uma árvore de decisão razoavelmente boa sem ter que fazer muito esforço de cálculo computacional. O algoritmo constrói esta árvore de decisão, no entanto a abordagem utilizada não garante que árvores melhores não tenham sido negligenciadas.

A estrutura básica do ID3 é iterativa. Um subconjunto do conjunto de treinamento, chamado de janela, é escolhido randomicamente e uma árvore de decisão é gerada a partir dele. Essa árvore classifica corretamente todos os objetos desta janela. Todos os outros objetos no conjunto de treinamento são classificados usando a árvore. Se a árvore fornecer a resposta correta para todos estes objetos então é correta para todo o conjunto de treinamento e o processo é terminado. Caso contrario, uma seleção dos objetos classificados incorretamente é adicionado a janela e o processo continua. Desta forma, árvores de decisões corretas são encontradas depois de poucas iterações, inclusive em um caso de estudo realizado onde o conjunto de treinamento contém 30.000 objetos descritos com termos de até 50 atributos. Evidencia empírica sugere que a árvore de decisão correta é usualmente achada mais rapidamente por este método do que formando a árvore através do conjunto completo de treinamento.

O algoritmo ID3 pode ser resumido da seguinte forma:

1. Pegue todos os atributos não usados e conte seus valores de entropia relativos a amostras de teste.
2. Escolha um atributo cuja entropia é mínima, ou a ganho é máximo.
3. Crie um nó contendo este atributo.

### Entropia de informação e Ganho de informação

Criar uma árvore de decisão é uma questão de escolher qual atributo testar em cada nó da árvore. Existem duas medidas utilizadas para este tipo de decisão que são utilizadas pelo ID3, *entropia* e o *ganho* de informação que é calculada usando *entropia*.

*Entropia*, de acordo com Mitchell (1997) caracteriza a impuridade de uma coleção arbitraria de exemplos. Em outras palavras, entropia é a medida de duvida, imprevisibilidade. *Entropia* é utilizada no algoritmo para determinar qual nó da árvore será subdividido no próximo passo. Quanto maior o valor de entropia, maior o potencial de melhoramento. A formula para calcular a *entropia* é:

 E(S) = - \sum^{n}_{j=1}  f_{S}(j) \log_{2} f_{S}(j) 

* E(S) é a entropia do conjunto S.
* n é o numero de diferentes valores do atributo em S.
* fs(j) é a frequência (proporção) do valor j no conjunto S.
* Quando entropia é igual a zero, identifica um conjunto perfeitamente classificado.

O *ganho* é calculado para estimar o ganho produzido por uma divisão de um atributo e quantifica a melhoria da entropia pela divisão de mais um atributo, sendo que um *ganho* maior é melhor. A equação do *ganho* é:

 G(S, A) = E(S) - \sum^{m}_{i=1}  f_{S}(A_{i}) E(S_{A_{i}})

* G(S,A) é o ganho do conjunto S depois de ter dividido sobre o atributo A.
* E(S) é a entropia de informação do conjunto S.
* m é o numero de diferentes valores do atributo A em S.
* fs(Ai) é a frequência dos items possuindo Ai como valor de A em S.
* Ai é o i-ésimo possível valor de A.
* SAi é o subconjunto de S contento todos os itens onde o valor de A seja Ai.

## Vantagens de usar Árvores de Decisões

* Utiliza um modelo caixa branca: Dado um resultado, a explicação deste resultado é facilmente replicada.
* Pode ser combinado com outras técnicas de decisão.
* Transparente: Explicita todas as alternativas e mapeia cada uma até sua conclusão.
* Especificidade: Permite definir valores específicos para o problema, decisões, e resultados de cada decisão.
* Natureza Compreensiva: Permite analisar as consequências de cada decisão possível.
* Fácil de usar: prove ilustração gráfica do problema e várias alternativas de forma simples, sem precisar de explicações.
* Resiliencia: Foca na relação entre vários eventos e replica o curso natural dos eventos, por essa razão permanece robusta e com pouca margem para erros.

## Desvantagens de usar Árvores de Decisões

* Complexidade: Apesar de ser fácil usabilidade e compreensão, preparar uma árvore de decisão, principalmente uma com varias ramificações, é um trabalho complexo e que consome bastante tempo.
* “Complicada”: Este problema ocorre quando a árvore é muito grande, fazendo com que seja difícil de apresenta-la. Desenhar uma árvore de decisão também requer que seja refeita varias vezes por não ser possível predizer o numero de ramificações que cada nó terá. No entanto alguns softwares já existem para resolver este ultimo problema.
* Muita informação: Uma das grandes vantagens pode se tornar uma desvantagem. Listar todas as possíveis ramificações pode fazer com que o processo de decisão seja mais lento na analise, pois precisa percorrer todos os ramos.
* Duplicação: Existe a possibilidade de sub-árvores serem duplicadas em caminhos diferentes.
* Regressão: Árvores de decisões não possuem ramos convergentes.

# Árvore de Decisão para agentes BDI

Diversas abordagens emergiram como candidatos ao estudo de sistemas orientados a agentes. Uma dessas arquiteturas vê o sistema como um agente racional tendo algumas atitudes mentais de crença, desejos e intenções (BDI), representando respectivamente a informação, a motivação e estados deliberativos do agente. Essas atitudes mentais determinam o comportamento do sistema e são críticos para alcançar desempenho adequado quando a deliberação está sujeita a recursos limitados.

Em um trabalho proposto por Rao (1995), é provido uma arquitetura unificante para um tipo particular de agente, BDI. Primeiro é estabelecido a necessidade das crenças, desejos e intenções para o sistema agir apropriadamente em uma classe de domínio de aplicação caracterizado por varias limitações e requisitos práticos. Algumas das importantes características deste tipo de domínio são:

1. Em qualquer instante de tempo, existem potencialmente inúmeras maneiras em que o ambiente pode evoluir e se alterar.
2. Em qualquer instante de tempo, existem diversas ações distintas ou procedimentos que o sistema pode executar.
3. Em qualquer instante de tempo, existem diversos objetivos distintos que o sistema necessita alcançar.
4. As ações ou procedimentos que melhor alcançam os diversos objetivos são dependentes do estado do ambiente e são independentes do estado interno do sistema.
5. O ambiente só pode ser percebido localmente.
6. O numero de ações que podem ser realizadas estão dentro de um limite aceitável para a taxa de alteração do ambiente.

Uma forma de modelar o comportamento de tal sistema, dadas as características 1 e 2, é por uma estrutura de árvore, onde cada ramo da arvore representa um caminho de execução. Cada nó na estrutura representa certo estado do mundo, e cada transição uma ação primitiva feita pelo sistema, um evento ocorrido pelo ambiente, ou ambos. Se diferenciar as ações realizadas pelo sistema e os eventos realizadas pelo ambiente, os dois tipos distintos não determinísticos se manifestam em dois tipos distintos de nós, que são denominados de nós de decisão e nós de incerteza (ou chance, probabilístico). Respectivamente representam as opções disponíveis pelo sistema e as incertezas do ambiente.

Nesta forma de modelo, é possível identificar os objetivos do sistema por caminhos particulares da estrutura de árvore de decisão, onde cada caminho informa que objetivo ele realiza e os benefícios e *payoff* de ter percorrido o caminho, caso necessário. Como o sistema precisa agir, é necessário selecionar as ações ou procedimentos apropriados para executar dentro do conjunto de possibilidades.

Dentro do domínio de características citado acima, existem pelo menos dois tipos de entrada de dados que são necessários para a tomada de decisão do sistema. Primeiro, o sistema precisa ter informação sobre o estado do ambiente, mas este conhecimento precisa ser ganho através de varias "sondagens" do ambiente e que a informação seja atualizada adequadamente depois de cada sondagem. Este tipo de informação pode ser considerada como a crença do sistema, podendo ser implementado por uma variável ou banco de dados. Assim, a crença pode ser vista como o componente informativo do sistema.

Segundo, é necessário que o sistema também tenha informações sobre os seus objetivos que necessitam ser alcançados, ou as prioridades e *payoffs* de alcançar estes objetivos. É possível que estes objetivos, ou prioridades sejam criadas em tempo de execução. Este componente do sistema é os desejos, que podem ser pensados como a representação do estado que motiva o sistema.

Levando em consideração as alterações no ambiente, e assumindo que as alterações significantes possam ser calculadas instantaneamente, é possível limitar a frequência de reconsiderações nas tomadas de decisão e alcançar um equilíbrio entre muita reconsideração e não reconsideração. Para isso funcionar, é necessário incluir um componente no sistema que represente o curso de ação escolhido atualmente. Esse componente de estado adicional é chamado de intenções do sistema. Em essência, as intenções capturam o componente deliberativo do sistema.

## Adaptando a Árvore de Decisão

Como já foi discutido no capitulo anterior, informalmente as árvores de decisão consistem de nós de decisão, de incerteza e terminais, e incluem funções probabilísticas que mapeia os nós de incerteza e função de *payoff* para os nós terminais. Para agentes BDI, é transformado este tipo de árvore de decisão para um modelo equivalente que represente as crenças, desejos e intenções como relacionamentos separados de acessibilidade sobre um conjunto de possibilidades. Essa transformação prove uma base alternativa para casos em que possui dados insuficientes nas probabilidades e *payoffs*.

Para fazer esta transformação, inicie com a árvore de decisão completa, que contem todos os caminhos possíveis. Dada essa árvore, começando da raiz até as folhas, para cada ramificação de nós de incerteza, cria-se uma árvore idêntica a original com a exceção que o nó de incerteza é removido e a ramificação é conectada ao sucessor do nó de incerteza. Esse processo é repetido até que todos os nós de incertezas sejam removidos. Esses passos geram um conjunto de árvores de decisões, onde cada árvore consiste em apenas nós de decisão e nós terminais, e cada árvore correspondendo a uma diferente possibilidade do estado do ambiente. Depois, aplica-se a função de *payoff* normalmente em cada árvore.

As possibilidades resultantes deste processo contem dois tipos de informação, representadas pelas possibilidades e seus valores de *payoff* para cada caminho. Fazendo a divisão destas informações, consegue representar as crenças pela informação de possibilidades e desejos pela de payoff. Caso deseje simplificar, remova todos os caminhos com payoff zero da árvore de desejos.

Dada a árvore de decisão e as resultantes da transformação acima, um agente agora é capaz de fazer uso da função de deliberação para decidir o melhor caminho de ações. Esse caminho pode ser representado como uma nova árvore de decisão, correspondendo as intenções do agente. A Figura 7 mostra um exemplo de árvore de decisão, e aplicado a transformação nessa árvore obtém os resultados ilustrados na Figura 8. Note que, em crenças, aparece a opção do agente se aposentar, mas o agente não possui nenhum desejo em se aposentar. Para simplificar a árvore, ou caso não possua os valores corretos de probabilidade e *payoff*, é possível alterar a árvore contendo apenas valores 0 e 1, como exemplo, algo é desejado ou não desejado.



Figura 7: Exemplo de árvore de decisão antes da transformação

****

Figura 8: Árvores de decisão geradas com a transformação

# Lógica *Fuzzy*

A lógica *fuzzy* (ZADEH, 1988) é uma extensão da lógica booleana que admite valores intermediários entre verdadeiro e falso. Isso significa que o valor lógico *fuzzy* é um valor entre 0 e 1. Com isso, enquanto na lógica booleana algo é verdadeiro ou falso, já na lógica *fuzzy* algo pode ser verdadeiro, falso, ou um valor intermediário. Estes valores intermediários, na lógica *fuzzy*, podem ser tratados e avaliados, tornando possível o uso de conceitos não quantificáveis, como por exemplo: quente, morno, feliz, apático. A Figura 9 ilustra um exemplo de valor intermediário na lógica *fuzzy*.

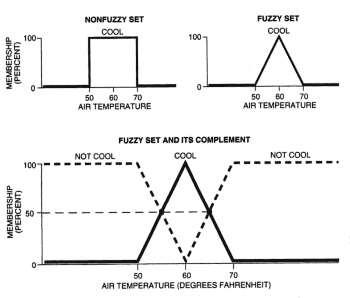


Figura : Exemplo de valor fuzzy

## Inferência *Fuzzy*

Assim como a lógica booleana, é possível fazer inferências usando a lógica *fuzzy*. Expressões do tipo *se x então y* podem ser adaptadas para que adotem noções difusas. Essa adaptação nada mais é que permitir que a sentença, os valores *x*  e *y* sejam difusos. Com essa adaptação, se torna mais fácil interpretar as inferências e implementar sistemas baseados em conhecimento humano. Um exemplo clássico de inferência *fuzzy* é: Se a temperatura é *alta* e a pressão é *alta* então o fluxo de combustível deve ser *pequeno*.

Outra forma de fazer inferência *fuzzy*, também conhecida como raciocínio *fuzzy*, é aplicando regras da lógica booleana como, por exemplo, *modus ponens* e *modus tolens*. Um exemplo usando estas regras e sabendo que "A implica B" de forma nítida, e tendo apenas um valor difuso de A, é possível calcular o valor de B. O Raciocínio *fuzzy* será explicado em mais delates na seção 4.2.

### Representação de Conhecimento

Para conseguir fazer inferências, é necessário que o conhecimento seja representado de alguma forma. A forma mais popular é representar o conhecimento através de regras. Um exemplo é dado abaixo, onde A e B são valores linguísticos definidos por conjuntos *Fuzzy* de universos de discursos X e Y.

*Se X é A então Y é B*.

* Uma regra é também chamada de implicação *Fuzzy*.
* "*X é A*" é a premissa da regra.
* "*Y é B*" é a consequência ou conclusão da regra.

## Raciocínio *Fuzzy*

O Raciocínio *fuzzy* pode ser dividido em 5 etapas:

1. Transformação das variáveis do problema em valores *fuzzy*, ou *fuzzificação*.
2. Aplicação dos operadores *fuzzy*.
3. Aplicação da implicação.
4. Combinação de todas as saídas *fuzzy* possíveis.
5. Transformação do resultado *fuzzy* em um resultado nítido, a *defuzzificação*.

No primeiro passo, para cada valor de entrada associamos uma função de pertinência, que permite obter o *grau de verdade* da proposição, também conhecido como *fuzzificação*.

* Determinar o grau de pertinência de cada conjunto (proposição).
* Limitar o valor da entrada entre 0 e 1.

O segundo passo é aplicar os operadores *fuzzy*, assim como os operadores da lógica nítida. Os operadores usados na lógica *fuzzy* são *and* e *or*, conhecidos como operadores de relação. Na lógica *fuzzy*, estes operados são utilizados para definir o grau máximo e mínimo de pertinência do conjunto.

O terceiro passo é aplicar o operador de implicação, usado para definir o peso do resultado e remodelar a função, ou seja, o terceiro passo consiste em criar a hipótese de implicação. Um exemplo segue abaixo:

* Serviço é excelente *ou* atendimento é rápido *então* pagamento é alto.

No quarto passo ocorre a combinação de todas as saídas em um único conjunto *fuzzy*, algo semelhante ao processo de união e interseção da teoria dos conjuntos.

O quinto e ultimo passo no processo do raciocínio *fuzzy*, é a *defuzzyficação* que consiste em retornar os valores e obter um valor numérico dentro da faixa estipulada pela lógica *fuzzy*. Um exemplo simples que demonstra o processo de pertinência do raciocínio *fuzzy* seria:

* A é identificado como "o tomate está vermelho".
* B como "o tomate está maduro".

Então: se é verdade que "o tomate está vermelho", é também verdade que "o tomate está maduro". Essa seria um exemplo pensado na lógica tradicional onde:

* Fato: x é A;
* Regra: se x é A então y é B;
* Conclusão: y é B

Esta regra aplica um conceito aproximado. Porem se pensarmos desta forma e temos a mesma regra de implicação "se o tomate está vermelho, então ele está maduro", e sabemos que o tomate está *mais ou menos* vermelho, é possível inferir que o tomate está mais ou menos maduro. Ou seja:

* Fato: x é A’ (quase A)
* Regra: se x é A então y é B
* Conclusão: y é B’ (quase B)

O conceito de *fuzzyficação* funciona da seguinte forma: se A’ está próximo de A (situação inicial) e B’ está próximo de B (situação inicial). A, A’, B e B’ fazem parte do conjunto universo, chegando assim ao paradigma do raciocínio *fuzzyano*, também chamado de *modus ponens* generalizado.

# Software Development Manager (SDM)

No jogo *Software Development Manager* (KOHWALTER *et al.*, 2011), o jogador possui uma equipe de funcionários que são utilizados para desenvolver softwares solicitados por clientes. Cabe ao jogador decidir estratégias de desenvolvimento e definir os papéis de cada funcionário da equipe para o desenvolvimento do software. Os softwares requisitados pelos clientes possuem requisitos que devem ser respeitados durante o desenvolvimento. Quando um software é concluído e entregue ao cliente, é realizada uma avaliação de qualidade do software e o pagamento de conclusão do projeto, ajustado de acordo com o resultado da avaliação. A Figura 10 ilustra um diagrama de classe simplificado do SDM. Neste capitulo será apresentado brevemente alguns elementos presentes no SDM, caso deseje saber mais detalhes do jogo, a monografia, o artigo publicado no *SBGames* e o download do jogo estão disponíveis no site <http://www.ic.uff.br/~tkohwalter/sdm>.

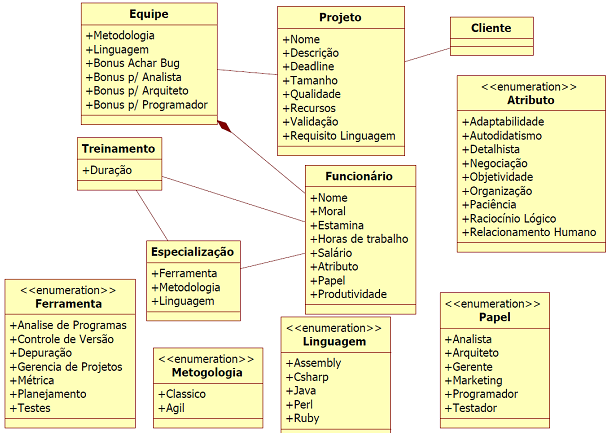


Figura 10: Diagrama de Classe parcial do SDM na sua primeira versão desenvolvido no inicio de 2011.

## Funcionários

Os funcionários em SDM representam a mão de obra trabalhadora do jogador. São eles que desempenham as tarefas de planejamento, desenvolvimento e controle de qualidade do software através de papéis que são atribuídos pelo jogador. Além das tarefas que podem ser atribuídas, os funcionários possuem algumas características que são utilizadas para determinar a afinidade do funcionário a exercer cada tarefa.

### Papéis

As tarefas mencionadas anteriormente são executadas pelos papéis que um funcionário pode desempenhar no jogo. Quando um papel for escolhido para um funcionário, este dedicará todo o seu tempo de trabalho exercendo a função desejada. Diferente de algumas empresas onde um funcionário pode exercer diferentes papéis, em SDM foi decidido que cada funcionário poderá apenas exercer um papel por vez. No entanto, o jogador pode a qualquer momento alterar o papel desempenhado por um funcionário.

Na concepção do SDM, foram escolhidos seis tipos diferentes de papéis que podem ser atribuídas a um funcionário. Estes papéis são de analista, arquiteto, gerente, marketing, programador e testador. O objetivo da utilização destes seis papéis é expandir as possibilidades das funções que podem ser desempenhadas pelos integrantes da equipe. Cada um destes papéis possui uma função especifica que contribui para o desenvolvimento do software.

### Atributos

Alem de papéis e cargos, os funcionários possuem atributos que influenciam no seu desempenho de trabalho. No entanto, esses atributos não são diretamente relacionados a cada papel, como por exemplo, um atributo de gerenciamento que seria utilizado para o papel de gerente. Para tornar o SDM ainda mais realista e ter mais um diferencial dos demais jogos, o sistema adotado para atributos foi à utilização de características humanas de uma pessoa. Esses atributos humanos são os atributos que o jogador pode consultar para formular uma avaliação de desempenho que um funcionário tem em determinado papel. Porem o mapeamento destes atributos humanos para os atributos de desempenho não é informado ao jogador. Com isso, ele precisa utilizar o bom senso e experimentos para conseguir descobrir quais atributos são mais importantes para cada função. No entanto, isto não significa que apenas alguns atributos influenciaram no desempenho. Na realidade, todos eles influenciam no desempenho de cada papel, porém em graus diferentes. Estes atributos são: adaptabilidade; autodidatismo; detalhista; negociação; objetividade; organização; paciência; raciocínio lógico; e relacionamento humano.

### Especializações

Outro aspecto que cada funcionário possui são as especializações. Estas especializações podem ser em relação a ferramentas, linguagens de programação ou técnicas de Engenharia de Software (ES). As especializações em linguagens de programação refletem quais as linguagens que o funcionário está apto a trabalhar. Especializações em técnicas são utilizadas para determinar se o funcionário está familiarizado com determinadas metodologias de ES que são adotadas pela equipe de desenvolvimento. Em ambos os tipos de especializações, caso o funcionário não esteja apto a trabalhar de acordo com os critérios escolhidos pelo jogador, este funcionário sofrerá penalidades em sua produtividade. Para contornar este problema, o jogador terá a opção de treinar o funcionário na especialização desejada.

Diferente das demais, a especialização de ferramentas é utilizada para auxiliar o funcionário a exercer sua função. Para cada papel existente no SDM, existem especializações relacionadas que são levadas em consideração, caso o funcionário as possua, para aumentar o rendimento produtivo dele. A falta deste tipo de especialização não levará a nenhuma penalidade para o funcionário.

# Abordagem Proposta

A proposta deste trabalho é expandir o jogo SDM, introduzindo elementos apresentados na disciplina de Introdução a Sistemas Multiagentes. No Capítulo 5 foi feita uma breve explicação do jogo, e neste capitulo será apresentado algumas alterações na estrutura do jogo para que possa introduzir conceitos de raciocínio de agentes nos funcionários quando desempenham os diferentes papeis.

O raciocínio será feito através de árvores de decisão, onde o funcionário usará árvores de decisão para cada papel que exercer. Com estas árvores, espera-se que o jogo fique mais diversificado, já que os mesmo funcionários podem interagir no ambiente de trabalho de forma diferente quando for jogar novamente o jogo ou quando continuar o jogo através de novos projetos.

## Mapeamento BDI

Como já foi explicado no capitulo 5, o jogador possui uma equipe de desenvolvimento. A empresa do jogador na qual possui a equipe de desenvolvimento pode ser tratada como uma organização os funcionários da equipe como agentes. Para simplificar, é adotado que todos os funcionários do jogador possuem os mesmos *desejos* que a organização, que é concluir com sucesso o desenvolvimento do software corrente. Na finalização do software, e quando este é entregue ao cliente para ser avaliado, os funcionários adquirem experiência e moral caso seja concluído com sucesso. Caso não tenha concluído o projeto antes do deadline, o moral dos funcionários é afetado negativamente e adquirem um menor valor de experiência. Motivos que podem levar a este cenário são: equipe pequena para o escopo do projeto; falta de treinamentos levando a um rendimento inferior ao esperado; decisões erradas no processo de desenvolvimento; falta de recursos que resulta em redução na produção dos funcionários e pode levar a decisão de saírem da empresa; escolha de papeis que não refletem o perfil dos funcionários.

A *crença* foi mapeada no jogo como o conhecimento do andamento do desenvolvimento do software, e o único funcionário que é autônomo é aquele que exerce o papel de gerente da equipe. Sendo assim, o gerente utiliza informações do andamento do projeto para poder tomar decisões. Os demais funcionários agem como agentes escravos, obedecendo as ordens passadas pelo gerente ou jogador. Atualmente a *crença* do gerente é 100% precisa, ou seja, as informações que ele possui são exatas e não possui margem de erro. A única informação que o gerente não garante ser verdade é em relação à quantidade total de *bugs* no sistema, pois só tem conhecimento dos que foram reportados pelos testadores. Existem planos para que adicione elementos que influenciem na percepção do gerente, permitindo ter variações nas estimativas. Sendo assim, um gerente pode fazer estimativas equivocadas, como estimar errado o tamanho do software ou avaliar os perfis de cada membro da equipe a fim de distribuir tarefas.

No jogo, cada dia marca a execução de uma ou mais ações dos funcionários, não permitindo ações parciais. Com isso, as *intenções* são mapeadas como estas ações que foram executadas, já que são atômicas. As ações executadas pelos papeis do jogo são armazenadas e podem ser analisadas através de um *Log* dentro do jogo. Este *Log* é explicado em mais detalhes na seção 7.3. A estrutura de armazenamento é feita através de listas, onde cada funcionário possui uma lista que contem todas as ações geradas por ele e em cada ação possui a descrição do caminho percorrido na árvore de decisão.

## Alterações no jogo SDM

Para introduzir as alterações propostas, será necessário fazer algumas mudanças no funcionamento do SDM. Uma dessas mudanças é no comportamento do funcionário quando exerce um papel. Na versão apresentada do jogo, os papeis desempenham tarefas fixas, ou seja, dado um papel, a tarefa executada será sempre a mesma, inclusive na forma em que será executa, alterando apenas os valores de saída de acordo com a situação e funcionário. Com isso, todos os papeis serão revisitados e elaborados de tal forma que incorpore o que foi proposto, com a exceção do papel de marketing, devido a sua pequena importância no impacto de desenvolvimento. Com as alterações propostas, os papeis deixaram de ser um *enumeration* e viram classes (cada papel é uma classe) contendo métodos de: *trabalhar*, *árvore de decisão*, funções que representam as *folhas* (nós terminais) da árvore de decisão, *gerar ação*, atualizar *histórico*, e outras funções auxiliares que cada papel utiliza no jogo.

Começando com o papel de *Analista*, o novo modelo para este papel exerce três categorias distintas de tarefas:

* Elicitação de requisitos e validação: Levantamento dos requisitos do sistema e validação por meio de reviews ou protótipos.
* Especificação de requisitos: Especificação dos requisitos descobertos pela elicitação.
* Geração de casos de testes de Aceitação.

Esta ultima tarefa é consequência de outra alteração no modelo do SDM, onde *bug* de software foi subdividido em quatro categorias:

* Unitário;
* Integração;
* Sistema;
* Aceitação.

O papel de *Arquiteto* sofre alterações em suas tarefas, sendo divididas em:

* Geração de casos de teste: Podendo ser de Integração e Sistema.
* Evolução: Trabalhar na arquitetura do software.
* Prototipação: Gerar protótipo para ser utilizado em conjunto com a Validação do analista.

O *Gerente* sofre várias alterações, já que este papel tem uma autonomia superior aos demais. Com isso suas tarefas são:

* Recursos Humanos: Administrar os membros da equipe, em relação a contratações, demissões, treinamento.
* Foco da iteração: Decidir o foco que será adotado na iteração, podendo ser:
  + Análise: Os funcionários serão alocados de tal forma em que a analise dos requisitos e validação sejam prioritários.
  + Desenvolvimento: Dará prioridade no desenvolvimento do software.
  + Qualidade: Prioridade em encontrar e reparar *bugs*, e melhorar a qualidade do código.
  + Balanceado: Não possui prioridades, mas tentará manter a distribuição de papeis e tarefas de forma balanceada.
* Carga de trabalho: Decidir a quantidade de horas que cada funcionário trabalhará e se o funcionário estará sobre pressão para produzir mais.
* Autonomia: Esta não é uma tarefa em si, mas irá fazer com que o gerente não necessite de comandos constantes do jogador. Caso o jogador não forneça nenhuma tarefa ao gerente, o gerente irá tomar suas próprias decisões.

O papel de *Programador* e *Testador* sofrem algumas alterações que são relacionadas entre si. Antigamente o papel responsável por remover os *bugs* do software era o testador, já que todos os *bugs* eram automaticamente detectados quando introduzidos no software. Agora, os *bugs* presentes no software não são mais visíveis ao jogador. Com isso, cabe ao papel de Testador em testar o software e reportar os *bugs* encontrados. Outra alteração é que o Testador perde a sua função de reparar os *bugs*, o que passa a ser uma tarefa do Programador.

Com isso, o *Programador* possui as seguintes tarefas:

* Reparar: Concertar os *bugs* reportados.
* Evoluir: Desenvolver o software.
* Refatoração: Refatorar o código.

Estas tarefas podem ser executadas de três maneiras:

* Ad-hoc: Executa a tarefa sem ter feito um planejamento.
* Draw-Code: Planeja antes de executar.
* Test-Driven: Planeja, executa e verifica se o que executou está adequado.

E por fim, o *Testador* agora só tem uma função, que é procurar e reportar os *bugs* encontrados. Porem pode ser executado de duas maneiras:

* Ad-hoc: Procura por *bugs* pela método de força-bruta.
* Executar casos de testes.

## Árvores de Decisão por papel

As árvores de decisão foram geradas a partir das alterações explicadas na seção 6.2, onde estas novas tarefas, na maioria dos casos, representam folhas da árvore. Devido ao tamanho das árvores de decisão, elas serão apresentadas em um arquivo aparte, que será entre junto a este trabalho. Nesta seção será apenas feita a descrição dos elementos que influenciam as árvores de decisão.

Pelo fato do SDM ser um jogo educacional, que está sendo desenvolvido para o Mestrado e orientado por Leonardo Murta e Esteban Clua, algumas propostas elaboradas no capitulo anterior ainda não foram debatidas e aprovadas por restrição de tempo. Quando for tal caso, será mencionado o que foi realizado diferente do proposto.

Começando pela árvore de decisão do analista, o primeiro nível da árvore se depara com uma decisão que vai determinar que ramo seguir. Esta primeira decisão é resultante de uma decisão do jogador ou do gerente. No ramo de Elicitação e Validação, é feita uma decisão de acordo com a disponibilidade de protótipos. Caso tenha um protótipo pronto, então será feita a elicitação e validação com o protótipo. Esses protótipos são criados pelo arquiteto quando este executa a tarefa de prototipação.

Assim como o analista, o primeiro nível da árvore de decisão do arquiteto depende da decisão do jogador ou do gerente. A tarefa de prototipação disponibiliza um protótipo para ser consumido pelo analista durante a tarefa de Elicitação e Validação. No próximo capitulo, serão explicadas melhor estas decisões.

A árvore de decisão do gerente é basicamente decidida dependendo das decisões do jogador, que restringe a autonomia do gerente. Caso não tenha nenhuma restrição, o gerente irá tomar decisões dependendo do estado de desenvolvimento. Devido a restrições, apenas a árvore de decisão que decide qual o foco na etapa de desenvolvimento foi criada. Para fins dessa disciplina, um método foi criado para que o gerente distribua tarefas para os funcionários de acordo com o foco de desenvolvimento (analise, codificação, qualidade, balanceado).

No caso do programador, a árvore foi dividida em duas planilhas, para reduzir o tamanho e facilitar o entendimento. A primeira planilha apenas informa qual tarefa que o programador irá executar, podendo ser de evolução ou de reparo. A segunda planilha indica como que o programador irá executar a tarefa, sendo utilizada pelos dois tipos de tarefas. O modo de refatoração não foi implementado, em termos de realizar alguma alteração no software.

Apesar do testador ter uma árvore de decisão na planilha, ela não foi implementada pois depende de um outro elemento que ainda não foi desenvolvido no SDM. Este elemento é *Casos de Teste*. Atualmente o testador efetua testes de todos os tipos, sequencialmente, de forma Ad Hoc.

Por fim, o funcionário de marketing não possui árvore de decisão, pois sua função é menos significante em relação aos outros papeis e ainda não foi dedicado tempo para sua elaboração. Por causa disso, o papel de marketing possui as mesmas tarefas da versão original do SDM.

# Implementação

Neste capitulo será mostrado alterações realizadas no SDM aplicando os conceitos apresentados no Capitulo 6. A Figura 11 ilustra a interface do SDM. Este capítulo está divido da seguinte forma: apresentação da plataforma de desenvolvimento utilizada na criação do SDM; Alterações realizadas no SDM; e por fim os resultados obtidos com estas alterações.



Figura : Interface do jogo SDM

## Introdução ao Unity3D

O SDM foi desenvolvido utilizando a ferramenta de desenvolvimento de jogos Unity3D (HIGGINS, 2010). O Unity3D é um ambiente de produção que facilita a criação de jogos, abstraindo elementos técnicos durante o desenvolvimento. Com esta abstração, o Unity3D permite que o desenvolvedor se concentre apenas na criação do jogo sem precisar se preocupar com elementos do motor. A Figura 12 exibe a interface de desenvolvimento do Unity3D. A imagem superior no canto esquerdo exibe a visão de desenvolvedor e a imagem inferior mostra como o jogo é visualizado enquanto está sendo jogado. As três colunas a direita são utilizadas para o desenvolvimento, onde a primeira, da esquerda para direita, exibe todos os objetos contidos no cenário. A segunda informa todos os elementos disponíveis para o desenvolvimento como, por exemplo, figuras, texturas e *scripts*. A última coluna mostra a visão de inspetor, que contém as características do objeto selecionado, permitindo que alterações sejam feitas.

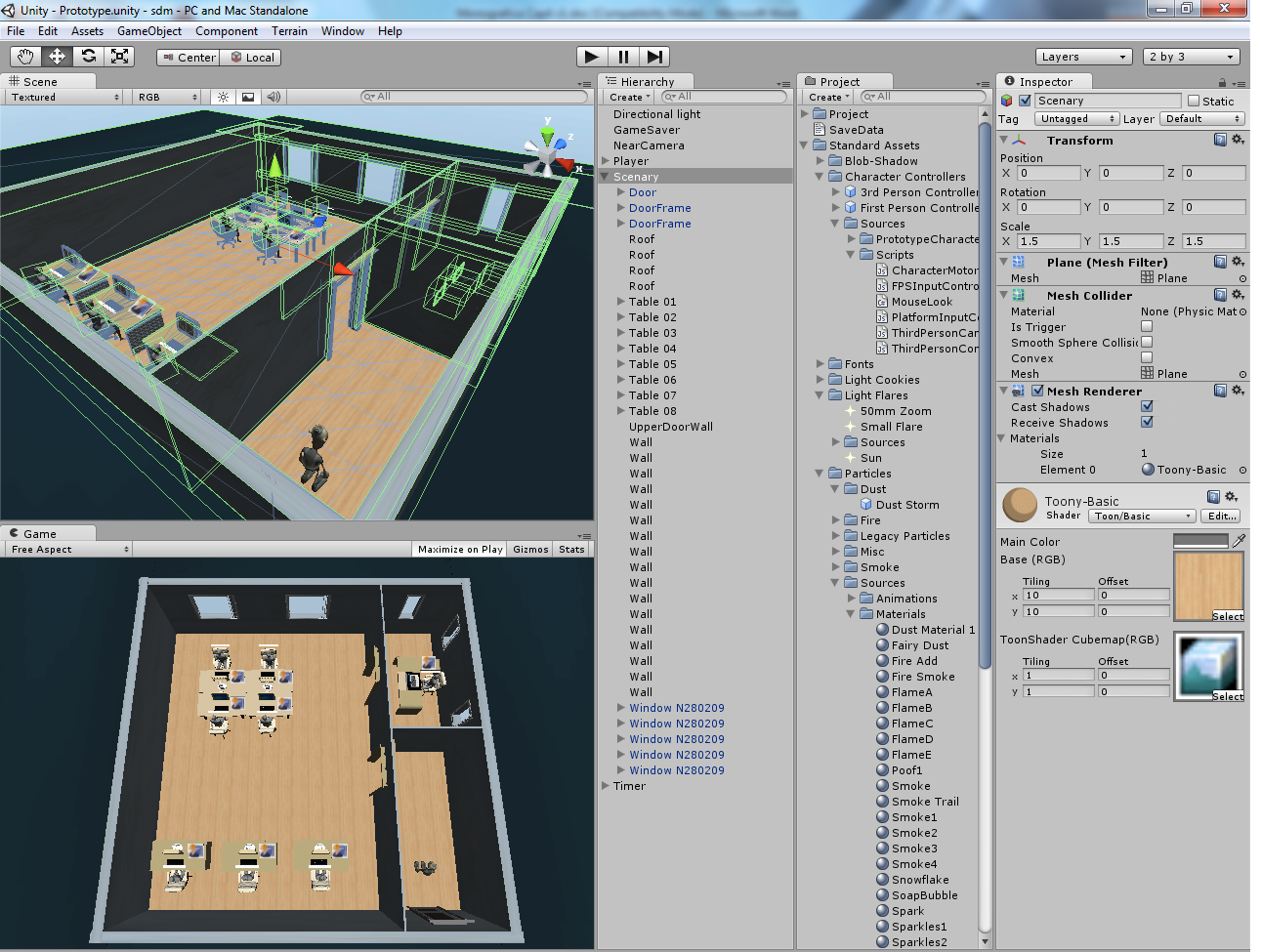


Figura 12: Área de trabalho do Unity3D, desenvolvendo o SDM

## Desenvolvimento

Como foi dito anteriormente na seção 6.3, algumas decisões utilizadas nas árvores de decisões precisam ser alimentadas por uma fonte externa, podendo ser tanto do jogador quanto do gerente. No caso do gerente, estas decisões são transparentes para o jogador, e o processo será mostrado mais a frente. Já no caso do Jogador, estas decisões podem ser configuradas por papel para cada funcionário da equipe. A Figura 13 exibe a janela de configuração, onde é dividida em campos pertencentes aos papeis que possuem decisões externas nas suas árvores de decisão. Todos os elementos mostrados são configuráveis pelo jogador ou podem ser configurados pelo gerente da equipe, caso o jogador queira que o gerente seja autônomo. A Figura 14 mostra o caminho de acesso desta janela, que é através da barra de controle dos integrantes da equipe, localizada no canto superior da interface.



Figura 13: Janela de Configuração de tarefas

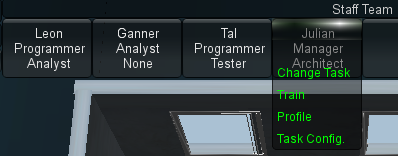


Figura 14: Acessando a janela Task Configuration

Outra forma de configurar as tarefas de cada funcionário é através do gerente. O gerente, quando decide qual papel que cada funcionário desempenhará também decide quais tarefas que aquele funcionário vai realizar no papel designado. Esta decisão depende em que modo de desenvolvimento será adotado. Por exemplo, se o gerente ou o jogador decidirem que a etapa de desenvolvimento adequada no momento é de qualidade, então o gerente ao designar papeis para os funcionários, os programadores terão a tarefa de reparar o software. Vale lembrar que isso só acontece caso o jogador de autonomia ao gerente. Caso contrario, é necessário que o jogador faça a configuração manual de cada funcionário. A escolha de papel e da tarefa é predefinida pelos modos existentes. O que se altera em cada sessão de jogo, ou caso tenha alguma variação na equipe, é a ordem em que os papeis serão alocados. Para cada funcionário da equipe, é avaliado seu perfil, em relação aos atributos humanos, e decidido os dois papeis que se encaixam melhor de acordo com o perfil. Depois de efetuar essa avaliação na equipe, inicia-se a distribuição de tarefas para cada funcionário tentando respeitar os perfis avaliados.

A Figura 15 exibe uma imagem ilustrativa do processo. A tabela mostrada representa, em cada linha, chamada de *Slot0#*, um funcionário. As duas colunas que possui papéis representam os dois papéis que possuem atributos mais elevados. Para fins de verificação, os valores resultantes da avaliação de perfil para cada papel são exibidos nas colunas seguintes, obedecendo a ordem de analista, arquiteto, programador e testador. Os botões na parte inferior da janela são botões de simulação. O primeiro, "*Clean*", remove os papeis de todos os funcionários, mantendo apenas o papel de gerente. O botão "*Set roles*" define os papéis de cada funcionário de acordo com a tabela. O botão "*Recalculate*" recalcula a tabela. Os quatro botões a direita que terminam com "*Mode*" escalona a equipe de acordo com a prioridade escolhida, definindo papeis e tarefas. Esta janela só tem a finalidade de exibir os resultados de cada ação e não estará disponível para o jogador. Todo o processo de escalonamento, como dito antes, é transparente para o jogador. Atualmente a avaliação dos papeis é 100% precisa. No entanto, existe a proposta em fazer com que esses valores sejam afetados pela percepção e proficiência do gerente corrente.

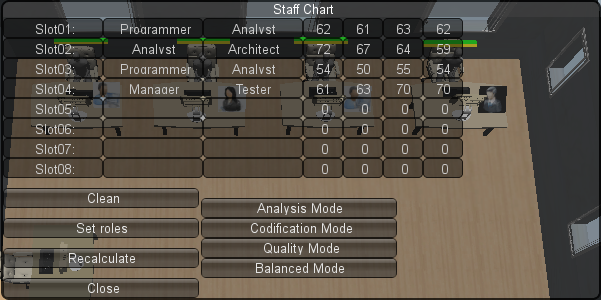


Figura 15: Janela ilustrativa do processo de escalonamento de papeis do gerente

Por fim, todas as ações executadas por um funcionário geram eventos de ações que são registrados e armazenados para consulta. A Figura 16 ilustra a janela contendo este "log", onde diz qual foi a ultima ação executada por cada funcionário, e caso selecionada, exibe a ação em mais detalhes, informando quem que a executou, quando foi executada, qual foi a papel do funcionário que a executou, qual foi a tarefa executada, se teve influencias externas e uma descrição textual da ação, dizendo por extenso o caminho percorrido da árvore de decisão. A única influencia externa que é registrada no momento é da relação arquiteto-programador. A influencia de "pressão" vinda do gerente não foi implementada até o presente momento, mas pode ser configurada pelo jogador. É possível percorrer a lista de ações através de botões presentes na janela de ações. Estas duas ultimas janelas apresentadas são acessadas apenas pelo menu exibido no canto direito da tela quando a tecla "ESC" é apertada.



Figura 16: Janela de exibe todas as ações realizadas pelos funcionários

# Conclusão

Ao decorrer deste trabalho foi feita uma apresentação teórica de técnicas de Inteligência Artificial que podem ser utilizado em sistemas multiagentes, junto com uma apresentação de u trabalho proposto que utiliza árvores de decisão para agentes BDI. Logo em seguida foi apresentada a base desse trabalho, o jogo SDM, e proposto alterações que seriam realizadas a fim de integrar alguns dos conceitos apresentados. Por fim, foram mostradas as alterações que foram realizadas no jogo para dar suporte ao que foi proposto.

Com implementação de árvores de decisão nos papeis propostos pelo SDM, a dinâmica do jogo foi alterada, permitindo que funcionários façam ações diferentes em cada partida do jogo e reajam de acordo com o ambiente de desenvolvimento do software. Apesar das árvores de decisões utilizadas neste trabalho serem simples em relação a tamanho e tipos de decisões, e do raciocínio do gerente da equipe tanto no processo de desenvolvimento quanto no escalonamento de tarefas, a diferença já é perceptível no jogo. As árvores tornam o jogo mais complexo e ao mesmo tempo oferece um grau maior de autonomia da equipe por causa do raciocínio embutido no gerente. Outro beneficio das árvores é que um mesmo papel pode exercer diferentes tarefas dependendo da situação, mas não simultaneamente, o que antes não era possível já que as tarefas de um papel eram fixas e não podiam ser alteradas. Sendo assim, permitindo um maior numero de tarefas que podem ser executadas por um mesmo papel e também como que as tarefas serão executadas.

Um exemplo disso é o papel de programador, que na versão anterior do jogo ele era responsável por apenas desenvolver o software. Agora, com árvore de decisão, ele possui a tarefa de desenvolver e de reparar o software, que pode ser determinada pelo jogador ou pelo gerente, mas é decidido pelo próprio funcionário, através da árvore, como que ele exercerá a tarefa (*ad hoc, draw-code, test-driven*).

Com isso, pode-se concluir que quanto mais complexas forem as árvores de decisão e o raciocínio utilizado pelos funcionários, maior será o grau de disponibilidade de tarefas, diversidade nas ações executadas e a forma em que os funcionários reagirão a mudanças no ambiente de trabalho. Isso torna o jogo mais dinâmico e até um pouco imprevisível, pois a mesma equipe de trabalho pode desenvolver o software de forma diferente em cada iteração do jogo dependendo das decisões tomadas.

# Referencias Bibliográficas

**Decision Tree Analysis**. Disponível em: <http://www.mindtools.com/dectree.html>. Acesso em: 26 maio. 2012.

HIGGINS, T. **Unity - 3D Game Engine**. Disponível em: <http://unity3d.com/>. Acesso em: 5 maio. 2011.

KOHWALTER, T.; CLUA, E.; MURTA, L. **SDM – An Educational Game for Software Engineering**. . Salvador: In: X SBGames. , 2011

MITCHELL, T. Machine Learning. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. p. 432.

MORET, B. Decision Trees and Diagrams. **In: ACM Computing Surveys (CSUR)**, v. 14, n. 4, p. 593–623, dez 1982.

QUINLAN, J. Induction of Decision Trees. **In: Machine Learning**, v. 1, p. 81–106, 1986.

RAO, A.; GEORGEFF, M. BDI-agents: from theory to practice. **In: Proceedings of the First Intl. Conference on Multiagent Systems**, p. 312–319, 1995.

ZADEH, L. Fuzzy Logic. **In: IEEE Computer Society Press Los Alamitos, CA, USA**, v. 21, n. 4, p. 83–93, abr 1988.