MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

IMPLEMENTAÇÃO DE ALGORITMOS DE INTELIGÊNCIA ARTIFICIAL PARA SOLUÇÃO DE PROBLEMAS DE COBERTURA EXATA

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/CNPq/INPE)

Regis Lima Claus (UNIFESP, Bolsista PIBIC/CNPq)
E-mail: regis.claus@unifesp.br

Dr. Rafael Duarte Coelho dos Santos (LAC/CTE/INPE, Orientador)
E-mail: rafael.santos@lac.inpe.br

Julho de 2010

SUMÁRIO

Resumo …. 1
1. Introdução …. 2
2. Problemas e Soluções …. 2
3. Problema de Cobertura Exata …. 3
4. Análise de Problemas de Cobertura Exata …. 6
    4.1. Jogo da Velha …. 6
    4.2. Connect Four …. 8
5. Tower Defense …. 12
    5.1. Problema de Melhor Caminho do Inimigo …. 13
    5.2. Problema de Melhor Posicionamento das Torres …. 15
6. Conclusão …. 19
7. Referências …. 19

RESUMO

Este trabalho, iniciado em agosto de 2009, iniciou com um estudo de problemas genéricos de cobertura completa, onde o objetivo é encontrar uma coleção de subconjuntos de um conjunto *S* de forma que cada elemento de *S* apareça uma única vez na coleção de subconjuntos. Algoritmos que resolvam este problema podem ser aplicados em alguns jogos de estratégia para, por exemplo, maximizar uma configuração de defesa ou ataque das peças no jogo.

Embora existam algoritmos eficientes para solução do problema de cobertura completa (como, por exemplo, o Algoritmo X de Donald Knuth), para determinadas aplicações pode ser possível identificar heurísticas mais simples, rápidas e que tenham eficiência aceitável.

O trabalho presentemente compreende um estudo sobre algoritmos de inteligência artificial e otimização para aplicá-los em uma situação onde o objetivo é impedir que um agente faça o melhor caminho. Situação que acontece, por exemplo, em um jogo conhecido como *Tower Defense*, no qual têm-se inimigos que, partindo de um ponto do mapa, desejam chegar a outro ponto no menor tempo e caminho possível. Para impedi-los, deve-se posicionar barreiras cujo número é limitado pela quantidade de recursos. A locomoção dos inimigos e o posicionamento das barreiras no mapa são representadas em uma Matriz de Adjacência.

Para a locomoção dos inimigos utiliza-se algoritmos para a solução de melhor caminho em grafos, como o Algoritmo A*. Para o melhor posicionamento das barreiras é criada uma base de testes de configuração de posições. Para a demonstração dos algoritmos a implementação é baseada na biblioteca gráfica Java2D, pois existe dificuldade em visualizar os resultados destes. E por fim é demonstrado um teste comparativo da eficiência do tempo de execução e uma utilização de um algoritmo Apriori de Mineração de Dados, para visualizarmos alguns padrões que podem aplica-se no posicionamento das torres.

1 INTRODUÇÃO

A busca por melhores soluções para problemas de cobertura exata, ou seja, forma que seja única é um grande desafio para muitos matemáticos e cientistas da computação. Umas das áreas que mais atua essa pesquisa é na área de Inteligencia Artificial, sendo essa umas das suas mais fortes motivações.

Para se achar tais soluções existem diversos estudos, como do algoritmo X de Donald Knuth, um dos mais renomeados pesquisadores no campo de analise de algoritmos e teoria da computação.

Esse trabalho esta divido em duas grandes parte de estudo. A primeira parte possui a explicação sobre problemas de cobertura exata (PCE), primeiramente descrevendo como identificarmos um problema, e como classificá-lo como sendo de cobertura exata. Seguido pela descrição de um problema solucionado e alguns problemas clássicos, como Soduko, N-Rainhas e Poliominós, e implementações para identificarmos um PCE nos jogos Jogo da Velha e *Connect Four*, focando numa análise de desempenho.

Na segunda parte temos como estudo de caso o jogo *Tower Defens*e, onde aplica-se a analise de um PCE para a posicionamento das torres. Temos também uma breve explicação de um algoritmo A* de busca em caminhos utilizada na movimentação de inimigos. E finalizando com uma aplicação de Arvores de Decisão e do algoritmo Apriori de Mineração de Dados para buscarmos padrões de acordo com a análise de PCE do *Tower Defense*.

2 PROBLEMAS E SOLUÇÕES

Para definir os problemas apresentados nesse trabalho defini-se que um Problema formalmente possui os seguintes componentes.

- **Conjunto de Estados:** Estados que podem ser definidos como por exemplo estado de um tabuleiro, representado como a tupla (x0, x1, …, xn);
- **Estado Inicial:** Por onde começa o problema;

- **Função sucessor:** Uma descrição de ações possíveis disponíveis, ou seja, dado um estado x, temos um sucessor y, representado pela tupla <estado(x), estado(y)>;
- **Teste de Objetivo:** Determina se estado atual esta em um estado objetivo.
- **Custo de Caminho:** Sendo caminho uma sequencia de estados conectados por uma sequencia de ações, temos um custo numérico para cada um deste.

3 PROBLEMA DE COBERTURA EXATA

Pela definição matemática um problema de cobertura exata (PCE) define-se em encontrar uma coleção de subconjuntos de um conjunto S de forma que cada elemento de S apareça uma única vez na coleção de subconjuntos. Um PCE é classificado como um problema NP-Completo e faz parte de um dos 21 Problemas NP-Completos de Karp.

Pode-se representar um PCE por Matrizes de Incidência ou um Grafo Bipartido.

Matriz de Incidência: Dada a Figura 1, que representa um grafo, pode-se representá-lo em uma matriz de 2 dimensões sendo uma dimensão são os vértices e outra são as arestas.

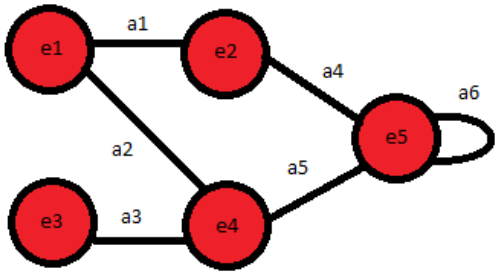

Figura 1: Grafo para representação da Matriz de Incidência

Os valores da matriz dão representados seguindo:

- 1 se o vértice incide na aresta
- 2 se há um laço (incide 2 vezes)
- 0 se não há incidência

Para o grafo da Figura 1 a matriz de Incidência é mostrada como na figura 2.

	a1	a2	a3	a4	a5	a6
e1	1	1	0	0	0	0
e2	1	0	0	1	0	0
e3	0	0	1	0	0	0
e4	0	1	1	0	1	0
e5	0	0	0	1	1	2

Figura 2: Matriz de Incidência

Grafo Bipartido: É um grafo cujo os vértices são 2 ou mais conjuntos em 2 conjuntos, nos quais não se pode ter arestas entre vértices de um mesmo conjunto.

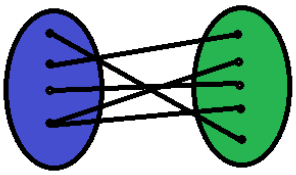

Figura 3: Grafo Bipartido

Sendo S = {A, B, C, D, E, F} uma coleção de subconjunto de um conjunto X = {1, 2, 3, 4, 5, 6, 7} tal que:

- A = {1, 4, 7}
- B = {1, 4}
- C = {4, 5, 7}
- D = {3. 5, 6}
- E = {2, 3, 6, 7}
- F = {2, 7}

Então um subconjunto S* = {B, D, F} é um PCE, onde cada elemento X contem esta contido em exatamente em único subconjunto:

- B = {1, 4}, D = {3, 5, 6}, F = {2, 7}

Podemos representar então o problema acima na Matriz de Incidência da Figura 4 e no Grafo Bipartido da Figura 5.

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Figura 4: Matriz de Incidência PCE

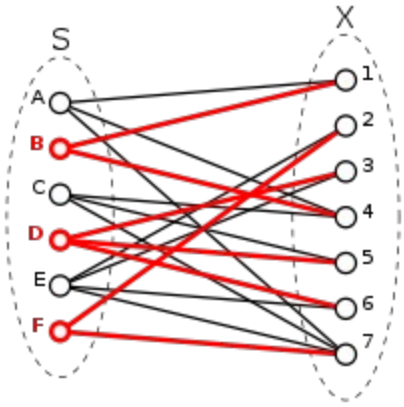

Figura 5: Grafo Bipartido PCE

Alguns dos problemas clássicos de PCE são:

- **Soduko:** identificação das combinações de números que cobrem completamente uma matriz 9x9;
- **Polominós**: identificação das combinações de formas geométricas que cobrem uma área;
- **N-Rainhas:** identificação das posições de N rainhas em um tabuleiro de xadrez NxN onde uma rainha não pode atacar a outra.

4 ANÁLISE DE PROBLEMAS DE COBERTURA EXATA

Nesse estudo para representarmos e buscarmos uma solução é utilizado o conceito de estrutura de dados como árvore. Uma árvore pode ser representada como uma coleção de vértices e uma de arestas entre estes vértices, onde esses vértices representam estados de uma solução.

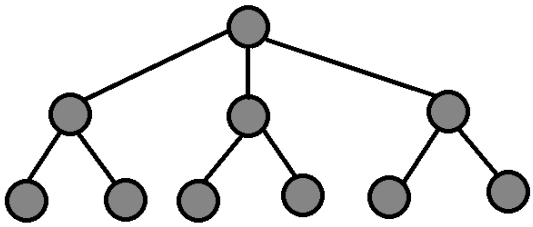

Figura 5: Grafo em Árvore

Para demonstração dos estudos de problemas de cobertura exata, utiliza-se como caso de uso primeiramente o jogo da velha e em seguida o jogo connect-four. Nessa etapa buscamos as todas as possíveis soluções, ou seja, finais de jogo, seja ela por vitória de algum dos jogadores ou por alguma regra que impeça a continuidade do jogo. O objetivo inicial é estudar o espaço de busca dos jogos para verificar a possibilidade de solução usando força bruta. Na primeira implementação tentamos calcular o número de folhas da árvore correspondente ao espaço de busca dos jogos.

4. 1 JOGO DA VELHA

Descrição: O tabuleiro é uma matriz de três linhas por três colunas. Dois jogadores escolhem uma marcação cada, geralmente um círculo (O) e um xis (X). Os jogadores jogam alternadamente, uma marcação por vez, numa lacuna que esteja vazia. O objetivo é conseguir três círculos ou três xis em linha, quer horizontal, vertical ou diagonal, e ao mesmo tempo, quando possível, impedir o adversário de ganhar na próxima jogada.

Pode-se definir um jogo da velha conforme a definição formal de um problema em:

- **Conjunto de Estados:** Cada estado seguindo o modelo a seguinte tupla. (x0, x1, x2, x3, x4, x5, x6, x7, x8, x9), onde xn é estado da lacuna, por exemplo, (O, ' ', X, ' ', O, X, ' ', ' ', ' ') que pode-se visualizar melhor no tabuleiro do jogo conforme a Figura 6.

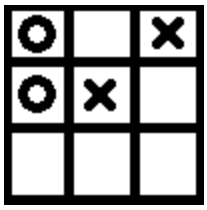

Figura 6: Estado de um Jogo da Velha

- **Estado Inicial:** No caso, (' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '), todas as lacunas vazias;
- **Função Sucessora:** Representado por um par <estado1, estado2>, melhor visualizado pela árvore de estados da Figura 7.
- **Teste de Objetivo:** Se há 3 círculos ou três xis definindo assim um jogador vencedor.
- **Custo de Caminho:** Para ir de um estado a outro o custo é apenas um.

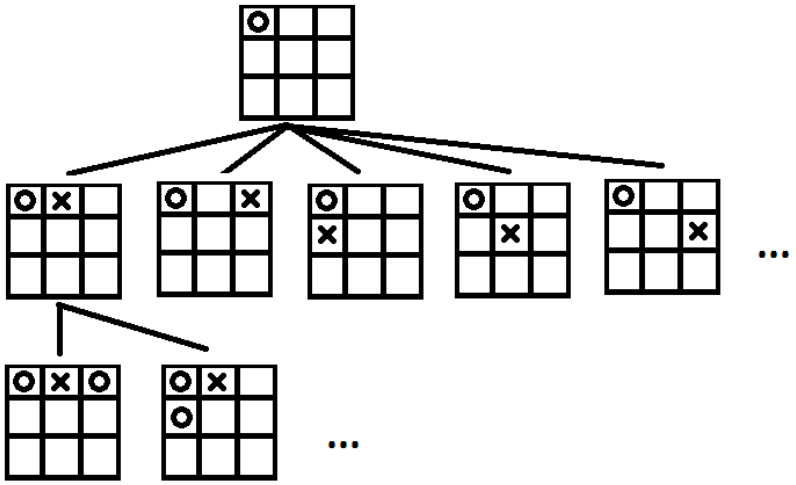

Figura 7: Parte da árvore de estados do Jogo da Velha

Teste: Como um teste de desempenho, pode-se observar na Figura 8 que o tempo de resposta é relativamente rápido para obter a arvore completa de estados e obtemos 362880 finais de jogo, ou seja, foram encontradas o Teste de Objetivo.

```
Arvore de possibilidades do Jogo da Ve-
lha:
 8557 ms.
 986410 vértices.
 986409 arcos.
 362880 folhas
```

Figura 8: Busca de todos estados do Jogo da Velha

4.2 CONNECT FOUR

Descrição: Connect-Four (também conhecido como quatro em uma linha) é um jogo para dois jogadores em cada jogador escolhe uma cor e depois se revezam colocando suas peças em um tabuleiro de sete colunas e seis linha. As peças caem direto para baixo, ocupando o espaço disponível na coluna mais abaixo. O objeto é conectar quatro de peças de cor semelhante ao lado da outra na vertical, horizontal ou diagonal antes de um oponente pode fazê-lo. Há muitas variações no tamanho da placa, o mais comumente utilizados são 7 × 6, seguido por 8 × 7, 9 × 7, e 10 × 7.

Pode-se definir um *Connect Four* conforme a definição formal de um problema em:

Conjunto de Estados: Cada estado seguindo o modelo a seguinte tupla. (L0, L1, L2, L3, …, LnXm), onde L é estado da lacuna, por exemplo como pode-se visualizar melhor no tabuleiro do jogo conforme a Figura 9.

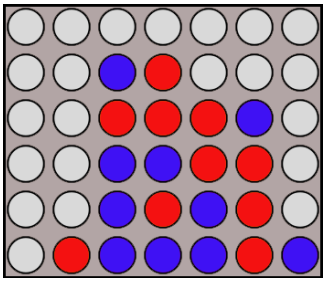

Figura 9: Estado de um Connect Four

- **Estado Inicial:** Representado pela Figura 10 tendo todas as lacunas vazias;

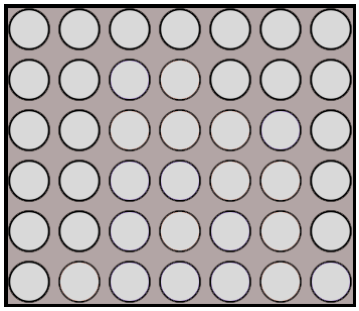

Figura 10: Estado Inicial do *Connect Four*

- **Função Sucessora:** Um um par <estado1, estado2>, visualizado pela árvore de estados da Figura 11.

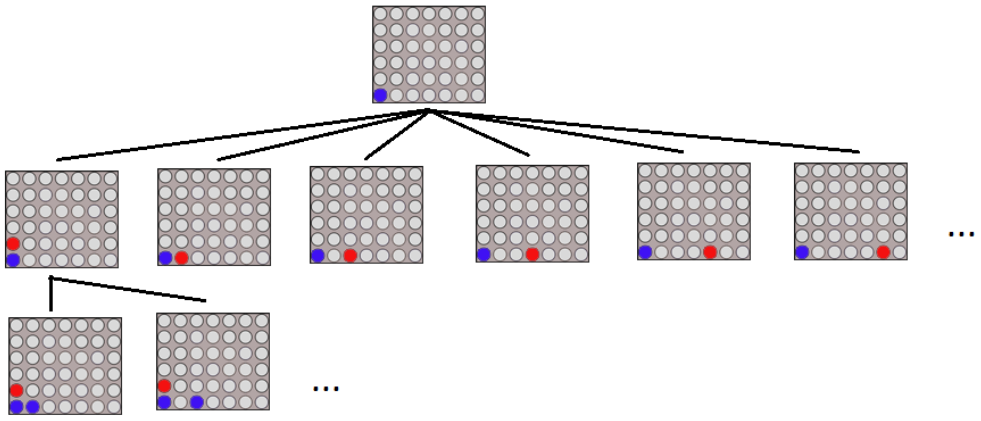

Figura 11: Arvore de Estados do *Connect Four.*

- **Teste de Objetivo:** Se há 4 peças da mesma cor alinhadas na horizontal, vertical ou diagonal.
- **Custo do Caminho:** Sendo apenas valor 1.

Teste: A figura 12 e 13 mostra o andamento da execução da aplicação para um tabuleiro 5x4, com cada bloco mostrando o número de iterações e dimensões da árvore Observa-se que conforme se aumenta o numero de iterações o tempo também de execução também aumenta, porem a partir de 10.000.000 o processamento em um computador desktop comum é inviável, necessitando de uma processamento maior como por exemplo um cluster ou de paralelismo.

A arvore de estados foi estimada em 10^{14} vértices por Allis em 1988.

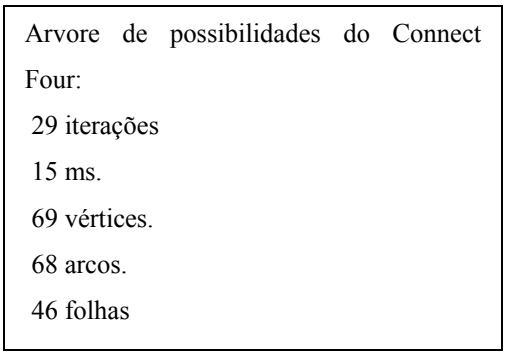

Figura 12: Busca por todos os estados do *Connect Four*, parte 1.

```
Arvore de possibilidades do Connect Four:
 118 iterações
 0 ms.
 155 vértices.
 154 arcos.
 93 folhas

Arvore de possibilidades do Connect Four:
 1016 iterações
 31 ms.
 1048 vértices.
 1047 arcos.
 558 folhas

Arvore de possibilidades do Connect Four:
 10017 iterações
 344 ms.
 10054 vértices.
 10053 arcos.
 4210 folhas

Arvore de possibilidades do Connect Four:
 100017 iterações
 3463 ms.
 100046 vértices.
 100045 arcos.
 42527 folhas
```

Figura 13: Busca por todos os estados do *Connect Four*, parte 2.

5 TOWER DEFENSE

Um *Tower Defense* (TD) é um jogo digital no gênero estratégia onde o jogador deve impedir que inimigos cheguem há um alvo, geralmente um ponto em um mapa com ou sem obstáculos. Para impedir o avanço dos inimigos são posicionadas torres, onde o grande desafio é buscar a melhor posicionamento destas. Geralmente há uma pontuação que é relativa a velocidade (tempo) em que se elimina os inimigos.

Os inimigos possuem os seguintes atributos:

- Velocidade de locomoção (VL) – valor inteiro
- Pontos de Vida (PV) – valor inteiro
- Imunidade a Status – valor booleano
- Aéreo – valor booleano

As torres possuem os seguintes atributos:

- Dano – valor inteiro
- Velocidade de ataque (VA) – valor inteiro
- Ataque aéreo – valor booleano
- Diminuir velocidade – valor inteiro

Uma *wave* é conjunto de X inimigos com seus devidos atributos. Por exemplo, uma *wave* de 20 inimigos de VL = 1, PV = 1 outra *wave* com 2 inimigos de VL = 2, PV = 10, Imune a Status = 1. E considera-se um *round* um conjunto de *waves*.

O jogador em um *round* possui um numero X de chances que pode deixar um inimigo chegar ao seu objetivo. Também possui um quantidade de recurso onde pode comprar mais torres ou dar *upgrade,* ou seja, aumentar valores como o Dano ou VA.

Concluindo, o jogo termina se o jogador chega ao final do *round*, ou se perde todas suas chances do inimigo chegarem ao objetivo.

Nesse trabalho estudamos os problemas a seguir.

5.1 PROBLEMA DE MELHOR CAMINHO DO INIMIGO

Para que um inimigo chegue a seu objetivo, ou seja, um ponto no mapa é preciso que ele percorra o menor caminho possível do inicio ao fim. Porém há um problema onde estando em uma posição atual no mapa qual posição se deve tomar como próxima. Juntamente há também um problema de como não fazer que um inimigo avance para uma posição onde contem uma torre.

Para formalizar o entendimento de uma posição nomeamos ela como quadro e exemplifica-se um mapa como abaixo.

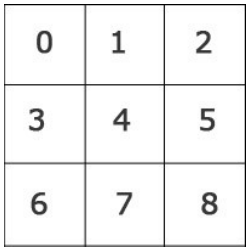

Figura 14: Exemplo de um *Tower Defense*

Então para definirmos esse problema, é citados seus componentes como:

- **Conjunto de Estado:** como por exemplo, em(quadro 5), em(quadro 7).
- **Estado Inicial:** pode-se definir como em(quadro 0).
- **Função Sucessora:** defini-se por um grafo de adjacência abaixo baseada no mapa da Figura 14, ou seja, para um estado atual em(quadro 3) podemos ter as seguintes tuplas <em(quadro 3), para(quadro 2)>, <em(quadro 3), para(quadro 5)>, <em(quadro 3), para(quadro 6)>.

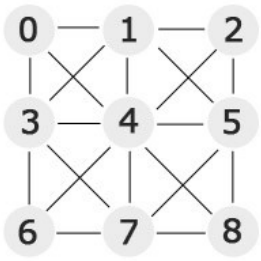

Figura 15: Exemplo de um Grafo de Adjacência

- **Teste Objetivo:** estar no estado final, no caso em(quadro 8).
- **Custo de Caminho:** para cada mudança de um estado a outro, defini-se um custo 1.

Solução:

Para solução desse problema de movimentação, utiliza-se um algoritmo A* pois sua heurística f(n) + g(n) é facilmente implementável Estando em um quadro n, f(n) é o custo para ir para um quadro adjacente, ou seja, custo um, e g(n) como o custo de ir até o quadro final.

Como tem-se a posição (x, y) do quadro atual e do quadro final, define-se a função g(n) como:

$$g(n)=\sqrt{(xQuadroFinal-xQuadroFinal)^2+(yQuadroFinal-yQuadroAtual)^2}$$

Para um teste simples como na Figura 16, podemos concluir que a busca A* funciona perfeitamente para o problema.

```
Testando movimentação do inimigo ...
Quadro Inicio: 0
Quadro Alvo: 8

Movimentando ...
Quadro atual: 0
Quadro atual: 4
Quadro atual: 8

Teste de movimentação ok!
```

Figura 16: Teste de Movimentação em Busca A*

5.2 PROBLEMAS DE MELHOR POSICIONAMENTO DAS TORRES

Como já citado para se ter melhores resultado, o jogador precisa posicionar suas torres da melhor forma possível. O que distingui um bom jogador de TD é conhecer essas melhores posições. Objetivo desse problema é o jogador minimizar suas perdas de chances de um inimigo chegar ao seu objetivo.

Então nesse estudo, classifica-se esse problema como um Problema de Cobertura Exata, onde como no Jogo da Velha citado anteriormente, tem-se um uma solução única para posição das torres. Porém as soluções dele é simples, ou seja, para cada torre já posicionada a próxima torre a ser posicionada não pode ocupar o mesmo espaço, tem-se uma solução como a abaixo por exemplo.

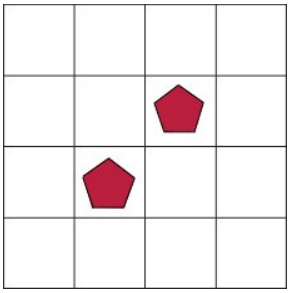

Figura 17: Exemplo de Posição de Torres

Podemos definir então o Problema de Melhor Posicionamento das Torres (PMT) nos componentes:

- **Conjunto de Estados:** posições do mapa ocupada *(true)* ou não *(false)* e uma tupla de quadros como da Figura 17 temos:

(false, false, false, false, false, false, true, false, false, true, false, false, false, false, false, false)

- **Estado Inicial:** todas as posições desocupadas ou seja:

(false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false)

- **Função Sucessora:** onde posicionada uma torre, para cada quadro livre, pode-se posicionar uma próxima torre como mostra a arvore de possibilidade da Figura 18.

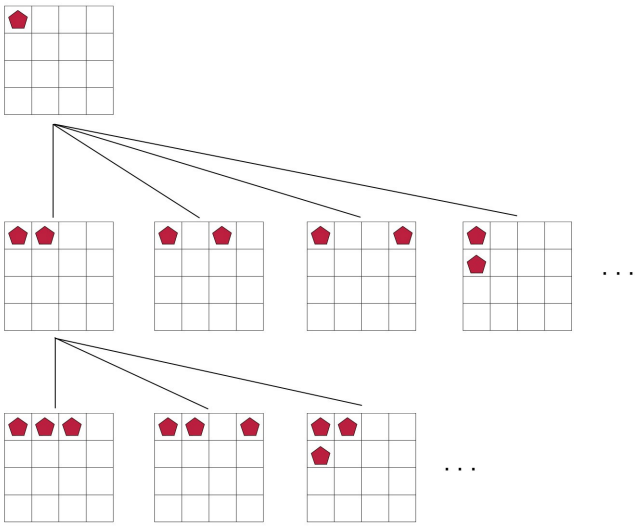

Figura 18: Arvore de Possibilidade de Posicionamento de Torres

- **Teste Objetivo:** Se estão posicionadas todas as x torres que o jogador possui para posicionar.
- **Custo de Caminho:** Simples, custo 1

Solução

Para o seguinte caso de teste:

- **Mapa:** 4x4, Quadro Inicio 0 e Quadro Final 15;
- ***Wave:*** 10 inimigos;
- **Inimigos:** Ponto de Vida = 1 / Velocidade de Movimentação = 1;
- **Numero de torres:** 2;
- **Chances do Jogador:** 10;

O objetivo é, a partir de uma simulação desse *round* com várias combinações de posições de torres e um número fixo de inimigos, tentar descobrir se existe algum tipo de configuração ótima que possa servir de estratégia para posicionar as torres.

Uma primeira análise é feita usando árvores de decisão para tentar determinar o conjunto de regras que garante o sucesso (poucos sobreviventes) ou fracasso (muitos sobreviventes) de uma sessão do jogo. Atributos da base que são redundantes foram eliminados, por exemplo, já que em todas as configurações das sessões do jogo temos 10 inimigos na entrada, e as posições P1 e P16 estão sempre sem torres (já que são entrada e saída) eliminamos estes atributos da base.

O algoritmo ID3 (implementado em Java no software Weka) foi usado, e esta é a árvore que gerou.

```
P15 = FALSE
| P12 = FALSE
| | P04 = FALSE
| | | P13 = FALSE
| | | | P14 = FALSE
| | | | | P09 = FALSE: 6
| | | | | P09 = TRUE
| | | | | | P02 = FALSE
| | | | | | | P07 = FALSE: 8
| | | | | | | P07 = TRUE: 7
| | | | | P02 = TRUE: 7
| | | P14 = TRUE
| | | | P07 = FALSE
| | | | | P09 = FALSE: 8
| | | | | P09 = TRUE: 10
| | | | P07 = TRUE: 7
| | P13 = TRUE: 8
| P04 = TRUE
| | P06 = FALSE
| | | P11 = FALSE
| | | | P05 = FALSE
| | | | | P07 = FALSE
| | | | | | P10 = FALSE: 10
| | | | | | P10 = TRUE: 8
| | | | | P07 = TRUE: 8
| | | | P05 = TRUE: 8
| | | P11 = TRUE: 7
| | P06 = TRUE: 7
| P12 = TRUE
| | P13 = FALSE
| | | P14 = FALSE: 7
| | | P14 = TRUE: 8
| | P13 = TRUE: 9
P15 = TRUE
| P04 = FALSE
| | P09 = FALSE: 7
| | P09 = TRUE: 9
| P04 = TRUE: 9
```

Figura 19: Árvore de Decisão para Teste de PMT

Esta árvore classifica corretamente 100% dos casos, mas sua interpretação é complexa.

Observa-se uma regra que a arvore mostra, destacado em azul. se não temos torre em P15, P12, P04, P13, P14 e P09 tem-se seis sobreviventes. Embora não aparenta muito conclusiva, por outro lado a regra destacada em verde, se temos torre em P15 e em P4, tem-se nove sobreviventes: mostra que colocar torres nestas duas posições não é uma boa decisão para o jogador.

Para um segundo tipo de análise procura-se regras de associação com base em uma algoritmo chamado Apriori que tenta achar associações generalizadas em uma base. Estas associações são do tipo "se X acontece então Y também acontece, e isto acontece na base N vezes".

O resultado desse algoritmo procura regras por mais fracas que sejam. Ao pedir para ele encontrar 25 regras esse foi o resultado:

```
Best rules found:

 1. P12=yes P13=yes 2 ==> OUT=9 2    conf:(1)
 2. P06=yes 12 ==> OUT=6 2    conf:(0.17)
 3. P11=yes 12 ==> OUT=7 2    conf:(0.17)
 4. P02=yes 14 ==> OUT=8 2    conf:(0.14)
 5. P03=yes 14 ==> OUT=7 2    conf:(0.14)
 6. P08=yes 14 ==> OUT=7 2    conf:(0.14)
 7. P10=yes 14 ==> OUT=7 2    conf:(0.14)
 8. P12=yes 14 ==> OUT=8 2    conf:(0.14)
 9. P12=yes 14 ==> OUT=9 2    conf:(0.14)
10. P09=yes 16 ==> OUT=9 2    conf:(0.13)
11. P14=yes 16 ==> OUT=7 2    conf:(0.13)
12. P04=yes 18 ==> OUT=9 2    conf:(0.11)
13. P13=yes 18 ==> OUT=9 2    conf:(0.11)
14. P13=yes 18 ==> OUT=10 2    conf:(0.11)
```

Figura 20: Teste do Algoritmo Apriori para um PMT

A primeira regra é a mais interessante: ela diz que quando as torres ficaram em P12 e P13 sempre saíram nove inimigos. Isto aconteceu duas vezes na base.

A regra 14 diz que em dois casos onde colocamos torre em P13 saíram 10 inimigos (existem 18 casos destes na base). Esta regra não é genérica, existem muitos outros casos que colocamos torres em P13 e saíram outro numero de inimigos.

6 CONCLUSÃO

Neste trabalho foram estudas e analisados alguns problemas de cobertura exata como jogo da velha e connect four. A busca por uma arvore de todas as soluções possíveis do jogo da velha é obtida sem nenhuma dificuldade, o que não ocorre no jogo Connect Four, onde mesmo com testes em computadores mais robustos, não se obteve por a árvore completa.

Na segunda parte desse trabalho analisou-se dois problemas relativo ao jogo Tower Defense. No Problema de Melhor Caminho de Inimigo, a implementação de um algoritmo A* se mostrou-se suficiente para solução deste.

No Problema de Posicionamento das Torres obteve-se a conclusão que algoritmo Apriori não dá uma solução para o melhor posicionamento porem ele tenta identificar heurísticas que indicam onde é que algumas torres combinadas podem dar bons e maus resultados.

Para uma continuidade deste trabalho, uma análise mais completa, como por exemplo, mapa onde possuem obstáculos, *waves* com atributos dos inimigos alterados, torres com mais dano podem gerar resultados mais precisos em relação aos resultados dos testes já estudados.

Também pode-se gerar testes dinamicamente, inserindo torres durante as iterações, já que o jogador pode obter mais torres durante o andamento do jogo.

E por fim uma aplicação do jogo com interface gráfica mostrando algumas configurações de posições encontradas nos testes podem gerar uma melhor análise dos resultados dinamicamente, ou seja, mostrar eliminação dos inimigos conforme o jogo for passando suas iterações.

7 REFERÊNCIAS

- Allis V., Searching for Solutions in Games and Artificial Inteligence, 1988
- Russel S., Norvig P., Inteligência Artificial, Ed. Campus, 2003
- Santos, R., Mineração de Dados, 2008