UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RODRIGO LAMBLET MAFORT

Problemas de Alocação de Pentominos

Rio de Janeiro
janeiro de 2009

RODRIGO LAMBLET MAFORT

PROBLEMAS DE ALOCAÇÃO DE PENTOMINOS

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Orientadora:
Prof[a]. Lilian Markenzon

Rio de Janeiro
janeiro de 2009

Ficha Catalográfica

M187 Mafort, Rodrigo Lamblet

Problemas de Alocação de Pentominos / Rodrigo Lamblet Mafort. – Rio de Janeiro: UFRJ/IM/PPGI, 2009.

95 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro. Programa de Pós-Graduação em Informática, Rio de Janeiro, 2009.

Orientadora: Lilian Markenzon.

1. Pentominos. 2. Poliminos. 3. Problemas de Alocação. 4. Algoritmos. I. Markenzon, Lilian. II. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Programa de Pós-Graduação em Informatica. III. Título.

CDD:

Problemas de Alocação de Pentominos

Rodrigo Lamblet Mafort

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovado por:

_____

Profª. Lilian Markenzon (Orientadora)

D.Sc., UFRJ (Universidade Federal do Rio de Janeiro), Brasil.

_____

Profª. Nair Maria Maia de Abreu

D.Sc., UFRJ (Universidade Federal do Rio de Janeiro), Brasil.

_____

Prof. Fábio Protti

D.Sc., UFRJ (Universidade Federal do Rio de Janeiro), Brasil.

_____

Prof. Mitre Costa Dourado

D.Sc., UFRRJ (Universidade Federal Rural do Rio de Janeiro), Brasil.

Rio de Janeiro, 09 de dezembro de 2008

Ao eterno Tio Luiz *(in memoriam)* ...

Às queridas avós *(in memoriam)*, Alice e Isabel,
que sempre se fizeram exemplos...

AGRADECIMENTOS

A Deus, por iluminar meu caminho e permitir que eu pudesse dar mais esse passo.

À minha família, em especial meus pais, Rita e Antonio José, por apoiar minhas decisões, amparar-me quando precisei e se alegrar e entristecer junto comigo durante essa caminhada.

Em especial, à minha orientadora, Lilian Markenzon, pelo acolhimento, pela paciência e pela dedicação.

Ao professor Oswaldo Vernet, pela oportunidade e confiança.

A todos os funcionários do PPGI/NCE, em especial à Lina Marchese Olivieiro dos Santos, Selma Regina Mendes Martins e Deise Lobo Cavalcante, pela presteza e atenção a mim dispensadas.

RESUMO

MAFORT, Rodrigo Lamblet. Problemas de Alocação de Pentominos. Rio de Janeiro, 2008. Dissertação (Mestrado em Informática), Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

Este trabalho tem como objetivo apresentar os problemas de alocação de pentominos e desenvolver uma biblioteca de algoritmos que permita lidar computacionalmente com os mesmos. Um pentomino é uma figura geométrica plana formada por cinco quadrados iguais, denominados minos, conectados entre si de modo que pelo menos um lado de cada mino coincida com um lado de outro mino qualquer. A alocação de pentominos em um tabuleiro dá origem a interessantes problemas matemáticos. Dentre os procedimentos apresentados, deve-se destacar os algoritmos de construção de tabuleiros de formato livre e, em especial, a análise comparativa efetuada entre as diferentes versões apresentadas. Para ilustrar a aplicação da biblioteca de algoritmos implementou-se um protótipo de um jogo baseado em um dos problemas de alocação descritos.

Palavras-chave: Pentominos, Poliminos, Problemas de Alocação, Algoritmos.

ABSTRACT

MAFORT, Rodrigo Lamblet. Problemas de Alocação de Pentominos. Rio de Janeiro, 2008. Dissertação (Mestrado em Informática), Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

The main goal of this dissertation is to present the Pentomino problems and develop a library of procedures for the implementation of these problems. A pentomino is a plane figure formed by five contiguous squares of the same size. The allocation of pentominoes on a board can originate interesting mathematical problems. Among the presented procedures in this work, the free boards' construction algorithms and the comparative analysis of these algorithms must be emphasized. To illustrate the application of the library of algorithms a prototype of a game based in one of the described allocation problems was created.

Keywords: Pentominoes, Polyominoes, Allocation Problems , Algorithms.

LISTA DE FIGURAS

Figura 1.1: Pentominos e seus respectivos nomes 16
Figura 1.2: Exemplo dos Problemas de Alocação 17
Figura 1.3: Exemplo de rotação 20
Figura 1.4: Exemplo de espelhamento horizontal 21
Figura 1.5: Exemplo de espelhamento vertical 21
Figura 1.6: Exemplos de esquemas diferentes obtidos pelo mesmo número de transformações 22
Figura 1.7: 17 novos pentocubos 23

Figura 2.1: Exemplo da Cobertura Exata 26
Figura 2.2: Exemplo do Processo do Problema dos 20 28
Figura 2.3: Exemplo de tabuleiro que não permite 3 coberturas 29
Figura 2.4: Exemplos do Problema dos 20 30
Figura 2.5: Exemplo do Problema da Triplicação 31
Figura 2.6: Exemplos de paralelogramos “dentados” 32
Figura 2.7: Exemplo do Jogo do Último Encaixe 33

Figura 3.1: Exemplo da representação computacional de um esquema do Pentomino F 41
Figura 3.2: Maior cadeia de 4 pentominos 42
Figura 3.3: Exemplo da representação computacional do tabuleiro 42
Figura 3.4: Exemplo de aplicação do algoritmo de rotação 43

Figura 3.5: Exemplo de aplicação do algoritmo de espelhamento vertical 45
Figura 3.6: Exemplo de aplicação do algoritmo de espelhamento horizontal . . . 46
Figura 3.7: Exemplo de aplicação do algoritmo de validação 48
Figura 3.8: Aglutinado de 2 pentominos 51
Figura 3.9: Origem das componentes isoladas 51
Figura 3.10: Perímetro do aglutinado 52
Figura 3.11: Menor perímetro de um aglutinado de 4 pentominos 54
Figura 3.12: Exemplo de redução de perímetro após alocação 55
Figura 3.13: Exemplo de caso onde existe diferença entre o tamanho de $PeriLista$ e o valor de $peri$ 56
Figura 3.14: Exemplo do funcionamento do procedimento $Perimetro$ 59
Figura 3.15: Exemplo do funcionamento da versão 1.1 do algoritmo 67
Figura 3.16: Exemplo do descarte do tabuleiro por sucessivas posições incorretas . 68
Figura 3.17: Exemplo da aplicação da fila de pentominos 68
Figura 3.18: Tabuleiros construídos pela versão 1.1 do algoritmo 69
Figura 3.19: Aplicação do teste de qualidade da versão 1.2 do algoritmo 71
Figura 3.20: Tabuleiros construídos pela versão 1.2 do algoritmo 72
Figura 3.21: Exemplo do funcionamento da versão 2.1 do algoritmo 77
Figura 3.22: Exemplo da falha na alocação 79
Figura 3.23: Novos pontos da alocação 80
Figura 3.24: Exemplo do cálculo de novos pontos para versão 2.2 do algoritmo . . 81

Figura 4.1: Protótipo: Tabuleiro em branco 85
Figura 4.2: Protótipo: Movendo o pentomino X 86
Figura 4.3: Protótipo: Pentomino X já alocado 86
Figura 4.4: Protótipo: Preparando o pentomino F para alocação 87
Figura 4.5: Protótipo: Menu *pop-up* de transformações 87
Figura 4.6: Protótipo: Tabuleiro totalmente coberto 88
Figura 4.7: Protótipo: Mensagem ao final da cobertura 88

Figura 4.8: Protótipo: Interface ao final da cobertura 89
Figura 4.9: Protótipo: Outro tabuleiro gerado 89
Figura 4.10: Protótipo: Cobertura automática de tabuleiros 90
Figura 4.11: Diagrama de Classes do Jogo dos 20 91

LISTA DE TABELAS

Tabela 1.1: 63 possíveis esquemas de pentominos 21
Tabela 1.2: Seqüência de transformações para obtenção de todos os esquemas . . 22
Tabela 2.1: Número de soluções de cada tabuleiro retangular 27
Tabela 3.1: Perímetro dos Pentominos 52
Tabela 3.2: Resultados da versão 1.1 do algoritmo 70
Tabela 3.3: Tabuleiros descartados por intervalo de perímetro 71
Tabela 3.4: Resultados da versão 2.1 do algoritmo 78
Tabela 3.5: Resultados da versão 2.2 do algoritmo 82
Tabela 3.6: Tabela comparativa entre as versões do algoritmo de criação de tabuleiros de formato livre 82

LISTA DE ALGORITMOS

1 Procedimento $Rotacionar(pento)$ 43
2 Procedimento $Espelhar\_V(pento)$ 44
3 Procedimento $Espelhar\_H(pento)$ 45
4 Função $Validar(vMatriz, pento, x, y) : booleano$ 47
5 Procedimento $Alocar(x, y, pento, vMatriz)$ 48
6 Procedimento $Remover(vMatriz, pento, x, y)$ 49
7 Função $Gerar\_Tabuleiro\_Retangular(h, l : inteiro) : TMatriz$ 50
8 Procedimento $Perimetro(vMatriz, PeriLista, peri)$ 58
9 Função $ValidarConstrucao(vMatriz, pos_x, pos_y, pento) : booleano$ . . 61
10 Procedimento $Aloca - Desaloca(x, y, pento, vMatriz, nome)$ 62
11 Procedimento $Inicializar(vMatriz, Pent)$ 63
12 Função $Gerar\_Tabuleiro\_V1.1(n) : vMatriz$ 66
13 Função $Gerar\_Tabuleiro\_V2.1(n) : vMatriz$ 75

SUMÁRIO

1 INTRODUÇÃO 16

1.1 História 17

1.2 Introdução aos pentominos 19

1.3 Pentocubos 23

2 PROBLEMAS DE ALOCAÇÃO 24

2.1 Características dos problemas 24

2.2 Cobertura Exata de Superfícies 25

2.3 Problema dos 20 27

2.4 Problema da Triplicação 29

2.5 Cobertura de Paralelogramos Dentados 31

2.6 Jogos decorrentes dos problemas 32

2.7 Jogos educacionais 34

2.8 Recursos na rede 35

3 ALGORITMOS BÁSICOS DE MANIPULAÇÃO 40

3.1 Estruturas de dados 40

3.2 Transformações de pentominos 42

3.3 Disposição de pentominos no tabuleiro 46

3.4 Construção de tabuleiros 49

3.5 Construção de tabuleiros de formato livre 50
3.5.1 Aglutinados 50
3.5.2 Algoritmos de construção 62
3.5.3 Algoritmo de construção: Versão 1.1 64
3.5.4 Algoritmo de construção: Versão 1.2 70
3.5.5 Algoritmo de construção: Versão 2.1 73
3.5.6 Algoritmo de construção: Versão 2.2 78
3.5.7 Comparações entre os algoritmos 82

4 O JOGO DOS 20: UM PROTÓTIPO 83
4.1 Implementação de um protótipo 84
4.2 Recursos ainda não implementados 91

5 CONCLUSÕES E TRABALHOS FUTUROS 92

REFERÊNCIAS 94

1 INTRODUÇÃO

Poliminos são figuras geométricas planas formadas por quadrados iguais, denominados minos, conectados entre si de modo que pelo menos um lado de cada mino coincida com um lado de outro mino qualquer. Dependendo do número de minos, o polimino recebe uma denominação diferente. Os nomes para os poliminos que possuem de um a oito minos são respectivamente monomino, dominó, triminó, tetrominó, pentomino[1], hexomino, heptomino e octomino. Este trabalho trata de pentominos.

Existem doze pentominos diferentes e estes são nomeados usando letras do alfabeto latino de acordo com a semelhança entre o pentomino e a letra, conforme pode ser visto na Figura 1.1.

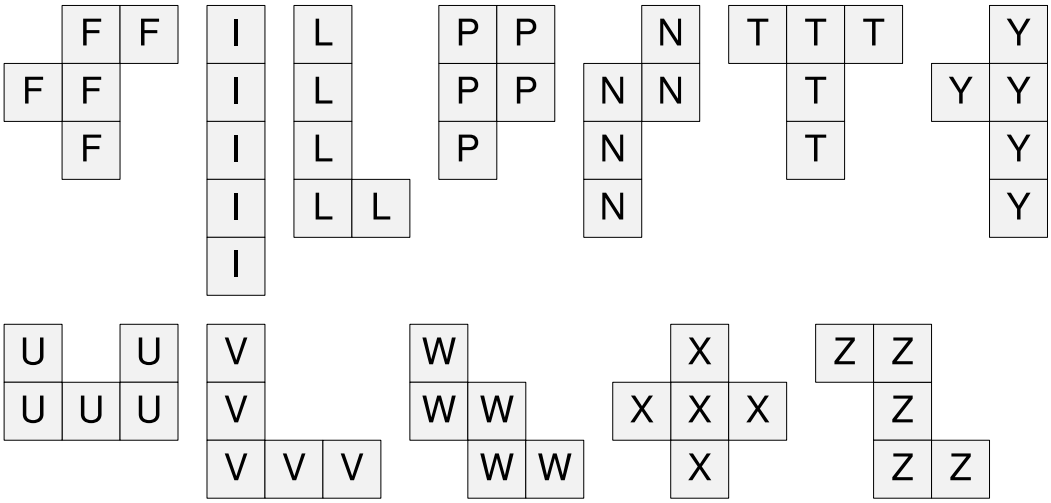


Figura 1.1: Pentominos e seus respectivos nomes

[1]Pentomino e Pentominoes foram registrados como uma marca comercial de Solomon W. Golomb, porém em 1982 ela expirou.

Pentominos dão origem a interessantes problemas, dentre os quais os de alocação, que serão discutidos no Capítulo 2. Problemas de alocação visam dispor x pentominos em uma determinada superfície, cuja área deve ser de no mínimo $5x$ minos. Esta superfície é denominada *tabuleiro*. Na Figura 1.2, pode-se encontrar uma solução para o Problema da Cobertura Exata (alocar doze pentominos distintos em um tabuleiro retangular).

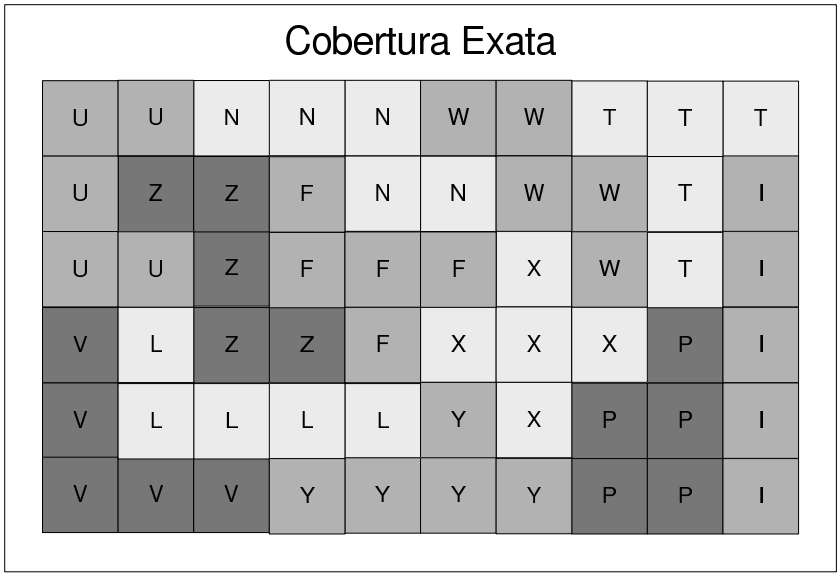


Figura 1.2: Exemplo dos Problemas de Alocação

No restante deste capítulo, serão apresentados a história dos pentominos, os principais conceitos e os pentocubos. O Capítulo 2 apresentará alguns problemas de alocação. Além disso, serão expostos alguns jogos decorrentes dos problemas apresentados, como estes jogos podem ser utilizados com finalidades educacionais e um levantamento dos *sites* disponíveis na rede sobre pentominos.

O Capítulo 3 apresentará e discutirá algoritmos que permitem a manipulação computacional dos pentominos e seus problemas. Estes algoritmos constituem a contribuição principal deste trabalho. Para ilustrar a aplicação destes algoritmos, um protótipo de um jogo foi desenvolvido, conforme pode ser visto no Capítulo 4.

1.1. História

A primeira menção histórica aos poliminos se refere ao jogo de dominó, cuja origem é a China. Segundo as lendas, o jogo teria sido inventado por um funcionário do imperador Hui Tsung. Uma outra versão diz que este foi criado nos anos de 234 a 181

a.C, quando teria vivido Huang Ming, um soldado-herói. O dominó é conhecido na China como “kwat p´ai”, significando “tabletes de osso”.

No ocidente, há indícios da existência do dominó no século XVIII, quando teria sido introduzido na Inglaterra e Itália pelos viajantes da época. Ainda outra versão afirma que o jogo apareceu espontaneamente em diversas partes do globo.

Em 1954, os poliminos foram introduzidos aos matemáticos quando Golomb[1] escreveu um artigo conceituando toda a família dos poliminos, apresentando também os problemas decorrentes de cada um, tais como a cobertura de um tabuleiro de xadrez com dominós e a cobertura de superfícies com hexominos.

Em 1957, os poliminos se tornaram mundialmente famosos quando Gardner[2] escreveu sobre eles em sua coluna na revista Scientific American, criando uma febre entre os leitores. Foram apresentados nesse artigo problemas que até hoje são estudados, como o Problema da Cobertura Exata. Além desses, Gardner expôs problemas menos conhecidos, como o Problema da Triplicação e o Jogo do Último Encaixe. Um tabuleiro em particular chamou a atenção de todos: um tabuleiro de dimensões de 8x8 com quatro furos. O grande questionamento é se todos os tabuleiros deste tipo são passíveis de cobertura (descartando os óbvios, como os tabuleiros com divisões onde nenhum pentomino poderia ser alocado). Além disso, Golomb afirma a Gardner que ninguém havia conseguido obter uma fórmula para o número de n-ominos como uma função de n, problema em aberto até hoje.

A primeira publicação que ligou algoritmos aos pentominos foi em 1958, quando Scott[3] apresentou algoritmos *backtracking* que resolviam problemas de alocação, inclusive os tabuleiros de dimensões de 8x8 com quatro furos. O artigo mostra detalhadamente os algoritmos e as podas que podem ser feitas na árvore de probabilidades. Outro artigo contendo algoritmos que resolvem problemas de alocação foi publicado em 1965 por Fletcher[4]. Diferentemente de Scott, Fletcher apresenta uma maneira de implementar os algoritmos utilizando macros.

Em 1975, um outro artigo utilizou os pentominos como exemplos da aplicação de algoritmos *backtracking*, quando Bitner e Reingold[5] apresentam outras técnicas de implementação.

Pentominos têm sido bastante utilizados na área da educação. Em 1990, Tracy e Eckart[6] fizeram uma das primeiras sugestões de uso de pentominos na educação. Em seu artigo afirmam que o uso destes nas escolas traz benefícios, tais como o aprimoramento da capacidade de resolução de problemas e da visão espacial, dentre outros. Outras possíveis utilizações dos pentominos na educação foram propostas por Onslow[7], tais como a prática da contagem de perímetros e da visão de congruência.

Em 1994, Golomb[8] publicou um livro sobre os poliminos, apresentando toda a

família e as noções matemáticas que a suportam.

Em 1996, Orman[9] apresentou provas de que, no jogo do Último Encaixe, o primeiro jogador sempre vence, desde que jogue de maneira ótima, o que inclui uma determinada ordem na alocação.

Outro interessante problema decorrente dos pentominos foi abordado por Anstreicher [10] em 1999. Este problema, denominado Problema da Exclusão de Pentominos, busca cobrir uma determinada superfície com monominos de maneira a não permitir que nenhum pentomino possa ser alocado. Anstreicher prova que são necessários vinte e quatro monominos necessários para impossibilitar a alocação de um pentomino em uma superfície de dimensões 8x8.

No ano 2000, Knuth[11] apresentou, em seu artigo, uma técnica que permite melhorar a eficiência dos algoritmos *backtracking*. Um dos exemplos é um algoritmo que visa resolver problemas de alocação de pentominos.

Um interessante problema de alocação foi proposto por Koth e Grosser[12], cujo objetivo é cobrir paralelogramos "dentados" de área sessenta com doze pentominos distintos. Este problema será melhor conceituado na Seção 2.5.

Em 2007, Larsen[13] apresentou um novo jogo baseado em pentominos, denominado *Battleships*. Neste jogo, inicialmente um jogador deve cobrir um tabuleiro de dimensões 8x8 com quatro furos, armazenando esta cobertura sem que o primeiro jogador a veja. Posteriormente, o primeiro jogador deve escolher quatro posições do tabuleiro. O segundo jogador deve então dizer ao primeiro quais minos correspondem às posições escolhidas. O jogo segue até que o primeiro jogador consiga descobrir a cobertura efetuada. Quanto menos posições lhe forem reveladas melhor é a sua pontuação.

Gravier et al.[14] apresenta em seu artigo, publicado em 2007, uma generalização do Problema da Exclusão de Pentominos em grafos. Esta generalização visa descobrir o menor número de vértices que devem ser removidos de um grafo G, de maneira que todas as componentes isoladas restantes tenham cardinalidade de no máximo um valor x pedido. Além disso, o autor prova que este é um problema NP-Completo.

1.2. Introdução aos pentominos

Como foi dito, existem doze pentominos distintos, cada um nomeado de acordo com a sua semelhança com uma letra do alfabeto latino, como visto na Figura 1.1.

A cada pentomino podem-se aplicar transformações, isto é, maneiras de obter outros *esquemas* do pentomino. A função destas transformações é adequar um pentomino qualquer a um espaço. É importante dizer que as transformações não alteram as adjacências entre os minos e um esquema diferente não constitui um pentomino diferente, mas apenas uma outra forma de representá-lo. Existem duas transformações que podem ser

aplicadas aos pentominos, que são as *rotações* e os *espelhamentos*.

As rotações são giros de noventa graus em sentido horário. Um exemplo de rotação pode ser encontrado na Figura 1.3. Rotações podem ser aplicadas sucessivamente.

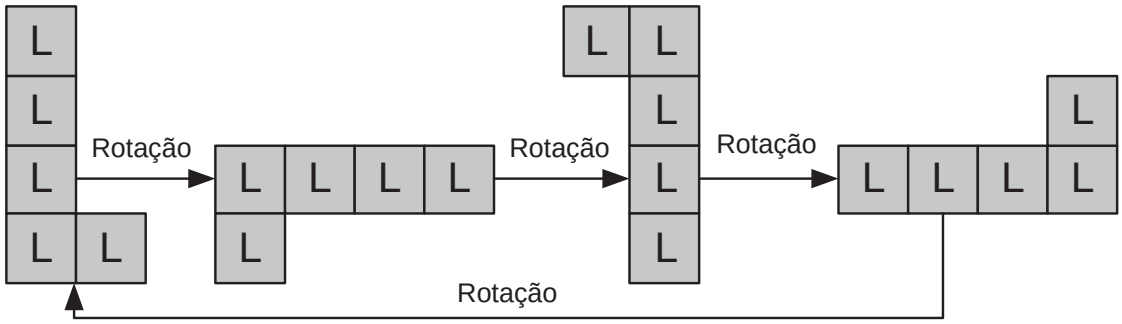


Figura 1.3: Exemplo de rotação

Existem dois tipos distintos de espelhamentos: horizontais e verticais. Nos horizontais, considera-se um espelho virtual no eixo horizontal, ou seja, no eixo x. A partir daí, o pentomino espelhado é a reflexão do pentomino original. Quanto ao vertical, a única diferença é a posição do espelho, que está situado no eixo vertical (eixo y). Assim como as rotações, os espelhamentos também podem ser aplicados sucessivamente.

Na Figura 1.4, apresenta-se um exemplo do espelhamento horizontal e na Figura 1.5, o espelhamento vertical.

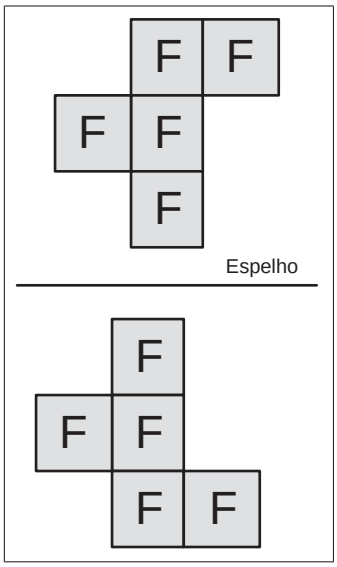


Figura 1.4: Exemplo de espelhamento horizontal

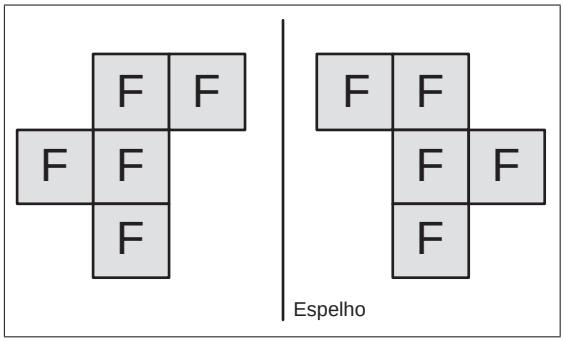


Figura 1.5: Exemplo de espelhamento vertical

Cada pentomino tem vários esquemas obtidos pela aplicação de sucessivas rotações e espelhamentos. A Tabela 1.1 apresenta o número de esquemas distintos de cada pentomino.

Pentomino	Rotações	Espelhamento	Total
F	4	4	8
I	2	0	2
L	4	4	8
P	4	4	8
N	4	4	8
T	4	0	4
U	4	0	4
V	4	0	4
W	4	0	4
X	1	0	1
Y	4	4	8
Z	2	2	4

Tabela 1.1: 63 possíveis esquemas de pentominos

Todos os esquemas dos pentominos podem ser obtidos através de rotações e espelhamentos horizontais. A Tabela 1.2 apresenta uma ordem de transformações para cada pentomino que permite obter todos os esquemas deste. Existem diversas ordens que permitem obter os mesmos esquemas.

	Transformação						
Pentomino	1^a	2^a	3^a	4^a	5^a	6^a	7^a
F	R	R	R	H	R	R	R
I	R						
L	R	R	R	H	R	R	R
P	R	R	R	H	R	R	R
N	R	R	R	H	R	R	R
T	R	R	R				
U	R	R	R				
V	R	R	R				
W	R	R	R				
X							
Y	R	R	R	H	R	R	R
Z	R	H	R				

Tabela 1.2: Seqüência de transformações para obtenção de todos os esquemas

É importante destacar que a ordem na qual as transformações são efetuadas resulta em esquemas diferentes. A Figura 1.6 ilustra um caso onde foram aplicadas seis rotações e um espelhamento horizontal em duas ordens distintas.

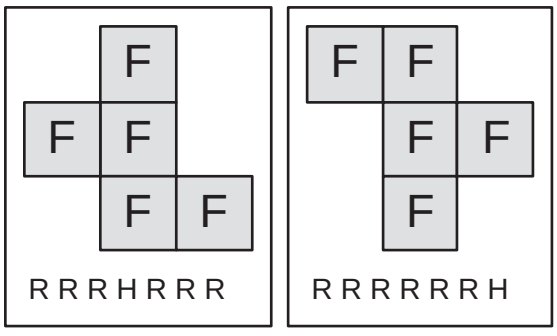


Figura 1.6: Exemplos de esquemas diferentes obtidos pelo mesmo número de transformações

1.3. Pentocubos

Até o momento, foram descritos apenas problemas que envolvem figuras planas, porém os pentominos podem ser utilizados em problemas 3D, bastando adicionar uma dimensão a cada mino. Portanto, cada mino será um cubo, e cada pentocubo será constituído de cinco minos (cubos).

Assim como os pentominos planos têm área cinco, os pentominos 3D têm volume cinco, ou seja, a superfície que os conterá deve ter volume cento e quarenta e cinco.

Além dos doze pentocubos que simplesmente adicionam uma dimensão aos pentominos conhecidos, dezessete novos pentocubos foram obtidos, uma vez que minos podem ser colocados em posições antes não consideradas. A Figura 1.7[2] apresenta estes dezessete novos pentocubos. Sendo assim, existem vinte e nove pentocubos.

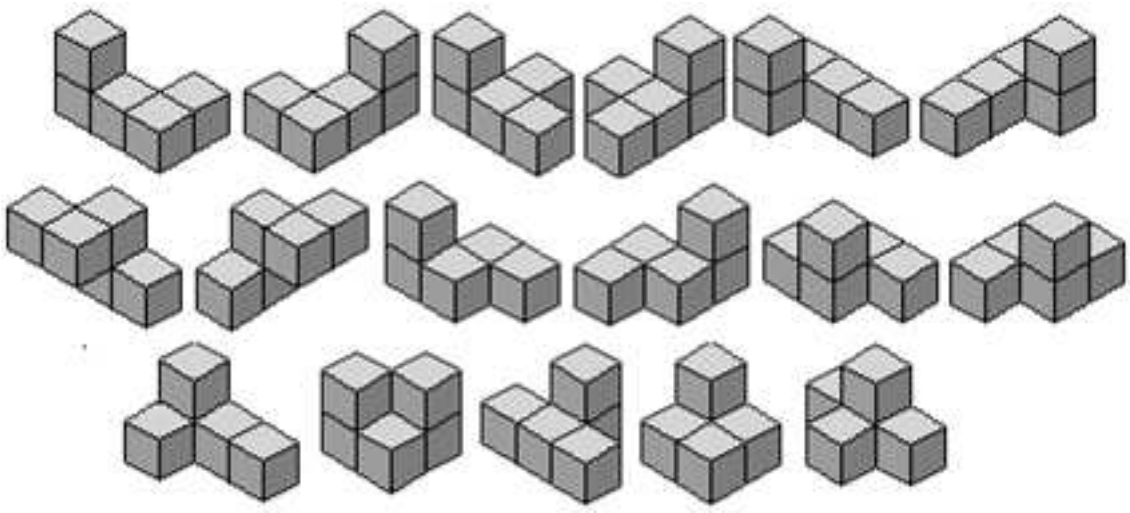


Figura 1.7: 17 novos pentocubos

Além dos movimentos de rotação e espelhamento, os pentominos 3D também podem ser rotacionados no eixo Z, o que garante mais esquemas.

[2]Fonte: http://www.recmath.com/PolyPages/PolyPages/Pictures1/pentacubes.gif. Acesso em: 07 out. de 2008

2 PROBLEMAS DE ALOCAÇÃO

A alocação de pentominos em superfícies pré-determinadas dá origem a interessantes problemas, descritos neste capítulo. Serão apresentados alguns problemas de alocação, tais como a Cobertura Exata e o Problema dos 20.

Inicialmente, serão apresentadas características que permitem uma melhor compreensão dos problemas. Na Seção 2.6, serão expostos jogos que decorrem dos problemas de alocação. Em seguida, os aspectos educacionais dos jogos serão tratados. Na Seção 2.8, encontram-se alguns sites interessantes sobre os pentominos.

Como foi dito na Seção 1.1, o primeiro artigo que apresentou os problemas de alocação foi publicado em 1954 por Golomb[1]. Neste artigo, Golomb mostra um tabuleiro de dimensões 8x8 que deve ser coberto utilizando-se doze pentominos e um tetromino. Posteriormente, em 1957, Gardner[2] expõe outros problemas de cobertura exata de pentominos, como o Problema da Triplicação, o problema do tabuleiro 8x8 com quatro furos e o Jogo do Último Encaixe. A primeira menção conhecida ao Problema dos 20 data de 1994, quando Golomb[8] apresentou sua definição em seu livro.

Antes de apresentar os problemas de alocação, é necessário definir algumas características peculiares aos mesmos que permitirão melhor compreendê-los.

2.1. Características dos problemas

Foram definidas quatro características comuns a todos os problemas de alocação. São estas:

1. Cardinalidade do conjunto:
    - Todos os 12 devem ser dispostos.
    - Somente um subconjunto deve ser disposto.

2. Repetições:

    - São permitidas, isto é, cada pentomino pode ser disposto várias vezes.
    - São proibidas, ou seja, cada pentomino só pode ser alocado no máximo uma vez.

3. Cobertura:

    - O tabuleiro deve ser totalmente coberto, ou seja, não existirão espaços vazios no final da cobertura.
    - São permitidas áreas vazias no final da cobertura, isto é, são permitidos espaços sem pentominos alocados.

4. Forma do tabuleiro:

    - Exata, o contorno externo é previamente conhecido, com forma regular.
    - Livre, isto é, seu contorno externo permite depressões e elevações e em seu interior são permitidos orifícios.

Deve-se ressaltar que a interdependência de características é possível. Por exemplo, se um tabuleiro cuja área é maior que sessenta unidades deve ser totalmente coberto, repetições precisam ser permitidas.

Os problemas descritos nas próximas seções serão definidos seguindo as características apresentadas.

2.2. Cobertura Exata de Superfícies

O objetivo deste problema é cobrir toda uma superfície, dada como entrada, com exatamente todos os pentominos, sem repetições. Tal problema exige, então, que todos os doze pentominos sejam alocados, sem repetições, de maneira que o tabuleiro, de formato exato, fique totalmente coberto.

Na Figura 2.1, podem-se encontrar um exemplo de tabuleiro retangular e uma de suas coberturas.

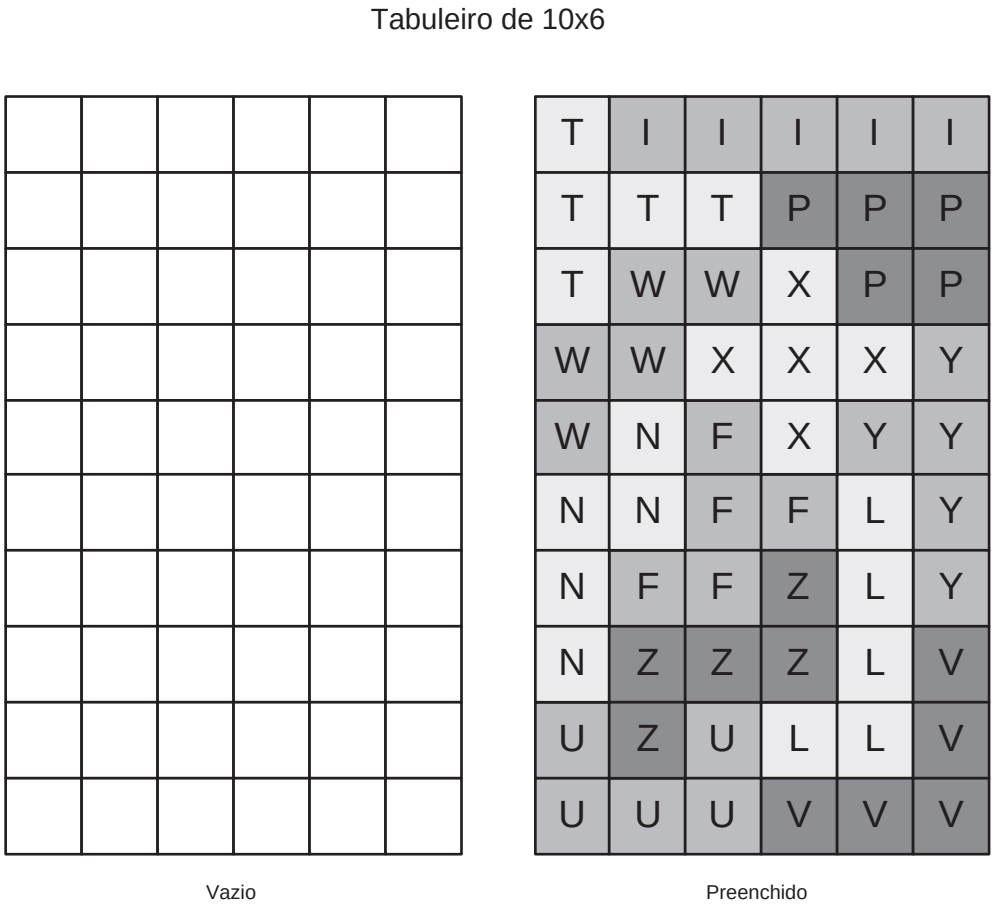


Figura 2.1: Exemplo da Cobertura Exata

Uma observação importante é a área da superfície que, para a cobertura exata, deve ser igual a sessenta unidades, uma vez que cada pentomino utiliza cinco unidades e existem doze pentominos a serem alocados. Somente para as superfícies exatas e retangulares podem-se enumerar todas as possibilidades de dimensões, que são:

- 3 x 20
- 4 x 15
- 5 x 12
- 6 x 10

Claramente as superfícies retangulares de 1 x 60 e 2 x 30 não podem ser cobertas, uma vez que existem pentominos que não se encaixariam.

Uma outra vertente desse problema busca encontrar todas as soluções para uma determinada entrada, e não apenas uma, ou provar que não existe solução. O algoritmo

para resolver este problema claramente é um algoritmo *backtracking*. Para esta vertente é possível enumerar todas as soluções de cada tabuleiro. A Tabela 2.1 apresenta o número de coberturas possíveis para cada tabuleiro exato e retangular, segundo Fletcher[4].

Tabuleiro	Soluções
20x3	2
15x4	368
12x5	1010
10x6	2339

Tabela 2.1: Número de soluções de cada tabuleiro retangular

O Problema da Cobertura Exata inicialmente foi proposto por Golomb em 1954 [1]. Além disso, este problema é a base de um jogo computacional, conhecido como o Jogo do Último Encaixe. Este jogo também foi proposto e comercializado por Golomb com o nome de *Pentominoes*. Em 1996, Orman[9] provou em seu artigo que neste jogo o primeiro jogador sempre vence, desde que jogue de maneira ótima.

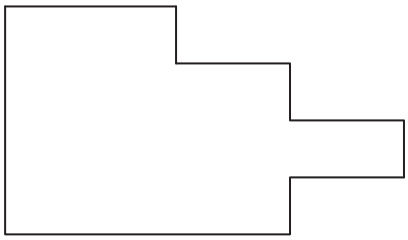
Um outro problema baseado neste foi abordado por Scott [3]. Neste caso o tabuleiro é uma superfície de dimensões 8x8 com um furo de dimensões 2x2 no centro. Dado este tabuleiro, o objetivo é descobrir todas as coberturas possíveis. Para sua resolução, foi criado um algoritmo *backtracking*. Tal algoritmo é um dos primeiros algoritmos *backtracking* conhecidos. Posteriormente, outros tabuleiros com furos aleatórios também foram estudados. No ano 2000, um novo algoritmo *backtracking* que resolve este problema em segundos foi proposto por Knuth [11].

2.3. Problema dos 20

Um outro problema de alocação conhecido é o Problema dos 20. Seu nome é devido à divisão do conjunto de pentominos em três conjuntos distintos de quatro pentominos e, como cada pentomino tem área cinco, o tabuleiro ocupado por cada conjunto tem área total igual a vinte.

Este problema visa cobrir totalmente três tabuleiros iguais, de formato livre e previamente fornecidos (a) com três conjuntos de quatro pentominos distintos, um de cada vez (b), sem repetições. A Figura 2.2 exemplifica este processo.

Deve-se acrescentar que a interseção dois a dois dos conjuntos de pentominos que constituem o Problema dos 20 é vazia e a união dos mesmos resulta no conjunto de todos os pentominos.



a)
Dado um tabuleiro, preencha-o com
três conjuntos de quatro pentominos.

V	V	V				
V	Z	V	W	W		
V	V	W	W	N	N	N
V	V	W	N	N		

U	U	X				
U	X	X	X	Y		
U	U	X	Y	Y	Y	Y
I	I	I	I	I		

b) Resultado:
O tabuleiro foi preenchido com três
conjuntos de quatro pentominos
distintos.

L	L	F				
L	F	F	F	T		
L	P	P	F	T	T	T
L	P	P	P	T		

Figura 2.2: Exemplo do Processo do Problema dos 20

Como foi dito anteriormente, os pentominos são divididos em três grupos que serão utilizados para cobrir três tabuleiros idênticos. Existem casos onde o tabuleiro não permite três coberturas distintas. A Figura 2.3 mostra um desses tabuleiros.

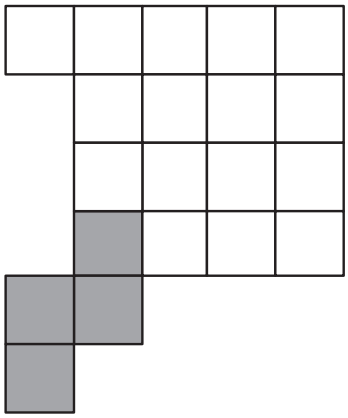


Figura 2.3: Exemplo de tabuleiro que não permite 3 coberturas

No tabuleiro da Figura 2.3, há posições críticas, escurecidas na figura, cujo formato só pode ser coberto por pentominos específicos que, no caso, são o pentomino N e o pentomino W. Sendo assim, existirão apenas duas coberturas. Na primeira, o pentomino N ocupará a posição crítica e na outra o W ocupará.

A Figura 2.4[1] mostra exemplos de tabuleiros cobertos. Como se pode perceber, não existem exigências quanto à forma do contorno fornecido.

Uma variação do Problema dos 20 é o Problema dos 30 que se diferencia do primeiro apenas no número de pentominos de cada subconjunto, ou seja, o Problema dos 30 tem dois conjuntos de seis pentominos, enquanto o Problema dos 20 tem três conjuntos de quatro pentominos.

2.4. Problema da Triplicação

Outro problema de alocação menos conhecido é o problema da triplicação, no qual dado um pentomino qualquer A, deve-se cobrir totalmente a forma triplicada de A com nove pentominos distintos dentre os onze restantes.

A Figura 2.5 apresenta as formas triplicadas dos pentominos W e P já preenchidas.

[1]Fonte: http://pentomino.wirisonline.net/bestanden/20p/image001.gif. Modificada. Acesso em: 07 out. de 2008

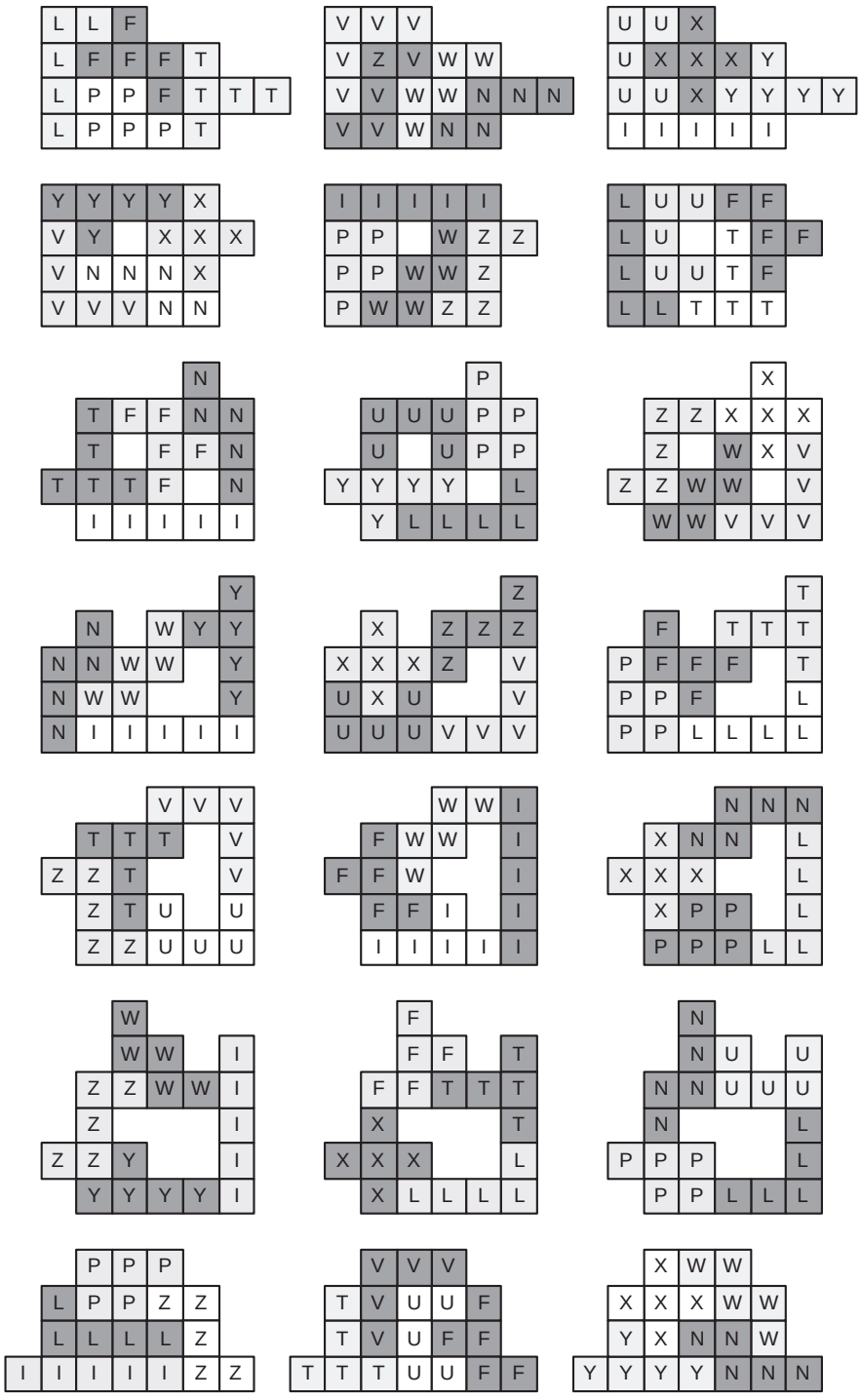


Figura 2.4: Exemplos do Problema dos 20

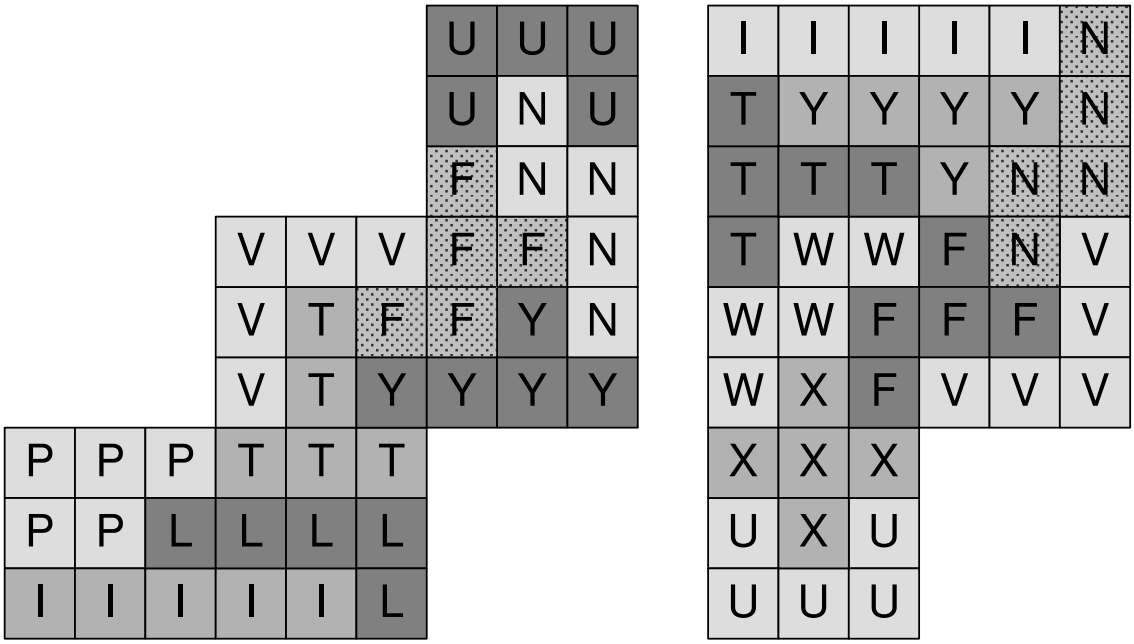


Figura 2.5: Exemplo do Problema da Triplicação

Este problema é proposto por Golomb[8] em 1994. Existe uma variação do mesmo que difere do original apenas pela forma a ser coberta. Enquanto o problema original triplica a forma do pentomino, esta variação visa cobrir a forma duplicada deste utilizando quatro pentominos dos onze disponíveis. Deve-se observar que os tabuleiros com o formato dos pentominos X e V duplicados não possuem coberturas possíveis.

2.5. Cobertura de Paralelogramos Dentados

Neste problema, proposto por Koth e Grosser[12], deve-se cobrir totalmente um paralelogramo "dentado" de área sessenta com doze pentominos distintos. A área de um paralelogramo é calculada através de uma multiplicação de sua base b por sua altura h. Já nos paralelogramos "dentados", a área é obtida através da multiplicação de sua base b, o número de "dentes" d e a altura desses "dentes" s, isto é, $\acute{A}rea = 60 = b * h = b * d * s$. Nota-se que os valores de b, d e s devem ser múltiplos de sessenta. No artigo de Koth e Grosser, podem-se encontrar todas as variações possíveis de paralelogramos "dentados".

A Figura 2.6 ilustra dois paralelogramos "dentados". As dimensões (b, d, s) do primeiro são $(10, 6, 1)$, enquanto as do segundo, $(6, 2, 5)$.

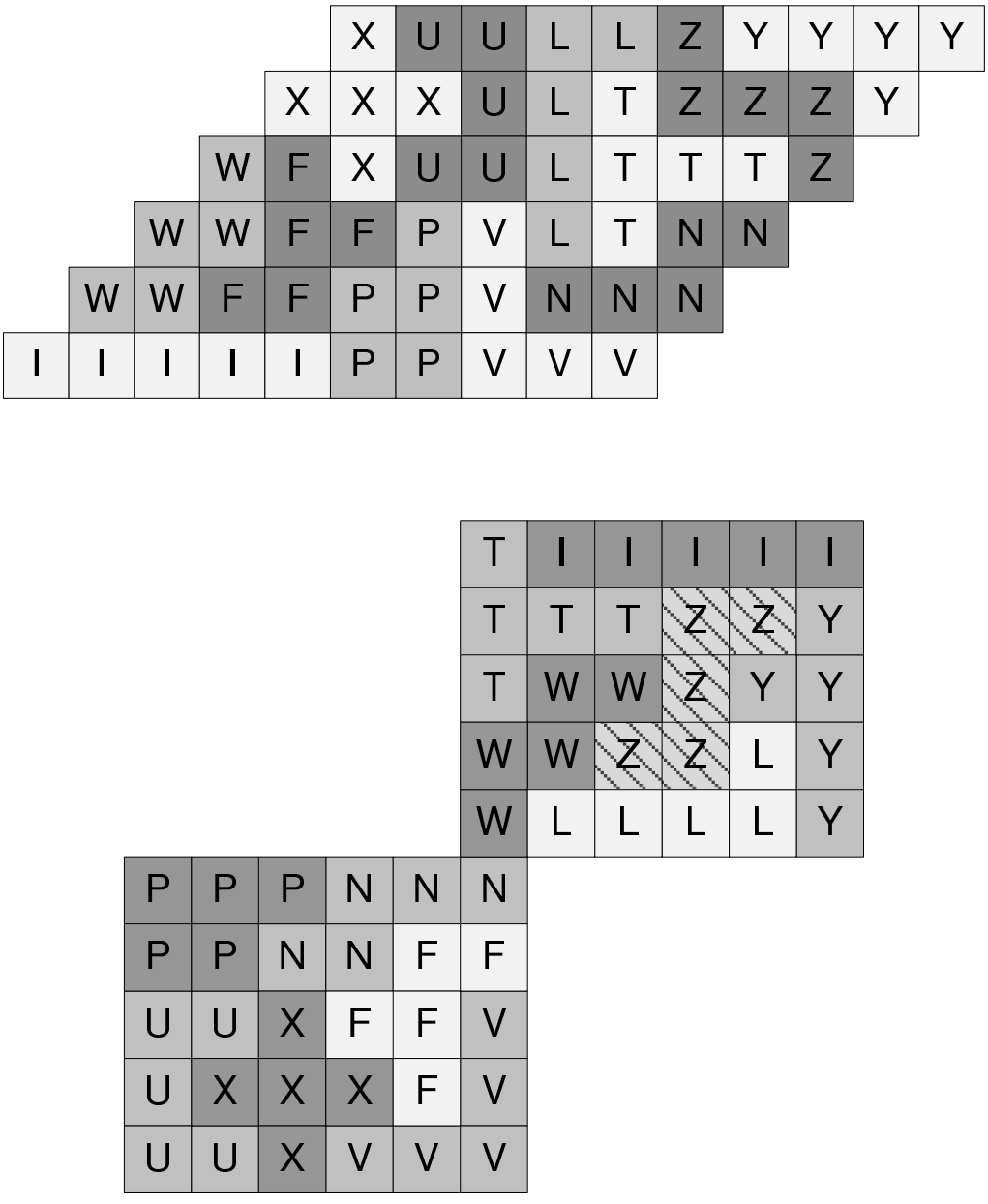


Figura 2.6: Exemplos de paralelogramos "dentados"

2.6. Jogos decorrentes dos problemas

Diversos jogos se encontram disponíveis na rede. Na Seção 2.8, encontram-se alguns endereços. Alguns dos jogos apresentados têm aplicações educacionais. A Seção 2.7 mostra alguns benefícios causados por tal aplicação.

Jogo da Cobertura Exata de Superfície

Número de Jogadores: 1

Objetivo: Encaixar todos os pentominos em um tabuleiro conhecido de tamanho igual a sessenta unidades (espaço ocupado por todos os pentominos).

Regra: É permitido aplicar a cada pentomino as operações de rotação e espelhamento de maneira a encontrar o esquema que encaixe nos espaços vazios.

Jogo do Último Encaixe

A partir do Problema da Cobertura Exata, pode-se obter um jogo conhecido como o Jogo do Último Encaixe. Neste jogo, existem dois ou mais usuários que, a cada rodada, tentam alocar um pentomino de maneira a ocupar o maior espaço possível. Nele, perde o jogador que não conseguir alocar um pentomino. A Figura 2.7 apresenta um tabuleiro totalmente coberto, utilizando cinco pentominos. Pode-se perceber que nenhum outro pentomino pode ser alocado. Golomb [1] afirma que cinco é o menor número de pentominos que podem cobrir um tabuleiro de modo que nenhum outro pentomino possa ser alocado.

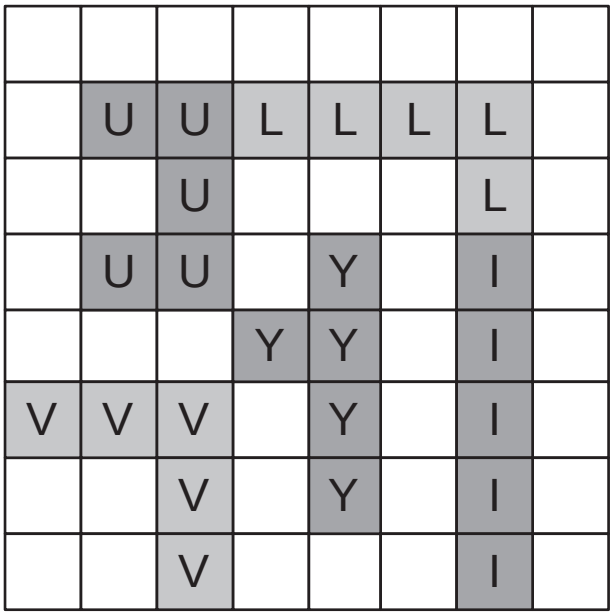


Figura 2.7: Exemplo do Jogo do Último Encaixe

Pode-se notar que a cobertura obtida pode conter os doze pentominos, resultando assim em um problema de cobertura exata. Neste caso, o vencedor será sempre o último jogador.

Número de Jogadores: 2 ou mais.
Objetivo: Ser o último jogador a efetuar uma alocação no tabuleiro.
Regra: A cada pentomino podem ser aplicados operações de rotação e espelhamento de maneira a encontrar o esquema que encaixe nos espaços vazios. Cada jogador efetua um encaixe e então passa a jogada para os outros jogadores que também devem fazer um encaixe. A partir daí, essa operação se repete até que não seja possível encaixar mais peças e, neste caso, esse jogador é considerado o perdedor.

Jogo dos 20

Assim como no Problema dos 20, o jogo automaticamente constrói um tabuleiro aglutinando quatro pentominos e o fornece ao usuário que deve preenchê-lo, usando os pentominos não utilizados.

Número de Jogadores: 1 ou mais.
Objetivo: Cobrir um tabuleiro dado com três conjuntos de quatro pentominos distintos.
Regra: Dado um tabuleiro, deve-se cobri-lo utilizando quatro pentominos distintos. Após essa cobertura, estes pentominos são excluídos do conjunto dos pentominos que podem ser utilizados na próxima cobertura. Dados os pentominos restantes, deve-se cobrir o tabuleiro mais uma vez. Após isso, existem apenas quatro pentominos disponíveis. Deve-se cobrir outra vez este tabuleiro com estes quatro pentominos restantes. Se, em um determinado momento, uma cobertura não for possível, pode-se retornar a uma cobertura anterior, esvaziá-la e refazê-la, utilizando novos pentominos.

Jogo dos 30

Este jogo utiliza-se de todas as regras definidas para o Jogo dos 20. Sua única diferença, conforme definido no Capítulo 2, é o número de pentominos que, no Problema dos 20, são quatro e, no Problema dos 30, são seis.

2.7. Jogos educacionais

A utilização de jogos educacionais que envolvem pentominos tem como objetivo o desenvolvimento do raciocínio lógico e geométrico, isto é, o desenvolvimento de habilidades de visualização e percepção de figuras geométricas. Além disso, os jogos envolvem a prática de relações espaciais e a criação de estratégias de resolução de problemas, desenvolvendo assim as capacidades de visualização, de percepção espacial, de análise e criatividade. Com isso, o jogador terá um pensamento mais analítico e dedutivo. Uma das primeiras sugestões de uso de jogos que envolvem pentominos data de 1990, quando Tracy e Eckart[6] apresentaram os benefícios de sua aplicação. Ainda em 1990, Onslow[7] apresentou outras sugestões de atividades baseadas nos pentominos e seus benefícios, tais como a melhoria na percepção espacial e o desenvolvimento da capacidade lógica.

2.8. Recursos na rede

São muitos os recursos disponíveis na rede sobre pentominos. Nesta seção serão apresentados jogos decorrentes de pentominos na rede e algumas páginas interessantes. O último acesso a estes sítios aconteceu no dia 8 de outubro de 2008. A maioria das páginas apresentadas está em inglês.

Jogos Disponíveis

Na Seção 2.6, foram apresentadas a descrição e as regras de alguns jogos decorrentes de pentominos. Alguns destes jogos já foram implementados e se encontram disponíveis na rede. A seguir, alguns destes serão exibidos.

Polyominoes

Disponível em: http://www.kevingong.com/Polyominoes/.

Este jogo foi desenvolvido por Kelvin L. Gong e engloba dois jogos distintos: o Jogo do Último Encaixe e o Jogo da Cobertura Exata de Superfícies.

O Jogo do Último Encaixe pode ser utilizado de duas maneiras neste aplicativo. Na primeira, não existe disputa e para ganhar basta ser o último a encaixar um pentomino. Já a segunda possibilidade consiste em um pódio entre o computador e o jogador.

O Jogo da Cobertura Exata também pode ser jogado de duas maneiras distintas. Na primeira, o aplicativo cria uma superfície e pede que o jogador a preencha de pentominos. Na outra, o jogador deve tentar encaixar o menor número de pentominos, de maneira a não permitir que sejam colocados mais pentominos.

Este jogo está disponível para *download* e para jogos *on-line*, bastando escolher a opção “Java Version”.

Tetris

Disponível em: http://clickjogos.uol.com.br/Jogos-online/Classicos/Tetris/.

A definição original do jogo envolvia apenas tetrominos, porém existem algumas versões mais atuais que envolvem os pentominos. O jogo Tetris surgiu em 1985 na Rússia.

Basicamente, o jogo de Tetris visa à construção de linhas horizontais ininterruptas. Para que estas linhas sejam formadas, as peças (tetrominos ou pentominos) caem e são guiadas pelo jogador, que pode rotacionar ou espelhar as peças para que estas se encaixem. Quando uma linha inteira é completada, esta é retirada e o jogador recebe pontos. O fim do jogo acontece quando as peças chegam ao topo da superfície determinada.

Chasing Vermeer

Disponível em: http://www.pentomino.nl/main.asp?action=download

Este é um jogo baseado na cobertura exata de superfícies, porém as superfícies podem ter tamanhos diferentes das sessenta unidades. Quando a superfície tem tamanho inferior a sessenta podem sobrar pentominos sem implicar falha.

***Sites* sobre pentominos**

Serão apresentadas algumas páginas disponíveis na rede sobre pentominos. Inicialmente serão expostos sítios focados na definição dos pentominos e dos problemas decorrentes.

http://en.wikipedia.org/wiki/Pentomino

Página muito completa, contendo todas as definições importantes, tanto dos problemas que envolvem pentominos quanto de toda a família de poliminos.

http://pentomino.wirisonline.net/indexe.html

Excelente página sobre os pentominos. Descreve detalhadamente a maioria dos problemas de alocação conhecidos, como, por exemplo, o Problema dos 20. Este sítio apresenta também algumas aplicações educacionais dos pentominos.

http://www.mathematische-basteleien.de/pentominos.htm

Define os pentominos e seus problemas. Entre estes problemas existem alguns incomuns, como o Problema do Anel, no qual se busca encontrar o maior anel fechado formado pelos doze pentominos. Outro problema apresentado é o Problema da Cobertura de Figuras Furadas, isto é, cobrir uma figura com diversos furos com os doze pentominos sem repetições.

http://www.andrews.edu/ calkins/math/pentos.htm

Texto muito bom sobre os pentominos, bem explicado e com boas referências. Apresenta detalhadamente todas as definições importantes para a compreensão do tema.

http://www.recmath.com/PolyPages/index.htm

Apresenta os poliminos, inclusive os 3D(policubos). Além disso, este sítio também expõe outras figuras geométricas bastante incomuns, como os Octacubos

http://www.gamepuzzles.com/polyintr.htm

Página pertencente a uma empresa que fabrica jogos baseados em pentominos. Apresenta toda a história dos pentominos e referências importantes de maneira extremamente detalhada.

http://www.ericharshbarger.org/pentominoes/

Página bastante interessante, pois apresenta alguns problemas incomuns, tais como o maior perímetro possível para um aglutinado.

http://www.monmouth.com/ colonel/pentodd/

Apresenta a definição de um novo problema, que consiste em formar figuras simétricas com um número ímpar de pentominos, isto é, formar figura com algum eixo de simetria. Como exemplo pode-se considerar o pentomino I, que apresenta simetria lateral e é constituído de apenas um pentomino.

http://www.ericharshbarger.org/lego/pentominoes.html

Apresenta a utilização de peças de Lego para construir os doze pentominos. Além disso, expõe diversas outras construções feitas com Lego, como, por exemplo, um tabuleiro de xadrez.

http://www.sasked.gov.sk.ca/docs/midlmath/g842note.html

Apresenta possibilidades do uso de pentominos na educação de geometria, tais como a prática da visão de congruência e simetria, entre outras.

http://www.cimt.plymouth.ac.uk/resources/puzzles/pentoes/pentoint.htm

Esta página mostra alguns dos problemas já apresentados de maneira ilustrada, inclusive o Problema da Triplicação e Duplicação.

A partir de agora, serão apresentados *sites* voltados a jogos baseados nos pentominos.

http://www.fwend.com/pentomino.htm

Apresenta um jogo *online* de cobertura de tabuleiros. Os tabuleiros são pré-estabelecidos e divididos em níveis de dificuldade. O usuário deve cobrir os tabuleiros sem deixar espaços vazios e sem repetir pentominos.

http://kevingong.com/Polyominoes

Apresenta o jogo Polyominoes. No jogo o usuário deve alocar um número pré-determinado de pentominos em tabuleiros também pré-terminados.

As páginas apresentadas a seguir têm como objetivo cobrir tabuleiros computacionalmente.

http://www.xs4all.nl/ gp/PolyominoSolver/Polyomino.html

Apresenta um programa que cobre tabuleiros expostos e previamente criados com poliminos.

http://math.hws.edu/xJava/PentominosSolver/

Este sítio contém um aplicativo que visa cobrir tabuleiros exatos de dimensões diversas.

3 ALGORITMOS BÁSICOS DE MANIPULAÇÃO

Geralmente, os problemas e jogos que envolvem pentominos consideram a alocação de pentominos em tabuleiros. Para a implementação computacional destes problemas e jogos, precisam ser desenvolvidos procedimentos responsáveis pela aplicação de transformações aos pentominos, construção de tabuleiros e a disposição destes pentominos nestes tabuleiros.

O conjunto de transformações válidas é composto de rotações e espelhamentos horizontais e verticais. Já a disposição de pentominos engloba a alocação, validação, remoção e movimentação dos mesmos. A construção de tabuleiros é responsável por preparar os tabuleiros que serão utilizados.

Sendo assim, existem três classes de procedimentos, que são:

- transformações de pentominos;
- disposição de pentominos em tabuleiros;
- construção de tabuleiros.

Para lidar com estes procedimentos computacionalmente, é necessário criar estruturas de dados para armazenar pentominos e tabuleiros. A Seção 3.1, apresenta estas estruturas. Em seguida, na Seção 3.2, os procedimentos que permitem aplicar transformações aos pentominos são mostrados. Na Seção 3.3, são expostos algoritmos que permitem a alocação e a remoção de pentominos em tabuleiros. A Seção 3.4 exibe um algoritmo para a construção de tabuleiros de forma exata. A Seção 3.5 apresenta quatro versões de um algoritmo de construção de tabuleiros de formato livre e uma análise comparativa dos mesmos.

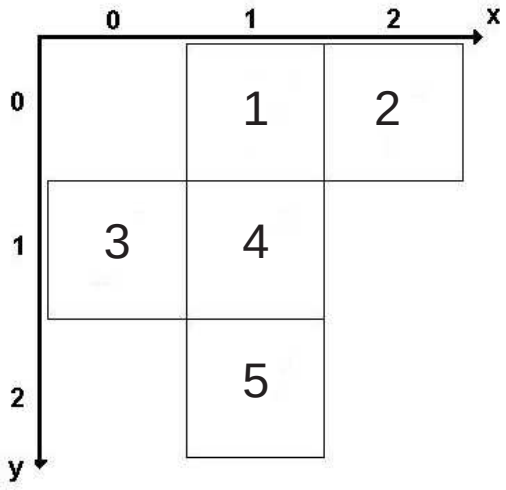
3.1. Estruturas de dados

A estrutura de dados utilizada para representar cada pentomino é um vetor de cinco posições, onde cada posição é um registro com duas variáveis inteiras denominadas

x e y. O número de posições se deve ao número de minos, que é cinco. Este vetor que representa computacionalmente o pentomino é denominado $vPosicoes$.

Cada posição do vetor corresponde a um mino do pentomino, e os valores de x e y dessa posição do vetor armazenam a posição deste mino no plano. É importante destacar que o eixo x cresce para a direita e o eixo y cresce para baixo, o que implica que a posição (0,0) esteja no canto superior esquerdo.

A Figura 3.1 exemplifica esta estrutura de dados.



1		2		3		4		5	
x	y	x	y	x	y	x	y	x	y
1	0	2	0	0	1	1	1	1	2

Figura 3.1: Exemplo da representação computacional de um esquema do Pentomino F

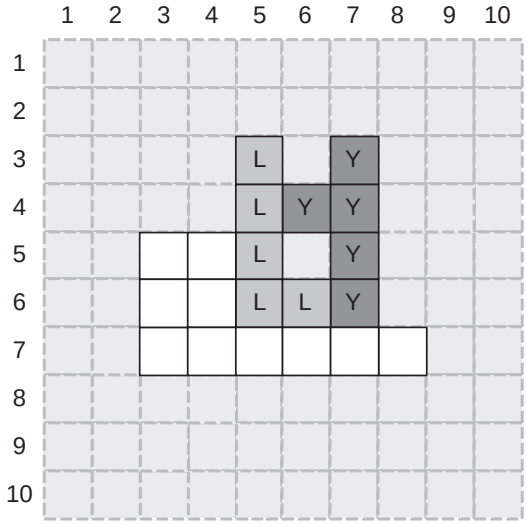
A representação computacional do tabuleiro utiliza uma área de trabalho, área esta que **contém o tabuleiro.** A área de trabalho é uma matriz quadrada, cujos lados devem ter tamanho suficientemente grande para armazenar o maior tabuleiro possível para aquele determinado problema.

Para o Problema dos 20, a matriz deve ter dimensões iguais a trinta e quatro unidades, uma vez que este é o dobro do comprimento do maior tabuleiro formado por quatro pentominos, ilustrado na Figura 3.2. Entretanto, para melhor visualização, os exemplos deste capítulo têm tabuleiros com dimensões menores.

Figura 3.2: Maior cadeia de 4 pentominos

Cada posição da área de trabalho recebe um caracter "*b*" quando a posição está vazia e pertence ao tabuleiro, "0"(zero) quando não pertence ao tabuleiro, ou o nome do pentomino alocado na posição.

A Figura 3.3 exemplifica essa representação computacional.



	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	L	0	Y	0	0	0
4	0	0	0	0	L	Y	Y	0	0	0
5	0	0	*b*	*b*	L	0	Y	0	0	0
6	0	0	*b*	*b*	L	L	Y	0	0	0
7	0	0	*b*	*b*	*b*	*b*	*b*	*b*	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Figura 3.3: Exemplo da representação computacional do tabuleiro

3.2. Transformações de pentominos

Existem três transformações de pentominos:

- rotações;
- espelhamentos horizontais;
- espelhamentos verticais.

Os procedimentos que implementam as transformações têm como entrada o parâmetro $pento$, que representa o pentomino a ser transformado.

Rotações são giros de 90° em sentido horário aplicados a pentominos. Computacionalmente, sua execução altera os valores de x e y de cada posição do vetor que representa o pentomino e sua implementação pode ser vista no procedimento a seguir.

Algoritmo 1 Procedimento $Rotacionar(pento)$

```
% Seja MAX_Y o maior valor de y contido no vetor;
para i:=1 até 5 faça
    old.x := pento.vPosicoes[i].x;
    old.y := pento.vPosicoes[i].y;
    pento.vPosicoes[i].x := MAX_Y − old.y;
    pento.vPosicoes[i].y := old.x;
```

A complexidade desse algoritmo é $O(1)$. A Figura 3.4 ilustra a sua execução.

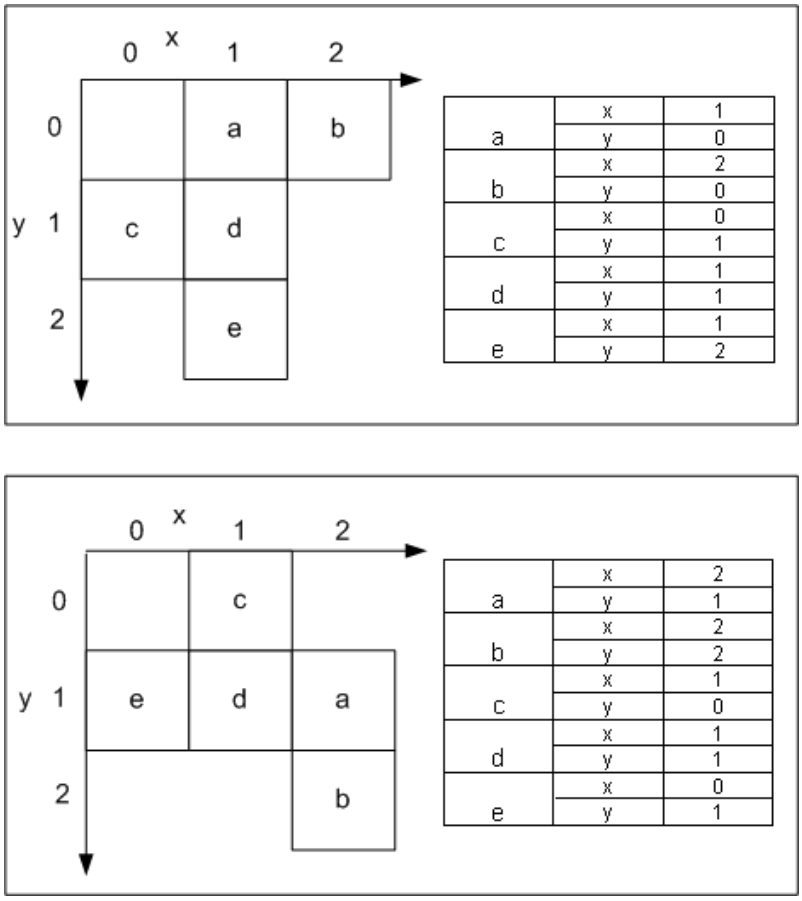


Figura 3.4: Exemplo de aplicação do algoritmo de rotação

Espelhamentos visam obter a reflexão dos pentominos utilizando espelhos virtuais e são classificados como horizontais e verticais de acordo com a posição do espelho.

No vertical, o espelho se localiza no eixo y (vertical) e o pentomino espelhado é a reflexão do pentomino original. O procedimento é visto a seguir.

Algoritmo 2 Procedimento $Espelhar\_V(pento)$

```
% Seja MAX_X o maior valor de x contido no vetor;
para i:=1 até 5 faça
    old.x := pento.vPosicoes[i].x;
    old.y := pento.vPosicoes[i].y;
    pento.vPosicoes[i].x := MAX_X − old.x;
    pento.vPosicoes[i].y := old.y;
```

A Figura 3.5 mostra um exemplo da aplicação desse algoritmo.

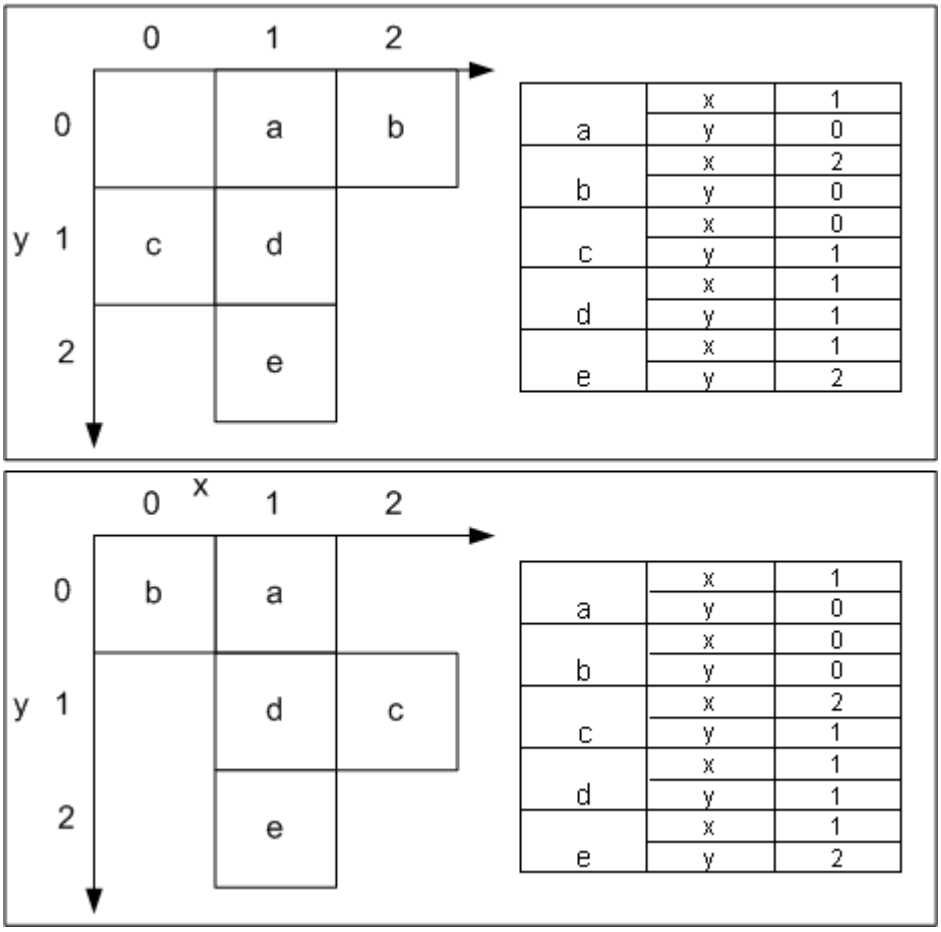


Figura 3.5: Exemplo de aplicação do algoritmo de espelhamento vertical

Já no espelhamento horizontal, o espelho está situado no eixo x (horizontal). Sua implementação pode ser encontrada no Algoritmo 3.

Algoritmo 3 Procedimento $Espelhar\_H(pento)$

```
% Seja MAX_Y o maior valor de y contido no vetor;
para i:=1 até 5 faça
    old.x := pento.vPosicoes[i].x;
    old.y := pento.vPosicoes[i].y;
    pento.vPosicoes[i].x := old.x;
    pento.vPosicoes[i].y := MAX_Y − old.y;
```

A Figura 3.6 mostra um exemplo da aplicação desse algoritmo.

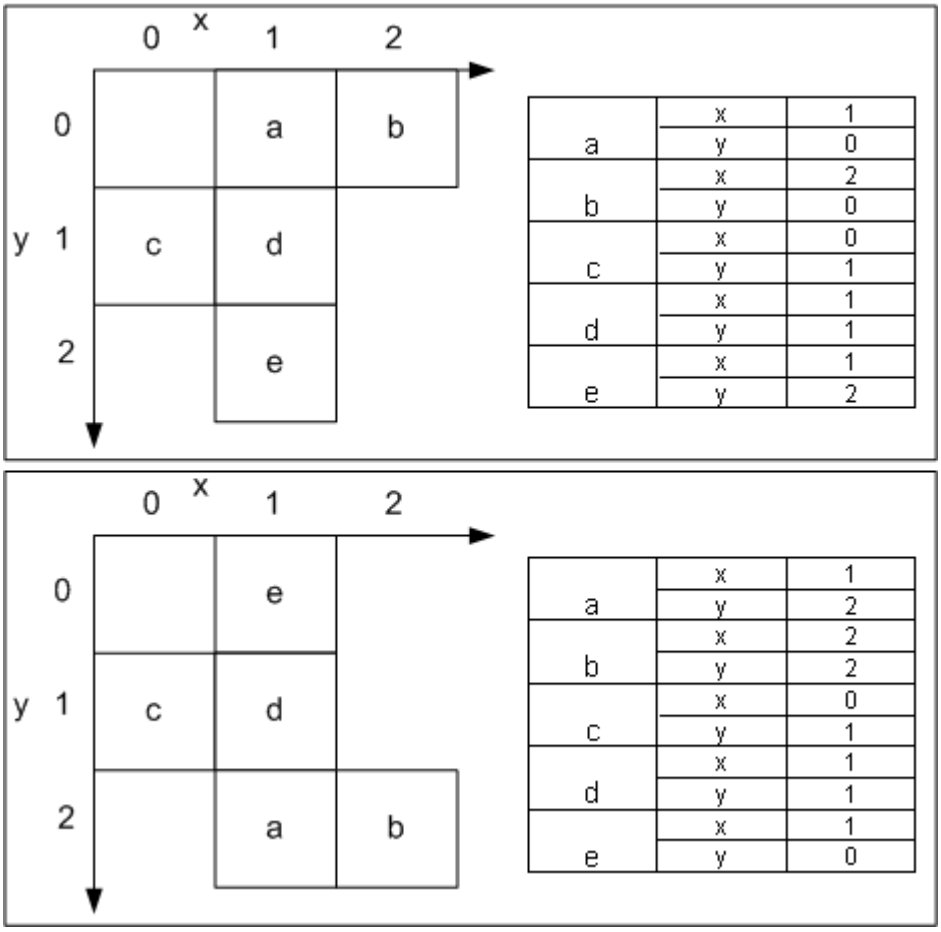


Figura 3.6: Exemplo de aplicação do algoritmo de espelhamento horizontal

Ambos os algoritmos têm complexidade $O(1)$.

3.3. Disposição de pentominos no tabuleiro

Nesta seção são apresentados algoritmos que permitem dispor os pentominos no tabuleiro, isto é, permitem a alocação, a remoção e a movimentação de pentominos. A movimentação é constituída de uma remoção seguida de uma realocação.

O procedimento de alocação é fundamental, tendo em vista que é executado cada vez que um pentomino entra no tabuleiro ou é movido. Para a alocação propriamente dita, é necessário primeiramente validar o posicionamento do pentomino considerado. O procedimento $validar$ é responsável por esta validação prévia.

O procedimento de validação verifica se a colocação de um pentomino no tabuleiro

dado viola as restrições quanto à alocação, ou seja, se o pentomino ultrapassa as bordas do tabuleiro ou se o pentomino se sobrepõe a algum outro já alocado, isto é, possuem pelo menos um mino ocupando a mesma posição no tabuleiro. O algoritmo que se segue tem como entrada a área de trabalho $vMatriz$, cujas dimensões são iguais à $tamMatriz$, o pentomino cuja alocação está sendo validada, denominado $pento$, e a posição (x, y) na qual o pentomino será alocado. A alocação será feita a partir da posição (x, y), isto é, o mino correspondente a posição (0,0) da representação computacional do pentomino será alocado na posição (x, y) e os demais em suas posições correspondentes.

O teste é dividido em dois passos:

1. Testar se algum mino ficará fora do tabuleiro. (a)
2. Testar se todas as posições que o pentomino ocupará estão vazias, garantindo que não existirão sobreposições. (b)

Algoritmo 4 Função $Validar(vMatriz, pento, x, y) : booleano$

$disponivel := \mathbf{Verdadeiro}$;
para $i := 1$ **até** 5 **faça**
    se $(x + pento.vPosicoes[i].y < 1)$ **OU** $(x + pento.vPosicoes[i].y > tamMatriz)$ **OU** $(y + pento.vPosicoes[i].x < 1)$ **OU** $(y + pento.vPosicoes[i].x > tamMatriz)$ **então** (a)
        $disponivel := \mathbf{Falso}$;
    senão
        se $vMatriz[x + pento.vPosicoes[i].y, y + pento.vPosicoes[i].x] \neq$ "b" **então** (b)
            $disponivel := \mathbf{Falso}$;
retornar $disponivel$;

A complexidade deste procedimento é $O(1)$. Sua execução está exemplificada na Figura 3.7. Nesta figura, inicialmente se alocou o pentomino V na posição $(4, 3)$, em seguida se alocou o pentomino F na posição $(4, 4)$. Feito isso, tentou-se alocar o pentomino P na posição $(3, 2)$. Neste caso, pode-se perceber que um mino ficou fora do tabuleiro (posição $(3, 2)$) e outro ficou sobreposto (posição $(4, 3)$), sendo assim, o procedimento de validação retornará $FALSO$, coibindo assim esta alocação. O mesmo raciocínio vale para a tentativa de alocação do pentomino I na posição $(2, 7)$, onde um mino ficou fora do tabuleiro (posição $(2, 7)$).

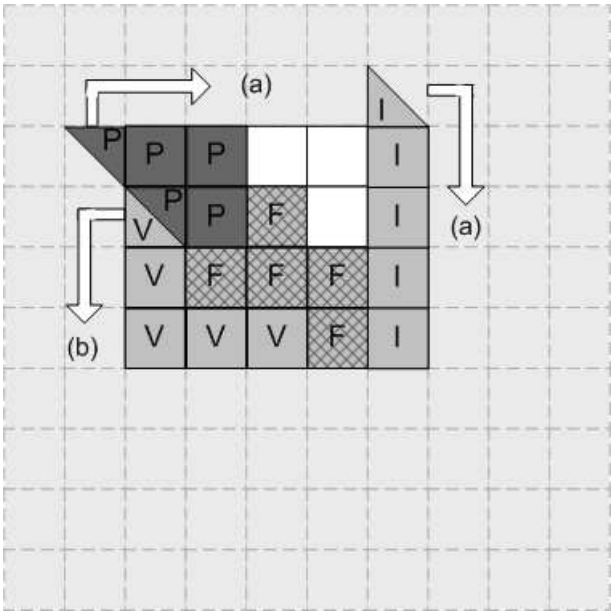


Figura 3.7: Exemplo de aplicação do algoritmo de validação

A alocação de um pentomino é feita através de uma sobreposição, ou seja, o pentomino a ser alocado será sobreposto ao tabuleiro, tendo como posição inicial o valor de (x, y). As demais posições são calculadas de acordo com essa posição inicial.

O procedimento de alocação tem como parâmetros de entrada a área de trabalho $vMatriz$, que contém o tabuleiro, cujas dimensões são iguais a $tamMatriz$, o pentomino a ser alocado, denominado $pento$, a posição na qual o pentomino será alocado (x, y).

Algoritmo 5 Procedimento $Alocar(x, y, pento, vMatriz)$

se $Validar(vMatriz, pento.vPosicoes, x, y) = \textbf{Verdadeiro}$ **então**
    para $i := 1$ **até** 5 **faça**
        $vMatriz[x + pento.vPosicoes[i].y, y + pento.vPosicoes[i].x] := pento.nome$;

O procedimento de alocação tem complexidade $O(1)$.

Assim como o procedimento de alocação, a remoção também é de extrema importância, pois é executada sempre que um pentomino é removido ou realocado.

O procedimento de remoção tem como parâmetros de entrada a área de trabalho $vMatriz$, que contém o tabuleiro, cujas dimensões são iguais a $tamMatriz$, o pentomino a ser removido, denominado $pento$, e a posição (x, y) na qual o pentomino a ser removido foi anteriormente alocado.

Algoritmo 6 Procedimento $Remover(vMatriz, pento, x, y)$

para $i := 1$ **até** 5 **faça**
    $vMatriz[x + pento.vPosicoes[i].y, y + pento.vPosicoes[i].x] :=$ "b";

A complexidade do algoritmo de remoção é $O(1)$.

3.4. Construção de tabuleiros

Como foi dito no Capítulo 2, existem dois tipos de tabuleiros. O primeiro tem formato exato e seus lados são previamente conhecidos. Já o segundo tem formato livre, isto é, seu contorno externo permite depressões e elevações e em seu interior são permitidos orifícios.

Os tabuleiros são utilizados em dois modos distintos, que são:

Modo Construção: Responsável por criar os tabuleiros utilizando regras previamente estabelecidas. Permite acesso livre a toda área de trabalho e a inclusão e exclusão de posições no tabuleiro. São permitidos apenas os valores "b" e "0".

Modo Utilização: Responsável por cobrir o tabuleiro previamente criado. Permite acesso somente as posições pertencentes ao tabuleiro e a conversão de posições vazias em posições ocupadas e vice-versa.

A partir de agora, o estudo da construção de tabuleiros é feito utilizando o Modo Construção.

A construção de tabuleiros retangulares é bastante simples, uma vez que suas dimensões são previamente conhecidas. São estas:

- 3 x 20
- 4 x 15
- 5 x 12
- 6 x 10

O procedimento de construção de tabuleiros retangulares é responsável pela transformação de sessenta posições da área de trabalho $vMatriz$ em posições vazias, isto é, atribuir o valor "b" a estas posições. A função a seguir apresenta o algoritmo de construção de tabuleiros retangulares. Suas entradas são as dimensões h (altura) e l (largura) do tabuleiro. Sua saída é o tabuleiro já formado, pronto para o Modo Utilização.

Algoritmo 7 Função $Gerar\_Tabuleiro\_Retangular(h, l : inteiro) : TMatriz$

para $ct1 := 1$ **até** $tamMatriz$ **faça**
    para $ct2 := 1$ **até** $tamMatriz$ **faça**
        $vMatriz[ct1, ct2] :=$ "0";
para $ct1 : 1$ **até** h **faça**
    para $ct2 := 1$ **até** l **faça**
        $vMatriz[ct1, ct2] :=$ "\b";
retornar $vMatriz$;

A complexidade do algoritmo é $O(tamMatriz^2)$, uma vez que $tamMatriz$ é maior ou igual a maior dimensão do tabuleiro a ser construído, conforme pode ser visto na Seção 3.1.

A construção de outros tipos de tabuleiros exatos segue a idéia geral apresentada nesta seção com variações condizentes com o formato desejado.

3.5. Construção de tabuleiros de formato livre

É tomada como base, nesta seção, a construção de tabuleiros de formato livre, isto é, formados pela aglutinação de $2 \leq n \leq 12$ pentominos, sem repetições. O valor da variável n é modificado de acordo com o problema de alocação, por exemplo, para o Problema dos 20 o valor de n é quatro, enquanto para o Problema dos 30 seu valor é seis.

A construção de tabuleiros é feita de forma iterativa, isto é, pentomino por pentomino. A cada iteração um novo pentomino deve ser acrescentado ao tabuleiro já formado. Este processo deve seguir algumas regras que garantem que a saída do mesmo atenderá as exigências. Tais regras serão apresentadas a seguir.

Antes de apresentar os algoritmos, é necessário definir alguns conceitos.

3.5.1. Aglutinados

Como foi definido anteriormente, os pentominos são formados por cinco minos, que são quadrados. Cada um desses minos tem quatro lados que serão denominados *contatos*. Dois pentominos são *adjacentes* se têm pelo menos um contato em comum. Estes contatos são denominados *internos*, enquanto os outros são *externos*.

Aglutinados são grupos de pentominos onde cada pentomino é adjacente a pelo menos um outro pentomino. Um exemplo pode ser visto na Figura 3.8, onde os pentominos U e P têm dois contatos internos cada.

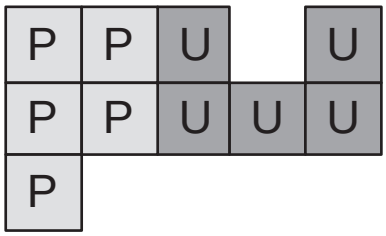


Figura 3.8: Aglutinado de 2 pentominos

Aglutinados não podem ter componentes isoladas, isto é, pentominos sem contatos internos. A geração de tabuleiros livres pode criar componentes isoladas, como a Figura 3.9 ilustra. Nesta figura inicialmente se alocou um pentomino F na posição $(3, 3)$ e, posteriormente, tentou-se aglutinar um pentomino X na posição $(6, 3)$, que é uma posição contígua ao pentomino, resultando, assim, no erro encontrado no exemplo.

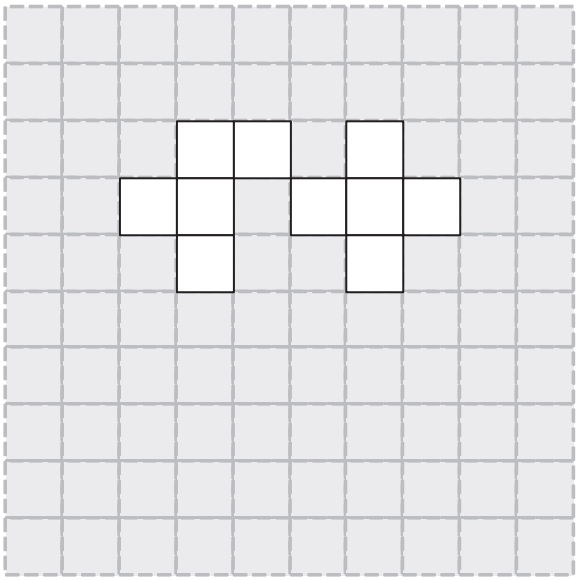


Figura 3.9: Origem das componentes isoladas

A construção de um aglutinado equivale à construção de um tabuleiro de formato livre.

O *perímetro* de um pentomino ou aglutinado corresponde ao número de seus contatos externos. Denota-se o perímetro de um pentomino ou aglutinado A qualquer por $Per(A)$. O perímetro dos pentominos é previamente conhecido, uma vez que as rota-

ções e os espelhamentos não modificam o número de contatos externos. Na Tabela 3.1, encontra-se o perímetro de cada pentomino.

Pentomino	Perímetro
F	12
I	12
L	12
P	10
N	12
T	12
U	12
V	12
W	12
X	12
Y	12
Z	12

Tabela 3.1: Perímetro dos Pentominos

A Figura 3.10 mostra um aglutinado de dois pentominos, cujo perímetro é igual a dezoito unidades. Observa-se que, na face externa, o perímetro totaliza quatorze unidades, embora o perímetro seja dezoito. Essa diferença de quatro unidades é devida ao perímetro da abertura no interior do aglutinado. Sendo assim, o perímetro é a soma do perímetro externo e do perímetro de orifícios internos, quando estes existirem.

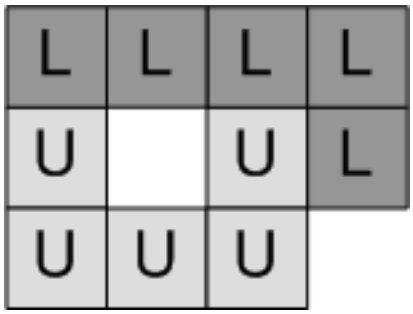


Figura 3.10: Perímetro do aglutinado

Lema 3.1. *A adjacência de dois contatos implica que o perímetro será menor em duas unidades do que o perímetro sem essa adjacência.*

Prova.
Considere dois pentominos A e B quaisquer sem contatos em comum. Considere também o perímetro de A, $Per(A)$ e o perímetro de B, $Per(B)$.

Sabe-se, por definição, que o perímetro é a contagem de contatos externos do aglutinado. Crie a adjacência entre um contato de A e um contato de B. Observe que este contato de A agora tornou-se interno, logo $Per(A)$ deve ser decrementado em uma unidade. O mesmo raciocínio vale para $Per(B)$. Como $Per(A)$ e $Per(B)$ perderam uma unidade cada, seu somatório diminuiu em duas unidades. □

Lema 3.2. *Seja A um aglutinado de $n \leq 12$ pentominos distintos.*

$$Per(A) \leq \begin{cases} \textit{10n + 2} & \textit{Se o pentomino P } \notin A \\ \textit{10n} & \textit{Se o pentomino P } \in A. \end{cases}$$

Prova.
Caso 1: O pentomino P não está contido em A.

> Seja um aglutinado A formado por dois pentominos. Como eles têm, no mínimo, um contato interno seu perímetro é $Per(A) \leq 12 + 12 - 2 = 22 = 10n + 2$.
> Suponha o lema válido para um aglutinado A' com $n - 1$ pentominos. Então
> $$Per(A') \leq 10(n-1) + 2 = 10n - 8.$$
> Ao acrescentar um pentomino ao aglutinado pelo menos um contato externo do aglutinado e um contato externo do pentomino se tornam internos. Então
> $$10n - 8 - 1 + 12 - 1 = 10n + 2.$$

Caso 2: O pentomino P está contido em A.

> O pentomino P tem perímetro de dez unidades, diferentemente dos outros pentominos. Essa diferença de duas unidades deve ser decrementada do total. Logo, o perímetro de um aglutinado que inclua o pentomino P será duas unidades menor. Como o total sem o P vale $10n+2$ e deve-se debitar duas unidades, o total do perímetro do aglutinado onde o pentomino P está incluído é de no máximo $10n + 2 - 2 = 10n$.

□

Com base nos Lemas 3.1 e 3.2, pode-se concluir que quanto menor for o perímetro de um aglutinado de n pentominos mais compacto ele é.

Lema 3.3. *O perímetro de qualquer aglutinado de n pentominos é par.*

Prova.
Sabe-se que o perímetro de todos os pentominos é par. Além disso, qualquer adjacência entre dois contatos implica redução do perímetro em duas unidades. Sabe-se também que a soma ou a subtração entre dois números pares resultará em outro número par.

Sendo assim, pode-se afirmar que a diferença entre o somatório dos perímetros dos pentominos Σ_{peri} e o somatório dos descontos por adjacências entre contatos Σ_{adj} resultará em um número par.

□

O menor perímetro para um aglutinado de quatro pentominos é dezoito unidades, uma vez que esse é o perímetro do menor retângulo que comporta esses quatro pentominos. A Figura 3.11 mostra esse retângulo e uma de suas possíveis coberturas.

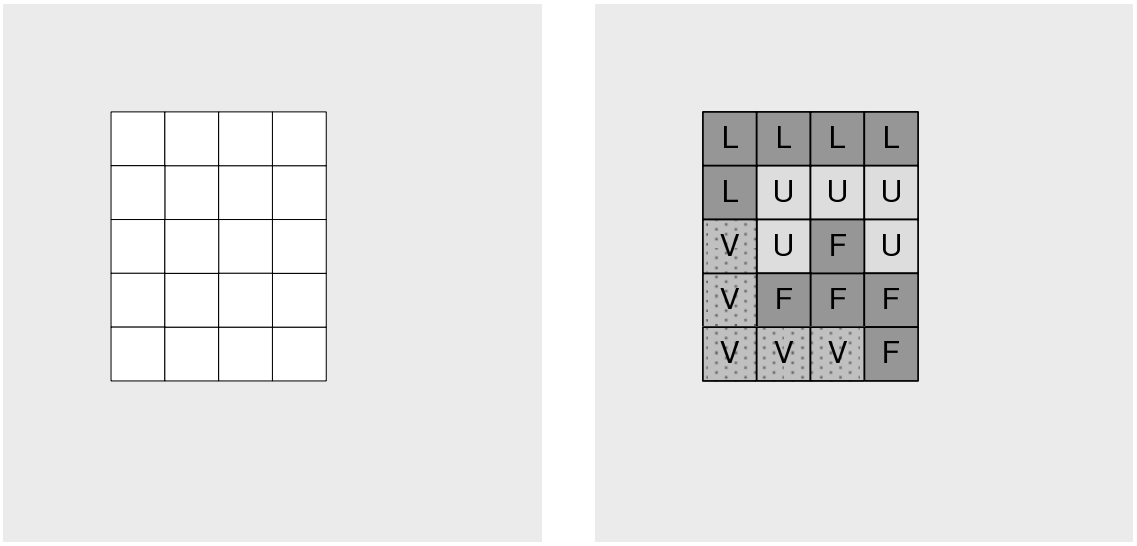


Figura 3.11: Menor perímetro de um aglutinado de 4 pentominos

Existem casos onde o perímetro de um aglomerado antes da alocação de um novo pentomino, é menor do que o perímetro após esta alocação. Conforme o Lema 3.1, a adjacência de dois contatos reduz o perímetro em duas unidades, ou seja, a criação de mais contatos internos diminui o perímetro. Sendo assim, caso uma alocação transforme um grande número de contatos externos em internos, o perímetro será reduzido. A Figura 3.12 apresenta um exemplo deste caso.

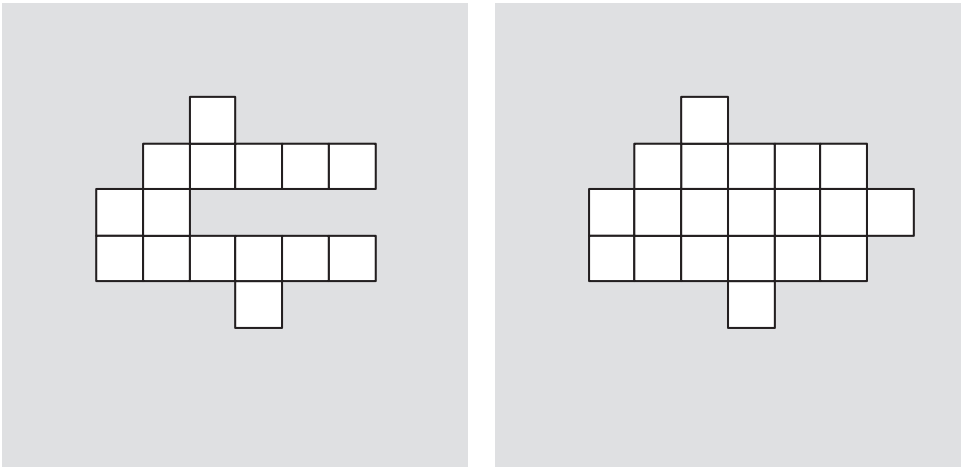


Figura 3.12: Exemplo de redução de perímetro após alocação

Para calcular o perímetro de um determinado aglutinado, o procedimento $Perimetro$ foi implementado. Sua entrada é a área de trabalho $vMatriz$ e suas saídas são um vetor, denominado $PeriLista$, com todas as posições válidas, sem repetições, do perímetro do aglutinado existente e um valor inteiro, conhecido como $peri$, referente ao perímetro do tabuleiro.

O procedimento $Perimetro$ varre a área de trabalho, parando ao encontrar uma posição (x, y) com valor "0". Ao parar, o procedimento verifica as posições ao redor da mesma. Caso a posição acima pertença à área de trabalho e seu valor seja "0" deve-se inserir a posição (x, y) em $PeriListaRep$. Este mesmo processo se aplica às posições situadas à direita, abaixo e à esquerda da posição (x, y). Ao final da varredura, a variável $PeriListaRep$ conterá apenas as posições válidas do perímetro, isto é, posições pertencentes à área de trabalho, cujo valor atribuído seja "0". Deve-se notar que uma mesma posição (x, y) pode ser inserida em $PeriListaRep$ múltiplas vezes. O perímetro $peri$ é calculado pela soma da dimensão de $PeriListaRep$ com o número de contatos no limite da área de trabalho.

Para evitar múltiplas considerações de uma mesma posição, as repetições contidas em $PeriListaRep$ devem ser removidas. Para isso, deve-se varrer todo o vetor $PeriListaRep$. Dada uma posição (x, y), deve-se inseri-la em $PeriLista$ e pular todas as posições iguais a esta. Ao encontrar uma posição diferente, deve-se aplicar este mesmo processo até finalizar a varredura de $PeriListaRep$. Este procedimento só é possível, pois o vetor $PeriListaRep$ é construído de maneira ordenada. Deve-se observar que, na maioria dos casos, o tamanho de $PeriLista$ não representa o perímetro. Este

valor é retornado através da variável $peri$.

A Figura 3.13 ilustra um caso onde o tamanho de $PeriLista$ e o valor de $peri$ são diferentes. No exemplo, o tamanho de $PeriLista$ é onze, enquanto o valor de $peri$ é trinta e quatro. Esta diferença de vinte e três unidades representa o número de posições repetidas excluídas de $PeriLista$ (sete) e de posições que não pertencem à área de trabalho (dezesseis). Um exemplo da inserção de posições repetidas pode ser visto na posição $(2, 4)$. Quando esta posição é considerada, seu valor é "0" e, por isso, seu entorno será testado. A posição acima tem valor "b" e, por isso, a posição $(2, 4)$ será inserida em $PeriListaRep$. A posição à direita também tem valor "b", isto é, $(2, 4)$ será inserida novamente. O mesmo vale para a posição à esquerda. Pode-se perceber que a posição $(2, 4)$ foi inserida três vezes, causando uma repetição em $PeriListaRep$.

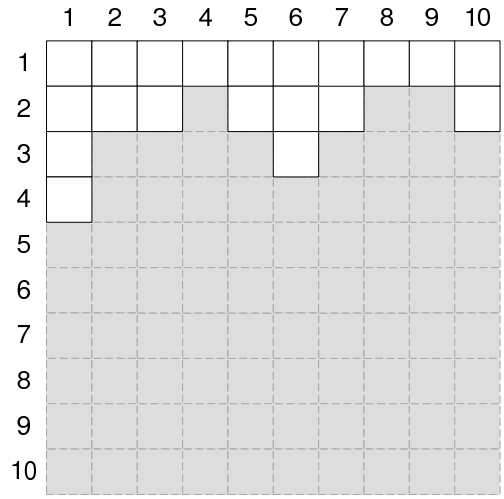


Figura 3.13: Exemplo de caso onde existe diferença entre o tamanho de $PeriLista$ e o valor de $peri$

O algoritmo desenvolvido remove as posições repetidas de $PeriLista$. O procedimento utiliza duas listas para manipular as posições pertencentes ao perímetro. A primeira é denominada $PeriListaRep$ e contém todas as posições pertencentes à área de trabalho contíguas ao perímetro, isto é, $PeriListaRep$ contém posições repetidas. A segunda lista é obtida removendo as posições repetidas de $PeriListaRep$ e é denominada $PeriLista$.

Descrição do algoritmo

Passo 1 Varrer todas as posições da matriz $vMatriz$; para as posições (x, y) cujo valor seja "0", isto é, as posições não pertencentes ao aglutinado, deve-se testar as posições acima, à direita, abaixo e à esquerda. Casos estas pertençam à área de trabalho e seu valor seja igual a "1" deve-se incluir (x, y) na lista $PeriListaRep$;

Passo 2 $peri := |PeriListaRep|$;

Passo 3 Varrer todas as posições contíguas aos limites da área de trabalho. Caso a posição contenha o valor "1" deve-se incrementar em uma unidade o valor de $peri$.

Passo 5 Varrer $PeriListaRep$ inserindo em $PeriLista$ uma única vez cada valor de $PeriListaRep$, removendo, assim, as repetições.

Passo 4 Retornar $peri$ e $PeriLista$;

Algoritmo 8 Procedimento $Perimetro(vMatriz, PeriLista, peri)$

```
{Passo 1:}
PeriListaRep := ∅;
para ct1 := 1 até tamMatriz faça
    para ct2 := 1 até tamMatriz faça
        se vMatriz[ct1, ct2] = "0" então
            se ct1 > 1 então
                se vMatriz[ct1 − 1, ct2] = "b" então
                    inserir(PeriListaRep, ct1, ct2);
            se ct2 < tamMatriz então
                se vMatriz[ct1, ct2 + 1] = "b" então
                    inserir(PeriListaRep, ct1, ct2);
            se ct1 < tamMatriz então
                se vMatriz[ct1 + 1, ct2] = "b" então
                    inserir(PeriListaRep, ct1, ct2);
            se ct2 > 1 então
                se vMatriz[ct1, ct2 − 1] = "b" então
                    inserir(PeriListaRep, ct1, ct2);
{Passo 2:} peri := |PeriListaRep|;
{Passo 3:}
para ct1 := 1 até tamMatriz faça
    se vMatriz[ct1, 1] = "b" então peri := peri + 1;
    se vMatriz[ct1, tamMatriz] = "b" então peri := peri + 1;
    se vMatriz[1, ct1] = "b" então peri := peri + 1;
    se vMatriz[tamMatriz, ct1] = "b" então peri := peri + 1;
ct1 := 1;
enquanto ct1 ≤ |PeriListaRep| faça
    inserir(PeriLista, PeriListaRep[ct1].x, PeriListaRep[ct1].y);
    enquanto (PeriLista[|PeriLista|] = PeriListaRep[ct1]) E (ct1 ≤
|PeriListaRep|) faça
        ct1 := ct1 + 1;
{Passo 4:} retornar peri, PeriLista;
```

Deve-se ressaltar que a saída deste procedimento é uma lista ordenada e sem repetições. Não é necessário aplicar métodos de ordenação para que a lista seja ordenada, uma vez que sua construção é feita seguindo uma ordem tal que o resultado seja uma lista ordenada. Para garantir a inexistência de repetições, aplica-se um método que as remove sem desordenar a lista.

Sua complexidade é $O(tamMatriz^2)$ e sua execução está ilustrada na Figura 3.14.

ct1=1
ct2=1
vMatriz[ct1,ct2] = “0" *(invalida)*
PeriListaRep = [ ]
. . Repetição do algoritmo até ct1=1 e ct2=3 .
ct1=1
ct2=3
vMatriz[ct1,ct2] = “0”
PeriListaRep = [ ]

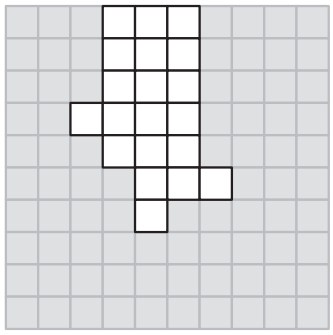
1) ct>1;
2) ct2< tamMatriz; vMatriz[1,3] <> vMatriz[1,4]
PeriListaRep.inserir(1,3)
3) ct1< tamMatriz; vMatriz[1,3] = vMatriz[2,3]
4) ct2>1; vMatriz[1,3] = vMatriz[1,2]

PeriListaRep = [(1,3)]

. . Fim da repetição .

PeriListaRep = [(1,3), (1,7), (2,3), (2,7), (3,3), (3,3), (3,7), (4,2), (4,7), (5,3), (5,3), (5,7), (5,7), (6,4), (6,4), (6,8), (7,4), (7,6), (7,6), (7,7), (8,5)]

vMatriz



peri := |PeriListaRep| = 21
. . Durante o teste das margens, ct1=4 .
vMatriz[ct1,1] <> “ b”
vMatriz[ct1,tamMatriz] <> “ b”
vMatriz[1,ct1] = “ b”
peri := peri +1 = 22
vMatriz[tamMatriz,ct1] <> “ b”
. . Fim dos testes das margens da área de trabalho .
peri = 24

PeriListaRep = [(1,3), (1,7), (2,3), (2,7), (3,3), (3,3), (3,7), (4,2), (4,7), (5,3), (5,3), (5,7), (5,7), (6,4), (6,4), (6,8), (7,4), (7,6), (7,6), (7,7), (8,5)]
PeriLista = [ ]
. . Repetição do algoritmo até ct1=5 .
ct1=5
PeriLista = [(1,3), (1,7), (2,3), (2,7)]

PeriLista.inserir(3,3)

PeriLista[|PeriLista|] = PeriListaRep[ct1] *((3,3) = (3,3))*
ct1:=ct1+1;
. . Fim da repetição da remoção das repetições .
PeriLista=[(1,3), (1,7), (2,3), (2,7), (3,3), (3,7), (4,2), (4,7), (5,3), (5,7), (6,4), (6,8), (7,4), (7,6), (7,7), (8,5)]

Retornar peri e PeriLista

Figura 3.14: Exemplo do funcionamento do procedimento $Perimetro$

Como foi dito no início desta seção, os tabuleiros são utilizados em dois modos distintos. No Modo Utilização, permite-se apenas acessar e modificar as posições pertencentes ao tabuleiro contido na área de trabalho, ou seja, durante esta etapa permite-se alocar e remover pentominos nas posições vazias do tabuleiro. Sendo assim, o algoritmo de validação neste modo deve apenas verificar se todas as posições que seriam ocupadas por uma determinada alocação são vazias e pertencem ao tabuleiro. Essa validação foi definida na Seção 3.3.

Durante o Modo Construção, é permitido acessar e modificar todas as posições da área de trabalho. Seu objetivo é criar tabuleiros aglutinando pentominos, de modo que não existam sobreposições de pentominos e pentominos isolados do restante do aglutinando. Sendo assim, um método de validação para o Modo Construção deve testar se todas as posições que serão ocupadas pelo pentomino a ser aglutinado não pertencem ao aglutinado existente, isto é, o valor destas posições na matriz que representa a área de trabalho contém o valor "0" e testar se este novo pentomino a ser aglutinado terá contatos internos.

A validação no Modo Construção consta de três passos.

Passo 1 Testar se algum mino ficará fora do tabuleiro.

Passo 2 Testar se todas as posições que o pentomino ocupará não pertencem ao aglutinado existente, garantindo que não existirão sobreposições.

Passo 3 Testar se o pentomino terá contatos internos.

O algoritmo a seguir mostra os detalhes desta validação. Suas entradas são a matriz $vMatriz$, que representa a área de trabalho onde o tabuleiro está sendo construído, a posição na qual o pentomino será aglutinado (pos_x, pos_y) e o pentomino $pento$ cuja aglutinação está sendo validada. O algoritmo é semelhante ao Algoritmo 4, exceto pela implementação do Passo 3.

Para verificar se o pentomino a ser aglutinado terá contatos internos, deve-se inicialmente calcular os pontos que este ocuparia se fosse aglutinado. Posteriormente, deve-se obter a $PeriLista$ do aglutinado existente. Feito isso, deve-se testar se alguma das posições calculadas é igual a uma posição pertencente à $PeriLista$, pois as posições desta representam as posições contíguas ao aglutinado existente.

Algoritmo 9 Função $ValidarConstrucao(vMatriz, pos_x, pos_y, pento) : booleano$

```
PosValida := Verdadeiro;
ct1 := 1;
enquanto (PosValida = Verdadeiro) E (ct1 <= 5) faça
    {Passo 1:}
    se pos_x + pento.vPosicoes[ct1].y > tamMatriz então
        PosValida := Falso;
    se pos_x + pento.vPosicoes[ct1].y < 1 então
        PosValida := Falso;
    se pos_y + pento.vPosicoes[ct1].x > tamMatriz então
        PosValida := Falso;
    se pos_y + pento.vPosicoes[ct1].x < 1 então
        PosValida := Falso;
    {Passo 2:}
    se PosValida = Verdadeiro então
        se vMatriz[pos_x+pento.vPosicoes[ct1].y, pos_y+pento.vPosicoes[ct1].x] ≠
"0" então
            PosValida := Falso;
    ct1 := ct1 + 1;
{Passo 3:}
se posValida = Verdadeiro então
    para ct1 := 1 até 5 faça
        posAlocaMino[ct1].x := pos_x + pento.vPosicoes[ct1].y;
        posAlocaMino[ct1].y := pos_y + pento.vPosicoes[ct1].x;
    posValida := Falso;
    Perimetro(vMatriz, PeriLista, peri);
    ct1 := 1;
    enquanto (ct1 ≤ |PeriLista|) E (posValida = Falso) faça
        ct2 := 1;
        enquanto (ct2 ≤ 5) E (posValida = Falso) faça
            se (PeriLista[ct1].x = posAlocaMino[ct2].x) E (PeriLista[ct1].y =
posAlocaMino[ct2].y) então
                posValida = Verdadeiro;
            ct2 := ct2 + 1;
        ct1 := ct1 + 1;
retornar posValida;
```

Como foi dito anteriormente, quando o Modo Construção é utilizado, todas as posições da área de trabalho podem ser alteradas. Seus possíveis valores durante este modo

são "$\not 0$" e "$0''$, que representam, respectivamente, pertence e não pertence ao tabuleiro.

No Modo de Utilização, uma posição vazia pode ser modificada aplicando-se o algoritmo de alocação. Já no Modo de Construção, as modificações na área de trabalho acontecem aglutinando pentominos ou removendo pentominos do aglutinado existente.

Os tabuleiros são criados pentomino a pentomino, isto é, um primeiro pentomino é posicionado e os outros são aglutinados a este. Para efetuar a aglutinação dos pentominos, o Algoritmo 10 foi criado.

Este algoritmo tem como pré-requisito a validade da alocação, isto é, esta função só pode ser executada caso a função $ValidarConstrucao$ retorne **Verdadeiro**.

As entradas do procedimento $Aloca - Desaloca$ são a posição (x, y) na qual o pentomino $pento$ será aglutinado e a matriz $vMatriz$ que representa a área de trabalho e o parâmetro $nome$. O parâmetro $nome$ somente pode assumir dois valores: "$\not 0$" ou "0". Seu valor indica que o pentomino será aglutinado ou removido, respectivamente. Deve-se ressaltar que para a remoção de um pentomino do aglutinado a posição (x, y) e o vetor $vPosicoes$, que representa o pentomino $pento$, devem ser iguais na aglutinação e na remoção.

Algoritmo 10 Procedimento $Aloca - Desaloca(x, y, pento, vMatriz, nome)$

se $(nome = \text{"}\not 0\text{"})$ **OU** $(nome = \text{"}0''\text{"})$ **então**
  para $i := 1$ **até** 5 **faça**
    $vMatriz[x + pento.vPosicoes[i].y, y + pento.vPosicoes[i].x] := nome$;

3.5.2. Algoritmos de construção

A função dos algoritmos de construção é fornecer tabuleiros, cujas formas são obtidas através da aglutinação de n pentominos distintos, definidos de maneira randômica.

Serão apresentadas quatro versões do algoritmo de construção. As versões 1.1 e 1.2 são heurísticas aleatórias, enquanto as versões 2.1 e 2.2 são heurísticas gulosas. A versão 1.1 aglutina os n pentominos de maneira totalmente randômica sem se preocupar com a qualidade do tabuleiro construído. A versão 1.2 implementa uma melhoria à versão 1.1. Nesta melhoria, o tabuleiro construído é submetido a uma averiguação de sua qualidade que pode aprová-lo ou não. Já a versão 2.1 agrega conhecimento à construção, isto é, ao invés de medir a qualidade do tabuleiro após a sua construção, esta versão busca a cada aglutinação melhorar ao máximo a qualidade do tabuleiro. A versão 2.2 incorpora uma nova melhoria aos resultados, baseando-se em uma particularidade do algoritmo de aglutinação.

A entrada dos algoritmos de construção é o número de pentominos $2 \leq n \leq 12$ que formam o aglutinado. Sua saída é a matriz $vMatriz$ que representa a área de trabalho,

na qual o tabuleiro está contido.

Em todas as versões implementadas, existem instruções em comum, por isso estas foram agrupadas no procedimento $Inicializar$. Sua função é inicializar a área de trabalho, sortear os n pentominos, ordená-los em uma lista $Pent$, que indicará a ordem em que serão considerados para formar o aglutinado que constitui o tabuleiro, aplicar de 0 a 3 rotações e 0 ou 1 espelhamentos verticais e horizontais a cada pentomino. Cada posição dessa lista é composta por dois campos, são estes:

$pentomino$ Armazena o pentomino a ser considerado. A este pentomino podem ser aplicadas rotações e espelhamentos.

$tocado$ Número de pentominos já aglutinados quando este pentomino foi processado pela última vez. A principal função deste campo é evitar *loops* infinitos, conforme será visto durante as descrições dos algoritmos.

Algoritmo 11 Procedimento $Inicializar(vMatriz, Pent)$

para $ct1 := 1$ **até** $tamMatriz$ **faça**
    para $ct2 := 1$ **até** $tamMatriz$ **faça**
        $vMatriz[ct1, ct2] := 0$;
para $ct1 := F$ **até** Z **faça** $usado[ct1] :=$ **Falso**;
para $ct1 := 1$ **até** n **faça**
    repita
        $temp := random(F, Z)$;
    Até que $usado[temp] =$ **Falso**;
    $Pent[ct1].pentomino := temp$;
    $Pent[ct1].tocado := 0$;
    $usado[temp] :=$ **Verdadeiro**;
    para $ct2 := 1$ **até** $random(0, 3)$ **faça**
        $Rotacionar(Pent[ct1].pentomino)$;
    para $ct2 := 1$ **até** $random(0, 1)$ **faça**
        $Espelhar\_V(Pent[ct1].pentomino)$;
    para $ct2 := 1$ **até** $random(0, 1)$ **faça**
        $Espelhar\_H(Pent[ct1].pentomino)$;

A função deste algoritmo de construção é retornar um tabuleiro de formato livre, formado pela aglutinação de n pentominos. O valor de n varia de acordo com o problema de alocação no qual o tabuleiro será utilizado, por exemplo: para o Problema dos 20 o valor de n é quatro. Este tabuleiro de formato livre será construído em uma área de trabalho.

3.5.3. Algoritmo de construção: Versão 1.1

Este algoritmo aglutina os pentominos de maneira randômica respeitando as restrições quanto à alocação e à definição de aglutinado, a qual pode ser vista na Seção 3.5.1, garantindo que não existam pentominos isolados dos demais.

Os n pentominos sorteados são inicialmente colocados em uma fila e serão considerados nesta ordem. Se, em um determinado momento, a alocação do pentomino não puder ser efetuada, ele será substituído pelo próximo da fila e perderá seu lugar, indo para o fim da fila. Se, ao voltar a ser considerado, nenhum outro pentomino tenha sido alocado (é a indicação de que o algoritmo entrou em *loop*), o tabuleiro, então, deve ser descartado. Para este processamento, cada posição da fila armazena além do pentomino um valor inteiro correspondente ao número de pentominos já alocados. Este campo é inicializado com 0.

Descrição do algoritmo:

Primeiro pentomino

Chamar procedimento $Inicializar(vMatriz, Pent)$, alocar o primeiro pentomino ($Pent[1]$) na posição ($tamMatrizDIV2$), atribuindo 1 a $Pent[1].tocado$.

Demais pentominos

Seja $Pent[ct1]$ o pentomino sendo considerado. Chamar o procedimento $Perimetro$. Sortear uma posição (x, y) de $PeriLista$. Chamar o procedimento $ValidarConstrucao$ para a posição (x, y). Caso a função retorne **Verdadeiro**, então deve-se alocar o pentomino $Pent[ct1].pentomino$ na posição (x, y), atualizar o valor de $Pent[ct1].tocado$ e repetir este passo para os demais pentominos. Caso a função retorne **Falso**, deve-se eliminar esta posição de $PeriLista$ e refazer o sorteio até encontrar uma posição cujo retorno seja **Verdadeiro**. Se $PeriLista$ for esgotada, o elemento $Pent[ct1]$ passa para o fim da fila, seu valor de $tocado$ é atualizado e o próximo elemento é considerado, o que só ocorre se o valor de $tocado$ deste novo pentomino for diferente do número de pentominos já alocados. Se não houver um pentomino a considerar, o tabuleiro deve ser descartado.

Baseado na descrição do processo de construção, o Algoritmo 12 foi desenvolvido.

Variáveis utilizadas:

$ct1$	Contador de iterações. Indica qual pentomino está sendo alocado e quantos já foram alocados $(ct1-1)$.
$ct2$	Contador temporário de tipo inteiro.
$descartar$	Variável do tipo booleano. Quando seu valor é verdadeiro o tabuleiro deve ser descartado e o algoritmo deve voltar ao início.
$vMatriz$	Área de trabalho.
$Pent$	Fila que armazena os pentominos sorteados e o número de pentomino alocados durante o último processamento deste pentomino.
$PeriLista$	Vetor que armazena as posições contíguas ao perímetro do aglutinado existente.
$tamPeriLista$	Variável que armazena o tamanho de $PeriLista$ inicial, isto é, antes que qualquer posição seja removida.
pos_x	Variável inteira que armazena a posição em que o pentomino deve ser alocado.
pos_y	Variável inteira que armazena a posição em que o pentomino deve ser alocado.
$PosValida$	Variável do tipo booleano utilizada durante a validação da alocação.

Algoritmo 12 Função $Gerar\_Tabuleiro\_V1.1(n) : vMatriz$

```
ct1 := 1;
descartar := Falso;
repita
    se ct1 = 1 OU descartar = Verdadeiro então
        ct1 := 1;
        descartar := Falso;
        Inicializar(vMatriz, Pent)
        meio := tamMatriz DIV 2;
        Aloca − Desaloca(meio, meio, Pent[1].pentomino, vMatriz, "b");
        Pent[1].tocado := 1;
        Perimetro(vMatriz, PeriLista);
    senão
        repita
            tamPeriLista := |PeriLista|;
            posicao := random(1, tamPeriLista);
            pos_x := PeriLista[posicao].x;
            pos_y := PeriLista[posicao].y;
            PosValida := ValidarConstrucao(vMatriz, pos_x, pos_y,
Pent[ct1].pentomino);
            se PosValida = Falso então
                PeriLista[posicao] := PeriLista[tamPeriLista];
                tamPeriLista := tamPeriLista − 1;
                se tamPeriLista = 0 então
                    Pent[ct1].tocado := ct1;
                    remover(Pent, ct1, pentoTemp);
                    inserir(Pent, n, pentoTemp);
                    se Pent[ct1].tocado ≥ ct1 então
                        descartar := Verdadeiro;
                    senão
                        Perimetro(vMatriz, PeriLista);
        Até que PosValida = Verdadeiro OU descartar = Verdadeiro;
        se descartar = Falso então
            Aloca − Desaloca(pos_x, pos_y, Pent[ct1].pentomino, vMatriz, "b");
            Pent[ct1].tocado := ct1;
            Perimetro(vMatriz, PeriLista);
    ct1 := ct1 + 1;
Até que (ct1 > n) E (descartar = Falso);
```

A Figura 3.15 exemplifica a execução do algoritmo.

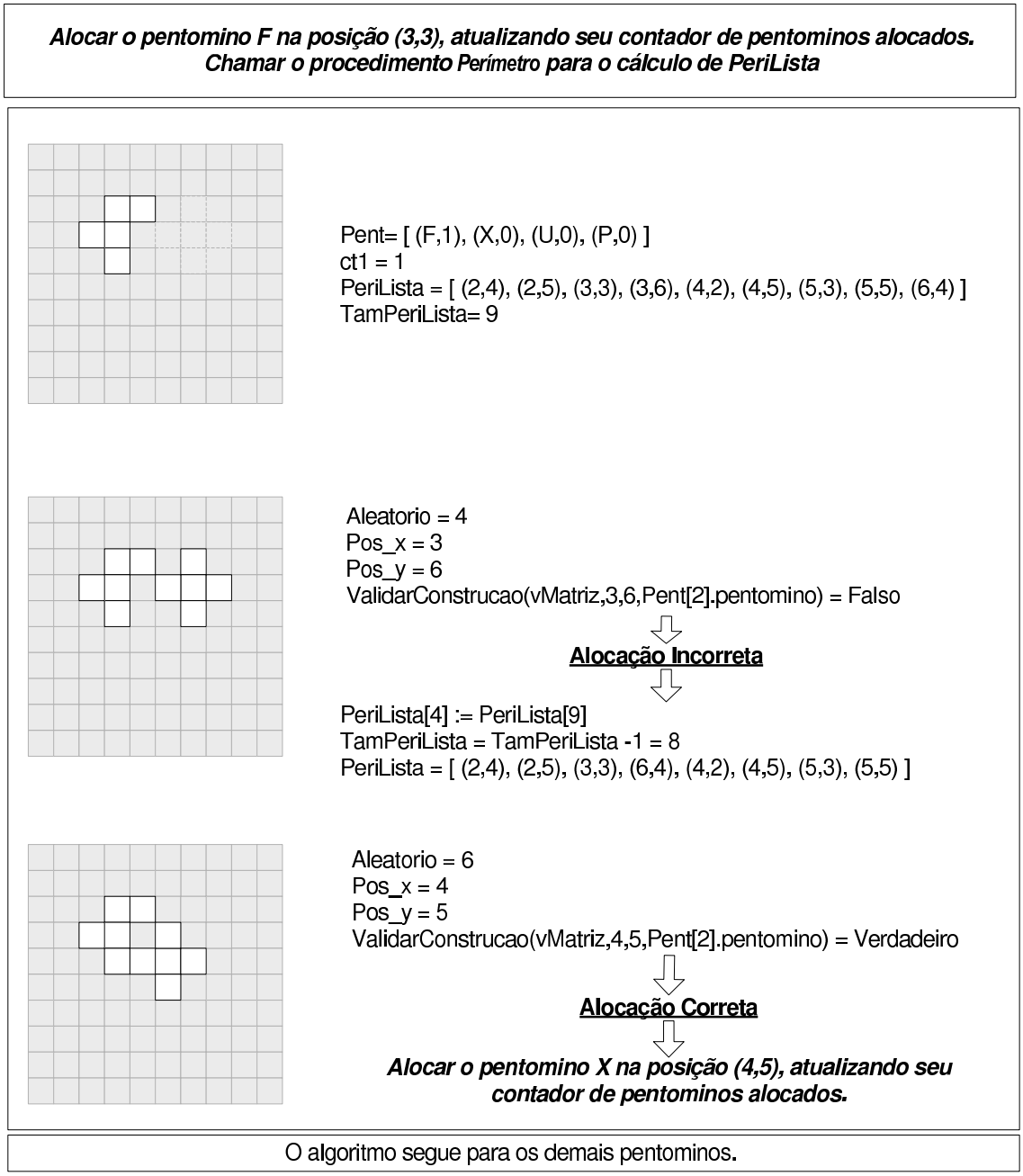


Figura 3.15: Exemplo do funcionamento da versão 1.1 do algoritmo

A Figura 3.16 ilustra um caso onde o tabuleiro é descartado. Já a Figura 3.17

expõe um caso onde um pentomino não pode ser alocado, é enviado para o fim da fila $Pent$ e em um outro momento é alocado, construindo um tabuleiro ao final do processo.

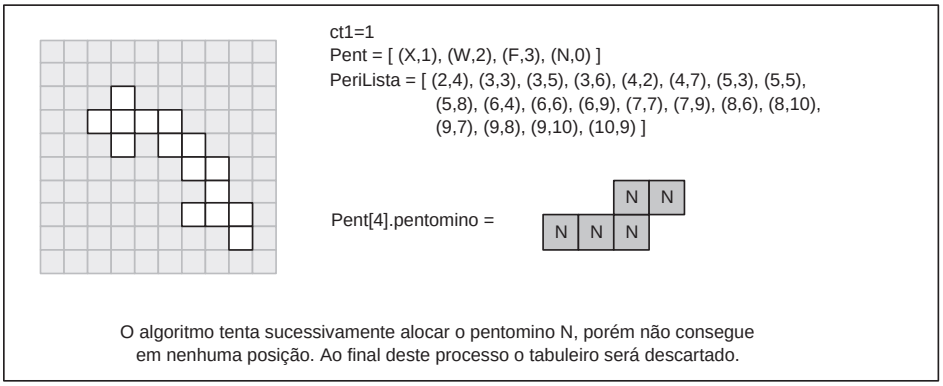


Figura 3.16: Exemplo do descarte do tabuleiro por sucessivas posições incorretas

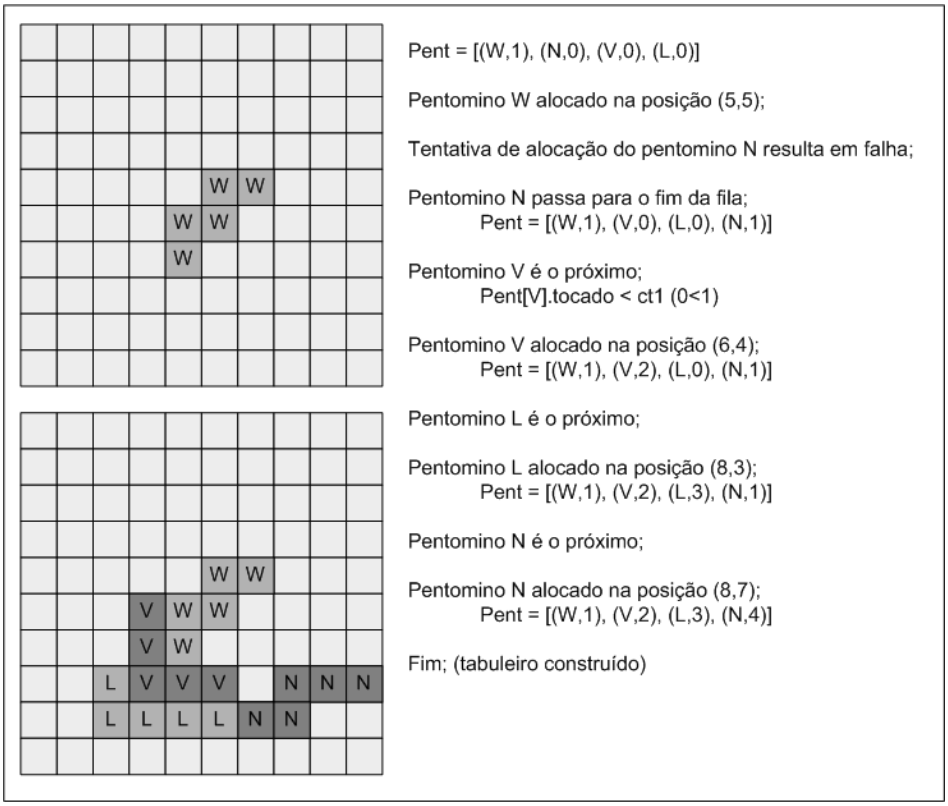


Figura 3.17: Exemplo da aplicação da fila de pentominos

Observou-se, utilizando alguns exemplos dentre os quais os exibidos na Figura 3.18, o baixo nível de coesão dos aglutinados resultantes do Algoritmo 12. Isto ocorre porque o algoritmo não efetua nenhum teste durante ou após a construção dos tabuleiros para regular o perímetro final. Resolveu-se, então, avaliar melhor a atuação do algoritmo, efetuando uma bateria de testes.

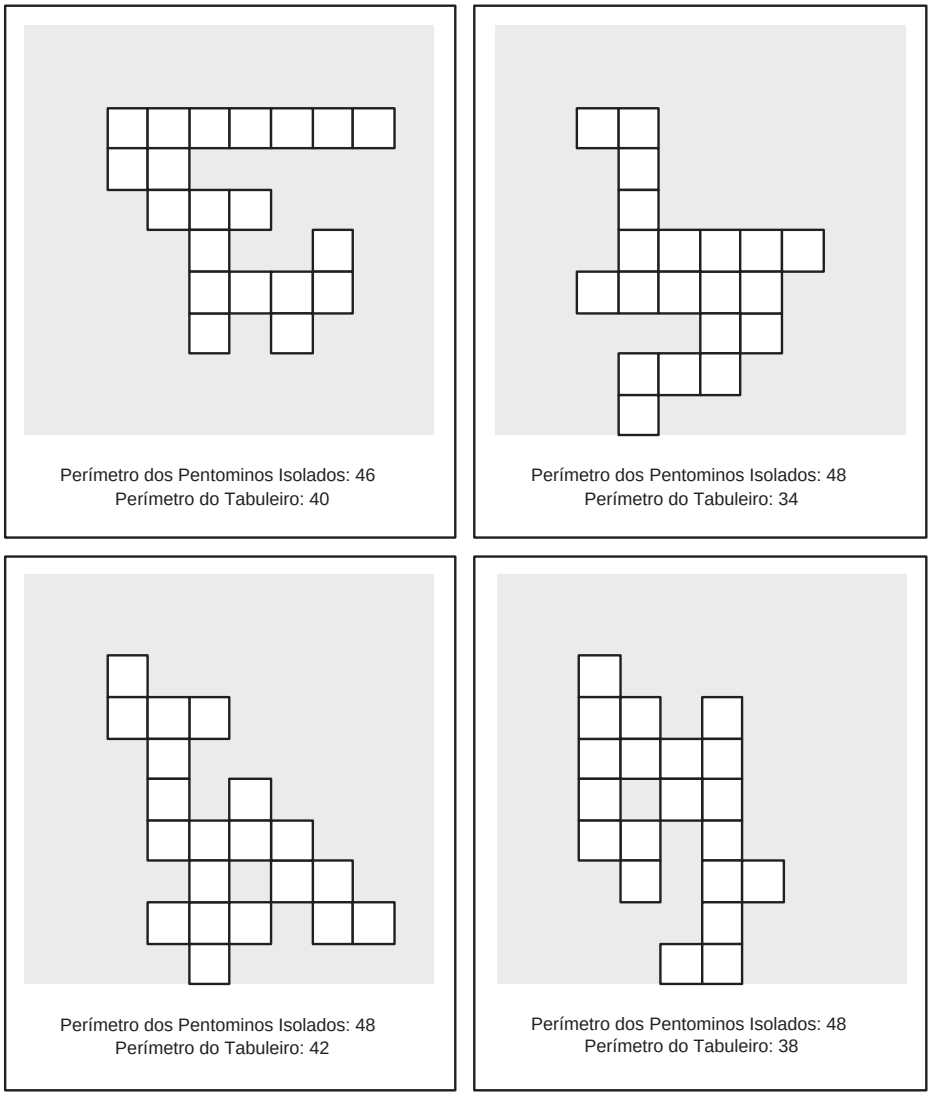


Figura 3.18: Tabuleiros construídos pela versão 1.1 do algoritmo

A Tabela 3.2 ilustra o resultado dessa avaliação. Esta tabela foi obtida através da construção sucessiva de dez mil tabuleiros. Foram destacadas cinco faixas de perímetro.

A primeira faixa contém apenas tabuleiros com perímetro igual a dezoito, uma vez que estes são os melhores tabuleiros possíveis e devem ser destacados.

Intervalos de perímetros	Número de tabuleiros	% de tabuleiros
18-18	0	0%
19-24	40	0,4%
25-30	1047	10,47%
31-36	5094	50,94%
37-42	3819	38,19%

Tabela 3.2: Resultados da versão 1.1 do algoritmo

A média dos perímetros dos tabuleiros é de 35,667, o que comprova o baixo nível de coesão dos tabuleiros construídos. Sendo assim, para melhorar estes resultados foi definido um teste de qualidade a ser incorporado ao algoritmo, conforme será visto na Seção 3.5.4.

3.5.4. Algoritmo de construção: Versão 1.2

Para melhorar os resultados obtidos pela Versão 1.1 do algoritmo, foi definido um teste de qualidade, isto é, o tabuleiro obtido no final do processo de construção será submetido a uma medição que pode aprová-lo ou reprová-lo.

Para testar a qualidade de um tabuleiro, utiliza-se seu perímetro. Quanto menor o perímetro, mais aglutinado é o tabuleiro, isto é, mais faces internas existem, como foi provado no Lema 3.1. Para um determinado tabuleiro ser aprovado, seu perímetro deve ser menor do que o perímetro solicitado.

Descrição do algoritmo:

Passo 1 Chamar a função $Gerar\_Tabuleiro\_V1.1(n)$.

Passo 2 Avaliar qualidade do tabuleiro construído.

Passo 2.1 Tabuleiro Reprovado: Descartar o tabuleiro e retornar ao Passo 1.

Passo 2.2 Tabuleiro Aprovado: Retornar tabuleiro e fim do algoritmo.

A Figura 3.19 ilustra a aplicação do teste de qualidade.

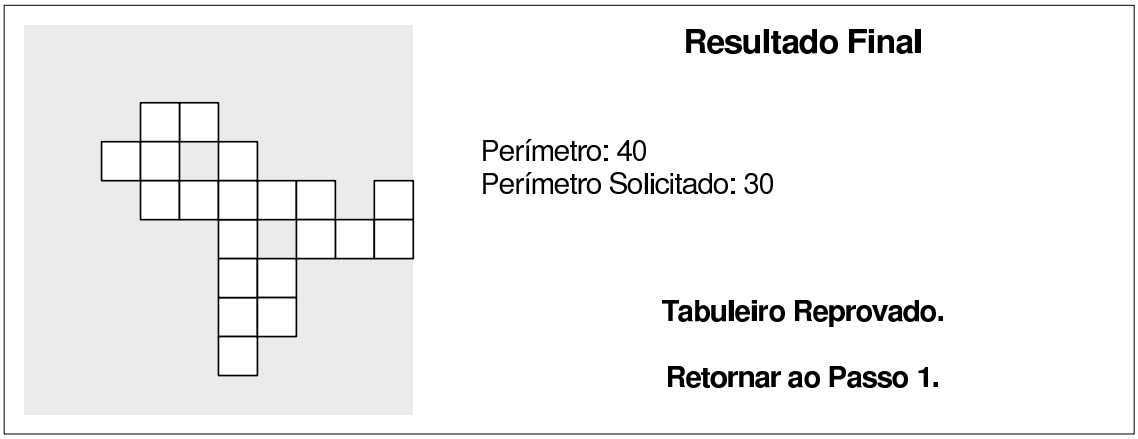


Figura 3.19: Aplicação do teste de qualidade da versão 1.2 do algoritmo

A Figura 3.20 exemplifica tabuleiros construídos pela versão 1.2 do algoritmo e seus respectivos perímetros. Pode-se perceber claramente nas figuras que o nível de coesão do tabuleiro é inversamente proporcional ao perímetro.

Como foi dito anteriormente, este algoritmo descarta tabuleiros que não atendam ao perímetro pedido. Isto claramente melhora os resultados do algoritmo, porém eleva o tempo de retorno, uma vez que o algoritmo terá que repetir a construção até que um tabuleiro com qualidade suficiente seja construído. Esse fato pode ser comprovado na Tabela 3.3, que demonstra o número de tabuleiros descartados por qualidade insuficiente. Esta tabela foi obtida pela construção sucessiva de dez mil tabuleiros para cada intervalo de perímetro. As colunas Menor e Maior representam, respectivamente, o menor e o maior número de tabuleiros reprovados por qualidade insuficiente para obter um tabuleiro aprovado. A coluna Média representa a média de tabuleiros reprovados para cada tabuleiro aprovado. O perímetro 18 foi omitido, pois o algoritmo não encontrou nos testes tabuleiros com esta qualidade mesmo se for solicitada.

	Tabuleiros descartados por qualidade		
Perímetro	Menor	Maior	Média
19-24	2	3531	662,3
25-30	0	149	12,908
31-36	0	7	0,855
37-42	0	0	0

Tabela 3.3: Tabuleiros descartados por intervalo de perímetro

Considerando os testes para a construção de tabuleiros com perímetro menor que trinta (média entre o maior e o menor perímetro possíveis), a média de descartes é de

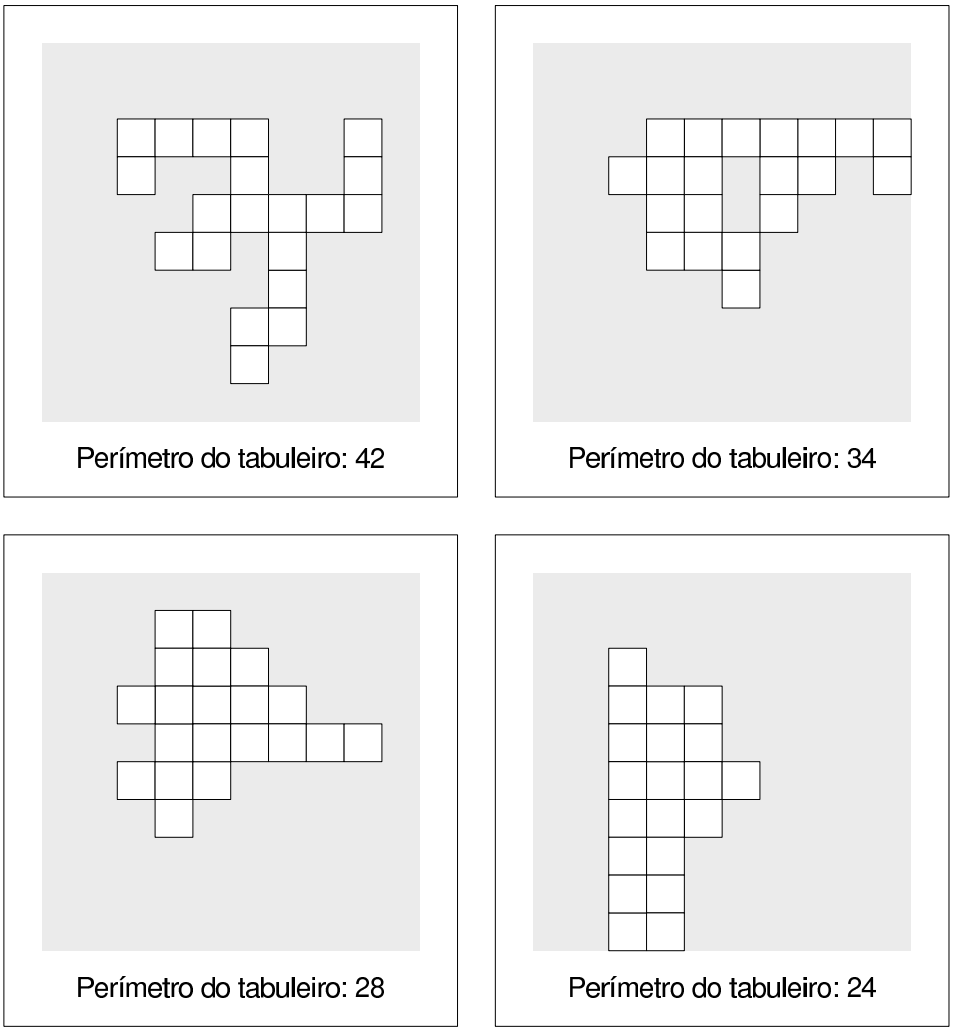


Figura 3.20: Tabuleiros construídos pela versão 1.2 do algoritmo

12,908, isto é, para cada tabuleiro construído outros 12,908 são descartados, ou seja, estes tabuleiros descartados tem perímetro maior que trinta unidades. Devido ao número de descartes ser elevado para o ponto médio dos perímetro, um novo algoritmo foi desenvolvido.

3.5.5. Algoritmo de construção: Versão 2.1

Com base nos resultados obtidos nas Versões 1.1 e 1.2, pode-se agregar mais conhecimento aos algoritmos de construção. As versões anteriores sorteavam a posição na qual os pentominos seriam alocados, isto é, não avaliavam os resultados antes de alocar. Esta nova versão escolhe a posição para a alocação do pentomino de modo a minimizar o perímetro após esta alocação. Este algoritmo, diferentemente do anterior, não efetua testes de qualidade, uma vez que a validade é garantida a cada alocação. Portanto, esta nova versão é uma heurística gulosa, enquanto os algoritmos anteriores eram aleatórios.

A entrada do algoritmo é o número de pentominos $2 \leq n \leq 12$ que formam o aglutinado. A saída é a área de trabalho $vMatriz$, de dimensões $tamMatriz$ contendo um tabuleiro com $5n$ espaços vazios.

Primeiro pentomino

Chamar procedimento $Inicializar(vMatriz, Pent)$, alocar o primeiro pentomino ($Pent[1]$) na posição ($tamMatrizDIV2$), atribuindo 1 a $Pent[1].tocado$.

Demais pentominos

Criar uma lista $PeriLista$ com as posições contíguas ao perímetro do aglutinado existente. Atribuir -1 a pos_x e pos_y e ∞ a $menor$. Cada posição de $PeriLista$ deve ser validada. Caso seja válida, deve-se medir o perímetro após essa aglutinação virtual e eliminar esta posição de $PeriLista$. Se o resultado for menor que $menor$, deve-se armazenar essa posição em pos_x e pos_y e o perímetro resultante em $menor$. Se $PeriLista$ for esgotada sem encontrar um perímetro menor, o pentomino $Pent[ct1]$ passa para o fim da fila, o valor de $Pent[ct1].tocado$ recebe o número de pentominos já alocados e o próximo pentomino da fila é considerado, o que só ocorre se $Pent[novo].tocado$ for diferente do número de pentominos já alocados. Se não houverem pentominos a considerar o tabuleiro deve ser descartado. Se, no final desse processo, $menor$ for menor que ∞ então deve-se aglutinar o pentomino nesta posição, atribuir o número de pentominos já alocados a $Pent[1].tocado$ e retornar ao início do procedimento para os demais pentominos. Em caso contrário, o tabuleiro deve ser descartado.

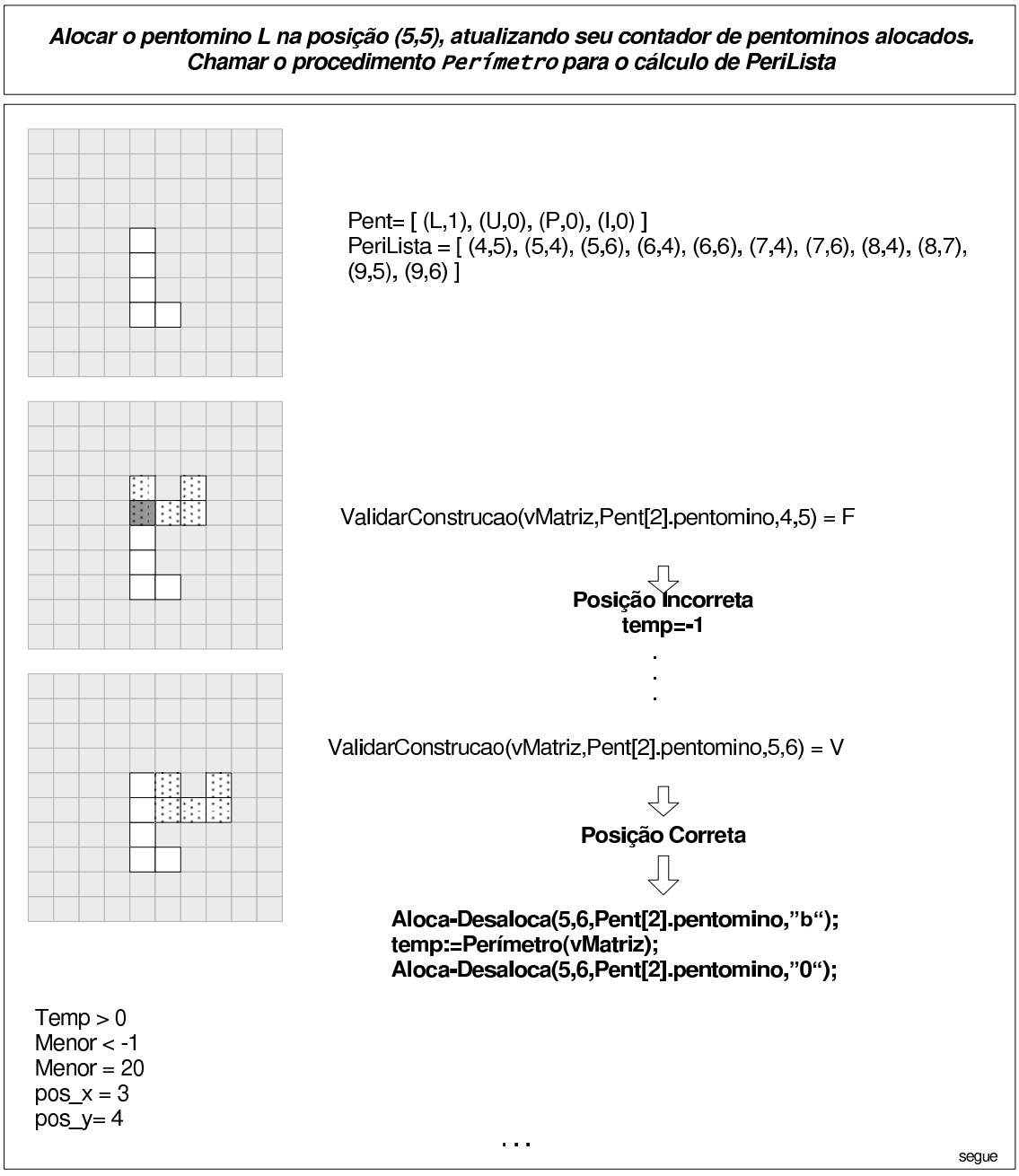
A diferença entre as Versões 1.1 e 2.1 está no processamento de posições nas quais a aglutinação é válida. Na Versão 1.1, quando a aglutinação de um pentomino é válida em uma posição, esta é imediatamente efetuada e o algoritmo, então, segue para os demais pentominos. Já na Versão 2.1, quando uma aglutinação é válida, esta é simulada e o perímetro é aferido. Caso este novo perímetro seja o menor já obtido, esta posição é armazenada. Quando todas as posições forem processadas, o algoritmo, então, aglutina o pentomino na posição armazenada, garantindo assim que o perímetro após esta aglutinação será o menor possível para a ordem de pentominos dada. A partir daí, o algoritmo segue para os demais pentominos. Essa diferença tem dois efeitos, um positivo e um negativo. O efeito positivo é a garantia de que o tabuleiro formado é o melhor possível para a ordem de pentominos dada, uma vez que isto é garantido a cada aglutinação. O efeito negativo é a necessidade de varrer toda a lista de posições contíguas ao aglutinado existente, elevando, assim, a complexidade do algoritmo, uma vez que, em seu melhor caso, a Versão 1.1 testa somente três posições enquanto a Versão 2.1 testa todas as posições de $PeriLista$. Contudo, o efeito positivo compensa o negativo.

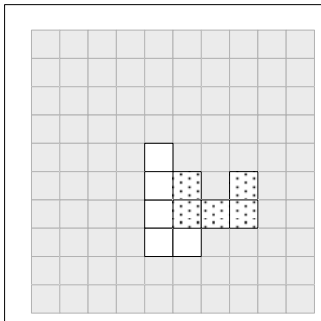
Algoritmo 13 Função $Gerar\_Tabuleiro\_V2.1(n) : vMatriz$

```
ct1 := 1;
descartar = Falso;
repita
    se ct1 = 1 OU descartar = Verdadeiro então
        ct1 := 1;
        pm := tamMatriz DIV 2;
        Inicializar(vMatriz, Pent)
        Aloca − Desaloca(pm, pm, Pent[1].pentomino, vMatriz, "b");
        Pent[1].tocado := 1;
    senão
        menor := ∞; ct2 := 1;
        Perimetro(vMatriz, PeriLista);
        repita
            tmp_x := PeriLista[ct2].x;
            tmp_y := PeriLista[ct2].y;
            PosValida := ValidarConstrucao(vMatriz, pos_x, pos_y,
Pent[ct1].pentomino);
            se PosValida = Verdadeiro então
                Aloca−Desaloca(tmp_x, tmp_y, Pent[ct1].pentomino, vMatriz, "b");
                temp := Perimetro(vMatriz);
                Aloca−Desaloca(tmp_x, tmp_y, Pent[ct1].pentomino, vMatriz, "0");
            senão
                temp := −1;
            se temp > 0 E temp ≤ menor então
                menor := temp; pos_x := tmp_x; pos_y := tmp_y;
            ct2 := ct2 + 1;
            se ct2 > |PeriLista| E menor = ∞ então
                Pent[ct1].tocado := ct1;
                remover(Pent, ct1, pentoTemp);
                inserir(Pent, n, pentoTemp);
                se Pent[ct1].tocado ≥ ct1 então
                    descartar := Verdadeiro;
                senão
                    ct2 := 1;
        Até que ct2 > |PeriLista| OU descartar = Verdadeiro;
        se descartar = Falso então
            Pent[ct1].tocado := ct1;
            Aloca − Desaloca(pos_x, pos_y, Pent[ct1].pentomino, vMatriz, "b");
            Perimetro(vMatriz, PeriLista);
    ct1 := ct1 + 1;
Até que ct1 > n E descartar = Falso
```

Resultados e conclusões:

A Figura 3.21 ilustra a execução da versão 2.1 do algoritmo de construção de tabuleiros.





Pent= [(L,1), (U,0), (P,0), (I,0)]
Menor = 20
pos_x = 3
pos_y= 4

ValidarConstrucao(vMatriz,Pent[2].pentomino.vPosicoes,6,6) = V

⇩

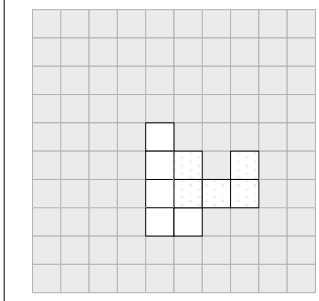
Posição Correta

⇩

Aloca-Desaloca(6,6,Pent[2].pentomino,"b");
temp := Perimetro(vMatriz);
Aloca-Desaloca(6,6,Pent[2].pentomino,"0");

Temp = 18
Temp < Menor
Menor = 18
pos_x = 6
pos_y= 6
descartar = falso

. . . O algoritmo segue para as demais posições.



Menor = 18
pos_x = 6
pos_y= 6

Aloca-Desaloca(6,6,Pent[2].pentomino,"b");
Pent[2].tocado := 2;

O algoritmo segue para os demais pentominos.

Figura 3.21: Exemplo do funcionamento da versão 2.1 do algoritmo

A Tabela 3.4 mostra a avaliação da versão 2.1 do algoritmo. Os dados da tabela foram obtidos através da construção sucessiva de dez mil tabuleiros. Pode-se perceber que a maioria dos perímetros dos tabuleiros construídos por esta nova versão se encontram entre vinte e cinco e trinta e seis enquanto os construídos pela versão 1.1 se situam entre trinta e um e quarenta e dois, o que é uma melhora significativa.

Intervalos de perímetro	Número de tabuleiros	% de tabuleiros
18-18	0	0%
19-24	642	6,42%
25-30	4647	46,47%
31-36	4209	42,09%
37-42	502	5,02%

Tabela 3.4: Resultados da versão 2.1 do algoritmo

A média dos perímetros dos tabuleiros é $30,72$.

Embora a melhora seja bastante considerável em relação à versão 1.1, pode-se aprimorará-la ainda mais.

3.5.6. Algoritmo de construção: Versão 2.2

Este novo aperfeiçoamento é baseado em uma característica do procedimento de alocação. Uma alocação é feita através de uma soma de deslocamentos, isto é, baseada em uma posição (x, y). A alocação se dá somando as posições armazenadas no vetor $vPosicoes$ à posição inicial, conforme foi visto na Seção 3.3. Por esta razão, as novas posições serão sempre maiores ou iguais aos da posição inicial. Esta questão implica que não sejam possíveis alocações feitas a partir de posições situadas acima e à esquerda do aglutinado. A Figura 3.22 ilustra um caso onde esta característica se torna evidente. No exemplo, inicialmente se alocou o pentomino P na posição $(3, 3)$ e depois se tentou aglutinar o pentomino X nas posições contíguas ao perímetro do pentomino P. Percebe-se então que a aglutinação somente ocorre à direita (posição $(3, 6)$) e abaixo (posição $(5, 4)$) do pentomino P. Quando se tenta aglutinar o pentomino X acima (posição $(2, 3)$) e a esquerda (posição $(3, 2)$) do pentomino P, ocorrem sobreposições, tornando esta aglutinação inválida.

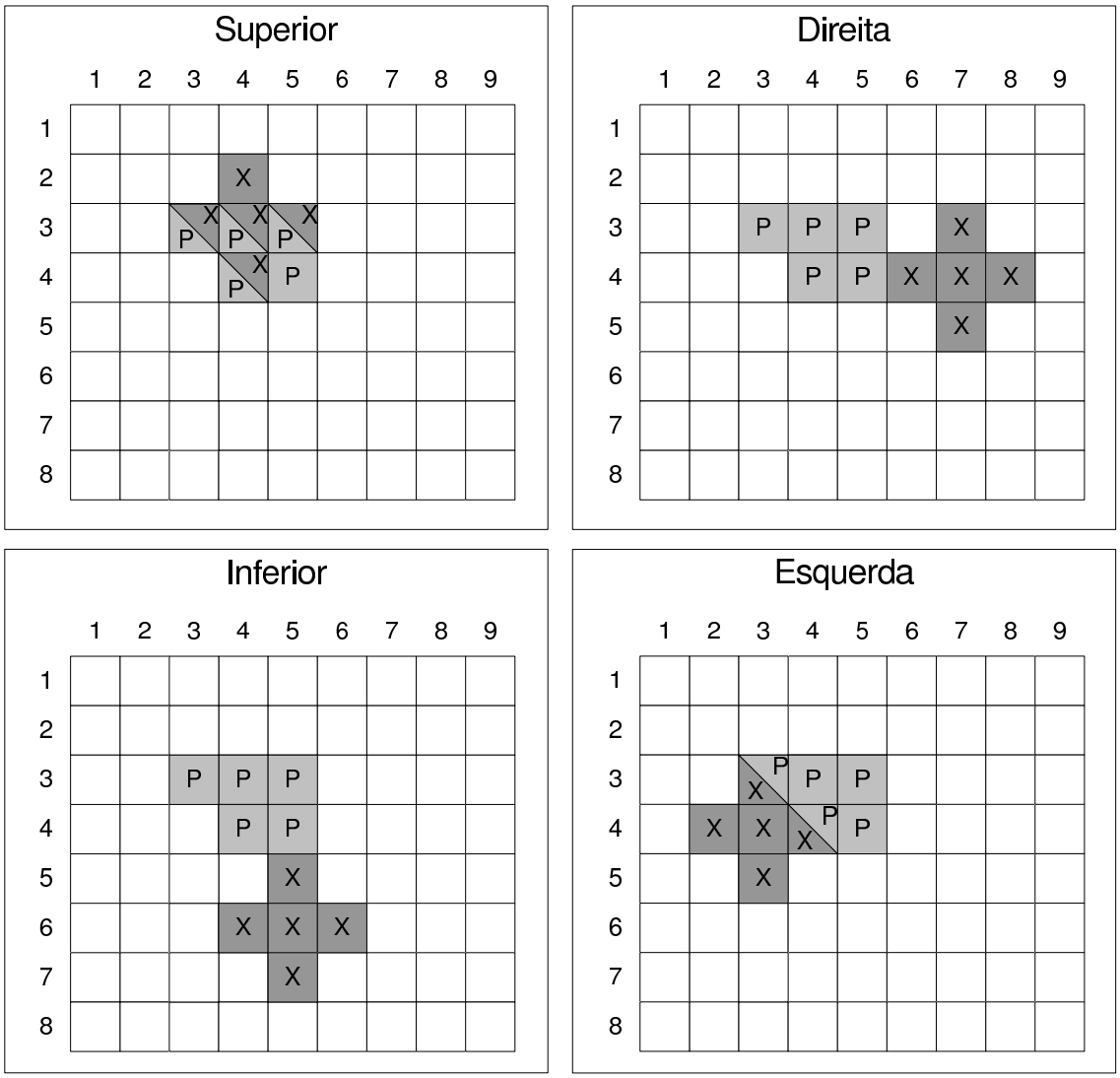


Figura 3.22: Exemplo da falha na alocação

Para contornar esta característica uma nova regra foi criada. Esta nova regra consiste em criar mais três pontos de alocação para cada ponto inicial, isto é, para cada ponto (x, y) pertencente à $PeriLista$ são adicionados novos três pontos, que permitirão ao algoritmo de construção efetuar alocações em qualquer lado do aglutinado existente. Estes novos pontos são obtidos utilizando as dimensões $MaxX$ e $MaxY$ do pentomino a ser alocado, que respectivamente significam o maior valor de X e de Y pertencentes ao vetor $vPosicoes$, que representa o esquema atual do pentomino. A Figura 3.23 apresenta estes novos pontos.

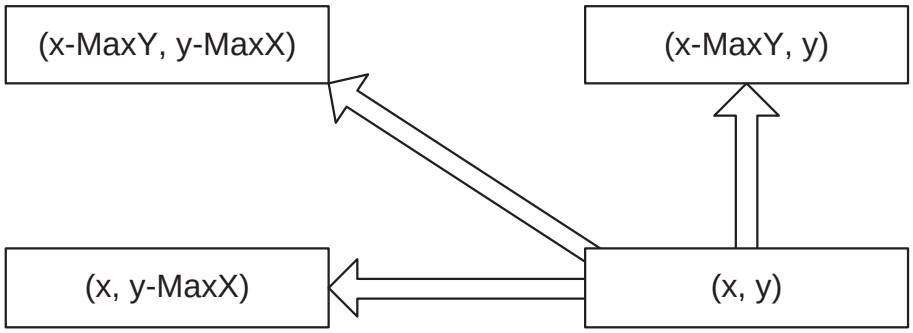


Figura 3.23: Novos pontos da alocação

A Figura 3.24 ilustra o cálculo de novos pontos para um caso onde se deseja obter todas as posições nas quais o pentomino T deve ter sua aglutinação testada.

Uma vez que para cada posição de $PeriLista$ são criadas outras três, seu tamanho aumenta em três vezes. Considerando que o algoritmo testa cada posição da lista, a remoção de posições repetidas também é necessária neste método. Muitos desses novos valores já pertenciam à $PeriLista$ anteriormente. Para remover as posições repetidas em tempo linear, deve-se previamente aplicar um método de ordenação. Utiliza-se a ordenação por caixas, uma vez que sua complexidade é linear e todos os valores pertencem a um intervalo conhecido, que varia de 1 até $tamMatriz$.

Passo 1 Para cada posição (x, y) de $PeriLista$ incluir os valores $(x, y - MaxX)$, $(x - MaxY, y - MaxX)$ e $(x - MaxY, y)$ no final de $PeriLista$, caso estes fiquem dentro da área de trabalho;

Passo 2 Aplicar a ordenação por caixas à $PeriLista$.

Passo 3 Remover as posições repetidas.

O Passo 1 tem complexidade $O(n)$, onde n é o número de pentominos já alocados. Já o Passo 2, tem complexidade $O(tamMatriz)$, uma vez que esse é o número de caixas. O Passo 3 tem complexidade $O(n)$, uma vez que varre toda a lista criada pelo Passo 1. Sendo assim, a complexidade é de $O(tamMatriz + n)$.

Os passos executados por esta versão são idênticos aos da versão 2.1, exceto pela atualização de $PeriLista$ de acordo com as novas regras criadas.

A Tabela 3.5 apresenta os resultados da versão 2.2 do algoritmo. Os dados da tabela foram obtidos através da construção sucessiva de dez mil tabuleiros.

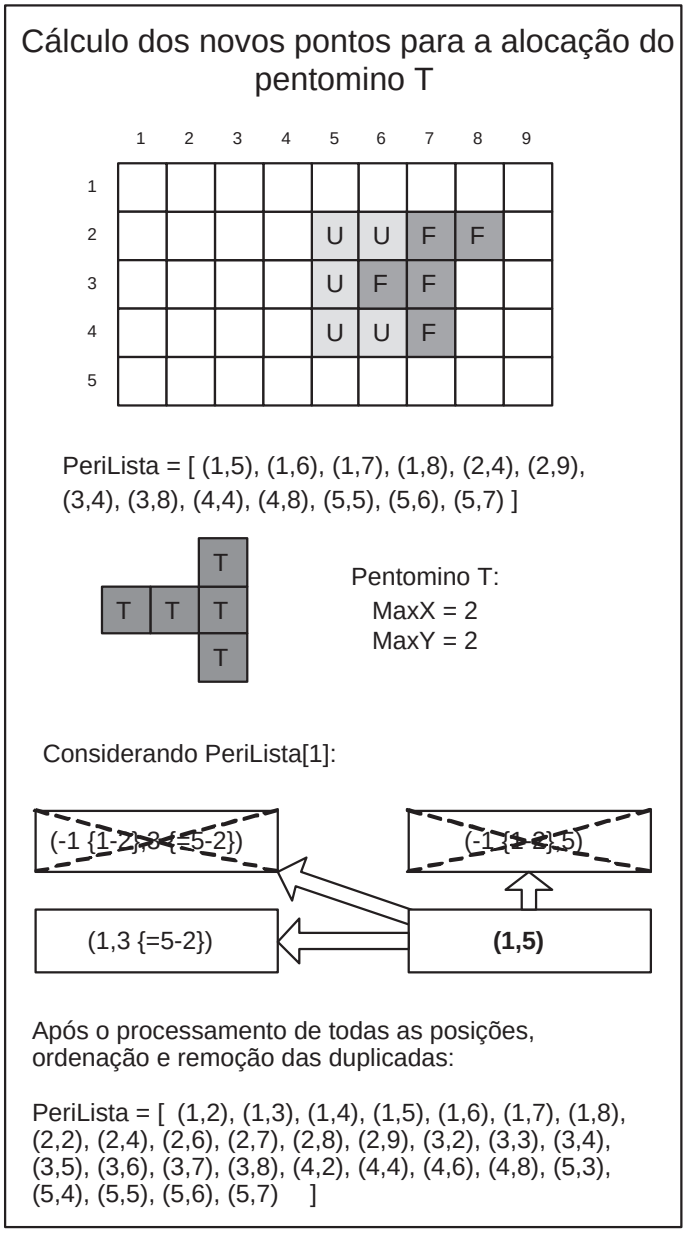


Figura 3.24: Exemplo do cálculo de novos pontos para versão 2.2 do algoritmo

Intervalos de perímetros	Número de tabuleiros	% de tabuleiros
18-18	11	0,11%
19-24	2853	28,53%
25-30	6694	66,94%
31-36	442	4,42%
37-42	0	0%

Tabela 3.5: Resultados da versão 2.2 do algoritmo

A média dos perímetros dos tabuleiros é de $26,51$.

Percebe-se pelos resultados que a modificação efetuada melhorou os resultados. Na versão 2.1, a maioria dos perímetros fica entre vinte e cinco e trinta e uma unidades. Já na versão 2.2, a maioria fica entre dezenove e trinta unidades, o que é uma melhora significativa considerando o aumento de processamento. Deve-se ressaltar que a versão 2.1 gerou quinhentos e dois tabuleiros com perímetro entre trinta e sete e quarenta e dois, que são os piores tabuleiros que podem ser construídos. Já a versão 2.2 não gerou nenhum tabuleiro com essa qualidade. Comparando o número de tabuleiros com a melhor qualidade percebe-se que a versão 2.2 gerou onze tabuleiros, enquanto a versão 2.1 não gerou nenhum.

Sendo assim, pode-se dizer que a melhora dos resultados foi suficiente para compensar o aumento no tempo gasto para a construção.

3.5.7. Comparações entre os algoritmos

Pode-se observar uma evolução dos resultados das quatro versões, uma vez que a cada modificação implementada os tabuleiros se mostram mais compactos. A Tabela 3.6 apresenta os resultados das versões 1.1, 2.1 e 2.2 dos algoritmos. A versão 1.2 foi omitida, pois o teste de qualidade inviabiliza esta comparação, uma vez que todos os resultados são filtrados pelo teste de qualidade.

	18-18		19-24		25-30		31-36		37-42	
Versão	N^o	%	N^o	%	N^o	%	N^o	%	N^o	%
1.1	0	0	40	0,4	1047	10,47	5094	50,94	3819	38,19
2.1	0	0	642	6,42	4647	46,47	4209	42,09	502	5,02
2.2	11	0,11	2853	28,53	6694	66,94	442	4,42	0	0

Tabela 3.6: Tabela comparativa entre as versões do algoritmo de criação de tabuleiros de formato livre

4 O JOGO DOS 20: UM PROTÓTIPO

Neste capítulo, é apresentada a implementação de um jogo para computador, baseado no problema dos 20. Neste jogo, um tabuleiro com área equivalente a quatro pentominos é exposto ao jogador. Dado este tabuleiro, o usuário deve cobri-lo três vezes utilizando quatro pentominos distintos em cada cobertura. Cada vez que um tabuleiro for totalmente coberto, os quatro pentominos utilizados serão descartados do conjunto de pentominos disponíveis para a próxima cobertura. Se uma determinada cobertura for impossível utilizando os pentominos disponíveis, o jogador pode retornar a um tabuleiro já coberto, esvaziá-lo e refazer esta cobertura utilizando outros pentominos. Deve-se acrescentar que os tabuleiros fornecidos permitem três coberturas (jogo ideal).

No final da terceira cobertura, o jogo salvará o tempo que o jogador levou para cobrir os tabuleiros e irá classificá-lo em um *ranking*. Serão salvos neste *ranking* o nome do jogador, o tempo que este levou e as três coberturas efetuadas.

Após o término das três coberturas ou a comando do jogador, o jogo descartará o tabuleiro corrente e construirá outro, liberando os doze pentominos para novas coberturas.

O jogador poderá solicitar auxílio do computador para cobrir o tabuleiro corrente com os pentominos disponíveis ou solicitar que o computador demonstre as três coberturas.

O jogo dos 20 também pode ser jogado em rede. Nesta modalidade, um jogador atuará como servidor e os outros como clientes. O servidor criará um tabuleiro e o propagará para os clientes. Após a confirmação de recebimento dos clientes, o servidor autorizará o início do jogo. O vencedor será o jogador que cobrir os três tabuleiros primeiro, utilizando as regras descritas anteriormente. O vencedor, seu tempo e suas coberturas serão inseridos no *ranking* de cada jogador. Caso *todos* os jogadores solicitem um novo tabuleiro, o processo é reiniciado, isto é, um novo tabuleiro será construído no servidor e propagado para os clientes.

4.1. Implementação de um protótipo

Para ilustrar a utilização dos procedimentos apresentados no Capítulo 3, desenvolveu-se um protótipo do Jogo dos 20. Este protótipo ainda não conta com todos os recursos do Jogo dos 20 ideal, proposto na seção anterior. No estágio atual é construído um tabuleiro utilizando a versão 2.2 do algoritmo de construção de tabuleiros exposto no Capítulo 3. Em seguida, este tabuleiro é apresentado ao jogador que deve preenchê-lo utilizando quatro pentominos. Para cada alocação que o jogador efetua, o algoritmo de alocação é executado. A critério do usuário podem ser aplicadas transformações aos pentominos, executando sobre um determinado pentomino o algoritmo correspondente à transformação solicitada. Após o preenchimento, é exibido para o jogador o tempo que este levou para cobrir o tabuleiro e é perguntado se o usuário deseja um novo tabuleiro. Em caso afirmativo, o tabuleiro reverte os pentominos a seus respectivos esquemas iniciais, constrói um novo tabuleiro e o fornece ao usuário, reiniciando o processo. A Seção 4.2 apresenta os recursos que, posteriormente, serão anexados ao protótipo.

Detalhamento da interface do protótipo

A Figura 4.1 ilustra a tela inicial do protótipo. Pode-se observar uma barra de comando no lado direito da figura. Nesta barra estão situados os botões de controle, que são, respectivamente, de cima para baixo:

Limpar Limpa o tabuleiro, removendo todos os pentominos.
Novo Jogo Constrói um novo tabuleiro para ser preenchido.
Rotacionar Rotaciona o pentomino em foco.
Espelhamento Horizontal Aplica um espelhamento horizontal no pentomino em foco.
Espelhamento Vertical Aplica um espelhamento vertical no pentomino em foco.
Resolver Cobre automaticamente o tabuleiro.
Ranking Exibe o ranking dos jogadores **(não implementado)**.
Rede Inicia o jogo em rede **(não implementado)**.

A Figura 4.2 ilustra a movimentação do pentomino X. No jogo, para mover os pentominos deve-se clicar na imagem do pentomino e o arrastar para a posição de destino.

Na Figura 4.3, o pentomino X foi solto na posição final. Após soltar o pentomino em uma determinada posição, este é alocado nesta posição, se a alocação for válida. A remoção é efetuada de maneira semelhante. O usuário deve clicar no pentomino alocado e o arrastar para fora do tabuleiro.

A Figura 4.4 ilustra a aplicação de um espelhamento vertical no pentomino F. Para aplicar os procedimentos de transformação basta clicar no pentomino, colocando-o em foco, e clicar nos botões desejados ou utilizar os atalhos de teclado (*Crl+R* para rotação, *Crl+H* e *Crl+V* para espelhamento horizontal e vertical, respectivamente). A aplicação de espelhamentos e rotações também pode ser efetuada através de um menu

Pop-up aberto quando se clica com o botão direito do *mouse* sobre a imagem de um pentomino, conforme ilustrado na Figura 4.5.

Na Figura 4.6, pode-se observar um tabuleiro totalmente coberto. Ao final da cobertura o jogo emite uma mensagem informando que o tabuleiro foi preenchido, o tempo levado desde a construção do tabuleiro até a finalização de sua cobertura, conforme a Figura 4.7 ilustra. Além disso, o jogo questiona se o usuário deseja iniciar um novo jogo. Em caso negativo, o jogo mantém o tabuleiro coberto e aguarda instruções do usuário, conforme pode ser observado na Figura 4.8. Em caso positivo, o jogo cria um novo tabuleiro e o fornece ao jogador, conforme pode ser visto na Figura 4.9.

O recurso da cobertura automática de tabuleiros (botão resolver) foi implementado para cobrir apenas um tabuleiro. A versão final desta função deve cobrir os três tabuleiros do jogo ideal. Quando o usuário solicita a cobertura do tabuleiro, uma das coberturas possíveis é demonstrada de maneira animada, isto é, o usuário pode visualizar as transformações dos pentominos, sua movimentação e sua sucessiva alocação. A Figura 4.10 ilustra um passo da animação que resulta em uma cobertura do tabuleiro.

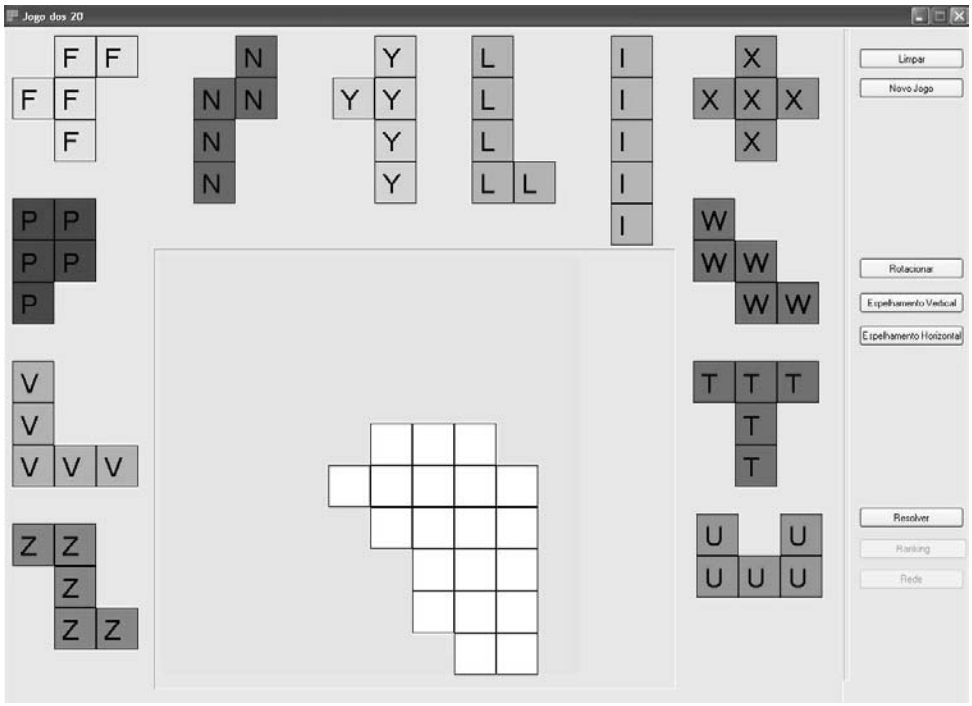


Figura 4.1: Protótipo: Tabuleiro em branco

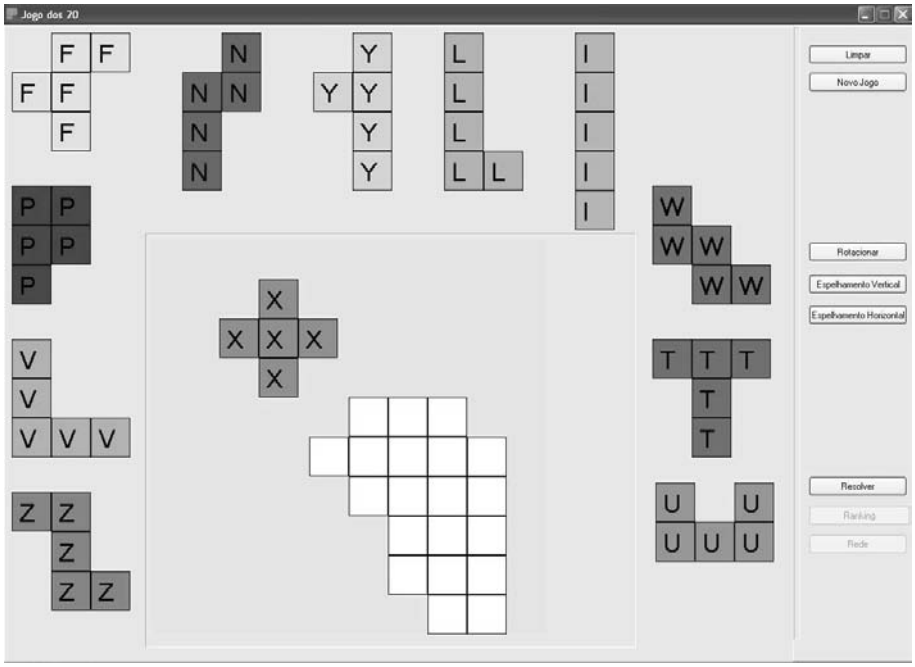


Figura 4.2: Protótipo: Movendo o pentomino X

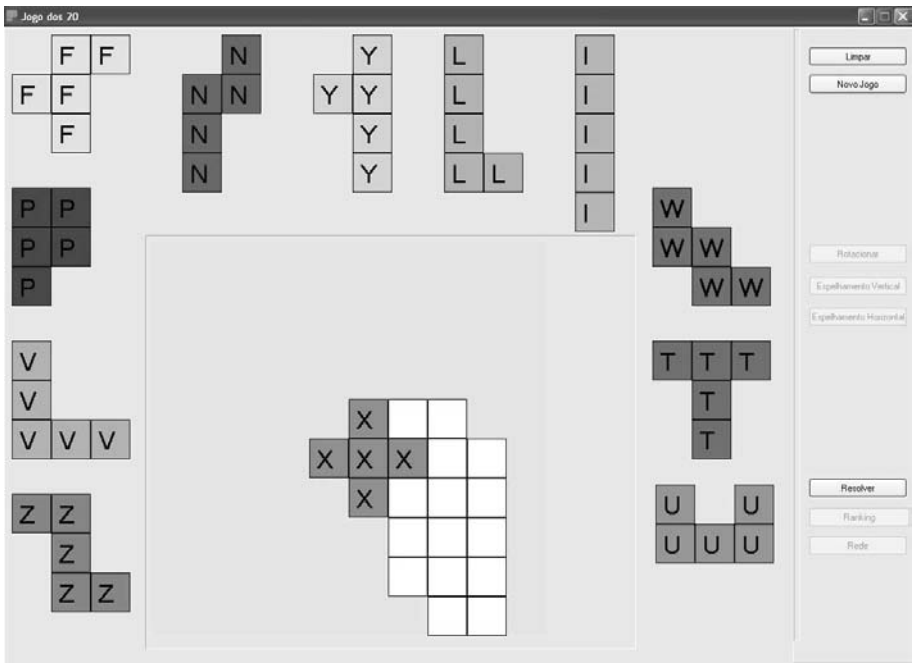


Figura 4.3: Protótipo: Pentomino X já alocado

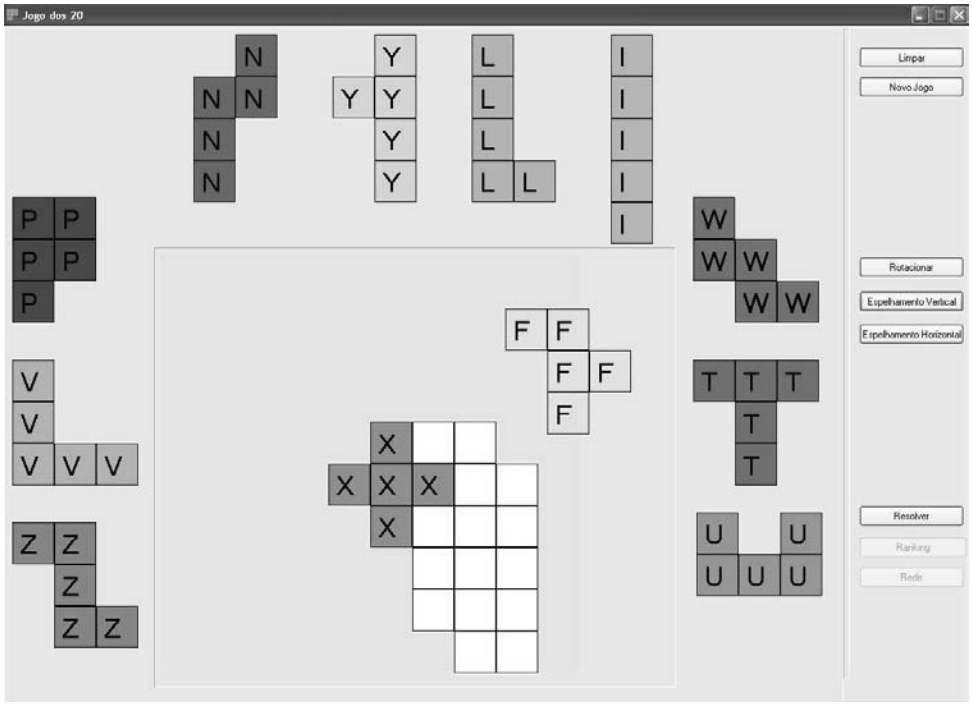


Figura 4.4: Protótipo: Preparando o pentomino F para alocação

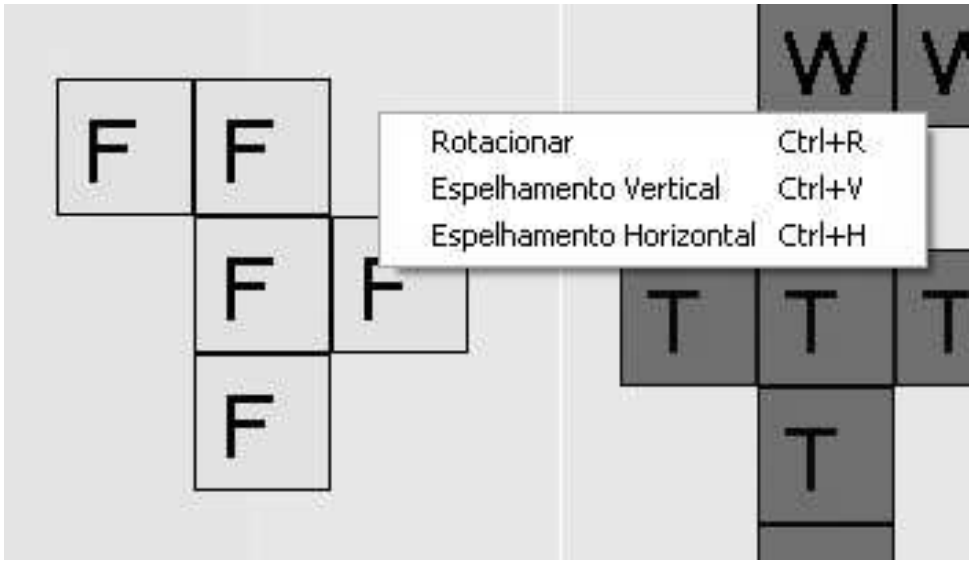


Figura 4.5: Protótipo: Menu *pop-up* de transformações

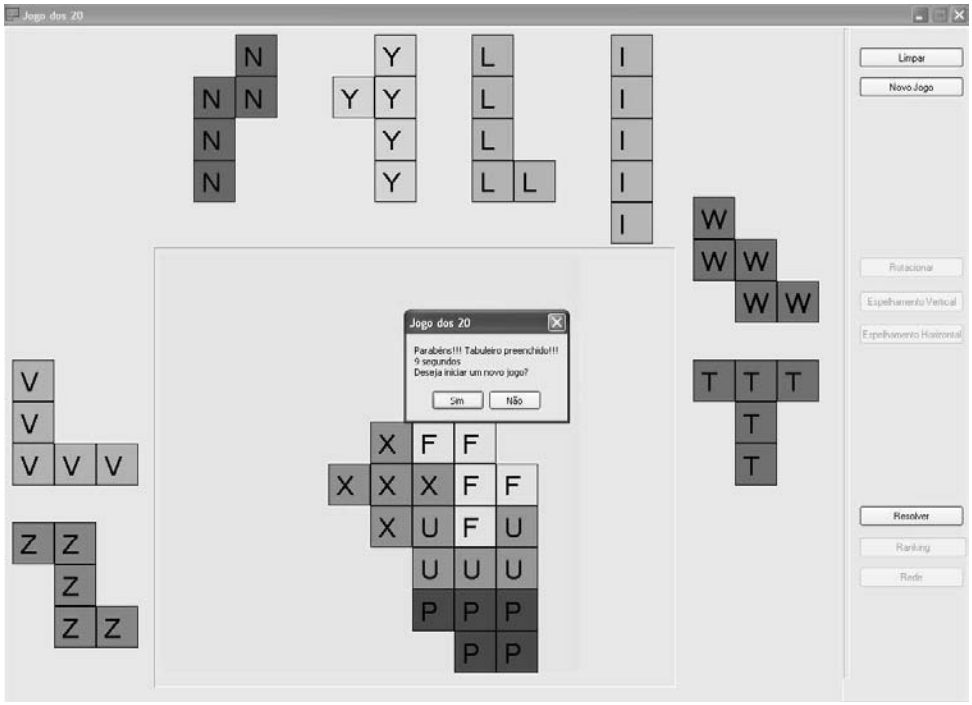


Figura 4.6: Protótipo: Tabuleiro totalmente coberto

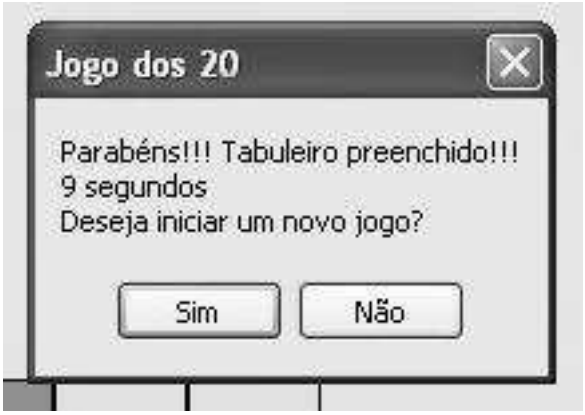


Figura 4.7: Protótipo: Mensagem ao final da cobertura

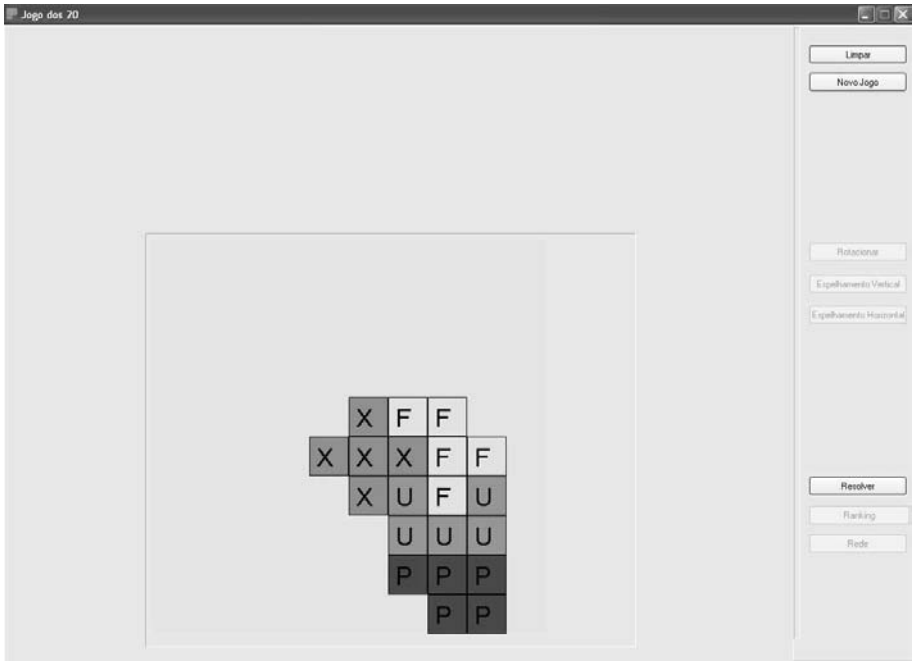


Figura 4.8: Protótipo: Interface ao final da cobertura

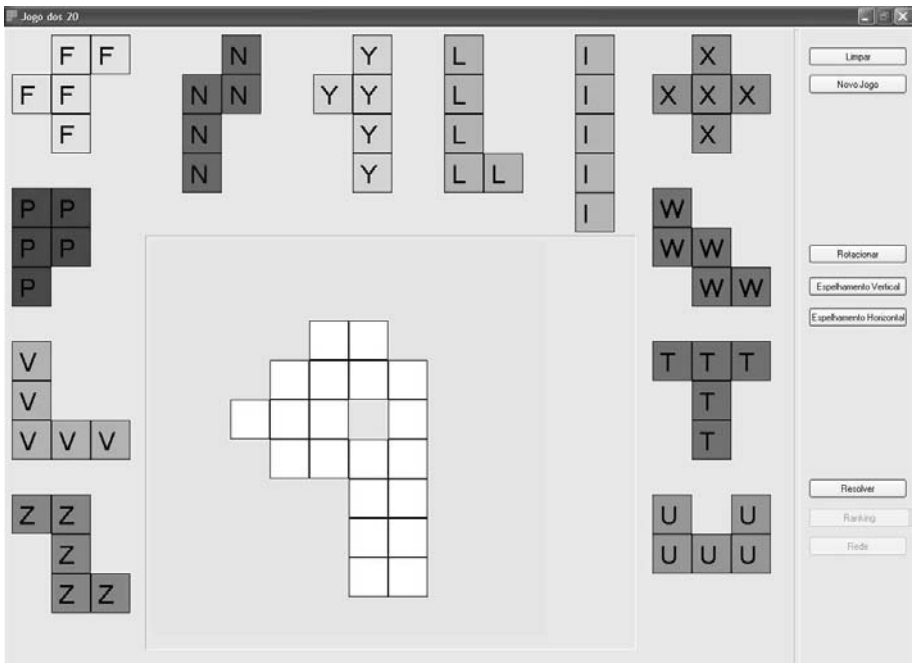


Figura 4.9: Protótipo: Outro tabuleiro gerado

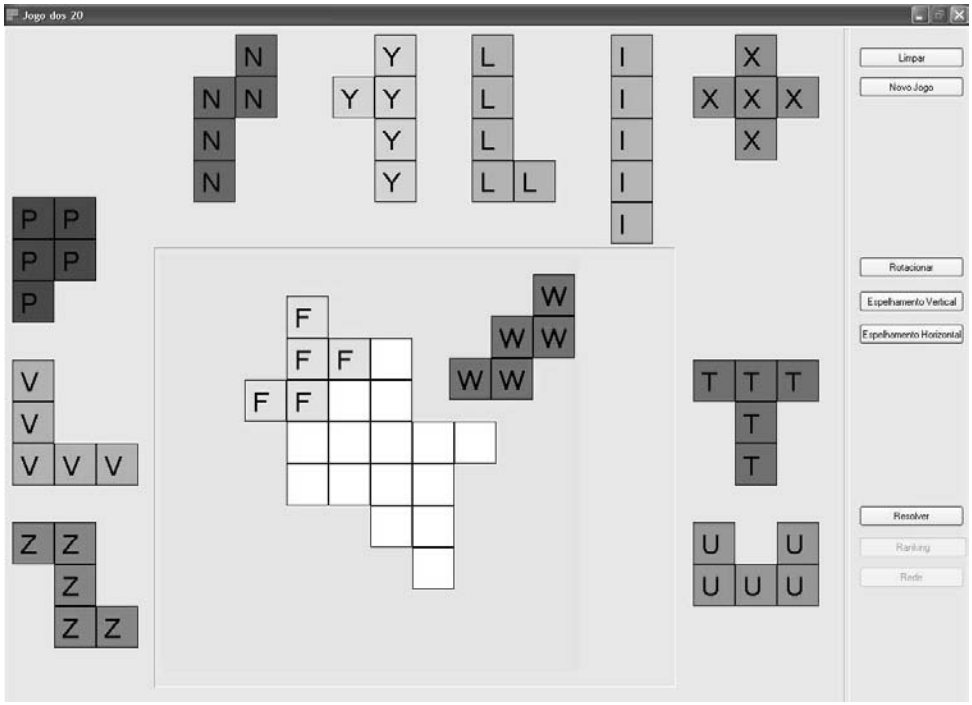


Figura 4.10: Protótipo: Cobertura automática de tabuleiros

Detalhamento da implementação do protótipo

A linguagem utilizada na implementação do protótipo é o Delphi. Os procedimentos descritos no Capítulo 3 são a base para a implementação do jogo.

Foram implementadas duas classes. A primeira, denominada *TPentomino*, contém os métodos de manipulação dos pentominos (rotação e espelhamentos) e a representação computacional dos pentominos, apresentados no Capítulo 3. Cada objeto desta classe representa um pentomino distinto.

A outra classe é denominada *TTabuleiro* e contém os métodos de manipulação de tabuleiros. Cada objeto da mesma armazena um tabuleiro e doze pentominos, um de cada tipo. Nela estão os algoritmos de alocação, validação, remoção e as quatro versões do algoritmo de construção de tabuleiros, além dos algoritmos neste utilizados. Todos estes algoritmos foram descritos no Capítulo 3.

A Figura 4.11 apresenta as duas classes e seus respectivos métodos e atributos.

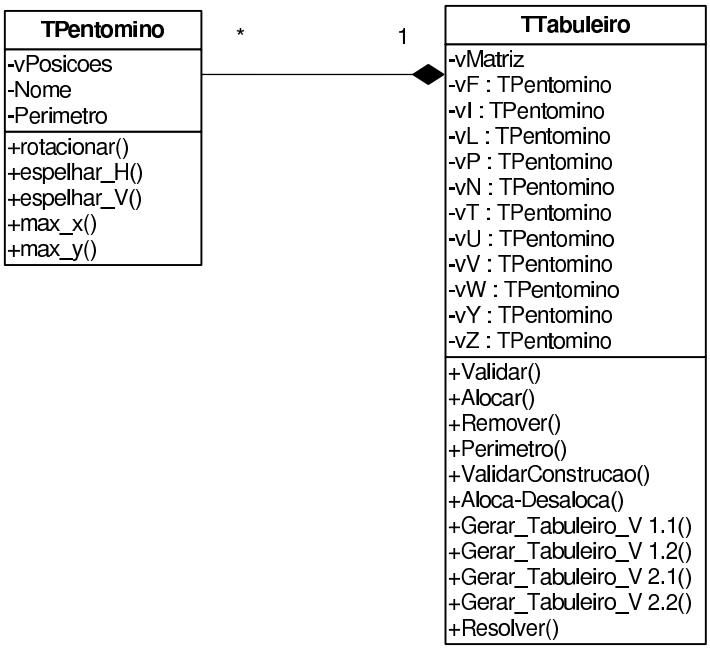


Figura 4.11: Diagrama de Classes do Jogo dos 20

4.2. Recursos ainda não implementados

Alguns recursos presentes no jogo ideal ainda não foram implementados. É importante ressaltar que estes recursos constituem aprimoramentos do protótipo já existente. O primeiro trata das múltiplas coberturas, isto é, o jogo deve oferecer ao usuário três tabuleiros idênticos para que este os cubra, conforme a definição do Problema dos 20. Atualmente o jogo oferece apenas um único tabuleiro. Para permitir o perfeito funcionamento desta funcionalidade deve-se garantir que todos os tabuleiros expostos ao jogador tenham três coberturas. Como o estágio atual do protótipo oferece apenas um único tabuleiro formado pela aglutinação de quatro pentominos, esta verificação foi dispensada, uma vez que é garantido que o tabuleiro tem pelo menos uma cobertura, que é constituída pelos pentominos que formaram o aglutinado.

Outro recurso que deve ser implementado é o jogo em rede, pois assim pode-se oferecer ao usuário a possibilidade de jogar acompanhado, estabelecendo uma competição na qual ganha o jogador que terminar as três coberturas mais rápido. Para armazenar os resultados dos jogadores, deve-se implementar um *ranking*, que permitirá ao usuário visualizar as coberturas anteriores e seus respectivos tempos.

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação teve como foco um membro em particular da família dos poliminos: os pentominos. Inicialmente, no Capítulo 1, além da apresentação dos pentominos, sua história foi abordada e introduzidos os conceitos de problemas de alocação e de transformações de pentominos. Além disso, os pentocubos foram apresentados.

No Capítulo 2, foram apresentadas características peculiares aos problemas de Alocação e, em seguida, definidos.Também foram expostos alguns jogos baseados nestes problemas, como, por exemplo, o Jogo dos 20. Os recursos na rede que envolvem os pentominos foram devidamente exibidos e comentados.

A seguir, no Capítulo 3, foram expostos procedimentos que permitem implementar os problemas de alocação descritos. Para tanto, novas estruturas de dados foram criadas para armazenar os pentominos e os tabuleiros. Feito isso, houve a explanação de procedimentos capazes de aplicar transformações aos pentominos, isto é, executar rotações e espelhamentos. Dando seguimento, houve a descrição de algoritmos para alocar e remover pentominos. Finalmente, foram propostos procedimentos para a criação de tabuleiros exatos e de formato livre. Quatro versões distintas para a construção destes foram descritas. A cada nova versão, implementou-se uma modificação para melhorar os resultados obtidos pela anterior. Além disso, efetuou-se uma análise comparativa entre as diferentes versões do algoritmo.

No Capítulo 4, o jogo ideal baseado no Problema dos 20 foi descrito e seu protótipo apresentado. Algumas funcionalidades do jogo ideal ainda não foram implementadas, como, por exemplo, a possibilidade de múltiplas coberturas. Este protótipo tem como principal objetivo ilustrar a aplicação dos algoritmos expostos anteriormente.

Este trabalho tem como contribuição uma biblioteca de algoritmos que permite lidar com os pentominos e seus problemas de alocação computacionalmente. O foco do mesmo são os pentominos e, por isso, a biblioteca de algoritmos foi desenvolvida apenas para estes. Um interessante trabalho futuro é sua extensão para toda a família dos poliminos.

A finalização do protótipo do Jogo dos 20 e a averiguação dos reais benefícios de sua aplicação educacional também são propostas a serem desenvolvidas.

REFERÊNCIAS

[1] GOLOMB, Solomon W., **"Checker Boards and Polyominoes"**, The American Mathematical Monthly, Vol. 61, No. 10, págs. 675-682, 1954.

[2] GARDNER, Martin, **"Mathematical Games: More About Complex Dominoes, Plus The Answers To Last Month Puzzles"**, Scientific American 197, págs. 126 - 140, 1957.

[3] SCOTT, Dana S., **"Programming A Combinatorial Puzzle"**, Technical Report No. 1, New Jersey: Princeton University Department of Electrical Engineering, 1958.

[4] FLETCHER, John G., **"A Program To Solve The Pentomino Problem By The Recursive Use Of Macros"**, Communications of the ACM 8, págs. 621 - 623, 1965.

[5] BITNER, James R.; REINGOLD, Edward M., **"Backtrack Programming Techniques"**, Communications of the ACM 11, págs. 651 - 656, 1975.

[6] TRACY, Dyanne M.; ECKART, Joyce A., **"Five Good Reasons to Use Pentominoes"**, School Science and Mathematics 90, págs. 665 - 673, 1990.

[7] ONSLOW, Barry, **"Pentominoes Revisited"**, The Arithmetic Teacher 37, págs. 5-9, 1990.

[8] GOLOMB, Solomon W., **"Polyominoes: Puzzles, Patterns, Problems and Packings"**, Princeton University Press, 1994.

[9] ORMAN, Hilarie K., **"Pentominoes: A First Player Win"**, Games of No Chance 29, págs. 339 - 344, 1996.

[10] ANSTREICHER, Kurt M., **"A Pentomino Exclusion Problem"**, University of Iowa, 1999. Disponível em: http://www.biz.uiowa.edu/faculty/anstreicher/pento.ps. Acesso em: 07 out. de 2008.

[11] KNUTH, Donald E., **"Dancing Links"**, Stanford University, 2000. Disponível em: http://www-cs-faculty.stanford.edu/ knuth/papers/dancing-color.ps.gz. Acesso em: 07 out. de 2008.

[12] KOTH, Maria; GROSSER, Notburga, **"Stufenparallelogramme aus zwölf Pentominos"**, Universität Wien Nordbergstr, 2004. Traduzido por Oswaldo Vernet.

[13] LARSEN, Mogens E., **"Pentomino Battleships"**, Kobenhavns Universitet, 2007. Disponível em: http://www.math.ku.dk/ mel/battle.pdf. Acesso em: 07 out. de 2008.

[14] GRAVIER, Sylvain; MONCEL, Julien; PAYAN, Charles, **"A Generalization of the Pentomino Exclusion Problem: Dislocation of Graphs"**, Discrete Mathematics 307, págs. 435-444, 2007.