SUDOKU - Open Source (GameMaker: Studio)

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Enigma do Sudoku Resolvido

O projeto faz parte de um trabalho acadêmico, da matéria de Inteligência Artificial, ministrada pelo Prof. Dr. Cleber Mira na Universidade Estadual do Mato Grosso do Sul, onde curso o 4° ano de Sistemas de Informação.

O objetivo é criar um "Agente jogador de Sudoku" utilizando algoritmos de busca e heurísticas. No meu caso escolhi utilizar buscar a solução por força bruta, utilizando backtracking quando ocorre falhas de posicionamento.

Segundo a Wikipedia:

> *Sudoku, por vezes escrito Su Doku (□□,sūdoku) é um quebra-cabeça baseado na colocação lógica de números. O objetivo do jogo é a colocação de números de 1 a 9 em cada uma das células vazias numa grade de 9×9, constituída por 3×3 subgrades chamadas regiões. O quebra-cabeça contém algumas pistas iniciais, que são números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos*

números em células que estejam vazias. Cada coluna, linha e região só pode ter um número de cada um dos 1 a 9. Resolver o problema requer apenas raciocínio lógico e algum tempo. Os problemas são normalmente classificados em relação à sua realização. O aspecto do sudoku lembra outros quebra-cabeças de jornal. Foi criado por Howard Garns, um projetista e arquiteto de 74 anos aposentado.

DESCRIÇÃO DO AGENTE JOGADOR DE *SUDOKU*

Ambiente:

- Grade 9 x 9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Ações:

- Inserir número (1 à 9);
- Avançar (Esquerda para direita, de cima para baixo);
- Retroceder (Direita para esquerda, de baixo para cima).

Percepções:

- Verificar se a célula está vazia;
- Verificar se a célula possui número, mas é uma posição anterior que já foram testados todos os números.

PSEUDO-CÓDIGO DO RESOLVEDOR:

```
Função resolvedor_sudoku(grade):

        Enquanto tamanho(lista_numeros ) > 0
                embaralhar(lista_digitos)
                posição = primeiro(lista_numeros)

                Para i de 1        até 9
                          valido = insere_na_grade(grade,
valor_de(lista_numeros))
                        Se valido == verdadeiro então
deleta(primeiro(lista_numeros))
```

```
                                adiciona(lista_visitados,
posição)
                        Fim se
        Fim enquanto
        Se valido == falso
                Se ultimo_backtrack = posição
                        voltas = voltas + 1
                Fim se
                Para i de 1 até voltas
                                adiciona(lista_numeros,
ultima(lista_visitados))
                        deleta(ultima(lista_visitados))
                Fim para
        Fim se

Fim função
```

DOWNLOAD DO CÓDIGO FONTE (GMZ – TODAS AS VERSÕES):

Sources – Google Drive

TESTE NO NAVEGADOR (HTML5 – Versão 0.9):

Clique aqui para acessar

DOWNLOAD (WINDOWS – Versão 0.9):

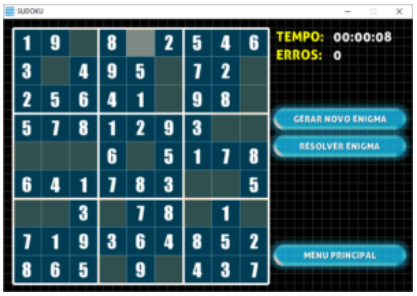
SUDOKU - Download (86 downloads)

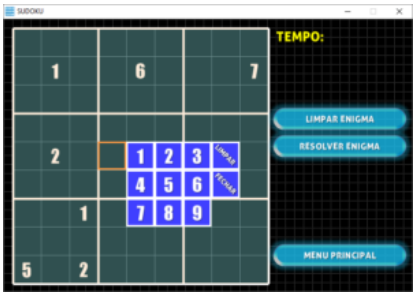
DOWNLOAD (ANDROID – Versão 0.9):

Use o mouse para jogar clicando nos botões. Você pode usar o teclado numérico depois que selecionar uma célula, e se quiser desabilitar o teclado virtual nas opções.

Irei disponibilizar uma versão só com os algoritmos ao decorrer da semana para quem não quer ter o trabalho de retirar toda a “carcaça” do jogo. Por enquanto foque nos scripts da pasta **Grid Scripts**.

IMAGENS DO JOGO:





VIDEO:

DEVLOG:

27-06-2017:

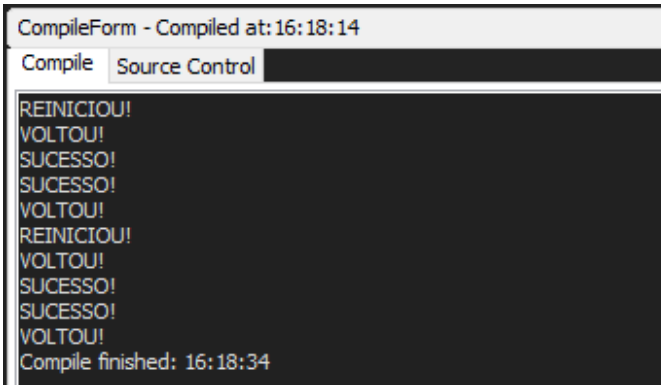
Meus primeiros testes não resultaram em soluções rápidas. Tentei primeiro preenchendo posições aleatórias, e caso uma inserção enviável acontecesse o jogo era reiniciado. Com esse método consegui a solução apenas 1 vez.

Li um pouco mais sobre os algoritmos para implementar o *backtracking*. Porém as condições que havia estabelecido para que a volta acontecesse não eram as ideais e o resultado demorava cerca de 5 min ou mais para ser encontrado. Segue um vídeo curto desse experimento:

Essa foi a primeira parte com o relatório entregue em 28/06/2017.

28-06-2017:

Continuando meus esforços para encontrar um algoritmo que fizesse a volta de forma eficiente, consegui um que resolvia dentro de 1 minuto. Ele ao fazer a volta verificava se já havia colocado o mesmo número na posição de volta, caso sim ele fazia mais uma volta e seguia em frente. Para evitar loops infinitos e reiniciar caso isso ocorresse, criei um vetor que guardava algumas ocorrências (SUCESSO, VOLTA e REINICIO). Reparei que no loop infinito uma certa sequência ocorria: R,V,S,S e V.



Log do compilador

Dessa forma conseguia evitar que travasse. Segue o código que utilizei nessa busca:

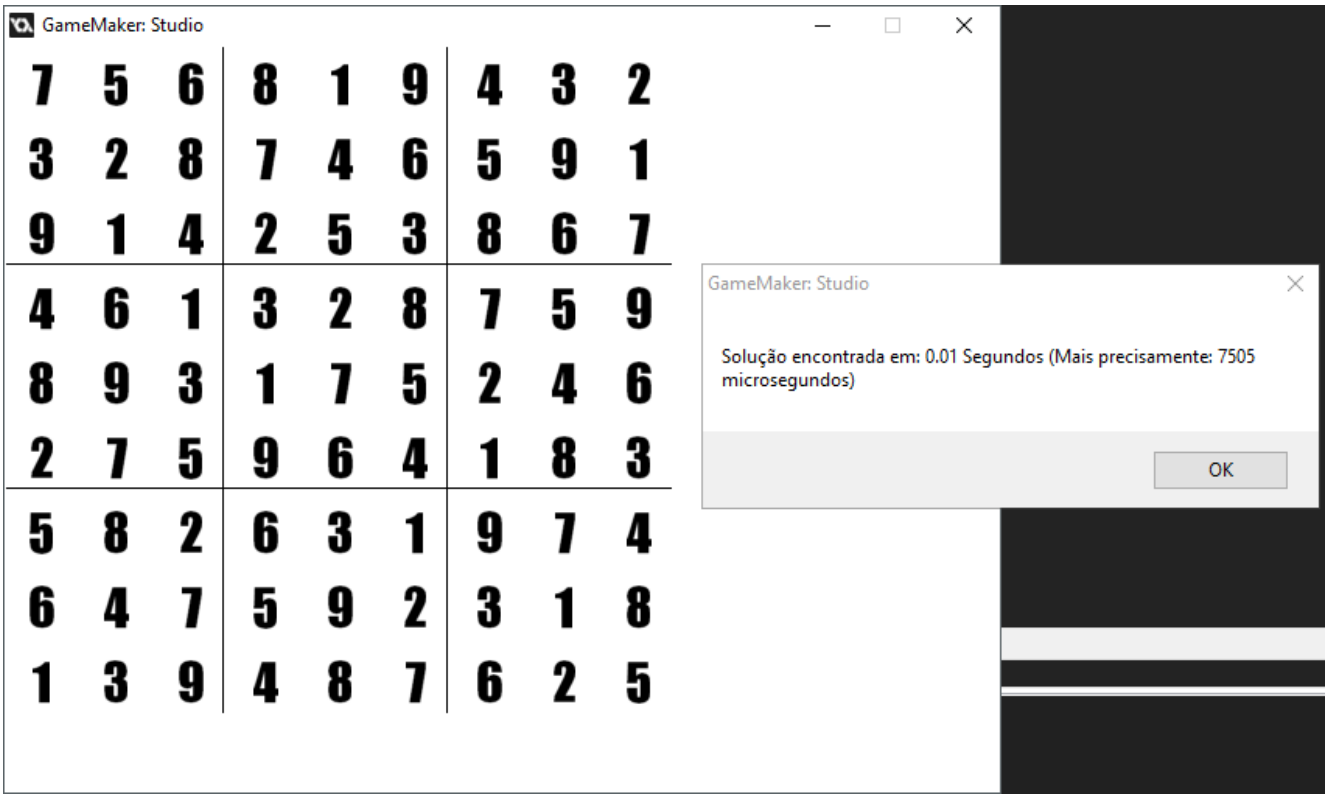
```
if !ds_list_empty(list) //and run
{
// Embaralha números [1-9]
ds_list_shuffle(numbers);

var xx = ds_list_find_value(list, 0) div 9;
var yy = ds_list_find_value(list, 0) mod 9;

// Tentativas por posição
for (var i = 0; i < 9; i++)
{
var val = ds_list_find_value(numbers, i);
var res = grid_place_number(xx, yy, val, grid);

// Caso seja um número viável que não viole as regras
if res == 1
```

```
{
if val != lastPlaced or lastPlaced == 0
{
lastPlaced = val;
ds_list_delete(list, 0);
show_debug_message("SUCESSO!");
pattern[k mod 5] = "s";
break;
}
else
{
//show_debug_message("REINICIOU!");
pattern[k mod 5] = "r";

if pattern[0] + pattern[1] + pattern[2] + pattern[3] +
pattern[4] == "vssvr"
room_restart();
else
{i = 9; break;}
}
}
}

if i == 9
{
//show_debug_message("VOLTOU!");
pattern[k mod 5] = "v";

//grid[# xx, yy] = 0;
ds_list_insert(list, 0, max(ds_list_find_value(list, 0)-1,
0));
}

k++;
}
```

29-06-2017:

Apesar de chegar no resultado, ainda estava muito mal otimizado e lento. Continuei fazendo minha pesquisa e decidi testar uma nova formula. Nessa nova tentativa o algoritmo memorizava a última posição em que fez o *backtracking* e também a quantidade de casas que deveria voltar (De inicio, apenas 1). Caso voltasse para mesma posição, aumentava a quantidade de casas que deveria voltar.

Dessa forma consegui o tempo médio de 0.03 segundos para resolver o enigma. Uma solução bastante satisfatória e que me permite criar tabelas completas e depois apagar algumas posições para criar os jogos.



Versão 0.1

Assim segue a primeira versão para download [Versão 0.1].

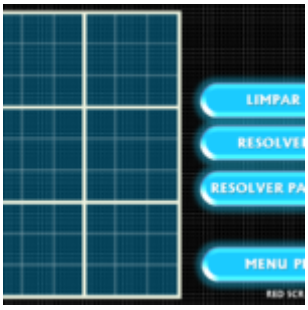
30-06-2017:

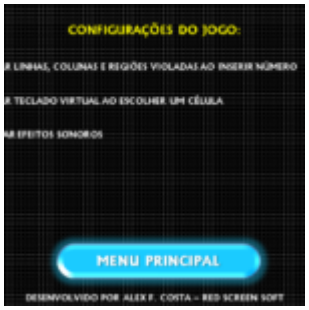
Fiz algumas imagens conceituais para montar o protótipo do que será o jogo.

Tela inicial

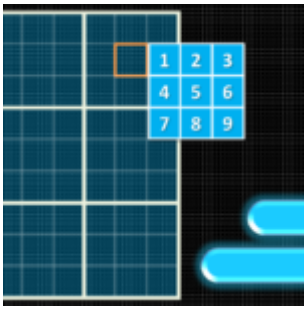
Seleção do nível

Modo de jogo 1


Modo de jogo 2



Configurações



Uso do teclado virtual

01-07-2017:

Consegui fazer vários aprimoramentos no jogo. Coloquei em prática o que fiz nas imagens dos protótipos, adequando melhor à realidade depois de alguns testes.

Essa será a **versão 0.5**, pois conta apenas com o modo de jogo para o usuário resolver enigmas (Apesar de no menu ainda haver a opção). Nesse modo, a máquina gera a tabela completa e depois apaga alguns números de acordo com a dificuldade escolhida.

02-07-2017:

Praticamente finalizei o jogo. Coloquei o modo de construção de enigmas, onde o usuário pode digitar na tabela em branco e o computador tentará resolver.

Tive que fazer várias alterações no algoritmo, pois agora

haviam números fixos a serem considerados. Fiz uma lista de posições já visitadas para que quando a volta for feita não seja apenas decrescida a posição, mas sim consultada na lista.

O jogo pode ser jogado via navegador (HTML5) e há uma versão para download (Windows).

04-07-2017:

Compilei uma versão para Android e enviei para a Google Play.

CRÉDITOS:

- Música: *Funky Element* por *BenSound* (bensound.com);
- Efeitos sonoros gerados usando o Linux Multimedia Studio;

PRINCIPAIS REFERÊNCIAS:

- SUDOKU
- SUDOKU: Solving Algorithms
- Solving SUDOKU in C
- What is the maximum number of solutions a Sudoku puzzle can have?
- Backtracking: Solve Sudoku in Java