

LUDO EM ASSEMBLY MIPS RELATÓRIO

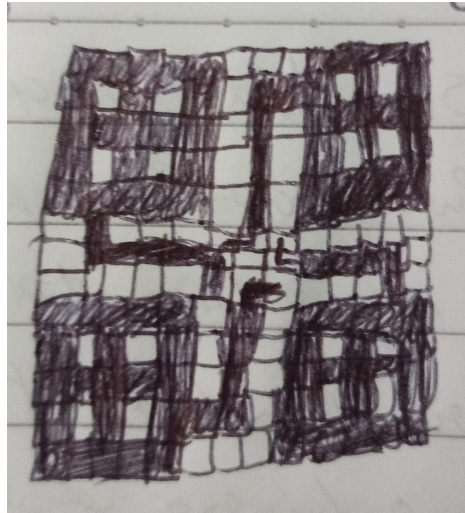
Aluno: Érick de Brito Sousa Lima

Disciplina: Arquitetura e Organização de Computadores

https://github.com/erickxbliv/ludo_asm

Basicamente, Ludo é um jogo de até 4 jogadores, onde cada time leva um turno para movimentar uma peça de sua escolha, a passos aleatórios de um dado, sendo o objetivo dar a volta completa no tabuleiro até entrar e se proteger nas suas respectivas "bases secretas". Portanto para implementar o jogo, eu comecei pelo caminho a ser percorrido, e para eu poder visualizar o andamento, escolhi um vetor de bytes, que em mips foi conveniente o uso de uma string, que representa a casa de nascimento do time no canto superior esquerdo e andava todas as casas do tabuleiro até a casa antes dessa citada anteriormente. O hífen será usado para mostrar as casas vazias, e as letras são as peças e onde elas nascem, já que os times são A, B, C e D. O sistema será: a-----b-----c-----d-----|.

O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
					-	-	-					
					-		b					
					-		-					
					-		-					
-	a	-	-	-				-	-	-	-	-
-												-
-	-	-	-	-				-	-	-	c	-
					-	-						
					-		-					
					-		-					
					d		-					
					-	-	-					



Iniciaremos a programação aplicando nossos dados, dados sobre os gráficos, dados sobre os jogadores, iniciamos nossos registradores com convenções próprias para organizar melhor o funcionamento, que seriam:

```

move $t0, $zero      #casa do a
li $t1, 11           #casa do b
li $t2, 22           #casa do c
li $t3, 33           #casa do d
move $t4, $zero      #backup casa anterior dos jogadores

move $t5, $zero      #registrador de uso instantaneo, imediato
move $t6, $zero      #registrador comum de mexer em enderecos
move $t7, $zero      #registrador de argumento das minhas funcoes

move $s0, $zero      #boolean de vitoria
move $s1, $zero      #endereço do indice no vetor matriz de gp pra mexer nos pixels
move $s2, $zero      #guarda o vencedor
move $s3, $zero      #instantaneo temporario
move $s4, $zero      #resultado do dado
move $s5, $zero      #backup do $ra

```

Também inicializaremos nossas funções, que vem antes da Main e não são executadas diretamente ao compilar. Após isso, temos a rodada de impressões gráficas, que basicamente é salvar em endereços de \$gp os valores referentes ao RGB das cores, que o BitMap display emulará para a gente. Tendo os índices da matriz dos pixels do tabuleiro, e destrinchando para um vetor sozinho, basta guardá-los em um outro vetor e andar suas posições para pegar índice um por um de onde em gp devemos inserir cor x. As peças são colocadas para dormir, e só podem acordar e iniciar o jogo ao retirar o valor 6 no dado. Após isso, entramos no loop do jogo, e dentro desse loop teremos o turno de cada time, que são basicamente as mesmas linhas de comando traduzidas para cada equipe uma de cada vez. Então explicarei apenas um dos times e isso valerá pelos 4. Mais especificamente, vamos falar sobre o time B (azul), pois ele tem uma condição especial que o time A não possui.

```

#MEXER B #####

add $t4,$t1,$zero    #backup fazendo backup
li $v0, 42
li $a1,6             #atualizando dados da funcao random
syscall
addi $a0,$a0,1       #colocar o valor obtido dentro do intervalo de um dado cubico
add $s4,$a0,$zero

la $t5,msg_b         #colocar a mensagem que se altera pra cada peao no instantaneo, e da vez ao usuario
jal usuario

lw $t5,status_b      #ve qual o status que o time esta, dormindo, jogando etc
beq $t5,0,acordar_b
beq $t5,1,jogar_b
beq $t5,2,tubulacao_b #cada possibilidade vai pra um bloco de tratamento diferente

la $s3,fim_b         #caso de erro, pois algum dos casos devia ter acontecido
jr $s3

```

É feito um backup da posição onde ele está e é gerado um número aleatório entre 1 e 6, é chamada a função usuário para mostrar pelo terminal ao usuário o que está acontecendo e pedir que ele interaja na jogada. É então verificado o status do time, se este for o primeiro turno dele, seu status será 0 pois está dormindo, e será chamada a função de acordá-lo.

```

acordar_b:

    bne $s4,6,fim_b #se tiver tirado 6 com a

    lw $t5,primeira_b
    la $t6,caminho
    add $t6,$t6,$t5
    lb $t5,0($t6)          #olha no
    bne $t5,45,fim_b       #so pode

    lw $t1,primeira_b      #ele pega
    lw $t5,b
    la $t6,caminho
    add $t6,$t6,$t1
    sb $t5,0($t6)          #colocar

    lw $t4,dormindo_b
    lw $t7,blue
    lw $s1,dormindo_b      #apagar c
    jal pixel

    lw $t7,dark_blue
    la $t6,estrelas
    lw $s1,4($t6)          #agora mc
    jal pixel

    la $t6,status_b
    li $t5,1               #mudar
    sw $t5,0($t6)

    la $t5,impressao
    jalr $t5               #aqui chama a i
    la $s3,fim_b          #encerr
    jr $s3

```

É verificado se o jogador tirou 6 para poder acordar, e depois se não há nenhuma peça na casa onde ela acorda, caso esteja tudo ok, a peça do time é inserida no caminho, seu pixel é impresso na sua casa estrela, e o rastro de sua soneca também é apagada; por último, seu status é alterado para 1, acordado. Seu turno é finalizado e passa a vez para o próximo, e quando sua vez chegar novamente, ao verificar o status, a execução será jogada para jogar_b:

```

jogar_b:

    add $t4,$t1,$zero
    add $t1,$t1,$s4        #a ja esta no jogo, entao anda no caminho o valor aleatorio que tirou
    bgt $t1,43,Ifb
    j Endifb
    Ifb:
        addi $t1,$t1,-43
        la $t6,ultima_b
        li $t5,9
        sw $t5,0($t6)

    Endifb:

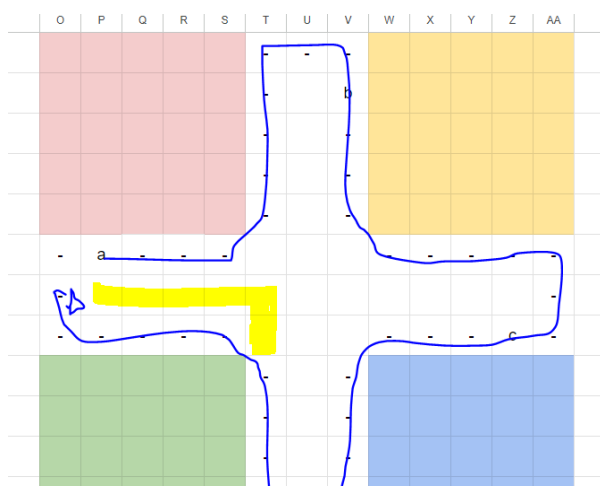
    lw $t5,vazio           #pra poder imprimir o '-' (45)
    la $t6,caminho         #pegar o endereco de caminho
    add $t6,$t6,$t4        #encontra o endereco do indice a ser modificado
    sb $t5,0($t6)         #coloca um '-' no indice anterior de a

    jal devolver_cor       #verifica se ela deixou rastro em uma estrela e ajeita como estava
    move $t5,$zero

    la $t6,rota            #pega o mesmo endereco do vetor dos pixels de caminho
    sll $t5,$t4,2          #tinha um bug porque o backup estava pegando o valor de sair do tabuleiro
    add $t6,$t6,$t5        #anda nesse endereco anda pro indice encontrado
    lw $s1,0($t6)         #pega qual o pixel no vetor-matriz tem que imprimir o pixel preto
    jal pixel

```

O primeiro passo na jogada do time é adicionar à sua posição o valor tirado no dado, e assim



é feita uma verificação importante que não acontece com o time A. Basicamente a peça caminha por todo o tabuleiro até chegar na casa antepenúltima a sua, depois disso ela anda novamente e entra no caminho final da vitória onde não há saída. Relembrando, “a-----b-----c-----d-----” é

uma string que representa nosso caminho, e no caso da peça A ela nasce no índice 0 e chega na porta do tubo na casa 41. O caso das outras peças é diferente, pois o B por exemplo nasce na casa 11 e ganha na casa 9, então como saber se ela já está apta a andar no tapete vermelho? A única peça que não dá a volta no tabuleiro é a 'a', então para as outras, em toda jogada é verificado se a peça bateu nos limites do caminho, para assim ela poder dar a volta completa, e neste momento, é definida qual o índice da casa que a peça precisa passar para ganhar, então sabemos que a casa da vitória é a 9, mas ela fica definida como 100, pois a casa de nascença (11) não é maior, até que o time dê a volta no tabuleiro e seu índice agora é menor que a casa da vitória, então ela é transformada para o valor original. Continuando, como a peça andou, é impresso um hífen na posição anterior dela guardada em \$t4, é chamada uma função para verificar se a peça andou por uma casa estrela, que não deve ser transformada na cor preta base do caminho ao apagar o rastro, que é feito logo em seguida.

```
lw $t5,ultima_b
bgt $t1,$t5,entrou_b
j nao_entrou_b

entrou_b:                                #se entrou no tubo, nao en
    sub $t1,$t1,$t5                      #a posicao atual da

    la $t6,status_b
    li $t5,2
    sw $t5,0($t6)                        #status = modo tubo

    lw $t5,b
    la $t6,vitoria_b
    add $t6,$t6,$t1
    sb $t5,0($t6)                        #andar dentro do ca

    lw $t7,dark_blue
    la $t6,tapete_b
    sll $t5,$t1,2                         #pra saber em qual
    add $t6,$t6,$t5
    lw $s1, 0($t6)
    jal pixel

    beq $t1,5,ganhar_b
    la $s3,terminar_imp_b               #terminar turno
    jr $s3
```

Então acontece uma verificação para saber se a peça passou da sua última casa no caminho e deve entrar no tubo para ganhar. Caso tenha entrado, é subtraído o índice atual pelo índice da última casa para saber onde está a peça no caminho, seu status é trocado para jogar de dentro do tubo e testar a vitória. O jogador é salvo dentro da string própria do caminho que o tubo percorre, e seu pixel é impresso, então é verificado se o jogador já ganhou de cara ou se ele vai ficar preso no tubo até tirar o valor exato para chegar na última casa.

```
nao_entrou_b:                            #se ainda nao entrou no tubo,
    beq $t1,$t0,bmatara
    beq $t1,$t2,bmatarc
    beq $t1,$t3,bmatard
    j pacifico_b
bmatara:
    la $t6,morrer_a
    jalr $t6
    j pacifico_b
bmatarc:
    la $t6,morrer_c
    jalr $t6
    j pacifico_b
```

Agora, caso não tenha entrado no tubo, a peça vai simplesmente andar normalmente pelo caminho, vai ser salva na nova posição da string e seu novo pixel impresso, mas antes disso será verificado se a posição atual da peça é a mesma de alguma outra, caso sim, é chamada a função para matar aquela peça, que mostrarei logo adiante.

Ao final da jogada, é impresso no terminal o caminho e passada a vez do time.

tubulacao_b:

```

add $t5,$t1,$s4
bne $t5,5,fim_b
add $t1,$t1,$s4

lw $t5,vazio           #pra poder
la $t6,vitoria_b       #pegar o en
add $t6,$t6,$t4        #encontra c
sb $t5,0($t6)          #coloca um

lw $t7,blue            #para poder i
la $t6,tapete_b        #peg.
add $t6,$t6,$t4        #encontra o
sb $t5,0($t6)          #coloca a no

lw $t7,dark_blue      #pega a cor
la $t6,tapete_b        #endereço do
sll $t5,$t4,2          #multiplica
add $t6,$t6,$t5        #coloca essa
lw $s1, 0($t6)         #bota no meu
jal pixel

lw $t5,b               #para poder i
la $t6,vitoria_b       #peg.
add $t6,$t6,$t1        #encontra o
sb $t5,0($t6)          #coloca a no

lw $t7,dark_blue      #pega a cor
la $t6,tapete_b        #endereço do
sll $t5,$t1,2          #multiplica
add $t6,$t6,$t5        #coloca essa
lw $s1, 0($t6)         #bota no meu
jal pixel

```

Caso o jogador estiver no tubo, também é simples, será feita a verificação de se o dado entregou o valor perfeito para chegar no último índice, caso sim, é vitória, então não pula pra fora da sequência de execução atual, apaga os dois rastros e imprime os 2 lugares novos da peça, e não encerra o turno, deixa ler linha por linha para que entre na função logo abaixo:

ganhar_b:

```

la $s2,msg_b
lw $t5,ultima_b

```

```

la $t6,fim
jr $t6

```

Aqui, será guardado qual o nome do time que venceu e o loop é encerrado bruscamente, com os dados suficientes para mostrar a vitória dessa peça específica.

morrer_b:

```

move $s5,$ra
lw $t5,vazio
la $t6,caminho
add $t6,$t6,$t1
sb $t5,0($t6)
#jal pixel

lw $t7,dark_blue
la $t6,dormindo_b
lw $s1, 0($t6)
jal pixel

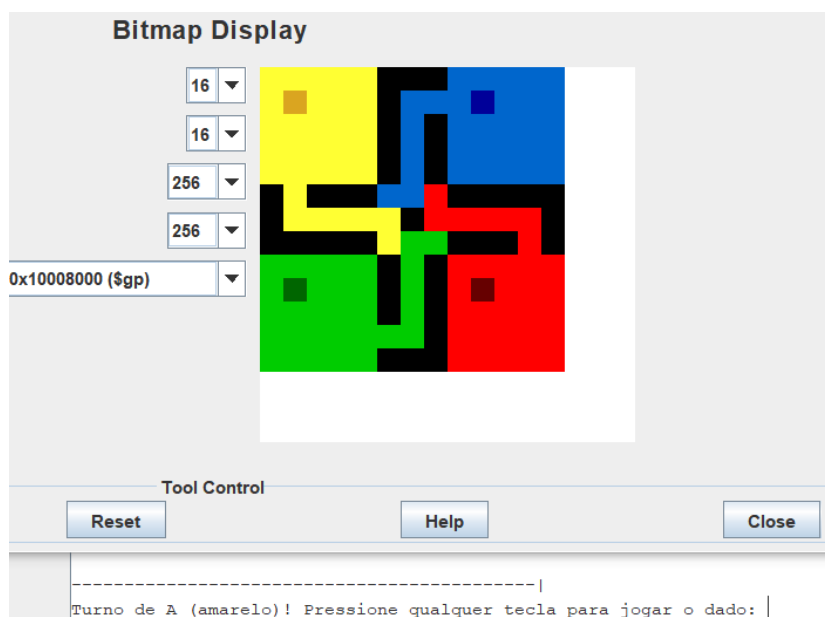
la $t6,status_b
li $t5, 0
sw $t5,0($t6)

```

Para morrer é simples, é o exato oposto de acordar, a posição do time é zerada, o status volta para 0, ele não precisa ser removido da string ou seu rastro apagado, pois já foi sobrescrito pela peça que o derrotou, então ele é mostrado dormindo e a função retorna para onde estava, pois a peça que morre nunca está no seu turno. Depois disso, vem o label que marca onde termina o turno de b, que é chamado de vez em

quando, então daí iniciam-se os turnos dos outros times até que o loop seja encerrado, seja por uma vitória ou mais iterações do que o sistema aguenta.

AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255



Bitmap Display

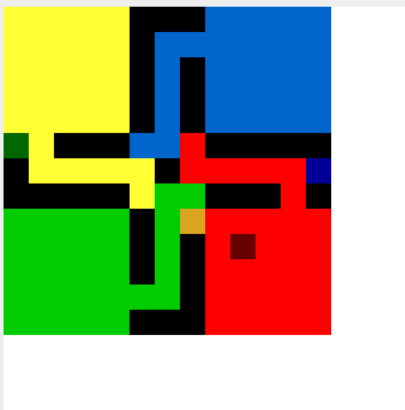
16

16

256

256

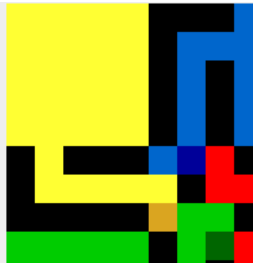
(\$gp)



Tool Control

resetHelpClose

Tirou 1 -----b-----a-----d---|
Turno de B (azul)! Pressione qualquer tecla para jogar o dado: g
Tirou 4 -----b-----a-----d---|
Turno de C (vermelho)! Pressione qualquer tecla para jogar o dado: g
Tirou 4 Turno de D (verde)! Pressione qualquer tecla para jogar o dado: g
Tirou 4 -----b-----a-----d---|
Turno de A (amarelo)! Pressione qualquer tecla para jogar o dado: |



Turno de A (amarelo)! Pressione qualquer tecla para jogar o dado: f
Tirou 1
Parabens! Vitoria de A (amarelo)!
-- program is finished running --