INSTITUTO FEDERAL Brasília	**Instituto Federal de Brasília** **Campus Taguatinga** **Superior em Computação**
	Trabalho 1 - AsciiGotchi **1/2019 - Algoritmos e Programação de Computadores** **Prof. João Victor de A. Oliveira**

Pontuação: 15% da média final.

Membros por grupo: 2 ou 3 estudantes.

Data de entrega: 14/06/19

Forma de Entrega:

- Deve ser enviado um arquivo .zip para o edmodo contendo uma pasta com o nome formado pelo primeiro e último nome concatenado de cada membro do grupo (ex.: JoaoOliveiraMariaSantosPedroLima).
- Dentro da pasta deve possuir apenas o código fonte do programa a ser desenvolvido na linguagem C, com o mesmo nome da pasta, acrescido da extensão .c (ex.: JoaoOliveiraMariaSantosPedroLima.c).

AsciiGotchi

O *Tamagotchi®,* é um brinquedo japonês lançado em 1996 pela *Bandai©*. A motivação deste brinquedo consiste em cuidar de um monstrinho virtual como se fosse real, alimentando-o, dando-lhe carinho, limpando-o, dentre outras atividades. Além disso, o monstrinho virtual, ao longo do tempo, se desenvolve e vai envelhecendo. Caso o monstrinho não seja bem cuidado pelo seu dono, há uma possibilidade de que ele venha a falecer. A figura 1 mostra um *tamagotchi*.

Figura 1: Exemplo de um Tamagotchi.

Neste trabalho criaremos uma versão simplificada de *Tamagotchi* chamada ***AsciiGotchi.*** Nosso monstrinho virtual será representado por um símbolo na tabela ASCII (figura 2), variando de “@”, quando o monstrinho é apenas um ovo” até “Z”, quando o monstrinho já é um senhor de idade pronto para partir…”.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

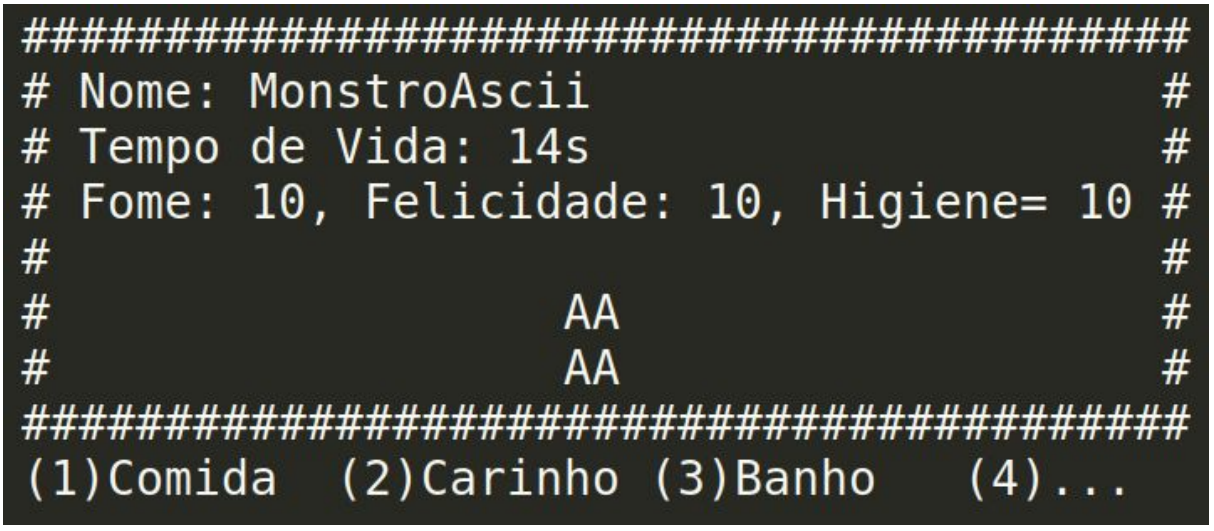
Figura 2. Tabela ASCII.

Nosso AsciiGotchi será desenvolvido na linguagem C padrão e poderá ser desenvolvido em grupo de até 3 estudantes. A seguir, são descritas as especificações técnicas deste programa.

Interface com o usuário (TELA padrão)

A cada segundo, a tela deve ser atualizada mostrando:

- Nome, tempo de vida, o *status* fome, felicidade e higiene do monstrinho;
- O monstrinho (formado por um bloco 2x2 usando o símbolo ASCII referente ao seu estágio de evolução;
- Opções do usuário: (1)Comida (2)Carinho (3)Banho (4)....



A tela do jogo deve ser atualizada a cada segundo.

Características do monstro ASCII

Todo monstro Ascii possui as seguintes características:

- **Nome:** a ser fornecido no começo do jogo;
- **Tempo de vida:** iniciado com zero no começo do jogo e alterado em uma unidade a cada segundo de jogo;
    - Obs.: Deve ser reiniciado no momento em que ovo nasce (o monstro se torna “A”);
    - Opcional: antes de nascer o tempo de vida aparece na tela com o termo “Tempo de gestação”;
- **Fome:** *status* que indica o grau de fome do monstrinho;
    - Inicia-se com valor 10 e pode variar de 0 a 15;

- **Felicidade:** *status* que indica o grau de felicidade do monstrinho;
    - Inicia-se com valor 10 e pode variar de 0 a 15;
- **Higiene:** *status* que indica o grau de higiene do monstrinho;
    - Inicia-se com valor 10 e pode variar de 0 a 15;
- **Grau de evolução:** é indicado por 4 caracteres ASCII (bloco 2x2) variando de “@” a “Z” (é o próprio monstrinho).

Ações do usuário

O usuário pode a cada 10 segundos agir com seu monstrinho selecionando alguma das seguintes opções:

1. **Comida:** Fornece alimentação ao monstrinho;
    a. Deve aparecer a mensagem: “<nomeMonstro> hora de comer!\n”, “Humm, que delícia mestre!\n”;
    b. Esta ação diminui em 2 o grau de fome e diminui em 2 a higiene do monstrinho;
2. **Carinho:** Fornece felicidade ao monstrinho;
    a. Deve aparecer a mensagem: “O carinho do mestre me faz bem :) \n”;
    b. Esta ação aumenta em 3 ao grau de felicidade e aumenta em 1 o grau de fome;
3. **Banho:** Limpa o monstrinho;
    a. Deve aparecer a mensagem: “<nomeMonstro> hora do banho!\n”, “Naooooooo!\n”;
    b. Esta ação aumenta em 7 o grau de higiene e diminui em 4 o grau de felicidade;
4. **… :** Ignora o monstro;
    a. Basicamente esta ação não faz absolutamente nada;

Caso o usuário digite outra opção, deve-se exibir a mensagem “O que foi isso mestre?! O que queria de mim?” e reduzir 5 pontos do grau felicidade do monstrinho.

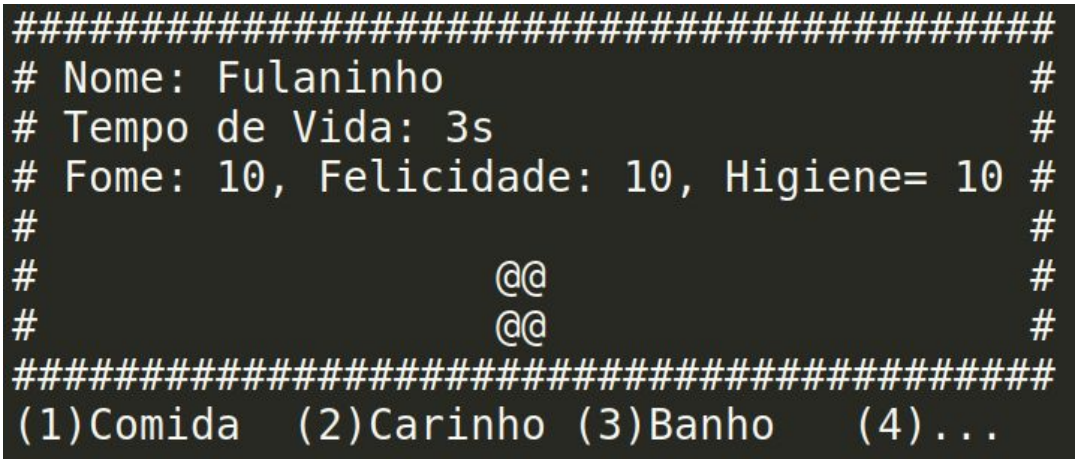
O usuário só poderá dar alguma ação novamente na próxima rodada (em 10

segundos).

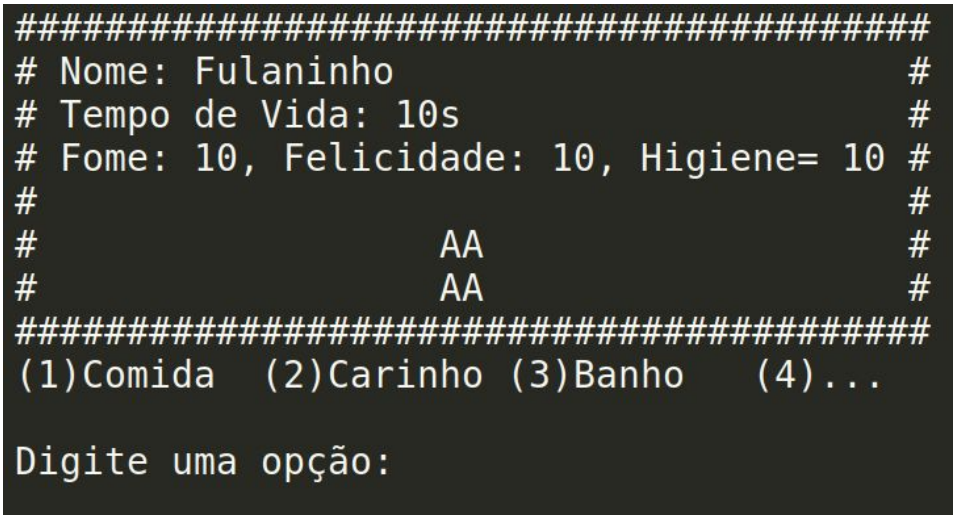
Obs.: o tempo é parado no momento em que o usuário precisa digitar alguma ação.

Andamento do jogo

O jogo começa perguntando ao usuário “Qual o nome do monstrinho Ascii?”. Após isso, é apresentada a tela inicial com o monstrinho formado por um bloco 2x2 com o caractere “@”, indicando que o monstrinho ainda é um ovo.



No 9° segundo de vida, o ovo deve eclodir, ou seja deve-se transformar o ovo em um bloco 2x2 de caracteres “A”. No próximo segundo (10º segundo de vida e em todos os múltiplos de 10 segundos) é dada a chance do usuário interagir com o monstrinho.



A cada ação do usuário, caso essa ação envolva a exibição de uma

mensagem, essa mensagem deve ficar disponível por 2 segundos e depois sumir da tela (esses dois segundos devem ser atualizados na tela do jogo).

O monstrinho evolui a cada **XX** segundos, onde **XX** é o decimal correspondente ao caractere ASCII que ele é formado (ex.: o caractere ‘A’ é mapeado no decimal 65, ou seja, ele se tornará ‘B’ em 65 segundos). Ao evoluir, deve aparecer uma mensagem por dois segundos: “Parabéns <nomeMonstro> evoluiu!”

A cada 5 segundos (apenas depois de ter nascido) nosso monstrinho pode aleatoriamente: ficar com fome (grau de fome é incrementado em 1); ou ficar carente (grau de felicidade decrementado em 1); ou se sujar (grau de higiene decrementado em 1).

Situações em que o monstrinho morre (“Game over!”):

- **“<nomeMonstro> morreu de fome! :(\n”**
  - Ocorre quando o grau de fome chega a 15;
- “**<nomeMonstro> morreu de bucho cheio!\n”**
  - Ocorre quando o grau de fome chega a zero;
- **“<nomeMonstro> morreu de desgosto =| \n”**
  - Ocorre quando o grau de felicidade chega a zero;
- **“<nomeMonstro> morreu de rir :D ops... :(\n”**
  - Ocorre quando o grau de felicidade chega a 15;
- **“<nomeMonstro> morreu de uma doença infecciosa! \n”**
  - Ocorre quando o grau de higiene chega a 0;
- “**<nomeMonstro> morreu por falta de anticorpos!\n”**
  - Ocorre quando o grau de higiene chega a 15;
- **“<nomeMonstro> teve uma bela vida :)”**
  - Ocorre quando o monstro chegar ao estágio de evolução “Z”.

Nas situações de morte, além de exibir as mensagens acima, deve-se terminar o programa.

Dicas de implementação

Como podemos simular o tempo neste jogo?

Para simular o tempo neste jogo usaremos a biblioteca **unistd.h,** que deve ser incluída no começo do código usando a seguinte expressão:

#include <unistd.h>

Para que o programa funcione a cada 1 segundo, usaremos a função pré-definida **sleep,** da seguinte maneira:

sleep(1), onde 1 é o tempo em segundos em que o programa fica parado

Como podemos atualizar (limpar) a tela, de modo que ela fique no mesmo local durante toda execução?

Para limpar a tela a cada segundo do programa, iremos usar a biblioteca **stdlib.h**, que deve ser incluída no começo do código usando a seguinte expressão:

#include <stdlib.h>

Podemos limpar a tela sempre que usarmos a função pré-definida **system("clear"),** para linux, ou **system("cls"),** para windows. Existem formas bem melhores de se fazer isso usando a biblioteca ncurses.h, por exemplo, mas foge do escopo de nossa disciplina.

Como podemos escolher aleatoriamente o que ocorrerá a cada 5 segundos com o monstrinho?

Podemos gerar números pseudo-aleatórios usando a biblioteca **stdlib. h** e a biblioteca **time.h**

Para isso, é necessário utilizar, uma única vez durante todo o programa a instrução:

srand(time(null))

Usada para gerar uma sequência de valores aleatórios através de uma sequência (que neste caso usa o tempo como gerador de aleatórios). Após usar a instrução acima, podemos coletar um número aleatório usando a função pré-definida rand:

numero = rand() % num

Onde um número aleatório, variando de zero a **num -1,** é armazenado na variável número.