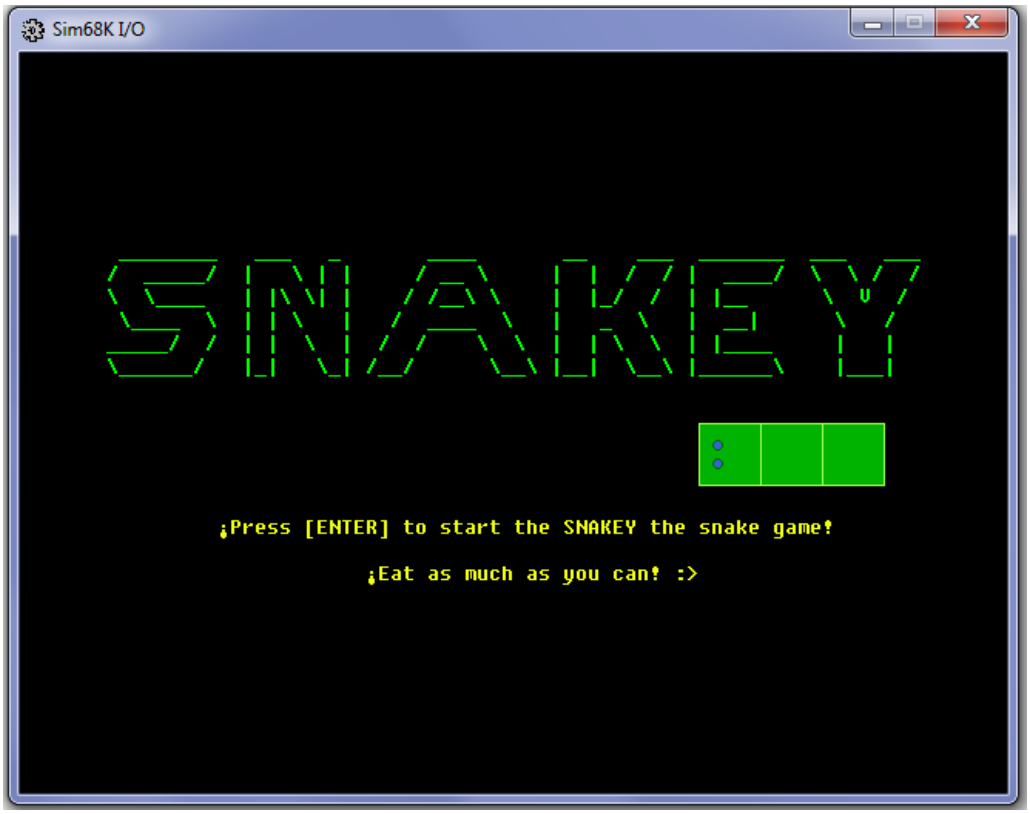
PRÀCTICA 2. SNAKEY



Curs 2015/2016

Estructura de Computadors II GEIN 2010

Documentació sobre la segona pràctica no-presencial de la asignatura Estructura de Computadors II.

Task Manager i Videojoc. Snakey

Miquel Àngel Román Colom.
43235792R

ESTRUCTURA DE COMPUTADORS II GEIN 2010

ÍNDEX.

1.- IDENTIFIER POOL

1.1.- IDP. BREU DESCRIPCIÓ

L'identifier pool és, com diu el seu nom, una "piscina" d'identificadors *assignables* a una tasca.

És a dir:

És el mòdul encarregat de la gestió d'els identificadors que assignarem a les diferents tasques que es vagin executant sobre el nostre *Task Manager*, i aquest, les manejarà.

1.2.- IDP. GET ID

La subrutina IDP_GET_ID s'encarrega de proveïr un identificador de tasca que estigui disponible.

L'output és assignat al registre d'el Motorola 68000, D0.W, que contendrà $FFFF si no queden identificadors disponibles.

```
*-----------------------------------------------------------
IDP_GET_ID:
* Description : Provides an available ID. Selects the first item
*               from the list of available IDs.
* Pre         : Nothing.
* Post        : D0.W: The ID or $FFFF if no ID available
* Modified    : Nothing: all registers are restored.
* Note        : When calling this subroutine, the required constants
*               are assumed to have the correct values.
*-----------------------------------------------------------
                MOVE.L A0,-(A7)

                MOVE.L #0,-(A7)
                MOVE.L #IDP_LIST,-(A7)
                BSR LS_GET_ITEM
                ADDQ.L #4,A7
                MOVE.L (A7)+,A0

                MOVE.W 4(A0),D0

                MOVE.L A0,-(A7)
                MOVE.L #IDP_LIST,-(A7)
                BSR LS_REMOVE
                ADD.L #8,A7


                MOVE.L (A7)+,A0
                rts
*-----------------------------------------------------------
```

1.3.- IDP. RELEASE ID

La subrutina IDP_RELEASE_ID s'encarrega d'afegir un identificador a la llista d'identificadors lliures d'aquest mòdul.

Aquest identificador vé donat per paràmetre al registre D0.W

```
*-----------------------------------------------------------
IDP_RELEASE_ID:
* Description : Returns the specified ID to the list of available
*               IDs so that they will be available in future calls
*               IDP_GET_ID
* Pre         : D0.W: ID to release
* Post        : ID released
* Modified    : Nothing: all registers are restored.
* Note        : When calling this subroutine, the required constan
*               are assumed to have the correct values.
*-----------------------------------------------------------

                MOVE.W D0,-(A7)
                MOVE.L #IDP_LIST,-(A7)
                BSR LS_PUT
                ADD.L #6,A7

                rts
*-----------------------------------------------------------
```

2.- DYNAMIC MEMORY MANAGER

2.1.- DMM. BREU DESCRIPCIÓ

El DMM és el mòdul encarregat de dur a terme la gestió de la memoria dinámica que empraran les nostres tasques. Aquest manejara una llista de blocs de memòria que s'aniran assignant a les tasques.

2.2.- DMM. RELEASE

La subrutina DMM_RELEASE s'encarrega d'alliberar un d'aquests blocs de memòria dels quals he parlat a l'apartat anterior. Els retorna a la llista de blocs de memòria lliures perque puguin esser assignats a una nova tasca que el requereixi.

```
*-----------------------------------------------------------
DMM_RELEASE:
* Description : Releases the specified memory chunk.
* Pre         : A0: Pointer to the memory chunk to release. Note that
*                   A0 points to the data, not to the NEXT field
* Post        : Memory chunk released.
* Modifies    : Nothing
* Note        : When calling this subroutine, the required constants
*               are assumed to have the correct values.
*-----------------------------------------------------------

                MOVE.L A0,-(A7)

                SUBQ.L #4,A0
                MOVE.L A0,-(A7)
                MOVE.L #DMM_MEMLIST,-(A7)
                BSR LS_REMOVE
                ADD.L #8,A7

                MOVE.L (A7)+,A0
                rts
*-----------------------------------------------------------
```

3.- TASK MANAGER

3.1.- TM. DESCRIPCIÓ

El *Task Manager* és el gestor multitasca que s'ens dona ja mig implementat i que nosaltres hem d'acabar de desenvolupar.

És el mòdul "*central*" que uneix tots els altres mòduls, fent-ne ús d'ells, per tal de aconseguir la gestió de varies tasques que serán implementades en altres mòduls tals com:

easy_tasks.X68 complex_tasks.X68 i *tasks.X68*.

Els dos primers ja s'ens donen completament desenvolupats per el professor (*Antoni Burguera Burguera*) i ens serviran com a *tests* per poder verificar el correcte funcionament de la part que haurem desenvolupat.

El darrer és el videojoc que haurem desenvolupat nosaltres. D'aquest mòdul en parlaré més envant en el seu propi apartat.

3.2.- TM. TRAP #0

El TRAP #0 *dispatcher* o també anomenat *TM_FGROUP_BASIC* és la part d'el *TM* que s'encarrega de dur a terme les funcions bàsiques d'introducció i borrat de tasques.

En aquest apartat, com a als altres, vull remarcar que només es documentarà el codi desenvolupat per l'estudiant ja que es considera que el corrector (*el pròpi professor*) es suposa que sap que fa i que no fa el seu codi.

La funció ***.F2*** s'encarrega de dur a terme "el borrat" de les tasques que tenguin el mateix identificador de grup que s'espera per l'input.

La funció que s'ha programat en aquesta secció està recollida en una subrutina d'usuari que espera tenir en D1.W el grup de les tasques que es volen eliminar.

Aquesta subrutina *TM_REMOVE_TASKS_BY_GROUP* serà descrita descrita en el seu apartat corresponent d'aquest document.

3.2.- TM. TRAP #1

El TRAP #1 *dispatcher* o també anomenat *TM_FGROUP_EXECUTION* és la part que s'encarrega de les funcionalitats relacionades amb l'execució de les tasques.

L' ***.F1*** s'encarrega d'accedir a la següent tasca que es trobi en un estat *executable*.

Per això s'empraran les subrutines ja desenvolupades per el professor *TM_GET_NEXT_EXECUTABLE_TCB* i *TM_PREPARE_TASK_EXECUTION* també s'ha reutilitzat el codi de la ***.F0*** d'aquest mateix *trap dispatcher*.

Si la primera subrutina esmentada ens ha retornat ***$FFFF FFFF*** significarà que no queden més tasques executables i que, per tant, és hora de començar un nou cicle d'el nostre *Task Manager* saltant un altre cop a la ***.F0*** d'aquest mateix *trap dispatcher*. Però abans de poder iniciar un nou cicle de *TM* necessitarem executar la fase de borrat i de *delay*, si és necessari.

Per tant, en el cas que ha estat remarcat, saltaríem a la etqueta *.newPhase:* i d'aquesta executariem les subrutines *TM_ERASE_ERASABLE_TASKS* i *TM_DELAY_FASTER_TASKS* que serán descrites en el seu apartat. Un cop fet això esteim en condicions de iniciar un nou cicle de *TM* i durem a terme el BRA ***.F0***.

A la ***.F2*** s'executa una OR entre els flags de !(*TM_TIME_OVER) i (TM_ALLOW_TIME_OVER*).

La resta de *funcions* d'aquest *trap dispatcher* (***.F4 i .F5***) són bastant símples i en veure-les s'*autodefineixen*.

```
.F4:            ; Function 4: Wake up task
                ; Input      : A2  Task handler
                ; Output     : ---
                ; Note       : Task handler must be valid
                ; -----------------------------
                ; | DONE. NOTHING MUCH TO TELL |
                ; -----------------------------
                MOVE.L 40(A7),A0
                MOVE.W #TM_TCB_STATUS_RUN, TM_TCB_STATUS_W(A0)
                bra     .END
```

```
.F5:            ; Function 5: Re-init task
                ; Input      : A2  Task handler
                ; Output     : ---
                ; Note       : Task handler must be valid
                ; -----------------------------
                ; | DONE. NOTHING MUCH TO TELL |
                ; -----------------------------
                MOVE.L 40(A7),A0
                MOVE.W #TM_TCB_STATUS_INIT, TM_TCB_STATUS_W(A0)
                bra     .END
```

3.3.- TM. TRAP #2

En aquest *trap dispatcher* només s'ha hagut de dur a terme el desenvolupament de una *funció* i aquesta fa un salt a una subrutina que no he tingut que desenvolupar jo i llavors fa un salt a una altra part d'el codi que tampoc he implementat jo; per tant consider que algunes fotografíes ho faran més evident que un bloc de text.

```
.F1:            ; Function 1: Get first task info
                ; Input      : ---
                ; Output     : D1.W Task ID
                ;            : D2.W Task Group
                ;            : D3.W Status
                ;            : A0   TMP
                ;            : A1   SMP (Shared Memory Pointer)
                ;            : A2   Task handler or #$FFFFFFFF if no tasks

                ; -------------------------------
                ; | NOTHING MUCH TO TELL EITHER |
                ; -------------------------------
                BSR TM_GET_FIRST_TCB
                BRA .OUTINFO
                bra     .END
```

```
.OUTINFO:       move.l  A0, 40(A7)
                cmp.l   #$FFFFFFFF, A0
                beq     .END
                move.w  TM_TCB_ID_W(A0), 6(A7)
                move.w  TM_TCB_GROUP_W(A0), 10(A7)
                move.l  TM_TCB_TMP_L(A0), 32(A7)
                move.l  #TM_SHARED_MEMORY, 36(A7)
                move.w  TM_TCB_STATUS_W(A0), 14(A7)
                bra     .END
```

3.4.- TM. TRAP #3

En aquest *trap dispatcher* no s'ha hagut d'implementar cap funció. Però aquest *trap dispatcher* és l'encarregat de dur a terme les funcions de configuració de les tasques.

Les configura segons les conveniencies de l'usuari que vendran indicades en forma de input en la funció corresponent.

3.4.- TM. SUBRUTINES DESENVOLUPADES PER L'ALUMNE

→TM_REMOVE_TASKS_BY_GROUP:

Aquesta subrutina s'encarrega de realitzar un recorregut a través de tota la llista de *TCBs* i marca com a *removables* tots aquells que tenguin el mateix identificador de grup que el passat per paràmetre.

→TM_ERASE_REMOVABLE_TASKS:

Aquesta part d'el mòdul s'encarrega de realitzar un recorregut de la llista de *TCBS* i elimina tots aquells **task control blocks** que hagin estat marcats com a removables. Alliberant el seu bloc de memòria assignat per el mòdul DMM descrit en apartats anteriors, alliberant l'identificador que li ha estat proporcionat a través d'el mòdul IDP que també ha estat descrit en apartats anteriors i finalment elimina el **TCB** de la llista de TCBs que maneja el *Task Manager*.

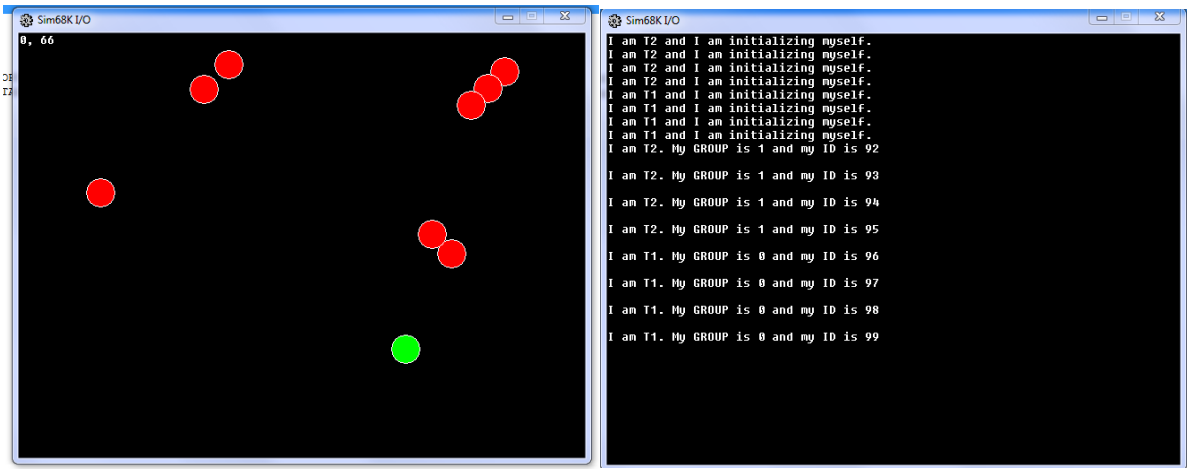
→TM_DELAY_FASTER_TASKS:

En aquesta part, és calcula el temps d'execució de les tasques mitjançant una diferència que vé indicada al codi. (*TM_PREVIOUS_TIME –(temps actual calculat amb una funció d'el TRAP #15)*).

Si aquesta diferència esmentada ara, és inferior a *TM_MIN_CYCLE_TIME*, s'en calcula la diferència i aquesta s'afegeix com a *delay* en una altra funció d'el TRAP #15

Aquesta subrutina també actualitza el flag *TM_TIME_OVER* sempre que el temps d'execució hagi estat superior al *TM_MIN_CYCLE_TIME*.

Abans de seguir cap al mòdul d'el videojoc, voldría afegir una sèrie de captures d'el correcte funcionament d'els mòduls donats per el professor.



4.- VIDEOJOC. *SNAKEY THE SNAKE*

4.1.- SNAKEY. DESCRIPCIÓ

Aquest mòdul d'el programa és on es troben les diferents tasques a executar per poder dur a terme el fet de poder jugar a un videojoc.

En el meu cas, m'he decidit per fer una versió d'el videojoc de *l'Snake* tradicional. El jugador jugarà com *l'Snakey* una serp afamolida que només coneix les direccions verticals i hortizontals per ser capaç d'arribar a les monedes blaves (*les seves preferides*) que li fan créixer 2 blocs d'el cos cada cop que s'en endrapa una.

4.2.- SNAKEY. TASK 0

Aquesta tasca és la tasca de menú un cop l'usuari engega el programa. Com a variables de tasca te el nom de la serp i un parell més de *Strings* per guiar l'usuari dins el programa cap al començament d'el joc.

4.3.- SNAKEY. TASK 1

És una de les tasques gestores. És la tasca d'el doble buffer, que ja venia implementada en el *complex_tasks.X68*.

S'encarrega d'habilitar el *DoubleBúffer*, inicialitzar la finestra i asignar-li la resolució que vé declarada com a constant al principi d'aquest fitxer de codi font.

També s'encarrega de despintar i tornar a pintar la pantalla.

4.4.- SNAKEY. TASK 2

És l'altra tasca gestora. Aquesta tasca és la que gestiona l'entrada de teclat i inicialitza la gran majoria de variables que estarán dins memòria compartida.

Aquesta tasca és la encarregada de discernir entre si el joc està en funcionament, o està en pantalla de menú. Només llegirà la tecla "*enter*" quan estigui en *estat* de menú i llegira l'es fletxes de la dreta d'el teclat quan estigui en *estat* de joc.

Fer el canvi d'estat de menú a estat de joc implica eliminar les tasques que formen part del menú i activar les que formen part de l'snake, el coin i l'score.

4.5.- SNAKEY. TASK 3

Aquesta, és la tasca *coin/token/moneda/fruita* o tots els noms que es vulguin assignar a l'element que l'*Snakey* haurà d'anar a cercar per tal de poder avançar en el joc.

En el joc, aquesta tasca estarà constantment mirant si *l'Snakey* se la menja. Si això passa és reproduïrà un sò, el comptador d'*score* agumentarà en 1 i es generarà una altra tasca d'aquest tipus amb un nou *spawn point*.

4.6.- SNAKEY. TASK 4

La tasca 4 correspon al comptador.

El comptador *Score* és el que informa a l'usuari de la puntació que porta. Augmenta en 1 cada cop que l'*Snakey* es menja una moneda. Quan es perd o es guanya es passa la puntuació a un altre lloc de memòria compartida per despres poder transmetre un missatge al jugador a les noves pantalles de menú.

4.7.- SNAKEY. TASK 5

Snakey. Tasca 5: Snakey.

És la tasca més complexa i amb diferència a totes les altres. També podriem dir que n'és la tasca principal ja que principalment és l'objecte "*de joc*" de tota aquesta *temática*.

L'Snakey és mou a través de la pantalla interaccionant amb el teclat a través de memòria compartida.

Aquesta tasca és la que mira si *l'Snakey* mor. Que *l'Snakey* mori pot implicar 2 coses:

→ Que *Snakey* hagi col·lisionat amb una paret. (Cantons de la pantalla).

→ Que *Snakey* s'hagi mossegat a ella mateixa i, haguem tengut que córrer cap a l'hospital.

Tot això s'analitza a la subrutina *T5_UPDATE* pertanyent a aquesta tasca.

Que *l'Snakey* mori implica eliminar la seva tasca, la de comptador, i la de coin que esteien operatives en el momento que s'ha produït la mort.

Cal indicar que *l'Snakey* és comporta de manera distinta si se li diu que pertany a un grup o un altre.

Si pertany al grup *TK_GROUP_MENU* farà voltes entorn a les lletres de les pantalles de menú i no respondrà als canvis de teclat.

Si pertany al grup *TK_GROUP_SNAKE* serà ella mateixa i respondrà a les ordres d'el jugador que vendran donades a través de la tasca *keyboard*.

4.8.- SNAKEY. TASK 6

La tasca 6, de manera idéntica duu a terme les funcions que duia la tasca 0.

És un menú més, però amb una sèrie d'*Strings* més que permeten imprimir per pantalla *loser* per indicar al jugador que ha matat a pobra *Snakey*. Entre d'altres també deixa veure la puntuació máxima que s'ha aconseguit prèviament, per indicar-li que ho pot fer millor.

Quan s'engega aquesta tasca, es pot tornar a inciar la fase de joc.

4.9.- SNAKEY. TASK 7

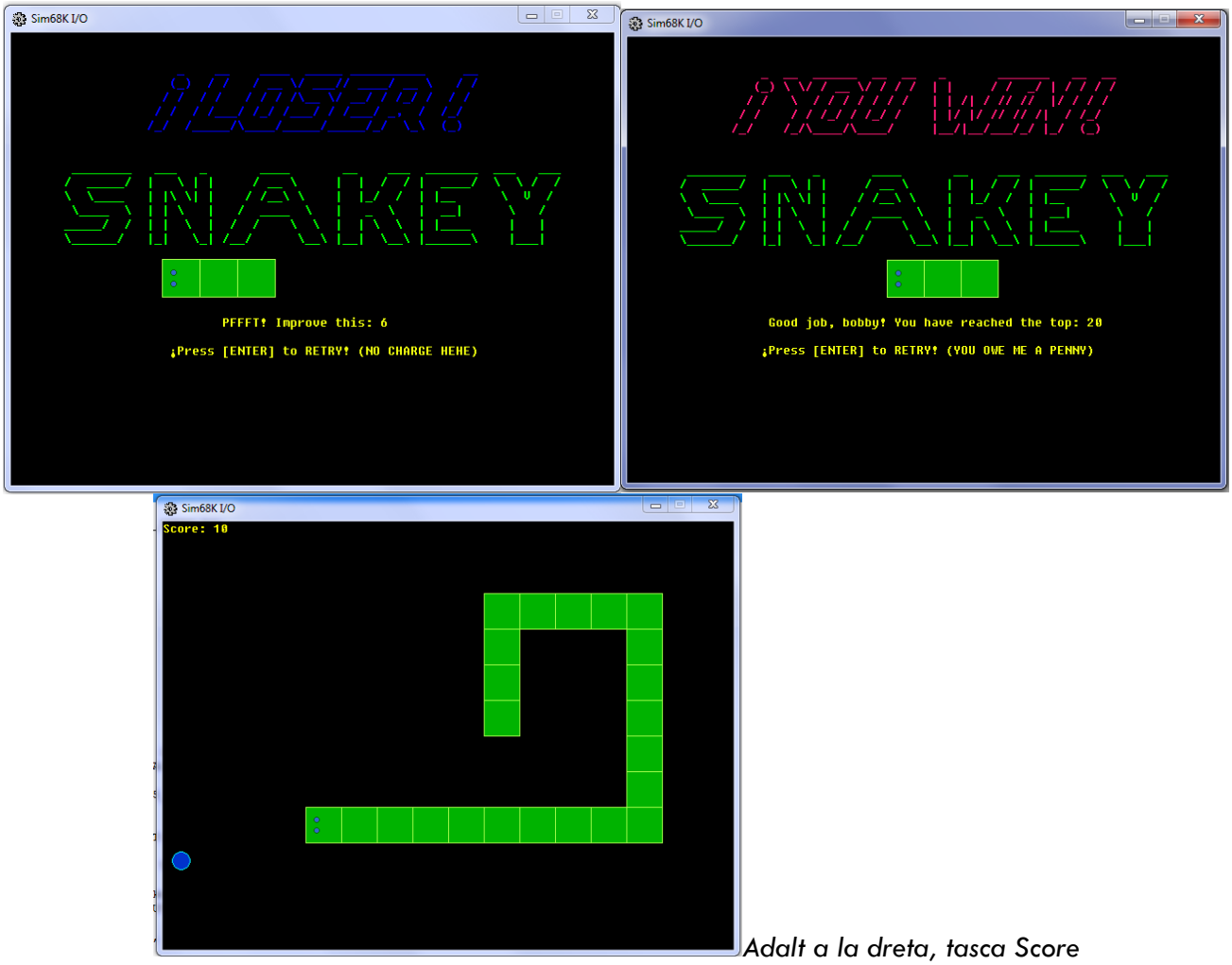
La tasca 7, de manera idéntica duu a terme les funcions que duia la tasca 0.

És un menú més, però amb una sèrie d'*Strings* més que permeten imprimir per pantalla *YOU WIN* per indicar al jugador que ha assedegat la fam de la pobra *Snakey*.

Entre d'altres també congratula al jugador per assolir l'objectiu d'el joc.

Quan s'engega aquesta tasca, es pot tornar a inciar la fase de joc.

Algunes imatges d'el joc mentre s'hi juga



Adalt a la dreta, tasca Score

Abaix a l'esquerra tasca Coin

El verd es l'Snakey en estat de joc.