

Trabajo Práctico grupal

Comisión 2

Organización de Computadoras 2022

Universidad Nacional de Quilmes

Información general

En el presente trabajo práctico grupal se trabajarán los contenidos que iremos aprendiendo durante la cursada.

*El objetivo consiste en aplicar los conocimientos teórico-prácticos de cada concepto, esto implica resolver ejercicios que integran varios conceptos; de esta manera, el trabajo práctico, será una evaluación integradora de todos los contenidos. Tener en cuenta que **para avanzar en el mismo será sumamente importante realizar las guías prácticas correspondientes** para una mejor comprensión y resolución de los ejercicios.*

Justificación de ejercicios

En todos los ejercicios se deberá justificar adecuadamente la resolución del mismo, se pida o no explícitamente en el enunciado. Esto implica redactar en español la respuesta, y de pertinente, incluir la simbología y sintaxis técnica correspondiente al tema. Recordar que el trabajo es una evaluación académica formal, la cual debe ser respetada como tal.

Introducción

La temática del trabajo consiste en resolver ejercicios relacionados a un videojuego. A continuación se detalla la información necesaria y general del dominio con el cual vamos a trabajar durante el mismo, sin embargo prestar atención, que en ciertos ejercicios, el enunciado se complementa con más información relevante para su resolución puntual.

Dominio

El **pacman** es un videojuego donde un personaje se mueve por un tablero en 4 posibles direcciones tratando de comer monedas para acumular puntos. Mientras se mueve, el pacman debe huir de los fantasmas, pero si sus esfuerzos son en vano y es alcanzado por uno de ellos, entonces pierde una vida. Si la cantidad de vidas llega al valor cero (0), el juego ha finalizado.

La persona que juega utiliza las flechas del teclado para mover el Pacman por el espacio de juego, limitado por las paredes del tablero.

Sistema de puntaje

Cada casillero del tablero puede contener diferentes tipos de premios (*prizes*): monedas (*coins*), o premios especiales (*bonus*). Cuando el pacman come una moneda se suman 10 puntos, y cuando come un premio especial se suman 100 puntos. Tener en cuenta, que cada casillero

puede tener sólo una moneda o un bonus, puede tener ambos premios o estar vacío. Por lo tanto, en caso que el casillero tenga ambos premios, el puntaje final se actualizará con la suma de ambos, mientras que si se encuentra vacío suma 0 puntos (nada).

Al acumular o superar 2048 puntos, se suma una vida y se reinicia el puntaje, en caso de no haber llegado a cero (0), manteniendo el excedente de puntos en caso que corresponda.

1. Precalentamiento (no obligatorio)

La presente sección no es parte de la entrega obligatoria, la misma tiene como objetivo el entendimiento del contexto (propiciado por el dominio del sistema), mediante el aprendizaje de los primeros conceptos de la materia. Los ejercicios propuestos, son de complejidad simple, pero necesarios para la resolución de los ejercicios integradores de la entrega.

1.1. Sistema de numeración: Representación del juego

La posición del pacman en el espacio del tablero se representa mediante dos coordenadas, denominadas **X** e **Y**, que están representadas en un sistema BSS(7). Para la representación del estado del juego se utiliza el formato **PacStatus** de **32 bits** que estructura los datos de la siguiente manera:

X (7b)	Y (7b)	V (2b)	N (5b)	P (11b)
--------	--------	--------	--------	---------

1. El campo X representa el valor de la coordenada **X**
2. El campo Y representa el valor de la coordenada **Y**
3. El campo V representa la cantidad de **vidas** que posee
4. El campo N representa el **nivel** actual
5. El campo P representa la cantidad de **puntos** que acumuló.

Esta información, en ciertas ocasiones deberá estar comprimida en hexadecimal, en cuyo caso la denominaremos con un nuevo formato que se llama **HexaStatus**.

A continuación se solicita responder las siguientes preguntas, justificando debidamente sus respuestas:

- (a) ¿Cuál es la dimensión del tablero?
- (b) En base a la dimensión mencionada en el ítem anterior, si se quisiera duplicar el tamaño del mismo ¿qué se debería modificar en el sistema de representación? y ¿cómo impacta dicha modificación en el formato general del juego?
- (c) ¿Cuál es la cantidad máxima de vidas que se puede tener?
- (d) Representar el **estado inicial** del pacman, en formato **PacStatus**. El juego inicia con el Pacman ubicado en el centro del tablero, en el primer nivel, sin puntos acumulados y con la máxima cantidad de vidas.
- (e) Comprimir la cadena anterior en formato **HexaStatus**.
- (f) Dar 2 ejemplos de estado del juego en formato **PacStatus** que representen 2 posibles jugadas. Justificar explicando qué valor representa cada campo.

1.2. Aritmética: Puntajes

Teniendo en cuenta el dominio del juego, responder las siguientes preguntas en base a las consideraciones planteadas en cada caso:

1. Suponiendo que en este momento, el estado del pacman en formato hexaStatus es 3A831D55, ¿cómo queda el hexaStatus luego de descontar una vida?
2. Suponiendo que en este momento, el estado del pacman en formato hexaStatus es 3A8217FF, y el pacman come otra moneda ¿cómo queda el hexaStatus luego de incrementar el puntaje?

2. Trabajo Práctico (obligatorio)

A continuación se especifican los ejercicios obligatorios a entregar para acreditar la materia. Será sumamente importante el entendimiento del dominio del juego, propiciado por la ejecución de las actividades del precalentamiento.

2.1. Programación y ensamblado: Inicialización del juego

El estado de los 4 fantasmas se representa en los 4 registros R0, R1, R2 y R3 respectivamente, con el formato `GhostStatus: xxxxxxxxyyyyyy00`, donde los bits `xxxxxxx` representan la coordenada **X** y los bits `yyyyyy` representan la coordenada **Y** de cada fantasma. El estado del juego se almacena en los registros R5 y R6.

1. Escribir en Q1 el **programa** `init` que inicialice el estado del juego teniendo en cuenta la representación del inciso d) del ejercicio 1.1, y la posición de los 4 fantasmas, ubicándolos en cada una de las esquinas del tablero.
2. Ensamblar el programa.

2.2. Desensamblar: Programa Init

Desensamblar el programa que se encuentra ensamblado en la siguiente porción de memoria.

	...
5003	1940
5004	7CFB
5005	1980
5006	0800
5007	1800
5008	0000
5009	1840
500A	01FC
500B	1880
500C	FE00
500D	18C0
500E	FFFC
500F	C000
	...
FFEF	0500
	...

2.3. Análisis de la ejecución de un programa: Init

Para comprender lo que ocurre en cada etapa de ejecución de un programa, se realiza una **simulación** del mismo, que es una ejecución paso a paso (un ejemplo de cómo estructurar esta información se encuentra en la sección 7.3 del libro, revisar su formato como guía).

Por lo cual se pide lo siguiente:

1. Simular la ejecución del programa descrito en el estado de memoria del ejercicio anterior, indicando cómo se va modificando cada registro y la memoria en cada etapa. Para esto asumir que el valor original del registro **SP** es `FFEE` y de **PC** es `5003`.
2. En base a la simulación realizada recientemente, responder y justificar las siguientes preguntas:
 - a) ¿Cuál es el valor de los registros de **uso general** (visibles a quien programa) al finalizar la ejecución de las instrucciones anteriores?

- b) ¿Cuál es el valor de los registros de **uso específico** (no visibles a quien programa) al finalizar la simulación de ejecución?
- c) ¿En qué momento se actualiza el registro SP y con qué objetivo?
- d) ¿Qué ocurre con la celda de memoria FFEF al finalizar la simulación en esta ejecución particular?
- e) ¿Es posible saber en qué celda esta la instrucción siguiente a la última simulada? ¿cómo?
- f) ¿Cuántos elementos tiene la pila **antes de comenzar** la simulación?
- g) ¿Cuántos elementos tiene la pila **luego de finalizar** la simulación?
- h) Describir los accesos a memoria sobre el ciclo de ejecución de la última instrucción del programa, a partir del estado de los buses de cada acceso.

2.4. Documentación: Programa Init

1. Documentar la rutina `init` (ejercicio 2.1)
2. Responder las siguientes preguntas:
 - a) ¿La rutina necesita parámetros?
 - b) De manera general ¿cómo se relaciona el campo “requiere” con la presencia/ausencia de parámetros?

2.5. Programación: Desplazamiento por el tablero

Para codificar los movimientos en el tablero, se utiliza una codificación de 16 bits, denominada **MoveCode**, la cual cuenta con el siguiente formato: 00000000000000AH, donde:

- **A**: indica si el Pacman debe **avanzar o retroceder**. Siendo A=1 avanzar, y A=0 retroceder.
- **H**: indica si el Pacman debe moverse de manera **horizontal o vertical**. Siendo H=1 un movimiento horizontal, y H=0 un movimiento vertical.

Escribir la rutina `checkAndMove`, para cumplir con el objetivo descrito en la siguiente documentación:

checkAndMove	
Requiere	en R5 y R6, el estado actual del pacman, en formato PacStatus , y en R3 el movimiento a realizar, en formato MoveCode
Modifica	COMPLETAR
Retorna	en R5 y R6, el estado del pacman luego del movimiento, en formato PacStatus . Ya sea con la posición actualizada, o con la posición actual, si dicho movimiento no pudo ser posible.

Información necesaria:

Para programar la rutina solicitada, **se cuenta con una serie de rutinas**: `nextToAWall`, `getCoord` y `movePacman`, las cuales respetan las siguientes documentaciones:

nextToAWall	
Requiere	en R1 y R2 las coordenada X e Y respectivamente, ambas en BSS(7), y en R3 un movimiento en formato MoveCode
Modifica	[A000]
Retorna	en R3 un 1, si el casillero adyacente a (X,Y) está libre, y un 0 si hay un muro.

getCoord	
Requiere	en R5 y R6 el estado actual del pacman, en formato PacStatus
Modifica	[A000]
Retorna	en R1 y R2 las coordenada X e Y respectivamente, del estado actual del pacman. Ambas en BSS(7)
movePacman	
Requiere	en R5 y R6 el estado actual del pacman, en formato PacStatus , y en R3 un movimiento en formato MoveCode
Modifica	R0
Retorna	en R5 y R6, el estado del pacman, en formato PacStatus , actualizado con la nueva posición en base al movimiento solicitado

2.6. Rutinas de Test: contenido del tablero

El contenido de un **casillero del tablero** se representa en BSS(16), con el siguiente formato: 000000000000CBWL.

Donde:

- El bit c (Coin) indica si el casillero contiene una moneda que permite acumular 10 puntos
- El bit B (Bonus) indica si el casillero contiene un premio especial que permite acumular 100 puntos
- El bit w (Wall) indica si en el casillero se encuentra un muro.
- El bit L (Limit) indica si se trata de un casillero próximo al límite del tablero

A) Programación de Tests

Nos han encargado la tarea de probar la rutina **getPrize** (ya programada), cuya documentación es la siguiente:

getPrize	
Requiere	en R5 y R6 el estado actual del pacman en formato PacStatus , y en R4 el contenido de un casillero del tablero
Modifica	la celda A000
Retorna	en R5 y R6 el estado del pacman, en formato PacStatus , con los datos actualizados en base al contenido del casillero

Para ello será necesario escribir las **rutinas de test** de los siguientes casos de prueba:

1. Rutina **testGetCoinPrize** para el caso de un casillero que contiene sólo una moneda (sin bonus).
2. Rutina **testNoPrize** para el caso de un casillero vacío
3. Rutina **testGetBothPrizes** para el caso donde:
 - El casillero contiene una moneda y un bonus
 - El Pacman posee 1 vida
 - El puntaje actual es de 1950 puntos

Recordar plantear el escenario completo, datos iniciales y el resultado esperado, para cada caso.

B) Ensamblado de tests

Ensamblar, a partir de la celda 6001, la rutina **testGetCoinPrize** programada previamente, asumiendo que la rutina **getPrize** está ensamblada a partir de la celda 0A0A.

2.7. Máscaras: Información del PacStatus

Escribir la rutina `getPacStatusInfo` en base a la siguiente documentación, actualizando el campo **Modifica** según corresponda:

getPacStatusInfo	
Requiere	en R5 y R6 el estado actual del pacman, en formato PacStatus
Modifica	COMPLETAR
Retorna	desde la celda de memoria 0xA123 hasta la celda 0xA127, cada uno de los datos que conforman el PacStatus, sin alterar su valor, y un dato por celda respetando el orden de los campos del formato.

Aclaración: cuando se menciona “sin alterar su valor”, significa que si se interpreta la cadena de cada campo se deberá obtener el valor original. Por ejemplo, si el PacStatus tiene 2 vidas, la celda 0xA125 deberá almacenar el valor 2. Revisar bien las cadenas resultantes antes de almacenarlas en las celdas correspondientes, por si es necesario aplicar alguna operación previa.

2.8. Modularización(reuso): Mayor puntaje

Escribir la rutina `MaxScore` en base a la siguiente documentación, actualizando el campo **Modifica** según corresponda:

MaxScore	
Requiere	desde la dirección de memoria 0x0000 hasta la dirección 0x0003, 2 estados de pacman, ambos en formato PacStatus
Modifica	COMPLETAR
Retorna	en R5 y R6 el estado de pacman, en formato PacStatus , con mayor puntaje. En caso coincidir el puntaje, se retorna el primer PacStatus.

2.9. Arreglos: Pacman ganador

Se cuenta con un listado de estados de pacman, en formato **PacStatus**, que representan las últimas jugadas ordenadas de manera cronológica.

Se quiere obtener el estado de pacman (PacStatus) ganador del juego, es decir aquel con mayor puntaje acumulado.

Para ello, se necesita **programar la rutina `pacmanWinner`, con su respectiva documentación**, que determine el PacStatus ganador, el cual deberá quedar almacenado entre las celdas 0x5563 y 0x5564.

Tener en cuenta que el listado comienza a partir de la celda con dirección 0x5566, y su longitud se encuentra en la celda con dirección 0xFA56.

2.10. Sistemas de números fraccionarios: Pacman recargado

En esta oportunidad, a modo de experimento, se desea incorporar al formato del Pacman, un nuevo campo de aceleración (A) representado en un sistema de numeración BSS(2,2). El objetivo es reflejar una alteración de la velocidad cada vez que se come un bonus o lo alcanza un fantasma. Para ello, se cuenta con la rutina **AcelerarPacman** que respeta la siguiente documentación:

AcelerarPacman	
Requiere	en R5 y R6 el pacStatus, y en R4 un flag (1 o 0), que representa el cambio de velocidad (incremento o disminución).
Modifica	–
Retorna	en R5 y R6 el nuevo pacstatus modificado, donde el campo A ha sido incrementado si R4 es igual a 1, o disminuido si es igual a 0. Si el campo A se encuentra en su valor máximo y R4 es 1, el campo A no se modifica. Dualmente, si A se encuentra en su valor mínimo y R4 es 0, tampoco hay una modificación.

Dado el dominio detallado, se necesita resolver los siguientes puntos:

1. Determinar el rango y la resolución del sistema de numeración que representa el campo A
2. Escribir el test **testAcelerarPacman** para probar la rutina “AcelerarPacman”, para el caso que el pacman alcance un bonus, sin tener el campo “A” en su valor máximo. Plantear detalladamente el escenario inicial y final antes de escribir el test
3. Indicar qué ventajas y desventajas se encuentran, si se solicitara cambiar el sistema de numeración que representa el campo “A”, por un **sistema de punto flotante**. Es válido apoyarse en ejemplos que utilicen más bits que los destinados actualmente (2 bits)

2.11. Memoria Caché: Init

Se cuenta con la siguiente arquitectura de caché:

- Cantidad de líneas: 2
- Tamaño de bloque de 4 celdas
- Función de correspondencia: asociativa
- Política de reemplazo: FIFO

Teniendo en cuenta la arquitectura detallada:

1. Indicar cuantos bits se le asignan, respectivamente, al tag y la palabra de las direcciones en la memoria cache detallada anteriormente
2. Considerando la ejecución simulada en la sección 2.3), detallar la información almacenada en la memoria caché al ejecutarse el programa Init. Se recomienda estructurar la información, mediante la siguiente tabla, indicando los aciertos y fallos que se suceden por cada instrucción ejecutada:

Dirección	Tag	Línea	Palabra	F/A	Etapa

3. Analizar la tabla del punto anterior, y explicar detalladamente, a modo de ejemplo, un caso de acierto y uno de fallo, indicando cómo se relaciona la información en cada caso.