Proyecto Pac-Man en MARIE

Ingeniería en Ciencias de la Computación
Universidad San Francisco de Quito (USFQ)
Gabriel Avalos, David Bucheli, Jhonatan Quiroga

***Abstract*—Este trabajo detalla la implementación del juego Pac-Man sobre el simulador MARIE: generación aleatoria de movimientos, manejo de colisiones, estado de juego (puntuación, vidas) y reinicios automáticos. Se describen la arquitectura de memoria, pseudocódigo de la lógica principal, resultados de ejecución y conclusiones.**

***Index Terms*—Pac-Man, MARIE, ensamblador didáctico, simulador educativo**

I. Introducción

El aprendizaje de la arquitectura de computadoras se ve potenciado mediante prácticas que relacionan conceptos de hardware y software. MARIE (Machine Architecture that is Really Intuitive and Easy) ofrece un conjunto reducido de instrucciones y memoria accesible para experimentos didácticos [1]. Bajo esta plataforma, se implementó Pac-Man para ilustrar:

- Gestión y representación de datos en memoria como un display bidimensional.
- Control de flujo con saltos y subrutinas.
- Manejo de variables globales para estado del juego.
- Reutilización de funciones y manejo de lógica.

II. Metodología

La implementación se organizó en las siguientes fases:

A. Representación en Memoria

MARIE tiene una memoria de 4096 bits, de ellos, 256 bits son ocupados para el display (16 px de largo, 16 px de ancho).

- `OFFSET=0xF00`: inicio de la zona de display.
- Cada celda en `OFFSET + row*DISPLAY_SIZE + column` contiene un valor hexadecimal que modifica el color.
- Mapas de muros, monedas y esteroides definidos en secciones etiquetadas (`MAPA`, `BOLITA_COLOR`, `ESTEROIDE_COLOR`).

B. Generación de Movimientos

En esta fase, los movimientos se generaron mediante scripts en Python antes de integrarlos al ensamblador MARIE:

- **pacMovimientos**: se crea un arreglo de 100 direcciones (1=arriba, 2=abajo, 3=izquierda, 4=derecha) con esta lógica:
  - Se evita el giro de 180° inmediato descartando la dirección opuesta a la anterior.
  - Se favorece que Pac-Man siga en la misma dirección, ponderando esa opción varias veces.
  - A partir de esta lista ponderada se elige la siguiente dirección al azar, logrando recorridos fluidos pero variados.
- **ghostMovimientos**: se genera un arreglo de 100 direcciones puras con distribución uniforme (cada paso es un entero aleatorio entre 1 y 4), garantizando un comportamiento impredecible de los fantasmas.

C. Lógica de Colisiones y Estado

Subrutinas principales:

- `isPared()`: previene que Pac-Man o fantasmas crucen muros.
- `touchPacman()`: detecta choques de fantasmas con Pac-Man; si ocurre, se invoca `HandleLifeLoss()` o a `getScore()` en caso de que el fantasma pueda ser comido.
- `getScore()`: incrementa `score` al comer monedas (+1) o fantasmas (+10).
- `HandleLifeLoss()`: decrementa `vidas`, reinicia posiciones si quedan vidas; si `vidas==0`, va a `FINAL`.
- Esteroides: al entrar en una casilla de color `ESTEROIDE_COLOR`, se activa modo agresivo por `STEP_LIMIT` pasos, modificable.

D. Lógica de Pac-Man (*pacmanLogic*)

La rutina `pacmanLogic()` gestiona el avance de Pac-Man tras validar:

- Llamada a `isPared()` para evitar muros.
- Detección de esteroides con `isEsteroide()`, activando modo agresivo.
- Actualización de puntuación con `getScore()` (bolitas y fantasmas).
- Verificación de colisión con fantasmas en `GameOver`.
- Limpieza de pantalla previa con `moverse()`.

E. Lógica de pérdida de vida (*HandleLifeLoss & GameOver*)

La etiqueta `GameOver` detecta contacto con fantasmas y salta a `HandleLifeLoss()`, que:

- Decrementa el contador `vidas`.
- Si `vidas>0`, invoca `ResetPositions()` para reiniciar posiciones sin perder el `score`.
- Si `vidas==0`, salta a `FINAL` y termina el juego mostrando el puntaje.

F. Restablecimiento de posiciones (`ResetPositions`)

La subrutina `ResetPositions()`:

- Borra del display los sprites actuales con color de fondo.
- Reubica a Pac-Man y a cada fantasma en sus coordenadas iniciales.
- Restaura colores originales de los fantasmas.
- Redibuja personajes con ayuda de `getHex()`.
- Retoma el flujo saltando a la rutina principal `PACMAN`.

G. Método de atravesar (`atravesar`)

La subrutina `atravesar()`, llamada desde `ghostLogic()`, permite a los fantasmas "atravesar" la celda previa:

- Detecta el fantasma actual (`id`) y guarda su color anterior en la variable correspondiente.
- Restaura ese color en la posición vieja con `StoreI`, de esta forma los fantasmas no eliminan monedas ni esteroides.
- Avanza la posición y redibuja al fantasma con su color respectivo.
- Permite continuar el bucle de `ghostLogic()` sin interrupciones.

H. Pseudocódigo

Durante la codificación en MARIE, se simuló la lógica del display (16×16 posiciones) y de las subrutinas clave en Excel, creando fórmulas para:

- Emular la secuencia de movimiento de los fantasmas y del Pac-Man.
- Verificar la ejecución deseada de las funciones, para evitar duplicados o desapariciones de los objetos.
- Analizar el estado de monedas y esteroides cuando algún objeto pasaba sobre ellos.

Gracias a ello, se facilitó la detección de inconsistencias y la correcta ejecución de las secuencias antes de implementar en MARIE.

Como parte del algoritmo funcional, utilizando las funciones y procesos propuestos en MARIE, se concluye el siguiente pseudocódigo.

```
# Variables iniciales
vidas <- 3
score <- 0
consumibles <- 0
esteroideActivo <- false
pasosEsteroide <- 0
ptrPac <- 0
ptrGhost <- [0,0,0,0]

# 1. Saltar al mapa y entrar al bucle principal
Jump MAPA

INICIO:
  # 2. Generar siguiente movimiento
  Call getMovPac()          # actualiza ptrPac, counter
  Call getMovGhost()        # actualiza ptrGhost[i], counter2
  Call INICIO               # determina direcci n y salta a UP/DOWN/LEFT/RIGHT

UP/DOWN/LEFT/RIGHT:
  Call getHex()             # calcula currentPosition y nextPosition
  Load id
  Subt idPacman
  SkipCond 0C00
    Call pacmanLogic()      # isPared, isEsteroide, getScore, GameOver, moverse
  Else
    Call ghostLogic()       # isPared(), touchPacman()
    If (nextPosition == BOLITA_COLOR or nextPosition == ESTEROIDE_COLOR or nextPosition == BLACK)
      Call atravesar()
    ElseIf nextPosition == ANY_GHOST_COLOR
      If prevColor_ghost != BLACK
        moverseFalso()
      ELSE
        moverse()
      EndIf
    EndIf
  EndIf
  Call increaseStepCounter()  # actualiza modo esteroide
  Call checkTurn()            # guarda posiciones previas
  Jump INICIO

# pacmanLogic: valida colisiones y estado
pacmanLogic():
  Call isPared()            # evita muros
  if (celdaActual == ESTEROIDE_COLOR) then
    Call isEsteroide()      # cambia fantasmas al estado comible
  end if
  if (esteroideActivo == true) then
    stepCounter <- 0          # reinicia contador
    Call goEsteroide()        # fantasmas pasan a comibles
  end if
  if (esteroideActivo) then
    stepCounter <- stepCounter + 1
    if (stepCounter >= STEP_LIMIT) then
      esteroideActivo <- false
      Call increaseStepCounter()    #reestablecer color fantasmas
    end if
  end if
  Call getScore()           # setScoreBolita o setScoreFantasma + ResetPositions
  Call GameOver()           # HandleLifeLoss si choca con fantasma
  Call moverse()          # limpia pixel previo
  Return

# ghostLogic: controla fantasmas
ghostLogic():
  Call isPared()
  Call touchPacman()        # decrementa vidas, ResetPositions o FINAL
  Call atravesar()          # si se encuentra al frente una moneda, un esteroide o un espacio vacio
  Call moverse()          # si se encuentra un fantasma al frente
  Call moverseFalso()    # si tiene algo en prevColor
  Return

# getScore: actualiza score y consumibles
getScore():
  If nextPosition == BOLITA_COLOR then
    Call setScoreBolita()    # +1 punto
  ElseIf nextPosition == FANTASMACOMIDO_COLOR then
```

```
    Call setScoreFantasma() # +10 puntos
    Call resetFantasma()    # el fantasma comido
        vuelve a su posicion original
  EndIf
  Return

# GameOver y HandleLifeLoss
GameOver():
  Call HandleLifeLoss()     # vidas--,
      ResetPositions o FINAL

HandleLifeLoss():
  vidas <- vidas - 1
  If vidas > 0 then
    Call ResetPositions()
  Else
    Jump FINAL              # termina ejecuci n
  EndIf

# ResetPositions: reposicionar todo
ResetPositions():
  # Restablece pacRow, pacColumn a sus valores
      originales. Restablece cada a fantasma su fila
      /columna inicial. Restaura los colores por
      defecto.
  Jump PACMAN               # retoma flujo

# FIN DEL JUEGO
FINAL:
  Output score, consumibles
  Halt
```

III. Resultados

Se verificaron los siguientes escenarios:

- Consumir todas las monedas: termina con el programa y score igual al número de monedas.
- Perder tres vidas por choques sucesivos: ”Game Over” y score acumulado hasta el último choque.
- Activar esteroides y comer fantasmas: puntaje aumenta en bloques de 10.
- Reinicio de posiciones validado al perder vida, sin reiniciar score.

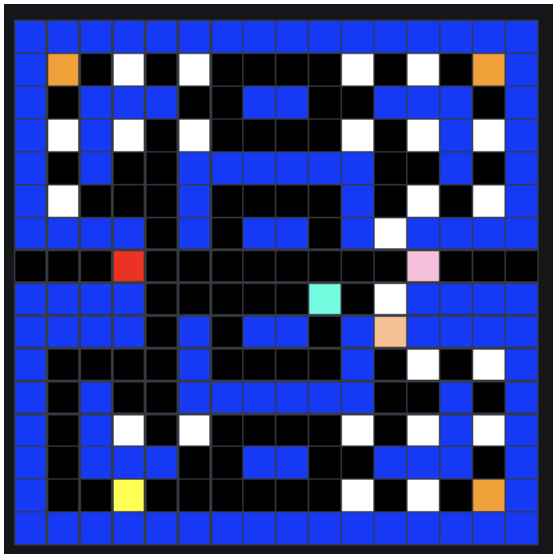

Fig. 1. Display en funcionamiento una vez ejecutado el código

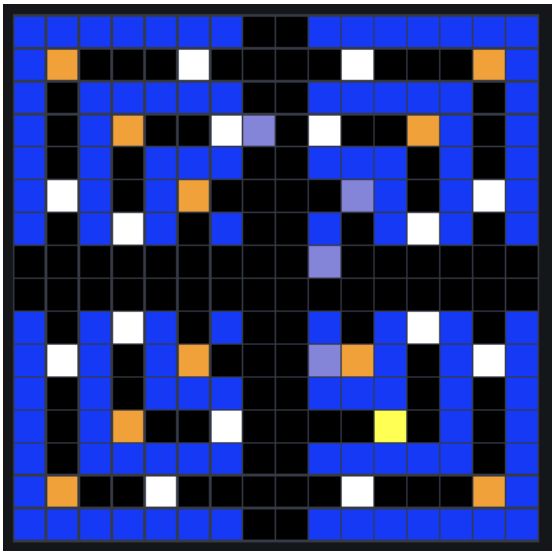

Fig. 2. Display cuando Pac-Man come un esteroide

- Consumir todas las monedas: termina con ”Victoria” y score igual al número de monedas.
- Perder tres vidas por choques sucesivos: ”Game Over” y score acumulado hasta el último choque.
- Activar esteroides y comer fantasmas: puntaje aumenta en bloques de 10.
- Reinicio de posiciones validado al perder vida, sin reiniciar score.

IV. Conclusiones

El proyecto de implementar PacMan en el simulador MARIE ha permitido profundizar en aspectos clave de la arquitectura de computadores: el control de flujo mediante saltos y subrutinas, el manejo de memoria para representar un display bidimensional y la gestión de estado a través de variables globales (puntaje, vidas, contador de pasos). La construcción iterativa de cada subrutina —desde la generación de movimientos aleatorios hasta la lógica de colisiones y el restablecimiento de posiciones— reforzó la comprensión de cómo un conjunto limitado de instrucciones puede orquestar un sistema interactivo completo.

Estas experiencias no solo cumplen con los objetivos académicos, sino que también sientan las bases para futuras extensiones, tales como:

1) La incorporación de heurísticas avanzadas para la IA de los fantasmas.
2) La generación procedural de laberintos.
3) La creación de interfaces gráficas más sofisticadas fuera del entorno de MARIE.

References

[1] M. J. Patel y Y. N. Patt, *Introduction to Computing Systems: From Bits & Gates to C & Beyond*, 3ª ed. McGraw-Hill, 2013.

[2] L. Null y J. Lobur, *The Essentials of Computer Organization and Architecture*, 3ª ed. Jones & Bartlett, 2020.

[3] J. Doe, ”MARIE Simulator User Manual,” Univ. Educational Press, 2010.