Writing a Snake game

in AQA Assembly Language

TEACHER’S GUIDE

Worksheet authored by Richard Pawson, Stowe School

[Creative Commons License](https://creativecommons.org/licenses/by-sa/4.0/)  
This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

# Introduction

# Story 3

## Complete code after Story 3

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

add r4,r4,#1 //Increment the head pointer by 1

str r1,[r4+256] //Draw new head

b moveSnake //Loop

# Story 4:

## Complete code after Story 4

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

add r4,r4,#1 //Increment the head pointer by 1

str r1,[r4+256] //Draw new head

b moveSnake //Loop

# Story 5

## Complete code after Story 5

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

str r2,[r3+256] //Reset tail to Background

add r3,r3,#1 //Increment the tail pointer by 1

moveHead:

str r1,[r4+256] //Draw new head

## b moveSnake //Loop

# Story 6

Note: At the end of this story the code should be reverted to its state at the end of Story 5.

# Story 7

## Complete code after Story 7

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

moveHead:

add r8,r8,#1 //Increment the head pointer

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Story 8

## Complete code after Story 8

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

add r4,r4,#1 //Increment the head location by 1

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Story 9

## Complete code after Story 9

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

cmp r9,#83 //S key

beq down

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Story 10

## Complete code after Story 10

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #520 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Story 11

## Complete code after Story 11

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

b moveSnake //Loop

body: dat 0 //Initial front of queue (screen address for tail)

# Story 12

## Complete code after Story 12

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

body: dat 0 //Initial front of queue (screen address for tail)

# Story 13

## Complete code after Story 13

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

add r4,r4,#1 //r9 holds the increment for the position...

b reDraw // Unconditional branch

down:

add r4,r4,#32 //...32 moves down one row on screen

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

b reDraw

left:

sub r4,r4,#1 //-1 moves left

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

body: dat 0 //Initial front of queue (screen address for tail)

# Story 14

## Complete code after Story 14

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Story 15

## Complete code after Story 15

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Story 16

## Complete code after Story 16

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

checkForCrossing: // check if the snake has hit itself

ldr r0,[r4+256] // read, from screen, colour of pixel head is moving to

cmp r0,r1 //If it is snake colour …

beq gameOver

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake //Otherwise just repeat the move cycle

gameOver:

halt //To stop program execution running into data area

body: dat 0 //Initial front of queue (screen address for tail)

# Story 17

## Complete code after Story 17

defineRegisters:

mov r1,#0x008844 //Snake colour (green)

mov r2,#0xffffff //Background colour (white)

mov r3, #271 //Tail position, initialised

mov r4, #272 //Head position, initialised

mov r5, #0 //Apple position

mov r6, #0xff8800 //Apple colour

mov r7, #body //Pointer front of queue, initialised to first data loc

add r8,r7,#1 //Pointer to head address in body data (1 after tail)

mov r9, #0 //ASCII value of last key pressed

mov r10, #767 //Constant representing the size of screen memory

mov r11, #1023 //Constant

mov r12, #68 //Current direction of movement, initialised to 'right'

InitialisePointers:

str r3, [r7] //r4 points to the tail address

str r4, [r8] //r3 points to the head address

drawSnake:

str r1,[r3+256] //Tail

str r1,[r4+256] //Head

createApple:

inp r5,8 // gets a random 32 bit pattern

and r5,r5,r11 // r11 has 1023

cmp r5,r10 // r10 has 767

bgt createApple // restrict random range

cmp r5,r4 // Make sure apples is not located on head of snake

beq createApple

moveSnake:

str r6,[r5+256] //Draw Apple each cycle, in case it is on snake

inp r9,4 //Read the last key pressed (but don’t wait for one)

switchOnKey:

cmp r9,#87 //W key

beq up

cmp r9,#65 //A key

beq left

cmp r9,#83 //S key

beq down

cmp r9,#68 //D key

beq right

dontChangeDirection:

mov r9, r12 //If not any of the recognised keys, use prev direction

b switchOnKey //and re-run the switch on key

right:

cmp r12,#65

beq dontChangeDirection

add r4,r4,#1 //Adding 1 to location moves right

and r0,r4,#31

cmp r0,#0

beq gameOver

b reDraw

down:

cmp r12,#87

beq dontChangeDirection

add r4,r4,#32 //32 moves down one row on screen

cmp r4,r10

bgt gameOver

b reDraw

up:

cmp r12,#83

beq dontChangeDirection

sub r4,r4,#32 //-32 moves up one row on screen

cmp r4,#0

blt gameOver

b reDraw

left:

cmp r12,#68

beq dontChangeDirection

sub r4,r4,#1 //-1 moves left

and r0,r4,#31

cmp r0,#31

beq gameOver

b reDraw

reDraw:

mov r12,r9 //Update current direction with latest key

cmp r4,r5 //If the head is in same location as apple...

beq moveHead //...Skip updating the tail, to make snake grow

moveTail:

ldr r0, [r7]

str r2,[r0+256] //Reset tail to Background

add r7,r7,#1 //Increment the tail pointer (for use next cycle)

cmp r7,#200 //Check pointer is still within memory

blt moveHead

mov r7, #body //If not loop pointer back to start of body data

moveHead:

add r8,r8,#1 //Increment the head pointer

cmp r8,#200 //Check pointer is still within memory

blt updatePointer

mov r8, #body //If not loop pointer back to start of body data

updatePointer:

str r4, [r8] //Store the new head location in data

checkForCrossing: // check if the snake has hit itself

ldr r0,[r4+256] // read, from screen, colour of pixel head is moving to

cmp r0,r1 //If it is snake colour …

beq gameOver

checkForMaxLength:

cmp r7,r8 //If the front has caught up with rear, then body data full

beq gameWin

str r1,[r4+256] //Draw new head

cmp r4, r5 //Check again if the apple was eaten this cycle

beq createApple //If so, loop back to creating the apple

b moveSnake /Otherwise just repeat the move cycle

gameWin:

mov r0, #message

out r0,8 //Write message as text into output window

gameOver:

halt //To stop program execution running into data area

message: //ASCII encoding of 'You win' message (in reverse)

dat 0x20756f59 //' uoY'

dat 0x006e6977 //'niw'

body: dat 0 //Initial front of queue (screen address for tail)