COMP2611: Computer Organization

Fall 2014

Programming Project: The Revenge of Pigs

Due Date: 28th Nov, 2014, 17:00

Introduction

Following the great victory of the birds at the “Piggy Island”, the Red Bird starts its quest to search for the precious Sluethwing eggs stolen by the pigs. Upon reaching the seashore, the Red Bird discovers a stone forest called “Azbadan”, where it is ambushed by an army of green piglets. Humiliated by the bitter lose at the “Piggy Island”, the piglets are determined to capture the Red Bird. They believe they could do so by a sudden ambush… The showdown of the piglets and the Red Bird will take place in the game episode, “The Revenge of the Pigs”, and you are going to complete the implementation of that game episode!

Game Objective

The objective of the game to keep the “Red Bird” safe from the army of green piglets. Player can score points by killing the green piglets. Green piglets could be killed in three ways:

1. When two or more green piglets collide with one another (become a skull),

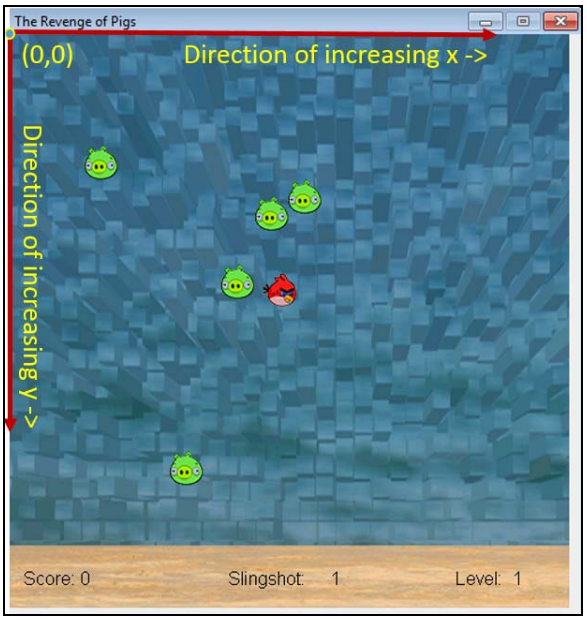
2. When the green piglet walks onto a piglet skull,

3. When the Red Bird uses the “sonic slingshot” and the piglets are adjacent to the bird.

Note: the green piglets will try to move closer to the Red Bird in each iteration, when they move, they move blindly without taking into account of the positions of other piglets and skulls.

Game objects

The game is played on a screen with a size of 510-pixel by 450-pixel (width by height). The X-coordinate value increases from the left to the right and the Y-coordinate value increases from the top to the bottom (i.e. the origin (0,0) is at the upper left corner of the screen), as shown in the figure below:



Each game object: the Red Bird, the Green Piglets, the Piglet Skull are square images. The locations of an object is specified by its (X,Y) coordinates. The (X,Y) coordinates of an object indicate the position of the upper-left corner pixel of the image. The sizes of the objects are shown in the table below:

Object	File name	Image width (in pixels)	Image height (in pixels)
Background	Background.gif	510	450+55 (lower 55 pixels for displaying game information only)
Red Bird1*	bird1.gif	30	30
Red Bird2*	bird2.gif	30	30
Green Piglet	pig.gif	30	30
Pig Skull	skull.gif	30	30

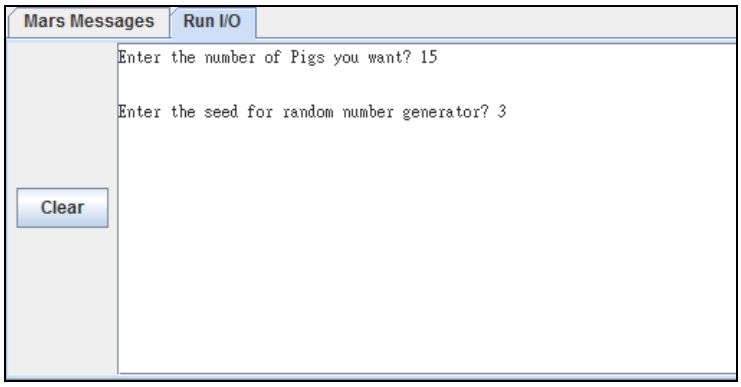
* bird1.gif and bird2.gif are different in posture and are use to generate the movement animation for the Red Bird.

Read the "Game Flow" part for the details.

Note that the coordinates of the game screen and the images are zero-based. (e.g., the upper-left pixel of the screen is (0,0) and the lower-right pixel of the screen is (509, 449).

Game flow

The player enters the initial number of piglets at the start of the game, then the player enters an integer seed to initalize the random number generator (shown in the figure below).



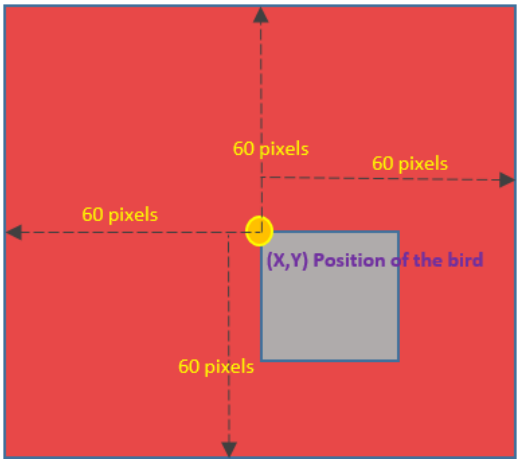
The game would then start at level 0. The Red Bird is positioned in the center

of the rectangular game screen (at (240,210)). A number (according to the user input) of piglets are positioned randomly on the screen. The piglets are controlled by the computer and they will try to move closer to the Red Bird in each iteration of the game (see the paragraph below for the details). The movement of the Red Bird on the other hand is controlled by the numpad of the keyboard:

a) Move the Red Bird one step (2 pixels) in a particular direction ("8" for up, "2" for down, "4" for left, "6" for right, "7" for upper left, "9" for upper right, "3" for lower right, "1" for lower left). The "bird1.gif" and "bird2.gif" pictures should be used alternatively for the keystrokes to generate a movement effect.

b) Keep the Red Bird in the same position ("5").

c) Use the "sonic slingshot" ("r") to kill all the piglets adjacent to the Red Bird, the Red Bird can give only one slingshot in each level. The "kill radius" of the sonic slingshot is 60 pixels as illustrated in the figure below, any piglet overlays with the red square (including the grey square) below will be killed.



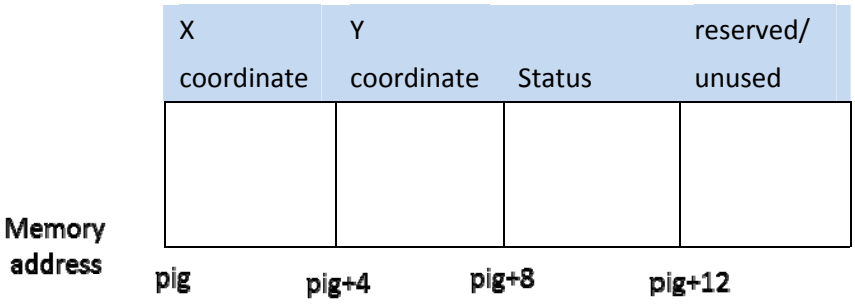
d) Teleport ("t") the Red Bird to a random position on the screen, there is no guarantee that the Red Bird would not collide with a piglet (in that case it will be captured).

Note that the Red Bird is not allowed to move out of the boundary of the screen, you will need to check and enforce that in the program.

Each of the piglets will move one step in the direction towards the Red Bird in each iteration of the game. If two or more piglets move to the same position (i.e (X,Y) coordinates identical), they will be killed and turn into a piglet skull in the colliding position. Any other piglet that walks onto the skull will also be killed. The level ends when either all the piglets are annihilated or the Red Bird is captured by one or more piglets. If all the piglets are annihilated, the game will move one level up, and the number of piglets will be doubled (max 99 piglets for all levels). If the Red Bird is captured, the game ends with the message "Red bird was captured!".

Game data structure

The "pig" array defined in the ".data" segment is an array holding the statuses of all the piglets. The array starts from the address pointed to by the label "pig" (exact memory address value could be loaded to a register using the load address, "la", instruction). Each piglet takes 4 words to store its status information. The first word is its "X-coordinate", the second word is its "Y-coordinate" and the third word is its current state (0 for healthy, 1 for skulled, 2 for killed by slingshot and thus will not be displayed). The fourth word is reserved for future extensions. The following picture depicted the 4-word data structure for storing the status of the first piglet. The 4-word status data structure of the 2nd piglet will start from the memory address "pig+16". In general the 4-word data structure of the nth piglet starts at address "pig+16*(n-1)".



The status of the bird is stored in the "bird" array. The first word of the "bird" array stores the current "X-coordinate" of the bird, the second word of the "bird" array stores the current "Y-coordinate" of the bird. There is no need to store the state of the bird (why? because the Red Bird only has two states: "normal" or "captured". When you are playing the game, the Red Bird is always in the "normal" state. Whenever the bird is captured the code will figure it out, and the game will end immediately).

Game scoring

Each piglet that gets killed by colliding with another piglet will increase the player score by 10 (ie. If three piglets collide, the player score will be incremented by 30). Each piglet killed by colliding with a skull will also increase the player score by 10. Piglets killed by the "sonic slingshot" will not increase the player score.

Your tasks

You must complete the game using the MIPS assembly language. You are given a custom-made MARS program (can be downloaded from the course web) and a skeleton program angrybird.s for the game. Please make sure you use the correct version of MARS. The skeleton program contains special syscall operations (syscall 100) that are not taught in the course. You do not need to understand all of them or use all of them in your codes. The complete list of services offered by the syscall 100 are listed at the end for your reference. Feel free to pick the ones you would like to use. You must read the skeleton program in detail in order to understand how to implement the game. Many of the functions have been implemented for you and you are only required to complete the functions mentioned below. To ensure the correctness, read the skeleton program carefully before you start.

Function	**Task**
setGameoverOutput	When the Red Bird is captured, set the gameover string. You need to 1.) set the string (using syscall 100 and setting $a0=13), 2.) set the object location (syscall 100, $a0=12), 3.) define string font (syscall 100, $a0=16), 4.) setting the string color (syscall 100, $a0=15).

bird_moves	Moves the Red Bird according to user inputs, possible inputs are “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “t”, and “r”, and carry out the appropriate actions. Make sure you check whether the Red Bird has reached the boundary of the screen.
update_state	update the internal game states like score, etc. Also check any new skull and the status of the Red Bird.
is_lv_up	check if the game needs to proceed to a new level Return value in $v0: 0 -- false, 1 -- true

MARS with syscall 100

Make sure you download the MARS 4.4 with syscall 100 from our course web at: https://course.cse.ust.hk/comp2611/Password_Only/Mars4_4_withSyscall100.jar

Put the MARS program, the MIPS assembly program (angrybird.s) and all the pictures (i.e. background.gif, bird1.gif, etc) into the same directory, otherwise the assembly program may not run correctly.

The details of syscall 100 are enclosed at the end of this document. Read them so that you would how to handle and display the text objects/pictures properly.

Grading

Test the program well to make sure it runs under MARS. **NO MARK** will be given if the program cannot be assembled or ran under MARS (we provided with syscall 100).

Extra implementation

Feel free to modify the program and add extra features like a game timer, special weapons, object energy levels, additional piglet type(s), additional backgrounds for different game levels, etc to the game. But these are just for fun, they will not earn you additional marks. Please make sure you **submit the original version of the game** for grading!

Game submission

Submit your assignment using the CASS system:

https://course.cse.ust.hk/cass

Make sure you select the correct course “comp2611”. You just need to submit the file angrybird.s. On the top of the file, write your name, student ID, email address as follows:

#NAME:

#ID:

#EMAIL:

Appendix: MIPS Syscall Code 100 (in modified version of MARS only)

Overview

A MIPS syscall of code 100 is defined in our special Mars package. It is for the game development using a MIPS program running in Mars.

Game Framework

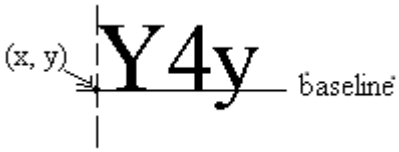
A game is composed of a number of game objects. It can also have an array of images stored in it. Each game object created has an unique ID for identification. A game object can represent one of the following three things (or none of them):

- Image – by associating the object with a non-negative index number to an image in the image array of the game.
- Text – by associating the object with a single-line text string. Note that new-line characters such as "\n" inside the string will be removed.
- Integer – by associating the object with an integer.

An object can represent only one piece of Image, Text or Integer. Setting an association of an object will remove any other associations of it. For example, associating Text with an object will remove the Image or Integer currently associated with it. To make an object represent nothing, we can associate it with a negative image index number. That is also the initial association of an object when it is created.

Game Drawing

The game draws each game object (if it does not represent nothing). The objects are drawn in an ascending order of their IDs. The game screen's pixels go from left to right in the x-axis and from top to bottom in the y-axis. Both x and y values start from 0. Each game object is drawn based on its pixel location (x, y), which can be set using the syscall. For drawing an Image object, the associated image will be drawn with its top-left corner at the (x, y) pixel of the game screen. For drawing Text or Integer object, the associated text string or integer will be drawn in such a way that the (x, y) pixel is the left-most pixel of the baseline of the text or integer. The following illustration shows where the baseline is.



If an object is Text or Integer, the color and font used for drawing it can also be set using the syscall.

Game Input

Any keystrokes on the game screen will be stored as the input on the MIPS program using Memory-Mapped Input Output (MMIO). Thus, the MIPS program should check or read the input according to the MMIO specification. In this game every keystroke generates an input. But the ASCII code stored in the input for a non-character keystroke is undefined.

Syscall Usage

For this syscall of code 100, $v0 is set to 100. $a0 is the *action code* set to indicate what action the syscall should do. The parameters for an action will be passed using some other registers (see the table below). Any errors generated during the syscall's action will terminate the MIPS program immediately. But the game screen, if any, will not be closed for debugging purpose. It can always be closed by clicking its window's Close button (the 'x' icon on the top-right corner).

Action	**Action code ($a0)**	**Parameters**	**Example**

Create game	1	$a1 = game screen width.
		$a2 = game screen height.
		$a3 = base address of a string for game's title.
		$t0 = base address of a string for the file path of game's background image (see the Note below).
			`.data`
			`title .asciiz "Star Wars"`
			`backImg .asciiz "back.gif"`
			`.text`
			`li $v0, 100`
			`li $a0, 1`
			`li $a1, 800`
			`li $a2, 600`
			`la $a3, title`
			`la $t0, backImg`
			`syscall`

Create game objects	2	$a1 = number of objects (at most 231) to create.
		All the objects initially represent "nothing" and are assigned the IDs 0, 1, 2, …, $a1 – 1, respectively.
			`li $v0, 100`
			`li $a0, 2`
			`li $a1, 20`
			`syscall`

Set the image array in game	3	$a1 = number of images (from the beginning of the array) to use.
		$a2 = base address of an array that stores the base address of each string for the file path of an image (see the Note below).
		The order of the images in the array are preserved in the game with the array index no. starting from 0.
			`.data`
			`tankImg .asciiz "tank.gif"`
			`gunImg .asciiz "gun.gif"`
			`imgList .word 0:2`
			`.text`
			`li $v0, 100`
			`li $a0, 3`
			`li $a1, 2`
			`la $a2, imgList`
			`la $t5, tankImg`
			`sw $t5, 0($a2)`
			`la $t5, gunImg`
			`sw $t5, 4($a2)`
			`syscall`

Create and show the game screen	4		`li $v0, 100`
			`li $a0, 4`
			`syscall`

Redraw the game screen showing any updates of the game objects since the last call of Redraw	5		`li $v0, 100`
			`li $a0, 5`
			`syscall`

Close the game window and destroy the game	6		`li $v0, 100`
			`li $a0, 6`
			`syscall`

Set game object to represent Image	11	$a1 = object ID
		$a2 = index no. to an image in the game's image array, or a negative number for resetting this object to represent nothing.
			`li $v0, 100`
			`li $a0, 11`
			`li $a1, 17`
			`li $a2, 0`
			`syscall`
			or
			`li $v0, 100`
			`li $a0, 11`
			`li $a1, 17`
			`li $a1, -1`
			`syscall`

Set game object's location	12	$a1 = object ID
		$a2 = x-coordinate
		$a3 = y-coordinate
		 
			`li $v0, 100`
			`li $a0, 12`
			`li $a1, 4`
			`li $a2, 220`
			`li $a3, 342`
			`syscall`

Set game object to represent Text (single-line text only)	13	$a1 = object ID
		$a2 = base address of the text string.
		Any new-line characters such as "\n" inside the string will be removed.
			`.data`
			`scoreText .asciiz "Score: "`
			`.text`
			`li $v0, 100`
			`li $a0, 13`
			`li $a1, 56`
			`la $a2, scoreText`
			`syscall`

Set game object to represent Integer	14	$a1 = object ID
		$a2 = the integer
			`li $v0, 100`
			`li $a0, 14`
			`li $a1, 56`
			`li $a2, 970  # i.e. a score`
			`syscall`

Set game object's color for drawing Text or Integer	15	$a1 = object ID
		$a2 = integer for a color that is a combination of red, green and blue components (byte 0 for blue, byte 1 for green, byte 2 for red, byte 3 is not used).
		An object has a black color set initially.
			`li $v0, 100`
			`li $a0, 15`
			`li $a1, 56`
			`li $a2, 0xff00ff  # purple`
			`syscall`

Set game object's font for drawing Text or Integer	16	$a1 = object ID
		$a2 = font size
		$a3 = non-zero for using Bold font, or else 0.
		$t0 = non-zero for using Italic font, or else 0.
		An object has a font of size 16 and Plain style set initially.
			`li $v0, 100`
			`li $a0, 16`
			`li $a1, 56`
			`li $a2, 32`
			`li $a3, 1`
			`li $t0, 0`
			`syscall`

Note: Use only GIF or JPG images. The file path of an image can be specified using *Relative* path (e.g., "back.gif" or "img/car.jpg") or *Absolute* path (e.g., "c:/img/car.gif" on Windows or "/img/tank.jpg" on Unix). A Relative path is relative to the current user folder of running Mars (by default, the folder of the Mars .jar program file). "/" not "\" should always be used to separate folders in specifying the path, even when running Mars on Windows.