COMP2611: Computer Organization

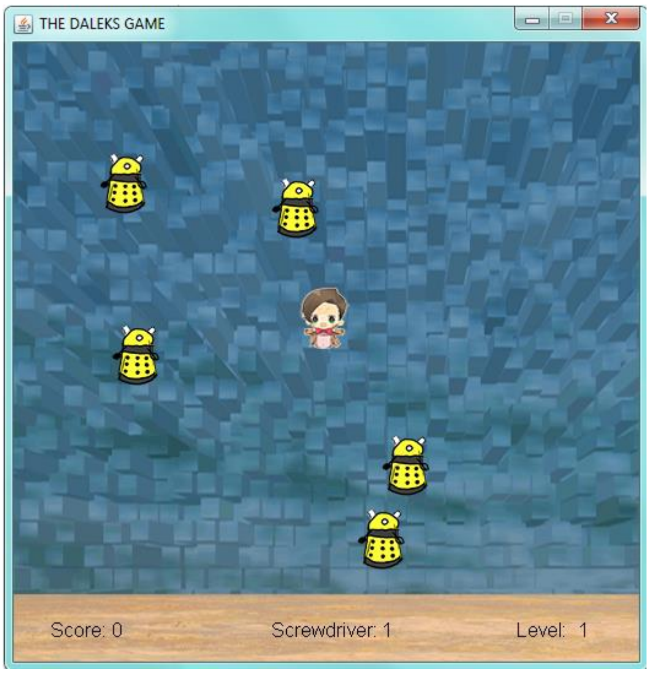
Spring 2015

Programming Project: The Daleks Game

Due Date: 30th April, 2015, 23:59 via CASS

Introduction

The Daleks are an extraterrestrial race of mutants from the planet Skaro, they travel around in tank-like mechanical casings, and are a race bent on universal conquest and destruction. One day, they arrived the lovely planet Earth and wanted to take possesion of it. Dr. Who, a handsome human-like Time Lord from the planet Gallifrey, started a war against the evil Daleks to prevent them from harming the innocent human beings. During the war, Dr. Who always used his ingenuity and minimal resources, such as his versatile sonic screwdriver, to destroy the Daleks. The war of the Daleks and Dr. Who will take place in the game episode, “The Daleks Game”, and you are going to complete the implementation of the game!



Game Objective

The objective of the game is to keep Dr. Who safe from the Daleks. Player can score points by destroying the Daleks. Daleks could be destroyed in three ways:

1. When two or more Daleks collide with one another (become a rubble afterwards);

2. When the Dalek walks onto a Dalek rubble;

3. When Dr. Who uses the "sonic screwdriver" and the Daleks in the range of the screwdriver will disappear.

Game objects

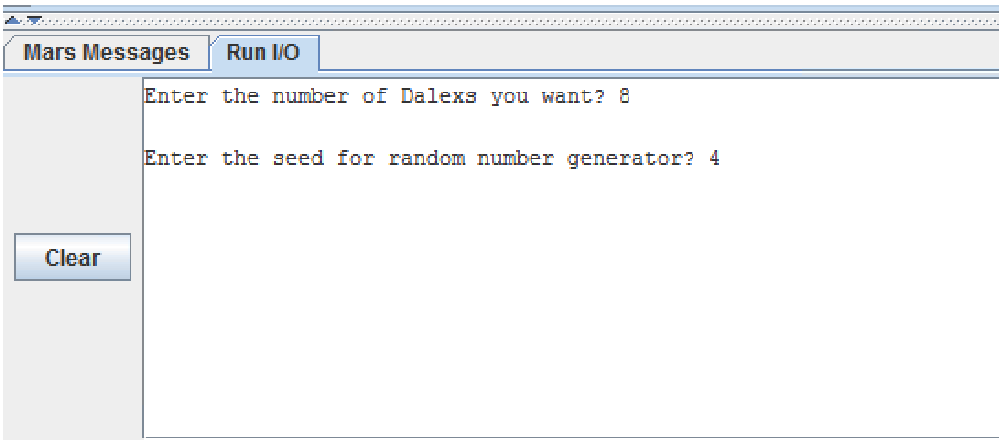
The game is played on a screen with a size of 510-pixel by 450-pixel (width by height). The X-coordinate value increases from the left to the right and the Y-coordinate value increases from the top to the bottom (i.e. the origin (0, 0) is at the upper left corner of the screen). Each game object: Dr. Who, the Daleks, the Dalek rubble are rectangular 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)**
Dr. Who1*	drwho1.png	40	50
Dr. Who2*	drwho2.png	40	50
Dalek	dalek.png	40	50
Dalek Rubble	rubble.png	40	50

* Note: drwho1.png and drwho2.png are different in posture and are use to generate the movement animation effect of Dr. Who.

Game flow

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



```
Enter the number of Dalexs you want? 8

Enter the seed for random number generator? 4
```

The game would then start at level 0. Dr. Who is positioned in the center of the rectangular game screen (at (230,200)). A number (according to the user input) of Daleks are positioned randomly on the screen. In each iteration of the game, Dr. Who will try to take an action, and Daleks will try to move closer to Dr. Who.

The action of Dr. Who is controlled by the numpad of the keyboard:

a) Move Dr. Who one step (10 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).

b) Keep Dr. Who in the same position (“5”).

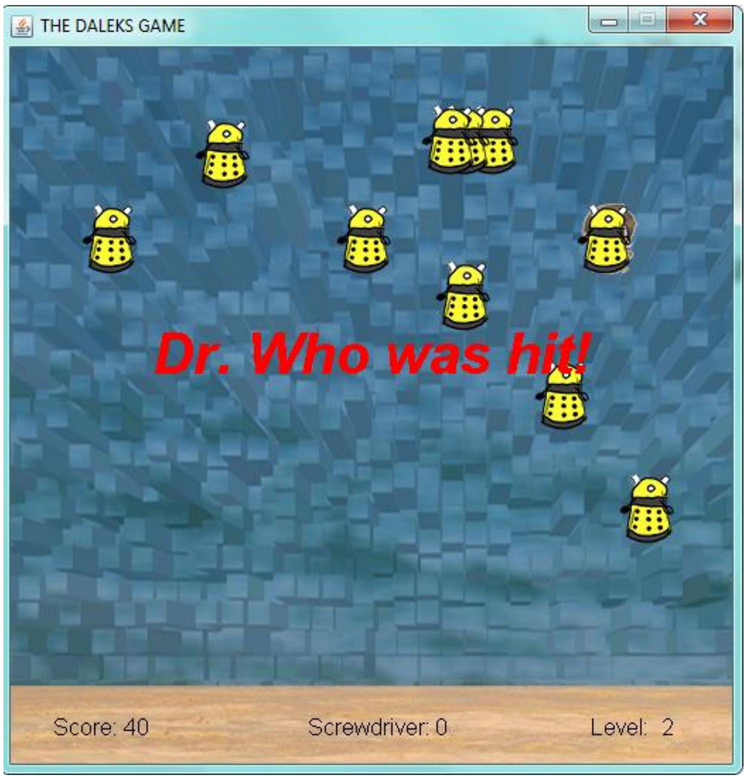
c) Teleport (“t”) Dr. Who to a random position on the screen, there is no guarantee that Dr. Who would not collide with a Dalek (in that case it will be captured).

d) Use the “sonic screwdriver” (“r”) to destroy all the Daleks adjacent to Dr. Who, Dr. Who can give only one screwdriver in each level.

Note that when Dr. Who reaches the boundary of the screen, it will not be able to move out of the boundary.

The movement of Daleks is controlled by the computer. Each of the Daleks will move one step (10 pixels) in the direction towards Dr. Who as long as Dr. Who took one action (“1”,”2”,”3”,”4”,”5”,”6”,”7”,”8”,”9”,”r”,”t”) in the current iteration. If no legal keystroke is received for Dr. who, the Daleks should stay still and wait for Dr. Who’s action.

If two or more Daleks move to the same position (i.e (X, Y) coordinates identical), they will be destroyed and turn into a Dalek rubble in the colliding position. Any other Dalek that walks onto the rubble will also be destroyed. The level ends when either all the Daleks are annihilated or Dr. Who is captured by one or more Daleks. If all the Daleks are annihilated, the game will move one level up, and the number of Daleks will be doubled (max 99 Daleks for all levels). If Dr. Who is captured, the game ends with the message “Dr. Who was hit!”.



Game scoring

Each Dalek destroyed by colliding with another Dalek will increase the player score by 10 (ie. If three Daleks collide, the player score will be incremented by 30). Each Dalek destroyed by colliding with a rubble will also increase the player score by 10. Daleks destroyed by the “sonic screwdriver” 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 Daleks.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 of the syscall 100 are listed at the end for your reference. You may read the skeleton program in details to understand the implementation of the game. But you only need to complete the code of the functions mentioned below for certain game tasks. Read the comments of the functions in the program and the game description above for doing the tasks correctly.

Function	**Task**
setGameoverOutput	When Dr. Who 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.) set the string color (syscall 100, $a0=15).
drwho_moves	Moves Dr. Who 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 Dr. Who has reached the boundary of the screen. Make sure if Dr. Who does not take action, the Daleks

	stay still.
update_state	Update the internal game states like score, etc. Also check any new rubble and the status of Dr. Who.
is_lv_up	Check if the game needs to proceed to a new level. Return value in $v0: 0 -- false, 1 – true.

<u>MIPS Syscall Code 100</u>

(for our custom-made 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

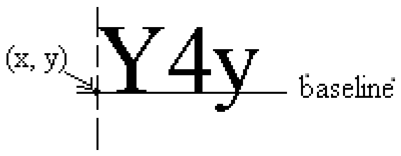
A game is composed of a number of game objects. It can also have an array of images stored in it. Each game object created has a unique ID for identification. A game object can represent one of the following three things (or none of them):

- 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*	.data scoreText .asciiz "Score: " .text li $v0, 100 li $a0, 13

		removed.	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.