

COMP2611 Computer Organization, Fall 2015

Programming Project: The Space Game

(Submission deadline: Nov 30, 5pm via Canvas)

1. Introduction

In this project, you are going to implement a Space game with MIPS assembly language. Figure 1 shows a snapshot of the game. A group of astronauts are wandering around the Space. The evil aliens might hurt the astronauts so the spaceship needs to wipe them off.

The spaceship emits bombs to kill aliens. The tricky part of the game is that the bombs may accidentally kill the astronauts too. Player needs to shoot and kill all aliens.

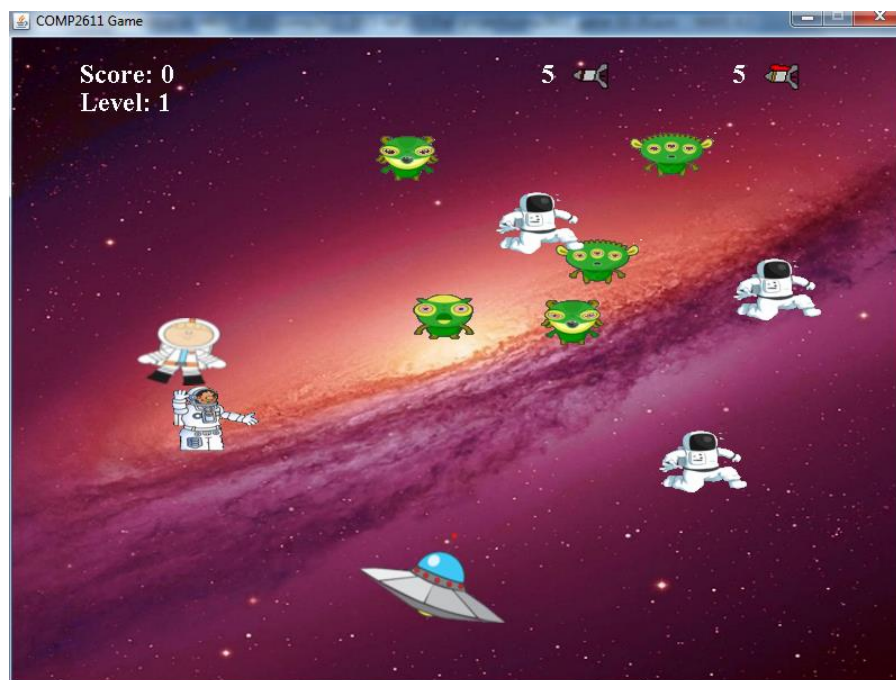


Figure 1 The snapshot of the Space game

2. Coordinate System

The game screen is of 800-pixel width by 600-pixel height as illustrated in Figure 2. The top-left is the origin of the coordinates, denoted as (0, 0). The value of x-axis increases from

the left to the right and the value of y-axis increases from the top to the bottom. This follows Java Swing coordinate system, which is used to support GUI in the game.

All game objects are represented as images with the rectangular shape. Moreover, each object's location is denoted as the coordinates of the top-left corner point of the rectangle. For example, the location of the alien in Figure 2 is specified by the point (540,300).

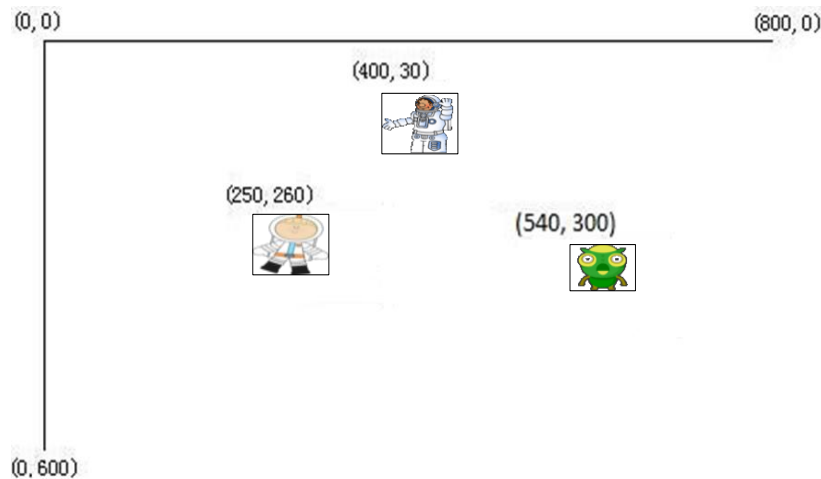


Figure 2 The coordinate system

3. Game Objects

There are five types of game objects: spaceship, astronaut, alien, simple bomb, and remote-control bomb. Every object has attributes as listed below. You can manipulate the above attributes through appropriate syscall services (details in following pages).

- **Current location:** a (x, y) coordinate which indicates the current location of the object.
- **Moving speed:** an integer variable indicating the moving speed of the game object.

Spaceship, alien and astronaut move horizontally at preset constant speeds of 4, 6, and 5 respectively, in pixels per time unit. Aliens slow down to speed of 3 when it's injured.

- **Moving direction:** a Boolean variable indicating the moving direction of the game object.

Spaceship, alien and astronaut moves from left to right when it's TRUE, from right to left otherwise. Bomb always moves upwards.

The next location of the object depends on its current location, moving direction and moving speed. All game objects should remain within the game screen, e.g. whenever the spaceship, the alien and the astronaut touch the border, it should turn around and move to the opposite direction.

- **Hit points (HP):** an Integer variable indicating the left life value of a game object. If the hit point of an object goes to zero, then the game object is “dead”. Astronaut and alien have initial hit point of 10 and 20 respectively. Both types of bomb and spaceship have the hit point of 1.

Object	Width (in pixels)	Height (in pixels)	Speed	Initial HP	Initial position
Spaceship	130	80	4	1	(320,440)
Astronaut	80	60	5	10	Random position with y coordinate in the range of [50, 350]
Alien	80	40	6 3 (injured)	20	Random position with y coordinate in the range of [100, 400]
Bomb	30	30	4	1	Decided by the location of Spaceship when the bomb is emitted

Table 1 Properties of game object


4. Game Details


4.1 Winning and losing of the game


The game has two levels: Level 1 and Level 2. Level 2 has more aliens to be destroyed. In each level, player targets to kill all aliens while keep astronauts alive. The game terminates when no astronaut is alive. Player wins the game when he/she passes both levels.

4.2 Bomb and hit

Bombs are emitted from the spaceship, moving upwards at the constant speed of 4. There are two types of bombs: simple and remote control.

Simple bomb is active once it's shot. Remote bomb is initially not destructive till it's activated by the player. Pressing the key 's' emits a simple bomb .

Pressing the key 'r' emits a remote-control bomb  which is inactive. Pressing the key 'a' afterwards

activates the remote bomb and it is shown as .

If the rectangle area of an active bomb intersects with the rectangle area of an astronaut, it's said that the bomb hits the astronaut and kills him, i.e. the hit point of the astronaut is set to be 0.

Things are little bit complicated when an active bomb hits the alien. Depending on the degree of intersection between the bomb and the alien, the alien might be killed or 'injured'.

In particular, suppose an alien currently locates at (x, y) on the screen. If the activated bomb intersects with the central area of the alien - the area which locates at $(x+35, y)$ with 10-pixel width and 40-pixel height, the alien is killed, and its hit point is reduced to 0, as illustrated by Figure 3(a). If the activated bomb intersects with the alien outside of the central area, as illustrated by Figure 3(b), the hit point of the alien is decreased by 5. Such alien will go to the "injured" state and its moving speed will be cut to half to 3. If the alien gets injured for a few times and its hit point reaches to 0, it's killed. The deducted hit points of aliens are accumulated as the game score.

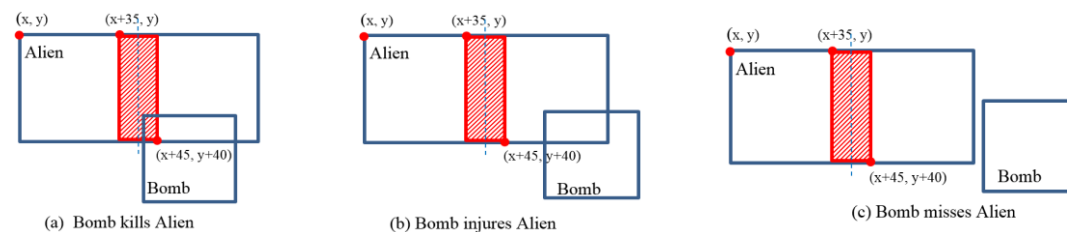


Figure 3 Different cases of bomb hit

Whenever the activated bomb explodes, the bomb itself is destroyed completely with its hit point reduced to 0. Please note an exploded bomb can cause damage to multiple objects simultaneously, an example is given by Figure 4.

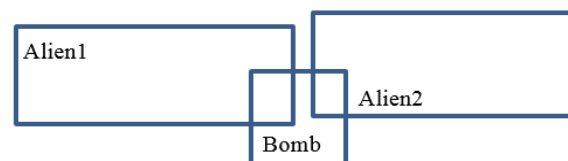


Figure 4 Bomb hits multiple objects

On the top of the game screen, the number of bombs available now is shown. Note if a bomb is exploded or falls outside the border of the game screen, it will be recycled back to use.

4.3 Game initialization

At the beginning of the game, the player can input the number of game objects such as astronauts, aliens, simple bombs and remote-control bombs. The number of each kind of game object is in the range [1, 10].

Given the initial parameters, the game engine draws the initial screen. Spaceship is put at the bottom center of the screen, heading to the right. Random pictures of astronauts and aliens are retrieved from image repository, and they are put in random position in upper part of the screen area, all heading to the right.

Level 2 of the game runs quite similar to level 1, with more aliens. The number of aliens in level 2 equals to number of aliens chosen by player plus 4.

4.4 Game skeleton

The skeleton code consists of a big loop, it proceeds as follows:

1. Get the current time: `jal get_time`.
2. Check whether the game reaches the ending condition: `jal check_game_end`. `game_end_status` then handle different actions for game over, game win, or promote from level 1 to level 2. If it's still in the middle of the game, go to step 3.
3. Update the game object's status: `jal update_object_status`. Check the game objects' status. Remove any killed aliens, astronauts or exploded bombs from the screen.
4. Process the keyboard input: `jal process_input`. The input key is stored using the Memory-Mapped I/O scheme. If the keyboard input is valid, perform the related action.

Input key	Action
s	Emit a simple bomb when available: <code>jal emit_one_bomb</code>
r	Emit a remote-control bomb when available: <code>jal emit_one_rbomb</code>
a	Activate all the remote-control bombs in the screen: <code>jal activate_rbombs</code>

5. Check bomb hits: `jal check_bomb_hits`. For each activate bomb, check whether it hits any astronauts or aliens. If any hit happens, update the status of astronaut or alien, and the bomb itself.
6. Move the spaceship: `jal move_ship`.
7. Move the astronauts: `jal move_Astronauts`.
8. Move the aliens: `jal move_Aliens`.
9. Move the bombs: `jal move_bombs`.
10. Update the score: `jal update_score`.
11. Refresh the game screen.
12. Take a nap if necessary: `jal have_a_nap`. The interval between two consecutive iterations of the game loop is about 30 milliseconds.
13. Go to step 1.

5. Your tasks

Read the skeleton and understand how it works. Then fulfill the following MIPS procedures in the skeleton code.

Note: You don't need to understand every single detail of the skeleton. You can discuss with your friends if you have difficulty in understanding the skeleton. But every single line of code should be your own work.

Procedure	Inputs	Outputs	Description
check_one_bomb_hit	\$a0: bomb id		<p>Given the bomb id, check whether it hits astronaut or alien.</p> <p>The astronaut is always killed by the bomb (hit point reduced from 10 to 0).</p> <p>The alien is either killed (hit point reduced to 0), or gets injured (deduct 5 hit points).</p> <p>This procedure pushes the coordinates of two game objects into stack, and then calls procedure <code>check_intersection</code>, to detect whether the two game objects intersect.</p>
check_intersection	recA: ((x1, y1), (x2, y2)) recB: ((x3, y3), (x4, y4))	\$v0: 1 for true(intersect with each other); 0 for false	<p>Check whether the given two rectangles are intersected.</p> <p>(x1, y1) and (x2, y2) are the coordinates of top-left and bottom-right of rectangle recA.</p> <p>(x3, y3) and (x4, y4) are the coordinates of top-left and bottom-right of rectangle recB.</p> <p>The eight coordinates are passed via stack by <code>check_one_bomb_hit</code>.</p>
move_Astronauts			<p>Move astronauts to the next location.</p> <p>If the astronaut passes across the border, turn it around and set proper new location and direction.</p>
update_score			<p>Game score is updated each iteration.</p> <p>The procedure increases the game score by the reduced hit points of Aliens in current iteration.</p>

Note: The skeleton file provided is just to help you understand the game thoroughly and ease your life. If you think it poses any restriction on you, feel free to have your own implementation from scratch. If you don't use the skeleton code, please state that at the beginning of the submitted .asm file.

6. Syscall Services in Usage

We have implemented a group of additional syscall services to support game related functions (e.g. GUI and sound). You should do your coding work using the modified Mars provided on our course website.

Syscall code should be passed to `$v0` before usage.

Service	Code	Parameters	Result
Create game screen	100	\$a0 = base address of a string for game's title; \$a1 = width \$a2 = height	
Create a Spaceship	101	\$a0 = id of this ship \$a1 = x_loc \$a2 = y_loc \$a3 = speed Note: id must be unique.	
Create an Alien	102	\$a0 = id, \$a1 = x_loc, \$a2 = y_loc \$a3 = speed \$t0 = image_id	
Create an Astronaut	103	\$a0 = id, \$a1 = x_loc, \$a2 = y_loc \$a3 = speed \$t0 = image_id	Note: the id of the game object must be unique!
Create a Text Object	104	\$a0 = id \$a1 = x_loc \$a2 = y_loc	Display the text message at the specified location.

		\$a3 = base address of a string for game's title;	
Play game sound	105	<p>\$a0 = sound id</p> <p>\$a1 = 1: (loop play), 0: play once</p> <p>The sound ids are described as follows:</p> <p>0: the sound effect of background;</p> <p>1: the sound effect of bomb exploding;</p> <p>2: the sound effect of emitting a bomb;</p> <p>3: the sound effect of game lose;</p> <p>4: the sound effect of game win;</p> <p>5: the sound effect of the bomb hitting an object</p>	Play a sound identified by \$a0.
Create a Simple Bomb	106	<p>\$a0 = id</p> <p>\$a1 = x_loc</p> <p>\$a2 = y_loc</p> <p>\$a3 = speed</p>	
Create a Remote Bomb	107	<p>\$a0 = id</p> <p>\$a1 = x_loc</p> <p>\$a2 = y_loc</p> <p>\$a3 = speed</p>	
Get Remote Bomb Status	108	\$a0 = id	\$v0 = 0: inactive, 1: active; -1: error
Activate Remote Bomb	109	\$a0 = id	Activate a remote bomb
Get Object Location	110	\$a0 = id	<p>\$v0 = x_loc;</p> <p>\$v1 = y_loc;</p>
Get Object Speed	111	\$a0 = id	\$v0 = speed;
Get Object Direction	112	\$a0 = id	\$v0 = 1: right; 0: left

Set Object Direction	113	\$a0 = id \$a1 = (0: left; 1:right)	
Deduct Hit Point of the Object	114	\$a0 = id \$a1 = point	deduct the hit point of a game object by the value of \$a1.
Get Object Score	115	\$a0 = id	\$v0 = score
Destroy an Object	116	\$a0 = id	Destroy the game object from the game screen (also from the java memory).
Update Game Score	117	\$a0 = score	
Get Hit Point of the Object	118	\$a0 = id	\$v0: the hit point
Refresh Screen	119		Redraw the game screen and all game objects which are alive.
Set Object Location	120	\$a0 = id \$a1 = x \$a2 = y	Manually set the location of the game object to be (x, y).
Update Object Location	121	\$a0 = id	Update the object location according to its current location, speed and direction.
Stop a game sound	122	\$a0 = sound id	If a background sound is played repeatedly, the syscall stops the sound.
Update bomb information	123	\$a0 = number of leftover simple bombs; \$a1 = number of leftover remote bombs;	
Update the Level message	124	\$fp = level of the game	Show the current level of the game (given in \$fp, either 1 or 2) or the top left of the screen

7. Submission

You should **ONLY** submit the file `comp2611_game_yourstudentid.asm` with your completed codes for the project. Please write down your name, student ID, and email address at the beginning of the file.

Submission is via Canvas. The deadline is a hard deadline. Try to avoid uploading in the last minute. If you upload multiple times, we will grade the latest version by default.

8. Grading

Your project will be graded on the basis of the functionality listed in the game requirements. Therefore, you should make sure that your code can be executed properly in the modified Mars.

9. Bonus

If you enjoy MIPS programming, and would like to make the Space game more fun, feel free to discuss your idea with instructor Dr. Cindy LI. With her pre-approval and successful implementation as planned, a bonus of up to 2 marks (out of 100 marks of course final score) will be assigned.

You may also propose your own game. Again, discuss with Dr. Cindy LI for pre-approval. A good game with reasonable complexity can also earn 15 marks project score plus the 5 marks bonus (out of 100 marks of course final score). Note you might need to modify the Mars and design your own syscall to support the new game.