

# COMP2611 Computer Organization, Fall 2018

## Programming Project: The Space Invaders

Submission deadline: 11:55PM, Dec 1 (Sat), CASS

### 1. Introduction

In this project, you are going to implement a Space Invaders game with MIPS assembly language. Figure 1 shows a snapshot of the game. The evil aliens are invading your space base so the spaceship needs to wipe them off. The spaceship emits bombs to kill aliens. Player needs to shoot and kill all aliens before they reach to your space station.

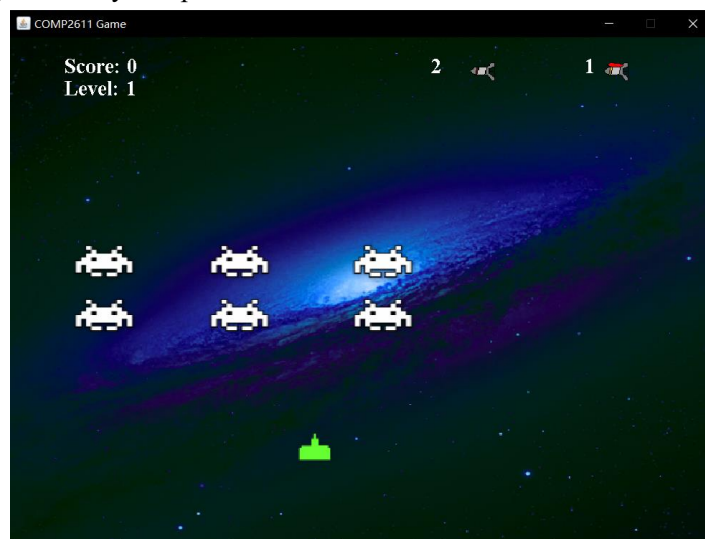


Figure 1. The snapshot of the Space Invaders.

### 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 with images of rectangular shape. Top left coordinate of each object is used to represent its location. For example, the alien in Figure 2 is at position (200, 120).

### 3. Game Objects

There are four types of game objects: Spaceship, Alien, Simple bomb, and Remote-control bomb. Every object has attributes as listed below. You can manipulate the attributes through appropriate syscall services (details in Section 6).

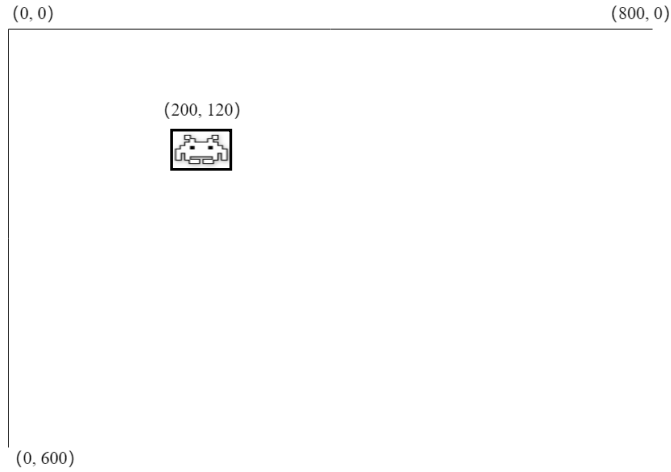


Figure 2. The coordinate system.add a rectangle

- **Current location:** the top-left (x, y) coordinate which indicates the current location of the object.
- **Speed:** the integer variable for the moving speed of the game object.
- **Direction:** a Boolean variable for the moving direction of the game object. Spaceship and alien move from left to right when it is TRUE, from right to left otherwise. Bomb shoot by Spaceship always moves upwards. The next location of the Alien depends on its current location, moving direction and moving speed. The next location of the spaceship depends on the input of player.
- **Hit points (HP):** the integer variable indicating the life value of a game object. If the hit point of an object goes to zero, then the game object is dead. Aliens have initial hit point of 20. Both types of Bomb and Spaceship have the hit point of 1.

Table 1. Properties of game objects.

Object	ID	Width	Height	Speed	Initial HP	Initial position
Spaceship	1	100	90	8	1	(320,440)
Alien	2, 3, and so on	80	60	6	20	(200, 120), (360, 120), (520,120), (200,180), (360,180), (520,180) Initial location of the 6 aliens in level one of the game
Simple Bomb	50, 51	30	30	6	1	Same as the location of the Spaceship when the bomb is emitted
Remote Bomb	80	30	30	6	1	Same as the location of the Spaceship when the bomb is emitted

## 4. Game Details




### 4.1 Winning and losing of the game

The game has four levels, with increasing number of aliens in each level, i.e. 6 aliens in level 1, and 3 additional alien for each higher level. The player wins in a particular level when he/she kills all aliens and promotes to next level. The player wins the game when he/she passes all levels.

The game terminates and player loses when there is an alien who touches the bottom of the game screen (i.e., y-label of the alien exceeds 420).

### 4.2 Bomb and Hit

Bombs are emitted from the spaceship, moving upwards at the constant speed of 6. There are two types of bombs: simple bomb and remote-controlled bomb.

Simple bomb is active once it is shot. Remote bomb is initially not destructive until it is activated by the player. Pressing the key ' ' (white space) 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 .

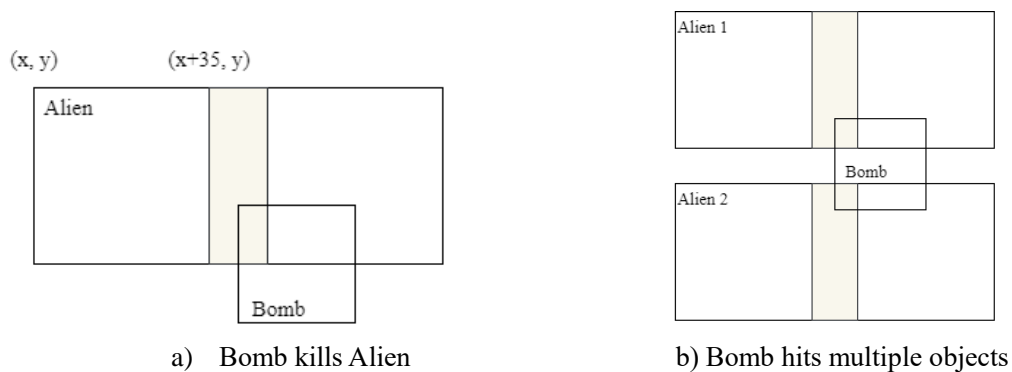


Figure 3. Different scenarios of bomb hit

As illustrated in Figure 3(a), suppose an alien currently locates at  $(x, y)$  on the screen. If a simple bomb or an activated bomb intersects with its central area, the area which locates at  $(x+35, y)$  with 10-pixel width and 40 60-pixel height, the alien is killed and its hit point is reduced to 0. Game score increased by 20. A dead alien's ID is updated to 0 (a flag that helps to check whether an alien is alive or dead.) The explosion of the bomb also destroys itself; therefore, its hit point is reduced to 0. Please note a bomb may cause damages to two aliens simultaneously when it happens to intersects with center of both.

For each level, there are 2 simple bombs and 1 remote-controlled bomb for player to use. `bomb_count` records the number of bombs that are currently flying. If `bomb_count` equals to 2, the player are temporarily banned from shooting simple bombs. For the remote-controlled bomb, if `rbomb_count` is 1, the player cannot shoot the remote-controlled bomb too. The number of bombs available to use is shown on the top left of the game screen.

### 4.3 Alien

Game level 1 has 6 aliens, in two rows, with ID 2, 3, and so on. All Aliens move horizontally at constant speed of 6. When touching the border, all Aliens go down 20 pixels. During the game, if the alien is destroyed by the bomb, its ID will be set to be 0.

Promotion to the next level of game will increase the number of alien by 3, speed by 3 and re-create all the aliens. `Alien_num` saves the number of Aliens in current game level. Array `Alien_ids` saves IDs of Aliens, starting from 2, 3, 4 until  $2 + \text{Alien\_num} - 1$ .

### 4.4 Game initialization

At the beginning of the game, the number of Aliens is 6 (2x3) and the numbers of simple bomb and remote bomb are 2 and 1, respectively. The following levels of the game run quite similar to the level 1, with more aliens fly with higher speed.

### 4.5 Game skeleton

The game runs as a big loop. Each loop print out a static game screen with all game objects. By properly set the time interval of game screen print, we can create an 'animated play' of the game.

It proceeds as follows:

1. Get the current time: `jal get_time`.
2. Check whether the game reaches the ending condition: `jal check_game_level_status` checks whether the user wins/loses/continues the current level. Then `jal game_end_status` handles different actions for game lose, game win or promote to next level. If the player is 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. Destroy any killed aliens or exploded bombs with `syscall 116`.
4. Process the keyboard input from player: `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
White space	Emit a simple bomb when available: <code>jal pi_emit_bomb</code>
R	Emit a remote-control bomb when available: <code>jal pi_emit_rbomb</code>
<b>E W</b>	Active all the remote-control bombs in the screen: <code>jal pi_active_rbombs</code>
A	Move the spaceship to left: <code>jal pi_ship_left</code>
D	Move the spaceship to right: <code>jal pi_ship_right</code>

5. Check bomb hits: `jal check_bomb_hits`: One-by-one, for each active bomb, check whether it hits any aliens by `jal check_one_bomb_hit`. If any hit happens, update the status of alien, and the bomb.
6. Move the aliens: `jal move_Aliens`.
7. Move the bombs: `jal move_bombs`.
8. Update the score: `jal update_score`.
9. Refresh the game screen.

10. Take a nap if necessary: jal have\_a\_nap. The interval between two consecutive iterations of the game loop is about 30 milliseconds.
11. Go to step 1.

## 5. Your tasks

Read the skeleton code and understand how it works. Then implement 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
pi_ship_left pi_ship_right			pi_ship_left sets ship direction to be left. Move the ship to next location according to speed. Note the ship can't move out of boarder. pi_ship_right does similar with ship direction to right.
check_intersection	recA: ((x1,y1), (x2,y2)) rec B: ((x3, y3)), ((x4, y4))	\$v0:1 for TRUE (intersect with each other); 0 for FALSE	Check whether the given two rectangles intersect. (x1, y1) and (x2, y2) are the coordinates of the top-left and bottom-right of rectangle recA. (x3, y3) and (x4, y4) are the coordinates of top-left and bottom-right of rectangle recB. The eight coordinates are passed via stack by check_one_bomb_hit.
check_one_bomb_hit	\$a0: bomb id	N/A	Given the bomb id, check whether the bomb hit any alien one-by-one. If the alien is hit by the bomb, its HP is reduced to 0. The bomb's HP is also reduced to 0. (Note a bomb may hit more than one aliens at one hit).

			This procedure pushes the top-left and bottom-right coordinates of bomb and an alien into stack, and then calls <code>check_intersection</code> to detect whether the two intersect.
<code>check_game_level_status</code>		<code>\$v0=0:</code> not end; <code>\$v0=1:</code> win; <code>\$v0=2:</code> lose.	Return the status of the current game level: win, lose or continue. One-by-one, check whether any alien touches the space base, i.e. the y-coordinate exceeds 420. If there's an alien touches the space base, lose. If all aliens are destroyed, win. Otherwise continue. Return status in <code>\$v0</code> .

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 with the modified Mars provided. Syscall code should be in `$v0` before usage.

Service	Code (in <code>\$v0</code> )	Parameters	Result
Create game screen	100	<code>\$a0</code> = base address of a string for game's title; <code>\$a1</code> = width <code>\$a2</code> = height	
Create a spaceship	101	<code>\$a0</code> = id of this ship <code>\$a1</code> = x_loc <code>\$a2</code> = y_loc <code>\$a3</code> = speed Note: id must be unique	

Create an Alien	102	\$a0 = id, \$a1 = x_loc, \$a2 = y_loc, \$a3 = speed, \$t0 = image_id	
Create a text object	104	\$a0 = id, \$a1 = x_loc, \$a2 = y_loc, \$a3 = base address of a string for game's title;	Display the text message at the specified location.
Play game sound	105	\$a0 = sound id, \$a1 = 1: (loop play), 0: play once The sound ids are described as follows: 0: the sound effect of background; 1: the sound effect of bomb exploding; 2: the sound effect of emitting a bomb; 3: the sound effect of game lose; 4: the sound effect of game win; 5: the sound effect of the bomb hitting an object	Play a sound identified by \$a0.
Create a simple bomb	106	\$a0 = id; \$a1 = x_loc; \$a2 = y_loc; \$a3 = speed;	
Create a remote bomb	107	\$a0 = id; \$a1 = x_loc; \$a2 = y_loc; \$a3 = speed;	
Get remote bomb status	108	\$a0 = id	\$v0 = 0: inactive, 1: active; -1: error.
Active remote bomb	109	\$a0 = id	Active a remote bomb.
Get object location	110	\$a0 = id	\$v0 = x_loc; \$v1 = y_loc.
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 on 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 CASS. 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 functionality listed in the game requirements. Make sure your code can be executed properly in the modified Mars.



## 9. Bonus

Everyone needs to finish the tasks as described in the project description.

But if you enjoy MIPS programming, and would like to make the Space Invader more fun, feel free to discuss your idea with your course instructor. With the pre-approval and successful implementation of additional functionalities 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 course instructor for pre-approval. A good game with reasonable complexity can also earn 2 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.