

COMP2611: Computer Organization, Spring 2013

Course Project: The Submarine Game

Due: May 24, 2013, 23:59

Introduction

In this project, you are going to implement a simple game called “The Submarine” using the MIPS assembly language. Figure 1 shows a snapshot of the game. Dolphins are swimming about in their ocean home, while submarines, the intruders, are cruising around the ocean as well. You are steering the ship with the mission to wipe out those intruders so as to bring the peace back for the dolphins. Your weapons consist of two types of bombs—simple bomb and remote-control bomb—that are equipped in the ship. Go ahead, and carefully use those bombs, as they might kill the dolphins accidentally. You win the game, when you destroy all the submarines and there is at least one dolphin still alive (too badly).



Figure 1: The snapshot of the game.

Coordinate system

The game screen is with 800-pixel width by 600-pixel height. Figure 2 illustrates the coordinate system. The top-left of the game screen is the origin of the coordinates. The x-axis grows from left to right; the y-axis grows from top to down. All the game objects are represented by images with rectangular shape and are located by the top-left corner. For example, the location of the submarine in Figure 2 is specified by the point (540,300). Note that this coordinate system is in accordance with the Java Swing coordinate system, as we use Java Swing to implement the GUI.

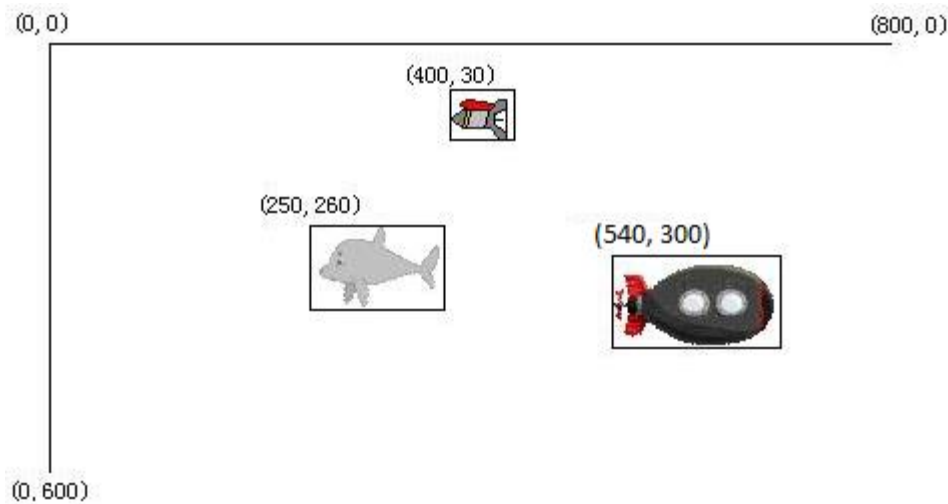


Figure 2: The coordinate system

Game objects

We abstract five types of game objects: ship, dolphin, submarine, simple bomb, and remote-control bomb. All the game objects share the following common attributes:

(1) Current location: (x, y): a two-tuple indicating the location of the game object.

We have already hidden the implementation details of the drawing function. However, you can still interact with the drawing system by setting locations of the game objects through syscall.

(2) Moving speed: an Integer variable indicating the moving speed of the game object.

The ship, submarines, and dolphins move horizontally at the constant speeds 4, 6, and 5 respectively, in pixels per time unit.

(3) Moving direction: a Boolean variable indicating the moving direction of the game object.

For the ship, dolphin, and submarine, “true” corresponds to the right direction; “right” means the left. For the bomb, “true” corresponds to the down direction. Note that “down” should be the

unique meaningful direction with respect to the bomb.

Note that, the next location of each game object depends on its current location, its moving direction, and moving speed. One caveat is to keep game objects from moving outside the border of the game screen. Specifically, whenever the ship, the submarine, and the dolphin touch the border, they need to turn to the opposite direction. To this end, you need to forecast whether the game object will cross the border at the beginning of the round and take measures to avoid that: setting the location (x, y) and alternating the moving direction.

(4) The hit point: an Integer variable indicating the left life value of a game object.

If the hit point value goes to zero, then it simply means the game object is dead. For example, if the hit point of the dolphin is zero, then it means you kill the dolphin unexpectedly. The dolphin initially has the hit point of 10; the submarine has the hit point of 20; the bomb has the hit point of 1; and the ship has the hit point of 1.

The above four are the common attributes shared by all five types of game objects. You can manipulate those attributes through appropriate syscalls (see later). Table 1 lists the image size of game objects.




Table 1: Image size of game object







Object	Image width (in pixels)	Image height (in pixels)
Ship	160	160
Dolphin	60	40
Submarine	80	40
Bomb	30	30



Next, we describe the most interesting and challenging parts of this game.

Bombs

A bomb is dropped from the bottom center of the ship, falling down vertically at the constant speed of 4. Each bomb is in either of the two states: Activated or Inactive. Only the activated bomb, when intersecting or touching either the submarine or the dolphin, can explode and cause damage. The inactive bomb is a dud and just falls through.

The simple bomb  is emitted through pressing the key 's'. It is initialized to be "activated" and cannot change its state. The remote-control bomb  is emitted through pressing the key 'r'. It is initialized to be "inactive". Pressing the key 'a' will trigger all the remote-control bombs in the screen to be activated  !

Whenever the activated bomb hits the dolphin, it results in permanent and deadly damage to the dolphin . The hit point of the dolphin will be deducted to be zero. When the activated bomb hits the submarine, the degree of damage depends on the hitting point. Suppose a submarine locates at (x, y) . If the activated bomb intersects with the middle of the submarine with 10-pixel width, that is, if the bomb intersects with the red rectangle area , locating at $(x+35, y)$ with 10-pixel width and 40-pixel height. We consider the bomb completely destroys the submarine  and thus the hit point of the submarine will be deducted to zero. On the other hand, if the activated bomb intersects with the submarine at any other areas, then the hit point of the submarine will be decreased by 5. The submarine goes to “damaged” state  and its moving speed is cut to half, becoming 3. Any deducted hit points of submarines are accumulated to the score **Score: 0**. When the hit point becomes zero, the damaged submarine is destroyed. When the activated bomb explodes, the bomb itself is destroyed (simple bomb: , remote-control bomb: ) with hit point being zero. It is worth noting that an exploded bomb can cause damage to multiple objects simultaneously. And the destroyed game objects remain on the game screen for a short while.

5   **4** indicates the available number of simple bombs and remote-control bombs that can be used instantaneously. In this case, 5 simple bombs and 4 remote-control bombs can be used. The maximum numbers of simple and remote-control bombs are set at the beginning of the game. When the bomb is exploded or the bomb falls outside the bottom border of the game screen, it will be reclaimed: the available number will be increased by one.

Game initialization

At the beginning of the game, the player is prompted to input the following four parameters: the number of dolphins, the number of submarines, the maximum number of simple bombs, and the maximum number of remote-control bombs.

The ship is initialized to be at $(x, y) = (320, 90)$ (being placed at the center of the screen on the surface of the sea), heading for the right. The submarines and dolphins are created and randomly placed. There are two constraints: i) the y-coordinates of submarine should be between 150 and 500; ii) the initial positions of submarines should not overlap; the initial positions of dolphins should not overlap as well. There is no constraint on the initial directions of dolphins and submarines.

Game winning or losing

The player wins the game, only when all the submarines are destroyed and at least one dolphin survives. Otherwise, the player loses the game. No matter wins or loses, a message is shown to the player.

The implementation and your tasks

We will provide a skeleton implementation to ease your life. Basically, it includes two parts.

The first part is with respect to initializing the game. After getting the initial number of dolphins, submarines, the simple bombs, and the remote-control bombs, the initialization procedure consists in creating the game screen, creating the ship, creating multiple dolphins, creating multiple submarines, and initializing the bomb settings. The detailed explanation, please see the comments in the provided skeleton source file.

Your tasks: implement the following two functions

Function name	Input params	Output params	Description
create_multi_dolphins	\$a0: the number of dolphins		Create multiple dolphins on the Game Screen. Constraint: the initial placement of the dolphins should not overlap.
create_multi_submarines	\$a0: the number of submarines		Create multiple submarines on the Game Screen. Constraint: the initial placement of the submarines should not overlap.

Note: To really ease your life, the implementation of the above two tasks has been provided in the skeleton file already.

The second part is the big loop of the game, which proceeds as follows:

1. Get the current time (T1): [jal get_time](#);
2. Check whether the game reaches the ending condition: [jal check_game_end](#). If the game ends, do the game ending processing; otherwise, go to step 3.
3. Update the game object's status: [jal update_object_status](#). Remove any destroyed submarines and dolphins from the game screen.
4. Process the keyboard input: [jal process_input](#). The input key is stored using the Memory-Mapped I/O scheme. If an input is available, read it and perform the action.

Input	Action
S	Emit a simple bomb (only if one is available): jal emit_one_bomb
R	Emit a remote-control bomb (only if one is available): jal emit_one_rbomb
A	Activate all the remote-control bombs in the screen: jal activate_rbombs

5. Check bomb hits: [jal check_bomb_hits](#). For each activated bomb, check whether it hits any dolphins or submarines. If any hit event happens, update the status of the damaged objects (dolphin or submarine) and that of the bomb itself.
6. Move the ship: [jal move_ship](#).
7. Move the dolphins: [jal move_dolphins](#).
8. Move the submarines: [jal move_submarines](#).
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](#). Get the current time (T2), and pause the program execution for (30 milliseconds - (T2 - T1)). Thus, the interval between two consecutive iterations of the game loop is about 30 milliseconds.
13. Go to step 1.

Your tasks: implement the following functions

Function	Input params	Outputs	Description
check_game_end		\$v0=0: not end; =1: win; =2: lose	Check whether the game is over!
update_object_status			1. if the dolphin is dead, then destroy the game object; 2. if the submarine is destroyed, then destroy the game object; 3. if the (r)bomb is already bombed, then destroy the game object; update the available number of bombs.
emit_one_bomb			1. check whether there are available bombs to use. 2. if yes, create one bomb object
emit_one_rbomb			1. check whether there are available remote bombs to use. 2. if yes, create one remote bomb object
activate_rbombs			Activate all the remote bombs: change their status to "activated"!
check_one_bomb_hit	\$a0: bomb id		Given the bomb id, check whether it hits with any dolphin or submarine. The dolphin will always hurt; but submarine depends! You need to deduct the hit points of the damaged dolphin or submarine, if necessary. This function relies on the function: check_intersection, to detect whether two rectangles intersect.
check_intersection	@rec1: ((x1, y1), (x2, y2)) @rec2: ((x3, y3), (x4, y4))	\$v0: 1: true; 0: false	Check whether the above two rectangles are intersected! Note: the eight parameters are passed through stack.
move_ship			Move the ship by one step, determined by its speed and direction. If the ship is going to cross the boarder, opposite the direction and set its new location appropriately!
move_dolphins			If a dolphin is going to cross the boarder, opposite the direction and set its location appropriately!
move_submarines			If a submarine is going to cross the boarder, opposite the direction and set its location appropriately!
move_bombs			If a bomb is going to cross the bottom, destroy the bomb and increase the available number of bombs.
update_score			The score is collected from submarines. If a submarine object is already destroyed, then 20 scores are earned. For the damaged submarine, since the object is still alive, the score can be got from the appropriate syscall.

Note: To really ease your life, the implementation of **move_ship** has been provided in the skeleton file already.

Syscall Usage

For this course project, we have implemented a group of extra syscalls to support the functionalities mentioned above.

Service	Code in \$v0	Parameters	Result
Create game	100	\$a0 = base address of a string for game's title; \$a1 = width \$a2 = height	
Create a Ship	101	\$a0 = id of this ship \$a1 = x_loc \$a2 = y_loc \$a3 = speed Note: id must be unique.	
Create a Submarine	102	\$a0 = id \$a1 = x_loc \$a2 = y_loc \$a3 = speed	
Create a Dolphin	103	\$a0 = id \$a1 = x_loc \$a2 = y_loc \$a3 = speed	Note: the id of the game object must be unique!
Create a Text Object	104	\$a0 = id \$a1 = x_loc \$a2 = y_loc \$a3 = base address of a string for game's title;	It will 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	It will 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
Activate Remote Bomb	109	\$a0 = id	If the remote bomb is inactive, then change its status to be activated!
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	This syscall will deduct the hit point of the game object by the value of \$a1.
Get Object Score	115	\$a0 = id	\$v0 = score
Destroy an Object	116	\$a0 = id	This syscall will destroy the game object from the game screen (also destroy 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 alive game objects.
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, the speed, and the direction.
Stop a game sound	122	\$a0 = sound id	If a sound is played in a loop manner, this syscall is to stop the sound.
Update bomb information	123	\$a0 = left number of simple bombs; \$a1 = left number of remote bombs;	

For other MIPS syscalls supported by Mars, refer to

<http://courses.missouristate.edu/kenvollmar/mars/help/syscallhelp.html>.

Submission

You should *ONLY* submit the file **comp2611_game.asm** with your completed codes for the project using the CASS (<https://course.cse.ust.hk/cass>). The CASS user manual is in this link <http://cssystem.cse.ust.hk/UGuides/cass/index.html>

At the top of the file, please write down your name, student ID, and email address in the following format:

#Name:

#ID:

#Email:

Important Note:

- The skeleton implementation is to help you understand the game more thoroughly and ease your life. If you think it poses any restriction on you, you are free to have a fresh implementation.
- **No late submission is allowed (the system will be closed on time).**

Grading

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

Cheating will be seriously punished!

In any case of detecting the suspicious codes, the student shall be required to defend the codes in a face-to-face session, which is to be organized during this examination period after the project due.