COMP2611: Computer Organization, Spring 2018

Programming Project: Pac-Man 2611

(Submission deadline May 9 at 5pm via CASS)

1. Introduction

In this project, you will implement (in part) a modified Pac-Man game named "Pac-Man 2611" using MIPS assembly language.

Figure 1 shows a snapshot of the game. The player navigates a Pac-Man through a maze containing various score points, and multiple colored ghosts. The game has two levels. The goal in each level is to eat ALL the score points in the maze. The Pac-Man has three lives at the beginning of the game. The ghosts roam the maze, trying to hit the Pac-Man. If the Pac-man bumps into any of the ghosts, the player needs to answer a quiz question. The Pac-Man loses a life if the answer is wrong, and the game will be over if all lives are gone. If the answer is correct, the Pac-Man would become shielded from the ghosts temporarily with a 'cloak of invisibility', allowing it to cross the ghosts without having to answer questions.

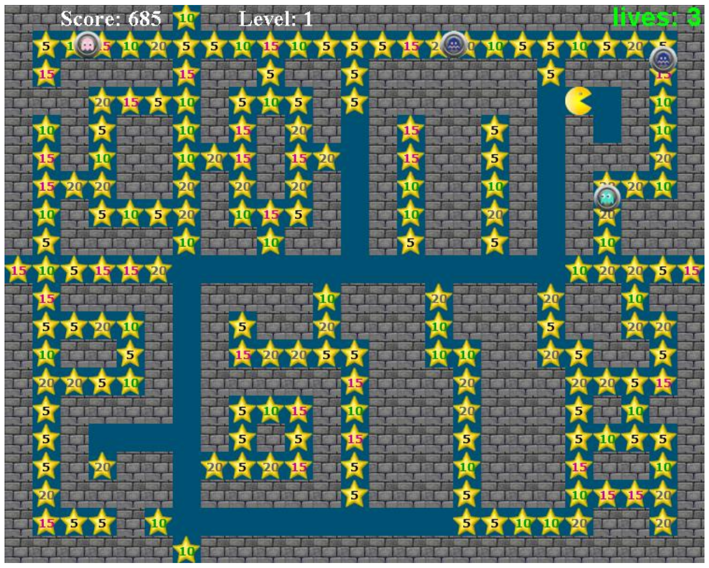


Figure 1. Game Screen of "Pac-Man 2611"

2. Coordinate System

The game screen size is of 750 x 600 pixels with the origin (0,0), being the top left corner as illustrated in Figure 2. Pixels in the game screen are arranged in a two-dimensional array of integers, each element being one pixel (or one colored dot). The top-left corner pixel being (0, 0), the bottom-right corner pixel is consequently (749, 599).

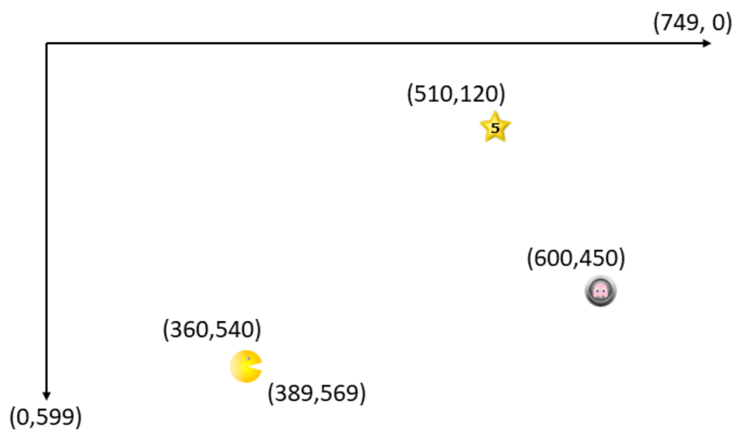


Figure 2. The coordinate system.

Each object in the game is represented by a rectangular image. For example, the Pac-Man character in Figure 2 is an image of 30x30 pixels. Its location is referenced by its top-left corner coordinate (360, 540) (therefore its bottom-right corner coordinate is (389, 569)).

The maze is of the same size as the game screen, with 20-rows by 25-columns grid of cells. Each cell is 30x30 pixels. A cell is either a wall ▨ or an open path ■. The maze is encoded as a two-dimensional bitmap array of 1s and 0s representing walls and paths respectively as shown in Figure 3. The maze has two 'escape routes' with no walls, one is horizontal and the other is vertical.

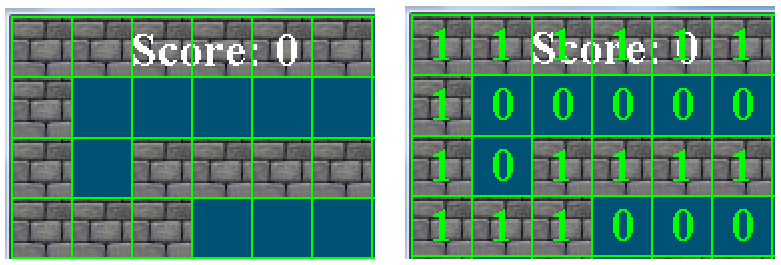


Figure 3. Illustration of the maze's grid (green square) and bitmap.

3. Game Objects

There are three types of game objects: Pac-Man, ghost and score point. Every object has attributes as listed below.

- **Current location:** a top-left (x, y) coordinate indicating the current location of the object
- **Moving speed:** an integer variable indicating the moving speed of the game object
- **Score Value (SV):** an integer variable indicating the score value of a score point object

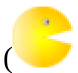
Object	**Width**	**Height**	**Speed (pixel)**	**SV**	**Initial location**
Pac-Man ( or the like)	30	30	3	-	(360, 540)
Ghosts (or the like)	30	30	2	-	Random path cell (far from Pac-Man). Ghosts do not overlap.
Score point (or the like)	30	30	-	Random	**ALL** path cells. Score points do not overlap.

Table 1: Images, sizes and attributes of the game objects

4. Game Details

4.1 Game initialization

Player chooses the number of ghosts at the beginning of the game. Then, the ghost objects, score point objects and one Pac-Man object are created with images retrieved from image repository. The game screen is then created and displayed.

The game has two levels: Level 1 and Level 2. Other initial settings at the beginning of each level include:

- The location of the Pac-Man object is set to (360, 540);

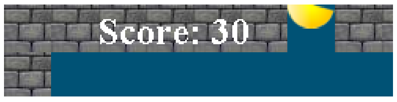
- Each ghost object is put on a random path cell of the maze far from the Pac-Man object, and no two ghost objects overlap;
- A score point object with random score value would be put on **every** empty path cell of the maze, no two score point objects overlap;
- The initial game score is 0;

Background music is played throughout the game. There are various sound effects for different game events (e.g. obtaining score points, entering/exiting from shield mode, etc.).

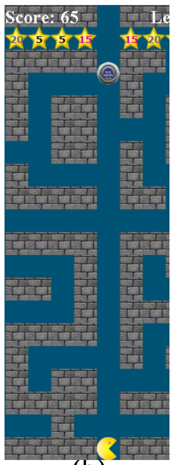
4.2 Object Movements

Pac-Man object moves vertically or horizontally when the player press “a”, ”s”, ”d”, “w”. Each keystroke moves the Pac-Man object by 3 pixels for 10 game iterations. There is no movement when the Pac-Man bumps a wall.

The Pac-Man can cross the border if there is a path (i.e., paths are toroids like the K-Map). This is shown in Figure 4 (a) where the Pac-Man crossed the border at the top to re-appear at the opposite border (at the bottom) as shown in Figure 4 (b).



(a)



(b)

Figure 4. (a) Pac-Man partially crosses the maze’s upper border. (b) Pac-Man is relocated to the maze’s bottom border.

During the game, the ghost objects patrol the maze in preset routes (by the game AI engine). Ghosts in Level 2 are smarter (or vampiric). If the ghost is far away from the Pac-Man, it teleports (with 1% probability) to a location nearby or even the same as that of the Pac-Man.

4.3 Score Points

At the beginning of the game, **every** empty path cell in the maze is placed with a score point object. Each score point object carries a score value (SV), which is visible to the player.

4.4 Collision between Game Objects

Pac-Man object collides with ghost or score point object when their images (rectangles) intersect. Figure 5 shows possible collision and no-collision scenarios:

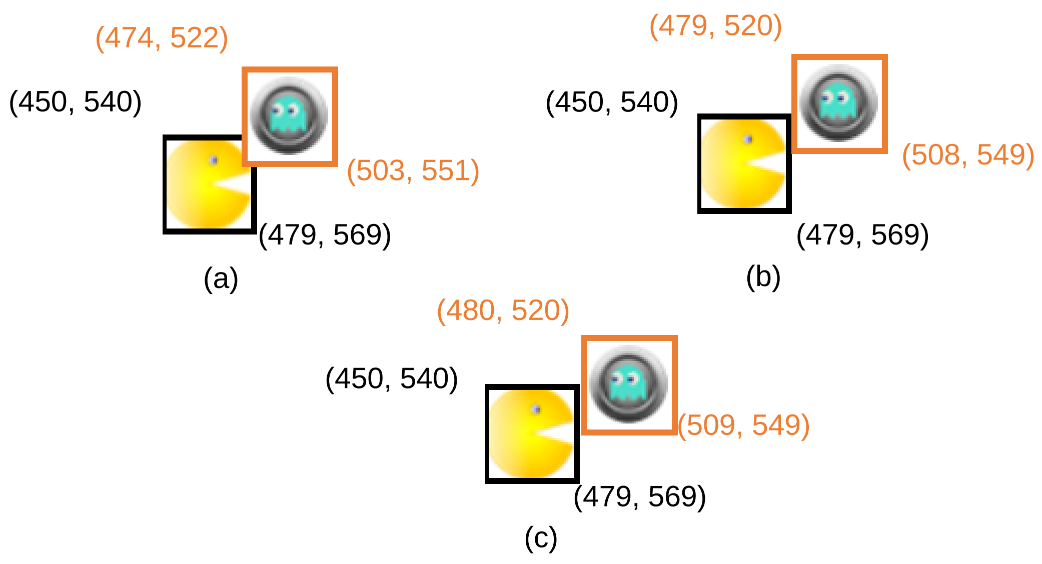


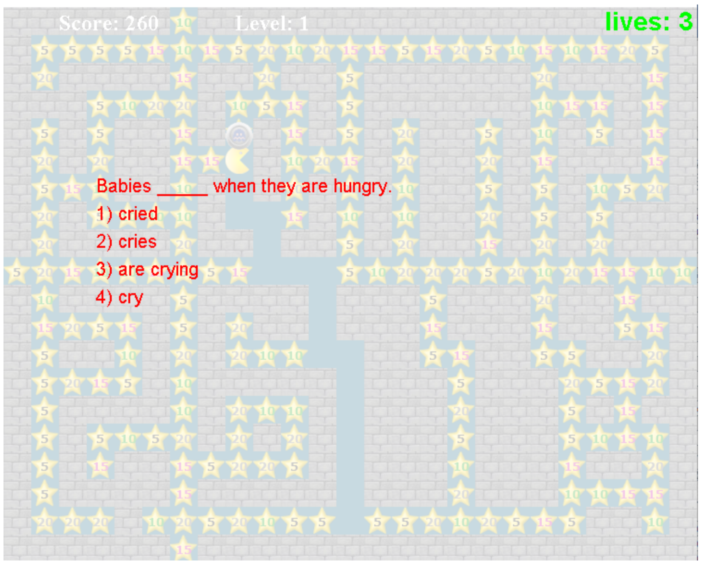
Figure 5. (a) Collision. (b) Collision. (c) No collision.

- **Collision of the Pac-Man and a score point object:** the value of the score point SV is added to game score. Remove the score point object from game.
- **Collision of the Pac-Man and a ghost object:** Quiz mode is triggered (described in Section 4.5).
- **Collision of the Pac-Man and multiple ghost objects at the same time:** Quiz mode is triggered only once.
- **Collision of the Pac-Man, a ghost object and a score point object at the same time:** take the score first, then enter into the quiz mode if the game is not over yet (some more score points still existed).

4.5 Shield Mode and Quiz Mode

Collision of Pac-Man and ghost objects triggers "quiz mode". During quiz mode:

- A MC quiz question is displayed (see an example in **Error! Reference source not found.**);
- All the ghost objects stop moving;
- Any inputs of the player other than those for answering the quiz are not processed.



Babies _____ when they are hungry.
1) cried
2) cries
3) are crying
4) cry

Figure 6. The Quiz mode.

- Answering a quiz question correctly puts Pac-Man in "shield mode" (represented as). Shield mode lasts for a pre-defined period of time and then everything goes back to normal.
- When the Pac-Man is in "shield mode", it is "invisible". No ghost can catch him. The collision does not trigger the quiz mode.
- Answering quiz question wrongly causes the player to lose a life, and the Pac-Man is teleported a random open-path cell of the maze without overlapping with any ghost object there.

4.6 Winning and Losing the Game

During the game play, if the player loses all lives, the player loses the game. If the Pac-Man accumulates **all** score points, the player will pass Level 1 and promote to Level 2. The player wins the game if he finishes both levels.

5. Game Skeleton

A game skeleton is in `pacman2611_skeleton.s` . After the game initialization, the game will proceed in a loop, whose iteration is said to be a game iteration. Each iteration follows the steps listed below:

1. Get the current time: `jal get_time`, which would be used in step 15.

2. Get the keyboard input: `jal get_keyboard_input`

   Note: Holding down the key generates a sequence of keystrokes, which moves Pac-Man object continuously. For MAC users, if it does not work, enable it by entering the following command in the Terminal application: "`defaults write -g ApplePressAndHoldEnabled -bool false`". Replacing `false` by `true` in the command will produce the opposite effect.

3. If the game is not in the Quiz mode, go to step 6. Otherwise, process the current keyboard input for the Quiz mode: `jal process_quiz_input`. If the input is valid, perform the corresponding action.

Valid input	**Action**
1	Answer the quiz by its answer choice 1 (end the Quiz mode, check the correctness of the answer and update the game score, trigger the Shield mode and/or perform the student teleportation based on that correctness).
2	Answer the quiz by its answer choice 2.
3	Answer the quiz by its answer choice 3.
4	Answer the quiz by its answer choice 4.

4. Check whether the game reaches the losing condition, and perform the corresponding action: `jal check_level_end`. If game is over, terminate the game.

5. Update the status of the mode: `jal update_quiz_status`. Go to step 13.

6. Check collisions with score point objects: `jal check_scorepoint_collisions`. For each existed score point object, check whether the Pac-Man object has collided with it. If a collision has happened, then update the game score, remove that score point object and skip further collision checking with any other score point objects.

7. Check whether the game level reached the winning condition, and perform the corresponding action: `jal check_level_end`. If the player wins the game, terminate the game. If the game has promoted to Level 2, go to step 1, otherwise go to step 6.

8. If the game is in the Shield mode, go to step 11.

9. Check collisions with ghosts: `jal check_ghost_collisions`. Check collision among pacman-ghost for every ghost object. If there is a collision, trigger the Quiz mode and go to step 16 afterward, skip further collision checking with any other ghost objects.

10. Move all the ghost objects: use the `syscall 213`.

11. Process the current keyboard input for changing the game status: `jal process_status_input`. If the input is valid, perform the corresponding action.

Valid input	**Action**
m	Toggle the playing of the background music.
0	If the game is not in the Shield mode, trigger the Shield mode. This is a cheat code built for you (in case you can’t answer quiz questions) to enforce the shield mode for easy debugging.

12. Continue any previous Pac-Man movement or process the latest movement input: `jal process_move_input`. First, save the current keyboard input as the latest movement input if it is a valid input as shown in the table below. Then, if a 10-iteration Pac-Man movement started in a past iteration has not finished yet, continue the movement, otherwise perform the corresponding action of the saved latest movement input (if any) and then remove this saved input.

Valid input	Action
w	Start the upward movement of the Pac-Man object (moving it upward by its speed for 10 game iterations starting from the current iteration, and if the game is in the Shield mode, also update the mode's status, using `jal update_shield_status`, in each of those iterations).
a	Start the leftward movement of the Pac-Man object.
s	Start the downward movement of the Pac-Man object.
d	Start the rightward movement of the Pac-Man object.

13. Display the number of remaining lives.

14. Refresh the game screen to reflect any screen changes: use the `syscall 201`.

15. Take a nap: `jal have_a_nap`. The interval between two consecutive game iterations of this loop is usually about 30 milliseconds. Then goto step 1.

6. Game Data

Many data structures are used in the game skeleton. Below are some of the data structures that might help you to understand the MIPS code.

`maze_bitmap: .byte`:

The game maintains a 0/1 bitmap for the 20-row by 25-column maze grid. 1 for a wall or 0 for a path. The `maze_bitmap` stores the bitmap as a 1D byte array.

`scorepoint_base: .word`

The first ID of all created score point objects, we assume that IDs of created score point objects are consecutive integers starting from `scorepoint_base`.

`scorepoint_num: .word`:

The total number of score point objects in the maze. Procedure `initialize_score_points` updates it when creating all score point objects in the maze.

`remaining_sp_num: .word`

The number of remaining score point objects not yet eaten by Pac-Man. When it is 0, Pac-Man eats all score point objects in a level.

```
total_score: .word
```

The total number of score points in the game (i.e. summation of the SV score values of all score point objects).

```
scorepoint_locs:  .word -1:600
```

Each score point object has coordinate (x, y), which is the top-left coordinate of its image. `scorepoint_locs` logically is a 1D array of structure. The (2i)th and (2i+1)th words of `scorepoint_locs` array hold x and y coordinates of score point object i, respectively. Assume there are at most 300 score points objects, therefore 600 coordinates.

When a score point object is 'eaten' by Pac-Man, its x coordinate is updated to -999, to create the 'disappear' effect on the game canvas.

```
scorepoint_sv:  .word 0:300
```

A 1D array of SV (score value) of score point objects. The ith word of `scorepoint_sv` array hold SV of score point object i. There are at most 300 score point objects, therefore 300 SVs.

```
pacman_id:   .word 0

pacman_locs: .word -1:2

pacman_speed:  .word 3
```

The ID, location and moving speed of Pac-Man. `pacman_locs` is an array of two elements, x and y coordinates of Pac-Man.

7. Tasks to Complete

Read the skeleton code and understand how it works. Complete the following MIPS procedures. You should not modify the skeleton code but only add your code to those procedures.

Note: Although you're highly recommended to read through the skeleton code and try to understand its structure, you don't need to understand every single detail of the skeleton code. You can discuss with your friends if you have difficulty in understanding it. But every single line of your code should be your own work (not something copied from your friends).

It would be helpful to maintain a mapping table (e.g. on a piece of paper) between data and which register its value been stored.

Procedure	**Inputs**	**Outputs**	**Description**
`initialize_score_points`			Put score point objects on all empty path cell in the maze. This procedure checks each cell in the 0/1 bitmap describing the cells in maze using a loop, and put a score point object using syscall if the cell is 0.
`move_pacman_up`	$a0 = 1 or 0	$v0 = 1 if a movement has been made, or else 0	Move the Pac-Man object upward by its speed for one game iteration. If $a0 is not 0, do not move the object if it will overlap with a wall cell in the movement.
`move_pacman_down`	$a0 = 1 or 0	$v0 = 1 if a movement has been made, or else 0	Move the Pac-Man object downward by its speed for one game iteration. If $a0 is not 0, do not move the object if it will overlap with a wall cell in the movement.
`check_intersection`	RectA: ((x1,y1), (x2, y2)) RectB: ((x3,y3), (x4, y4))	$v0: 1 for true (intersection happened); 0 for false	Check whether the given two rectangles intersect. (x1, y1) and (x2, y2) are the coordinates of top-left and bottom-right corners of rectangle RectA. (x3, y3) and (x4, y4) are the coordinates of top-left and bottom-right corners of rectangle RectB. The eight coordinates are passed via the stack.
`check_scorepoint_collisions`			Check whether the Pac-Man object has collided with a score point object. If a collision is found, then remove the score point object, increase the game score by its SV and skip further collision checking with any other score point objects. This procedure pushes the coordinates of two game objects into the stack, and

			then calls the procedure check_intersection to detect whether they intersect each other.
`check_ghost_collisions`		\$v0 = 1 if a collision has been found, or else 0	Check whether the Pac-Man object has collided with a ghost object. This procedure pushes the coordinates of two game objects into the stack, and then calls the procedure check_intersection to detect whether they intersect each other.

8. Syscall Services

COMP2611 team have implemented a group of additional syscall services to support (and simplify) game related functions (e.g. Pac-Man movements). For your code to work, you should use the modified MARS (`NewMars.jar`) provided in the project section of our course website.

COMP2611 teaching team, CSE, HKUST has the full copyright of the modified MARS. It's for your personal usage in the course. You should NOT upload it to any website for distribution, either free or profit.

Note that not all the new syscalls are needed in your work, some are described here for you to understand the skeleton.

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

Service	**Code**	**Parameters**	**Result**
Create the game screen	200		
Refresh the game screen	201		
Play game sound or stop playing it	202	\$a0 = sound ID \$a1 = 0: play once; 1: play repeatedly in loop; 2: stop playing The sound IDs are described as follows:	Play the sound of the given sound ID or stop any playing of it, depending on the given value in \$a1.

		0: the background music; 1: the sound effect of obtaining game scores; 2: the sound effect of losing game scores; 3: the sound effect of losing the game; 4: the sound effect of passing a game level; 5: the sound effect of ending Shield mode.	
Set the game score	203	$a0 = new game score	
Set the game level	204	$a0 = new level no.	
Create an object	205	Parameters: $a0 = unique ID of the object $a1 = x-coordinate $a2 = y-coordinate $a3 = object type: 0 for score point; 1 for Pac-Man; 2 for ghost Return value: $v0 = the score value of created score point object	A new object of the given ID and object type is created, replacing any existing object of the same ID and object type. The location of the object is set to the given x- and y-coordinates.
Set the location of an object	206	$a0 = ID of the object $a1 = x-coordinate $a2 = y-coordinate $a3 = the object's type: 0 for score point; 1 for Pac-Man	The location of the object is set to the given x- and y-coordinates.
Create a Text object	207	$a0 = unique ID of the object $a1 = x-coordinate	Create an object for displaying the text string at the given x- and y-coordinates.

		$a2 = y-coordinate $a3 = base address of a text string	
Get the location of a ghost object	208	$a0 = ID of the object	$v0 = x-coordinate of the location. $v1 = y-coordinate of the location.
Get a random path cell of the maze	209		$v0 = x-coordinate (at the top-left corner) of the cell. $v1 = y-coordinate of the cell.
Update the Quiz mode	211	$a0 = 0 for ending the mode; 1 for triggering the mode; 2 for updating the mode's display near its expiration	$v0 = choice no. of the quiz's correct answer choice.
Update the Shield mode	212	$a0 = 0 for ending the mode; 1 for triggering the mode; 2 for updating the mode's display near its expiration	
Move ghosts	213		Move all the ghost objects for one game iteration.

9. Submission

You should *ONLY* submit the file comp2611_project_yourstudentid.s with your completed code for the project. Please write down your name, student ID, and email address (as code comments) at the beginning of the file.

Submission is via CASS. Make sure you submit to COMP2611 folder. 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.

10. Grading

Your project will be graded based on the functionality listed in the project description and requirements. You should ensure that your completed program can run properly in our modified MARS. It must also work in a computer in our lab room 4213.