

COMP2611 Computer Organization Fall 2022

Programming Project: Tetris

Deadline 11:55PM, Nov 30 via Canvas

Copyright: All project related materials (this project description, modified MARS, project skeleton) is for personal usage of students of HKUST COMP2611 Fall 2022 offering. Posting it on a website other than the official course web page constitutes a breach of the copyright of COMP2611 teaching team, CSE and HKUST.

1 Introduction

In this project, you will use MIPS assembly program to implement a (modified and simpler) Tetris Game.

Tetris is a puzzle video game created by Soviet software engineer Alexey Pajitnov in 1984. It has been published by several companies for multiple platforms.

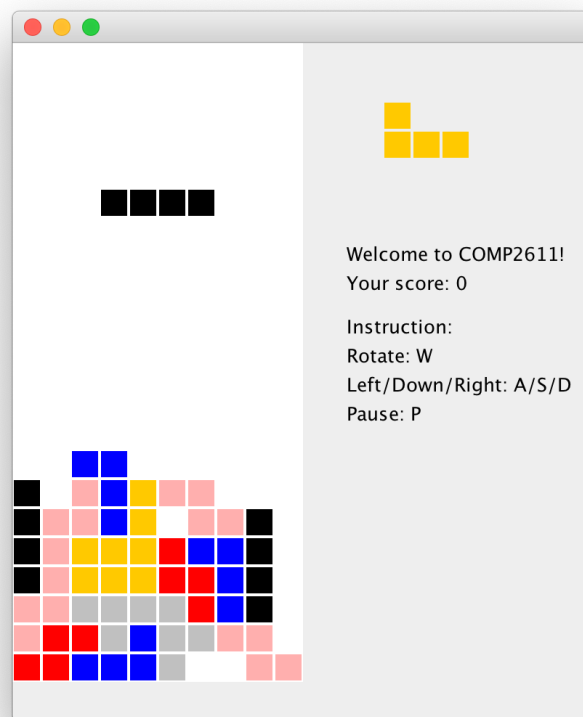


Fig. 1: Snapshot of Tetris Game

Figure 1 shows a snapshot of COMP2611 Tetris game. In Tetris, players complete lines by moving differently shaped pieces, called “tetrominoes”, which descend onto the playing field. (In what follows, we call “tetrominoes” as “**blocks**”.) The completed

lines disappear and grant the player points, and the player can proceed to fill the vacated spaces. The game ends when the uncleared lines reach the top of the playing field. The longer the player can delay this outcome, the higher their score will be.

In this project, we implement a simpler version of Tetris.

2 Coordinate System

The game field is of 10×23 squares area as illustrated in Figure 2. The top-left corner square is (0, 0) and the bottom-right corner square is consequently (9, 22).

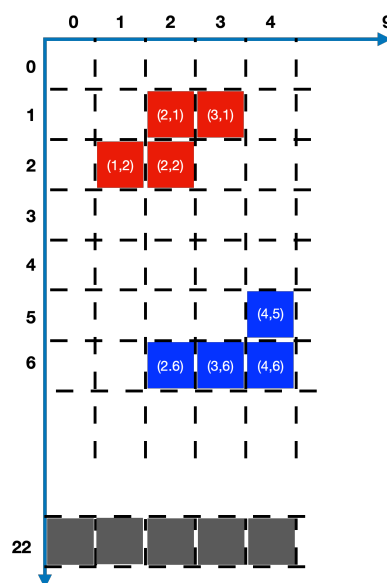


Fig. 2: The coordinate system

The space that the block can reach is limited to the first 22 rows (e.g. row 0 to row 21). The last row is the bottom border. The game field is encoded as a two-dimensional bitmap array: 0 for available, and 1 for occupied or the bottom border.

3 Game Objects

There is only one game object, **block** (tetromino) in this project. There are 7 types of blocks available to choose from. All blocks are made of four **squares**. The block switches in between four **modes** (sequentially anti-clockwise) by rotation. Attributes for a block are listed below.

- ✧ **block_x_loc**: the current x coordinate of the block
- ✧ **block_y_loc**: the current y coordinate of the block
- ✧ **block_mode**: the mode of the current block
- ✧ **block_id**: the type of current block

When the block moves to left or right, its x coordinate will be updated. When it moves down, its y coordinate will be updated. Rotation doesn't update the (x, y) coordinates of the block, but it does change the location of the squares in the block. Calculation of the coordinate of each square in a block will be introduced in Section 5.1.

Table 1 shows some detailed information about the attributes of blocks.





















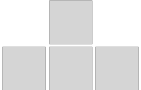




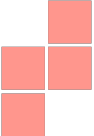
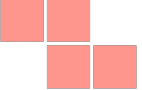
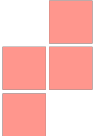
ID	Initial typical position (x,y)	Mode 0	Mode 1	Mode 2	Mode 3
0 Type I	(4,1)				
1 Type J	(5,1)				
2 Type L	(4,1)				
3 Type O	(4,1)				
4 Type S	(5,1)				
5 Type T	(5,1)				
6 Type Z	(6,1)				

Table 1: Block Attributes

4 Game Details

You should also read comments in the skeleton code to know more about game details.

4.1 Game initialization

The initial game setting, block properties, and game field matrix are stored in the Data Segment of memory. Before the game starts, we initialize the game via *init_game* in the skeleton code. The block object is created via a *syscall*. After using *syscall* 100, a GUI will appear and generate the first two blocks. And this *syscall* will return the ID of the current block.

4.2 Player actions

The player controls the current block to move to an adjacent (3 directions) valid path with “a”, “s”, and “d”. When the player presses “w”, the current block will rotate 90 degrees counterclockwise. In each game iteration, only one keystroke can be handled. It will operate the current block for one action (e.g., move or rotate). The block doesn’t have the superpower of ghostwalking so no action should be executed when it may cross the boundary or overlap with fixed squares.

The player can also pause the game with “p”, it can stop the block from moving down automatically which is very useful in debugging.

4.3 Game functions

The block keeps falling during the game. This *down* action can be performed by the player or the free fall. When the block can’t fall downwards anymore, the game needs to check whether the game is over. If the game is not over yet, the game needs to check whether there are full rows in the game field matrix and update accordingly. At last, a new block is created to start the next round of game.

4.4 Winning and Losing Conditions

The player will lose the game when the fixed squares reach the top of the game field and prevent the arrival of additional blocks.

The game never ends with the player's victory. The player can only complete as many lines as possible before an inevitable loss.

5 Game Implementation

5.1 Data Structure

The game field is encoded in a 2D matrix of size 10×23 . Each cell is initialized to be 0 (e.g. empty) except the last row to be 1 (e.g., bottom wall). It is internally saved as a 1D array *basic_matrix_bitmap* (row-major).

When the block is rotated, its (x, y) coordinates remains the same. But the locations of the four squares in the block need to be updated. This can be a tedious work. We simplify this by introducing two mode matrices of size 16×7 , one for the x coordinate of squares and the other for the y coordinate of squares.

For example, suppose the Type I block is now in location (x, y) and mode 2. Then by referring to Fig 3 first row, third column, we know the x coordinates of four squares in the block are $x-1$, $x+0$, $x+1$ and $x+2$.

	mode 0				mode 1				mode 2				mode 3			
Type I id = 0	-1	0	1	2	0	0	0	0	-1	0	1	2	0	0	0	0
Type J id = 1	-1	-1	0	1	-1	0	0	0	-1	0	1	1	0	0	0	1

⋮

Fig. 3: Data structure of the mode matrix

The mode matrices are 2D and are internally saved as 1D arrays *mode_x_loc* and *mode_y_loc* (row-major). Adding the current x coordinate of the block's location (*block_x_loc*) with the corresponding relative x coordinate in *mode_x_loc*, you can get its absolute x coordinate of the squares. For the y coordinate, it's the same.

Other game-related data are saved in quite a few variables and simple 1D arrays. All data structures used are static. Check the data segment of the skeleton at the very beginning of the skeleton code. Make sure you understand the data type of each variable carefully.

5.2 Game Loop

During game initialization, *init_game* creates the game objects and puts them in the game canvas.

The game then starts. It works as a big loop. Each iteration follows a collection of steps, e.g., checks game status; gets the keyboard input; executes player instruction; checks whether auto down, etc. Toward the beginning of each game iteration, losing conditions are checked to decide whether the game continues or not.

6 Tasks

When you code with high-level programming language, you always go through problem specification, algorithm design/workflow analysis, coding, debugging and documentation. Coding with low-level assembly programming language is pretty much the same.

Tetris game in MIPS assembly is challenging, but don't worry, you won't start everything from scratch. The fancy user interface is handled by modified Mars (copyright: COMP2611 teaching team). The MIPS code mainly works on the logic of the game. The programming assignment package already includes a skeleton file for you to start with.

'Divide-n-conquer' strategy is used in the skeleton. The skeleton code is loooong, but you will find it is well organized with sub-tasks handled by different MIPS procedures. The tasks of the programming assignment include a reading task (Task 0), you will need to read through the skeleton and grasp a big picture of the code structure.

The remaining tasks (Task 1-5) are coding tasks. You will focus on implementing a few MIPS procedures with well-defined interface.

6.1 Task 0 Reading Task

Spend a few hours to read the skeleton code. You should 1. understand the data structures used in the skeleton; 2. trace the game loop 3. figure out the functionality of each procedure.

Show your understanding of the big picture of the project by drawing a flow chart (<https://en.wikipedia.org/wiki/Flowchart>). Feel free to draw the flow chart with drawing tools or simple paper-and-pencil.

Your flowchart should

- include enough details, e.g., every single procedure should be included in your flow chart.
- have good layout so that it's easy to trace and understand.

6.2 Task 1 to 5 Programming Tasks

Once you build up a big picture of the project, you can zoom into the following coding tasks.

Table 2 lists all the programming tasks you need to implement. For some programming tasks in the skeleton code, you need to remove the code in the answer space and write your own code.

Note: You should not modify the skeleton code but only add your code to those procedures. Try to follow good conventions, e.g. comment your code properly and use registers wisely.

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). Do not show your code to others. Do not upload your code to GitHub or cloud unless you set the access right to be private.

Task	Input	Output	Description
Task1: block_move_right			Move the block rightward by one step. Move the object only when the object will not overlap with a wall or already fixed blocks.
Task2: block_rotate			Rotate the block by 90 degrees counterclockwise. Rotate the object only when the object will not overlap with a wall or already fixed blocks.
Task3: check_movement_valid	\$a0: potential x_loc, \$a1: potential y_loc, \$a2: potential block_mode \$a3: current block id	\$v0 = 1 if the movement is valid, otherwise 0.	Check the validity of all types of actions. Check whether the movement will cause overlapping with a wall or already fixed blocks
Task4: update_basic_matrix	\$a0: x_loc of block to be fixed \$a1: y_loc of block to be fixed \$a2: mode of block to be fixed \$a3: id of block to be fixed		Update data of the basic matrix when a block is going to be fixed
Task5: process_full_row	\$a0: the full row number \$a1: the width of matrix		Remove the specific full row and let the blocks placed above fall one rank. Update score and the Basic matrix in java code.

Table 2: Programming Tasks

7 Syscall Services

A modified MARS (NewMars.jar) with additional set of syscall services is needed to support the game (e.g. fancy UI, music, etc.). Table 3 list all the provides syscall services to implement the game.

Note that not all the new syscalls are necessary in your code, some are described here for you to understand the skeleton. Syscall code should be passed to \$v0 before usage. The GUI part of this game is implemented in MARS, but the game logic, e.g., whether the block touch the boundary, is determined by MIPS code. Therefore, you may find If the block crosses the wall, it will cause some errors when executing the initial skeleton code.

Note that the Basic matrix for game field exists in both Java code and MIPS code. The two matrices are initialized the same. The Basic matrix in MIPS is used for logical operations, and the one in java is for the GUI to draw correctly. Every time the MIPS basic matrix is updated, the Java basic matrix is also updated to keep them synchronized.

Service	Code	Parameters	Result
Create game screen	100		\$v0 = ID of the first block generated by the game
Refresh screen	101		
Play game sound	102	\$a0 = sound ID \$a1 = 0: play once 1: play repeatedly in loop; 2: stop playing The sound IDs are described as follows: 0: the background music; 1: the sound effect of player action 2: the sound effect of moving the full rows 3: the sound effect of losing the game	Play the sound of the given sound ID or stop any playing of it, depending on the given value in \$a1.
Create new block	103		Create a new block and return the ID of this new block. \$v0 = ID of the new block

Update Display Matrix	104	<p>\$a0 = the x coordinate of the current block's typical location</p> <p>\$a1 = the y coordinate of the current block's typical location</p> <p>\$a2 = the mode of the current block</p>	Display matrix only exists in java code. It tells the GUI where to paint on the game canvas.
Update Basic Matrix	105	<p>\$a0 = the x coordinate of the typical location of the block to be fixed</p> <p>\$a1 = the y coordinate of the typical location of the block to be fixed</p> <p>\$a2 = the mode of the block to be fixed</p>	Update the new fixed block to the Basic matrix in java code. Note that this syscall can only be used to update the Basic matrix in adding blocks.
Update game status	106		Inform java code the game status becomes false.
Update score and Basic Matrix	107	\$a0 = the row number to be removed in the Basic matrix	Inform java code: score ++ and update the removed full row to the Basic matrix. Note that this syscall can only be used to update the Basic matrix in removing specific row.

Table 3: Syscall Services 100 - 107

8 Submission

You should ***ONLY*** submit a single file **comp2611_project_yourStudentID.zip**.

The zip file should include a picture/pdf of the flow chart and your completed code for the project **comp2611_project_yourStudentID.s**. Please write down your name, student ID, and email address (as code comments) at the beginning of the files.

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

9 Grading

Your project will be graded by checking the functionalities listed in the project description and requirements. You should ensure that your completed program can run properly in our modified MARS. If the program can't be assembled, we will have to assign 0 point to your submission.