CITS1401/4406 Problem Solving and Programming

Project 2 Semester 2 2016

Set date: 17 October 2016

Submission deadline: Friday 04 November 2016 6:00 pm

Maximum Mark: 25

Submission on cssubmit

Overview

Project 2 involves developing a computer game. It takes place on a 5x5 board of squares on which a sequence of tiles appears, each holding a number. Players move tiles around by *sliding* rows and columns, with the aim of combining identical tiles to make higher numbers. The player's score is determined largely by the tiles on the final board.

You can play similar games such as *2048* online to get familiar with the game. In this project you will construct a computer player and also an interface for a human player.

The rules of the game

Tiles

The game is played on a 5x grid of squares. Each square is either empty, or it holds a tile holding a number in {2, 4, 8, 16, 32, 64, ...}, i.e. a power of two. At the start of the game the board holds between one and twenty five 2-tiles.

Moves

The player makes a sequence of moves, each of which is Left, Right, Up, or Down. The description below is for Up moves - probably you can translate for the other three possibilities, otherwise they are described at the end in detail.

An Up move causes all columns on the board to attempt to slide upwards.

- If no column can legally slide, nothing happens.
- Otherwise all legal Up slides are implemented, and then a new 2-tile (tile having a number 2) appears in the rightmost empty square of the bottom row.

Testing slides

A column's attempt to slide upwards is deemed legal if either or both of the following apply.

- A non-empty square in that column has an empty square above it.
- Two adjacent squares in that column hold identical tiles.

A column cannot slide upwards if it has *X* > 0 tiles at the top with no adjacent identical tiles and *4-X* empty squares at the bottom.

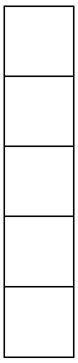
Implementing slides

When a slide is implemented, the following occurs.

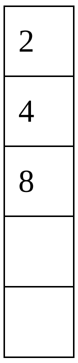
- All squares slide to the top of the column, then
- where there are two adjacent squares holding identical tiles *K*, those tiles combine into a single tile *2K* on the higher square, and all lower tiles slide up again. A tile can participate in only one combination in a given move.

Examples

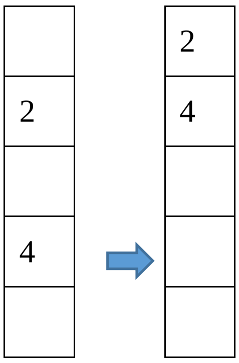
The following examples should help.



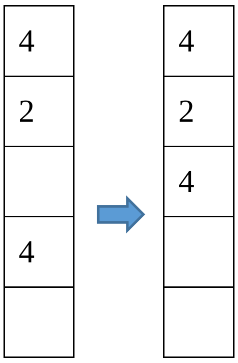
Illegal slide



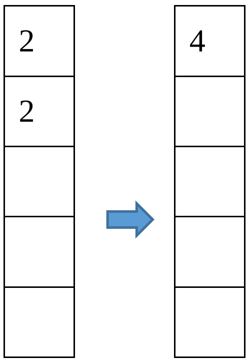
Illegal slide



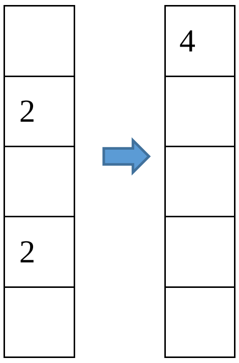
No combinations, only sliding



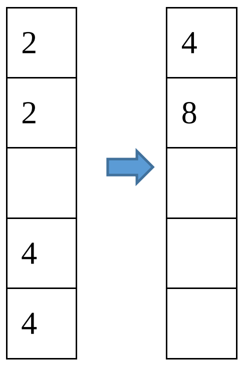
No combinations, only sliding



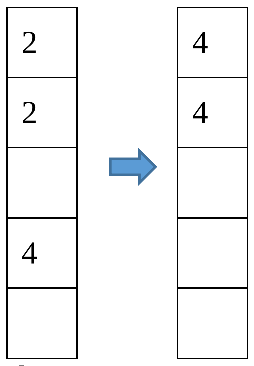
One combination



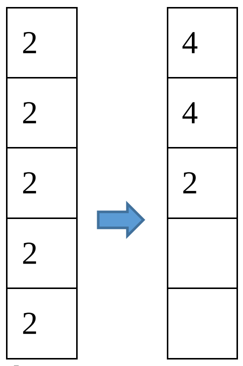
One combination



Two combinations



Combination is not recursive



Combination is not recursive

Game over

The game is over when no more successful (i.e. legal) moves are possible.

Scoring

Each successful move gets one point, and each unsuccessful move loses one point - so the score can go up or down as the game proceeds.

Tasks

Your program will be graded according to the number of tasks achieved by your program. You are allowed to use the graphics and math library files. If you want to use any other library files then you need to consult and seek approval before use.

Task 1:

Design the square board having the following components. You may select your own preferred layout for it.

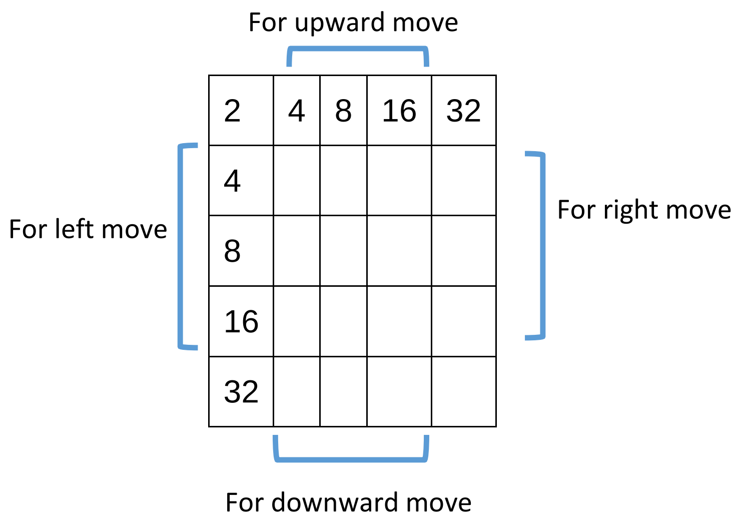
- `5 x 5 squares`
- `Score board`: showing the existing score of the game.
- `Status panel`: showing the status of the game or information about the game such as `Game over with final score`, `movement not possible`, `right movement is successful`, etc.
- `Quit button`: to terminate the program and quit the game.
- `New game button`: to start a new game. User must be asked after that whether human wants to play or computer.

Task 2:

Each square has can show a legal value clearly and ensure that squares which recently moved are highlighted or their text is having different colour than normal squares. No illegal values should be displayed in the squares.

Task 3:

The border rows or columns should be able to accept mouse clicks and make a legal move. Remember to exclude corner squares to avoid confusion. Look at the image below



Task 4:

Embed the upward move correctly when upward squares are clicked.

Task 5:

Embed the downward move correctly when downward squares are clicked.

Task 6:

Embed the right move correctly when right squares are clicked.

Task 7:

Embed the left move correctly when left squares are clicked.

Task 8:

The status of the game should be shown correctly on the `Status panel` such as `Game over with final score`, `movement not possible`, `right movement is successful`, etc.

Task 9:

The score of the game should be updated regularly on the `Score board.`

Task 10:

The feature of the `new game button` is embedded correctly. The new game will be started at any time when `new game button` is clicked irrespective of the status of the game even if games is in progress. The game in progress should be terminated. On the click of the `new game button`, the user must be provided with two options: whether a `human` or `computer` wants to play the new game.

- If option of `human` is clicked then human interface of the game should be enabled and game will progressed as human click the legal squares as mentioned in Task 3.
- If option of `computer` is clicked then computer will play the game until game is over or `new game button` is clicked or `quit button` is clicked. The moves

are selected by computer randomly among the four legal moves which are upward, downward, right and left. All features of the game will remain the same such as showing the status, score, update of square, etc.

Remember to add time between random selected moves so that status, score of the game and values of the square can be viewed.

Additional Tasks:

You may add the following tasks to your project in order to regain the marks lost in the required tasks.

- The program may have to ability to generate variable size of the board.
- The computer player is intelligent enough to get high scores.
- The fancy interface which makes the game look professional and marketable.

REMEMBER, YOU CAN NOT GET MORE THAN 100% MARKS.

Submission:

You need to submit a single zipped file via *cssubmit*. Only one submission per team is required, but make sure that names and student IDs are clearly mentioned in each file. Following are the files which are expected to be submitted.

- Python file containing your code. If you have multiple coding files then submit all of them.
- Associated library files if you have used any after getting permission.
- Readme.doc file that explains which of the mentioned tasks are achieved. It should also describe your approach to achieve all the tasks. Include the references if required in this document.

Follow the programming guidelines discussed during the lectures. Your program must run on the ECM computers. It is your responsibility to thoroughly test your code and ensure that it is error free.

Assessment:

Your submission will be assessed on

- Completeness of tasks: how many of the tasks your code has achieved;
- correctness: whether your code implemented the specifications exactly;
- clarity: whether your code is clear to read and understandable;
- display: how attractive your display is.

You can (theoretically) get 100% in this project by implementing completely

- all game tasks, and
- a computer player that selects a random move to attempt at each turn, and
- a basic screen display as mentioned in the tasks.

However, if you lose marks in the basic phase of the project, you can regain some of those marks by going further with some additional tasks of the game.

If you extend your program in an attempt to get bonus marks, please make that clear in your submitted readme.doc file. Note that it is not possible to get more than 100% of the project mark.

Moves in details:

All the moves are explained in detail below.

Up moves

An Up move causes all columns on the board to attempt to slide upwards.

- If no column can legally slide, nothing happens.
- Otherwise all legal slides are implemented, and then a new 2-tile appears in the rightmost empty square of the bottom row.

Testing slides

A column's attempt to slide upwards is deemed legal if either or both of the following apply.

- A non-empty square in that column has an empty square above it.
- Two adjacent squares in that column hold identical tiles.

A column cannot slide upwards if it has *X* tiles at the top with no adjacent identical tiles, and *4-X* empty squares at the bottom.

Implementing slides

When a slide is implemented, the following occurs.

- All squares slide to the top of the column, then
- where there are two adjacent squares holding identical tiles *K*, those tiles combine into a single tile *2K* on the higher square, and all lower tiles slide up again. A tile can participate in only one combination in a given move.

Down moves

A Down move causes all columns on the board to attempt to slide downwards.

- If no column can legally slide, nothing happens.
- Otherwise all legal slides are implemented, and then a new 2-tile appears in the leftmost empty square of the top row.

Testing slides

A column's attempt to slide downwards is deemed legal if either or both of the following apply.

- A non-empty square in that column has an empty square below it.
- Two adjacent squares in that column hold identical tiles.

A column cannot slide downwards if it has *X* tiles at the bottom with no adjacent identical tiles, and *4-X* empty squares at the top.

Implementing slides

When a slide is implemented, the following occurs.

- All squares slide to the bottom of the column, then
- where there are two adjacent squares holding identical tiles *K*, those tiles combine into a single tile *2K* on the lower square, and all higher tiles slide down again. A tile can participate in only one combination in a given move.

Left moves

A Left move causes all rows on the board to attempt to slide leftwards.

- If no row can legally slide, nothing happens.
- Otherwise all legal slides are implemented, and then a new 2-tile appears in the highest empty square of the right column.

Testing slides

A row's attempt to slide leftwards is deemed legal if either or both of the following apply.

- A non-empty square in that row has an empty square left of it.
- Two adjacent squares in that row hold identical tiles.

A row cannot slide leftwards if it has *X* tiles on the left with no adjacent identical tiles, and *4-X* empty squares on the right.

Implementing slides

When a slide is implemented, the following occurs.

- All squares slide to the left of the row, then
- where there are two adjacent squares holding identical tiles *K*, those tiles combine into a single tile *2K* on the left square, and all tiles on the right slide left again. A tile can participate in only one combination in a given move.

Right moves

A Right move causes all rows on the board to attempt to slide rightwards.

- If no row can legally slide, nothing happens.
- Otherwise all legal slides are implemented, and then a new 2-tile appears in the lowest empty square of the left column.

Testing slides

A row's attempt to slide rightwards is deemed legal if either or both of the following apply.

- A non-empty square in that row has an empty square right of it.
- Two adjacent squares in that row hold identical tiles.

A row cannot slide rightwards if it has *X* tiles on the right with no adjacent identical tiles, and *4-X* empty squares on the left.

Implementing slides

When a slide is implemented, the following occurs.

- All squares slide to the right of the row, then
- where there are two adjacent squares holding identical tiles *K*, those tiles combine into a single tile *2K* on the right square, and all tiles on the left slide right again. A tile can participate in only one combination in a given move.