Assignment 1

How will I be assessed?

You will be assessed in two parts

1. Automated Testing (~20%)
2. Review by your tutor (~80%)

Automated testing:

This will test individual parts of your program against scenarios. Each test case successfully met will contribute 1 point. Note that you can press "Mark" as many times as needed. We will mark only the last attempt.

Review of your work by your tutor

Your tutor will look over your work as a whole (parts 1-8). This will align with the rubric included.

If you are a student of FIT1053, you will also complete parts 9-11 which will also be reviewed by your tutor.

Parts 12-13 are optional and will not be marked.

Don't stress, all they're to confirm is that you are able to use the different techniques you've learnt about so far in the unit (e.g. do you understand how to meaningfully design a conditional to achieve some broader goal)

You will also see some components listed as "Unmarked" or "Not assessed". These are to give you an opportunity to develop a more complete version of the game (perhaps even one you could play with a friend). The simplified version of this exists to allow you to auto-test your work while the latter is more representative of a game.

Maximising your marks from TA review

You should aim to ensure your program includes the elements in the rubric (next page).

To ensure this is the best representation of your ability, you should ensure you contribute to tasks in workshops and applied classes so that the **feedback** your group receives will be **meaningful** for you and will guide you in improving proficiency.

Academic integrity

How will this be checked?

Your code will be compared against every other students' work in the unit.

You should also assume that anything you are able to google can be easily found by the teaching team and compared against your work.

How can I avoid academic integrity issues?

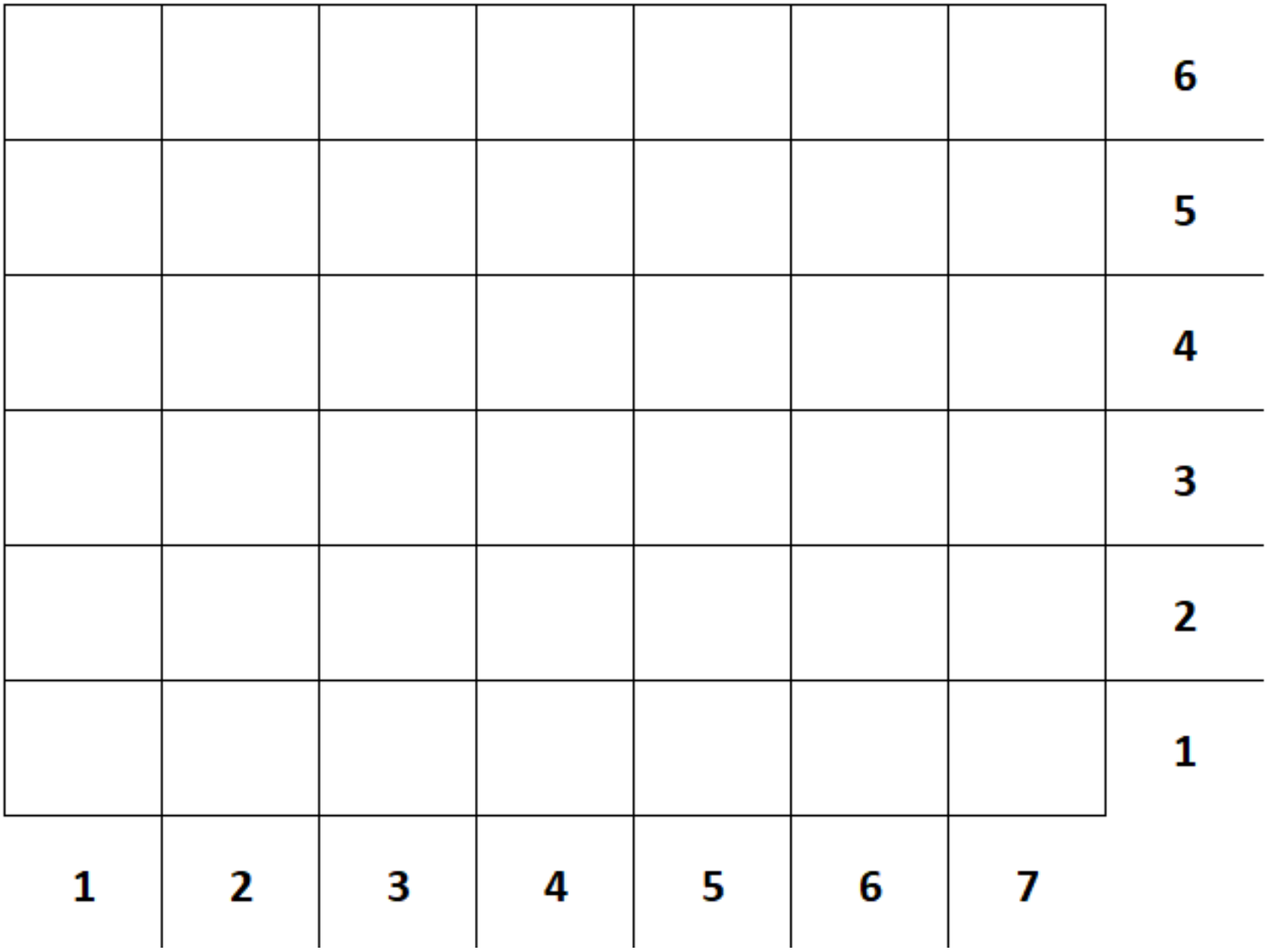
- Never copy code from anywhere. If you learn something useful online, rewrite it from scratch. It's also the best way to make sure you have understood it.
- If a fellow student asks you for a solution to a question, try instead to figure out what it is that they do not understand about the unit's content that prevents them from finding the solution themselves. *Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime.* Also, remember that giving your solution is just as much of an Academic Integrity breach as receiving it!
- You may find yourself in a situation where you feel like you physically cannot submit the assignment on time. Remember that you can submit an extension request (see Moodle AU or Moodle MA), and you can seek help (see Moodle AU or Moodle MA). If nothing works, remember that failures are part of the learning journey. And what is more important to you, an assignment mark or your honour?

Assignment Rubric

Description of the task and background information

This program will have you simulating the Hasbro game of connect 4.

This game involves a vertical board with 6 rows and 7 columns as shown below.

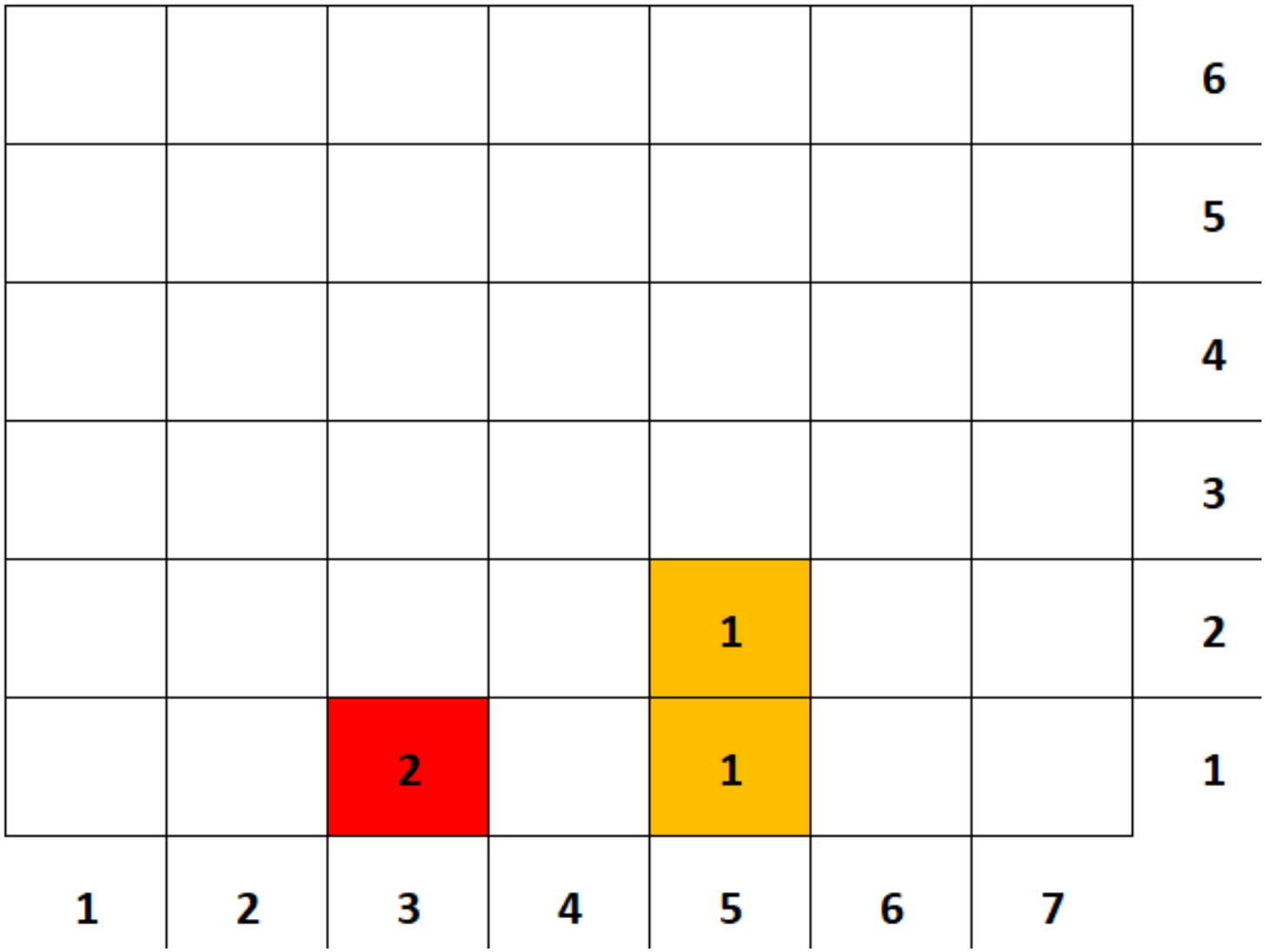


Rows increase from the bottom to the top (i.e. the bottom row is row 1 and the top is row 6).

Columns increase from left to right (i.e the leftmost column is column 1 and the rightmost is column 7).

Taking turns

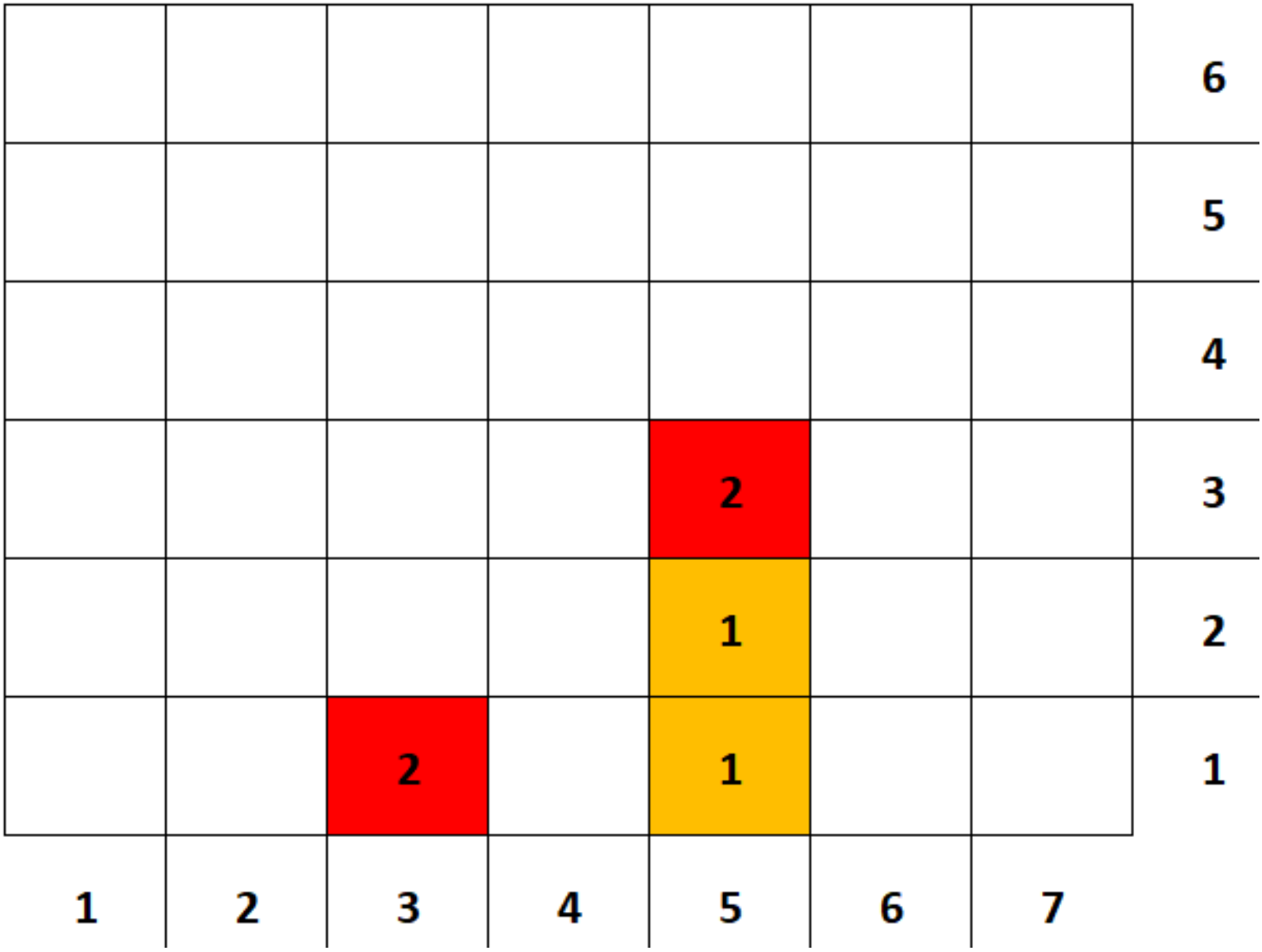
Players take turns dropping a coloured token into one of the columns as shown below.



State of the game after player 1 (orange) has dropped two tokens into column 5 and player 2 (red) has dropped one token into column 3.

Each turn adds a piece onto the top of the existing pieces (or at the bottom row if the column is empty), e.g. the next move by player 2 (red) could appear on the bottom row if in columns 1,2,4,6 or 7. If in column 3 it would go above their own piece and if in column 5 it would be above the other player's piece.

Let's assume they played in column 5, the result is given below

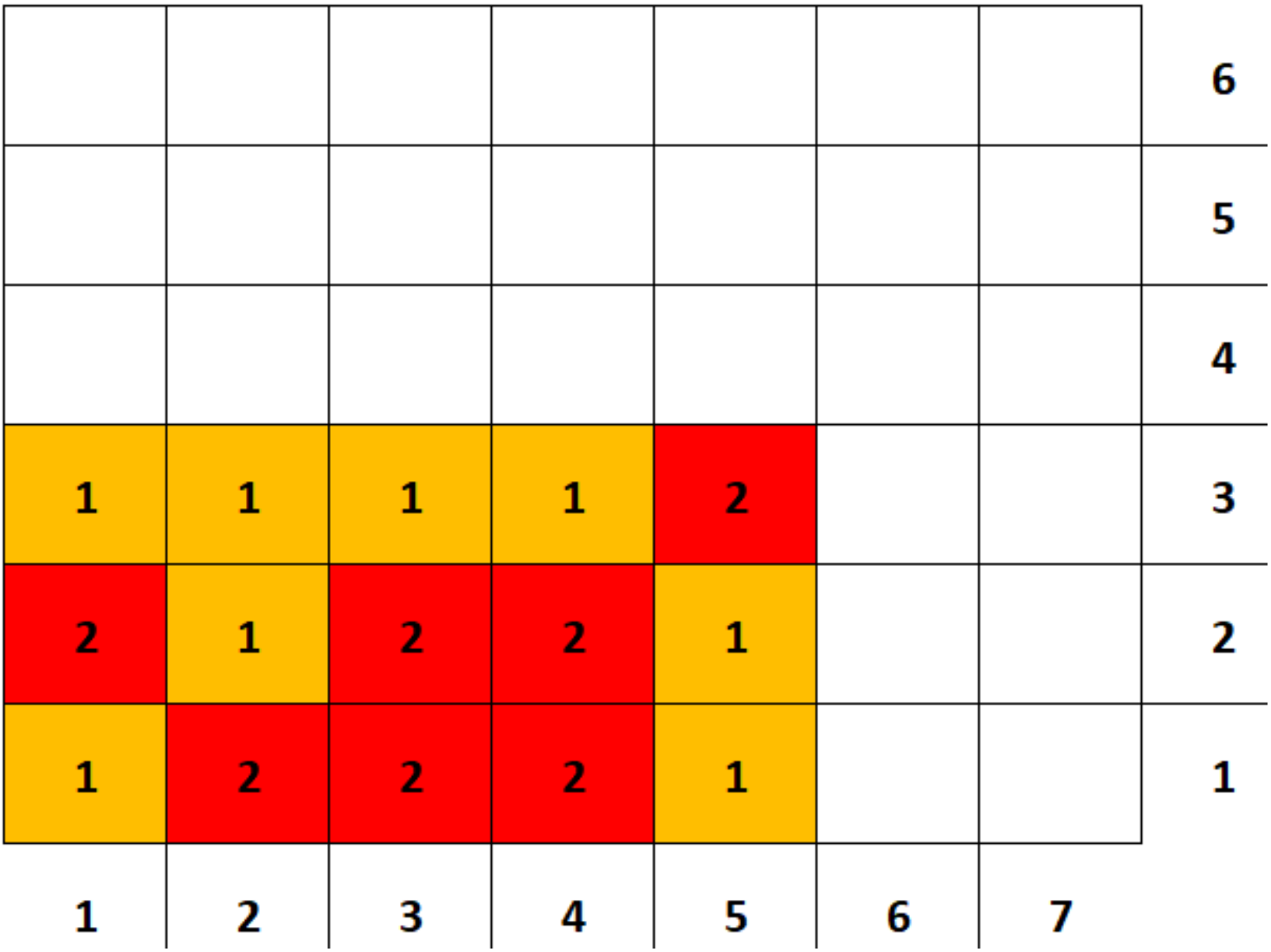


Ending a game

Players can continue alternating to drop tokens into board until either...

- one player has a 4 or more token line (hence the name **connect 4**)
    - this player then wins
- there are no more spots in the board
    - this becomes a draw

Some examples of win states:



Player 1 (orange) has a horizontal win in row 3, columns 1-4 (all pieces in these spots are orange)

2							6
1	2			1			5
1	1	1		1		2	4
1	1	2	2	2	1	2	3
2	1	2	2	1	1	2	2
1	2	2	2	1	1	2	1
1	2	3	4	5	6	7	

Player 2 (red) has a vertical win in column 7, rows 1-4 (all pieces in these spots are red)

Hang on... what about a diagonal line?

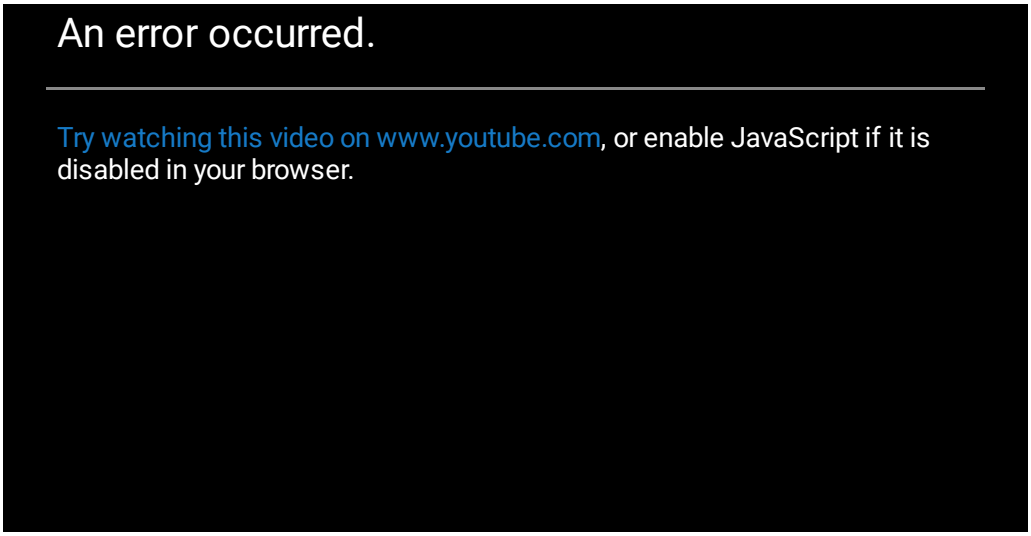
We won't be worrying about that for this implementation of the game :)

Your task

You will eventually construct a program that simulates two players playing this game but it will be broken into small stages to help you develop your program and to allow you to test your work and get feedback via automated scripts.

Video description

Aside from the fact that we will be ignoring the diagonal line win condition in this game, this video gives a good description of the game:


An error occurred.
Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Representation of the game

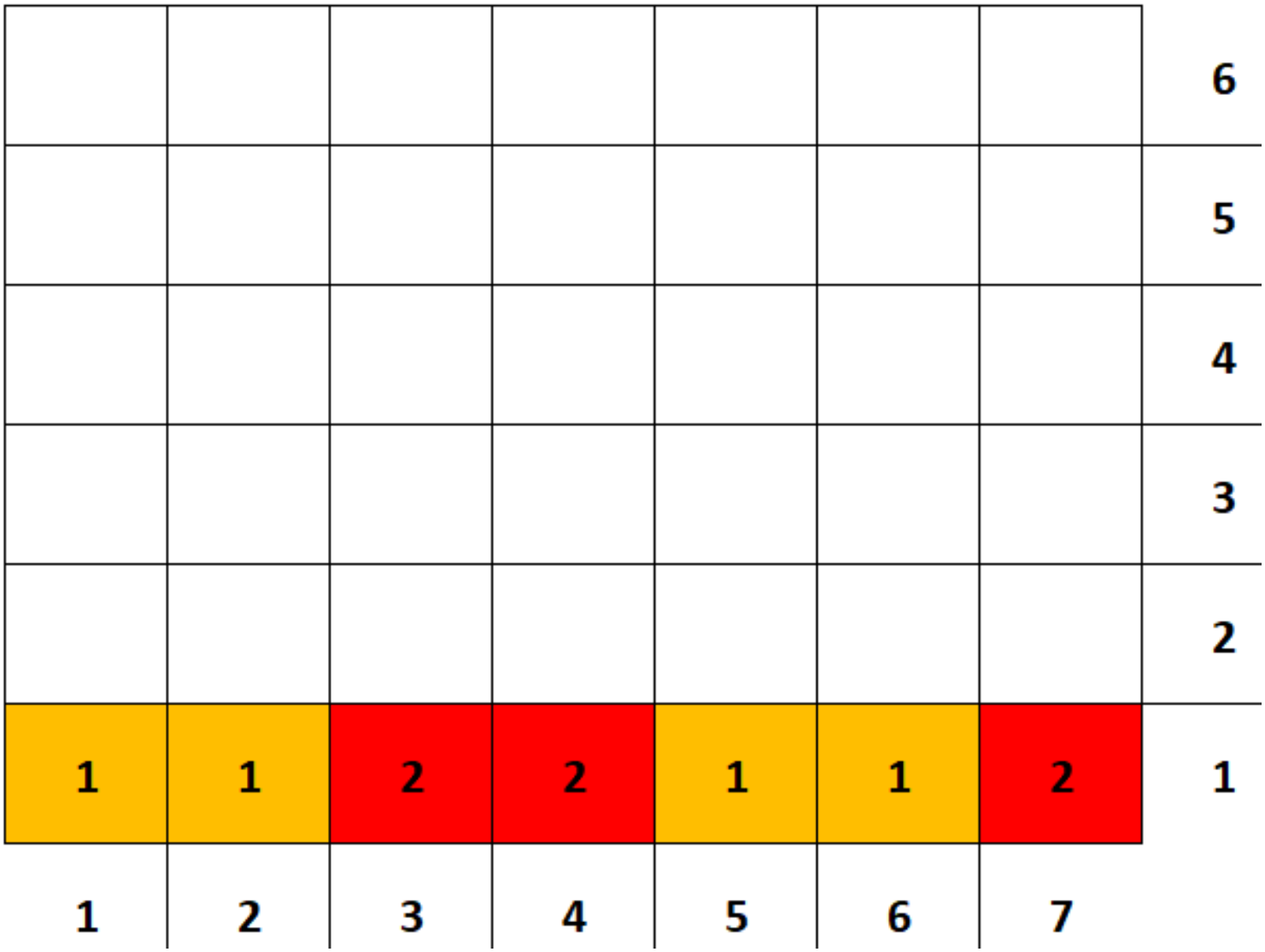
In order to play the game, we have to have a way of representing the current state of the game.

In this case we will use six 7-digit numbers to represent each of the 6 rows in the board.

Each digit represents a particular column with 0,1 or 2 representing an empty spot, player 1's token or player 2's token respectively.

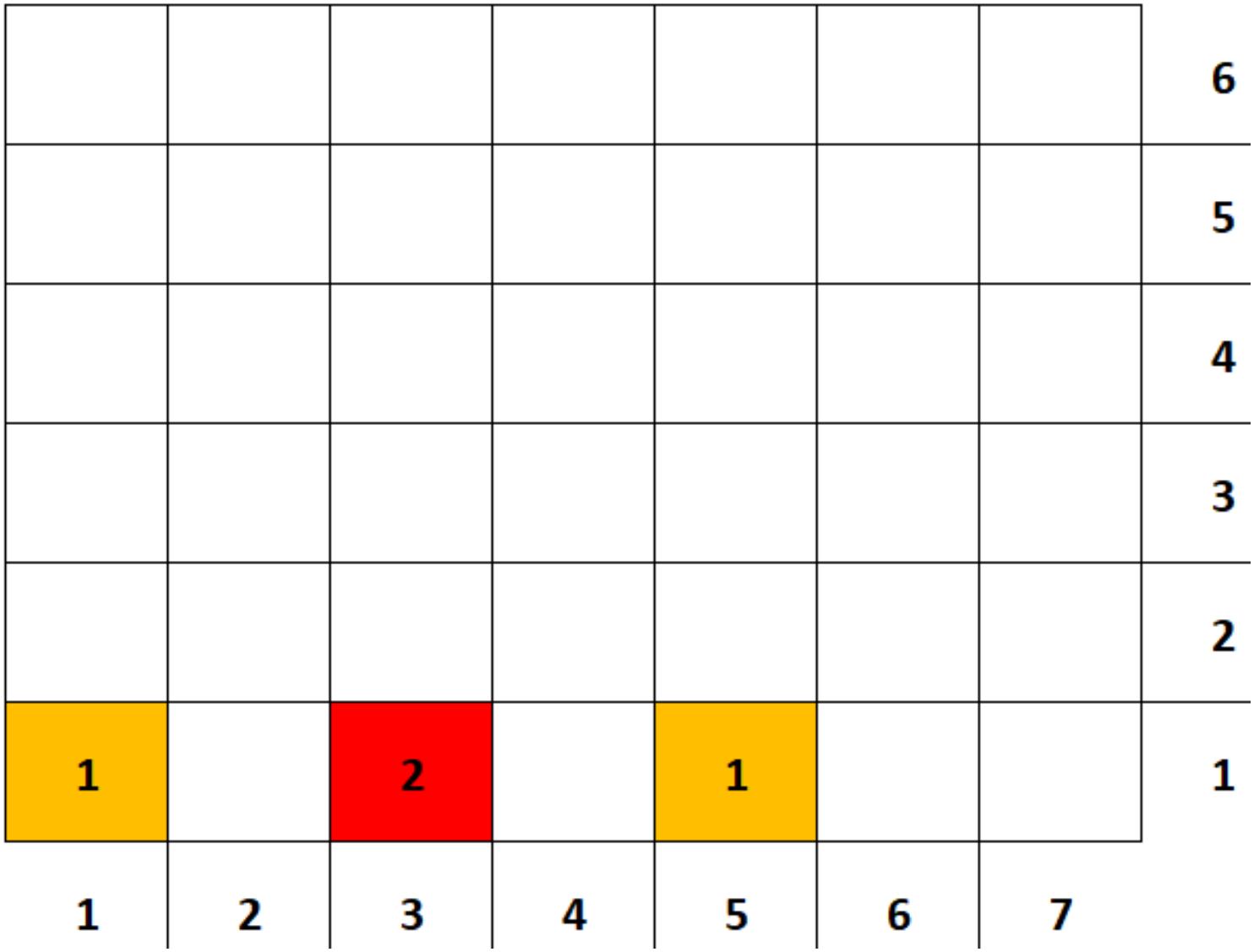
> ⚠ In your implementation, you must use a number to represent each row of the board

Some examples:



In the board above there are orange pieces (player 1) in row 1, columns 1, 2, 5 and 6) and red pieces (player 2) in row 1, columns 3, 4 and 7 and

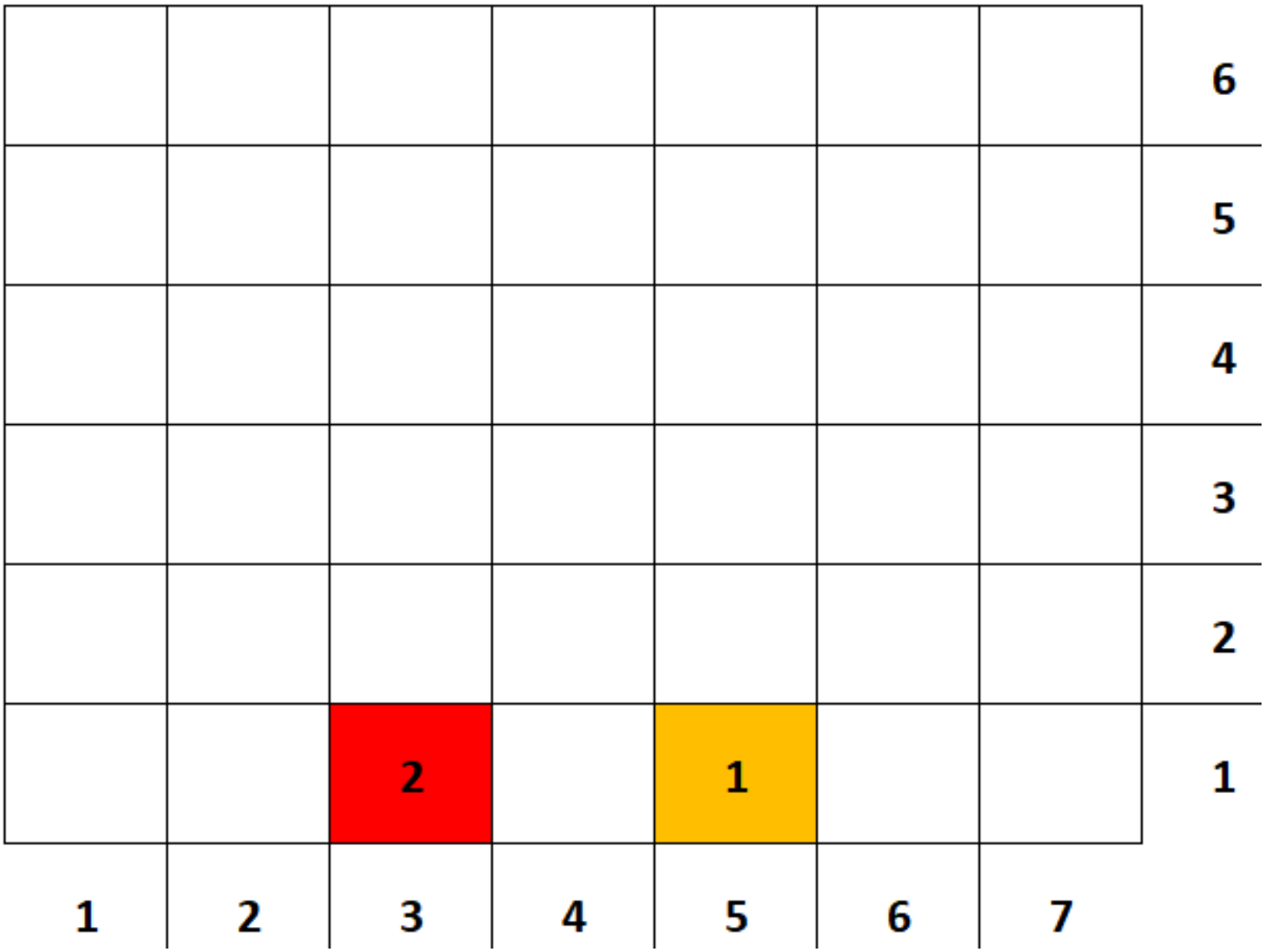
This is represented by the number 1122112 which is the same as reading the player numbers in the row from left to right



In the board above there is a red piece (player 2) in row 1, column 3 and an orange piece (player 1) in row 1, column 1 and 5) but not all the columns are filled.

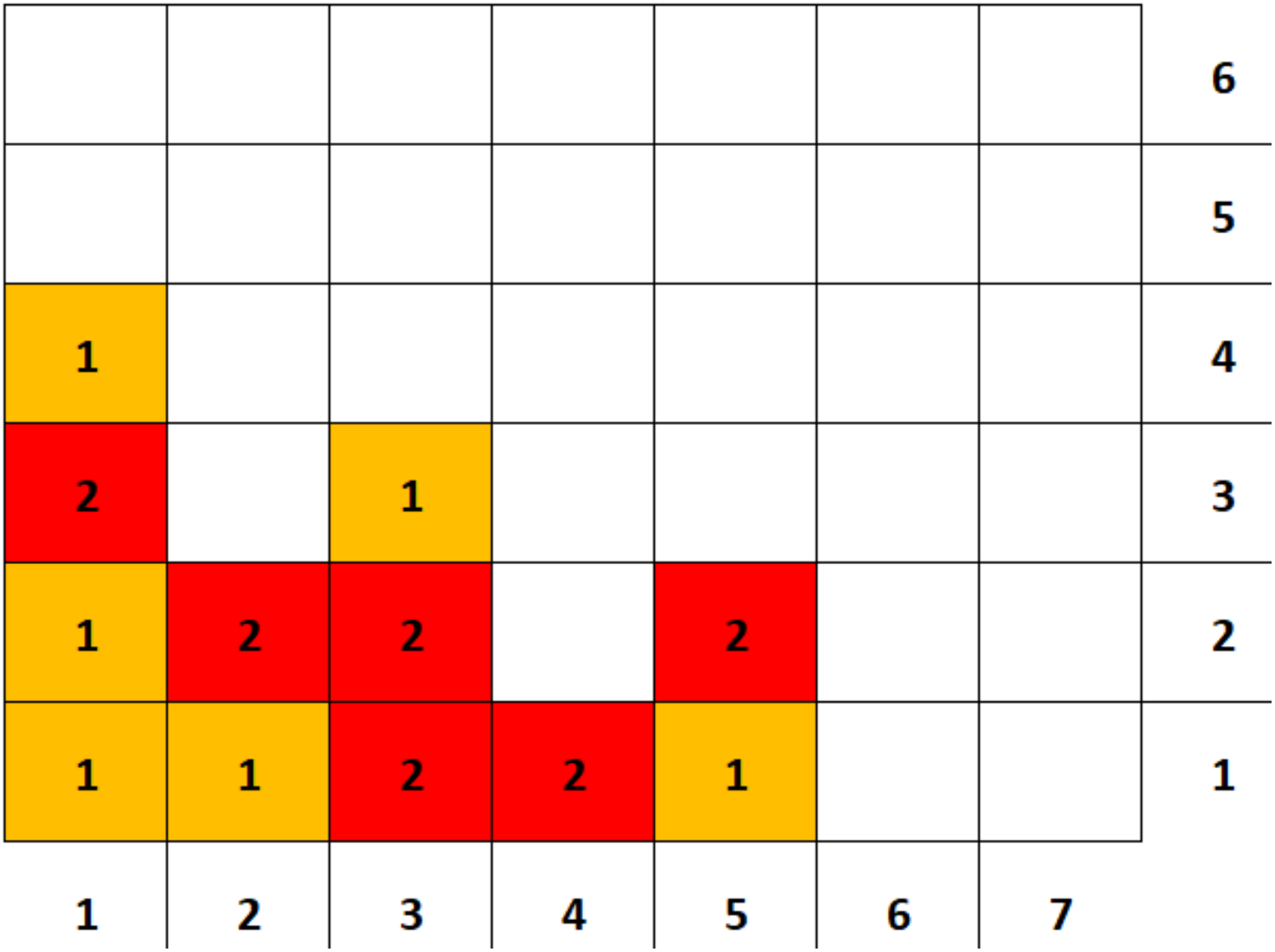
This is represented by the number 1020100 which we can also interpret as

Player1 | empty | player2 | empty | player 1 | empty | empty



In the board above you may notice there isn't a piece in position 1 but there's a red (2) and orange (1) at columns 3 and 5 respectively and all other spots are empty.

Essentially this is represented by 0020100 however whole numbers do not start with zeroes so it can simply be written as 20100



In this board, we have pieces across multiple rows now with the numbers for rows 4,3,2,1 being

1000000

2010000

1220200

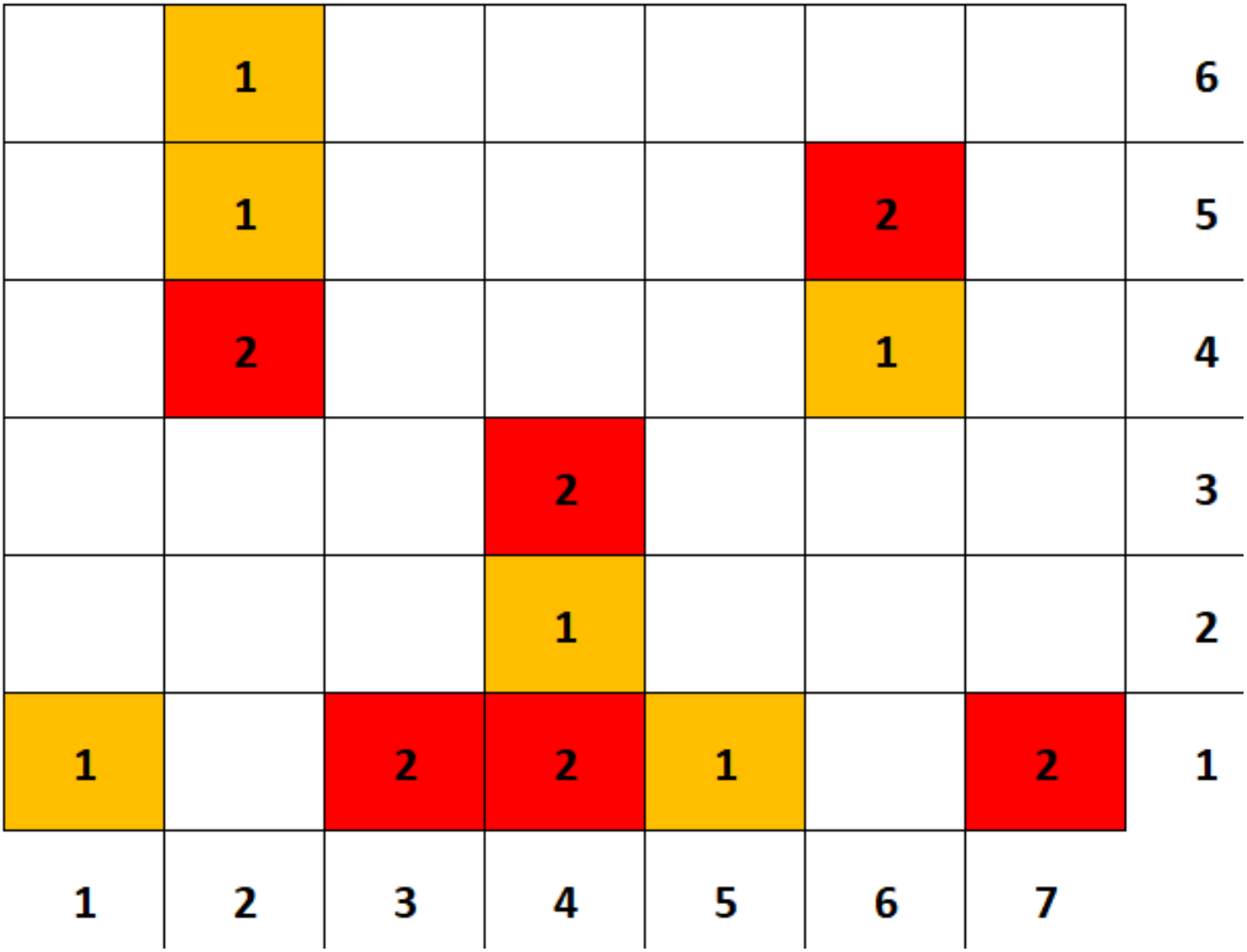
1122100

respectively.

Row 1 represents the bottom most row and row 4 is the highest which includes pieces.

Invalid board configurations

In the traditional game, pieces fall through gravity from the top of the board down to the next available position, so the following board is impossible



The above board would be represented by the numbers

0100000

0100020

0200010

0002000

0001000

1022102

In this pieces in columns 2 and 6 are suspended in the air.

Even though those in rows 5 and 6 do have a piece beneath them, there are no pieces **beneath** those in row 4

Important: Do's and Dont's

Do

- represent your board as a set of (7 digit) numbers.
- run your work regularly and test with different scenarios.
- If you find yourself writing the same line several times, test the first version **extensively** before duplicating as this will save you a great deal of effort (alternatively use a loop).

Don't

- change the board representation (you should treat them as numbers except when reading or writing them).
- the above means: do not use any list (or other data structures, like dictionaries), anywhere in your code.
- import any libraries or packages (you don't need them).
- use string operations, except for question 9.
- use a prompt string in `input()` as our marking scripts do not expect them (see samples of input and output).

1. Printing the board

Assume you have a board represented by the following numbers

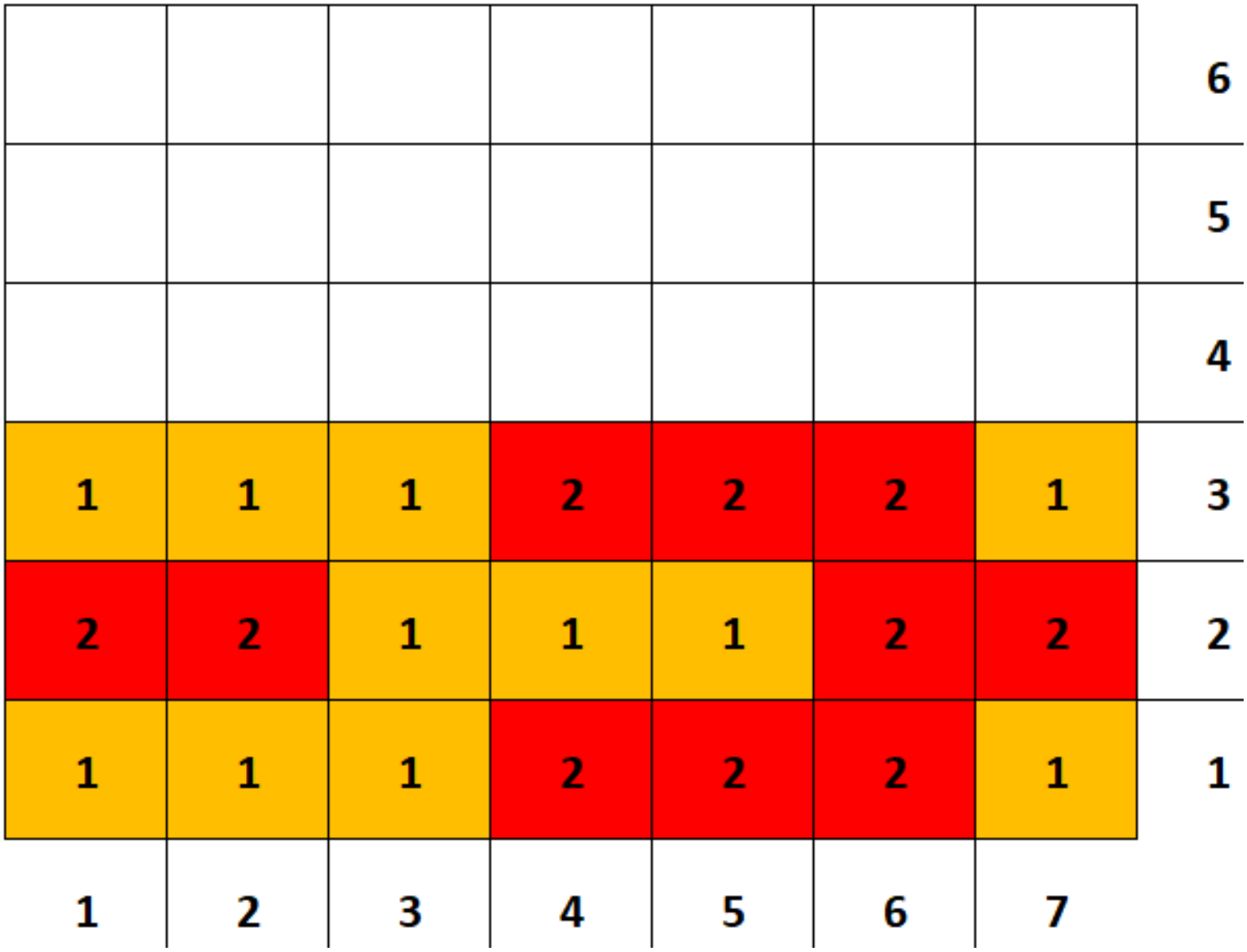
1112221

2211122

1112221

with all rows above being empty

i.e.



Your task

Write a program which

- Takes a single number as input, which represents a new row placed above the three given above.
- Prints out the current board, namely the new row and the previous three.

You will need to use...

```
input()
print()
```

You may find this helpful...

```
"\n" # a character representing a new line (allows you to write across multiple lines with one line
```

Note that the order in which the lines are printed needs to match the examples we provide in order to pass the test. This is true for all questions in the assignment.

Sample input and output

Example 1

Input

```
1100221
```

Output

```
1100221
1112221
2211122
1112221
```

2. Getting the piece in a particular column

In this task you will receive a single number representing a row and you will need to find out the piece living in column 5.

Supporting information

To do so, you will need to consider how each number is broken up into digits.

For example, for the number **12345**,

we have **1** ten thousands, **2** thousands, **3** hundred, **4** tens, and **5** ones.

this can be written as

$$12345 = 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

If we wanted to get at the second digit, there are a few ways to do so but the easiest option would be to perform an integer division (//) of the number by 1000 which will leave us with just the quotient ("12") and then taking the remainder (%) after dividing by 10 which would give us the "2".

Your task

Write a program which

- Takes a number (using `input()`) from the user.
- Performs some arithmetic operations to leave you with just the digit at position 5 (from the left)
- Output this to the screen
- **Hint:** Don't forget, our board has 7 columns but when writing whole numbers we don't include the zeroes at the start (e.g. 201 is the same as 0000201, so the 5th digit here is 2).
- **Hint2:** You don't have to worry about numbers with more than 7 digits or those with individual digits exceeding 2.

You will need to use...

```
int()
% # gives the remainder after dividing by a number (e.g. 25 % 4 would give 1)
```

You may find these helpful...

```
// # gives the quotient after dividing by a number (e.g. 25 // 4 would give 6)
other arithmetic methods (e.g. multiplication, exponentiation, subtraction, etc.)
```

Sample input and output

Example 1

Input

```
1000201
```

Output

```
2
```

Example 2

Input

```
2222021
```

Output

```
0
```

Example 3

Input

```
2021
```

Output

```
0
```

3. Getting the piece in any column

Previously we looked at how we can get at the digit in position 5, we now want you to extend this to work with any chosen digit.

You will receive input for both the column to look at (first input) and the number represent the row to extract from.

Your task

Write a program which

- Takes a column index from the user (C).
- Takes a board row (7-digit number) from the user.
- Performs some arithmetic operations to leave you with just the digit at position C (from the left).
- Output this to the screen.
- **Hint:** you don't have to worry about numbers with more than 7 digits or invalid column numbers.

Sample input and output

Example 1

Input

```
1
1000201
```

Output

```
1
```

Example 2

Input

```
5
2222021
```

Output

```
0
```

Example 3

Input

```
7
2021
```

Output

```
1
```

4. Adding a piece to a particular row

Now we want to replicate adding a new piece into an existing row.

You will be given a row representing the bottom of the board (you can assume all rows above are empty) as well as the player currently making a move and the column they choose to play the piece in.

You will need to use your knowledge from the previous tasks to include this new piece inside the row representation.

Your task

Write a program which

- Takes the following inputs (one at a time, in the following order):
  - a row (represented as a 7-digit number), assumed to be a valid row,
  - a player number, assumed to be 1 or 2,
  - a column number, assumed to be between 1 and 7.
- If the move is impossible, print "invalid move".
- Otherwise, prints out the current board (assuming all rows above are empty).

You will need to use...

```
your code from the previous steps
```

Sample input and output

Example 1

Input

```
1000221
2
4
```

Output

```
1002221
```

Example 2

Input

```
221
1
3
```

Output

```
10221
```

Example 3

Input

```
221
1
7
```

Output

```
invalid move
```

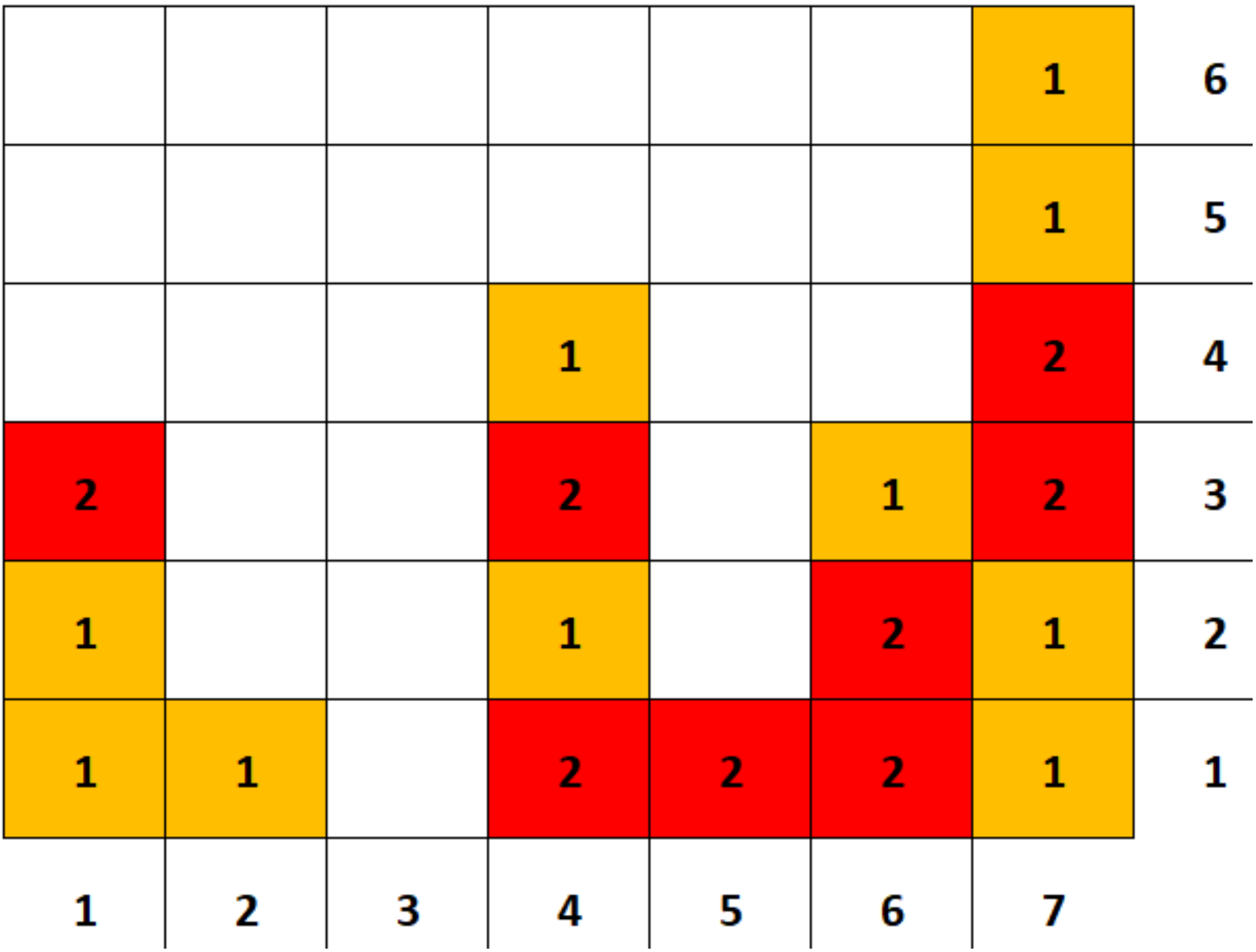
5. Finding the row to add a piece to

We currently have code that can add a piece to a given row, let's work on figuring out which row the piece should go into.

Supporting information

As you may recall from the game description, pieces are placed in any column (with room) and land atop any existing piece in that column

Imagine you have a board as shown below.



This is represented by the rows (from top to bottom):

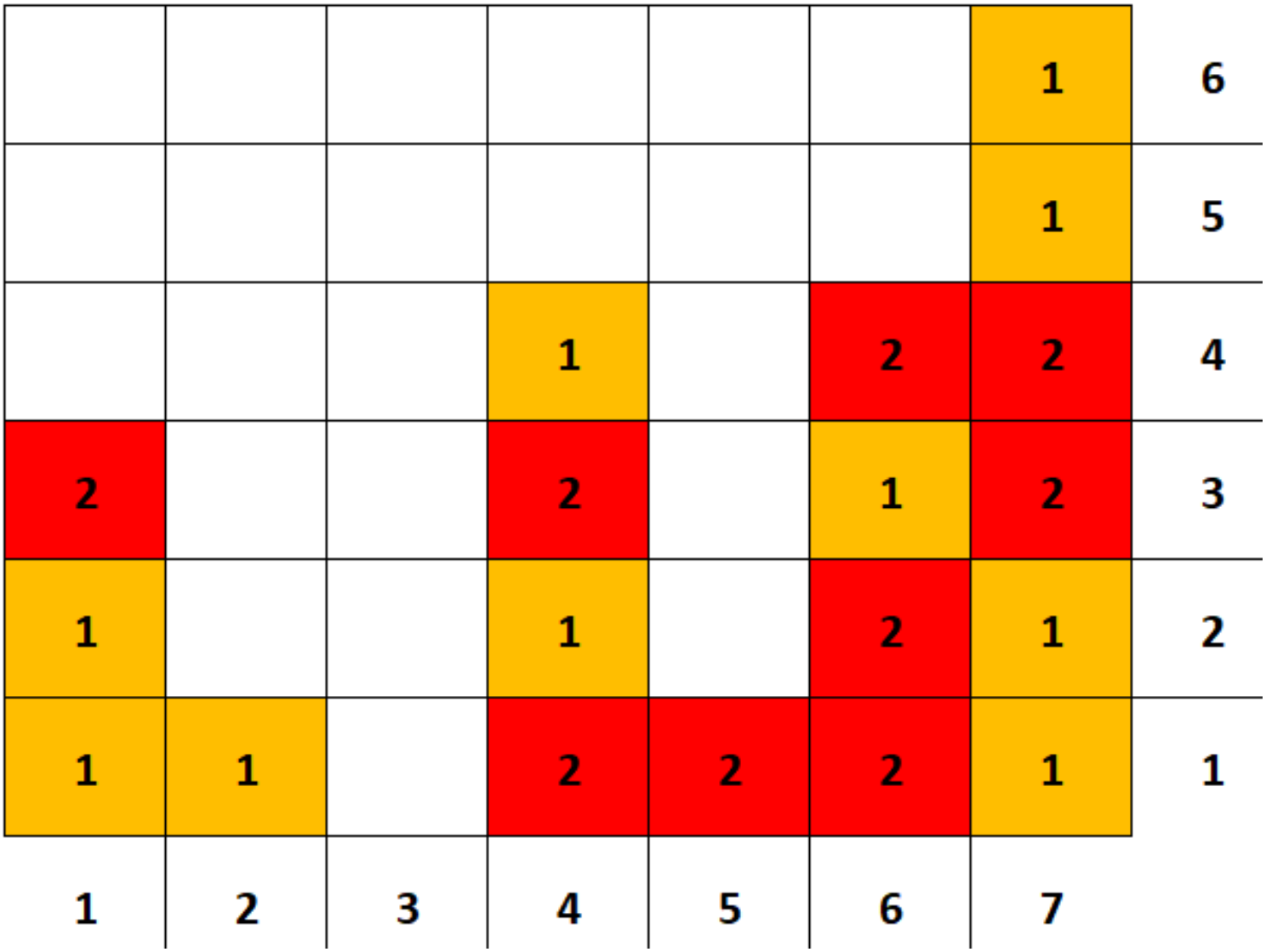
```
0000001
0000001
0001002
```

```
2002012
1001021
1102221
```

In this instance, a player can play a piece into any of columns 1-6 (column 7 is already full).

For columns 1-6, the valid row positions are: 4,2,1,5,2,4 respectively.

For instance, if player 2 dropped a piece in position 6 it would end up at row 4 as shown below.



This is represented by the rows (from top to bottom):

```
0000001
0000001
0001022
2002012
1001021
1102221
```

Your task

Write a program which

- Takes the following inputs (one at a time, in the following order):
    - a column number,
    - up to 6 rows (represented as a 7-digit numbers) holding the current state of the board.
- Prints out the row number with room for a piece as soon as it is known.
- If there's no room in that column print "no room in column X".

If your program receives less than 6 rows, you should assume the rows beneath are completely full

How to test this task

you can copy paste the sample board from this slide and paste it into the terminal to pass each line one at a time as input. You can also prepare your own boards in a similar way (e.g. in a text editor / spreadsheet)

You will need to use...

```
your code from the previous steps
```

Sample input and output

Example 1

Input

```
7
0000001
```

Output

```
no room in column 7
```

Example 2

Input

```
4
0000001
0000001
0001022
```

Output

```
5
```

Example 3

Input

```
3
0000001
0000001
0001022
2002012
1001021
1102221
```

Output

```
1
```

6. Looking for a horizontal win

You now nearly have all the pieces you need to play a game.

At this point we want to take a row and figure out if a particular player has won based on that row.

Recall that 4 pieces of the same colour next to each other represents a win for that player.

Your task

Write a program which

- Takes the following inputs (one at a time, in the following order):
  - a row (7-digit number),
  - a player (1 or 2).
- Prints out `True` if the player wins or `False` otherwise.

You will need to use...

```
your code from the previous steps
```

Sample input and output

Example 1

Input

```
1102221
1
```

Output

```
False
```

Example 2

Input

```
1122221
1
```

Output

```
False
```

Example 3

Input

```
1122221
2
```

Output

```
True
```

7. Looking for a vertical win

Almost there!

Now we want to take a board (and a particular column) and check if a particular player has won based on that column.

Your task

Write a program which

- Takes the following inputs (one at a time, in the following order)
  - a player (1 or 2),
  - a column number,
  - up to 6 rows (represented as a 7-digit numbers) holding the current state of the board. If a win should be detected before the 6th row, then fewer rows need to be read from the input.
- Prints out True if the player wins or False otherwise.
- **Hint:** your code to the previous task is very close to this already.

How to test this task

Once again, you can copy paste the sample board from this slide and paste it into the terminal to pass each line one at a time as input. You can also prepare your own boards in a similar way (e.g. in a text editor / spreadsheet)

You will need to use...

```
your code from the previous steps
```

Sample input and output

Example 1

Input

```
1
7
0000001
0000001
0000001
0000021
```

Output

```
True
```

Example 2

Input

```
1
5
0000001
0000001
0000001
0000021
0000222
1200122
```

Output

```
False
```

Example 3

Input

```
2
7
0000001
0000001
0000001
0000021
0000222
1200122
```

Output

```
False
```

Example 4

Input

```
2
5
0000210
0000120
0000210
0000210
0000210
0000120
```

Output

```
False
```

8. Changing the representation to use lists

For problems of this nature, it's very common to use lists to represent a board or similar structure.

Now we want you to look over the representation slide again and respond to the following:

Question *Submitted Mar 19th 2022 at 2:06:00 pm*

If we used lists for the representation how would you represent the board and individual rows?

(Feel free to draw a diagram if helpful)

Represent each rows and column with a list indexing like for example *column is represented with y* and *row is represented with x*. So each specific position in the board have a representation like column 7 row 3 is also known as [x_2, y_6] and column 3 row 2 is known as [x_1, y_2]. With this, it is easier to check if a position is empty or filled with a piece on the board.

But the first element in a list is always starting with a 0 and not 1. So to print the first thing in the list, one will need to type in the_list_name [0].

End of Assignment for FIT1045

The questions below are not assessed for the FIT1045 cohort. But we encourage you to work on them for your own learning.

9. Padding a number with zeroes

THIS IS AN OPTIONAL QUESTION FOR FIT1045. NO MARKS AWARDED FOR FIT1045.

This task extends your printing of a board to capture cases where the number is less than 5 digits (again, leading zeroes are not included when displaying a whole number).

Supporting information

A 5 digit number can also be seen as a 7 digit number with leading zeroes.

We take again the example of the number 12345.

In our earlier case we showed this as

$$12345 = 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

But we could also equally say

$$12345 = 0 \times 10^6 + 0 \times 10^5 + 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

Your task

Write a program which

- Takes a number (with `input()`), which may have fewer than 7 digits.
- Performs some string operations to pad the left side with as many zeroes as needed to become 7 digit.
- Output this to the screen.

You may find these helpful...

```
condition statements (e.g. if (X):)
loops (e.g. while)
```

Sample input and output

Example 1

Input

```
1000201
```

Output

```
1000201
```

Example 2

Input

```
22021
```

Output

```
0022021
```

Example 3

Input

```
1
```

Output

```
0000001
```

10. Verifying a row is valid

THIS IS AN OPTIONAL QUESTION FOR FIT1045. NO MARKS AWARDED FOR FIT1045.

In order to do some of the checking you'll need to for later tasks, we'll get you at this stage to take a row and make sure it meets the requirements for representing our board.

Please review the "Representation of the game" slide if you need a reminder.

Your task

Write a program which

- Takes a number (with `input()`). In this question you can assume the input is an integer.
- Performs arithmetic operations to confirm the received number represents a valid row in the board.
- Output the result ("valid" / "invalid") to the screen.

You may find these helpful...

```
condition statements (e.g. if (X):)
loops (e.g. while)
```

Sample input and output

Example 1

Input

```
1000201
```

Output

```
valid
```

Example 2

Input

```
22321
```

Output

```
invalid
```

Example 3

Input

```
1
```

Output

```
valid
```

Example 4

Input

```
-1
```

Output

```
invalid
```

Example 5

Input

```
1212100121
```

Output

```
invalid
```

11. Changing the representation to use lists

THIS IS AN OPTIONAL QUESTION FOR FIT1045. NO MARKS AWARDED FOR FIT1045.

Now review how you've approached the task over each of the sub parts (1-10) in light of your response to question 8.

Question

Assuming you represented the board as you describe, how would you have to change your code in previous tasks?

(aim for a one-paragraph explanation)

No response

End of Assignment for FIT1053

The questions below are not assessed for any cohort. But we encourage you to work on them for your own learning.

12. Final game (simplified for testing)

THIS IS AN OPTIONAL QUESTION. NO MARKS AWARDED.

With that we now have all the pieces we need to run the full game.

Note: This will be the last stage with automated testing.

Prepare a program which will alternate turns between players 1 and 2 until one of the following conditions are reached:

- One of the two players achieves a win (4 pieces in a vertical or horizontal line),
- There are no spots left on the board,
- Invalid move or user input,
- A period (full stop) is given as input by the user.

A useful consideration for this is that only the most recent move can result in a win (e.g. if 5 pieces have already been played, any win at the next step will involve piece 6 either vertically or horizontally as the previous pieces will already have been checked).

Your task

Write a program which

- repeats until the game ends (win or draw) or the user enters a period (".")
    - takes a column position as input,
    - attempts to place in that column,
    - checks for a win,
    - swaps to the next player.
- If the game ends display the board and print the outcome ("draw", "player 1 wins", or "player 2 wins").
- If invalid user input is received, end the game, display the **last valid** board state and print "invalid move".
- If the user enters a period, end the game, display the board and print "HALT".

How to test this task

You can copy paste the move sequence from this slide and paste it into the terminal to run a long sequence of operations with a single copy-paste. You can also prepare your own boards in a similar way (e.g. in a text editor / spreadsheet).

You will need to use...

```
almost all your code from the previous steps
one or more while loops
conditional statements
```

You may find it helpful to...

```
use previous program's input validation --> this will help isolate bugs to particular areas
```

Sample input and output

Example 1

Input

```
5
4
6
5
.
```

Output

```
0000000
0000000
0000000
0000000
0000200
0002110
HALT
```

Example 2

Input

```
5
5
5
8
2
3
```

Output

```
0000000
0000000
0000000
0000100
0000200
0000100
invalid move
```

Example 3

Input

```
5
5
5
6
6
7
5
7
5
4
5
```

Output

```
0000100
0000100
0000100
0000100
0000212
0002122
player 1 wins
```

Example 4

Input

```
3
4
4
5
3
6
4
7
1
1
1
```

Output

```
0000000
0000000
0000000
0001000
0011000
0012222
player 2 wins
```

Example 5

Input

```
7
7
7
7
7
7
5
6
6
6
6
6
6
4
5
5
5
5
5
4
4
4
4
4
3
3
3
3
3
1
2
2
2
2
2
2
1
1
1
1
1
3
```

Output

```
1222112
2111221
1222112
2111221
```

```
1222112
2112121
draw
```

13. Final game

THIS IS AN OPTIONAL QUESTION. NO MARKS AWARDED.

Now take what you've done (and fully tested) through the previous task extend it as you see fit to get it to replicate the original game

Some recommendations

- display the board after every move
- respond gracefully to invalid inputs (tell the user what's wrong and re-request until they resolve the issue)

Testing your work:

Since you are using your discretion here, it's not possible to provide a general testing strategy but broadly you may wish to read up on blackbox testing strategies (essentially things like boundary/edge cases, different classes of valid and invalid cases, etc.) and try these by making moves and observing the effects on your board.

Change log

14/03 at 17:05
Changed the last column of the rubric (on documentation) to be more lenient and clear.