HW9: PROJECT DESCRIPTION

A. Goals

This is a closed-ended project that provides you the opportunity to apply everything you have learned in CIS 110 to one of the options below. You will be responsible for choosing one of the four given problems, analyzing that problem to determine how to solve it, decomposing the functionality of your program to identify its class structure and the API for each class, and then implementing and testing your program. You will **not** be given any skeleton code for this project. You must write your own files and make the best design and data structure choices as you see fit.

You are **not** permitted to work with a partner on this project.

Start early and reach out to the Instructors/TAs if you have any questions or concerns.

You may be tempted to use code directly from the internet or use such code as a base template. Please do not do so. We are extremely adept at catching such cases and strict action will be taken.

B. Project Requirements

In this assignment, you will implement 1 of 4 games below. Your project must use **object-oriented** design. If you implement your entire project as static, with no object instances, you will lose significant points. Each project has their own requirements enumerated at the end of their writeup.

C. Project Ideas

Your project must implement one of the following 4 ideas. You should look at the individual tabs for more information. You may NOT suggest an alternative idea. You must implement one of the 4 options below.

1. Connect4 - implement the game Connect4, with both a 2-player mode and a player vs. computer mode.
2. 2048 - implement 2048 game using keyboard controls.
3. 6x6 Sudoku Solver - Write a solver for 6x6 Sudoku problems

4. Minesweeper - a classic puzzle game where you have to be careful where you click! Implement the beginner difficulty of minesweeper.

Sudoku Solver

In this project, you will implement a 6x6 Sudoku Solver. You can practice solving strategies by playing here.

Sudoku is a puzzle where a player must fill an **n**x**n** (typically 9) grid with numbers 1-**n** such that no single number appears twice in any row, column, or grid box. Typically, Sudoku puzzles are written such that they have 1 and only 1 solution. This means most Sudoku's can be solved by deductive reasoning.

The input sudoku table will be passed as a text file. For example, this text file represents the Sudoku puzzle below.

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

This file should be passed in via **command line arguments**. Each digit is used for a square that is filled in at the start, while a whitespace character `' '` is used for an empty square. Your program should produce meaningful error message if the input is incorrectly formatted (such as not 6x6 characters, using characters besides whitespace and digits 1-6, or an illegal grid, such as two 4s in the same row as a starting point.).

Note that in 6x6 Sudoku, the grid boxes are all 2 rows by 3 columns. A single digit cannot appear twice in a bolded box.

Your program would solve this sudoku and output the following text to the console

:

```
162345
435216
```

```
516432
243561
654123
321654
```

If you cannot solve the Sudoku puzzle, simply print that you cannot solve the Sudoku, and print the grid at the time you get stuck.

Tips

Most obviously, if you are not familiar with Sudoku, I do not recommend doing this project. That said, 6x6 Sudoku is **substantially** easier than 9x9 Sudoku, and is a great entry point into Sudoku.

Do NOT attempt a 9x9 solver. A 9x9 solver requires advanced techniques, where any solvable 6x6 can be solved explicitly via deduction. Further, if you submit a 9x9 Sudoku Solver, you will receive a zero for the assignment.

Use deductive reasoning: if a given square can ONLY be 1, then set it equal to 1. If a given square in a row is the only possible square that can hold 4, make that square 4.

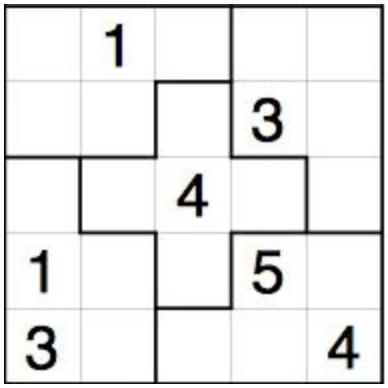
Do not use guesses, this should be solved deterministically strictly through deduction.

Try to plan out your program's structure and functionality in advance so you can define helper methods and avoid duplicative code. Don't repeat yourself!

Your program does not need any graphic component, though you are welcome to write one.

Additional Challenge (not extra credit, just a fun way to challenge yourself):

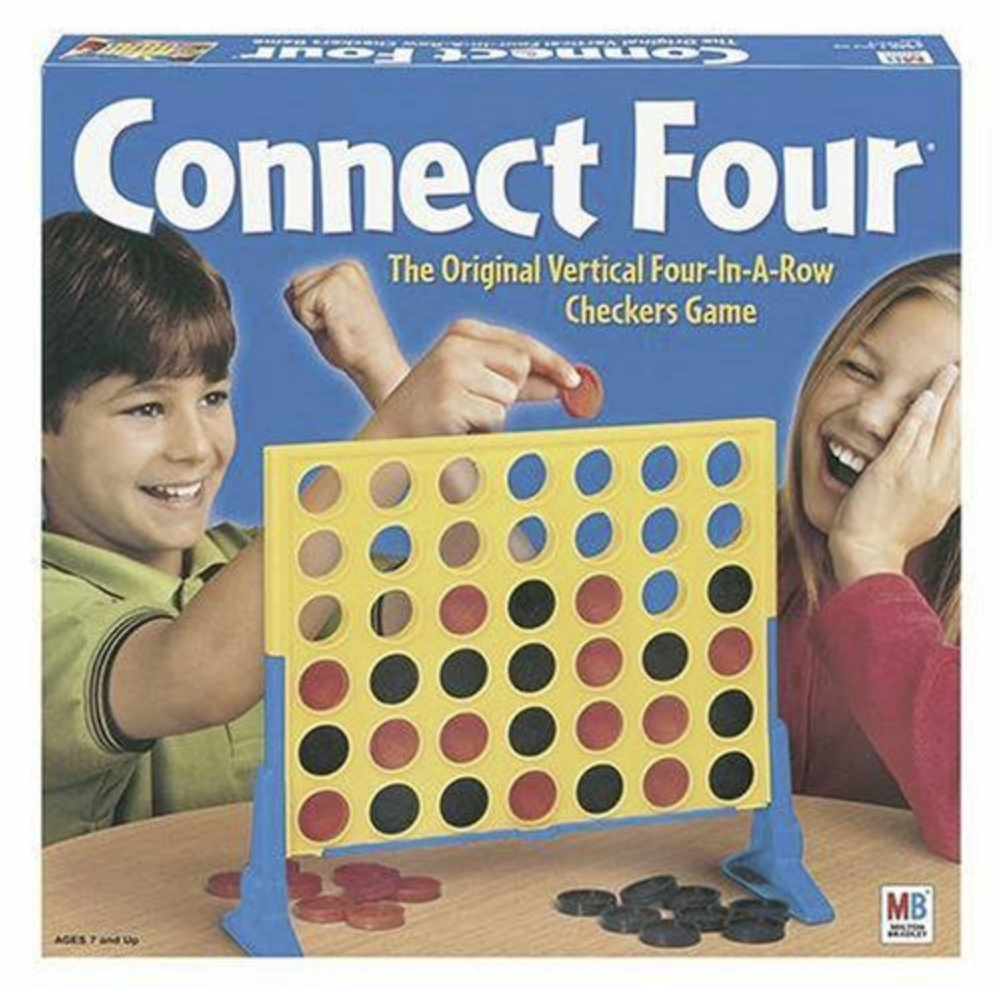
Write a SEPERATE program that does 5x5 Sudoku:



The awkward shape of each section makes this seem more challenging, but if you designed your 6x6 solver well, you may find it's easier to adapt that approach to a 5x5 solver.
Make sure what you submit is just a 6x6 solving program!

Connect 4

Connect 4 is a classic board game where two players (typically "red" and "yellow" or "red" and "black") take turns dropping disks in a grid of 7 columns by 6 rows. The discs fall to the lowest empty space in their column (possibly landing on top of other pieces). The winner of the game is the first to have 4 discs of their color in a row. This can be horizontally, vertically, or diagonally.



If you are unfamiliar with the game, this video should help out. It talks about how Connect Four is a solved game, and if you played against a correctly programmed computer who goes first, you will always lose.

In this program, you must program a Connect 4 game that uses mouse clicks as input and has both a 2-player mode and a 1-player mode. When your program starts, the game should have some start screen where the player can select either 2-player mode or 1-player mode using mouse clicks.

Players will interact in the game using mouse clicks. If a player clicks on a column, it should drop a piece into that column if possible. However, if the column is already full, the player cannot drop a

piece in that column, nothing should happen (your program must not crash here). Once a player has successfully played a piece, their turn is over. The game should check if the move created four-in-a-row of the same color. If it has, the game is over, and your game should print the winner. If not, it should simply pass the turn to the other player.

If all 42 spaces in the grid are filled, the game is a draw, and no one wins. Below is an example of a draw.

In the **2 player mode**, players will use mouse clicks to select which column they wish to drop their piece in and end that player's turn. The game should then prompt that it is the other player's turn. The game should clearly indicate which player's turn it is using their disc color (for example, you could have text saying "It's red's turn"). The turns should go back and forth until the game ends.

In **1 player mode**, the player can play against an automatic opponent that you program. This does not have to be a "good" AI, just some process by which moves are taken automatically without player input. The game should randomly decide to have either the human or the computer go first (since moving first is a big advantage). From there, the game proceeds until it ends.

In either case, when the game ends, the game should show some text indicated either who won or that the game was a draw. The game must then return to the "start" screen, where a player can once again choose either 1-player mode or 2-player mode.

Requirements:

- In this program, you must program a Connect 4 game that uses **mouse clicks** as input and has both a 2-player mode and a 1-player mode. You must use mouse clicks!
- The game must follow the rules of Connect 4
    - Players alternate turns
    - Players cannot place discs in full columns
    - A 4 in a row ends the game and says who wins
- Must have a menu on startup to select 1 or two player mode.
- Must have a two player mode where two human players take turns using mouse inputs
- Must have a 1 player mode against an AI opponent

Tips

Do not spend a lot of time working on the AI until after your program is working. You will not get extra credit if your AI can beat the instructor, though it can be fun to try. Focus on getting the game working first with 2 players.

Try to implement an object to represent the board first. Consider what functions you'll need, like "addMove" and "isGameOver". Implement and test this before you try to add a user interface.

Look back to HW01's StampSketch.java if you need a refresher on how to get the location of a mouse click.

Consider what objects you can use, as well as what functions you can use, to represent the board.

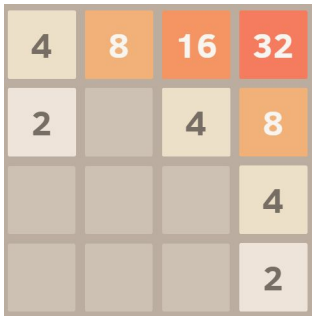
Additional Challenge (not extra credit, just a fun way to challenge yourself):

Try to add an AI to the game that most people can't beat! This is a fun project that can be rewarding and difficult, but will really get you into the thought process of logic. Fun fact: Connect 4 is a solved game. That is, if the first player moves correctly, they will never lose. But this is quite difficult.

Implement moving graphics, where the pieces drop and make a sound effect when they land!

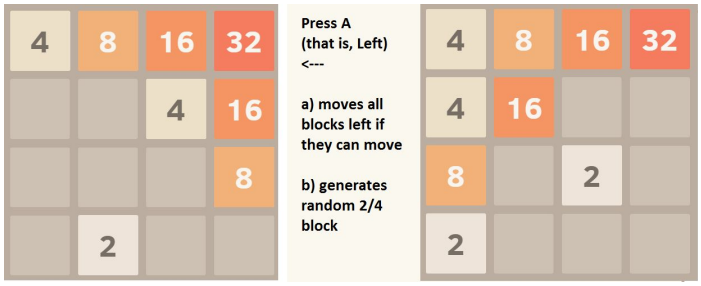
2048

For this option you will be implementing the highly addictive 2048 game. Here is a link to a description of the game.

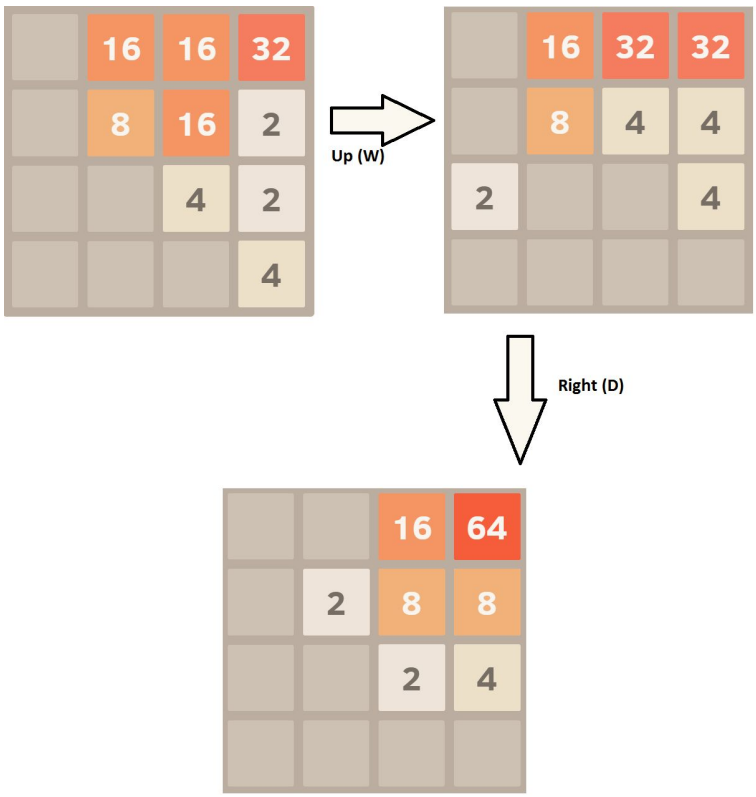


The best way to understand the game is to play it yourself. You can find an online version here.

Your game should have a 4x4 grid. Using WASD controls (W is up, A is left, D is right, S is down). Pressing in a direction moves all blocks in that direction. Example:

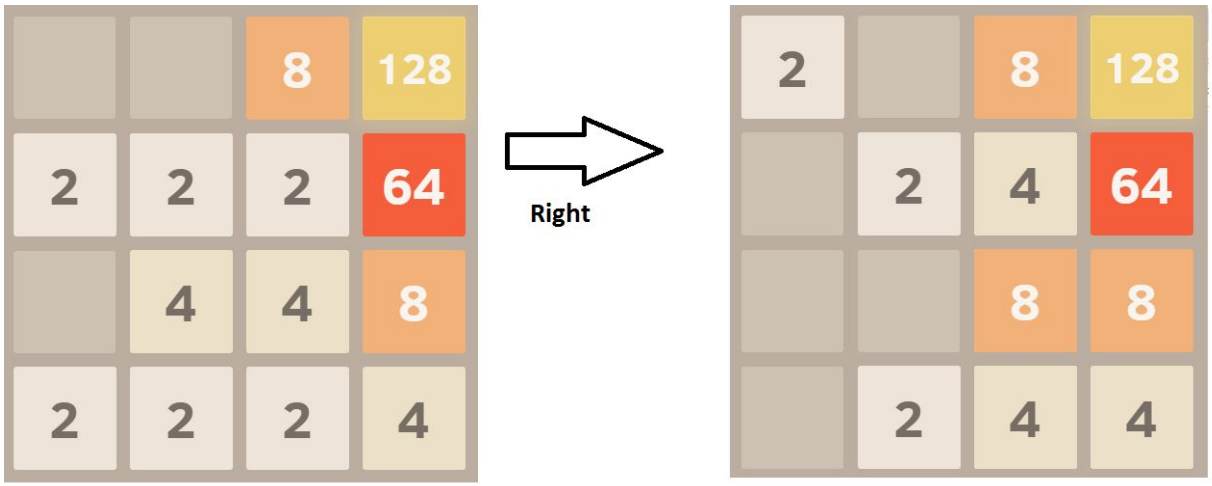


Like numbers can be "combined". That is, if two numbers "crash" into each other, they combine to be the sum of the two numbers. For example, two "2" blocks would combine to be a "4", and two "4" blocks would combine to be an "8". Example:

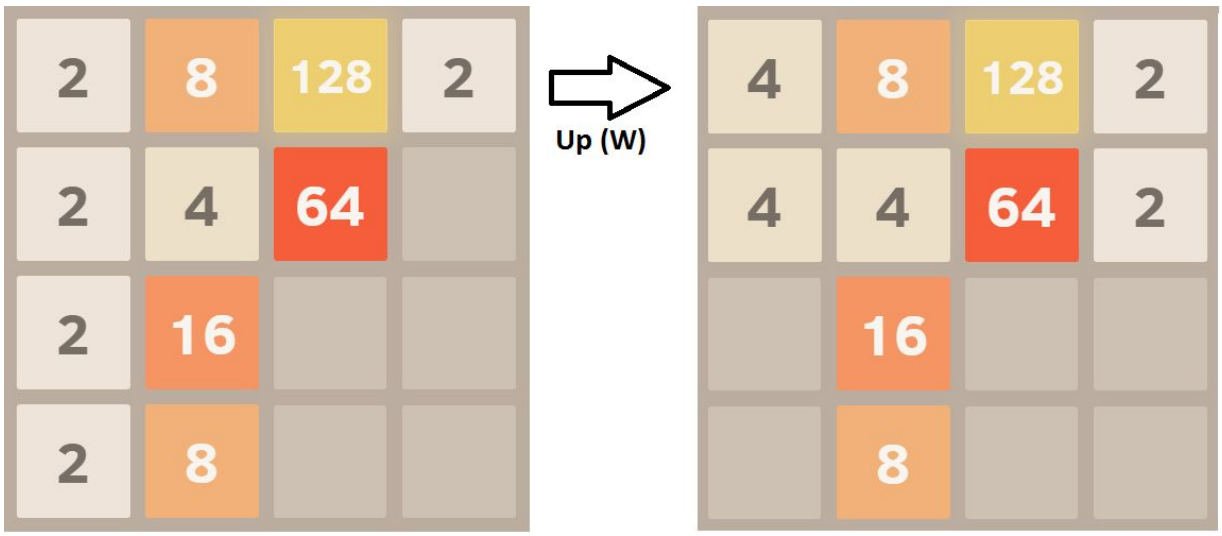


There are two special cases worth considering, 3 elements in a row or column and four in a row or column.

Example of 3 in a row:



Notice that the two "2" blocks closest to the direction of movement (right) are the two that combine. The third (leftmost) "2" in the bottom row and the second row down do not get combined.



Notice in the case of 4 matching "2" blocks in a row, both pairs combine into "4" blocks. However, the 4 blocks themselves wouldn't combine unless the player hit up a second time.

Requirements:

- The user's screen should be a 4x4 grid. Each square on the grid either contains a number tile or not.
    - You are not required to match any aesthetic style. I.e., the number colors are not required. However, the numbers should be visible within each cell, and should be easily legible.

- The user presses one of four keys - w, s, a, d which represent the directions up, down, left, right. All the current tiles on the board shift to the direction specified by the user's key press. If a number tile "crashes" into another tile with the same number - then they merge to form a single tile with twice the value.
- During every turn, a new number tile consisting of 2 or 4 should enter the board. The tile must enter at any random, unoccupied position on the board. Note that this deviates from the official “rules” of how a new tile is placed.
- If the grid is full of number tiles and there is no move possible for the user then the game is over. If the user has managed to bring a 2048 tile onto the board (by merging 2 1048 tiles) then the user wins.
    - Do not end the game if there are possible moves, even if the board is full!
- The game should keep track of the number of moves made by the user and output this in the victory/defeat message.

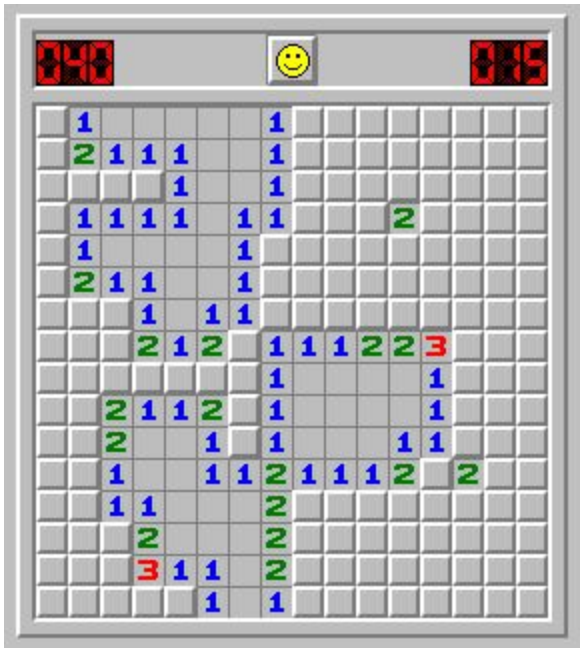
Additional Challenge (not extra credit, just a fun way to challenge yourself):

You are not required to show the animation (that is, the blocks sliding). However, as enrichment, it can be a fun challenge to add that.

Try adding the color gradient shown above, where each number has a unique color.

Try writing a version of the game that plays itself, using some AI to play as optimally as possible. If your AI can routinely get 2048, let us know!

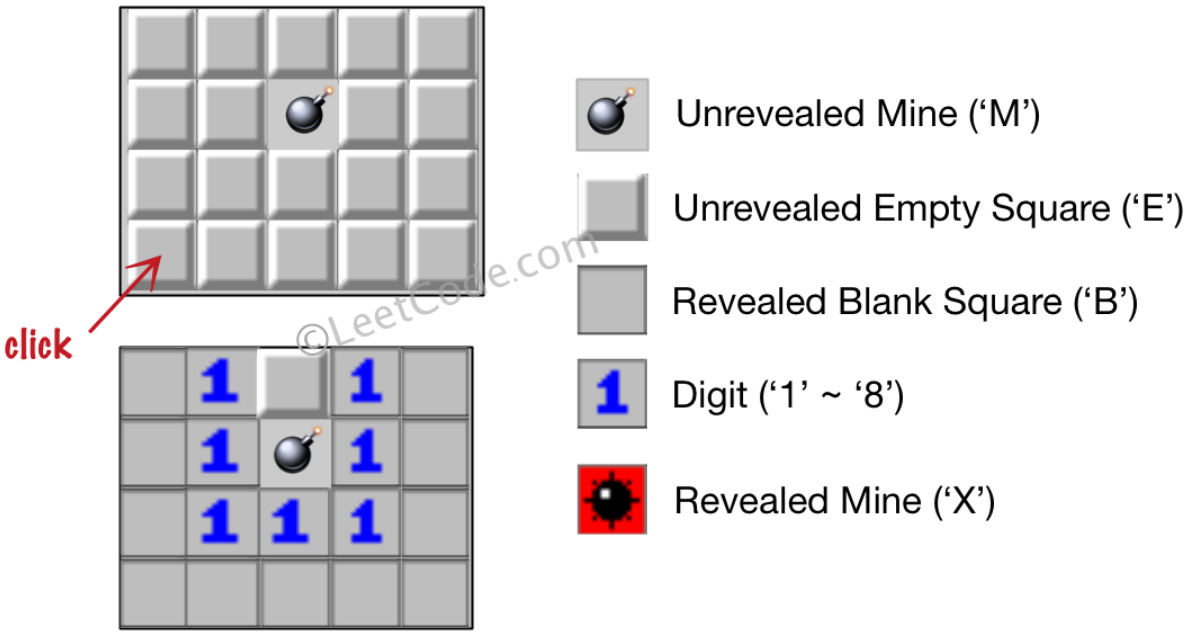
Minesweeper



Minesweeper, soon to be a major motion picture, is a classic puzzle game (which can be played here). In the game, a single player tries to sweep a minefield to clear out every space that isn't a mine. Each space is either **blank**, a **number space**, or a **mine.** The contents of each space are initially hidden. By clicking on a space, you reveal it.

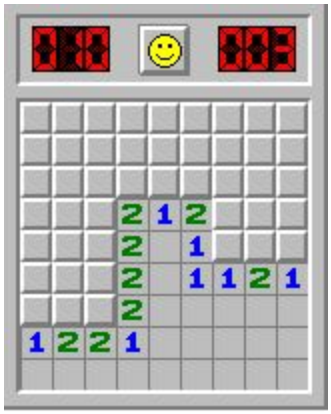
A mine is a square that if you click on, you lose the game. You can avoid clicking on mine squares by using the hints provided by **number spaces**.

A number space indicates that there is a mine either left, right, up, down, or diagonal of a mine. Clicking on a number square reveals that square and that square only. Example, a 1 means there is 1 mine nearby. The maximum possible digit is 8, which would mean all 8 squares around the space are mines.



A blank space is like a numbered square where the number is zero. That is, there are no mines in any of the adjacent 8 spaces. If you click on a blank space, it reveals both itself and all adjacent squares. If any of the adjacent squares are blank, it should further reveal all adjacents squares. For example, in the below game, when I click on the bottom right corner, I got about a

third of the board shown:



The player wins the game if they click on every space that isn't a mine. However, if the player clicks on a single square that is a mine, the player loses the game.

Below is an example of a completed minesweeper game.

The rules for your implementation:

1. The game must be implemented in PennDraw.
2. The board must be 9x9 (That is, the same as beginner difficulty).
3. There must be exactly 10 mines
    a. The mines must be placed randomly, they should be in a different place each game
    b. The mines should not be visible until a player clicks on one (at which point the player loses the game).
    c. The first square a player picks must never be a mine. The easiest way to handle this is to generate the location of the mines after the player clicks the first square.
4. You must generate the numbered squares and blank spaces correctly. That is, if I click on a numbered square and see a "2", there should be exactly 2 mines touching that square.

5. If a player reveals every space that isn't a mine, the game should end and the player should get a message saying they won
6. If a player reveals a mine, they game should end and the player should get a message saying they lost
7. You may use any aesthetic style you like (you are not required to match the windows aesthetic). However, the following must be true:
    a. The game must be 9x9 grid
    b. There should be a visible difference between an **unrevealed** square and a **revealed square**
8. PennDraw does **not** support right clicking, so you do not need to implement the flag mechanic. Similarly, the "middle clicking" tool many minesweeper players use will also not be possible to implement in PennDraw.
9. The game must be replayable after a game over (either win or loss) without rerunning the program. The board should be reset - you cannot use the same board as the last game.

Additional Challenge (not extra credit, just a fun way to challenge yourself):

In minesweeper, it is valuable to be able to "mark" spaces you know aren't safe. This is done typically using a flag. In most variations of the game, this is done by right clicking with the mouse. However, PennDraw does NOT support right clicking. Try to implement another way to use flagging in the game. This could include things like:

- Have a button on the screen that toggles to "flagging" mode. While in flagging mode, clicking on a square does NOT reveal it. It does mark it with a flag, however. This button would switch the user between "flagging" mode and "revealing" (that is, click reveals the tile) mode.
- Have it where if a user clicks while holding space bar, or control, or some other button, instead of revealing, it flags the tile. This approach may be a bit more intuitive to play with, but is harder to implement.

Implement a menu with multiple difficulties.

Implement visual effects to make it look cooler.

Submission

When you are ready to submit, put all the code and files for your project in a folder and zip it. The zip file should be named according to the name of your project - `minesweeper.zip, connect4.zip, sudoku.zip, 2048.zip`. Submit this file and the project readme through the dashboard.

Congratulations on finishing the last assignment of CIS 110. From Dr. McBurney and all the TAs, we hope you enjoyed the class. Best of luck on the final exam!

This project was written by Will McBurney (Sudoku Solver, Connect 4, Minesweeper, 2048), Rohan Bhide (2048). The game ideas do not belong to us.