CMPT 440 – Spring 2019: The Complexity of Minesweeper

Phil Kirwin
Due Date: 16/5/2019

Abstract

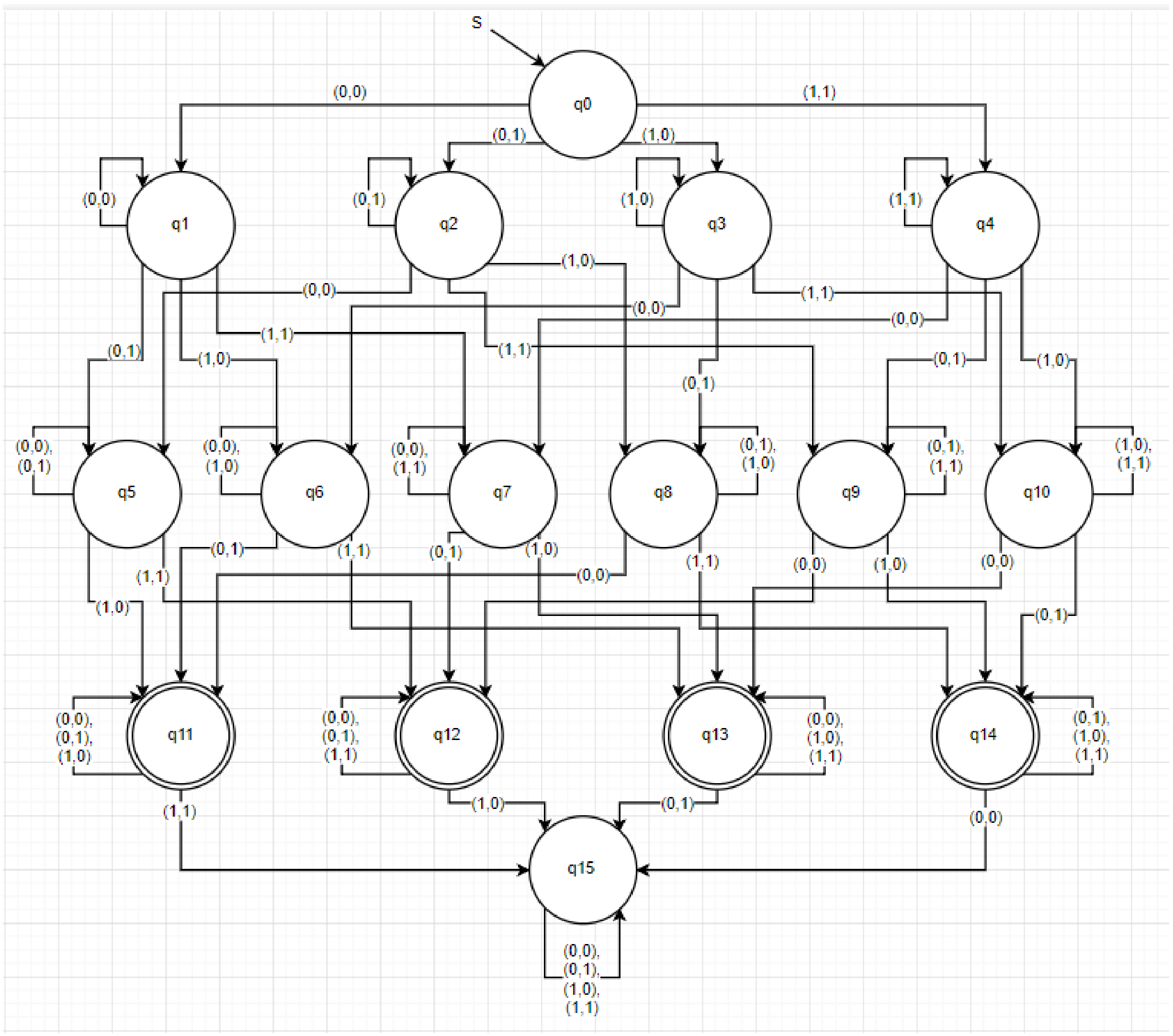
Often in software development, it is necessary to add layers of abstraction to an application so that the user does not need to understand complex underlaying processes to use the application properly. An excellent example of this principle is the game Minesweeper, wherein the player navigates through a series of board states without having to consider how the game processes their input and makes the necessary state transition. From the point of view of the user, they need only click on a tile and the application handles the rest silently until returning with an updated board state. This is easily modeled by creating a Deterministic Finite Automata of potential board states, thus demonstrating every possible sequence of moves on a board of a given size. Then, by using a Java method to place a mine on this board at random, it is possible to have a functioning game of minesweeper that uses this DFA to handle the game state and determine if the game has been won or lost. It is also quite simple to demonstrate how the number of possible board states grows exponentially with each additional tile that is added to the board.

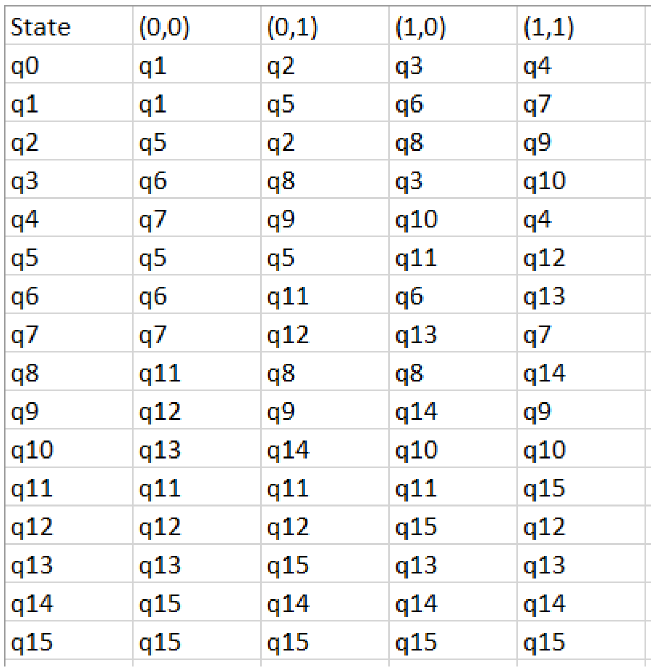
Introduction

For this project, I used a DFA to model the potential game states that can occur during a simplified version of the game Minesweeper. A DFA can model this game very effectively given that each of the players actions results in a new board state that makes it very obvious when the player has reached an accepting state, where the game is won, or an error state, where the player has clicked a mine and has lost. Given the great volume of possible board states that can occur in a standard game of Minesweeper, I modeled an especially small board in order to demonstrate how much more complex games can get simply by a small increase in board size. This model was created from the perspective of the player, and thus assumes no knowledge of the location of any mines. This way, the DFAs is a more honest representation of how a player must make decisions and strategize in a typical game of Minesweeper. Each state then advances based on the remaining possible moves and their possible results, either ending in an error state if the selected space contains a mine, an accepting state if the number of remaining spaces is equal to the number or mines on the board, or a continuing state if neither condition is met. It should thus be expected that increasing the number of spaces on the game board or the number of mines on the board will have a drastic effect on the number of potential board states. I hoped that with this project, I could demonstrate how well-designed Minesweeper is both as a game and as a general application. It is a deceptively simple game from the perspective of the user, but the underlaying ability to handle every potential decision of the user end up getting very complex relative to the size of the game board. As such, I hope that this project can be used to demonstrate to software developers that their work should reflect the deceptive simplicity of Minesweeper. In other words, Minesweeper is an application that is very simple for users to interact with without having to understand its more complex mechanisms. Additionally, I hope that this project can grant users an appreciation of the inner workings of a seemingly simple software application, which would in turn grant them an appreciation for the much more massive applications that they interact with on a daily basis.

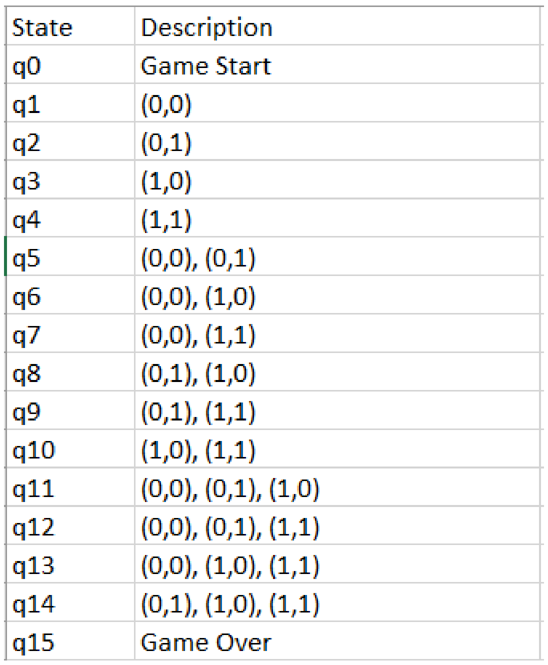
System Description

In order to make a DFA to model the potential states in a game of Minesweeper on a 2 by 2 board, the alphabet should be defined as the set of potential moves. That is, each transition on the DFA will be made based on an ordered pair of coordinates that each represent the position of a single tile on the board that can be clicked. Next, a generic DFA can be made that does not assume the location of the mine on the board. This means that in this DFA, all moves are legal and do not result in a game over, thereby creating four possible accepting states wherein the player has clicked on three unique tiles ($q11$, $q12$, $q13$, and $q14$). The DFA therefore represents the transitions between having one tile selected, then two, then three where the game is won. It can also be assumed that selecting the fourth tile from one of these accepting states will always result in a game over ($q15$), as the board will always contain a single mine. Additionally, in every state other than $q0$, selecting a set of coordinates that has already been selected will simply loop back to the same state, as it is technically not an illegal move, though it does not advance the game in any way.


S
q0
(0,0)
(1,1)
(0,1)
(1,0)
q1
q2
q3
q4
(0,0)
(0,1)
(1,0)
(1,1)
(1,0)
(0,0)
(1,1)
(0,0)
(0,0)
(1,1)
(0,1)
(1,0)
(1,1)
(0,1)
(0,1)
(1,0)
q5
q6
q7
q8
q9
q10
(0,0), (0,1)
(0,0), (1,0)
(0,0), (1,1)
(0,1), (1,0)
(0,1), (1,1)
(1,0), (1,1)
(0,1)
(1,1)
(0,1)
(1,0)
(0,0)
(1,1)
(0,0)
(1,0)
(0,0)
(1,1)
(1,0)
(0,1)
q11
q12
q13
q14
(0,0), (0,1), (1,0)
(0,0), (0,1), (1,1)
(0,0), (1,0), (1,1)
(0,1), (1,0), (1,1)
(1,1)
(1,0)
(0,1)
(0,0)
q15
(0,0), (0,1), (1,0), (1,1)

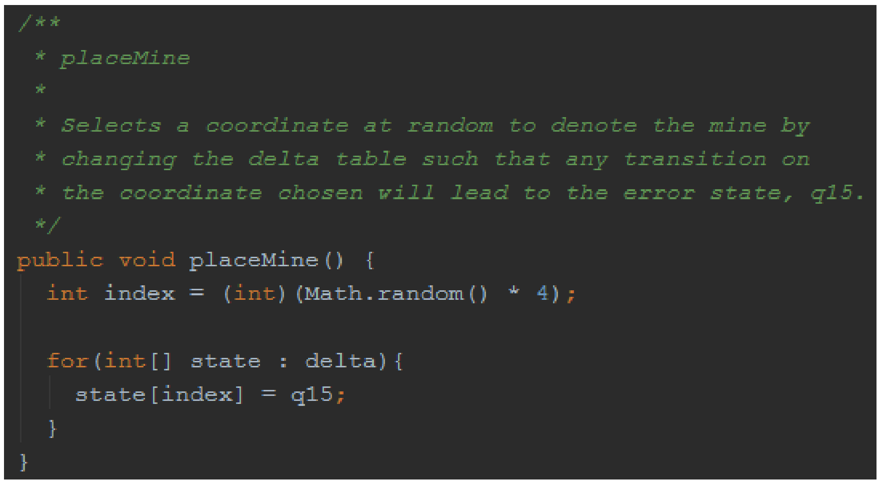


State	(0,0)	(0,1)	(1,0)	(1,1)
q0	q1	q2	q3	q4
q1	q1	q5	q6	q7
q2	q5	q2	q8	q9
q3	q6	q8	q3	q10
q4	q7	q9	q10	q4
q5	q5	q5	q11	q12
q6	q6	q11	q6	q13
q7	q7	q12	q13	q7
q8	q11	q8	q8	q14
q9	q12	q9	q14	q9
q10	q13	q14	q10	q10
q11	q11	q11	q11	q15
q12	q12	q12	q15	q12
q13	q13	q15	q13	q13
q14	q15	q14	q14	q14
q15	q15	q15	q15	q15



State	Description
q0	Game Start
q1	(0,0)
q2	(0,1)
q3	(1,0)
q4	(1,1)
q5	(0,0), (0,1)
q6	(0,0), (1,0)
q7	(0,0), (1,1)
q8	(0,1), (1,0)
q9	(0,1), (1,1)
q10	(1,0), (1,1)
q11	(0,0), (0,1), (1,0)
q12	(0,0), (0,1), (1,1)
q13	(0,0), (1,0), (1,1)
q14	(0,1), (1,0), (1,1)
q15	Game Over

The next important element of this system is the placeMine() method, seen here:



```
/**
 * placeMine
 *
 * Selects a coordinate at random to denote the mine by
 * changing the delta table such that any transition on
 * the coordinate chosen will lead to the error state, q15.
 */
public void placeMine() {
    int index = (int)(Math.random() * 4);

    for(int[] state : delta){
        state[index] = q15;
    }
}
```

This method is very simple, but it is what makes the application function like an actual game of Minesweeper. First, it selects an index at random that is a least 0 and is less than the number of tiles in the game. Then it takes advantage of the fact that the delta function is table-driven and stored in a two-dimensional array by matching this chosen index to a column in that table and changing each transition in that column to $q15$, the "Game Over" state. By doing this, this method essentially cuts down the "neutral" DFA that is provided such that only one accepting state remains: the one in which only tiles that don't contain mines are selected.

Though my initial plan was to create a DFA for a 2 by 2 Minesweeper game, observe how many possible states there were, and then scale up the board size to a 3 by 3 board, I realized that doing so would create a far more complex DFA than I would be able to reasonably and legibly create. This is due to the fact that the number of possible states for a given game board grows exponentially as the number of tiles on the board grows.

Proposition 1. *The number of possible board states is equal to 2^n where n is the number of tiles on the board and $n \in \mathbb{N}$ where $\mathbb{N}$ is the set of all natural numbers.*

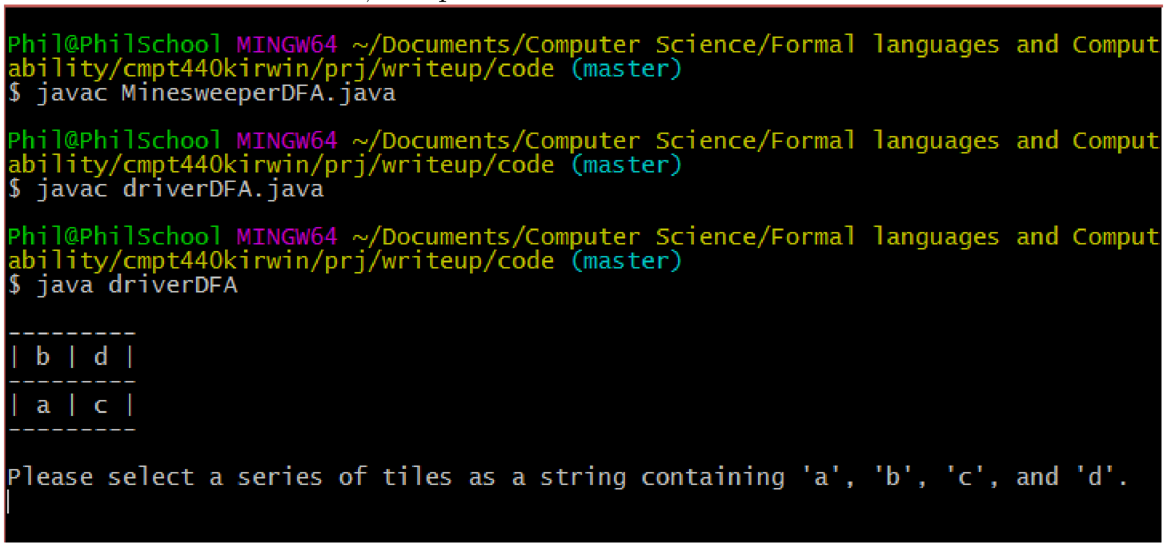
Proof. Base Case: Let $n = 1$. Thus we have that $2^n = 2^1 = 2$. We also know that a board with 1 tile has 2 possible states: one where the tile is selected and one where it is not selected. Thus the proposition holds for $n = 1$.

Inductive Case: Assume that the proposition holds for $n = k$ where $k \in \mathbb{N}$. This means that a board of k tiles has 2^k possible board states. Now consider $n = k + 1$. A board of $k + 1$ tiles has the exact same set of states as a board of k tiles, except that the new tile can either be selected or not selected. Thus we have that a board of $k + 1$ tiles has twice as many possible states as a board of k tiles. Given that a board of k tiles has 2^k possible board states, we have that a board of $k + 1$ tiles has $2^k * 2$ possible board states. This can be rewritten as $2^k * 2^1$, or as 2^{k+1}. Thus we have that the proposition holds for $n = k+1$. $\square$

Following this proof, one may calculate that a 3 by 3 board, having 9 tiles, would have $2^9 = 512$ possible board states, while a standard 16 by 16 Minesweeper board has $2^{256} = 1.158 * 10^{77}$ possible board states.

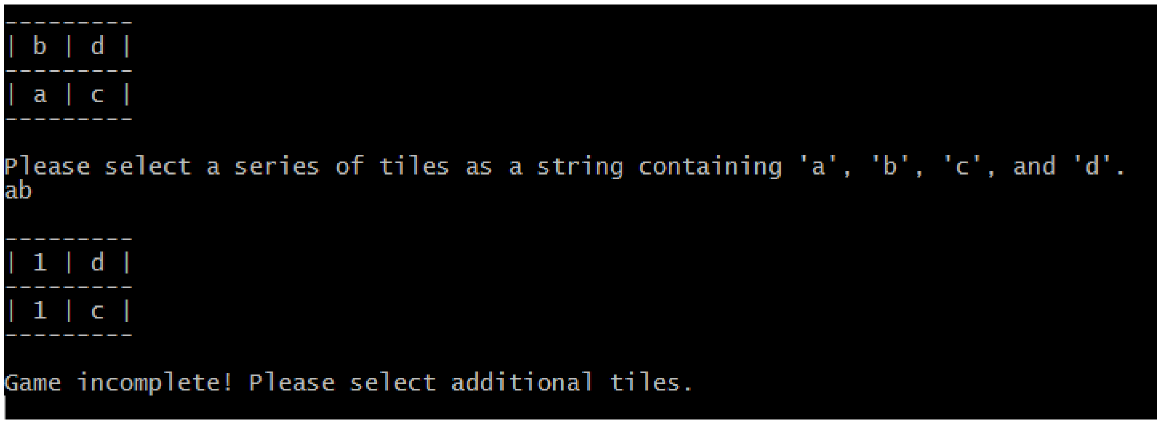
User Manual

The system is built in such a way that a user can play a simplified game of Minesweeper from the command line. First, compile and run the Java files like so:



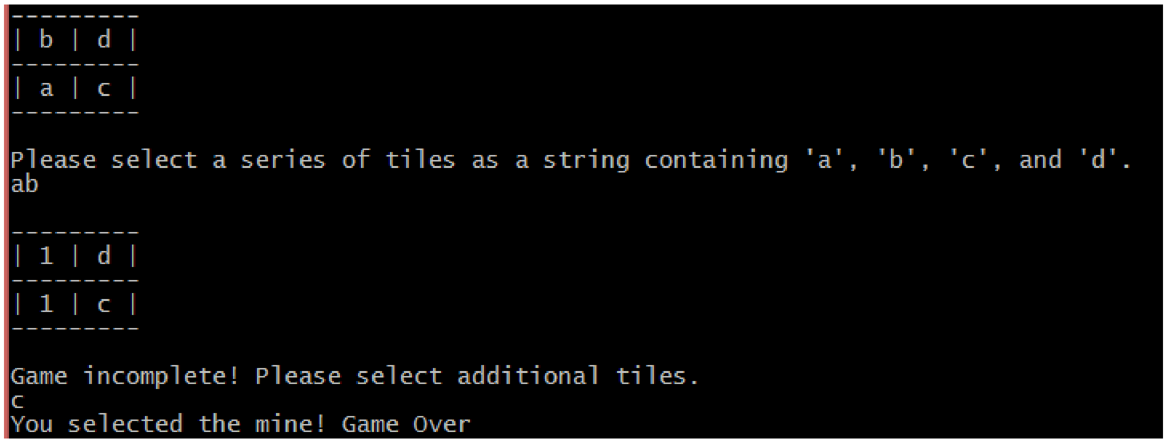
```
Phil@PhilSchool MINGW64 ~/Documents/Computer Science/Formal languages and Comput
ability/cmpt440kirwin/prj/writeup/code (master)
$ javac MinesweeperDFA.java

Phil@PhilSchool MINGW64 ~/Documents/Computer Science/Formal languages and Comput
ability/cmpt440kirwin/prj/writeup/code (master)
$ javac driverDFA.java

Phil@PhilSchool MINGW64 ~/Documents/Computer Science/Formal languages and Comput
ability/cmpt440kirwin/prj/writeup/code (master)
$ java driverDFA

---------
| b | d |
---------
| a | c |
---------

Please select a series of tiles as a string containing 'a', 'b', 'c', and 'd'.
```

For the purpose of hiding complexity, I chose to add a very simple UI to allow the user to select tiles based on arbitrary labels given to each, rather than having to input specific coordinates. Additionally, selecting tiles that are not the mine will print a new game board showing which tiles have been selected.



```
---------
| b | d |
---------
| a | c |
---------

Please select a series of tiles as a string containing 'a', 'b', 'c', and 'd'.
ab

---------
| 1 | d |
---------
| 1 | c |
---------

Game incomplete! Please select additional tiles.
```

Lastly, the game will end when either the mine has been selected, or the three tiles that don't contain the mine are selected. In either case, a message will indicate what has happened as shown here:



```
---------
| b | d |
---------
| a | c |
---------

Please select a series of tiles as a string containing 'a', 'b', 'c', and 'd'.
ab

---------
| 1 | d |
---------
| 1 | c |
---------

Game incomplete! Please select additional tiles.
c
You selected the mine! Game Over
```

It should also be noted that entering any character besides *a*, *b*, *c*, or *d* is considered bad input and will consider this a game over. Additionally, entering a tile that has already been selected will not advance the game at all, but will not result in a game over. Instead, the previous board state will be printed again.

Conclusion

The goal of this project was to demonstrate how even an application as simple to use as the game Minesweeper is capable of hiding a large amount of complexity. The project demonstrates that even in its simplest form, a 2 by 2 grid with only 1 mine, a DFA modeling potential board states has 16 states and can be modified in 4 different ways in the code to accommodate for the possible placements of the mine. Additionally this project includes a proof that the number of potential states in a given board grows exponentially as more tiles are added. This means that simply increasing the size of the board used in this project to a 3 by 3 grid would produce a DFA with 512 states. Lastly, this project demonstrates how well designed Minesweeper is by successfully recreating the game in the command line, while also hiding the complexity of transitioning to a new board state from the user. By doing so, this project provides an example for software developers of what to strive for as far as user friendliness goes. Additionally, it provides users some insight into just how much work can go into making software seamless and easy to use.