Date

CISC 205 – OOPS C++

Professor Larry Forman

Telephone: 619.388.3666

E-Mail: [LForman@sdccd.edu](mailto:lforman@sdccd.net)

Mail Box: Room A-8

Office: BT-210-G

**TRAINING ASSIGNMENT #1.3CA: CELLULAR AUTOMATA**

**=== DUE ===**

**=============**

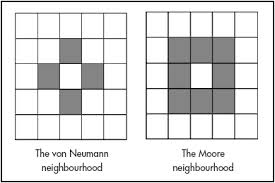
**TASKS:**

**0 –** First, read this Task Sheet!

Second, put a check mark by each Task number **and** letter when you complete it. Third, hand in this completed Task Sheet and Time Sheet when you demo this TA.

**1 –** **READ**: **HO#1.3, TDB**

**2 –** **TA OBJECTIVES**

* Develop a menu-based C++ program to produce interactive screen output
* Incorporate if-else, switch, loops and arrays
* Introduce error-checking
* Maintain effective Michelangelo documentation

**3 – BACKGROUND / SPECIFICATIONS**

Cellular automata are important “creatures” in computer science that have been used to study biological processes, chemical reactions, cryptography, chaos theory, fractal analysis, image processing, dynamical systems in physics, and even philosophy – to mention just a few applications! In fact, some researchers have daringly suggested that everything in the universe – from atomic particles on up – ultimately emerges from the computational structures of cellular automata. In a sense, cellular automata represent a life-form in the game of life.

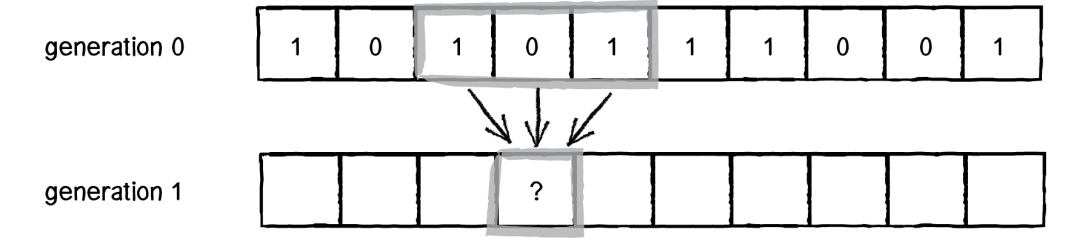
Informally, think of cellular automata as a game-playing process on an empty chess board. Imagine randomly placing individual pieces of chocolate candy on some of the squares on the board. Now, pick a particular chess square with chocolate on it. You get to either eat the chocolate OR else leave it alone. Which of these two outcomes takes place is going to be based on a “rule”. And, how this rule works will be determined by what’s already on all the immediately adjacent squares. For example, if I’m standing on a particular square and it is surrounded by more squares with candy than squares without candy, then the rule might say that I can eat the chocolate from that particular square. But, if the opposite condition occurs, then the rule might say that I must leave the chocolate in place. And, if there’s an equal number of squares with and without chocolate surrounding the particular square, then the rule might say I should flip a coin to see if I eat the morsel or not. Well, all this represents a part of one kind of “rule” for playing one kind of cellular automata game. So, that’s just some food-for-thought to start things off . . .

The most elementary kind of cellular automaton can be viewed as starting with a row of “cells” that contain “states” with values of either 0 or 1. Then, applying a specific computational rule, one can calculate the values for each successive row of states based on the states of the previous row of cells. Specifically, a cell’s new state is a function of all the states in the cell’s “neighborhood” from the previous row. Usually, the current cell’s “neighbors” are the three cells in the previous row that are touching the current cell. Thus, the three neighbors of a particular cell are:

1. The cell directly above the particular cell
2. The cell above and to the left of the particular cell
3. The cell above and to the right of the particular cell

We calculate a new cell’s state value by looking at the previous row’s neighbor states – and we do this for EVERY cell in a row. For example, consider:

Column 1 2 3 4 5 6 7 8 9 10



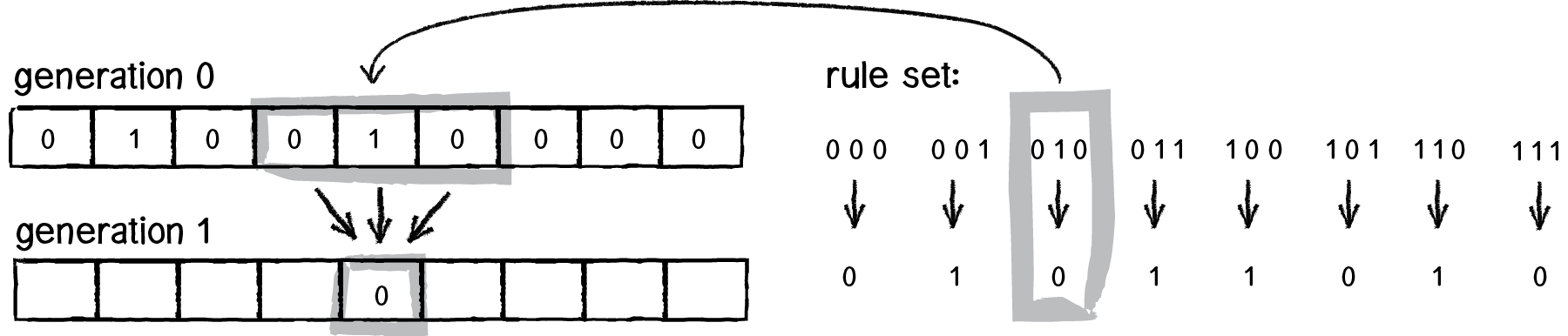
In the above diagram, the new value for the cell in column 4 of the second row (generation 1) will be determined by a mathematical rule – soon to be revealed – using the values in columns 3, 4 and 5 of the previous row (generation 0). And, this computational process continues for EVERY cell in the second row by means of its own three neighbors from the previous row. Moreover, this process continues to generate successive row after row after row . . .

The basic idea is to begin by “seeding” the first row, i.e., generation 0, by either randomly or deliberately putting in 0s and 1s throughout the first row of cells. Then, a rule – and there are many different ones from which to choose – is used to generate the second row. We continue this generation process for as many rows as we want.

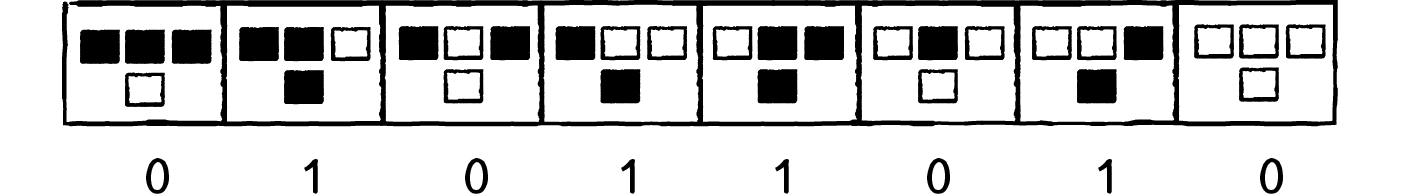
BTW: How do we deal with computing the values for cells at both the beginning and end of a row, namely in column 1 and column 10? Answer: typically, we assume there are “invisible” columns – number 0 and here number 11 – at each end which always are value 0.

The basic questions about this kind of cellular automaton include: What kinds of “interesting” patterns arise (over lots of rows) from different kinds of rules? How complicated or simple do the rules have to be to generate complex patterns? How do these patterns change – if at all – based on how the first row is “seeded”? How do small variations in “seeding” affect the patterns – do small changes in seeding merely produce small or no changes in patterns? Or is there a “butterfly affect”?

To help clarify matters, let’s consider a specific “rule”. BTW, how many possible rules are available based on having three neighbors influencing the state of a cell in the next row? Guess? Here’s one of those many(?) rules – which I’ll explain in class:



Or alternatively, the above rule appears schematically as the following, if the order is reversed with white corresponding to zero and black as one:



Make sense? Well, here’s the rule we *actually* will use in the TA: The “Odd Tie” rule

Imagine our classroom consists of a series of rectangular rows of seats from the front of the room to the back. The following “odd” situation in class today will determine whether or not you will wear an odd-looking tie at each subsequent class. So, here is the “odd tie” rule:

1. Look at the people in the three adjacent seats behind you, namely: directly behind you and the one to the left behind you and one to the right behind you. Suppose today in those three adjacent seats there are an odd number of people wearing odd-looking ties? Then at our next class you will wear an odd-looking tie. Otherwise, you won’t wear a tie.
2. Once you start wearing an odd-looking tie to class, you will continue to wear an odd-looking tie to class until further notice.
3. Today, I am going to invite everyone in class NOT to wear a tie to our next class EXCEPT for the person sitting in the middle of the last row. This colleague will wear an odd-looking tie at our next class

Based on the structure and schematic of the diagrams above (to illustrate a completely different “rule”), what do you guess is the “rule set” for our special “Odd Tie” rule? Well, draw it! Yup, go ahead and do it . . . Now . . . Right now!

Based on our rather “odd” but pretty simple rule, what kind(s) of pattern(s) will emerge over time? To “tie” to answer this, create a project, LarrysCA (no spaces or punctuation), that includes each of the following as separate functions with self-descriptive names and appropriate modifications – in your main(), where you deploy ONLY custom-defined functions along with any necessary local variables and constants. **BTW: NO GLOBAL VARIABLES ALLOWED**:

1. The "Welcome" function with your name via a **const** to greet the user.
2. The “Menu” function with appropriate error-handling of “dirty data” from user. Assume replies are NOT case-sensitive so lower- and uppercase values are valid
3. A separate function for EACH menu item: Sign-in, Logo, Odd Tie, Exit (etc. for some STARS)
4. “Nest” a "Hit ENTER to Continue" message function in any of the above items to enhance readability. But do NOT deploy your hitEnter function in the main.

**STAR: Allow only one sign-in. Alert user it can’t be done if user tries again**

**STAR: Only proceed after sign-in is done, although it’s okay to exit**

**STAR: Only proceed if sign-in is done first, although it’s okay to exit**

**STAR: Allow only 3 invalid menu entries after which program “exits”**

**STAR: Expand previous STAR to warn user at 3rd invalid entry that program will exit if the 4th try is invalid.**

**STAR: Instead of re-writing the menu for “Sorry” message, be “ecological “and move the cursor back to the prompt point, but be sure to pause for a moment and then erase the now unnecessary “Sorry” part of the message**

**STAR: Expand previous STAR to be ecological for all appropriate menu selections**

**STAR: Display the cellular output fully across 25 rows and 80 columns**

**STAR: When user selects C, play “Celebration”-type of music and stop it when menu reappears**

**STAR(1+): Research the history of cellular automata and present a PowerPoint presentation highlighting the origins and intriguing facts and tidbits about cellular automata and pioneers in the field.**

**STAR(1+): Extend previous STAR by adding “pizzazz” to the PowerPoint presentation with dynamic graphics, transitions, audio and possibly relevant video**

**4** – **SCREEN OUTPUT**

Display the following with appropriate blank lines between each section:

Welcome to Larry’s Cellular Automaton Program

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

C – Display the “Odd Tie” Pattern

X – Exit Program

Please enter your selection and hit RETURN: z

Sorry, but “z” is not a valid entry.

Time to try again . . .

**(NOTE: Do complete “error-handling” of “dirty” data)**

MENU

S – Sign in

L – Display Logo

C – Display the “Odd Tie” Pattern

X – Exit Program

Please enter your selection and hit RETURN: s

Thank you for selecting “s” . . .

Time to sign in!

Please sign in with your first name: Albert

Ah, Albert, so nice you have you join us for

a cell-ibration . . .

MENU

S – Sign in

L – Display Logo

C – Display the “Odd Tie” Pattern

X – Exit Program

Please enter your selection and hit RETURN: L Thank you for selecting “L” . . .

Time for Larry’s logo!

(Display your logo from TA #1.2Q)

<<< Hit ENTER to continue >>>

MENU

S – Sign in

L – Display Logo

C – Display the “Odd Tie” Pattern

X – Exit Program

Please enter your selection and hit RETURN: c

Thank you for selecting “c” . . .

Time to cell-ibrate some odd patterns!

(Using one-dimensional arrays, display the results of your cellular automaton for 79 columns (#0 - #78) and 22 rows with row zero (at the top of the screen) “seeded” only with ‘1’ in column #39 (where column #0 is the far left column) and all other columns “seeded” with ‘0’

<<< Hit ENTER to continue >>>

**STAR: Display all the rows and columns with a “space” for ‘0’s and an interesting extended ASCII graphics character, like \333, for each ‘1’**

**STAR: Expand previous STAR by using color for the special ASCII graphics character (representing the ‘1’s)**

**STAR: When C is selected, prompt user for number of rows to be displayed instead of automatically showing 22 rows**

**STAR: Add another menu item to display an “Even Tie” pattern, where the current cell becomes 1 only if in the three neighbors, there is an EVEN number of 1s, otherwise the cell becomes 0. How does the output compare for the “Odd Tie”?**

**STAR: Add another menu item that can randomly seed all the cells in the first row and then displays the resulting 22 row pattern**

**STAR: Add another menu item that allows user to enter any number of “seeds” ( = 1s) for specific locations in the first row**

**STARS(2): Experiment with different “rules” and select one that you find especially appealing to be included as another menu item. Use any “seeding” you wish, but experiment with several and describe – in a brief screen output message – any oddities you find.**

**STARS(2+): Create a simple way to let the user choose any of the 2^8 = 256 possible rules to display the cellular output.**

**STARS(2+): Experiment with different “non-binary” rules that use seed values of 0, 1 and 2. Select one that you find especially appealing to be included as another menu item. Use any “seeding” you wish, but experiment with several and describe – in a brief screen output message – any oddities you find.**

**STARS(2): Expand previous STAR to display the output with 3 different and visually clear and evocative colors for the 0, 1 and 2.**

**STARS(2): Add a menu item to display not a one-dimensional cellular automaton, but a two-dimensional one**

**STARS(2): Add another menu item to display the results of C in reverse order, i.e., display the bottom row at the top, etc. until you display the top row at the bottom, by means of storing the info using two-dimensional arrays**

MENU

S – Sign in

L – Display Logo

C – Display the “Odd Tie” Pattern

X – Exit Program

Please enter your selection and hit RETURN: x

Thank you for selecting “x” . . .

Time to x-cape!

Farewell, Albert, and thanks for visiting Larry’s Cellular Automaton Program

Display the current date and time

Display your complete ID INFORMATION (nicely formatted),

CREDITS and any STARS you did in the following format (BTW – Credit given only for STARS that are completed and fully displayed here):

STARS

1. Fancy logo
2. “Even Tie” pattern
3. & 4. PowerPoint presentation about cellular automata

TOTAL STARS = 4

<<< Hit ENTER to continue >>>

**5** – **SAVE** your file early and often -- like every 5 minutes. And, use your backup "disk"

**6** – **TEST** your file early and often -- like every 5 minutes -- How do you eat an apple?

**7** – **PROGRAM DOCUMENTATION**

1. First, include via comments your complete ID INFORMATION, PROGRAM DESCRIPTION, **CUSTOM-DEFINED FUNCTION LIST** and CREDITS (to those who helped you and whom you helped).
2. **Include "inline credits" to acknowledge specifically where you were helped.**
3. Add comments immediately before each segment of your program to describe "highlights" of coming attractions. Insert at least one blank line before each of these comments. Make all function, variable and const names self-descriptive, clear and fully formed (no abbreviations or secret code names). **Use verbs for function names, like “displayXyz” or “getXyz” or “calculateXyz” or “doXyz” and nouns for constants and variables.**
4. Define a const for your name and use it whenever your name appears
5. Add “banners” for: your prototypes to include description of each function, all constants, variables, start of function definitions, end of function definitions and in the function definitions with a description of each function (you can use the same ones as in your prototypes). See Handout #1.2 for all the details.
6. **FUNCTION PROTOTYPES – In your function descriptions, be sure to include how the parameters are used and for non-void functions identify what information will be returned. Make sure to say “return x” for non-void functions. For example, consider this hypothetical function prototype:**

**//NAME: getFavoriteNumber**

**//DESCRIPTION: Prompts user by name for a favorite number and**

**returns it**

**int getFavoriteNumber( string userName )**

1. IMPORTANT: Remember to embrace the basic Michelangelo structure for **every** C++ program.

**8** – **DEMO (= beta testing)** your program in the Lab with a completed TASK SHEET

**9** – **HAND IN HARDCOPY** of this TASK SHEET and your accurate TIME SHEET

**10** – **STARS (One STAR for each item, but no “double-dipping”)**

1. Work in a 2-3 person team to write and demo one program together that contains all the required information. Additionally, include the names of all the team members (as separate constants) in the welcome and farewell AND include each team member’s individual logo when displaying the logos. Add additional and different cellular automata patterns, one for each teammate. In the ID INFO, individually show hours and difficulty for each person. **Also, in each function, document the names of the contributors and the percentage each one contributed to the function (1 STAR per teammate).**
2. Create a new, enhanced “pizzazzy” logo
3. In an engaging way, use >=4 windows.h colors throughout
4. Implement this assignment by creating and using a custom header file to hold all your function prototypes and definitions. Include your Michelangelo documentation with ID INFO, etc. in the header file
5. Add extra pizzazz and briefly explain what you did:
6. Effectively use three new "Advanced" features (= not yet introduced). List them:
7. Demo before the due-date (N.B.: You still can do more STARS on due-date)

**“A river cuts through rock, not because of its power, but because of its persistence.”**

**Jim Watkins**

**“Courage doesn’t always roar, sometimes it’s the quiet voice at the end of the day whispering ‘*I will try again tomorrow*.’”**

**Mary Anne Radmacher**

**“If you can’t fly then run, if you can’t run then walk, if you can’t walk then crawl, but whatever you do you have to keep moving forward.”**

**Martin Luther King, Jr.**

