Problem Set 3

Due before midnight on Monday, October 8, 2018.

1 Assignment Goals

1. Produce an application with more than one class, appropriately split into multiple files.
2. Learn how to use a constructor to produce a semantically valid non-trivial data structure.
3. Learn how to use classes and objects to model a physical structure.
4. Learn how to driver program to exercise and test a class.
5. Learn how to code within a prescribed and restricted subset of the language.

2 Think-A-Dot

2.1 Some history

Think-a-Dot is a mathematical toy introduced by E.S.R. Inc. in the 1960's.

http://www.jaapsch.net/puzzles/images/thinkadot.jpg

It is covered by U.S. patent 3,388,483, issued June 18, 1968 to Joseph A. Weisbecker. (See Fig. 1.)

1

3,388,483
COMPUTER-TYPE GAME AND TEACHING AID
Joseph A. Weisbecker, 1220 Wayne Ave.,
Cherry Hill, N.J. 08034
Continuation-in-part of application Ser. No. 462,349, June 8, 1965. This application July 21, 1966, Ser. No. 566,881
3 Claims. (Cl. 35—30)

ABSTRACT OF THE DISCLOSURE

This invention relates essentially to an instruction and play device of the computer type wherein an enclosure is provided to receive checks, which serve to produce computer effects upon gravitational movement through the enclosure.

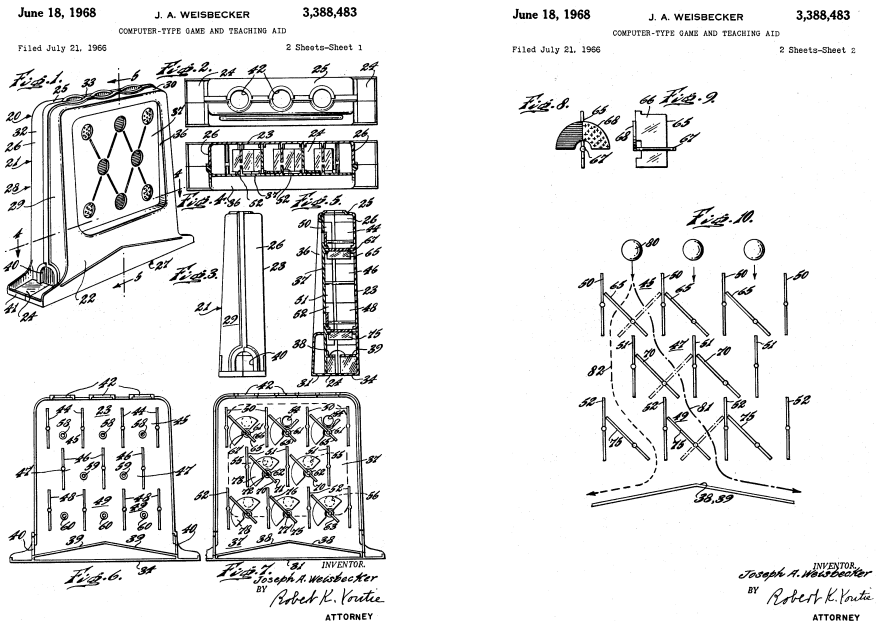


Figure 1: U.S. patent 3,388,483, issued June 18, 1968 to Joseph A. Weisbecker.

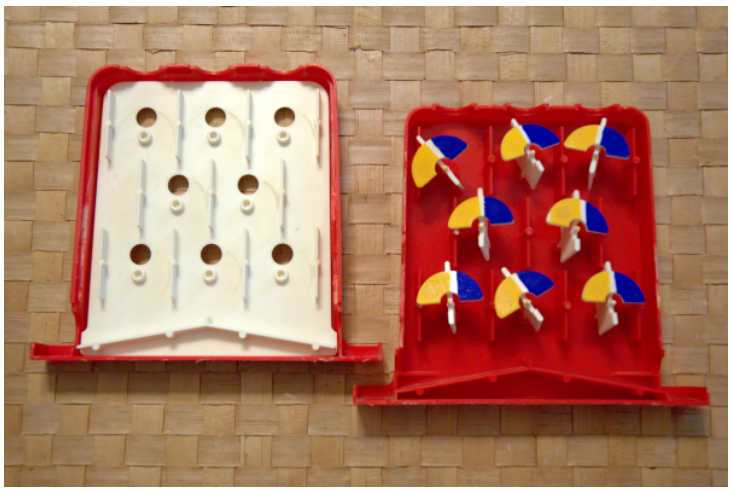


Figure 2: Looking inside a slightly-modified box.

Ask some questions:

1. What is the structure of the machine? (See Figure 3.)
2. Starting from the all-yellow pattern, can one drop in marbles so as to make it all blue?
3. If so, can one get back to the all-yellow pattern?
4. How many of the $2^8 = 256$ possible patterns can one reach from the initial state (all-yellow)?
5. Given that pattern s_2 is reachable from pattern s_1, how many marbles are needed? (Call this the *directed distance* from s_1 to s_2.)
6. Is the distance from s_1 to s_2 always the same as the distance from s_2 to s_1?
7. What is the largest distance between any pair of states for which the distance is defined?

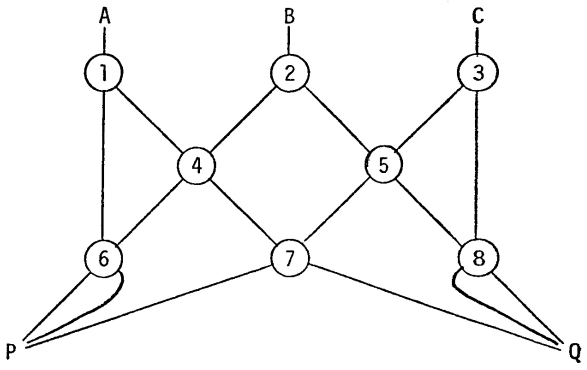
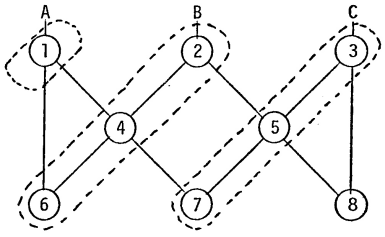


Figure 3: Structure of the machine.

Binary Counter

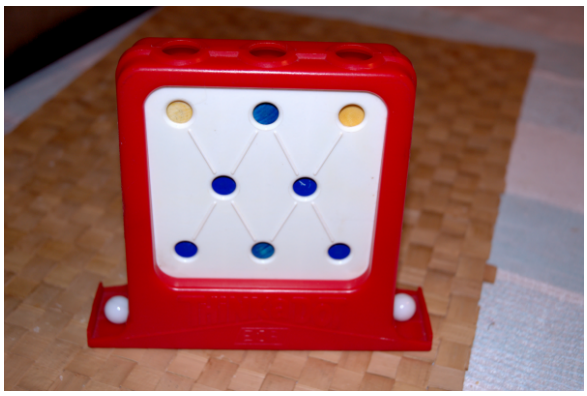
Eight balls through hole C will cause gates 7–5–3 to behave like a binary counter and cycle through all eight possibilities.

Gates to the above and left (1, 2, 4, 6) are not affected.

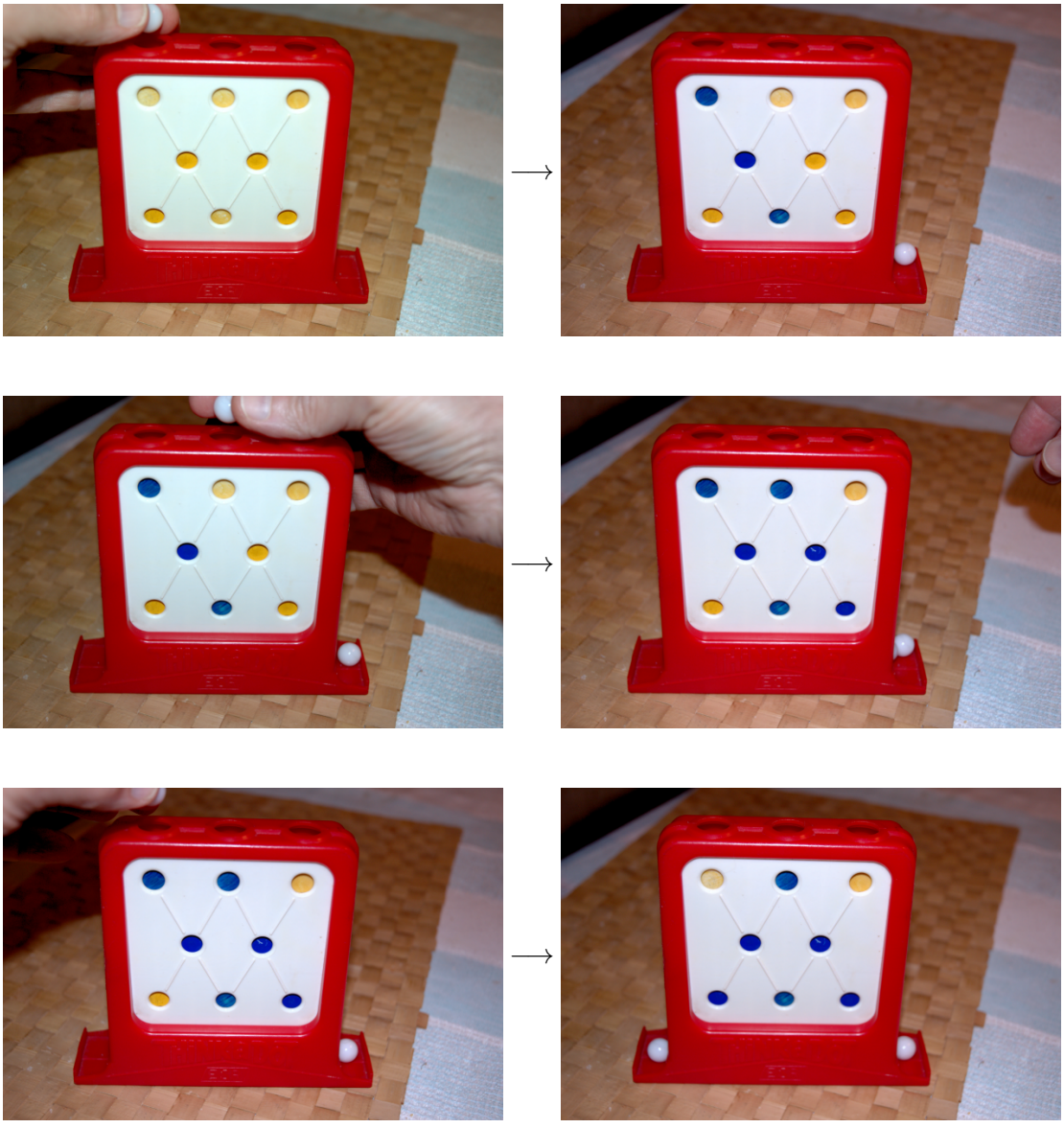


How can you get to a particular pattern?

Starting from all yellow, how can one reach this goal?



Here's one solution:



2.2 Further references

Google returns many hits for the search term "think-a-dot".

1. Some of the early pre-E.S.R. Think-a-Dot history.
2. A realistic Think-a-Dot simulator that you can play with, written Scratch. This shows the original unmodified dot pattern that appears when the device is tipped to the right.
3. A Think-a-Dot-inspired electronic game from 2002.
4. Some of the mathematical theory behind Think-a-Dot (from Wikipedia, Think-a-Dot).
   (a) Schwartz, Benjamin L. (1967), "Mathematical theory of Think-a-Dot", Mathematics Magazine, 40 (4): 187193, doi:10.2307/2688674, MR 1571696.
   (b) Beidler, John A. (1973), "Think-a-Dot revisited", Mathematics Magazine, 46: 128136, doi:10.2307/2687967, MR 0379077.
   (c) Gemignani, Michael (1979), "Think-a-Dot: a useful generalization", Mathematics Magazine, 52 (2): 110112, doi:10.2307/2689850, MR 1572295.

3 Problem

You are to model a Think-a-Dot device and its behavior through a collection of C++ classes. You are also to write a command `tad` that allows a user to interact with your simulated Think-a-Dot device. User inputs are single letters commands. All command letters are case insensitive, so 'Q' and 'q' for example have the same effect. The commands are:

- 'A', 'B', 'C' simulate the action of the machine when a ball is dropped in hole 'A', 'B', or 'C', respectively.
- 'L', 'R' cause the gates to be reset to all point the same way – all to the left or all to the right, respectively.
- The flip-flops should be colored as shown for the modified box in Figure 2.
- 'P' prints the state of the machine using three lines of text, e.g.,

```
R L R
 L L
L L L
```

- 'H' prints a brief version of these instructions.
- 'Q' exits the program.

Your program will prompt the user to enter a command letter, check it for validity, and print the hole at which the ball exits the machine (hole 'P' or 'Q' as shown in Fig. 3).

4 Programming Notes

You will define and implement three classes: `ThinkADot`, `FlipFlop`, and `Game`. Class `ThinkADot` models the Think-A-Dot device. `FlipFlop` models a single flip-flop within the Think-a-Dot. `Game` controls the user-interaction with the Think-A-Dot. It prompts the user to get command letters (from `cin`) and to print results (to `cout`). It interacts with the Think-A-Dot to determine how the device responds to the various operations that can be performed on it.

Class `Game` should have a public function `play()` that starts the game. `play()` first creates a `ThinkADot` object where the flip-flops are colored as shown in Figure 2. It then enters the interactive loop that prompts the user for a command letter and performs the corresponding action.

Class `FlipFlop` models a single flip-flop. The state of a flip-flop is either "left" or "right" and should be represented by an `enum` type. (See 08-brackets `Token` class for an example.) There should be a `print()` function that just prints a single letter 'L' or 'R' according to the current state of the flip-flop. There should also be a function `flip()` that flips the state from "left" to "right" or vice versa and returns the side of the flip-flop ("left" or "right") the ball is on when leaving the flip-flop. Thus, if the flip-flop is in the "left"-leaning state initially, the ball will pass to the right, and the new state will be "right".

For this class, it is okay to have public functions `getState()` and `setState()` to be used by member functions of class `ThinkADot`. A superior design would nest the entire `FlipFlop` class inside of the `ThinkADot` class, but for this assignment, `FlipFlop` should be a separate class at the same level as the others. (We will get to nested classes later in the course.)

Class `ThinkADot` models the device. It has a private array (*not* a `vector`) of eight FlipFlop objects that store the current state of each of the eight flip-flops. Its constructor should initialize all of the flip-flops to the "left" position, the same as the 'L' command. It has public functions `reset()`, `play()`, and `print()` that carry out the actions 'A', 'B', 'C', 'L', and 'R' (with appropriate parameters) that can be performed on the device. The flip-flop states must be accessible *only* from these required member functions. In particular, there should be *no* getter or setter functions for the flip-flops.

The file `main.cpp` will have the same form as in PS1. The global function `run()` should only have two lines – one to instantiate `Game` and the other to call the Game object's `play()` function.

4.1 Computing the next state

The tricky part of this assignment is how to update the state when a ball is dropped through one of the three holes 'A', 'B', or 'C'. Referring to Figure 2, you can see seven channels through which the ball can pass. If we number them from 0 through 6, starting at the left, then we see that a ball dropped in hole 'A' enters channel 1. After passing through the first flip-flop it moves to either channel 0 or to channel 2, depending on the state of the flip-flop. If it goes to channel 0, it drops straight through to the bottom and comes out on the left side. If it goes to channel 2, it encounters the first flip-flop in the second row. After passing through it, the ball enters channel 1 or 3, etc.

Your code should trace the path of a ball through the machine as described above, flipping each of the flip-flops encountered on the way, and recording the last channel the ball was in. Clearly that will be channel 1, 3, 5, or 7. If it's 1 or 3, the ball exits the machine through hold 'P'. Otherwise, it exits through hole 'Q'. Note that it would be ambiguous where the ball exits if it were coming from channel 4, but that is not possible.

4.2 No-no's

There are many ways to implement a Think-a-Dot. For this assignment, you must do it as described above. Here are a few no-no's, not because they're necessarily wrong but because I want you to learn the particular techniques described above.

1. Don't use `new` or `delete`.
2. Don't use any Standard Library container functions such as `vector`.
3. Don't use a table lookup to find the next state.

4. Don't use nested classes.
5. Don't use language features that have not been presented in lecture or in any of the class examples. Don't use prohibited features such as non-const global variables or goto's.

If you think you need to violate any of these restricts, please ask me for help.

5 Grading Rubric

Your assignment will be graded according to the scale given in Figure 4 (see below).

#	Pts.	Item
1.	1	All relevant standards from PS1 are followed regarding submission, identification of authorship on all files, and so forth.
2.	1	A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++17`.
3.	1	Running `make` successfully compiles and links the project and results in an executable file `tad`.
4.	1	`tad` smoothly interacts with the user. Clean, easily understood user prompts and help messages are given.
5.	2	Bad user inputs are handled gracefully and do not result in fatal errors.
6.	6	All of the functionality in section 3 is correctly implemented. In particular, each of the eight command letters works properly in both upper and lower case and carries out its assigned action correctly.
7.	3	The structure of the program matches the specification and restrictions given in in section 4. No dynamic storage is used.
8.	1	Each function definition is preceded by a comment that describes clearly what it does.
9.	4	The program shows good style. All functions are clean and concise. Inline initializations, functions, and `const` are used where appropriate. Variable names are appropriate to the context. Programs are consistently indented according to the course indenting style. Each class has a separate `.hpp` file and, if needed, a separate `.cpp` file.
	20	Total points.

Figure 4: Grading rubric.