# Problem Set 2

Due before midnight on Monday, September 26, 2016.

## 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.

## 2 Think-A-Dot

### 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.

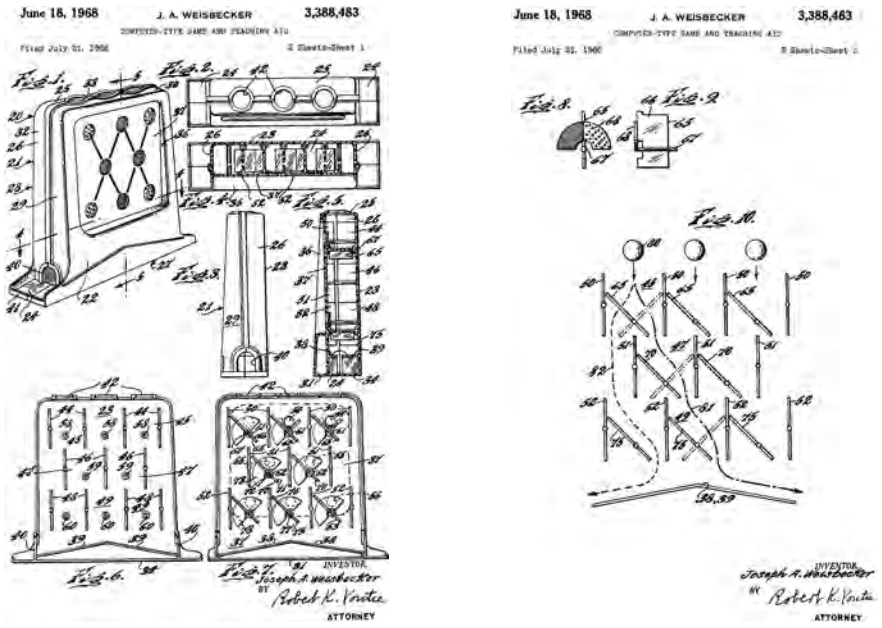
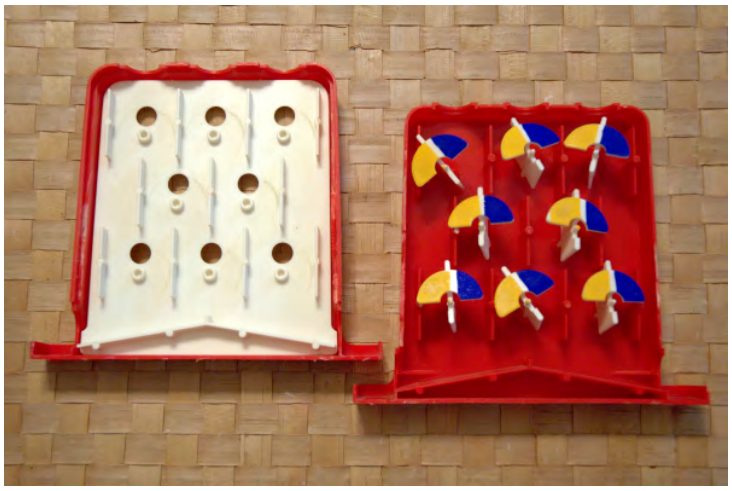



Figure 1: U.S. patent 3,388,483, issued June 18, 1968 to Joseph A. Weisbecker.

**Look inside…**



**. . . and ask some questions:**

1. What is the structure of the machine? (See Figure 2.)
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?

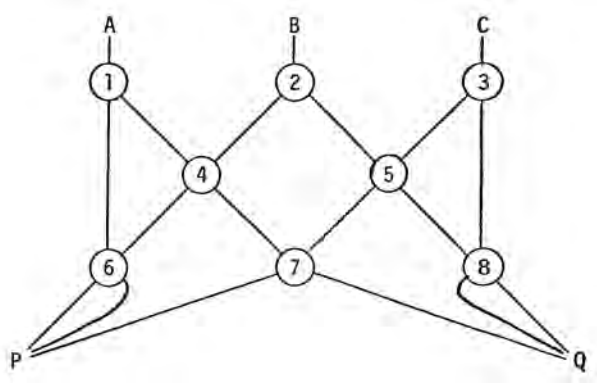
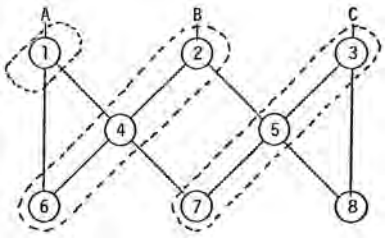



Figure 2: Structure of the machine.

**Binary Counter**

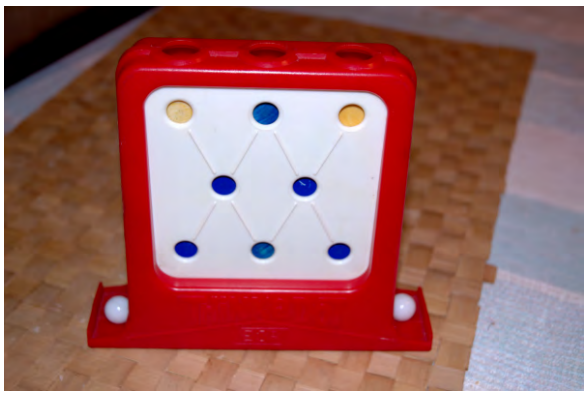
Eight balls through hole C will cause gates 7–5–3 to behave like a binary counter and cycle through all eight possibilities.

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

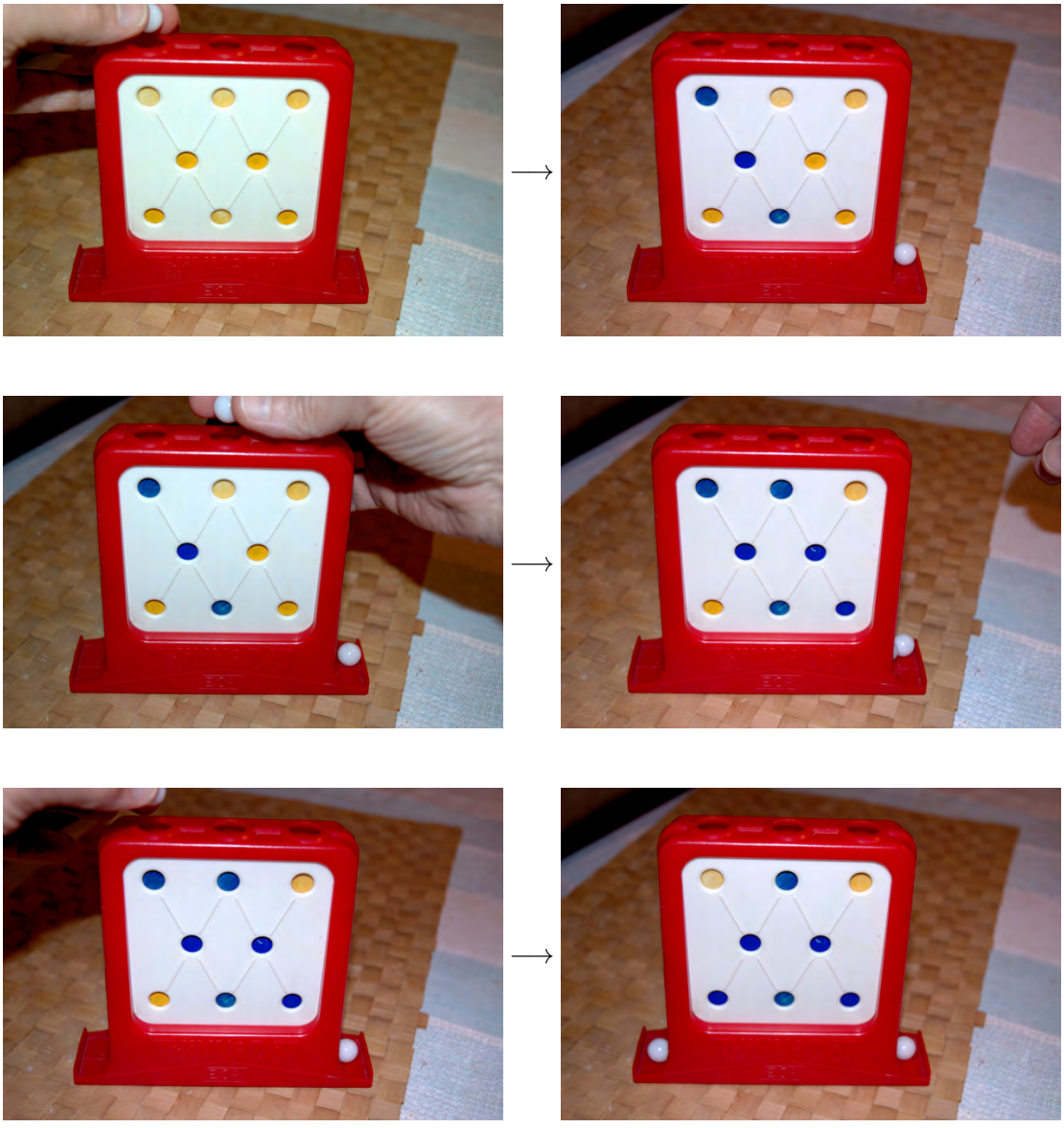


**How can you get to a particular pattern?**

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



Here's one solution:



# 3 Problem

You are to write a program to interact with the user to simulate a Think-A-Dot. 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.
- '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. 2).

# 4 Programming Notes

You should define (at least) two classes: `ThinkADot` and `Game`. Class `ThinkADot` models the Think-A-Dot device. Class `Game` controls the Think-A-Dot. It interacts with the user to get command letters and to print results. 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. It then enters the interactive loop that prompts the user for a command letter and performs the corresponding action.

Class `ThinkADot` models the device. It has private data members 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 publicly 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 with one exception: Instead of the try-block in `main()` consisting of a single call on a global function `run()`, the try-block will now have two lines – one to instantiate `Game` and the other to call Game's `play()` function.

# 5 Grading Rubric

Your assignment will be graded according to the scale given in Figure 3 (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++14`. |
| 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 in section 4. No dynamic storage is used (i.e., no need for `new` and `delete` in this assignment). |
| 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 an accepted indenting style. Each class has a separate `.hpp` file and, if needed, a separate `.cpp` file. |
| | 20 | Total points. |

Figure 3: Grading rubric.