Homework 5

Elliott Pryor

5 Nov 2020

Problem 1

A

1. What system did you download? I downloaded BoardGame Bay `https://sourceforge.net/projects/gamebay/files/gamebay_chess_0.7/` which is a Chess game.
2. What does it do? It is a chess simulator. So players can play chess using it.
3. How many Lines of Code does it have? How did you calculate this? This has 5187 lines of code. I wrote a python script that walks through the directories and opens all the .java files and counts the lines.

B

`design pattern finder output.txt`

1.
```
Found 16 files that possibly contain design patterns.

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\chess\play\condition
                \impl\my\ConditionFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\chess\play
                \impl\my\PlayFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\chess\play
                \moving\impl\my\MovingFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\chess\play
                \turn\impl\my\TurnFactoryMyImpl.java
Possible patterns: Factory
```

```
D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\config
                \player\impl\my\PlayerFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \action\impl\my\ActionFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \board\impl\my\BoardFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \condition\impl\my\ConditionFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \figure\impl\my\FigureFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \impl\my\PlayFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \moving\impl\my\MovingFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \moving\rule\impl\my\RuleFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \player\impl\my\PlayerFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \position\impl\my\OccupationStateFasteningBrokerMyImpl.java
Possible patterns: State

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \position\impl\my\PositionFactoryMyImpl.java
Possible patterns: Factory

D:\pryor\Downloads\gamebay_chess_src_0.7\net.sf.gamebay\src-impl\net\sf\gamebay\game\play
                \turn\impl\my\TurnFactoryMyImpl.java
Possible patterns: Factory
```

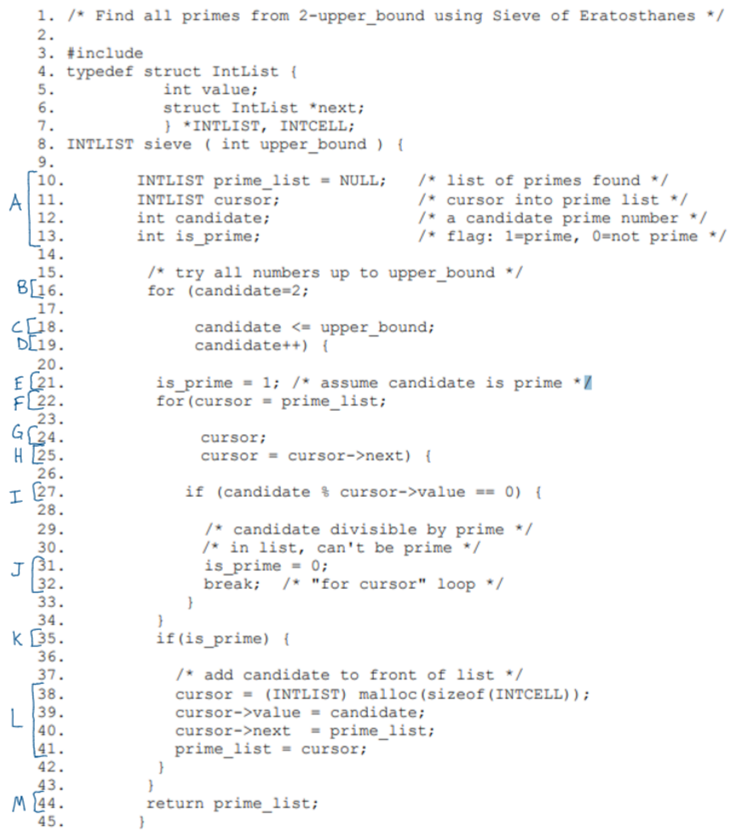
---

2. Since we did not use the ‘Search in File Content’ option it looks at the file name. So it is only guessing the design pattern by the filename. So it relies on standard conventions such

as naming factory classes with the name Factory. If we had checked this option it would presumably scan through each file and look for similar conventions.

3. I think that this method works. It is not very intelligent at actually detecting the design patterns. This algorithm relies entirely on standard naming conventions. So it would not detect it if you were to rename variables or use different conventions it would not work. But, this is the only practical way that I can think of to implement this. It is not feasible or practical to write an algorithm to analyze code structure intelligently in order to detect all of these cases. Perhaps it is possible with some complicated pattern matching and machine learning. But this is too computationally expensive. I would also implement something that was based on naming conventions as it can quickly give an overall picture of what is happening under the hood.

Problem 2



```
/* Find all primes from 2-upper_bound using Sieve of Eratosthanes */

#include
typedef struct IntList {
          int value;
          struct IntList *next;
          } *INTLIST, INTCELL;
INTLIST sieve ( int upper_bound ) {

       INTLIST prime_list = NULL;   /* list of primes found */
       INTLIST cursor;              /* cursor into prime list */
       int candidate;               /* a candidate prime number */
       int is_prime;                /* flag: 1=prime, 0=not prime */

        /* try all numbers up to upper_bound */
        for (candidate=2;

             candidate <= upper_bound;
             candidate++) {

         is_prime = 1; /* assume candidate is prime */
         for(cursor = prime_list;

              cursor;
              cursor = cursor->next) {

            if (candidate % cursor->value == 0) {

              /* candidate divisible by prime */
              /* in list, can't be prime */
              is_prime = 0;
              break;  /* "for cursor" loop */
            }
         }
         if(is_prime) {

           /* add candidate to front of list */
           cursor = (INTLIST) malloc(sizeof(INTCELL));
           cursor->value = candidate;
           cursor->next  = prime_list;
           prime_list = cursor;
         }
        }
        return prime_list;
       }
```

Figure 1: Labeled Nodes in Code

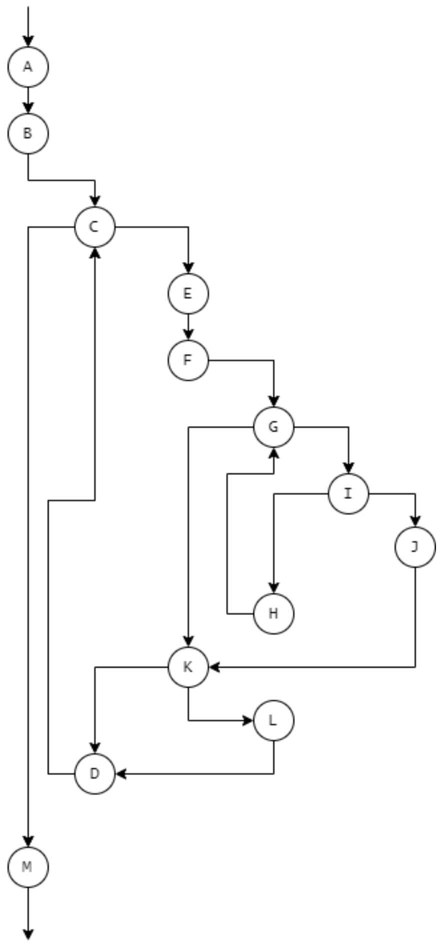


Figure 2: Control Flow Graph

a) upper_bound = 50, sieve(50). This tests all the nodes (and edges). The set 2...50 contains primes and non-primes so at least one candidate will trigger each node

b) upper_bound = 50, sieve(50). As mentioned in a) this tests all edges. Any test set that contains one prime and one non prime will cover all edges. Also in this case, node coverage is equivalent to edge coverage since it is not possible to cover all the nodes without using all the edges. The only edge that could be skipped is $G \rightarrow K$ which is not possible to have a set that does this since you can't have a list of numbers 2..x that is only non-primes so transition $J \rightarrow K$ must not be used for at least one candidate.

c) No this is not possible in general. There could be infinite inputs or non-reachable segments of code, so these non-reachable segments could never be covered by any test case.

Problem 3

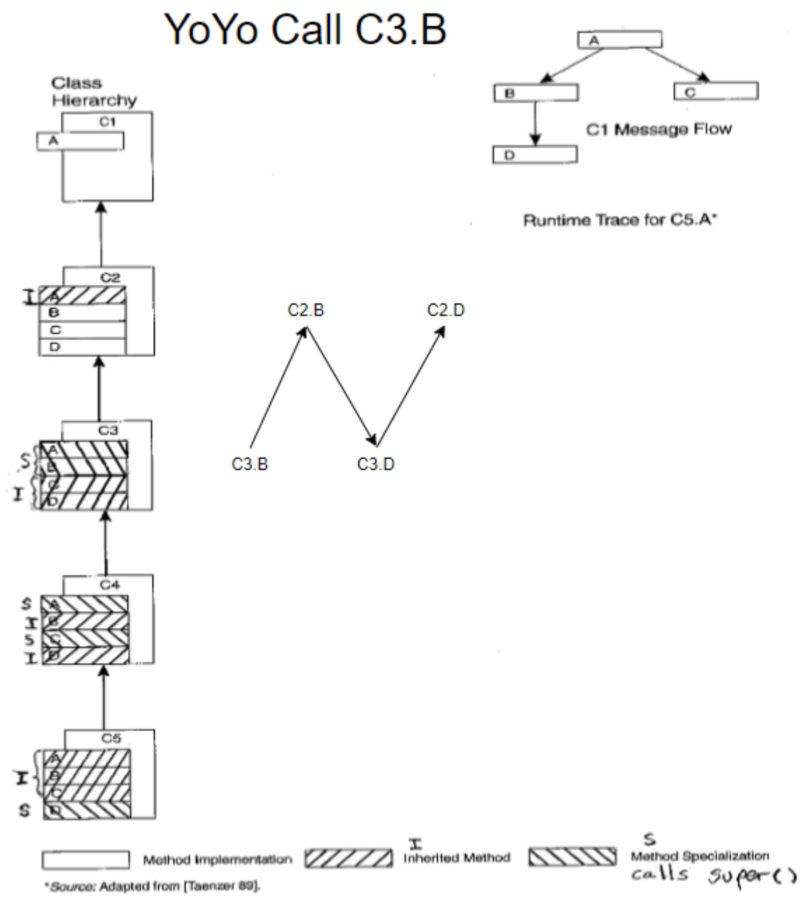


Figure 3: Yo Yo call from C3.B

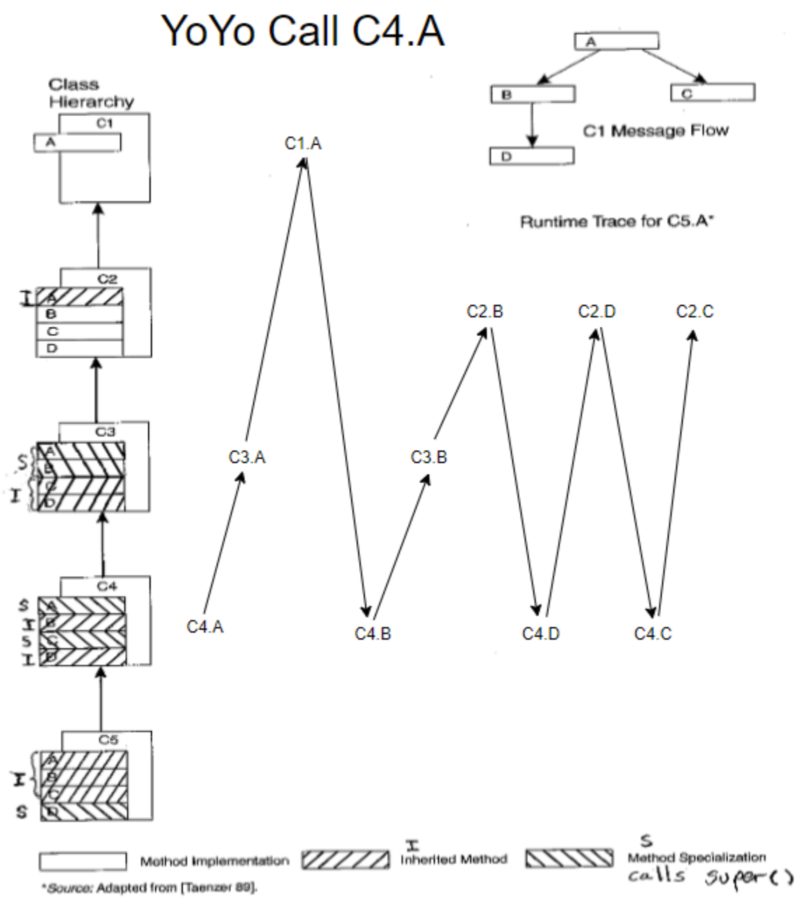


Figure 4: Yo Yo call from C4.A

a)

b) It errors as class C1 does not have method D

Problem 4

a) input = 1 $\Longrightarrow$ return 0 instead of 1

b) input = 0 $\Longrightarrow$ return 0 instead of 1

c) input = 5 $\Longrightarrow$ return 16 instead of 8