GUIssing game

Izaak van Dongen

June 10, 2018

Contents

List of Listings

1 Introduction

This project is the sequel to the popular `assignment_guessing`, at `https://github.com/elterminad0r/assignment_guessing`, now featuring a very useful and fluid graphical interface.

It uses the same techniques to search both $\mathbb{Q}$ and $\mathbb{Z}$, but doesn't implement the linear approach to either, as it's not really preferable in any circumstance.

2 Programs

2.1 Abstract

Despite the fact that the same algorithms from last time can be reapplied, their implementations have to be suited the the event-driven idiom. This is reflected in listing 1, which contains the 'boilerplate' code. It represents the broad protocol that a component implementing guessing should follow.

This is that a guesser may must implement a method to ask a question, and a separate method that receives the answer. The guesser should appropriately modify its internal state so that it knows what question is being answered, and what to ask next. This is really a kind of poor man's synchronous coroutine, as implemented for example by Python's generators, where the internal state is simply a stack frame. However, these do not come with fpc.

The other thing to note is that a guesser being able to deduce a number is considered exceptional behaviour, and hence is implemented by an exception, which should carry a message including what the correctly guessed number is, which may then be caught. If the guesser is not sure of the user's number, it should simply ask a normal question to verify if a guess is correct. This is implemented for example in listing 3.

It is implemented as an abstract class rather than as an interface because the main code also wants to be able to tear the engine down, with a `Free` call to prevent memory leakage. Unfortunately, destructors can't be included in interfaces, so there is no guarantee that an implementing class can be freed. Because of this, I instead use an object, which *is* understood to have a `Free` method.

```
1  {$MODE OBJFPC}
2
3  unit UGuesser;
4
5  interface
6
7  uses SysUtils;
8
9  type
10     TGuesser = class abstract
11         function ask_question: string; virtual; abstract;
12         procedure answer_question(reply: boolean); virtual; abstract;
13     end;
14     EGuessSuccessful = class(Exception);
15
16 implementation
17
18 end.
```

Listing 1: UGuesser.pas: Boilerplate and definitions for guessing objects

Listing 2 shows a class implementing this protocol, namely by performing a binary search. As previously discussed, this version of binary search works on the entirety of $\mathbb{Z}$, by determining bounds in a similar 'binary' manner (at least, it should be able to find sensible bounds in $O(\log_2(n))$ (and then guess the number in $O(\log_2(n))$).

```
1  {$MODE OBJFPC}
2
3  unit UBinarySearch;
4
5  interface
6
7  uses UGuesser, SysUtils;
8
9  const
```

```
10    // Symbolic constant representing the starting upper bound. 64 chosen to
11    // encompass most 2-digit numbers.
12    STARTING_BOUND = 64;
13
14  type
15    // Simple binary search, using the interval $Z & [lo, up)$.
16    // These bounds are determined by continually multiplying up by 2 until the
17    // user's number lies in the range.
18    TBinarySearcher = class(TGuesser)
19    protected
20      // various flags representing current stage of guessing
21      has_bounds, has_sign: boolean;
22      // bounds, and current pivot
23      lo, mid, up: integer;
24    public
25      constructor Create;
26      function ask_question: string; override;
27      procedure answer_question(reply: boolean); override;
28    end;
29
30  implementation
31
32  constructor TBinarySearcher.Create;
33  begin
34    has_bounds := False;
35    has_sign := False;
36    up := STARTING_BOUND;
37    lo := 0;
38  end;
39
40  function TBinarySearcher.ask_question: string;
41  begin
42    if not has_bounds then begin
43      Result := Format('Is %d \le \abs{n} < %d?', [lo, up]);
44    end else if not has_sign then
45      Result := 'Is n \ge 0?'
46    else begin
47      mid := (lo + up) div 2;
48      Result := Format('Is n \ge %d?', [mid]);
49    end;
50  end;
51
52  procedure TBinarySearcher.answer_question(reply: boolean);
53  var
54      tmp: integer;
55  begin
56    if not has_bounds then
57      if not reply then begin
58        lo := up;
59        up := up * 2;
60      end else
```

```
61        has_bounds := True
62    else if not has_sign then begin
63      if not reply then begin
64        tmp := lo;
65        // negate bounds, accounting for (non)?strictness
66        lo := -up + 1;
67        up := -tmp + 1;
68      end;
69      has_sign := True;
70    end else begin
71      if reply then
72        lo := mid
73      else
74        up := mid;
75      if up - lo <= 1 then
76        raise EGuessSuccessful.Create(Format('Your number was %d.', [lo]));
77    end;
78  end;
79
80  end.
```

Listing 2: UBinarySearch.pas: Implementation of unbounded binary search

Listing 3 shows a class implementing a search on $\mathbb{Q}$. This is separated from listing 2 as when considering the case of integers, this effectively degenerates into a slow linear search (taking the consecutive upper mediants of $\frac{0}{1}$ and $\frac{1}{0}$ results in the sequence $\frac{1}{1}$, $\frac{2}{1}$, $\frac{3}{1}$…).

```
1   {$MODE OBJFPC}
2
3   unit USternBrocotSearch;
4
5   interface
6
7   uses UGuesser, SysUtils;
8
9   type
10    // class to descend a Stern-Brocot tree in search of a rational, with added
11    // logic to deal with negative rationals.
12    // Stern-Brocot searching works by taking the mediant of upper and lower
13    // bounds. The median is guaranteed to be strictly between a and b, but also
14    // preserves the property of "lower complexity" - ie the denominator grows
15    // additively rather than multiplicatively. By setting the upper bounds as
16    // 1/0, we can easily search the entirety of Q+, and need only determine sign.
17    TSternBrocotSearcher = class(TGuesser)
18    protected
19      // flags to determine current stage of guessing
20      has_sign, is_nonzero, is_mid: boolean;
21      // numerator and denominator for each fraction
22      lo_n, lo_d, mid_n, mid_d, up_n, up_d: integer;
23    public
24      // user interfacing methods
```

```
25      constructor Create;
26      function ask_question: string; override;
27      procedure answer_question(reply: boolean); override;
28    end;
29
30  implementation
31
32  constructor TSternBrocotSearcher.Create;
33  begin
34    has_sign := False;
35    is_nonzero := False;
36    is_mid := True;
37    up_n := 1; up_d := 0;
38    lo_n := 0; lo_d := 1;
39  end;
40
41  function TSternBrocotSearcher.ask_question: string;
42  begin
43    if not is_nonzero then
44      result := 'Is q = 0?'
45    else if not has_sign then
46      result := 'Is q \ge 0?'
47    else if is_mid then begin
48      mid_n := lo_n + up_n;
49      mid_d := lo_d + up_d;
50      result := Format('Is q = \frac{%d}{%d}?', [mid_n, mid_d]);
51    end else
52      result := Format('Is q > \frac{%d}{%d}?', [mid_n, mid_d]);
53  end;
54
55  procedure TSternBrocotSearcher.answer_question(reply: boolean);
56  begin
57    if not is_nonzero then
58      if reply then
59        raise EGuessSuccessful.Create('Your number was 0')
60      else
61        is_nonzero := True
62    else if not has_sign then begin
63      // for negative q, set lower bound to -1/0 (acting as -inf)
64      if not reply then begin
65        lo_n := -1; lo_d := 0;
66        up_n :=  0; up_d := 1;
67      end;
68      has_sign := True;
69    end else if is_mid then
70      if reply then
71        raise EGuessSuccessful.Create(Format('Your number was \frac{%d}{%d}', [mid_n,
          ↪  mid_d]))
72      else
73        is_mid := False
74    else begin
```

```
75      if reply then begin
76        lo_n := mid_n; lo_d := mid_d;
77      end else begin
78        up_n := mid_n; up_d := mid_d;
79      end;
80      is_mid := True;
81    end;
82  end;
83
84  end.
```

Listing 3: USternBrocotSearch.pas: Implementation of unbounded rational search

Listing 4 shows a dummy class that just displays a message whenever it is queried. This is useful for the main form code to show a persistent message to the user while no other searching engine is instantiated.

```
1   {$MODE OBJFPC}
2
3   unit UDummyGuesser;
4
5   interface
6
7   uses UGuesser, SysUtils;
8
9   const
10    STARTING_BOUND = 64;
11
12  type
13    TDummyGuesser = class(TGuesser)
14    protected
15      msg: string;
16    public
17      constructor Create(msg_: string);
18      function ask_question: string; override;
19      procedure answer_question(reply: boolean); override;
20    end;
21
22  implementation
23
24  constructor TDummyGuesser.Create(msg_: string);
25  begin
26      msg := msg_;
27  end;
28
29  function TDummyGuesser.ask_question: string;
30  begin
31      result := msg;
32  end;
33
34  procedure TDummyGuesser.answer_question(reply: boolean);
35  begin
```

```
36 end;
37
38 end.
```

Listing 4: UDummyGuesser.pas: Dummy message-displaying object

2.2 Boring

Listing 5 contains an abridged version of the lfm (Lazarus Forms) file, serving as a brief summary of all that I did in the object inspector.

```
1  object Guessing: TGuessing
2    Caption = 'Guessing game'
3    KeyPreview = True
4    OnCreate = FormCreate
5    OnKeyPress = KeyIntercept
6    object YesBtn: TButton
7      Caption = 'Yes'
8      OnClick = YesBtnClick
9    end
10   object NoBtn: TButton
11     Caption = 'No'
12     OnClick = NoBtnClick
13   end
14   object QuestionLbl: TLabel
15     Caption = 'This is where the question goes'
16   end
17   object IntBtn: TButton
18     Caption = 'n \in \mathbb{Z}'
19     OnClick = SetIntSearch
20   end
21   object FracBtn: TButton
22     Caption = 'q \in \mathbb{Q}'
23     OnClick = SetFracSearch
24   end
25   object ExplanationLbl: TLabel
26     Caption = 'The GUI-ssing game'
27   end
28 end
```

Listing 5: (Heavily redacted) UGUIssing.lfm: Layout and programmatic properties of Form elements

Listing 6 shows the 'main' code that deals with the `TForm`. This part implements all the callbacks specified for each form component, and relays the user's actions to the `TGuesser` currently in action.

My favourite part of this section is `KeyIntercept`, which enables the user not to have to press any buttons or really use the GUI at all, upgrading its usefulness to near CLI levels.

```
1 unit UGUIssing;
2
3 {$mode objfpc}{$H+}
```

```
4
5   interface
6
7   uses
8     // Lazarus forms units
9     Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
10    ExtCtrls,
11
12    // Custom units from project
13    UGuesser, UBinarySearch, USternBrocotSearch, UDummyGuesser;
14
15  type
16    TGuessing = class(TForm)
17      FracBtn, IntBtn: TButton;
18      YesBtn, NoBtn: TButton;
19      ExplanationLbl: TLabel;
20      QuestionLbl: TLabel;
21      procedure FormCreate(Sender: TObject);
22      procedure YesBtnClick(Sender: TObject);
23      procedure NoBtnClick(Sender: TObject);
24      procedure SetIntSearch(Sender: TObject);
25      procedure SetFracSearch(Sender: TObject);
26      procedure KeyIntercept(Sender: TObject; var Key: char);
27      procedure AnsQn(reply: boolean);
28      procedure AskQn;
29    private
30      Guesser: TGuesser;
31    end;
32
33  var
34    Guessing: TGuessing;
35
36  implementation
37
38  {$R *.lfm}
39
40  { TGuessing }
41
42  procedure TGuessing.FormCreate(Sender: TObject);
43  begin
44    writeln('Initialising game');
45    ExplanationLbl.Font.Size := 30;
46    Guesser := TDummyGuesser.Create(
      ↪  'Welcome to the guessing game! Think of a number and select its domain.');
47    AskQn;
48  end;
49
50  procedure TGuessing.SetIntSearch(Sender: TObject);
51  begin
52    writeln('Entering unbounded integer binary search');
53    Guesser.Free;
```

```
54    Guesser := TBinarySearcher.Create;
55    AskQn;
56  end;
57
58  procedure TGuessing.SetFracSearch(Sender: TObject);
59  begin
60    writeln('Entering Stern-Brocot search');
61    Guesser.Free;
62    Guesser := TSternBrocotSearcher.Create;
63    AskQn;
64  end;
65
66  procedure TGuessing.YesBtnClick(Sender: TObject);
67  begin
68    writeln('Answer Yes');
69    AnsQn(true);
70  end;
71
72  procedure TGuessing.NoBtnClick(Sender: TObject);
73  begin
74    writeln('Answer No');
75    AnsQn(False);
76  end;
77
78  procedure TGuessing.KeyIntercept(Sender: TObject; var Key: char);
79  begin
80    writeln('Pressed key ', Key);
81    case Key of
82      'y':
83        YesBtnClick(Sender);
84      'n':
85        NoBtnClick(Sender);
86      'z':
87        SetIntSearch(Sender);
88      'q':
89        SetFracSearch(Sender);
90    end;
91  end;
92
93  procedure TGuessing.AnsQn(reply: boolean);
94  begin
95    try
96      Guesser.answer_question(reply);
97    except
98    on e: EGuessSuccessful do begin
99      writeln('Guessed: ', e.Message);
100     Guesser.Free;
101     Guesser := TDummyGuesser.Create(e.Message +
        ↪  ' Think of another number, and select its domain.');
102   end;
103   end;
```

```
104   AskQn;
105 end;
106
107 procedure TGuessing.AskQn;
108 begin
109   QuestionLbl.Caption := Guesser.ask_question;
110   writeln('Question: ', QuestionLbl.Caption);
111 end;
112
113 end.
```

Listing 6: UGUIssing.pas: Implementing the Forms functionality

3 Result

A screenshot of the application is shown in fig 1. The colouring is not specifically set for this game, but is just my personal GTK theme (Arc-Dark), which is detected by Lazarus.

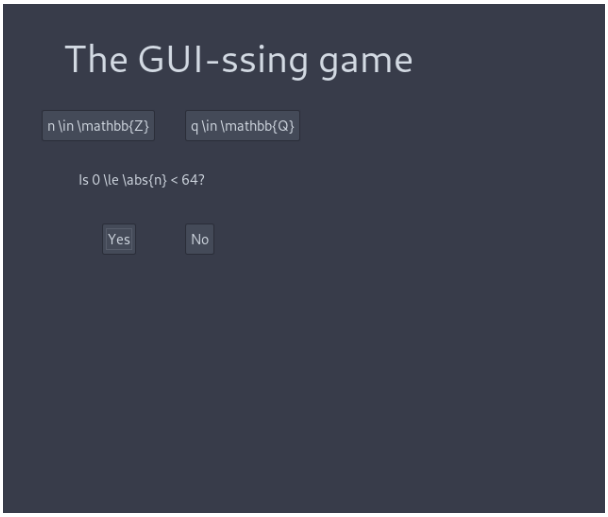

Figure 1: Screenshot of the game

You may notice that all of the mathematics is formatted in the form of LaTeX source. This is the second best way to format maths, beside rendered LaTeX.

The program can be interacted with by clicking the buttons, or, as covered earlier, by pressing the appropriate keys. Because of the various `writeln` statements I had included, I can easily capture the actions executed by the user and program without needing multiple screenshots. I first compiled using `lazbuild` or Lazarus, and then ran `./GUIssing > writeup/output.txt`. This produced the following file, after I executed a guess for $\forall S \in \{\mathbb{Z}, \mathbb{Q}\}$.

```
1 Initialising game
2 Question: Welcome to the guessing game! Think of a number and select its domain.
3 Pressed key z
4 Entering unbounded integer binary search
5 Question: Is 0 \le \abs{n} < 64?
6 Pressed key y
7 Answer Yes
```

```
Question: Is n \ge 0?
Pressed key n
Answer No
Question: Is n \ge -31?
Pressed key y
Answer Yes
Question: Is n \ge -15?
Pressed key y
Answer Yes
Question: Is n \ge -7?
Pressed key y
Answer Yes
Question: Is n \ge -3?
Pressed key n
Answer No
Question: Is n \ge -5?
Pressed key y
Answer Yes
Question: Is n \ge -4?
Pressed key n
Answer No
Guessed: Your number was -5.
Question: Your number was -5. Think of another number, and select its domain.
Pressed key q
Entering Stern-Brocot search
Question: Is q = 0?
Pressed key n
Answer No
Question: Is q \ge 0?
Pressed key y
Answer Yes
Question: Is q = \frac{1}{1}?
Pressed key n
Answer No
Question: Is q > \frac{1}{1}?
Pressed key n
Answer No
Question: Is q = \frac{1}{2}?
Pressed key n
Answer No
Question: Is q > \frac{1}{2}?
Pressed key y
Answer Yes
Question: Is q = \frac{2}{3}?
Pressed key n
Answer No
Question: Is q > \frac{2}{3}?
Pressed key n
Answer No
Question: Is q = \frac{3}{5}?
Pressed key n
```

```
59 Answer No
60 Question: Is q > \frac{3}{5}?
61 Pressed key n
62 Answer No
63 Question: Is q = \frac{4}{7}?
64 Pressed key y
65 Answer Yes
66 Guess
```

Listing 7: Example session with program

This demonstrates the program correctly feeding information between the user and the underlying search engine. I have performed several more tests, including the cases

- 0
- 1
- -1
- 200
- -130
- 16

in $\mathbb{Z}$, and

- 0
- 1
- 3
- -3
- $\frac{4}{3}$
- $\frac{5}{2}$
- $-\frac{5}{2}$

in $\mathbb{Q}$. These were not included as entire logs as this would be a serious waste of paper.

4 Source

The full project in its directory structure, including this document (as a full-colour PDF and TEX file), can be found at `https://github.com/elterminad0r/GUIssing`.

I would also like to take this moment to apologise for the aesthetics of my previous PDFs, including the usage of the Computer Modern Font, but especially the fact that the code listings didn't use a proper fixed-width font.