GUIssing game

Izaak van Dongen

June 24, 2019

Contents

List of Listings

1 Introduction

This project is the sequel to the popular `assignment_guessing`, at `https://github.com/goedel-gang/assignment_guessing`, now featuring a very useful and fluid graphical interface, more object orientation, and cleaner general coding practice.

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 Interesting

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   // base classes
11   TGuesser = class abstract
12     function ask_question: string; virtual; abstract;
13     procedure answer_question(reply: boolean); virtual; abstract;
14   end;
15   EGuessSuccessful = class(Exception);
16
17 implementation
18
19 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 $\mathcal{O}(\log_2(n))$ (and then guess the number in $\mathcal{O}(\log_2(n))$).

```
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 $\frac{1}{1}$, $\frac{2}{1}$, $\frac{3}{1}\ldots$).

```
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    // dummy type to display a message
14    TDummyGuesser = class(TGuesser)
15    protected
16      msg: string;
17    public
18      constructor Create(msg_: string);
19      function ask_question: string; override;
20      procedure answer_question(reply: boolean); override;
21    end;
22
23  implementation
24
25  constructor TDummyGuesser.Create(msg_: string);
26  begin
27      msg := msg_;
28  end;
```

```
29
30 function TDummyGuesser.ask_question: string;
31 begin
32     result := msg;
33 end;
34
35 procedure TDummyGuesser.answer_question(reply: boolean);
36 begin
37 end;
38
39 end.
```

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

2.2 Boring

Listings 5 and 6 contains abridged versions of the lfm (Lazarus Forms) files, serving as a brief summary of all that I did in the object inspector.

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

```
33 end
```

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

```
object UserGuessing: TUserGuessing
  Caption = 'Guessing game'
  KeyPreview = True
  OnCreate = FormCreate
  OnKeyPress = KeyIntercept
  object LoLbl: TLabel
    Caption = 'Lower Bound'
  end
  object UpLbl: TLabel
    Caption = 'Upper Bound'
  end
  object GenerateBtn: TButton
    Caption = 'Generate Number'
    OnClick = GenerateBtnClick
  end
  object AnswerLbl: TLabel
    Caption = 'This is where the answers go'
  end
  object GuessLbl: TLabel
    Caption = 'Guess:'
  end
  object SwitchBtn: TButton
    Caption = 'Back to computer guessing'
    OnClick = SwitchBtnClick
  end
  object LoEdt: TEdit
    MaxLength = 8
    Text = '0'
  end
  object UpEdt: TEdit
    MaxLength = 8
    Text = '128'
  end
  object GuessEdt: TEdit
    MaxLength = 8
    OnEditingDone = GuessEdtEditingDone
    Text = ' '
  end
end
```

Listing 6: (Heavily redacted) UUserGuessing.lfm: See 5

Listing 7 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.

```
unit UComputerGuessing;

{$mode objfpc}{$H+}

interface

uses
  // Lazarus forms units
  Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
  ExtCtrls,
  // Other form
  UUserGuessing,
  // Custom units from project
  UGuesser, UBinarySearch, USternBrocotSearch, UDummyGuesser;

type

  { TComputerGuessing }
  // Main form - where the computer guesses the user's number
  TComputerGuessing = class(TForm)
    SwitchBtn: TButton;
    FracBtn, IntBtn: TButton;
    YesBtn, NoBtn: TButton;
    ExplanationLbl: TLabel;
    QuestionLbl: TLabel;
    procedure FormClose(Sender: TObject; var CloseAction: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure SwitchBtnClick(Sender: TObject);
    procedure YesBtnClick(Sender: TObject);
    procedure NoBtnClick(Sender: TObject);
    procedure SetIntSearch(Sender: TObject);
    procedure SetFracSearch(Sender: TObject);
    procedure KeyIntercept(Sender: TObject; var Key: char);
    procedure AnsQn(reply: boolean);
    procedure AskQn;
  private
    Guesser: TGuesser;
  end;

var
  ComputerGuessing: TComputerGuessing;

implementation

{$R *.lfm}

{ TComputerGuessing }

procedure TComputerGuessing.FormCreate(Sender: TObject);
begin
  writeln('Initialising game');
```

```
52    ExplanationLbl.Font.Size := 30;
53    Guesser := TDummyGuesser.Create(
      ↪  'Welcome to the guessing game! Think of a number and select its domain.'
      ↪  );
54    AskQn;
55  end;
56
57  procedure TComputerGuessing.SwitchBtnClick(Sender: TObject);
58  begin
59    writeln('Switching to user guessing mode');
60    UserGuessing.Show;
61  end;
62
63  procedure TComputerGuessing.FormClose(Sender: TObject; var CloseAction:
    ↪  TCloseAction);
64  begin
65    writeln('Closing application');
66    Guesser.Free;
67  end;
68
69  procedure TComputerGuessing.SetIntSearch(Sender: TObject);
70  begin
71    writeln('Entering unbounded integer binary search');
72    Guesser.Free;
73    Guesser := TBinarySearcher.Create;
74    AskQn;
75  end;
76
77  procedure TComputerGuessing.SetFracSearch(Sender: TObject);
78  begin
79    writeln('Entering Stern-Brocot search');
80    Guesser.Free;
81    Guesser := TSternBrocotSearcher.Create;
82    AskQn;
83  end;
84
85  procedure TComputerGuessing.YesBtnClick(Sender: TObject);
86  begin
87    writeln('Answer Yes');
88    AnsQn(true);
89  end;
90
91  procedure TComputerGuessing.NoBtnClick(Sender: TObject);
92  begin
93    writeln('Answer No');
94    AnsQn(False);
95  end;
96
97  procedure TComputerGuessing.KeyIntercept(Sender: TObject; var Key: char);
98  begin
99    write(Format('Pressed key %s ', [Key]));
```

```
100   case Key of
101     'y':
102       YesBtnClick(Sender);
103     'n':
104       NoBtnClick(Sender);
105     'z':
106       SetIntSearch(Sender);
107     'q':
108       SetFracSearch(Sender);
109     's':
110       SwitchBtnClick(Sender);
111   end;
112 end;
113
114 procedure TComputerGuessing.AnsQn(reply: boolean);
115 begin
116   try
117     Guesser.answer_question(reply);
118   except
119   on e: EGuessSuccessful do begin
120     writeln('Guessed: ', e.Message);
121     Guesser.Free;
122     Guesser := TDummyGuesser.Create(e.Message +
      ↪  ' Think of another number, and select its domain.');
123   end;
124   end;
125   AskQn;
126 end;
127
128 procedure TComputerGuessing.AskQn;
129 begin
130   QuestionLbl.Caption := Guesser.ask_question;
131   writeln('Question: ', QuestionLbl.Caption);
132 end;
133
134 end.
```

Listing 7: UComputerGuessing.pas: Implementing the Forms functionality

Listing 8 contains similar code but for the mode where the user must guess the computer's number. This is perhaps even more mundane as really all it does is handle a lot of conversions and check a couple of cases.

```
1  unit UUserGuessing;
2
3  {$mode objfpc}{$H+}
4
5  interface
6
7  uses
8    Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
```

```
9     MaskEdit, LCLType;
10
11  type
12
13    { TUserGuessing }
14    // Auxiliary form letting a user guess the computer's number
15    TUserGuessing = class(TForm)
16      GuessEdt: TEdit;
17      LoEdt: TEdit;
18      SwitchBtn: TButton;
19      GenerateBtn: TButton;
20      AnswerLbl: TLabel;
21      GuessLbl: TLabel;
22      LoLbl: TLabel;
23      UpEdt: TEdit;
24      UpLbl: TLabel;
25      procedure FormCreate(Sender: TObject);
26      procedure GenerateBtnClick(Sender: TObject);
27      procedure GuessEdtEditingDone(Sender: TObject);
28      procedure KeyIntercept(Sender: TObject; var Key: char);
29      procedure SwitchBtnClick(Sender: TObject);
30    private
31      Num: integer;
32    end;
33
34  var
35    UserGuessing: TUserGuessing;
36
37  implementation
38
39  {$R *.lfm}
40
41  { TUserGuessing }
42
43  procedure TUserGuessing.FormCreate(Sender: TObject);
44  begin
45    writeln('Initialising user-guessing game');
46    randomize;
47    GenerateBtnClick(Sender);
48  end;
49
50  procedure TUserGuessing.GenerateBtnClick(Sender: TObject);
51  var
52    lo, up: integer;
53    success: boolean = True;
54  begin
55    success := success and TryStrToInt(LoEdt.Text, lo);
56    success := success and TryStrToInt(UpEdt.Text, up);
57    if success then begin
58      Num := random(up - lo) + lo;
59      writeln(Format('Generated %d between %d, %d', [Num, lo, up]));
```

```
60      AnswerLbl.Caption := Format('I''m thinking of a number between %d and %d',
        ↪  [lo, up]);
61    end else
62      AnswerLbl.Caption := 'Invalid boundary';
63  end;
64
65  procedure TUserGuessing.GuessEdtEditingDone(Sender: TObject);
66  var
67    guess: integer;
68  begin
69    if TryStrToInt(GuessEdt.Text, guess) then begin
70      if guess = Num then
71        AnswerLbl.Caption := 'Correct! You guessed it'
72      else if guess < Num then
73        AnswerLbl.Caption := 'Too low!'
74      else
75        AnswerLbl.Caption := 'Too high!';
76    end else
77      AnswerLbl.Caption := 'Invalid number';
78  end;
79
80  procedure TUserGuessing.KeyIntercept(Sender: TObject; var Key: char);
81  begin
82    write(Format('Pressed key %s ', [Key]));
83    case Key of
84      'g':
85        GenerateBtnClick(Sender);
86      'b':
87        SwitchBtnClick(Sender);
88    end;
89  end;
90
91  procedure TUserGuessing.SwitchBtnClick(Sender: TObject);
92  begin
93    writeln('Closing user-guessing game');
94    Close;
95  end;
96
97  end.
```

Listing 8: UUserGuessing.pas: Implementing the Forms functionality

My favourite part of the form code 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.

3 Result

Screenshots of the application are shown in fig 1. The colouring is not specifically set for this game, but is just my personal GTK theme (Arc-Dark), which is used as Lazarus implement Forms with GTK2.

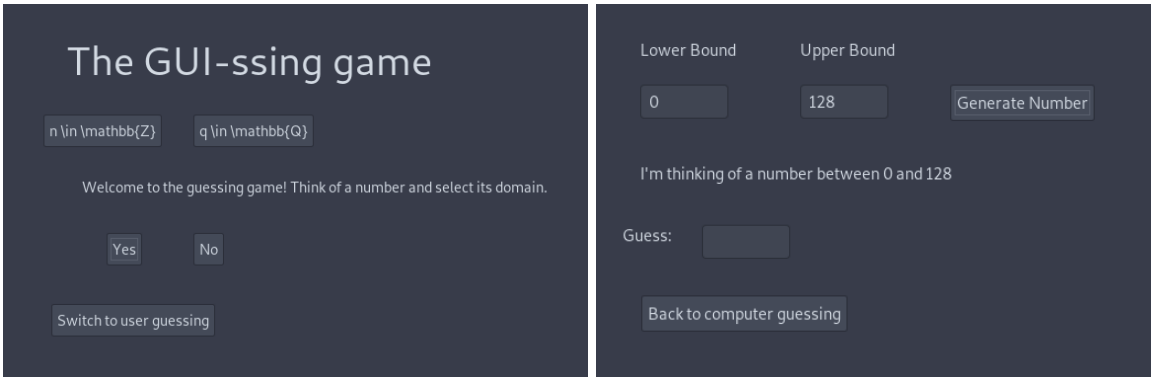

Figure 1: Screenshots of the game

You may notice that all of the mathematics is formatted in the form of LaTeX source. This is the second best[1] 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. I have also chosen `TLabel` to communicate with the user rather than a `TListBox` as it is more compact, and I feel better represents the spirit of a guessing game (I consider it a vital component that your internal state should be kept in your head, and would be furious if my opponent were to write anything down).

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 file in listing 9, 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  Initialising user-guessing game
4  Generated 114 between 0, 128
5  Pressed key z Entering unbounded integer binary search
6  Question: Is 0 \le \abs{n} < 64?
7  Pressed key n Answer No
8  Question: Is 64 \le \abs{n} < 128?
9  Pressed key y Answer Yes
10 Question: Is n \ge 0?
11 Pressed key n Answer No
12 Question: Is n \ge -95?
13 Pressed key n Answer No
14 Question: Is n \ge -111?
15 Pressed key y Answer Yes
16 Question: Is n \ge -103?
17 Pressed key y Answer Yes
18 Question: Is n \ge -99?
19 Pressed key n Answer No
20 Question: Is n \ge -101?
21 Pressed key y Answer Yes
```

[1] For example, it beats straight utf-8 because it allows a user to render the maths using whatever font they like in their head

```
22 Question: Is n \ge -100?
23 Pressed key n Answer No
24 Guessed: Your number was -101.
25 Question: Your number was -101. Think of another number, and select its
   ↪ domain.
26 Pressed key q Entering Stern-Brocot search
27 Question: Is q = 0?
28 Pressed key n Answer No
29 Question: Is q \ge 0?
30 Pressed key y Answer Yes
31 Question: Is q = \frac{1}{1}?
32 Pressed key n Answer No
33 Question: Is q > \frac{1}{1}?
34 Pressed key y Answer Yes
35 Question: Is q = \frac{2}{1}?
36 Pressed key n Answer No
37 Question: Is q > \frac{2}{1}?
38 Pressed key n Answer No
39 Question: Is q = \frac{3}{2}?
40 Pressed key n Answer No
41 Question: Is q > \frac{3}{2}?
42 Pressed key n Answer No
43 Question: Is q = \frac{4}{3}?
44 Pressed key y Answer Yes
45 Guessed: Your number was \frac{4}{3}
46 Question: Your number was \frac{4}{3} Think of another number, and select its
   ↪ domain.
47 Pressed key s Switching to user guessing mode
48 Closing application
```

Listing 9: Example log of 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

in $\mathbb{Z}$:	0	1	-1	200	-130	16	
in $\mathbb{Q}$:	0	1	3	-3	$\frac{4}{3}$	$\frac{5}{2}$	$-\frac{5}{2}$

These were not included as entire logs as this would be a serious waste of paper.

Note that this program will fail if the user decided on a number not representible by a Pascal `integer`, as shown in listing 10. To solve this problem permanently would require an implementation of some kind of `BigInteger`. This would require lots more memory management and boilerplate, which in turn needs time. I'm currently mildly strapped for time so I've opted to let it break.

This is not a problem for the computer-generated guessing mode as each TEdit limits its input to 8 characters, leaving the user to input at largest $10^8 - 1$. This is well within the range of a signed 32-bit integer ($|n| \lessapprox 2.1 \cdot 10^9$). Another non-bug in this part is if the user enters a lower bound larger than the upper bound (let $a > b$). In this case, a random number in the interval $[a, 2a - b) \cap \mathbb{Z}$ is generated. For example, $a = 100, b = 50$ results in a number in $\{100..149\}$. This is due to the fact that Pascal's `random` function treats its input as an absolute value. This

is ideal behaviour[2].

```
Question: Is 536870912 \le \abs{n} < 1073741824?
Pressed key n
Answer No
Question: Is 1073741824 \le \abs{n} < -2147483648?
Pressed key n
Answer No
Question: Is -2147483648 \le \abs{n} < 0?
Pressed key n
Answer No
Question: Is 0 \le \abs{n} < 0?
Pressed key n
Answer No
Question: Is 0 \le \abs{n} < 0?
```

Listing 10: Overflow leading to critical failure

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/goedel-gang/GUIssing`.

[2] Users trying to mess with the program get messed with instead