# Play Minesweeper automatically!

by Paul J. Fenwick
brian.d.foy@gmail.com

You sit down at the machine, it's running Windows. Those other operating systems are pretty good, but they're lacking that one killer app, the one reason why Windows still has market dominance: Minesweeper!

Yes, good old Minesweeper! You'll just play one game, on intermediate and then start on your real work. Just a little morsel of relaxation. Three hours later, you're wondering what happened to your day.

Does this story sound familiar or have you a friend or work-mate who's in the jaws of the Minesweeper vice? I know that I was, so one day, I decided to do something about it. I'd program my machine to play Minesweeper for me.

I'd automated the rest of my life; I have programs to read my mail, to plan my exercise, and to pay my bills. Automating my leisure time was the obvious next step. Having a program to play Minesweeper for me would give me more time for the more important things in life, like Solitaire. It was from this realisation that `App::SweeperBot` was born.

## Why a Minesweeper bot?

I'm a lazy person. I hate having to do things myself, and I really hate having to write my own code. Indeed, one of the motivating factors for writing `App::SweeperBot` is that much of the hard work had already been done for me.

Matt Sparks (*http://www.f0rked.com*) had already written an almost complete Minesweeper bot, doing what I had considered all the "hard parts". Matt's code was able to locate the board, start a new game, and identify the contents of various squares. All I had to do was to write an algorithm to get it to play.

Of course, when asked by serious professionals with lots of money why I wrote `App::SweeperBot`, I claim it's because it provides an excellent working example of being able to examine and control GUI applications using Perl. These techniques can be a great advantage in testing and automation.

## Locating Minesweeper

The first step for SweeperBot in playing Minesweeper is to locate the game on the screen. SweeperBot uses the `Win32::GuiTest` module for GUI manipulation, and this also makes locating the Minesweeper board easy:

```
use Win32::GuiTest qw(FindWindowLike);

my ($window_id) = FindWindowLike(
   0, '^Minesweeper' );
```


`FindWindowLike` returns a list of matching windows, but I'm only interested in the first one. The first argument to `FindWindowLike` is the root of my window search; it can be an existing window if I'm looking for a dialog box or similar window spawned by an application, but for SweeperBot we're using `0` to look for any window that matches my criteria.

The second argument is a regexp that matches my window name. In the case of SweeperBot, I'm simply looking for any window that starts with `Minesweeper`.

Once I've found my window, the next task is to determine how large it is; SweeperBot needs this information to determine the size of the Minesweeper grid, and to locate the all-important smiley face that's used to determine our game state (won/lost/playing) and to reset the game.

```
use Win32::GuiTest qw(GetWindowRect);

my ($left, $top, $right, $bottom)
   = GetWindowRect($window_id);

my $width  = $right  - $left;
my $height = $bottom - $top;
```

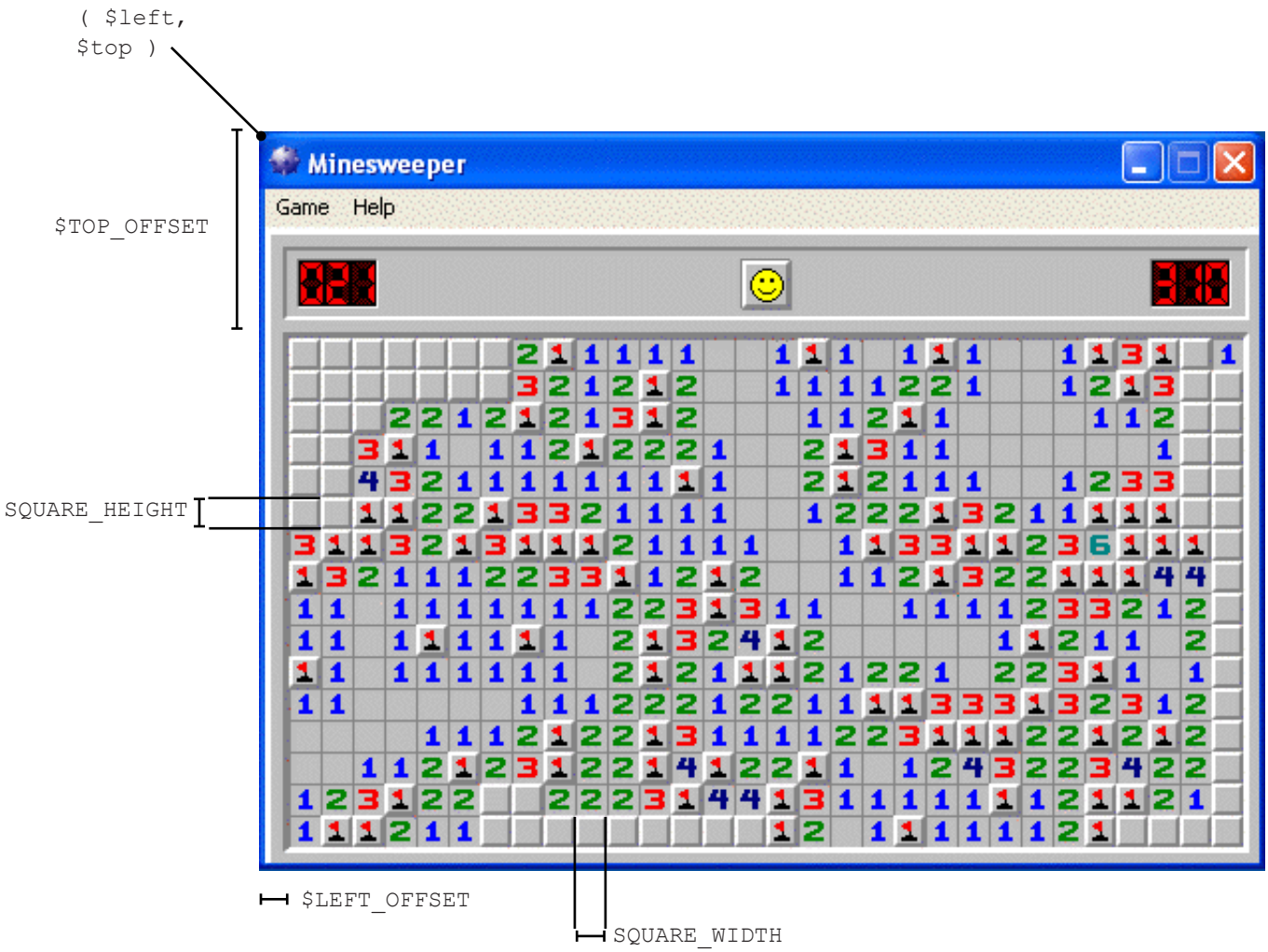


*Figure 1: To get to any particular square, I need to know how to get past the window decorations and into the game grid. Each uncovered square is either blank, or has a colored numeral. Each numeral has a different color. Flags represent squares I think have mines. The happy face at center top means I'm still in the game.*

The `GetWindowRect` function returns the window location, with the top-left of the entire display being considered pixel (0,0).

I don't show the calculations for determining the size of the grid and the position of the smiley, but they mostly consist of basic arithmetic (each square is 16 by 16 pixels, and the smiley is always in the centre-top part of the window).

*For more about Windows programming with Perl, visit*

*http://win32.perl.org*

## ■ Reading the screen----------

To actually read the screen I use another module, `Win32::Screenshot`, which provides a `CaptureRect` function for capturing parts of the screen:

```
use Win32::Screenshot;

my $image = CaptureRect(
  $left + $LEFT_OFFSET + $sx * SQUARE_WIDTH,
  $top  + $TOP_OFFSET  + $sy * SQUARE_HEIGHT,
  SQUARE_WIDTH,
  SQUARE_HEIGHT
  );
```

The CaptureRect function takes an X,Y offset to the top-left of the rectangle desired, and the desired width and height of the rectangle.

$LEFT_OFFSET and $TOP_OFFSET are values that take me to the top-left of the Minesweeper grid, past any window decorations. $sx and $sy are the square SweeperBot wishes to sample, and SQUARE_WIDTH and SQUARE_HEIGHT are constants containing the width and height of a Minesweeper square; in this case they're 16 pixels each. Figure 1 shows how these variables relate to the Minesweeper window.

As a convention, I use $sx and $sy when measuring an offset in Minesweeper squares, and $x and $y when measuring in pixels. Without this convention, parts of the application code would be significantly more difficult to understand.

The resulting $image is actually an Image::Magick object, which is extremely versatile, and therefore provide a great amount of flexibility in how I can analyse the information contained within.

## ■ What's in that square? ----------

SweeperBot takes a very simple approach to determining the contents of each square; it simply takes a SHA-256 digest of the image (a fingerprint) and looks that up in a hash of known values:

```
my $signature = $image->Get("signature");

my $contents =
   $contents_of_square{$signature};
```

This works admirably, but as I discovered somewhat later is not particularly portable. The results of Get("signature") can vary depending upon the version of Image::Magick installed, so SweeperBot needs to contain multiple signatures to cover all of them.

## ■ Manipulating the mouse ----------

The Win32::GuiTest module provides a way both to move the mouse and press buttons, and using this I can construct a simple subroutine to click on an arbitrary part of the display:

```
use Win32::GuiTest
   qw(MouseMoveAbsPix SendMouse);

sub click {
   my ($this, $x, $y, $button) = @_;
   $button ||= '{LEFTCLICK}';
   MouseMoveAbsPix($x, $y);
   SendMouse($button);
   return;
   }
```

However I don't want to be clicking anywhere on our display, I specifically want to be clicking on Minesweeper squares. Luckily, I already know the size of them, and the position of our Minesweeper window, so writing a subroutine to do this becomes easy:

```
sub press {
   my ($this, $sx, $sy, $button) = @_;

   $this->click(
      $left + $LEFT_OFFSET + $sx *
           SQUARE_WIDTH  + SQUARE_WIDTH/2,
      $top  + $TOP_OFFSET  + $sy *
           SQUARE_HEIGHT + SQUARE_HEIGHT/2,
      $button
      );

   return;
   }
```

The SQUARE_WIDTH/2 and SQUARE_HEIGHT/2 terms ensure we're clicking on the middle of each square.

We can now use press to easily define subroutines that flag a mine (right-click) or pressing all squares adjacent to it (middle-click, what I call a "stomp").

```
sub flag_mine {
   my ($this, $sx, $sy) = @_;

   return $this->press(
      $sx, $sy, '{RIGHTCLICK}');
   }

sub stomp {
   my ($this, $sx, $sy) = @_;

   return $this->press(
      $sx, $sy, '{MIDDLECLICK}');
   }
```

## ■ Putting it all together ----------

With the techniques seen, it's now possible to play a basic game of Minesweeper. I have code to locate the window, sample the contents, and manipulate controls.

In App::SweeperBot, the algorithm to actually play is found in make_move, which I won't reproduce here. Put simply, it walks through every square and looks at all the adjacent squares at each point. If by examining just those adjacent square it can determine that a square is a mine or safe, then it flags or presses the square as appropriate. If SweeperBot walks through the entire board without flagging or pressing any squares, it takes a guess and steps on a random covered square.

This isn't an optimal Minesweeper strategy since looking further than just the adjacent squares can reveal more information about each square. Likewise, when guessing, it should be possible to use the information available to determine the safest guesses and try those first. Presently, SweeperBot does neither.

Luckily, you can write your own algorithm for playing by sub-classing App::SweeperBot:

```
package My::SweeperBot;

use base qw(App::SweeperBot);

sub make_move {
   my ($this, $state) = @_;

   # Make your moves here. If you haven't
   # won or lost, it will re-sample
   # the board and call make_move again.
   }
```

The `$state` variable is a reference to an array of arrays which represent the board. The top-left square is `[1,1]` (and not `[0,0]`), since that's what Matt's original code used.

## ■ Cheating

Because my algorithm for playing Minesweeper isn't very good, SweeperBot can also be run using the xyzzy cheat, that reveals information about the contents of each mine by changing the colour of top-left pixel of the entire display. If cheating is enabled (it is by default) then SweeperBot will reconsider if picking a square at random would result in hitting a mine.

I have Audrey Tang to thank for information about the xyzzy cheat; having a bot that wins every time is much more impressive than one that wins only some of the time.

## ■ Packaging

An application isn't any good unless people actually use it. However for a typical Windows user, installing Perl can seem like quite a bit of effort, let alone installing a set of custom modules and their dependencies. In fact, just getting `Win32::Screenshot`, `Image::Magick` and `Win32::GuiTest` working together on my own system required more work than I had expected.

Luckily, packaging stand-alone applications on Windows becomes much easier with the use of PAR, the Perl Archiver. Put simply, PAR allows a Perl program, its associated modules, and any other files it depends upon to be bundled up into a single file. Under the hood, it uses the excellent `Module::ScanDeps` to determine what additional modules should be included; so for more programs it works right out of the box.

I used `PAR::Packer`, or more specifically its command-line `pp` program, to create my stand-alone application. Because it needed to contain my code, a perl interpreter, supporting modules, and a number of `Image::Magick` DLL files, the resulting program is seven megabytes in size. Despite the large size, I now had something that could be dropped onto a virgin Windows system and work with nothing more than the user double-clicking the icon.

One particular challenge was finding a portable version of `Image::Magick` that does not depend upon local registry settings, which PAR cannot bundle. Luckily, J-L Morel has a pre-built `Image::Magick` distribution at *http://bribes.org/perl/ppmdir.html* which doesn't use the Windows registry settings at all. As advertised, it's completely compatible with PAR.

To see the complete build script that produces the executable, check out `MakeExe.pl` that comes with the `App::SweeperBot` distribution. This file can also be found at *http://github.com/pfenwick/sweeperbot/tree/master/MakeExe.pl*.

## ■ Improvements

SweeperBot has ample room for improvement. The current method of sampling each square individually, computing an SHA-256 hash of the contents, and using the result to determine the square contents is extremely slow, although it was the "lazy" option that allowed me to re-use the largest amount of Matt's original code. A better way would be to sample the entire board at once. Since each numbered square uses a different colour, individual pixels could be examined to determine the board configuration. I imagine such an improvement would increase SweeperBot's speed by an order of magnitude or more.

Changing the methods for sampling the board also provides for the potential to remove the dependency on `Image::Magick`, which is responsible for a significant portion of the final executable.

Much of the code in `App::SweeperBot` reflects its history of being a stand-alone script. As such, it's not possible to create multiple SweeperBot objects for playing multiple games of Minesweeper simultaneously, since too much information is read from package variables. This information should be moved to be internal to each individual `App::SweeperBot` object.

The other obvious flaw with SweeperBot is that it plays Minesweeper only about as well as I do. Luckily this can easily be changed by sub-classing and writing a new `make_move` method, as described earlier in this article.

## ■ Further Information

SweeperBot has its own webpage at *http://sweeperbot.org/*, which includes an FAQ, links to the source code, and a promotional video explaining why SweeperBot is an excellent productivity tool.

The SweeperBot code repository is managed with git. Even if you've never used git before, you can still obtain a copy of the source with:

```
git clone git://github.com/pfenwick/sweeperbot.
git
```

You can also browse the source and its history on-line at *http://github.com/pfenwick/sweeperbot*.

Perl Training Australia (*http://perltraining.com.au*) has written a short tutorial on PAR, which also contains links to other resources. It can be found at *http://perltraining.com.au/tips/2008-05-23.html*

## ■ About the author

Paul Fenwick is the managing director of Perl Training Australia. He is the author of the new `autodie` pragma, experienced international conference speaker, and author of many of Perl Training Australia's free Perl Tips (*http://perltraining.com.au/tips/*). His interests include coffee, mycology, scuba diving, applied statistics, and lexically-scoped user pragmata.