```
function russell_demo()

% Do the example in ch 17 (p501) of Russell and Norvig
% (1,1) is top left corner.

r = 3; c = 4; p = 0.8; action_cost = -1/25;
obstacle = zeros(r,c); obstacle(2,2)=1;
terminal = zeros(r,c); terminal(1,4)=1; terminal(2,4)=1;
absorb = 1;
wrap_around = 0;
noop = 0;
T = mk_grid_world(r, c, p, obstacle, terminal, absorb, wrap_around, noop);
% Add rewards for terminal states
nstates = r*c + 1;
if noop
  nact = 5;
else
  nact = 4;
end
R = action_cost*ones(nstates, nact);
R(10,:)  = 1;
R(11,:)  = -1;
R(nstates,:)  = 0;
discount_factor = 1;


V = value_iteration(T, R, discount_factor);
%reshape(V(1:end-1),[r c])
%    0.8116    0.8678    0.9178    1.0000
%    0.7616    0.7964    0.6603   -1.0000
%    0.7053    0.6553    0.6114    0.3878
% Same as the book p501

Q = Q_from_V(V, T, R, discount_factor);
[V, p] = max(Q, [], 2);


use_val_iter = 1;
% (I-gT) is singular since g=1 and there is an absorbing state (i.e., T(i,i)=1)
% Hence we cannot use value determination.
[p,V] = policy_iteration(T, R, discount_factor, use_val_iter);

%reshape(V(1:end-1),[r c])
%    0.8115    0.8678    0.9178    1.0000
%    0.7615    0.7964    0.6603   -1.0000
%    0.7048    0.6539    0.6085    0.3824
```

```
%M \logo [VIM] {VIM} % Needed for the title

%D \module
%D   [      file=t-vim,
%D        version=2006.12.26,
%D          title=\VIM\ to \CONTEXT,
%D       subtitle=Use \VIM\ to generate code listing,
%D         author=Mojca Miklavec \& Aditya Mahajan,
%D          email=adityam at umich dot edu,
%D           date=\currentdate,
%D      copyright=Public Domain]

%M \usemodule [vim]
%M \usemodule[int-load]
%M \loadsetups[t-vim.xml]
%M \usetypescript[modern-base][texnansi] \setupbodyfont[modern]
%M \setuptyping[option=color]

%D \section   {User Manual}
%D
%D \CONTEXT\ has an excellent pretty printing capabilities for many languages.
%D The code for pretty printing is written in \TEX, and due to catcode
%D jugglary verbatim typesetting is perhaps the trickiest part of \TEX. This
%D makes it difficult for a \quotation{normal} user to define syntax
%D highlighting rules for a new language.  This module, takes the onus of
%D defining syntax highlighting rules away from the user and uses \VIM\ editor
%D to generate the syntax highlighting.    There is a helper
%D \filename{2context.vim} script to do the syntax parsing in \VIM. This is a
%D stop|-|gap method, and hopefully with \LUATEX, things will be much easier.
%D
%D The main macro of this module is \type{\definevimtyping}.  The best way to
%D explain it is by using an example.  Suppose you want to pretty print ruby
%D code in \CONTEXT. So you can do
%D \starttyping
%D \definevimtyping [RUBY]  [syntax=ruby]
%D \stoptyping
%D after which you can get ruby highlighting by
%D \starttyping
%D \startRUBY
%D ....
%D \stopRUBY
%D \stoptyping
%D
%D For example
%D \startbuffer
%D \definevimtyping [RUBY] [syntax=ruby]
%D
%D \startRUBY
```

```
%D #!  /usr/bin/ruby
%D # This is my first ruby program
%D puts "Hello World"
%D \stopRUBY
%D \stopbuffer
%D {\getbuffer}
%D This was typed as \typebuffer
%D
%D The typing can be setup using \type{\setupvimtyping}.
%D
%D \showsetup{setupvimtyping}
%D
%D Here \type{syntax} is the syntax file in \VIM\ for the language
%D highlighting that you want.  See \type{:he syntax.txt} inside \VIM\ for
%D details.  \type{colorscheme} provides the sytax highlighting for various
%D regions.  Right now, only one colorscheme (\type{default}) is defined.  It is
%D based on \filename{ps_color.vim} colorscheme in \VIM. If there is a
%D particular colorscheme that you will like, you can convert it into
%D \CONTEXT. \type{space=(yes|on|no)} makes the space significant, visible,
%D and unsignificant respectively.  \type{tab} specifies the number of spaces a
%D tab is equivalent to.  It's default value is 8.  \type{start} and \type{stop}
%D specify which lines to read from a file.  These options only make sense for
%D highlighting files and should not to be set by \type{\setupvimtyping}.
%D \type{numbering} enables line numbering, and \type{step} specifies which
%D lines are numbered.  \type{numberstyle} and \type{numbercolor} specify the
%D style and color of line numbers.
%D
%D A new typing region can be define using \type{\definevimtyping}.
%D
%D \showsetup{definevimtyping}
%D
%D Minor changes in syntax highlighting can be made easily.  For example, Mojca
%D likes \quote{void} to be bold in C programs.  This can be done as follows
%D
%D \startbuffer
%D \definevimtyping [C] [syntax=c,numbering=on]
%D
%D \startvimcolorscheme[default]
%D
%D \definevimsyntax
%D   [Type]
%D   [style=monobold]
%D
%D \stopvimcolorscheme
%D
%D \startC
%D #include <stdio.h>
%D #include <stdlib.h>
%D
```

```
%D void main()
%D {
%D    printf("Hello World\n") ;
%D    return;
%D }
%D \stopC
%D \stopbuffer
%D \typebuffer which gives {\getbuffer}
%D
%D The second command provided by this module is \type{\definetypevimfile} for type-
setting files.
%D The syntax of this command is
%D
% %D \showsetups{definetypevimfile}
%D
%D For example, to pretty print a ruby file you can do
%D \starttyping
%D \definetypevimfile[typeRUBY] [syntax=ruby]
%D \stoptyping
%D after which one can use
%D \starttyping
%D \typeRUBY[option]{rubyfile}
%D \stoptyping
%D
%D We hope that this is sufficient to get you started.  The rest of this
%D document gives the implementation details of the module.  If you want to
%D change something, read ahead.

%D \section   {Module Details}
\writestatus  {loading}   {Context Module for ViM Sytax Highlighting}

\startmodule[vim]

\unprotect

\definesystemvariable {vs}  % Vim Syntax

%D First of all we take care of bold monotype.  By default, \CONTEXT\ uses
%D latin modern fonts.  If you want to get bold monotype in latin modern, you
%D need to use \type{modern-base} typescript.  For example:
%D \starttyping
%D \usetypescript[modern-base][texnansi] \setupbodyfont[modern]
%D \starttext
%D {\tt\bf This is bold monotype}
%D \stoptext
%D \stoptyping
%D \CONTEXT\ does not provide any style alternative for bold monotype, so we
%D provide one here.  This will only work if your font setup knows about bold
%D monotype.
```

```
\definealternativestyle [\v!bold\v!mono,\v!mono\v!bold] [\tt\bf] []

%D \macros{startvimcolorscheme}
%D To start a new vim colorscheme.

\def\startvimcolorscheme[#1]%
  {\pushmacro\vimcolorscheme
   \edef\vimcolorscheme{#1}}

\def\stopvimcolorscheme
  {\popmacro\vimcolorscheme}

%D \macros{definevimsyntax, definevimsyntaxsynonyms}
%D These macros should always occur inside a \type{\startvimcolorschme}
%D \unknown \type{\stopvimcolorscheme} pair.    The \type{\definevimsyntax}
%D macro defines syntax highlighting rules for \VIM's syntax highlighting
%D regions.  It takes three arguments \type{style}, \type{color} and
%D \type{command}.  The most common \VIM\ syntax highlighting regions are defined
%D in the end of this file.  The \type{\definevimsyntaxsynonyms} macro just
%D copies the settings from another syntax highlighting region.

\def\definevimsyntax
  {\dodoubleargumentwithset\dodefinevimsyntax}

\def\dodefinevimsyntax[#1]% [#2]
  {\getparameters[\??vs\vimcolorscheme#1]} %[#2]

\def\definevimsyntaxsynonyms
  {\dodoubleargumentwithset\dodefinevimsyntaxsynonyms}

\def\dodefinevimsyntaxsynonyms[#1][#2]%
  {\copyparameters[\??vs\vimcolorscheme#1][\??vs\vimcolorscheme#2]
                  [\c!style,\c!color,\c!command]}


%D \macros{vimsyntax}
%D This is just a placeholder macro.  The \filename{2context.vim} script marks
%D the highlightin reigions by \type{\s[...]{...}}.  While typing the generated
%D files, we locally redefine \type{\s} to \type{\vimsyntax}.

\def\vimsyntax[#1]#2%
  {\dostartattributes{\??vs\vimcolorscheme Normal}\c!style\c!color\empty%
   \dostartattributes{\??vs\vimcolorscheme #1}\c!style\c!color\empty%
   \getvalue{\??vs#1\c!command}{#2}%
    \dostopattributes%
    \dostopattributes}

%D \macros{setupvimtyping, typevimfile}
```

```
%D There are three settings for \type{\setupvimtyping}:  \type{syntax}, which
%D tells \VIM\ which syntax rules to use; \type{tab}, which sets the
%D \type{tabstop} in \VIM; and \type{space} which takes care of spaces.
%D
%D \type{\typevimfile} macro basically calls \VIM\ with appropriate settings and
%D sources the \filename{2context.vim} script.  The result is slow, because
%D parsing by \VIM\ is slow.  Do not use this method for anything larger than a
%D few hundred lines.  For large files, one option is to pre||prase them, and
%D then typeset the result.  We have not provided any interface for that, but
%D it is relatively easy to implement.
%D
%D Taking care of line||numbering is more tricky.  We could not get
%D \type{\setuplinenumbering} to work properly, so implement our own
%D line||numbering mechanism.  This is a bit awkward, since it places
%D line||number after each \type{^M} in the source file.  So, if the source
%D code line is larger than one typeset line, the line number will be on the
%D second line.  To do it correctly, we need to read lines from the vimsyntax
%D file one|-|by|-|one.  Our own mechanism for line||numbering is plain.
%D Unlike \CONTEXT's core verbatim highlighting, multiple blank lines are
%D displayed and numbered.

\def\setupvimtyping
  {\dosingleargument\getparameters[\??vs]}

\def\typevimfile
  {\dosingleempty\dotypevimfile}

\def\notypevimfile[#1][#2]#3%
  {\dotypevimfile[#1,#2]{#3}}

\def\dotypevimfile[#1]#2%
  {\doiffileelse{#2}
   {\dodotypevimfile[#1]{#2}}
   {\reporttypingerror{#2}}}

\def\dodotypevimfile[#1]#2%
  {\@@vsbefore
   \bgroup
   \initializevimtyping{#1}
   \runvimsyntax{#2}
    % The strut is needed for the output to be the same when not using
    % numbering.  Otherwise, multiple par's are ignored.  We need to figure out
    % a mechanism to imitate this behaviour even while using line numbering.
    \startlinenumbering[method=line]
    \input #2-vimsyntax.tmp\relax%
    \stoplinenumbering
   \egroup
   \@@vsafter}
```

```
\makecounter{vimlinenumber}

\def\doplacevimlinenumber
  {%Always place the first linenumber
   \showvimlinenumber
   %Calculate step in futute
   \let\placevimlinenumber\dodoplacevimlinenumber
   \pluscounter{vimlinenumber}}

\def\dodoplacevimlinenumber
  {\ifnum\numexpr(\countervalue{vimlinenumber}/\@@vsstep)*\@@vsstep\relax=\numexpr\countervalue{viml
      \showvimlinenumber
  \fi
   \pluscounter{vimlinenumber}}

\def\showvimlinenumber
  {\inmargin%TODO: make configurable
      {\dostartattributes\??vs\c!numberstyle\c!numbercolor\empty
       \countervalue{vimlinenumber}
       \dostopattributes}}

\def\initializevimtyping#1
  {\setupvimtyping[#1]
   %Make sure that stop is not empty
   \doifempty{\@@vsstop}{\setvalue{\@@vsstop}{0}}
   \doifelse{\@@vsstart}{\v!continue}
    {\setvalue{@@vsstart}{\countervalue{vimlinenumber}}}
    {\setcounter{vimlinenumber}{\doifnumberelse{\@@vsstart}{\@@vsstart}{1}}}
   \whitespace
  %\page[\v!preference]} gaat mis na koppen, nieuw:  later \nobreak
   \setupwhitespace[\v!none]%
   \obeylines
   \ignoreeofs
   \ignorespaces
   \activatespacehandler\@@vsspace
   \let\s=\vimsyntax
   \def\tab##1{\dorecurse{##1}{\space}}% TODO: allow customization
   \def\vimcolorscheme{\@@vscolorscheme}
   \processaction[\@@vsnumbering]
   [     \v!on=>\let\placevimlinenumber\doplacevimlinenumber,
        \v!off=>\let\placevimlinenumber\relax,
    \s!unknown=>\let\placevimlinenumber\relax,
    \s!default=>\let\placevimlinenumber\relax,
   ]
   \def\obeyedline{\placevimlinenumber\par\strut}
   }

\def\runvimsyntax#1
   {\executesystemcommand
```

```
      {texmfstart bin:vim
          "-u NONE  % No need to read unnessary configurations
           -e       % run in ex mode
%            -V10log  % For debugging only, will go away later.
           -c \letterbackslash"set noswapfile\letterbackslash"
           -c \letterbackslash"set tabstop=\@@vstab\letterbackslash"
           -c \letterbackslash"set cp\letterbackslash"
           -c \letterbackslash"syntax on\letterbackslash"
           -c \letterbackslash"set syntax=\@@vssyntax\letterbackslash"
           -c \letterbackslash"let contextstartline=\@@vsstart\letterbackslash"
           -c \letterbackslash"let contextstopline=\@@vsstop\letterbackslash"
           -c \letterbackslash"source kpse:2context.vim\letterbackslash"
           -c \letterbackslash"wqa\letterbackslash"
            " #1}}


%D \macros{definetypevimfile}
%D This macro allows you to define new file typing commands.  For example
%D \starttyping
%D \definetypevimfile[typeRUBY] [syntax=ruby]
%D \stoptyping
%D after which one can use
%D \starttyping
%D \typeRUBY[option]{rubyfile}
%D \stoptyping

\def\definetypevimfile
  {\dodoubleargument\dodefinetypevimfile}

\def\dodefinetypevimfile[#1][#2]%
  {\unexpanded\setvalue{#1}{\dodoubleempty\notypevimfile[#2]}}

%D \macros{definevimtyping}
%D
%D This macro allows you to pretty print code snippets.  For example
%D \startbuffer
%D \definevimtyping [RUBY] [syntax=ruby, numbering=on]
%D \startRUBY
%D # This is my first ruby program
%D puts "Hello World"
%D \stopRUBY
%D \stopbuffer
%D \typebuffer gives \getbuffer

\def\definevimtyping
  {\dodoubleargument\dodefinevimtyping}

\def\dodefinevimtyping[#1][#2]%
  {\setevalue{\e!start#1}{\noexpand\dostartbuffer[vimsyntax][\e!start#1][\e!stop#1]}%
```

```
   \setvalue{\e!stop#1}{\dodotypevimfile[#2]{\TEXbufferfile{vimsyntax}}}

%D Some defaults.

\setupvimtyping
  [        syntax=context,
           \c!tab=8,
         \c!space=\v!yes,
         \c!start=1,
          \c!stop=0,
        \c!before=,
         \c!after=,
     \c!numbering=\v!off,
   \c!numberstyle=\v!smallslanted,
   \c!numbercolor=,
          \c!step=1,
      colorscheme=default,
  ]

%D Pre-defined Syntax :  {{{
%D This is based on \filename{ps_color.vim}, which does not use any bold
%D typeface.

%D \VIM\ uses hex mode for setting colors, I do not want to convert them to rgb
%D values.

\startvimcolorscheme[default]

\setupcolor[hex]

\definecolor  [vimsyntax!default!Special]     [h=907000]
\definecolor  [vimsyntax!default!Comment]     [h=606000]
\definecolor  [vimsyntax!default!Number]      [h=907000]
\definecolor  [vimsyntax!default!Constant]    [h=007068]
\definecolor  [vimsyntax!default!PreProc]     [h=009030]
\definecolor  [vimsyntax!default!Statement]   [h=2060a8]
\definecolor  [vimsyntax!default!Type]        [h=0850a0]
\definecolor  [vimsyntax!default!Todo]        [h=e0e090]

\definecolor  [vimsyntax!default!Error]       [h=c03000]
\definecolor  [vimsyntax!default!Identifier]  [h=a030a0]
\definecolor  [vimsyntax!default!SpecialKey]  [h=1050a0]
\definecolor  [vimsyntax!default!Underline]   [h=6a5acd]


\definevimsyntax
  [Normal]
  [\c!style=\v!mono,\c!color=\maintextcolor]

```

```
\definevimsyntax
  [Constant]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Constant]

\definevimsyntaxsynonyms
  [Character,Boolean,Float]
  [Constant]

\definevimsyntax
  [Number]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Number]

\definevimsyntax
  [Identifier]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Identifier]

\definevimsyntaxsynonyms
  [Function]
  [Identifier]

\definevimsyntax
  [Statement]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Statement]

\definevimsyntaxsynonyms
  [Conditional,Repeat,Label,Operator,Keyword,Exception]
  [Statement]

\definevimsyntax
  [PreProc]
  [\c!style=\v!mono,\c!color=vimsyntax!default!PreProc]

\definevimsyntaxsynonyms
  [Include,Define,Macro,PreCondit]
  [PreProc]

\definevimsyntax
  [Type,StorageClass, Structure, Typedef]
  [\c!style=\v!mono, \c!color=vimsyntax!default!Type]

\definevimsyntax
  [Special]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Special]

\definevimsyntax
  [SpecialKey]
  [\c!style=\v!mono,\c!color=vimsyntax!default!SpecialKey]

\definevimsyntax
```

```
  [Tag,Delimiter]
  [\c!style=\v!mono]

\definevimsyntax
  [Comment,SpecialComment]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Comment]

\definevimsyntax
  [Debug]
  [\c!style=\v!mono]

\definevimsyntax
  [Underlined]
  [\c!style=\v!mono,\c!command=\underbar]

\definevimsyntax
  [Ignore]
  [\c!style=\v!mono]

\definevimsyntax
  [Error]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Error]

\definevimsyntax
  [Todo]
  [\c!style=\v!mono,\c!color=vimsyntax!default!Todo]

\stopvimcolorscheme
% }}}

\protect

\stopmodule


%D An example usage:  {{{

\doifnotmode{demo}{\endinput}

\setupcolors[state=start]

\setupbodyfont[10pt]

\definevimtyping  [MATLAB]  [syntax=matlab]

\starttext
\startMATLAB
function russell_demo()
```

```
% Do the example in ch 17 (p501) of Russell and Norvig
% (1,1) is top left corner.

r = 3; c = 4; p = 0.8; action_cost = -1/25;
obstacle = zeros(r,c); obstacle(2,2)=1;
terminal = zeros(r,c); terminal(1,4)=1; terminal(2,4)=1;
absorb = 1;
wrap_around = 0;
noop = 0;
T = mk_grid_world(r, c, p, obstacle, terminal, absorb, wrap_around, noop);
% Add rewards for terminal states
nstates = r*c + 1;
if noop
  nact = 5;
else
  nact = 4;
end
R = action_cost*ones(nstates, nact);
R(10,:)  = 1;
R(11,:)  = -1;
R(nstates,:)  = 0;
discount_factor = 1;


V = value_iteration(T, R, discount_factor);
%reshape(V(1:end-1),[r c])
%    0.8116    0.8678    0.9178    1.0000
%    0.7616    0.7964    0.6603   -1.0000
%    0.7053    0.6553    0.6114    0.3878
% Same as the book p501

Q = Q_from_V(V, T, R, discount_factor);
[V, p] = max(Q, [], 2);


use_val_iter = 1;
% (I-gT) is singular since g=1 and there is an absorbing state (i.e., T(i,i)=1)
% Hence we cannot use value determination.
[p,V] = policy_iteration(T, R, discount_factor, use_val_iter);

%reshape(V(1:end-1),[r c])
%    0.8115    0.8678    0.9178    1.0000
%    0.7615    0.7964    0.6603   -1.0000
%    0.7048    0.6539    0.6085    0.3824
\stopMATLAB

\page

\typevimfile[numbering=on,numbercolor=red,step=5]{\jobname.tex}
```

```
\stoptext

% }}}
```