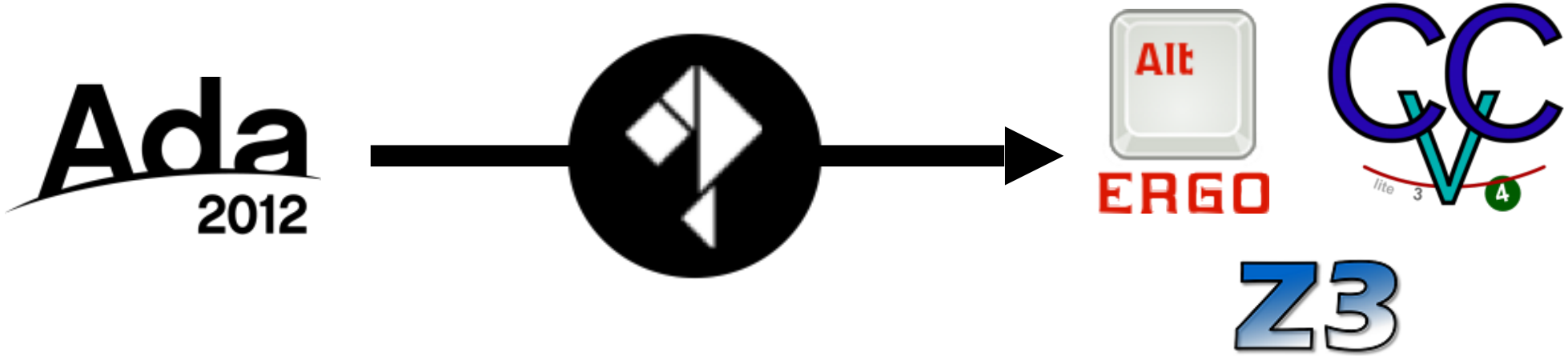
Prove with SPARK: No Math, Just Code

Yannick Moy – SPARK Product Manager – AdaCore

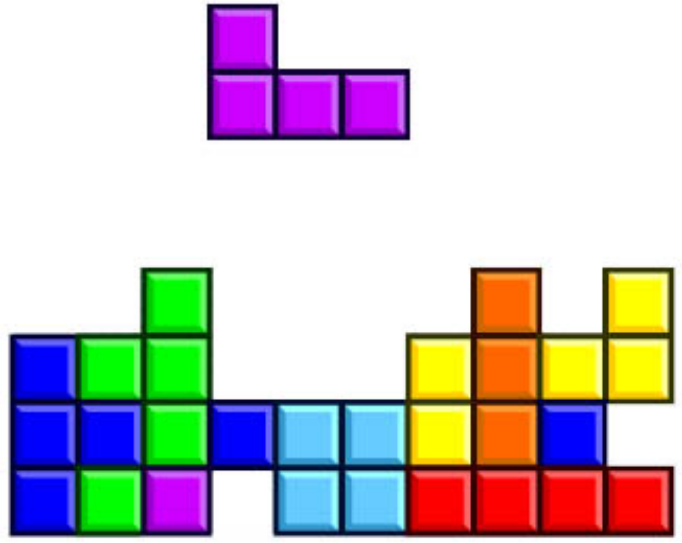
SPARK2014



Program Integrity

Correct data initialization

No language error or exception (division by zero, buffer overflow)



Functionality

Correct data flow

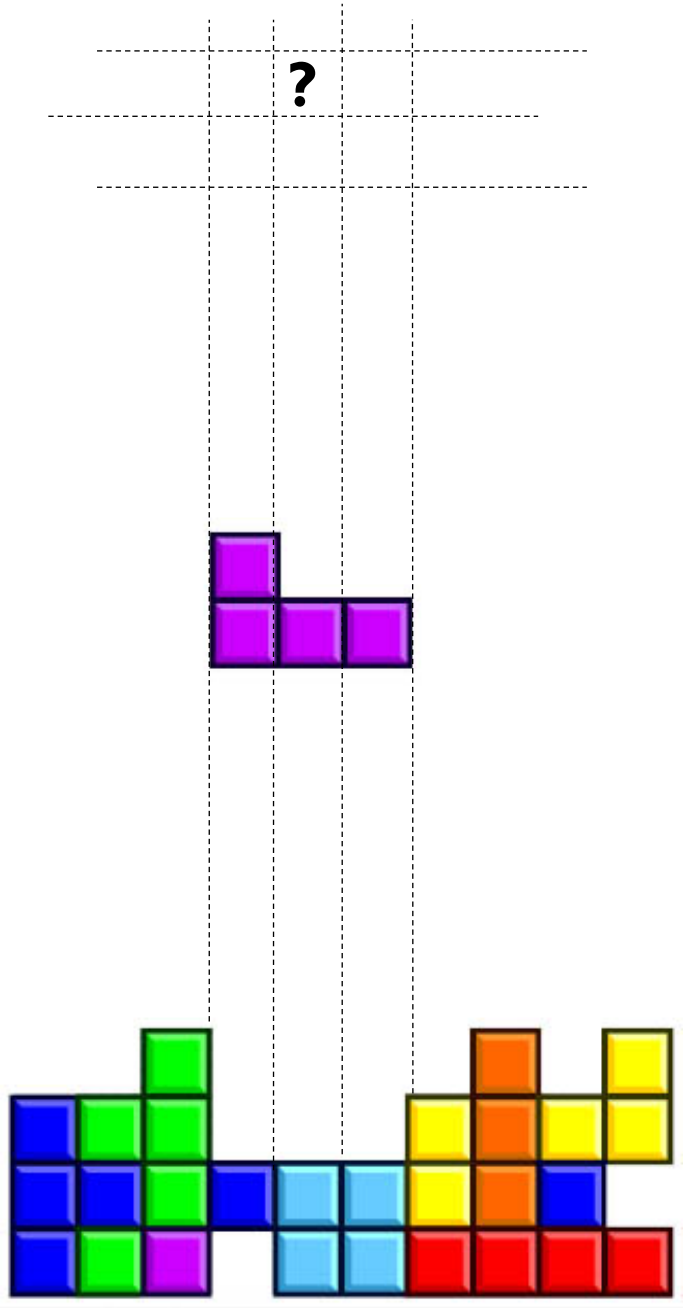
Complete lines are removed

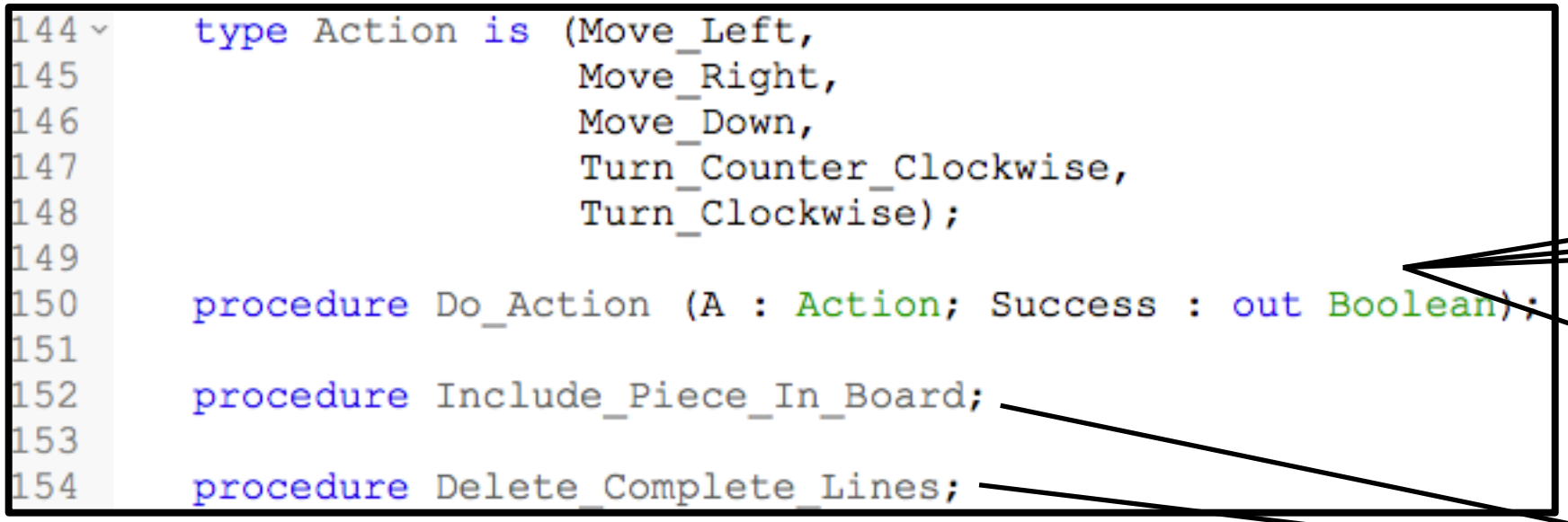
Falling piece does not overlap with fallen pieces or board background

```
31    type Cell is (Empty, I, O, J, L, S, T, Z);
32
33    subtype Shape is Cell range I .. Z;
34
35    subtype Three_Shape is Cell range J .. Z;
```

```
48    subtype PX_Coord is Integer range -1 .. X_Size - 1;
49    subtype PY_Coord is Integer range -1 .. Y_Size - 1;
50
51    type Direction is (North, East, South, West);
52
53    type Piece is record
54       S : Shape;
55       D : Direction;
56       X : PX_Coord;
57       Y : PY_Coord;
58    end record;
59
60    Cur_Piece : Piece;
```

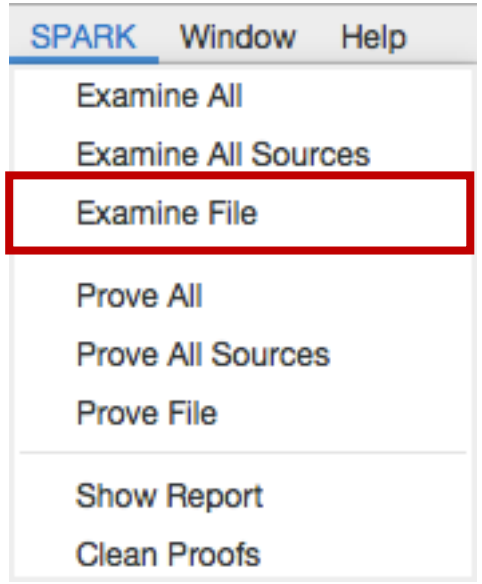
```
40    subtype X_Coord is Integer range 1 .. X_Size;
41    subtype Y_Coord is Integer range 1 .. Y_Size;
42
43    type Line is array (X_Coord) of Cell;
44    type Board is array (Y_Coord) of Line;
45
46    Cur_Board : Board;
```





```
144    type Action is (Move_Left,
145                    Move_Right,
146                    Move_Down,
147                    Turn_Counter_Clockwise,
148                    Turn_Clockwise);
149
150    procedure Do_Action (A : Action; Success : out Boolean);
151
152    procedure Include_Piece_In_Board;
153
154    procedure Delete_Complete_Lines;
```

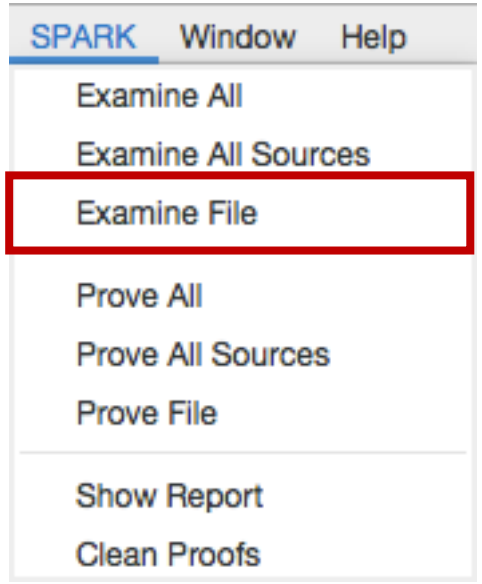
```
171  begin
172     --  Init the devices
173     Leds.Init;
174     Oled_Init;
175
176     --  Game loop
177     Game : loop
178        --  Clear board
179        Cur_Board := (others => (others => Empty));
180        Nbr_Pieces := 0;
181        Oled_Clear;
182        Led_Step := 0;
183
184        Piece : loop
185           --  Add a new piece
186           Rnd := Rnd * 1103515245 + 12345;
187
188           Cur_Piece := (S => Cell'Val (1 + ((Rnd / 65536) mod 7)),
189                         D => North,
190                         X => X_Size / 2,
191                         Y => Y_Coord'First);
192
193           Cur_State := Piece_Falling;
194
195           --  Stop the game when the piece cannot appear
196           exit Piece when not Valid_Configuration;
197
198           --  Draw the board, including the new piece
199           Draw_Board (True);
200
201           Fall_Period := Slow_Fall_Period;
202
203           --  Make the piece falling
204           Fall : loop
205              Set_Led (False);
206
207              --  Led chaser
208              Set_Led1 (Led_Step = 0);
209              Set_Led2 (Led_Step = 1);
210              Set_Led3 (Led_Step = 2);
211              Led_Step := Led_Step + 1;
212
213              --  At most 2 rotations or moves per fall
214              for J in 1 .. 2 loop
215                 --  Check button
216                 if Button1_Pressed then
217                    Do_Action (Move_Right, Success);
218                 elsif Button2_Pressed then
                      Do_Action (Turn_Clockwise, Success);
220                 elsif Button3_Pressed then
221                    Do_Action (Move_Left, Success);
222                 elsif Button_Pressed then
223                    Fall_Period := Quick_Fall_Period;
224                 end if;
225
226                 Now := Now + Fall_Period;
227                 delay until Now;
228
229                 Draw_Board (True);
230              end loop;
231
232              --  Fall and continue unless the piece hits the ground
233              Do_Action (Move_Down, Success);
234              exit Fall when not Success;
235
236              Draw_Board (True);
237           end loop Fall;
238
239           --  Done with that piece
240           Cur_State := Piece_Blocked;
241           Nbr_Pieces := Nbr_Pieces + 1;
242
243           Set_Led (True);
244           Set_Led1 (True);
245           Set_Led2 (True);
246           Set_Led3 (True);
247
248           Include_Piece_In_Board;
249           Delete_Complete_Lines;
250
251           Now := Now + Fall_Period;
252           delay until Now;
253        end loop Piece;
254
255        --  Lost!
256        declare
257           Toggle : Boolean := True;
258        begin
259           loop
260              --  Wait for any button
261              exit when Button1_Pressed or Button2_Pressed or Button3_Pressed
262                or Button_Pressed;
263
264              --  Change the seed of the random generator
265              Rnd := Rnd + 1;
266
267              --  Toggle leds
268              Set_Led (Toggle);
269              Set_Led1 (not Toggle);
270              Set_Led2 (Toggle);
271              Set_Led3 (not Toggle);
272
273              Now := Now + Blink_Period;
274              delay until Now;
275
276              Toggle := not Toggle;
277           end loop;
278        end;
279     end loop Game;
280  end Main;
```




```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: analysis of data and information flow ...
```

no message?
no reads of uninitialized data

Program Integrity




```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: analysis of data and information flow ...
```

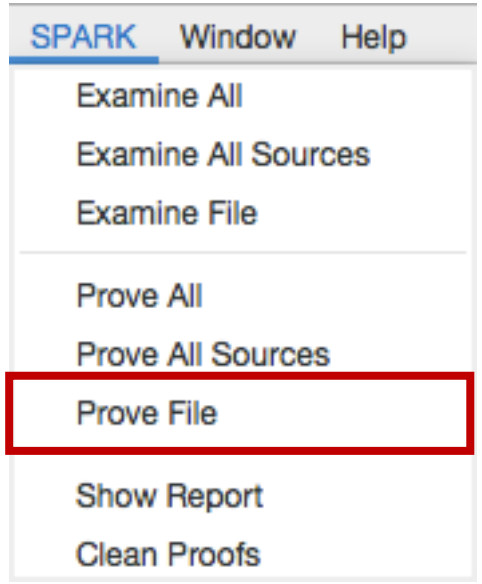
no message?
no reads of uninitialized data

Program Integrity

```
178    procedure Do_Action (A : Action; Success : out Boolean) with
179      Global => (Input => Cur_Board, In_Out => Cur_Piece);
180
181    procedure Include_Piece_In_Board with
182      Global => (Input => Cur_Piece, In_Out => Cur_Board);
183
184    procedure Delete_Complete_Lines with
185      Global => (In_Out => Cur_Board);
```

no message?
previous + correct data dependencies

Functionality

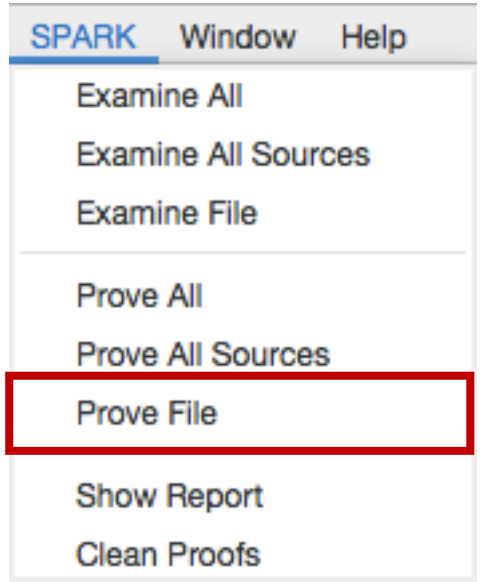



```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
```

Program Integrity

no message?
previous + no run-time error (division by zero, buffer overflow…)

here: 6 messages on buffer overflows + 4 on scalar ranges




```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
```

Program Integrity

no message?
previous + no run-time error (division by zero, buffer overflow…)

here: 6 messages on buffer overflows + 4 on scalar ranges

```
154    function Within_Bounds (Y : Integer; X : Integer) return Boolean is
155       (Y in Y_Coord and then X in X_Coord);
156
157    function Within_Bounds (P : Piece) return Boolean is
158       (case P.S is when O => Within_Bounds (P.Y, P.X) and then ...
159
160    procedure Include_Piece_In_Board with
161      Global => (Input => Cur_Piece, In_Out => Cur_Board),
162      Pre    => Within_Bounds (Cur_Piece);
```

no message!

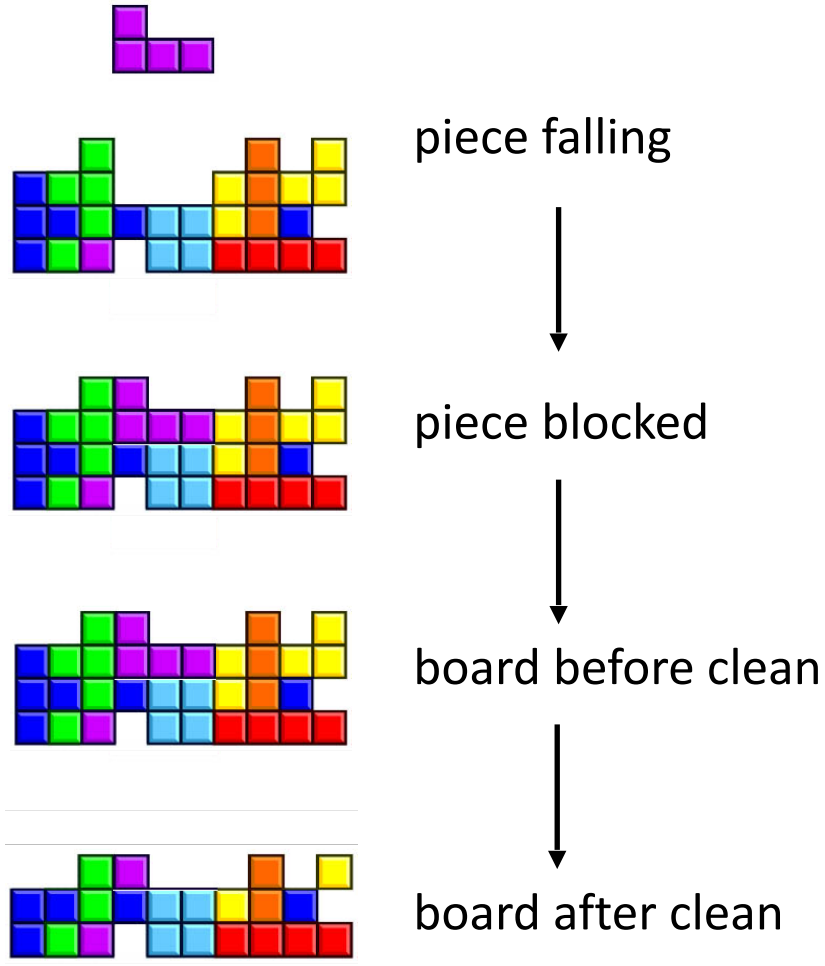
Functionality NO_COMPLETE_LINES - Complete lines are removed

```
147    function Is_Complete_Line (L : Line) return Boolean is
148      (for all X in X_Coord => L(X) /= Empty);
149
150    function No_Complete_Lines (B : Board) return Boolean is
151      (for all Y in Y_Coord => not Is_Complete_Line (B(Y)))
152    with Ghost;
```

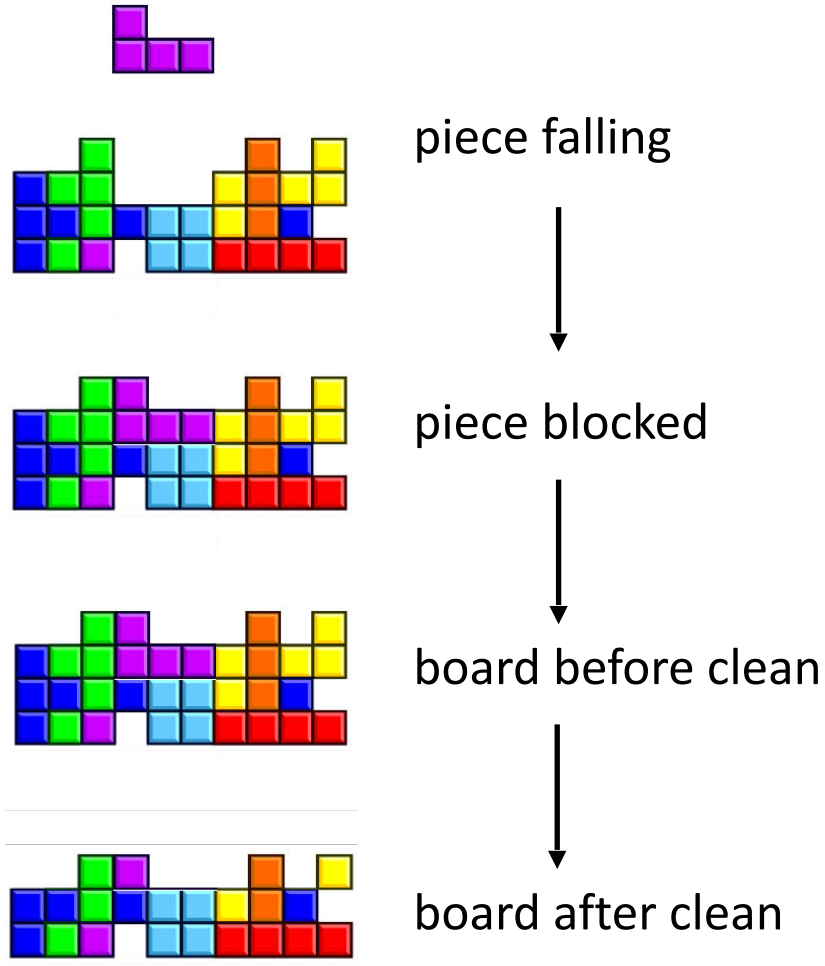
Functionality NO_OVERLAP - Falling piece does not overlap with fallen pieces or board background

```
155    function Is_Empty (B : Board; Y : Integer; X : Integer) return Boolean is
156       (X in X_Coord and then Y in Y_Coord and then B(Y)(X) = Empty);
157
158    function No_Overlap (B : Board; P : Piece) return Boolean is
159       (case P.S is
160          when O => Is_Empty (B, P.Y, P.X) and then Is_Empty (B, P.Y, P.X + 1) and then
161                    Is_Empty (B, P.Y + 1, P.X) and then Is_Empty (B, P.Y + 1, P.X + 1),
162          when I =>
163            (for all Y in I_Delta =>
164               (for all X in I_Delta =>
165                  (if Possible_I_Shapes (P.D) (Y, X) then Is_Empty (B, P.Y + Y, P.X + X)))),
166          when Three_Shape =>
167            (for all Y in Three_Delta =>
168               (for all X in Three_Delta =>
169                  (if Possible_Three_Shapes (P.S, P.D) (Y, X) then Is_Empty (B, P.Y + Y, P.X + X))))));
```

State automaton



State automaton

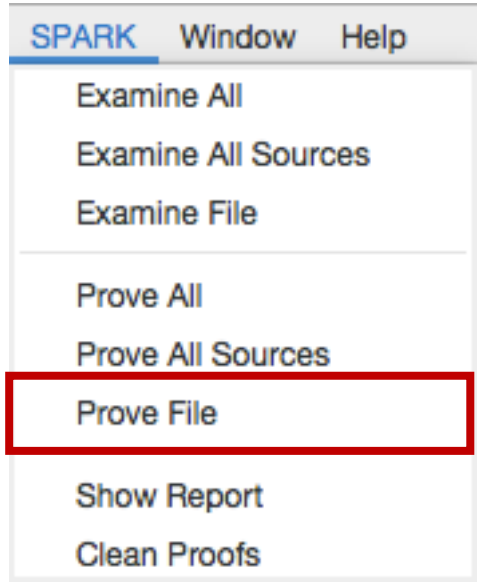


Ghost code

```
82    type State is (Piece_Falling,
83                   Piece_Blocked,
84                   Board_Before_Clean,
85                   Board_After_Clean) with Ghost;
86
87    Cur_State : State with Ghost;
```

```
173    function Valid_Configuration return Boolean is
174       (case Cur_State is
175          when Piece_Falling | Piece_Blocked =>
176            No_Overlap (Cur_Board, Cur_Piece),
177          when Board_Before_Clean => True,
178          when Board_After_Clean =>
179            No_Complete_Lines (Cur_Board))
180    with Ghost;
```

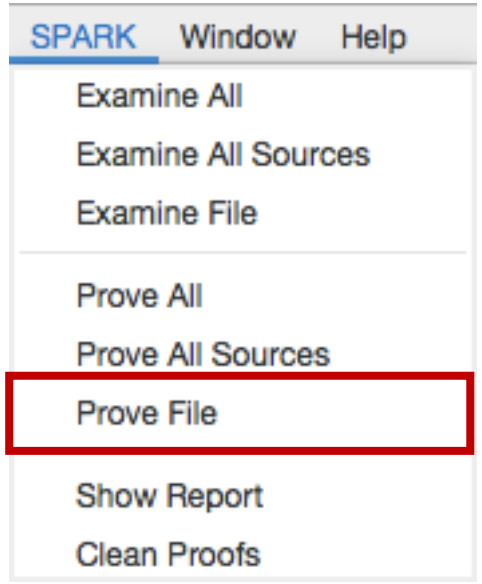
```
216    procedure Include_Piece_In_Board with
217      Pre    => Cur_State = Piece_Blocked and then
218                Valid_Configuration,
219      Post   => Cur_State = Board_Before_Clean and
220                Valid_Configuration;
```




```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
```

no message?
previous + code implements specification

Functionality




```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
```

no message?
previous + code implements specification

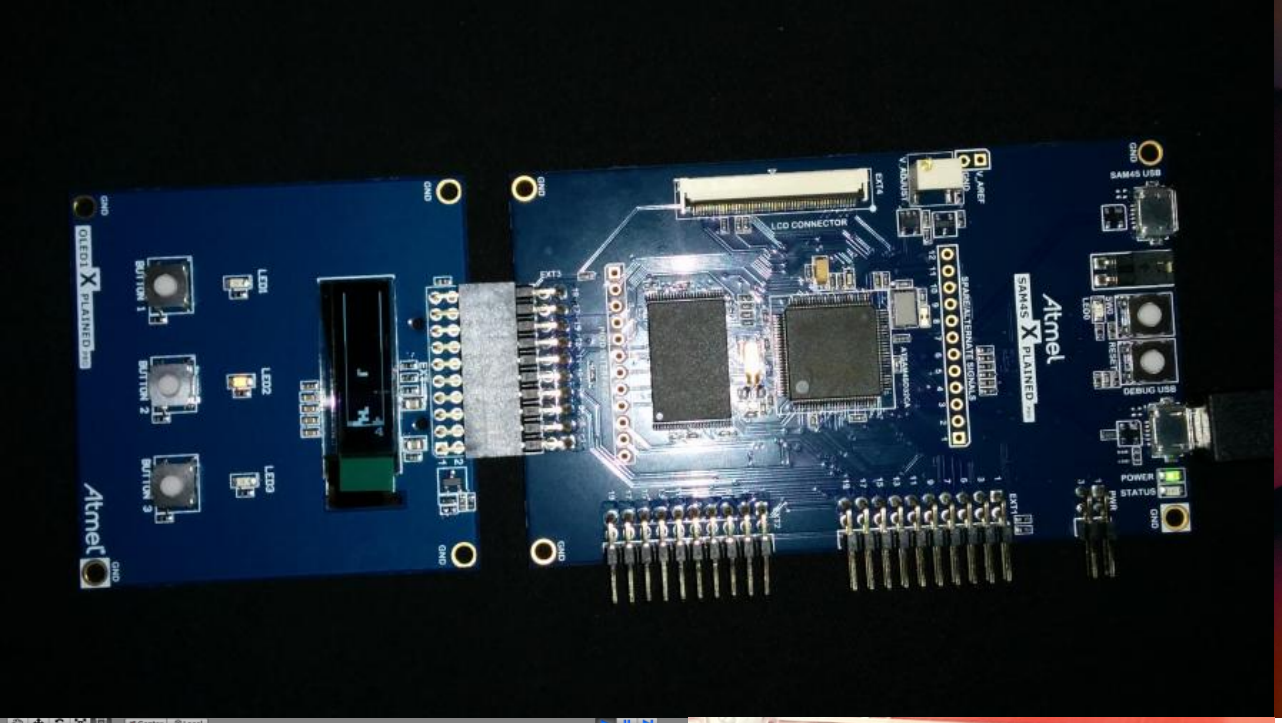
Functionality

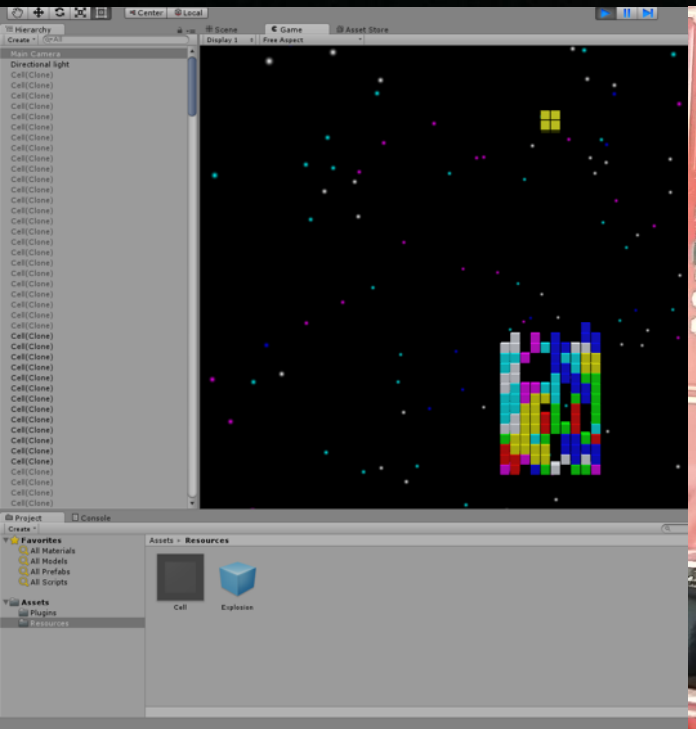
How hard is it?

Fully proved at level 0 in 11 seconds

…out of 5 levels!
level 0 =
- one prover only
- ≈ 1 sec timeout
- no splitting

…on one core!
4 seconds with multicore





blog.adacore.com

libre.adacore.com